Cleft Lip Aesthetics Tool

A cross-platform app that allows the success of a cleft palate surgery to be determined through symmetry

Mohammed Farbas Miah

BSc Computer Science

Submission Date: 2nd May 2017

Supervisor: Dr Harry Strange

Abstract

This project is about determining the success of cleft lip and palate surgeries. Using a mobile app, paediatric plastic surgeons should be able to evaluate the aesthetic outcome of the surgery by determining how symmetrical the lips are. The user should be able to draw around the lip region of the target image and then receive a symmetry score, determining the successfulness of the surgery. This would replace the previous subjective method of having a panel of people determine success. Having the app be multi-platform would be ideal to target more users. There should also be offline functionality which means the app will be synced to an online server to allow data to be transferred as required.

The app was developed with Cordova to allow multiplatform functionality to be implemented quicker. The user interface was developed with focus on ease of access. Once all the pages were set up, the app was linked to a server which allows downloading and uploading of drawings and images to the online database. The drawing feature was then implemented and pixel comparison was used to produce a symmetry score based off the drawing.

The goals of the project have successfully been reached which means a multi-platform app with offline features has been produced which allows users to create a drawing based off a patient's image and receive a symmetry score. These drawings can be uploaded to the cloud for future analytical work by researchers to allow for further improvements to the app. A user manual was produced to demonstrate the app's features in a simple manner. A system manual has also been produced regarding the setup of the project to assist in further development.

Contents

1.Introduction

1.1. Problem

Children born with a cleft lip and palate often have surgery at a young age to fix the problems associated with the condition. One of the improvements on the patient, post-surgery would be the aesthetic improvement to the lip region [1]. To determine the success of this procedure, a panel is used to determine how aesthetic the post-operation lip region is [2]. This is quite subjective and requires multiple people which is inefficient. This project aims to develop an app that can use symmetry to determine the success of a surgery to replace the current method being used. This would save a lot of time and provide faster and more accurate results once surgery has been completed.

1.2. Goals & aims

The successful completion of this project requires multiple key goals to be fulfilled which have been listed below.

Goal 1: A fully functional multi-platform application that determines the symmetry of the lips from a patient's image to determine the success of the surgery.

Goal 2: Full and clear documentation of the app for future development.

Goal 3: Concise user manual to demonstrate how to use the app.

Goal 4: An intuitive method of drawing around lip regions of a patient's image to produce a trace for determining success.

Goal 5: A method of determining the symmetry score of a drawing created by the user.

Goal 6: Integrated cloud connectivity with syncing and offline features for the app.

Goal 7: A final report detailing the planning, methodology and results of the project.

To accomplish each of these deliverables, the aims listed below need to be achieved first.

Aim 1: Understanding of Cordova and the programming languages required to develop an app that is functional on multiple platforms. The bulk of the design is created in HTML and

CSS with JavaScript being used for scripting. Individual platforms may also require their corresponding native programming language.

Aim 2: Knowledge of good coding practice to allow for code to be clearly presented and understood easily by other developers.

Aim 3: Understanding of the programming languages and tools required to set up a server and database system for app connectivity. SQL and PHP are used in this project to achieve this.

1.3. Project overview

The project was carried out in distinct sections, relating to the aims. The first section related to research and background knowledge to determine a clear set of requirements for the project and to determine the approach of development. This is documented in Chapters 2 and 3.

The next section, Chapter 4, relates to the development portion which was split further into sub-sections. Offline and Online portions of the app were developed individually and then linked together. The main drawing and symmetry score feature was then implemented. Finally, the multi-platform functionality was enabled by adding additional code to allow for Windows and Android compatibility. The app is not currently built for iOS devices due to the requirement of needing a Mac device to build, which was unavailable.

Lastly, evaluation and documentation was carried out. This involved testing the app as shown in Chapter 5 and logging issues and future ideas for development, in Chapter 6. Documentation produced includes a user and system manual for further guidance to future app users and developers, respectively which can be found in the Appendices.

1.4. Report overview

Chapter 1 – Introduction

Provides an outline of the problem being worked on and what the challenges and goals are.

Chapter 2 – Context

Details background information on the subject matter, going over previous work and research carried out for the project. The tools and software used to develop the app are explained here.

Chapter 3 – Requirements & Analysis

Provides a clear and structured set of requirements for the project. Expectations for the final product are established.

Chapter 4 – Design & Implementation

Shows how each piece of the system architecture has been implemented and how they are linked together. The reasoning behind specific design decisions are given here.

Chapter 5 – Results Evaluation

Details the testing approach with results and specifics for the app that has been developed.

Chapter 6 – Conclusions

This chapter gives a summary of what the project has achieved with a critical analysis. Ideas for future work on the project and final thoughts are given.

Chapter 7 – Bibliography

A list of sources that were made use of during the project which have not been linked directly in the text.

Chapter 8 – Appendices

The Appendices provide additional relevant information that have not been included in the other chapters. This includes manuals, documentation produced throughout the course of the project and code samples.

2.Context

2.1. Background information

To determine the success of a cleft lip and palate surgery, a panel of experts is required to visually judge whether the outcome has been successful. The aim of this project is to produce a means of replacing the current method with an app that can be used to judge the symmetry of a post-surgery image. The reasoning behind this is due to the flaws and inefficiencies in the current method. Currently, multiple people are required for each post-surgery panel which wastes time and resources. The judgement of success being left to people to decide is subjective as everyone has different preferences and ideal facial structures. In using a panel to determine the outcome, there would be a large delay between the time of the surgery and the judgement.

From creating an app to determine the success of a cleft lip and palate surgery, the above inefficiencies can be bypassed, therefore allowing for a quicker, more accurate and cheaper solution. In the past, attempts have been made to seek solutions to the given problem. Research on these are discussed below.

2.2. Related work

In the early stages of the project, similar work that have been carried out on the topic were reviewed. From doing so, a better understanding of the need for a solution was gained. The issues of the current method for determining success of a surgery were also better understood.

The first research paper being reviewed discusses the applicability of an Index that had been developed to determine the successfulness of a unilateral cleft lip and palate surgery for five year olds. [3] This Index is required to help determine success at a fast rate instead of waiting until the patient was older to determine the outcome. In doing so, surgeons could relate minor changes in the procedure to the outcome of a surgery. The paper only covers unilateral cleft lips which excludes the more severe case of bilateral cleft lips. However, in the case of unilateral cleft lips, the paper concluded that the Index was the fastest indicator of success of a surgery.

A computer program to determine lip symmetry post-surgery is discussed in another paper. SymNose, a computer program, compares the patient's lip regions to a control group who have never had a cleft lip. [4] The study only seeks to compare unilateral cleft lips and excludes bilateral cleft lips. Usage of the application was carried out by letting the user draw around the lip region of a patient. To determine symmetry, the percentage mismatch was calculated from overlaying the lip regions. The results demonstrated that the use of SymNose was a good tool as a quantitative assessment of success of surgery. In addition, this application presents an efficient and cheap solution to the problem at hand.

In a follow-up paper, the use of SymNose for comparison of bilateral cleft lip patients is examined. [5] Although the sample size was small at 15, the results supported the usage of SymNose for assessing the symmetry of bilateral cleft lip patients. Post-surgery, a significant level of asymmetry occurred for bilateral patients' lips whereas the nose region proved to be more symmetrical. This contradicts the original panel assessments and therefore suggests nose symmetry should play a part in determining the success of a surgery.

From reviewing the above three papers, an outline of the requirements for the app were obtained. It would be important to allow the determination of success for both unilateral and bilateral cleft lip patients. The results should be obtainable quickly to allow the analysis of results to occur faster, therefore letting surgeons know how minor variations in methodology affects a patient's facial symmetry. Use of nose regions provides an additional source of comparison to determine success, especially for bilateral cleft lip patients. The SymNose application is not publicly available which meant an examination of the program was not possible. In producing an open source app, future work to advance the project would be much simpler by allowing the problem at hand to be solved and improved upon faster.

2.3. Research done

Before beginning development for a multi-platform app, it was important to know what skills would be required beforehand. Therefore, preparatory learning of some subjects took place before the development phase. Once the decision was made to develop a multi-platform app, research was undertaken to determine which pieces of software would be used to carry this out. Additionally, during the development phase, further research took

place to determine the best solution to problems and to assist in solving them. The main resources used for research and understanding aspects of the project are listed below.

Stack overflow

Throughout the course of the project, multiple minor issues arose resulting in code not working in the expected manner or a fix to a certain problem could not be identified. These are problems that have occurred to other people in the past that have been solved already. Stack Overflow [6] is a Q&A style forum with topics based on computer programming that can help to answer a large amount of the problems that occurred during this project. Using Stack Overflow, many issues have been solved through searching for related problems.

Apache Cordova

Once it was decided to produce a multi-platform app with a single set of code, Apache Cordova [7] was chosen as the best solution. Cordova is a development framework for mobiles that helps to build multi-platform applications with a single set of code in HTML, CSS and JavaScript. It is simple to build and test apps on multiple platforms including Windows, iOS and Android. Documentation for both Cordova was constantly referred to during development.

Cordova plugins

When creating apps that are not native, they would presumably be limited by the capabilities of web apps. However, plugins in Cordova allow for the app to carry out native functionalities such as storage or camera usage. Specific native features were required for this project which would have to be implemented using plugins. npmJS [8] was repeatedly used to search for plugins to use for the app.

Programming tutorial sites

To code the app, HTML, CSS and JavaScript would be used for the front-end side. These languages had been partially learnt prior to commencing the project so there was not a need to learn these from scratch. Nonetheless, multiple aspects needed to be brushed up on or referenced which was done by using W3Schools [9]. W3Schools is a web development learning site which provides tutorials, references and examples for HTML, CSS and

JavaScript. For the back-end, PHP and MySQL was to be used which are also taught on W3Schools.

Being new to development with Cordova, tutorials for example apps were viewed to gain a better understanding of how aspects of the technology worked. PhoneGapPro [10] is a tutorial site that demonstrates how functionalities of Cordova can be implemented. From viewing the multiple tutorial apps available on the site, the applications of specific parts of Cordova were understood. These tutorials included usage of key plugins such as device storage and the application of PHP with MySQL databases.

2.4. Tools & software

Various software, library code, frameworks and other tools were used in the development of the app. Each of these are listed below with the reason for selection and what it was used for outlined.

GitHub

As a means of version control for source code, GitHub [11] was used. GitHub allows all previous uploads of code to be viewable from multiple computers. This also worked as a means of backing up the content produced over the course of the project. A private repository was created for free using a student account. This software was chosen over alternatives due to previous experience using GitHub, resulting in familiarity with the features.

Cordova

Cordova has multiple features that were made use of during development of the app. The key feature of this framework is to convert web page code into an app without having to rely on native, platform-specific APIs. To create an initial code template for the Cordova app, the Cordova command-line interface (CLI) [12] was used. This included all the relevant files and plugins to set up an app using HTML, CSS and JavaScript. The Cordova CLI allowed for various functionalities to be used which include the building of the app file, such as an APK for Android. It also allowed for quick testing of the app by building and running it directly onto a mobile device.

To implement native functionalities of a platform, plugins were used. A specific plugin was implemented to allow for SQLite databases [13] to be created and edited which would otherwise have not been possible to do. A file transfer plugin [14] was used to allow for files to be downloaded onto a device. To display in-app notifications to a user in Windows, a dialogs plugin was used [15]. This was required due to Windows apps not supporting the default alert function in JavaScript.

Android SDK

The Android Software Development Kit (SDK) [16] contains a comprehensive set of tools required for development on the Android platform. The SDK is a requirement to allow for APK packages to be created. When an Android app is built using Cordova, the Android SDK is required.

Windows 10 SDK

The Windows Software Development Kit for Windows 10 [17] which comes integrated with Visual Studio 2015 was also used. This SDK is a requirement when creating any Windows 10 app. This SDK is required when building a Windows 10 app with Cordova. Windows 10 is unique in that an AppX package can target all device families in the Windows 10 line-up.

Font Awesome

Font Awesome [18] is a font and icon toolkit which provides over 600 different scalable vector icons. It is the second most popular third-party Font Script with a 20% market share. This toolkit was used to provide visual icons for specific commands in the app, such as a drawing icon in front of the create a drawing button.

Paper.js

Paper.js [19] is a vector graphics scripting framework which makes use of the HTML Canvas. It provides a range of functionalities to create and edit drawings, making use of bezier curves and control points. This framework was used in the development of the app to create the drawing feature which would allow the user to draw the lip regions of a patient so that a symmetry score can be generated.

Snap.js

Snap.js [20] is a simple library for creating a navigation drawer with JavaScript. It provides multiple features to improve user experience such as the ability to slide the drawer with a flick of the finger. This library was used as part of the core user interface of the app, providing the navigation drawer which is used in most pages of the app.

jQuery

jQuery [21] is JavaScript library which makes multiple aspects of development simpler with an API that is easy to use. Some major features are its event handling, animation and Ajax functionalities. jQuery is the most widely used JavaScript library and is therefore supported on many platforms. For this app, it was used in multiple features with the main aspects being its Ajax and event handling implementation.

000webhost

000webhost [22] is a free web hosting service which allows for PHP and MySQL to be utilised with almost no restrictions or ads. The service provided is highly reliable and widely used with a 99% uptime guarantee. This service was chosen for use in handling the server-side aspects of the app. SQL tables were created which contain relevant data for each patient which the app users can download. Another table was also used to allow for users to upload and download drawings that have been created for a patient.

3.Requirements & Analysis

3.1. Detailed problem statement

To deliver a high quality and robust product, the problem detailed in Chapter 1 was expanded upon. This was done through multiple methods which included identifying the user's needs, comparison with previous research and discussions with the project supervisor.

Firstly, a series of meetings were held with the project supervisor to determine what the full problem was and what was required to solve it. Initially unstructured aims were produced to help solve the problem. The app would have to be multi-platform to allow for users to be able to access it from a variety of settings, such as from a desktop in a hospital. It was determined that a server would be required to download images of patients to the local device. From then on the user would be able to create a drawing and generate a prediction score.

Emphasis was placed on certain aspects of the app to ensure that it suited its target users. These aspects were obtained after carrying out user stories from the two main users of the app, a clinician and a researcher. The user stories can be seen in Table 3.1 below.

User	**Viewpoint**
Clinician	As a clinician, I want to be able to use the app without having to spend a long time learning how to use it so that I can save time.
Clinician	As a clinician, I want to obtain symmetry scores on a patient's image quickly without having to go through large amounts of options so that I can use the app without much hassle.
Researcher	As a researcher, I want to easily view relevant input and output data for images so that I can analyse the results without much hassle.

Table 3.1 – User stories

From a clinician's point of view, they desire an app that is as simple as possible to use with a minimalist style of content. This meant that the app to be developed would need to have a very simple and consistent user interface with only the required functionalities.

For the researcher, they want to be able to access all drawings of a particular patient and their corresponding symmetry scores. The app is intended to be capable of running offline which means there needs to be a system in which users can upload their drawings to the server.

All the above in this section represent the starting point of the project, from which a more detailed set of requirements were derived.

3.2. Structured list of requirements

Based off the detailed problem statement, a structured set of requirements were produced. These were produced in the MoSCoW style which details the priority levels of each individual requirement through Must have, Should have, Could have and Would have. As the project progressed in the early stages, these were refined slightly which resulted in the final 16 requirements, categorised by its activity, as shown in Table 3.2 below.

ID	Details	Type	Activity	Priority
RQ1	The app shall let the user sync their data for each image with the cloud so new drawings and symmetry scores are uploaded.	Functional	Cloud	Must have
RQ2	The app shall be connected to a cloud based server in which images are stored.	Functional	Cloud	Must have
RQ3	The app shall have an intuitive method of allowing users to draw around the lip regions of images stored locally.	Functional	Drawings	Must have
RQ4	The app shall determine symmetry scores from the user's drawings and display them to determine success of surgery.	Functional	Processing	Must have
RQ5	The app shall allow users to view corresponding lip drawings and symmetry scores for each image on the cloud.	Functional	User interface	Must have

RQ6	The app shall allow users to view lip drawings and symmetry scores for each image stored locally.	Functional	User interface	Must have
RQ7	The app shall display a list of all images stored locally with options to draw lip regions and view previous drawings and symmetry scores.	Functional	User interface	Must have
RQ8	The app shall require minimal training to use in a clinical setting.	Non-functional	Accessibility	Must have
RQ9	The app shall have a clean interface based on Android guidelines.	Non-functional	User interface	Must have
RQ10	The app shall be capable of running on multiple platforms.	Functional	Accessibility	Should have
RQ11	The app shall allow users to download images locally from the cloud.	Functional	Cloud	Should have
RQ12	The app shall be built in a manner so that other people can build upon it easily due to good coding practices being followed.	Non-functional	Development	Should have
RQ13	The app shall process the user's drawings to make them smoother.	Functional	Drawings	Could have
RQ14	The app shall allow users to delete locally stored patient images and drawings.	Functional	User interface	Could have
RQ15	The app shall have control points for drawing around the lip regions with further precision.	Functional	Drawings	Could have
RQ16	The app shall have offline functionality.	Functional	Accessibility	Could have

Table 3.2 – List of MoSCoW style requirements

3.3. Use cases

Use cases were made to show the interactions between an actor and the system to achieve a goal. In doing so, an overview of what the system does and does not do was determined.

Some of the issues that arise from each step an actor takes to achieve a goal were also understood here. The use case titles can be seen in Table 3.3 below with the full set of use cases appearing in the Use cases section of the Appendices.

Use case title	**Alternative flow**
DownloadImage	DownloadImage:Error
CloudImage	CloudImage:Error
DownloadDrawing	DownloadDrawing:Error
CloudDrawing	CloudDrawing:Error
LocalImage	N/A
DeleteImage	N/A
SyncImage	SyncImage:Error
CreateDrawing	CreateDrawing:Redo
LocalDrawing	N/A
DeleteDrawing	N/A
SyncDrawing	SyncDrawing:Error
CloudDrawing:Error	CloudDrawing:Error

Table 3.3 – Use case titles

3.4. Requirements analysis

The above sections in Chapter 3 were used to analyse in greater detail what is needed when developing. Therefore, the user stories were used to set up a general layout of the app. The features of the app should be minimalist to allow all the available features to be easily understood. All content should be easily accessible so that the user can navigate through the content efficiently. It was decided to use a navigation drawer style of pages for the app so that major sections of the app could be quickly accessed. The standard Android interface style was to be used so that users could already be familiar with how the buttons and content layout work. Page consistency was a crucial element when content was similar to different pages. For example, the local patient list and cloud patient list would be the same in that they display the list of patients with a dropdown menu of options for each. Similarly, the image viewing pages would be the same too, allowing users to be familiar with the page layouts quickly.

Having the app be multi-platform is an important aspect in the availability of the app. This led to the use of Apache Cordova tools for development which supports a large range of platforms. Both Android and Windows 10 devices were targeted with the app to allow for usage on mobiles, tablets and desktops. Due to iOS and OS X apps requiring a Mac to build, these two platforms were not targeted owing to not having access to such a device. However, these two platforms can easily be added in the future because of the simple manner in which Cordova allows apps to be built in different platforms. All dependencies and plugins to be used are capable of running on Windows, Android and Apple devices.

3.5. App structure

Using the structured requirements and use cases, all of the initial pages needed for the app were decided. An activity diagram was created to visualise the links between each page and the functionality provided by each of them. The activity diagram can be seen in Figure 3.4 below.

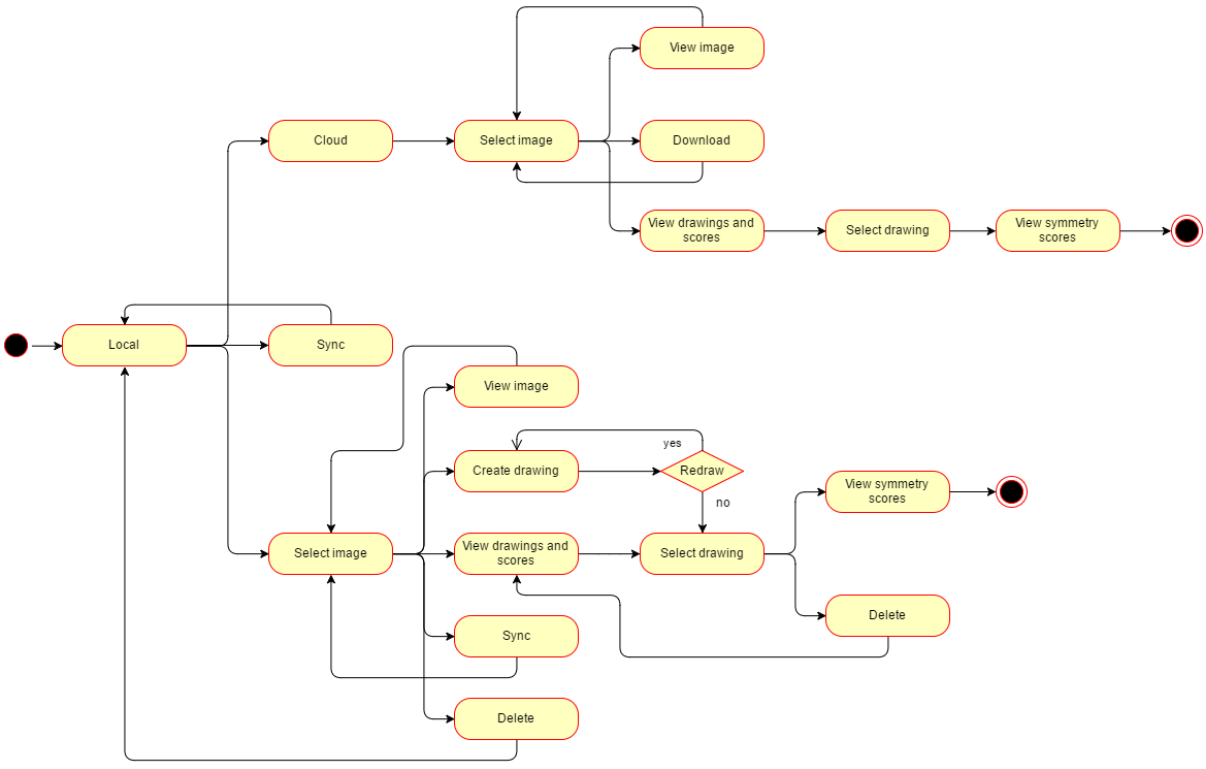


Figure 3.4 – Activity diagram

With the intent of having the pages be as simple as possible, the local and cloud patient lists were of the same design and layout. The drawings & scores page on both the local and cloud portions also utilised the same design, with the main difference being that instead of

having a list of patients, it was a list of drawings. All four of these pages have a toggle button to display a list of options for each list item. Likewise, there are a total of four image viewing pages with an option to view a patient image and a patient's drawing for both local and cloud versions. These pages are to be identical in design with a simple layout and a back button on the top left. The drawing creation page needs to have a great deal of free space to ensure that the user can precisely draw over their patient's image. Therefore, this page would only have a small banner at the top and bottom for navigation options.

3.6. Databases

Having cloud connectivity with a place to download patient images from and where drawings can be stored is an essential aspect of the app. This means that a server would need to be set up with a database to store these items. The images of patients are anonymised as they only display a small portion of the face which includes the nose and lip regions. Due to this, further detail is required when downloading an image because simply viewing it is not enough to verify they are the correct patient. Additional information which includes the patient's name and date of upload must therefore be connected to each patient image. Having a database table for all of the patients would be the best method in which to do this.

For the uploading and viewing of drawings, a separate table can be used with the uploaded drawing being connected to the symmetry score and its patient image the drawing was based off. The structure of the two database tables, shown as schemas are in Figure 3.5.

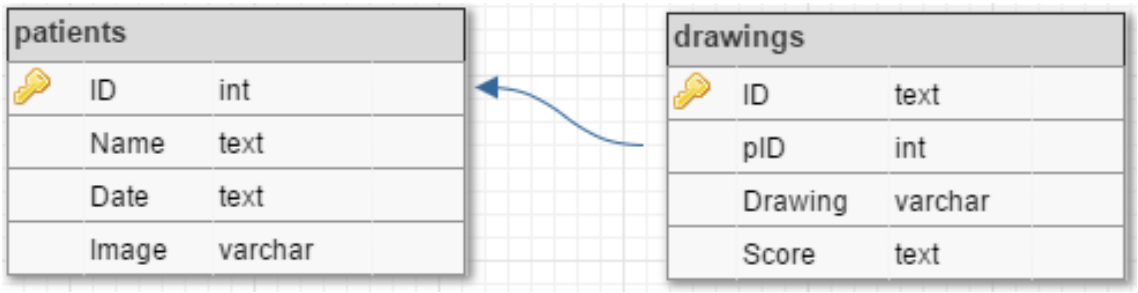


Table 3.5 – Database schema

The primary key for each table is the ID. When drawings are uploaded, the pID is entered as the ID of the patient. The image and drawing sections are used to store the images for the patient and drawing tables respectively. These are in the form of base64 which are simply images in text form.

4.Design & Implementation

4.1. Application architecture

The application has two distinct sections to it, the local app and the server-side database. The two pieces are connected through the downloading and uploading of content. The connectivity between all of the content can be seen in Figure 4.1.

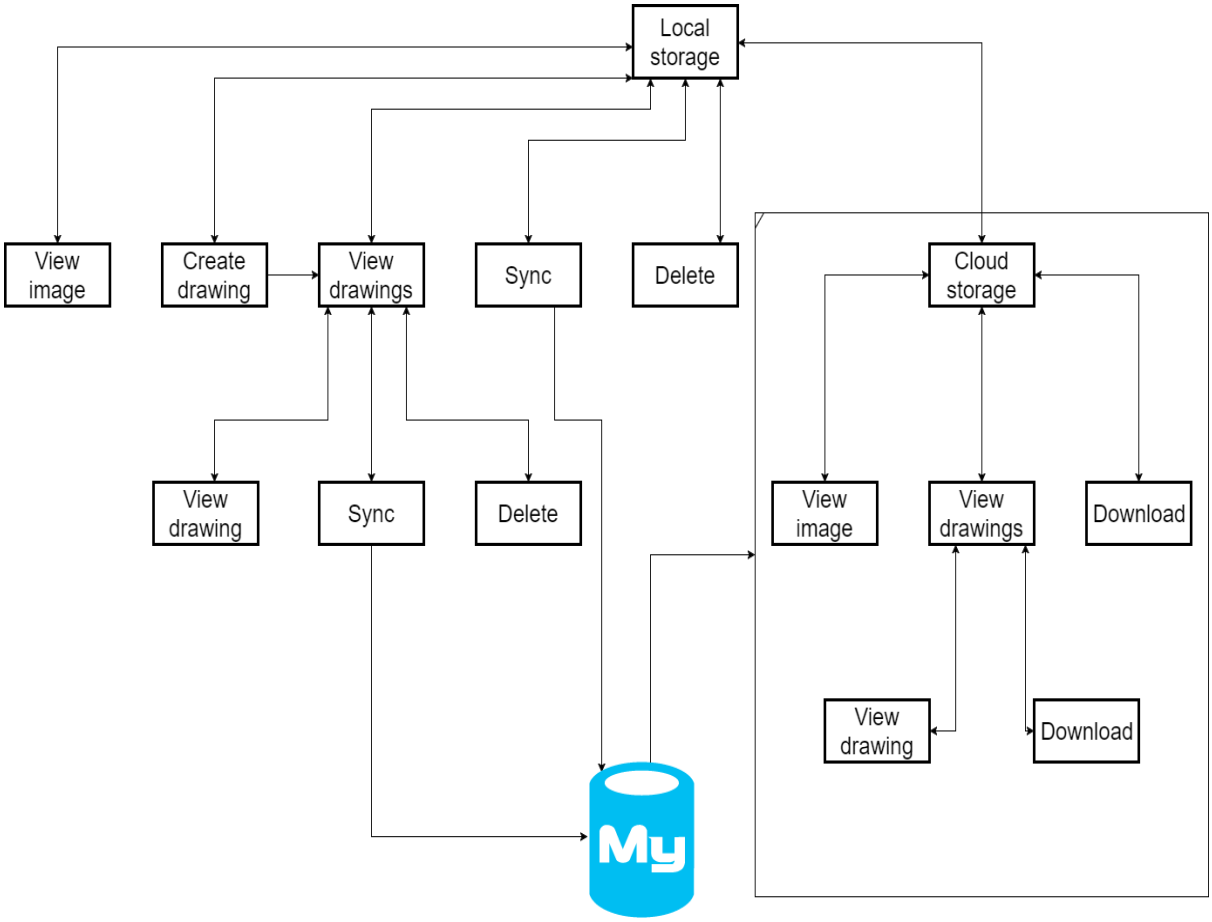


Table 4.1 – Connectivity between app pages and server-side database

The starting page of the app is the local patient list. This page has options for downloaded patients which let the user view the enlarged image, sync created drawings, delete the patient, create a drawing and to view previous drawings. If the enlarged image is viewed, the user can go back once satisfied. Syncing for a patient results in all drawings created for that patient being uploaded to the cloud database. Deleting a patient removes the patient and all corresponding drawings from the local device. Creating a drawing lets the user produce a trace over the lip regions of the patient to generate a symmetry score. Viewing

the patient's drawings and scores lists all of the locally stored drawings for the chosen patient.

In the drawing and scores page, drawings are listed in the same manner as the local patient list. The options for each drawing let the user view the enlarged drawing, view the patient's image, sync the drawing and delete the drawing. Syncing and deleting on this page results in only the individual drawings being impacted.

Through the navigation drawer, the user can navigate to the cloud patient list. All patients from the cloud database are listed here, letting the user download selected images. The user can also view a cloud patient's drawings that have been uploaded previously and download them if desired.

4.2. User interface

The aim of the user interface design was to provide a familiar style that users would have previous experience in using. This was done by using the standard Android navigation drawer on the left side of the screen with the title banner at the top of the page. Patient and drawing list items were displayed with clear separators between each item, having a minimalist detail layout. A tab icon was implemented for each list item, signalling to the user that clicking on it displays the full list of options.

All pages of the app have a consistent layout with a distinct theme to allow users to get accustomed to the app's features faster. The theme consists of a light and dark blue, with backgrounds of pages being a light grey colour. An app icon and banner were created for the app, displayed in Figure 4.2. The icon is displayed in the app and also as the launching icon for opening the app.

Figure 4.2 – App icon and banner

In keeping with the theme of the app, Font Awesome icons were also used as a visual aid for each button, in blue. These icons are vectors which allows the icons to stay clear, even as the screen size increases.

4.3. Drawings

To allow the user to create a trace of the patient's lip regions, a drawing feature was implemented. This was done with the Paper.js framework which provides a range of suitable functionalities.

The HTML canvas is made use of to allow the drawing to take place. The canvas is set to a size depending on the device's width and height, in a square shape which is the same shape of the patient images. This ensures the user doesn't need to scroll to view all of the content. A margin and border are laid around the canvas to display the boundaries of the drawing region.

Once the page has loaded, Paper.js is used to set the background of the canvas as the patient's image, known as a raster. The patient image is obtained from the previous page in which the user selected which patient to create a drawing for, through the *sessionStorage* property. This allows content to be stored in a temporary variable, accessible in JavaScript, for as long as the app is open.

Two different tools have been set up in Paper.js to create drawings, with the first tool being active by default. This first tool is used to create the drawing in a translucent, blue colour with a fill. Blue was chosen due to the distinguishability over all skin colours and it was made translucent to allow for the user to still see the patient's image underneath, allowing for greater precision. Once the user finishes the drawing, the drawing created is automatically closed by connecting the first point to the last point. Control points are added to each point the user drew over and then simplified so that there is a minimal amount of control points whilst still allowing for a suitable level of adjustment.

The second tool, which has not been implemented, has been set out in the code to allow for a user to manually adjust their drawing's control points to provide greater precision.

Once the user is satisfied with their drawing, they click on a button to generate a symmetry score. This is done through a function which removes the raster from the canvas and then sends the remaining canvas data for analysis.

4.4. Symmetry scores

Once the user has chosen to generate the symmetry score for a drawing, the remaining canvas data, after the raster has been removed, is obtained. The drawing is examined to check if it has an even width of pixels, and if not, it is adjusted to be even.

A 2D array is created with the dimensions being equal to the size of the drawing. The drawing is then inspected, pixel by pixel, to read the RGB values. The blue value is the only important value so each pixel is checked to see if the blue colour is present as this was the colour used for the drawing. If the blue colour is not present, then it means the pixel is just the background of the drawing. Values of 1 and 0 are inserted into each pixel's corresponding position in the 2D array with 1 meaning blue is present and 0 meaning it isn't. This was achieved through the use of nested for loops, shown in Figure 4.3.

```
for (var i = 0; i < canvas.height; i++) {
      for (var j = 0; j < canvas.width; j++) {
            if (pixelData[blue] != 0) {
                  initial[i][j] = 1;
            } else initial[i][j] = 0;
            blue += 4;
      }
}
```

Figure 4.3 – Nested *for* loops iterating through pixel values

Three more 2D arrays are created, named *left*, *right* and *overlap*. The initial 2D array created is split in half with the left side going into the new *left* array and the right side going into the *right* array. Once this has been done, the *right* array has to be reflected over the y-axis so that it would have a degree of overlap with the *left* array.

Each array value is then compared in the *left* and *right* arrays to determine the degree of overlap. This is done by summing the value of each 2D array value in the *left* and *right* arrays and placing the sum in the *overlap* array. As a result of this, every pixel that has an overlap would have a value of 2, no overlap would have a value of 1 and 0 would be the background in the *overlap* array. To determine the symmetry score, the percentage overlap is calculated

by dividing the total number of 1s and 2s by the number of 2s and then multiplying by 100. The procedure is demonstrated in further detail in Figure 4.4 by showing sample array values.

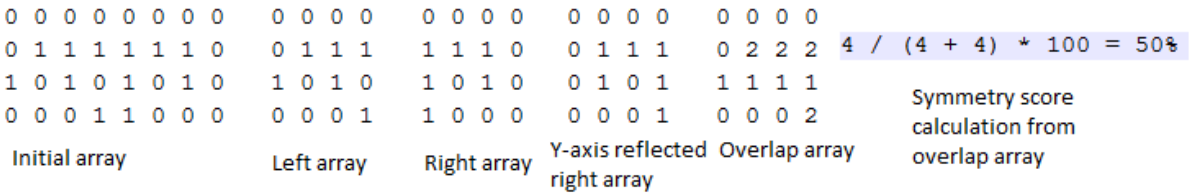


Figure 4.4 – Example array values demonstrating steps to obtain a symmetry score

4.5. List data

There are four different lists of data displayed in the app. These are all displayed in the same format to aid in the ease of use and to allow for familiarity. The default starting page of the app is for the local patients list which would be empty initially. Through the navigation drawer, the user can select the cloud patient list to download patients locally. The content being displayed is in a minimalist style so that only necessary information is displayed with options being hidden underneath a tab button. For patients, this means data that identifies the patient is displayed, the name, date of upload and a thumbnail of the patient's face.

Each list page displays relevant options in the details tab, depending on the content. For example, the cloud patient list and cloud drawing list would have a download button whereas their local counterparts would have a sync and delete feature instead. Figure 4.5 shows an expanded tab in the cloud patient list page.

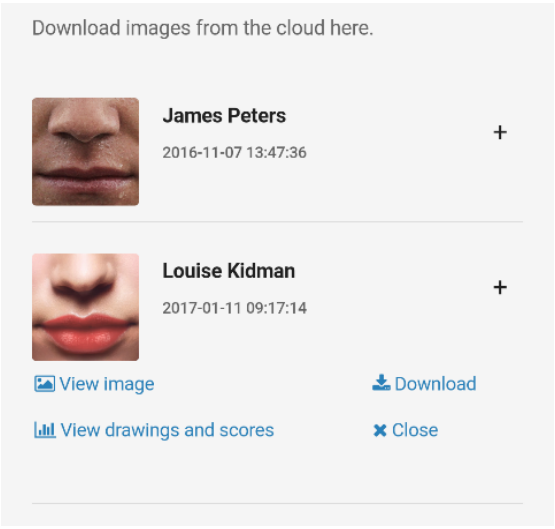


Figure 4.5 – An expanded tab for the cloud patient list

4.6. Local storage

The local storage consists of an SQLite database of two tables, set up with the *cordova-sqlite-storage* plugin for Cordova. All downloaded images are stored in a table called *images* with the unique ID, name, date of upload and base64 form of the image being stored in each column. Once a drawing for a specific patient is created, the patient's ID, symmetry score, drawing in base64 and the text of *No* are stored in a table called *drawings*. The value of *No* is placed in a column which says whether or not the drawing has been synced with the cloud. Once a drawing is synced, the value is changed to *Yes* and if a drawing is downloaded, it is automatically set to *Yes*.

Each page in the app that makes use of the SQLite database requires the database to first be opened before doing anything else. The starting page of the app, with *main.js*, contains all of the required setup of the database. The database is first opened and if it doesn't exist, it is created. After this the two tables are set up, if they do not exist. Commands for SQLite are the same as regular SQL with the content placed inside relevant executing functions through the plugin.

4.7. Cloud storage

All patients and drawings that are uploaded are stored in a database on a server. The MySQL database is managed through PHPMyAdmin, a feature through 000webhost. Similar to the local storage, there are two tables in the database, *images* and *drawings*. Each of the columns for the tables are the same as their local counterparts.

The database can be accessed from the app using PHP code to connect to the database and to upload and retrieve content. All patients in the MySQL database can be viewed from the cloud patient list page in the app. As a result, different patients can be added to the cloud database without much hassle to app users.

4.8. Viewing images

There are four different image viewing pages in the app. These are for viewing local patient images, local drawing images and their cloud based counterparts. These pages all have the same layout with a simple display and a back button on the top of the page. The pages are set so that the image being displayed occupies the maximum amount of space on the page without requiring the user to scroll to view the whole content. In keeping with the app's theme, the colour scheme and margins of the page remain consistent. Figure 4.6 shows the design of how the page looks.

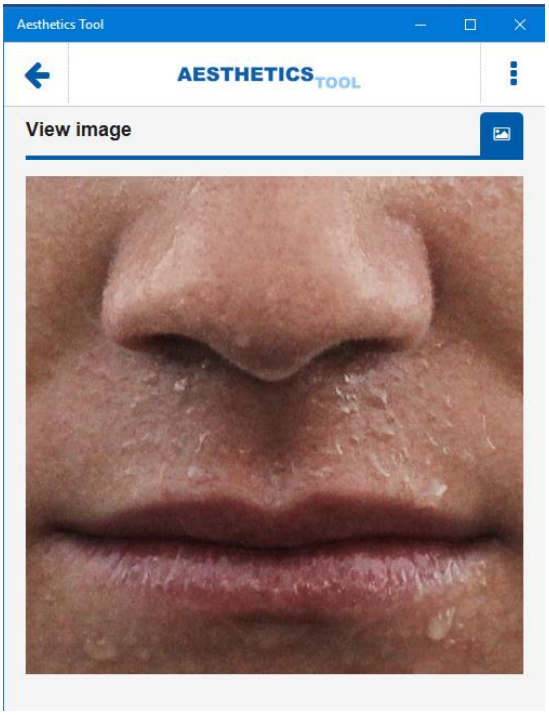


Figure 4.6 – Viewing a cloud patient's image

Like the drawings page, the page receives the image's data through the *sessionStorage* property in JavaScript.

4.9. Syncing data

Drawings created for patients can be uploaded to the cloud database so that researchers and other users can view them. The user has two ways to sync drawings, either all drawings created of a patient or individual drawings.

A specific drawing is synced by uploading each of the column values for that drawing from the SQLite database to the MySQL database. This is done by making use of jQuery AJAX with

HTTP POST. The PHP code responsible for uploading drawings is called and the data is then transferred. In the SQLite database, the fact that the drawing has been uploaded is logged so that the user cannot upload duplicate content. The user can also distinguish whether or not specific drawings have been uploaded.

The administrator can easily delete drawings from the MySQL database so that content can be regulated. Users can also delete any content stored locally so as to not clog up the page with older content.

4.10. Multi-platform functionality

All of the content of the app has been developed with the intention of allowing the app to run on Android, Windows 10, iOS and macOS. However, the app has been built and tested for on Android and Windows 10 platforms only. This is due to the requirement of needing Xcode when building for Apple platforms, a tool only available on Macs.

The ease of building the app with multi-platform functionalities was due in large part to the Cordova framework. Code was created primarily in HTML, CSS and JavaScript which was then converted into each platform's native source. The plugins used for the app have all been tested for being capable of running on all of the required platforms.

Targeted platforms and their details are listed in config.xml, a configuration file. The minimum Android version is listed as 4.0 and only runs on Windows 10 platforms for Windows. This was set based off the minimum requirements of content used for the app.

Access to the app's cloud database content in Android is not allowed by default which meant a whitelist had to be set. This was done by listing which URLs the app was allowed to view. For Windows 10 devices, there is no support for the JavaScript *alert* function. This would cause issues with providing notifications to the users regarding whether downloads or uploads had been completed successfully. To rectify this, a plugin, *cordova-plugin-dialogs* was made use of. This plugin provides very similar features to the native JavaScript function through the use of another function called *navigator.notification.alert*.

The app's content is designed in a manner so that everything is visible and clear to access, regardless of the device's screen size. Through JavaScript and CSS, the app's margins and paddings are dynamically adjusted depending on the size of the screen. Larger screens are set to have a larger margin whereas smaller devices require more room to display content so the margins are set smaller. If a user rotates or resizes their screen, the app detects the change and adjusts the content as appropriate. Figure 4.6 demonstrates the difference between a 5.5" and a 15.6" screen.

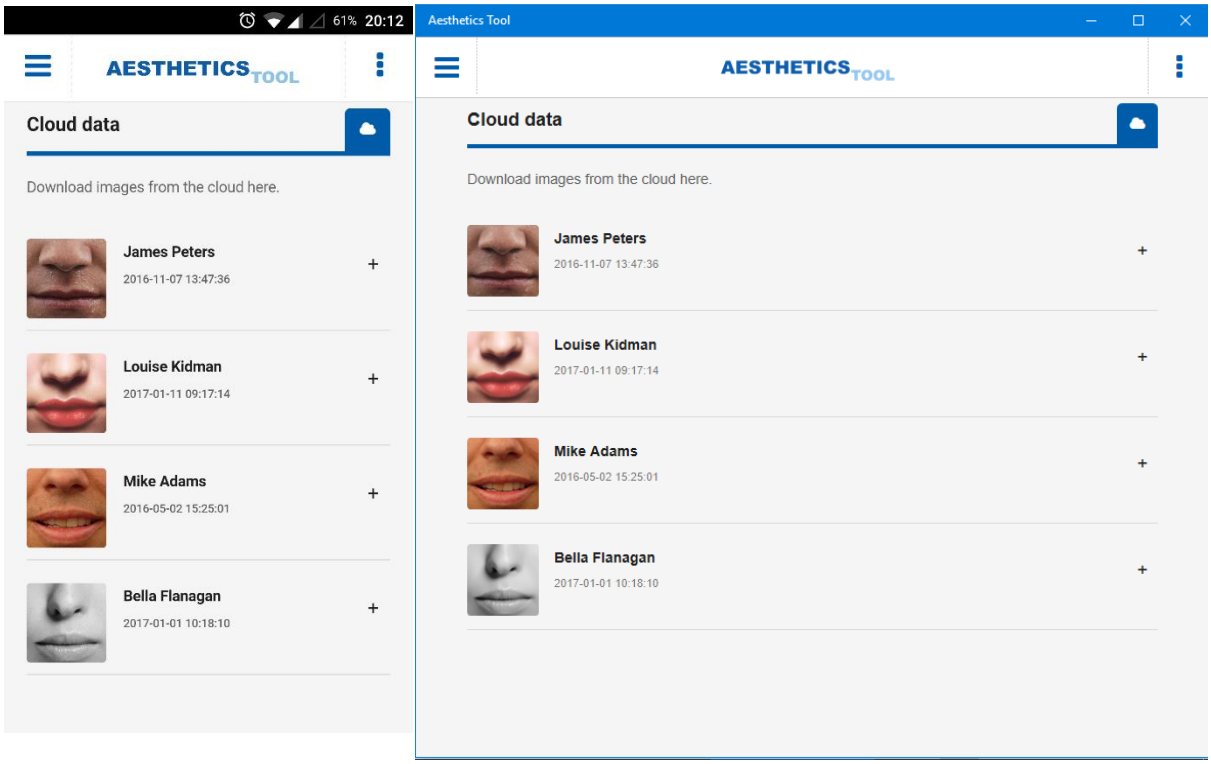


Figure 4.6 – Comparison of app layout over different devices

5.Testing

5.1. Overview

Multiple different testing strategies were undertaken to validate that the app worked as expected. The use of testing also allowed for verification that each of the MoSCoW style requirements were achieved and that the target user was able to successfully use the app. Cordova plugins that were made use of for this app have all been individually tested by their respective developers on several platforms and they have all passed their respective tests. The test logs can be viewed on each plugin's page as listed in the references.

The testing strategy comprised of both automated and manual variations on multiple different devices. Due to the limited availability of tools in automated testing for Cordova apps, an emphasis was placed on manual testing. As a result of this, bugs were detected and solved, redundancies were fixed and potential methods of breaking the app were accounted for. Each testing strategy is detailed in its own section below and the test results can be found in the Testing section of the Appendices.

5.2. Stress testing

Automated stress testing was conducted on each main page of the Android app through the use of Android's Monkey testing. This program generates a random sequence of click, swipes and gestures which can be repeated in the same sequence if required. The tests were conducted with 500 different events for each page, with the same events being repeated if any issues were found for a specific page.

One issue was indirectly discovered on the drawing page when the Done button was clicked without creating a drawing. This resulted in the symmetry score being null as there would be a divide by 0 error when carrying out the calculation. The issue was solved by preventing users from submitting a blank drawing. Although this problem wasn't highlighted by the stress testing results due to no errors occurring, the problem would not have been identified otherwise.

5.3. Symmetry score testing

The calculation for determining the symmetry score was tested by examining the results for multiple different input drawings on various patient images and devices. The symmetry score is a result of the percentage symmetry of the drawing image produced.

Whilst examining the results on different patient images, it was discovered some of the scores provided did not seem accurate. After examining the drawing, the reason was determined to be due to the drawing being off-centre. A fix for this was to make sure the patient images available for download were all centred so that the traces produced wouldn't be unevenly positioned. This issue is demonstrated in Figure 5.1 with similar drawings being compared.

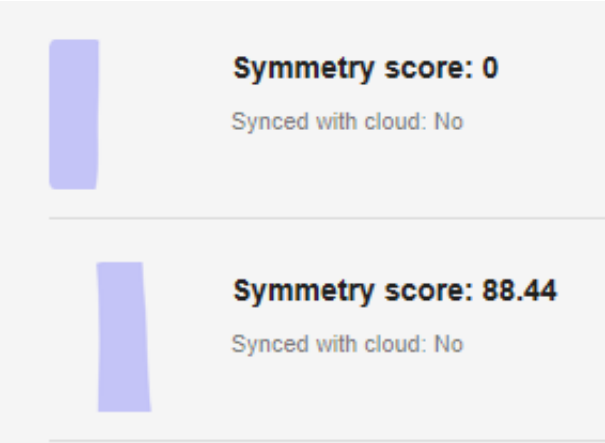


Figure 5.1 – Comparison of an off-centre drawing (top) and centred drawing (bottom)

Another issue found was regarding the screen's page width. In calculating the symmetry score, the image is converted to a binary 2D array and split in half to two separate arrays. If the user's device has a screen width of an odd value, an error would occur when trying to split the array, leading to no result being provided. The reason for having an odd number of pixels for the screen width was due to the drawing page adapting to the user's screen size. This issue was fixed by always ensuring the drawing region's width was always an even value.

5.4. Functional testing

Functional testing was carried out to determine whether or not each MoSCoW style requirement was achieved. A series of repeatable steps were carried out for each requirement, independently of other requirements. The tests for each requirement can be

seen in further detail in the Testing section of the Appendices. Steps taken for each requirement are given with any prerequisites listed. Any issues are noted in the provided column.

From carrying out these tests, it was deemed that all Must have and Should have requirements were achieved. One Could have requirement was not fulfilled, RQ15. This requirement needed the simplified control points of the user's drawing to be adjustable to provide greater precision.

5.5. Debugging & refinement

Throughout the development stage multiple features were tweaked, removed or replaced. This led to redundant code which were never called in multiple files. An example of this would be in the CSS files when the drawing feature was improved upon. Running the code on Google Chrome allowed for unused CSS to be automatically detected through the use of the Audits feature.

JavaScript source files were compared with each other to remove duplicate code by comparing the function names. Due to the similar page layouts throughout the app, multiple duplicates were found. For example, four files contained the same navigation drawer code. A new JavaScript file was created to store this code, removing a significant portion of redundant code. Not all cases of duplicate code were dealt with however, due to refinement causing difficult to understand code. For this reason, some files contain very similar code for specific functions, such as the download feature.

5.6. Device & compatibility testing

The app was developed for Android and Windows 10 with the expectation that all features would be functional on Android 4.0+ phones and tablets and Windows 10 phones, tablets and computers. To verify that this was the case, all of the use cases were manually tested against on five different devices. The devices are listed in Table 5.2 below.

Device	**Operating system**	**Screen size**	**Type**
Xiaomi Redmi Note 3	Android 5.1.1	5.5”	Physical
Android tablet	Android 4.4.4	7.0”	Emulated

HP Pavilion	Windows 10	15.6”	Physical
Windows phone	Windows 10	5.0”	Emulated
Windows tablet	Windows 10	8.0”	Emulated

Table 5.2 – Tested devices

The use cases carried out all worked as anticipated, so no issues were found. The user interface adapted to the screen size as expected which includes when the window was resized and rotated, as shown in Figure 5.3.

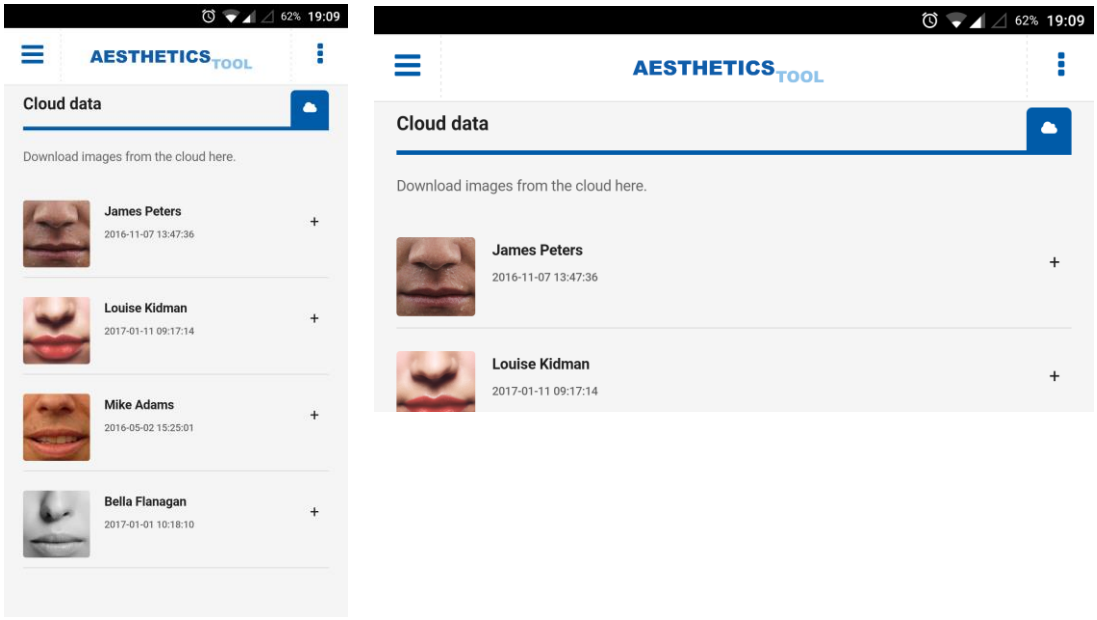


Figure 5.3 – App interface in portrait (left) and landscape (right)

All of the page content was visible for each device. However, the drawing feature was more difficult to use precisely on the smaller mobile screens. Therefore, a recommended screen size for the app should be at least 7.0”.

5.7. Acceptance testing

Acceptance testing was carried out to determine whether or not the app was suitable for fulfilling its objectives in the real world. Three volunteers were used to carry out all of the use cases on the app, from a laptop. Before they began, an explanation of the app and its purpose was given so that the participants knew what they were supposed to achieve. The volunteers all had experience using smartphones and in using mobile apps. Each participant

logged their experiences, noting any general comments, issues or ideas. They also answered Yes or No to whether or not they felt the app based non-functional requirements were met.

The use cases were all carried out successfully by each user, demonstrating the app runs as expected. Regarding the non-functional requirements, all participants agreed that it was easy and intuitive in learning how to use the app correctly and that the user interface was simple and had a familiar feel to it. Notable comments by each participant are shown in the Testing section for the Appendices.

With regards to the participant's comments, two issues were found which required fixing. The first of which was a problem with the drawing feature's fill feature. As the user drew around the lip region, it was difficult to see whether or not the drawing was accurate because of the opaque fill covering up the image. This issue was solved by making a translucent drawing fill so that the user could still see the patient's image underneath the drawing. The differences between before and after implementing the feature can be seen in Figure 5.4 below.

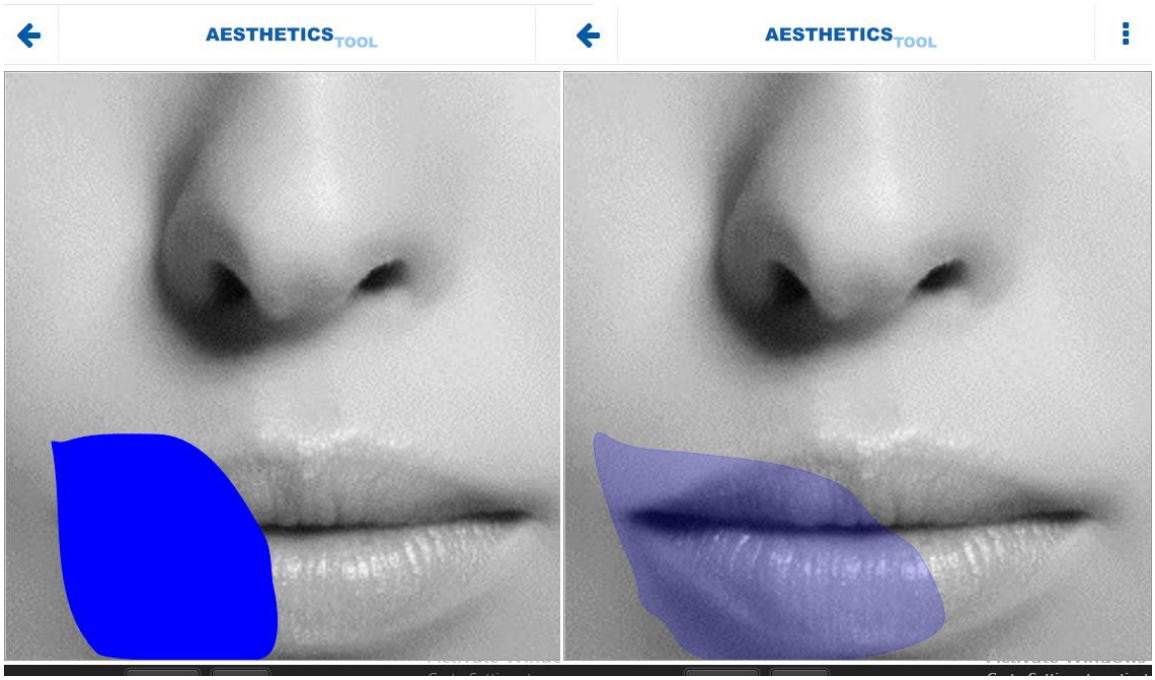


Table 5.4 – Before and after translucent fill

The second issue found was regarding the opening of the tabs for each patient listed in the app. Two participants experienced non-responsiveness when attempting to press the button to open the tab. After examining this issue, it was discovered that the actual icon

does not respond to touching and therefore does not open the tab to display the details. Instead, the surround area of the icon was being used to open the tabs. This was fixed by adjusting the code so that that the tabs would open when clicking on the icon or the surrounding region.

6.Conclusions

6.1. Summary of achievement

In this project, there were seven main goals to achieve in order to effectively deliver a working product, as detailed in Chapter 1. These were complimented with 16 MoSCoW style requirements which detailed specifics on what was required. All of these goals and 15 requirements were successfully achieved over the course of the project.

The first goal was to deliver a multi-platform app which contained all the required functionalities on each platform. This was successfully achieved through the use of Cordova to produce an Android and Windows 10 app. The second requirement was for full and clear documentation of the app for future development. All code produced was fully commented and clearly laid out with proper indentation to ensure easy readability and understanding. A system manual was also created to explain and demonstrate how to set up the project, assisting future developers. Similarly, a user manual was created to fulfil the third goal of having a means of demonstrating how to use the app and all its functionalities to its users.

The fourth and fifth goal require an intuitive method of creating a drawing and determining the corresponding symmetry score, respectively. Both of these were achieved, with the drawing feature being completed with a framework to produce a trace of the lip region. The symmetry score is then automatically calculated once the user accepts their drawing, displaying the score almost immediately. The sixth goal required offline functionalities and cloud connectivity with a syncing feature for drawings. This was completed through the use of an online server and database which users can download images and drawings from and also upload their own drawings to. Once a user downloads a patient's image, the app can be used offline to obtain symmetry scores. The final goal required a report to be produced detailing the planning, methodology and results of the app which can be seen in this report.

6.2. Critical evaluation

The aim of this project was to develop a means of determining the success of a cleft lip and palate surgery without the need for a panel decision. This was achieved through an app which allows users to trace over a patient's lip regions to obtain a symmetry score. All of the goals of the project were met with some being met to a higher standard than others.

The aim of having a multi-platform app was highly successful as all three major platforms were targeted and can be built for with a simple command, with the added potential of adding further platforms. This was achieved through using a single source code base written in HTML, JavaScript and CSS. As a result, the ease of adjusting features for the app can be carried over to all platforms by simply rebuilding the app. A major drawback of targeting multiple platforms was the difficulty in testing the app to a high standard. There were very few free tools available to provide automated testing for Cordova apps which meant a compromise was made by placing an emphasis on manual testing.

The server-side was created with the aim of allowing users to download from and upload to a database. The database was created with SQL and connectivity between the app and the database was achieved through PHP. This was a relatively simple task and the technology used fit their purpose of achieving the goal soundly.

The drawing feature, although working fine to obtain a symmetry score, is not as precise and detailed as initially aimed for. With the initial aim being to develop an Android only app, this would mean that the user would likely be creating the drawing on a small screen with a touchscreen interface. This is not an ideal method of using the app as it would be difficult to precisely draw the trace of the lip regions for a patient. After examining varying screen sizes, it was determined that on a touchscreen device, the user would ideally be using a pen to draw on a screen of at least 7 inches. The Windows desktop version of the app however, is the best platform for creating drawings to a high degree of precision. This is due to the usage of a mouse to draw, a much more accurate method.

The symmetry score was obtained through converting the user's drawing into a 2d array of 1s and 0s and then splitting them in half to see how much they overlap. This fits the goal well as the drawing created only contains a transparent background or blue pixels. Having the drawing pixels being an odd width were accounted for when splitting the array in half by ensuring the width was always an even value. This method also allowed for a very fast score calculation, leading to a result being provided to the user almost instantly.

The design and user interface of the app is a particular strong point with Android guidelines being followed to produce a familiar and clean looking layout. This meant it was simple to

navigate the app and clear what each feature or button did. An important aspect of the design was to ensure that it looked professional and to a high standard. This was achieved through keeping the design simple and consistent with a clear colour scheme and through having a custom icon and title banner.

The key implementation technology chosen was Cordova to allow for the multi-platform capabilities. This was chosen over creating a single platform Android app. Alternative multi-platform technologies were also available to use but Cordova was chosen as the ideal solution due to the larger range of capabilities the technology offered. There is a large user base with clear documentation and native API capabilities. For this reason, using Cordova was the correct decision and aided greatly in creating a successful app.

Native functionalities of each individual platform were required for local data storage. This was achieved through an SQLite plugin over alternative storage methods. This technology was chosen due to its compatibility with all of Android, Windows and iOS without much modification of code, a crucial requirement. Taking into account this requirement, some major storage methods were unfitting such as WebSQL and IndexedDB. A simple API, LocalStorage, was a viable choice but was unsuitable for when there is a large amount of data. There is a limit of around 5MB of data which would cause a problem if the user downloaded even a few images locally. In comparison, SQLite offered practically unlimited storage so it was the ideal candidate.

The drawing feature was implemented with Paper.js which is a library that provides a wide range of drawing editing features. This piece of technology was chosen after carrying out a comparison with multiple other drawing libraries which include jsDraw2D, SVG-edit, Method-Draw, Raphael.js and Fabric.js. Each of these technologies were evaluated to determine the ease of implementation of the drawing feature, their quality of implementation and for their additional features which could be of use in future development.

For the server-side, PHP was used due to the ease of connection with an SQL database. These are widely supported and commonly used for simple server-side connections. The quantity of PHP code required was very little and was a familiar language which meant it

would not make sense to look into unfamiliar alternatives when the current technology was suitable.

6.3. Future work

Although the current app works successfully and has fulfilled its goals, there are still some development ideas that can further enhance the usefulness of the app. These extended goals could potentially have been implemented if there was additional time for the project and could possibly be added in the future.

The app currently supports Android and Windows devices. Although these hold a large market share, Apple iOS and OS X platforms hold a significant portion of users. These platforms were not implemented due to the need for a Mac device to build Apple applications, which was not available over the duration of the project. Adding in these platforms would be very simple through Cordova due to the code and plugins used taking into account the potential usage of these platforms.

Currently, this project targets only the lip regions of the patient to determine a symmetry score. Even though the app allows the user to draw around nose regions with additional drawings, this feature has not been tested or examined to see if it works sufficiently to improve the accuracy of the symmetry score. In the future, development could include the nose and other aspects of the face to determine a symmetry score. This would provide a more reliable measurement for the success of a surgery due to the fact that the nose is also effected by a cleft lip and palate.

This app allows for a much faster method of determining the success of the surgery compared to the previous method. Nonetheless, further efficiencies could be added to make the user spend even less time using the app to obtain a symmetry score. Having the app automatically detect where the lip and nose regions are would mean the user doesn't have to spend time precisely drawing out their traces. Instead, the user could simply verify if the automatic detection was accurate and adjust it if required. Implementing this could potentially produce more accurate results as the drawings would be more precise.

Creating a website version of the app may allow it to be more accessible and provide a wider range of users. The app has already been optimised for browser usage so all content is

displayed correctly. There would need to be a logging in system so that users can be distinguished and it would have to be fully online so there would be no need to download anything locally. This could also allow for the uploading of private patient images which can only be viewed by specific users.

A cleft lip and palate is just one of many ways in which the aesthetics of a person's face are changed. Some others include burn victims, cosmetic plastic surgery patients or oral and maxillofacial surgery patients. Post-surgery, these patients also require a means to determine the outcome of their surgery. Future developments could extend the current code base to determine the differences in pre-and post-surgery patients. For example, a person with a burn on their left cheek could get a score based on the symmetry of the RGB pixel values for each cheek. This would have further reaching benefits as a significantly larger portion of people would be candidates for using the app.

6.4. Final thoughts

This is the second time I have worked on a major healthcare related project. I am very interested in how technology assists in medical procedures and enjoyed learning more about the process of treating a cleft lip and palate. Unlike the previous project in second year, this was individual which meant I had to be responsible for all aspects of the planning and development. The user interface and server-side were relatively smooth for me to implement but I felt the advanced drawing features and testing were something I was not strong at. This project helped me gain confidence in my own coding abilities to work on the more technical parts independently.

I have gained a great deal of appreciation for the use of frameworks and libraries for various tasks. Initially I felt I could simply create the navigation drawer without a code base. After comparing my own version with a library, I realised the difference in detail and how much time I could have saved. From then on, I have always looked to see if something has already been implemented before beginning coding. An example of this would be the drawing feature which makes use of Paper.js.

A major learning goal for me in this project was to improve my report writing skills for larger documents. Going into a non-tech career, this is something I feel will be particularly

important for me. I am pleased with having produced a structured and clearly laid out report for a project of this size, the largest document produced from my time at university.

Looking back through the years with each major development project, I have seen how much I have improved. A major improvement for me has been in noticing how important the planning stages are before development. The lessons I have learnt through experience in these projects are aspects of my university education that I am sure will assist me greatly in the future.

7.Bibliography

7.1. List of references

[1] repair, C. (2017). Cleft lip and palate repair: MedlinePlus Medical Encyclopaedia. Medlineplus.gov. Retrieved from https://medlineplus.gov/ency/article/002979.htm

[2] Sharma VP, e. (2017). Outcomes in facial aesthetics in cleft lip and palate surgery: a systematic review. - PubMed - NCBI. Ncbi.nlm.nih.gov. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/22591614

[3] Atack, Nikki E., et al. "A new index for assessing surgical outcome in unilateral cleft lip and palate subjects aged five: reproducibility and validity." *The Cleft palate-craniofacial journal* 34.3 (1997): 242-246.

[4] McKearney, Richard M., John V. Williams, and Nigel S. Mercer. "Quantitative computer-based assessment of lip symmetry following cleft lip repair." *The Cleft Palate-Craniofacial Journal* 50.2 (2013): 138-143.

[5] Russell, James HB, Harriet C. Kiddy, and Nigel S. Mercer. "The use of SymNose for quantitative assessment of lip symmetry following repair of complete bilateral cleft lip and palate." *Journal of Cranio-Maxillofacial Surgery* 42.5 (2014): 454-459.

[6] Stack Overflow. (2017). Stackoverflow.com. Retrieved from http://stackoverflow.com/

[7] Documentation - Apache Cordova. (2017). Cordova.apache.org. Retrieved from https://cordova.apache.org/docs/en/latest/

[8] cordova. (2017). npm. Retrieved from https://www.npmjs.com/package/cordova

[9] W3Schools Online Web Tutorials. (2017). W3schools.com. Retrieved from https://www.w3schools.com/

[10] Phonegap Tutorial for Beginners to Advanced | PhonegapPro. (2017). PhoneGapPro.com. Retrieved from https://phonegappro.com/phonegap-tutorial/

[11] Build software better, together. (2017). GitHub. Retrieved from https://github.com/

[12] Creating your first Cordova app - Apache Cordova. (2017). Cordova.apache.org. Retrieved from https://cordova.apache.org/docs/en/latest/guide/cli/#installing-the-cordova-cli

[13] cordova-sqlite-storage. (2017). npm. Retrieved from https://www.npmjs.com/package/cordova-sqlite-storage

[14] cordova-plugin-file-transfer. (2017). npm. Retrieved from https://www.npmjs.com/package/cordova-plugin-file-transfer

[15] cordova-plugin-dialogs. (2017). npm. Retrieved from https://www.npmjs.com/package/cordova-plugin-dialogs

[16] Notes, S. (2017). SDK Tools Release Notes | Android Studio. Developer.android.com. Retrieved from https://developer.android.com/studio/releases/sdk-tools.html

[17] Windows 10 SDK - Windows app development. (2017). Developer.microsoft.com. Retrieved https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk

[18] Gandy, D. (2017). Font Awesome, the iconic font and CSS toolkit. Fontawesome.io. Retrieved from http://fontawesome.io/

[19] Paper.js — About. (2017). Paperjs.org. Retrieved from http://paperjs.org/about/

[20] jakiestfu/Snap.js. (2017). GitHub. Retrieved from https://github.com/jakiestfu/Snap.js/

[21] jquery.org, j. (2017). jQuery. Jquery.com. Retrieved from https://jquery.com/

[22] Free Web Hosting with PHP, MySQL and cPanel, No Ads | 2017. (2017). Free Web Hosting. Retrieved from https://www.000webhost.com/

7.2. Bibliography

Monkey, U. (2017). UI/Application Exerciser Monkey | Android Studio. Developer.android.com. Retrieved from https://developer.android.com/studio/test/monkey.html

Genymotion – Fast & Easy Android Emulator. (2017). Genymotion – Fast & Easy Android Emulator. Retrieved from https://www.genymotion.com/

websql. (2017). npm. Retrieved from https://www.npmjs.com/package/websql

cordova-plugin-indexeddb2. (2017). npm. Retrieved from https://www.npmjs.com/package/cordova-plugin-indexeddb2

Javascript Graphics Library jsDraw2D draw line, circle, rectangle, polygon, curve, ellipse. (2017). Jsdraw2d.jsfiction.com. Retrieved from http://jsdraw2d.jsfiction.com/

SVG-Edit/svgedit. (2017). GitHub. Retrieved from https://github.com/SVG-Edit/svgedit

duopixel/Method-Draw. (2017). GitHub. Retrieved from https://github.com/duopixel/Method-Draw

Baranovskiy, D. (2017). Raphaël—JavaScript Library. Dmitrybaranovskiy.github.io. Retrieved from http://dmitrybaranovskiy.github.io/raphael/

Fabric.js Javascript Canvas Library. (2017). Fabricjs.com. Retrieved from http://fabricjs.com/

Flowchart Maker & Online Diagram Software. (2017). Draw.io. Retrieved 22 April 2017, from https://www.draw.io/

8.Appendices

8.1. Use Cases

The full list of use cases for the app can be seen below.

USE CASE	DownloadImage
ID	UC1
BRIEF DESCRIPTION	A member of hospital staff wants to download post-surgery images to determine success of surgery.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients on the cloud. 2) The user selects an image to download locally. 3) The image is saved locally on the device.
POST CONDITIONS	Image stored on device.
ALTERNATIVE FLOWS	DownloadImage:Error

USE CASE	DownloadImage:Error
ID	UC1.1

BRIEF DESCRIPTION	A member of hospital staff unsuccessfully attempts to download an image.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients on the cloud. 2) The user selects an image to download locally. 3) The system displays a notification to check the internet connection.
POST CONDITIONS	None
ALTERNATIVE FLOWS	None

USE CASE	CloudImage
ID	UC2
BRIEF DESCRIPTION	A member of hospital staff wants to view a post-surgery image of a patient.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher

PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients on the cloud. 2) The user selects an image to view. 3) Selected image is shown to the user.
POST CONDITIONS	Image is displayed on screen.
ALTERNATIVE FLOWS	CloudImage:Error

USE CASE	CloudImage:Error
ID	UC2.1
BRIEF DESCRIPTION	A member of hospital staff unsuccessfully attempts to view a patient's image.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients on the cloud. 2) The user selects an image to view. 3) The system displays a notification to check the internet connection.

POST CONDITIONS	None
ALTERNATIVE FLOWS	None

USE CASE	DownloadDrawing
ID	UC3
BRIEF DESCRIPTION	A member of hospital staff wants to download a patient's lip drawing and corresponding symmetry score to store locally.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients on the cloud. 2) The user selects a drawing to download locally. 3) The drawing is saved locally on the device.
POST CONDITIONS	Drawing stored on device.
ALTERNATIVE FLOWS	DownloadDrawing:Error

USE CASE	DownloadDrawing:Error
ID	UC3.1
BRIEF DESCRIPTION	A member of hospital staff unsuccessfully attempts to download a drawing.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients on the cloud. 2) The user selects a drawing to download locally. 3) The system displays a notification to check the internet connection.
POST CONDITIONS	None
ALTERNATIVE FLOWS	None

USE CASE	CloudDrawing
ID	UC4
BRIEF DESCRIPTION	A member of hospital staff wants to view a drawing of a patient's lip region.

PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients on the cloud. 2) The user selects a drawing to view. 3) Selected drawing is shown to the user.
POST CONDITIONS	Drawing is displayed on screen.
ALTERNATIVE FLOWS	CloudDrawing:Error

USE CASE	CloudDrawing:Error
ID	UC4.1
BRIEF DESCRIPTION	A member of hospital staff unsuccessfully attempts to view a patient's lip drawing.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must be connected to the internet.

MAIN FLOW	1) The user selects the option to view patients on the cloud. 2) The user selects a drawing to view. 3) The system displays a notification to check the internet connection.
POST CONDITIONS	None
ALTERNATIVE FLOWS	None

USE CASE	LocalImage
ID	UC5
BRIEF DESCRIPTION	A member of hospital staff wants to view a post-surgery image of a patient.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must have already downloaded the image.
MAIN FLOW	1) The user selects the option to view patients stored locally. 2) The user selects an image to view. 3) Selected image is shown to the user.
POST CONDITIONS	Image is displayed on screen.

ALTERNATIVE FLOWS	None

USE CASE	DeleteImage
ID	UC6
BRIEF DESCRIPTION	A member of hospital staff wants to delete a patient stored locally.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must have already downloaded the image.
MAIN FLOW	1) The user selects the option to view patients stored locally. 2) The user selects an image to delete. 3) Selected image is deleted.
POST CONDITIONS	Updated list of locally stored patients are displayed.
ALTERNATIVE FLOWS	None

USE CASE	SyncImage
ID	UC7

BRIEF DESCRIPTION	A member of hospital staff wants to upload an image's drawings and symmetry scores to the cloud.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients stored locally. 2) The user selects an image to sync. 3) The system displays a notification that sync has completed.
POST CONDITIONS	Drawings and symmetry scores are uploaded to the cloud.
ALTERNATIVE FLOWS	SyncImage:Error

USE CASE	SyncImage:Error
ID	UC7.1
BRIEF DESCRIPTION	A member of hospital staff unsuccessfully uploads an image's drawings and symmetry scores to the cloud.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher

PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients stored locally. 2) The user selects an image to sync. 3) The system displays a notification to check the internet connection.
POST CONDITIONS	None
ALTERNATIVE FLOWS	None

USE CASE	CreateDrawing
ID	UC8
BRIEF DESCRIPTION	A member of hospital staff wants to determine the success of a surgery from an image.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must have saved an image locally.
MAIN FLOW	1) The user selects a locally stored image. 2) The user selects the option to create a drawing of lip regions. 3) The user draws around the lip regions. 4) The user selects the option to generate symmetry scores.

	5) The system displays the symmetry score.
POST CONDITIONS	Symmetry score is generated and displayed.
ALTERNATIVE FLOWS	CreateDrawing:Redo

USE CASE	CreateDrawing:Redo
ID	UC8.1
BRIEF DESCRIPTION	A member of hospital staff drew around the lip regions incorrectly.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user wants to redraw around the lip regions.
MAIN FLOW	1) The user draws around the lip regions incorrectly. 2) The user selects the option to redraw around the lip regions. 3) The use case is repeated until the user is satisfied with the drawing and selects the option to generate symmetry scores. 4) The system displays the symmetry score.
POST CONDITIONS	Symmetry score is generated and displayed.

ALTERNATIVE FLOWS	CreateDrawing:Redo

USE CASE	LocalDrawing
ID	UC9
BRIEF DESCRIPTION	A member of hospital staff wants to view a drawing of a patient's lip region.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The image must have drawings associated with it.
MAIN FLOW	1) The user selects the option to view patients stored locally. 2) The user selects the option to view drawings of an image. 3) The user selects a drawing to view. 4) Selected drawing is shown to the user.
POST CONDITIONS	Drawing is displayed on screen.
ALTERNATIVE FLOWS	None

USE CASE	DeleteDrawing

ID	UC10
BRIEF DESCRIPTION	A member of hospital staff wants to delete a drawing stored locally.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must have drawings stored locally.
MAIN FLOW	1) The user selects the option to view patients stored locally. 2) The user selects the option to view drawings of an image. 3) The user selects a drawing to delete. 4) Selected drawing is deleted.
POST CONDITIONS	Updated list of locally stored drawings are displayed.
ALTERNATIVE FLOWS	None

USE CASE	SyncDrawing
ID	UC11
BRIEF DESCRIPTION	A member of hospital staff wants to upload a single drawing of an image's lip regions and symmetry scores to the cloud.

PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher
PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients stored locally. 2) The user selects the option to view drawings of an image. 3) The user selects a drawing to sync. 4) The system displays a notification that sync has completed.
POST CONDITIONS	Drawings and symmetry score is uploaded to the cloud.
ALTERNATIVE FLOWS	SyncDrawing:Error

USE CASE	SyncDrawing:Error
ID	UC11.1
BRIEF DESCRIPTION	A member of hospital staff unsuccessfully uploads an image's drawing and symmetry score to the cloud.
PRIMARY ACTORS	Clinician
SECONDARY ACTORS	Researcher

PRECONDITIONS	The user must be connected to the internet.
MAIN FLOW	1) The user selects the option to view patients stored locally. 2) The user selects the option to view drawings of an image. 3) The user selects a drawing to sync. 4) The system displays a notification to check the internet connection.
POST CONDITIONS	None
ALTERNATIVE FLOWS	None

System Manual

Cleft Lip Aesthetics Tool

This system manual has been created for the purpose of allowing for future development to take place on this project without much confusion. Outlined below are the details required for setting up and running the project.

Initial requirements

The computer being used for building the app must meet the following minimum system requirements:

- Microsoft® Windows® 7/8/10 (32- or 64-bit)
- 1.6 GHz or faster processor
- 3 GB RAM minimum, 8 GB RAM recommended
- 9 GB of available hard disk space (Includes Android and Windows SDK)
- 1280 x 768 minimum screen resolution

1. Downloading dependencies

Cordova runs on Node.js which needs to be downloaded and installed first from: https://nodejs.org/en/

Once Node.js has been installed, Cordova can be installed from the command prompt by typing npm install -g cordova.

Full details can be seen on: https://cordova.apache.org/#getstarted

Additional platform dependencies need to be installed, in this case Android and Windows.

To set up the SDK environment for Android, the Java Development Kit 7 or later must first be installed from: http://www.oracle.com/technetwork/java/javase/downloads/index.html

The JAVA_HOME environment variable needs to be set depending on the Java installation path.

Android Studio must next be downloaded and installed, which comes with the Android SDK, for full Cordova functionality from:
https://developer.android.com/studio/install.html?pkg=studio

The highest SDK version should be installed. The latest build-tools should also be installed.

The Android_HOME environment variable needs to be set depending on the Android SDK installation path.

Full details can be seen on:
https://cordova.apache.org/docs/en/latest/guide/platforms/android/index.html

The Windows SDK is available for download with Visual Studio 2015 or newer from:
https://www.visualstudio.com/downloads/

The feature to be selected for installation with Visual Studio is Universal Windows App Development Tools.

Full details can be seen on:
https://cordova.apache.org/docs/en/latest/guide/platforms/win8/index.html

2. Retrieving source code

All of the source code for the project is available in a private repository at:
https://github.com/Farbas1/Cleft-Lip-Individual-Project

A GitHub username and password have been supplied elsewhere to provide access to this repository.

The code for the server-side can be found in the Server side folder. The Cordova app code can be found in the Aesthetics Tool App folder.

3. Setting up project

The main source code for the app is stored in the www folder. The files can be edited in any text editor such as Notepad++. Alternatively, the project can be opened up in the Visual Studio IDE by going to the platforms and then windows folder and clicking on CordovaApp.Windows10.jsproj. This will automatically load all relevant files, available for editing, debugging and running.

4. Building & running app

To build and run the app, open a command prompt in the Aesthetics Tool App directory. For Android, type cordova run android to run the app on a physical device. Alternatively, cordova emulate android can be used to run the app on an emulator. To build the app and not run the app, type cordova build android.

For Windows, type cordova run windows --arch=x64 into the command prompt to build and run the app. The –-arch value must be stated and can be either x64, x32 or ARM. Alternatively, the app be built and not run with cordova build windows --arch=x64.

If the project is being used in the Visual Studio IDE, simply select the CPU type at the top of the screen and then select to run the app on either Local Machine, Simulator or Device. This is demonstrated in Figure 8.1 below.

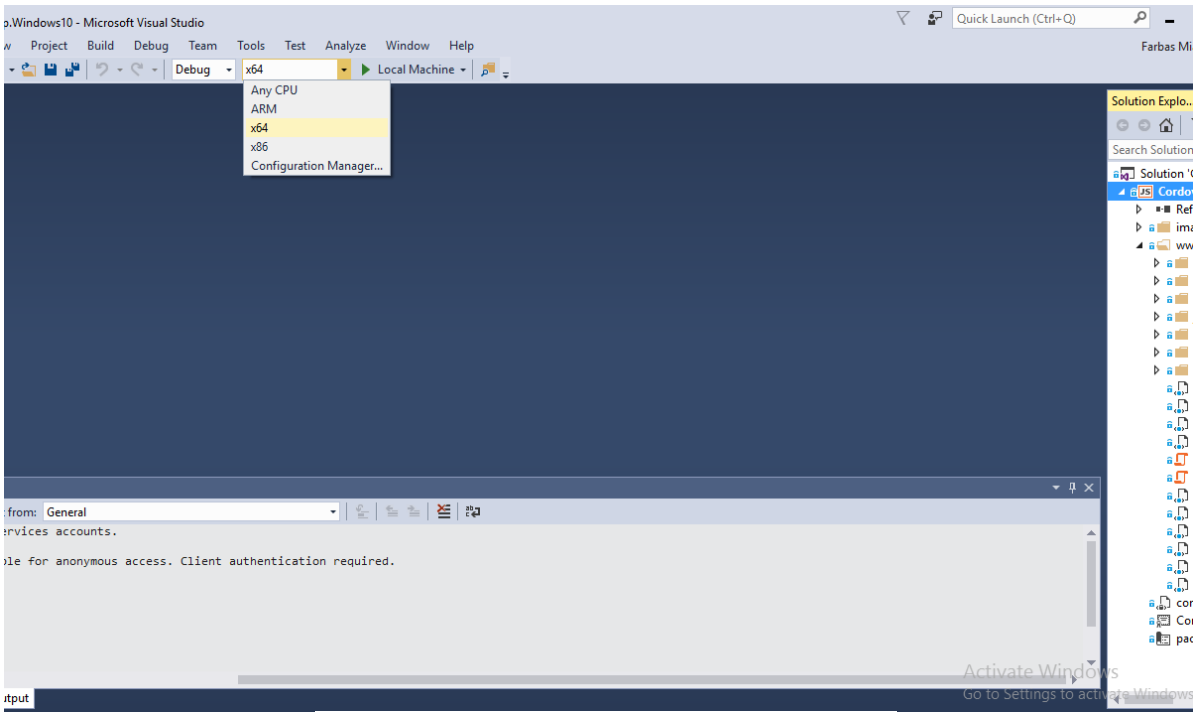


Figure 8.1 – Running in Visual Studio

5. Accessing server-side

The server has been set up on a free host and is accessible from:
https://www.000webhost.com/

Login details have been provided elsewhere to provide access to the server.

Once logged in, the PHP code can be accessed by clicking on the File manager tab at the top of the page.

The database is accessible by clicking on the Manage database tab at the top of the page and then opening the database with PhpMyAdmin.

6. Adding new patients

Once the database has been opened with PhpMyAdmin, two tables are available. The drawings table simply contains data related to the drawings uploaded by users. The images table is where patients are located.

To add a new patient, click on the images table and then the Insert tab at the top of the page. There are four columns of data to input, the first of which is the ID and can be ignored as it increments automatically. The following three columns to input data for are name, date and image. The name and date can be inserted in any format, preferably with a standard format. The image column accepts only base64 data which is an image in text format.

Once the data has been filled in, click Go to complete adding a patient into the database. New patients will be visible to users immediately on all platforms.

8.3. User manual

User Manual

Cleft Lip Aesthetics Tool

This user manual has been created for the purpose of detailing to app users how to make use of all of the features on the app. Installation details are given for both Android and Windows platforms. Outlined below are the steps required for installing and using the app.

Initial requirements

The computer being used for running the app must meet the following minimum system requirements:

- Microsoft® Windows® 10 (32- or 64-bit)
- 100 MB of available hard disk space
- 1280 x 768 minimum screen resolution

The mobile being used for running the app must meet the following minimum system requirements:

- Android 4.0 (API level 14) or higher
- 100 MB of available hard disk space
- 1280 x 720 minimum screen resolution

7. Downloading and installing

To download the app on Windows, download the APPX file from here: http://bit.ly/2ox9AM0

Once downloaded, go to the file location and right click to open the Properties. At the bottom of the tab click on the Unblock box and then press OK as shown in Figure 8.2.

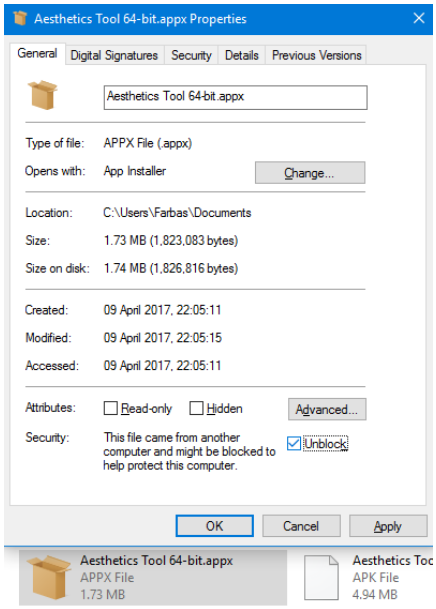


Figure 8.2 – Unblocking APPX file

Double-click on the APPX file to open and click Install. Once installed, click Launch to open the app. The app will now be accessible from the Start menu.

To download the app on Android, download the APK file from here: http://bit.ly/2oTUKl4

Before installing, make sure the device allows installation of apps from unknown sources. This option can be found in the Settings with the location varying between devices. Figure 8.3 demonstrates this option.

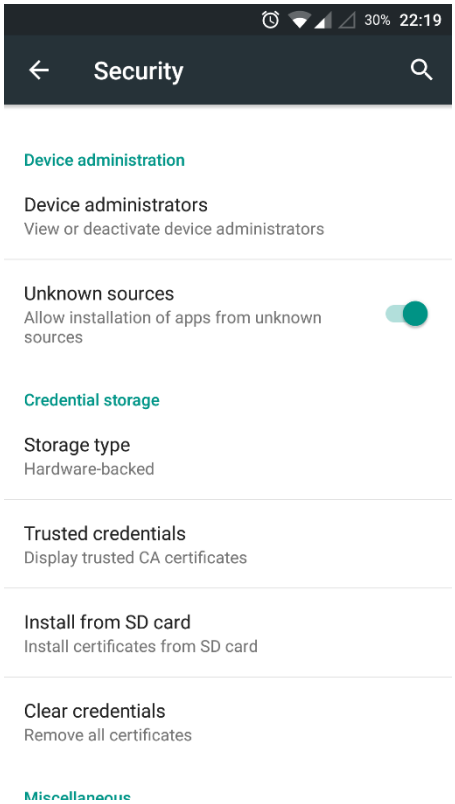


Figure 8.3 – Installing from unknown sources

Go to the APK download location, open it and then click INSTALL to begin installation. The app will now be accessible from the device's app list. Click on the app icon to open.

8. Download patient images & drawings

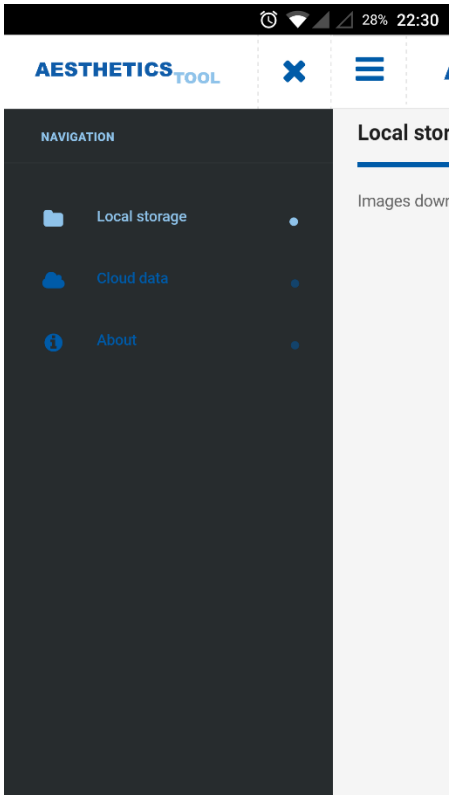


Figure 8.4 – Navigation drawer

To download a patient locally, open the navigation drawer with the toggle button on the top left. Click on Cloud data to load the patient list.

All of the available patients will then be listed. Click on the plus symbol for a specific patient to toggle options.

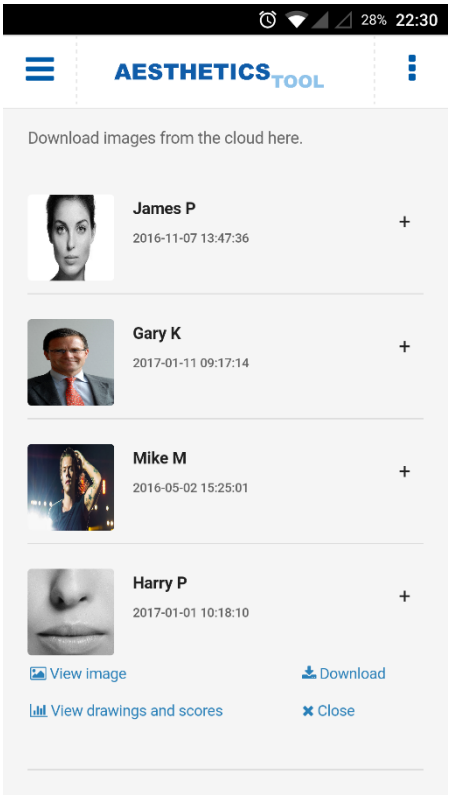
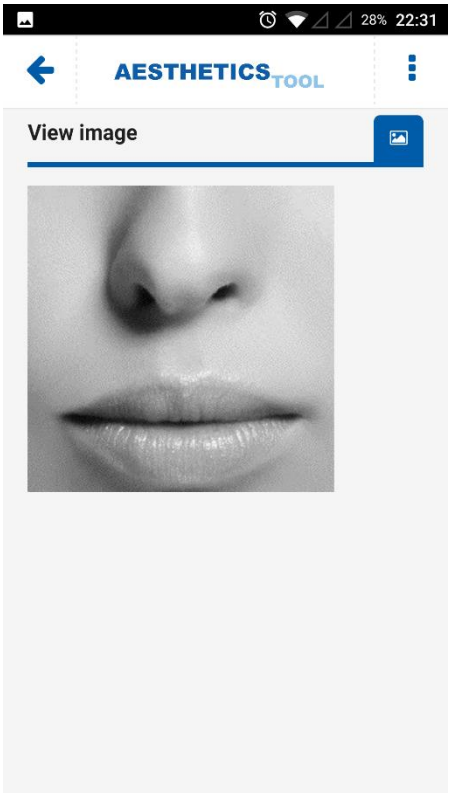


Figure 8.5 – Cloud patient list

To expand the patient's image, click on View image.



Press the back button on the top left to return to the patient list.

To download an image locally, click Download. You will receive a notification once the image has downloaded.

Figure 8.6 – Viewing a patient image

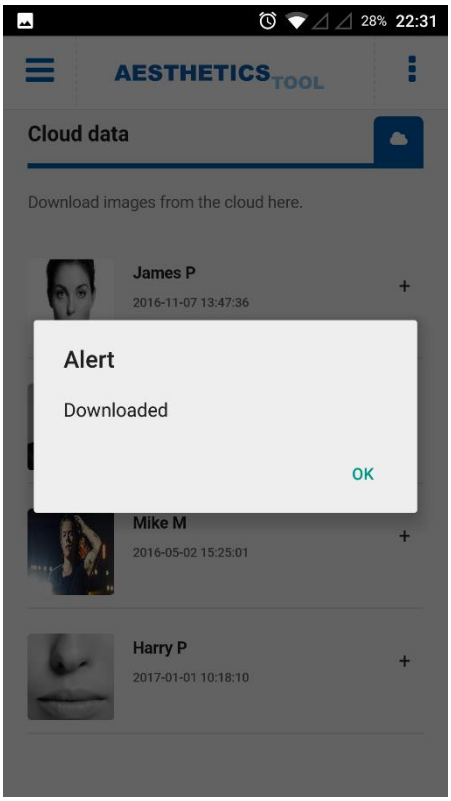


Figure 8.7 – Downloading an image

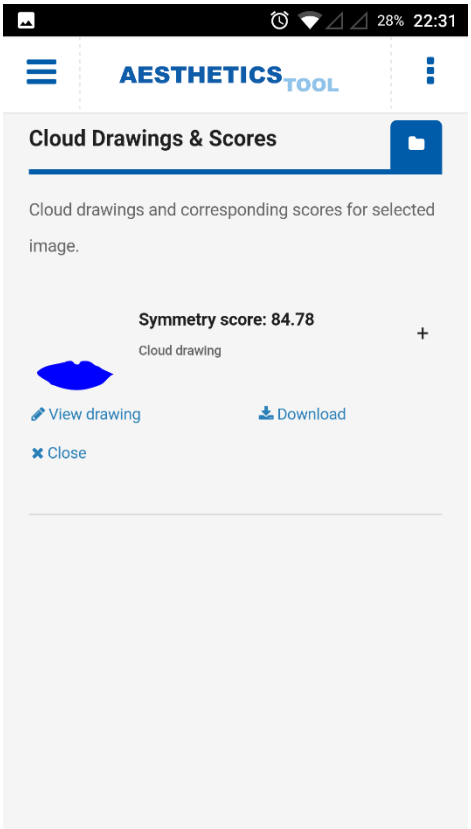


Figure 8.8 – Viewing cloud drawings and symmetry scores

To view drawings that have already been uploaded by others previously, click on View drawings and scores. Like the patient images, the drawings will be listed with options that can be toggled for each drawing. Just like the patients, to view the drawing click View drawing and to download the drawing locally click Download.

9. Creating a drawing

To create a drawing to generate a symmetry score, navigate to the Local storage from the navigation drawer. Patients downloaded locally will be shown here. Use the plus sign to toggle the options for the chosen patient and select Create drawing.

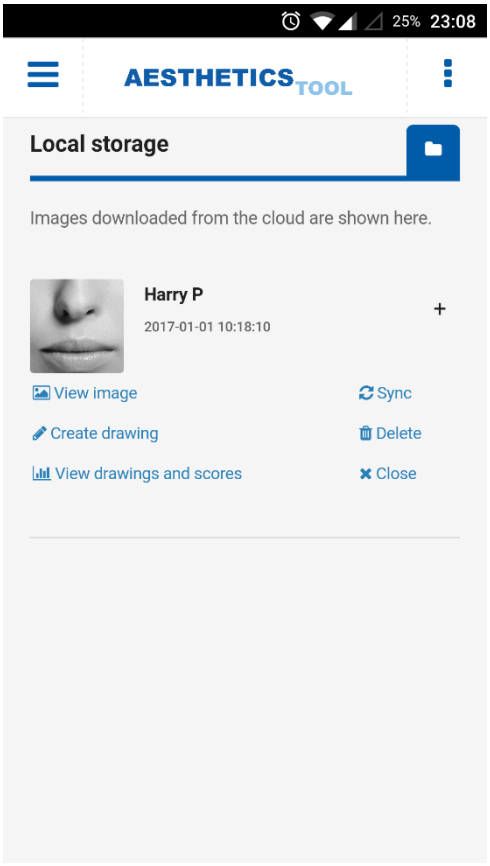


Figure 8.9 – Details for local patients

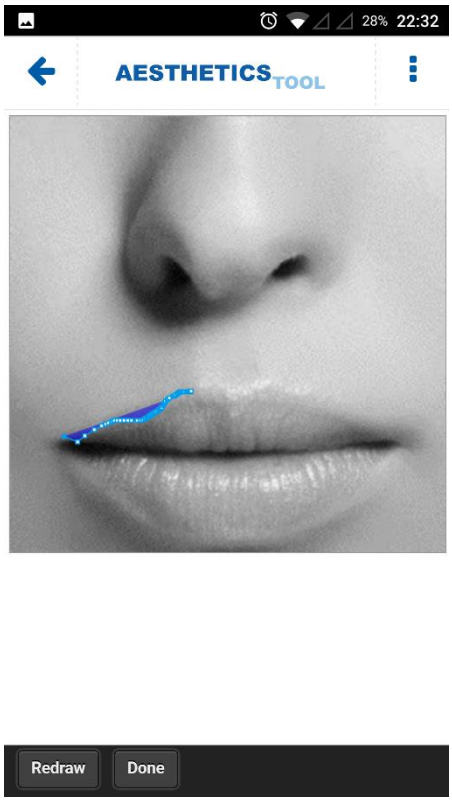


Figure 8.10 – Creating a drawing

Draw over the patient's image to create a trace of the lip regions and once satisfied, click Done.

All of the local drawings for the specific patient will now be displayed with their corresponding symmetry scores.

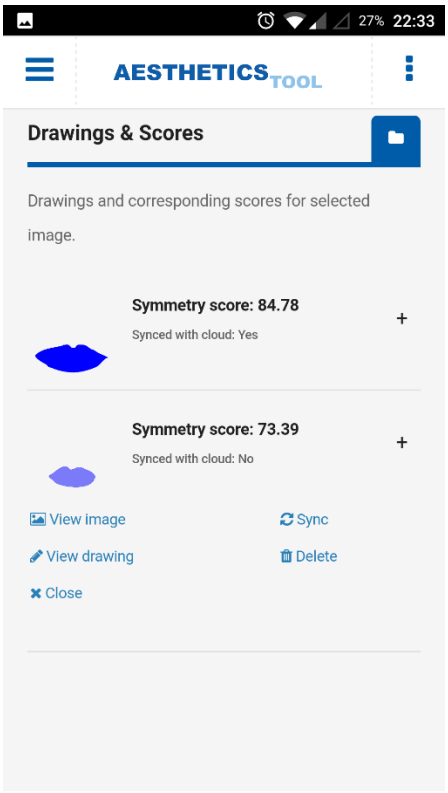


Figure 8.11 – Viewing local drawings and symmetry scores

10.Deleting content

In the local storage for the patient list and drawing lists, each patient image and drawing has an option to be deleted. To delete an item, toggle the options with the plus sign and click Delete. If deleting a patient image, all of its corresponding drawings will be deleted too.

11.Syncing drawings

Similar to deleting items, the syncing option is available for each patient stored locally and for each drawing stored locally. To sync all drawings for a patient, toggle the options with the plus sign in Local Storage and click Sync. You will receive a notification when the upload has been completed. Alternatively, you can sync individual drawings by selecting Sync in the drawing list for a chosen patient.

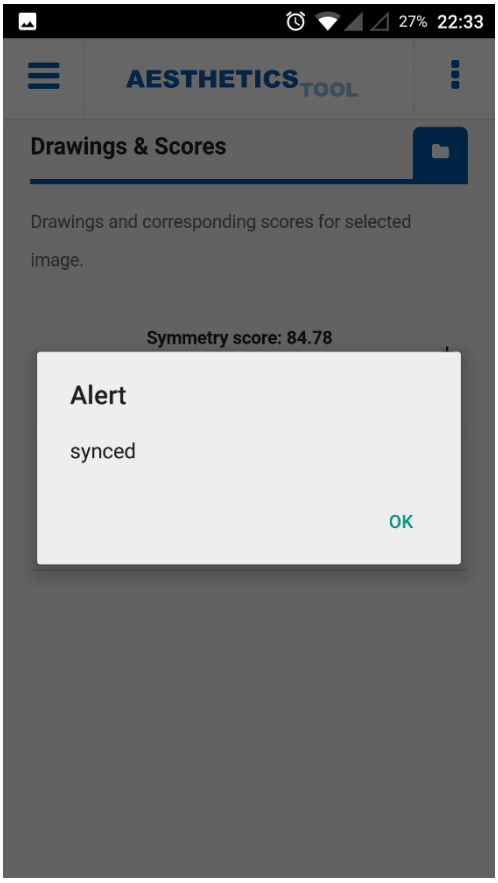


Figure 8.12 – Syncing patient data

8.4. Test results

Functional testing

Functional test results are listed below, covering each MoSCoW requirement. The steps taken, issues and success are detailed.

Tested requirement	**Prerequisites**	**Steps taken**	**Success**	**Issues**
RQ1	Patient image downloaded locally and drawing created.	1. View local patient list. 2. Open details tab and click sync button.	Yes	None
RQ2	None	1. View cloud patient list.	Yes	None
RQ3	Patient image downloaded locally.	1. View local patient list. 2. Open details tab and click create drawing button. 3. Draw over image to create a trace.	Yes	None
RQ4	Patient image downloaded locally and drawing in progress.	1. Click done button.	Yes	None
RQ5	None	1. View cloud patient list. 2. Open details tab and click view drawings & scores button. 3. Open details tab and click view drawing.	Yes	None

RQ6	Patient image downloaded locally and drawing created.	1. View local patient list. 2. Open details tab and click view drawings & scores button. 3. Open details tab and click view drawing button.	Yes	None
RQ7	Patient images downloaded locally.	1. View local patient list. 2. Open details tab.	Yes	None
RQ8	None	Non-functional requirement 1. Evaluated for ease of use with three test users.	Yes	None
RQ9	None	Non-functional requirement 1. Evaluated for design familiarity with three test users. 2. Android design guidelines examined.	Yes	None
RQ10	None	1. Use cases conducted on Android app. 2. Use cases conducted on Windows app.	Yes	None
RQ11	None	1. View cloud patient list. 2. Open details tab and click download button. 3. View local patient list.	Yes	None
RQ12	None	Non-functional requirement	Yes	None

		1. Source code fully commented with clear separation between content. 2. System manual created.		
RQ13	Patient image downloaded locally and drawing in progress.	1. Draw over image to create a trace.	Yes	This feature minimises the number of control points for a user's drawing so that it is simpler to edit the drawing. The benefits won't be seen unless RQ15 is implemented.
RQ14	Patient image downloaded locally and drawing created.	1. View local patient list. 2. Open details tab and click delete button.	Yes	None
RQ15	Patient image downloaded locally and drawing in progress.	1. Draw over image to create a trace. 2. Drag drawing control points to refine the trace.	No	The control points are unable to be moved.
RQ16	Patient image downloaded locally and internet	1. View local patient list. 2. Open details tab and click create drawing button.	Yes	None

	connection disabled.	3. Draw over image to create a trace. 4. Click done button. 5. Open details tab and click view drawing button.		

Acceptance testing

Notable comments provided by each participant testing the app are listed below in a summarised format.

Participant	Comments
1	Unresponsive tab button.
2	N/A
3	Unresponsive tab button. Drawing traces difficult to draw accurately.

8.5. Project plan

Mohammed Farbas Miah

Supervisor: Dr Harry Strange

Project title: Cleft Lip Aesthetics Tool

Aims

The aim of the project is about determining the success of cleft lip and palate surgeries. Through the use of a mobile app, paediatric plastic surgeons should be able to evaluate the aesthetic outcome of the surgery by determining how symmetrical the lips are. The user should be able to draw around the lip region of the target image and then receive a set of symmetry scores, determining the successfulness of the surgery. This would replace the previous, subjective method of having a panel of people determine success.

Objectives

The following objectives need to be accomplished in order to satisfy the project's aims:

- Evaluate the researched options of development tools and choose the best option in order to produce a working multi-platform app.
- Focus on ease of access and uncluttered content to help produce an easy to use app.
- Create an intuitive method of drawing traces around lip regions with a focus on precision to allow for more accurate results.
- Ensure lip tracings and symmetry scores are produced in a format which allows for easier analysis at a later date.

Deliverables

The following deliverables are expected to be produced:

- A fully functional multi-platform application that determines the symmetry of the lips from a patient's image to determine the success of the surgery.

- Fully documented and clear documentation of the app for future development.
- Concise user manual to demonstrate how to use the app.
- An intuitive method of drawing around lip regions of a patient's image to produce a trace for determining success.
- Integrated cloud connectivity with syncing and offline features for the app.
- A simple method to extract and analyse traces and scores of images for future analytical work.
- A final report detailing the planning, methodology and results of the project.

Work plan

The following tasks are expected to be completed under the timeframes shown below:

Project start to early November

Relevant literature review.

Establish main project goals.

Research potential ideas and solutions to problems.

Create and refine project goals and requirements.

Preparation for development through use cases, MoSCoW style requirements, user stories, a Gantt chart and diagrams.

Simple mock ups of app screens.

Drafting and submission of the project plan.

Mid November to mid-December

Development of initial app screens.

Setup of the cloud which stores images as well as produced traces and symmetry scores.

Linking of the app to the cloud.

End December to end January

Development of initial lip drawing features to produce traces.

Addition of symmetry score calculations from traces to determine surgery success.

Drafting and submission of the interim report.

Testing of the features implemented thus far on different platforms.

Early February to mid-March

Enhanced cloud functionality related to traces and symmetry scores.

Development of advanced lip drawing features to enhance precision and reliability.

Addition of offline access and sync features to upload new data to the cloud.

Further symmetry score features based off multiple lip tracings of a single image.

Drafting of the final report.

Late March to early April

Further testing of the system that has been developed.

Production of a user manual.

Submission of the final report.

8.6. Interim report

Mohammed Farbas Miah

Supervisor: Dr Harry Strange

Project title: Cleft Lip Aesthetics Tool

Current status

A majority of the app has been completed and the project is running on schedule. Most of the requirements have been fulfilled barring two major parts, described later. The cloud database portion has been fully complete with two tables in a database for images and drawings. The syncing and offline access on the app both work with drawings and scores being uploaded to the cloud database.

A low-quality drawing feature has been implemented and a method of determining the symmetry score has been set up. An offline database is used for each user of the app with SQLite which stores downloaded images, drawings and other relevant data. The user can delete things locally from this database.

All the screens for the app have been set up with an emphasis on user accessibility, based on Android's guidelines. Drawings and images on the cloud can be viewed and downloaded. Local copies are viewable and editable.

Remaining work

The drawing feature implemented currently is of low quality. This is the biggest piece of work remaining on the project. It is expected that an advanced lip drawing feature will be developed by the end of February which will be connected with the symmetry score feature. Possible improvements to the symmetry score calculation may be made as an additional feature if time permits, allowing multiple drawings in a calculation.

Another major task is to make the app work on multiple platforms. Currently the app has been developed only for Android. This is due to unexpected additional coding requirements

for allowing plugins to work on multiple platforms in Cordova. This is of a lower priority compared to the drawing features which means this task will likely be carried out in early March, assuming the drawing features have been completed.

The app needs to be tested on Android and iOS to check if all the features can work smoothly and to check for missed bugs. Multiple users should also use the app to understand its ease of access and usability. This is planned to be carried out after the features above have been implemented, or mid-March.

A user manual and documentation need to be produced to make usage of the app and future development easier which shall be produced in early April once development has completed.

8.7. Code listing

The most interesting files of code from the app are listed below with a description detailing what the code is responsible for. This code listing is incomplete due to space limitations. The full source code can be found in the GitHub repository.

config.xml

A global configuration file. Controls multiple parts of the Cordova app's behaviour such as icon data, plugins, and build platforms.

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="com.miah.farbas" version="1.0.0"
xmlns="http://www.w3.org/ns/widgets"
xmlns:gap="http://cordova.apache.org/ns/1.0">
      <name>Aesthetics Tool</name>
      <description>
            An application that determines the success of cleft lip and
palate surgeries.
      </description>
      <author email="zcabmfm@ucl.ac.uk" href="http://ucl.ac.uk">
            Farbas Miah
      </author>

      <content src="index.html" />

      <preference name="DisallowOverscroll" value="true" />
      <preference name="android-minSdkVersion" value="14" />
      <preference name="windows-target-version" value="10.0" />

      <plugin name="cordova-plugin-file" source="npm" spec="~4.3.0" />
      <plugin name="cordova-plugin-file-transfer" source="npm"
spec="~1.5.0" />
      <plugin name="cordova-plugin-network-information" source="npm"
spec="~1.2.0" />
      <plugin name="cordova-plugin-whitelist" source="npm" spec="~1.3.0" />
      <plugin name="cordova-sqlite-storage" spec="~1.5.3" />
      <plugin name="cordova-plugin-dialogs" spec="~1.3.2" />

      <icon src="res/icon.png" />
      <platform name="android">
      <icon density="ldpi" src="res/android/ldpi.png" />
      <icon density="mdpi" src="res/android/mdpi.png" />
      <icon density="hdpi" src="res/android/hdpi.png" />
      <icon density="xhdpi" src="res/android/xhdpi.png" />
      <icon density="xxhdpi" src="res/android/xxhdpi.png" />
      <icon density="xxxhdpi" src="res/android/xxxhdpi.png" />
      </platform>

      <platform name="windows">
      <icon src="res/windows/storelogo.png" target="StoreLogo" />
      <icon src="res/windows/smalllogo.png" target="Square30x30Logo" />
      <icon src="res/Windows/Square44x44Logo.png" target="Square44x44Logo"
/>
      <icon src="res/Windows/Square70x70Logo.png" target="Square70x70Logo"
/>
```

```
      <icon src="res/Windows/Square71x71Logo.png" target="Square71x71Logo"
/>
      <icon src="res/Windows/Square150x150Logo.png"
target="Square150x150Logo" />
      <icon src="res/Windows/Square310x310Logo.png"
target="Square310x310Logo" />
      <icon src="res/Windows/Wide310x150Logo.png" target="Wide310x150Logo"
/>
      </platform>

      <access origin="*" />
      <allow-intent href="http://*/*" />
      <allow-intent href="https://*/*" />
      <allow-intent href="tel:*" />
      <allow-intent href="sms:*" />
      <allow-intent href="mailto:*" />
      <allow-intent href="geo:*" />

      <platform name="android">
            <allow-navigation href="*" />
            <allow-intent href="market:*" />
      </platform>

      <engine name="windows" spec="~4.4.3" />
      <engine name="android" spec="~6.1.0" />
</widget>
```

cloud.html

HTML code for displaying the patients list from the cloud. Similar source code for index.html, drawingsscores.html and clouddrawings.html which all display a list of items.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="format-detection" content="telephone=no" />
    <meta name="msapplication-tap-highlight" content="no" />
    <meta name="viewport" content="user-scalable=no, initial-scale=1,
maximum-scale=1, minimum-scale=1, width=device-width" />

      <title>Cloud data - Aesthetics Tool</title>

      <link rel="stylesheet" type="text/css" href="styles/main.css">
      <link rel="stylesheet" type="text/css" href="styles/font-
awesome.css">

      <script type="text/javascript" src="cordova.js"></script>
      <script type="text/javascript" src="js/jquery-3.1.1.js"></script>
      <script type="text/javascript" src="js/angular.min.js"></script>
      <script type="text/javascript" src="js/paddings.js"></script>
      <script type="text/javascript" src="js/navdrawer.js"></script>
      <script type="text/javascript" src="js/cloud.js"></script>
</head>

<body class="left-sidebar">

<!-- Header banner at the top of the page -->
<div id="navigation-header" class="header-border">
```

```
      <a class="navigation-icon open-left-sidebar" href="#"><i class="fa
fa-navicon fa-2x"></i></a>
      <a class="overflow-icon" href="#"><i class="fa fa-ellipsis-v fa-
2x"></i></a>
      <a class="header-label" href="#"><img src="images/label.png"
alt="label"></a>
</div>

<div class="all-elements">
      <div class="snap-drawers">

            <!-- Header banner at the top of the navigation drawer  -->
            <div class="navigation-drawer-header">
                  <a href="#"><img src="images/label.png" alt="label"></a>
                  <a class="close-sidebar" href="#"><i class="fa fa-times
fa-2x"></i></a>
            </div>

            <p class="sidebar-divider">Navigation</p>

            <!-- Menu options for the navigation drawer -->
            <div class="sidebar-menu">
                  <a class="navigation-drawer-option" href="index.html">
                        <i class="fa fa-folder"></i>
                        <em>Local storage</em>
                        <i class="fa fa-circle"></i>
                  </a>
                  <a class="navigation-drawer-option navigation-drawer-
option-active" href="cloud.html">
                        <i class="fa fa-cloud"></i>
                        <em>Cloud data</em>
                        <i class="fa fa-circle"></i>
                  </a>
                  <a class="navigation-drawer-option" href="about.html">
                        <i class="fa fa-info-circle"></i>
                        <em>About</em>
                        <i class="fa fa-circle"></i>
                  </a>
            </div>

            <!-- Main page content goes here -->
            <div id="content" class="snap-content">
                  <div class="content">
                        <div class="header-separation"></div>

                        <!-- Page's title banner -->
                        <div class="title-banner container half-bottom">
                              <i class="fa fa-cloud"></i>
                              <h1>Cloud data</h1>
                              <div class="title-icon-border dark-
blue"></div>
                              <div class="title-border dark-blue"></div>
                        </div>

                        <!-- Page's description -->
                        <div class="container">
                              <p>
                                    Download images from the cloud here.
                              </p>
                        </div>
```

```
                        <!-- List of patients is obtained with JS and
placed in this div -->
                        <div id="list">Loading...</div>

                  </div>
            </div>

      </div>
</div>

</body>
</html>
```

viewcloudimage.html

HTML code for displaying full sized patient images from the cloud. Similar source code for viewlocalimage.html, localdrawingimage.html and clouddrawingimage.html which all display full sized images of patients or drawings.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="format-detection" content="telephone=no" />
    <meta name="msapplication-tap-highlight" content="no" />
    <meta name="viewport" content="user-scalable=no, initial-scale=1,
maximum-scale=1, minimum-scale=1, width=device-width" />

      <title>View image - Aesthetics Tool</title>

      <link href="styles/main.css" rel="stylesheet" type="text/css">
      <link href="styles/font-awesome.css" rel="stylesheet"
type="text/css">

      <script type="text/javascript" src="js/jquery-3.1.1.js"></script>
      <script type="text/javascript" src="js/angular.min.js"></script>
      <script type="text/javascript" src="js/paddings.js"></script>
      <script type="text/javascript" src="cordova.js"></script>
</head>

<body>

<!-- Header banner at the top of the page -->
<div id="navigation-header" class="header-border">
      <a class="navigation-icon" href="cloud.html"><i class="fa fa-arrow-
left fa-2x"></i></a>
      <a class="overflow-icon" href="#"><i class="fa fa-ellipsis-v fa-
2x"></i></a>
      <a class="header-label" href="#"><img src="images/label.png"
alt="label"></a>
</div>

<div class="all-elements">
      <div class="snap-drawers">

            <!-- Main page content goes here -->
            <div id="content" class="snap-content">
                  <div class="content">
                        <div class="header-separation"></div>
```

```
                        <!-- Page's title banner -->
                        <div class="title-banner container half-bottom">
                              <i class="fa fa-picture-o"></i>
                              <h1>View image</h1>
                              <div class="title-icon-border dark-
blue"></div>
                              <div class="title-border dark-blue"></div>
                        </div>

                        <!-- Image is obtained with JS and placed in this
div -->
                        <div id="image">Loading...</div>
                        <script>
                              document.getElementById("image").innerHTML =
'<img class="view-image-size" src="' + sessionStorage.getItem("link") + '"
alt="img">';
                        </script>

                  </div>
            </div>

      </div>
</div>

</body>
</html>
```

createdrawing.html

HTML code for the drawing creation page. Contains inline JavaScript code making use of Paper.js to allow users to draw.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="format-detection" content="telephone=no" />
    <meta name="msapplication-tap-highlight" content="no" />
    <meta name="viewport" content="user-scalable=no, initial-scale=1,
maximum-scale=1, minimum-scale=1, width=device-width" />

      <title>Create drawing - Aesthetics Tool</title>

      <link href="styles/main.css" rel="stylesheet" type="text/css">
      <link href="styles/font-awesome.css" rel="stylesheet"
type="text/css">
      <link href="styles/drawing.css" rel="stylesheet" type="text/css">

      <script type="text/javascript" src="js/jquery-3.1.1.js"></script>
      <script type="text/javascript" src="js/angular.min.js"></script>
      <script type="text/javascript" src="js/paper-full.js"></script>
      <script type="text/javascript" src="js/createdrawing.js"></script>
      <script type="text/javascript" src="cordova.js"></script>
</head>

<body>

<!-- Header banner at the top of the page -->
<div id="navigation-header" class="header-border">
```

```
      <a class="navigation-icon" href="index.html"><i class="fa fa-arrow-
left fa-2x"></i></a>
      <a class="overflow-icon" href="#"><i class="fa fa-ellipsis-v fa-
2x"></i></a>
      <a class="header-label" href="#"><img src="images/label.png"
alt="label"></a>
</div>

<!-- Stops overlap of the canvas and banner -->
<div class="header-separation"></div>

<canvas id="canvas" style="overflow:hidden;position:relative;border:#999999
1px solid;margin:1vw;"></canvas>


<!-- Bottom toolbar with buttons -->
<div class="toolbar">
      <div class="case">
            <div class="box">
                  <a class="navbtn" href="createdrawing.html">Redraw</a>
            </div>
            <div class="box">
                  <a class="navbtn" onclick="prepDrawing()">Done</a>
            </div>
      </div>
</div>

<!-- Script which allows drawing over an image and then saves it -->
<script>
      var canvas = document.getElementsByTagName('canvas')[0];

      // Checks if drawing has been created
      var drawn = 0;

      // Sets the canvas to fit the largest region of the device's screen
      if (window.innerWidth <= window.innerHeight) {
            canvas.width = window.innerWidth * 0.98;
            canvas.height = window.innerWidth * 0.98;
      }
      else {
            canvas.width = window.innerHeight * 0.98 - 110;
            canvas.height = window.innerHeight * 0.98 - 110;
      }

      paper.install(window);
      // Keep global references to both tools, so the HTML links below can
access them.
      var tool1, tool2;
      var raster;

      window.onload = function() {
            paper.setup('canvas');

            // Sets patient's image as background
            raster = new Raster(sessionStorage.getItem("canvas"));
            raster.size = paper.view.viewSize;

            // Move the image to the centre of the view
            raster.position = paper.view.center;

            var path, segment;
```

```
            var hitOptions = {
            segments: true,
            stroke: true,
            fill: true,
            tolerance: 5
        };

            // Tool 1 allows the drawing to be created with segments
            tool1 = new Tool();
            tool1.onMouseDown = function(event) {
                  // If we produced a path before, deselect it:
            if (path) {
                path.selected = false;
            }

            // Create a new path and set its stroke color to black:
            path = new Path({
                segments: [event.point],
                strokeColor: new Color(0, 0, 1, 0.2),
                        fillColor: new Color(0, 0, 1, 0.2),
                // Select the path, so we can see its segment points:
                fullySelected: true
            });
            }

            tool1.onMouseDrag = function(event) {
                  path.add(event.point);
                  drawn = 1;
            }

            // When the mouse is released, we simplify the path:
        tool1.onMouseUp = function(event) {
                  path.closed = true;

            // When the mouse is released, simplify it:
            path.simplify(10);

            // Select the path, so we can see its segments:
            path.fullySelected = false;
        }

            // Tool 2 allows the user to adjust the segments (control
points) to refine the shape of the drawing. NOT IMPLEMENTED
            tool2 = new Tool();
            tool2.minDistance = 20;
            tool2.onMouseDown = function(event) {

            }

            tool2.onMouseDrag = function(event) {

            }

            tool2.onMouseMove = function(event) {

            }
      }

      // Save the canvas drawing
      function prepDrawing() {
```

```
            if (drawn == 1) {
                  setTimeout(function() {
                        saveDrawing();
                  }, 100);
                  raster.remove();
            }
            else navigator.notification.alert('Create a drawing before
submitting.');
      }

</script>
</body>
</html>
```

createdrawing.js

The JavaScript code used by createdrawing.html. Contains functions which calculate the symmetry score from the user's drawing and then saves the relevant data to the SQLite database.

```
// Global database variable
var myDB;

// Once device is ready, open the database
document.addEventListener("deviceready",onDeviceReady,false);
function onDeviceReady(){
      myDB = window.sqlitePlugin.openDatabase({name: "mySQLite.db",
location: 'default'});
}

// Save the drawing and calculate a symmetry score from it
function saveDrawing() {
      var drawing = document.getElementById("canvas").toDataURL();
      var pid = sessionStorage.getItem("pid");
      var uploaded = "No";
      var score;

      var initial = [];
      var img = new Image();
      img.onload = function() {
            var canvas = document.createElement('canvas');

            // Make sure the image has an even number of width pixels
            if (img.width%2)
                  canvas.width = img.width - 1;
            else canvas.width = img.width;

            canvas.height = img.height;
            canvas.getContext('2d').drawImage(img, 0, 0, canvas.width,
canvas.height);

            // Create a 2d array of 1s and 0s for each pixel with 1s being
pixels with blue in them, indicating a drawing
            for (var i = 0; i < canvas.height; i++) {
                  initial[i] = [];
            }
            var pixelData = canvas.getContext('2d').getImageData(0, 0,
canvas.width, canvas.height).data;
            var blue = 2;
```

```
            for (var i = 0; i < canvas.height; i++) {
                  for (var j = 0; j < canvas.width; j++) {
                        if (pixelData[blue] != 0) {
                              initial[i][j] = 1;
                        } else initial[i][j] = 0;
                        blue += 4;
                  }
            }

            var left = [];
            var right = [];
            var overlap = [];

            // Split the 2d array in half and compare the two to determine
an overlap percentage
            for (var i = 0; i < initial.length; i++) {
                  left[i] = [];
                  right[i] = [];
                  overlap[i] = [];
            }

            var ones = 0;
            var twos = 0;

            for (var i = 0; i < initial.length; i++) {
                  for (var j = 0; j < initial[0].length / 2; j++) {
                        left[i][j] = initial[i][j];
                  }
            }

            for (var i = 0; i < initial.length; i++) {
                  for (var j = initial[0].length / 2; j <
initial[0].length; j++) {
                        right[i][j - initial[0].length / 2] =
initial[i][j];
                  }
            }

            for (var i = 0; i < right.length; i++) {
                  for (var j = 0; j < right[i].length / 2; j++) {
                        var temp = right[i][j];
                        right[i][j] = right[i][right[i].length - j - 1];
                        right[i][right[i].length - j - 1] = temp;
                  }
            }

            for (var i = 0; i < left.length; i++) {
                  for (var j = 0; j < left[0].length; j++) {
                        overlap[i][j] = left[i][j] + right[i][j];
                        if (overlap[i][j] == 1) {
                              ones++;
                        }
                        else if (overlap[i][j] == 2) {
                              twos++;
                        }
                  }
            }

            // Obtaine the overlap percentage
            score = twos / (ones + twos) * 100;
```

```
            score = Math.round(score * 100) / 100;

            // Save the symmetry score and relevant data locally
            myDB.transaction(function(transaction) {
                  transaction.executeSql("INSERT INTO drawings_local
(drawing, score, pid, uploaded) VALUES (?,?,?,?)", [drawing, score, pid,
uploaded], function(tx, result) {
                        sessionStorage.setItem("pid", pid);
                        window.open('drawingsscores.html', '_self', false);
                  },
                  function(error) {
                        navigator.notification.alert('Error occurred');
                  });
            });
      }
      img.src = drawing;

}
```

main.js

The JavaScript code used by index.html. Prepares the SQLite database and displays a list of the patients. Contains functions for each option for the list items including syncing and deleting. Similar code is present in drawingsscores.js with slightly different functions.

```
// Global database variable and list array
var patients = [];
var myDB;

// Once device is ready, open the database and create the relevant tables
if they have not been created already
document.addEventListener("deviceready",onDeviceReady,false);
function onDeviceReady() {
      myDB = window.sqlitePlugin.openDatabase({name: "mySQLite.db",
location: 'default'});

      myDB.transaction(function(transaction) {
            transaction.executeSql('CREATE TABLE IF NOT EXISTS
patients_local (id text, name text, date text, image text)', []);
      });
      myDB.transaction(function(transaction) {
            transaction.executeSql('CREATE TABLE IF NOT EXISTS
drawings_local (drawing text, score text, pid text, uploaded text)', []);
      });

      showTable();
}

// Obtain the data of the patients that have been downloaded locally
function showTable() {
      myDB.transaction(function(transaction) {
      transaction.executeSql('SELECT * FROM patients_local', [], function
(tx, results) {
            var len = results.rows.length, i;
            for (i = 0; i < len; i++) {
                  patients[i] = {"id":results.rows.item(i).id,
"name":results.rows.item(i).name, "date":results.rows.item(i).date,
"image":results.rows.item(i).image};
            }
```

```
            displayList(patients);
      }, null);
      });
}

// Displays the list of data obtained with selectable options
function displayList(arr) {
      var i;
      var out = '';

      for (i = 0; i < arr.length; i++) {
            out +=
            '<div class="activity-item container">' +
                  '<div ng-app="croppy"><img src="' + arr[i].image + '"
width="70" height="70" alt="thumbnail"></div>' +
                  '<h2>' + arr[i].name + '</h2>' +
                  '<em>' + arr[i].date + '</em>' +
                  '<a href="#" onclick="toggle(\'' + arr[i].id + '\')"
class="activity-item-toggle" style="font-size:18px;">+</a>' +
                  '<div id="selected' + arr[i].id + '" class="activity-
item-detail">' +
                        '<table style="width:100%">' +
                              '<tr>' +
                                    '<td><a href="#" onclick="viewImage(\''
+ arr[i].image + '\')"><i class="fa fa-picture-o"></i> View image</a></td>'
+
                                    '<td><a href="#" onclick="sync(\'' +
arr[i].id + '\')"><i class="fa fa-refresh"></i> Sync</a></td>' +
                              '</tr>' +
                              '<tr>' +
                                    '<td><a href="#" onclick="drawing(\'' +
arr[i].image + '\', \'' + arr[i].id + '\')"><i class="fa fa-pencil"></i>
Create drawing</a></td>' +
                                    '<td><a href="#"
onclick="removePatient(\'' + arr[i].id + '\')"><i class="fa fa-trash"></i>
Delete</a></td>' +
                              '</tr>' +
                              '<tr>' +
                                    '<td><a href="#"
onclick="drawingsScores(\'' + arr[i].id + '\')"><i class="fa fa-bar-
chart"></i> View drawings and scores</a></td>' +
                                    '<td><a href="#"><i class="fa fa-
times"></i> Close</a></td>' +
                              '</tr>' +
                        '</table>' +
                  '</div>' +
            '</div>' +
            '<div class="border"></div>';
      }

      document.getElementById("list").innerHTML = out;
}

// Deletes the selected patient and its drawings
function removePatient(id) {
      myDB.transaction(function(transaction) {
            transaction.executeSql("DELETE FROM patients_local where id=?",
[id], function(tx, result) {
                  myDB.transaction(function(transaction) {
                        transaction.executeSql("DELETE FROM drawings_local
where pid=?", [id], function(tx, result) {
```

```
                              location.reload();
                        },
                        function(error){
navigator.notification.alert('Something went Wrong deleting drawings.');});
                  });
            },
            function(error){ navigator.notification.alert('Something went
Wrong');});
      });
}

// Uploads all of the selected patient's drawings to the cloud database
function sync(id) {
      var uploaded = "No";

      myDB.transaction(function(transaction) {
            transaction.executeSql("SELECT * FROM drawings_local where
pid=? AND uploaded=?", [id,uploaded], function(tx, results) {
                  if (results.rows.length == 0) {
                        navigator.notification.alert("Already synced");
                  } else {
                        var success = 1, i;
                        uploaded = "Yes";

                        for (i = 0; i < results.rows.length; i++) {
                              var dataString = "drawing=" +
results.rows.item(i).drawing + "&score=" + results.rows.item(i).score +
"&pid=" + results.rows.item(i).pid + "&uploaded=" + uploaded + "&insert=";
                              $.ajax({
                                    type: "POST",
                                    url: "https://aesthetics-
tool.000webhostapp.com/upload_drawing.php",
                                    data: dataString,
                                    crossDomain: true,
                                    cache: false,
                                    success: function(data) {
                                          if (data == "success") {

      myDB.transaction(function(transaction) {

      transaction.executeSql("UPDATE drawings_local SET uploaded=? WHERE
pid=?", [uploaded,id], function(tx, result) {
                                                      },

      function(error){success = 0;});
                                                });
                                          } else if (data == "error") {
                                                success = 0;
                                          }
                                    }
                              });
                        }

                        if (success == 1) {
                              navigator.notification.alert("Sync
complete");
                        } else {
                              navigator.notification.alert("Something went
wrong");
                        }
                  }
```

```
            },
            function(error){ navigator.notification.alert('Something went
Wrong');});
      });
}

// Displays the selected patient's image
function viewImage(image) {
      sessionStorage.setItem("link", image);
      window.open('viewlocalimage.html', '_self', false);
}

// Displays all of the local drawings for a selected patient
function drawingsScores(id) {
      sessionStorage.setItem("pid", id);
      window.open('drawingsscores.html', '_self', false);
}

// Allows the user to draw the trace to create a symmetry score
function drawing(image, id) {
      sessionStorage.setItem("canvas", image);
      sessionStorage.setItem("pid", id);
      window.open('createdrawing.html', '_self', false);
}
```

cloud.js

The JavaScript code used by cloud.html. Prepares the SQLite database and displays a list of the patients listed on the cloud database. Contains functions for each option for the list items including downloading and viewing drawings. Similar code is present in clouddrawingsscores.js with slightly different functions.

```
// Global database variable and list array
var patients = [];
var myDB;

// Once device is ready, open the database
document.addEventListener("deviceready",onDeviceReady,false);
function onDeviceReady() {
      myDB = window.sqlitePlugin.openDatabase({name: "mySQLite.db",
location: 'default'});

      cloudTable();
}

// Obtain the data of the patients that are available for download on the
cloud database.
function cloudTable() {
      var count = 0;
      var url = "https://aesthetics-
tool.000webhostapp.com/patient_list.php";
      $.getJSON(url, function(result) {
            $.each(result, function(i, field) {
                  patients[count] = {"id":field.id, "name":field.name,
"date":field.date, "image":field.image};
                  count++;
            });
            displayList(patients);
```

```
	});
}

// Displays the list of data obtained with selectable options
function displayList(arr) {
	var i;
	var out = '';

	for (i = 0; i < arr.length; i++) {
		out +=
		'<div class="activity-item container">' +
			'<div ng-app="croppy"><img src="' + arr[i].image + '"
width="70" height="70" alt="thumbnail"></div>' +
			'<h2>' + arr[i].name + '</h2>' +
			'<em>' + arr[i].date + '</em>' +
			'<a href="#" onclick="toggle(\'' + arr[i].id + '\')"
class="activity-item-toggle" style="font-size:18px;">+</a>' +
			'<div id="selected' + arr[i].id + '" class="activity-
item-detail">' +
				'<table style="width:100%">' +
					'<tr>' +
						'<td><a href="#" onclick="viewImage(\''
+ arr[i].image + '\')"><i class="fa fa-picture-o"></i> View image</a></td>'
+
						'<td><a href="#"
onclick="downloadFile(\'' + arr[i].id + '\', \'' + arr[i].name + '\', \'' +
arr[i].date + '\', \'' + arr[i].image + '\')"><i class="fa fa-
download"></i> Download</a></td>' +
					'</tr>' +
					'<tr>' +
						'<td><a href="#"
onclick="cloudDrawingsScores(\'' + arr[i].id + '\')"><i class="fa fa-bar-
chart"></i> View drawings and scores</a></td>' +
						'<td><a href="#"><i class="fa fa-
times"></i> Close</a></td>' +
					'</tr>' +
				'</table>' +
			'</div>' +
		'</div>' +
		'<div class="border"></div>';
	}

	document.getElementById("list").innerHTML = out;
}

// Displays the selected patient's image
function viewImage(image) {
	sessionStorage.setItem("link", image);
	window.open('viewcloudimage.html', '_self', false);
}

// Displays all of the cloud drawings for a selected patient
function cloudDrawingsScores(id) {
	sessionStorage.setItem("pid", id);
	window.open('clouddrawingsscores.html', '_self', false);
}

// Downloads the selected patient
function downloadFile(idDB, nameDB, dateDB, imageDB) {
	var myDB = window.sqlitePlugin.openDatabase({name: "mySQLite.db",
location: 'default'});
```

```
	myDB.transaction(function (tx) {
		tx.executeSql("SELECT id FROM patients_local WHERE id=?",
[idDB], function (tx, result) {
			if (result.rows.length == 0) {
				myDB.transaction(function(transaction) {
					var executeQuery = "INSERT INTO
patients_local (id, name, date, image) VALUES (?,?,?,?)";
					transaction.executeSql(executeQuery, [idDB,
nameDB, dateDB, imageDB]
						, function(tx, result) {
							console.log('Inserted');

navigator.notification.alert("Downloaded");
						},
						function(error){

navigator.notification.alert('Error occurred');
						});
				});
			}
			else  navigator.notification.alert("You have already
downloaded this image.");
		});
	});
}
```

padding.js

Contains code to set the borders and paddings for all pages in the app. It also accounts for resizing of the app.

```
document.addEventListener("deviceready",onDeviceReady,false);
function onDeviceReady() {
	//Make container fullscreen and dynamically adjust the screen when
resized.
	function create_paddings() {
		var no_padding = $(window).width();

		if($(window).width() < 766) {
			$('.content').css('padding-left', '20px');
			$('.content').css('padding-right', '20px');
			$('.container-fullscreen, .image-
fullscreen').css('margin-left', '-21px');
			$('.container-fullscreen, .image-
fullscreen').css('width', no_padding +2);
		}
		if($(window).width() >= 766) {
			$('.content').css('padding-left', '50px');
			$('.content').css('padding-right', '50px');
			$('.container-fullscreen, .image-
fullscreen').css('margin-left', '-51px');
			$('.container-fullscreen, .image-
fullscreen').css('width', no_padding +2);
		}
	}

	$(window).resize(function() {
		create_paddings();
	});
```

```
	create_paddings();
}
```

navdrawer.js

A partial listing of the code used by the four HTML pages which contain navigation drawers. The Snap.js code is contained in this file. It also contains functions for list item image thumbnails and details tab toggling.

```
// Displays the dropdown menu for a selected item with the onclick event
function toggle(selected) {
    document.getElementById("selected" +
selected).classList.toggle("show");
}

window.onclick = function(event) {
	if (!$(event.target).hasClass('activity-item-toggle')) {
		var details = document.getElementsByClassName("activity-item-
detail");
		var x;
		for (x = 0; x < details.length; x++) {
			var expand = details[x];
			if (expand.classList.contains('show')) {
				expand.classList.remove('show');
			}
		}
	}
}

// Navigation drawer toggle
$(document).ready(function() {
	var $div = $('<div />').appendTo('body');
	$div.attr('id', 'footer-fixed');
	$div.attr('class', 'not-active');

	var drawer = new Snap({
		element: document.getElementById('content'),
		elementMirror: document.getElementById('navigation-header'),
		elementMirror2: document.getElementById('footer-fixed'),
		disable: 'right',
		tapToClose: true,
		touchToDrag: true,
		maxPosition: 266,
		minPosition: -266
	});

	$('.close-sidebar').click(function(){drawer.close();});

	$('.open-left-sidebar').click(function() {
		if (drawer.state().state == "left") {
			drawer.close();
		} else {
			drawer.open('left');
		}
		return false;
	});

	drawer.on('open', function() {
```

```
            $('.back-to-top-badge').removeClass('back-to-top-badge-
visible');
        });
});
```

main.css

This is the main CSS code for the app. It is made use of by all HTML pages for formatting of content to fit the theme of the app.

```
/* Header banner at the top of the page */
#navigation-header {
        opacity: 1;
        position: fixed;
        width: 100%;
        height: 60px;
        background-color: #FFFFFF;
        z-index: 2;
}
.header-border {
        border-top: rgba(0, 0, 0, 0.2) 1px solid;
        border-bottom: rgba(0, 0, 0, 0.2) 1px solid;
}
.header-border .navigation-icon {
        color: #005ba8;
        float: left;
        width: 60px;
        height: 55px;
        border-right: rgba(0, 0, 0, 0.2) dashed 1px;
}
.header-border .navigation-icon i {
        height: 55px;
        width: 60px;
        text-align: center;
        line-height: 57px;
}
.header-border .header-label {
        position: absolute;
        left: 50%;
        margin-left: -90px;
}
.header-border .header-label img {
        width: 170px;
        margin-top: 20px;
}
.header-border .overflow-icon {
        color: #005ba8;
        float: right;
        width: 60px;
        height: 55px;
        border-left: rgba(0, 0, 0, 0.2) dashed 1px;
}
.header-border .overflow-icon i {
        width: 60px;
        height: 55px;
        line-height: 57px;
        text-align: center;
}
.header-separation {
```

```
        height: 60px;
}

/* Header banner at the top of the navigation drawer */
.navigation-drawer-header {
        height: 60px;
        border-bottom: rgba(255, 255, 255, 0.1) solid 1px;
        background-color: #FFFFFF;
}
.navigation-drawer-header a img {
        width: 150px;
        margin-top: 20px;
        margin-left: 25px;
}
.navigation-drawer-header a:first-child {
        width: 130px;
        float: Left;
}
.navigation-drawer-header a:last-child {
        width: 60px;
        height: 60px;
        line-height: 60px;
        color: #005ba8;
        position: absolute;
        left: 205px;
        text-align: center;
        border-left: rgba(0, 0, 0, 0.2) dashed 1px;
        border-right: rgba(0, 0, 0, 0.2) dashed 1px;
}
.navigation-drawer-header a i {
        width: 54px;
        height: 60px;
        line-height: 60px;
        text-align: center;
}

/* Menu options for the navigation drawer */
.navigation-drawer-option {
        height: 50px;
        color: #005ba8;
        transition: all 200ms ease;
}
.navigation-drawer-option-active .fa-circle {
        color: #8fc3ea!important;
        opacity: 1!important;
        font-size: 6px!important;
        margin-left: 231px!important
}
.navigation-drawer-option-active {
        color: #8fc3ea;
        transition: all 200ms ease;
}
.navigation-drawer-option em:hover {
        color: #8fc3ea;
        transition: all 200ms ease;
}
.navigation-drawer-option i:first-child {
        position: absolute;
```

```
        font-size: 18px;
        width: 20px;
        text-align: center;
        top: 50%;
        margin-top: -8px;
        margin-left: 30px;
}
.navigation-drawer-option i:last-child {
        position: absolute;
        font-size: 5px;
        margin-left: 232px;
        top: 50%;
        margin-top: -1px;
        opacity: 0.5;
}
.navigation-drawer-option em {
        font-family: 'Roboto', sans-serif;
        display: block;
        line-height: 52px;
        font-style: normal;
        padding-left: 75px;
        font-size: 13px;
        font-weight: 500;
        transition: all 200ms ease;
}

/* Formatting for navigation drawer */
.sidebar-menu {
        margin-bottom: 30px;
}
.snap-drawer, .snap-drawers {
        background-color: #272c2e!important;
}
.sidebar-divider {
        font-family: 'Roboto', sans-serif;
        margin-top: 10px;
        margin-bottom: 20px;
        padding-bottom: 5px;
        font-size: 10px;
        padding-left: 30px;
        font-weight: 800;
        text-transform: uppercase;
        color: #8fc3ea;
        border-bottom: rgba(255, 255, 255, 0.1) solid 1px;
}

/* Sets up page content */
.all-elements * {
        -webkit-text-size-adjust: none;
        -webkit-transform: translateZ(0);
        min-height: auto;
        max-height: auto;
}
.snap-drawers {
        position: absolute;
        top: 0;
        right: 0;
        bottom: 0;
        left: 0;
}
```

```
/* Main page content that moves when navigation drawer is opened */
.snap-content {
	position: absolute;
	top: 0;
	right: 0;
	bottom: 0;
	left: 0;
	width: auto;
	height: auto;
	z-index: 2;
	overflow: auto;
	-webkit-overflow-scrolling: touch;
	-webkit-transform: translate3d(0, 0, 0);
	-moz-transform: translate3d(0, 0, 0);
	-ms-transform: translate3d(0, 0, 0);
	-o-transform: translate3d(0, 0, 0);
	transform: translate3d(0, 0, 0);
}

/* Page's title banner */
.title-banner {
	cursor: default;
	margin-top: 10px;
}
.title-banner .title-border {
	clear: both;
	display: block;
	height: 4px;
	margin-top: 13px;
}
.title-banner i {
	color: #FFFFFF;
	z-index: 2;
	position: absolute;
	right: 0px;
	font-size: 14px;
	margin-top: -5px;
	width: 40px;
	height: 40px;
	text-align: center;
	line-height: 40px;
}
.title-banner .title-icon-border {
	position: absolute;
	height: 40px;
	width: 40px;
	right: 0px;
	top: -5px;
	z-index: 1;
	border-top-left-radius: 5px;
	border-top-right-radius: 5px;
}
.half-bottom {
	margin-bottom: 15px!important;
}

/* Page's description */
.container {
	margin-bottom: 30px;
	display: block;
```

```
}

/* Sets the size of the images when viewed individually */
.view-image-size {
	max-width: 100%;
	max-height: 600px;
}

/* Each list item which contains an image, title and sub-title */
.activity-item:hover {
	cursor: pointer;
}
.activity-item img {
	width: 70px;
	height: 70px;
	border-radius: 3px;
	position: absolute;
}
.activity-item h2 {
	font-weight: 600;
	font-size: 14px;
	margin-left: 85px;
	margin-bottom: 0px;
	padding-top: 3px;
}
.activity-item em {
	display: block;
	margin-left: 85px;
	font-size: 11px;
	font-style: normal;
	font-weight: 500;
	color: rgba(0, 0, 0, 0.5);
}

/* Each list item can be toggled to display further detail */
.activity-item-toggle {
	font-size: 8px;
	position: absolute;
	height: 50px;
	width: 30px;
	top: 0px;
	right: 0px;
	line-height: 50px;
	text-align: center;
	color: #1f1f1f;
}
.activity-item-detail {
	padding-top: 20px;
	font-size: 13px;
	display: none;
}
.show {
	display: block;
}

/* Separating border between list items */
.border {
	height: 1px;
	width: 100%;
```

```
        display: block;
        background-color: rgba(0, 0, 0, 0.1);
        margin-bottom: 20px;
        clear: both;
}

/* Prevent text selection */
* {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
        -webkit-touch-callout: none; /* iOS Safari */
        -webkit-user-select: none; /* Chrome/Safari/Opera */
        -khtml-user-select: none; /* Konqueror */
        -moz-user-select: none; /* Firefox */
        -ms-user-select: none; /* Internet Explorer/Edge */
        user-select: none; /* Non-prefixed version, currently not supported
by any browser */
        -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
        tap-highlight-color: rgba(0, 0, 0, 0);
}

/* Multiple formatting styles */
a {
        color: #2980b9;
        text-decoration: none;
        display: block;
        position: relative;
}
h1 {
        font-size: 18px;
        line-height: 22px;
        font-weight: 700;
        color: #1f1f1f;
}
h2 {
        font-size: 16px;
        line-height: 20px;
        font-weight: 700;
        color: #1f1f1f;
}
body {
        font-size: 14px;
        font-family: 'Roboto', sans-serif;
        line-height: 30px;
        font-weight: 400;
        color: #666666;
        margin: 0;
        padding: 0;
        overflow-x: hidden;
}

/* Sets the background colour */
#content {
        background-color: #F5F5F5;
        overflow-x: hidden;
}
```

```
/* Title bar logo */
.button {
	border: none;
	color: white;
	width: 30%;
	padding: 15px 17px;
	text-align: center;
	text-decoration: none;
	display: inline-block;
	font-size: 16px;
}

/* Theme colours - Note the colour codes are used in this file multiple
times */
.light-blue {
	background-color: #8fc3ea!important;
}
.dark-blue {
	background-color: #005ba8!important;
}
```

drawing.css

CSS code used for the createdrawing.html page to ensure correct formatting of the content.

```
body {
	/* prevent text selection on ui */
	user-select: none;
	-webkit-user-select: none;
	-moz-user-select: none;
	-ms-user-select: none;

	/* prevent scrolling in windows phone */
	-ms-touch-action: none;

	/* prevent selection highlight */
	-webkit-tap-highlight-color: rgba(0,0,0,0);
}

/* Provides the buttons with a shadow on click */
.navbtn {
	cursor: pointer;
	float:left;
	padding: 6px 10px;
	font-weight: bold;
	line-height: 18px;
	font-size: 14px;
	color: #eee;
	text-shadow: 0px -1px #000;
	border: solid 1px #111;
	border-radius: 4px;
	background-color: #404040;
	box-shadow: 0 0 1px 1px #555,inset 0 1px 0 0 #666;
}
.navbtn-hover, .navbtn:active {
	color: #222;
	text-shadow: 0px 1px #aaa;
	background-color: #aaa;
	box-shadow: 0 0 1px 1px #444,inset 0 1px 0 0 #ccc;
}
```

```
/* Provides the bottom toolbar where the options are placed */
.toolbar {
	position: absolute;
	background-color: #222;
	text-align: center;
	bottom: 0px;
	left: 0px;
	right: 0px;
	height: 42px;
	padding:2px;
}

/* Sections in the toolbar for buttons with spacing */
.case {
	width:350px;
	margin:auto;
	text-align:center;
}
.box {
	float:left;
	padding:2px 6px 2px 6px;
}

/* Sets the cursor to a crosshair for easier drawing */
#canvas{
	cursor:crosshair;
	background-color:#fff;
}
```

patient_list.php

Server-side code which lets the patient data be obtained from the SQL database. Similar code is in the drawing_list.php file which obtains drawing data. Makes use of db.php which connects to the database.

```
<?php
include "db.php";

$data = array();
$q = mysqli_query($con, "select * from `images`");

while ($row = mysqli_fetch_object($q)) {
	$data[] = $row;
}

echo json_encode($data);
?>
```

upload_drawing.php

Server-side code which lets users upload their drawings to the SQL database.

```
<?php
include "db.php";

if (isset($_POST['insert'])) {
	$drawing = mysqli_real_escape_string($con, $_POST['drawing']);
	$score = $_POST['score'];
```

```
	$pid = $_POST['pid'];
	$uploaded = $_POST['uploaded'];

	$q = mysqli_query($con, "INSERT INTO `drawings` (`drawing`, `score`,
`pid`, `uploaded`) VALUES ('$drawing', '$score', '$pid', '$uploaded')");

	if($q)
		echo "success";
	else
		echo "error";
}
 ?>
```