CSCI 5308
Advanced Topic in Software Development

Project Documentation

Group 23
Development Team of: Be The Donor

Prof: Tushar Sharma

Team Members

Manali Shah B00890746
Prachi Raval B00883324
Dharmik Soni B00867641
Dhairya Shah B00900984
Jayshree Ramasubramaniam B00894948

Project Documentation

Overview of Be The Donor

Be the Donor website is made aiming to assist covid sufferers in obtaining their necessities. They might find themselves in not a good financial situation or won't be able to go to buy essentials by themselves. To solve this problem, patients can pick which of their necessary things will be visible to order. Following that, anyone can help them by becoming a donor and making a financial contribution towards patients. Donors can assist many patients, depending on their needs. They also can donate anonymously towards the cause. Riders can assist patients by bringing essentials to their location of which the donations have been made.

Flow of Be The Donor

Users must register on the site by giving relevant information and specifying their user type, such as Patient, Donor, Rider, or Admin. The patient will be brought to a page where they can choose the item and other necessities that they require. The order would then be placed in the donor section, where it would be available for donations. The patient can check the status of their order, such as whether it is in "Order placed and pending payment," "pending delivery," "ready to deliver," or "delivered" status. The orders placed by the patients are made accessible for donation on the donor page. Donors can donate to any order or donate anonymously, of which the amount will be automatically assigned to orders based on the time and total amount of order. The orders that have been paid for will be displayed in the rider's section enabling riders to choose whether to deliver or not. The rider would also be given a tip to assist them.

Technologies Used:

Frontend: Html, CSS, JS, Thymeleaf
Backend: Java
Framework: springboot
Architecture: MVC layered architecture
Database: mysql
Hosted on: Heroku

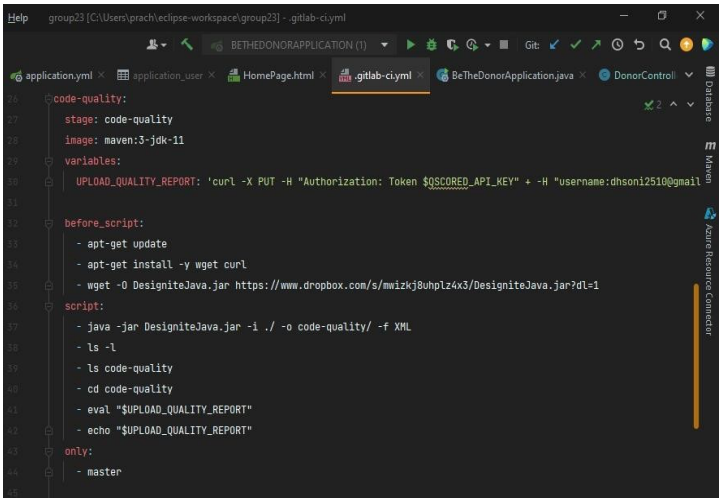
CI/CD Stages:

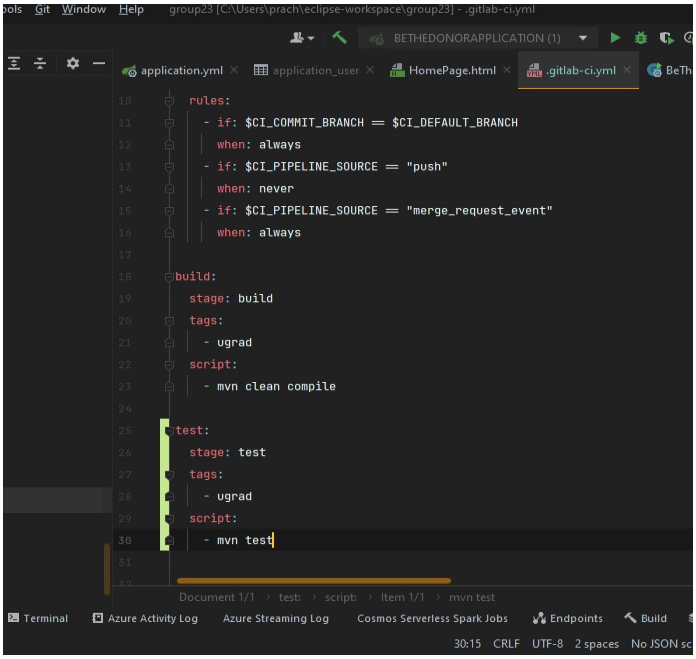
We have set the following stages for the CI/CD pipeline

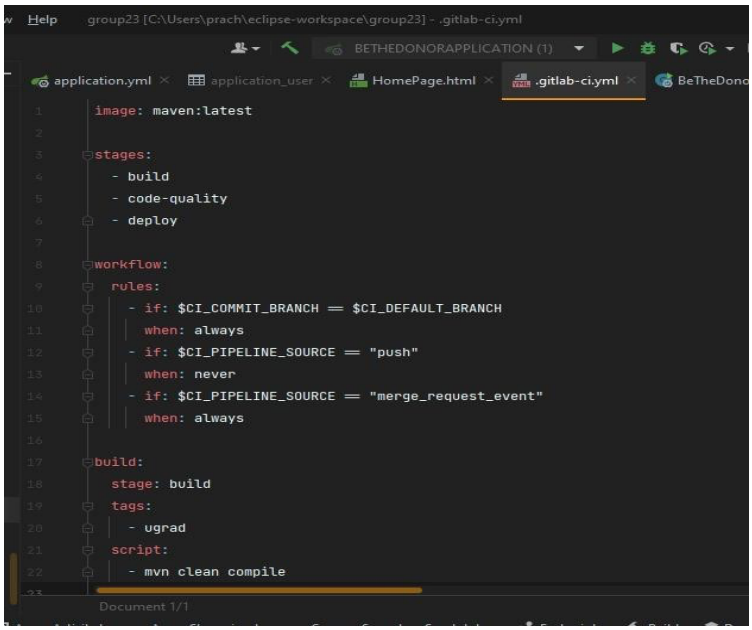
Build – To run our application perform the following command

Test – To test our application we have setup a test pipeline

Code Quality – By running this code, we generate a Qscore document for code smells. Deploy – Deployment on Heroku with this command.







Dependencies:

The following are the ***Internal Dependencies*** used for building this project:

- Spring Boot JPA

```
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-data-jpa</artifactid> </dependency>
```

- Spring Boot Mail

```
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-mail</artifactid> </dependency>
```

- Spring Boot Security

```
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-security</artifactid> </dependency>
```

- Gson

```
<dependency>
  <groupid>com.google.code.gson</groupid>
  <artifactid>gson</artifactid>
  <version>2.9.0</version>
</dependency>
```

- Npm

```
<dependency>
  <groupid>org.webjars.npm</groupid>
  <artifactid>bootstrap-autocomplete</artifactid>
  <version>2.3.7</version>
</dependency>
```

- Persistence API

```
<dependency>
  <groupid>javax.persistence</groupid>
  <artifactid>persistence-api</artifactid>
  <version>1.0.2</version>
</dependency>
```

- Mysql Connector

```
<dependency>
  <groupid>mysql</groupid>
  <artifactid>mysql-connector-java</artifactid>
  <scope>runtime</scope>
</dependency>
```

- Project Lombok

```
<dependency>
  <groupid>org.projectlombok</groupid>
  <artifactid>lombok</artifactid>
  <optional>true</optional>
</dependency>
```

- Spring Boot Test

```
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-test</artifactid>
  <scope>test</scope>
</dependency>
```

- Spring Security Test

```
<dependency>
  <groupid>org.springframework.security</groupid>
  <artifactid>spring-security-test</artifactid>
  <scope>test</scope>
</dependency>
```

- Spring Boot Thymeleaf

```
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-thymeleaf</artifactid> </dependency>
```

- Spring Boot Validation

```
<dependency>
  <groupid>org.springframework.boot</groupid>
                    <artifactid>spring-boot-starter-validation</artifactid>
</dependency>
```

- Model Mapper

```
<dependency>
  <groupid>org.modelmapper</groupid>
  <artifactid>modelmapper</artifactid>
  <version>2.4.5</version>
</dependency>
```

- JAVAX Blind

```
<dependency>
  <groupid>javax.xml.bind</groupid>
  <artifactid>jaxb-api</artifactid>
  <scope>runtime</scope>
</dependency>
```

- Spring Framework

```
<dependency>
  <groupid>org.springframework</groupid>
  <artifactid>spring-context</artifactid>
  <version>5.3.16</version>
</dependency>
```

- Spring Framework Boot

```
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-devtools</artifactid> </dependency>
```

- Spring Framework Security Core

```
<dependency>
  <groupid>org.springframework.security</groupid>
  <artifactid>spring-security-core</artifactid>
  <version>5.6.2</version>
</dependency>
```

- Spring Framework Security Crypto

```
<dependency>
  <groupid>org.springframework.security</groupid>
  <artifactid>spring-security-crypto</artifactid>
  <version>5.6.2</version>
</dependency>
```

- Webjars

```
<dependency>
  <groupid>org.webjars</groupid>
  <artifactid>webjars-locator</artifactid>
  <version>0.42</version>
</dependency>
```

- JSON

```
<dependency>
  <groupid>com.googlecode.json-simple</groupid>
  <artifactid>json-simple</artifactid>
  <version>1.1.1</version>
</dependency>
```

- JJWT

```
<dependency>
  <groupid>io.jsonwebtoken</groupid>
  <artifactid>jjwt</artifactid>
  <version>0.2</version>
</dependency>
```

- Bootstrap

```
<dependency>
  <groupid>org.webjars</groupid>
  <artifactid>bootstrap</artifactid>
  <version>5.1.1</version>
</dependency>
```

- Jupiter API

```
<dependency>
  <groupid>org.junit.jupiter</groupid>
  <artifactid>junit-jupiter-api</artifactid>
  <scope>test</scope>
</dependency>
```

- Jupiter Engine

```
<dependency>
  <groupid>org.junit.jupiter</groupid>
  <artifactid>junit-jupiter-engine</artifactid>
  <scope>test</scope>
</dependency>
```

- Thymeleaf

```
<dependency>
  <groupid>org.thymeleaf</groupid>
  <artifactid>thymeleaf</artifactid>
  <version>3.0.11.RELEASE</version>
</dependency>
```

- Thymeleaf Spring

```
<dependency>
  <groupid>org.thymeleaf</groupid>
  <artifactid>thymeleaf-spring5</artifactid>
  <version>3.0.11.RELEASE</version>
</dependency>
```

- Junit Launcher

```
<dependency>
  <groupid>org.junit.platform</groupid>
  <artifactid>junit-platform-launcher</artifactid>
  <scope>test</scope>
</dependency>
```

- Junit Engine

```
<dependency>
  <groupid>org.junit.vintage</groupid>
  <artifactid>junit-vintage-engine</artifactid>
  <scope>test</scope>
</dependency>
```

- H2Database

```
<dependency>
  <groupid>com.h2database</groupid>
  <artifactid>h2</artifactid>
  <scope>test</scope>
</dependency>
```

- Stripe Java

```
<dependency>
  <groupid>com.stripe</groupid>
  <artifactid>stripe-java</artifactid>
  <version>20.112.0</version>
</dependency>
```

- Mockito

```
<dependency>
  <groupid>org.mockito</groupid>
  <artifactid>mockito-inline</artifactid>
  <version>3.8.0</version>
  <scope>test</scope>
</dependency>
```

- Spring Boot Web

```
<dependency>
  <groupid>org.mockito</groupid>
  <artifactid>mockito-inline</artifactid>
  <version>3.8.0</version>
  <scope>test</scope>
</dependency>
```

- Spring Boot JSON

```
    <exclusion>
      <groupid>org.springframework.boot</groupid>
      <artifactid>spring-boot-starter-json</artifactid>
    </exclusion>
  </exclusions>
</dependency>
```

Plugins:

- Spring Boot Maven Plugin

Build and Deployment:

- mvn clean install

Use Case Diagram

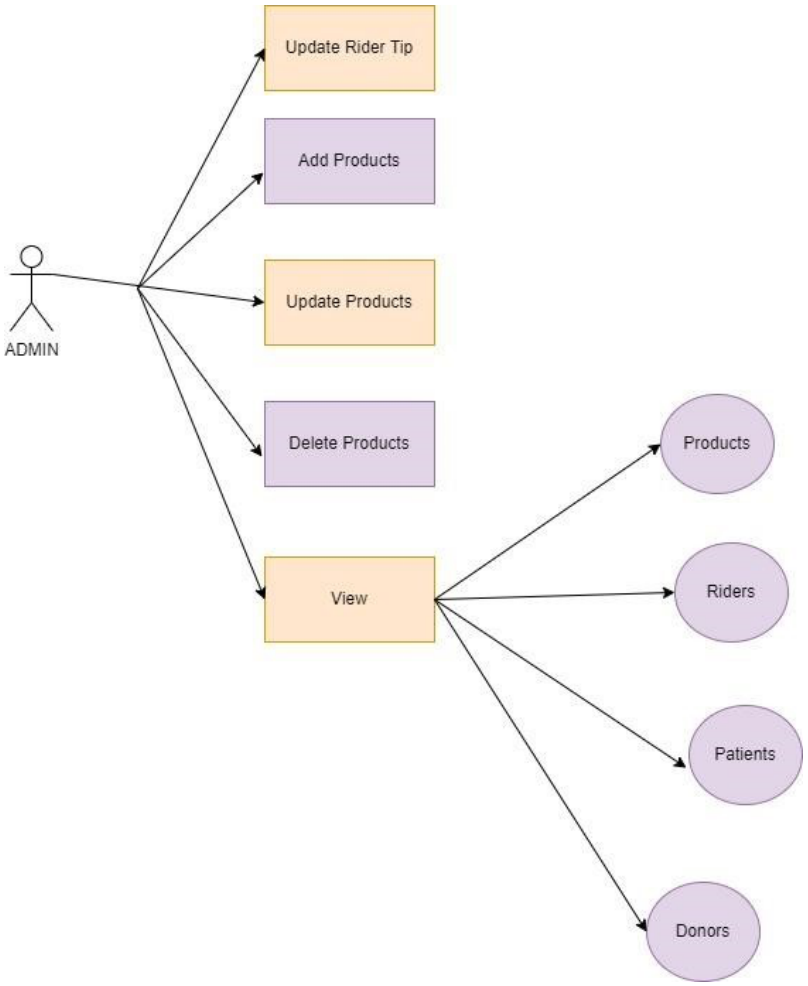


Figure 1

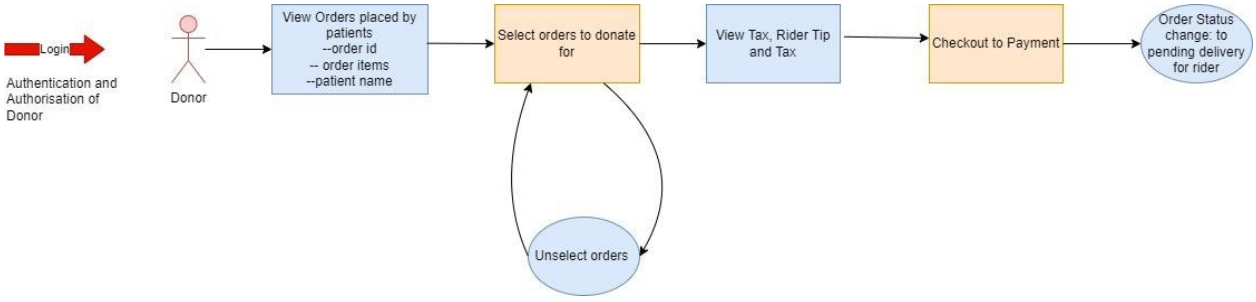


Figure 2

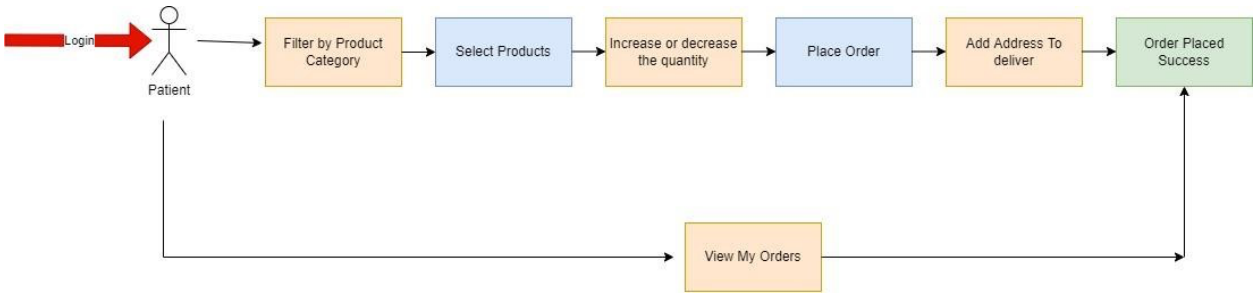


Figure 3

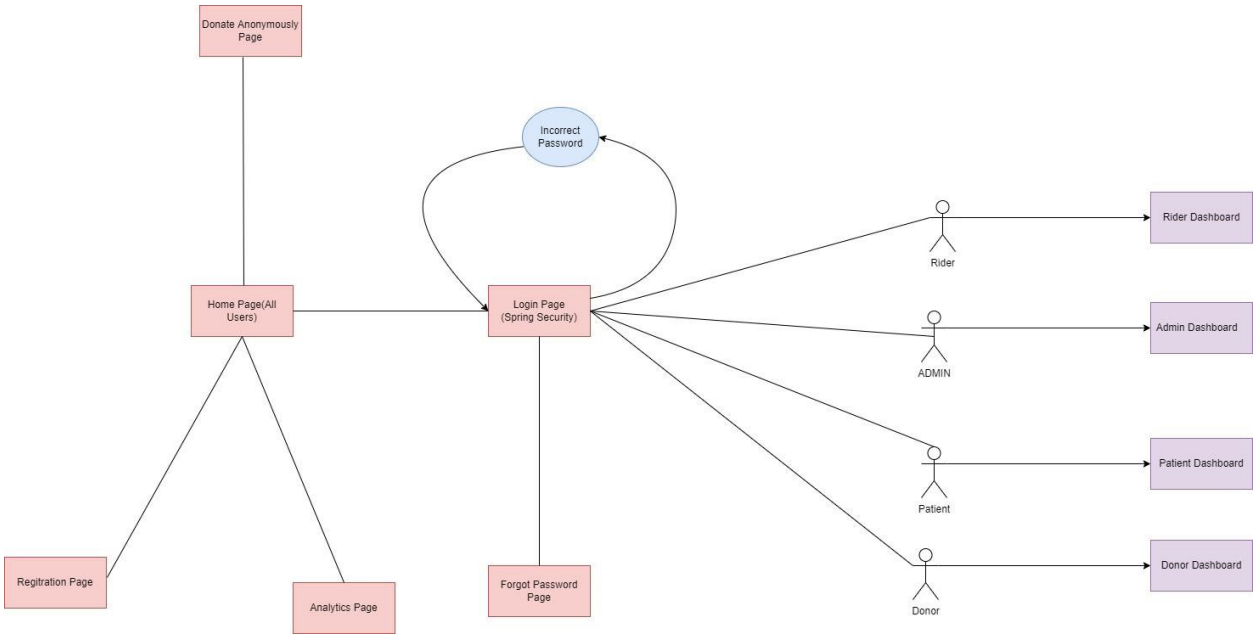


Figure 4

User Scenarios:

Use Case	1
Scenario	Registration
Input	First Name, Last Name, Email, Phone Number, Role, Password, Confirm Password
Output	User must receive a verification mail.
Description	As part of registering on the application, users must enter all valid details in-order to receive a verification mail to register successfully.

Use Case	2

Scenario	Verification Mail
Input	Receive an email
Output	Confirm by verifying on the email and get register successfully.
Description	After verification on email, the user gets register successfully and would be able to login into the application.

Use Case	3
Scenario	User Login Authorisation
Input	Email Id, Password
Output	User must log in successfully based on the role they registered as.
Description	A role base log in would be done with spring security. According to the user role, user would be directed to respective role pages and won't be able to access pages of different roles.

Use Case	4
Scenario	User Login Authentication
Input	Email Id, Password
Output	Successful login based on correct credentials
Description	Checks the password and username

Use Case	5
Scenario	Admin Page – Add Products
Input	Product Name, Quantity, Price, Category, Comment
Output	Product added in database.
Description	Admin can add products in the database as per the inventory available. Adding these products would make these products available to be displayed and selected on the patient module screen for ordering purpose.

Use Case	6
Scenario	Admin Page – Update Product
Input	Quantity, Price
Output	Quantity or price or both of product would be updated on the basis of input.
Description	Admin can update the details of product such as quantity, price of individual products according to the inventory he/she has.

Use Case	7
Scenario	Admin Page – Delete Product
Input	Click of delete product
Output	Product would be deleted

Description	Admin can delete the product according to the need. If a particular product is not present in the repository and if admin wants to delete that particular product, then admin can do so by clicking the delete button for particular product.

Use Case	8
Scenario	Admin Page – View Patients, Donors, Riders
Input	Null
Output	Would be able to view list and details of patients, donors and riders.
Description	Admin has the rights to view the details of users who have registered as patient, donor, or rider. Each of the 3 roles have different tables in which the details are stored and can be viewed by the admin.

Use Case	9
Scenario	Admin Page – Update Rider Tip
Input	Rider tip percentage
Output	The tip percentage of the rider would be updated.
Description	When a donor donates anonymously, the tip for the rider is not defined. So, this helps in giving a pre-defined tip percentage of the total amount to the rider.

Use Case	10
Scenario	Patient Page – Categories
Input	Select categories
Output	Products on display will be displayed on the basis of the category they belong to.
Description	Patients can look upon products based on the categories they belong to, so as to make the selection of products easier for the patients.

Use Case	11
Scenario	Patient Page – Select products
Input	Quantity of products
Output	Total amount of products chosen will be displayed.
Description	As patient, patient can select the items and essentials they want to order from the list of products on the display list.

Use Case	12
Scenario	Patient Page – Delivery Address
Input	First Name, Last Name, Address, Country, City, Province, Postal Code
Output	The delivery address for the patient would be set.
Description	After the patients select the items and essential to order, they are directed to a delivery page where details of address is taken so as to save the address of the patient where the order is to be delivered. This would be reflected in the rider module.

Use Case	13
Scenario	Patient Page – Order Request Success
Input	Click continue
Output	Order request placed successfully.
Description	After the patient sets the delivery address, the order request has been placed successfully and is made available on the donor section for donation.

Use Case	14
Scenario	Donor Page – Show Orders
Input	Select order to donate
Output	Grand total including subtotal, tax and rider tip will be displayed for the donor to donate.
Description	Donor can select single/multiple orders to donate to. This is auto populating the amount fields according to the order(s) amount. Also, total donations made previously by this particular donor person is displayed on the top section.

Use Case	15
Scenario	Donor Page – Payment Gateway
Input	Card Number, Expiry Date, CVV, Email Address
Output	The total amount will be paid from the card of the donor towards the donation amount.
Description	After selecting the order(s) to help, the donor can move to the payment gateway to make the payment towards the donation. By providing the card details and email, the donor can make the donation and these orders will be available on the riders section to deliver.

Use Case	16
Scenario	Rider Page – Search by location
Input	Location
Output	Orders placed for delivery at this particular location will be showed.
Description	Riders who have logged in can search orders to deliver based on the location they chose to deliver to.

Use Case	17
Scenario	Rider Page – Show items
Input	Click show items
Output	The total number of items, their name and quantity will be showed.
Description	Every shown order which has been placed has an option to see the items in the order details. Clicking the show items will open a side box to show all the items in that order.

Project Flow and snippets

Home

The project flow starts from the Home Page. Home page gives an overview of project strucutre and features. Using the home page all the user's would be able to view the details of the avaialble features depending on the role. User's can also donate anonymously by clicking on the link on home page. User's can also donate anonymously by clicking on the link on home page.
User's can click on register and sign in from the home page. Registration
is done based on email id and user role.

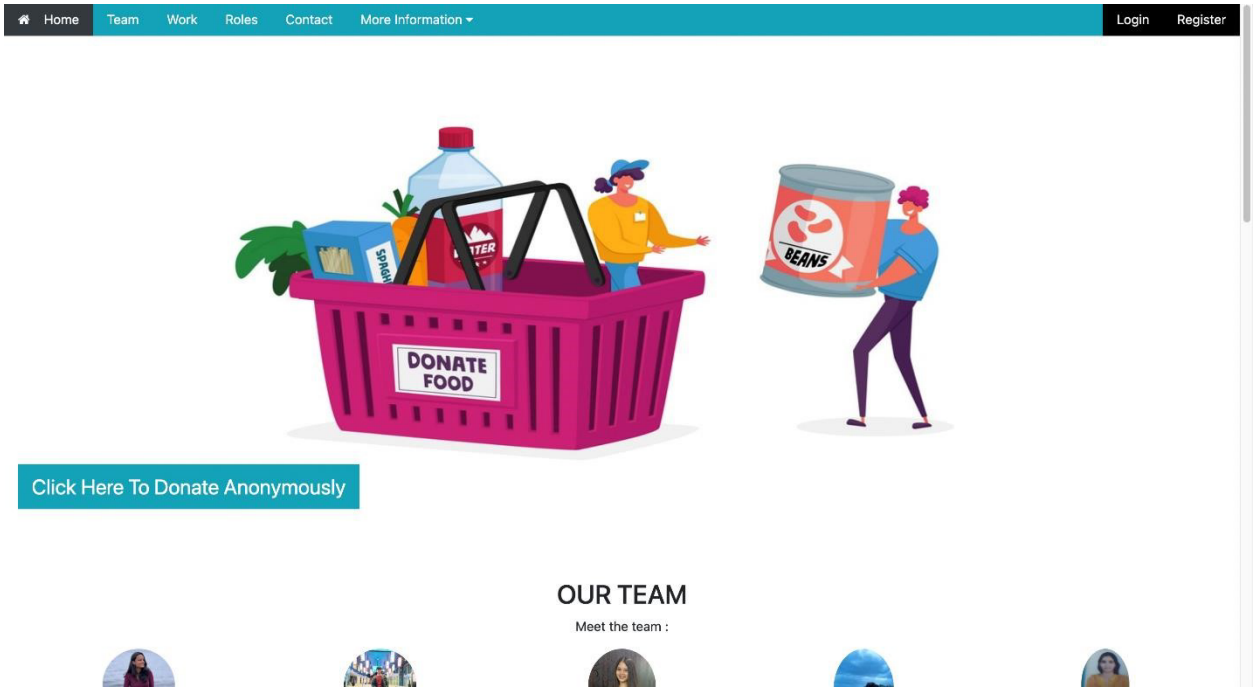


Figure 5

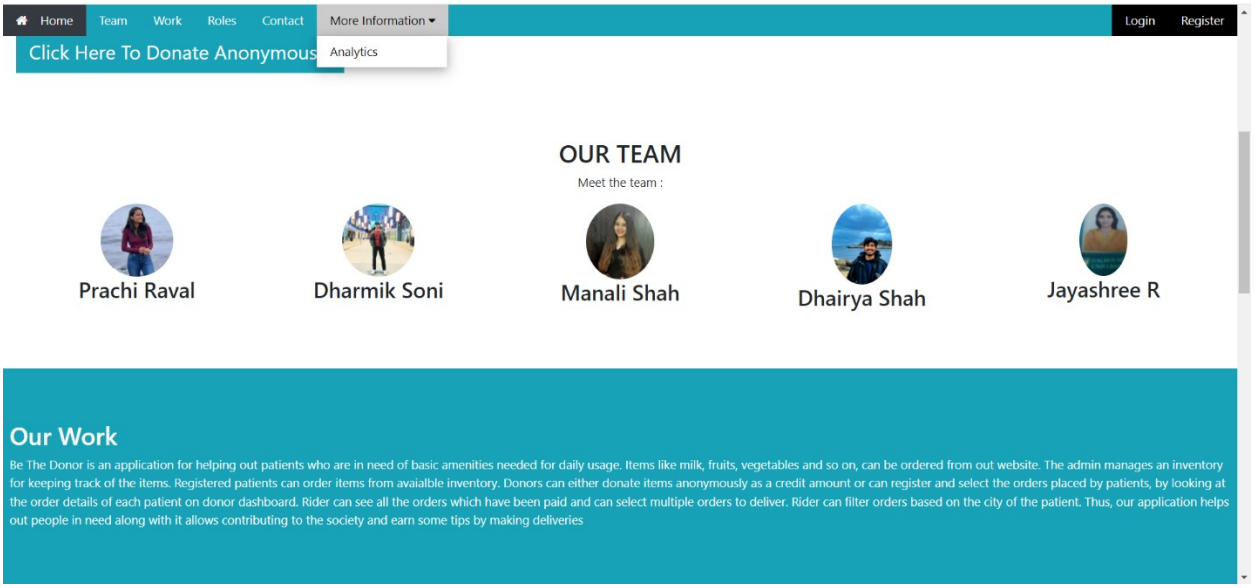


Figure 6

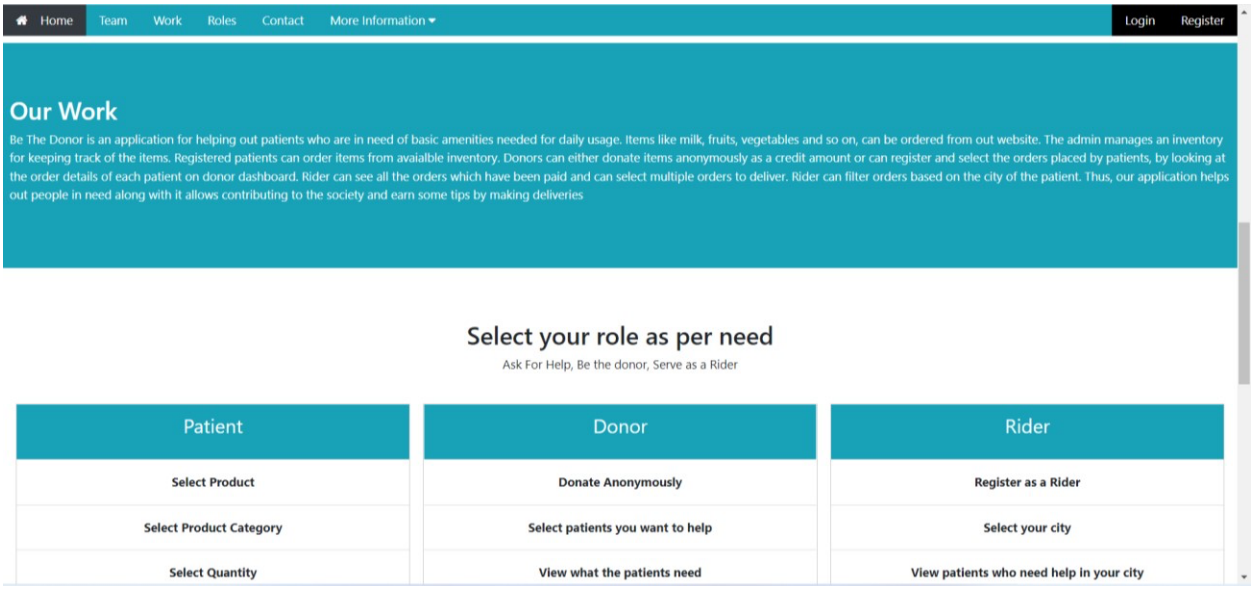


Figure 7

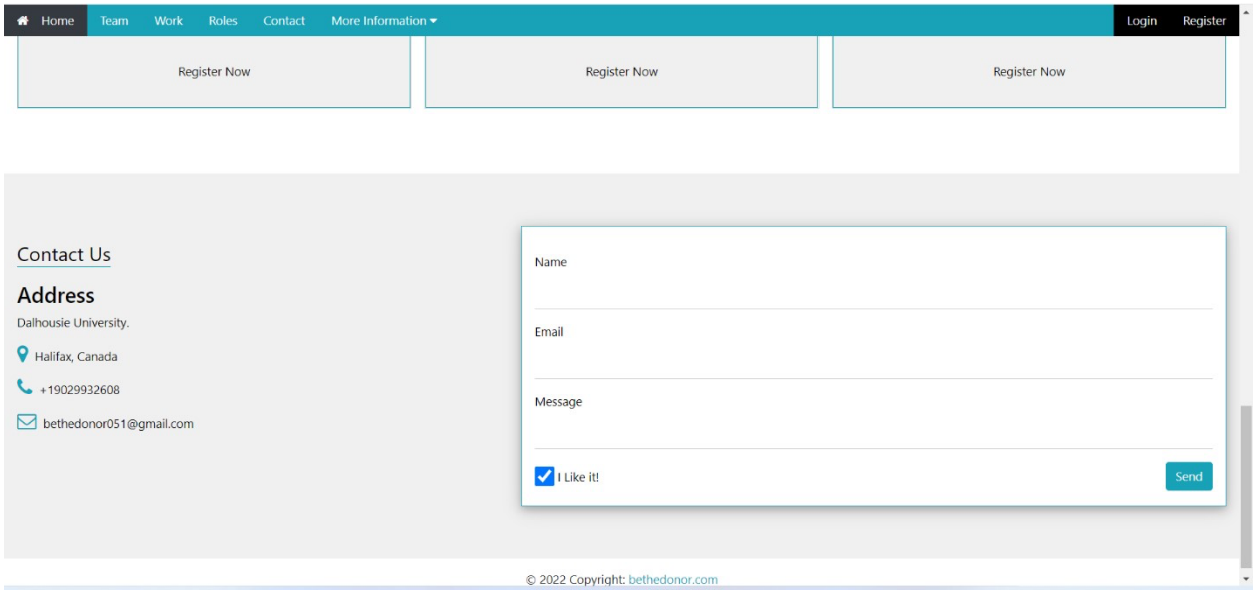


Figure 8

Sign Up

After registering user will recieve a confirmation email to activate their account. We have stored the password in encrypted form.

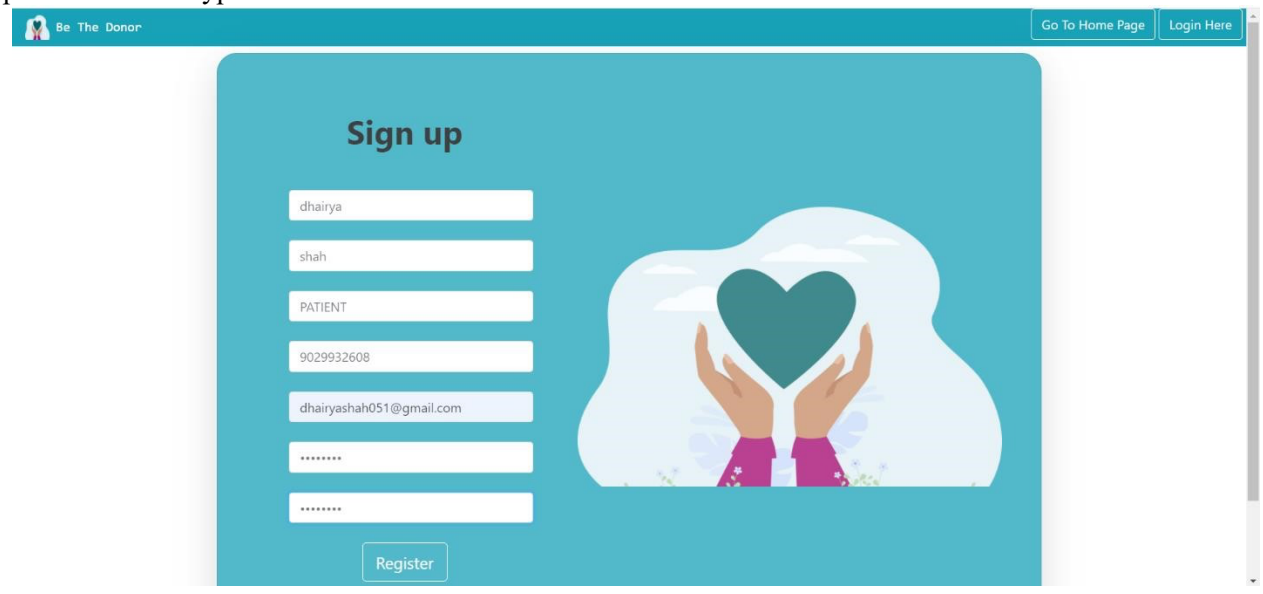


Figure 9

Verification mail

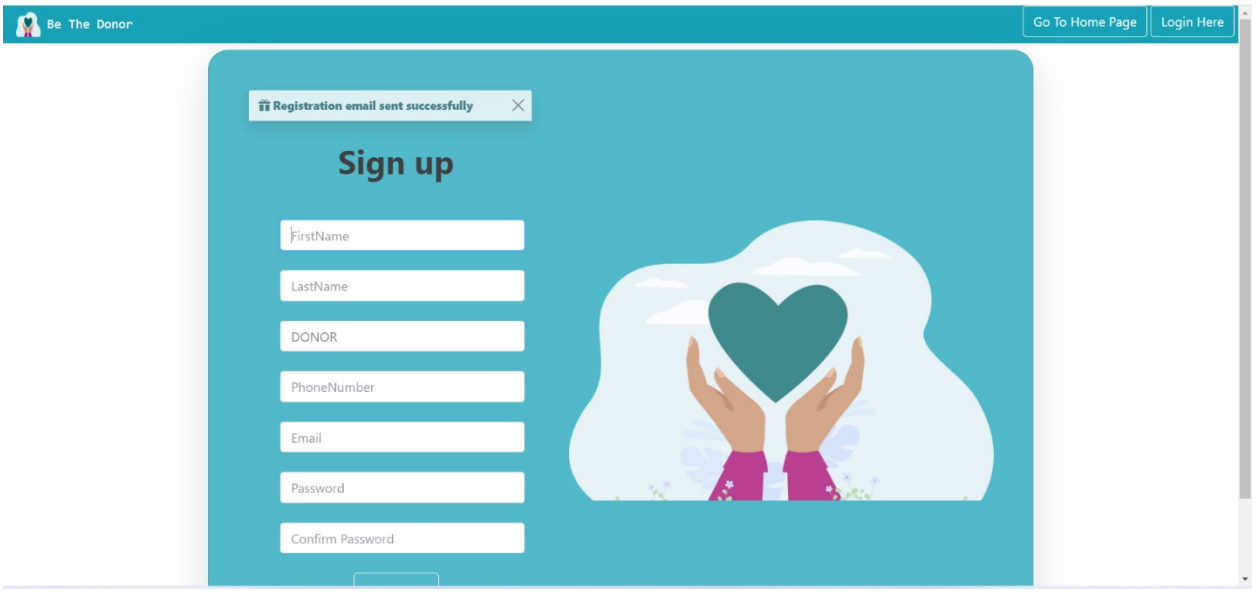


Figure 10

Verification Mail
After registering user will recieve a confirmation email to activate their account.

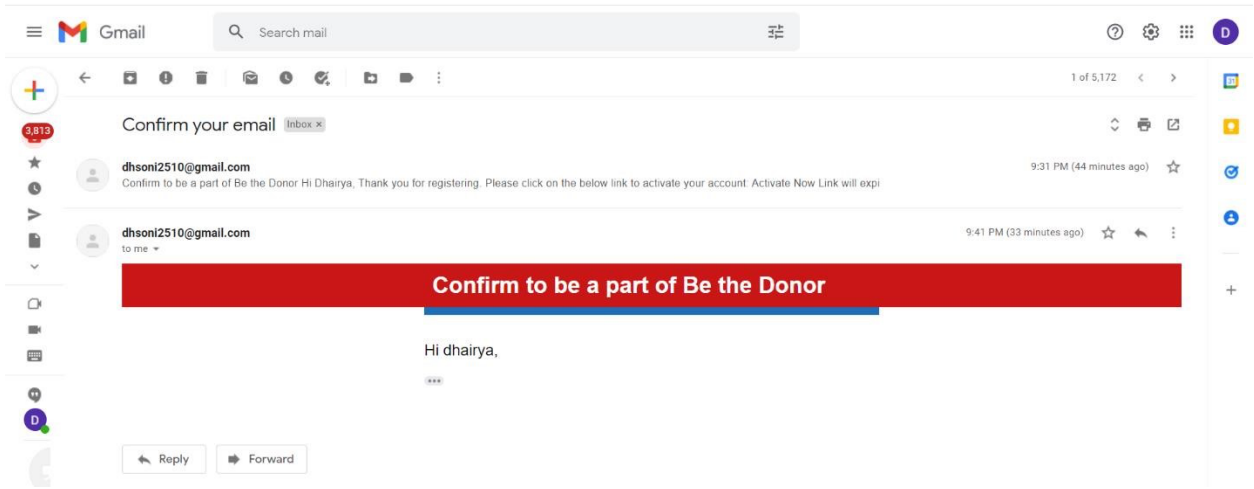


Figure 11

Login

For login and session magaement we used jwt initially, but it could not support the token for multiple roles of user. Thus, we later used spring security which is used to login based on user role. There are 3 roles - rider , donor, and patient. Each type of user would only be able to login to their respective views and will not be able to access other views. The spring handles both user authorisation and authentication.

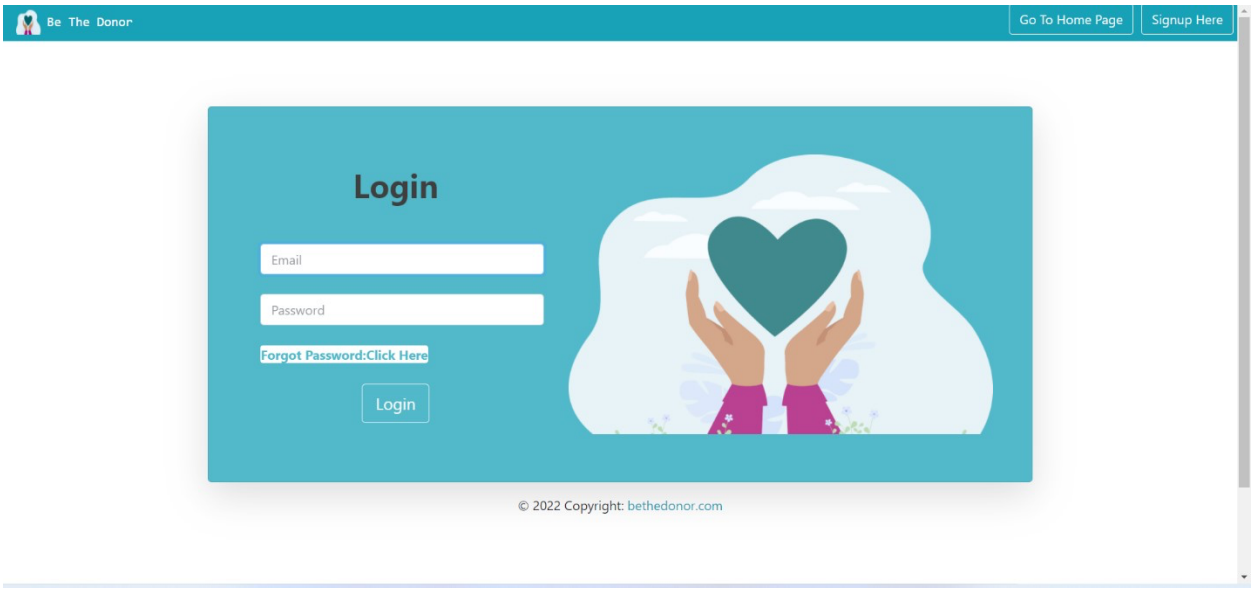


Figure 12

Admin

Admin can add products, delete them and update the quantity based on the inventory avaiablable. Admin has the authority to view the avaiable patients , riders, and donors registered.

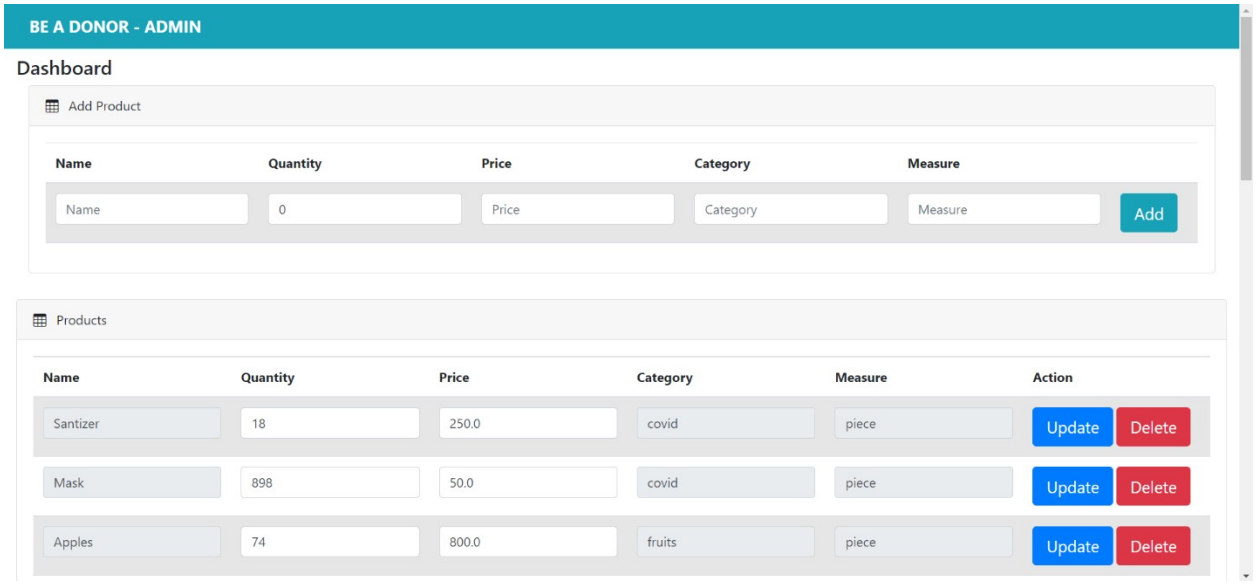


Figure 13

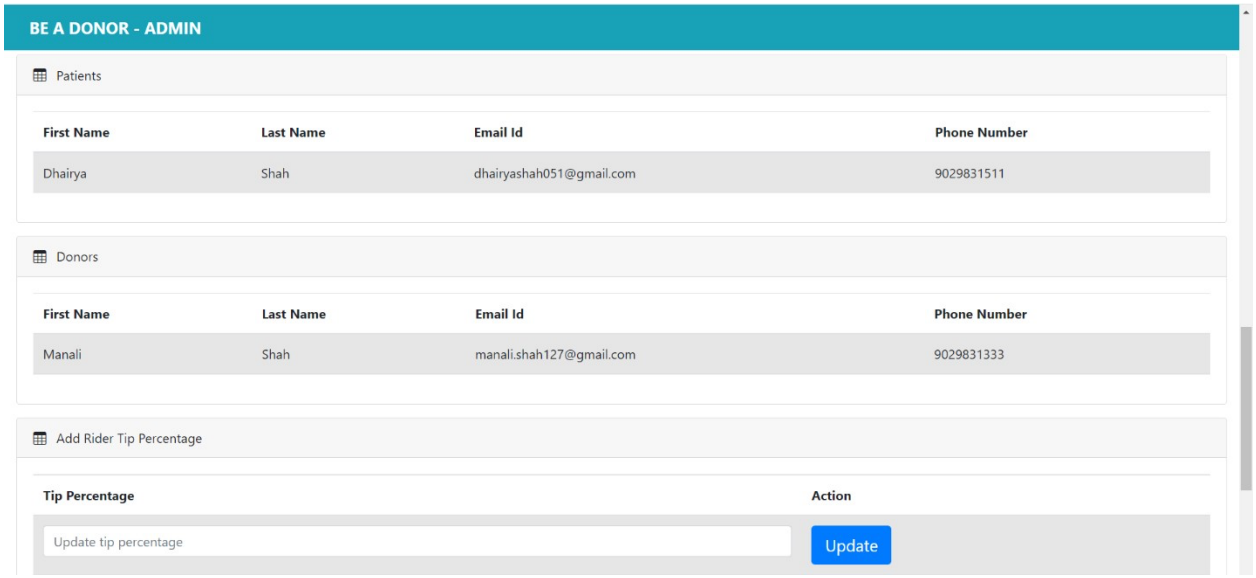


Figure 14

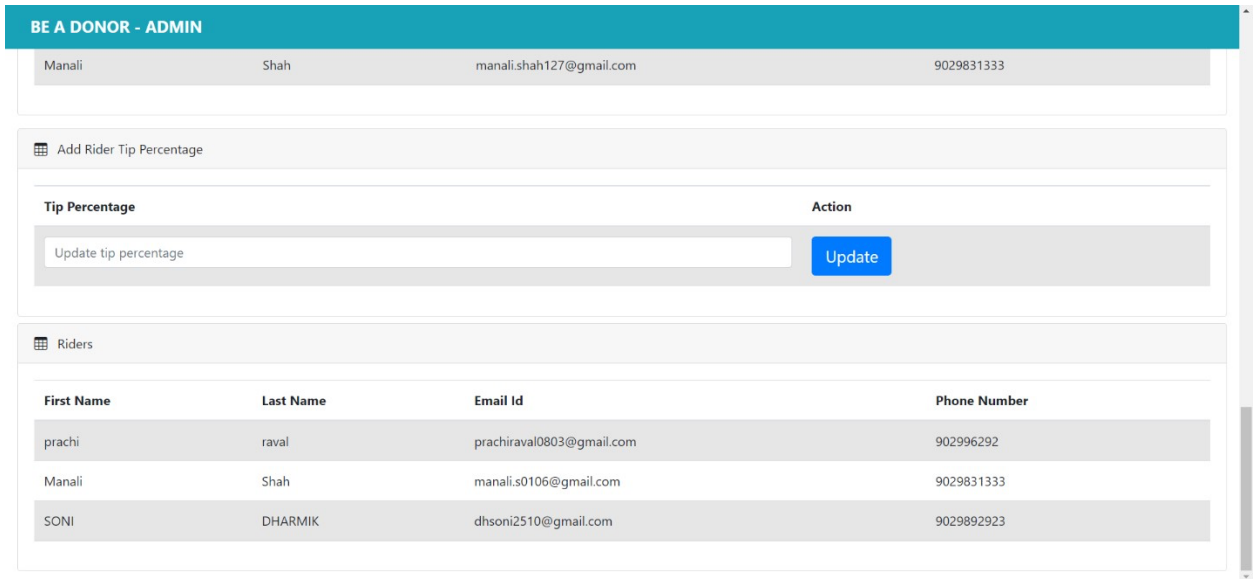


Figure 15

Patient

After logging in the patient will be able to view the available orders and can order products by filtering the category. After that patient enters their address and then places the orders. Patients can also view their older orders on my orders page

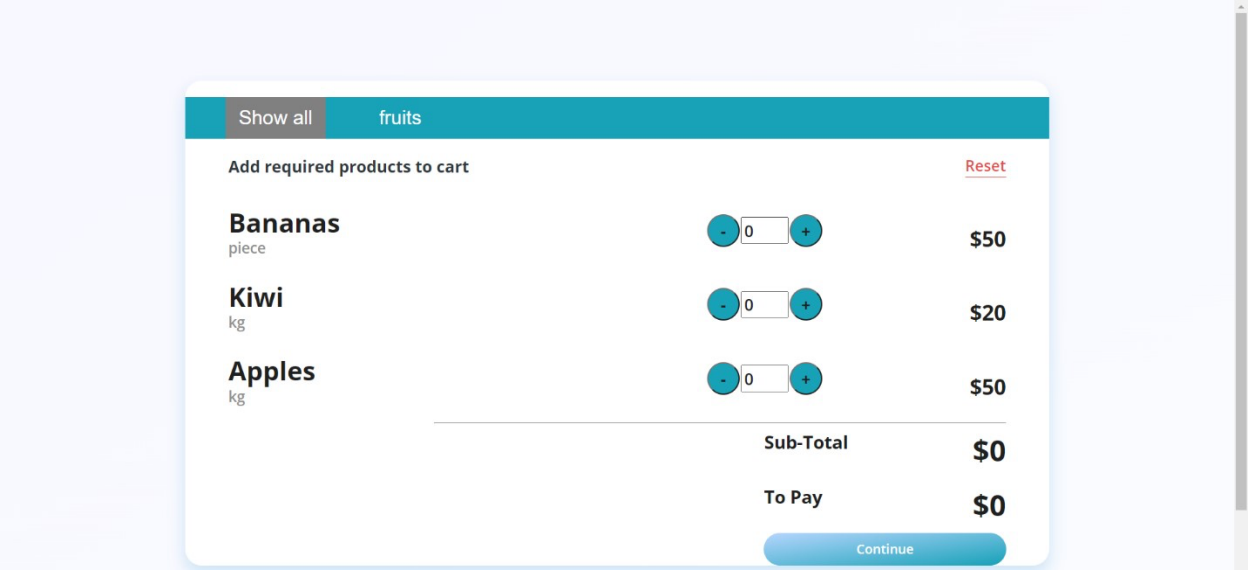


Figure 16

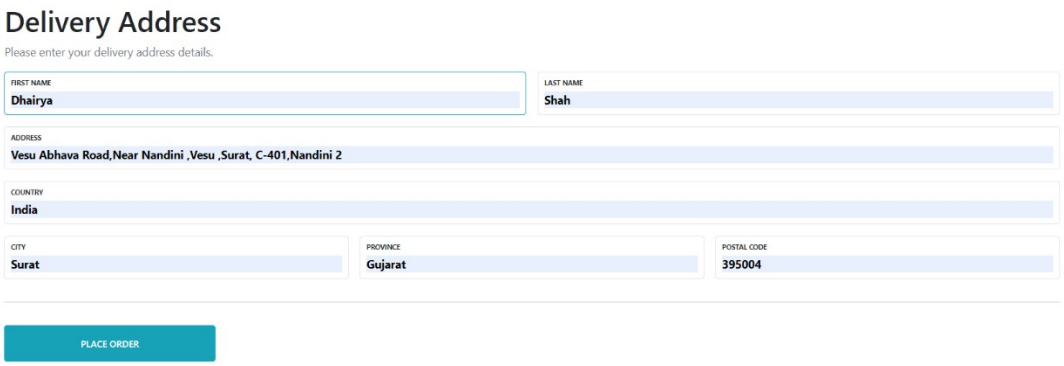


Figure 17

Donor

After logging in donors can view the available orders, which have not been paid off. Donors can see the order id, patient name, order products and total amount of the orders. Donors can select and unselect the orders and view the total amount with tax and rider tip added. Donor will click on pay and will be redirected to the payment gateway. After that the order status will change to opening deliver for the rider to view.
After logging in the particular donor will be able to view total donations amount made by them in the past.

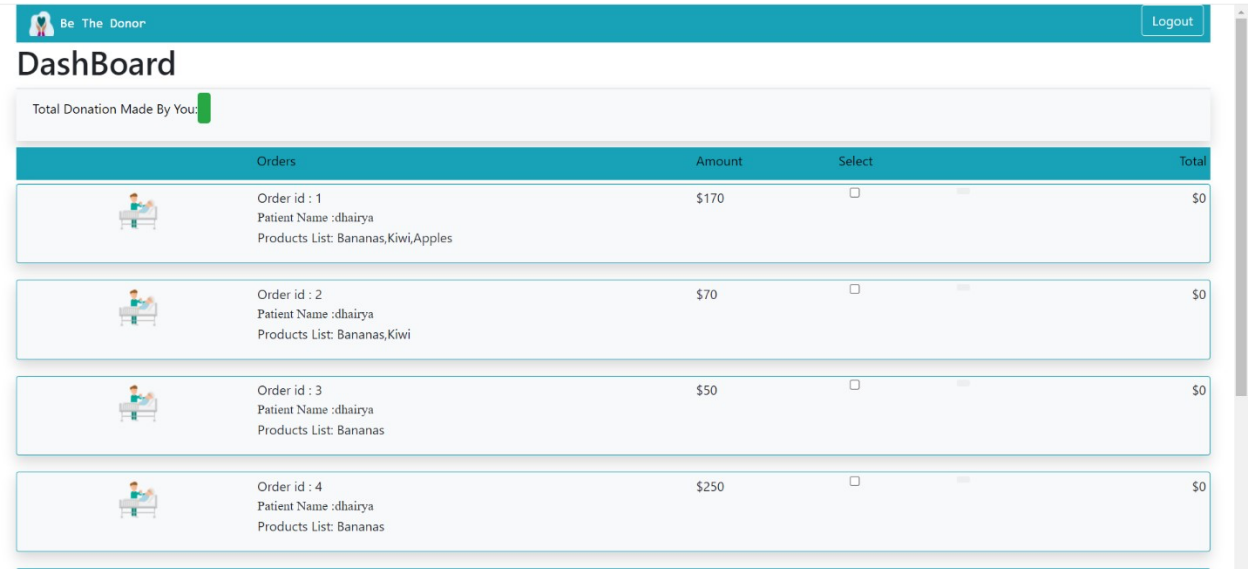


Figure 18

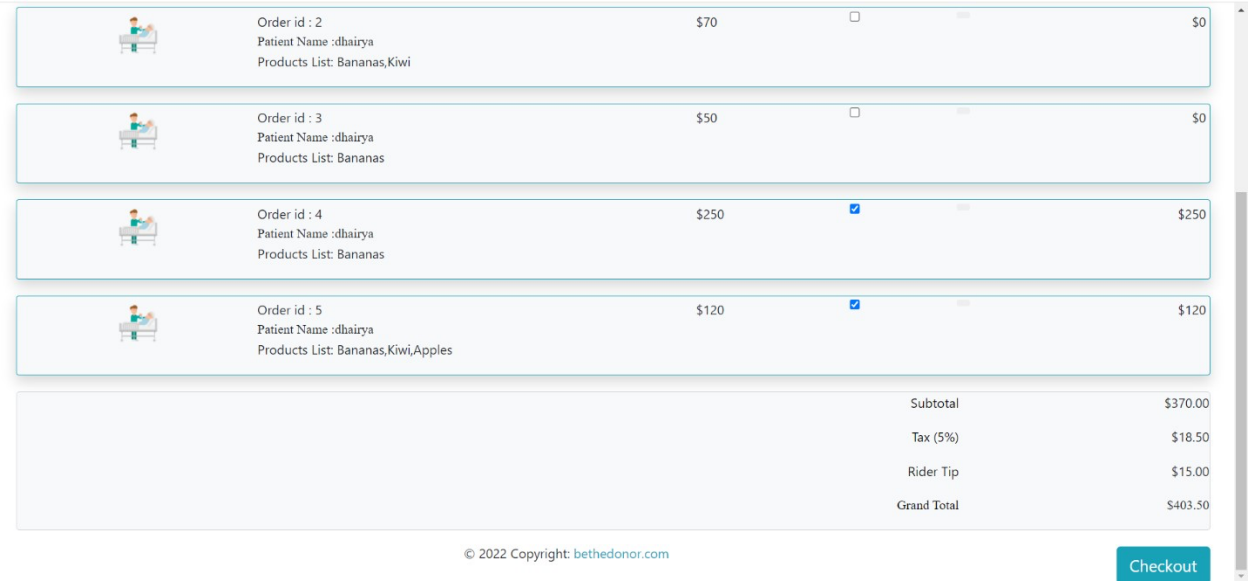


Figure 19

Payment

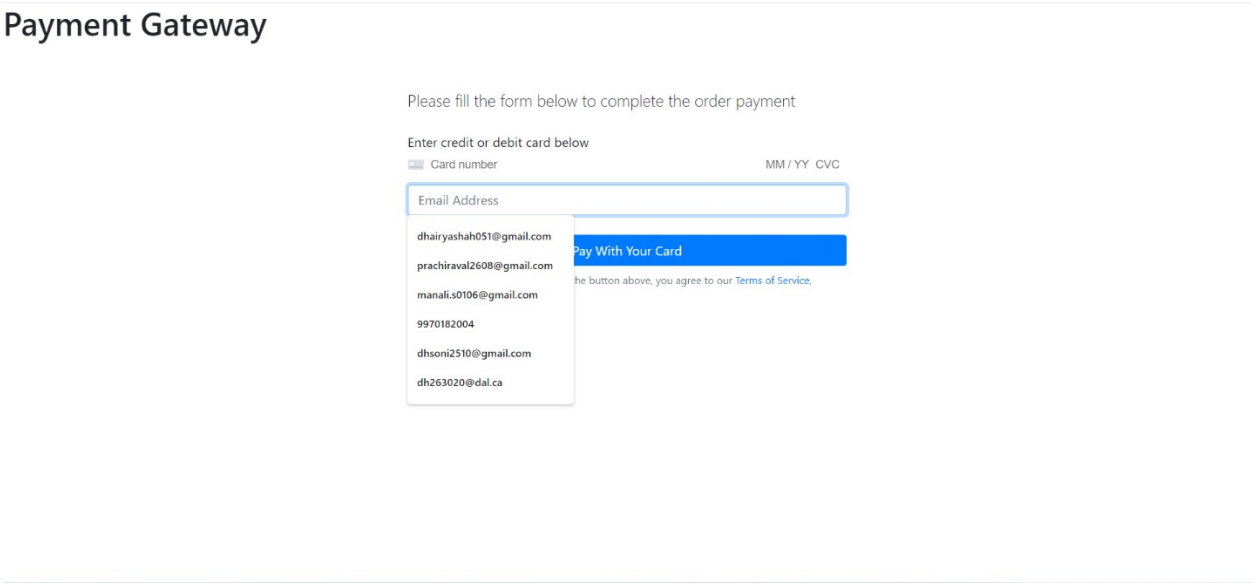


Figure 20

1. Donor will click on pay and will be redirected to the payment gateway. After that the order status will change to delivery pending for the rider to view.

2. We have implemented stripe payment API to introduce the payment simulation.

3. User will be able to add the payment details and actual payment will be made.

4. Security is maintained by the Stripe.

Rider

1. After logging in rider can search the city and find the orders based on the city.
2. Here we have implemented Autocompletion Search so that this functionality becomes more user-friendly.
3. As shown in figure [21] if rider could not find any orders corresponding to the given city pop-up message will be shown showing no orders available for the given city name.
4. If rider successfully found the orders for the given city name all the orders whose order_status is “Pending Delivery” will be shown in the page as shown in figure [22].

5. Rider has an ability to select the order he/she wants to deliver, and he /she can select multiple orders by clicking on the checkbox figure [22].
6. As a rider If I click on the add orders button [23], I will be able to go to the next page which is responsible to show the selected orders where rider can remove that selected order by clicking on remove or deliver that order by clicking on deliver button as shown in figure [24].
7. After clicking deliver button the order_status will be changed to delivered and date of delivery will be changed.

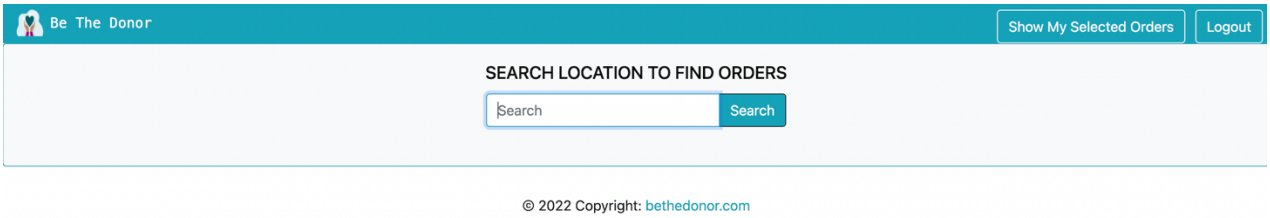


Figure 20

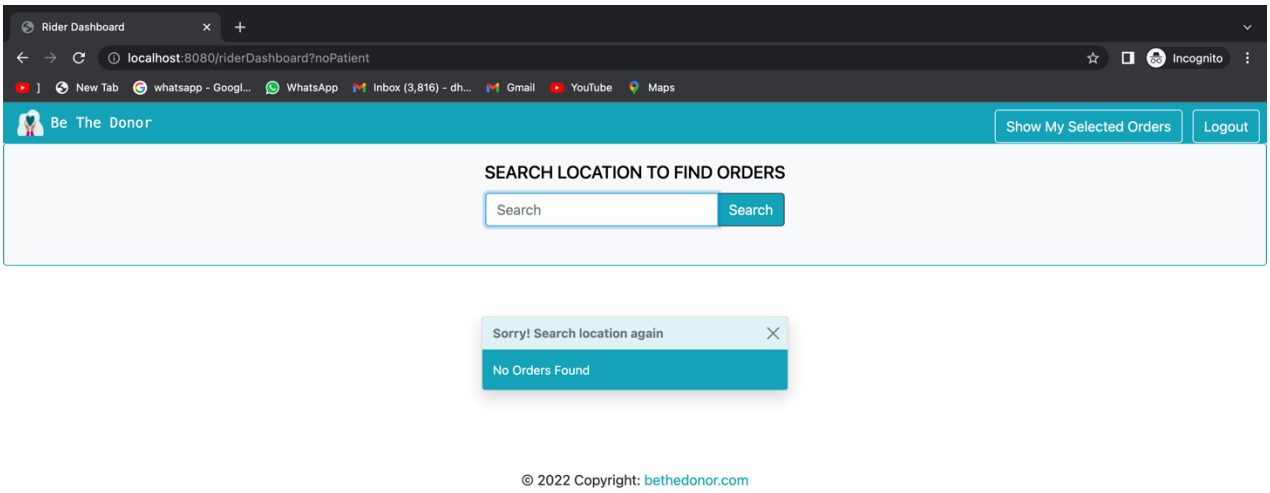


Figure 21

1

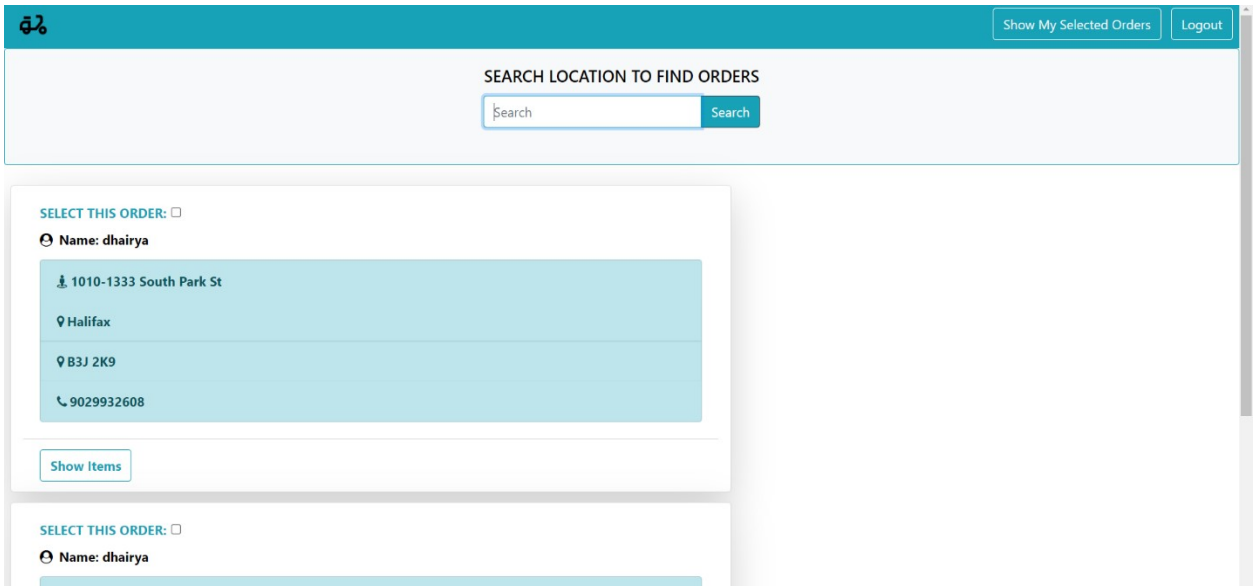


Figure 22

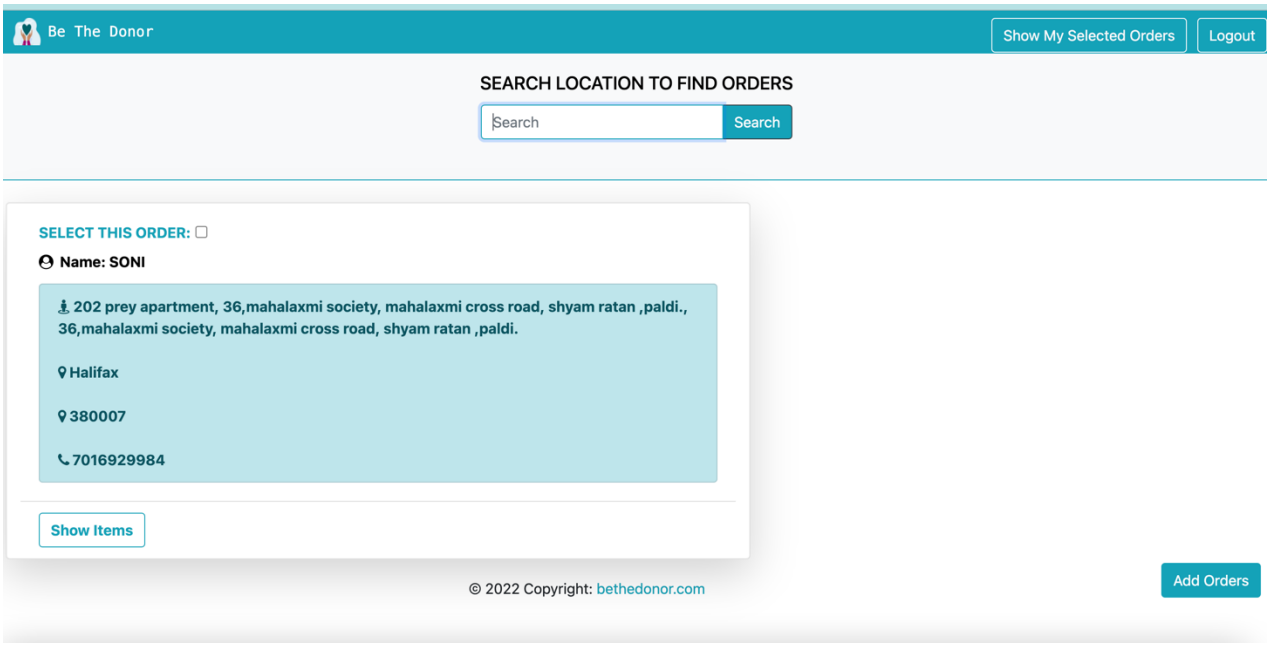


Figure 23

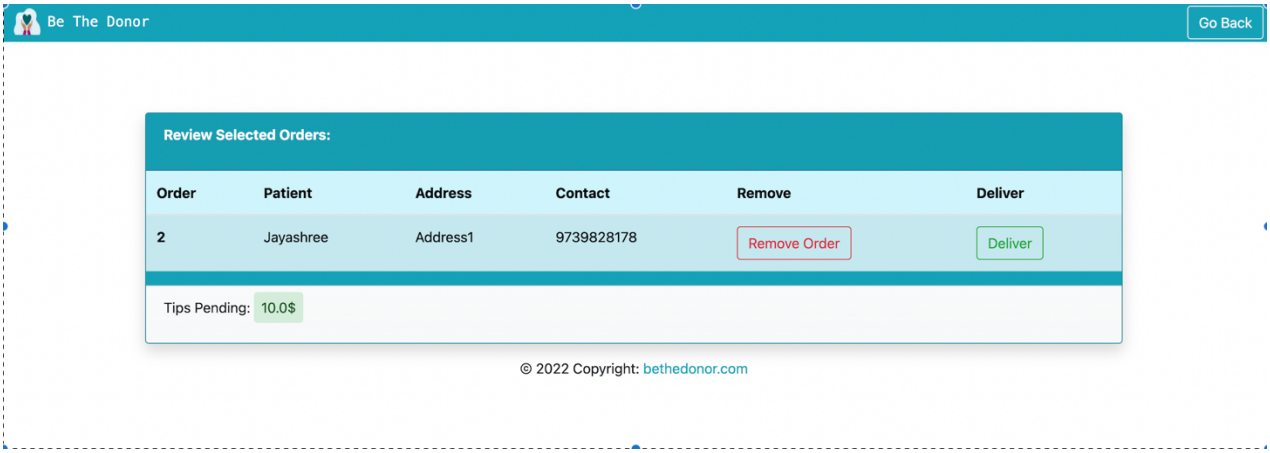


Figure 24