A picture containing logo

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | DIMA-eShop | |
| Design Document | |
| Etion Pinari | 10619348 |
| Pietro Guglielmo Moroni | 10625198 |
|  | AY 2021/2022 |

*Table of contents*

1. INTRODUCTION pg.2
   1. Document Purpose pg.2
   2. Application Purpose pg.2
   3. Definitions pg.2
2. OVERALL DESCRIPTION pg.3
3. APPLICATION DESIGN pg.6
   1. Functionalities pg.6
   2. Interface design pg.7
   3. System Architecture pg.13
4. BACK-END IMPLEMENTATION pg.14
   1. E-R Schema pg.14
   2. Logical Schema pg.16
5. EXTERNAL SERVICES AND LIBRARIES pg.17
6. USE CASES & TESTS pg.19
   1. Use Case example pg.19
      1. Client scrolling the homepage pg.20
      2. Client using the search bar pg.21
      3. Client buying a product pg.22
      4. Client logging in pg.23
      5. Client registration pg.24
      6. Client adding items to the cart pg.25
      7. Client uses map pg.26
   2. Tests & Widget Testing pg.27
7. REFERENCES pg.32
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. Application Purpose

Our application DIMA-eShop has the specific purpose of creating a simple but slick, user-friendly interface which connects clients and the vendors allowing them to respectively buy and sell products. It works as an intuitive interface for clients to browse and buy products from our shop. The application’s role is to fetch the product catalogue data from our server and provide it to the user that will then be able to both purchase and keep track of products they like by saving them as “favorites” for later.

The application is designed around the concept of not overwhelming the customer with a plethora of different options at the same time, but rather quietly present the products one at a time.

DIMA-eShop allows the vendors to share their products with all the users of the application as to achieve a higher degree of publicity.

The application is also designed to work for registered and unregistered users, providing most functionalities to both categories. Registered users will benefit from their data being saved on the server and persisting across multiple devices, while unregistered users’ data will be saved locally.

* 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. Overall description

With this project we attempt to create a user-friendly, adaptive, and reactive mobile application that connects clients and vendors with one-another. We try to offer a seamless experience for all types of handheld devices such as smartphones and tablets and our DIMA-eShop is available for Android and iOS devices so that adaptability will never be a problem. We try to help the users, and future clients of this application with their shopping by allowing them to check different products that might be of their interest with the tap of a finger.

The application is divided into four main sections:

* Home screen

This section is the first meeting point of the user with the application and products that are offered, so it will try to attract clients with a smooth and simple interface with the point of focus being not overcrowding the user’s mind with too many different choices as it is seen that it would be harder for them to make a decision on which product they would like to buy[1,2]. The home screen will thus solely contain a carousel with the featured products of the day and, following it will be a one-by-one view of all the products that the client has not yet seen. What we mean by a one-by-one view, is that each product’s image, title and/or description, and finally cost will be shown on the screen one after the other, without ‘competitor products’ congesting the screen. From the home screen, the user will be able to access a search view in which, by typing into an input field, they can query for desired products.

* Map screen

This section will try to be a connecting point between the physical shops that are found close to the client and the client itself. Not only will it show the shops’ position but also a featured list of products that the shop contains, so to know what to expect. The featured list will be a short but concise list of products, as to follow the philosophy of the application of being simple-but-effective.

This part of the application is important for the clients that just browse products on DIMA-eShop but want to see the physical product before buying it.

* Product’s screen

This screen is important because it is the deciding step of a client’s acquisition of the product. As such it will allow the product to be the focus of the screen without trying to push for other ‘similar’ products that the user might want to buy with the scope of guaranteeing focus on each product.

* Cart’s screen

In this screen the user will be able to which products are currently contained inside their cart. For each product in the list, the user will be able to add more or remove it or navigate to the product’s page to check its details. Once content with the current cart, the user will be able to proceed to check out and complete the purchase of their order.

1. Application design

DIMA-eShop’s design philosophy is simplicity and the focus on helping the client carry out the operations that they want. As such the application design will represent these ideas in the areas of functionalities and interface design. The system architecture will define the underlying structure which will help with the operability of the functions the application provides.

* 1. Functionalities

The most important functionality we provide is the ability to view each product singularly and focus on the visual preview of the product instead of its name or description in the home screen and let the attractiveness of the product draw the client to buying it.

After a client decides to buy the product, our application provides the ability to request the product at one’s desired location and preferred date and sends this information to the vendor of the product.

With the help of external services, the payment is carried out inside the application without the need to log in. The client will need to fill the form with its personal information and credit card details. This information will be then only used for the order and not associated with a user for privacy reasons.

As cited in the [map section](#MapScreenSectionDescription), we offer the possibility to view shops close to the client and the associated products to that shop. This is done with the help of the Google Maps external service and when a point on the map is pressed the list-view of that shop’s products will be shown.

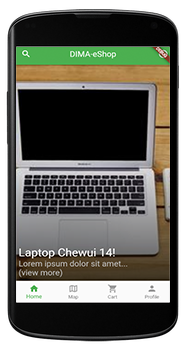
Another important feature of the application is the ability to give all products the ability to be visualized by each client and as such in the main screen a simple recommender system will show different products each time the user opens the application.

* 1. Interface design

With the focus on simplicity in design and aiding the user on the interaction with the products to be sold, we implement an interface that contains as little text as needed.

The home screen will contain a carousel shows one by one, all products of the application. The client can view these items by scrolling in the vertical direction. It is important to note that only one product will be shown in this list.

In every screen we also see the bottom bar navigation which contains the Home-Screen icon, Map icon, User icon, and Cart icon. Each button sends the user to the section of the application it is named after. An important aspect of the navigation is the fact when the user changes screens from a starting section to a destination section, the navigation history will not be lost from the starting section.

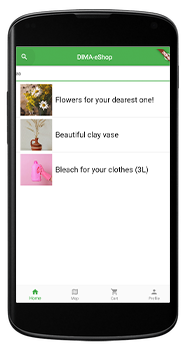


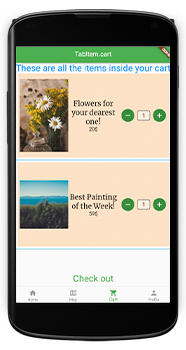
Successively the map screen will look as in the following figures, with the screen on the left showing only the shops close by and the screen on the right, after a marker press, showing the related products.

After tapping on a product from anywhere inside the application, the application will redirect the client to the product’s singular page where the description will be available to be read by the client. In this screen a cart button and a buy button are present with the usual behavior that is expected, also detailed in the [‘Client buying a product’](#ClientBuyingAProduct) sequence diagram.

The user can tap the back button to navigate back to the previous page or tap any of the bottom bar navigation buttons to be redirected to any of their screens without losing information on the current screen.

Tapping on the magnifying glass icon in the top left corner of the home screen will redirect the user to the product search page. Here they will be able to look for any desired product by querying for its name. The query results are displayed in a list view and show an image of the product and its name.



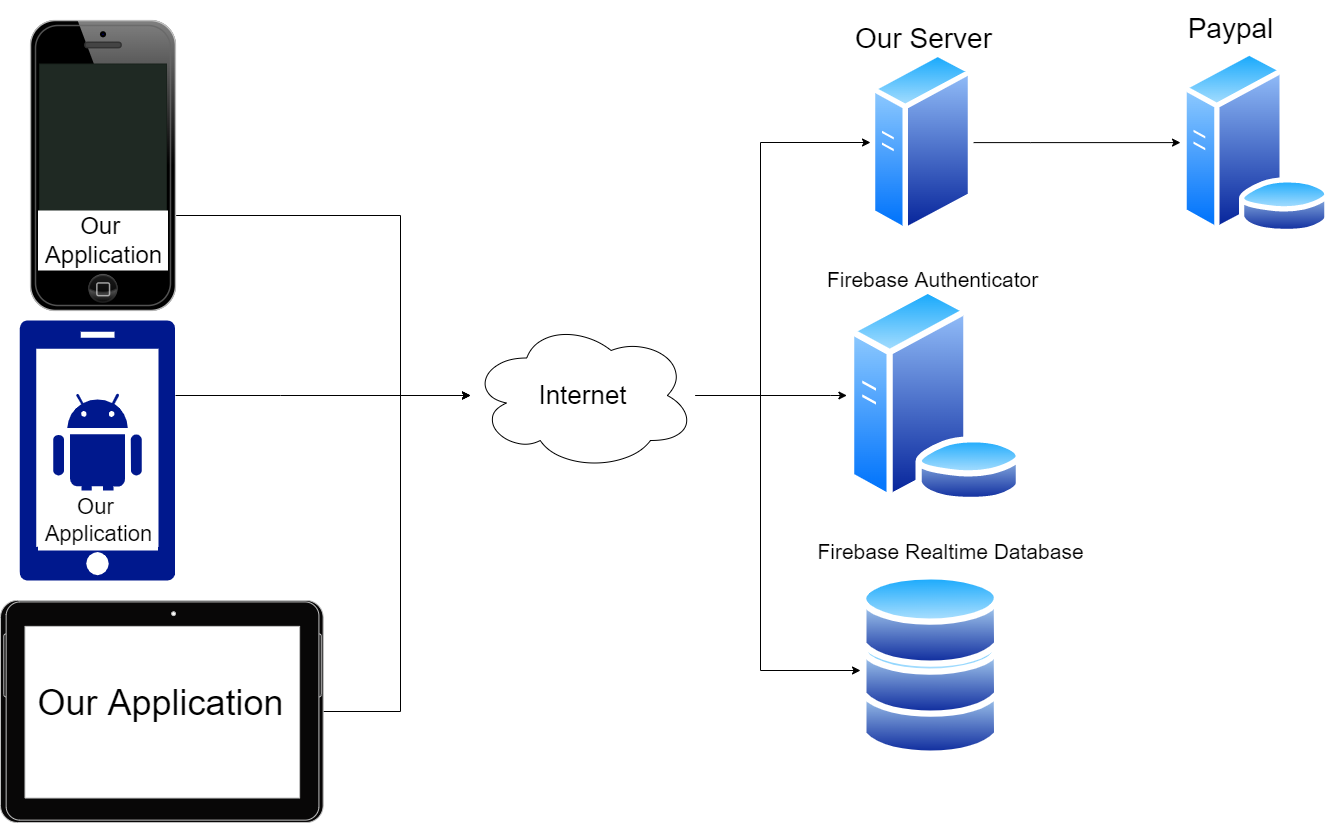
  
The cart screen will contain all the products that the user wants to buy but has not finalized buying yet. It will be a list view with the quantities of the requested products plus the unitary price, image, and title of the product. The client can press the plus button to add more of the same product to the cart or press the minus button to remove one product at a time. In the bottom of the page, a checkout button will be shown so the client might finalize their shopping experience. The following shows one example screen that the user might see.

The user screen will contain the name of the user, and two buttons to visualize their favorite products list and the purchase history. Optionally, if the client is not logged in, two more buttons will be shown allowing the client to log in or register on the website. A close-up of a cell phone

Description automatically generated with medium confidenceA close-up of a cell phone

Description automatically generated with low confidence

* 1. System Architecture

Our application’s architecture is simple and when possible, the device communicates directly with [the service provider](#ExternalServicessection), such as the authenticator service provider (Firebase Authenticator) or the database provider (Realtime Database). We also decided to handle payments through an API server that acts as a man-in-the-middle between our front end and the back end of the chosen Payment Service Provider (PSP). This effectively decouples the view from how the payment is processed and allows us to make changes (e.g. maintaining and updating the API, changing the PSP) without having to take service down or affecting the user experience.

1. Back-end implementation
   1. E-R Schema

Diagram, schematic

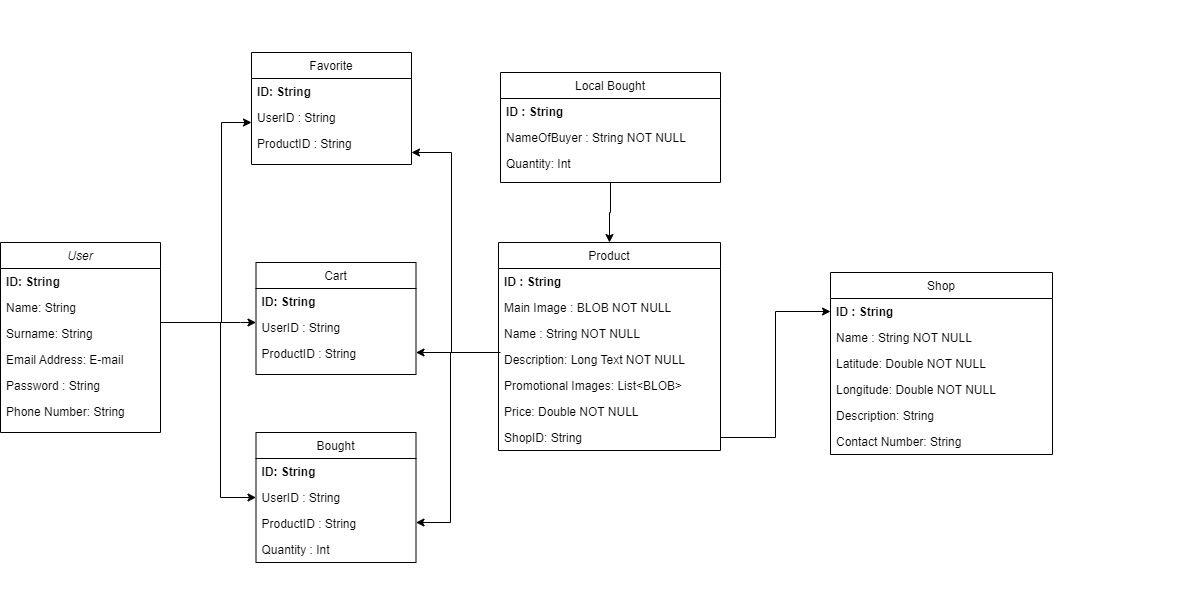
Description automatically generatedWith regards to the back-end we decided to use a document-oriented database with the help of Firebase’s [Realtime database](https://firebase.google.com/docs/database) with the objective to have an easy to use and responsive DBMS (more about it in the [external service’s 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. As such a global database manager class will be used to convert and locally store all JSON-data, into usable variables inside the application’s data model.

In the case of the user which does not log in the application, every transaction of his will be kept into account by a progressive global ID, alongside the information of how many products he bought, and which ones.

1. External Services and Libraries

Our DIMA-eShop 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:

-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.

-The payment REST-API resides on a custom web server that exposes an endpoint for the user to make payments when purchasing products from any shops. This server will communicate with the Payment Service Provider’s API to make sure the payment goes through and notify the user if the payment has failed.

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 4 in [Logical Schema](#LogicalScheme).

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 utilizing the publish, subscribe pattern. 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. Another important package is the location package which retrieves the current position of the device, used mainly in the map screen to show close by shops.

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 generic product from the shop. 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 the desired product’s 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 desired product 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

Diagram

Description automatically generatedOnce 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.

* + 1. Client logging in

A client might want to register or log in to DIMA-eShop 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 DIMA-eShop 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. Client adding items to the cart

If a client wants to add items in the cart to buy many products contemporarily, he simply will need to press the cart button on the product’s screen and the application will store the product for a later purchase in the local memory. If the client is logged in this data will be stored in the database too as shown in the following sequence diagram.



* + 1. Client uses map

When the client wants to see the shops nearer to him, he might simply press the map button inside the bottom navigation bar. This will cause the application to send a query to the database to retrieve all shops that are close to the phone’s location. If any of the markers shown on the map are pressed, then a preview of the products of that certain shop will pop up on the screen while the map is still present which allows the client to know what products are featured in each shop.

* 1. Tests & Widget Testing

We have implemented many widget tests, for each of the subcomponents that are used everywhere in the application such as the Product Card, or the list view of the items placed in the cart, order history or favorites. Within the widget test we control whether the widgets show the desired item, such as the button to buy a certain product, the button to add it in the cart or the image of the product.

Regarding integration tests, we have carried out the most important operations using fake, local databases and bypassing communication with the database server. The operations which are common to all users are the ones regarding the state of the cart, favorite items, scrolling of the products and the order placement.

Furthermore, a usability evaluation has been carried out in many phases through user testing. Users showed that the interface was easy to use although some had problems with the fact that they did not like how going back to previous pages was initially implemented. Different iterations of user testing allowed the developers to remove all unexpected behavior of the application.

UNIT TESTING

The application is not particularly suited for unit testing as widget and integration tests already cover the majority of the logic.

WIDGET TESTING

|  |  |  |
| --- | --- | --- |
| Widget | Description | Notes |
| Home Page | The test builds the basic structure of the app and makes sure that the bottom nav bar is constructed correctly by asserting that the buttons and their relative data is visible on the screen |  |
| Map Page | The test instantiates the application and places a fictitious shop at a predefined location on the map. It then asserts that the map is rendered. | To not make the test depend on the on-line database, the test uses hard-coded mock data. |
| Payment Page | The test builds and asserts that all the elements of the payment page are loaded and rendered correctly. |  |
| Payment Details Page | The test renders the payment details page inside of a scaffold and populates it with mock data, to make sure that it works as expected. |  |
| Product Page | The test builds a product page and asserts that every element is properly rendered. | The test uses mock data to avoid any dependencies with the on-line database. |
| Widget | Description | Notes |
| Sign In Page | The test asserts that the sign in page is rendered correctly. |  |
| Sign Up Page | The test asserts that the signup page is rendered correctly. |  |
| Payment Successful Page | The test builds a mock cart with some hard coded products to assert that the payment summary is properly displayed to the user. |  |
| ShoppingProduct’s product variant | This test asserts that rendering a ShoppingProduct widget of the “product” variant actually displays the add/remove buttons. | The widget is assigned a mock product for testing. |
| ShoppingProduct’s favorites variant | This test asserts that rendering a ShoppingProduct widget of the “favorites” variant actually displays the favorite and add to cart buttons. | The widget is assigned a mock product for testing. |
| ShoppingProduct’s history variant | This test asserts that rendering a ShoppingProduct widget of the “history” variant actually displays the purchase date. | The widget is assigned a mock product for testing. |
| Search bar renders | This test asserts that loading the search page standalone actually renders the search bar correctly. |  |
| Search bar interactivity | This test asserts that loading the search bar’s value persists after the test has been edited. |  |
| Shop page renders | This test makes sure the shop page actually displays the information for the correct shop. | The widget is provided mock data for testing. |
| Shop page displays products | This test asserts that the shop’s products are displayed in the shop page. |  |

INTEGRATION TESTING

|  |  |  |
| --- | --- | --- |
| Widget | Description | Notes |
| Navigating the application | This test asserts that once that the application is loaded the bottom navbar is fully functional and allows the user to navigate the application’s pages freely. |  |
| Check out procedure | This test simulates the checkout procedure. The widget tester navigates through the payment and payment details page filling out the forms and submitting the payment. | The cart is populated with mock data to avoid database dependencies in the test. |
| Increasing product quantity in the cart | This test first asserts that the cart page is properly rendered, then proceeds to increasing the quantity of the product that is already in the cart to be included in the checkout order. | The cart is populated with mock data to avoid database dependencies in the test. |
| Tapping a product from the home screen | The test loads up the application and taps on the first product it finds. It will then wait for the page transition to be over and asserts that the product’s information is visible in the product page. | The product catalog is populated with mock data. |
| Searching for a product that does not exist | This test runs a lightweight version of the app equipped with only the search page. When the widget tester interacts with the input field and searches for an item that is not in the catalog the test asserts that the correct message is displayed to the user. | The product catalog is populated with a single mock element.  The item name that is entered in the field is just random gibberish that is only supposed to be different from the name of the only product in the database. |
| Searching for a product that exists | This test runs a lightweight version of the app equipped with only the search page. When the widget tester interacts with the input field and searches for an such item the test asserts that the query results are properly displayed on the screen. | The product catalog is populated with a single mock element. |

1. References

[1]:  [S. Mathew, J. Joseph, Differential Impact of Information Display Formats on Consumer Decision-making, page 175-179, Proceedings of the 2014 Annual Conference of the Emerging Markets Conference Board January 9–11, 2014,](https://www.researchgate.net/profile/Baba-Gnanakumar/publication/309359634_Modelling_the_Nested_Mark_Modelling_the_Nested_Markets'_Prices_during_ets'_Prices_during_Festival_Season/links/5c53176c92851c22a39e3427/Modelling-the-Nested-Mark-Modelling-the-Nested-Markets-Prices-during-ets-Prices-during-Festival-Season.pdf#page=213)

[2] : [G. A. Pignatiello , R. J. Martin, R. L. Hickman Jr. , Decision fatigue: A conceptual analysis, Journal of Health Psychology 2020, Vol. 25](https://journals.sagepub.com/doi/pdf/10.1177/1359105318763510)