Universidad Distrital Francisco José de Caldas

Faculty of engineering

Technical Report Online Platform

Advanced programming

Cristian Santiago López Cadena 20222020027

Carlos Alberto Barriga Gámez 20222020179

Bogotá, D.C.

1. Project Description:

The project consists of the development of monolithic software for an online store. In terms of technical decisions, the project was limited to the processes of purchasing items on the website and the appearance of the items purchased by the user within a tab called purchased items.

The companies that would use this software have a business model focused on the sales of various items through the Internet.

Concerning stakeholders, we consider that the main interested parties in software with these characteristics are the owners of sales companies that intend to expand into online sales and all users who will benefit from being able to buy online. Likewise, we think that students in training can find in this project a tool to complement their knowledge through the review of software currently used by different companies.

Now, among the tools necessary to develop this software, and taking into account that it will be a monolith, it was decided to use Python for the development of the back-end together with the fast-api framework. For the graphical interface, it was decided to use HTML, CSS, JavaScript, Apache server, and Django framework. Finally, for the database, the SQLAlchemy software will be used, since it allows managing database relationships with a syntax similar to that of the programming language.

Because the software is required to be developed cooperatively, the GitHub software will be used to manage software versions.

Regarding the hardware that is required to be used for the development of the online sales platform, we will use 1 GB memory RAM, 1 GB of hard drive, and a tenth-generation Intel Core i5 processor.

2. User Stories:

- UH1: As a manager I want a platform where I can publish the different products that I offer to anyone so what the number of sales increases.

- UH2: As a community manager I want my company's logo to be seen on the main page so what all users who enter can see and recognize my company.

- UH3: As a manager, I want to show on the main page the products that I consider relevant to the customer, so what the user can view the products that he can buy.

- UH4: As an administrator, I want an option for the buyer to log in or register if they do not have an account, so what I can identify the buyer in the future.

- UH5: As a buyer I want to create an account so what i can see all the purchases that I made on the platform.

- UH6: As a buyer, I want to create an account to save my shipping address, so what I can use it in future purchases.

- UH7: As a buyer I want to access the main page so what i can select the product I want to buy.

- UH8: As a buyer, I want to see the products in the main interface, with name and price, and also an image so what i can identify it more easily and thus select the option that I consider appropriate.

- UH9: As a manager I want the application to not show those products that are out of stock, so what the user only buys the avaible products.

- UH10: As a manager I want it to be verified at the time of payment, so what the user or buyer has identified themselves as a user of the application and if not, force them to identify themselves.

- UH11: As a buyer I want to be able to purchase more than one unit of a product, so what i can see the total I must pay.

- UH12: As a buyer, I would like to have a search bar, so what i can search the product that I want.

- UH13: As a manager, I want to provide the platform with a way to search for products by name and display on the same page the searched product and similar products so what the user can buy the product that best suits to he needs.

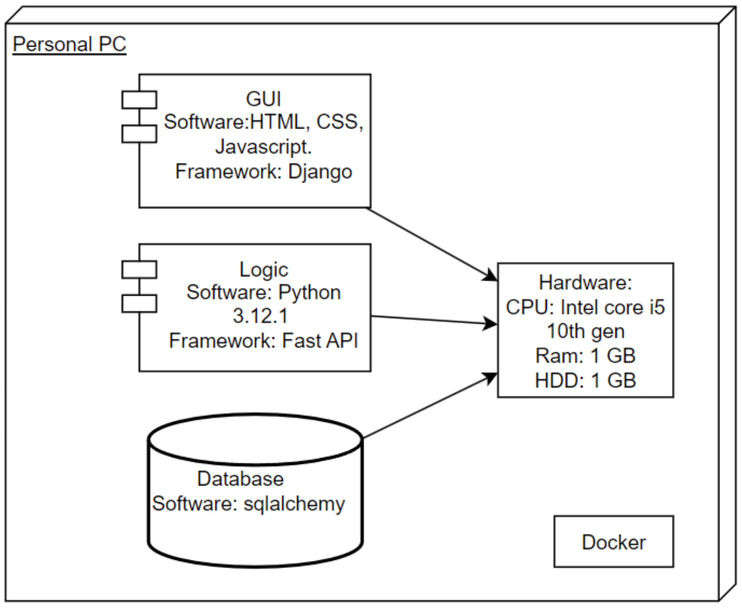
- UH14: As a manager I would like to be able to add information about the products that are for sale such as their name, current stock, main characteristics, department to which the product belongs, price, description, images of the product, and store that sells the product, so what the user knows all the characteristics of the product.

- UH15: As a manager, I want to show a preview of the product with the following structure: a general image of the product, the name, the price, and a buy button, so what the user can access easily to the product that he wants.

- UH16: As a manager I want the preview to go to a main view in which all the details of the product are seen but the purchase button is maintained, so what the user can buy the product when he wants.

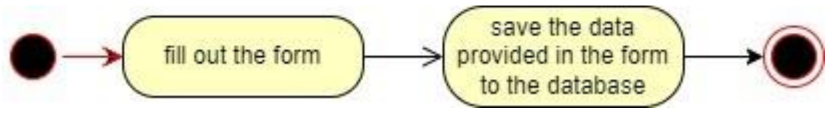
3. UML diagrams

- Deploy diagram: This deployment diagram shows the basic characteristics of the machine where the application

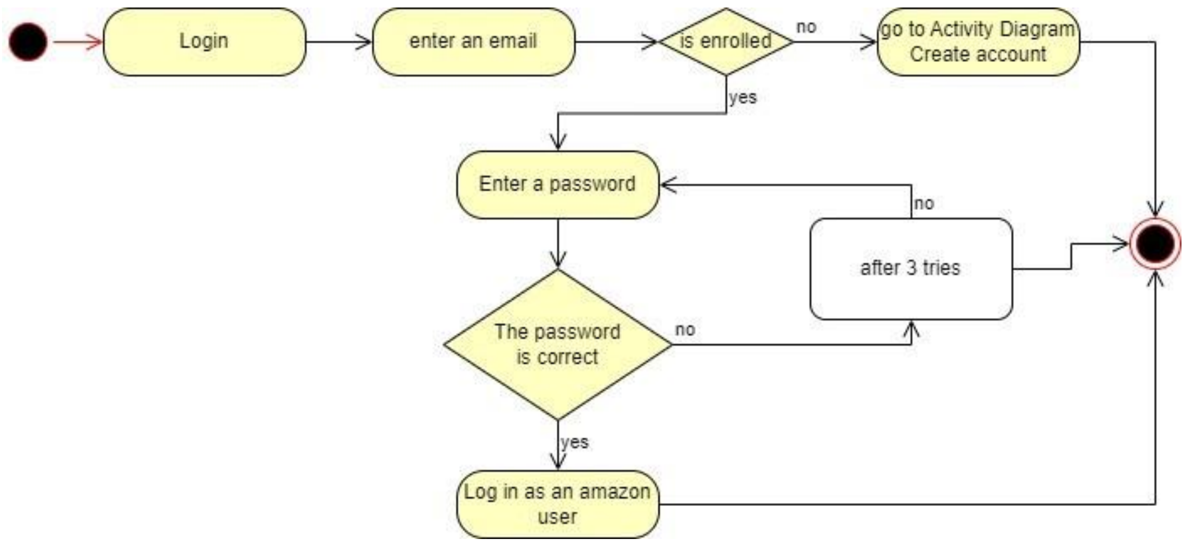


- Activity Diagrams:

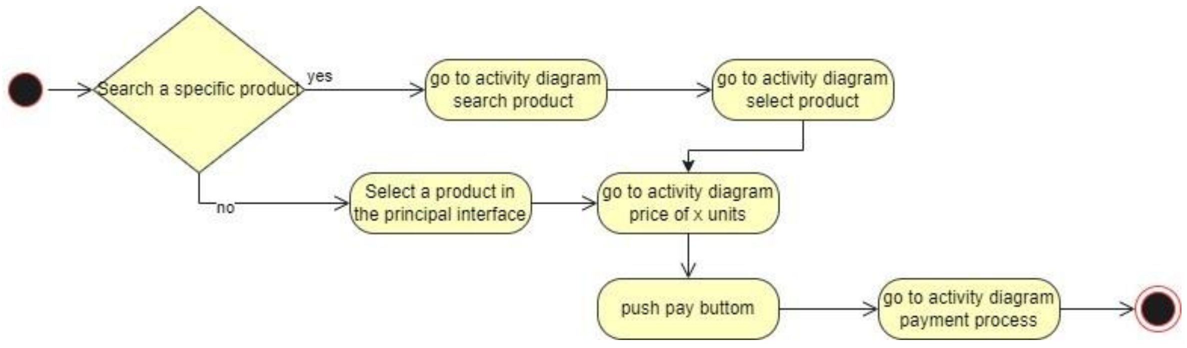
Activity Diagram Create account: This diagram shows the steps to follow in the application to create a new user, which consists of filling out a form whose data will be sent directly to the database using a web services.



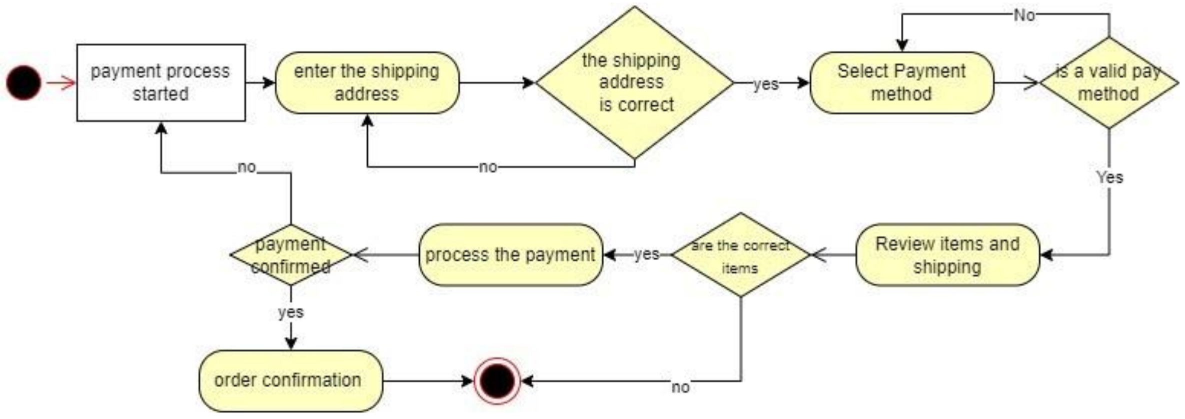
Activity Diagram Login: This diagram shows the steps to follow in the application to validate the credentials of the user, which consists of filling out a form whose data will be compared directly to the data stored in the database using a web service.



Activity diagram Buy a product: This diagram shows all the steps necessary to purchase a product in the app, these steps consist of selecting a product, selecting the quantity of this product, and finally pay the product. These steps can be seen in more detail in the next diagrams.

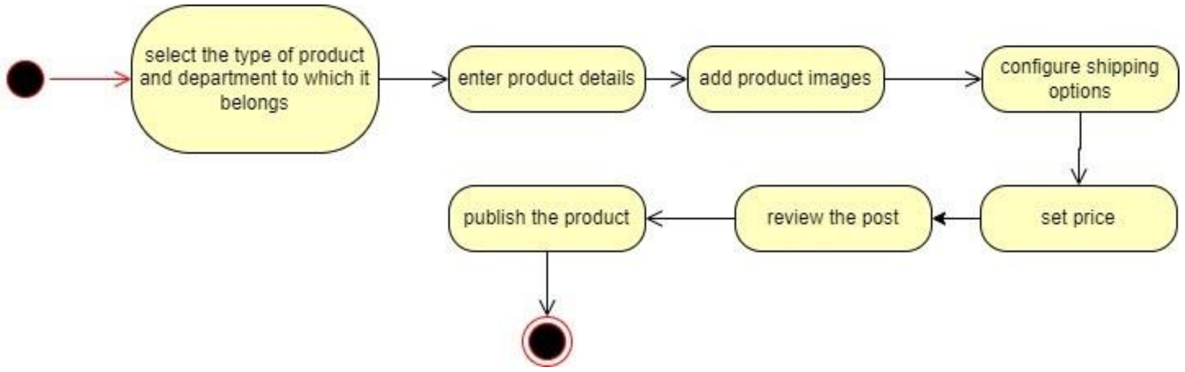

Search a specific product
yes
go to activity diagram search product
go to activity diagram select product
no
Select a product in the principal interface
go to activity diagram price of x units
push pay buttom
go to activity diagram payment process

Activity Diagram Payment Process: This diagram shows all the steps necessary to pay for a product in the application, these steps consist of once the first steps of the Buy a product diagram have been completed, an address and a payment method are provided, these will be visible to the user so that they can modify them if necessary. If necessary, once this process is carried out, the product being purchased will be shown. If it is the desired product, payment will be made through a button. If the payment is successful, an order confirmation will be generated, If the product is not the desired one, this process will be completed and if the payment cannot be validated, the entire process will be restarted.



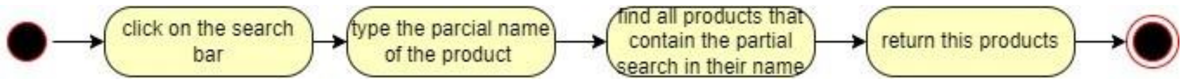
Activity Diagram Post Product:

This diagram shows how the administrator can create a new product by entering the price, name and image details of the product and then publishing the product.


select the type of product and department to which it belongs
enter product details
add product images
configure shipping options
set price
review the post
publish the product

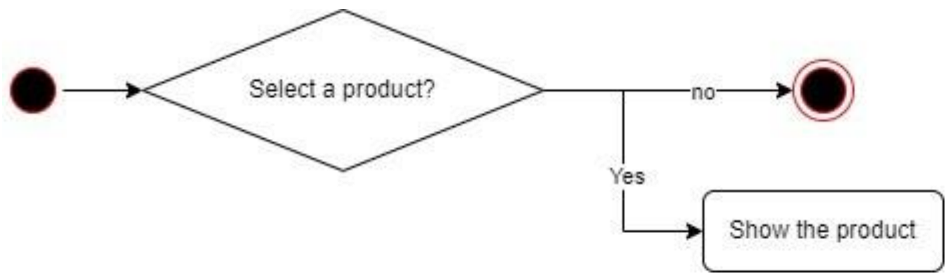
Activity diagram search product:

In this diagram the user has the possibility to enter the name of the product they want to buy and the system displays this product.


click on the search bar
type the parcial name of the product
find all products that contain the partial search in their name
return this products

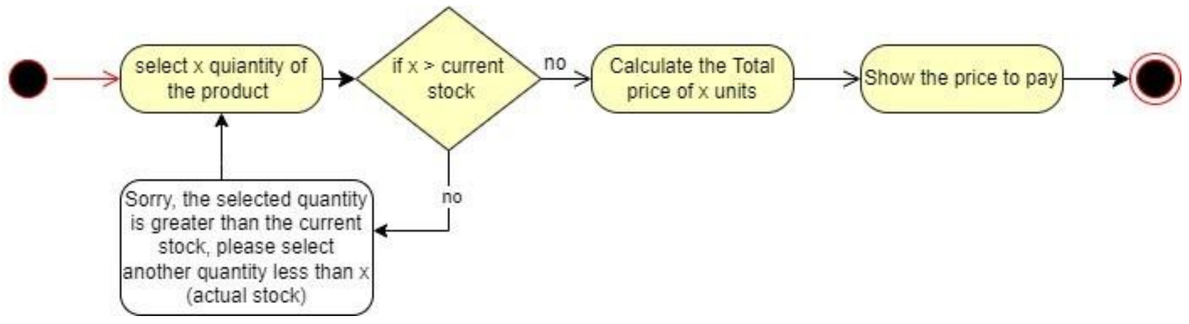
Activity diagram: select product.

This diagram shows how the user has the possibility of selecting one of the products shown in the application, if they choose it the system redirects them to that specific product, otherwise the user will continue in the same interface.



Activity diagram: Price of x units

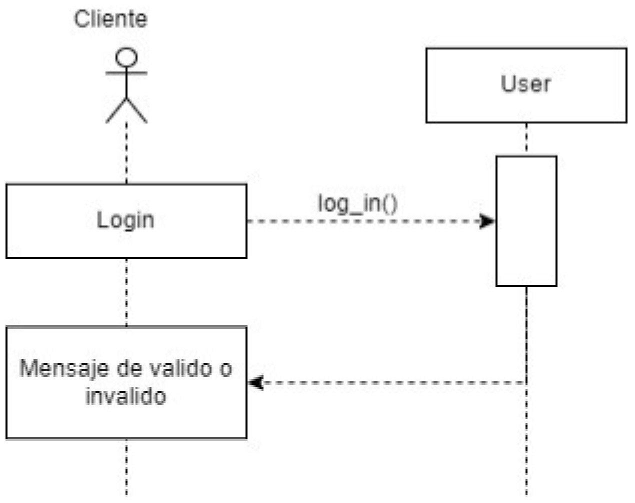
This diagram shows how the user can choose the number of units they want to buy of the product. If the stock is sufficient, the system calculates the total price and shows it to the user.


select x quiantity of the product
if x > current stock
no
Calculate the Total price of x units
Show the price to pay
no
Sorry, the selected quantity is greater than the current stock, please select another quantity less than x (actual stock)

-Sequence diagrams:

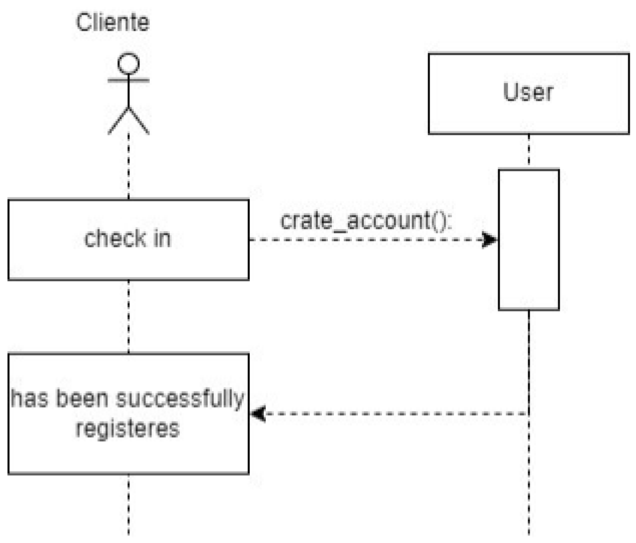
Sequence Diagram: Login

In this case, the user tries to log in to the system, and the system returns a message of validity or invalidity depending on the data entered.



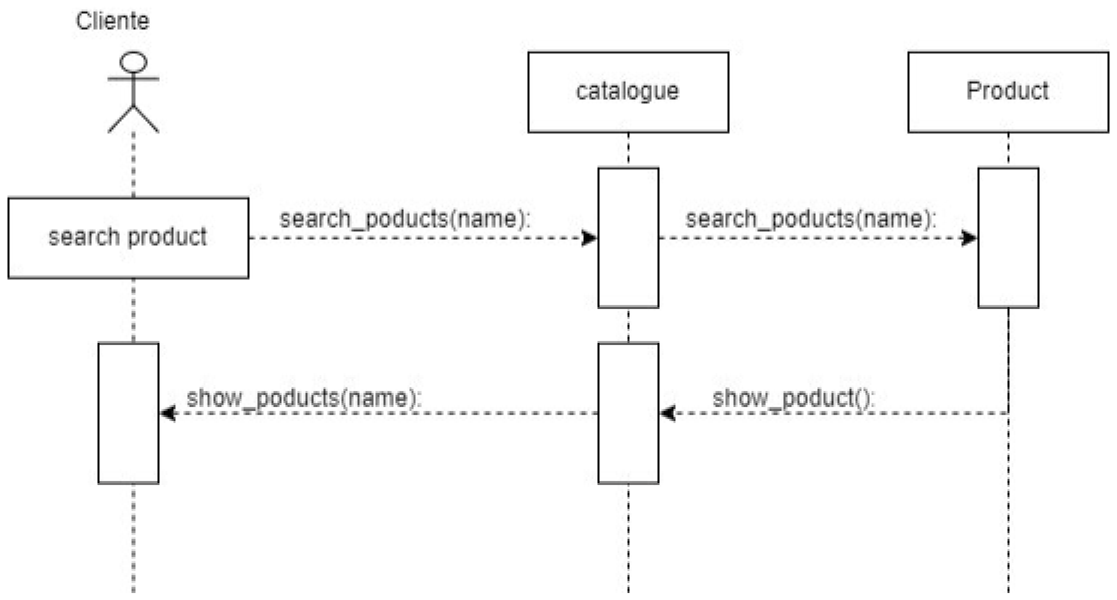
Sequence Diagram: Create user

This sequence diagram shows how the customer tries to create an account, and the system returns a registration message if he has entered the expected data.



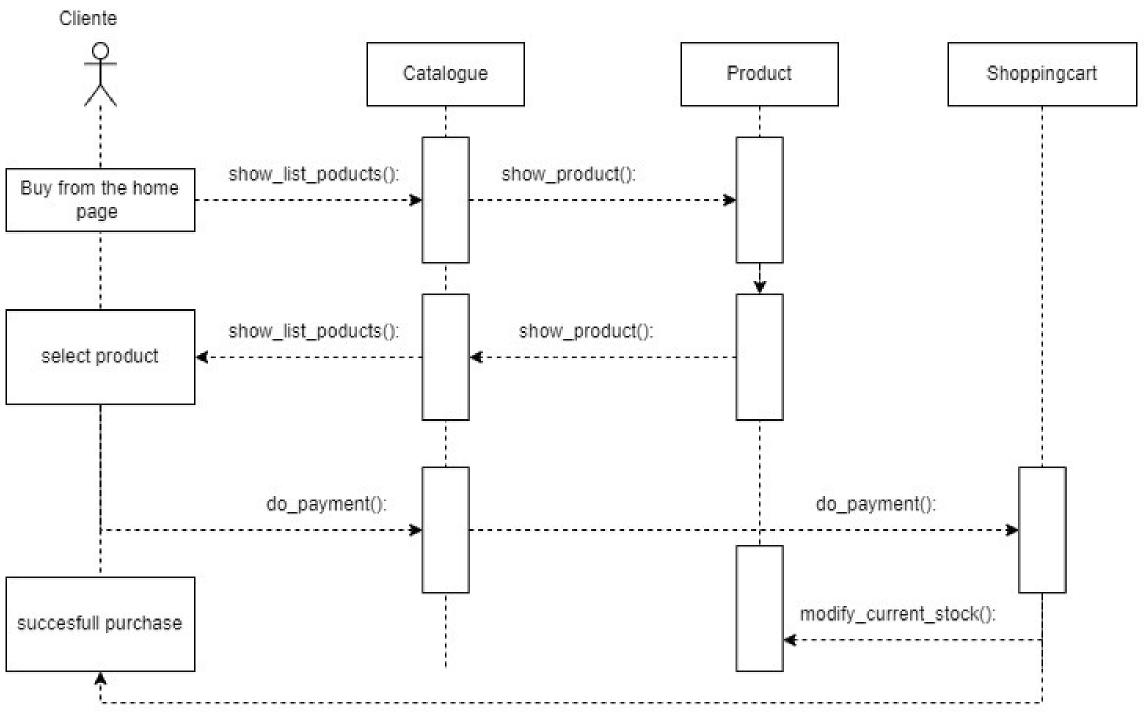
Sequence Diagram: Search Product:

This diagram shows how the user searches for a product with his name, this request is received by the catalog which also requests the product object for a product with these conditions, to finish the product object returns the requested product and then the catalog is shows it to the user.



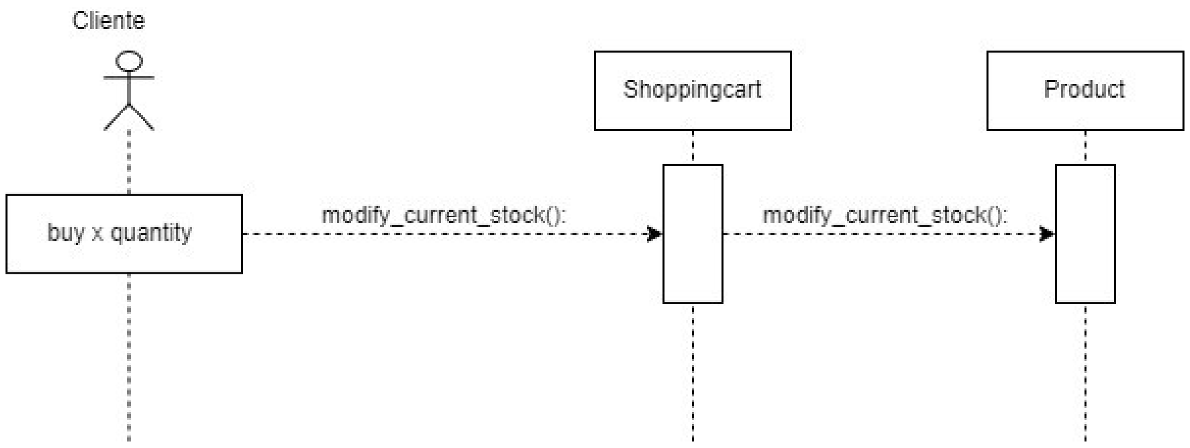
Sequence Diagram: Buy product.

This diagram shows the process to pay for a product, in this case the user requests a product from the catalog, the catalog requests it from the product and the product is shown to the user thanks to the catalog, then the user requests payment, therefore the shopping cart accepts it and the system shows a purchase made message to the user.



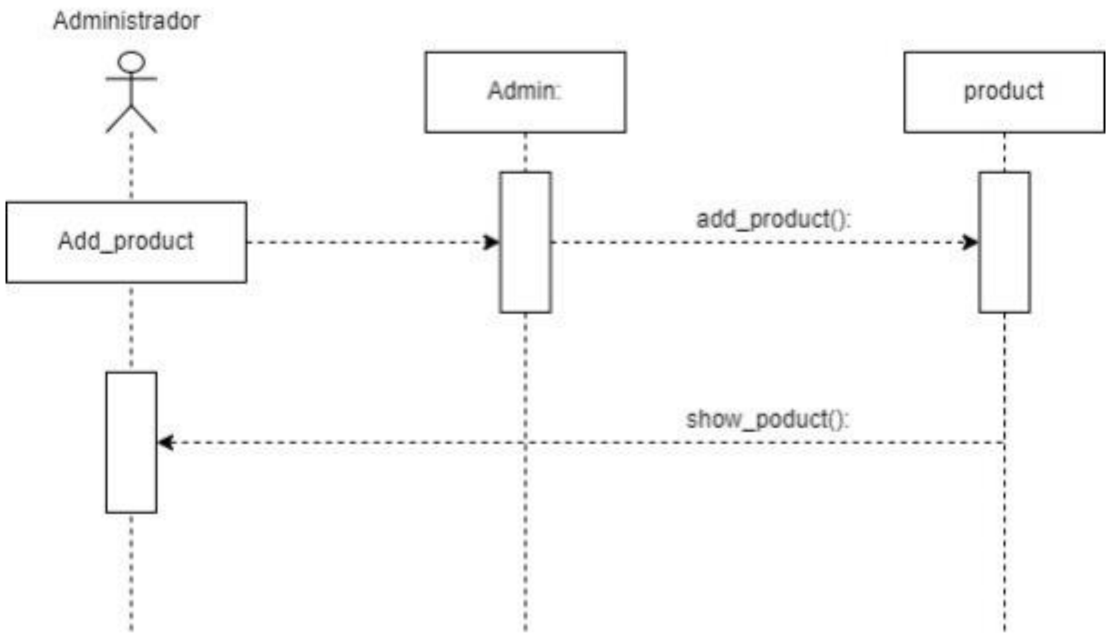
Sequence diagram: modify current stock:

This diagram shows how the stock of a specific product is modified, in this way the customer purchases a certain quantity of the product, and this process is sent to the shopping cart, which reduces the stock of the product.



Sequence diagram: add product:

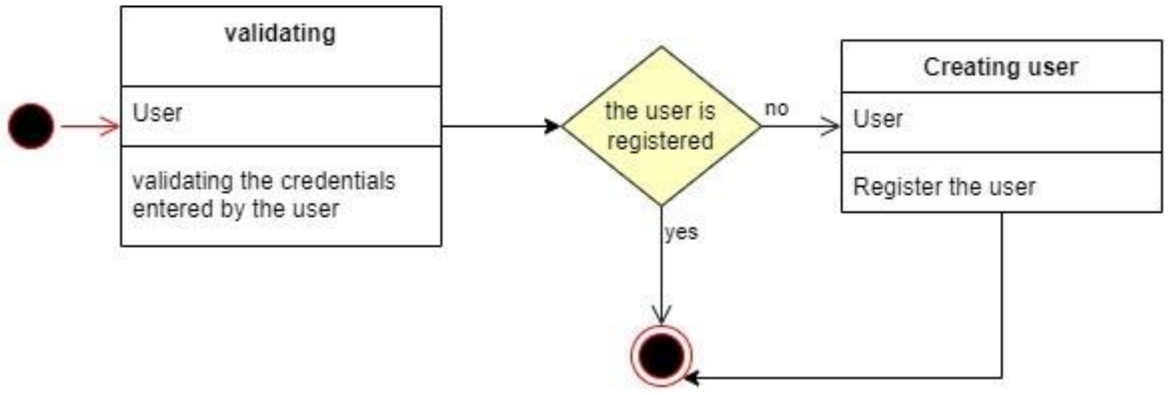
In this diagram the administrator adds a product to the system through the Admin object and the system returns the product that was already added to the store.



-State diagrams:

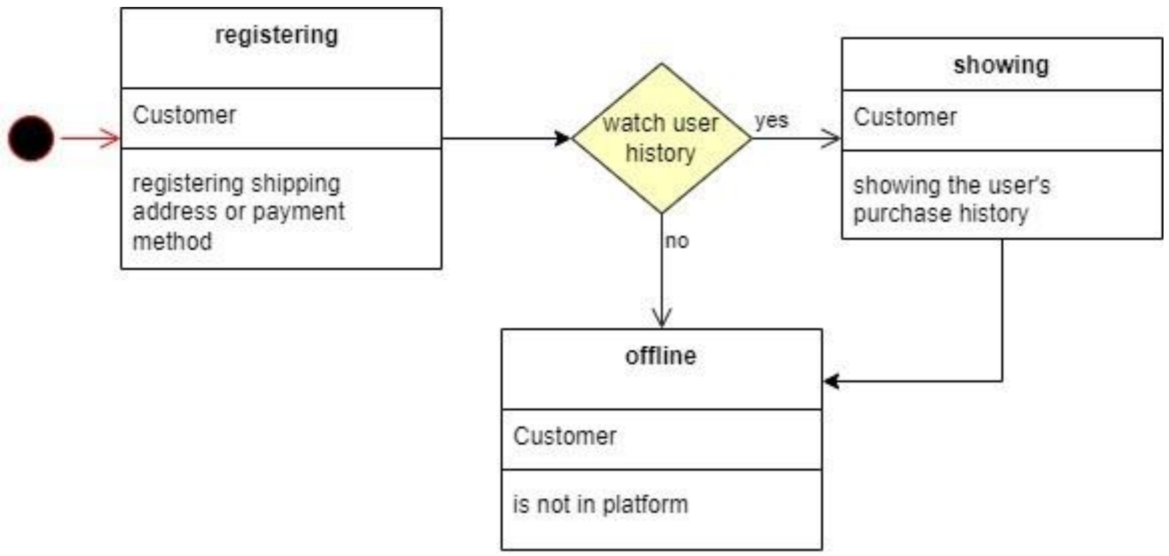
State diagram: User:

This diagram shows how the user's statuses vary, in this way initially the user's credentials are being validated and subsequently if the user is not registered, its status changes to "Creating user", if the user is registered its status changes to "created user".



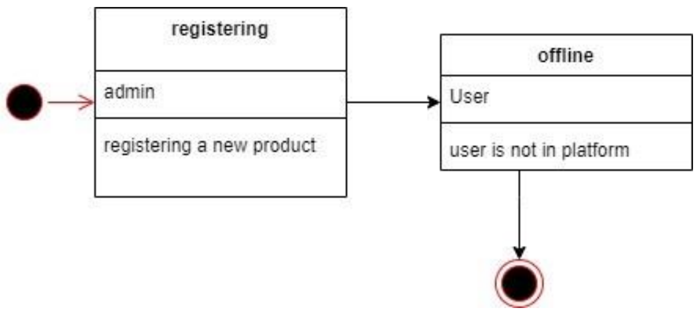
State diagram: Customer:

This diagram shows how the customer's states vary, thus initially the user has a registering state, in which they enter the shipping address and their selected payment method, subsequently the customer goes to a "showing" state in the which shows the purchase history, finally the user when leaving the store has an "offline" status.



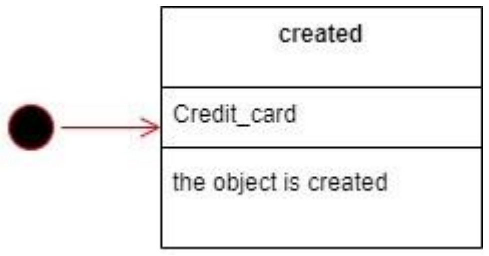
State diagram: Admin:

This diagram shows how the admin states vary, thus initially it is in a state registering a new product, and in case of leaving the store the admin goes to an "offline" state.



State diagram: credit card

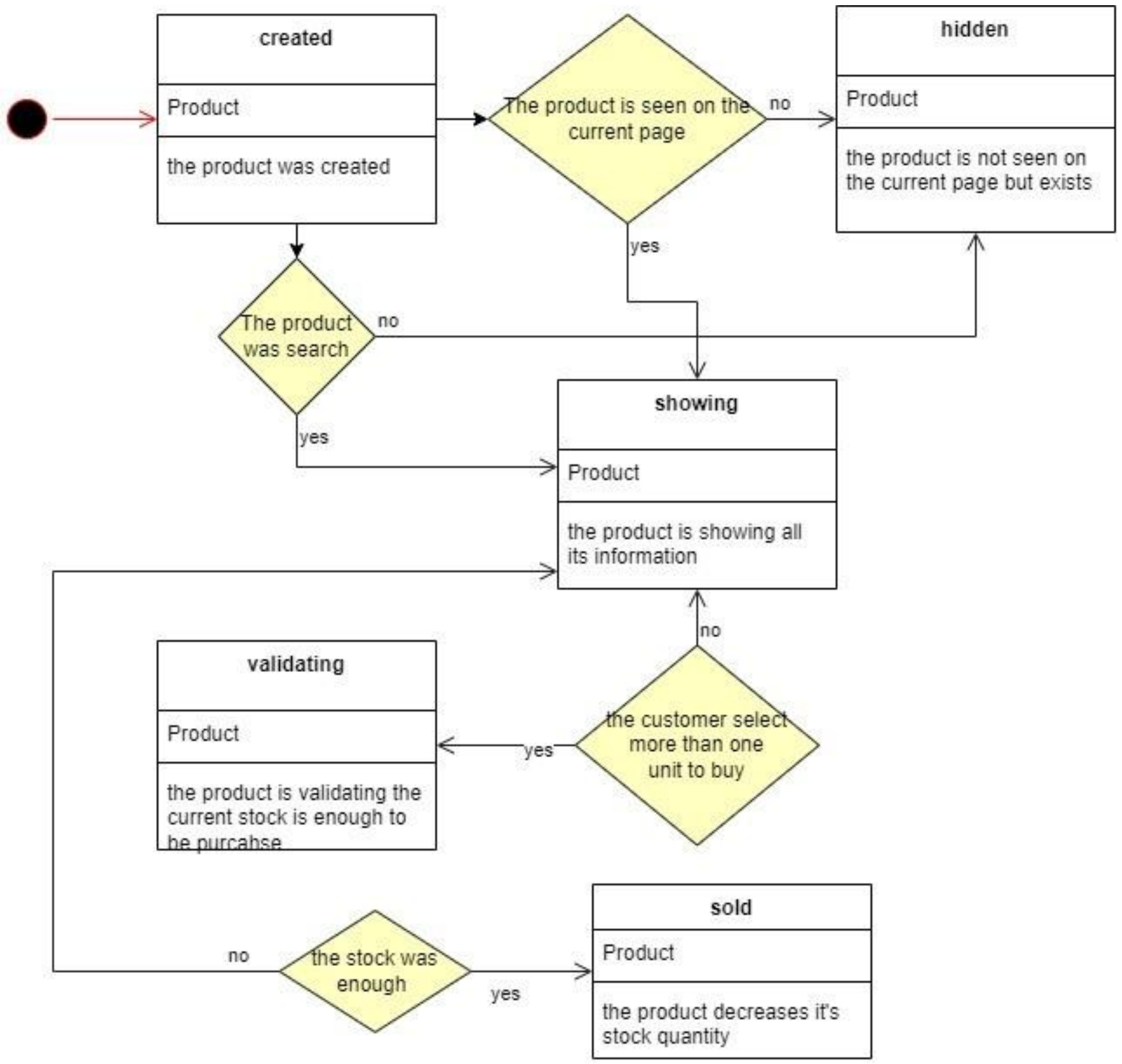
This state diagram shows the variation of states for the credit card, in this way the credit card has a single state called "created".



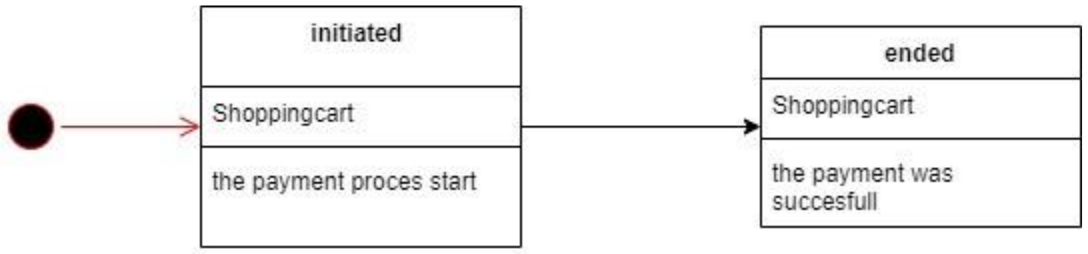
State diagram: Product

The product state diagram shows how the product is initially created, in the case that the product is not shown in the interface, the product has a hidden state, but if the product is being shown in the interface at that moment the product will go to a displaying state.

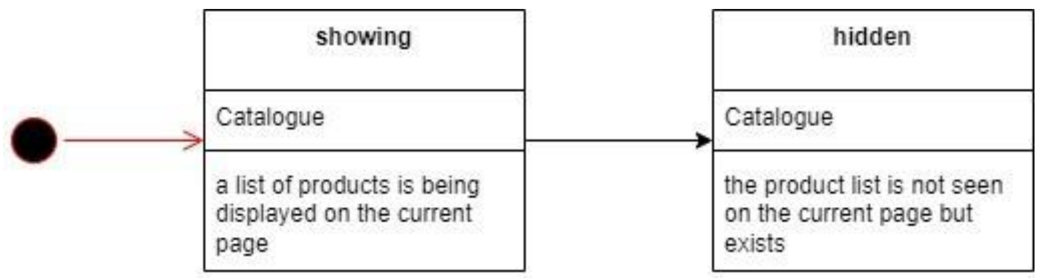
On the other hand, when the user selects this product to purchase, it will have a validating status in which it will be checked if the product is in stock, and finally, the product will have a sold status when the purchase is made.


created
Product
the product was created
The product is seen on the current page
no
yes
hidden
Product
the product is not seen on the current page but exists
The product was search
no
yes
showing
Product
the product is showing all its information
no
validating
Product
the product is validating the current stock is enough to be purcahse
yes
the customer select more than one unit to buy
no
the stock was enough
yes
sold
Product
the product decreases it's stock quantity

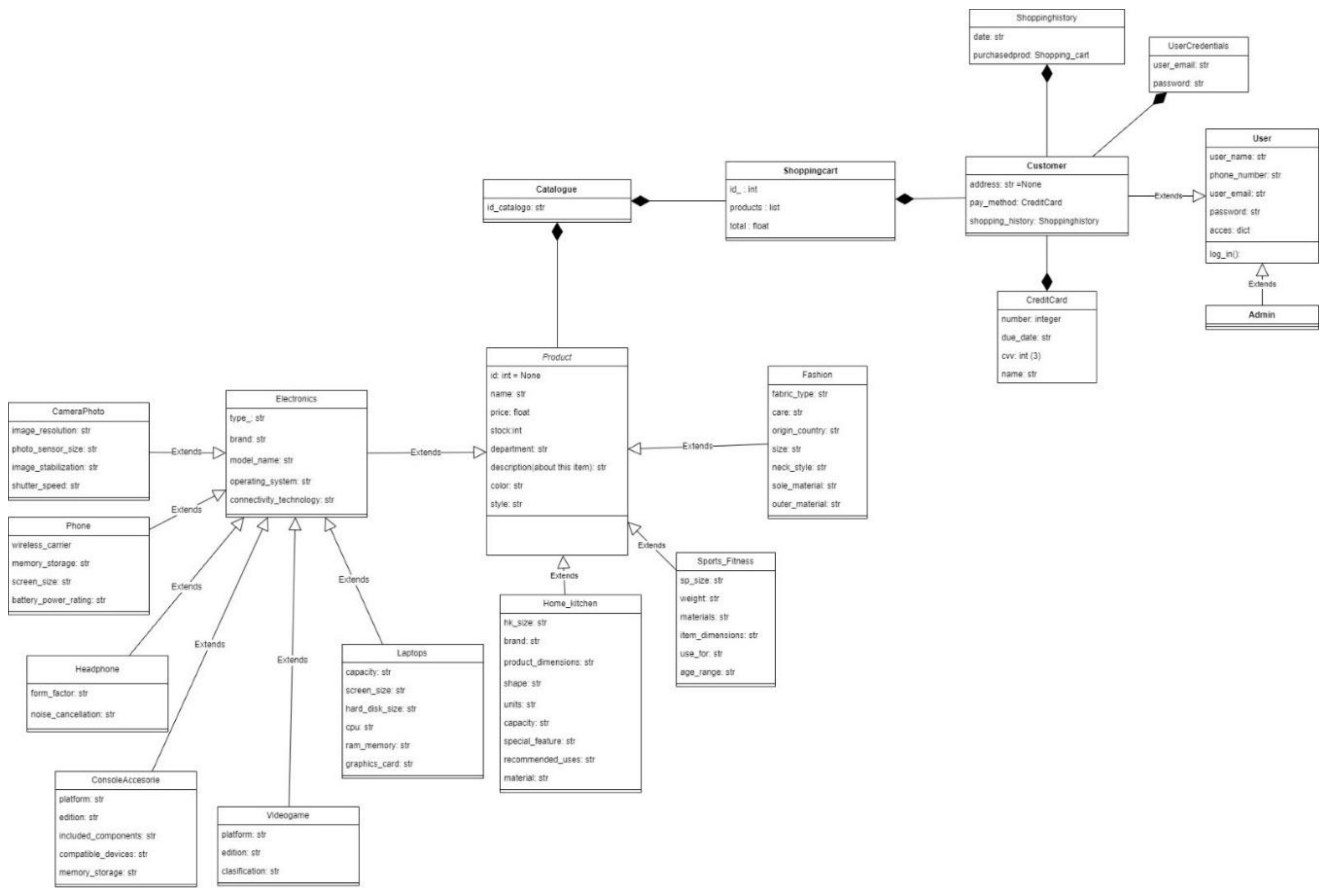
State diagram Shopping cart: This diagram shows the variation of states for the Shopping cart, that was initiated which means that the user pressed the pay button and started the process to pay for the product and ended which means that the purchase was successful, and the payment finished.



State diagram Catalogue: This diagram shows the state that the Catalog class can have, this class has two states, they show that it is when the user is viewing the products that are in the catalog, and hidden is when the user is in other sections of the application such as login.


showing
Catalogue
a list of products is being displayed on the current page
hidden
Catalogue
the product list is not seen on the current page but exists

4. Class Diagram



5. CRC Cards

User	
Responsibility	Collaborator

Validate the integrity and structure of data provided to services	Customer and Admin

UserCredentials	
Responsibility	Collaborator
Validate the integrity and structure of data provided to services	Customer and Admin

Customer	
Responsibility	Collaborator
Validate the integrity and structure of data provided to services	User

Shoppinghistory	
Responsibility	Collaborator
Validate the integrity and structure of data provided to services	Customer

Admin	
Responsibility	Collaborator
Validate the integrity and structure of data provided to services	User

CreditCard	
Responsibility	Collaborator
Validate the integrity and structure of data provided to services	Customer

Product	
Responsibility	Collaborator
Validate the integrity and structure of data provided to services	Catalogue

Shopping cart	
Responsibility	Collaborator
Validate the integrity and structure of data provided to services	Customer Catalogue

Catalogue	
Responsibility	Collaborator
• Validate the integrity and structure of data provided to services	Product Shopping cart

6. Technical decisions.

Regarding technical decisions, from the beginning, it was decided to structure the project around the three-layer architecture, to develop a scalable, reliable, and maintainable system. As a consequence of this approach it was decided to divide the application into Frontend and Backend where the Frontend contains the presentation layer and the Backend would contain the application and data layers. Regarding the technical decisions for the backend, once the design stage is completed in which all the necessary components are obtained to solve the problem, it was decided to use the following libraries:

Pydantic, this Python library is designed for data validation and serialization. It uses Python type annotations to define how data should be structured and validated, in our case we used Pydantic's 'BaseModel' class, which allowed us to declare the attributes of a model with their respective data types, and that Pydantic will automatically validate the input data and convert it to the specified types. This allowed us to directly integrate these models into services and databases, without worrying about type errors.

In conjunction with Pydantic, the Optional function of the typing library was used, which allows us to indicate that a variable or argument can be of the specified type or 'None'.

SqlAlchemy, it was decided to use this Python library to interact with the app's database since it gave us the possibility of performing SQL queries using Python expressions and functions, it also provides a very easy way to connect to the database and execute queries, although for this it is also necessary to use the psycopg2-binary library since SQLAlchemy does not include by default the specific drivers to connect to different databases. This library acts as the driver that allows SQLAlchemy to connect and communicate with the PostgreSQL database.

Finally, FastAPI was used to communicate the Frontend and the Backend, through web services. Initially, they wanted to use the Routers, but due to import errors, it was decided to leave all the services in a single file.

7. Information about the data of classes.

The application will have the following information according to each class:

-User class: There will be a unique identifier (id), of type integer (int) which will have an initial value of 'None', "the value of this id will be provided directly by the database", a username of type string (str), a password of type str, an email of type str, a phone number of type str, and a permission of type bool. The Administrator and Client classes will have this same data structure since they inherit from the User class. However, the Customer class will add an address of type str, a purchase history of type Shoppinghistory, and payment methods of type CreditCard. These last attributes could be of the specified type or None, this with the help of the 'Optional' tool provided by the 'typing' library. This will also apply to the id attribute of the User class.

-Shoppinghistory class: Will have a date attribute of type string, and the purchased products will be of type Shoppingcart.

-CreditCard class: Will have the attributes username of type str, the card number of type str, the expiration date of the card of type str, the security code of the CVV card of type int, and the name of the cardholder, the type str.

-Shoppingcart class: Will have an id, under the same conditions as the id of the User class, it will have a list of products of type list and the total price of type float.

-UserCredentials class: will have the user email and password both of type str.

-Product class: Will have an id under the same conditions as the id of the User class, a name of type str, a price of type float, a stock of type int, a department, description, color, and style all of type str. The Fashion, SportsFitness, HomeKitchen, and Electronic classes will have this same data structure since they inherit from the Product class. Also, the CameraPhoto, Phone, Headphone, ConsoleAccesorie, Videogame, and Laptop classes will have the same data structure since they inherit from the Electronic class which inherits from the Product class. However, all these child classes add their attributes corresponding to specific characteristics of each type of product, all these attributes can be of type string or ‘None’ thanks to the ‘Optional’ functionality mentioned above.

-Catalogue: We will have an integer type identifier “ID”.

8. Services

-Create_admin --> Post: {Admin} This service will receive as a parameter an object of the admin class and will proceed to save it in the database.

-Create_customer --> Post: {Customer} This service will receive as a parameter an object of the Customer class and will proceed to save it in the database.

-Login --> Post: {UserCredentials} This service will receive as a parameter an object of type UserCredentials, this object has the attributes user email and password, this service will be in charge of consulting the customer and admin tables of the database looking for a match with the attributes of the parameter in case of finding a User with these credentials will allow you to enter the application, otherwise, a message will return that says: Incorrect username or password

-Add_shipping_address --> Put: {address, user_name} This service will receive as parameters the address that the user of type str wants to add and the name of the user who is adding it. Once the parameters have been received, it will proceed to search the database for the user with the name provided and make an update there. the address column, which will have a 'Nule' value by default

-Add_fashion_product --> Post: {Fashion} This service receives as a parameter an object of type Fashion, it will be responsible for saving the attributes inherited from the parent class Product in the products table, then it will obtain the id generated in the database for this new product, and the result will be the foreign key in the fashion table in which the specific attributes of the Fashion class will be stored.

-Add_sportfit_product --> Post: {SportsFitness} This service receives as a parameter an object of type SportsFitness, it will be responsible for saving the attributes inherited from the parent class Product in the products table, then it will obtain the id generated in the database for this new product, and the result will be the foreign key in the sportfit table in which the specific attributes of the SportsFitness class will be stored.

-Add_homekitchen_product --> Post: {HomeKitchen} This service receives as a parameter an object of type HomeKitchen, it will be responsible for saving the attributes inherited from the parent class Product in the products table, then it will obtain the id generated in the database for this new product, and the result will be the foreign key in the homekitchen table in which the specific attributes of the HomeKitchen class will be stored.

-Add_cameraphoto_product --> Post: {CameraPhoto} This service receives as a parameter an object of type CameraPhoto, it will be responsible for saving the attributes inherited from the parent Product class in the products table, then it will obtain the ID generated in the database for this new product, and the result will be the Foreign key in the camera table in which the specific attributes of the CameraPhoto class and the attributes inherited from the Electronic class will be stored.

-Add_phone_product --> Post: {Phone} This service receives as a parameter an object of type Phone, it will be responsible for saving the attributes inherited from the parent Product class in the products table, then it will obtain the ID generated in the database for this new product, and the result will be the Foreign key in the phone table in which the specific attributes of the Phone class and the attributes inherited from the Electronic class will be stored.

-Add_headphone_product --> Post: {Headphone} This service receives as a parameter an object of type Headphone, it will be responsible for saving the attributes inherited from the parent Product class in the products table, then it will obtain the ID generated in the database for this new product, and the result will be the Foreign key in the headphone table in which the specific attributes of the Headphone class and the attributes inherited from the Electronic class will be stored.

-Add_console_accesori_product --> Post: {ConsoleAccesorie} This service receives as a parameter an object of type ConsoleAccesorie, it will be responsible for saving the attributes inherited from the parent Product class in the products table, then it will obtain the ID generated in the database for this new product, and the result will be the Foreign key in the console table in which the specific attributes of the ConsoleAccesorie class and the attributes inherited from the Electronic class will be stored.

-Add_videogame_product --> Post: {Videogame} This service receives as a parameter an object of type Videogame, it will be responsible for saving the attributes inherited from the parent Product class in the products table, then it will obtain the ID generated in the database for this new product, and the result will be the Foreign key in the videogame table in which the specific attributes of the Videogame class and the attributes inherited from the Electronic class will be stored.

-Add_laptop_product --> Post: {Laptop} This service receives as a parameter an object of type Laptop, it will be responsible for saving the attributes inherited from the parent Product class in the products table, then it will obtain the ID generated in the database for this new product, and the result will be the Foreign key in the laptop table in which the specific attributes of the Laptop class and the attributes inherited from the Electronic class will be stored.

-Show_products --> Get: This service returns all the products that are stored in the database

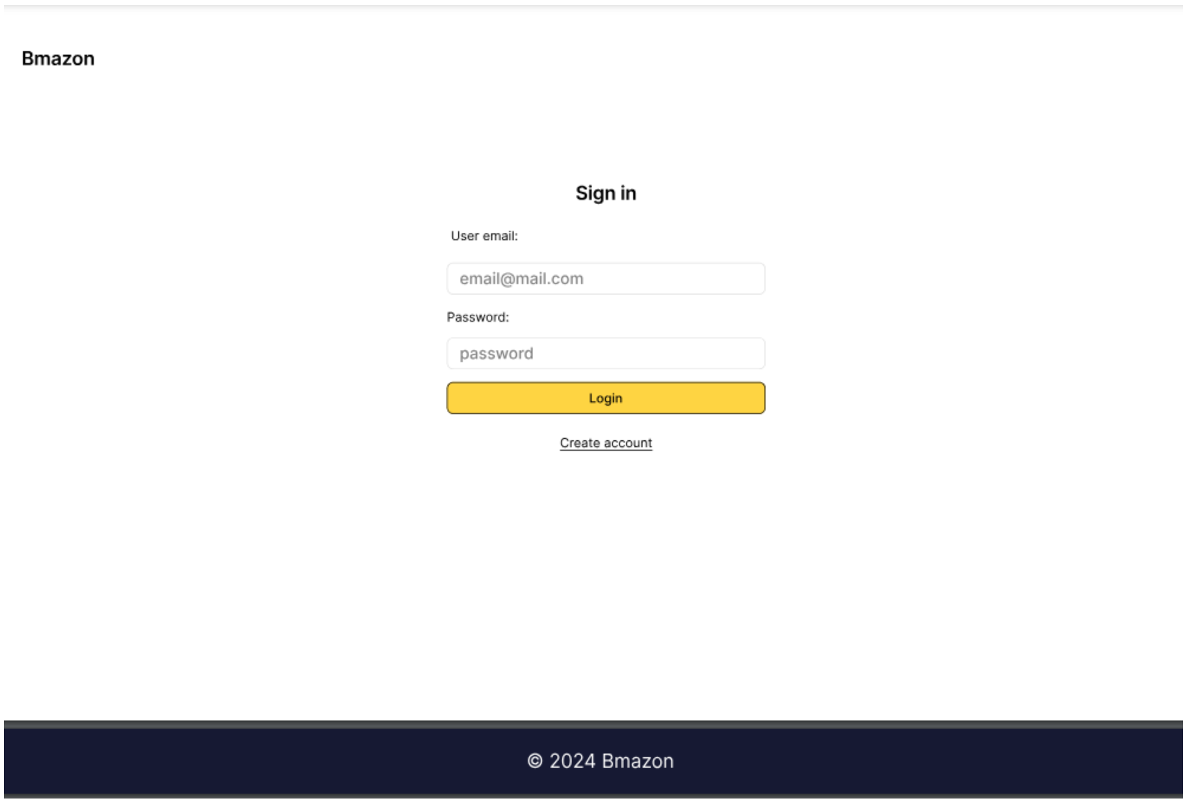
9. User interface design

-Decisions on colors used in interfaces:

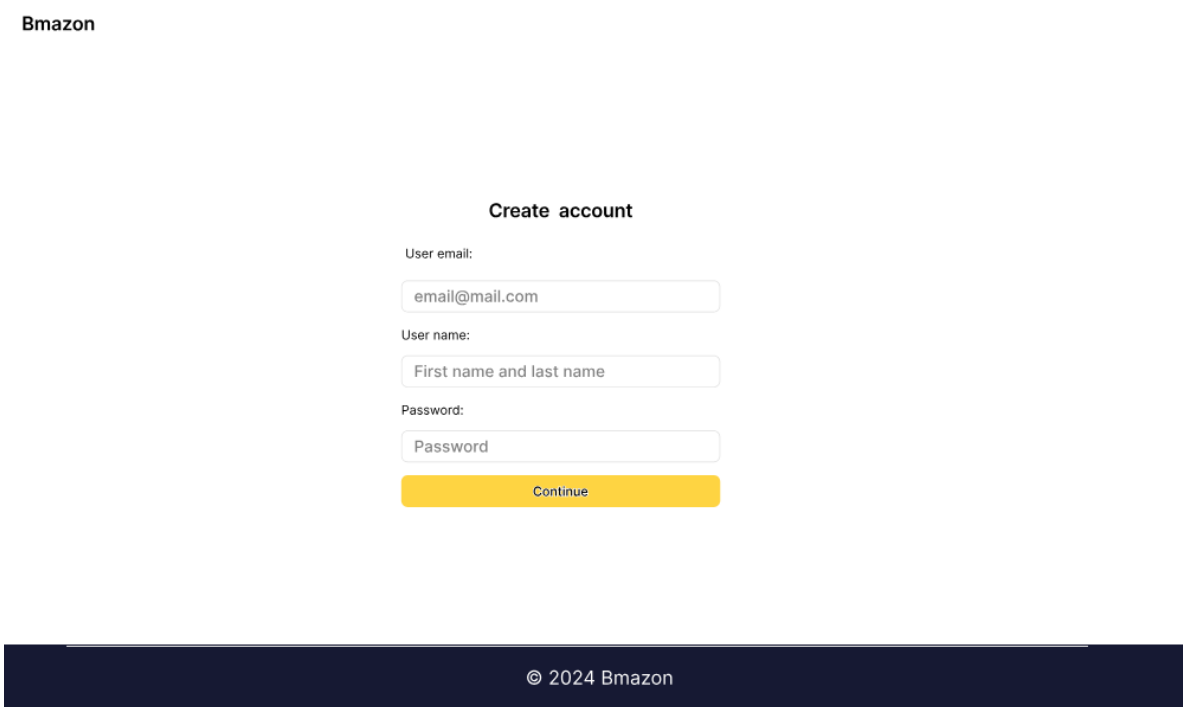
For the development of the graphic interfaces, the colors were selected taking into account the color palette that includes yellow, blue and white, this is because we consider that it is a combination that allows fluid navigation for the user, in addition to being a combination used in several online stores such as Amazon or Mercado Libre.

-Mockups:

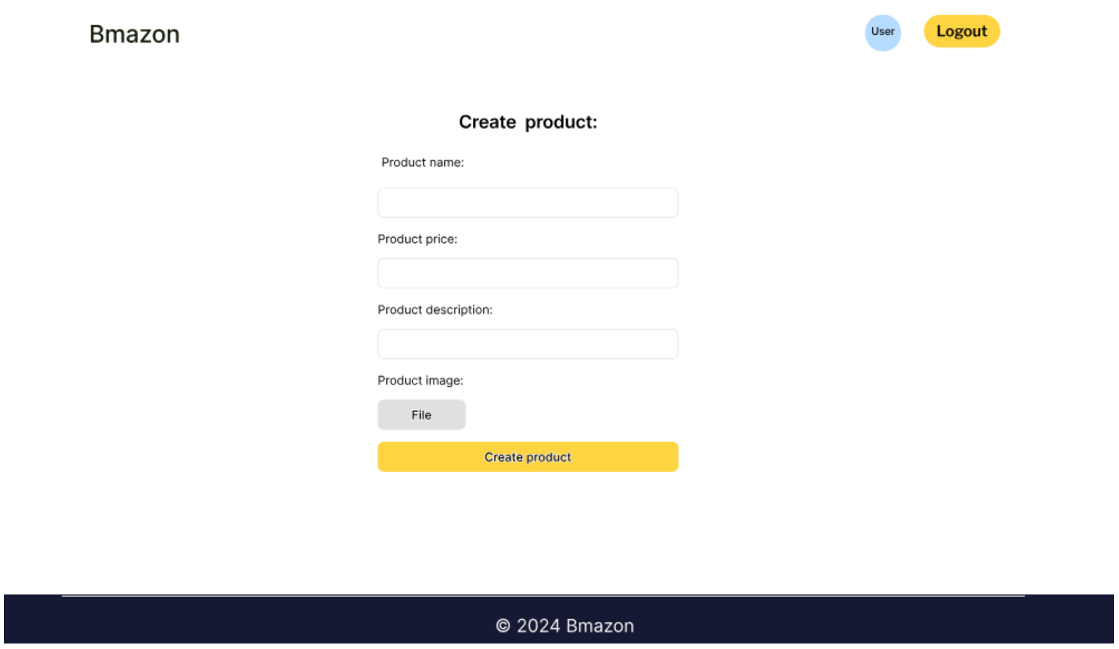
Mockup Log in: This mockup shows the login interface, in which the user enters their email and password and when they press the login button they are redirected to home.



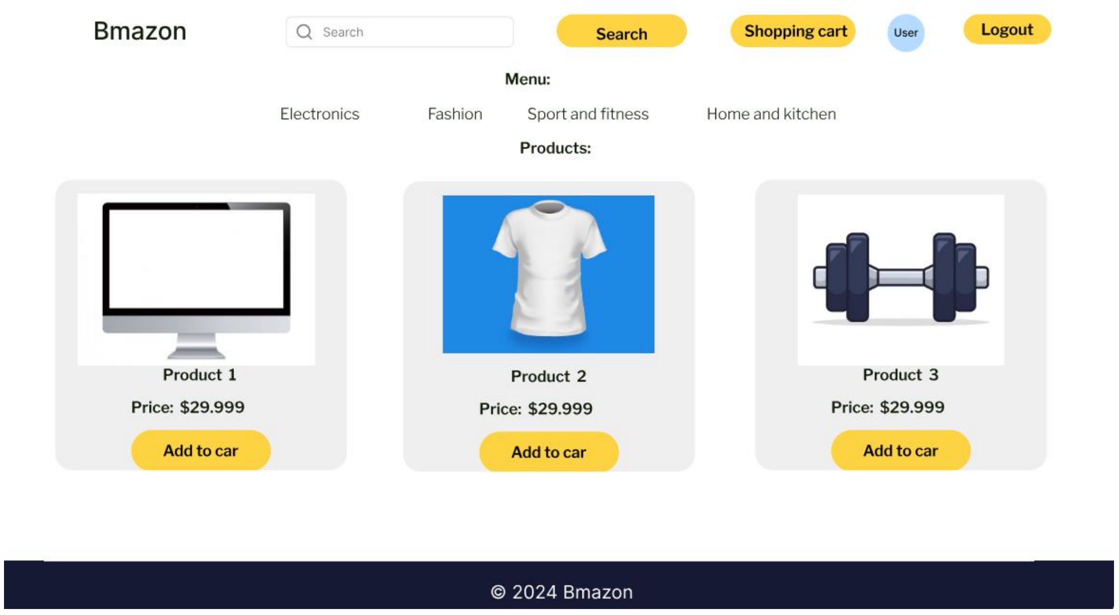
Mockup Create account: This mockup shows the interface in which the user registers and when pressing the create account button, the user is registered in the database.



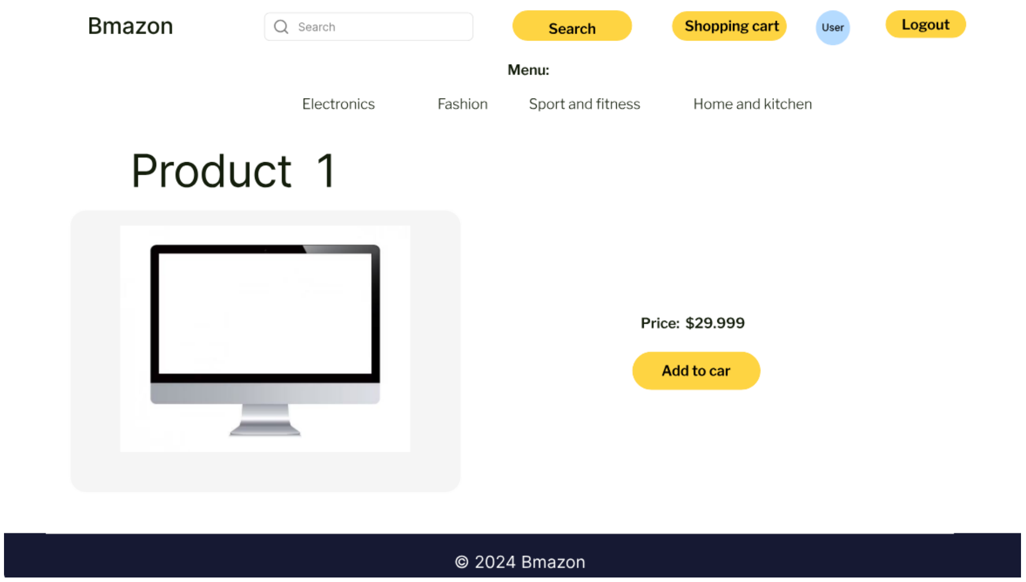
Mockup Create product: This mockup shows the interface in which the admin creates the product by filling out the form and pressing the create account button.



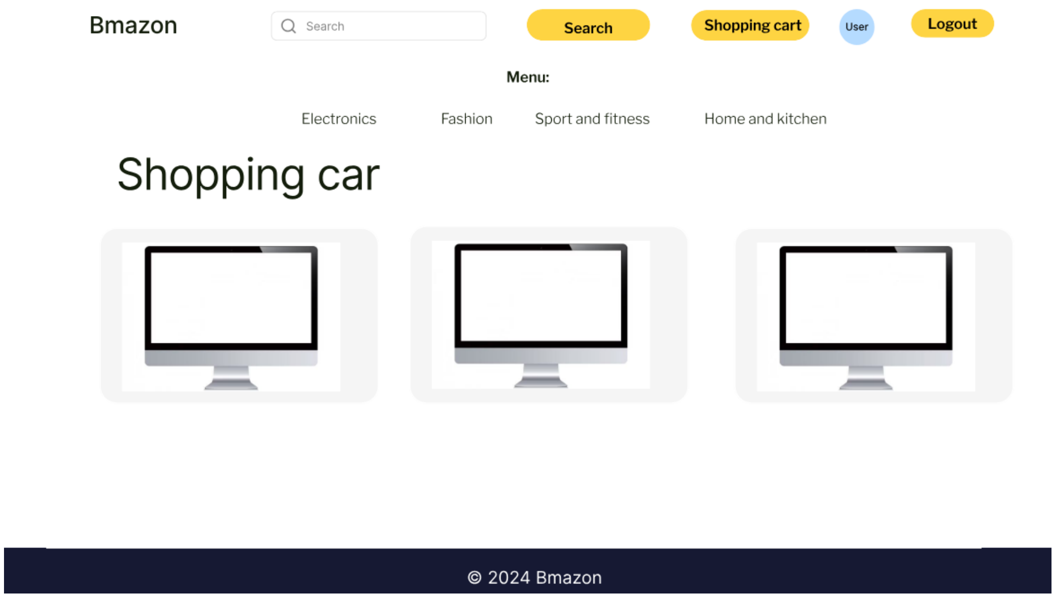
Mockup home: In this interface all the products housed in the store are shown, and at the same time you have the possibility of choosing a product, entering the shopping cart, searching for a product or logging out.



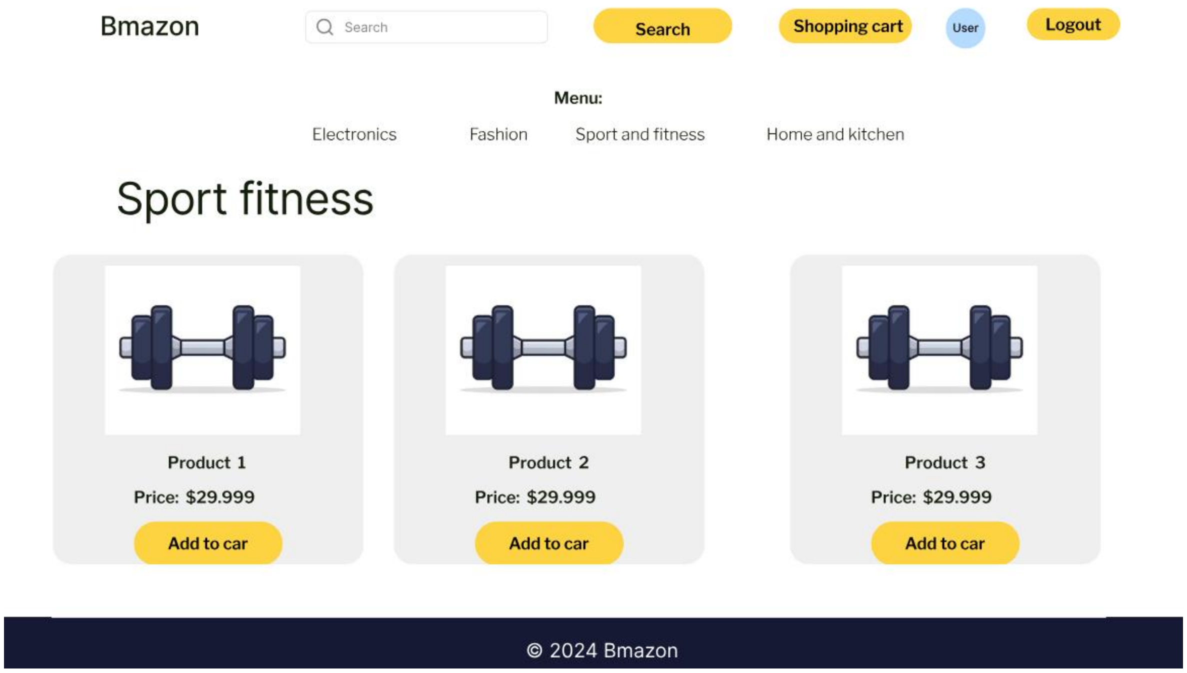
Mockup of a product: This mockup shows an example of how a product would be displayed in the store.



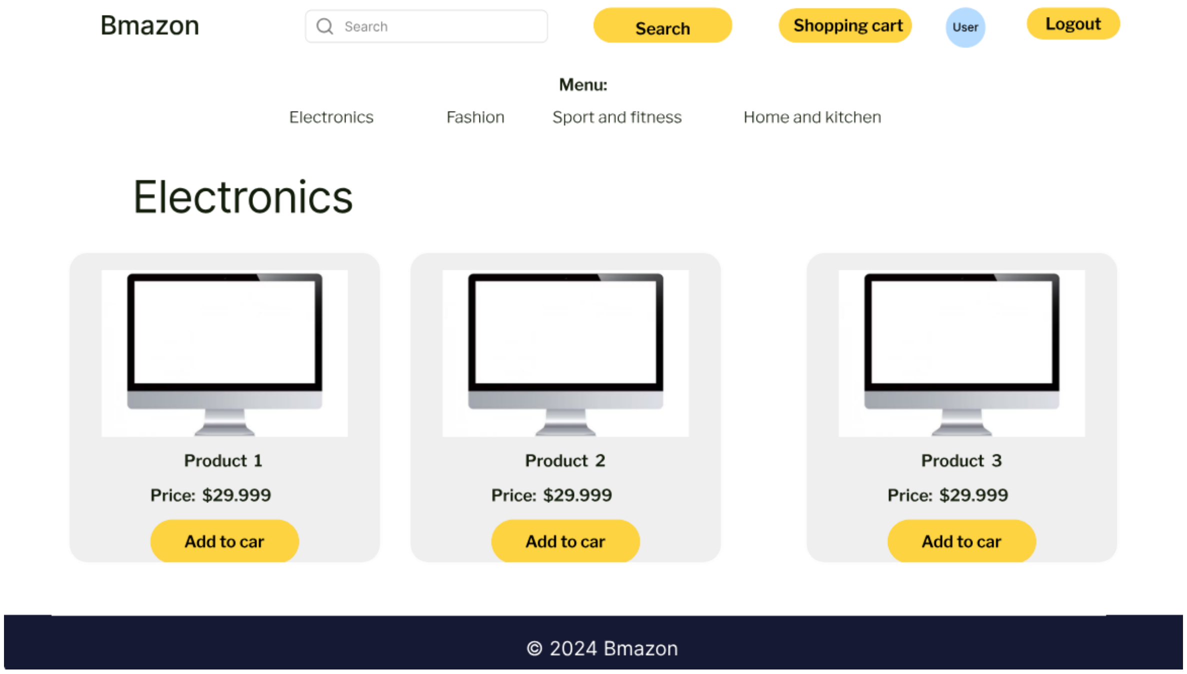
Mockup shopping cart section: This prototype shows the expected interface for the user's shopping cart.



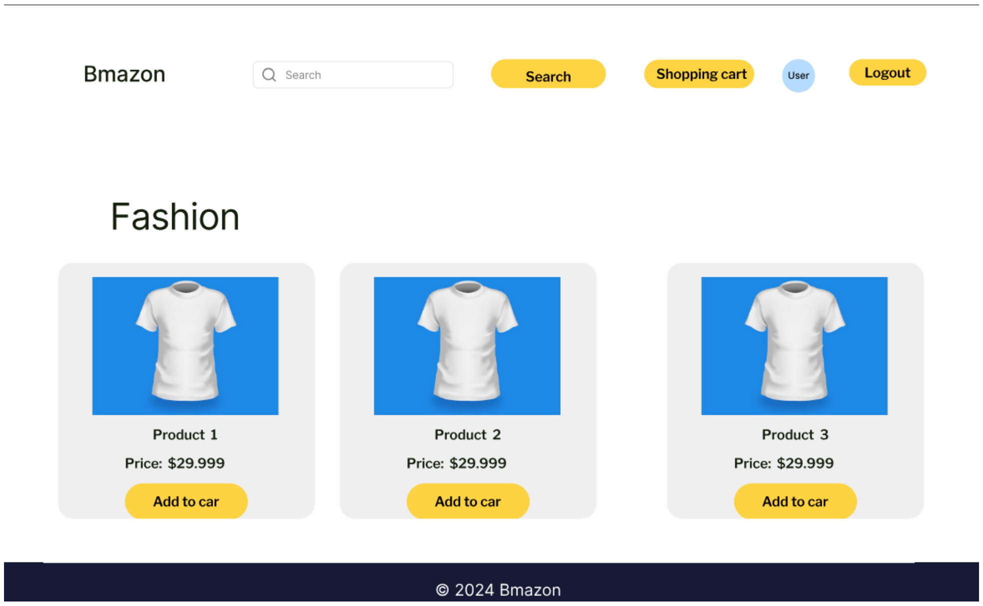
Sport fitness section mockup: This mockup shows the interface of the Sport-fitness product section.



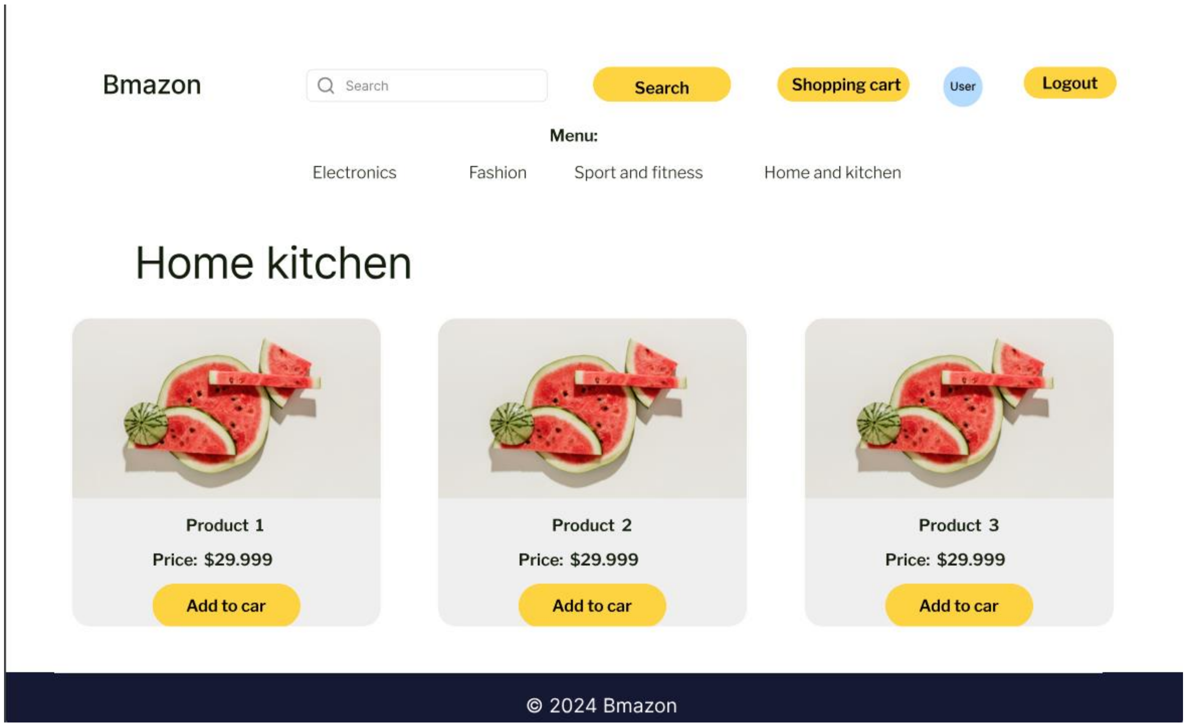
Electronics section mockup: This mockup shows the interface of the electronics products section.



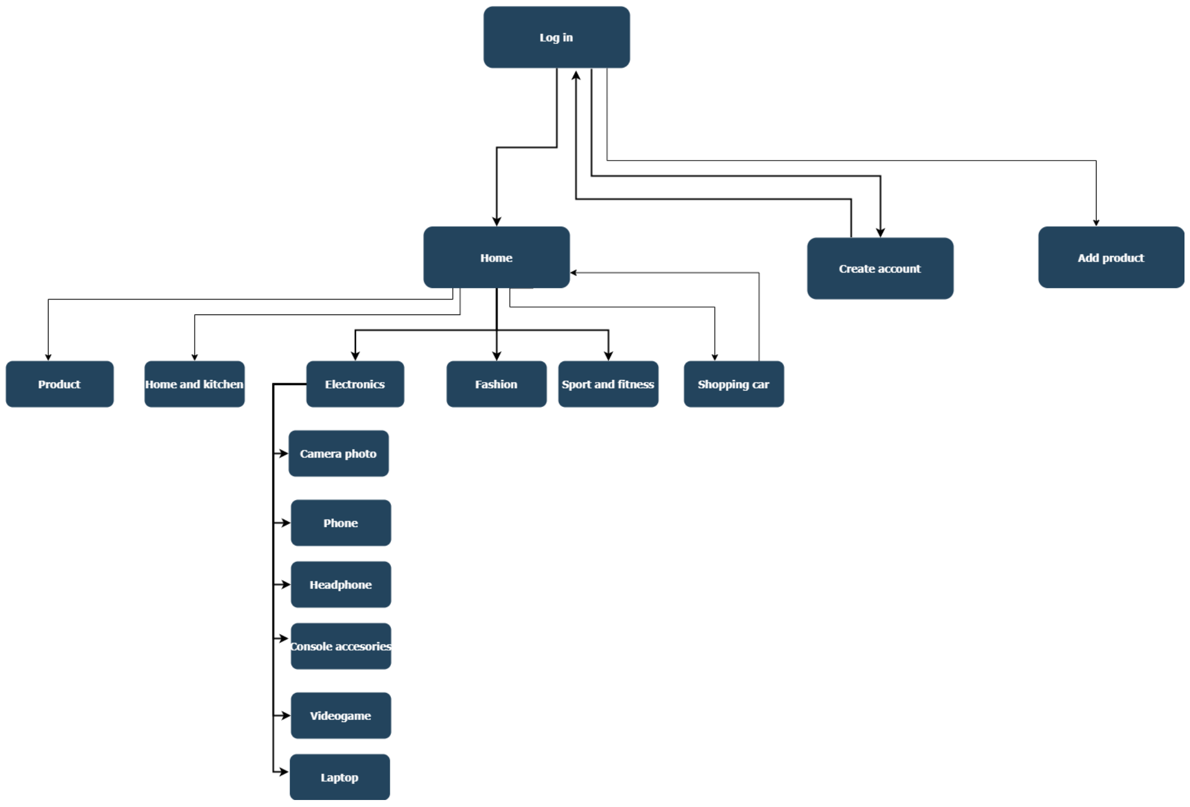
Fashion section mockup: This mockup shows the interface of the fashion product section.



Home and kitchen section mockup: This mockup shows the interface of the home and kitchen product section.



-Navigation map: Below is the navigation map between interfaces for the online store, as you can see the login interface gives way to the other interfaces discussed in the mockups.


Log in
Home
Create account
Add product
Product
Home and kitchen
Electronics
Fashion
Sport and fitness
Shopping car
Camera photo
Phone
Headphone
Console accesories
Videogame
Laptop