Neiva, 17 de noviembre 2021

Señores

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD SURCOLOMBIANA

Ciudad Neiva

El (Los) suscrito(s):

Arturo Polanco Lozano, con C.C. No. 1075249293,

Maria Ximena Rodríguez Borda, con C.C. No. 1075296030,

Autor(es) de la tesis y/o trabajo de grado titulado Clasificación de Lesiones Dermatológicas Usando Redes Neuronales Convolucionales, Metodos Ensemble y Servicio Web presentado y aprobado en el año 2021 como requisito para optar al título de

INGENIERO DE SOFTWARE;

Autorizamos al CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN de la Universidad Surcolombiana para que, con fines académicos, muestre al país y el exterior la producción intelectual de la Universidad Surcolombiana, a través de la visibilidad de su contenido de la siguiente manera:

- Los usuarios puedan consultar el contenido de este trabajo de grado en los sitios web que administra la Universidad, en bases de datos, repositorio digital, catálogos y en otros sitios web, redes y sistemas de información nacionales e internacionales "open access" y en las redes de información con las cuales tenga convenio la Institución.

- Permita la consulta, la reproducción y préstamo a los usuarios interesados en el contenido de este trabajo, para todos los usos que tengan finalidad académica, ya sea en formato Cd-Rom o digital desde internet, intranet, etc., y en general para cualquier formato conocido o por conocer, dentro de los términos establecidos en la Ley 23 de 1982, Ley 44 de 1993, Decisión Andina 351 de 1993, Decreto 460 de 1995 y demás normas generales sobre la materia.

- Continúo conservando los correspondientes derechos sin modificación o restricción alguna; puesto que, de acuerdo con la legislación colombiana aplicable, el presente es un acuerdo jurídico que en ningún caso conlleva la enajenación del derecho de autor y sus conexos.

De conformidad con lo establecido en el artículo 30 de la Ley 23 de 1982 y el artículo 11 de la Decisión Andina 351 de 1993, "Los derechos morales sobre el trabajo son propiedad de los autores" , los cuales son irrenunciables, imprescriptibles, inembargables e inalienables.

EL AUTOR/ESTUDIANTE: Aldo Polanco

Firma: _____

EL AUTOR/ESTUDIANTE: Mara Xinera Rodriguez

Firma: Mara Xinera R.

TÍTULO COMPLETO DEL TRABAJO: Clasificación de Lesiones Dermatológicas Usando Redes Neuronales Convolucionales, Métodos Ensemble y Servicios Web

AUTOR O AUTORES:

Primero y Segundo Apellido	Primero y Segundo Nombre
Polanco Lozano	Arturo
Rodríguez Borda	Maria Ximena

DIRECTOR Y CODIRECTOR TESIS:

Primero y Segundo Apellido	Primero y Segundo Nombre
Castro Silva	Juan Antonio

ASESOR (ES):

Primero y Segundo Apellido	Primero y Segundo Nombre

PARA OPTAR AL TÍTULO DE: INGENIERO DE SOFTWARE

FACULTAD: INGENIERIA

PROGRAMA O POSGRADO: INGENIERIA DE SOFTWARE

CIUDAD: NEIVA **AÑO DE PRESENTACIÓN:** 2021 **NÚMERO DE PÁGINAS: 255**

TIPO DE ILUSTRACIONES (Marcar con una **X**):

Diagramas_X_ Fotografías___ Grabaciones en discos___ Ilustraciones en general___ Grabados___ Láminas___ Litografías___ Mapas___ Música impresa___ Planos___ Retratos___ Sin ilustraciones___ Tablas o Cuadros_X_

SOFTWARE requerido y/o especializado para la lectura del documento:

MATERIAL ANEXO:

PREMIO O DISTINCIÓN *(En caso de ser LAUREADAS o Meritoria)*:

PALABRAS CLAVES EN ESPAÑOL E INGLÉS:

	Español	Inglés
1.	Sistema de información	Information system
2.	Clasificar	Classifier
3.	Servicio web	Web Service
4.	Aplicación	App
5.	Lesiones de piel	Skin Lesions

RESUMEN DEL CONTENIDO: (Máximo 250 palabras)

El proyecto consiste en desarrollar un sistema de información web y móvil que permita clasificar diferentes lesiones de la piel, entre las que se encuentran: Melanoma, Carcinoma de Células Basales, Queratosis Benigna y Melanocito Nevi. Estas lesiones representan distintos riesgos para los pacientes, por lo tanto, es importante ayudar a diagnosticar las que pueden llegar a comprometer la vida de los pacientes o tranquilizarlos si no representan peligro. Gracias a la competencia SIIM-ISIC Melanoma Classification Challenge, se cuenta con un conjunto de datos que contiene imágenes distribuidas públicamente, además de los modelos de predicción ganadores de cada uno de los años. Se implementa un sistema de diagnóstico asistido por computador (CADx) usando un servicio web desarrollado con la librería FastAPI, que está alojado en la nube de Google Cloud Platform. Como producto final tenemos una aplicación web y móvil desarrollada en el framework React Native que permite la comunicación e interacción del cliente con el servicio web.

ABSTRACT: (Máximo 250 palabras)

The project consists of developing a web and mobile information system that allows classifying different skin lesions, among which are: Melanoma, Basal Cell Carcinoma, Benign Keratosis and Nevi Melanocyte. These injuries represent different risks for patients, therefore, it is important to help diagnose those that can compromise the lives of patients or to reassure them if they do not represent a danger. Thanks to the SIIM-ISIC Melanoma Classification Challenge, there is a data set that contains publicly distributed images, in addition to the winning prediction models for each of the years. A computer-assisted diagnostic system (CADx) is implemented using a web service developed with the FastAPI library, which is hosted on the Google Cloud Platform. As a final product we have a web and mobile application developed in the React Native framework that allows communication and interaction of the client with the web service.

APROBACION DE LA TESIS

Nombre Presidente Jurado: Fernando Rojas Rojas.

Firma: Rojas.

Nombre Jurado: Luis Gregorio Ramón C.

Firma: Luis Gregorio Ramón C.

Nombre Jurado: Fe. Ley Medina Rojas

Firma:

Clasificación de Lesiones Dermatológicas Usando Redes Neuronales Convolucionales, Métodos Ensemble y Servicios Web

Arturo Polanco Lozano

Código 20142130002

María Ximena Rodríguez Borda

Código 20152142076

Juan Antonio Castro Silva

Director

Universidad Surcolombiana

Facultad de Ingeniería

Programa de Ingeniería de Software

2021

Tabla de Contenido

Lista de Figuras

1 Introducción

El proyecto consiste en desarrollar un sistema de información web y móvil que permita clasificar diferentes lesiones de la piel, entre las que se encuentran: Melanoma, Carcinoma de Células Basales, Queratosis Benigna y Melanocito Nevi. Estas lesiones representan distintos riesgos para los pacientes, por lo tanto, es importante ayudar a diagnosticar las que pueden llegar a comprometer la vida de los pacientes o tranquilizarlos si no representan peligro. Gracias a la competencia SIIM-ISIC Melanoma Classification Challenge, se cuenta con un conjunto de datos que contiene imágenes distribuidas públicamente, además de los modelos de predicción ganadores de cada uno de los años. Se implementa un sistema de diagnóstico asistido por computador (CADx) usando un servicio web desarrollado con la librería FastAPI, que está alojado en la nube de Google Cloud Platform. Como producto final tenemos una aplicación web y móvil desarrollada en el framework React Native que permite la comunicación e interacción del cliente con el servicio web.

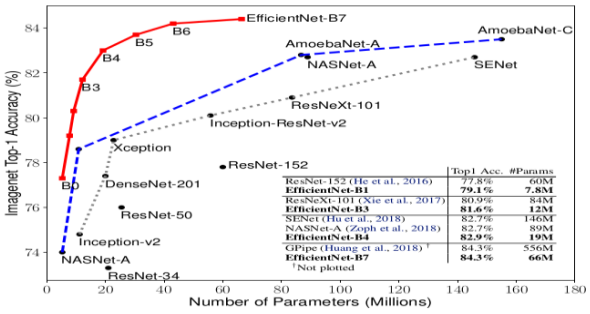
2 Antecedentes y Justificación

2.1 Antecedentes

Desde el año 2012, las "redes neuronales convolucionales" o CNN se convirtieron en el estado del arte para la clasificación de imágenes (Krizhevsky, A., Sutskever, I. y Hinton, GE, 2012). Las CNN se caracterizan por realizar un proceso de extracción de características visuales jerárquicas en sus capas convolucionales, dónde las primeras detectan patrones básicos y primitivos, mientras que en las capas posteriores se usan los patrones detectados en las capas previas para crear nuevas características más complejas. Desde ese entonces, se ha venido realizando investigación para crear variaciones qué mejoren su rendimiento.

Las CNN pueden variar dependiendo de su diseño, se pueden controlar parámetros cómo las dimensiones de los filtros en la capa convolucional, cantidad de capas convolucionales, variaciones entre las conexiones de sus capas, etc. Dependiendo de su configuración tendremos una arquitectura, estas cómo estándar en la academia e industria son evaluadas y rankeadas en una misma labor, la exactitud al clasificar las imágenes del conjunto de datos ImageNet. Esta nos servirá cómo un punto de referencia para seleccionar nuestro modelo para clasificar lesiones de piel:

Figura 1 Comparación tamaño de modelos CNN vs precisión



Comparación de tamaño de modelos CNN vs precisión, Adaptado de Mingxing Tan, & Quoc V. Le. (2020). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.

En el gráfico anterior se puede apreciar cómo sobresale EfficientNet-B7 en dos aspectos:

- Exactitud al clasificar imágenes.
- Menor número de parámetros.

Si bien es importante qué un modelo alcance la mayor exactitud posible, también lo es que este no posea una gran cantidad de parámetros. El hecho de que una arquitectura posea menos parámetros significa una reducción en la memoria para almacenarse y un menor uso de recursos computacionales para realizar inferencias (toma de decisiones).

Un procedimiento usado para mejorar el desempeño final de un sistema de aprendizaje automático es conocido cómo métodos en conjunto o “ensemble”, donde son entrenados diferentes clasificadores cuyas predicciones se tendrán en cuenta en una decisión final.

La implementación ganadora de la competencia ISIC del año 2020 (Ha, Liu and Liu, 2020) usa métodos en conjunto con una gran variedad de modelos de CNN, un total de 18 modelos en los que se utilizan las arquitecturas EfficientNet (5 variaciones B3, B4, B5, B6, B7), SE-ResNeXt-101 y ResNeSt-101.

De forma similar, otras implementaciones (Nadipineni, H. 2020) recientes para la detección de melanoma reafirman el uso de EfficientNet y ResNets combinados con métodos en conjunto, se discuten el impacto del preprocesamiento de los datos antes de entrenar el modelo, realizando técnicas de “aumentado de datos” al alterar las imágenes del modelo y ingresarlas nuevamente, el efecto de convertir la imagen a escala de grises, reajustar el tamaño de la imagen, segmentación para sólo introducir al modelo la imagen con la región afectada extrayendo la piel y validación

cruzada dónde se divide el conjunto de entrenamiento en diferentes grupos para crear distintos modelos.

Una práctica estándar para realizar una tarea de clasificación qué utilice CNN es el uso de la "transferencia de aprendizaje", con está, los pesos o valores iniciales de los filtros en la capa convolucional no se asignan de forma aleatoria sino qué se toman de otro modelo previamente entrenado en una tarea de visión por computadora (Manzo & Penillo, 2020), debido a qué no existe un modelo base estándar para las lesiones de piel, se utilizan modelos que hayan sido entrenadas con el conjunto de datos ImageNet, si bien las lesiones de piel no se asemejan mucho a este conjunto de datos, ImageNet posee una rica representación de características visuales en una gran variedad de imágenes, esto ayuda para qué el modelo no aprenda desde cero y reduzca los tiempos de entrenamiento. La entrada es estándar con 3 canales de color y dimensiones de la imagen de 224 píxeles de ancho por 224 píxeles de largo, las capas de "neuronas totalmente conectadas" se ubican posteriormente a las convolucionales se remueven, ya que se reemplazan por una o más capas que toman la decisión.

En otra investigación se prueba el efecto positivo de usar una función de pérdida mejor adaptada al desbalanceo de clases (Le, Hieu, Lua, & Ngo, 2020), una alternativa que funciona mejor cuando unas clases están mayormente representadas con más ejemplos se conoce cómo la pérdida focal. Además, se complementa añadiendo pesos a cada una de las clases de la función de coste, donde el valor se eleva de forma proporcional a la cantidad de ejemplos por clase (de manera que el impacto de clasificar una clase más representada sea menor, mientras que cometer un error en la clase menos representada será mayor) y de esta forma compensar el desbalance.

En el mercado se analizaron algunas aplicaciones que permiten detectar diferentes lesiones de piel, en base a sus calificaciones con los usuarios analizaremos las siguientes:

2.1.1 SkinVision

Aplicación que detecta exclusivamente el cáncer de piel, tiene una precisión por encima del 70% en el diagnóstico de melanomas (Arteaga, 2015), realizando también un seguimiento de la evolución de manchas y lunares en la piel para marcar cambios significativos a través del tiempo (Servicio de pago). Por último, el resultado del diagnóstico está acompañado de un indicador de peligrosidad según la lesión de piel, donde si es alto se recomienda acudir al Dermatólogo.

Figura 2 Logotipo SkinVision

Logotipo de App SkinVision. Adaptada de SkinVision, SkinVision,2021, Fuente. https://www.skinvision.com/

2.1.2 Molexplore

Detección de cáncer de piel a través de un auto seguimiento de las manchas cutáneas, logrando detectar a tiempo diferentes problemas de la piel. Como aplicación permite crear perfiles personalizados controlando de manera independiente lunares y manchas de cualquier miembro de la familia. Otro servicio, son consejos sobre la protección de la piel y prevención de las principales variedades de cáncer: Carcinoma de Merkel, Sarcoma de Kaposi, el Linfoma Cutáneo, Melanoma y Melanoma agresivo, Dermatofibrosarcoma, Carcinoma Basocelular; junto a este servicio se encuentra las notificaciones sobre el nivel de rayos ultravioletas del lugar donde se encuentre el usuario, asimismo como la información relativa de la humedad y la sensación térmica (Tomás, 2021).

Figura 3 Logotipo App Molexpore

Logotipo de App Molexplore. Adaptada de Molexplore Skin Cáncer App, Molexplore,2014, Fuente. https://molexplore.com/es/

2.1.3 MoleScope

Aplicación móvil, que cuenta con servicios como seguimiento de lunares, Mapa corporal 3D, análisis de imagen, consulta con médicos expertos, recordatorio de calendario y almacenamiento en la nube (*MoleScope*^{TM}, 2015).

Figura 4 Logotipo App Molescope

Logotipo de App MoleScope. Adaptada de MoleScope, MoleScope ,2021, Fuente. https://www.molescope.com/

Al analizar las alternativas de software que se ofrecen en el mercado, se encontraron los siguientes puntos de mejora:

2.1.3.1 Mejora en el Desempeño.

Los algoritmos de aprendizaje profundo se actualizan constantemente, el implementar arquitecturas modernas nos permite obtener un mejor rendimiento.

2.1.3.2 Cantidad de lesiones que se detectan.

Ofrecer la detección de múltiples lesiones que llegan al clasificador permitiría escalar y brindar un mejor servicio al usuario.

2.1.3.3 Precio.

Se propone un sistema de libre uso sin ninguna carga económica.

2.1.3.4 Mantenimiento.

Nuestro algoritmo se reentrena usando los datos que brindan los usuarios, este aspecto es clave para mejorar iterativamente el modelo.

2.1.3.5 Servicios online y offline

Se puede hacer uso del clasificador más sofisticado o de una versión optimizada para desplegarse en dispositivos embebidos a costa de una pequeña degradación en su desempeño. Estas opciones se brindan con el propósito de impactar en regiones donde el acceso a internet es limitado.

2.1.3.6 Tiempo de Inferencia

El servicio tradicional puede necesitar hasta 30 segundos para realizar una detección, esto se puede atacar al hacer un buen uso de la computación en la nube para reducir el retardo.

2.1.3.7 Aplicabilidad

Los usuarios no tienen acceso a detalles clave de la imagen qué motiva al algoritmo a tomar una decisión.

2.2 Justificación

El cáncer de piel es el más común de todos los tipos de cáncer, a nivel mundial. El melanoma representa el 1% de este grupo, pero es el mayor causante de muertes (*Melanoma - Estadísticas*, 2021). El riesgo de mortalidad prevalece en la capacidad del melanoma en realizar metástasis en su última etapa de evolución.

La organización mundial de la salud (*Radiation: Ultraviolet (UV) Radiation and Skin Cancer*, 2017) reportó de 2 a 3 millones de casos tipo no melanoma (benigno) y 132,000 tipo melanoma en todo el mundo. Debido a que la principal razón de origen son los rayos ultravioletas, se estima que un deterioro del 10% de la capa de ozono causaría un incremento de 300,000 casos tipo no melanoma y 4,500 tipo melanoma.

Las mayores tasas de supervivencia están asociadas con etapas más tempranas, mientras que las etapas posteriores acarrean una disminución significativa en la probabilidad de sobrevivir del paciente. Un factor clave es que la evolución desde la etapa 0 a una etapa terminal se puede dar en un tiempo tan corto cómo 6 semanas.

Teniendo en cuenta las cifras anteriores sobre los casos y muertes que se presentan sobre el melanoma, la importancia del diagnóstico temprano, el desconocimiento sobre los factores de riesgo y las limitaciones de los sistemas actuales del mercado, se propone elaborar un sistema para realizar un diagnóstico asistido por computadora, que facilite la detección temprana y pueda recolectar nuevos datos para la mejora continua del modelo y la utilización de estos, para investigaciones y datos estadísticos posteriores.

3 Formulación del Problema

Existe una gran cantidad de lesiones de piel que pueden aparecer en cualquier momento de nuestras vidas, el sistema detectara 3 de ellas o señala que el usuario posee un inofensivo lunar (Melanocito Nevi), los tipos de cáncer son el Carcinoma de Células Basales, Carcinoma de Células Escamosas y el Melanoma, este último representa el cáncer de piel más letal por su característica de realizar metástasis en una etapa avanzada.

El Melanoma es responsable del 75% de las muertes por cáncer cutáneo a nivel mundial, donde la tasa de supervivencia del paciente depende de la etapa y el tiempo en años que se encuentre (Sánchez, 2021).

Para realizar un diagnóstico oportuno, el paciente debe realizar una consulta médica y procesos adicionales como una Biopsia de la piel para confirmar la presencia del melanoma, con especialistas en el área de medicina y dermatología, donde una cita médica podría tener un coste alto para las personas de bajo recursos o de poca accesibilidad a los servicios de salud.

Se necesita desarrollar un nuevo sistema de fácil acceso, altamente preciso y que pueda ser de apoyo médico, por lo tanto, en este proyecto de investigación se plantea como problema:

¿Cómo construir un clasificador de lesiones de piel para apoyar la toma de decisiones en medicina?

4 Objetivos

4.1 Objetivo General

Construir un clasificador de lesiones de piel utilizando redes neuronales convolucionales, métodos ensembles implementados como un servicio web, una aplicación móvil y aplicación web; para apoyar la toma de decisiones en medicina.

4.2 Objetivos Específicos

1. Generar un modelo de clasificación de lesiones de piel utilizando CNN y métodos en conjunto para obtener el mejor rendimiento.

2. Construir una aplicación web desarrollada con la librería Expo de React que permita utilizar los perfiles de Administrador y Dermatólogo.

3. Construir una aplicación móvil empleando React Native que permita consumir los servicios web para la plataforma Android.

5 Alcance y Limitaciones

5.1 Alcance

1. Se plantea crear un sistema de diagnóstico asistido por computadora, este tiene como propósito ayudar a los médicos o pacientes a detectar la existencia de una patología, el sistema no pretende reemplazar el trabajo de un dermatólogo en su versión inicial.
2. El sistema debe recibir la realimentación de los diagnósticos del dermatólogo para mejorar su desempeño en el futuro y escalar a diferentes lesiones.
3. El sistema será un prototipo tanto en web como móvil, por lo tanto, estará en periodo de prueba para mejorar su funcionamiento y precisión.

5.2 Limitaciones

1. Los algoritmos de aprendizaje automático depende de las imágenes, debido a que sólo se cuenta con 9 clases que corresponden al conjunto de datos del desafío ISIC, que son el Melanoma, Melanocito Nevi, Carcinoma de Células Basales, Queratosis Actínica, Queratosis Benigna, Dermatofibroma, Lesión Vascular, Carcinoma de Células Escamosas y ninguna de las anteriores; el sistema no tendrá manera de qué pueda ayudar a detectar otro tipo de lesiones, el número de lesiones qué se detecta es aún más pequeño debido a la eliminación de 5 de estas clases debido a la pequeña cantidad de ejemplos qué se tenían. Un dermatólogo está entrenado para detectar muchos más tipos de lesiones de piel, por lo tanto, este algoritmo no está próximo a reemplazarlo.

2. La cantidad de datos es muy desbalanceada debido a que existen clases con pocos ejemplos, esto impedirá qué se puedan incluir algunas de las lesiones en el modelo final por la falta de muestras, también afectará el desempeño final del modelo ya qué puede sesgar a predecir las clases de las qué posea más muestras; por lo tanto, esto imposibilita su implementación en una aplicación médica real en su primera versión.
3. La fase de entrenamiento de las CNN requiere de recursos computacionales de unidad de procesamiento gráfico “GPU”, al disponer de una con una capacidad de 8 Gigabytes no se podrá realizar entrenamiento a los modelos más sofisticados.
4. La aplicación móvil dará soporte al sistema operativo Android únicamente.
5. La aplicación web está desarrollada para los perfiles de Dermatólogo y Administrador.

6 Marco Teórico

6.1 Investigación Técnica Modelo de Clasificación

6.1.1 Inteligencia Artificial (IA).

La inteligencia artificial es la simulación de procesos de inteligencia humana por parte de máquinas, especialmente sistemas informáticos.

El término AI fue acuñado por John McCarthy en la conferencia Dartmouth de 1956, quien la definió como "la ciencia y la ingeniería de crear máquinas inteligentes, especialmente programas de computación inteligentes". Está relacionada con la tarea similar de utilizar ordenadores para comprender la inteligencia humana, pero la IA no se limita a métodos que sean observables biológicamente".

6.1.2 Aprendizaje Automático (Machine Learning)

Es un subcampo de la inteligencia artificial que trata de enseñar a las máquinas a aprender. Es definido por Tom Mitchell (Machine Learning, 1997, Mitchell) cómo “El estudio de algoritmos de computación que mejoran automáticamente su rendimiento gracias a la experiencia. Se dice que un programa informático aprende sobre un conjunto de tareas, gracias a la experiencia y usando una medida de rendimiento, si su desempeño en estas tareas mejora con la experiencia.”

Se puede catalogar el tipo de aprendizaje dependiendo del origen de los datos y la naturaleza del problema, existen otros grupos intermedios, pero los más grandes son:

- Aprendizaje supervisado.
- Aprendizaje no supervisado.
- Aprendizaje en refuerzo.

Debido a que la clasificación de melanomas se soluciona con algoritmos de aprendizaje supervisado, sólo en esa se hará énfasis.

6.1.3 Aprendizaje Supervisado

Se aplica a problemas dónde se debe usar un modelo para mapear datos de entrada a una variable (o varias) categórica o numérica de salida (cómo lo es mapear una imagen de lesión de piel a una categoría que sugiere un diagnóstico). Estos modelos son entrenados con un conjunto de datos donde el algoritmo debe realizar hipótesis que expliquen cómo convertir los predictores en la salida a la cual se le fue asignada, en caso de que la variable objetivo sea discreta se le llamara clasificación, en caso de que sea continua se le llamara regresión; después se debe evaluar esta hipótesis con más datos que el algoritmo no haya visto antes para asegurarse de qué está haciendo un buen trabajo, posteriormente las teorías que explican los datos deben ser refinadas o descartadas dependiendo del resultado usando una métrica, finalmente se itera este proceso hasta que se cumple un criterio de parada.

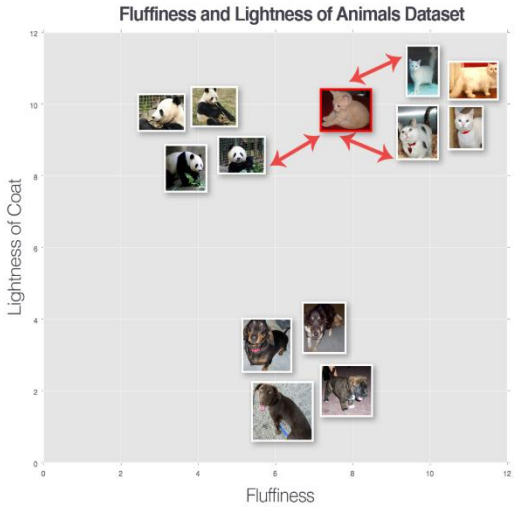


Figura 2. Clasificación imagen de gato dependiendo de sus características, Fuente (Adrian Rosebrock, Deep Learning for Computer Vision with Python Starter Bundle, PyImageSearch)

6.1.4 Aprendizaje Representación

El desempeño de los algoritmos de aprendizaje automático depende en gran manera en la forma en cómo se presentan los datos, cada pieza de información relevante de un problema se llama característica y estas serán introducidas al modelo en la fase de entrenamiento. Los modelos funcionan naturalmente bien con datos tabulares o "estructurados", sin embargo, también existen los datos "crudos" (no tabulares) cómo lo son las imágenes o el audio, para poder ser usados por un algoritmo de aprendizaje automático se debe hacer uso de una "representación" de los datos que nos permita conocer la estructura fundamental qué contienen. Históricamente en las imágenes se han usado extractores de características llamados "manuales", ya que son creados usando la pericia de un humano para emplearlas, en este grupo se incluyen métodos cómo los histogramas de gradientes orientados y los Haarcascades, ellos reciben una imagen y proveen un vector con las características que contiene la información de la imagen.

En el aprendizaje de representación no sólo plantea mapear de características a predicciones, sino que cómo paso previo aprenden una representación de los datos, un enfoque que usualmente dirige a un mejor desempeño que usar extractores de características manuales ya que se adaptan a nuevas tareas fácilmente sin intervención humana. (Introduction, Deep Learning Book, Goodfellow, Bengio, Courville)

6.1.5 Redes Neuronales Artificiales (RNA).

Algoritmo de aprendizaje automático inspirado en el cerebro que se especializa en reconocimiento de patrones y aprendizaje usando datos. Posee la gran ventaja de aprender representaciones de los datos, estos son usados para discriminar entre distintas clases.

Figura 5 Diagrama de Venn

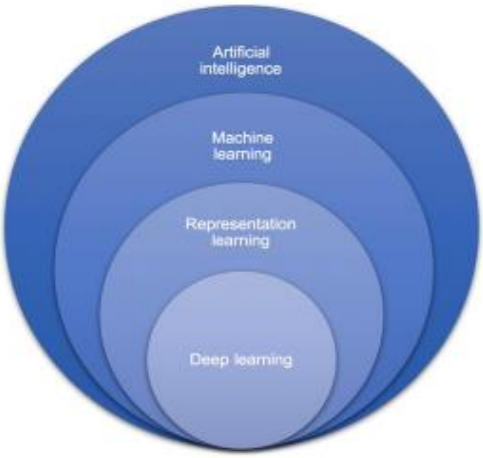
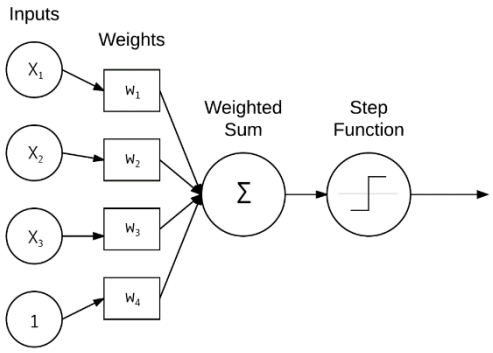


Diagrama de Venn relación inteligencia artificial, aprendizaje automático, aprendizaje de representación y aprendizaje profundo, Fuente (Mokli et al., 2019)

6.1.6 Neurona artificial.

El principal componente de este grupo de algoritmos se llama la neurona artificial o perceptrón, una unidad computacional que posee entradas a señales que son ponderadas con unos pesos, en el interior de la neurona se realiza una suma y produce una señal de salida al pasar el resultado por una función de activación.

Figura 6 Neurona Artificial



Neurona Artificial Fuente (Fuente (Adrian Rosebrock, Deep Learning for Computer Vision with Python Starter Bundle, PyImageSearch))

6.1.6.1 Peso de la neurona

Es un parámetro cuyo valor debe ser calculado con la ayuda de un algoritmo de optimización que determina la importancia de cada señal de entrada o predictor, mientras más grande sea, mayor será la relevancia de esta característica al tomar la decisión en la salida. La neurona también posee un valor llamado sesgo, este se multiplica siempre por uno y se adiciona en el interior de la neurona antes de pasar a la función de activación. Al crear la neurona, está se les asignan valores a los pesos iniciales de forma aleatoria, usualmente tomados de una distribución gaussiana de media 0 y desviación estándar de 1 (Deep Learning Book, Goodfellow, Bengio, Courville).

6.1.6.2 Función de Activación

Permite mapear la suma ponderada de su entrada a la salida de la neurona. El término "activación" hace referencia al valor umbral al que la suma debe obtener para poder activar la salida de la neurona. Una razón importante de porque las redes neuronales artificiales se comportan cómo aproximadores de funciones universales es debido a las funciones de activación, al ser no lineales permiten realizar operaciones muy complejas. Las funciones de activación se

seleccionan dependiendo de la ubicación de la neurona en la red neuronal y la tarea que intenta resolver (regresión, clasificación binaria, clasificación de múltiples clases, etc.

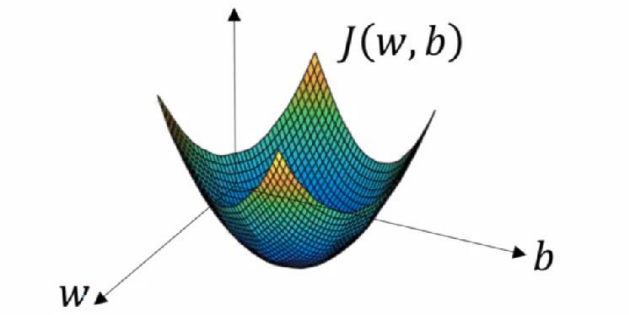
6.1.6.3 Función de Coste

Función que permite expresar la disimilitud entre las predicciones que realiza una red neuronal (u otro tipo de algoritmo de aprendizaje automático) y los valores reales. El valor se incrementa conforme los valores sean más distintos, por lo tanto, se debe disminuir para garantizar un buen funcionamiento del modelo. Esta debe tener una naturaleza convexa, de tal forma de que el valor más pequeño posible de la función sea único, también llamado óptimo global o mínimo global; el propósito es qué los valores óptimos sean fácilmente encontrados, en caso contrario se encontraron una variedad de configuraciones de parámetros subóptimas también conocidos cómo óptimos locales o mínimos locales. La función toma cómo parámetros los pesos de la red neuronal, realiza predicciones usando las características del conjunto de datos de entrenamiento (un paso conocido cómo propagación hacia adelante o forward propagation), compara las predicciones con las etiquetas o valores reales de las salidas y finalmente realiza una sumatoria para expresar en un sólo valor el desempeño actual del modelo con todos los datos.

6.1.7 Gradiente Descendiente

Algoritmo de optimización que permite actualizar los pesos de la red neuronal para reducir la función de coste. Una vez se sabe cómo difieren las predicciones del modelo con los valores reales, se deben maximizar o minimizar los pesos de las neuronas para poder alcanzar el mínimo global.

Figura 7 Gradiente descendente



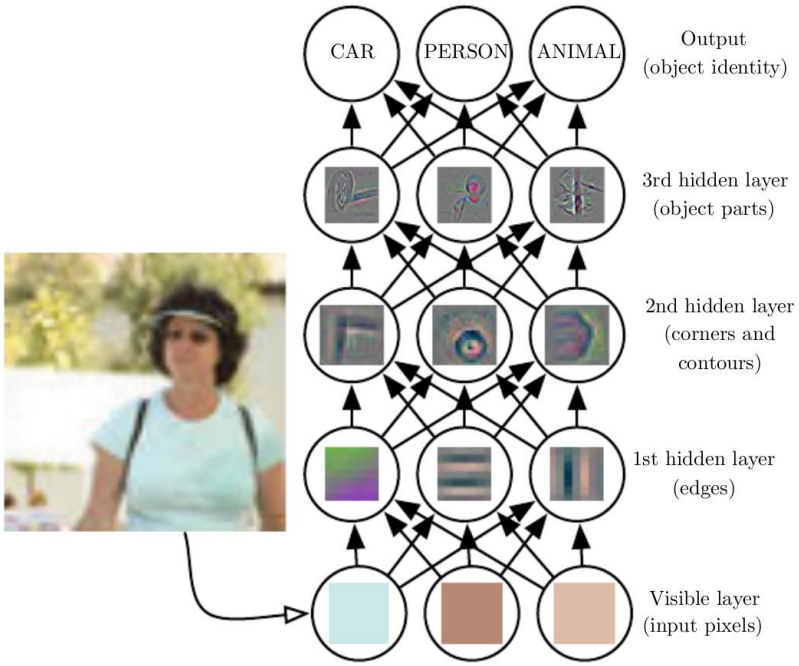
Gradiente descendente, Fuente Especialización en Deep Learning, curso Neural Networks and Deep Learning, Video gradiente descendente, diapositiva 2. Andrew NG.

Este algoritmo funciona calculando la derivada parcial de la función de coste con respecto a cada uno de los parámetros o pesos de la red neuronal que afectan el resultado, esta variable se conoce cómo el gradiente, qué es un vector con dirección y magnitud que indicará cómo se deben modificar las variables para reducir la función de coste. Una vez se conoce la dirección y magnitud, se debe multiplicar por un parámetro de valor escalar designado por el diseñador llamado la taza de aprendizaje que se encargará de dar el paso en el que se actualizan los parámetros.

6.1.8 Redes Neuronales Profundas

Al apilar neuronas artificiales juntas se obtiene una red neuronal artificial profunda (de su profundidad proviene el término Deep, de Deep Learning). Este nuevo proceso no sólo implica mapeos de predictores a salidas, sino aprender representaciones jerárquicas abstractas de los mismos datos, que ayudarán a realizar el mapeo de una forma más eficiente. El aprendizaje de representación permite el modelamiento de funciones complejas a través de otras más simples, cómo lo es la detección de personas a través de atributos más primitivos cómo líneas y curvas, que servirán para predecir piernas y brazos, que finalmente serán útiles para detectar a las personas en la salida.

Figura 8 Redes neuronales profundas



Redes neuronales profundas, Fuente Ian Goodfellow and Yoshua Bengio and Aaron Courville (2016) Deep Learning

La complejidad de la jerarquía de las representaciones se incrementará de acuerdo con la profundidad de las capas de la red neuronal, es por eso por lo que las redes mejoran el rendimiento conforme se incrementan las capas. También es necesario tener más datos para que el algoritmo pueda aprender estas representaciones, esto significa un mayor costo de recursos computacionales a cambio de un mayor desempeño a comparación de otros algoritmos de aprendizaje automático tradicionales.

Figura 9 Relación del rendimiento de Algoritmos

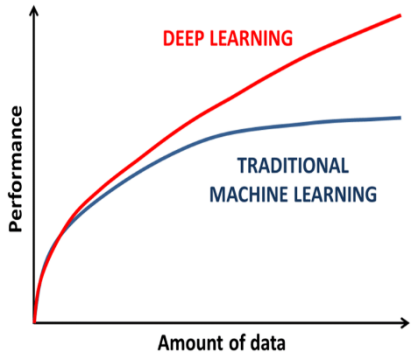


Figura 2. Relación del rendimiento de algoritmos tradicionales aprendizaje automático vs aprendizaje profundo cuando existen mayor cantidad de datos de entrenamiento, Fuente (Pesapane et al., 2018).

6.1.9 Representación Computacional de Imágenes

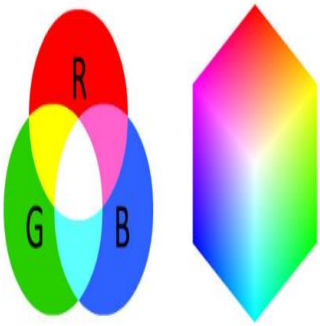
Las imágenes son representadas en forma matricial donde cada elemento contiene la unidad básica para expresar la existencia de un color llamada píxel, esta toma un rango de valores que va desde 0 donde se tiene la ausencia del color hasta 255 dónde se representa el color en su mayor intensidad.

Figura 10 Representación Computacional de Imágenes

Representación computacional de imágenes, Fuente (Fuente (Adrian Rosebrock, Deep Learning for Computer Vision with Python Starter Bundle, PyImageSearch))

Las imágenes a color se representan con 3 colores primarios, el rojo, verde y azul.

Figura 11 Canales RGB



Canales RGB, Fuente (Fuente (Adrian Rosebrock, Deep Learning for Computer Vision with Python Starter Bundle, PyImageSearch))

6.1.10 Redes Neuronales Convolucionales CNN

Son una variedad de las redes neuronales artificiales que se aplican para problemas de visión por computadora y series de tiempo, poseen unas capas especiales llamadas convolucionales, que aprenden características visuales de las imágenes.

Las convoluciones son realizadas por kernels o filtros, una matriz de números (también llamados pesos) que dependiendo de sus valores tienen la capacidad de activarse cuando cierto tipo de característica esté presente, por ejemplo, en la siguiente imagen se tiene un detector de bordes:

Figura 12 Detector de Bordes

0	-1	0
-1	4	-1
0	-1	0

edge detection filter

Detector de bordes. Adoptada Convolutional Neural Networks, 2018.Fuente. https://cezannec.github.io/Convolutional_Neural_Networks/

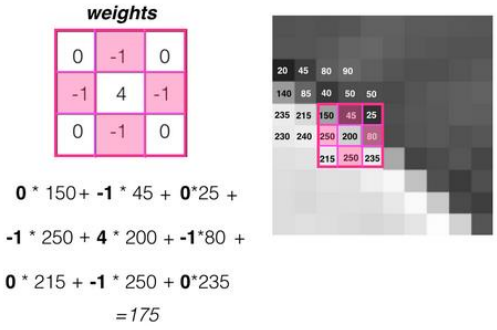
Si se enfoca en una región de una imagen del mismo tamaño del filtro se podría visualizar de la siguiente manera:

Figura 13 Pixeles en una imagen

Píxeles en una imagen, Adoptada de Neural Network,2018, Fuente. https://cezannec.github.io/Convolutional_Neural_Networks/)

Al aplicar el filtro a esta sección de la imagen se puede notar que el producto punto entre ellos obtiene un valor alto en caso de ubicarse en un borde, a esta operación se le conoce cómo convolución:

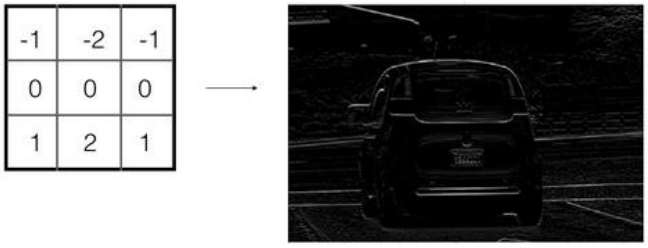
Figura 14 Convolución



Convolución, Adoptada de Neural Network,2018, Fuente. https://cezannec.github.io/Convolutional_Neural_Networks/)

El resultado de la convolución se le conoce cómo mapa de características, en ella se puede notar cómo se resaltan todos los bordes de la imagen original:

Figura 15 Mapa de Características



Mapa de Características, Adoptada de Neural Network,2018, Fuente. https://cezannec.github.io/Convolutional_Neural_Networks/

Las capas convolucionales posteriores aprenderán atributos no de las imágenes originales, sino de los mapas de características de sus capas predecesoras. Los pesos de los filtros se inician aleatoriamente y se modifican usando un algoritmo de optimización basado en gradiente descendente.

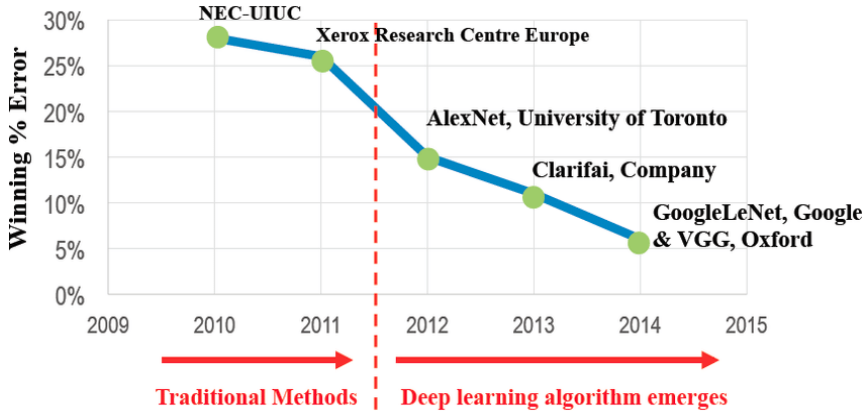
Una práctica común es aplicar una capa llamada max pooling a el mapa de características resultante de la convolución, con el objetivo de reducir las dimensiones del mapa y preservar las características que se hallaron.

Figura 16 Max Pooling

Max Pooling, Adoptada de Neural Network,2018, Fuente. https://cezannec.github.io/Convolutional_Neural_Networks/

En el caso de la visión por computadora, se tiene como punto de referencia para seleccionar el algoritmo estado del arte en la clasificación de imágenes la competencia ILSVRC (ImageNet Large Scale Visual Recognition Challenge), dónde se analiza la precisión de estos en una base datos que contiene mil categorías a clasificar y 1.2 millones de imágenes. Desde el 2012 se pueden ver a las redes neuronales convolucionales (CNN) y sus variaciones como el algoritmo de aprendizaje de mayor desempeño:

Figura 17 Ganadores Competencia ILSVRC



Ganadores Competencia ILSVRC desde 2010 al 2015, Fuente Shawahna, Ahmad & Sait, Sadiq & El-Maleh, Aiman. (2018) FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification

Los parámetros y las conexiones de la red diferencian de forma particular, a esto se le conoce cómo su tipo de arquitectura, cuando una arquitectura de CNN se convierte en el estado del arte en la competencia ILSVRC, esta puede ser utilizada con sus pesos como modelo base para una nueva tarea de clasificación, esto con el propósito de hacer uso de la detección de patrones en la amplia base de datos ImageNet, de tal forma de que sólo se necesite adaptar la red a detectar patrones un poco distintos en lugar de iniciar un aprendizaje de patrones desde cero con pesos aleatorios.

6.1.11 Arquitecturas de una CNN

Las CNN poseen parámetros configurables por el diseñador, cuando cierta manera de usar filtros o conectar las capas se hace exitosa se le bautiza con el nombre de una arquitectura, a continuación, se expondrán las que fueron usadas en el proyecto.

6.1.12 Arquitectura ResNet

Arquitectura ganadora de la competencia ILSVRC del 2015. Su mayor innovación es la implementación de bloques residuales, estos se ubican en las capas convolucionales y poseen conexiones de salto qué conectan la entrada directamente con la salida cómo se muestra en la figura.

Figura 18 Bloque Residual de unas ResNet

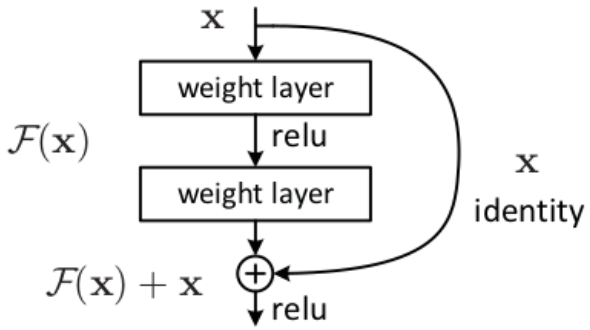


Figure 2. Residual learning: a building block.

. Bloque residual de una ResNet. Adoptada de Cornell University,2015, Fuente. https://arxiv.org/abs/1512.03385

Está estructura logra obtener redes más profundas (por primera vez más de 100 capas cuando fue creada) combatiendo los siguientes problemas de estabilidad en el entrenamiento:

1. Desvanecimiento de gradiente: redes muy profundas generan gradientes muy pequeños qué no actualizan los pesos de las neuronas.

2. Explosión de gradientes: redes muy profundas generan gradientes muy grandes qué producen actualizaciones de pesos tan grandes que impiden a la red neuronal converger a una solución.

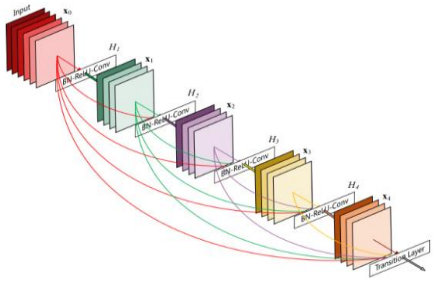
6.1.13 Arquitectura MobileNet

Está arquitectura está optimizada y utiliza convoluciones separables profundas para construir redes neuronales profundas y livianas. Introduce hiper parámetros globales simples qué pueden comprometer el tiempo y la precisión, estos permiten al diseñador seleccionar el modelo de tamaño apropiado para su aplicación en función de las restricciones del problema.

6.1.14 Arquitectura DenseNet

Implementación de redes neuronales convolucionales densas, dónde la parte convolucional está compuesta por bloques convolucionales, y en ellos cada capa individual está conectada a todas las capas siguientes hacia adelante del bloque.

Figura 19 Conexiones entre un bloque convolucional denso de una DenseNet

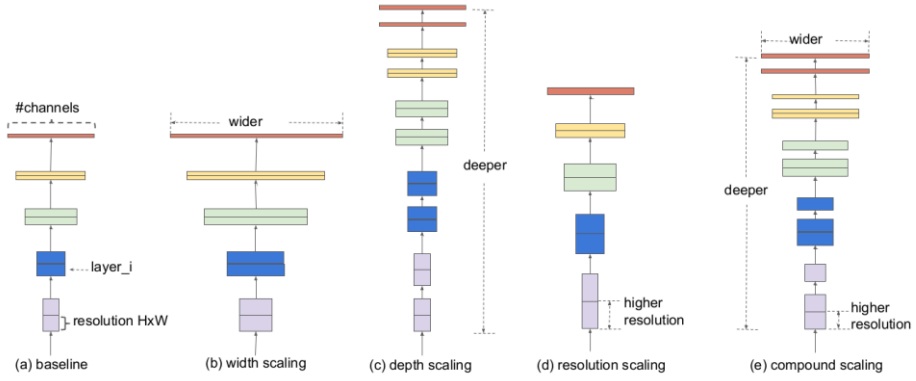


Conexiones entre un bloque convolucional denso de una DenseNet, Adaptada de Cornell University,2015, Fuente https://arxiv.org/abs/1608.06993

6.1.15 Arquitectura EfficientNet

Métodos tradicionales para mejorar el desempeño de una red neuronal incluyen un marco de aprendizaje residual para facilitar el entrenamiento de redes más profundas cómo la ResNet (Arxiv, s.f.). Sin embargo, en esta arquitectura se aplica el escalado del modelo balanceando la profundidad, anchura y resolución usando un coeficiente compuesto para mejorar su desempeño. El funcionamiento intuitivo consiste en que, si la imagen es más grande, la red convolucional necesita más capas para incrementar su campo receptivo y más canales para aprender patrones más detallados.

Figura 20 Diferentes tipos de escaladas EfficienNet



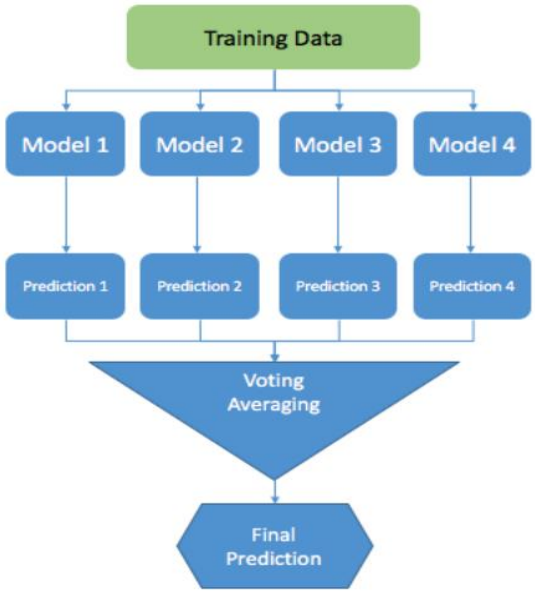
Diferentes tipos de escalados EfficientNet, Fuente Mingxing Tan and Quoc V. (2020) EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.

6.1.16 Aprendizaje en Conjunto (Ensemble Learning)

Para lograr un mayor desempeño se planea crear múltiples clasificadores de CNN que trabajen en conjunto, de esta manera se crea un super clasificador en conjunto (ensemble) que funcione

mejor que cada uno de ellos de forma individual, así aprovechando las habilidades de cada modelo y crear un clasificador más robusto.

Figura 21 Aprendizaje en Conjunto



Aprendizaje en conjunto, algoritmo qué toma el promedio de las predicciones de cada clasificador individual. Fuente Vijayalakshmi Natarajan, Dipayan Sarkar Ensemble Machine Learning Cookbook.

6.1.17 Métodos de Aprendizaje en Conjunto

- Modelo en conjunto promedio (Model average ensemble) cada modelo es tenido en cuenta de igual manera al tomar las predicciones y sacar un promedio con las probabilidades.
- Modelo en conjunto promedio ponderado (Weighted average ensemble) permite la contribución de cada miembro del conjunto qué sea ponderado de forma proporcional a

la confianza de cada miembro usando el conjunto de datos de evaluación, la ponderación de cada modelo se encuentra haciendo uso de búsqueda exhaustiva.

- Modelo en conjunto re-muestreado (Resampling Ensembles) consiste en realizar diferentes particiones al conjunto de datos y realizar entrenamientos independientes para cada partición (a esto se le conoce también cómo validación cruzada) y así obtener los miembros del ensemble.
- Modelos de épocas contiguas con conjuntos de votación horizontal (Models from Contiguous Epochs with Horizontal Voting Ensembles) obtiene los diferentes miembros del ensemble de diferentes épocas en un mismo proceso de entrenamiento.
- Método en conjunto snapshot (Snapshot Ensemble) también usa los modelos qué resultan de un mismo proceso de entrenamiento, pero realiza variaciones agresivas a la taza de aprendizaje qué reducen la varianza de un modelo.
- Método en conjunto de generalización apilada (Stacked Generalization Ensemble) entrena un nuevo modelo independiente qué aprende la mejor manera de combinar las predicciones de modelos existentes.
- Método en conjunto con pesos promedios del modelo (Average Model Weight Ensemble) toma el peso promedio de cada uno de los pesos durante el proceso de entrenamiento, este método reduce el ruido del proceso de optimización del modelo qué entrega un grupo de pesos más estables y frecuentemente resultados mejores.

6.1.18 Métricas para evaluar modelos de aprendizaje automático.

- Al crear modelos clasificadores, estos cuando son evaluados deben asignar etiquetas a nuevas muestras, en este proceso se puede encontrar los siguientes escenarios:

- A una muestra le es asignada correctamente que contiene la etiqueta que el clasificador debe asignar, a esto se le conoce como un verdadero positivo **TP**. (Un ejemplo es cuando una persona con cáncer es correctamente diagnosticada por el modelo).
- A una muestra no le es asignada de forma correcta la etiqueta que provee el clasificador, a esto se le conoce como un verdadero negativo **TN**. (Un ejemplo es cuando una persona saludable es diagnosticada por el modelo qué no posee la enfermedad)
- A una muestra le es asignada incorrectamente la etiqueta que provee el clasificador, a esto se le conoce como un falso positivo **FP** (Un ejemplo es cuando una persona sana le es diagnosticada con cáncer)
- A una muestra no le es asignada incorrectamente la etiqueta que provee el clasificador, a esto se le conoce como un falso negativo **FN** (Un ejemplo es cuando una persona con cáncer es diagnosticada como saludable)

De los anteriores, el error más fatal es un Falso Negativo **FN**. Esto significa decirle a un paciente con la enfermedad qué se encuentra bien, e impedir qué este realice más pruebas y empiece tratamiento lo antes posible, este error podría costar la vida del paciente. Un Falso Positivo **FP** es incómodo ya qué un paciente saludable es erróneamente diagnosticado con la enfermedad, sin embargo, después de realizada la biopsia se podrá confirmar el error y ninguna vida humana será comprometida.

6.1.19 Matriz de Confusión

Forma tabular de representar la cantidad de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

6.1.19.1 Exactitud

Cantidad de predicciones que fueron realizadas de forma correcta.

$$\frac{TP + TN}{TP + TN + FP + FN}$$

6.1.19.2 Sensibilidad

Medida de cuántos pacientes que padecen la enfermedad fueron correctamente identificados.

$$\frac{TP}{TP + FN}$$

El modelo debe maximizar la sensibilidad, ya qué está se optimiza al encontrar la mayoría de las personas enfermas.

6.1.19.3 Especificidad

Medida de cuántos pacientes que son sanos fueron correctamente identificados.

$$\frac{TN}{TN + FP}$$

El modelo debe tener en cuenta cuantos de los pacientes saludables fueron encontrados, sin embargo, se determina a la sensibilidad con mayor prioridad.

6.1.19.4 Curva ROC

Gráfica que enseña cómo están relacionadas la sensibilidad y la especificidad en un modelo.

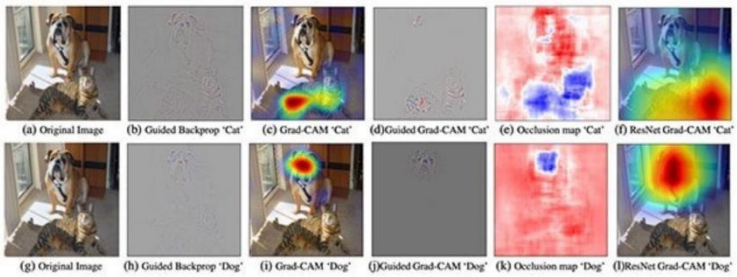
6.1.19.5 Área Bajo la Curva AUC

Medida del área por debajo de la curva ROC.

6.1.20 Interpretabilidad

Grad-CAM (Cámara de Gradientes): Algoritmo qué permite visualizar los mapas de calor en una imagen producidas por las salidas de la red convolucional de un modelo entrenado

Figura 22 Visualización mapa de calor



Visualización mapa de calor para clasificador de perros y gatos usando Grad-CAM Fuente (PyimageSearch)

6.2 Investigación Técnica Ingeniería de Software

6.2.1 Ingeniería de Software

La ingeniería de software es una de las áreas de la informática, que se compone de un conjunto de metodologías y etapas para el desarrollo e implementación de software de calidad. Se conoce también como desarrollo de software o producción de software. Donde generalmente se emplea el ciclo de vida del software, que se trata de etapas como análisis, diseño, desarrollo, pruebas, calidad e implementación. (Unir, 2021)

6.2.2 Metodología de Desarrollo de Software

A través de procedimientos y una estructura definida, las metodologías se encargan de la organización y ejecución de un proyecto de software; teniendo como objetivo principal garantizar la calidad del software. (Santander Universidades, 2021)

6.2.2.1 Metodología Tradicional

Teniendo en cuenta la filosofía de desarrollo de las metodologías, aquellas con mayor énfasis en la planificación, control, documentación, especificación precisa de requisitos y modelado de un proyecto de software son las Metodologías Tradicionales; destacadas por ser metodologías más predecibles que adaptables.

6.2.2.2 Metodología Ágil

A diferencia de las metodologías tradicionales, las ágiles proponen un nuevo paradigma en el desarrollo de software, orientando su estructura a las personas más que al propio proceso, donde basan su estructura organizacional y de planificación en la adaptación del proyecto a diferentes entornos y circunstancias.

Entre las metodologías ágiles más destacable, se tienen: Scrum, Extreme Programming (XP) Programación Extrema, Lean Kanban, Dynamic Systems Development Methods (DSDM) Métodos de Desarrollo de sistemas dinámicos, Feature Driven Development (FDD) Desarrollo basado en funcionalidades. (Feda, 2019)

6.2.2.2.1 SCRUM

Es la metodología más popular y utilizada para proyectos de software, su enfoque está en ser una metodología de trabajo orientada a los roles y la distribución de objetivos y tareas; que se evalúan, analizan y se adaptan constantemente mediante las conocidas Scrum Daily Meeting reuniones diarias del equipo SCRUM. Para este proyecto SCRUM ofrece el manejo de documentación y organización ideal para adaptarnos a los diferentes cambios y resultados que se obtienen de la investigación y planteamiento de nuestro proyecto. (Euroforum, 2019)

6.2.3 Diagrama UML

El lenguaje Unificado de Modelado (UML) está desarrollado para describir, diseñar, especificar, visualizar, construir y documentar los componentes de un software. Mediante diferentes diagramas como son los diagramas de clases, componentes, despliegue, actividades, secuencia, casos de uso, entre otros. (Quirós, 2021)

6.2.3.1 Diagrama de Estado

Representa los estados por los cuales un objeto recorre durante su vida en una aplicación en respuesta a eventos, respuestas y acciones. (Zr, 2019)

6.2.3.2 Diagrama de Caso de Uso

Son diagramas que describen de forma general los requisitos funcionales para cada usuario también llamado actor. (Cevallos, 2015)

6.2.3.3 Diagrama de Despliegue

Se utiliza para modelar la disposición física de los componentes del software.

Los diagramas de UML se pueden diseñar con diferentes herramientas de escritorio o web. (Cillero, 2016)

6.2.3.3.1 StarUML

Es un software desarrollado para modelar los diferentes diagramas de UML.

6.2.4 Arquitectura de Software

El diseño general del sistema se entiende como la arquitectura, que va a seguir el software en relación con sus componentes y comunicación. Existen 10 patrones de arquitectura dependiendo de la relación y las partes que conforman el software a desarrollar. Los patrones son: Patrón de Capas, Cliente-Servidor, Maestro-Esclavo, Filtro de tubería, Intermediario, de Igual a igual, Bus de evento, Modelo-Vista-Controlador, Pizarra e Intérprete. (Novoseltseva, 2021)

6.2.4.1 Patrón de Capas

Es una arquitectura de software que establece tres secciones como son la capa de presentación, la capa de negocio y la capa de datos.

6.2.5 Base de Datos

Son datos que en conjunto pertenecen a un mismo contexto y se almacenan sistemáticamente para su posterior uso. Existen principalmente dos tipos de base de datos; la relacionales y no relacionales. Las bases relacionales son un conjunto de datos que se organizan mediante tablas formalmente descritas, la información se establece en partes pequeñas que se integran y relacionan para luego realizar diferentes acciones a través del lenguaje de consultas estructurado (SQL), los gestores de bases de datos relacionales más conocidas y utilizadas son Oracle, MySQL Microsoft SQL Server, PostgreSQL y DB2.

Por otro lado, las bases de datos no relacionales no utilizan SQL como lenguaje de consultas, no se definen estructuras fijas como tablas de almacenamiento de datos y resultan siendo más adaptables a las actualizaciones. Las bases de datos no que se destacan son Casandra, Redis, MongoDB y Firebase. (Rendón, 2019)

6.2.5.1 Firebase

Es una plataforma de Google que ofrece servicios tanto para aplicaciones web como móviles. Para el almacenamiento de la información se utiliza Realtime Database, Store y Firestore.

6.2.6 Diseño de Interfaz de Usuario GUI

Tanto las aplicaciones web como móviles, deben tener una interfaz de usuario donde el cliente pueda interactuar con la aplicación de manera gráfica y deductiva. Wireframe, es la forma por la cual se realizará un primer diseño general de la aplicación, como parte de un prototipo. Para construir este diseño podemos hacer uso de software y herramientas como Lucidchart, Adobe XD, Sketch, entre otros. (Workana, 2021)

6.2.6.1 Adobe XD

Es un editor gráfico desarrollado para diseñar prototipos de experiencia de usuario dirigidos para páginas web y aplicaciones móviles.

6.2.7 Lenguajes de Programación

Comprende un lenguaje formal que está diseñado para organizar algoritmos y procesos lógicos que se emplean en un ordenador o sistema de información permitiendo controlar su comportamiento físico, lógico y su comunicación con el usuario. (Equipo editorial, Etecé, 2021)

6.2.7.1 Desarrollo Web

Siendo las aplicaciones web un software común para comercio, educación, los medios de comunicación entre otros. Todo desarrollo web está compuesto por dos partes una es el Back-End donde tenemos la parte del servidor y base de datos. Y el Front-End donde podemos definir la parte visual o interfaz de usuario. (de OpenClassrooms, 2017)

6.2.7.1.1 FastApi

Es un Framework web rápido, moderno y de alto rendimiento para la utilización de API con Python.

6.2.7.2 Desarrollo Móvil

Para el desarrollo móvil el lenguaje de programación se define generalmente según el sistema operativo al cual esté dirigido, ya sea Android o iOS donde se considera como Desarrollo Nativo y Cross-Platform (Desarrollo Multiplataforma). (Siripathi, 2017)

6.2.7.3 Desarrollo Multiplataforma

Se entiende como el desarrollo de aplicaciones en conjunto para más de una plataforma, donde su uso frecuentemente se da en el desarrollo híbrido entre aplicaciones web y móviles. Donde para esto, se utilizan Frameworks como Ionic, Angular o React. (ABAMobile, 2021)

6.2.7.3.1 React Native

Es un framework de JavaScript para el desarrollo de aplicaciones nativas para IOS y Android, basado en React.

6.2.8 Pruebas de Software

Son procesos que se enfocan en evaluar la calidad del software de forma interna y externa; Existen dos técnicas para realizar Pruebas de Software y tipos de pruebas que se clasifican en funcionales o no funcionales. (Lee, 2021)

6.2.8.1 Técnica de caja negra (Funcionalidad)

Son pruebas que se enfocan en las entradas y salidas del sistema, teniendo en cuenta los requerimientos del software y especificaciones funcionales.

6.2.9 Despliegue de Modelos de Deep Learning

Se entiende como despliegue, cuando llevamos un modelo ya construido a un ambiente en donde diferentes clientes puedan acceder y hacer uso de este en tiempo real. (Rodríguez, 2020)

6.2.9.1 Google Cloud Plattform

Es una gran infraestructura en la nube que se encarga de proporcionar diferentes servicios en la creación e implementación de aplicaciones y sitios web con Google.

6.2.9.2 Docker

Es una plataforma que permite crear, probar e implementar aplicaciones de software en contenedores individuales.

6.2.9.3 Centos

Es un sistema operativo de Linux que se utiliza en los servidores por su estabilidad, consistencia y fácil administración.

6.3 Investigación Médica de las Lesiones de Piel

6.3.1 Lesiones de la Piel

Existen diferentes tipos de lesiones de piel, entre esas se encuentra las pápulas, que se caracterizan por que son elevadas, suelen medir menos de 10 mm de diámetro y se pueden sentir o palpar. Las lesiones que pertenecen a este tipo son lunares, verrugas, liquen plano, picaduras de insectos, queratosis seborreicas y actínicas, algunas lesiones por acné y cánceres de piel. (Benedetti, 2021)

6.3.2 Cáncer

Es un proceso de crecimiento y diseminación incontrolados de células, que se puede manifestar en cualquier parte del cuerpo. Cuando ya existe un tumor este suele invadir el tejido circundante y llegar a provocar metástasis en otros órganos y lugares del cuerpo. La mayoría de cáncer se podrían prevenir evitando la exposición a factores de riesgos comunes, como también se pueden curar mediante cirugía, radioterapia o quimioterapia principalmente si se detectan en una fase temprana. (Puente, 2019)

6.3.3 Cáncer de Piel

Siendo una característica principal del cáncer, el crecimiento y la diseminación incontrolados de células, para el cáncer de piel, las células que presentan estos procesos son específicamente las que conforman los tejidos de la piel. Generalmente se suele desarrollar en la piel que ha sido expuesta al sol, como también en zonas que no están frecuentemente expuestas a la luz solar. Son tres los principales cánceres de piel: Carcinoma Basocelular, Carcinoma espinocelular, Melanoma. (Valera, 2020)

6.3.4 Melanoma

Es uno de los tres cánceres de piel y se considera es el más letal, su desarrollo ocurre en las células llamadas melanocitos que producen melanina, el cual es el pigmento que da color a la piel. El principal signo de alarma de la presencia de un melanoma es el cambio de tamaño, forma, color o textura de un lunar. (Sánchez-Monge, 2021)

Figura 23 Melanoma

Imagen de un Melanoma. Adaptada de Melanoma, Mayo Clínica, 2021.Fuente. https://www.mayoclinic.org/es-es/

6.3.5 Melanocito Nevi

También llamado lunar, son maculas o nódulos de color piel a color marrón, que se conforman por nidos de melanocitos nevocitos, y suelen aparecer en la infancia o adolescencia. Una de sus características es la similitud al melanoma, es poco probable que un lunar se torna maligno, pero pacientes con gran número de lunares benignos (>50) tienen mayor riesgo de desarrollar melanoma. (Rodríguez, 2019)

Figura 24 Melanocito Nevi

Imagen de un Melanocito Nevi. Adaptada de Imágenes de Lunares, Manual MSD,2020, Fuente. https://www.msdmanuals.com/

6.3.6 Carcinoma de Células Basales

Es el cáncer de piel más frecuente, inicia con un crecimiento descontrolado o lesiones de las células basales que se localizan en la capa celular basal de la parte inferior de la epidermis. Se calcula que hasta un 50 % de las personas que han sido diagnosticadas con carcinoma basocelular desarrollan un nuevo cáncer dentro de los próximos 5 años. (BBC News Mundo, 2017).

Figura 25 Carcinoma de Células Basales

Imagen de un Carcinoma de Células Basales. Adaptada de Ejemplos de Carcinoma Basocelular, Manual MSD,2020, Fuente. https://www.msdmanuals.com/

6.3.7 Queratosis Benigna

Esta lesión se caracteriza por el aumento en la cantidad de ortoqueratina y/o paraqueratina, es de las lesiones blancas más comunes. El área bucal que afecta es extensa involucrando borde lateral y la superficie ventral de la lengua. (Pérez, 2015)

Figura 26 Queratosis Benigna

Imagen de Queratosis Benigna. Adaptada de Queratosis Benigna, Salud Dental Para Todos,2015, Fuente. https://www.sdpt.net/

7 Metodología

7.1 Metodología Modelo de Clasificación

A continuación, se listan los pasos para la creación de los modelos de clasificación:

- Exploración del conjunto de datos: Se visualizó la información qué contiene los metadatos de los pacientes, las imágenes de las lesiones de piel y las etiquetas de diagnóstico, se hizo uso de Pandas, una librería de ciencia de datos utilizada para explorar y hacer reportes de todas las características de cada individuo. Se visualizó las imágenes con ayuda de Matplotlib, librería para manipular imágenes y visualizaciones, esto se realizó en compañía de un dermatólogo para corroborar que las imágenes y sus diagnósticos estén libres de errores (aunque no se inspeccionaron el total de las imágenes), también de que estos datos son representativos de lo que se encontraría en el mundo real. Posteriormente se analizó la distribución de las clases. Se debe tener en cuenta cuáles clases están sobre representadas o subrepresentadas y se tomaron estrategias para realizar el entrenamiento ya que esto puede sesgar el modelo a predecir comúnmente la clase qué mayores muestras posee.
- Preprocesamiento de las imágenes: Adaptación de las imágenes para que puedan ingresar al modelo, esto incluye considerar ajuste de tamaño que depende de la CNN que se planea usar y reducción de canales de imágenes de color a grises. Finalmente se examinaron técnicas de aumento de datos, ¿qué modificaciones se pueden realizar a las imágenes para poder ingresarlas en la fase de entrenamiento como ejemplos nuevos? Se consideró: adición de ruido aleatorio, elevar el brillo, rotación, traslación, giro, volcados horizontal y vertical, zoom in y zoom out.
- Entrenamiento del modelo: Se realizó la carga de los diferentes modelos para realizar transferencia de aprendizaje. Se hicieron descargas desde TensorFlow Hub de modelos

EfficientNet, ResNet, MobileNet y DenseNet pre entrenados con ImageNet, esto reducirá los tiempos de entrenamiento. Posteriormente, se probaron diferentes maneras para mejorar las redes de forma individual.

- Evaluación del modelo: Se monitoreo las métricas Falso Negativo **FP**, Área Bajo la Curva **AUC,** perdida, y Exactitud, también se analizó la matriz de confusión para saber con qué clases se presentan más dificultades y se tomaron decisiones para mejorar los modelos.
- Iteración: Se reiteraron los pasos anteriores hasta obtener los mejores modelos posibles con los recursos disponibles para las 4 arquitecturas propuestas.
- Aplicabilidad: Se usó la técnica GradCam para visualizar los píxeles qué más influyen en la toma de la decisión.
- Generación modelo para dispositivos embebidos: Haciendo uso de TensorFlow Lite, se hizo un recorte a las capacidades de MobileNet para qué funcione de forma más eficiente en un entorno computacional restringido.
- Aprendizaje en conjunto (ensemble): Se usó el método en conjunto de votos promedio para optimizar el comportamiento de todos los modelos en la tarea de clasificar lesiones de piel. Las demás propuestas en el marco teórico no fueron exploradas por limitaciones de tiempo.

7.2 Metodología Ingeniería de Software

7.2.1 Análisis

Para realizar la construcción y desarrollo de la aplicación web y móvil; se utilizará la metodología ágil SCRUM por la flexibilidad y eficiencia de su estructura, además de ser la metodología en la que se tiene mayor conocimiento y documentación para uso.

De esta metodología se trabaja con artefactos como Historias de Usuarios necesarias para establecer las funcionalidades del sistema, el Product y Sprint Backlog que se compone de una lista con descripciones generales de los requisitos del sistema, prioridad, esfuerzo y el orden de realización de cada uno de estos.

7.2.2 Base de Datos

El uso de una base de datos en el sistema es para realizar la persistencia de datos, ya que se mantendrá los datos principalmente para estudios estadísticos, el reentrenamiento del modelo posteriores a un largo periodo de implementación del software, y funciones del Dermatólogo en el aplicativo. De lo anterior se utilizan los servicios de Firebase como Realtime Database, Store y Authentication. Se elige el uso de Firebase por la extensa documentación disponible y la experiencia previa de su funcionamiento.

7.2.3 Diseño

7.2.3.1 Diagrama UML

Se trabajará con el Diagrama de Despliegue, el Diagrama de Estado y Diagrama de casos de uso, debido a su estructura y concepto se consideran adecuados para describir todo el sistema desde la interacción del cliente con el sistema, hasta la relación general del software con sus componentes. El diseño de los diagramas se realiza con la aplicación StarUML.

7.2.3.2 Arquitectura de Software

Se realizará un diseño del sistema donde se trabajará el Patrón de Capas, dividiendo el sistema en tres secciones: presentación, negocio y datos.

7.2.3.3 Mockups

La herramienta con la que se trabaja para el Wireframe es Adobe XD, se elige este software de diseño porque se tiene experiencia previa en cuanto al funcionamiento y sus servicios son los adecuados para el diseño de la aplicación web y móvil.

7.2.4 Desarrollo

7.2.4.1 Web

Se desarrollará con React Native junto a Expo por la experiencia previa que se tiene en el uso de estas herramientas y la reutilización del código del aplicativo móvil.

7.2.4.2 Back-End

Para el funcionamiento del servidor y la comunicación con el Modelo de Clasificación, se trabaja con PYTHON y su Framework FastApi, porque es el lenguaje más utilizado en esta área y en el que tiene mejor desempeño en estos proyectos.

7.2.4.3 Móvil

El desarrollo móvil será un desarrollo nativo para Android, programado con el de Lenguaje JavaScript. Utilizando el Framework de React Native.

7.2.5 Pruebas

Se llevarán a cabo pruebas de software, con la técnica de caja negra debido al alcance del proyecto y su funcionalidad. Se utilizarán Pruebas Unitarias y pruebas de funcionalidad.

7.2.6 Despliegue Desarrollo

Para llevar a cabo este despliegue del modelo de Deep Learning y poder validar los requerimientos funcionales, utilizamos Docker con los servicios de Google Cloud Platform en un servidor con Centos.

8 Secuencia y Actividades que se Desarrollarán

8.1 Modelo de Clasificación

8.1.1 Exploración del conjunto de datos.

Visualización de la información del paciente.

8.1.2 Preprocesamiento de las imágenes

Se adaptarán las imágenes para poder ser ingresadas al modelo de aprendizaje automático y se consideran opciones para aumento de datos.

8.1.3 Entrenamiento del modelo.

Prueba de diferentes CNN, ajuste de parámetros y entrenamiento.

8.1.4 Evaluación del modelo.

Estudio de la matriz de confusión, de las métricas sensibilidad y especificidad usando AUC. Modificar el modelo partiendo de la matriz de confusión.

8.1.5 Iteración.

Repetir la selección de modelos y evaluación hasta obtener el mejor modelo posible con los recursos de tiempo y GPU disponible.

8.1.6 Aprendizaje en conjunto.

Selección de los mejores modelos, probar técnicas de aprendizaje en conjunto, mayoría de votos y votos promedio.

8.2 Ingeniería de Software

8.2.1 Análisis

1. Levantamiento de información: Se realizó la investigación médica correspondiente al contexto del problema y una entrevista a un dermatólogo de la ciudad.
2. Historias de Usuario: Se desarrollaron los requerimientos del sistema, según la información recolectada en el paso anterior
3. Backlog: Se trabajó con las plantillas de SCRUM para el Product y Sprint Backlog, donde definiremos la prioridad, tareas y el esfuerzo de cada una.

8.2.2 Diseño

1. Diagrama UML: Se construyeron los diagramas de Estado, Casos de uso, y Despliegue. También se elabora el diagrama de Entidad Relación que representa la estructura de la base de datos del aplicativo como también el Diccionario de Datos.
2. Arquitectura de Software: Se construyó el proyecto en base a la arquitectura por capas o también llamada tres niveles, capa de presentación, capa de negocio y capa de datos.
3. Mockups: Se realizó un diseño general y gráfico de la aplicación web y móvil.

8.2.3 Programación e Integración

1. Conexión con Modelo Inteligencia Artificial: Se establece la comunicación e interacción de la aplicación móvil con la capa de

negocio (Servicio Web) que tenemos en la nube de Google Cloud Plattform.

2. Conexión con Firebase: Se implementa al proyecto los servicios de Firebase como Base de datos, autenticación y almacenamiento
3. Programación Web y Móvil: Se realiza el desarrollo de la aplicación web y móvil con las respectivas tecnologías mencionadas anteriormente.

8.2.4 Pruebas

Se implementan las pruebas unitarias y de funcionalidad a los componentes más importantes de la aplicación móvil.

8.2.5 Despliegue

Se realiza la creación de la instancia, para luego configurarla con el sistema operativo Centos, e iniciar la implementación del servicio web.

9 Cronograma

Cronograma de Actividades												
Actividades	1	2	3	4	5	6	7	8	9	10	11	12
Modelo de Clasificacion												
1. Exploracion del conjunto de datos												
2. Preprocesamiento de las imágenes												
3. Entrenamiento del modelo												
4. Evaluacion del modelo												
5. Iteracion												
6. Aprendizaje en Conjunto												
Aplicación web y movil												
Analisis												
1. Levantamiento de Informacion												
2. Historias de Usuario												
3. Producto Backlog												
4 Sprint Backlog												
Diseño												
5. Mockups												
6. Diseño de Arquitectura												
7. Diagrama UML												
Programación e Integración												
8. Conexión con el modelo de IA												
9.Programacion web y movil												
10. Pruebas de Software												
11. Despliegue												

10 Análisis y Diseño

10.1 Análisis

10.1.1 Entrevista al Dermatólogo

¿Qué datos adicionales tiene en cuenta para determinar el tipo de lesión?

Ocupación, exposición al sol, antecedentes de melanoma, antecedentes de otros tipos de cáncer, ubicación en el cuerpo.

¿Queremos con el aplicativo almacenar las imágenes que ingresen, es antiético hacia el paciente realizarlo?

No considera que sea un problema guardar las imágenes, mientras sean con fines educativos o estadísticos.

Notas: Todas las demás preguntas se pueden encontrar en los Anexos.

10.1.2 Historias de Usuario

Las siguientes historias de usuario se desarrollan en base a la información recolectada de una entrevista realizada con un Dermatólogo de Previred y de la investigación previa sobre el tema.

HU-01

Enunciado de la historia					Criterios de aceptación			
Identificador (ID) de	Rol	Característica / Funcionalidad	Razón / Resultado	Número (#) de	Criterio de aceptación (Título)	Contexto	Evento	Resultado / Comportamiento esperado
HU-01	como usuario	necesito acceder a la aplicación	para hacer uso de los servicios del aplicativo	1	Botones de Inicio	Cuando el usuario ingrese al aplicativo		El sistema mostrara un boton 'Ingresar
				2	Logo	Cuando el usuario ingrese al aplicativo		El sistema mostrara en el centro de la vista, el logo del aplicativo.

HU-02

HU-02	como usuario	necesito acceder a la aplicación	para hacer uso de los servicios del aplicativo	1	Registro	Luego de dar click ' Ingresar'	Click en 'Registro'	El sistema mostrara dos input, para el correo y la contraseña, y un boton para terminar el registro.
				2	Inicio de Sesion	Cuando el usuario ingrese al aplicativo	Click en 'Inicia Sesion'	El sistema mostrara dos input, para el correo y la contraseña, un boton para recuperar contraseña 'Olvidaste tu contraseña' , y un boton para ingresar a la aplicación 'Ingresa'

Se registró un total de 11 historias de usuario, donde tenemos 5 historias dirigidas a los usuarios, 2 historias al rol Dermatólogo, 1 historia dirigida al rol Administrador, y 3 historias al sistema.

Nota: Podemos visualizar todas las historias de usuario en los Anexos.

10.1.3 Backlog

10.1.3.1 Rúbrica de Priorización

Para evaluar el grado de prioridad o impacto de una historia de usuario en el desarrollo de un proyecto, se puede determinar a partir de la siguiente Tabla:

PUNTUACIÓN	METODOLOGÍA				
	5	4	3	2	1
RÚBRICAS	Requisito que tiene relación directa con la entrega del artefacto (impacto considerable).	Historia de usuario que tiene dos o más dependencias, para ser completado.	Historia de usuario que depende de que otra ya esté completa.	Historia de usuario que se puede hacer en cualquier momento, que puede afectar el tiempo de entrega del artefacto pero tiene relevancia para retrasar otras actividades.	Historia de usuario que se puede hacer en cualquier momento que no afecta la entrega del artefacto.

10.1.3.2 Product Backlog

El backlog está construido a partir de la Rúbrica de priorización y de la técnica Planning Póker para la valoración del esfuerzo que exige cada historia de usuario.

Identificador (ID) de la Historia	Enunciado de la Historia	Alias	Estado	Dimensión / Esfuerzo	Iteración (Sprint)	Prioridad
HU-01	Como usuario, necesito una pantalla de bienvenida	HU-01-Bienvenida	Terminada	8	1	1
HU-02	Como usuario, necesito hacer uso de la aplicación donde tengo la opcion de registro o login	HU-02-Registro y Login	Terminada	8	1	1
HU-03	Como usuario, necesito verificar el correo ingresado, y recuperar la contraseña	HU-03-Verificar correo y recuperar contraseña	Terminada	21	2	2
HU-04	Como sistema, necesito controlar el acceso a la aplicación mediante roles	HU-04-Roles	Terminada	55	2	4
HU-05	Como un usuario, necesito visualizar , tener y elminar los resultados que he generado	HU-05-Opciones Usuario	Terminada	55	2	4

Nota: La información adicional del Product Backlog, se encuentra en los Anexos.

10.1.3.3 Sprint Backlog

Teniendo en cuenta el Product Backlog, detallamos las tareas correspondientes a cada historia y definimos que empleamos 2 Sprint.

Identificador (ID) de item de product backlog	Enunciado del item de Product Backlog	Tarea	Dueño / Voluntario	Estatus
HU-01	Como usuario, necesito una pantalla de bienvenida	Diseñar los mockups	Maria Ximena Rodriguez	Terminada
		Codificar la vista e interaccion	Maria Ximena Rodriguez	Terminada
HU-02	Como usuario, necesito hacer uso de la aplicación donde tengo la opcion de registro o login	Diseñar los mockups de login y registro	Maria Ximena Rodriguez	Terminada
		Codificar la interfaz grafica y su funciones	Maria Ximena Rodriguez	Terminada
		Validar los campos	Maria Ximena Rodriguez	Terminada
HU-03	Como usuario, necesito verificar el correo ingresado, y recuperar la contraseña	Codificar el boton para recuperar contraseña	Maria Ximena Rodriguez	Terminada
		Codificar el envio del enlace para confirmar correo	Maria Ximena Rodriguez	Terminada

Nota: La información adicional del Sprint Backlog se encuentra en los Anexos.

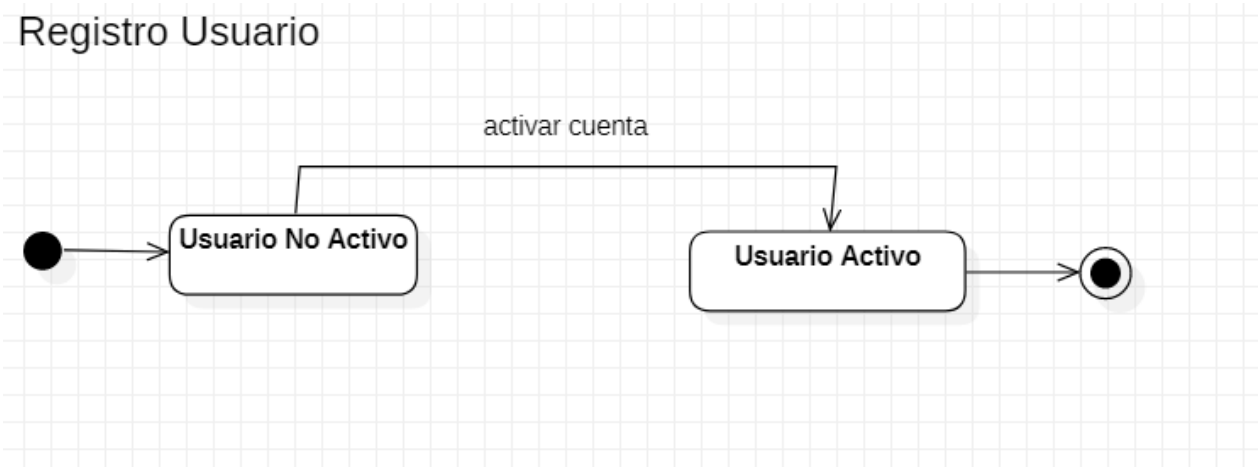
10.2 Diseño

10.2.1 Diagrama UML

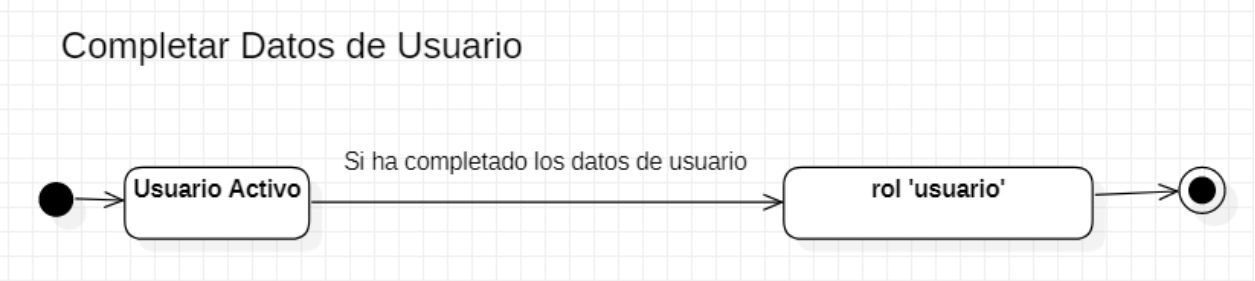
10.2.1.1 Diagrama de Estado

Se describe el flujo de estados que se emplean en la aplicación móvil.

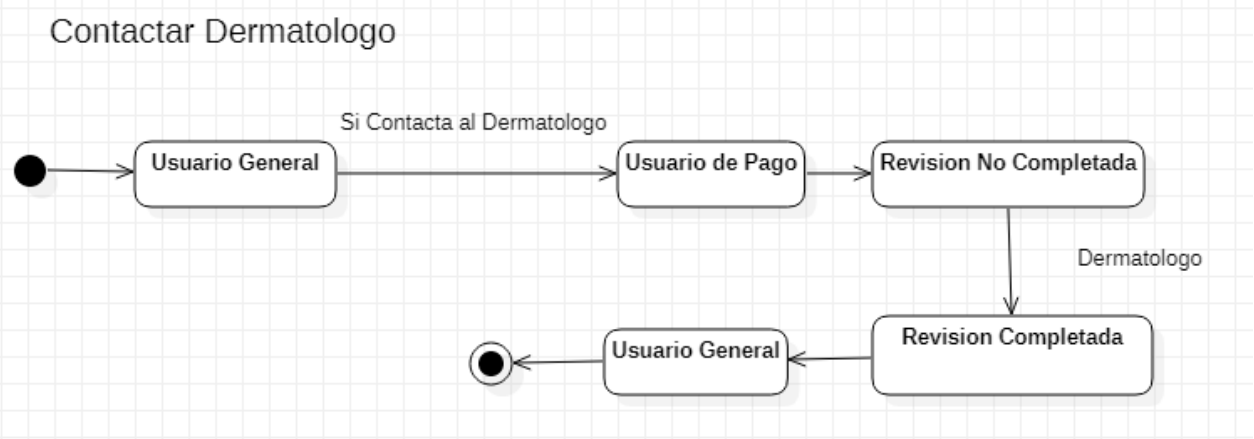
Cada vez que un usuario se registra queda como NO activo. Cuando el usuario active su cuenta a través del enlace de verificación que ha sido enviado a su correo, se actualizará el estado a activo y podrá ingresar a la aplicación.



Cuando un usuario está activo e ingresa por primera vez a la aplicación, deberá completar su registro a través de un formulario, luego de que finalice su registro, se le asignará un rol ‘usuario’ que le dará acceso a la pantalla principal.

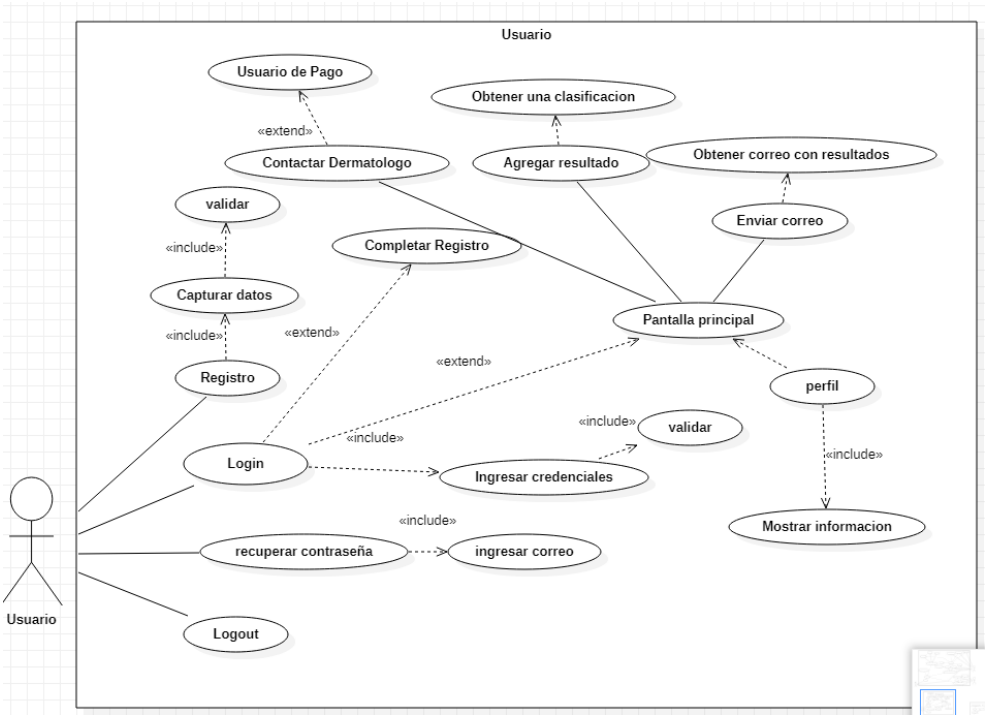


Inicialmente todo usuario tiene como estado Usuario General, cuando un usuario general desee contactar con el dermatólogo, su estado cambia a Usuario de Pago. Una vez el Dermatólogo marque al usuario como revisado este volverá a su estado inicial.



10.2.1.2 Diagrama de Caso de Uso

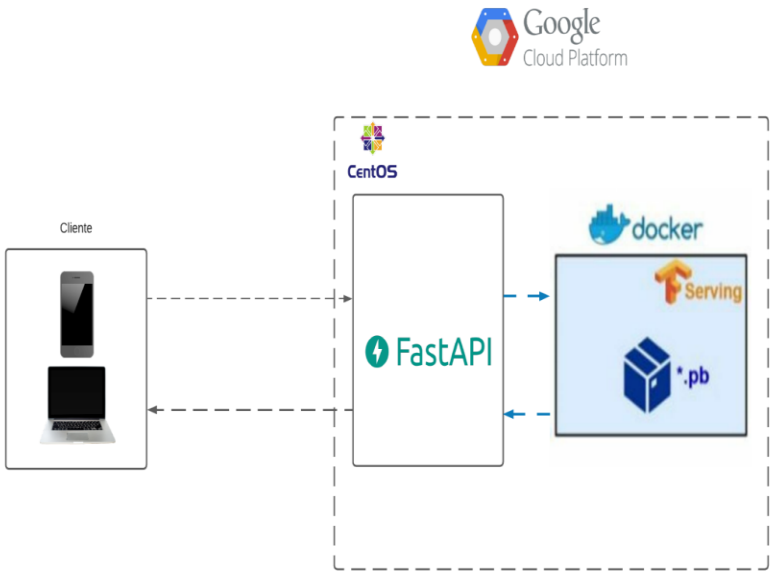
El diagrama de caso de uso se realiza describiendo la interacción de cada actor con el sistema individualmente, como por ejemplo el siguiente caso de uso es del actor Usuario.



Nota: El diagrama de caso de uso para Administrador y Dermatólogo se encuentran en los Anexos.

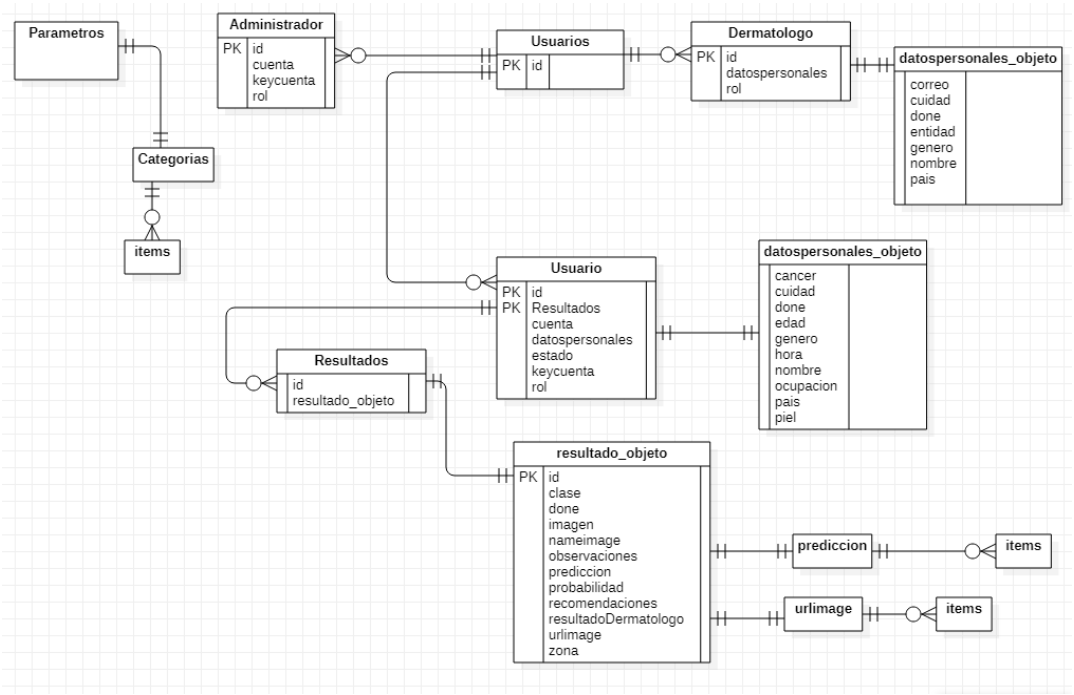
10.2.1.3 Diagrama de Despliegue

Se describe cómo está estructurado el despliegue del servicio web.



10.2.1.4 Diagrama de Entidad Relación

Se describe la estructura y relación de las colecciones, de cómo se almacena la información en Firebase.



10.2.1.5 Diccionario de Datos

Se describen los campos o atributos de cada colección con su respectivo tipo de dato y descripción.

Colección Usuario

Colección Usuario		
Campo	Tipo de dato	Descripcion
id	PK	PrimaryKey
Resultados	Colección	resultados del usuario
cuenta	Texto	correo del usuario
datospersonales	Objeto	informacion personal del usuario
estado	Boleano	cambiar de estado
keycuenta	Texto	id de la cuenta
rol	Número	perfil al que pertenece el usuario

Colección de los Resultados

Colección Resultados		
Campo	Tipo de dato	Descripcion
id	PK	PrimaryKey
resultado_objeto	Objeto	informacion de cada resultado

Información de cada Resultado

resultado_objeto		
Campo	Tipo de dato	Descripcion
id	PK	PrimaryKey
clase	Texto	clase que da el modelo
done	Boleano	cambiar estado
imagen	URL	url de la imagen en Firebase
nameimage	Texto	nombre de la imagen
observaciones	Texto	las observaciones que da el dermatologo
prediccion	Arreglo	los valores que da la prediccion respecto a todas las clases
probabilidad	Número	probabilidad de la clase que da el modelo
recomendaciones	Texto	las recomendaciones que da el dermatologo
resultadoDermatologo	Texto	el nuevo resultado del dermatologo
urlimage	Arreglo	las url de las imágenes adicionales
zona	Texto	ubicación de la lesion

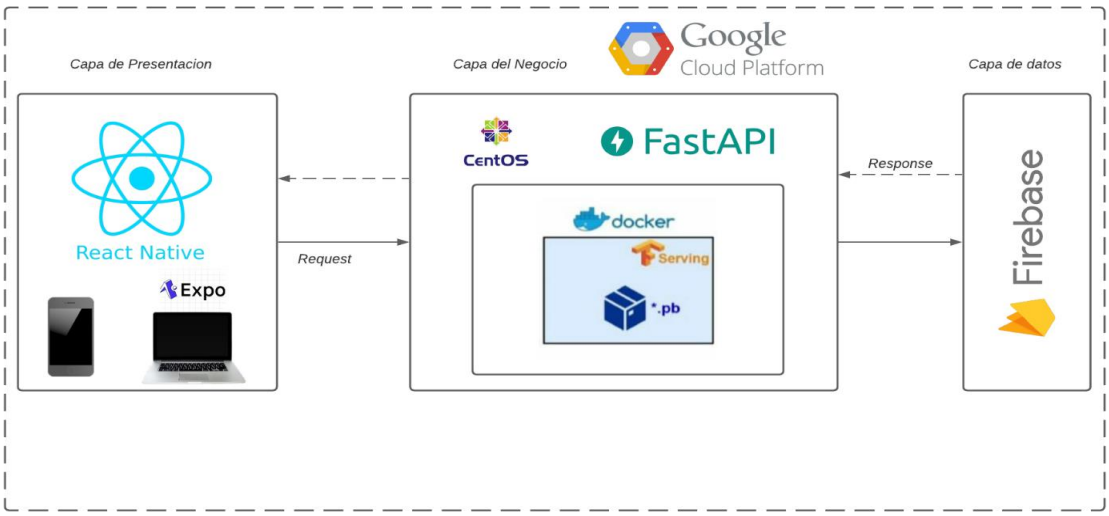
Información del Usuario

datospersonales_objeto / Usuario		
Campo	Tipo de dato	Descripcion
cancer	Texto	antecedentes de cancer
cuidad	Texto	cuidad
done	Boleano	cambio de estado
edad	Número	edad
genero	Texto	genero
hora	Texto	cantidad de horas que se expone el usuario al sol
nombre	Texto	nombre
ocupacion	Texto	ocupacion
pais	Texto	pais
piel	Texto	antecedentes de cancer de piel

Nota: Las colecciones adicionales se encuentran en Anexos.

10.2.2 Arquitectura de Software

Arquitectura de tres niveles, capa de presentación, capa del negocio, y capa de datos



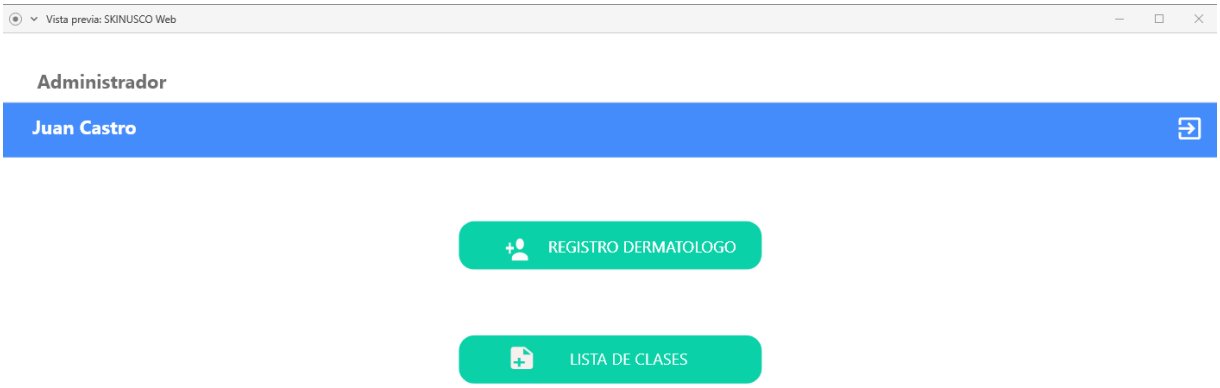
10.2.3 Mockups

Los diseños de cada aplicación se realizaron en Adobe XD

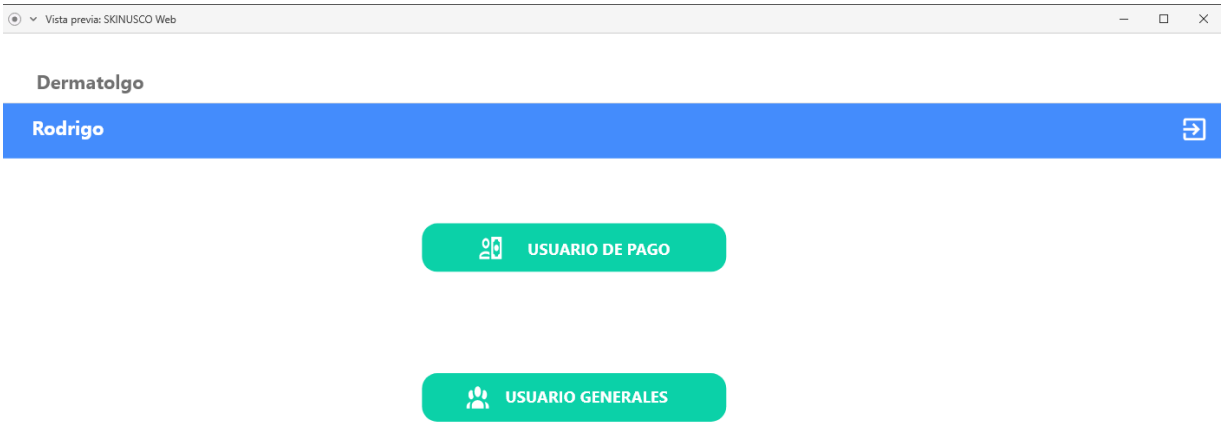
10.2.3.1 Web

A continuación, tenemos los dos perfiles de usuario de la aplicación web, con su respectivo diseño de la pantalla principal.

Administrador



Dermatólogo

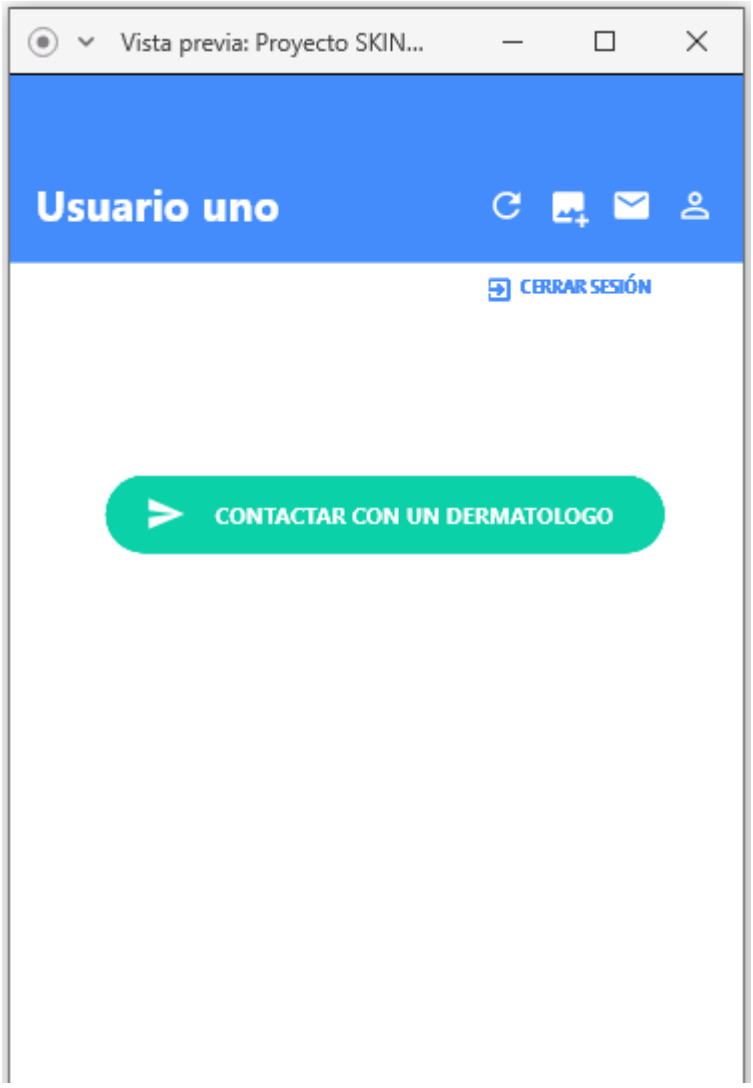


Nota: Los diseños adicionales de la aplicación web se encuentran en los Anexos.

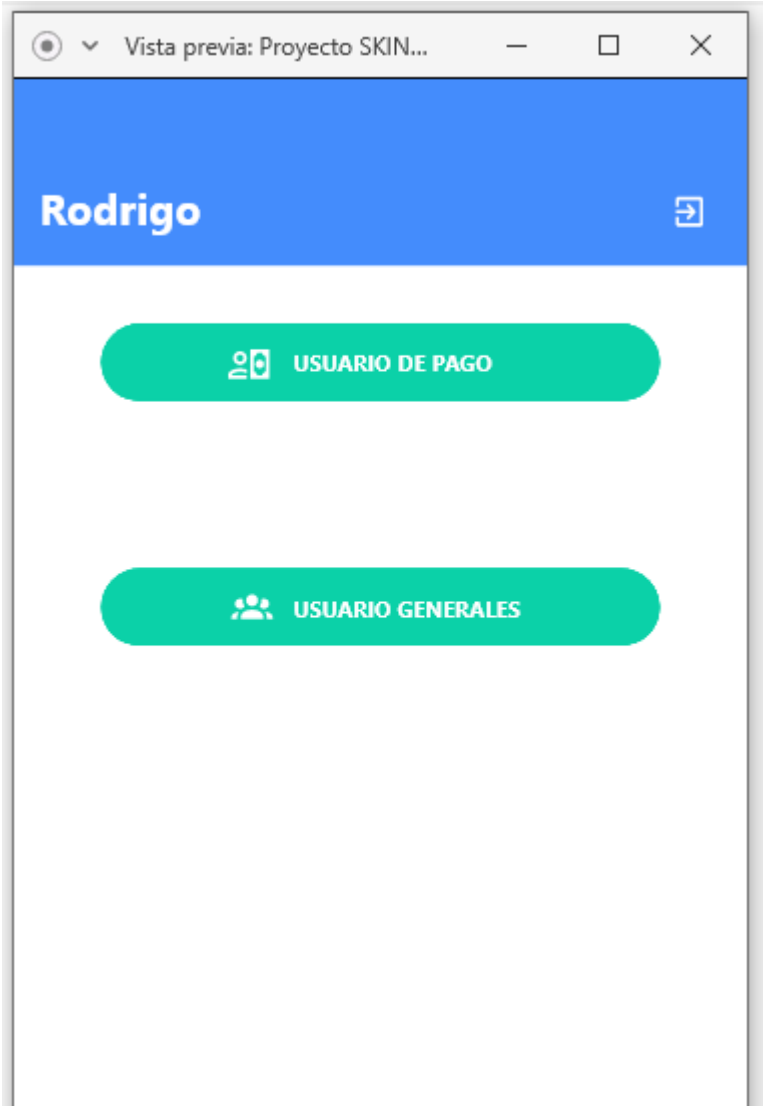
10.2.3.2 Móvil

Los diseños de los perfiles de usuario con su respectiva pantalla principal.

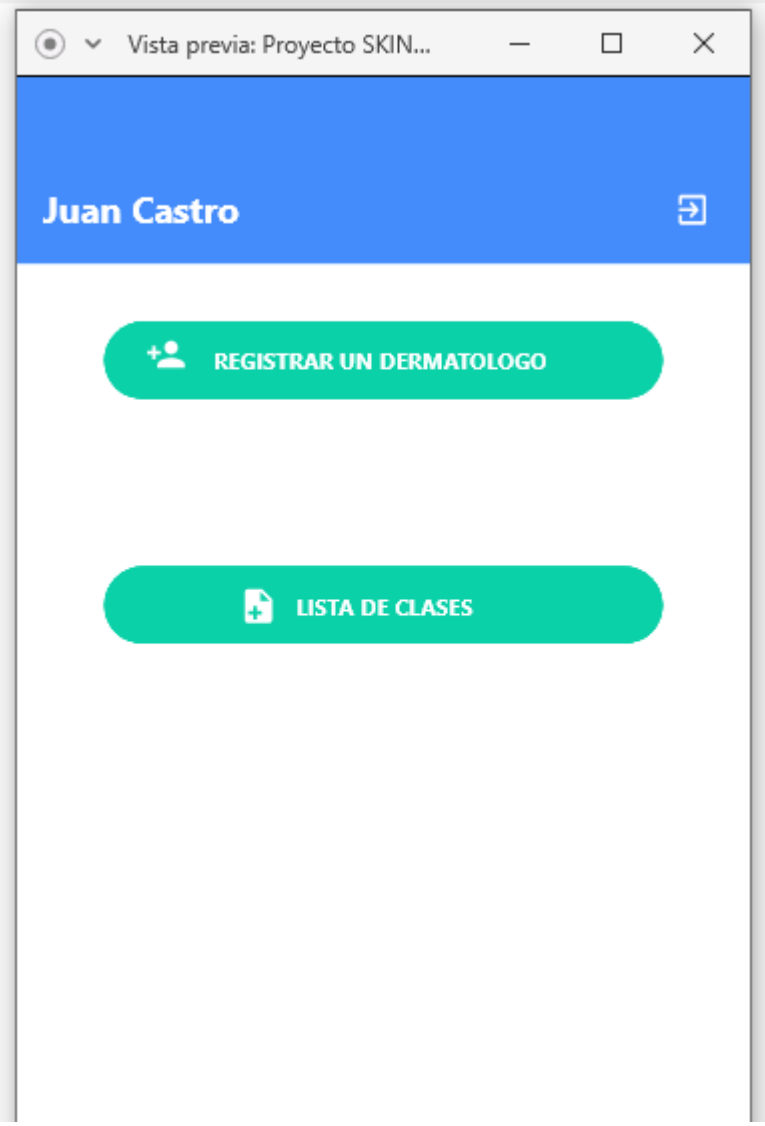
Usuario



Dermatólogo



Administrador



Notas: Los diseños adicionales de toda la aplicación están en los Anexos

11 Implementación Modelo de Clasificación

11.1 Creación modelo de redes neuronales convolucionales

11.1.1 Adquisición de datos

Las imágenes de las lesiones de piel se obtienen de la competencia ISIC del 2019, al descargar el contenido, se obtienen los siguientes archivos:

- **ISIC_2019_Training_Input**: Carpeta que contiene las imágenes de las lesiones de piel.
- **ISIC_2019_Training_GroundTruth.csv**: archivo que asocia la ubicación de las imágenes con su correspondiente etiqueta.
- **ISIC_2019_Training_Metadata.csv**: archivo que contiene datos adicionales de los pacientes cómo su edad y ubicación de la lesión de piel, esta información no se incluyó en el modelo de entrenamiento y por lo tanto no fue utilizada.

11.1.2 Análisis de datos

Se analizaron los datos desde el archivo csv, usando la librería **Pandas** para el análisis de datos, este archivo se convierte en una estructura de datos llamado Data Frame, el cual facilita la manipulación de la información y entrega la siguiente lectura de los primeros cinco registros:

```
import pandas as pd
df = pd.read_csv('ISIC_2019_Training_GroundTruth.csv')
df.head()
```

	image	MEL	NV	BCC	AK	BKL	DF	VASC	SCC	UNK
0	ISIC_0000000	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	ISIC_0000001	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	ISIC_0000002	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	ISIC_0000003	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	ISIC_0000004	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Exploración conjunta de datos.

Acá se puede ver que el nombre de la imagen carece de ubicación del archivo y extensión de la imagen (JPEG, PNG, etc..), también de que la lesión está determinada por un valor 1.0 en la columna correspondiente de cada enfermedad. Al explorar las dimensiones del Data Frame se puede ver que existen 25331 imágenes en total y 10 columnas:

```
print(f"El dataset consiste en {df.shape[0]} records {df.shape[1]} columnas")
```

```
El dataset consiste en 25331 records 10 columnas
```

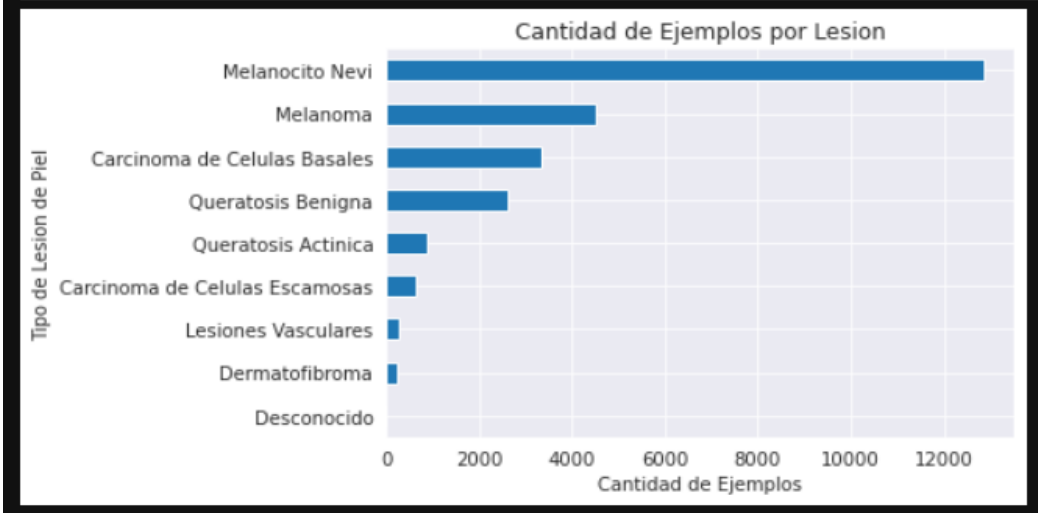
```
print(f"Las columnas son {df.columns} \n Donde  tenemos el nombre del archivo de la direccion de la imagen y 9 clases
```

```
Las columnas son Index(['image', 'MEL', 'NV', 'BCC', 'AK', 'BKL', 'DF', 'VASC', 'SCC', 'UNK'], dtype='object')
 Donde  tenemos el nombre del archivo de la direccion de la imagen y 9 clases
```

Exploración de datos, filas y columnas del conjunto de datos.

Después de cambiar las abreviaciones de los nombres por las lesiones correspondientes, se realizó un diagrama de barras para analizar la cantidad de ejemplos que existen de cada una de las clases:

```
df.drop(["imagen"], axis=1).sum().sort_values().plot(kind="barh")
plt.title("Cantidad de Ejemplos por Lesion")
plt.xlabel("Cantidad de Ejemplos")
plt.ylabel("Tipo de Lesion de Piel");
```



Exploración de datos, distribución de clases.

Del diagrama se puede apreciar que se presenta un desbalance de la cantidad de ejemplos por cada clase, idealmente, debería existir una cantidad similar de ejemplos para cada una de las enfermedades, pero en este caso la mayoría de las imágenes corresponden a Lunares (Melanocito Nevi), mientras que hay una menor proporción de Dermatofibroma y Lesiones Vasculares, se observa que la clase “Desconocido” no posee ejemplo alguno y puede ser descartada

Se incluye la ubicación de las imágenes en la carpeta **ISIC_2019_Training_Input** con la extensión jpg:

```
BASE_DIR = 'ISIC_2019_Training_Input'
EXTENSION = '.jpg'
```

```
def path_converter(input_dir):
    return os.path.join(BASE_DIR, input_dir + EXTENSION)
```

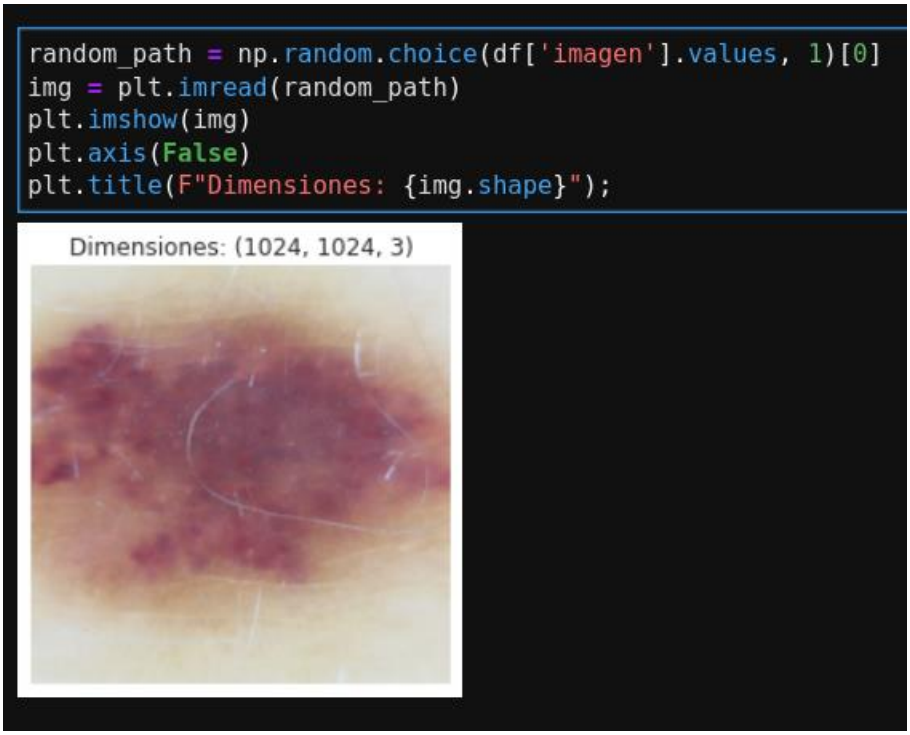
```
df['imagen'] = [path_converter(x) for x in df['imagen']]
```

```
df.head()
```

	imagen	Melanoma	Melanocito Nevi	Carcinoma de Celulas Basales	Queratosis Actinica	Queratosis Benigna	Dermatofibroma	Lesiones Vasculares	Carcinoma de Celulas Escamosas
0	ISIC_2019_Training_Input/ISIC_0000000.jpg	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	ISIC_2019_Training_Input/ISIC_0000001.jpg	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	ISIC_2019_Training_Input/ISIC_0000002.jpg	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	ISIC_2019_Training_Input/ISIC_0000003.jpg	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	ISIC_2019_Training_Input/ISIC_0000004.jpg	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Exploración de datos después de limpieza.

Con la ayuda de la librería Matplotlib, se realiza la visualización de una imagen muestra:

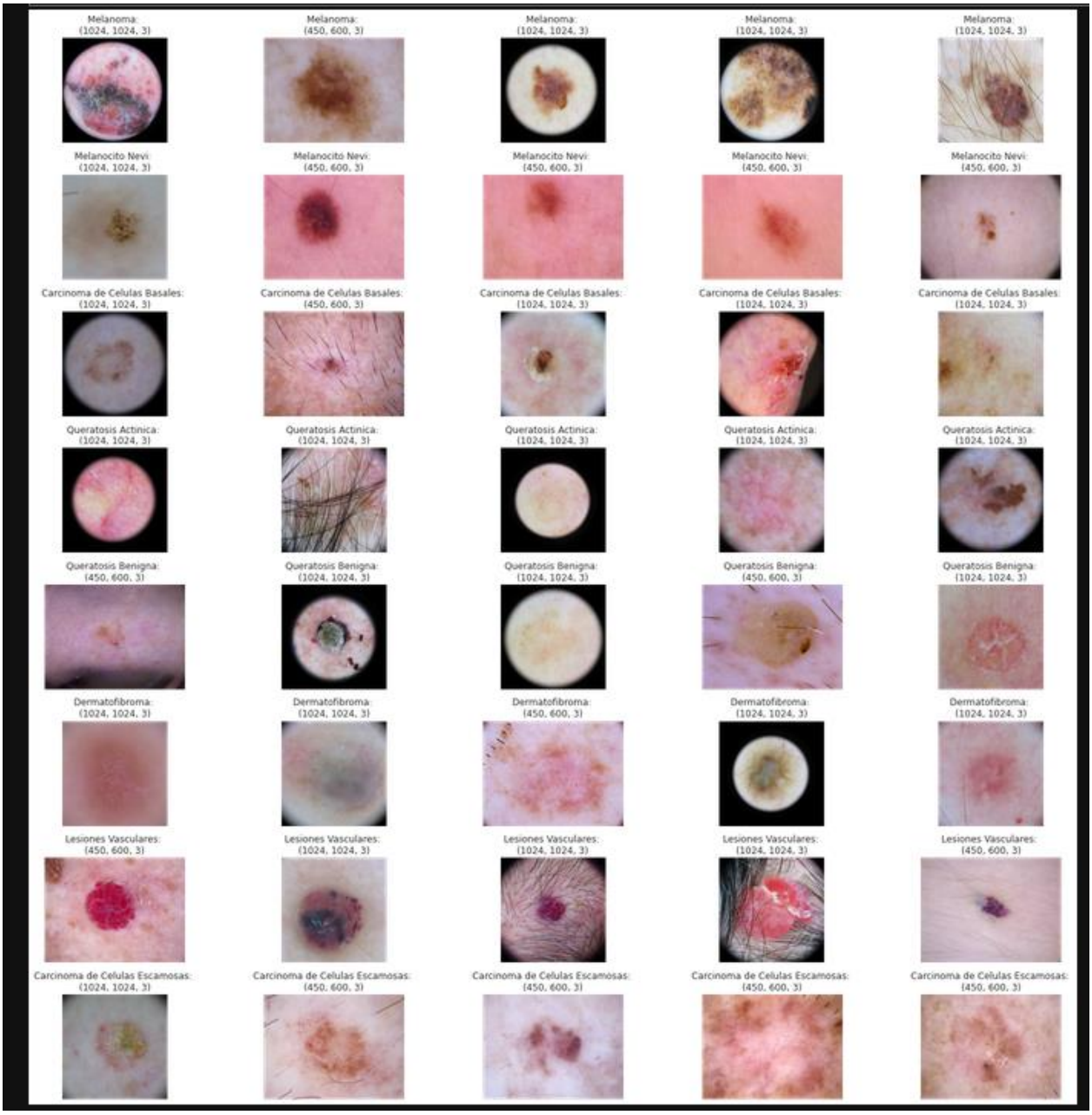


Exploración de imágenes de lesiones de piel.

Después de visualizar diferentes muestras, se puede notar qué la anchura y la imagen no es consistente y que esta varia en diferentes visualizaciones, los 3 canales de color se mantienen consistentes (no existen imágenes a blanco y negro):

Inconsistencia en las dimensiones de las imágenes de lesiones de piel.

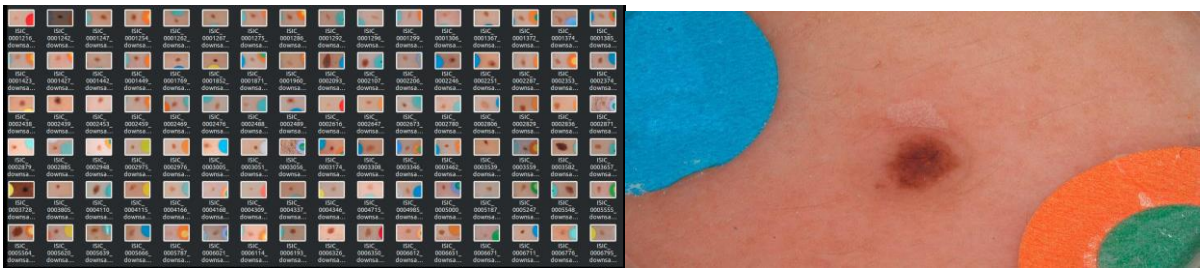
Visualización de 5 ejemplos correspondientes a cada una de las clases:



Visualización de todas las clases de lesiones de piel incluidas en el dataset.

Al observar las imágenes, se encontró con que algunas de las lesiones de piel fueron señaladas con marcadores por parte de los dermatólogos, con el propósito de qué está estos no sean determinados como un predictor por parte del algoritmo para la toma de decisión, se verifica de

forma manual de qué ninguna imagen posea marcador, y de tener uno asegurarse de ser removido de la imagen:



Problemas en algunas imágenes del conjunto de datos por presencia de marcadores qué usan los dermatólogos para apuntar a las lesiones.

11.1.3 Creación de los conjuntos de datos de entrenamiento, validación y evaluación:

Con el propósito de entrenar el algoritmo, tenemos que diferenciar qué imágenes se usarán para qué aprenda las características y qué imágenes se usarán para evaluar el rendimiento del clasificador, se hace uso de la librería de aprendizaje automático scikit learn para hacer la partición, se destina un 70% de datos para entrenamiento, 15% para validación y el 15% restante para evaluación:

```
from sklearn.model_selection import train_test_split

train, validation = train_test_split(df, test_size=0.3)
validation, test = train_test_split(validation, test_size=0.5)
```

Partición del conjunto de datos para entrenamiento, validación y evaluación.

Una vez cada imagen es asignada a un grupo, estas deben ser asignadas a diferentes carpetas para poder diferenciarse y no mezclarse entre diferentes grupos, la fuga de una imagen en otro grupo se conoce como “data leakage” y podría introducir resultados engañosos al modelo. Al incluir cada imagen en su correspondiente carpeta, se debe asegurar que estas contengan unas

dimensiones estándar de 224 de anchura por 224 de altura y sus 3 canales de colores, de que cada imagen sea un tipo de dato arreglo de Numpy que contenga valores float de 32 bits y que sea asignado en una subcarpeta de su respectiva lesión de piel.

```
from tqdm import tqdm
from tensorflow.keras.preprocessing.image import load_img, img_to_array, save_img

for img_dir, clase in tqdm(zip(test["imagen"], test["clase"])):
    img = load_img(img_dir, target_size=(224, 224))
    array = img_to_array(img, dtype=np.float32)
    save_img(os.path.join("data/test",  clase, os.path.split(img_dir)[1]), array)
```

```
3800it [00:58, 64.73it/s]
```

Preprocesamiento de datos.

Debido al fuerte desbalance que existe en la cantidad de ejemplos de cada clase, es importante mitigar este problema al crear datos sintéticos, estos se pueden generar a partir de realizar modificaciones a las imágenes del conjunto de entrenamiento cuya clase sea menos representada, se puede usar la clase Data Generator de TensorFlow para añadir cambios a cada imagen, cómo:

- Incrementar el brillo.
- Disminuir brillo.
- Girar.
- Realizar zoom (in & out).
- Desplazamiento horizontal.
- Desplazamiento vertical.
- Añadir ruido aleatorio.

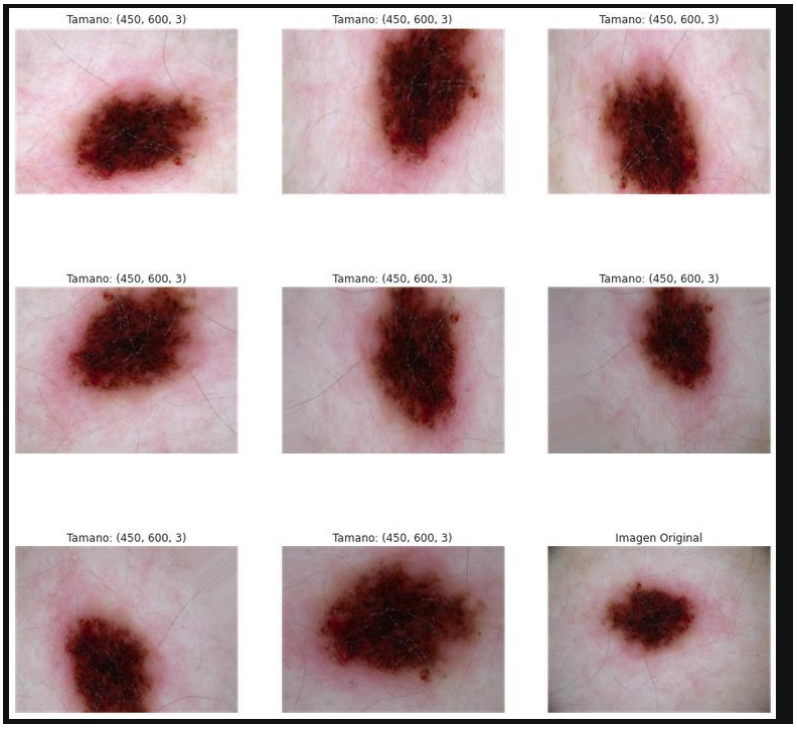


Imagen de lesión de piel después de usar aumentó de datos.

Aunque el aumento de datos es una técnica ampliamente utilizada en problemas de desbalanceo, se estimó que las clases Queratosis actínica, Carcinoma de Células Escamosas, Lesiones Vasculares y Dermatofibroma eran muy pocas y su inclusión podría afectar el desempeño del modelo, por lo tanto, no serán tenidas en cuenta en este proyecto.

```
train["clase"].value_counts()

Melanocito Nevi                  9032
Melanoma                         3192
Carcinoma de Celulas Basales     2327
Queratosis Benigna               1810
Queratosis Actinica               597
Carcinoma de Celulas Escamosas    417
Lesiones Vasculares               180
Dermatofibroma                    176
Name: clase, dtype: int64
```

Distribución de clases.

Cómo la cantidad de ejemplos máxima es el de Melanocito Nevi con 9032 imágenes, se procede a aumentar todas las otras clases hasta esta cantidad umbral cómo criterio de parada en el conjunto de entrenamiento:

```python
# recorremos la ubicacion de cada una de las imagenes con melanoma
for dir_melanoma_img in tqdm(files):
    # anadimos al directorio la ubicacion base
    #sample_img_dir = os.path.join(base_dir, dir_melanoma_img)
    sample_img_dir = os.path.join(aug_dir, dir_melanoma_img)
    # tomamos la direccion y leemos la imagen
    sample_img = load_img(sample_img_dir, target_size=(224, 224))
    # se convierte la imagen en un arreglo de numpy
    data = img_to_array(sample_img, dtype=np.float32)
    # expansion de la dimension en uno para poder ingresarlo al
    # aumentador de datos
    samples = np.expand_dims(data, 0)

    # instanciamos el aumentador
    it = train_datagen.flow(samples, batch_size=1)
    if length >= limit:
        break
```

```
# creamos un ciclo de 4 iteraciones para 4 nuevas imagenes
for i in range(5):
    # generamos una imagen
    if length >= limit:
        break
    batch = it.next()
    # generamos la imagen de la bachada
    image = batch[0].astype(np.float32)
    target_dir = os.path.split(dir_melanoma_img)[1]
    target_dir = target_dir[:-4]
    # tomamos el nombre original de la imagen y el numero de aumentado
    # para asignar un nuevo nombre a la imagen creada
    new_img_name = 'aumentado_' + target_dir + "_" + str(i + 1) + '_.jpg'
    # unimos nombre nuevo con el directorio destino
    new_img_dir = os.path.join(aug_dir, new_img_name)
    # guardamos la imagen
    save_img(new_img_dir, image)
    # anadir a lista de imagenes
    length += 1
    #augmented_images.append(new_img_dir)
```

Implementación aumento de datos en clases menos representadas.

11.1.4 Creación de modelos

Las variaciones del modelo que se probaron consisten en una única capa convolucional extractora de características que se modifica usando modelos de EfficientNet, ResNet, DenseNet y MobileNet, estas contienen capas convolucionales previamente entrenadas descargadas del Hub de TensorFlow cuyos pesos se ajustaban en el proceso de entrenamiento, precedida por una capa final de neuronas totalmente conectadas que corresponden a la cantidad de clases del modelo.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Flatten, Dense, Input
from tensorflow.keras.layers import GlobalAveragePooling2D, BatchNormalization
from tensorflow.keras.applications import EfficientNetB7


import tensorflow_hub as hub

model = tf.keras.Sequential([
   hub.KerasLayer("https://tfhub.dev/tensorflow/efficientnet/b0/feature-vector/1",
       #"https://tfhub.dev/google/imagenet/nasnet_mobile/feature_vector/4",
                  trainable=True),
   tf.keras.layers.Dense(num_neuron_output, activation="softmax"),
])
```

```
model.summary()
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_2 (KerasLayer)   (None, 1280)              4049564
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 5124
=================================================================
Total params: 4,054,688
Trainable params: 4,012,672
Non-trainable params: 42,016
```

Carga del modelo EfficientNet B0 para iniciar transferencia de aprendizaje.

12 Resultados

12.1 Creación modelo de redes neuronales convolucionales

12.1.1 Resultado Entrenamiento

Los algoritmos de clasificación de imagen con mejor desempeño en la base de datos de ImageNet corresponden a la familia de EfficientNet, sin embargo, en la fase de entrenamiento la GPU juega un papel clave para agilizar el entrenamiento, ya que, si una arquitectura es muy compleja, está podría agotar los recursos computacionales y no hará posible el entrenamiento, este problema se presentó y produjo el siguiente error:

```
Epoch 1/10
---------------------------------------------------------------------------
ResourceExhaustedError                    Traceback (most recent call last)
<ipython-input-48-c7d73e32683f> in <module>
----> 1 history = model.fit(train_gen,
      2                     steps_per_epoch = train_steps,
      3                     epochs=EPOCHS,
      4                     validation_data = valid_gen,
      5                     validation_steps = validation_steps,
```

Agotados recursos computacionales debido a limitaciones de la GPU.

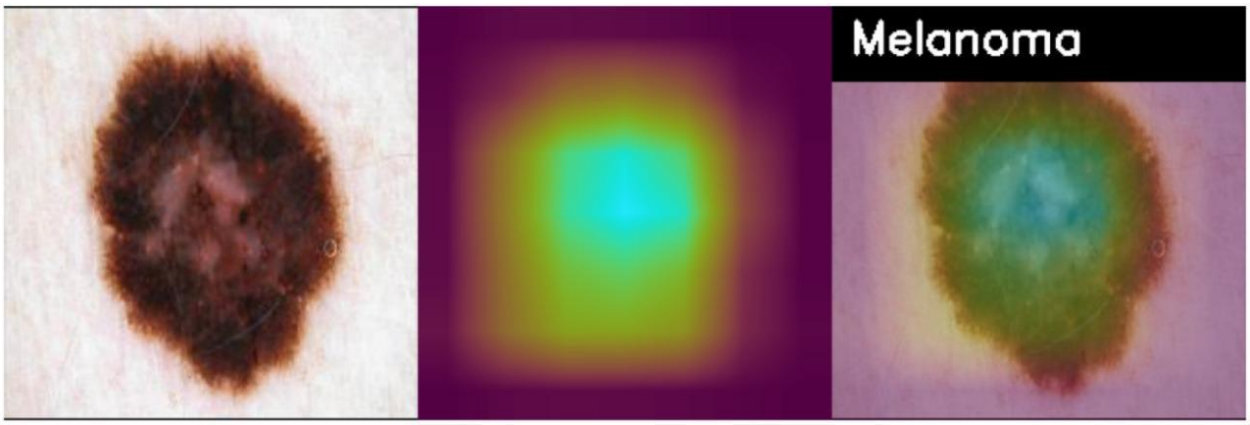
Esto condiciono al proyecto a hacer uso de versiones más básicas de EfficientNet (B5), se obtuvieron los siguientes resultados:

Nombre Modelo	Exactitud Evaluación	Tamaño
EfficientNet B5	74.04	51.8 Megabytes
MobileNetV2	69.72	30.7 Megabytes
ResNet 150	63.03	517.4 Megabytes
DenseNet	54.87	157.2 Megabytes

Destaca las metricas de EfficientNet B5 sobre todas las demás arquitecturas, MobileNetV2 le sigue aunque vemos qué su arquitectura logró un tamaño mucho menor.

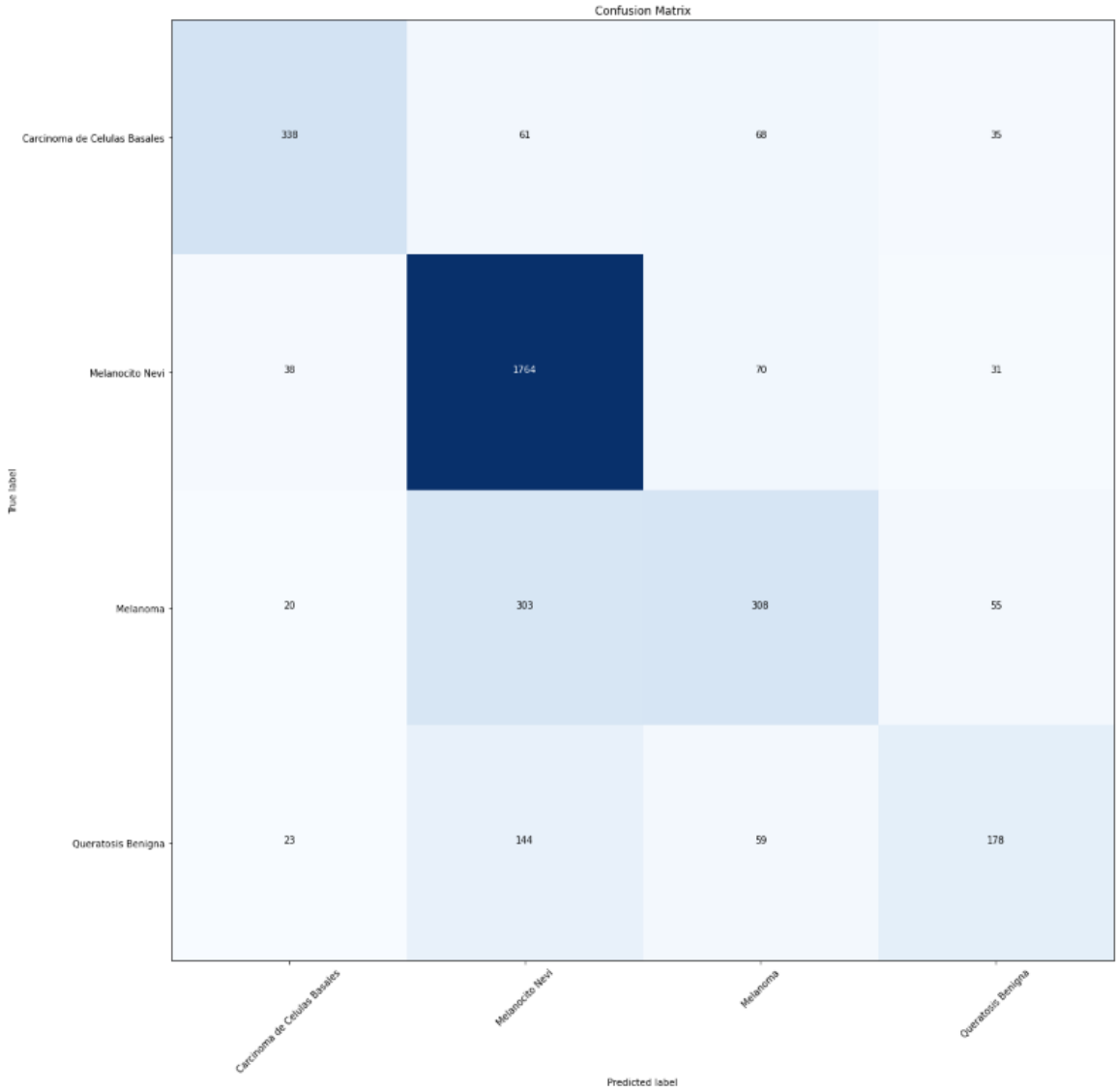
Debido a qué MobileNet V2 posee una arquitectura optimizada para realizar inferencias en sistemas embebidos, se optimizó utilizando TensorFlow Lite para producir el modelo final que tomará decisiones de manera offline, el resultado fue una reducción de su tamaño de 30.7 MB a 9.9 MB.

A continuación, se hizo uso de la técnica Grad-CAM para visualizar los píxeles más influyentes de una imagen para tomar la decisión, de esta forma, nuestro algoritmo dará una justificación de sus decisiones al usuario en lugar de comportarse como una “caja negra” qué toma decisiones, este paso es importante para una adopción final del sistema:



Aplicación de Grad-CAM para explorar píxeles más influyentes en la toma de decisión.

Por último se genera el modelo de aprendizaje en conjunto promedio haciendo uso de todos los modelos anteriormente obtenidos a excepción de DenseNet, ya que su inclusión degrado el desempeño final, obtenemos 76.1% de exactitud.



Matriz de confusión de modelo final usando promedio de conjunto.

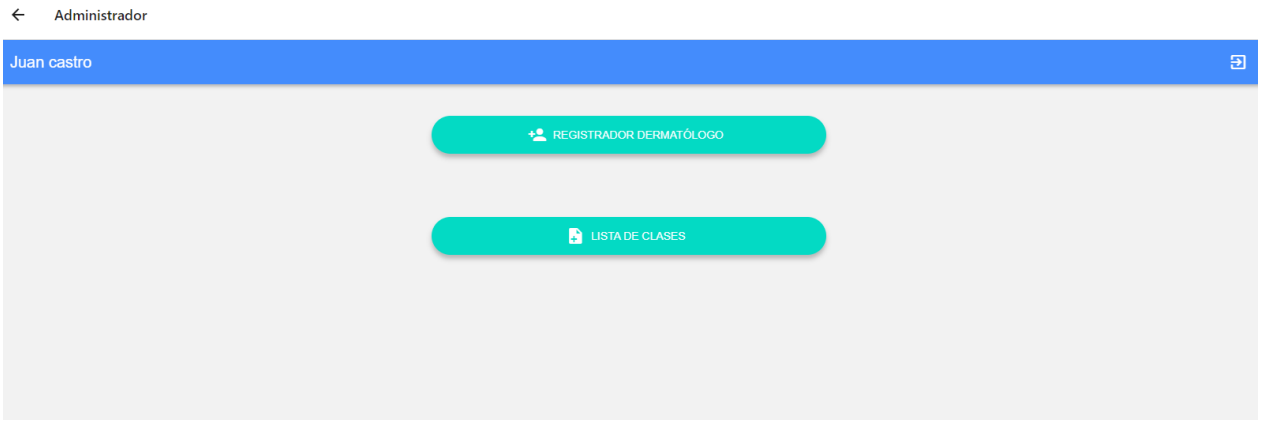
En la matriz de confusión se puede ver como la mayoría de las clases se encuentran correctamente asignadas a su correcta etiqueta, sin embargo, existe un sesgo del modelo a asignar la etiqueta de Melanocito Nevi a algunas clases, esto se puede reducir al añadir una mayor variabilidad a de muestras de las otras clases en lugar de realizar aumentado de datos.

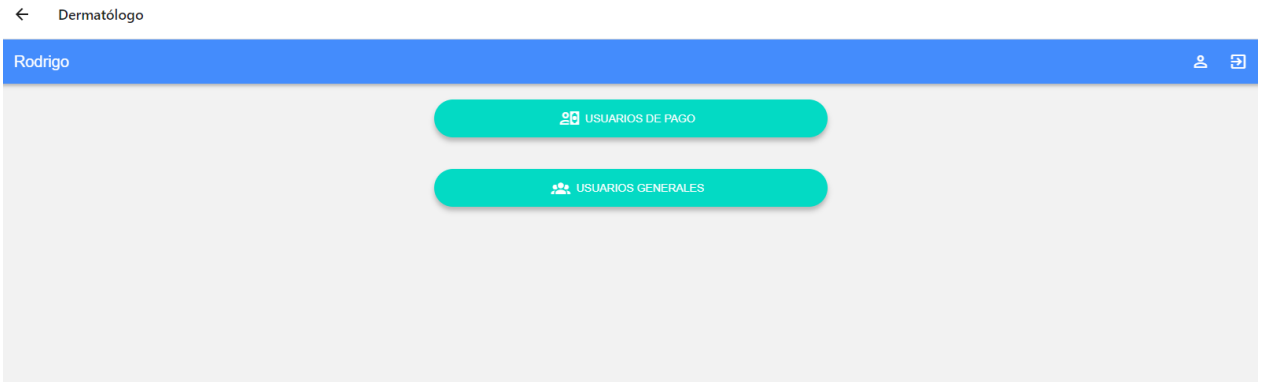
12.2 Ingeniería de Software

Se presentan los resultados que se lograron del desarrollo de la aplicación e integración con el servicio web.

12.2.1 Aplicación Web

Funciona exclusivamente para el perfil de Administrador y Dermatólogo



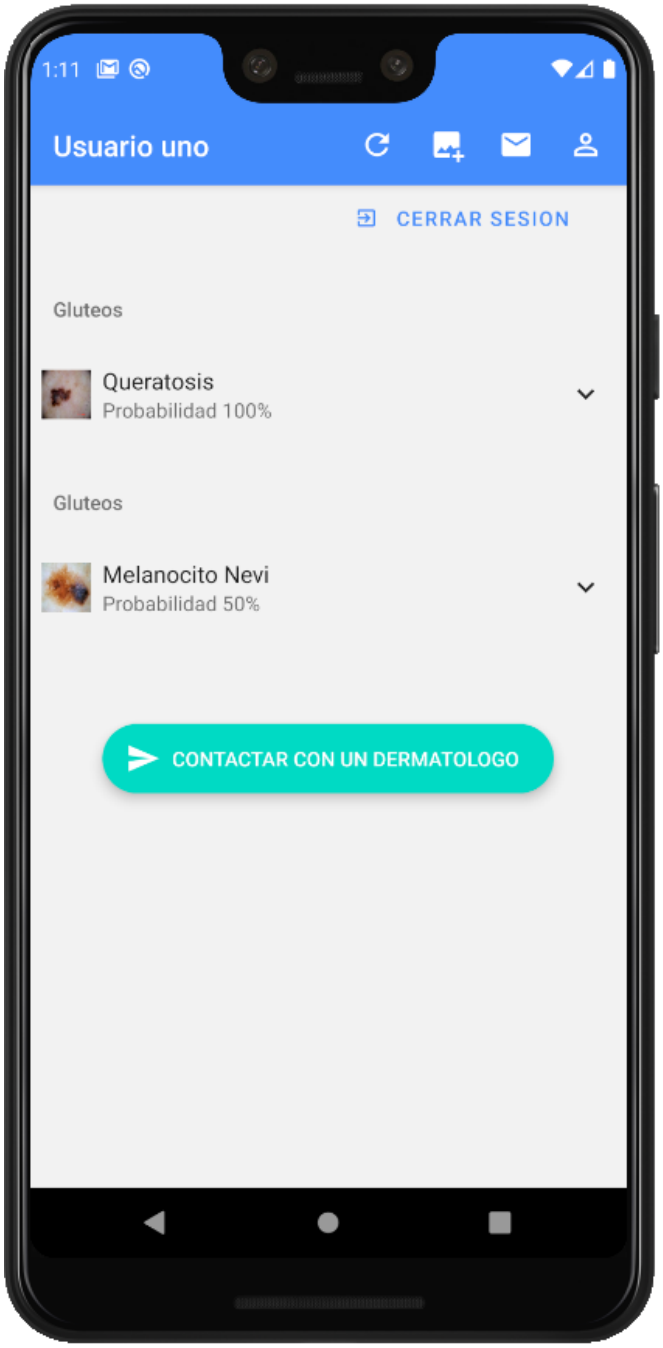


Nota: Todas las capturas de la aplicación web se encuentran en los Anexos

12.2.2 Aplicación Móvil

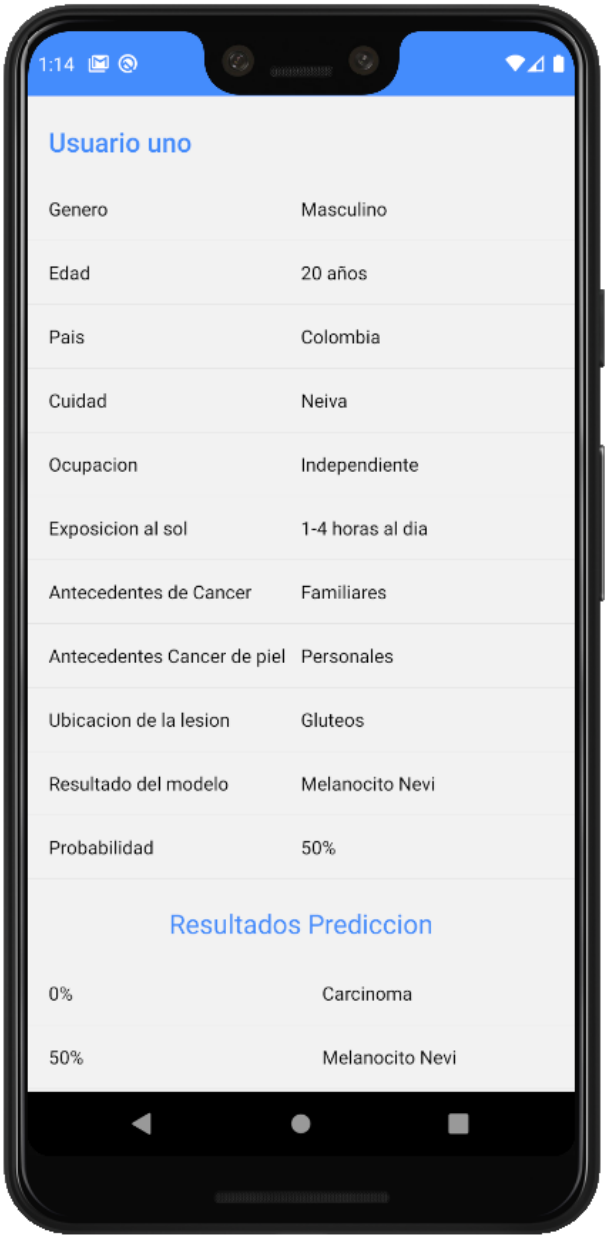
A continuación, algunas capturas de la aplicación.

Pantalla Principal Usuario – Historia de Usuario HU-04



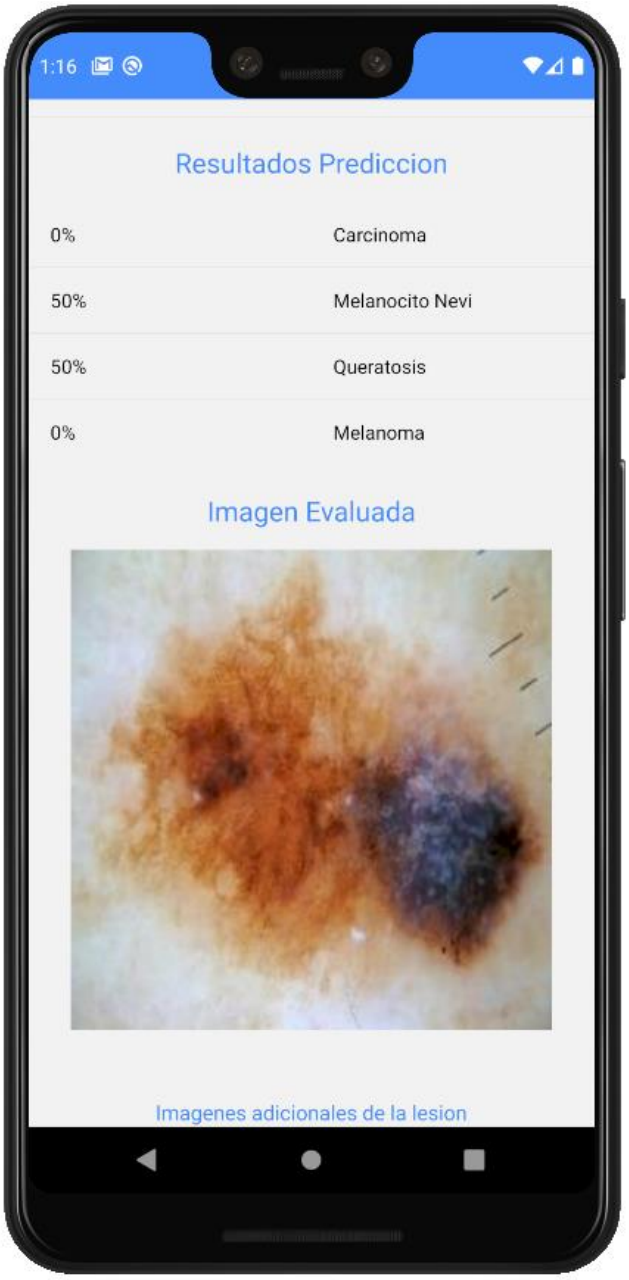
Detalle del Resultado – Historia de Usuario HU-08

1. Información del usuario

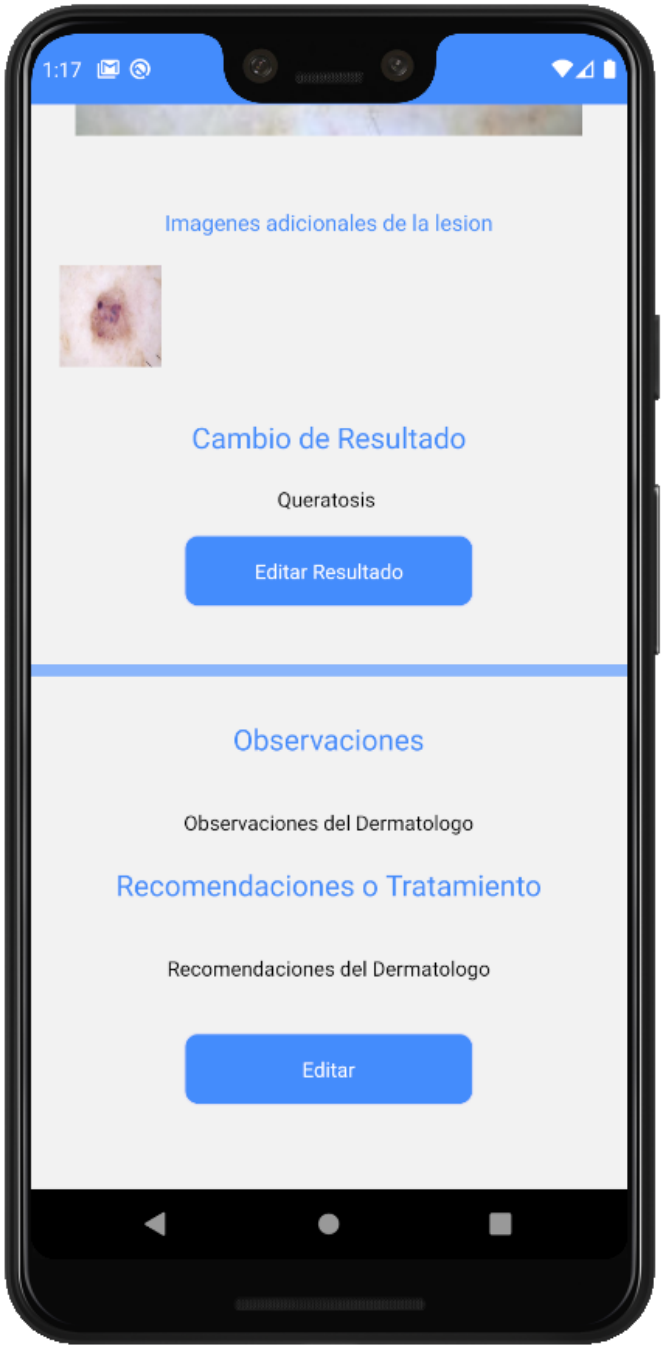


Esta información es tomada al completar el registro del usuario, y sus campos están basados en la entrevista que se realizó con un Dermatólogo.

2. Resultados Predicción e Imagen evaluada por el modelo



3. Imagen adicional y Cambios del dermatólogo

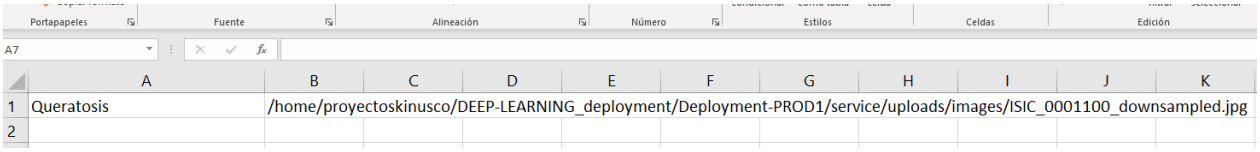


El cambio de resultado hecho por el dermatólogo nos ayuda a recolectar nueva información para la mejora continua del modelo, a través de una consulta a la Firebase podemos filtrar qué

resultados obtuvieron un cambio de clase y generar así un archivo .csv con el nuevo resultado y la respectiva imagen evaluada.

```
#Conexion a Firebase
fb_app = firebase.FirebaseApplication('https://skinusco-4b57a-default-rtdb.firebaseio.com/', None)
ref = db.reference('Usuarios')

#Abrir un archivo .csv
archivo = open("archivo.csv", "w")

#Obtener los datos de Firebase
snapshot = ref.get()

#Ruta de las imagenes
directorio = '/home/proyectoskinusco/DEEP-LEARNING_deployment/Deployment-PROD1/service/uploads/images/'

#Filtrar informacion a los campos que necesitamos resultadoDermatologo(Cambio de resultado) y nameimage (Nombre de la imagen)

for key1 in snapshot:
    #Especificamos la consulta
    result = fb_app.get('/Usuarios/'+key1+'/Resultados/', None)
    if (result == None):
        print('')
    else:
       for dd in result:
           #Nos ubicamos en la informacion de cada resultado
           result1 = fb_app.get('/Usuarios/'+key1+'/Resultados/'+dd, None)
           resultado = result1['resultadoDermatologo']
           xx = len(resultado)
           #Evaluamos cuales resultados han sido cambiados
           if (xx == 0):
               print('')
           else:
               print(resultado)
               imagen = result1['nameimage']
               #Escribimos en el archivo .csv
               archivo.write(resultado)
               archivo.write(";")
               archivo.write(directorio+imagen)
               archivo.write('\n')

#Cerramos
archivo.close()
```

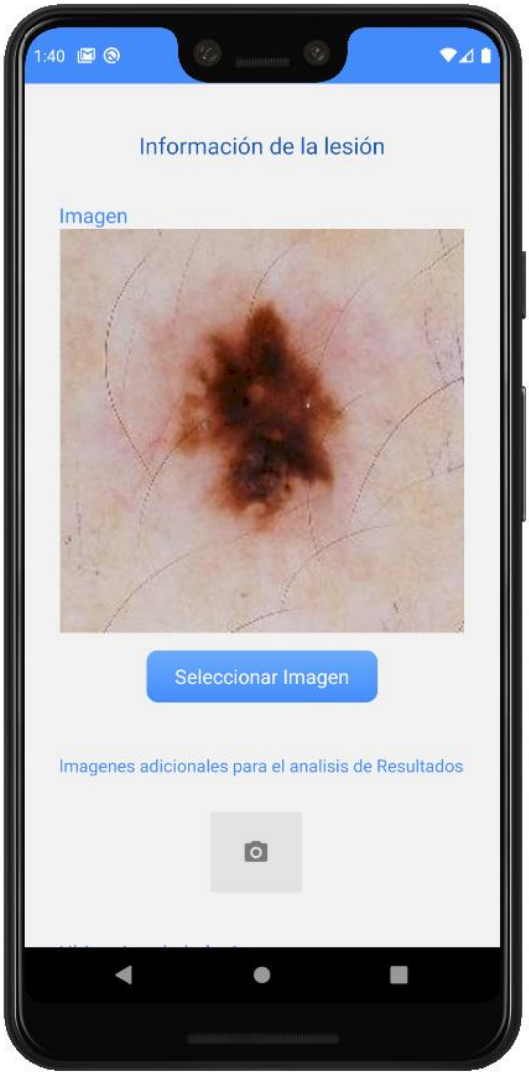
Archivo .csv



12.2.2.2 Flujo de la aplicación para obtener la clasificación de una imagen realizando la petición desde la aplicación móvil.

Luego de seleccionar la imagen, desde la galería del dispositivo. La visualizamos de la siguiente manera.

Agregar un nuevo Resultado – Historia de Usuario HU-06



En el mismo momento, el modelo recibe la imagen y nos devuelve la siguiente información desde el servicio web.

```
Filename received: ISIC_0000518.jpg

Filename stored: uploads/images/ISIC_0000518.jpg

Image: uploads/images/ISIC_0000518.jpg
Model: skinusco
Model version: 1
Port: 9501
URI: http://127.0.0.1:9501/v1/models/skinusco:predict

predictions: [3.61562451e-12, 1.0, 2.65188067e-08, 5.7479177e-09]
Clasificacion ['0%', '100%', '0%', '0%']
Index: 1
Resultados
Pred: Melanocito Nevi
Prob: 100%
{'index': 1, 'resultado': {'label': 'Melanocito Nevi', 'score': '100%', 'clasificar': ['0%', '100%', '0%', '0%']}, 'prediccionclases'
{'success': True, 'predictions': [{'label': 'Melanocito Nevi', 'score': '100%', 'clasificar': ['0%', '100%', '0%', '0%']}]}
INFO:     186.84.21.33:19892 - "POST /model/predict/ HTTP/1.1" 200 OK
```

Y en la aplicación móvil recibimos lo siguiente.

```
LOG  Resultado de la clasificacion
LOG  {"clasificar": ["0%", "100%", "0%", "0%"], "label": "Melanocito Nevi", "score": "100%"}
```

12.2.2.3 PDF informativos generados por la aplicación

Cuando el usuario desee tener un resumen de los resultados obtenidos, recibirá el PDF a su correo electrónico, el PDF contiene lo siguiente.

Nombre: Usuario uno Genero: Masculino Edad: 20 años

Fecha: 2021-10-20 03:55

Resultado del Modelo

Resultado	Zona	Clasificacion	Probabilidad	Imagen
1	Gluteos	Queratosis	100%	

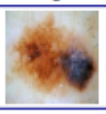
proyectoskinusco@gmail.com Universidad Surcolombiana

Cuando el dermatólogo finalice la revisión del resultado, notificará al usuario de la revisión a través de un PDF que llegará al correo electrónico del usuario.

Nombre: Maria Ximena Rodriguez Genero: Femenino Edad: 25 años

Fecha: 2021-10-20 03:31

Resultado del Modelo

Zona	Clasificacion	Probabilidad	Imagen
Gluteos	Melanocito Nevi	50%	

Valoracion del Dermatologo :

Resultado : (Valoracion Actualizada)

Observaciones : no hay observaciones

Recomendaciones : no hay recomendaciones

Resultado de la clasificacion :

Carcinoma 0%

Melanocito Nevi 50%

Queratosis 50%

Melanoma 0%

12.3 Pruebas

12.3.1 Funcionales

En los anexos podemos encontrar todo el flujo de la aplicación que cumple con el funcionamiento requerido en las historias de usuario.

12.3.2 Unitarias

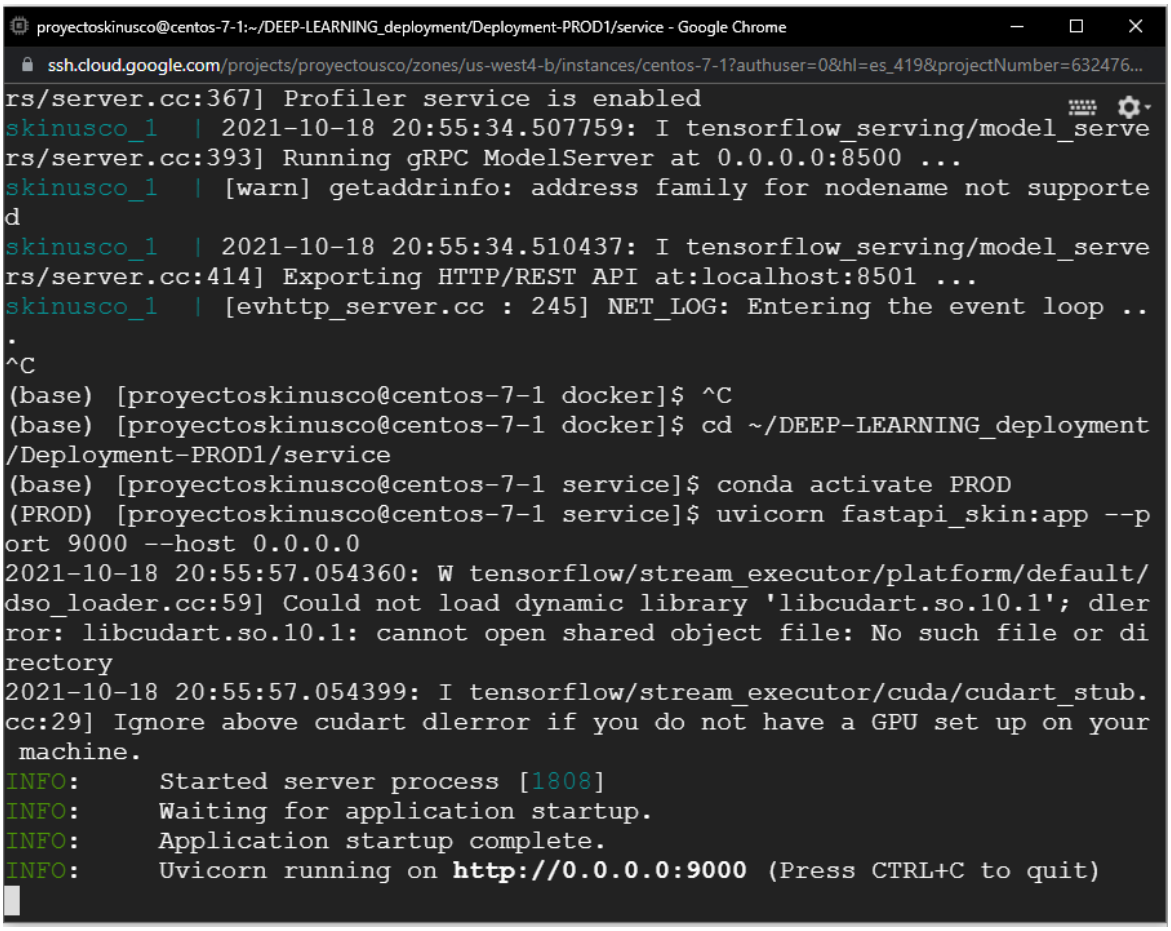
Se hacen pruebas unitarias al api, y a algunos componentes de la aplicación móvil, con la herramienta Testing Library.

```
_tests_ > JS App-test.js > describe("<UsuarioResultados/>") callback
import 'react-native';
import React from 'react';
import UsuarioResultados from '../vistas/UsuarioResultados';


// Note: test renderer must be required after react-native.
import {render,fireEvent} from '@testing-library/react-native';

let component;

describe("<UsuarioResultados/>", () => {
  beforeEach(() =>{
    //Se llamada a la funcion de Javascript fetch, que luego de un llamado se va a esperar un resultado
    global.fetch= jest.fn(()=> Promise.resolve({
      json:() => Promise.resolve([{
        "clasificar": [1.32161114e-13, 2.22921973e-10, 1.40622824e-9, 1], "label": "Melanoma", "score": 1 // ejemplo de lo que retorna el modelo por cada imagen
      }]),
    })); // definir fetch para realizar la llamada a la API
    component = render(<UsuarioResultados/>);
  });

//Caso 1 Componente UsuarioDatos (Elementos del formulario)
  it("funciona correctamente",() => {
    expect(component).toBeDefined();// Se comprueba que el componente UsuarioResultados este definido
    expect(component.getByTestId("image-app")).toBeDefined();// Se comprueba la existencia del elemento Image que esperar mostrar la imagen del usuario luego de ser seleccionada
    expect(component.getByTestId("image-add")).toBeDefined();// Se comprueba la existencia del elemento Avatar que esperar mostrar las imagenes adiccionales
    expect(component.queryAllByTestId("result")).toBeDefined();// Se comprueba que el estado que guardara el arreglo que retorna el modelo con la prediccion exista y se encuentre vacio


  });

});
```

12.4 Despliegue

Instancia luego de ser creada y configurada con el sistema operativo Centos

Servicio Web Iniciado



Nota: La configuración e implementación de las diferentes tecnologías que se utilizaron para la instalación y uso del servicio web, firebase, la aplicación web y móvil, se encuentran en Anexos.

13 Conclusiones y Recomendaciones

13.1 Conclusiones

- Las redes neuronales convolucionales son, a la fecha, el estado del arte para realizar clasificación de imágenes. Por convención, las redes más profundas podrán realizar operaciones más complejas y extraer características con mayor relevancia qué ayudarán a un algoritmo de aprendizaje automático a tomar una mejor decisión. El conjunto de datos ImageNet es la referencia para comparar los diferentes tipos de redes convoluciones, en este proyecto se encontró como usar versiones más recientes de Efficient Net obtenían un mejor desempeño.
- El fuerte desbalance presente en la base de datos sesgo el modelo al seleccionar la clase “Melanocito Nevi” más que a otras, esto afecta fuertemente la implementación del modelo en una aplicación médica real.
- Se pueden optimizar modelos para la toma de decisión en sistemas embebidos, sin embargo, la disminución del rendimiento es un precio alto qué se debe pagar, ya qué en el campo médico esto significa poner la vida de los pacientes en riesgo.
- La técnica Grad-CAM es de suma importancia ya qué explica al usuario las razones de una toma de decisión y esto fortalece la confianza en el sistema.
- Los métodos en conjunto generan una mejora en el modelo y una disminución en la varianza en sus predicciones.
- Poder tener un acercamiento con un profesional del área a trabajar, es fundamental para iniciar con una perspectiva más clara y quizás innovadora al conocer y entender las necesidad y contexto del área. Ya que con una investigación de la parte médica y con base en datos estadísticos no hubiera sido suficiente.

- Para trabajar con un proyecto de software y con un modelo de inteligencia artificial, al principio puede ser complejo, pero existen las herramientas y metodologías que permiten que esta integración cumpla con los requerimientos establecidos.
- Tener el servicio web en la nube, permite que el modelo esté disponible para implementarse en cualquier proyecto a través de una API.
- El proyecto en el transcurso de su duración tuvo grandes cambios, desde sus requerimientos, su implementación, su desarrollo, que hicieron que el tiempo se extendiera más de lo necesario. No obstante, los cambios dieron paso a extender nuestro alcance y emplear mejores herramientas en su desarrollo.
- Aunque el proyecto termina con muy buenos resultados, cabe destacar que trabajando con un modelo de negocio bien establecido y una planificación ajustada a los requerimientos del proyecto, se podía obtener resultados en menor tiempo y con un mayor impacto comercial.

13.2 Recomendaciones

- Existe gran investigación en visión por computadora para clasificación de imágenes, la selección del modelo de red neuronal convolucional debe ser constante y se debe actualizar conforme varía el estado del arte.
- Existe un sesgo hacia la clase Melanocito Nevi, este se puede disminuir al incluir mayor variabilidad de muestras de las clases menos representadas, en lugar de usar aumentó de datos para incrementar el número de ejemplos.

- Gran cantidad de las imágenes corresponden a piel y en un menor porcentaje a la lesión de piel, debido a que la señal que genera las predicciones se encuentra en la lesión como tal, un zoom hacia las lesiones debe mejorar el rendimiento del sistema. Esto se descartó debido a que este proceso se debía realizar de forma manual en más de treinta mil imágenes, pero es un aspecto prometedor para mejorar el sistema.
- Por cuestión de tiempo sólo se exploraron los métodos en conjunto dónde se considera el promedio de las predicciones, sin embargo, en la fase teórica del proyecto se describieron otros métodos interesantes qué vale la pena investigar.
- El proyecto queda en proceso de mejora y ajuste, para enfocarlo a una idea de negocio clara con más servicios.

14 Recursos, Costos y Fuentes de Financiación

14.1 Recursos

Para la elaboración de este proyecto, los recursos humanos y los recursos materiales. como personal de trabajo, computadoras e instalaciones, son de carácter PROPIO caracterizados por ser de libre utilización y disponibilidad.

14.1.1 Recurso Hardware

Componente	Referencia	Marca	Valor enero 2021
Procesador-CPU	i7 9700K (3,6-12M-8 CORE) - IX	Intel Core	$ 1.640.000
Refrigeración para Procesador	COOLER MÁSTER LITE 120	Cool Máster	$ 250.000
Tarjeta Gráfica - GPU	8 GB GEFORCE-GTX 2080 SUPER.OC. DUAL	Asus	$ 3.020.000
Memoria - RAM	KIT DDR4 16 GB (8x2)	Corsair Lpx	$ 550.000
Motherboard	TUF Z390 PLUS GAMING.WiFi.HD.DP.CROSS.A.V.R.4DDR4.IX	Asus	$ 735.000
Fuente de Poder	Fuente Real 750W 80 PLUS GOLD	Gigabyte	$ 459.000
Disco Duro - HDD	1 TB Sata III (64 m-7200)	Seagate	$ 175.000
Unidad de estado sólido - SSD	Solido Sata (SSD) 240 GB A400	Kingston	$ 185.000

De Hardware, tenemos un costo total de $7.564.000 pesos colombianos, correspondientes a 8 componentes, que cumplirán principalmente funciones de preprocesamiento y entrenamiento del Modelo de inteligencia artificial, siendo este recurso útil en la duración total del proyecto.

14.1.2 Recurso Humano

El proyecto se desarrolló con la intervención de 2 personas que se encargan de las dos grandes secciones del proyecto, Construcción y análisis de un Modelo de Inteligencia Artificial e Implementación de la Ingeniería de Software al proyecto.

14.2 Costos

Los costos de los recursos empleados en el desarrollo del proyecto son evaluados en relación con el tiempo que se requieren, el proyecto tiene una duración de 12 meses.

Cargo	Salario Mensual	Tiempo (Meses)	Salario total
Ingeniero de Software	$3,000,000.00	12	$36,000,000
Ingeniero Machine Learning	$4,000,000.00	12	$48,000,000

14.3 Fuentes de Financiación

El tipo de fuente de financiación empleada en este proyecto es financiación interna o también llamada propia, donde los integrantes del proyecto aportan sus propios recursos en cualquier momento para cubrir con costos y necesidades.

15 Bibliografía

Krizhevsky, A., Sutskever, I. y Hinton, GE (2012). Clasificación de ImageNet con redes neuronales convolucionales profundas. En F. Pereira, CJC Burges, L. Bottou y KQ Weinberger (Reds), Avances en sistemas de procesamiento de información neuronal (Vol. 25). Furgoneta de Opgehaal https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Tan, M. y Le, QV (2020). EfficientNet: Repensar el escalado de modelos para redes neuronales convolucionales. arXiv [cs.LG] . Furgoneta de Opgehaal http://arxiv.org/abs/1905.11946

Manzo, M. y Pellino, S. (2020). Conjunto de funciones de aprendizaje de transferencia profunda y modelos de clasificación para la detección del melanoma. arXiv [eess.IV] . Furgoneta de Opgehaal http://arxiv.org/abs/2009.08639

Ha, Q., Liu, B. y Liu, F. (2020). Identificación de imágenes de melanoma usando EfficientNet Ensemble: solución ganadora para el desafío de clasificación de melanoma SIIM-ISIC. arXiv [cs.CV] . Furgoneta de Opgehaal http://arxiv.org/abs/2010.05351

Le, DNT, Le, HX, Ngo, LT y Ngo, HT (2020). Transferir aprendizaje con función de pérdida focal y ponderada por clase para la clasificación automática del cáncer de piel. arXiv [cs.AI] . Furgoneta de Opgehaal http://arxiv.org/abs/2009.05977

ABAMobile. (2021, 18 mayo). Apps multiplataforma. Qué son y características. Recuperado 21 de junio de 2021, de https://abamobile.com/web/apps-multiplataforma-que-son-y-caracteristicas/

Arteaga, S. (2015, 24 agosto). SkinVision, una app que detecta el cáncer de piel con el móvil. Recuperado 13 de septiembre de 2020, de https://computerhoy.com/noticias/apps/skinvision-app-que-detecta-cancer-piel-movil-33259

BBC News Mundo. (2017, 14 febrero). Qué es el carcinoma basocelular, el cáncer de piel que sufre el actor Hugh Jackman. Recuperado 15 de septiembre de 2020, de https://www.bbc.com/mundo/noticias-38976788

Benedetti, J. (2021, 9 septiembre). Descripción de las lesiones cutáneas. Recuperado 3 de octubre de 2021, de https://www.msdmanuals.com/es-co/professional/trastornos-dermatol%C3%B3gicos/abordaje-del-paciente-dermatol%C3%B3gico/descripci%C3%B3n-de-las-lesiones-cut%C3%A1neas#:%7E:text=Las%20erosiones%20son%20%C3%A1reas%20abiertas,por%20ara%C3%B1azos%2C%20fricciones%20o%20rascado.&text=y%20otras%20rickettsiosis).

Cevallos, K. (2015, 26 julio). UML: Casos de Uso. Recuperado 20 de septiembre de 2020, de https://ingsotfwarekarlacevallos.wordpress.com/2015/06/04/uml-casos-de-uso/

Cillero, M. (2016, 12 octubre). Diagrama de Despliegue. Recuperado 21 de septiembre de 2020, de https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/diagrama-de-despliegue/

de OpenClassrooms, M. (2017, 11 septiembre). ¿Qué es el desarrollo web? Recuperado 23 de septiembre de 2020, de https://blog.openclassrooms.com/es/2017/09/11/que-es-el-desarrollo-web/

Equipo editorial, Etecé. (2021, 6 agosto). Lenguaje de Programación - Concepto, tipos y ejemplos. Recuperado 12 de septiembre de 2021, de https://concepto.de/lenguaje-de-programacion/

Euroforum. (2019, 4 junio). Metodología Scrum: definición, herramientas y ejemplos de proyectos. Recuperado 20 de septiembre de 2020, de

https://www.euroforum.es/blog/metodologia-scrum-definicion-herramientas-y-ejemplos-de-proyectos/

Feda, E. D. N. (2019, 20 mayo). GESTIÓN ÁGIL vs GESTIÓN TRADICIONAL DE PROYECTOS ¿CÓMO ELEGIR? - Escuela de Negocios FEDA. Recuperado 17 de septiembre de 2020, de https://www.escueladenegociosfeda.com/blog/50-la-huella-de-nuestros-docentes/471-gestion-agil-vs-gestion-tradicional-de-proyectos-como-elegir

Lee, G. (2021, 17 marzo). Tipos de pruebas de software: diferencias y ejemplos. Recuperado 22 de julio de 2021, de https://www.loadview-testing.com/es/blog/tipos-de-pruebas-de-software-diferencias-y-ejemplos/

Melanoma - Estadísticas. (2021, 6 abril). Recuperado 14 de septiembre de 2020, de https://www.cancer.net/es/tipos-de-c%C3%A1ncer/melanoma/estad%C3%ADsticas

MoleScopeTM. (2015, 8 junio). Recuperado 13 de septiembre de 2020, de https://apps.apple.com/us/app/molescope/id1003576096

Novoseltseva, E. (2021, 9 junio). 5 principales patrones de Arquitectura de Software. Recuperado 10 de julio de 2021, de https://apiumhub.com/es/tech-blog-barcelona/principales-patrones-arquitectura-software/

Pérez, M. S. L. (2015, 6 mayo). Lesiones traumáticas en la mucosa oral de los adultos mayores. Recuperado 12 de noviembre de 2020, de https://scielo.isciii.es/scielo.php?script=sci_arttext&pid=S0213-12852015000300003#:%7E:text=La%20fricci%C3%B3n%20o%20acci%C3%B3n%20mec%C3%A1nica,color%20blanco%20(queratosis%20friccionales).

Puente, J. (2019, 16 diciembre). ¿Qué es el cáncer y cómo se desarrolla? - SEOM: Sociedad Española de Oncología Médica © 2019. Recuperado 20 de noviembre de 2020, de https://seom.org/informacion-sobre-el-cancer/que-es-el-cancer-y-como-se-desarrolla

Quirós, D. S. (2021, 27 agosto). Qué es UML: Unified Modeling Language. Recuperado 1 de septiembre de 2021, de https://openwebinars.net/blog/que-es-uml-unified-modeling-language/

Radiation: Ultraviolet (UV) radiation and skin cancer. (2017, 16 octubre). Recuperado 14 de septiembre de 2020, de https://www.who.int/news-room/q-a-detail/radiation-ultraviolet-(uv)-radiation-and-skin-cancer

Rendón, Y. A. (2019, 28 mayo). Bases de datos relacionales vs. no relacionales. Recuperado 25 de septiembre de 2020, de https://www.pragma.com.co/academia/lecciones/bases-de-datos-relacionales-vs.-no-relacionales

Rodríguez, M. (2019). Nevus melanocitico atipico (lunares). Recuperado 6 de octubre de 2020, de https://asocolderma.org.co/enfermedades-de-la-piel/nevus-melanocitico-atipico-lunares

Rodriguez, M. J. (2020, 12 enero). Despliegue de modelos tensorflow. Recuperado 25 de noviembre de 2020, de https://www.udemy.com/join/login-popup/?next=/course/deep-learning-despliegue-tensorflow-mirko-rodriguez/learn/lecture/22257870#overview

Sánchez, J. O. (2021, 6 abril). Descripción de la respuesta clínica de los pacientes con melanoma maligno localmente avanzado o metastásico con mutaciones del gen BRAFV600E que reciben Vemurafenib en el Instituto Nacional de Cancerología. Recuperado 14 de septiembre de 2020, de https://repositorio.unbosque.edu.co/handle/20.500.12495/5710

Sánchez-Monge, M. (2021, 29 abril). Melanoma. Recuperado 17 de junio de 2021, de https://cuidateplus.marca.com/enfermedades/cancer/melanoma.html

Santander Universidades. (2021, 31 agosto). Metodologías de desarrollo software | Blog. Recuperado 2 de septiembre de 2021, de https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html

Siripathi, S. (2017, 24 julio). Lenguajes de Desarrollo Para Móvil. Recuperado 25 de septiembre de 2020, de https://code.tutsplus.com/es/articles/mobile-development-languages--cms-29138

Tomás, E. (2021, 9 julio). Molexplore: pasado, presente y futuro. Recuperado 13 de septiembre de 2020, de https://molexplore.com/es/

Unir, V. (2021, 11 octubre). Ingeniería de software: qué es, objetivos y funciones del ingeniero. Recuperado 18 de octubre de 2021, de https://colombia.unir.net/actualidad-unir/ingenieria-de-software-que-es-objetivos/

Valera, J. G. (2020, 27 noviembre). Cáncer de piel. Recuperado 5 de enero de 2021, de https://cuidateplus.marca.com/enfermedades/cancer/cancer-piel.html

Workana. (2021, 8 enero). Interfaz gráfica de usuario o GUI: Qué es y Para qué sirve | Workana. Recuperado 1 de agosto de 2021, de https://i.workana.com/glosario/que-es-la-interfaz-grafica-de-usuario-gui/

Zr, C. (2019, 20 febrero). Diagrama de estado UML, ejemplo. Recuperado 20 de septiembre de 2020, de https://carloszr.com/diagrama-de-estado-uml-ejemplo/

16 Anexos

16.1 Anexos Análisis

16.1.1 Preguntas Dermatólogo

1. ¿Qué datos adicionales tiene en cuenta para determinar el tipo de lesión?

Ocupación, exposición al sol, antecedentes de melanoma, antecedentes de otros tipos de cáncer, ubicación en el cuerpo.

2. ¿Queremos con el aplicativo almacenar las imágenes que ingresen, es antiético hacia el paciente realizarlo?

No considera que sea un problema guardar las imágenes, mientras sean con fines educativos o estadísticos.

3. ¿Ha usado alguna aplicación similar?

No ha usado ninguna aplicación.

4. ¿Es necesario el zoom óptico para el diagnóstico de lesiones de piel?

No es absolutamente necesario, pero amplifica las capacidades, se llama dermatoscopio.

6. ¿Es un problema para ustedes examinar regiones donde el paciente siente vergüenza?

No es tan común.

7. ¿Cómo impacta una aplicación así en su flujo de trabajo? ¿Lo mejoraría? ¿lo haría menos o más eficiente?

Se tienen en cuenta segundas opiniones de otros dermatólogos, podría ser importante para corroborar un diagnóstico.

El dermatólogo no puede remitir el paciente a cirugía sin antes tomar una biopsia y confirmar el diagnóstico, el estudio puede tomar más de 15 días debido a que los laboratorios están en Bogotá.

16.1.2 Historias de Usuario

HU-03

HU-03	como usuario	necesito verificar el correo ingresado, y recuperar contraseña	para poder acceder al aplicativo	1	Verificar correo	Cuando el usuario ingrese el correo y la contraseña	Click en el boton 'Terminar Registro'	El sistema debera guardar las credenciales en la base de datos, y enviar un correo de verificacion de acceso al correo registrado. El usuario no podra loguearse con ningun rol hasta no estar verificado el correo electronico que fue ingresado
				2	Recuperar Contraseña	Cuando el usuario intente iniciar sesion y no recuerde su contraseña de ingreso	Click en el boton 'Olvidaste tu contraseña'	El sistema enviara un correo electronico al usuario, con un enlace para recuperar su contraseña

HU-04

HU-04	como sistema	necesito controlar el acceso a la aplicación mediante roles	Para uso dar las funciones correspondientes a cada rol	1	Usuario Registrado	Cuando el usuario ya ha ingresado por primera vez y haya completado su registro, y desee iniciar sesion ingresando sus creedenciales	Click en 'Ingresa'	El sistema debera mostrar la vista inicial del Usuario, con la lista de resultados que ha generado, un boton de 'Contactar Dermatologo' , y una barra de navegacion en la parte superior con el nombre del usuario, y cuatro botones que corresponde a actualizar la lista, a agregar un nuevo resultado, a enviar por correo los resultados obtenidos , un boton de perfil que muestra la informacon del usuario , y por ultimo un boton de cerrar sesion
				2	Usuario No Registrado	Cuando el usuario va a ingresar por primera vez, debera llenar un formulario para completar su registro	Click en 'Ingresa'	El sistema mostrara cinco input para los datos de nombre, edad, pais, cuidad, ocupacion, deben tener su respectiva validacion. Se debe implementar tres select, para el genero, los antedcedentes de cancer, antecedentes de cancer de piel y cantidad de horas en que se expone al sol por dial. y por ultimo se mostrara un boton 'Guardar' para guardar la informacion ingresada
				3	Dermatologo	Cuando el Dermatologo desee iniciar sesion y se encuentre registrado, e ingrese sus credenciales	Click en 'Ingresa'	El sistema mostrara dos botones correspondiente a los tipos de usuarios del sistema, usuarios de pago y generales. Y una barra de navegacion con el nombre del dermatologo mas un boton de cerrar sesion
				4	Administrador	Cuando el administrador desee iniciar sesion e ingrese sus credenciales	Click ' Ingresa'	El sistema mostrara una vista con dos botones de 'Regitrar Dermatologo' y 'Lista de clases', tambien debera ir una barra de navegacion en la parte superior con el nombre del administrador y un boton de cerrar sesion

HU-05

HU-05	como usuario	necesito ver los resultados que he generado	para conocer su clasificacion y ternerlos disponibles	1	Lista de resultados	Cuando el usuario haya generado resultados	Click en el icono 'Actualizar'	El sistema mostrara en la pantalla del usuario, una lista con los resultados generados, mostrando la zona en que se encuentra la lesion, la clasificacion obtenida, el porcentaje de precision y la imagen de cada uno de los resultados, tambien debe tener la opcion de eliminar cada resultado
				2	Correo	Cuando el usuario tenga uno o mas resultados generados	Click en el icono 'Correo'	El sistema mostrara un input para ingresar el correo a donde se desea que sean enviados los resultados que se han obtenido hasta el momento, y por ultimo un boton de 'Enviar resultados'

HU-06

HU-06	como usuario	necesito realizar una clasificacion al modelo	para obtener un resultado de la lesion	1	Informacion de la lesion	Cuando el usuario desee agregar una nueva clasificacion	Click en el icono 'nuevo resultado '	El sistema mostrara una pantalla con un boton para seleccionar la imagen que se va a clasificar, otro boton para agregar imágenes adiccionales de la lesion que complementen el analisis, la imagenes seran tomadas desde la galeria del dispositivo, se debe tener un select con las diferentes ubicaciones en la que puede estar la lesion, y por ultimo un boton 'Enviar informacion'
				2	Imágenes	Cuando el usuario ingrese la imagen a clasificar y las imágenes adicionales	Click en el boton 'Enviar informacion'	El sistema debe guardar el nuevo resultado con las imágenes y los datos ingresados en la base de datos, tambien enviara al modelo la imagen a clasificar para obtener una clasificacion que se guardara junto al resto de informacion del resultado. El nuevo resultado debera mostrarse en la lista de los resultados generados por el usuario

HU-07

HU-07	como Dermatologo	necesito saber que usuarios son de pago y cuales no	para observar sus resultados y dar mi diagnostico	1	Usuario de pago	Cuando el usuario haya solicitado los servicios con el dermatologo	Click 'Contactar un Dermatologo'	El sistema mostrara un boton 'Usuarios de pago', al ingresar se mostrara la lista de usuarios de pago, cada usuario tendra como opciones 'ver los resultados' o cambiar el estado del usuario a 'Revisado'.
				2	Usuario Generales	Cuando el usuario no haya solicitado los servicios con el dermatologo		El sistema mostrara un boton 'Usuarios de generales', al ingresar se mostrara la lista de usuarios de generales, cada usuario tendra como opciones 'ver los resultados' o cambiar el estado del usuario a 'Revisado'.

HU-08

HU-08	como Dermatologo	necesito visualizar, controlar, y notificar sobre los resultados de los usuarios	para adiccionar las observaciones o modificaciones de los resultados	1	Pendientes y revisados	Cuando el dermatologo seleccione a un usuario	Click en 'Ver resultados'	El sistema mostrara dos secciones 'Pendientes' y 'Revisado', mostrando el estado de los resultados según las opciones respectivas. Los resultados pendientes deben tener la opcion de 'Visualizar informacion' y 'Revision Completada' (Que cambia el estado del resultado a la seccion revisados). Y los resultados revisados tendran la opciones de 'Marcar como pendiente' (Que cambia el estado del resultado a la seccion pendientes)y 'Enviar
				2	Detalles de los resultados	Cuando el dermatologo desee ver los detalles de cada resultado	Click en 'Visualizar informacion'	El sistema mostrara una pantalla con lo sigueitne, el nombre del usuario, luego una tabla con la siguiente informacion: genero, edad, pais, cuidad, ocupacion, exposicion al sol, antencedentes de cancer, antecedentes de cancer de pioel, ubicación de la lesio, resultado del modelo y la probabilidad. Luego mostrara una tabla con los resultados de la prediccion y su respectiva clase. Luego se mostrara la imagen que ha sido evaluada y la opcion de hacerle zoom, luego mostrara las imagenes adicionales que se adjuntaron al resultado, con la posibilidad de visualizarla y hacerle zoom.
				3	Cambios por el dermatologo	Cuando el dermatolgo ya haya visualizado la informacion del resultado y desee hacer cambios	Click 'Editar'	El sistema mostrara un boton para editar el resultado, donde debe mostrar el cambio del resultado o en su defecto un select para hacer el cambio, y un boton de actualizar cuando ya se tenga el cambio. Luego tenemos un boton de editar para poder agregar o editar observaciones y recomendaciones o tratamientos que quiera hacer el dermatologo, con un boton de actualizar para finalizar los cambios. Los datos ingresados por el dermatologo se deben almacenar en el restultado que se esta revisando, para la mejora conitnua del modelo
				4	Notificacion por correo	Cuando el Dermatologo finalice la revision de un resultado, tendra la opcion de notificar los cambios al usuario mediante correo electronico	Click 'Enviar Notificacion por correo'	El sistema tomara el correo que esta asociado al usuario, y enviara un pdf con el resultado que ya ha sido revisado, y los cambios o comentarios que ha agregado el Dermatologo

HU-09

HU-09	como Administrador	Necesito registrar dermatologos y editar la lista de clases	Para que la aplicación se adapte a lo que se necesite	1	Registro Dermatologo	Cuando el Administrador necesite agregar un nuevo Dermatologo	Click 'Registrar Dermatologo'	El sistema mostrara cinco input para los siguientes datos: correo electronico, nombre , pais, cuidad y entidad de salud. Por ultimo mostrara un select para el genero, y un boton de 'Terminar Registro'. Los datos seran guardados en la base de datos
				2	Lista de clases	Cuando el administrado necesite editar la lista de clases	Click 'Lista de clases'	El sistem mostrara la lista de clases actual y un boton de editar lista, cuando se de click en editar lista, el sistema mostrara un input para agregar un nuevo elemento de la lista junto a un boton de agregar, y las clases actuales con la opcion de eliminarla, por ultimo un boton de 'Guardar'

HU-10

HU-10	como Sistema	necesito responder las solicitudes del cliente	Para dar funcionamiento a los servicios	1	Envio de Correo de los resultados y de notificacion	Cuando el usuario o dermatologo utilicen la funcion de enviar correo		El sistema debera crear un pdf con la informacion que se le ha solicitado y luego debera enviar un correo electronico con el pdf adjunto al correo electronico del usuario
				2	Manejo de la lista de clases	Cuando el administrador modifique o actualice la lista de la clases		El sistema debera trabajar para la clasificacion de la imagen con la nueva lista de clases que se ha ingresado.

HU-11

HU-11	como sistema	necesito procesar la imagen	Para poder procesarla en el modelo y obtener un resultado	1	Conexión con el servidor	Cuando la imagen la tome el sistema	Click en el boton 'Enviar Informacion'	El sistema debera hacer una solicitud al servicio web por una conexión http, donde recibira la imagen
				2	Pre-Procesamiento	Cuando la imagen ingresa al servidor	Click en el boton ' Enviar Informacion'	El sistema debe procesar la imagen, asegurandose que la imagen cumpla con el tamaño requerido(224,224), y luego la enviamos al modelo para que sea clasificada
				3	Procesamiento	Cuando la imagen al modelo	Click en el boton ' Enviar Informacion'	El sistema procesa la imagen y retorna un json con las predicciones que se han obtenido
				4	Clasificacion	cuando la imagen ha sido procesada	Click en el boton ' Enviar Informacion'	El sistema tomara la mayor prediccion que se obtuvo para asignarle su respectiva clase dentro de las cuatro que trabajamos, 'Melanoma', Melanocito Nevi', 'Carcinoma', 'Queratosis'. En esta parte el sistema debera guardar la imagen localmente para la mejora continua del modelo.
				5	Resultado	cuando la imagen ha sido procesada y clasificada	Click en el boton ' Enviar Informacion'	El sistema servidor retornar un json con los resultados, que seran guardados en la base de datos, que seran mostrados al usuario en la lista de resultados generados .

16.1.3 Product Backlog

HU-01	Como usuario, necesito una pantalla de bienvenida	HU-01-Bienvenida	Terminada	8	1	1
HU-02	Como usuario, necesito hacer uso de la aplicación donde tengo la opcion de registro o login	HU-02-Registro y Login	Terminada	8	1	1
HU-03	Como usuario, necesito verificar el correo ingresado, y recuperar la contraseña	HU-03-Verificar correo y recuperar contraseña	Terminada	21	2	2
HU-04	Como sistema, necesito controlar el acceso a la aplicación mediante roles	HU-04-Roles	Terminada	55	2	4
HU-05	Como un usuario, necesito visualizar , tener y elminar los resultados que he generado	HU-05-Opciones Usuario	Terminada	55	2	4
HU-06	Como un usuario, necesito agregar resultados	HU-06- Agregar Resultado	Terminada	144	1	5
HU-07	Como un Dermatologo, necesito conocer que usuarios son de pago y cuales no	HU-07- Dermatologo	Terminada	21	2	1
HU-08	Como un Dermatologo,necesito visualizar, controlar y notificar sobre los resultados de los usuarios	HU-08-Opciones Dermatologo	Terminada	55	2	4
HU-09	Como un Administrador, necesito registrar dermatologos y editar la lista de clases que maneja el sistema	HU-09-Administrador	Terminada	55	2	4
HU-10	Como un sistema, necesito responder las solicitudes del usuario	HU-10-Sistema respuesta del servidor	Terminada	144	1	5
HU-11	Como un sistema, necesito procesar y relacionar la imagen con la prediccion	HU-11-Sistema proceso de clasificacion	Terminada	144	1	5

16.1.4 Sprint Backlog

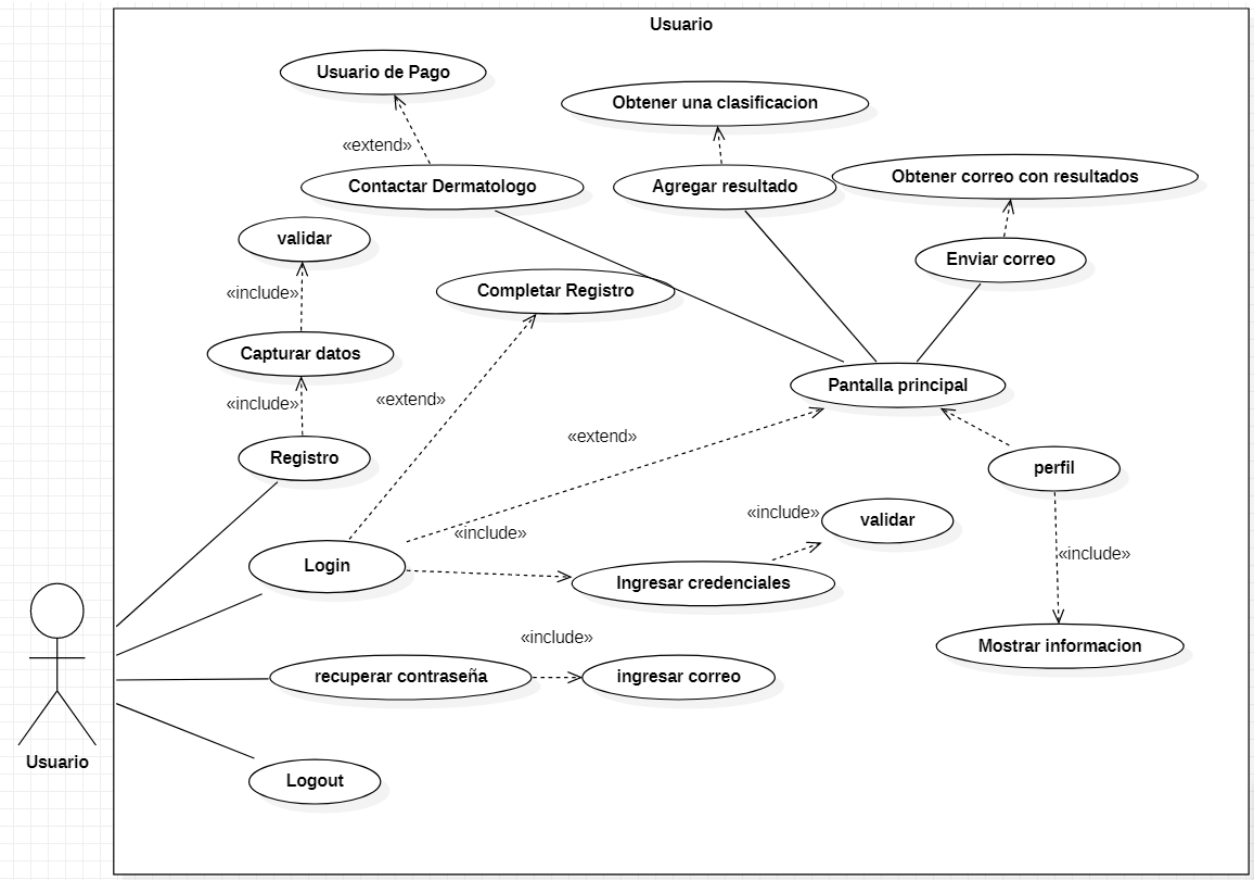
HU-04	Como sistema, necesito controlar el acceso a la aplicación mediante roles	Diseñar los mockups del Formulario	Maria Ximena Rodriguez	Terminada
		Diseñar los mockups de la pantalla principal del usuario	Maria Ximena Rodriguez	Terminada
		Diseñar los mockups de la pantalla principal del Dermatologo	Maria Ximena Rodriguez	Terminada
		Diseñar los mockups de la pantalla principal del Administrador	Maria Ximena Rodriguez	Terminada
		Codificar la pantalla del usuario y formulario	Maria Ximena Rodriguez	Terminada
		Codificar la pantalla del dermatolgo	Maria Ximena Rodriguez	Terminada
		Codificar la pantalla del administrador	Maria Ximena Rodriguez	Terminada
		Codificar la funciones y manejo de los roles	Maria Ximena Rodriguez	Terminada
HU-05	Como un usuario, necesito visualizar , tener y elminar los resultados que he generado	Diseñar los mockups para la lista de los resultados y el envio del correo	Maria Ximena Rodriguez	Terminada
		Codificar	Maria Ximena Rodriguez	Terminada
HU-06	Como un usuario, necesito agregar resultados	Diseñar los mockups de agregar un nuevo resultado	Maria Ximena Rodriguez	Terminada
		Codificar la pantalla de agregar un nuevo resultado	Maria Ximena Rodriguez	Terminada
		Codificar el almacenamiento de las imágenes ingresada	Maria Ximena Rodriguez	Terminada
		Codificar la visualizaicon de las imágenes ingresada	Maria Ximena Rodriguez	Terminada
HU-07	Como un Dermatologo, necesito conocer que usuarios son de pago y cuales no	Diseñar los mockus de usuarios de pago y generales	Maria Ximena Rodriguez	Terminada
		Codificar	Maria Ximena Rodriguez	Terminada

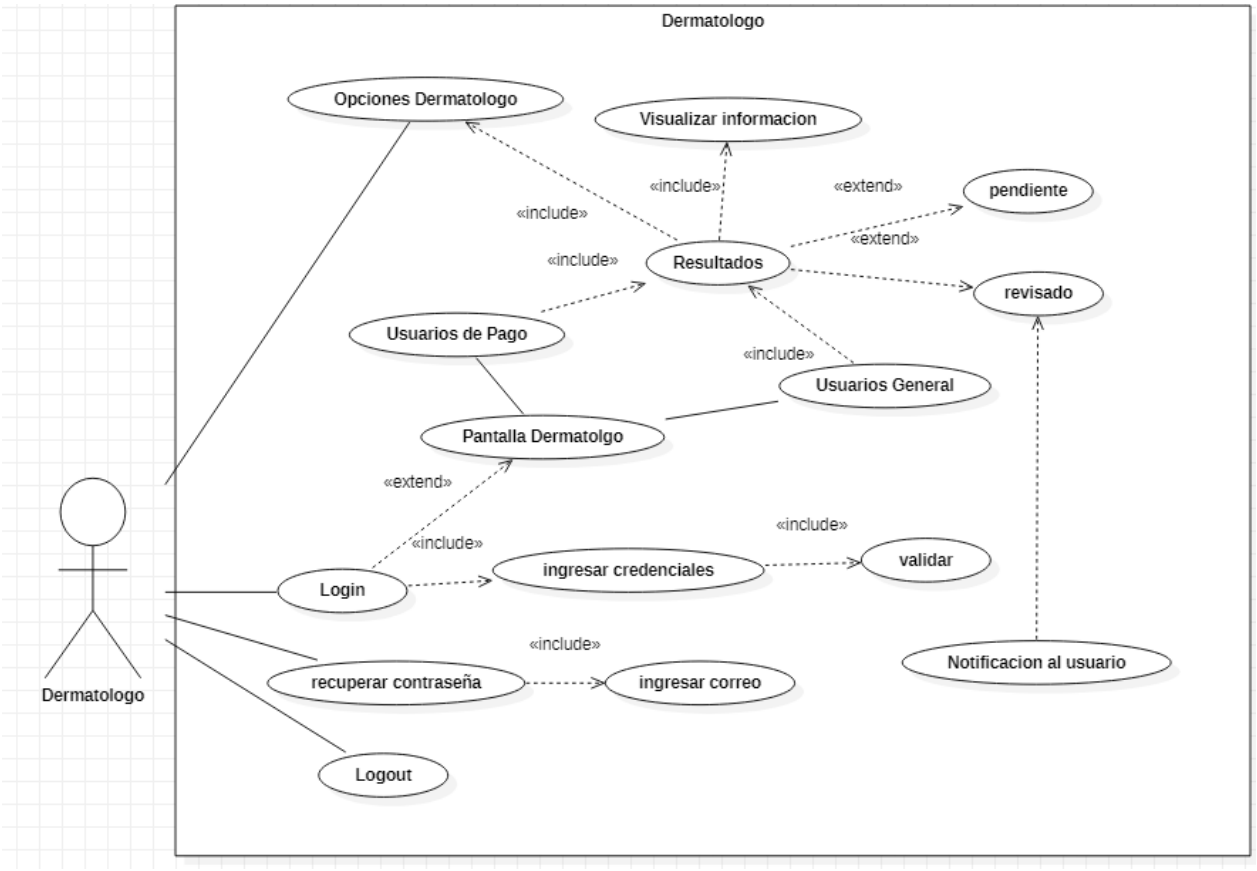
HU-08	Como un Dermatologo,necesito visualizar, controlar y notificar sobre los resultados de los usuarios	Diseñar los mosckups de las secciones de pendiente y revisado	Maria Ximena Rodriguez	Terminada
		Diseñar los mosckups la informacion del resultado	Maria Ximena Rodriguez	Terminada
		Diseñar los mosckups de los cambios del dermatologo	Maria Ximena Rodriguez	Terminada
		Codificar la seccion de pendiente y revisado	Maria Ximena Rodriguez	Terminada
		Codificar la informacion del resultado	Maria Ximena Rodriguez	Terminada
		Codificar los cambios que puede hacer el dermatologo	Maria Ximena Rodriguez	Terminada
		Codificar el almacenamiento y actualizacion de los cambios	Maria Ximena Rodriguez	Terminada
HU-09	Como un Administrador, necesito registrar dermatologos y editar la lista de clases que maneja el sistema	Diseñar los mosckups del registro del dermatologo	Maria Ximena Rodriguez	Terminada
		Diseñar los mosckups de la lista de clases	Maria Ximena Rodriguez	Terminada
		Codificar el registro del dermatologo	Maria Ximena Rodriguez	Terminada
		Codificar la lista de clases	Maria Ximena Rodriguez	Terminada

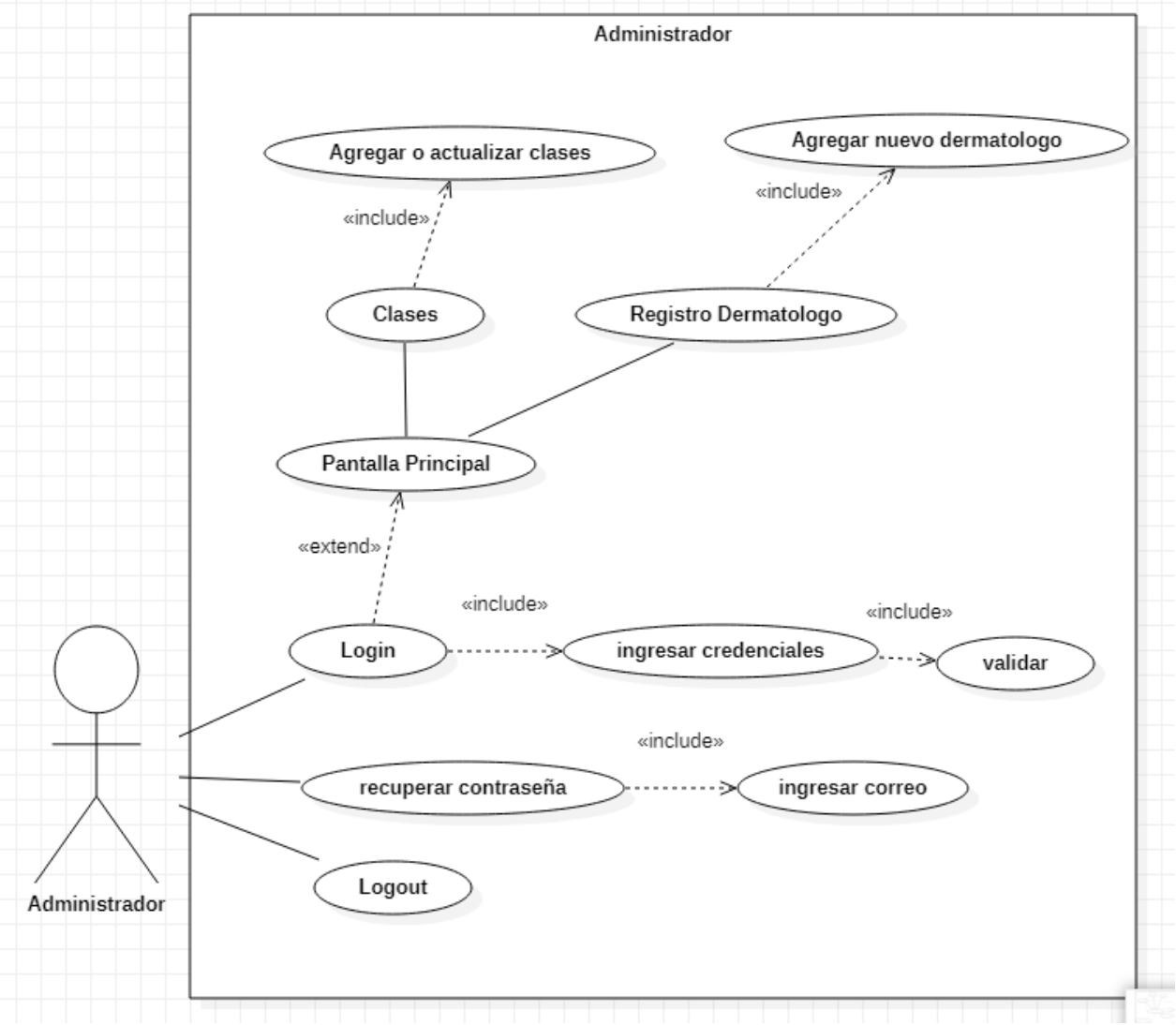
HU-10	Como sistema, necesito responder las solicitudes del cliente	Codificar la creacion y envio del pdf al correo con la informacion que se solicite	Maria Ximena Rodriguez	Terminada
		Codificar la actualizacion de las clases	Maria Ximena Rodriguez	Terminada
HU-11	Como un sistema, necesito procesar y relacionar la imagen con la prediccion	Codificar la conexión de la aplicación con el servicio web y envio de la imagen	Maria Ximena Rodriguez	Terminada
		Codificar el pre procesamiento de la imagen para luego ser clasificada	Maria Ximena Rodriguez	Terminada
		Codficar la obtencion de los valores de la prediccion que da el modelo	Maria Ximena Rodriguez	Terminada
		Codificar la relacion entre las clases y los valores para la clasificacion	Maria Ximena Rodriguez	Terminada
		Codificar el retorno de un json con los resultados a la aplicación movil	Maria Ximena Rodriguez	Terminada

16.2 Diseño

16.2.1 Diagrama de caso de Uso




Dermatologo
Opciones Dermatologo
Visualizar informacion
«include»
«include»
«extend»
pendiente
«extend»
«include»
Resultados
revisado
Usuarios de Pago
«include»
Usuarios General
Pantalla Dermatolgo
«extend»
«include»
«include»
ingresar credenciales
validar
Login
Dermatologo
«include»
Notificacion al usuario
recuperar contraseña
ingresar correo
Logout


Administrador
Agregar o actualizar clases
Agregar nuevo dermatologo
«include»
«include»
Clases
Registro Dermatologo
Pantalla Principal
«extend»
«include»
«include»
Login
ingresar credenciales
validar
«include»
recuperar contraseña
ingresar correo
Logout
Administrador

16.2.2 Diccionario de Datos

Colección Dermatologo		
Campo	Tipo de dato	Descripcion
id	PK	PrimaryKey
datospersonales	Objeto	informacion personal del usuario
rol	Número	perfil al que pertenece el dermatologo

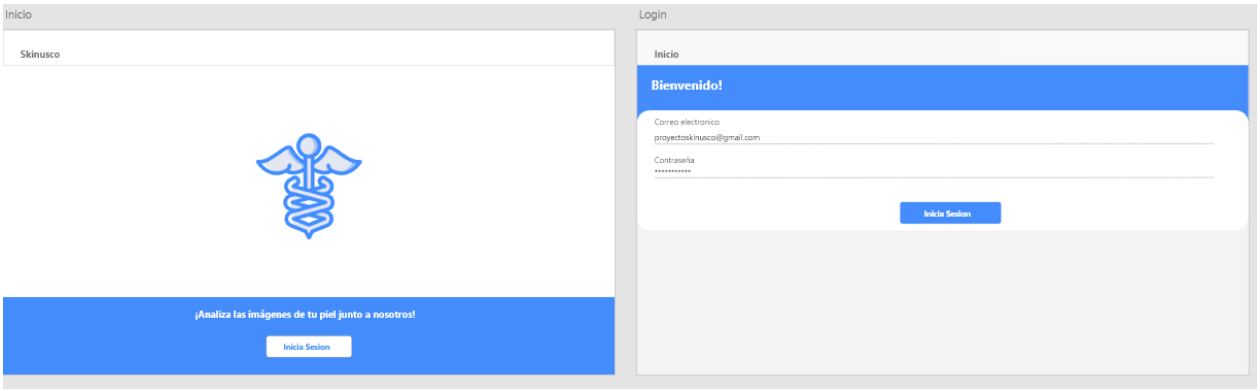
datospersonales_objeto / Dermatologo		
Campo	Tipo de dato	Descripcion
correo	Texto	correo del dermatologo
cuidad	Texto	cuidad
done	Boleano	cambio de estado
entidad	Texto	Lugar donde trabaja el dermatologo
genero	Texto	genero
nombre	Texto	nombre
pais	Texto	pais

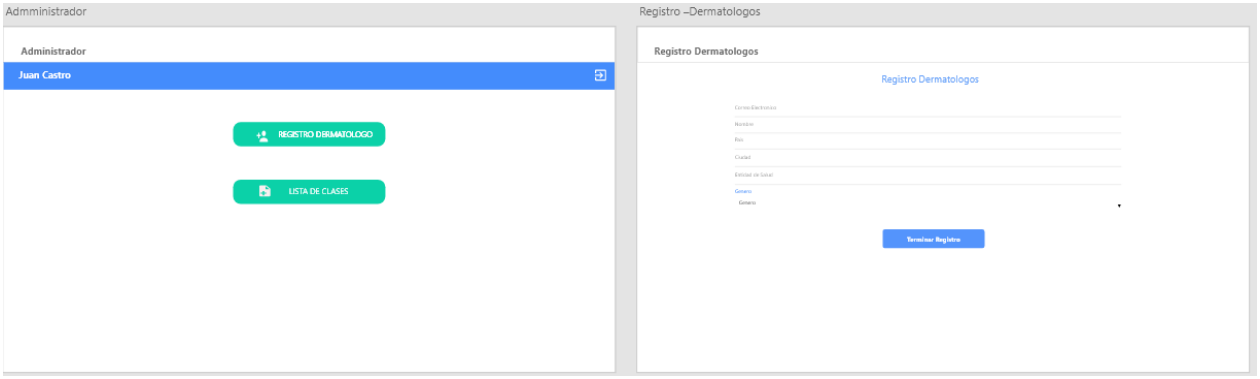
Colección Administrador		
Campo	Tipo de dato	Descripcion
id	PK	PrimaryKey
cuenta	Texto	correo
keycuenta	Texto	id cuenta
rol	Número	perfil al que pertenece el administrador

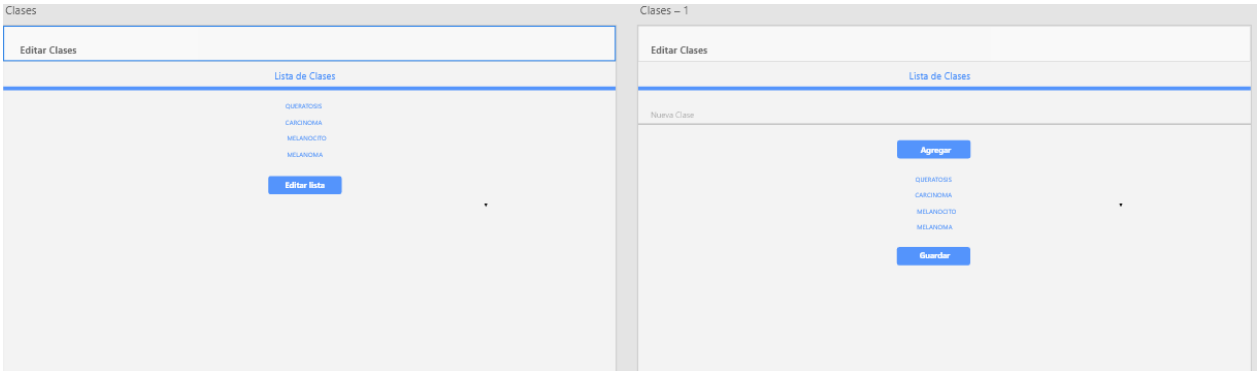
Colección Parametros		
Campo	Tipo de dato	Descripcion
Caterogias	Arreglo	las clases que utiliza el modelo para la clasificacion

16.2.3 Mockups

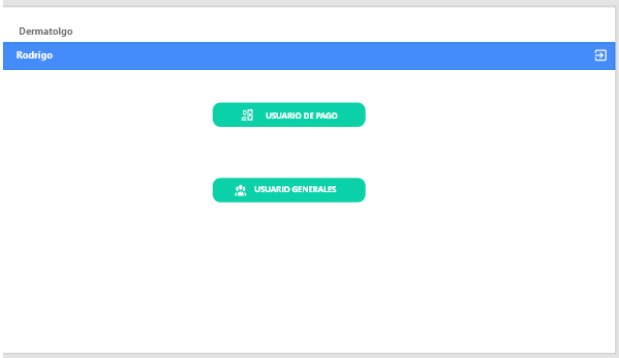
16.2.3.1 Web



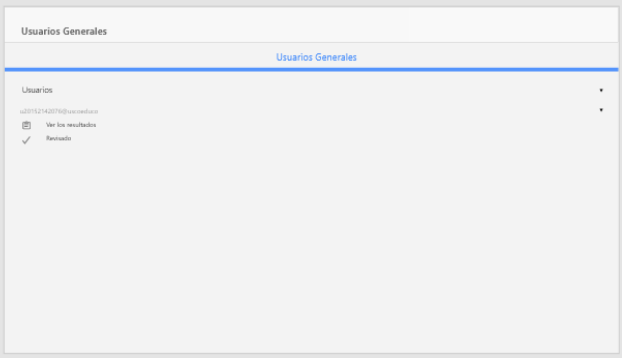




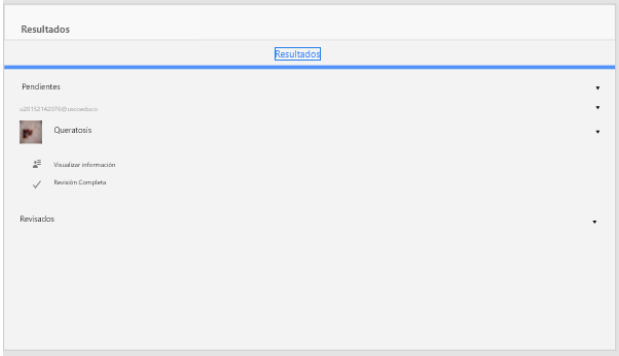
Dermatologo



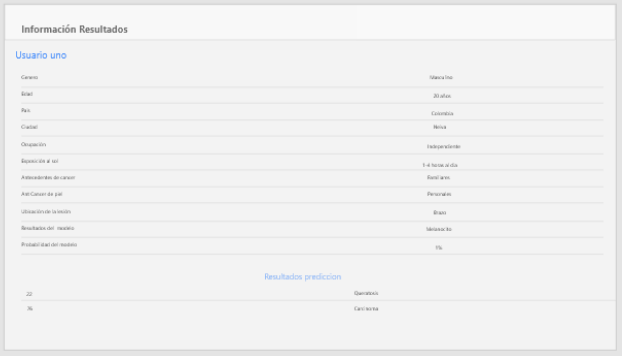
Usuarios



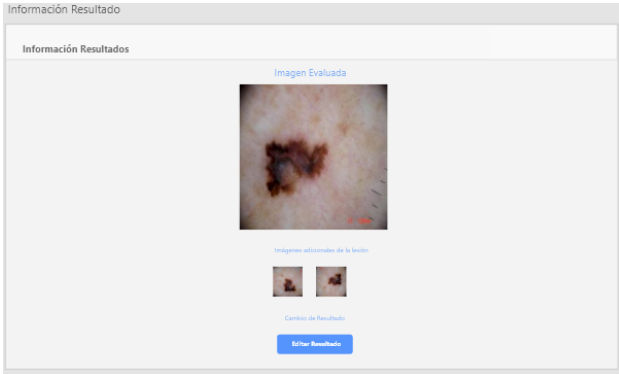
Usuarios-1



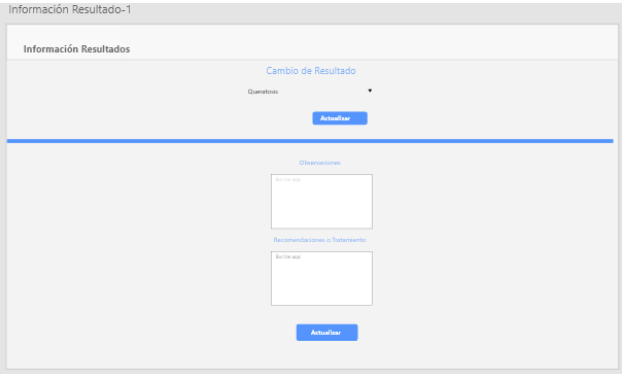
Usuarios-1 – 1



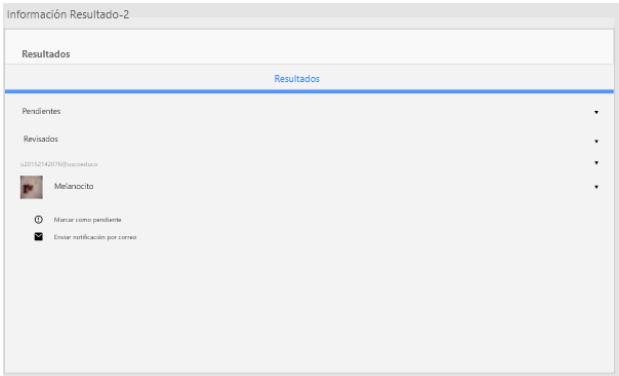
Información Resultado



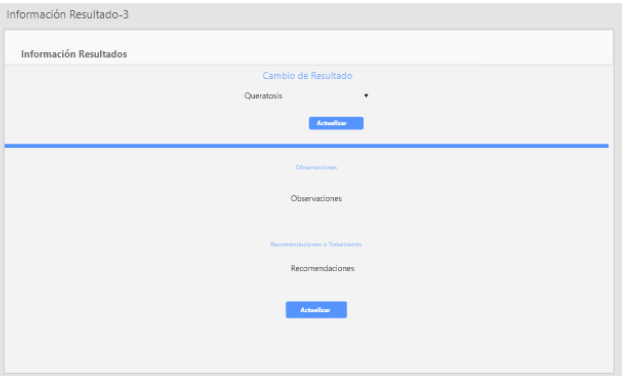
Información Resultado-1



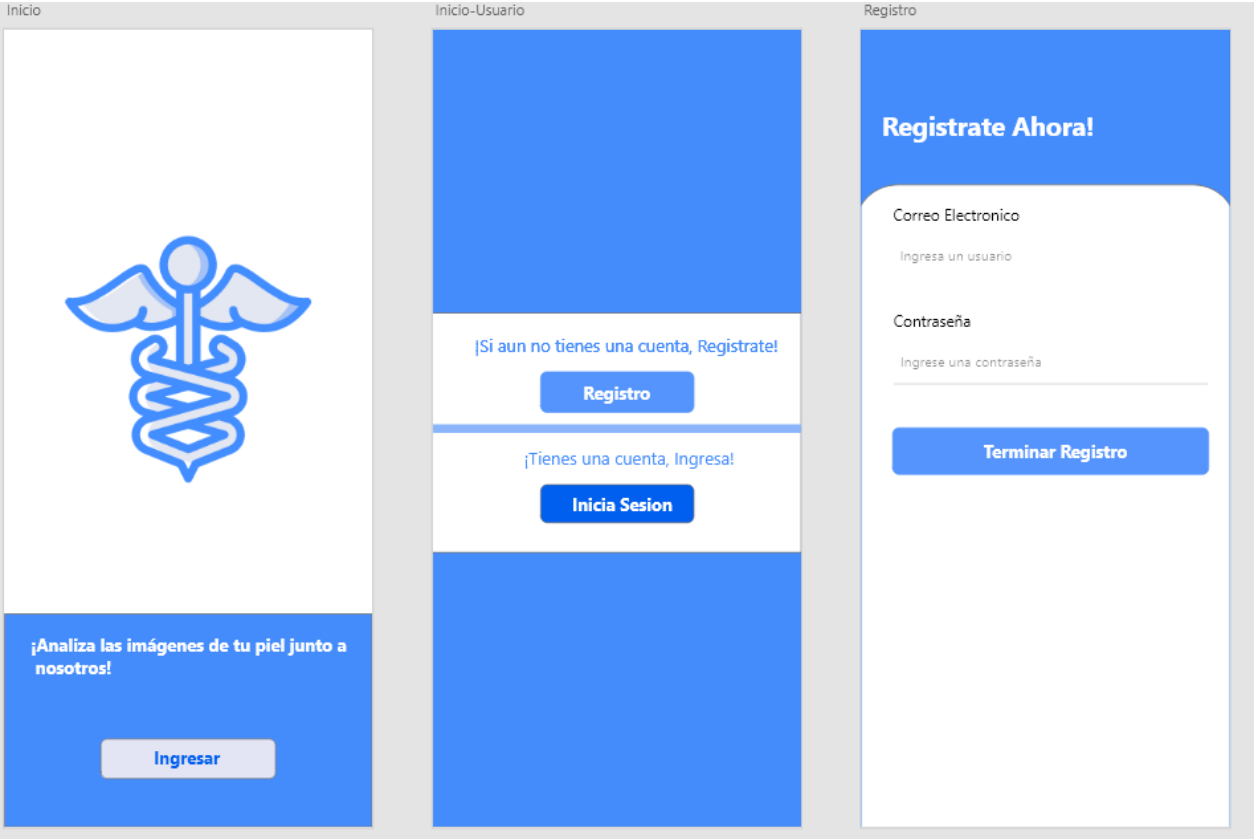
Información Resultado-2

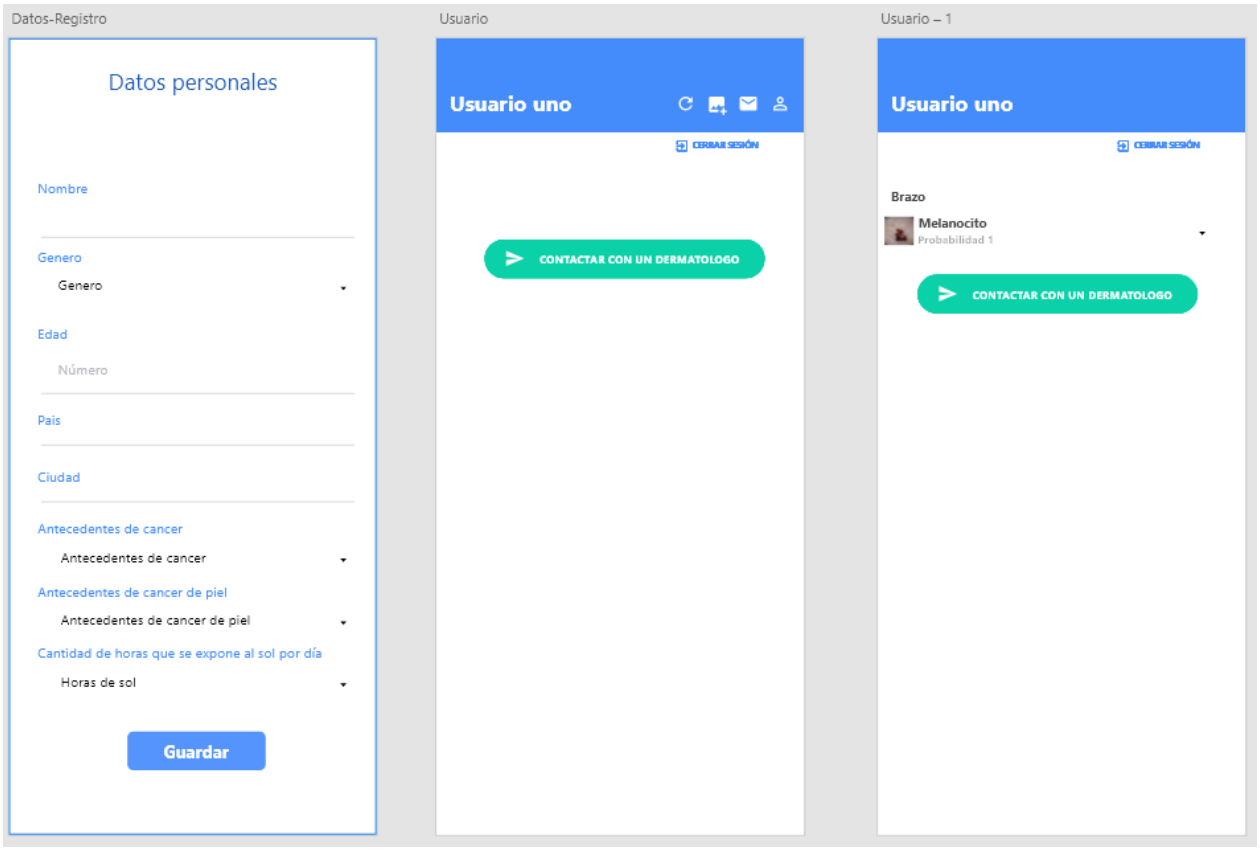


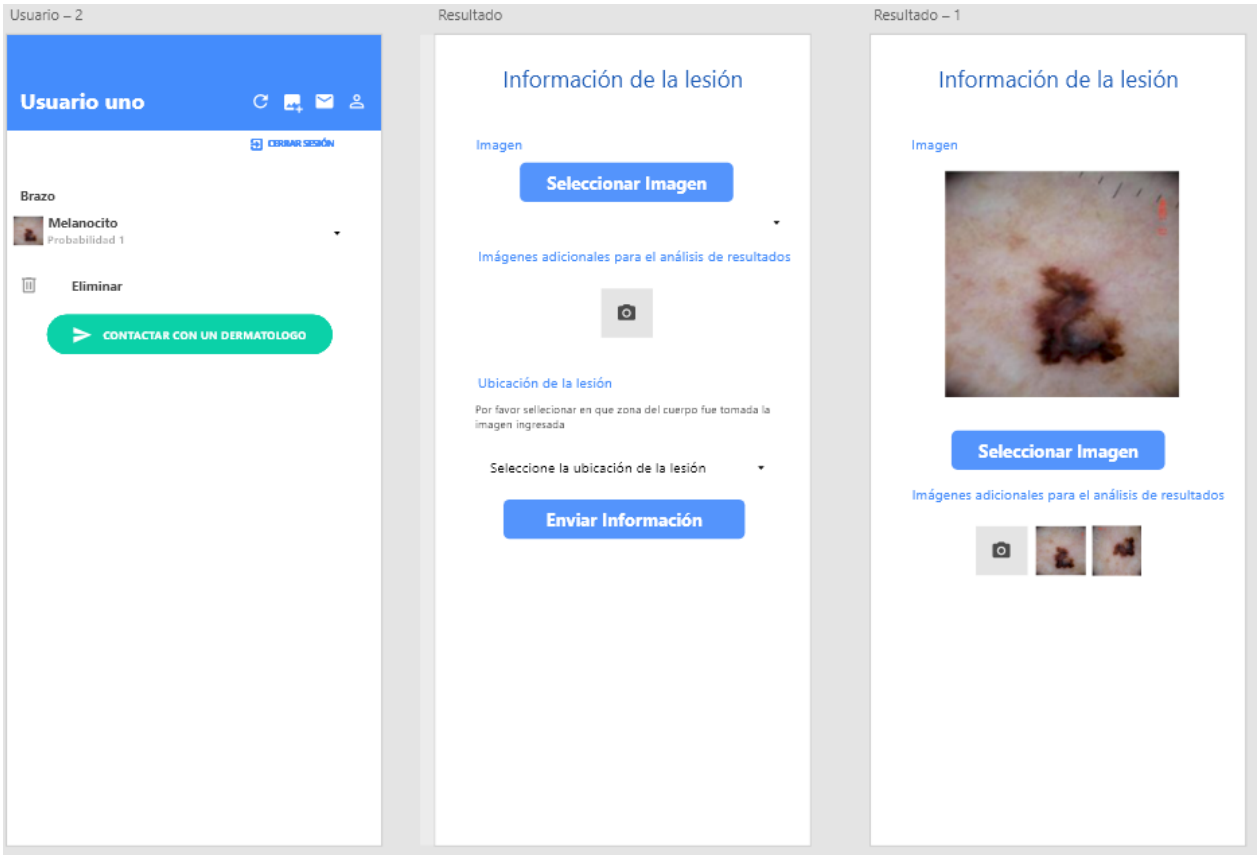
Información Resultado-3

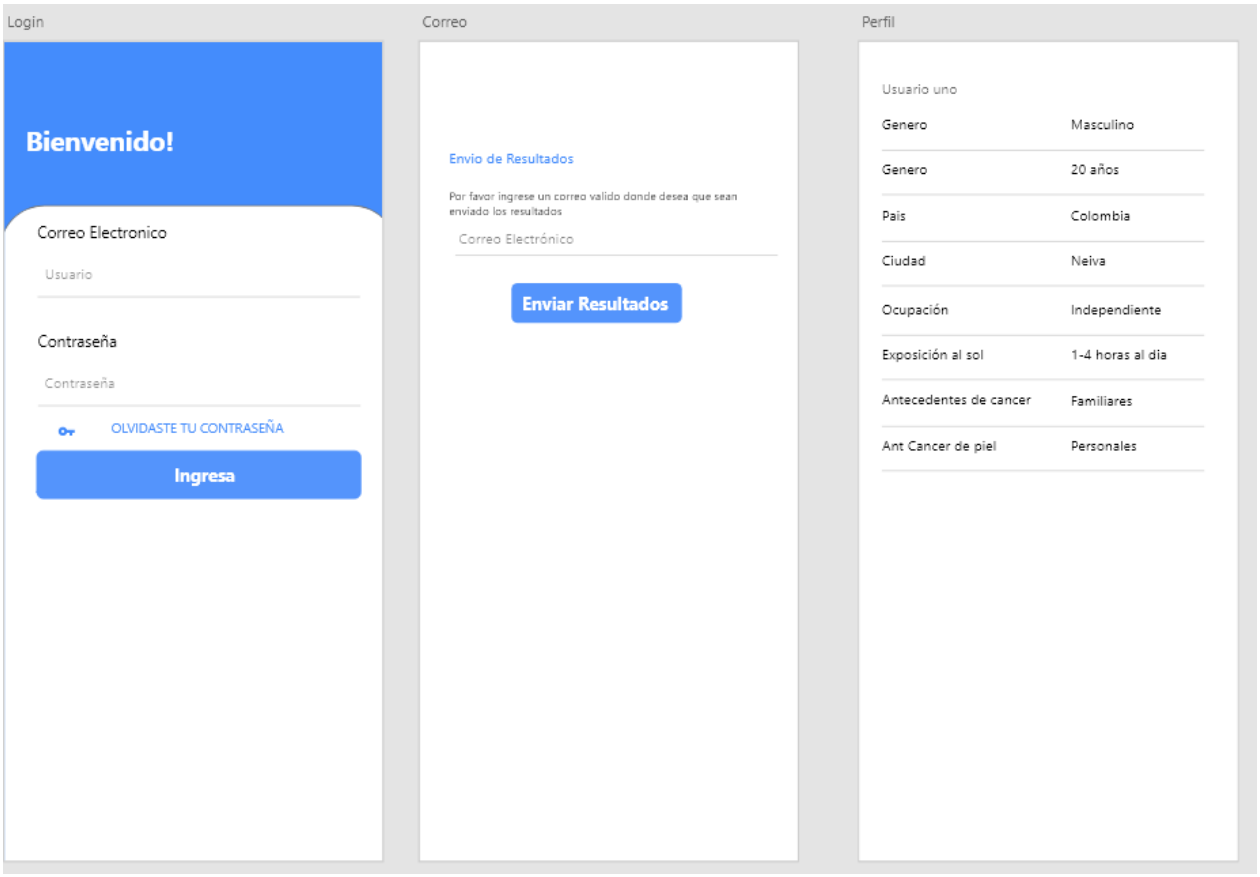


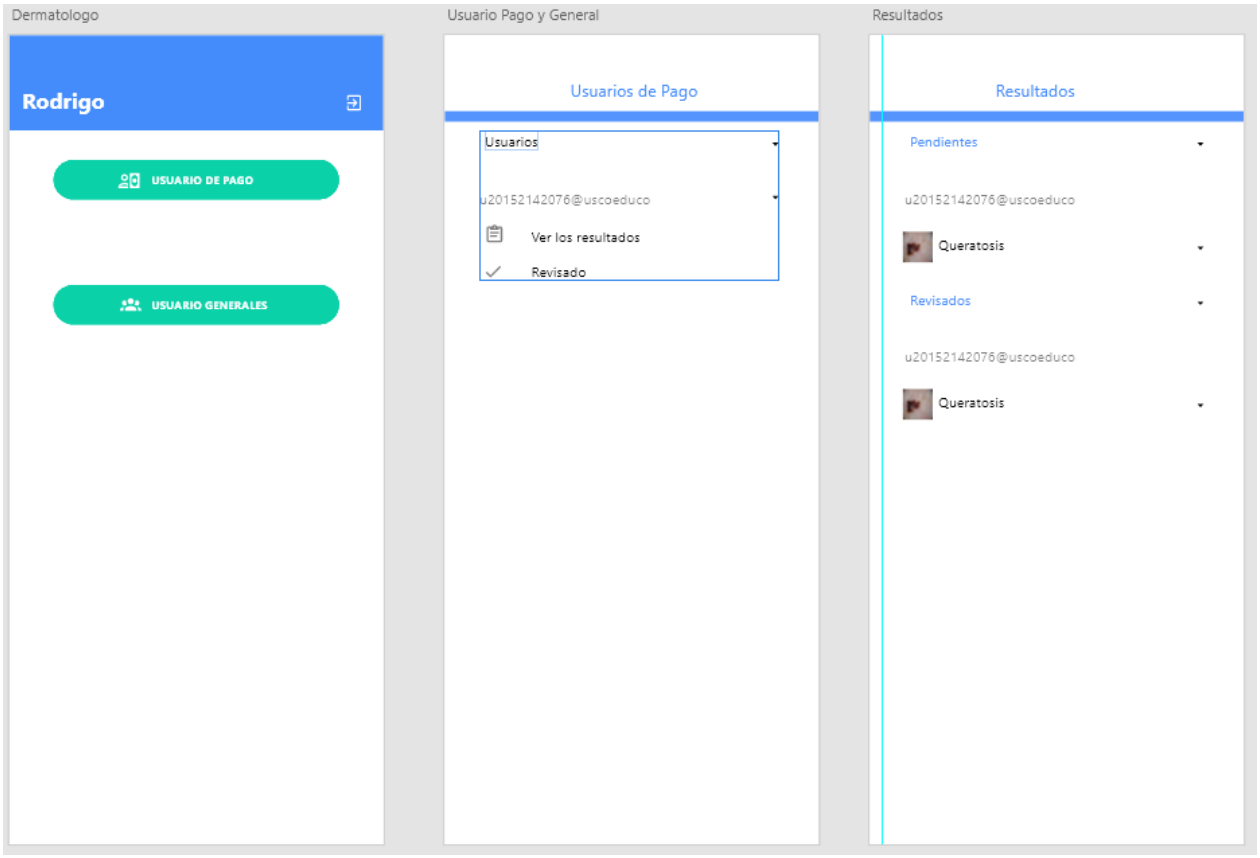
16.2.4 Móvil

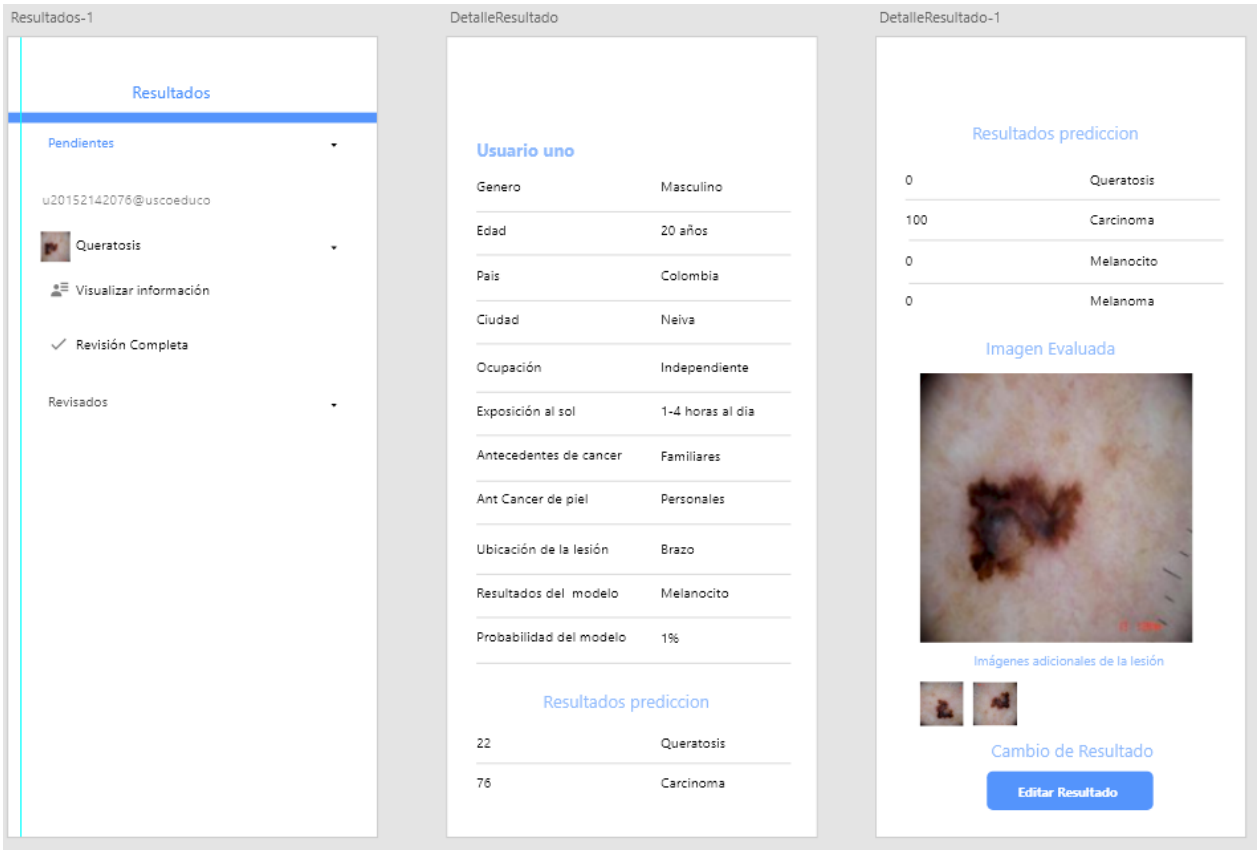

Inicio
¡Analiza las imágenes de tu piel junto a nosotros!
Ingresar
Inicio-Usuario
¡Si aun no tienes una cuenta, Registrate!
Registro
¡Tienes una cuenta, Ingresa!
Inicia Sesion
Registro
Registrate Ahora!
Correo Electronico
Ingresa un usuario
Contraseña
Ingrese una contraseña
Terminar Registro

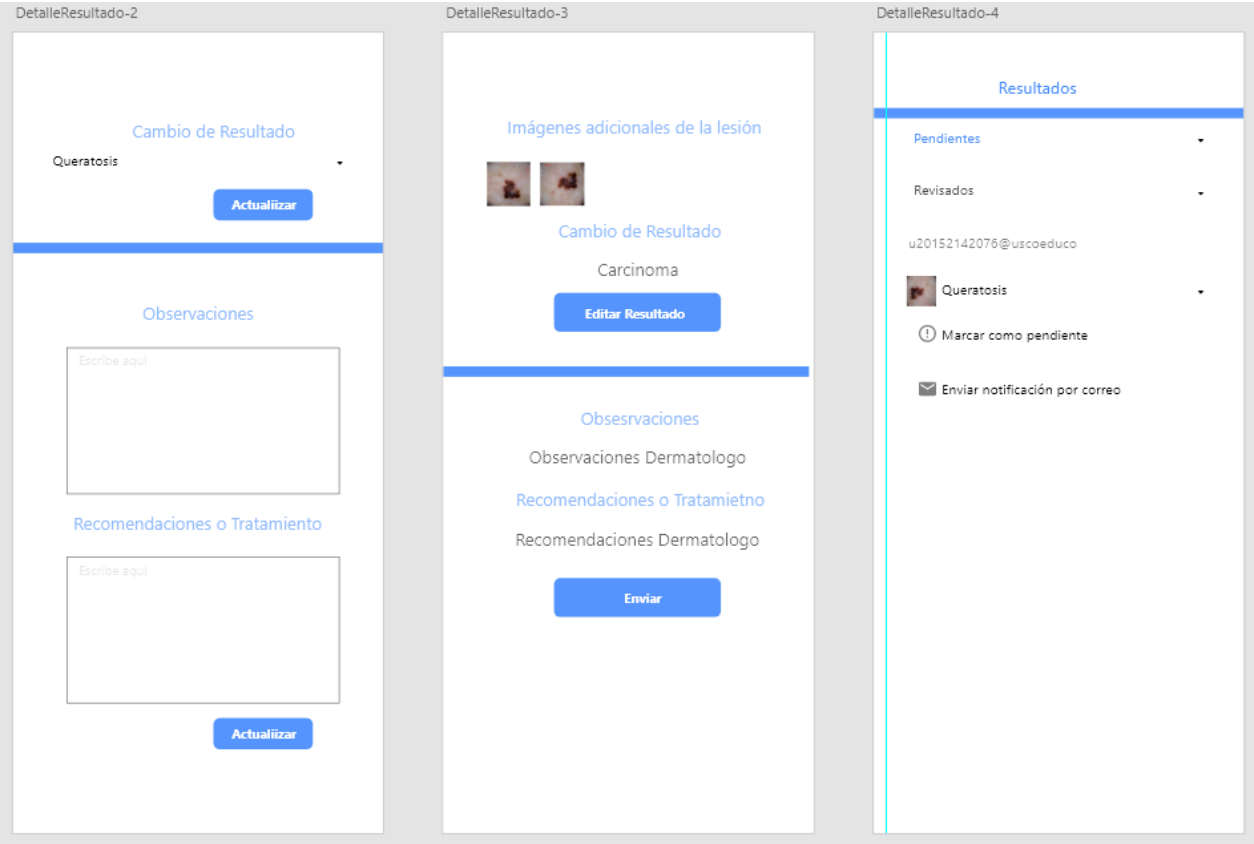

Datos-Registro
Datos personales
Nombre
Genero
Genero
Edad
Número
Pais
Ciudad
Antecedentes de cancer
Antecedentes de cancer
Antecedentes de cancer de piel
Antecedentes de cancer de piel
Cantidad de horas que se expone al sol por día
Horas de sol
Guardar
Usuario
Usuario uno
CERRAR SESIÓN
CONTACTAR CON UN DERMATOLOGO
Usuario – 1
Usuario uno
CERRAR SESIÓN
Brazo
Melanocito
Probabilidad 1
CONTACTAR CON UN DERMATOLOGO

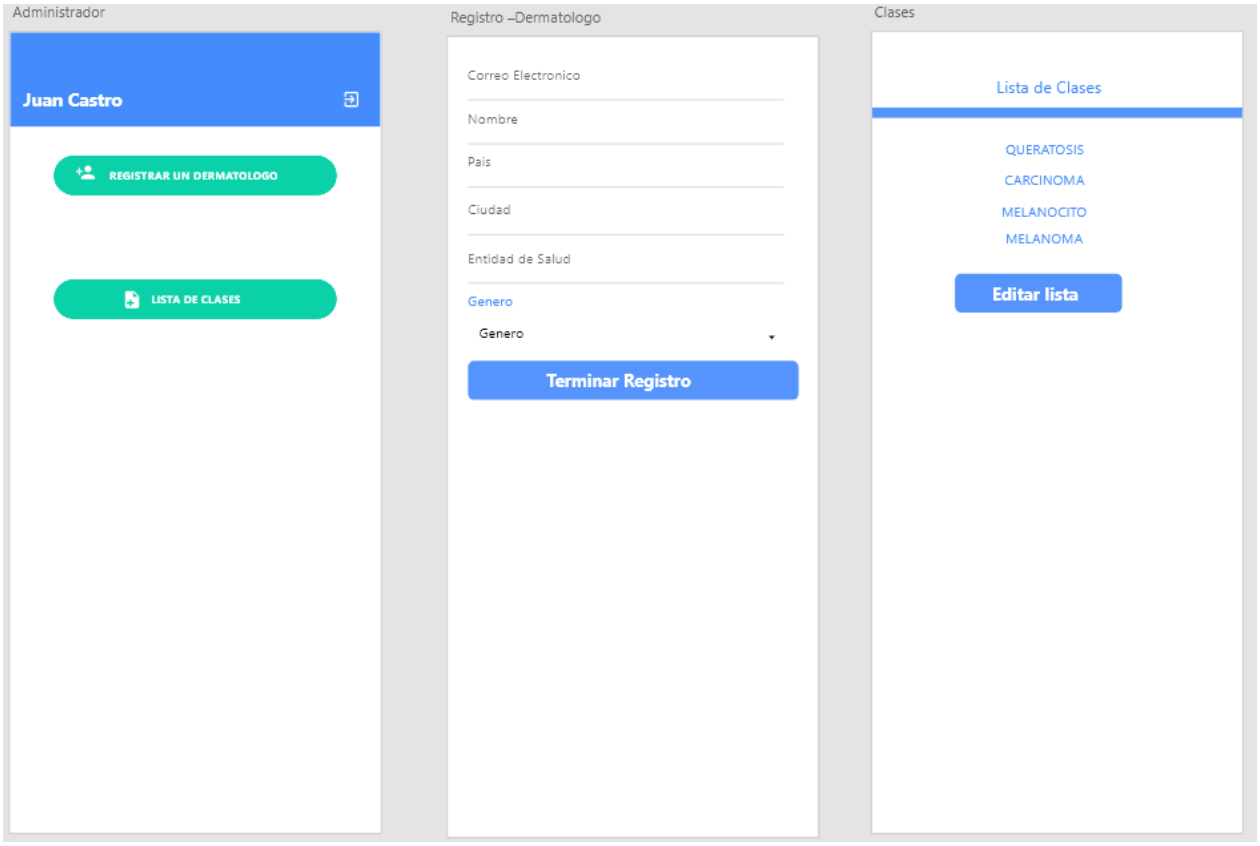

Usuario – 2
Usuario uno
CERRAR SESIÓN
Brazo
Melanocito
Probabilidad 1
Eliminar
CONTACTAR CON UN DERMATOLOGO
Resultado
Información de la lesión
Imagen
Seleccionar Imagen
Imágenes adicionales para el análisis de resultados
Ubicación de la lesión
Por favor sellecionar en que zona del cuerpo fue tomada la imagen ingresada
Seleccione la ubicación de la lesión
Enviar Información
Resultado – 1
Información de la lesión
Imagen
Seleccionar Imagen
Imágenes adicionales para el análisis de resultados


Login
Bienvenido!
Correo Electronico
Usuario
Contraseña
Contraseña
OLVIDASTE TU CONTRASEÑA
Ingresa
Correo
Envio de Resultados
Por favor ingrese un correo valido donde desea que sean enviado los resultados
Correo Electrónico
Enviar Resultados
Perfil
Usuario uno
Genero
Masculino
Genero
20 años
Pais
Colombia
Ciudad
Neiva
Ocupación
Independiente
Exposición al sol
1-4 horas al dia
Antecedentes de cancer
Familiares
Ant Cancer de piel
Personales


Dermatologo
Rodrigo
USUARIO DE PAGO
USUARIO GENERALES
Usuario Pago y General
Usuarios de Pago
Usuarios
u20152142076@uscoeduco
Ver los resultados
Revisado
Resultados
Resultados
Pendientes
u20152142076@uscoeduco
Queratosis
Revisados
u20152142076@uscoeduco
Queratosis


Resultados-1
Resultados
Pendientes
u20152142076@uscoeduco
Queratosis
Visualizar información
Revisión Completa
Revisados
DetalleResultado
Usuario uno
Genero Masculino
Edad 20 años
Pais Colombia
Ciudad Neiva
Ocupación Independiente
Exposición al sol 1-4 horas al dia
Antecedentes de cancer Familiares
Ant Cancer de piel Personales
Ubicación de la lesión Brazo
Resultados del modelo Melanocito
Probabilidad del modelo 1%
Resultados prediccion
22 Queratosis
76 Carcinoma
DetalleResultado-1
Resultados prediccion
0 Queratosis
100 Carcinoma
0 Melanocito
0 Melanoma
Imagen Evaluada
Imágenes adicionales de la lesión
Cambio de Resultado
Editar Resultado

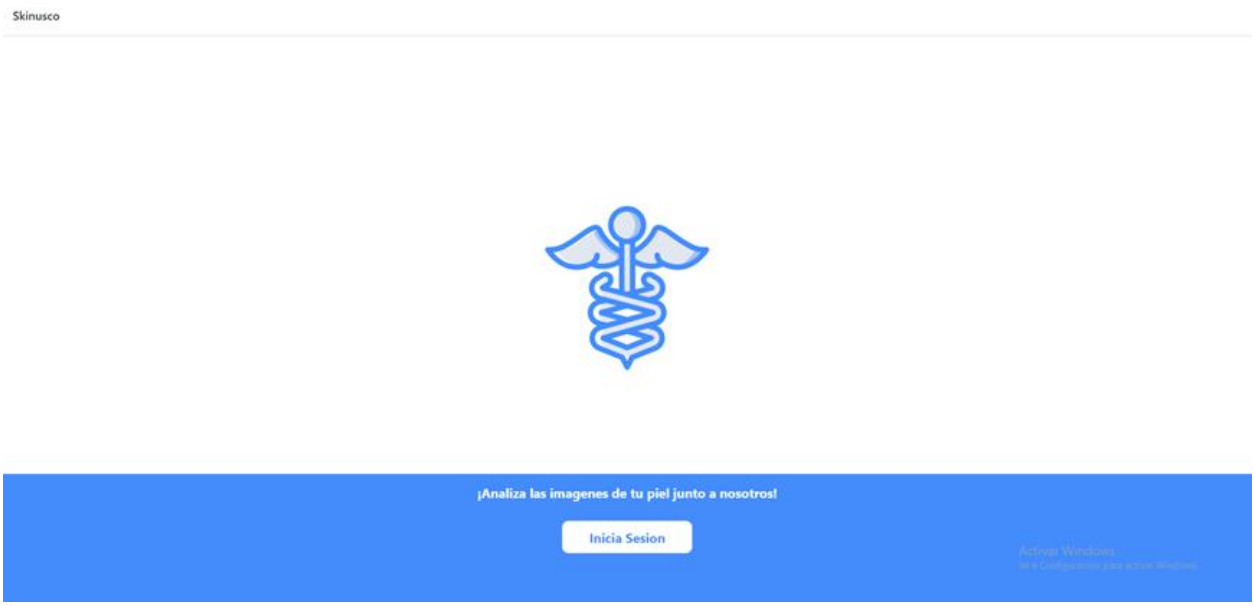

DetalleResultado-2
Cambio de Resultado
Queratosis
Actualiizar
Observaciones
Escribe aqui
Recomendaciones o Tratamiento
Escribe aqui
Actualiizar
DetalleResultado-3
Imágenes adicionales de la lesión
Cambio de Resultado
Carcinoma
Editar Resultado
Obsesrvaciones
Observaciones Dermatologo
Recomendaciones o Tratamietno
Recomendaciones Dermatologo
Enviar
DetalleResultado-4
Resultados
Pendientes
Revisados
u20152142076@uscoeduco
Queratosis
Marcar como pendiente
Enviar notificación por correo


Administrador
Juan Castro
REGISTRAR UN DERMATOLOGO
LISTA DE CLASES
Registro –Dermatologo
Correo Electronico
Nombre
Pais
Ciudad
Entidad de Salud
Genero
Genero
Terminar Registro
Clases
Lista de Clases
QUERATOSIS
CARCINOMA
MELANOCITO
MELANOMA
Editar lista

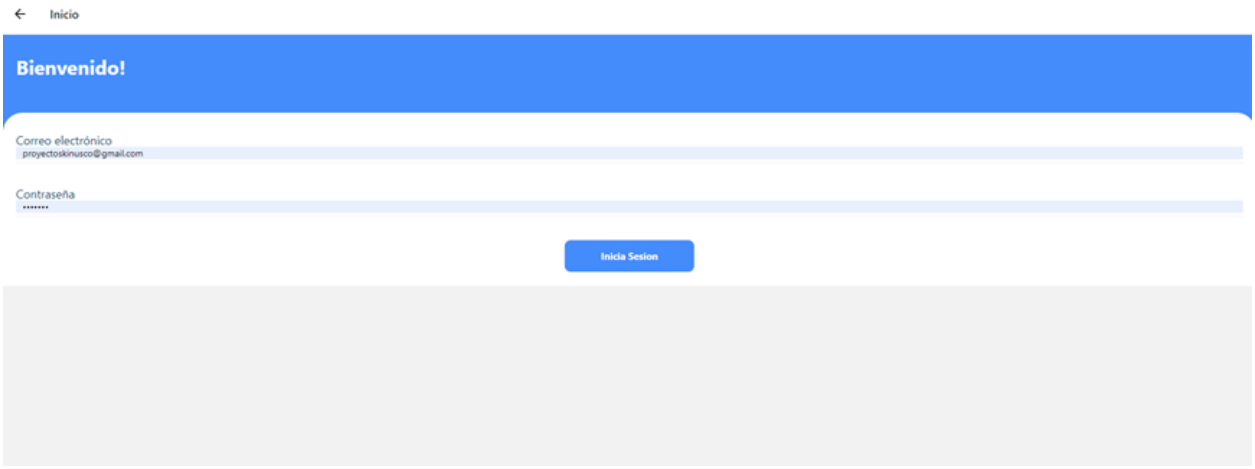
16.3 Anexos Resultados

16.3.1 Desarrollo Web

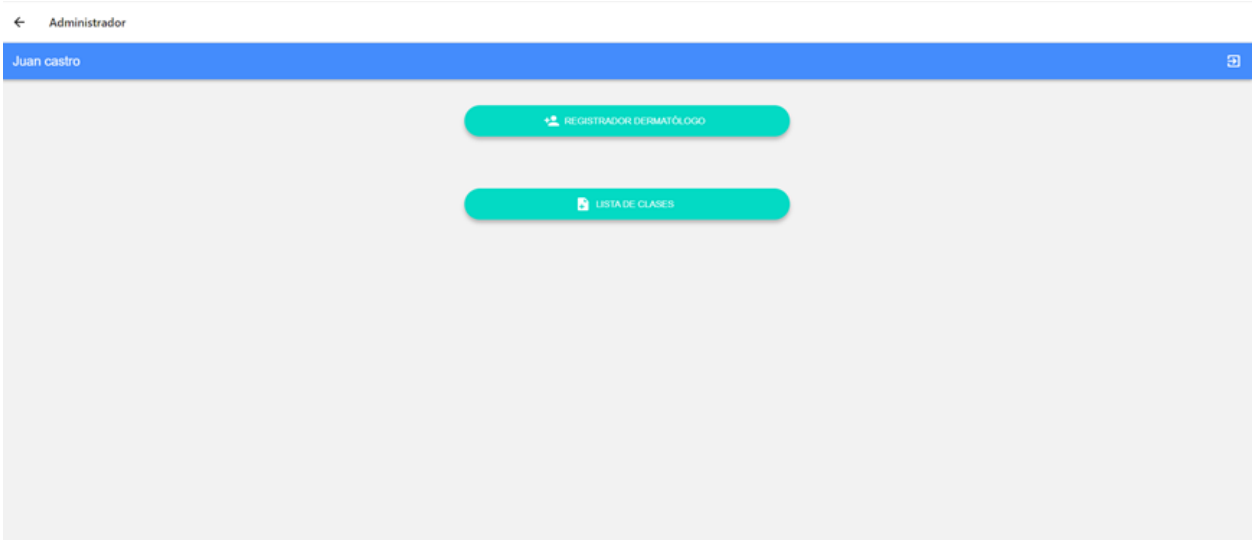
Pantalla de Bienvenida



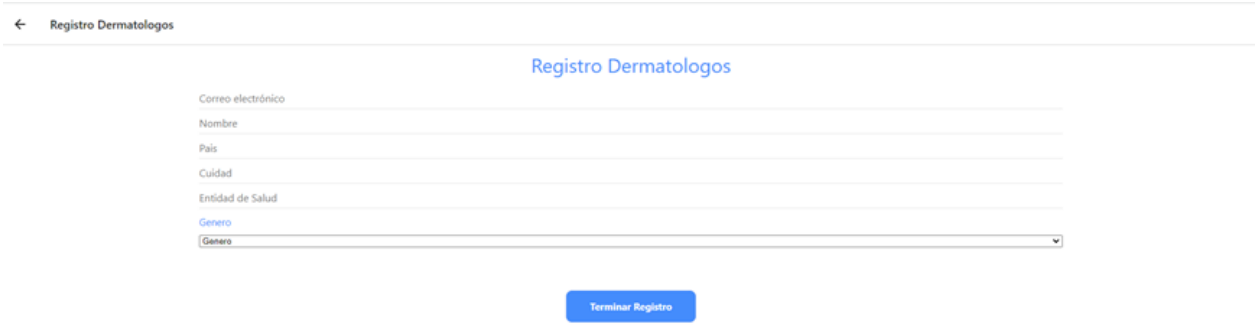
Login



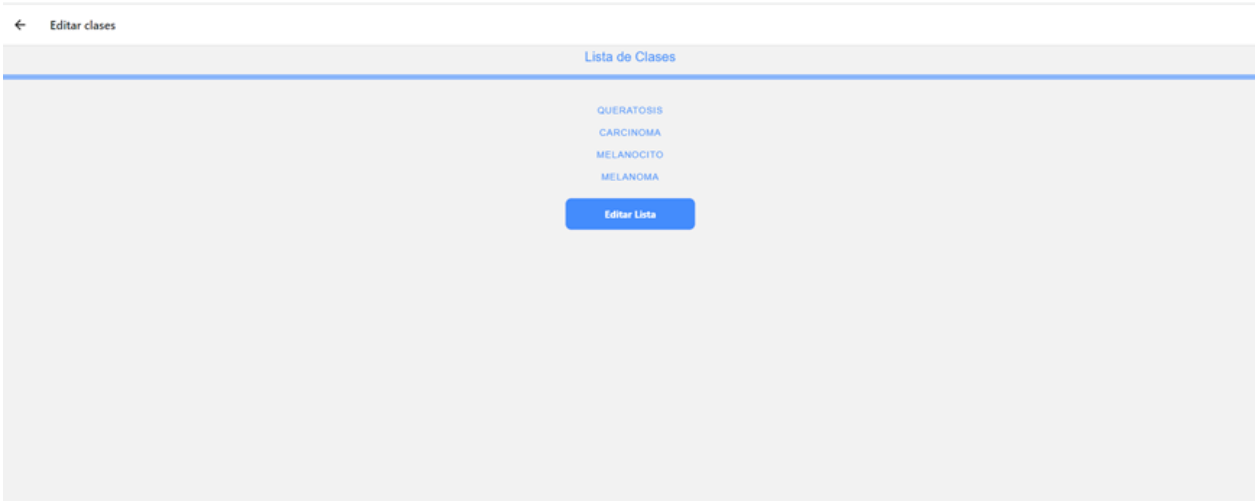
Pantalla principal Administrador

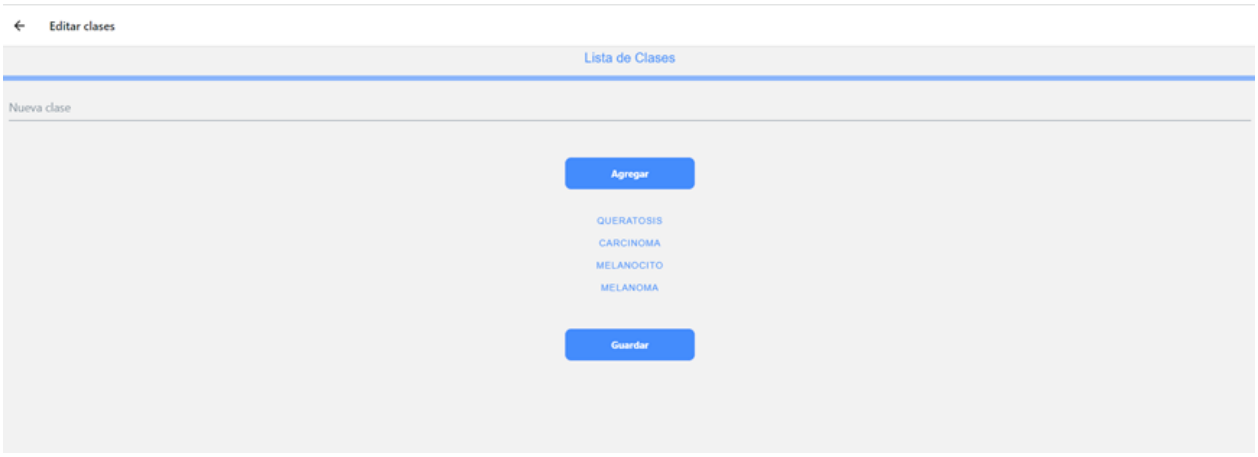


Registro Dermatólogo por parte del Administrador

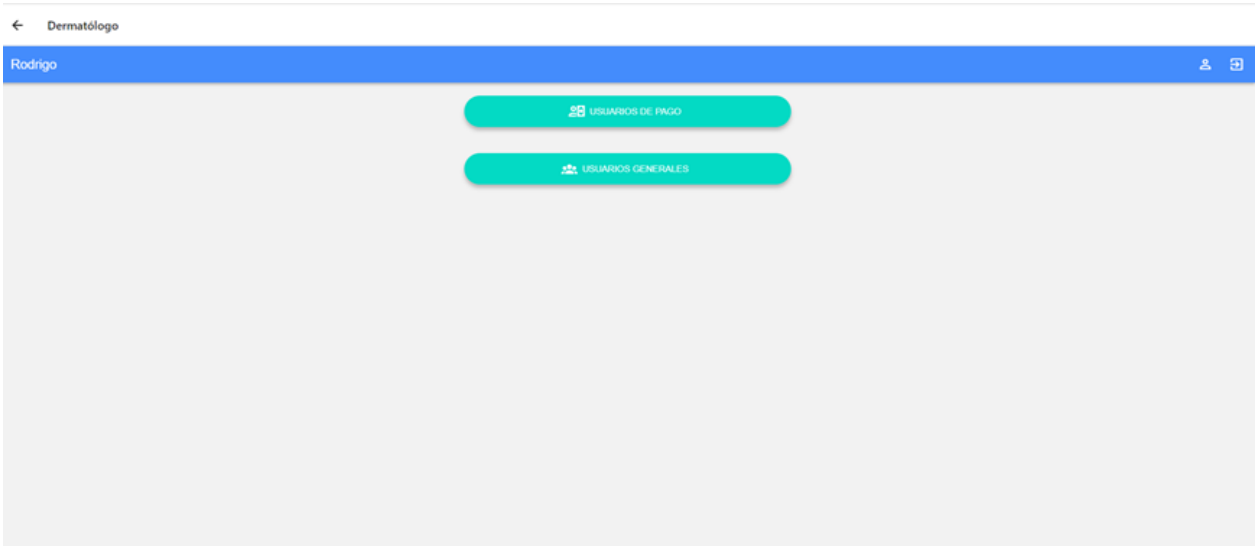


Cambio o actualización de las clases que utiliza el modelo - Administrador

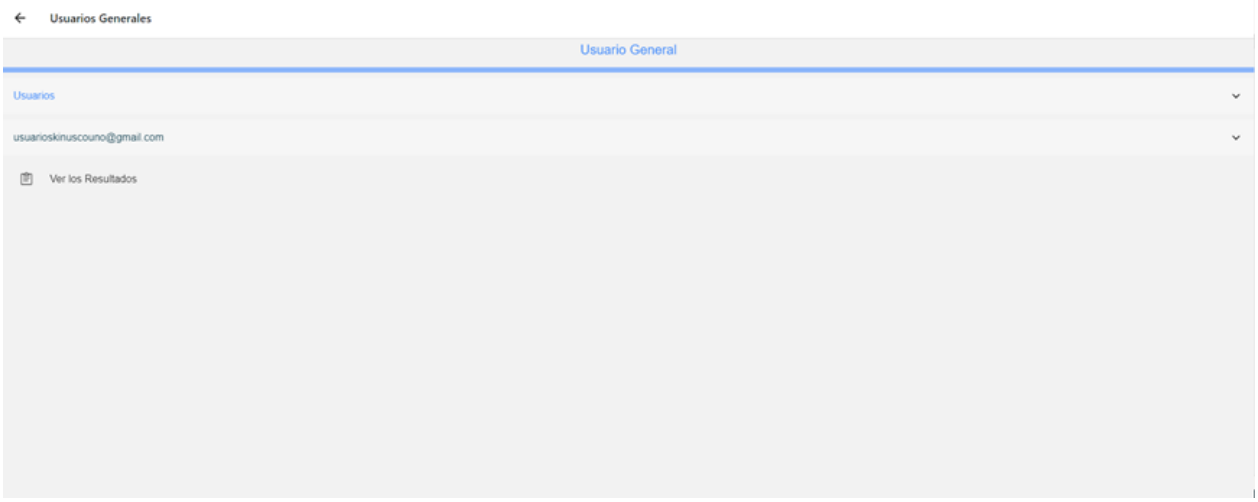




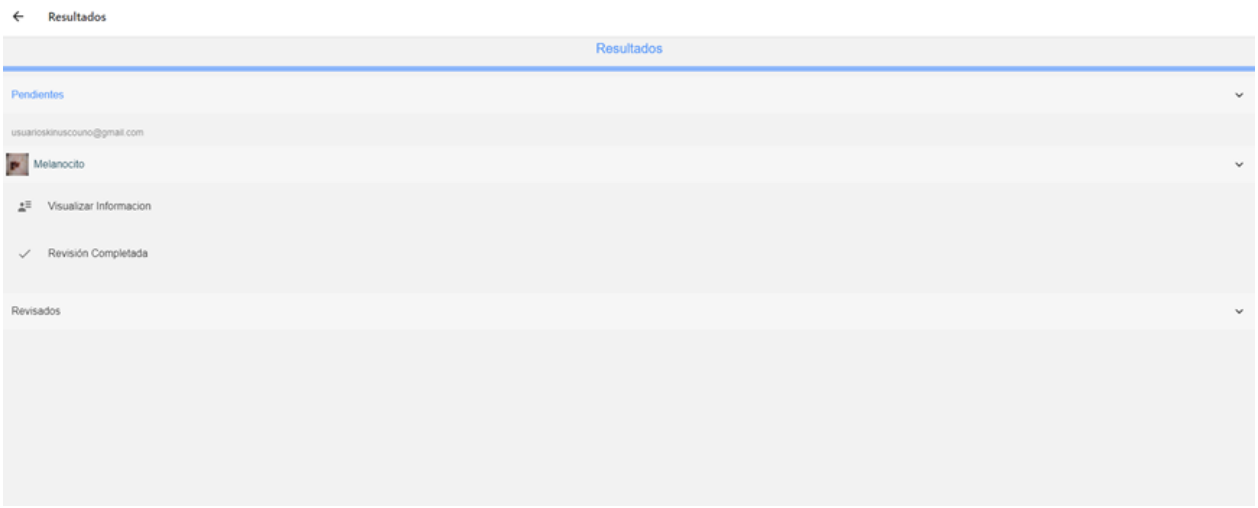
Pantalla principal Dermatólogo



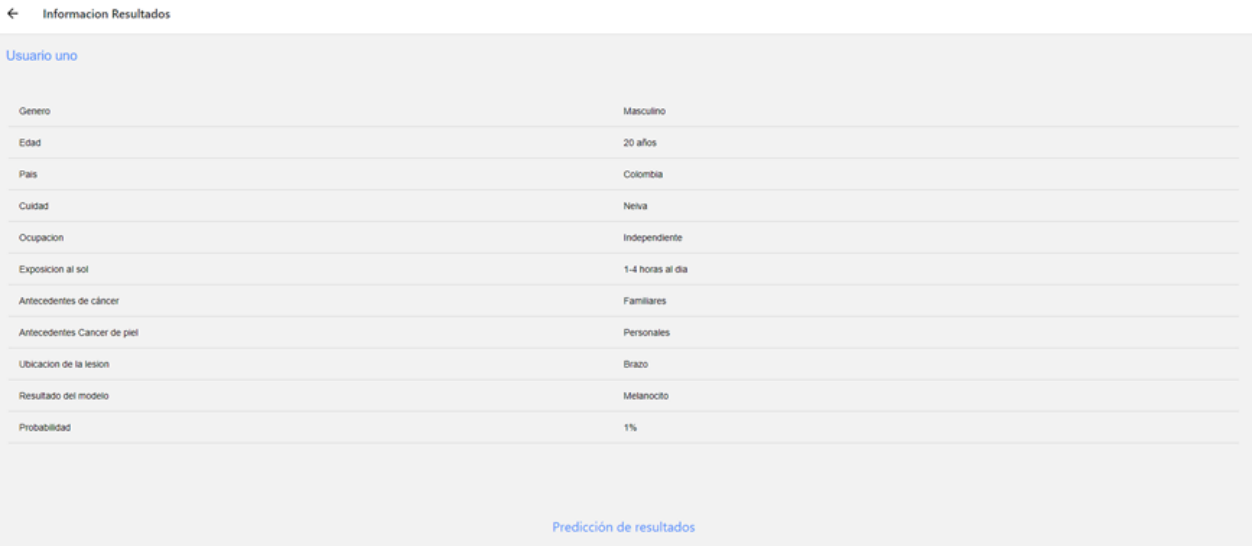
Lista de usuarios de pago o general, con opciones de ver resultados y marcar como revisado

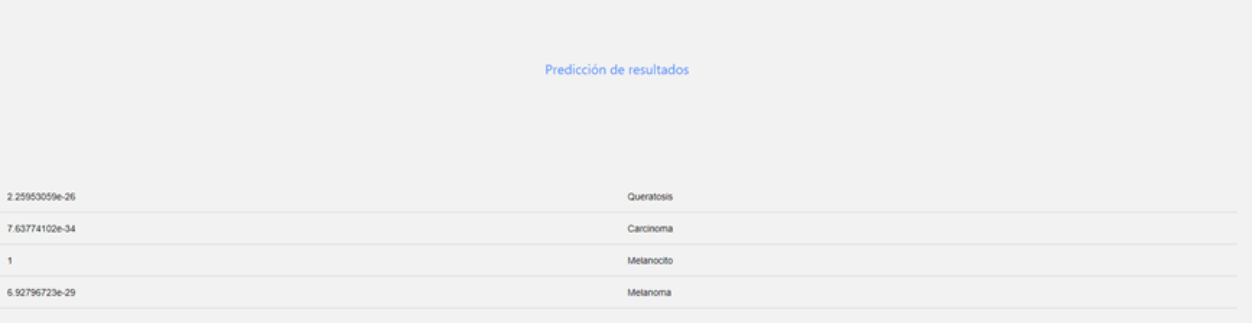


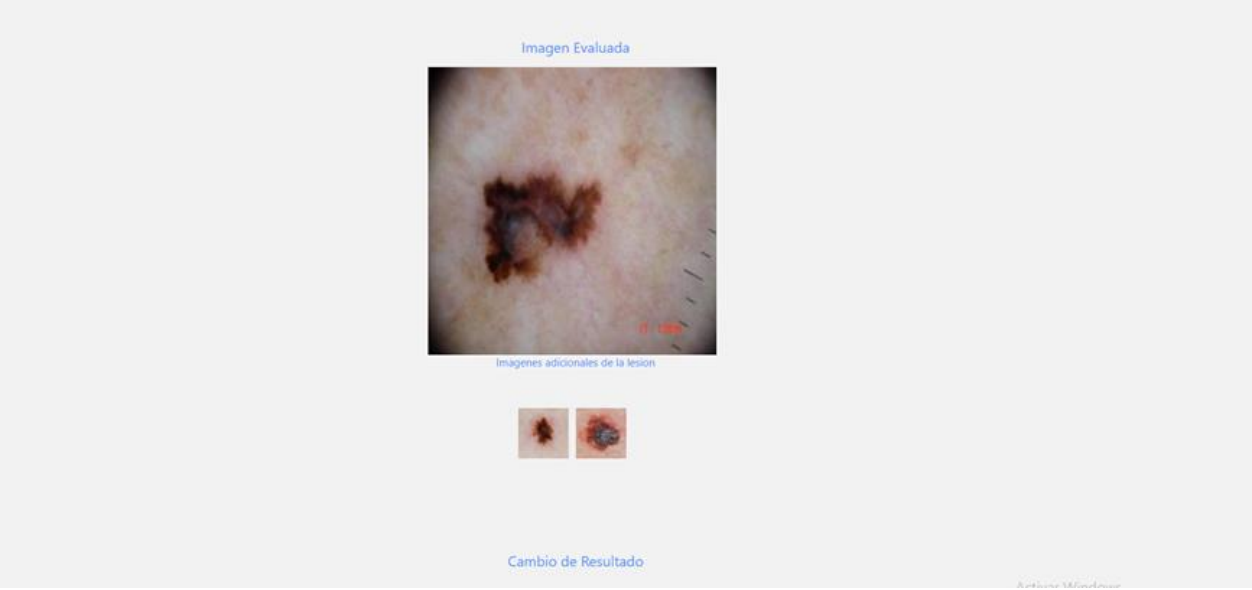
Lista de resultados, seccionado en pendientes y revisados.



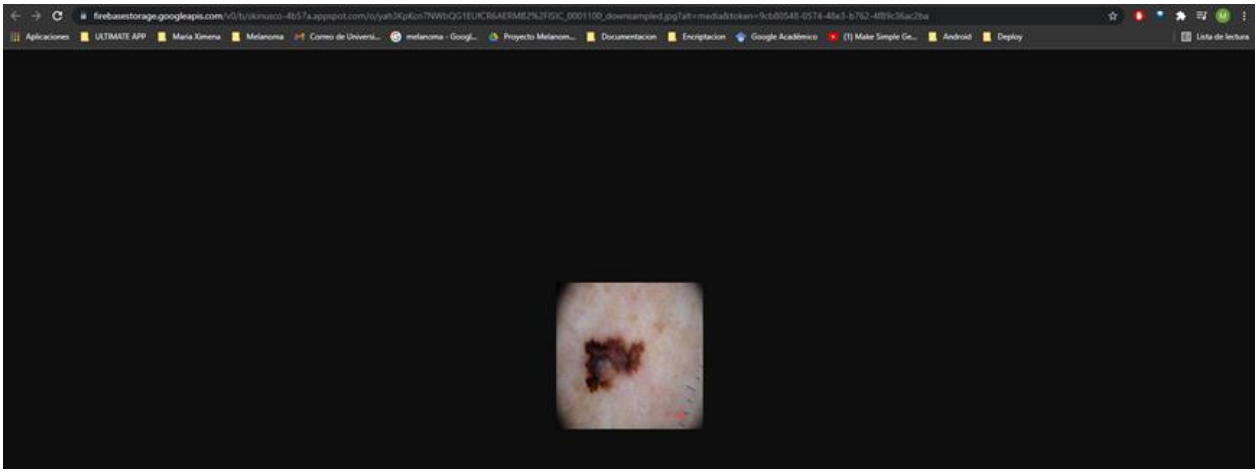
Detalle e información del usuario respecto a cada resultado.



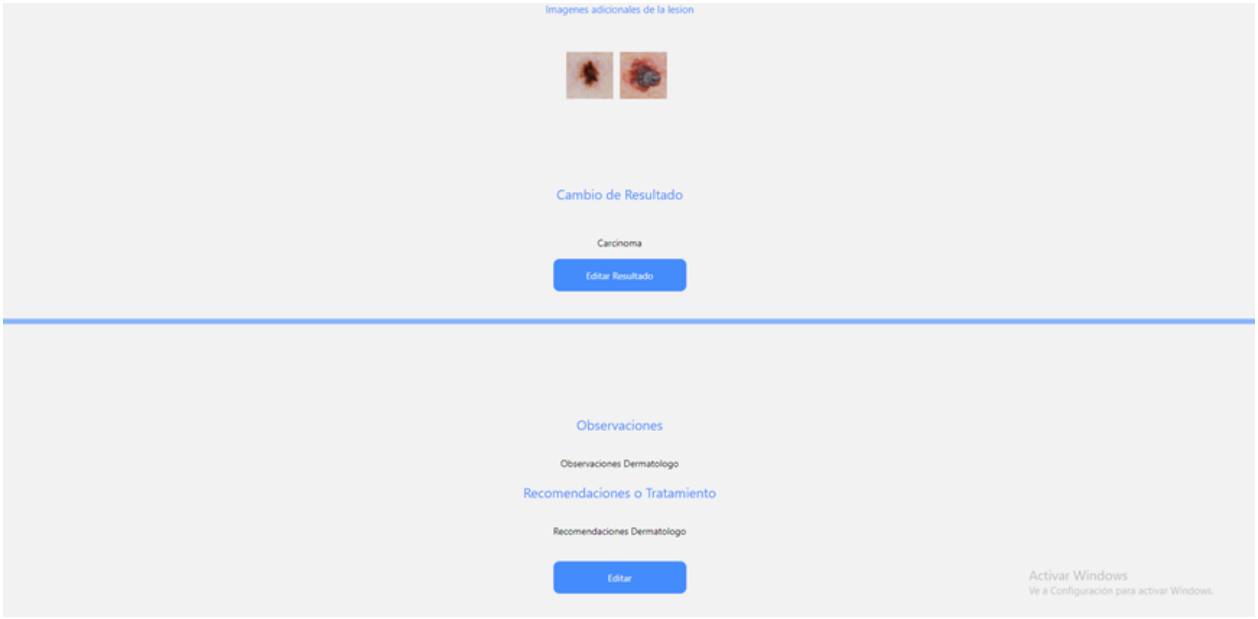




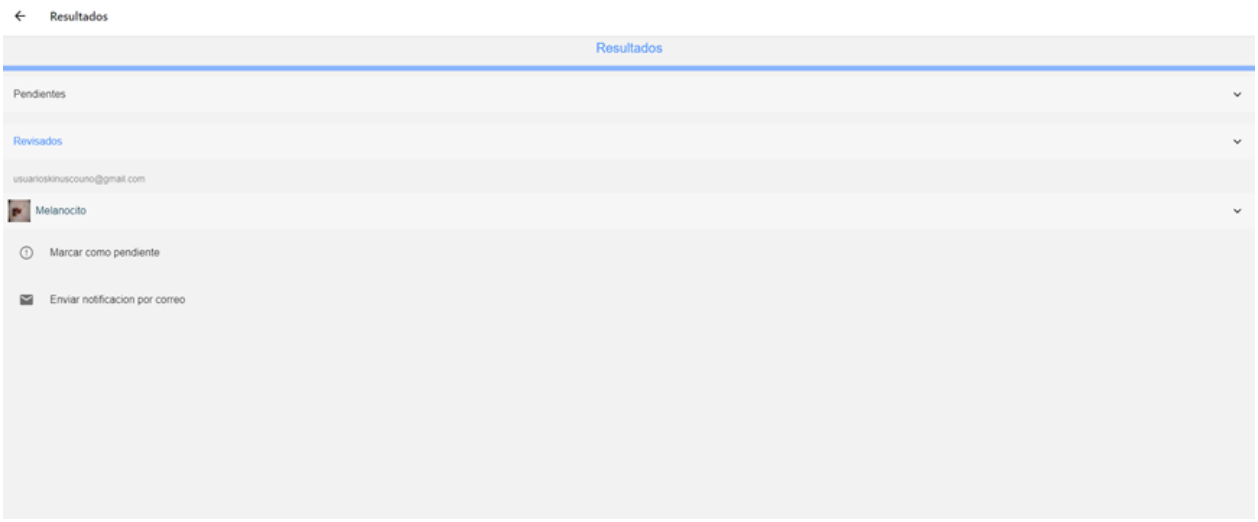
Las imágenes se podrán visualizar al dar clic sobre ellas.



En esta sección de la información del resultado, el dermatólogo tiene las opciones de cambiar el resultado que se obtuvo del modelo, como también de agregar Observaciones, Recomendaciones o Tratamiento según sea la lesión y el caso.



Cuando el resultado de un usuario de pago o general ya sea revisado y se encuentre en esta sección de la lista, el dermatólogo tendrá la opción de marcarlo como pendiente nuevamente, o de enviar un PDF al correo electrónico del usuario con los cambios y opiniones hechas por el dermatólogo.

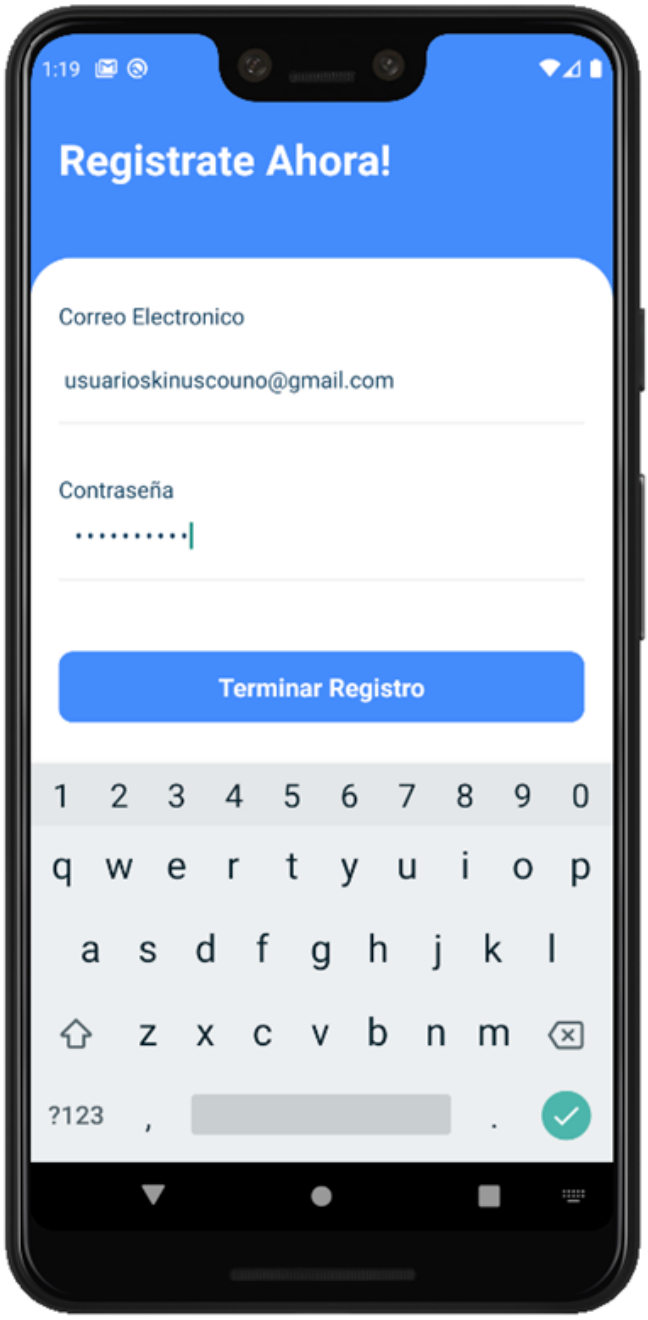


16.3.2 Desarrollo Móvil

1. Esta es la pantalla de bienvenida del aplicativo.

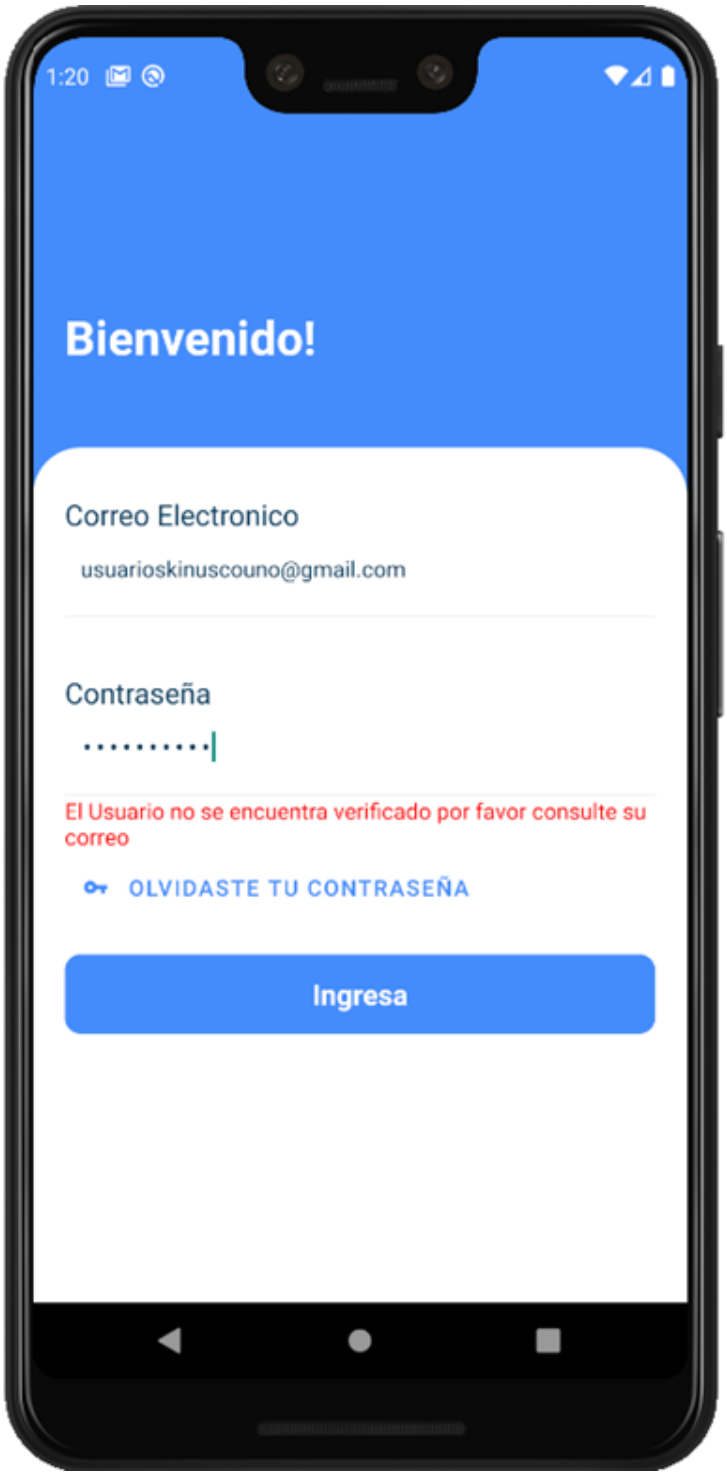
2. Aquí tenemos las opciones de registro o inicio de sesión.

3. Al ingresar a Registro, tenemos esta pantalla donde, vamos a registrar un usuario nuevo.



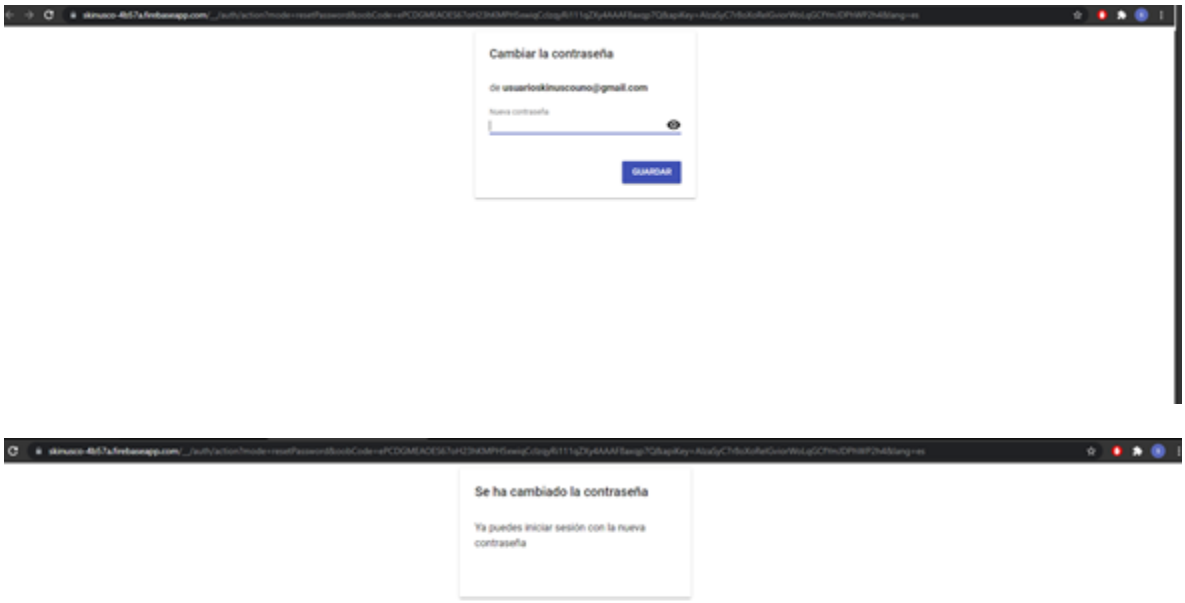
4. Luego de que el usuario ha sido registrado, se enviará un enlace de verificación al correo registrado.

5. Si el usuario no ha verificado su correo mediante el enlace, e intenta iniciar sesión el aplicativo mostrará el siguiente mensaje.

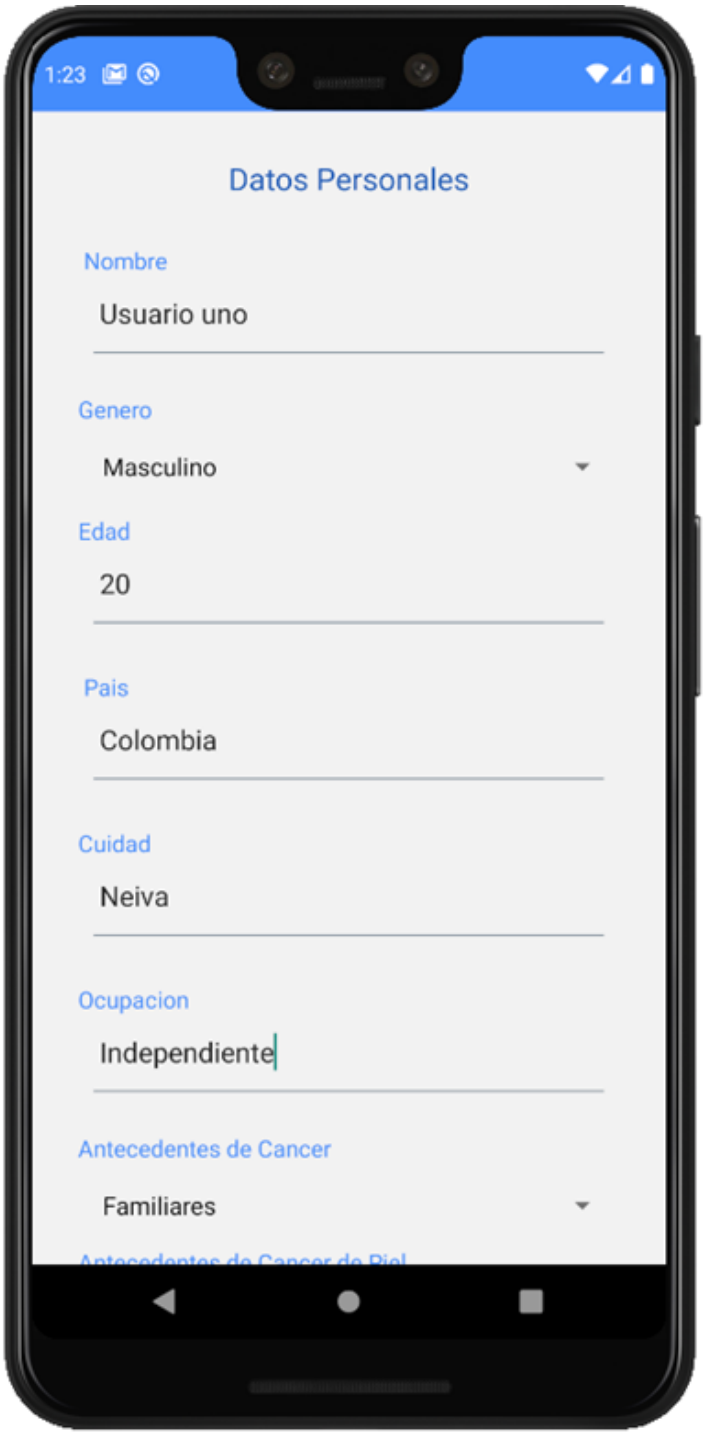


6. Si el usuario al ingresar no recuerda su contraseña, al dar clic ‘Olvidaste tu contraseña’, el sistema enviará un enlace al correo electrónico para realizar el cambio de contraseña.

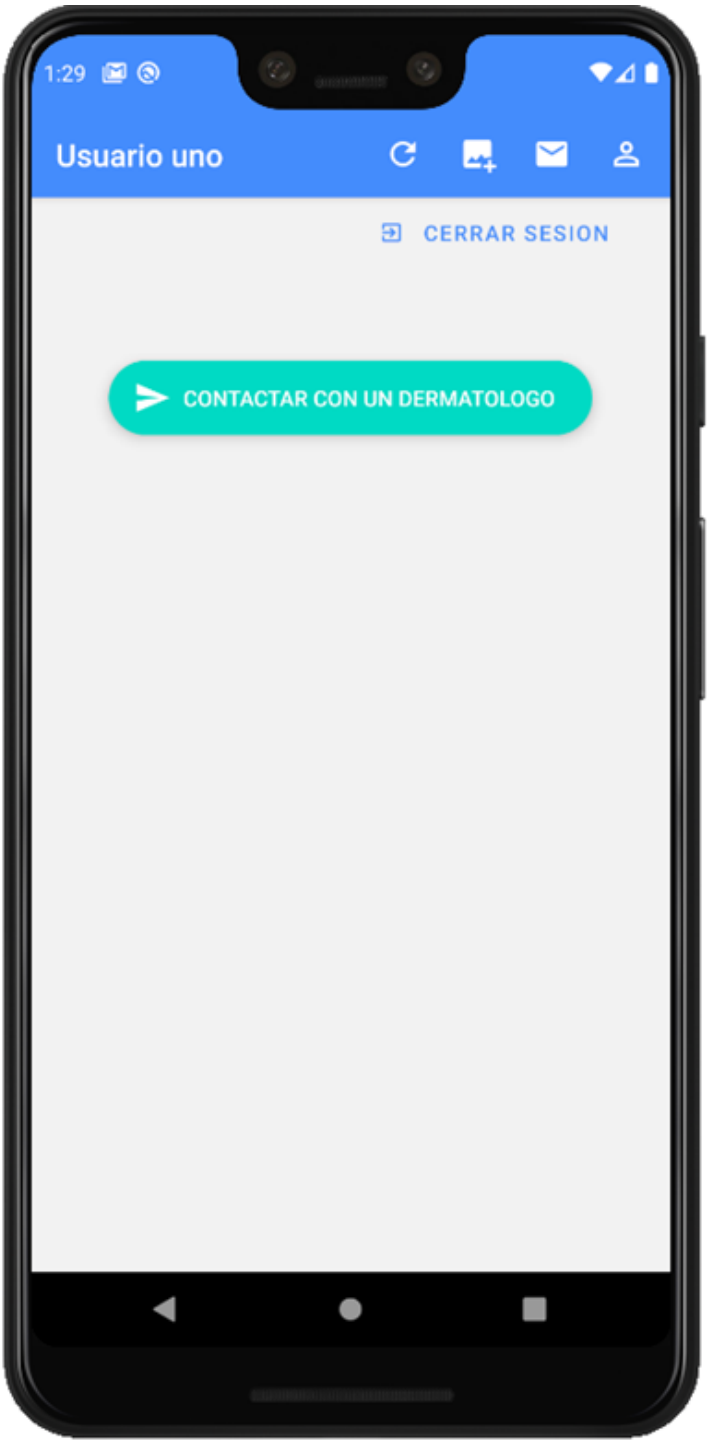
7. El usuario tendrá la siguiente interfaz para ingresar una nueva contraseña.



8. Cuando el usuario ha verificado su correo electrónico, e inicia sesión el sistema lo dirige a un formulario donde debe completar su registro.



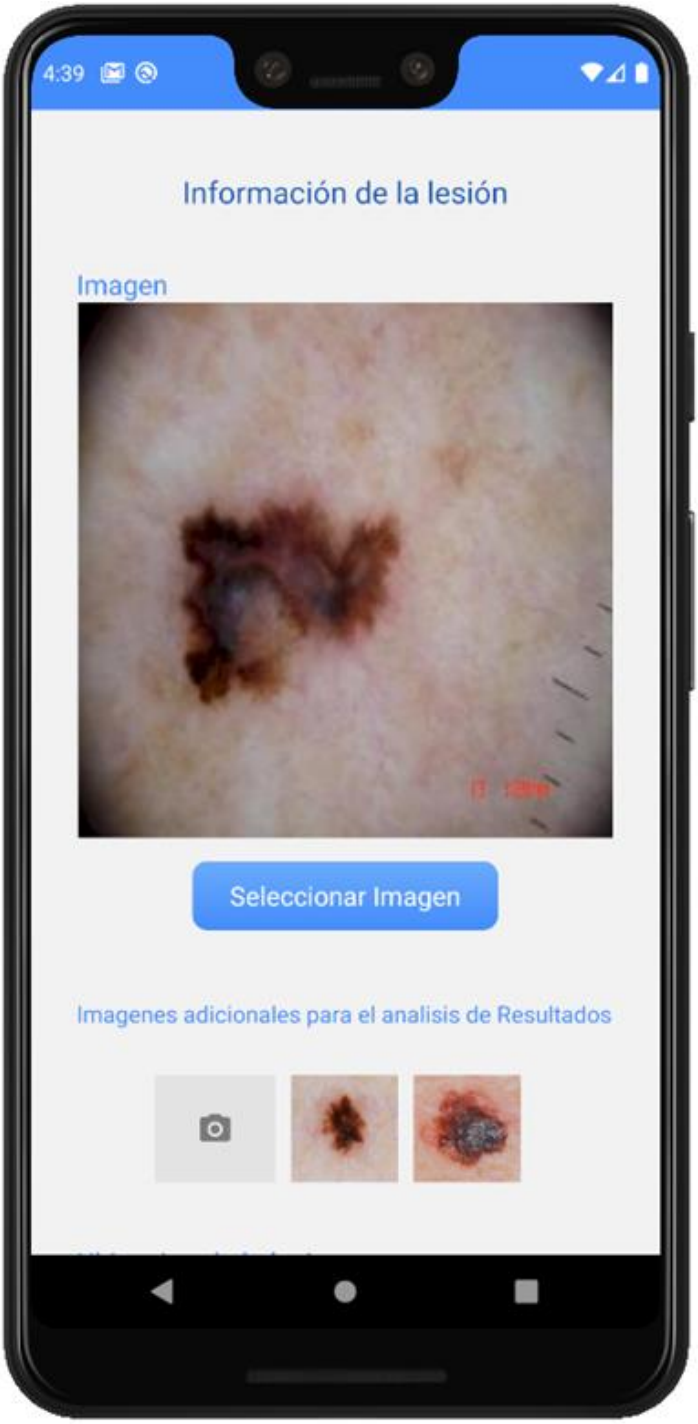
9. Cuando el usuario ya ha completado su registro, el sistema lo dirige a la pantalla principal. Al dar clic en el botón Contactar dermatólogo el usuario se enlistará en los Usuarios de pago



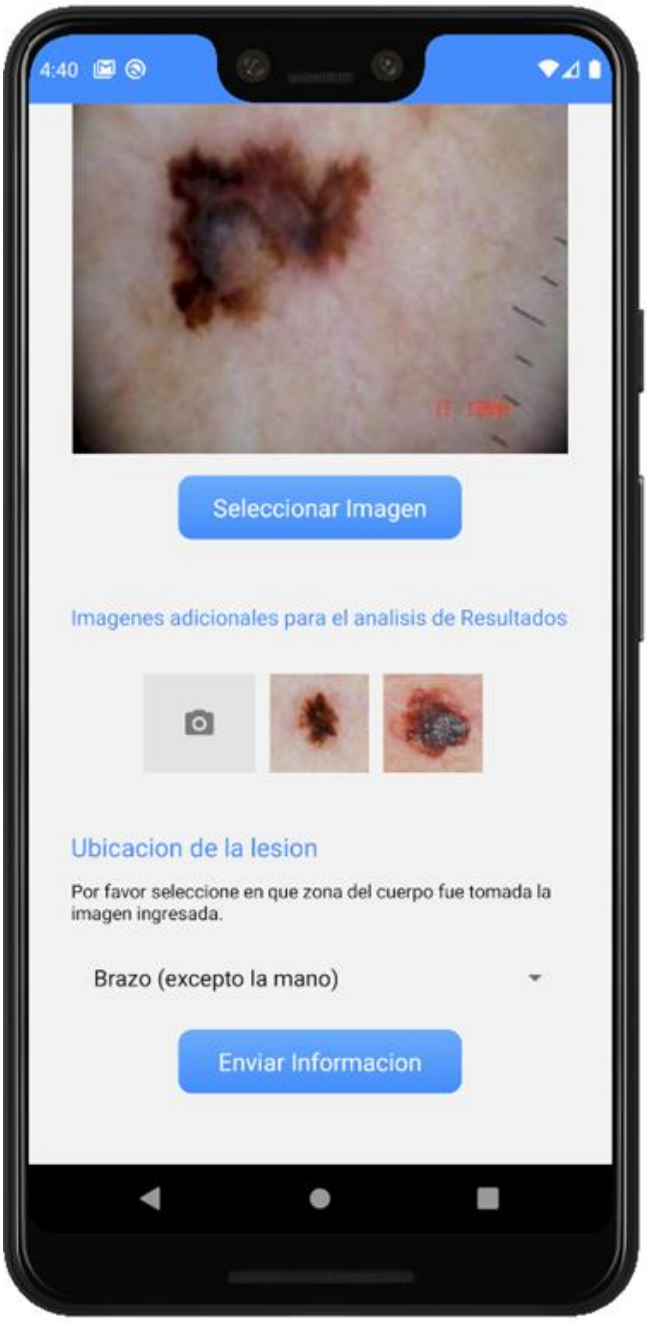
10. Cuando el usuario da clic en el siguiente icono, el sistema mostrará la pantalla para agregar una nueva clasificación o resultado.

11. Cuando el usuario da clic en seleccionar una imagen y adiciona imágenes para el análisis, el sistema lo enviará a la galería de su dispositivo.

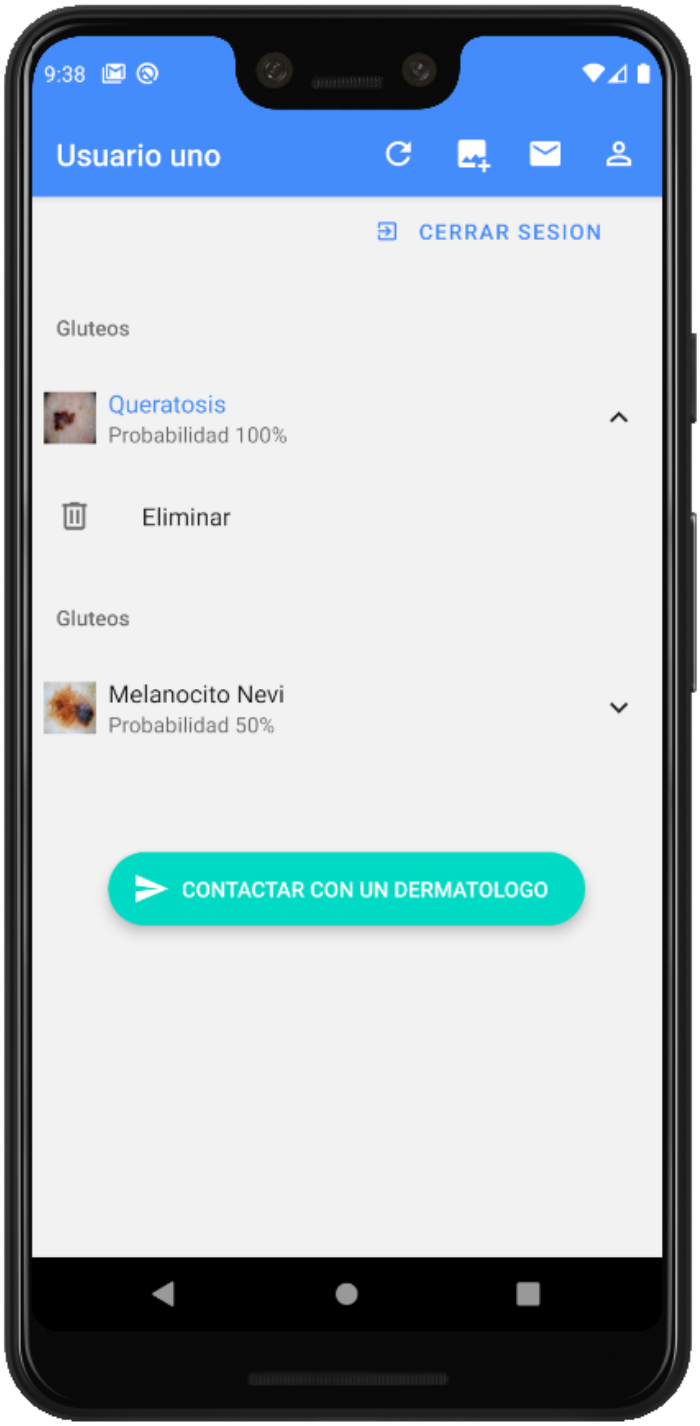
12. El sistema mostrará en la pantalla las imágenes que han sido seleccionadas por el usuario.



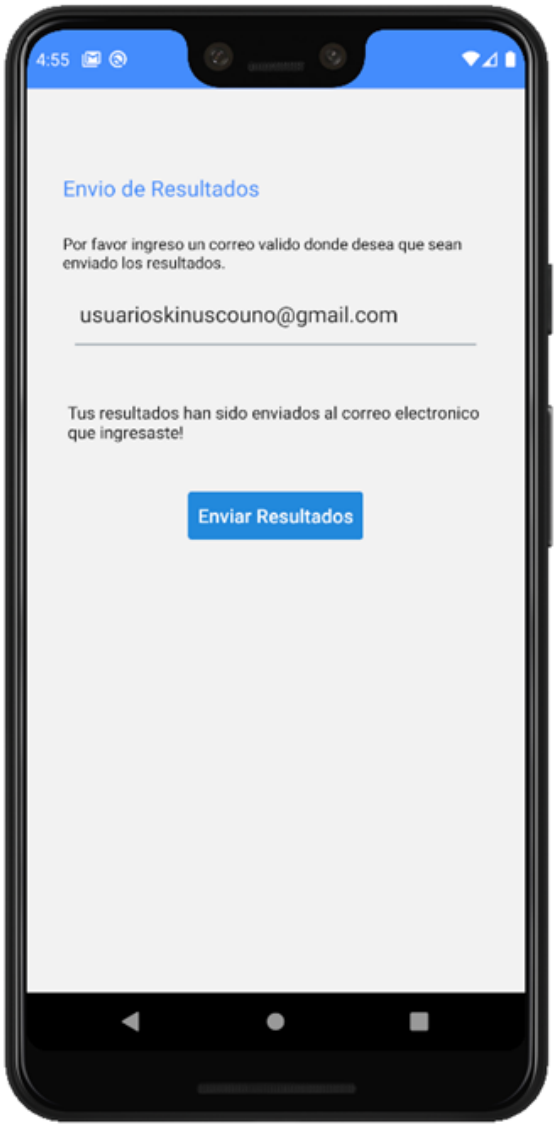
13. El usuario deberá seleccionar la zona de su cuerpo donde se encuentra la lesión.



14. Cuando el usuario da clic en Enviar Información, el sistema lo dirige a la pantalla principal donde en lista el resultado ingresado, con la opción de poder eliminar cada resultado.

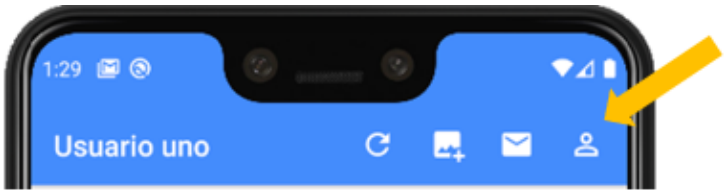


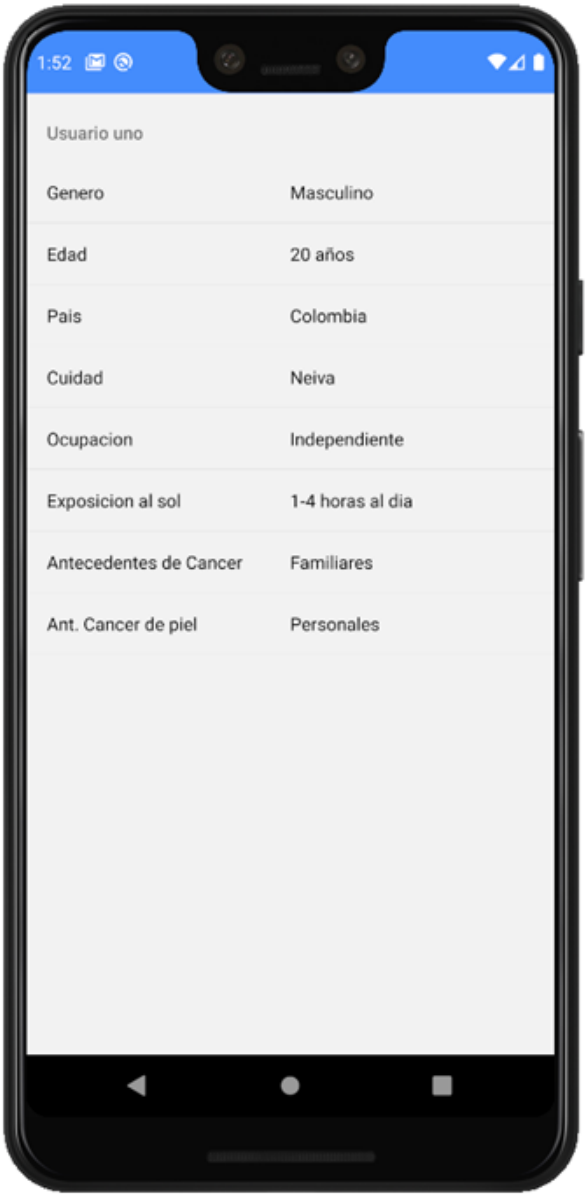
15. Cuando el usuario desee tener un resumen o reporte de los resultados que ha obtenido, dará clic al siguiente icono, donde lo enviará a la pantalla que enviará un PDF al correo electrónico que se ingrese.



16. El usuario recibe su correo electrónico.

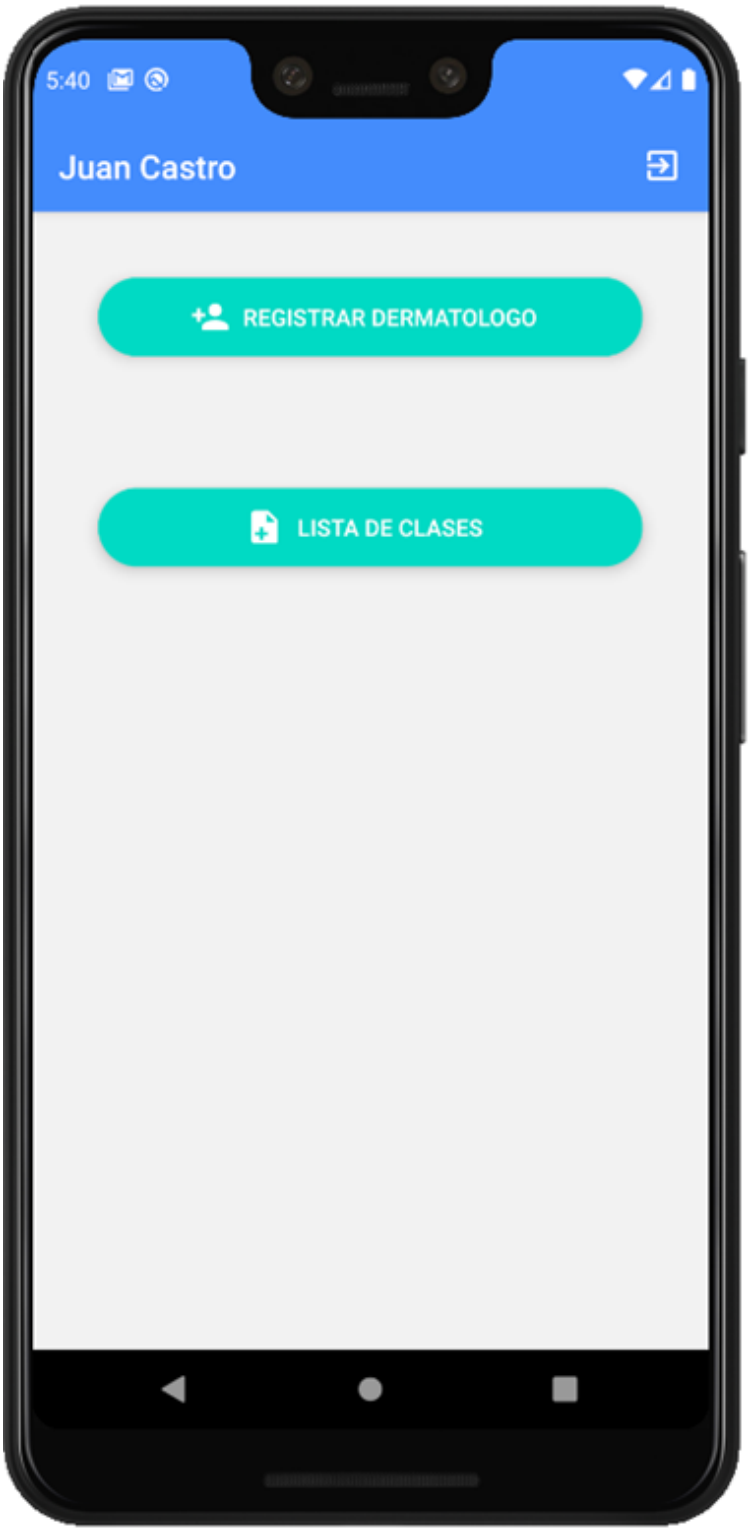
17. Cuando el usuario da clic en el siguiente icono, el sistema mostrará la información personal que ha registrado.



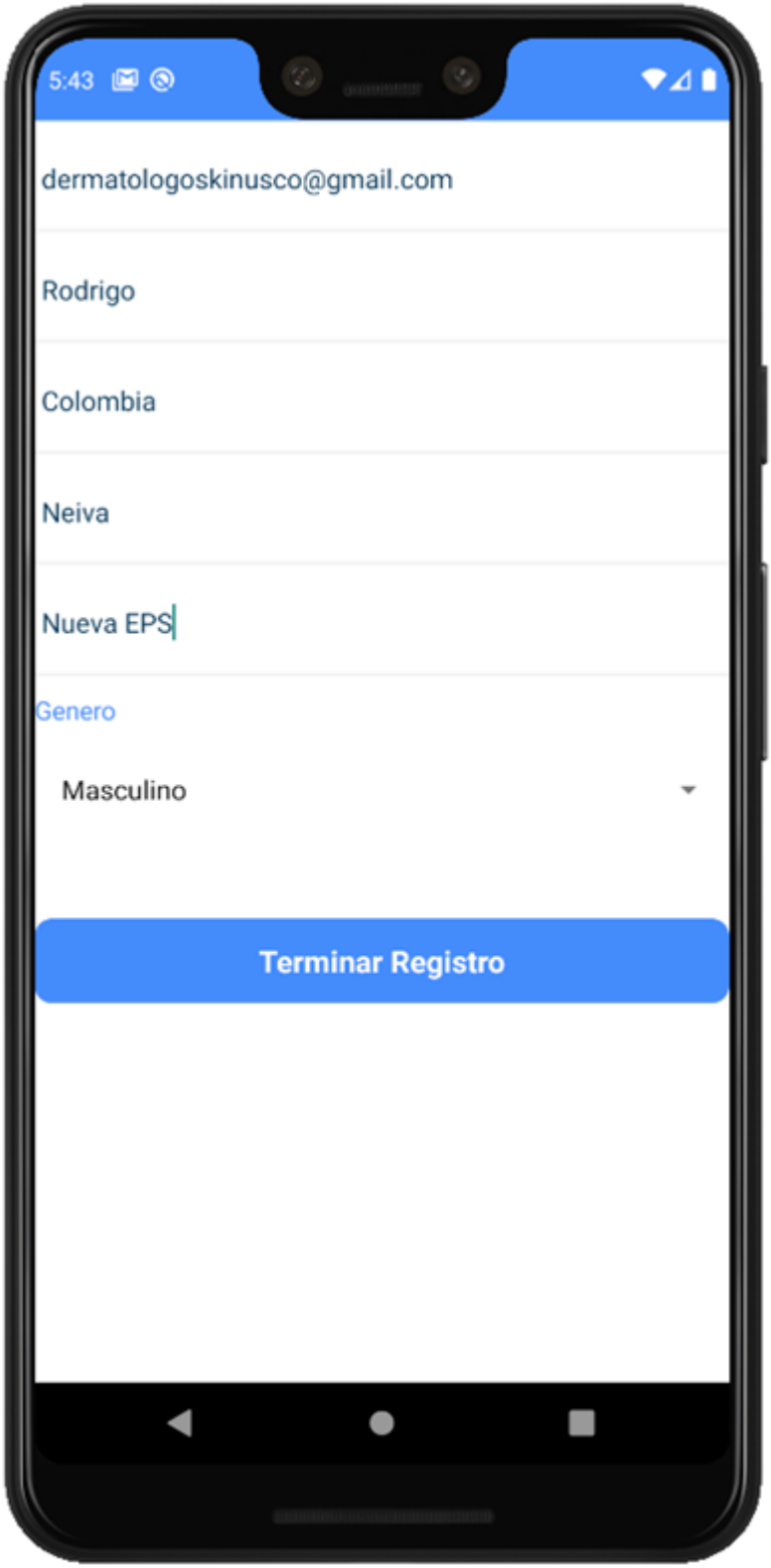


18. Tenemos un administrador del aplicativo que ingresa con la siguiente cuenta.

19. Pantalla principal del administrador



20. Registrar Dermatólogo, por parte del Administrador.



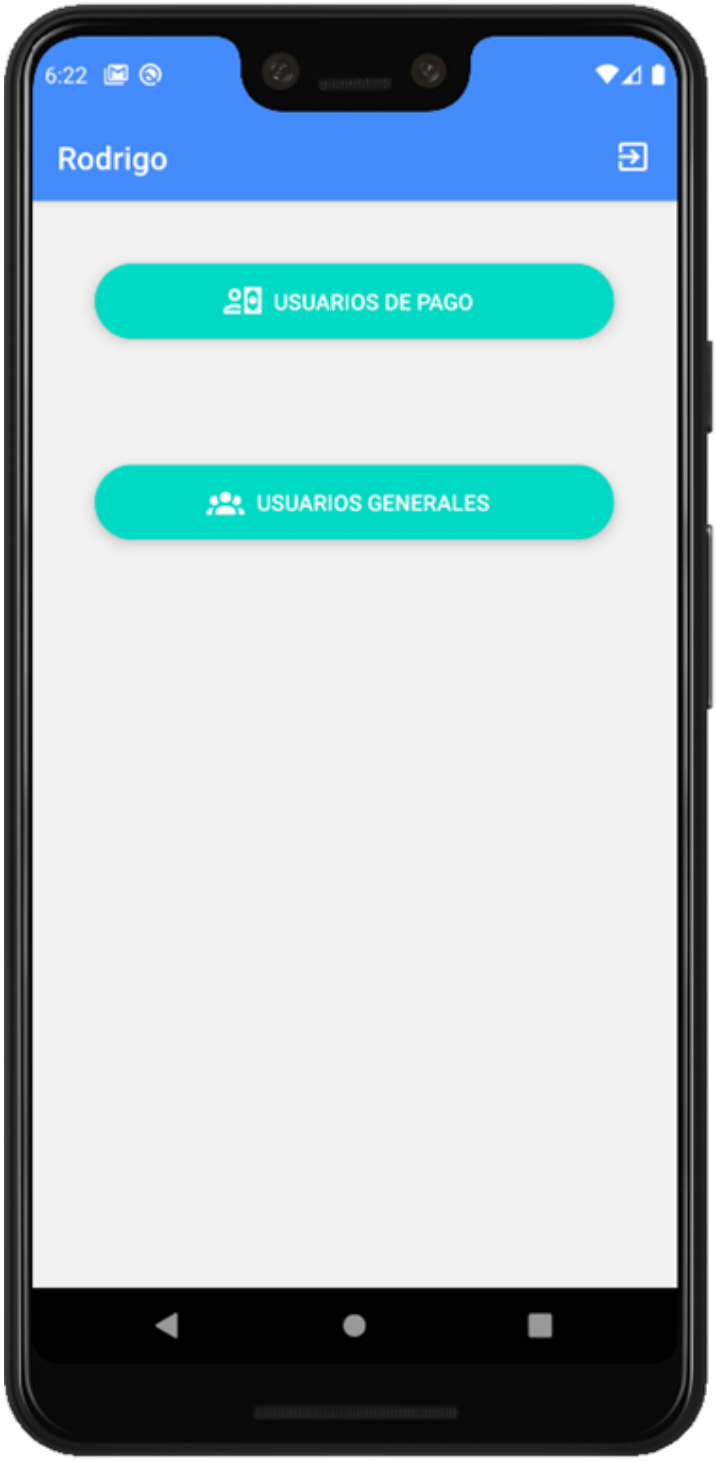
21. Lista de elementos existentes.

22. Agregar o Eliminar las clases que maneja el modelo por parte del Administrador


5:46
Lista de Clases
Nueva clase
Agregar
QUERATOSIS
Eliminar Imagen
¿Estas seguro que quieres este elemento?
NO
SI

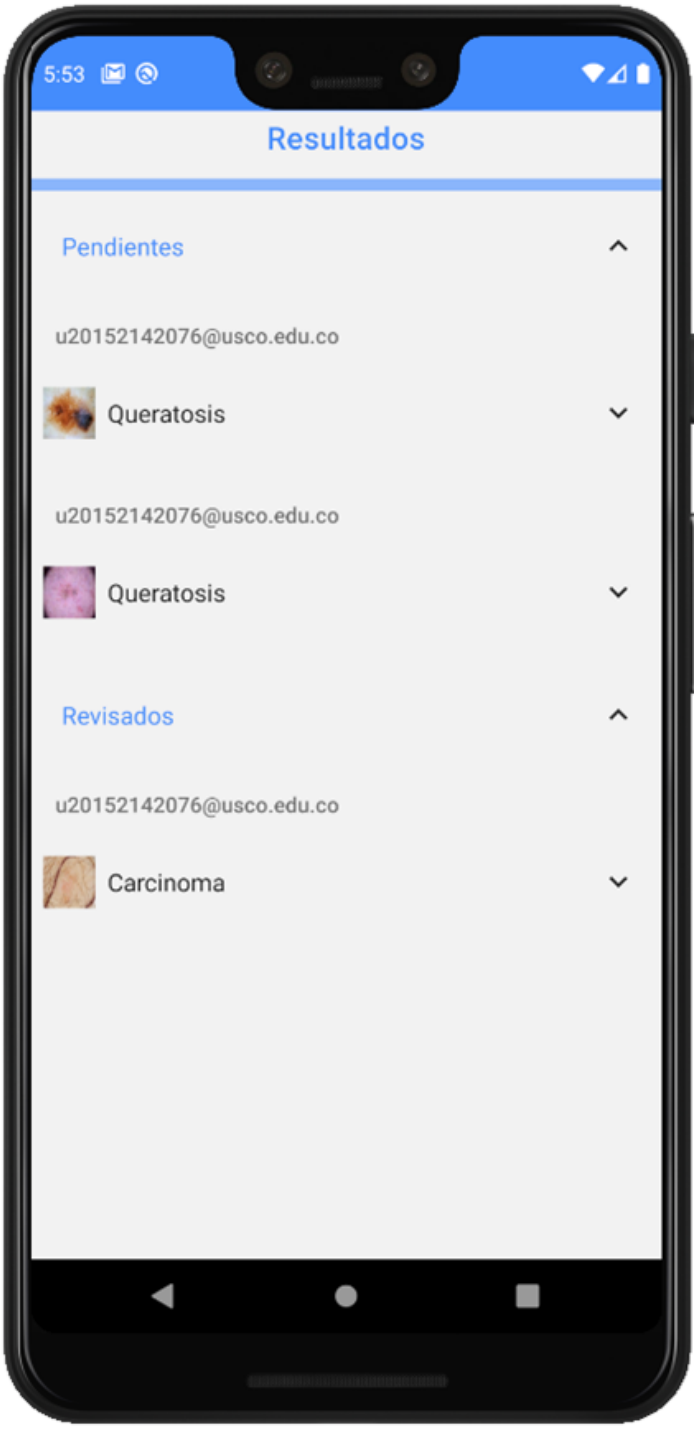
23. Ingresó como dermatólogo. dermatólogo que no ha verificado su correo mediante el enlace de Firebase.

24. Cuando el dermatólogo haya verificado su correo electrónico, ingresará en la pantalla principal.

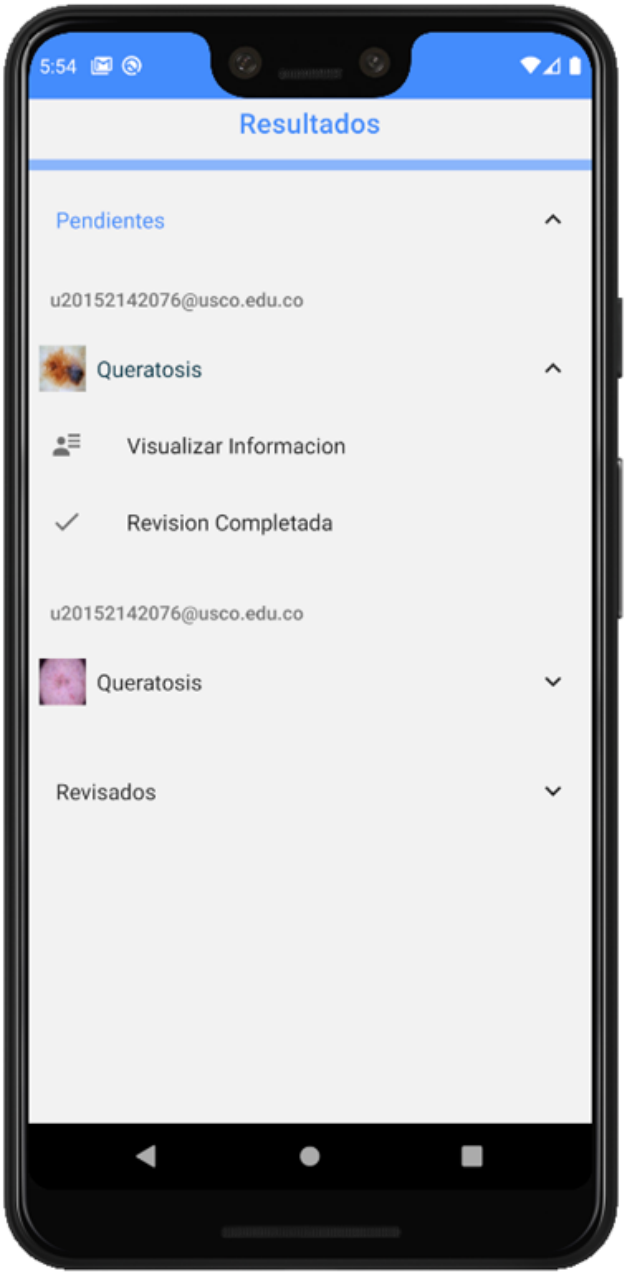


25. Usuario de pago, tenemos una lista de los usuarios que han decidido contactar con los servicios de revisión por parte del dermatólogo. Con las opciones de ver los resultados que tiene cada usuario y de marcar al usuario como revisado.

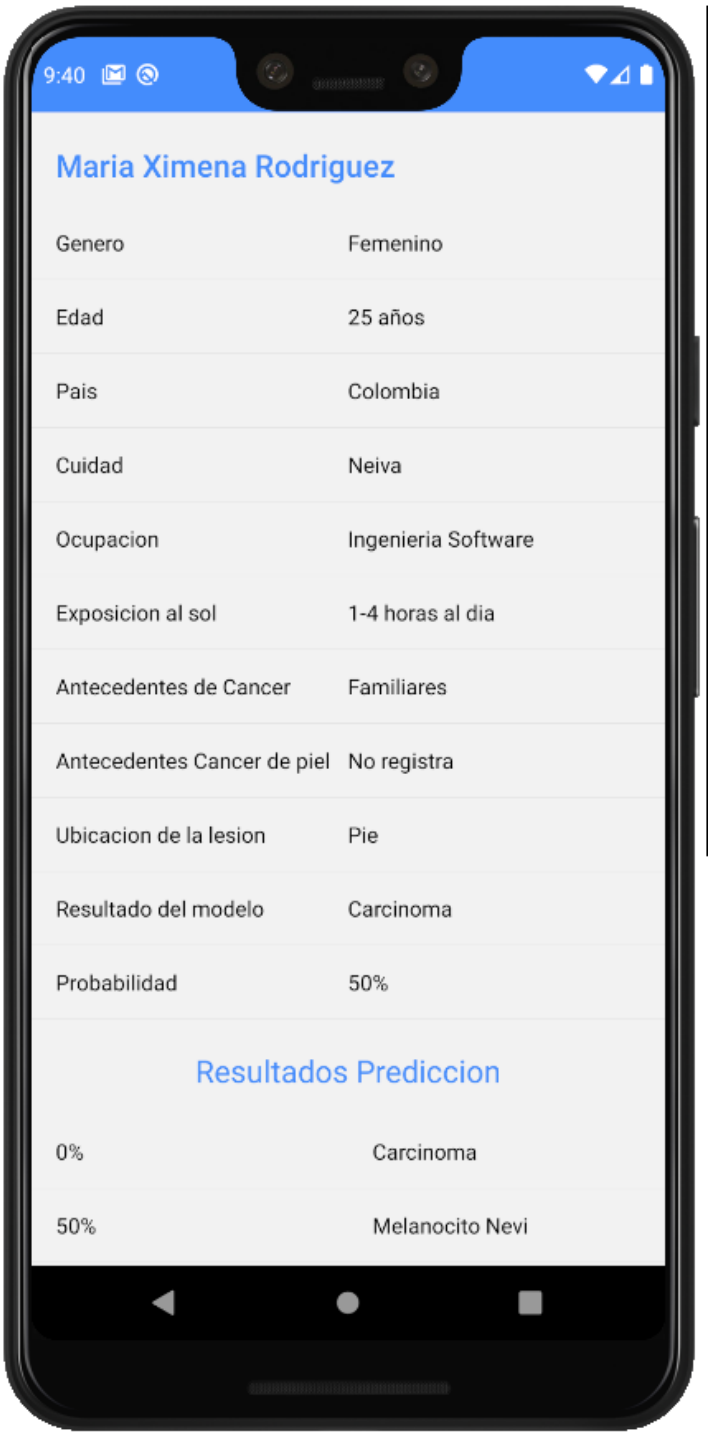
26. Ver resultados, en esta pantalla visualizamos dos secciones, los resultados que están pendientes por revisión y los resultados que están revisados.

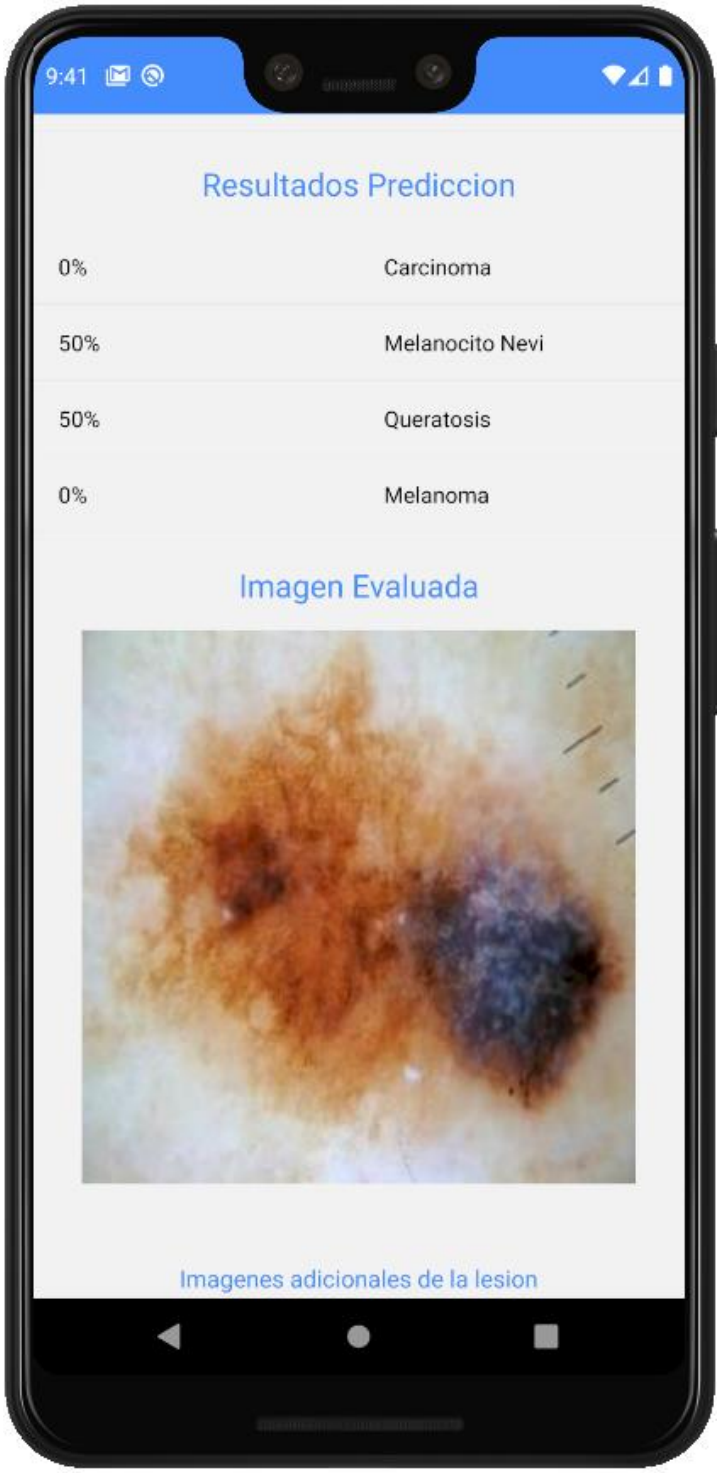


27. Cada resultado del usuario tiene las opciones de visualizar información y de marcar como revisión completada.

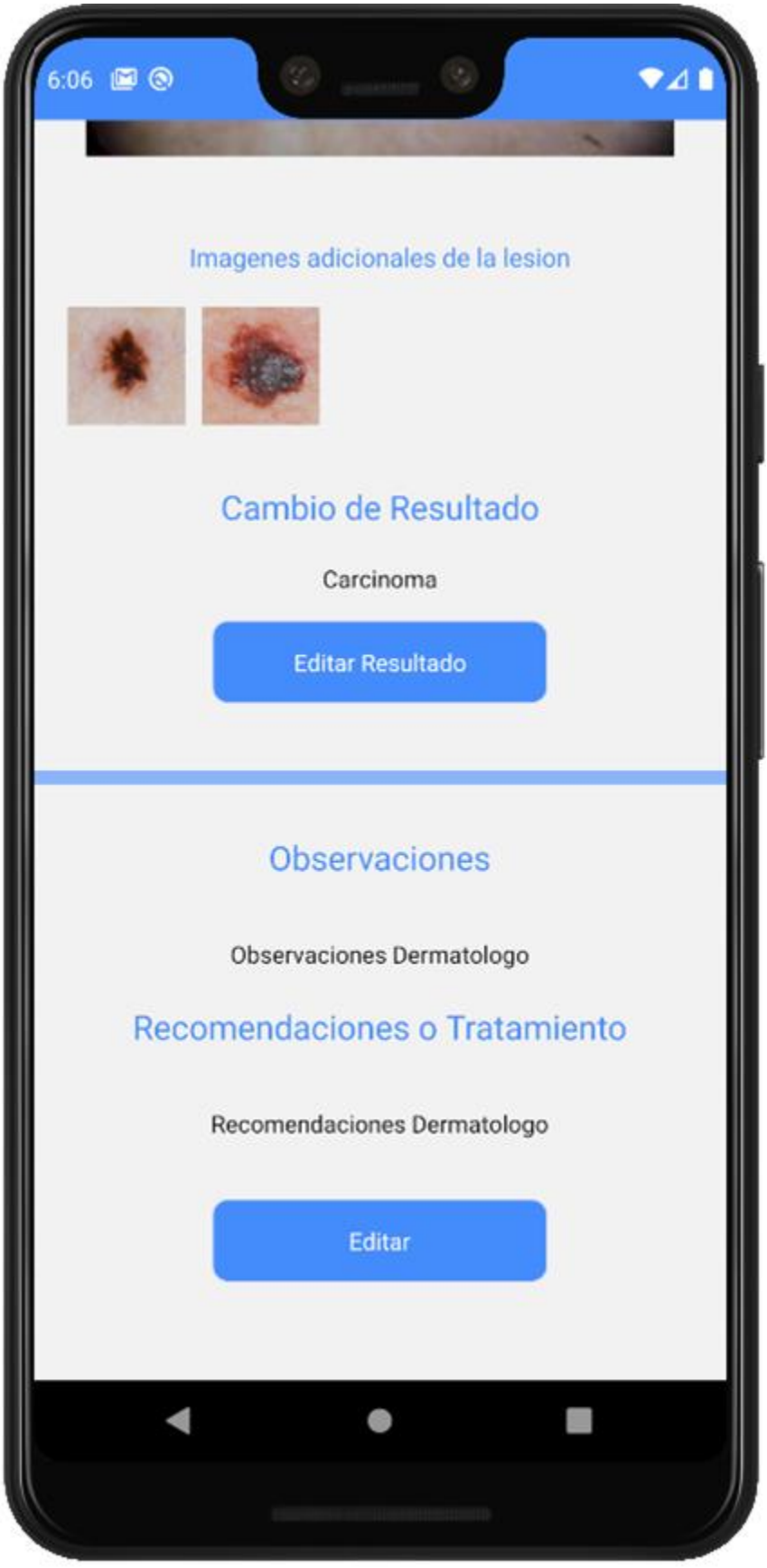


28. La información que se muestra de cada resultado es la siguiente.



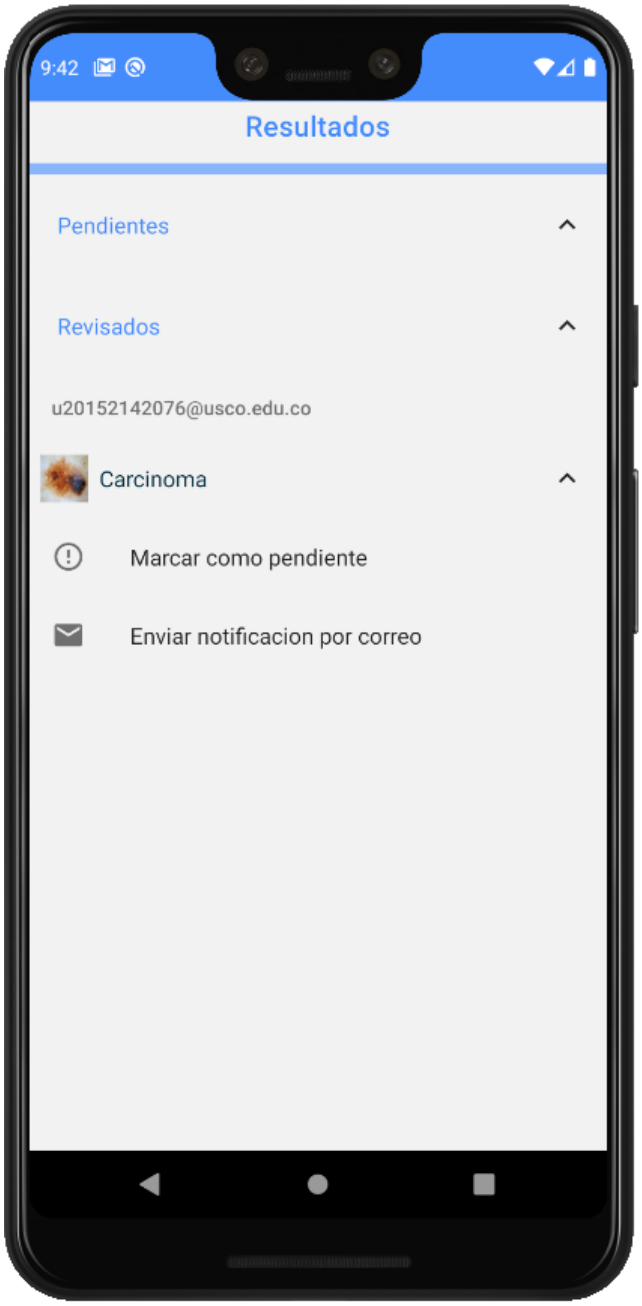

9:41
Resultados Prediccion
0%
Carcinoma
50%
Melanocito Nevi
50%
Queratosis
0%
Melanoma
Imagen Evaluada
Imagenes adicionales de la lesion

29. En esta sección de la información del resultado, el dermatólogo tiene las opciones de cambiar el resultado que se obtuvo del modelo, como también de agregar Observaciones, Recomendaciones o Tratamiento según sea la lesión y el caso.


6:06
Imagenes adicionales de la lesion
Cambio de Resultado
Carcinoma
Editar Resultado
Observaciones
Observaciones Dermatologo
Recomendaciones o Tratamiento
Recomendaciones Dermatologo
Editar

30. También tenemos las mismas opciones y estructura para evaluar a los usuarios generales, que no han solicitado valoración con el dermatólogo. El dermatólogo deberá analizar también a estos pacientes con el fin de obtener información para reentrenar el modelo

31. Cuando el resultado de un usuario de pago o general ya sea revisado y se encuentre en esta sección de la lista, el dermatólogo tendrá la opción de marcarlo como pendiente nuevamente, o de enviar un PDF al correo electrónico del usuario con los cambios y opiniones hechas por el dermatólogo.



32. Si realizamos el envío del PDF, de este resultado en específico obtenemos lo siguiente.

Nombre: Maria Ximena Rodriguez Genero: Femenino Edad: 25 años

Fecha: 2021-10-20 03:31

Resultado del Modelo

Zona	Clasificacion	Probabilidad	Imagen
Gluteos	Melanocito Nevi	50%	

Valoracion del Dermatologo :

Resultado : (Valoracion Actualizada)

Observaciones : no hay observaciones

Recomendaciones : no hay recomendaciones

Resultado de la clasificacion :

Carcinoma 0%

Melanocito Nevi 50%

Queratosis 50%

Melanoma 0%

proyectoskinusco@gmail.com Universidad Surcolombiana

16.3.3 Código

```
//Librerias
import React from 'react'
import {
  View,
  Text,
  TouchableOpacity,
  Dimensions,
  StyleSheet,
} from 'react-native'
import * as Animatable from 'react-native-animatable'
import LinearGradient from 'react-native-linear-gradient'
import { useNavigation } from '@react-navigation/native'

//Vista de Bienvenida
const Inicio = () => {
  const navigation = useNavigation()

  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <Animatable.Image
          animation="bounceIn"
          duraton="2500"
          source={require('../static/img/logoq.png')}
          style={styles.logo}
          resizeMode="stretch"
        />
      </View>
      <Animatable.View
        style={{ backgroundColor: '#448cfc', height: 200 }}
        animation="fadeInDown"
      >
        <Text style={[styles.title, { color: '#FFFFFF' }]}>
          ¡Analiza las imagenes de tu piel junto a nosotros!
        </Text>
        <View style={styles.button}>
          <TouchableOpacity
            onPress={() => navigation.navigate('InicioUsuario')}
            style={[
              styles.signP,
              { borderColor: '#e3e4f4', borderWidth: 1, marginTop: 10 },
            ]}
          >
            <LinearGradient
              colors={['#e3e4f4', '#e3e4f4']}
              style={styles.signP}
            >
              <Text style={[styles.textSign, { color: '#005FEF' }]}>
                Ingresar
              </Text>
            </LinearGradient>
          </TouchableOpacity>
        </View>
      </Animatable.View>
    </View>
  )
}
export default Inicio

```

```
const {height} = Dimensions.get("screen");
const height_logo = height * 0.28;

const styles = StyleSheet.create({
    container: {
        flex: 1,
        backgroundColor: '#FFFFFF'
    },
    header: {
        flex: 2,
        justifyContent: 'center',
        alignItems: 'center'
    },
    logo: {
        width: height_logo,
        height: height_logo,
        marginTop: 60
    },
    title: {
        color: '#FFFFFF',
        fontSize: 20,
        fontWeight: 'bold',
        alignSelf:'center',
        padding: 20

    },
    button: {
        alignItems: 'center',
        marginBottom: 5
    },
    signP: {
        width: 200,
        height: 50,
        justifyContent: 'center',
        alignItems: 'center',
        borderRadius: 10
    },
    textSign: {
        color: '#448cfc',
        fontWeight: 'bold',
        fontSize: 20
    },
    buttonin: {
        alignItems: 'center',
        flexDirection: 'column',
        marginBottom: 30
    }
})
```

Inicio Usuario.js (Script de acceso a Registro y Login)

```
//Librerias
import React from 'react'
import { View, Text, TouchableOpacity, StyleSheet } from 'react-native'
import * as Animatable from 'react-native-animatable'
import LinearGradient from 'react-native-linear-gradient'
import { useNavigation } from '@react-navigation/native'

const InicioUsuario = () => {
  const navigation = useNavigation()

  return (
    <View style={styles.container}>
      <Animatable.View
        style={{ backgroundColor: '#FFFFFF', marginTop: 250 }}
        animation="fadeInDown"
      >
        <Text style={[styles.text, { color: '#448cfc' }]}>
          ¡Si aun no tienes una cuenta, Registrate!
        </Text>
        <View style={styles.button}>
          <TouchableOpacity
            onPress={() => navigation.navigate('Registro')}
            style={[styles.signP, { borderColor: '#e3e4f4', borderWidth: 1 }]}
          >
            <LinearGradient
              colors={['#5494fc', '#5494fc']}
              style={styles.signP}
            >
              <Text style={[styles.textSign, { color: '#ffffff' }]}>
                Registro
              </Text>
            </LinearGradient>
          </TouchableOpacity>
        </View>
        <View
          style={{
            marginTop: 10,
            borderBottomColor: '#8ab5fb',
            borderBottomWidth: 8,
          }}
        />
        <Text style={styles.text}>¡Tienes una cuenta, Ingresa!</Text>
        <View style={styles.buttonin}>
          <TouchableOpacity
            onPress={() => navigation.navigate('Login')}
            style={[styles.signP, { borderColor: '#005FEF', borderWidth: 1 }]}
          >
            <LinearGradient
              colors={['#005FEF', '#005FEF']}
              style={styles.signP}
            >
              <Text style={[styles.textSign, { color: '#fff' }]}>
                Inicia Sesion
              </Text>
            </LinearGradient>
          </TouchableOpacity>
        </View>
      </Animatable.View>
    </View>
  )
}

export default InicioUsuario

```

```
//Estilos
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#448cfc',
  },
  text: {
    color: '#448cfc',
    marginTop: 0,
    padding: 20,
    flexDirection: 'column',
    fontSize: 17,
    alignSelf: 'center',
  },
  button: {
    alignItems: 'center',
    marginBottom: 5,
  },
  buttonin: {
    alignItems: 'center',
    flexDirection: 'column',
    marginBottom: 30,
  },
  signP: {
    width: 200,
    height: 50,
    justifyContent: 'center',
    alignItems: 'center',
    borderRadius: 10,
  },
  textSign: {
    color: '#448cfc',
    fontWeight: 'bold',
    fontSize: 20,
  },
})

```

Registro.js (Registro de Usuarios)

```
//Librerias
import React , {useState} from 'react';
import { View, Text, TouchableOpacity,TextInput,Platform,StyleSheet,StatusBar} from 'react-native';
import * as Animatable from 'react-native-animatable';
import LinearGradient from 'react-native-linear-gradient';
import firebase from '../firebase/fire';


//Registro de Usuarios
const Registro = ({navigation}) => {
    const [emailr, setEmailr] = useState('');
    const [contraseñar, setContraseñar] = useState('');
    const [errorreg, setError] = useState('');
    const [statedatos,setStatedatos] = useState({
        genero: "",
        edad:"",
        pais:"",
        nombre:"",
        cuidad:"",
        ocupacion:"",
        correo:"",
        cancer:"",
        hora:" ",
        piel:" ",
        done: false

    });

    const Registrar = async () => {

        try {
            //Crear usuario Firebase
            const response = await firebase.auth().createUserWithEmailAndPassword(emailr,
contraseñar);
            const correo = response.user;
            const key = response.user.uid;
            const bdReferencia = firebase.database().ref('/Usuarios/'+key)
            bdReferencia.set({
                cuenta:emailr,
                keycuenta:key,
                datospersonales:statedatos,
                estadousuario: true,
                estadodermatologo: false,

            })
            // Enviar enlace de Verificacion
            correo.sendEmailVerification().then(function(){
                console.log('Correo de vrificacion enviado')
            }).catch(function(error){
                console.log('Correo de verificacion no enviado')
                console.log(error)
            })
            console.log('Se registro correctamente')
            console.log(emailr)
            navigation.navigate('Login')
            // Errores de Autenticacion
        } catch (err) {
            if (err.code === 'auth/invalid-email'){
                let mensaje= 'La direccion de correo electronico tiene un formato incorrecto'
                setError(mensaje);
            } else if (err.code === 'auth/email-already-in-use') {
                let mensaje= 'La direccion de correo electronico ya se encuentra registrada'
                setError(mensaje);
            }
            else if (err.code === 'auth/weak-password'){
                let mensaje= 'La contraseña debe tener al menos 6 caracteres'
                setError(mensaje);
            }
        }

    }

```

```
return  <>
    <View style={styles.container}>
        <StatusBar backgroundColor='#448cfc' barStyle="light-content"/>
    <View style={styles.header}>
        <Text style={styles.text_header}>Registrate Ahora!</Text>
    </View>
        <Animatable.View animation="fadeInUpBig"style={styles.footer}>
        <Text style={styles.text_footer} >Correo Electronico</Text>
    <View style={styles.action}>
        <TextInput placeholder="Ingresa un usuario" style={styles.text_footer}
autoCapitalize="none" value={emailr} onChangeText={setEmailr} keyboardType='email-address'/>
    </View>
        <Text style={[styles.text_footer, {marginTop: 35}]}>Contraseña</Text>
    <View style={styles.action}>
        <TextInput placeholder="Ingresa una contraseña" style={styles.textInput}
autoCapitalize="none" value={contraseñar} onChangeText={setContraseñar} secureTextEntry/>
    </View>
        {
         errorreg ?
        <Text style={{ color: 'red' }}>{errorreg}</Text>
        : null
        }

    <View style={styles.button}>
        <TouchableOpacity style={styles.signIn} onPress={() => Registrar()} >
            <LinearGradient colors={['#448cfc', '#448cfc']} style={styles.signIn}>
                <Text style={[styles.textSign, {color:'#fff'}]}>Terminar Registro</Text>
            </LinearGradient>
        </TouchableOpacity>
    </View>
        </Animatable.View>
  </View>

</>

}

export default Registro;
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#448cfc',
  },
  footer: {
    flex: Platform.OS === 'ios' ? 3 : 5,
    backgroundColor: '#fff',
    borderTopLeftRadius: 30,
    borderTopRightRadius: 30,
    paddingHorizontal: 20,
    paddingVertical: 30,
  },
  header: {
    flex: 1,
    justifyContent: 'flex-end',
    paddingHorizontal: 20,
    paddingBottom: 50,
  },
  text_header: {
    color: '#fff',
    fontWeight: 'bold',
    fontSize: 30,
  },
  text_footer: {
    color: '#05375a',
    fontSize: 16,
  },
  action: {
    flexDirection: 'row',
    marginTop: 10,
    borderBottomWidth: 2,
    borderBottomColor: '#f2f2f2',
    paddingBottom: 5,
  },
  textInput: {
    flex: 1,
    marginTop: Platform.OS === 'ios' ? 0 : -12,
    paddingLeft: 10,
    color: '#05375a',
  },
  button: {
    alignItems: 'center',
    marginTop: 50,
  },
  signIn: {
    width: '100%',
    height: 50,
    justifyContent: 'center',
    alignItems: 'center',
    borderRadius: 10,
  },
  textSign: {
    fontSize: 18,
    fontWeight: 'bold',
  },
})

```

DatosRegistro.js (Formulario de Registro)

```
//Librerias
import React , {useState} from 'react';
import {  View, Text, TouchableOpacity,  StyleSheet ,ScrollView, } from 'react-native';
import { Input} from 'react-native-elements';
import LinearGradient from 'react-native-linear-gradient';
import RNPickerSelect from 'react-native-picker-select';
import firebase from '../firebase/fire';
import { connect } from 'react-redux';
import {useNavigation} from '@react-navigation/native'


//Registro de Usuarios
const DatosPersonales = ({keyg}) => {
    const navigation = useNavigation()
    const itemRef = firebase.database().ref('/Usuarios/'+keyg)
    function actualizar(){
      //Guardar informacion
      itemRef.update({
        datospersonales: {genero:genero,
          edad: state.edad,
          nombre:nombre,
          pais: pais,
          cuidad: cuidad,
          ocupacion:ocupacion,
          hora: hora,
          cancer:cancer,
          piel:piel,
          done: true,
          },
          rol: 1,

      })

  }

      const [ nombre, setNombre ] = useState("");
      const [ cuidad, setCuidad ] = useState("");
      const [ ocupacion, setOcupacion ] = useState("");
      const [ genero, setGenero ] = useState("");
      const [ cancer, setCancer ] = useState("");
      const [ piel, setPiel ] = useState("");
      const [ hora, setHora ] = useState("");
      const [ pais, setPais ] = useState("");
      const [state,setState] = useState({
        genero: "",
        edad:"",
        pais:"",
        nombre:"",
        cuidad:"",
        ocupacion:"",
        correo:"",
        done: true

    });


      //Cambiar la edad a numero
      const onChanged = (edad) => {
        let newEdad = '';
        let numbers = '0123456789';

        for (var i=0; i < edad.length; i++) {
            if(numbers.indexOf(edad[i]) > -1 ) {
                newEdad = newEdad + edad[i];

            }
            else {
             alert("solo numeros");
            }
        }
      setState({ edad: newEdad });
      };
```

```
1
2 //Verificar campos vacios
3 const AddUser = async () => {
4   typeof state.edad
5   if(state.edad  === ''){
6       alert('Por favor ingrese su edad')
7   } else if (pais === '') {
8       alert('Por favor ingrese el pais')
9   } else if (nombre === ''){
10      alert('Por favor nombre')
11  } else if (cuidad === ''){
12    alert('Por favor ingrese la cuidad')
13  } else if (ocupacion === ''){
14    alert('Por favor ingrese su ocupacion')
15  }
16  else if (genero=== '' || genero === 'no'){
17    alert('Por favor seleeccione su genero')
18  }
19  else if (cancer=== '' || cancer === 'no'){
20    alert('Por favor seleeccione los antecedentes de cancer')
21  }
22  else if (piel=== '' || piel === 'no'){
23    alert('Por favor seleeccione los antecedentes de cancer de piel')
24  }
25  else if (hora=== '' || hora === 'no'){
26    alert('Por favor ingrese la cantidad de horas de exposicion al sol')
27  }
28
29    actualizar()
30    navigation.navigate('Login')
31    console.log('Se guarda correctamente en Firebase')
32
33 };
34
35
36
37      return (
38          <ScrollView>
39
40          <View style={styles.container}>
41          <Text  style={{fontSize: 20  , color: '#1C5BB7' , marginBottom:25 , fontStyle: 'normal'}}>Datos
    Personales</Text>
42          <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf: 'baseline', marginTop:5}}>
    Nombre</Text>
43           <Input
44           keyboardType='default'
45           onChangeText={(nombre) => setNombre(nombre)}
46          />
47
48          <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf: 'baseline'}}>Genero</Text>
49          <RNPickerSelect
50                  placeholder={{ }}
51                  onValueChange={(genero) => setGenero(genero)}
52                  items={[
53                      {label:"Genero", value:'no'},
54                      { label: "Masculino", value: "Masculino" },
55                      { label: "Femenino", value: "Femenino" },
56                  ]}
57              />
58
59          <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf: 'baseline', marginTop:5}}>Edad</Text>
60          <Input
61          keyboardType='numeric'
62          onChangeText={(edad)=>onChanged(edad)}
63          placeholder='Número'
64          maxLength={2}
65          />
```

```
   <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf: 'baseline', marginTop:5}}> Pais</Text>
            <Input
            onChangeText={(pais) => setPais(pais)}
           />
           <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf: 'baseline',
marginTop:5}}>Ciudad</Text>
            <Input
            onChangeText={(cuidad) => setCuidad(cuidad)}
           />
           <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf: 'baseline',
marginTop:5}}>Ocupacion</Text>
            <Input
              onChangeText={(ocupacion) => setOcupacion(ocupacion)}
           />
            <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf: 'baseline'}}>Antecedentes de
Cancer</Text>
           <RNPickerSelect
                   placeholder={{ }}
                   onValueChange={(cancer) => setCancer(cancer)}
                   items={[
                       {label:"Antecedentes de Cancer", value:'no'},
                       {label:"Familiares", value:"Familiares"},
                       { label: "Personales", value: "Personales" },
                       { label: "No registra antecedentes" , value:"No registra"}
                   ]}
               />

           <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf: 'baseline'}}>Antecedentes de Cancer de
Piel</Text>
           <RNPickerSelect
                   placeholder={{ }}
                   onValueChange={(piel) => setPiel(piel)}
                   items={[
                     {label:"Antecedentes de Cancer de Piel", value:'no'},
                     {label:"Familiares", value:" Familiares"},
                     { label: "Personales", value: "Personales" },
                     { label: "No registra antecedentes" , value:"No registra"}
                 ]}
                />

        <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf: 'baseline'}}>Cantidad de horas en qué se
expone al sol por día</Text>
           <RNPickerSelect
                   placeholder={{ }}
                   onValueChange={(hora) => setHora(hora)}
                   items={[

                     {label:"Horas al sol", value:'no'},
                     { label: "Menos de 1 hora" , value:"Menos de 1 hora"},
                     {label:"1-4 horas", value:"1-4 horas"},
                     { label: "4-8 horas", value: "4-8 horas" },
                     { label: "8-12 horas" , value:"8-12 horas"},
                     { label: "No se expone al sol" , value:"No registra"}
                 ]}
                />


             <TouchableOpacity
             style={styles.button}
             onPress={AddUser}>
           <LinearGradient
                colors={['#6cacfc', '#448cfc']}
                style={styles.signIn}
             >
                 <Text style={[styles.textSign, {
                     color:'#fff'
                 }]}>Guardar</Text>
             </LinearGradient>
           </TouchableOpacity>

            </View>
           </ScrollView>


       );
   };
   //Redux
   const mapStateToProps = state => ({
       keyg:state.key

   })
   export default connect(mapStateToProps,{})(DatosPersonales) ;
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    padding: 30,
  },
  over: {
    height: 350,
    width: 350,
  },
  over1: {
    marginBottom: 20,
    marginTop: 20,
  },
  tinyLogo: {
    width: 300,
    height: 50,
  },
  buttonStyle: {
    backgroundColor: '#448cfc',
    borderWidth: 0,
    color: '#FFFFFF',
    borderColor: '#307ecc',
    height: 40,
    alignItems: 'center',
    borderRadius: 30,
    marginLeft: 35,
    marginRight: 35,
    marginTop: 15,
    width: 200,
  },
  image: {
    width: 400,
    height: 350,
    resizeMode: 'contain',
  },
  buttonTextStyle: {
    color: '#FFFFFF',
    paddingVertical: 10,
    fontSize: 16,
  },
  buttonR: {
    backgroundColor: 'darkblue',
    marginTop: 30,
  },
  button: {
    marginBottom: 20,
  },
  signIn: {
    width: 200,
    height: 45,
    justifyContent: 'center',
    alignItems: 'center',
    borderRadius: 10,
    marginTop: 10,
  },
  textSign: {
    fontSize: 17,
  },
})

```

Login.js (Inicio de Sesión)

```
//Librerias
import React , {useState} from 'react';
import { View, Text, TouchableOpacity, Dimensions,TextInput,Platform,StyleSheet,StatusBar,} from
'react-native';
import * as Animatable from 'react-native-animatable';
import LinearGradient from 'react-native-linear-gradient';
import { Button } from 'react-native-paper';
import firebase from '../firebase/fire';
import {useNavigation} from '@react-navigation/native'
import { connect } from 'react-redux';


//Login
const LoginUsuario = ({agregarinformacion,agregardatos,agregardermatologo}) =>{
    const [email, setEmail] = useState('');
    const [contraseña, setContraseña] = useState('');
    const [error, setError] = useState('');
    const navigation = useNavigation()


    //Validacion del Rol
    const Ingresar = async () => {

        function Rol (id){
            const bd = firebase.database().ref('/Usuarios/'+id)
            bd.on('value', function(snap){
                let loginus = snap.val();
                const rolusuario = loginus.rol
                const usuario = loginus.datospersonales
                const key = id
                agregarinformacion(key)
                switch (rolusuario) {
                    case 1:
                        console.log('Rol Usuario')
                        agregardatos(usuario)
                        navigation.navigate('Usuario')
                        bd.off();
                      break;
                    case 2:
                        console.log('Rol Dermatologo')
                        agregardermatologo(usuario)
                        navigation.navigate('Dermatologo')
                        bd.off();
                      break;
                    case 3:
                        console.log('Rol Administrador')
                        navigation.navigate('Administrador')
                        bd.off();
                      break;
                    default:
                        navigation.navigate('DatosRegistro')
                        bd.off();
                      break;
                  }
            })
        }

```

```
        try {
            // Autenticacion de Usuarios - Envio de correo verificacion
            const responsel = await firebase.auth().signInWithEmailAndPassword(email, contraseña);
            const emailverificado = responsel.user.emailVerified
            if (emailverificado === false) {
                console.log('No tiene acceso')
                let mensaje= 'El Usuario no se encuentra verificado por favor consulte su correo'
                setError(mensaje);

            }
            else {
               console.log('Tiene Acceso')
               const id = responsel.user.uid
                Rol(id)

            }
            //Errores de Autenticacion
        } catch (error) {
            if (error.code === 'auth/user-not-found'){
                let mensaje= 'El Usuario no se encuentra registrado'
                setError(mensaje);
            } else if (error.code === 'auth/wrong-password'){
                let mensaje= 'La contraseña es incorrecta para el usuario ingresado'
                setError(mensaje);
            } else if (error.code === 'auth/invalid-email'){
                let mensaje= 'Por favor ingrese el email'
                setError(mensaje);
            }


        }
    }

    //Olvidar Contraseña
    const Contraseña = async () => {
        try {
            let res = await fetch(
              'http://34.132.148.15:9000/contra/',
              {
                method:'post',
                mode:'no-cors',
                headers: {
                  'Accept': 'application/json',
                  'Content-Type': 'application/json'
                },
                body: JSON.stringify({
                  correo: email

                })

              }
            );
            let ver = await res.json();
            setContra(ver);
            console.log('Respuesta de Fastapi Recuperar Contraseña')
            console.log(ver);
          } catch (err) {
              alert('Unknown Error: ' + JSON.stringify(err));
              throw err;
          }

    }
```

```
  return <>
   <View style={styles.container}>
           <StatusBar backgroundColor='#448cfc' barStyle="light-content"/>
       <View style={styles.header}>
           <Text style={styles.text_header}>Bienvenido!</Text>
       </View>
           <Animatable.View animation="fadeInUpBig"style={[styles.footer, {}]}>
           <Text style={[styles.text_footer, { }]}>Correo Electronico</Text>
       <View style={styles.action}>
           <TextInput placeholder="Usuario"placeholderTextColor="#666666"style={[styles.textInput, {}]}
               autoCapitalize="none"
               value={email}
               onChangeText={setEmail}
               keyboardType='email-address'
           />
       </View>
           <Text style={[styles.text_footer, {marginTop: 35}]}>Contraseña</Text>
       <View style={styles.action}>
           <TextInput placeholder="Contraseña"placeholderTextColor="#666666" style={[styles.textInput, {}]}
               autoCapitalize="none"
               value={contraseña}
               onChangeText={setContraseña}
               secureTextEntry
           />
       </View>
           {
               error ?
               <Text style={{ color: 'red' }}>{error}</Text>
               : null
           }
           <Button icon="key"  mode="text" color='#448cfc' style={styles.contraseña} onPress={() =>
Contraseña()}>
               Olvidaste tu contraseña
           </Button>
       <View style={styles.button}>
           <TouchableOpacity style={styles.signIn} onPress={() => Ingresar()}>
               <LinearGradient colors={['#448cfc', '#448cfc']} style={styles.signIn}>
               <Text style={[styles.textSign, {color:'#fff'}]}>Ingresa</Text>
               </LinearGradient>
           </TouchableOpacity>
       </View>
           </Animatable.View>
   </View>
 </>
};



//Redux
const mapStateToProps = state => ({

})
const mapDispatchToProps = dispatch => ({
    agregarinformacion(key){
        dispatch({
            type:'Agregar Informacion',
            key
        })
    },
    agregardatos(usuario){
        dispatch({
            type:'Agregar datos',
            usuario
        })
    },
    agregardermatologo(usuario){
        dispatch({
            type:'Agregar dermatologo',
            usuario
        })
    }
})

export default connect(mapStateToProps,mapDispatchToProps)(LoginUsuario);
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#448cfc',
  },
  header: {
    flex: 1,
    justifyContent: 'flex-end',
    paddingHorizontal: 20,
    paddingBottom: 50,
  },
  footer: {
    flex: 3,
    backgroundColor: '#fff',
    borderTopLeftRadius: 30,
    borderTopRightRadius: 30,
    paddingHorizontal: 20,
    paddingVertical: 30,
  },
  text_footer: {
    color: '#05375a',
    fontSize: 18,
  },
  action: {
    flexDirection: 'row',
    marginTop: 10,
    borderBottomWidth: 1,
    borderBottomColor: '#f2f2f2',
    paddingBottom: 5,
  },
  contraseña: {
    width: 250,
    height: 45,
    justifyContent: 'center',
    borderRadius: 10,
    marginLeft: 10,
  },
  signIn: {
    width: '100%',
    height: 50,
    justifyContent: 'center',
    alignItems: 'center',
    borderRadius: 10,
  },
  textSign: {
    fontSize: 18,
    fontWeight: 'bold',
  },
  text_header: {
    color: '#fff',
    fontWeight: 'bold',
    fontSize: 30,
  },
  textInput: {
    flex: 1,
    marginTop: Platform.OS === 'ios' ? 0 : -12,
    paddingLeft: 10,
    color: '#05375a',
  },
})

```

```
//Librerias
import React, { useState, useEffect } from 'react'
import { Text, StyleSheet } from 'react-native'
import { useNavigation } from '@react-navigation/native'
import firebase from '../firebase/fire'
import { ScrollView } from 'react-native-gesture-handler'
import { List, Button } from 'react-native-paper'
import { Avatar, Overlay } from 'react-native-elements'
import { Appbar } from 'react-native-paper'
import { FAB } from 'react-native-paper'
import { connect } from 'react-redux'

//Vista del Usuario
const Usuario = ({ cuentainfo, keyre, agregarinformacion, agregardatos }) => {
  const navigation = useNavigation()
  const [visible, setVisible] = useState(true)
  const [list2, setList2] = useState([])
  const [itemr, setItem] = useState('')
  const itemRefHola = firebase.database().ref('/Usuarios/' + keyre + '/')
  const itemRef2Hola = firebase
    .database()
    .ref('/Usuarios/' + keyre + '/Resultados')

  const toggleOverlay = () => {
    setVisible(!visible)
  }

  useEffect(() => {
    informacionusuario()
  }, [])

  //Cambiar estado - Usuario Pago
  function usuariodermatologo() {
    itemRefHola.update({
      estado: true,
    })
    setVisible(!visible)
  }

  //Cerrar Sesion
  function Cerrar() {
    firebase.auth().signOut()
    const usuario = ''
    const key = ''
    agregardatos(usuario)
    agregarinformacion(key)
  }

  //En listar los resultados
  function informacionusuario() {
    setItem(itemRef2Hola)
    let item2 = []
    itemRef2Hola.on('value', function (snap) {
      let a_ = snap.val()
      for (let x in a_) {
        item2.push({
          probabilidad: a_[x].probabilidad,
          zona: a_[x].zona,
          imagen: a_[x].imagen,
          clase: a_[x].clase,
          done: a_[x].done,
          key: x,
        })
      }
      setList2(item2)
    })
  }

  //Eliminar un Resultado
  function Delete(key) {
    itemr.child(key).remove()
    informacionusuario()
  }
}

```

```
return (
       <ScrollView>

         <Appbar.Header  theme={{colors:{primary:'#448cfc'}}} key={cuentainfo.key}>
           <Appbar.Content title={cuentainfo.nombre} titleStyle={{color:'#FFFFFF'}} />
           <Appbar.Action icon="refresh" color='#FFFFFF' onPress={informacionusuario}/>
           <Appbar.Action icon="image-plus" color='#FFFFFF' onPress={()
=>navigation.navigate("UsuarioResultados")}/>
           <Appbar.Action icon="email" color='#FFFFFF' onPress={() =>
navigation.navigate("UsuarioCorreo")}/>
           <Appbar.Action icon="account-outline" color='#FFFFFF'  onPress={() =>
navigation.navigate("PerfilUsuario")} />
         </Appbar.Header>


            <Button icon="exit-to-app" color='#448cfc' style={styles.editar} mode="text" onPress={() =>
{Cerrar()
            navigation.navigate("InicioUsuario")}}>Cerrar Sesion</Button>

       {list2.map((item) => {

               return (

                   <List.Section  title={item.zona} key={item.key}>
                       <List.Accordion
                       title={item.clase}
                       description={'Probabilidad '+item.probabilidad}
                       left={props => <Avatar source={{uri:item.imagen}}/>}
                       theme={{colors:{primary:'#448cfc'}}}
                       >
                           <List.Item title="Eliminar"  left={props => <List.Icon {...props} icon="trash-
can-outline" />}  onPress={()=>{Delete(item.key)}}>
                           </List.Item>
                       </List.Accordion>
                  </List.Section>
                       )
       })}

    <FAB
        style={styles.fab}
        big
        label='Contactar con un dermatologo'
        icon="send"
        color='#FFFFFF'
        onPress={usuariodermatologo}
    />
    {!visible ?  <Overlay isVisible={!visible} onBackdropPress={toggleOverlay} overlayStyle={styles.over} >
           <Text style={styles.over1}>Tus resultados han sido enviados a nuestro Dermatologo, el se
comunicara contigo mediante correo electronico!</Text>

       </Overlay>:  null}


    </ScrollView>


    );
};

//Redux
const mapStateToProps = state => ({
  keyre: state.key,
  cuentainfo: state.datospersonales
})

const mapDispatchToProps = dispatch => ({
  agregarinformacion(key){
      dispatch({
          type:'Agregar Informacion',
          key
      })
  },
  agregardatos(usuario){
      dispatch({
          type:'Agregar datos',
          usuario
      })
  }
})
export default connect(mapStateToProps,mapDispatchToProps)(Usuario) ;
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  fab: {
    position: 'relative',
    margin: 50,
    right: 0,
    bottom: 0,
  },
  image: {
    width: 150,
    height: 150,
    resizeMode: 'contain',
    alignSelf: 'center',
    marginRight: 50,
  },
  buttonStyle: {
    backgroundColor: '#448cfc',
    borderWidth: 0,
    color: '#FFFFFF',
    borderColor: '#307ecc',
    height: 40,
    alignItems: 'center',
    borderRadius: 30,
    marginLeft: 35,
    marginRight: 35,
    marginTop: 40,
    width: 200,
  },
  editar: {
    width: 200,
    height: 45,
    justifyContent: 'center',
    borderRadius: 10,
    marginLeft: 200,
    marginBottom: 10,
  },
  editar2: {
    width: 200,
    height: 45,
    justifyContent: 'center',
    borderRadius: 10,
    marginBottom: 10,
    marginLeft: 10,
  },
  signIn: {
    width: 200,
    height: 45,
    justifyContent: 'center',
    borderRadius: 10,
    marginLeft: 100,
  },
  textD: {
    color: '#448cfc',
    fontSize: 20,
    alignItems: 'center',
    justifyContent: 'center',
  },
  over: {
    height: 150,
    width: 250,
  },
  over1: {
    marginBottom: 20,
    marginTop: 20,
  },
})

```

UsuarioResultado.js (Script para agregar una nueva clasificación o resultado del usuario)

```
//Librerias
import React , {useState ,useEffect} from 'react';
import { View, Text, TouchableOpacity,StyleSheet ,ScrollView, Image,Alert } from 'react-native';
import {Icon, Avatar} from 'react-native-elements';
import DocumentPicker from 'react-native-document-picker';
import LinearGradient from 'react-native-linear-gradient';
import RNPickerSelect from 'react-native-picker-select';
import firebase from '../firebase/fire';
import {useNavigation} from '@react-navigation/native'
import { connect } from 'react-redux';
import { size, filter } from 'lodash'


//Agregar un nuevo resultado-clasificacion
const UsuarioResultados = ({keyresult}) => {

    const navigation = useNavigation()

    const [Imagen, setImagen] = useState('');
    const [imagesSelected, setImagesSelected] = useState([])
    const [ zona, setZona] = useState("");
    const [bodypost, setbodypost] = useState(null);
    const [result, setResult] = useState(null);
    const [uploaded, setUploaded] = useState("");
    const [uploadedimg, setUploadedimg] = useState("");
    const [uploadedimg2, setUploadedimg2] = useState("");
    const [ text, setText ] = useState("");


    const itemRef = firebase.database().ref('/Usuarios/'+keyresult+'/Resultados/')

    //Retorno a la vista Usuario

    useEffect(()=>{
      if (bodypost){
         itemRef.push(bodypost)

        navigation.navigate("Usuario")
      }


    }, [bodypost])

    //Guardar Resultado
    useEffect(()=>{

      if (uploaded.length){
        setbodypost(
          {
            zona:zona,
            clase: result.label,
            probabilidad: result.score,
            imagen: uploaded,
            observaciones: text,
            resultadoDermatologo:"",
            recomendaciones:"",
            done: false,
            nameimage: Imagen.name,
            urlimage: uploadedimg2,
            prediccion: result.clasificar
          }
        )
      }


    }, [uploaded,uploadedimg2])

```

```
//Seleccionar y enviar imagen al Modelo
const SeleccionarImagen = async () => {
  try {
    const res = await DocumentPicker.pick({
      type: [DocumentPicker.types.images],
    })
    setImagen(res)
    if (res != null) {
      console.log('Contenido de la imagen')
      console.log(res)
      const fileToUpload = res
      const data = new FormData()
      data.append('file', fileToUpload)

      let rest = await fetch('http://34.125.255.5:9000/model/predict/', {
        method: 'post',
        headers: {
          'Content-Type': 'multipart/form-data',
        },

        body: data,
      })
      let result = await rest.json()
      setResult(result)
      console.log('Resultado de la clasificacion')
      console.log(result)
    }
  } catch (err) {
    if (DocumentPicker.isCancel(err)) {
      alert('Canceled from single doc picker')
    } else {
      alert('Unknown Error: ' + JSON.stringify(err))
      throw err
    }
  }
}

//Guardar imagenes
const GuardarResultado = async () => {
  if (zona === '') {
    alert('Por favor ingrese su edad')
  }
  //Imagen clasificada guardar en Firebase
  else {
    const storageref = await firebase.storage().ref('/' + keyresult)
    const fileRef = storageref.child(Imagen.name)
    console.log(fileRef)
    const response = await fetch(Imagen.uri)
    const blob = await response.blob()
    console.log(blob)
    let url
    fileRef.put(blob).then(async () => {
      url = await storageref.child(Imagen.name).getDownloadURL()
      setUploaded(url)
      console.log('Guardo imagen en Firebase')
      console.log(url)
    })

    console.log('Se guarda correctamente en Firebase')
  }
  // Imagenes adicionales guardar firebase
  console.log(imagesSelected)
  let up = []
  const storageimages = firebase
    .storage()
    .ref('/' + keyresult + '/Imagenes Extras/')

  for (const x of imagesSelected) {
    console.log(x.name)
    const re = x.name
    const starsRef = storageimages.child(re)
    starsRef
      .getDownloadURL()
      .then(function (url1) {
        up.push(url1)
      })
      .catch(function (error) {})
  }
  setUploadedimg2(up)
}

```

```
// Seleccionar las imagenes adicionales

const Multiple = async () => {
  try {
    const results = await DocumentPicker.pick({
      type: [DocumentPicker.types.images],
    })
    const storageimages = firebase
      .storage()
      .ref('/' + keyresult + '/Imagenes Extras/')
    const fileimg = storageimages.child(results.name)
    console.log(fileimg)
    const responseimg = await fetch(results.uri)
    const blobimg = await responseimg.blob()
    console.log(blobimg)
    let urlimg
    fileimg.put(blobimg).then(async () => {
      urlimg = await storageimages.child(results.name).getDownloadURL()
      setUploadedimg(urlimg)
      console.log('Guardo imagen en Firebase')
      console.log(uploadedimg)
    })
    let resultsimg = [results]
    console.log(resultsimg)
    for (const res of resultsimg) {
      setImagesSelected([ ... imagesSelected, res])
    }
    console.log(imagesSelected)
  } catch (err) {
    if (DocumentPicker.isCancel(err)) {
    } else {
      throw err
    }
  }
}

//Eliminar una imagen seleccionada

const removeImage = (image) => {
  console.log(image.name)
  const nombreimg = image.name
  const eliminar = () => {
    const refimages = firebase
      .storage()
      .ref('/' + keyresult + '/Imagenes Extras/' + nombreimg)
    refimages
      .delete()
      .then(function () {
        console.log('Se elimino correctamente')
      })
      .catch(function (error) {
        console.log('No')
      })
  }
  Alert.alert(
    'Eliminar Imagen',
    '¿Estas seguro que quieres eliminar la imagen?',
    [
      {
        text: 'No',
        style: 'cancel',
      },
      {
        text: 'Sí',
        onPress: () => {
          setImagesSelected(
            filter(imagesSelected, (imageUrl) => imageUrl =/= image)
          )
          eliminar()
        },
      },
    ],
    { cancelable: false }
  )
}

```

```
  return (
        <ScrollView>
            <View style={styles.container}>
                    <Text style={{fontSize: 20  , color: '#1C5BB7' , marginBottom:30 , marginTop:10
 ,fontStyle: 'normal'}}>Información de la lesión</Text>
                    <Text style={{fontSize: 18  , color: '#448cfc' ,  alignSelf: 'baseline', marginTop:5}}
 >Imagen
                    </Text>
                    {Imagen ?   <Image style={styles.image} source={{uri:Imagen.uri}} />:  null}

                <View style={styles.button}>
                        <TouchableOpacity style={styles.signIn} onPress={SeleccionarImagen}>
                            <LinearGradient colors={['#6cacfc', '#448cfc']} style={styles.signIn}>
                                <Text style={[styles.textSign, {color:'#fff' }]}>Seleccionar Imagen</Text>
                            </LinearGradient>
                        </TouchableOpacity>
                </View>
                <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf: 'baseline', marginTop:
 30}}>Imagenes adicionales para el analisis de Resultados</Text>
                <ScrollView
                            horizontal
                            style={styles.viewImages}
                        >
                            {
                                size(imagesSelected) < 4 && (
                                    <Icon
                                        type="material-community"
                                        name="camera"
                                        color="#7a7a7a"
                                        containerStyle={styles.containerIcon}
                                        onPress={Multiple}
                                    />
                                )
                            }
                            {
                                imagesSelected.map( (imageRestaurant, index) => (
                                    <Avatar
                                        key={index}
                                        style={styles.miniatureStyle}
                                        source={{ uri: imageRestaurant.uri }}
                                        onPress={() => removeImage(imageRestaurant)}
                                    />

                                ))
                            }

                </ScrollView>
                    <Text style={{fontSize: 18  , color: '#448cfc' ,  alignSelf: 'baseline', marginTop:
 40}}>Ubicacion de la lesion</Text>
                    <Text style={{fontSize: 13 , color: 'black' ,  alignSelf: 'baseline', marginTop: 10,
 marginBottom:10}}>Por favor seleccione en que zona del cuerpo fue tomada la imagen ingresada.</Text>
                    <RNPickerSelect
                        placeholder={{ }}
                        onValueChange={(zona) => setZona(zona)}
                        items={[
                        { label: "Rostro (cara,cuello y zonas cercanas)", value: "Rostro" },
                        { label: "Central (pecho y abdomen)", value: "Central" },
                        { label: "Espalda (excepto gluteos)" , value:"Espalda"},
                        { label: "Brazo (excepto la mano)" , value:"Brazo"},
                        { label: "Mano (muñeca , dorso , palma y dedos)" , value:"Mano"},
                        { label: "Genitales (Toda la zona de genitales)" , value:"Genitales"},
                        { label: "Gluteos (Toda la zona de gluteos)" , value:"Gluteos"},
                        { label: "Pierna (parte frontal y posterior)" , value:"Gluteos"},
                        { label: "Pie (Toda las zonas del pie)" , value:"Pie"},
                    ]}/>
                        <TouchableOpacity style={styles.button}onPress={GuardarResultado}>
                            <LinearGradient colors={['#6cacfc', '#448cfc']} style={styles.signIn}>
                                <Text style={[styles.textSign, { color:'#fff'}]}>Enviar Informacion</Text>
                            </LinearGradient>
                        </TouchableOpacity>

            </View>
        </ScrollView>


    );
};
//Redux
const mapStateToProps = state => ({
  keyresult: state.key,
  cuentadatos: state.datospersonales,
  email:state.cuenta
})


export default connect(mapStateToProps,{})(UsuarioResultados);
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    padding: 30,
  },
  containerIcon: {
    alignItems: 'center',
    justifyContent: 'center',
    marginRight: 10,
    height: 70,
    width: 79,
    backgroundColor: '#e3e3e3',
  },
  viewImages: {
    flexDirection: 'row',
    marginHorizontal: 20,
    marginTop: 30,
  },
  miniatureStyle: {
    width: 70,
    height: 70,
    marginRight: 10,
  },
  over: {
    height: 350,
    width: 350,
  },
  over1: {
    marginBottom: 20,
    marginTop: 20,
  },
  tinyLogo: {
    width: 300,
    height: 50,
  },
  buttonStyle: {
    backgroundColor: '#448cfc',
    borderWidth: 0,
    color: '#FFFFFF',
    borderColor: '#307ecc',
    height: 40,
    alignItems: 'center',
    borderRadius: 30,
    marginLeft: 35,
    marginRight: 35,
    marginTop: 15,
    width: 200,
  },
  image: {
    width: 400,
    height: 350,
    resizeMode: 'contain',
  },
  buttonTextStyle: {
    color: '#FFFFFF',
    paddingVertical: 10,
    fontSize: 16,
  },
  buttonR: {
    backgroundColor: 'darkblue',
    marginTop: 30,
  },
  button: {
    marginBottom: 20,
  },
  signIn: {
    width: 200,
    height: 45,
    justifyContent: 'center',
    alignItems: 'center',
    borderRadius: 10,
    marginTop: 10,
  },
  textSign: {
    fontSize: 17,
  },
})

```

UsuarioCorreo.js (Envío del PDF con el resumen de los resultados, al correo electrónico)

```
//Librerias
import React, { useState } from 'react'
import {
  View,
  Text,
  StyleSheet,
  ScrollView,
  TouchableOpacity,
} from 'react-native'
import { Input } from 'react-native-elements'
import { connect } from 'react-redux'
import LinearGradient from 'react-native-linear-gradient'

const UsuarioCorreo = ({ keycorreo }) => {
  const [correo, setCorreo] = useState(null)

  //Enviar informacion a Fastapi para generar PDF resultados
  const EnviarCorreo = async () => {
    try {
      let res = await fetch('http://34.125.255.5:9000/correo/', {
        method: 'post',
        mode: 'no-cors',
        headers: {
          Accept: 'application/json',
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          correo: correo,
          llave: keycorreo,
        }),
      })
      let corr = await res.json()
      console.log('Envio de datos')
    } catch (err) {
      alert('Unknown Error: ' + JSON.stringify(err))
      throw err
    }
  }


```

```
return (
        <ScrollView>

        <View style={styles.container}>
            <Text style={{fontSize: 18  , color: '#448cfc' ,  alignSelf: 'baseline', marginTop: 40,marginBottom:15}}>Envio de Resultados</Text>
            <Text style={{fontSize: 13 , color: 'black' ,  alignSelf: 'baseline', marginTop: 10, marginBottom:10}}>Por favor ingrese un correo valido donde desea que sean enviado los resultados.</Text>
            <Input
            keyboardType='default'
            onChangeText={(correo) => setCorreo(correo)}
            placeholder='Correo Electronico'
            />
            <TouchableOpacity style={styles.button}
            onPress={EnviarCorreo}
            >
                <LinearGradient colors={['#6cacfc', '#448cfc']} style={styles.signIn}>
                    <Text style={[styles.textSign, { color:'#fff'}]}>Enviar Resultados</Text>
                </LinearGradient>
            </TouchableOpacity>
        </View>

        </ScrollView>


    );
};
//Redux
const mapStateToProps = state => ({
    keycorreo: state.key,

  })
export default connect(mapStateToProps,{}) (UsuarioCorreo);
const styles = StyleSheet.create({
    container:{
        flex:1,
        alignItems: 'center',
        padding: 30,

    },
    over1:{
        marginBottom:20,
        marginTop:20

    },
    signIn: {
        width: 200,
        height: 45,
        justifyContent: 'center',
        alignItems: 'center',
        borderRadius: 10,
        marginTop:10
    },
    textSign: {
        fontSize: 17,
    }
})

```

PerfilUsuario.js (Información que ha registrado el usuario)

```
//Librerias
import React from 'react'
import { connect } from 'react-redux'
import { ScrollView } from 'react-native'
import { DataTable } from 'react-native-paper'
import { List } from 'react-native-paper'
import { View, StyleSheet } from 'react-native'

//Perfil del Usuario
const InformacionP = ({ keyinformacion, datosinformacion }) => {
  console.log(keyinformacion)
  const item = datosinformacion
  return (
    <ScrollView>
      <List.Section
        titleStyle={styles.textD}
        title={item.nombre}
        key={item.key}
      >
        <View style={styles.container}>
          <DataTable>
            <DataTable.Row>
              <DataTable.Cell>Genero</DataTable.Cell>
              <DataTable.Cell>{item.genero}</DataTable.Cell>
            </DataTable.Row>
            <DataTable.Row>
              <DataTable.Cell>Edad</DataTable.Cell>
              <DataTable.Cell>{item.edad + ' años'}</DataTable.Cell>
            </DataTable.Row>
            <DataTable.Row>
              <DataTable.Cell>Pais</DataTable.Cell>
              <DataTable.Cell>{item.pais}</DataTable.Cell>
            </DataTable.Row>
            <DataTable.Row>
              <DataTable.Cell>Cuidad</DataTable.Cell>
              <DataTable.Cell>{item.cuidad}</DataTable.Cell>
            </DataTable.Row>
            <DataTable.Row>
              <DataTable.Cell>Ocupacion</DataTable.Cell>
              <DataTable.Cell>{item.ocupacion}</DataTable.Cell>
            </DataTable.Row>
            <DataTable.Row>
              <DataTable.Cell>Exposicion al sol</DataTable.Cell>
              <DataTable.Cell>{item.hora + ' al dia'}</DataTable.Cell>
            </DataTable.Row>
            <DataTable.Row>
              <DataTable.Cell>Antecedentes de Cancer</DataTable.Cell>
              <DataTable.Cell>{item.cancer}</DataTable.Cell>
            </DataTable.Row>
            <DataTable.Row>
              <DataTable.Cell>Ant. Cancer de piel</DataTable.Cell>
              <DataTable.Cell>{item.piel}</DataTable.Cell>
            </DataTable.Row>
          </DataTable>
        </View>
      </List.Section>
    </ScrollView>
  )
}

const mapStateToProps = (state) => ({
  keyinformacion: state.key,
  datosinformacion: state.datospersonales,
})

export default connect(mapStateToProps, {})(InformacionP)
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
})

```

Dermatologo.js (Pantalla principal del Dermatólogo)

```
//Librerias
import React from 'react'
import { StyleSheet } from 'react-native'
import { useNavigation } from '@react-navigation/native'
import { closeSession } from '../utils/actions'
import { ScrollView } from 'react-native-gesture-handler'
import { FAB } from 'react-native-paper'
import { Appbar } from 'react-native-paper'
import { connect } from 'react-redux'

//Vista del Dermatologo
const Dermatologo = ({ dermatologo }) => {
  const navigation = useNavigation()

  return (
    <ScrollView>
      <Appbar.Header theme={{ colors: { primary: '#448cfc' } }}>
        <Appbar.Content
          title={dermatologo.nombre}
          titleStyle={{ color: '#FFFFFF' }}
        />
        <Appbar.Action
          icon="exit-to-app"
          color="#FFFFFF"
          onPress={() => {
            closeSession()
            navigation.navigate('InicioUsuario')
          }}
        />
      </Appbar.Header>

      <FAB
        style={styles.fab}
        big
        label="Usuarios de Pago"
        icon="account-cash-outline"
        color="#FFFFFF"
        onPress={() => navigation.navigate('UsuarioPago')}
      />
      <FAB
        style={styles.fab}
        big
        label="Usuarios Generales"
        icon="account-group"
        color="#FFFFFF"
        onPress={() => navigation.navigate('UsuarioGeneral')}
      />
    </ScrollView>
  )
}
//Redux
const mapStateToProps = (state) => ({
  dermatologo: state.dermatologos,
})
export default connect(mapStateToProps, {})(Dermatologo)
const styles = StyleSheet.create({
  fab: {
    position: 'relative',
    margin: 40,
    right: 0,
    bottom: 0,
  },
})

```

UsuariosPago.js (Lista de usuarios que han contactado servicios Dermatólogo)

```
//Librerias
import React, {useState, useEffect} from 'react';
import {View, StyleSheet} from 'react-native';
import firebase from '../firebase/fire';
import { ScrollView } from 'react-native-gesture-handler';
import { List} from 'react-native-paper';
import {Title} from 'react-native-paper';


const bdusuariop = firebase.database().ref('/Usuarios/')

//Vista Usuarios de Pago
const UsuarioPago = (props) => {

    const [list, setList] = useState([])
    const [donepago, setDonepago] = useState(true)
    const [actualizarpago, setActualizarpago] = useState(false)
    const [llavepago, setLlavepago] = useState('')
    useEffect(() => {
        fetchDataUsuarioP()
      }, [])

      useEffect(() => {
          if (actualizarpago) {
            bdusuariop.child(llavepago).update({ estado: donepago })
            setActualizarpago(false)
            fetchDataUsuarioP()


          }


      }, [donepago]);

      function UpdatePago(key) {
          setActualizarpago(true)
          setLlavepago(key)

      }

  //Lista de Usuarios
      function fetchDataUsuarioP(){
          let item = [];
          bdusuariop.on('value', function(snap){
              let a_ = snap.val();
              for (let x in a_){
                  item.push({estado:a_[x].estado,cuenta:a_[x].cuenta,key:x})
              }
              setList(item)


          } )


      }
      const lista = [ ...list]


```

```
return (
        <ScrollView>
        <Title style={{alignSelf:'center', color:'#448cfc'}}>Usuarios de Pago</Title>
        <View
        style={{
            marginTop:10,
            borderBottomColor: '#8ab5fb',
            borderBottomWidth: 8,
        }}
        />
    <List.Section>
        <List.Accordion title=' Usuarios' theme={{colors:{primary:'#448cfc'}}}>
        {lista.map((usuaripago,i) => {
        let d = 0
            return (
               usuaripago.estado && (

                <List.Section key={usuaripago.key}>
                <List.Accordion
                   title={usuaripago.cuenta}
                  theme={{colors:{primary:'#003244'}}}
                  >

                  <List.Item title="Ver los Resultados" left={props => <List.Icon {...props} icon="clipboard-text-outline" />} onPress={()=>props.navigation.navigate("Resultados",{userid:{key:usuaripago.key, cuenta:usuaripago.cuenta}})} >
                    </List.Item>
                    <List.Item title="Revisado" left={props => <List.Icon {...props} icon="check" />} onPress={()=>{UpdatePago(usuaripago.key);setDonepago(!donepago); setList(lista)}} >
                    </List.Item>
                </List.Accordion>
               </List.Section>

               )

                    )
    })}
        </List.Accordion>
    </List.Section>

</ScrollView>
 );

}

export default UsuarioPago;
const styles = StyleSheet.create({
    container:{
        flex:1,
        alignItems: 'center',
        justifyContent: 'center'
    },
    fab: {
        position: 'absolute',
        margin: 16,
        right: 0,
        bottom: 0,
      },
    image:{
        width:150,
        height:150,
        resizeMode:'contain',
        alignSelf:'center',
        marginRight:50
      },
      buttonStyle: {
        backgroundColor: '#448cfc',
        borderWidth: 0,
        color: '#FFFFFF',
        borderColor: '#307ecc',
        height: 40,
        alignItems: 'center',
        borderRadius: 30,
        marginLeft: 35,
        marginRight: 35,
        marginTop: 40,
        width: 200
      },
      editar: {
        width: 200,
        height: 45,
        justifyContent: 'center',
        borderRadius: 10,
        marginLeft:200,
        marginBottom:10,

    },
    signIn: {
        width: 200,
        height: 45,
        justifyContent: 'center',
        borderRadius: 10,
        marginLeft:100
    },
    textD: {
        color:'#448cfc',
        fontSize: 20,
        alignItems: 'center',
        justifyContent: 'center'
    },
});
```

UsuariosGenerales.js (Lista de usuarios que no han contactado servicios con Dermatólogo)

```
//Librerias
import React, {useState, useEffect, Component} from 'react';
import {View,  StyleSheet} from 'react-native';
import firebase from '../firebase/fire';
import { ScrollView } from 'react-native-gesture-handler';
import { List} from 'react-native-paper';
import {Title} from 'react-native-paper';

const refUsuarioG = firebase.database().ref('/Usuarios')

//Vista Usuarios Generales
function UsuarioG (props) {

    const [list, setList] = useState([])
    const [expanded, setExpanded] = React.useState(true);
    const [doneg, setDoneG] = useState(true)
    const [actualizar, setActualizar] = useState(false)
    const [llave, setLlave] = useState('')

    useEffect(() => {
        fetchDataUsuarioG()
    }, [])

    useEffect(() => {
        if (actualizar) {
            refUsuarioG.child(llave).update({ estado: doneg })
            setActualizar(false)
            fetchDataUsuarioG()


        }


    }, [doneg]);


   //Lista de usuarios
    function fetchDataUsuarioG(){
        let itemg = [];
        refUsuarioG.on('value', function(snap){
            let a_ = snap.val();


            for (let x in a_){

                const rolusuario = a_[x].rol

                if(rolusuario===1){
                    console.log('si')
                    itemg.push({cuenta:a_[x].cuenta,done:a_[x].done,estado:a_[x].estado,key: x})
                    setList(itemg)
                }else {
                    console.log('no')
                }

            }


        } )
    }

    const lista = [ ...list]

```

```
return (
        <ScrollView>
            <Title style={{alignSelf:'center', color:'#448cfc'}}>Usuario General</Title>
            <View
            style={{
                marginTop:10,
                borderBottomColor: '#8ab5fb',
                borderBottomWidth: 8,
            }}
            />
        <List.Section>
            <List.Accordion title=' Usuarios' theme={{colors:{primary:'#448cfc'}}}>
            {lista.map((usuaripago,i) => {

                return (
                   !usuaripago.estado && (

                    <List.Section key={usuaripago.key}>
                    <List.Accordion
                       title={usuaripago.cuenta}
                      theme={{colors:{primary:'#003244'}}}
                      >

                       <List.Item title="Ver los Resultados" left={props => <List.Icon {...props}
  icon="clipboard-text-outline" />} onPress={()=>props.navigation.navigate("Resultados",{userid:
  {key:usuaripago.key, cuenta:usuaripago.cuenta}})} >
                            </List.Item>

                    </List.Accordion>
                   </List.Section>

                   )

                       )
        })}
            </List.Accordion>
        </List.Section>


    </ScrollView>


    );
};

export default UsuarioG;

const styles = StyleSheet.create({
    container:{
        flex:1,
        alignItems: 'center',
        justifyContent: 'center'
    },
    fab: {
        position: 'absolute',
        margin: 16,
        right: 0,
        bottom: 0,
      },
    image:{
        width:150,
        height:150,
        resizeMode:'contain',
        alignSelf:'center',
        marginRight:50
      },
      buttonStyle: {
        backgroundColor: '#448cfc',
        borderWidth: 0,
        color: '#FFFFFF',
        borderColor: '#307ecc',
        height: 40,
        alignItems: 'center',
        borderRadius: 30,
        marginLeft: 35,
        marginRight: 35,
        marginTop: 40,
        width: 200
      },
      editar: {
        width: 200,
        height: 45,
        justifyContent: 'center',
        borderRadius: 10,
        marginLeft:200,
        marginBottom:10,

    },
    signIn: {
        width: 200,
        height: 45,
        justifyContent: 'center',
        borderRadius: 10,
        marginLeft:100
    },
    textD: {
        color:'#448cfc',
        fontSize: 20,
        alignItems: 'center',
        justifyContent: 'center'
    },
});
```

Resultados.js (Lista de los resultados del usuario)

```
//Librerias
import React, {useState, useEffect} from 'react';
import {View, StyleSheet} from 'react-native';
import firebase from '../firebase/fire';
import { ScrollView } from 'react-native-gesture-handler';
import {Avatar} from 'react-native-elements';
import { List,FAB,Title} from 'react-native-paper';


//Lista de resultados por usuario
function ResultadosD (props) {
    const fe = props.route.params.userid
    const key = fe.key
    const cuenta = fe.cuenta
    const refResultados = firebase.database().ref('/Usuarios/'+key+'/Resultados')
    const [list, setList] = useState([])
    const [done, setDone] = useState(false)
    const [actualizar, setActualizar] = useState(false)
    const [llave, setLlave] = useState('')

    useEffect(() => {
      fetchDataResultados()
    }, [])

    useEffect(() => {
        if (actualizar) {
            refResultados.child(llave).update({ done: done })
            setActualizar(false)
            fetchDataResultados()


        }


    }, [done]);

    function UpdateResultados(key) {

        setActualizar(true)
        setLlave(key)

    }
    //Mostrar cada resultado
    function fetchDataResultados(){
        let item = [];
        refResultados.on('value', function(snap){
            let a_ = snap.val();
            for (let x in a_){
                item.push({clase:a_[x].clase,imagen:a_[x].imagen,done:a_[x].done,key: x,cuenta:cuenta})
            }
                setList(item)


        } )
    }
    //Enviar correo de notificacion
    const enviar = async (keyu) => {

          try {
          let res = await fetch(
            'http://34.125.255.5:9000/resultado/',
            {
              method:'post',
              mode:'no-cors',
              headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json'
              },
              body: JSON.stringify({
                llave: key,
                resultadounico:keyu
              })

            }
          );
          let corr = await res.json();
          console.log('Envio de datos')

        } catch (err) {
            alert('Unknown Error: ' + JSON.stringify(err));
            throw err;
        }



      }

    const lista = [ ...list]

```

```
    return (
        <ScrollView>
            <Title style={{alignSelf:'center', color:'#448cfc'}}>Resultados</Title>
            <View
            style={{
                marginTop:10,
                borderBottomColor: '#8ab5fb',
                borderBottomWidth: 8,
            }}
            />
        <List.Section>
            <List.Accordion title=' Pendientes' theme={{colors:{primary:'#448cfc'}}}>
            {lista.map((usuario,i) => {
            let d = 0
                return (
                   !usuario.done && (

                    <List.Section key={usuario.key} title={usuario.cuenta}>
                    <List.Accordion
                       title={usuario.clase}
                      theme={{colors:{primary:'#003244'}}}
                      left={props => <Avatar source={{uri:usuario.imagen}}/>}
                      >

                      <List.Item title="Visualizar Informacion" left={props => <List.Icon {...props}
icon="account-details" />} onPress={()=>props.navigation.navigate("DetalleResultado",{userid:
{key:usuario.key, cuenta:fe}})} >
                        <FAB
                              style={styles.fab}
                              small
                              icon="plus"
                              onPress={() => console.log('Pressed')}
                          />
                          </List.Item>
                          <List.Item title="Revision Completada" left={props => <List.Icon {...props}
icon="check" />}  onPress={()=>{UpdateResultados(usuario.key, usuario.done);setDone(!done); setList(lista)}}>
                          </List.Item>
                     </List.Accordion>
                    </List.Section>

                    )

                        )
        })}
            </List.Accordion>
        </List.Section>

        <List.Section>
            <List.Accordion title=' Revisados' theme={{colors:{primary:'#448cfc'}}}>
            {lista.map((usuario,i) => {
            let d = 0
                return (
                   usuario.done && (
                    <List.Section key={usuario.key} title={usuario.cuenta}>
                    <List.Accordion
                       title={usuario.clase}
                      theme={{colors:{primary:'#003244'}}}
                      left={props => <Avatar source={{uri:usuario.imagen}}/>}
                      >
                     <List.Item title="Marcar como pendiente" left={props => <List.Icon {...props}
icon="alert-circle-outline" />}  onPress={()=>{UpdateResultados(usuario.key, usuario.done);setDone(!done);
setList(lista)}}>
                          </List.Item>
                      <List.Item title="Enviar notificacion por correo" left={props => <List.Icon {...props}
icon="email" />}onPress={() => enviar(usuario.key)}   >
                          </List.Item>

                     </List.Accordion>
                    </List.Section>

                    )

                        )
        })}

            </List.Accordion>
        </List.Section>

    </ScrollView>


    );
};

export default ResultadosD;

```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  fab: {
    position: 'absolute',
    margin: 16,
    right: 0,
    bottom: 0,
  },
  image: {
    width: 150,
    height: 150,
    resizeMode: 'contain',
    alignSelf: 'center',
    marginRight: 50,
  },
  buttonStyle: {
    backgroundColor: '#448cfc',
    borderWidth: 0,
    color: '#FFFFFF',
    borderColor: '#307ecc',
    height: 40,
    alignItems: 'center',
    borderRadius: 30,
    marginLeft: 35,
    marginRight: 35,
    marginTop: 40,
    width: 200,
  },
  editar: {
    width: 200,
    height: 45,
    justifyContent: 'center',
    borderRadius: 10,
    marginLeft: 200,
    marginBottom: 10,
  },
  signIn: {
    width: 200,
    height: 45,
    justifyContent: 'center',
    borderRadius: 10,
    marginLeft: 100,
  },
  textD: {
    color: '#448cfc',
    fontSize: 20,
    alignItems: 'center',
    justifyContent: 'center',
  },
})

```

DetalleResultado.js (Información detallada de cada resultado)

```
//Librerias
import React, {useState, useEffect} from 'react';
import {View, Text, StyleSheet, TouchableOpacity} from 'react-native';
import { TextInput } from 'react-native-paper';
import firebase from '../firebase/fire';
import { ScrollView } from 'react-native-gesture-handler';
import { List } from 'react-native-paper';
import { Image } from 'react-native';
import ImageZoom from 'react-native-image-pan-zoom';
import { DataTable } from 'react-native-paper';
import RNPickerSelect from 'react-native-picker-select';
import { Avatar} from 'react-native-elements';



//Informacion de cada Resultado
function DetalleUsuario(props) {


    const fe = props.route.params.userid
    const llave1 = fe.cuenta.key
    const llave = fe.key
    const refDetalle = firebase.database().ref('/Usuarios/'+llave1+'/Resultados/'+llave)
    const refDetalle2 = firebase.database().ref('/Usuarios/'+llave1)
    const refDetalle3 = firebase.database().ref('/Administrador/elementos/')
    const [list, setList] = useState([])
    const [clases, setClases] = useState([])
    const [clases2, setClases2] = useState([])
    const [select, setSelect] = useState('')
    const [text, setText] = useState('')
    const [recomendaciones, setRecomendaciones] = useState('')
    const [user, setUser] = useState('')
    let [isUpdating, setIsUpdating] = useState(false)
    let [currentKey, setCurrentKey] = useState('')
    const [result, setResult] = useState(false)
    const [resultados, setResultados] = useState('')
    const [ nueva, setNueva ] = useState("");

    useEffect(() => {
        fetchDataDetalle()
        fetchDataDetalleResult()
        fetch()

      }, [])



      //Traer informacion del resultado desde Firebase
      useEffect(() => {
        refDetalle.on('value', function(snap){
            let item = [];
            let a_ = snap.val();
            const pred = a_.prediccion
            item.push({nombre:
  user.nombre,genero:user.genero,edad:user.edad,cuidad:user.cuidad,pais:user.pais,ocupacion:user.ocupacion,hora
  :user.hora,cancer:user.cancer,piel:user.piel,zona:a_.zona,imagen:
  a_.imagen,observaciones:a_.observaciones,clase:a_.clase,probabilidad:a_.probabilidad,resultadoDermatologo:a_.
  resultadoDermatologo,recomendaciones:a_.recomendaciones,urlimage:a_.urlimage, prediccion: a_.prediccion, key:
  fe.key})
            setList(item)
            setClases(pred)

        } )
        refDetalle.off();

      }, [user])


      useEffect(() => {


            let clss = []
            clases2.map( (ite, index) =>(
                clss.push({label:ite,value:ite})

            ))
            setSelect(clss)
            console.log(clss)



      }, [clases2])



    //Editar y Actualizar Observaciones
    function addResultado(key) {

        //Actualizar el resultado
        if (result) {
            refDetalle.update({ resultadoDermatologo: resultados })
            setResult(false)
            fetchDataDetalleResult()
            refDetalle.off();
        }
        }
        function handleResultado(key) {
            setResult(true)
            setCurrentKey(key)

        }

```

```
//Consulta a Firebase, traer datos por usuario
function fetchDataDetalleResult() {
  refDetalle2.on('value', function (snap) {
    let a_ = snap.val()
    setUser(a_.datospersonales)
  })
  refDetalle2.off()
}

const reemplazar = (image) => {
  setNueva(image)
}

function addTextHandleDetalle(key) {
  //Actualizar Recomendaciones y Observaciones
  if (isUpdating) {
    refDetalle.update({ observaciones: text, recomendaciones: recomendaciones })
    setText('')
    setRecomendaciones('')
    setIsUpdating(false)
    fetchDataDetalle()
  }
}
function handleUpdateDetalle(key, text, recomendaciones) {
  setIsUpdating(true)
  setText(text)
  setRecomendaciones(recomendaciones)
  setCurrentKey(key)
}

//Consulta a Firebase, traer datos por usuario
function fetchDataDetalle() {
  refDetalle2.on('value', function (snap) {
    let a_ = snap.val()
    setUser(a_.datospersonales)
  })
  refDetalle2.off()
}
function fetch() {
  console.log('x')
  refDetalle3.on('value', function (snap) {
    console.log('d')
    let a_dd = snap.val()
    console.log(a_dd)
    console.log('dsa')
    const elem = a_dd
    setClases2(elem)
  })
}

```

```
        return (
            <ScrollView>
                {list.map((item,index) => {

                        return (
                           <List.Section titleStyle={styles.textD} title={item.nombre}>
                                <View style={styles.container}>
                                        <DataTable>
                                            <DataTable.Row>
                                                <DataTable.Cell>Genero</DataTable.Cell>
                                                <DataTable.Cell>{item.genero}</DataTable.Cell>
                                            </DataTable.Row>
                                            <DataTable.Row>
                                                <DataTable.Cell>Edad</DataTable.Cell>
                                                <DataTable.Cell>{item.edad+' años'}</DataTable.Cell>
                                            </DataTable.Row>
                                            <DataTable.Row>
                                                <DataTable.Cell>Pais</DataTable.Cell>
                                                <DataTable.Cell>{item.pais}</DataTable.Cell>
                                            </DataTable.Row>
                                            <DataTable.Row>
                                                <DataTable.Cell>Cuidad</DataTable.Cell>
                                                <DataTable.Cell>{item.cuidad}</DataTable.Cell>
                                            </DataTable.Row>
                                            <DataTable.Row>
                                                <DataTable.Cell>Ocupacion</DataTable.Cell>
                                                <DataTable.Cell>{item.ocupacion}</DataTable.Cell>
                                            </DataTable.Row>
                                            <DataTable.Row>
                                                <DataTable.Cell>Exposicion al sol</DataTable.Cell>
                                                <DataTable.Cell>{item.hora+' al dia'}</DataTable.Cell>
                                            </DataTable.Row>
                                            <DataTable.Row>
                                                <DataTable.Cell>Antecedentes de Cancer</DataTable.Cell>
                                                <DataTable.Cell>{item.cancer}</DataTable.Cell>
                                            </DataTable.Row>
                                            <DataTable.Row>
                                                <DataTable.Cell>Antecedentes Cancer de piel</DataTable.Cell>
                                                <DataTable.Cell>{item.piel}</DataTable.Cell>
                                            </DataTable.Row>
                                            <DataTable.Row>
                                                <DataTable.Cell>Ubicacion de la lesion</DataTable.Cell>
                                                <DataTable.Cell>{item.zona}</DataTable.Cell>
                                            </DataTable.Row>
                                            <DataTable.Row>
                                            <DataTable.Cell >Resultado del modelo</DataTable.Cell>
                                            <DataTable.Cell>{item.clase}</DataTable.Cell>
                                        </DataTable.Row>
                                        <DataTable.Row>
                                            <DataTable.Cell >Probabilidad</DataTable.Cell>
                                            <DataTable.Cell>{item.probabilidad + '%'}</DataTable.Cell>
                                        </DataTable.Row>

                                        </DataTable>
                                </View>

```

```
        <View style={styles.container2}>
              <Text style={[styles.textD,{marginBottom:15}]} >Resultados Prediccion</Text>
                                  </View>

                                  <View style={styles.container5}>
                                  <View style={styles.item}>
                              {
                                  clases.map( (elemento, index) => {
                                    return(
                                        <View>

                                            <DataTable>
                                                <DataTable.Row>
                                                    <DataTable.Cell>{elemento}</DataTable.Cell>
                                                </DataTable.Row>
                                            </DataTable>

                                        </View>
                                    )
                                })
                                        }
                                    </View>
                                        <View style={styles.item}>

                                        {
                                                clases2.map((it,index) => {
                                                    return(
                                                    <View>
                                                    <DataTable>
                                                        <DataTable.Row>
                                                            <DataTable.Cell>{it}</DataTable.Cell>
                                                        </DataTable.Row>
                                                    </DataTable>
                                                    </View>

                                                    )
                                                })
                                            }

                                        </View>
                            </View>

                                        <View style={styles.container2}>
                                            <Text style={[styles.textD,{marginBottom:15}]} >Imagen Evaluada</Text>
                                            <ImageZoom cropWidth={400}
                                                cropHeight={350}
                                                imageWidth={350}
                                                imageHeight={350}>

                                                <Image style={styles.image}
                                                    source={{uri:item.imagen}}/>
                                                </ImageZoom>
                                          <List.Item
                                                    title="Abrir imagen"
                                                    ></List.Item>
                                                    <Text style={[{ color:'#448cfc', fontSize: 15,
  marginBottom:20,marginBottom:20}]} >Imagenes adicionales de la lesion</Text>
                                        </View>
```

```
<ScrollView horizontal style={styles.viewImages}>

  {

  item.urlimage.map( (imageRestaurant, index) => (
  <Avatar
  key={index}
  style={styles.miniatureStyle}
  source={{ uri: imageRestaurant }}
  onPress={() => reemplazar(imageRestaurant)}
  />

  ))
  }


</ScrollView>
  {nueva ? <ImageZoom cropWidth={400}
  cropHeight={250}
  imageWidth={350}
  imageHeight={350}>

  <Image
  style={styles.image2}
  source={{uri:nueva}} />
</ImageZoom>  :  null}




<View style={styles.container}>


<Text style={[styles.textD,{marginTop:15}]}>Cambio de Resultado</Text>
{result ?


<RNPickerSelect
  placeholder={{ }}
  onValueChange={(resulta) => setResultados(resulta)}
  items={select}
/> :  <Text style={{marginTop:20, marginBottom:10}}>{item.resultadoDermatologo}</Text>}




</View>



<View style={styles.container}>
    {result ?
    <TouchableOpacity
    onPress={addResultado}
    style={[styles.signIn,{
    backgroundColor:'#5494fc',
    borderColor:'#5494fc',
    borderEndWidth:1,
    marginTop:10
    }]}>
    <Text style={[styles.textSign,{
    color:'#ffff'
    }]}>Actualizar</Text>
    </TouchableOpacity> :  <View style={styles.button}>
    <TouchableOpacity
    onPress={() => handleResultado(item.key)}
    style={[styles.signP,{
    borderColor:'#e3e4f4',
    borderWidth:1,
    marginBottom:30,
    backgroundColor:'#448cfc'
    }]}>
    <Text style={[styles.title, {
    color:'#FFFFFF'
    }]}>Editar Resultado</Text>


</TouchableOpacity>
</View>
}
</View>
<View
style={{
marginTop:10,
borderBottomColor: '#8ab5fb',
borderBottomWidth: 8,
}}
/>
```

```
<View style={styles.container}>
      <Text style={[styles.textD,{marginTop:30}]}>Observaciones</Text>
   {isUpdating ?
   <TextInput
   mode="outlined"
   multiline={true}
   numberOfLines={8}
   style={styles.textInput}
   label="Escribe aqui"
   placeholder=""
   right={<TextInput.Affix etext="/100">
   </TextInput.Affix>}
  theme={{colors: {primary:'#448cfc'}}}
  value={text} onChangeText={(e)⇒setText(e)}/> : <Text style={{marginTop:20,padding:15}}>
{item.observaciones}</Text>}


  <Text style={[styles.textD,{marginTop:5}]}>Recomendaciones o Tratamiento</Text>
  {isUpdating ?
  <TextInput
  mode="outlined"
  multiline={true}
  numberOfLines={5}
  style={styles.textInput}
  label="Escribe aqui"
  placeholder=""
  right={<TextInput.Affix etext="/100">
  </TextInput.Affix>}
 theme={{colors: {primary:'#448cfc'}}}
 value={recomendaciones} onChangeText={(e)⇒setRecomendaciones(e)}/> : <Text style=
{{marginTop:20,padding:15}}>{item.recomendaciones}</Text>}

                {isUpdating ?
                 <TouchableOpacity
                 onPress={addTextHandleDetalle}
                 style={[styles.signIn,{
                     backgroundColor:'#5494fc',
                     borderColor:'#5494fc',
                     borderEndWidth:1,
                     marginTop:10
                 }]}>
                     <Text style={[styles.textSign,{
                         color:'#ffff'
                     }]}>Actualizar</Text>
                 </TouchableOpacity> :
                 <View style={styles.button}>
                     <TouchableOpacity
                     onPress={() ⇒
handleUpdateDetalle(item.key,item.observaciones,item.recomendaciones)}
                     style={[styles.signP,{
                         borderColor:'#e3e4f4',
                         borderWidth:1,
                         marginTop:15,
                         marginBottom:50,
                         backgroundColor:'#448cfc'
                     }]}>
                         <Text style={[styles.title, {
                             color:'#FFFFFF'
                         }]}>Editar</Text>


                     </TouchableOpacity>
                 </View>

                 }

                </View>

                </List.Section>

                   )
      })}

  </ScrollView>


);
};

export default DetalleUsuario;


```

```
//Estilos
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  container2: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    marginTop: 20,
  },
  viewImages: {
    flexDirection: 'row',
    marginHorizontal: 20,
    marginBottom: 20,
  },
  miniatureStyle: {
    width: 70,
    height: 70,
    marginRight: 10,
  },
  image: {
    width: 350,
    height: 350,
    resizeMode: 'contain',
    alignItems: 'center',
    justifyContent: 'center',
  },
  image2: {
    width: 350,
    height: 350,
    resizeMode: 'contain',
    marginLeft: 20,
    marginBottom: 20,
    alignItems: 'center',
    justifyContent: 'center',
  },
  buttonStyle: {
    backgroundColor: '#448cfc',
    borderWidth: 0,
    color: '#FFFFFF',
    borderColor: '#307ecc',
    height: 40,
    alignItems: 'center',
    borderRadius: 30,
    marginLeft: 35,
    marginRight: 35,
    marginTop: 40,
    width: 200,
  },
  editar: {
    width: 200,
    height: 45,
    justifyContent: 'center',
    borderRadius: 10,
    marginLeft: 250,
    marginBottom: 10,
  },
  signIn: {
    width: 120,
    height: 45,
    justifyContent: 'center',
    borderRadius: 10,
    marginLeft: 145,
  },
  textSign: {
    fontSize: 15,
    alignSelf: 'center',
  },
  textInput: {
    width: 300,
    alignSelf: 'center',
    borderRadius: 20,
  },
  textD: {
    color: '#448cfc',
    fontSize: 20,
    marginBottom: 20,
  },
  textC: {
    color: '#448cfc',
    fontSize: 13,
  },
  signIn: {
    width: 120,
    height: 45,
    justifyContent: 'center',
    borderRadius: 10,
    marginLeft: 145,
  },
  textSign: {
    fontSize: 15,
    alignSelf: 'center',
  },
  textInput: {
    width: 300,
    alignSelf: 'center',
    borderRadius: 20,
    marginTop: 20,
  },
  textD: {
    color: '#448cfc',
    fontSize: 20,
  },
  textC: {
    color: '#448cfc',
    fontSize: 13,
  },
  button: {
    alignItems: 'center',
    marginTop: 5,
  },
  signP: {
    width: 200,
    height: 50,
    justifyContent: 'center',
    alignItems: 'center',
    borderRadius: 10,
  },
  container5: {
    flex: 1,
    flexDirection: 'row',
    flexWrap: 'wrap',
    alignItems: 'flex-start',
  },
  item: {
    width: '50%',
  },
})

```

Administrador.js (Pantalla Principal)

```
//Librerias
import React from 'react'
import { ScrollView, StyleSheet } from 'react-native'
import { Appbar } from 'react-native-paper'
import { FAB } from 'react-native-paper'
import { connect } from 'react-redux'
import { closeSession } from '../utils/actions'

//Vista del Administrador
const Administrador = ({ navigation, admin }) => {
  return (
    <ScrollView>
      <Appbar.Header theme={{ colors: { primary: '#448cfc' } }}>
        <Appbar.Content
          title={admin.nombre}
          titleStyle={{ color: '#FFFFFF' }}
        />
        <Appbar.Action
          icon="exit-to-app"
          color="#FFFFFF"
          onPress={() => {
            closeSession()
            navigation.navigate('InicioUsuario')
          }}
        />
      </Appbar.Header>

      <FAB
        style={styles.fab}
        big
        label="Registrar Dermatologo"
        icon="account-plus"
        color="#FFFFFF"
        onPress={() => navigation.navigate('DermRegistro')}
      />
      <FAB
        style={styles.fab}
        big
        label="Lista de Clases"
        icon="file-plus"
        color="#FFFFFF"
        onPress={() => navigation.navigate('Clases')}
      />
    </ScrollView>
  )
}

const mapStateToProps = (state) => ({
  admin: state.admin,
})

export default connect(mapStateToProps, {})(Administrador)
const styles = StyleSheet.create({
  fab: {
    position: 'relative',
    margin: 40,
    right: 0,
    bottom: 0,
  },
  editar: {
    width: 200,
    height: 45,
    justifyContent: 'center',
    borderRadius: 10,
    marginLeft: 200,
    marginBottom: 10,
  },
})

```

Clases.js (Lista de clases)

```
//Librerias
import React, {useState, useEffect} from 'react'
import { ScrollView, Text, View , TouchableOpacity, StyleSheet, Alert} from 'react-native'
import { Title} from 'react-native-paper';
import { Button} from 'react-native-paper';
import { Input} from 'react-native-elements';
import LinearGradient from 'react-native-linear-gradient';
import { filter} from 'lodash'
import firebase from '../firebase/fire';

    //Vista de las clasaes
    const Clases = () => {

        useEffect(() => {
            DataClase()
        },[clases])

        const admin = firebase.database().ref('/Administrador/')
        const  [clase, setClase] = useState('')
        const  [clases, setClases]= useState([])
        const  [mostrar, setMostrar]= useState(false)

        //Agregar clase
        const nuevaclase = () =>{
            let clasesrr = []
            if (clase === null) {
                Alert.alert('Ingre una nueva clase')

            }
            else {
                setClase(null)
                clasesrr.push( ... clases,clase)
                setClases(clasesrr)
            }

        }
        //Guardar clases
        const guardar = () =>{

            admin.set({
                elementos: clases
            })


        }
        //Mostrar los elementos  existentes
        function DataClase(){
            admin.on('value', function(snap){
                let a_ = snap.val();
                const inform = a_.elementos
                setClases(inform)

            } )


        }
```

```
1
2 //Eliminar clase
3 const eliminar = (item) => {
4     Alert.alert(
5         "Eliminar Imagen",
6         "¿Estas seguro que quieres este elemento?",
7         [
8             {
9                 text: "No",
10                style: "cancel"
11            },
12            {
13                text: "Si",
14                  onPress: () => {
15                    setClases(
16                        filter(clases, (itemid) => itemid =/= item)
17                    )
18
19                }
20
21            }
22        ],
23        { cancelable: false }
24    )
25 }
26
27        return (
28            <ScrollView>
29            <Title style={{alignSelf:'center', color:'#448cfc'}}>Lista de Clases</Title>
30            <View
31            style={{
32                marginTop:10,
33                borderBottomColor: '#8ab5fb',
34                borderBottomWidth: 8,
35            }}
36            />
37
38            {mostrar ?   <View>
39
40                <Input style={{marginTop:20}}
41                keyboardType='default'
42                placeholder='Nueva clase'
43                onChangeText={(clase) => setClase(clase)}
44                value={clase}
45                />
46        <View style={styles.container}>
47        <TouchableOpacity
48          style={styles.button}
49          onPress={nuevaclase}>
50        <LinearGradient
51              colors={['#6cacfc', '#448cfc']}
52              style={styles.signIn}
53          >
54              <Text style={[styles.textSign, {
55                  color:'#fff'
56              }]}>Agregar</Text>
57          </LinearGradient>
58        </TouchableOpacity>
59
60        </View>
61
62
```

```
   {clases.map((item) => {

                return (
                   (

                           <View key ={item.key}>
                               <Button  mode="outline" onPress={() => eliminar(item)} theme={{colors:
 {primary:'#448cfc'}}}>
                             {item}
                             </Button>

                           </View>


                    )

                         )
            })}

          <View style={styles.container}>
        <TouchableOpacity
          style={styles.button}
          onPress={() => {guardar(),setMostrar(false)}}
         >
        <LinearGradient
              colors={['#6cacfc', '#448cfc']}
              style={styles.signIn}
          >
              <Text style={[styles.textSign, {
                  color:'#fff'
              }]}>Guardar</Text>
          </LinearGradient>
        </TouchableOpacity>

        </View>

            </View>:   <View style={styles.container}>
            {clases.map((item) => {

           return (
              (

                     <View>
                        <Button  mode="outline" onPress={() => eliminar(item)} theme={{colors:
 {primary:'#448cfc'}}}>
                       {item}
                       </Button>

                     </View>


              )

                    )
       })}
        <TouchableOpacity
          style={{marginTop:20}}
          onPress={() => {setMostrar(true)}}>
        <LinearGradient
              colors={['#6cacfc', '#448cfc']}
              style={styles.signIn}
          >
              <Text style={[styles.textSign, {
                  color:'#fff'
              }]}>Editar lista</Text>
          </LinearGradient>
        </TouchableOpacity>

        </View>}

    </ScrollView>

        )

    }

export default Clases

const styles = StyleSheet.create({
    container:{
        flex:1,
        alignItems: 'center',
        padding: 30,

    },
    signIn: {
        width: 200,
        height: 45,
        justifyContent: 'center',
        alignItems: 'center',
        borderRadius: 10,

    },
    textSign: {
        fontSize: 17,
    }
})

```

DermRegistro.js (Registro de Dermatólogos)

```
//Librerias
import React , {useState} from 'react';
import {
    View,
    Text,
    TouchableOpacity,
    TextInput,
    Platform,
    StyleSheet,
} from 'react-native';

import LinearGradient from 'react-native-linear-gradient';
import firebase from '../firebase/fire';
import RNPickerSelect from 'react-native-picker-select';

//Registro de dermatologos
const Registro = ()=>{


    const [email, setEmail] = useState('');
    const [nombre, setNombre] = useState('');
    const [ genero, setGenero ] = useState("");
    const [pais, setPais] = useState('');
    const [password, setPassword] = useState('1234567');
    const [error, setError] = useState('');
    const [ lugar, setLugar ] = useState("");
    const [ cuidad, setCuidad ] = useState("");

    //Crear y Autenticar usuario Firebase
    const Registrar = async () => {

        try {
            const response = await firebase.auth().createUserWithEmailAndPassword(email, password);
            const enviarcorreo = response.user;
            const id = response.user.uid
           console.log(password)
           //Enviar correo de verificacion de Usuarios
            enviarcorreo.sendEmailVerification().then(function() {
                console.log('Correo enviado')

                }).catch(function(error) {

                });
            console.log('Se registro correctamente')
           console.log(email)
           actualizar(id)

        } catch (err) {
            if (err.code === 'auth/invalid-email'){
                let mensaje= 'La direccion de correo electronico tiene un formato incorrecto'
                setError(mensaje);
            } else if (err.code === 'auth/email-already-in-use') {
                let mensaje= 'La direccion de correo electronico ya se encuentra registrada'
                setError(mensaje);
            }
            else if (err.code === 'auth/weak-password'){
                let mensaje= 'La contraseña debe tener al menos 6 caracteres'
                setError(mensaje);
            }

        }
        //Validar campos vacios y Guardar la informacion del usuario
        function actualizar (id) {

            if(email  === ''){
                alert('Por favor ingrese su edad')
            } else if (pais === '') {
                alert('Por favor ingrese el pais')
            } else if (nombre === ''){
                alert('Por favor nombre')
            } else if (cuidad === ''){
              alert('Por favor ingrese la cuidad')
            }else if (genero === '' || genero === 'no'){
              alert('Por favor seleeccione su genero')
            }
            else if (lugar=== '' ){
                alert('Por favor seleeccione su genero')
              }
              const dermatologos = firebase.database().ref('/Usuarios/'+id)
              dermatologos.set({
                datospersonales: {genero:genero,
                    correo: email,
                    nombre:nombre,
                    pais: pais,
                    cuidad: cuidad,
                    entidad:lugar,
                    done: true,
                    },
                rol: 2
            })
            console.log('Se guarda correctamente en Firebase')

        }


```

```
return (
       <View style={styles.container}>

       <View style={styles.action}>
           <TextInput
               placeholder="Correo Electronico"
               //placeholderTextColor="#666666"
               style={[styles.text_footer, {
                           /* definir color*/
               }]}
               autoCapitalize="none"
               value={email}
               onChangeText={setEmail}
               keyboardType='email-address'
           />


       </View>
       <View style={styles.action}>
       <TextInput
               placeholder="Nombre"
               style={[styles.text_footer, {

               }]}
               autoCapitalize="none"
               value={nombre}
               onChangeText={setNombre}

           />

       </View>
       <View style={styles.action}>
       <TextInput
               placeholder="Pais"
               style={[styles.text_footer, {

               }]}
               autoCapitalize="none"
               value={pais}
               onChangeText={setPais}

           />

       </View>

       <View style={styles.action}>
       <TextInput
               placeholder="Cuidad"
               style={[styles.text_footer, {

               }]}
               autoCapitalize="none"
               value={cuidad}
               onChangeText={setCuidad}

           />

       </View>
       <View style={styles.action}>
       <TextInput
               placeholder="Entidad de Salud"
               style={[styles.text_footer, {

               }]}
               autoCapitalize="none"
               value={lugar}
               onChangeText={setLugar}

           />

       </View>


```

```
 <Text style={{fontSize: 15  , color: '#448cfc' ,  alignSelf:
'baseline',marginBottom:10,marginTop:10}}>Genero</Text>
            <RNPickerSelect
                placeholder={{}}
                onValueChange={(genero) => setGenero(genero)}
                items={[
                    {label: "Genero", value: "no" },
                    {label: "Masculino", value: "Masculino" },
                    {label: "Femenino", value: "Femenino" },
                ]}/>

        {
                    error ?
                        <Text style={{ color: 'red' }}>{error}</Text>
                        : null
                }

        <View style={styles.button}>
            <TouchableOpacity
                style={styles.signIn}
                onPress={() => Registrar()}
            >
            <LinearGradient
                colors={['#448cfc', '#448cfc']}
                style={styles.signIn}
            >
                <Text style={[styles.textSign, {
                    color:'#fff'
                }]}>Terminar Registro</Text>
            </LinearGradient>
            </TouchableOpacity>

        </View>

    </View>
    )



};


export default Registro;

const styles = StyleSheet.create({
    container: {
      flex: 2,
      backgroundColor: '#FFFFFF'
    },
    header: {
        flex: 1,
        justifyContent: 'flex-end',
        paddingHorizontal: 20,
        paddingBottom: 50
    },
    footer: {
        flex: Platform.OS === 'ios' ? 3 : 5,
        backgroundColor: '#fff',
        borderTopLeftRadius: 30,
        borderTopRightRadius: 30,
        paddingHorizontal: 20,
        paddingVertical: 30
    },
    text_header: {
        color: '#fff',
        fontWeight: 'bold',
        fontSize: 30
    },
    text_footer: {
        color: '#05375a',
        fontSize: 16
    },
    action: {
        flexDirection: 'row',
        marginTop: 10,
        borderBottomWidth: 2,
        borderBottomColor: '#f2f2f2',
        paddingBottom: 5
    },
    textInput: {
        flex: 1,
        marginTop: Platform.OS === 'ios' ? 0 : -12,
        paddingLeft: 10,
        color: '#05375a',


    },
    button: {
        alignItems: 'center',
        marginTop: 50
    },
    signIn: {
        width: '100%',
        height: 50,
        justifyContent: 'center',
        alignItems: 'center',
        borderRadius: 10
    },
    textSign: {
        fontSize: 18,
        fontWeight: 'bold'
    },
    textPrivate: {
        flexDirection: 'row',
        flexWrap: 'wrap',
        marginTop: 20
    },
    color_textPrivate: {
        color: 'grey'
    }
  });
```

Rutas.js (En este script se define las rutas de navegación de la aplicación)

```
import React from 'react'
import { createStackNavigator } from '@react-navigation/stack'

//Pantallas
import Inicio from './Inicio'
import InicioUsuario from './InicioUsuario'
import Registro from './Registro'
import Login from './Login'
import DatosRegistro from './DatosRegistro'
import Usuario from './Usuario'
import UsuarioResultados from './UsuarioResultados'
import UsuarioCorreo from './UsuarioCorreo'
import Dermatologo from './Dermatologo'
import UsuarioPago from './UsuarioPago'
import PerfilUsuario from './PerfilUsuario'
import UsuarioGeneral from './UsuarioGeneral'
import Resultados from './Resultados'
import DetalleResultado from './DetalleResultado'
import Administrador from './Administrador'
import DermRegistro from './DermRegistro'
import Clases from './Clases'

const RutasStack = createStackNavigator()

//Definicion de cada pantalla
const Rutas = () => (
  <RutasStack.Navigator headerMode="none">
    <RutasStack.Screen name="Inicio" component={Inicio} />
    <RutasStack.Screen name="InicioUsuario" component={InicioUsuario} />
    <RutasStack.Screen name="Registro" component={Registro} />
    <RutasStack.Screen name="Login" component={Login} />
    <RutasStack.Screen name="DatosRegistro" component={DatosRegistro} />
    <RutasStack.Screen name="Usuario" component={Usuario} />
    <RutasStack.Screen name="UsuarioResultados" component={UsuarioResultados} />
    <RutasStack.Screen name="UsuarioCorreo" component={UsuarioCorreo} />
    <RutasStack.Screen name="Dermatologo" component={Dermatologo} />
    <RutasStack.Screen name="UsuarioPago" component={UsuarioPago} />
    <RutasStack.Screen name="PerfilUsuario" component={PerfilUsuario} />
    <RutasStack.Screen name="UsuarioGeneral" component={UsuarioGeneral} />
    <RutasStack.Screen name="Resultados" component={Resultados} />
    <RutasStack.Screen name="DetalleResultado" component={DetalleResultado} />
    <RutasStack.Screen name="Administrador" component={Administrador} />
    <RutasStack.Screen name="DermRegistro" component={DermRegistro} />
    <RutasStack.Screen name="Clases" component={Clases} />
  </RutasStack.Navigator>
)

export default Rutas

```

App.js (En este script se define la configuración base de la aplicación)

```
import * as React from 'react'
import { NavigationContainer } from '@react-navigation/native'
import { Provider } from 'react-redux'
import store from './store/store'
import Rutas from './vistas/Rutas'

const App = () => {
  return (
    <Provider store={store}>
      <NavigationContainer>
        <Rutas></Rutas>
      </NavigationContainer>
    </Provider>
  )
}

export default App

```

Fire.js (En este script esta la configuración de Firebase)

```
//Libreria y configuracion de Firebase
import * as firebase from 'firebase'
import 'firebase/storage'

const firebaseConfig = {
  apiKey: 'AIzaSyC7r8oXoReIGviorWoLqGCfYmJDPhWP2h4',
  authDomain: 'skinusco-4b57a.firebaseapp.com',
  databaseURL: 'https://skinusco-4b57a-default-rtdb.firebaseio.com',
  projectId: 'skinusco-4b57a',
  storageBucket: 'skinusco-4b57a.appspot.com',
  messagingSenderId: '735515532283',
  appId: '1:735515532283:web:8dc44051762c0881adc95e',
}

if (!firebase.apps.length) {
  firebase.initializeApp(firebaseConfig)
}

export default firebase

```

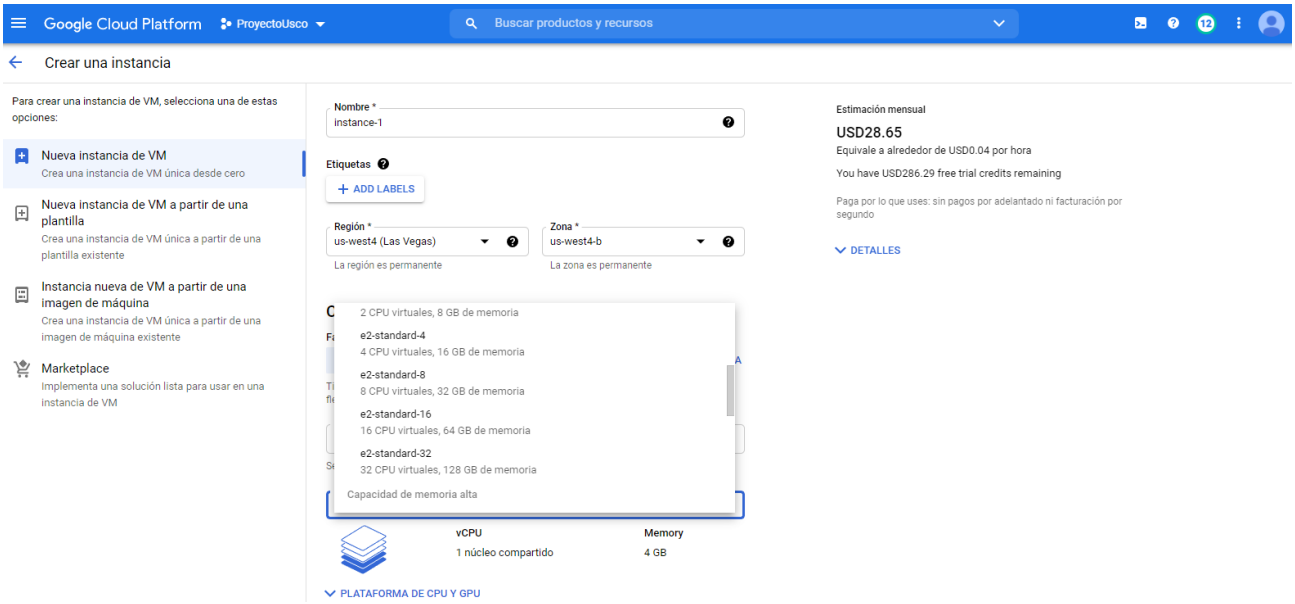
Store.js (Configuración y funciones de Redux)

```
import { createStore } from 'redux'

//Objeto inicial
const inicialState = {
  key: '',
  cuenta: '',
  datospersonales: '',
  dermatologos: '',
  usuario: '',
  admin: { nombre: 'Juan Castro', correo: 'proyectoskinusco@gmail.com' },
}

//Funciones
const reducer = (state = inicialState, action) => {
  if (action.type === 'Agregar Informacion') {
    return {
      ...state,
      key: action.key,
    }
  }
  if (action.type === 'Agregar datos') {
    return {
      ...state,
      datospersonales: action.usuario,
    }
  }
  if (action.type === 'Agregar dermatologo') {
    return {
      ...state,
      dermatologos: action.usuario,
    }
  }

  if (action.type === 'Agregar InformacionUsu') {
    return {
      ...state,
      usuario: action.nombre,
    }
  }

  return state
}

export default createStore(reducer)

```

16.4 Despliegue e Instalación de las herramientas usada en el Desarrollo

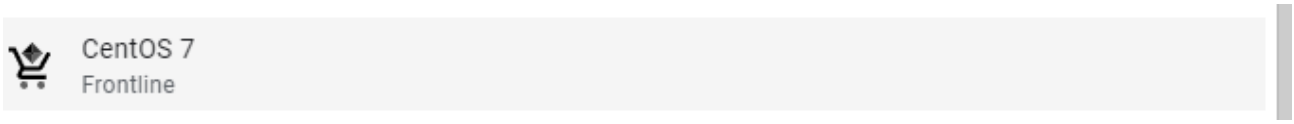
16.4.1 Crear una instancia en Google Cloud Platform

Debemos tener una cuenta activa en Google Cloud Platform

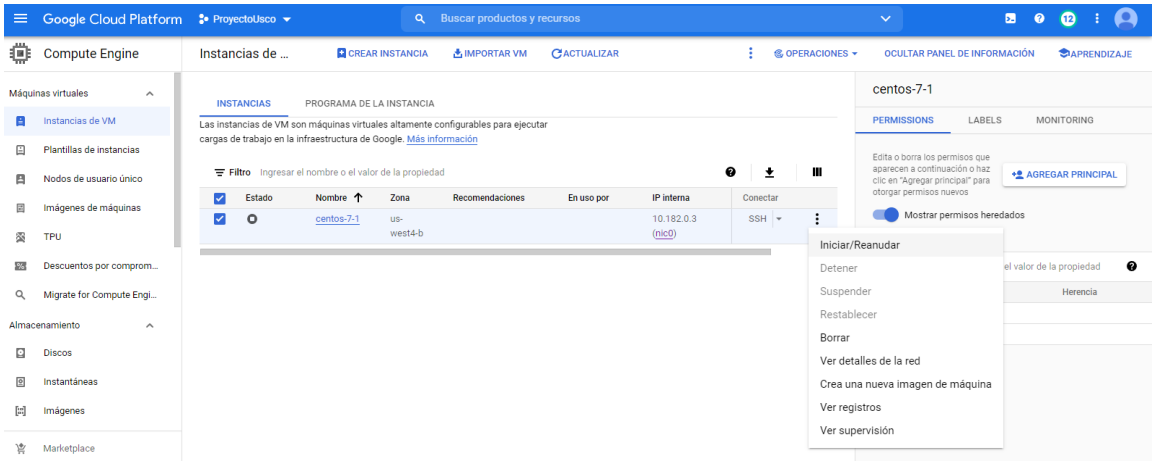


Le damos un nombre a la instancia, cambiamos el tipo de máquina por una e2-standard-4 que posee 4 CPU virtuales, y 16 GB de memoria, las demás configuraciones las dejamos por defecto.

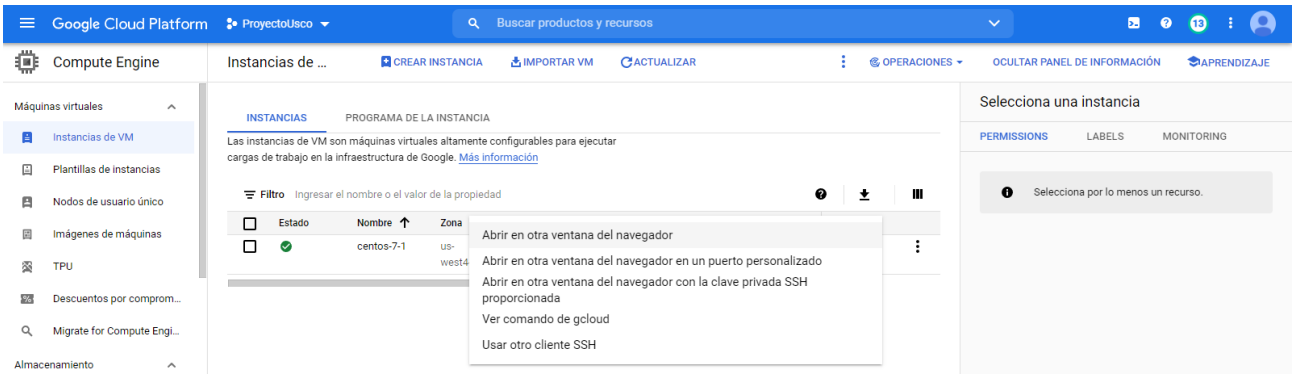
Cuando ya ha sido creada la instancia, buscamos el sistema operativo con el que vamos a trabajar y procedemos a instalarlo.



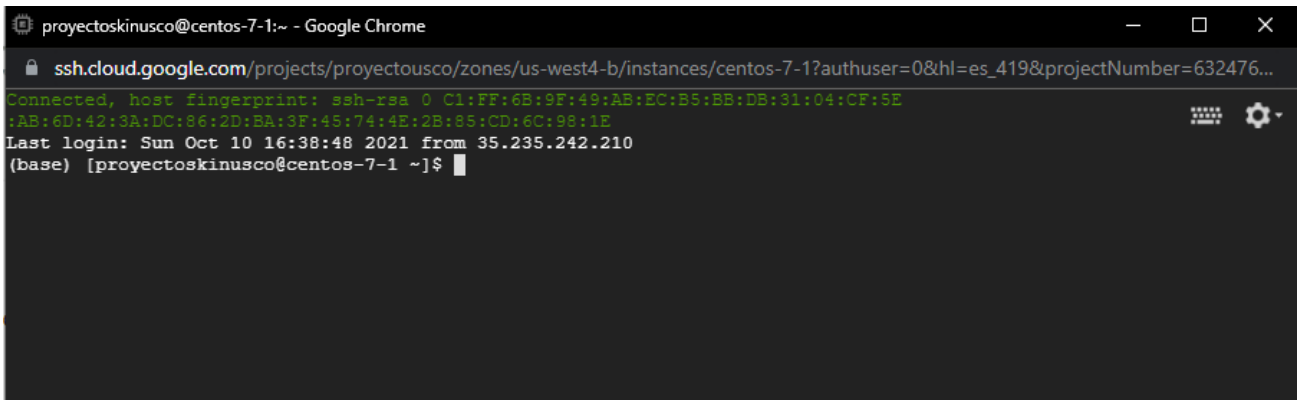
Procedemos a iniciar la instancia



Una vez iniciada podemos acceder al sistema operativo a través de una ventana del navegador.



Donde tenemos este espacio para realizar la instalaciones y configuraciones necesarias.



16.4.2 Configuración para el servidor CentOs 7 y ejecución del servicio web

sudo yum –y update (Actualizar paqueteria)

sudo yum install -y zip unzip nano git tree wget (Instalar algunas herramientas)

Instalar Miniconda

curl -LO https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh

sh ./Miniconda3-latest-Linux-x86_64.sh

source ~/.bashrc

Creamos un espacio de trabajo con Miniconda e instalamos Python, luego lo activamos

conda create -n PROD pip python=3.7.0

conda activate PROD

Procedemos a instalar todas librerías y paquetes necesarios

pip install --no-cache-dir tensorflow==2.3.0

pip install tensorflow-serving-api

pip install fastapi

pip install uvicorn

pip install python-multipart

pip install pillow

Creamos una carpeta donde vamos a almacenar el modelo

mkdir -p ~/models

cd ~/models

Descargamos el modelo en el sistema operativo mediante un enlace de Google Drive, de la siguiente manera.

export FILEID=1dA6T9pgFhKSoiNldsog27bgAea2UEI0g

*wget --load-cookies cookies.txt *

*"https://docs.google.com/uc?export=download&confirm=$(wget *

*--quiet *

*--save-cookies cookies.txt *

*--keep-session-cookies *

*--no-check-certificate 'https://docs.google.com/uc?export=download&id='${FILEID} *

*-O- | sed -rn 's/.*confirm=([0-9A-Za-z_]+).*/\1\n/p')&id=${FILEID}" *

-O models.zip && rm -rf cookies.txt

unzip models.zip && rm -rf models.zip && cd ~

Instalación y configuración de Docker, utilizamos los siguientes comandos.

sudo yum install -y yum-utils device-mapper-persistent-data lvm2

sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

sudo yum install -y docker-ce docker-ce-cli containerd.io

sudo systemctl start docker (Ejecutamos docker)

sudo usermod -aG docker $USER

newgrp docker

docker pull tensorflow/serving (Imagen necesaria para crear el contenedor de docker)

Instalamos Docker Compose, para realizar la configuración de docker y dar inicio a la servicio web.

sudo curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

Creamos una carpeta llamada Docker, y dentro de ella un archivo llamado

compose-config.yml

con la siguiente información

```
services:
  skinusco:
    image: tensorflow/serving
    ports:
      - 9500:8500
      - 9501:8501
    volumes:
      - ${MODEL_PB}:/models/skinusco
    environment:
      - MODEL_NAME=skinusco
```

Para ejecutar este archivo y dar inicio al servicio, debemos darle la ubicación del modelo a la variable MODEL_PB

export MODEL_PB=$(pwd)/models/tf2x/tensorflow

luego iniciamos Docker

sudo systemctl start docker

y ejecutamos la configuración anterior, con el siguiente comando.

docker-compose -f compose-config.yml up &

```
(base) [proyectoskinusco@centos-7-1 docker]$ docker-compose -f compose-config.yml up &
[1] 1713
Starting docker_skinusco_1 ... done
Attaching to docker_skinusco_1
skinusco_1  | 2021-10-11 20:16:28.086913: I tensorflow_serving/model_servers/server.cc:89] Building single TensorFl
ow model file config:  model_name: skinusco model_base_path: /models/skinusco
skinusco_1  | 2021-10-11 20:16:28.087706: I tensorflow_serving/model_servers/server_core.cc:465] Adding/updating mo
dels.
skinusco_1  | 2021-10-11 20:16:28.087743: I tensorflow_serving/model_servers/server_core.cc:591]  (Re-)adding model
: skinusco
skinusco_1  | 2021-10-11 20:16:28.188899: I tensorflow_serving/core/basic_manager.cc:740] Successfully reserved res
ources to load servable {name: skinusco version: 1}
skinusco_1  | 2021-10-11 20:16:28.188995: I tensorflow_serving/core/loader_harness.cc:66] Approving load for servab
le version {name: skinusco version: 1}
skinusco_1  | 2021-10-11 20:16:28.189019: I tensorflow_serving/core/loader_harness.cc:74] Loading servable version
{name: skinusco version: 1}
skinusco_1  | 2021-10-11 20:16:28.189095: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:38] Reading
 SavedModel from: /models/skinusco/1
skinusco_1  | 2021-10-11 20:16:28.360610: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:90] Reading
 meta graph with tags { serve }
skinusco_1  | 2021-10-11 20:16:28.360678: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:132] Readin
g SavedModel debug info (if present) from: /models/skinusco/1
skinusco_1  | 2021-10-11 20:16:28.360832: I external/org_tensorflow/tensorflow/core/platform/cpu_feature_guard.cc:1
42] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU i
nstructions in performance-critical operations:  AVX2 FMA
skinusco_1  | To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
skinusco_1  | 2021-10-11 20:16:28.988520: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:206] Restor
ing SavedModel bundle.
skinusco_1  | 2021-10-11 20:16:29.049168: I external/org_tensorflow/tensorflow/core/platform/profile_utils/cpu_util
s.cc:114] CPU Frequency: 2200210000 Hz
skinusco_1  | 2021-10-11 20:16:30.678357: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:190] Runnin
g initialization op on SavedModel bundle at path: /models/skinusco/1
skinusco_1  | 2021-10-11 20:16:31.131425: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:277] SavedM
odel load for tags { serve }; Status: success: OK. Took 2942313 microseconds.
skinusco_1  | 2021-10-11 20:16:31.229487: I tensorflow_serving/servables/tensorflow/saved_model_warmup_util.cc:59]
No warmup data file found at /models/skinusco/1/assets.extra/tf_serving_warmup_requests
skinusco_1  | 2021-10-11 20:16:31.232646: I tensorflow_serving/core/loader_harness.cc:87] Successfully loaded serva
ble version {name: skinusco version: 1}
skinusco_1  | 2021-10-11 20:16:31.234380: I tensorflow_serving/model_servers/server_core.cc:486] Finished adding/up
dating models
skinusco_1  | 2021-10-11 20:16:31.234533: I tensorflow_serving/model_servers/server.cc:367] Profiler service is ena
bled
skinusco_1  | 2021-10-11 20:16:31.237436: I tensorflow_serving/model_servers/server.cc:393] Running gRPC ModelServe
r at 0.0.0.0:8500 ...
skinusco_1  | [warn] getaddrinfo: address family for nodename not supported
skinusco_1  | [evhttp_server.cc : 245] NET_LOG: Entering the event loop ...
skinusco_1  | 2021-10-11 20:16:31.240539: I tensorflow_serving/model_servers/server.cc:414] Exporting HTTP/REST API
 at:localhost:8501 ...
(base) [proyectoskinusco@centos-7-1 docker]$
```

Ahora debemos activar nuestro entorno de trabajo y ejecutar el script de FastApi. Para esto creamos una carpeta llamada service que tendrá el script con nombre fastapi_skin.py.

conda activate PROD (Entorno de trabajo)

uvicorn fastapi_skin:app --port 9000 --host 0.0.0.0 (Ejecutamos el script)

Debemos crear también una carpeta uploads dentro de la service, y una subcarpeta images, donde almacenaremos localmente en el servidor las imágenes que ingresan al sistema.

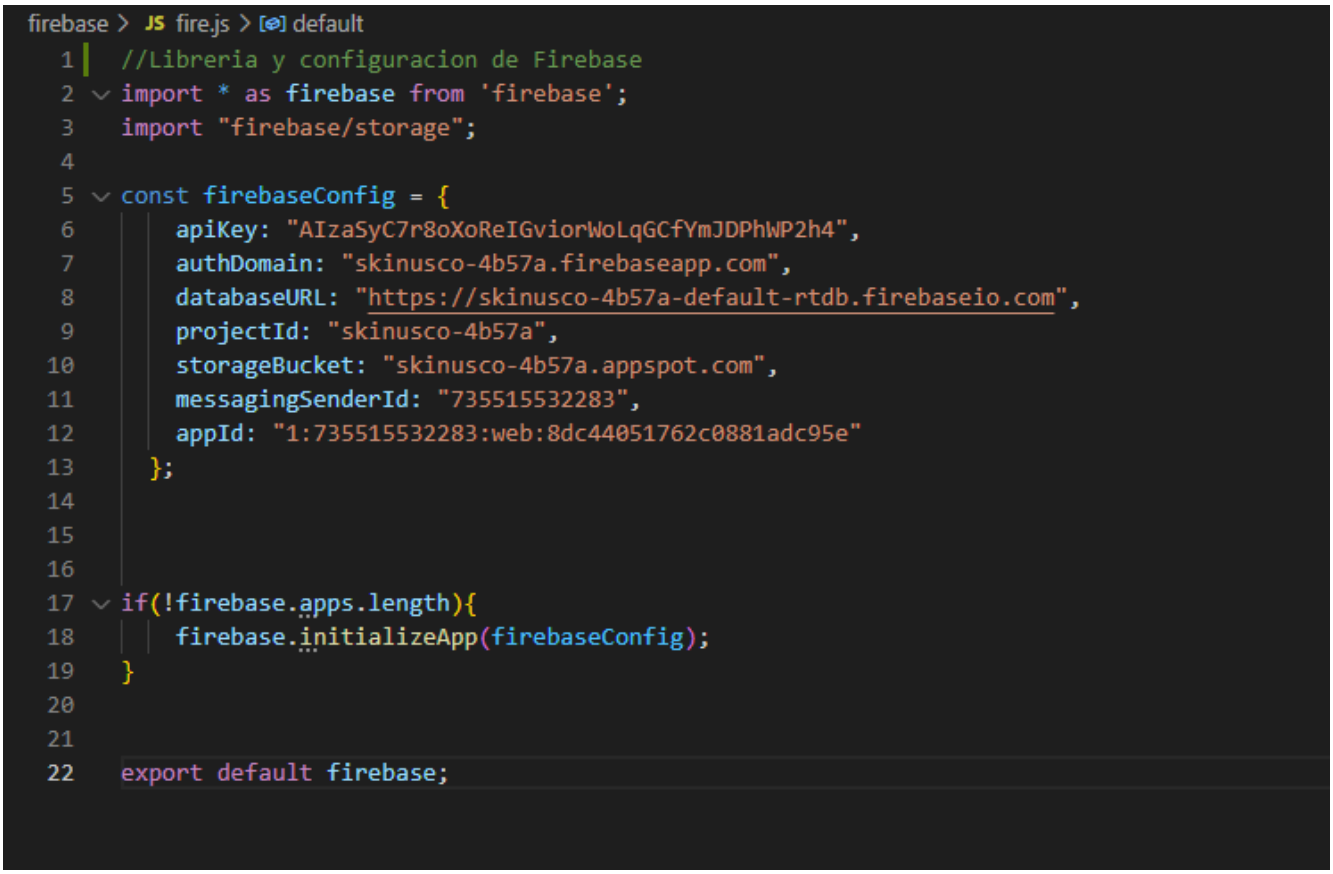
```
(base) [proyectoskinusco@centos-7-1 uploads]$ tree
.
├── cookies.txt
├── gdown.pl
├── images
│   ├── ISIC_0000052_downsampled.jpg
│   ├── ISIC_0000518.jpg
│   ├── ISIC_0001100_downsampled.jpg
│   ├── ISIC_0001118_downsampled.jpg
│   ├── ISIC_0012388_downsampled.jpg
│   ├── ISIC_0024431.jpg
│   ├── ISIC_0025818.jpg
│   ├── ISIC_0026766.jpg
│   ├── logoq.png
│   ├── readme.txt
│   └── shutterstock_163593809.jpg
```

Aquí tenemos el servicio web disponible para cualquier solicitud.

```
(PROD) [proyectoskinusco@centos-7-1 service]$ uvicorn fastapi_skin:app --port 9000 --h
2021-10-11 20:19:43.558011: W tensorflow/stream_executor/platform/default/dso_loader.c
2021-10-11 20:19:43.558053: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignor
INFO:     Started server process [1921]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:9000 (Press CTRL+C to quit)
```

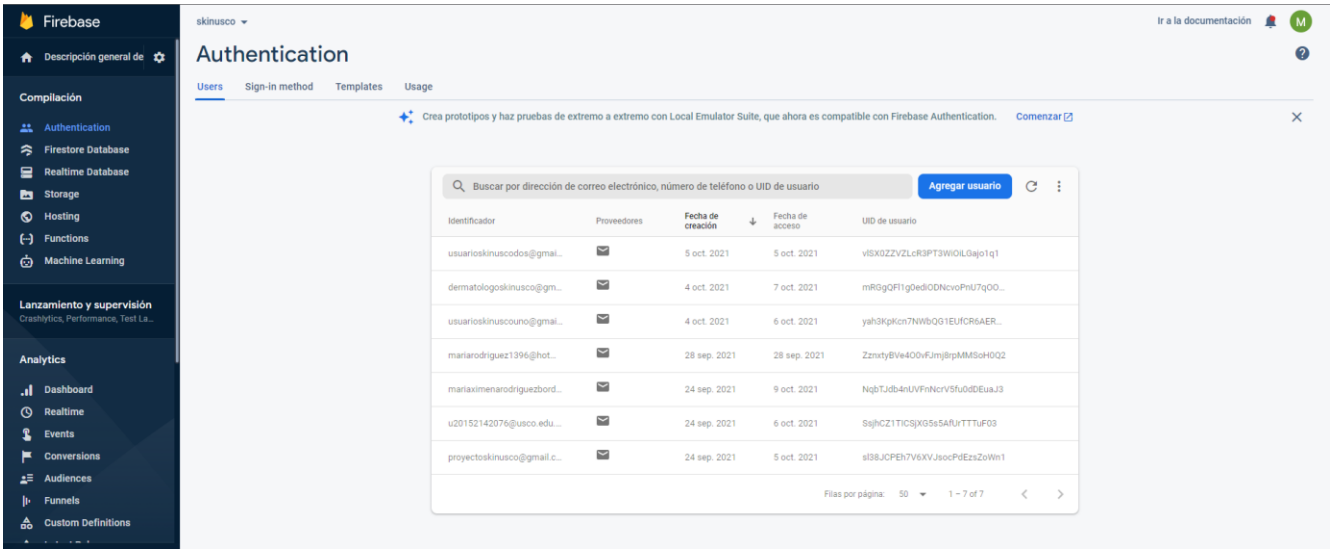
16.4.3 Configuración Firebase

Luego de crear un nuevo proyecto, tomamos la configuración que nos da firebase y enlazamos la aplicación con el siguiente script.

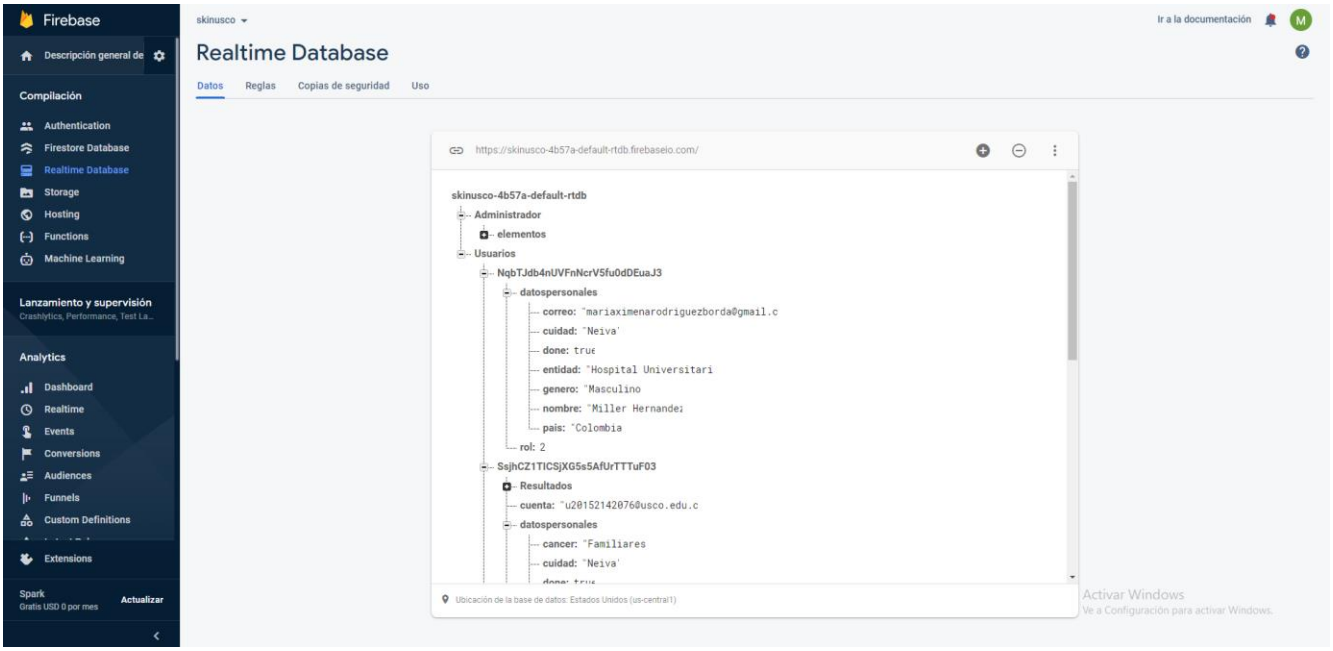


De firebase utilizaremos tres servicios como lo son:

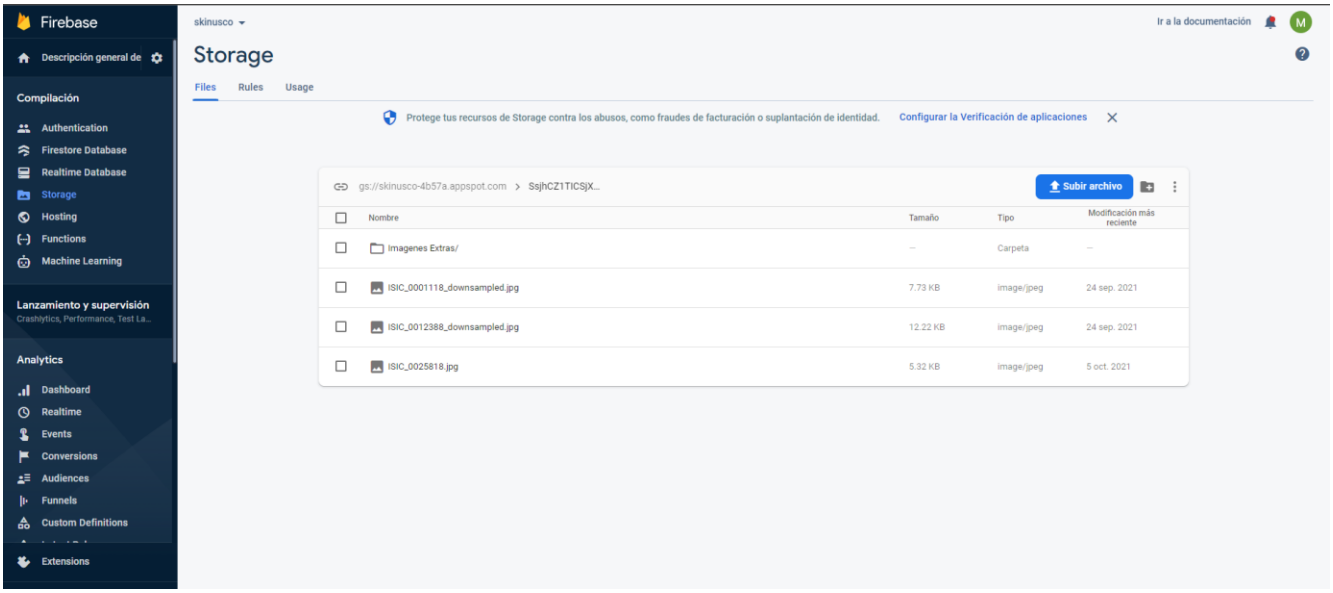
Autenticación por medio de correo electrónico y contraseña



Realtime Database es donde almacenamos la información de los usuarios en datos tipo JSON.



Storage, este servicio lo utilizamos para el almacenamiento de las imágenes que ingresan al sistema.


Firebase
Descripción general de
Compilación
Authentication
Firestore Database
Realtime Database
Storage
Hosting
Functions
Machine Learning
Lanzamiento y supervisión
Crashlytics, Performance, Test La...
Analytics
Dashboard
Realtime
Events
Conversions
Audiences
Funnels
Custom Definitions
Extensions
skinusco
Ir a la documentación
Storage
Files
Rules
Usage
Protege tus recursos de Storage contra los abusos, como fraudes de facturación o suplantación de identidad.
Configurar la Verificación de aplicaciones
gs://skinusco-4b57a.appspot.com
SsjhCZ1TICSjX...
Subir archivo
Nombre
Tamaño
Tipo
Modificación más reciente
Imagenes Extras/
Carpeta
ISIC_0001118_downsampled.jpg
7.73 KB
image/jpeg
24 sep. 2021
ISIC_0012388_downsampled.jpg
12.22 KB
image/jpeg
24 sep. 2021
ISIC_0025818.jpg
5.32 KB
image/jpeg
5 oct. 2021