Swinburne University of Technology

School of Software and Electrical Engineering

ASSIGNMENT AND PROJECT COVER SHEET

Subject Code: SWE30003 Unit Title: Software Architectures and Design

Assignment number and title: 3, Design Implementation Due date: [redacted]

Tutorial Day and time: ____________________ Original Project Group: ________

Lecturer: [redacted] [redacted]

To be completed as this is an individual assignment
I declare that this is an individual assignment and that no part of this submission has been copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part been written for us by another person.

Student ID	Name	Signature
[redacted]	[redacted]	[redacted]

Marker's comments:

Total Mark: ________________

Extension certification:

This assignment has been given an extension and is now due on ________________________

Signature of Convener: ____________________

Assignment 2 Design Discussion:

1. **Good Aspects:**
    - Assumptions were made correctly and descriptive enough to help simplify the project implementation.
    - Use of a control class (Menu class) helped abstracting the complex functional requirements of the system and made easy to implement.
    - Design Heuristics helped in maintaining the code clean enough to debug errors.
    - Even though some of bootstrap processes required change it was still really helpful in direct implementation.

2. **Missing Aspects:**
    - Firstly, very important thing that was missing was Product Class and its presence in the UML.
    - Another thing that was missing was the Shopping Cart Class. It wasn't that much important though for the implementation of Shopping Cart feature it was required.
    - Database design was also missing.

3. **Flawed Aspects:**
    - There was some ambiguity in the UML diagram some relations were not clear.
    - Even though Customer Class and Administrator class inherited form the parent User Class both the classes had duplicate members that could have been inherited from the User class.
    - ProductOrders and ProductOrder classis's function was somewhat ambiguous and not seemed write to put those functions separately during implementation.
    - SalesRecord class was not really required its functionality can be implement just by using ProductOrder Class.
    - Some part of Bootstrap Processes became wrong due to absence of Product and Cart class and also due to removal of ProductOrders and SalesRecord class.

4. **Level of Interpretation Required:**
   It required more time than expected to come up with the final implementation. Preparation of Detailed Design helped a lot figure the flaws the and correct the mistakes in the designs. Because of some mistakes in initial design that were reflected in Detailed Design, it required a few changes to detailed design during implementation.

Lessons Learnt from Assignment 2

- After preparing a complete initial Design take some break and checking the Design after some time helps figuring out potential flaws in the design.
- Always try to correlate the design with existing system or real word object to figure out the relations correctly.
- Do not miss any Entity from the system because it can lead to more and more errors in the Design and may have to design the system completely from scratch.
- Always consider whether the function or feature we are designing can be implemented or not.
- A good understanding of Modeling language is also important before trying to design a system.

Detailed Design

1. Introduction

This report aims to act as an introduction to the Design and Development Stage of this project for All Electronics Store. As a development company, we are required to provide an initial business specification report, which was responsible for laying out the client's non functional requirements.
In this report, we will delve into more depth onto the technical approach that will be used to develop this final project. From the technology that will be used, frameworks, UML diagrams, classes, and interfaces. Project structure, software architecture, and design patterns (SOLID, MVVM & DRY Principle) will also be discussed.
This document then is aimed for internal use between developers and technical stakeholders.

1.1: Domain Vocabulary:

Since this design is domain driven, it means that there are certain terms and definitions which have to comprise of both functional and non-functional requirements, that can be understood by the customer and the developer.
Below is a list of definitions, that will be used throughout this report, as a way to abstract complex concepts, and to keep this report user friendly:

- **AES** - All Your Electronics, the organization that hired SwinSoft for this Job
- **Customer** -any member of the public that uses the online store to make a purchase
- **Administrator** - a member of AES, that has root access to the site, and can generate reports, and issue new products to the Buyers
- **Merchandise** - products (electronics) that buyers purchase through the online store
- **Deliverable** - a component that forms part of the overall project, and represents a major milestone
- **Services** - a bit more technical, but these form components that the online store will communicate with to manage emails, receive notifications and make payments. All these services will be managed by a common backend
- **Backend -** a common api gateway, that manages the entire system, from services, to pushing and receiving data from the clients
- **Clients** - Any application or website (could be mobile or browser based) that consumes (calls the backend api) and allows the Customers to use the online store. This effectively is the product (it is the online store that Customers use to browse merchandise and manage their accounts)

1.2: Outlook of the Solution:

The system, will comprise of multiple components:

1. **Web Client (Browser Application):** This is the application that customers will interact with
2. **Backend of the System:** Manages all the requests that the web application will send, such as when users generate new memberships or purchase new products. Backend

will then communicate with microservices, which are each responsible for a specific domain.

1.3: Tradeoffs on Design:

User Interface:

- Multiple Forms will exist, which each are responsible for allowing users to make payments and create subscriptions. Each form has to have a maximum field length, and must be verified on the client and the server. This means that there must be extra work for both ends of the system.
- Upon Invalid passwords detected on the form, the user will need to be prompted with an alert, notifying them that their password is not correct, and needs to be validated again.

Backend:

Identity Server must be integrated with the backend, to manage identity tokens. We cannot exchange raw and sensitive information within request headers. All user claims must be stored within Encrypted cookies that will be handled by the login manager, which will contain the user information. This info will then be related to a separate module for validating user identity and authentication. This project will not store user passwords or user information, for security reasons.
Identity Server Project: https://github.com/IdentityServer/IdentityServer4

Costs:

Hosting on the cloud (IaaS = Infrastructure as a Service) will cost money. For this project, we need three separate Virtual Machines (VM), a basic load balancer (Weighted Round Robin), 3 SQL Databases for the Microservices, and Docker Containers hosted inside each VM.
Total cost of maintaining this system, is outlined in the Microsoft pricing information below:
(Basic Load Balancers are Free)
- SQL Costs:
https://azure.microsoft.com/en-us/pricing/details/azure-sql-database/single/
- VM Costs:
https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/

1.4: Initial Assumptions:

For this project to be successful, we are assuming the following points:

- AES has enough funds to pay for the cloud infrastructure costs, to maintain and scale their system
- Microsoft will not change their pricing to dramatically, which will prevent us from having to migrate to a new provider

1.5: Guidelines for the Code Base:

The following design patterns and conventions are to be maintained throughout development of this application:

1. All Source Code is to be properly documented and commented on.

2. All Classes are to be given appropriate names describing their roles, Single Responsibility and DRY Principle is to be followed to keep source code clean and maintainable
3. Variable names are to be kept meaningful and be provided with the appropriate access modifiers for data encapsulation.

2. Problem Analysis:

In the previous report, we created object design for the system. In this section we will refine and extend the initial high-level object-oriented design from assignment 2, to arrive at a detailed design suitable for direct object-oriented implementation.

Key functionality of this system include:

1. Ability to Create Memberships and Orders
    - Users should be able to sign up and authenticate via the web client, and store
    - Users should be able to make orders and purchase new products, and go through a standard checkout store workflow, while viewing all the products available for purchase
    - Users should be able to purchase subscriptions to gain access to premium features

2. List and create customer receipts and invoices, cancel purchases and create sales reports

3. Relay purchase orders to the delivery company, to make deliveries to the customers in a timely manner

4. Customer page, where each customer can manage their own profiles, and change their details (address delivery information, name, etc.).

5. Users should be able to cancel their subscription

6. Administrators (AES Members) need to be able to create promotions and promotional products that will be sent to the customers for advertising.

2.1: Assumptions:

Below are the list of assumptions which will guide each of these key requirements:

1. Users will only purchase electronic products
2. AES already has a delivery company available, a warehouse, and an API the backend can consume, to relay purchase orders too.
3. All Products must be given a unique ID, a timestamp of creation, manufacturer ID, and other metadata used to identify each product.
4. If a customer is not a member, they will be relayed to the login screen to make an account, before being allowed to make a purchase

5. Users can always remove their details from the system if they want, and no accounts are disabled if they are removed. This ensures that customer information is never leaked, and maintains compliance with Data Protection Laws.

2.2: Simplifications:

Based on the above problem analysis, and initial assumptions for this project, below are some simplifications that have allowed us to minify (optimize) the project and development time:

1. Each Microservice will have its own database, and at most have 4 tables. A one-to-many relationship will exist in each domain, to make sure that Entity Framework only has to query against a single model, instead of multiple models.
2. To optimize costs we will use only a basic load balancer (which is free) and with minimal storage on the VM and database
3. Each microservice will contain at most 3 modules and follow the Onion Architecture (Domain Driven Design), which will reduce the coupling between the core and the consumers.
4. Each microservice will also contain unit test projects, to simplify testing during the development phase of this project, to avoid having to debug against the API Gateway.

3. Candidate Classes

3.1 Candidate class list

- Menu
- User
    1. Customer
    2. Administrator
- Product
- ProductOrder
- Cart
- Payment
    1. Card
    2. PayPal

3.2 UML Diagram

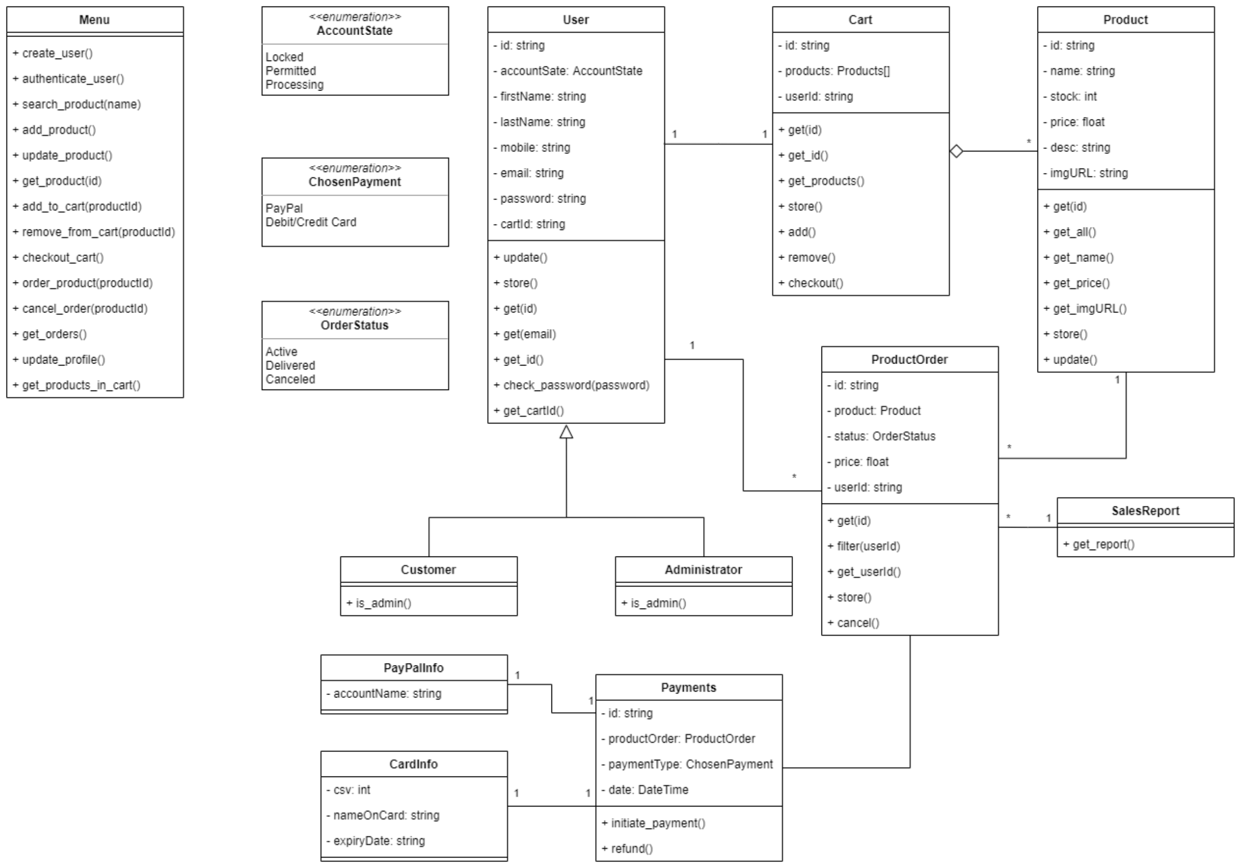


Fig.3.1 - UML Diagram

3.3 CRC Cards

3.3.1 Menu

Class Name: Menu **Super Class:** -	
The menu call will be like the control class for the whole system, from which every aspect of the system can be controlled.	
Responsibilities	**Collaborators**
Creates and initializes process for other classes according to user input	User Class, Customer Class, Administrator Class
Knows a customer’s sales record	ProductOrder Class
Can update database about customer page based on user input	Database Context (Relays all updates to the database through a single transaction)

3.3.2 User

Class Name: User **Super Class:** -	
A User class is an abstraction for any user of the website be it a customer or administrator or any other user. It stores all the info about a user. It refers to a unique user each time and manages user information in the database.	
Responsibilities	**Collaborators**
Creates and initializes process for other classes	Customer, Administrator

3.3.3 Customer

Class Name: Customer **Super Class:** - User	
The customer class stores all the info about a customer that All your Electronics website wants to store. It refers to a unique customer each time and manages customer information in the database.	
Responsibilities	**Collaborators**
Interacts with the customer input	Graphical User interface of the website
Knows all required customer details i.e., name, phone number, email, address, preferred payment methods etc.	NA
Can modify and update customer data in the database	

3.3.4 Administrator

Class Name: Administrator **Super Class:** - User
The administrator class is for users of All your Electronics website who have the administrative privileges and logins to access the website. It refers to a unique administrator if there are more than one administrator each time and manages their information in the database.

Responsibilities	**Collaborators**
Interacts with the administrator input	Graphical User interface of the website
Knows all required administrator details i.e., name, email etc.	NA
Can manage and create new promotions for the website	Admin Accounts

3.3.5 Product

Class Name: Product **Super Class:** -	
Product class stores all the info about a product. It refers to a unique product each time and manages product information in the database.	
Responsibilities	**Collaborators**
Interacts with the administrator input	Graphical User interface of the website
Knows all required product details i.e., name, price etc.	NA
Can modify and update product data in the database	Database Provider

3.3.5 ProductOrder

Class Name: ProductOrder **Super Class:** -	
ProductOrder class provides the ability to update the Order details and database if the need arises to cancel the order or cancel the subscription. This class process customers' orders and initialises the payment process	
Responsibilities	**Collaborators**
Can access information from the database for current customer product orders	Customer
Can send cancellation info to delivery services or partners	Payment Gateway

Can update the database in case of a subscription cancellation.	Stored Procedures and SQL Jobs (Being executed every day, to check for records that need to be changed).
Knows ordered product info such as product quantity, name, price etc.	Database Provider
Can create and initializes the Payment class by joining the product information from database	Payment

3.3.6 Payment

Class Name: Payment **Super Class:** -	
Payment class processes the customer order payment process and passes the order information to ProductOrder class.	
Responsibilities	**Collaborators**
Can provide customers with the payment options such as PayPal, Cards	PayPalInfo, CardInfo
Knows information such as customer payment information entered by the customer and if there are more than one method knows the customers preferred one.	Payment Gateway

3.3.7 Card

Class Name: Card **Super Class:** -	
CardInfo is a subclass of Payment. It allows the customer to pay by card.	
Responsibilities	**Collaborators**
Stores and Knows the customer entered details of one or multiple cards	Payment Gateway (Stripe), which will manage a consumable payment or a subscription for the customer. Receipts and Transaction IDs need to be kept for future reference in case there is an issue with the customer’s card

3.3.8 PayPal

Class Name: PayPal **Super Class:** -	
PayPalInfo is a subclass of Payment. It allows the customer to pay by PayPal.	
Responsibilities	**Collaborators**
Stores and Knows the customer entered details their PayPal account.	

4. Design Quality

Design patterns are often used in Object-oriented programming languages. They can cause recurring problems when we design and implement the software, resulting in improving the readability of source codes. There are three main categories of design pattern: Creational, Structure and Behavior.

4.1: Design Heuristic:

Based on ‘Object-Oriented Design Heuristic’ (Riel 1996), there are Heuristics used in this section to improve the implementation and object-oriented design.

- **H1:** Do not create any God classes/ objects or super classes in the design. God class has some weakness, its strong coupling with other codes, so it's hard to maintain/debug. In the design, there are many different classes with different responsibilities so God class should be avoided.
- **H2:** Only one key abstraction should be captured by a class. There are many classes in the system and each class should have a unique role. This helps to keep the source code clean and easy to follow.
- **H3:** Do not create a class for a single operation. Based on the previous Heuristic, each class has a unique role, responsibilities and capabilities but not a single operation. All the class will work together to perform a number of operations at a time.
- **H4:** All the abstract classes should be defined as base classes.
- **H5:** All the base classes should be defined as abstract classes.
- **H6:** The base should not be able to have knowledge about its derived classes, but the derived classes can have knowledge about its base class. For example, we have an User base class and Customer as derived class. The User class doesn’t know anything about the extended characteristics of the Member class. While Member class is implemented the requirement from the User class
- **H7:** Create a consistent design of programming language, actions and diagrams. This helps the source code easier to understand and maintain in the future. As we mentioned in section 1.5 of this report, there is a guideline which is used for the source code to keep the consistency.

- **H8:** Derived classes with methods that do nothing, can't override the base classes.
- **H9:** Avoid collaborating between classes in the system.

Misassigning and unidentified responsibilities for classes can happen with too many collaborations in the system.

4.2 Creational Pattern:

Factory Method Pattern:

Using this pattern to increase the flexibility and reusability of the design. Objects in the code are created without specifying class to instantiate. We can create different classes with different roles but sharing some commonalities. Factory pattern helps the extension, maintenance and The update process becomes easy and convenient.
In this design, we apply the factory method pattern to create two subclasses: Customer and Admin. They can share the common interface and not impact the logic of the system. Members and Admin are the users who can use the AES to shop online and interact with the software. This pattern can be used to manage order and generate receipts for the customer when they shop online on the AES store.
Factory method pattern can cooperate with Strategy pattern and Observer pattern which will later be discussed in the below section. Strategy pattern is applied to create two subclasses of payment methods. They have different roles and responsibilities but use the same interface where users can select alternative payment methods. While the observer pattern is used to send the promotions, discount to the customer.

4.3 Structure Pattern:

Model - View - Controller pattern:

AES is an online store with many pageviews of products and categories. We need to split the workload into smaller modules to have a better design and easier programming. Model - View - Controller pattern provides front-end and back-end for data processing components, database and user interface. However, the development process may require a lot of time.
User will interact with "View" to input their request; the requested data is sent to "Controller" for processing. Then the data will be connected with the database to save or receive information.

4.4 Behavior Pattern:

Strategy pattern:

In this design, we use the Strategy pattern as the behavior design pattern. This pattern is the perfect solution for an online store with a payment system. Instead of implementing all the payment method algorithms into one class, it splits them into a family of algorithms to use. For example in our system, we have a base Payment class and two classes (CardInfo and PaypalInfo) that are implemented from it. Each of the derived classes will respond differently. When the new payment methods are required to be updated in the future, new classes are created so we don't have to modify existing derived classes and impact the system.

Observer pattern:

Observer pattern can maintain a list of its dependencies, notify and then automatically update them. For the AES system, we use this pattern to send promotions, discounts and notifications to subscripted users. At the later stage of development and when the requirements are changed, we can use this pattern to manage product details, product price, update inventories from the administration interface.

5. Bootstrap Process

5.1: User Registration:

This bootstrapping process shows the action of a customer registering with the service:
Main creates Menu Class.
Menu Class creates Customer class.
Customer Class takes user input.
Customer Class stores user information in the database.

5.2: Authenticate User:

This bootstrapping process shows the action of a user authenticating with the service:
Main creates Menu class.
Menu Class fetches the User from database.
User Class it created based on data from database.
User Class checks the entered password.
If password matches user is authenticated.

5.3: Search Product:

This bootstrapping process shows the action of a user searching for product on store:
Main creates Menu class.
Menu Class fetches all the Product data from database.
Menu Class filters the product data base on user input.

5.4: Add Product:

This bootstrapping process shows the action of a admin adding new product:
Main creates Menu class.
Menu class creates Product object based on input.
Product object stores product information in the database.

5.5: Update Product:

This bootstrapping process shows the action of a admin update the information of an existing product:
Main creates Menu.
Menu fetches Product data from database.
Product Class creates Product object.
Product object updates the product information based on input.
Product object stores the product information in the database.

5.6: Add Product to Cart:

This bootstrapping process shows the action of a customer adding product in cart:
Main creates Menu.
Menu fetches cart details from database.
Cart class creates Cart object from fetched data.
Cart object adds the product with provided product ID.
Cart object store the cart information in the database.

5.7: Remove Product from Cart:

This bootstrapping process shows the action of a customer removing product from cart:
Main creates Menu.
Menu fetches cart details from database.
Cart class creates Cart object from fetched data.
Cart object removes the product with provided product ID.
Cart object store the cart information in the database.

5.8: Checkout Cart:

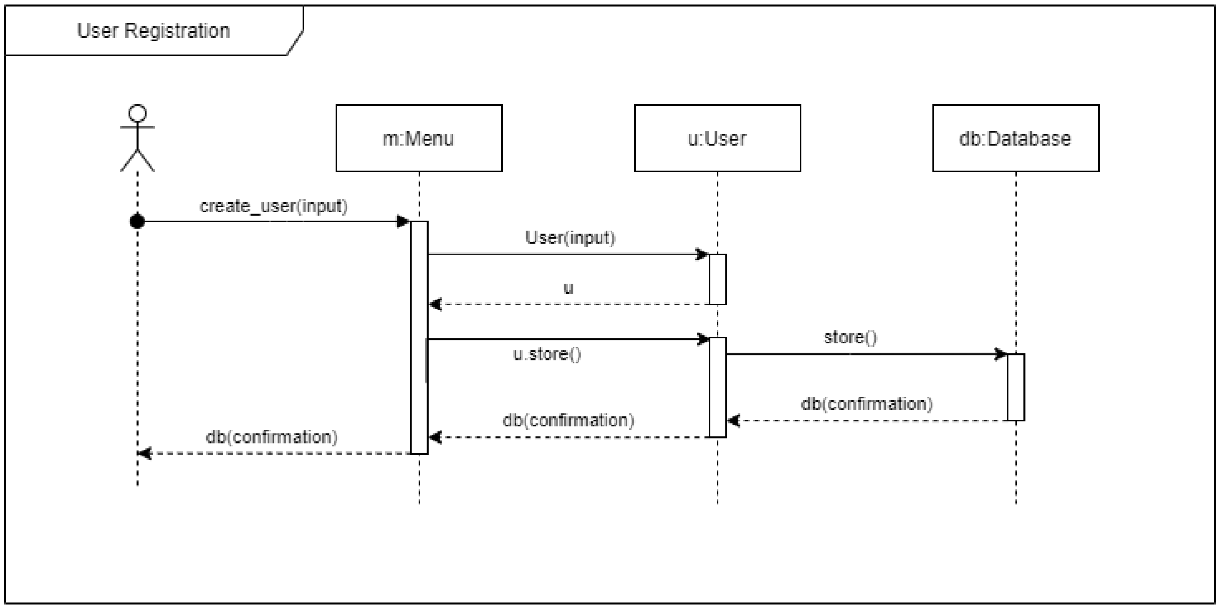
This bootstrapping process shows the action of a customer purchasing products present in their car:
Main creates Menu Class.
Menu fetches cart details from database.
Cart class creates Cart object from fetched data.
Cart class creates ProductOrder object for the products in cart and removes the product in cart.
ProductOrder creates Payment Class based on combined product/s information from the database.
Payment class creates payment options, stores selected options in database.
Payment Class creates SalesRecord Class based on Customer Class, ProductOrder Class and Payment Class collective information.

5.9: Order Product:

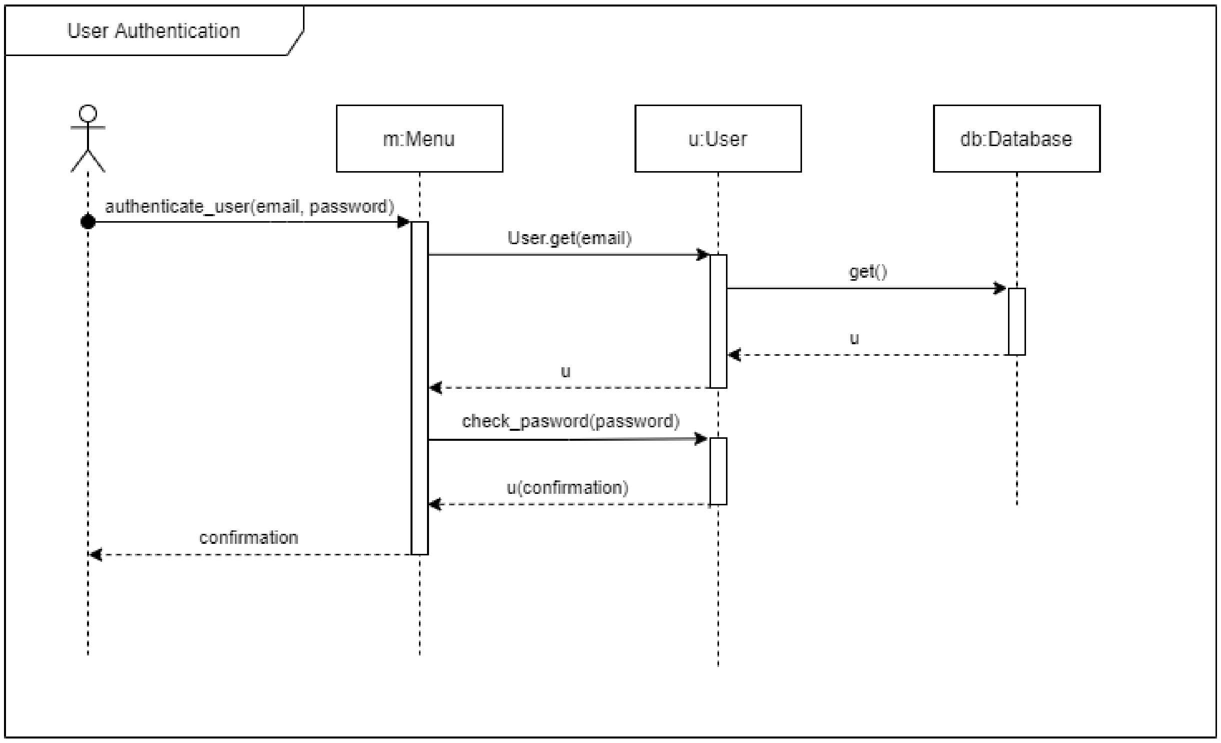
This bootstrapping process shows the action of a customer purchasing product:
Main creates Menu Class.
Menu fetches product details from database.
Product class creates Product object from fetched data.
Menu class creates ProductOrder object for the Product.
ProductOrder creates Payment Class based on combined product/s information from the database.
Payment class creates payment options, stores selected options in database.
Payment Class creates SalesRecord Class based on Customer Class, ProductOrder Class and Payment Class collective information.

6. Sequence Diagram

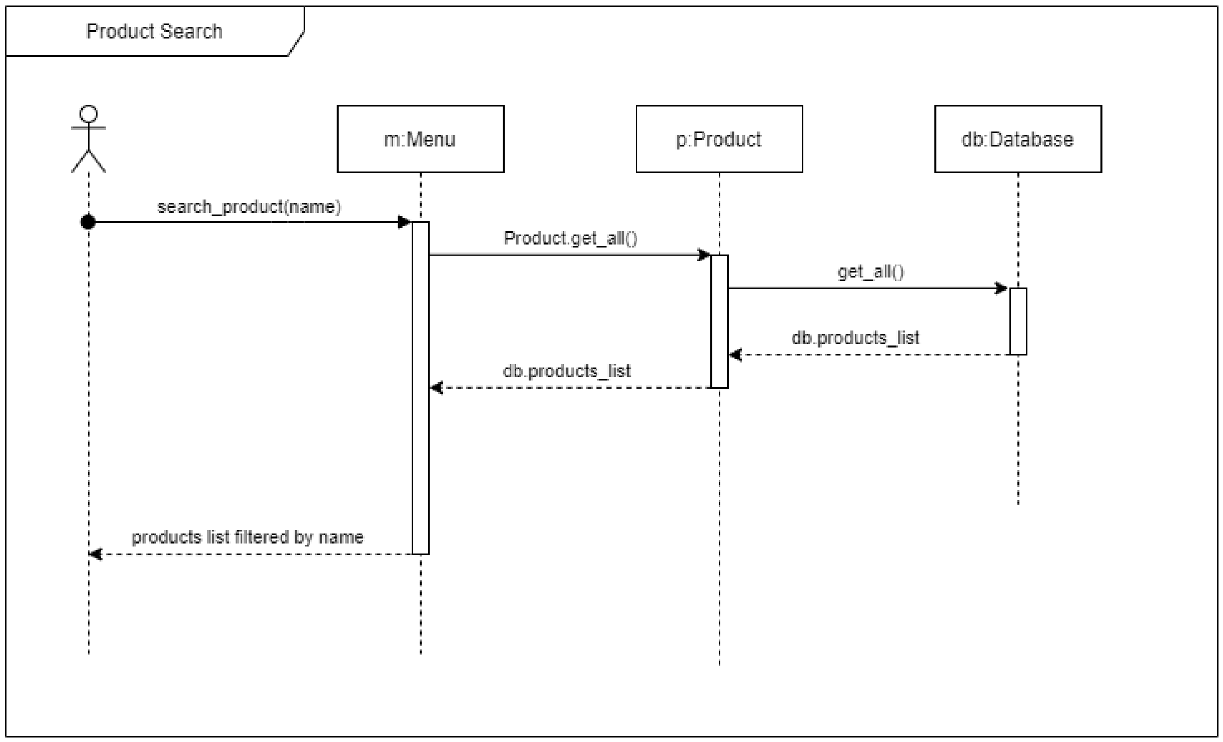
6.1: User Registration:



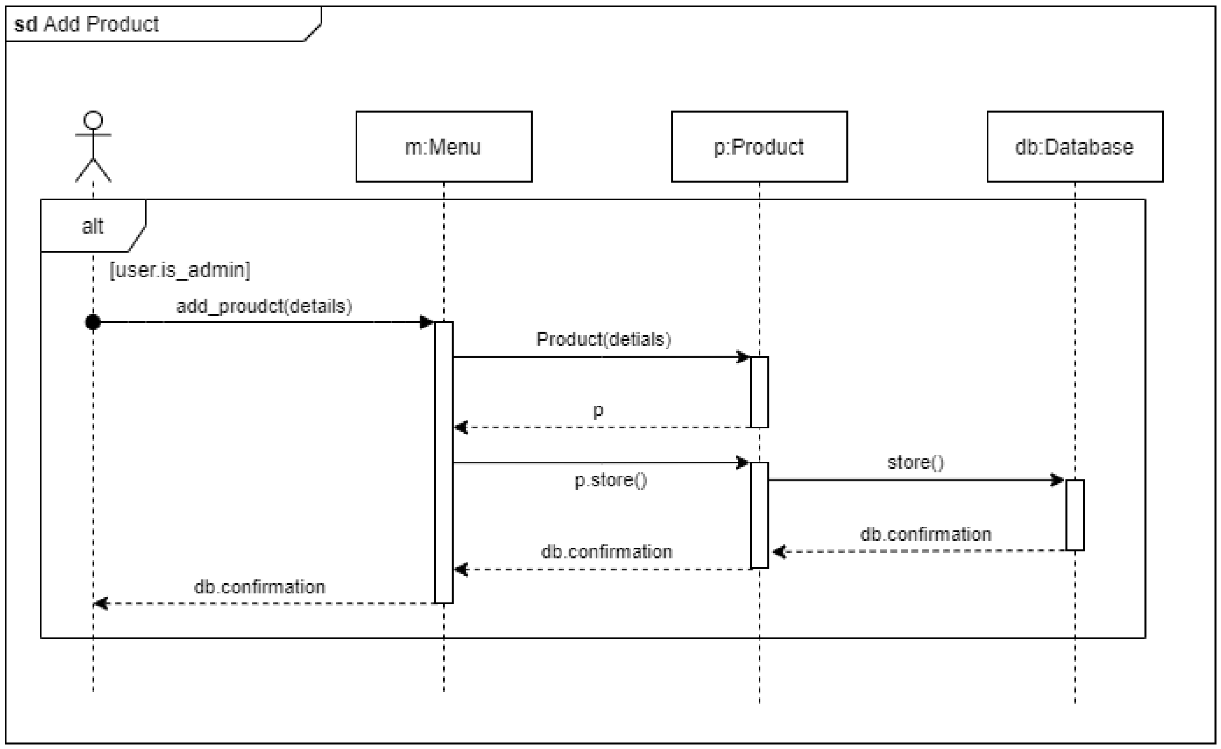
6.2: Authenticate User



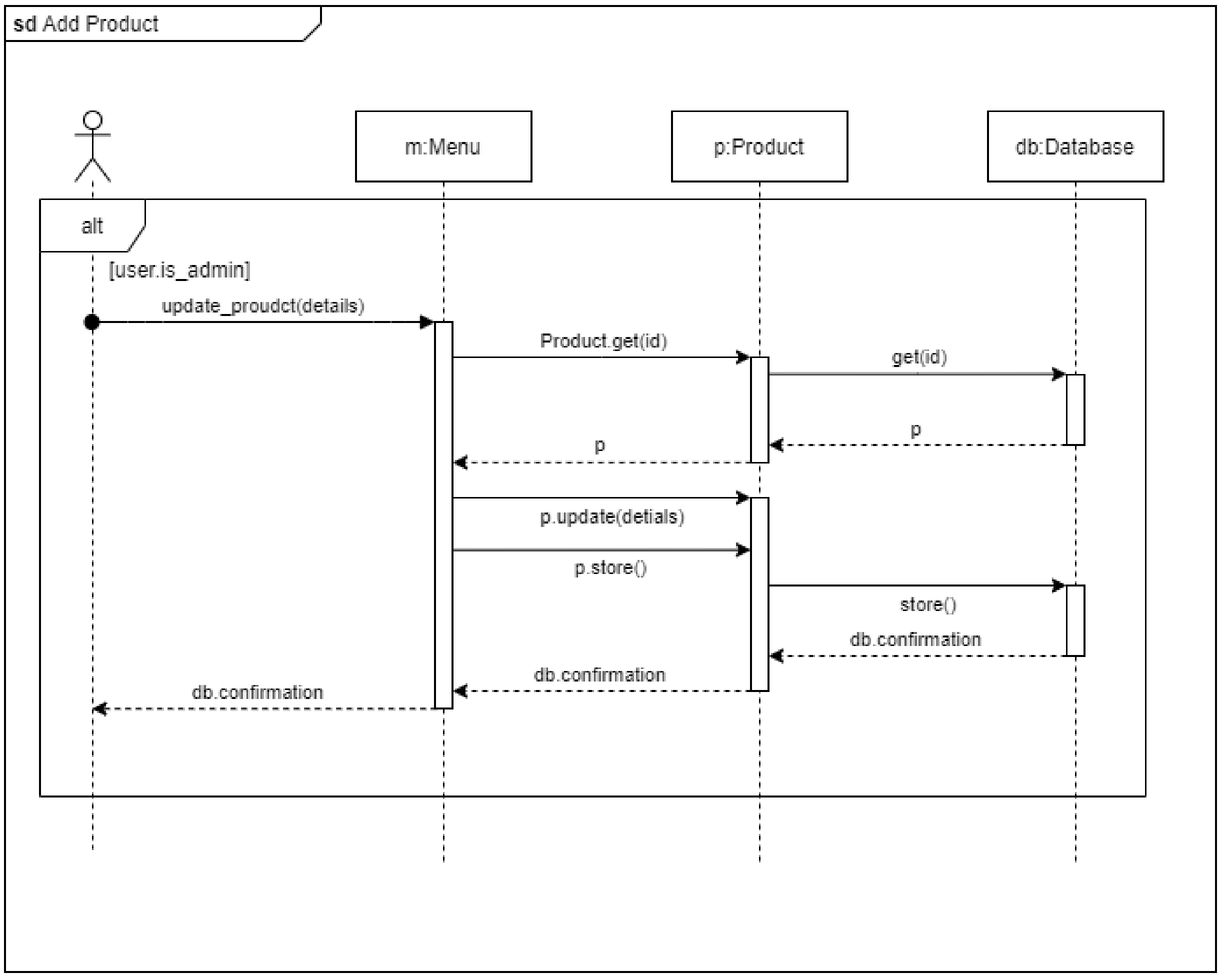
6.3: Search Product



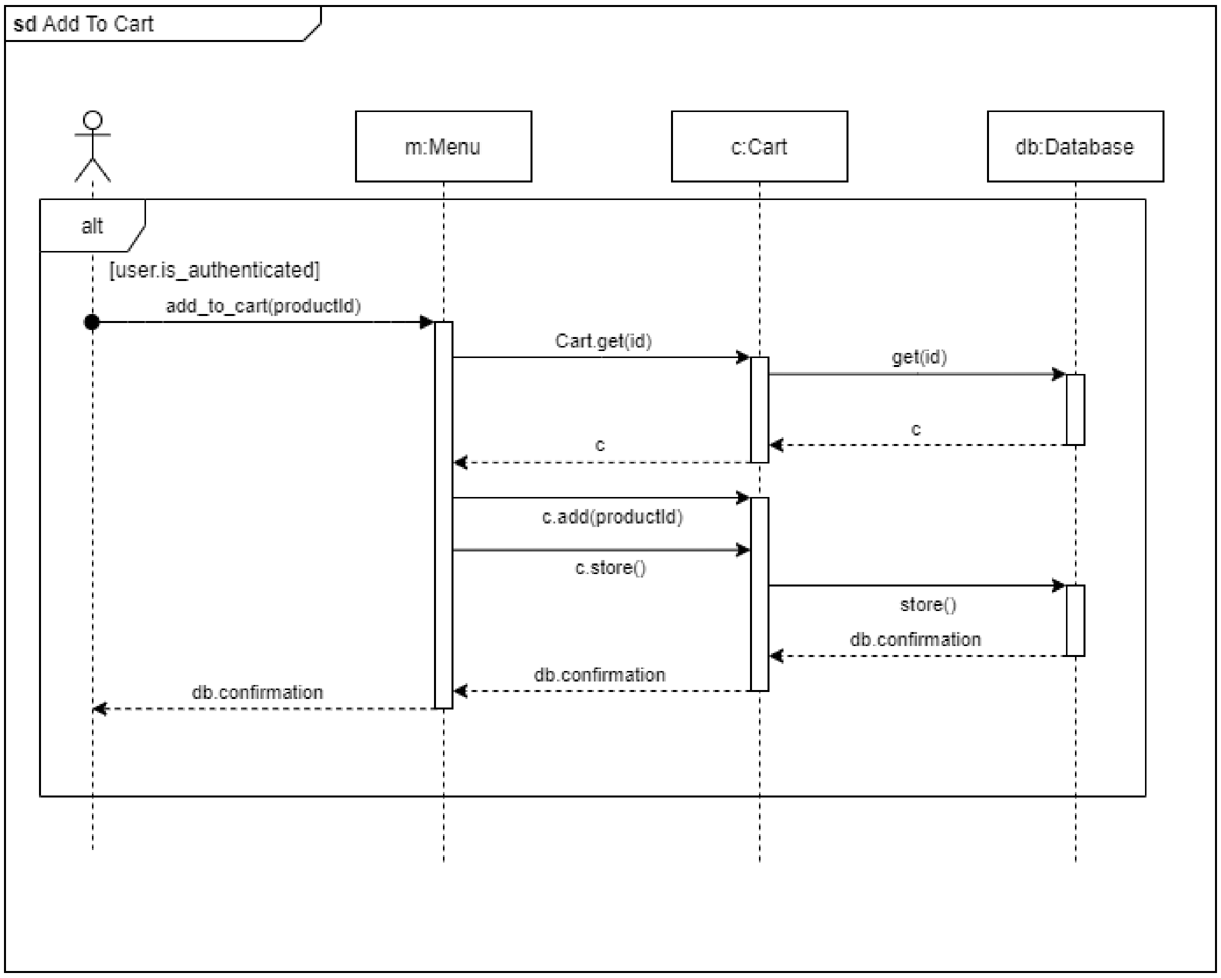
6.4: Add Product



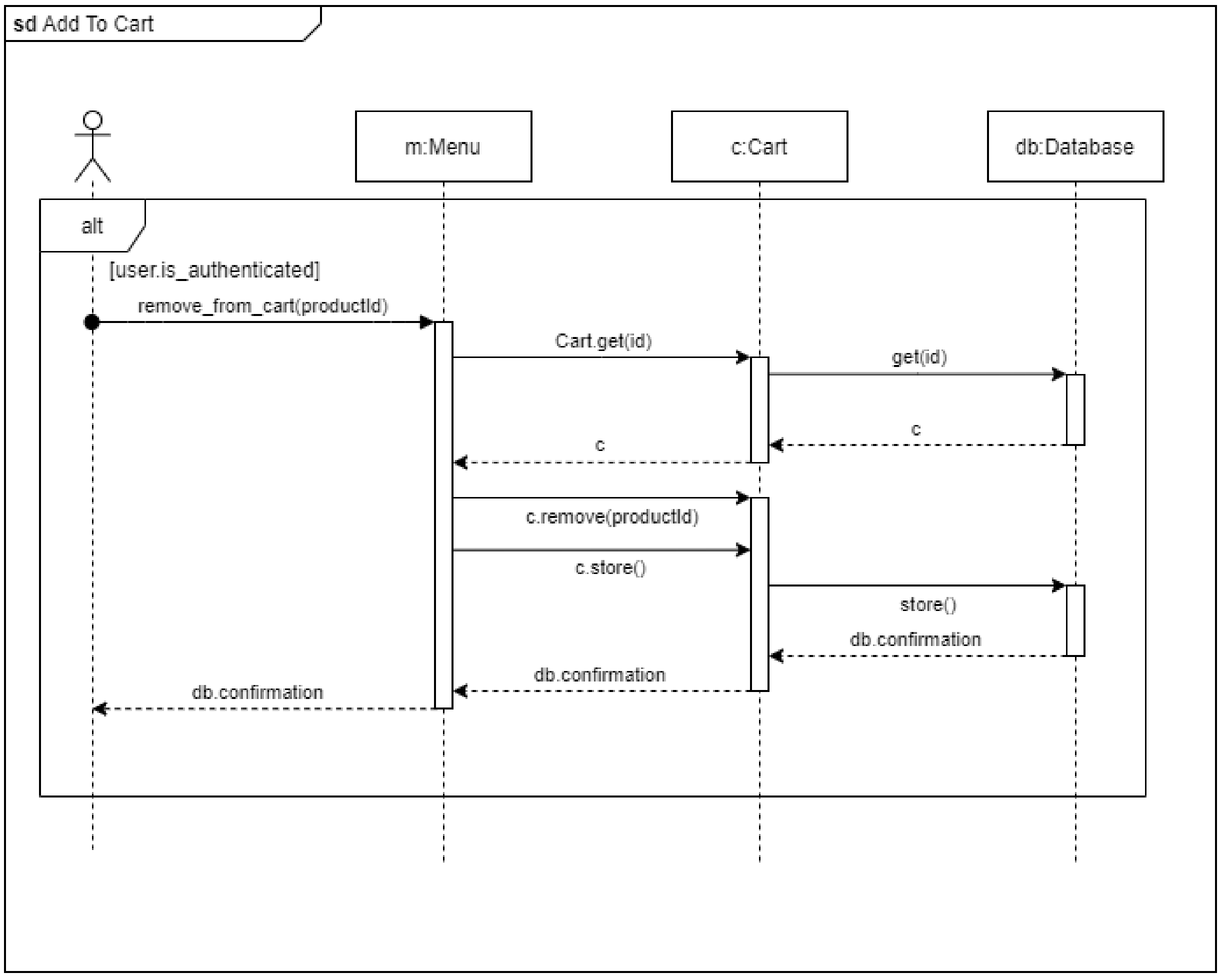
6.5: Update Product



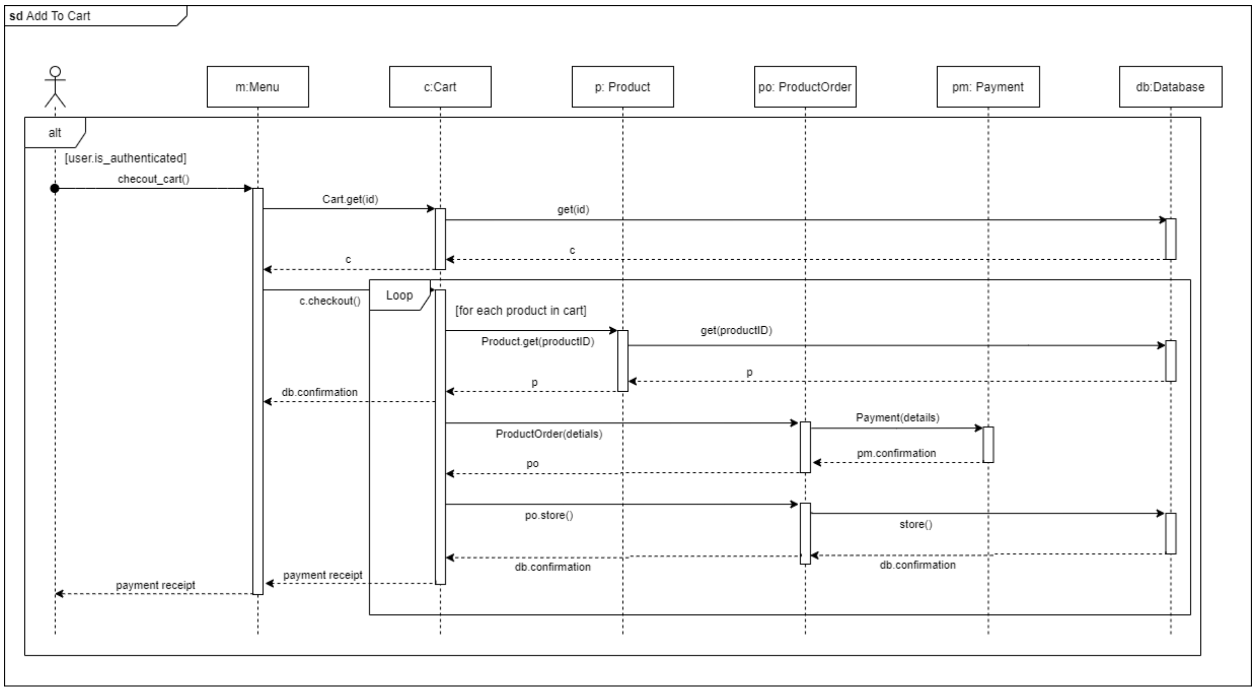
6.6: Add Product to Cart



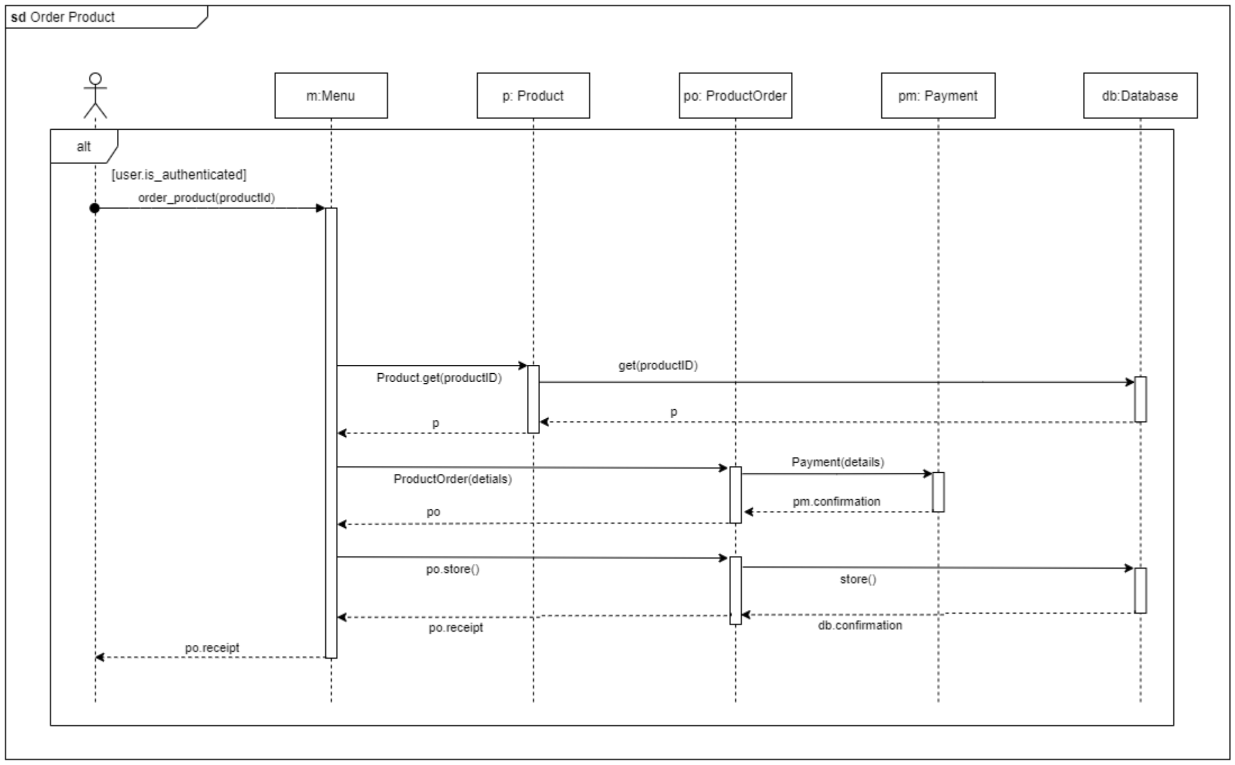
6.7: Remove Product from Cart



6.8: Checkout Cart



6.9: Order Product



7. Database Design

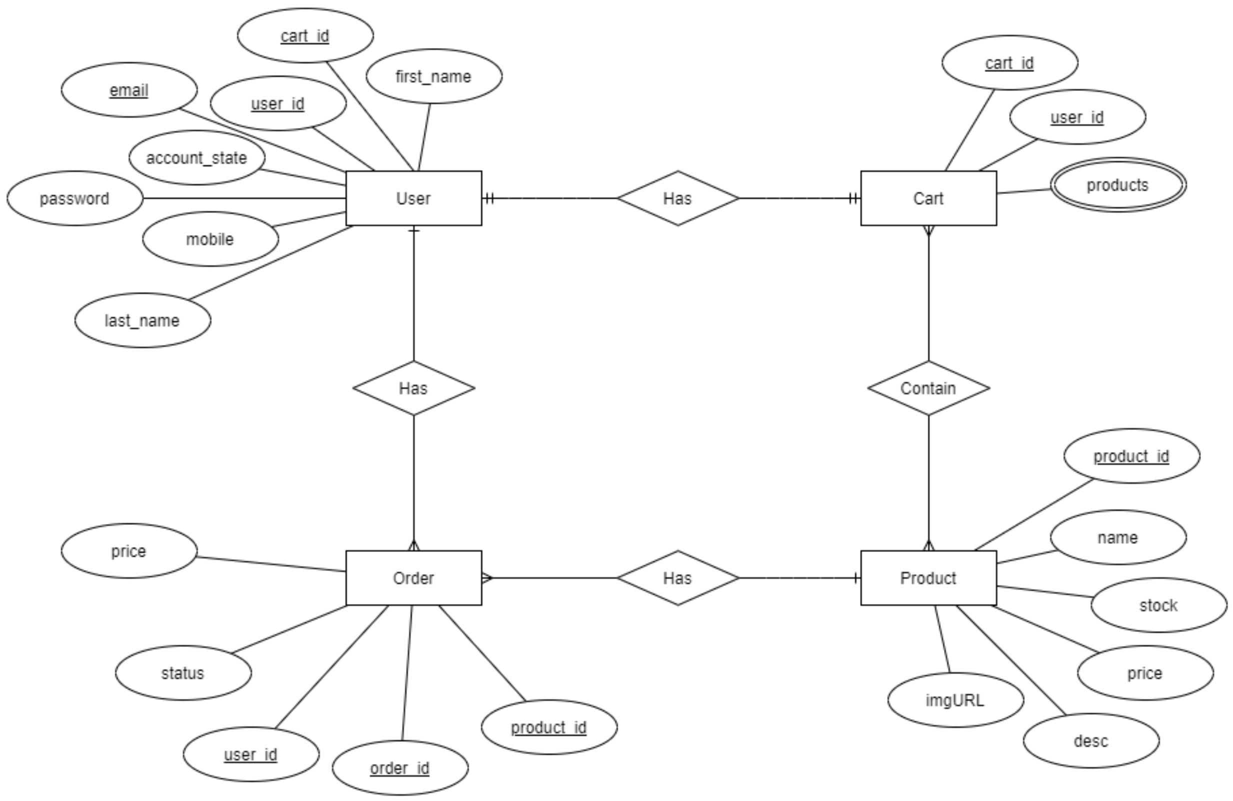


Fig.7.1: ER diagram

Execution and operation

1. Setting up Environment

a. Download and install python3.9 from https://www.python.org/downloads/

b. Open Command Prompt (in Windows) or terminal (in Mac or Linux) and run following code to make sure python is installed successfully: -

```
python --version
```

If "Python 3.9.x" is displayed then installation was successful. In case it shows "Python 2.x.x" then try running following command:

```
python3 --version
```

If this works then for every upcoming command replace "python" with "python3" and "pip" with "pip3"

c. Install virtualenv by running following command:

```
pip install virtualenv
```

d. Change directory to project directory.

e. Create a virtual environment by running following command:

```
python -m venv env
```

f. Activate the virtual environment by running following command:

For Windows

```
.\env\Scripts\activate
```

For Max or Linux

```
source env/bin/activate
```

g. Install all project dependencies by running following command:

```
pip install -r requirements.txt
```

2. Starting Project

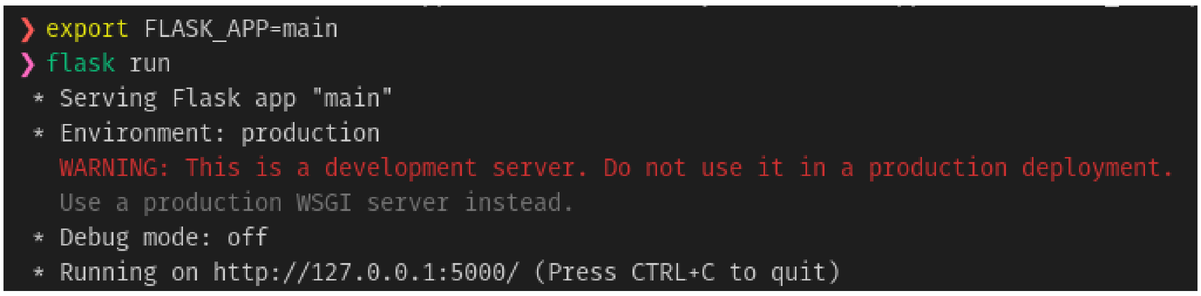
Run following commands to start the project:

In Windows:

```
set FLASK_APP=main
flask run
```

In Mac or Linux:

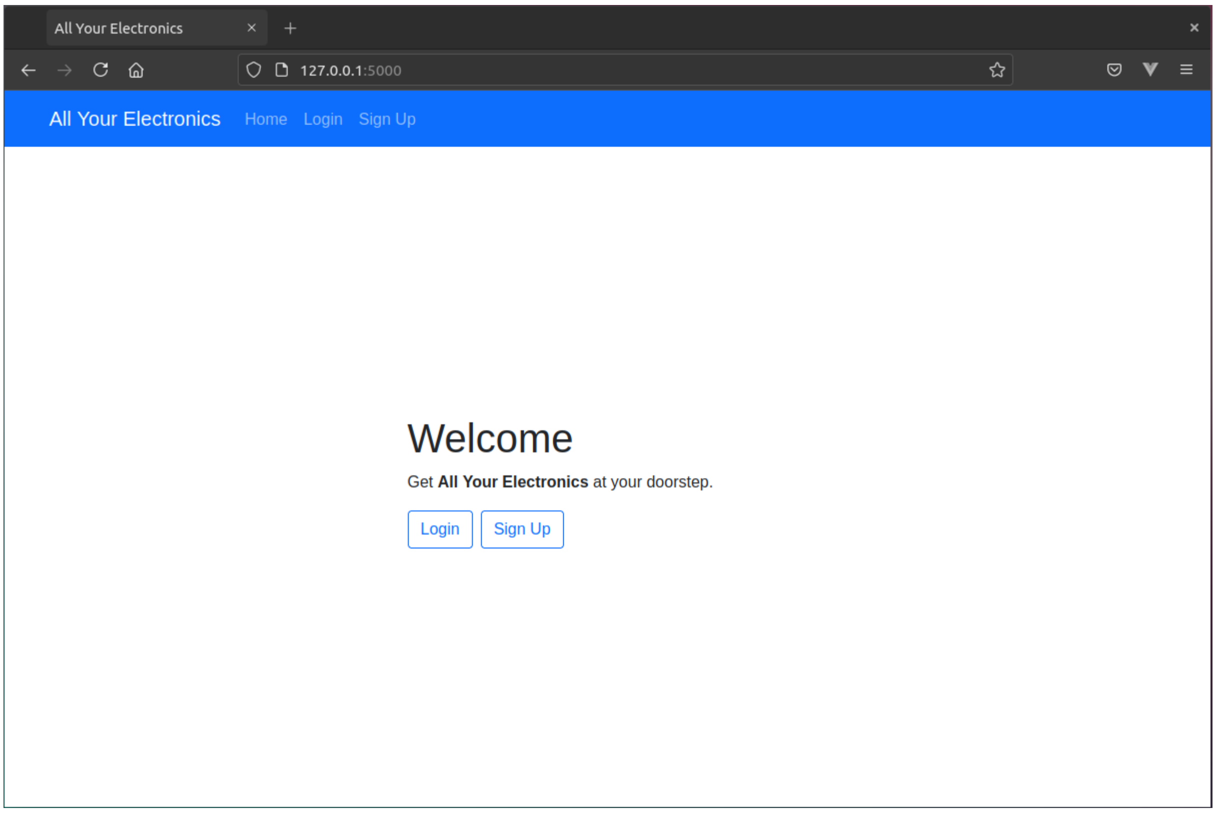
```
export FLASK_APP=main
flask run
```



```
❯ export FLASK_APP=main
❯ flask run
 * Serving Flask app "main"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

3. Screenshots of Implemented System Capabilities

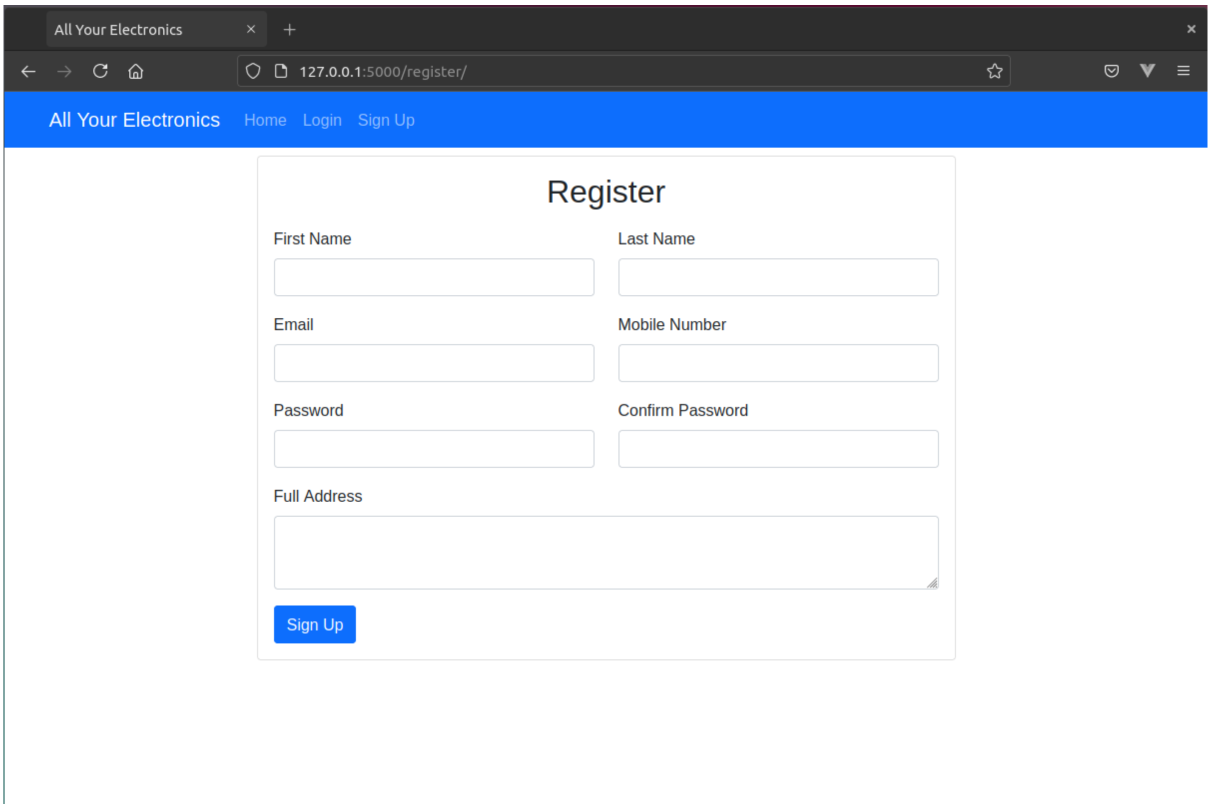
a. Home Screen



Welcome

Get **All Your Electronics** at your doorstep.

b. User Registration or Sign Up



Register

First Name

Last Name

Email

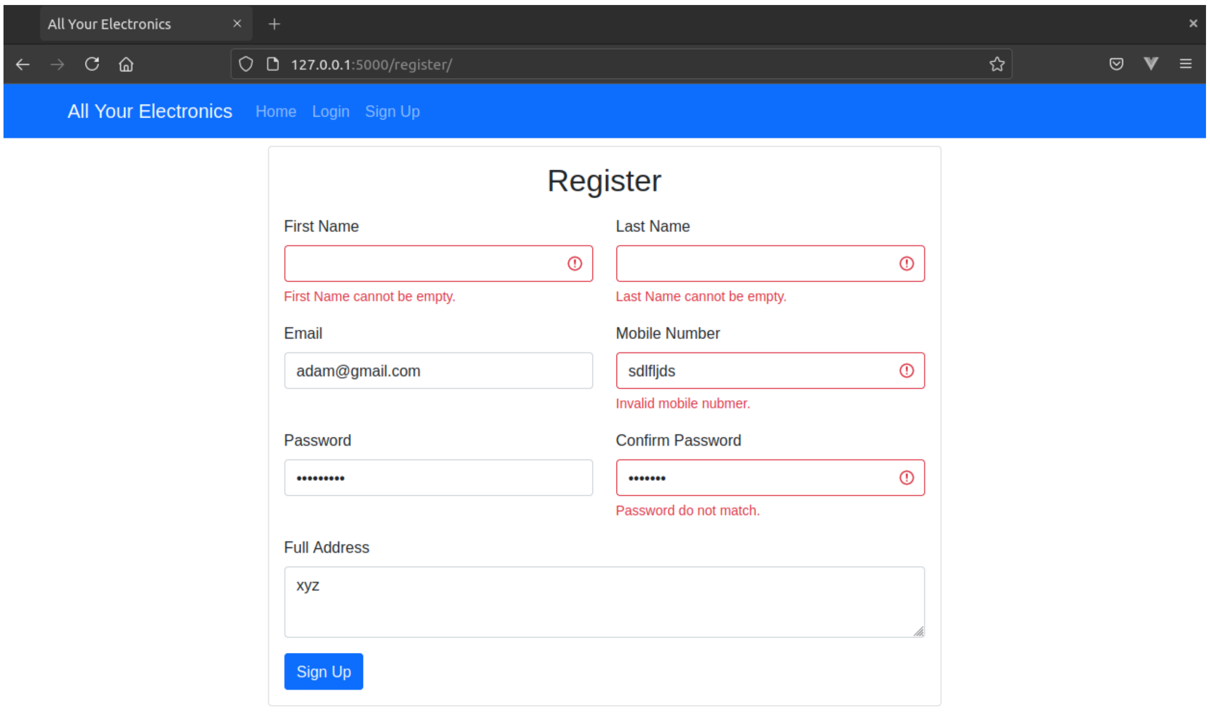
Mobile Number

Password

Confirm Password

Full Address

Initial screen



All Your Electronics

127.0.0.1:5000/register/

All Your Electronics Home Login Sign Up

Register

First Name

First Name cannot be empty.

Last Name

Last Name cannot be empty.

Email

adam@gmail.com

Mobile Number

sdlfljds

Invalid mobile nubmer.

Password

•••••••••

Confirm Password

•••••••

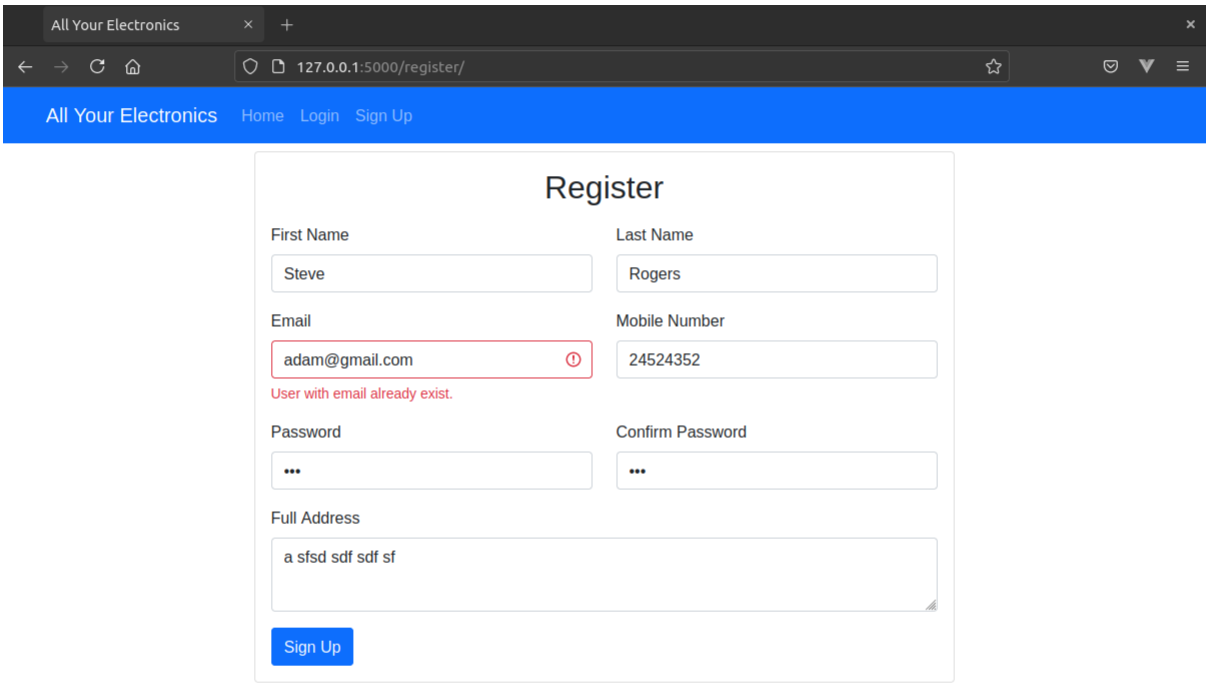
Password do not match.

Full Address

xyz

Sign Up

screen showing form validation 1



All Your Electronics

127.0.0.1:5000/register/

All Your Electronics Home Login Sign Up

Register

First Name

Steve

Last Name

Rogers

Email

adam@gmail.com

User with email already exist.

Mobile Number

24524352

Password

•••

Confirm Password

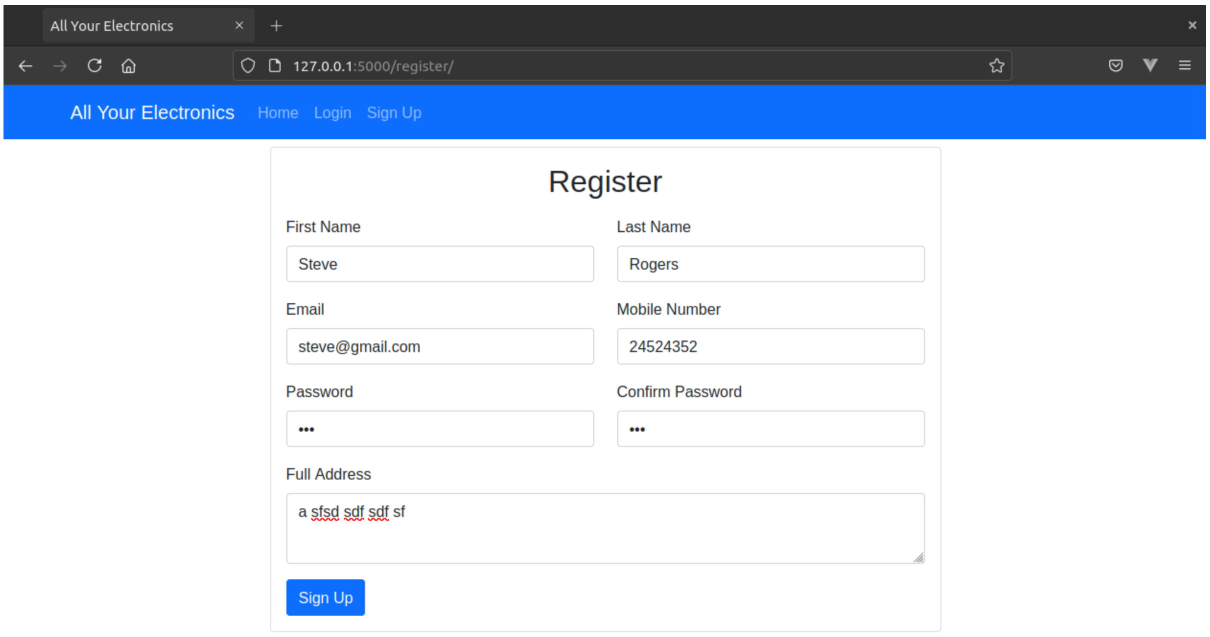
•••

Full Address

a sfsd sdf sdf sf

Sign Up

Screen showing form validation 2



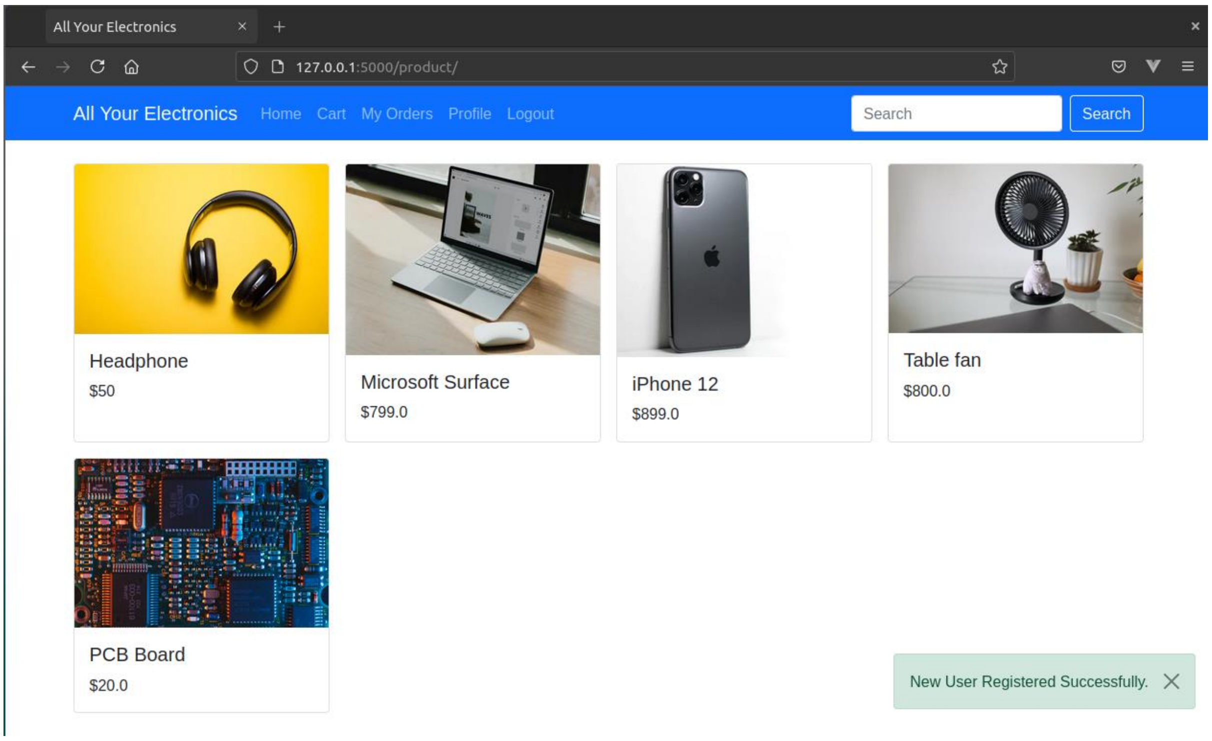
Register

First Name	Last Name
Steve	Rogers
Email	Mobile Number
steve@gmail.com	24524352
Password	Confirm Password
•••	•••

Full Address

a sfsd sdf sdf sf

Screen showing form with correct data



Headphone
$50

Microsoft Surface
$799.0

iPhone 12
$899.0

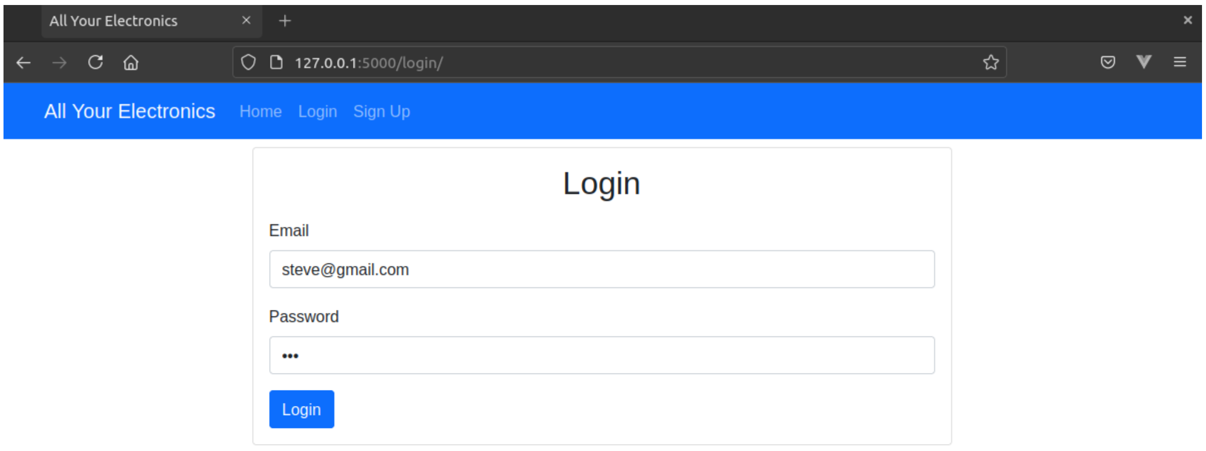
Table fan
$800.0

PCB Board
$20.0

New User Registered Successfully.

Screen showing success message

c. User Login


All Your Electronics

127.0.0.1:5000/login/

Login

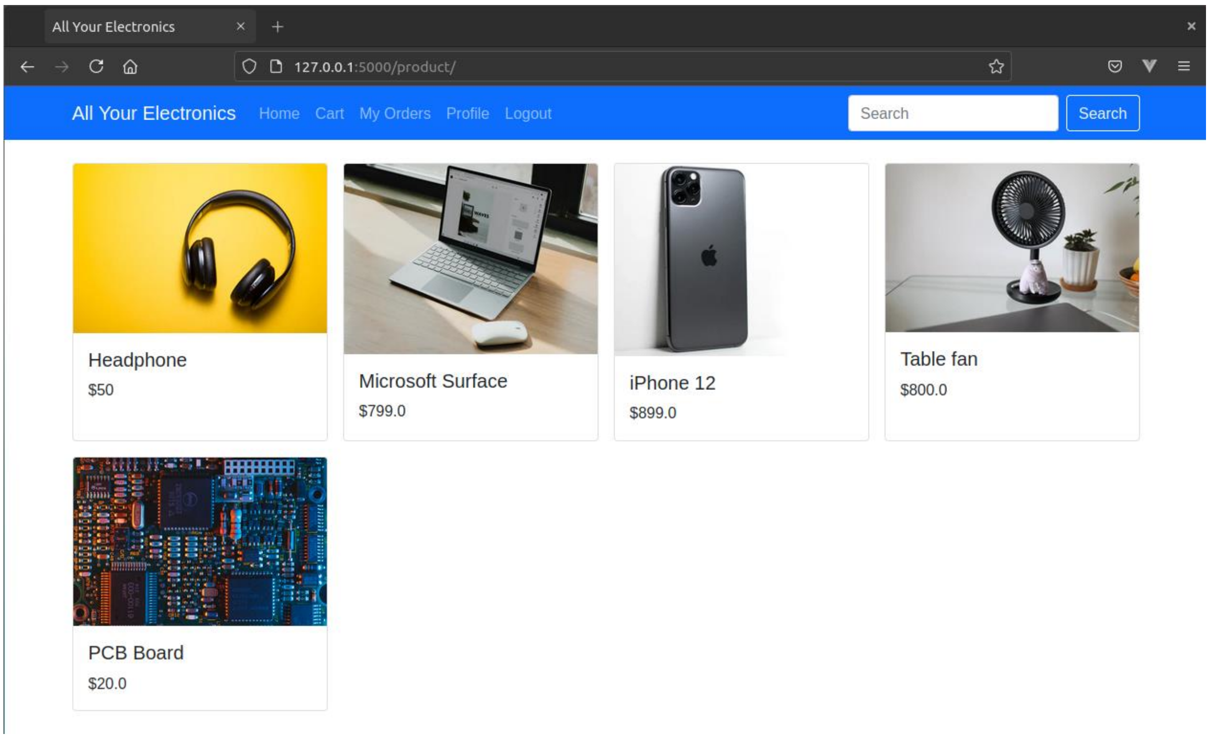
Email

steve@gmail.com

Password

•••

Initial screen with valid input


All Your Electronics

127.0.0.1:5000/product/

Headphone

$50

Microsoft Surface

$799.0

iPhone 12

$899.0

Table fan

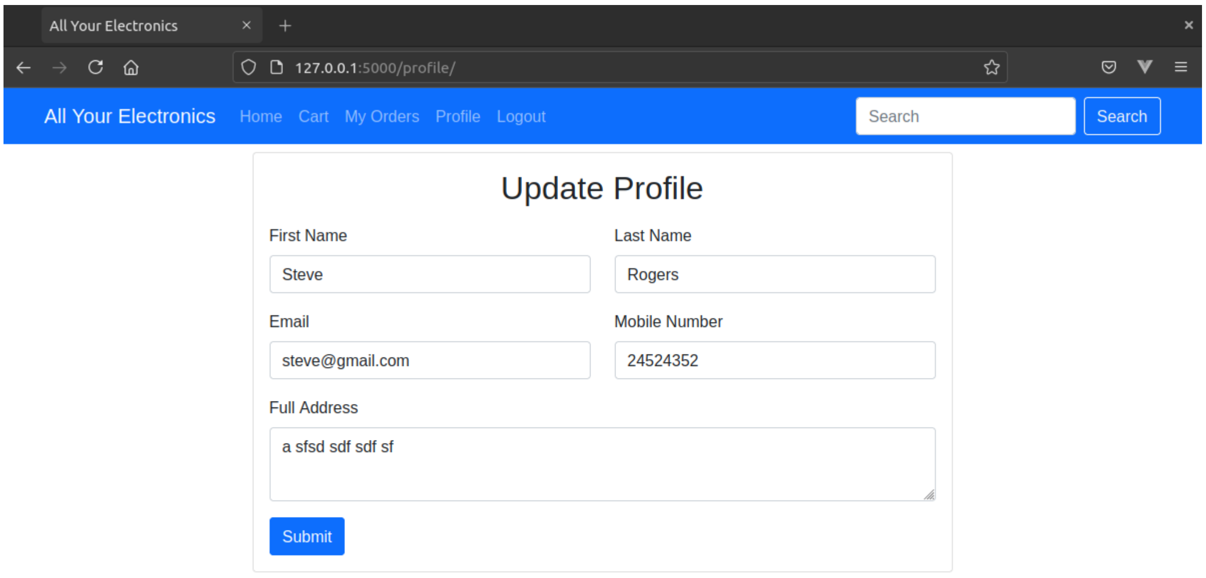
$800.0

PCB Board

$20.0

Screen after clicking login button

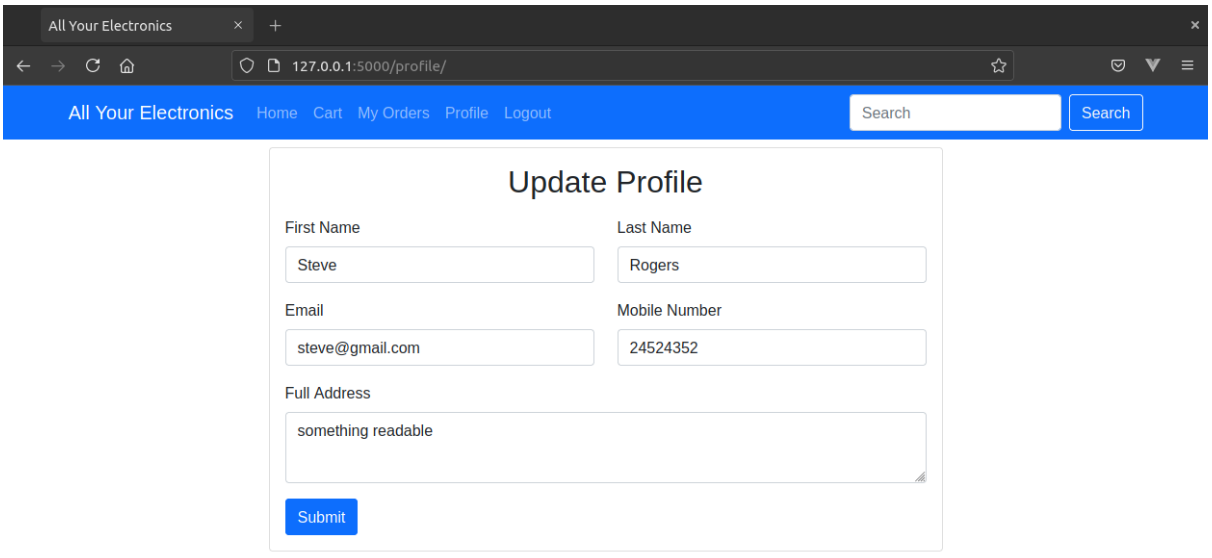
d. Update User/Customer Information



Update Profile

First Name: Steve
Last Name: Rogers
Email: steve@gmail.com
Mobile Number: 24524352
Full Address: a sfsd sdf sdf sf

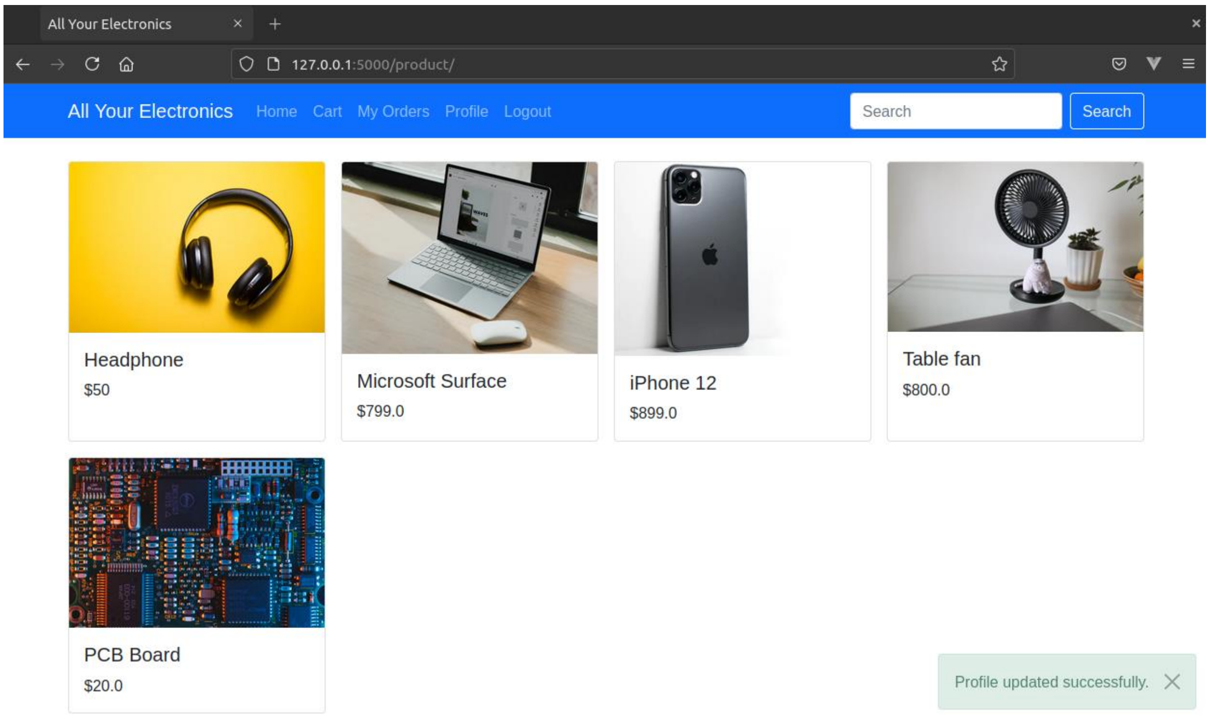
Initial screen



Update Profile

First Name: Steve
Last Name: Rogers
Email: steve@gmail.com
Mobile Number: 24524352
Full Address: something readable

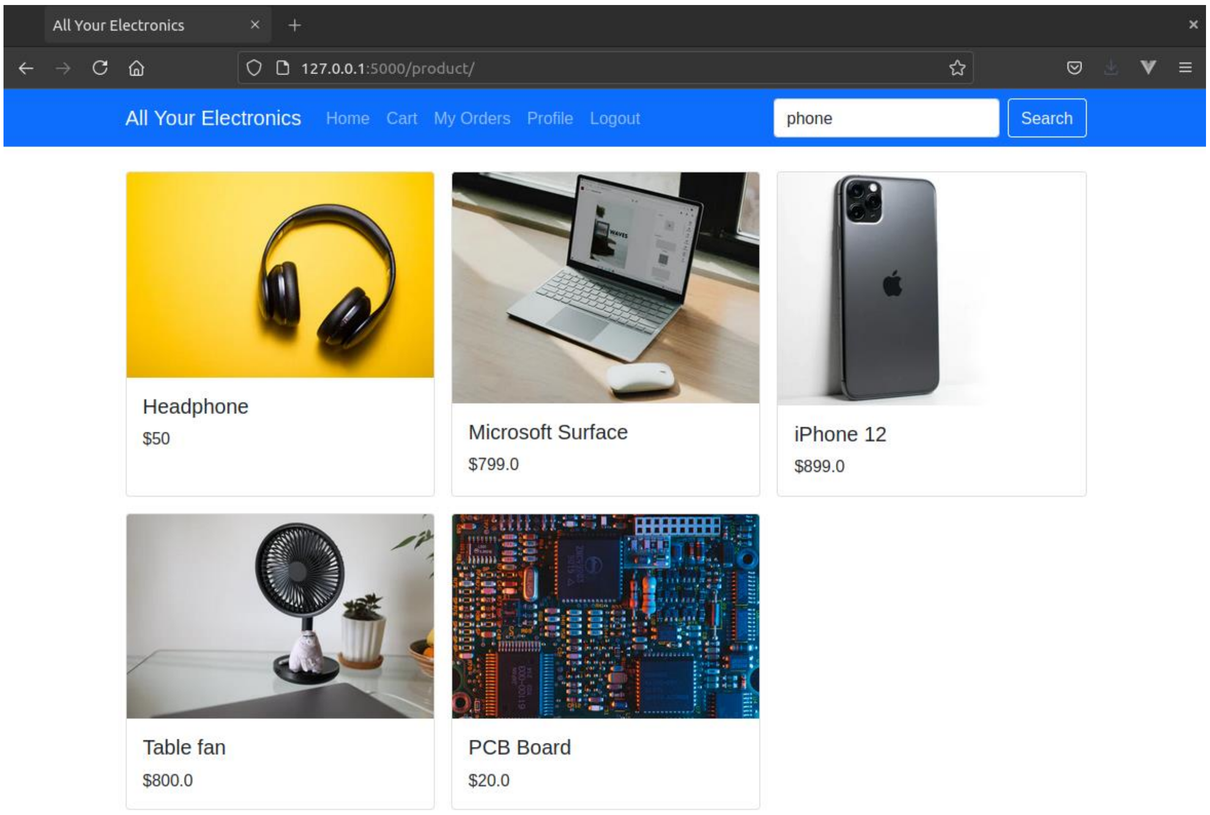
Screen with new input



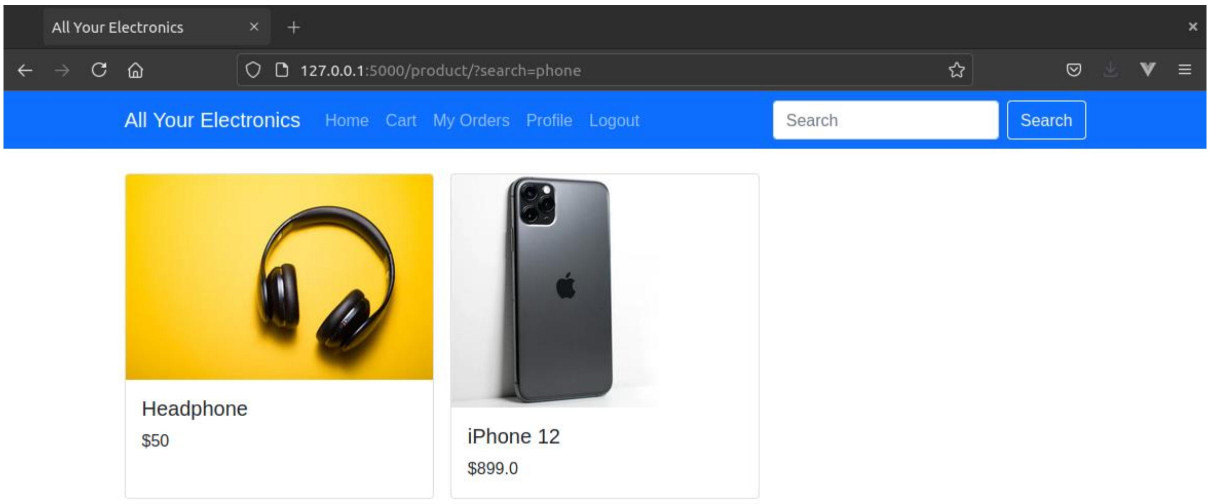
Profile updated successfully.

Screen after clicking submit showing success message

e. Product Search



Initial screen showing text in search box

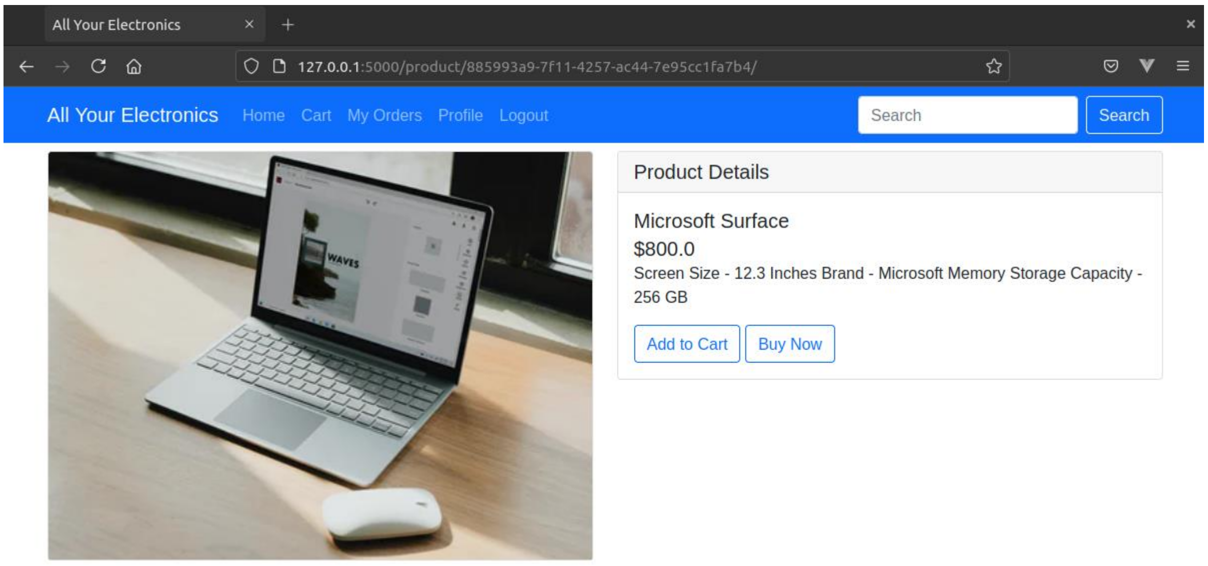


Headphone
$50

iPhone 12
$899.0

Screen showing result of search

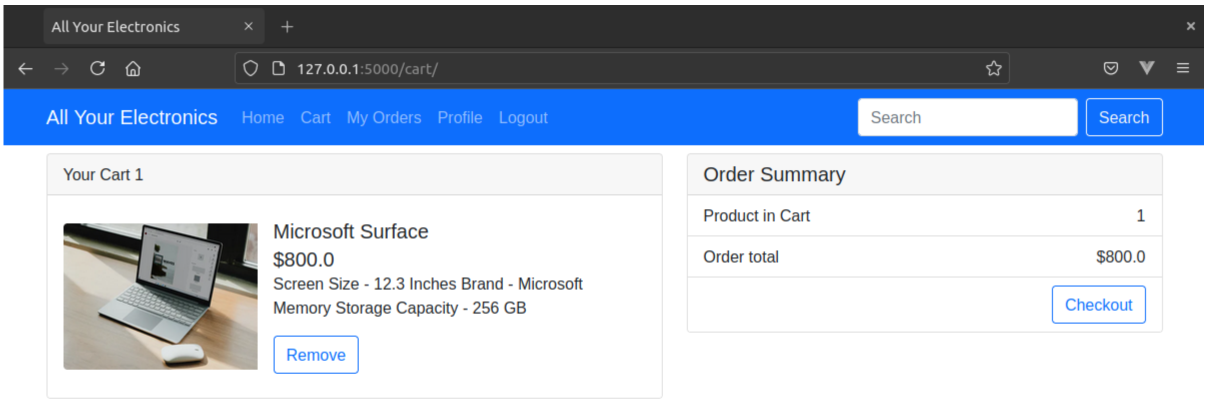
f. Add to Cart



Product Details

Microsoft Surface
$800.0
Screen Size - 12.3 Inches Brand - Microsoft Memory Storage Capacity - 256 GB

Initial screen



All Your Electronics

127.0.0.1:5000/cart/

All Your Electronics Home Cart My Orders Profile Logout Search Search

Your Cart 1

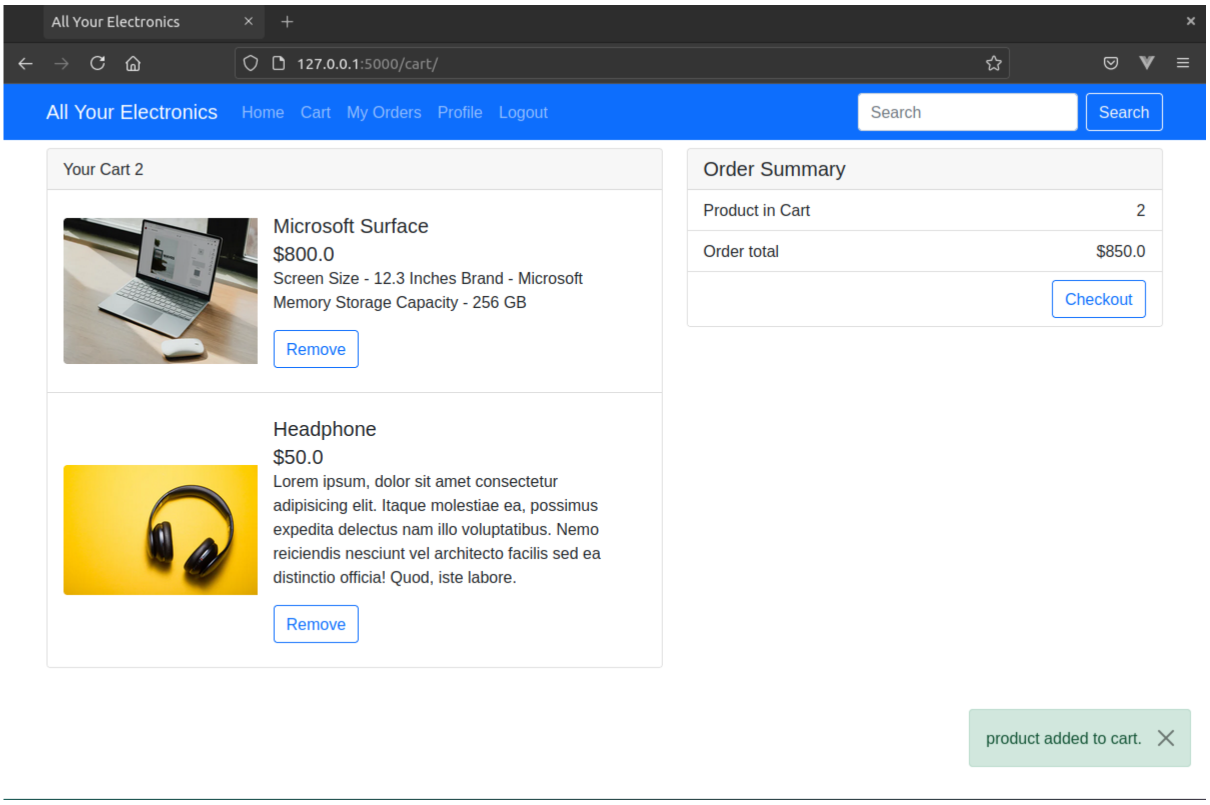
Microsoft Surface
$800.0
Screen Size - 12.3 Inches Brand - Microsoft Memory Storage Capacity - 256 GB
Remove

Order Summary

Product in Cart	1
Order total	$800.0

Checkout

Screen showing cart after clicking add to cart



All Your Electronics

127.0.0.1:5000/cart/

All Your Electronics Home Cart My Orders Profile Logout Search Search

Your Cart 2

Microsoft Surface
$800.0
Screen Size - 12.3 Inches Brand - Microsoft Memory Storage Capacity - 256 GB
Remove

Headphone
$50.0
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Itaque molestiae ea, possimus expedita delectus nam illo voluptatibus. Nemo reiciendis nesciunt vel architecto facilis sed ea distinctio officia! Quod, iste labore.
Remove

Order Summary

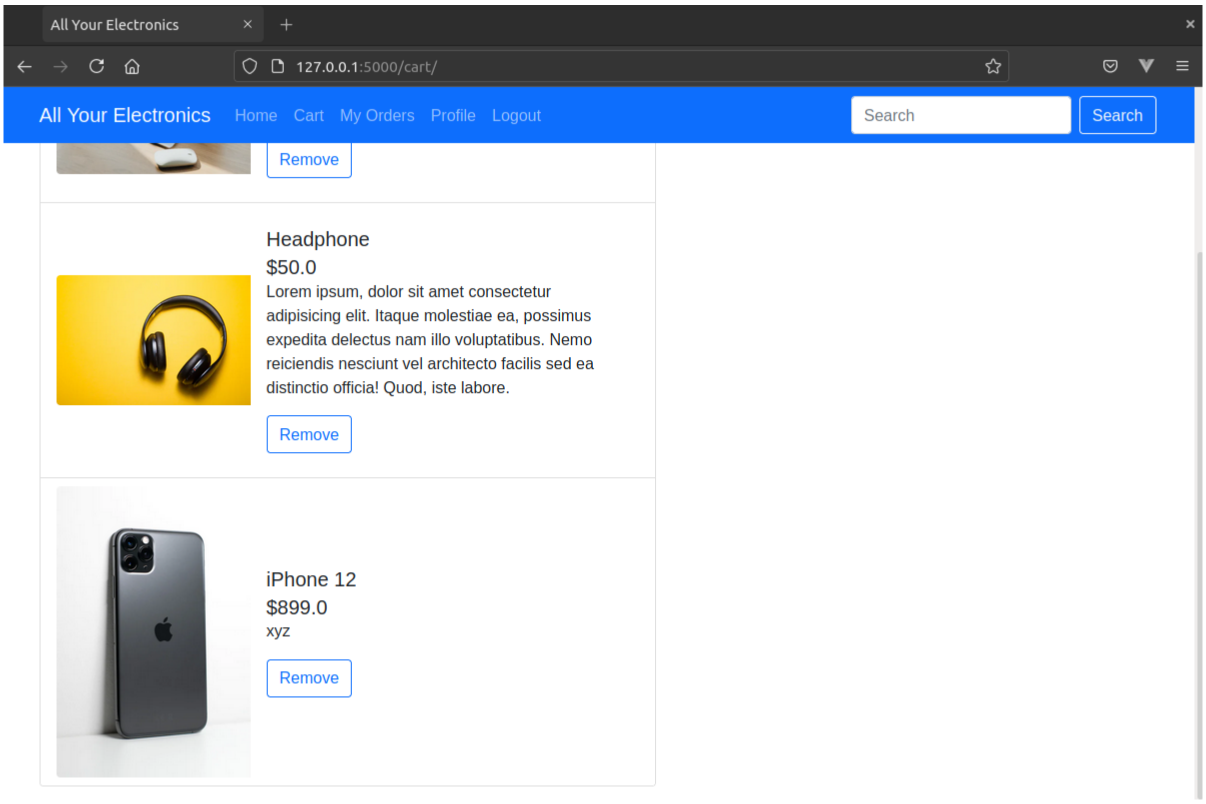
Product in Cart	2
Order total	$850.0

Checkout

product added to cart.

Screen showing cart after clicking add to cart for another product

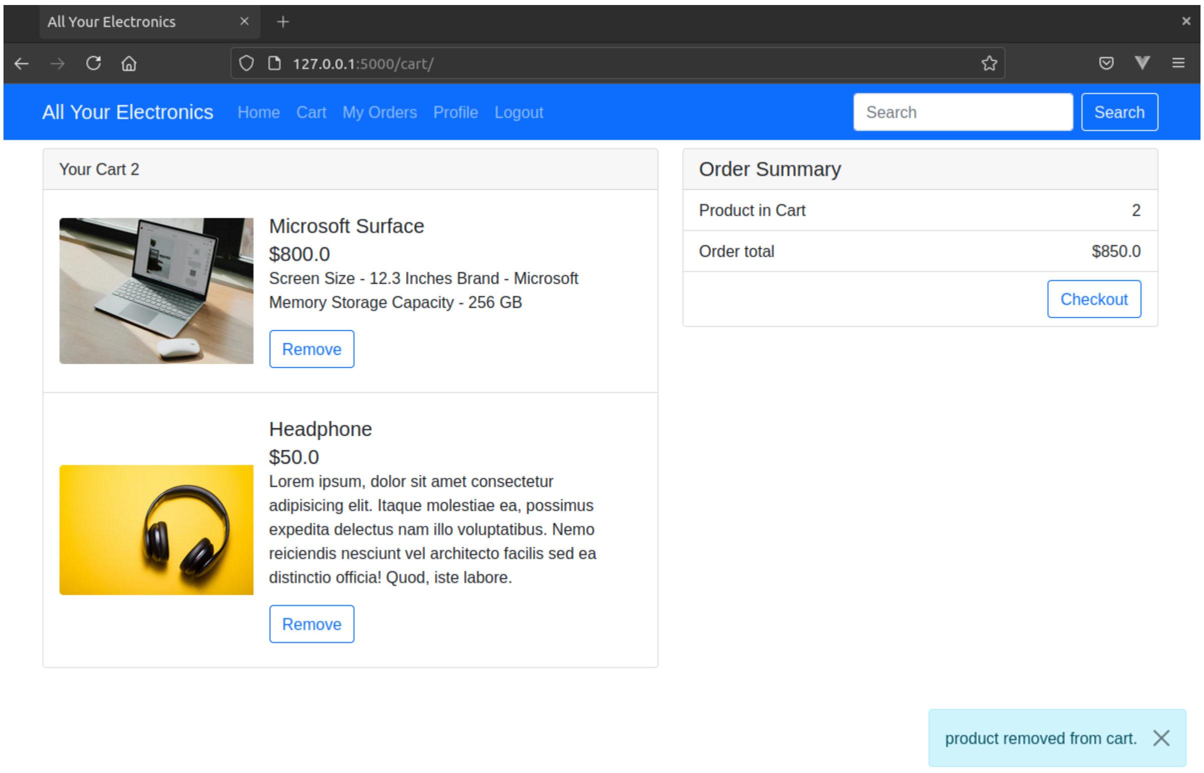
g. Remove from Cart



Headphone
$50.0
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Itaque molestiae ea, possimus expedita delectus nam illo voluptatibus. Nemo reiciendis nesciunt vel architecto facilis sed ea distinctio officia! Quod, iste labore.

iPhone 12
$899.0
xyz

Initial screen showing products in cart



Your Cart 2

Microsoft Surface
$800.0
Screen Size - 12.3 Inches Brand - Microsoft Memory Storage Capacity - 256 GB

Headphone
$50.0
Lorem ipsum, dolor sit amet consectetur adipisicing elit. Itaque molestiae ea, possimus expedita delectus nam illo voluptatibus. Nemo reiciendis nesciunt vel architecto facilis sed ea distinctio officia! Quod, iste labore.

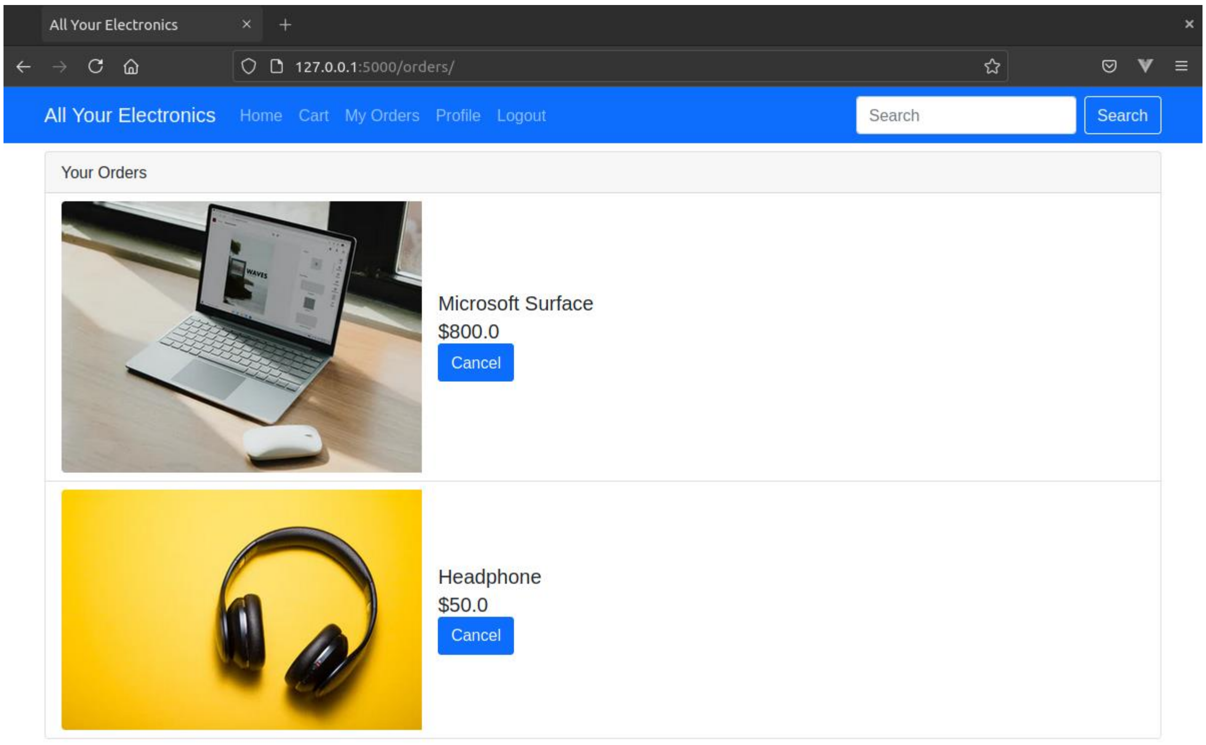
Order Summary

Product in Cart	2
Order total	$850.0

product removed from cart.

Screen showing products list after click remove button for last product

h. Cart checkout



All Your Electronics

127.0.0.1:5000/orders/

All Your Electronics Home Cart My Orders Profile Logout Search Search

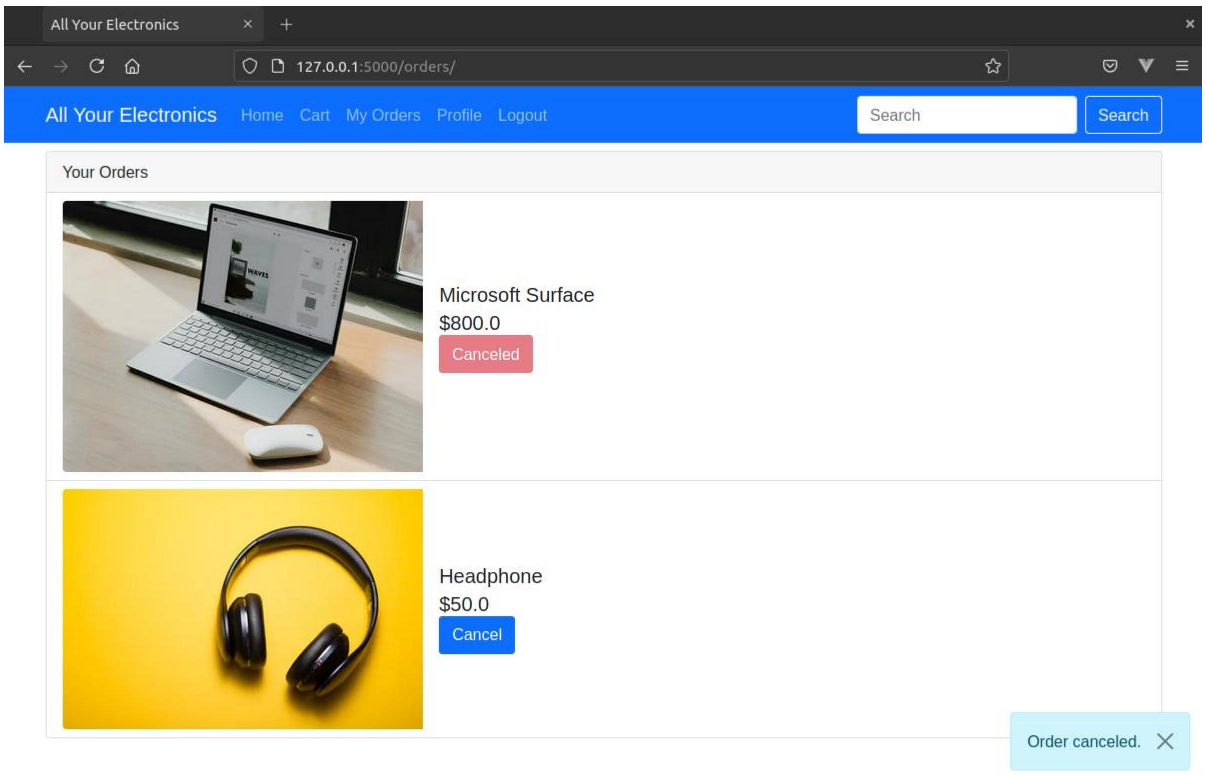
Your Orders

Microsoft Surface
$800.0
Cancel

Headphone
$50.0
Cancel

Screen showing orders list after clicking checkout button

i. Cancel Order



All Your Electronics

127.0.0.1:5000/orders/

All Your Electronics Home Cart My Orders Profile Logout Search Search

Your Orders

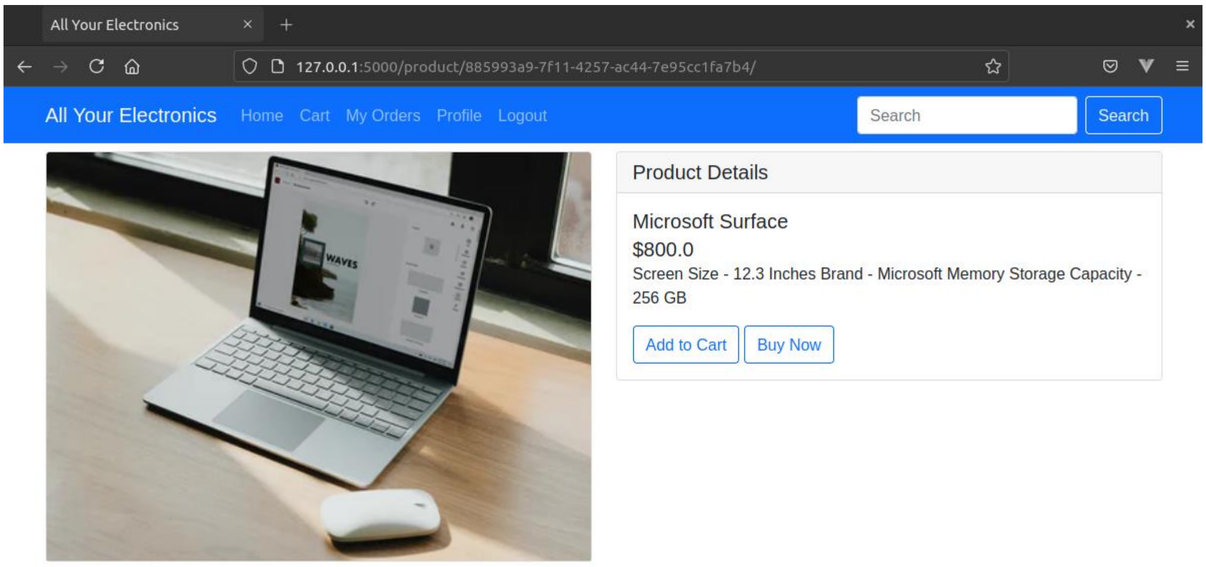
Microsoft Surface
$800.0
Canceled

Headphone
$50.0
Cancel

Order canceled.

Screen showing orders list after clicking cancel button for first order

j. Buy Now



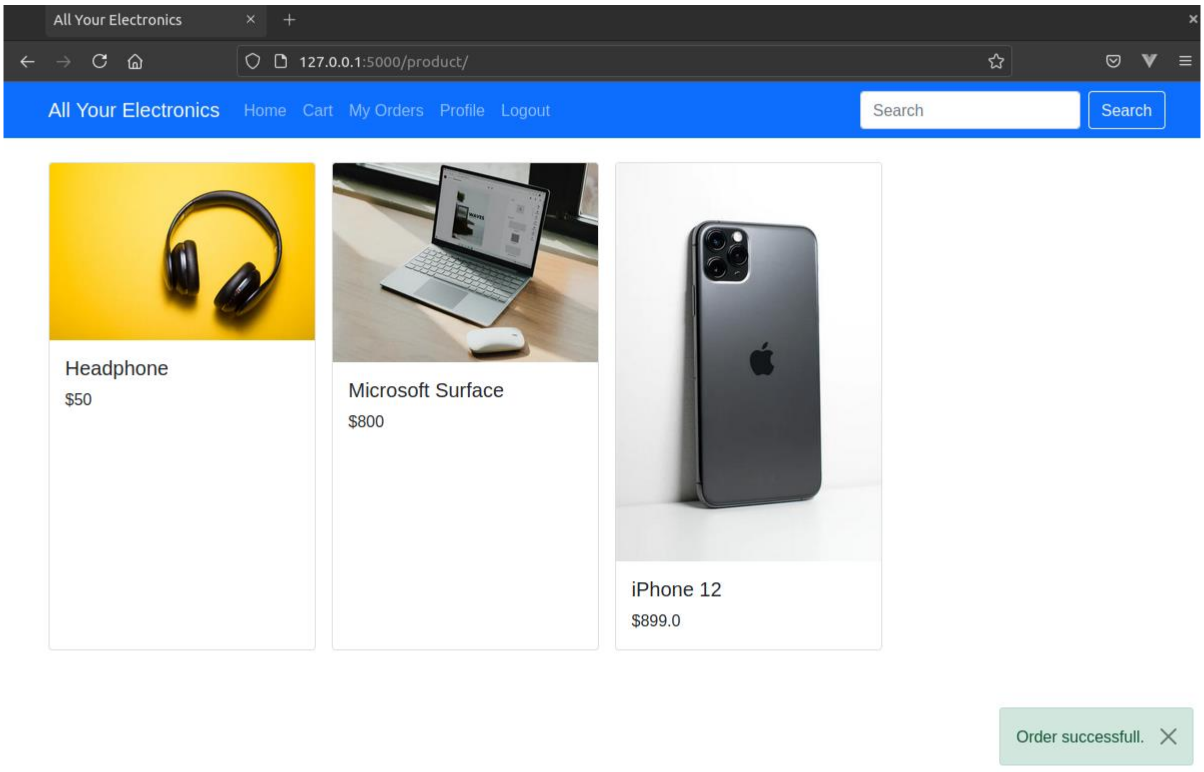
Product Details

Microsoft Surface

$800.0

Screen Size - 12.3 Inches Brand - Microsoft Memory Storage Capacity - 256 GB

Initial screen showing product detail



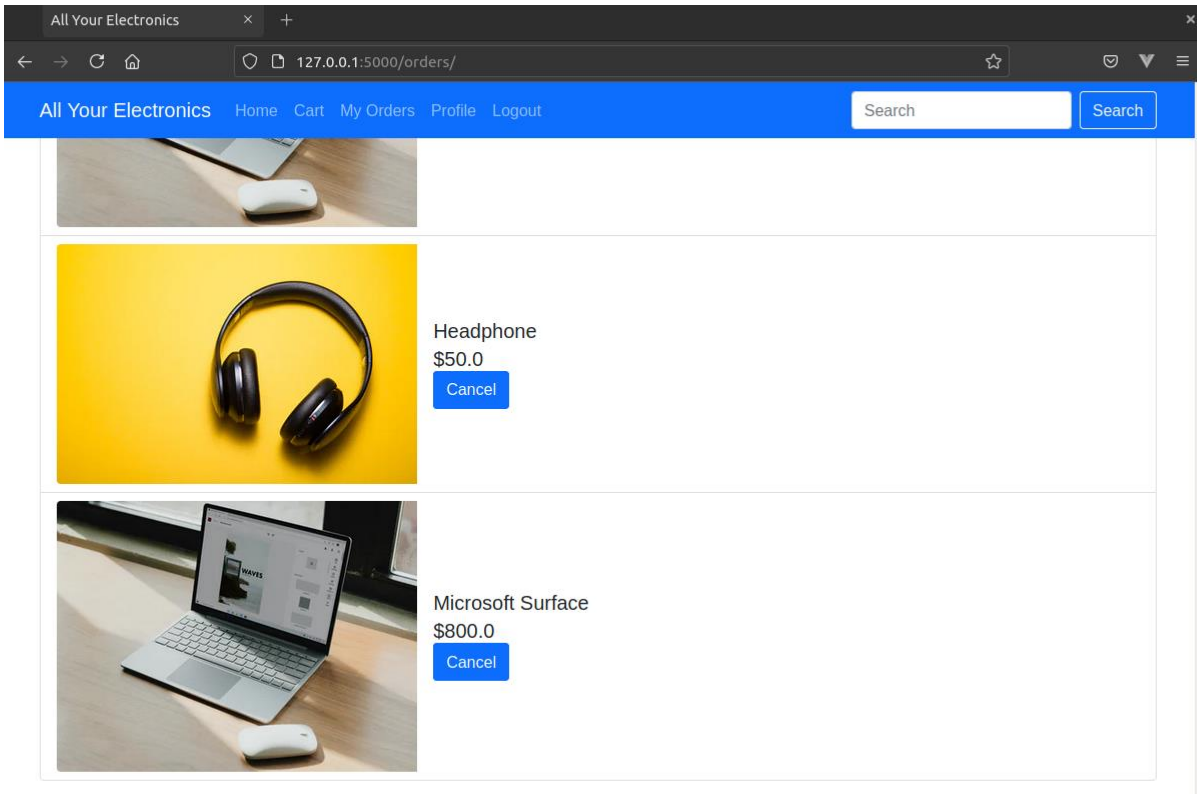
Headphone
$50

Microsoft Surface
$800

iPhone 12
$899.0

Order successfull.

Screen after clicking buy now showing success message.

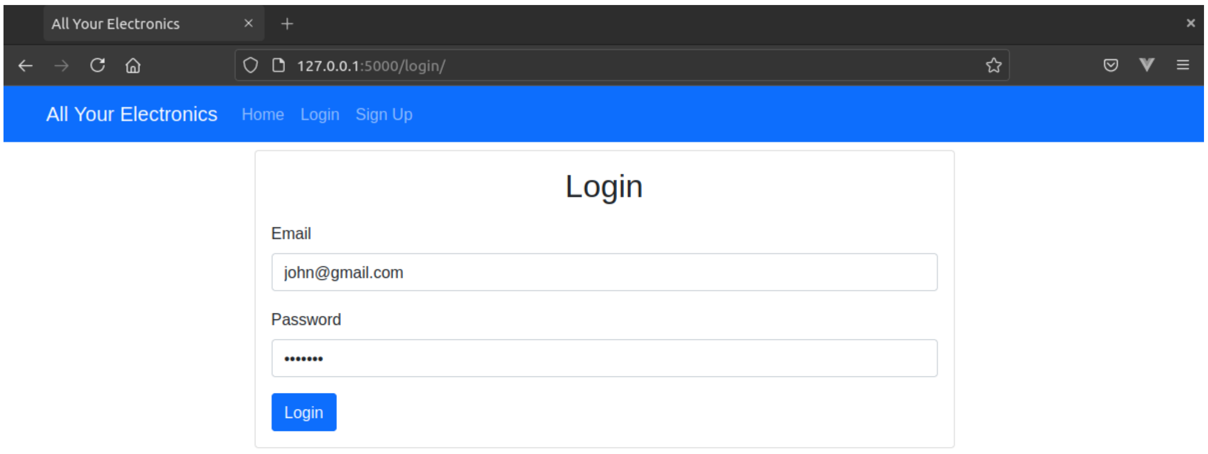


Headphone
$50.0
Cancel

Microsoft Surface
$800.0
Cancel

Screen showing orders list after clicking buy now.

k. Administrator Login



Login

Email

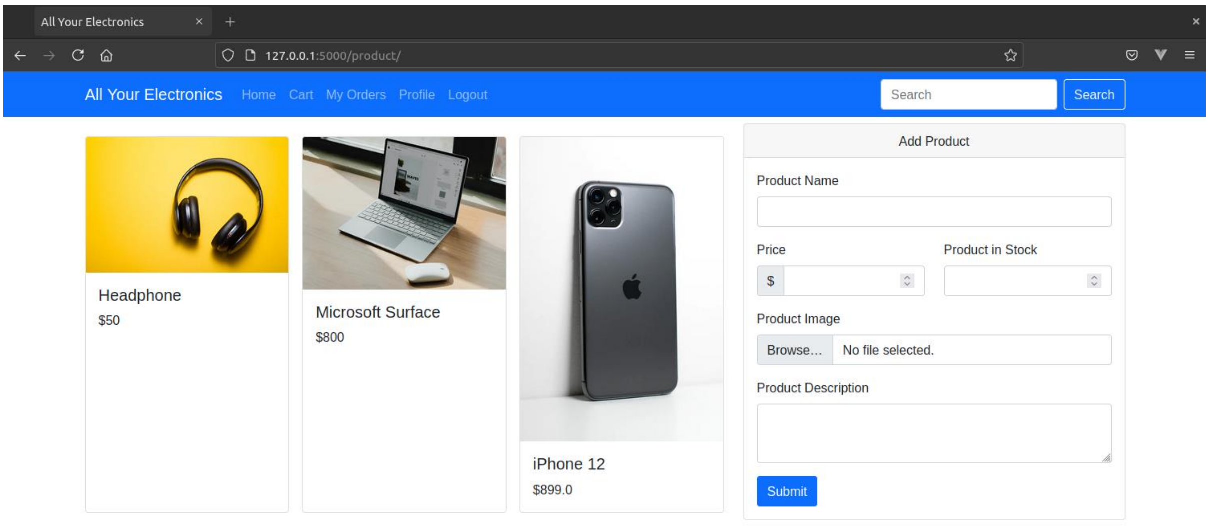
john@gmail.com

Password

•••••••

Login

Initial screen showing valid input of an administrator



Headphone
$50

Microsoft Surface
$800

iPhone 12
$899.0

Add Product

Product Name:

Price: $

Product in Stock:

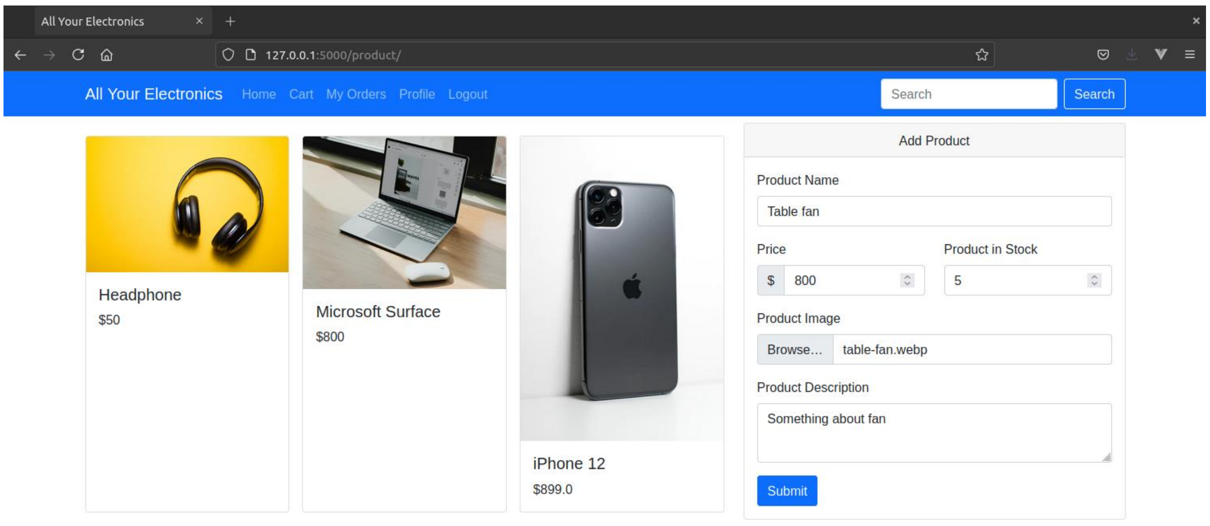
Product Image: Browse… No file selected.

Product Description:

Submit

Screen after clicking login button

l. Update Product Catalogue



Headphone
$50

Microsoft Surface
$800

iPhone 12
$899.0

Add Product

Product Name: Table fan

Price: $ 800

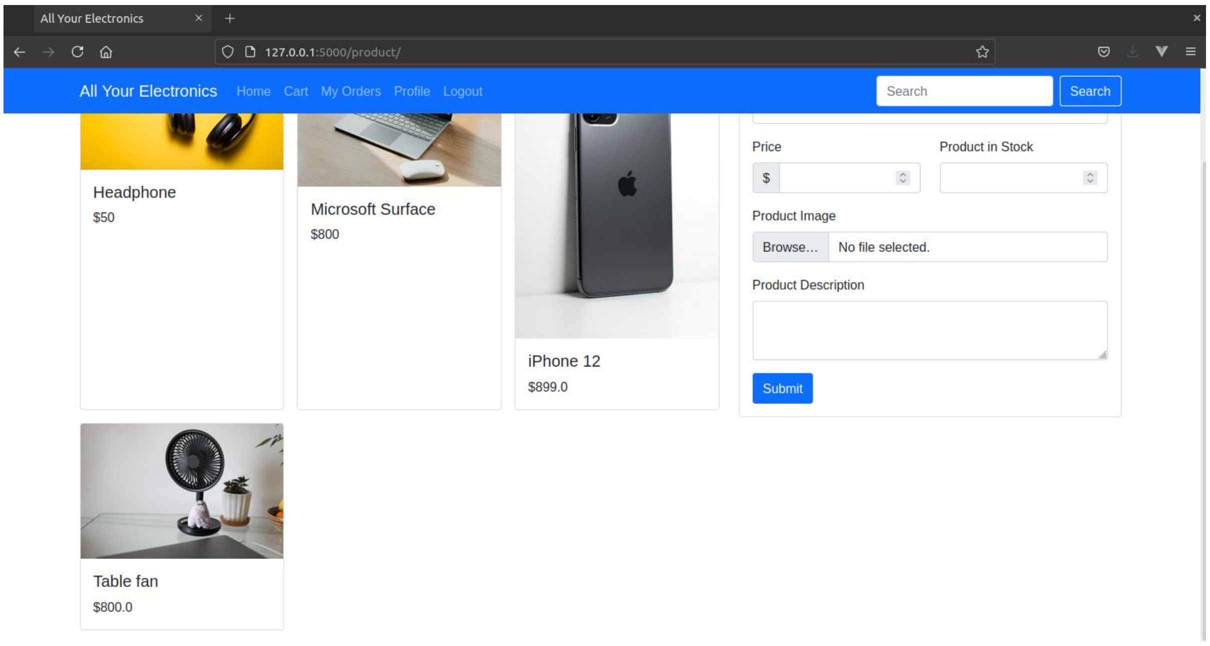
Product in Stock: 5

Product Image: Browse… table-fan.webp

Product Description: Something about fan

Submit

Screen showing valid input for new product to be added.



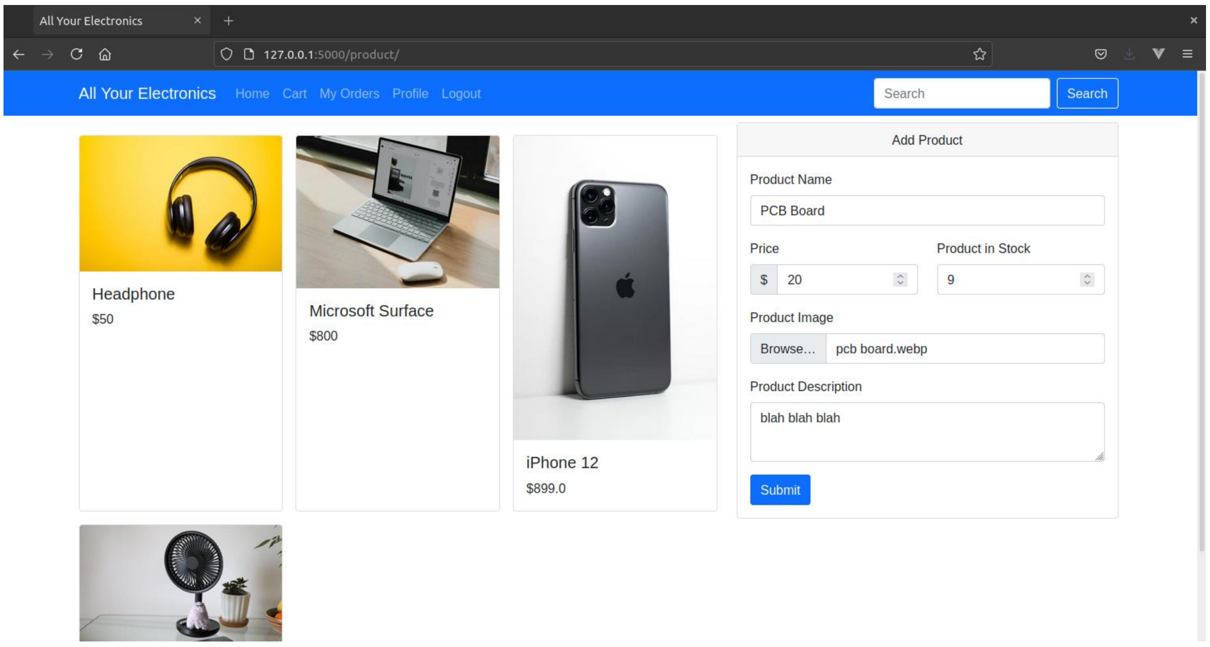
Price: $

Product in Stock:

Product Image: No file selected.

Product Description:

Screen after clicking submit button.



Product Name: PCB Board

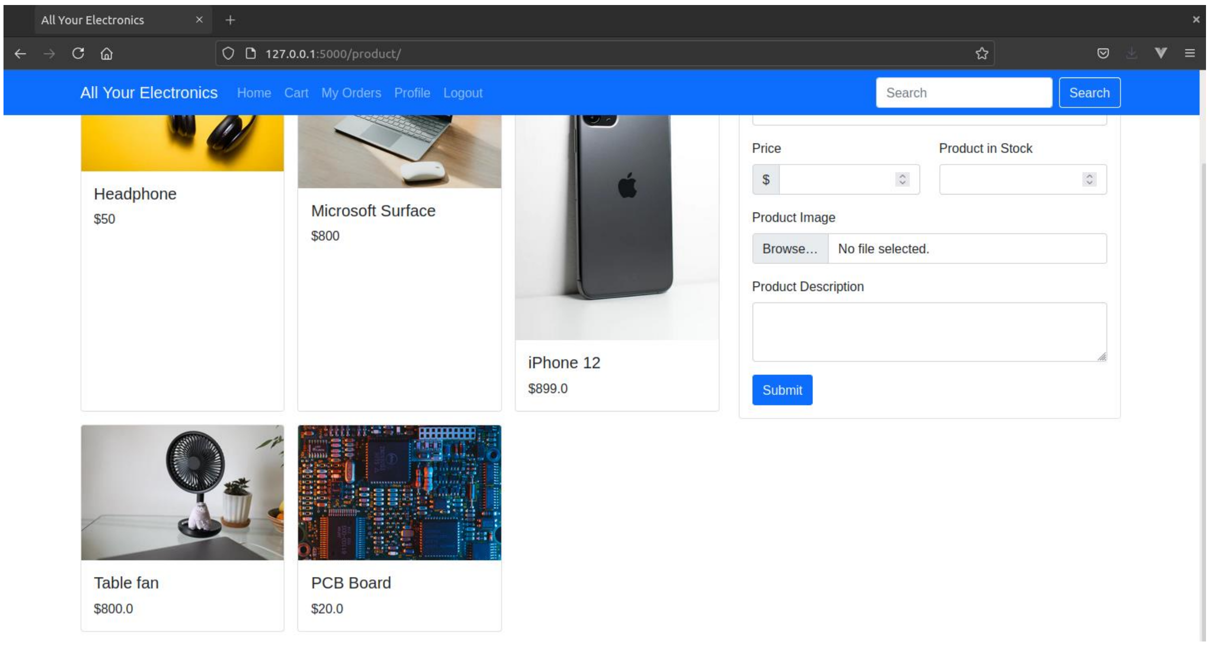
Price: $ 20

Product in Stock: 9

Product Image: pcb board.webp

Product Description: blah blah blah

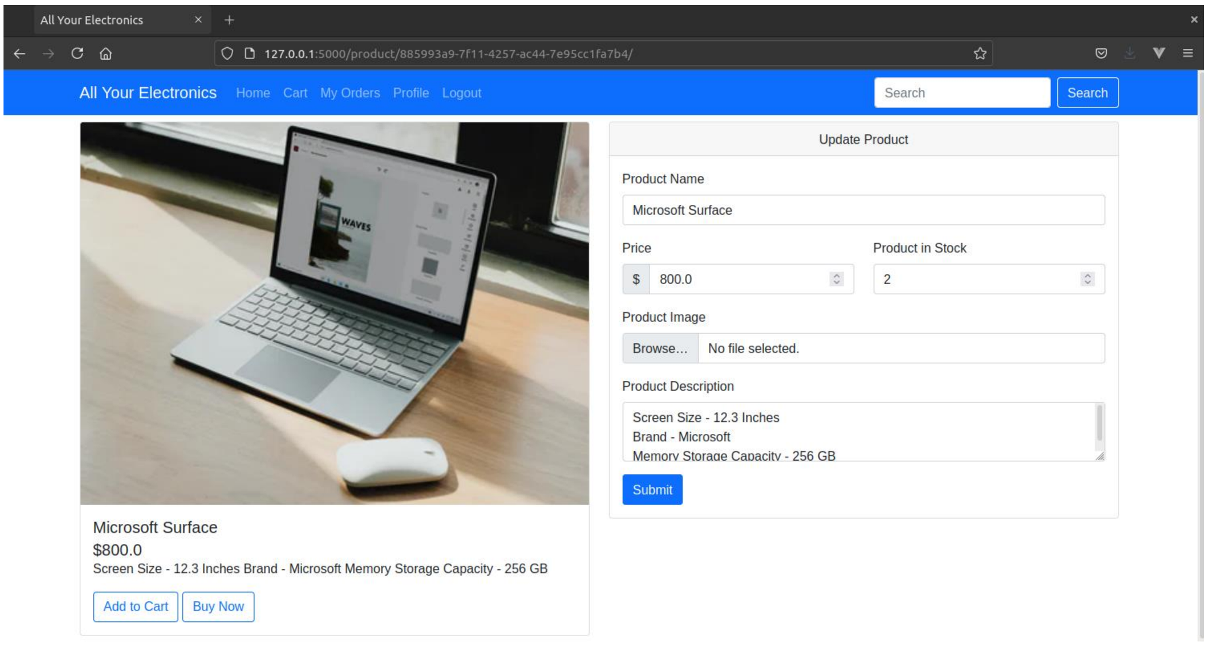
Screen showing valid input for new product to be added.



Headphone
$50
Microsoft Surface
$800
iPhone 12
$899.0
Table fan
$800.0
PCB Board
$20.0

Price
$
Product in Stock
Product Image
Browse... No file selected.
Product Description
Submit

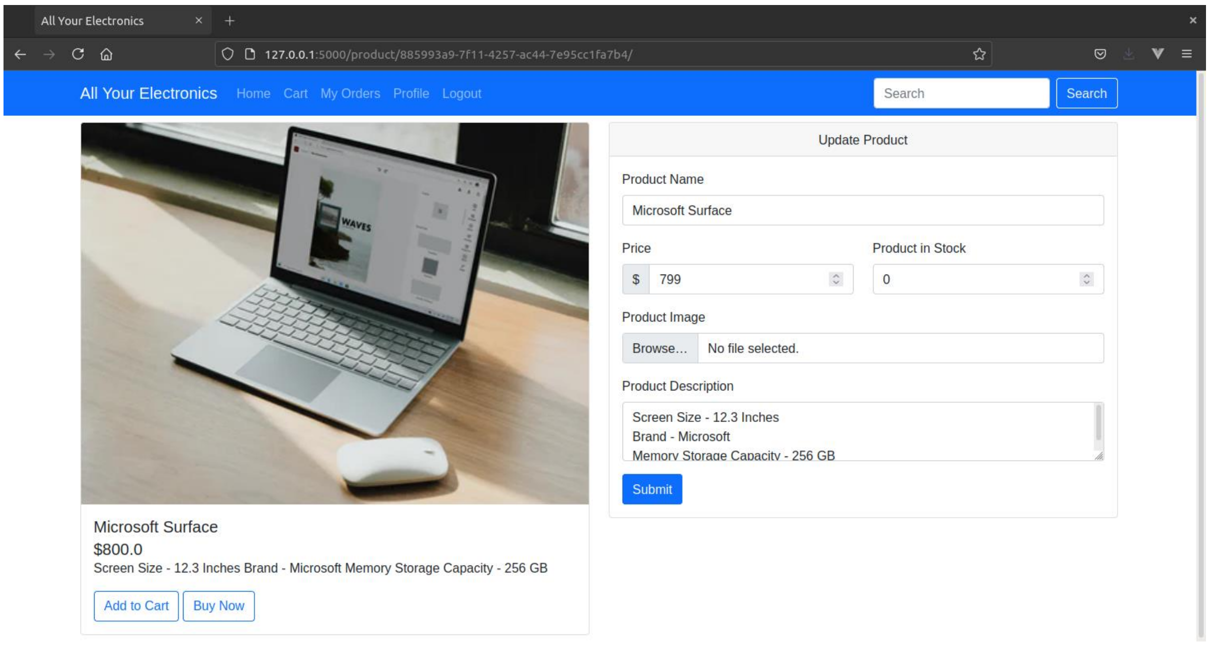
Screen after clicking submit button.



Microsoft Surface
$800.0
Screen Size - 12.3 Inches Brand - Microsoft Memory Storage Capacity - 256 GB
Add to Cart
Buy Now

Update Product
Product Name
Microsoft Surface
Price
$ 800.0
Product in Stock
2
Product Image
Browse... No file selected.
Product Description
Screen Size - 12.3 Inches
Brand - Microsoft
Memory Storage Capacity - 256 GB
Submit

Initial screen showing product update form.


All Your Electronics

127.0.0.1:5000/product/885993a9-7f11-4257-ac44-7e95cc1fa7b4/

All Your Electronics Home Cart My Orders Profile Logout Search Search

Update Product

Product Name: Microsoft Surface

Price: $ 799

Product in Stock: 0

Product Image: Browse… No file selected.

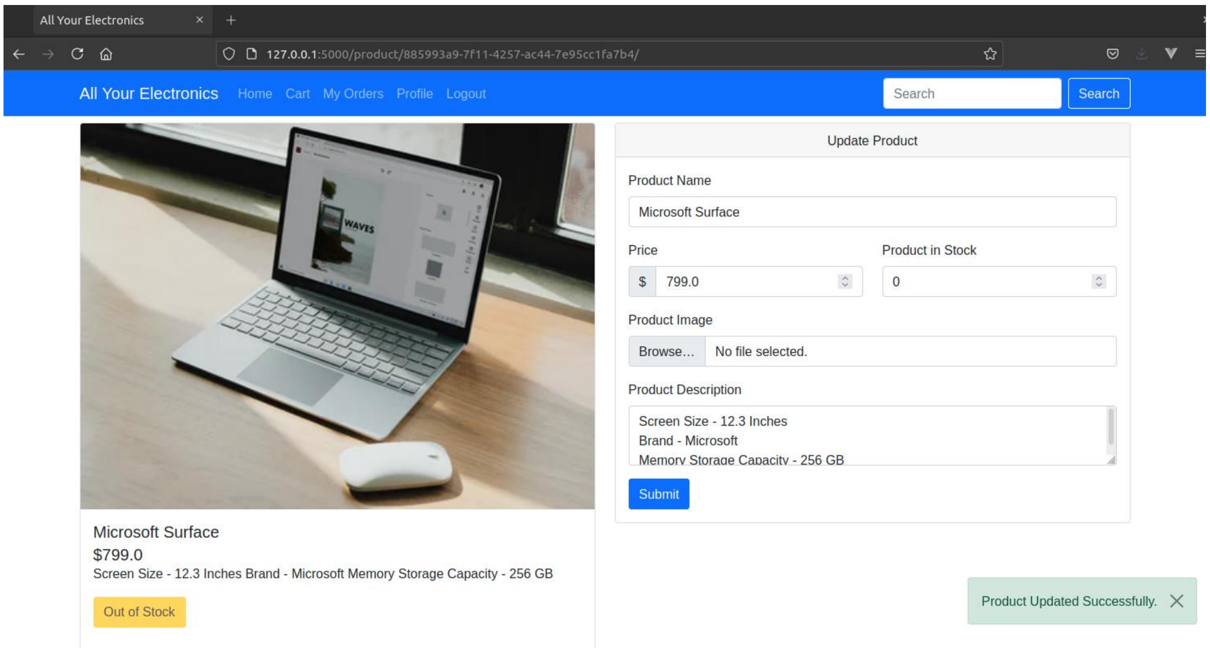
Product Description:
Screen Size - 12.3 Inches
Brand - Microsoft
Memory Storage Capacity - 256 GB

Submit

Microsoft Surface
$800.0
Screen Size - 12.3 Inches Brand - Microsoft Memory Storage Capacity - 256 GB

Add to Cart Buy Now

Screen showing new valid input.


All Your Electronics

127.0.0.1:5000/product/885993a9-7f11-4257-ac44-7e95cc1fa7b4/

All Your Electronics Home Cart My Orders Profile Logout Search Search

Update Product

Product Name: Microsoft Surface

Price: $ 799.0

Product in Stock: 0

Product Image: Browse… No file selected.

Product Description:
Screen Size - 12.3 Inches
Brand - Microsoft
Memory Storage Capacity - 256 GB

Submit

Microsoft Surface
$799.0
Screen Size - 12.3 Inches Brand - Microsoft Memory Storage Capacity - 256 GB

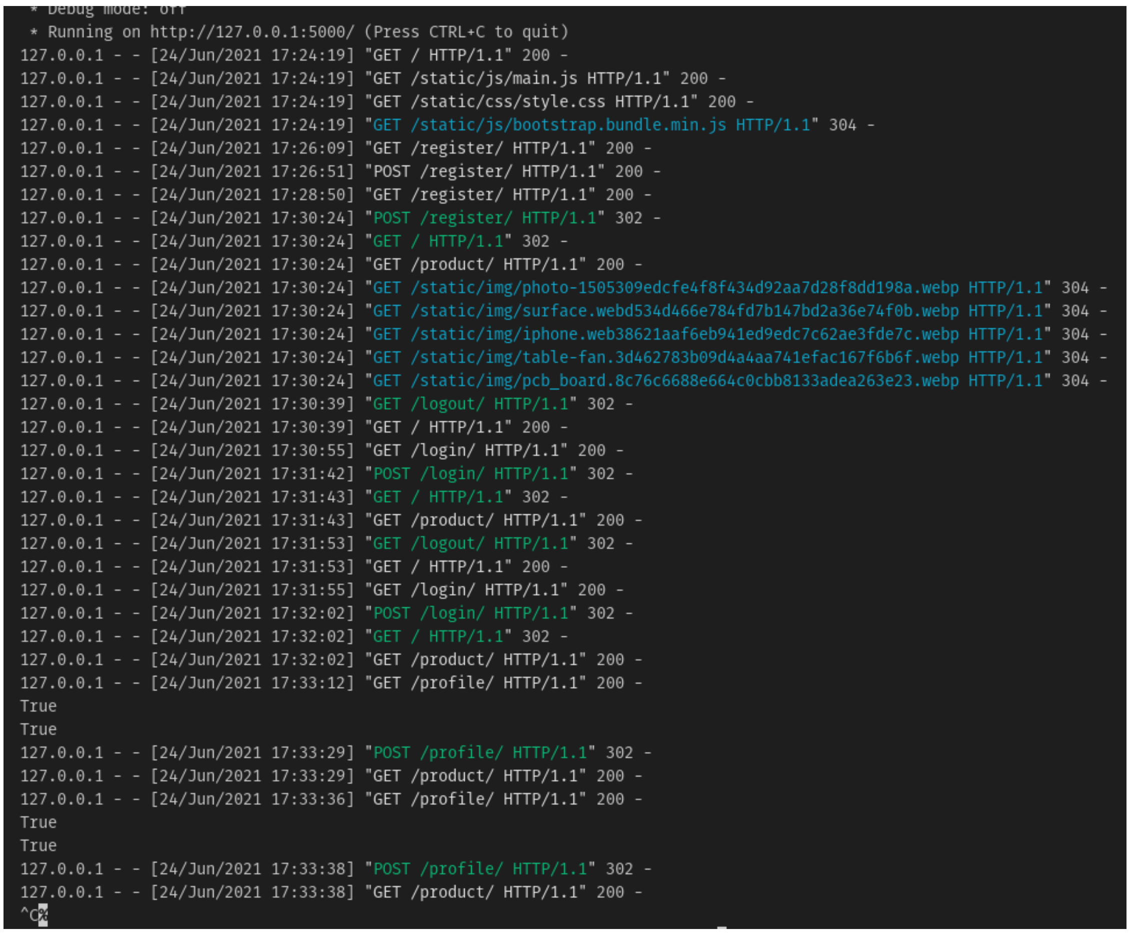
Out of Stock

Product Updated Successfully.

Screen after clicking submit button.

4. Stopping the Project

Press "Ctrl + C" on Windows or Linux and "Command + dot/period" on Mac



```
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [24/Jun/2021 17:24:19] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:24:19] "GET /static/js/main.js HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:24:19] "GET /static/css/style.css HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:24:19] "GET /static/js/bootstrap.bundle.min.js HTTP/1.1" 304 -
127.0.0.1 - - [24/Jun/2021 17:26:09] "GET /register/ HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:26:51] "POST /register/ HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:28:50] "GET /register/ HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:30:24] "POST /register/ HTTP/1.1" 302 -
127.0.0.1 - - [24/Jun/2021 17:30:24] "GET / HTTP/1.1" 302 -
127.0.0.1 - - [24/Jun/2021 17:30:24] "GET /product/ HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:30:24] "GET /static/img/photo-1505309edcfe4f8f434d92aa7d28f8dd198a.webp HTTP/1.1" 304 -
127.0.0.1 - - [24/Jun/2021 17:30:24] "GET /static/img/surface.webd534d466e784fd7b147bd2a36e74f0b.webp HTTP/1.1" 304 -
127.0.0.1 - - [24/Jun/2021 17:30:24] "GET /static/img/iphone.web38621aaf6eb941ed9edc7c62ae3fde7c.webp HTTP/1.1" 304 -
127.0.0.1 - - [24/Jun/2021 17:30:24] "GET /static/img/table-fan.3d462783b09d4a4aa741efac167f6b6f.webp HTTP/1.1" 304 -
127.0.0.1 - - [24/Jun/2021 17:30:24] "GET /static/img/pcb_board.8c76c6688e664c0cbb8133adea263e23.webp HTTP/1.1" 304 -
127.0.0.1 - - [24/Jun/2021 17:30:39] "GET /logout/ HTTP/1.1" 302 -
127.0.0.1 - - [24/Jun/2021 17:30:39] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:30:55] "GET /login/ HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:31:42] "POST /login/ HTTP/1.1" 302 -
127.0.0.1 - - [24/Jun/2021 17:31:43] "GET / HTTP/1.1" 302 -
127.0.0.1 - - [24/Jun/2021 17:31:43] "GET /product/ HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:31:53] "GET /logout/ HTTP/1.1" 302 -
127.0.0.1 - - [24/Jun/2021 17:31:53] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:31:55] "GET /login/ HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:32:02] "POST /login/ HTTP/1.1" 302 -
127.0.0.1 - - [24/Jun/2021 17:32:02] "GET / HTTP/1.1" 302 -
127.0.0.1 - - [24/Jun/2021 17:32:02] "GET /product/ HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:33:12] "GET /profile/ HTTP/1.1" 200 -
True
True
127.0.0.1 - - [24/Jun/2021 17:33:29] "POST /profile/ HTTP/1.1" 302 -
127.0.0.1 - - [24/Jun/2021 17:33:29] "GET /product/ HTTP/1.1" 200 -
127.0.0.1 - - [24/Jun/2021 17:33:36] "GET /profile/ HTTP/1.1" 200 -
True
True
127.0.0.1 - - [24/Jun/2021 17:33:38] "POST /profile/ HTTP/1.1" 302 -
127.0.0.1 - - [24/Jun/2021 17:33:38] "GET /product/ HTTP/1.1" 200 -
^C
```

Appendix

Assignment 2:

https://drive.google.com/file/d/1dT9XK5W1VWCT-2T-UH5SiD-ryCEn7cL-/view?usp=sharing