#### 

TEAM #3 FLIXTYLE

DESIGN

SPECIFICATION

horizontal line

**Team 3** May 19th, 2019

Flixtyle

**Software Design Specification**

Introduction to Software Engineering-41

Table of Contents

[**1. Preface**](#_zgc6ymbabshi)9

[1.1 Objective](#_w9pujtxgfovy) 9

[1.2 Readership](#_ctza0hth15mj) 9

[1.3 Document Structure](#_vj2lvr1yea7) 9

[1.3.1 Preface](#_idbe4ebq435m) 9

[1.3.2 Introduction](#_f84kdxqfl56c) 9

[1.3.3 System Architecture](#_1bhm28gidmll) 9

[1.3.4 Hearts System](#_sz0gyeollibr) 9

[1.3.5 Discovery System](#_5m8g0j4ov65c) 10

[1.3.6 Recommendation System](#_gozukqynu2ma) 10

[1.3.7 Classification System](#_jo1o1cusqd65) 10

[1.3.8 Account Management System](#_mn7ko98rqbtg) 10

[1.3.9 API](#_9wz07fxk64xu) 11

[1.3.10 Database](#_7pil8a7u6ygo) 11

[1.3.11 Testing plan](#_pfcboy7ddqlz) 11

[1.3.12 Development plan](#_1s55buba1b69) 11

[1.4 Glossary](#_sqlhreo8sguq) 11

[**2. Introduction**](#_9eh01f91e2aw)13

[2.1 Objective](#_ocesuj64zcoh) 13

[2.2 Applied diagrams](#_sflm1ax7rwxt) 13

[UML](#_j6466h4khb0v) 13

[Package Diagram](#_7zs11uch728i) 14

[Deployment Diagram](#_rsvh6le3al7n) 18

[Class Diagram](#_rsvh6le3al7n) 18

[State Diagram](#_rsvh6le3al7n) 18

[Sequence Diagram](#_rsvh6le3al7n) 18

[2.3 Applied tools](#_k1wsxlfgc8xn) 19

[2.4 Project scope](#_k1wsxlfgc8xn) 19

[**3. System Architecture**](#_1a9jho7vnybe)20

[3.1 Objectives](#_5vh0bqc2zaal) 20

[3.2 System Organization](#_xhfbsx5e4k8f) 20

[3.2.1 Hearts System](#_v0aljsywjqxf) 20

[3.2.2 Discovery System](#_ukm4qtn9uvln) 21

[3.2.3 Recommendation System](#_lodnh7r4z1md) 21

[3.2.4 Classification System](#_v54kxxzigrh1) 22

[3.2.5 Account Management System](#_mjlxb9st384a) 22

[3.3 Package diagram](#_twf4grdnsnh7) 23

[3.4 Deployment diagram](#_q0ap4bj0oyim) 24

[**4. Hearts System**](#_pfy4xa3eshe1)25

[4.1 Objective](#_5uzf9lo1oue6) 25

[4.2 Class Diagram](#_ld2bkf1bklgh) 25

[4.2.1 User](#_dcfkreoq3otv) 26

[4.2.1.1 Attributes](#_17v72v82u4qj) 26

[4.2.1.2 Methods](#_rdqtg72hckpp) 26

[4.2.2 DB handler](#_5y9rmnnnap9l) 26

[4.2.2.1 Methods](#_w9n066we0abz) 26

[4.3 Sequence Diagram](#_m58m9yulyb3z) 27

[4.3.1 User](#_7vhser2koebz) 27

[4.4 State Diagram](#_4o6vgug51rj9) 27

[4.4.1 Items](#_12fbac4sdzsd) 27

[**5. Discovery System**](#_fcdp4ziyc9hg)28

[5.1 Objective](#_cez8xvpdyylc) 28

[5.2 Class Diagram](#_m6do9x3n282m) 28

[5.2.1 User](#_ewxdel7lyt4j) 28

[5.2.1.1 Attributes](#_gmm2r9tdzdyb) 28

[5.2.1.2 Methods](#_8zbr5y860nvv) 29

[5.2.2 DB handler](#_io2dokl25vby) 29

[5.2.2.1 Methods](#_kcmwecfjtsu5) 29

[5.2.3 Items](#_thj1dywlc99v) 30

[5.2.3.1 Attributes](#_ebs9zjh9f8k) 30

[5.2.3.2 Methods](#_dheh8plhjuvz) 30

[5.3 Sequence Diagram](#_rcxmocwtt90o) 30

[**6. Recommendation System**](#_21fyr172hjuk)31

[6.1 Objective](#_135yancjgvkw) 31

[6.2 Class Diagram](#_2df470r5j55f) 31

[6.2.1 User](#_xc1fptdoopmi) 31

[6.2.1.1 Attributes](#_k80a1rwclqbc) 31

[6.2.1.2 Methods](#_720j9zmxxrkh) 32

[6.2.2 Item](#_v88f94q8db3f) 32

[6.2.2.1 Attributes](#_x60qih6d9hqt) 32

[6.2.2.2 Methods](#_k2lmqzqbeo54) 32

[6.2.3 Item Categories](#_ephu84lxo2sq) 33

[6.2.3.1 Attributes](#_rz9yrjdj0g45) 33

[6.2.3.2 Methods](#_rz85znby47yr) 33

[6.3 Sequence Diagram](#_hjg0q9x2ne0d) 33

[6.3.1 Show recommendations function](#_lrnjzk28yvg6) 33

[6.3.2 Add to “Hearts” list function](#_82y5lpgutymd) 34

[**7. Classification System**](#_v0xh7he8iy4z)34

[7.1 Objective](#_ug9ypsizjfe1) 34

[7.2 Class Diagram](#_v2qrr7tlprc5) 34

[7.2.1 User](#_7k5ayiqq36sl) 35

[7.2.2.1 Attributes](#_j5cpah31julp) 35

[7.2.2.2 Methods](#_tj011mvaehmg) 35

[7.2.2 DB Handler](#_xra2k5ousfu4) 35

[7.2.2.1 Methods](#_k00k36a7m3zr) 35

[7.2.3 Item\_Categories](#_o7eul6f3cqnb) 35

[7.2.2.1 Attributes](#_fazyiid55b9p) 35

[7.2.2.1 Methods](#_2zobbyaz3xx5) 35

[**8. Account Management System**](#_nqr9j8o4mm0a)37

[8.1 Objective](#_ibnu6tlapz1s) 37

[8.2 Class Diagram](#_vh5z95j23rnt) 37

[8.2.1 User](#_2t678af2sljw) 37

[8.2.1.1 Attributes](#_b3nremcg9lgf) 37

[8.2.1.2 Methods](#_c8nqfk46dgdk) 38

[8.2.2 DB Handler](#_d7iwbhw7gi5s) 39

[8.2.2.1 Methods](#_unp81l434t0g) 39

[8.2.3 SNS\_Account](#_z835ihnzoskg) 39

[8.2.3.1 Attributes](#_mbyjujnv9y0z) 39

[8.2.3.2 Methods](#_pbojtnb2kphp) 39

[**9. API Design**](#_2e3zzvfd7t9k)40

[9.1 Objective](#_c7syv01tztxq) 40

[9.2 Variables](#_m4zmp62ligpr) 40

[9.3 Request and Responses](#_a8htav2782hv) 40

[9.3.1 Request item thumbnail](#_lp7u5xbxocd3) 40

[9.3.2 Request item website URL](#_xonei82ip6ng) 41

[9.3.3 Create account request](#_yom2dzg34gui) 41

[9.3.4 View account request](#_5inv6y2nm5y4) 41

[9.3.5 Update account information request](#_xy76wpbii8l1) 41

[9.3.6 Delete account request](#_8kxa0kadb9lg) 42

[9.3.7 View “Hearts” list request](#_9t59mb4k8bh1) 42

[9.3.8 Add item to “Hearts” list request](#_xyn7u1s268l6) 42

[9.3.9 Remove item from “Hearts” list request](#_y9oehuqdz8o8) 42

[9.3.10 View recommendations request](#_5fg758bpqeld) 42

[9.3.11 Request Discoverable Items](#_3k5tidlaenh7) 42

[9.3.12 Send item liked request](#_31v6jc63oas2) 43

[**10. Database Design**](#_n2r2orlml9ae)43

[10.1 Objective](#_hz9vxafsswmm) 43

[10.2 Data types](#_jvwbsquhepxe) 43

[10.3 Table](#_clydlqfjaw6r) 44

[10.3.1 User](#_t2erw8heg1zx) 44

[10.3.2 SNS Account](#_5mdy2lvg61wg) 44

[10.3.3 Item](#_mykanzf4xzt2) 45

[10.3.4 Item Categories](#_g8nz7zxk5qgi) 45

[**11. Testing Plan**](#_4ixbejobkjd5)45

[11.1 Objectives](#_4cgpshnz0voq) 45

[11.2 Testing Policy](#_5jp21ad1wyme) 45

[11.2.1 Development Testing](#_fwu0zogc6qib) 45

[11.2.1.1 Component Testing](#_f746esofl293) 46

[11.2.1.2 Integration Testing](#_clpba4p7xrlb) 46

[11.2.1.3 System Testing](#_gj1r5o9bar85) 46

[11.2.1.4 Acceptance Testing](#_iiy0cwhvghaa) 46

[11.2.2 Release Testing](#_szmfpkhkqnlr) 46

[11.2.3 User Testing](#_bg9xylhv5gsb) 46

[11.3 Test Case](#_6k1v2l7z6b7n) 47

[11.3.1 User Management System](#_d1j3vwo6mfg2) 47

[11.3.1.1 Login](#_gdhr1bvuvx26) 47

[11.3.1.2 Sign Up](#_q0hvj9hdx8nh) 47

[11.3.1.2 Logout](#_ou6eehmfyyi2) 47

[11.3.2 Discovery System](#_rkpsd3pcw4f7) 48

[11.3.2.1 Swipe Left](#_4xma4a3xalr9) 48

[11.3.2.2 Swipe Right](#_nlznu7atcpo5) 48

[11.3.2.3 Swipe Up](#_3pnycjnir1s3) 48

[11.3.3 Recommendation System](#_4ros4rr71y4y) 49

[11.3.3.1 Base case](#_ps6v5clckh87) 49

[11.3.3.2 Add to “Hearts” list](#_b9jw55fpnnrf) 49

[11.3.4 Classification System](#_ou2sbix51sfk) 49

[11.3.4.1 Process Likes to categories](#_z01muqq2pzb) 49

[11.3.5 Account Management System](#_145r7w5lbtlr) 50

[11.3.5.1 Create an account by email](#_6d0rgenb4zol) 50

[11.3.5.2 Update user’s information](#_6oja6q4it8dc) 50

[11.3.5.3 Link an SNS Account](#_p90o3ebd3qwv) 50

[**12. Development Plan**](#_tmt30eeosseg)51

[12.1 Objectives](#_2wjf3fl2qi19) 51

[12.2 Programming Language & IDE](#_1sgemh1ahxm) 51

[12.2.1 Programming Language](#_xtpfr4gie1m2) 51

[12.2.1.1 Java](#_7bk3pkq5d52z) 51

[12.2.1.2 Node.js](#_4dkcbb3tk6bp) 51

[12.2.2 IDE](#_nvinmrnmwxbm) 51

[12.3 Version Management Tool](#_43w6cqr93onq) 51

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Initial | 05/19/2019 | Initial version | 0.1 |
|  |  |  |  |

# 1. Preface

## 1.1 Objective

The purpose of this document is to outline the system design and structure specifications of the “Flixtyle” application. The document includes the architecture of each system using UML, API design, Database Design, and Testing plan. By reading this document, the reader can understand how to work the system and relationship of its components.

## 1.2 Readership

This document is written for the project manager, project team and development team. Also, this document is for the end users and other various stakeholders who are involved in the support, maintenance of the system. In other words, the readers of this document are all members of the development and maintenance teams of the “Flixtyle” application which introduced in this document. It is recommended to know most of the terms in the glossary section before reading through the document.

## 1.3 Document Structure

### 1.3.1 Preface

The ‘Preface’ defines the reader of this document and introduces the structure of the document. Moreover, it describes the purpose and outline of each table of contents when introducing structures.

### 1.3.2 Introduction

The ‘Introduction’ defines describes the diagrams and tools used in this document. Furthermore, it describes the scopes of the “Flixtyle” application which introduced in this document.

### 1.3.3 System Architecture

The ‘System Architecture’ defines the general description of the systems that our team wants to develop. In addition, the overall structure of each system is represented as a block diagram. Each relationship and how it is actually used is described as a Package Diagram and Deployment Diagram.

### 1.3.4 Hearts System

The ‘Hearts System’ describes how a user can manage their Heart list. Users can heart their favorite items through the discovery and search process. These favorite items are stored in the user database and user can be checked in the user personal account by the Account system. Moreover, Users can manage items from the heart list by adding or deleting them. The document expresses and describes of the Hearts System structure by using Class Diagram, Sequence Diagram, and State Diagram.

### 1.3.5 Discovery System

The ‘Discovery System’ describes how to get user preference. Since the recommendation system is a customized service according to the user's preference, the system must have to investigate user's preference data by Discovery System. User can represent their preferred items through adding or not adding to likes. These items are stored in the recommendation database and used to create the recommendation item list. Additionally, the Heart system allows users to store their favorite items in the heart list. The document expresses and describes of the Discovery System structure by using Class Diagram, Sequence Diagram, and State Diagram.

### 1.3.6 Recommendation System

The “Recommendation System” describes the system that recommends items to the user as a list. Recommendation system makes a recommendation item list by using the Discovery System to select the preferred item for users and saving these items in Recommendation database. The items recommended to the user depends on the actual data provided by the user. The document expresses and describes of the Recommended System structure by using Class Diagram, Sequence Diagram, and State Diagram.

### 1.3.7 Classification System

The “Classification System” describes how the system classifies items for user and system. Classification system sort categories according to the type of items and store them in the database. This classification system not only makes programs easy to manage items but also users can efficiently search items. The document expresses and describes of the Classification System structure by using Class Diagram, Sequence Diagram, and State Diagram.

### 1.3.8 Account Management System

The “Account Management System” describes how to manage the user account. It is about how the user can either sign up or log in using User database Account Management System compares the user values to the Account database, and a user can enter to a user account if it matches. If the user doesn't have an account, the user can sign up and the Account database saves the user profile that the system needs by default. Additionally, the Account Management System allows users to freely change user profiles or check user’s Heart list. The document expresses and describes the Account Management System structure by using the Class Diagram, Sequence Diagram, and State Diagram.

### 1.3.9 API

The “API” describes the interfaces between the different systems and components. The interfaces are going to be implemented following the RESTful API standard. RESTful is a stateless API that uses JSON to communicate. Javascript Object Notation(JSON) is a human readable text format that a standard in the industry. JSON can be parsed and edited by many programming languages out of the box without needing to add any external libraries.

### 1.3.10 Database

The “Database” definess database diagram that the project will be based on. The designed database is represented by the ER diagram, and the data of the program is structured and stored through the created ER diagram.

### 1.3.11 Testing plan

The “Testing plan” describes the Test Policy and Test Case. The purpose of this chapter is to establish a preliminary plan for system testing to determine if the system is operating as intended to identify and analyze the system for defects after completion of the system. This part is divided into a testing policy chapter and a testing case chapter. The Testing Policy sets the criteria for testing and The test case gives a set of more detailed input and output.

### 1.3.12 Development plan

The “Development plan” describes the programming language and the IDE that is going to be used. Also, it explains the version of programming languages and management tool.

## 1.4 Glossary

* Seller   
  The affiliates to the system who sells products on the Flixtyle service.
* User   
  Customers who use Flixtyle services to purchase products.
* Fashion item   
  Wearable items. Limited to clothing, hats, swimming suits
* Product   
  A fashion item the seller has registered to the system
* Recommendations  
  Products which are shown to relevant customers and acts as an advertisement to the user
* Discovery   
  The process of verifying whether the user is actually interested in the products the system predicts.
* Save   
  The act of adding a product to the “Heart” List, so that the user will be able to quickly find the product in the future.
* “Heart” List   
  The list of the products that a user has saved.
* Swipe   
  The act of swiping across the touchscreen.
* Authentication   
  The process of verifying the user’s identity.
* Request to the server  
  A message sent with the HTTP protocol to the server with information from client.
* Filtering   
  The process of separating unwanted items from a list of items, where these unwanted items are decided from the options.
* SNS   
  Social networking service, for example, Facebook, and Twitter.

# 

# 2. Introduction

## 2.1 Objective

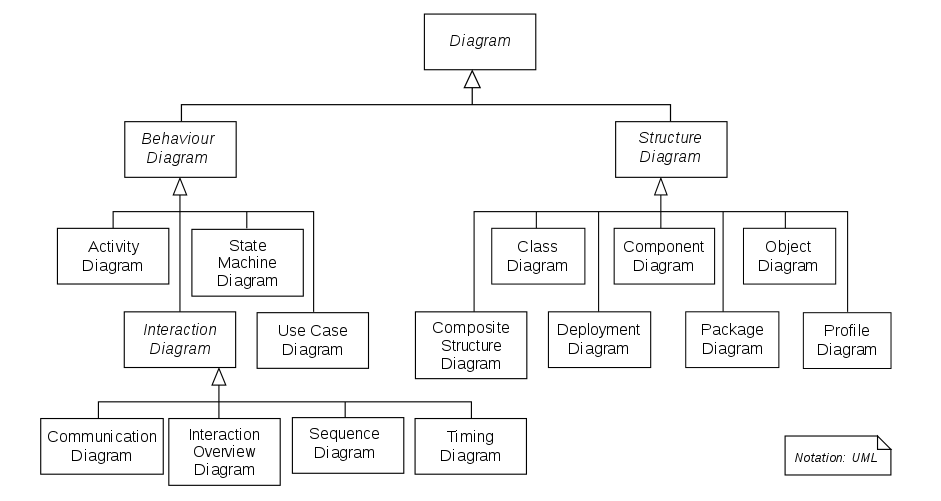
This chapter will explain the diagrams used in this document, state the applied tools and outline the project scope.

## 2.2 Applied diagrams

#### UML

This document will feature several diagrams from the Unified Modelling Language (UML). UML is a general-purpose modeling language made for standardizing the visualization of a software system. The modeling language was developed in 1994-1995, by Ivar Jacoson, Grady Booch and James Rumbaugh and has evolved steadily since.

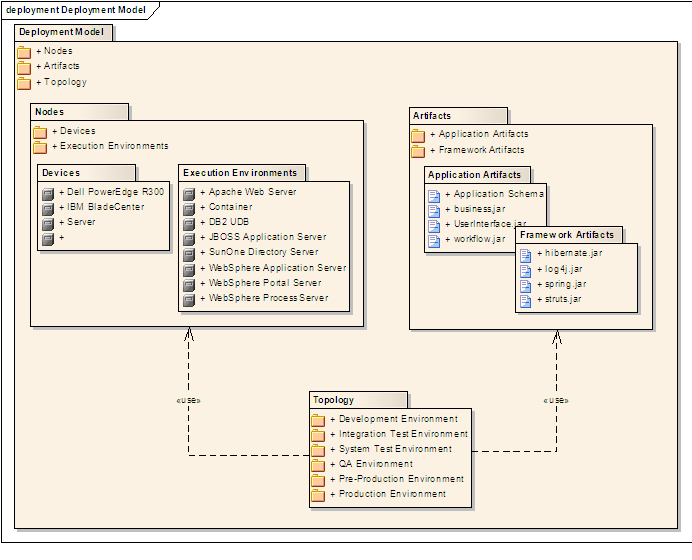
One of the main reasons behind UML’s success is how the language offers intuitive ways to visualize a software system’s architecture, including system elements like activities, components, and their interactions. UML can be divided into two different views, static and dynamic. The static (or structural) view describes the static structure of the system using objects, attributes, and relationships. Class, deployment and package diagrams belong to this view. The dynamic view, on the other hand, emphasizes the dynamic behavior of the system. It does this by describing interactions among objects and how these changes the internal states of the various objects. Important diagrams belonging to this view are state and sequence diagrams. A general overview of the various UML diagrams is provided below.



#### Package Diagram

Package diagrams aim to depict the dependencies between the packages that make up a model. A package is a general-purpose element for organizing model elements and diagrams into groups. Elements can be classes, interfaces, objects, tables or other packages. Generally, these models aim to describe the relationships between major elements in the system (as opposed to component diagrams). Package diagrams may also include use cases.

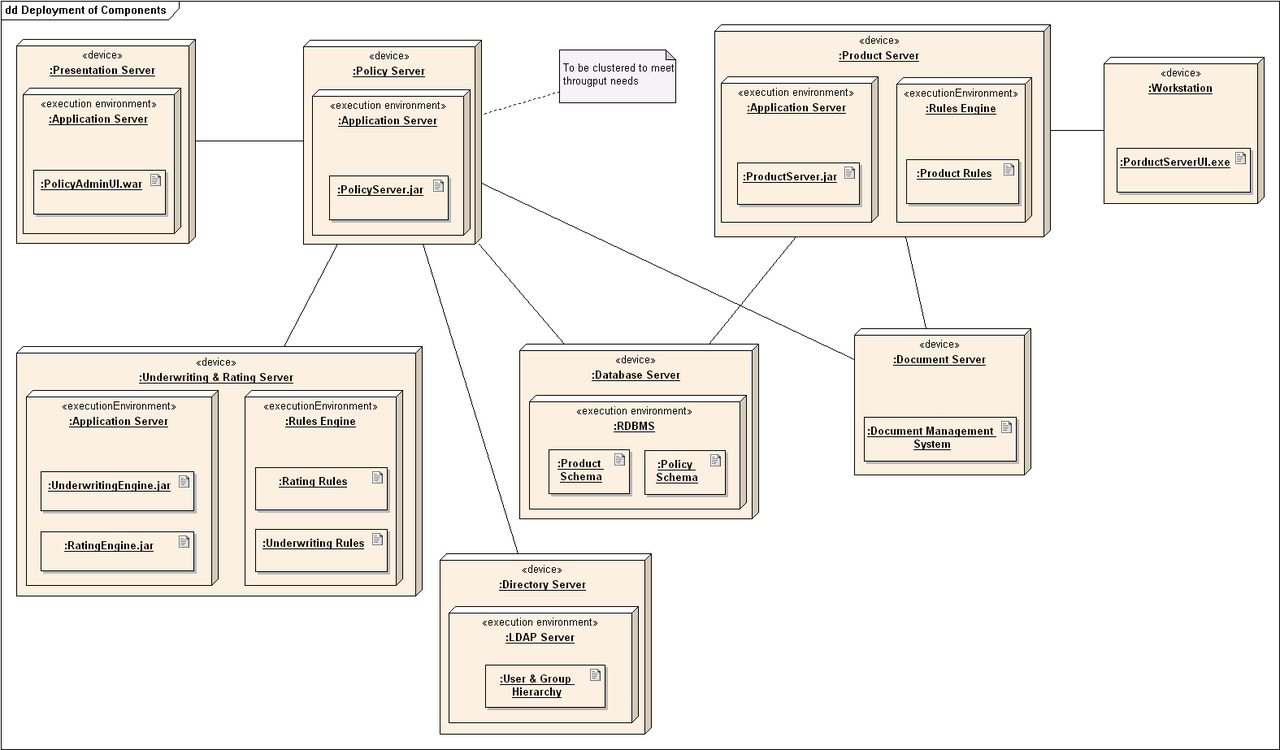
An example of a package diagram can be seen below.



#### Deployment Diagram

Deployment diagrams aim to model the physical of artifacts (software components) on nodes (hardware components) in the system. The purpose of these diagrams is to provide stakeholders with an overview of how the system will be implemented physically. This can help engineers determine what hardware components they need, what criteria these components should fulfill and the dependencies between the hardware components. This is very valuable to identify potential fault propagation.

Deployment diagrams consist of nodes, shown as boxes. These nodes represent physical hardware devices or a virtual execution environment. Each node has a number of artifacts (elements), describing the functions, classes, etc. that will run in that particular device or environment. An example of a deployment diagram can be seen below.

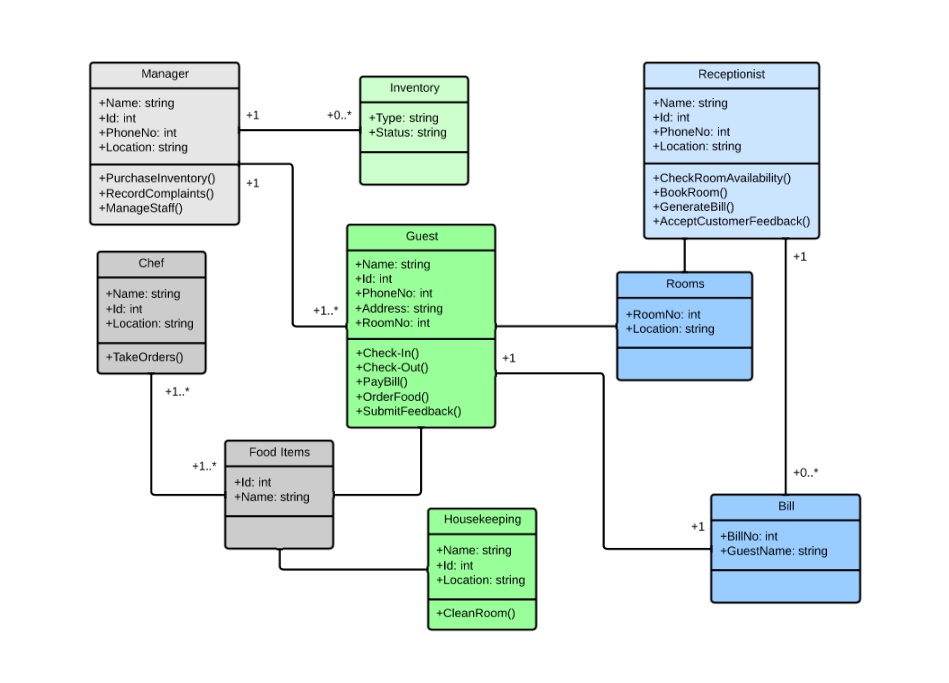


#### Class Diagram

Class diagrams are structural diagrams that aim to describe the classes of a software system, their attributes (i.e. variables, functions) and the relationships among objects. Class diagrams are one of the most detailed UML diagrams and provide a relatively easy conversion from modeling to actual code.

The system classes are represented as boxes divided into three compartments. The top compartment simply contains the name of the class. The middle contains variables in a form that describes their names and type. The bottom compartment contains the class’ operations (functions), these are represented with an intuitive name and input parameters. This way, if the class, variables and functions are given intuitive names, one can easily identify and understand the role a given class will play in the system.

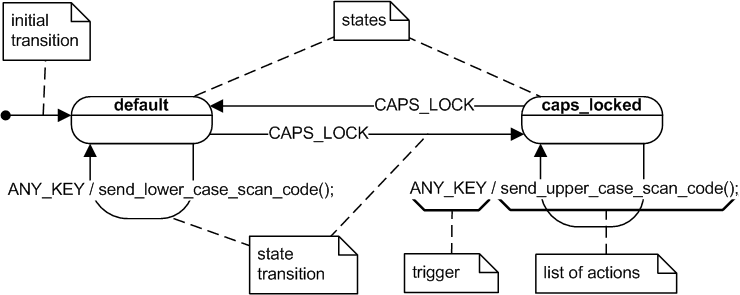
Furthermore, these classes are then grouped together in a class diagram that describes the static relationship between them. The class-boxes are connected with lines, indicating which external classes they interact with and in what fashion (one-to-one, one-to-many, etc.). An example of a class diagram can be seen below.



#### State Diagram

The UML state diagrams are dynamic diagrams that aim to illustrate the various states of a software system and what events will lead to a specific state. State diagrams are built up by nodes denoting states and edges (connectors) denoting state transitions. The states are represented as rounded rectangles labeled with the relevant state name. The state transitions are represented as arrows, labeled with the triggering event. Note that a state diagram should be cyclic so that the system won’t be stuck in a state that it’s impossible to transition from. These diagrams provide a graphical abstraction that may prevent programming errors, as event-based state transition often leads to.

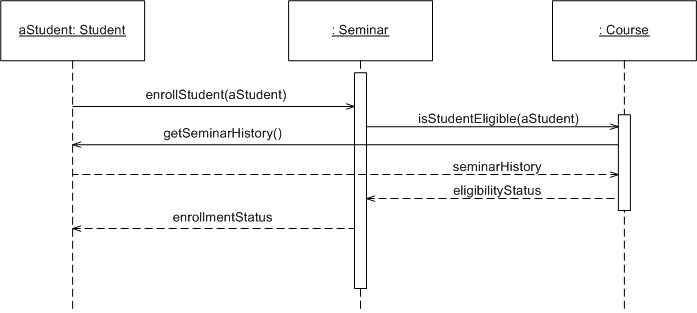
Below, an example of a state diagram showing the states of a keyboard is shown.



#### Sequence Diagram

Sequence diagrams are dynamic diagrams showing object interactions, sequenced by time in a top-to-bottom fashion. This provides a graphical illustration of how the messages (queries) flow between the objects when an event happens.

Parallel vertical lines (called lifelines) illustrates objects and horizontal arrows illustrates the messages being transmitted between them, in the order that they occur. When an event occurs, a query may be asked from the leftmost lifeline (object) to the lifeline to its right. This process continues rightwards until the response to the query has been found. This could be updating a table in the database or a request for information. Thereafter, the object producing the response will send the information to the left, back to the original querying object. This message can contain a simple boolean specifying that a table was updated successfully or specific information. Below, a sequence diagram describing a call for enrolling a student in a class is shown.



## 2.3 Applied tools

We will use Google’s Firebase for our back-end database. Firebase is an easy-to-use mobile and web application database, built on Google infrastructure. This allows applications to scale automatically, in addition to provide lots of support and analytic tools. More about this in chapter 10.

Android Studio will be used for the main app development. This is an Java-based IDE for developing android applications, made by JetBrains. It supports several useful features like integrated Android environments for testing, drag-and-drop features for UI-design and Gradle-based build support.

We will use the Node.js platform to provide a framework for our application. Node.js is an event-based asynchronous I/O framework built on Google’s V8 JavaScript Engine. This way, we can make use of running JavaScript on both the client and server side and that way benefit from the lack of context switching, in addition to non-blocking and event-driven queries. Our server only needs to perform GET or POST calls when an event occurs.

To provide concurrent documentation of the development process, we have mainly used Google Docs. This text-editor allows all the team members to work on a single document simultaneously.

In addition, Microsoft’s PowerPoint will be used for presenting our system. PowerPoint is a very easy-to-use software for creating presentation slides and graphs.

## 2.4 Project scope

The system mainly consists of two parts: backend database server and the frontend graphical user interface (GUI). The backend runs algorithms to classify users based on their likes and recommend liked items from other users of the same group. The application’s GUI will consist of four main pages; the login page, the account page, the discovery page and the recommendations page.

After login, the user will be presented to the three main functional pages. The account page, the discovery page, and the recommendation page. The account page will contain functionality for changing the password, gender, age and country of residence, in addition to an option to link accounts (if the user has logged in previously through another authentication method).

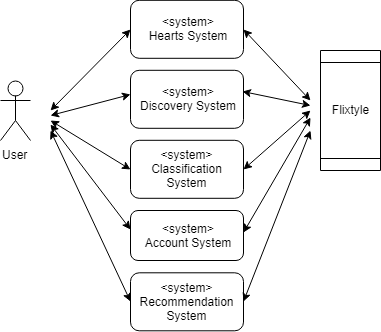
# 

# 3. System Architecture

## 3.1 Objectives

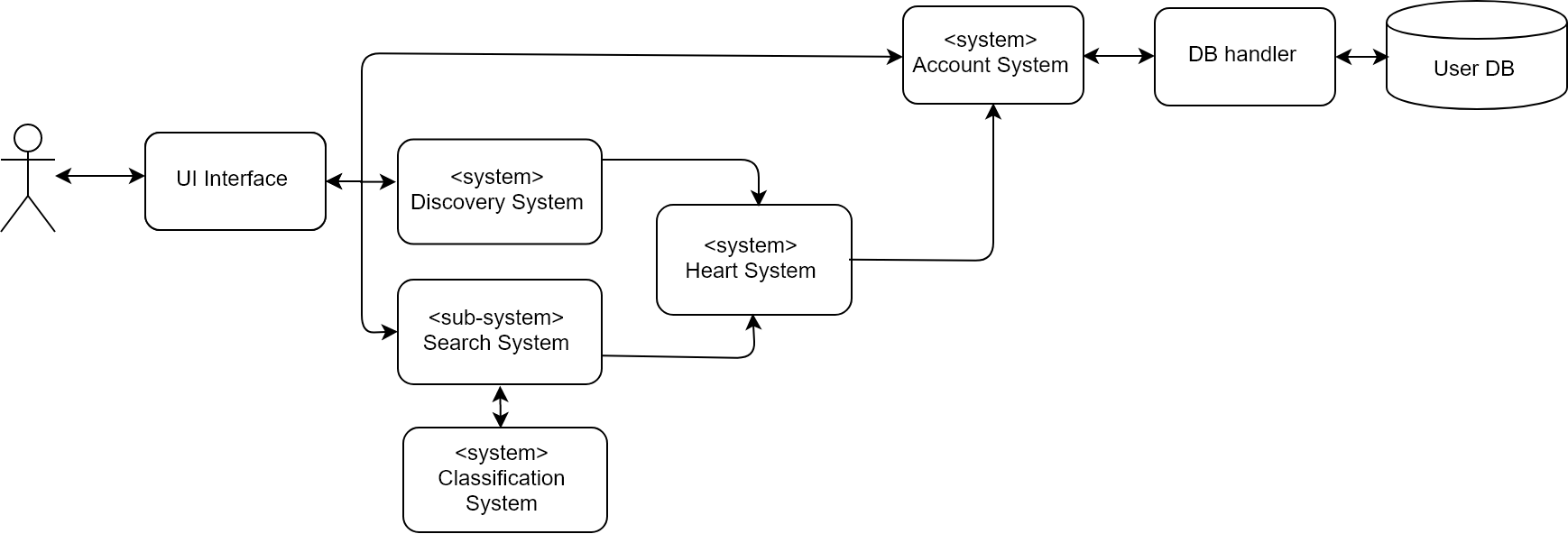
System Architecture will provide a general description of the systems that our team wants to develop. In addition, the overall structure of each system is represented as a block diagram. Each relationship and how it is actually used is described as a Package Diagram and Deployment Diagram.

## 3.2 System Organization



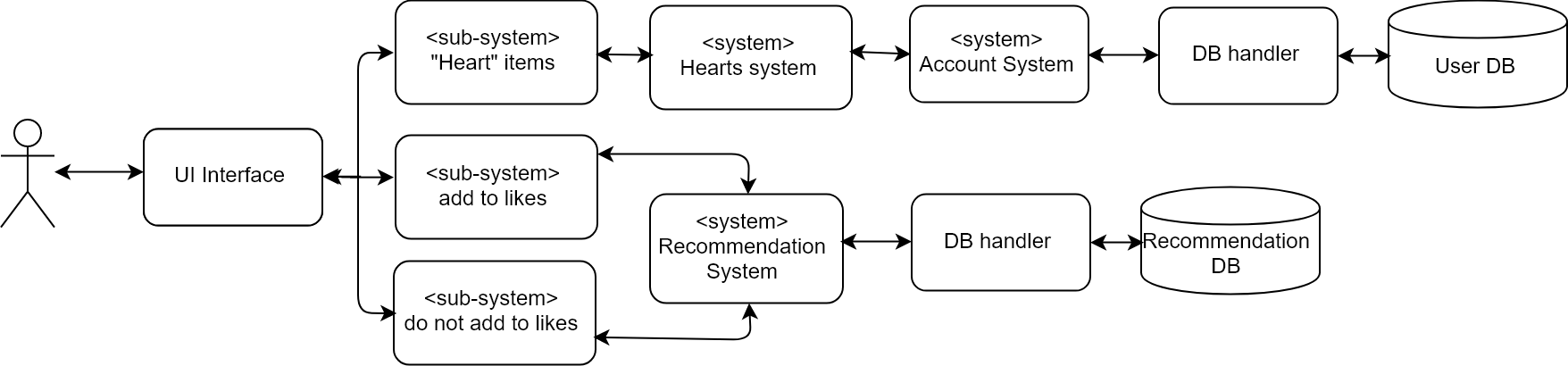
To provide the Flixtyle service, it has five systems overall. Each system stores and updates information in a database using Firebase and recalls data from the database by the request of the user or system.

### 3.2.1 Hearts System



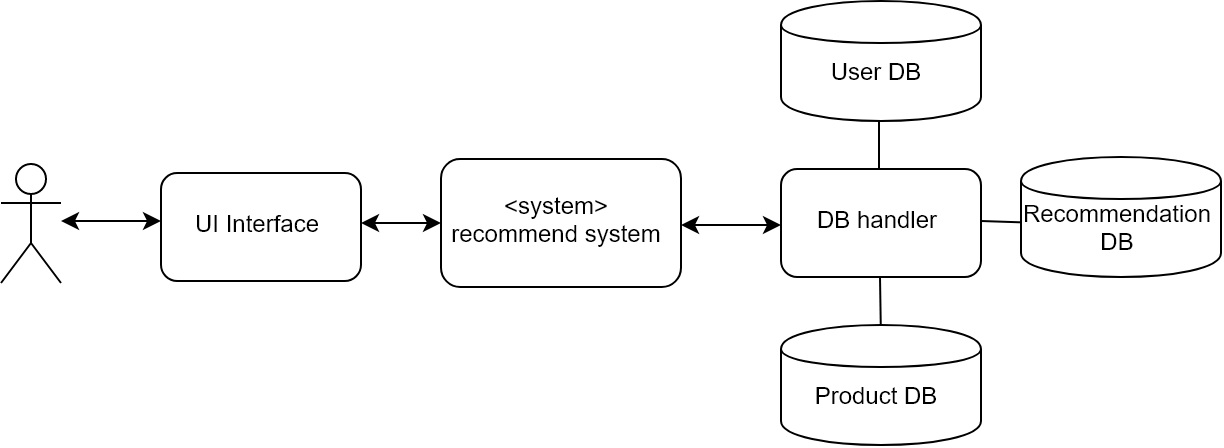
Users can heart their favorite items through the discovery and search process. These favorite items are stored in the user database and user can be checked in the user personal account by the Account system. Moreover, Users can manage items from the heart list by adding or deleting them.

### 3.2.2 Discovery System



In the Discovery System, user can represent their preferred items through adding or not adding to likes. These items are stored in the recommendation database and used to create the recommendation item list. Additionally, the Heart system allows users to store their favorite items in the heart list.

### 3.2.3 Recommendation System



Recommendation system makes a recommendation item list by using the Discovery System to select the preferred item for users and saving these items in Recommendation database. Items of recommendation depends on the actual data from the user.

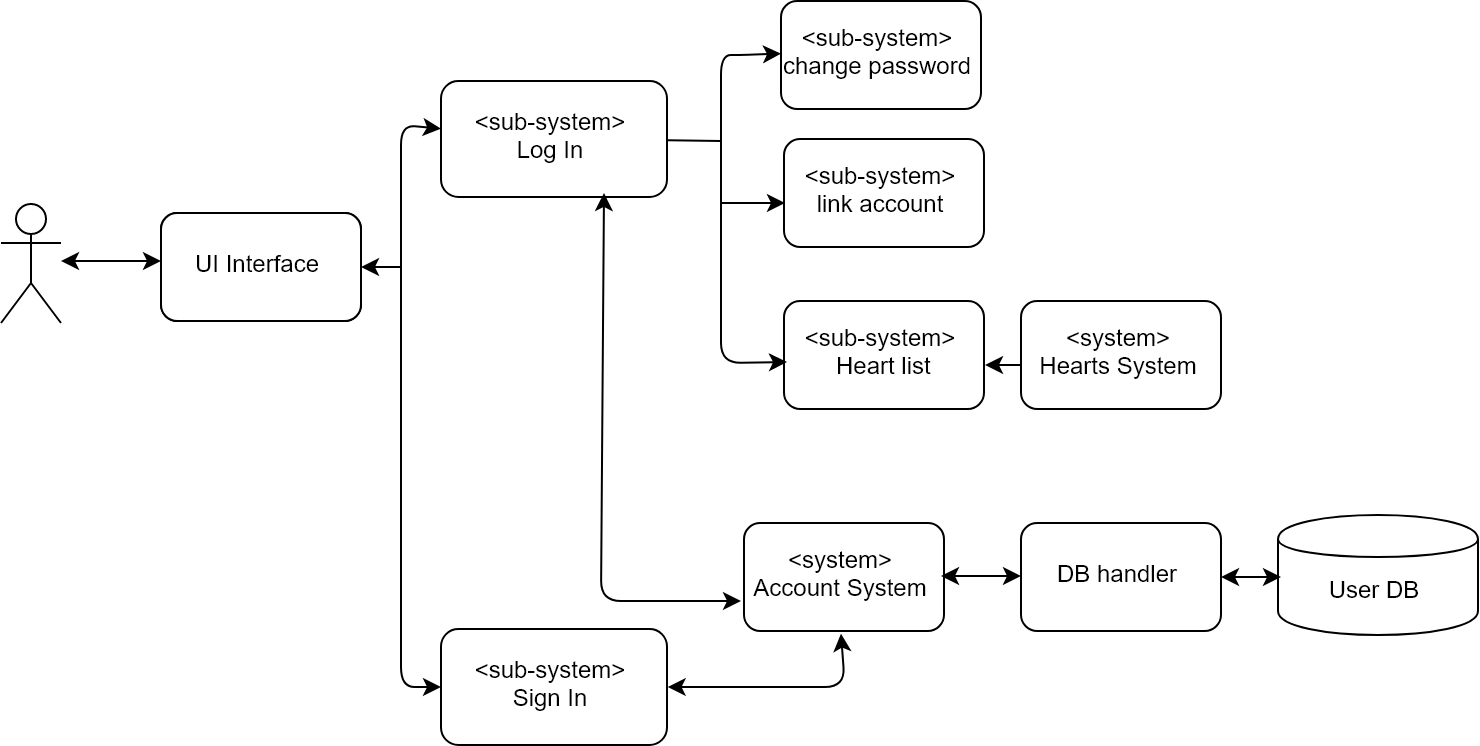
### 

### 3.2.4 Classification System



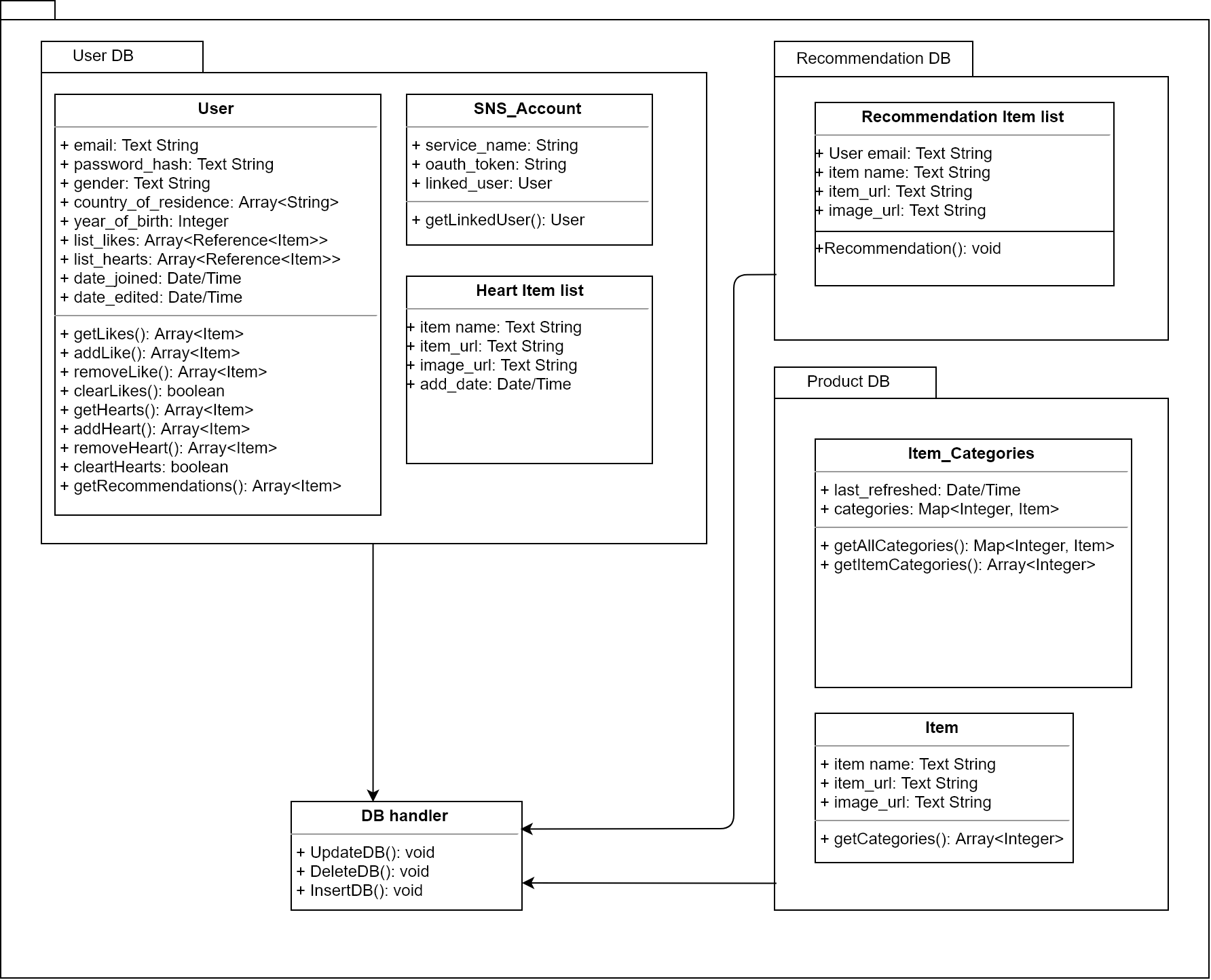
Classification system sort categories according to the type of items and store them in the database. This classification system not only makes programs easy to manage items but also users can efficiently search items.

### 3.2.5 Account Management System

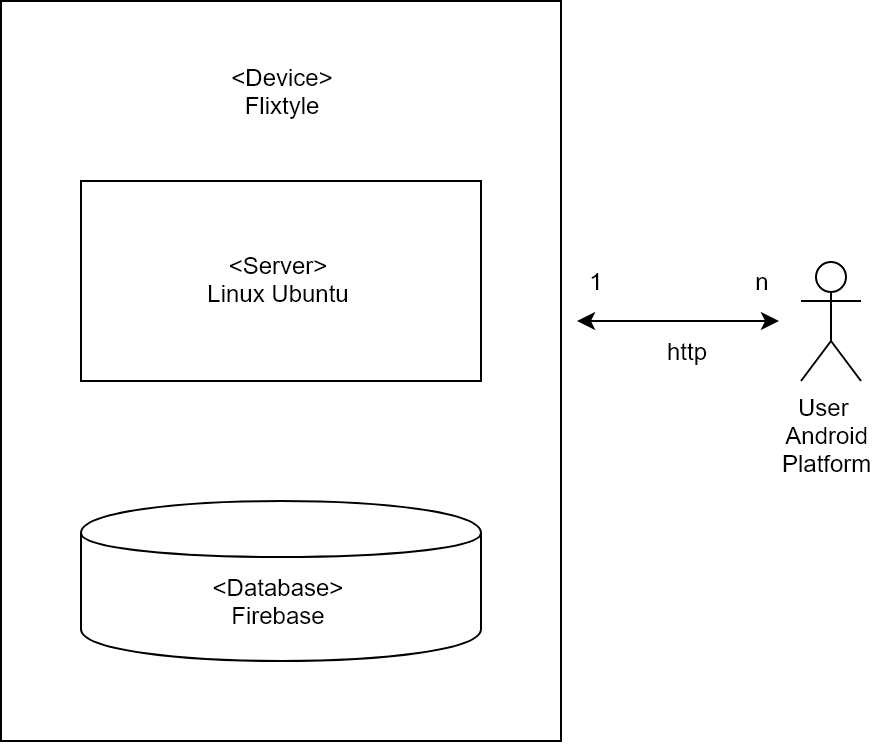


Account Management system compares the user values to the Account database, and the user can enter the user account if it matches. If the user doesn't have an account, the user can sign up and Account database saves the user profile that the system needs by default. Additionally, the Account Management System allows users to freely change user profiles or check user’s Heart list.

## 3.3 Package diagram



## 3.4 Deployment diagram

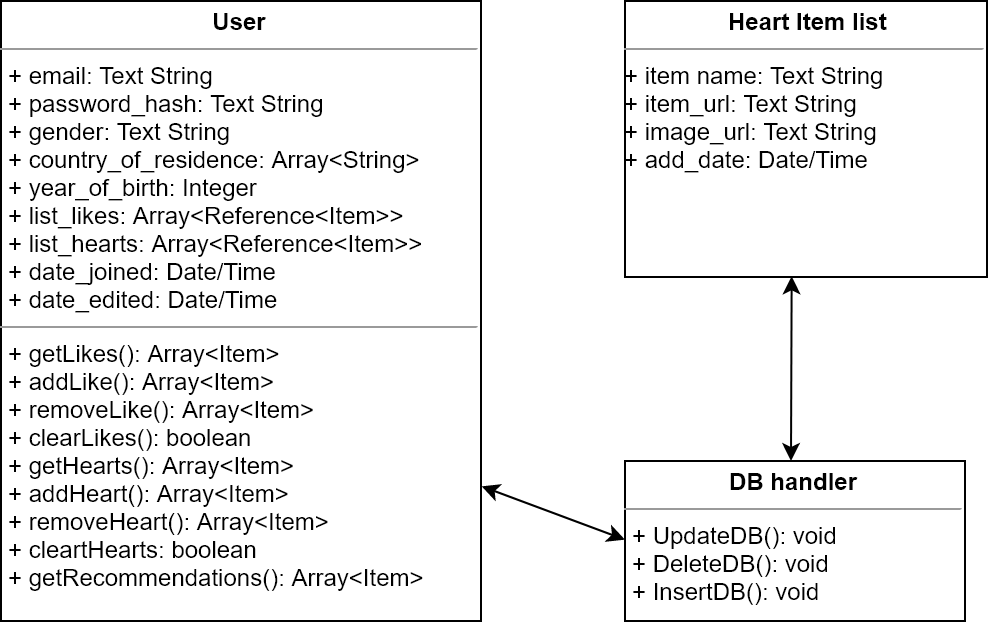


# 4. Hearts System

## 4.1 Objective

This application presents various products to users. Let's call Hearts what the user has expressed his preference for the product. Here we describe the Hearts System, which adds or manages the list of Hearts.

## 4.2 Class Diagram



### 

### 4.2.1 User

#### 4.2.1.1 Attributes

|  |  |
| --- | --- |
| list\_hearts | List of items the user has “Hearted”, contains a reference to the item object and also the date of when the user “Hearted” the item. |

#### 4.2.1.2 Methods

|  |  |
| --- | --- |
| getHearts() | Getter for user’s hearts. |
| addHeart() | Appends an item to the user’s “Heart” list and saves the resulting list. |
| removeHeart() | Pops an item from the user’s “Heart” list and saves resulting list. |
| clearHearts() | Sets the user’s “Heart” list to an empty array. |

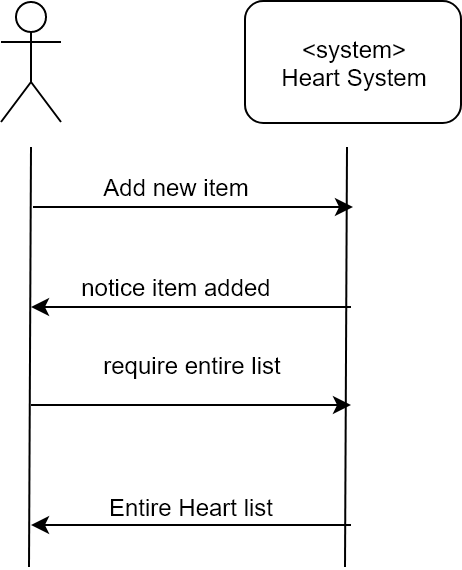
### 4.2.2 DB handler

#### 4.2.2.1 Methods

|  |  |
| --- | --- |
| updateDB() | When a user chooses a new ‘heart’ item, add that to heart list. |
| DeleteDB() | Delete heart item from the list |
| insertDB() | When a new account is created, make a new Heart list for the new user. |

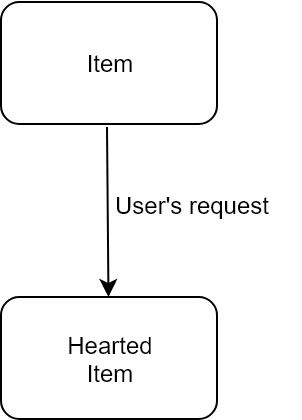
## 4.3 Sequence Diagram

#### 4.3.1 User



## 4.4 State Diagram

#### 4.4.1 Items

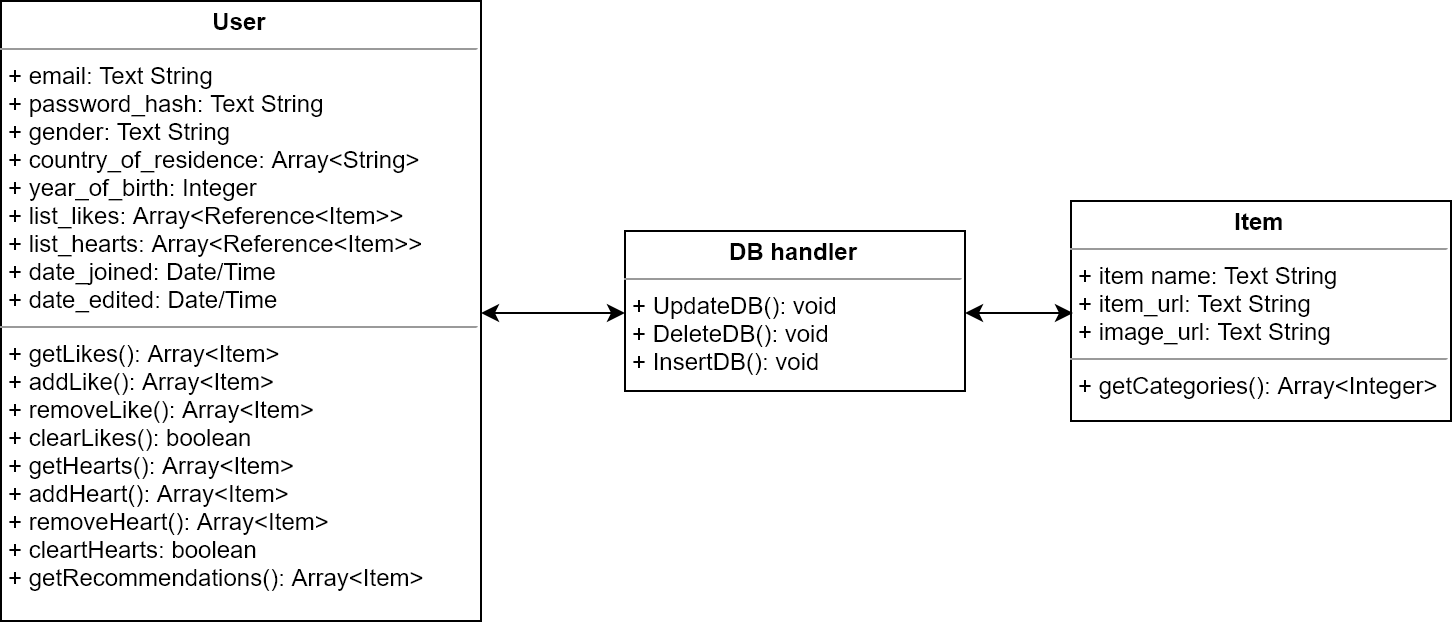


# 5. Discovery System

## 5.1 Objective

Present various products to users. The user can enjoy efficient shopping through Recommended products.

## 5.2 Class Diagram



### 5.2.1 User

#### 5.2.1.1 Attributes

|  |  |
| --- | --- |
| email | User’s email. In the form of (.^+)@(.^+)\.(.^+) |
| password\_hash | Hashed and salted user password for authentication |
| gender | Users preferred gender for clothes. Male/Female/Both |
| country\_of\_residence | Users country of residence. |
| year\_of\_birth | User’s age. Greater than 1900 and lower than the current year. |
| list\_likes | List of items the user has liked, contains a reference to the item object and also the date of when the user liked the item. |
| list\_hearts | List of items the user has “Hearted”, contains a reference to the item object and also the date of when the user “Hearted” the item. |
| date\_joined | Datetime of when the user’s account was created. |
| date\_edited | Datetime of when the user’s account was last edited. |

#### 5.2.1.2 Methods

|  |  |
| --- | --- |
| getLikes() | Getter for user’s likes. |
| addLike() | Appends an item to the user’s likes and saves the resulting list. |
| removeLike() | Pops an item from the user’s likes and saves the resulting list. |
| clearLikes() | Sets the user’s like list to an empty array. |
| getHearts() | Getter for user’s hearts. |
| addHeart() | Appends an item to the user’s “Heart” list and saves the resulting list. |
| removeHeart() | Pops an item from the user’s likes and saves the resulting list. |
| clearHearts() | Sets the user’s “Heart” list to an empty array. |
| getRecommendations() | Returns a list of recommendations based on the user’s likes and the classification system’s classifications. |

### 5.2.2 DB handler

#### 5.2.2.1 Methods

|  |  |
| --- | --- |
| updateDB() | Update the DB for showing new items |

### 

### 5.2.3 Items

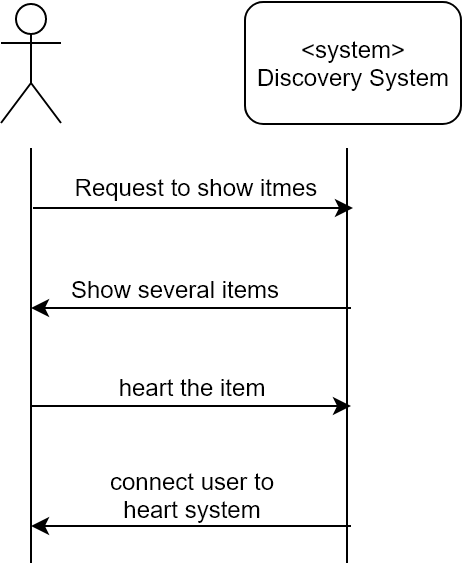
#### 5.2.3.1 Attributes

|  |  |
| --- | --- |
| item\_name | The Unique name of the item |
| item\_url | Shopping Mall that selling the item’s url. |
| image\_url | The item’s image which describes the item’s shape. |

#### 5.2.3.2 Methods

|  |  |
| --- | --- |
| getCategories() | Get the item’s category from the database. |

## 5.3 Sequence Diagram



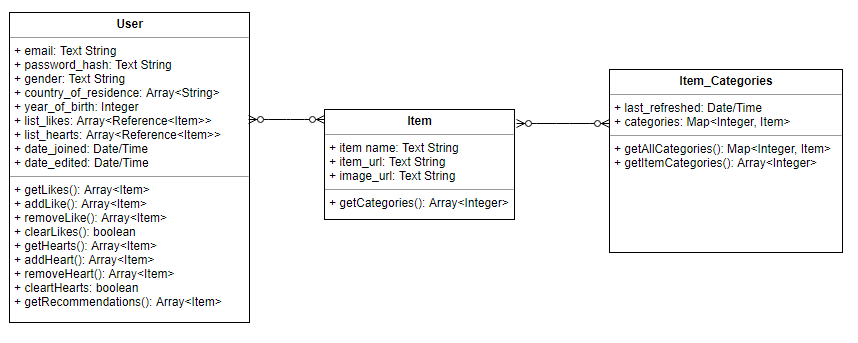
# 

# 6. Recommendation System

## 6.1 Objective

The user will interact with this system when on the main page of the application. This chapter describes the function of the recommendation system through class- and sequence diagrams.

## 6.2 Class Diagram



### 6.2.1 User

#### 6.2.1.1 Attributes

|  |  |
| --- | --- |
| email | User’s email. In the form of (.^+)@(.^+)\.(.^+) |
| password\_hash | Hashed and salted user password for authentication |
| gender | Users preferred gender for clothes. Male/Female/Both |
| country\_of\_residence | Users country of residence. |
| year\_of\_birth | User’s age. Greater than 1900 and lower than the current year. |
| list\_likes | List of items the user has liked, contains a reference to the item object and also the date of when the user liked the item. |
| list\_hearts | List of items the user has “Hearted”, contains a reference to the item object and also the date of when the user “Hearted” the item. |
| date\_joined | Datetime of when the user’s account was created. |
| date\_edited | Datetime of when the user’s account was last edited. |

#### 6.2.1.2 Methods

|  |  |
| --- | --- |
| getLikes() | Getter for user’s likes. |
| addLike() | Appends an item to the user’s likes and saves the resulting list. |
| removeLike() | Pops an item from the user’s likes and saves the resulting list. |
| clearLikes() | Sets the user’s like list to an empty array. |
| getHearts() | Getter for user’s hearts. |
| addHeart() | Appends an item to the user’s “Heart” list and saves the resulting list. |
| removeHeart() | Pops an item from the user’s likes and saves the resulting list. |
| clearHearts() | Sets the user’s “Heart” list to an empty array. |
| getRecommendations() | Returns a list of recommendations based on the user’s likes and the classification system’s classifications. |

### 6.2.2 Item

#### 6.2.2.1 Attributes

|  |  |
| --- | --- |
| item\_name | Name of an item |
| item\_url | URL where the item was scraped from. |
| image\_url | URL of the item’s image. |

#### 6.2.2.2 Methods

|  |  |
| --- | --- |
| getCategories() | Returns the item’s classified categories. |

### 6.2.3 Item Categories

#### 6.2.3.1 Attributes

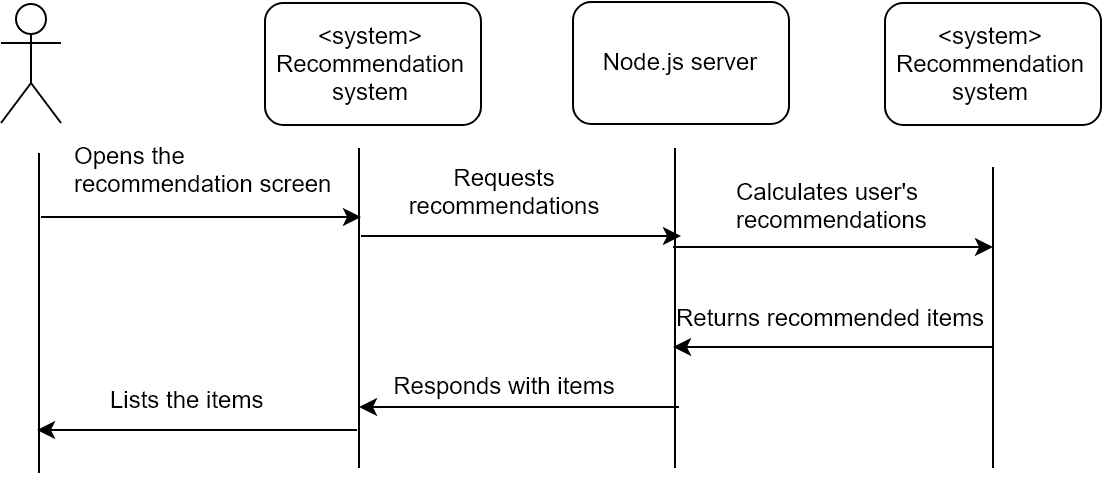
|  |  |
| --- | --- |
| last\_refreshed | Name of an item |
| categories | URL where the item was scraped from. |

#### 6.2.3.2 Methods

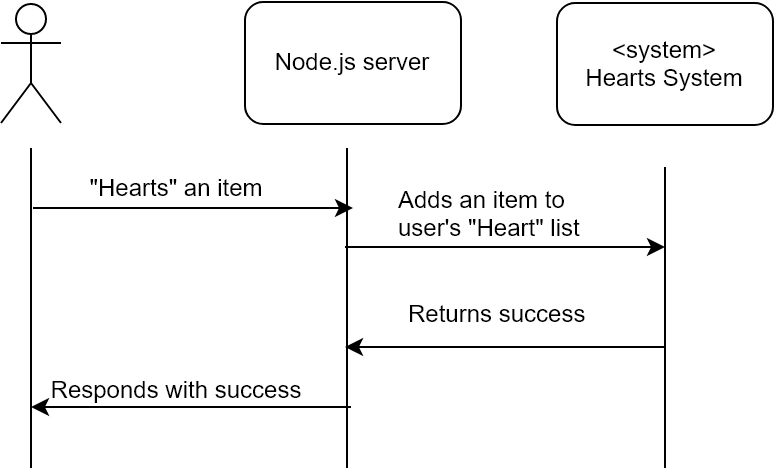
|  |  |
| --- | --- |
| getAllCategories() | Returns all categories and the items. |
| getItemCategories() | Returns a specific item’s categories. |

## 6.3 Sequence Diagram

### 6.3.1 Show recommendations function



### 6.3.2 Add to “Hearts” list function

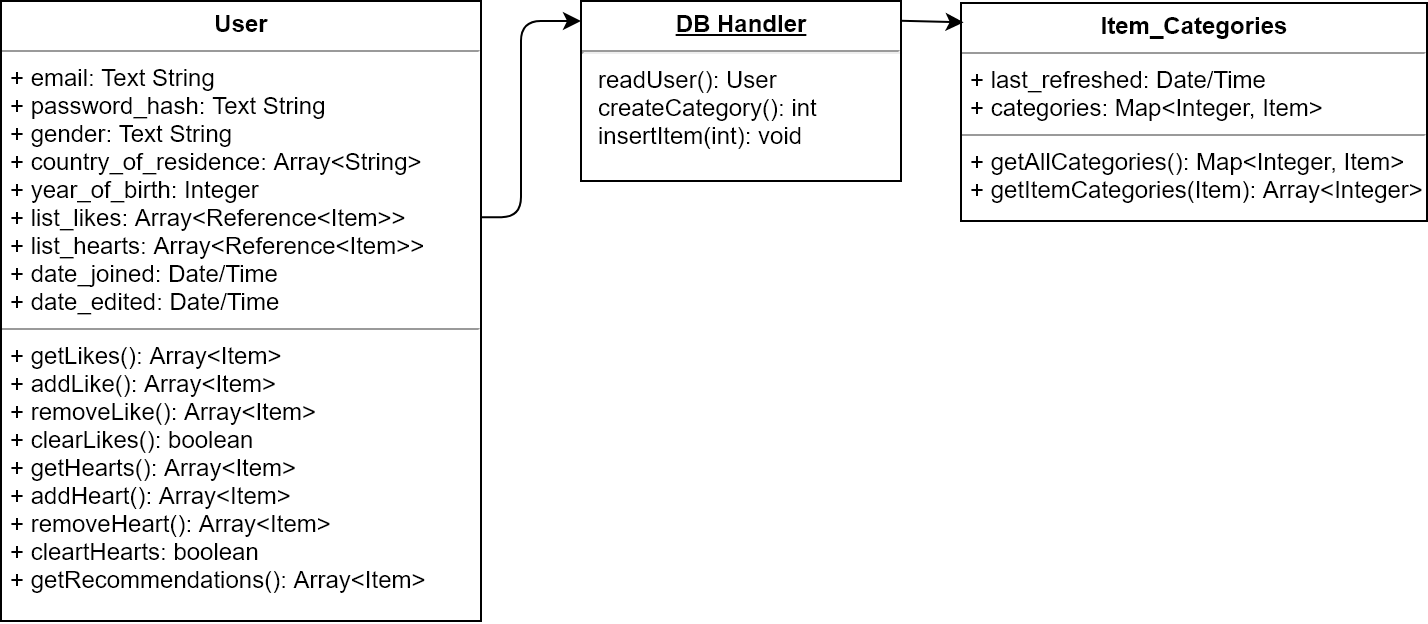


# 7. Classification System

## 7.1 Objective

The system provides its recommendations to the user by the item categories generated by the classification system. The system uses the data from the Discovery system to generate and determine categories of each item.

## 7.2 Class Diagram



### 7.2.1 User

#### 7.2.2.1 Attributes

|  |  |
| --- | --- |
| list\_likes | An array of the references to the item the user has liked. Contains the reference to an item and a timestamp on when the user has performed the action. |

#### 7.2.2.2 Methods

|  |  |
| --- | --- |
| getLikes() | Returns an array of the items the user has liked. Unlike attribute list\_likes, the method returns the actual data of the items in the list, not the references. |

### 7.2.2 DB Handler

#### 7.2.2.1 Methods

|  |  |
| --- | --- |
| readUser() | Reads a user from the User database. A User object is returned. |
| createCategory() | Creates an item category in the Item\_Categories database. An integer representing the new category is returned. |
| insertItem() | Inserts an item to an item category. Does not return any values. |

### 7.2.3 Item\_Categories

#### 7.2.2.1 Attributes

|  |  |
| --- | --- |
| last\_refreshed | The timestamp on the completion of the last refresh. |
| categories | A map of each category and the items in an array. The key is an integer, and the value is the array of Items. |

#### 7.2.2.1 Methods

|  |  |
| --- | --- |
| getAllCategories() | Returns the attribute categories. Represents a dump of the processed item categorization result. |
| getItemCategories() | Returns the list of categories the item belongs in. An Item is accepted as the argument. |
| insertItem() | Inserts an item to an item category. Does not return any values. |

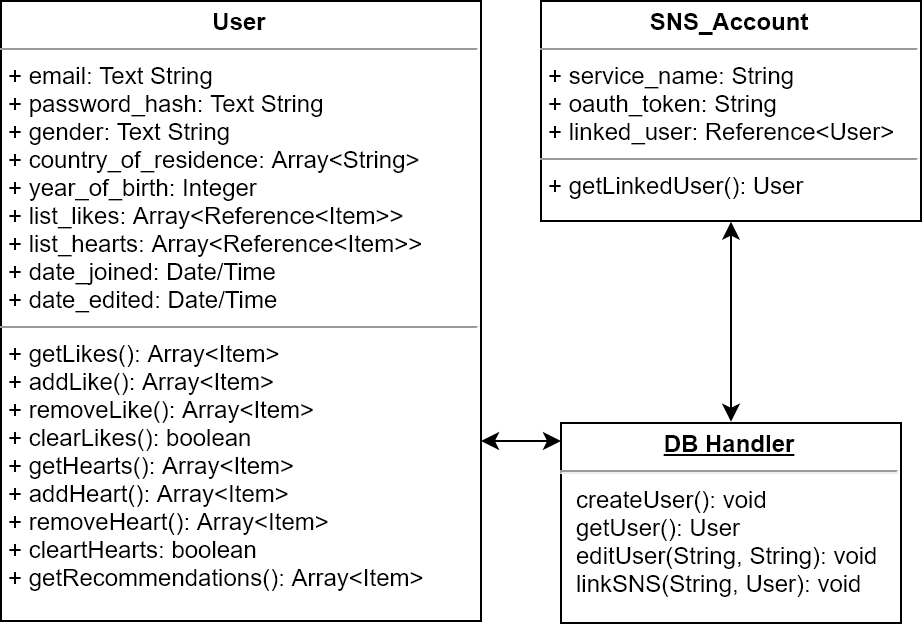
# 

# 8. Account Management System

## 8.1 Objective

The goal of the account management system is to provide an intuitive way for the user to log in to our service. Also, it is an integral part of the system as it manages the personalized data for each user which makes it possible for the recommendation system to work.

## 8.2 Class Diagram



### 8.2.1 User

#### 8.2.1.1 Attributes

|  |  |
| --- | --- |
| email | The email the user uses to log in. |
| password\_hash | Password processed through salting and hashing |
| gender | Optional information from the user about their preference for clothes by gender |
| country\_of\_residence | Optional information from the user about the country the user lives in in order to filter out countries that do not offer to ship to the user’s country |
| year\_of\_birth | Optional information from the user about the year of birth of the user, which is used to make better recommendations to the user by age groups. |
| list\_likes | An array of the references to the item the user has liked. Contains the reference to an item and a timestamp on when the user has performed the action. |
| list\_hearts | An array of the references to the item the user has added to the “Heart” list. Contains the reference to an item and a timestamp on when the user has performed the action. |
| date\_joined | The timestamp of the account creation date. |
| date\_edited | The timestamp of the last edit on the account information (excluding list\_likes and list\_hearts) |

#### 8.2.1.2 Methods

|  |  |
| --- | --- |
| getLikes() | Returns an array of the items the user has liked. |
| addLike() | Adds an item to the list of liked items. Returns the updated list of liked items, or null when failed. |
| removeLike() | Removes an item from the list of liked items. Returns the updated list of liked items, or null when failed. |
| clearLikes() | Removes all items from the list of liked items. Returns a boolean indicating whether the operation was successful or not (true is success). |
| getHearts() | Returns an array of the items the user has added to the Heart list. |
| addHeart() | Adds an item to the Heart list.  Returns the updated Heart list, or null when failed. |
| removeHeart() | Removes an item from the Heart list. Returns the updated Heart list, or null when failed. |
| clearHearts() | Removes all items from the Heart list. Returns a boolean indicating whether the operation was successful or not (true is success). |
| getRecommendations() | Returns an array of the items recommended to the user. |

### 8.2.2 DB Handler

#### 8.2.2.1 Methods

|  |  |
| --- | --- |
| createUser() | Creates a new entry of a user in the database User. |
| getUser() | Retrieves a user from the database User. Returns an instance of class User |
| editUser() | Modifies an attribute of a user. Receives two strings as the arguments. The first is the attribute name, and the second is the new value. |
| linkSNS() | Creates a new entry on database SNS\_Account, and links it with a User entry. Receives a String and a User as the arguments. The string is the SNS type and token, and the User is the account the SNS account should be linked to. |

### 8.2.3 SNS\_Account

#### 8.2.3.1 Attributes

|  |  |
| --- | --- |
| service\_name | A string representing the type of the SNS service. |
| oauth\_token | A string of the token the linked SNS service has provided. |
| linked\_user | A reference to the User entry that the SNS account is linked to. |

#### 8.2.3.2 Methods

|  |  |
| --- | --- |
| getLinkedUser() | Returns a User instance of the linked user. |

# 

# 9. API Design

## 9.1 Objective

The objective is to describe the interfaces between the different systems and components. The interfaces are going to be implemented following the RESTful API standard. RESTful is a stateless API that uses JSON to communicate. Javascript Object Notation(JSON) is a human readable text format that a standard in the industry. JSON can be parsed and edited by many programming languages out of the box without needing to add any external libraries.

## 9.2 Variables

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Explanation** |
| email | Text String | The user’s email |
| password\_hash | Text String | The user’s password |
| gender | Text String | The user’s gender |
| country\_of\_residence | Array<String> | The user’s country of residence |
| year\_of\_birth | Integer | The user’s year of birth |
| list\_likes | Array<Reference<Item>> | The user’s list of likes |
| list\_hearts | Array<Reference<Item>> | The user’s list of hearts |
| date\_joined | Data/Time | The date the user joined |
| item\_name | Text String | Name of the item in the website |
| item\_url | Text String | URL of the item in the website |

## 9.3 Request and Responses

### 9.3.1 Request item thumbnail

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| GET | /items/{item\_id}/thumbnail | item\_id | Image file(.jpg, .png) |

### 9.3.2 Request item website URL

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| GET | /items/{item\_id}/viewpage | item\_id | item\_url |

### 9.3.3 Create account request

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| POST | /accounts/create | email  password  gender  country\_of\_residence  year\_of\_birth | <Success> with the session ID |

### 9.3.4 View account request

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| GET | /accounts/view | Session | email  gender  country\_of\_residence  year\_of\_birth  date\_joined |

### 9.3.5 Update account information request

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| POST | /accounts/patch | email  password  gender  country\_of\_residence  year\_of\_birth | <Success> |

### 

### 9.3.6 Delete account request

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| POST | /accounts/delete | Session | <Success> |

### 9.3.7 View “Hearts” list request

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| GET | /hearts | Session | Array<Item> |

### 9.3.8 Add item to “Hearts” list request

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| POST | /hearts/add | item\_id | <Success> |

### 9.3.9 Remove item from “Hearts” list request

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| POST | /hearts/remove | item\_id | <Success> |

### 9.3.10 View recommendations request

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| GET | /recommendations | Session ID | Array<Item> |

### 9.3.11 Request Discoverable Items

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| GET | /discovery/getitems | Item count: Integer | Array<Item> |

### 9.3.12 Send item liked request

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **URL** | **Value** | **Response** |
| POST | /discovery/sendlike | item\_id | <Success> |

# 10. Database Design

## 10.1 Objective

This section shows database diagram that the project will be based on. Firebase (a Google Cloud product) will be used as the database of the application.

## 10.2 Data types

Cloud Firestore is a NoSQL Database. Cloud firestore stores data in documents organized in collections. The following table lists the data types supported by Cloud Firestore that are going to be used in the project.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Sort Order** | **Explanation** |
| Array | By element values | An array cannot contain another array value as one of its elements.  Within an array, elements maintain the position assigned to them. When sorting two or more arrays, arrays are ordered based on their element values.  When comparing two arrays, the first elements of each array are compared. If the first elements are equal, then the second elements are compared and so on until a difference is found. If an array runs out of elements to compare but is equal up to that point, then the shorter array is ordered before the longer array. |
| Date/time | Chronological | When stored in Cloud Firestore, precise only to microseconds; any additional precision is rounded down. |
| Integer | Numeric | 64-bit, signed |
| Text String | UTF-8 encoded byte order | Up to 1,048,487 bytes (1 MiB - 89 bytes). Only the first 1,500 bytes of the UTF-8 representation are considered by queries. |

## 10.3 Table

This section describes the fields per table that make up the database.

### 10.3.1 User

|  |  |
| --- | --- |
| **Name** | **Type** |
| email | Text String |
| password\_hash | Text String |
| gender | Text String |
| country\_of\_residence | Array<String> |
| year\_of\_birth | Integer |
| list\_likes | Array<Reference<Item>> |
| list\_hearts | Array<Reference<Item>> |
| date\_joined | Data/Time |
| date\_edited | Date/Time |

### 10.3.2 SNS Account

|  |  |
| --- | --- |
| **Name** | **Type** |
| service\_name | String |
| oauth\_token | String |
| linked\_user | User |

### 10.3.3 Item

|  |  |
| --- | --- |
| **Name** | **Type** |
| item\_name | Text String |
| item\_url | Text String |
| image\_url | Text String |

### 10.3.4 Item Categories

|  |  |
| --- | --- |
| Name | Type |
| last\_refreshed | Date/Time |
| categories | Map<Integer,Item> |

# 11. Testing Plan

## 11.1 Objectives

This section establishes a preliminary plan for system testing to determine if the system is operating as intended to identify and analyze the system for defects after completion of the system. This part is divided into testing policy, which sets the criteria for testing and test case, which gives a set of more detailed input and output.

## 11.2 Testing Policy

### 11.2.1 Development Testing

It is the concept of applying testing practices consistently throughout the software development life cycle model. This method of testing ensures detection of bugs at the right time which further ensures avoidance of any kind of risk in terms of time and cost. It basically refers to testing performed during the development process. The main goal of development testing is to evaluate the system's compliance with the specified needs. Depending on what you are testing, you will be divided into different testing levels which help to check behavior and performance for development testing. These testing levels are designed to recognize missing areas and reconciliation between the development lifecycle states. The main four testing levels are:

#### 11.2.1.1 Component Testing

A Component is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately.

The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. This kind of testing is performed by developers.

#### 11.2.1.2 Integration Testing

Integration means combining. For example, In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing.

Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

#### 11.2.1.3 System Testing

System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

#### 11.2.1.4 Acceptance Testing

Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

### 11.2.2 Release Testing

Test the final system before release to the user. Ensure that the requirements written in the requirements specification are properly reflected.

### 11.2.3 User Testing

The user tests the system in the user's environment.

## 11.3 Test Case

### 11.3.1 User Management System

#### 11.3.1.1 Login

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| User tries logging in with ID and password | Compare with the data stored in User DB. | Login with corresponding ID. | System alert, "The username or password does not match."  Returns to login screen with login and sign up button. |

#### 11.3.1.2 Sign Up

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| The user will fill out six fields: E-mail, password, password confirmation, gender, age, and country of residence. The email, password, and password confirmation fields will be mandatory. | Checks if passwords match and mandatory fields are filled out. | Stores user’s information in the DB. | System alert, "Passwords do not match." or “Fill out mandatory fields.”  Returns to sign up screen. |

#### 11.3.1.2 Logout

|  |  |
| --- | --- |
| User’s action | System’s action |
| User tries logging out. | Logout. |

### 11.3.2 Discovery System

#### 11.3.2.1 Swipe Left

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| Swiping left will indicate that the user is not interested and will show a frowny face in the back. | Gets rid off information about the item. | The user can view the item on the next card and keep swiping. | Fails to go on to the next card to view the next discovery item. |

#### 11.3.2.2 Swipe Right

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| Swiping right will indicate that the user is interested in the product and show a smiley face in the back. | Saves information that the user likes the item. Sends to recommendation page. | The user can view the item on the next card and keep swiping. | Fails to go on to the next card to view the next discovery item. |

#### 11.3.2.3 Swipe Up

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| Swiping up will indicate that the user is very interested in buying the product. | System will save item to the “Heart” List. | The user can view the item on the next card and keep swiping. | Fails to go on to the next card to view the next discovery item. |

### 

### 11.3.3 Recommendation System

#### 11.3.3.1 Base case

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| User opens the recommendation view | Sends a request to the recommendation system and returns a list of recommended items | The user can view the items in the application window and scroll up and down. These items also are similar to the items the user has liked. | The system fails to deliver the list of items to the user, be it because of connection issues or anything else. Another type of failure would be when the list doesn’t reflect the user’s likes. |

#### 11.3.3.2 Add to “Hearts” list

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| User clicks on the “Heart” icon on an item | Sends a request to the “Hearts” system and saves the item to the user’s “Hearts” list. | The system successfully updates the user’s “Hearts” list. | The system fails to update the “Hearts” list of the user. |

### 11.3.4 Classification System

#### 11.3.4.1 Process Likes to categories

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| The regular time trigger starts the system | Reads all users’ list of likes and processes it into nodes of correlated products, and output a classification. | The system saves the classification data to database Item\_Categories. | The system fails to process the users’ data due to some cause such as bugs in the program, or insufficient memory allocation. |

### 11.3.5 Account Management System

#### 11.3.5.1 Create an account by email

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| The user creates an account with an email and a password. | Create a new entry in the User database, and return the access token. | The application run by the user receives the token and successfully logs in with the newly created account. | The system fails to create a new entry in the database. |

#### 11.3.5.2 Update user’s information

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| The user enters new information about themselves and sends the data to the server by the application. | Update the new data in the appropriate account of database User, and responds to the updated data. | The account information is updated and the user receives the updated data. | The system fails to update the entry in the database. |

#### 11.3.5.3 Link an SNS Account

|  |  |  |  |
| --- | --- | --- | --- |
| User’s action | System’s action | Success | Failure |
| The user links an SNS account with the account. | The system stores the token and the SNS type and links it to an account. | A new entry is created in the database SNS\_Account and linked to an account. | The system fails to update the entry in the database or the new SNS token is faulty. |

# 

# 12. Development Plan

## 12.1 Objectives

Describes the programming language and the IDE that is going to be used.

## 12.2 Programming Language & IDE

### 12.2.1 Programming Language

#### 12.2.1.1 Java

Java is used as the development programming language. JAVA is an object-oriented language and has the advantage of running on any platform. It is widely used in Android application development language and is suitable for our project.

#### 12.2.1.2 Node.js

Node.js is used for back-end development. Using node.js requires using javascript so we will use JavaScript for server-side programming. JavaScript is an object-based scripting programming language. Node.js will handle networking.

### 12.2.2 IDE

The IDE used in the project development process is Android Studio. Android Studio is the official integrated development environment for Android and Android applications. Java and Kotlin are the programming languages ​​used in Android Studio.

## 12.3 Version Management Tool

GitHub will be used for code management and more efficient development. GitHub is a web hosting service that provides extended project management support to Git, a version management system. It is currently one of the most popular web hosting services for open source project management around the world.