ESCOLA SUPERIOR DE TECNOLOGIA

Computer Systems Engineering

Project Report

GreenSwap

Authors:

> Eduardo Rebelo (21105)
> Gonçalo Gonçalves (26019)
> João Oliveira (26001)
> Pedro Rei (26013)
> Rui Rodrigues (26016)

Software Development Project **May 2024**

Index

Introduction

In the current scenario, the demand for organic and sustainable products is on the rise, driven by the growing interest of consumers in adopting healthier and environmentally responsible lifestyles. In this context, VerDev emerges as an innovative company aiming to transform the way organic products are marketed, connecting independent producers with consumers through the new application, GreenSwap.

GreenSwap is a digital platform that revolutionizes the trade of organic products, offering an intuitive and accessible interface for both independent producers and consumers to interact directly and efficiently. Through this application, producers have the opportunity to sell their products, highlighting their quality and sustainable origin, while consumers can explore a wide variety of options and make their purchases conveniently and securely.

This report aims to analyze in detail the GreenSwap application and put into practice the knowledge acquired throughout the Bachelor's degree in Computer Systems Engineering, within the context of everyday life.

I. Project Vision

Problem description:

Currently, the decline of the agricultural sector and excessive outsourcing in the commercialization of natural products have created recurring difficulties for independent producers. That's why our main goal is to facilitate the buying and selling of these products without going through unwanted intermediaries, making the relationship between the customer and the seller secure and reliable. Additionally, we aim to encourage people to gain new knowledge about this topic.

Objectives and benefits:

- Enhanced ease of buying and selling organic products;
- Knowledge about various topics related to this application;
- A large user base with expertise in various subjects;
- A support center where you can easily address your inquiries.

Stakeholders:

- VerDev administrators;
- GreenSwap administrators;
- Development team;
- Product Owner;
- Software users;
- Investors.

Actors:

- **Client >**It represents the users of the application/website who want to purchase organic products directly from producers/sellers. To do so, creating an account on this software is necessary.

- **Seller >** It represents the users of the application/website who sell their organic products to other users. To do so, creating an account on this software and adding one or more products to their digital store is necessary.

- **Certifier ->** It represents the entity that is responsible for the validation, creation and sending seller certificates for the users that want to be a seller in this system.

I. Business Processes

First Business Model (Immediate Purchase):

The customer using the GreenSwap application can browse all available products for purchase through a search tab that allows searching by name or other attributes. Upon finding a product of interest, the customer will have the option to either buy a desired quantity of the product for the advertised price for future delivery or submit a proposal to the seller with the desired price (up to a maximum of 40% below the original price). If a direct purchase is made, the customer proceeds with the payment immediately. If the customer submits a proposal, the seller may either accept or reject it. If the proposal is accepted, the customer can proceed with the payment; otherwise, they can make a new proposal or cancel the purchase.

Once the purchase is confirmed, the seller receives notification of the sale and prepares the product for shipment. When the customer receives the product, they confirm its receipt, and then the seller receives payment.

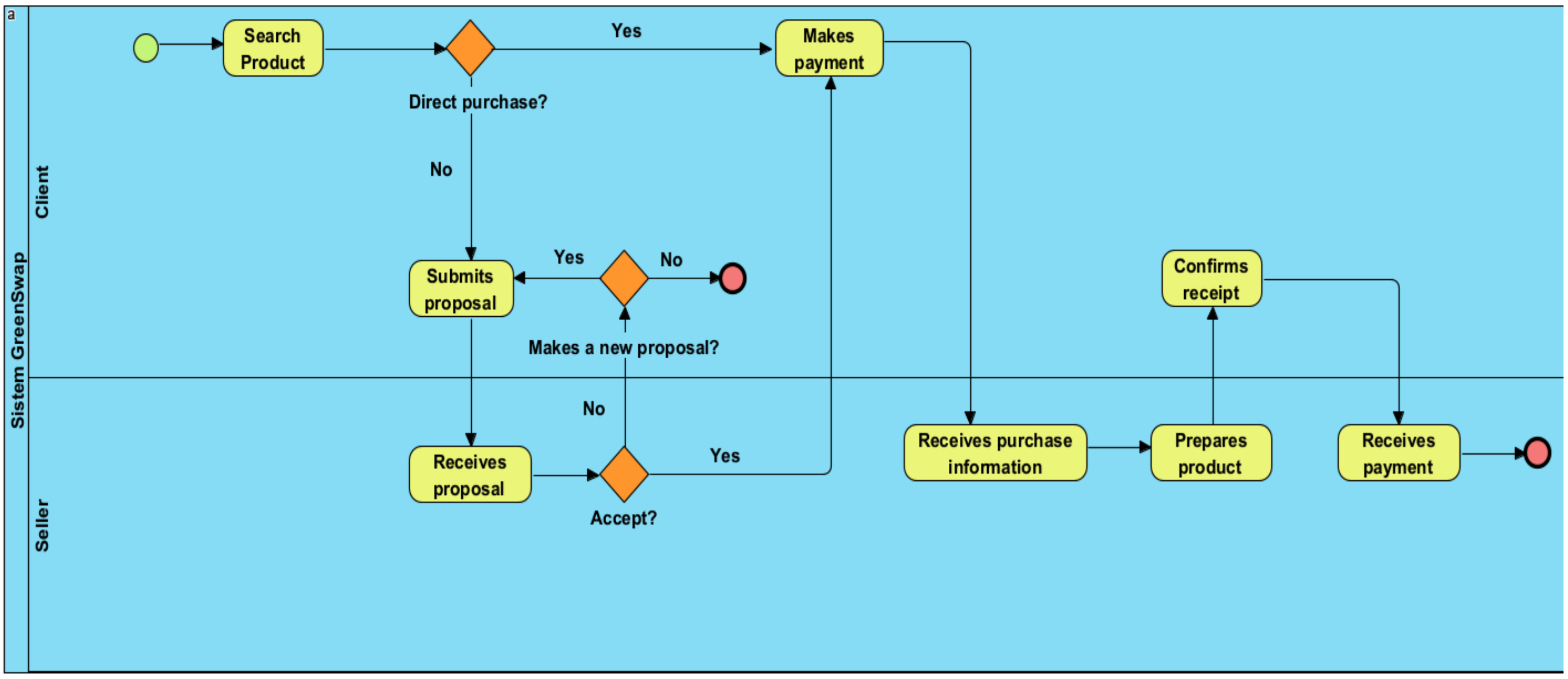


Fig.1 - BPMN diagram of the purchase of a product

Second Business Model (Future purchase of a product):

The customer using the GreenSwap application can browse all available products for purchase through a search feature that allows searching by name or other attributes.

After finding a product of interest, the customer will have the option to either buy a desired quantity of the product for future delivery at the advertised price or submit a proposal to the seller with the desired price (up to a maximum of 40% below the original price). If a proposal is submitted, it can be accepted or rejected by the seller, and the customer is given the opportunity to make a new proposal or cancel the purchase. In the direct purchase option, the customer immediately defines the desired quantity, and the purchase is executed without price negotiation. The seller then receives the purchase details and verifies if it's feasible to proceed. If not, the purchase is canceled; if yes, the purchase proceeds, and the products are reserved, with confirmation to the customer.

Once informed that their purchase is confirmed, the customer proceeds with the payment. The seller prepares the order for the scheduled date, and upon arrival, the seller ships the product. Upon receiving the product, the customer confirms receipt in the system, and the seller receives payment, finalizing the process.

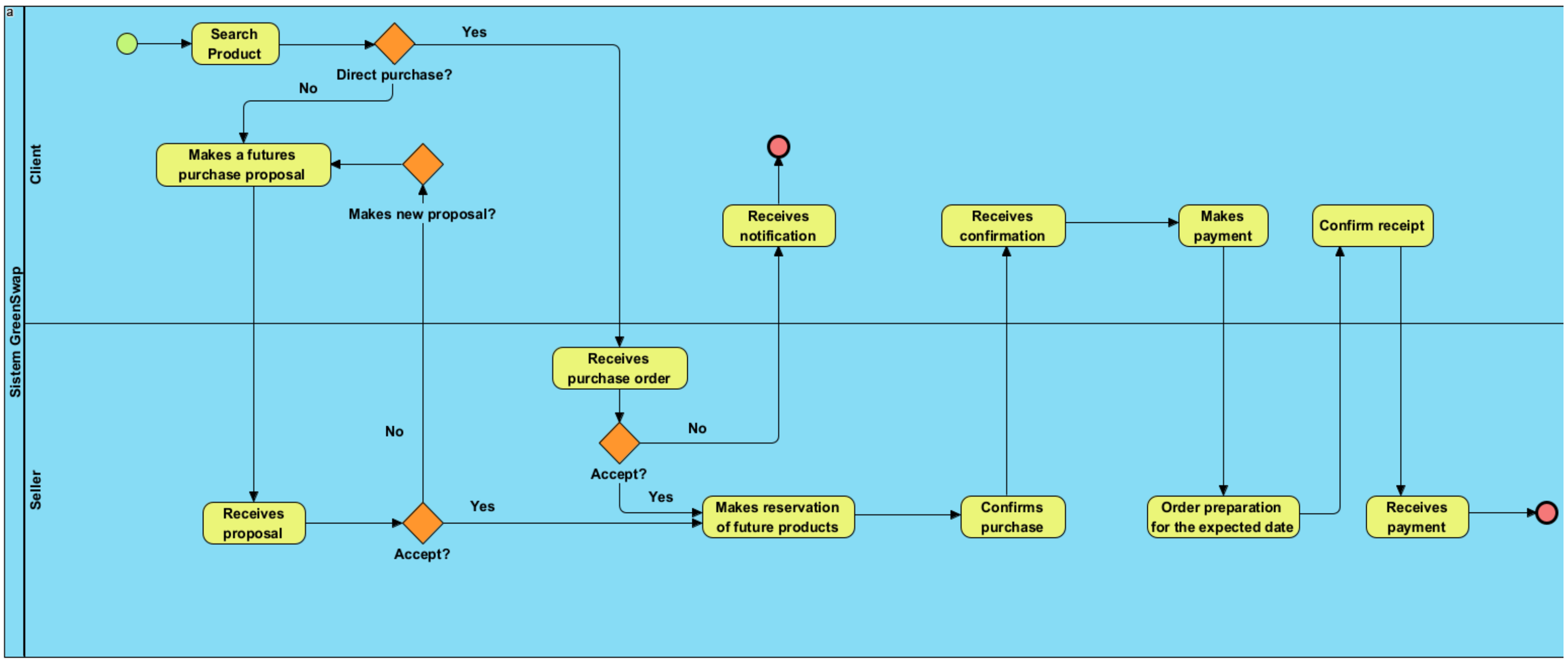


Fig.2 - BPMN diagram of the future purchase of a product

Third Business Model (Periodic Purchase):

The customer, while using the GreenSwap application, can search for all available products for purchase through a search tab that allows searching by name or other attributes.

After finding a product of interest, the customer will have the option to buy a desired quantity of the product for the advertised price in the listing or submit a proposal to the seller with the desired amount (up to a maximum of 40% below the original price). If a proposal is made, it can be accepted or rejected by the seller, and the customer is given the opportunity to submit a new proposal or cancel the purchase. With the direct purchase option, the customer immediately defines the quantity desired, and the purchase is executed without price negotiation. Next, the seller receives the purchase details and verifies if it will be possible to fulfill it; if not, the purchase is canceled, otherwise, the purchase proceeds, the products are reserved, and confirmation is sent to the customer.

After being informed that their purchase is confirmed, the customer makes the payment accordingly. The seller prepares the order for the scheduled date, and when that date arrives, the seller ships the product and the customer receives it. Upon receiving the product, the customer confirms receipt in the system, and the seller receives the payment. If the periodic purchase has ended, the process is finalized; otherwise, the operation returns to the customer's confirmation of receipt and repeats the process.

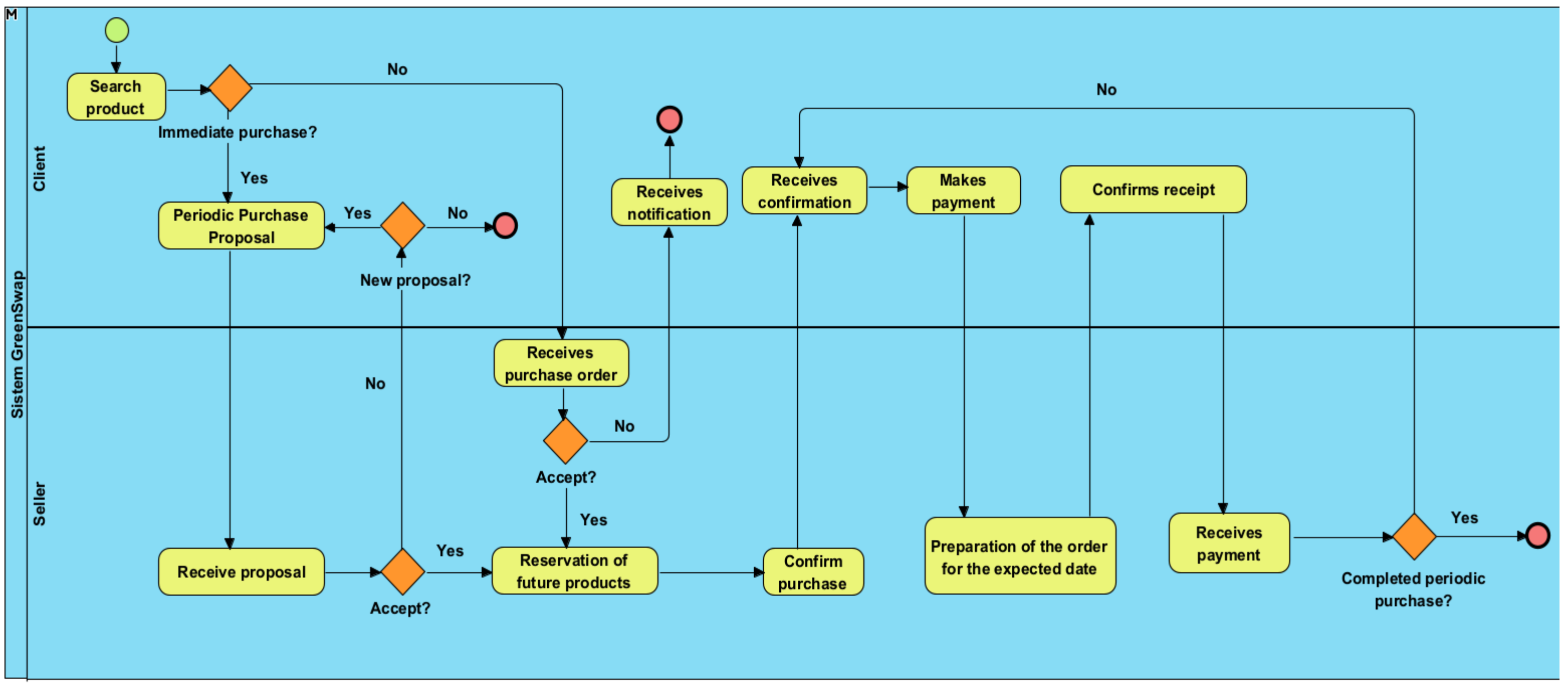


Fig.3 - BPMN diagram of periodic purchasing

<u>Fourth Business Model (Certification):</u>

The user who wants to become a seller needs to submit a request to the responsible entity. The entity receives the request and, after checking whether it is valid to be analyzed, analyzes it and makes the decision to accept it or not. If you do not accept, the seller can submit a new order or withdraw. If accepted, the entity confirms the request and after creating the certificate, sends it to the seller who receives it. From then on, the seller is already certified and can make sales on the application.

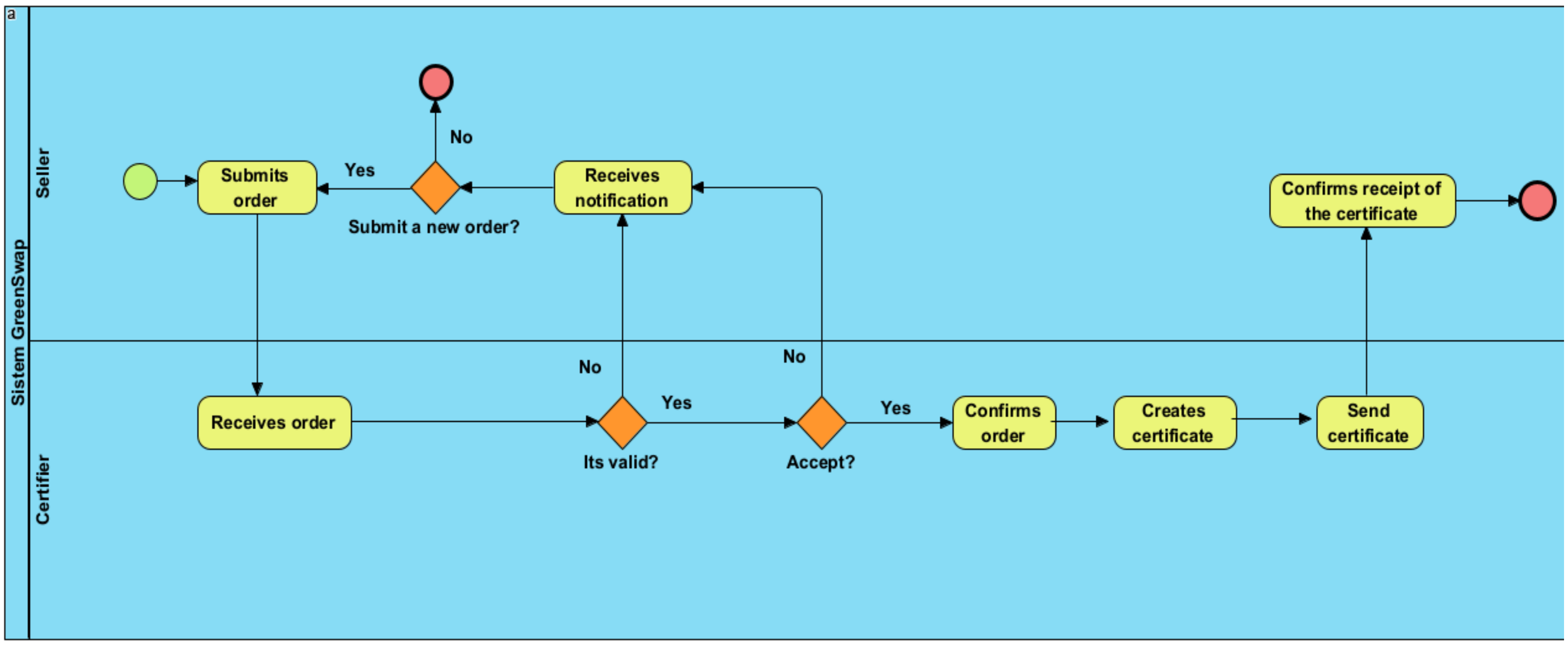


Fig. 4- Certification BPMN Diagram

I. Class Diagram

This class diagram is a representation of the GreenSwap system structure and its functionalities.

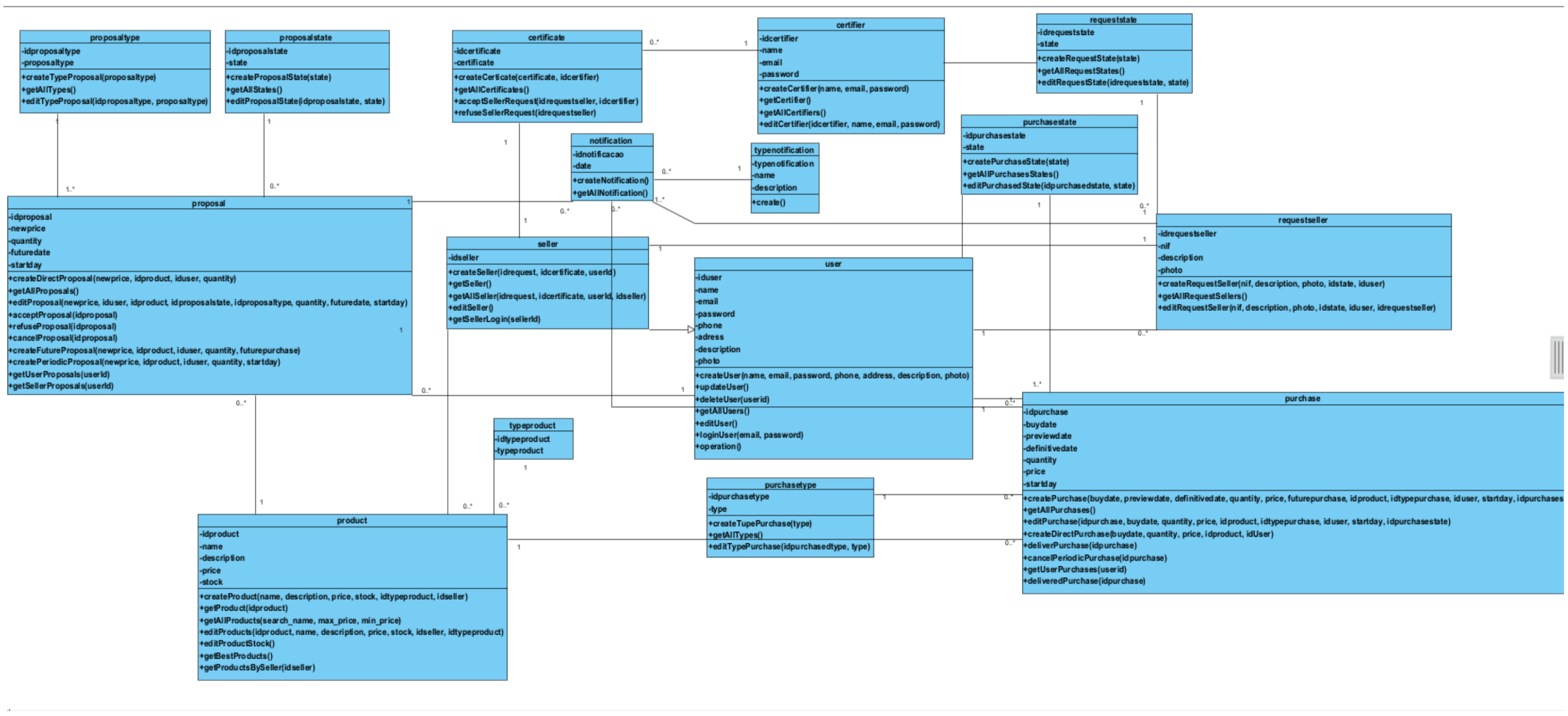

proposaltype
-idproposaltype
-proposaltype
+createTypeProposal(proposaltype)
+getAllTypes()
+editTypeProposal(idproposaltype, proposaltype)
proposalstate
-idproposalstate
-state
+createProposalState(state)
+getAllStates()
+editProposalState(idproposalstate, state)
certificate
-idcertificate
-certificate
+createCerticate(certificate, idcertifier)
+getAllCertificates()
+acceptSellerRequest(idrequestseller, idcertifier)
+refuseSellerRequest(idrequestseller)
certifier
-idcertifier
-name
-email
-password
+createCertifier(name, email, password)
+getCertifier()
+getAllCertifiers()
+editCertifier(idcertifier, name, email, password)
requeststate
-idrequeststate
-state
+createRequestState(state)
+getAllRequestStates()
+editRequestState(idrequeststate, state)
notification
-idnotificacao
-date
+createNotification()
+getAllNotification()
typenotification
-typenotification
-name
-description
+create()
purchasestate
-idpurchasestate
-state
+createPurchaseState(state)
+getAllPurchasesStates()
+editPurchasedState(idpurchasedstate, state)
proposal
-idproposal
-newprice
-quantity
-futuredate
-startday
+createDirectProposal(newprice, idproduct, iduser, quantity)
+getAllProposals()
+editProposal(newprice, iduser, idproduct, idproposalstate, idproposaltype, quantity, futuredate, startday)
+acceptProposal(idproposal)
+refuseProposal(idproposal)
+cancelProposal(idproposal)
+createFutureProposal(newprice, idproduct, iduser, quantity, futurepurchase)
+createPeriodicProposal(newprice, idproduct, iduser, quantity, startday)
+getUserProposals(userId)
+getSellerProposals(userId)
seller
-idseller
+createSeller(idrequest, idcertificate, userId)
+getSeller()
+getAllSeller(idrequest, idcertificate, userId, idseller)
+editSeller()
+getSellerLogin(sellerId)
requestseller
-idrequestseller
-nif
-description
-photo
+createRequestSeller(nif, description, photo, idstate, iduser)
+getAllRequestSellers()
+editRequestSeller(nif, description, photo, idstate, iduser, idrequestseller)
user
-iduser
-name
-email
-password
-phone
-adress
-description
-photo
+createUser(name, email, password, phone, address, description, photo)
+updateUser()
+deleteUser(userid)
+getAllUsers()
+editUser()
+loginUser(email, password)
+operation()
typeproduct
-idtypeproduct
-typeproduct
purchasetype
-idpurchasetype
-type
+createTupePurchase(type)
+getAllTypes()
+editTypePurchase(idpurchasedtype, type)
purchase
-idpurchase
-buydate
-previewdate
-definitivedate
-quantity
-price
-startday
+createPurchase(buydate, previewdate, definitivedate, quantity, price, futurepurchase, idproduct, idtypepurchase, iduser, startday, idpurchases
+getAllPurchases()
+editPurchase(idpurchase, buydate, quantity, price, idproduct, idtypepurchase, iduser, startday, idpurchasestate)
+createDirectPurchase(buydate, quantity, price, idproduct, idUser)
+deliverPurchase(idpurchase)
+cancelPeriodicPurchase(idpurchase)
+getUserPurchases(userid)
+deliveredPurchase(idpurchase)
product
-idproduct
-name
-description
-price
-stock
+createProduct(name, description, price, stock, idtypeproduct, idseller)
+getProduct(idproduct)
+getAllProducts(search_name, max_price, min_price)
+editProducts(idproduct, name, description, price, stock, idseller, idtypeproduct)
+editProductStock()
+getBestProducts()
+getProductsBySeller(idseller)

Fig.5 - Class Diagram

I. Entity Relationship Diagram

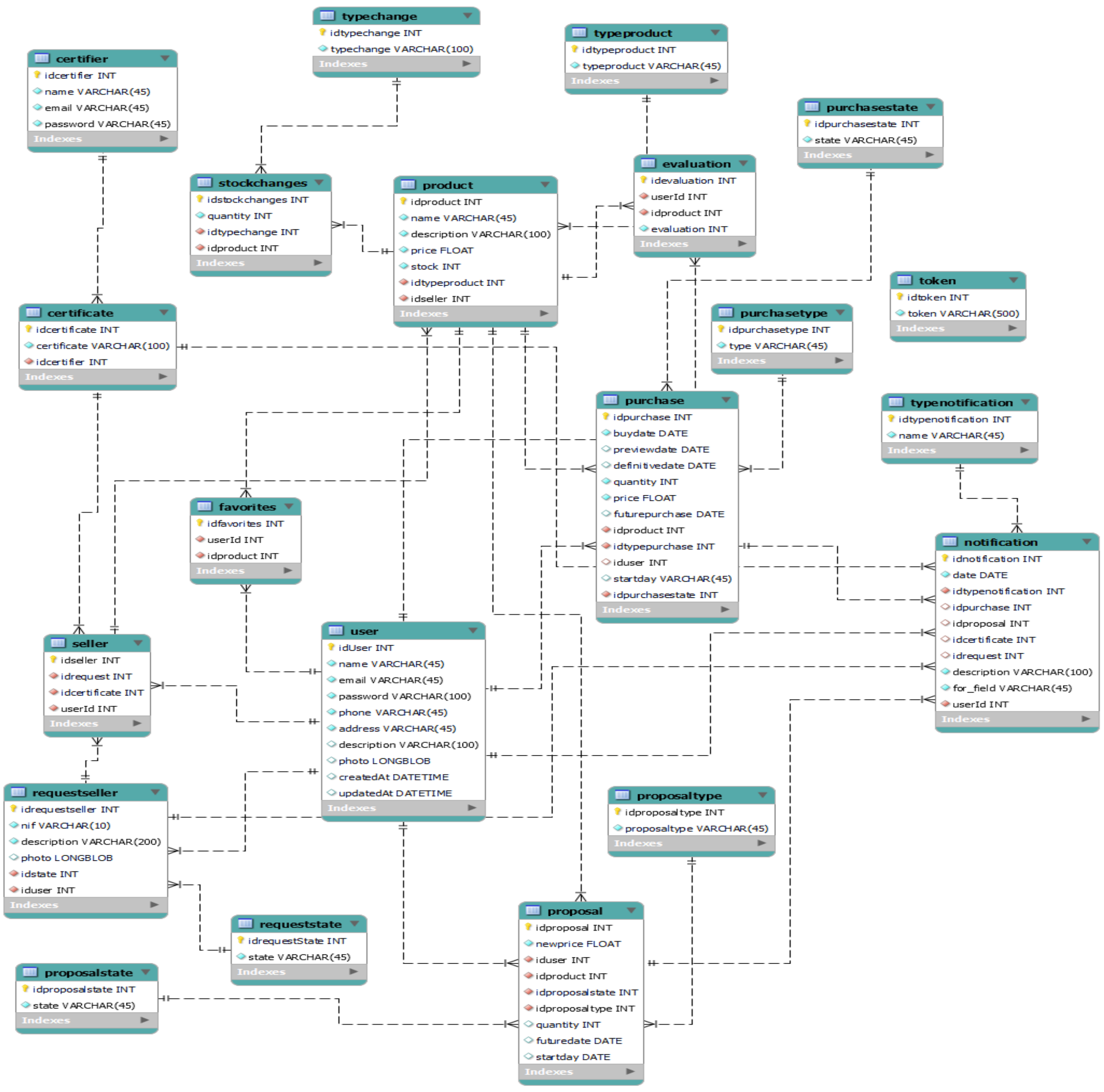


Fig.6 - Application ER diagram

I. Use Case Models

Client:

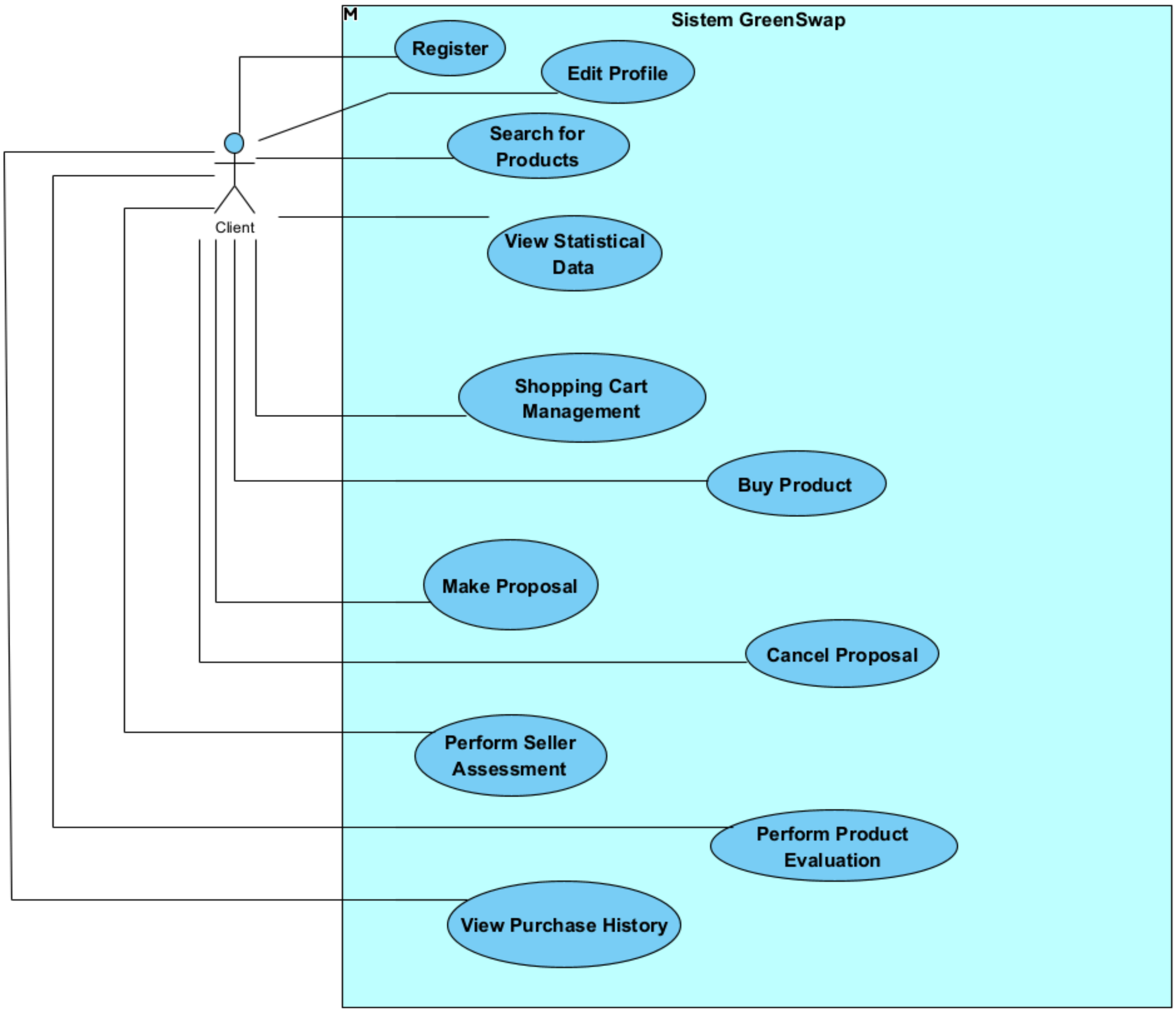


Fig.7 - Customer Use Case Diagram

- **UC01.01:** Register - The customer registers in the system;
- **UC01.02:** Edit Profile - The customer edits their profile (such as name, email, password, etc...);
- **UC01.03:** Search for Products: The customer searches for products and filters the search;

- **UC01.04:** View Statistical Data: The customer can view statistical data (such as number of purchases);
- **UC01.05:** Shopping Cart Management: The customer can manage the shopping cart (add products, remove products, change quantity, etc...);
- **UC01.06:** Buy Product - The customer buys the product;
- **UC01.07:**Make Proposal - The customer makes a proposal per product instead of making an immediate purchase;
- **UC01.08:** Cancel Proposal - The client cancels a proposal he previously made;
- **UC01.09:** Perform Seller Assessment - The customer evaluates the seller after receiving the product from the seller;
- **UC01.10:** Perform Product Evaluation - The customer evaluates the seller after receiving it;
- **UC01.11:** View Purchase History - The customer views the purchase history.

Seller:

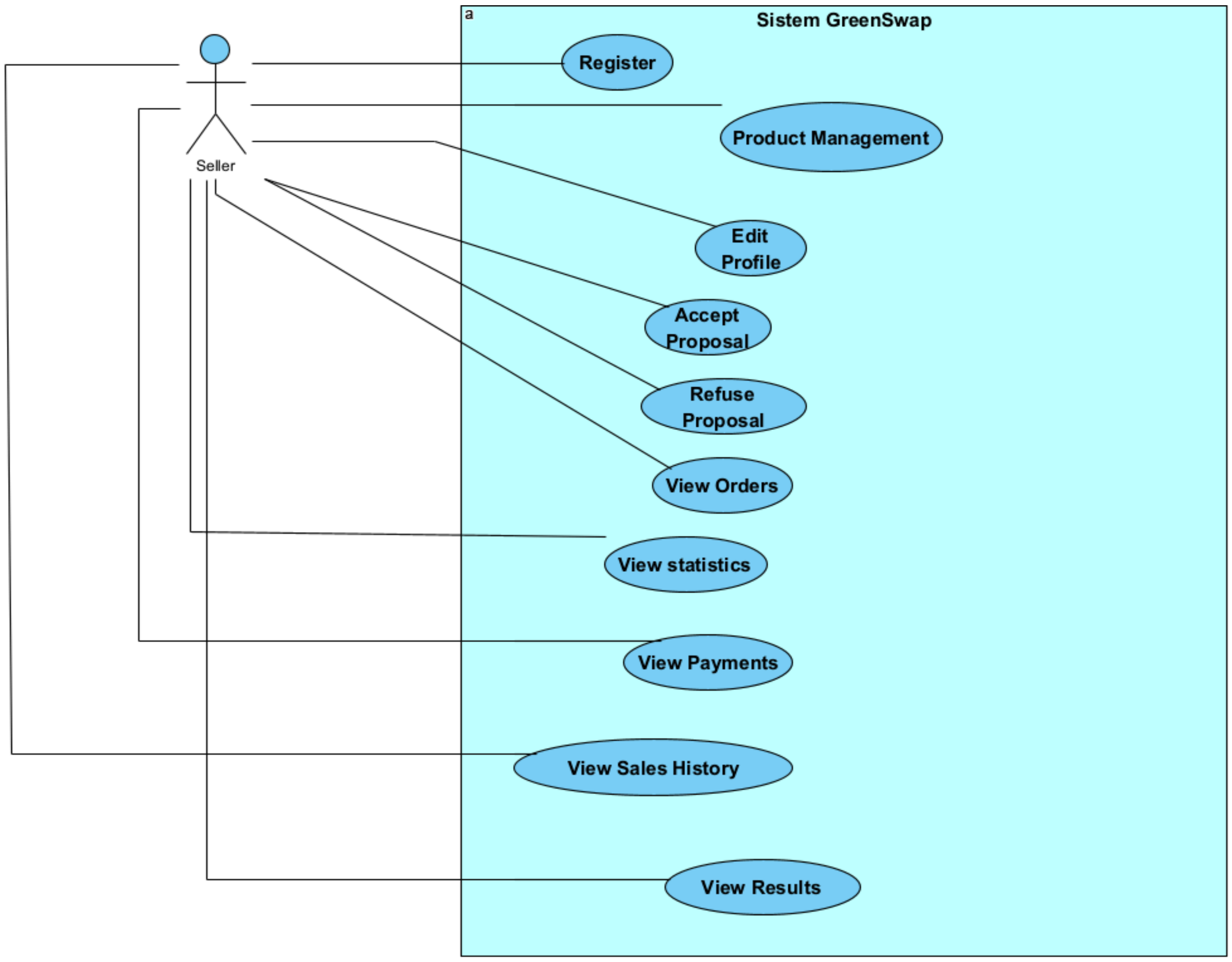


Fig.8 - Seller Use Case Diagram

- **UC02.01:** Register - The seller registers in the system;
- **UC02.02:** Product Management - The seller can manage their products (Add, Remove, Edit, etc...);
- **UC02.03:** Edit Profile - The seller edits their profile (such as name, email, password, etc...);
- **UC02.04:** Accept Proposal - The seller accepts proposals from the customer;
- **UC02.05:** Refuse Proposal - The seller refuses the client's proposals;
- **UC02.06:** View Orders - The seller views orders placed by customers;
- **UC02.07:** View Statistics - The seller views the statistics**;**
- **UC02.08:** View Payments - The seller views customer payments for their products;
- **UC02.09:** View Sales History - The seller views the sales history of the products he has sold;
- **UC02.10:** View Reviews - The seller views the reviews that customers have given about their services and products.

Certifier:

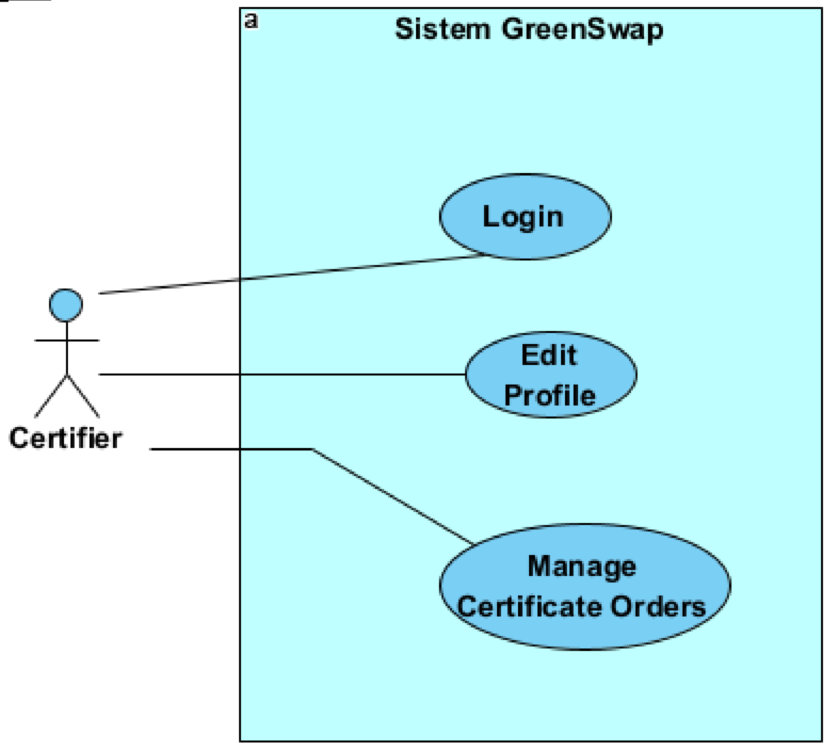
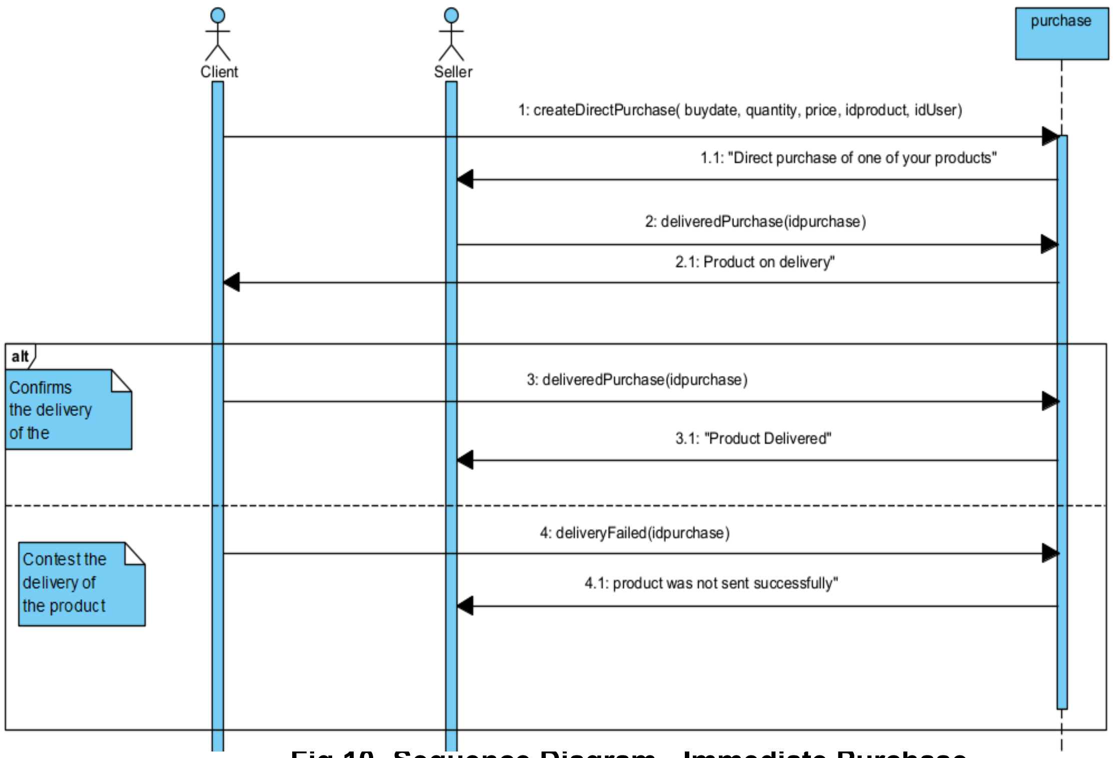


Fig.9 - Certifier Use Case Diagram

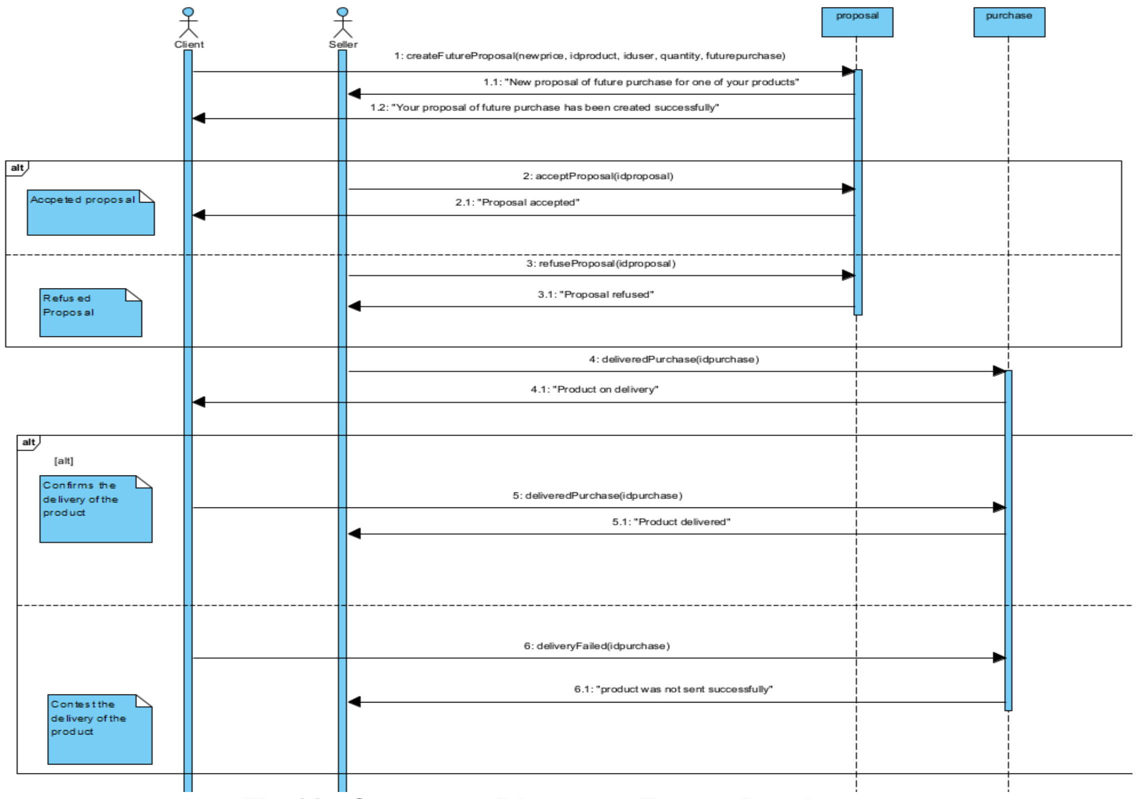
- **UC01.01:** Login - The certifier logs into the system, without the need for registration as he has an account created by the system administrators
- **UC01.02:** Edit Profile - The certifier edits their profile (such as name, email, password, etc...);
- **UC01.03:** Manage Certificate Orders - The certifier can manage certificate order by validate, create and send certificate to users that want to be sellers in the system

I. Sequence Diagrams



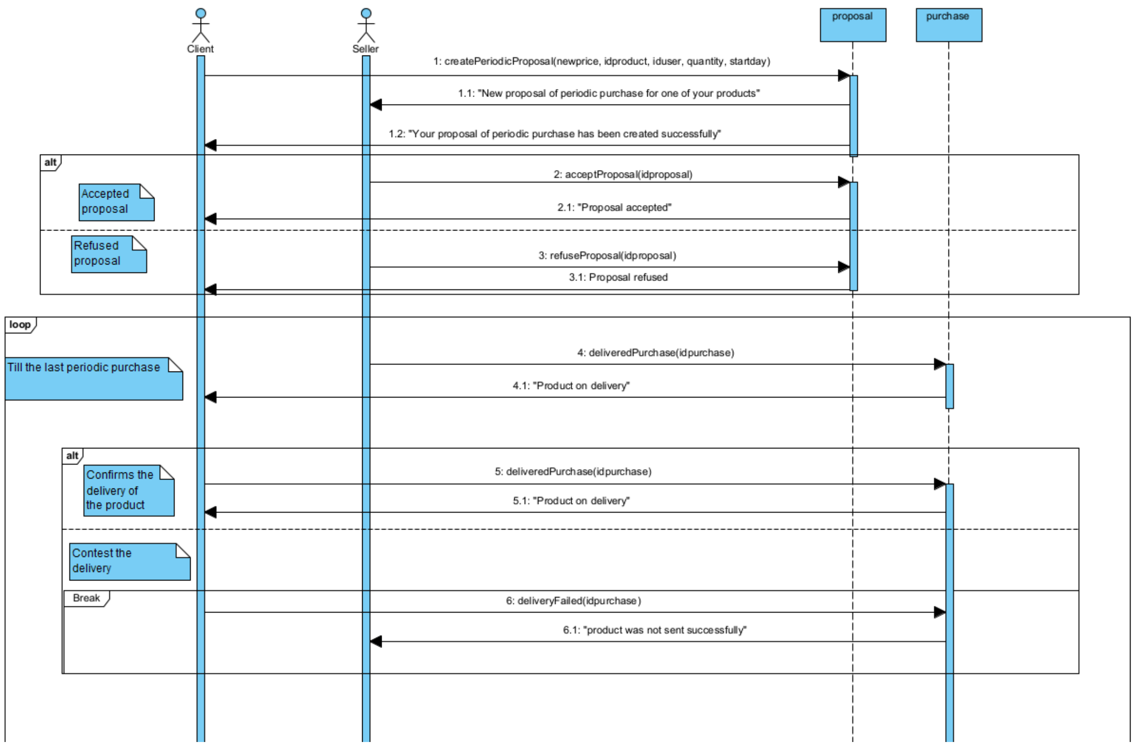
Client
Seller
purchase
1: createDirectPurchase(buydate, quantity, price, idproduct, idUser)
1.1: "Direct purchase of one of your products"
2: deliveredPurchase(idpurchase)
2.1: Product on delivery"
alt
Confirms the delivery of the
3: deliveredPurchase(idpurchase)
3.1: "Product Delivered"
Contest the delivery of the product
4: deliveryFailed(idpurchase)
4.1: product was not sent successfully"

Fig.10- Sequence Diagram - Immediate Purchase



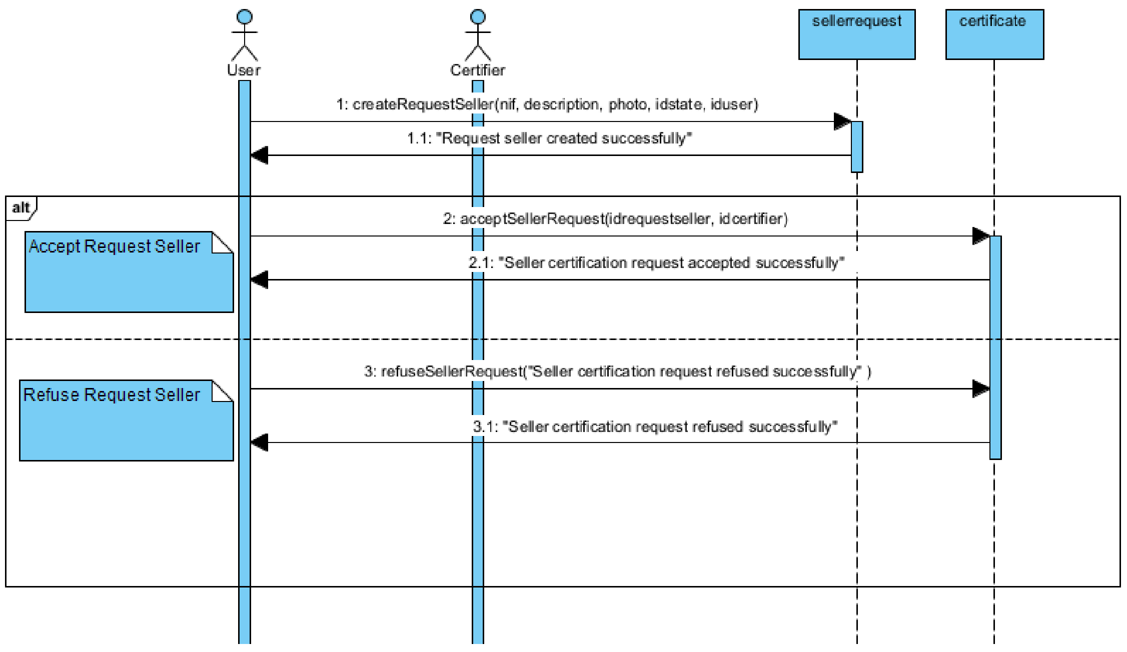
Client
Seller
proposal
purchase
1: createFutureProposal(newprice, idproduct, iduser, quantity, futurepurchase)
1.1: "New proposal of future purchase for one of your products"
1.2: "Your proposal of future purchase has been created successfully"
alt
Accpeted proposal
2: acceptProposal(idproposal)
2.1: "Proposal accepted"
Refused Proposal
3: refuseProposal(idproposal)
3.1: "Proposal refused"
4: deliveredPurchase(idpurchase)
4.1: "Product on delivery"
alt
[alt]
Confirms the delivery of the product
5: deliveredPurchase(idpurchase)
5.1: "Product delivered"
6: deliveryFailed(idpurchase)
6.1: "product was not sent successfully"
Contest the delivery of the product

Fig.11 - Sequence Diagram - Future Purchase



Client
Seller
proposal
purchase
1: createPeriodicProposal(newprice, idproduct, iduser, quantity, startday)
1.1: "New proposal of periodic purchase for one of your products"
1.2: "Your proposal of periodic purchase has been created successfully"
alt
Accepted proposal
2: acceptProposal(idproposal)
2.1: "Proposal accepted"
Refused proposal
3: refuseProposal(idproposal)
3.1: Proposal refused
loop
Till the last periodic purchase
4: deliveredPurchase(idpurchase)
4.1: "Product on delivery"
alt
Confirms the delivery of the product
5: deliveredPurchase(idpurchase)
5.1: "Product on delivery"
Contest the delivery
Break
6: deliveryFailed(idpurchase)
6.1: "product was not sent successfully"

Fig.12 - Sequence Diagram - Periodic Purchase



User
Certifier
sellerrequest
certificate
1: createRequestSeller(nif, description, photo, idstate, iduser)
1.1: "Request seller created successfully"
alt
Accept Request Seller
2: acceptSellerRequest(idrequestseller, idcertifier)
2.1: "Seller certification request accepted successfully"
Refuse Request Seller
3: refuseSellerRequest("Seller certification request refused successfully")
3.1: "Seller certification request refused successfully"

Fig.13 - Sequence Diagram - Certificate

I. Product Backlog

Functional requirements:

- Registration/Login System;
- Manage profile/account:
  - Allow the user to edit profile;
  - Allow deleting your account in the application;
- Notification System;
- Product management:
  - Allow adding a product to sale;
  - Allow writing product characteristics (Product Type, Description);
  - Allow changing product data;
- Purchasing management:
  - Allow buying/selling product;
  - Allow canceling proposals;
  - Allows you to make proposals and counter-proposals for a product;
- Sales management:
  - Allow to sell product;
- Evaluation system:
  - Allow rating a product/seller (1 to 5 stars);
  - Average rating of a product;
- Ranking system:
  - Best product ranking;
  - Best seller ranking;
  - Ranking best-selling products;
- Filtered and advanced search system:
  - The user must be able to browse a list of available agricultural products;
  - Allow you to search for products;
  - Allow search filters on products;
- Support System:
  - Allow user support;
  - Allow complaints books;
  - Ticket system;
- Favorites System;
- Purchase/Sales History;
- Sales history;
- Viewing profile data:
  - Allow viewing of seller/customer statistics;
- Payment visualization;

- Order viewing.

<u>Non-functional requirements:</u>

- The system must be compatible with the Google Chrome and Safari web browsers, with the possibility of extending compatibility to other browsers in the future.
- Strong data security must be implemented to protect customer and professional information;
- The platform must be capable of managing multiple transactions simultaneously efficiently and securely;
- It must be able to handle multiple users simultaneously without significant loss of performance;
- A page should load as quickly as possible;
- It must be responsive and adaptable to different screen sizes such as mobile devices;
- Robust security practices such as SSL encryption;
- User passwords must be stored securely, taking advantage of hashing algorithms;
- Accessibility: The platform is accessible to any type of user, regardless of age, gender, ethnicity, orientation, disability, etc... The aim is to include everyone and make the platform easy to navigate and use.

I. Mockups

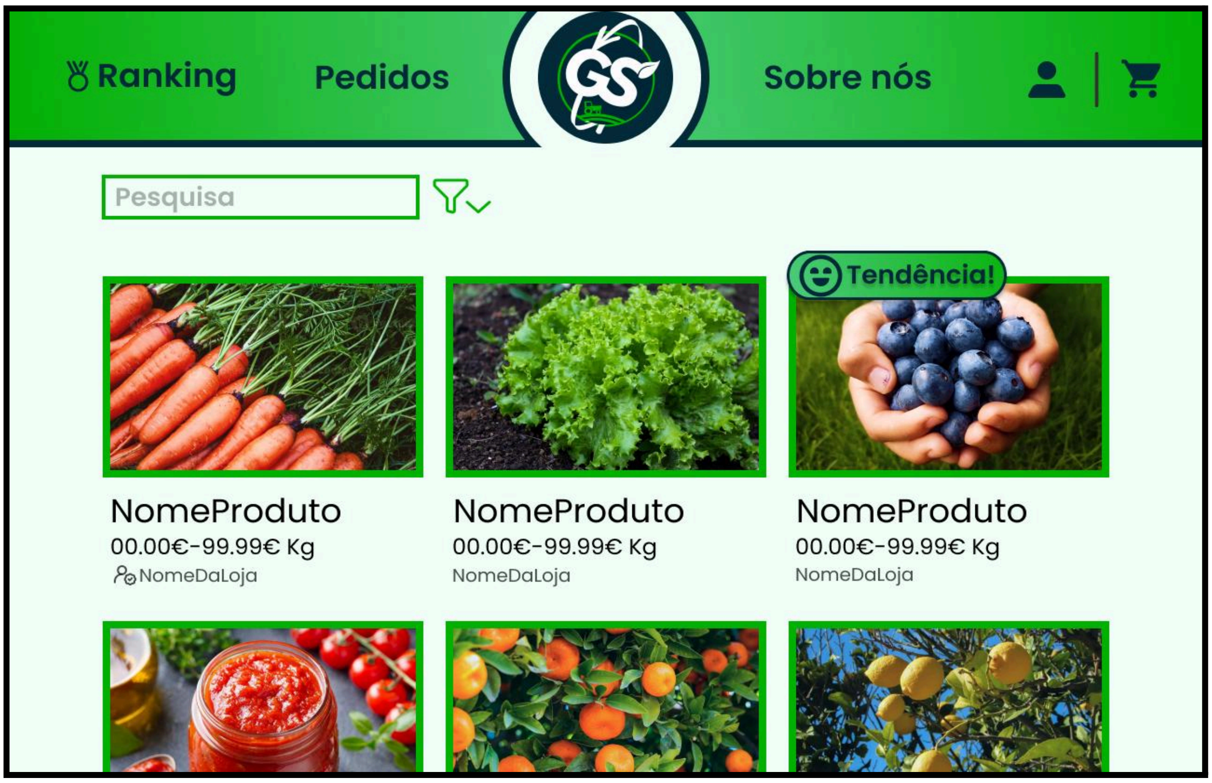


Fig.14 - Home page mockup

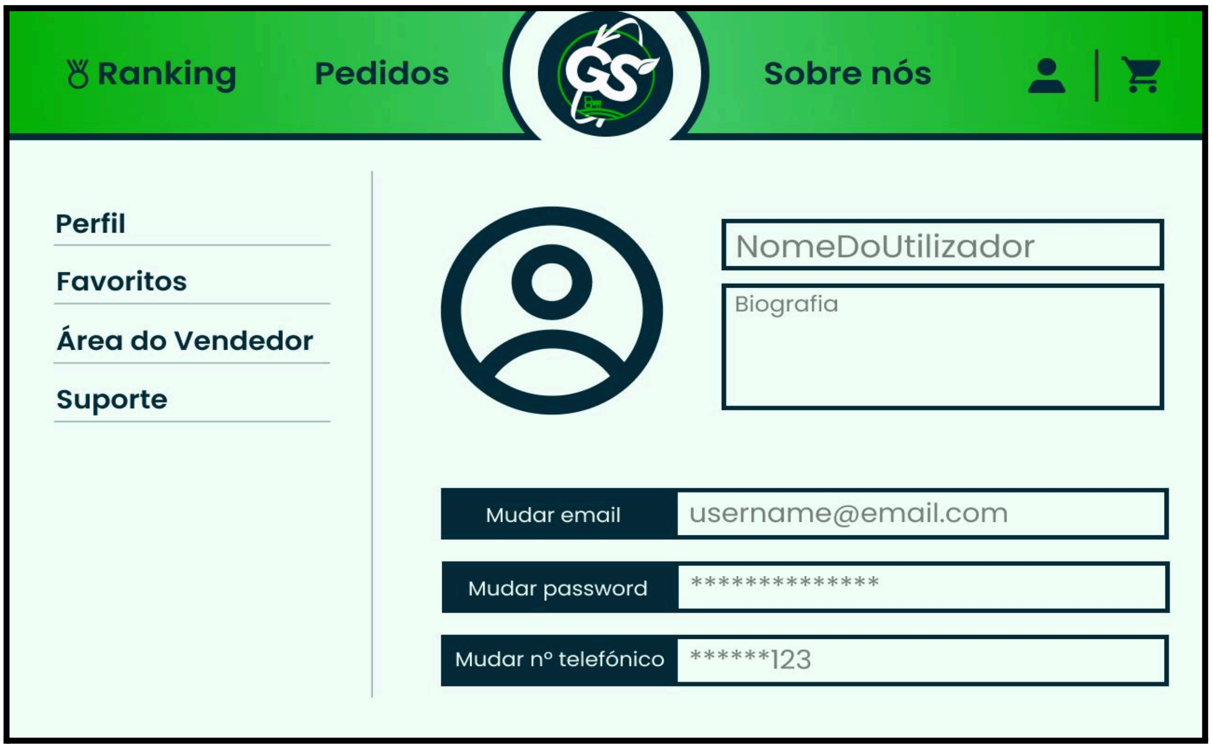


Fig.15 - Profile page mockup

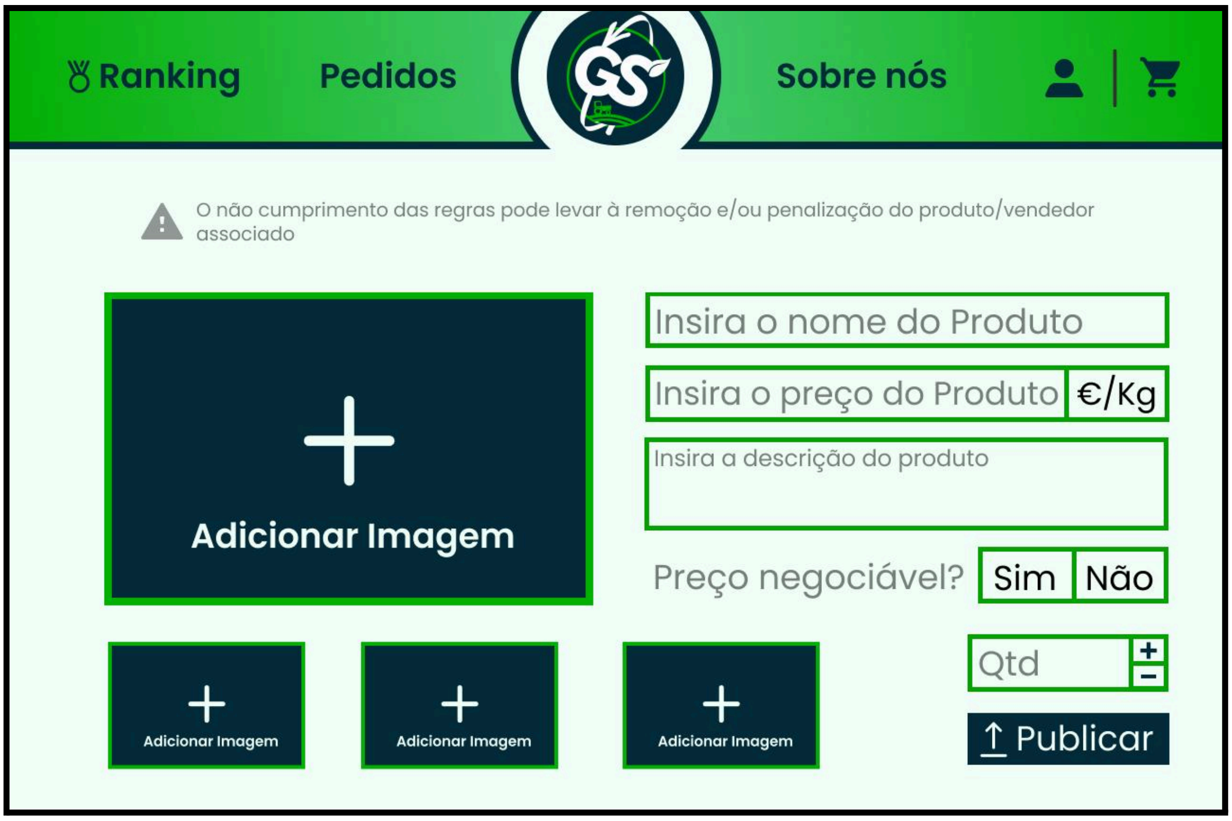

Ranking
Pedidos
Sobre nós
O não cumprimento das regras pode levar à remoção e/ou penalização do produto/vendedor associado
Adicionar Imagem
Insira o nome do Produto
Insira o preço do Produto
€/Kg
Insira a descrição do produto
Preço negociável?
Sim
Não
Qtd
Adicionar Imagem
Adicionar Imagem
Adicionar Imagem
Publicar

Fig.16 - Add product page mockup

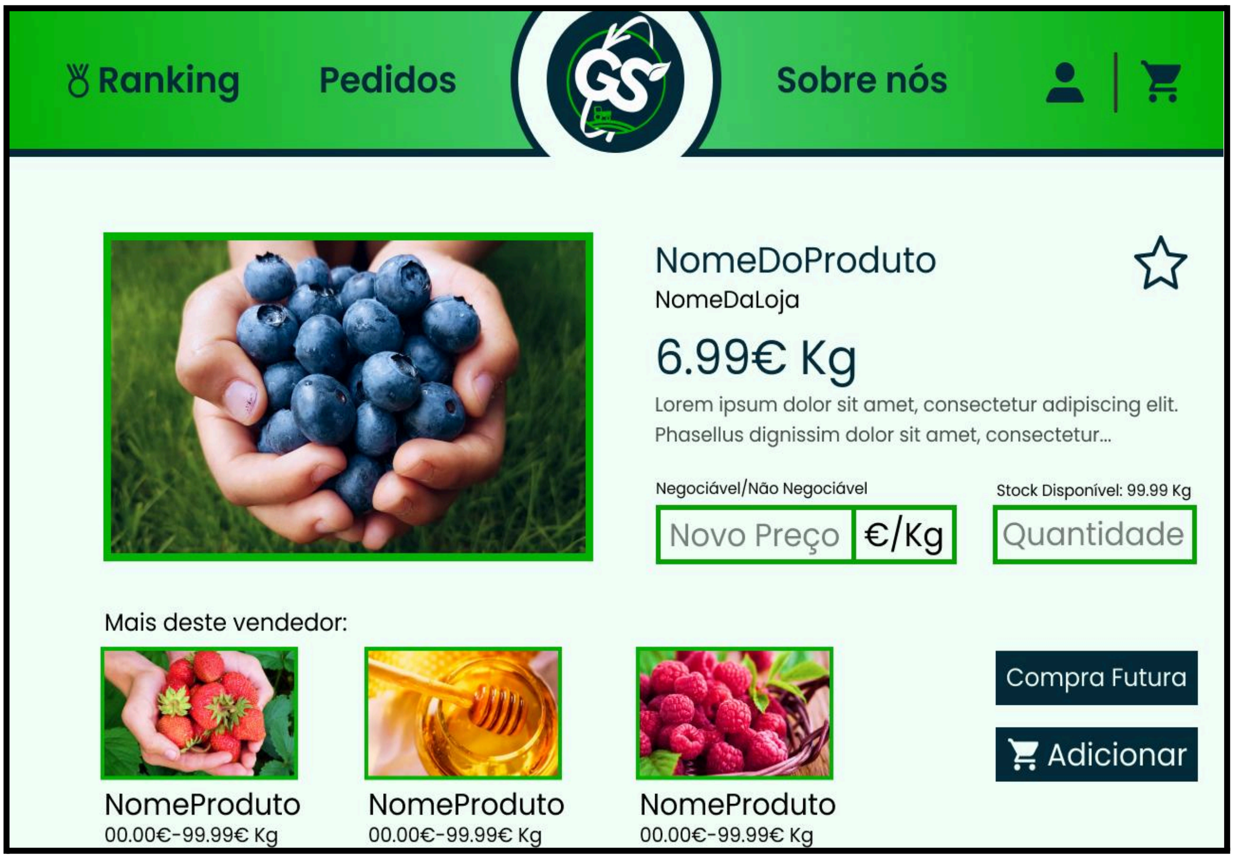


Fig.17 - Purchase page Mockup

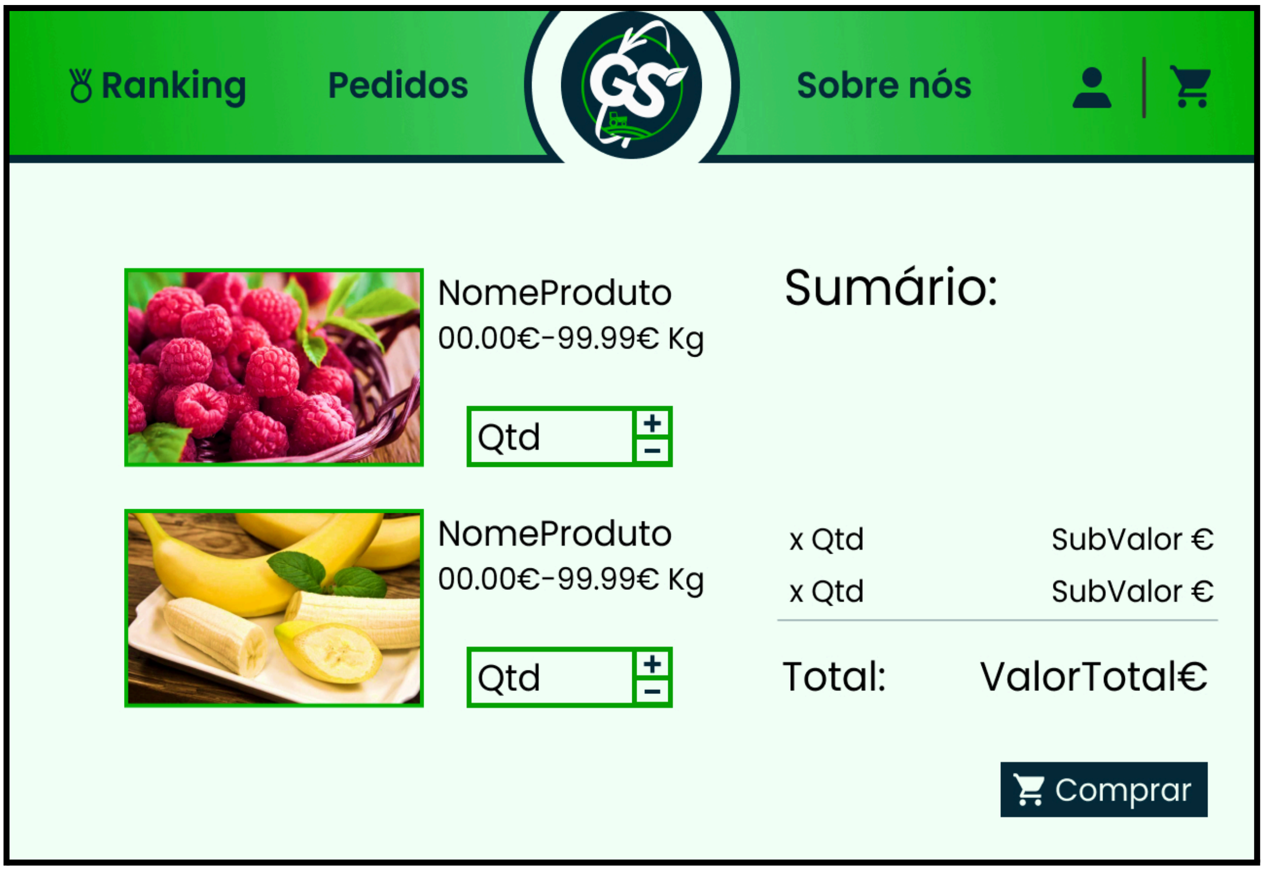


Fig.18 - Check-out page Mockup

II. Data Base

Databases are crucial tools for organizing, storing and efficiently retrieving information. They play an important role in the functioning of the API, facilitating data management and improving application performance.

To create the database, we used the **MySQL** management system. It uses the SQL language and allows you to access, add, edit and manage data. In addition to security and easy data maintenance, one of the characteristics of MySQL is its client-server architecture, where client requests are processed directly. To connect the database to the API, Sequelize is used. Sequelize is an ORM (Object-Relational Mapping) library that facilitates interaction with SQL databases and allows developers to define data models in JavaScript, simplifying CRUDs.

To facilitate code maintenance and improve database performance, Stored Procedures were used. Stored Procedures are a set of SQL commands that function as routines within a database management system. They serve as a more controlled and repetitive way of executing SQL commands, making them run more securely, with more performance and with greater ease of maintenance.

Installation:

For the installation of Sequelize, a directory must be created with a Node.js project (as you need Node.js and "npm" installed to install Sequelize). To install, we use the command "npm install sequelize "base de dados" (in the case of MySQL, it is "mysql2") in the terminal. To initialize Sequelize we use the command "npx sequelize-cli init". We can install MySQL directly from the official website, "https://dev.mysql.com/downloads/mysql/.".

II. API

The API is the most important part of the construction of our project. It's in the API that all the communication between different components of the software occurs, allowing trading information to occur in a safe and efficient way. The chosen architecture is REST (a RESTful API was used for the connection between components), which privileges simplicity and flexibility. In choosing this architecture, we used HTTP methods (GET, POST, PUT, DELETE) that make the implementation easier.

For the API development, various tools were used for the construction of the back-end, front-end, database and necessary connections between them. We had as an objective to guarantee good practices, facilitate maintenance and improve the quality of the code.

As stated above, several tools were used to develop the API, mainly **Node.js**, which was the environment chosen for the Project. Node.js is a server-side JavaScript runtime environment that allows developers to use JavaScript to write code that runs on the server to provide a unified approach to full-stack development. The "npm" (Node Package Manager) was also used, which is the standard Node.js package manager allowing developers to install, share and manage open source libraries and packages.

To test the API, we use two tools. One of them is JEST, which is a testing framework for JavaScript that performs a set of unit tests on the functions created. It is designed to be simple to set up and use, while providing the features needed to test code effectively. The other tool used is Postman, which is a tool used by developers not only to test, but to develop and document APIs efficiently. Unlike unit tests performed with Jest, which focus on verifying functions, methods or classes in isolation, Postman performs validation tests that aim to validate the complete functionality of the system, simulating an interaction between the user and the software.

Installation:

To install Node.js, connect to the official website "nodejs.org". To confirm the installation, we open the terminal and use the "node -v" command, which will indicate the version of node.js installed (if it is installed correctly). Along with Node.js, "npm" is also installed.

To install Jest, a directory must be created with a Node.js project (as you need node.js and "npm" installed to install Jest). To start the installation, we write the command “npm install --save-dev jest” in a terminal. After installation, a npm script can be configured, and it will be possible to use the “npm test” command in the terminal to run unit tests.

We can install Postman directly from the official website, “postman.com”.

The API structure is composed as follows:

- **config:** configuration file for the application, in this case database configuration. For this configuration, Sequelize (ORM) is used;

- **controllers**: these are the controllers that process the request processing logic. In the case of this project, controllers are responsible for handling HTTP requests and interactions with the database (using Sequelize), thus ensuring that operations are carried out in an asynchronous and secure manner;

- **models:** defines the schemas and data models that represent the application's resources. It is by using Sequelize that we are able to represent a database table as an application object with models. This makes access simpler and more productive;

- **node_modules:** node_modules is a directory automatically generated by “npm” that contains all the project’s dependencies. These dependencies are essential and necessary for the project to function correctly;

- **routes:** set of files where endpoints are defined, making a route to the appropriate methods in the “controllers”. Furthermore, they help keep the code organized, facilitating maintenance;

- **services:** services contain the application's business processes and are responsible for coordinating operations between controllers and other system components;

- **tests:** contains unit tests, which are used to ensure that functions are working correctly and guarantee the quality of the code. In this case, JEST (JavaScript testing library) was used.

It also contains 4 files that are necessary to run the API:

- **app.js:** Responsible for configuring the Express application;

- **package-lock.json:** Manages the project's dependencies and scripts, containing metadata about the project (e.g. name, version, description, etc…);

- **package.json:** This file is automatically generated by npm and is used to lock the exact versions of project dependencies;

- **server.js:** is responsible for creating and starting the HTTP server using the Node.js HTTP module.

II. Website

In projects developed with Node.js, web servers are essential components. They are responsible for managing customer requests, serving resources, and executing application logic, ensuring data is processed and delivered efficiently.

For the development of the web server, one of the most popular frameworks for Node.js was used, which is Express. Express is a minimalist and flexible web framework that provides varied functionality for building web applications and APIs.

In addition to this framework, **React** was also used. This JavaScript library is widely used for its efficiency and flexibility and allows you to create complex user interfaces. One of the most important features of React are the components it is based on. These components are reusable building blocks that make maintenance easier.

Installation:

To install Express, a directory must be created with a Node.js project (as you need Node.js and "npm" installed to install Express). After creating the Node.js project (using the "npm init" command), we enter the "npm install express" command in the terminal, and the installation will be carried out.

To install React, a directory must be created with a Node.js project (as it requires Node.js and "npm" installed to install React). First, we install the Create React App in the terminal with the command "npm install -g create-react-app". When installing, we were able to create a React project with the command "npx create-react-app "project name"" within the chosen directory. With the installation done, you should now be able to use the "npm start" command to start the development server.

II. Setup Instructions

#API
1. Install Node.js, Sequelize, and Express if not already installed.
2. Navigate to the `api` directory.
3. Install dependencies with `npm install`.
4. Configure the database connection in `config` foulder.
5. Run API tests using the command `npm test`.
6. Start the server with `node server.js`.

#Website
1. Navigate to the `website` directory.
2. Install dependencies with `npm install`.
3. Start the development server with `npm start`.

#Database
1. Install MySQL.
2. Create a new database and configure the connection in the API.
3. Create necessary tables and populate with sample data if needed.

#Server (Azure)
- The server is hosted on Azure for flexible and scalable infrastructure.
- Configuration details and deployment instructions can be found in the `azure_server` directory.

Conclusion

In short, the Green Swap software developed by VerDev is a comprehensive and affordable solution for managing an Organic Products Marketplace. With an intuitive interface and robust functionalities, Green Swap facilitates interaction between customers and sellers, promoting a sustainable and conscious lifestyle.

This project reflects VerDev's commitment to offering practical and innovative solutions to meet the growing needs of the market, standing out as a vital tool in promoting organic agriculture and promoting healthier and more responsible consumption habits.

Bibliography

- Powerpoints for Software Development Project classes;
- https://www.w3schools.com/nodejs/
- https://www.w3schools.com/REACT/DEFAULT.ASP