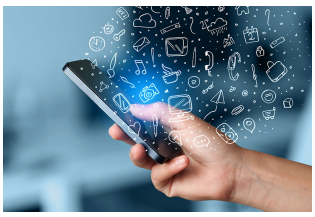
ISEL – Instituto Superior de Engenharia de Lisboa
ADEETC – Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

MEIM
Mestrado em Engenharia Informática e Multimédia
Curricular Unit of Master thesis

Mobile platform for selling used items with recommendations and an image-based system for recognizing similar items



Miguel Távora (45102)

Supervisor

Professor [Doctor] Rui Jesus
Professor [Doctor] Gonçalo Marques

Jury

President Professor [Doctor] Pedro dos Santos
Juror Professor [Doctor] Pedro Fazenda
Juror Professor [Doctor] Rui Jesus

February, 2024

Abstract

The purpose of this report is to describe the whole process that led to the final product obtained, which is the e-commerce mobile application and the recommendation system of products based on images.

The methodologies used in this project are divided into two important key factors. The first one is the impact of software architecture and how it can be applied to explore components like infrastructure needed, technologies, and programming languages. The second one is the importance of the recommendation system in an e-commerce application and how it might enhance the user experience of the application.

The software architecture has the objective of creating the whole system from the ground up, utilizing the software engineering principles. The main objective of software engineering should be to develop a system as simple as possible with minimal entropy possible. To achieve this, it is necessary to create the software architecture to use it as a guideline to build the project and also to expand it in the future. This was achieved through the requirement specification, logical and detailed architecture using a top down approach. Despite the time-consuming nature of creating the architecture and its diagrams, in the long run, it helps to reduce the entropy of the system facilitating the development of good quality code with a good organization.

The recommendations system proposed is based on searching for similar images. The user uploads one or more images, then the system will use deep learning using a multi class classifier and a distance metric. To evaluate the quality of the system several tests were conducted, obtaining a `MAP` of 0.897. However, this methodology has some key limitations related to the images themselves. For example, if a user wants a red item but the image has poor illumination or the photo is taken at a bad angle, it might affect the search results.

Resumo

O objetivo deste documento é descrever todo o processo que levou ao produto final obtido. Este produto é uma aplicação movel de comercio eletrónico com um sistema de recomendação baseado em imagens.

As metodologias utilizadas neste projeto estão divididas em dois importantes fatores. O primeiro é o impacto da arquitetura de software e como ela pode ser aplicada para explorar componentes como infraestrutura necessária, tenologias e linguagens de programação. O segundo fator é o sistema de recomendação numa aplicação de comercio eletrónico e como ele pode ajudar a aumentar a experiência do utilizador.

A arquitetura de software tem o objetivo de criar um sistema inteiro de início ao fim, utilizando os princípios da engenharia de software. O objetivo principal deve ser desenvolver um sistema o mais simples possível com o mínimo de entropia possível. Para isso é necessário criar uma arquitetura de software para utilizar como guia para construir o projeto e também expandi-lo no futuro. Isto foi alcançado através da especificação de requisitos e arquiteturas lógica e detalhada utilizando uma abordagem `top down`. Apesar da natureza demorada da criação da arquitetura e os seus diagramas, a longo prazo ela irá ajudar a reduzir a entropia do sistema facilitando o desenvolvimento de código de grande qualidade e com uma boa organização.

O sistema de recomendação proposto é baseado na procura de imagens similares. O utilizador submete uma ou mais imagens, de seguida o sistema utiliza aprendizagem profunda através de um classificador multi classe e uma métrica de distância. Para avaliar a qualidade do sistema foram feitos diversos testes, obtendo no final um `MAP` de 0,897. Contudo, esta metodologia possui algumas limitações relacionadas com as próprias imagens. Por exemplo, se um utilizador quer um produto vermelho, mas a imagem tem uma má iluminação ou a foto foi tirada num mau ângulo, o sistema pode apresentar maus resultados.

Acknowledgments

I would like to express my gratitude for the continuous support throughout my master's degree. I also would like to thank Rui Jesus and Gonçalo Marques, both of whom served as advisors for my thesis. Their guidance significantly helped improve the overall outcome of the project.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The main idea behind this project is that sometimes people have many items they no longer need and those items take up space. To address this issue, they can sell them in a thrift shop or sell them using an application. The second option is normally the best option due to its convenience, giving a wider audience, a more profitable potential and being always accessible. Another key factor is that most e-commerce applications use algorithms to analyze the browsing history and purchasing history to provide personalized recommendations.

With that in mind, this project aims to create a mobile application for users to sell their used items, similar to the traditional e-commerce applications. In this application, users can post items for sale, and other users can purchase those items. However, there are already a lot of applications that implement this kind of service such as `Wallapop` and `OLX`.

To enhance the project value, a new way to find products will be implemented in the application. For instance, sometimes users are trying to find an ideal product, but most of the searches are made by entering text-based queries written by the users. These queries, in some cases, are not the best way to find products, because they might not include the right words or because it is hard to describe a product. To address this, a recommendation system will be created to differentiate the application.

The idea of the recommendation system is to move away from the text-

based searches and explore alternative methods. The approach found is to use images. The process involves the user taking a picture from his phone or selecting a picture already stored on his device and send it to the application. The application will calculate the most similar images and return the products that correspond to those images.

These recommendations are based on deep learning algorithms, in particular, convolutional neural networks (`CNN`) [1]. Currently, `CNNs` can predict with very high accuracy the content of a given image based on its feature vectors. With those same vectors, it is also possible to determine the degree of similarity between images. This concept will be utilized to implement the recommendation system.

An essential aspect is ensuring that the application development follows the software engineering practices. This includes developing the architecture before developing the actual code. With this approach, it is possible to develop a more concise and consistent code structure. There are two important characteristics that the application must have. Firstly, the application must function correctly. This includes having the least amount of operation errors. Secondly, the application should run on all operating systems for mobile phones. For the success of the application it should also be appealing visually and be intuitive.

1.1 Motivation

By creating this project, it is a good opportunity to unleash creativity, improve problem solving skills, and craft technical skills to specific needs and challenges. Through this project, it is possible to apply some of the knowledge learned during the master's degree and learn many other things.

This presents a good opportunity to refine the knowledge in the context of a real world project. This aligns with current market demand for user-centered mobile solutions to ensure the project relevance and potential impact. To implement the application it is possible to delve into the latest technologies that are shaping the tech landscape. The development of mo-

bile applications exposes problems such as: the complexity of the design of user interfaces, responsiveness, and best performance across multiple devices.

Moreover, the project combines machine learning techniques, mobile applications development and the implementation of `REST APIs` [2]. This creates additional complexity, which is common in the current projects. In essence this project goes through four different fields: software architecture, mobile applications, machine learning and web services. The combination of these elements enhances the proficiency in each domain but also creates a comprehensive understanding of how each domain integrates in a bigger solution.

1.2 Contributions

The contributions created in this project are essentially an e-commerce application and all the artifacts created before that. The contributions are:

- **Software architecture**: this contribution is essential for organizing and structuring the system. It helps create a scalable and maintainable system. It serves as living documentation, capturing design decisions and helping maintain a shared understanding of the system.
- **Figma prototype**: this contribution helps create an interactive prototype that simulates the user experience. This prototyping ensures consistency across different screens and components. This can be used to test with users to gather insights about user interactions and preferences.
- **Backend implementation**: the backend must be implemented in order for the application to work. This implementation is done using `NodeJS` and `ExpressJS`. The backend is responsible for receiving requests from the user and give the appropriate responses using `REST`.
- **Frontend implementation**: the frontend is the part where the user interacts with the application. This implementation was made only for mobile. The technology used is `React Native` to create a multiplatform application that functions both on Android and iOS.

- **Recommendation system implementation**: this contribution was created to increase the value of the application, with a new way to recommend products to the user based on images. This was done using machine learning with `TensorFlow` and `Keras`. The programming language used was Python. A method for the server to obtain the recommendations is necessary. To achieve this, an endpoint was established using `Django`.

1.3 Document organization

The document is organized into several key chapters, each focusing on different parts of the project. The following list describes each chapter and topics covered within.

- **Related work**: this chapter provides an overview of existing projects that are closely related to the project developed. It serves to contextualize the project, highlighting the key findings of other related works.
- **Proposed Model**: the proposed model encompasses both software architecture and user interface prototype. The main focus here is on software architecture, which outlines the system's structure. The user interface prototype provides a visual representation of the layouts and navigation.
- **Model Implementation**: this chapter outlines the implementation steps taken to create the final product. It covers the backend development, security measures, recommendation system integration and frontend design and implementation.
- **Validations and Tests**: the validations and tests chapter evaluates the implementation of backend, the accuracy of the recommendation system results, and the responsiveness of the mobile application's design.

The link to the code of the project is in the following GitHub repository: GitHub repository. The repository is divided into branches, each branch corresponds to a different part of the project.

Chapter 2

Related work

Before starting developing the architecture of the project it was necessary to search for products currently on the market. This analysis serves as a foundation to understand the prevailing solutions, strategies and features. The objective is to gain insight of the different functionalities and approaches adopted by the industry leaders.

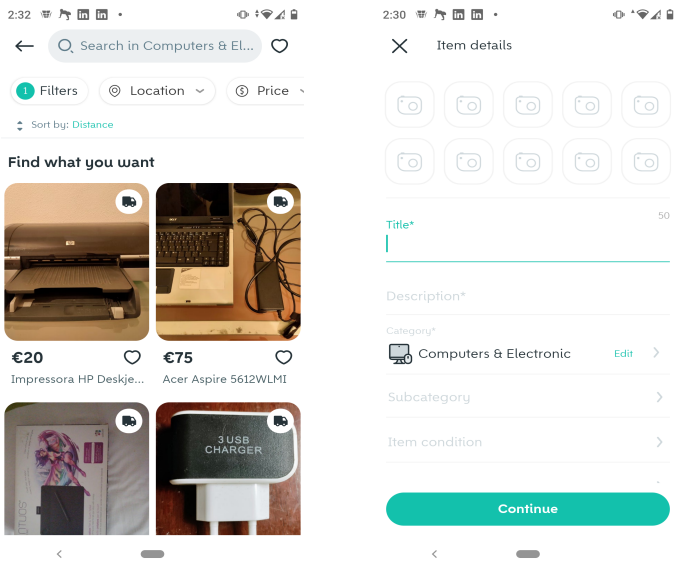
2.1 Mobile application Wallapop

As mentioned before, the main objective of the application is to build an e-commerce application. The application that looked the best in terms of user interface and user experience is Wallapop. Therefore, this application will be used as a guideline to create the screens.

Wallapop [3] is a peer-to-peer marketplace which stands out for its simplicity and intuitiveness. The platform has a really strong user-centered interface. It makes a big effort to ensure that users can buy and sell items with the minimum effort. The emphasis on simplicity does not compromise on functionality, instead it enhances the overall user experience, allowing users to easily connect with one another.

By analyzing the product, the effectiveness and user experience of our solution are enhanced. This allows to identify successful design patterns, user preferences, and both positive and negative user interactions. By examining this implementation, a perspective on how to address common challenges

and leverage solutions will be obtained. This approach will accelerate the development and make strategic choices to create a more robust, user-friendly and competitive solution. Examples of screens from Wallapop are shown in the Figure 2.1.

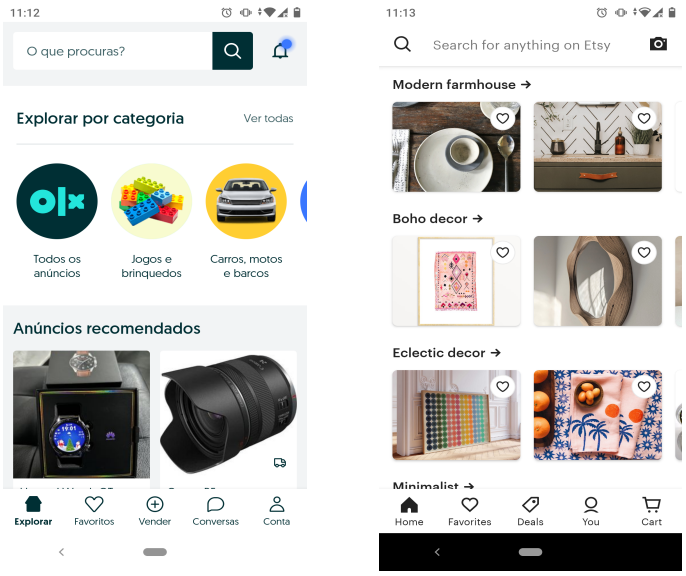


(a) Screen of products from Wallapop

(b) Screen to add a item from Wallapop

Figure 2.1: Example of screens in Wallapop

In addition to this application, other applications were also examined, including Etsy [4], OLX [5], CustoJusto and others. Examples of screens from these applications are shown in Figure 2.2.



(a) Screen of products from OLX

(b) Screen of products from Etsy

Figure 2.2: Example of screens in other applications

2.2 Search similar images with Google images

The proposed application also uses a recommendation system based on images. One well-known application for searching images based on images is Google Images. Therefore, the functionalities of this application will be examined to create a similar system that can be applied to our solution.

Google Images [6] has a functionality that allows users to perform searches based on visual content instead of text. The user, instead of writing keywords, uploads an image, and Google Images uses image recognition to identify and retrieve visually similar or related images from its database.

This type of search is useful for tasks like: search for the source of an image, find higher resolution versions, identify objects and landmarks, and find visually similar images. To accomplish this, it uses a sophisticated computer vision algorithms to analyze the visual characteristics of the uploaded

image. This process involves identifying patterns, colors, shapes and other visual features, creating a unique feature vector for the image. The system then compares the feature vectors with a massive index of images on the internet, searching for matches or visually related images. An example of how the search works for an image is shown in the Figure 2.3.

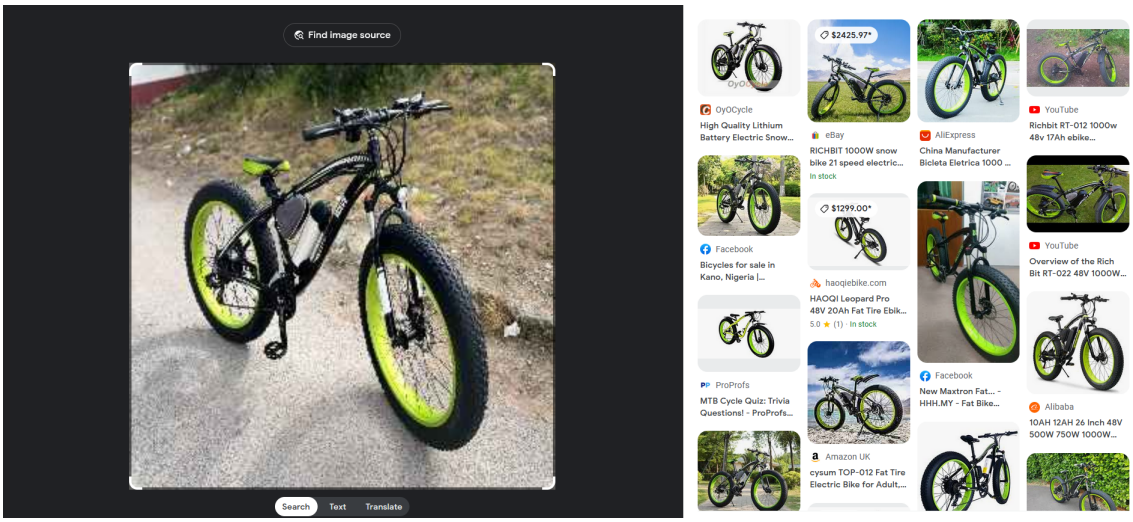


Figure 2.3: Example of search in Google Images

This functionality can be employed in an e-commerce application. Users can upload images of products they are interested in, and Google Images can provide information about the product, including price, reviews, and potential sources for purchase.

Google Images can also be integrated with Google Lens [7], which is a powerful visual recognition tool. This image search enables users to gather information about objects and gain insights into the world captured by the image. It can extract details about famous landmarks, artworks or any identifiable object. This turns the visual search into a comprehensive visual information retrieval tool. An example of the usage of Google Lens for the same image is shown in Figure 2.4.

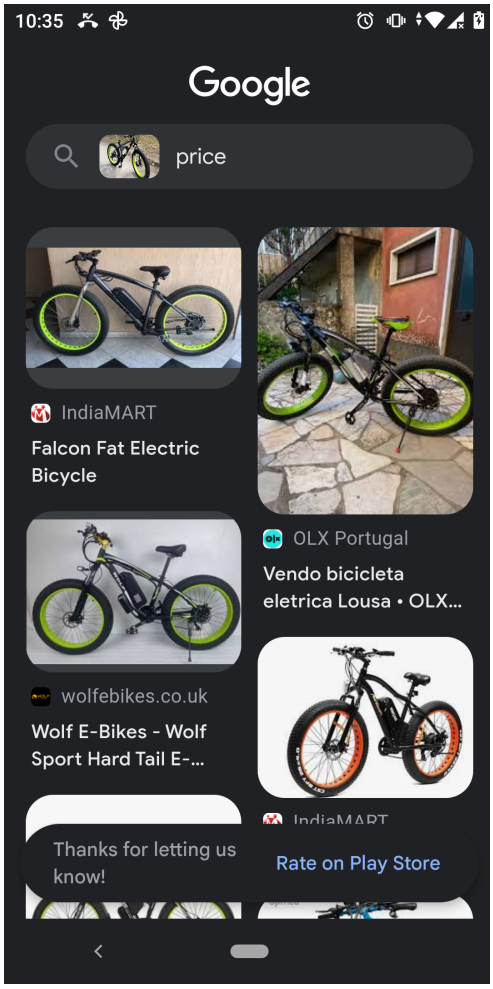


Figure 2.4: Example of search in Google Lens

2.3 CamFind application

Apart from Google Images, there are also other applications that search for images based on images, and one of them is `CamFind`. This application was used in the research to provide a more diverse term of comparison for possible outcomes of image searches.

`CamFind` [8] is an image recognition application created to provide users with information about objects, products and, other visual content using their mobile devices. It provides an advanced image recognition technology

to analyze photographs taken by users and delivers relevant information related to the image. `CamFind` provides users a wide range of objects, including products, landmarks, artwork and more.

The application attempts to identify objects, provide relevant details, and potentially relevant links. This can be particularly useful for users who want to know the origin, features, or purchase options associated with that item. Within the links, it can include online shopping options. The result obtained from testing the application with the same image tested before is shown in Figure 2.5. The result is the text below the image, which reads "*black and green BMX bike*".

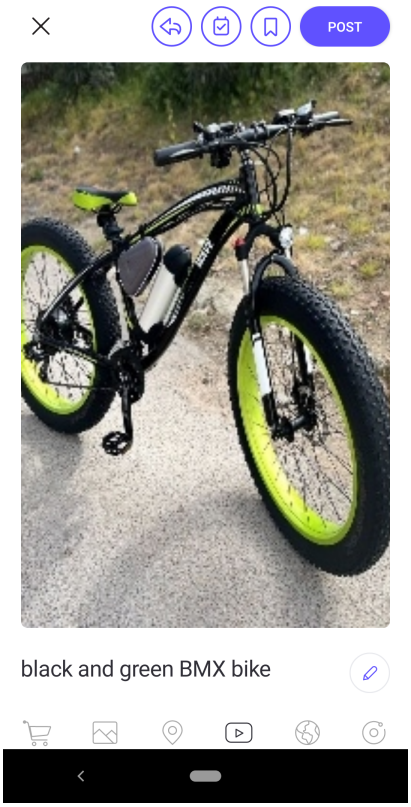


Figure 2.5: Example of search in CamFind

2.4 Image-based Product Recommendation System with CNN

In terms of recommendation systems for image-based products, an approach similar to the one implemented was published in [9], inspiring the proposed model.

This paper presents an approach to enhance online shopping experiences through the development of a recommendation system based on images. It uses the power of neural networks to interpret images from products. By employing `CNNs` the system classifies product images into categories and calculates similarity scores to recommend relevant items. For this purpose it uses models like `AlexNet` and `VGG` for classification. Then the system uses metrics like Jaccard similarity and cosine similarity to quantify similarities accurately.

The dataset used in the test has 3.5 million images, which covers 20 different products. The experimental evaluation for classification ranged from 26.79% to 87.69% with a root mean square error (RMSE) of 0.1524. The results obtained from this paper are shown in the Figure 2.6.

Figure 2.6: Results from the paper

Chapter 3

Proposed Model

In this chapter, the requirements will be discussed for the project as well as the subsequent implementation. In this matter, using the information retrieved from the previous chapter, the software architecture will be created.

3.1 Software principles

The development process of software might vary depending on the system requirements. On this project it will follow the following steps:

- Analysis
- Conception
- Construction
- Verification

In the analysis phase, it is necessary to understand the problem. This involves identifying, analyzing and comprehending the problem. Typically, this is achieved by gathering information from the stakeholders, which involves understanding the needs of the users and defining the features and functionalities that the software should have. However, in this particular case, this approach will not be taken, instead the information is gathered from the current products on the market. The advantage of this phase is that ideas might be ambiguous, incomplete and inconsistent. With this phase the ideas get written making it more concise and objective. This is done through the

vision document.

The conception phase is created based on the analysis and provides a high-level description of the system made using the modeling language UML (*Unified Modeling Language*). A model is an abstract representation of a system to handle complexity. This phase is critical because it defines the overall architecture and structure of the software system. This involves breaking down the system into modules, and specifying their interactions. The conception design delves into the specifics of each module, defining data structure, algorithms and user interfaces. This was achieved through the software architecture.

The construction phase is when the developer or developers write the source code based on the conception phase. During the development they should follow the standards of the chosen programming language. Normally the code is reviewed by peers. Some optimizations can be applied for performance fine-tuning. This phase marks the crucial transaction from design concepts to a working software product, paving the way for testing and deployment stages. This phase corresponds to the whole Model Implementation chapter.

The testing phase is used to ensure the quality of the software. It begins with the unit testing, where individual components are tested in isolation. Integration testing follows to verify the interaction between integrated components. In this phase, there are many more tests to verify metrics such as performance, security and others. These tests were not conducted with a high level of detail due to the size of the application and the amount of time it would take.

3.2 Vision document

3.2.1 Description

The vision document [10] is a strategic document that describes the overall vision of the project in general terms. This includes what is the operational context, objectives of the project, organization and what are the parts in-

volved and characteristics. It serves as a high-level road-map, providing clear and concise descriptions of what the project aims to achieve. The vision document typically includes information about purpose, target audience, key features, and the outcomes. This document is often created at the beginning of the project and serves as a reference throughout the project.

3.2.2 Execution

The creation of the vision document was made using *Word*, which is an editor for *docx* documents. Since this document should be simple, it was divided into paragraphs. Initially, it describes that the project will be an online store market, where users can sell and buy products / items. This store will also have a system that can search for similar products based on images.

The most important parts of the document were the execution environment and characteristics summary. The diagram of the execution of the system is shown in the Figure 3.1.

Figure 3.1: Execution environment

This was the initial scenario created. The scenario shows two different users one that posts products to sell and another that views the products and buys them. This scenario does not cover all the possible interactions of the user, serving only as a guideline. This will be developed into more realistic scenarios.

The characteristics summary, as the name says, represents the characteristics that the system will have. The characteristics obtained from the document are:

- The user can create an account on the application.
- The user can logout of the current account logged on the application.
- The user can see the products that are being sold on the app.
- The server stores the most recent products viewed.
- The app provides the user with recommended products based on the searches done on the user side.
- The user can search products based on the category.

The characteristics summary is a really long list, so it is shown a small part of it. The rest of the characteristics will be in the vision document. This document is in the Appendix .3.

With this information, it is now possible to start creating two different things. The first one is the user interface, because the overall characteristics are defined. Simultaneously, the requirements specification can also be developed. In the case of this project, the user interface was created first, because it is a mobile application. This approach makes it easier to develop the requirements specification, because the user interface is already created and the interactions can be visualized.

3.3 User interface

The user interface refers to the point of interaction between a user and a computer or software application. Its primary goal is to facilitate effective communication between the user and the system, ensuring that users can easily understand, navigate, and interact with the software. A well-designed user interface enhances the user experience by proving a visually appealing, intuitive, and efficient way for users to accomplish tasks and access functionalities within the software or system.

3.3.1 Prototype of user interface

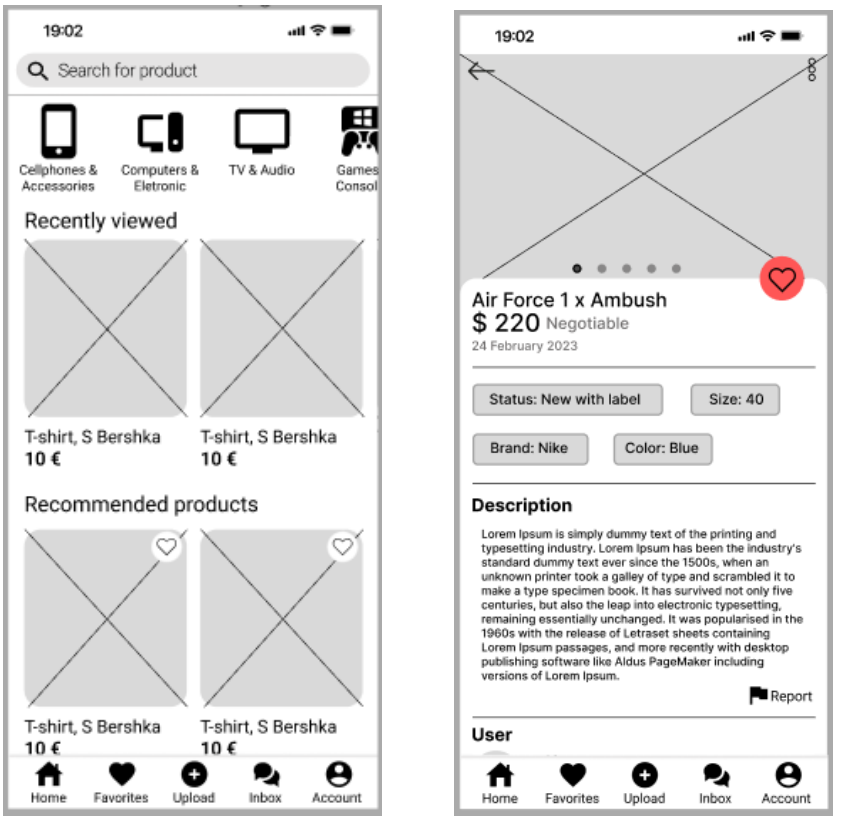
The user interface when built should follow the UCD (*User-Centered Design*), which is an approach focused on the end-user needs and preferences

made throughout the design and development process. This normally involves iterative cycles of design and evaluation, incorporating feedback from users at various stages. Although this process is very useful to enhance the user interface, correct usability and user experience errors, this approach was not applied during the prototyping of the user interface because it was not the focus of the project.

3.3.2 Wireframes

Initially the user interface was built using wireframes. The wireframes are skeletal, low-fidelity representations of a digital interface that serves as a visual guideline for the layout and structure. They focus on key elements, such as navigation, content sections and interactive features, without detailing the visual aesthetics or graphic design aspects. The primary purpose of wireframes is to map out the spatial arrangement and flow of elements within a user interface. They should align with the expectations of the person who is doing them, since it is not evaluated with users. The wireframes were created based on the functionalities written on the vision document, by analyzing related applications and introducing new ideas.

To develop the wireframes, the Figma tool was used because it has a user-friendly interface. While using Figma, the navigation on the prototype was also implemented, making it closer to the final result. The result of the wireframes consists of 24 different screens excluding the pop-ups that appear when the user clicks on a button to confirm an action. These pop-ups are essential because they may prevent mistakes made by the user, for example removing a product from the favorites. Because the wireframes have too many pages, it is not possible to display all of them. Instead, two pages are shown and the rest are available through the Figma link. The outcome of two wireframes is shown in Figure 3.2.



(a) Wireframe of Homepage (b) Wireframe of Product Details

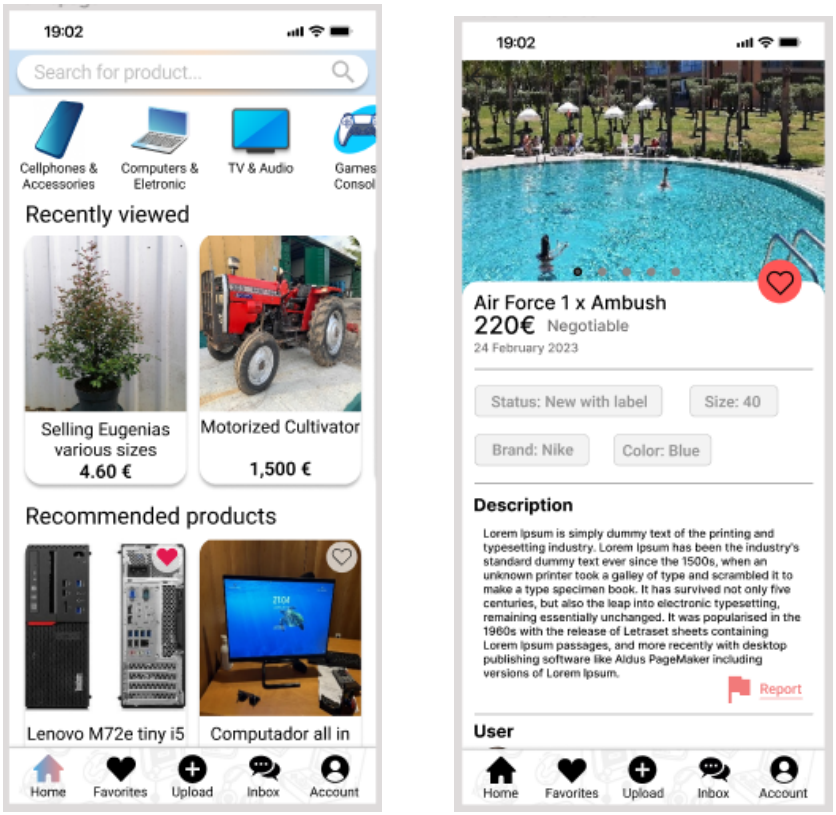
Figure 3.2: Examples of wireframes

3.3.3 Mockups

The mockups are the second phase of user interface development. They are a visual representation with high-fidelity, simulating the appearance of the user interface before it is fully developed. Created after the wireframes, the transaction between wireframes to mockups allows the designer to visualize the design aesthetics, color schemes and typography. Mockups facilitates a more realistic user experience evaluation, helping identify potential issues and contributing to a more successful user-friendly product.

The color scheme selected is mainly blue and yellow, sometimes using both in a linear gradient. The typography used was mainly `Roboto`. The mockups suffered some modifications during the development of both the

software architecture and the development of the application. The outcome of the mockups from the previous wireframes is shown in Figure 3.3. The link to the rest of the work is in Figma link.



(a) Mockup of Homepage (b) Mockup of Product Details

Figure 3.3: Examples of mockups

3.4 Requirements Specification

The requirements specifications refers to a comprehensive document that outlines the functional and non-functional requirements of a software system. This document provides a clear understanding of what the software is expected to achieve and how it should perform.

The document can include use cases, user stories, diagrams and other artifacts to illustrate the system expected behavior. In this project, the use

case approach will be followed. Using a top-down approach, where the starting point is the vision document, then is developed the boundary between the system and the actors. In this regard, the resulting general case for the use cases is shown in Figure 3.4.

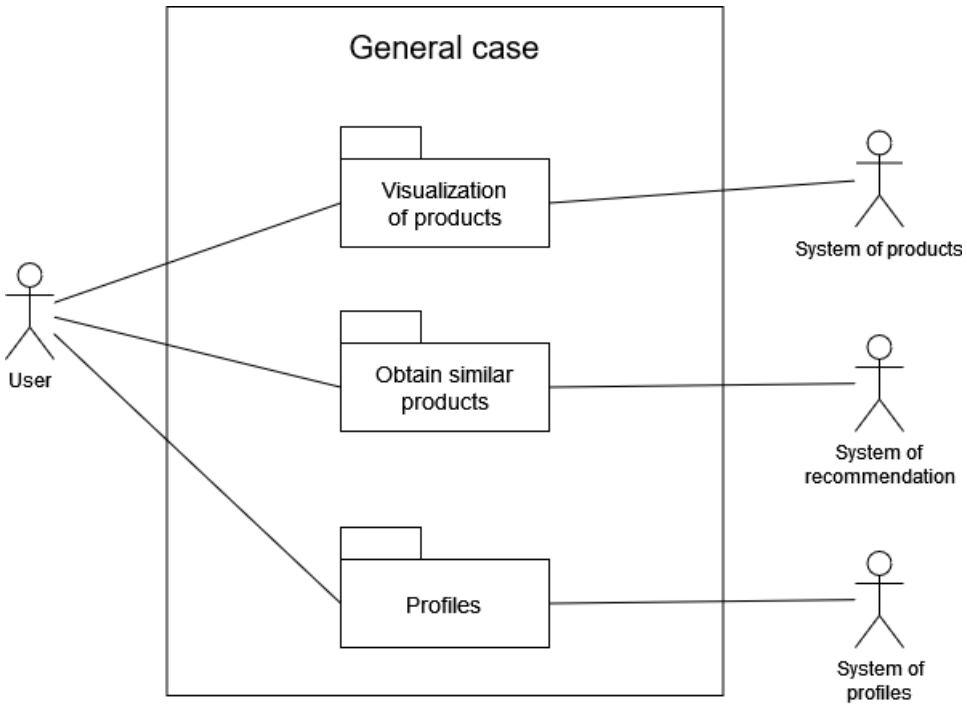


Figure 3.4: General case of use cases

In the Figure 3.4, the box called `General case` represents the boundary of the system, limiting what is the system responsibility and what is not. The interior of the boundary is the system responsibility and outside represents the actors that interact with the system.

After defining the general case, it is now possible to develop the use cases. The use cases [11] are a detailed description of how a system interacts with an external entity. In reality, the use cases are the realization of objectives from the actors, which means that it is the way the system responds to events from the actors. The use cases created are very long and extensive, making it hard to follow them all. As an example, Figure 3.5 shows one use case called "`View product details`", which will be the focus of this document.

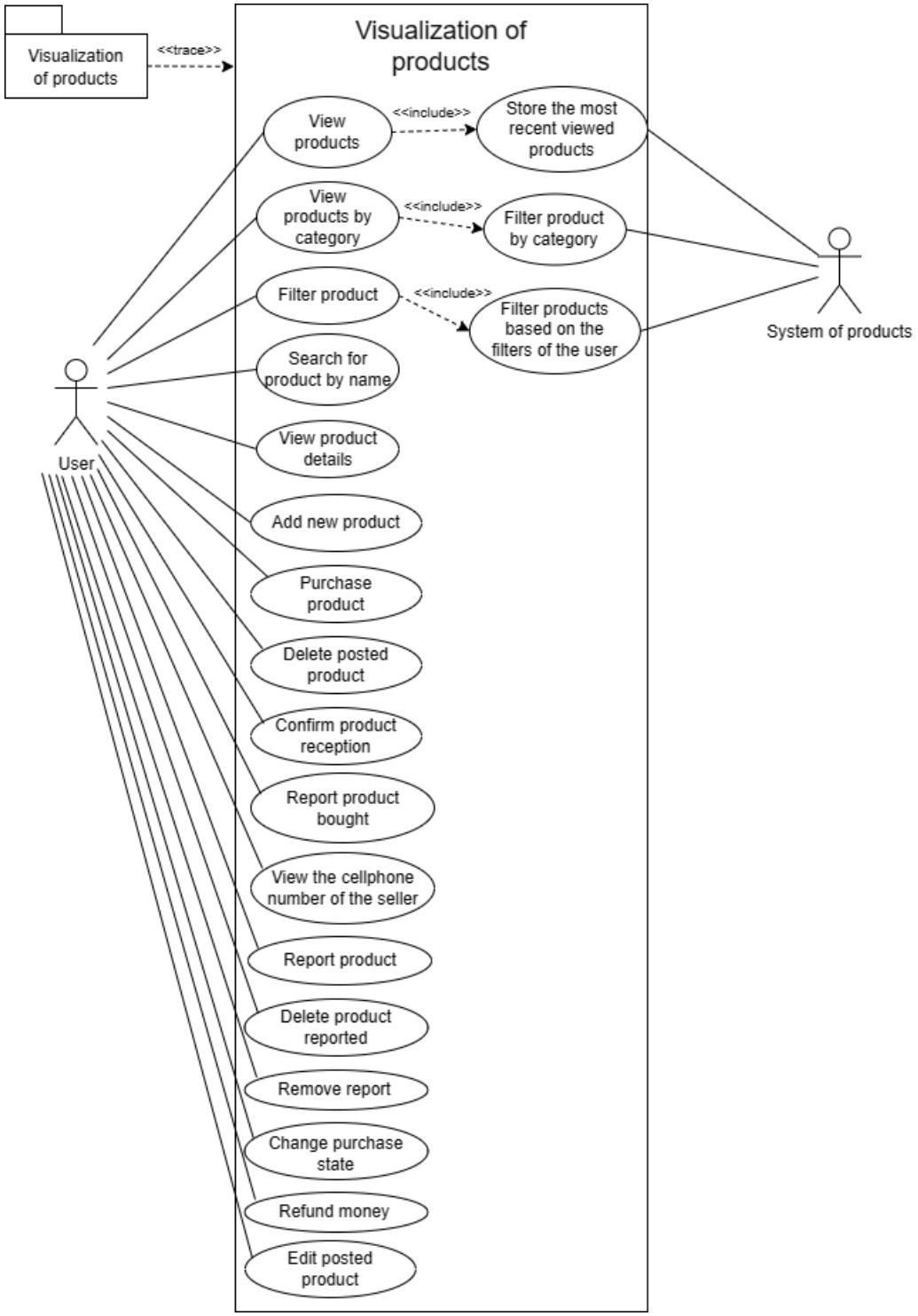


Figure 3.5: System of products use cases

After defining the use cases, it is now necessary to describe them. The description of the use cases has the objective of describing the main event flows and alternative flows of a specific use case. Normally, this also includes information about the actor involved, main flows and alternative flows, and conditions. The Figure 3.5 contains numerous use cases, but as mentioned before, the focus is on the "`View product details`" use case. The description of this use case is shown in Figure 3.6.

4.1.5 View product details

Summary: The use case allows the user to see all the images, details, contact the seller, or buy the product.

Actors: User, System of products

Pre-conditions: The user is already logged in to his account.

Principal Scenario:

1. The use case starts when the user clicks on a product shown on the homepage or when the products are being filtered.
2. The user interface sends a request, with and id of the product, to receive all the content from the product selected to the system of products.
3. The system of products obtains all the content of a product.
4. The system of products stores the recent viewed product.
5. The system returns all the information obtained to the user interface.
6. The user interface uses the information to show to the user.

Figure 3.6: Use case description

From the Figure 3.6, it is now possible to understand how the use case will work. In this particular case, the use case has only one scenario: the user clicking on a product, following the inherent flow of the system until the product is shown on the screen. Additionally, this use case shows a summary, what actors participate on the use case and which conditions must be completed before reaching this use case. All these components mentioned were applied to all the use cases, making a clearer understanding of the behavior of the system.

In the requirements specification, there are also two important fields: the supplementary specifications and the glossary.

- The **supplemental specification** typically addresses aspects that might not be easily captured in the primary requirements. This typically includes constraints, performance and non-functional aspects.

- The **glossary** gives a list of terms and their definitions, particularly those that are specific to the domain of the software being developed. This document gives a common understanding to the terminology used on the project.

3.5 Logical Architecture

The logical architecture is built from the description of the use cases, with the glossary and the supplemental specification. The logical architecture is a high-level design that focuses on functionalities instead of implementation details. It defines the system organization in terms of logical entities and their interaction, emphasizing the relations between multiple components. This architecture is built using `Unified Modeling Language` (`UML`). The result of the logical architecture is models, and each model represents an abstract representation of a system. Those abstract representations are used to handle complexity. These models are very useful to develop software, because software development is a complex task.

When developing the modulation of a system, there are three levels:

- **Independent of the computational model**: describes the business model, the context of system use, its behavior, and expected characteristics. This architecture was implemented through requirements analysis.

- **Independent of the execution platform**: describes the system in as much detail as possible independently of the execution platform. This level is known as logical architecture, which will be developed in this chapter.

- **Specific to the execution platform**: describes the realization of the system for a specific platform. This level is known as detailed architecture and will be developed in the next chapter.

When developing code, there are multiple paradigms. In this particular project, the object-oriented paradigm will be used. In this paradigm the system is modulated as a set of objects which interact with each other to produce the final result.

3.5.1 Domain model

The domain model [12] is a conceptual representation of key concepts, entities, associations and rules of the domain. This model serves as a visual structure of the real-world environment that a system is intended to address. With this model, it becomes possible to create the entity classes, with their attributes. Additionally, it includes associations that represent relations between entities.

The components of this diagram include:

- **Entities**: representations of objects, concepts or things within the domain.
- **Attributes**: characteristics or properties that describe the entities.
- **Relationships**: connections or associations between different entities, indicating how they interact or relate to each other. Relations add context to the model.
- **Business rules**: constraints that define how the entities behave within the domain.

To identify this information, it is necessary to identify the elements, where names can represent classes, attributes and objects. The verbs represent relations. The result of the domain model is shown in Figure 3.7.

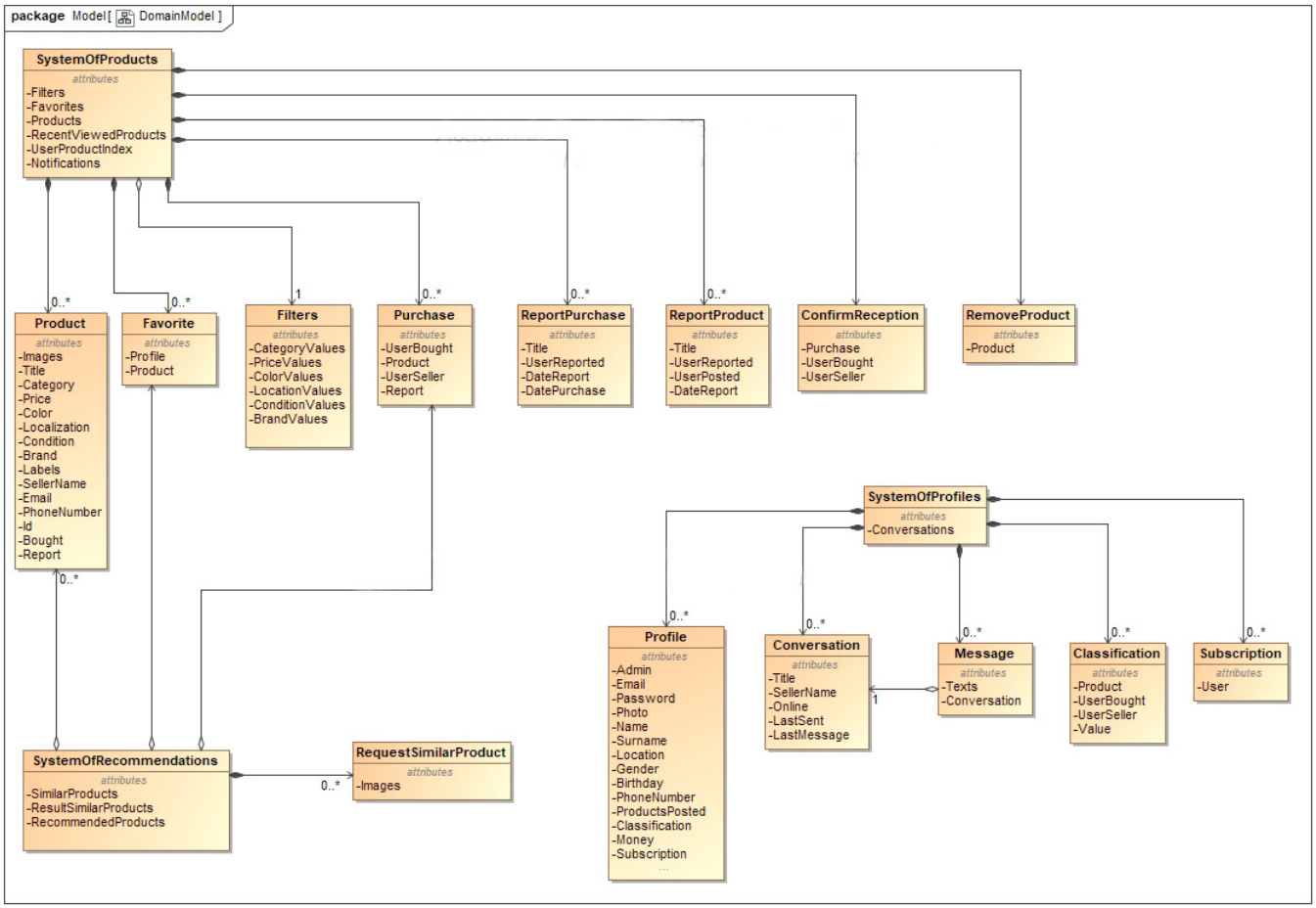


Figure 3.7: Domain model

With Figure 3.7, it is possible to see entities and who operates them. The operators of the entities always have the name '`System`', and the entities are all the others. It is also possible to see the attributes, which are the values inside each entity. The relations are the lines connecting the entities to the `Systems` or to other entities. With this diagram the system is starting to gain shape, and it is now possible to see that there are three operators, and which entities they operate. The `SystemOfProducts` is responsible for all the entities related to the products. The `SystemOfProfiles` is responsible for all the information about the user. Lastly, the `SystemOfRecommendations` is responsible for the recommendations to the users.

3.5.2 Interaction diagrams

An interaction diagram is a diagram that illustrates how objects collaborate to accomplish a specific behavior. In the case of this project, only sequence diagrams were employed. The sequence diagram [13] is a type of interaction diagram that focuses on the chronological sequence of messages. It illustrates the interaction with various elements in a system in a step-by-step manner, emphasizing the order in which messages are sent and received.

When developing the sequence diagrams, there must be selected a pattern used for the design of the user interface. There are three different patterns:

- MVC – Model-View-Controller.
- MVP – Model-View-Presenter.
- MVVM – Model-View-ViewModel.

They provide a set of principles and best practices to address common design challenges and promote maintainability, scalability and flexibility. In this project, the MVVM pattern will be exclusively utilized. This pattern gives: data binding, separation of functionalities, and the introduction of a dedicated component (ViewModel) to manage the presentation logic.

In this project, the Domain-Driven Design (`DDD`) principles were implemented. This approach on software emphasizes the importance of understanding and modeling the problem. It incorporates strategic design principles for organizing large-scale systems and refining the structure of individual components. In this matter, for the server side it was used:

- **Services**: Services encapsulate business logic or operations. They are responsible for coordinating activities between multiple entities or carrying out complex operations.
- **Repositories**: Repositories provide a way to retrieve and store entities from and to the database. Repositories act as a bridge between the domain model and the database, which is called the data access.

- **Model**: Model is a logical object oriented representation of the data that an application works with. In this particular case, it corresponds to a database table. Each instance of the model represents a record in the database table, and the attributes corresponds to the fields in the table.

- **DTO**: DTOs (Data Transfer Objects) are used to transfer data between different layers of an application, such as between the frontend and the backend.

The resulting diagram of the `View product details` is Figure 3.8. The diagram shows the chronological sequence of messages between all the services, views and repositories using the entities. It provides more details than the use case description. This is because the sequence diagrams were updated while developing the code, ensuring a direct correspondence with the implemented code.

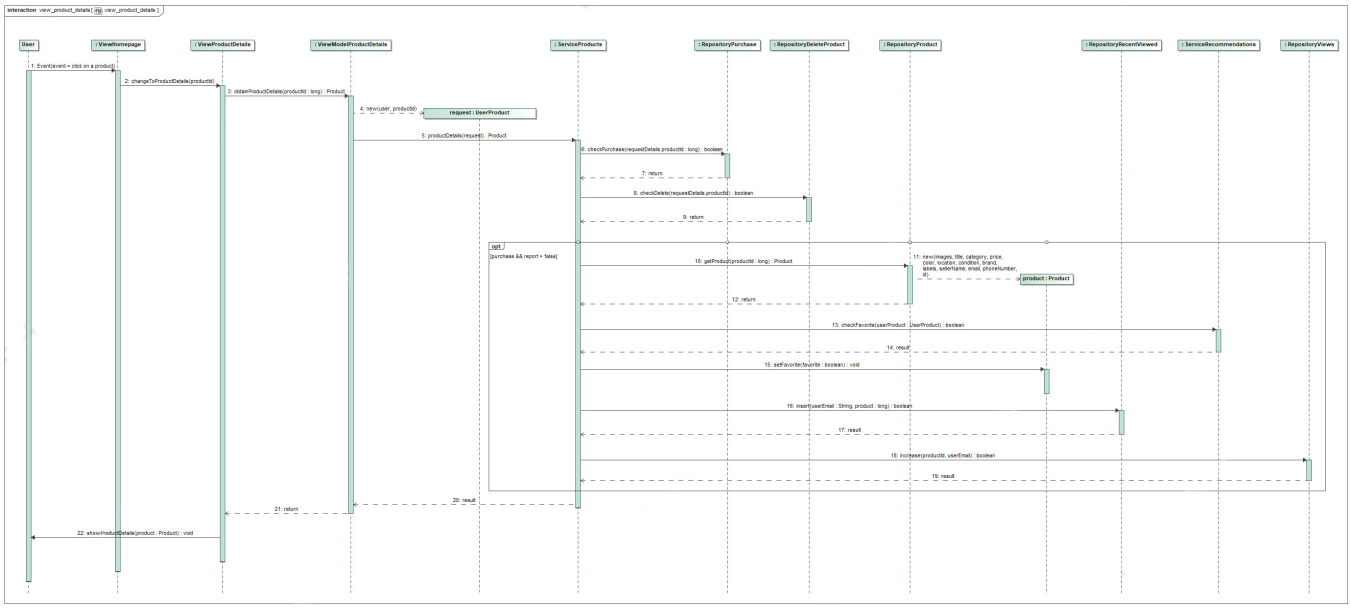


Figure 3.8: Sequence diagram of view product details

The sequence diagram has a bidirectional organization where the vertical axis represents the time and the horizontal axis the structure. In this diagram there are four main elements: lifeline, activation bar, message and operator.

The lifeline represents the time evolution and the activation bar represents the execution of the operations. The message consists of DTOs and models. In this project, the models are the database entities. The operators have a specific semantic representing optional elements such as loops or optionals. Although this diagram is quite extensive and could be divided into smaller pieces, the project already had many diagrams, so it was preferred to use a bigger one. It was built one sequence diagram for each use case created.

3.5.3 Class diagrams

A class diagram [14] represents the static structure of a system by showing the classes in the system, their attributes, methods, and the relationships between them. The key components of the class diagram are:

- **Class**: a class represents a blueprint for creating objects. It encapsulates attributes and methods.
- **Attributes**: they represent the data that a class holds. Each attribute has a name and a data type.
- **Methods**: methods represents the behavior or operations that can be performed by an object of a class.
- **Relationships**: relationships between classes are done using lines connecting them. Common types includes associations, aggregations, composition, inheritance and dependencies.

The relations can be:

- **Associations**: represents a connection between two classes. It may include a multiplicity to indicate the number of instances involved.
- **Aggregation** and **Composition**: illustrates relationships where one class encompasses another, indicating a structure of whole and part. Aggregation implies a weaker relationship, while composition implies a stronger relationship.
- **Inheritance**: represents an is-a relationship, where one class inherits attributes and behaviors from another class.

- **Dependency**: represents a relationship where one class depends on another but is not part of the structure of the system.

The class diagrams obtained were divided into two different categories. The division was made in front-end and back-end. The back-end was further divided into services. This approach makes the architecture more readable and reduces entropy. In total, there were six different services, therefore six different class diagrams were created for all the back-end. The front-end was also divided, where each diagram represents a page on front-end, meaning each view and modelView has its own class diagram. The resulting diagrams of the use case `'View product details'` are shown in Figures 3.9 and 3.10. In the diagrams, the DTOs and models used by each service and view are also displayed.

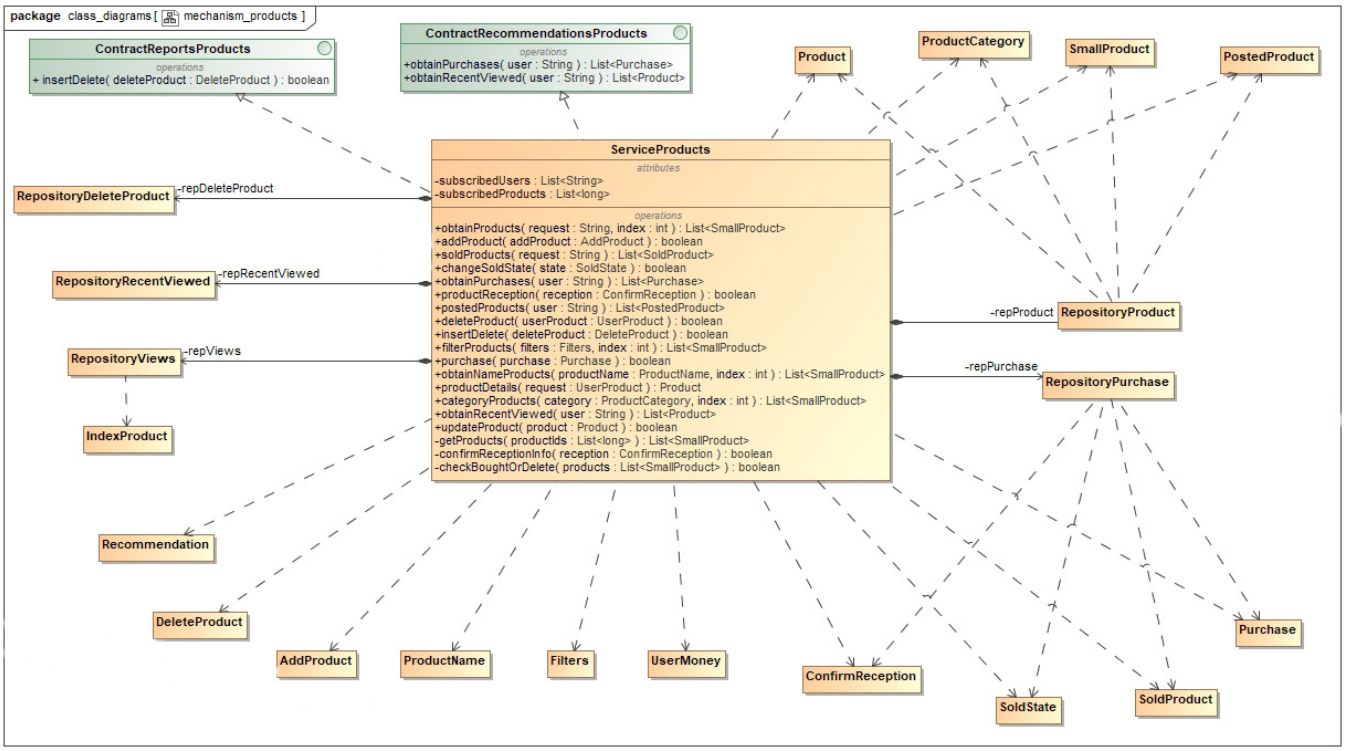


Figure 3.9: Class diagram of products service

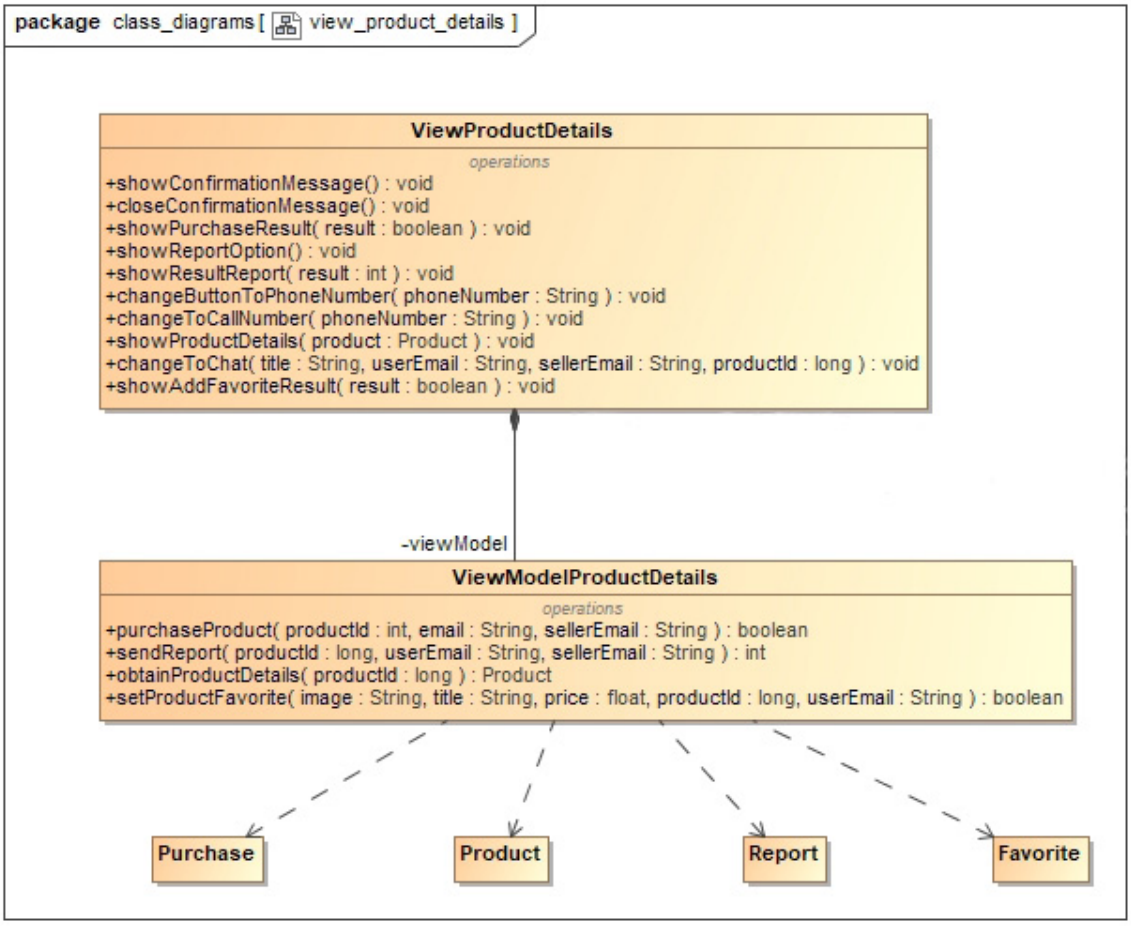


Figure 3.10: Class diagram of view of product details

The resulting services obtained from all the class diagrams are:

- **Products Service**: this service is responsible for all the functionalities of the products. These functionalities include information about the products, which products the user viewed, the purchases, the products deleted, and number of views of each product. Each product contains: images of the product being sold, the title, category, price, who is selling and the phone number of the person who is selling. Optionally, the product contains information about the location, color, condition, brand and other relevant information to the product.

- **Profiles Service**: this service is responsible for handling information about the profiles of users and conversations. This profile information

includes the credentials for the user to login, the address to receive the products, the demographic information, and information about whether the user is an administrator. The conversation stores information about who participates in the conversation, which product the conversation is related to and the messages of the conversation.

- **Recommendations Service**: this service has the responsibility of handling the favorite products, the recommendations and the similar products. The recommendations are products that might be interesting to the user. The similar products are products being sold similar to the images sent by the user. This service stores both the images sent by the user as well as the results of the similar products.

- **Calculate Recommendations Service**: this service is a sub-model of the recommendations service. This service is responsible to communicate with the algorithm that calculates both the recommendations and similar products.

- **Money Service**: the service is responsible for managing the money on the accounts of the users in the application. This includes adding money on the account, withdrawing money, increasing and reducing the money when a purchase is made, and other functionalities.

- **Reports Service**: this service is responsible for managing two types of reports. The reports of products and reports of purchases. The reports of products are made to prevent illegal items to be sold on the application, the admin can delete the product or can remove the report. The reports of purchases are made by the user who bought a product. He can report the purchase if the product did not arrive in a certain amount of time. The admin can refund the money to the buyer, or can send a message to the buyer to figure out what happened. This service is only accessible if the user is an admin, otherwise it will return the information always empty.

3.5.4 Subsystems architecture

A package model is an overall representation of the organization and structure of packages in a system. A subsystem is a logical group of related

packages that together form a higher-level unit. Subsystems will be represented using packages. These diagrams are created using the class diagrams, which represent a higher level of abstraction. The subsystem packages show how the system works as a whole, and what the subsystem's relations are. For the solution of this project, the three model layers were utilized. In this case, the services and entities are stored in the domain, the repositories in the data access, and the views and viewModels are stored in the presentation.

The resulting higher-level subsystems are shown in Figure 3.11. Each subsystem will have all the classes created previously in the class diagrams inside it.

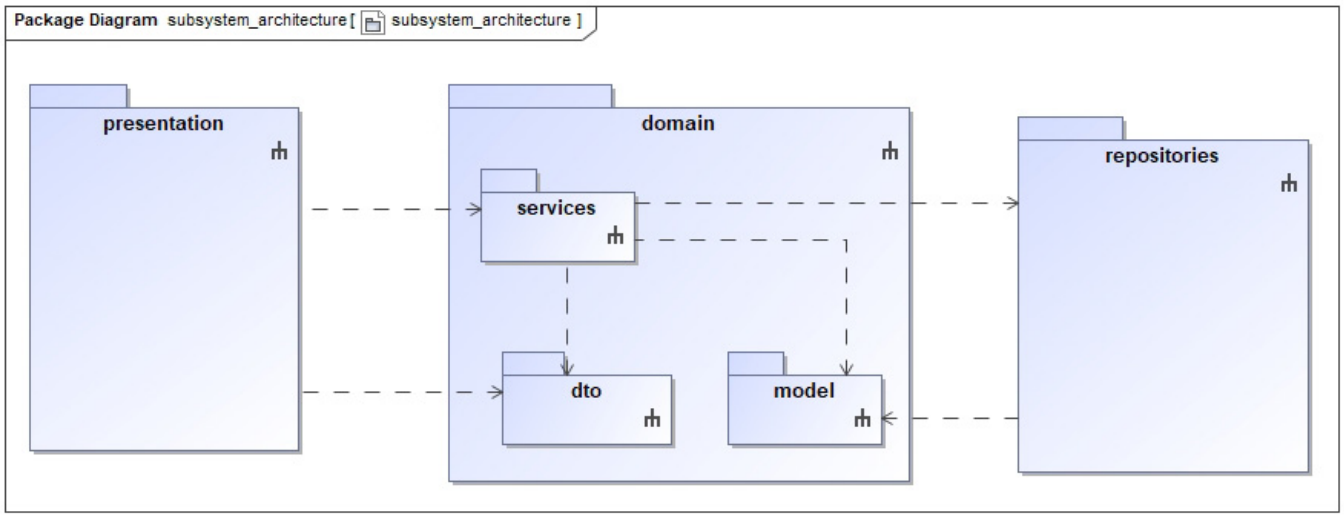


Figure 3.11: Subsystem architecture

The resulting subsystem architecture was divided into six different diagrams. The diagrams were divided into: presentation, domain, services, DTO, model, and repositories.

An important detail when implementing the subsystem architecture is the fact that the presentation subsystem should never access the model from the domain. If that happens, it is a signal of failure in the design of the system. The end-user should never know what information is stored on the database, creating possible security risks and making it easier to malicious users to manipulate the database.

3.6 Detailed Architecture

The detailed architecture refers to a low-level design of the software system. This architecture refers to the process of concretization to a specific platform.

3.6.1 Dynamic model

The dynamic model, also called state machine diagram [15], is a representation of the system behavior over time. It focuses on capturing the dynamic aspects of the software, illustrating how different components interact and collaborate during run time.

- The structure shows the parts and the relationships between parts of a system.
- The dynamics show how the parts and relations of parts evolve over time.
- The structure and dynamics together form the behavior that represents how a system acts in response to stimuli from the environment.

This model was used in two different circumstances. The first one is to search for a product using text provided by the user. This search will be divided into three different steps: search by title, search by labels, and search by location. The second one is the algorithm used for searching products to show in the homepage. The second diagram has several states. The sequence is: firstly it shows the recommendations, then it searches for products where the user that posted is subscribed. After that, it searches for products by number of views and finally checks if the products has been purchased or deleted to remove them. To facilitate the interpretation, it is only shown the search for product by text in the Figure 3.12.

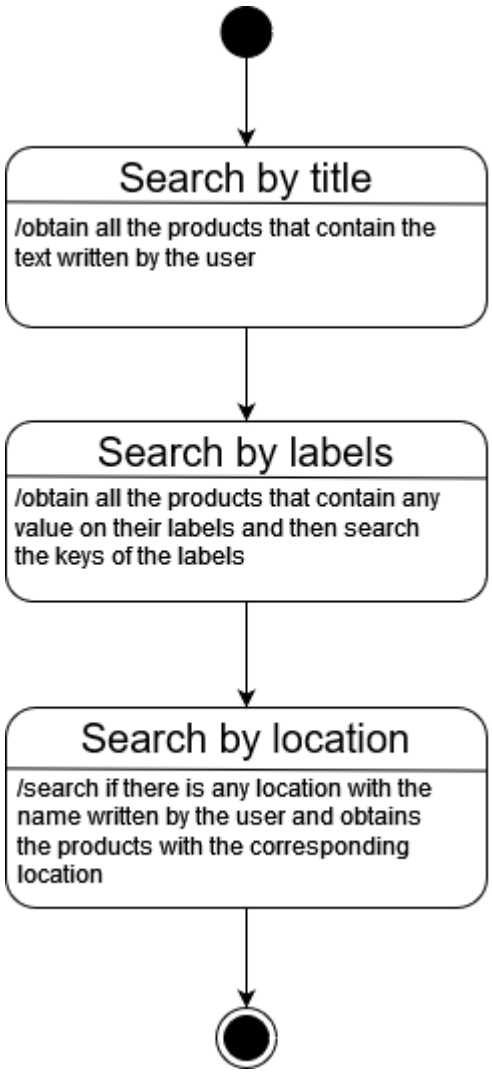


Figure 3.12: Dynamic model of search for product by text

3.6.2 Test architecture

The test architecture is a systematic design and organization of the testing processes for the components within a software system. This layer aims to test the domain layer. Inside this architecture is where the tests take place for the different use cases. The higher result of the test architecture is shown in Figure 3.13.

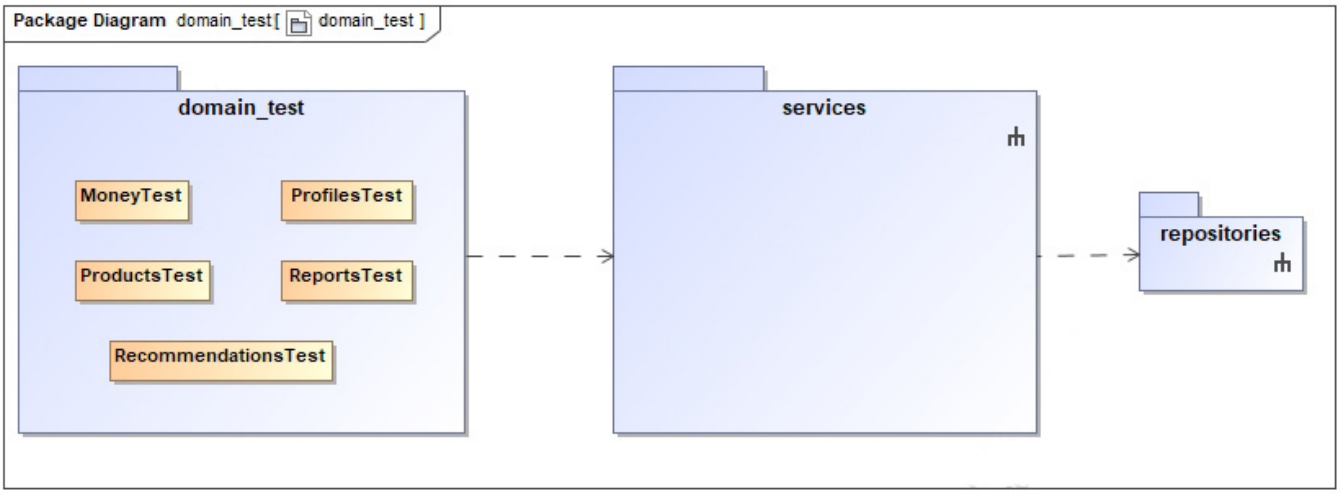


Figure 3.13: Subsystem of test architecture

In this project, five main services were created. The services are: the service of products, the service of profiles, the service of recommendations, the service of money and the service of reports. For each service, two separate diagrams were created to test the functionalities. The first one is a sequence diagram illustrating how the classes will be created and executed. The second one is a class diagram of the classes that will be instantiated to execute the tests. In each '`Test`' class, it calls the method name on the service but with '`Test`' at the end. With this approach is possible to verify if the services are working properly.

3.6.3 Technology Stack

This type of diagram provides a visual representation of the various software components, tools, frameworks, programming languages, and infrastructure elements used to build and run a software application. The resulting technology stack is shown in the Figure 3.14.

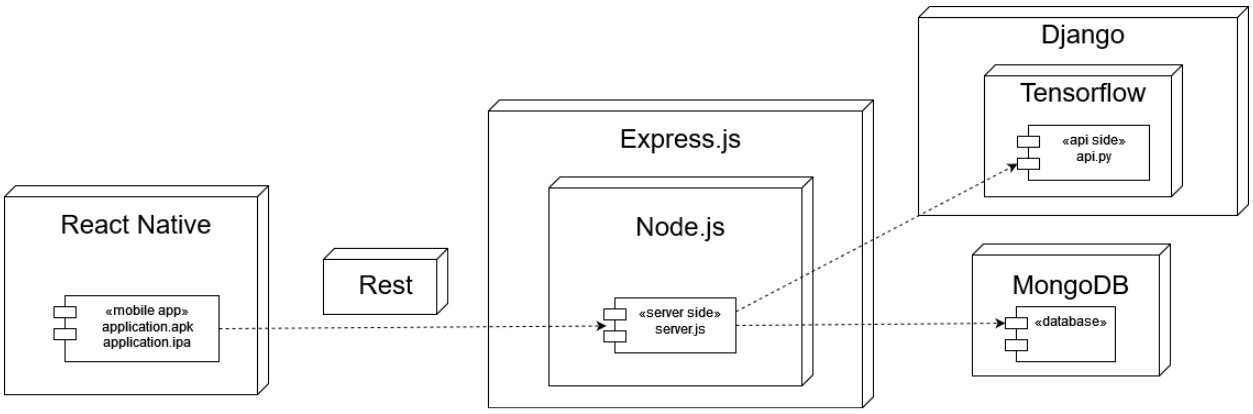


Figure 3.14: Technology stack

The reasons to choose those technologies are:

- React Native: is a cross-platform that enables the development of mobile applications that can run on both iOS and Android platforms. It has a component based architecture, enabling the reuse of code across platforms, contributing to a more efficient development process.

- NodeJS: is a server-side JavaScript runtime engine. It allows developers to execute JavaScript outside of the web browser. It creates scalable and high-performance web applications, making it particularly popular for developing backend applications.

- ExpressJS: is a minimal and flexible web application framework for NodeJS. It simplifies the handling of HTTP requests and routing. Together, NodeJS and ExpressJS form a powerful combination for building modern, scalable web applications.

- MongoDB: is a NoSQL database that provides flexible, scalable data storage solutions. Its document-oriented structure allows for easy representation of complex data. With the advantage of scaling horizontally by sharing data across multiple servers.

- Django: is a powerful and flexible framework to serve as an endpoint for obtaining information from a neural network created using Tensorflow.

The advantage of using NodeJS and ExpressJS compared to using Firebase lies in the flexibility and control over the server side. They allow the construction of the application as needed. They provide the ability to choose the database that fits the project requirements and control over the scalability of the application. Another consideration is the vendor lock-in, the application starts to be heavily dependent on Firebase services, making a migration almost impossible. This approach is normally the best one for big applications due to its flexibility.

Chapter 4

Model Implementation

In this Chapter, it is explained in high-level detail how each block was implemented. The block diagram resulting from the architecture is represented in Figure 4.1.

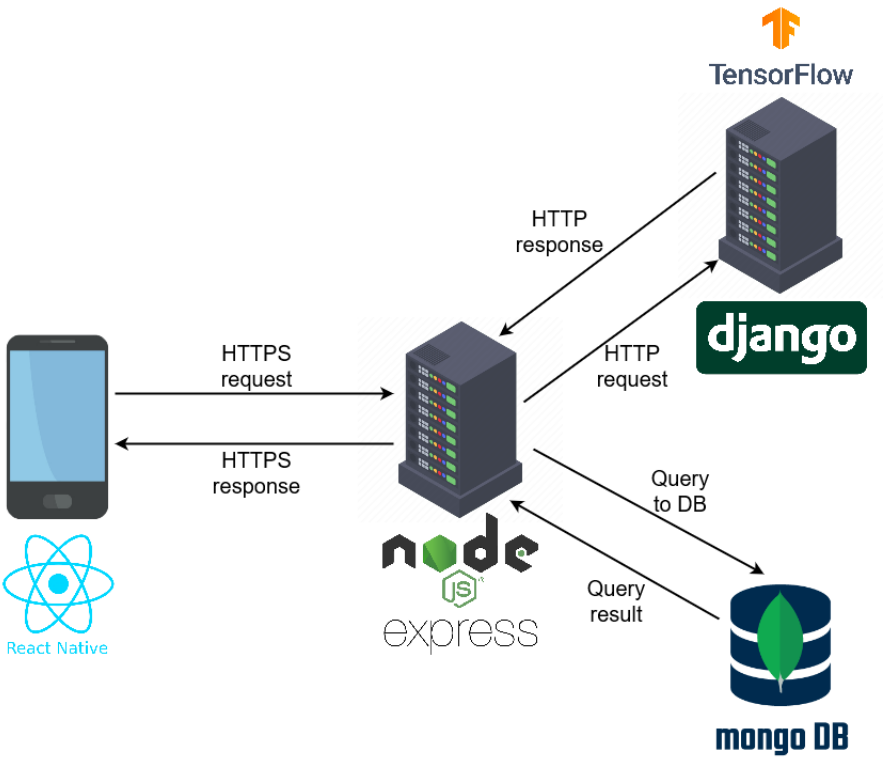


Figure 4.1: Blocks diagram of technology stack

4.1 Backend implementation

In Section 4.1, it will be explained how the server side was implemented. The server side includes the NodeJS, ExpressJS and MongoDB. The Django application corresponds to the recommendations.

4.1.1 Database setup

Initially, before starting to develop the code, the database must be configured and running. In the case of the project, MongoDB is used. This database offers two different methods to have it running. Those methods are:

- Download community server.
- Using MongoDB Atlas, this solution offers serverless instances and multi-region and multi-cloud support.

The method chosen was MongoDB Atlas [16], because it gives a more realistic approach, instead of all the modules being local giving a false perception of performance. Another advantage is the security; being in the cloud inherently provides a higher level of security compared to end-users implementing their own security measures on hardware and software. The fact that the database is already in the cloud is an advantage if the application is deployed there.

To use MongoDB Atlas, an account must be created on their website and then create a cluster. The Atlas provides free usage of the database, which is limited in terms of CPU and storage. The storage limit is 500Mb, which will be an issue for the recommendation system. The issues arise when the algorithm obtains the recommendations, most of the recommended products are not in the database. The number of products that exist is 21000, but with the storage limitation, it was only possible to store 300. To allocate all the necessary storage, it had to be paid $330 per month. Consequently, the recommendation system may suggest products that do not exist in the database. To connect the server to the database, it must be created an user account of the database access, which is different from the account on the website. After that, MongoDB generates a string which is used to access the database.

4.1.2 Project setup

Initially, to build the application, it is necessary to have NodeJS installed on the machine. Now it is possible to start the project, for that it was used the command `npm init` which creates a `package.json`. That file will track all the dependencies installed. Every time the code is pushed to the repository, for example, on GitHub, to install all the dependencies, it is only necessary to run the command `npm install`.

4.1.3 Server implementation

All the implementation of the backend is done using NodeJS with ExpressJS [17]. As mentioned before, ExpressJS is very useful when building web applications because it simplifies the handling of HTTP requests and routing. The implementation follows the Service Pattern, where the service is decomposed into multiple layers. Those layers are:

- **Controllers**: The controller is responsible for handling incoming requests from the user interface or external systems. It processes the input and calls the service layer to perform the business logic and provide the appropriate responses.
- **Services**: The service is responsible for the business logic, and acts as an intermediary between the controller and the access layer (repositories). Services may interact with multiple repositories and orchestrate the overall flow of data and logic.
- **Repositories**: The repository pattern is used to abstract the data access logic from the rest of the application. Repositories are responsible for retrieving and storing data from/to a database.
- **DTOs (`Data Transfer Objects`)**: DTOs are objects that carry data between processes or layers of an application. They are used to encapsulate data in a format suitable for the communication between parts of the application. DTOs will be used to transfer data between the frontend and backend.
- **Models**: Models are used to perform business operations and interact with data access layer. This means that they correspond to a database,

and each model corresponds to a table. Each instance of the model represents a record in a table in the database.

4.1.4 Controllers implementation

The implementation in the project was based on software architecture. The functionalities implemented followed the diagrams, making it easier to develop the code. In the implementation of the `Controllers`, an instance of the ExpressJS was created. Additionally, it is necessary to have a `bodyParser` in ExpressJS, otherwise it cannot read the messages received. The `bodyParser` can only receive text messages, but sometimes the requests also include images. To receive those images, it is necessary to use the `multer` library. Finally, to have the server running and waiting for requests it is needed to call the method `listen` on a certain port.

Although the server is running, it does not necessarily mean that it is going to do something after receiving a request. Therefore, the last step is to create the endpoints to receive the messages and then return the responses. To create an endpoint with ExpressJS it is only needed to call the following command: `app.post('/endPoint',(req, res)`. In the previous code, the values are:

- The `app` is the instance of ExpressJS.
- The `post()` is the type of HTTP request received, in this case is a POST.
- The `'/endPoint'` represents the endpoint used to make a certain request.
- The `req` and `res` represent the request and response respectively. The request is used to receive the information, and the response is used to send the response to who made the request.
- Optional: In the endpoint that receives images, it is needed to call `multer`, specifying the name of the field on request with the files. Something like: `upload.array('images')`, where `upload` is the instance of

`multer`, `array` specifies that it is receiving more than one, and `images` is name of the field.

Another implementation consideration is the fact that NodeJS when accessing information that might take a while, it uses a `Promise`. A `Promise` is an asynchronous function that is launched, and the code keeps running. However, in this project, most of the time it is required to receive that information first to perform the following operation. For example, asynchronous operations can be accessing the database. To make the code execute synchronously, it is necessary to use the annotation '`async`' on the name of the method, and then place '`await`' before the `Promise`. This way, the code will execute synchronously. Despite being called'`async`', the objective of the annotation is indeed to enable synchronous code execution.

Finally, it is important to address error handling. The error handling is used to prevent the application to crash if an error occurs. When an error occurs, it sends a message with status code 500, indicating an internal server error. This was done using `express-async-errors`.

4.1.5 Services implementation

In the services is where the business logic of the application is made. The services create the repositories to complete the CRUD (`Create, Read, Update, Delete`) operations. With the repositories, it is possible to complete the functionalities needed in the application. The services are also responsible for receiving a message and returning a response to the controller.

The services are where the software architecture is more useful. The implementation of the services is converting the sequence and class diagrams into the code in the specific language. This makes the code much easier to implement in terms of functionalities and reduces the code entropy.

In the architecture of software, the implementation of full independent services has always been followed. This is achieved through the implementation of microservices, which involves breaking the software into small, independent, and modular services that operate as autonomous entities. Each

microservice is responsible for a specific business area and communicate with others through well defined APIs. This approach gives several advantages, because enhances the scalability by allowing individual services to be deployed and scaled independently, optimizing resources. To achieve this, it is also necessary to create one controller for each service and each service has its own repositories. One service can not access repositories from another service. The only part that is common to all services is the database, which is shared among all services.

To make the communication between services, which sometimes is required, it was used the library `axios`. This library is employed to make HTTP requests from the web and NodeJS environments. With this library it is possible to obtain information from other services.

4.1.6 Models implementation

The models are records in the database used to store and retrieve information from the database. In MongoDB since is a NoSQL database, it is divided in three types.

- **Database**: MongoDB organizes data into databases just like relational databases. Each database can have multiple collections.
- **Collection**: A collection contains documents. The collection is analogous to tables in relational databases, but with the flexibility of not having a fixed schema.
- **Document**: Each document is a JSON-like object that contains data. Documents are records within a collection.

Although it is possible to have a collection inside a document, it was used only a two level organization. This results in just having a collection and documents inside it. The total number of collections is equal to the total number of models, which corresponds to the database entities.

To implement the models, it was used a '`Schema`' [18]. This '`Schema`' is really useful, because it defines a structure that the document must follow.

This '`Schema`' can define which fields the document will have, whether they are required or not, the type of the field, and it can validate information about the field. This '`Schema`' prevents the documents from having inconsistent data, which is common in NoSQL databases.

4.1.7 Repositories implementation

The repositories are where the connection to the MongoDB Atlas is created. The implementation of repositories is done using the '`Schema`' of the model. The implementation of the models has two representations, the first one is just a JavaScript object and the other is the '`Schema`'. The repositories are the only components that use the '`Schema`' objects. They receive the JavaScript object from the services and converts it to the `Schema` object. With the instance of the '`Schema`' it is given methods to store, update, read and delete the object from the database. When querying the collection, it is possible to apply filters just like a SQL query. With that query, it is possible to retrieve only the essential information from the database.

4.1.8 DTOs implementation

The implementation of the DTOs is simple. Essentially, for each DTO, a class is created with the values passed into the constructor of the class.

4.2 Security in the server

The implementation of the security measures in the server is of paramount importance, because it serves as the foundation against unauthorized access, data breaches and malicious attacks. The security is essential for maintaining the trust of users and customers.

The implementation of the server follows five different security measures:

- **Input validation**: validates the requests received on the endpoints. This prevents injection attacks, such as SQL injection or XSS(Cross-Site Scripting). This also helps ensuring data integrity and can increase performance since it does not process unnecessary information.

- **Logging**: implements logging. This can help detect attacks and prevent them from happening again.
- **Authentication**: implements authentication to access certain information on the server that no other user can access.
- **HTTPS**: usage of HTTPS to transfer data. This protocol safeguards sensitive information such as login credentials, payment details and personal data.
- **Encrypt password**: in the event of a breach, if the passwords are exposed, the attackers would face significant challenges in decrypting the password. Another advantage is the fact that developers should not know the password of the users.

Although all the security measures were implemented in the code, they were not applied to all services. This is because the entire implementation takes a significant amount of time to implement and test. The only service that has all those functionalities is the service of profiles.

4.2.1 Implementation of input validation

The implementation of validation on the requests received on the endpoints is made using the '`Joi`' library. This library allows filtering the type of request, such as string, number, or others, and can also check if the field is required or not. It also validates if it is an email, minimum and maximum of a number, and other functionalities. When the validation fails, the server sends a response with the code 400, which means a bad request.

To run the validation, it must be explicitly added as an argument of the endpoint of ExpressJS like this: `app.post('/endPoint', Validator.validateInfo, (req, res) =>` . In the previous line, the `Validator.validateInfo` handles the validation of the request. All the validations were done inside the package `validator`.

4.2.2 Implementation of Logging

The implementation of logging is done using the library `winston` [19]. This library, just like other logging libraries, provides a format for messages and

the level of logging. The level chosen is 'info' and the logging goes to both the console and a logging file. The logging was implemented in all the services. One important detail to consider is that it is never a recommended practice to log passwords, so that information was never recorded in the logs.

4.2.3 Implementation of Authentication

The implementation of the authentication is done using the `jwt` library. The `jwt` stands for JSON Web Token. This library gives methods to generate a token based on certain information, can verify if a token is valid or not, and checks if it has expired. To generate a token, three things are required: the value we want to store, a secret key only the server knows to generate the token, and the duration validity of the token.

The creation of the token occurs when users complete the login. When the credentials are correct, they receive two tokens. The first token is used to access the endpoints that require authentication, and the other is to refresh the token when it expires. To check if the token received is valid or not, it was created a method for that purpose. The endpoints that require authentication call that method as an argument, just like the input validation. To refresh the token, it was created a dedicated endpoint. When someone sends a request with an invalid token, the response will have the code 401 Unauthorized. In this case, the user must send a request to the endpoint to refresh the token and then make the request again with the new token. This approach reduces the risk of someone stealing the token to access some features. The authentication was only implemented in the service of profiles.

4.2.4 Implementation of HTTPS

The implementation of HTTPS is quite simple in terms of code. The difficulty lies in the generation of the public and private keys. To generate the keys there is two ways:

- Generate a self-signed certificate. This approach is valid in the development phase, but it is not recommended for production. It is not recommended because when a web encounters an unknown certificate, it displays a red warning indicating that it does not recognize the owner

of the certificate. This warning occurs because it could be an attacker providing a forged certificate and being able to read the messages sent.

- Obtain a certificate from a website that generates those keys. This approach is used in production phase. However, to obtain these certificates, most of the time, an annual fee must be paid.

In the end, only the self-signed certificate was used to avoid paying. To generate the certificates, it is necessary to install the OpenSSL [20]. After that, it is necessary to add it to the computer's path to execute command lines with it. Finally, it is necessary to run three command lines: (1) generate a `.key` file, (2) generate a `.csr` file, and (3) generate a `.pem`. The private key is the file with extension `.key` and the public key is the file with extension `.pem`. The code to use HTTPS in NodeJS is called `https.createServer(file.key, file.pem, app).listen(port)`.

4.2.5 Encrypt password

The encryption of the passwords was done using '`bcrypt`'. This library generates a hash of the password based on salt rounds. That information is stored in the database. To verify if the password is correct, '`bcrypt`' was also used, calling the method `compareSync` to check if the password is correct or not.

4.3 Recommendation System

The objective of the recommendation system is to develop a system that can show similar products based on images. The steps followed by the recommendation system are:

- **Classification**: this step is used to restrict the space search. When an image is classified into a certain class, it will be compared only with the images belonging to that class.
- **Compute distances**: in this step, the distances between the submitted image and all the images of the previously obtained class are calculated.

- **Sort values**: finally, the values are sorted to obtain only the best ones. With these values, the corresponding products are obtained and returned.

The entire recommendation system was developed in Python. The reason to choose Python is because it has a rich ecosystem of libraries for machine learning, such as Tensorflow and PyTorch. To create the recommendation system, first, it is necessary to define what the classes will be. These classes will represent a group of products that are different from the others in some way, and could be relevant to the users. The advantage of using classes compared to not using them is a gain in terms of time required to calculate the similar images. This way, it is only calculated a small part of all the images stored, resulting in a much smaller processing time. After some research in multiple applications, the total number of classes is 25, these classes are in the Table 4.1, sorted by column.

Agriculture	Books	Child	DVDs	Motorbikes
Antiques	Cameras	Clothing	Electronics	Real estate
Appliances	Cars	Computers	Furniture	Sports
Audios	Cellphone accessories	Construction	Games	Tablets
Bikes	Cellphones	Decoration	Garden	TVs

Table 4.1: Classes of the images

Although there are a lot of classes, they might not cover all the possible products. Thus, the number of classes might increase in the future depending on the needs of the users.

4.3.1 Dataset creation

The recommendation system is a supervised model, and all the supervised models must have the data labeled to guide the model during training. This way the model can learn patterns and relationships allowing it to make accurate predictions when presented new unseen data. Without labeled data the model lacks the necessary guidance to learn effectively.

In this matter, to implement the recommendation system, it is necessary to have a labeled dataset to train and test the models created. This was a big

challenge because even though `Keras` has some datasets, they did not even cover half of the total classes. Even on the internet there are some datasets for some classes, but most of them have poor quality and do not include some of the required classes.

The solution found was to retrieve images from websites that have images related to the classes needed. Those images must not have white background but should be in a real environment with some noise, poor illumination, and some rotation. This is because when a user sends an image, most of the time, the user takes a picture in a real environment. It is assumed that is really rare for a user to have an image in the gallery with perfect quality and white background.

To retrieve images from websites, there are two possible solutions. The first one is obtaining the images manually by going to the pages and store the pages, then retrieve only the images. This approach might take some weeks or even months to obtain a reasonable dataset in each class. The second solution is to extract images programmatically. To do this it was used Python with '`Selenium`'. '**Selenium**' is a library that provides a way to interact with web browsers programmatically.

The way '`Selenium`' works is, firstly, a driver object is created to access a specific browser. In this case, it was used the Firefox. Afterwards, it is necessary to specify the URL on the driver to redirect the browser opened by '`Selenium`' to the desired page. '`Selenium`' provides a set of methods to interact with the page programmatically, like clicking on elements on the page and access information such as text, URLs, and others. Although it can click on elements, it comes with a problem, which is the time the page needs to load the content. '`Selenium`' provides a method to wait a predetermined amount of time, if it did not load until that time it throws an error. The timing was the most difficult part to implement because sometimes the page loads really fast, and the program had to wait that amount of time; other times, it loads really slowly and throws the error.

To minimize the number of errors, the process was divided into two

phases. In the first one, the program stores all the links to the pages with useful images on a text file. Secondly, it opens each page with the driver using the URL from the text file. Then, it is necessary to store the images locally. To store the images, the Python `requests` library is used. Each page contains a link to the images in the cloud. To store the images, the URL is passed to the `requests`, and then the content is stored locally. With this approach, it was possible to create the dataset.

After retrieving all the images, there were still two problems to solve. The first one is the fact that in some pages, there are repeated images. In the dataset, these repeated images could represent as much as twenty percent of the total number of images. The second issue was the existence of random images that did not make any sense to have in certain classes.

To solve the first problem, an algorithm was created using `OpenCV`. This algorithm calculates the difference in pixels from one image to all the images in the same class. If the sum of the differences is equal to 0, it means that the images are identical. This process takes a really long time because each package of images has a lot of images and it must calculate the difference between all the pixels. The best solution found to speed up the process was to create `Threads` in Python. Each `Thread` was created for the function that calculates the pixels difference between images from each package.

The second problem had no choice but to manually check each image and delete the images that are not relevant for a certain class. Initially, a dataset with 81000 examples was created. Some models were trained with the previous dataset. After some time, a bigger dataset with 176663 examples was created and the final models were created using that dataset.

4.3.2 Images classification

The recommendation system was implemented using `TensorFlow` and `Keras`. The process of training neural networks, in most cases, takes a long time to run. By default, `TensorFlow` uses the CPU as the running hardware. However, the CPU takes about 10 times longer than GPU, so `TensorFlow` was configured to use the GPU.

To calculate similar images based on an image, firstly, it is necessary to calculate which class the image belongs to. Based on the prediction of the convolutional neural network, it is calculated the distances between the feature maps of the received image and the feature maps of all the images in a certain class from the database.

To train a neural network it is necessary to load the images from the dataset, this dataset is divided into train and test. The training dataset contains 123,376 images and the test contains 53,287 images. Those numbers are too big to fit entirely in the RAM. To address this, `TensorFlow` already provides a way to divide the dataset into batches. A batch is a subset of the dataset. To do this, an `ImageDataGenerator` is necessary, and this class also provides real-time data augmentation and normalization. To load the batches of images, the method '`flow_from_directory`' is called twice, one time for train and other for test. One important aspect is that pre-trained models will be tested, and each model uses a different pre-processing on the images. Therefore, it is necessary to apply that pre-processing to the `ImageDataGenerator`.

In the `ImageDataGenerator` the parameters used was:

- Image size used: (299, 299)
- In the train dataset, it was shuffled the data to improve the model training and generalization. This shuffle is not applied to the test dataset.

It is necessary to create a convolutional neural network (CNN). This type of neural network is widely used for image classification due to the fact that it is designed to recognize patterns in a grid-like structure. `Keras` already provides a list of pre-trained models with `ImageNet` weights. `ImageNet` is a large dataset containing millions of labeled images across thousands of classes, which is commonly used for training computer vision algorithms. To create the model needed, a machine learning technique called transfer learning is used, where a model learns from one task and applies it to different but similar problem. This can achieve faster training times, is computationally

less expensive and can often achieve better performance results. Another process that will take place is fine-tuning, it refers to the process of taking a pre-trained model and further train it on a new dataset or task. This process allows the model to adapt its learned features to better suit the specifics of the new data, leading to improved performance.

The pre-trained models used from `Keras` were `MobileNet` and `Xception` [21]. The pre-trained neural network that produced the best result was `Xception`. To achieve this outcome, several tests were conducted in order to reach this conclusion. The tests were:

- Creation of a simple neural network. This neural network will serve as a reference for its classification value.
- Create a pre-defined neural network from scratch. In this case, the `MobileNet` neural network was used because the other networks did not work for training from scratch.
- Create a pre-defined `MobileNet` from scratch but with additional layers. This produced worse results than just using the network without more layers.
- Create a model (neural network) with `Xception` using the weights of `ImageNet`. The training of all models before this, and including this one, was done using a dataset with 81,000 examples. All the following models used 176,663 examples.
- Creating the same model with `Xception` and `ImageNet`, but now adding more layers.
- Creating the same model, where the layers are trained by unfreezing those layers, in this case, the layers 5 and 6. This test produced the best results.
- Creating the same model but unfreezing all the layers from layer 5 until the end.
- Create a model that unfreezes only the last layer, which is the layer 14. This model did not produce the best results but varied by only 1%.

However, the idea of unfreezing the last layer is better than unfreezing layers in the middle of the model. Because unfreezing layers in the middle might corrupt the learning of the network, which will produce bad results for recommendations.

- The last test, a model was created where it was unfrozen the last layer, which is layer 14, and apply data augmentation.

Neural network	**Precision**
Simple neural network	14.36 %
`MobileNet` from scratch	44.39 %
`MobileNet` from scratch with more layers	39.71 %
`Xception` pre-trained	60.79 %
`Xception` pre-trained with more layers	61.9 %
`Xception` unfreezing layers 5 and 6	64.08 %
`Xception` unfreezing layers 5 until the end	58.53 %
`Xception` unfreezing last layer (14)	63.06 %
`Xception` unfreezing last layer (14) with data augmentation	58.89 %

Table 4.2: Precision of each neural network tested

4.3.3 Compute image similarities

After the convolutional neural network is trained, it is possible to calculate distances between feature maps. With those distances, it is possible to decide which images are more similar than the others [22].

The model used to obtain the feature maps for calculating similar images is the CNN previously trained for the classification task. However, this model must suffer some changes in order to produce better results. This can be achieved through the `Model` from `TensorFlow`. To use the `Model`, the input layer must be specified, which is equivalent to the classification CNN. The output can vary, multiple solutions were tested, including the use of the layers 10, 11, 12, 13 and 14 from `Xception`. However, the best results were obtained using the entire CNN without the top dense layer, which is used for classification. The layer obtained at that position corresponds to the `GlobalAveragePooling`. Then, the feature maps obtained from the

`GlobalAveragePooling` are used to calculate distances between feature maps of different images, resulting in a value which represents the similarity. Other solutions for the output layer will be discussed in the Chapter 5.

Initially, three different images are classified loaded in different ways. One of them uses the `ImageDataGenerator` from `TensorFlow`, and the other uses `Image` from `PIL` library. This test aims to verify whether the classification of the images produces the same result or not. The reason to conduct this test is that, when the API receives a request, it will not load the image using `ImageDataGenerator`, instead it will utilize the `PIL` library.

To calculate distances initially, a simple example was created to simulate the user sending three images, with a dataset containing sixteen images. With this in mind, the distances were calculated for each image in relation to all the images in the dataset. The distance metrics used to calculate the distances are the Euclidean distance and the cosine similarity. The result of the test is in Figure 4.2.

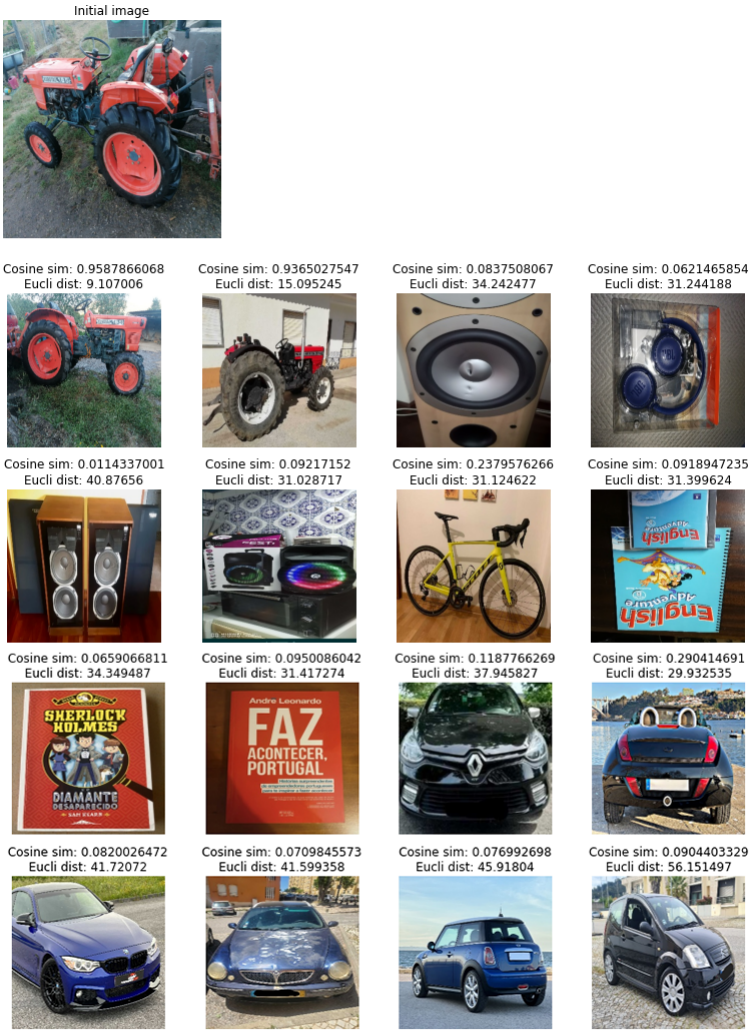


Figure 4.2: Result of the calculus of the distances

In Figure 4.2, the results of the distances between an image sent by the user and the resulting distances obtained from the dataset are shown. The objective of this test is to check if the algorithm would select the first image of the dataset as the most similar and the second one as the second most similar. The results were the expected ones, but this is a very specific example. More detailed tests will be conducted in the Chapter 5.

To apply this algorithm to a real scenario, it is necessary, first, to calculate the feature maps of all images in the dataset. This approach helps

prevent the program from recalculating all the feature maps each time a user sends an image, which significantly improves the performance. The algorithm used to calculate the distances is exactly the same as the one applied in the Figure 4.2. The only missing part is the need to sort the values based on the metrics and to only show a certain number of images. The number selected was 10 images. The sequence that the program follows is in Figure 4.3.

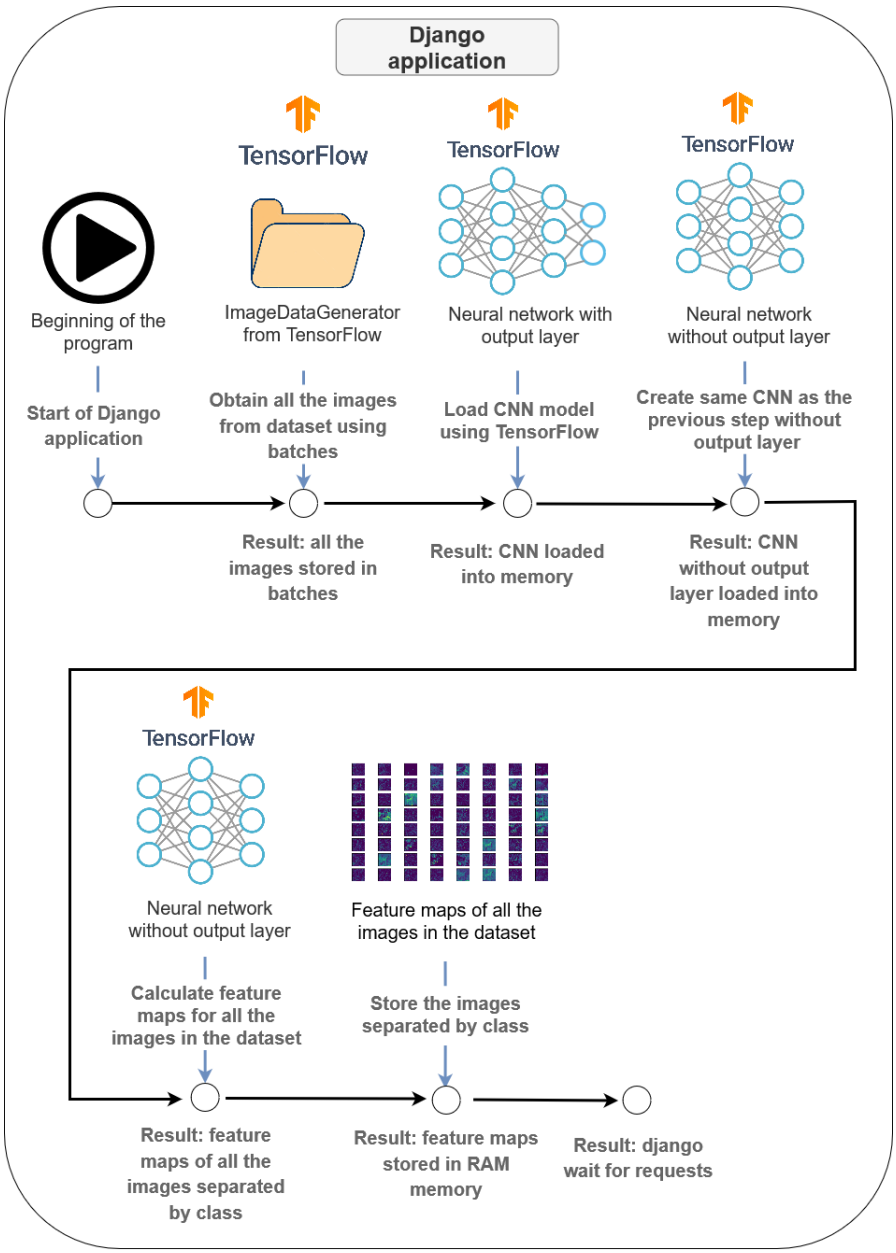


Figure 4.3: Diagram of the process to calculate the feature maps for the dataset

One thing to take into account is the time the program takes to calculate the distances between one image and the images in the dataset. The fact that the user has to wait for the program to calculate the distances is a problem. If the process takes too much time, the user might lose interest and may not

use the functionality again. The issue will be discussed in the Chapter 5.

4.3.4 Implementation of the algorithm in Django

After the implementation of all the steps in the `Jupyter Notebook`, it is necessary to have a way for the server implemented to use the neural network. The solution found was by using the framework `Django`. `Django` is used for building web applications using Python. The implementation in `Django` was simple since it only needs two endpoints.

The implementation was made by modifying only three files. The first one is `urls.py`, where the endpoints are defined. Two endpoints were created: one to obtain the recommendations, and another was to calculate similar images based on the images sent by the user. The other two files will be explained after in the document.

Although `Django` has two endpoints, both of them use the same algorithm, which is to calculate similar images based on received images. However, as mentioned before, it has a problem related to the database storage. Due to the fact that it is using a free database plan, the storage is limited to 500MB, while the dataset has 35.5GB. This value is much higher than the maximum capacity of the database. The alternative found was to use the local computer as a database, storing information related to the product, such as the title, price, seller and others, in a text file.

The implementation of the body of the endpoints was made in two files, the file `views.py` and `utils.py`. The main body of the endpoint is in the file `views.py`. These functions basically receive the images from the requests and convert them to a format that the neural network can work with. After the conversion, the body of the endpoint calls the implementation from the `utils.py` to obtain the similar images with the corresponding information of the product. The implementation of the functions in the `utils.py` comes from the implementation made in `Jupyter Notebook`. First, it loads the dataset images in batches and then calculates the feature maps. Now, every time a user sends images, it classifies the images and based on the class of every image, it calculates the distances between that image and all the images

of that class. After calculating everything, it sorts the values, obtains the images and retrieves the information about the product. Then it returns the information to the sender. The whole process is shown the Figure 4.4.

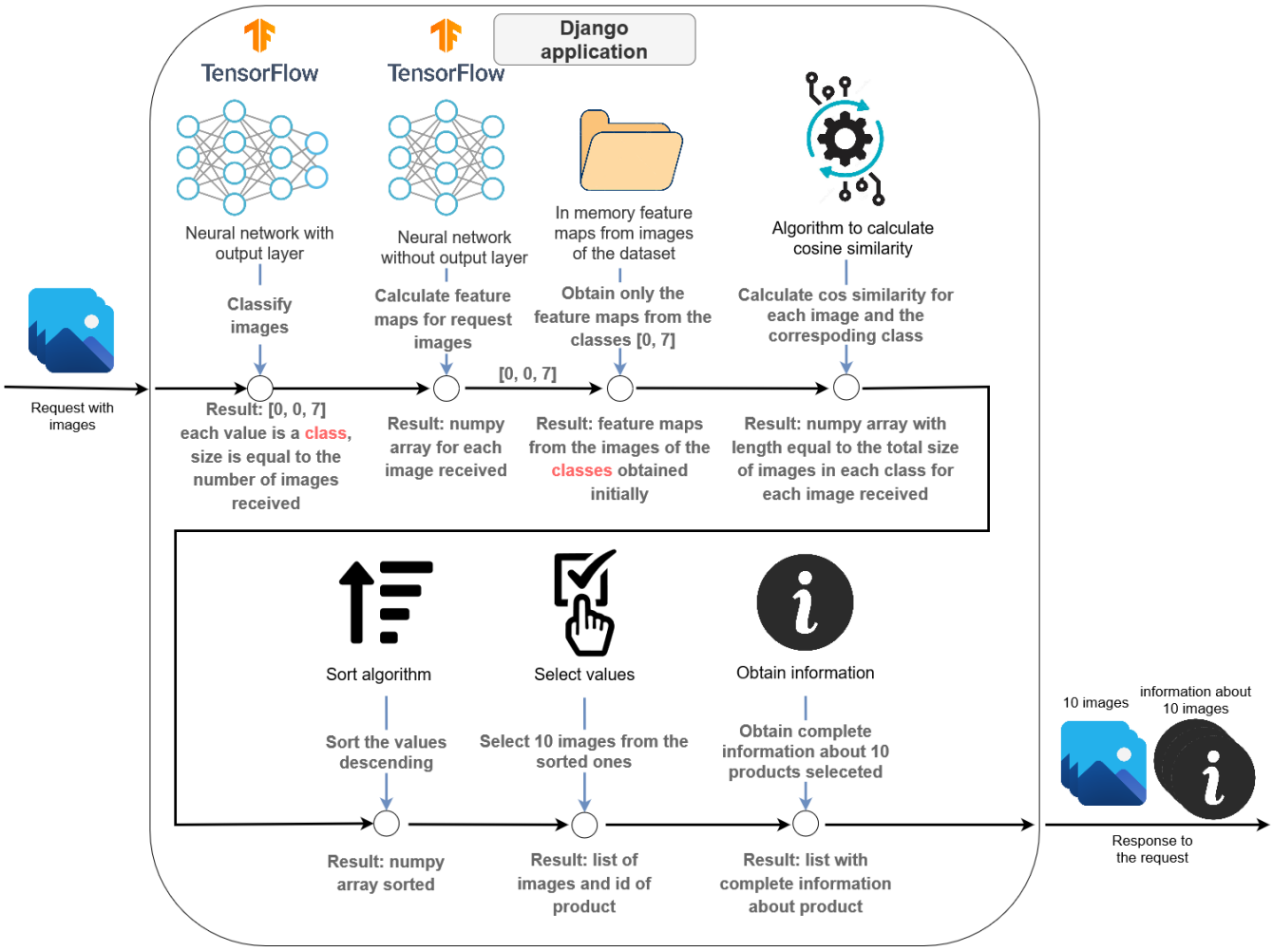


Figure 4.4: Diagram of the process of receiving requests in Django

4.4 Frontend Implementation

The application part shown to the user is called frontend, which is implemented in `React Native`. `React Native` is a open-source framework that enables the creation of cross-platform mobile applications with a single codebase, both for iOS and Android. Developers implement the desired `UI` (User

Interface) state, and `React Native` handles the underlying native UI components accordingly. A key feature is the ability to achieve near-native performance.

In `React Native`, there is a starting point defined in the file `package.json`. In the case of the project, it is called `App.js`. This file is responsible for loading the fonts used in the application, otherwise it results in an error because the fonts are used before being loaded.

The development was made using `Expo` [23]. This framework is designed to abstract some complexities of native mobile development. It allows the developer to focus more on building features instead of doing platform-specific configurations.

4.4.1 Implementation of routing

After defining the starting point, a router is needed. A router is a component that facilitates the navigation and organization of screens. It manages the screens while maintaining a consistent user interface. The library used is `React Navigation`, in the project is used a `Stack Navigator` and `Tab Navigator` to define the navigation. With routing is possible to enhance the user experience using smooth transitions with a simple declarative syntax.

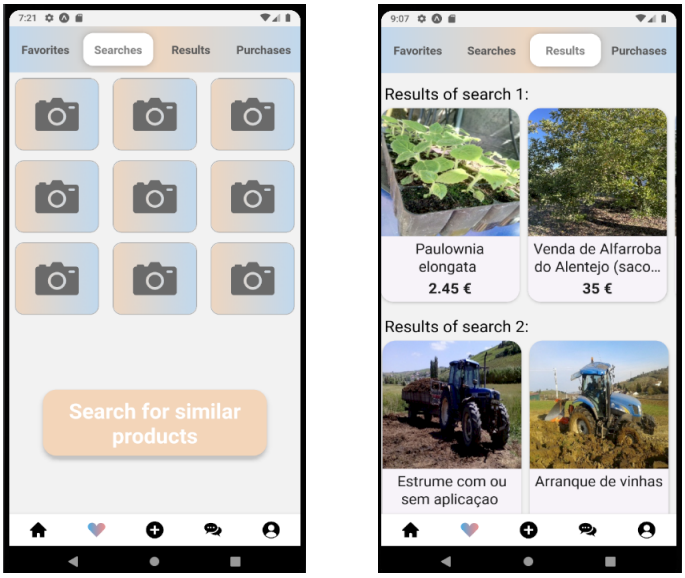
The `Stack Navigator` uses a hierarchical stack structure. It maintains a stack of screens where each screen is pushed onto the stack and going back removes the top screen.

The `Tab Navigator` creates a tab navigation structure. It uses a tab bar for users to switch between different screens or sections in the app. In the project, both a top bar and a bottom bar were used.

4.4.2 Implementation of the pages

After implementing the router, the application pages can be implemented. The majority of the time was spent creating the pages. To build the pages

it is used `JSX`, standing for JavaScript XML. `JSX` allows the embedding of JavaScript expressions within curly braces {}. This enables dynamic content and logic within the markup. This method allows the creation of UI code with JavaScript code. Example of two pages are in the Figure 4.5. The resulting implementation of the pages is in Appendix 6.



(a) Page search for similar products

(b) Page result of search for similar products

Figure 4.5: Examples of pages related to similar images

To build the page, it is necessary to define a function in JavaScript, and the body of the function is what is rendered. The basic idea is to use multiple simple elements to build more complex elements. The basic wrapper element is the `View` tag, which functions like the `div` in `HTML`. To write text, the tag `Text` is used; images use the tag `Image`; input text uses the `TextInput`; scroll elements use `ScrollView` or `FlatList`, and finally, the `TouchableOpacity` to create buttons.

With the elements created, it is possible to style them using inline styles or using `StyleSheet.create`. The styles resemble regular `CSS` but with `camelCase`, for example `backgroundColor` instead of `background-color`.

With `flexbox`, it is easier to create a responsive design across different screens. For instance, applying the `flex` property to a child element to have a certain dimension compared to the father element. The horizontal responsiveness of the application was achieved by using a percentage of width from the screen. The vertical responsiveness was solved using a scroll view on all pages.

Inside the function to render the page, it is also possible to define logic. This might include fetching data, condition rendering and event handling. The logical part of the application is typically encapsulated using JavaScript functions.

An important feature in `React Native` is the usage of states, which refers to the data that can change over time within a component. A state represents variables that can be modified during the component's lifecycle, either by interactions or as data updates. This is achieved by the `useState` hook, which is used to declare and update state variables. When the variable is updated, the page also changes, otherwise, the page never changes.

4.4.3 Specific page elements

In some pages, there was the need to use functionalities such as a form, a map and a mechanism to store sensitive information. The form functionalities were implemented using the `Formik` library with `yup` library (Schema builder). `Formik` creates a form with very useful tools. The most useful are the validations, showing validation errors and reset the form after submitting. If these functionalities were implemented manually it would require a large amount of code and would be hard to track all the possible exceptions. `Yup` is used to create a `schema` to validate the form. This library is highly flexible and allows fields in a request to be required or not, and supports various data types such as numbers, strings, emails and others.

To use the map, it is necessary to install the libraries `expo-location` and `react-native-maps` [24]. To display the map on a page the tag `MapView` is used, and a default location must be defined. This tag has a click listener

that retrieves the coordinates on the map. The screen of the map is shown in Figure 4.6.

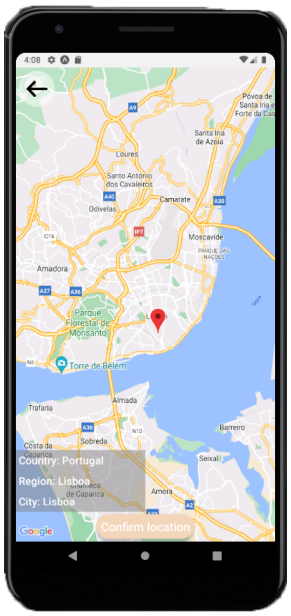


Figure 4.6: Map screen implemented

Finally, there must be a secure place in the memory to store sensitive information. This information can be access tokens, passwords or other information. In the case of this application, it was used to store the user's email, the access token, and refresh token. This can be achieved using the `expo-secure-store` library. By using an object provided by the library, it is possible to store and retrieve values every time they are needed using a key. An example of storing is `SecureStore.setItemAsync('email', emailVariable);`.

4.4.4 Components in pages

A component in `React Native` is a self-contained and reusable unit of user interface that encapsulates a specific piece of functionality. Components can be simple elements like buttons or more complex structures. Components allow to create, reuse and manage UI elements efficiently. They have their own logic, styling and state. What makes components useful are their reusability, once defined they can be reused throughout the application. This promotes

a modular and maintainable code structure. An example of a component is the red square on the Figure 4.7.

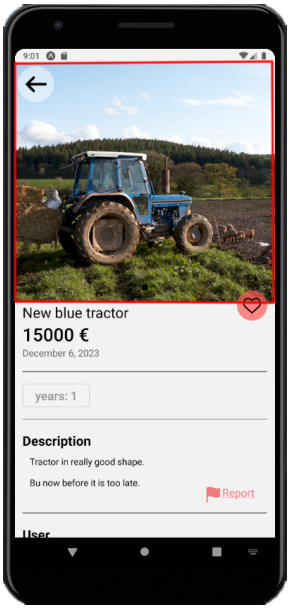


Figure 4.7: Component in a page

4.4.5 Controllers DTOs and Enums

After implementing the pages, the last needed part is how the application makes requests to the server. The functions that make requests are inside a controller. These controllers are totally different from the ones in the backend. The name controller is only used due to a convention, but in reality it is the `ModelView` defined in the architecture. These controllers use the library `axios` to make the requests to the endpoints.

To make the requests to the endpoints, it is necessary to have DTOs that are exactly the same as the ones in the server side. This is because both the backend and frontend are written in JavaScript. For the application to make requests as a user navigates to a certain page requiring information from the database, it uses a hook called `useEffect`. This function can run multiple times, but for this particular case, only runs once.

Finally, in the architecture, some variables were defined as enumerators.

All of these enumerators were implemented as numbers. This approach enables the reduction of the message size being sent from the server to the application, because instead of sending a string it sends a number which results in a smaller message. Enumerators are used to determine which text corresponds to a certain numeric value. For example, the sold state of a product follows the Table 4.3.

Value	**Text**
0	Order Processing
1	Order Packaging
2	Shipping
3	Delivery

Table 4.3: Sold product pattern

Chapter 5

Validations and Tests

5.1 Backend tests

In the architecture, the test architecture was defined. The aim in this architecture was to test all backend services. All the tests implemented follow the test architecture. The idea behind the implementation of the tests was to be as close as possible to a real device, which makes connections to the server. With that in mind, the tests were divided into two separate files.

The first file is the class with the service name followed by `Test`. This class constrains all the methods of the service, with `Test` appended to the method name. An example of a method is `obtainProductDetailsTest()`, which is part of the class `ServiceProductsTest`. These methods basically receive information to send to the server as parameters, build the DTOs, and make the requests to the endpoints using `axios`.

The second file functions as the starting point to run the tests. These files contain the name `main`, followed by the name of the service to identify the test. In those files, the service that is supposed to be tested and the `Test` class are instantiated. In the main file, the methods of the `Test` class are called with the appropriate arguments, and then the `Test` class makes the requests. To validate if the program worked as expected, commentaries with the supposed results were written. In these tests, almost all the possible outcomes were tested such as duplicated messages, null messages, invalid arguments and valid ones.

Although these tests were very useful to check if the server worked as expected, unit tests with the appropriate libraries should also be implemented. Those tests were not implemented due to the amount of time it would take.

5.2 Recommendation System Evaluation

The recommendations were mostly about testing and checking what produces the best results. The initial tests were created to determine which neural network produced the best classification results. The first set of tests is described in the Section 4.3.2. In this section, it will be described in more detail the results of the best neural network, along with an explanation of the preference for a certain neural network over the others and the reasons for selecting specific arguments.

The convolutional neural network used to calculate the similarity between images was `Xception`. The main reasons for selecting `Xception` were because it has a higher score in the `Keras` applications of `ImageNet`. Secondly, it has fewer parameters (22.9M) compared to VGG (138.4M) or ResNet50 (25.6M). The 'M' represents millions of parameters. The last reason is that it uses the images with larger size (299, 299) compared to others that uses (224, 224). The fine-tuning applied from the tests is described in Section 4.3.2. Where it was unfrozen the last layer of `Xception`, data augmentation was not used, and `GlobalAveragePooling2D` was utilized to reduce the dimensions.

As mentioned before, the dataset used was divided into 25 classes, and the total number of images for training was: 123376. The number of test examples was: 53287. The resulting value of classification was: 19681 errors in 53287. The percentage of images classified correctly is 63.07%. The resulting confusion matrix is shown in Figure 5.1.

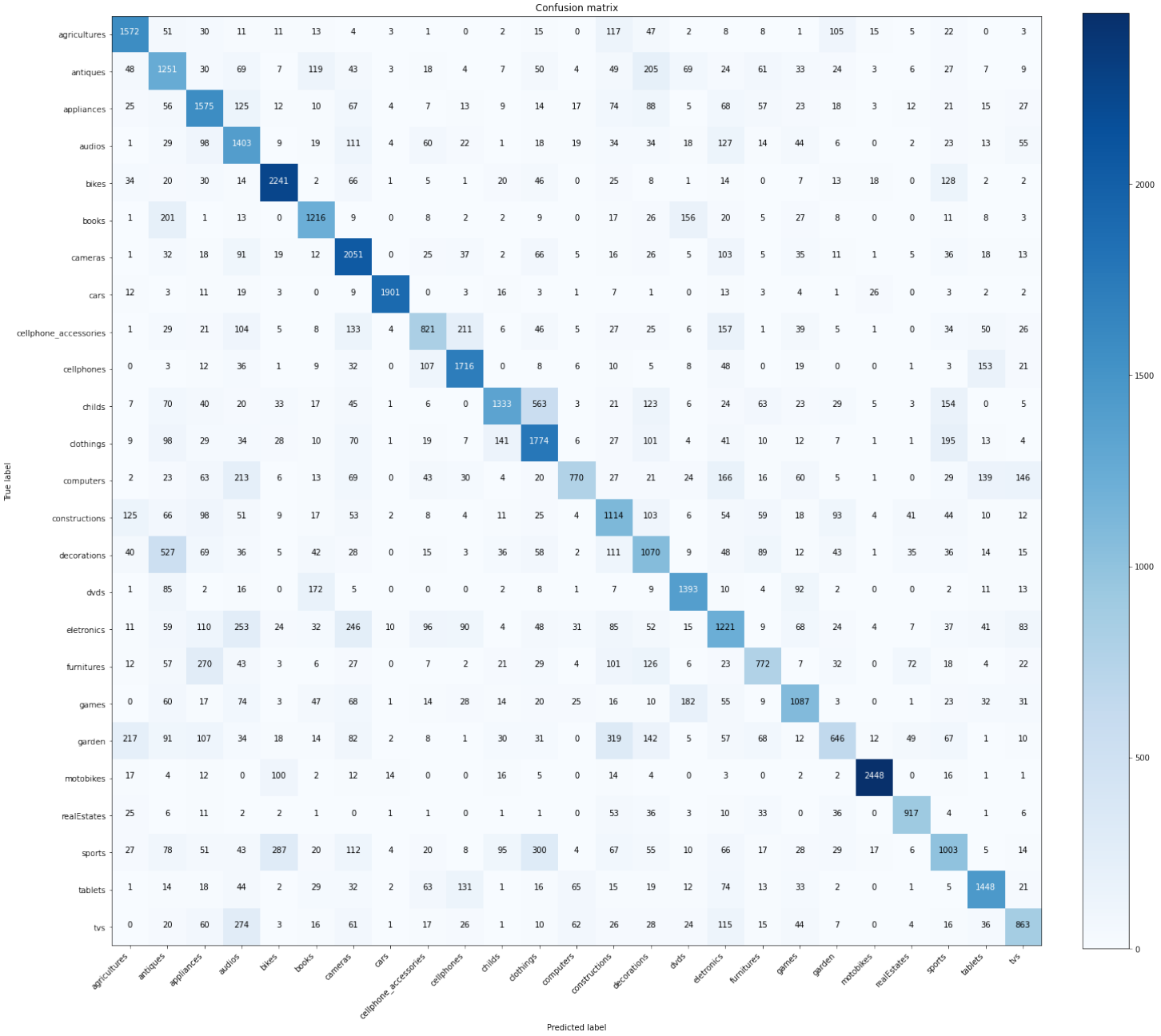


Figure 5.1: Confusion matrix of the classification

From the confusion matrix, it is possible to see that the convolutional neural network is classifying many images from the class `antiques` in the class `decoration`. In reality, this happens not because the network is not learning correctly, but because the images themselves are very similar to each

other. Another factor affecting the classification is that some classes contain certain objects that, even though they belong to that class, are more common in other class or classes. For example, there is cycling, and there is shirts for cycling, but there is a class for sports that contains many more examples of shirts than cycling, interfering in the classification process.

5.2.1 Similar images in different classes

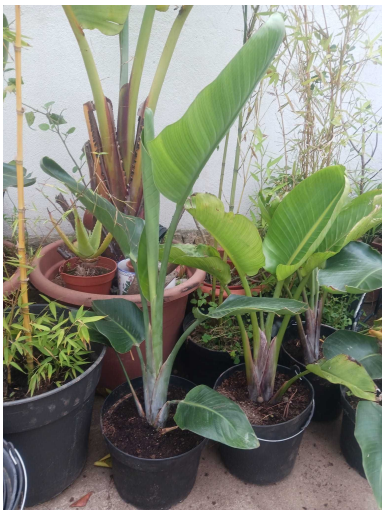
Similar images in different classes occur in multiple classes, but there are some classes where it occurs very often. The classes with more similar images between each other follow each row of Table 5.1.

Agriculture	Garden	
Antiques	Decorations	
Appliances	Furniture	
Computers	Electronics	TVs
Bikes	Sports	
Cellphone	Cellphone accessories	
Child	Clothing	

Table 5.1: Different classes with similar images

From the previous classes, two examples of images that are similar in different classes are in the Figures 5.2 and 5.3.

(a) Image from class Agriculture's



(b) Image from class Garden

Figure 5.2: Images from Agriculture and Garden

(a) Image from class Child

(b) Image from class Clothes

Figure 5.3: Images from Child and Clothes

5.2.2 Image similarity results

It is necessary to check the results of calculating similar images using the neural network for a real dataset. For these tests, the entire dataset, including both the training and test folders were used. In total, the dataset

contains 174457 examples. For this process, three different neural networks were used. The only difference between all of them is the layer right before the classification, which is the `Average Pooling`. None of the neural networks contain the dense layers of classification. The values are:

- Global Average Pooling: reduces the spatial dimensions, resulting in a single value per feature map. Converting the exit from (10, 10, 2024) which produces 204800 feature maps, into just 2048.
- Average Pooling (2, 2): computes the average value of each region in the input specified by a pool size and stride. The pool size and strides are both (2, 2), converting 204800 into 51200.
- Average Pooling (3, 3): converts the 204800 into 18432.

For the tests, two different metrics were used: the cosine similarity and Euclidean distance. Initially, the difference between cosine similarity and Euclidean distance for a single neural network will be compared. The folder of agriculture has 6828 examples, which is the one tested here. The remaining tests will be in Appendix 6. The results are shown in the Figures 5.4 and 5.5.

Figure 5.4: Result of cosine similarity with average pooling (2, 2)

Figure 5.5: Result of euclidean distance with average pooling (2, 2)

Based on the results obtained, it is possible to see that the cosine similarity obtains better results compared to the Euclidean distance. Even though it was only shown one neural network, all the others achieved better results with cosine similarity. The results for the other networks are presented in Figures 5.6 and 5.7.

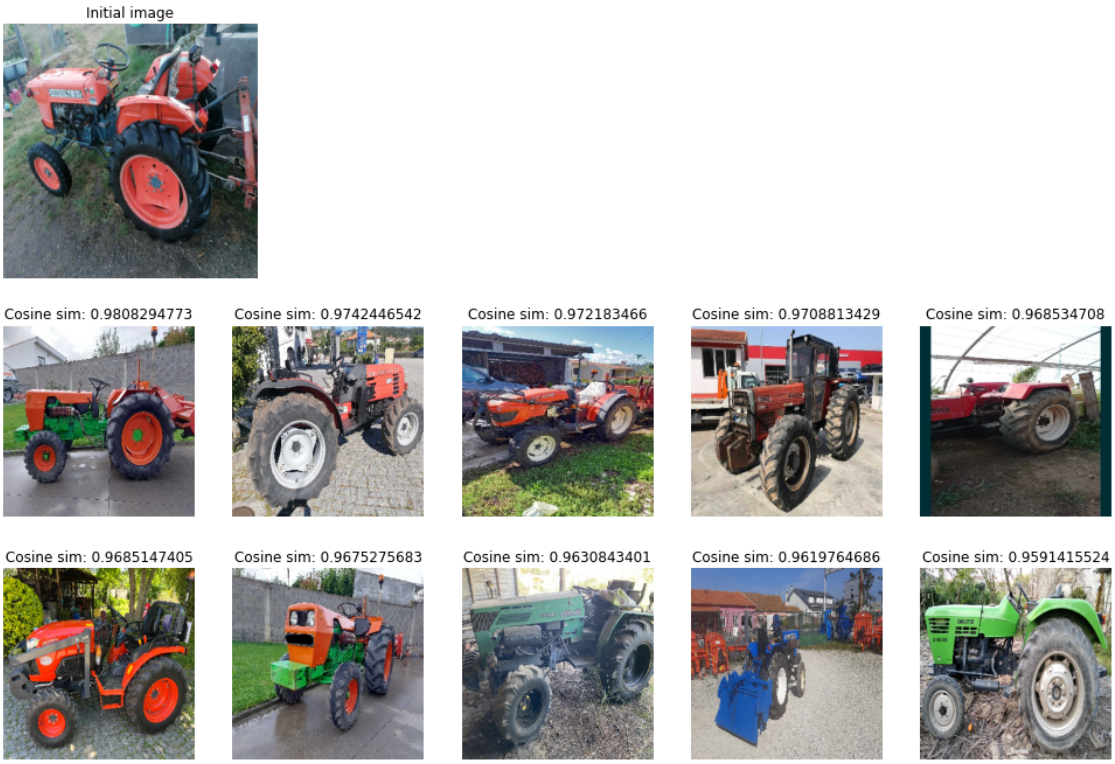


Figure 5.6: Result of cosine similarity with average pooling (3, 3)

Figure 5.7: Result of cosine similarity with global average pooling

From the obtained results, the neural network that produces the best results is the one with average pooling (3, 3). Another example for the selected neural network applied to a different class of images is shown in the Figure 5.8. For this example, the class has 8166 examples. The remaining tests are in the Appendix 6.

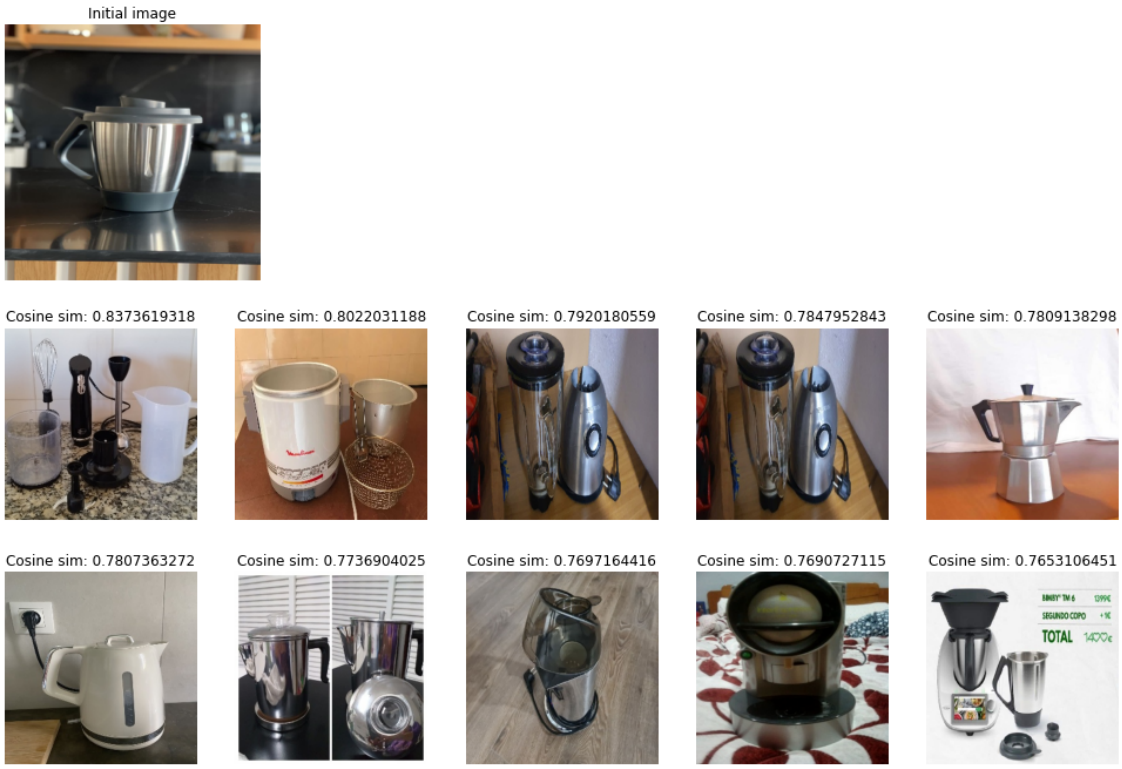


Figure 5.8: Result of cosine similarity with average pooling (3, 3) for other image

5.2.3 Mean average precision

The mean average precision was used to evaluate the results obtained from the recommendation system. This value is a widely used metric in information retrieval and object detection tasks. It measures the average precision across multiple classes or categories. This provides a single scalar value representing the overall performance of the model.

In the particular case of the project, the Mean Average Precision (`MAP`) was measured for each class, and for all classes. For each class, three different images were used to conduct the test. To evaluate, a `MAP` of 5 was used, which means the search is conducted until five relevant images are found. If all the first five images were relevant, the value would be 1. If one image is not relevant at the fifth position and, the sixth position is relevant the result would be: $(1 + 1 + 1 + 1 + (5/6))$. Since the tests were conducted for three

images per class, the final result for each class would be the sum of all values divided by three, which is number of examples. This resulting value for each class is called average precision (`AP`) [25]. The results obtained are shown in Table 5.2.

Class	AP
Agriculture	0.956
Antiques	0.858
Appliances	0.845
Audios	0.918
Bikes	1.0
Books	0.958
Cameras	1.0
Cars	1.0
Cellphone accessories	0.701
Cellphones	0.865
Child	0.914
Clothing	0.940
Computers	0.942

Class	AP
Construction	0.821
Decoration	0.964
DVDs	1.0
Electronics	0.475
Furniture	0.975
Games	0.713
Garden	0.918
Motorbikes	1.0
Real estate	1.0
Sports	0.691
Tablets	0.989
TVs	1.0
MAP	**0.897**

Table 5.2: Mean average precision for each class

5.2.4 Similar images computing time

One important aspect in the recommendation system is the time it takes to produce results. If this takes too long, the user may lose interest in the feature and not use again. To calculate the time, it is necessary to know the dataset size and the hardware it is running on. The dataset size is 174,457. The hardware used was a GPU, specifically the NVIDIA 2060 RTX.

Operation	Time
Calculate distances for 6 images	7.37 seconds
Execution time in django for 6 images	46.56 seconds
Execution time in django for 3 images	26.45 seconds
Execution time in django for 2 images	18.11 seconds.
Execution time in django for 1 image	7.02 seconds.

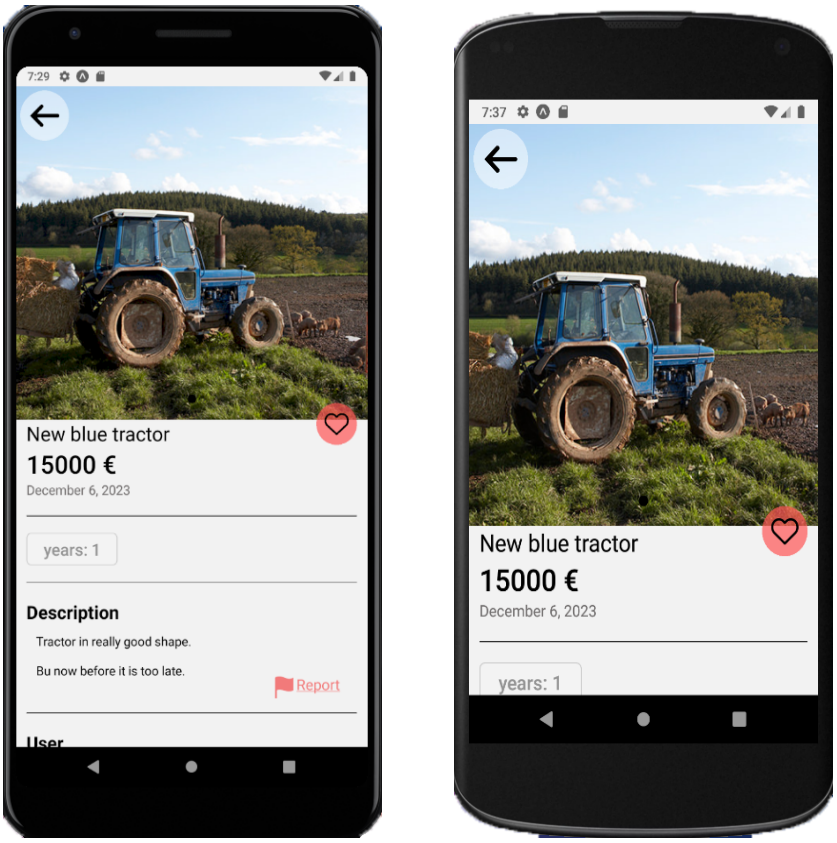
Table 5.3: Execution times for the neural network with average pooling (3, 3)

In Table 5.3, the second row refers to the time taken to classify 6 images and calculate the distances between each image sent and all the images belonging to the classified class. The following rows correspond to the time taken to classify the images, calculate the distances, and search for the product that corresponds to the images. With the results obtained in the previous table, it is possible to see that the time taken to calculate distances is much smaller than the whole execution of the request in `Django`. In reality, this is because most of the images used do not contain any product information. The program has to iterate through all the products stored locally with a text file, only to find nothing. This problem would be much quicker if the information was stored in the database, requiring just a query to retrieve the information from the database.

This process time can vary depending on the demand from the users, which means that the table represents the least amount of time possible, because it is only one user sending requests. Thus, it is necessary a way to entertain the user while he is waiting. This must be incorporated in the frontend and can include visual cues like a loading animation or a real-time progress percentage. These elements transform potential frustration into an interactive experience, ensuring users stay connected until the process ends.

5.3 Application responsive design

Responsive design refers to the application's ability to fit and adapt to screens on multiple devices. To test this, multiple emulators were installed with different sizes and resolutions. The results are shown in Figure 5.9. The remaining tests for responsive design are in the Appendix .3.



(a) Mobile phone with 1080x2160 (b) Mobile phone with 768x1280

Figure 5.9: Images from two different phones

Although on the specific page shown, the application fits well in those two screens. There are still some pages that do not adjust well to fit on all screens. But overall, most of the screens work fine in different devices with various sizes and resolutions.

Chapter 6

Conclusions and Future Work

In this project, it was possible to create an application from the ground up, following the good practices both of software engineering as well as REST APIs.

The creation of the software architecture before writing the actual code is indispensable for the success of the project. This provides a clear understanding of the project requirements, creating a clearer implementation, and promoting a better long-term maintainability.

The development of the backend server using `NodeJS` with `ExpressJS` to create REST APIs with JavaScript. The fact that both the backend and frontend use the same language creates a more seamless and cohesive codebase.

Using MongoDB as database brings flexibility due to its NoSQL design, promoting scalable implementations and making it a good choice for data-intensive projects.

The usage of react native allows to write code and deploy both on Android and iOS. The framework hot reloading accelerates the development by providing real-time updates.

Finally, the creation of the recommendations system involves using an endpoint in `Django` to access a neural network powered by `Keras` and `TensorFlow`.

This enhances the functionality of the application while also cultivating skills in building AI features.

In terms of results, all the features that were intended to implement in the application were implemented. All the functionalities were tested, and the results were good. Both the application and recommendation system worked, even with the limited size of the database. Both backend and frontend worked according to the software architecture. The recommendation system obtained a MAP of 0.89, taking into account 25 classes of products.

The future work is mostly about fixing some issues currently present in the application and implementing additional functionalities. The first task is to implement unit tests on the server side. The second task is to migrate the project to the cloud, making real-time recommendations possible. The third task is to improve the speed on the frontend side, as the application suffers from performance issues and needs to be fixed. The rest of the future work involves all the processes to bring the application to the market. This includes deploying it in the cloud with a load balancer using `Kubernetes`, validating the design with users and making necessary changes based on their feedback. Finally, the creation of the quality assurance tests is needed.

The recommendation system could also be improved. Firstly, the number of classes could be increased to prevent similar images from being placed in different classes. The second possible improvement is to decide on a metric when two or more classes contain a high value in the classification. This can happen when an image could belong to more than one class. An algorithm could be created to decide how many images from each class could be calculated as similar images.

Bibliography

[1] Sumit Saha. What is Convolutional Neural Networks (CNN). (2023, March 22).
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[2] Red Hat. What is REST APIs. (2023, April 08).
https://www.redhat.com/en/topics/api/what-is-a-rest-api

[3] Wallapop mobile application. (2023, March 18).
https://play.google.com/store/apps/details?id=com.wallapop&hl=en_US&pli=1

[4] Etsy mobile application. (2023, March 19).
https://play.google.com/store/apps/details?id=com.etsy.android&hl=en_US

[5] OLX mobile application. (2023, March 20).
https://play.google.com/store/apps/details?id=com.fixeads.olxportugal&hl=en_US

[6] Google images. (2023, March 21).
https://images.google.com/

[7] Google Lens mobile application. (2023, March 22).
https://play.google.com/store/apps/details?id=com.google.ar.lens&hl=pt_PT&gl=US

[8] CamFind mobile application. (2023, March 22).
https://play.google.com/store/apps/details?id=com.msearcher.camfind&hl=en_US

[9] Luyang C., Yang F., Heqing Y., (2019). Image-based Product Recommendation System with Convolutional Neural. for CamFind mobile application.

[10] IBM. What is the Vision document in software architecture. (2023, April 07).
https://www.ibm.com/docs/en/engineering-lifecycle-management-suite/lifecycle-management/7.0.1?topic=requirements-vision-document

[11] Usability.gov. What is the Use Cases, benefits and how to create them. (2023, April 15).
https://www.usability.gov/how-to-and-tools/methods/use-cases.html

[12] toughtworks. What is the Domain Model and how to build it. (2023, May 01).
https://www.thoughtworks.com/insights/blog/agile-project-management/domain-modeling-what-you-need-to-know-before-coding

[13] VisualParadigm. What are the Sequence Diagrams and how create them. (2023, May 10).
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/

[14] VisualParadigm. What are the Class Diagrams and how create them. (2023, May 29).
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

[15] Sparx. What are State Machine Diagrams and how to build them. (2023, June 09).
https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html

[16] MongoDB. Link to use MongoDB Atlas. (2023, December 23).
https://www.mongodb.com/atlas/database

[17] A. Smith. What are the differences between NodeJs and ExpressJS. (2023, June 15).
https://web-and-mobile-development.medium.com/what-are-the-prime-differences-between-node-js-and-express-js-b560b19b8b33

[18] Mongoose. How to use 'Schema' in MongoDB. (2023, June 21).
https://mongoosejs.com/docs/validation.html

[19] crowdstrike. How to use Logging in NodeJS. (2023, June 25).
https://www.crowdstrike.com/guides/nodejs-logging/

[20] OpenSSL. Link to download OpenSSL. (2023, June 30).
https://www.openssl.org/source/

[21] Keras. Keras available models with their performances. (2023, July 09).
https://keras.io/api/applications/

[22] Medium. Querying similar images with TensorFlow. (2023, July 17).
https://medium.com/@luchonaveiro/querying-similar-images-with-tensorflow-59e3a7aad40e

[23] Expo. What is expo and core concepts. (2023, August 01).
https://docs.expo.dev/core-concepts/

[24] LogRocket. How to implement a map in React Native. (2023, August 10).
https://blog.logrocket.com/react-native-maps-introduction/

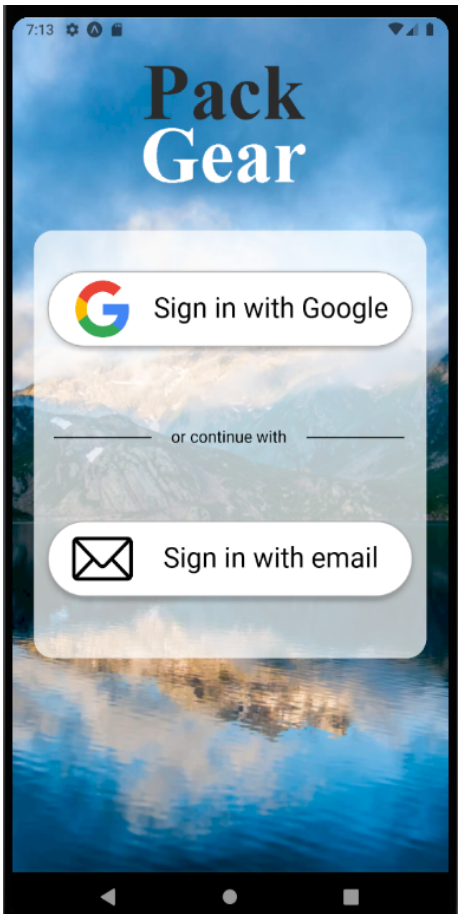
[25] Ren Jie Tan. Formula to calculate the Mean Average Precision (MAP). (2023, November 13).
https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52

Links for the Figma

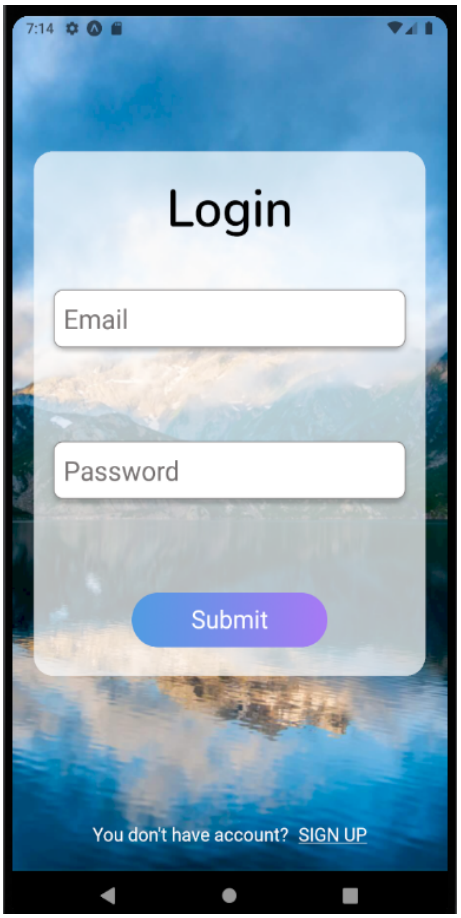
Wireframes link: Wireframes.

Mockups link: Mockups.

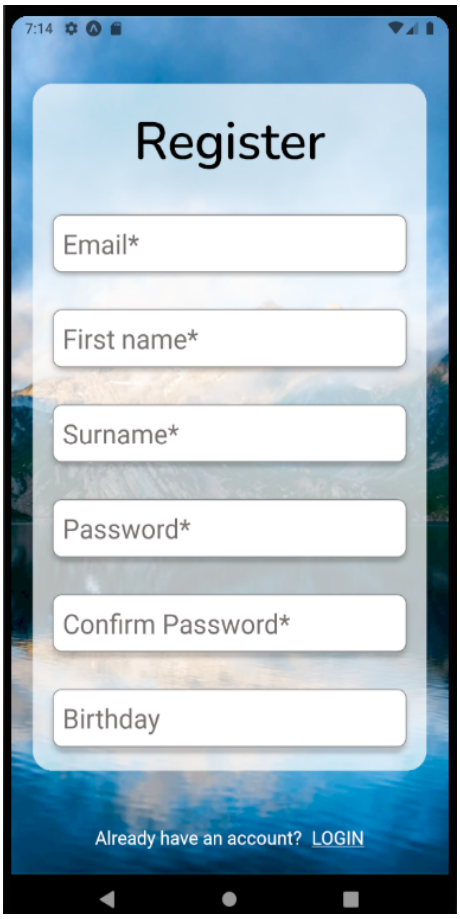
Resulting pages implemented



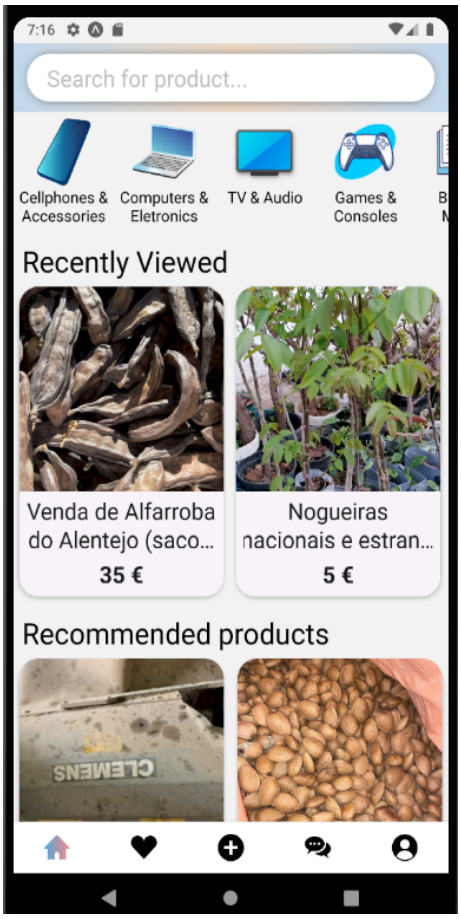
(a) Page select login



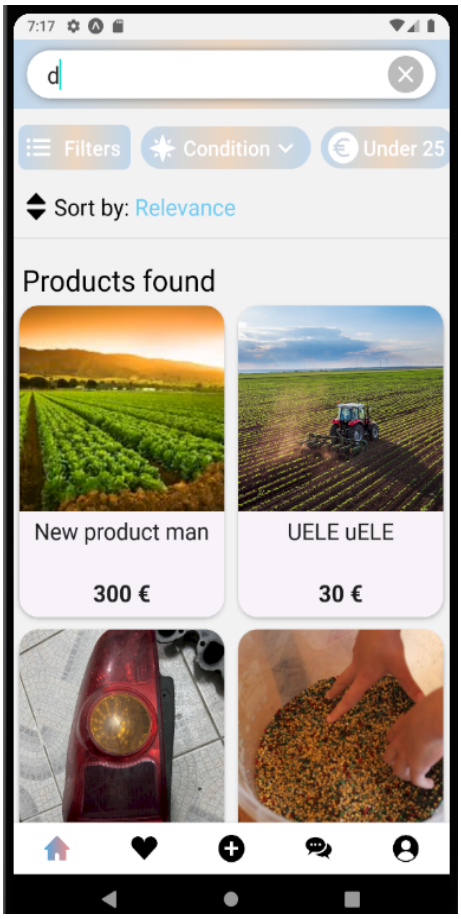
(b) Page login



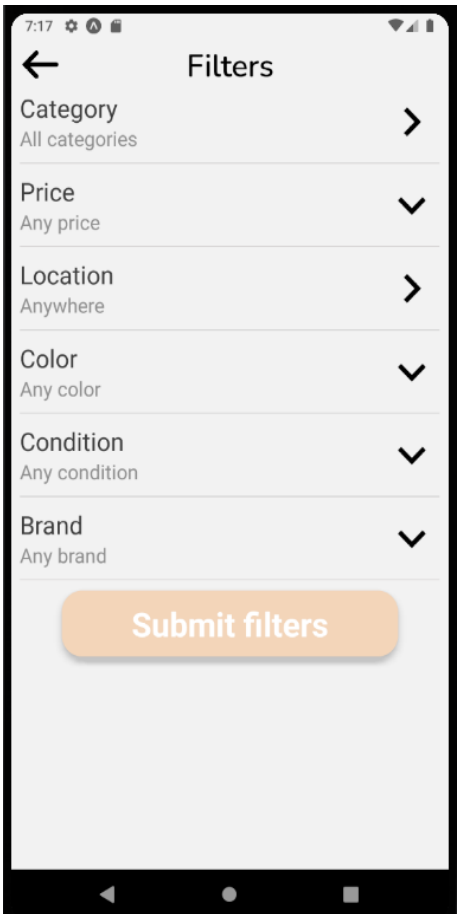
(c) Page register



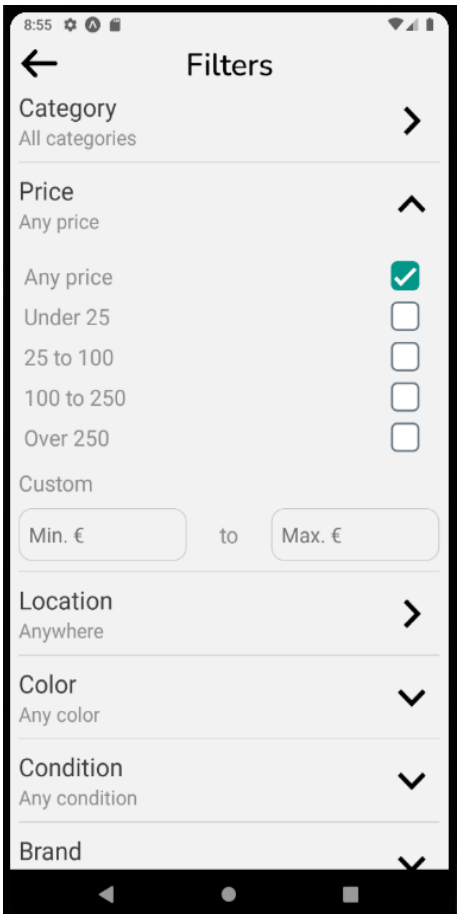
(d) Page homepage



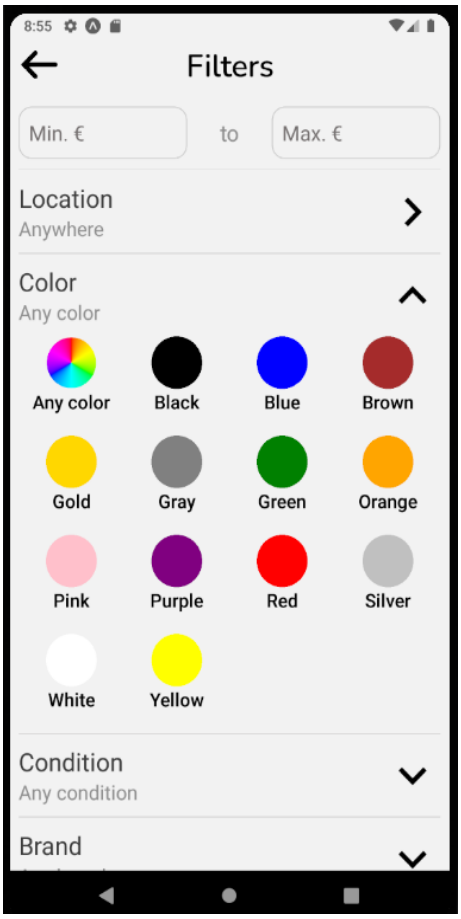
(e) Page homepage search



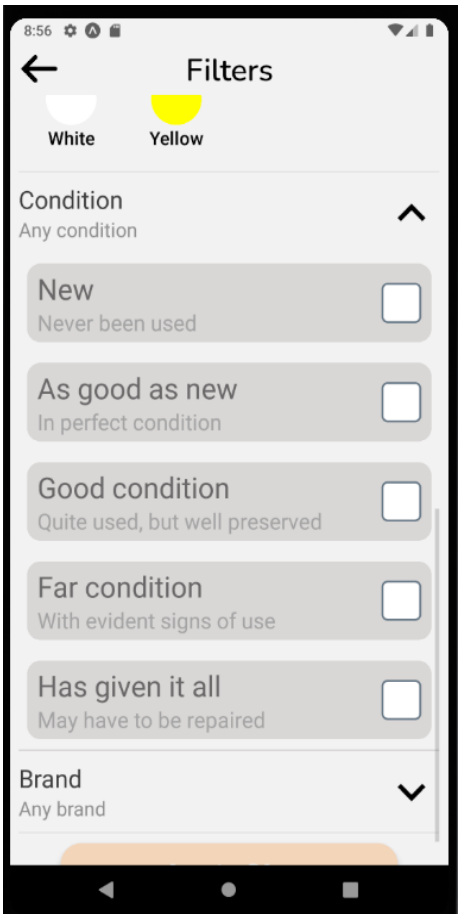
(f) Page filters



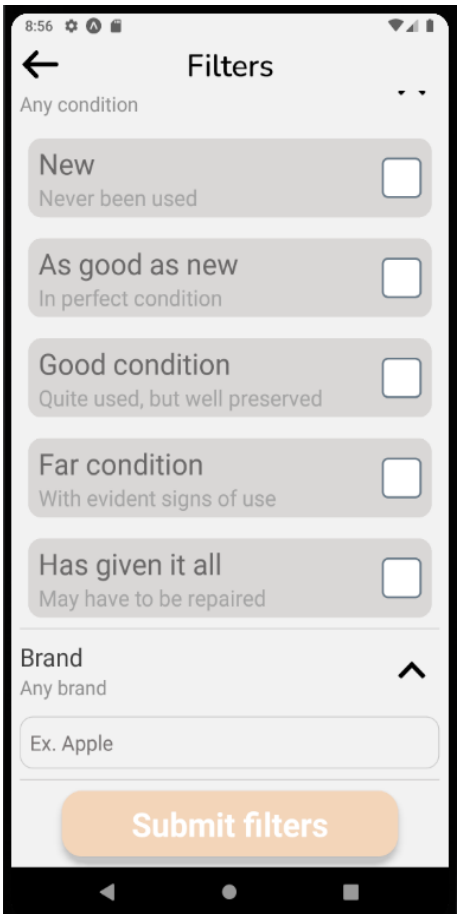
(g) Page filters 2



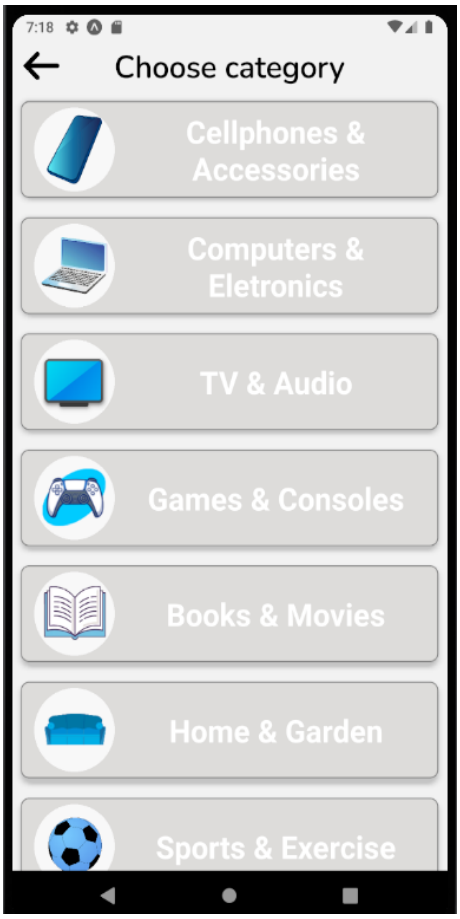
(h) Page filters 3



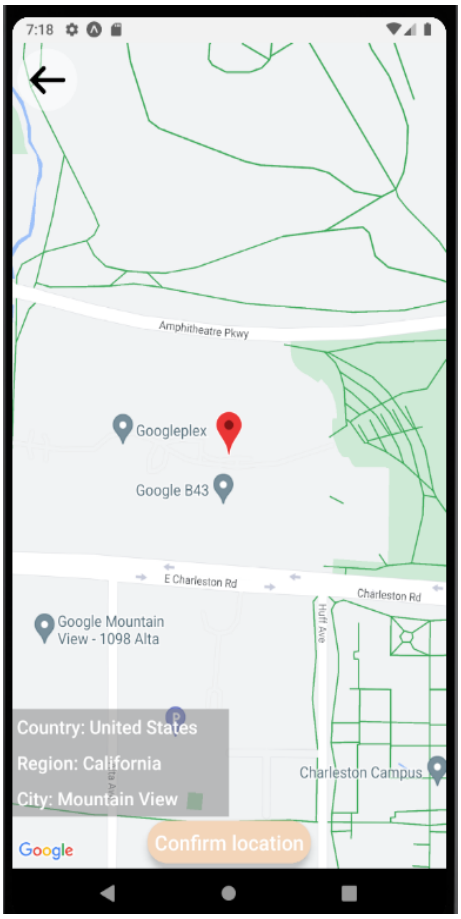
(i) Page filters 4



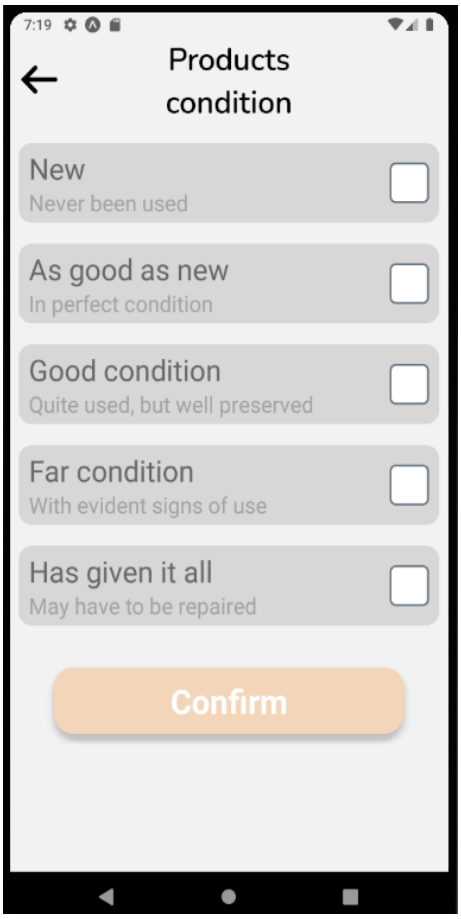
(j) Page filters 5



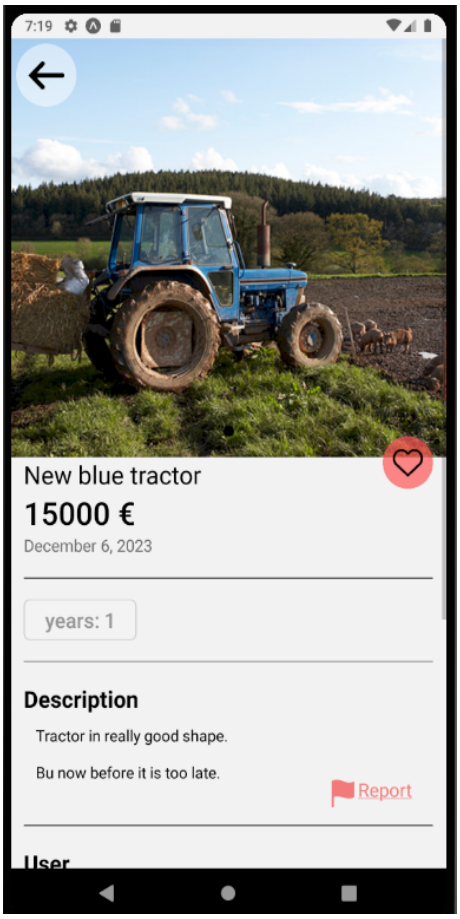
(k) Page select category



(l) Page location

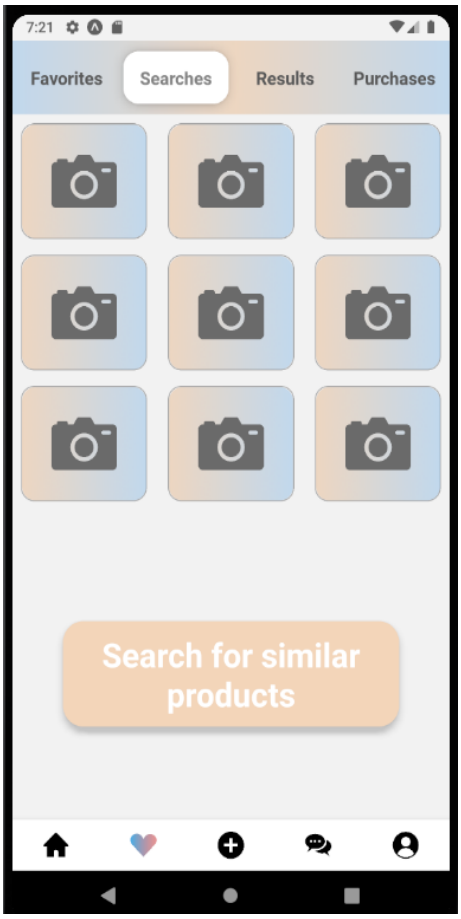


(m) Page select condition

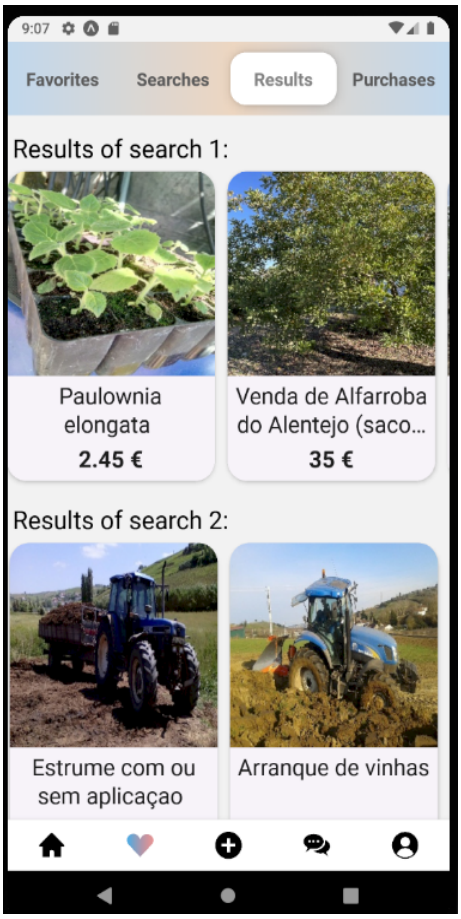


(n) Page product details

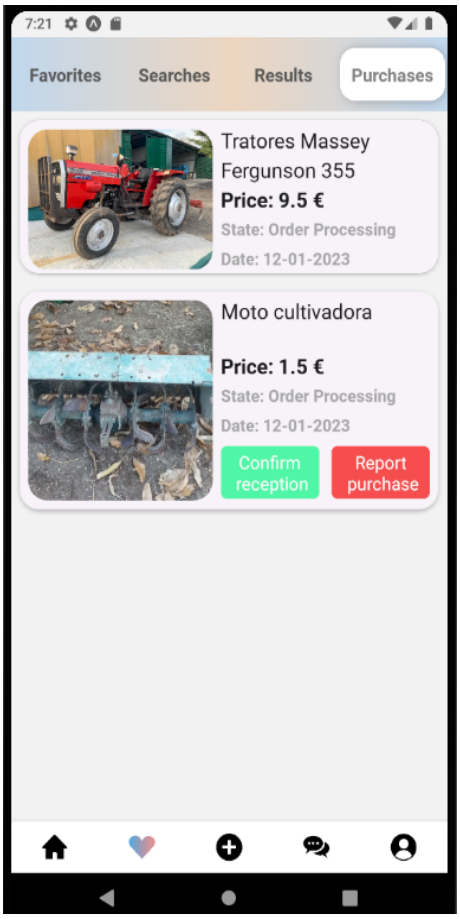
(o) Page favorites



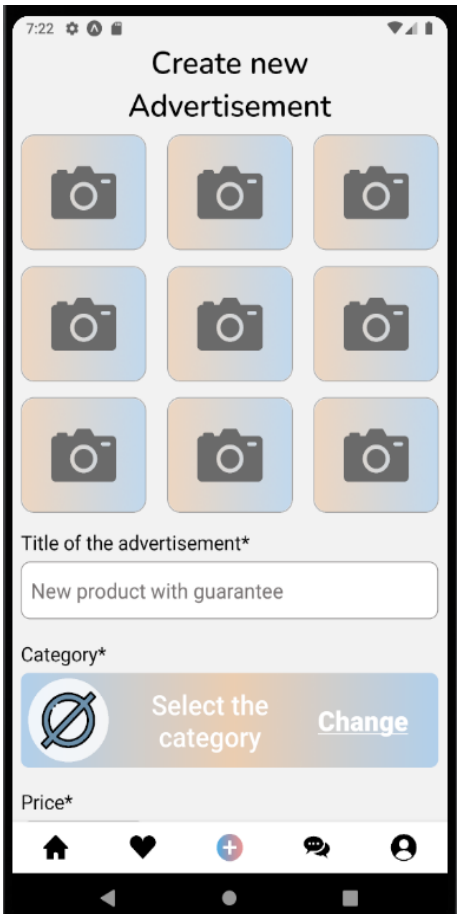
(p) Page search for similar products



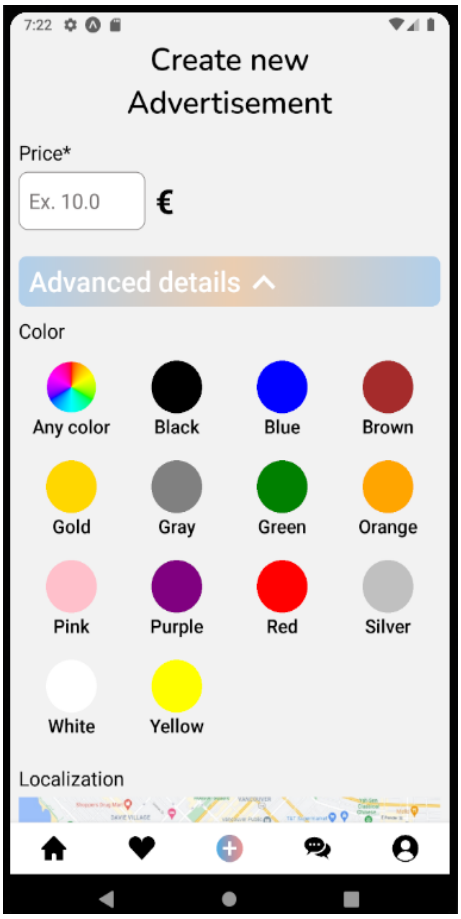
(q) Page result similar products



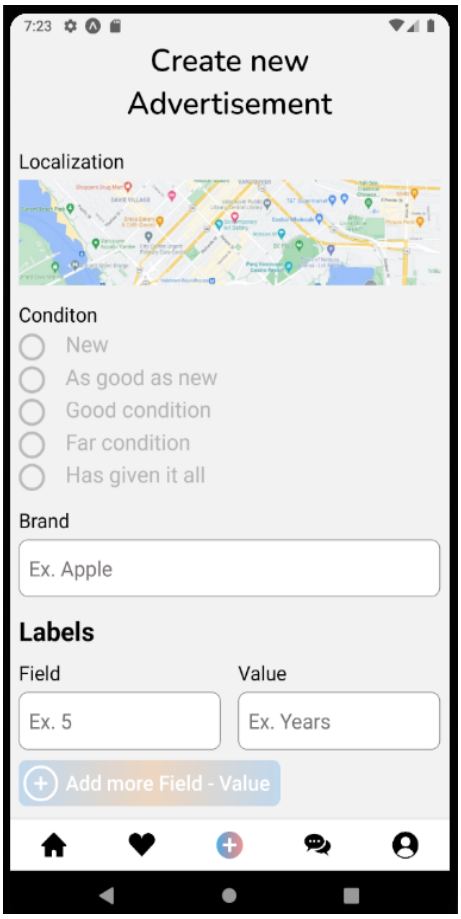
(r) Page purchases



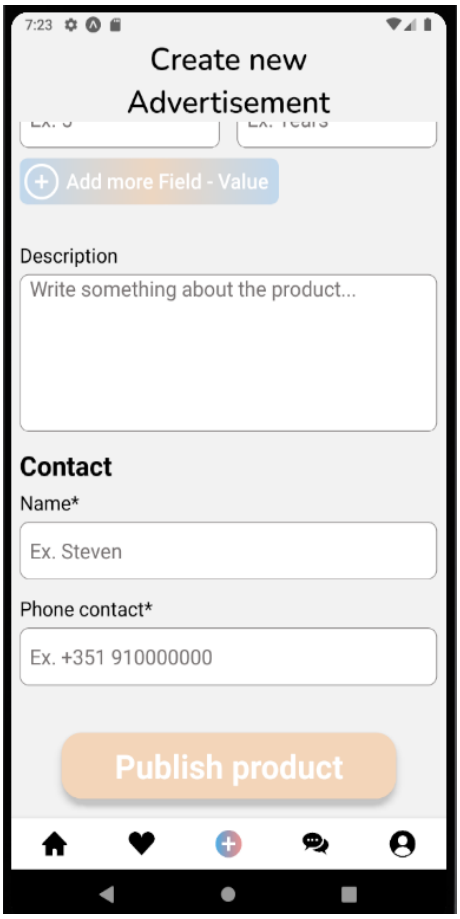
(s) Page add product



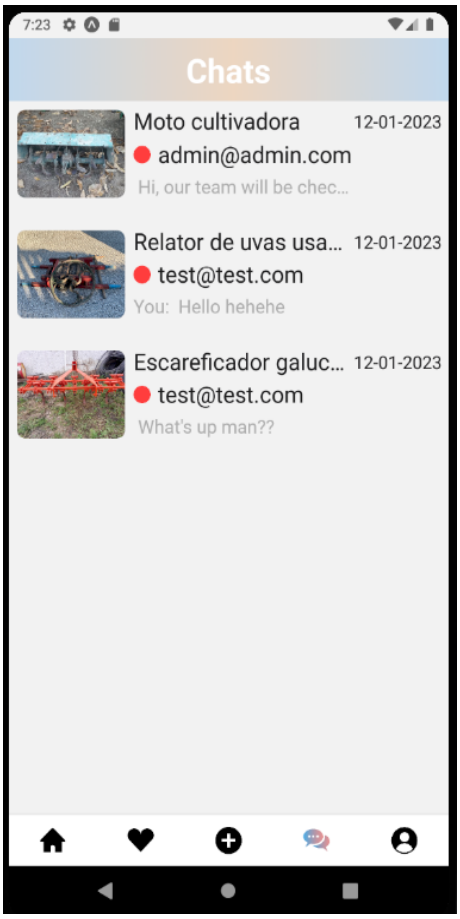
(t) Page add product 2



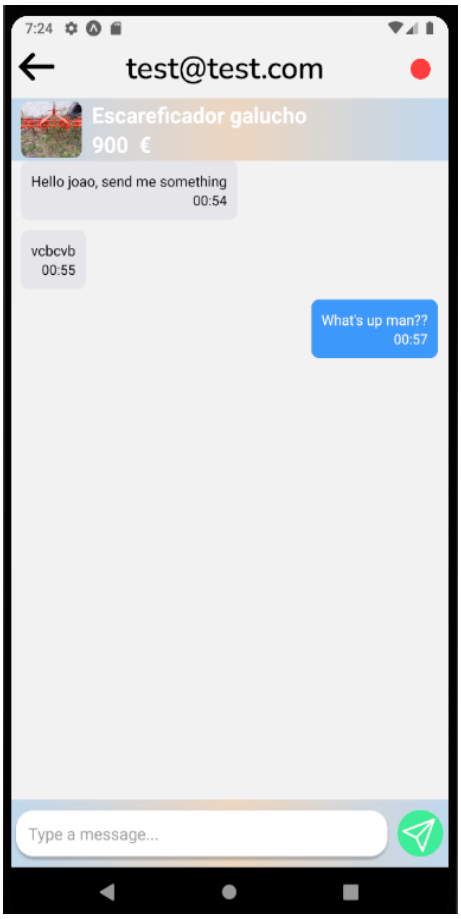
(u) Page add product 3



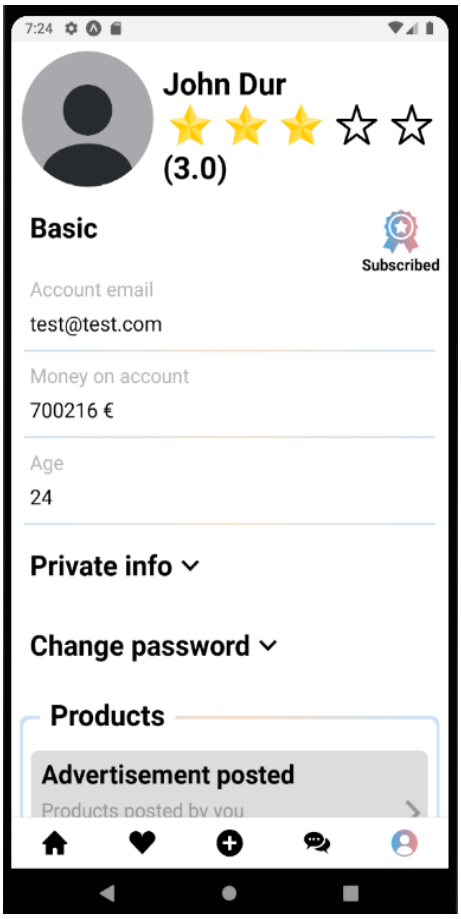
(v) Page add product 4



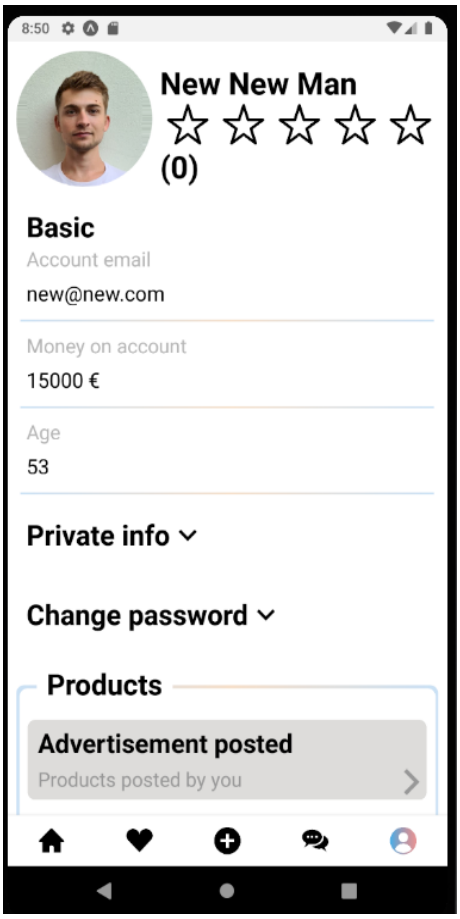
(w) Page inbox conversations



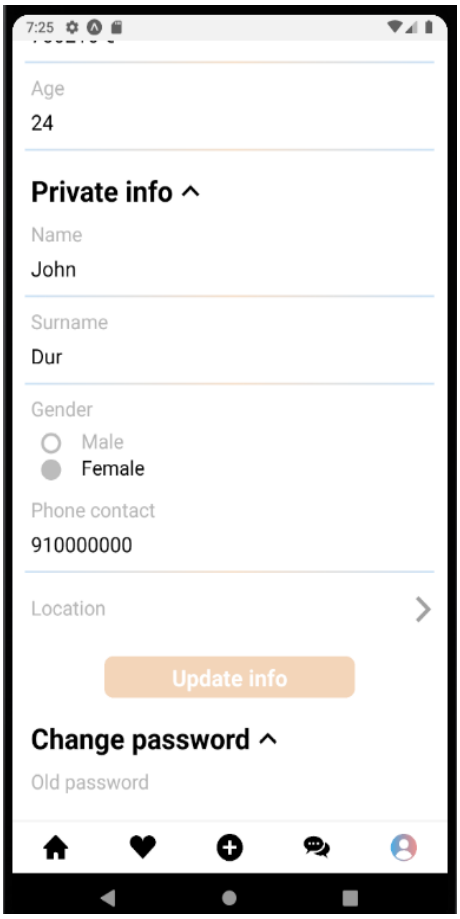
(x) Page conversation messages



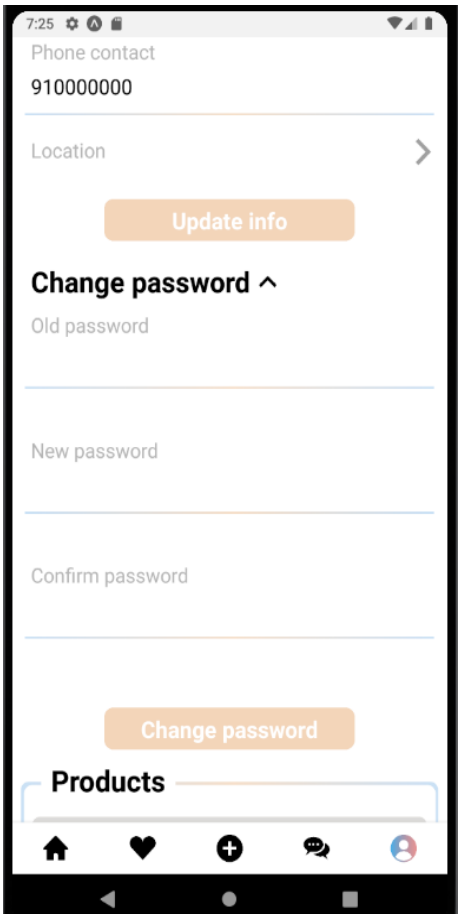
(y) Page profile



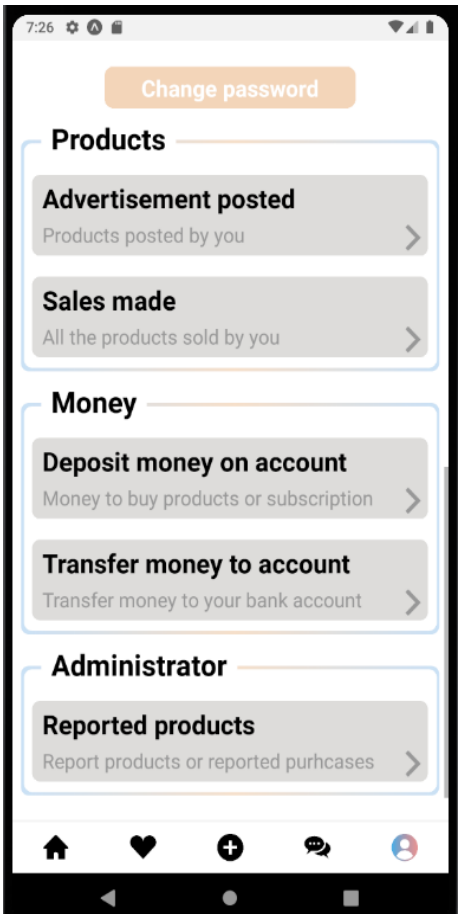
(z) Page profile 2



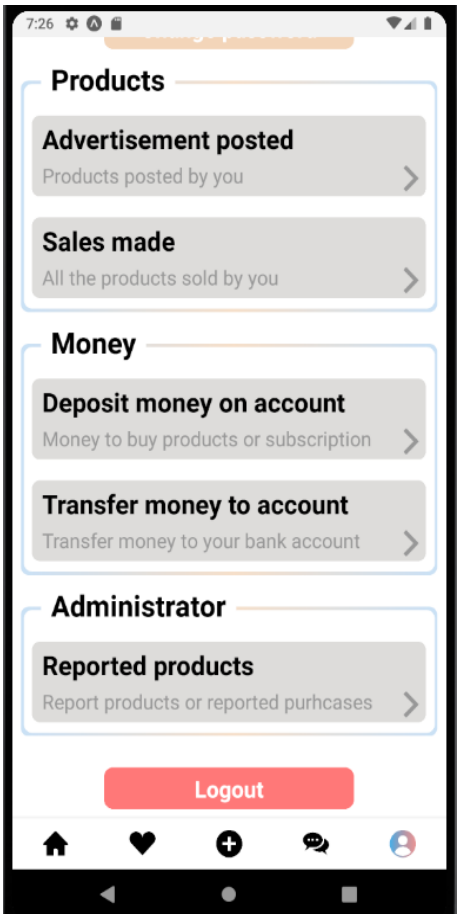
() Page profile 3



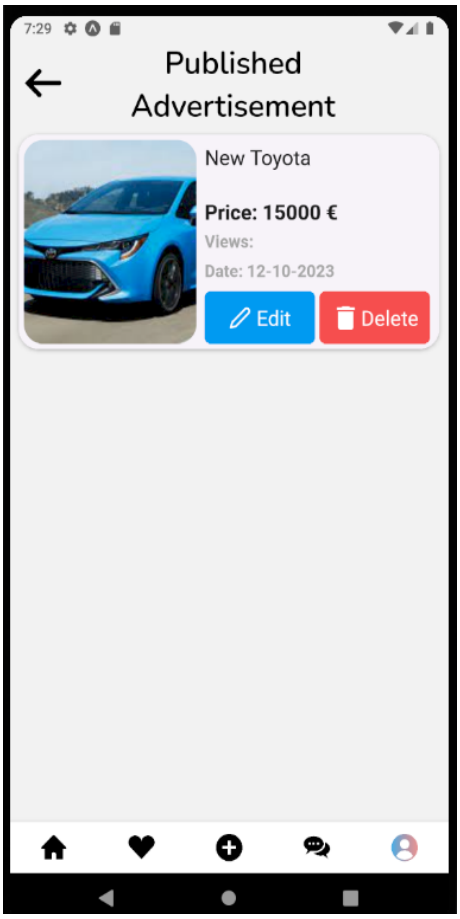
() Page profile 4



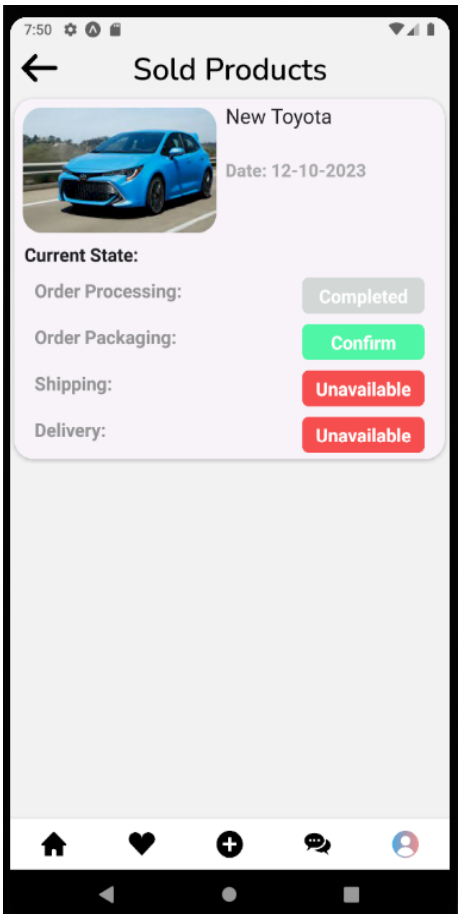
() Page profile 5



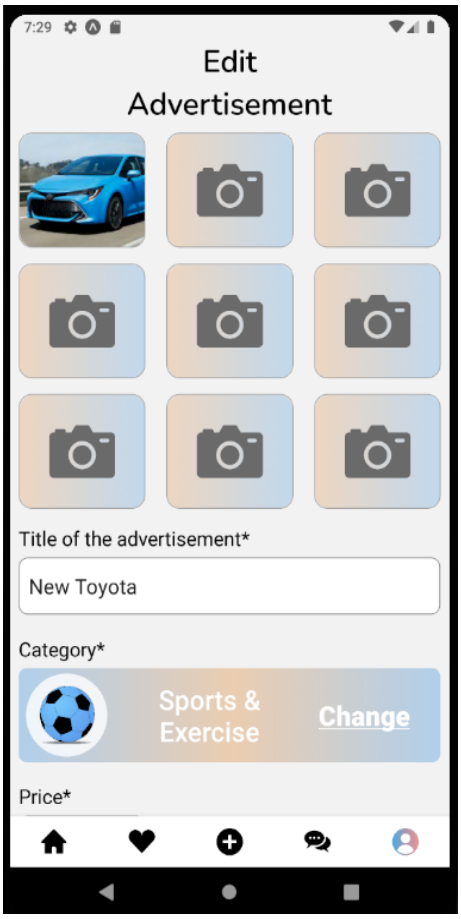
() Page profile 6



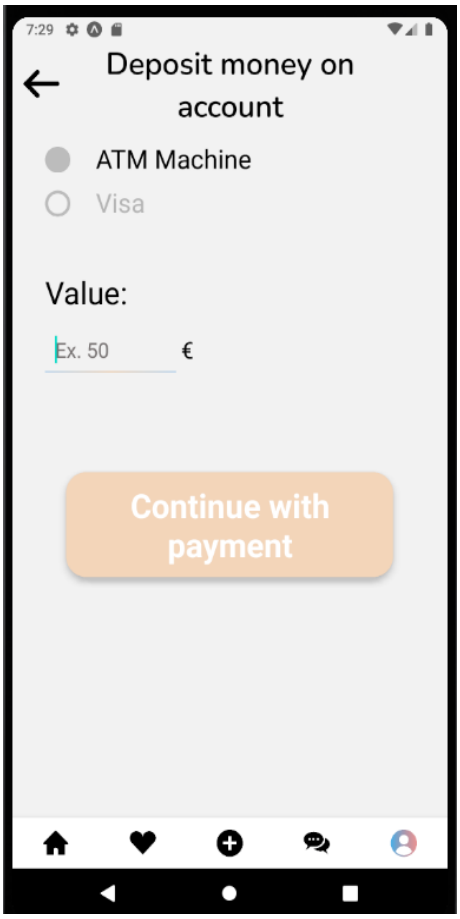
() Page published products



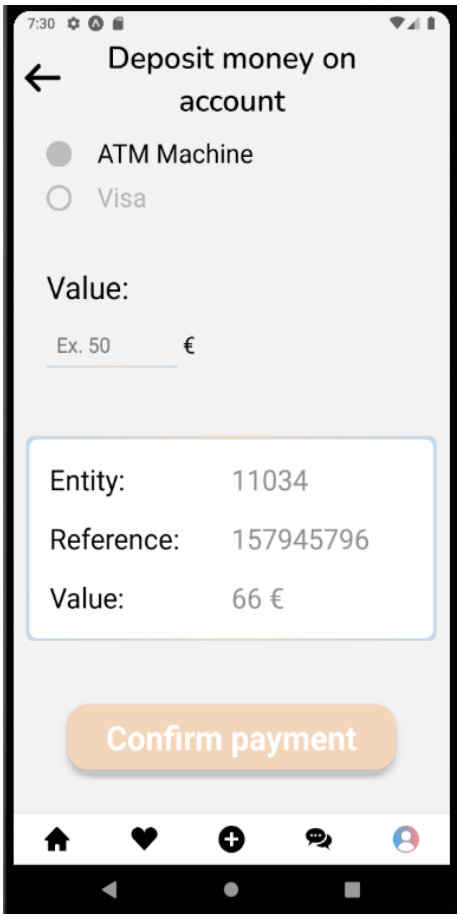
() Page change product state



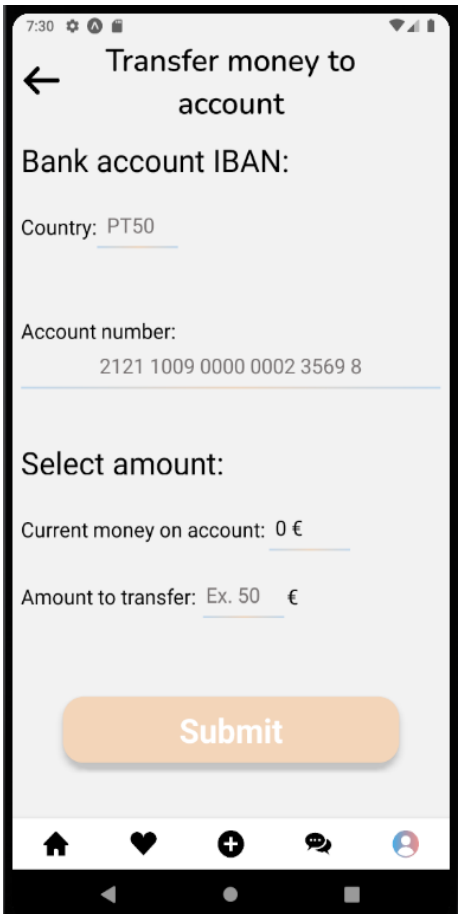
() Page edit product



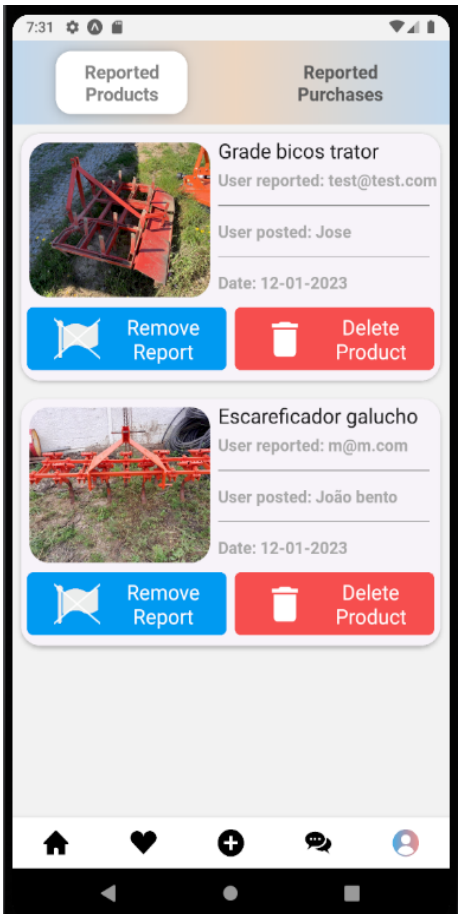
() Page deposit money



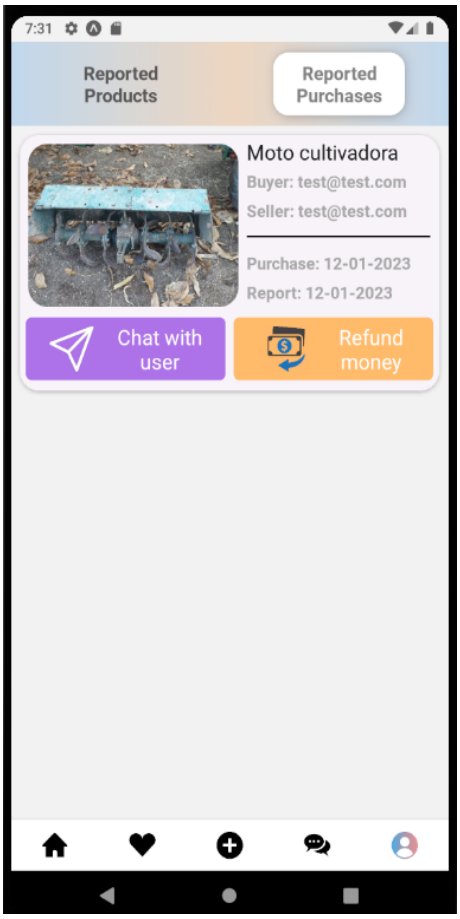
() Page deposit money 2



() Page transfer money



() Page report product



() Page report purchase

Result of similar images for multiple examples

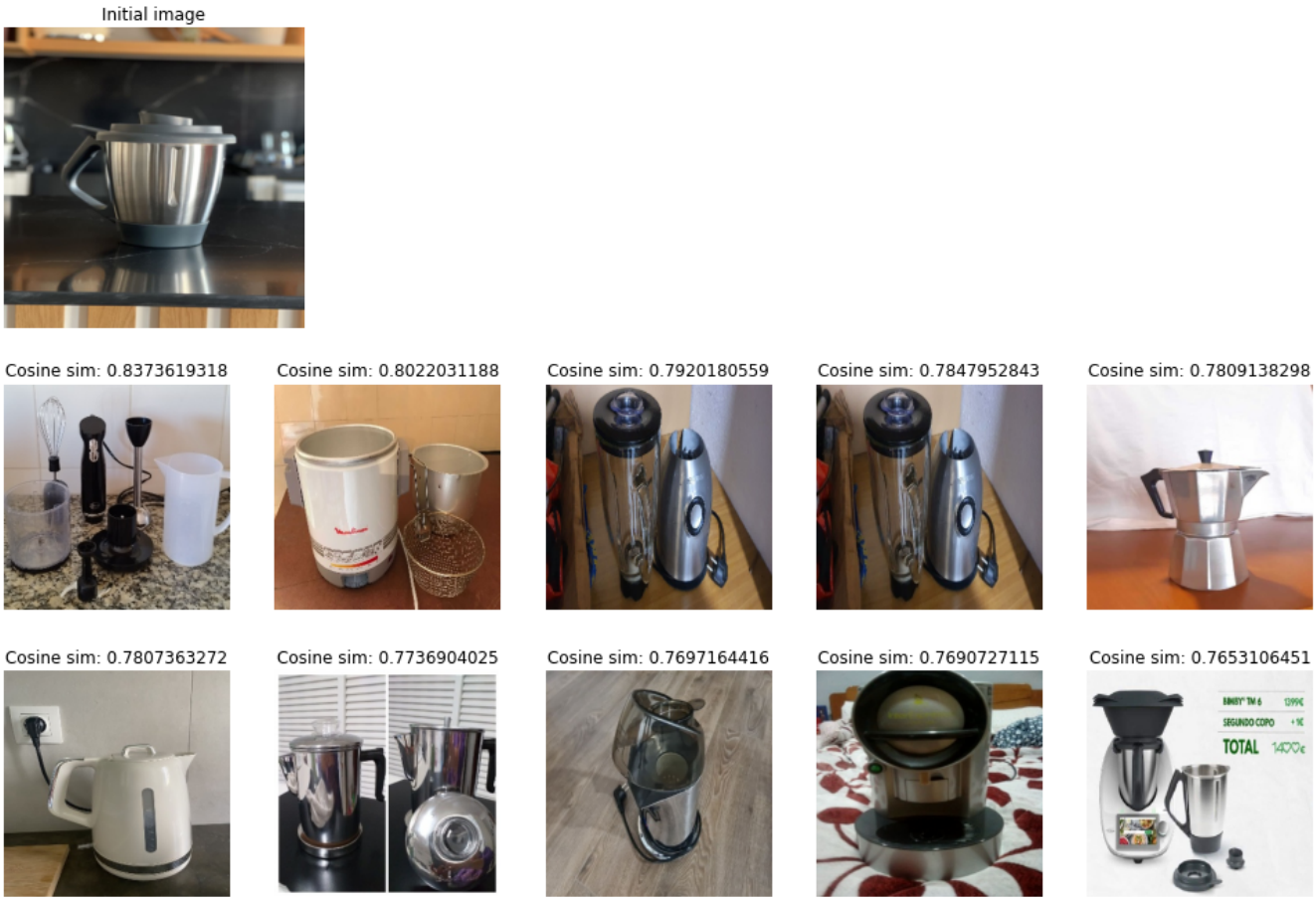


Figure 1: Result of similar images for Average pooling (3, 3) ex.1

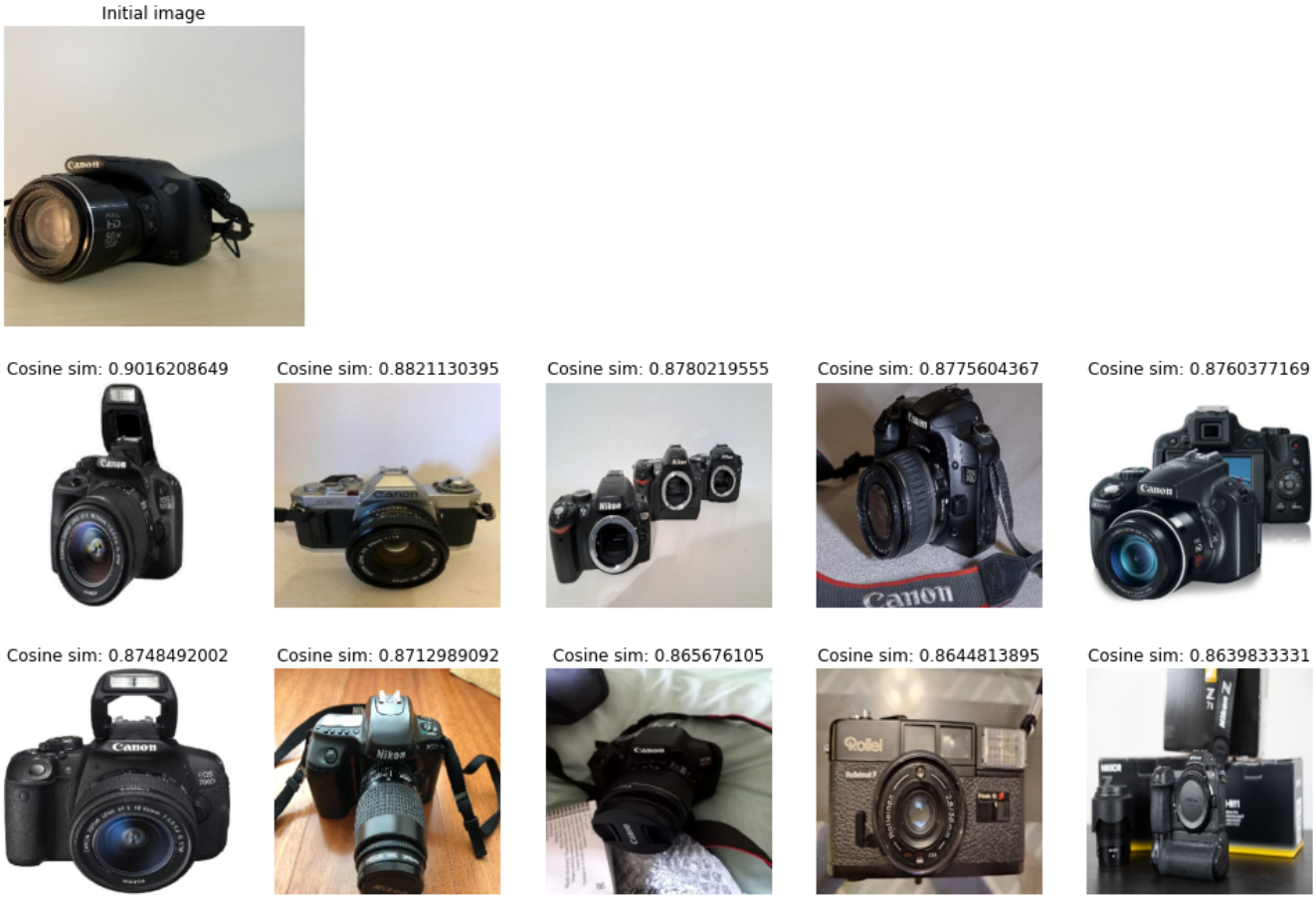


Figure 2: Result of similar images for Average pooling (3, 3) ex.2

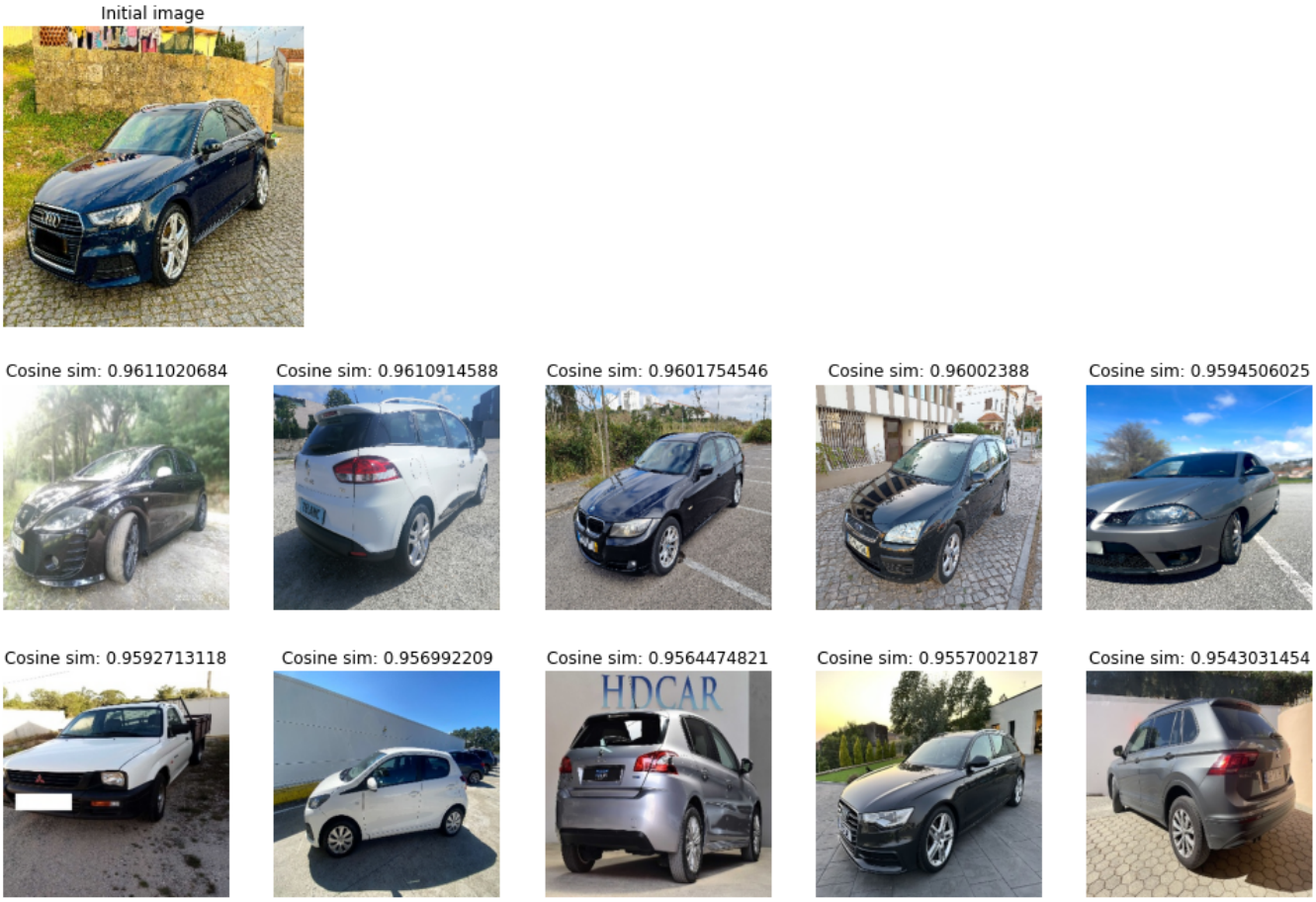


Figure 3: Result of similar images for Average pooling (3, 3) ex.3

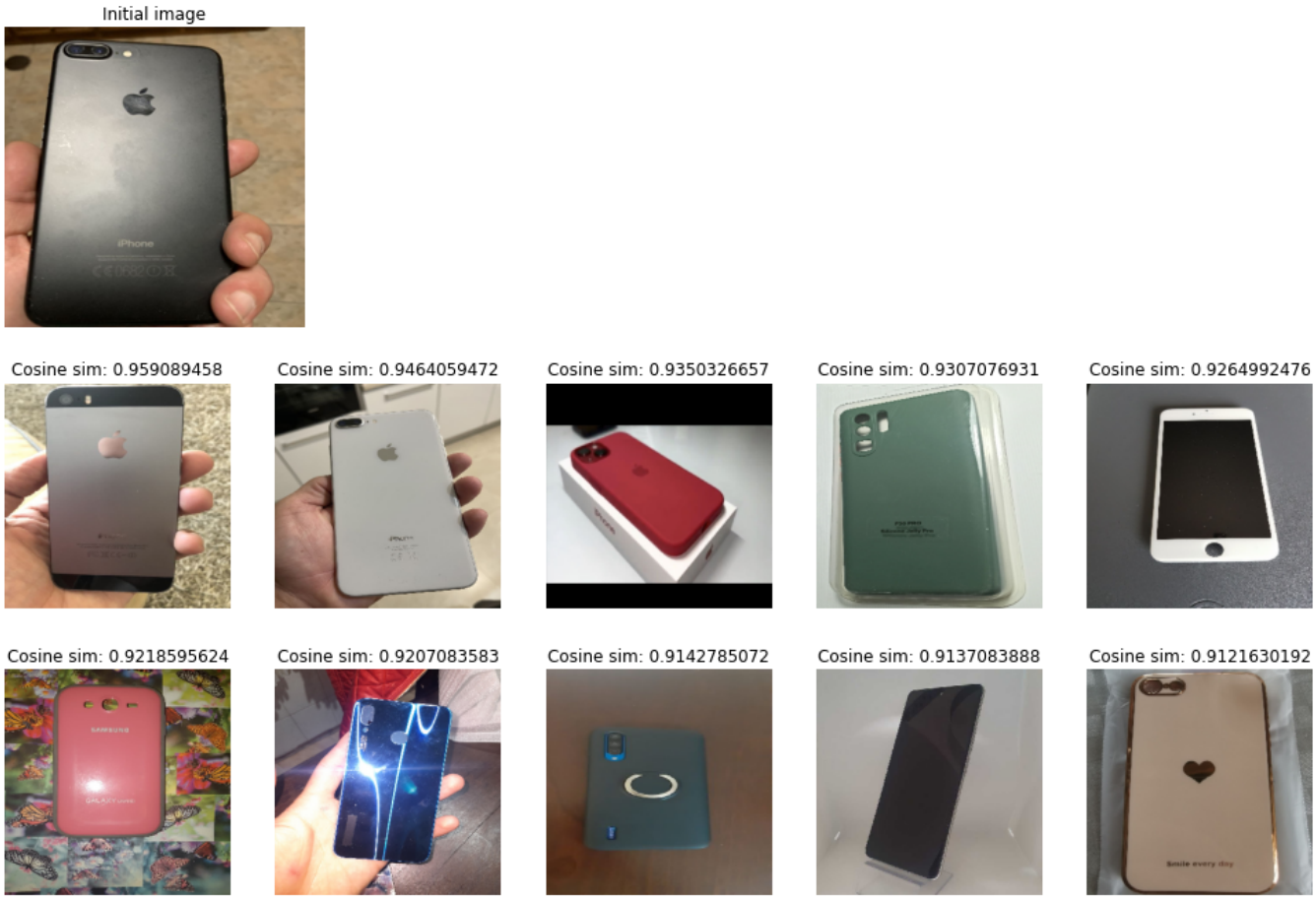


Figure 4: Result of similar images for Average pooling (3, 3) ex.4

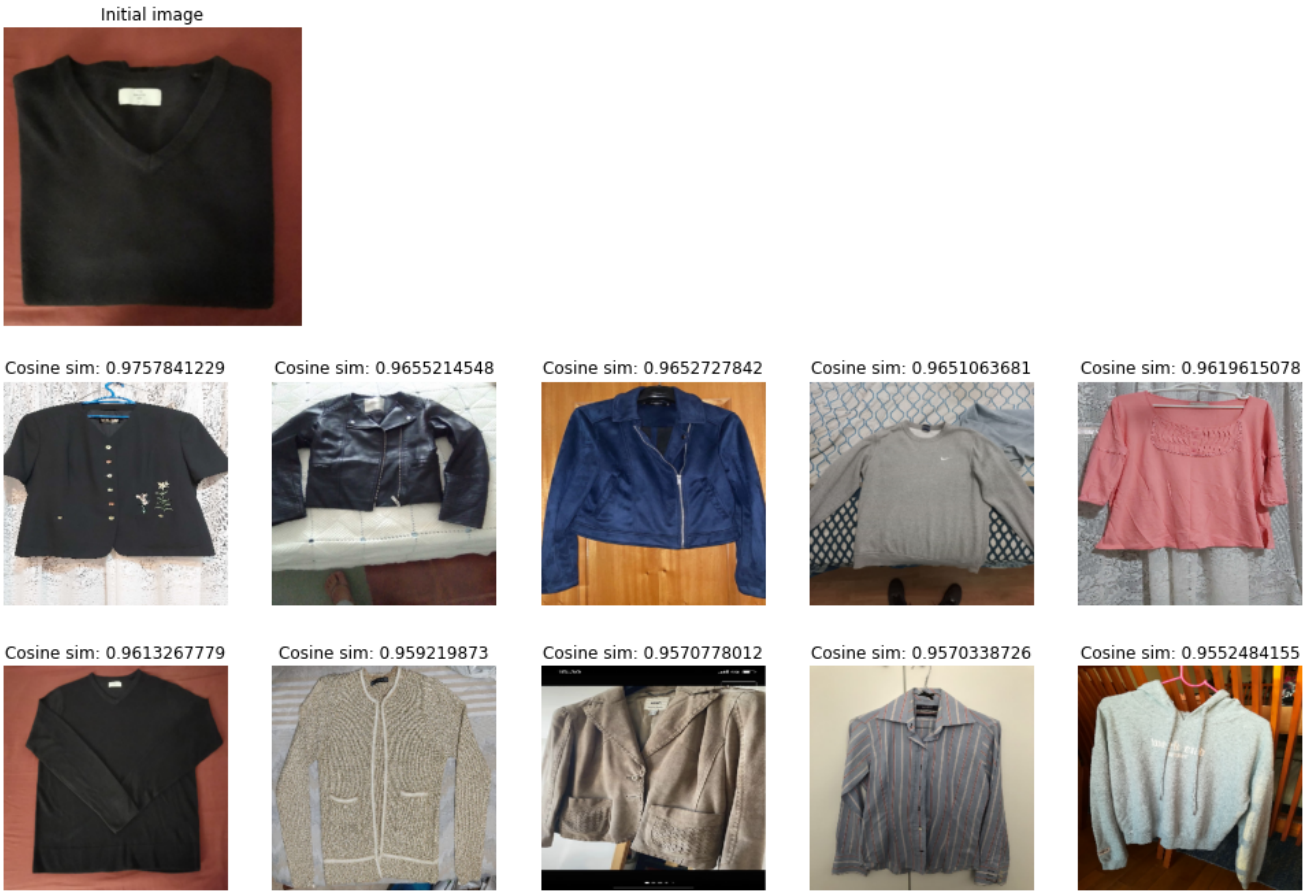


Figure 5: Result of similar images for Average pooling (3, 3) ex.5

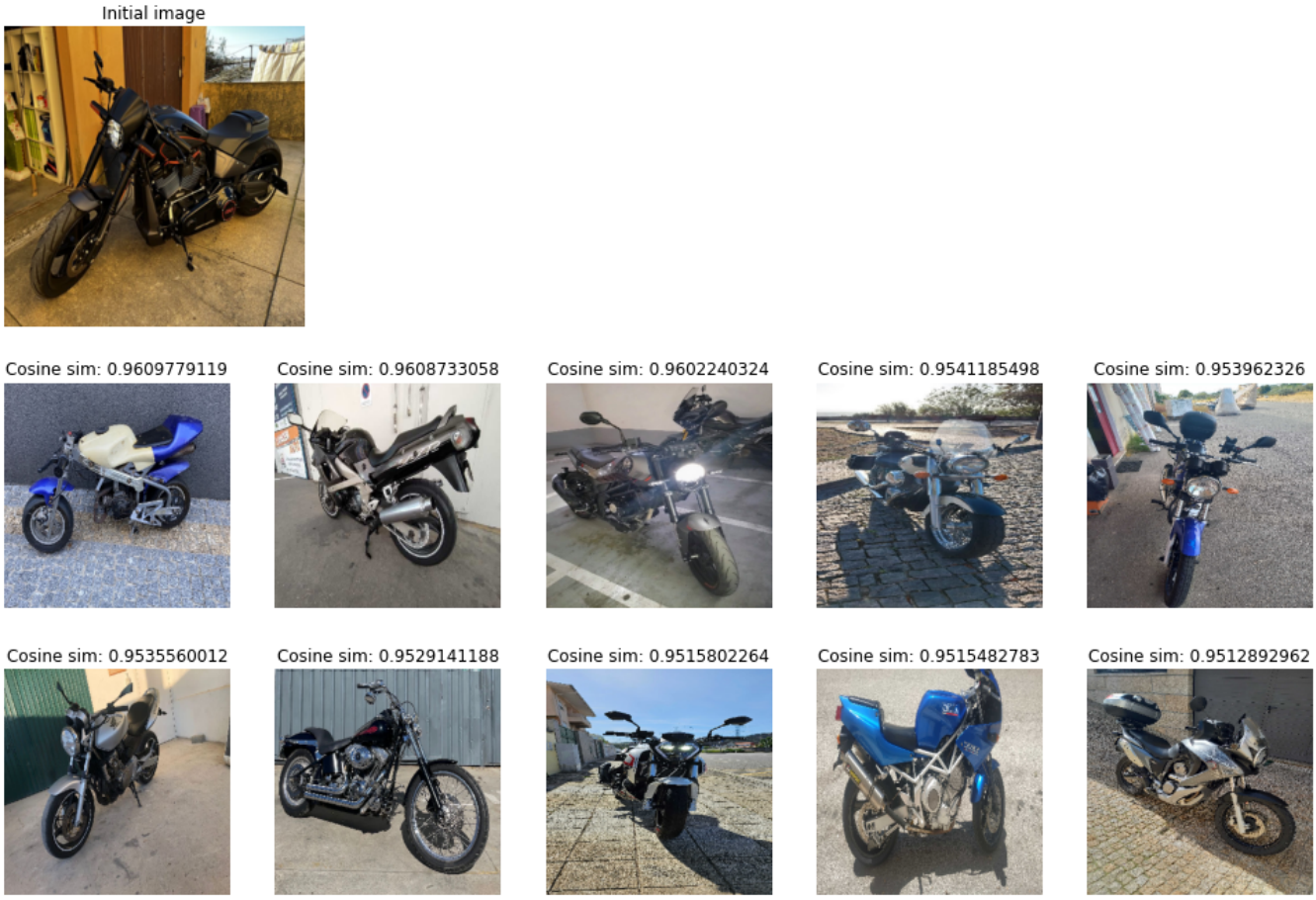


Figure 6: Result of similar images for Average pooling (3, 3) ex.6

All the similar images tests

.1 Test on tractor

Figure 7: Result of cosine similarity and Average pooling (2, 2)

Figure 8: Result of euclidean distance and Average pooling (2, 2)

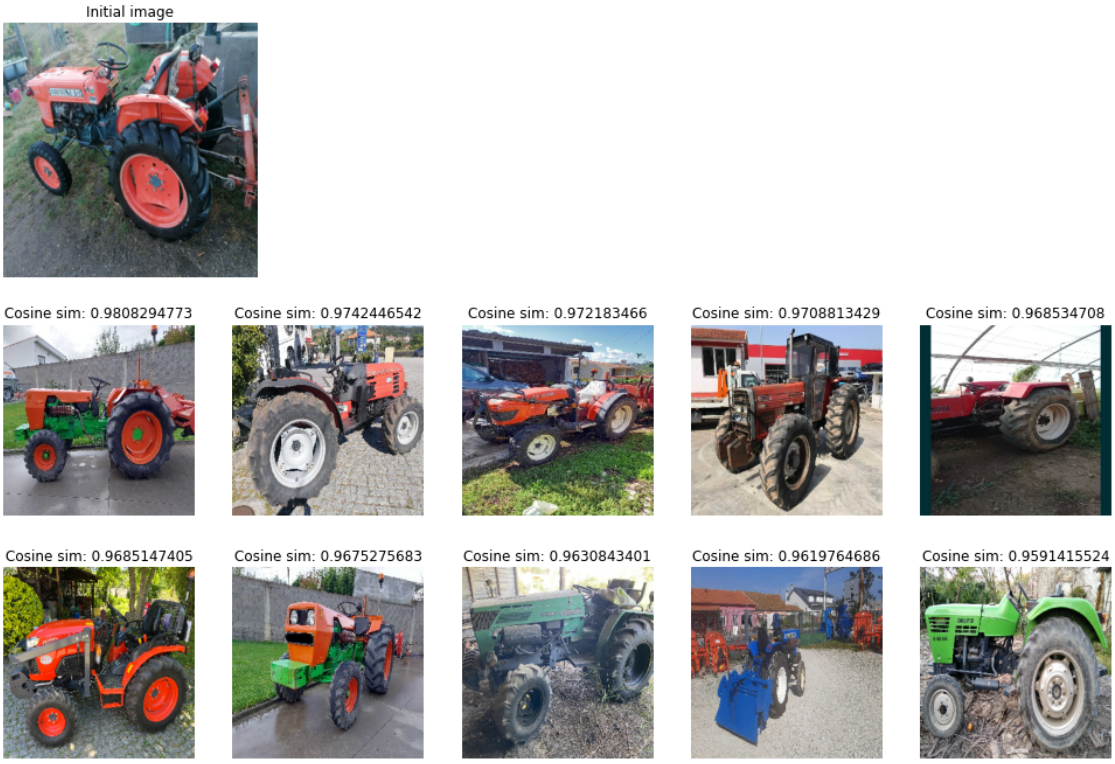


Figure 9: Result of cosine similarity and Average pooling (3, 3)

Figure 10: Result of euclidean distance and Average pooling (3, 3)

Figure 11: Result of cosine similarity and Global Average pooling

Figure 12: Result of euclidean distance and Global Average pooling

.2 Test on a bike

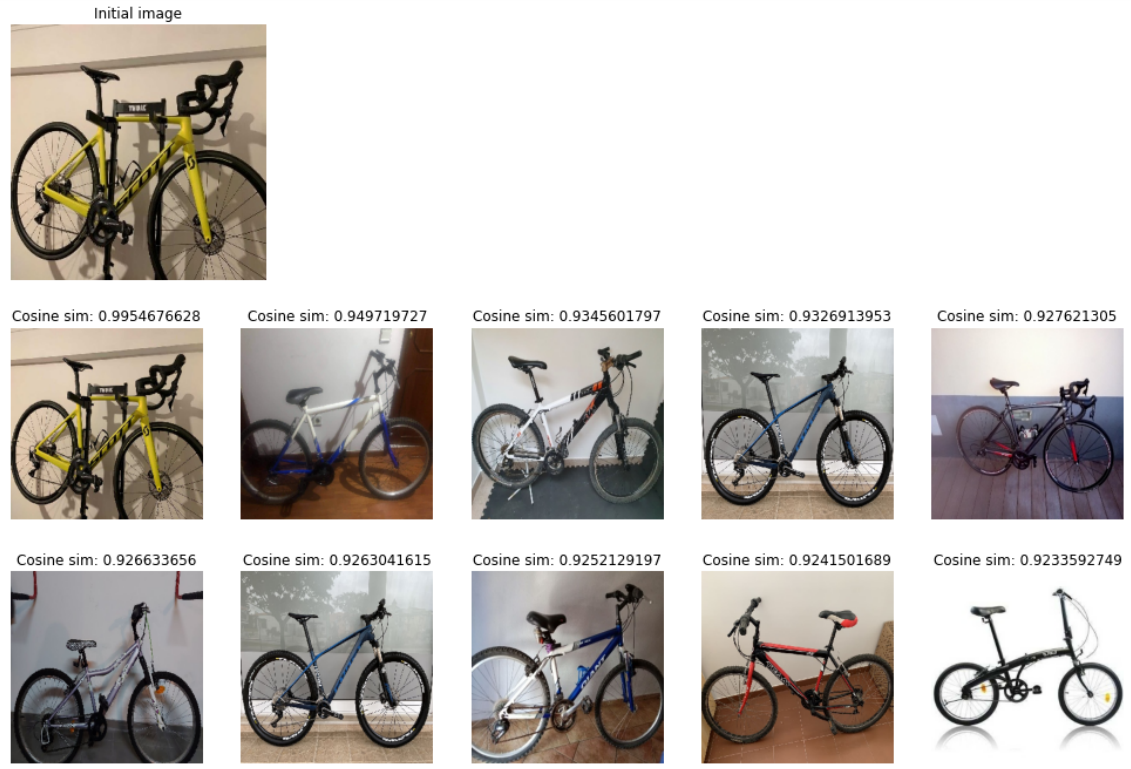


Figure 13: Result of cosine similarity and Average pooling (2, 2)

Figure 14: Result of euclidean distance and Average pooling (2, 2)

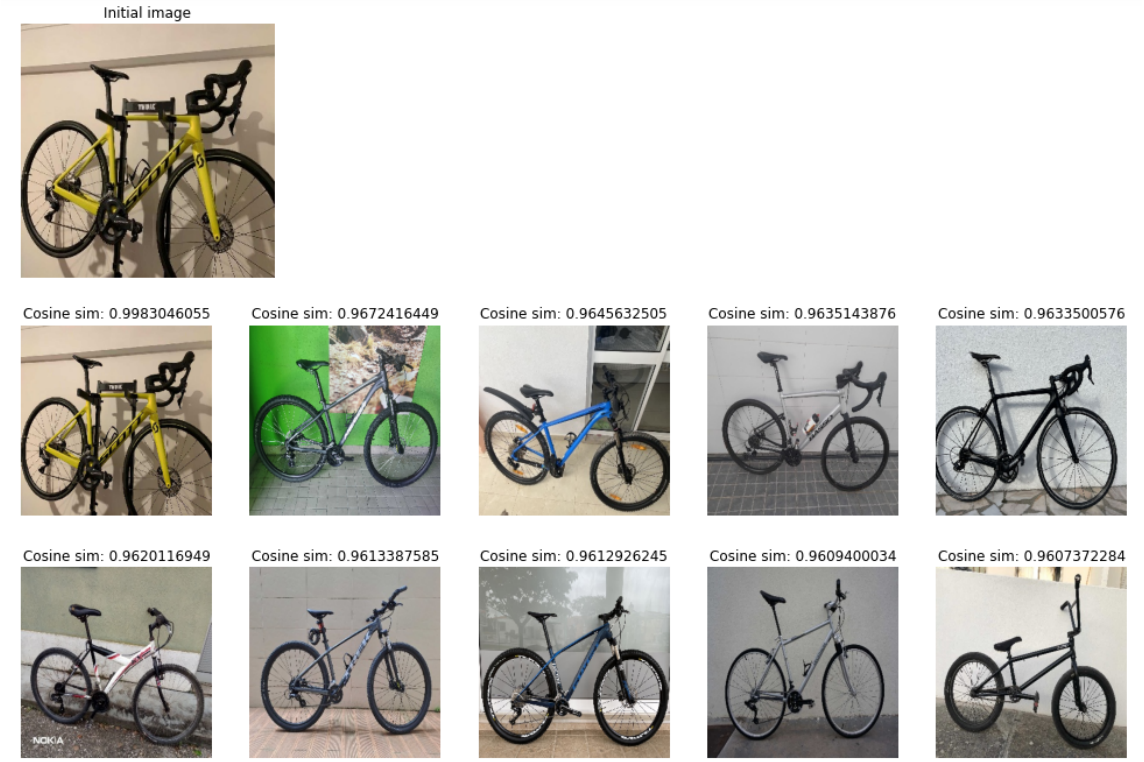


Figure 15: Result of cosine similarity and Average pooling (3, 3)

Figure 16: Result of euclidean distance and Average pooling (3, 3)

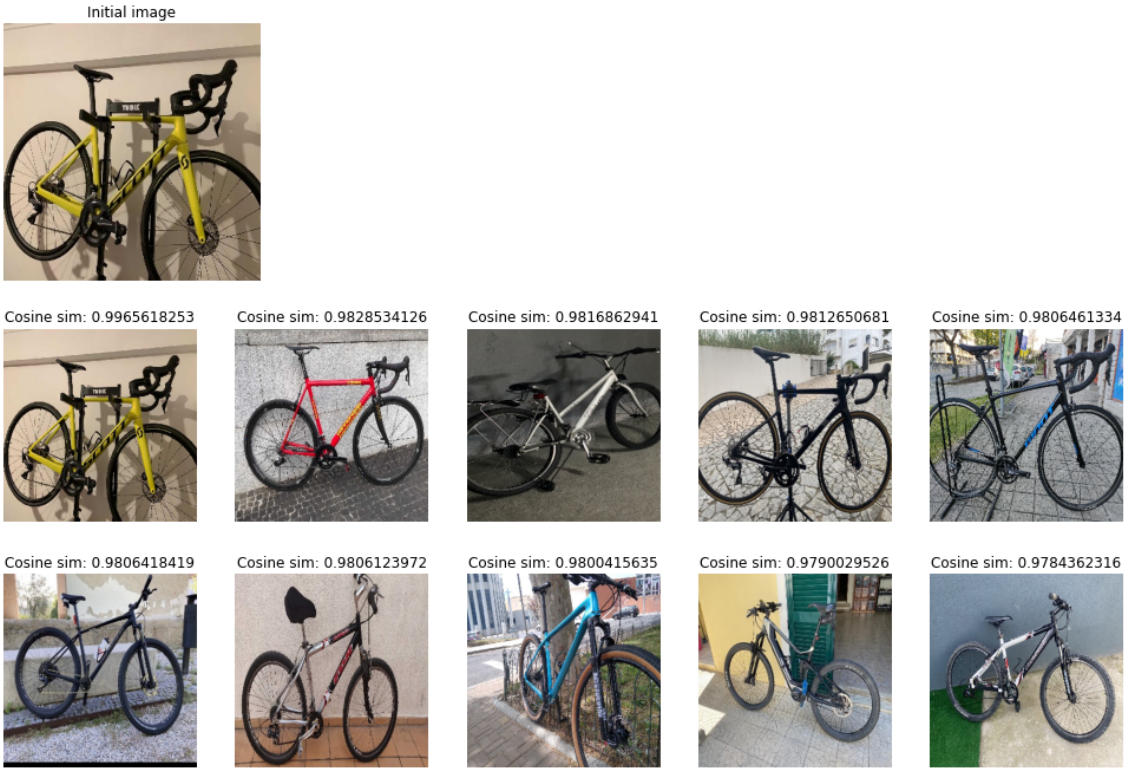


Figure 17: Result of similarity and Global Average pooling

Figure 18: Result of euclidean distance and Global Average pooling

.3 Test on a car

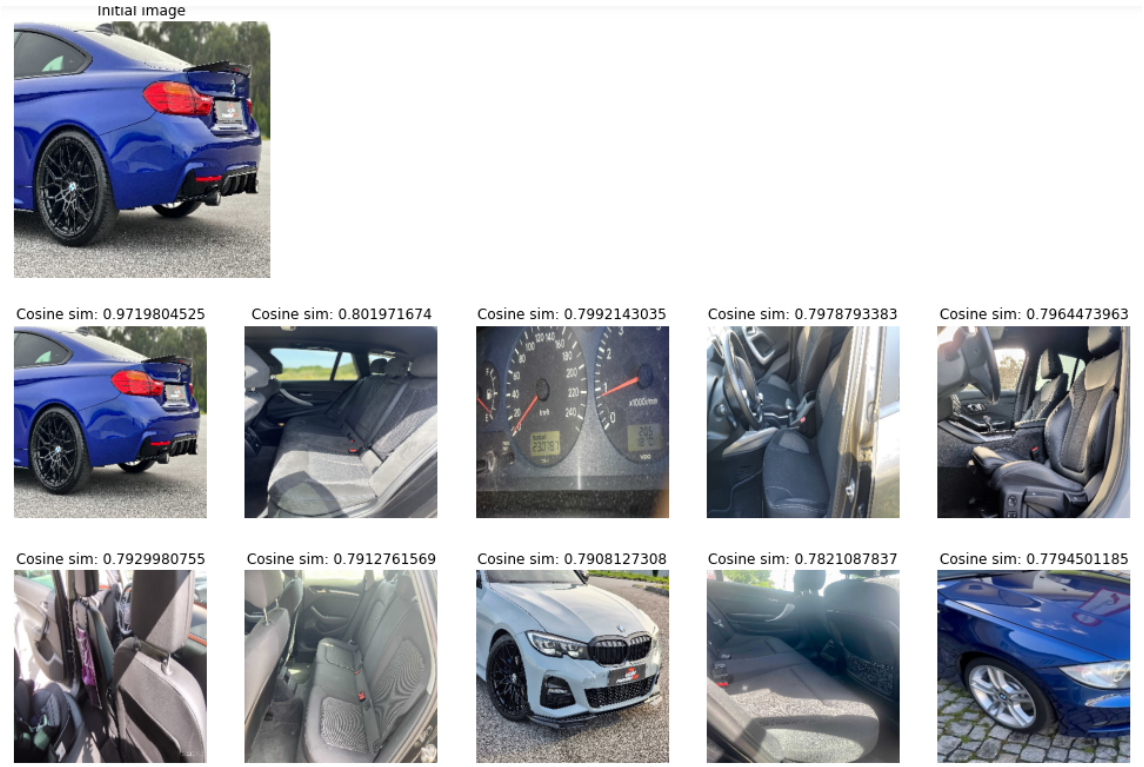


Figure 19: Result of cosine similarity and Average pooling (2, 2)

Figure 20: Result of euclidean distance and Average pooling (2, 2)

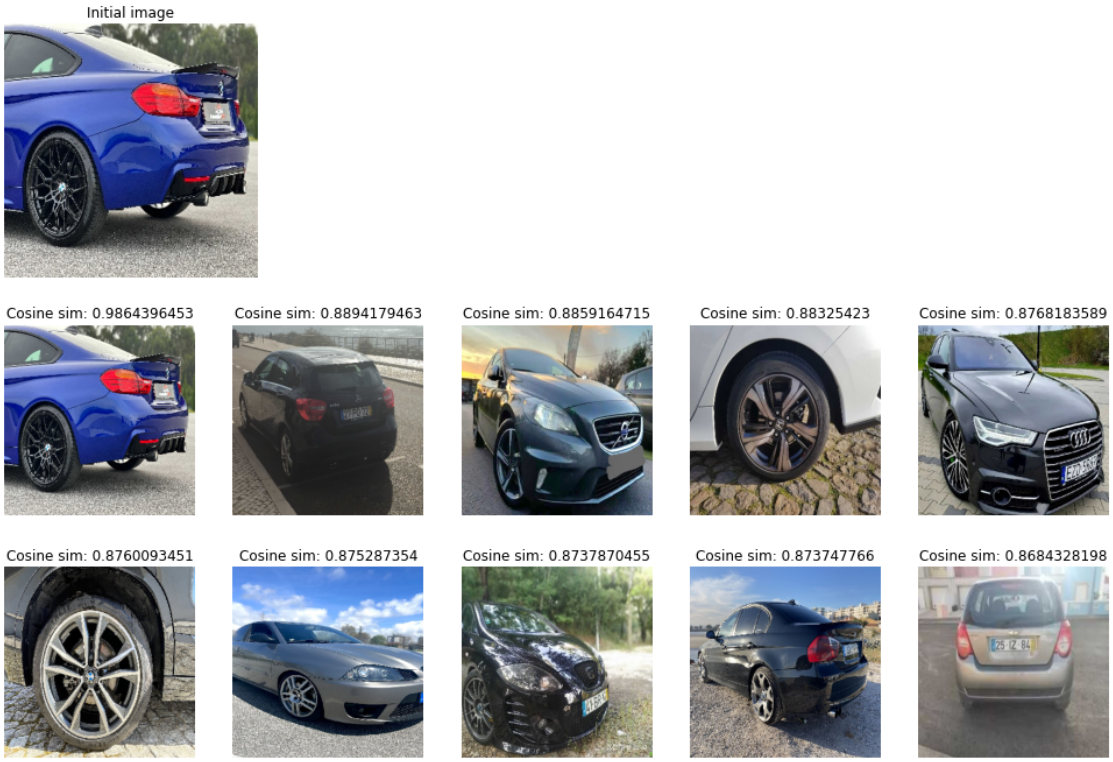


Figure 21: Result of cosine similarity and Average pooling (3, 3)

Figure 22: Result of euclidean distance and Average pooling (3, 3)

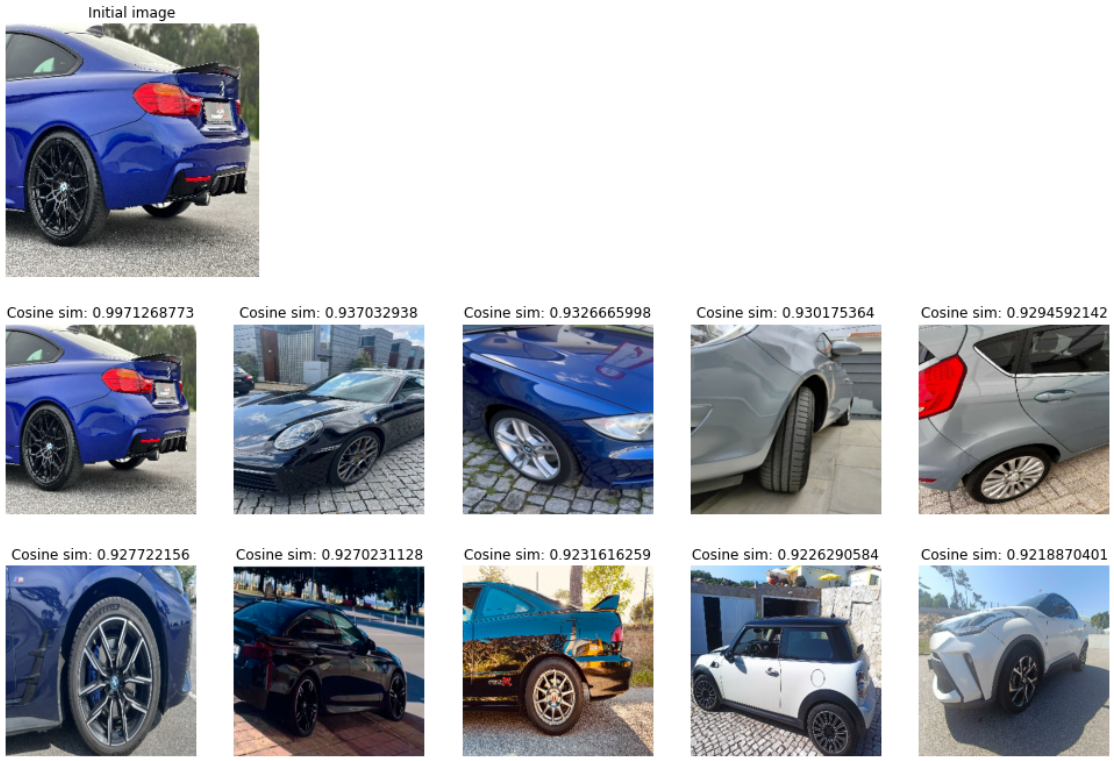


Figure 23: Result of cosine similarity and Global Average pooling

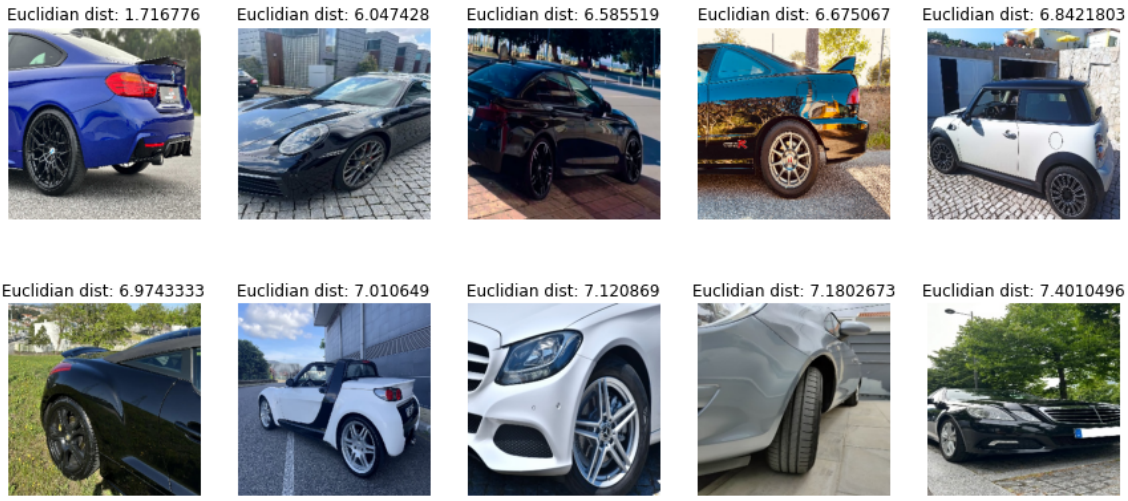
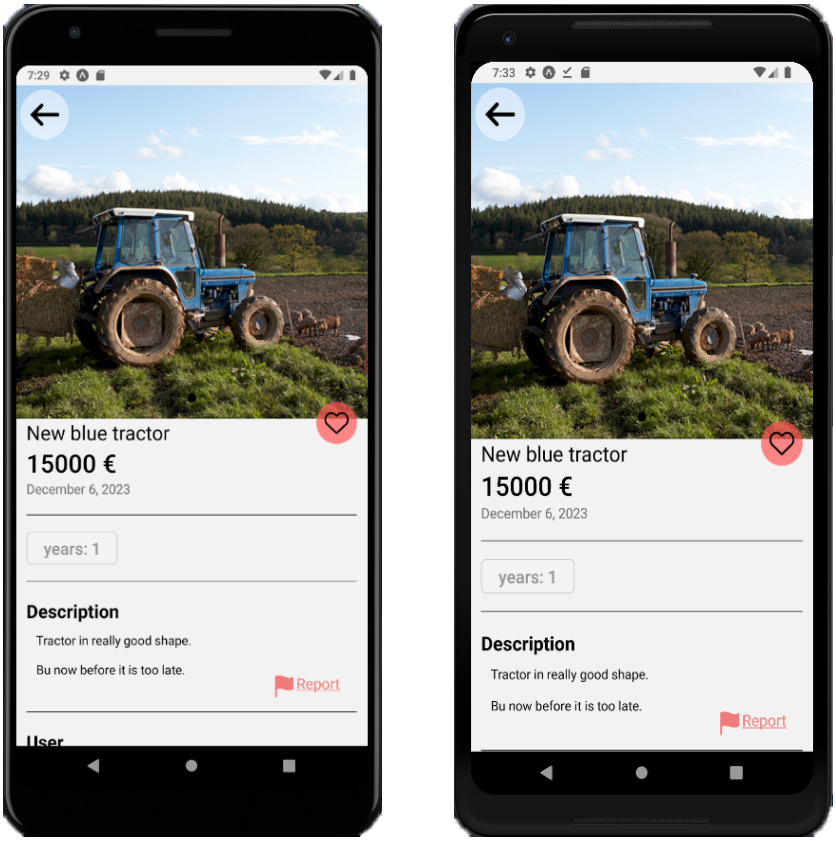


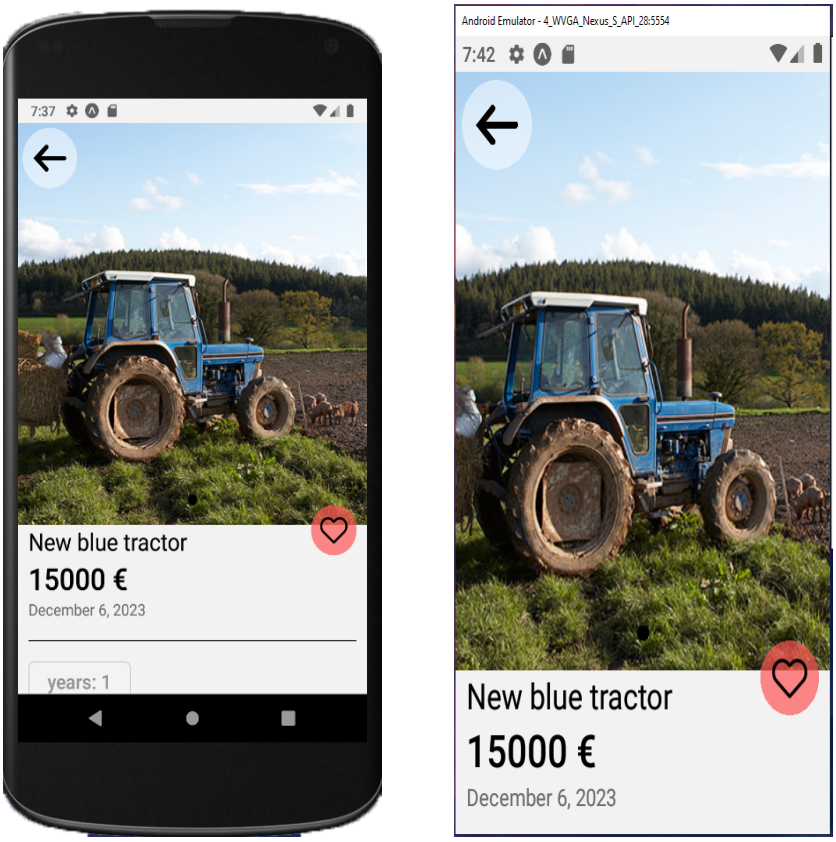
Figure 24: Result of euclidean distance and Global Average pooling

Responsive design



(a) Mobile phone with 1080x2160 (b) Mobile phone with 1440x2880

Figure 25: Images from two different phones



(a) Mobile phone with 768x1280 (b) Mobile phone with 480x800

Figure 26: Images from two different phones smaller

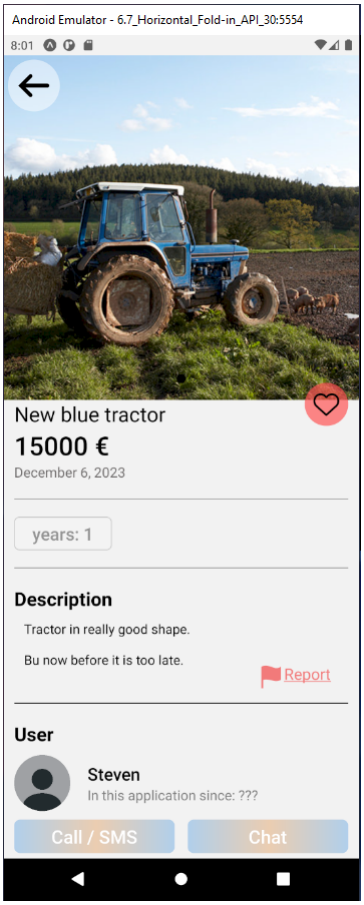


Figure 27: Images for a phone 1080x2636

Examples of images with different classes that are similar

(a) Image from class Agriculture's

(b) Image from class Garden

Figure 28: Images from Agriculture and Garden

(a) Image from class Antiques

(b) Image from class Decoration

Figure 29: Images from Antiques and Decoration

(a) Image from class Appliances

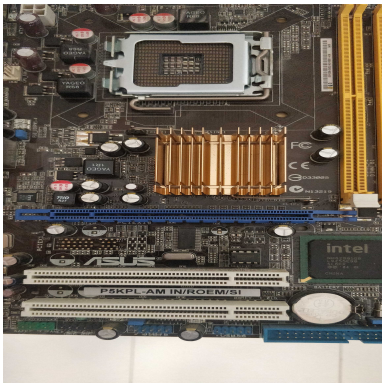
(b) Image from class Furniture

Figure 30: Images from Appliances and Furniture

(a) Image from class Bikes

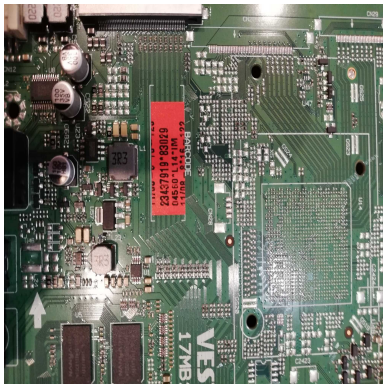
(b) Image from class Sports

Figure 31: Images from Bikes and Sports



(a) Image from class Computer

(b) Image from class Eletronic



(c) Image from class TVs

Figure 32: Images from Computer, Eletronics and TVs

(a) Image from class Cellphone acessories

(b) Image from class Cellphone

Figure 33: Images from Cellphone acessories and Cellphone

(a) Image from class Child

(b) Image from class Clothes

Figure 34: Images from Child and Clothes

Vision document

Vision document

Introduction

In this document, it is intended to specify which functionalities the application will do. The application will be a digital store where the articles to sell are posted by users and the articles will be bought by the users. Additionally, there will be a system that can recognize similar products based on an image or multiple images given by the user.

Characteristics

The characteristics required by the users will be the application to work properly, including being able to see the most relevant products based on previous searches, the search by product based on images given by the user to produce results more or less accurate. The bought products should be sent by the seller if not, the administration offers a refund to the payment.

Problem description

The user wants to buy a product for sale. For that there should be an application that has a user interface easy to use and easy-to-learn. This means that the user of the application should not need a specialist.

The problem	Create an application to buy products
Affects	Clients and potential clients
Solution	Creation of an application where the users can easily search and find the product and buy it

User summary

The users will be people that are searching for a specific product that they want to buy. Other clients can be people that just want to have an idea of the prices practiced in the market. The last type can be people that want to buy cheap products to resell for a higher price.

Execution environment

The application should be able to work personal cellphones in both android and iOS operating system, without any additional resource.

Characteristics summary

- The user can create an account on the application.
- The user can do login with the google account or the created account.
- The user can logout of the current account logged on the application.
- The user can see the products that are being sold on the app.
- The server stores the most recent products viewed.
- The app provides the user with recommended products based on the searches done on the user side.
- The user has an option where he can buy the product from the seller, and the seller send the product once it is paid.
- The user can search products based on the category.
- The user can search for a product writing the name of the product.
- The user can select filters for the products shown on the search based on the written name.
- The user can see the details of the product such as: labels related to the product, all the images, description, contact the seller, see the location and buy the product.
- The user can add a product to his favorites, which also be used to recommend products.
- The user can add a product to sell with his images, price, location, description, category and labels. The user can also add labels such as: how many years does the product have, registration if it's a car and anything that the user wants.
- The user can see the products that clicked as favorites.
- The user can search for similar products based on images stored on the device. The result is given after a short period of time or a long time depending on the methodology chosen by the user.
- The user can edit the product posted.
- The user can see the purchases done by him.
- The user can confirm that the item was received with success and give score based on the received item or service.
- The user can see the cellphone number of the seller or send a private message.

- The user can edit the basic information of the user, such as: image, name, cellphone, email.
- The user can see all his products posted to sell.
- The user can send messages to the seller, where he can discuss a meeting place or the condition to sell the product.
- The filters are:
    - Location -> Separated by district
    - Date of post -> More recent or less recent
    - Recommendation -> Using data mining or neural network
    - Price -> Range between the cheaper and more expensive
    - Color -> the user can select more than one color at once, the colors will be:
        - Beige
        - Black
        - Blue
        - Brown
        - Colorless
        - Gold
        - Gray
        - Green
        - Orange
        - Pink
        - Purple
        - Rainbow
        - Red
        - White
        - Yellow
        - Not important
    - Item condition ->
        - New (never been used)
        - As good as new (In perfect condition)
        - Good condition (quite used preserved),
        - Fair condition (with evidents signs of use),
        - Has given it all (may have to be repaired)
    - Brand -> it will depend on the category.

Restrictions

There can't be selling illegal products, substances or services on the application. The users that do that will be blocked.

The search is sorted by relevance. The relevance is: 1. Premium users appears first, 2.