PROJECT REPORT

CART SHARE

Term project- CMPE 275
Group 10

Instructor
Prof. Charles Zhang

Project Members
Nithya Kuchadi (013769665)
Kowshhal Uppu (014538134)
Vamsi Mundra (014608464)
Yash Trivedi (014509339)

Motivation and Introduction

CartShare is a web application that allows neighbors living close by to take turns in shopping and delivering groceries. The prime motive behind this application is providing customers more flexible options for buying their groceries. The proposed idea saves time to users which they would spend on driving to grocery stores every week. It also saves money on gasoline charges. It aims to reduce pollution by allowing users to drive less. It helps customers by reducing co2 consumption.

High level and component level design

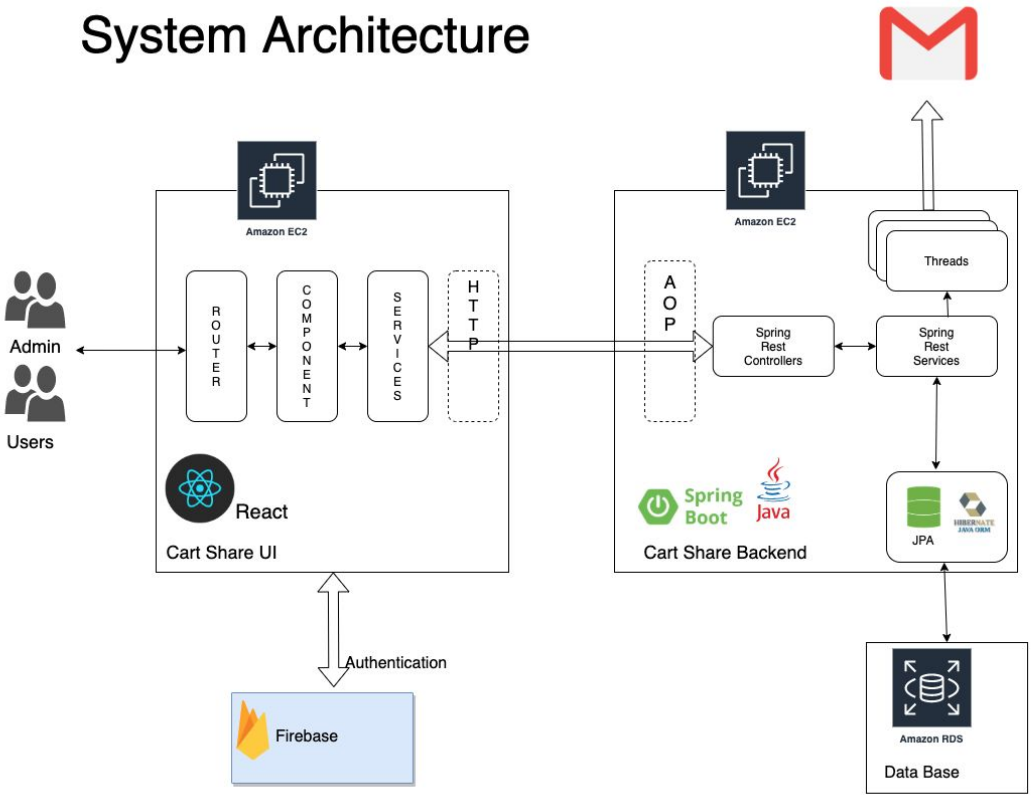


Figure 1. shows the architecture of our application. We have followed MVC architecture. We have used React.js to develop the user interface of the application and Spring Boot to develop the backend REST APIs of the application.

Firebase is used for authentication. Used AOP to validate the input data. Amazon RDS database is used to store the application data. In the data access layer, we have used JPA and hibernate to store and retrieve the data. Images are stored in Amazon S3. Implemented multi-threading with JavaMailSender to send emails through the web application. Dockerized and deployed the application in an Amazon EC2 container.

React Level design:

Separate components are developed for different views of the application. Depending on the url path, React router routes the requests to specific components. We have used axios to make backend REST API calls to save data and retrieve the data.

Rest API design:

We have used Spring boot to develop the REST APIs. All the incoming requests are validated with AOP before the requests are handled by controllers. Rest Controllers are used to handle the incoming requests and requests are mapped to the appropriate service handler method. All the business logic is written in the Rest Service layer. JPA and Hibernate are used as ORM. Response will be returned from controller to frontend service.

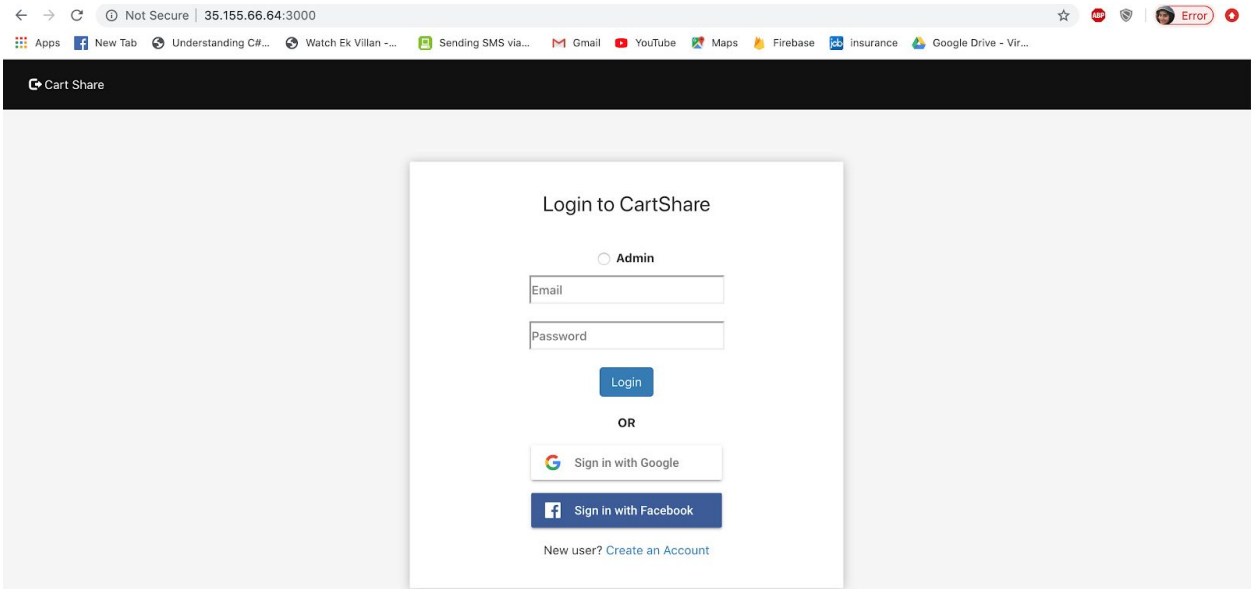
Technology choices

1) Spring Boot for backend APIs, React.js for frontend and MySQL RDS as a database.
2) Firebase for authentication, and JPA with Hibernate implementation as ORM.
3) Implemented Multi-threading to send emails with JavaMailSender,
4) Used Spring Autowiring for Dependency injection, and Rest controller and services for APIs.
5) Used transactions for all DB operations and AOP for input validation. Used Amazon S3 to store all images.
6) Used Docker and Amazon EC2 container to deploy the application.

Description of features with final screenshots

User Login:

Users can login to the application by using car share credentials or sign in with Google or Facebook.

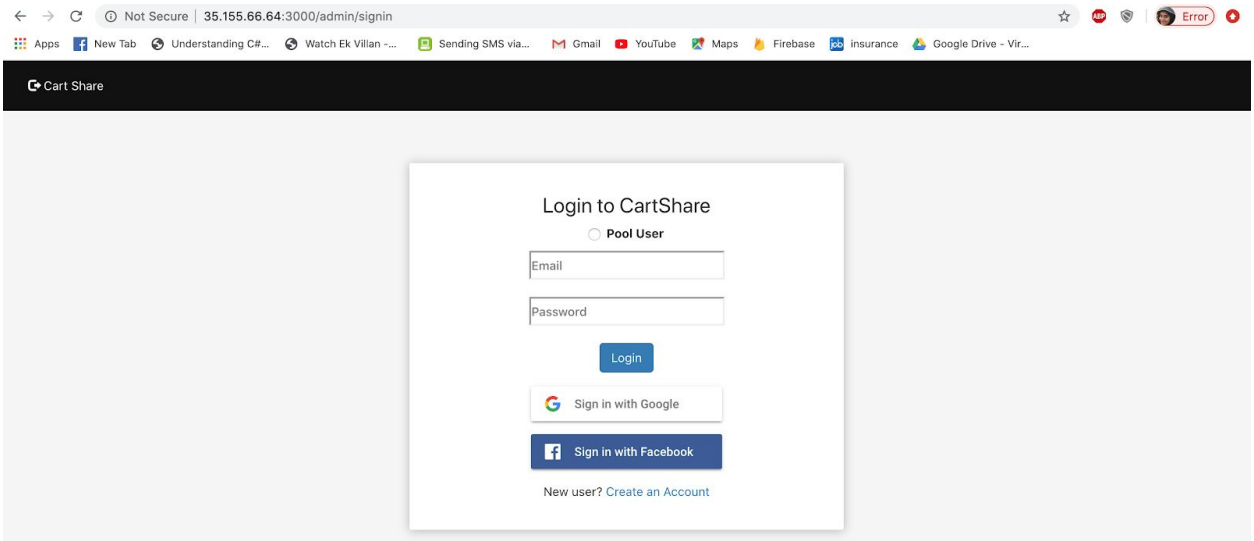


Admins can login to the application by clicking on radio button Admin. It redirects to the admin login page.

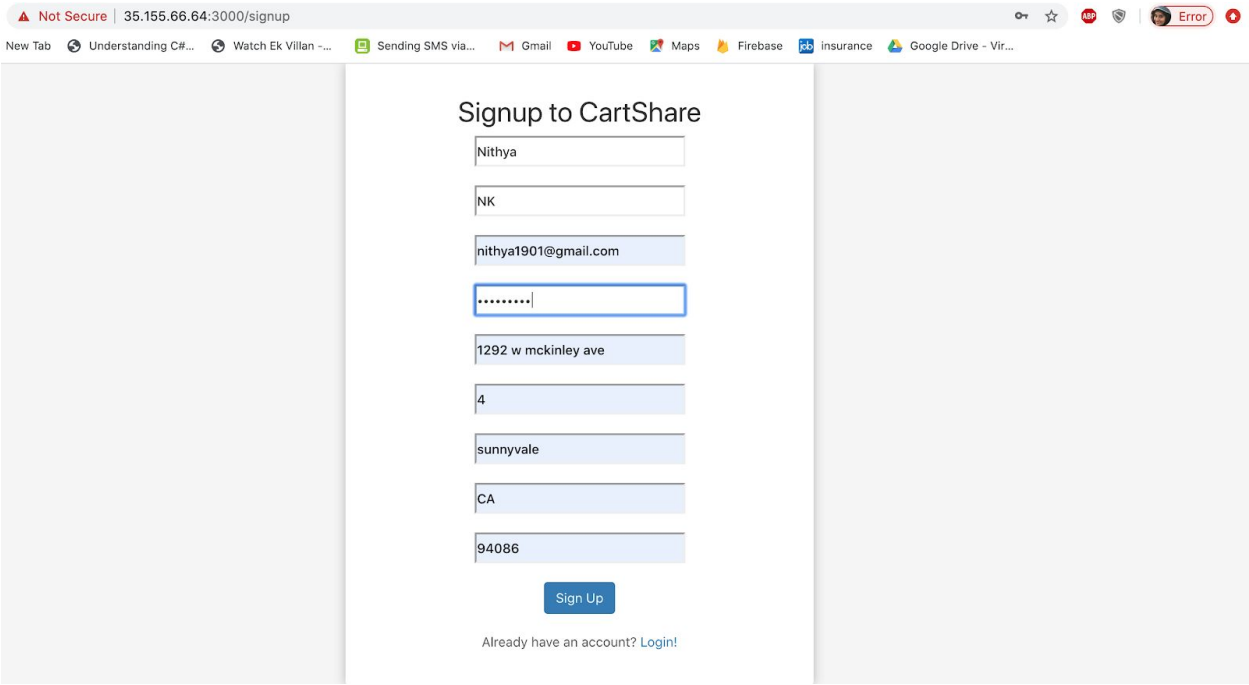
Admin Login:

Admins can login to the application by using car share credentials or sign in with Google or Facebook. Admin can use only sjsu email id to login.

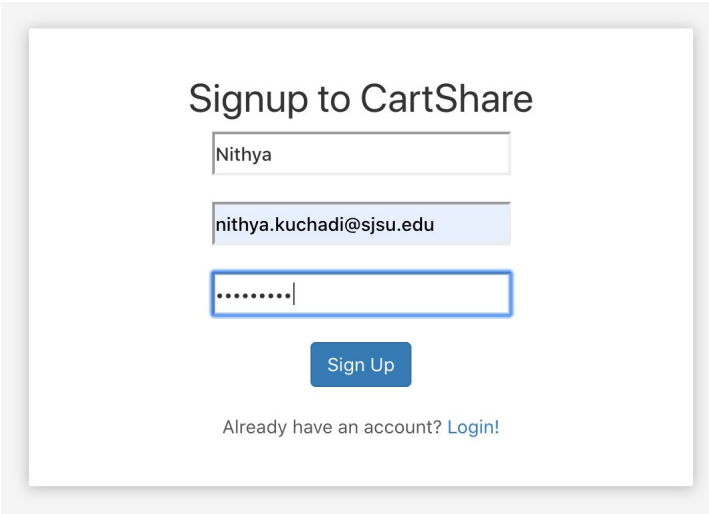
To sign in as a pooler, admin can select the Pool user radio button.



Pool user signup:

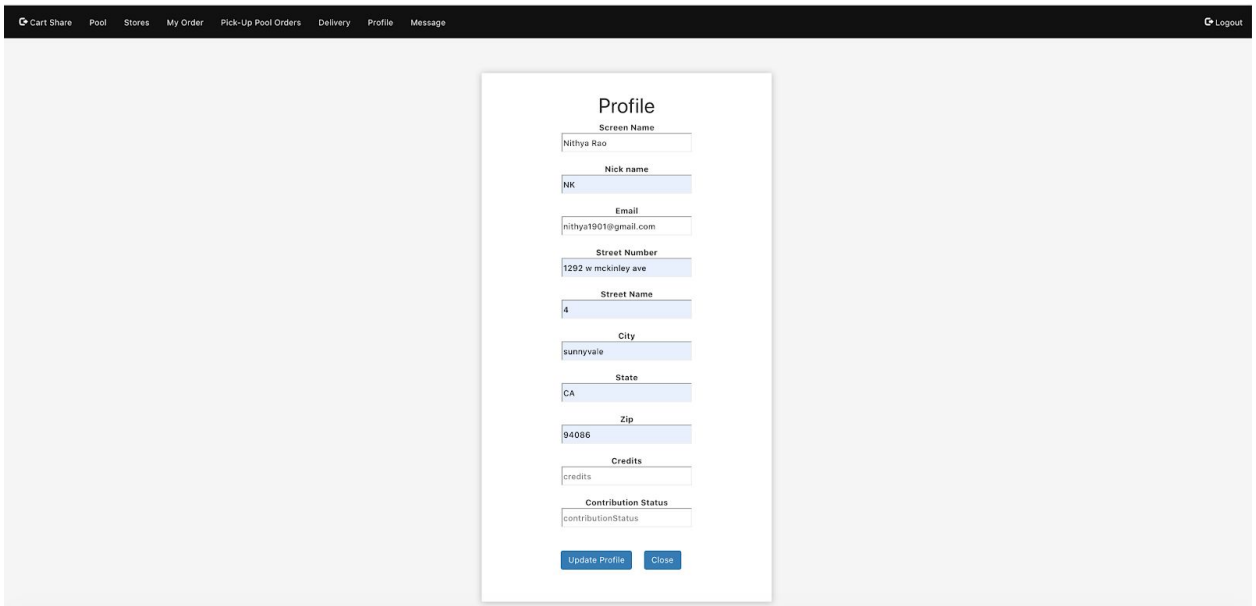


Admin Signup:



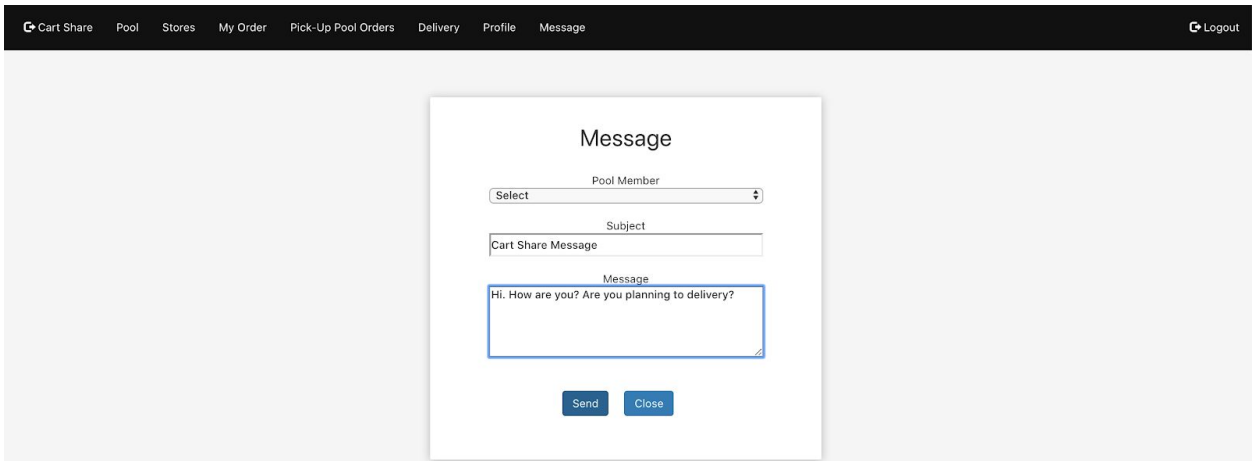
User profile:

Users can view the profile details such as Credits, Contribution Status, Email, nickname, screen name and address details. Users can update Nick name and address details from this view.



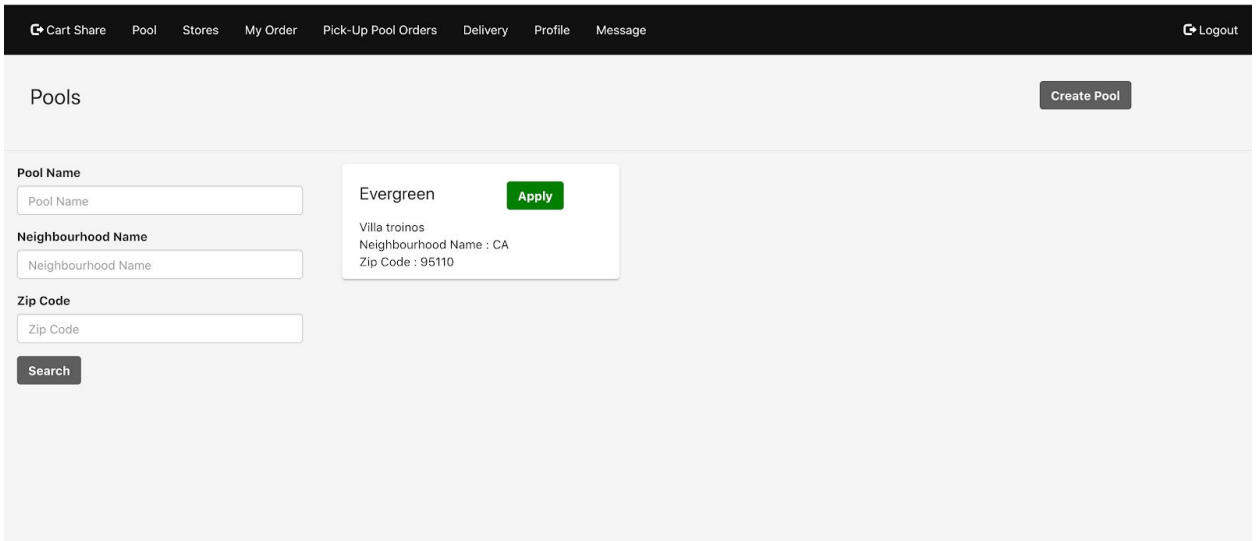
Message:

Users can message any other pool users through this view.



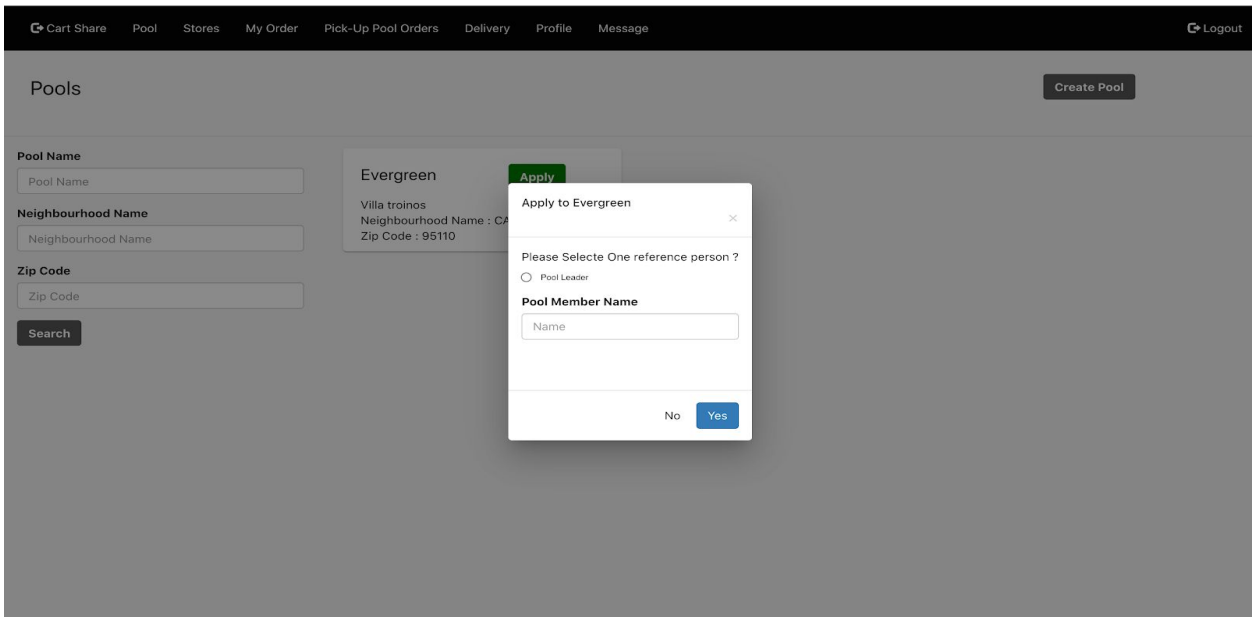
User Pool Landing Page:

Users can view all the pools. He can filter the pools by Pool Name , Neighbourhood Name and Zip Code. User can create pool as well.



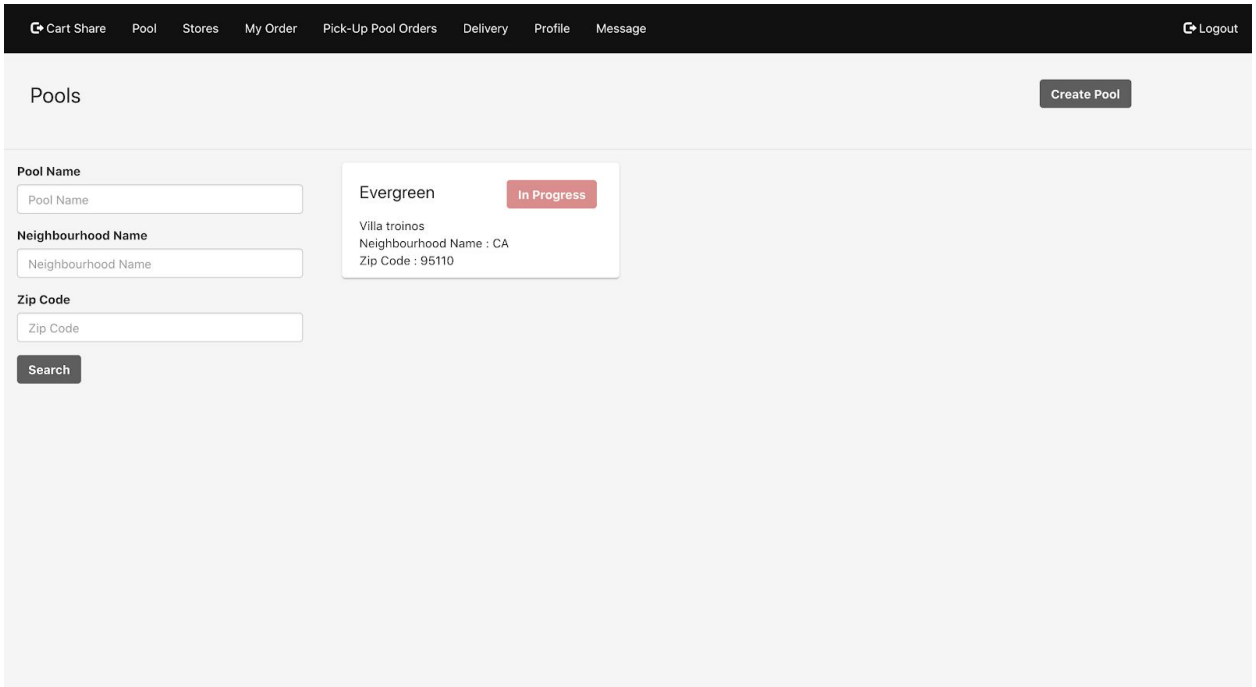
Apply Pool :

Users can apply to the pool by pool leader reference or pool member reference.



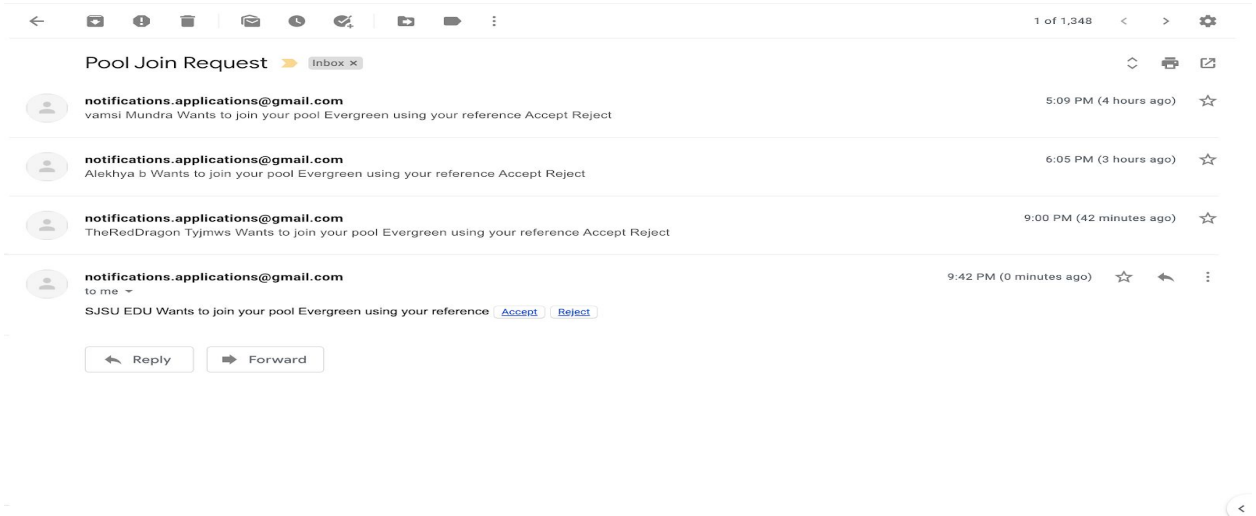
Apply Pool :

After applying, the pool user can check the status.



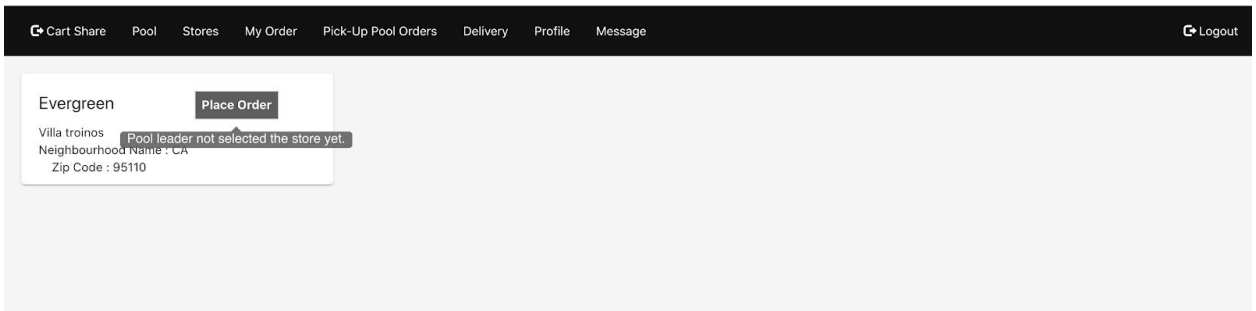
Pool Request:

Pool Reference person will get a Pool request mail. If he is not the pool leader.After his acceptance Pool Leader will get the final mail. After Pool leader approval, User will be joined to the pool.



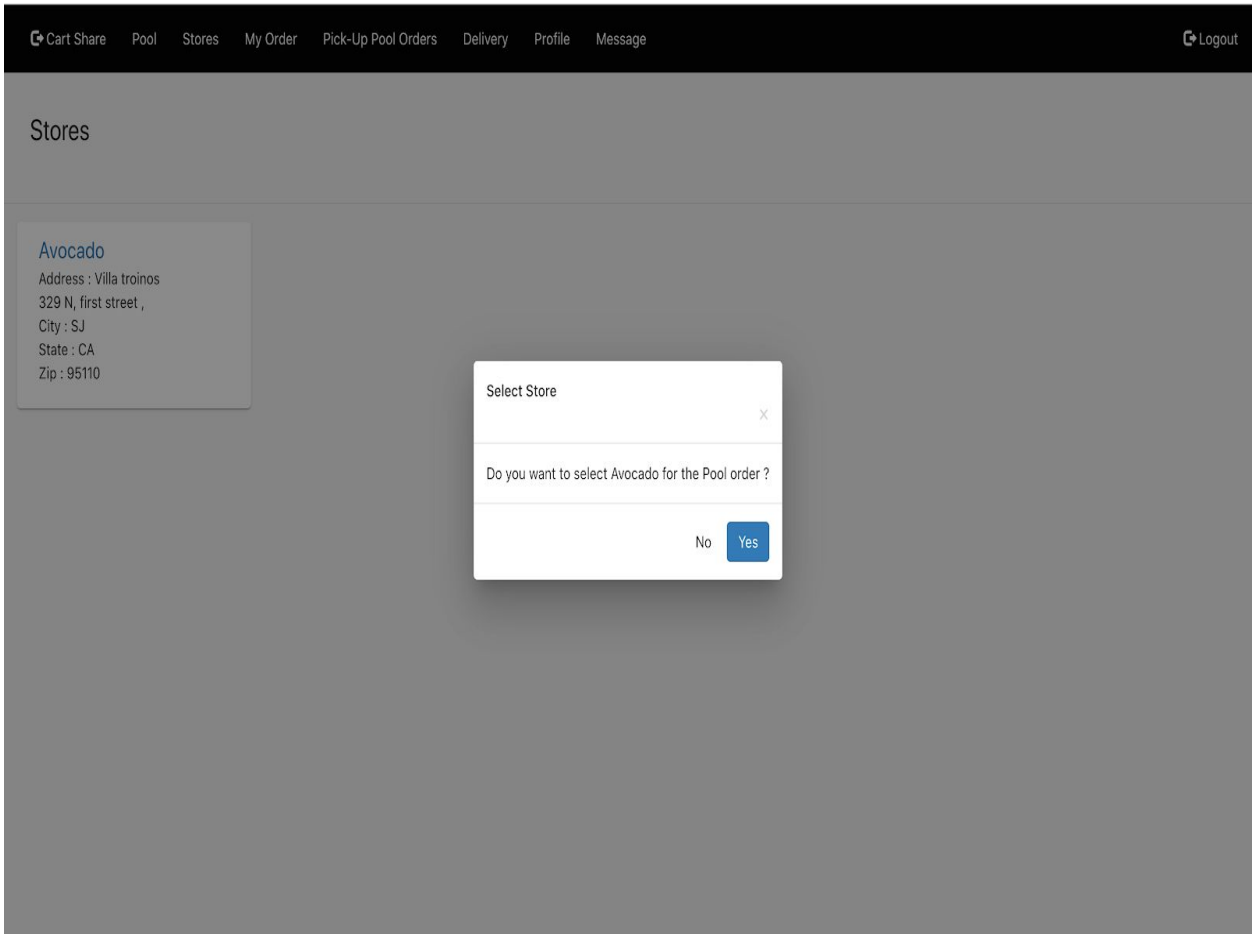
Pool User :

After joining the pool if the store is not selected Pool leader has to select the store.



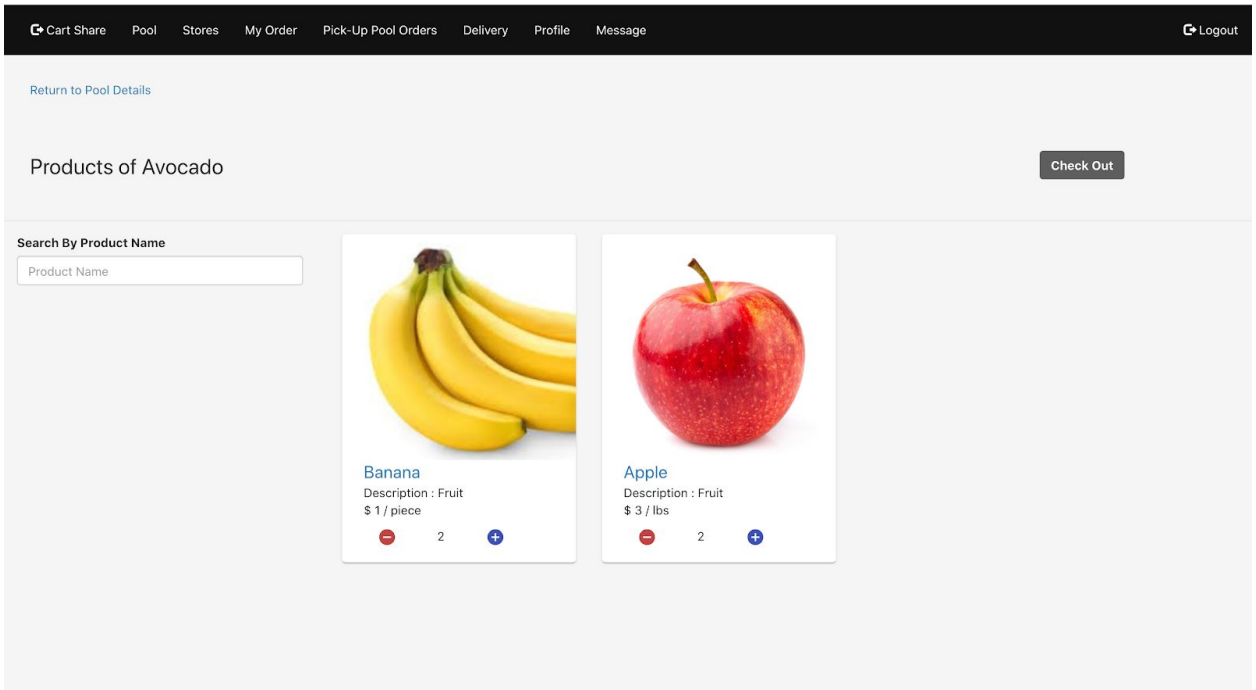
Pool User :

Pool leader has to select the store.



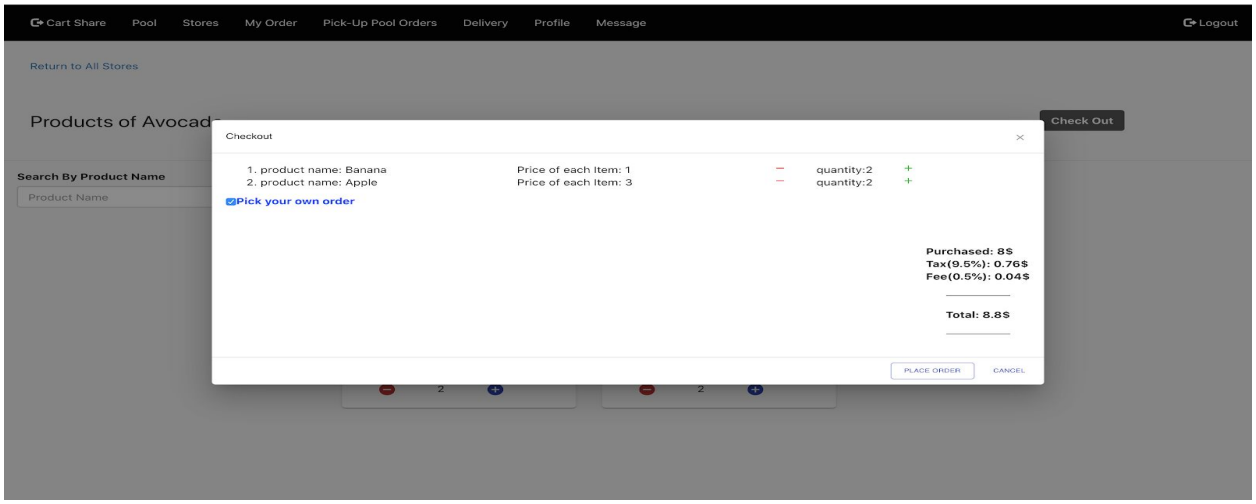
Pool Store:

After selecting a store for the pool. Pool User can select the products and check out.



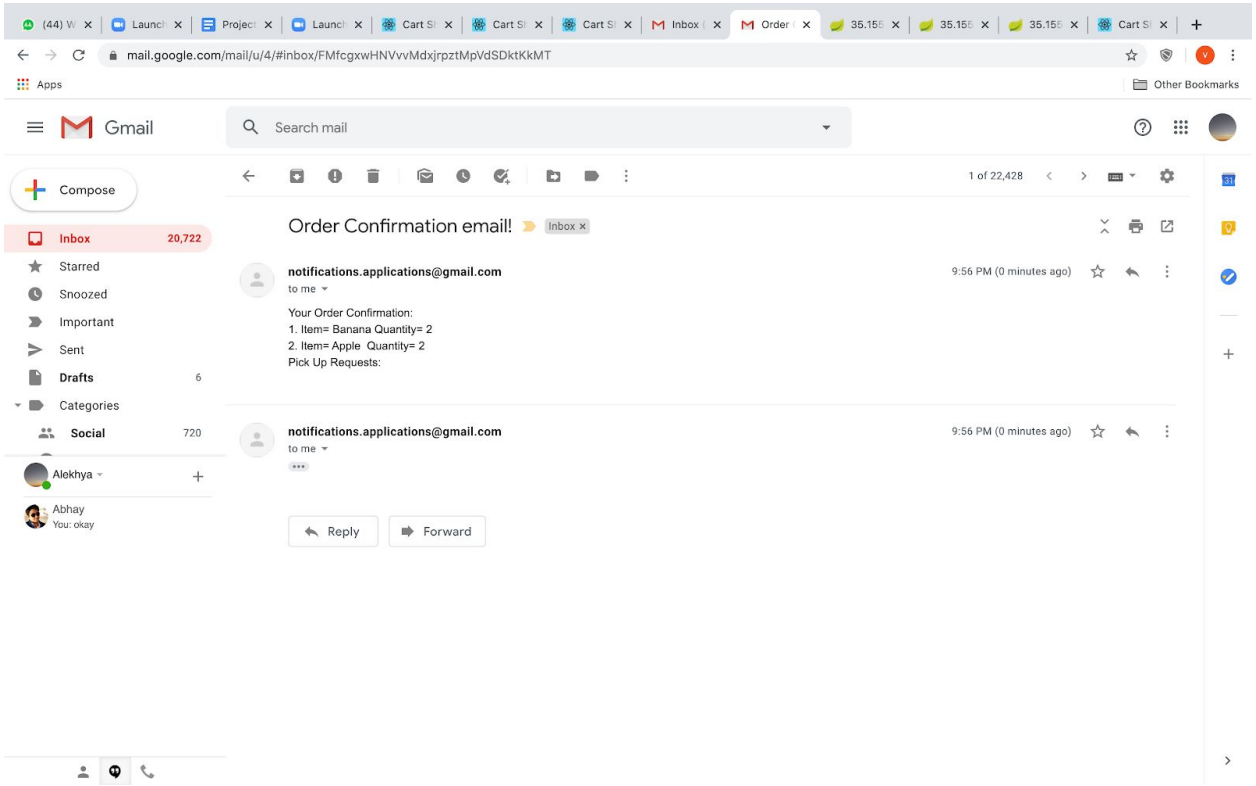
Pool Store:

At the time of checkout Pool user can select whether he picks the order or pool users to pick.



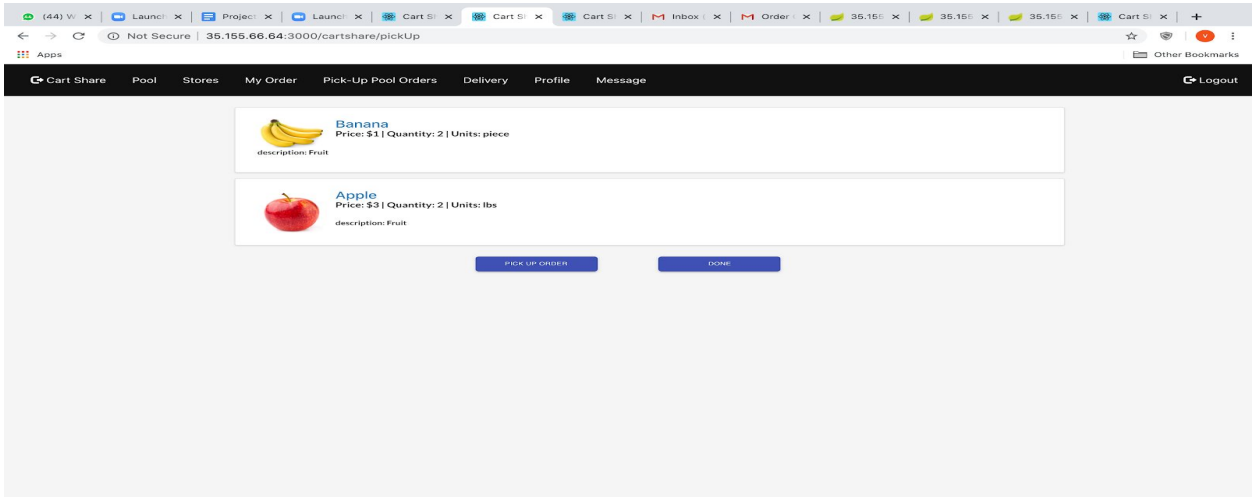
Pool Order:

Pool user receives the order details in Email.



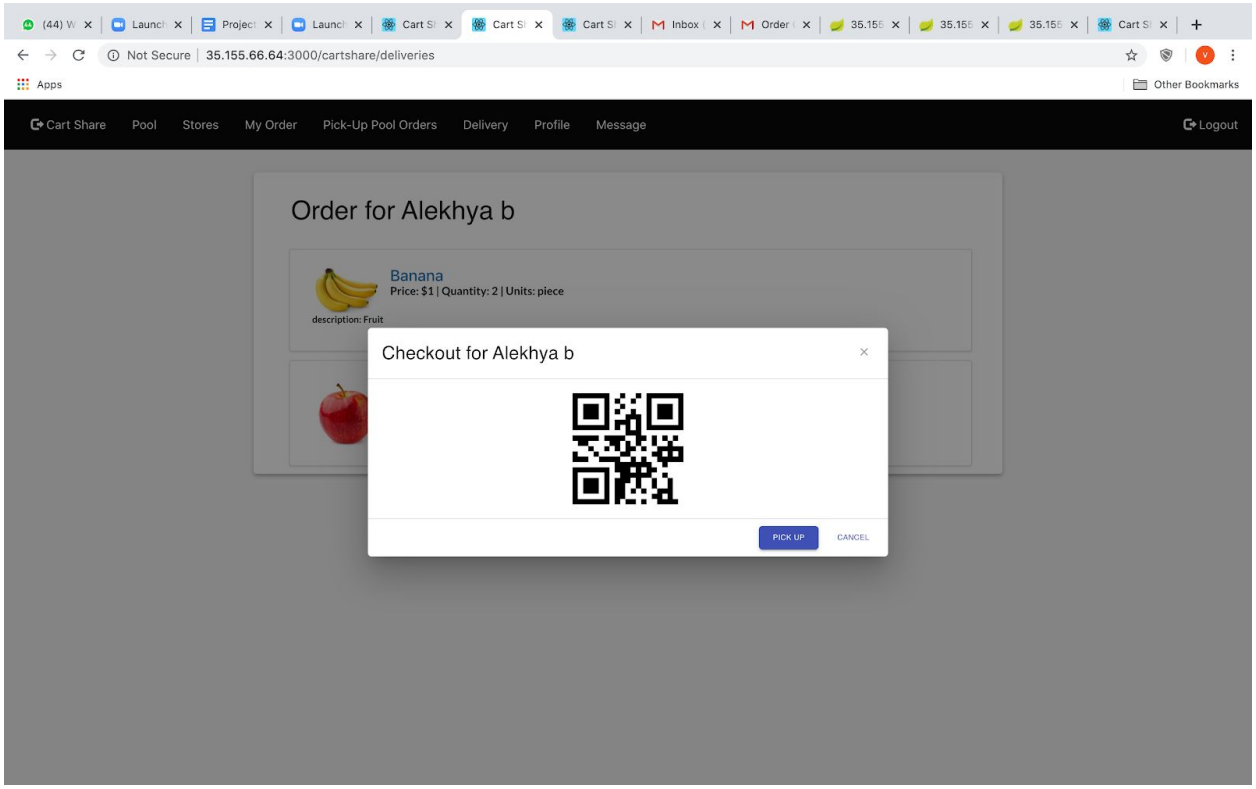
Pick Up Orders:

Pool user can see the available pool orders for Pick UP. He can pick orders in order.



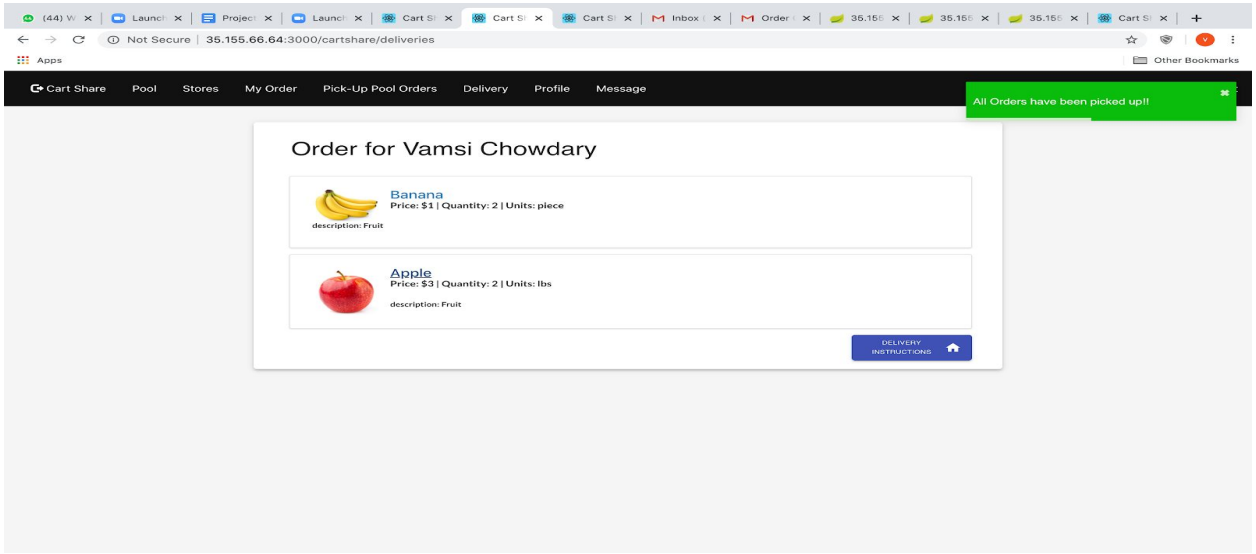
Checkout Orders:

Pool user can pick up and check out all orders from delivery..



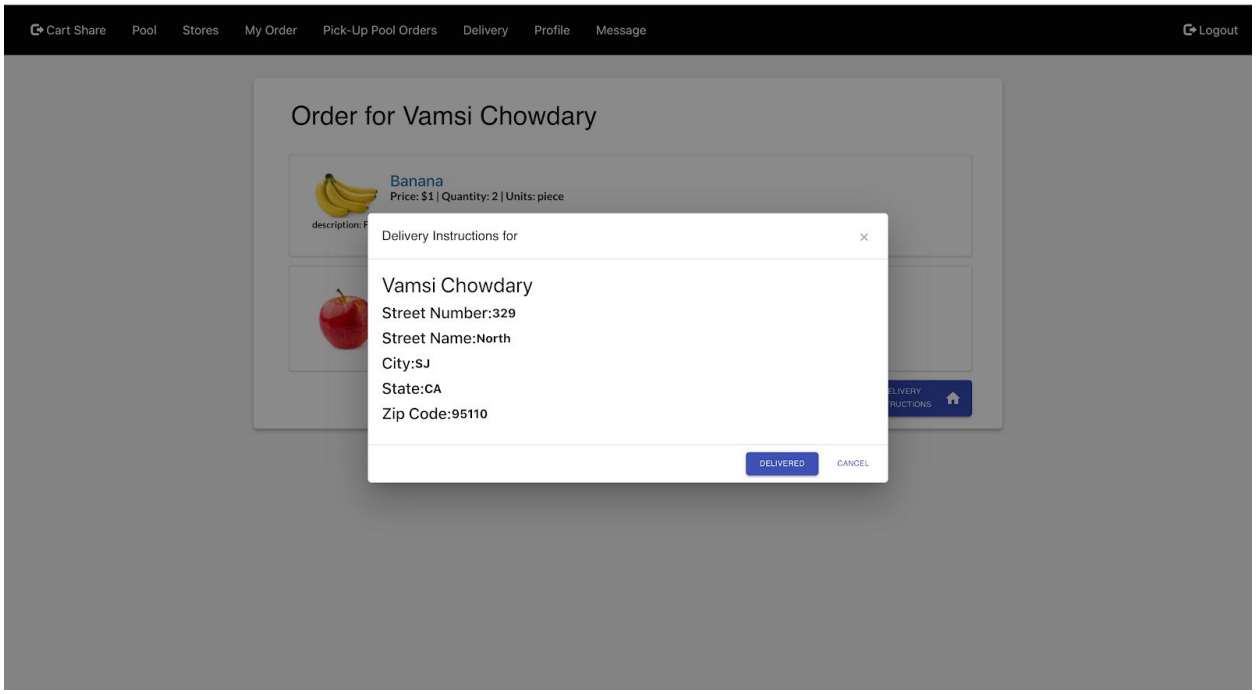
Picked Up Orders:

Pool user after pick up orders can check the delivery instructions.



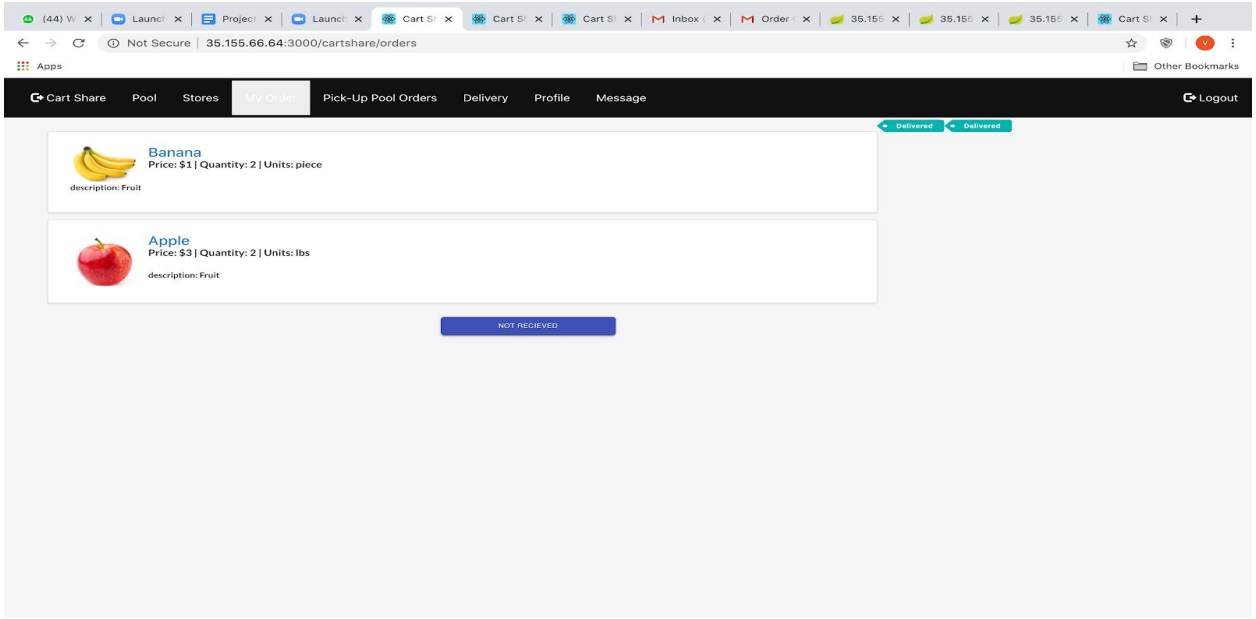
Picked Up Orders:

Pool user will get delivery instructions and he can deliver the order.



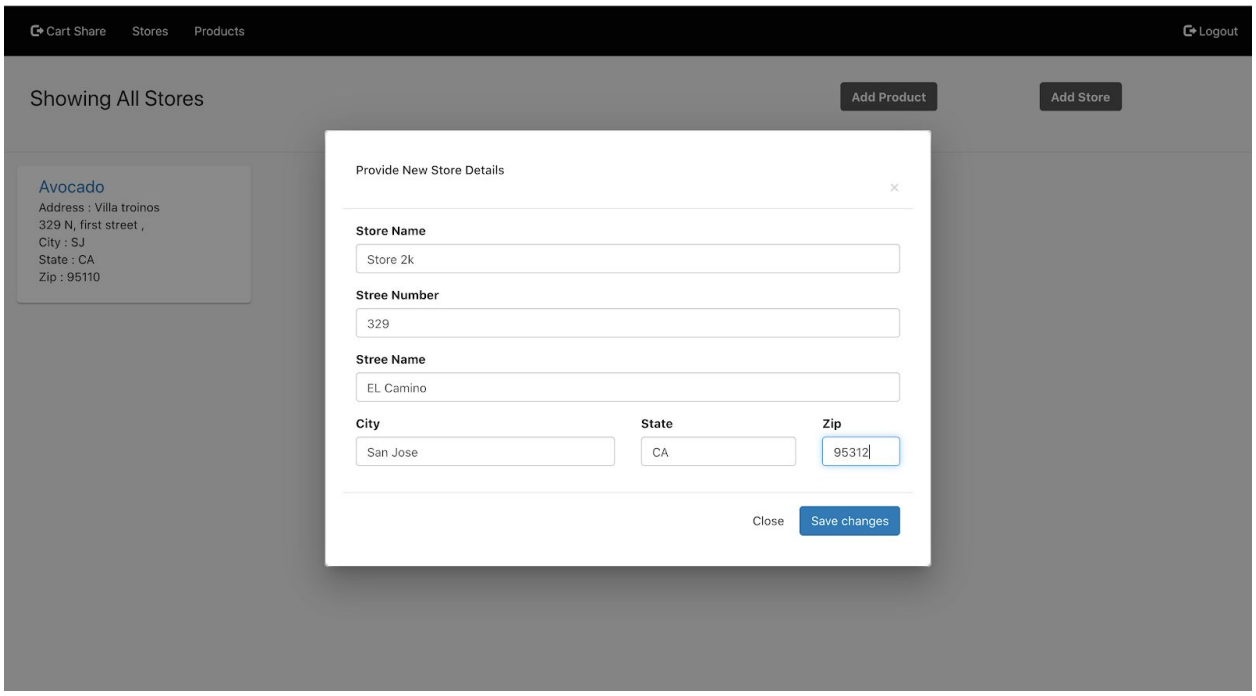
Picked Up Orders:

Pool user can check his order status.He can report if the order is not delivered yet



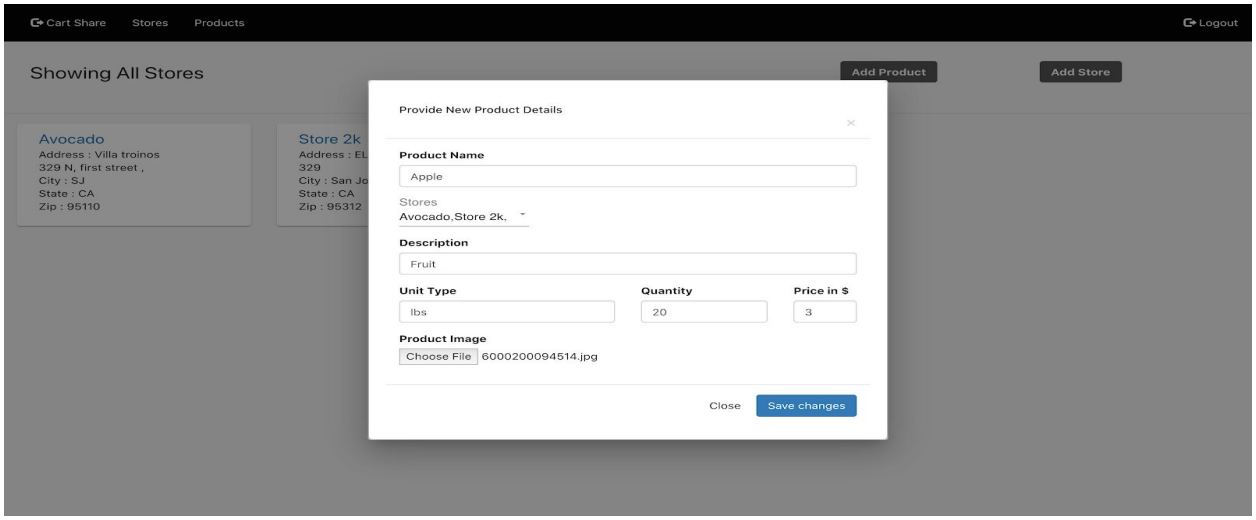
Admin :

Admin can create the store .



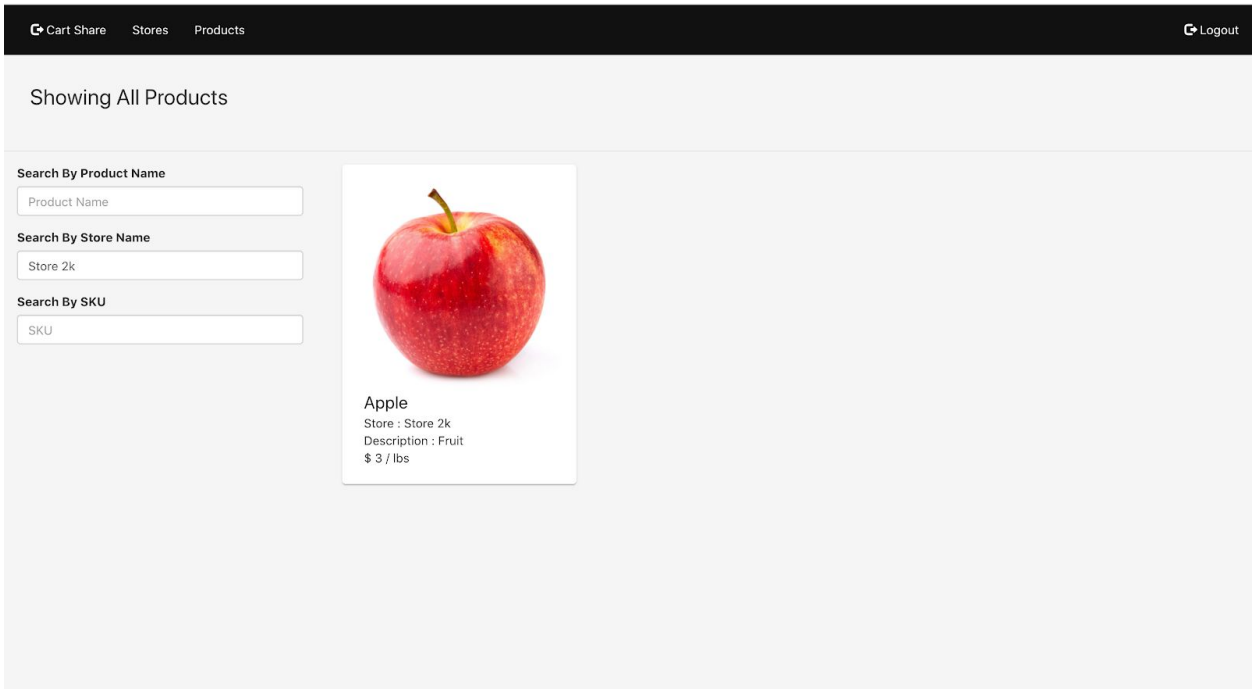
Admin :

Admin can add product to multiple stores.



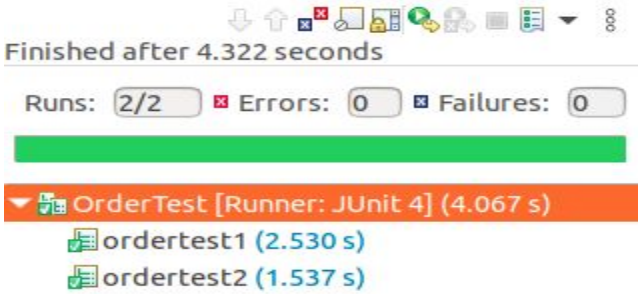
Admin Products :

Admin can see all the products. He can filter the products by Product Name , Store Name and SKU.



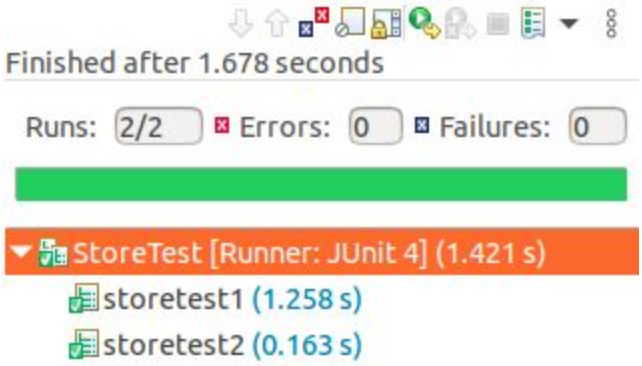
Testing plan executed and results

We have done junit testing to test the backend REST APIs**.**
Testing is performed using JUnit testing. Junit is performed for unit testing. In Unit Testing individual components are tested and validated. The test is performed for multiple types of cases like successful and unsuccessful. Below is the sample results of JUnit testing on Orders and Store features.
OrderTest: Testing the order features using JUnit.




```java
@RunWith(SpringRunner.class)
public class OrderTest {

    public void setup(){


    }

    @Test
    public void ordertest1() throws Exception {
        RestTemplate restTemplate = new RestTemplate();
        final String baseUrl = "http://35.155.66.64:" + "8080" + "/order/user/4";
        URI uri = new URI(baseUrl);
        ResponseEntity<String> result = restTemplate.getForEntity(uri, String.class);

        Assert.assertEquals(200, result.getStatusCodeValue());

    }
```

StoreTest: Testing Store feature using JUnit.




```java
@RunWith(SpringRunner.class)
public class StoreTest {
    public void setup(){

    @Test
    public void storetest1() throws Exception {
        RestTemplate restTemplate = new RestTemplate();
        final String baseUrl = "http://35.155.66.64:" + "8080" + "/admin/store/allStores/1";
        URI uri = new URI(baseUrl);
        ResponseEntity<String> result = restTemplate.getForEntity(uri, String.class);

        Assert.assertEquals(200, result.getStatusCodeValue());
        Assert.assertEquals('1', result.getBody().charAt(7));
    }
```

Lessons learned

1) Schema design is very important in developing the application. Initially, we have designed the schema by reading high level requirements. But later, while implementing, we have to change a few components in the schema. Because of this, we had to modify the implemented part as well and wasted a lot of time.

2) Learned practical implementation of frameworks such as Spring and Hibernate.
3) Learned how to use firebase for Oauth authentication.

Possible future work

1) Feature to select and display time when the deliverer is planning to pick up and deliver the groceries should be added.
2) Using the shortest distance algorithm, stores available near to the neighborhood should be displayed in the application with a map feature.