COMP2606 – Software Engineering Project

CURAME

THE SKIN PREDICTION APPLICATION

By: The Curame Team.

Team Members:

Name	*ID*
Tyrese Lake	816015110
Aaron Boodoo	816013494
Josiah Jones	816012886

Course Number: INFO3604
Course Name: Project
Department: DCIT
Lecturer/Advisor: Ms. Cynthia Cudjoe
Date Submitted: 16/04/2021

Abstract

Problem/Opportunity

As a result of the crisis known as the ‘Covid-19’ pandemic currently affecting the entire globe, it has become increasingly more difficult for individuals to leave their homes in search of early medical treatment; Skin Condition Diagnosis and Treatment to be exact. When an individual develops a skin ailment within current times, they tend to wait a very long time before seeking medical attention because they live in fear of contracting the Covid-19 virus should they leave their homes. Another point to consider is, for Dermatologists, the process of diagnosing skin conditions is a long, tedious, and time-consuming one.

Solution

To resolve the aforementioned problem, this project will aim to deliver a mobile android application that is capable of making a prediction of skin diseases/ailments (80% accurate) based on user image input. The application will provide users with a general description of the predicted skin ailment & will recommend treatment for the predicted skin condition. By making this a mobile application, users are now presented with a potential means to skin-disease prediction from an early stage, should they prefer not to leave their homes. The software will also recognize features from the skin conditions so it may aid dermatologists with their diagnoses by acting as a second opinion. The application will be suitable for use by an older audience, providing enhanced visual elements and text to speech capabilities.

TABLES OF CONTENTS

Table of Figures

Introduction

Problem Description

It is crucial to diagnose signs of skin disease at an early stage to reduce the exposure to the substances causing it. Early identification allows one to determine the extent of the problem, and whether it is confined to one individual who had an existing skin problem or more widespread. By doing so, appropriate actions and treatments can be instilled if necessary, preventing the spread of the disease, and in some cases, preventing damage to many individuals. Additionally, access to dermatologists can be difficult/expensive to attain by some individuals, especially during the current Covid-19 pandemic crisis, therefore it is important to gain an idea of what the skin condition might be and determine if any precautionary action is necessary from “early o’clock”. If an individual does have a contagious skin illness, it is important for such individuals to exercise caution when seeking medical treatment to avoid spreading illnesses to other patients seeking healthcare in areas such as vehicles, waiting rooms and examination rooms. With skin disease scanning and identification applications, like Curame, early detection of skin illnesses can be achieved using one's mobile device, thus the spreading of contagious skin disease can be minimized.

Scope

This project will deliver software (android application) that is able to diagnose skin conditions based on image input and provide users with a general description of the skin condition. The software will recognize features from the skin conditions so that it can help with dermatologist diagnoses. The application will be suitable for use by an older audience providing enlarged text and text to speech.

Project Objectives

- To develop a machine learning model (Convolutional Neural Network) capable of identifying skin diseases.
- To create a mobile application that utilizes the developed model and the mobile camera/gallery to make predictions.
- To create a mobile application that provides extensive information on an identified skin illness that would be stored on an online database.
- To store the results of predictions on the device and recall them easily.
- To facilitate, in the mobile application, various visual elements and text to speech to cater to older users with impaired senses.
- To produce a working application that can be installed and used fully on a mobile device.

What Resources are Required?

Resource	Short Description
Android Coding Environment	A workspace / development environment software necessary for laying out, modulating, and building the segments of code required for application development. For this project, Android Studio was used.
Android Device	A working android device equipped with a camera of decent quality and up to date with the most recent version of the android OS is required.
Machine Learning Environment	An environment for developing and training a Convolutional Neural Network capable of predicting diseases. For this project, Google Collab was used.
Training Data	A training set, or pool of data (in the form of image libraries) is required so that it may be utilized for the training of the skin disease prediction model.

Figure 1.0: Required Resources

Risk Analysis

Risk	Description	Potential Impact
Time	The amount of time put into project implementation is directly proportional to the level of accuracy of the disease prediction model. This suggests that the larger the time frame for project completion, the more time will be put into training and refinement of the disease prediction model. This results in a higher chance of the application's model returning an accurate prediction on a user's skin condition. Similarly, if the time frame were to be reduced, so would the accuracy of the model.	High
Size of Training Data Set	The number of images supplied to the model for training is directly proportional to the level of accuracy of the disease prediction model. This suggests that the larger the number of images supplied to the application's model for training, the higher the chance of the application's model returning an accurate prediction on a user's skin condition. Similarly, if a small number of images were supplied to the model (a reduction in the training set size), the prediction accuracy would also plummet.	High

Figure 2.0: Risk Analysis

Deliverables

Deliverable	Short Description	Due Date (DD/MM/YY)
Weekly Report	A report on the project's incremental status and progress to be drafted once every week.	Weekly, from 22/01/21 to 17/04/21
Milestone 1	Project Scope Document	09/02/21
Milestone 2	Project Timeline, Work Breakdown, Use Case Guidelines, System & User Specifications & Requirements Analysis Document	23/02/21
Milestone 3	Project Design and Implementation Document	16/03/21
Mid Semester Presentation	A presentation to a 3rd party stakeholder which would give insight as to the early stages of project development (gives the audience a sense of what direction the project is heading in)	02/03/21
Fully Functional Convolutional Neural Network (TensorFlow) Model	A functional Convolutional Neural Network (CNN), made with TensorFlow, capable of accepting a skin diseases image as input and producing a prediction on what skin disease it is. This model would have an accuracy greater than 70%	29/03/21
Fine Tuned TensorFlow Lite Model	A lite version of the CNN model that can be used by Mobile Devices, such as Android phones, to make predictions.	29/03/21
Fully Functional Android Application	A fully functional Android Application that provides all the functional and non-functional requirements outlined in this document and can be installed on mobile Android devices.	05/04/21
Final Presentation & Video	Final demonstration and walkthrough of the fully implemented application to a 3rd party	15/04/21
Final Project Report	Preparation of the final documentation of the entire planning, design, implementing, testing and deployment of the project and application.	16/04/21
Final Project Website	A website external to the actual implementation of the application. This website will be used to post summarized information from the project so that external users may gain a quick insight on our system without browsing the lengthy project documentation.	16/04/21

Figure 3.0: Deliverables

Task Schedule/Work breakdown structure

To get an early understanding of the work to be done for the Curame Team developed a Work Breakdown Diagram early on, outlining the work to be done, the weightings and who is allocated to what job.

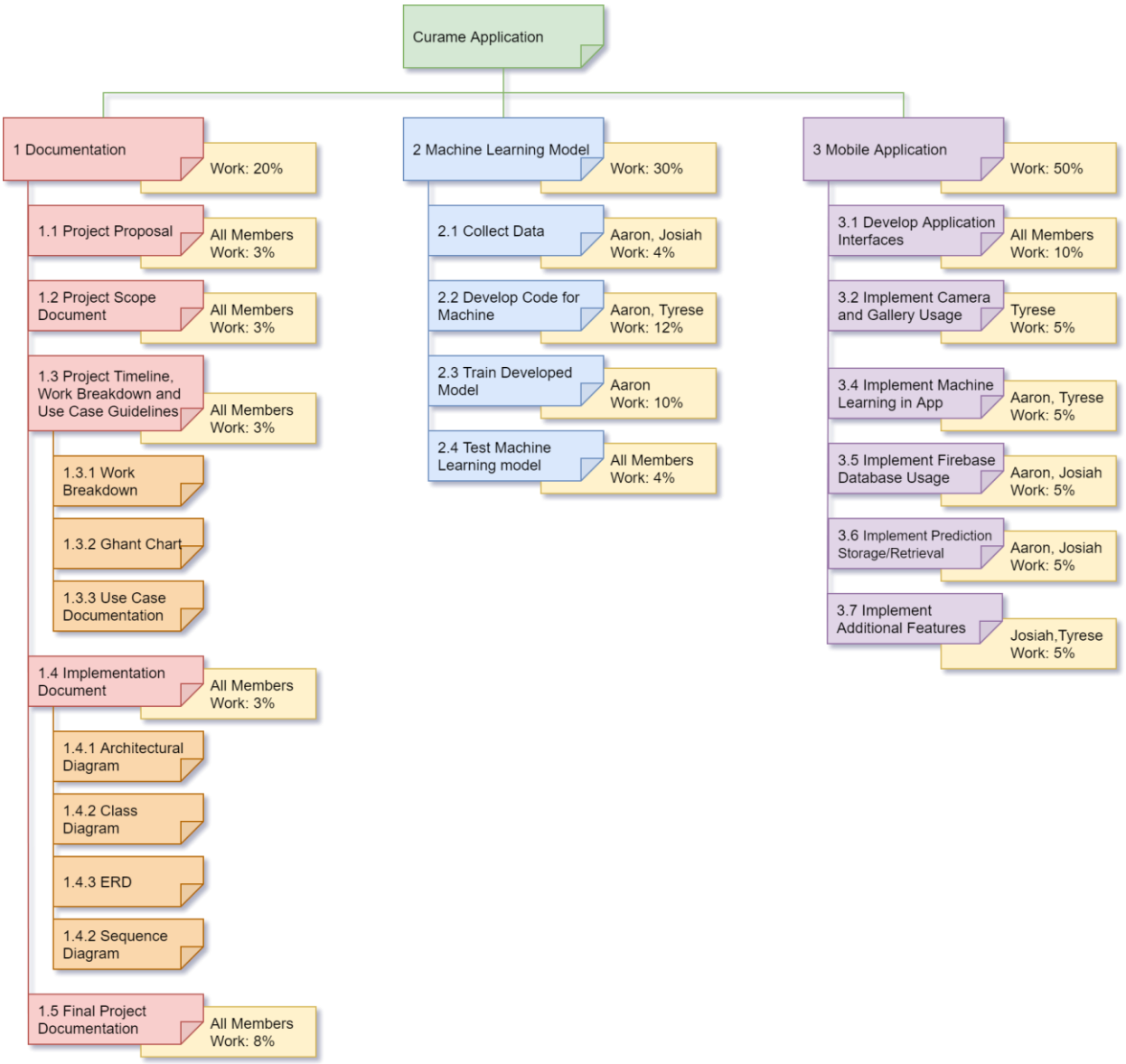


Figure 4.0: Work Breakdown

Gantt Chart

CURAME Project Start: Mon, 1/18/2021
Group Name - Curame Display Week: 00/01/1900 1:12:00 am
Group Members: Tyrese Lake
Aaron Boodoo
Josiah Jones

TASK	START	END	DAYS
0.0 Project Idea Selection	1/20/21	1/21/21	2
0.1 Project Proposal	1/21/21	1/23/21	3
1.0 Project Planning	1/25/21	1/29/21	5
1.1 Research Problem Area	1/25/21	1/27/21	3
1.2 Analyze Existing Similar Products	1/27/21	1/29/21	3
2.0 Requirements Analysis	2/1/21	2/7/21	7
2.1 Problem Statement	2/1/21	2/1/21	1
2.2 User & System Requirements	2/2/21	2/3/21	2
2.3 Non-Functional Requirements	2/4/21	2/4/21	1
2.4 Project Scope Statement	2/4/21	2/7/21	4
3.0 Project Design	2/8/21	2/14/21	7
3.1 User Stories	2/8/21	2/9/21	2
3.2 Users & Use Cases	2/8/21	2/11/21	4
3.3 Sequencing Diagrams	2/12/21	2/12/21	1
3.4 Class Diagram	2/12/21	2/13/21	2
3.5 Architectural Diagram	2/14/21	2/14/21	1
4.0 Project Implementation Stage 1	2/15/21	2/23/21	9
4.1 Gather Data for Machine Learning	2/15/21	2/17/21	3
4.2 Develop Code for Machine Learning Model	2/16/21	2/20/21	5
4.3 Train Developed Model	2/18/21	2/22/21	5
4.6 Test Machine Learning Model	2/21/21	2/23/21	3
5.0 Project Implementation Stage 2	2/24/21	3/2/21	7
5.1 Develop Application Interface	2/24/21	2/27/21	4
5.2 Implement Camera Useage	2/27/21	3/1/21	3
5.3 Display Pictures Taken	2/28/21	3/2/21	3
6.0 Project Implementation Stage 3	03/03/2021	3/11/21	9
6.1 Implement Machine Learning Model	03/03/2021	3/9/21	7
6.2 Test Machine Learning Model	07/03/2021	3/9/21	3
6.3 Add Information on Skin Desease	3/9/21	3/11/21	3
7.0 Project Implementation Stage 4	12/03/2021	3/21/21	10
7.1 Implement Text-to-Speech	12/03/2021	3/13/21	2
7.2 Modify Final Application Appearance	3/14/21	3/15/21	2
7.3 Deploy and Test on Android Device	3/16/21	3/16/21	1
7.4 Fine Tuning of Application	3/17/21	3/21/21	5
8.0 Final Project Documentation	3/22/21	3/25/21	4

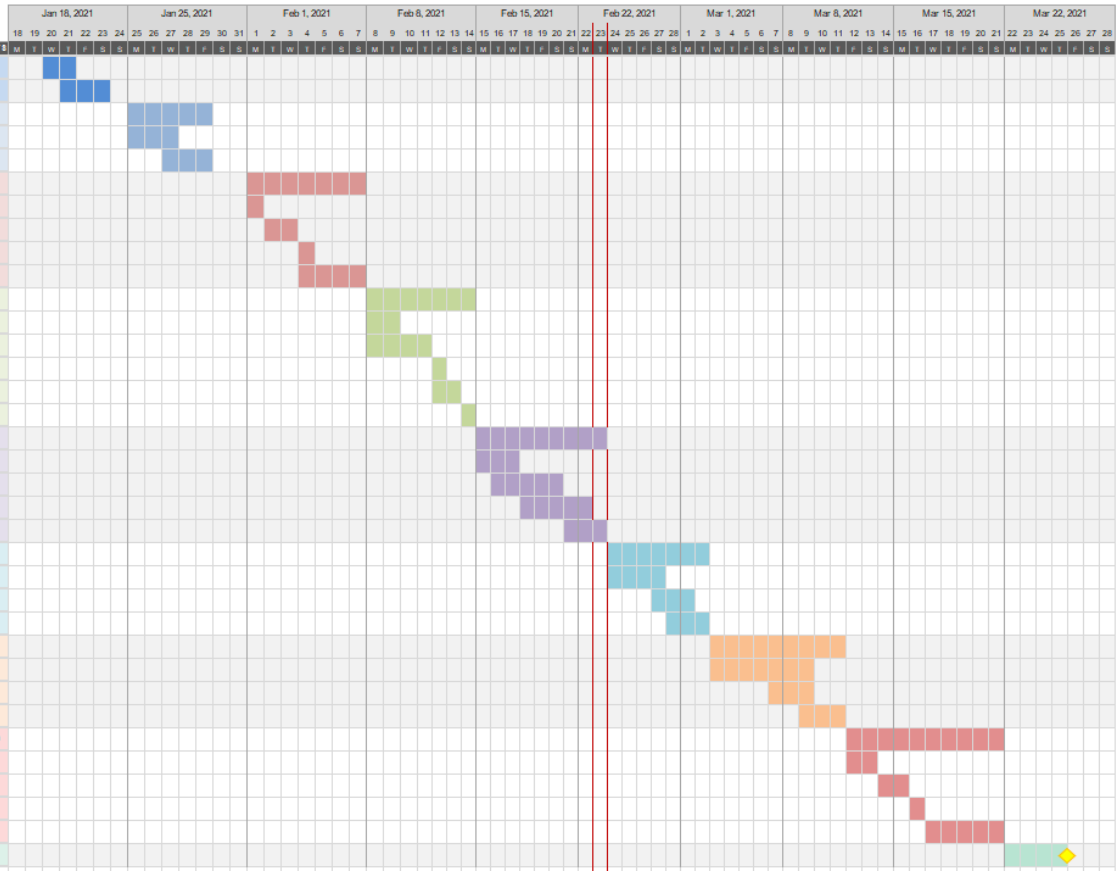


Figure 5.0: Gantt Chart

Positioning

The Curame app targets individuals with skin diseases, therefore most people worldwide would be a potential stakeholder of the application. By effectively advertising and obtaining endorsements from certified dermatologists to use the app, Curame has the potential to gain a firm footing as an actual medical resource for skin disease diagnostics.

Thorough research was done by the Curame Dev Team on the existing competition (other skin disease classification apps). All competition on the Google Playstore (e.g. Medgic) does not offer diagnostics based on deep neural networks, instead they take the manual approach of comparing images that look similar to the picture taken from an online database. Our application is unique in the sense that it looks at the features found from the disease image and makes a disease prediction based on the features found, rather than formulating that prediction solely off of visually similar images. Because of the nature of the application's model, it can be continuously updated to be adaptable to a multitude of skin diseases, making it somewhat futureproof.

Stakeholders & Target Audience

Project Development Team

The development team is the most invested in the project. The project development team is responsible for the design, implementation, and testing of the Curame application.

General Public

The public is the most important stakeholder as the app is catered towards public use by offering a free reliable way to screen for potential skin diseases. The app is not meant to be a final diagnosis but aims to give insight to the public on whether they should make the decision to have their skin ailment looked at by a dermatologist or not. This being said, the general public would be the stakeholder who holds the most value within the Curame app. They will be using the app as a mass screening device to receive information on skin conditions they may potentially have.

Dermatologists

Dermatologists also are key stakeholders who hold a special place within our system's design, as dermatologists using the app give validity to the Curame application as a genuine tool for use in the medical field. Dermatologists using our application will also aid in improving the accuracy of the model used in the application as we hope to have them use the app in tandem with their diagnoses (as a 2nd hand tool). By withdrawing their insight on diagnoses, we can obtain information on diagnoses to further help improve the application's performance. Future implementations aim to move from a model centric AI to a data centric AI so that we may compile skin disease images and classifications on a large scale to support training of the model.

Product Overview

Curame will be a skin disease scanning and identification application for mobile devices. This application will be suitable for individuals ages 13 and older. Users will be able to scan their own, or someone else's, skin illness and receive a and receive a prediction of what the illness might be as accurately as possible. Users may also receive information such as the severity of the illness, expected symptoms, some medical advice, causes and known treatments. Users should also be able to view the most recent predictions made by the application, as well as utilize text to speech to read the names and descriptions of illnesses aloud. The application's display will not rely on colors as a guide from feature-to-feature for ease of use by color blind users. This application, however, will not replace the need for dermatologists, but instead would be strictly intended for medical and healthcare assistance.

In the healthcare industry Curame can be used as a reference point as to what a skin disease may be. It can aid dermatologists in a more accurate diagnosis for patients' skin diseases. Not only can it aid them in diagnosing diseases, but it provides a valid second opinion to already diagnosed skin diseases. Curame can also be used by the public to get a second opinion on their skin diseases as second opinions are usually expensive. Curame will bring an overall increase to the safety and assurance of an individual's medical state and wellbeing, thus giving rise to a more streamlined medical diagnosis of skin diseases using artificial intelligence

Requirements Specification

Before designing the system, the Curame team came together and brainstormed to gather requirements for the system. Requirements were also gathered through research into current image classification, particularly in health care, as well as through reviewing research papers on machine learning in dermatology. After this was done, the team gathered several functional requirements and non-functional requirements.

Functional Requirements

User Requirements

1. The User shall be able to take a scan/image of the potentially affected area of the skin
    - 1.1. Users shall have access to a Use Camera Feature
    - 1.2. Users shall have the option to retake the image where necessary.
2. The User shall be able to select an image stored on their phone from the picture gallery.
    - 2.1. Users shall have access to a Upload Image from Gallery Feature
    - 2.2. Users shall have a preview of the image before selecting.
    - 2.3. Users shall have the option to re-select an image where necessary.
3. The User shall be able to view detailed information on a predicted skin illness.
    - 3.1. Users shall view a comprehensive description of a selected skin illness.
    - 3.2. Users shall view a list of potential symptoms of a selected skin illness.
    - 3.3. Users shall view a list of available treatments for a selected skin illness.
4. The User shall be able to view a list of their recent skin scan history.
    - 4.1. Users shall be able to view when a scan was made
    - 4.2. Users shall be able to view details on the prediction and information on the skin illnesses from the history view if needed.
    - 4.3. Users shall be able to delete items from the scan history if needed.
5. The User shall be able to select text-to-speech to have the application's text read to them.

System Requirements

1. The System shall use machine learning classification to predict a skin disease given an image from the camera/gallery.
    - 1.1. The system shall be able to utilize a tensor flow lite model to make predictions for skin diseases.
    - 1.2. The system should have a prediction rate of, on average, 80%
    - 1.3. The prediction should compose of the highest confidence prediction as well as any other major prediction confidence, where necessary.
    - 1.4. All relevant predictions must be displayed on the screen with their confidence percentage.
2. The System shall store images and details on previous predictions made and allow for easy/removal retrieval of this information.
    - 2.1. The system shall store the files locally on the Users Android device.
    - 2.2. The system shall store each prediction in their own separate directory with a unique name.
    - 2.3. The system shall store each prediction with the respective image.
    - 2.4. The system shall perform ‘clean ups’ on empty files and folders after an image is taken, uploaded or canceled and when a prediction is deleted.
3. The System shall be able to fetch information on skin diseases from an online database.
    - 3.1. The system shall be compatible with and utilize Firebase Firestore to store and retrieve detailed information on a skin disease.
    - 3.2. The system shall output the correct error messages if the information, for some reason, was not received.
    - 3.3. The system shall output the correct error messages if the server, for some reason, is not available.
4. The system shall minimize the RAM usage when rendering images.
    - 4.1. The system shall scale down images by rendering images with the minimum amount of pixel per inch to represent an image in each frame.
    - 4.2. The system shall fit the images to any required view.

Non-Functional Requirements

1. **Usability** - The program shall have a layout with a monochromatic scheme that is easy to use and feature various visual elements and text to speech for users with impaired senses. The system language and text-to-speech capability shall be English (Trinidad and Tobago).
2. **Compatibility and Portability** - The system shall work all Android Devices that have version SDK 19 and up.
3. **Mobile Requirements** - The system shall only require a minimum of 100MB of RAM and 1GB of storage.
4. **Performance** - System shall process picture in at maximum 5 second and have a maximum of 2 seconds load time. Minimize frame dropouts when rendering images and animations.
5. **Response Time** - The response time to fetch information from the online database shall be a maximum of 5 seconds.
6. **Availability** - The availability of the online database shall be up 24 hours a day with a maximum of 30 minutes down time every 2 weeks.

User Stories

To gather information on the system requirements, the team constructed several User Stories outlining what the system should do to be considered working and complete. Later, these user stories would be the premise for our system testing.

Simple Use Cases from User Stories

1. Scan Skin Illness
2. Select Image of Skin Illness Directly from Device
3. Read Skin Illness Details with Text to Speech
4. View Scan History

Simple Use Case 1:

Use Case Name	Scan Skin Illness
Actor/s	Any user of the 'Curame' mobile application
Basic Flow	The user opens the 'Curame' mobile application. The user then selects the 'Scan Skin' option from the home screen. The user then takes a clear image of the affected area. The user may then choose to keep the image taken or discard the image and take another. The system scans and processes the image taken and predicts what it may be. The user is then presented with 1 (or more) options on what the skin illness was predicted to be. The user may then select the image to obtain detailed information on the predicted skin illness.

Figure 6.0: Scan Skin Illness Use Case

Simple Use Case 2:

Use Case Name	Select Image of Skin Illness Directly from Gallery
Actor/s	Any user of the ‘Curame’ mobile application
Basic Flow	The user opens the ‘Curame’ mobile application. When presented with the home screen of the app, the user then selects the ‘upload an image’ option. The user selects an image from their mobile device’s gallery (the image selected must be a snapshot of the affected area on the user’s skin or else the image will not be identified as valid). If the image is determined valid, the system scans and processes the image taken and predicts what it may be. The user is then presented with 1 (or more) options on what the skin illness was predicted to be. The user may then select the image to obtain detailed information on the predicted skin illness.

Figure 7.0: Select Image from Gallery Use Case

Simple Use Case 3:

Use Case Name	Read Skin Illness Details with Text-to-Speech
Actor/s	Any user of the ‘Curame’ mobile application that is disabled or possesses a medical condition that hinders their vision
Basic Flow	The user opens the ‘Curame’ mobile application. A makes a scan, or selects a previous scan from history, then selects an illness to view information on that illness. Whilst viewing the information, the user may select the text to have the description be read aloud to them.

Figure 8.0: Use of Text-to-Speech Use Case

Simple Use Case 4:

Use Case Name	View Scan History
Actor/s	Any user of the ‘Curame’ mobile application
Basic Flow	The user opens the ‘Curame’ mobile application. The user selects the ‘View History’ option from the selections available on the application’s home screen. They then browse the list of previous scans sorted by date. The user may then select an item to view the predicted illnesses and then process to viewing information on these illnesses. When browsing previous scans, the user may also select the ‘trash’ icon atop each entry to delete the item from their history.

Figure 9.0: Scan History Use Case

Ranking Use Cases

Using the Use Cases derived from the User stories, the team then ranked the Use Cases based on the following criteria below. After these were ranked, the team determined the priority that this use case should be developed and at what build cycle (iteration) it would be developed at.

Evaluates use cases on a scale of 1-5 against 6 criteria.

1. Significant impact on architectural design.
2. Easy to implement but contains significant functionality.
3. Includes risky, time-critical, or complex functions.
4. Involves significant research or new or risky technology.
5. Includes primary business functions.
6. Will increase revenue or decrease costs.

Use Case Name	Ranking Criteria, 1 to 5						Total Score	Priority	Build Cycle
	1	2	3	4	5	6			
Scan Skin Illness	5	5	5	5	5	5	**30**	High	1
View Scan History	5	5	3	3	5	3	**24**	Medium	2
Select Image of Skin Illness Directly From Device	5	5	3	3	4	3	**23**	Medium	2
Fetch Information from an Online Database	2	2	5	4	4	4	**21**	Medium	3
Read Skin Illness Details with Text To Speech	3	2	2	2	1	1	**11**	Low	4

Figure 10.0: Use Case Priority Matrix

System Diagram

To gain a better understanding of the high-level context of the system diagram, a Context Level Data Flow Diagram was constructed. This diagram helped the team to understand what external entered were interacting with the Curame system.

Additionally, this context level diagram was further expanded to a Level-1 Data Flow Diagram was constructed to show where these external entities interact with the Curame System as well as how the inner subsystems of the Curame System interact with each other and the inner datastores.

Context Level System Diagram

As seen in Figure 11.0, the User Entity would send images to the Curame system and receive Predictions and Information's as well as send Commands to the Curame System to make changes. Also, the Curame System sends Requests to the Database Entity and receives Information. Lastly Skin Disease Data Set Entity would send Images and Classifications to the Curame System which would be later shown to be used for Machine Learning Training.

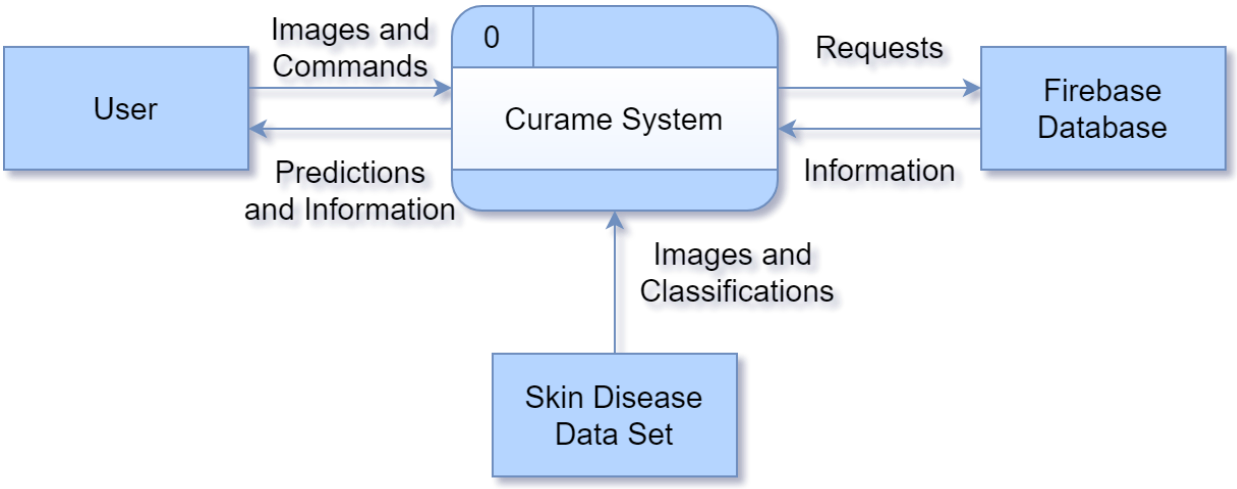


Figure 11.0: Context Level System Diagram.

Level-1 System Diagram

As seen in Figure 12.0 below, the Entities User, Skin Disease Data Set and Firebase Database interface with the Curame System through the various Subsystems (Image Input System, Image Classification System, Skin Disease Information System and, Prediction History System). These systems would communicate data between each other to carry out System Activity as well as utilizing the System Datastores (Disease Labels and Predictions).

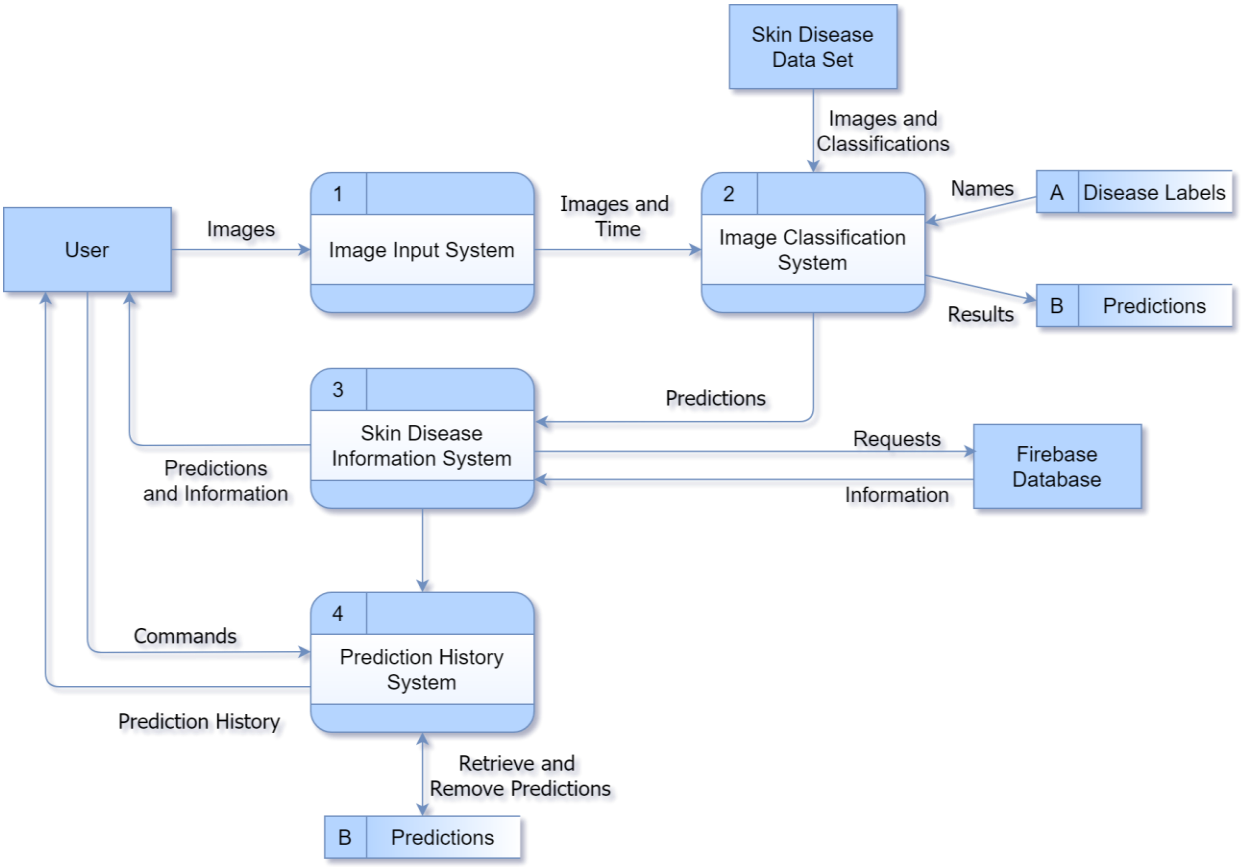


Figure 12.0: Level-1 System Diagram

Use Case Diagram

The Team constructed a Use Case Diagram of the soon to be Curame System to further gather requirements.

As seen in Figure 13.0, the external Users/External Systems are Users and an Online Database. These Users/External Systems may access the system through the various outlines Use Cases which may include additional use cases that the system will perform. Some Use Cases may also extend to other use cases also accessible by the User.

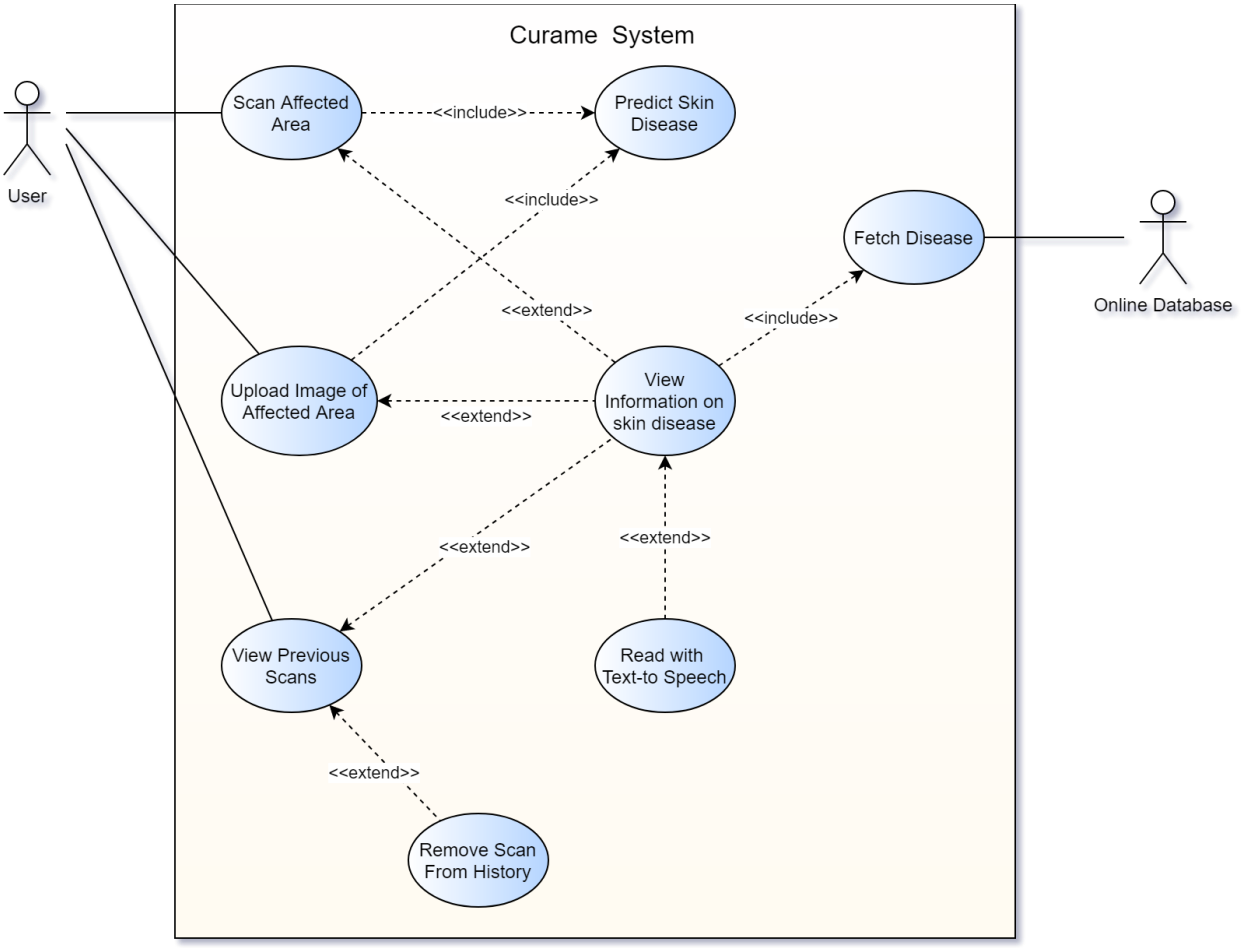


Figure 13.0: Use Case Diagram

Sequence Diagrams

In requirements gathers, the sequence diagram below was constructed describing the process of each task that users can perform when using the Curame system, namely disease prediction by scanning, disease prediction by uploading an image and viewing scan history. The diagram displays the external User, interacting with the Curame Interface on a mobile android device and all the inner components of the application exchanging information between each other to accomplish the given task.

Disease Prediction by Scanning

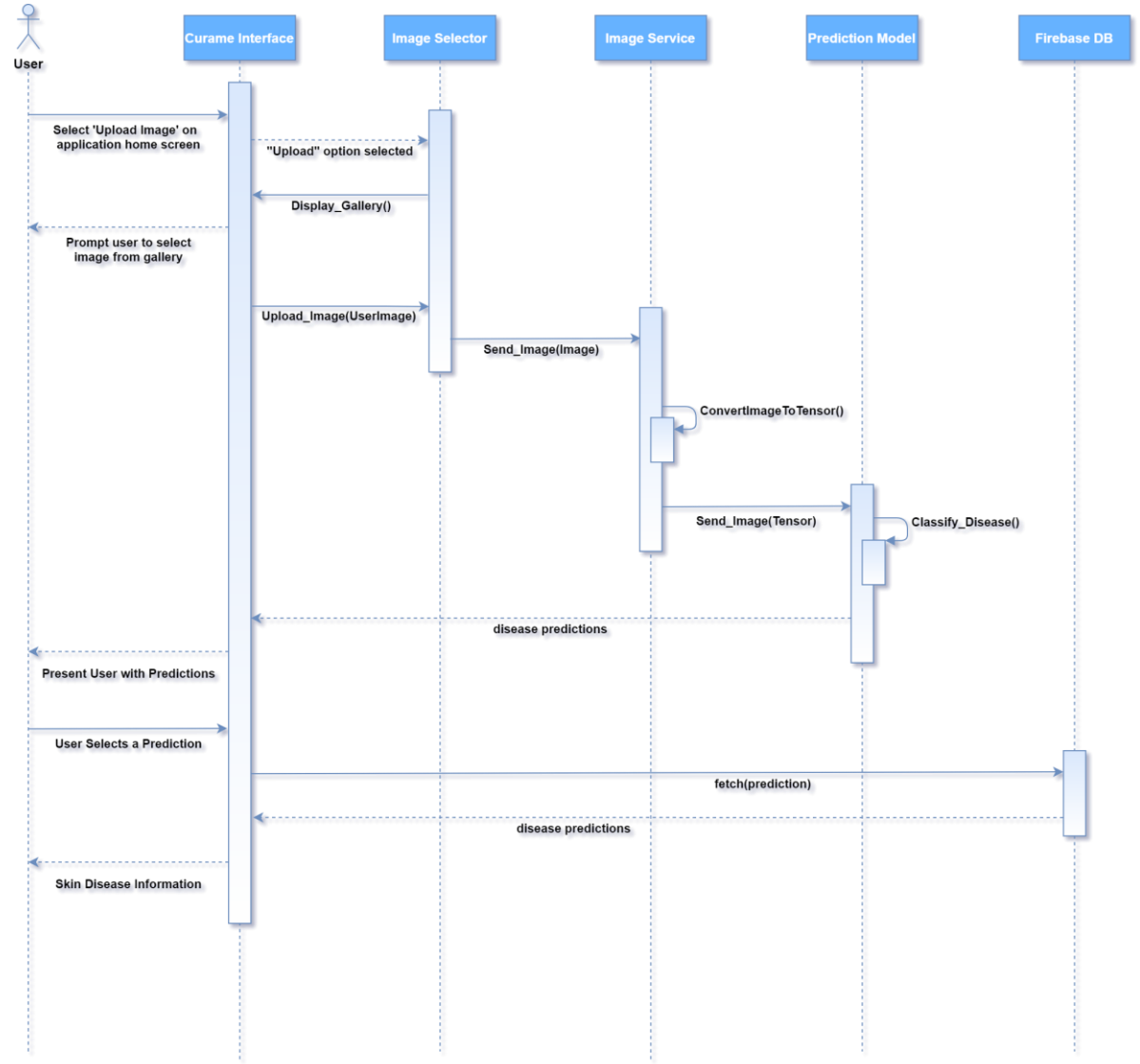


Figure 14.0: Sequence Diagram for Disease Prediction by Scanning

Disease Prediction by Selecting from Gallery

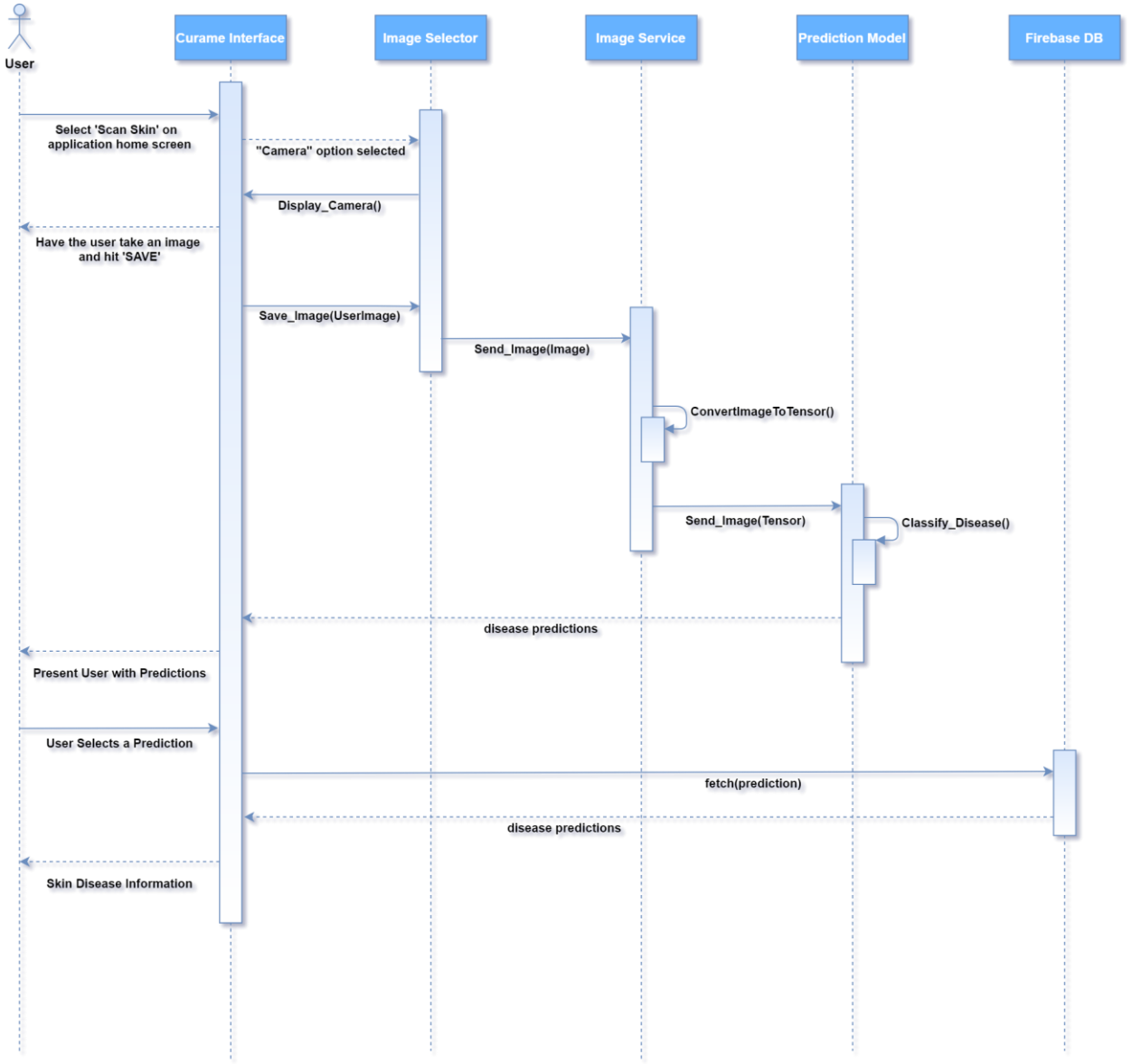


Figure 15.0: Sequence Diagram for Disease Prediction by Selecting from Gallery

View Scan History:

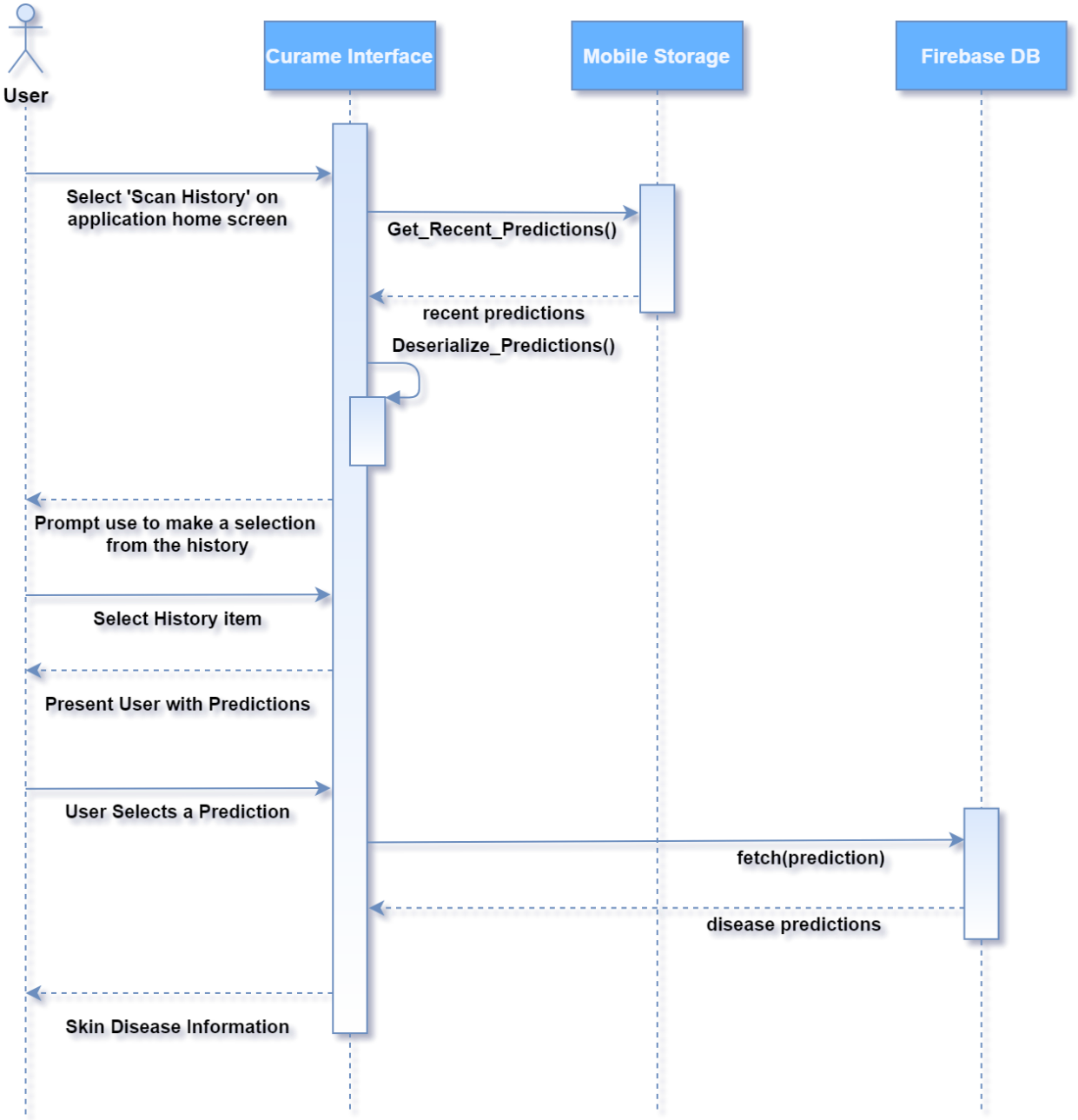


Figure 16.0: Sequence Diagram for Viewing Scan History

ERD (Entity-Relationship-Diagram)

The Curame Development Team also constructed a Entity Relationship Diagram which modeled the system as interacting Data Entities, namely Picture, Tensor, Prediction, Disease and History.

As illustrated by Figure 17.0 Tensors are constructed with Pictures. Each of these Tensors would be used along with Diseases to make predictions. These predictions are then stored within a History Item with its respective Picture (based on the utilized Tensor).

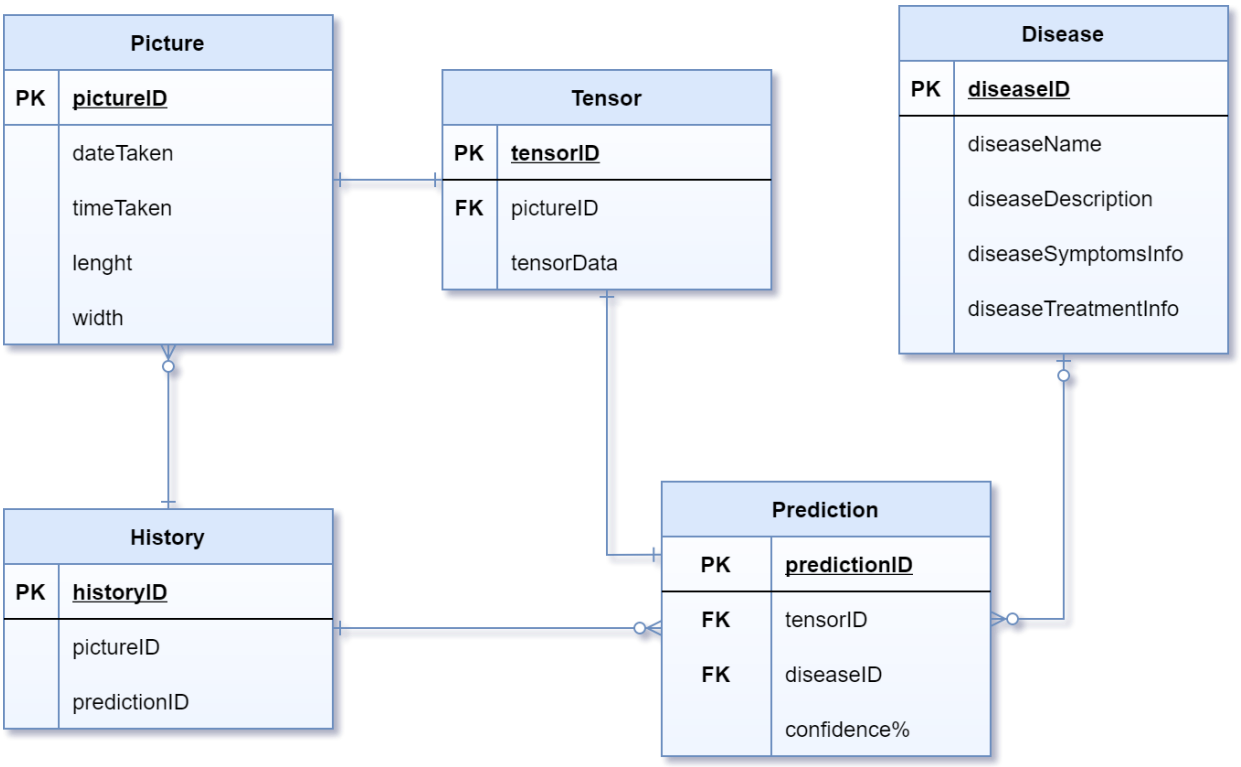


Figure 17.0: Entity-Relationship-Diagram

Technical Constraints

From the numerous amounts of requirements analysis methods conducted by the Curame Team as well as additional research, several technical constraints were realized.

Constraint	Description	Risk Factor / Impact Level
Unattainable High Accuracy	The team does not have enough time and obtainable resources (Training Data) to make the prediction model above 90% accurate. The team gathered 7000 classifications for training/testing. However, with a model made from such a small sample size, the accuracy of disease prediction models may just remain above average (75-80% prediction rate).	**High**
Limited Unique Trainable Disease Sets.	The amount of obtainable, unique disease datasets accessible is sparse and the Team does not have enough time to spare searching for disease datasets. As such, in the 10 days spent locating disease datasets before model training, 7 unique disease datasets were obtained.	**High**
No 'Local' Available Datasets	Datasets found were mostly obtained from locations outside of Trinidad. As such, the app would not be as applicable to Trinidadians.	**Medium**
Potential Difficulty Testing Curame Feature	Due to the team members not being able to contact anyone with these skin diseases, testing the camera feature on a live target would be impossible. Other testing methods would be used (Print Outs and Digital Images).	**Medium**
Android requires a minimum SDK version of 19.	App Development is restricted to libraries with a minimum compatibility of android SDK 19 and up as the main Android library used is TF-Lite, which has a minimum SDK requirement of 19.	**Low**
Camera Quality Restrictions	In Order to test the Mobile Camera usage for disease prediction, a decently quality mobile camera is required. Only 1 team member fulfils these requirements, thus, testing this feature may be slow.	**Low**

Figure 18.0: Technical Constraints

Architectural Design

The Curame Team researched into the various Architectural Designs that Mobile Applications may encompass. The team settled on a modified version of the Model View Controller (MVC) application architecture. Figure 19.0 illustrates the use of a MVC architecture that would be used by Curame System.

Users would only interact with the View of the Curame System. This view would be constructed using XML to render various intractable objects and would display all the interfaces of the System. Here, users may interact with these objects to make a scan, upload an image or view the history.

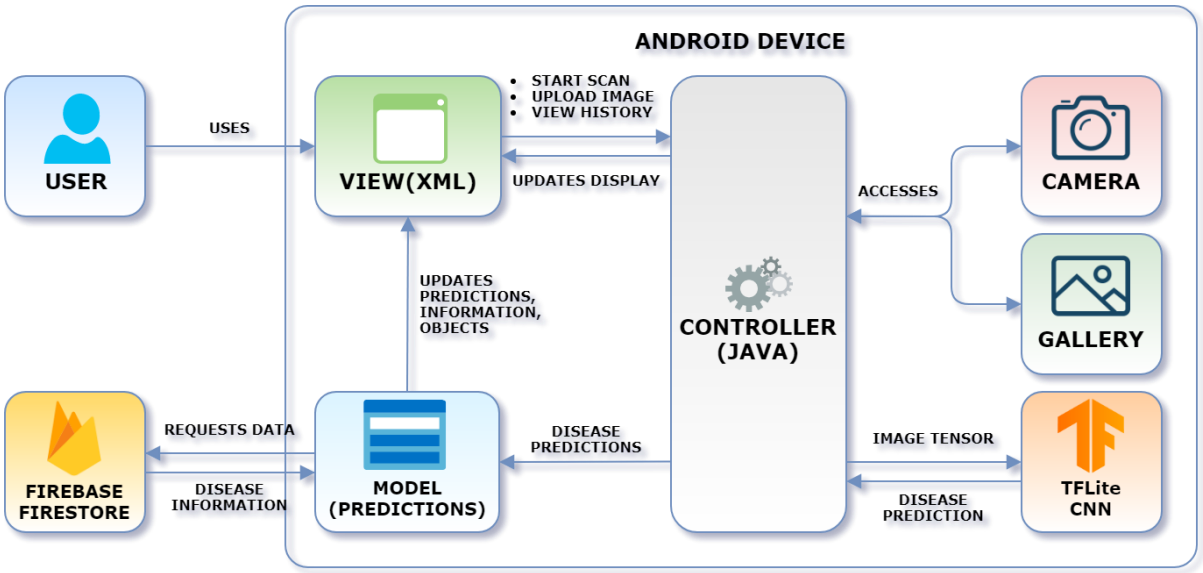


Figure 19: High-Level Architecture Diagram

Description

After this is done, the controller, made using java, would process the actions done. For instance, if a user selects Start Scan or Upload Image, the controller would access the Camera or Gallery and render it to the user. When users select an image to process, the controller would convert it to a tensor and utilize a TensorFlow Lite Convolutional Neural Network (CNN) Model to make a disease prediction. After predictions are made, the controller would transfer this to the Model.

The Model would store or retrieve these predictions from local storage and would be responsible for requesting skin disease information from an external Firebase Database. As illustrated in the figure, the model would then update the display with predictions, disease information and even make updates to the visual objects displayed to the user by the view.

Reason for Selection

A MVC was chosen as opposed to other architectural designs for various reasons:

- It would be the ideal architecture for developing very complex but very small and lightweight mobile applications such as the Curame App. The MVC divides this complex system into easy-to-understand components that would greatly aid the team in the design development process.
- The team can easily integrate other frameworks and architectures such as Camera and Gallery Access, the TF-Lite Library and Firebase Libraries.
- MVC architectures would enhance the test-driven development of the Curame system and improve the overall testability. Since most of the components of the Curame system are interface based, this architecture makes it easy to test the functionality of these user interface tests or tests using mock objects.

Possible Alternative Design

An alternative architecture for this system would have been to model the system as a Progressive Web Application (PWA) rather than an android application and utilize a progressive application client server architecture illustrated in Figure 20.0.

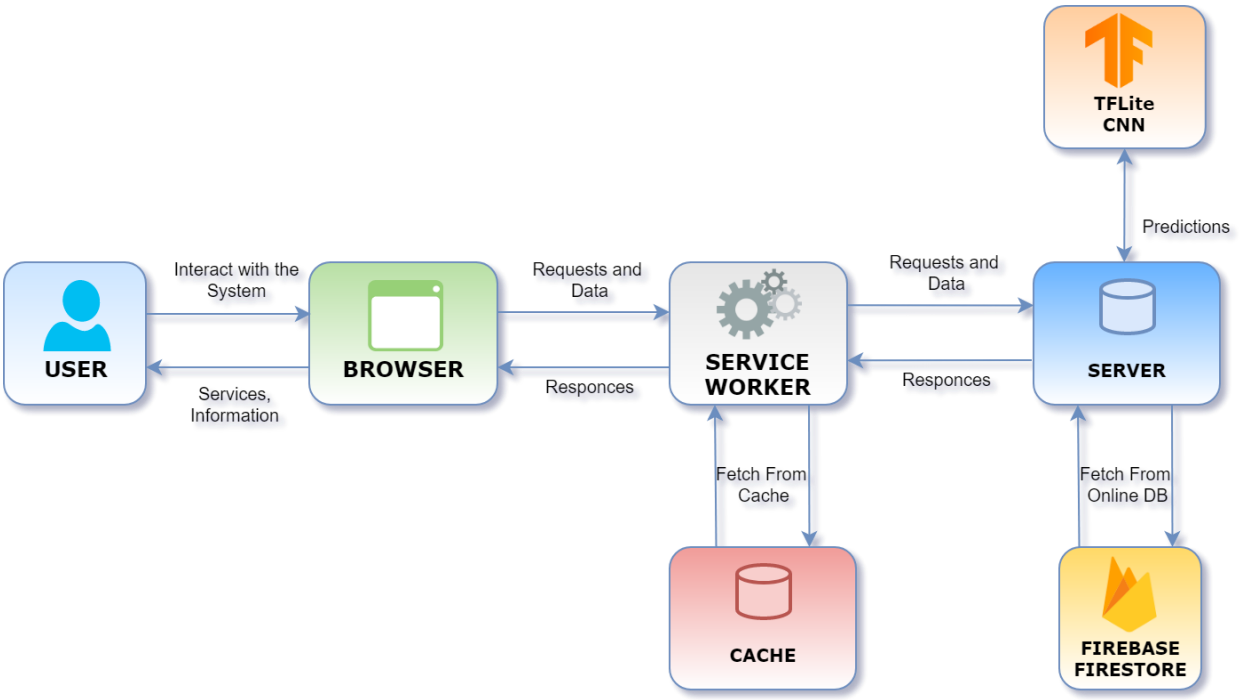


Figure 20.0: Alternative PWA architecture

Description

With a PWA Architectural Approach, Users would instead use their android devices (or other devices for that matter) to interact with the System from their built-in browser. The system would have been built using React Native or other similar technologies. The browser would request permissions to use the phone's Camera and Gallery. A service worker would be responsible for providing the browser with information when requests are made. The service worker would use the webserver to make requests to and receiver information such as predictions and skin disease information. Users that make these requests would receive this information while online but additionally the service worker would store certain pieces of data, such as predictions and history items, in the phone/browsers cache for offline retrieval. The server would work like the controller in the MVC architecture, except it would be hosted online, using the TF-Lite model and Firestore components.

This would have provided several benefits to the Curame app. It would have been progressive as they work for any user and responsive as they fit to any screen on any device. IT would be easily updatable and installable as it is not completely online.

Reason for not choosing this

The offline support, however, a nice feature, is very limited which would limit the possibilities that the Curame team would have with skin disease prediction. The PWA would be slower in fetching and retrieval than the Android Architecture and there would be very limited access to hardware components on User Devices. Lastly, it may not be supported on older Android Devices.

Class Diagram

After selecting the architectural model of the Curame system, the team constructed a Class Diagram to outline how the object instances interact with each other. Figure 21.0 displays how each class would have state information describing the class and behavior, outline what the class can do and how it can interact with other classes. One can see the sequence of interactions from the user interacting with the Curame Interface which makes requests to the Curame System. The Curame System is responsible for carrying out the requests made by the user.

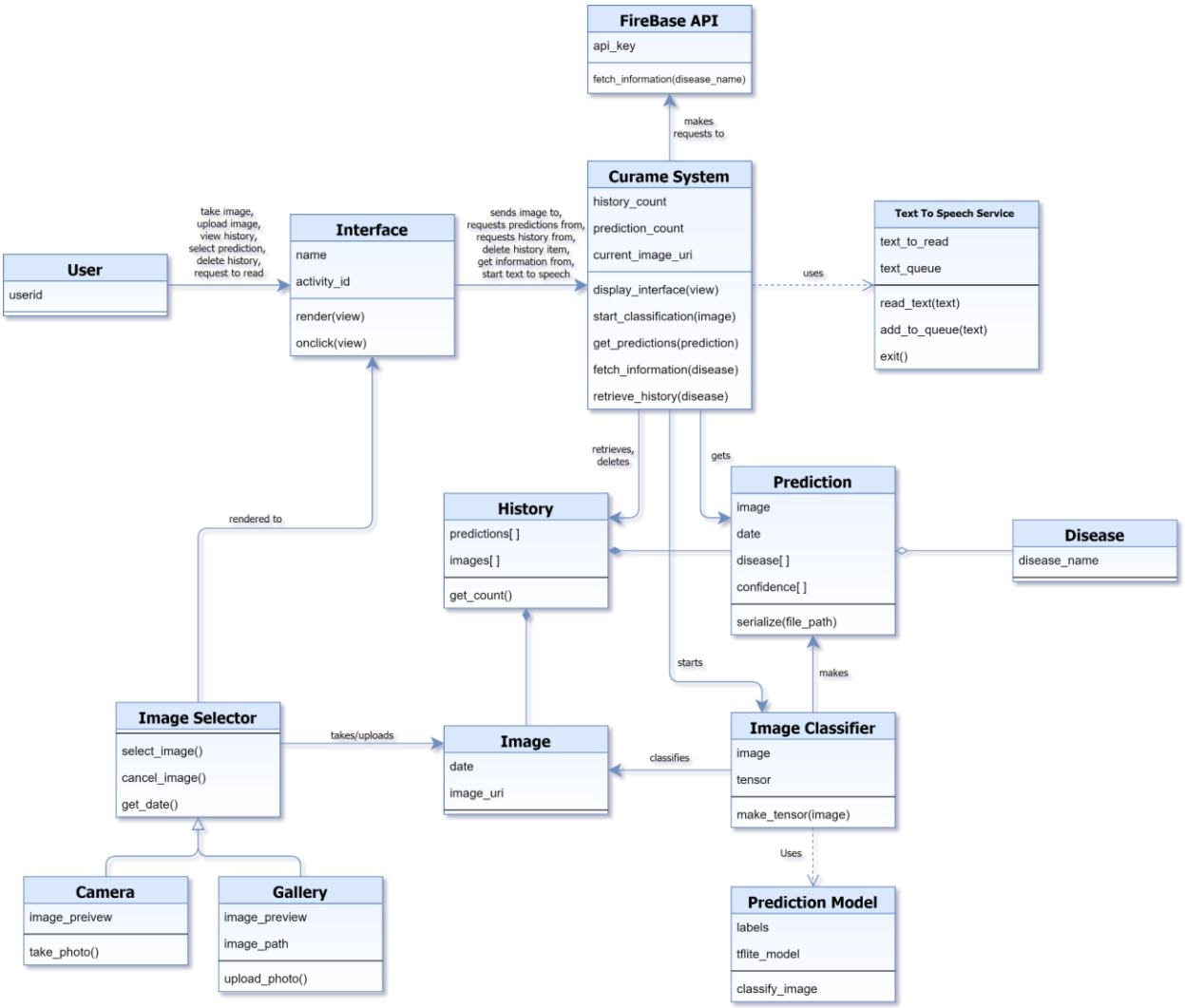


Figure 21.0: Class Diagram

State Transition Diagram

State transition diagrams were also constructed and used to show the flow of the system. Each state, in the case of android programming, is a different android activity, each activity having its own interface and its own transitions to other states. Figure 22.0. outline which Activity will be displayed on open and how to access every other activity from the start.

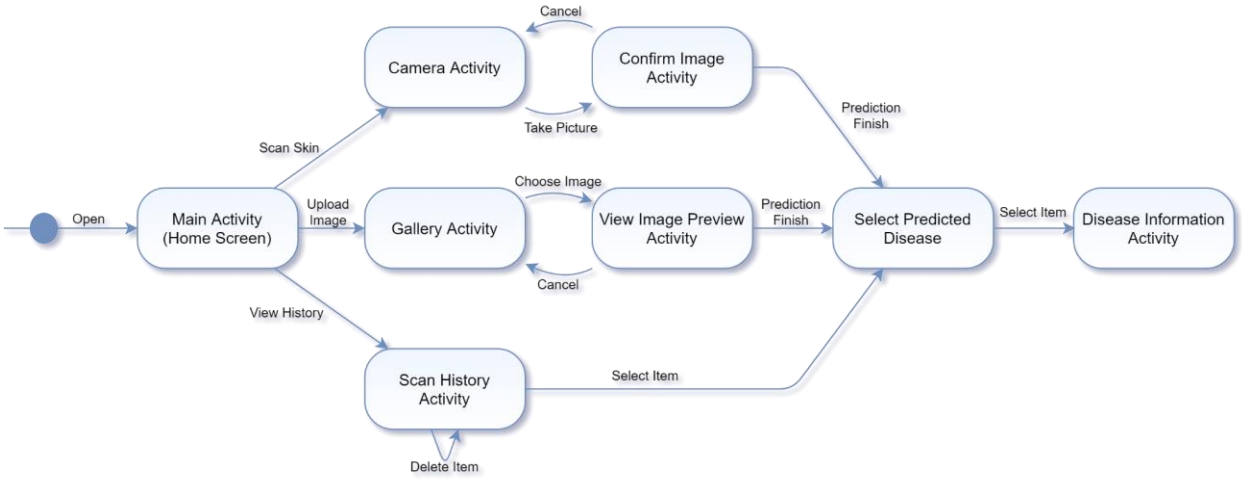


Figure 22.0: State Transition Diagram

System Components

The Design of the Curame System can be broken down into 5 major components. These components are:

- Image Input Component,
- Prediction History Component,
- Skin Disease Prediction Component,
- Disease Information Component and,
- Firebase Component.

The interaction of these components is visualized in Figure 23.0 and explained in the following section.

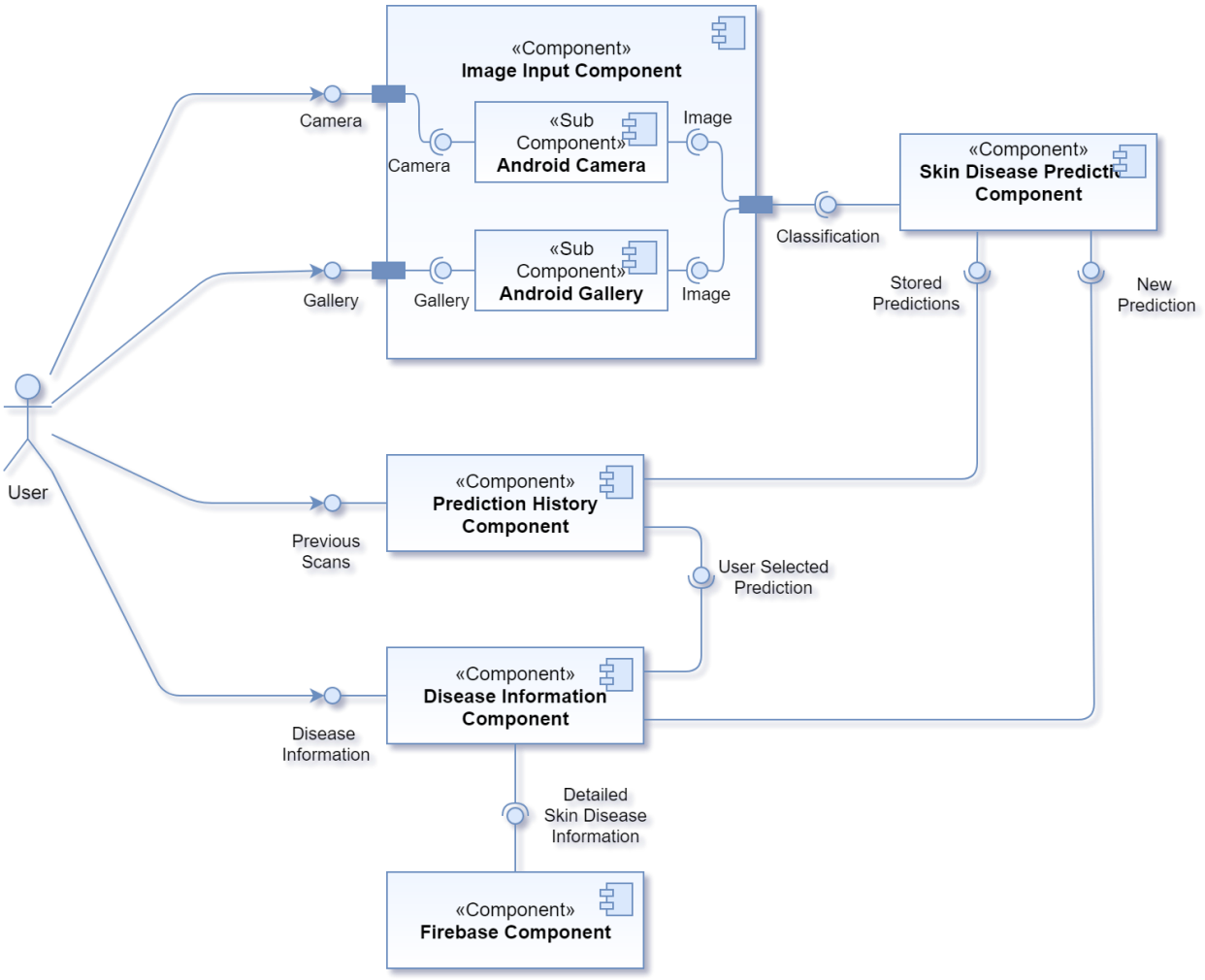


Figure 23.0: High Level Component Diagram

Component 1 - Image Input Component

This component would be responsible for allowing the User of the Curame System to input images of their skin and would be accessed from the Home Screen Interface. This feature would also allow the user to preview the image before passing it to the Skin Disease Prediction Component to be classified. This component is split into two Sub-Components:

- Android Camera Sub-Component and,
- Android Gallery Sub-Component.

Both these components utilize built in Android Components and as such, would request appropriate permissions from users before being accessed.

Component 1.1 - Android Camera

This subcomponent would access the Android Camera Component and would allow the User to take images using their mobile Camera.

Component 1.2 - Android Gallery

This subcomponent would access the Android Gallery Component and would allow the User to select an image directly from the Gallery. This Component would be configured to only allow from the selection of a single image file.

Component 2 - Prediction History Component

This component would display, to the User, a list of recent scans made by the user and some details on the scan. This component would be accessed from the Home Interface. The Prediction History Component would display the recent scans in a Scan History Interface, rendering each scan as its own selectable item, displaying for each item the following:

- The image of the scan,
- The date of the scan,
- The predicted skin diseases for that image,
- The respective accuracy of the prediction and,
- A ‘trash’ icon.

The user may then be able to select any of these history items to be taken to the Disease Information Component for that scan. The user would also be able to delete the scan by selecting the trash icon atop each item. These items are taken from the phone’s mobile storage to the Curame App, where they were stored by the Skin Disease Prediction Component, and as such, would require storage access permissions.

Component 3 - Skin Disease Prediction Component

The skin disease component is accessed by the Image Input Component after an image is confirmed by either the camera or the gallery. After an image is selected, it is passed to the Skin Disease Prediction Component, where the image would be saved to the Curame Picture directory on the phone's local storage in its own folder. The image is then passed to the Classification Sub Module.

Component 3.1 - Classifier Module

This Subcomponent is responsible for Classifying a given image. This component would take an image, convert it to a tensor and then utilize a flite model file to produce a prediction of what diseases it thinks the image may be. This information is then filtered to select the relevant values only (i.e., the disease(s) that it may be classified as) then saved to the same file location as the image. The location of the folder containing the image and the prediction is then passed to the Disease Information Component.

Component 4 - Disease Information Component

This component is accessed after the user selects an image from the history screen or after an image is selected by the Image Input Component then classified by the Skin Disease Prediction Component. This component displays to the user two interfaces. The first interface is a List Skin Diseases Interface which lists each skin disease in the image prediction file as their own skin disease item. These disease items are displayed in a similar fashion as the History Interface, displaying the name of the skin disease and the Accuracy of the prediction. When the user selects any of these skin disease items, the Disease Information Component renders another interface, Information Interface, the information interface would display the image for that prediction, the name of the skin disease selected, and the accuracy of the prediction. The Skin Information Component also accesses the Firebase Component, where it passes the name of the disease to and receives information on the skin disease. This information is also displayed to the user on the Information Interface.

Component 5 - Firebase Component

This component accepts the name of a skin disease then performs a fetch to the Curame Firebase Firestore Database which would return information on a skin disease. The fetch either results in an array list containing the name, description, symptoms and treatment of a skin disease or an empty array (if no disease was found or the server did not respond). This information is then returned to the calling component.

Home Page

This is the main navigation page for the Curame App. Here the user will be able to access all the other interfaces and components of the Curame App. The main buttons on this page are “Scan Skin”, “Upload an Image” and “Scan History”.

- When the user selects “Scan Skin” the Android Camera Component would be accessed, and the Camera Interface would be displayed.
- When the user selects “Upload an Image”, the Android Gallery Component would be accessed, and the Gallery Interface would be displayed.
- When the user selects “Scan History”, the History Component is accessed, and the History Interface is displayed.

From any of these components, the user can return to the Home page by selecting the ‘back’ button on any Android Device.

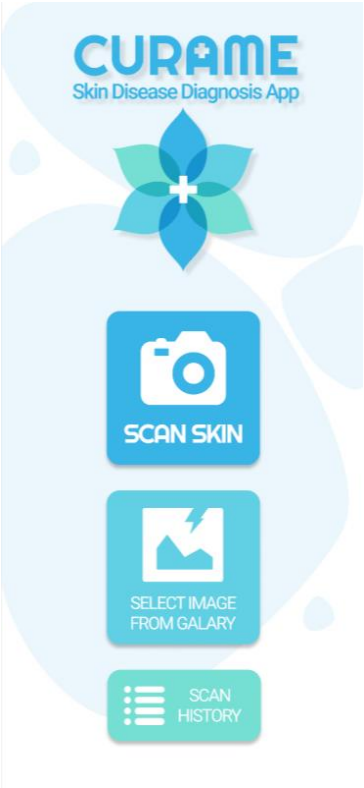


Figure 24.0: System Home Interface Design

Camera Interface

This is a built-in Android interface accessed when the user selects the “Scan Skin” option from the Home Interface. From this interface, the user would be able to take an image of their skin and then they would be prompted to confirm or decline the image.

- When the user declines the image, they would be able to retake the image.
- When the user confirms the image, the Skin Disease Prediction Component is accessed and they would be taken to the Select Illness Interface.

Figure 25.0: Camera Interface Design

Figure 26.0: Camera Confirm Interface Design

Gallery Interface

This is another built-in Android interface accessed when the user selects “Upload an Image” option from the Home Interface. From this interface, the user may select a single image and then would be prompted with another Interface to preview the selected image as well as confirm or decline the image.

- When the user declines the image, they would be able to reselect the image from the gallery.
- When the user confirms the image, the Skin Disease Prediction Component is accessed, and they would be taken to the Select Illness Interface.

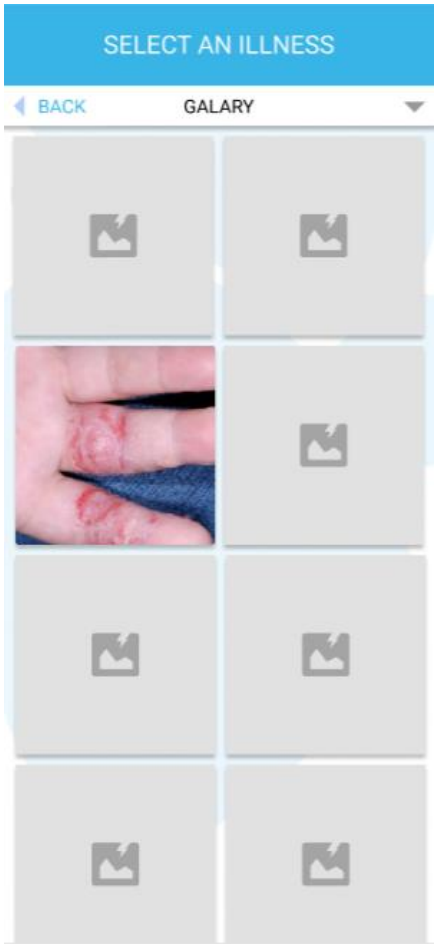


Figure 27.0: Gallery Interface Design

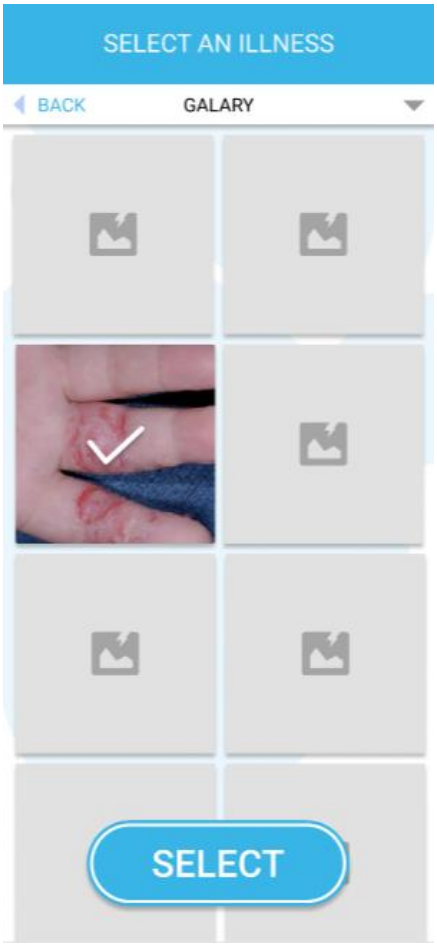


Figure 28.0: Gallery Confirm Interface Design

History Interface

This interface can be accessed by the user from the Home Interface by selecting the “Scan History” option. This interface would present the user with a list of recent scans and for each scan, the date the scan was taken, the predicted disease percentage accuracy for that skin disease. Users may then select any of the history items to start the Disease Information Component which would display an intractable list of predicted diseases. Users may also select the trash button above each history item to delete that scan. If there are no scans, the appropriate message would be displayed.

Figure 29.0: History Interface Design

List Predicted Skin Disease Interface

This interface is accessed after the user makes, selects/takes an image in the Image Input Component or would be accessible front the History interface by selecting a scan. This interface would display to the user the list of predicted illnesses for a selected scan. Each predicted disease item contains the name of the disease and the percentage accuracy of the prediction. Users would be able to then select any of the predicted skin disease items to start viewing more information about the selected disease.

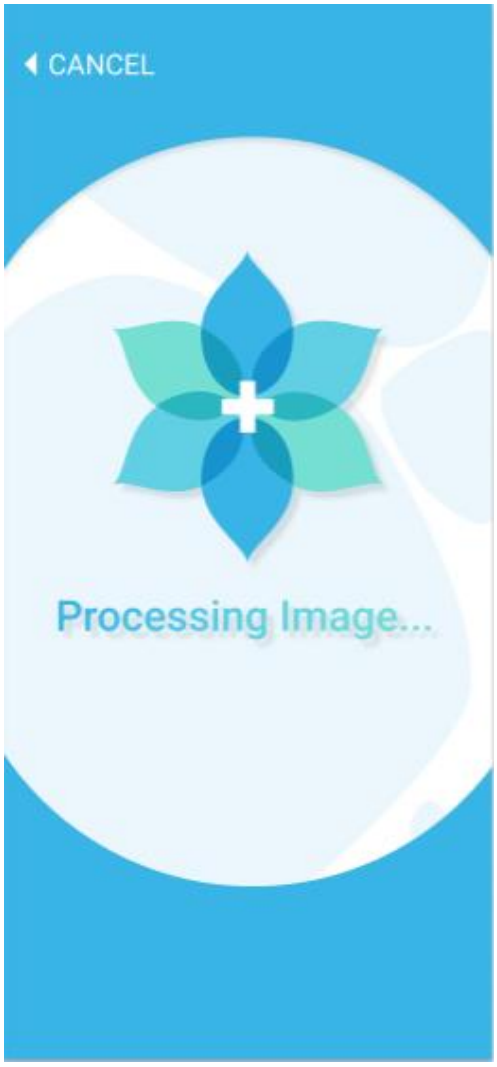


Figure 30.0: Loading Screen Design

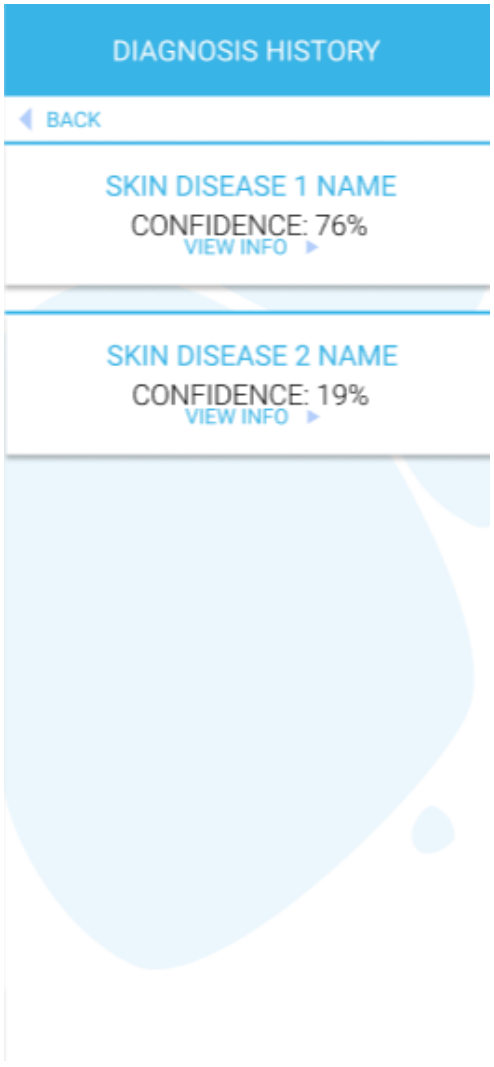


Figure 31.0: Select Diagnosis Interface Design

Disease Information Interface

This interface is accessed after the user selects a skin disease from the List of Predicted Skin Disease Interface. This interface displays information about a selected skin disease. This information contains:

- Image of the scanned skin area,
- Name of the predicted disease,
- The accuracy of the prediction,
- A description of the skin disease,
- A description/list of symptoms associated with the disease and,
- A description/list of treatments for the disease.

Users additionally, would be able to select the image at the top to gain an enlarged version of the image. Users would also be able to select any of the information to have it read aloud to them using text-to-speech.

Figure 32.0: Disease Information Interface Design

Implementation

Proceeding the comprehensive requirements analysis and system design, the Curame Team began the creation process of Curame, the Skin Disease Prediction App.

Approach and Methodology

Firstly, the methodology the Curame Team elicited was that of the Agile Methodology.

An agile methodology to software development is based on planning, developing and testing software in iterations. With an agile methodology, the team has access to adaptive planning which involves planning taking place multiple times for each iteration and subsequent planning learning from previous iterations. This was beneficial as the nature of the Curame App was highly uncertain and complex. Agile methodology also gave way to evolutionary development where the team could have a product of the system available at the end of each iteration then alter product requirements and plan changes to the products current state. This approach was also very flexible with rapid iterations.

The team utilized an agile approach based on figure 33.0 below. Firstly, the team got together and outlined all the functional and non-function, user, and system requirements for the Curame App. These requirements were general and left room for adaptation and evolution at each iteration. The team then did research, planning and design for the selected requirements at the start of each iteration, then divided work accordingly. After development, the product was tested, both individually and then integrated with the rest of the system (where possible). This was repeated until the final product was developed and ready to be tested all together then deployed.

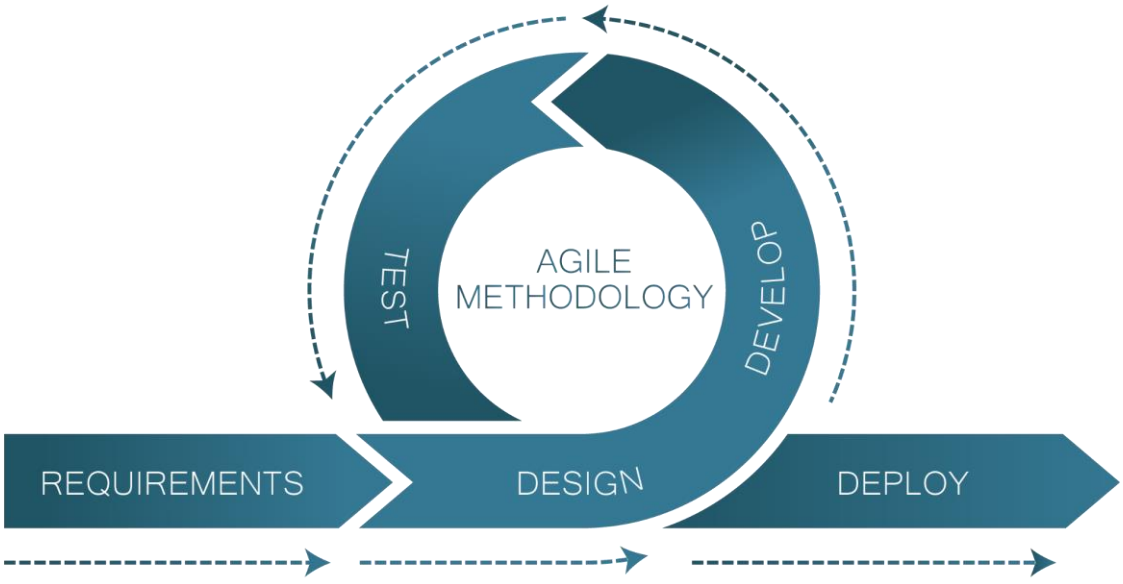


Figure 33.0: Agile Approach Diagram

Technologies

During development at the start of each iteration, the team performed research into potential technologies that could be used to ease application development, improve the quality or make the app more modern, user friendly and progressive.

Google Colab

Google Colab is a free python coding environment that the team used to develop and train a convolutional neural network (CNN) for predicting a skin disease based on a given image. This allowed the team to write and execute portions of python code through the browser and was an ideal platform for machine learning development. With Google Colab, there was access to 13GB of RAM and 2.2GHz processing power. This allowed for training the CNN in the background whilst performing other tasks. The language used in Google Colab used is Python and the main libraries used were TensorFlow, Keras and Pyplot. Python is very diverse and is capable of utilizing many libraries, ideal for machine learning.

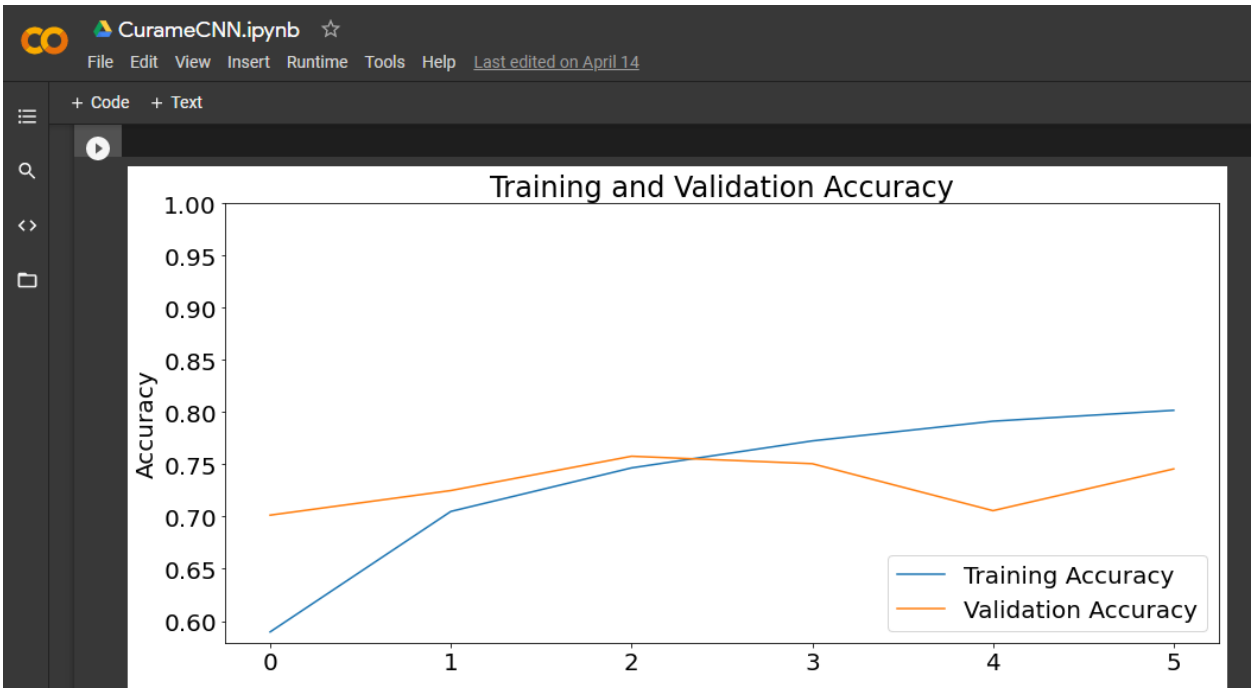


Figure 34.0: Google Collab Usage

Tensor Flow

TensorFlow is a machine learning library available to several different programming languages. TensorFlow allowed Curame to access functionality to build and train a convolutional neural network to accurately classify skin diseases. TensorFlow enables GPU usage for training machine learning models which speeds up the process of training. Curame used this functionality to train through Google Colab. TensorFlow is a low-level library that allows Curame to use functionality of other libraries running on it.

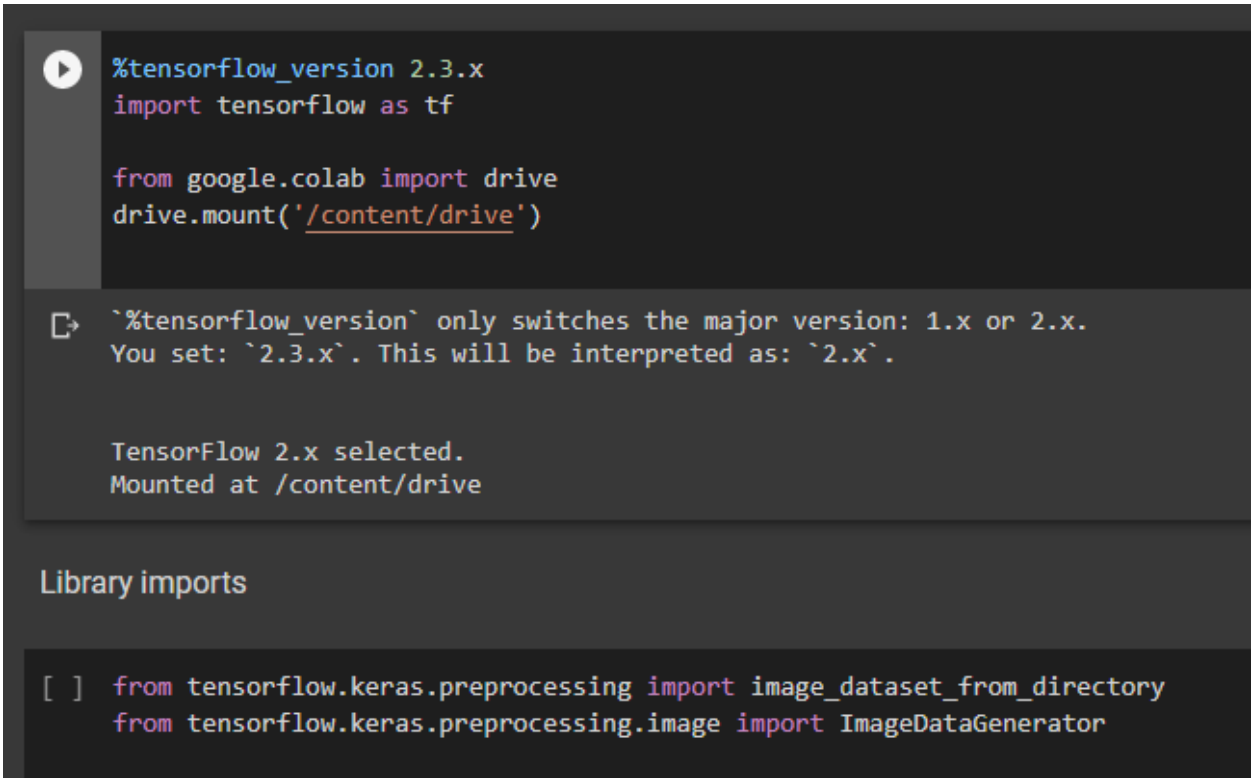


Figure 35.0: TensorFlow Usage

Keras

Keras is a library like TensorFlow that allows the Curame team to access machine learning functionalities. Keras is a high-level library running on TensorFlow. The team used Keras to access a pre-trained model called MobileNet and used functionality from Keras to add extra layers to the model and used a technique called transfer learning to retrain the top layers of the model without affecting the weights of the lower layers. Keras also allowed the team to pre-process images to be used by the model.

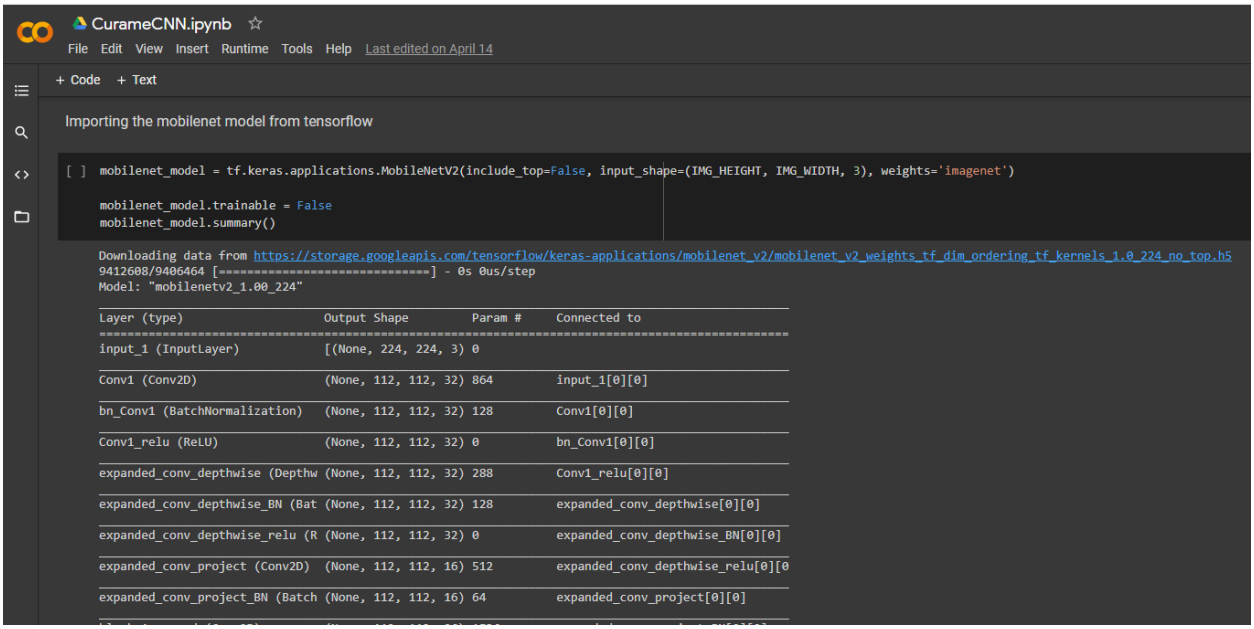


Figure 36.0: Keras Usage

Pyplot

Pyplot is a collection of functions found in the Matplotlib library. It allowed the Curame team to plot line graphs to demonstrate the increase in accuracy of the convolutional neural network during training.

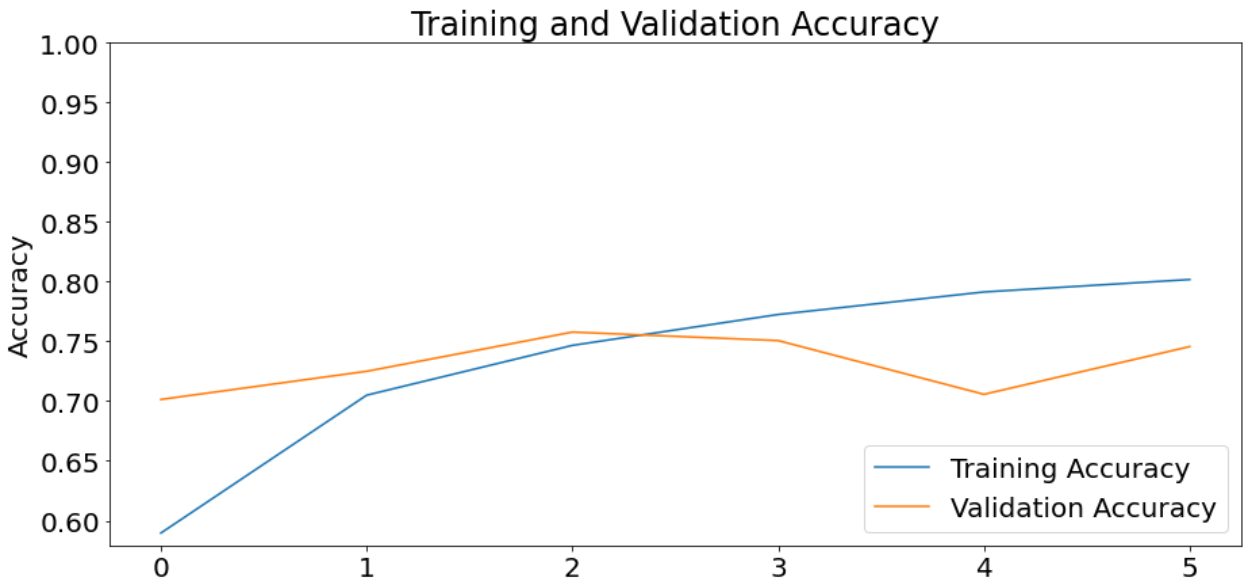


Figure 37.0: Pyplot Usage

Android Studio

The environment the team used to program the Curame Application was Android studio. Android studio is an android programming environment by Google. The team chose Android studio for its development environment as it was equipped with a number of features that aided in application design.

Android studio had a built-in layout editor that allowed not only the coding of activity layouts but an additional visualization and editable activity layout interface. Android studio also can use Materialize, particularly Material Cards that was used by the team to create a modern and professional interface.

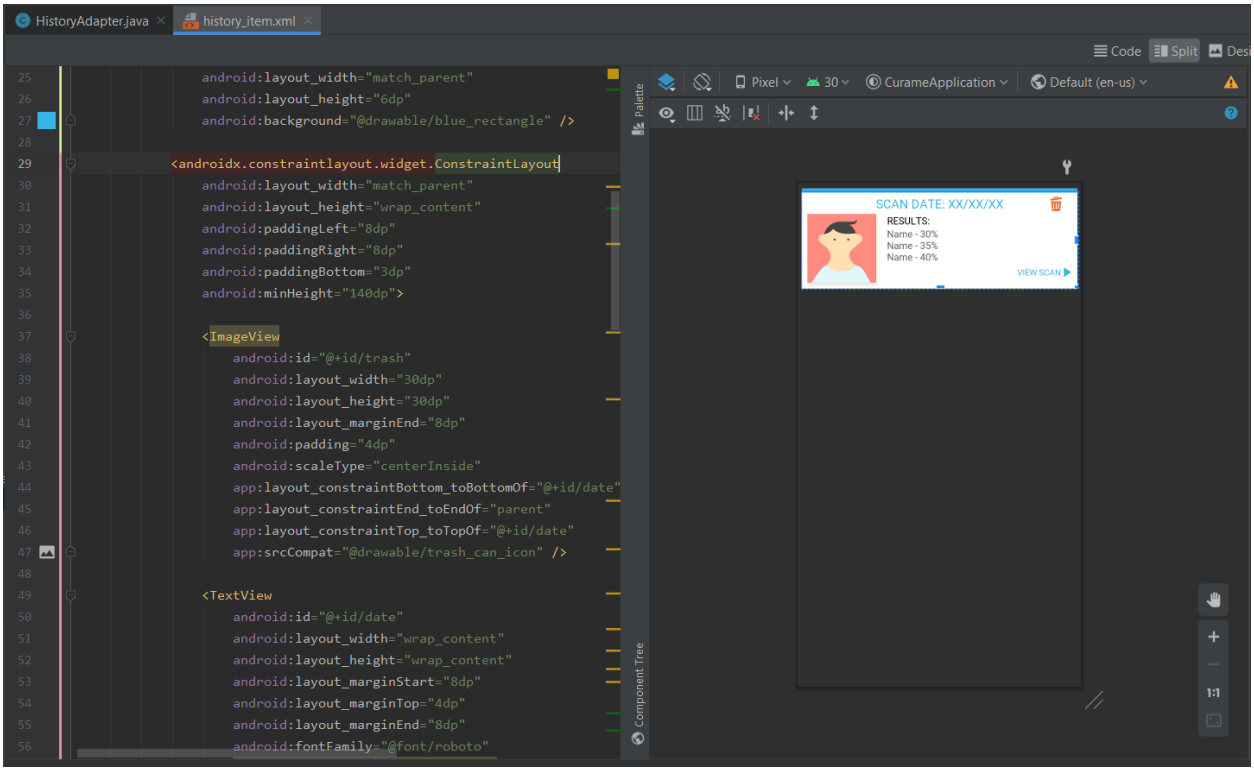


Figure 38.0: Android Studio Layout View Usage

Android studio also facilitates the use of build in android intents. For the Curame project, the team utilized the built in Android Camera Activity and Gallery Access Activity. These were used to allow the user to either take an image of a skin disease or use the gallery interface to select an image from the gallery.

```
//THis function starts the Camera Activity
public void goToCamera(View view) {
    //If permission is not granted for Camera and/or external storage, request user to grant permission to application
    if(ContextCompat.checkSelfPermission(
            getApplicationContext(),
            Manifest.permission.CAMERA
    ) != PackageManager.PERMISSION_GRANTED ||
    ContextCompat.checkSelfPermission(
            getApplicationContext(),
            Manifest.permission.WRITE_EXTERNAL_STORAGE
    ) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(
                activity: MainActivity.this,
                new String[]{
                        Manifest.permission.CAMERA,
                        Manifest.permission.WRITE_EXTERNAL_STORAGE
                },
                REQUEST_CODE_PERMISSIONS_CAMERA
        );
    }else {
        //if permissions are already granted, call the camera
        callCaptureImageIntent();
    }
}//goToCamera
```

Figure 39.0: Android Studio Camera Activity Call Usage

Android studio also comes with the ability to utilize android emulators. The team utilized this to run the Curame Application without the need to have an appropriate mobile device.

Figure 40.0: Android Emulator Usage

The primary programming languages used for this project were:

- Java and,
- XML.

Java

Android studio can use Java, C++ and Kotlin. Java was chosen as the language to develop our application controller as is the language the team was mostly familiar with. Java allowed for the creation of activities that would render views, created with XML, as well as the creation of classes such as Disease, Prediction, and the Machine Learning model. Java also facilitated the use of TF-Lite, which was the library used for image prediction. Java also facilitated the use of JUnit 5 which was used to unit test the Object Classes for the Curame application.

XML

XML is the language used by android studio to create the view's displayed by the java activities. The team used XML to outline the layout of activities such as the buttons, views and cards that created the Curame app's user interface. XML also gives access to Recycler views, which can be initialized with java. This allows lists of similar data to be displayed in their own individual views with their own data stored in it. The user may then scroll through the items displayed.

The team also utilized Android Studio's capability of accessing a number of useful libraries. The main libraries used were:

- TensorFlow Lite,
- Firebase Firestore,
- Picasso and,
- JUnit.

TensorFlow Lite

The Curame team used the TensorFlow lite nightly dependency to run inferences on the TF-Lite model. The library simplified converting images to bitmaps, loading images into byte buffers, and automatically determining the float size that the model inputs. It also assisted Curame with interpreting the floating-point outputs made by the model as well as the mapping of the floating point outputs to a skin disease.

Firebase Firestore

The Firestore dependency allowed the Curame team to use CRUD (Create, Read, Update, Delete) operations on the Firestore database. The Firestore functionality was used to fetch disease information when a significant classification on an image was made (The top x classifications) by querying the disease name against the document name in the Firestore database.

```
    //Cloud Firestore instance
    public FirebaseFirestore db = FirebaseFirestore.getInstance();
    private Disease diseaseInfo;
```

```
//This function fetches the disease from Firebase, if the disease is found, it displays it, if
private void fetchAndDisplayDisease(String diseaseName) {
    //This function fetches the named disease from the firebase db base
    DocumentReference docRef = db.collection( collectionPath: "Diseases").document(diseaseName);

    //Create a listener to check when the document is completed loading
    docRef.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {

        //Check if the document did not complete loading
        @Override
        public void onComplete(@NonNull Task<DocumentSnapshot> task) {

            //If the task is successful
            if (task.isSuccessful()) {
                //get the result of the fetched document
                DocumentSnapshot document = task.getResult();

                //if there is data
                if (document.exists()) {
```

Figure 41.0: Firebase Firestore Library Usage

Picasso

Picasso is a very useful library in Android studio for displaying Images. With Picasso, images are displayed in the most optimal way possible to greatly improve the performance of the Curame App, especially on devices with low memory. Picasso takes a selected image then fits the image to a selected frame. By fitting the image, the minimum number of pixels needed to display the image on the device, for a frame of a given size, is used. This means that large, high pixel per inch (PPI) images are scaled down and rendered at the optimum PPI, reducing the memory requirements for displaying many images on one Activity.

```
        //Get Image taken from camera
        imageUri = (Uri) getIntent().getParcelableExtra( name: "IMAGE_DATA");

        //Set Image using picasso
        Picasso
                .get() Picasso
                .load(imageUri) RequestCreator
                .placeholder(R.drawable.empty)
                .fit()
                .centerInside()
                .into(imagePreview);
    }//onCreate
```

Figure 42.0: Picasso Usage

JUnit 5

JUnit 5 is a library used by Android Studio with Java to perform Unit testing on Object Class. The team used JUnit 5 to perform unit testing on the Disease Class, the Prediction Class, and the History Class. This was used to unit test the Object classes for correct functionality and ensure that nothing breaks after changes are made.

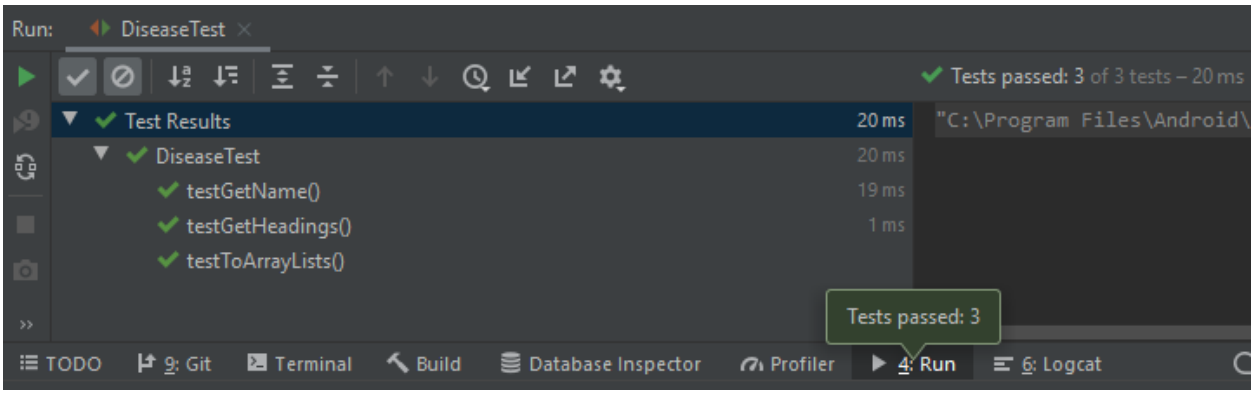


Figure 43.0: JUnit5 Usage

Firebase

Firebase is a platform provided by google that gives developers easy access to tools they may need for their web or mobile applications. For the Curame team, Firebase allowed us to build a fully functioning database to store disease information through Firestore. Firestore is a noSQL database that is available through the firebase tool. Firestore allowed the Curame team to easily build an API to query for information about diseases. Firestore simplified fetching data from a database because it automatically updates information once it is changed in the firestore database without any added calls.

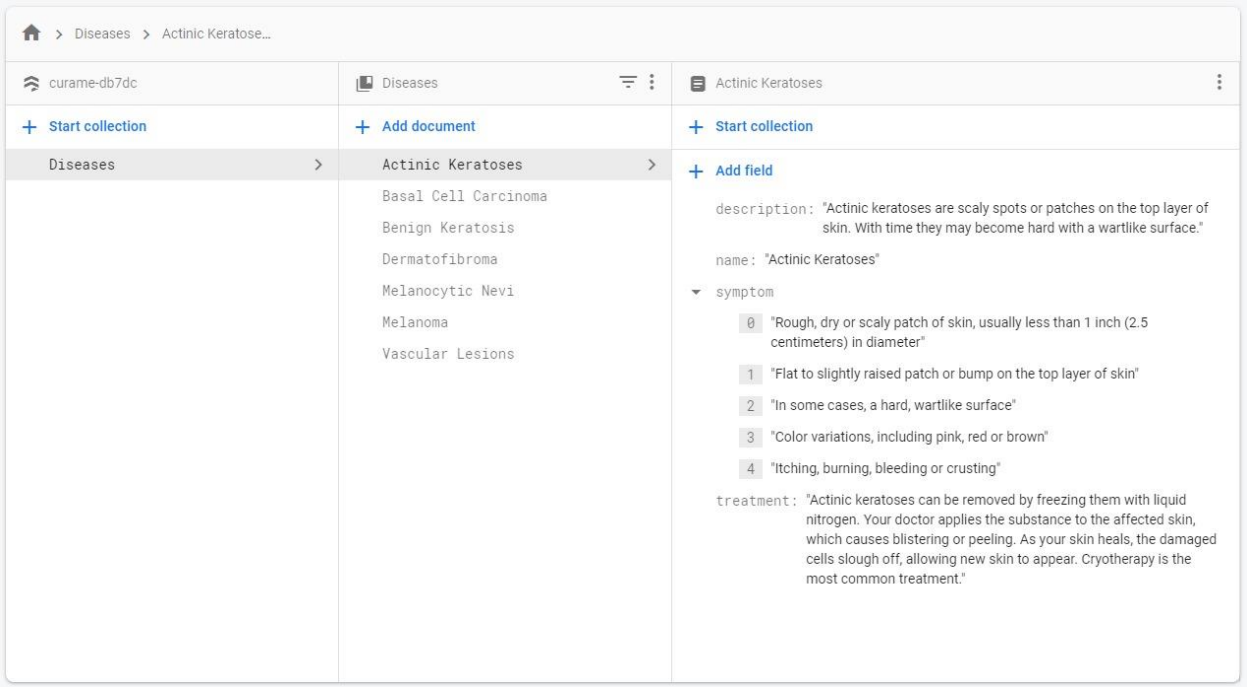


Figure 44.0: Firebase Usage

Trello

Trello is a free online web application used by the Curame team to organize works and activities throughout project development. Trello uses movable cards to display activities that are backlogged, in progress and completed and issues that have been or need to be resolved.

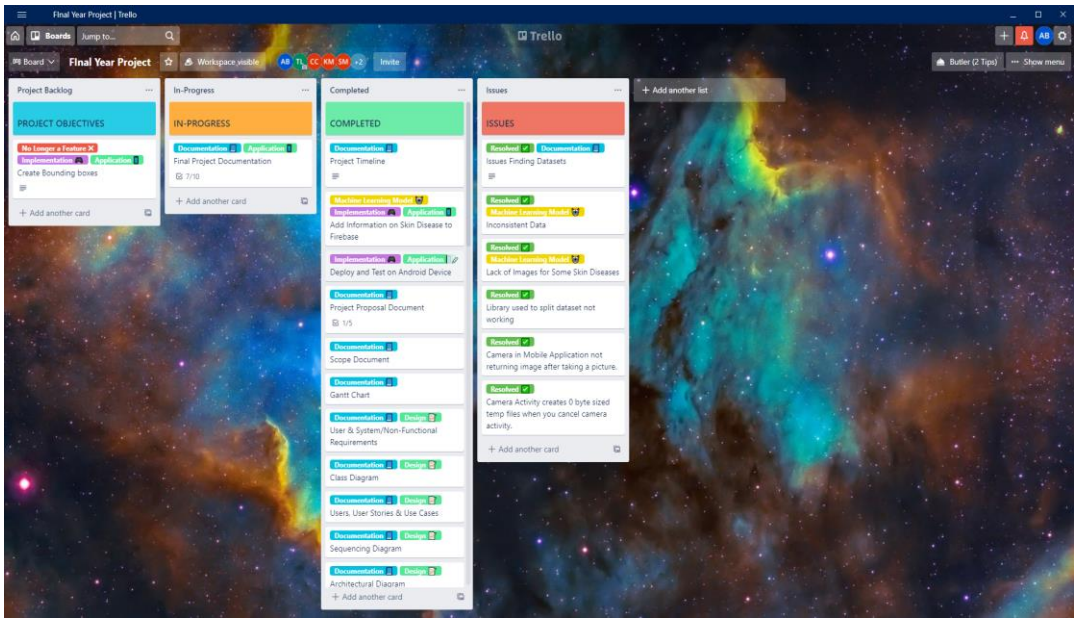


Figure 45.0: Trello Usage

Figma

Figma is a prototyping web application used by the Curame team to develop an early prototype of the Curame application. With Figma, a usable prototype could be created to simulate use to the application. This allowed the team to get an idea of the user experience of the app and aid in overall design. Figma was used to gain an idea of how the application would work before any development takes place.

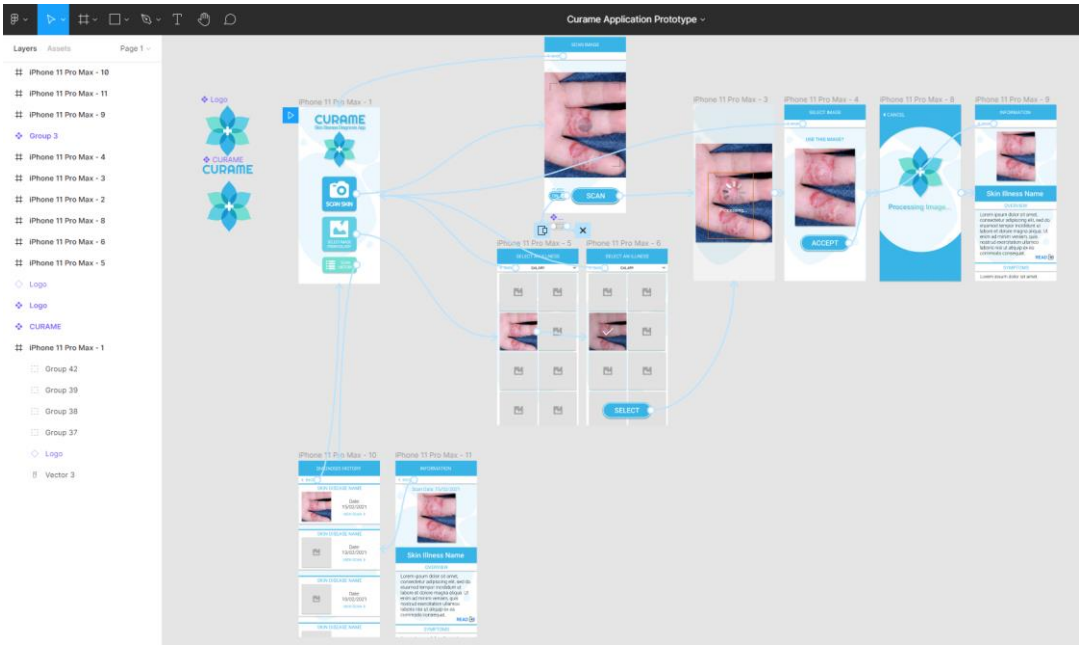


Figure 46.0: Figma Usage

Adobe Illustrator

Adobe illustrator is a vector image creation suite used by the Curame team to develop icons and graphics for the Curame Application. This allowed the team to create the ideal User Interface for the application. With Illustrator, vectors of the icons and graphics are created then exported as high quality ‘png’ images where they can then be used in the Application Layout or Figma Prototype.

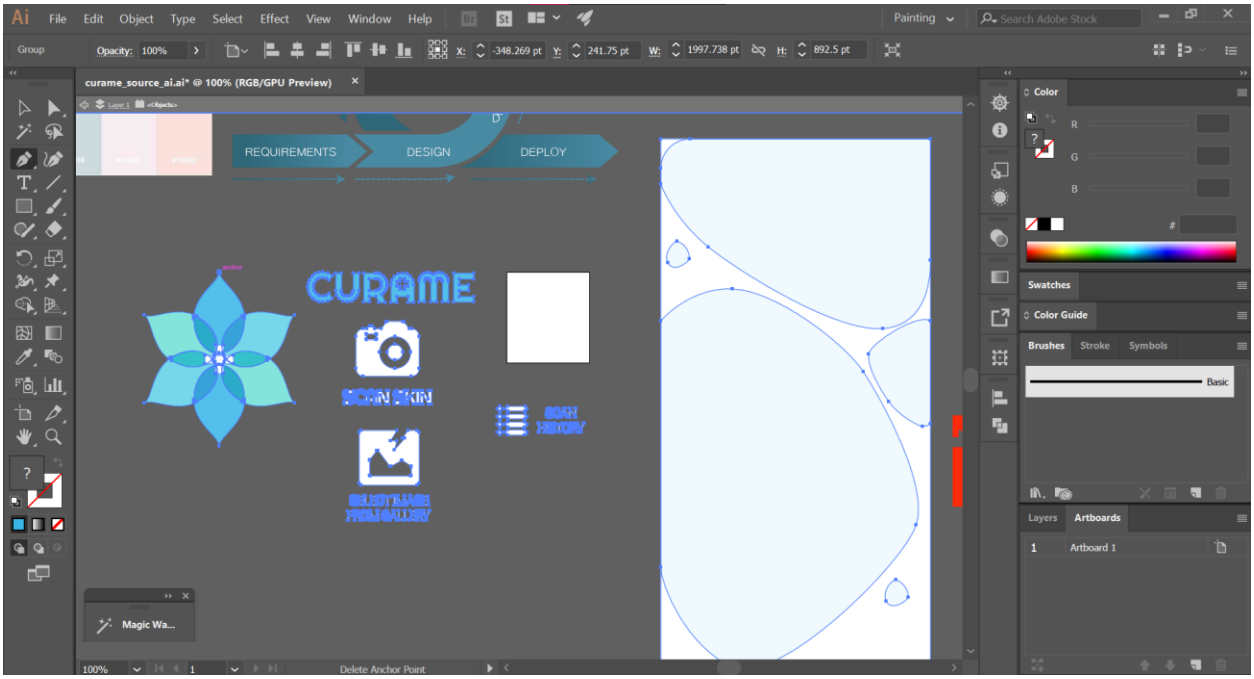


Figure 47.0: Adobe Illustrator Usage

Draw.io

The last technology used by the team is Draw.io. This is a simple website used to create the diagrams for the project deliverables such as Use Case diagrams, Class diagrams, Sequence diagrams etc. Draw.io is flexible and easy to use.

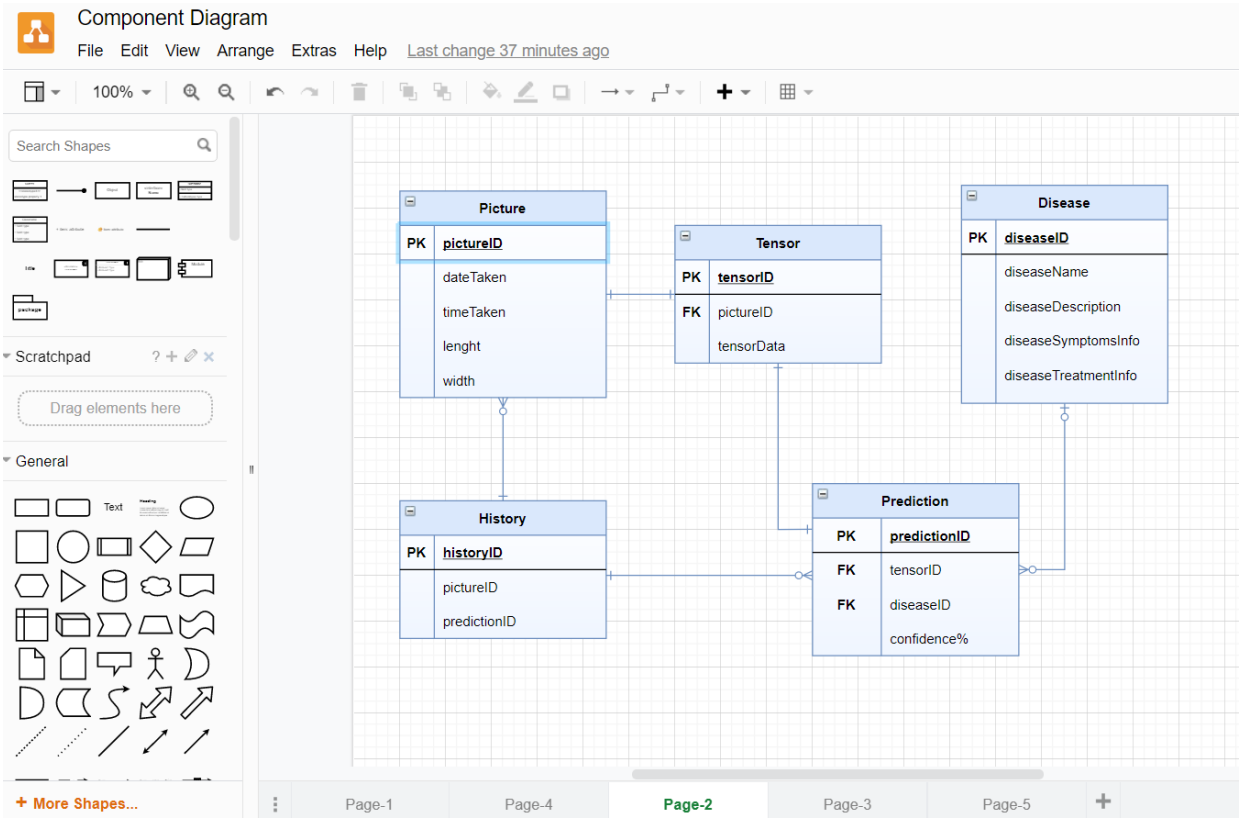


Figure 48.0: Draw.io Usage

Curame Application

Using the technologies outlined above, the Curame application was created with all the features specified in -SECTION X.X-. Figure -FIGURE X.X- below is a Navigation map of the Curame App. The diagram outlines how the user would access each feature starting from the Curame Home Page.

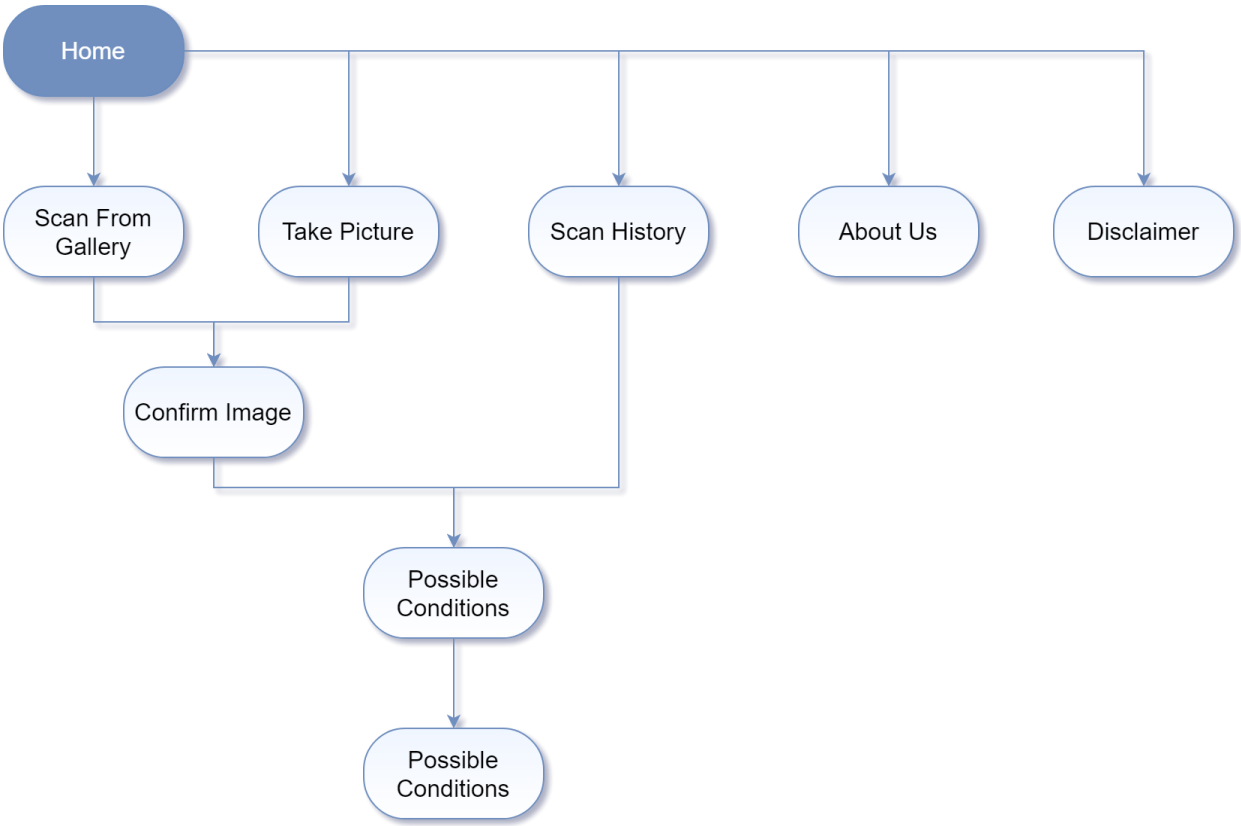


Figure 49.0: Navigation Map

Home Page

This is the main navigation page of the Curame Application. This was created using Materialize Cards for each of the options.

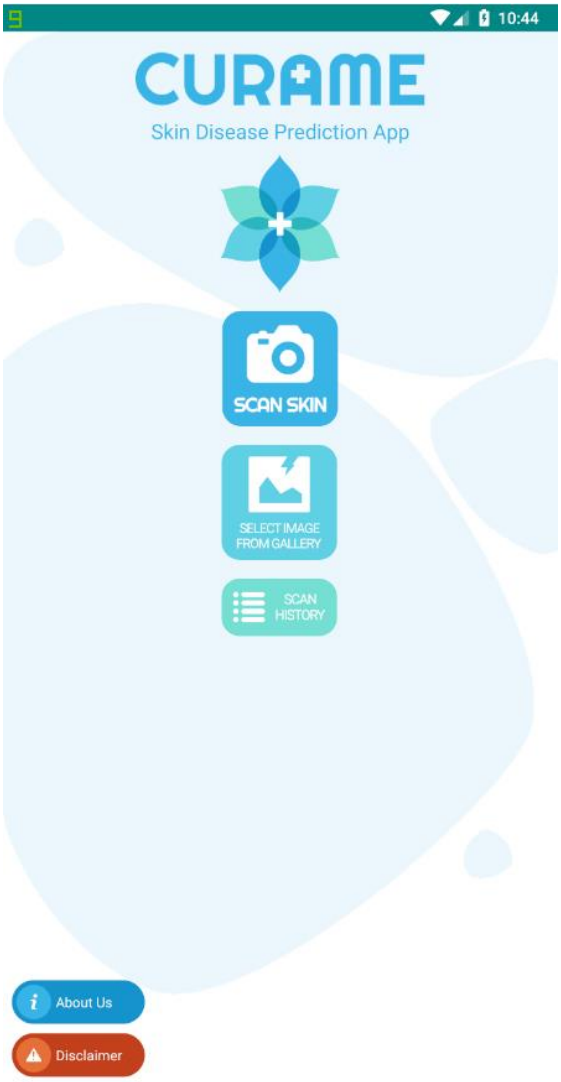


Figure 50.0: Home Interface

Scan an Image

From the home page, the user may access the ‘Scan Skin’ option. This creates an instance of the built-in Camera Activity. The user can take a picture of their skin or cancel to return to the Home Page. If an image is taken, the user would be prompted to Save the image, where the image would then be classified and then the application would list the predicted skin diseases, or select discard to retake a photo.

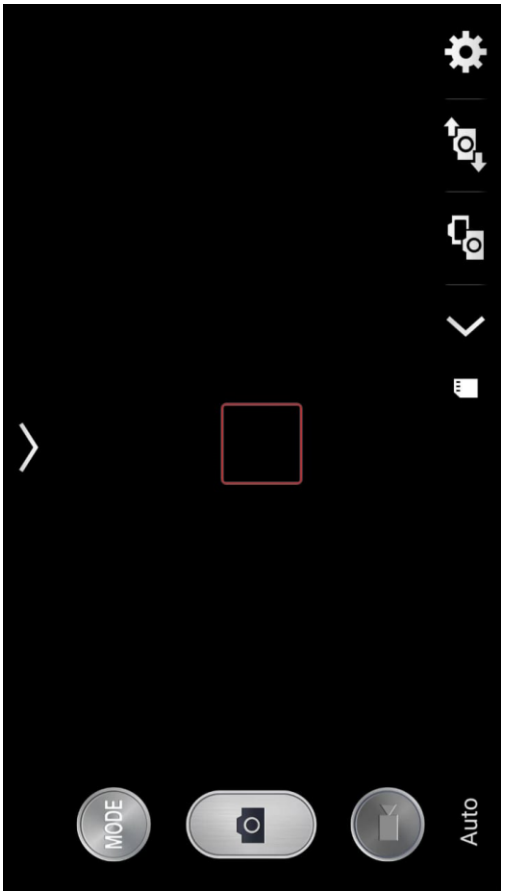


Figure 51.0: Camera Interface

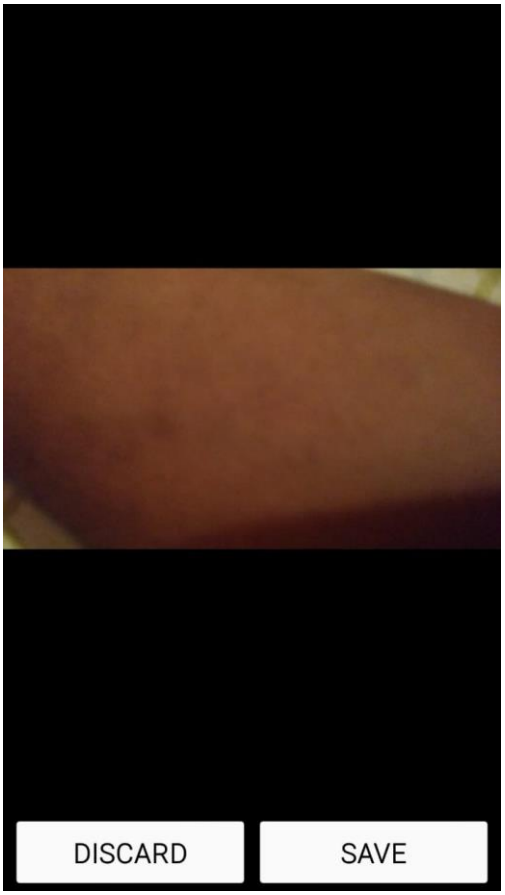


Figure 52.0: Camera Confirm Interface

Upload an Image

From the home page, the user may access the ‘Select Image from Gallery’ option. This function starts an instance of the built-in Gallery Activity where they can upload a single image. After selecting an image and hitting select, the system would start an activity that allows the user to preview the image then accept or reject it. If the image is selected, the image would then be classified and then the application would list the predicted skin diseases if the image were canceled, they may select a different picture.

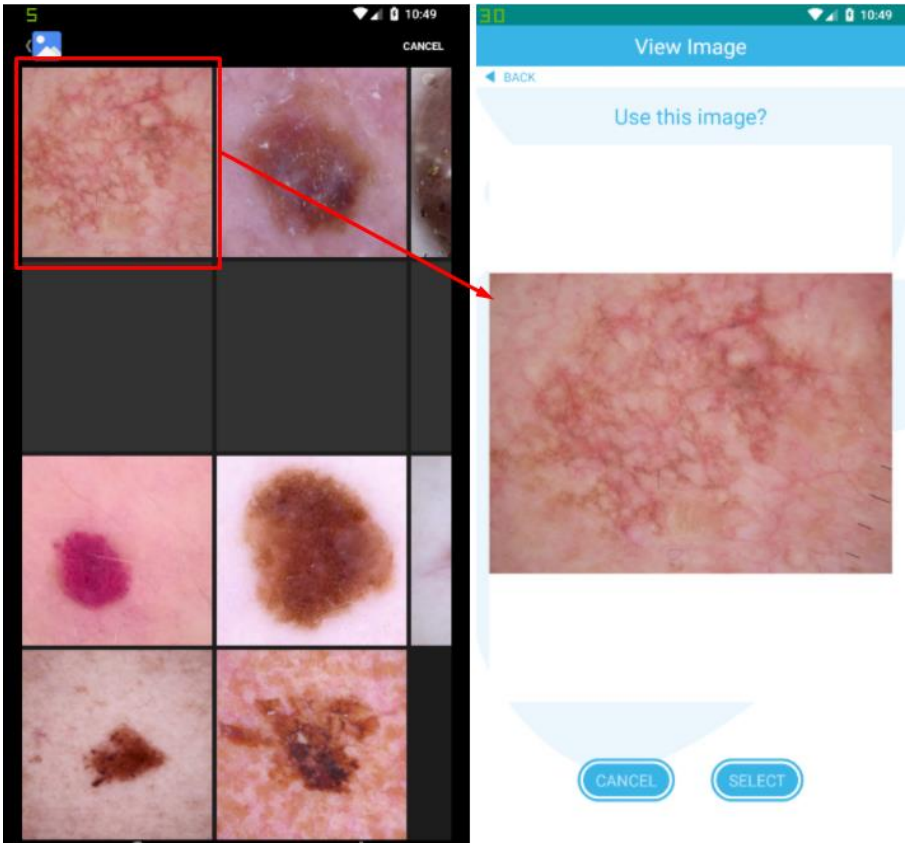


Figure 53.0: Gallery Interface

Prediction History

From the home page, the user may access the ‘Scan History’ option. When this is selected, the system gets all the scans from the local application storage and reads the data. After the data is read, it is displayed using a recycler view where the user can select one of these items to view the list of predicted skin diseases. Each item also has a trash icon where the user is prompted to delete that scan item.

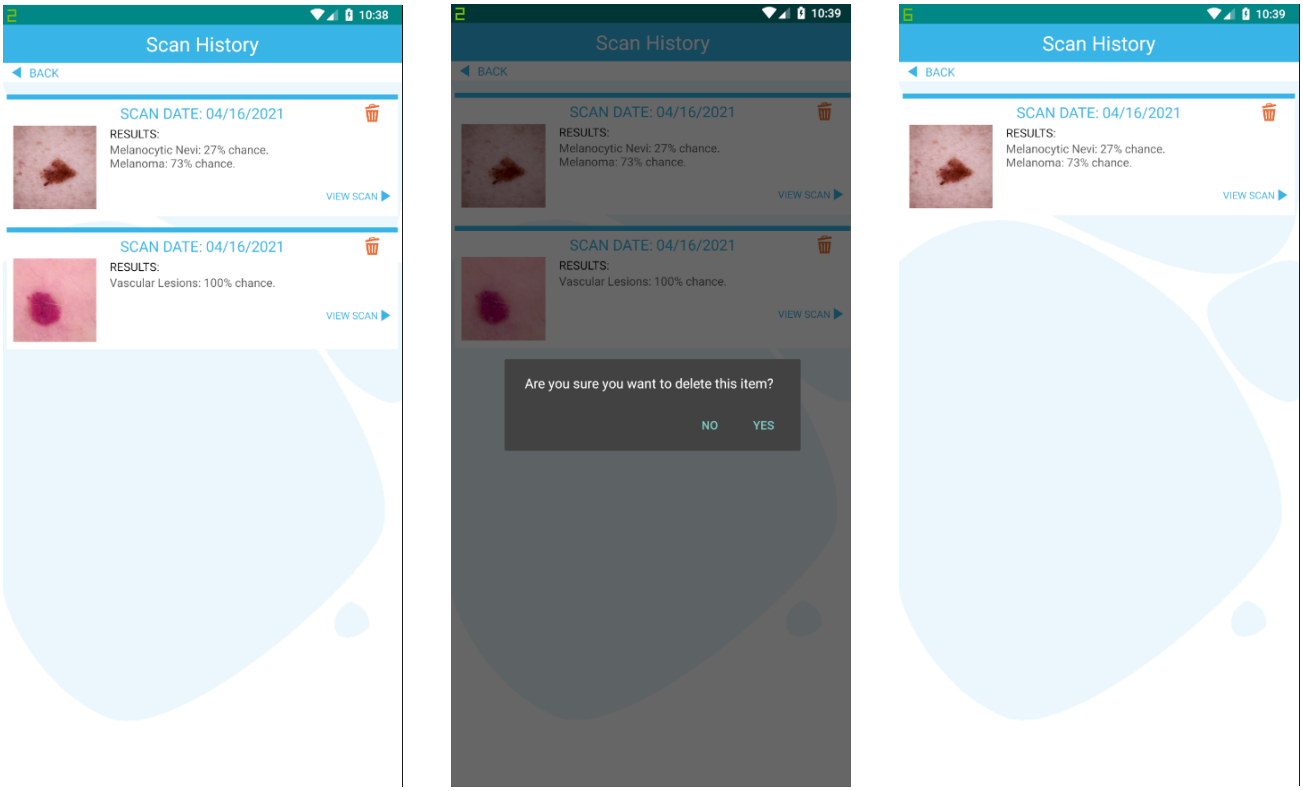


Figure 54.0: Prediction History Interface

List Predicted Skin Diseases

A list of predicted skin diseases is displayed after the user takes, uploads or selects from history an image. This is done using a Recycler View to display each predicted skin disease and their percentage chance. Any of these predictions can be selected to start an Skin Disease Information activity for that skin disease.

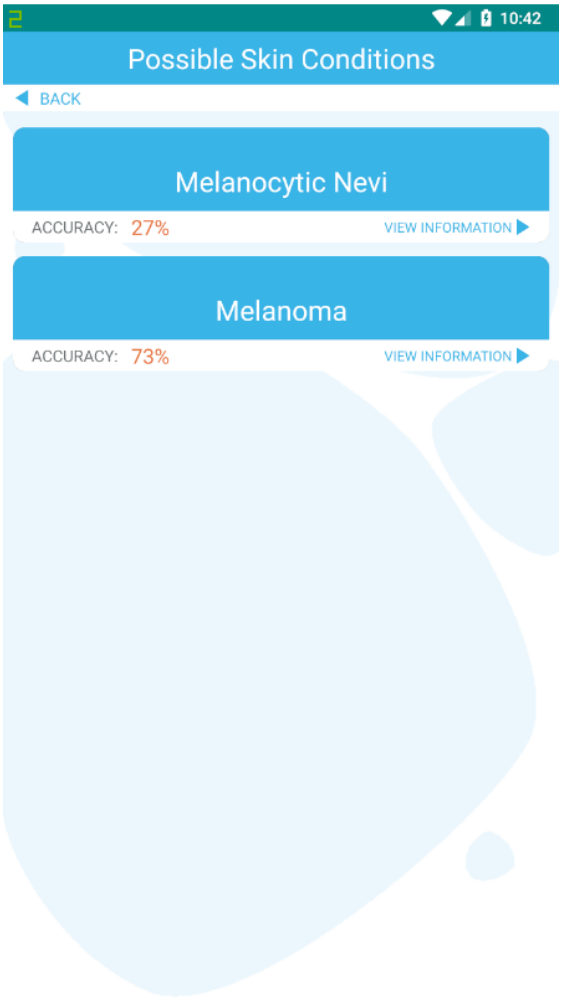


Figure 55.0: Skin Prediction List Interface

Skin Disease Information

When a skin disease is selected, the Skin Disease Information Activity is started for that selected image. On this page, the disease information is fetched from the Firebase Firestore and then displayed using a Recycler View. Users may select an image which would then have the text be read allowed by a text to speech instance.

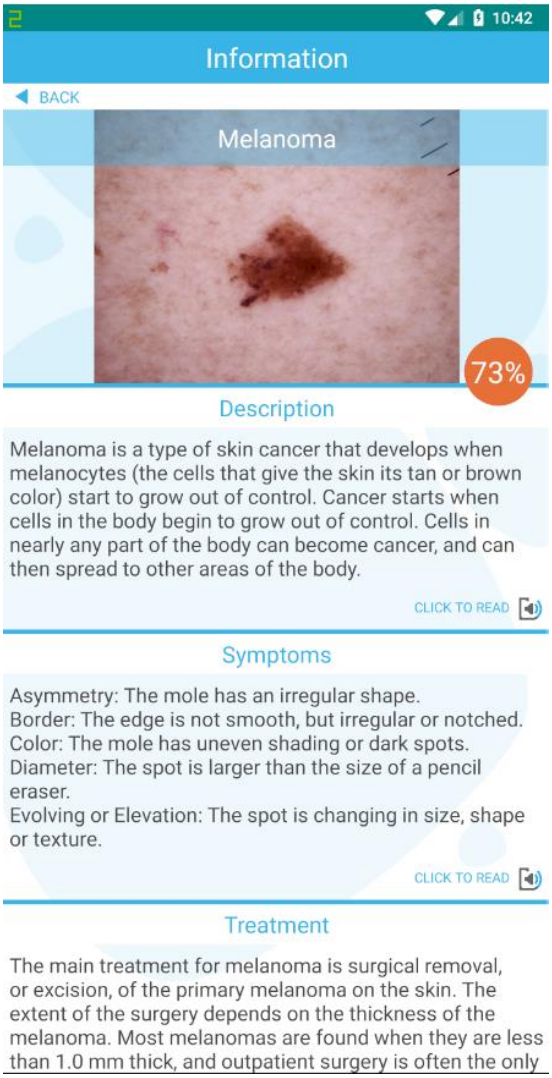


Figure 56.0: Disease Information Interface

About Us Page

The user may access the About Us page from the home screen then select anywhere to close the activity and return home.

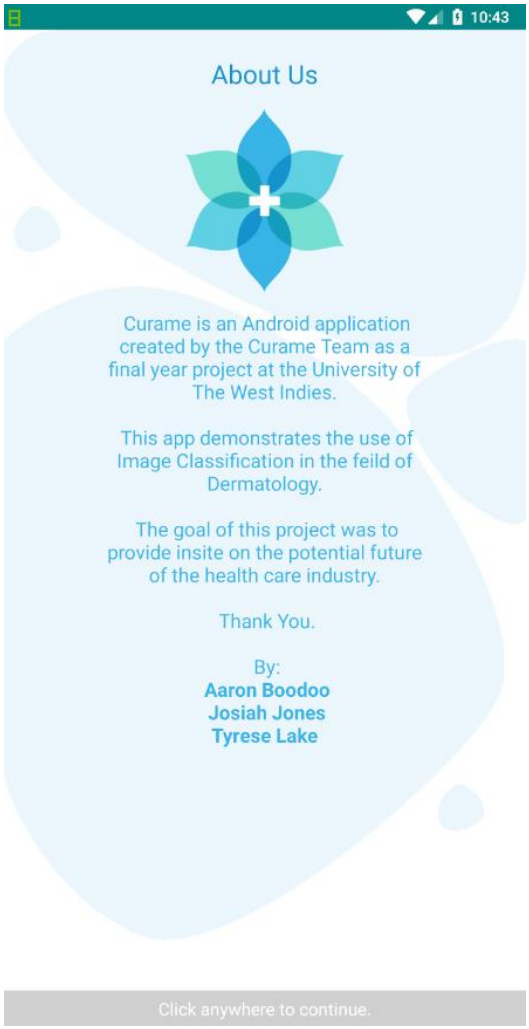


Figure 57.0: About Us Interface

Disclaimer Page

The user may access the Disclaimer page from the home screen then select anywhere to close the activity and return home.

Figure 58.0: Disclaimer Interface

Testing

Test Plan

During and After the development of the Curame Project with the use of the Agile Methodology, various test plans were created to ensure that nothing 'broke' the development process and that the final product worked as intended. Two main types of tests were used for the testing of the Curame App: Unit Tests and Component tests.

What Constitutes a Successful Application?

The system is successful if it can provide all of the requirements and is able to perform all the user stores. With respect to image classification, the team has determined the project to be successful when it allows users to take images of affected areas or upload images of affected areas that may be:

- Actinic Keratoses
- Basal Cell Carcinoma
- Benign Keratosis
- Dermatofibroma
- Melanoma
- Melanocytic Nevi
- Vascular Lesions

The system should then be able to make a prediction of which of these skin diseases the affected area may be and display the percentage. If the system thinks it may be multiple skin diseases, it should display all relevant skin diseases. With respect to displaying information, the system is successful if it can display a brief paragraph or a few points on the description, symptoms and treatment for the skin disease. Lastly, with respect to text-to-speech, simply reading the description, symptoms and treatment would be sufficient.

Unit Testing

Unit Tests were used to test the main object classes used for the Curame presentation, namely the Disease, Prediction and History object classes. These unit tests were automatic and done using JUnit5 in java and ran directly from Android Studio.

Unit Tests for Disease Class

Function - "testToArrayLists( )"

This function tests to see if the disease object returns the arraylist of arraylists with the expected output.

```
@Test
public void testToArrayLists() {
    ArrayList<String> sym = new ArrayList<>();
    sym.add("test_symptom_1");
    sym.add("test_symptom_2");
    Disease test_disease = new Disease( name: "test_name",  description: "test_description", sym,  treatment: "test_treatment");

    ArrayList<ArrayList<String>> expected_output = new ArrayList<~>();
    ArrayList<String> description = new ArrayList<~>();
    description.add("test_description");
    ArrayList<String> treatment = new ArrayList<~>();
    treatment.add("test_treatment");
    expected_output.add(description);
    expected_output.add(sym);
    expected_output.add(treatment);

    assertEquals(expected_output, test_disease.toArrayLists());
}
```

Function – "testGetHeadings( )"

This function tests to see if the getHeadings() function returns the headings declared in the disease object in an ArrayList format.

```
@Test
public void testGetHeadings() {
    ArrayList<String> sym = new ArrayList<>();
    sym.add("test_symptom_1");
    sym.add("test_symptom_2");
    Disease test_disease = new Disease( name: "test_name",  description: "test_description", sym,  treatment: "test_treatment");
    ArrayList<String> expected_headings =  new ArrayList<>();
    expected_headings.add("Description");
    expected_headings.add("Symptoms");
    expected_headings.add("Treatment");

    assertEquals(expected_headings, test_disease.getHeadings());
}
```

Function – “testGetName( )”

This function checks to see if the testGetName() function returns the correct string.

```
@Test
public void testGetName() {
    ArrayList<String> sym = new ArrayList<>();
    sym.add("test_symptom_1");
    sym.add("test_symptom_2");
    Disease test_disease = new Disease( name: "test_name", description: "test_description", sym, treatment: "test_treatment");

    assertEquals( expected: "test_name", test_disease.getName());
}
```

Unit Tests for Prediction Class

Function – “testGetData()”

This function checks to see if the current date matches up with the date returned by the Prediction object.

```
@Test
public void testGetDate() {
    Calendar calendar = Calendar.getInstance();
    SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "MM/dd/yyyy");
    String expected_date = dateFormat.format(calendar.getTime());

    Map<String, Float> valueMap = new HashMap<~>();
    valueMap.put("name1", new Float( value: 10.2));
    valueMap.put("name2", new Float( value: 21.5));

    Prediction test_prediction = new Prediction(valueMap);

    assertEquals(expected_date, test_prediction.getDate());
}
```

Function – “testGetPrediction( )”

This function checks to see if the map returned by the object is the same as the one declared.

```
@Test
public void testGetPrediction() {
    Map<String, Float> valueMap = new HashMap<~>();
    valueMap.put("name1", new Float( value: 10.2));
    valueMap.put("name2", new Float( value: 21.5));

    Prediction test_prediction = new Prediction(valueMap);

    assertEquals(valueMap, test_prediction.getPrediction());
}
```

Function – “testEquals( )”

Checks to see if two maps returned by two objects that are expected to be the same returns the equal maps. It also checks to see if a different object that is expected to return a different map returns the different map.

```
@Test
public void testEquals() {
    Map<String, Float> valueMap = new HashMap<~>();
    valueMap.put("name1", new Float( value: 10.2));
    valueMap.put("name2", new Float( value: 21.5));

    Prediction test_prediction1 = new Prediction(valueMap);
    Prediction test_prediction2 = new Prediction(valueMap);

    Map<String, Float> valueMap2 = new HashMap<~>();
    valueMap.put("name1", new Float( value: 10.5));
    valueMap.put("name2", new Float( value: 20.1));

    Prediction test_prediction3 = new Prediction(valueMap2);

    assertTrue(test_prediction1.equals(test_prediction2));
    assertFalse(test_prediction1.equals(test_prediction3));
}
```

Unit Tests for History Class

Function – “addHistoryItem( )”

Tests if the history item returns the expected image and prediction value.

```
@Test
public void addHistoryItem() {
    History test_history = new History();

    Uri test_imageUri = Uri.parse("empty.png");

    Map<String, Float> valueMap = new HashMap<~>();
    valueMap.put("name1", new Float( value: 10.2));
    valueMap.put("name2", new Float( value: 21.5));

    Prediction test_prediction = new Prediction(valueMap);

    test_history.addHistoryItem(test_imageUri, test_prediction);

    ArrayList<Uri> expected_uris = new ArrayList<>();
    expected_uris.add(test_imageUri);

    ArrayList<Prediction> expected_predictions = new ArrayList<>();
    expected_predictions.add(test_prediction);

    assertEquals(expected_uris, test_history.getImages());
    assertEquals(expected_predictions, test_history.getPredictions());
}
```

Function – “getImages( )”

Tests that the history object returns the array of image URIs stored for rendering to history to screen.

```
@Test
public void getImages() {
    History test_history = new History();

    Uri test_imageUri = Uri.parse("empty.png");

    Map<String, Float> valueMap = new HashMap<~>();
    valueMap.put("name1", new Float( value: 10.2));
    valueMap.put("name2", new Float( value: 21.5));

    Prediction test_prediction = new Prediction(valueMap);

    test_history.addHistoryItem(test_imageUri, test_prediction);

    ArrayList<Uri> expected_uris = new ArrayList<>();
    expected_uris.add(test_imageUri);

    assertEquals(expected_uris, test_history.getImages());
}
```

Function – “getPredictions( )”

Tests if the history object returns the list of predictions to render to screen.

```
@Test
public void getPredictions() {
    History test_history = new History();

    Uri test_imageUri = Uri.parse("empty.png");

    Map<String, Float> valueMap = new HashMap<~>();
    valueMap.put("name1", new Float( value: 10.2));
    valueMap.put("name2", new Float( value: 21.5));

    Prediction test_prediction = new Prediction(valueMap);

    test_history.addHistoryItem(test_imageUri, test_prediction);

    ArrayList<Prediction> expected_predictions = new ArrayList<>();
    expected_predictions.add(test_prediction);

    assertEquals(expected_predictions, test_history.getPredictions());
}
```

Function – “getHistoryItemCount( )”

Checks if the count of history items (image, date, confidence) in the history object returns as expected.

```
@Test
public void getHistoryItemCount() {
    History test_history = new History();

    Uri test_imageUri = Uri.parse("empty.png");

    Map<String, Float> valueMap = new HashMap<~>();
    valueMap.put("name1", new Float( value: 10.2));
    valueMap.put("name2", new Float( value: 21.5));

    Prediction test_prediction = new Prediction(valueMap);

    test_history.addHistoryItem(test_imageUri, test_prediction);

    assertEquals( expected: 1, test_history.getHistoryItemCount());
}
```

Component Testing

Component Testing is a type of component tests designed for testing mobile applications via interactions to the user interface. The Curame Team Designed these tests to ensure proper and complete functionality of each core component and interface of the Curame System. The components tested with Component testing were:

- Home Interface
- Camera Component
- Gallery Component
- Skin Disease Prediction Component
- Prediction History Component
- Disease Information Component
- Firebase Component

The Use Cases would determine how these elements should function to be considered as ‘Working’.

Home Interface Test Cases

Case 1 – Select ‘Scan Skin’ Option

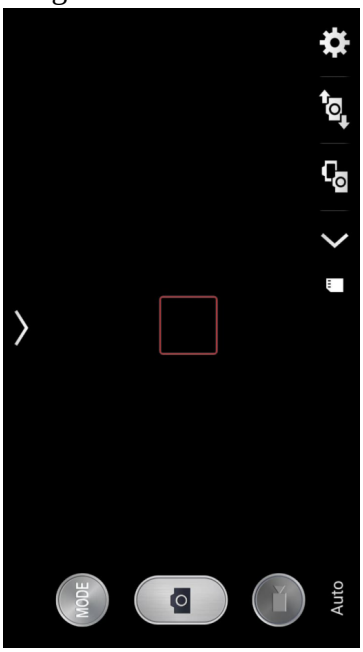
Precondition: None

Input: The ‘Scan Skin’ option is selected.

Expected Result: The built-in Android Camera is started.

Actual Result:

The Camera Interface was started.
Image Result:



Status: Pass

Case 2 – Select ‘Select Image from Gallery’ option

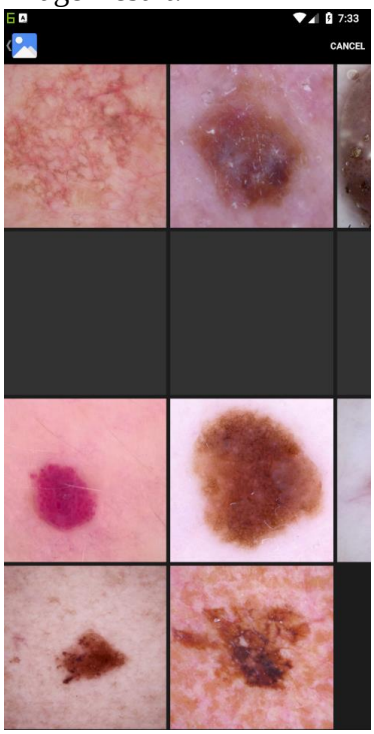
Precondition: None

Input: The ‘Upload Image from Gallery’ option is selected.

Expected Result: The built-in Android Gallery is started.

Actual Result:

The Gallery Interface Opens.
Image Result:



Status: Pass

Case 3 – Select ‘Scan History’ Option

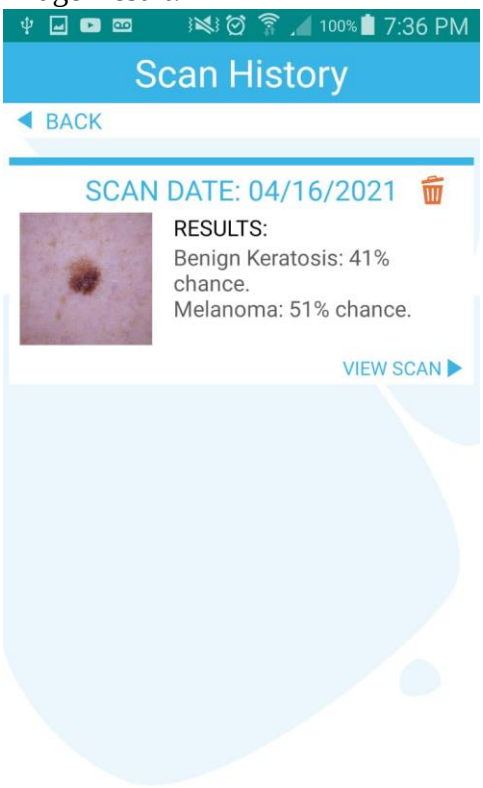
Precondition: None

Input: The ‘Scan History’ option is selected.

Expected Result: The user is taken to the Scan History Page

Actual Result:

The Scan History Page opens.
Image Result:



Status: Pass

Case 4 – Select ‘About Us’ Option

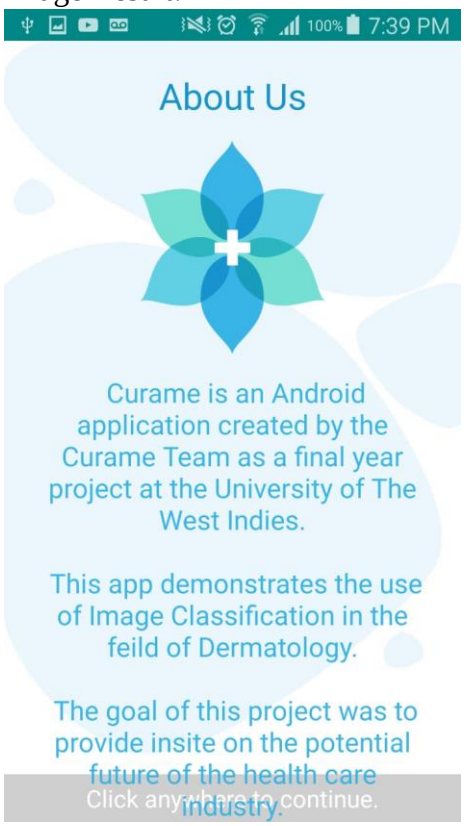
Precondition: None

Input: The ‘About Us’ option is selected.

Expected Result: The user is taken to the About Us Page

Actual Result:

The about us page opens.
Image Result:



Status: Pass

Case 5 – Select ‘Disclaimer’ Option

Precondition: None

Input: The ‘Disclaimer’ option is selected.

Expected Result: The user is taken to the Disclaimer Page

Actual Result:

Disclaimer Page opens
Image Result:

Status: Pass

Camera Component Testing

Case 1 – ‘Taking an Image’

Precondition: The has selected the ‘Scan Skin’ option from the home page

Input: The user positions the camera and takes an image of their skin.

Expected Result:

The Application prompts the user if they would like to use the taken image.

Actual Result:

Application successfully prompted user

Status: Pass

Case 2 – Select ‘Back’ after opening the Camera

Precondition: The has selected the ‘Scan Skin’ option from the home page

Input: The user pressed the back button on their device

Expected Result:

The Camera Activity is ended, and the user is returned to the Home interface.

Actual Result:

The activity closes and the app returns to the home screen.

Status: Pass

Case 3 – After taking an image, User discards the image.

Precondition: The user has taken an image of the affected error

Input: The user selected the ‘Discard’ option.

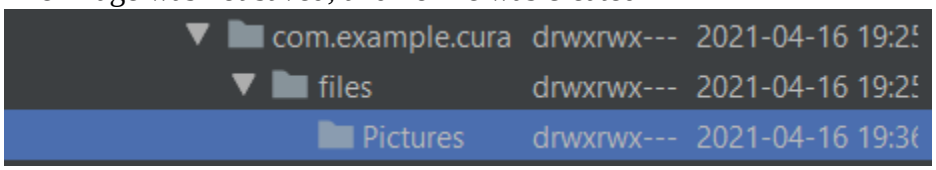
Expected Result:

The built-in Android Camera is restarted, and the user is prompted to retake the image.

The image is not saved to the directory.

Actual Result:

The app prompts the user to retake the image.
The image was not saved, and no file was created



Status: Pass

Case 4 – After taking an image, the user ‘Saves’ the image.

Precondition: The user has taken an image of the affected error

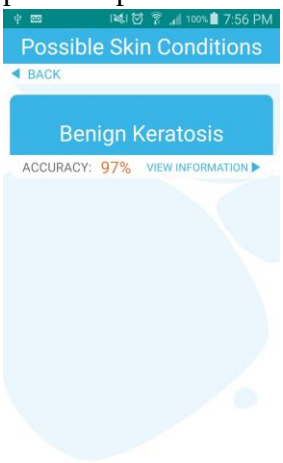
Input: The user selected the ‘Save’ option.

Expected Result:

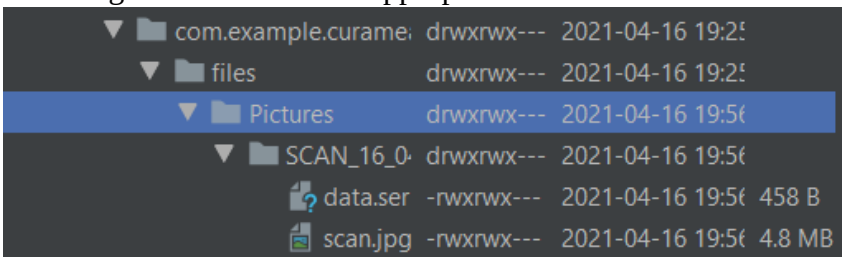
The Android Camera Activity is closed, and the Prediction Select Activity is started.
The image is saved to its own directory and named “scan.jpg”.
The path of image is passed to the Skin Disease Prediction Component to be classified.

Actual Result:

The image was passed to the classification activity which would display the possible predictions screen.



The image is saved with the appropriate name.



Status: Pass

Gallery Component Testing

Case 1 – User selects an image from the Gallery.

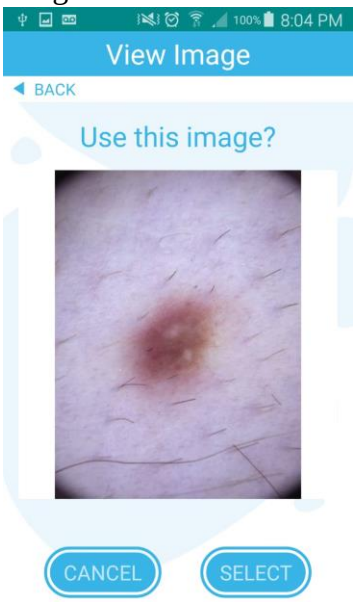
Precondition:
The has selected the ‘Select Image from Gallery’ option from the home page

Input:
The user selects a folder then selects an image from the gallery.

Expected Result:
The Application displays an Interface displaying the image and prompts the user to select the image or cancel the image.

Actual Result:

Image and correct prompt was displayed.
Image Result:



Status: Pass

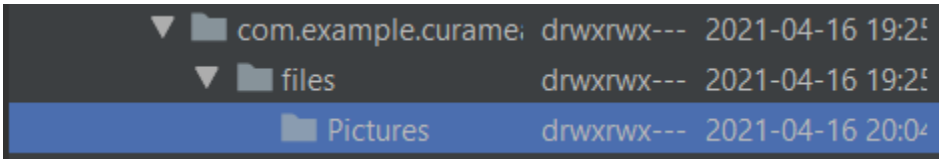
Case 2 – User selects an image, then selects the ‘back’ button.

Precondition:
The has selected the ‘Select Image from Gallery’ option from the home page

Input:
The user pressed the back button on their device

Expected Result:
The Gallery Activity is ended, and the user is returned to the Home interface.
The image is not saved to the directory.

Actual Result:
The current activity ended, and the app returned to the Home screen.
No file was created.



Status: Pass

Case 3 – User selects an image, the selects ‘Cancel’

Precondition: The user has selected an image from the gallery.

Input: The user selected the ‘Cancel’ option.

Expected Result:

The built-in Android Gallery is restarted, and the user is prompted to select an image.

The image is not saved to the directory.

Actual Result:

The activity ends and the gallery is reopened, prompting the user to select again.

Status: Pass

Case 4 – The user selects and image, then confirms their selection.

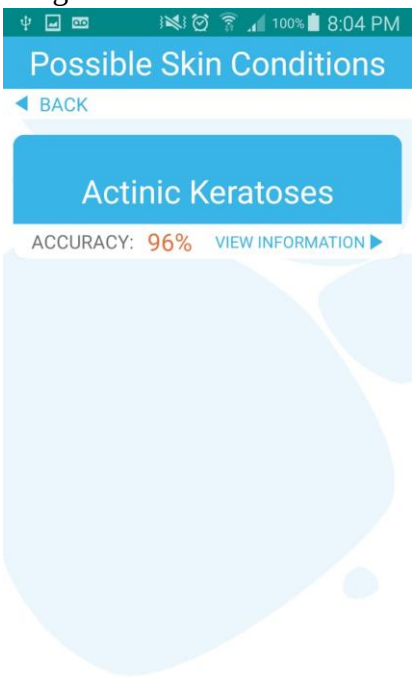
Precondition: The user has selected an image from the gallery.

Input: The user selected the ‘Select’ option.

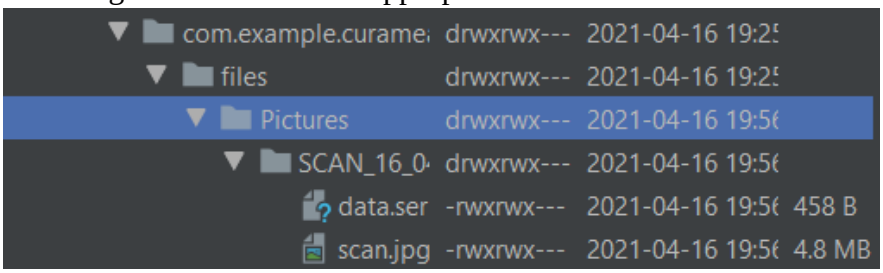
Expected Result:
The Android Gallery Activity is closed, and the Prediction Select Activity is started.
The image is saved to its own directory and named “scan.jpg”.
The path of the image is passed to the Skin Disease Prediction Component to be classified.

Actual Result:

The image was passed to the classification activity which would display the possible predictions screen.
Image Result:



The image is saved with the appropriate name.



Status: Pass

Skin Disease Prediction Component Test Case

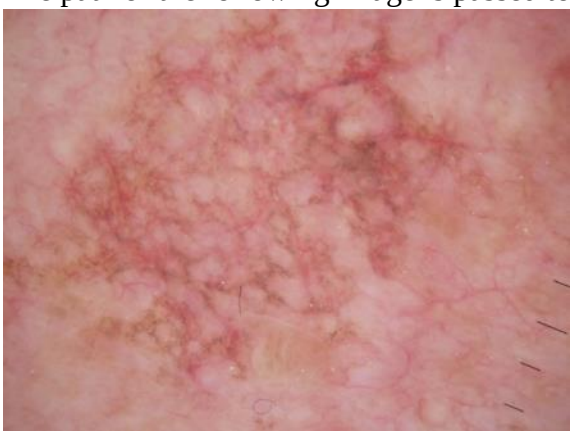
Case 1 - Valid Input 1 (Actinic Keratosis)

Precondition:

The user has taken an image from the Camera Activity and selected the ‘Save’ option or the User has selected an image from the gallery activity and selected the ‘Select’ option.

Input:

The path of the following image is passed to the skin disease classifier:



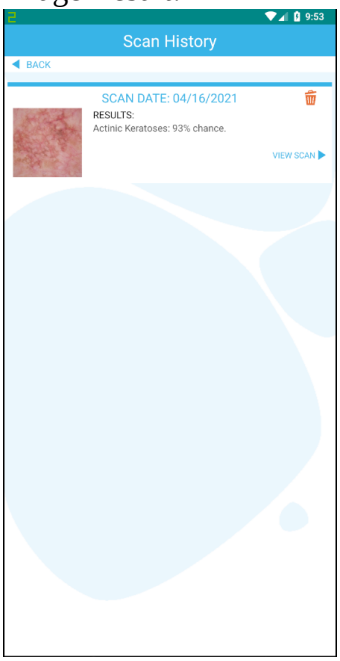
Expected Result:

The prediction: [Actinic Keratosis: x%] is made and the prediction is saved. Where x is the highest percentage.
The Select Skin Disease Prediction is displayed.

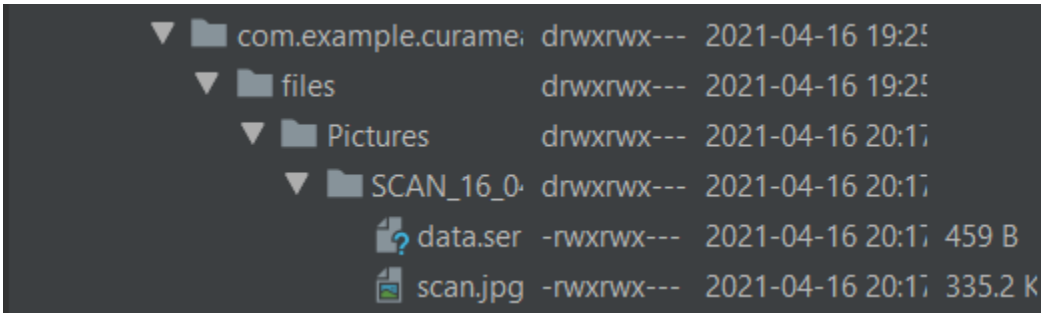
Actual Result:

The following prediction is made:
{Actinic Keratoses=0.90247995, Basal Cell Carcinoma=0.038877722, Benign Keratosis=0.020805284, Dermatofibroma=3.0840843E-6, Melanoma=0.03492584, Melanocytic Nevi=0.0027889735, Vascular Lesions=1.190634E-4}

Image Result:



Actinic Keratoses is the highest percentage.
The prediction is serialized and saved in the same location as the image passed to the skin disease classifier as “data.ser”.



The Select Skin Disease Prediction is displayed for the predictions made.

Status: Pass

Case 2 - Valid Input 2 (Vascular Lesions)

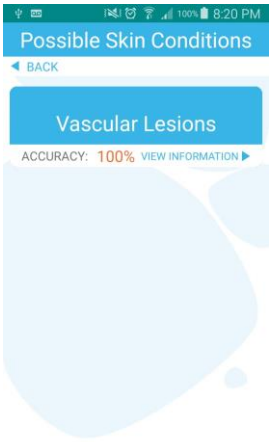
Precondition:
The user has taken an image from the Camera Activity and selected the ‘Save’ option or the User has selected an image from the gallery activity and selected the ‘Select’ option.

Input:
The path of the following image is passed to the skin disease classifier:

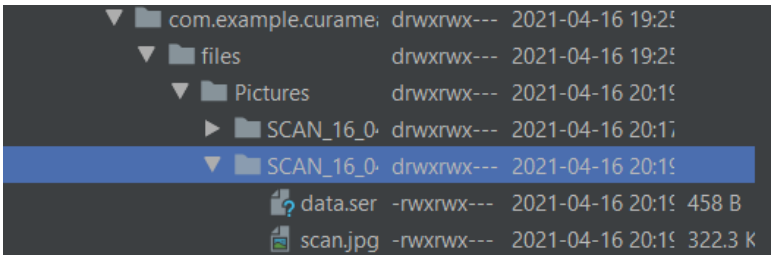
Expected Result:

The prediction: [Vascular Lesions: x%] is made and the prediction is saved, where x is the highest %.
The Select Skin Disease Prediction is displayed.

Actual Result:
The following prediction is made:
{Actinic Keratoses=0.0012792893, Basal Cell Carcinoma=7.451528E-5, Benign Keratosis=2.6897521E-4, Dermatofibroma=3.787315E-9, Melanoma=8.4383455E-6, Melanocytic Nevi=3.3748986E-6, Vascular Lesions=0.99836546}



Vascular Lesions is the highest percentage.
The prediction is serialized and saved in the same location as the image passed to the skin disease classifier as “data.ser”.



The Select Skin Disease Prediction is displayed for the predictions made.

Status: Pass

Case 3 - Invalid Input

Precondition:
The user has taken an image from the Camera Activity and selected the ‘Save’ option or the User has selected an image from the gallery activity and selected the ‘Select’ option.

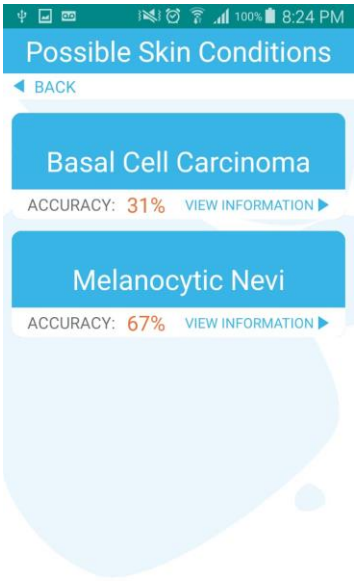
Input:
The path of the following image is passed to the skin disease classifier:

This is not a valid skin disease.

Expected Result:
The system informs the user that the image taken is not a valid image and prompts the user to reselect the image.
The image and directory is deleted, and no prediction is made.

Actual Result:

The following prediction is made:



The prediction is serialized and saved.

Status: Fail

Reason:

Currently, the image classification model was only trained to predict 7 selected diseases and thinks that this image is indeed a skin disease. The classifier was not yet trained to distinguish between what is a skin disease and what is not.

Prediction History Component

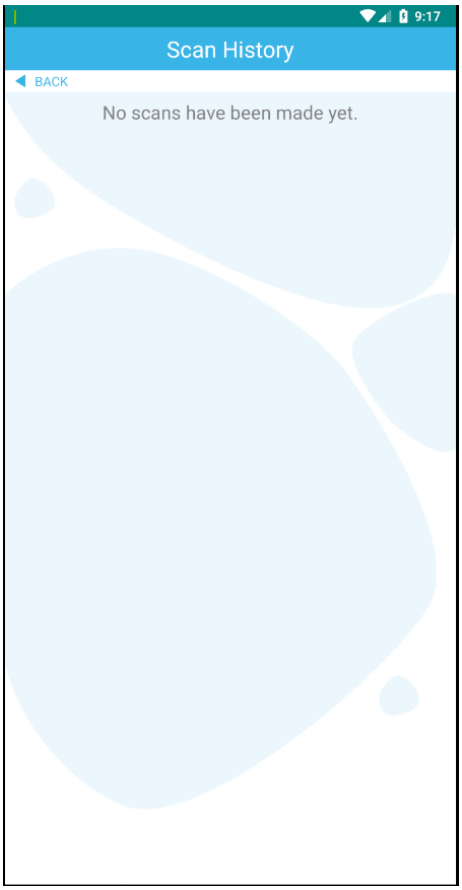
Case 1 – History with no predictions made

Precondition:
No scans have been made yet. The User has selected the “Scan History” option from the Home Interface.

Input: None

Expected Result:
An appropriate message is displayed.

Actual Result:



Status: Pass

Case 2 – History with 1 scan made

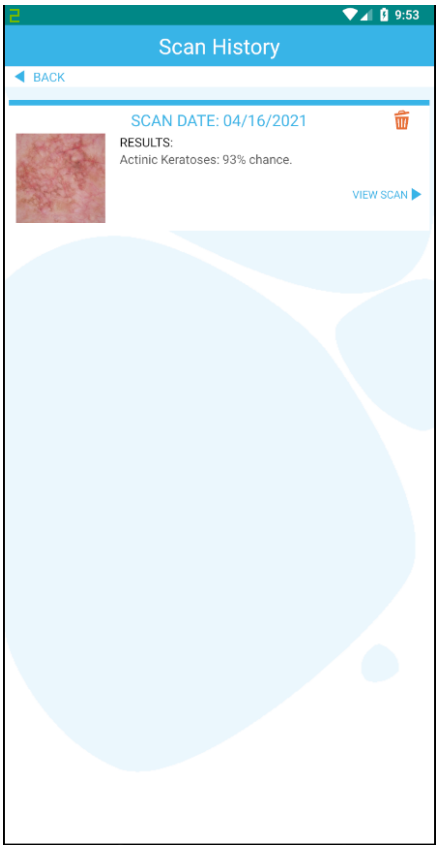
Precondition:
1 Scan has been made. The User has selected the “Scan History” option from the Home Interface.

Input:
The following image was scanned:

Expected Result:
The 1 scan item is displayed for the correct image

Actual Result:

1 scan made and displayed
Image Result:



Status: Pass

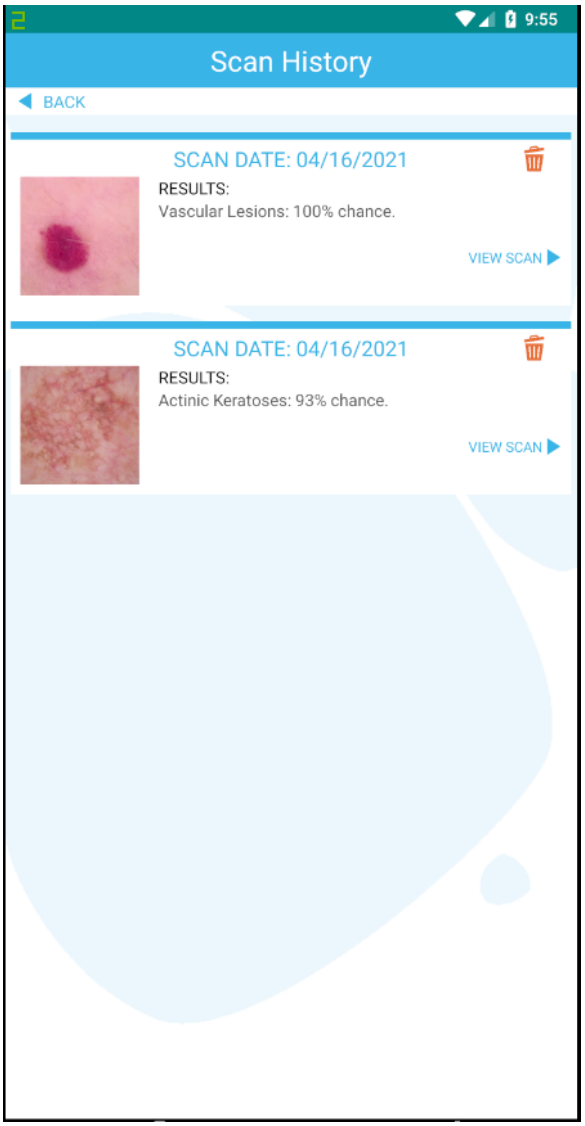
Case 3 – Scan History with Many Scans.

Precondition:
Multiple scans have been made of different images at different times. The User has selected the "Scan History" option from the Home Interface.

Input:
None

Expected Result:
All scanned items are displayed in order of date.

Actual Result:



Status: Pass

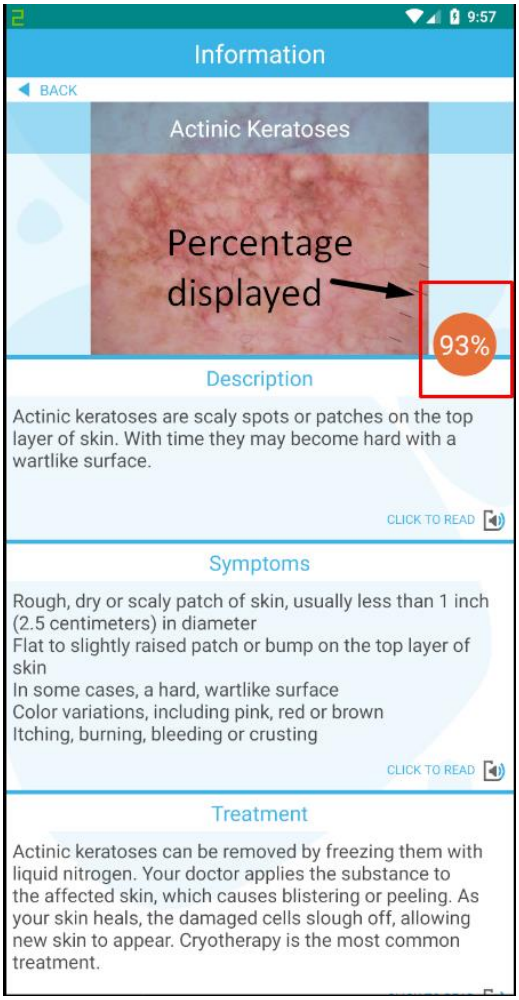
Case 4 – Scan Selected from History

Precondition:
At least 1 scan has been made. The User has selected the “Scan History” option from the Home Interface.

Input:
The user selects one of the scans

Expected Result:
The Select Skin Disease Prediction is displayed.

Actual Result:
The Select Skin Disease Prediction is displayed for the selected scan.



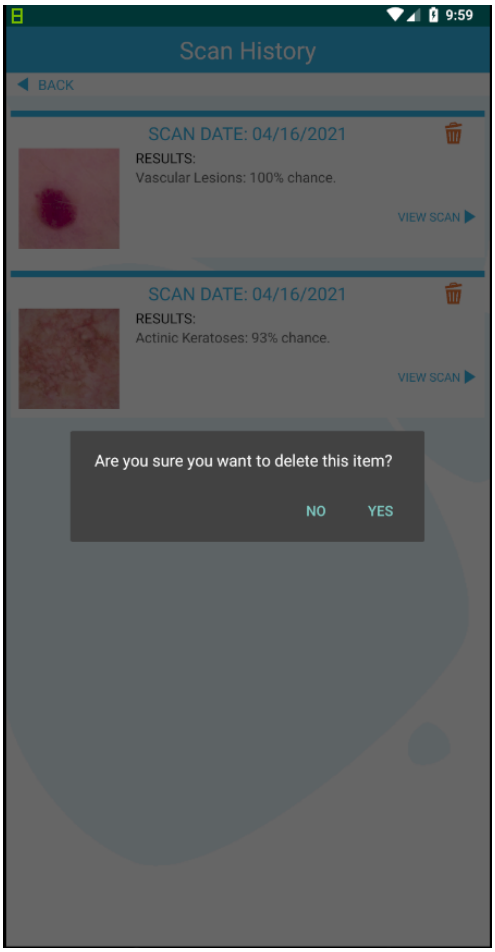
Status: Pass

Case 5 – User selects the ‘trash’ button while viewing the history.

Precondition:
At least 1 scan has been made. The User has selected the “Scan History” option from the Home Interface.

Input:
The user selects the ‘trash’ button above an item.

Expected Result:
The User is prompted if they are sure they want to delete the image



Actual Result:

Status: Pass

Case 6 – User selects ‘no’ when prompted to delete item from history.

Precondition:

The user selects the ‘trash’ button above an item on the ‘Scan History’ interface

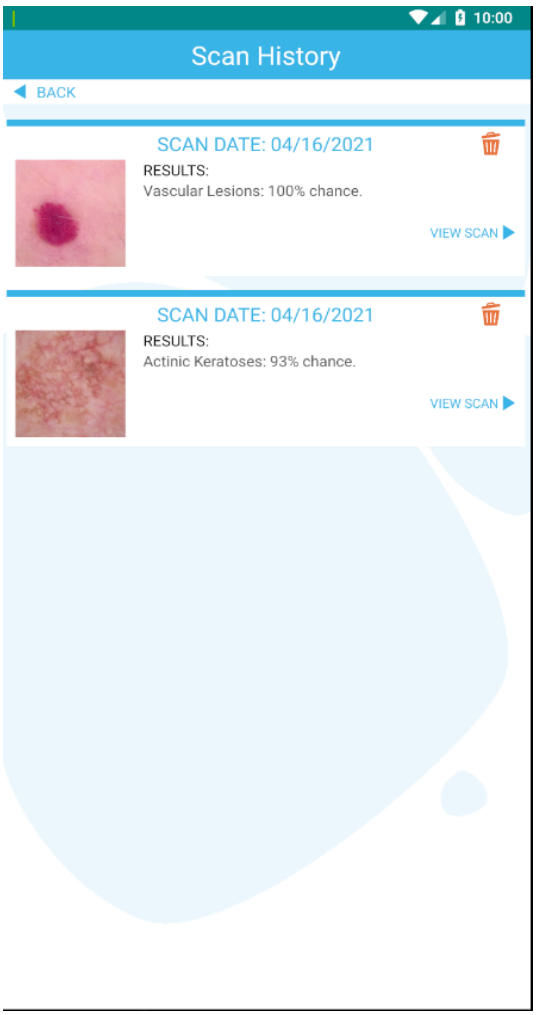
Input: The user selects the ‘NO’ in the prompt

Expected Result:

The prompt closes and nothing happens.

Actual Result:

The prompt closes and nothing happens.



Status: Pass

Case 7 – User selects ‘yes’ when prompted to delete an item from history.

Precondition:

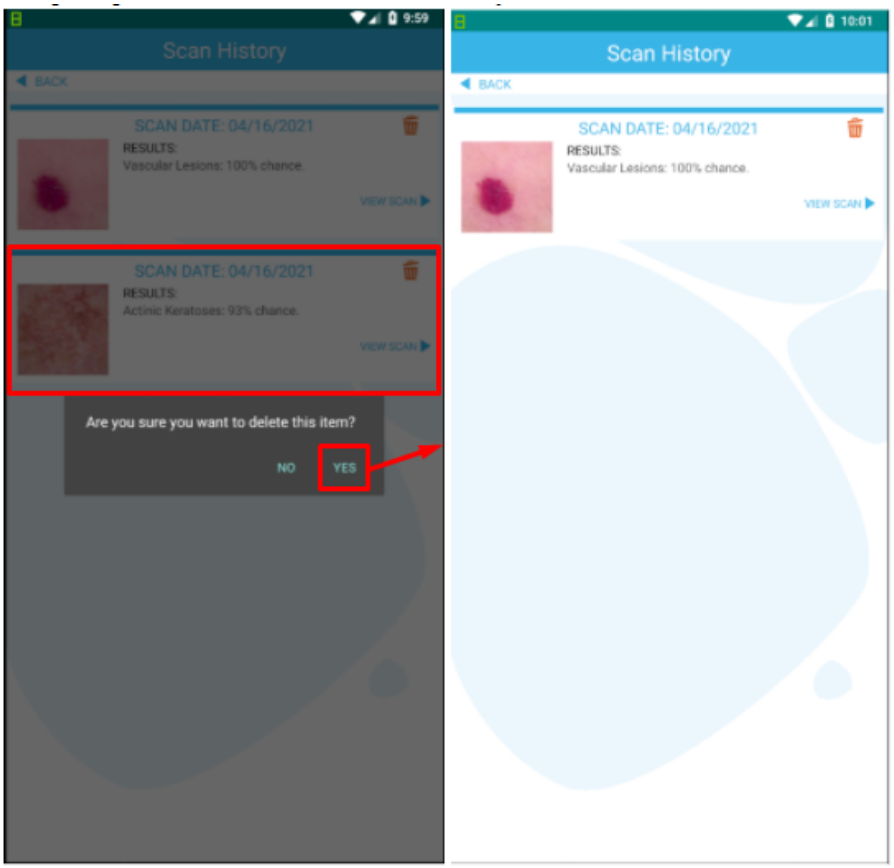
The user selects the ‘trash’ button above an item on the ‘Scan History’ interface

Input:
The user selects the ‘YES’ in the prompt

Expected Result:
The prompt closes, the selected scan history item is deleted.

Actual Result:

The prompt closes, the selected scan history item is deleted and the scan history interface is reloaded.



Status: Pass

Disease Information Component

Case 1 – User views Information all predictions made for a disease.

Precondition:

An image is finished classifying, or a history item is selected from the Scan History Interface, and the Prediction, and Image, is sent to the Disease Information Component.

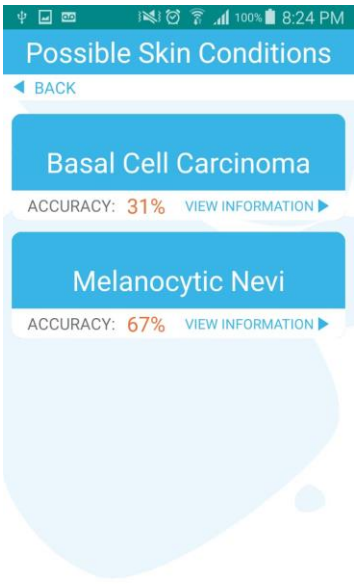
Input:
The following prediction was passed:

Prediction	
date	04/15/2021
prediction	[“Basal Cell Carcinoma: 31%”, “Melanocytic Nevi: 67%”]

Expected Result:
Each item is displayed on the Select Skin Disease Prediction Interface.

Actual Result:

All required diseases were displayed.
Image Result:



Status: Pass

Case 2 – User selects a disease from the predictions list

Precondition:

An image is finished classifying, or a history item is selected from the Scan History Interface, and the Prediction, and Image, is sent to the Disease Information Component.

Input:
The user selects a skin disease from the list.
[Date: 04/15/2021, “Actinic Keratosis: 90%”]

Expected Result:
The Display Skin Disease Information Interface is displayed

Actual Result:

All information for the disease were displayed
Image Result:

The Display Skin Disease Information Interface is displayed for the selected disease.

Status: Pass

Case 3 – System fetches the result from Firebase server after a disease is select.

Precondition:

The user selects a skin disease from the list of skin diseases on the Select Skin Disease Prediction Interface.

Input:
The selected Skin Disease is {“Actinic Keratosis: 90%”}

Expected Result:
The image, name and percentage are displayed on the interface along with the correct fetched description, symptoms, and treatment for the disease

Actual Result:
The image is displayed from the image URI.
The name and percentage are displayed.
Image Result:

A fetch request was made to the Firebase server and the description, symptoms and treatment was fetched and retrieved without error.

The fetch results were:

“Document Snapshot data: {name=Actinic Keratoses, description=Actinic keratoses are scaly spots or patches on the top layer of skin. With time they may become hard with a wart-like surface., symptom=[Rough, dry or scaly patch of skin, usually less than 1 inch (2.5 centimeters) in diameter, Flat to slightly raised patch or bump on the top layer of skin, In some cases, a hard, wart-like surface, Color variations, including pink, red or brown, Itching, burning, bleeding or crusting], treatment=Actinic keratoses can be removed by freezing them with liquid nitrogen. Your doctor applies the substance to the affected skin, which causes blistering or peeling. As your skin heals, the damaged cells slough off, allowing new skin to appear. Cryotherapy is the most common treatment.}"

This matches the content on the Firebase Server.

Status: Pass

Case 4 – Fetch is made when the user is unable to connect to FB server.

Precondition:
The user selects a skin disease from the list of skin diseases on the Select Skin Disease Prediction Interface. The firebase server is down, or the internet is down.

Input:
The selected Skin Disease is {“Melanoma: 80%”}

Expected Result:
The image, name and percentage are displayed and the appropriate error message is displayed.

Actual Result:

The image is displayed from the image URI

The name and percentage are displayed

A fetch request was made to the Firebase server, the fetched failed and “Failed to get information from server.” was displayed to the user

Status: Pass

Case 5 – User uses Text-to-Speech functionality

Precondition:
The prediction {“Melanoma: 80%”} was selected and information on the Melanoma disease was selected.

Input:
The Description is selected by the user.

Expected Result:
The content of the Description is read aloud to the user using text-to-speech.

Actual Result:

An instance of the text-to-speech class is created, and the text is passed to the instance which reads the text out loud.

Status: Pass

Integration Testing

After each iteration, the Curame Team conducted an Integration test. These are tests done on multiple components of the Curame system integrated together. These were done to ensure that nothing broke in the system during development. All bugs discovered during this process were rectified.

System Testing

After the entire system was developed by the Curame team, all components were integrated together and tested all at once. These were done to verify the system's function and nonfunctional requirements as well as the interaction between the components of the system. This was done by inputting several images into the system with both the camera and gallery and checking if the predictions made were outputting the correct result. System testing also consisted of executing the instructions outlined in the user stories.

Acceptance Testing

Acceptance testing was conducted after the unit, component and system testing had taken place for the Curame App. To carry out these tests, various test scenarios were identified and developed for acceptance testing. These tests will be derived from the following documents: use cases, user stories, sequence diagrams, class diagrams and test plans. Acceptance testing was carried out by the Curame Team as well as various alpha testers. These alpha testers were composed of students and adults. Any acceptance tests that failed were fixed and retested. The acceptance test phase was not completed until all acceptance tests were passed successfully.

Features that were not tested

In the testing process, the team was not able to fully test certain aspects of the Skin Disease Classifier with respect to the Camera Activity. This was due to the inability to access someone who had a skin disease to be tested in person. As such, tests remained with uploaded images or images of printed skin diseases.

Business Aspect

Due to the recent global outbreak of the Covid-19 virus known today as, “Coronavirus”, this has brought about an increase in the number of individuals seeking medical aid. As a result, the global healthcare economy has incurred a vast pool of challenges as the current system was not designed to cope with such a crisis. This situation has now forced the global healthcare industry to begin shifting from traditional diagnosis and treatment methods to more automated and contactless solutions via use of emerging technologies.

Our application/system, “Curame”, seeks to take advantage of this Covid-19 crisis by developing a mobile android application which provides one of the previously mentioned technologies for potentially aiding the global medical sector. This application targets individuals seeking a secondary means to measure, determine, and make predictions on skin conditions/ailments/diseases that they may have. The main functionality of the application presents itself in the form of allowing users to either scan their skin or upload an image of a skin condition via their mobile android device, after which the system would then return an educated guess on which skin condition/disease/ailment the user may potentially possess. Along with this prediction of the condition, a trove of detailed information on the predicted skin condition is returned to the user, along with possible treatments &/ remedies for such.

The marketability & novelty of our application is very high as it:

1. Is one of a kind in nature (there are little to no applications that exist which are similar in nature and functionality to our system), suggesting that there is next to zero competition within the respective field.
2. Possess a simple yet aesthetic user interface, optimized for operation and visibility for audiences of any/all ages 13 and up (13+). The application also permits the usage of a text-to-speech feature which reads all contents displayed on the current interface out loud to the user. The feature greatly aids any visually impaired user in operating the application, heightening the overall level of usability of the system even further.

Main Flaw/Issue:

One of the main issues with our application presents itself in the level of prediction accuracy. While our application presents a high enough accuracy of 75-80% to the point where it can provide a fairly accurate prediction, it cannot yet be fully utilized as was intended unless an accuracy of 95% and above is achieved. This suggests that, the stage at which the app is currently at, it cannot yet be used as a 2nd hand tool for diagnostics.

Individual Contributions

Tyrese Lake - Lead android application programmer and graphic artist [avatar image]

Responsible for:

- Documentation
- Training MobileNet
- Android components to fetch, store, delete and display images and information.
- App design using Figma
- App design using materialize
- Icon and Layout Development
- Unit testing, Component Testing, Cross Validation between models, System testing

Aaron Boodoo - Lead python programmer [avatar image]

Responsible for:

- Cleaning data for training
- Augmenting data for training and creating training/validation directories
- Training MobileNet
- Converting Protobuf to TF-Lite format
- TensorFlow integration into the Android application
- Unit testing, Component Testing, Cross Validation between models
- Debugging
- Documentation (JavaDocs)
- Connecting Firestore API and writing fetch queries

Josiah Jones - Product Owner / Lead Documenter / Lead Presentation Coordination

Responsible for:

- Coordinating the project
- Training MobileNet
- Creating and populating the firebase database
- Developing project architecture
- App design using Figma
- Documentation
- Creating demonstrations for the model using Python
- Running inference on TF-Lite model through python for testing
- Developing presentations for stakeholders
- Creating videos and planning out presentations.

COCOMO (Cost Projections)

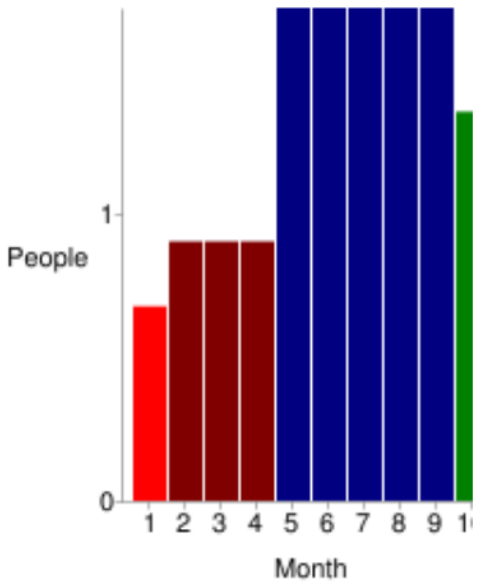
Results

Software Development (Elaboration and Construction)

Effort = 11.6 Person-months
Schedule = 8.2 Months
Cost = $93006

Total Equivalent Size = 2449 SLOC
Effort Adjustment Factor (EAF) = 1.53

Staffing Profile



Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	0.7	1.0	0.7	$5580
Elaboration	2.8	3.1	0.9	$22322
Construction	8.8	5.2	1.7	$70685
Transition	1.4	1.0	1.4	$11161

Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.1	0.3	0.9	0.2
Environment/CM	0.1	0.2	0.4	0.1
Requirements	0.3	0.5	0.7	0.1
Design	0.1	1.0	1.4	0.1
Implementation	0.1	0.4	3.0	0.3
Assessment	0.1	0.3	2.1	0.3
Deployment	0.0	0.1	0.3	0.4

Figure 59.0: COCOMO Output

Budget

Baseline Costs	Prices
Developer Salary (Three Months Duration)	$6,000 USD * 3 = $18,000 USD
Advertisement and Marketing	$1000 USD
Maintenance	$1000 USD
Google Collab	$9.99 * 3 = $30 USD
Firebase Database	FREE
Total	$20,030 USD
Projected Revenue	N/A

Figure 60.0: Budget Outline

Non-Profit Options

This project is non-profit. As such, bank financing can be a suitable option to initial development cost. Maintenance can be funded by donation-based websites such as Parteon.com.

Conclusions

Level of Completion

The project team's solution of developing a mobile android application for making predictions on skin conditions in response to the Covid-19 pandemic situation to provide a small amount of aid to the medical sector was completed to 90% of the initial project scope. Most features, functionalities, requirements, and designs defined in the project scope and specification were met with little deviations.

- The application can take a live scan or image of the user's skin condition as intended.
- The application can upload an image of a skin condition from the user's android device gallery as intended.
- The application can predict a skin condition with 75-80% accuracy as intended.
- The application provides the user with detailed information on the predicted skin condition along with treatments for that condition after an image has been provided.
- The application creates a record of all information on every scan or upload the user has ever made and stores these records for access by the user on the application's "Scan History" interface/page.
- The application can read all contents displayed on the current interface out loud to the user via a text-to-speech feature, as intended.

Whilst the 3-month period given to complete the project was determined to be sufficient during the project's planning phase, the team ran into many factors which were unaccounted for during the development phase. Time taken to complete Assignments, Projects, Examinations, Classes and Study periods for other courses were not considered to be a threat/factor when planning and developing the project task schedule (Gantt Chart). This resulted in a great deal of time being withdrawn from the 3-month period required for project implementation, preventing the team from completing two (2) major features of the application promised in the scope:

1. The team was not able to implement the bounding boxes that were meant to locate, encompass, and highlight the affected area of skin found in the supplied scan/uploaded image. (maybe expand this)
2. The lack of time also did not allow for the model to be accurately trained to differentiate between what a skin condition is, and what a skin condition is not. For e.g if a user were to supply an image of a random object other than a skin ailment, the application's model would still treat the image of that object as a skin condition, and wrongly perform and produce a prediction on that object (which should not be the case).

What have we learnt?

Have alternative implementation solutions prepared beforehand - Whilst the project development team did not run into any catastrophic errors which resulted in the migration to different implementation methods or approaches (thankfully), the team did still experience minor setbacks where very small changes to the implementation plan were necessary. We now know and understand the importance of having 2 or 3 additional implementation approaches to fall back on, should we run into system failures during project implementation for future projects.

How to properly divide and conquer - Because of the size of the project, the project team had to resort to delegating & assigning roles and certain parts of the project's development to specific individuals on the project's development team. Because of this, the project was completed and fully functional all within due time, further ascertaining the need for delegation and separation of roles to team members.

Do not save documentation for last - The team did not plan to do any final documentation during the implementation of the project as we were initially confident enough that we would have sufficient time upon implementation completion to finalize the project's final report. It was only until we began final project documentation write up when we realized the importance of documenting each process or phase of the project as we went along to ensure efficient time management. Had it not been for the compulsory milestone documents and weekly reports required by the university, the project team may not have been able to produce a final project document of decent quality (such as this very document).

How to appropriately utilize Java Unit Testing - Whilst the team had already been sufficiently proficient in the Java programming language, it was not until this project's development did we understand the need and significance of Java Unit Tests. We learnt that Java Unit Tests are very important to establish from an early stage of development to ensure all code produces desired results pertaining to the project's requirements and does not stray from the project scope's promised functionality.

Summary of Feasibility

The research on convolutional neural networks in skin disease diagnosis is purely theoretical and has not been applied in a proper diagnostic scenario. However, the research points out that neural networks will be a big step in automating the time-consuming manual process involved in traditional manual skin disease diagnostics methods (ALEnezi, 2019). Though the app only works for 7 image classes right now, there is potential to create an app that correctly diagnoses all skin diseases given more data for the model to train on. Curame's implementation to classify 7 diseases show that it is a feasible app in skin diagnostics.

Future Work

The first step towards future implementation would be to firstly finalize development of the bounding boxes feature left undeveloped during the project's current development stage (maybe expand). The next initial step would be to improve the application's overall performance by continuing training on the skin disease prediction model over a lengthened period of time to increase the overall precision of the model. The level of accuracy which we would aim for in future versions of the application would be within the range of 90-99%. Should the application's model achieve such a level of accuracy, the next step would be to have the application be tested by professional dermatologists to determine its significance in medical diagnosis. Once the application has been approved by dermatologists, a centralized model from which all dermatologists can utilize and train on simultaneously will be implemented and deployed publicly. The final step would be to alter the project's scope and publicly announce its ability to diagnose skin diseases as a 2nd alternative to dermatologists.

Bibliography

The Agile System Development Life Cycle (SDLC). [Online]. Available: http://www.ambysoft.com/essays/agileLifecycle.html. [Accessed: 17-Apr-2021].

“Android Fade In / Out Animations with Examples,” Tutlane. [Online]. Available: https://www.tutlane.com/tutorial/android/android-fade-in-out-animations-with-examples. [Accessed: 17-Apr-2021].

“Android quickstart : TensorFlow Lite,” TensorFlow. [Online]. Available: https://www.tensorflow.org/lite/guide/android. [Accessed: 17-Apr-2021].

“Augmentor¶,” Augmentor. [Online]. Available: https://augmentor.readthedocs.io/en/master/. [Accessed: 17-Apr-2021].

“Documentation | Firebase,” Google. [Online]. Available: https://firebase.google.com/docs/. [Accessed: 17-Apr-2021].

“Image classification : TensorFlow Core,” TensorFlow. [Online]. Available: https://www.tensorflow.org/tutorials/images/classification. [Accessed: 17-Apr-2021].

“Image classification : TensorFlow Lite,” TensorFlow. [Online]. Available: https://www.tensorflow.org/lite/examples/image_classification/overview. [Accessed: 17-Apr-2021].

“Java Documentation - Get Started,” Oracle Help Center, 12-Mar-2021. [Online]. Available: https://docs.oracle.com/en/java/index.html. [Accessed: 17-Apr-2021].

Kumar, Gaurav, and Pradeep Kumar Bhatia. "Impact of agile methodology on software development process." International Journal of Computer Technology and Electronics Engineering (IJCTEE) 2, no. 4 (2012): 46-50.

ALEnezi, Nawal Soliman ALKolifi, “A Method Of Skin Disease Detection Using Image Processing And Machine Learning,” Procedia Computer Science, vol. 163, pp. 85–92, 2019.

Picasso. [Online]. Available: https://square.github.io/picasso/. [Accessed: 17-Apr-2021].

R. Madachy, COCOMO II - Constructive Cost Model. [Online]. Available: http://softwarecost.org/tools/COCOMO/. [Accessed: 17-Apr-2021].

“Recognize Flowers with TensorFlow Lite on Android | Google Codelabs,” Google. [Online]. Available: https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android. [Accessed: 05-Mar-2021].

Links

GitHub

https://github.com/Aaron-AB/Curame

Website

https://tlfoot.wixsite.com/website