



College of Engineering

CS CAPSTONE PROJECT HAND OFF

JUNE 10, 2019

APP TO SUPPORT FIELD DIAGNOSTICS IN VETERINARY MEDICINE

PREPARED FOR

OREGON VETERINARY DIAGNOSTIC LABORATORY

DR. CHRISTIANE LOEHR

Signature

Date

PREPARED BY

GROUP 72 MALSANO

BRANDON JOLLY

Signature

Date

KATHERINE JEFFREY

Signature

Date

BRADFORD WONG

Signature

Date

Abstract

Currently, there are many difficulties for veterinary pathologists trying to perform remote diagnostics. There are not any effective ways for people out in the field collecting samples to communicate with specialized experts located in laboratories. As a result, this project will involve creating an Android mobile application that will be used as a bridge to connect the field personnel with the veterinary pathologists in laboratories. With this mobile application, the field personnel will be able to take pictures of the individual that is being analyzed and then send the pictures along with other data such as the patient, location, and time to a pathologist. The pathologist will then be able to use the provided information to perform a necropsy and send feedback to the field personnel. This project is intended to support remote field diagnostics in veterinary medicine.

CONTENTS

1	Introduction to Project	6
1.1	The Project	6
1.2	The Clients	6
1.3	The Team	6
2	Requirements Document	6
2.1	Change Log	6
2.2	System Purpose	8
2.3	System Overview	9
2.3.1	System Context	9
2.3.2	System Functions	9
2.3.3	User Characteristics	9
2.4	Definitions	9
2.5	Requirements	9
2.5.1	Functional Requirements	9
2.5.2	Usability Requirements	10
2.5.3	Performance Requirements	10
2.6	Interfaces	10
2.6.1	User Interface	10
2.6.2	Software Interfaces	10
2.7	System Modes and States	10
2.7.1	Network Connection	10
2.7.2	Database Synchronization	11
2.8	System Security	11
2.8.1	Information Management	11
2.8.2	Policies and Regulations	11
2.9	Verification	11
2.9.1	Debugging	11
2.9.2	Field Tests	11
2.10	Appendices	11
2.10.1	Assumptions and Dependencies	11
2.10.2	Acronyms and Abbreviations	11
2.11	Gantt Chart	12
3	Design Document	12
3.1	Change Log	12
3.2	Introduction	22
3.3	Android Application	22
3.3.1	Home Screen	22

		2
	3.3.2	Buttons 22
	3.3.3	Toolbar Menu 24
	3.3.4	Create Submission Screen 24
	3.3.5	Adding Pictures Screen 25
	3.3.6	Past Submissions Screen 26
	3.3.7	View Submissions Screen 26
	3.3.8	Settings Screen 26
	3.3.9	Instructions Screen 27
3.4	Database	28
	3.4.1	Overview 28
	3.4.2	Data Dictionary 28
	3.4.3	Connection between Databases 31
3.5	Website (Stretch Goal)	31
	3.5.1	Users 31
	3.5.2	Registration 31
	3.5.3	Login 31
	3.5.4	Navigation 31
	3.5.5	Submissions 32
	3.5.6	Messaging 32
	3.5.7	Account 32
	3.5.8	About 32
3.6	Project Timeline - Alpha Level Release (Week 6, Winter Term)	32
	3.6.1	Overview 32
	3.6.2	App Pages 32
3.7	Project Timeline - Beta Level Release (Finals Week, Winter Term)	33
	3.7.1	Overview 33
	3.7.2	Create Submission Screen 33
	3.7.3	Add Picture Screen 33
	3.7.4	Submissions Screen 34
	3.7.5	Detailed Submission Page 34
	3.7.6	User Login 34
	3.7.7	Drafts 34
	3.7.8	Instructions Screen 34
	3.7.9	Add Samples Screen 34
3.8	Project Timeline - Engineering Expo (Week 6, Spring Term)	35
	3.8.1	Overview 35
	3.8.2	Toolbar menu 35
	3.8.3	Create Submission Screen 35
	3.8.4	Adding Pictures Screen 35
3.9	Conclusion	35

3.10	Glossary	35
4	Tech Review - Brandon Jolly	36
4.1	Introduction	36
4.2	Database Selection	36
4.2.1	MySQL	36
4.2.2	MongoDB	36
4.2.3	SQL Server	37
4.3	Database Visuals	37
4.3.1	MySQL WorkBench	37
4.3.2	PHPMyAdmin	37
4.3.3	Console	38
4.4	Web Frameworks	38
4.4.1	CakePHP	38
4.4.2	Laravel	38
4.4.3	Phalcon	38
4.5	Conclusion	39
5	Tech Review - Katherine Jeffrey	39
5.1	Introduction	39
5.2	IDE	39
5.2.1	Visual Studio	39
5.2.2	Android Studio	39
5.2.3	Eclipse	40
5.3	Data Transfer	40
5.3.1	SOAP	40
5.3.2	REST	40
5.3.3	JSON	41
5.4	Image Quality Analysis	41
5.4.1	Android Camera	41
5.4.2	Xamarin	41
5.4.3	Mitek MiSnap	41
5.5	Conclusion	42
6	Tech Review - Bradford Wong	42
6.1	Introduction	42
6.1.1	Role	42
6.1.2	What the Team is Trying to Accomplish	42
6.2	Mobile Application Frameworks	42
6.2.1	React Native	42
6.2.2	Flutter	43

6.2.3	Native Android	43
6.2.4	Overall Recommendation and Conclusion	43
6.3	Automated Testing	43
6.3.1	Robotium	44
6.3.2	Appium	44
6.3.3	UI Automator	44
6.3.4	Overall Recommendation and Conclusion	44
6.4	Languages	45
6.4.1	Java	45
6.4.2	Kotlin	45
6.4.3	LUA	45
6.4.4	Overall Recommendation and Conclusion	45
6.5	Conclusion	46
7	Weekly Blog Posts	46
7.1	Brandon Jolly	46
7.1.1	Fall	46
7.1.2	Winter	47
7.1.3	Spring	48
7.2	Katherine Jeffrey	48
7.2.1	Fall	48
7.2.2	Winter	49
7.2.3	Spring	51
7.3	Bradford Wong	51
7.3.1	Fall	51
7.3.2	Winter	52
7.3.3	Spring	53
8	Final Poster	54
9	Project Documentation	54
9.1	How Does the Project Work?	54
9.2	How to Install the Software	55
9.3	How to Run the Project	55
9.4	Hardware Requirements	55
9.5	API Documentation	55
10	Build Instructions/User Guide	56
10.1	Install Android App with APK	56
10.2	Alternative: Set Up Android Studio	56
10.3	Clone the GitHub Repository (can be found here: https://github.com/jeffreykat/MyVetPath)	56

		5
10.4	Open the Project	57
10.5	Setup MySql Server	57
11	Recommended Technical Resources	58
11.1	Helpful Websites	58
11.2	Helpful People on Campus	58
12	Conclusions and Reflections	58
12.1	Brandon Jolly	58
12.2	Katherine Jeffrey	59
12.3	Bradford Wong	59
13	Appendix 1: Essential Code Listings	61
14	Appendix 2	66
	References	66

1 INTRODUCTION TO PROJECT

1.1 The Project

The project was requested by Dr. Christiane Loehr. The project was requested because the Oregon Veterinary Diagnostic Laboratory wanted a mobile application in order to improve remote diagnostics. This project is significant because it solves the OVDL's problem where field personnel couldn't effectively communicate with personnel in the lab, especially in areas with limited internet connectivity.

1.2 The Clients

The clients were Dr. Christiane Loehr of the Oregon Veterinary Diagnostic Laboratory and Bill Baxter of WHB Consulting. Dr. Loehr told the team about her idea for the project and what the requirements were. She also gave suggestions on how to structure the databases and provided the team use cases. Lastly, she gave the team props for the team to use in the booth during Engineering Expo.

Bill Baxter assisted the team by meeting with the team each week during winter term. During these meetings, he answered any questions the team had. He was also responsible for setting up the server and developing the API.

1.3 The Team

Brandon Jolly developed the SQLite and MySQL databases. Katherine Jeffrey and Bradford Wong developed the Android application.

2 REQUIREMENTS DOCUMENT

2.1 Change Log

Section	Original	New
System Functions	<ul style="list-style-type: none"> • Can create field reports that can contain pictures and text information such as date, location, and patient • When viewing a report, a user can write messages that will be sent to the other users involved in the report • Take pictures using Android device's camera • Can write additional text details using drop-down menus in the application • When connected to the internet, the user who created the report can send it to another user • When not connected to the internet, the application will send the report as soon as a connection is established • Sending a report automatically updates the MySQL database • The MySQL database will have a web API Interface • A SQLite database for native storage on the phone • Both databases will be expandable and searchable • Only users with the proper credentials can use the application • Only the users who created and received the report can view the report • Stretch Goal: Provide instant feedback on image quality for each picture taken 	<ul style="list-style-type: none"> • Now uses the word "submission" instead of "report". • Now any user can use the application, but only users with proper credentials can send a submission through the application.
Functional Requirements	<p>When field personnel need to send reports to the lab, they can fill out a form on the app which will have menus and prompts as well as a quota for images and text entry. The app needs to receive messages and feedback about reports from people in the lab. They might need to request more information or give instructions regarding next steps in the diagnostic process. The app will need to ensure all needed information has been included in each report to minimize requests for additional or forgotten information. Users will need to log in to the app to access stored information and to attach their credentials to any reports they send. This will give them permission to see previous reports and messages sent to them from the lab. They should be able to log out through the menu or settings screen.</p> <p>Users in the lab will be able to log into the database and search its contents for past reports. They will be doing this using a web interface to allow multiple user to access the database without having to search using SQL. Through this web interface, users will be add comments to reports and then send them back to the user.</p> <p>The app needs to receive messages and feedback about reports from people in the lab.</p>	<ul style="list-style-type: none"> • Now uses the word "submission" instead of "report". • Clarified that the website is a stretch goal.

Section	Original	New
User Characteristics	This Android application will be used by staff members and clients of the Oregon Veterinary Diagnostics Laboratory. There will be people out in the field who are using the application to create and share reports. In addition, there will be users in the laboratory who will use the application to view reports and send feedback to the field personnel.	<ul style="list-style-type: none"> Now uses the word "submission" instead of "report".
User Interface	Users will need to log in to the app to access stored information and to attach their credentials to any reports they send. This will give them permission to see previous reports and messages sent to them from the lab. They should be able to log out through the menu or settings screen. When field personnel need to send reports to the lab they can fill out a form on the app which will have menus and prompts as well as a quota for images and text entry.	<ul style="list-style-type: none"> Now uses the word "submission" instead of "report".
Software Interfaces	Each report will be assigned a case number so it can be easily found again by querying the database from the app or the interface in the lab. The lab interface is yet to be determined. The app could send the field reports and the lab could review the reports using a web interface, but that might be out of the scope of the project and is a stretch goal.	<ul style="list-style-type: none"> Now uses the word "submission" instead of "report". Clarified that the web interface is a stretch goal.
Information Management	Users will need to log in to the app to access stored information and to attach their credentials to any reports they send. This will give them permission to see previous reports and messages sent to them from the lab.	<ul style="list-style-type: none"> Now uses the word "submission" instead of "report". Clarified that the web interface is a stretch goal.
Information Management	Users will need to log in to the app to access stored information and to attach their credentials to any reports they send. This will give them permission to see previous reports and messages sent to them from the lab.	<ul style="list-style-type: none"> Now uses the word "submission" instead of "report". Clarified that the web interface is a stretch goal.
Field Tests	The delivered app should be able to collect the data and images, send them to the lab through a database, and receive messages from the lab. It will need to pass a field test to prove it works and all the features are correctly implemented.	Clarified that the app will be able to interact with an API.

2.2 System Purpose

This mobile application is intended to serve as a means of communication between personnel in the field and pathologists in the laboratory. Its purpose is to improve a team's ability to perform remote diagnostics by providing a convenient way for teams to communicate information.

2.3 System Overview

2.3.1 System Context

The Oregon Veterinary Diagnostic Laboratory (OVDL) wants to create an efficient way for field personnel and lab staff to communicate for remote diagnostics. A lack of communication can make remote diagnostics difficult because the pathologists in the laboratories can't analyze the sample and make decisions about sample processing until the sample is back in the laboratory. This can prove problematic if the pathologist decides that further action is needed. For example, they may decide that they require additional samples and data, but the field personnel may not still be in the field or the original specimen may no longer be available when this decision is made.

2.3.2 System Functions

- Can create field submissions that can contain pictures and text information such as date, location, and patient
- When viewing a report, a user can write messages that will be sent to the other users involved in the report
- Take pictures using Android device's camera
- Can write additional text details using drop-down menus in the application
- When connected to the internet, the user who created the report can send it to another user
- When not connected to the internet, the application will send the report as soon as a connection is established
- Sending a report automatically updates the MySQL database
- The MySQL database will be able to use a web API
- A SQLite database for native storage on the phone
- Both databases will be expandable and searchable
- Only users with the proper credentials can send submissions through the application
- Only the users who created and received the report can view the report
- Stretch Goal: Provide instant feedback on image quality for each picture taken

2.3.3 User Characteristics

This Android application will be used by staff members and clients of the Oregon Veterinary Diagnostic Laboratory. There will be people out in the field who are using the application to create and share submissions. In addition, there will be users in the laboratory who will use the application to view submissions and send feedback to the field personnel.

2.4 Definitions

TABLE 1: Definitions

Term	Definition
Necropsy	Autopsies of non-human species
Pathology	The study of the causes and effects of diseases, especially the branch of medicine that deals with the laboratory examination of samples of body tissue for diagnostic or forensic purposes

2.5 Requirements

2.5.1 Functional Requirements

The OVDL wants a native Android mobile application that collects field data and images, stores the information on a native SQLite database, sends them to the lab's MySQL database, and gets real time feedback from the lab. When field

personnel need to send reports to the lab, they can fill out a form on the app which will have menus and prompts as well as a quota for images and text entry.

The app needs to receive messages and feedback about submissions from people in the lab. They might need to request more information or give instructions regarding next steps in the diagnostic process. The app will need to ensure all needed information has been included in each report to minimize requests for additional or forgotten information.

Users will need to log in to the app to access stored information and to attach their credentials to any submissions they send. This will give them permission to see previous submissions and messages sent to them from the lab. They should be able to log out through the menu or settings screen.

Users in the lab will be able to log into the database and search its contents for past submissions. They will be doing this using a web interface (stretch goal) to allow multiple user to access the database without having to search using SQL. Through this web interface, users will be able to add comments to submissions and then send them back to the user.

2.5.2 Usability Requirements

A stretch goal for the app is having a feature that analyzes the images taken using the phone's built-in camera and gives the user feedback on image quality. It should only send images that are well lit, don't have shadows or reflections, and clearly show the subject. If there is time we will try to implement it because it would be beneficial for the client.

2.5.3 Performance Requirements

Some of the remote locations might not have a stable internet or cellular connection so the data will need to be stored until a connection can be established. Otherwise the data is stored locally on the app.

2.6 Interfaces

2.6.1 User Interface

The User Interface will be the Android app which should be easy to navigate for a user with only a basic knowledge of the app's functions.

Users will need to log in to the app to access stored information and to attach their credentials to any submission they send. This will give them permission to see previous submissions and messages sent to them from the lab. They should be able to log out through the menu or settings screen. When field personnel need to send submissions to the lab, they can fill out a form on the app which will have menus and prompts as well as a quota for images and text entry.

2.6.2 Software Interfaces

A SQLite database must be able to store data and images from field users. The lab will review these images by using API which is connected to a MySQL Server. Each report will be assigned a case number so it can be easily found again by querying the database from the app or the interface in the lab. The lab interface is yet to be determined. The app could send the field reports and the lab could review the reports using a web interface, but that is a stretch goal.

2.7 System Modes and States

2.7.1 Network Connection

Some of the remote locations might not have a stable internet or cellular connection so the data will need to be stored until a connection can be established. Once the connection is made the data should be uploaded and the lab personnel should be alerted.

2.7.2 Database Synchronization

The app must work with and without an internet connection and be able to store data until an internet connection is made so the data can be sent to the database. An optional feature is allowing the user to decide if they want the data sent automatically when the phone connects to the internet or if they want to manually send the data once they find an internet connection.

2.8 System Security

The application must adhere to the OVDL's information security needs and keep track of users account permissions.

2.8.1 Information Management

The field personnel should only be able to access the data they gathered and the information sent to them by the lab, not data gathered by other field personnel. There is a hierarchy of permissions for people working in the lab as well and they must only have access to the data they are allowed to see. Users will need to log in to the app to access stored information and to attach their credentials to any submissions they send. This will give them permission to see previous submissions and messages sent to them from the lab. They should be able to log out through the menu or settings screen.

2.8.2 Policies and Regulations

We must get permission before using any of OSU's logos or images.

2.9 Verification

2.9.1 Debugging

The app's code will be consistently debugged by Android Studio's debugger.

2.9.2 Field Tests

The delivered app should be able to collect the data and images, and save them in the phone's local storage with a SQLite database. It should also be able to interact with an API that will be used to transfer data between the phone and the server. It will need to pass a field test to prove it works and all the features are correctly implemented.

2.10 Appendices

2.10.1 Assumptions and Dependencies

- We are assuming users will be English speakers and have some experience using Android phones.

2.10.2 Acronyms and Abbreviations

- OVDL - Oregon Veterinary Diagnostic Laboratory
- OSU - Oregon State University

2.11 Gantt Chart

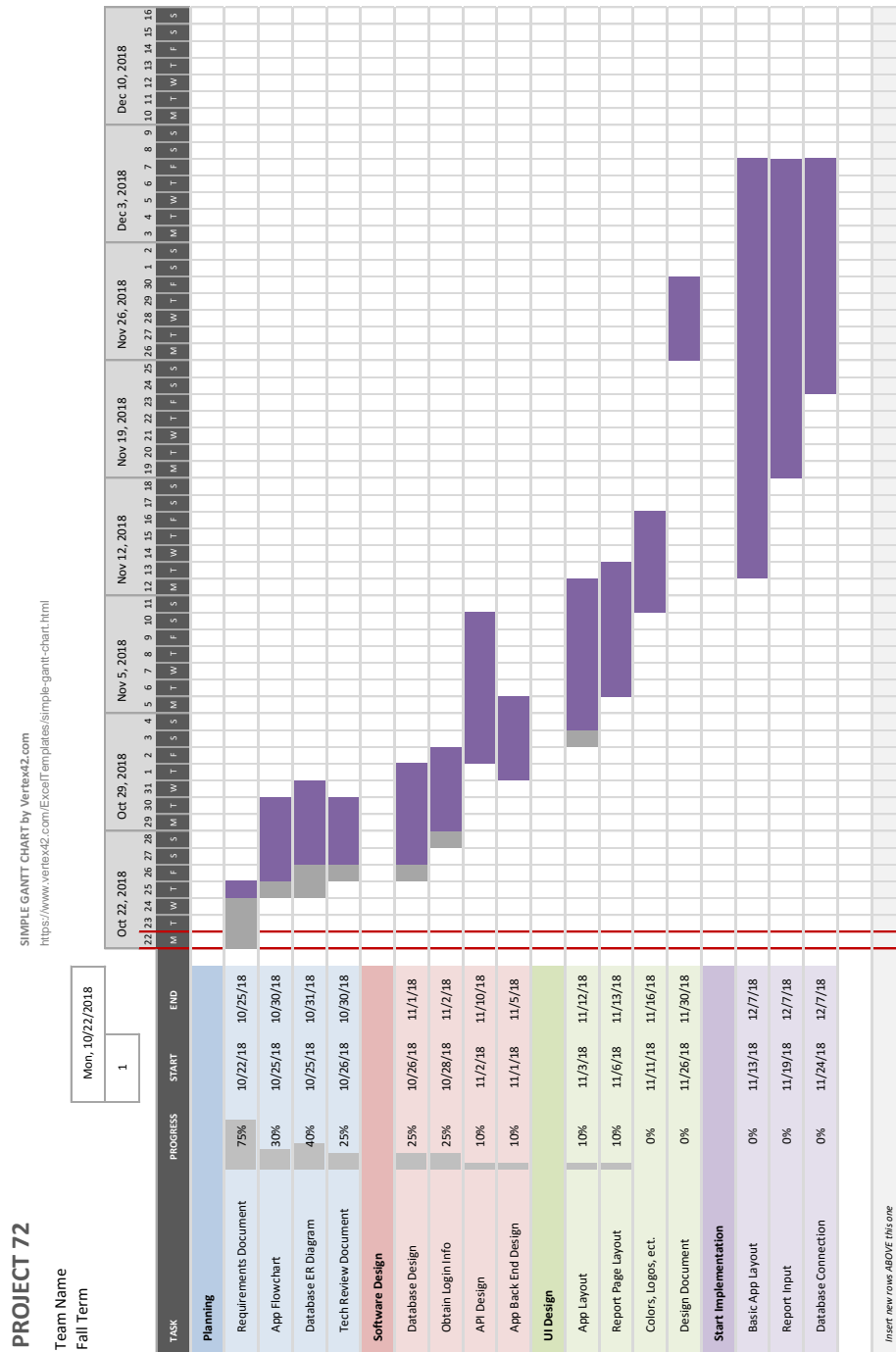


Fig. 1: Fall Term 2018

3 DESIGN DOCUMENT

3.1 Change Log

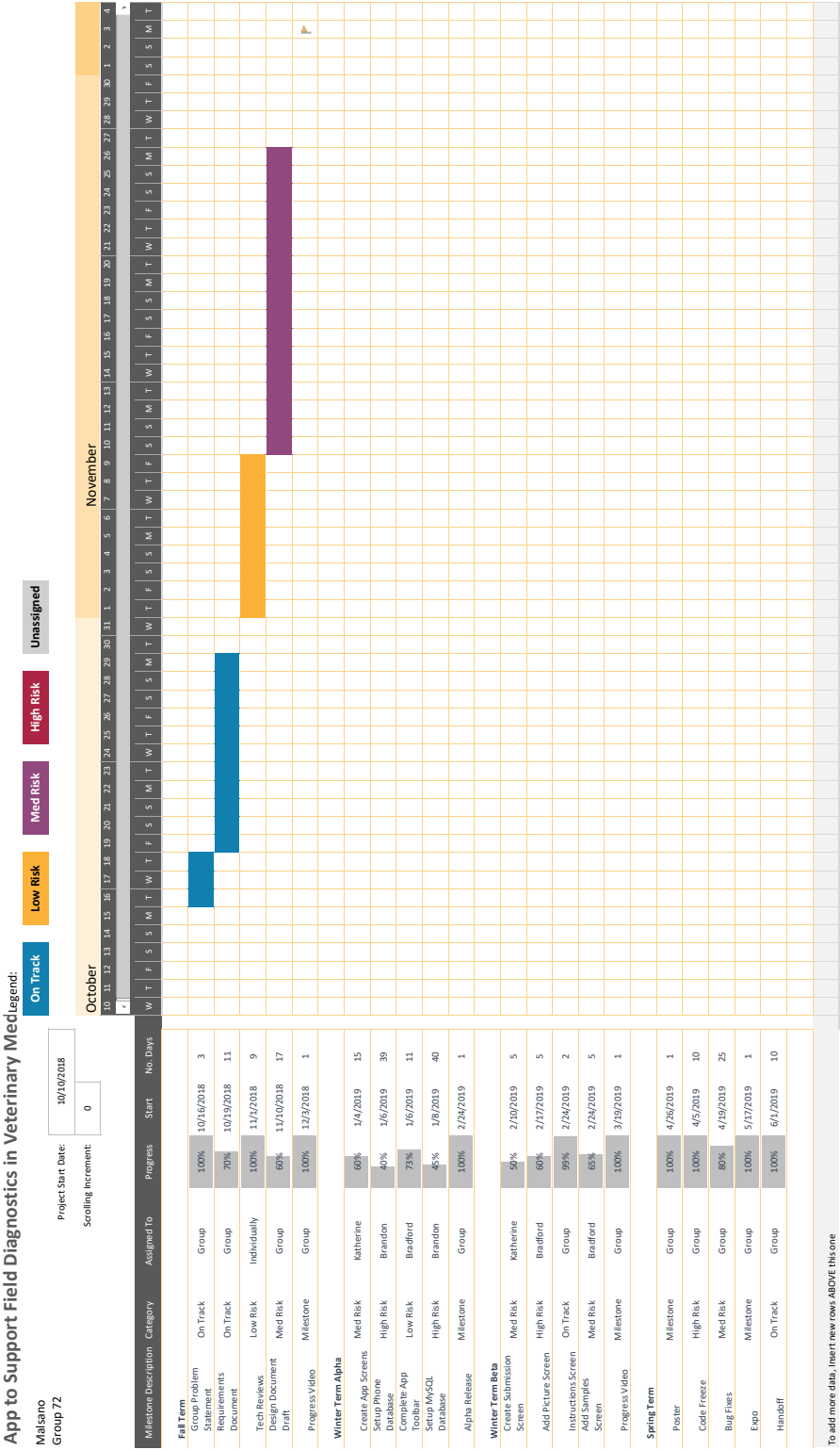


Fig. 2: Final Gantt Chart Fall Term

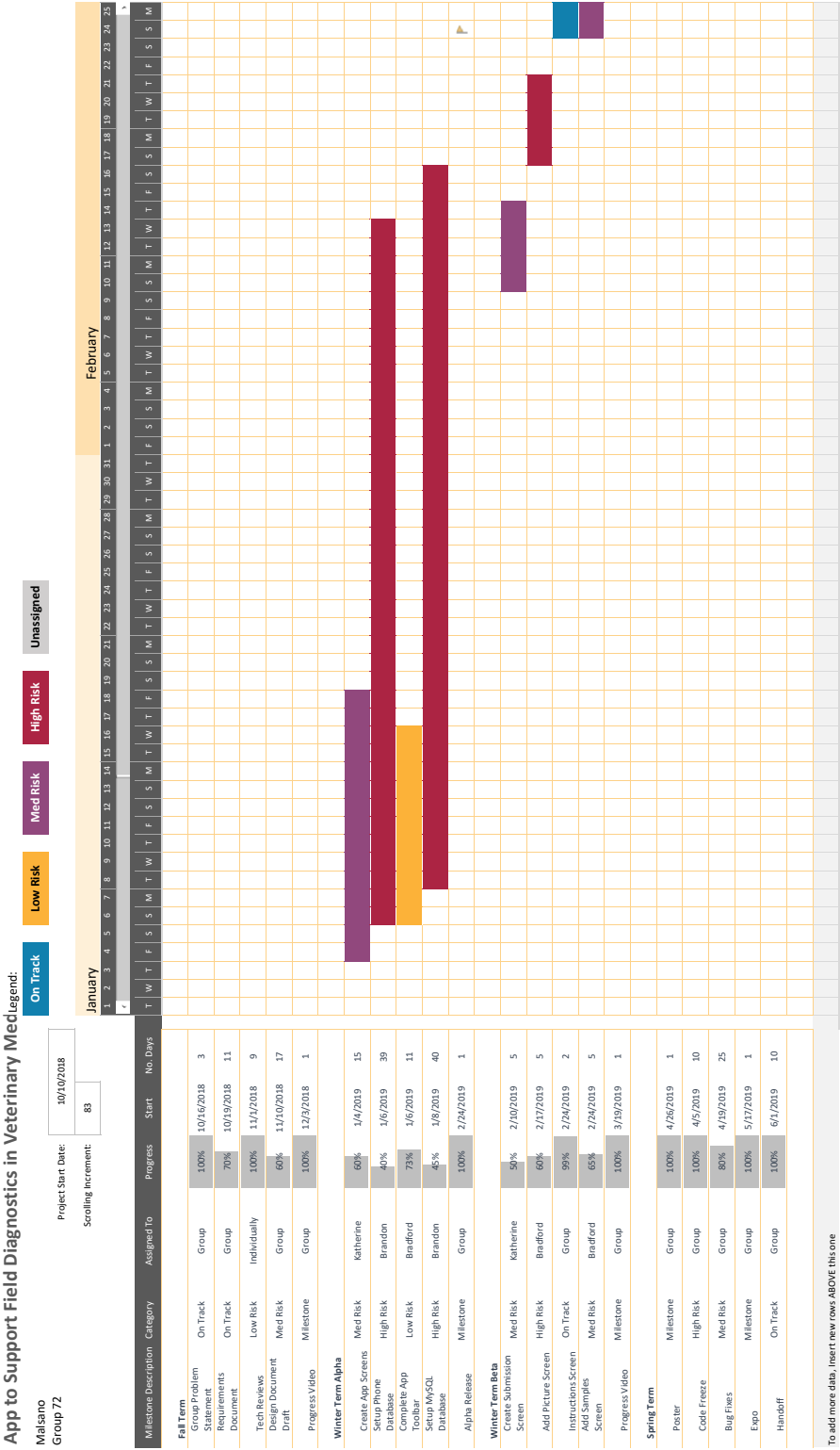


Fig. 3: Final Gantt Chart Winter Term Alpha

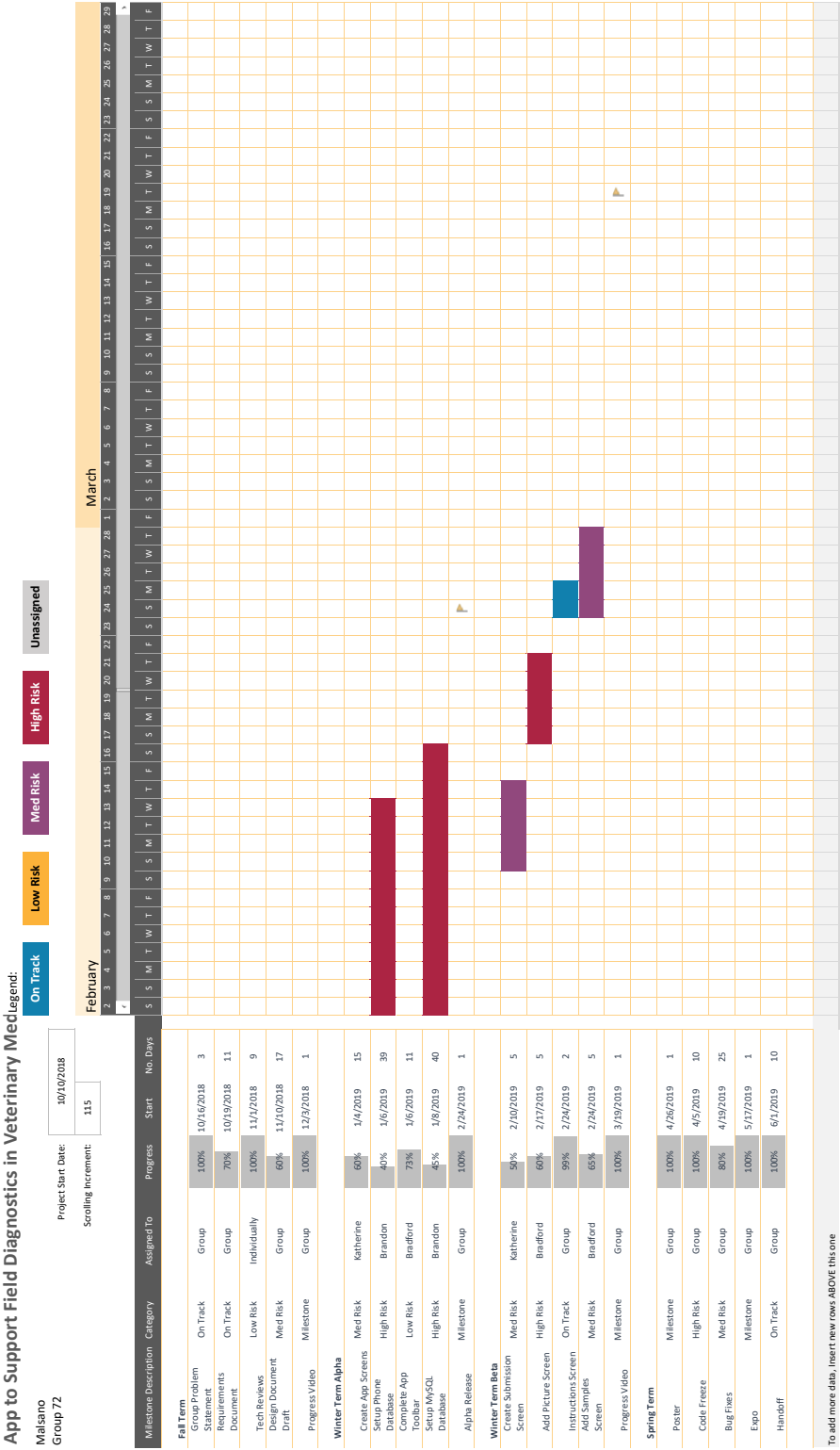


Fig. 4: Final Gantt Chart Winter Term Beta

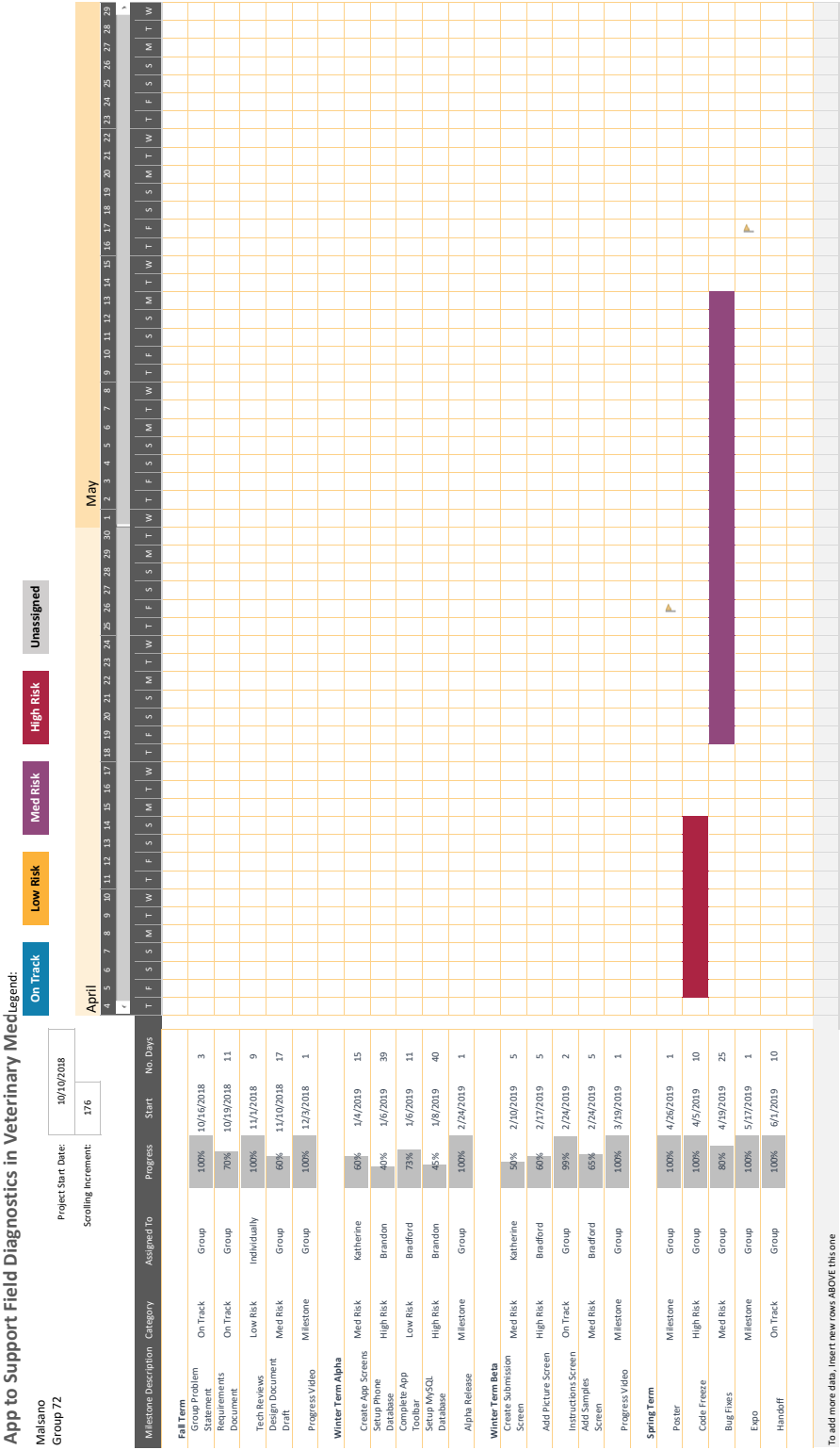


Fig. 5: Final Gantt Chart Spring Term

Section	Original	New
Home Screen	There will be four NavButtons	<ul style="list-style-type: none"> Replaced "NavButtons" with "buttons"
Button	3.2 NavButton	<ul style="list-style-type: none"> Changed section item to "Buttons"
Button	This custom class will inherit from the Button class.	<ul style="list-style-type: none"> Deleted text and explained that there will need to be onClick listeners.
Button	This class should also have getter and setter functions for each of the attributes.	<ul style="list-style-type: none"> Replaced text with explanation that said the buttons will be added in the screen's layout XML file.
Past Submissions Screen	Each row will also have the Case ID in green text if it was successfully sent to the server.	<ul style="list-style-type: none"> Changed it with explanation saying that each row will say when the submission was sent to the server. Added text saying that if the submission wasn't sent yet, then it will say "pending".
View Submissions Screen	The pictures of the submission will show up in a GridLayout. The pictures will initially just have a camera icon. Clicking on the icon will actually downloading the image.	<ul style="list-style-type: none"> Deleted text and gave updated description of what information will be provided.
Settings Screen	There will be a login button that will prompt the user with a customized AlertDialog for their username and password to save their credentials to the phones local storage.	<ul style="list-style-type: none"> Now says that clicking on a button will go to a different screen instead of having an AlertDialog.
Add Samples Screen	This section did not originally exist.	<ul style="list-style-type: none"> Added description of the Samples Screen
Create Submission Screen	The icon will also have a badge, which will say the number of pictures that were added to the submission.	<ul style="list-style-type: none"> Deleted part about badge. Added description about "Add Samples" button.
Create Submission Screen	There will be a combination of text fields and dropdown menus.	<ul style="list-style-type: none"> Added calendar fragments to the list

Section	Original	New
App Pages	Each entry should be clickable and clicking on an entry will take the user to a detailed page of that submission, but navigating to a submission's page won't be implemented in time for the Alpha.	<ul style="list-style-type: none"> Removed part about the Alpha.
App Pages	Submissions and Server Database - After creating a submission, the submission will automatically be sent to the server if there is a stable internet connection. If there isn't stable server connection, it will just be stored locally on the phone. Once there is an internet connection, the submissions will automatically be sent to the server.	<ul style="list-style-type: none"> This item was removed.
Beta Level Release (Finals Week, Winter Term)	At this point, a user will be able to create a submission and successfully send it to the server.	<ul style="list-style-type: none"> Changed it so it says that submission is stored into the SQLite database instead of the server
Web Application	<ul style="list-style-type: none"> Users will be able to register a new account Users will be able to login Users will be able to see data on the database Users will be able to see submissions that were created and sent to them Users will be able to create comments on specific submissions 	<ul style="list-style-type: none"> This section was removed.
Drafts	When a user is creating a submission, they will have the option to save their current submission as a draft and won't necessarily submit it. There will be a button at the bottom of the "Create Submission" screen that says "Save Draft". Clicking on it will automatically save the draft and return the user to the home page. Additionally, the home page will have a button called "View Drafts", and clicking on this button will take the user to a page that shows all the user's saved drafts. The drafts will be listed in a table similar to what is shown in the "View Submissions" screen. Clicking on a draft will take the user to the "Create Submission" page where all of the information saved in the draft is automatically filled out for the user.	<ul style="list-style-type: none"> This section was moved to the Beta Level Release Section (now section 6.1.3).

Section	Original	New
Settings Page (6.3.5)	<p>This page will have two options that the user can click on ("Manage account" and "Delete App").</p> <ul style="list-style-type: none"> • Manage Account - Clicking on this link will send the user to the website • Delete App - Deletes app 	<ul style="list-style-type: none"> • This was removed.
Instructions Screen (6.3.6)	<p>This page will have four sections (Registering a New User, Creating a Submission, Sending a Submission, and App Usage) that have information underneath them detailing how to do each process</p> <ul style="list-style-type: none"> • Registering a New User - Provides a link to the web page that allows users to create an account • Creating a Submission - Details how to navigate to the page to create a report and how drafts work • Sending a Submission - Explains that clicking on the "Submit" button at the bottom of a Submission will send it to the server if there is an internet connection. If there isn't an internet connection, then it will automatically send when the user regains a connection • App Instructions - Provides a broad explanation on how to use the application 	<ul style="list-style-type: none"> • This section was moved to the Beta Level Release section of the timeline.
App Pages	<p>The toolbar will have white text and a background color that depends on what page the user is in (blue for "Create Report", red for "View Past Reports", and green for "View Drafts").</p>	<ul style="list-style-type: none"> • Deleted part about the background color.

Section	Original	New
App Pages	On all pages but the home screen, there will be back arrow on the left of the header bar.	<ul style="list-style-type: none"> • Added "Add Pictures" screen and "Add Samples" screen to list. • Added description about back arrow
6.2.2 Create Submission Screen	<p>All of the missing fields will be added. The following information will be collected on this screen:</p> <ul style="list-style-type: none"> • Group Name - Input using text field • Is this research? - Input using checkbox • VDL Account number - Input using number field • Has the user's information changed - Input using checkbox • Submitter's information <ul style="list-style-type: none"> - Submitter - Address - Input using text field - City - Input using text field - State - Input using text field - Zip - Input using text field - Phone - Input using text field - Fax - Input using text field - Email - Input using text field - Submitting Veterinarian - Input using text field • The Owner's information (if other than submitter) <ul style="list-style-type: none"> - Owner - Address - Input using text field - City - Input using text field - State - Input using text field - Zip - Input using text field - Phone - Input using text field - Previous Accessions - Input using text field - Copy Results To - Input using text field - Email/Fax 	<ul style="list-style-type: none"> • All of this was removed.

Section	Original	New
6.2.2 Create Sub-mission Screen	<ul style="list-style-type: none"> • Animal Identification <ul style="list-style-type: none"> – Number - Automatically provided by app – Name/identifier No. - Input using text field – Species - Input using text field – Breed - Input using text field – Sex - Input using text field – Date of Birth - Input using text/date field – Date Specimens taken - Input using text field – Date Specimens Submitted - Input using text field – Number of each sample type <ul style="list-style-type: none"> * Whole Animal - Input using number field * Fresh Tissue - Input using number field * Formalin Fixed Tissue - Input using number field * Blood, whole - Input using number field * Serum - Input using number field * Plasma - Input using number field * Milk - Input using number field * Urine - Input using number field * Voided - Input using checkbox * Catheterized - Input using checkbox * Cystocentesis - Input using checkbox * Fluid (and origin information) - Input using number field for sample number and text field for origin information * Swab (and origin information) - Input using number field for sample number and text field for origin information * Other (and origin information) - Input using number field for sample number and text field for origin information – Histopathology on Biopsy (and source information) - Input using checkbox and text field for source information – Necropsy only - Input using checkbox – Necropsy with Histology - Input using checkbox – Necropsy and Complete Diagnostic Work up - Input using checkbox – Other - Input using checkbox and textfield • Care of Remains (Small animals only) <ul style="list-style-type: none"> – Routine Disposal - Input using checkbox – Cremation (Specify Company) - Input using checkbox • A clickable button that navigates the user to a screen that will allow them to input pictures. The details of this page will be outlined in a later section. • A clickable button that says "Save Draft". • A clickable button that says "Submit". Clicking on this button will finish the submission and save its details to the phone's local database. 	<ul style="list-style-type: none"> • Replaced this with an updated list of what is being collected and how the data is being collected. • Added description about "Add Pictures", "Save Draft", "Add Samples", and "Submit" button.

3.2 Introduction

The completed project components are intended to serve as a means of communication between personnel in the field and pathologists in the laboratory. Its purpose is to improve a team's ability to perform remote diagnostics by providing a convenient way for teams to communicate information. The OVDL wants a native Android mobile application that collects field data and images, stores the information on a native SQLite database, sends them to the lab's MySQL database, and gets real time feedback from the lab. The lab will interact with the database and user's submissions through the website.

3.3 Android Application

3.3.1 Home Screen

There will be four buttons (described later). Clicking on each button will take the user to the corresponding page. The NavButtons will be contained in a vertical linear layout. Here is a list of each button and what their attributes will be:

- A button with an image of a persons shape for the image, light green for the background color, and "Create Account" for the text. Clicking on this button will open up the web browser and take the user to the registration page
- A button with a plus icon for the image, blue for the background color, and "Create submission" for the text
- A button with a closed folder icon for the image, red for the background color, and "View submission" for the text
- A button with an icon an open folder for the image, dark green for the background color, and "View Drafts" for the text

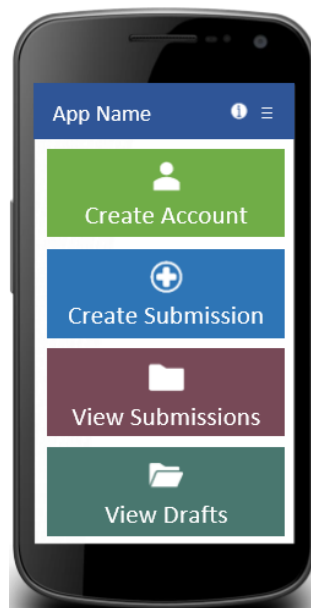


Fig. 6: Home Screen Mockup

3.3.2 Buttons

These buttons must have onClick listeners that will take the user to the appropriate page once clicked. Their attributes should include an image, background color, and text. All the text color will be set by default to white. These buttons will be added in the screen's layout XML file.

Section	Original	New
6.2.9 Add Samples Screen	This section didn't exist before.	<ul style="list-style-type: none"> Added description about the screen and how data is being collected.
Database	Table SickElement	Now called Patient
Database	Column in Table SickElement called SickElementName	Now Called PatientName
Database	Replies For submission table will connect Submission and Replies	This table has been dropped and the columns from the table are moved to replies
Database	Client table will hold the information for clients, Pathologist table will hold information pertaining to a Pathologist	These tables have been dropped and the columns have been moved to the user table
Database	There was none previously	Added column Submission Review to the Report Table. This column will hold the initial review a Pathologist has regarding a submission.
Database	There was none previously	Added the Date Closed Column to the Report table. This column will hold the data of when a report has been closed
Database	The tables will have the naming convention of ALL CAPS	The tables have now been changed to have the naming convention of Camel Case
Database	The tables primary keys will have the naming convention of camel case	The Primary keys will now have the naming convention primary_ID
Database	The Image table	<ul style="list-style-type: none"> This has been renamed as the Picture Table The column imagePath has been renamed picturePath The column image_ID has been renamed Picture_ID
Database	The User table will have a column called Authorized	The column has been renamed Authenticate.
Database	The table submission table will have a primary key of internal_id	It has now been changed to master_Id and the foreign keys have reflected that.
Glossary	There was non previously.	Added, "Reply - A message sent between the user and the client" to the glossary.
Settings	The settings screen will have a logout button that will remove the users account credentials from the phones local storage. They will need to authenticate their account again before they can save or send a submission.	Clarified that users only have to log-in whenever they send data to the server. They can save data locally without having to authenticate.
Connection between Databases	Once the new ID is given to the submission and all of the connections to the submission are updated, then the submission will be added to the MySQL database. This process is done by turning the submission and its components would then be used by an API to connect to the database to upload the information into storage.	<ul style="list-style-type: none"> Replaced "new ID" with "master_ID". Reworded last sentence

3.3.3 Toolbar Menu

This will use the Toolbar class. Every page will have a toolbar, and the text in the toolbar will be the same as the current screen's name. There will be an overflow menu on the right of the toolbar with the following options: "Home", "Create submission", "View Submissions", "Instructions", and "Settings". Clicking on one option will take the user to the corresponding page.

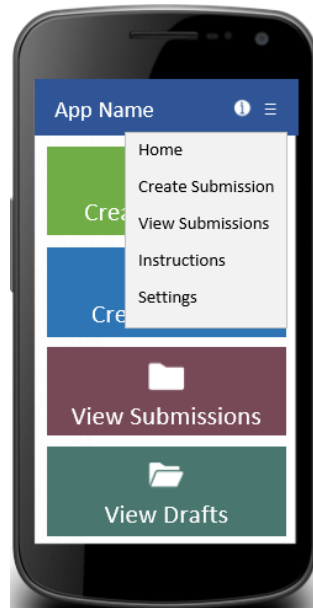


Fig. 7: Menu Screen Mockup

3.3.4 Create Submission Screen

There will be a combination of text fields and dropdown menus. The Spinner class will be used for the drop down menus and the EditText class will be used for text fields. There will be a button that says "Submit". Clicking on this button will create a submission in the phone's database and save all the submitted information. It will then try to send the submission to the database if there is an internet connection. There will also be a button that says "Save Draft", which will store the current draft with all its filled out information on the phone's database. There will also be a button with a camera icon on it. Clicking on it will take the user to the page where they can add and remove pictures to the submission. The icon will also have a badge, which will say the number of pictures that were added to the submission.

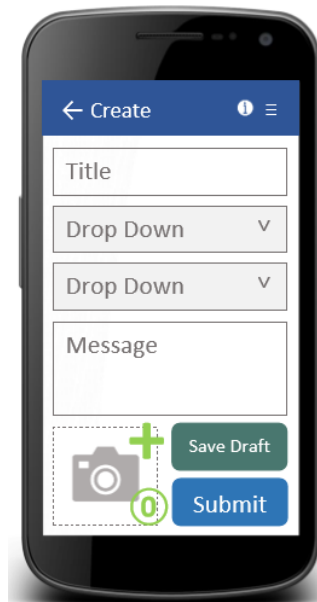


Fig. 8: Create Screen Mockup

3.3.5 Adding Pictures Screen

This screen will use a grid layout. Each element will be of the ImageButton class. If a picture hasn't been added to a cell, then the cell's image will be a camera icon. If a picture has already been added, then the image will be the actual picture that was added. There will also be a floating, circular button on the bottom right of the screen with a checkmark icon. It will be of the FloatingActionButton class. Clicking on it will save the images and return the user back to the "Create" screen. The grid will have the following properties:

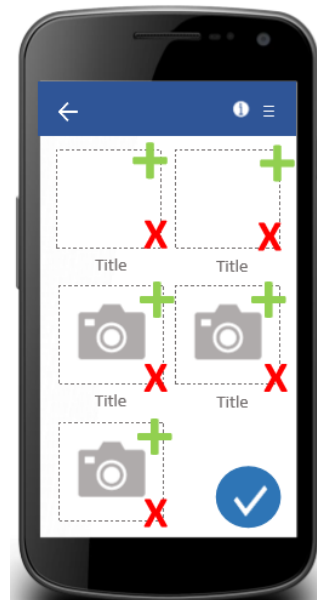


Fig. 9: Picture Screen Mockup

- The grid will have two total columns

- The first column will have three elements
- The second column will have two elements

3.3.6 Past Submissions Screen

This screen will have a ListView with a Custom Adapter in order to populate the rows. Each row will have the title of the submission and the date it was created. Each row will say when the submission was sent to the server. If the submission hasn't been sent yet, then it will say "pending". Clicking on an entry will navigate the user to the detailed page of that submission.

3.3.7 View Submissions Screen

This screen will have a vertical LinearLayout where all of the information in the submission is added to the layout. This screen will show all content relevant to the submission such as the title, comment, samples, patient information, pictures, and replies.

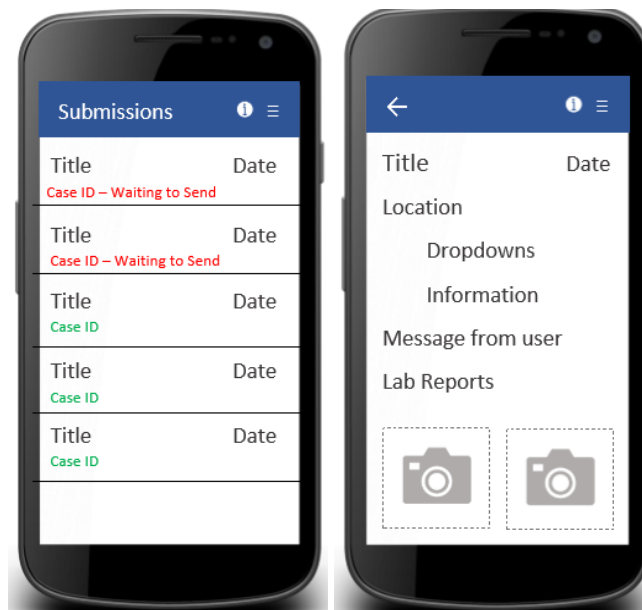


Fig. 10: Submissions and Details Screen Mockups

3.3.8 Settings Screen

The settings screen will have a logout button that will remove the users account credentials from the phones local storage. They will need to authenticate their account again before they can send a submission. There will be a login button that the user can click on to go to a different screen and input their login credentials. These credentials will be saved to the phone's local storage. To do this they must have an internet connection so their credentials can be matched in the database.



Fig. 11: Settings Screen Mockup

3.3.9 Instructions Screen

The instructions screen will have lists of steps explaining how to use the app, how to register an account, how to create a submission, and how to send messages attached to the submissions. There will also be a link to the website and contact information for the OVDL.

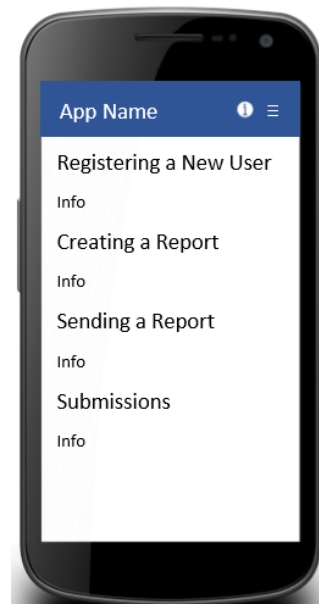


Fig. 12: Instructions Screen Mockup

3.4 Database

3.4.1 Overview

This project will utilize two databases using the structure presented in the figure below. One database is a MySQL database which will be storing the majority of the information for the app. This will be stored on an apache server and the information will be accessed using a web interface. Our second database is SQLite. The reason for this second database is to allow field workers to store information on the app while not connected to the internet. Instead the database will be stored locally within the phone and will only contain the current user's information.

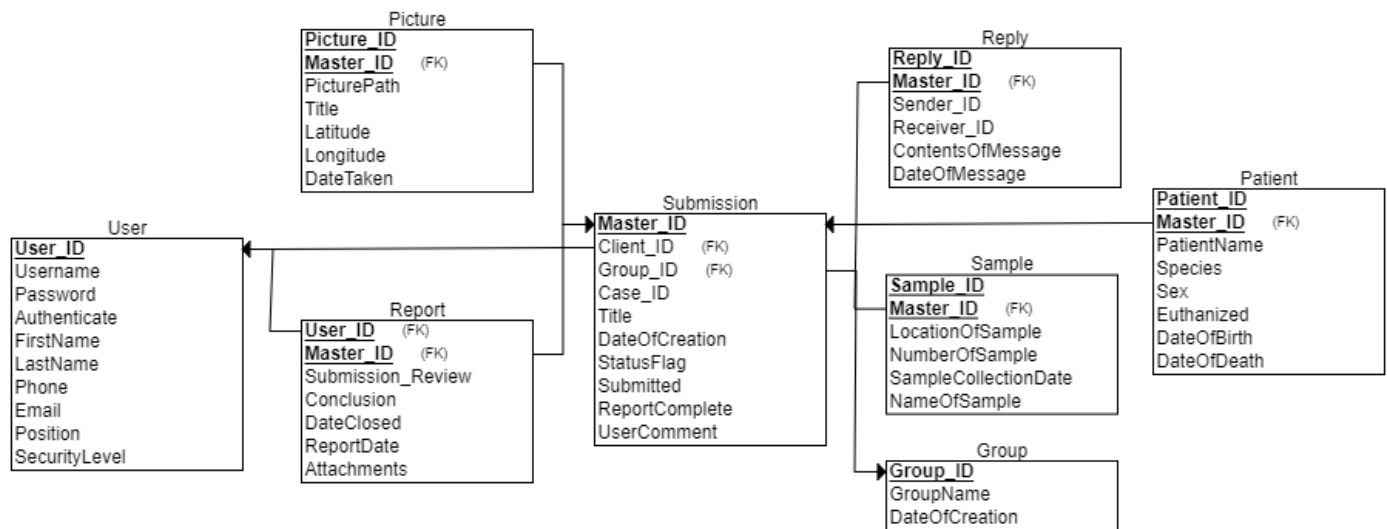


Fig. 13: ER Diagram 4/14/19

3.4.2 Data Dictionary

- User:

The User table will contain all the information for a user in our database. This includes their UserName, password, and their contact information. This table will also include the Position if they have one and if their securityLevel

- User_ID: The primary key which is a unique integer value generated using a running total. Created when a new user is added to the table.
- Username: A unique varchar(16) value created when a user creates an account using the web interface.
- Password: A varchar(16) value created when a user creates an account using the web interface.
- First Name: A varchar(30) value containing the user's first name.
- Last Name: A varchar(30) value containing the user's last name.
- Phone Number: A varchar(16) value containing the user's phone number in "XXX-XXX-XXXX" format.
- Email: A varchar(40) value containing the user's email address.
- Authorized: A binary value representing if a user has permission to modify reports, delete reports, etc.
- Position: A varchar(60) value containing what position a user holds. (Assistant, Director, etc.)
- Security Level: An interger value used to determine what permissions a user is able to use.

- Submission:

The submission table contains submissions a Client creates and then sends to a Pathologist to review.

- Master_ID: The primary key for the Submission table. It is created using a running total on the SQLite database.
- Client_ID: A foreign key from the Client table to determining which client has created the report.
- Case_ID: An integer number created by using the last two digits of the year and then a six digit running total. It's primary use is to allow a pathologist to easily recognizing what submission they are working on. May be dropped in the future when the Use of this app expands.
- Group_ID: A foreign key from the Group table which holds what group this submission is apart of. Defaulted to blank.
- Title: A varchar(255) value containing the title a Client Created. The title has to be unique for the client only.
- DateOfCreation: A datetime value representing the date the submission was first created. Formated as "YYYY/MM/DD HH:MM".
- StatusFlag: An integer value represented the stage of a submission. 0, for draft, 1 for submitted, 2 for review, 3 for closed.
- Submitted: A datetime value representing when the submission was sent to the MySQL database. Formated as "YYYY/MM/DD HH:MM".
- ReportComplete: A datetime value for when the submission has been closed and the reports for the submission have been finished. Formated as "YYYY/MM/DD HH:MM".
- UserComment: A varchar(255) value containing the comments a Client inputed when they where creating the submission.

- Picture:

The Picture table contains the images used in a submission. There could be many images which belong to the same submission. Also contains where the image was taken, when it was taken, and a Client created title.

- Picture_ID: The primary key for the image table. Created using a running total for the specific submission.
- Master_ID: A foreign key from the Submission table and a primary key in combination with Image ID.
- PicturePath: A varchar(255) containing the image file.
- Title: A varchar(255) containing a user created title. Only has to be unique to the submission.
- Latitude: A floating point value containing the latitude coordinates.
- Longitude: A floating ponit value containing the longitude coordinates.
- Date Taken: A datetime value containing when the picture was taken. Formated as "YYYY/MM/DD HH:MM".

- Patient:

The Patient table contains the information regarding the animal in a submission. Such as the animal's name, it's sex, etc.

- patient_ID: A primary key, an integer value which is generated with a running total on the SQLite database.
- Internal ID: A foreign key from the submission table. A primary key in combination with the patient_ID.

- patient_Name: A varchar(30) value contain a Client created name for the animal.
- Species: A varchar(30) value containing what species the animal is.
- Sex: A varchar(1) value containing the sex of the animal. M for male, and F, for female.
- Euthanized: A binary value representing if the animal is euthanized or not.
- DateOfBirth: A date value representing the birthday of the animal. Formated as "YYYY/MM/DD".
- DateOfDeath: A date value representing the date of death for the animal. Formated as "YYYY/MM/DD".

- Sample:

The Sample table contains the information associated with the sample of the submission.

- sample_ID: A primary key in the form of an integer value. Generated with a running total on the SQLite Database.
- master_ID: A foreign key from the Submission table. Also a primary key in combination with Sample ID.
- LocationOfSample: A varchar(255) value which contains the information of where the sample was taken from. For example Right leg, or blood from mouth.
- NumberOfSample: An integer representing the amount of a sample taken.
- SampleCollection Date: A date value representing when the sample was collected. Formated as "YYYY/MM/DD".
- NameOfSample: A varchar(30) representing the name of the sample.

- Group:

The Group Table contains the information about a group. The name of the group and when it was created. A group must be created online before it can be used.

- group_ID: The unique identifier for group. Incremented in the mysql Database
- GroupName: a varchar(30) value which represent the group name.
- DateOfCreation: A date value representing when the group was created. Formated as "YYYY/MM/DD".

- Reply:

The Reply table stores information regarding the messages being sent back and forth between the Client and the Pathologist.

- reply_ID: The primary key which is generated using a running total.
- master_ID: The second portion of the primary key. A foreign key to the Submission table
- SenderID: The ID of who sent a reply.
- ReceiverID: The ID of who received the reply.
- ContentsOfMessage: A Varchar(255), which contains the actual message.
- Date of Message: A date value for when the message was sent. Formatted as "YYYY/MM/DD HH:MM".

- Report:

The Report table is where the pathologist put the information from their review. Many Pathologist have the ability to review a submission but a report must pertain to a single submission.

- user_ID: A foreign key from the Pathologist table and a primary key.
- Master_ID: A foreign key from the Submission Table and the other primary key for Report.

- SubmissionReview: A varchar(255). The initial comments regarding a submission are placed here by a pathologist in the mysql database through the use of an api.
- Conclusion: A varchar(255). The last comments for a report. Stating the Submission is closed.
- DateClosed: a date value which represents when the report was closed. Formated as, "YYYY/MM/DD"
- ReportDate: A date value which represents when the report was created. Formated as "YYYY/MM/DD".
- Attachments: A field for any attachments associated with the report.

3.4.3 *Connection between Databases*

When a submission is ready to be sent to the MySQL database a number of triggers activate. First the app will determine if the user is connected to the internet. If not the submission will not send and will be saved as a submission only locally on the phone(As a draft). If the user is connected to the internet, then a prompt will pop up asking the user to type in their username and password (If they have not done so already). Once they enter a correct username, a number will be generated by using the running total from the MySQL database. This would be the new internal ID for the submission. Once the Master_ID is given to the submission and all of the connections to the submission are updated, then the submission will be added to the MySQL database. This process will be done by using an API supplied by the client.

3.5 **Website (Stretch Goal)**

The website will be the portal for users to register their accounts and to see submissions as well as send messages about the submissions. The website will use and update the MySQL database through an API like the one used for the app.

3.5.1 *Users*

Different types of users such as Lab Pathologists and not OVDL affiliated field users will be logging in through the same screen and they will have different levels of permissions associated with their accounts. Field users will only be able to see their own submissions and any replies to them. Pathologists should be able to see any report and their own replies to them. There will also be an option to add reports to groups and any user in a group will be able to see reports for that group.

3.5.2 *Registration*

Registration on the user side will be simple, just filling in fields for name, email, password, and permission level. The permission level will be a dropdown menu that will allow them to select one option. The server side will be much more complex. There must be checks for email and username uniqueness and password security. There will also be an authentication check to be sure users have the permission level they are supposed to.

3.5.3 *Login*

When users log in with their registered credentials the database will find their credentials and fill the submissions table with their submissions.

3.5.4 *Navigation*

Once the user has logged in they will be directed to the main page of the website. This will have a navigation bar across the top with links to the About and Account pages. It will also have the Logout button. This bar will be present at the top of all the screens to make navigating the site pages consistent.

3.5.5 Submissions

The main page will have a table with all of the submissions the user is allowed to see. There will be columns for the user who made the submission, the title of the submission, the case ID for the submission, and the date the submission arrived in the database. When a row is clicked a page with all the submission information will be shown.

3.5.6 Messaging

On the pages showing submission information there will be a message button that stays at the bottom of the screen. When clicked it will open a text field where users can write messages about the submission. Pathologists can give feedback or instructions to the submitter, and the submitter can provide additional information or respond. When a message is added to a submission the submitter will receive a notification on their phone app.

3.5.7 Account

The Account page will have a form where users can change their email, name, and password. They will need to enter their current credentials and the desired changes before submitting the form. This will update the database and the app will update when it syncs with the database.

3.5.8 About

The About page will contain paragraphs about the app, website, and the OVDL. It will have lists of instructions on how to use the app and website and how to contact the OVDL with questions.

3.6 Project Timeline - Alpha Level Release (Week 6, Winter Term)

3.6.1 Overview

Near the midpoint of Winter 2019, the team will have completed very basic functionality that will allow users to create a submission and store it locally on their phone. It is expected that users will be able to navigate to a page in the app that will allow them to create the submission and to a page that lets them see the submissions that they made.

3.6.2 App Pages

- Create a submission screen - This screen will contain all the information that the users need to fill out in order to create a complete submission. For the Alpha, this page will just collect the submission title. The rest of the fields will be added in by the Beta.
- "View Submissions" - This screen will show a list of every report the user has created. Each entry will show the submission's title, date of creation, and the case ID. Each entry should be clickable and clicking on an entry will take the user to a detailed page of that submission.
- "Home Screen" - This screen will have four UI buttons that the users can interact with. Tapping on a button will take the user to a different page in the app. For the Alpha, only "Create account", "Create Submission" and "View Submissions" will take the user to the corresponding pages. The rest of the pages will be implemented in time for the Beta release. All the buttons will be colored differently will have different icons. The text in all the buttons will have the same fonts. The specific buttons and background colors are as follows:
 - Create account - Green background with an icon of a person's shape. Clicking on this will open a web browser to the page where they can register an account.

- Create Submission - Blue background with an icon of a '+' sign. Clicking on this will open up a blank report that they can fill out.
 - View Submissions - Red background with an icon of a single folder. Clicking on this will take them to the "View Submissions" screen.
 - View Drafts - Dark green background with an icon of an open folder. The user will not be able to navigate to this page in the Alpha.
- Phone's Local Database - After creating a submission, the submission will be stored in the device's local database. The user will be able to see information about the database when they go to the "View Submissions page".
 - Toolbar - The toolbar will have white text. The text will display each page's name. On the far right, there will be a "more" icon and clicking on that will show different pages that the user can navigate to ("Home", "Create Submission", "Past Submissions", "Instructions", "Settings"). Only "Home", "Create Submission", and "Past Submissions" will be able to be accessed during the Alpha. On all screens but the home screen, add pictures screen, and add samples screen, there will be a back arrow on the left side of the header bar. Clicking on the arrow will take the user to the previous screen.

3.7 Project Timeline - Beta Level Release (Finals Week, Winter Term)

3.7.1 Overview

The Beta will have all of the basic functions of the app. At this point, a user will be able to create a submission store all of the relevant data into the SQLite database. Users will also be able to view the detailed pages of each submission and leave messages on the submission.

3.7.2 Create Submission Screen

The application will now be able to collect all data relevant to the submission. The following data (and form of input) are listed below:

- Submission Title - EditText
- Group Name - EditText
- Patient Name - EditText
- Species - EditText
- Sex - Spinner
- Euthanized - Checkbox
- Date of Birth - Calendar Fragment
- Date of Death - Calendar Fragment

There will also be an "Add Pictures" button, a "Save Draft" button, an "Add Samples" button, and a "Submit" button. Clicking on "Add Pictures" will take the user to the "Add Pictures". Clicking on "Save Draft" will save the submission as a draft into the SQLite database. Clicking on "Add Samples" will take the user to the "Add Samples" screen. Lastly, clicking on "Submit" will save the submission into the SQLite database.

3.7.3 Add Picture Screen

The user can navigate to this page from the "Create Submission" page. This page will initially have five boxes with camera icons in them. Clicking on a box will allow the user to insert a picture (by opening the phone's camera or going

through the phone's local gallery). Once the user selected a picture, the box will show the picture instead of the camera icon. The user can add a maximum of five pictures. There will be a button at the bottom of the screen and once the user clicks it, the pictures will be saved to the submission and they will be returned to the "Create Submission" page. Long pressing on a submitted picture will give the user a prompt to remove the picture from the submission.

3.7.4 Submissions Screen

The Case ID will be color coded based on whether or not the report has been successfully sent to the server yet. It will be red if it is still waiting to send, and it will be green if it was successfully sent. Clicking on a report entry in the list will take the user to a detailed page of the report.

3.7.5 Detailed Submission Page

This is the page shown when a user clicks on a report in the "Submission Screen". The user will see all the information that they included when they created the submission. The user can also see any messages that were sent to them by people who commented on the submission.

3.7.6 User Login

Users will be able to login to the app

3.7.7 Drafts

When a user is creating a submission, they will have the option to save their current submission as a draft and won't necessarily submit it. There will be a button at the bottom of the "Create Submission" screen that says "Save Draft". Clicking on it will automatically save the draft and return the user to the home page. Additionally, the home page will have a button called "View Drafts", and clicking on this button will take the user to a page that shows all the user's saved drafts. The drafts will be listed in a table similar to what is shown in the "View Submissions" screen. Clicking on a draft will take the user to the "Create Submission" page where all of the information saved in the draft is automatically filled out for the user.

3.7.8 Instructions Screen

This page will have four sections (Registering a New User, Creating a Submission, Sending a Submission, and App Usage) that have information underneath them detailing how to do each process

- Registering a New User - Provides a link to the web page that allows users to create an account
- Creating a Submission - Details how to navigate to the page to create a report and how drafts work
- Sending a Submission - Explains that clicking on the "Submit" button at the bottom of a Submission will send it to the server if there is an internet connection. If there isn't an internet connection, then it will automatically send when the user regains a connection
- App Instructions - Provides a broad explanation on how to use the application

3.7.9 Add Samples Screen

The user can navigate to this page from the "Create Submission" screen by clicking on the "Add Samples" button. This screen will have EditText fields to collect the location and name of the sample from the user. There will be a NumberPicker that the user can interact with to set the number of samples. There will also be an "Add Sample" button.

When this button is clicked, the application will take all the data and add the sample data into a list at the bottom of the screen. Lastly, there will be a floating action button that will save the samples when it is clicked.

3.8 Project Timeline - Engineering Expo (Week 6, Spring Term)

3.8.1 Overview

By the time of the Engineering Expo, the team will have polished up all aspects of the application and make it more user friendly. Additionally, the team will have implemented other useful features that weren't crucial enough to be developed in time for the Beta or Alpha.

3.8.2 Toolbar menu

The toolbar menu will now have the following options and functionality:

- Clicking on the "Instructions" option will navigate the user to the "Instructions" page
- Clicking on the "Settings" option will navigate the user to the "Settings" page

3.8.3 Create Submission Screen

The button that the user clicks on to be taken to the screen where they add pictures to the Submission will now be badged with the number of pictures currently added. This means that if the user added 3 pictures to the Submission, then the number "3" will show up as a badged icon on the button.

3.8.4 Adding Pictures Screen

Instead of always having 5 camera icons that the user can tap on to add a picture, there will only be one of these icons shown at a time (in addition to pictures that were already added). In order to add a new picture, the user will have to click on this one icon in order to add a picture. The icon won't show up when the user has already added 5 pictures since 5 is the maximum limit.

3.9 Conclusion

The goal of this project is to deal with the disconnect and create an effective means of communication between the on-the-ground field personnel and the veterinary pathologists in laboratories. The OVDL wants to test how effective remote diagnostics can be. To do this, the data sent from the field must be accurate and precise. Also, the method of sending data should not require much training to use in the field. If the tests are successful, then it could lead to an international spread of remote diagnostic work.

3.10 Glossary

- **Adapter** - An object in Android that connects a view with the data for that view
- **Client** - The user who creates submission and takes the pictures out in the field for a Pathologist to review the information
- **Database** - An organized collection of information and data to be used by software for easy access
- **Data Dictionary** - A section describing the data types in a database. Used to describe how the tables in a database are set up and what columns each table contains. Also used to describe what each column's data type is and what is stored in the column

- **Field** - Inside a table there are columns which are assigned data types and store information based on the data type assigned to it
- **GridLayout** - A type of layout in Android where items are arranged in a rectangular grid
- **LinearLayout** - A type of layout in Android where items are arranged in either a single column or row
- **ListView** - A view group in Android that shows a list of scrollable items
- **Pathologist** - A user who creates reports by reviewing submissions and then sending feedback to the client
- **Report** - A document which contains the feedback provided by a Pathologist who reviewed a submission. A typical report contains feedback, when the submission is closed, and the author of the report
- **Submission** - A document which contains information reading an animal in the field. The submission would typically contain images, the amount of samples taken, and general information about the animal
- **Table** - Used in a database as the storage structure
- **Varchar(#)** - A datatype in MySQL which allows a string of characters to be stored in a field and can be up to a length defined by (#)
- **Reply** - A message sent between the user and the client.

4 TECH REVIEW - BRANDON JOLLY

4.1 Introduction

The technology to be described in this review will be focusing on the type of database we will be using, how we will be interacting with the database, and what phpserver the database will be interacting with. The three databases we looked into was MySQL, MongoDB, and SQL SERVER. Since MongoDB and SQL server each have their own interface while MySQL has implemented many, we will be looking into the different ways to view a MySQL Database. Such as with PHPmyAdmin, MySQLWorkBench, and the console. The last portion of the technology review will be looking at which web frameworks we will be using. The options we will currently be looking at are: CakePHP, Laravel, and Phalcon.

4.2 Database Selection

4.2.1 MySQL

MySQL is an open sourced database management system owned by Oracle [1]. In the acronym LAMP (Linux, Apache, MySQL, PHP), MySQL is central to its structure. Many website depend on a MySQL database such as Facebook and Twitter [2]. Some of the benefits that are included with MySQL are the ability to have nested SELECT Statements and use of Triggers. One major benefit of a MySQL Database system is the ability to easily be connected to an existing PHP server. There are a few other Databases that share this ease of connection but MySQL is a popular choice and as such shares the most compatibility.

Some of the limitations with a MySQL server is if we are using a database engine other than InnoDB, then we will fail to comply with SQL Standards, and thus lose some functionality. The major one being foreign key references. Triggers are limited to one per action. This would limit our ability to allow multiple triggers to activate after a single UPDATE action. They also can not be defined on Views.

4.2.2 MongoDB

MongoDB is another open source language and they are classified as a NoSQL database program. Meaning MongoDB is a database structure which does not use the common place tabular relations which a relational database uses [3]. Instead

MongoDB uses JSON-like documents for their data storage [4]. Despite not having a tabular structure, MongoDB still shares many of the same attributes of a relational database. Such as indexing, and ease of replication. To search information in a MongoDB the use of AD hoc queries is implemented [4]. Allowing a user, the same functionality as a typical SQL Database.

Some limitations of this database format is a major security flaw. There have been multiple instances where a MongoDB was held hostage due to a hacker getting past their security systems. Another issue is the possibility of rolling back write statements without proper authorization. This is done when there is an application which can connect to two distinct MongoDB processes, but they cannot access each other.

4.2.3 SQL Server

SQL Server is another relational database invented by Microsoft . Its primary purpose is to be used as a storage system for other software applications, such has Microsoft Excel. The application may run on the same machine or over a network connection. SQL Server has many different editions to fulfill many different types of requirements. Allowing an edition to be focused on a specific task and avoids being a jack of all trades type of application. SQL Server allows a user to define their own composite types if they so choose.

Some Limitations of SQL server is due to their focused on working with specific software applications. Because of this they do not have a strong connection to a website interface with the use of an additional third-party program to facilitate the connection or by purchasing another package to allow website connections. While the database itself has many powerful features such as the capability to provide analysis services [5] and a service broker which controls messages between applications through a tcp/ip connection [6]. It does not have a strong connection to a php server.

4.3 Database Visuals

4.3.1 MySQL WorkBench

MySQL workbench is developed by Oracle and thus has the greatest compatibility with a MySQL Database. This interface is the second most popular download on the MySQL website with over 250,000 downloads a month. All of the functions a user needs in the MySQL are located in one east screen with the need to change to a different window for each table, view, or trigger. Allowing a user to easily navigate their database without the hassle of trying to find the right window to click.

Because of their hands-off approach however this mean a new user may be easily confused on how to interact with the database. The tool doing little to add a new user such as having little help online. While writing SQL there are times when the debugger provide unhelpful error codes leaving the user left to debug their own code with the assistance of software. Once again this isnt a problem for an experience database administrator, but it is for somebody new to the tool.

4.3.2 PHPMYAdmin

PHPMyAdmin is a free and open source database admin tool for MySQL and MariaDB. phpMyAdmin not only acts as a way to interact with a users database but it could also double as a web hosting service. With new users in mind this tool helps somebody who is not familiar to databases an easy way to get to learn how to use them. There are many helpful wizards to create and modify a table. Taking a lot of the responsibility away from the users allows more people to use the software without the worry they are damaging the main database.

An issue with the interface with PHPMyAdmin is the interface is geared towards new users. Forcing them to go through specific pages in order to implement a new view or to add a new column to table. Typically defaulting their wizard approach instead of using SQL. Making very simple tables and adding constraints a hassle to an experience user.

4.3.3 Console

By not using any database admin tool and just using the console provided with MySQL we give the user the greatest amount of freedom. When using the console, we do not have to worry about updating the admin tool or having to deal with any bugs associated with the tool. Allowing a user who is proficient with consoles in general the ability to make rapid changes to the database, without having the need to find the correct button to do so.

One negative with using the console is it allows a gives a lot of power to a user. If the non-proficient users gains access to the database through the use of a console, they may make changes they are not approved for. There is also a chance of data loss where they would be no way to retrieve the information. Another issue with the console is the barebones visuals when seeing the tables in the database. Something an admin tools does a much better job of.

4.4 Web Frameworks

4.4.1 CakePHP

CakePHP is an open-sourced web framework which follows the Model-View-Controller (MVC) method of implementation. One major benefit of CakePHP is the ease of getting it up and running. Their website provides multiple resources in order to help with any new users of the software. Allow for easy problem solving when something goes wrong. Being made with php allows us easing access to connecting to our database where we wont need a third-party program to help facilitate the connection. The framework also uses many well-known engineering philosophies such as: the MVC model mentioned earlier, convention over configuration, and association data mapping.

Since CakePHP is in active development there are bugs we would have to deal with each new implementation. Since the framework is designed to be beginner friendly some of the more advance admin controls are hidden by the developers which would reduce the amount of customization we could do with the product.

4.4.2 Laravel

Laravel is a web framework which uses the MVC framework for its implementation. Designed for rapid development, Laravel has become one of the more popular frameworks used in the industry and as a huge community of developers [3]. One major benefit of Laravel is the fact that it is a free framework as opposed to Phalcon, or CakePHP. Another benefit is that Laravel also helps with code organization to allow users to easily read and debug code in case an error appears during implementation. Larval also has a strong focus on rapid development allowing us to quickly implement and test our app [7]. If we went with MangoDB as our database, then Laravel would be an excellent framework to pair it with.

One issue with Laravel is the fact that it runs a lot of queries to the database. Slowing down the overall performance and where time is essential could be a deal breaker for using this framework. Since it is an open source code with multiple developers there is a possible security risk involved because the framework is so well known including its bugs.

4.4.3 Phalcon

Phalcon is a web framework which follows the MVC model as well. Compared to many frameworks Phalcon runs at a much faster pace due to its on being built on a c-extension. Speeding past Laravel and CakePHP which both relay on

PHP for their connections to a database. Another unique aspect of Phalcon is how it uses its own SQL Dialect, PHQL. Phalcon also has been geared towards working with MangoDB and other noSQL formats.

As opposed to CakePHP and Laravel, Phalcon is not as open sourced and thus as slower update times. A bug may stay for as long as the developers let it. Leading to long periods where users would have to work the bug due to lack of transparency with the source code.

4.5 Conclusion

Our group will be choosing to use MySQL for our databases because it gives us the ability to work with the existing systems at the lab. For the database visual representation, our group will be using MySQL Workbench because the client in charge of maintaining the database is proficient and does not need the hand holding built into the PHPMyAdmin software. If our group gets to the stretch goal of designing a web interface for our app we will be using Laravel due to the rapid development time and we do not have to worry about a massive amount of queries being run on our database.

5 TECH REVIEW - KATHERINE JEFFREY

5.1 Introduction

The OVDL wants a native Android mobile application that collects field data and images, stores the information on a native SQLite database, sends them to the lab's MySQL database, and gets real time feedback from the lab. A stretch goal for the app is having a feature that analyzes the images taken using the phone's built-in camera and gives the user feedback on image quality. It should only send images that are well lit, do not have shadows or discoloration, and clearly show the subject. If there is time, the team will try to implement it because it would be beneficial for the client.

5.2 IDE

To create the Android app the team will use an Integrated Development Environment (IDE) to design and build the app software. It is very important to choose one that provides all the functionality needed from simple app layout, usability and image capture to database connectivity and image quality assessment.

5.2.1 Visual Studio

Visual Studio is Microsoft's IDE, which has only in the past few years started offering native mobile development with Java for Android. With some downloaded extensions it can have Java autocomplete, which shows options for functions and parameters as they are typed. Apps can be tested on connected devices or an Visual Studio's Android emulator. Visual Studio comes with a debugger, a feature every good IDE has. There are some downfalls with Visual Studio like the GUI and usability of resource files. It has no drag and drop feature for layouts, everything must be either programmed manually or created somewhere else and imported. Visual Studio is however the best option for creating a mixed Android app with C++ and Java. No other IDE does this very well. It is a useful option, but for this project a mixed app is not necessary so Visual Studio is not the only option. [8]

5.2.2 Android Studio

Android Studio is an IDE made specifically for Android development by Google. It supports Java and Kotlin, but java is much more popular. Java autocomplete is especially helpful in Android Studio with the boost of IntelliJ. It is great for beginners because it provides lots of helpful hints, code completion, and an excellent debugger. The best feature,

which is unique to Android Studio, is the drag and drop UI which makes creating layouts convenient. Developing apps can be tested on a connected device or one of Android Studio's emulators. Another nice thing about Android Studio is how fast it works, building projects in half the time of other IDEs. For building this app Android Studio is the obvious choice, as speed and an easy learning curve will be essential to the success of the project. [8], [9]

5.2.3 *Eclipse*

Eclipse has been a popular IDE for many years, and not primarily for Android development. It is a Java IDE and has functionality for much more than mobile development. Because it is large and has so many options, it is very slow in every part of development. Android apps are often programmed in Java so supporting them is not a leap for Eclipse. It has Java autocomplete, which is helpful for new and experienced developers. Eclipse does not have a nice GUI like Android Studio, which makes it more difficult to design screen layouts. There is no drag and drop feature, everything must be coded manually. While technically functional, Eclipse is a clunky and outdated IDE for Android Development and it would take too long to learn for the less experienced developers on the team. [9]

5.3 **Data Transfer**

A significant amount of data will be processed through the app and stored in a remote database, and the app needs an Application Programming Interface (API) to handle the sending and receiving of that data. Some of the data is sensitive and confidential therefore data security might be a concern for the clients. The MySQL database that will be storing the data is server based, and because the phones will not always have internet connection they need a serverless way to store data locally, which is why an SQLite database will be used. The API will need to transfer data from the local SQLite database to the MySQL database on the server.

5.3.1 *SOAP*

Simple Object Access Protocol (SOAP) was developed by Microsoft for accessing web services using XML, and is still the most secure way to implement messaging services over the internet. It is large and has an extensive and very strict set of rules that standardize it. This makes it secure, but harder to use and much harder to implement than REST. A nice feature of SOAP is the built in error handling, which provides helpful feedback for fixing errors when they are encountered. SOAP is transport independent, meaning it can be used over HTTP or SMTP, which provides some flexibility when building web applications. It is a strong contender because of this independence and for information security, which could be useful for the confidential data gathered in the app. [10]

5.3.2 *REST*

Representational State Transfer (REST) provides access to web services over the HTTP protocol in a simple, flexible way. REST is flexible because it can output data in many different forms such as CSV, JSON, and RSS, not just XML, which makes it easy to parse in any language. It is simple because it can just use a URL to obtain information without a rigid structure. REST is easy to implement and the data it supplies is simple to process, making it a preferred method for many web and mobile applications. It does compromise on data security in favor of efficiency and speed, which unless the data being transferred is highly sensitive, is not often a problem. However, for important passwords and sensitive information REST may not be the best tool, even though it is so simple and efficient. [10]

5.3.3 JSON

JavaScript Object Notation (JSON) is a simple format used to organize and store data for transfer. JSON uses the universal data structures ordered lists and name/value pairs to organize data so it can be used with any programming language. JSON is often used with REST APIs to make data exchange easier for programmers. The JSON format is easy for humans to read and edit, not like XML which uses metadata and tags in a less defined format. For this project data gathered in the app will be stored in JSON format, stored on the SQLite database, and converted and transferred to the MySQL database by an API. [11]

5.4 Image Quality Analysis

The most important part of a remote autopsy report is the images; the pathologists need to clearly see the specimen to make an accurate diagnosis. Any discoloration, shadow, or blur in the image could lead to an inaccurate diagnosis or force a pathologist in the lab to request more images from the user. Checking the image quality before adding it to the report is a stretch goal, but a worthy one the team hopes to achieve.

5.4.1 Android Camera

The Android Camera package can be easily included in an Android Studio Project, and provide functionality that will be necessary for this app. There are classes that can gather information about the phone's cameras such as number, quality, and settings. There are also classes to select which camera to open, the front or back facing camera, open it within the app with a customizable appearance, and capture an image as a JPEG or RAW image. Most importantly, Android Camera has configurations that can be set for each image captured to ensure quality. There are fields for color correction, brightness, exposure, focus, orientation, and many more that can be configured with the phone's sensor and flash. A field to record the location of the image is also included, which will absolutely be used in the project. [12]

5.4.2 Xamarin

Xamarin is a cross-platform product which can be used with Visual Studio for Android Development that contains a Camera Class API. This class has options for controlling the phone's camera functions from within an app. It is used to set "image capture settings, start/stop preview, snap pictures, and retrieve frames for encoding for video" [13]. The Camera Class is a client for the Camera service, which manages the actual camera hardware. Xamarin includes classes for detecting camera information such as orientation, focus, and flash, all of which could be useful for this project. The team will not be using Xamarin because the project will not be done using Visual Studio and it is not needed in Android Studio. [13]

5.4.3 Mitek MiSnap

MiSnap is an image capture SDK that can be included in a native Android app and provide image quality feedback instantly with the phone's camera using machine learning and computer vision algorithms. It can detect brightness, glare, focus, and ensure the image captures all necessary information. It is mostly used for mobile banking transactions and automatically filling out forms online. Pieces of the software could be used to make sure images taken with this app are clear and do not have shadows or glare obfuscating the object of the image. It is customizable and could supposedly be integrated into the app easily. The customization could be complicated due to the large scale of the SDK, and it is not open source so obtaining the rights is not free. [14]

5.5 Conclusion

The IDE that will be used for the project is Android Studio. It is the easiest IDE to learn, with a drag-and-drop GUI, high build speeds, and the project only requires an app for the Android Operating System. Because the team is using Android Studio it will also use the Android Camera package because it can be easily implemented with the IDE. SQLite will be used for database storage on the phones because that is what the client wants and it is the best way to store the data locally. A REST API will be used to transfer data between the app, website, and database. The API will utilize the JSON format for organizing the data during transitions. The Android Camera package provides some of the image analysis functionality, but supplementary software will be needed for a higher caliber of image analysis. This is a stretch goal for the project, and might be implemented by the clients after the project is completed. All decisions must also be approved by the client.

6 TECH REVIEW - BRADFORD WONG

6.1 Introduction

6.1.1 Role

Bradford will be doing native android development for this project. In this tech review, he will be writing about mobile application frameworks, frameworks for automated testing, and languages for mobile applications

6.1.2 What the Team is Trying to Accomplish

The team is attempting to create a mobile Android application to improve pathologists' ability to perform remote diagnostics. This application will allow pathologists in a lab to be able to communicate with personnel out in the field. The field personnel will be able to take pictures and write text to create a report. They will then be able to send the report to the personnel in the laboratories, who can then send feedback to the field personnel.

6.2 Mobile Application Frameworks

There are a variety of choices when it comes to using a framework to develop this mobile application. The team will most likely use native Android because that is what the client specified, but it is still useful to look at the other frameworks. This paper will look at the following criteria: how many platforms can take advantage of the framework, the development and compilation speed, how popular the framework is, and any extra features in order to evaluate the frameworks.

6.2.1 React Native

A framework that can be used to develop mobile apps is React Native, which is a framework that only uses JavaScript. Native Android, native iOS, and React Native use all of the same fundamental UI building blocks. JavaScript and React are built by putting these blocks together. Additionally, React Native builds faster than native Android because the app can be reloaded instantly after changes instead of having to recompile all the files again. Many mobile apps use React Native including Facebook, Instagram, and Pinterest [15]. The main benefits of React Native are that an application using this framework will work on both iOS and Android, it is faster to build using this framework, and a developer can make changes to the code while the app continues to run and automatically reloads after changes. The drawbacks are that there are less tools to create navigation components to enable seamless UX for users, it lacks some custom modules,

and some components such as cameras and sensors still need native developers and knowledge. Lastly, Facebook owns React Native, which can be problem if they decide to stop supporting this framework [16].

6.2.2 Flutter

Google also created a mobile application SDK (software development kit) called Flutter, which creates native interfaces on both iOS and Android quickly. It is free and open source. Flutter also has fast development because like React Native, it has Hot Reload. Hot Reload allows the emulator that the developer is using to update automatically and quickly after code changes without needing to recompile every file [17]. Like React Native, Flutter enables cross-platform development. It also provides its own widgets, which are created with a high-performance rendering engine. The main benefits of Flutter are that there is Hot Reload, the code runs on both Android and iOS platforms, and Flutter apps look the same on older operating systems. The problems with Flutter are that it is still in beta, the libraries and support still aren't as rich as native Android, and it still isn't widely supported by Continuous Integration services such as Jenkins [18]. Continuous Integration services are services that developers use to merge code into a single repository and verify that the applications build correctly. Some apps that use Flutter are Alibaba, Google Ads, and Hamilton Musical [19].

6.2.3 Native Android

Native applications are platform-specific apps that are built using a programming language that is specific to that platform. Since native applications are developed for a particular platform, they are fully able to access features specific to that device such as the camera, GPS, and Bluetooth. Native applications work on the devices operating system, meaning that they require total access to all of that devices hardware and functionality. Native android provides developers with a standardized software development kit with tools, libraries, and documentation. The benefits of using native Android is that the app will perform faster, work offline, and have a recognizable look and feel for users. The drawbacks are that there is no flexibility or cross-platform development and requires frequent app updates [20]. Since there is no cross-platform development, the app will only run on Android devices and will not work on iOS devices. This is the option that the team will take because it was specified by the client and because it is the most stable between the three options.

6.2.4 Overall Recommendation and Conclusion

It is recommended to use native Android for this project because of a variety of reasons. The largest reason is that the client specified that they wanted a native Android application. React Native is a solid option because developing with it would be fast, but it is risky because Facebook could decide at any time to stop supporting the framework. Flutter is very risky because it doesn't have as large of a community as native Android. Due to the smaller community, if the group runs into a problem and needs help, there will be less resources that the team can reference. Furthermore, Flutter is currently only in beta, meaning that it does not have as many libraries as native Android. Overall, native Android is the best and recommended option because of its stability, large community and vast resources, and the fact that the client specified that they wanted a native Android application.

6.3 Automated Testing

Automated testing is useful for ensuring the quality of the application. In order to evaluate the quality of an automated testing framework, this paper will look at criteria such as how fast the framework is, how easy it is to use, and how much it can test the app.

6.3.1 Robotium

Robotium is an open-source framework that is used for automated testing with Android applications. Developers can use Robotium in order to test various scenarios for Android activities. Robotium enables developers to provide automated testing for UI test cases with Android applications. The benefits of Robotium are that it is easy to write code and it isn't necessary to spend a lot of time to create solid test cases. Additionally, test cases can be written without needing extensive knowledge of the application. Robotium automatically finds views, has automatic timing and delays, is able to make its own decisions (such as when to scroll the application), and integrates smoothly with Maven. The drawbacks are that it can only handle one application at a time and cannot simulate clicking on the software keyboard, meaning that that developers will have to write code to enter the text. Robotium also can't interact with status bar notifications and can be relatively slow, especially on older devices [21]. Robotium seems like the best testing framework because it can test so many Android features.

6.3.2 Appium

Another automated testing framework is Appium. Appium is useful because it supports cross-platform test automation, meaning that it can test both iOS and Android applications. Additionally, it supports tests on any framework using any language. The developers behind Appium believe that testers shouldn't have to recompile the app or modify it to test it, testers shouldn't be locked into specific languages or frameworks in order to write and run tests, automation frameworks shouldn't reinvent the wheel, and automation should be open source. Appium uses a standard API across all platforms, meaning that testers won't have to modify code or recompile the app [22]. Additionally, Appium is free, open source, has a well supported and active group, and should be relatively future proof. The drawbacks are that it doesn't support any intelligent waiting and there is limited support for gestures and lacks support for older Android versions [23].

6.3.3 UI Automator

UI Automator is a UI testing framework that is used to test the UI in mobile applications. The framework includes a viewer that allows the tester to inspect the layout hierarchy. It also has an API (application programming interface) to retrieve information regarding the current state of the device and perform operations on it. Additionally, the framework has APIs that support cross-app UI testing [24]. UI Automator is also supported by Google. The problem with this framework is that it only supports native Android apps [25]. Furthermore, UI Automator only works on some Android devices and doesn't support webview, meaning that it isn't possible to easily access Android objects [26].

6.3.4 Overall Recommendation and Conclusion

This paper recommends Robotium for a variety of reasons. Robotium is easy to use and will not require a significant amount of time for the team to use. Also, the framework should be able to test every functionality that the team will want to verify. While writing tests with Appium seems fast because the app does not have to be compiled, it is not recommended because it doesn't support many gestures that the team might want to test. Additionally, it lacks support for older Android devices, which is a problem because the team will want to test as many devices as possible. UI Automator also seems like a solid option, but it only support Android devices with API level 16 or above and doesn't support webview. This framework lacks support for a decent amount of Android devices and objects. Overall, Robotium is the recommended framework because it is easy to use and can test the most features on a wide range of Android devices.

6.4 Languages

When it comes to languages, there are several different options. This paper will look at the ease of use, functionality, and the popularity of each language in order to determine what language will be used.

6.4.1 Java

Java is an object oriented programming language with many benefits. There are many Java libraries in the Android SDK, and apps built using Java build relatively fast and are fairly light [27]. There is also a very large community of developers who use Java when developing Android apps, which means that it is easier to find resources for help [28]. However, there are some cons with Java. Java has limitations such as that it requires a larger amount of memory. It is also a verbose language, meaning that it requires more code. Having more code can potentially lead to a higher amount of bugs [27]. Java seems like the best option because of its wide popularity and the fact that there will be a lot of resources that the team can turn to for help.

6.4.2 Kotlin

Kotlin was initially created by the developers from JetBrains with the purpose of adding more modern features to Java that will be useful for mobile development. The benefits of using Kotlin are that it is easy to use and is fairly concise, which means that there are less opportunities for bugs in the code [27]. Kotlin is much more concise than a language like Java because many methods such as getters, setters, and toString() don't need to be specified because Kotlin generates them automatically. Java is currently the most used language for Android development, but this may change since Kotlin was declared by Google to be Android's official language [28]. Additionally, Kotlin works with Java, meaning that developers can create new modules using Kotlin and those modules will be able to work alongside the already existing Java code. Kotlin is compatible with every Java library and framework. The drawbacks of using Kotlin are that it has a relatively steep learning curve due to the nature of the concise syntax and it is usually slower to compile than Java. Furthermore, there is a smaller community of developers who use Kotlin, meaning that there are less resources for help and it is more difficult to find answers. Additionally, Android Studios compilation typically runs slower when there are Kotlin modules mixed in with Java code as opposed to just Java [27].

6.4.3 LUA

LUA is an open source scripting language. It is lightweight, fast, and powerful. It is used in applications such as Warcraft and Angry Birds. Developers who code using LUA also use the Corona framework which provides some advantages as well. It is a cross-platform framework meaning that the code in LUA will work for many different platforms. Additionally, it is possible to call native libraries such as Java [29]. LUA is also able to run on an emulator without needing to be compiled first, limiting the down time in development. However, LUA isn't supported by Android Studio, meaning that developers will need to find a different text editor. LUA is also fairly limited when it comes to Android app functionality. As a result, LUA is more ideal for mobile apps that are fairly simple, which likely isn't the case for this project [30].

6.4.4 Overall Recommendation and Conclusion

Java is the recommended language for this project. There is a large community of developers who use this language, which means that there will be a lot of resources that the team will be able to reference. Furthermore, Java apps are fairly

fast and light. Kotlin also seems like a great option because it is easy to use and concise. However, Kotlin has a small community, which means that the team might not be able to find help for certain problems that they may face. Also, Kotlin has a steep learning curve and is slower to compile. LUA is a powerful language, but it doesn't seem appropriate for this application. This language is limited when it comes to Android functionality and is typically used for simple mobile apps and games. Since this project won't be simple and is not a game, LUA doesn't seem like an appropriate language for the team. Overall, Java is the recommended language because of its large community and the fact that Java apps are fairly fast.

6.5 Conclusion

There are a lot of different options when it comes to mobile application frameworks, automated testing, and languages. For this project, it is recommended to use native Android, Robotium, and Java. Native Android is recommended because it is stable, has a large community, and the client specified that they wanted a native Android application. Robotium is the recommended automated testing framework because is easy to use and should be able to test all the functionality that the team will want to verify. Lastly, Java is the recommended language because it has a large community, Java apps are fairly fast and light, and it seems the most appropriate for this type of application.

7 WEEKLY BLOG POSTS

7.1 Brandon Jolly

7.1.1 *Fall*

- Week 3: On Tuesday group 72 had a face to face meeting with the other members of the group. We gave our ideas on what the project might look like and stated what skills we could provide the group with. We then determined who would be emailing the client later that Tuesday. A meeting was set up for tomorrow at 3:45 in a conference room on the second floor of Kelly.

On Wednesday group 72 met face to face with our client, Dr. Loher and Bill (Forgot his title). During the meeting we discussed what the projects end goal would be and some ideas for what features could be included in the project if we had met the goal earlier. We then discussed what problem Dr. Loher was trying to solve. Her words where, Helping those in the field who lack the expertise to do so themselves. After determining what the problem was we, discussed with Dr. Loher what the app must have and what it could do without. At the end of the meeting we gave Dr. Loher a rough timetable for when to complete the project milestones throughout the year.

On Thursday group 72 set up a meeting with the TA for Wednesday at 2:30pm at Kelly. We also create a repository for our group project and determined how we would organize the repo.

- Week 4: On Wednesday group 72 met with the Richard. We got our questions answered then started to work on our progress report. We edited finished and turned the report that Wednesday. On Friday we started a rough draft on the requirements document. And plan to meet on Monday to work out more of the details.
- Week 5: Tuesday, 10/23 Created a rough draft for the requirements document. Wednesday 10/24 Talked with the Richard to gain a rough idea of what some acceptable topics for the tech review would be. Thursday, 10/25 Had a group meeting discussing plans to work on the database design. Drew a rough ER diagram creating a basis for the database we would be developing. Picked our team name, and then ending the meeting discussing a meeting

time with our client. Friday 10/26 Sent an email describing the progress to our client and to set up a meeting time.

- Week 6: Tuesday 10/30 Created team standards for Group 72. Determined who will be our SCRUM master and how our team will communicate. Wednesday 10/31 Meet with the TA, got our questions answered. Meet with the client. presented our designs so far and received feedback. Began actively modifying the database design. Brought everybody to the same page and answered any questions regarding our current database design and user interface. Friday 11/2 Turned in the tech review for the week.
- Week 7: Didn't really work much on the project this week was dedicated to the Tech review. On Tuesday we discussed some more designs for the app interface and talked about who who might be in charge of what during the project implementation
- Week 8: Meet with the client to review terms used in our database on Monday. On Wednesday we asked our questions with the TA. Such has how to improve our tech reviews, and how to improve our Requirements document
- Week 9: On Tuesday We met with the client and had an two hour long meeting discussing the time line for next term and how we wish to connect the two databases with each other. We presented how the data dictionary would look in the final product so they would be able to learn how the database is connected and which each variable means in the long run.

7.1.2 *Winter*

- Week 1: Met with the team and began delegating tasks for the term.
- Week 2: Progress: got set up with android studio and created the sqlite database and a submission's table.
Problems: was unable to push to the repo despite having read/write permissions,
Plans: Fixed read and write permissions causing the 403 error on android studios.
- Week 3: Begain working with the Mysql Database.
- Week 4: Got the mysql database running. Next week going to fill it out.
- Week 5: Progress:Finished some tables in the SQLite database.
Plan:Next week plan on working with the new API
Problems: possible issues with the new API.
- Week 6: Progress: Created Sample, Sick Element, and the Picture tables in the SQLite database.
Problems: I created a bad pull request and essentially had to recreate my pull request. What I did the first time was just create a branch and the did a pull request to that branch. But that branch original came from an older master which had not been updated properly. Because of this I had to create backups for my work, delete the branches, then update them all over again. Then create a pull request to the master branch with my already approved changes.
Plans: Finish the SQLite database for next week.
- Week 7: Finished adding the SQLite database into the android app
Problem: There was some string errors when adding the tables, so the create table would look like this, "CREATE TABLE SUBMISSION"
Plans Finish the MySQL Database
- Week 8: Progress: Worked on setting page, and info page

Problems: haven't developed an android app so I had to do a lot of learning this week.

Plans: Finish the setting page and info page.

- Week 9: Progress: Started development on connecting our website with the database.

Problems: Needed to catch up on learning Node.js

Plans: Begin testing website database interaction.

- Week 10: The test was successful but for only a one time use. If we tried to connect to the database after the first time an error would pop up. We have reported this to the client.

7.1.3 *Spring*

- Week 1: Progress: Meeting with the client

Plans implement changes the client made

Problems: the server for our api may crash the service may have problems.

- Week 2: Progress: met with client and re worked the mysql database

Problem: api still in development, River and wifi slowed progress down

Plan: Complete documents for next week

- Week 3: Progress: Was able to recreate database using a script

Problem: client still working on the API

Plans: meet with client next week to discuss the API.

- Week 4: Progress: Finished team poster and client edits

Plans: work with client on the api.

- Week 5: Progress: Tested Smartphones with the app

Problem: They didn't work at first but that was because the Phones were not in developer mode

Plans: Work with client on the api.

- Week 6: Progress: Got pictures to show up on the phone

Problem: pictures were not displaying pictures with the camera function

Plans: Work on why when you save a picture in drafts it doesn't save.

7.2 Katherine Jeffrey

7.2.1 *Fall*

- Week 3: This week the projects were assigned on Monday and I emailed my teammates that afternoon. They responded right away and we decided I would email the client to set up a meeting. I emailed the client on Monday and she responded right away. We decided to meet Wednesday afternoon so I reserved a conference room in Kelley. The team had our first team meeting on Tuesday after class. We discussed the project and our upcoming meeting with the client. We also talked about the skills we each have that can contribute to the project. I have android development and database experience, Brandon has database experience, and Bradford has mobile app development experience. We agreed on some questions for the client. The meeting with the client on Wednesday went very well. We clarified what the project requirements are and received information we can use to start the design process. We have not had any problems with the team or the client so far, and the only problem I can see in the future is feature creep. This project has a lot of potential for growth and the client has

ideas for it that might not be possible for us to complete this year. Next week we will be refining the requirements in the document and meeting with the TA for the first time.

- Week 4: This week we worked on the Problem Statement. Each team member wrote one individually and then we met to combine them into one better document. We used bits of each members paper and submitted it early. We also met with our TA for the first time this week. The meeting was short because we did not have any problems or many questions. After the meeting the group decided to attend the special Friday office hours to get more information about the requirements document.
- Week 5: We worked on the Requirements Document and the database design. We made a rough draft of the app UI flow and set up a meeting with the clients.
- Week 6: This week the team finished the Requirements Document, the Team Standards Document, and had a design meeting with the clients. Brandon created a database diagram which we edited in the meeting to make sure we were all on the same page and understood the layout. I created mockups of the app UI with my understanding of what the app should do and look like. I showed them to the team first, and we came up with some additions. We used them during the meeting and the clients had lots of good feedback, changes, and answers to all of our questions. We are now ready to create a new set of mockups and move forward with database implementation.
- Week 7: This week the team members finished their Tech Reviews individually. I wrote mine on IDEs, Data Transfer options, and Image Quality Analysis. I met with the team on Tuesday and I said I would make mockups of the web interface, Bradford said he would make Gantt charts for the year, and Brandon said he would contact one of our clients about setting up the database and web hosting.
- Week 8: This week we started the Design Document, deciding on who would work on which components and how they would be structured in the document. We worked out a timeline for the rest of the year and decided which parts we would get done for the alpha, beta, and expo releases. We also set up a meeting with our clients for next Monday to finalize design decisions, clear up questions about implementation, and go over plans for next term. I think were in a good place to complete our goals on schedule. For the rest of the week and next week we will be focusing on the Design Doc.
- Week 9: On Monday my team had a meeting with the clients. We got confirmation of our database and app mockups design. We got good feedback on the website as well. The client promised to think of a name for the app and website over the break. The team will work on the design document over break. We already divided up the sections and components. Next week we have another meeting planned with our clients where we will discuss their ideas and the design doc.

7.2.2 *Winter*

- Week 1: During the first week the team met to make plans for how to work on the project. We assigned parts of the project to team members and Bradford and I have done our parts. He created issues on github and placeholder screens for the app. I made the view submissions screen and the submission details screen with example submissions. We also confirmed a date with the TA for weekly meetings. Next week we will have 2 team meetings and the first TA meeting.
- Week 2: This week we have been having some problems with connecting to the database and Brandon has been having difficulty with the github repo. Somehow he is not using the right account but we will try to fix it by the

end of the week. We had a meeting with the client which just revealed more server connection issues that the client will look into. We will keep working on the app and website layouts so they are ready when the database is finished. We are also working on the elevator pitch and poster.

- Week 3: This week the initial elevator pitch and poster were due. Both need more work but I think they will get better once more of the project is complete and we can talk about our accomplishments instead of just our plans. The team also continued working on the app's SQLite database and saving submissions to it. I will be finishing that before our team meeting next Tuesday. On Thursday we met with our client Bill to work on the server and the MySQL database. We made some progress and Brandon will continue working on it. We are keeping up with our schedule and will most likely be ahead of schedule by next week.
- Week 4: The team has not had any problems. Our clients have been very helpful, we meet with them on a regular basis. The team has made steady progress on the app and server so we are on track to accomplish our goals for the alpha release. I am working on the SQLite database, which is new for me but I've made good progress.
- Week 5: This week the team accomplished a lot. We had a helpful poster critique session that gave us good ideas on how to improve our poster. I got the SQLite database working with Bradford's help so my parts of the alpha release are done. I plan to continue working on the SQLite database for the next few weeks. Brandon has finished some work on the database. Next the team will get started on the beta release requirements. The team also needs to complete our alpha release report and video.
- Week 6: The android app has the most work done, the server is almost ready and the website layout is ready to fill when the API from the client is ready. The paper report will be done by Monday. My goals for next week are to optimize the Android code, finish input checking for the website, and get the progress report done on time.
- Week 7: This week our client has made progress on the api and it is ready to connect to the MySQL database. Brandon will be populating the MySQL database so it will be complete by next week. Bradford and I are working on the android app, focusing on finishing the SQLite database, gathering all the data and displaying it in the correct locations. Bradford is working on the image capture portion, making sure we are sending the right files in the correct formats to the server. Our clients are leaving the country for a few weeks so we will be on our own for connecting all the pieces of the project for the end of term beta release. The style of the app and navigation are complete pending the review of the clients.
- Week 8: This week the team has worked on the android app. Brandon worked on completing the MySQL and SQLite database. Bradford worked on the picture saving features. I worked on collecting the submission data from inputs, storing them in the database, and displaying them on other screens. Our client is out of the country but he said the API was ready to test so hopefully we will hear from him soon about how to do that. Next we will be working on gathering and storing the rest of the data in the SQLite database. I will also be working on the website.
- Week 9: This week we are finishing up work with the android app and working on the website. Bradford and I had a few things to finish on the app and Brandon and I are working on the website. I put the website files I started on the server and Brandon is researching ways to connect to the api and server. Our clients are coming back next week and we will check in with them about our progress and get their approval on the work.
- Week 10: Our client returned and has worked on the api. It's still unclear if we will be able to implement it before the end of the term. The client is the only one who knows how the api is set up, so we are stalled until we can

work with him to get it connected. The team will finish our report and video this weekend and I think we will have plenty of time Spring term to complete the project to everyone's satisfaction.

7.2.3 *Spring*

- Week 1: This week the team met with the clients to discuss progress and start planning for the term. We made some changes to the database structure and the clients will come up with a logo for the app and website. The company hosting the server with the database and website is having problems so that is on hold until it comes back online. In the coming week I will be refactoring the SQLite database, Bradford will be fixing bugs, and Brandon will be documenting our changes.
- Week 2: We are working out the last bugs and making sure everything that we can is implemented.
- Week 3: The team completed and submitted the code for the code freeze. The team also completed the updated design and requirements documents with tables detailing the changes. The documents were combined and submitted as one PDF as well as added to the group github repo. The clients have approved the documents and changes and expressed satisfaction with the status of the project.
- Week 4: This week the team finalized the poster design and showed it to the clients for approval. The poster was updated with current screenshots of the app and website as well as a team photo. The team submitted model release forms for the expo. The team also discussed plans for how to present at expo including a video and phones with the app for visitors to tinker with.
- Week 5: This week the team did more work on the poster. We received some change requests from the client and fixed the formatting errors. The poster was approved and sent to the printers. The team has also been preparing for expo, making some improvements to the app and working towards stretch goals with the website. We have some old Android phones that will be on display running the app for visitors to use and we are testing to make sure the app is compatible with the older versions of Android.
- Week 6: This week the group is making final adjustments to the project for the Expo presentation. We are using old Android phones to demo the project and they all have old API levels and OS versions that create specific issues, mostly with the camera. We met with the clients and they are satisfied with our progress, they will be at the Expo with some fun things for the table.

7.3 **Bradford Wong**

7.3.1 *Fall*

- Week 3: This week I met with my group, met with our client, set up the group's Github repository, and wrote the project description. At this point, there aren't a whole lot of problems. However, we still have some tasks that we need to accomplish. This includes setting up a working agreement with the team and finalizing more specific details like drawing up the mocks for our Android app. Right now, we plan to start designing the mocks for the application, and then eventually showing them to the client.
- Week 4: This week I met with the group's TA, finished the group problem statement, and then went to office hours to look at sample requirement documents.
- Week 5: We met up and decided who is going to write what in the tech review. We also completed a draft of the requirements document and sent it to the client for feedback.

- Week 6: This week we did our tech review and met up with our client to discuss designs. There haven't been any problems so far.
- Week 7: This week we finished the tech review and talked more about our design for the app. There haven't been any problems so far.
- Week 8: This week our group met up and talked about the timeline and design document. There haven't been any major problems. We plan on meeting with the client next Monday to discuss further details.
- Week 9: This week we met with the client to talk about an updated version of our design for the app and the database. We also talked about the timeline for the project and what should be completed for each major date (alpha, beta, engineering expo). We plan to talk with the client again next week to finalize details regarding the design. We haven't had any significant problems.

7.3.2 *Winter*

- Week 1: This week, we made some work on the app by adding placeholder screens. The home screen also has all the buttons already. We haven't had any problems so far. We have plans to work on the database and the toolbar of the app. We also plan on looking into the server.
- Week 2: This week, we developed the view submissions screen further by adding the ability to delete an entry. We also worked on the SQLite database and the website. We did make some progress with the server, but we are still having problems with connecting to it and developing the database there. We plan on meeting again next week and trying to make more progress with the server.
- Week 3: This week, we worked on being able to add entries to the SQLite database. We are also finishing up the toolbar menu so that it navigates the user to different screens. We also worked more on the server, but we still haven't made much progress there. We worked with the client, but we can't connect to the server using MySQL Workbench. We plan on working more with the server next week.
- Week 4: This week, we were able to figure out how to connect to the MySQL server. However, we are still having problems with the databases. We plan on working on the database more next week and planning more on how exactly to receive input from the user that will eventually be placed in the database.
- Week 5: This week, we worked on the SQLite database and added a way to check internet connectivity. We plan on adding more fields in the submission form, further developing the database, and enabling pictures. We had some problems with reading from the database, but we were able to debug and fix it.
- Week 6: This week, we finished up adding submissions to the SQLite database. We also added the ability to upload pictures and track metadata. The client had difficulties developing the API, which will block us on some work. We plan on working more with the databases next week.
- Week 7: This week, we worked on the SQLite database and storing data into the database. We had problems with getting the full-size image of pictures. We plan on working with reading data from the database and displaying it on the screen next week.
- Week 8: This week, we worked on saving and reading data from the database. We did have some problems with the draft because we weren't sure about how we wanted to implement it. We are also still waiting on the API from the client, which will slow us down. Next week we plan on working more with the website.
- Week 9: This week, we worked on finishing up the drafts and connecting the website to the database on the server. We also finished the instructions and settings screen and also implemented login on the Android app. We

still don't have the API, and I don't think we will get it next week. I doubt we will be able to work on any of the API related features next week. Next week, we plan on finishing up the website.

- Week 10: This week, we worked with client on the API. We had problems because it was difficult to connect to the server and the API still isn't in a usable state for the Android application. We plan on updating the write-up and the video on Sunday.

7.3.3 *Spring*

- Week 1: This week we met with our client and fixed some bugs. The client talked about new features that they wanted, but we need to finish all the API related tasks before we can get to them. The server also went down, which slowed the client's development of the API. Next week we plan on fixing more bugs and trying to work with the API.
- Week 2: This week we worked on network connectivity, replies, and refactoring the database. We met up with our clients as well. The API still isn't finished, so we won't be completing the API related tasks before the code freeze. We plan on finishing up some loose ends of the project this weekend and then revising the documents next week.
- Week 3: Earlier this week, we finished up aspects of the project for the code freeze. For example, we fixed some bugs, implemented some new features, and made the app ready for the API once we get it. We did have some problems. We had difficulties with getting the build instructions for the mysql server because some of the users were accidentally deleted. The client is also still working on the API. Next week, we plan on working more with the API.
- Week 4: This week we revised our poster. We also got feedback from our client about the documents and made the requested revisions. I also looked at what the client has done so far with the API. I had some difficulties starting the server, but then I learned that I was trying to start it in a different directory. Next week, we can start working with the one part of the API that does work and maybe other small enhancements to the application.
- Week 5: This week, we fixed our poster and documents based on our clients' and Kirsten's comments. We also worked on incorporating parts of the API into our app. There were some problems with using a bearer token to authenticate the POST request, but I was able to get it working. The app can now send data about a submission from an Android phone to the database on the server. Next week, we will try working with more of the API (provided that there are more parts ready for us) and further improving the code that currently uses the API. We will also try to meet with the client to discuss use cases.
- Week 6: This week we met with our clients and went over some use cases. We had some problems because we discovered that using an actual phone produces bugs that we can't replicate on the emulator. We tried to fix some of them this week. We also plan on fixing more bugs next week before engineering expo and further integrating the API.

8 FINAL POSTER

COLLEGE OF ENGINEERING
Electrical Engineering and Computer Science
CS72




Fig. 1: OVDL Lab Personnel

OREGON VETERINARY DIAGNOSTIC LABORATORY

- The OVDL is a publicly supported facility at OSU and provides a range of animal disease diagnostic services.
- The OVDL wants to improve support of field necropsies (autopsies for non-human species performed away from the laboratory).
- A problem the OVDL is facing is there is not an effective way for personnel in the field to communicate with staff in the laboratories.
- The goal for this project is to replace the paper forms the OVDL has been using by creating a platform through which experts at the laboratory can provide guidance for field necropsies and sample collection.





Fig. 2: Necropsy Submission Form



MYVETPATH

A new mobile app to aid veterinary pathologists in collecting field data

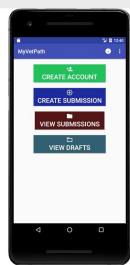
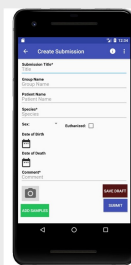





Fig. 3: Home Screen

Fig. 4: Create Submissions Screen

Fig. 5: View Submissions Screen

CLIENT SIDE

- App is fully native and can perform most operations without a network connection
- Creating Submissions to send to the OVDL or save as a draft to complete later
- Uses phone camera to capture and save pictures for submissions
- Users must enter login credentials before submitting for security
- Detailed instructions screen that teaches users how to use all app features

SERVER SIDE

- Data is stored in a relational MySQL database on an Ubuntu server.
- Data is transferred through a RESTful API.


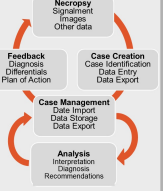



Fig. 6: Data Flow Chart Fig. 7: Pathologist Work Flow Chart

PROJECT DEVELOPMENT

- The team created a native Android application, "MyVetPath", that helps users create submissions with information and pictures, save submissions locally on the phone, and send the submissions to the OVDL.
- Users can send and receive messages about their submissions through the app.
- Pathologists can also write comments and feedback on a submission, which will be sent to the user who created the submission.
- The Android app was created with Android Studio using Java.
- The local Android database was created with SQLite and the server database with MySQL.
- We created the web application using HTML, CSS, Javascript, and Node.js.




Fig. 8: OVDL Entrance

TEAM MEMBERS AND CLIENTS

- Brandon Jolly (jollyb@oregonstate.edu)
- Bradford Wong (wongbra@oregonstate.edu)
- Katherine Jeffrey (jeffreys@oregonstate.edu)
- Clients: Christiane Loehr of the Oregon Veterinary Diagnostic Laboratory and Bill Baxter of WHB Consulting



Fig. 14: Group 72 Poster

9 PROJECT DOCUMENTATION

9.1 How Does the Project Work?

The project is a typical Android application. Android applications have various screens that a user can navigate to. Each screen is an activity, and an Android application is comprised of a series of linked activities. Each activity has a corresponding layout, which is a specification of what the contents of a screen should look like. All layout files can be found in the `/app/src/main/res/layout/` folder. The layout files begin with "content" and then contain the name of the activity they are associated with. The activity files themselves can be found in `/app/src/main/java/com/myvetpath/myvetpath`, and their file names end with "Activity.java". The code in these activity files are executed at run time when the user navigates to the corresponding screen. Android activities have various predefined methods that are overridden. For example, the "onCreate" method is a predefined method that is called when the user first navigates to that screen. This method is overridden in order for the application to initialize the screen.

The Android application also uses a SQLite database. The tables in the SQLite database can be found in `/app/src/main/java/com/myvetpath/myvetpath/data`, and the name of their files always ends in "Table.java".

Lastly, the project also uses a MySQL database. This database is saved on an Ubuntu server. An API sends data from the Android application to this server.

9.2 How to Install the Software

In order to install the software, clone the repository at <https://github.com/jeffreykat/MyVetPath>. Android Studio will also be necessary to develop the application, which can be found at <https://developer.android.com/studio>. To install Android Studio, click on the "Download Android Studio" and then run the installation wizard once it downloads. The wizard will then guide you through all of the installation steps. Once Android Studio install, you will also need to install a virtual device so that you can run the project. Click on the green "play button" at the top right and then click on the "Create New Virtual Device" button. Afterwards, select any phone on the list, click on "Next", then download a system image by clicking on "Download" next to the API you want, and then agree to the terms and download it.

9.3 How to Run the Project

Open Android Studio and then open up the cloned project (go to "File -> Open" and then select the cloned project). Once the project loads, there will be a green "play" button at the top right of Android Studio. Clicking on that button will open a window with a list of virtual devices that you want to run the project on. It is also possible to run the application on an actual phone by selecting that phone from the list (provided the phone is connected to the computer with a USB cable). Click on a an available device and then click on the "OK" button. The project will now run.

9.4 Hardware Requirements

It is recommended to have at least 8 GB of RAM when developing an Android application when using a virtual device. While the project should run on all devices, it may run differently depending on the device and its operating system. The team did most of the testing on a phone with an API level of 27.

9.5 API Documentation

At the time of writing this document, the API is still very incomplete. To get more information about the current status of the API, contact Bill Baxter, who is one of our clients and is the one who was developing it. The specifications of all working routes as of this moment that are relevant to the application are as follows:

- Post /users (create user). Input: "id", "first", "last", "email", "phone", "password", "createdAt", "updatedAt". Output: "message", "user", "token", "success"
- Get /users (Return user). Input: JWT. Output: "user": "id", "first", "last", "email", "phone", "password", "createdAt", "updatedAt", "success". This will return the user data for a single user.
- Get /users/all (return all users). Output: user: "id", "first", "last", "email", "phone", "password", "createdAt", "updatedAt", "success" for all users.
- Put /users (update user). Input: Any users fields except email or phone number. Output: "message", "success"
- Delete /users (delete user). Input: JWT, output: none.
- Post /users/login (User Login Authentication). Input: email or phone number, password. Output: "token", "user": id, first, last, email, phone, password, createdAt, updatedAt, "success"
- Post /submissions. Input: "submission_id", "User_id", "Group_id", "Case_id", "Title", "createdAt", "updatedAt".

10 BUILD INSTRUCTIONS/USER GUIDE

10.1 Install Android App with APK

1. Copy app-release.apk to Android phone storage.
2. On phone select apk file in storage and run install.
3. Check box to allow installations from unknown sources in Settings if not already checked.
4. Wait for installation to complete and open app.

We have done most of our testing using an Android emulator with an API level of 27. If you use an older phone, it is possible that the phone won't be able to use some of the features. If that happens to you, then try using Android Studio

Fig. 15: Script to install the Android App with APK

10.2 Alternative: Set Up Android Studio

1. Download and run the Android Studio installation setup wizard here: <https://developer.android.com/studio>
2. The wizard will ask you if you want to install "Android Studio" and "Android Virtual Device". Make sure both boxes are checked and click "Next"
3. The wizard will then ask you where you want to install Android studio. Choose where you want to put it and click on "Next"
4. The wizard will ask you to choose a start menu folder. The default option should work fine. Click on "Next". The wizard will now start installing.
5. Once the wizard is finished, start Android Studio
6. The setup wizard will now ask you if you want a "Standard" or "Custom" setup for Android Studio. Select "Standard" and click on "Next"
7. The wizard will then ask you what UI theme you want. Choose whatever you prefer and click on "Next".
8. The wizard will ask you to verify the installation settings. Click on "Finish". Android Studio will now start downloading components.
9. Once all the components have finished downloading, click on "Finish".

Fig. 16: Script to install the Android App with Android Studio

10.3 Clone the GitHub Repository (can be found here: <https://github.com/jeffreykat/MyVetPath>)

10. Open up a terminal and navigate to the directory you want to clone the repository to
11. Clone the repository ("git clone <https://github.com/jeffreykat/MyVetPath.git>")

Fig. 17: Cloning the GitHub Repo

10.4 Open the Project

12. When Android Studio opens up for the first time, it will give you a list of options of what you can do. Click on "Open An Existing Project" (if the actual Android Studio opens up, then go to File -> Open...)
13. Find the directory where you cloned our repository (in step 11) and click the item that says "MyVetPath" (It should have the Android Studio icon next to it). Click on the "OK" button.
14. Android studio will take some time to sync all the files, run a build, and index. Wait until all these processes are done.
15. Click on the green play button near the top of Android Studio.
16. If you don't already have a virtual device, then you will need to install one (go to step 16.A). If you do already have one, skip to step 17. A. Click on the "Create New Virtual Device" button B. Select any phone on the list (such as the Nexus 5x) C. Click on "Next" D. Download a system image by clicking on "Download" next to the API you want. We tested our application by downloading Pie (API level 28). Agree to the terms and download it. E. Once the components have downloaded, click on "Finish" F. Click on the system image you just downloaded and click on "Next" G. Click on the "Finish" button on the "Verify Configuration" page
17. Click on an available virtual device and then click on the "OK" button
18. The emulator will now launch and our application should automatically install and open up

It is possible that you might get problems when you first build the app (but everything should work fine the first time). Common fixes are to first clean the project (go to "build" -> "clean project") and then do a gradle resync (go to "file" -> "Sync Project With Gradle Files").

Fig. 18: Open the ProjectScript

10.5 Setup MySql Server

1. Run the Script MySQLInstall on a Ubuntu machine using Bash. Command: "bash MySQLInstall" (Found in this repo on the front page above Readme)
2. When prompted, "This operation will take X Space. Do you want to continue?" type, "y"
3. Once the install script is completed, start the mysql shell by typing, "sudo mysql -u root"
4. Once the mysql shell has started, copy the text from the file "Create Database SQL" (Found in this repo on the front page above ReadMe)
5. Paste and enter the text in the command line. (It should start running the sql)
6. Once the command is finished run, "SHOW DATABASES;"
7. After the database My_Vet_Path is displayed, then run the sql command, "USE My_Vet_Path;"
8. To see a list of the tables run the sql, "SHOW tables;"
9. To see the create statements for a table, run the sql, "SHOW CREATE TABLE `tableName`;"

Fig. 19: Run the MySQL Script

11 RECOMMENDED TECHNICAL RESOURCES

11.1 Helpful Websites

- <https://stackoverflow.com>
- <https://developer.android.com>

11.2 Helpful People on Campus

Rob Hess was a professor that taught a class on Android development. His lessons were very helpful to the team.

12 CONCLUSIONS AND REFLECTIONS

12.1 Brandon Jolly

- What technical information did you learn?
 - I learned how to install a database and the tables inside the database by using a Bash script. Something I have never done before. Most of the information I gained was android code. Before this takes I had not used android studio or any app development software. Over the course of this year I had gained a basic knowledge of how you develop apps and some debugging techniques.
- What non-technical information did you learn?
 - I learned about the process of developing a project from start to finish. Before I only knew the theory about how to create a project with a database and only then it was a rough idea. Now I had worked on a project from start to finish. Something I hadn't done in the past.
- What have you learned about project work?
 - Working with a properly document design document could solve a lot of later issues. For example with our group their was some confusion on what data types to used for storing images. In our planing stage we had all of the other data types pinned down, which allowed us to implement them early on in the process. After a group meeting we were able to solve the picture issue, but it was something we could have planned for earlier.
- What have you learned about project management?
 - If we have a schedule written out it helps with the timing of tasks. By doing this we were able to meet our goals for the project, and it allowed us to not go over board with the features.
- What have you learned about working in teams?
 - Communication can solve a lot of issues. There was not really any ambiguity in who was assigned a task. Another important thing I learned was do not be afraid to ask for help. The team wants to succeed and asking for help is not shame full. In fact, it actually makes the product better for it.
- If you could do it all over, what would you do differently?
 - I think including the client in some more meetings would have solved a lot of headaches. During the end of winter term one of our functions was not working and we had failed to communicate to the client when our winter term deadline was. He didn't know we had one so he was working on other projects. Once we informed him we were able to get some work done but it was only a partial of the end goal.

12.2 Katherine Jeffrey

- What technical information did you learn?
 - I furthered my knowledge of Android development including SQLite databases, the Room persistence library, API level differences, and new UI features. I also learned about setting up a server.
- What non-technical information did you learn?
 - I learned about the Oregon Veterinary Diagnostics Lab and the work they do there. I had not heard of it before this project. The lab and the staff there provide important services for the community and the world and I was happy to help them on this project.
- What have you learned about project work?
 - It's important to have clear goals for the project from the beginning. The people working on the project should know what they are responsible for at all times and be aware of the time constraints on the project. Having a consistent method of communication between all people on the project is also important.
- What have you learned about project management?
 - It's good to make detailed plans and create a schedule early. Then the team needs to stick to the schedule, but also expect delays. Feature creep is a real problem and it's best to put a stop to it early by clearly defining the project requirements and being aware of how much time the work will take.
- What have you learned about working in teams?
 - This team was excellent at communicating and participating which is essential for a team to function effectively. The team was also able to work democratically, no member tried to enforce their ideas over another's. If this had not been the case it would have been helpful to have conflict resolution strategies agreed upon in advance.
- If you could do it all over, what would you do differently?
 - I would have asked for more information from the clients sooner to inform the design process early on. Too many important decisions were made late in the year and this could have been avoided if we had a better understanding of what the clients wanted from the start. It also would have been helpful to have a larger team because there was a substantial amount of work to be done and we could have delivered a much better product with more people working on it.

12.3 Bradford Wong

- What technical information did you learn?
 - I learned a lot about how to make an Android application. For example, I learned about activities and the activity lifecycle, various ways of laying out the UI on the screen, user preferences, working with XML files, using image assets, how to work with a SQLite database, and how to use the camera and gallery to get pictures. Additionally, I learned how to use Android Studio's debugger to find and fix bugs. I also learned about APIs. I learned about what they were and how to make API requests through Postman and

an Android application. I learned about how to use Retrofit, an HTTP client, to specify a request's header and body in order to make an API request in the Android application.

- What non-technical information did you learn?
 - I learned a lot about how to effectively communicate information. For example, I got better at communicating with my teammates and clients as time went on. I also learned about how to create documentation for a project. For example, I learned about how to write a design document and how to effectively explain the design of a software project.
- What have you learned about project work?
 - I learned about working with multiple branches and how to resolve merge conflicts. I also learned about the importance of thinking about edge cases and potential ways to make the project crash. I learned this because some bugs went unnoticed for a while since we weren't thorough enough in our testing. If we were more thorough and thoughtful during testing, then we would have reduced the number of bugs in the project.
- What have you learned about project management?
 - I learned about the importance of carefully planning the project and always trying to think about the future. It is important to think about how certain tasks might be blocked until other tasks are completed first. As a result, we had to think about which tasks needed to be completed first. I also learned about how there can be many unexpected issues that occur during the development process. For instance, setting up the server took us much longer than we expected. As a result, I learned that you have to be flexible during the development process and be ready to work on other tasks that weren't originally assigned to you during the sprint planning meeting. Lastly, I learned about how easily feature creep can be introduced during the development process and how to deal with it.
- What have you learned about working in teams?
 - I learned about how often merge conflicts happen and how to effectively resolve them. I also learned the importance of writing lots of comments, because there were times when people had difficulty understanding the code that a different teammate wrote. In addition, I learned about how important good communication is in general, because there were also times when team members were confused when they saw changes to the application that weren't communicated to them.
- If you could do it all over, what would you do differently?
 - I would definitely start testing with actual phones sooner. We didn't start testing with actual phones until spring term, and we found numerous bugs that existed on the phones but couldn't be replicated on the emulator. As a result, we had to spend a lot of time fixing those bugs before the Engineering Expo. I would also try to be more careful about feature creep. The client and team kept coming up with new ideas for the project, but it was unrealistic to think that we could develop all of these new features within our time frame. I would make sure that the client and team both understand what the expectations for the project are and what can be reasonably be done within the time frame.

Fig. 20: XML File of Main Activity (Home Screen)

```

5  @Override
6  protected void onCreate(Bundle savedInstanceState) {
7      super.onCreate(savedInstanceState);
8      setContentView(R.layout.activity_main);
9      Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
10     setSupportActionBar(toolbar);
11     getSupportActionBar().hide();
12
13     //Initialize buttons and set the on click listeners
14     create_sub_activity = new Intent( packageContext: this, CreateSubActivity.class);
15     view_subs_activity = new Intent( packageContext: this, ViewSubsActivity.class);
16
17     create_sub_button = findViewById(R.id.createSubButton);
18     create_sub_button.setOnClickListener((view) -> {
19         startActivity(create_sub_activity);
20     });
21
22     viewModel = ViewModelProviders.of( activity: this ).get( MyVetPathViewModel.class );
23
24     create_account_activity = new Intent(Intent.ACTION_VIEW, Uri.parse("https://myvetpath.com/register.html"));
25     create_account_button = findViewById(R.id.createAccountButton);
26     create_account_button.setOnClickListener((view) -> {
27         if (create_account_activity.resolveActivity(getPackageManager()) != null) { //if there is a web browser on phone, go ahead and navigate to that page
28             startActivity(create_account_activity);
29         }
30     });
31
32     view_subs_button = findViewById(R.id.viewSubsButton);
33     view_subs_button.setOnClickListener((view) -> {
34         startActivity(view_subs_activity);
35     });
36 }

```

Fig. 21: Main Activity's onCreate Code

The code snippet above is the Java code for the Main Activity, which represents the home screen of the application. The onCreate function is an essential method in all Android activities because this is the method that initializes the screen its variables. For example, the code snippet above shows the toolbar being initialized in lines 62-64. Additionally, the snippet shows how to implement navigation within an Android application. Line 68 initializes an intent that will navigate to the ViewSubsActivity screen. On line 91, the code finds the view_subs_button that was created in Main Activity's XML file. Line 92 assigns an onClickListener to the button, meaning that the code inside of that block of code is executed anytime a user clicks on the button. In this case, the only code that is executed is the code in line 95, which is what will actually navigate the user to the ViewSubsActivity screen.

```

629 //This will use the API (which we don't have at the moment) to send the submission to the server. Call this function after clicking on "submit"
630 private void sendToServer() {
631     String MyVetPath_Base_Url = "208.113.134.137:8000/v1";
632     //build the retrofit that will make the query
633     Retrofit.Builder builder = new Retrofit.Builder()
634         .baseUrl("http://208.113.134.137:8000/v1/")
635         .addConverterFactory(GsonConverterFactory.create());
636
637     Retrofit retrofit = builder.build();
638     MyVetPathAPI userClient = retrofit.create(MyVetPathAPI.class);
639
640     //Send the Submission
641     Call<SubmissionTable> call = userClient.submission(newSub);
642     call.enqueue(new Callback<SubmissionTable>() {
643         @Override
644         public void onResponse(Call<SubmissionTable> call, Response<SubmissionTable> response) {
645             if(response.isSuccessful()){//if the submission was successfully sent to server, then update the submission in the SQLite DB
646                 newSub.Submitted = Calendar.getInstance().getTime().getTime();
647                 //todo: set case_id when the api has been further developed
648                 viewModel.updateSubmission(newSub);
649                 Toast.makeText(context, CreateSubActivity.this, text: "Submission was sent to server", Toast.LENGTH_SHORT).show();
650             }else{
651                 Toast.makeText(context, CreateSubActivity.this, text: "Error: Submission wasn't sent.", Toast.LENGTH_SHORT).show();
652             }
653         }
654     });
655
656     @Override
657     public void onFailure(Call<SubmissionTable> call, Throwable t) {
658         Toast.makeText(context, CreateSubActivity.this, text: "Failed to send to server. Check your internet connection and try again.", Toast.LENGTH_SHORT).show();
659     }
660 }

```

Fig. 22: Code that Makes a POST Request Using the API

The code above shows how to send a submission to the server's database by making a POST request. The Android application uses Retrofit, which is an HTTP client that is used to make API requests. The code in lines 631-638 sets up Retrofit. Lines 641-660 is the code that actually sends the submission to the server. The onResponse method is called after getting a response back from the server. If the response was successful, then the application will update the submission and display a message to the user saying that submission was sent successfully. If the server sent an error back, then the application will just display an error message instead. The application can receive a failure response if something was wrong about the API call. For example, if the POST request went to the wrong route, the Bearer Token was incorrect, or if server was expecting a different body of data. The API is what sends these failure responses. the onFailure method is called whenever the application fails to make the API request and doesn't get a response from the server. This typically happens when the server isn't down or if the user isn't connected to the internet.

```

55 //the bearer token expires after a set time. If you get an unauthorized response,
56 // then post user again and take the bearer token sent in response.
57 // We will need to figure out a better way to handle this when we get login
58
59 @Headers({"Content-Type: application/json",
60         "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9." +
61         "eyJ1c2VyX2lkIjoxMywiaWF0IjoxNTU3OTY3NjMwLCJleHAiOiJlNTgwNTQwMzB9." +
62         "SArgvMg9PcDm6KaFLlRrdANWAYK3KlXhvyIDuxTj-X8"})
63 @POST("submissions")
64 Call<SubmissionTable> submission(@Body SubmissionTable submission);

```

Fig. 23: Code that Defines the API Request

The code above shows how to specify a POST request using the API. The header should always have "application/json" as the content-type. The authorization is the Bearer Token that is assigned whenever a user is created. The "submissions" part following the "@Post" part is the route. Finally, the body of the API call is just the class that is being sent to the server, which is "SubmissionTable" in this example.

```

409 //purpose: this method is called automatically after the app gets a picture from the UploadPicturesPrompt
410 @Override
411 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
412     super.onActivityResult(requestCode, resultCode, data);
413
414
415     if (chosen_method == 1) { //user uploaded a picture using the camera
416         File file = new File(mCurrentPhotoPath);
417         picturePaths[selectedImageView] = mCurrentPhotoPath;
418
419         Bitmap bitmap = null; //get the actual picture
420         try {
421             bitmap = MediaStore.Images.Media.getBitmap(this.getContentResolver(), Uri.parse(mCurrentPhotoPath));
422             bitmap = modifyOrientation(bitmap, mCurrentPhotoPath.substring(5));
423
424         } catch (IOException e) {
425             e.printStackTrace();
426         }
427         images[selectedImageView].setImageBitmap(bitmap);
428         imageBitmaps[selectedImageView] = bitmap;
429         imageDates[selectedImageView] = Calendar.getInstance().getTime().getTime(); //get the current date
430         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) { //Check for permissions
431             if (ActivityCompat.checkSelfPermission(context, this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
432             && ActivityCompat.checkSelfPermission(context, this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
433                 requestPermissions(new String[] {Manifest.permission.ACCESS_COARSE_LOCATION,
434                     Manifest.permission.ACCESS_FINE_LOCATION, Manifest.permission.INTERNET}
435                     , requestCode, 10);
436             }
437             return;
438         }
439     }
440

```

Fig. 24: Code that Gets a Picture (Part 1)


```

440 //Get the coordinates of the phone. won't be called if the phone doesn't have permission
441 locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, minTime: 5000, minDistance: 10, listener);
442
443 shouldCheckPhoneCoordinates = true;
444 Log.d( tag: "filepath", msg: "onActivityResult: ");
445 }else if(chosen_method == 0){ //The user chose to upload a picture using the gallery
446     shouldCheckPhoneCoordinates = false;
447     if (requestCode == PICK_IMAGE_REQUEST && resultCode == RESULT_OK && data != null && data.getData() != null) { //check if the data is valid
448         Uri uri = data.getData();
449         String picturePath = getPath( this.getApplicationContext(), uri ); //this is the path of the file in the SD card
450         picturePaths[selectedImageView] = picturePath;
451         //Get and store the image name
452         File f = new File(picturePath);
453         imageNames[selectedImageView] = f.getName();
454
455         //Get and store the GPS coordinates
456         InputStream in = null;
457         try {
458             in = getContentResolver().openInputStream(uri);
459             String col = MediaStore.Images.ImageColumns.DATA;
460             Cursor c = getApplicationContext().getContentResolver().query(uri,
461                 new String[] {col},
462                 selection: null, selectionArgs: null, sortOrder: null);
463             ExifInterface exifInterface = null;
464             if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.N) {
465                 exifInterface = new ExifInterface(in);
466             } else {
467                 if (c != null && c.moveToFirst()) {
468                     exifInterface = new ExifInterface(c.getString(c.getColumnIndex(col)));
469                     c.close();
470                 } else {
471                     exifInterface = new ExifInterface(uri.getPath());
472                 }
473             }

```

Fig. 25: Code that Gets a Picture (Part 2)

```

475 String longitude = exifInterface.getAttribute(ExifInterface.TAG_GPS_LONGITUDE);
476 String latitude = exifInterface.getAttribute(ExifInterface.TAG_GPS_LATITUDE);
477 String latitudeRef = exifInterface.getAttribute(ExifInterface.TAG_GPS_LATITUDE_REF);
478 String longitudeRef = exifInterface.getAttribute(ExifInterface.TAG_GPS_LONGITUDE_REF);
479 String dateString = exifInterface.getAttribute(ExifInterface.TAG_DATETIME);
480
481 //The returned date comes in as a string, so convert it into date
482 SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "yyyy:MM:dd hh:mm:ss");
483 Date convertedDate = new Date();
484 try {
485     if(dateString != null) {
486         convertedDate = dateFormat.parse(dateString);
487     }
488     imageDates[selectedImageView] = convertedDate.getTime();
489 } catch (ParseException e) {
490     // Auto-generated exception
491     e.printStackTrace();
492 } catch (java.text.ParseException e) {
493     //Auto-generated exception
494     e.printStackTrace();
495 }

```

Fig. 26: Code that Gets a Picture (Part 3)

```

497         if(latitudeRef != null) {
498             if (latitudeRef.equals("N")) { //if the degrees should be positive
499                 latitude = Float.toString(convertToDegree(latitude));
500             } else { //If the degrees should be negative
501                 latitude = Float.toString(0 - convertToDegree(latitude));
502             }
503             if (longitudeRef.equals("N")) { //if the degrees should be positive
504                 longitude = Float.toString(convertToDegree(longitude));
505             } else { //else the degrees should be negative
506                 longitude = Float.toString(0 - convertToDegree(longitude));
507             }
508         } else {
509             latitude = "";
510             longitude = "";
511         }
512
513         longitudes[selectedImageView] = longitude;
514         latitudes[selectedImageView] = latitude;
515     } catch (IOException e) {
516         // Handle any errors
517     } finally {
518         if (in != null) {
519             try {
520                 in.close();
521             } catch (IOException ignored) {}
522         }
523     }
524 }

```

Fig. 27: Code that Gets a Picture (Part 4)

```

525         try { //set and save the actual image
526
527             Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(), uri);
528
529             images[selectedImageView].setImageBitmap(bitmap);
530             imageBitmaps[selectedImageView] = bitmap;
531
532         } catch (IOException e) {
533             e.printStackTrace();
534         }
535     }
536 }
537 }

```

Fig. 28: Code that Gets a Picture (Part 5)

The five figures above show a picture is obtained in the "AddPicturesActivity" screen. This method is called automatically whenever the user selects a picture from gallery or takes a picture from the camera. If the user took a picture with the camera, then the application will execute lines 416-444. Lines 416-426 is what gets the actual picture and stores it into a bitmap. Some phones have an issue where taking a picture automatically rotates or flips it. To deal with this, line 422 calls a function that simply rotates it back to normal. Lines 427-428 saves the picture into the variables in the class, which will actually display the picture on the screen. Lines 429-444 obtains the metadata of the picture such as GPS coordinates and the time it was taken. Lines 430-435 checks to see if the phone has permission to get the location metadata.

If the user selected a picture from the gallery, then the code from lines 446-536 will be called. The application first checks to see if it has permission and if the data that came back is valid in line 447. Lines 448-453 is what gets the picture. Lines 456-523 is where the application tries to get the metadata of the picture. Finally, lines 525-534 gets the bitmap of the picture and saves it into the class' variables.

14 APPENDIX 2

REFERENCES

- [1] Sun microsystems announces completion of mysql acquisition; paves way for secure, open source platform to power the network economy. [Online]. Available: <https://web.archive.org/web/20080228025123/http://www.sun.com/aboutsun/pr/2008-02/sunflash.20080226.1.xml>
- [2] O'reilly mysql ce 2010: Mark callaghan, MySQL at Facebook. [Online]. Available: <https://www.youtube.com/watch?v=Zofzid6xIZ4>
- [3] S. Reigns. 11 best php frameworks for modern web developers in 2018. [Online]. Available: <https://coderseye.com/best-php-frameworks-for-web-developers/>
- [4] D. Kerby. Why mongodb is the way to go - dzone database. [Online]. Available: <https://dzone.com/articles/why-mongodb-is-worth-choosing-find-reasons>
- [5] Minewiskan. Data mining concepts. [Online]. Available: <https://docs.microsoft.com/en-us/sql/analysis-services/data-mining/data-mining-concepts>
- [6] Archiveddocs. Overview (service broker). [Online]. Available: [https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms166104\(v%3dsql.105\)](https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms166104(v%3dsql.105))
- [7] Laravel - the php framework for web artisans. [Online]. Available: <https://laravel.com/>
- [8] M. James. (2015, November) Visual studio v android studio. [Online]. Available: <https://www.i-programmer.info/news/193-android/9163-visual-studio-v-android-studio.html>
- [9] H. Fakhruddin. (2015, March) Eclipse vs android studio: Which ide is better for android developers? [Online]. Available: <https://teks.co.in/site/blog/eclipse-vs-android-studio-ide-better-android-developers/>
- [10] J. Mueller. (2013, January) Understanding soap and rest basics and differences. [Online]. Available: <https://smartbear.com/blog/test-and-monitor/understanding-soap-and-rest-basics/>
- [11] A. Monus, "Soap vs rest vs json comparison [2018]," *Raygun*, August 2018.
- [12] A. Developers. (2018, August) Camera api. [Online]. Available: <https://developer.android.com/guide/topics/media/camera#custom-camera>
- [13] Microsoft. (2018) Camera class. [Online]. Available: <https://developer.xamarin.com/api/type/Android.Hardware.Camera/>
- [14] Mitek. Mitek misnap. [Online]. Available: <https://www.miteksystems.com/mobile-capture>
- [15] Facebook. React native. [Online]. Available: <https://facebook.github.io/react-native/>
- [16] N. Chrzanowska. (2017, August) React native - pros and cons of facebook's framework. [Online]. Available: <https://www.netguru.co/blog/react-native-pros-and-cons>
- [17] Flutter. [Online]. Available: <https://flutter.io/>
- [18] A. Mroczkowska. (2019, May) Flutter in mobile app development pros risks for app owners. [Online]. Available: <https://www.thedroidsonroids.com/blog/flutter-in-mobile-app-development-pros-and-cons-for-app-owners/>
- [19] Showcase. [Online]. Available: <https://flutter.dev/showcase>
- [20] U. Khan. (2018, June) The pros and cons of native apps. [Online]. Available: <https://clutch.co/app-developers/resources/pros-cons-native-apps>
- [21] TutorialsPoint. Mobile testing - robotium framework. [Online]. Available: https://www.tutorialspoint.com/mobile_testing/mobile_testing_robotium_framework.htm
- [22] H. Noon. (2017, November) An introduction to appium: The best open source mobile app automation tool. [Online]. Available: <https://hackernoon.com/an-introduction-to-appium-the-best-open-source-mobile-app-automation-tool-940f5eaabae9>
- [23] TutorialsPoint. Mobile testing - appium framework. [Online]. Available: https://www.tutorialspoint.com/mobile_testing/mobile_testing_appium_framework.htm
- [24] A. Developers. Ui automator. [Online]. Available: <https://developer.android.com/training/testing/ui-automator>
- [25] N. Minaev. (2017, September) The top 5 android ui frameworks for automated testing. [Online]. Available: <https://saucelabs.com/blog/the-top-5-android-ui-frameworks-for-automated-testing>
- [26] J. Chakraborty. (2015, August) Top 5 android testing frameworks. [Online]. Available: <https://www.linkedin.com/pulse/top-5-android-testing-frameworks-jaybrata-chakraborty/>
- [27] K. Jackowski. (2018, February) Kotlin vs. java: Which one you should choose for your next android app. [Online]. Available: <https://www.netguru.com/blog/kotlin-java-which-one-you-should-choose-for-your-next-android-app>

- [28] J. Paul. (2018, February) Java vs. kotlin - which language should android developers start with? [Online]. Available: <https://dzone.com/articles/java-vs-kotlin-which-language-android-developer-sh>
- [29] C. Labs. Corona. [Online]. Available: <https://coronalabs.com/>
- [30] A. Sinicki. (2017, December) I want to develop android apps what languages should i learn? [Online]. Available: <https://www.androidauthority.com/develop-android-apps-languages-learn-391008/>