FoodSpartan

Final Report

Group M

Supervisor: Dr Nikolay Nikolaev

Project Team:

Robert Bechara | Project Leader & Vision

Robert Killick | Architecture & Prototyping

Lendl Munoz | Designer & Programmer

Temurhon Kamariddinov | Market Research & Programmer

Volkan Kunduru | Programmer & Tester

Danil Vorobjov | Tester

30 April 2018

ABSTRACT

We live in a globalised world that is fast moving and has become more interconnected than ever before. It's easy for us to sometimes forget the importance of healthy eating, which forms the basis for a well functioning body and mind. Healthy eating contributes to disease prevention, maintenance of a healthy weight and quality of life.

This project seeks to optimise the fit between technology and people by utilising the power of gamification to enhance people's quality of life and reward healthy eating. Gamification is widely known as "the use of game design elements in non-game contexts". [1] Using the constructs of game design we seek improvement in people's engagement, motivation, inducing a change their behaviour.

ACKNOWLEDGEMENTS

The project team would like to thank our supervisor, Dr Nikolay Nikolaev for his guidance, support and expertise throughout the evolution of this project.

The project team would also like to thank the Department of Computing at Goldsmiths, University of London, for the opportunity to work on this project.

The project team would also like thank all those who participated in providing data and feedback, through the participation of surveys and focus groups which has helped us shape and evolve the direction of this project.

FIGURES & TABLES

List of Figures:

1 INTRODUCTION

1.1 Motivation

We live in a fast moving world which in recent decades has become more interconnected thanks to the advancement of technology. It's quite easy for us to forget the importance of healthy eating, which forms the basis of a functioning body and mind, contributing to disease prevention and improvement to our quality of life. With many health apps & programmes that coexist on the marketplace, many are either too complex, or don't offer any motivational drives for users to consume healthy foods. Our goal is revolutionise the way we think about healthy eating using social trends and delivering users a simple, instant and rewarding connected experience.

1.2 Scope and Objectives

Food Spartan is an Android OS app where our objective is to use gamification as a platform to encourage & motivate healthy eating using social trends. The app would require users to take a photo of their meal which would then be uploaded and through image recognition, points instantly awarded based on how healthy the food items are within the image. The photos would then be displayed on their profile and shared on a timeline allowing scope for social interaction with other users through 'comments' and 'likes'. Users will be able to follow each other allowing for a connected experience. To reward users, they will be able to earn bonus points participating in timed events in a form of 'accolades' and will be able to spend points awarded in exchange for coupons & promotions with partners and affiliates. We hope in time to develop an ecosystem around our app and platform to allow scope for content moderation to keep our app safe and secure for users and the growth of functions to allow deeper rooted social experiences.

1.3 Intended Audience

This report is intended for all individuals participating and supervising the project and presumes some technical knowledge and assumes the reader has read the initial project proposal.

1.4 Coverage

This report includes seven chapters, excluding the introduction:

Chapter 2 - Literature Review: provides an brief overview on what was outputted in the project proposal and includes theoretical background and research into image recognition and deep learning and the android framework

Chapter 3 - Project Development Process: covers project development methodology providing an overview of the development model adopted for this project and source code management.

Chapter 4 - Analysis: requirements specification covering functionalities of the project and requirements analysis through use case interactions.

Chapter 5 - Design: describes the architectural design and artefacts of our app.

Chapter 6 - Implementation: covers how our app was implemented.

Chapter 7 - Evaluation & QA Testing: formative evaluation and functional testing.

Chapter 8 - Project Conclusions: provides reflective conclusions of the project and proposed future work.

2 LITERATURE REVIEW

In this section, the background to the project is discussed where we review the foundations and vision of our project. We also consider the technologies and theories used to develop our application including the basic concepts of the Android framework and achieving image recognition through neural networks.

2.1 Background

This report builds on our project proposal where we laid the foundations and vision behind our project - establishing our stakeholders to conducting market research with the analysis of the Octalysis gamification framework [2] as well as social and marketplace trends. We analysed various emerging technologies around machine learning and image recognition and conducted surveys to better understand our users. From this we were able to establish credible user requirements for our project and the production of a functional requirements specification outlining desirable and essential functionalities around our app. We were then able to construct UML artefacts of our of system - consisting of static use case and class diagrams and dynamic sequential and activity diagrams. These forge blueprints around how app works and will be referred to in this report. Some prototyping was also conducted allowing us to establish an early look into UI concepts around our app.

2.2 The Android Framework

Android is a mobile operating system that is based on a modified version of Linux and is widely used in the market today - found on many mobile devices. Beyond a mobile OS, Android also provides a software stack which includes middleware and a number of applications.

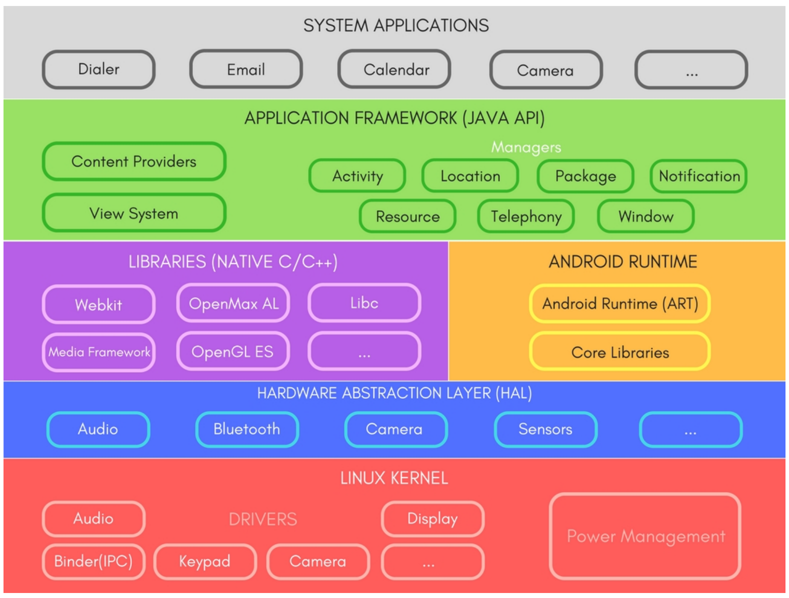


Figure 1: overview of the Android architecture [3]

Figure ** shows the architecture of the system. Android OS is based on Linux kernel 2.6 and is located at the bottom of the stack and acts as a bridge between software and hardware resources.

Above the Linux kernel several libraries are used. These libraries are written in multiple languages and serve as the basis of where the actual framework is built. The Android Runtime (ART) handles the process management. This has been built to accommodate low memory constraints of Android devices in order to optimise the end user experience.

The next level represents the application framework. This is where system services and developer APIs are defined. The system services are used to access low-level functionalities while the APIs are used to wrap up these functionalities into components to make them reusable.

All installed applications are placed in the application layer. All applications are written by using developer APIs and reusable components provided by other applications.

Java API: The cardinal necessity of Android development is the need to program in the Java programming language. Java is a well known object-orientated programming language developed by Sun Microsystems.

Extensible Markup Language (XML): Used for simple encoding of documents in the creation of Android applications. This programming language is used to specify the content of necessary files such as ***Android Manifest 3.1*** or the layout of various screens known as ***Activities***. This is discussed further in Section 6.

Android Software Development Kit (SDK): Developing for Android OS requires working with the Android SDK which includes different components used to build an Android application and corresponds well with MVP design patterns (outlined in section 5.1.1). The Android SDK includes everything needed to develop, test and debug an Android application. It includes various components that are paramount to this. This includes Android APIs providing access to the Android stack, development tools to compile, run and debug application, emulation to visually run applications, documentation and sample code.

For our project we have chosen to use *Android Studio (IDE)* which includes a virtual machine manager and the Android SDK along with its features [4]. Its usage is outlined in our implementation phase (see Section 6).

2.2.1 Activities

An Android application is made up of different types of components however a fundamental component for any application are *Activities*. This represents the visible screen where a user interacts with the application and has different states.

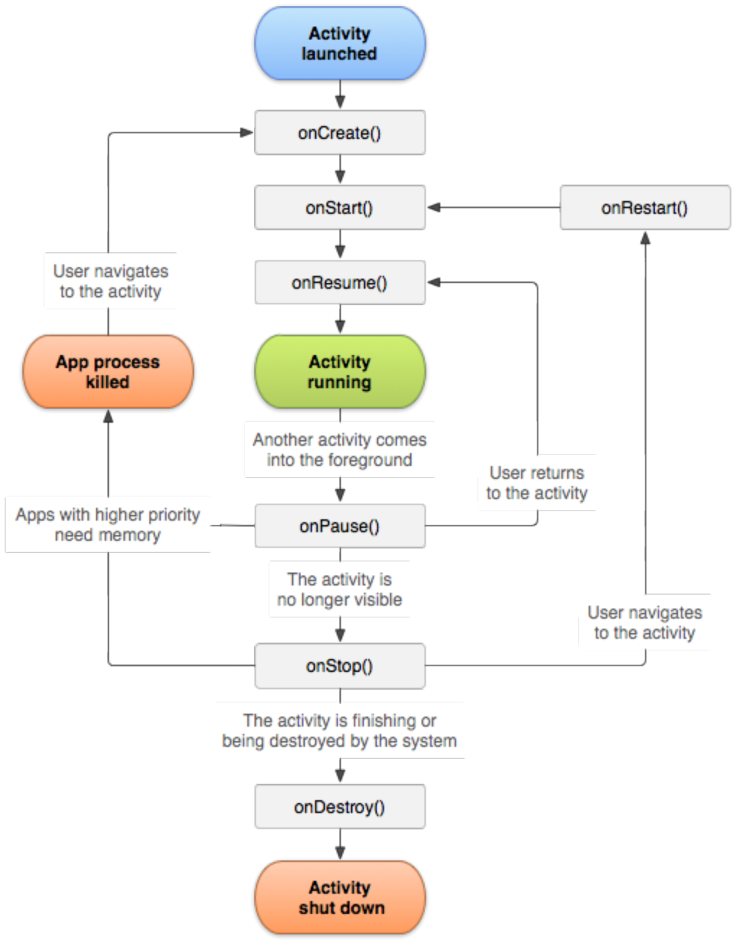

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 2: Activity lifecycle overview [5]

This shows the lifecycle of an activity which is inclusive of the various stages it goes through from start to finish. The Android API documentation states "the activity traverses between states at different times during its lifecycle. The correct manipulation of these states is a key component for good design and development of an application." [5] These methods can be overridden thus changing how the application reacts under certain condition. For example, when an activity is created, the method *onCreate()* is called.

Activities contain fragments - these are panels capable of containing different Views. These are the components users use to traverse the application.

2.2.2 Views

In Android, a view represents a widget that are shown on the screen. Examples include radio buttons, text boxes, labels etc. Views are used together with a combination of event listeners in order to create responsive interactive layouts for users.

2.2.3 Intents

Intents are used to chain multiple activities together within an application - interconnecting different activities and services together.

2.2.4 Services

Similar to activities in operation - however a service usually runs in the background, performing long term tasks. Services don't require graphical interfaces and may be started to perform a task or at application startup.

The implementation process we establish allows us to put together all the encompassing features of the android framework needed to develop our project. This includes - designing and implementing the necessary *Activities* and their respective *Views* in order to form the application panels. *Intents* and *Services* need to be implemented in order to put together the *Activities* which form the application. Logic implementation will take place using the standard *Java API*.

2.3 Image Recognition & Deep Learning

In recent years, there have been profound advances in deep learning making tasks such as image recognition possible. Deep Learning is a subset of machine learning algorithms designed to recognise patterns but typically requires a large number of data. In applying the image recognition of food to our project - the team took a high-level approach in exploring various techniques and theoretical options building a deep neural network that can recognise images.

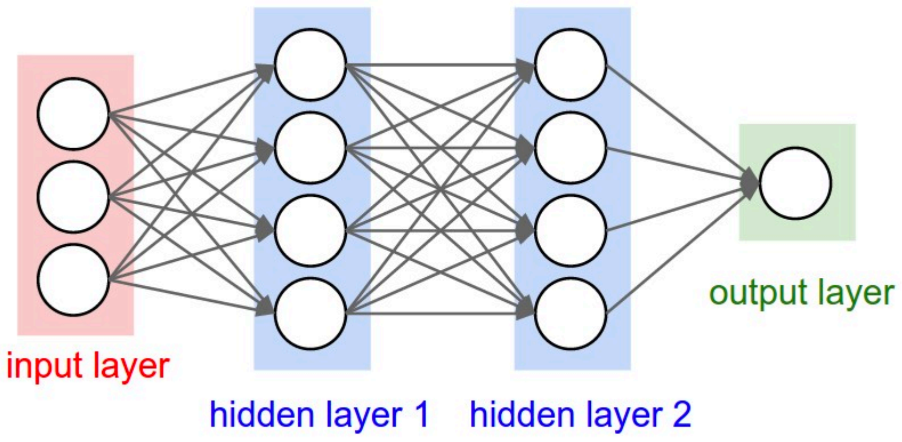


Figure 3: Neural Network representation [6]

Deep learning excels in recognising objects in images as its implemented using three or more layers of artificial neural networks where each layer is responsible for extracting one or more features of the image.

A neural network is a computational model which works in a similar way to the neurones in a human brain. Each neutron takes an input and performs some operations then passing the output to the next neurone.

Our team explored using Python and TensorFlow[7] - an open source deep learning framework created by Google.

Machine learning can be used to classify images - performing computations to convert images to numbers.

3 PROJECT DEVELOPMENT PROCESS

3.1 Methodologies

The project development process included a set number of stages to our app's evolutionary lifecycle [8]. First we began with the requirements phase which outlined the goals of what our application will be capable of doing. Next, was the design phase which provided a blueprint to how our app was to be created. The implementation phase is where programmers implement functionalities outlined by the design artefacts of our system. Finally comes evaluation and testing phase which allowed for iterations of our app to be tested and evaluated, whilst also addressing bugs to ensure the fulfilment of project requirements.

There are various development models that currently exist in software engineering [9] that consist of an abstract representation of a software process through various conventions. Our project team considered various models and approaches to software development.

The Waterfall Model [10] offers a linear-sequential lifecycle model, where each phase must be completed fully before the next phase can begin. At the end of each a phase a review takes place that determines if a the project can progress to the next stage. This can work well for smaller projects where requirements and clearly understood. However this model is unsuitable for long and ongoing projects where requirements are subject to changing and it cannot produce a working application until later in the development lifecycle. Lastly, when the app reaches the testing phase, this model makes it quite difficult to go back and change something that might have been flawed in the concept phase.

The Incremental Model [11] approaches software development by dividing requirements into various builds. This consists of multiple development cycles divided into smaller more manageable modules, where each module goes through the requirements, design, implementation and testing phases with a working version of the application produced in the first module. Subsequent modules adds functionality to the previous release. This process continues until all project requirements are fulfilled. While this model is flexible and produces a working application quickly, it does however require good planning and design of the entire system from the get go before it can be implemented incrementally.

The Spiral Model [12] is quite similar to the incremental model, with the inclusion of risk analysis. It includes, planning, risk analysis, engineering & evaluation. Software development process is presented in a spiral based on iterations. Subsequent spirals build on the baseline spiral. Requirements are gathered during the planning phase, then a process is undertaken to identify risks with alternate solutions proposed during the risk analysis phase. A prototype is produced at the end of the risk analysis phase. This model works well for larger projects where the costs of development is an important factor, with avoidance of risks playing an important role. However risk analysis requires great expertise in that parameter and project success highly dependent on the outcome of risk analysis.

3.2 Agile Model

The Agile Model [13] is another variant of the incremental model. Software development process is done in incremental, rapid cycles, with small incremental releases with each release building on previous functionality. Each release is tested for quality assurance. This model has several advantages including frequent delivery of working software and it allows scope for changes to the requirements later in the lifecycle. This is characterised by the model's flexibility for regular adaptation to changing circumstances and continued technical evolution of a project

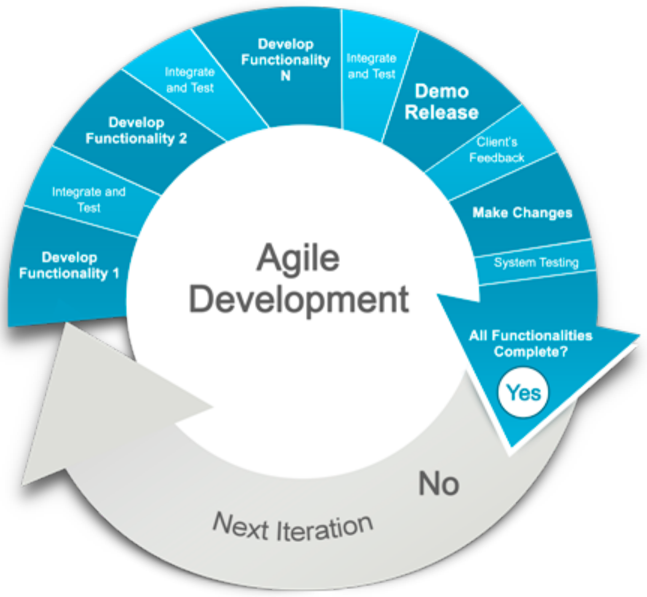


Figure 4: The Agile Model

The Agile Model was chosen by the project team as the best approach to the development process of this project. One of the most important reasons for this is because this model allows for changes to project requirements giving scope to respond to formative evaluation with our users and provides frequent incremental releases and iterations of our Android application. Furthermore, with frequent 'sprint planning', it allows our team to develop our application in increments with each one building and improving on previous release. In each increment, there will some changes to the requirements and improvements to the design leading up to new software development cycles. Furthermore our application will need to be tested on multiple devices which are running different versions of Android OS. This creates a need for a software development methodology which can handle change and provide a way to meet the rapid development requirements.

The phases of the agile model are followed in the development process of project, with the following taking place:

Requirements Specification: Description of the behaviour of the system to be developed and fulfilled.
Design: Blueprints of the system's architecture and its important components
Implementation: Design artefacts and blueprints implemented
Evaluation & Testing: Test cases to ensure the system has met its specification and evaluation of successes of previous phases with modifications made to them when applicable.

3.3 Source Code Management

An important part of the development process of our application is how the team chooses to manage our application's source code. In recent years, advances in version control systems such as git have provided powerful tools allowing for concurrent development and traceability.

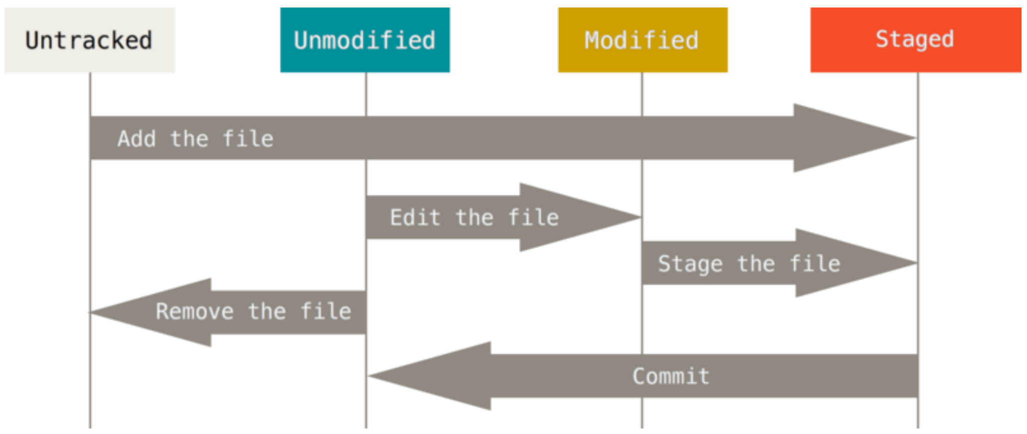


Figure 5: File lifecycle in GIT SCM [14]

The project team uses gitlab [15] - a source code repository which provides storage, management and tracking of changes made to our code throughout implementation of our system.

4 ANALYSIS

We have already established a set of functional requirements derived from our project proposal for our application to fulfil. Following some formative evaluation with our users via various focus groups - we made some refinements and additions to our requirements.

4.1 Specifications

Our system requirements are divided between functional and non-functional requirements. We look to various conventions defined in software engineering to aid us in the analysis of the requirements of the system under development. [16]

4.1.1 Functional Requirements

Functional requirements defines the functionality and components of our system.

Requirement ID	Requirement Name	Statement	Must/Want
FR001	Register Username, Password. Logging In.	The user shall be able to login to use, work and access information. For example, checking out how many points they have earned on their account. The user shall be able to register before storing any sort of information.	Must
FR002	Edit Profile Info	They user shall be able to edit their profile information once they have registered. They should be able to edit are their profile information & picture to identify themselves.	Must
FR003	Taking a Photo	The user shall have the ability to take a picture within the app & using image recognition they will be able to earn points.	Must
FR004	Uploading Photos	The user shall be able to upload photos of their food after registering. Once they upload a photo. A recognition algorithm should be able to identify what the food it is.	Must
FR005	Edit Photo Data	The user should be able to edit the data of the photo taken. This will include the location and the contents of the photo.	Want

Requirement ID	Requirement Name	Statement	Must/Want
FR006	Sharing Photo	The user should be able to share the a photo taken to various external platforms.	Want
FR007	Gain Points	The user should gain points allocated based on timed events and the contents of the photo uploaded.	Want
FR008	Earn Accolades	They users should be able to earn various timed and non-timed accolades/awards based on their activity.	Want
FR009	Spending Points	Once the user has sufficient points the user should be able to spend their reward points they have earned as virtual currency on various virtual/non-virtual items from the store!	Want
FR010	Alter/Delete Comments	The user should be able the change and delete their comments.	Want
FR011	Comment & Like Photo	The user shall be able to comment on like other user's photos on the timeline.	Must
FR012	Manage Photos	The users should be able to manage photos stored on the database. The user can view and rearrange the photos in ascending/descending order.	Want
FR013	Search Photos	The user shall be able to access other users photos by searching for their username or the photo attributes such as the location, name or the type of food that's included in a photo.	Must
FR014	Filter Searching	The user should be able to filter photo search based on location, points achieved, type of food included with a photo.	Want
FR015	Present Photos	The user shall be able to present photos stored on the database, to their timeline or profile space.	Must
FR016	Logging out	The user shall be able to sign out once they're done with the application or if they are inactive. This will insure efficient security and robust inactivity.	Must

Table 1: Functional Specification

4.1.2 Non-functional Requirements

Non-functional requirements support our functional requirements - describes how the system will fulfil them. For example covering areas important to our users such as usability, reliability, security, performance and costs. These are important requirements to our users which was indicated by our focus groups [15].

Requirement ID	Description	Statement
NFR001	The system will be able to run on all Android OS devices.	The system will be developed to run on all Android OS devices such as mobiles/tablets.
NFR002	The system must be designed to be able to evolve with new functionality and operations.	The system must be designed in a way that the team can add new functionality and features to the source code.
NFR003	The graphical user interface must intuitive and dynamic to fit all screen sizes.	The layouts and GUI experience of our application should be universal to all screen sizes. The UI experience should be simple and consist for our users across.
NFR004	The system must lack of bugs and inform the user of any wrong operations encountered.	With each incremental release the system must be thoroughly tested with error message logs implemented to inform a user of any errors in operations encountered.
NFR005	The system will have a fast response time.	The system should run operations quickly. Operations that require longer execution times will present the user a 'loading icon' until the operation is fulfilled.

Table 2: Non-functional Requirements

4.2 Use case diagrams

Use case diagrams are static behaviour diagrams from which are built on in the design process denoted in Section 5.

In our project, the actors are the primary users from our stakeholders defined in our proposal which describe a set of operations and interactions that users can perform with our system.

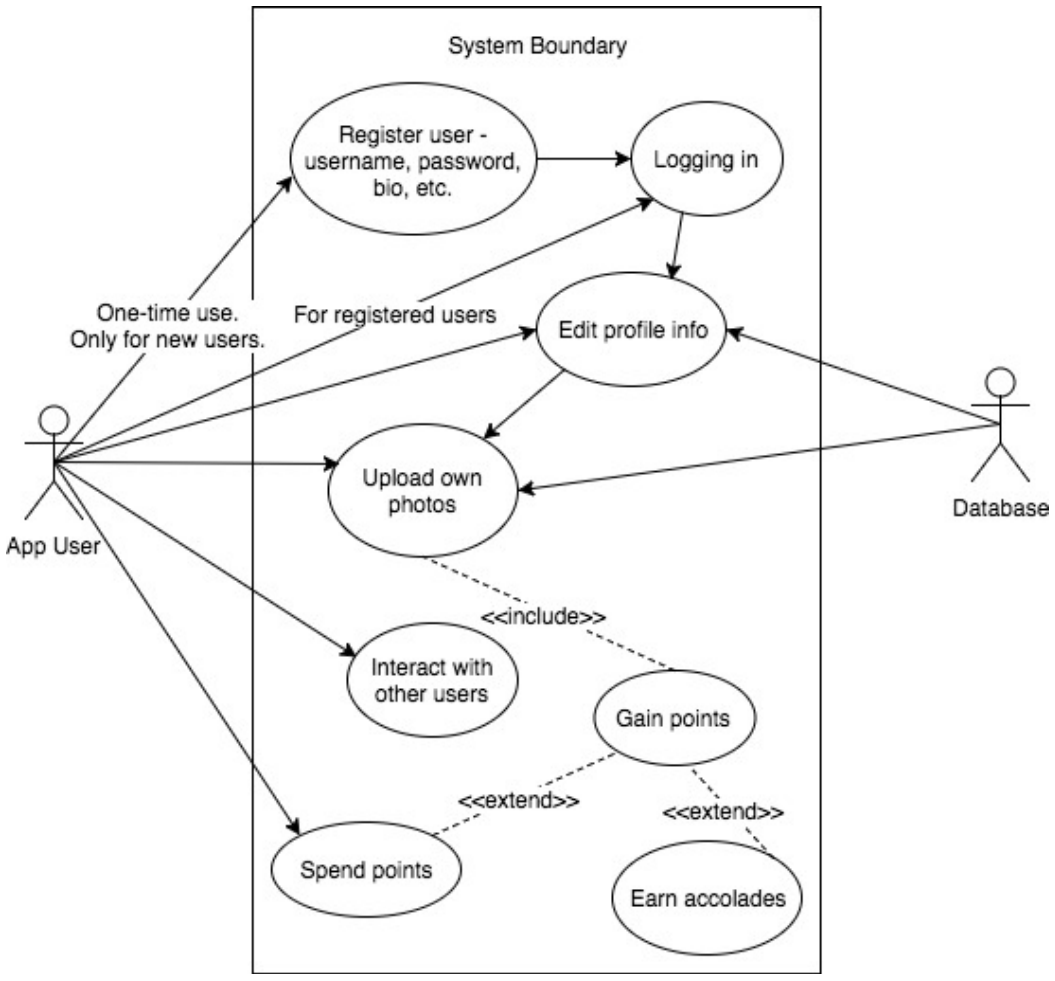


Figure 6: Profile use case diagram

See **Appendix A** for other use case interactions with our system.

4.2.1 Use Case Scenarios

We are able to describe the use case interactions with our system such as the ones in Figure 3 in more detail in form of use case scenarios. Every use case represents a user scenario and provides information on the name, the actors, the components, a description, a abstract basic flow, a precondition and its trigger [17]. This enables traceability of the requirement and helps us to evaluate its outcome.

User Case Name	Take a photo.
Context	Taking a photo to upload to the server.
Primary Actor	User
Precondition	The user has to be logged in in order to access and upload it to the right server.
Success Post Condition	The photo will be uploaded there will be a dialogue message telling the user that the photo is uploaded.
Trigger	The user wants to upload a new photo
Main Success Scenario	1.The user directs to the upload navigation bar 2.The user pressed the upload photo button 3.Android will display the photo storage of the user's current device 4.The user will choose one of the photos they want to select and hits send 5.The storage option will close and the process of uploading a photo will occur. a.If the photo upload is successful, a dialogue will appear and it says "Upload successful". b.If the photo upload is not successful, a dialogue will appear with an error message and what caused it.
Exceptions	The user taking photos of an image of a food. A photo of a food being covered by other objects. The photo of the food is out of focus.

Table 3: photo use case scenario

See **Appendix B** for other use case scenarios

5 DESIGN

UML ‘Unified Modelling Language’ allows us to forge visual designs to construct and document artefacts of our software system. [17] UML makes these artefacts scaleable, secure and robust in execution. We build on these designs from our project proposal taking our most up-to-date functional requirements of our users to map out interactions with our system. This is also aided by the agile methodology outlined in Section 3.

In this section we discuss the design of our application and its database with the purpose of meeting the requirements set out in Section 4. We started by addressing the application’s architectural design followed by its design models providing an overview of the final design of our software.

5.1 Architectural Design

In thinking about the architectural design of our project, it needed to be simple and easy to understand so that it aids in the development of a reusable and commonly understandable application. Furthermore, the architecture required a minimum dimension of modularity to ensure that this application can be developed in increments, that can expanded and refined in each of its interactions - allowing for an agile approach project development process.

5.1.1 Model-View-Presenter

The basic architecture behind this project is the Model-View-Presenter (MVP) design pattern. This model is responsible for implementing the logic of the application. It contains the code required for data interaction, database queries and other algorithmic functionality including data structures.

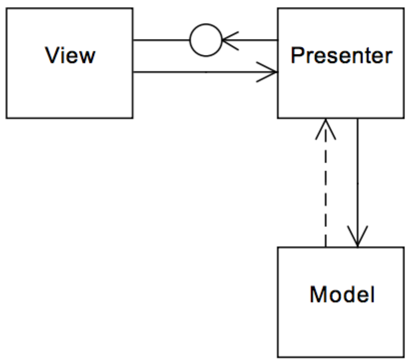


Figure 7: Model-View-Presenter design pattern

Developing for Android OS requires working with the Android SDK which includes different components used to build an Android application (see Section 6). These form a structure that matches well to MVP design pattern.

The ***View*** component is defined and represented by XML layout files which are used by activities and various other objects that are displayed inside an activity. They do not contain any logic and all user input are forwarded to activities. This component renders the user interface.

The ***Presenter*** component contains the activities, handling the user input, switches between the screens, communicates with the model and updates the view. *Adapters* are used by activities to update lists and *Comparators* are used to sort them.

The ***Model*** component contains data structures, database access and service functionality encapsulated in modules. The *DataTransferObject* is used to transport data from the *Database* to the presenter and vice versa. The *SyncService* connects with a server to synchronise the local database with a database on a server. The database encapsulates *SQLite*, to provide access to the database.

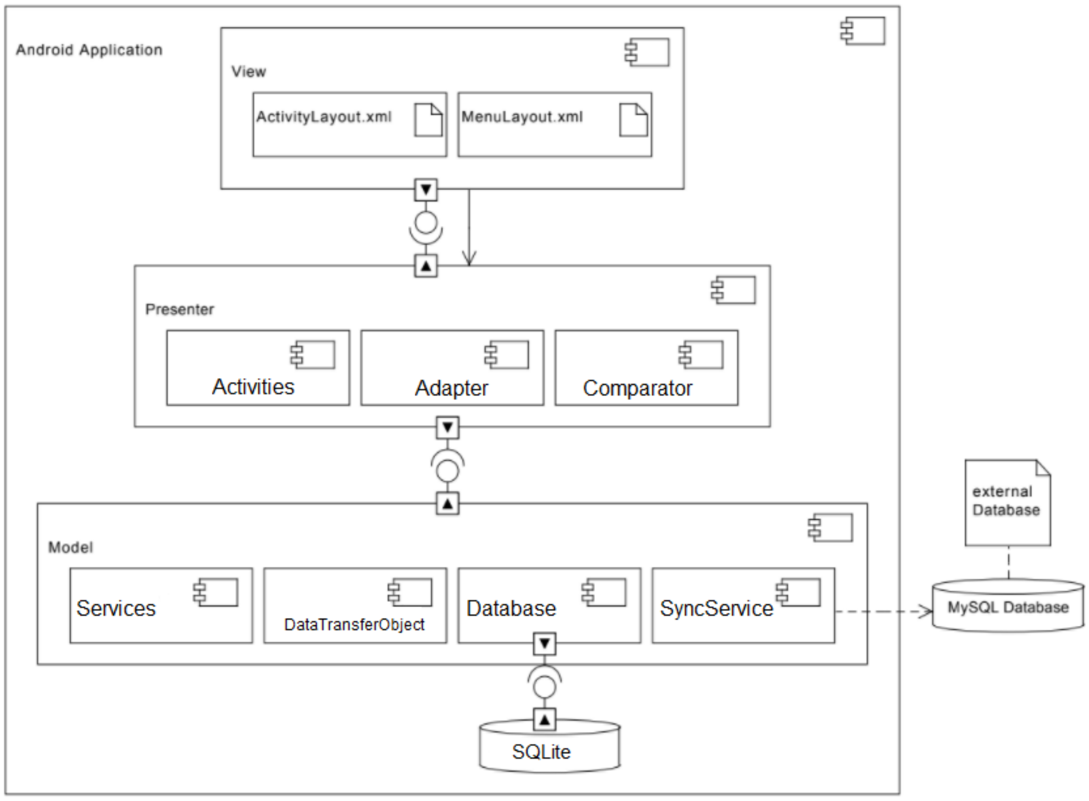


Figure 8: Architectural structure of components

Figure 6 shows how these components inside the MVP pattern will coexist with components added to existing iterations and releases of our application.

5.2 Design Models

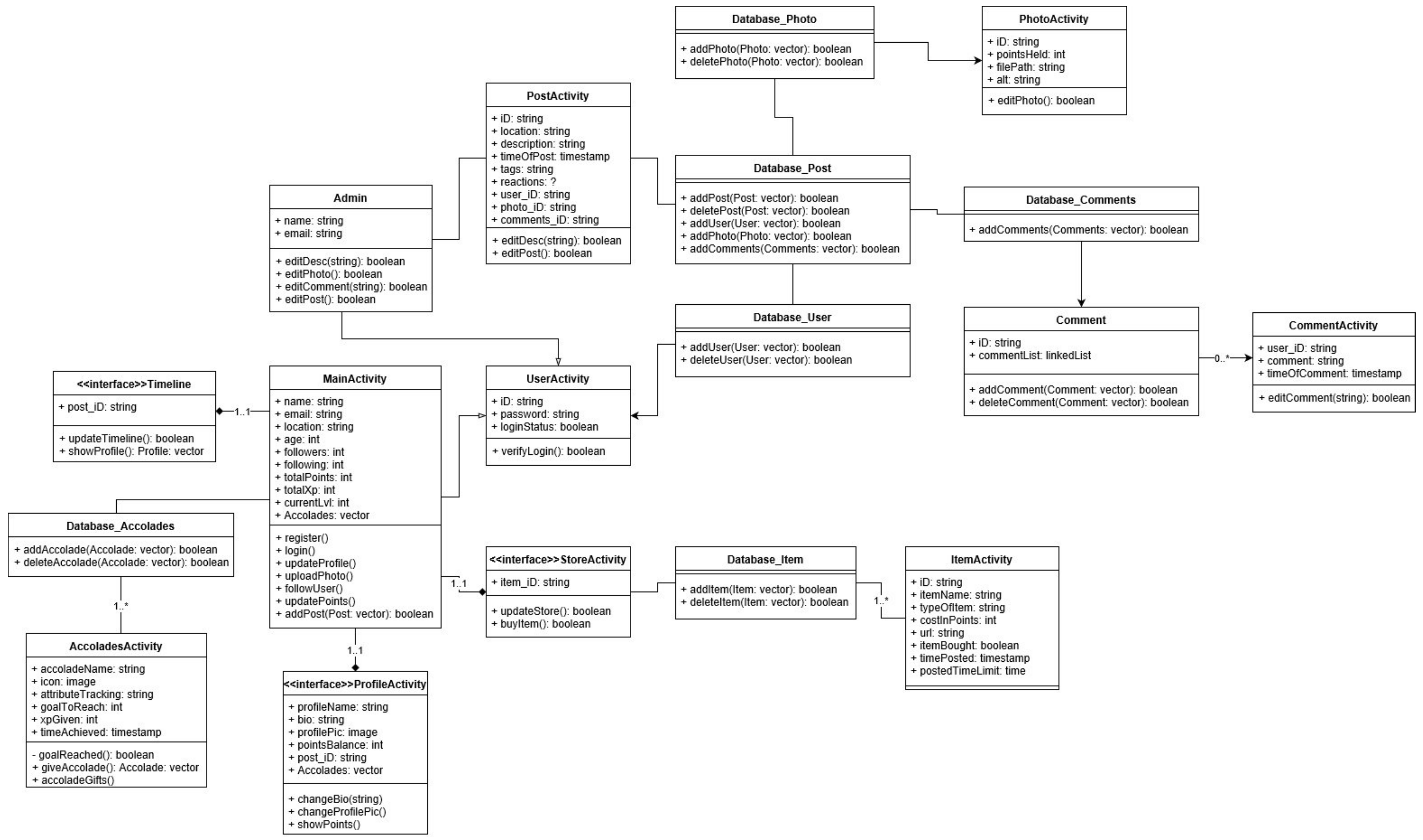
With our architectural design established, we can begin to describe the designs around the structure of our system in the lead up to implementation.

5.2.1 Class Diagram

The class diagram allows us to statically illustrate member data and member functions that make up the structure of the system. The present the main classes of the system and its operations. The diagram shows the relationships and dependencies between the classes. For example if we want to display MainActivity.class requires the precondition the user has logged in. Each class which has the suffix Activity enumerates a screen of the application.

MainActivity: This activity is the main home screen of the application. From here a user can visit their bespoke timeline, access the StoreActivity, ProfileActivity etc and their respective member functions.

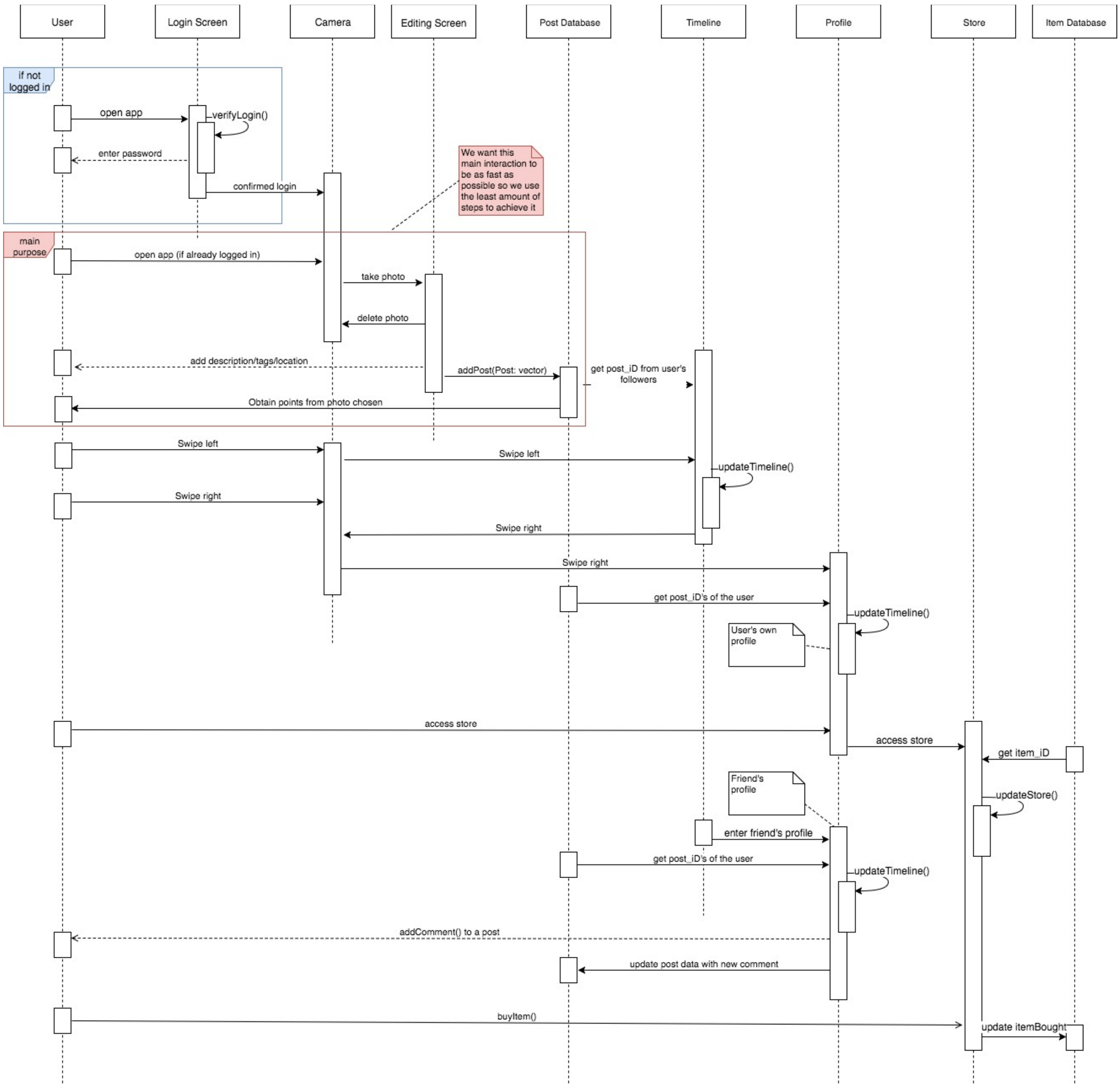
Figure 9: Class diagram shown on the next page.


Database_Photo
+ addPhoto(Photo: vector): boolean
+ deletePhoto(Photo: vector): boolean
PhotoActivity
+ iD: string
+ pointsHeld: int
+ filePath: string
+ alt: string
+ editPhoto(): boolean
PostActivity
+ iD: string
+ location: string
+ description: string
+ timeOfPost: timestamp
+ tags: string
+ reactions: ?
+ user_iD: string
+ photo_iD: string
+ comments_iD: string
+ editDesc(string): boolean
+ editPost(): boolean
Database_Post
+ addPost(Post: vector): boolean
+ deletePost(Post: vector): boolean
+ addUser(User: vector): boolean
+ addPhoto(Photo: vector): boolean
+ addComments(Comments: vector): boolean
Database_Comments
+ addComments(Comments: vector): boolean
Admin
+ name: string
+ email: string
+ editDesc(string): boolean
+ editPhoto(): boolean
+ editComment(string): boolean
+ editPost(): boolean
Database_User
+ addUser(User: vector): boolean
+ deleteUser(User: vector): boolean
Comment
+ iD: string
+ commentList: linkedList
+ addComment(Comment: vector): boolean
+ deleteComment(Comment: vector): boolean
0..*
CommentActivity
+ user_iD: string
+ comment: string
+ timeOfComment: timestamp
+ editComment(string): boolean
<<interface>>Timeline
+ post_iD: string
+ updateTimeline(): boolean
+ showProfile(): Profile: vector
1..1
MainActivity
+ name: string
+ email: string
+ location: string
+ age: int
+ followers: int
+ following: int
+ totalPoints: int
+ totalXp: int
+ currentLvl: int
+ Accolades: vector
+ register()
+ login()
+ updateProfile()
+ uploadPhoto()
+ followUser()
+ updatePoints()
+ addPost(Post: vector): boolean
UserActivity
+ iD: string
+ password: string
+ loginStatus: boolean
+ verifyLogin(): boolean
Database_Accolades
+ addAccolade(Accolade: vector): boolean
+ deleteAccolade(Accolade: vector): boolean
1..*
AccoladesActivity
+ accoladeName: string
+ icon: image
+ attributeTracking: string
+ goalToReach: int
+ xpGiven: int
+ timeAchieved: timestamp
- goalReached(): boolean
+ giveAccolade(): Accolade: vector
+ accoladeGifts()
1..1
<<interface>>StoreActivity
+ item_iD: string
+ updateStore(): boolean
+ buyItem(): boolean
Database_Item
+ addItem(Item: vector): boolean
+ deleteItem(Item: vector): boolean
1..*
ItemActivity
+ iD: string
+ itemName: string
+ typeOfItem: string
+ costInPoints: int
+ url: string
+ itemBought: boolean
+ timePosted: timestamp
+ postedTimeLimit: time
1..1
<<interface>>ProfileActivity
+ profileName: string
+ bio: string
+ profilePic: image
+ pointsBalance: int
+ post_iD: string
+ Accolades: vector
+ changeBio(string)
+ changeProfilePic()
+ showPoints()

5.2.2 Sequence Diagram

A sequence diagram represents the behaviour and interactions of the system by showing the interactions between objects in sequential order. Sequence diagrams dynamically focus on the 'lifelines' of an object and their communication in performing a function before a lifeline ends. We look at interactions around if the user isn't logged in and the main activities.

Figure 10: Sequence diagram encompassing the main dynamic interactions of the system



5.3 Database Design

Below shows the structure of the database implemented for this project. The table *users* is used to store the users. The table *posts* is used to store all posts that been created by *users*. The table *comments* is matched to *posts* and a *photo* table to store photos in JSON and points.

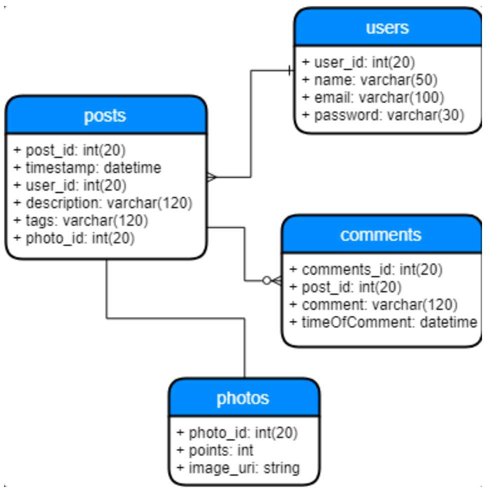


Figure 11: simplified entity-relation diagram

All the tables in the database contain a primary key. The lines in the diagram represent a foreign key relation, which means the foreign key of one table points to a primary key in another table. The use of the foreign keys is also important in our database design as it allows operations like insertion and deletion to be more efficient. The only tables that will have foreign keys are *posts* and *comments*.

All comments for all posts will be stored into one table. When a comment is made for a post, it will have its own *comments_id* as a primary key and will have the *post_id* as its foreign key. It is then inserted into the comments table with the date-time recorded as well. This way all comments with the same *post_id* will be collected and can also be ordered by the time when the comment was made. If a post was to be deleted, all comments linked to the post through the foreign key will be deleted along with it without the need of extra database commands.

users		
Attributes	**Datatype**	**Description**
user_id (PRIMARY KEY)	INT	The unique id for the user.
name	VARCHAR	The name of the user.
email	VARCHAR	The unique email of the user.
password	VARCHAR	The password of the user.

Table 4: users database table attributes

posts		
Attributes	**Datatype**	**Description**
post_id (PRIMARY KEY)	INT	The unique id for the post.
timestamp	DATETIME	The time when the post was made.
user_id (FOREIGN KEY)	INT	The user's id for the post.
description	VARCHAR	The description provided for the post.
tags	VARCHAR	Tags related to the post.
photo_id (FOREIGN KEY)	INT	The id for the photo for the post.

Table 5: posts database table attributes

See **Appendix C** for other database table attributes.

6 IMPLEMENTATION

This section outlines the implementation process that was undertaken by the project team - utilising the android framework and the development and design methodologies discussed in previous sections to forge a functional application. We also discuss unforeseen problems the we encountered during development and some of the solutions to them.

6.1 Approach

To develop our application an iterative approach is chosen as discussed in section 3.2. Each iteration of our development cycle covers new components with its outcome forming a new increment of our application.

Iteration 1: front-end development undertaken with the setup and deployment of our Android project and the development environment including tools provided by the Android SDK.

Iteration 2: design and building of the graphical user interface with a basic GUI developed in this increment that will be expanded.

Iteration 3: development and deployment of the backend including connection to a local database

Iteration 4*: integration of external services - outcome leading to many core functionalities, external databases and APIs around image recognition developed and deployed.

6.1.1 Basic vs core Implementation

Given time constraints and dependencies around implementation - the project team took a conscious decision to divide the development of our application into two phases. We prioritised basic development where we looked to implement essential functionality vs core development* which involved extending the application's functionality. This was denoted in our functional specification in Section 4.1.1.

6.2 Frontend development

The components that are part of the android framework used for development are outlined in Section 2.2.

6.2.1 Android Studio Components

Android Manifest:

One of the core files of every Android project includes ***AndroidManifest.xml*** which is located in the root directory. This contains the code required to run the application. These include ***Activities*** which enumerates all the basic screens - discussed in Section 2.2.1. ***Permissions*** which grant the application specific permissions such as accessing the internet. ***API level*** which states the minimum API level of the device targeted and ***Linked libraries*** which includes a list of libraries needed for the application to function.

Gradle:

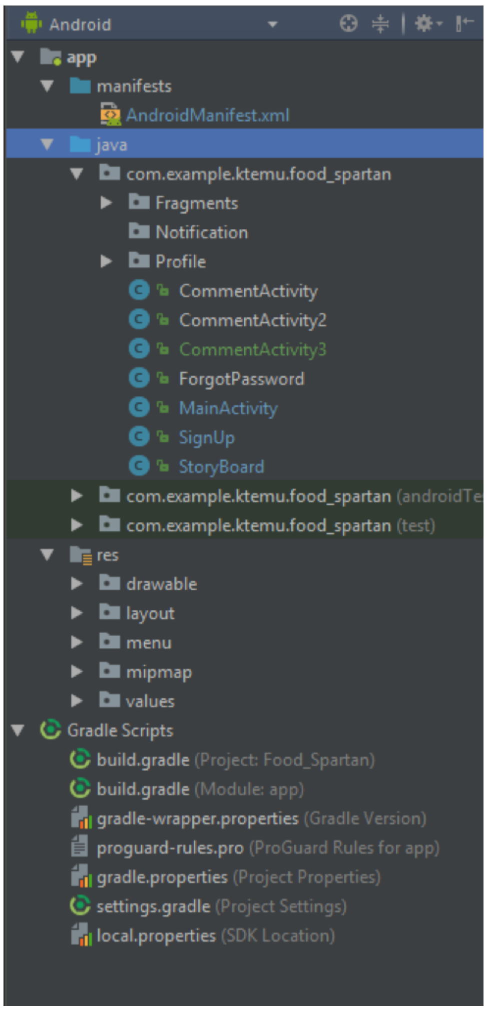
Known as ***build.gradle*** is another important component of an Android project. This a automated Java Virtual Machine (JVM) build system that takes all source files from the project to create one compressed APK file.

Project Structure:

Android follows a specific folder hierarchy to maintain its file system. The figure below shows structure of our project.

food_spartan\src\ : This is the directory for java files

food_spartan\res\ : This is the directory for resources like images and .XML files. It includes subdirectories discussed below.



```
Android
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ com.example.ktemu.food_spartan
      ▶ Fragments
        Notification
      ▶ Profile
        CommentActivity
        CommentActivity2
        CommentActivity3
        ForgotPassword
        MainActivity
        SignUp
        StoryBoard
    ▶ com.example.ktemu.food_spartan (androidTe
    ▶ com.example.ktemu.food_spartan (test)
  ▼ res
    ▶ drawable
    ▶ layout
    ▶ menu
    ▶ mipmap
    ▶ values
▼ Gradle Scripts
    build.gradle (Project: Food_Spartan)
    build.gradle (Module: app)
    gradle-wrapper.properties (Gradle Version)
    proguard-rules.pro (ProGuard Rules for app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

Figure 12: Food Spartan project structure

Resources:

res/layout: This is the directory that stores all XML files defining the layouts of a screen res/menu: This is the directory that stores layout files for the menu bar

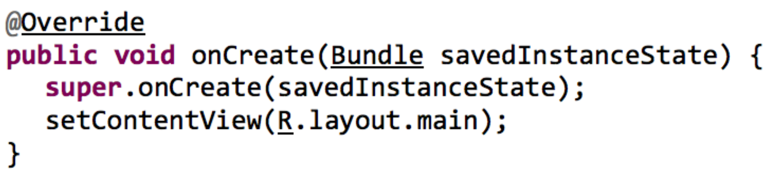
res/values: This includes resources for strings, dimensions and styles in the XML files.

6.3 Graphical User Interface

Providing a GUI to allow a user to traverse our application. Android applications are highly dependent on user friendly interfaces to provide a good experience for our users. To create a user friendly interface this project follows the Android User Interface Guidelines [18]

Extensible Markup Language (XML):

Android layout design is written in XML. The user interface logic code is written in ***Activities*** - this is used to display UI which contains ***widgets*** such as buttons, labels, text boxes etc. XML is loaded by calling the ***setContentView()*** method in the activity class which implements the ***onCreate()*** event handler shown in Section 2.2.1.



```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
}
```

Shows the initialisation of an Android layout file

Layouts:

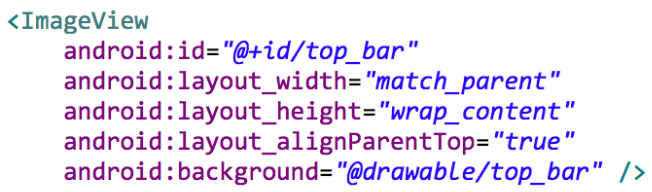
Every layout file contains one root element (ScrollView, RelativeLayout or LinearLayout). Additional layout objects can be added as child elements and are saved as an XML file in res/layout.

UI Widgets:

We describe different types of widgets used in this project:

TextVIews: This is used to display text in the UI

ImageViews: This is used to display images in the UI - local images are stored inside res/drawable



```
<ImageView
    android:id="@+id/top_bar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:background="@drawable/top_bar" />
```

Defines a ImageView widget

ListViews: This is a subclass of the View class explained in Section 2.2. This allows the display of an infinite number of items in a scrollable list.

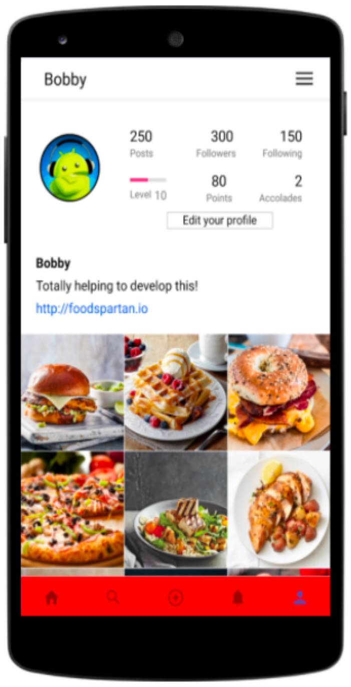


Figure 13: ProfileActivity UI of our application

6.4 Backend development

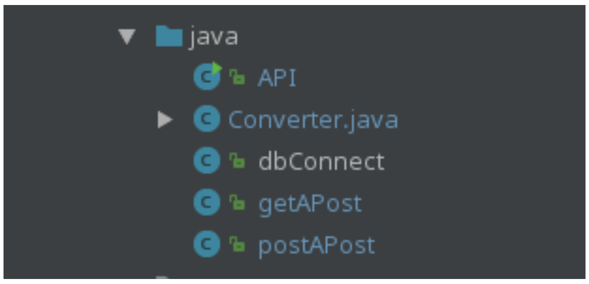
In the back-end there has been a change in framework and a much better understanding of what the requirement for a deployed app will be. It's general structure is largely unchanged from the proposal design and is expected to be implemented as originally laid out as such. The ease at which new routes can be made and with only creating the accompanying DB queries and correct data parsing being the only non trivial part, having the API serve any data in any format is completely feasible.

6.4.1 Changes on initial ideas:

The choice to move framework to Spark from Play comes largely from it's inherent ease of access and simplicity as a micro framework and the potential issues that come from Play's history of code breaking changes from version to version and byte code manipulation. And as all it needs to do is serve the API, most of the functionality and tools found is Play would go unused making the implementation unnecessarily bloated. As Spark is fully compliant to Java syntax the speed at which you can get started made it an easy choice.

6.4.2 Current state of the back-end and achieving user-authentication using Firebase[19]:

The current state of the back-end is an API with GET and POST routes that can query and modify a DB through getAPost and postAPost, dbConnect that establishes DB connections and a JSON parser Converter so that the ResultObject that is produced by the DB can be returned as JSON. The last piece to fit in is the user authentication from Firebase Auth REST API to allow users to only use the API when logged in. Once this has been completed and tested any further database structure modifications can be made, API routes expanded, and the back-end logic to fully handle user access implemented.

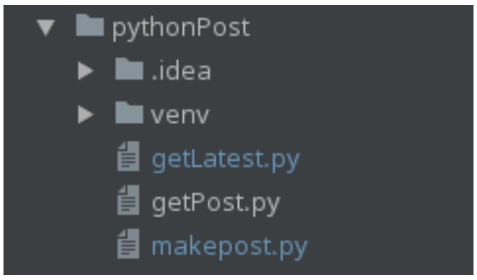


```
▼ java
    API
  ▶ Converter.java
    dbConnect
    getAPost
    postAPost
```

Firebase API exposes all the functionality needed to create users and login, all it requires is a JSON payload with the email and password of the user attempting to login, it will create an idToken, refreshToken and an expiry time. This will allow us to restrict and manage which users are allowed to see and modify what data by adding ownership logic and boolean flags to the database - such as a boolean IsPrivate field to allow only the creator to see their post.

6.4.3 Development:

All back-end software is currently configured to run locally on a developer's machine, as such a small database setup guide was made to allow others to configure their MariaDB instance and the dbConnect class is configured for local access. When the API server and the database are separated the only changes need with be to allow the database user remote access and the ip and port updated in this connector class. Testing of the API is done through a collection python scripts that make GET and POST requests to the backend and print out status codes and returned data. We don’t current envision that any design limitations with the back-end that will restrict or limit out deployment options.



```
▼ pythonPost
  ▶ .idea
  ▶ venv
    getLatest.py
    getPost.py
    makepost.py
```

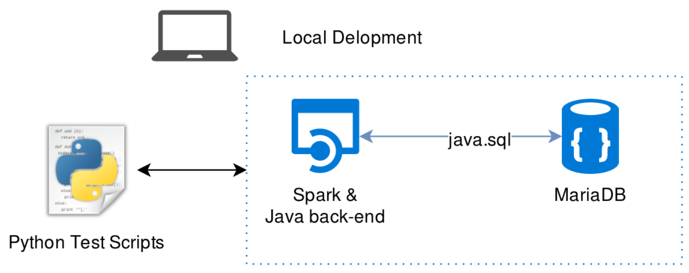
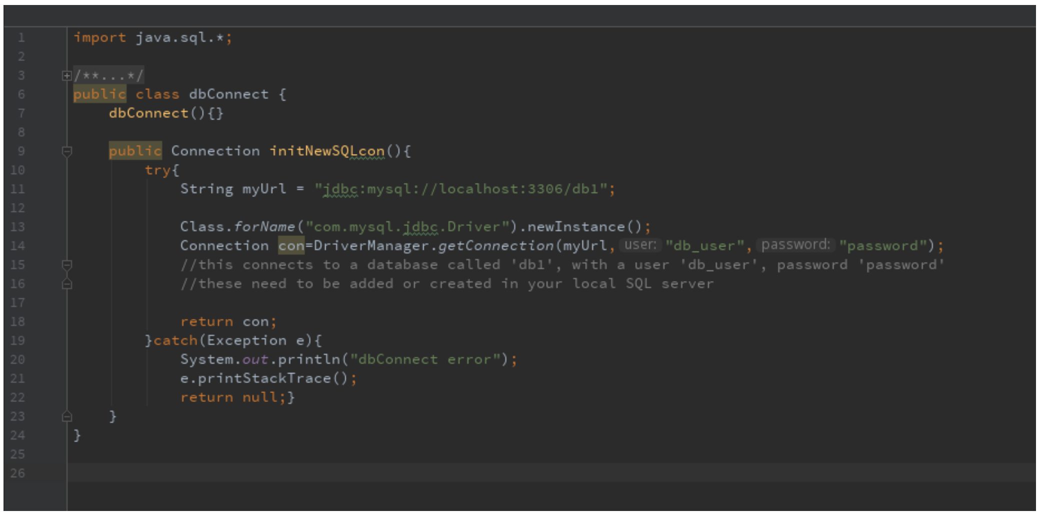


Figure 14: local development of the back-end



```java
import java.sql.*;

/**...*/
public class dbConnect {
    dbConnect(){}

    public Connection initNewSQLcon(){
        try{
            String myUrl = "jdbc:mysql://localhost:3306/db1";

            Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection con=DriverManager.getConnection(myUrl, user: "db_user", password: "password");
            //this connects to a database called 'db1', with a user 'db_user', password 'password'
            //these need to be added or created in your local SQL server

            return con;
        }catch(Exception e){
            System.out.println("dbConnect error");
            e.printStackTrace();
            return null;}
    }
}
```

Figure 15: dbConnect class

6.4.4 Limitations of the current implementation:

One of the bigger considerations for our data storage is access times. With only placeholder data the time to retrieve data is almost instant, this is not representative of what the released app will experience in terms of database size and concurrent access requests. This becomes more important and the database grows and it's vital that there is now degradation in performance over time.

For example as tags come and go in popularity and are created by the users they must have the facility to track and handle new tags or trending tags quickly to allow the app to react to trending behaviour. The current implementation is a string stored TINYTEXT column with '#' separators, this is a limited implementation as to find an individual tag the string will have to be split on '#' with REGEX and then a string a matching function will need to parse each tag to find a match, there is also no way of guaranteeing that tags are actually '#' separated. A solution is to use a JSON data type (actually LONGTEXT in MariaDB), moving to this data type will allow MariaDB to enforce the tags field is JSON formatted the be query the data type with JSON_VALID. As any incorrectly formatted data will be rejected, the error can be logged and sent back to the app to assist with development.

Searching this data will be more safe and reliable and allows for extra data to be added in nested JSON, for example automatically adding related tags under the user entered tags e.g. for a #apple tag adding the related #fruit and #5aDay tags could be useful in producing better search results. Things like UK to US spelling, acronyms or slang could also be handled in this way too.

6.4.5 Future considerations:

For a more robust and performant implementation for the database would be the use of foreign keys to join a table of tags and post IDs. Instead this table can be searched for posts with a specific tag, in this situation searching will always be fast as a string matching algorithm is avoided altogether. An important consideration is if a post is made with an unknown tag that it is added to this table.

Furthermore the direct use of java.sql for database access could be replaced with Object/Relational Mapping (ORM) framework such as Hibernate. This would offload the complexity need to get useful data from the DB as the back-end we will no longer have to directly deal with with ResultObjects.

To get simple scalability with only one database any number of Redis [20] caching servers and be integrated with Spark, when database queries come in, Redis will be checked for the data in system memory and if found returned without touching the DB, completely skipping the slow access of non-volatile storage of the DB. With these changes, deployment and wider scale testing with virtual, or real clients will allow for real validation of the back-end's capabilities.

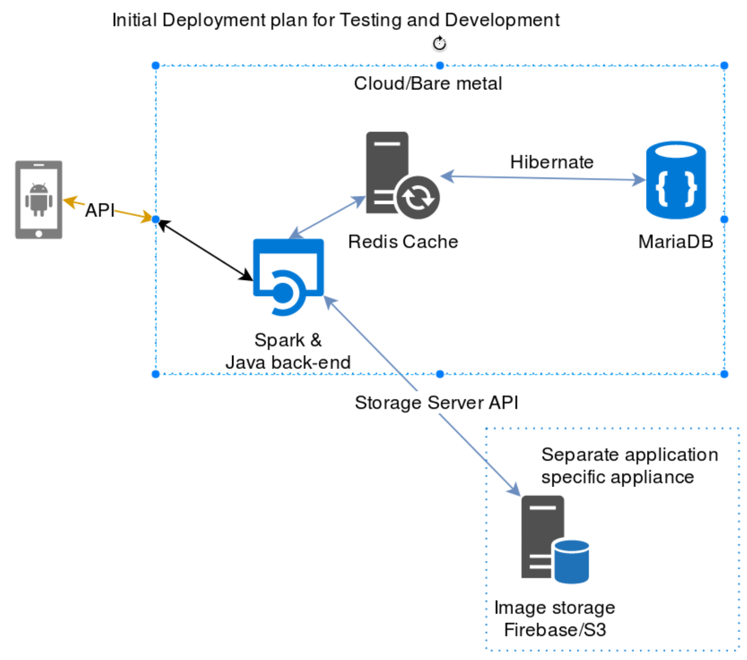


Figure 16: backend deployment

6.5 Unforeseen challenges & solutions

During the implementation phase, the project team encountered a number of challenges that hindered development and solutions considered.

6.5.1 Software Fragmentation

Android OS receives many updates - this creates inconsistency in development as not all users would have upgraded meaning split development focus between different versions of the OS. A possible solution the team looked to was use of a built in tool in the Android SDK that specifies the minimum and maximum versions (API levels) where compilation takes place.

6.5.2 Framework Limitations

Design and performance problems arose from screen limitations - with the use of ListView to display a list of items in a vertically scrolling list, its not designed for a great number of items to be displayed at once, which is a requirement of our project given the nature of the use cases. The methodology used to overcome this was to create a modified version of ListView to hold editable fields - this was achieved by overriding the existing ListView.class to custom implementation to achieve the desired results.

6.5.3 Time and Technical Constraints

One of the biggest challenges encountered was the time and technical constraints during the implementation phrase. With the project team having diverse levels of technical experience and expertise with both Android development and software engineering methodologies - it meant more time was ultimately spent on learning the tools and conventions to the project. However this provided the team with exposure to experiencing a trial run of software development and its lifecycle.

7 EVALUATION & QA TESTING

This section presents the evaluation and testing of our project. During development we established various testing regimes in order to enhance maintainability and prevent bugs in the long run. This would need to ensure our solution is secure, robust and credible in fulfilling project requirements. We explore various testing and evaluative techniques used in software engineering. [16]

The project team took several approaches defined by software engineering conventions. The first approach involves *white box testing* - methods around unit testing the internal structure, design and implementation of our project usually tested from the prospective of the development team. The second approach involves *black box testing* - which include methods around testing a project by users to which the internal structure, design and implementation of the project is unknown.

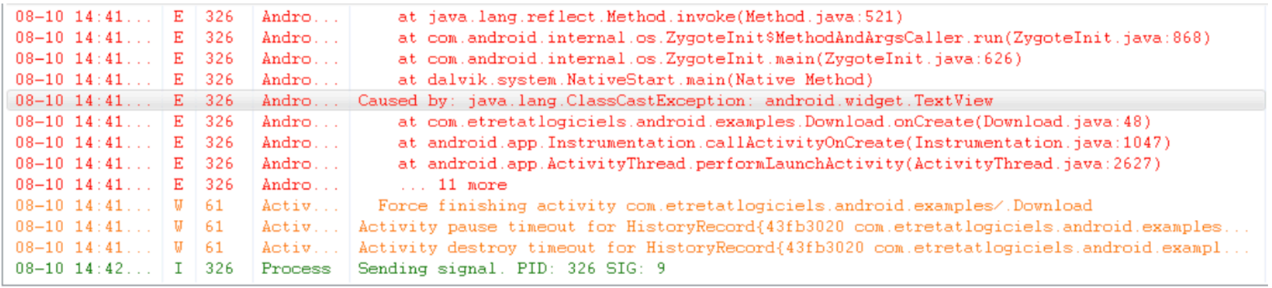
The Android Studio Integrated Development Environment (IDE) provided the team appropriate tools to debug our code and a virtual environment to test our application through emulation. [testing tools]

7.1 Debugging

The Android SDK, integrated in the development environment provides a set of tools that allowed us to successfully debug our application. Throughout the development of the application, most of the debugging was achieved using the Android logging system (LogCat). This a command line tool that is embedded in the IDE.

LogCat [21] provided several fields which assisted the team greatly during the implementation phases of our project.

Priority: which shows how severe the message is. *Time:* shows when the log was generated. *Message:* shows where and what failure that occurred.



```
08-10 14:41...  E  326  Andro...     at java.lang.reflect.Method.invoke(Method.java:521)
08-10 14:41...  E  326  Andro...     at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:868)
08-10 14:41...  E  326  Andro...     at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:626)
08-10 14:41...  E  326  Andro...     at dalvik.system.NativeStart.main(Native Method)
08-10 14:41...  E  326  Andro...  Caused by: java.lang.ClassCastException: android.widget.TextView
08-10 14:41...  E  326  Andro...     at com.etretatlogiciels.android.examples.Download.onCreate(Download.java:48)
08-10 14:41...  E  326  Andro...     at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1047)
08-10 14:41...  E  326  Andro...     at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2627)
08-10 14:41...  E  326  Andro...     ... 11 more
08-10 14:41...  W  61   Activ...   Force finishing activity com.etretatlogiciels.android.examples/.Download
08-10 14:41...  W  61   Activ...  Activity pause timeout for HistoryRecord{43fb3020 com.etretatlogiciels.android.examples...
08-10 14:41...  W  61   Activ...  Activity destroy timeout for HistoryRecord{43fb3020 com.etretatlogiciels.android.exampl...
08-10 14:42...  I  326  Process   Sending signal. PID: 326 SIG: 9
```

Figure 17: LogCat output example taken from IDE

Memory debugging was achieved using the Android Profiler in Android Studio [24]. This includes a number features that help profile the performance of our application. Features such as tracking memory allocation of objects through the system, provision of access to the device file system and viewing heap usage of a specific process all proved useful.

7.2 Functional Requirements Testing

We began by testing the functional requirements of our project using the Android SDK included in Android Studio (IDE). The Android framework provides tools to setup and run the tests. We tested aspects of our application against the functional requirements specification (defined in Section 4) with various iterations of our application produced during implementation.

The following tables show the test cases produced to evaluate the correct functionality of the application. Each test has a unique identifier, followed by the functional requirement it tests, the test input and the pass criteria.

Test Reference	TS-01
Requirement	FR001
Content	Register Username, Password. Logging In.
Input	User enters credentials in the required fields
Pass Criteria	Application displays a loading screen and successfully transports the user to home screen.

Test Reference	TS-02
Requirement	FR002
Content	Edit Profile Info
Input	User goes to the Profile page and clicks “Edit Profile” button, to edit their username, bio, email and password.
Pass Criteria	User edits their profile. 'Tick’ button leads to a dialogue displayed with confirmation of saved changes. Reflected on the database.

Test Reference	TS-03
Requirement	FR003
Content	Taking a Photo
Input	User clicks on the ‘Camera’ icon and takes a picture of their meal.
Pass Criteria	Photo is successfully taken with good resolution from the camera module.

See **Appendix D** for more test cases produced

Test Reference	Result
TS-01	**PASS**
TS-02	**PASS**
TS-03	**PASS**
TS-04	***PARTIAL PASS***
TS-05	**PASS**
TS-06	**PASS**
TS-07	**PASS**
TS-08	**PASS**

Table 6: FR Test Case Results

Table: outlines the test results of the test cases. All functional requirements were met during the implementation phases with the exception of TS-04 which passed partially. This is because photos were successfully uploaded to the database however image recognition functionality hasn't yet been developed in the latest iteration of our application (see Section 6 for more details).

7.3 UI Testing

In addition to unit testing of functional components that up our application. Its also vital to test the usability and behaviour of the application user interface UI running on diverse set of devices. This fundamentally ensures that the application returns the correct output and response to a sequence of user actions.

There are several approaches we followed in testing the UI. The first approach was running tests manually on bespoke android device - providing verification that the application behaves in the correct way. The second approach was to automate UI testing using a software testing framework. A automated test involves using tools to perform tasks to cover specific usage scenarios.

A number of tools are provided by the Android SDK that allow us to run automated tests. UI Automator & UI Automator Viewer [23] - a GUI tool to scan and analyse UI components and a Java library that contains APIs to create bespoke functional UI tests, and a exception engine to automate and run the tests.

The following table shows abstract UI test cases that evaluate the correct UI output of our application with each test case tested using a manual and automated approach.

Test Reference	Content	Result
TSUI-01	Checks if the application UI gives scope for keyboard to allow input in EditText objects found in forms such as username, password etc	**PASS**

Test Reference	Content	Result
TSUI-02	Checks if the application UI handles all orientations correctly	**PASS**
TSUI-03	Progress dialogs shown correctly when loading data from a database.	**PASS**
TSUI-04	Checks if the application UI shows all context menus when ListView items are selected.	**PASS**
TSUI-05	Checks if the application UI shows buttons correctly in every screen	**PASS**
TSUI-06	Checks if the application UI handles headers in the correct way	**PASS**

Table 7: UI Test Cases

7.4 Performance Assessment

In recent years, the rising performance of applications and mobile devices has increased the aims and expectations in fulfilling the performance demands of our users. Providing users with a excellent user experience is highly dependent on a number of factors around usability, error handling and response times - see section 4.1.2.

With these expectations in mind - the team performed a number tests to evaluate the performance of our application on different hardware resources and configurations using official Android profiling tools - see Section 7.1.

Performance evaluation in this project is based on resource consumption - expressed in terms of time (speed).

We ran a number of trials on a diverse range of Android OS devices with different configurations * - using the Android emulator built into the Android Studio IDE.

- Google Pixel 2 Phone (2017) - OS: Android 8.0 CPU: Octa-core 2.35GHz x 4 + 1.9GHz x 4 RAM: 4GB
- Samsung Galaxy S7 (2015) - OS: Android 7.0 CPU: Octa-core 2.3GHz x 4 + 1.6GHz x 4 RAM: 4GB
- HTC One Max Phone (2013) - OS: Android 4.3 CPU: Quad-core 1.7GHz RAM: 2GB
- Asus Zenpad Z8 Tablet (2016) - OS: Android 6.0 CPU: Hexa-core 1.4GHz x 4 + 1.8GHz x 2 RAM: 2GB
- Asus Zenpad Z10 Tablet (2016) - OS: Android 6.0 CPU: Hexa-core 1.4GHz x 4 + 1.8GHz x 2 RAM: 3GB
- Asus Google Nexus 7 (2013) - OS Android 5.0 CPU: Quad-core 1.5GHz x 4 RAM: 2GB

We calculated the mean results to give an average outcome of response times through the traversal of different parts of the application.

Test Phase	AndroidOS Phones	AndroidOS Tablets
Login/Signup page - Home	2.1 seconds	2.8 seconds
Home - Store	1.2 seconds	2.0 seconds
Store - Upload	2.0 seconds	1.8 seconds
Upload - Camera	1.2 seconds	1.5 seconds
Upload - Notifications	1.0 seconds	1.0 seconds
Notifications - Profile	1.0 seconds	1.2 seconds

Table 8: average calculated response times*

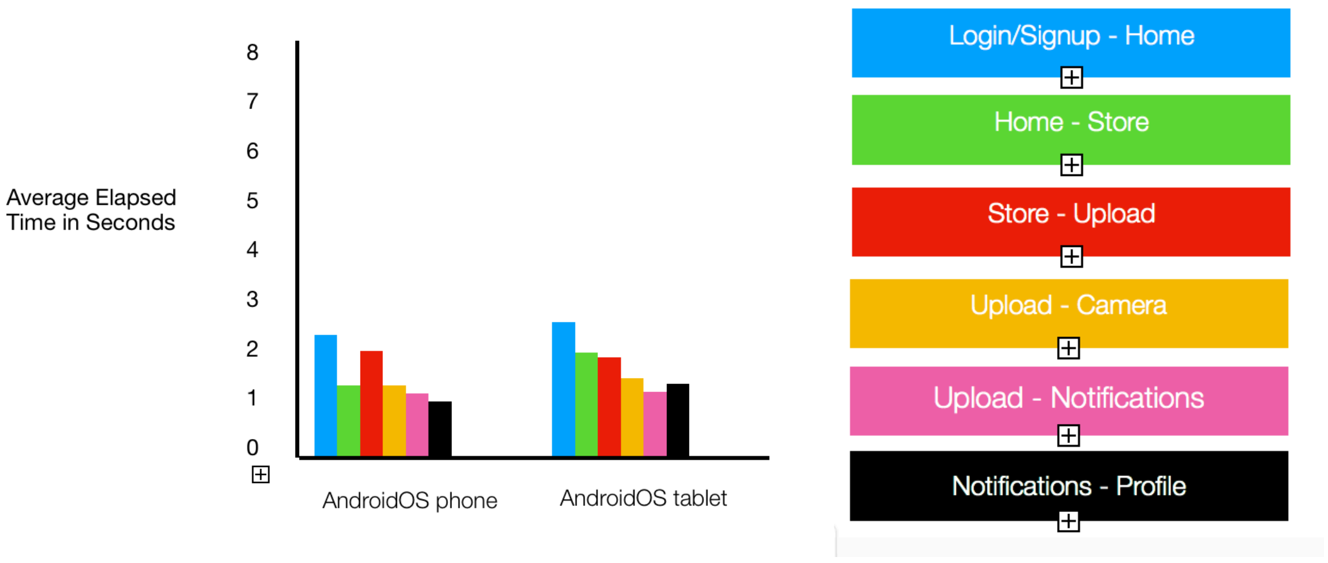


Figure 18: graph depicting average response times

7.5 Formative Evaluation

In previous parts to this section we looked at techniques around testing and evaluation from a viewpoint of a developer. Another important dimension to testing/evaluation is to approach this from a user prospective. We conducted a number of focus groups at different development phases to get user feedback around areas of usability.

One of the most important ones we held was a cognitive walkthrough evaluation [22] during the design phase of our development process. This is a usability evaluation method in which evaluators work through a series of tasks in our application and ask a set of questions from their prospective. The focus of the cognitive walkthrough is to get a better understanding of our application's learnability for new users. Feedback received was important in shaping our designs and the user experience of our application through the implementation phase.

8 PROJECT CONCLUSION

8.1 Reflection

The project team had an ambitious goal of revolutionising the way we think about healthy eating - by making it a rewarding and connected experience. This project gave the team with different technical expertise the opportunity to work on a project in a simulated environment to design and develop an application. For many of us it was the first time we learnt about approaching the software development ecosystem - starting with conceptual ideas to software engineering conventions to software development such as agile and UML modelling, right through to implementation. The other dimension to this project was how well members of the worked together in meeting our objectives collectively.

8.2 Future Work

Having taken a iterative approach combined with strong ambitions and a clear vision around the output of our project - there is scope for expansion of functionality taking the project to new heights in the areas of machine learning and image recognition.

APPENDICES

Appendix A: use case diagrams

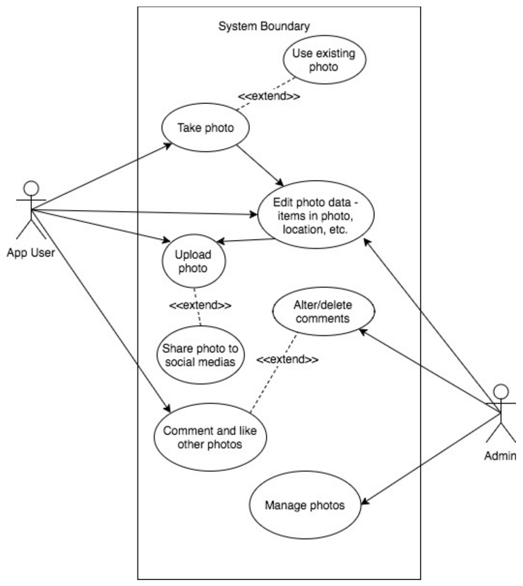
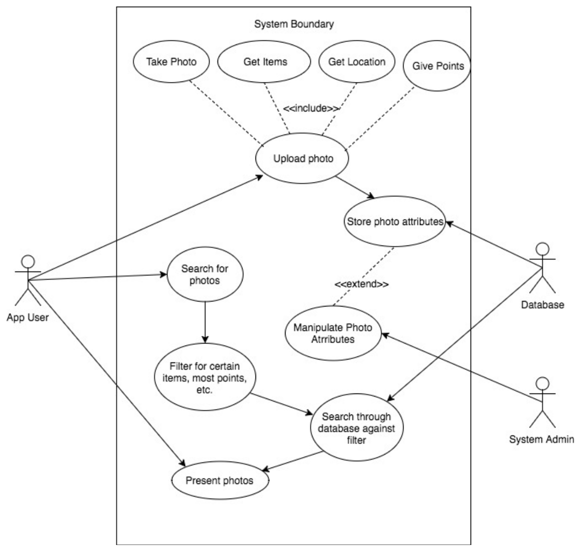


Photo use case diagram



Database use case diagram

Appendix B : use case scenarios

Use Case Name	Earn accolades
Context	The user earns accolades by uploading images on to the app
Primary Actor	User
Precondition	The user needs to be logged in to access and earn accolades
Success Post Condition	The image will be processed via image recognition algorithm to identify whether there are any food related objects.
Trigger	The user wants to earn accolades
Main Success Scenario	1.The user directs to the upload navigation bar 2.The user presses the upload photo button 3.Android will display the photo storage of the user's current device 4.The user will choose one of the photos they want to select and hits send 5.The storage option will close and the process of uploading a photo will occur. 6.The image recognition algorithm will execute once the image is selected. a.If the recognition is successful, an appropriate number of accolades will be wired to the user's account. b.If the recognition is unsuccessful, no accolades will be given to the user and upload will be rejected.
Exceptions	The user will not get any accolades if they choose to cancel their image upload(if successful).

Use Case Name	Comment
Context	Commenting on a post
Primary Actor	User
Precondition	The user needs to be logged to comment
Success Post Condition	The comment will be posted as it will be accessible by other accounts
Trigger	The user wants to comment
Main Success Scenario	1.The user scrolls through the timeline. 2.The user presses the comment button, it will direct the person to a new window 3.The user scrolls through the comments (if there are any) 4.The user writes something and clicks the icon to post it. 5.The comment will be shown after clicking post a.If the comment is successful then it will post it alongside other comments. b.If there are offensive languages written in the comment, it will decline.

Use Case Name	Like a post
Context	Liking a picture
Primary Actor	User
Precondition	The user needs to be logged in and has followed the account in order to do this action
Success Post Condition	The like will be seen by other users who are using this application
Trigger	The user wants to comment
Main Success Scenario	1.The user scrolls through the timeline. 2.The user presses the like button. 3.The other accounts will be notified when their post is liked. 4.The user writes something and clicks the icon to post it. 5.The like button will be successful on one condition: a.If the user has followed a specific account. b.If the post has already been liked, the like button will do the opposite task.
Exceptions	The user can undo liking a post by pressing the same button again - the user notification will not be effected.

Appendix C: Database table attributes

photos

Attributes	Datatype	Description
photo_id (PRIMARY KEY)	INT	The unique id for the photo.
points	INT	The amount of points this photo holds.
image_uri	STRING	Uri containing the photo.

comments

Attributes	Datatype	Description
comments_id (PRIMARY KEY)	INT	The unique id for the comment.
post_id (FOREIGN KEY)	INT	The id of the post that the comment is linked to.
comment	VARCHAR	The comment provided.
timeOfComment	DATETIME	The time when the comment was made. This is to get them in a certain order when called.

Appendix D: Functional Requirement Test Cases

Test Reference	TS-05
Requirement	FR015
Content	Present Photo
Input	User clicks on the ‘Add to timeline’ button to share their photo on the app timeline for other users to view.
Pass criteria	Photo appears on timeline with a timestamp attached.

Test Reference	TS-06
Requirement	FR0011
Content	Comment & Like Photo
Input	User double taps the ‘Like’ button and clicks ‘Comment’ to add a text comment on a photo
Pass criteria	User is able to successfully leave a like/comment on a photo

Test Reference	TS-07
Requirement	FR0013
Content	Search Photos
Input	User clicks on the ‘magnifying glass’ iconic at the header of the app and searches for a specific photo using a hashtag, location or a username
Pass Criteria	Displays photos of that are allocated to those key search terms

Test Reference	TS-08
Requirement	FR0016
Content	Logging Out
Input	Once user is finished using the app they click the ‘Sign out’ button.
Pass criteria	User clicks “Sign out” and are directed to the ‘Login’ page where they can login again.

REFERENCES

[1] Deterding et al. 2011

[2] Octalysis Gamification Framework - http://yukaichou.com/gamification-examples/octalysis-complete-gamification-framework/

[3] Android architecture - http://developer.android.com/topic/libraries/architecture

[4] Android studio - http://developer.android.com/studio

[5] Activity lifecycle overview - http://developer.android.com/guide/components/activites/activity-lifecycle

[6] Convolutional Neural Networks for Visual Recognition - http://cs231n.github.io/

[7] TensorFlow Tutorial - https://www.tensorflow.org/get_started/

[8] Software Development Process (2013) - http://www.computerhope.com/jargon/softdeve.htm

[9] Ian Sommerville, Software Engineering, 8th Edition, 2007

[10] Waterfall Model (2015) - http://istqbexamcertification.com/what-is-waterfall-model

[11] Incremental Model (2015) - http://istqbexamcertification.com/what-is-incremental-model

[12] Spiral Model (2015) - http://istqbexamcertification.com/what-is-spiral-model

[13] Agile Model (2015) - http://istqbexamcertification.com/what-is-agile-model

[14] GIT Source code management - https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

[15] Project GitLab - http://gitlab.doc.gold.ac.uk/rkill002/Software-Project-Y2-M

[16] Software Engineering, A Practitioners Approach, 8th Edition, Pressman & Maxim

[17] UML Distilled 3rd Edition, Martin Fowler

[18] Android User Interface Guidelines - https://developer.android.com/design/

[19] Firebase guide - https://firebase.google.com/docs/guides/

[20] Regis - http://oldblog.antirez.com/post/take-advantage-of-redis-adding-it-to-your-stack.html

[21] LogCat - http://developer.android.com/studio/command-line/logcat

[22] Cognitive walkthrough method - The cognitive walkthrough method: a practitioner's guide, Nielsen & Mack “Usability Inspection Methods”, 1994

[23] UI Automator - https://developer.android.com/training/testing/ui-automator

[24] Android Profiler - https://developer.android.com/studio/profile/android-profiler