|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Submission Coversheet (All Programmes) | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **Student ID Number**  *(Do not include student name as anonymous marking is implemented)* | RAD18457818  DRO18457544  ANI18457381 | | | | | | |
| **Programme Title** | COMPUTING TECHNOLOGIES | | | | | | |
| **Module Title** | Mobile App Design and Development | | | | | | |
| **Module Code** *(listed on Moodle and in LTAFP)* |  | | | | | | |
| **Module Convenor** | Tendai Mhlanga | | | | | | |
| **Coursework Title** | Barter Trader-Assignment 1 | | | | | | |
| **Academic Declaration:**  *Students are reminded that the electronic copy of their essay may be checked, at any point during their degree, with Turnitin or other plagiarism detection software for plagiarised material.* | | | | | | | |
| **Word Count** |  | | | **Date Submitted** | | **26/11/2019** | |
|  |  | | |  | |  | |
| Please save your file as:  student id\_module code\_module\_title\_assessment name  for example: COR15416549\_QAB020N592S\_Managing Organisations)\_Essay 1 Teamwork | | | | | | | |

Table of Contents

[**Project Details** 4](#_Toc24920364)

[Project Name 4](#_Toc24920365)

[Description 4](#_Toc24920366)

[**Functional Requirements** 4](#_Toc24920367)

[Anonymous User - Functional Requirements 4](#_Toc24920368)

[Registered User - Functional Requirements 4](#_Toc24920369)

[**Mobile Application Map** 6](#_Toc24920370)

[**Wireframes** 6](#_Toc24920371)

[**Use Case Diagram** 6](#_Toc24920372)

[Diagram Design 6](#_Toc24920373)

[Brief Descriptions of Use Cases 6](#_Toc24920374)

[**Application Design Model** 7](#_Toc24920375)

[Architectural Pattern 7](#_Toc24920376)

[**Class Diagram** 10](#_Toc24920377)

[Class Diagram Design 10](#_Toc24920378)

[Description of Classes 11](#_Toc24920379)

[**Firebase Design** 17](#_Toc24920380)

[Firebase’s Entity Relationship Diagram 17](#_Toc24920381)

[Authentication 18](#_Toc24920382)

[Cloud Firestore 18](#_Toc24920383)

[Users Collection 18](#_Toc24920384)

[Products Collection 18](#_Toc24920385)

[Offers Collection 18](#_Toc24920386)

[Reviews Collection 18](#_Toc24920387)

[Cloud Storage 18](#_Toc24920388)

[**Marketing & Pricing Strategy** 18](#_Toc24920389)

[**User Interface** 19](#_Toc24920390)

[Main View 19](#_Toc24920391)

[Select Category Gadgets 19](#_Toc24920392)

[Select Category Clothes 19](#_Toc24920393)

[Select Category Tools 19](#_Toc24920394)

[Select Category Bikes 19](#_Toc24920395)

[Select Category All 19](#_Toc24920396)

[View Selected Product 19](#_Toc24920397)

[Barter for Selected Product 19](#_Toc24920398)

[View Reviews on Selected Product 19](#_Toc24920399)

[View User’s products on Selected item 19](#_Toc24920400)

[View Products’ Photo 19](#_Toc24920401)

[View Products’ Video 19](#_Toc24920402)

[Main Menu Selection 19](#_Toc24920403)

[Sign in Page 19](#_Toc24920404)

[Sign in –Reset Password Page 19](#_Toc24920405)

[Sign up Page 19](#_Toc24920406)

[About Page 19](#_Toc24920407)

[Contact Page 19](#_Toc24920408)

[My Account 19](#_Toc24920409)

[Profile Page – View Profile 19](#_Toc24920410)

[Profile Page – User’s Reviews 19](#_Toc24920411)

[Products – View User’s Products 19](#_Toc24920412)

[Products – View Product 19](#_Toc24920413)

[Offers – View Received Offers 19](#_Toc24920414)

[Offers – View Offer 19](#_Toc24920415)

[Reviews – View User’s Accepted/Rejected Offers 19](#_Toc24920416)

[Reviews – View and Set Review for Accepted/Rejected Offers 19](#_Toc24920417)

[Add Product 19](#_Toc24920418)

[Add Product Page 20](#_Toc24920419)

[Add Product – Select Category 20](#_Toc24920420)

[Add Product – Select Photo 20](#_Toc24920421)

[Add Product – Select Video 20](#_Toc24920422)

[Add Product 20](#_Toc24920423)

[**Test Documentation** 20](#_Toc24920424)

[**Appendix:** 21](#_Toc24920425)

[**References:** 21](#_Toc24920426)

# **Project Details**

## Project Name

Barter Application

## Description

The Project Barter App

Mecachrome Software Ltd has employed us for creating a new application called ” Barter Trader”. The purpose of the app is to provide a platform to facilitate the barter (exchange) of goods among users who wish to barter various items like clothes, gadgets, tools, toys, bicycles etc.

# **Functional Requirements**

## Anonymous User - Functional Requirements

* Can view products by category in a grid arrangement
* Can view a product
* Can sign in, after registration
* Can read About, contact, terms and conditions of the application when the person decide to register.
* An anonymous can not register without accepting terms and conditions of the app

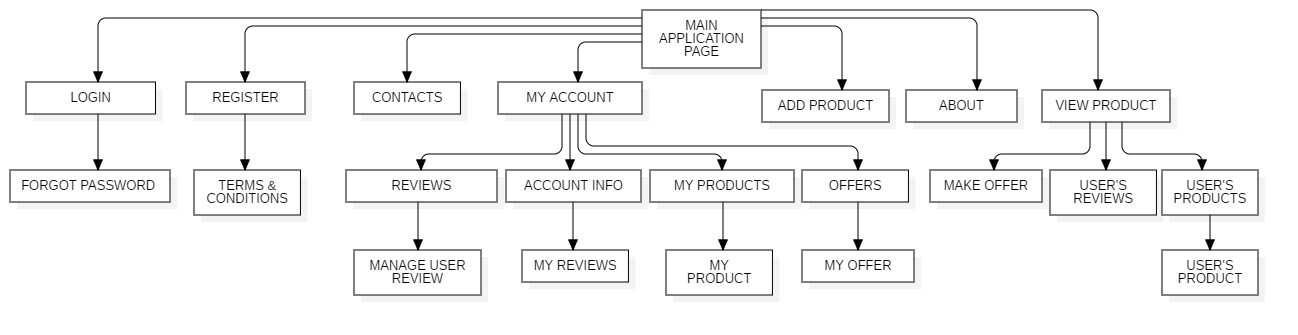
## Registered User - Functional Requirements

* Can login into a different account;
* Can register a new account;
* Can sign out
* Can recover the password
* Access His Account information:
* View Profile- including his first name, surname, telephone number, alias, email address, rating, if user is a flagged one-negative feedback and average number of the rating from other users. When rating falls below zero and the user is flagged more than 5 times, the user can no longer add new items and cannot message other users.
* View his Products-including image and title; if the user click on his product than he can see image of his product, recording, title and description
* View His Offers
* View reviews- on this page user has the option to add reviews of the user who posted the item for barter and to message the user;
* Add a product -including image, video, title, description of the item and option available to choose the right category
* Can read reviews from other users
* Can login to their accounts using their unique email address and password
* Can read About, contact, terms and conditions of the application when the person decide to register.
* Add a product including image, recording, title and description and option to choose the categories
* View the categories- From the Dashboard, when a category of item is selected (**Gadgets, Clothes, Tools, Bicycles, Others)**, the user is taken to a list of items for barter in that category
* - Categories listed for logged and anonymous users are: Gadgets, Clothes, Tools, Bicycles, Others;
* If users are selecting one of the four categories than all gadgets are displayed in a scrollable list allowing for navigability of the list;
* - The object of class Categories are dynamically read from a firebase database and then set on the activity;
* Details of each item/gadget will include: description of the item, rating of the user who posted the item, option to list all the items that this user whose item the logged user is currently viewing;
* If user is interested in the item and want to offer something in the exchange of it than user is able to message the other user with his/her offer;
* If the users agree with the barter than status of the items is changed to agreed;
* Once the item status is changed to agreed it is no longer listed in the application;

**Non-functional requirements:**

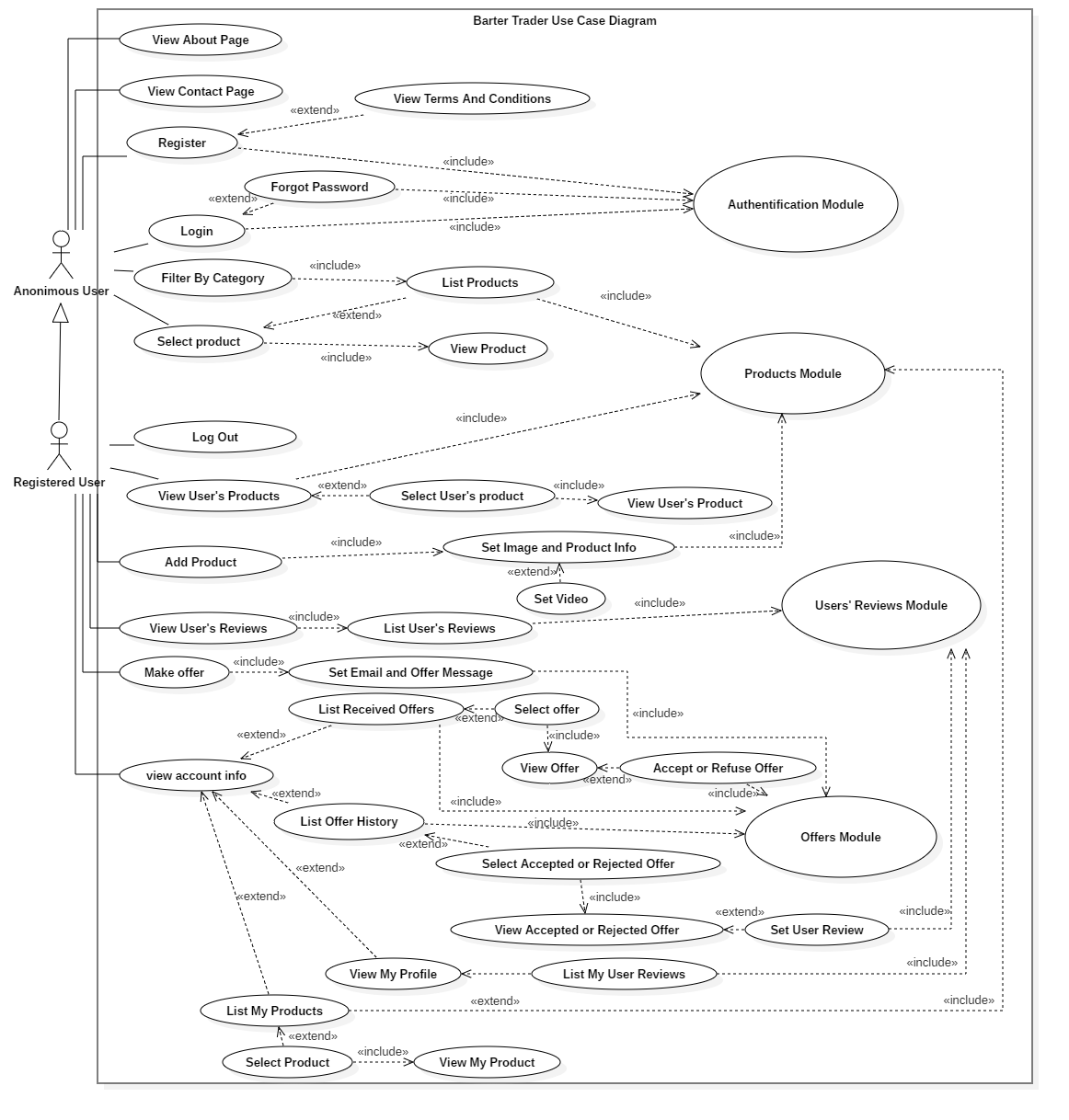
* Each category lists the same gadgets;
* When user get a call in the same time when he is using the app, then when he opens the app he should log in into his account;
* When user is adding a new product, he is acknowledged with message: “Product added successfully”
* When user open the app, it is loading within 10 seconds in such way that user doesn’t have to wait too long for app to respond

# **Mobile Application Map**



# **Wireframes**

# **Use Case Diagram**



# **Application Design Model**

## Architectural Pattern

We’ve chosen the Model-View-View Model (MVVM) architectural pattern because Google embedded this pattern into the Android’s framework so that developers can easily implement their logic into mobile applications. Beside the ease of use, it is one of the architectural patterns which enforces separation of concerns, which divides the user interface from the business logic. The advantages of this is that the code is easy to maintain by dividing the work to frontend and backend developers, it is simple to isolate, and unit test the components and least but not last the reusability and extensibility of the classes increases the implementation speed of the project.

As seen in the figure below the implementation is divided into three layers:

* Model:

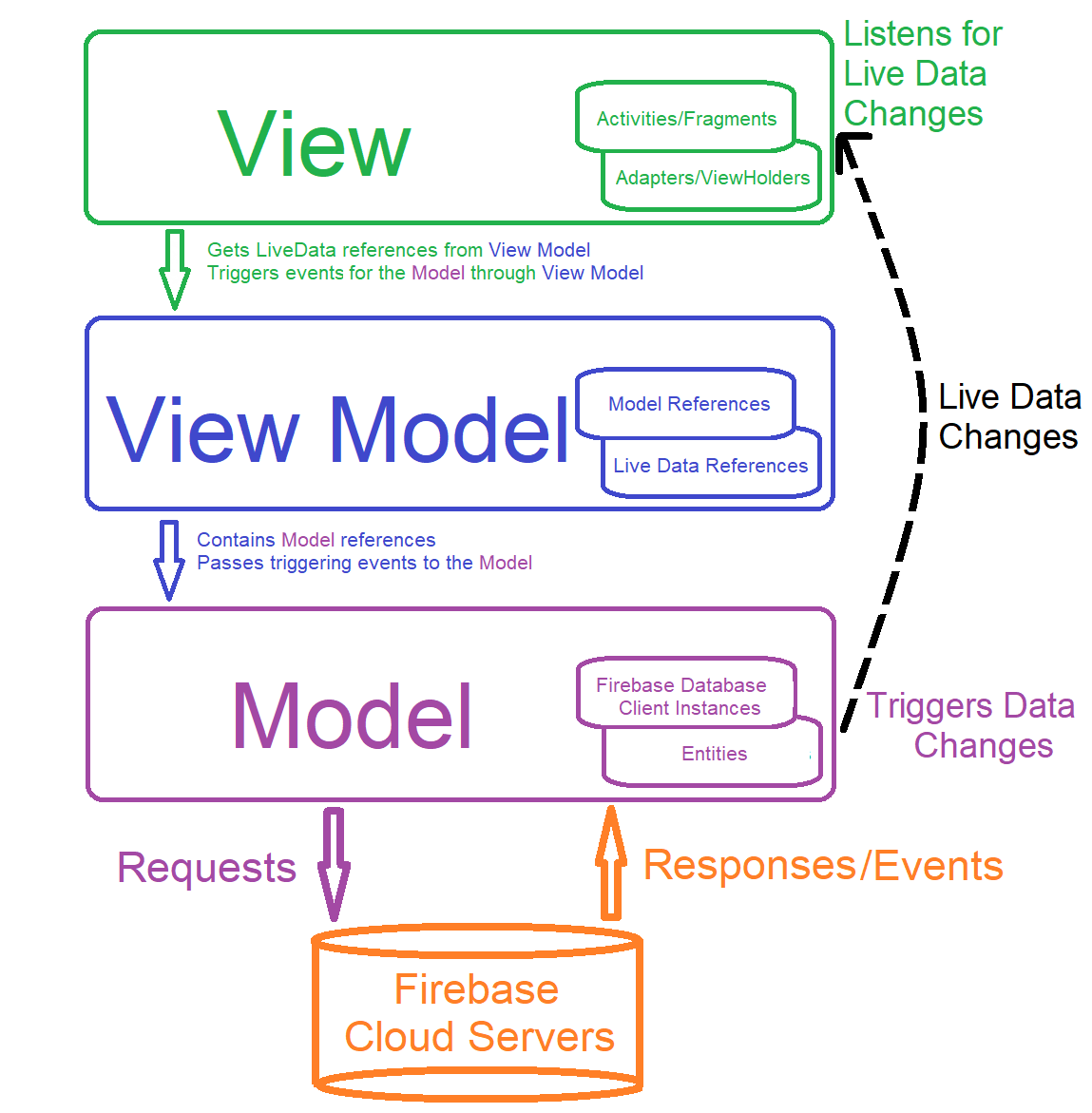
This layer represents the business logic of the application and exposes its data through the framework’s own observable data holder class, called LiveData. Beside triggering data changes, the model also contains entity classes used for structuring the data in the application and in the database. The Model also contains Firebase instances of its different clients for requesting and receiving data from the Google’s Servers.

* View Model:

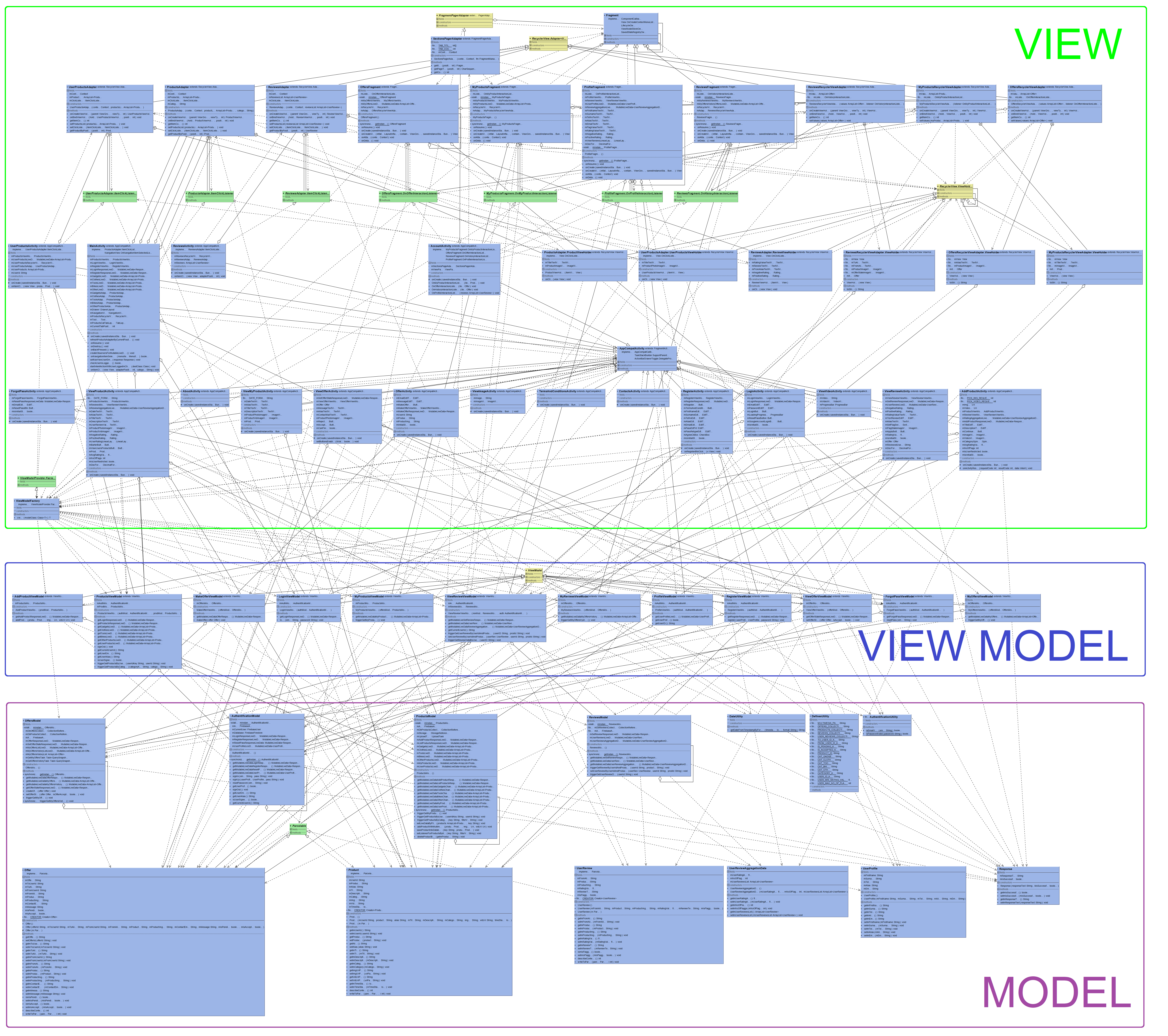
The View Model manages the Model components and provides references of the observers to the View components. Aside this, it also provides methods for passing events to the business logic. The whole idea behind the View Model is to decouple the View components from the implementation logic of the application. Android’s framework facilitates this through the ViewModel class, which is lifecycle aware and because of this allows data to be persistent even if views change their state.

* View:

The View layer only manages updates on Activity and Fragment classes with the help of their Adapters and View Holders. On this layer the View components observe for data changes in the Model and update the user interface accordingly. It also triggers events for the Model through the View Model in order to get data changes.



**Class Diagram**

Class Diagram Design

### Description of Classes

Because the system is designed as a MVVM architectural pattern and the number of classes surpasses 50, the description of the classes will be divided into Model/View/View Model categories and the common descriptions of the classes will be gathered under each group.

**View Classes**

All Activity classes extend from the AppCompatActivity and override some of the lifecycle call-back methods like onCreate(), onResume(), onDestroy. In the onCreate() method we only initialize, populate and set listeners for the UI elements, create the required view models, get the references of the exposed LiveData objects and create observers for them. In the onResume() method we only trigger events for refreshing the data in the UI elements.

Because of the complexity of the project we’ve used fragments which extend from the Fragment class and at least override onCreate(), onResume(), onCreateView, onAttach(), onDetach().The onCreateView() method is used in the fragments classes to have access and inflate its layout and also populate and set listeners to the UI elements. The onAttach() and onDetach() methods are also used in fragments to set/unset the base activity’s listener. To trigger events to the base activity, the fragments contain inner public interfaces which must be implemented by the activity in order to receive click events.

Beside static views, the project also has dynamic lists of UI elements, because of this the views use the RecyclerView UI element to list items with the help of adapter classes. This adapter classes extend from the RecyclerView.Adapter<> abstract static class and contain an inner class which extends from RecyclerView.ViewHolder abstract static class. The adapter classes are used to hold and manage the data collections, and the inner view holder classes are used to set and manage the view elements. The adapters of fragments use the fragment’s inner interface as a listener to trigger click events to the base activity, and the adapters of activities have their own inner interfaces to trigger the click events.

|  |  |
| --- | --- |
| **Class Name** | **Description** |
| MainActivity | It is the launcher activity, and it exposes the products categories through a Tab Layout in a two columns grid Recycler View. The class contains adapters for each category, allowing to set the filtered group of products whenever the tab is selected. Also, there is the option to directly add a product through a Floating Action Button, view product details or access the main menu. From the main menu you can login, register, view account information, add a new product, sign out, access the about or contact page. |
| ProductsAdapter | It is used by the MainActivity class to manage the products by category. It contains a list of Product class entities and handles the RecyclerView layout in its ViewHolder inner class. |
| ViewProductActivity | This activity shows the added date, title, description, image and/or video thumbnail of the product, and the user’s alias, average ratings as a rating bar UI element with a floating value and the total number of flags it has. From this view the logged in user can view the user’s reviews, products and can barter with the user. The anonymous user is restricted to interact in this activity, he can only view the page’s content. |
| ReviewsActivity | It receives the user reviews through its intent and shows them as a list in its recycler view. It uses the ReviewsAdapter class to manage the reviews. |
| ReviewsAdapter | It contains a list of UserReview class entities, and it is used by the ReviewsActivity class to handle the user’s reviews. |
| UserProductsActivity | It is used to list the user’s products. It receives the user id through its intent in order to query for products in the database by id. Each product can be viewed in detail by clicking on it. |
| UserProductsAdapter | It contains a list of Product class entities and it is used by the UserProductsActivity class to handle the user’s products. |
| OfferActivity | The activity is used to make an offer on the selected product to the item owner. It receives the user id, product id and product’s image Uri through its intent in order to tie the newly created offer to the user’s product. It also adds the email and the offeror’s message to the new offer. |
| ViewImageActivity | This activity gets the image URI of the product from its intent and shows it with the help of the Picasso Library in an ImageView element. |
| ViewVideoActivity | This activity gets the video URI of the product from its intent and loads the video in a VideoView element with a MediaController. Because the Video is usually downloaded slowly there is a ProgressBar element which shows that it is loading. |
| AddProductActivity | This activity can only be used by logged in and unrestricted users. In order to insert a product into the database the user must add an image, set a title and a description of the item, and can add a video and select a category. The activity shows the video and the image after selection as a thumbnail in their ImageView containers. The categories are loaded in a Spinner element from the strings.xml file found in resources folder. |
| LoginActivity | It is used to login as a registered user with email and password, or to recuperate the user’s password. When the user presses the log in button, the email and password TextView elements are checked for validity and then signs into the account. |
| RegisterActivity | It is used to register for a new account with first name, surname, telephone, alias, email and password. When the user presses the register button, all the text fields mentioned above are checked for validity and checks if the terms and conditions have been agreed by selecting the CheckBox element. |
| ForgotPassActivity | It is used to reset the user’s password by email. |
| TermsAndConditionsActivity | It is used to read the terms and conditions of the application. |
| AboutActivity | It is used to read some information about the project. |
| ContactsActivity | It is used as a contact page in which you can find information on how to contact the company. |
| AccountActivity | It is the base activity for the ProfileFragment, MyProductsFragment, OffersFragment and ReviewsFragment. It implements the fragments’ inner interfaces as listeners to react to certain click events inside their views. It contains the SectionsPagerAdapter in order to handle the fragments when the TabLayout element is pressed. |
| SectionsPagerAdapter | The class is used to manage the tab selection in the TabLayout element and load the fragment holder according to the tab index. It extends the FragmentPagerAdapter abstract class and overrides the getItem(int), getPageTitle(int) and getCount() methods. |
| ProfileFragment | The fragment is constructed as a singleton class, used to show user’s profile details like reviews average value, number of flags, first name, surname, telephone, alias and email. To show all this information it requests data through the ProfileViewModel and ViewReviewViewModel classes. |
| MyProductsFragment | The fragment is constructed as a singleton class, used to show the logged user’s products by id. It uses the MyProductsViewModel class to query for data and the MyProductsRecyclerViewAdapter to contain and show the products. |
| MyProductsRecyclerViewAdapter | It contains a list of Product class entities and it is used by the MyProductsFragment class to handle the user’s products. |
| ViewMyProductActivity | It shows the added date, title, description, image and/or video thumbnail of the product, and the user’s alias. |
| OffersFragment | The fragment is constructed as a singleton class, and it lists the user’s received offers. To show the received offers it requests data through the MyOffersViewModel class. It uses the OffersRecyclerViewAdapter class to manage and show the offers. |
| OffersRecyclerViewAdapter | It contains a list of Offer class entities and it is used by the OffersFragment class to handle the user’s received offers. |
| ViewOfferActivity | It shows details of the offer like product’s image, offeror’s alias and email. The offer can be refused or accepted through predefined buttons. The offer is loaded from it’s intent and it changes the state of the entity in the database through the ViewOfferViewModel class. |
| ReviewsFragment | The fragment is constructed as a singleton class, and it lists the user’s accepted or rejected offers. To show the offers’ history it requests data through the MyOffersViewModel class. It uses the ReviewsRecyclerViewAdapter class to manage and show the rejected or accepted offers. |
| ReviewsRecyclerViewAdapter | It contains a list of Offer class entities and it is used by the ReviewsFragment class to handle the user’s accepted or rejected offers. |
| ViewReviewActivity | This activity shows the user’s review on the accepted/rejected offer. It uses the ViewReviewsViewModel class to fetch the current review of the offer. If the offer doesn’t have yet a user review it will show as an initial state with nothing set or inserted in the review. User can modify the review even if it is already set. The user can change the user rating bar and value, the review and a Switch element representing if the receiver is flagged or not. |

**View Model Classes**

The view model classes extend from the ViewModel abstract class, to be life cycle aware when the views change their state and to comply with the embedded Android’s MVVM architecture pattern. The view model classes use and expose MutableLiveData<> object holders to their views.

|  |  |
| --- | --- |
| **Class Name** | **Description** |
| ViewModelFactory | It implements the ViewModelProvider.Factory interface and thus exposes the view model objects through the Factory Method design pattern. |
| ProductsViewModel | It uses the AthentificationModel class to get user details and to sign out, and the ProductsModel class to trigger products related events. |
| LoginViewModel | It uses the AthentificationModel class to login into the Firebase’s servers. |
| RegisterViewModel | It uses the AthentificationModel class to register a new account for Firebase’s servers. |
| ForgotPassViewModel | It uses the AthentificationModel class to trigger a password resetting event. |
| MakeOfferViewModel | It uses the OffersModel class to send an offer message. |
| AddProductViewModel | It uses the ProductsModel class to add a new product into Firebase’s database. |
| ProfileViewModel | It uses the AthentificationModel class to get the user related information. |
| MyProductsViewModel | It uses the ProductsModel class to trigger a get event of the user’s products. |
| MyOffersViewModel | It uses the OffersModel class to trigger a get event of the user’s received offers. |
| ViewOfferViewModel | It uses the OffersModel class to set the selected offer as accepted or rejected. |
| MyReviewsViewModel | It uses the OffersModel class to trigger a get event of the user’s offers’ history. |
| ViewReviewViewModel | It uses the AthentificationModel class to get user’s id and the ProductsModel class to trigger reviews related events and set the current user’s review. |

**Model Classes**

The model classes like AuthentificationModel, ProductsModel. OffersModel and ReviewsModel are constructed as singleton design patterns so that view models don’t reallocate them and also enables them to reuse the code by directly or combining their instances. The classes mentioned above use the FirebaseAuth instance to login, register, reset password or get user account related information.

The entity classes: Product, Offer and UserReview implement Parcelable interface in order to serialize the object for data sharing between views.

|  |  |
| --- | --- |
| **Class Name** | **Description** |
| AuthentificationModel | In general, it is used for authentication related actions. The FirebaseFirestore instance is used to create the user’s profile separate from the signed in account. |
| ProductsModel | This class manages product related actions and uses the FirebaseFirestore instance to add new products into the database and to query for products with different filters. The FirebaseStorage instance is used to store the multimedia data related to the products. |
| OffersModel | The class manages offers related actions and uses the FirebaseFirestore instance to add new offers into the database, to set them and to query them with different filters. The model is also tied to the Products entities, so when an offer is accepted the related product is deleted from the database. |
| ReviewsModel | It manages user reviews related actions and uses the FirebaseFirestore instance to add new reviews or reset them. |
| UserProfile | The class is used as an entity in FirebaseFirestore database to save and extract the extra data of the user’s information. |
| UserReview | It is used as an entity in FirebaseFirestore database to save and extract the user’s reviews. |
| UserReviewAggregationData | It is only used to contain the reviews aggregation data extracted from FirebaseFirestore database in order to show the number of flags and the average value of the ratings. |
| Offer | It is used as an entity in FirebaseFirestore database to save and extract the user’s offers. |
| Product | It is used as an entity in FirebaseFirestore database to save and extract the user’s products. |
| Response | This class is used to notify the views, by wrapping it in a MutableLiveData<> container, when the Firebase servers receive a response. It contains a Boolean type value which states if the response was successful or not, and a String which can contain exception or internal messages. |

# **Firebase Design**

## Firebase’s Entity Relationship Diagram

## Authentication

The Firebase Authentication database is only used to store minimal account information required to sign in. As modelled in the figure above the application only needs the email and display name. The user id is auto-generated on account creation.

## Cloud Firestore

The Firebase Firestore database is comprised of four primary collections: Users, Products, Offers and Reviews.

### Users Collection

### Products Collection

### Offers Collection

### Reviews Collection

## Cloud Storage

The Firebase Firestorage is used to

# **Marketing & Pricing Strategy**

The main marketing strategy behind the “Barter Trader” project is that the app allows anyone to have access to view the bartered items and have a feel of our solution. This approach will exponentially attract customers to register, because anonymous users can see the offers in the app, and if they want to, they have the option to register and barter for the desired item.

At first, we will not introduce advertisements, because we need to make a good impression on the market and attract as many registered users as we can. After surpassing 250000 reads a day in the Cloud Firestore, which means the Firebase costs will surpass $25 a month, we can safely introduce ads among the products. I’m not relating to database writes, because most of the clients will be consisted of anonymous and some registered users. Technically the advertisements will be inserted after a pre-set number of browsed products in the MainActivity view, with no development changes required, because the ad will show just like an item.

When we will reach the goal of having tens of thousands of products constantly active each day, the costs will substantially rise because of our policy of allowing anonymous users to view the items. This is because some of the registered users are generating item views and the anonymous and logged in users are watching the items, which means we can have millions of views each day. Reaching this level of database reads, we need to add other meanings of income. One of the strategies is to introduce multiple types of accounts from basic to premium. Basic accounts will still have ads shown among the products and limited number of inserted products, but the other account types will pay weekly/monthly/annually packages with different advantages like having unlimited number of product inserts or being able to promote items. But we don’t want to limit the basic accounts or loose them because they are not power users and because it is not feasible for them to barter one/two products. So, we will allow payable options for one day/a week/a month to promote their desired products or to add another extra product to their maximum number of items.

# **User Interface**

## Main View

### Select Category Gadgets

### Select Category Clothes

### Select Category Tools

### Select Category Bikes

### Select Category All

## View Selected Product

### Barter for Selected Product

### View Reviews on Selected Product

### View User’s products on Selected item

### View Products’ Photo

### View Products’ Video

## Main Menu Selection

### Sign in Page

### Sign in –Reset Password Page

### Sign up Page

### About Page

### Contact Page

## My Account

### Profile Page – View Profile

### Profile Page – User’s Reviews

### Products – View User’s Products

### Products – View Product

### Offers – View Received Offers

### Offers – View Offer

### Reviews – View User’s Accepted/Rejected Offers

### Reviews – View and Set Review for Accepted/Rejected Offers

## Add Product

//TO DO show both options of adding a product(from menu and from button)

### Add Product Page

### Add Product – Select Category

### Add Product – Select Photo

### Add Product – Select Video

## Add Product

# **Test Documentation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **Input** | **Output** | **Result** |
| Start Application | Tap the application’s icon. | The application starts in the main view showing the products under the gadgets category. | Successful |
| Register |  |  |  |
| Sign in |  |  |  |
| Sign Up |  |  |  |
| Sign out |  |  |  |
| About |  |  |  |
| Contact |  |  |  |
| Add Button |  |  |  |
| Categories |  |  |  |
| Add Product |  |  |  |
| Search item |  |  |  |
| Flagg user |  |  |  |
| Rate user |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# **Appendix:**

# **References:**