Things to be defined:

‘APP-NAME’

‘PRODUCT-NAME’ Example in the use case of a product being bought (such as a shirt or phone)

A picture containing logo

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | DIMA – E-SHOPPING ‘APP-NAME’ | |
| Design Document | |
| Etion Pinari | 10619348 |
| Pietro Guglielmo Moroni |  |
|  | AY 2020/2021 |

*Table of contents*

1. INTRODUCTION pg.2
2. OVERALL DESCRIPTION pg.7
3. APPLICATION DESIGN: pg.14
   1. Functionalities pg.14
   2. Interface design: pg.17
   3. System Architecture pg.47
4. BACK-END IMPLEMENTATION pg.49
   1. E-R Schema pg.49
   2. Logical Schema pg.49
5. EXTERNAL SERVICES USED pg.59
6. USE CASES & TESTS pg.61
   1. Use Case full example pg.61
      1. Client scrolling the homepage pg.61
      2. Client using the search bar pg.61
      3. Client buying a product pg.61
      4. Client logging in and registering pg.61
   2. Tests & Unit Testing pg.61
7. REFERENCES pg.61
8. Introduction
   1. Document Purpose

This document has the purpose to clearly define the functionalities that the system-to-be will provide, the goals it strives to achieve and indicate general use cases. It will also define the general behavior and specific limitations of the system. This document is primarily addressed to the programmers and mostly includes technical language.

* 1. Definitions

Users: Every person who uses our application.

Clients: All users except for vendors.

Vendor: An entity which can use the application to publish its products to be sold.

* 1. Application Purpose

Our application ‘APP-NAME’ has the specific purpose of creating a simple but slick , user-friendly interface which connects clients and the vendor allowing them to respectively buy and sell products. ‘APP-NAME’ does not try to encapsulate many different markets and show all of them in the same application but rather (…?)

‘APP-NAME’ allows the vendors to share their products with all the users of the application as to achieve a higher degree of publicity.

1. Overall description

With this project we attempt to create a user-friendly, adaptive, and reactive mobile application that connects clients and the vendor with one-another.

1. Application design
   1. Functionalities
   2. Interface design
   3. System Architecture
   4. S
2. Back-end implementation
   1. E-R Schema

Diagram, schematic

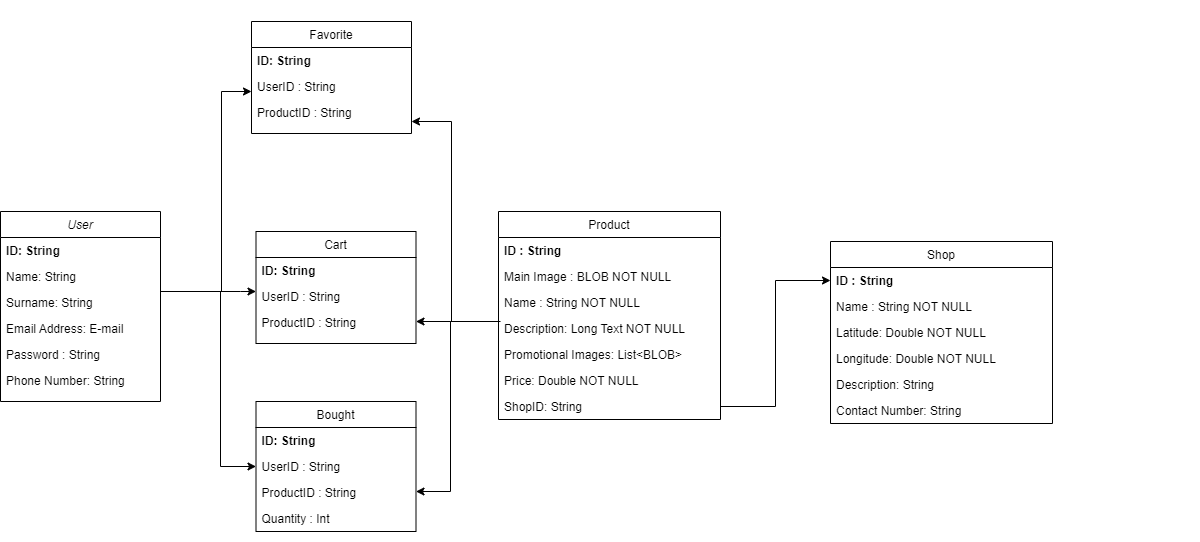
Description automatically generatedWith regards to the back-end we decided to use a document-oriented database with the help of Firebase’s [Realtime database](https://firebase.google.com/docs/database) with the objective to have an easy to use and responsive DBMS (more about it in the [external service’s section](#ExternalServicessection)). With it we will implement the following E-R scheme:

-Entities:

* Users: The persons who will use the application to buy products
  + Attributes: The most interesting attributes of the users are their personal data so as to allow us to identify different people from each-other. As such we can say that the user’s full name, email address, phone number and location are key attributes to store. This last attribute will also allow us to suggest close-by stores to the person’s current location.
* Products: The article to be sold
  + Attributes: The image of the product, its price, name, and descriptions all are key attributes.
* Shops: The (possibly many) vendor(s) which publish products.
  + Attributes: The shop’s location, name, description, and contact number are important attributes to allow the application to function correctly.

-Relationships:

* Bought: A relation between the user and product which counts how many times the product has been bought by the user. This could be useful for a future implementation of the recommender system.
* Cart: A relation between the user and product which stores in memory all the products which the users will possibly buy in the future.
* Favorite: A relation between the user and product which stores in the database all the products which the user really likes.
* Publishes: A relation between the shop and the product that is or was available to be bought.
  1. Logical Schema

As for the logical schema we decide to implement the following schema:

We have opted for this solution, following the requirements of the E-R schema, general rule of thumbs and common design patterns used for the creation of a database. The main problem with the database would be the conversion from the NoSQL, JSON-formatted text data to something that is application-usable for the on-app representation.

1. External Services and Libraries

Our ‘APP-NAME’ strives to be an intermediary for all clients and vendors, both with respect to their connection and their orders and payments. In that regard it needs to use external services which are ready off-the-shelf. Starting from:

-PayPal / Stripe, allows the users to pay for the products online safely and securely. It lets (…?)

-Firebase Authenticator provides the main identification method of all users, storing their email, password, and unique user-id. This last attribute is used as the key inside the Firebase Realtime Database to store all other information regarding the user’s relations with the products. Firebase authenticator was chosen to be used for its ease of use in helping build secure authentication systems, and to improve the authentication experience for end users.

-Google Maps contributes to the extension of the main functionality of our project. It allows users to find close-by shops with the help of the much popular interface that Google Maps provides.

-Firebase Realtime Database has been chosen as the storage platform and DBMS for its advantage in terms of scalability and security. In it we store everything shown in the [logical schema](#LogicalScheme). Since the Realtime Database stores the data in a NoSQL model, some slight modifications are needed to store the many-to-many relationships. For instance, we store in each user not only the information shown in the logical schema but also three other documents with key

\* ‘Bought’ to identify all the product ids of the products that the user bought,

\* ‘Favorites’ to identify all the liked products’ ids of each user

\* ‘Cart’ to identify all product ids of the soon-to-be-bought products. This is also stored in the database in case the user does not buy the product immediately and wants to continue shopping from another device.

All this information is stored in a document-based model, which need to be converted from a JSON-formatted text to our model’s objects as specified in section 3 of Application Design (…?).

Many libraries have been used to improve the look-and-feel of the application such as the provider package, which implements lazy loading of the product’s data and simplified resource management. Another one is the carousel slide, used in many parts of the application. This library allows for example to show the featured products in a more condensed way. ( and could be used to show different images of the same product…? )

1. Use Cases & Tests
   1. Use case example

The first use case that comes to mind is of a client C. who wants to buy a ‘PRODUCT-NAME’. He first opens our application and sees in the carousel slide all the featured products. Since C. is not interested in any of them, he searches the main screen’s list view to find the product he is searching for. Because he is not logged in and the recommender system cannot suggest any products for him, he uses the search bar to search for ‘PRODUCT-NAME’. In there he sees two products for which he is interested. He taps the first one and sees that it is not what he specifically wants. He taps the back button and taps the second item that interested him. This time, the product is of C.’s liking, and he decides to buy it. A screen shows that he needs to log-in or register before going on with the purchase. After entering his personal information, location, and credit card details the order for the ‘PRODUCT-NAME’ is placed.

In the following sections we show all the sequence diagrams of this example and furthermore other diagrams that clarify the intended implementation of the map and cart components.

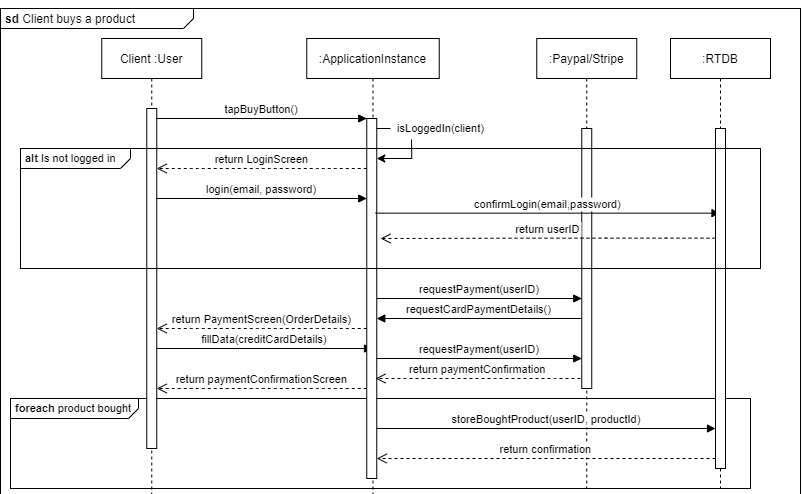
* + 1. Client scrolling the homepage

As soon as our client C. opens our application, a future instantiator will be called to get the reference of the Realtime Database (RTDB) so that the application can fetch all data regarding the products. In the meantime, the application returns the loading screen until the database reference has been returned at which point a query for the first N products will be carried on. The application afterwards will build the screen to be viewed from the client.

* + 1. Client using the search bar

If the client C. requires another product which has not been shown on his screen, he can simply tap the search bar on top of the screen and enter the product name. At that instant the application sends the query to the database which will return all the products that satisfy the query. Once the application has retrieved the information, it will use it to create a view on top of the screen (without navigating to another screen) and update the current screen. This new list-view can be scrolled through and each element inside of it might be pressed to redirect to the product’s page, so a better view can be had.

* + 1. Client buying a product

Once a client has decided to buy a product one way to carry out the payment is to click on the buy button. This is the only operation that requires an identification for reasons related to the payment, so if the user is not logged in yet, he is redirected to the login screen. After having logged in, the payment screen will be shown with a form to enter the order details such as the location, preferred delivery date and credit card details. This order is then subdivided by each product and only the tuple   
(userID x productID x location x deliveryDate) will be stored in the database. Another supply chain management application will then connect the order to each shop, but this part is outside of the scope of the application.

preferred delivery time (…? Add on the database ER SCHEMA.)

(…? What if we buy without logging in?)

* + 1. Client logging in

A client might want to register or log in to ‘APP-NAME’ so that he might have his cart automatically updated between different devices which he might want to use to enter inside our application.  
The login process follows standard procedures as in many other applications, during which the application must verify that the fields have been filled correctly and that the user exists already.

After logging in, the user screen (also known as LoggedIn screen) will be returned to the user with the list of his favorite products and user details that he might want to update.

* + 1. Client registration

A client might want to register or log in to ‘APP-NAME’ so that he might have his cart automatically updated between different devices which he might want to use to enter inside our application. The registration process follows standard procedures as in many other applications, during which the application must verify that the fields have been filled correctly and that another user with the same email address does not already exist.

The cart that had been stored in a local variable of the application memory is then stored in the database as well and the LoggedIn screen gets returned.

* 1. Tests & Unit Testing

We have implemented many unit tests such as(…?). Furthermore, a usability evaluation has been carried out with user testing. Users showed that the interface was easy to use although some had problems with the fact that they could not navigate (…?) :

1. References
2. Etc…