

Birkbeck College, University of London
Department of Computer Science and Information Systems

MSc Computer Science
Project Report

PATHOGEN TRACKER

AN ANDROID APP TO ASSIST WITH DATA CAPTURE
FOR PATHOGEN RESEARCH PROJECTS

Author: Laurent Mignot
Supervisor: George Roussos

17 September 2018

This report is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service. This report may be freely copied and distributed provided the source is explicitly acknowledged.

TABLE OF CONTENTS

ABSTRACT	8
1 BACKGROUND AND INTRODUCTION	9
1.1 Report structure.....	9
1.2 Background.....	9
1.2.1 Virus vector research.....	9
1.2.2 Obstacles to data gathering and analysis: distance & accuracy.....	10
Distance.....	10
Accuracy.....	10
1.3 Introduction.....	11
1.3.1 Project aims.....	11
1.3.2 Project scope.....	11
2 SOLUTION OVERVIEW.....	12
3 REQUIREMENTS AND SPECIFICATIONS.....	14
3.1 Requirement triage.....	15
3.2 Categorized requirements	15
3.2.1 Server.....	15
3.2.2 PT Android application requirements.....	16
3.3 PT app specifications	16
3.3.1 Authentication.....	16
3.3.2 User preferences	17
3.3.3 First-time use configuration.....	17
3.3.4 Survey capture	17
1. Mosquito Batch.....	17
2. Mosquito	18
3. Patient.....	18
3.3.5 Survey review and modification.....	18
3.3.6 Survey upload	18
4 SCHEDULE.....	19
4.1 Stage one schedule.....	19
5 TOOLS AND LIBRARIES.....	21
5.1 Investigating and selecting an Android programming language.....	21
5.2 Tools and libraries	22
5.2.1 Development environment	22
5.2.2 Version control.....	22
5.2.3 Third-Party libraries.....	22
Effortless Permissions	23
Location Manager	23
Realm Database, Kotlin extensions	24
Android Material Stepper	24

	Anko	24
6	ARCHITECTURE.....	25
6.1	A brief introduction to Android components.....	25
6.1.1	Activity	25
6.1.2	Service	25
6.1.3	Intent.....	25
6.2	Architectural patterns	26
6.2.1	Model View Controller (MVC).....	26
	Model.....	26
	View.....	26
	Controller.....	26
6.2.2	Model View Presenter (MVP)	27
	Model.....	27
	View.....	27
	Presenter.....	27
6.2.3	Model View ViewModel (MVVM).....	27
6.2.4	Architecture choice	28
6.2.5	Repository pattern.....	28
6.3	PT app structural design	29
6.3.1	Authentication and first-time use.....	30
6.3.2	Show a list of surveys.....	32
6.3.3	Configure user preferences	33
6.3.4	Add surveys	34
	Add a patient survey	34
	Add a mosquito batch survey	36
	Add a mosquito detail survey.....	37
6.3.5	Review surveys	39
	Patient survey review.....	39
	Mosquito batch survey review	39
	Mosquito survey review	39
6.3.6	Background survey upload.....	40
6.4	PT app database layer	41
6.4.1	Client-side: Realm Database.....	41
6.4.2	Server-side: Firebase Cloud Firestore.....	42
7	IMPLEMENTATION	43
7.1	Development process.....	43
7.2	Android SDK.....	43
7.3	Dependency Injection and App class.....	44
7.4	Interfaces and abstractions.....	44
7.5	Resources: strings, layout and graphical icons.....	45
7.6	UI design.....	45
7.7	Kotlin language features	46

7.7.1	Sealed classes: algebraic data types.....	46
7.7.2	Extensions.....	46
7.7.3	Data classes.....	46
7.8	Authentication	47
7.9	Onboarding	47
7.9.1	Authentication.....	47
7.9.2	First-time configuration.....	47
7.10	Surveys	48
7.10.1	Survey item list.....	49
7.10.2	Survey transmission	50
	JobScheduler	50
	PT app services.....	51
7.11	BaseSurveyActivity	51
7.12	Recording a Patient survey	52
7.12.1	LocationView.....	52
7.12.2	MultiSpinner.....	52
7.12.3	Add a current disease entry.....	53
7.13	Recording a Mosquito Batch survey	54
7.14	Recording a Mosquito detail survey.....	55
7.15	Reviewing surveys.....	56
7.16	Preferences.....	57
8	TESTING.....	58
8.1	Behaviour driven development.....	58
8.2	PT app user stories	59
8.2.1	Restrict access to authorized users	59
	Narrative.....	59
	Scenarios	59
8.2.2	Allow a user to sign out.....	59
	Narrative.....	59
	Scenarios	59
8.2.3	First time user should grant device location permission.....	60
	Narrative.....	60
	Scenarios	60
8.2.4	First time user should grant device camera and storage permissions.....	60
	Narrative.....	60
	Scenarios	60
8.2.5	First time user should be notified about device encryption	60
	Narrative.....	60
	Scenarios	60
8.2.6	User should be encouraged to record surveys	61
	Narrative.....	61
	Scenarios	61

8.2.7	User can select survey type(s).....	61
	Narrative.....	61
	Scenarios	61
8.2.8	User can choose upload via metered/unmetered network.....	63
	Narrative.....	63
	Scenarios	63
8.2.9	User can choose photo optimization.....	64
	Narrative.....	64
	Scenarios	64
8.2.10	Capture location in survey	65
	Narrative.....	65
	Scenarios	65
8.2.11	Capture photo in survey	66
	Narrative.....	66
	Scenarios	66
8.2.12	Assign survey date to survey automatically.....	67
	Narrative.....	67
	Scenarios	67
8.2.13	Change survey date in survey	67
	Narrative.....	67
	Scenarios	67
8.2.14	Assign unique survey identifier	68
	Narrative.....	68
	Scenarios	68
8.2.15	Select one data value within restricted set.....	68
	Narrative.....	68
	Scenarios	68
8.2.16	Select multiple data values within restricted set.....	69
	Narrative.....	69
	Scenarios	69
8.2.17	Save a survey	69
	Narrative.....	69
	Scenarios	69
8.2.18	Record a patient survey.....	70
	Narrative.....	70
	Scenarios	70
8.2.19	Record a mosquito batch survey.....	70
	Narrative.....	70
	Scenarios	70
8.2.20	Record a mosquito detail survey and assign to batch.....	71
	Narrative.....	71
	Scenarios	71

8.2.21	List all saved surveys.....	71
	Narrative.....	71
	Scenarios	71
8.2.22	Review saved surveys.....	72
	Narrative.....	72
	Scenarios	72
8.2.23	Securely transmit survey to server.....	73
	Narrative.....	73
	Scenarios	73
8.2.24	Securely transmit survey photos to server.....	74
	Narrative.....	74
	Scenarios	74
8.3	Unit and Instrumentation Testing.....	75
8.3.1	Unit tests.....	75
	Boolean Extensions.....	75
	OnBoardingViewModel.....	75
8.3.2	Instrumentation and Integration tests.....	76
	First time user	76
	Surveys list.....	77
	SurveysLlistViewModel	78
	Preferences.....	78
	Photo repositories.....	79
	Realm survey repository.....	79
	Firebase Firestore survey repository	80
	Add a Patient survey	80
	Add a Mosquito Batch survey	81
	Add a Mosquito detail survey.....	82
	Utility tests	82
8.4	User testing.....	83
8.4.1	Stage one user testing.....	83
8.4.2	Stage two user testing.....	83
	User testing results	84
	Defects	85
8.4.3	Summary.....	85
9	SUMMARY AND EVALUATION	86
9.1	Evaluation.....	86
9.2	Improvements	86
9.2.1	Mosquito survey: allow a DNA sample to be copied from a portable device.....	87
9.2.2	Mosquito survey: allow a photo to be selected from the device's photo gallery.....	87
9.2.3	Mosquito survey: allow a sound recording of a mosquito in flight	87
9.2.4	Download surveys from server	87
9.2.5	Identify authenticated user in surveys.....	87

9.2.6	Further validation in surveys	87
9.2.7	Data binding.....	88
9.2.8	Save survey state when PT app is paused.....	88
9.2.9	Allow surveys to be edited on the device.....	88
9.3	Summary.....	89
10	BIBLIOGRAPHY	90
11	TABLE OF FIGURES	94
APPENDIX A: USER JOURNEYS.....		95
	First-time use	95
	User only collects Mosquito surveys.....	96
	User only collects Patient surveys	97
	User collects both Patient and Mosquito surveys.....	97
	Survey Review (Patient, Mosquito Batch).....	98
	Survey Review (Mosquito detail).....	99

ABSTRACT

Zika Virus was declared a Public Health Emergency of International Concern by the World Health Organisation in February 2016. In recent years, a number of research projects have been launched to investigate Zika Virus, its vectors, origin and spread.

This project report summarizes work undertaken on 'Pathogen Tracker' (PT), an android-based app developed to solve the data collection challenges faced by researchers who operate in remote locations with negligible mobile connectivity. PT aims to facilitate data gathering surveys despite limited connectivity, while allowing for the data to be securely submitted to a central repository for analysis.

PT has been implemented as a proof of concept, and is not yet a production ready system.

1 BACKGROUND AND INTRODUCTION

1.1 Report structure

This report consists of four main sections:

- **Background and Introduction**, provides a recap of the proposal and a brief introduction to this project and its documentation.
- **Solution Overview** provides a high-level overview of how the solution handles the problem domains, the client-server architecture and the selected cloud-based backend service acting as the server.
- **PT app's implementation**, including: **Requirements and Specifications**, project **Schedule, Planning, Architecture, Implementation**, and **Testing**.
- **Summary** presents a critical analysis of the project, and proposes a set of improvements should further development be performed.

1.2 Background

This section briefly describes background research that gave rise to the development of PT.

1.2.1 Virus vector research

Zika Virus (ZIKV) was originally discovered in Uganda in 1947. It resurfaced with major outbreaks in 2007 and gained global notoriety after a large-scale outbreak in Brazil. Between October 2015 and May 2016 thousands of reported cases in Cape Verde led that country to declare a ZIKV epidemic.¹⁻⁵

*Zika virus surveillance in human and mosquito populations in Cape Verde*¹ or the Zika Project (ZP) is a research project at the London School of Hygiene & Tropical Medicine (LSHTM) investigating the ZIKV outbreak in Cape Verde. ZP and similar research programs⁷ collect and analyse data in order to map the origin, spread, and variants (including drug-resistance) of ZIKV and other mosquito-borne pathogens.

As "*pathogen genomes cannot be usefully analysed in isolation*"⁶ it is necessary to combine genomic data with relevant epidemiological metadata in order to extrapolate meaningful information.

1.2.2 Obstacles to data gathering and analysis: distance & accuracy

Distance

Researchers investigating pathogen origins and spread frequently perform on-site capture of physical samples, such as: DNA samples from mosquitos and human blood and urine samples. Physical samples are tagged before additional meta-data is collected. Samples are then transmitted to a central location for processing and analysis. ZIKV sampling sites are widely dispersed throughout the world⁸

ZP's team advise that their data collection and sampling is done in Cape Verde while their sample analysis is usually carried out in London and note that the accuracy and reliability of the data capture process is complicated by the consequent distance and time differences between data capture, data transmission and research and analysis.

Accuracy

Data can be collected in different ways, such as: paper forms, tagged serological samples, and digital entries in electronic systems. Accuracy in the data collected permits⁶ meaningful and reliable analysis. The possibility of human error must factor and so a robust process of validation that identifies and minimises errors is crucial to improving data accuracy.

1.3 Introduction

1.3.1 Project aims

The primary aim of this project is to provide a technological solution to those obstacles outlined in 1.1.2, above. To this end an Android application, Pathogen Tracker (PT app), was developed. PT app enables:

- the capture and secure storage of survey data on a smartphone device, and
- handles the secure transmission of data to a remote database system that
- quickly is made available for data analysis by researchers and
- is responsible for authenticating users to PT app, thereby restricting use to authorized individuals.

The project also aimed to learn from and take advantage of the knowledge and experience of the Android app development community in order to plan a well-designed Android app developed in line with current best practices for Android apps.

1.3.2 Project scope

The project was approached as a real-world project undertaken for a client (ZP). PT app's requirements and specifications were collected and prioritized in collaboration with ZP team.

Time constraints restricted the number of ZP's desired features that could be implemented and necessitated triage. The project chose to focus on: 1. security of data at-rest and in-transit, 2. ease of use, 3. offline capability, and 4. automated transmission to the internet-based system.

2 SOLUTION OVERVIEW

This chapter introduces the software solution developed during the project. Specifications and requirements are discussed in chapter 3.

The solution to the obstacles set out in 1.1.2 consists of an Android application (PT app) that enables users to capture surveys whether a device is connected to the internet or not. Users can record data for specified types of surveys and save them on a smartphone device. The device's geolocation capabilities and camera can be used to capture location information and take photographs. Surveys are automatically and securely transmitted to an online database once the device has re-acquired internet connectivity.

Figure 1 shows a high-level overview of how the application works.

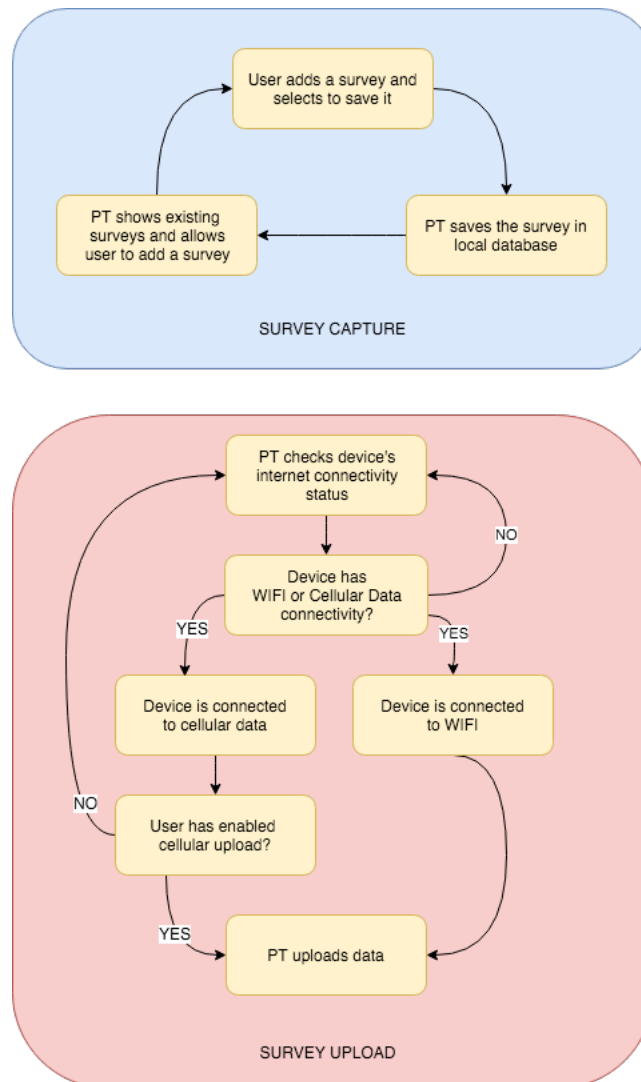


Figure 1 – High-level overview of survey capture and survey upload

Figure 2 presents a high-level view of the overall system architecture. Due to time constraints the cloud-based server layer uses Firebase as a third-party provider⁹. Firebase is a Software as a Service¹⁰ solution that provides a number of application backend services, of which the following are used:

- Firebase Authentication¹¹ (identity)...
handles the system's security. Only approved users can sign in to the app and capture surveys.
- Firebase Cloud Firestore¹² (database)...
handles the database solution, and securely receiving and storing the survey data transmitted by PT.
- Firebase Cloud Storage¹³ (file storage)...
enables photos to be captured and attached to surveys. These are then securely uploaded to Firebase Cloud Storage.

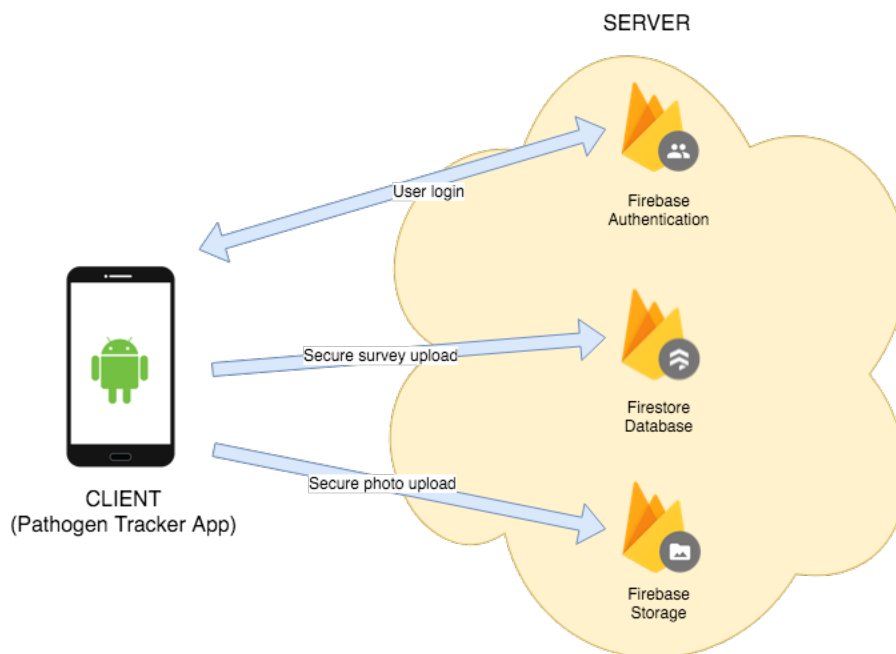


Figure 2 – High-level overview of project architecture

3 REQUIREMENTS AND SPECIFICATIONS

Communication with ZP team determined the particular challenges faced by their researchers in order to propose a tailored software solution.

The requirement-gathering phase of this project was accomplished in two phases:

1. **Phase one** involved written and telephone communication with ZP team, during which they provided a list of desired functionalities. From that list, a set of initial functional and non-functional requirements was compiled, enough to proceed to phase two.
2. **Phase two** involved an interactive prototype produced with the assistance of MarvelApp¹⁴. MarvelApp is a web-based application for creating interactive prototypes of a user interface (UI). Graphics representing the various screens and components of the images are uploaded and, using MarvelApp software, links are added between the image. These links are selected areas of the image that can then be interacted with by a user. A user can tap or click on a link and the app will display the link's target image. This allows a degree of navigation between the various images and so simulates a user's journey through the application. The resulting prototype can then be run in a web browser or directly on a smartphone.

ZP team then used and interacted with the prototype and provided comments and observations on the proposed user journeys and overall UI. ZP team's input informed the production of a second and final version of the prototype for further testing.

A final phone conversation resulted in an agreed set of requirements and specifications which were then prioritized.

The user journeys created as part of this process are included in Appendix A. The images used in the interactive prototypes are included with the application source code attached to this report.

3.1 Requirement triage

Due to time constraints, requirements were triaged into “Must Have” (MH) and “Could Have” (CH) categories¹⁵. Requirements deemed absolutely necessary to PT app's success and feasible to implement within the allowed time frame were categorised MH. Requirements deemed useful yet not absolutely necessary for proof of concept were categorised CH.

3.2 Categorized requirements

Due to time constraints further triage became necessary, resulting in certain MH requirements changing to CH status. These changes are documented in chapter 9.

The categorisation of PT app's requirements is set out in 3.2.1 below:

3.2.1 Server

Requirement	Categorization
Add an authorized user to the system	MH
Allow an authorized user to sign in to the system	MH
Allow survey data to be uploaded and securely stored in a database	MH
Allow survey photos to be uploaded and stored securely	MH
Restrict survey transmission to authorized and signed-in users	MH
All data transmission to and from system must be secured	MH

3.2.2 PT Android application requirements

Requirement	Categorization
Allow a user to sign in	MH
Allow multiple users to use the app on the same device	CH
Display a list of captured surveys	MH
Allow a Patient survey to be captured offline	MH
Allow a Mosquito Batch survey to be captured offline	MH
Allow a Mosquito survey to be captured offline and associated with a batch	MH
Add geolocation coordinates to a survey	MH
Take a photo of a mosquito	MH
Take multiple photos of a mosquito	CH
Transmit surveys to Server when device has internet connectivity	MH
Optionally restrict survey transmission to unmetered networks	CH
Allow a user to select their primary survey type	CH
Allow surveys to be modified on the device	CH
Allow survey data to be reviewed on the device	MH
Allow a DNA fragment to be attached to a survey	MH
Allow a sound sample of a flying mosquito to be recorded and attached to a survey	CH
Ensure all communication to and from Server is secured	MH
Survey data should be securely stored on the device	MH

3.3 PT app specifications

PT app is designed to be quick to learn and easy to use. In 2014, Google introduced Material Design¹⁶, a visual language for user interface (UI) design that ensures applications implementing this language in their user interface will be familiar to users of the Android platform. PT app follows this visual language wherever possible.

3.3.1 Authentication

Only authorized users are allowed to use the application. A first-time user of PT app will be requested to sign in using their email address and password. PT app should verify that a user is registered in the server application. If a registered user enters incorrect information they should be presented with an error message to that effect. If a non-registered user attempts to sign in they should be presented with a message explaining that only registered users can use the application.

3.3.2 User preferences

A user should be able to configure the app to allow the capture of either Patient or Mosquito (mosquito batch, mosquito detail) surveys, or both.

A typical user is often situated in a location where internet connectivity is poor. They may be conducting research away from their home country and therefore subject to cellular data roaming costs. A user should be able to choose whether surveys will be uploaded via cellular data or an unmetered network (WIFI, Bluetooth).

A user should be able to determine whether photos should be optimized when stored.

3.3.3 First-time use configuration

A user should be presented with a set of configuration questions the first time they use the app. This process should:

- Ask a user their primary and secondary survey types and store the selected settings.
- Request permission to access device location and camera if permission has not already been granted.
- Present a message encouraging users to encrypt their device if mandated by their organization's data privacy policies.

3.3.4 Survey capture

PT app should automatically assign a unique identifier to a survey. PT app should assign the date and time a survey was captured, and allow it to be modified by a user.

When a survey is completed the data should be securely stored on the device.

PT app should provide forms that allow a user to capture the three types of surveys listed below:

1. Mosquito Batch

ZP researchers collecting mosquitos for analysis tend to collect a batch or batches of mosquitos at various locations within their designated geographical research area. A mosquito batch should contain any number of mosquito detail surveys. PT app should provide a form to allow mosquito batch meta data to be stored. A user should be able to capture device geo-coordinates and include them with the survey.

2. Mosquito

Each mosquito in a batch is analysed at a lab and associated data is stored in a mosquito detail survey. PT app should provide a form allowing individual mosquito meta data to be stored. This survey should be associated with its respective mosquito batch survey. PT app should allow a user to take a photo of the mosquito and include it with the survey, it should also allow a sequenced DNA file to be attached to the survey. PT app could, if possible, allow a sound recording to be captured and attached to the survey.

3. Patient

ZP researchers collecting physical patient samples, such as: blood, serum, and urine, tag and collect meta-data to be associated with the sample collected. PT app should provide a survey allowing meta data to be captured. A user should be able to capture device geocoordinates and include them with the survey. As there are a large number of fields in this survey type, PT app could provide an easy way to navigate through the many fields. Patient surveys should allow a survey to be flagged for urgent review.

3.3.5 Survey review and modification

A user should be able to review surveys that have been captured on the device. PT app should provide a user interface listing all surveys on the device. The list should clearly distinguish between survey types. A user should be able to select a survey in the list and review it in detail.

The survey list should display an icon against each survey to indicate whether or not it has been uploaded, it should also display an icon against Patient surveys to indicate whether they are flagged or not.

If possible, PT app could provide a UI allowing a user to modify survey details after they have been captured.

3.3.6 Survey upload

PT app should automatically upload surveys to the server system. If a user has configured PT app to prevent transfer over metered networks, PT app should wait until the device has a WIFI connection before commencing upload. Once survey data has been uploaded, PT app should update the survey entry on the device with the date it was uploaded.

4 SCHEDULE

This chapter explores the overall project schedule, from gathering requirements through implementation, testing, and documentation.

The project was carried out in several phases over two stages. Stage one was carried out between June and October 2017, stage two between June and September 2018.

4.1 Stage one schedule

Stage one included the following phases:

19 June – 15 July 2017

- Establishing requirements and specifications, discussed in chapter 3, including:
 - Creating interactive prototypes and reviewing with ZP team.
 - Compiling list of requirements.
 - Triaging requirements between MH and NH categories
 - Compile PT app specifications.
 - Write BDD user stories, discussed in chapter 6
- Planning and architecture discussed in chapters 5 and 6, including:
 - Select and prepare development tools and programming language
 - Review and select component architecture for PT
 - Firebase service account creation and setup
 - High-level application component plan and organization

15 July – 5 October 2017

- Implementation, discussed in chapter 7, including:
 - Development of features related to capturing and storing surveys on a smartphone device.
 - Development of features related to reviewing surveys which are stored on the device.
 - Release beta version of the app with above features for user testing

Stage two included the following phases:

1 June – 17 September 2018

- Implementation continued:
 - Development of features related to authenticating users against the Firebase authentication service
 - Development of features related to upload of locally stored surveys to Firebase Cloud Firestore database
 - Releasing app with all features for user testing
- Documentation (this report)

5 TOOLS AND LIBRARIES

This chapter explores research and selection applied to the programming language, tools, and software libraries used to develop PT app. It also introduces some of the terms and concepts discussed in chapter 6: architecture.

5.1 Investigating and selecting an Android programming language

When this project was proposed, a new programming language, Kotlin¹⁷, had been created by JetBrains¹⁸. Kotlin was created as a multi-purpose language including support for developing Android applications. A brief investigation revealed that Kotlin included a number of language features that would be useful to a developer with little experience developing for Android. Of especial interest were reduced boilerplate and protection against `NullPointerException`s¹⁷.

The term boilerplate, as relates to software engineering, refers to a portion or portions of code that are repeated throughout the code with little to no alteration¹⁹. A common example of boilerplate are methods in a class that allow an object's properties to be read and/or modified.

A `NullPointerException`²⁰ is an error where an application's code attempts to access or write to some property or object that does not exist. Preventing an application from throwing these exceptions typically involves a lot of boilerplate code.

This phase of the project comprised the investigation as to whether Kotlin was suitable for PT app. The following questions and answers were posed:

- Is Kotlin mature enough and does it contain enough features to be used? As the language was fairly new and not officially released at the time of proposing this project, some investigation into whether it was suitable and stable enough for use was carried out.

Google announced official support for using Kotlin to develop Android apps²¹ in May 2017; as Google owns and develops the android operating system, this was deemed a satisfactory response to the question.

- Could enough time be allocated towards learning a complex, new language? Online research was carried out to review the language specification and determine whether it was well documented.

It was found that the online software engineering community had started using Kotlin prior to its official release and adoption. A fair number of introductory and detailed blog posts and tutorials had been published online^{22–26}. Additionally, complete documentation of the language and its features was available and maintained on the

kotlinlang.org website²³.

- Do third-party libraries exist that support Kotlin? Due to project time constraints it was assumed that open source third-party libraries may be used to provide functionality required by PT. Some research was undertaken to determine whether there were libraries available.

A number of libraries were found that could be useful. Kotlin is interoperable with Java²⁷ so any libraries written in Java would work if Kotlin was to be used as the application's primary language.

As a result of this evaluation, Kotlin was chosen as the main programming language for PT app, with the provision that, should it prove necessary, parts of the application could be developed in Java.

5.2 Tools and libraries

The following tools and third-party libraries were used to develop PT app.:

5.2.1 Development environment

Android Studio, an Integrated Development Environment (IDE) application designed for producing Android apps, is officially recommended by Google. As this project used Kotlin as the programming language, a beta version of Android Studio 3, specifically released for Kotlin development, was used to develop PT app.

5.2.2 Version control

Bitbucket cloud, an online git repository provider, was used as the version control repository for PT app code. As each feature was completed, all new and modified code was committed locally, commits were periodically pushed to this remote repository to maintain an online backup.

5.2.3 Third-Party libraries

Besides Google Play Services, Android support, and Firebase libraries, the following third-party open source libraries were used to provide functionality required when developing PT app.

Effortless Permissions

An Android application must declare the device hardware APIs it intends to access. Certain permissions are categorized as dangerous and require the user's explicit consent. Previously this consent was requested as an application was being installed on the device. As of Android 6.0 (API level 23), a new permission model was introduced. This model allowed users to revoke an app's permissions at any time, thus runtime permission checks are required.

As this process is quite involved, it was decided to use this library to reduce the amount of boilerplate code that would have to be written to check these permissions in every Activity class accessing hardware APIs.

Location Manager

Android provides APIs to acquire the geolocation coordinates of a device. These APIs involve a lot of repeatable code to deal with the large number of checks required each time a location is requested. These checks include:

- Determining if a device has location services enabled. Asking a user to enable location services if not.
- Determining if a user has revoked the app's permission to request a device's location, and requesting they grant permission in this case.
- Determining if an accurate 'last known location' (LKL) is available for the device. Android operating system can provide this directly if another application has been or is accessing the device's location. This speeds up the geolocation process.
- Requesting location updates if no LKL is available, for example, if the user has disabled location services, the LKL will not be known.
- Listening to location updates and then informing the system that updates are no longer needed. Subscribing to location updates means an app will consume a lot of battery power, so maintaining an ongoing subscription when not required is not optimal.

It was decided to use LocationManager²⁸ as it handles the above cases and meant minimal code would have to be duplicated in components requiring geolocation coordinates.

Realm Database, Kotlin extensions

Realm Database is a DBMS specifically designed for mobile applications and purported to speed up application development²⁹. The data schema can be described by sub-classing the RealmObject interface and using annotations against model properties to define primary and secondary keys. Realm provides a simple query language and full transaction support.

An additional feature of interest is its built-in support for an encrypted database, however due to export license considerations³⁰, this was not used.

This library was selected as it enabled rapid implementation of the PT app's local database, and provided flexibility should the models need to change as development proceeded.

Kotlin-Realm-Extensions³¹ is an additional library that was used. It takes advantage of Kotlin language features and provides a simpler interface for querying and writing to a Realm database, and auto-closing transactions.

Android Material Stepper

The Patient survey specification, described in chapter 4, contained a fair number of data fields. While gathering requirements with ZP it was agreed that this survey form could perhaps be divided over multiple screens. Android Material Stepper³² is a library that provides Interfaces and methods to navigate through multiple views in an Android activity. This library was used to facilitate the creation of the Patient survey components.

Anko

Anko is self-described as: *“a Kotlin library which makes Android application development faster and easier. It makes your code clean and easy to read, and lets you forget about rough edges of the Android SDK for Java”* ³³.

A number of Anko's features were used to develop PT app, these are described in chapter 7 when relevant.

6 ARCHITECTURE

This chapter presents the project's research into best practices and architectural patterns for organizing Android app components. It then presents the architecture and design of PT app.

6.1 A brief introduction to Android components

While delving into the details of the Android software development kit (SDK) and application programming interface (API) is beyond the scope of this report, some introduction to the main Android components that concern this project is useful to provide context while discussing app component architecture.

6.1.1 Activity

An Activity³⁴ is a component that enables a user to interact with the application. It presents a UI that contains design elements that form the view. Android applications have at least one Activity which serves as the entry point into the application. An extensible markup language (XML) layout file³⁵ is typically used to describe the UI components that compose an activity's view. This file is loaded by activity when it is created and used by Android to construct the UI.

An activity can manage different views if required. This is accomplished by using one or more Fragments³⁶. These fragments can be interchanged as the user interacts with the app and each fragment can use its own XML layout file.

6.1.2 Service

An Android service³⁷ is a component that allows an app to carry out background tasks that require no user interaction. These tasks typically perform long-running operations or communication with remote systems.

6.1.3 Intent

An Android Intent³⁸ is a message that can be published by apps. It allows unconnected components in Android to communicate tasks to each other and respond³⁹. Intents have a number of purposes, such as:

- Starting Activities; optionally passing data to the starting Activity.
- Using functionality in other apps such as the camera app. An Activity allowing a user to take a photo will start an Intent to take a photo. The camera app will allow the user to take a photo and then start an Intent to return with the data, in this case the photo.
- Starting services.

6.2 Architectural patterns

This section describes the research into current best practices and architectural patterns for organizing android app components that was performed during the design of PT app.

6.2.1 Model View Controller (MVC)

The MVC pattern³⁸ is often used in software development projects with user interfaces. It serves to separate an application's concerns into three interconnected parts:

Model

The model is responsible for data and logic management. It typically consists of some database management system (DBMS) and any interfaces or abstractions that facilitate storing and querying the data.

View

The view is responsible for displaying data to a user and allowing interaction. Implementation of the View layer depends on where the application will be used, but generally it involves presenting the UI and allowing a user to perform actions which are forwarded to the Controller.

Controller

The Controller sends data to the View and user commands to the Model.

In an Android app, an Activity acts as the Controller and an XML layout acts as the View. As such, the Activity essentially acts as the Controller and View and not much separation of these two concerns is achieved as the Controller and View layers are tightly coupled.

The Model is not specific to Android architecture and can be implemented with any interfaces and classes that are required.

6.2.2 Model View Presenter (MVP)

MVP⁴⁰ was introduced in the 1990s and extends on MVC. It aims to solve challenges MVC introduces by removing the tight coupling between View and Controller. The goal being to produce code that is easier to test.

Model

In MVP, the Model is unchanged from MVC

View

The View is a passive interface which simply displays the data and passes commands to the Presenter.

Presenter

The Presenter interacts with the Model and the View. It fetches data from the Model and formats it for use in the View. The presenter is generally system agnostic and should not require knowledge of system-specific APIs. This enables easier unit testing as the class is decoupled from the system and is not tied to any specific View.

In an Android app, an Activity with its layout file forms the View layer, with all functionality relating to interacting with the Model moved to the Presenter.

6.2.3 Model View ViewModel (MVVM)

MVVM⁴¹ is similar to MVP, differing slightly in that the Presenter in MVP often maintains a reference to the View, though this depends on implementation, whereas the ViewModel in MVVM does not. It is instead instantiated by the View and requires no knowledge thereof. This enables complete decoupling of this component and so allows for easier testing.

A key feature of MVVM is that it implements the observer pattern. The ViewModel presents observable methods and properties to which the View can subscribe. The ViewModel sends notifications to the View when there are changes in the model, which enables the view to automatically update the data being displayed.

Figure 3 depicts the components and communication involved in the MVVM pattern.

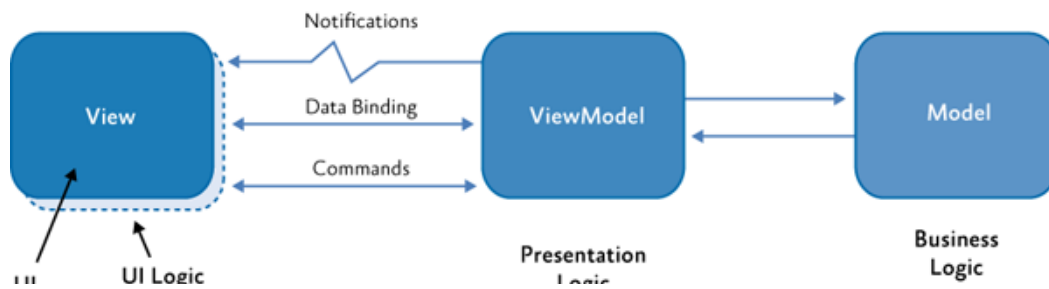


Figure 3 – MVVM architecture

6.2.4 Architecture choice

It was decided to try and use MVVM in the development of PT app. Due to time constraints this pattern was not strictly observed. The ViewModels were decoupled from any View, however automatic binding of data to the View was not implemented due to time constraints. Instead the ViewModel manages communication with the Model and exposes methods the View can use to request and submit data, which more closely resembles MVP.

6.2.5 Repository pattern

The repository pattern⁴² abstracts the data access layer of an application away from its implementation by isolating it via an interface. As a result, objects consuming and providing data are unaware of how the data layer works, they simply access methods described in the repository interface. This pattern allows multiple implementations of a common interface to be provided to a consumer and provides loose coupling between data persistence and consumption layers in an application.

PT is required to store surveys when the device is offline and to transmit data to a remote server when online, so both a local and remote database is required. PTs data layer implemented the repository pattern as it enabled consumers to not have explicit knowledge of the data's destination.

6.3 PT app structural design

UML is a language that provides a standardized way to visually describe the design of a system⁴³. When determining the approach to take with the architecture and design of PT app, UML modelling of user cases, use case diagrams, and sequence diagrams was considered.

As not much was known up front about the details of implementing Android applications, this project opted to design the system in terms of the behaviour of a user by using Behaviour Driven Development (BDD)⁴⁴, and user stories to describe expected behaviour and acceptance criteria. BDD and user stories are described in more detail in chapter 8.

A UML class diagram shows classes involved in a use case and relationships between them. The class diagrams (in 6.3.1) were created in order to provide a clear overview of classes and immediate relationships pertaining to the user journeys in PT app.

PT app was programmed to interfaces⁴⁵, however, in the diagrams presented the implementing classes are depicted for brevity. As Kotlin allows interfaces to implement methods⁴⁶, in some cases the interfaces are shown directly as they provide functionality not overridden in an implementing class.

The term 'starts a' is used in the diagrams in 6.3.1; an Android Activity does not construct another Activity class, instead it broadcasts the desired operation using a "Start Activity" Intent.

6.3.1 Authentication and first-time use

Android apps have one activity assigned as the “Main Activity” which is the entry point to the app. AppLauncher is PT app’s Main Activity.

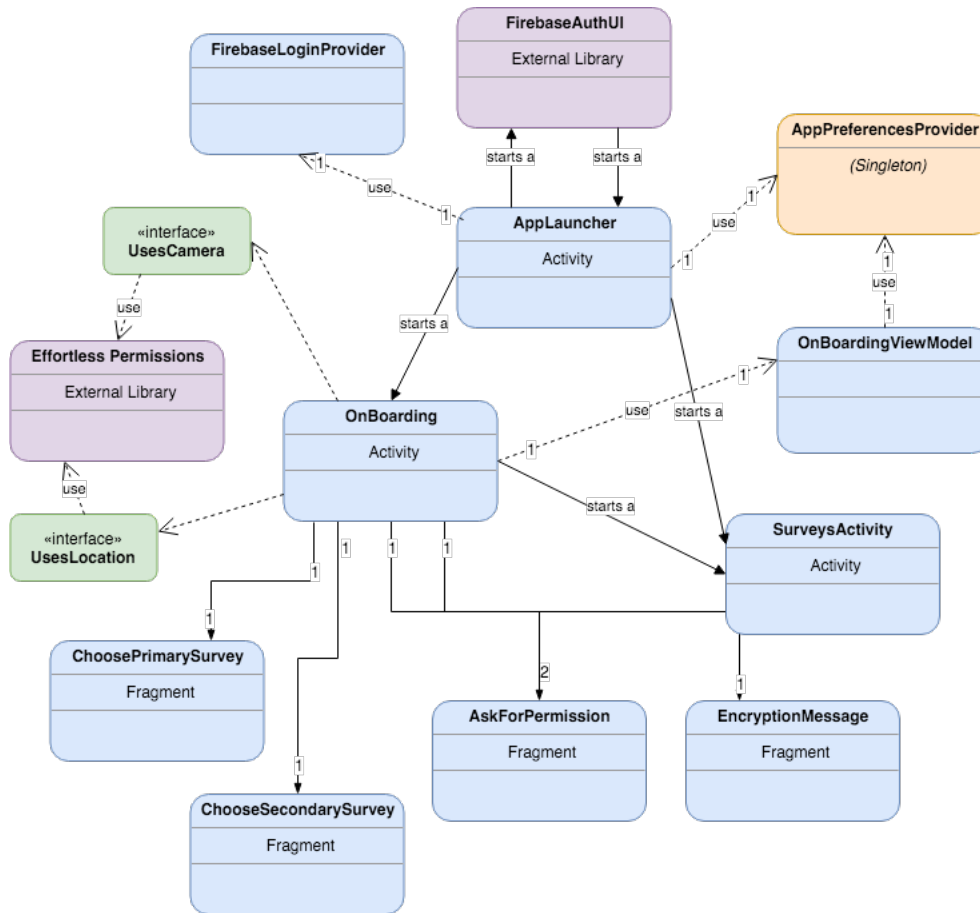


Figure 4 – Class diagram for “authenticate and set preferences” user journey

PT app starts up with AppLauncher object which uses FirebaseLoginProvider object to determine if a user has authenticated. If not, AppLauncher starts FirebaseAuthUI which manages the user authentication process. FirebaseAuthUI in turn starts AppLauncher and provides the authentication results.

If a user session exists AppLauncher uses AppPreferencesProvider object to determine if a user has completed the on-boarding process. If so, AppLauncher starts SurveysActivity object.

If not, it starts OnBoarding Activity. OnBoarding commences by instantiating OnBoardingViewModel object. OnBoardingViewModel stores the state of the OnBoarding journey using AppPreferencesProvider.

OnBoarding creates and presents different views to the user using the depicted Fragment objects. Initially it presents ChoosePrimarySurvey which, when a user chooses their primary survey, signals to OnBoarding the user's choice which is then passed to OnBoardingViewModel and saved by AppPreferencesProvider.

OnBoarding then presents ChooseSecondarySurvey which repeats the process.

OnBoarding then presents AskForPermission, specifying which permission is being requested, AskForPermission signals to OnBoarding which permission is being requested. OnBoarding uses EffortLessPermissions to request the permissions and then repeats the process for any further permissions required.

Finally, OnBoarding presents EncryptionMessage. The user selects "Done" which signals to OnBoarding that the process is complete which is then passed to OnBoardingViewModel and AppPreferencesProvider to store the state. OnBoarding then launches SurveysActivity.

6.3.2 Show a list of surveys

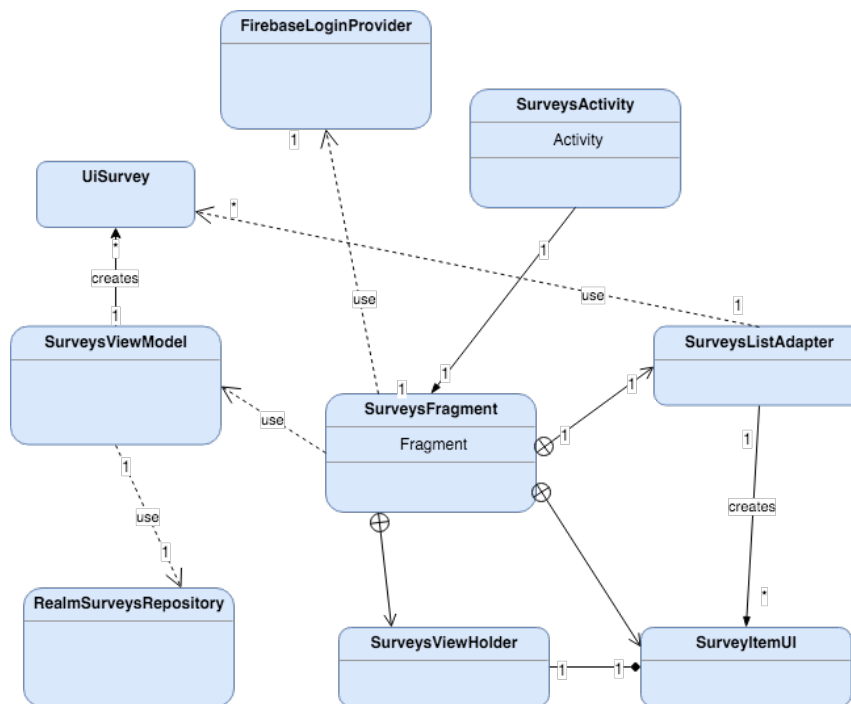


Figure 5 – Class diagram for “show a list of surveys” user journey

Once a user has signed in and completed on-boarding, when PT app starts, a user is presented with **SurveysActivity**. **SurveysActivity** creates a **SurveysFragment** object which manages the survey list view.

SurveysFragment instantiates a **SurveysViewModel** object, then creates **SurveysListAdapter** which requests **UISurvey** objects from the **SurveysViewModel**.

SurveysViewModel uses **RealmSurveysRepository** to request data from **RealmDatabase**. It then constructs **UISurvey** objects from the data and returns them to **SurveysListAdapter**.

SurveysListAdapter uses **UISurvey** objects to construct **SurveyItemUI** objects, which in turn constructs a view representing the **UISurvey** and constructs a **SurveysViewHolder** with the view. Finally **SurveysListAdapter** displays the **SurveysViewHolder** items in the **SurveysFragment** view.

6.3.3 Configure user preferences

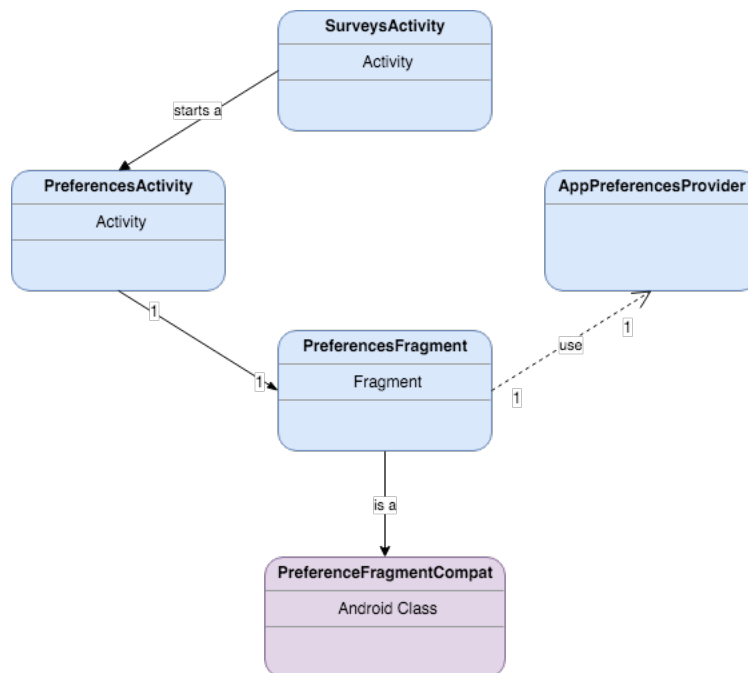


Figure 6 – Class diagram for "user can configure app" user journey

SurveysActivity, when a user selects the "Settings" menu option, starts PreferencesActivity. PreferencesActivity creates and presents PreferencesFragment. PreferencesFragment uses AppPreferencesProvider to perform some validation when a user selects the Primary/Secondary survey types. PreferencesFragment extends PreferenceFragmentCompat which is a type of Fragment provided by android that can read from and write to an app's preferences.

6.3.4 Add surveys

Add a patient survey

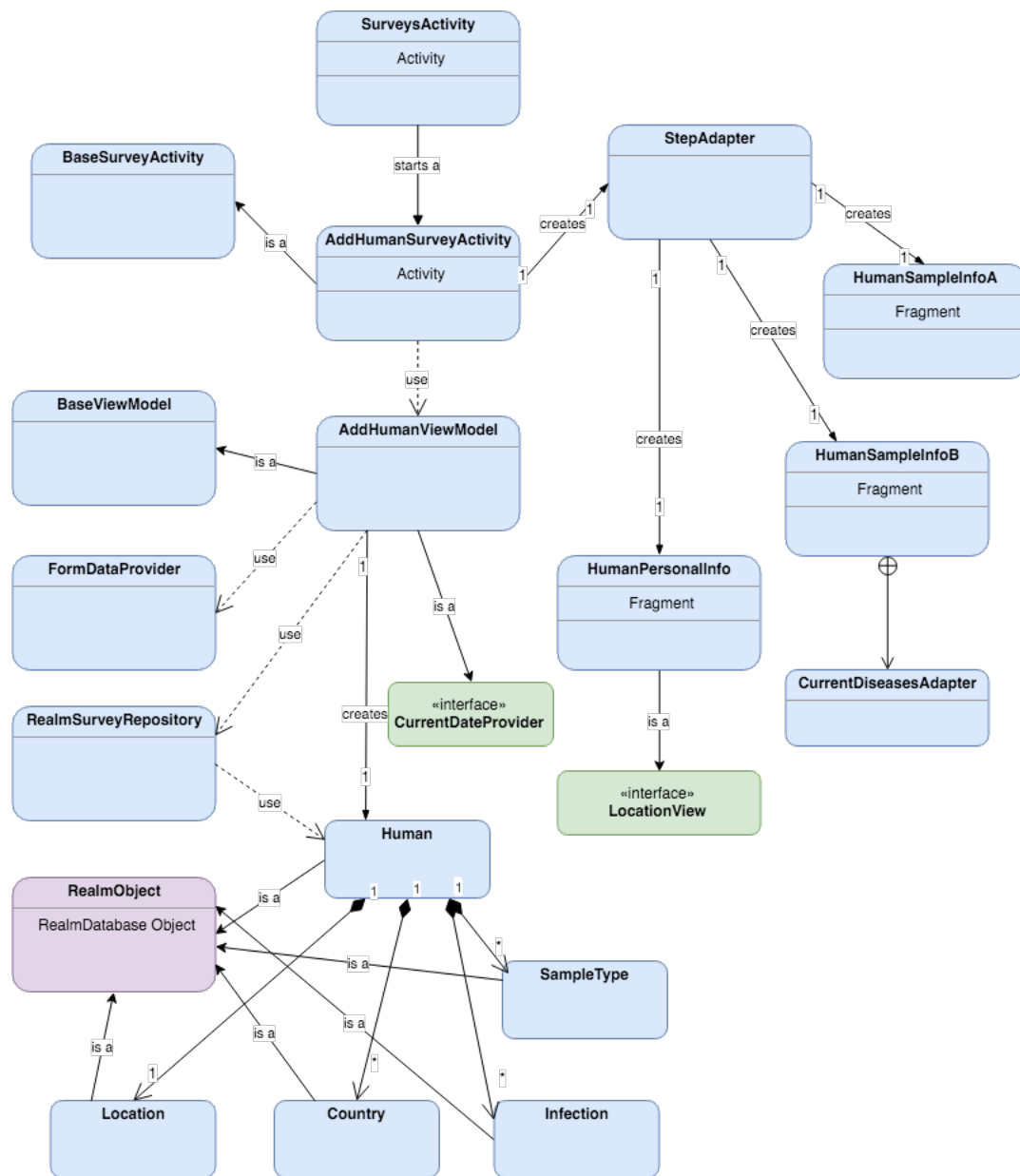


Figure 7 – Class diagram for "add a patient survey" user journey

From **SurveysActivity**, the user selects to add a patient survey and **SurveysActivity** starts **AddHumanSurveyActivity**.

AddHumanSurveyActivity, which extends abstract class **BaseSurveyActivity**, instantiates **AddHumanViewModel** object. **AddHumanSurveyActivity** then creates a **StepAdapter** object which creates and manages presenting **HumanPersonalInfo**, **HumanSampleInfoA**, and **HumanSampleInfoB** Fragments as the user selects to change the views in the form.

HumanPersonalInfo implements LocationView which manages fetching the geolocation coordinates using LocationManager external library. HumanSampleInfoB constructs a CurrentDiseasesAdapter which manages presenting the list of CurrentDisease values the user has entered.

AddHumanViewModel extends BaseViewModel and implements CurrentDateProvider which manages initializing the survey with the current date.

When a user selects to save the survey, AddHumanSurveyActivity requests the inputted data from the view fragments by way of StepAdapter. It then provides the data to AddHumanViewModel and uses it to construct Human a data object which it provides to RealmSurveysRepository to store in the database.

Human data object is composed of fields as well as SampleType, Country, and Infection objects, and a Location data object.

Add a mosquito batch survey

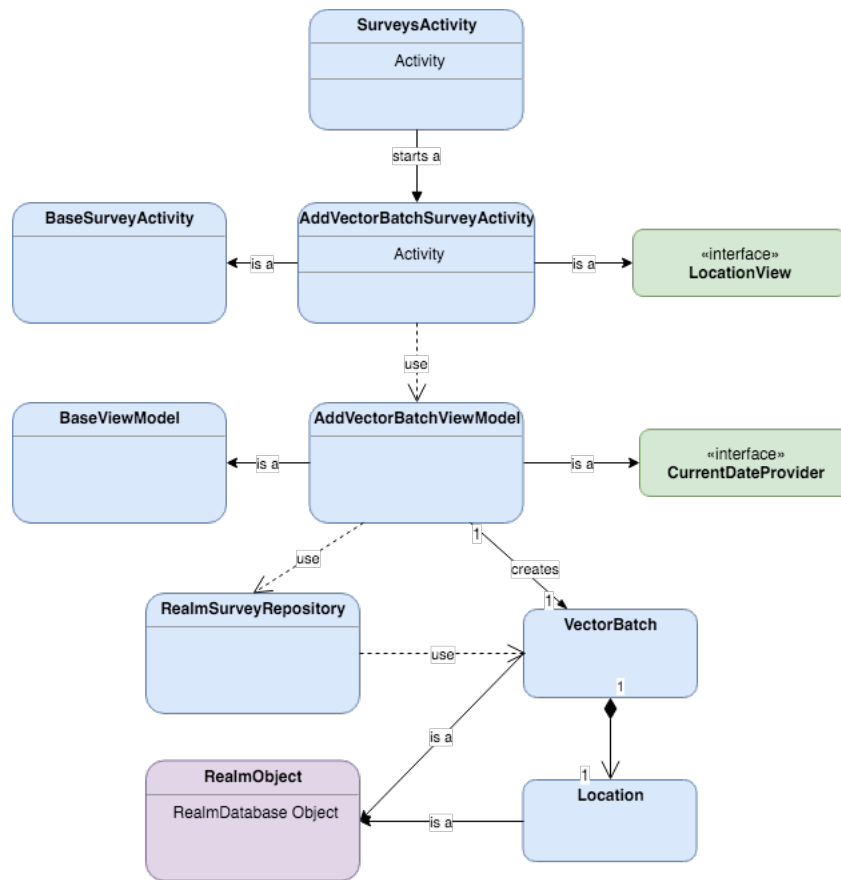


Figure 8 – Class diagram for "add a mosquito batch survey" user journey

From **SurveysActivity**, the user selects to add a mosquito batch survey and **SurveysActivity** starts **AddVectorBatchSurveyActivity**. **AddVectorBatchSurveyActivity**, which extends abstract class **BaseSurveyActivity**, instantiates **AddVectorBatchViewModel** object. **AddVectorBatchSurveyActivity** also implements **LocationView** which manages fetching the geolocation coordinates using **LocationManager** external library.

AddVectorBatchViewModel extends **BaseViewModel** and implements **CurrentDateProvider** which manages initializing the survey with the current date.

When the user selects to save the survey, **AddVectorBatchSurveyActivity** provides the data to **AddVectorBatchViewModel** which then constructs a **VectorBatch** data object which is then provided to **RealmSurveyRepository** and stored in the database.

Add a mosquito detail survey

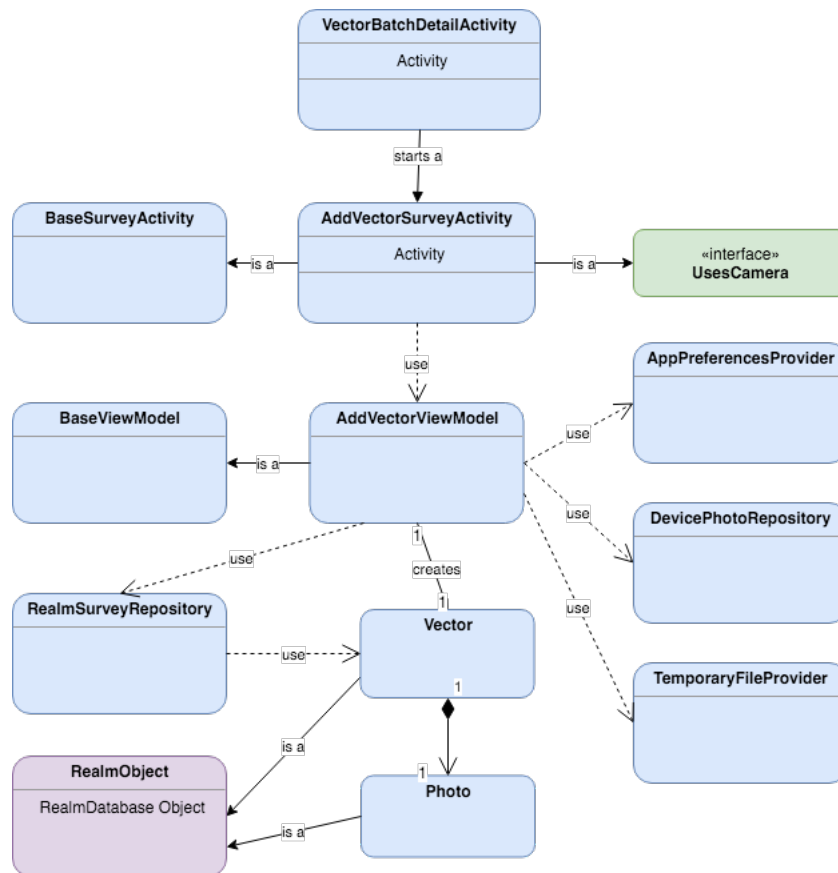


Figure 9 – Class diagram for "add a mosquito detail survey" user journey

From **VectorBatchDetailActivity**, the user selects to add a mosquito survey and **VectorBatchDetailActivity** starts **AddVectorSurveyActivity**. **AddVectorSurveyActivity**, which extends abstract class **BaseSurveyActivity**, instantiates **AddVectorViewModel** object which extends **BaseViewModel**. **AddVectorBatchSurveyActivity** also implements **UsesCamera** which manages device permissions related to capturing a photo, using external library **EffortlessPermissions**.

When the user wishes to take a photo, **AddVectorSurveyActivity** checks that the user has permission, asks **AddVectorViewModel** to create a temporary file. **TemporaryFileProvider** creates the file upon request and returns the reference to **AddVectorViewModel** which returns it to **AddVectorSurveyActivity**. **AddVectorSurveyActivity** then creates an Intent to signal to Android that it wishes to take a photo and have it stored in the temporary file, and Android launches the camera app. Once the user has taken a photo and chooses to keep it, camera app returns the photo and **AddVectorSurveyActivity** saves the bitmap and displays it in the view.

When the user selects to save the survey, `AddVectorSurveyActivity` provides the data to `AddVectorViewModel` which then constructs a `Vector` data object which is then provided to `RealmSurveyRepository` and stored in the database. The `Vector` data object contains a `Photo` data object with the file name and path of the stored file. `AddVectorViewModel` provides the `Photo` data object as well as the bitmap to `DevicePhotoRepository` which stores the photo in the device's file system at the specified path.

6.3.5 Review surveys

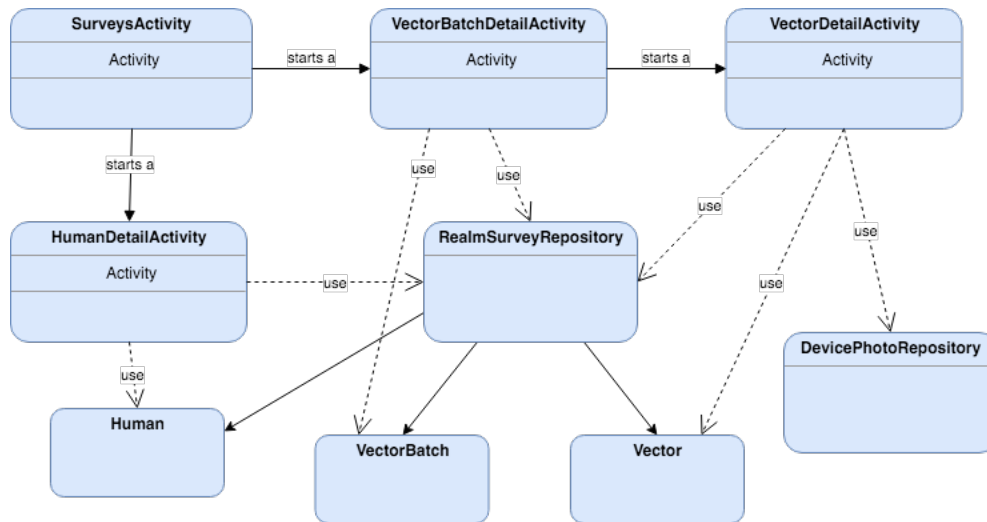


Figure 10 – Class diagram for "review surveys" user journey

Patient survey review

From SurveysActivity a user selects a Patient survey to review, SurveysActivity then launches HumanDetailActivity. HumanDetailActivity uses RealmSurveyRepository to fetch the Human data object from the Realm Database and presents it in the view.

Mosquito batch survey review

From SurveysActivity a user selects a VectorBatch survey to review, SurveysActivity then launches VectorBatchDetailActivity. VectorBatchDetailActivity uses RealmSurveyRepository to fetch the VectorBatch data object from the Realm Database and presents it in the view.

Mosquito survey review

From VectorBatchDetailActivity a user selects a Vector survey to review, VectorBatchDetailActivity then launches VectorDetailActivity. VectorDetailActivity uses RealmSurveyRepository to fetch the Vector data object from the Realm Database and presents it in the view, and uses DevicePhotoRepository to fetch the photo, if one was taken in the survey, from the device's file system and renders the bitmap in the view.

6.3.6 Background survey upload

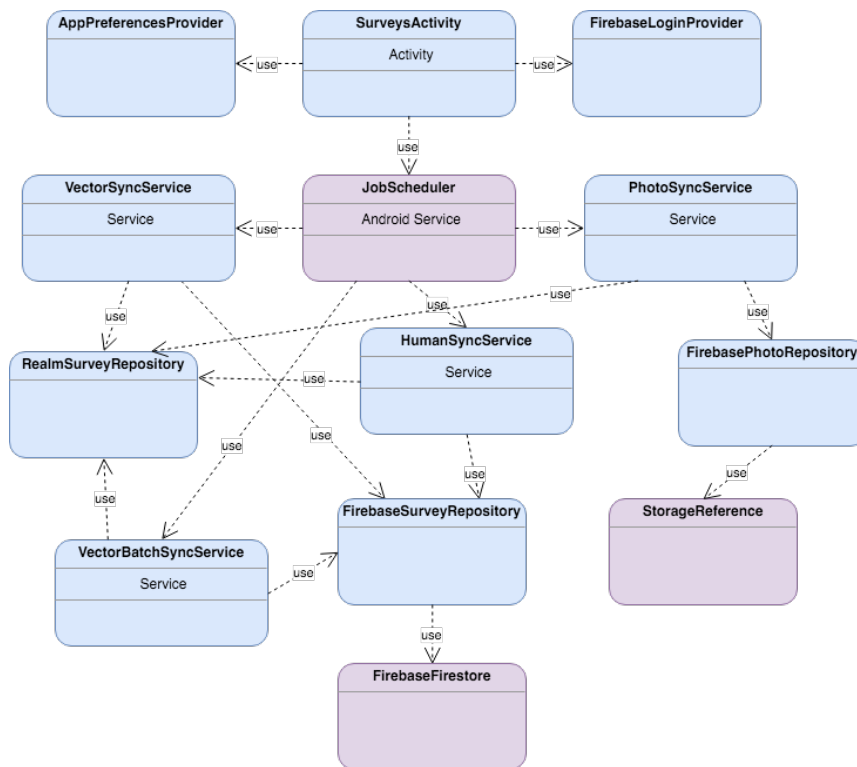


Figure 11 – Class diagram for "automatically upload surveys"

SurveysActivity uses FirebaseLoginProvider to check if a user has logged in. If they have, it uses JobScheduler to create jobs. JobScheduler creates VectorSyncService, VectorBatchSyncService, HumanSyncService, and PhotoSyncService objects as the jobs.

VectorSyncService, VectorBatchSyncService, and HumanSyncService use RealmSurveyRepository to fetch their respective survey data which they then send to FirebaseSurveyRepository which in turn uploads the survey data using FirebaseFirestore external library.

PhotoSyncService uses RealmSurveyRepository to fetch Photo data objects which it passes on to FirebasePhotoRepository. FirebasePhotoRepository uses the path to the photo to stream the photo to Firebase Storage using Firebase StorageReference external library.

6.4 PT app database layer

This section describes the database systems and models used with PT app. It presents the client-side data layer using Realm Database, followed by the server-side layer provided by Firebase Cloud Firestore.

6.4.1 Client-side: Realm Database

As discussed in chapter 5, when storing survey data locally on an Android device, PT app uses Realm Database DBMS. Realm uses a “live object” data model, storing data objects that are created by the application directly in the database.

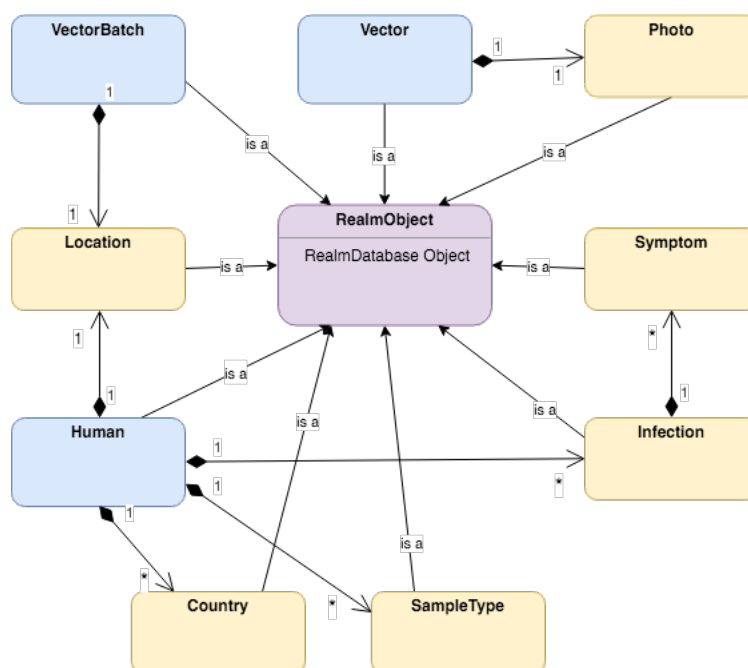


Figure 12 – PT app data objects

Here we see that all data objects extend RealmObject, this is required in order to store them in the database. Realm data objects, like any object in Kotlin or Java can be composed of other objects. For clarity, the ‘sub objects’ are shown in yellow.

The main data objects are:

- Human, representing patient survey data
- VectorBatch, representing mosquito batch survey data
- Vector, representing mosquito detail survey data.

Human objects contain one Location object and any number of Country, SampleType, and Infection objects. Infection objects contain any number of Symptom objects. VectorBatch objects contain one Location object and Vector objects contain one photo object.

6.4.2 Server-side: Firebase Cloud Firestore

Cloud Firestore is a NoSQL document database⁴⁷ (DD) system. DD systems pair keys, identifying the data, with a key-value structure known as a document. Depending on the implementation, a document can contain other documents. Some DD systems such as Cloud Firestore, support collections of documents, and documents can themselves contain collections of documents. This forms a hierarchical system supporting numerous scenarios.

The Cloud Firestore library for Android manages serialization of data objects being provided. Prior to transmission, PT app adapts Realm data objects into a simpler structure, these objects are then serialized by Cloud Firestore and stored as documents in the cloud.

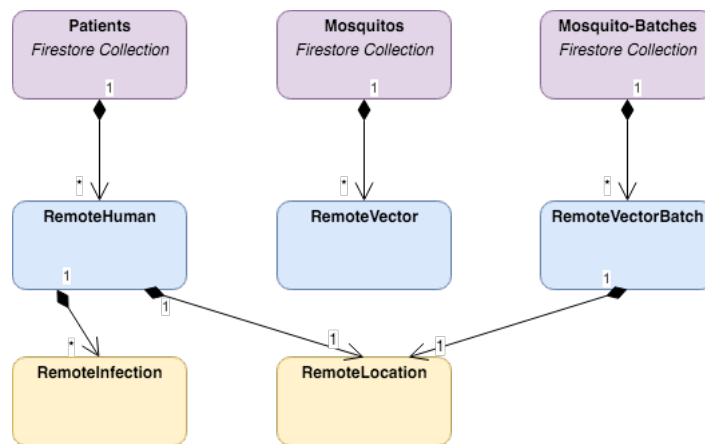


Figure 13 – Cloud Firestore data model

Here we see that Patient, Mosquito Batch, and Mosquito surveys are stored as documents in their respective collections.

Infection objects and Location objects are serialized by Firestore to key value maps which are stored in Patient and Mosquito Batch documents.

Photo objects on the client side are used to store information about the photograph, such as the filename and the path on the device. RemoteVector objects store the photo id string to identify the remote photo once uploaded, so the Photo data object is not required.

Country, SampleType, Symptoms are simple data objects containing a String. At the time PT app was programmed Realm required all lists in a RealmObject to contain Realm objects. On the server, a simple list of the string values suffices, hence, they are not represented in this model.

7 IMPLEMENTATION

This chapter discusses the implementation of PT app. It presents the development process followed by the main points of interest in PT app.

7.1 Development process

The development of PT app was separated into phases, each comprising a group of features. This enabled volunteer user testing to be carried out at two stages throughout the development. Additionally, this meant that should there be insufficient time to meet all requirements, some functionality would be complete and demonstrable.

The development process at each phase involved:

- Feature planning, including investigating feature dependencies and determining if all features would be programmed completely or if suitable open source third-party library could be used.
- Designing the interface.
- Writing tests against the feature specification if possible.
- Developing the feature so all requirements are met, and any unit tests pass.
- Investigating whether further tests are needed for edge cases and writing these when required.

A number of tests use the Espresso framework which allows a test to simulate user interaction.

Espresso tests were written to test some of the user journeys in the application and required the completion of some initial development before sufficient information to write the test became available.

The testing process is described in further detail in Chapter 8.

7.2 Android SDK

PT app targets Android API level 27 (version 8.1.0), however it was developed to support all versions between API level 23 (version 6.0) and 27. These versions account for roughly 68% of android devices in use⁴⁸ and was chosen as a good balance between being able to use new platform features while still supporting a large number of devices.

Besides the third-party libraries discussed in chapter 5, PT app uses some Android support, Google Play Services, and Firebase libraries. These are mentioned below where relevant.

7.3 Dependency Injection and App class

Dependency injection (DI) is a common term used to describe the “inversion of control” software development pattern which enables loose-coupling of components in an application⁴⁹.

There exist a number of DI libraries for Android, Dagger⁵⁰ being one example. However, there was not sufficient time available to learn how to implement a DI library.

The main class of an Android app is the Application class. This class initializes the application’s context and launches the designated Main activity. Some of the included libraries in PT app require a singleton instance to be used throughout the app.

It was decided to extend the Application class and provide methods to access these singleton instances. As there is only ever one instance of the Application class in an app, this seemed like a reasonable solution.

Where possible, constructor injection has been used to inject dependencies provided by the App class, however in some cases this is not possible, for example in Activity and Service classes they are constructed by Intents or Jobs and as such constructor arguments are not available.

7.4 Interfaces and abstractions

A consideration applied to the development of PT app was to achieve a separation between the main components of the app and any third-party components. An example of this is FirebaseAuth. Rather than directly accessing the FirebaseAuth instance in activity classes, an interface “LoginProvider” is used. It contains methods necessary to check if a user has authenticated, retrieve the user object when necessary, and remove a user session, allowing them to sign out of the app. An implementation class was created, “FirebaseLoginProvider” which implemented the interface using the FirebaseAuth library. As a result, should a different authentication provider be desired if PT app were to be used by ZP team, the only requirement would be to program a new LoginProvider implementation and swap out the reference in the App class.

The same was applied to the repositories: SurveyRepository interface is implemented by RealmSurveyRepository and FirebaseSurveyRepository, and PhotoRepository is implemented by DevicePhotoRepository and FirebasePhotoRepository. The result being that both Realm database and Firebase Cloud Firestore can be replaced by implementing SurveyRepository, and Firebase Storage can be replaced by implementing PhotoRepository.

7.5 Resources: strings, layout and graphical icons

The Android development fundamentals guide⁵¹ was followed as much as possible in the development of PT app. Android OS allows multiple apps to run simultaneously, so special care was taken to observe the Activity lifecycle methods to prevent memory leaks and other exceptions from occurring if PT app is running in the background.

Activity and Fragment layout resources were defined in separate XML layout files as per Android layout guidelines⁵² to allow clear separation between the UI components and code defined in the classes. In addition, all strings used in the app were extracted to a strings resource XML file which permits separation of localization, allowing PT app to be presented in a different language should this prove useful.

All icons used within the app UI are stored in scalable vector graphics (SVG) format. This not only reduces the size of the application package, it allows the UI to be drawn on different size screens without loss of image quality.

7.6 UI design

One of ZP specifications is for PT app to be easy to use. Consequently, PT app attempted to follow Google's Material Design language where possible. This involved using a number of android support libraries provided for this purpose. This was somewhat challenging; specifically, where locating documentation for some of the desired functionality was concerned.

PT app uses two main colour palettes to clearly distinguish between Patient and Mosquito surveys. Colour palettes were defined in a styles values XML file as per Android recommendations⁵³.

A number of icons were used in the application, these were used both to improve the user experience and clearly represent actions that a user could take. The icons used were downloaded from the Material Design Icons web library⁵⁴, available for use under the Apache license⁵⁵.

7.7 Kotlin language features

The Kotlin programming language contains a number of features, not present in Java, that were useful in the development of PT app.

7.7.1 Sealed classes: algebraic data types

An algebraic data type (ADT) is a type represented by other types. It is a mathematical model for type representation⁵¹. For example, a type “TrafficSignal” as an ADT would have representation types “Red”, “Yellow”, and “Amber”.

In Kotlin, an ADT is implemented by using a sealed class⁵⁶. A sealed class can have sub classes, but cannot be extended outside its class file. This means that instead of using if/else expressions, a when expression can be used (“when” is Kotlin’s equivalent to a switch expression). In the case of a sealed class, no “default” or “else” statement is required as the sealed class can only ever be represented by one of the sub types. This allows for concise code, and reduces required edge case testing as the outcome of a when expression on a sealed class can be mathematically proven.

7.7.2 Extensions

Kotlin enables classes to be extended without requiring a new inheriting class to be created⁵⁷. This feature was useful as it enabled adding utility functions to some Android components, without requiring a full class to extend the component and all the boilerplate programming that would entail.

7.7.3 Data classes

When dealing with the data layer of an application, a model class is often created to represent an entity in the database. The class typically contains only properties, and the associated getter and setter methods. Kotlin provides a type of class specifically for this use case, a data class. Data classes remove a lot of the boilerplate involved in this type of class, automatically generating the get/set methods as well as other typical methods such as copy, equals etc.

This was a useful feature of the language and removed a lot of boilerplate from the code base. It was not used however for the Realm data objects however, as Realm did not yet support data classes in the version used by PT app.

7.8 Authentication

One of PT app's requirements, is that only authorized users can record and access surveys.

The Firebase Authentication service serves as the authentication backend of PT app. Firebase UI is a library, created by the Firebase team, containing a number of UI components that enable easy integration with Firebase backend services.

Rather than creating the sign in/out UI components, PT app made use of the Firebase UI for Auth⁵⁸ (AuthUI) component in order to quickly integrate the authentication process. Justifications for this decision were: time constraints on the project, and that a research team may choose to use PT app with an alternative server system and would then require a change to the authentication integration.

7.9 Onboarding

AppLauncher, is PT app's main activity. It serves to direct the user to the appropriate activity depending on whether or not they have authenticated and completed the onboarding process. AppLauncher uses LoginProvider to determine if a user has authenticated, and AppPreferencesProvider to determine if a user has completed onboarding.

7.9.1 Authentication

If a user has not authenticated, or has signed out, AppLauncher launches the authentication process. AuthUI handles the complete authentication flow and then returns the user to AppLauncher.

7.9.2 First-time configuration

AppLauncher then checks if a user has completed the onboarding process, if not, it presents OnBoarding activity.

OnBoarding uses OnBoardingViewModel to manage state, and four Fragment classes to present the views.

This allowed for separation of logic in the views.

The first view presented is ChoosePrimarySurvey Fragment as depicted in figure 14. The user has two buttons they can tap, each button represents one of the two allowed survey types.

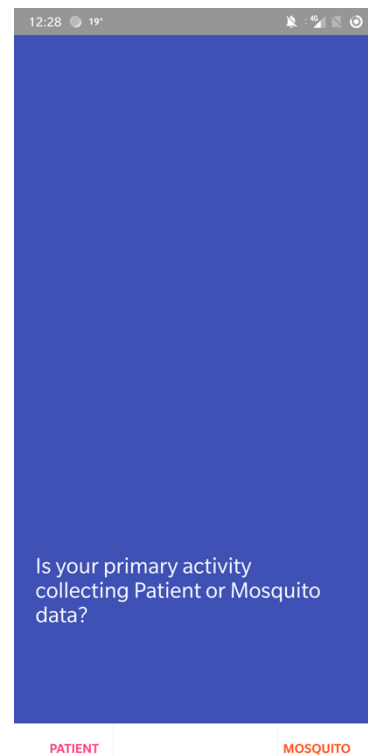


Figure 14 – Choose primary ST

The sealed class “SurveyType” (ST) is used to represent the three options available to a user. A user is either taking VECTOR or PATIENT surveys. The remaining option, NONE applies to the “user only takes one type of survey” scenario.

The next view, shown in figure 15 presents the user with the choice to be able to take both types of surveys. In this case, the buttons represent ST states VECTOR and NONE.

The remainder of the onboarding process requests that the user grants the device permissions needed by PT app, and finishes with a message to the user asking them to encrypt their device if their organization’s security policies so mandate. OnBoarding activity uses AskForPermission fragment for permission requests, injecting the permission requested when instantiating the fragment. The final fragment, EncryptionMessage displays the message and provides a “Done” button. When the user selects this button, OnBoarding uses AppPreferencesProvider to store the user preferences in PT app’s shared preferences.

Shared Preferences is an Android object containing key-value pairs and simple methods to read and write to them. It is useful for storing simple values where a database is not required. SharedPreferences can either be private or shared with other apps on the device. PT app uses private shared preferences.

7.10 Surveys

When PT app starts, AppLauncher, if a user has authenticated and completed onboarding launches Surveys activity.

SurveyViewModel provides data to Surveys activity when requested. It uses RealmSurveyRepository to retrieve patient and mosquito batch surveys from the database. As shown in in figure 16, if there are no surveys, a welcome message is displayed, using LoginProvider to retrieve the user’s name from FirebaseAuth. The message directs their attention to the add survey button.



Figure 15 – Choose secondary ST

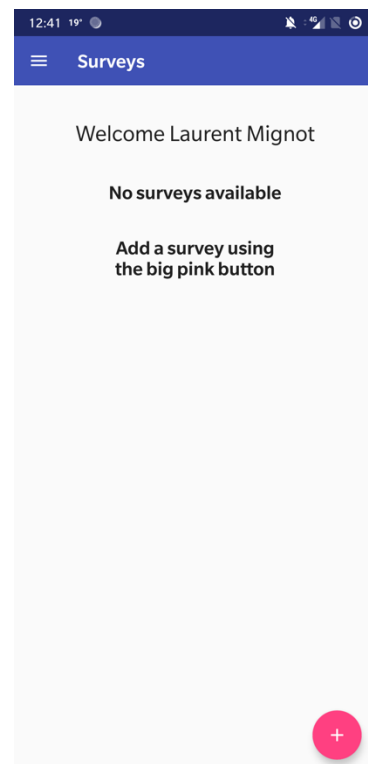


Figure 16 – No surveys recorded

When a user taps the button, Surveys activity uses `AppPreferencesProvider` to determine whether the user has configured the app for Patient or Mosquito surveys or both.

If the user has selected only one, the appropriate activity is launched. If the user has selected both, it displays a dialog with two buttons as per figure 17. When either of the two buttons is tapped, Surveys activity launches the appropriate activity.

Surveys also provides a menu button. This button opens a “Drawer menu”, depicted in figure 18, which contains two buttons, “Settings” and “Sign out”. When sign out is tapped, Surveys uses `LoginProvider` to direct `FirebaseAuth` to destroy the user session and then launches `AppLauncher` which then initiates the authentication process. The settings button launches `AppPreferencesActivity`, discussed in 7.18.

7.10.1 Survey item list

`SurveysFragment` is used to manage the survey items list. If a user has already taken surveys, `SurveysViewModel` maps each `Human` and `VectorBatch` survey object to a `UiSurvey` data object and provides these to `SurveysFragment`. `UiSurvey` contains properties common to both survey types and allows them to be displayed in the same list.

`SurveysFragment` uses an inner class, `SurveysAdapter`, that implements an adapter for an Android view component called “`RecyclerView`”. A `RecyclerView` component handles displaying a large number of items in a list by creating/destroying view items as the list is navigated⁵⁹.

To display the individual items in the list, and allow them to be interacted with, an inner class, `SurveyItemUi`, manages creating a view for each survey item.

`SurveyItemUi` is an example of PT app’s use of the Anko library, previously introduced in chapter 5. Anko provides a

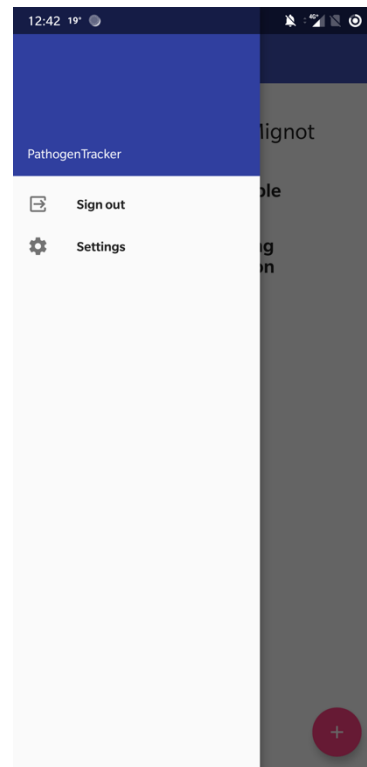


Figure 17 – Drawer menu open

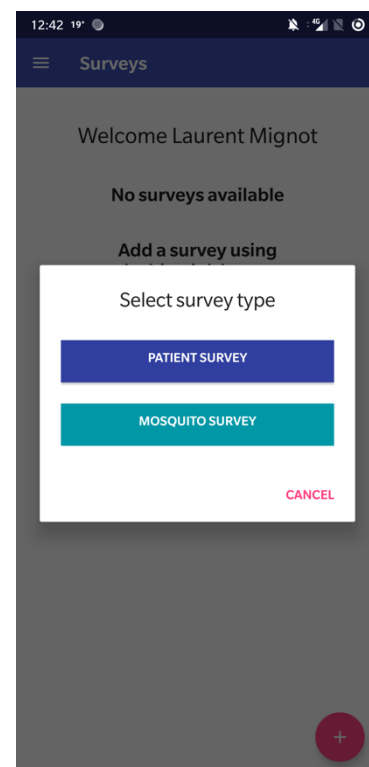


Figure 18 – Choose survey type

domain specific language (DSL) for Android views. A DSL is some programming language that specialises in a particular application domain⁶⁰. In this case, Anko Layouts DSL specialises in a declarative syntax for describing an Android view component.

Figure 19 shows Surveys activity when a user has captured some surveys. Patient surveys are represented by a “H” icon and Mosquito surveys by a teal “M” icon.

Patient surveys that have been flagged display the flag in the list. The cloud icon serves to indicate that the survey has been transmitted to the remote server.

When a user taps on a survey in the list, the appropriate survey detail activity is started; the id of the survey is provided with the Intent.

7.10.2 Survey transmission

The final responsibility of Surveys activity, is to start the background service jobs that handle uploading of surveys and photos to the server.

First, LoginProvider is used to verify that a user is signed in. Then AppPreferencesProvider is used to determine whether a user has permitted upload via metered network (cellular data).

JobScheduler

JobScheduler is an Android system service that is used to schedule an app's background tasks. It accepts a JobInfo object which contains the class name of the service being scheduled, as well as any job settings. A number of settings are available, of interest to PT app are the network type, and persistence.

Network type options are NETWORK_TYPE_NONE, NETWORK_TYPE_UNMETERED, and NETWORK_TYPE_NOT_ROAMING. PT app uses the first if a user has allowed cellular upload, and the second if not.

The persistence setting allows jobs to repeat even if PT app is no longer active.

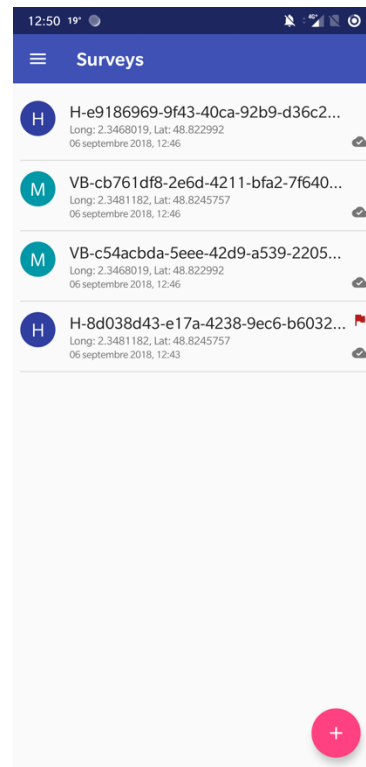


Figure 19 – Surveys activity displaying surveys

PT app services

PT app starts four services, one for each type of survey and one for photos. The survey services each query RealmSurveysRepository for any surveys of their type that have not been uploaded. This is determined by checking the “uploadedAt” property in the models, it either contains a timestamp or is null. If null, the survey has not been transmitted. The service then uses FirebaseSurveysRepository to transmit the surveys. When a survey has been successfully transmitted, the uploadedAt property is given the current timestamp.

The photo service again queries RealmSurveysRepository for Photo objects. The Photo objects represent a photo attached to a Vector survey. They contain the id of the vector survey as well as the file name of the photo and the path to the photo file itself. It then uses FirebasePhotoRepository to transmit the photos, and again sets the uploadedAt property when complete. If an upload fails for any service, the uploadedAt property is not set and the survey remains queued for transmission.

7.11 BaseSurveyActivity

All survey form Activity classes extend BaseSurveyActivity which abstracts repeated functionality such as what happens when a user taps on the toolbar buttons.

The save button initiates saving the survey to the database.

If a user taps the close button (X) or the Android device’s hardware back button, a dialog is displayed to warn the user that the survey has not been saved.

7.12 Recording a Patient survey

When a user chooses to record a Patient survey, AddHumanSurveyActivity is launched.

As this survey has a fair few questions, a third-party library “MaterialStepper” is used to allow the user to easily navigate between the three views forming the survey.

MaterialStepper manages loading fragments as the user swipes back and forth or as they tap the “Next” and “Back” buttons displayed when appropriate.

AddHumanViewModel extends BaseViewModel and CurrentDataProvider interface. BaseViewModel provides some abstraction for methods repeated between survey activities. CurrentDataProvider contains methods and manages automatically setting the present date in the survey.

A survey id is automatically generated by all survey view models. The id cannot be change by the user.

7.12.1 LocationView

The first fragment, HumanPersonalInfo, depicted in figure 20, implements LocationView interface, which provides methods related to retrieving geolocation coordinates from Android system.

LocationView uses a third-party library, LocationManager, which handles all the conditions that could occur when requesting a location.

7.12.2 MultiSpinner

The second fragment, HumanSampleInfoA, shown in figure 22, makes use of a custom view component, MultiSpinner, programmed for PT app. A spinner menu is a menu displaying a list of values from which any one can be selected. PT app requirements are that a user should be able to select a number of values out of a discrete set. This component allowed a user to tap an “Add another” button

12:43 19°

× New Patient SAVE

Patient ID
H-8d038d43-e17a-4238-9ec6-b6032074e53d

Survey date
06 septembre 2018

Location Collected
Tap to update location

Date of Birth

Gender
☐ Female ☐ Male ☐ Other

Is the subject pregnant?
☐ Yes ☐ No

... NEXT >

Figure 20 – Patient survey, view 1

12:45 19°

× New Patient SAVE

Sample type(s)
Blood
ADD ANOTHER

Recent travel history
Liberia
× Martinique
× French Polynesia
ADD ANOTHER

Past infection(s)
Dengue
× Zika
ADD ANOTHER

< BACK ... NEXT >

Figure 21 – Patient survey, view 2

which creates a new spinner object from which a value can be selected. The X button allows the user to remove a value.

MultiSpinner makes use of Anko Layouts DSL to construct the view components required.

This required a fair amount of work to accomplish as the author was new to Android and not familiar with the details of creating Android components. The result, however, performs as desired and provides the required functionality.

7.12.3 Add a current disease entry

The final fragment, HumanSampleInfoB, allows the user to enter a patient's current disease history, including the symptoms for each disease. As a patient could possibly be infected with more than one disease, the requirement called for some imaginative implementation.

An extension of MultiSpinner was considered and dismissed due to time constraints. Instead, the user is presented with a dialog, shown in figure 22, which provides a spinner for the disease and a set of checkboxes for the symptoms. PT app is designed to collect surveys on mosquito borne pathogens and the disease and symptom values reflect this.

When a user has completed the survey, they tap the save button, present on every view, and AddHumanViewModel uses RealmSurveyRepository to store the Human data object in the database.

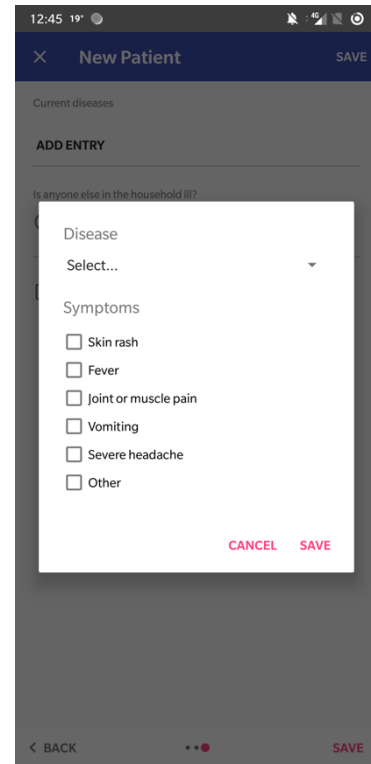


Figure 22 – Patient survey, view 3

7.13 Recording a Mosquito Batch survey

VectorBatchSurveyActivity is launched when a user chooses to create a Mosquito Batch survey. Figure 23 shows the view presented to the user. Color differentiation has been used to distinguish between SurveyTypes to make it clear to a user the type of survey they are recording.

As with adding a patient survey, the ViewModel class, in this case VectorBatchViewModel, provides the unique ID and sets the survey date. LocationView is used to manage retrieving the location.

When the save button is pressed a VectorBatch object is created with the survey data stored in the repository.

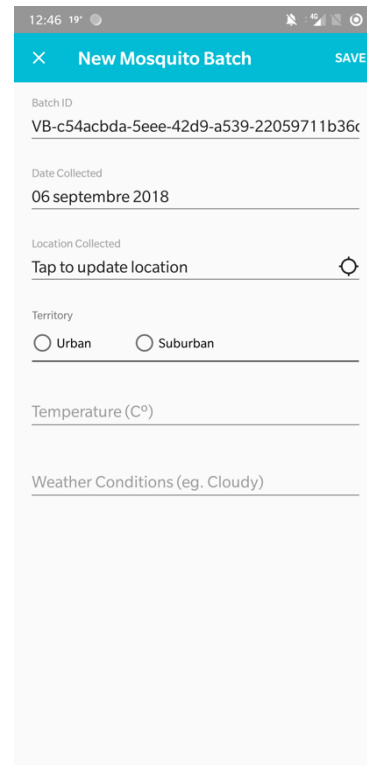
A screenshot of a mobile application interface for recording a Mosquito Batch survey. The screen has a light blue header bar with a back arrow, the title 'New Mosquito Batch', and a 'SAVE' button. Below the header, the form contains several input fields: 'Batch ID' with a long alphanumeric string, 'Date Collected' with the date '06 septembre 2018', 'Location Collected' with a text field 'Tap to update location' and a location icon, 'Territory' with two radio buttons labeled 'Urban' and 'Suburban', 'Temperature (C°)' with a text field, and 'Weather Conditions (eg. Cloudy)' with a text field. The status bar at the top shows the time '12:46' and battery level '19%'.

Figure 23 – Mosquito Batch survey

7.14 Recording a Mosquito detail survey

When a user wishes to record a Mosquito detail survey, they first open the Mosquito Batch survey to which the mosquito belongs from Surveys list, and from there tap the button shown in figure 26 which starts AddVectorSurveyActivity.

The activity, aside from the usual fields, allows a user to take a photo. They tap the Take Photo button which starts the process.

The activity requests a temporary file from AddVectorViewModel, which uses TemporaryFileProvider to create a temporary file in the file system directory belonging to PT app. This path is stored in the view model temporarily. A camera Intent is constructed with the temporary file as an argument. Then the camera app activity is started. When the user has taken the photo and selected “ok” camera app starts PT app again, providing the photo bitmap in the Intent.

AddVectorViewModel saves the bitmap temporarily.

If a user taps the “replace photo” button, the temporary file and bitmap are discarded and the process is repeated.

When the user taps the save button, a Photo data object is constructed containing the path, the filename, and the vector survey id. Next, a Vector data object is constructed containing the Photo object. DevicePhotoRepository is used to store the bitmap in the app’s directory for images, and RealmSurveyRepository saves the Vector object.

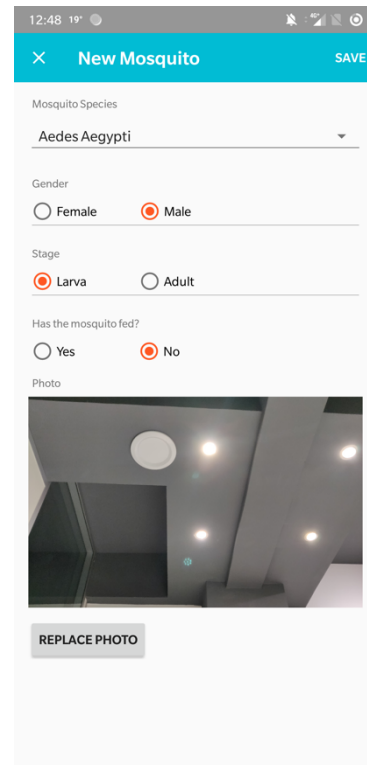
The image is a screenshot of a mobile application interface for recording a mosquito survey. At the top, there is a teal header bar with a close button (X), the title 'New Mosquito', and a 'SAVE' button. Below the header, the form contains several sections: 'Mosquito Species' with a dropdown menu showing 'Aedes Aegypti'; 'Gender' with radio buttons for 'Female' and 'Male' (where 'Male' is selected); 'Stage' with radio buttons for 'Larva' and 'Adult' (where 'Larva' is selected); and 'Has the mosquito fed?' with radio buttons for 'Yes' and 'No' (where 'No' is selected). At the bottom of the form is a 'Photo' section featuring a placeholder image of a modern interior with a circular light fixture and a button labeled 'REPLACE PHOTO'. The status bar at the very top shows the time as 12:48 and the battery level at 19%.

Figure 24 – Mosquito detail survey

7.15 Reviewing surveys

The process for reviewing surveys is similar for Patient and Mosquito batch surveys. From Surveys list, a user taps on the survey they wish to review, and the appropriate activity is started with the survey id provided in the Intent: respectively, HumanDetailActivity and VectorBatchDetailActivity. For Mosquito detail surveys, first the Mosquito batch is opened, then the user taps on the Mosquito survey in the batch list. This launches VectorDetailActivity with the Mosquito batch id as well as the mosquito id. This is so that when the Mosquito detail view is closed, or the user presses the device's back button, the appropriate Mosquito Batch view is presented.

Figures 25-27 present the detail views. Figure 26 shows how the toolbar collapses as a user scrolls through the view when it extends beyond the device's screen.

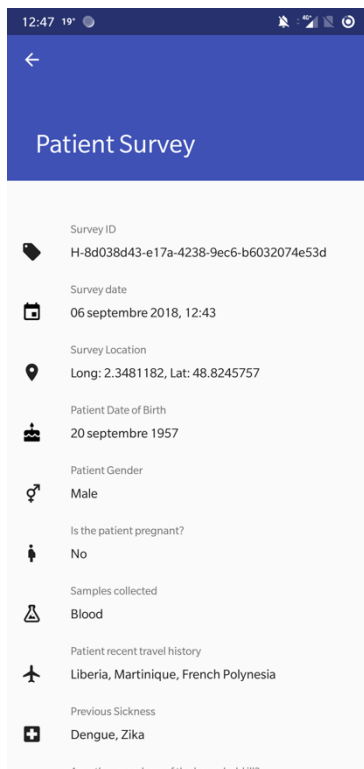


Figure 25 – Patient detail

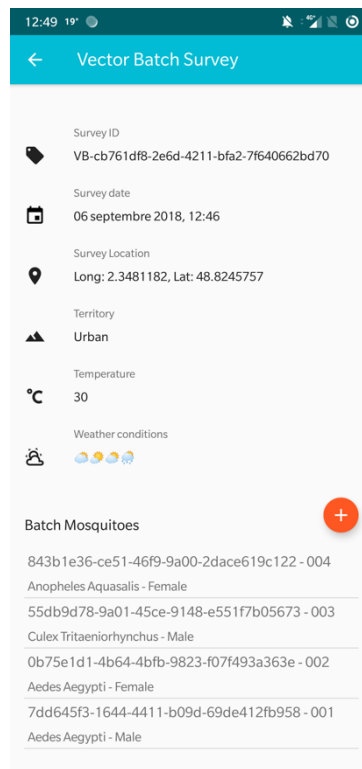


Figure 26 – Mosquito Batch detail

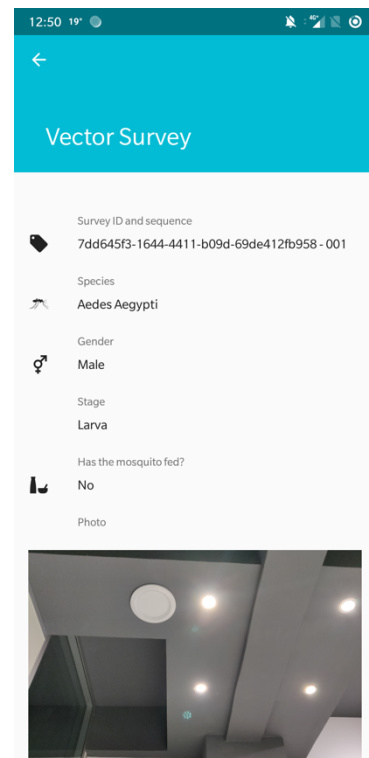


Figure 27 – Mosquito detail

7.16 Preferences

From the menu drawer in Surveys activity, shown previously in figure 18, a user can tap on the “Settings” button. This launches `AppPreferencesActivity`.

`AppPreferencesActivity` instantiates `AppPreferencesFragment`. This fragment class extends an android support component, `PreferenceFragmentCompat`, which enables a preferences view to be declared in a preferences XML file. From this the preferences view is drawn, as depicted in figure 28.

`PreferenceFragmentCompat` manages all user interactions and automatically updates an application’s shared preferences as the user changes any settings.

`AppPreferencesFragment` adds an “`onPreferenceChange`” function to listen to changes to both Primary Survey and Secondary survey options.

The listener functions check that a user does not set the Primary and Secondary survey values to the same `SurveyType`.

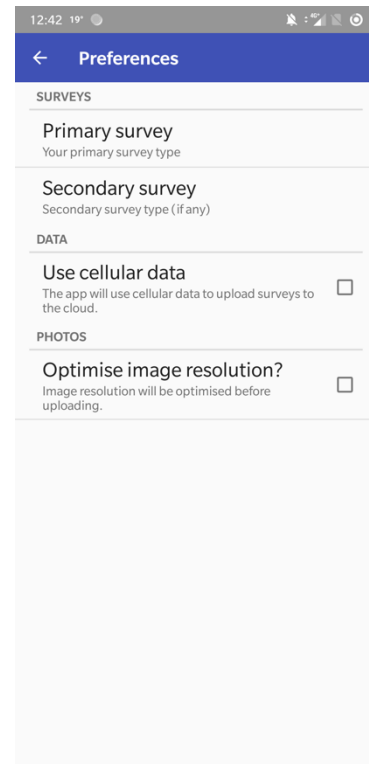


Figure 28 – Preferences view

8 TESTING

This chapter starts with explaining BDD and how it was applied to PT app and lists the user stories PT app was designed to satisfy. It then describes the automated tests written while developing PT app and concludes with user testing of PT app and the feedback the test volunteers provided.

8.1 Behaviour driven development

BDD extends on test-driven-development⁴⁴ by prioritizing business logic and user journeys when writing tests. In BDD, user stories are prepared which describe in detail what the system should do as the user interacts with a part of the system.

User stories in BDD serve to clearly describe how a system should be tested in terms of the expected behaviour of the system. BDD user stories have been somewhat formalized using a style borrowed from object-oriented analysis and design.

A user story contains three parts:

- A clear, short title.
- A short introduction stating who the actor is, what effect the actor wishes to achieve, and why the actor will derive value from the story
- The scenarios describing each specific case of the narrative. Each scenario has the following structure:
 - An initial condition believed to be true is specified
 - The event triggering the scenario is clearly described
 - Finally, it states the expected outcome

8.2 PT app user stories

The “Given When Then”⁶¹ style was used to describe PT app’s user stories. Following are the user stories that were used when developing PT app.

8.2.1 Restrict access to authorized users

Narrative

As a research team providing PT app to researchers
In order to ensure data is secured from unauthorized access
I want to require users to authenticate themselves before using the app

Scenarios

1. **Users should first sign in to the app**
Given the app is installed on a user's device
And they have not yet used the app
When the app is started
Then they must be prompted to sign in
2. **Users should be informed if invalid credentials are provided**
Given a user has been prompted to sign in
When they provide invalid credentials
Then they should be informed their credentials are invalid

8.2.2 Allow a user to sign out

Narrative

As a user using the app
In order to allow another user to use the app
I want to sign out of the app

Scenarios

1. **User signs out of the app**
Given I have signed in to the app
And another user needs to use the app
When I choose to sign out
Then the app should sign me out and present a sign in prompt

8.2.3 First time user should grant device location permission

Narrative

As a first-time user of the app

In order to add geolocation data to surveys

I should grant permission to the app to access my device's location

Scenarios

1. **First time user asked to grant location permission**

Given a user is signed in to the app

When use the app for the first time

Then they should be prompted to grant device location permissions to the app

8.2.4 First time user should grant device camera and storage permissions

Narrative

As a first-time user of the app

In order to add photos to surveys

I should grant permission to the app to access my device's camera and photo storage

Scenarios

1. **First time user asked to grant camera and storage permission**

Given a user is signed in to the app

When they use the app for the first time

Then they should be prompted to grant camera and storage permissions to the app

8.2.5 First time user should be notified about device encryption

Narrative

As a research team providing PT app to researchers

In order to ensure survey data is securely stored

I should encourage users to encrypt their android devices when using PT app

Scenarios

1. **First time user notified to encrypt device**

Given a user is signed in to the app

When they use the app for the first time

Then they should be encouraged to encrypt their device

8.2.6 User should be encouraged to record surveys

Narrative

As a research team providing PT app to researchers
In order to ensure researchers can start recording surveys
I should inform them how to start recording surveys

Scenarios

1. **User has not taken any surveys**
Given a user has opened the app
When no surveys have been stored on the device
Then they should be informed how to start recording a survey

8.2.7 User can select survey type(s)

Narrative

As a researcher performing one or both types of surveys
In order to reduce actions required to recording a survey
I should be able to configure the app to use either patient or mosquito survey types or both

Scenarios

1. **First time user selects primary survey**
Given a user is signed in to the app
When they use the app for the first time
Then they should be able to set their primary survey type
2. **First time user selects secondary survey**
Given a user is signed in to the app
And they use the app for the first time
When they have set their primary survey
Then they should be able to set a secondary survey type
3. **First time user selects no secondary survey**
Given a user is signed in to the app
And they use the app for the first time
When they have set their primary survey
Then they should be able to set no secondary survey type

4. **User changes primary survey type**
Given a user has opened the app
And they have previously set a primary survey
When they wish to change their survey type
Then the app should allow them to change their primary survey type
5. **User changes secondary survey type**
Given a user has opened the app
And they have previously set a secondary survey
When they wish to change their survey type
Then the app should allow them to change their secondary survey type
6. **User removes secondary survey type**
Given a user has opened the app
And they have previously set a secondary survey
When they wish to change their survey type
Then the app should allow them to remove their secondary survey type
7. **Secondary and Primary survey types should not match**
Given a user has opened the app
And they have previously set a primary survey type
When they set a secondary survey type matching the primary
Then the app should prevent the change and inform the user
8. **Primary and Secondary survey types should not match**
Given a user has opened the app
And they have previously set a secondary survey type
When they set a primary survey type matching the secondary
Then the app should prevent the change and inform the user
9. **User takes patient and mosquito surveys**
Given a user has opened the app
And they have set both a primary and secondary survey type
When they select the add survey option
Then the app should allow them to choose to create a patient or mosquito survey
10. **User takes only patient surveys**
Given a user has opened the app
And they have selected patient surveys as their only survey type
When they select the add survey option
Then the app should directly present the patient survey form

11. **User takes only mosquito surveys**

Given a user has opened the app

And they have selected mosquito surveys as their only survey type

When they select the add survey option

Then the app should directly present the mosquito batch survey form

8.2.8 User can choose upload via metered/unmetered network

Narrative

As a user of the app

In order to reduce roaming costs

I need to choose whether data is transmitted by cellular data or not

Scenarios

1. **Cellular transmission should be disabled by default**

Given a user has never enabled cellular transmission

When the device is not connected to a WiFi network

Then survey transmission to the server should be prevented

2. **Cellular transmission can be enabled**

Given cellular transmission is disabled

When the user wishes to enable cellular transmission

Then the app should allow them to enable cellular transmission

3. **Cellular transmission can be disabled**

Given cellular transmission is enabled

When the user wishes to disable cellular transmission

Then the app should allow them to disable cellular transmission

8.2.9 User can choose photo optimization

Narrative

As a user of the app

In order to optimize device storage space

I need to choose whether photos are optimized before saving or not

Scenarios

1. **Photo optimization should be disabled by default**
Given a user has never enabled photo optimization
When a survey with a photo is saved
Then the photo should not be optimized
2. **Photo optimization can be enabled**
Given photo optimization is disabled
When the user wishes to enable photo optimization
Then the app should allow them to enable photo optimization
4. **Photo optimization can be disabled**
Given photo optimization is enabled
When the user wishes to disable photo optimization
Then the app should allow them to disable photo optimization

8.2.10 Capture location in survey

Narrative

As a user recording surveys

In order to be able to geolocate the survey data

I need to be able to capture the device's location

Scenarios

1. **Take a patient survey**
Given a user has granted permission to access the device's location
And the device's location services are enabled
When they are recording a patient survey
Then the app should allow the device's location to be recorded in the survey
2. **Take a mosquito batch survey**
Given a user has granted permission to access the device's location
And the device's location services are enabled
When they are recording a mosquito batch survey
Then the app should allow the device's location to be recorded in the survey
3. **Prompt when location services disabled**
Given the device's location services are disabled
When the user attempts to record the device's location
Then the app should prompt them to enable location services
4. **Retrieve location when location services enabled after prompt**
Given a user has been prompted to enable location services
When the user enables location services
Then the app should retrieve the device's location
5. **Prompt when location permissions revoked**
Given a user has revoked location permission or not granted them
When the user attempts to record the device's location
Then the app should prompt them to grant this permission
6. **Retrieve location when location permission granted after prompt**
Given a user has been prompted to grant location permission
When the user grants this permission
Then the app should retrieve the device's location

8.2.11 Capture photo in survey

Narrative

As a user recording mosquito detail surveys

In order to have a photo record of the mosquito

I need to be able to capture a photo of the mosquito

Scenarios

1. **Capture photo in mosquito detail survey**
Given a user has granted permission to access the device's camera and storage
When they are recording a mosquito detail survey
Then the app should allow a photo to be captured, stored on the device, and recorded in the survey
2. **Replace photo in mosquito detail survey**
Given a user has captured a photo and assigned it to the survey
And the survey has not yet been saved
When they wish to replace the photo
Then the app should allow a new photo to be captured and replace the previous photo
3. **Prompt when camera and storage permissions revoked**
Given a user has revoked camera and storage permissions or not granted them
When the user attempts to capture a photo
Then the app should prompt them to grant these permissions
4. **Capture photo when camera and storage permissions granted after prompt**
Given a user has been prompted to grant camera and storage permissions
When the user grants these permissions
Then the app should allow them to capture a photo

8.2.12 Assign survey date to survey automatically

Narrative

As a user recording surveys

In order to record the survey date

I need the present date to be assigned by default

Scenarios

1. **Assign present date as survey date**

Given a user has selected to record a survey

When the survey form is initialised

Then the app should automatically set the survey date to the present date

8.2.13 Change survey date in survey

Narrative

As a user recording surveys

In order to record a survey captured at a previous date

I need to be able to change the survey date

Scenarios

1. **Allow survey date change**

Given a user has selected to record a survey

When the survey form is initialised with the present date

Then the user should be permitted to change the survey date

8.2.14 Assign unique survey identifier

Narrative

As a user recording surveys

In order to distinguish a survey from the others

I need an identifier to be assigned to the survey

Scenarios

1. **Survey should automatically generate a unique identifier**

Given a user has selected to record a survey

When the form is initialised

Then the app should automatically generate a unique identifier and assign it to the survey

8.2.15 Select one data value within restricted set

Narrative

As a researcher recording surveys

In order to restrict survey data values to one value in a provided set

I should be able to select only one of the permissible values

Scenarios

1. **Provide restricted set of values**

Given a user is recording a survey

When they are recording data for a restricted field

Then the app should only allow one value to be selected

8.2.16 Select multiple data values within restricted set

Narrative

As a researcher recording surveys

In order to record multiple values within a field restricted to a discrete set of values

I need to be able to select multiple values from the permissible set

Scenarios

1. **Select value from permissible values**

Given a user is recording a survey

When they are recording data for a restricted field that accepts multiple values

Then the app should allow any of the permissible values to be selected

2. **Select additional value from permissible values**

Given a user is recording a survey

When they have recorded data for a restricted field that accepts multiple values

When they add another value

Then the app should only allow any of the permissible values to be selected

3. **Only allow as many values as exist in the set**

Given a user is recording a survey

When they have recorded all the values in the set

Then the app should not permit more values to be selected

8.2.17 Save a survey

Narrative

As a researcher recording surveys

In order to retain the survey data

I need the data to be saved securely

Scenarios

1. **App should allow user to save the survey**

Given the survey form has been filled in

When the user wishes to save the survey

Then the app should store the survey in the local database

8.2.18 Record a patient survey

Narrative

As a researcher collecting patient samples
In order capture the required meta data
I need to be able to record a patient survey

Scenarios

1. **Start a patient survey**
Given a user has selected patient survey as their primary or secondary survey type
When they choose to start recording a patient survey
Then the patient survey form should be presented
2. **Patient survey should allow user to navigate between form views**
Given the patient survey form has been selected
When the user wishes to select a previous or subsequent view
Then the app should allow the user to navigate to the desired view
3. **Patient survey should allow a survey to be flagged for review**
Given the patient survey form has been selected
When the user wishes to flag the survey
Then the app should allow the survey to be flagged

8.2.19 Record a mosquito batch survey

Narrative

As a researcher collecting a batch of mosquitos
In order capture the required meta data for the batch
I need to be able to record a mosquito batch survey

Scenarios

1. **Start a mosquito batch survey**
Given a user has selected mosquito survey as their primary or secondary survey type
When they choose to start recording a mosquito batch survey
Then the mosquito batch survey form should be presented

8.2.20 Record a mosquito detail survey and assign to batch

Narrative

As a researcher collecting a batch of mosquitos

In order to capture meta data for each mosquito in the batch

I need to be able to record a mosquito detail survey

Scenarios

1. **Start a mosquito detail survey**

Given a user has recorded a mosquito batch survey

And they have opened the mosquito batch survey

When they choose to start recording a mosquito detail survey

Then the mosquito detail survey form should be presented and the survey should be associated with its batch survey

8.2.21 List all saved surveys

Narrative

As a researcher recording surveys

In order to see all surveys that have been saved

I need to be presented with a list of surveys

Scenarios

1. **List all patient and mosquito batch surveys**

Given a user has saved patient and mosquito batch surveys

When they have opened the app

Then the app should present a list of all patient and mosquito batch surveys

2. **Distinguish between patient and mosquito batch surveys**

Given a user has saved patient and mosquito batch surveys

When the list of all patient and mosquito batch surveys is presented

Then the app should distinguish between patient and mosquito batch surveys

3. **List all mosquito detail surveys in a batch**

Given a user has saved mosquito detail surveys in a batch

When the mosquito batch survey is being reviewed

Then the app should present a list of all mosquito detail surveys in the batch

8.2.22 Review saved surveys

Narrative

As a researcher recording surveys

In order to review a survey

I need to be presented with the survey data

Scenarios

1. **Review patient survey**

Given a user has saved patient surveys

When they select a patient survey from the survey list

Then the app should present the data for the selected patient survey

2. **Review mosquito batch survey**

Given a user has saved mosquito batch surveys

When they select a mosquito batch survey from the survey list

Then the app should present the data for the selected mosquito batch survey

3. **Review mosquito detail survey**

Given a user has saved mosquito detail surveys in a batch

And the mosquito batch survey is being reviewed

When the user selects a mosquito detail survey from the list

Then the app should present the data for the selected mosquito detail survey

8.2.23 Securely transmit survey to server

Narrative

As a researcher recording surveys

In order to collaborate with my research team

I need my saved surveys to be securely transmitted to the server

Scenarios

1. **Transmit metered**

Given a user has selected transmission over metered networks

And they have saved surveys on the device

When the system has any internet connection

Then the system should transmit the surveys

2. **Transmit unmetered when connected to WIFI**

Given a user has selected transmission over unmetered networks only

And they have saved surveys on the device

When the system has a WIFI internet connection

Then the system should transmit the surveys

3. **Transmit unmetered when not connected to WIFI**

Given a user has selected transmission over unmetered networks only

And they have saved surveys on the device

When the system does not have a WIFI internet connection

Then the system should not transmit the surveys

8.2.24 Securely transmit survey photos to server

Narrative

As a researcher capturing photos for mosquito surveys
In order to collaborate with my research team
I need the photos to be securely transmitted to the server

Scenarios

1. **Transmit metered**
Given a user has selected transmission over metered networks
And they have saved survey photos on the device
When the system has any internet connection
Then the system should transmit the photos
2. **Transmit unmetered when connected to WIFI**
Given a user has selected transmission over unmetered networks only
And they have saved survey photos on the device
When the system has a WIFI internet connection
Then the system should transmit the photos
3. **Transmit unmetered when not connected to WIFI**
Given a user has selected transmission over unmetered networks only
And they have saved photos on the device
When the system does not have a WIFI internet connection
Then the system should not transmit the photos

8.3 Unit and Instrumentation Testing

Instrumentation tests are required for components that depend on Android features and require a device or emulator to run. Unit tests run on the JVM and require no device or emulation.

Full test coverage was not achieved for PT app; however, tests were written to cover a significant portion of key functionality. Test results, valid for PT app version Beta-1.6.0, are included with the project attached to this report, in the “Test Results” folder.

8.3.1 Unit tests

Unit tests were created for all components that had no dependency on the Android system. Components include Kotlin extension functions and the OnBoardingViewModel class.

Boolean Extensions

Test Description	Expected Result	Pass/Fail
Boolean extension ‘asYesOrNo’ returns ‘No’ if false	‘No’	PASS
Boolean extension ‘asYesOrNo’ returns ‘Yes’ if true	‘Yes’	PASS

OnBoardingViewModel

Test Description	Expected Result	Pass/Fail
Allow user to set primary survey type to vector	Shared Preferences updated	PASS
Allow user to set primary survey type to patient	Shared Preferences updated	PASS
Save onboarding status to complete	Shared Preferences updated	PASS
Allow user to set secondary survey type to none	Shared Preferences updated	PASS
Allow user to set secondary survey type to vector	Shared Preferences updated	PASS
Allow user to set secondary survey type to patient	Shared Preferences updated	PASS
Reset onboarding status	Shared Preferences updated	PASS

8.3.2 Instrumentation and Integration tests

Instrumentation and integration tests were written to test as many of the scenarios described in PT app's user stories as possible within the allocated time.

The results below are valid for version Beta-1.6.0 of PT app which was delivered with this report.

First time user

Test Description	Expected Result	Pass/Fail
A user must sign in to the app	FirebaseAuthUI shown if user has not signed in	PASS
A user can sign out of the app	User is signed out and FirebaseAuthUI is shown	PASS
Allow user to select Patient surveys as primary Survey type	OnBoarding view shows Patient option and stores value when selected	PASS
Allow user to select Mosquito surveys as primary Survey type	OnBoarding view shows Patient option and stores value when selected	PASS
Allow user to set Patient as secondary Survey Type	OnBoarding view shows Yes option when Mosquito selected for Primary and stores Patient value when selected	PASS
Allow user to set Mosquito as secondary Survey Type	OnBoarding view shows Yes option when Patient selected for Primary and stores Mosquito value when selected	PASS
Allow user to set None as secondary Survey Type	OnBoarding view shows No option when either Patient or Mosquito selected for Primary and stores value when selected	PASS
Ask for Location permission	Once test has run cannot reset permissions, manually tested	No automation
Ask for Camera and Storage permission	Once test has run cannot reset permissions, manually tested	No automation

Surveys list

Test Description	Expected Result	Pass/Fail
Welcome message shown if no surveys recorded, directs user to 'add survey' button	Message should be shown if no surveys have been recorded	PASS
List of surveys shown if user has recorded surveys	A list of surveys should be shown	PASS
Welcome message hidden if user has recorded surveys	Message should be hidden if no surveys have been recorded	PASS
'Add survey' button goes directly to Mosquito Batch form if Primary survey is Mosquito and Secondary is None	Mosquito Batch form should be shown	PASS
'Add survey' button goes directly to Patient form if Primary survey is Patient and Secondary is None	Patient form should be shown	PASS
'Add survey' button shows Mosquito and Patient options when both Primary and Secondary surveys are not None	A dialog is shown with a button each for Patient and Mosquito Batch surveys	PASS
Allow user to set None as secondary Survey Type	OnBoarding view shows No option when either Patient or Mosquito selected for Primary and stores value when selected	PASS
Allow a user to review a Patient survey when selecting it from the survey list	Patient survey details shown	PASS
Allow a user to review a Mosquito Batch survey when selecting it from the survey list	Mosquito Batch details shown	PASS

SurveysLlistViewModel

Test Description	Expected Result	Pass/Fail
Should return a list of surveys if a user has recorded surveys	A list of 'UiSurvey' objects	PASS
Should return an empty list if a user has not recorded surveys	An empty list	PASS

Preferences

Test Description	Expected Result	Pass/Fail
A user should be able to change their Primary survey type	'true', if Primary does not equal Secondary. Otherwise 'false'	PASS
A user should be able to change their Secondary survey type	'true', if Secondary does not equal Primary. Otherwise 'false'	PASS
A user should be able to set their Secondary survey type to None	Shared preferences are updated and Secondary survey type is 'None'	PASS
A user should be able to enable cellular data transmission	Shared preferences are updated and cellular transmission is enabled	PASS
A user should be able to disable cellular data transmission	Shared preferences are updated and cellular transmission is disabled	PASS
A user should be able to enable image optimization	Shared preferences are updated and optimization is enabled	PASS
A user should be able to disable image optimization	Shared preferences are updated and optimization is disabled	PASS

Photo repositories

Test Description	Expected Result	Pass/Fail
If no photo is provided to DevicePhotoRepository, it should not be stored	A null photo is not written to the file system	PASS
If a photo is provided to DevicePhotoRepository, it should be stored	The photo is saved in the specified directory with the specified file name	PASS
If a photo is provided to RemotePhotoRepository, it should be stored in Firebase Cloud Storage	The photo is uploaded	PASS

Realm survey repository

Test Description	Expected Result	Pass/Fail
Should store a Mosquito Batch survey	When a survey is stored, and then retrieved, it should match the original	PASS
Should store a Mosquito detail survey	When a survey is stored, and then retrieved, it should match the original	PASS
Should store a Patient survey	When a survey is stored, and then retrieved, it should match the original	PASS
Should retrieve a Mosquito Batch survey when a survey id is provided	The survey matching the id, or nothing	PASS
Should retrieve a Mosquito detail survey when a survey id is provided	The survey matching the id, or nothing	PASS
Should retrieve a Patient survey when a survey id is provided	The survey matching the id, or nothing	PASS

Should retrieve Mosquito detail surveys belonging to a Mosquito Batch when a survey id and batch id are provided	A list of surveys matching the survey id and batch id, or an empty list	PASS
Should retrieve a list of Patient surveys	Given Patient as a type, a list of Patient surveys if any exist, or an empty list	PASS
Should retrieve a list of Mosquito Batch surveys	Given Mosquito Batch as a type, a list of Mosquito Batch surveys if any exist, or an empty list	PASS
Should retrieve a list of surveys to upload	All surveys for the given type where 'uploadedAt' is empty, or an empty list if all have been uploaded	PASS

Firebase Firestore survey repository

Test Description	Expected Result	Pass/Fail
Should store a Mosquito Batch survey	When a survey is uploaded, and then retrieved, it should match the original	PASS
Should store a Mosquito detail survey	When a survey is uploaded, and then retrieved, it should match the original	PASS
Should store a Patient survey	When a survey is uploaded, and then retrieved, it should match the original	PASS

Add a Patient survey

Test Description	Expected Result	Pass/Fail
It should set a unique id automatically	When the survey form is presented, a unique id is set	PASS

It should set the present date as the survey date automatically	When the survey form is presented, the date matches the present date	PASS
It should retrieve the device's location when the location field is tapped	A Location object containing coordinates is returned and set in the form field	PASS
It should allow multiple values for a patient's infection history	A user can add and remove multiple values from within the specified data set	PASS
It should allow multiple values for a patient's travel history	A user can add and remove multiple values from within the specified data set	PASS
It should allow multiple values representing the physical sample types associated with the survey	A user can add and remove multiple values from within the specified data set	PASS
It should allow multiple values for a patient's current infections; each value contains the disease and symptoms	A user can select a current infection, and symptoms associated, multiple times.	PASS
A user should be able to navigate back and forth between the survey form pages	When the user taps the 'BACK' or 'NEXT' buttons, the view updates with the appropriate page	PASS
A user should be able to save a survey	When the form is filled in, and the save button is tapped, the survey is stored in the local surveys repository	PASS

Add a Mosquito Batch survey

Test Description	Expected Result	Pass/Fail
It should set a unique id automatically	When the survey form is presented, a unique id is set	PASS
It should set the present date as the survey date automatically	When the survey form is presented, the date matches the present date	PASS

It should retrieve the device's location when the location field is tapped	A Location object containing coordinates is returned and set in the form field	PASS
A user should be able to save a survey	When the form is filled in, and the save button is tapped, the survey is stored in the local surveys repository	PASS

Add a Mosquito detail survey

Test Description	Expected Result	Pass/Fail
It should use the provided survey id	The survey is saved with the id provided	PASS
It should use the provided Mosquito Batch id	The survey is saved with the batch id provided	PASS
A user should be able to save a survey	When the form is filled in, and the save button is tapped, the survey is stored in the local surveys repository	PASS

Utility tests

Besides the above tests, some of the Kotlin extensions written for android components were tested. They are omitted for brevity.

8.4 User testing

As stated in chapter 4, user testing was performed at the end of stage 1 and stage 2 of PT app implementation.

8.4.1 Stage one user testing

In stage one, the Google Play store was used to upload the app and create a “Beta release” which allowed it to be restricted to invited users only. This worked satisfactorily and users were able to download and install the app.

This round of testing was rather informal and released to ZP team only in order to get a general consensus on the suitability of the app and determine if it ran on a variety of devices. Users were also requested to report any bugs they found.

A defect was discovered by one of the users related to the onboarding experience and requesting device permissions. An exceptional use case had not been accounted for and the user’s testing uncovered this defect. This was subsequently corrected and the appropriate test added.

8.4.2 Stage two user testing

In stage two, volunteers in the following countries were enlisted in testing PT app: UK, USA, Australia, and France. Each was given a particular configuration to apply in the onboarding experience and were encouraged to take surveys. As two users in stage two reported difficulty locating the app in the Google Play store, the signed APK (application package) was uploaded to a private web server and a download URL was provided.

At this stage, transmission to the remote server functionality had been implemented. Part of the test process involved the author using the Firebase Cloud Firestore web interface to investigate the data collected by the testers.

In total, six volunteers tested the app. The configurations selected included different possible combinations of survey types (ST) and other settings:

1. Primary ST: Mosquito, secondary: Patient. “Use cellular data” and “optimize photos” settings enabled.
2. Primary ST: Patient, secondary: None. “Use cellular data” and “optimize photos” settings left at default setting (disabled).
3. Primary ST: Patient, secondary: Mosquito. “Use cellular data” and “optimize photos” settings left at default setting (disabled). Testing user was requested to take surveys when not connected to WIFI, and then connect to WIFI after a period of 15 minutes.

4. Primary ST: Mosquito, secondary: None. “Use cellular data” and “optimize photos” settings enabled.
5. Primary ST: Mosquito, secondary: Patient. “Use cellular data” and “optimize photos” settings disabled. Testing user was requested to toggle these settings of and on using the settings view.
6. Primary ST: Patient, secondary: Mosquito. Use cellular data” and “optimize photos” settings left at default setting (disabled).

User testing results

All testing users reported that the app was easy to use with no instruction, and were able to complete their allocated surveys for their selected survey types.

One user reported that the orange button used to add vector detail surveys wasn't clear to them. This was a specific feature agreed with ZP team, that a vector detail be added from the vector batch screen, and was unfamiliar to a user without detailed knowledge of the relationship between Mosquito Batch and Mosquito surveys. It is worth noting as a feature that could be improved or made clearer to users.

A number of users commented on the “other” value in some of the spinners. It was agreed with ZP that when a user selected “other” as a value, this would suffice. An app update could provide functionality allowing a bespoke value to be entered when a user selected “other”.

Some of the user's responses are included below.

“I took the mosquito survey and it worked well. I would have liked to see Fahrenheit as an option for temperature values but that's just because I'm US-based. Might be useful for Americans in the field though”

“I like how the survey screen tells you that a survey's been uploaded with the little cloud icon”

“The icons in the app are fantastic! People don't give enough thought into the visual elements”

“The icons in the survey screens are great”

“Great app, easy to get started and use. Makes the entries clear so you can't make mistakes.”

“The app is very simple and easy to navigate. It's obvious the user experience has been thought through”

Defects

User testing revealed one usability defect: when a patient gender was set to “Male”, a user could still select “Yes” as the answer to “Is the patient pregnant?”. This was subsequently corrected in the version of the app included with this report, Beta-1.6.0

8.4.3 Summary

A fair amount of testing was performed by geographically distributed users, testing a variety of scenarios. Feedback was generally positive and a couple of defects were caught and corrected.

All users were able to perform their designated tasks and were happy with PT app’s design, usability, and functionality.

The author was able to verify that user surveys had uploaded to Cloud Firestore and the documents contained expected data.

Throughout implementation PT app was tested by the author on OnePlus 3 and OnePlus 6 smartphones, running Android version 7 and 8.1 respectively.

In conclusion, user testing proved a benefit to the development of PT app, some defects were detected and usability issues corrected.

9 SUMMARY AND EVALUATION

This chapter evaluates the PT app project and presents a list of enhancements that could be implemented to improve PT app. It concludes with a summary of the project and the author's experience developing PT app.

9.1 Evaluation

Overall, PT app implementation can be considered a success. With one exception, all the 'Must Have' requirements listed in 3.2.1. were met and the app was successfully tested by two groups of users at different stages in development. User reviews were positive and all test volunteers were able to complete their assigned tasks.

One MH requirement to, 'allow a user to attach a DNA fragment to a Mosquito survey' had to be re-classified to "Could Have" status and was not implemented. This was due to the project's time-constraints. PT app is a sizeable project and the work completed within the allocated time frame was significant.

In its current phase, PT app provides a reasonable solution to the main problems it attempted to solve as it:

- allows researchers to record surveys when their device is offline and automates the transmission of survey data once the device is online.
- provides an easy to use mechanism for capturing survey data, consolidating methods currently in use, paper forms and disparate applications, into a unified user experience.
- any mention about security and authentication as part of the aims?

The additional non-functional objective, gaining experience implementing an Android app was also achieved.

9.2 Improvements

The current version of PT app, Beta-1.6.0, delivered all 'Must Have' requirements as per their final classification. Further development could involve improving the user experience and providing additional functionality that further serves the overall objectives, and improves the application overall. PT app was designed to be extensible and modular so implementing enhancements should not present any obstacles for an experienced software engineer.

The following suggested list of improvements is by no means exhaustive.

9.2.1 Mosquito survey: allow a DNA sample to be copied from a portable device

This specification, originally classified 'Must Have', had to be omitted for reasons stated in 9.1, above. The specification calls for a configured device to present a sampled DNA fragment in an agreed format and copied to PT app via some network connection. The sample involves a DNA extract of a specified Mosquito and would be stored with the Vector survey data.

9.2.2 Mosquito survey: allow a photo to be selected from the device's photo gallery

Rather than taking a photo during each survey, a researcher using PT app may prefer to take a number of photos of all the mosquitos in a batch, and be able to select photos from the device's gallery to attach to the survey.

9.2.3 Mosquito survey: allow a sound recording of a mosquito in flight

This feature was suggested by a member of ZP team testing PT app in stage one of user testing. They had read a paper⁶² which proposes using mobile device microphones to record the frequencies of a mosquito's wingbeat in flight, and using this to identify individual species. The data referenced in the paper reported that *"smartphone microphones are now sensitive enough to record the sound of mosquito wings in flight"*.

PT app could use a suitable microphone on the device it is installed on to perform a recording and attach it to the mosquito survey for subsequent upload to the server.

9.2.4 Download surveys from server

The final version of PT app transmits surveys in one direction only, from the smartphone device to the server. The FirebaseSurveyRepository implementation omitted the "get" operations as the functionality was not required for PT app. The proposed feature would not be difficult to implement and would allow survey data to be corrected on the server, with the corrections being downloaded to update the device's version. Additionally, it would enable the 9.2.5 improvement.

9.2.5 Identify authenticated user in surveys

If an identifier representing the user recording the survey was attached to each survey, it would be possible to download all previously recorded surveys if the user changed device, or had to reset the device, or uninstall and reinstall PT app. This would first require the implementation of 9.2.4.

9.2.6 Further validation in surveys

PT app currently does not validate whether all fields in a survey have been completed as it was not required by ZP team. In fact ZP team suggested that no field aside from ID and

current date was required. Further analysis could be performed to determine which fields are essential and the user prompted if required fields are not filled in.

9.2.7 Data binding

Android 8 was released while PT app was in development and introduced useful new features. At that time a fair amount of development was complete and it was not practical to reprogram the app to take advantage of additional features.

One feature, 'Data Binding'⁶³, is a support library that allows declarative rather than programmatic binding of view elements to data objects. This binding works in both directions and allows a model to be updated as a user inputs data. Implementing this feature would remove the need for a number of methods and allow PT app to more closely adhere to the MVVM pattern.

9.2.8 Save survey state when PT app is paused

Due to time constraints, survey state is not saved when the Activity lifecycle "onStop" method is called. Android calls this method in any active activity in an app when the app has been running in the background for a long time without being used and the system requires resources to be freed for more active applications.⁶⁴ This means that in certain cases a user could start filling in a survey and if they did not save the survey prior to switching apps, their unsaved data would be lost.

9.2.9 Allow surveys to be edited on the device

This feature was a 'Could Have' feature and so was not implemented. It would, simply, allow a user to choose to edit a survey if the data required modification. This feature would benefit from implementing the Data binding feature mentioned in 9.2.7.

9.3 Summary

PT app project was a challenging learning experience in designing and developing Android applications, and in working with the new programming language, Kotlin.

PT app has wide application and could prove useful to research teams other than ZP who perform surveys related to mosquito-borne pathogens. Some modifications may be required, but due to the way PT app was designed these could be trivial, subject to the requirements.

An idea proposed in this project's proposal could cause PT app to be more generally useful. The idea is to construct a DSL representing a survey form, PT app would then use this DSL, which could be transmitted from a server, to dynamically construct forms at runtime. This would be a fairly challenging project but could prove interesting.

10 BIBLIOGRAPHY

- 1 London School of Hygiene & Tropical Medicine (2016) “Zika virus surveillance in human and mosquito populations in Cape Verde” . , p. 9777170. [online] Available from: <http://www.bloomsbury.ac.uk/studentships/studentships-2017/zika-virus-surveillance-in-human-and-mosquito-populations-in-cape-verde> (Accessed 14 December 2016)
- 2 Hennessey, Morgan, Fischer, Marc and Staples, J. Erin (2016) ‘Zika Virus Spreads to New Areas — Region of the Americas, May 2015–January 2016’. *MMWR. Morbidity and Mortality Weekly Report*, 65(3), pp. 55–58. [online] Available from: <http://www.cdc.gov/mmwr/volumes/65/wr/mm6503e1.htm> (Accessed 25 March 2017)
- 3 Faria, N. R., Azevedo, R. d. S. d. S., Kraemer, M. U. G., Souza, R., et al. (2016) ‘Zika virus in the Americas: Early epidemiological and genetic findings’. *Science*, 352(6283), pp. 345–349. [online] Available from: <http://www.sciencemag.org/cgi/doi/10.1126/science.aaf5036> (Accessed 25 March 2017)
- 4 Barreiro, Pablo (2007) ‘Evolving RNA Virus Pandemics: HIV, HCV, Ebola, Dengue, Chikungunya, and now Zika!’ *AIDS reviews*, 18(1), pp. 54–5. [online] Available from: <http://www.ncbi.nlm.nih.gov/pubmed/27028271>
- 5 Barzon, Luisa, Trevisan, Marta, Sinigaglia, Alessandro, Lavezzo, Enrico and Pal??, Giorgio (2016) ‘Zika virus: From pathogenesis to disease control’. *FEMS Microbiology Letters*, 363(18), pp. 1–17.
- 6 Luheshi, Leila, Raza, Sobia, Moorthie, Sowmiya, Hall, Alison, et al. (2015) *Pathogen Genomics Into Practice*, [online] Available from: <http://www.phgfoundation.org/reports/16857/> (Accessed 28 March 2017)
- 7 ScienceDaily (n.d.) ‘Zika Virus Research News -- ScienceDaily’. [online] Available from: https://www.sciencedaily.com/news/plants_animals/zika_virus_research/ (Accessed 31 March 2017)
- 8 Weiss Robert & Plante Beth (2018) ‘World Map of Areas With Risk of Zika’. *Online*, p. 1. [online] Available from: https://www.fertilitycenter.com/fertility_cares_blog/world-map-of-areas-with-risk-of-zika/ (Accessed 12 September 2018)
- 9 Google (n.d.) ‘Firebase | App success made simple’. 2016. [online] Available from: <https://firebase.google.com/> (Accessed 6 April 2017)
- 10 Paul, Gil (n.d.) ‘What Is “SaaS” (Software as a Service)?’ *About*. [online] Available from: http://netforbeginners.about.com/od/s/f/what_is_SaaS_software_as_a_service.htm (Accessed 12 September 2018)
- 11 Firebase, Inc. (2018) ‘Firebase Authentication | Firebase’. *Firebase, Inc.*, p. 1. [online] Available from: https://firebase.google.com/docs/auth/?gclid=CjwKCAjw8uLcBRACEiwAaL6MSX1xXFqWXYCnrqL-sqPLz5fvkbY1dyAmanpbmgXz4DjFd_HW-1HuhoCqIAQAvD_BwE (Accessed 12 September 2018)
- 12 Firebase (n.d.) ‘Cloud Firestore | Firebase’. [online] Available from: <https://firebase.google.com/docs/firestore/> (Accessed 12 September 2018)
- 13 Google-inc (2018) ‘Cloud Storage | Firebase’. , p. 1. [online] Available from: <https://firebase.google.com/docs/storage/> (Accessed 12 September 2018)
- 14 Marvel (n.d.) ‘Turn sketches, mockups and designs into web, iPhone, iOS, Android and Apple Watch app prototypes.’ [online] Available from: <https://marvelapp.com> (Accessed 24 March 2017)
- 15 Wikipedia (n.d.) ‘MoSCoW Method’. [online] Available from: https://en.wikipedia.org/wiki/MoSCoW_method (Accessed 12 September 2018)
- 16 Google (n.d.) ‘Design - Material Design’. [online] Available from: <https://material.io/design/> (Accessed 12 September 2018)
- 17 JetBrains s.r.o. (n.d.) ‘Kotlin Programming Language’. [online] Available from: <http://kotlinlang.org/> (Accessed 30 March 2017)
- 18 JetBrains s.r.o. (n.d.) ‘JetBrains: Development Tools for Professionals and Teams’.

- [online] Available from: <https://www.jetbrains.com/> (Accessed 30 March 2017)
- 19 Anon (n.d.) 'Boilerplate code - Wikipedia'. [online] Available from: https://en.wikipedia.org/wiki/Boilerplate_code (Accessed 30 March 2017)
- 20 Google (n.d.) 'NullPointerException | Android Developers'. [online] Available from: <https://developer.android.com/reference/java/lang/NullPointerException> (Accessed 12 September 2018)
- 21 Titus, Jason (n.d.) 'Android Developers Blog: Google I/O 2017: Empowering developers to build the best experiences across platforms'. [online] Available from: <https://android-developers.googleblog.com/2017/05/google-io-2017-empowering-developers-to.html> (Accessed 15 September 2018)
- 22 Anon (n.d.) 'Kot. Academy'. [online] Available from: <https://blog.kotlin-academy.com/> (Accessed 15 September 2018)
- 23 JetBrains (n.d.) 'Kotlin Programming Language'. [online] Available from: <https://kotlinlang.org/docs/reference/> (Accessed 15 September 2018)
- 24 Anon (n.d.) 'A Complete Guide To Learn Kotlin For Android Development'. [online] Available from: <https://blog.mindorks.com/a-complete-guide-to-learn-kotlin-for-android-development-b1e5d23cc2d8> (Accessed 15 September 2018)
- 25 Thornsby, Jessica (n.d.) 'Java vs. Kotlin: Should You Be Using Kotlin for Android Development?' [online] Available from: <https://code.tutsplus.com/articles/java-vs-kotlin-should-you-be-using-kotlin-for-android-development--cms-27846> (Accessed 21 March 2017)
- 26 Wharton, Jake (n.d.) 'Using Project Kotlin for Android'. [online] Available from: <https://docs.google.com/document/d/1ReS3ep-hjxWA8kZioYqDbEhCqTt29hG8P44aA9W0DM8/edit?hl=en&forcehl=1#heading=h.wsmcine11x68> (Accessed 21 March 2017)
- 27 Anon (n.d.) 'Calling Java from Kotlin - Kotlin Programming Language'. [online] Available from: <https://kotlinlang.org/docs/reference/java-interop.html> (Accessed 15 September 2018)
- 28 Bayramoğlu, Yahya (n.d.) 'LocationManager'. [online] Available from: <https://github.com/yayaa/LocationManager>
- 29 Anon (2017) 'Realm (database)'. *Wikipedia*. [online] Available from: <https://realm.io/products/realm-database> (Accessed 14 September 2018)
- 30 Realm (2017) 'Realm: Create reactive mobile apps in a fraction of the time'. *Realm Help-Documentation*. [online] Available from: <https://realm.io/docs/java/latest/#encryption> (Accessed 14 September 2018)
- 31 Murcia, Víctor Manuel Pineda (n.d.) 'Kotlin Realm Extensions'. [online] Available from: <https://github.com/vicpinm/Kotlin-Realm-Extensions>
- 32 StepStone Services (n.d.) 'Android Material Stepper'. [online] Available from: <https://github.com/stepstone-tech/android-material-stepper>
- 33 Kotlin (n.d.) 'Anko'.
- 34 Google Android (2015) 'Introduction to Android | Android Developers'. *Android Developers*. [online] Available from: <https://developer.android.com/guide/components/activities/intro-activities> (Accessed 12 September 2018)
- 35 Google (2009) 'Android Developers'. *httpdeveloperandroidcom*. [online] Available from: <https://developer.android.com/guide/topics/ui/declaring-layout> (Accessed 12 September 2018)
- 36 Google (2009) 'Android Developers'. *httpdeveloperandroidcom*. [online] Available from: <https://developer.android.com/guide/components/fragments> (Accessed 12 September 2018)
- 37 Android.com (n.d.) 'Services | Android Developers'. [online] Available from: <https://developer.android.com/guide/components/services.html> (Accessed 30 March 2017)
- 38 Filters, Intent and Developers, Android (2012) 'Intents and Intent Filters'. , pp. 1–11. [online] Available from: <https://developer.android.com/guide/components/intents-filters> (Accessed 15 September 2018)
- 39 Anon (n.d.) 'Starting Another Activity'. [online] Available from: <https://stuff.mit.edu/afs/sipb/project/android/docs/training/basics/firstapp/starting->

- activity.html (Accessed 14 September 2018)
- 40 Lenarduzzi, Valentina and Taibi, Davide (2016) 'MVP Explained: A Systematic Mapping Study on the Definitions of Minimal Viable Product', in *2016 42th Euromicro Conference on Software Engineering and Advanced Applications*, IEEE, pp. 112–119. [online] Available from: <http://ieeexplore.ieee.org/document/7592786/> (Accessed 24 March 2017)
- 41 Gossman, John (n.d.) 'Tales from the Smart Client : Introduction to Model/View/ViewModel pattern for building WPF apps'. [online] Available from: <http://blogs.msdn.com/johngossman/archive/2005/10/08/478683.aspx> (Accessed 15 September 2018)
- 42 Edition, Farsi (n.d.) 'Repository Pattern'. [online] Available from: <https://deviq.com/repository-pattern/> (Accessed 15 September 2018)
- 43 Booch, Grady., Rumbaugh, James. and Jacobson, Ivar. (1998) *The Unified Modeling Language User Guide* 2nd ed., Addison-Wesley. [online] Available from: <http://www.informit.com/store/unified-modeling-language-user-guide-9780321267979> (Accessed 14 September 2018)
- 44 North, Dan (2006) 'Introducing BDD'. *Better Software Magazine*. [online] Available from: <http://dannorth.net/introducing-bdd/>
- 45 Anon (n.d.) 'Program to an interface'. [online] Available from: <https://dzone.com/articles/programming-to-an-interface> (Accessed 14 September 2018)
- 46 JetBrains (n.d.) 'Kotlin Programming Language'. [online] Available from: <https://kotlinlang.org/docs/reference/interfaces.html> (Accessed 14 September 2018)
- 47 MongoDB (2016) 'NoSQL Databases Explained | MongoDB'. [online] Available from: <https://www.mongodb.com/nosql-explained> (Accessed 15 September 2018)
- 48 Google (2018) 'Distribution dashboard | Android developers'. [online] Available from: <https://developer.android.com/about/dashboards/> (Accessed 15 September 2018)
- 49 Fowler, Martin (2004) 'Inversion of Control Containers and the Dependency Injection pattern'. *Interface*, M, pp. 1–19. [online] Available from: <https://martinfowler.com/articles/injection.html> (Accessed 15 September 2018)
- 50 Anon (n.d.) 'Dagger'. [online] Available from: <https://google.github.io/dagger/> (Accessed 15 September 2018)
- 51 Google (2013) 'Application Fundamentals - Android Developers', <http://developer.android.com/guide/components/fundamentals.html>. *Http://Developer.Android.Com/Guide/Components/Fundamentals.Html*. [online] Available from: <https://developer.android.com/guide/components/fundamentals> (Accessed 15 September 2018)
- 52 Developers, Android (2014) 'Layouts | Android Developers'. [online] Available from: <https://developer.android.com/guide/topics/ui/declaring-layout> (Accessed 15 September 2018)
- 53 Developers, Android (2014) 'Styles and Themes'. , pp. 1–6. [online] Available from: <https://developer.android.com/guide/topics/ui/look-and-feel/themes> (Accessed 15 September 2018)
- 54 Google (2017) 'Material Design Icons'. *Material.io*. [online] Available from: <https://materialdesignicons.com/> (Accessed 16 September 2018)
- 55 Apache (n.d.) 'Apache License, Version 2.0'. [online] Available from: <http://www.apache.org/licenses/LICENSE-2.0> (Accessed 16 September 2018)
- 56 Anon (n.d.) 'Sealed Classes - Kotlin Programming Language'. [online] Available from: <https://kotlinlang.org/docs/reference/sealed-classes.html> (Accessed 15 September 2018)
- 57 JetBrains (n.d.) 'Kotlin Programming Language'. [online] Available from: <https://kotlinlang.org/docs/reference/extensions.html> (Accessed 16 September 2018)
- 58 Google (n.d.) 'Firebase UI for Auth'. [online] Available from: <https://github.com/firebase/FirebaseUI-Android/blob/master/auth/README.md> (Accessed 12 September 2018)
- 59 Anon (n.d.) 'Create a List with RecyclerView | Android Developers'. [online] Available from: <https://developer.android.com/guide/topics/ui/layout/recyclerview> (Accessed 15 September 2018)

- 60 Fowler, Martin (2011) *Domain-specific languages*, Addison-Wesley.
- 61 Fowler, Martin (n.d.) 'GivenWhenThen'. [online] Available from:
<https://martinfowler.com/bliki/GivenWhenThen.html> (Accessed 15 September 2018)
- 62 Mukundarajan, Haripriya, Hol, Felix Jan Hein, Castillo, Erica Araceli, Newby, Cooper
and Prakash, Manu (2017) 'Using mobile phones as acoustic sensors for high-
throughput mosquito surveillance'. *eLife*, 6. [online] Available from:
<https://elifesciences.org/articles/27854> (Accessed 16 September 2018)
- 63 Anon (n.d.) 'Data Binding Library | Android Developers'. [online] Available from:
<https://developer.android.com/topic/libraries/data-binding/> (Accessed 16 September
2018)
- 64 Google Inc.b (2018) 'Understand the Activity Lifecycle | Android Developers'. *18.05.18*,
p. 1. [online] Available from:
<https://developer.android.com/guide/components/activities/activity-lifecycle>
(Accessed 16 September 2018)
- 65 Mears, Chris (2013) 'Personas - The Beginner's Guide - The UX Review'. *The UX
Review*. [online] Available from: [https://theuxreview.co.uk/user-journeys-beginners-
guide/](https://theuxreview.co.uk/user-journeys-beginners-guide/) (Accessed 12 September 2018)

11 TABLE OF FIGURES

Figure 1 – High-level overview of survey capture and survey upload...	Error! Bookmark not defined.
Figure 2 – High-level overview of project architecture.....	Error! Bookmark not defined.
Figure 3 – MVVM architecture.....	28
Figure 4 – Class diagram for “authenticate and set preferences” user journey.....	30
Figure 5 – Class diagram for “show a list of surveys” user journey.....	32
Figure 6 – Class diagram for “user can configure app” user journey.....	33
Figure 7 – Class diagram for “add a patient survey” user journey.....	34
Figure 8 – Class diagram for “add a mosquito batch survey” user journey.....	36
Figure 9 – Class diagram for “add a mosquito detail survey” user journey.....	37
Figure 10 – Class diagram for “review surveys” user journey.....	39
Figure 11 – Class diagram for “automatically upload surveys”.....	40
Figure 12 – PT app data objects.....	41
Figure 13 – Cloud Firestore data model.....	42
Figure 14 – Choose primary ST.....	47
Figure 15 – Choose secondary ST.....	48
Figure 16 – No surveys recorded.....	48
Figure 17 – Drawer menu open.....	49
Figure 18 – Choose survey type.....	49
Figure 19 – Surveys activity displaying surveys.....	50
Figure 20 – Patient survey, view 1.....	52
Figure 21 – Patient survey, view 2.....	52
Figure 22 – Patient survey, view 3.....	53
Figure 23 – Mosquito Batch survey.....	54
Figure 24 – Mosquito detail survey.....	55
Figure 25 – Patient detail.....	56
Figure 26 – Mosquito Batch detail.....	56
Figure 27 – Mosquito detail.....	56
Figure 28 – Preferences view.....	57
Figure 29 – First-time use journey.....	95
Figure 30 – Mosquito survey journey.....	96
Figure 31 – Patient survey journey.....	97
Figure 32 – user collects both Patient and Mosquito surveys journey.....	97
Figure 33 – Mosquito Batch survey review	Figure 34 – Patient survey review.....98
Figure 35 – Mosquito survey review.....	99

APPENDIX A: USER JOURNEYS

A user journey is the experiences a user has when interacting with an application. They focus on what the user can see and do⁶⁵. Specifications related to the user journeys presented below are discussed in chapter 3.

First-time use

1. The first time a user opens PT app, they are requested to log in via Firebase's authentication user interface.
2. The user is then presented with questions and selectable answers regarding their intended use of the application, and are requested to allow PT app permission to access the device's location, camera, and storage.
3. The main Surveys screen is shown with a message encouraging the user to enter a survey.

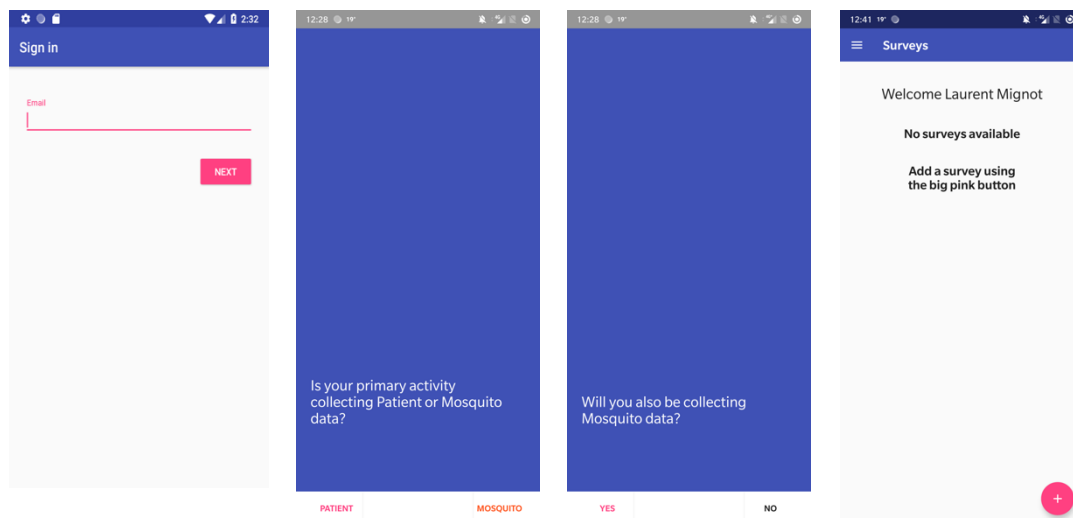


Figure 29 – First-time use journey

User only collects Mosquito surveys

1. The user selects “add survey” using the button provided on the main Surveys screen.
2. The user is presented with a form for Mosquito Batch data, which they complete and then select to save the survey.
3. The user is returned to the Surveys screen where they can select their newly recorded Mosquito Batch survey.
4. They are shown the survey review screen from which they can use the provided button to add a Mosquito Detail survey.
5. The user is presented with a form for Mosquito Detail data, including the option to take a photo of the mosquito. They save the survey and are returned to the Mosquito Batch review screen from step 3.
6. The user repeats steps 2–5 as often as required.

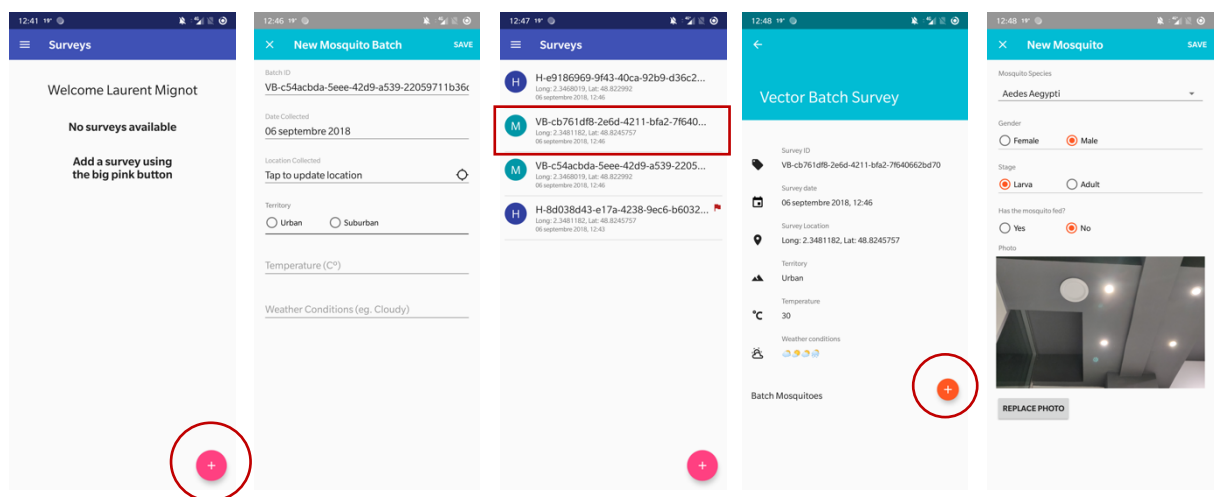


Figure 30 – Mosquito survey journey

User only collects Patient surveys

1. The user selects “add survey” using the button provided on the main Surveys screen.
2. The user is presented with a form for entering Patient data on three separate screens.
3. The user enters the survey data while proceeding between the three screens.
4. The user saves the survey and is returned to the Surveys screen from which they can repeat steps 1–3 as required.

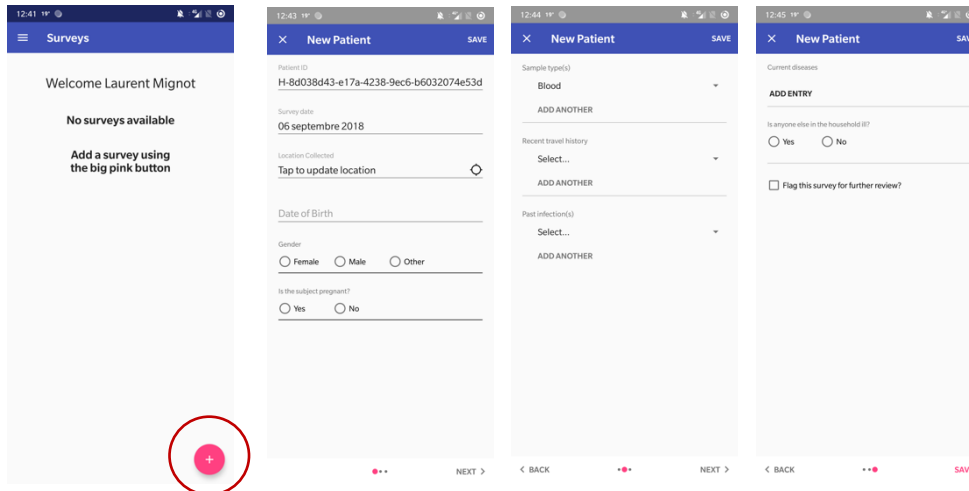


Figure 31 – Patient survey journey

User collects both Patient and Mosquito surveys

1. The user selects to add a survey using the button provided on the main Surveys screen.
2. The user is asked to select which type of survey they would like to add.
3. The user then follows either the Mosquito or Patient journeys
4. Steps 1–3 are repeated as required.

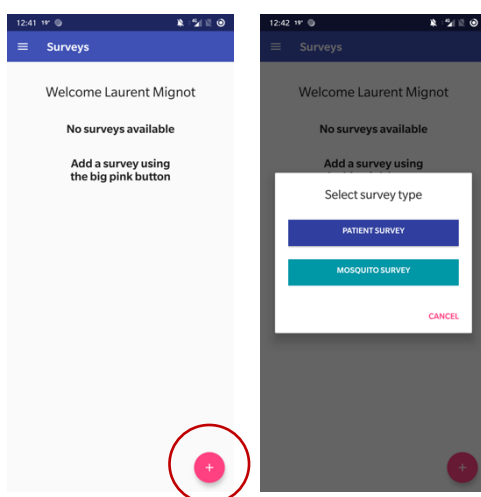


Figure 32 – user collects both Patient and Mosquito surveys journey

Survey Review (Patient, Mosquito Batch)

1. The user opens PT app
2. The main Surveys screen is shown
3. The user selects the survey to review from the list
4. The user is presented with the survey detail screen which displays the data for their selected survey
5. The user can return to the main Surveys screen and repeat steps 2–4 as required.

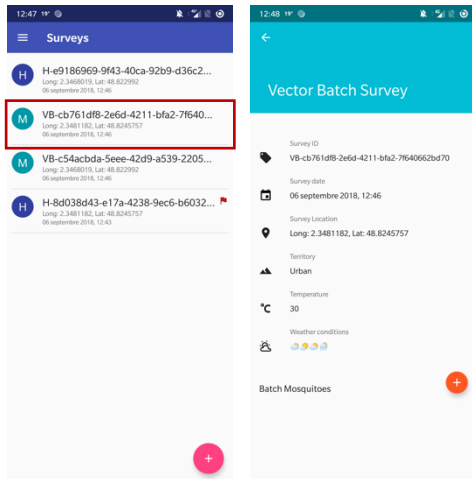


Figure 33 – Mosquito Batch survey review

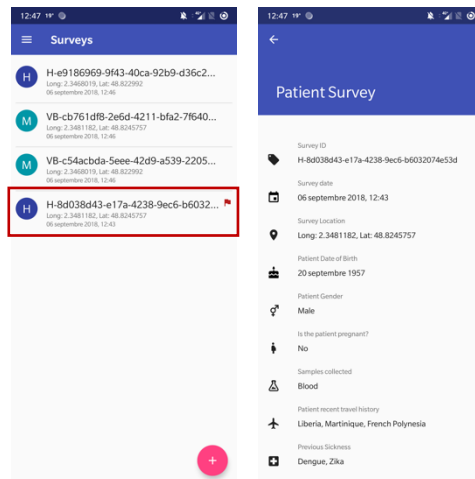


Figure 34 – Patient survey review

Survey Review (Mosquito detail)

1. The user opens PT app
2. The main surveys screen is shown
3. The user selects a Mosquito Batch survey from the list
4. The user is presented with the survey detail screen for their selected Mosquito Batch
5. The user selects their Mosquito Detail survey from the list
6. The user is presented with a survey detail screen for their selected Mosquito
7. User can return to Mosquito Batch survey and either return to Main Surveys screen and repeat steps 3–6 or select another Mosquito survey for review

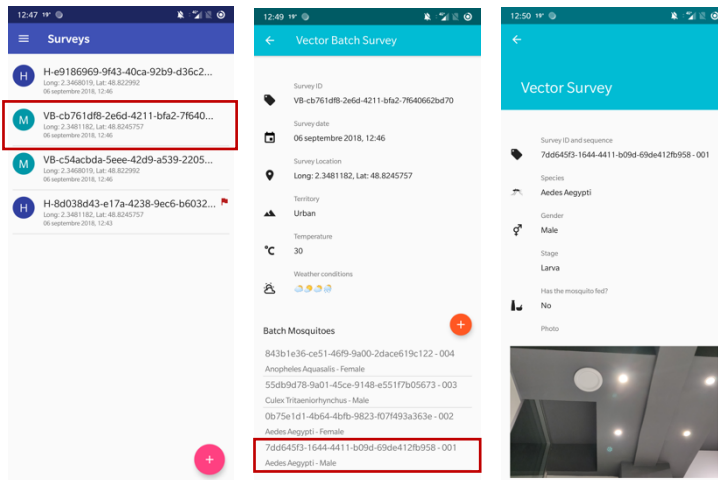


Figure 35 – Mosquito survey review