DS Assignment 2 – Train Ticket Online Booking System

Module: **Distributed Systems**

Assignment: **2 - Rest API**

Report

Software Engineering Weekday Batch

IT 16 1787 00	D.I.K.Rajapakshe

Table of Contents

1. INTRODUCTION…….3

2. ARCHITECTURE……4

3 BACKEND……6

3.1 Workflow……6

3.2 Use of Model Class...…….7

3.3 Athentication and Security..…….8

3.4 Use of MongoDB with Spring Boot..……9

4. WORKFLOW (FRONTEND)…...…….10

5. APPENDIX……...11

5.1 Backend…..……..15

5.1.1 Controlers…...……..15

5.1.2 Modles……...……..23

5.1.3 Repository..……...……..30

5.1.4 Service……...……..32

5.1.4.1 Service Interface..……....……..36

5.1.4.2 Service Implementation…….47

5.2 Frontend…...……...……..47

5.2.1 JavaScript Page……....……..47

5.2.2 JavaScript Functions……...……..62

1. Introduction

This report is for the 2nd assignment of Distributed System module which is about implementing a scenario similar to an online train ticket booking System establishment which lets customers book multiple tickets and pay for it by using a credit/debit card or add the amount to the Dialog bill. The application is a combination of a website(frontend) using ReactJs and a REST API (backend) using Spring Boot which is accompanied by a MongoDB database.

1. Website

- This is a collection basic ReactJs + JavaScript + CSS pages, Axios library for sending HTTP requests to the REST service, and jQuery library for dynamically changing the content of the web page.

2. REST API

- This is implemented based on SOA principles by using Spring Boot framework with Java and runs on its own embedded server instance and exposes a number of services over HTTP which Spring Tool Suite Software.

Following is the sequence steps of fetching tickets by the frontend (using axios).

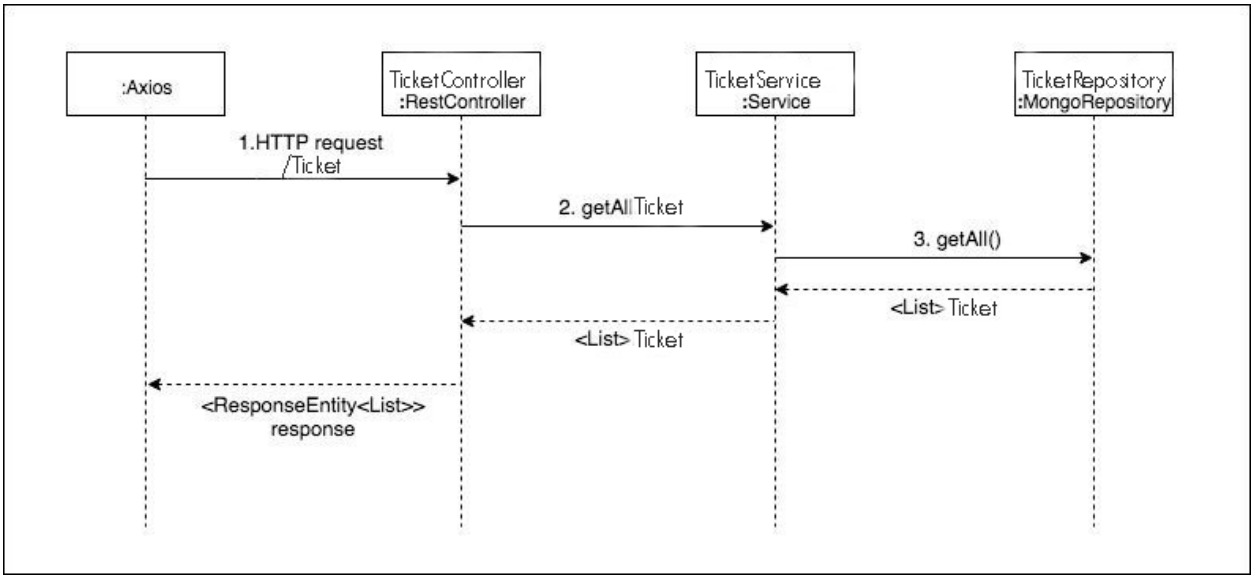

Fig 1: Fetch food items by axios

2. Architecture

The backend is designed with SOA principles in mind using MVC Architecture. All of the functionality is provided as a set of an API requests/responses using paths. This allows us to modify the internal mechanisms routing without changing the frontend too much. All of the functions are divided into three categories:

1. Ticket

2. Payment

3. User

4. Session

Due to this sort of grouping, we can let each of them run independently without the existence of the others while greatly improves scalability and functional cohesiveness routing while decreasing coupling nature. But all of these services use a common NoSQL database using MongoDB.

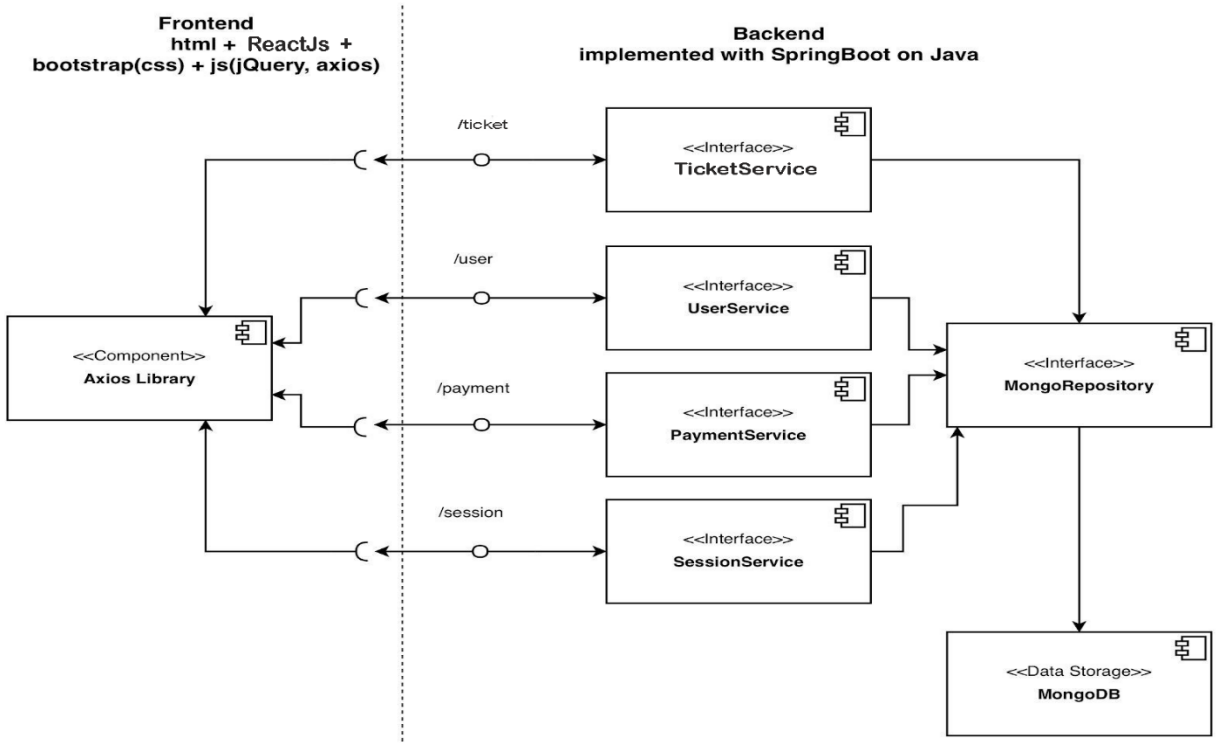

Fig 2: High Level Design diagram of the overall system.

For the designing the backend, SOA was adopted by implementing the major functions of the system as web services that are exposed as an API for consumers to use for the system. Since this involves data being passed via network over HTTP protocol, an authentication key is used for the major part of API calls. For communication between clients and the backend, JSON and axios is used as the only method of communication since the use of MongoDB is well complemented by for the web system. The system consists of four major services mentioned above and each of those services provide another set of sub functionality and services; and the client basically consumes these sub services given that they are authenticated via the system. For implementing the services, each service was looked at in four different perspective packages.

1. Controller
2. Model
3. Service
4. Implementation of service
5. Repository

Each service has its own controller which handles the user requests mapped to its URL and these controllers make their respective services independent of each other on system. The model is what everything revolves around. Each model represents an entity that is involved in the online booking system such as Ticket, User, Payment and Session. Since each service offer a range of sub functions, a service interface is used to expose each of these sub functions to connect. These mainly involve manipulation of data to cater for different types of system requests that the services support.

Since the system is divided into a backend and a frontend where the user is connected with the front end, JSON was chosen as the sole communication medium since its light in weight, easy to manipulate in JavaScript which is the only language that the browsers understand easily. Talking of frontend, it is implemented as a basic website which relies on API calls for all the processing since no data is processed at the frontend own.

3.1 Workflow (Backend)

The coordinator of all backend tasks is handling by the Controller class of each service in the system. Booking Ticket Service as an example, any and all requests that are mapped for /ticket will be handled by TicketController class. Depending on the full URL that follows /ticket, the controller will call the appropriate method calling. For an example, for a GET request of URL /ticket, the controller will call the method that is responsible for returning the list of ticket items available on the database. This is where other interfaces and classes come into executing. The method in the TicketController will call a method implemented in TicketServiceImpl (implementation class). Inside that implementation, the TicketRepository is used to deal with the MongoDB instance path. Here, the TicketRepository is the only interface which we don't implement but define method we can freely call and execute since this interface extends MongoRepository interface which will implement the defined methods during the runtime. The methods that we define in the TicketRepository are MongoDB queries in the form of Java methods and variables. Once one of those methods has fetched data from the Database (in this case, ticket items), the runtime will then use Ticket model class to map the details of each ticket item to an instance of Ticket class. So, what bridges the gap between MongoDB and Java is the model class Ticket via packages. This workflow is common to all the services offered by the backend on the system.

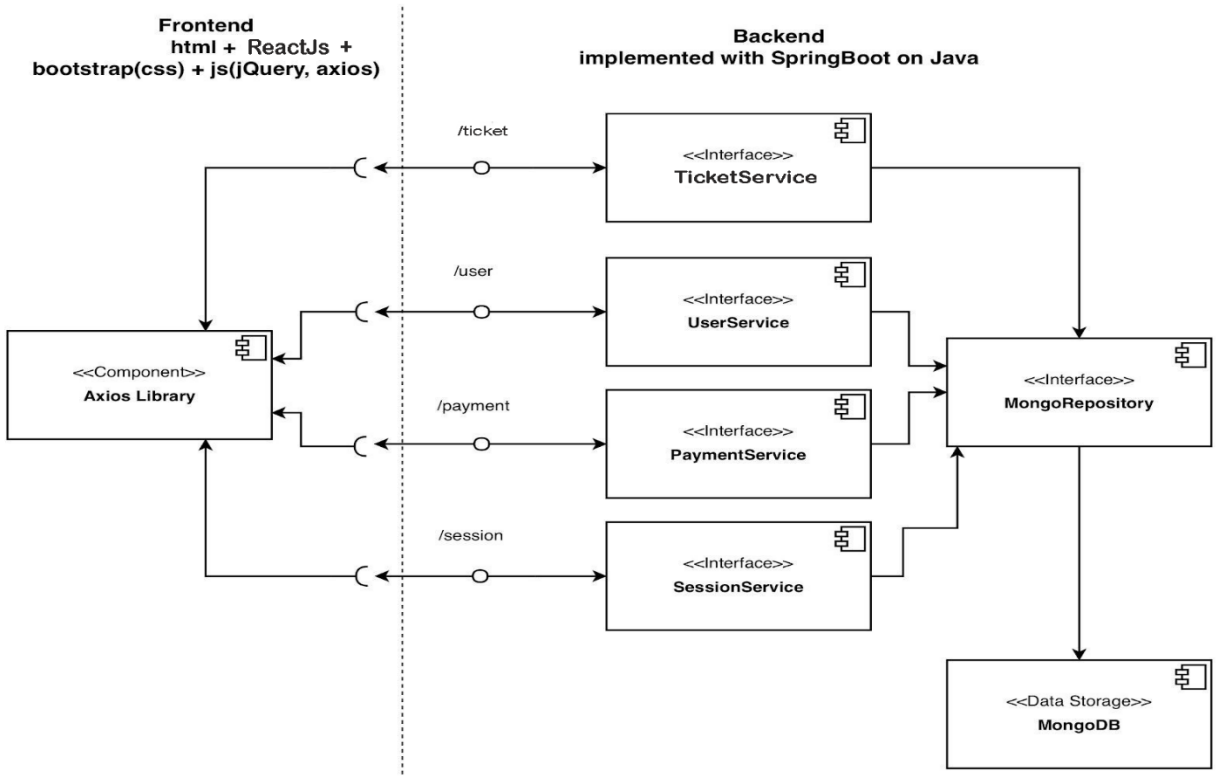

Fig 3: Generic flow of serving a request by the backend

3.2 Use of Model Class

Model classes are basically Java classes which represent an entity, thus its name, Model class which have variables. In this system, there are four model classes.

1. Ticket
2. Payment
3. User
4. Session

The main reason these classes are used is consistency and ease of saving and retrieving entries from MongoDB database and ease of manipulating. Since we are using Spring Boot along with MongoDB, we can let Spring Boot to parse the JSON request from the front end and map the JSON data to an instance of the relevant model class in the system. This makes all the database related tasks much easier and consistent as long as the data types of are interoperable there. But, the flexibility of manipulating data on our own is also there since we can get the JSON data as a Map<String, Object> in Java which we can use as we want in any time. Also, we are retrieving data from the database, Spring Boot can map the data into instances of the relevant model class and provide a Optional List of objects or a single object depending on the data retrieved.

3.3 Authentication and Security

For certain API requests like getting all the ticket items available, no authentication is required but from there onwards, majority of the requests must contain the userID and the Authentication Key in the header which will be validated against the available sessions on particular user. Simply, when a user logs in, he/she will be issued an authentication key and that authentication key is stored in the database in system. This key will be valid as long as the user hasn't logged out and when logging out, the key will be erased from the database permanently.

Therefore, authentication indirectly works as a session manager and each session is given a role out of User and Admin. A user is considered logged in as long as his/her session is present to frontend in the database. Authentication key is generated by hashing the user's email and multiplying it by 12. For hashing, the inbuilt hashCode() method provided by Java is used. When saving user's password, the hashed value of the textual password is saved in the database instead of saving the plain textual password to ensure that the password itself will never be leaked and also even an admin who has access to the database will not be able to retrieve the actual password of the user which encrypted by the system.

3.4 Use of MongoDB with Spring Boot

The notion to use MongoDB(which is a JSON based NoSQL database) came with native support for JSON by JavaScript(given that JSON itself stands for JavaScript Object Notation), we can have a solid level of consistency across the frontend using ReactJS and the backend and since Spring Boot is capable of mapping JSON data to model classes(discussed under Use of Model classes) or to a Map<String, Object> which can be used as we want to use.

Another benefit to using MongoDB with Spring Boot is not having to declare queries in the same sense as defining SQL queries using Prepared Statements using in Java. Here, by extending our own interface as a MongoRepository and simply outlining the query as a method signature we can let Spring Boot to implement those query methods during the runtime saving us the hassle of implementing such redundant queries from scratch for use. For more complex requirements where we cannot directly declare a method signature, we can use multiple simple queries to fulfill such a request with easily.

4. Workflow (Frontend)

Frontend is basically a set of web pages that are designed by using HTML, ReactJS, JavaScript and CSS framework Bootstrap. Content is dynamically changed by using a JavaScript framework called jQuery and for Axios library is used to manage all the requests sent to the backend as well as their responses.

The overall architecture of the frontend loosely follow the MVC architecture since the web pages themselves are the view and the JavaScript functions act as controllers while there's no clear model as such in the system.

5. Appendix

For a clearer view, you can also refer to my Github repositories:

https://github.com/KaushiRajapakshe/DS_Frontend

https://github.com/KaushiRajapakshe/DS_Backend

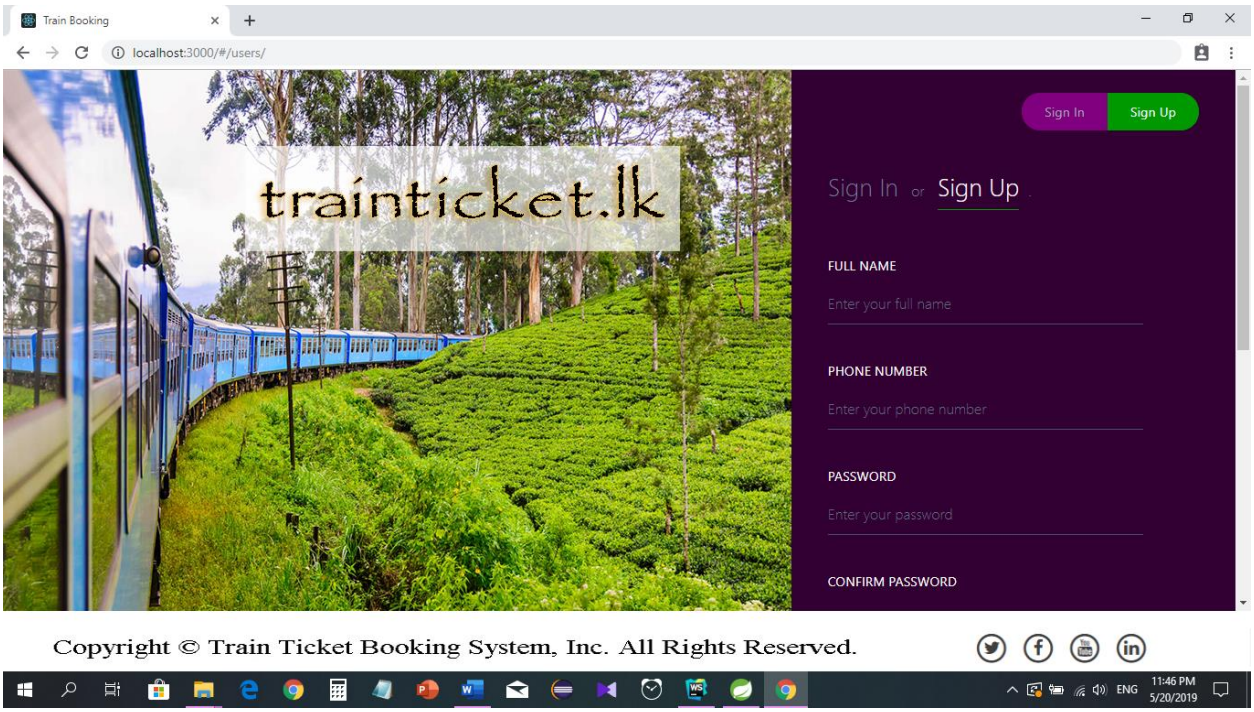

Fig 4: SignUp Page

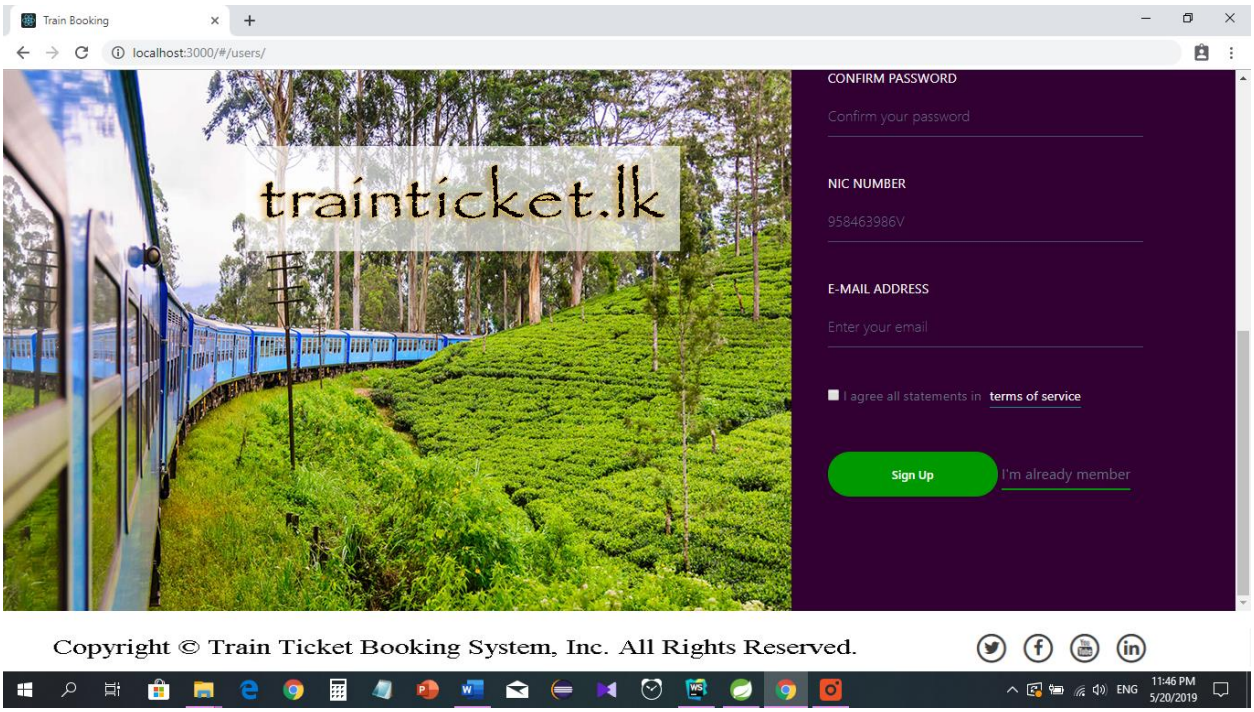

Fig 5: SignUp Page

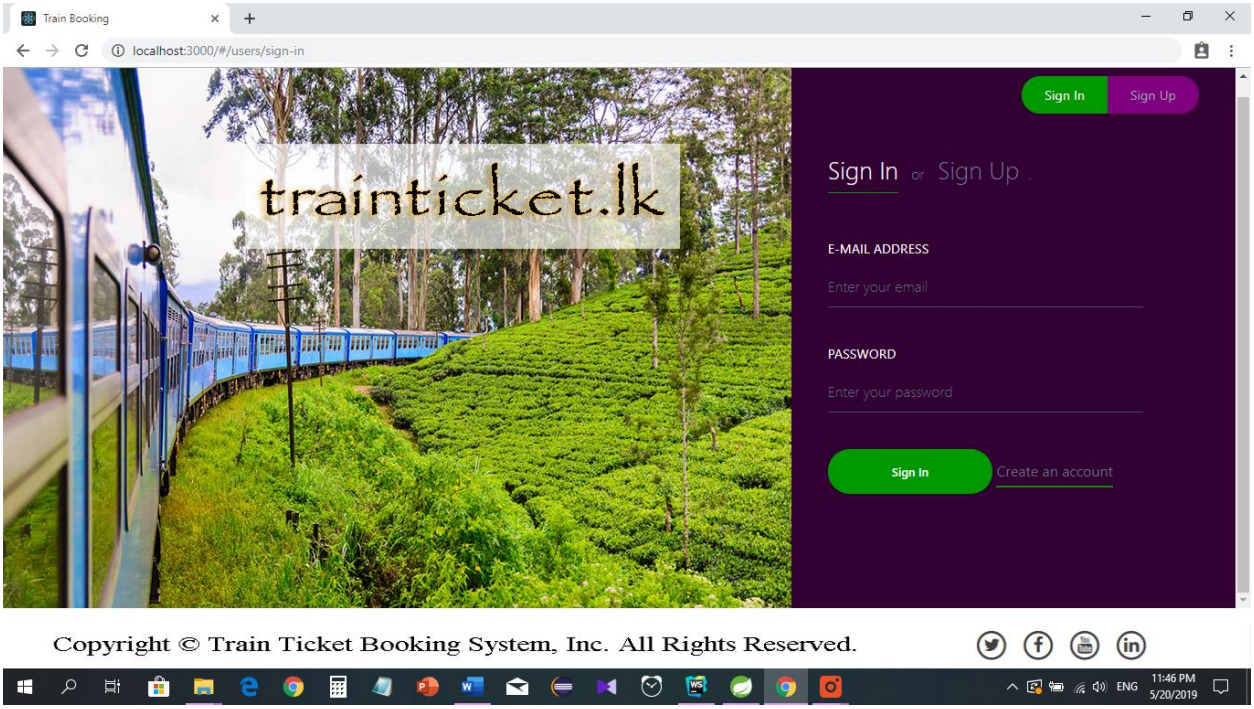

Fig 6: SignIn Page

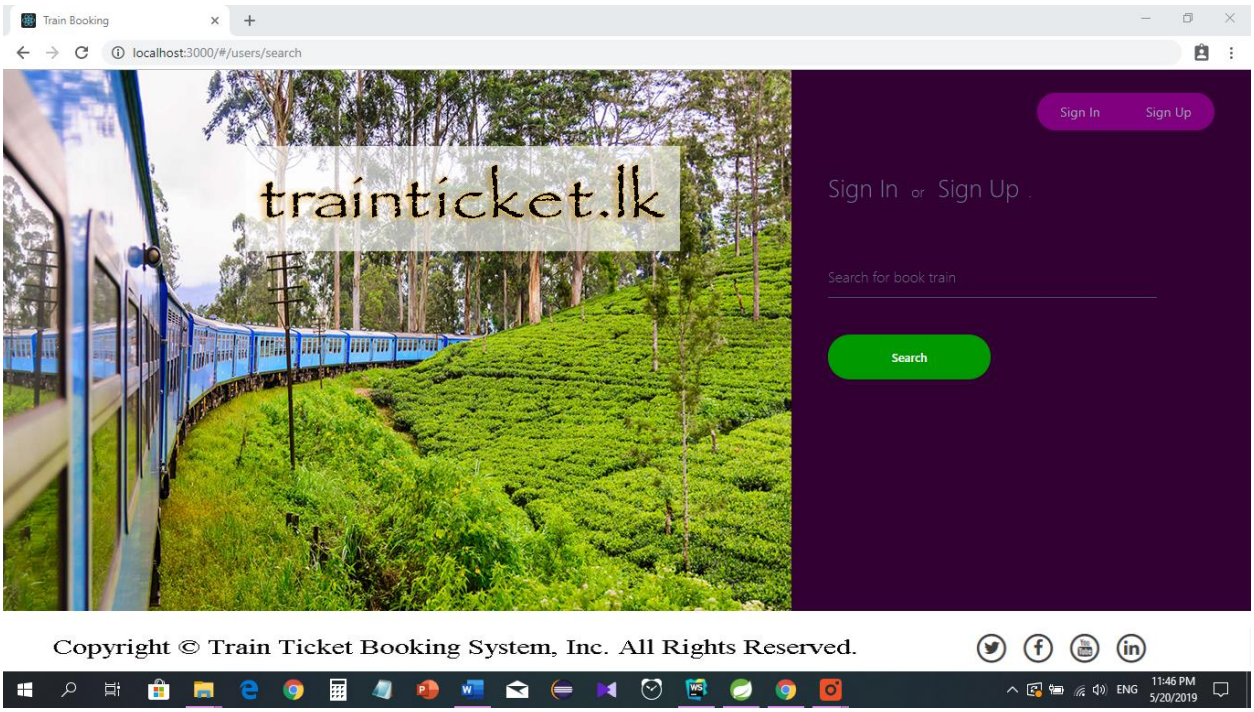

Fig 7: Search Page

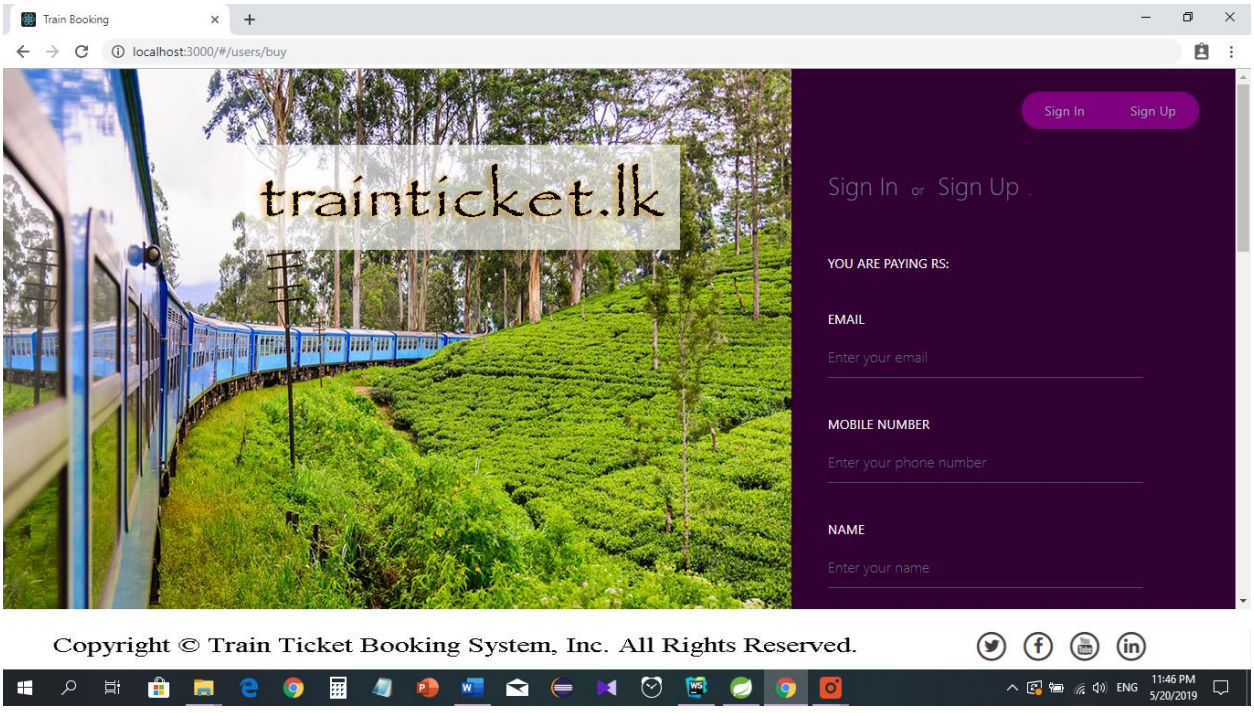

Fig 8: Payment Page

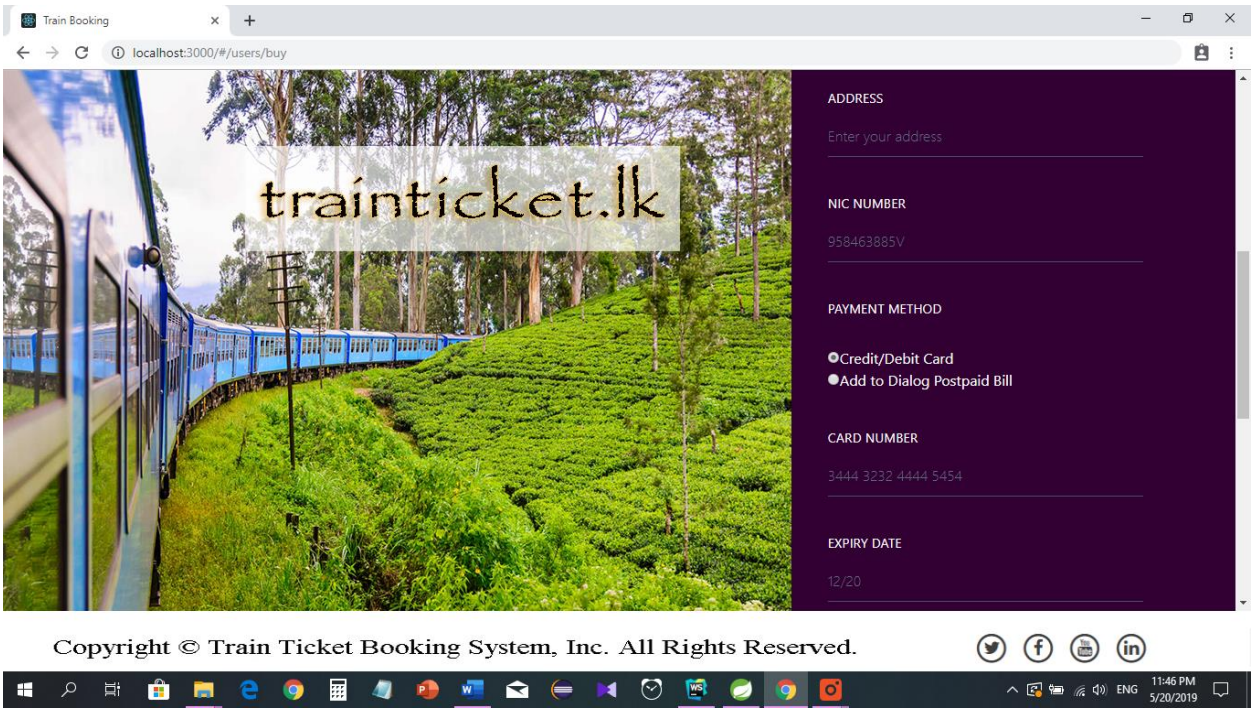

Fig 9: Payment Page

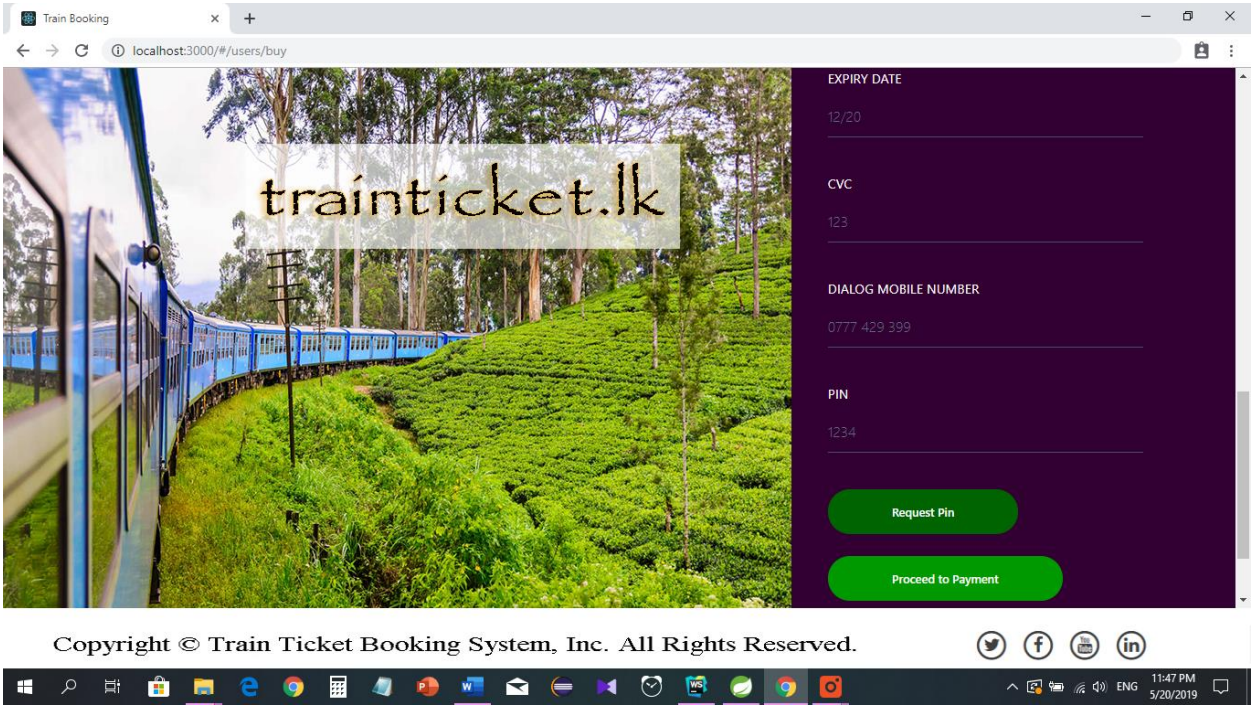

Fig 10: Payment Page

5.1 Backend

5.1.1 Controllers

TicketBookingController

```
package com.example.sliit.ds.controller;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.example.sliit.ds.config.Config;
import com.example.sliit.ds.entities.Ticket;
import com.example.sliit.ds.repository.TicketRepository;

@RestController
@RequestMapping(value = "/api")
@CrossOrigin(origins = Config.allowedOrigin)
public class TickectBookingController {

	@Autowired
	private TicketRepository ticketRepository;

	// Retrieve operation
	@RequestMapping(method=RequestMethod.GET, value="/tickets")
    public Iterable<Ticket> ticket() {
        return ticketRepository.findAll();
    }

	// Create operation
    @RequestMapping(method=RequestMethod.POST, value="/tickets")
    public Ticket save(@RequestBody Ticket ticket) {
	ticketRepository.save(ticket);

        return ticket;
    }

    @RequestMapping(method=RequestMethod.GET, value="/tickets/{id}")
    public Ticket show(@PathVariable String id) {
        return ticketRepository.findByTicketId(id);
    }
```

```
    // Update operation
    @RequestMapping(method=RequestMethod.PUT, value="/tickets/{id}")
    public Ticket update(@PathVariable String id, @RequestBody Ticket
ticket) {
        Ticket optticket = ticketRepository.findByTicketId(id);
         Ticket t = optticket;
         if(ticket.getPassengerName() != null)
             t.setPassengerName(ticket.getPassengerName());
         if(ticket.getBookingDate() != null)
             t.setBookingDate(ticket.getBookingDate());
         if(ticket.getSourceStation() != null)
             t.setSourceStation(ticket.getSourceStation());
         if(ticket.getDestStation() != null)
             t.setDestStation(ticket.getDestStation());
         if(ticket.getEmail() != null)
             t.setEmail(ticket.getEmail());
         ticketRepository.save(t);
         return t;
    }

    // Delete operation
    @RequestMapping(method=RequestMethod.DELETE,
value="/tickets/{id}")
    public String delete(@PathVariable String id) {
         Ticket optticket = ticketRepository.findByTicketId(id);
         ticketRepository.delete(optticket);

         return "";     }


}
```

UserController

```
package com.example.sliit.ds.controller;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.*;

import com.example.sliit.ds.entities.User;
import com.example.sliit.ds.repository.UserRepository;
import com.example.sliit.ds.config.Config;

@RestController
@RequestMapping(value = "/api")
@CrossOrigin(origins = Config.allowedOrigin)
public class UserController {
      @Autowired
    UserRepository userRepository;

      // Retrieve operation
    @RequestMapping(method=RequestMethod.GET, value="/users")
    public Iterable<User> user() {
        return userRepository.findAll();
    }

    // Insert operation
    @RequestMapping(method=RequestMethod.POST, value="/users")
    public User save(@RequestBody User user) {
      userRepository.save(user);

        return user;
    }

    @RequestMapping(method=RequestMethod.GET, value="/users/{id}")
    public Optional<User> show(@PathVariable String id) {
        return userRepository.findById(id);
    }

    // Update operation
    @RequestMapping(method=RequestMethod.PUT, value="/users/{id}")
    public User update(@PathVariable String id, @RequestBody User
user) {
        Optional<User> optcontact = userRepository.findById(id);
        User u = optcontact.get();
        if(user.getName() != null)
```

```
            u.setName(user.getName());
        if(user.getEmail() != null)
            u.setEmail(user.getEmail());
        if(user.getPassword() != null)
            u.setPassword(user.getPassword());
        if(user.getCpasswrd() != null)
            u.setCpasswrd(user.getCpasswrd());
        if(user.getNic() != null)
            u.setNic(user.getNic());
        userRepository.save(u);
        return u;
    }

    // Delete operation
    @RequestMapping(method=RequestMethod.DELETE, value="/users/{id}")
    public String delete(@PathVariable String id) {
        Optional<User> optuser = userRepository.findById(id);
        User user = optuser.get();
        userRepository.delete(user);

        return "";
    }
}
```

PaymentController

```
package com.example.sliit.ds.controller;

import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.example.sliit.ds.entities.Payment;
import com.example.sliit.ds.payment.connecter.PaymentConnecter;
import com.example.sliit.ds.service.PaymentServiceImpl;
import com.example.sliit.ds.service.SessionServiceImpl;
import com.example.sliit.ds.service.TicketServiceImpl;

public class PaymentController {
      @Autowired
    private PaymentServiceImpl paymentService = new
PaymentServiceImpl();

    @Autowired
    private SessionServiceImpl sessionService = new
SessionServiceImpl();

    @Autowired
    private TicketServiceImpl ticketService = new TicketServiceImpl();

    private PaymentConnecter paymentConnecter = new
PaymentConnecter();


    // GET all payment entries in the database.
    @RequestMapping(method = RequestMethod.GET)
    public ResponseEntity<List<Payment>> getAllPaymentEntries() {
        return new ResponseEntity<>(paymentService.getAllPayments(),
HttpStatus.OK);
    }

    // GET payment by its payment id.
    @RequestMapping(value = "/pid/{pid}", method = RequestMethod.GET)
    public ResponseEntity<Payment> getPayment(@PathVariable("pid")
String pid) {
```

```
        return new ResponseEntity<>(paymentService.findByPid(pid),
HttpStatus.OK);
    }

    // GET payment entries by the user id.
    @RequestMapping(value = "/uid/{uid}", method = RequestMethod.GET)
    public ResponseEntity<List<Payment>>
getAllPaymentEntriesByUser(@PathVariable("uid") String uid) {
        return new ResponseEntity<>(paymentService.findByUid(uid),
HttpStatus.OK);
    }

    // GET total price of food ordering.
    @RequestMapping(value = "/total", method = RequestMethod.POST)
    public ResponseEntity<Map<String, Object>>
getTotalAmountToPay(@RequestHeader("Authentication") long authKey,
@RequestBody Map<String, Object> payload) {
        Map<String, Object> response = new HashMap<>();
        double total = 0;

        // go through each ticket id and it's count.
        for (String tid: payload.keySet()) {
            int itemCount =
Integer.parseInt(payload.get(tid).toString());
            double itemPrice = ticketService.getPriceOf(tid);
            double subtotal = itemPrice * itemCount;

            total += subtotal;
        }

        response.put("success", "true");
        response.put("amount", total);

        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    // GET loyalty points count of a given user.
    @RequestMapping(value = "/{uid}/loyalty", method =
RequestMethod.GET)
    public ResponseEntity<Map<String, String>>
getTotalLoyaltyPoints(@RequestHeader("Authentication") String authKey,
@PathVariable("uid") String uid) {
        Map<String, String> response = new HashMap<>();

        if (sessionService.authenticate(authKey)) {
            List<Payment> paymentsByUser =
paymentService.findByUid(uid);
            double tot = 0;

      for(Payment payment: paymentsByUser) { tot += payment.getLoyaltyPoints(); }
```

```
            response.put("success", "true");
            response.put("loyalty", Double.toString(tot));
        }
        else {
            response.put("success", "false");
        }

        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    // ADD new entry for payment.
    @RequestMapping(method = RequestMethod.POST)
    public ResponseEntity<Map<String, String>>
addPayment(@RequestHeader("Authentication") String authKey,
@RequestBody Payment payment) {
        Map<String, String> response = new HashMap<>();

        if (sessionService.authenticate(authKey)) {
            payment.setPid((payment.getUid() +
payment.getPaymentDate().toString() + new
Date().toString()).toString());

            // instead the fId and count of each is sent.
            double tot = 0;
            Map<String, Integer> items = payment.getItemsAndCounts();
            // calculate
            for (String tId: items.keySet()) { tot +=
(ticketService.getPriceOf(tId) * items.get(tId)); }
            payment.setAmount(tot);

            // give loyalty points for the amount.
            payment.setLoyaltyPoints(tot * (0.01/100));

            // for credit cards we only keep the last 4 digits of the
card number for,
            // security concerns.
            if (payment.getPaymentType().equals("card")) {
                Map<String, String> paymentDetails =
payment.getPaymentDetails();
                String cardNumber = paymentDetails.get("number");
                String censoredCardNumber = "xxxx-xxxx-xxxx-" +
cardNumber.substring(cardNumber.length()-4, cardNumber.length());
                paymentDetails.put("number", censoredCardNumber);
            }

            // but we'll direct if the payment is successful.
            paymentService.savePayment(payment);
            response.put("success", "true");
            response.put("redirect", "home.html");
            response.put("pid", payment.getPid());
```

```
            }
            else {
                response.put("success", "false");
                response.put("redirect", "buy.html");
            }

            return new ResponseEntity<>(response, HttpStatus.OK);
        }


}
```

5.1.2 Models

Ticket

```
package com.example.sliit.ds.entities;

import java.util.Date;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class Ticket {

	@Id
	private String ticketId;
	private String passengerName;
	private Date bookingDate;
	private String sourceStation;
	private String destStation;
	private String email;
	private double price;

	public Ticket() {}

	public double getPrice() {
		return price;
	}

	public void setPrice(double price) {
		this.price = price;
	}

	public String getTicketId() {
		return ticketId;
	}
	public void setTicketId(String ticketId) {
		this.ticketId = ticketId;
	}
	public String getPassengerName() {
		return passengerName;
	}
	public void setPassengerName(String passengerName) {
		this.passengerName = passengerName;
	}
	public Date getBookingDate() {
		return bookingDate;
	}
	public void setBookingDate(Date bookingDate) {
		this.bookingDate = bookingDate;
```

```
	}
	public String getSourceStation() {
		return sourceStation;
	}
	public void setSourceStation(String sourceStation) {
		this.sourceStation = sourceStation;
	}
	public String getDestStation() {
		return destStation;
	}
	public void setDestStation(String destStation) {
		this.destStation = destStation;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}

	public double getPriceOf(String ticketId){
		return price;
	}
	@Override
	public String toString(){
		return "Ticket [ticketId = "+ticketId+", passengerName =
"+passengerName+", bookingDate = "+bookingDate+", sourceStation =
"+sourceStation+", destStation = "+destStation+", email = "+email+"]";
	}
}
```

User

```
package com.example.sliit.ds.entities;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "user")
public class User {

	@Id
	private String id;
	private String name;
	private String email;
	private String password;
	private String cpasswrd;

	public String getCpasswrd() {
		return cpasswrd;
	}

	public void setCpasswrd(String cpasswrd) {
		this.cpasswrd = cpasswrd;
	}

	public String getNic() {
		return nic;
	}

	public void setNic(String nic) {
		this.nic = nic;
	}

	private String nic;

	public User(){}

	public String getId() {
		return id;
	}

	public void setId(String id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
```

```
		this.name = name;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}
}
```

Payment

```
package com.example.sliit.ds.entities;

import java.util.Date;
import java.util.Map;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "payment")
public class Payment {

    @Id
    private String pid;
    private String uid;
    private double amount;
    private double loyaltyPoints;
    private Map<String, Integer> itemsAndCounts;
    private String paymentType;
    private Date paymentDate;
    private Map<String, String> paymentDetails;

    // two payment types:- credit card and dialog post paid.
    public Payment() { }

    public String getPid() {
        return pid;
    }

    public Date getPaymentDate() {
        return paymentDate;
    }

    public void setPaymentDate(Date paymentDate) {
        this.paymentDate = paymentDate;
    }

    public void setPid(String pid2) {
        this.pid = pid2;
    }

    public String getUid() {
        return uid;
    }

    public void setUid(String uid) {
        this.uid = uid;
    }
```

```
    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public double getLoyaltyPoints() {
        return loyaltyPoints;
    }

    public void setLoyaltyPoints(double loyaltyPoints) {
        this.loyaltyPoints = loyaltyPoints;
    }

    public Map<String, Integer> getItemsAndCounts() {
        return itemsAndCounts;
    }

    public void setItemsAndCounts(Map<String, Integer> itemsAndCounts)
{
        this.itemsAndCounts = itemsAndCounts;
    }

    public String getPaymentType() {
        return paymentType;
    }

    public void setPaymentType(String paymentType) {
        this.paymentType = paymentType;
    }

    public Map<String, String> getPaymentDetails() {
        return paymentDetails;
    }

    public void setPaymentDetails(Map<String, String> paymentDetails)
{
        this.paymentDetails = paymentDetails;
    }
}
```

Session

```
package com.example.sliit.ds.entities;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "session")
public class Session {

    @Id
    private String uid;
    private String authKeyOfUid;
    private String role;

    public Session() { }

    public String getUid() {
        return uid;
    }

    public void setUid(String uid) {
        this.uid = uid;
    }

    public String getAuthKeyOfUid() {
        return authKeyOfUid;
    }

    public void setAuthKeyOfUid(String authKeyOfUid) {
        this.authKeyOfUid = authKeyOfUid;
    }

    public String getRole() {
        return role;
    }

    // registering.
    public void setRole(String role) {
        this.role = role;
    }
}
```

5.1.3 Repositories

TicketReposirory

```
package com.example.sliit.ds.repository;

import java.util.Optional;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.example.sliit.ds.entities.Ticket;

@Repository
public interface TicketRepository extends CrudRepository<Ticket,
String>{
	@Override
	void delete(Ticket deleted);
	Ticket findByTicketId(String ticketId);
	Ticket findByPassengerName(String name);
	Optional<Ticket> findByBookingDate(String date);
}
```

UserReposirory

```
package com.example.sliit.ds.repository;

import java.util.Optional;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.example.sliit.ds.entities.User;

@Repository
public interface UserRepository extends CrudRepository<User, String>{
	@Override
	void delete(User deleted);
	Optional<User> findById(String id);
}
```

PaymentRepository

```
package com.example.sliit.ds.repository;

import java.util.Date;
import java.util.List;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.example.sliit.ds.entities.Payment;

@Repository
public interface PaymentRepository extends CrudRepository<Payment,
String> {

    Payment findByPid(String pid);

    List<Payment> findByPaymentDate(Date date);
    List<Payment> findByUid(String uid);

    void deleteByPid(Payment payment);
}
```

SessionRepository

```
package com.example.sliit.ds.repository;

import java.util.List;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.example.sliit.ds.entities.Session;

@Repository
public interface SessionRepository extends CrudRepository<Session,
String> {

    Session findByUid(String uid);
    Session findByAuthKeyOfUid(String authKey);
    List<Session> findByRole(String role);

    void deleteByAuthKeyOfUid(String authKey);
    void deleteByUid(String uid);
}
```

5.1.4.1 Service Interface

TicketBookingService

```
package com.example.sliit.ds.service;

import java.util.List;
import java.util.Optional;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.example.sliit.ds.entities.Ticket;

@Service("ticketService")
@Transactional
public interface TicketBookingService {
      Ticket findById(String tId);

      Ticket findByName(String name);

    List<Ticket> findPassengerName(String name);

    Optional<Ticket> findBookingDate(String date);

    void saveTicket(Ticket ticket);

    void updateTicket(Ticket ticketUpdate);

    void deleteTicketById(String tId);

    List<Ticket> findAllTicket();

    public boolean isFoodExist(Ticket ticket);
}
```

UserService

```
package com.example.sliit.ds.service;

import java.util.List;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.example.sliit.ds.entities.User;

@Service("userService")
@Transactional
public interface UserService {

      User findByUid(String uid);

    User findByEmail(String email);

    long getUidOfEmail(String email);

    String getEmailOfUid(long uid);

    List<User> findUsersHavingName(String name);

    void saveUser(User user);

    void updateUser(long uid, User userUpdate);

    void deleteUserByEmail(String email);

    List<User> findAllUsers();

    boolean isUserExist(long uid);

    boolean isPasswordCorrect(String email, long password);


}
```

PaymentService

```
package com.example.sliit.ds.service;

import java.util.Date;
import java.util.List;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.example.sliit.ds.entities.Payment;

@Service("paymentService")
@Transactional
public interface PaymentService {

	Payment findByPid(String pid);

	List<Payment> findByUid(String uid);

	List<Payment> findByPaymentDate(Date date);

	List<Payment> getAllPayments();

    void savePayment(Payment payment);

    void updatePayment(String pid, Payment paymentUpdate);

    void deletePayment(Payment payment);

    boolean isPaymentExist(Payment payment);
}
```

SessionService

```
package com.example.sliit.ds.service;

import java.util.List;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.example.sliit.ds.entities.Session;

@Service("sessionService")
@Transactional
public interface SessionService {

	Session getSessionOf(String uid);
// returns both authentication key and the user id as an Object.

	List<Session> getAdminSessions();
// returns sessions that has an Admin role.

	List<Session> getUserSessions();
// returns sessions that has user role.

	String getUidOfSession(String authKey);
// returns the user id of the user to whom the authentication key
belongs.

	String getRoleOfAuthentication(String authKey);

	boolean authenticate(String autheKey);
// simply checks if the authentication key exists in the database :-)

	void saveSession(Session session);

	void invokeSession(Session session);

	void invokeSessionOfUser(String uid);
// delete the session entry that belongs to the user id.

	void invokeSessionOfKey(String authKey);
// delete the session entry that contains the given auth key.

}
```

5.1.4.2 Service Implementation

TicketServiceImpl

```
package com.example.sliit.ds.service;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.sliit.ds.entities.Ticket;
import com.example.sliit.ds.repository.TicketRepository;

@Service("ticketService")
@Transactional
public class TicketServiceImpl implements TicketBookingService{

	@Autowired
    private TicketRepository ticketRepository;

    private static List<Ticket> ticket;

	@Override
	public Ticket findById(String tId) {
		return ticketRepository.findByTicketId(tId);
	}

	@Override
	public Ticket findByName(String name) {
		return ticketRepository.findByPassengerName(name);
	}

	@Override
	public List<Ticket> findPassengerName(String name) {
		Iterable<Ticket> foodIterable = (Iterable<Ticket>)
ticketRepository.findByPassengerName(name);
		List<Ticket> foodList = new ArrayList<>();

		foodIterable.forEach(foodList::add);
		return foodList;
	}

	@Override
	public Optional<Ticket> findBookingDate(String date) {
		Optional<Ticket> foodIterable =
ticketRepository.findByBookingDate(date);
		return foodIterable;
	}
```

```
	@Override
	public void saveTicket(Ticket ticket) {
		ticketRepository.save(ticket);

	}

	@Override
	public void updateTicket(Ticket ticketUpdate) {
		Ticket ticketExisting =
findById(ticketUpdate.getTicketId());

	   // update the item we retrieved.
	   if (ticketExisting != null) {

	ticketExisting.setSourceStation(ticketUpdate.getSourceStation());

	ticketExisting.setPassengerName(ticketUpdate.getPassengerName());

	ticketExisting.setPassengerName(ticketUpdate.getPassengerName());

	ticketExisting.setDestStation(ticketUpdate.getDestStation());
		ticketExisting.setEmail(ticketUpdate.getEmail());
		 saveTicket(ticketExisting);
	   }

	}

	@Override
	public void deleteTicketById(String tId) {
		ticketRepository.findByTicketId(tId);

	}

	@Override
	public List<Ticket> findAllTicket() {
		return (List<Ticket>) ticketRepository.findAll();
	}

	@Override
	public boolean isFoodExist(Ticket ticket) {
		return ticketRepository.exists(ticket.getTicketId());
	}

	public double getPriceOf(String tId) {
		Ticket ticket = new Ticket();
		return ticket.getPriceOf(tId);
	}

}
```

UserServiceImpl

```
package com.example.sliit.ds.service;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.example.sliit.ds.entities.User;
import com.example.sliit.ds.repository.UserRepository;

@Service("userService")
@Transactional
public class UserServiceImpl {

      @Autowired
    private UserRepository userRepository;

    public Optional<User> findById(String uid) {
        return userRepository.findById(uid);
    }

}
```

PaymentServiceImpl

```
package com.example.sliit.ds.service;

import java.util.Date;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.sliit.ds.entities.Payment;
import com.example.sliit.ds.repository.PaymentRepository;

@Service("paymentService")
@Transactional
public class PaymentServiceImpl implements PaymentService{

      @Autowired
    private PaymentRepository paymentRepository;

    @Override
    public Payment findByPid(String pid) {
        return paymentRepository.findByPid(pid);
    }

    @Override
    public List<Payment> findByUid(String uid) {
        return paymentRepository.findByUid(uid);
    }

    @Override
    public List<Payment> findByPaymentDate(Date date) {
        return paymentRepository.findByPaymentDate(date);
    }

    @Override
    public List<Payment> getAllPayments() {
        List<Payment> l = (List<Payment>) paymentRepository.findAll();
        System.out.println(l.size());
        return (List<Payment>) paymentRepository.findAll();
    }
    @Override
    public void savePayment(Payment payment) {
        paymentRepository.save(payment);
    }
    @Autowired
      @Override
    public void updatePayment(String pid, Payment paymentUpdate) {
```

```
        if (pid != paymentUpdate.getPid()) {
paymentUpdate.setPid(pid); }

        Payment paymentExisting = findByPid(pid);

        if (paymentExisting != null) {
            paymentExisting.setAmount(paymentUpdate.getAmount());

paymentExisting.setPaymentDetails(paymentUpdate.getPaymentDetails());

paymentExisting.setPaymentType(paymentUpdate.getPaymentType());
            paymentExisting.setUid(paymentUpdate.getUid());

            savePayment(paymentExisting);
        }
    }
    @Override
    public void deletePayment(Payment payment) {

    }
    @Override
    public boolean isPaymentExist(Payment payment) {
        return false;
    }
}
```

SessionServiceImpl

```
package com.example.sliit.ds.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.example.sliit.ds.entities.Session;
import com.example.sliit.ds.repository.SessionRepository;

@Service("sessionService")
@Transactional
public class SessionServiceImpl implements SessionService {

    @Autowired
    private SessionRepository sessionRepository;

    @Override
    public Session getSessionOf(String uid) {
        return sessionRepository.findByUid(uid);
    }
```

```
    @Override
    public List<Session> getAdminSessions() {
        return sessionRepository.findByRole("admin");
    }

    @Override
    public List<Session> getUserSessions() {
        return sessionRepository.findByRole("user");

    }

    @Override
    public String getUidOfSession(String authKey) {
        Session session =
sessionRepository.findByAuthKeyOfUid(authKey);
        return (session != null) ? session.getUid() : "0";
    }

    @Override
    public String getRoleOfAuthentication(String authKey) {
        Session session =
sessionRepository.findByAuthKeyOfUid(authKey);
        String role = "invalid";

        if (session != null) {
            role = session.getRole();
        }

        return role;
    }

    @Override
    public boolean authenticate(String autheKey) {
        Session session =
sessionRepository.findByAuthKeyOfUid(autheKey);
        return (session != null && session.getAuthKeyOfUid() ==
autheKey);
    }

    @Override
    public void saveSession(Session session) {
        sessionRepository.save(session);
    }

    @Override
    public void invokeSession(Session session) {
        sessionRepository.delete(session);
    }

    @Override
```

```
    public void invokeSessionOfUser(String uid) {
        sessionRepository.deleteByUid(uid);
    }

    @Override
    public void invokeSessionOfKey(String authKey) {
        sessionRepository.deleteByAuthKeyOfUid(authKey);
    }
}
```

PaymentGatewaySampath

```
package com.example.sliit.ds.payment.gateway.sampath;

import java.util.Map;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping(value = "spg/card/authenticate")
public class PaymentGatewaySampath {

      @RequestMapping(value = "/", method = RequestMethod.POST)
    public ResponseEntity<String> authenticatePayment(@RequestBody
Map<String, Object> payload) {
        String response;
        int number =
Integer.parseInt(payload.get("number").toString());
        int ccv = Integer.parseInt(payload.get("ccv").toString());
        String expiry = payload.get("expiry").toString();

        if (ValidatorPayment.areCardDetailsValid(number, ccv, expiry))
{
            response = "authorized";
        }
        else {
            response = "unauthorized";
        }

        return new ResponseEntity<>(response, HttpStatus.OK);
    }
}
```

ValidationPayment

```
package com.example.sliit.ds.payment.gateway.sampath;

import java.util.Calendar;

public class ValidatorPayment {

	/*
	* @param expiry
	*		states the expire date of the card, in the format mm/yy => 11/22 means 11th month of 2022.
	*/
    public static boolean areCardDetailsValid(int number, int ccv, String expiry){
		// since we are emulating a dummy payment gateway, we assume any card number of length 16,
		// is correct.
		boolean digitValidity = ( (Integer.toString(number).length() == 16) && (Integer.toString(ccv).length() == 3) );

		// validating expiry date.
		boolean dateValidity = false;
		if (expiry.contains("/")) {
			String[] dateParts = expiry.split("/");
			int month = Integer.parseInt(dateParts[0]);
			int year = Integer.parseInt(dateParts[1]);
			int currentYear = Calendar.getInstance().get(Calendar.YEAR);


			dateValidity = ( (month > 0 && month <= 12) && (year >= currentYear) );
		}

		return (digitValidity && dateValidity);
    }
}
```

PaymentConnecter

```
package com.example.sliit.ds.payment.connecter;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.util.Map;

import com.example.sliit.ds.entities.Payment;

public class PaymentConnecter {
     private static final String gatewayAuthenticationURL =
"http://localhost:8081/spg/card/authenticate";

    // secondary method will be called.
    public boolean makePayment(Payment payment) throws IOException {
        switch (payment.getPaymentType()) {
            case "card":
                return makeCardPayment(payment.getPaymentDetails());

            case "bill":
                return
makeDialogBillPayment(payment.getPaymentDetails());

            default:
                return false;
        }
    }

    public boolean makeCardPayment(Map<String, String> paymentDetails)
throws IOException {
        // contact Samptha Payment Gateway.
        URL reqUrl = new
URL(PaymentConnecter.gatewayAuthenticationURL);
        doPOST(reqUrl, paymentDetails);
        return true;
    }

    public boolean makeDialogBillPayment(Map<String, String>
paymentDetails) {
        return false;
    }
```

```
    private String doPOST(URL requestUrl, Map<String, String> payload)
throws IOException {
        String urlParams = "number=" + payload.get("number") + "&" +
                           "cvc=" + payload.get("cvc") + "&" +
                           "expiry" + payload.get("expiry");
        byte[] urlParamsAsPostData =
urlParams.getBytes(StandardCharsets.UTF_8);
        HttpURLConnection connection =
(HttpURLConnection)requestUrl.openConnection();

        connection.setRequestMethod("POST");
        // set headers.
        connection.setRequestProperty("User-Agent", "Java Client");
        connection.setRequestProperty("Content-Type", "application/x-
www-form-urlencoded");
        connection.setDoOutput(true);

        // send request.
        try(DataOutputStream out = new
DataOutputStream(connection.getOutputStream())) {
            out.write(urlParamsAsPostData);
        }

        // read response.
        String response = "";
        try (BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()))) {
            String line;

            while ( (line = in.readLine()) != null ) {
                response += line;
            }

            System.out.println(response);
        }
        finally {
            connection.disconnect();
        }

        return response;
    }
}
```

5. 2 Frontend

5.2.1 JavaScript Pages

App.js

```
import React, {Component} from 'react';
import {HashRouter as Router, Route, Link, NavLink} from 'react-router-dom';
import SignUp from './pages/SignUp';
import SignIn from './pages/SignIn';
import Buy from './pages/Buy';
import Search from './pages/Search';
import footer from './images/footer.png';

import './App.css';

class App extends Component {
    render() {
        return (
            <Router basename="/users/">
                <div className="App">
                    <div className="App__Aside"></div>
                    <div className="App__Form">
                        <div className="PageSwitcher">
                            <NavLink to="/sign-in"
activeClassName="PageSwitcher__Item--Active"
                                     className="PageSwitcher__Item"
                            >Sign In</NavLink>
                            <NavLink exact to="/"
activeClassName="PageSwitcher__Item--Active"
                                     className="PageSwitcher__Item"
                            >Sign Up</NavLink>
                        </div>

                        <div className="FormTitle">
                            <NavLink to="/sign-in"
activeClassName="FormTitle__Link--Active"
                                     className="FormTitle__Link">Sign In
                            </NavLink> or <NavLink exact to="/"
activeClassName="FormTitle__Link--Active"

className="FormTitle__Link"
                        >Sign Up</NavLink>.
                        </div>

                        <Route exact path="/" component={SignUp}>
                        </Route>
                        <Route path="/sign-in" component={SignIn}>
                        </Route>
                        <Route path="/search" component={Search}>
                        </Route>
                        <Route exact path="/buy" component={Buy}>
                        </Route>
                    </div>
```

```
                    <div className="footer">
                        <Footer/>
                    </div>
                </div>
            </Router>

        );
    }
}

class Footer extends Component {
    render() {
        return (
            <div className="footer">
                <img className="footerM" src={footer} style={{width: '100%'}}
alt={footer}/>
            </div>
        );
    }
}

export default App;
```

App.css

```
.App {
  height: 90vh;
  display: flex;
  color: white;
  float: top;
}

.App__Aside {
  width: 90%;
  background-image: url("./images/back1.png");
}

.App__Form {
  width: 50%;
  background-color: #330033;
  padding: 25px 40px;
  overflow: auto;
}

.PageSwitcher {
  display: flex;
  justify-content: flex-end;
  margin-bottom: 10%;
}

.PageSwitcher__Item {
  background-color: #800080;
```

```
  color: #9DA6B1;
  padding: 10px 25px;
  cursor: pointer;
  font-size: .9em;
  border: none;
  outline: none;
  display: inline-block;
  text-decoration: none;
}

.PageSwitcher__Item--Active {
  background-color: #009900;
  color: white;
}

.PageSwitcher__Item:first-child {
  border-top-left-radius: 25px;
  border-bottom-left-radius: 25px;
}
.PageSwitcher__Item:last-child {
  border-top-right-radius: 25px;
  border-bottom-right-radius: 25px;
}

.FormCenter {
  margin-bottom: 100px;
}

.FormTitle {
  color: #707C8B;
  font-weight: 300;
  margin-bottom: 50px;
}

.FormTitle__Link {
  color: #707C8B;
  text-decoration: none;
  display: inline-block;
  font-size: 1.7em;
  margin: 0 10px;
  padding-bottom: 5px;
}

.FormTitle__Link:first-child {
  margin-left: 0;
}

.FormTitle__Link--Active {
  color: white;
  border-bottom: 1px solid #009900;
}

.FormField {
  margin-bottom: 40px;
}
```

```
.FormField__Label {
  display: block;
  text-transform: uppercase;
  font-size: .9em;
  color: white;
}
.FormField__Input {
  width: 85%;
  background-color: transparent;
  border: none;
  color: white;
  outline: none;
  border-bottom: 1px solid #445366;
  font-size: 1em;
  font-weight: 300;
  padding-bottom: 10px;
  margin-top: 10px;
}

.FormField__Input::placeholder {
  color: #616E7F;
}

.FormField__Button1 {
  background-color: darkgreen;
  color: white;
  border: none;
  outline: none;
  border-radius: 25px;
  padding: 15px 70px;
  font-size: .8em;
  font-weight: 500;
}
.FormField__Button {
  background-color: #009900;
  color: white;
  border: none;
  outline: none;
  border-radius: 25px;
  padding: 15px 70px;
  font-size: .8em;
  font-weight: 500;
}

.FormField__Link {
  color: #66707D;
  text-decoration: none;
  display: inline-block;
  border-bottom: 1.5px solid #009900;
  padding-bottom: 5px;
}

.FormField__CheckboxLabel {
  color: #646F7D;
  font-size: .9em;
}
```

```
.FormField__Checkbox {
  position: relative;
  top: 1.5px;
}

.FormField__TermsLink {
  color: white;
  border-bottom: 1px solid #199087;
  text-decoration: none;
  display: inline-block;
  padding-bottom: 2px;
  margin-left: 5px;
}

.footer{
  position: absolute;
  left: 0;
  bottom: 0;
  width: 100%;
  float: bottom;
  display: flex;
  padding-bottom: 5px;
  padding-top: 15px;
}
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App />, document.getElementById('root'));

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

Index.css

```
body {
  margin: 0;
  padding: 0;
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto",
"Oxygen",
```

```
    "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans", "Helvetica Neue",
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, "Courier New",
    monospace;
}
```

SignUp.js

```
import React, { Component } from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';
import '../jquery-3.3.1.min.js';
import '../bundle.js';

class SignUp extends Component {

    constructor(props) {
        super(props);

        this.state = {
            name: '',
            phoneNo: '',
            password: '',
            cpassword: '',
            nic: '',
            email: '',
            hasAgreed: false
        };

        this.handleChange = this.handleChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
    }

    handleChange(e) {
        let target = e.target;
        let value = target.type === 'checkbox' ? target.checked :
target.value;
        let name = target.name;

        this.setState({
            [name]: value
        });
    }

    handleSubmit(e) {
        e.preventDefault();
```

```
        const { name, phoneNo, password, cpassword, nic, email } =
this.state;
        console.log('The form was submitted with the following data:');
        console.log(this.state);
        axios.post('/api/users', {name, phoneNo, password, cpassword, nic,
email })
            .then((result) => {
                this.props.history.push("/")
            });
    }
        render() {
            const { name, phoneNo, password, cpassword, nic, email } =
this.state;
            return (
                <div className="FormCenter">
                    <form onSubmit={this.handleSubmit} className="FormFields"
onSubmit={this.handleSubmit}>
                        <div className="FormField">
                        <label className="FormField__Label"
htmlFor="name">Full Name</label>
                        <input type="text" id="name"
className="FormField__Input" placeholder="Enter your full name" name="name"
required value={this.state.name} onChange={this.handleChange} />
                    </div>
                    <div className="FormField">
                        <label className="FormField__Label"
htmlFor="phoneNo">Phone Number</label>
                        <input type="text" id="phoneNo"
className="FormField__Input" pattern="[0-9]{10}" placeholder="Enter your
phone number" name="phoneNo" required value={this.state.phoneNo}
onChange={this.handleChange} />
                    </div>
                     <div className="FormField">
                         <label className="FormField__Label"
htmlFor="password">Password</label>
                         <input type="password" id="password"
className="FormField__Input" placeholder="Enter your password"
name="password" required value={this.state.cpassword}
onChange={this.handleChange} />
                     </div>
                    <div className="FormField">
                        <label className="FormField__Label"
htmlFor="cpassword">Confirm Password</label>
                        <input type="password" id="cpassword"
className="FormField__Input" placeholder="Confirm your password"
name="cpassword" required value={this.state.cpassword}
onChange={this.handleChange} />
                    </div>
                     <div className="FormField">
                         <label className="FormField__Label"
htmlFor="nic">NIC Number</label>
                         <input type="text" id="nic"
className="FormField__Input" placeholder="958463986V" name="nic" required
value={this.state.nic} onChange={this.handleChange} />
                     </div>
                    <div className="FormField">
```

```
                        <label className="FormField__Label"
htmlFor="email">E-Mail Address</label>
                        <input type="email" id="email"
className="FormField__Input" placeholder="Enter your email" name="email"
required value={this.state.email} onChange={this.handleChange} />
                    </div>

                    <div className="FormField">
                        <label className="FormField__CheckboxLabel">
                            <input className="FormField__Checkbox"
type="checkbox" name="hasAgreed" required value={this.state.hasAgreed}
onChange={this.handleChange} /> I agree all statements in <a href=""
className="FormField__TermsLink">terms of service</a>
                        </label>
                    </div>

                    <div className="FormField">
                        <Link to="/sign-in"><button id="signup"
className="FormField__Button mr-20">Sign Up</button></Link> <Link to="/sign-
in" className="FormField__Link">I'm already member</Link>
                    </div>
                </form>
            </div>
        );
    }
}
export default SignUp;
```

SignIn.js

```
import React, {Component} from 'react';
import {Link} from 'react-router-dom';
import axios from 'axios';
import '../jquery-3.3.1.min.js';
import '../bundle.js';

class SignIn extends Component {
    constructor() {
        super();

        this.state = {
            email: '',
            password: ''
        };

        this.handleChange = this.handleChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
    }

    handleChange(e) {
        let target = e.target;
```

```
        let value = target.type === 'checkbox' ? target.checked :
target.value;
        let name = target.name;

        this.setState({
            [name]: value
        });
    }

    handleSubmit(e) {
        e.preventDefault();
        const { email, password } = this.state;
        axios.post('/api/users', { email, password })
            .then((result) => {
                this.props.history.push("/sign-in")
            });
        console.log('The form was submitted with the following data:');
        console.log(this.state);
    }

    render() {
        const { email, password } = this.state;
        return (
            <div className="FormCenter">
                <form onSubmit={this.handleSubmit} className="FormFields"
onSubmit={this.handleSubmit}>
                    <div className="FormField">
                        <label className="FormField__Label"
htmlFor="email">E-Mail Address</label>
                        <input type="email" id="email"
className="FormField__Input" placeholder="Enter your email"
                               name="email" value={this.state.email}
onChange={this.handleChange} required/>
                    </div>

                    <div className="FormField">
                        <label className="FormField__Label"
htmlFor="password">Password</label>
                        <input type="password" id="password"
className="FormField__Input"
                               placeholder="Enter your password"
name="password" value={this.state.password}
                               onChange={this.handleChange} required/>
                    </div>

                    <div className="FormField">
                        <a target="_blank" href="">
                            <Link to="/search">
                                <button id="signin"
className="FormField__Button mr-20">Sign In</button>
                            </Link>
                        </a> <Link to="/" className="FormField__Link">Create
an account</Link>
                    </div>
                </form>
            </div>
```

```
        );
    }
}

export default SignIn;
```

Search.js

```
import React, {Component} from 'react';
import {Link} from 'react-router-dom';
import '../../node_modules/bootstrap/dist/css/bootstrap.css';
import axios from 'axios';
import '../jquery-3.3.1.min.js';
import '../bundle.js';

import '../App.css';

class Search extends Component {

    constructor() {
        super();

        this.state = {
            search:'',
        };
        this.handleChange = this.handleChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
    }

    handleChange(e) {
        let target = e.target;
        let value = target.type === 'checkbox' ? target.checked :
target.value;
        let name = target.name;

        this.setState({
            [name]: value
        });
    }

    handleSubmit(e) {
        e.preventDefault();

        console.log('The form was submitted with the following data:');
        console.log(this.state);

        const search1 = {
            search: this.state.search,
        };
        const { search } = this.state;
        axios.post('/api/search', { search1 })
            .then((result) => {
                this.props.history.push("/search")
            });
```

```
        console.log('The form was submitted with the following data:');
        console.log(this.state);
    }

    render() {
        return (
            <div className="search">
                <div className="FormCenter">
                    <div className="FormField">
                    </div>
                    <form onSubmit={this.handleSubmit}
className="FormFields">
                        <div className="FormField">
                            <input type="text" id="searcht"
className="FormField__Input mr-20" placeholder="Search for book train "
                                   name="searcht" required
value={this.state.search} onChange={this.handleChange}/>
                        </div>
                        <div className="FormField">
                            <Link to="/buy">
                                <button id="search"
className="FormField__Button mr-20">Search</button>
                            </Link></div>
                    </form>
                </div>
            </div>
        );
    }
}

export default Search;
```

Buy.js

```
import React, {Component} from 'react';
import {Link} from 'react-router-dom';
import '../../node_modules/bootstrap/dist/css/bootstrap.css';
import axios from 'axios';
import '../jquery-3.3.1.min.js';
import '../bundle.js';

import '../App.css';

class Buy extends Component {

    constructor() {
        super();

        this.state = {
            email: '',
            mobile: '',
            name: '',
            address: '',
            nic: '',
```

```
            paymentMethod: '',
            cardNo: '',
            expireDate: '',
            cvc: '',
            mobileNo: '',
            pin: '',
        };
        this.handleChange = this.handleChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
    }

    handleChange(e) {
        let target = e.target;
        let value = target.type === 'checkbox' ? target.checked :
target.value;
        let name = target.name;

        this.setState({
            [name]: value
        });
    }

    handleSubmit(e) {
        e.preventDefault();

        console.log('The form was submitted with the following data:');
        console.log(this.state);

        const buy = {
            email: this.state.email,
            mobile: this.state.mobile,
            name: this.state.name,
            address: this.state.address,
            nic: this.state.nic,
            paymentMethod: this.state.paymentMethod,
            cardNo: this.state.cardNo,
            expireDate: this.state.expireDate,
            cvc: this.state.cvc,
            mobileNo: this.state.mobileNo,
            pin: this.state.pin,
        };
        const { email, mobile, name, address, nic, paymentMethod, cardNo,
expireDate, cvc, mobileNo, pin } = this.state;
        axios.post('/api/buy', { email, mobile, name, address, nic,
paymentMethod, cardNo, expireDate, cvc, mobileNo, pin })
            .then((result) => {
                this.props.history.push("/buy")
            });
    }

    render() {
        return (
            <div className="buy">
                <div className="FormCenter">
                    <div className="FormField">
                        <label className="FormField__Label" htmlFor="pay">You
```

```
are paying Rs:</label>
                    </div>
                    <form onSubmit={this.handleSubmit}
className="FormFields">
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="email">Email</label>
                            <input type="email" id="email"
className="FormField__Input" placeholder="Enter your email"
                                name="email" required
value={this.state.email} onChange={this.handleChange}/>
                        </div>
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="mobile">Mobile Number</label>
                            <input type="text" id="mobile" pattern="[0-
9]{10}" className="FormField__Input"
                                placeholder="Enter your phone number"
name="mobile" required
                                value={this.state.mobile}
onChange={this.handleChange}/>
                        </div>
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="name">Name</label>
                            <input type="text" id="name"
className="FormField__Input" onChange={this.handleChange}
                                placeholder="Enter your name" name="name"
required value={this.state.name}/>
                        </div>
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="address">Address</label>
                            <input type="text" id="address"
className="FormField__Input"
                                placeholder="Enter your address"
                                name="address" required=""
value={this.state.address} onChange={this.handleChange}/>
                        </div>
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="nic">NIC Number</label>
                            <input type="text" id="nic"
className="FormField__Input"
                                placeholder="958463885V"
                                name="nic" required=""
value={this.state.nic} onChange={this.handleChange}/>
                        </div>
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="paymentMethod">Payment Method</label><br/>
                            <input type="radio" name="paymentMethod"
id="credit" value="card" checked/>Credit/Debit Card <br/>
                            <input type="radio" name="paymentMethod"
id="mobile" value="bill"/>Add to Dialog Postpaid Bill
                        </div>
```

```
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="cardNo">Card Number</label>
                            <input type="text" id="cardNo" pattern="[0-
9]{16}" className="FormField__Input"
                                placeholder="3444 3232 4444 5454"
name="cardNo" required value={this.state.cardNo}
                                onChange={this.handleChange}/>
                        </div>
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="expireDate">Expiry Date</label>
                            <input type="text" id="expireDate"
className="FormField__Input" placeholder="12/20"
                                name="expireDate" required
value={this.state.expireDate}
                                onChange={this.handleChange}/>
                        </div>
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="cvc">CVC</label>
                            <input type="text" pattern="[0-9]{3}" id="cvc"
className="FormField__Input"
                                placeholder="123"
                                name="cvc" required value={this.state.cvc}
onChange={this.handleChange}/>
                        </div>
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="mobileNo">Dialog Mobile Number</label>
                            <input type="text" id="mobileNo" pattern="[0-
9]{10}" className="FormField__Input"
                                placeholder="0777 429 399" name="mobileNo"
required value={this.state.mobileNo}
                                onChange={this.handleChange}/>
                        </div>
                        <div className="FormField">
                            <label className="FormField__Label"
htmlFor="pin">Pin</label>
                            <input type="text" id="pin"
className="FormField__Input" placeholder="1234" name="pin"
                                required="" value={this.state.pin}
onChange={this.handleChange}/>
                        </div>
                        <button id="pin" className="FormField__Button1 mr-
1">Request Pin</button>
                        <br/>
                        <br/>
                        <div className="FormField">
                            <Link to="/sign-in">
                                <button id="buy" className="FormField__Button
mr-20">Proceed to Payment</button>
                            </Link></div>
                    </form>
                </div>
            </div>
```

```
        );
    }
}

export default Buy;
```

Footer.js

```
import React, {Component} from 'react';
import footer from "../images/footer.png";

class Footer extends Component{
    render(){
        return(
            <div className="footer">
                <img className="footerM" src={footer} style={{width:'100%'}}
alt={footer}/>
            </div>
        );
    }
}
export default Footer;
```

5.2.2 JavaScript Functions

Bundle.js

```
let tickets = [];

function countTickets(tickets) {
    let countedTickets = tickets.reduce((prev, cur) => {
        prev[cur] = (prev[cur] || 0) + 1;
        return prev;
    }, {});

    return countedTickets;
}

let headers = {};
function getHeaders() {
    let headers = {
        'Content-Type': 'application/json',
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Credentials': true,
        'Authentication': localStorage.getItem("authKey"),
        'ClientId': localStorage.getItem("uid")
    };

    return headers;
}

// for everything but seeing all food items, we need to have the
authentication key in the API call.
// but every time a page is reloaded, bundle.js will run replace the baseURL
to the one at line #1(which
// doesn't contain the authentication key).
// this is done to maintain the baseURL with authentication key included
given that the client has already,
// being given an authentication key by the server.
if (localStorage.getItem('authKey') !== undefined) {
    headers.Authentication = localStorage.getItem('authKey');
    headers.ClientId = localStorage.getItem('uid');
}

// re-directions.
$('#signup').click(function () {
    window.location.href = 'SignUp.js';
});
$('#signin').click(function () {
    window.location.href = 'SignIn.js';
});
$('#buy').click(function () {
    if (localStorage.getItem('authKey') !== undefined &&
localStorage.getItem('uid') !== undefined) {
        window.location.href = 'buy.html';
    }
    else {
        window.location.href = 'SignUp.js';
```

```
    }
});

function redirectToHome() {
    window.location.href = 'App.js';
}
// name, phoneNo, password, cpassword, nic, email
$('#signup').click(function () {
    let data = {
        uid: 0,
        name: $('#name').val(),
        phoneNo: $('#phoneNo').val(),
        password: $('#password').val(),
        cpassword: $('#cpassword').val(),
        nic: $('#nic').val(),
        email: $('#email').val(),
    };

    axios.post('/user', data, { headers: getHeaders() })
        .then(response => {
            console.log(response.status);
            if (response.status === 500) {
                alert("Please fill all the fields.");
            }
            window.location.href = response.data.redirect;
        })
        .catch(reject => {
        });
});

/*
 * Initial executions. These happen right away when a page is loaded.
 */
$(document).ready(function () {
    let uid = localStorage.getItem("uid");

    $('#search').hide();  // enable if an item is selected.

    // show login,logout and register buttons appropriately.
    if (uid == undefined) {
        $('#signin, #signup').show();
        $('#signin').hide();
    }
    else {
        $('#signin, #signup').hide();
        $('#signin').show();
    }

    $('#buy').hide(); // credit/debit card payment radio button is going to
be selected by default.

    $('#searcht').empty();
    showTickets();
    getTotalPay();
});
```

```
/*
 * Handling login.
 */
$('#signin').click(function () {
    logMeIn();
});
function logMeIn() {
    // send credentials to the session API.
    let data = {
        email: $('#email').val(),
        password: $('#password').val()
    };

    axios.post('/user/authenticate', data, { headers: getHeaders() })
        .then(response => {
            let responseBody = response.data;

            if (responseBody.success === true) {
                // axios anyways store the response body in data. And the
json reponse itself has a data attribute which contains the session.
                localStorage.setItem('authKey',
responseBody.data.authKeyOfUid);
                localStorage.setItem('uid', responseBody.data.uid);

                // update the headers.
                headers.Authentication = localStorage.getItem('authKey');
                headers.ClientId = localStorage.getItem('uid');

                // hide the login and reg button and replace them with a
logout button.
                $('#signin, #signup').hide();
                $('#signin').show();

                // redirection.
                window.location.href = responseBody.redirect;
            }
        })
        .catch(reject => {
            console.log(reject);
        })
}

$('#logout-btn').click(function () {
    logMeOut();
});
function logMeOut() {
    // tell the server to invoke the authentication key.
    axios.delete('/user/invoke', { headers: getHeaders() })
        .then(response =>{
            console.log(response.data);
        })
        .catch(reject => {
            console.log(reject);
        });

    // clear all local storage variables(uid, authKeyOfUid, fid)
```

```
    localStorage.clear();

    // go back to homepage.
    window.location.href = 'home.html';
}

// when the buy button assigned to a certain food item is clicked.
// we need to store this on local storage so that all the proceeding pages,
// know which item is being purchased.
function addToCart(tId) {
    tickets.push(tId);
    localStorage.setItem('tickets', JSON.stringify(countTickets(tickets)));

    // let the user know.
    alert("Item added to the cart");
    // show the payment button.
    $('#search').val('Pay for ' + tickets.length + ' item(s).').show();
}

// for the buying page to show the details.
function showFoodAndUserDetails() {
    // populate payment form with user's known details.
    axios.get('http://localhost:8081/user/id/' + localStorage.getItem('uid'),
{ headers: getHeaders() })
        .then(response => {
            let user = response.data;

            $('#email').val(user.email);
            $('#phoneNo').val(user.phoneNo);
            $('#name').val(user.name);
            $('#nic').val(user.nic);

            console.log(user);
        })
        .catch(reject => {

        });
}

function getTotalPay() {
    let data = (JSON).parse(localStorage.getItem('foodItems'));

    axios.post('/payment/total', data, { headers: getHeaders() })
        .then(response => {
            $('#amount').html(response.data.amount);
        })
        .catch(reject => {

        });
}

$('#pin').click(function () {
    alert('Your pin is: 1234');
    // we aren't sending the pin to a mobile phone so we just display it here
:(
});
```

```
function makePayment() {

    let paymentType = $("input[name='paymentRadios']:checked").val();

    // basic information.
    let data = {
        pid: 0,
        uid: localStorage.getItem('uid'),
        paymentType: paymentType,
        paymentDate: new Date(),
        itemsAndCounts: JSON.parse(localStorage.getItem('tickets'))
    };

    // payment information.
    switch (paymentType) {
        case 'card':
            let cardDetails = {
                number: $('#card-number').val(),
                cvc: $('#cvc').val(),
                expiry: $('#exp-date').val()
            };
            data.paymentDetails = cardDetails;
            break;

        case 'bill':
            let billDetails = {
                handler: 'Dialog',
                mobile: $('#dialog-number').val(),
                pin: $('#pin').val()
            };
            data.paymentDetails = billDetails;
            break;
    }

    console.log(paymentType);
    console.log(data);
    axios.post('/payment', data, { headers: getHeaders() })
        .then(response => {
            let data = response.data;
            if (data.success === true) {
                localStorage.setItem('pid', data.pid.toString());
                tickets = [];
                localStorage.setItem('tickets', '');  // we need to erase the
food items from local storage since the checkout has completed.
            }

            if (data.redirect === 'App.js') { alert('Success! Redirecting to
home.'); }
            window.location.href = data.redirect;
        })
        .catch(reject => {
            console.log(reject)
        });
}
```

```
// for home page to show all the food items.
function showTickets() {
    axios.get('/ticket', { headers: getHeaders() })
        .then(response=> {
            let entries = response.data;
            mapFoodResults(entries, 'ticket-list', true);
        })
        .catch(rejection => {

        });
}

// show relevant food items as the user is typing.
$('#ticket-search').keypress(function () {
    let keyword = $('#search').val();

    // remove the current contents of the list first.
    $('#ticket-list').empty();

    console.log(getHeaders());
    axios.get('/ticket/' + keyword, { headers: getHeaders() })
        .then(response=> {
            let entries = response.data;
            mapFoodResults(entries, 'tickets', true);
        })
        .catch(rejection => {

        });
});


/*
 * Handling payment.
 */
function completePayment() {
    // collect information from the forms.
    // common details.
    let data = {

    };
    let paymentType = $('[name=paymentRadios]:checked').val();
    console.log(paymentType);
}

// show appropriate payment information form depending on which payment type
is selected.
//      1) Credit/Debit Card radio -> cardPayment form
//      2) Add to Dialog Postpaid Bill radio -> billPayment form.
// always hide the irrelevant form.
$('#paymentRadiosCard').click(function () {
    $('#cardPayment').show();   $('#billPayment').hide();
});

$('#paymentRadiosBill').click(function () {
    $('#billPayment').show();   $('#cardPayment').hide();
});
```

```
// Food items retrieved by API call to /food will be mapped to the food-list
UL.
// This can be used for showing all the food items in the db or just to show
// the,
// search results.

// @param entries
// json array containing food items.
function mapFoodResults(entries, targetHtmlTag, appendBtn) {
    for (let i = 0; i < entries.length; i++) {
        // id of each list item element should be the food if of the food it,
        // contains.
        let entry = entries[i];
        console.log(i);
        let compositeHtmlElement =
            '<li class="list-group-item d-flex justify-content-between align-
items-center">' +
            '<p>'+
            '<b>' + entry.name + '<b/> <br /><small>' + entry.ingredients +
'</small> <br />' +
            '<span class="badge badge-primary badge-pill">Rs: ' + entry.price
+ '/=</span>' +
            '</p>';

        if (appendBtn) {
            compositeHtmlElement += '<button id="' + entry.fId + '"
type="button" class="btn btn-success"
onclick="addToCart(this.id)">Buy</button>';
            // the reason why we append an underscore to the entry.fId's
value is, since we give the same value as the id of the LI element,
            // buyThisFoodItem will get the LI element as a whole as the
parameter if we directly pass the fId is its own id as well.
        }

        compositeHtmlElement += '</li>';
        // when we call the onClick function, it will actually get the whole
        // <li> element as the parameter since,
        // li element has the fId as its id.

        $('#' + targetHtmlTag).append(compositeHtmlElement);
    }
}

// POST the reg details to the server.
// name, phoneNo, password, cpassword, nic, email
$('#signin').click(function () {

    let data = {
        "uid": 0,
        "name": $('#name').val(),
        "phoneNo": $('#phoneNo').val(),
        "password": $('#password').val(),
        "cpassword": $('#cpassword').val(),
        "nic": $('#nic').val(),
        "email": $('#email').val()
```

```
    };

    // make sure to include uid attribute with 0 as its value.
    // otherwise the server will complain about a missing parameter.
    axios.post('/user', data, headers)
        .then(response => {
            window.location.href = 'SignUp.js';
        })
        .catch(rejection => {
            // for some reason axios catch a rejection even when the,
            // server accepts the POST data.
            window.location.href = 'SignUp.js';
        });
});


/* * Validations * */
function checkForNumericOnly(str) {
    return !isNaN(parseFloat(str)) && isFinite(str);
}

function checkForAlphabeticOnly(str) {
    let pattern = /^[A-Za-z]+$/;
    return str.match(pattern);
}

function checkForAlphaNumericOnly(str) {
    let pattern = /^([a-zA-Z0-9 _-]+)$/;
    return str.match(pattern);
}

$('#name').keypress(function () {
    let value = $('#name').val();
    let elem = $('#name');

    if (value != null && checkForAlphabeticOnly(value)) {
        elem.addClass('border border-success').removeClass('border-danger');
    }
    else {
        elem.addClass('border border-danger').removeClass('border-success');
    }
});

$('#phoneNo').keypress(function () {
    let value = $('#phoneNo').val();
    let elem = $('#phoneNo');

    if (value != null && checkForNumericOnly(value)) {
        elem.addClass('border border-success').removeClass('border-danger');
    }
    else {
        elem.addClass('border border-danger').removeClass('border-success');
    }
});

$('#cardNo').keypress(function () {
```

```
    let value = $('#cardNo').val();
    let elem = $('#cardNo');

    if (value != null && checkForNumericOnly(value)) {
        elem.addClass('border border-success').removeClass('border-danger');
    }
    else {
        elem.addClass('border border-danger').removeClass('border-success');
    }
});

$('#cvc').keypress(function () {
    let value = $('#cvc').val();
    let elem = $('#cvc');

    if (value != null && checkForNumericOnly(value)) {
        elem.addClass('border border-success').removeClass('border-danger');
    }
    else {
        elem.addClass('border border-danger').removeClass('border-success');
    }
});

$('#mobileNo').keypress(function () {
    let value = $('#mobileNo').val();
    let elem = $('#mobileNo');

    if (value != null && checkForNumericOnly(value)) {
        elem.addClass('border border-success').removeClass('border-danger');
    }
    else {
        elem.addClass('border border-danger').removeClass('border-success');
    }
});

$('#pin').keypress(function () {
    let value = $('#pin').val();
    let elem = $('#pin');

    if (value != null && checkForNumericOnly(value)) {
        elem.addClass('border border-success').removeClass('border-danger');
    }
    else {
        elem.addClass('border border-danger').removeClass('border-success');
    }
});
```