universidade de aveiro
deti departamento de eletrónica, telecomunicações e informática

# Complementary Report

Tiago Fonseca - 107266

João Gaspar – 114514

Universidade de Aveiro

Base de Dados

# Index

# Introduction

DETI Store is an online merchandising shop that implements a database, interactable through the front-end.

There will be one main entity, the *Users*:

- After a *User* creates an account, to protect the account in case he loses access to the account or to the **2FA**[1], he can recover the account using one of the **N** associated *emergency_codes* that will verify that the *User* is indeed the owner of the account
- *Users* will have one *Cart* that is *User* specific.
- Each *Cart* contains **N** *Products,* and each *product* can be in *Carts* from **M** different *Users*.
- After a *User* buys *products* from a *cart*, an *Order* will be placed. Each *Order* is associated to a *User* and **N** products.
- *Users* will also be able to make one *review* on one *product*, but they can have **N** *reviews* on **N** *products*.

[1] **2FA** – Two Factor Authentication

# Requirements

## Functional Requirements

***Users***

- Create an account with a persistent cart
- Add and remove products to cart
- Complete orders
- Edit account information
- See previous orders
- Review bought products
- Filter and search for products

***Administrators***

- Add and remove products to the shop
- Edit products data
- See current selling products on shop
- Process and/or delete orders
- See shop statistics

## Non-Functional Requirements

**Security**
Provide **2FA**[1] and *Emergency Codes* for *Users*, securing the authentication process and the respective accounts.

**Performance**
The shop must be quick and have a short response time.

**Reliability**
It must persist during and after issues, maintaining the integrity of critical information.

**Ease-of-use**
It should be intuitive and accessible for everyone to use.

# Technologies Used

To develop this application, we used:

- **Flask** – A micro web framework for Python, ideal for building web applications and APIs. Flask's lightweight nature and modular design make it perfect for scaling applications and integrating with various extensions.
- **HTML** – The standard markup language for creating web pages. It provides the structure and content for our application's user interface.
- **JavaScript** – Essential for web development and interactive user interfaces. JavaScript allows for dynamic content updates, form validations, and enhanced user experiences.
- **CSS** – Used for styling HTML elements. CSS ensures our application is visually appealing and provides a responsive design for various devices.
- **SQL Server** – A proprietary relational database management system from Microsoft. It handles data storage, retrieval, and management efficiently, supporting complex queries and transactions.
- **Version Control** – Experienced in using Git/GitHub for software tracking and collaborative development. Git ensures we maintain a history of changes, facilitates team collaboration, and allows for effective version management and branching strategies.

# Software Architecture

The software is divided into *Client*, *Server* and *Database*. The Client component, through a *Browser*, presents an *HTML* page with various features. These features are managed by the server, that makes use of a *SQL Database*.

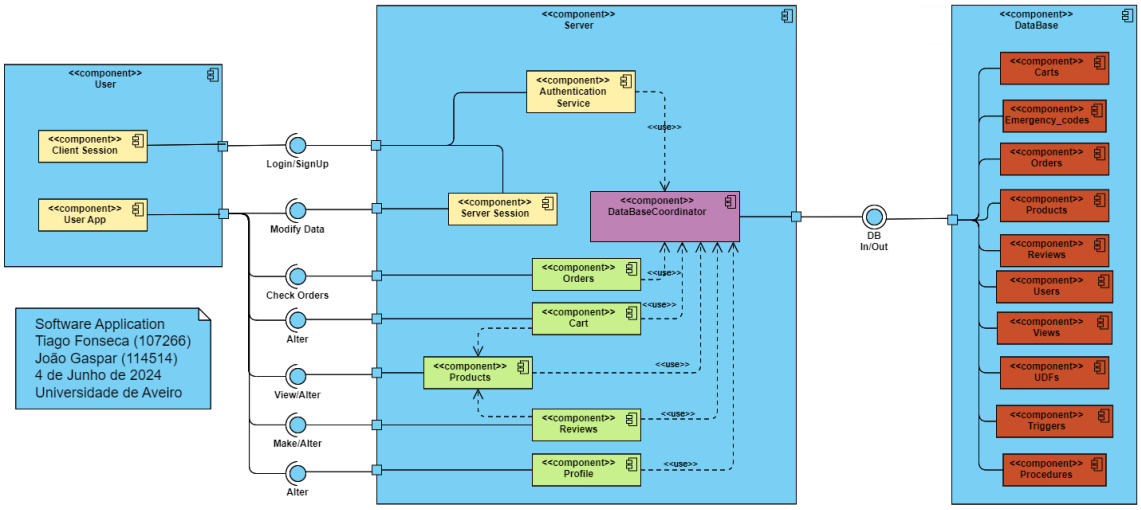


*Figure 1. Software Architecture Component Diagram*

The articulation between the components occurs in the following way:

The session starts when the user *Logs In* into the platform, thereby being authenticated by the server, verifying in the *Database* if his data is presents as a user and if it is correct.
Following the authentication, through the *User Application* (served using a *Browser*), the user can *"Check Orders"*, *"Alter Cart"*, *"View/Alter Products"* (depending on the ***admin_role***[2]), *"Make/Alter Reviews"* and *"Alter Profile"*.

[2] **admin_role –** Attribute of the *Users* Entity

# Physical Installation Architecture

- The components contained in the *Server* are installed on the *server* that will host the service.
- *Users* access the platform through a *web* page, represented by the *Frontend Client* component, which interacts with the *Views* component to redirect the user to the desired page.
- To serve the desired *web* pages, the *Views* component uses the following components:
  - *Static* – Stores static content to the *Front-end.*
    - *JS* – Stores the *JavaScript* scripts.
    - *CSS* – Stores the *CSS* that is used in the *Templates.*
    - *Images* – Stores the images used in the *Templates.*
  - *Templates* –Stores the HTML pages.
  - *Catalog* – Stores the *DETI Store's Catalog* images.
  - *Handlers* – Set of python scripts that handles complex data manipulation tasks.
- The *DataBaseCoordinator* is part of the *Handlers* component, and it makes use of the *Queries* component to, on server startup, check if the components in the *Database* exist.
- The *Database* is installed on the *IETTA* server that will host the service, accessed only by the *DataBaseCoordinator* as shown here.

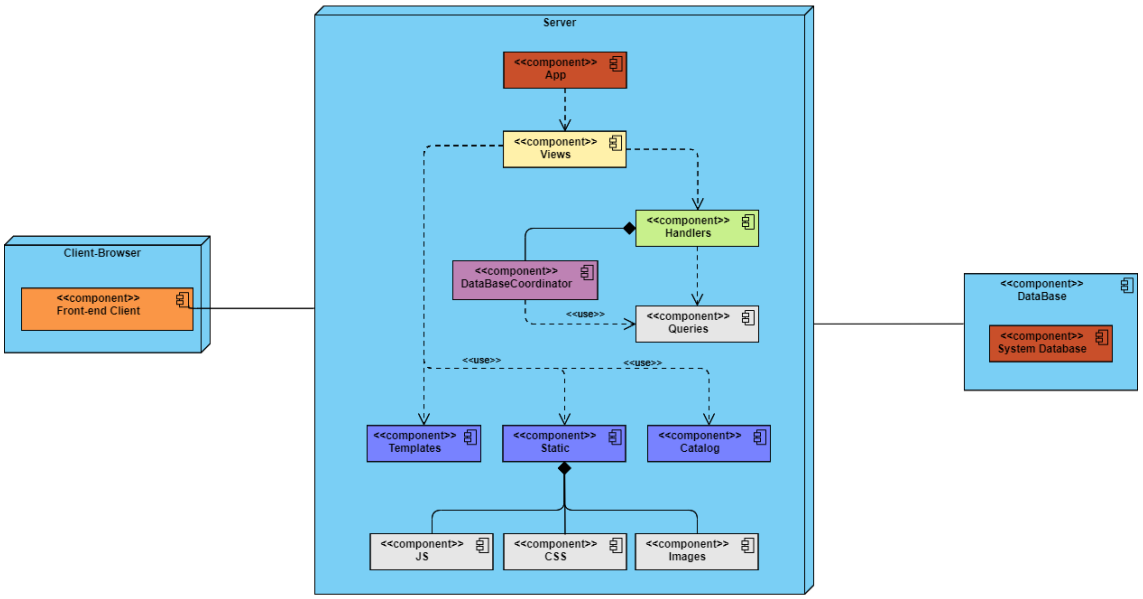


*Figure 2. Physical Installation Architecture Deployment Diagram*

# Important Information

## Using your Database

To use your database, follow these steps:

1. Create a new *database* in your *SQL Server*.
2. Open the `DataBaseCredentials.json` file (present in the `Second Assignment/src/credentials/` directory):
    a. Change the `host` variable to your *database URI*.
    b. Change the `dbname` variable to your *database* name.
    c. Change the `user` variable to your *database* username.
    d. Change the `password` variable to your *database* password.

## Documentation

There is a README.md in the `Second Assignment/src` directory, this file contains additional information about the project.

## License

This project is licensed under the *MIT License*, the license is present in the `Second Assignment/src` directory.