**Wentworth Institute of Technology**

**COMP 4690 - Software Engineering**

**Software Design Document**

**For**

**FreshGo**

Version <1.2>

**Team Members**

Donna Xu

Duy Vo

Viet Hung Bui

**GitHub Link**

<https://github.com/vod6atwit/fresh_GO>

**Date:** 12/12/2022

**Contents**

1. **Introduction**
   1. **Document purpose**
   2. **Product overview**
      1. Problem statements
      2. Proposed solution
      3. Novelty
   3. **Product functionality**
2. **System Requirements**
   1. **Functional requirements**
      1. Authentication
      2. Manage accounts
      3. Manage products
      4. Manage orders
   2. **Non-functional requirements**
      1. Performance
      2. Security
      3. Compatibility
   3. **Other requirements**
      1. Development
      2. Legislative
3. **System architecture**

**3.1. Overall architecture**

**3.2. Components mapping**

3.2.1. Functional requirements

3.2.2. Non-functional requirements

**3.3. Technology stack selection**

1. **System Design**
   1. **UI**
      1. Signup
      2. Login
      3. Customers Settings
      4. Admin Settings
      5. Home Page
      6. Product Overview
      7. Customer review
      8. Product
      9. Payment
   2. **Database**
   3. **Sequence/activity diagram**
2. **References**

**Revision History**

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 11/10/2022 | 1.0 | Initial Draft | Duy Vo  Donna Xu  Viet hung Bui |
| 11/20/2022 | 1.1 | Completed System Requirement and System architecture | Duy Vo  Donna Xu  Viet hung Bui |
| 12/11/2022 | 1.2 | Final document | Duy Vo  Donna Xu  Viet hung Bui |

1. **Introduction**
   1. **Document purpose**

The System Design document (SDD) is a written report of a software product’s design, describing its overall architecture. A software design document helps to ensure the design specs of the software are understood and it’s clear to all. It specifies what is possible with the product and how it can be accomplished.

* 1. **Product overview**
     1. **Problem statements**

Technologies are getting more advanced nowadays and so is the internet. People are so used to online shopping nowadays because it’s so convenient for them to just buy things and get delivered to their homes. In 2019, Covid hitted and everyone was forced to be quarantined so they could be safe. More people were using online shopping and groceries is something that people needed to go out to shop for. It’s very convenient because without going to the store, you can get your groceries.

* + 1. **Proposed solution**

Our website application provides a lite weight, user friendly (both users and staff/owners) online shopping and management platform. This feature enables owners and staff to manage their website online remotely and simply.

* + 1. **Novelty**

Our software for groceries online will make your online shopping experience so much easier along with simple features, effortless for users to buy and pay. Besides, our software provides an all-in-one system that reduces the amount of software training time for all people within the workplace. There are different user interfaces for different account types ( users, staff, owners ). Moreover, we know that security is very important for online stores, so our software will provide the latest security functions.

* 1. **Product functionality**
* Authentication
* Manage users
* Manage products
* Manage orders

1. **System Requirements**
   1. **Function requirements**
      1. **Authentication**
         1. **Sign up**
            1. The system shall require user to deliver their information:

* Full name
* Email
* Password
* PasswordConfirm
  + - * 1. The system shall save user information into the database
        2. The system shall require the password to have at least 8 characters with at least one uppercase, lowercase and number.
        3. The system shall require the user's email to be unique.
        4. The system shall encrypt the password
        5. The system shall save password in the database
        6. The system shall send a welcome email automatically.
      1. **Login**
         1. The system shall allow registered user to log into their account by entering their profile
* Email
* Password
  + - * 1. The system shall validate email and password by using the registered user profile in the database
        2. The system shall display an error message “Incorrect username/password” to the users when logging in unsuccessfully.
        3. The system shall inform when the error occurs.
      1. **Logout**
         1. The system shall allow users to logout the account.
         2. The system shall show the message “Logout successful”.
      2. **Forgot/Reset Password**
         1. The system shall allow the user to reset their password
         2. The system shall send the reset password link to the user.
         3. The system shall allow the user to enter the new password.
         4. The system shall save the new password into the database.
    1. **Manage users**
       1. **Get user**
          1. The system shall allow admin to get user information from the database.
          2. The system shall notify users by displaying error message “You are not admin” if the users does not have AUTH to do so
       2. **Get all Users**
          1. The system shall allow admin to get all the users information in the database
          2. The system shall notify users by displaying error message “You are not admin” if the users does not have AUTH to do so
       3. **Update Users**
          1. The system shall allow the users to change their information.
* Name
* Email
  + - 1. **Delete User**
         1. The system shall allow admin to remove existing user in the database
         2. The system shall notify users by displaying error message “You are not admin” if the users does not have AUTH to do so
    1. **Manage products**
       1. **Get product**
          1. The system shall allow users to get product details
       2. **Get all products**
          1. The system shall allow users to get all the products
    2. **Manage orders**
       1. **Get order**
          1. The system shall allow users to get their history orders.
       2. **Get all orders**
          1. The system shall allow admin to get all orders from the database.
  1. **Non-functional requirements**
     1. **Performance**
        1. The system shall send email within 30 seconds after the user account is created.
        2. The system shall update or change products within 5 seconds after the change is being made.
     2. **Security**
        1. The system shall have 2 accounts type (admin/user) with different AUTH.
        2. The system shall implement the JWT Authentication method.
        3. The system shall not allow the users to change their account AUTH.
     3. **Compatibility**
        1. The system shall be able to run on Windows and MacOS successfully.
        2. The system shall be able to run on Microsoft Edge, Google Chrome (latest publicly released version), Mozilla FireFox (latest publicly released version) and Safari (latest publicly released version) successfully.
  2. **Other requirements**
     1. **Development** 
        1. The code shall be reviewed before merging to the main or develop branch.
        2. The code will not be pushed to the main or develop branch directly.
     2. **Legislative** 
        1. The system will have terms and conditions on the privacy of user data.

1. **System Architecture**
   1. **Overall architecture**

Figure 1.1 shows the overall architectural design of the FreshGo web application.

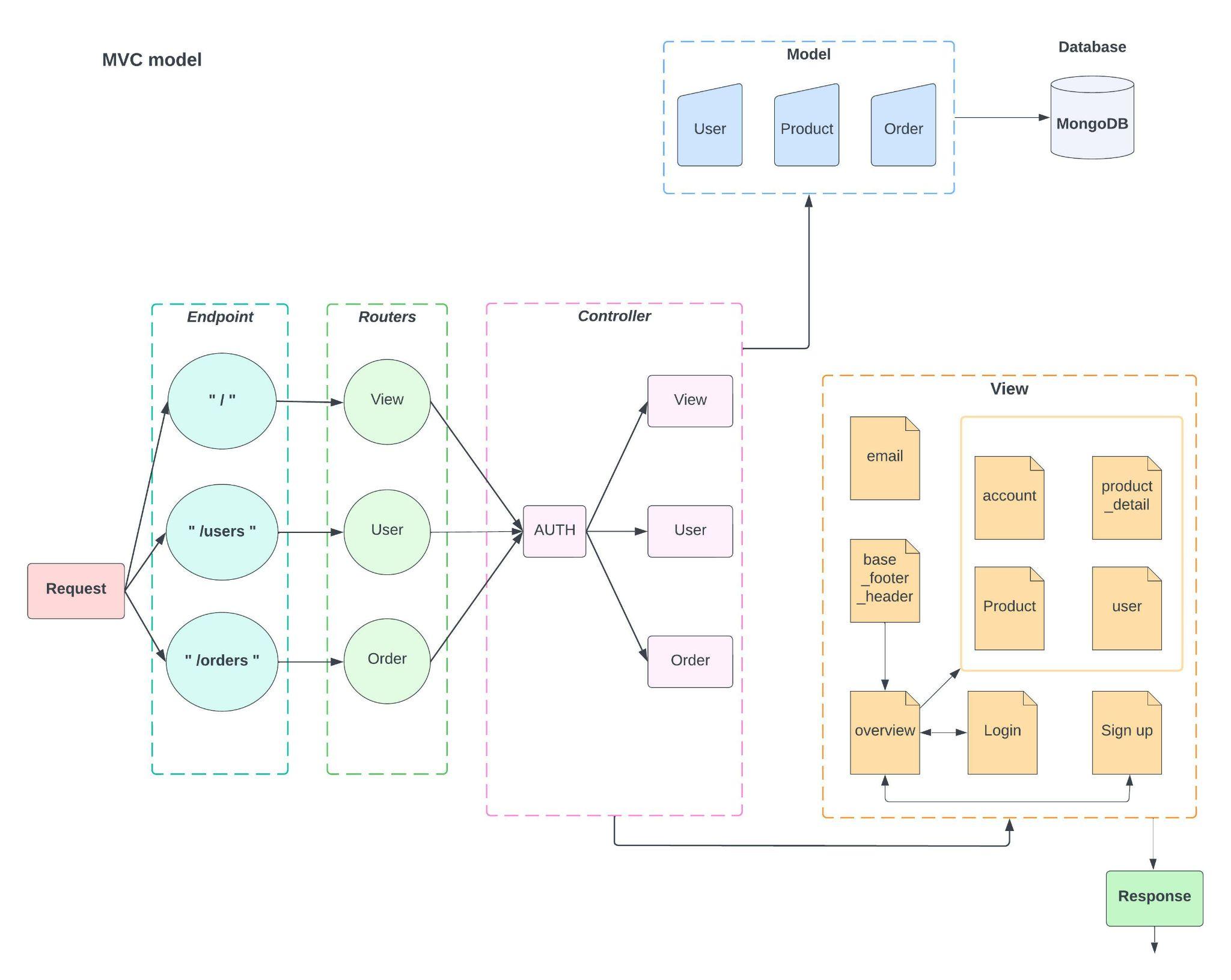


Figure 1.1 Overall MVC Architecture

MVC architectural pattern is used because:

* It is a common architectural pattern for developing web application
* It allows for multiple view for different user level
* It has the ability to change UI without interfering with the Model/Business logic.
* It allows simultaneous development among the model, view, and controller

Here are the responsibilities of each of the components in Figure 1.1:

* **Model** is responsible for representing data objects of the application. E.g., User, Product, Order
* **View** is responsible for the UI of the web application, consisting of web pages.
  + E.g.
    - Home page
    - Sign up page
    - Login page
    - Account page
    - Product page
    - User page
* **Controller** is responsible for handling all requests from users using a web browser, asking the model for the data and view for generating responses based on the data retrieved from the model.
  + UserController: handle account management. E.g., getAllUsers, getUser, updateUser, deleteUSer
  + viewController: handle product. E.g., getOverview, getLoginForm, getSignupForm, getAccount, getProductdetail, getMyOrders, getAllProducts, getAllUsers, getAllOrders
  + OrderController: handle order. E.g., getCheckoutSession, getAllOrders, getOrder, updateOrder, deleteOrder
  + authController: handle authorization E.g., signup, login, logout, protect, resetPassword, updatePassword
* **DB** is responsible for storing data. E.g., Product, User, Order
  1. **Components Mapping**
     1. **Functional requirements**
* **Authentication**

| **Design Elements** | **List of Requirements** |
| --- | --- |
| Sign Up | Generating an account using an email and password |
| Login | User signing in the account |
| Logout | User signing out the account |
| Forgot/Reset Password | Changing password for user |

* + Sign Up
    - Controller: AuthController, ViewController
    - View: Sign up Page
    - Model: UserModel
  + Login
    - Controller: AuthController, ViewController
    - View: Login Page
    - Model: UserModel
  + Logout
    - Controller: AuthController
    - View: Overview Page
    - Model: UserModel
  + Forgot/ResetPassword
    - Controller: AuthController
    - View: ResetPassword page
    - Model: UserModel
* **Manage Users**

| **Design Elements** | **List of Requirements** |
| --- | --- |
| Get User | Get an user from database |
| Get All User | Get all user from database |
| Update User | Updating information for user |
| Delete User | Delete user from database |

* + Get user
    - Controller: AuthController, UserController
    - View: Account Page
    - Model: UserModel
  + Get all Users
    - Controller: AuthController, UserController
    - VIew: Account Page
    - Model: UserModel
  + Update Users
    - Controller: AuthController, UserController
    - View: Account Page
    - Model: UserModel
  + Delete User
    - Controller: AuthController, UserController
    - View: Account Page
    - Model: UserModel
* **Manage Products**

| **Design Elements** | **List of Requirements** |
| --- | --- |
| Get Product | Get a product from database |
| Get All Product | Get all products from database |

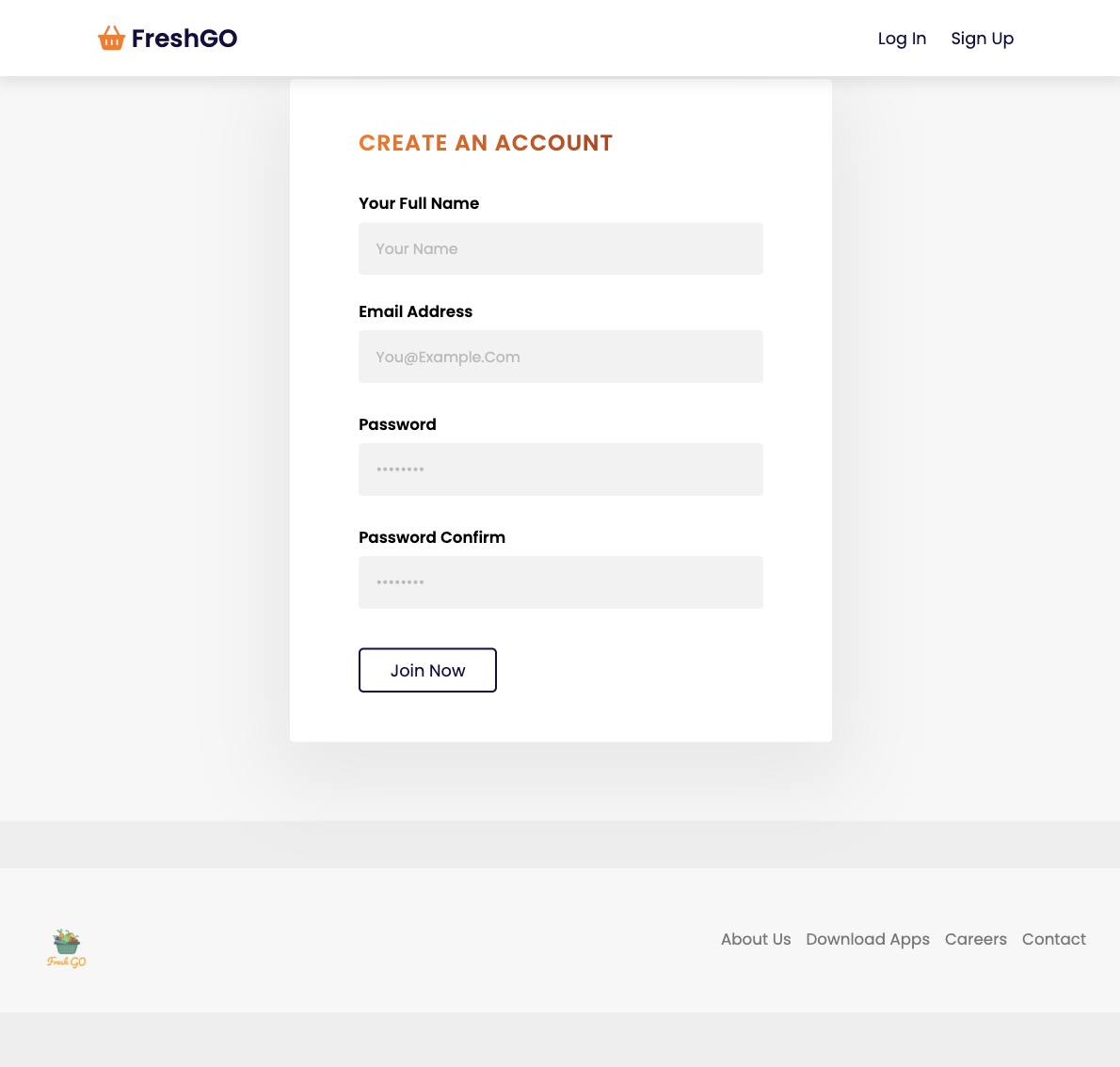
* + Get product
    - Controller: viewController
    - View: Product\_detail Page
    - Model: productModel
  + Get all products
    - Controller: viewController
    - View: Product\_detail Page
    - Model: productModel
* **Manage Orders**

| **Design Elements** | **List of Requirements** |
| --- | --- |
| Get Order | Get an user’s order from database |
| Get All Order | Get all user orders from database |

* + Get order
    - Controller: orderController
    - View: Account Page
    - Model: orderModel
  + Get all orders
    - Controller: orderController
    - View: Account Page
    - Model: orderModel
    1. **Non-functional requirements**
* For the Components mapping of these No-Functional requirements, it's all about safety and security.
  + For the Signup and Login part: customers email shall be unique, customers passwords shall have multiple restrictions for security and implement JWT authentication method.
  1. **Technology stack selection**
* **UI:** HTML, CSS
  + Light weight in structure
  + Easy to learn and use
  + Supported by all web browsers
* **View:** PUG, JS
  + Server-side rendering engine
  + Dynamically render views for the user depending on their request
  + Works well with Node.js and Express
  + Interact design and handle user behaviors
* **Web Server:** Node.js and Express
  + Easy to learn and use
  + Facilitate the creation of a web server and routes within the application
  + Help produce well-organized server-side code
* **Database:** MongoDB
  + Works well with web applications
  + Works well with MERN stack
  + Cloud Cluster creation

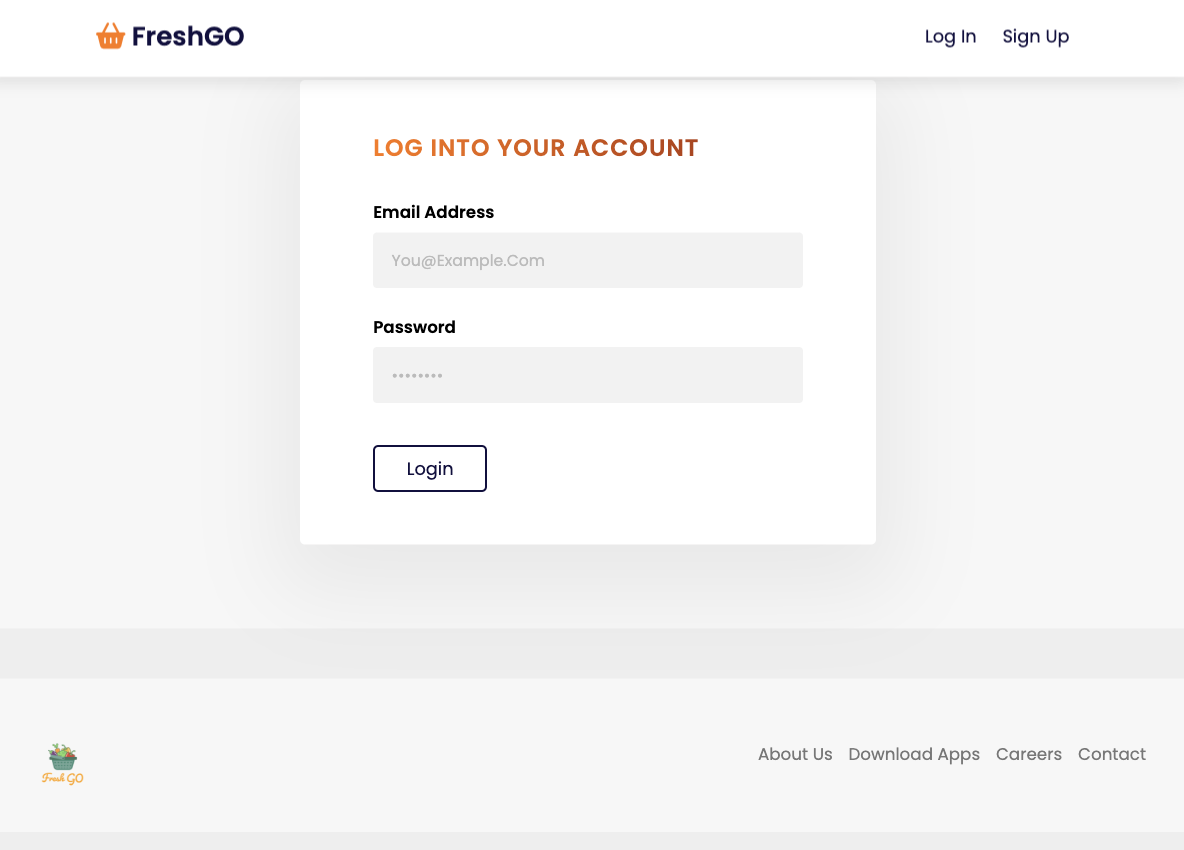
1. **System Design**
   1. **UI**
      1. **Sign up**

Registration page for new users



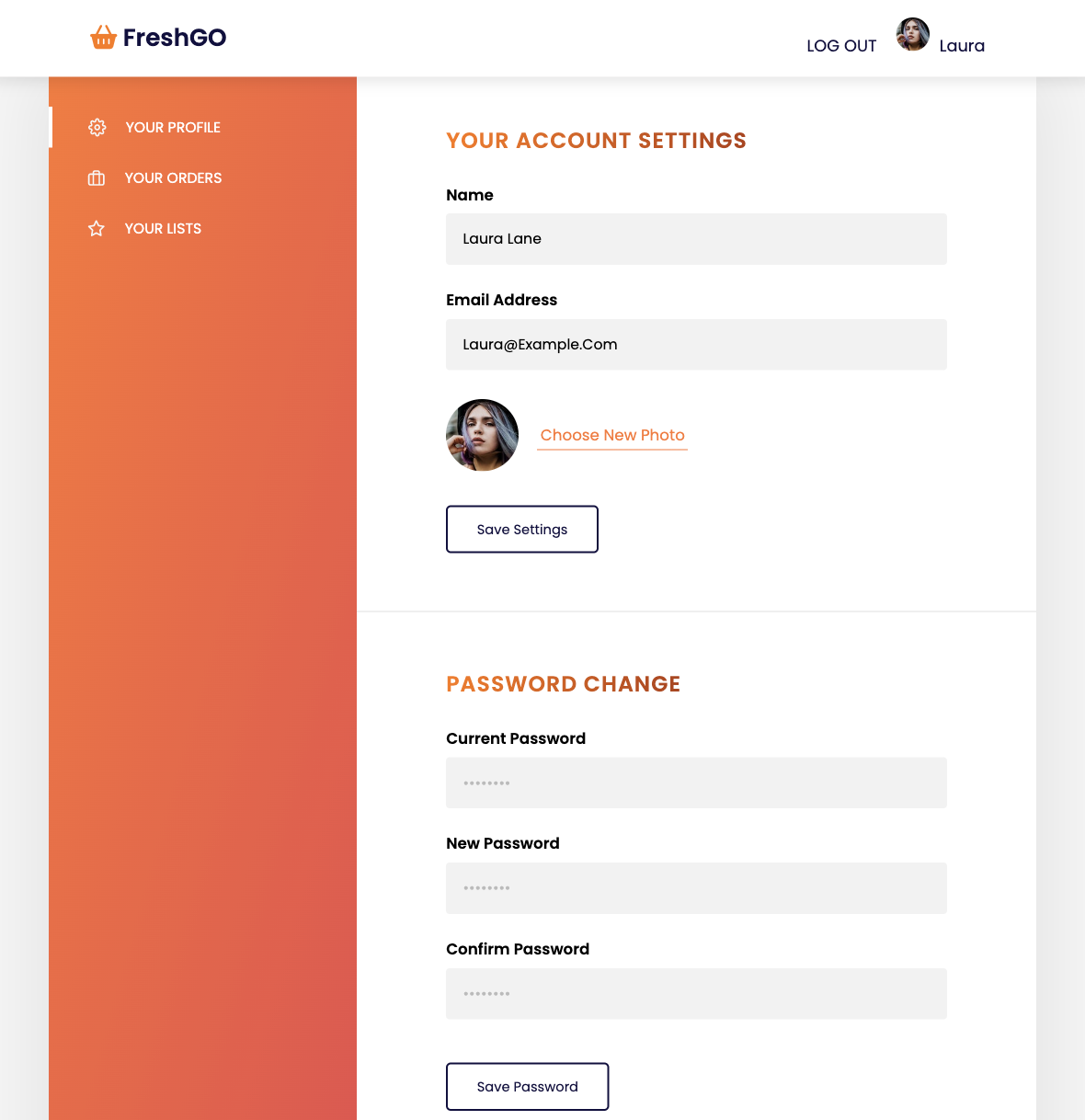
* + 1. **Login**

Login page for user account

****

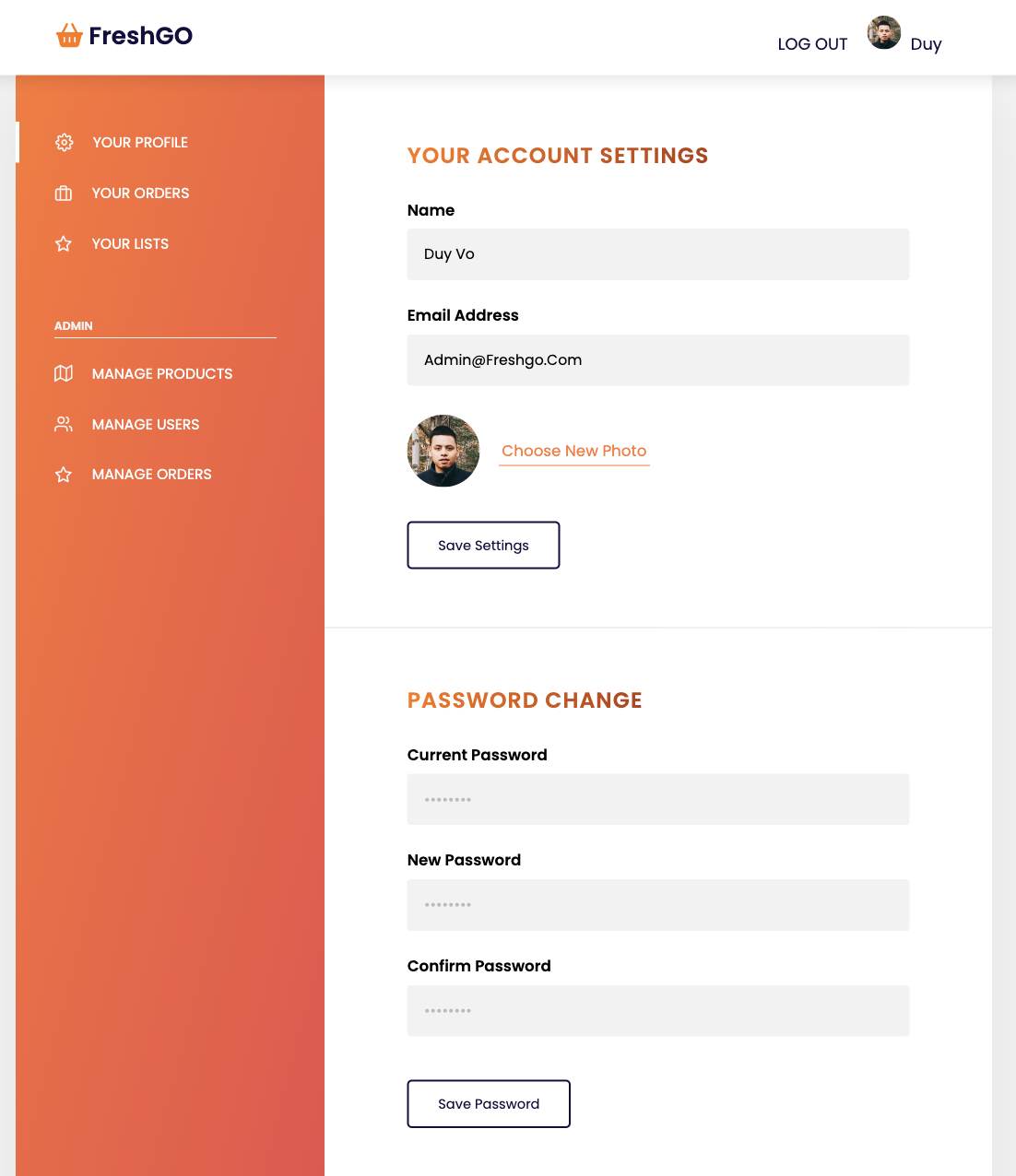
* + 1. **Customers Settings**

Customer setting page where customers can change their name and password

****

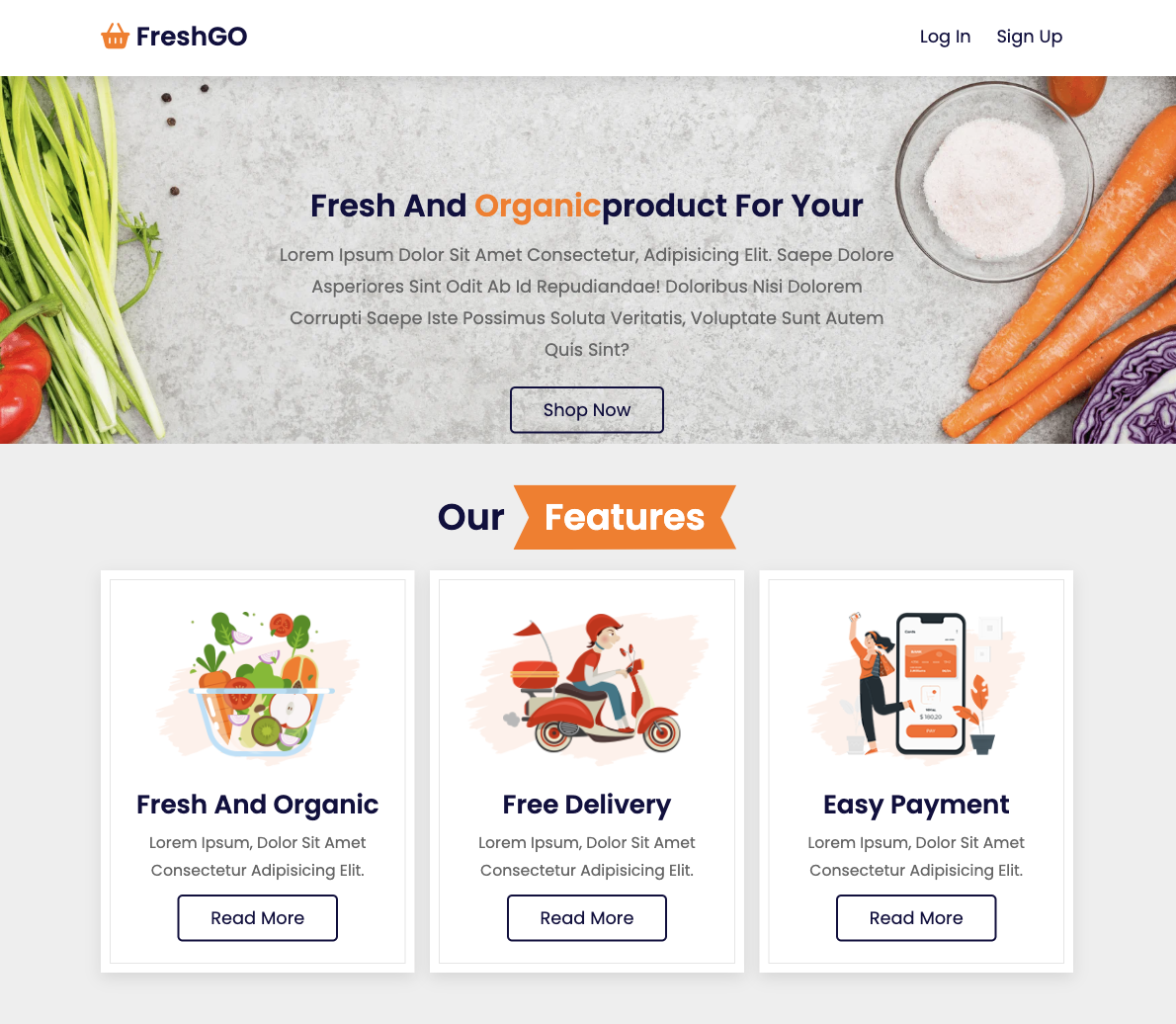
* + 1. **Admin Settings**

Admin setting page where admins can manage products and users

****

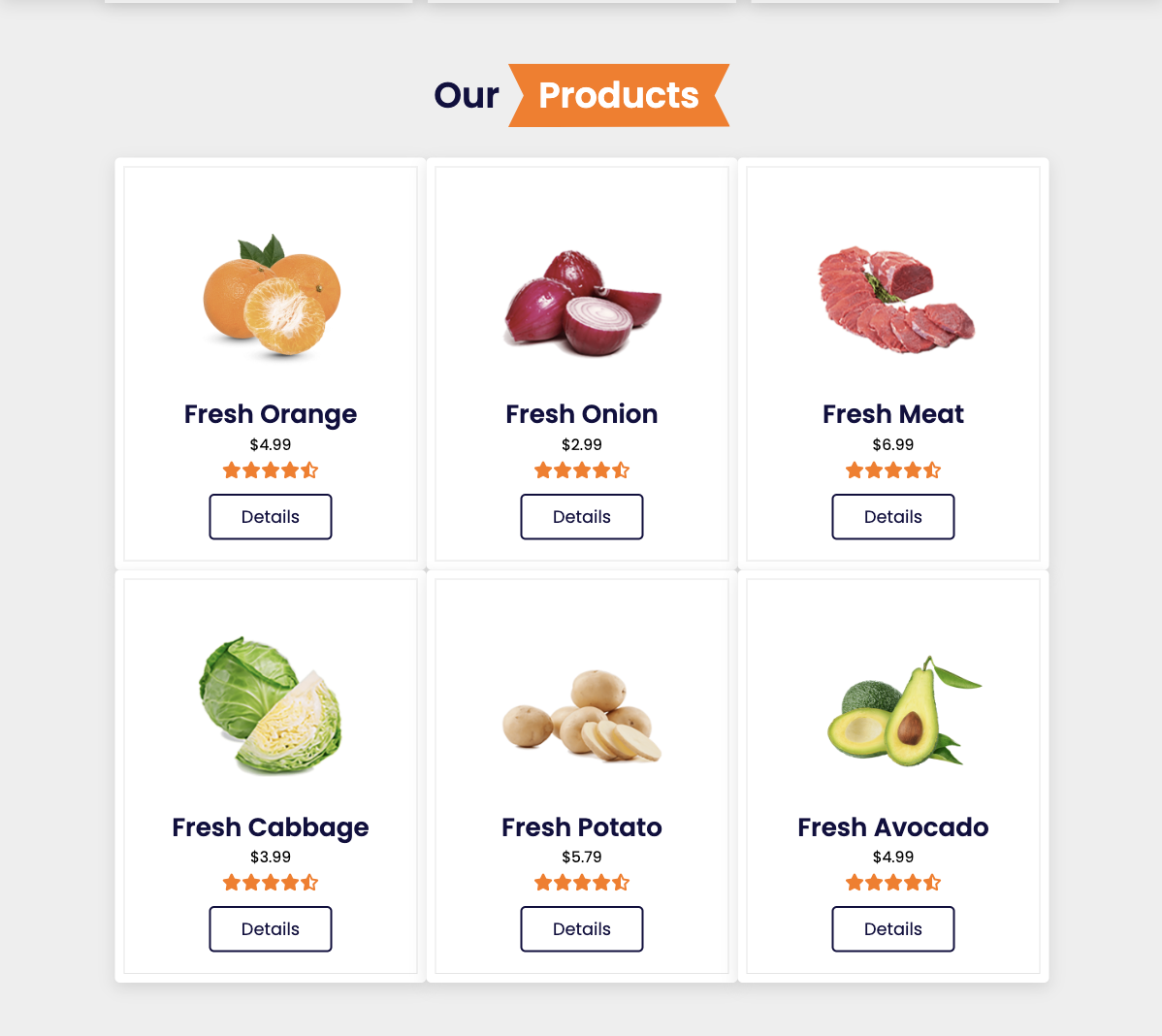
* + 1. **Home page**

Homepage of FreshGo Website

****

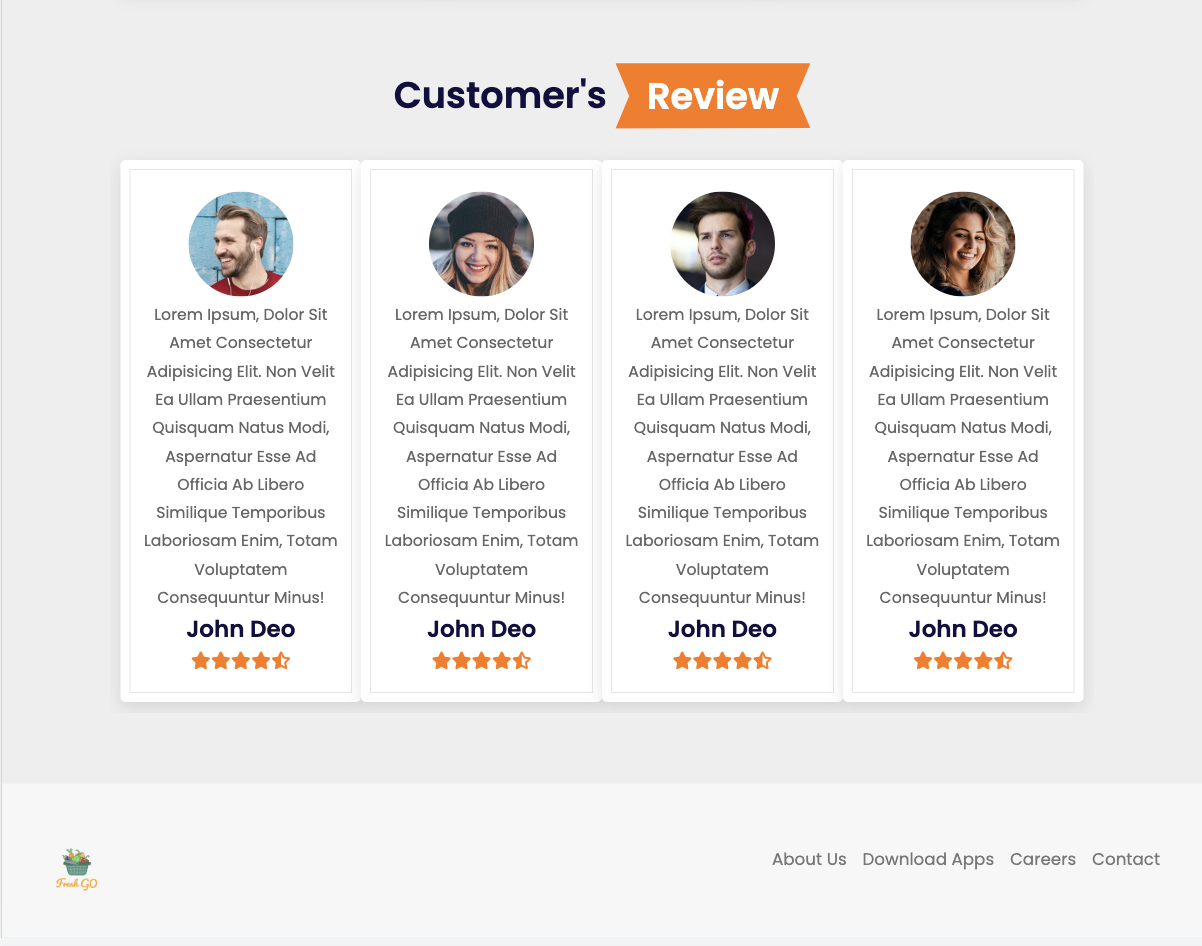
* + 1. **Product Overview**

An overview of the product page

****

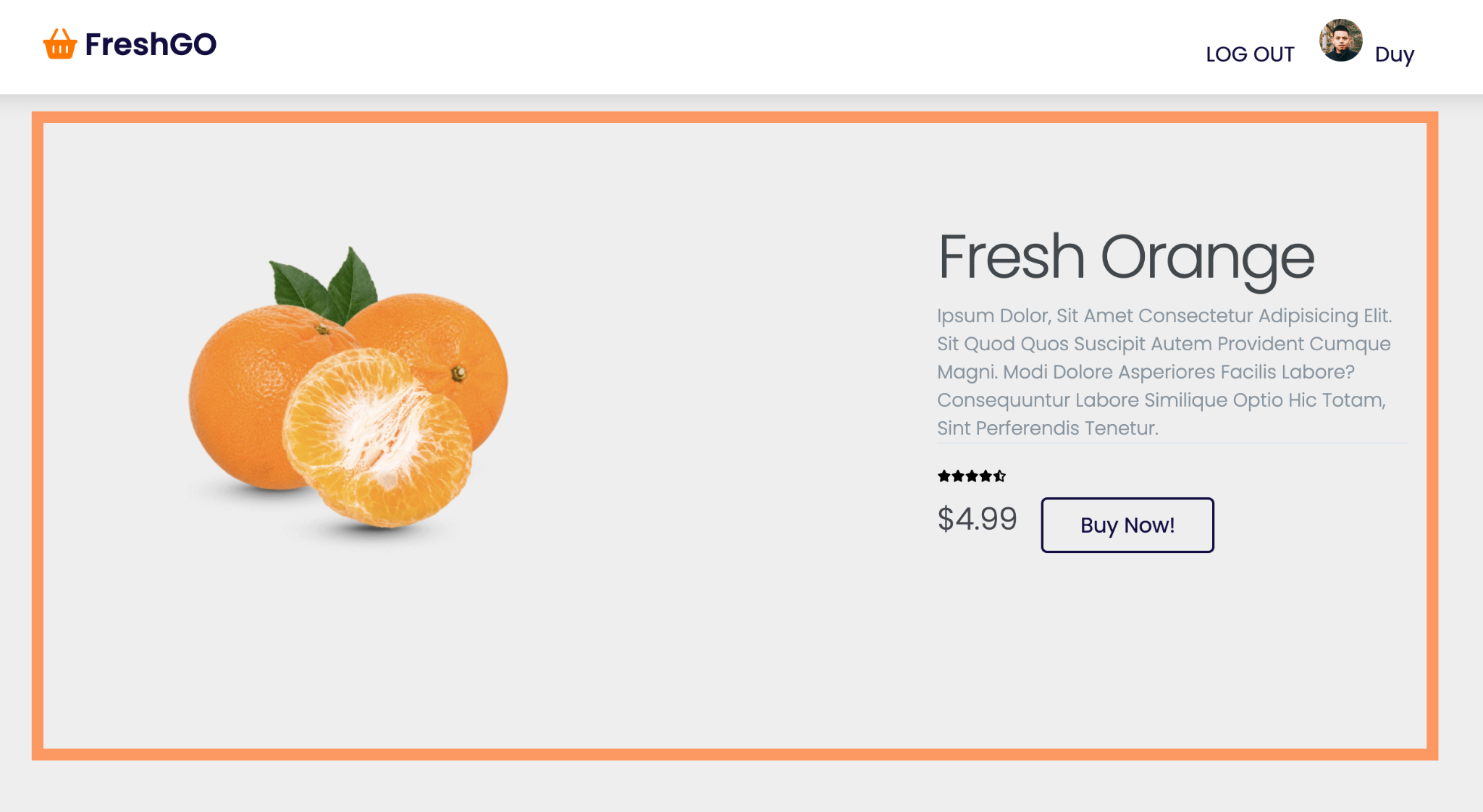
* + 1. **Customer Review**

A page of customer review for the company

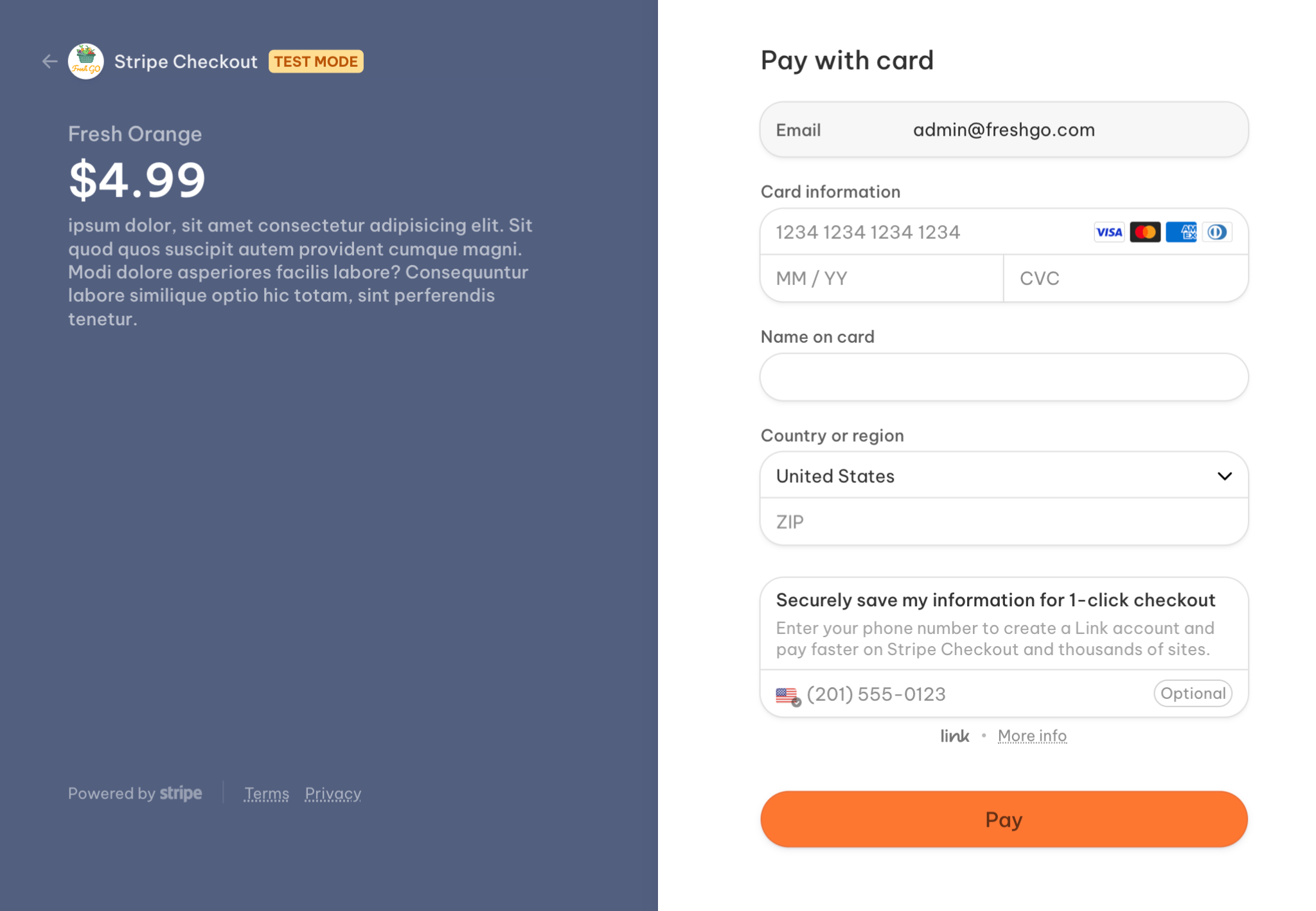
****

* + 1. **Product page**

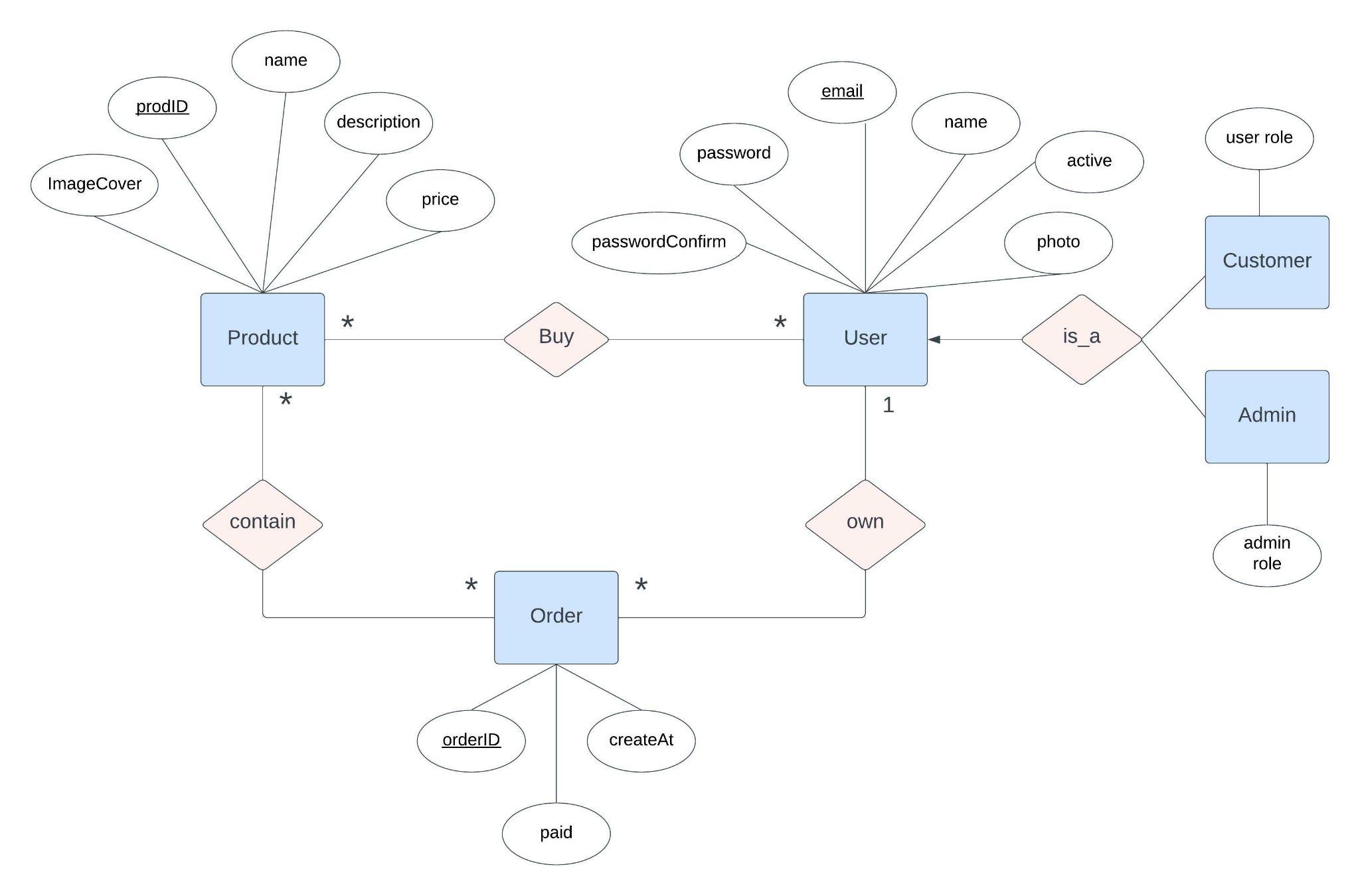
Product page of the product selected

****

* + 1. **Payment**

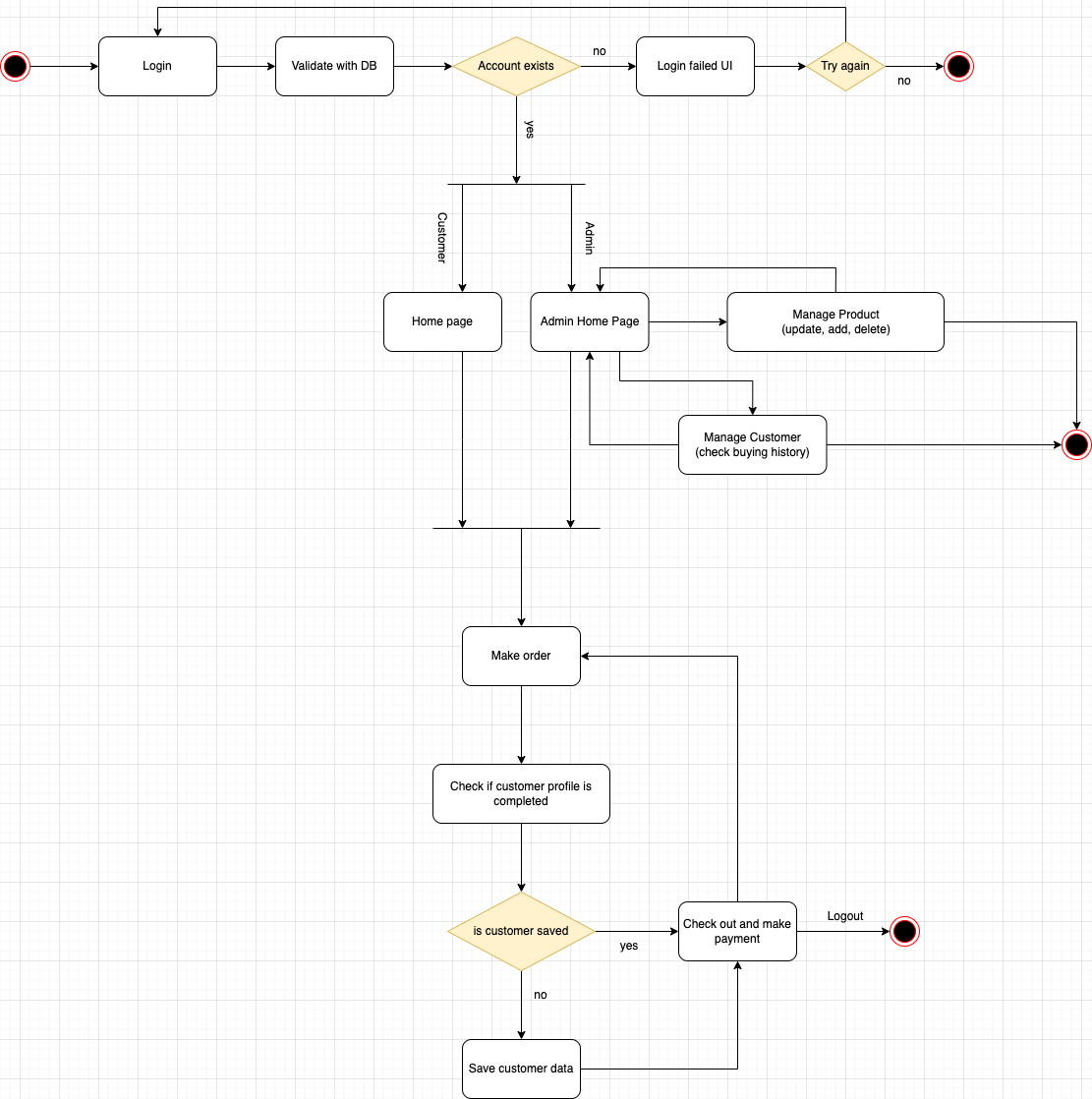
Payment Page for checking out

* 1. **Database**

****

* An entity Products
  + An attributes of entity Products:
    - prodID ( a key )
    - imageCover
    - name
    - price
    - description
* An entity Orders
  + An attributes of entity Orders:
    - orderID ( a key )
    - createAt
    - paid
* An entity Users
  + An attributes of entity Users:
    - userID
    - email ( a key )
    - password
    - passwordConfirm
    - name
    - Photo
    - active
* An is\_a relationship represents a generalization relationship between two entities (Users vs Admin, Users vs Customer)
* Entity Relationship:
  + 1 User can own Many Orders
  + 1 Order can be owned by 1 User
  + 1 User can buy Many Products
  + 1 Products can be bought by Many Users
  + 1 Orders can contain Many Products
  + 1 Products can be contained in Many Orders
  1. **Sequence/activity diagram**

This is the Sequence activity diagram for our project. As shown in the diagram, the program starts with the home page UI. After inputting the username and password, the program shall validate with the database and if the user exists. It shall have two options, either login as the admin or the customer. If the login failed, the user shall be asked to try again, and the login page shall still appear. If a user login successfully as a customer, they shall have the overview page to order the items that they want to buy, and they shall have the setting page to edit their profile. If a user login successfully as an admin, they shall have the manage page, they will have the ability to manage products, manage users. They also have the ability to manage orders such as checking customer’s buying history.

**Diagram:**  

1. **References**
2. <https://app.diagrams.net/>
3. <https://www.lucidchart.com/>