

San Jose State University

Fall 2017



KAYAK Project Report

CMPE 273 – Team 7

SUBMITTED BY:

Aditya Parmar
Dishant Kimtani
Meenakshi Paryani
Siddharth Suthar
Sunil Tiwari

SUBMITTED TO:

Prof. Simon Shim

Project Contribution:

- **Aditya Parmar:**

I helped in creating the design of the project. I created the user interface of the entire Kayak project. I also integrated various modules such as search, booking, payment, user, admin etc. with the UI. I have created user account verification service using email. I have enhanced and managed user session. I have also maintained constant redux state throughout the project.

- **Dishant Kimtani:**

I created the MongoDB schema for kayak. We are storing car, flight and hotel information in MongoDB. Also, we have created separate databases for vendor information in MongoDB. I also created the backend API's and corresponding Kafka services for Searching the flights, hotels and car. I also created the backend API's for adding/deleting/listing the vendors. I have also created connection pooling for MongoDB.

- **Meenakshi Paryani:**

I created the MySQL schema for kayak. We are storing user, logging, user activity and booking information in MySQL. Also, I also created the backend API's and corresponding Kafka services for booking and payments of the flights, hotels and car booking. I also created the front-end components for booking of a flight/hotel and car and integrated with the backend. I also created provision for caching the SQL queries using Redis server. I have also created connection pooling for MySQL.

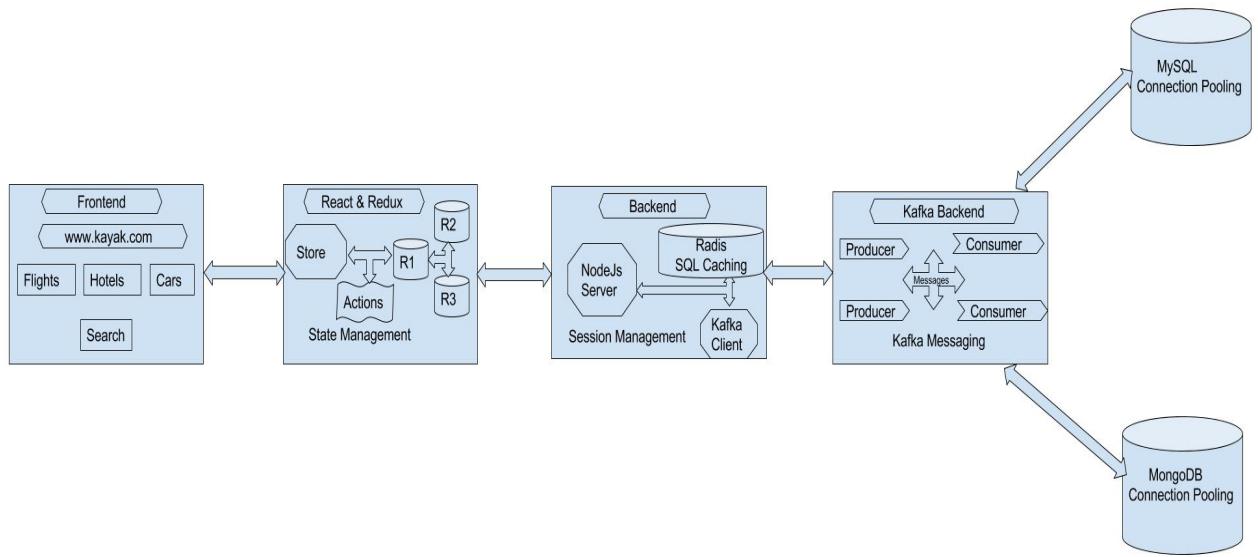
- **Siddharth Suthar:**

I created the backend API's and corresponding Kafka services for user and admin module. I also created user authentication service using PassportJS. Also, i created third party (google) authentication using PassportJS. I have also done various other validation such as password encryption and zip code validation.

- **Sunil Tiwari:**

I created the user analytics & state management part of the project. I create the initial state management for the project using the Redux. I also create a provision to track activity of the user such as button click, page navigation etc. I also analyzed the information stored in the database to draw meaningful analytics out it. I also created the dashboard for the admin module.

System Design:



Object Management Policy:

Overall architecture of the system consists of following components

- **Frontend:** The front end of the system is an interactive UI built in React to take the input from the user and respond to the user events and actions.
- **Zip Validation:** Zip is validated for each user while entering the address information.
- **React & Redux:**
 - **State management:** The state of the system is managed using the Redux (Store). Redux provides a predictable state managements architecture for web applications. Events dispatch actions and actions invoke reducers to modify the state of the system. The in redux state is stored as a Single object tree inside the store. There can be only one store in an application.
 - **React State is stored as follows:**
 - User Profile:{
 isLoggedIn,
 firstname,
 lastname,
 isadmin,
 email,
 address,
 zipcode,
 phonenumer,
 imgpath...}

- User History:{UserHistoryInformation...}
 - User Search:{UserSearchInformation...}
 - User Booking:{ UserBookInformation...}
 - Vendor:{UserVendorInformation...}
- **Backend:**
 - **Authentication:** The user authentication process is handled through the PassportJS. Passport is authentication middleware for Node.js.
 - **API management:** It receives calls to signUp, signIn, Search, Book, History, Profile and other APIs and generates appropriate messages for KAFKA producer.
 - **Encryption:** Encryption is done using the ‘crypto’ library available for the NodeJS.
 - **Automatic Mail:** We have created Automatic email service for the newly registered users using the Gmail’s pop3 service.
 - **Session management:** The session of the system is managed using the Redux provides a predictable state managements architecture for web applications. Events dispatch actions and actions invoke reducers to modify the state of the system. The redux state is stored as a Single object tree inside the store. There can be only one store in an application

“Heavyweight” Resources Management Policy:

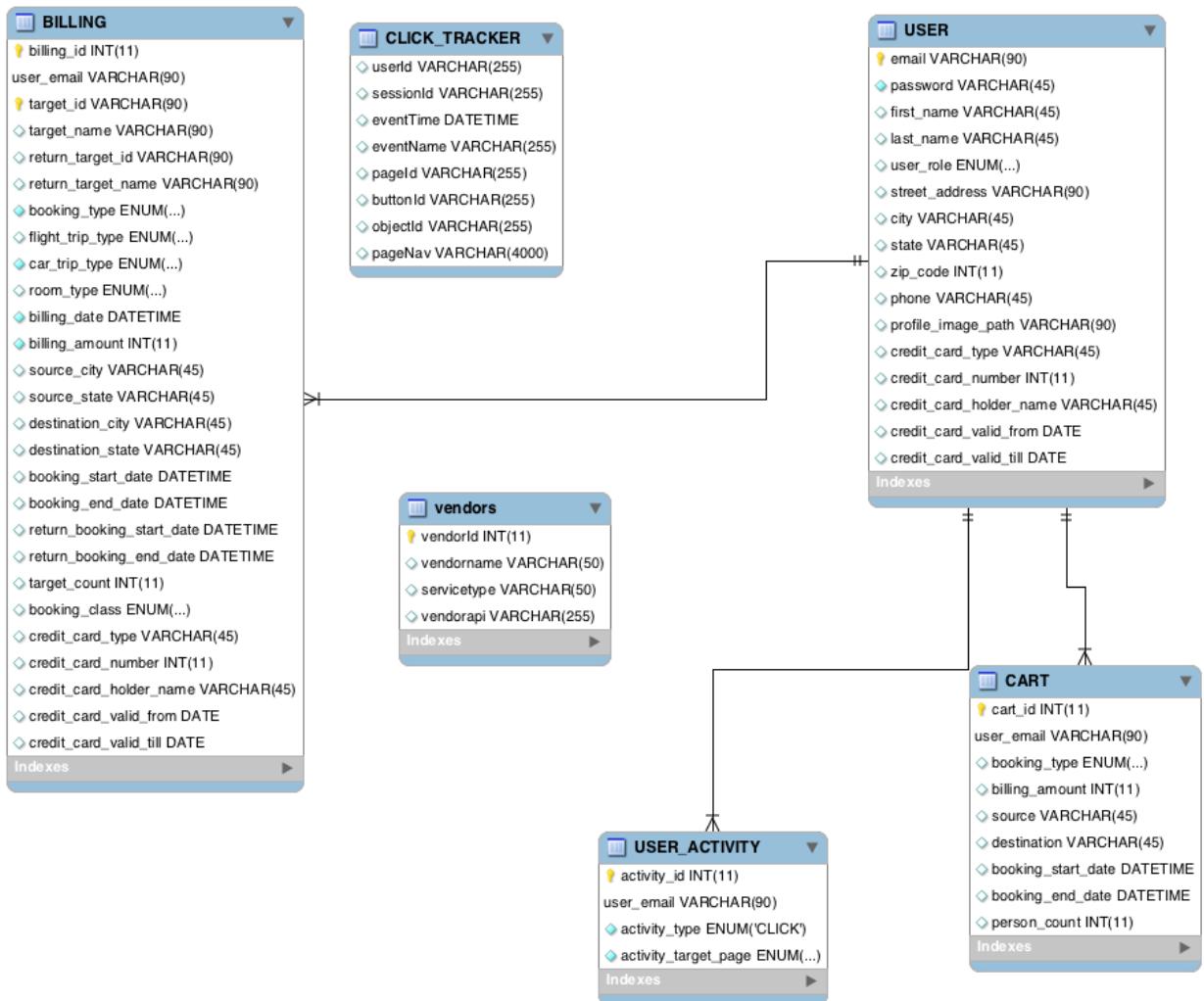
- **Kafka-Backend:**
 - **Messaging system:** Kafka is used as a messaging system. It allows you to create multiple consumers and multiple producers. Producers and consumers can subscribe to various topics in Kafka hence send and receive messages.
 - **Query building:** It consumes Kafka messages and build MongoDB queries. It also builds producers to provide data and acknowledgement from MongoDB.

Read Write Policy:

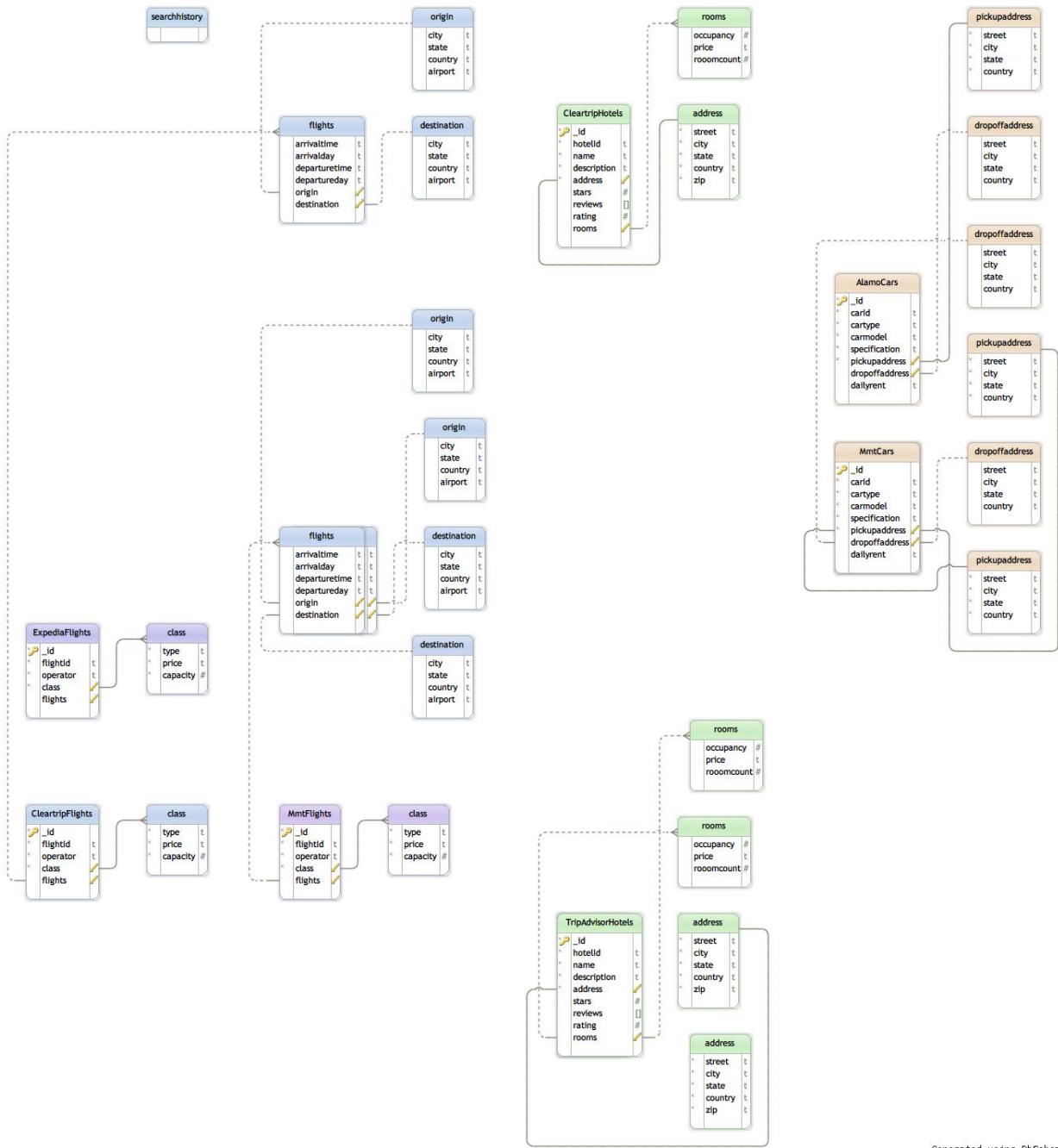
- **Database (MongoDB & MySQL):** The user, user activity, logging and booking information is stored, retrieved and manipulated in the MySQL database. The car, hotel, flights and vendor information is stored, retrieved and manipulated in the MongoDB database.
- **SQL Caching:** We have used Redis server based SQL caching in the project to cache the queries of the MongoDB database.

- **Connection Pooling:** We have used the built-in connection pooling mechanism to cache the result of the queries in both MongoDB and MySQL. The TTL for caching strategy is set to 60 seconds.

The schema of the MySQL database is as follows:



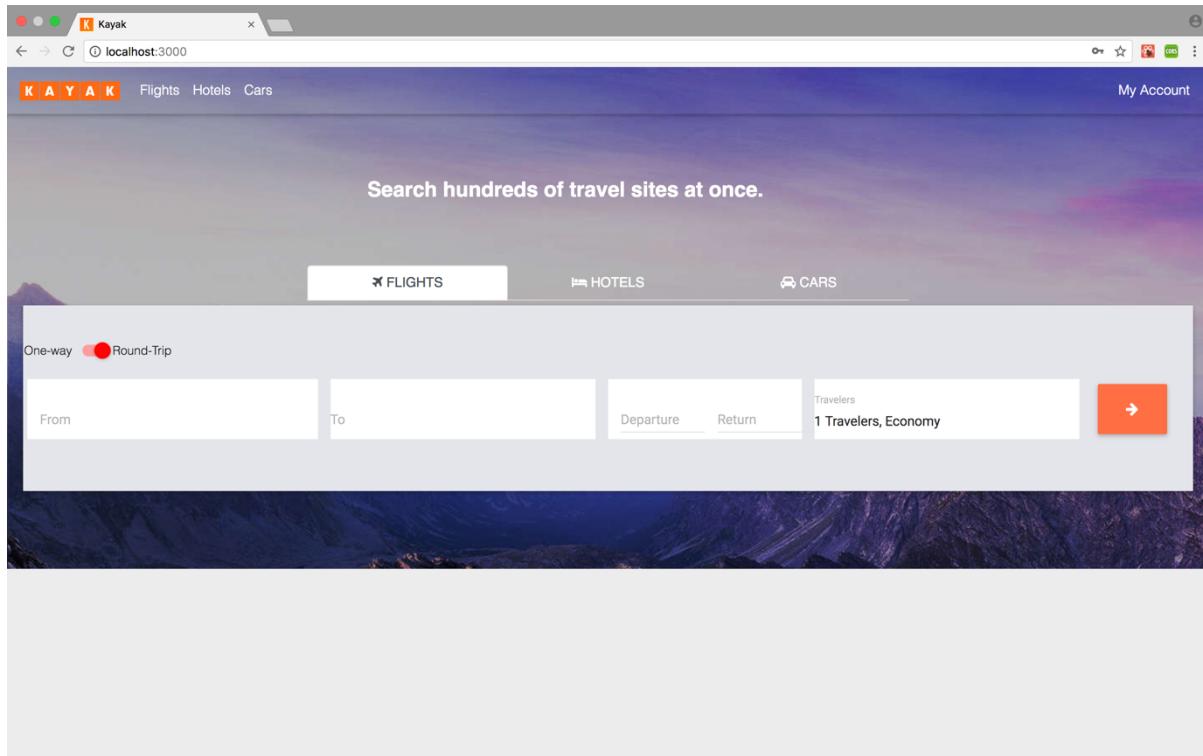
The schema of the MongoDB database is as follows:



Generated using DbSchema

Screenshots:

Home Page (UI):



Home Page (Node Backend):

A screenshot of an IDE (IntelliJ IDEA) showing the project structure and code editor. The project structure on the left includes "react-front-end", "kafka-back-end", and "kafka-front-end". The code editor shows "App.js" with the following content:

```
1 import React, {Component} from 'react';
2 import './App.css';
3 import Home from './components/home';
4 import MuiThemeProvider from 'material-ui/styles/MuiThemeProvider';
5 import getMuiTheme from 'material-ui/styles/getMuiTheme'
6
7 const muiTheme = getMuiTheme(
8   );
9
10 class App extends Component {
11   render() {
12     return (
13       <div className="App">
14         <MuiThemeProvider muiTheme={muiTheme}>
15           <Home/>
16         </MuiThemeProvider>
17       </div>
18     );
19   }
20 }
21
22 export default App;
```

The terminal window at the bottom shows the Kafka back-end starting up:

```
sunil28s-MacBook-Pro:kafka-back-end sunil28$ npm start
> kafka-back-end@1.0.0 start /Users/sunil28/Desktop/KayakFinal/Kayak/kafka-back-end
> node server.js

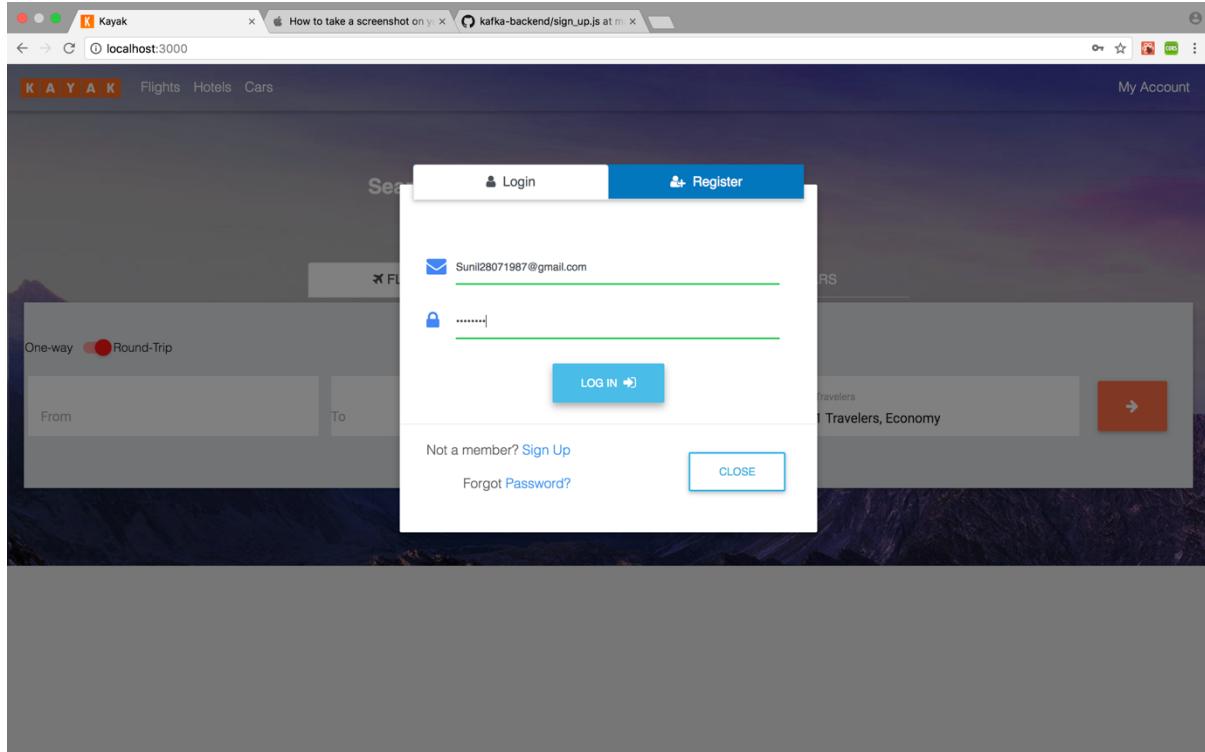
producer ready
server is running
undefined
client ready!
```

Home Page (Kafka Backend):

The screenshot shows a Java IDE interface with the following details:

- Project Explorer:** Shows the project structure under "Kayak". The "react-front-end" module is expanded, showing its sub-directories like .idea, node_modules, public, and src.
- Code Editor:** Displays the content of the App.js file. The code defines a class App extending Component, which renders a div with the class name "App". Inside the div, there is a MuiThemeProvider component with the prop muiTheme set to getMuiTheme(). The code also imports React, Component, App.css, and Home from their respective files.
- Terminal:** Shows the command-line output of running the application. It includes the command "npm start", the nodemon process starting, and the message "producer ready Listening on port 3001".
- Bottom Status Bar:** Includes icons for Version Control, Terminal, TODO, Dockerfile detection, and Event Log.

Signup Page (UI):



Signup Page (Node Backend):

The screenshot shows the IntelliJ IDEA IDE with the project structure of the Kayak application. The current file is `server.js`, specifically the `loginButton()` function under the `telephoneCheck(str)` block. The code handles user input validation and sends a POST request to the API to log in. The response is checked for a status of 201, indicating success, and the user profile is updated. The terminal below shows the execution of the application and a successful login attempt.

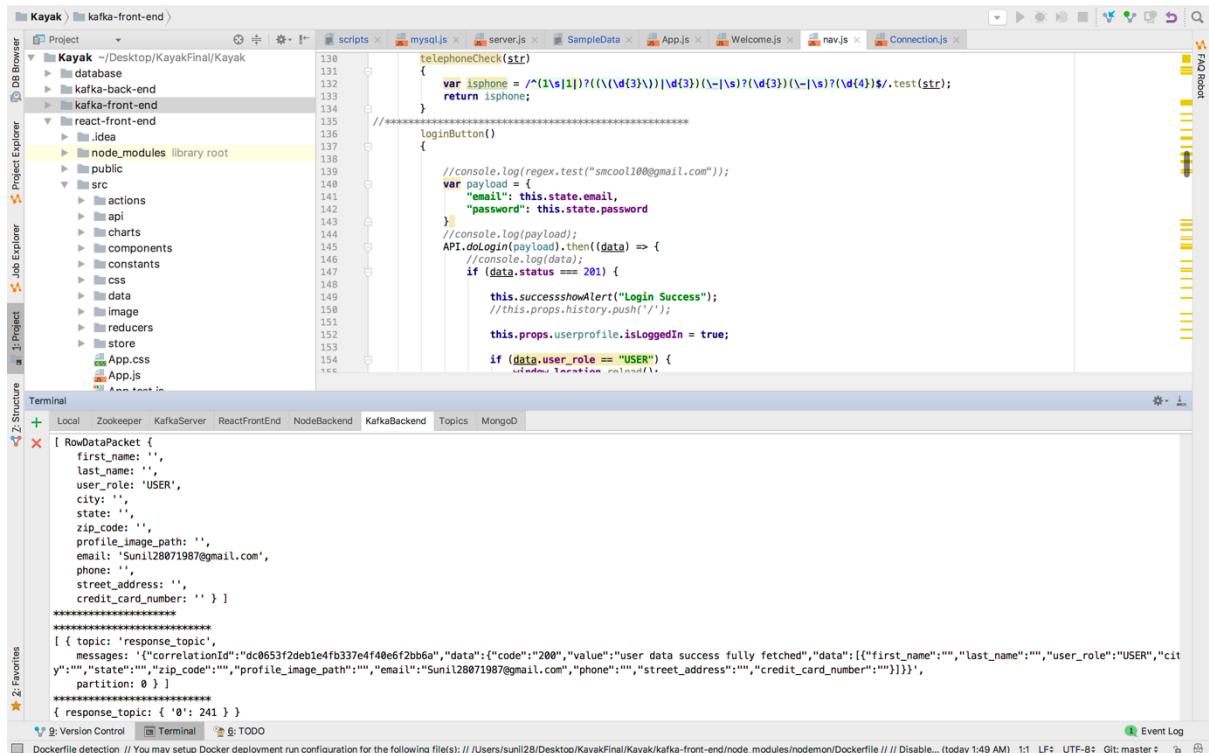
```
telephoneCheck(str)
{
    var isPhone = /^(\\d{1}|\\d{2})|(\\d{3})|\\d{3}(\\-|\\.)?\\d{3}(\\-|\\.)?\\d{4}$/.test(str);
}

//*****loginButton()
{
    //console.log(regex.test("smcool100@gmail.com"));
    var payload = {
        "email": this.state.email,
        "password": this.state.password
    }
    //console.log(payload);
    API.doLogin(payload).then(data => {
        //console.log(data);
        if (data.status === 201) {
            this.successShowAlert("Login Success");
            //this.props.history.push('/');
            this.props.userprofile.isLoggedIn = true;
            if (data.user_role === "USER") {
                window.location.reload();
            }
        }
    })
}
```

```
{ code: '200',
  value: 'user data success fully fetched',
  data:
   { first_name: '',
     last_name: '',
     user_role: 'USER',
     city: '',
     state: '',
     zip_code: '',
     profile_image_path: '',
     email: 'Sunil28071987@gmail.com',
     phone: '',
     street_address: '',
     credit_card_number: '' } } }
```

```
GET /user/checkSession 200 6.646 ms - 287
Session destroyed
POST /user/logout 201 1.475 ms - 42
inside the login path with the body{object Object}
```

Signup Page (Kafka Backend):



```
telephoneCheck(str)
{
    var isphone = /^(1\s\d{1})?((\(\d{3}\))|\d{3})(-\|\s)?(\d{3})(-\|\s)?(\d{4})$/.test(str);
    return isphone;
}

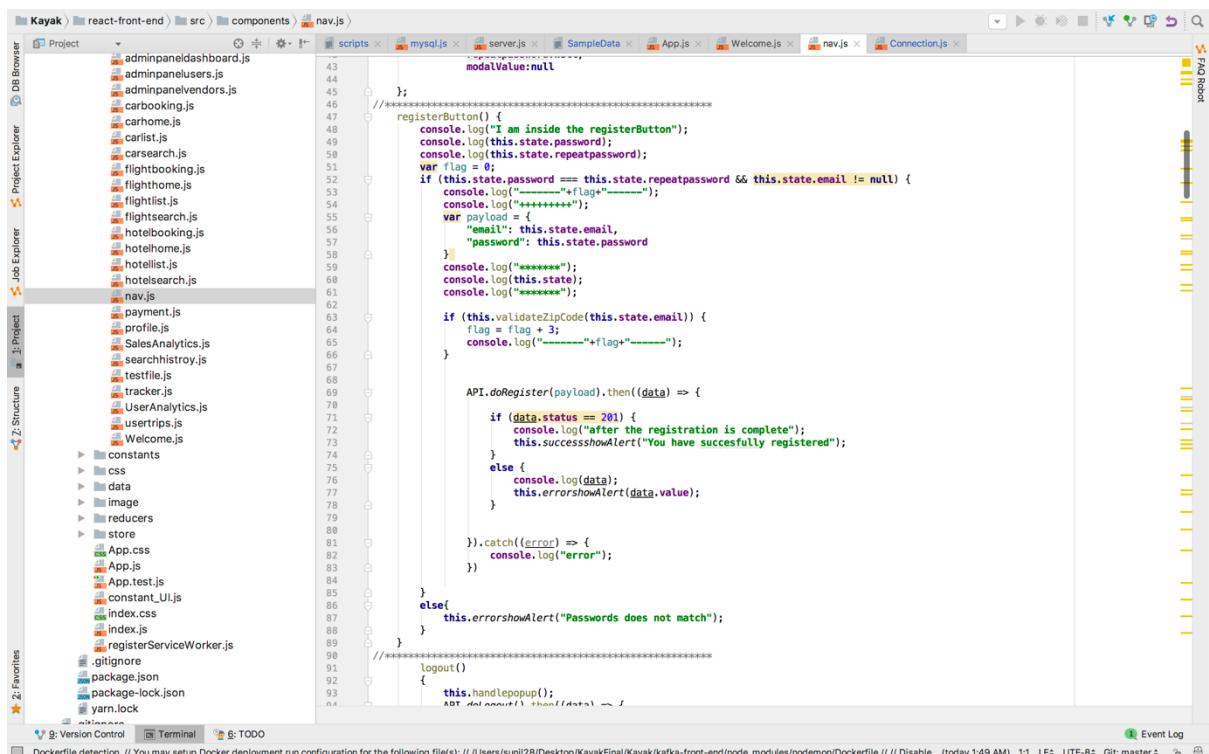
loginButton()
{
    //console.log(regex.test("smcool00@gmail.com"));
    var payload = {
        "email": this.state.email,
        "password": this.state.password
    }
    //console.log(payload);
    API.doLogin(payload).then(data => {
        //console.log(data);
        if (data.status === 201) {
            this.successshowAlert("Login Success");
            //this.props.history.push('/');
            this.props.userprofile.isLoggedIn = true;
            if (data.user_role === "USER") {
                window.location.reload();
            }
        }
    })
}

RowDataPacket {
    first_name: '',
    last_name: '',
    user_role: 'USER',
    city: '',
    state: '',
    zip_code: '',
    profile_image_path: '',
    email: 'Sunil28071987@gmail.com',
    phone: '',
    street_address: '',
    credit_card_number: '' } ]
*****  

[ { topic: 'response_topic',
  messages: '{ "correlationId": "dc0653f2debe4fb337e4f40e6f2bb6a", "data": {"code": "200", "value": "user data success fully fetched", "data": [{"first_name": "", "last_name": "", "user_role": "USER", "city": "", "state": "", "zip_code": "", "profile_image_path": "", "email": "Sunil28071987@gmail.com", "phone": "", "street_address": "", "credit_card_number": ""}] }',
  partition: 0 } ]
*****  

{ response_topic: { '0': 241 } }
```

Signup (Business Logic):



```
modalValue:null
};

registerButton() {
    console.log("I am inside the registerButton");
    console.log(this.state.password);
    console.log(this.state.repeatpassword);
    var flag = 0;
    if (this.state.password === this.state.repeatpassword && this.state.email != null) {
        console.log("-----flag-----");
        console.log("*****");
        var payload = {
            "email": this.state.email,
            "password": this.state.password
        }
        console.log("*****");
        console.log(this.state);
        console.log("*****");

        if (this.validateZipCode(this.state.email)) {
            flag = flag + 3;
            console.log("-----"+flag+"-----");
        }

        API.doRegister(payload).then(data => {
            if (data.status == 201) {
                console.log("after the registration is complete");
                this.successshowAlert("You have successfully registered");
            } else {
                console.log(data);
                this.errorshowAlert(data.value);
            }
        }).catch(error => {
            console.log("error");
        })
    } else {
        this.errorshowAlert("Passwords does not match");
    }
}

logout()
{
    this.handlepopup();
    ADT.logout().then(data => {
        this.props.history.push('/');

        if (data === "Logout successful") {
            this.props.history.push('/');
        }
    })
}
```

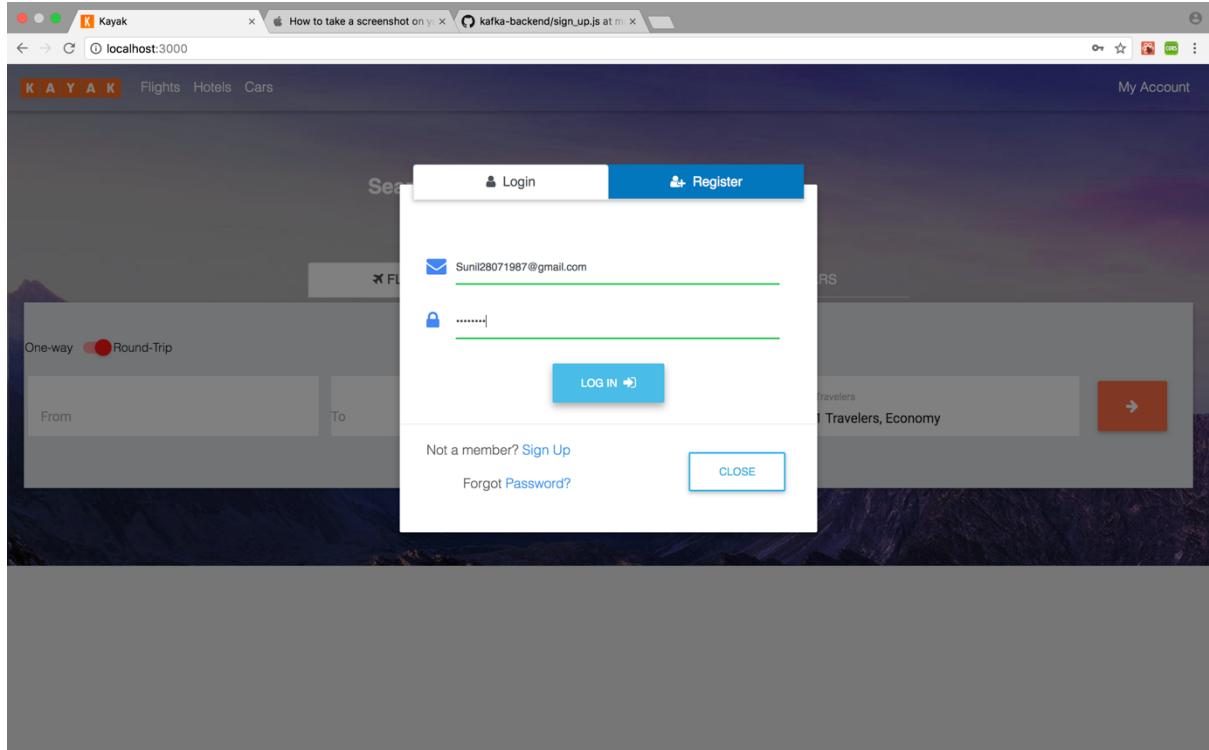
Signup Page (UI):

Signup Page (Node Backend):

Signup Page (Kafka Backend):

Signup (Business Logic):

Sign in Page (UI):



Sign in Page (Node Backend):

A screenshot of an IDE (DB Browser for MySQL) showing the file structure of a project named "kafka-front-end". The "src" directory contains "actions", "api", "charts", "components", "constants", "css", "data", "image", "reducers", and "store". Inside "store", there is an "App.css" file and an "App.js" file. The "App.js" file is open in the editor, showing a snippet of JavaScript code for a login function. The code includes regex patterns for email and phone numbers, a payload object with "email" and "password" keys, and an API call to "doLogin". It also handles success and failure responses, setting a success alert and updating the user profile state. In the bottom left, a terminal window shows the command "node App.js" being run and its output, which includes a log message about a successful login and a status code of 201. The terminal also shows a Dockerfile detection message.

Sign in Page (Kafka Backend):

The screenshot shows a code editor interface with the following details:

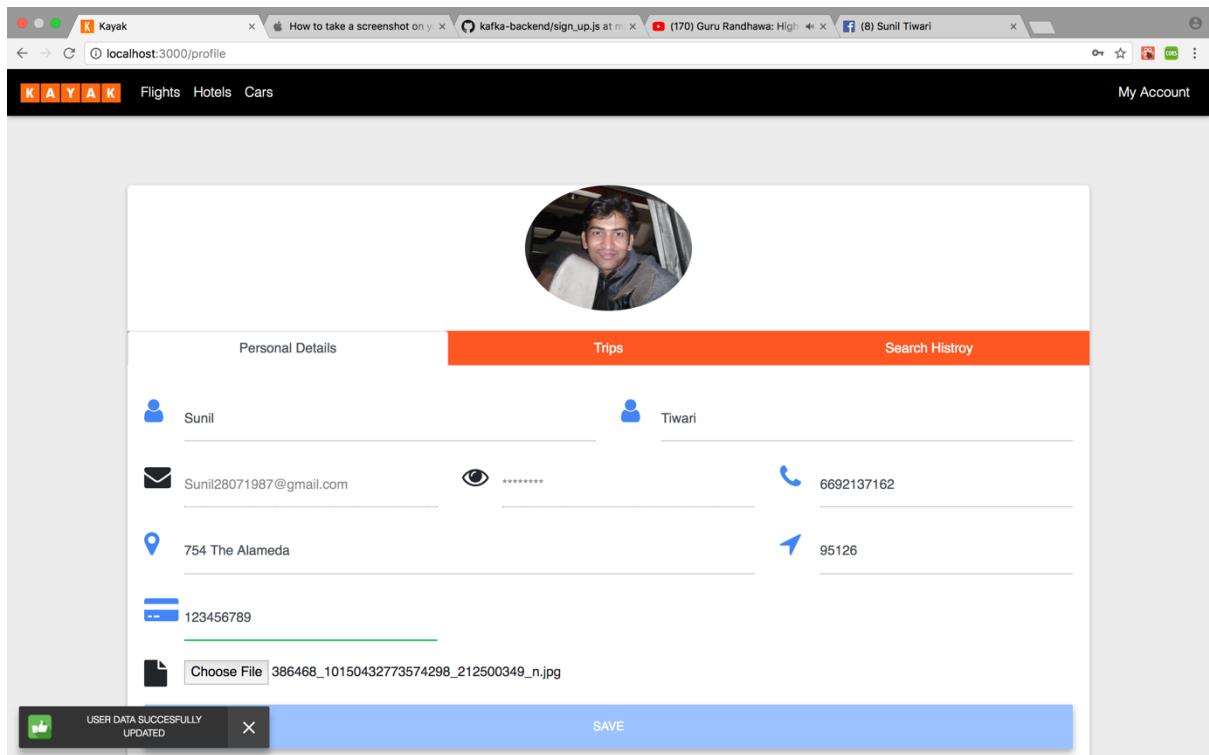
- Project Structure:** The project is named "Kayak" and contains a "kafka-front-end" folder. Inside "kafka-front-end", there are "react-front-end" and "node_modules" (library root).
- Code Editor:** The main pane displays a file named "Connection.js". The code includes a "telephoneCheck" function and a "loginButton" function. It uses regular expressions to validate phone numbers and emails, and interacts with an API via "API.doLogin(payload)".
- Terminal:** Below the code editor, a terminal window shows a log message from Kafka. It includes a correlation ID ("correlationId"), a reply-to topic ("response_topic"), and a message payload containing an email address ("Sunil28071987@gmail.com").
- Bottom Status Bar:** Shows Dockerfile detection and other system information like date, time, and file encoding.

Sign in (Business Logic):

The screenshot shows a code editor interface with the following details:

- Project Structure:** The project is named "react-front-end" and contains a "src" folder. Inside "src", there are various component files like "adminpaneldashboard.js", "adminpanelusers.js", "carbooking.js", etc., and a "components" folder which contains "nav.js".
- Code Editor:** The main pane displays a file named "nav.js". The code includes a "telephoneCheck" function and a "loginButton" function. It uses regular expressions to validate phone numbers and emails, and interacts with an API via "API.doLogin(payload)".
- Bottom Status Bar:** Shows Dockerfile detection and other system information like date, time, and file encoding.

User Profile Page (UI):



User Profile Page (Node Backend):

```
183     }
184     if (this.state.phonenumber != "") {
185       if (!this.telephoneCheck(this.state.phonenumber)) {
186         flag = flag + 5;
187         this.errorshowAlert("Phone number not valid");
188       }
189
190       if (flag == 0) {
191         console.log("About to push the data");
192         console.log(this.state.lmpath);
193         API.doupdate(payload).then((data) => {
194           if (data.status == 201) {
195             console.log("Successfull push");
196
197             this.successshowAlert("User data succesfully updated");
198           } else {
199             this.errorshowAlert("Error while updating the user info ");
200           }
201         })
202       }
203     }
204   }
205 }
```

The screenshot shows the Eclipse IDE interface with the Node.js code for updating user profile data. The code uses the Axios library to make a POST request to the API endpoint 'doupdate' with the payload containing user information. It logs the user's state and the local machine path (lmpath). It then checks the response status. If successful (status 201), it shows a success alert; otherwise, it shows an error alert. The code also handles errors during the update process.

```
*****  
{ email: 'Sunil28071987@gmail.com',  
  firstname: 'Sunil',  
  lastname: 'Tiwari',  
  address: '754 The Alameda',  
  zipcode: '95126',  
  phonenumber: '6692137162',  
  imgpath: 'Sunil28071987@gmail.com386468_10150432773574298_212500349_n.jpg',  
  creditcard: '123456789' }  
*****  
Sunil28071987@gmail.com  
in make request from the queueupdate  
...in kafkarpc....  
in response  
in response1  
true  
in response2  
{ update: { '0': 3 } }  
msg received  
response { code: '200', value: 'User successfully registered' }  
Dockerfile detection // You may setup Docker deployment run configuration for the following file(s): // /Users/sunil28/Desktop/KayakFinal/Kayak/kafka-front-end/node_modules/nodemon/Dockerfile // // Disable... (today 1:49 AM) 1:1 LF: UTF-8 Git: master : Event Log
```

User Profile Page (Kafka Backend):

```
if (this.state.phonenumber != "") {
  if (!this.telephoneCheck(this.state.phonenumber)) {
    flag = flag + 5;
    this.errorshowAlert("Phone number not valid");
  }
}

if (flag == 0) {
  console.log("About to push the data");
  console.log(this.state.lmpath);
  API.doUpdate(payload).then((data) => {
    if (data.status == 201) {
      console.log("Successful push");

      this.successshowAlert("User data successfully updated");
    } else {
      this.errorshowAlert("Error while updating the user info ");
    }
  });
}
}

SQL Query::update billing set user_email='Sunil28071987@gmail.com'where user_email='Sunil28071987@gmail.com';

Connection closed...
executed the set billing key query
*****+
SQL Query::update user set email='Sunil28071987@gmail.com', first_name='Sunil', last_name='Tiwari',street_address='754 The Alameda',phone='6692137162',profile_image_path='Sunil28071987@gmail.cc
m386468_10150432773574298_212500349_n.jpg',credit_card_number='123456789',zip_code='95126' where email='Sunil28071987@gmail.com';

Connection closed.
*****+
[ { topic: 'response_topic',
  messages: [ { 'correlationId': '9f413a950b7c645e0395dce23f4cf629', "data": { "code": "200", "value": "User successfully registered" } },
  partition: 0 } ]
*****+
{ response_topic: { '0': 251 } }
```

User Profile (Business Logic):

```
  this.props.errorshowAlert("Email does not match");
}

logout()
{
  this.handlepopup();
  API.doLogout().then((data) => {
    if (data.status == 201) {
      console.log("Logout successful");
      this.successshowAlert("Logout successful");
      this.props.userprofile.isLoggedIn = false;
      this.props.history.push('/');
    }
  });
}

//*****+
validateZipCode(elementValue)
{
  var zipCodePattern;
  if (elementValue.indexOf("-") > -1) {
    zipCodePattern = /^\d{5}?\d{4}-\d{4}$/;
  } else {
    zipCodePattern = /^\d{5}\$/;
  }

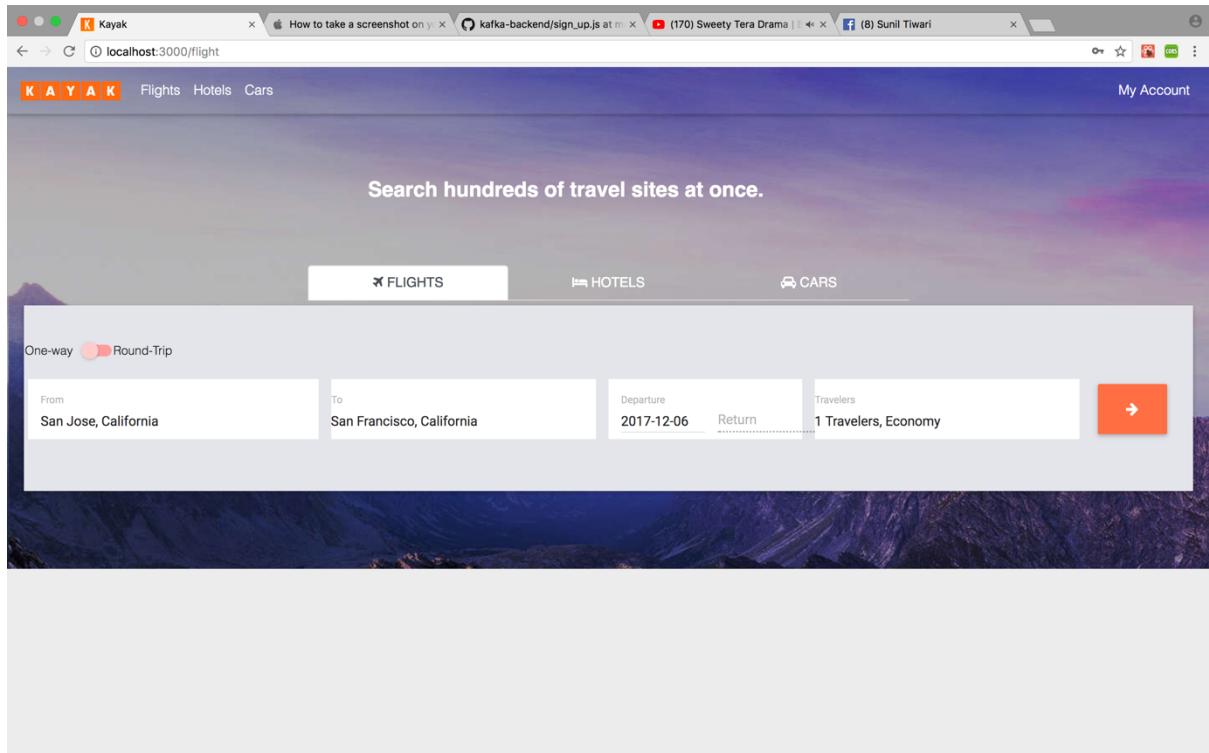
  //console.log("Zip Validation : ",zipCodePattern.test(elementValue));
  return zipCodePattern.test(elementValue);
}

//*****+
validateEmail(mail)
{
  if (^w+([.-]?)?w+@[.\w-]?\w+*(.\w{2,3})+$/.test(mail)) {
    console.log("true");
    return (true)
  }
  console.log("false");
  return (false)
}

//*****+
telephoneCheck(str)
{
  var isphone = /(^\d{1}|\d{1})?((\d{3}\d{3})|\d{3})(-\|\s)?(\d{3})(-\|\s)?(\d{4})$/.test(str);
  return isphone;
}

//*****+
loginButton()
{
```

Flight Search Page (UI):



A screenshot of a web browser showing the Kayak flight list page. The URL in the address bar is `localhost:3000/flightlist`. The page has a header with the Kayak logo and navigation links. On the left, there are three filter sections: "Price" (with a slider from Low:\$150 to High:\$500), "Departure Time" (with a slider from 00:00 to 23:59), and "Arrival Time" (with a slider from 00:00 to 23:59). On the right, there is a list of flight options. Each option includes the airline logo, departure time, arrival time, flight number, destination, and price. There is also a "BOOK" button for each flight.

Airline	Departure Time	Arrival Time	Flight Number	Destination	Price
Air China Airlines	10:00	06:00	MMT101	San Francisco	\$150
Deccan Airlines	09:00	15:00	CT100	San Francisco	\$500
Air India	18:00	02:00	ED101	San Francisco	\$300

Kayak Flights Hotels Cars My Account

Price

Low:\$150 High:\$500

Departure Time

00:00 23:59

Arrival Time

00:00 23:59

Air China Airlines 10:00 06:00 non stop San Jose San Francisco MMT101 \$150 Economy BOOK

Origin Day: Tue Destination Day: Wed Time: 06:00 Airport: San Jose Airport City: San Jose State: California

Deccan Airlines 09:00 15:00 non stop San Jose San Francisco CT100 \$500 Economy BOOK

Kayak Flights Hotels Cars My Account

Price

Low:\$150 High:\$330

Departure Time

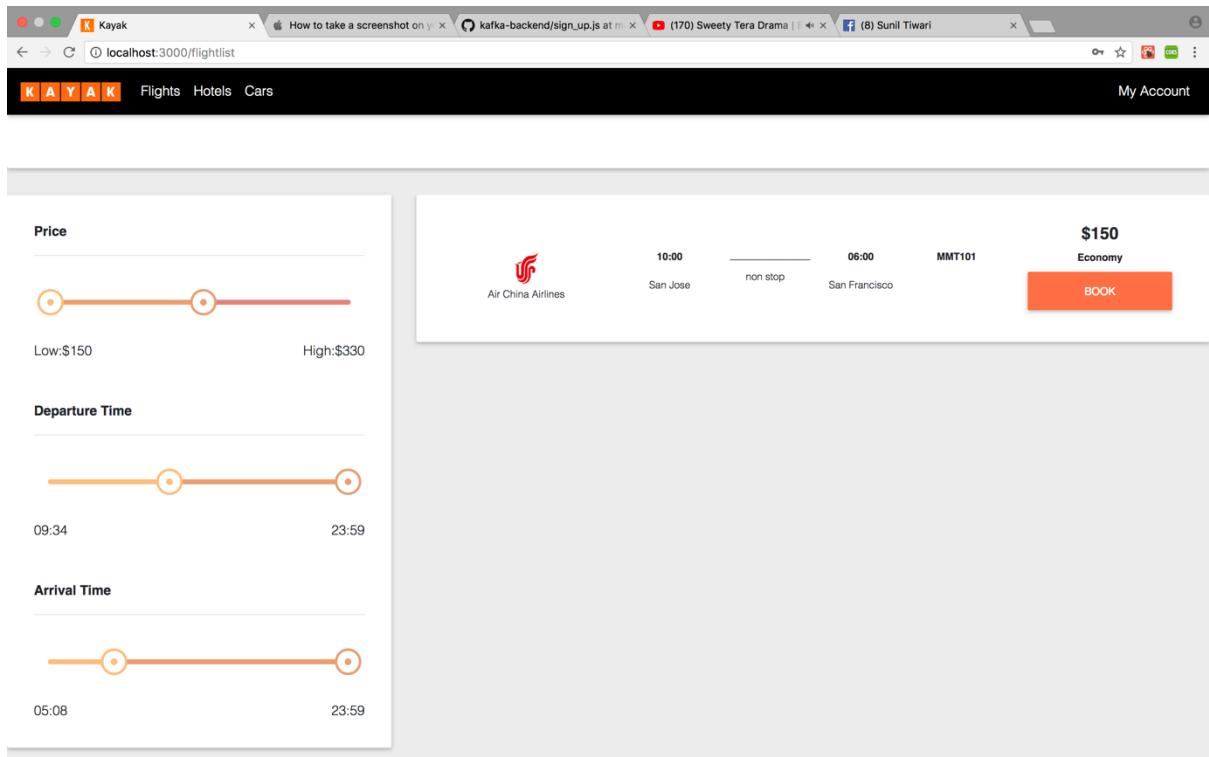
00:00 23:59

Arrival Time

00:00 23:59

Air China Airlines 10:00 06:00 non stop San Jose San Francisco MMT101 \$150 Economy BOOK

Air India 18:00 02:00 non stop San Jose San Francisco ED101 \$300 Economy BOOK



Flight Search (Node Backend):

```

const payload = {
  'originCity': this.state.from.split(" ")[0].trim(),
  'originState': this.state.from.split(" ")[1].trim(),
  'destinationCity': this.state.to.split(" ")[0].trim(),
  'destinationState': this.state.to.split(" ")[1].trim(),
  'departureDay': moment(this.state.startDate).toString().split(" ")[0],
  'arrivalDay': moment(this.state.endDate).toString().split(" ")[0],
  'flightClass': this.state.class,
  'startdate': startdate,
  'enddate': enddate,
  'passenger': this.state.traveler,
  'searchType': 'flight'
};

API.checkSession().then((data) => {
  console.log("inside the check session response");
  // ...
});
    
```

The screenshot shows an IDE (DB Browser for SQLite) with a Node.js project structure. The `flightsearch.js` file is open, containing code for generating a flight search payload based on user input. The payload includes fields like originCity, originState, destinationCity, destinationState, departureDay, arrivalDay, flightClass, startdate, enddate, passenger, and searchType. Below the code, a terminal window shows log output from a Kafka RPC call, indicating a successful response with a flight ID of 200 and a duration of 111.589 ms.

Flight Search (Kafka Backend):

```
const payload={  
    'originCity':this.state.from.split(",")[0].trim(),  
    'originState':this.state.from.split(",")[1].trim(),  
    'destinationCity':this.state.to.split(",")[0].trim(),  
    'destinationState':this.state.to.split(",")[1].trim(),  
    'departDate': moment(this.state.startDate).format("MM/DD/YYYY"),  
    'tripType':this.state.returnDateenable==true?"Two-Way":"One-Way",  
    'arriveDay':moment(this.state.endDate).toString().split(" ")[0],  
    'flightClass':this.state.class, //  
    'startdate': startdate,  
    'enddate': enddate,  
    'passenger': this.state.traveler,  
    'searchtype':'flight'  
}  
  
API.checkSession().then((data)=>{  
    console.log("inside the check session response");  
});
```

Flight List: [{ _id: 5a252e9c0cbbedeb45942264,
 flightId: "ED101",
 operator: 'Air India',
 imageUrl: 'airindia.jpg',
 class: [Object], [Object], [Object]],
flights:
 { arrivaltime: '02:00',
 arrivalday: 'Tue',
 departuretime: '18:00',
 departureday: 'Wed',
 origin: [Object],
 destination: [Object] } }]

[{ topic: 'response_topic',
 messages: [{ 'correlationId': '8381ff9ab8e0c709080469b8e101b45cf', "data": { "code": "200", "value": [{ "_id": "5a252e9c0cbbedeb45942264", "flightId": "ED101", "operator": "Air India", "imageUrl": "airindia.jpg", "class": [{ "type": "Economy", "price": 300, "capacity": 50 }, { "type": "First", "price": 400, "capacity": 50 }, { "type": "Business", "price": 500, "capacity": 10 }], "flights": [{ "arrivaltime": "02:00", "arrivalday": "Tue", "departuretime": "18:00", "departureday": "Wed", "origin": { "city": "San Jose", "state": "California", "country": "USA", "airport": "San Francisco International Airport" } }] }] }]

{ response_topic: { '0': 260 } }

Flight Search (Business Logic):

```
handleUpdateFromInput = (value, textbox) => {  
    this.setState({from:value})  
    var fromsuggestion = cities().map((item,i)>item.city+" "+item.state)  
    this.setState({  
        fromsuggestion: fromsuggestion  
    });  
};  
handleUpdateToInput = (value, textbox) => {  
    this.setState({to:value})  
    var tosuggestion = cities().map((item,i)>item.city+" "+item.state)  
    this.setState({  
        tosuggestion: tosuggestion  
    });  
};  
handleStartDate(event, date){  
    this.setState({startdate: date})  
}  
handleEndDate(event, date){  
    this.setState({enddate: date})  
}  
handlepopup(){  
    this.setState({travelerpopup:!this.state.travelerpopup})  
}  
changeTravelerUp()  
{  
    const state = this.state;  
    if(gpu=="v") {  
        this.setState({traveler:this.state.traveler+1})  
    } else {  
        if(this.state.traveler != 1){  
            this.setState({traveler:this.state.traveler-1})  
        }  
    }  
}  
changeClass(g){  
    this.setState({class:g})  
}  
displaypopup(){  
    if(this.state.travelerpopup){
```

```
displaypopup(){
    if(this.state.travelerpopup){
        return <div style={{marginTop:"6%",minWidth:"300px",marginLeft:"60%",marginRight:"10%",borderRadius:"0",zIndex:"2"}} className="card">
            <div className="card-body">
                <button type="button" className="close" aria-label="Close" onClick={()>this.handlepopup()}></span>
            </button>
            <h6>艸Cabin</h6>
            <button type="button" className="btn btn-outline-primary waves-effect" onClick={()>this.changeclass("Economy")}>Economy</button>
            <button type="button" className="btn btn-outline-primary waves-effect" onClick={()>this.changeclass("Business")}>Business</button>
            <button type="button" className="btn btn-outline-primary waves-effect" onClick={()>this.changeclass("Premium")}>Premium</button>
            <button type="button" className="btn btn-outline-primary waves-effect" onClick={()>this.changeclass("First")}>First</button>
            <hr/>
            <h6>艸Travelers</h6>
            <div className="row">
                <div className="col-sm-4">
                    <button type="button" onClick={()>this.changetraveler("-")} className="btn btn-outline-primary waves-effect"><minus;></button>
                </div>
                <div className="col-sm-4" style={{textAlign:"center",marginTop:"10px}}>(this.state.traveler)</p>
                <div className="col-sm-4">
                    <button type="button" onClick={()>this.changetraveler("+")} className="btn btn-outline-primary waves-effect"><plus;></button>
                </div>
            </div>
        </div>
    }
}
styles = {
    toggle: {
        marginBottom: 16,
    },
    thumbOff: {
        backgroundColor: '#ffcccc',
    },
    trackOff: {
        backgroundColor: '#ff9d9d',
    },
    thumbSwitched: {
        backgroundColor: 'red',
    },
    trackSwitched: {
        backgroundColor: '#ff9d9d',
    },
},
```

The screenshot shows the Visual Studio Code interface with the following details:

- Project Explorer:** Shows the project structure under "Kakay".
- Editor:** Displays the content of the `flight.js` file.
- Bottom Status Bar:** Shows "g: Version Control", "Terminal", "g: TODO", and "Event Log".

```
1 var express = require('express');
2 var kafka = require('./kafka/client');
3 var search = require('./search');
4 var router = express.Router();
5
6 /* GET flight list */
7 router.get('/flights', function (req, res) {
8
9     var data = { 'searchtype': 'flight', 'searchquery': req.query };
10
11     search.searchFromApi(data, function (err, results) {
12
13         if(err || !results){
14             res.send({ 'status': 401 });
15         }
16         res.send({ 'flights': results, 'status': 201 });
17     });
18 });
19
20 router.get('/getflights', function (req, res) {
21
22     var searchcriteria=JSON.parse(req.query.data);
23
24     var vendor = req.query.vendor;
25
26     var data={ 'vendor': vendor, 'searchcriteria': searchcriteria};
27
28     kafka.make_request(vendor,data, function(err,results){
29
30         if(err){
31             res.send({ 'status': 401 });
32         }
33         else
34         {
35             if(results.code == "200"){
36                 res.send({ 'status': 200 , 'api_results' : results.value});
37             }
38             else {
39                 res.send({ 'status': 401 });
40             }
41         }
42     });
43 });
44
45 router.post('/addflight', function (req, res) {
46     console.log('Request is _____');
47     console.log(req.body);
48
49     var reqObject = {
50         email : req.session.email,
51         flight : req.body
52     };
53
54 });
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
235
236
237
237
238
239
239
240
241
242
243
244
245
245
246
247
247
248
249
249
250
251
252
253
254
255
255
256
257
257
258
259
259
260
261
262
263
264
265
265
266
267
267
268
269
269
270
271
272
273
274
275
275
276
277
277
278
279
279
280
281
282
283
284
285
285
286
287
287
288
289
289
290
291
292
293
294
295
295
296
297
297
298
299
299
300
301
302
303
304
305
305
306
307
307
308
309
309
310
311
312
313
314
314
315
315
316
316
317
317
318
318
319
319
320
321
322
323
324
325
325
326
326
327
327
328
328
329
329
330
331
332
333
334
335
335
336
336
337
337
338
338
339
339
340
341
342
343
344
345
345
346
346
347
347
348
348
349
349
350
351
352
353
354
355
355
356
356
357
357
358
358
359
359
360
361
362
363
364
365
365
366
366
367
367
368
368
369
369
370
371
372
373
374
375
375
376
376
377
377
378
378
379
379
380
381
382
383
384
385
385
386
386
387
387
388
388
389
389
390
391
392
393
394
395
395
396
396
397
397
398
398
399
399
400
401
402
403
404
405
405
406
406
407
407
408
408
409
409
410
411
412
413
414
415
415
416
416
417
417
418
418
419
419
420
421
422
423
424
425
425
426
426
427
427
428
428
429
429
430
431
432
433
434
435
435
436
436
437
437
438
438
439
439
440
441
442
443
444
445
445
446
446
447
447
448
448
449
449
450
451
452
453
454
455
455
456
456
457
457
458
458
459
459
460
461
462
463
464
465
465
466
466
467
467
468
468
469
469
470
471
472
473
474
475
475
476
476
477
477
478
478
479
479
480
481
482
483
484
485
485
486
486
487
487
488
488
489
489
490
491
492
493
494
495
495
496
496
497
497
498
498
499
499
500
501
502
503
504
505
505
506
506
507
507
508
508
509
509
510
511
512
513
514
515
515
516
516
517
517
518
518
519
519
520
521
522
523
524
525
525
526
526
527
527
528
528
529
529
530
531
532
533
534
535
535
536
536
537
537
538
538
539
539
540
541
542
543
544
545
545
546
546
547
547
548
548
549
549
550
551
552
553
554
555
555
556
556
557
557
558
558
559
559
560
561
562
563
564
565
565
566
566
567
567
568
568
569
569
570
571
572
573
574
575
575
576
576
577
577
578
578
579
579
580
581
582
583
584
585
585
586
586
587
587
588
588
589
589
590
591
592
593
594
595
595
596
596
597
597
598
598
599
599
600
601
602
603
604
605
605
606
606
607
607
608
608
609
609
610
611
612
613
614
615
615
616
616
617
617
618
618
619
619
620
621
622
623
624
625
625
626
626
627
627
628
628
629
629
630
631
632
633
634
635
635
636
636
637
637
638
638
639
639
640
641
642
643
644
645
645
646
646
647
647
648
648
649
649
650
651
652
653
654
655
655
656
656
657
657
658
658
659
659
660
661
662
663
664
665
665
666
666
667
667
668
668
669
669
670
671
672
673
674
675
675
676
676
677
677
678
678
679
679
680
681
682
683
684
685
685
686
686
687
687
688
688
689
689
690
691
692
693
694
695
695
696
696
697
697
698
698
699
699
700
701
702
703
704
705
705
706
706
707
707
708
708
709
709
710
711
712
713
714
715
715
716
716
717
717
718
718
719
719
720
721
722
723
724
725
725
726
726
727
727
728
728
729
729
730
731
732
733
734
735
735
736
736
737
737
738
738
739
739
740
741
742
743
744
745
745
746
746
747
747
748
748
749
749
750
751
752
753
754
755
755
756
756
757
757
758
758
759
759
760
761
762
763
764
765
765
766
766
767
767
768
768
769
769
770
771
772
773
774
775
775
776
776
777
777
778
778
779
779
780
781
782
783
784
785
785
786
786
787
787
788
788
789
789
790
791
792
793
794
795
795
796
796
797
797
798
798
799
799
800
801
802
803
804
805
805
806
806
807
807
808
808
809
809
810
811
812
813
814
815
815
816
816
817
817
818
818
819
819
820
821
822
823
824
825
825
826
826
827
827
828
828
829
829
830
831
832
833
834
835
835
836
836
837
837
838
838
839
839
840
841
842
843
844
845
845
846
846
847
847
848
848
849
849
850
851
852
853
854
855
855
856
856
857
857
858
858
859
859
860
861
862
863
864
865
865
866
866
867
867
868
868
869
869
870
871
872
873
874
875
875
876
876
877
877
878
878
879
879
880
881
882
883
884
885
885
886
886
887
887
888
888
889
889
890
891
892
893
894
895
895
896
896
897
897
898
898
899
899
900
901
902
903
904
905
905
906
906
907
907
908
908
909
909
910
911
912
913
914
915
915
916
916
917
917
918
918
919
919
920
921
922
923
924
925
925
926
926
927
927
928
928
929
929
930
931
932
933
934
935
935
936
936
937
937
938
938
939
939
940
941
942
943
944
945
945
946
946
947
947
948
948
949
949
950
951
952
953
954
955
955
956
956
957
957
958
958
959
959
960
961
962
963
964
965
965
966
966
967
967
968
968
969
969
970
971
972
973
974
975
975
976
976
977
977
978
978
979
979
980
981
982
983
984
985
985
986
986
987
987
988
988
989
989
990
991
992
993
994
995
995
996
996
997
997
998
998
999
999
1000
1001
1002
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1011
1012
1013
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1021
1022
1023
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1031
1032
1033
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1041
1042
1043
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1061
1062
1063
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1071
1072
1073
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1111
1112
1113
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1121
1122
1123
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1131
1132
1133
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1141
1142
1143
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1161
1162
1163
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1171
1172
1173
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1211
1212
1213
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1312
1313
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1412
1413
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1765
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1775
1776
1776
1777
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1785
1785
1786
1786
1787
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1795
1796
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1805
1806
1806
1807
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1815
1815
1816
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1825
1826
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1835
1836
1836
1837
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1845
1846
1846
1847
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1855
1856
1856
1857
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1865
1865
1866
1
```

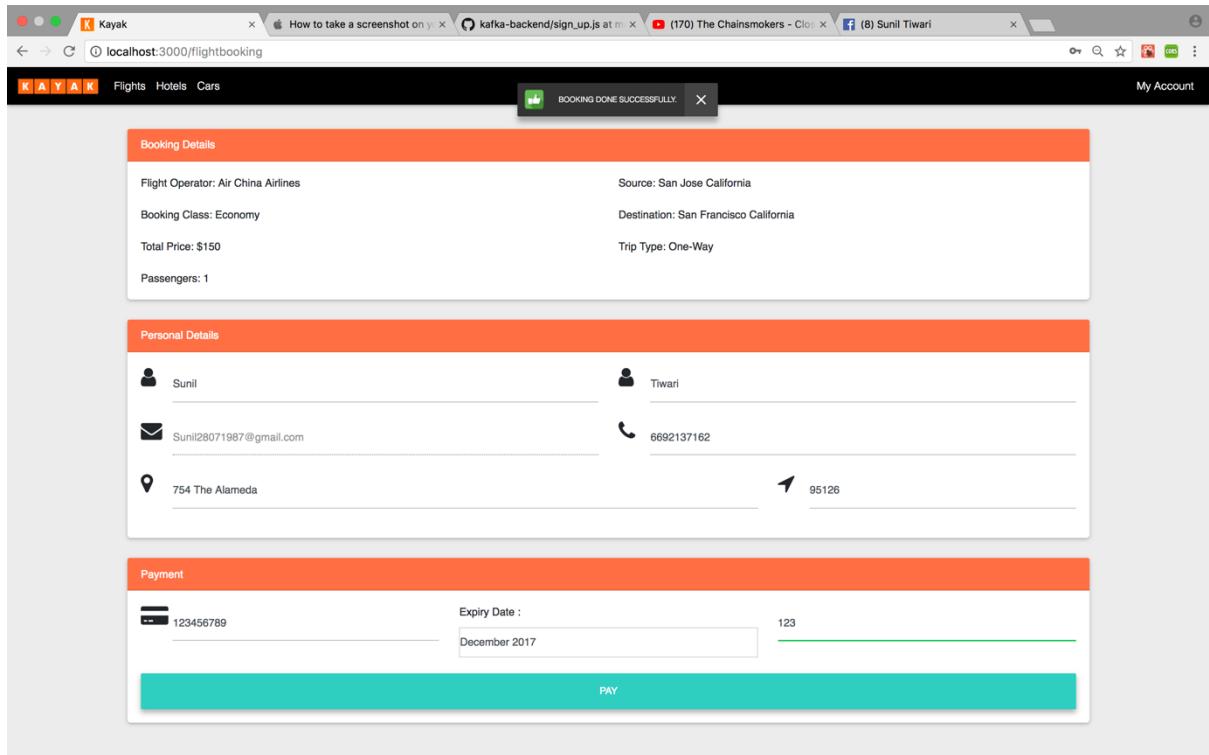
The screenshot shows a code editor interface with the file `flight.js` open. The code is part of a Node.js application, specifically a route handler for adding flights. It uses `req` and `res` objects from Express.js, and `kafka` to make requests to an external service. The code includes logging statements and conditional logic for handling errors and returning results.

```
45 router.post('/addflight', function (req, res) {
46   console.log('Request is -----');
47   console.log(req.body);
48
49   var reqObject = {
50     email : req.session.email,
51     flight : req.body
52   };
53
54   kafka.make_request("addflight", reqObject, function(err,results){
55
56     if(err){
57       console.log('Returning Error ----' + err);
58       res.send({'status': err.code, 'message' : err.message});
59     }
60     else {
61
62       console.log('Returning results ----' + results);
63       if(results.code == "200"){
64         res.send({'status': results.code });
65       }
66       else {
67         res.send({'status': results.code });
68       }
69     }
70   });
71 });
72
73
74 router.post('/book', function (req, res) {
75   console.log(req.body);
76   var queueName = "BookFlight";
77
78   var reqObject = {
79     email : req.session.email,
80     booking : req.body.booking,
81     credit_card : req.body.credit_card
82   };
83
84   kafka.make_request(queueName, reqObject, function(err,results){
85
86     if(err){
87       console.log('Returning Error ----' + err);
88       res.send({'status': err.code, 'message' : err.message});
89     }
90     else {
91
92       console.log('Returning results ----' + results);
93       if(results.code == "200"){
94         res.send({'status': results.code , 'api_results' : results.value});
95       }
96       else {
97         res.send({'status': results.code , 'api_results' : results.value});
98       }
99     }
100   });
101
102
103
104
105 module.exports = router;
```

This screenshot shows the same `flight.js` file as the first one, but with some changes made to the code. The `module.exports = router;` statement has been moved to the bottom of the file. The code structure and logic remain the same, handling requests for adding flights and booking flights, and returning results based on the status code.

```
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105 module.exports = router;
```

Flight Booking Page (UI):



Flight Booking Page (Node Backend):

A screenshot of an IDE interface showing the 'flight.js' file from the 'kafka-front-end' project. The code is a Node.js script for booking a flight. It defines a 'booking' object with various properties like operator, origin city, destination city, trip type, class, price, and passenger count. It also includes a 'credit_card' object with card number, valid till date, and CVV. The 'travellerinfo' object contains personal information such as first name, last name, email, phone number, address, and zipcode. The script uses Kafka's 'make request' function to queue a flight booking. It handles responses and logs messages to the console, including a successful insertion query and a response indicating the flight was booked successfully.

```
function makeRequest() {
    var booking = {
        flight: {
            operator: 'Air China Airlines',
            originCity: 'San Jose',
            originState: 'California',
            destinationCity: 'San Francisco',
            destinationState: 'California',
            tripType: 'One-Way',
            flightClass: 'Economy',
            capacity: 50,
            price: 150,
            bookingStartDate: '11/6/2017',
            bookingEndDate: '',
            passengers: 1,
            flightId: 'MMT101',
            sourceAirport: 'San Francisco International Airport',
            destinationAirport: 'San Jose Airport' },
        credit_card: { card_number: '123456789', valid_till: '2017-12', cvv: '123' },
        travellerInfo: {
            firstname: 'Sunil',
            lastname: 'Tiwari',
            email: 'Sunil28071987@gmail.com',
            phoneno: '6692137162',
            address: '754 The Alameda',
            zipcode: '95126' } }
    var kafka = require('kafka-node');
    var client = new kafka.KafkaClient();
    var producer = client.createProducer();
    var topic = 'queueBookFlight';
    var message = JSON.stringify(booking);
    producer.send([ { topic: topic, value: message } ], function(err, data) {
        if (err) {
            console.log("Error in sending message: " + err);
        } else {
            console.log("Message sent successfully!");
        }
    });
}
```

Flight Booking Page (Kafka Backend):

The screenshot shows an IDE interface with the file `flight.js` open. The code handles a POST request to '/book' and sends a message to a Kafka topic named 'BookFlight'. The message contains booking details like email, booking, and credit card information.

```
router.post('/book', function (req, res) {
  console.log(req.body);
  var queueName = "BookFlight";
  var reqObject = {
    email : req.session.email,
    booking : req.body.booking,
    credit_card : req.body.credit_card
  };
  kafka.make_request(queueName, reqObject, function(err,results){
    callback for post()
  })
});
```

The terminal below shows the results of a database query:

```
DB Results:--  
Total Seats Booked in the Current Flight are - 0  
capacity is 50  
One way Flight available for booking MMT101  
SQL Query::insert into BILLING('user_email','target_id','target_name','source_airport','destination_airport','booking_type','billing_amount','target_count','source_city','source_state','destination_city','destination_state','flight_trip_type','booking_class','booking_start_date','booking_end_date','credit_card_type','credit_card_number','credit_card_holder_name','credit_card_valid_from','credit_card_valid_till') values('Sunil28071987@gmail.com','MMT101','Air China Airlines','San Francisco International Airport','San Jose Airport','FLIGHT','150','1','San Jose','California','San Francisco','California','One-Way','Economy','11/6/2017','','undefined','123456789','undefined','undefined','2017-12');
```

Connection closed.
{ code: '200', value: 'One-Way Flight booked Successfully' }

[{ topic: 'response_topic',
 messages: [{ "correlationId": "47eebe418b5e8de810633ca8deba58bd", "data": { "code": "200", "value": "One-Way Flight booked Successfully" } },
 partition: 0 }]

{ response_topic: { '0': 264 } }

Flight Booking (Business Logic):

The screenshot shows an IDE interface with the file `flight.js` open. The code handles a POST request to '/book' and sends a message to a Kafka topic named 'BookFlight'. It includes error handling for the Kafka request. The code also exports the router at the bottom.

```
router.post('/book', function (req, res) {
  console.log(req.body);
  var queueName = "BookFlight";
  var reqObject = {
    email : req.session.email,
    booking : req.body.booking,
    credit_card : req.body.credit_card
  };
  kafka.make_request(queueName, reqObject, function(err,results){
    if(err){
      console.log('Returning Error ----' + err);
      res.send({status: err.code, 'message' : err.message});
    }
    else {
      console.log('Returning results ----' + results);
      if(results.code == '200'){
        res.send({status: results.code , 'api_results' : results.value});
      }
      else {
        res.send({status: results.code , 'api_results' : results.value});
      }
    }
  });
});
```

```
module.exports = router;
```

The terminal below shows the results of a database query:

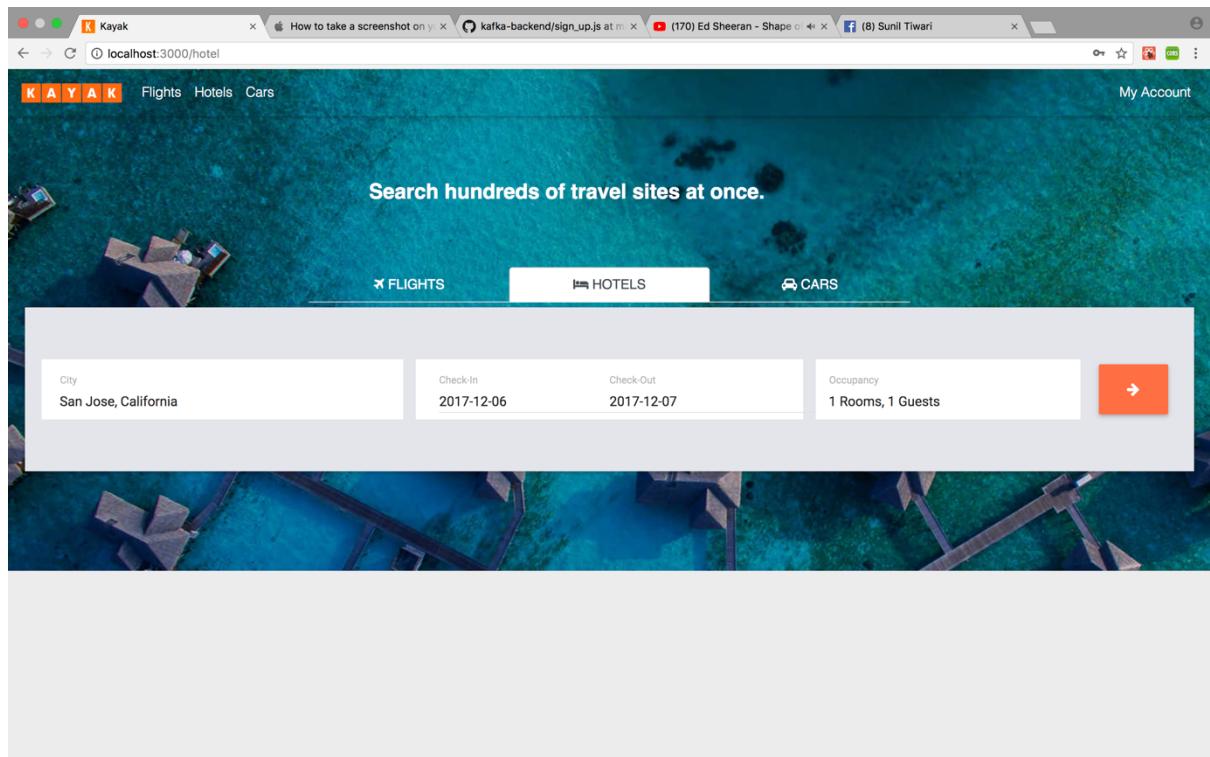
```
DB Results:--  
Total Seats Booked in the Current Flight are - 0  
capacity is 50  
One way Flight available for booking MMT101  
SQL Query::insert into BILLING('user_email','target_id','target_name','source_airport','destination_airport','booking_type','billing_amount','target_count','source_city','source_state','destination_city','destination_state','flight_trip_type','booking_class','booking_start_date','booking_end_date','credit_card_type','credit_card_number','credit_card_holder_name','credit_card_valid_from','credit_card_valid_till') values('Sunil28071987@gmail.com','MMT101','Air China Airlines','San Francisco International Airport','San Jose Airport','FLIGHT','150','1','San Jose','California','San Francisco','California','One-Way','Economy','11/6/2017','','undefined','123456789','undefined','undefined','2017-12');
```

Connection closed.
{ code: '200', value: 'One-Way Flight booked Successfully' }

[{ topic: 'response_topic',
 messages: [{ "correlationId": "47eebe418b5e8de810633ca8deba58bd", "data": { "code": "200", "value": "One-Way Flight booked Successfully" } },
 partition: 0 }]

{ response_topic: { '0': 264 } }

Hotel Search Page (UI):



A screenshot of a web browser showing the Kayak hotel list interface. The URL is `localhost:3000/hotelist`. The page has a dark header with the Kayak logo, 'Flights', 'Hotels', 'Cars', and 'My Account'. On the left, there are two filter sections: 'Price' with a slider from \$120 to \$520 and 'Star' with a slider from 1 to 5. The main content area displays two hotel cards. The first card is for 'Holiday Inn Express' (Delux room) with a price of \$120, a rating of 8.5, and 3 reviews. The second card is for 'Lemon Tree' (Delux room) with a price of \$520, a rating of 9.5, and 3 reviews. Each card includes a 'BOOK' button.

Hotel	Room Type	Price	Rating	Reviews	Action
Holiday Inn Express	Delux	\$120	8.5	3 Reviews	BOOK
Lemon Tree	Delux	\$520	9.5	3 Reviews	BOOK

Kayak Flights Hotels Cars My Account

localhost:3000/hotellist

Price

Low:\$120 High:\$520

Star

1 5



Holiday Inn
★★★★★
8.5
3 Reviews

\$120
Delux
BOOK

\$220
Super Delux
BOOK

\$320
Premium
BOOK

Details Map Review

Very good hotel. Easily accessible by car and good for sightseeing. Great rooms and fantastic service. Awesome vibe.

Address:
101 E San Fernando Street
San Jose, California - 95112

\$520

Kayak Flights Hotels Cars My Account

localhost:3000/hotellist

Price

Low:\$120 High:\$295

Star

4 5



Hyatt
★★★★★
7.5
3 Reviews

\$120
Delux
BOOK

\$440
Super Delux
BOOK

\$1000
Premium
BOOK

Hotel Search Page (Node Backend):

The screenshot shows an IDE interface with a project structure for a 'kafka-front-end' application. The 'routes' folder contains a file named 'hotel.js'. The code implements a GET endpoint for hotels. It uses the 'request' module to interact with a search API, passing parameters like 'searchtype: hotel' and 'searchquery: req.query'. It handles errors and sends back JSON results. The terminal below shows several successful HTTP requests to the endpoint.

```
var kafka = require('kafka-client');
var search = require('./search');
var router = express.Router();

/* GET car list */
router.get('/hotels', function (req, res) {
  console.log("Inside Hotels");
  var data = {'searchtype': 'hotel', 'searchquery': req.query};
  search.searchFromApi(data, function (err, results) {
    if(err || !results){
      res.send({'status': 401});
    } else {
      res.send({'hotels': results, 'status': 201});
    }
  });
});

GET /hotels?vendor=TripAdvisorHotels&data=%7B%22city%22:%22San+Jose%22,%22state%22:%22California%22,%22occupancy%22:%221%22%7D 200 7.792 ms - 591
GET /hotels?city=San%20Jose&state=California&occupancy=1 304 93.975 ms -
GET /images/holidayinn.jpg 304 2.754 ms -
GET /images/lemontree.jpg 304 2.712 ms -
GET /images/hyatt.jpg 304 2.651 ms -
```

Hotel Search Page (Kafka Backend):

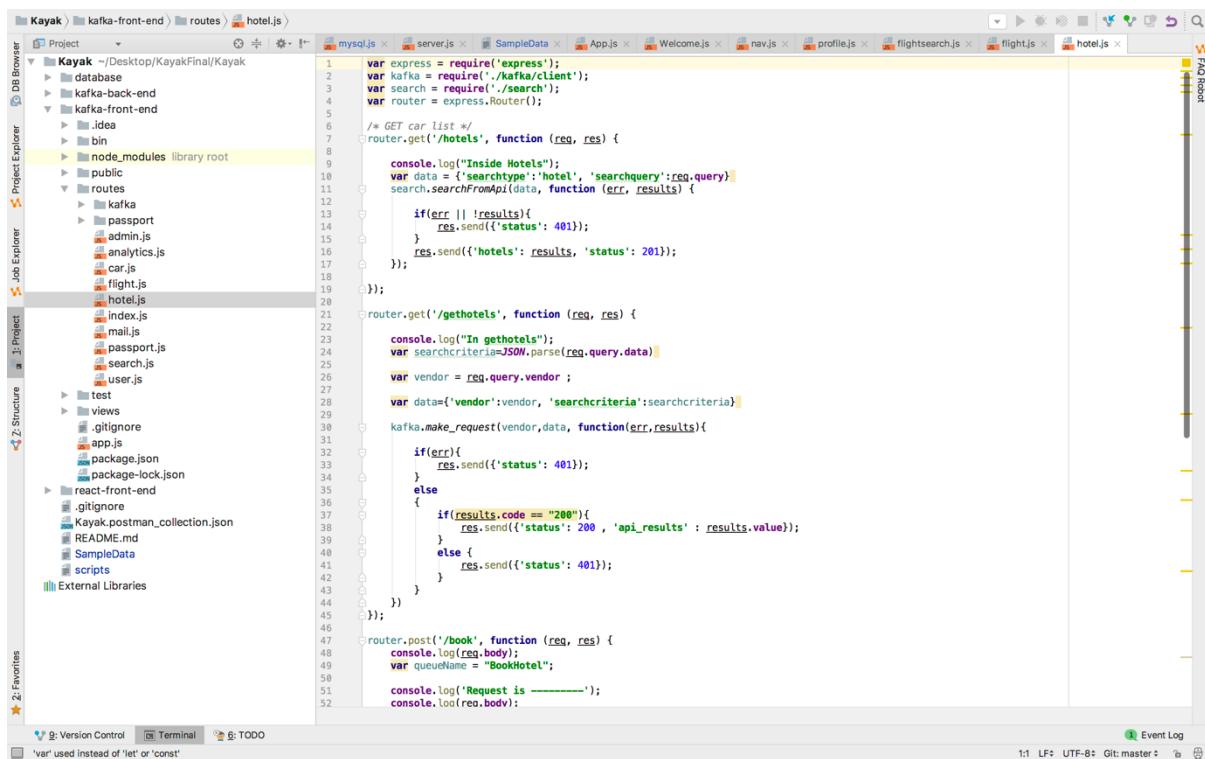
The screenshot shows an IDE interface with a project structure for a 'kafka-back-end' application. The 'routes' folder contains a file named 'hotel.js'. The code is identical to the Node.js version above, implementing a GET endpoint for hotels. The terminal shows the same series of successful HTTP requests to the endpoint.

```
var kafka = require('kafka-client');
var search = require('./search');
var router = express.Router();

/* GET car list */
router.get('/hotels', function (req, res) {
  console.log("Inside Hotels");
  var data = {'searchtype': 'hotel', 'searchquery': req.query};
  search.searchFromApi(data, function (err, results) {
    if(err || !results){
      res.send({'status': 401});
    } else {
      res.send({'hotels': results, 'status': 201});
    }
  });
});

GET /hotels?vendor=TripAdvisorHotels&data=%7B%22city%22:%22San+Jose%22,%22state%22:%22California%22,%22occupancy%22:%221%22%7D 200 7.792 ms - 591
GET /hotels?city=San%20Jose&state=California&occupancy=1 304 93.975 ms -
GET /images/holidayinn.jpg 304 2.754 ms -
GET /images/lemontree.jpg 304 2.712 ms -
GET /images/hyatt.jpg 304 2.651 ms -
```

Hotel Search (Business Logic):



```
var express = require('express');
var kafka = require('../kafka/client');
var search = require('../search');
var router = express.Router();

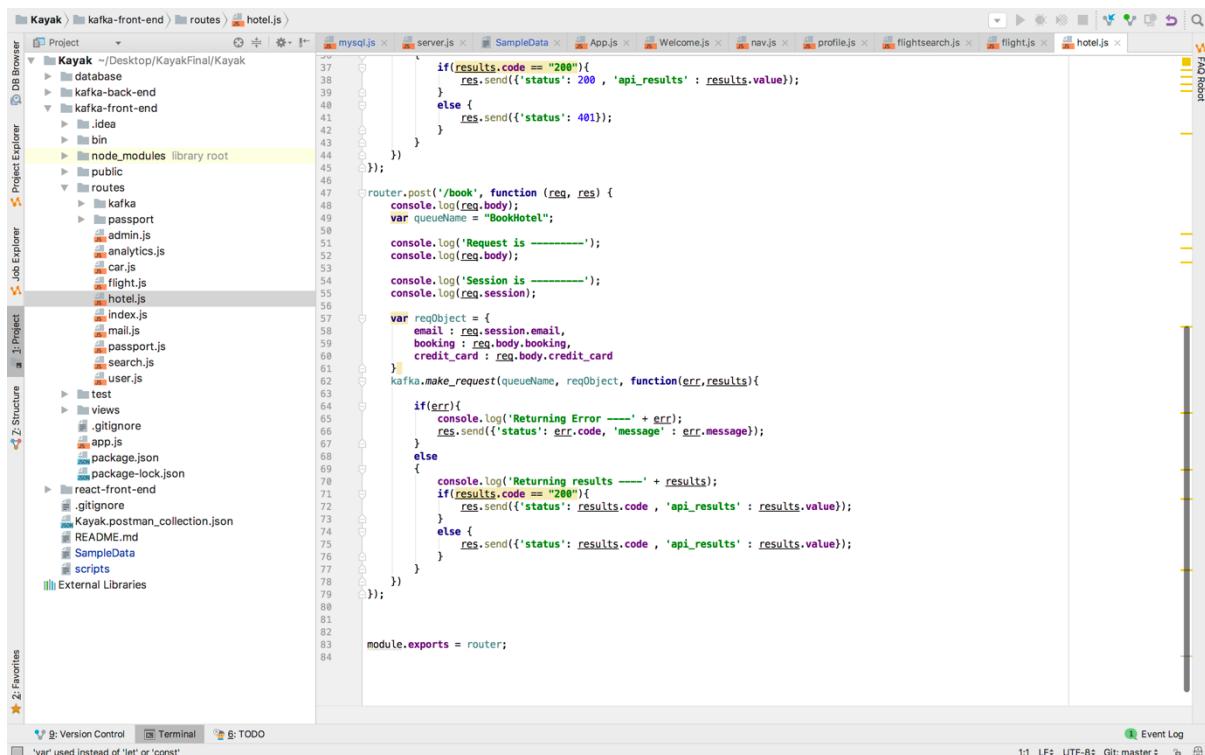
/* GET car list */
router.get('/hotels', function (req, res) {
  console.log("Inside Hotels");
  var data = {'searchtype': 'hotel', 'searchquery': req.query};
  search.searchFromApi(data, function (err, results) {
    if(err || !results){
      res.send({'status': 401});
    }
    res.send({'hotels': results, 'status': 201});
  });
});

router.get('/gethotels', function (req, res) {
  console.log("In gethotels");
  var searchcriteria=JSON.parse(req.query.data);
  var vendor = req.query.vendor;
  var data={ 'vendor': vendor, 'searchcriteria': searchcriteria};
  kafka.make_request(vendor,data, function(err,results){
    if(err){
      res.send({'status': 401});
    }
    else {
      if(results.code == "200"){
        res.send({'status': 200 , 'api_results' : results.value});
      }
      else {
        res.send({'status': 401});
      }
    }
  });
});

router.post('/book', function (req, res) {
  console.log(req.body);
  var queueName = "BookHotel";
  console.log('Request is -----');
  console.log(req.body);
  console.log('Session is -----');
  console.log(req.session);

  var reqObject = {
    email : req.session.email,
    booking : req.body.booking,
    credit_card : req.body.credit_card
  };
  kafka.make_request(queueName, reqObject, function(err,results){
    if(err){
      console.log('Returning Error ----' + err);
      res.send({'status': err.code, 'message': err.message});
    }
    else {
      console.log('Returning results ----' + results);
      if(results.code == "200"){
        res.send({'status': results.code , 'api_results' : results.value});
      }
      else {
        res.send({'status': results.code , 'api_results' : results.value});
      }
    }
  });
});

module.exports = router;
```



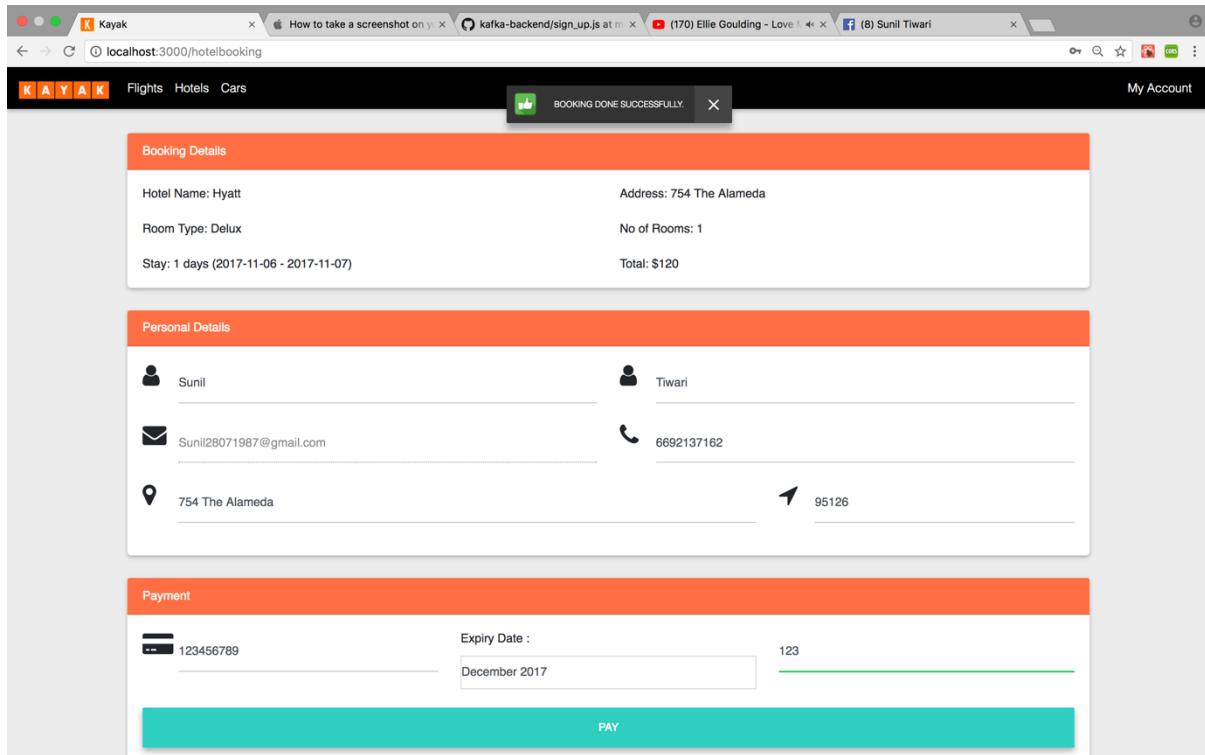
```
if(results.code == "200"){
  res.send({'status': 200 , 'api_results' : results.value});
}
else {
  res.send({'status': 401});
}

router.post('/book', function (req, res) {
  console.log(req.body);
  var queueName = "BookHotel";
  console.log('Request is -----');
  console.log(req.body);
  console.log('Session is -----');
  console.log(req.session);

  var reqObject = {
    email : req.session.email,
    booking : req.body.booking,
    credit_card : req.body.credit_card
  };
  kafka.make_request(queueName, reqObject, function(err,results){
    if(err){
      console.log('Returning Error ----' + err);
      res.send({'status': err.code, 'message': err.message});
    }
    else {
      console.log('Returning results ----' + results);
      if(results.code == "200"){
        res.send({'status': results.code , 'api_results' : results.value});
      }
      else {
        res.send({'status': results.code , 'api_results' : results.value});
      }
    }
  });
});

module.exports = router;
```

Hotel Booking Page (UI):

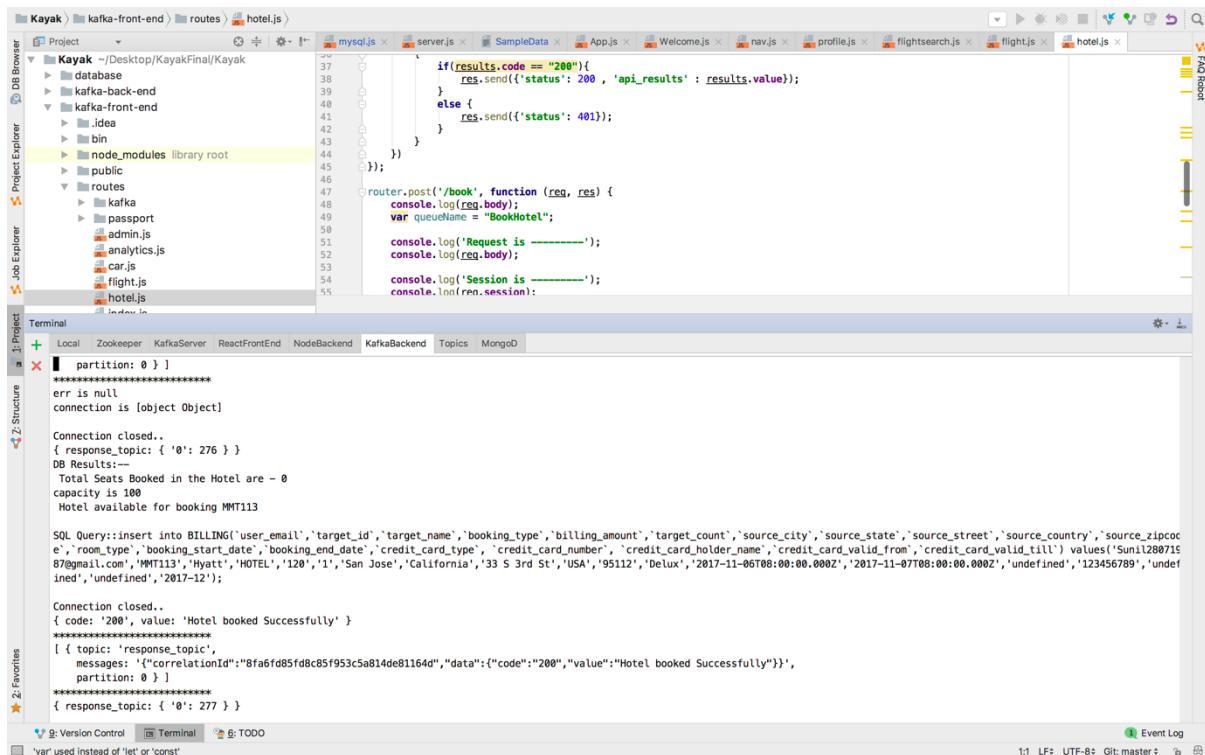


Hotel Booking Page (Node Backend):

The screenshot shows the Eclipse IDE interface with the 'Node.js' project selected. The 'Project Explorer' view shows files like 'mysql.js', 'server.js', 'SampleData.js', 'App.js', 'Welcome.js', 'nav.js', 'profile.js', 'flightsearch.js', 'flight.js', and 'hotel.js'. The 'Terminal' view shows the command-line interface with the following log output:

```
Request Body : { userId: 'Sunil28071987@gmail.com',
  sessionId: 'session1',
  eventTime: '2017-12-4 4:59:28',
  eventName: 'HotelBooking',
  pageId: 'HotelBooking',
  buttonId: 'HotelBooking',
  objectId: 'HotelBooking',
  pageNav: 'HotelSearch HotelBooking' }
in make request from the queueclick_tracker_req
...in kafkarpc....
in response
in response1
true
in response2
{ click_tracker_req: { '0': 28 } }
in response2
{ BookHotel: { '0': 0 } }
msg received
response { code: '200', value: 'DB inset query successfull !!!.' }
POST /analytics/clicktracker 200 11.704 ms - 57
msg received
response { code: '200', value: 'Hotel booked Successfully' }
Returning results ----[object Object]
POST /hotel/book 200 26.772 ms - 58
```

Hotel Booking Page (Kafka Backend):



```
if(results.code == "200"){
    res.send({'status': 200 , 'api_results' : results.value});
}
else {
    res.send({'status': 401});
}

router.post('/book', function (req, res) {
    console.log(req.body);
    var queueName = "BookHotel";
    console.log('Request is -----');
    console.log(req.body);

    console.log('Session is -----');
    console.log(req.session);

    var partition: 0 } }

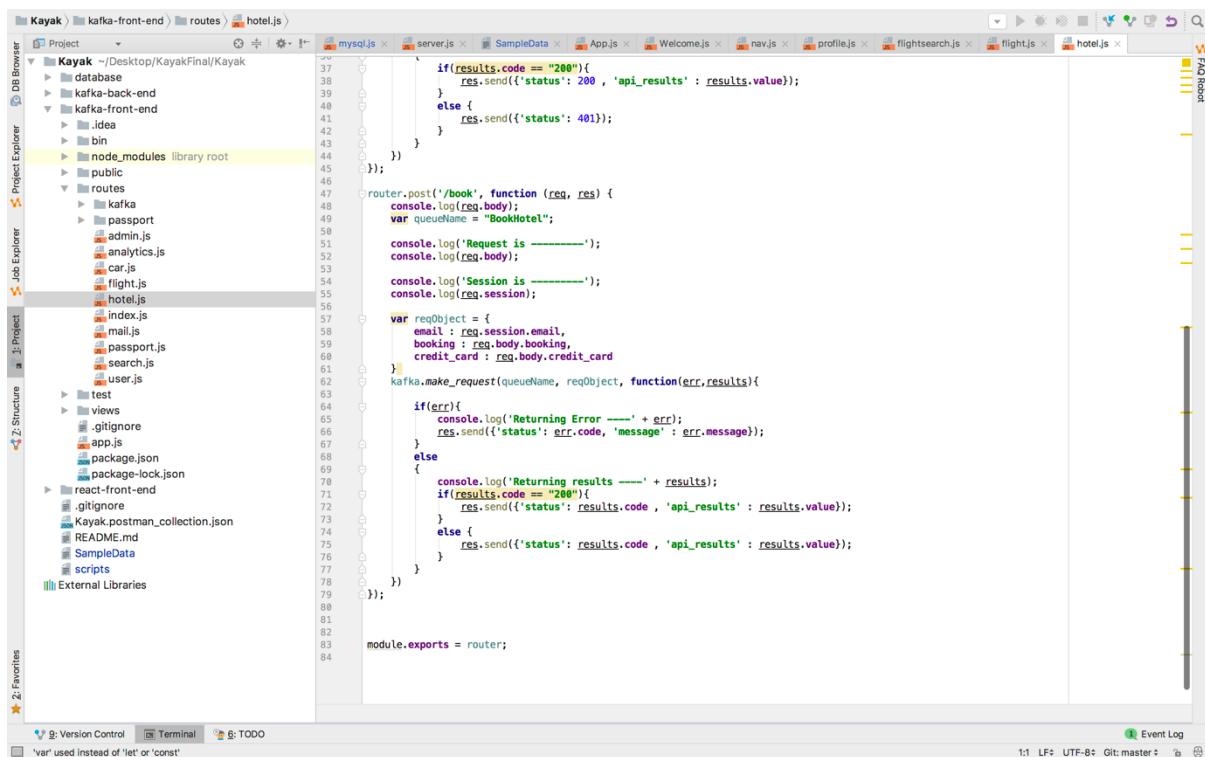
*****err is null
connection is [object Object]

Connection closed..
{ response_topic: { '0': 276 } }
DB Results:--
Total Seats Booked in the Hotel are - 0
capacity is 100
Hotel available for booking MMT113

SQL Query::insert into BILLING('user_email','target_id','target_name','booking_type','billing_amount','target_count','source_city','source_state','source_street','source_country','source_zipcode','room_type','booking_start_date','booking_end_date','credit_card_type','credit_card_number','credit_card_holder_name','credit_card_valid_from','credit_card_valid_till') values('Sunil28071587@gmail.com','MMT113','Hyatt','HOTEL','120','1','San Jose','California','33 S 3rd St','USA','95112','Delux','2017-11-07T08:00:00Z','2017-11-07T08:00:00Z','undefined','123456789','undefined','2017-12');

Connection closed..
{ code: '200', value: 'Hotel booked Successfully' }
*****
[ { topic: 'response_topic',
  messages: [ { "correlationId": "0fa6fd85fd8c85f953c5a814de81164d", "data": { "code": "200", "value": "Hotel booked Successfully" } },
  partition: 0 } ]
*****
{ response_topic: { '0': 277 } }
```

Hotel Booking (Business Logic):



```
if(results.code == "200"){
    res.send({'status': 200 , 'api_results' : results.value});
}
else {
    res.send({'status': 401});
}

router.post('/book', function (req, res) {
    console.log(req.body);
    var queueName = "BookHotel";
    console.log('Request is -----');
    console.log(req.body);

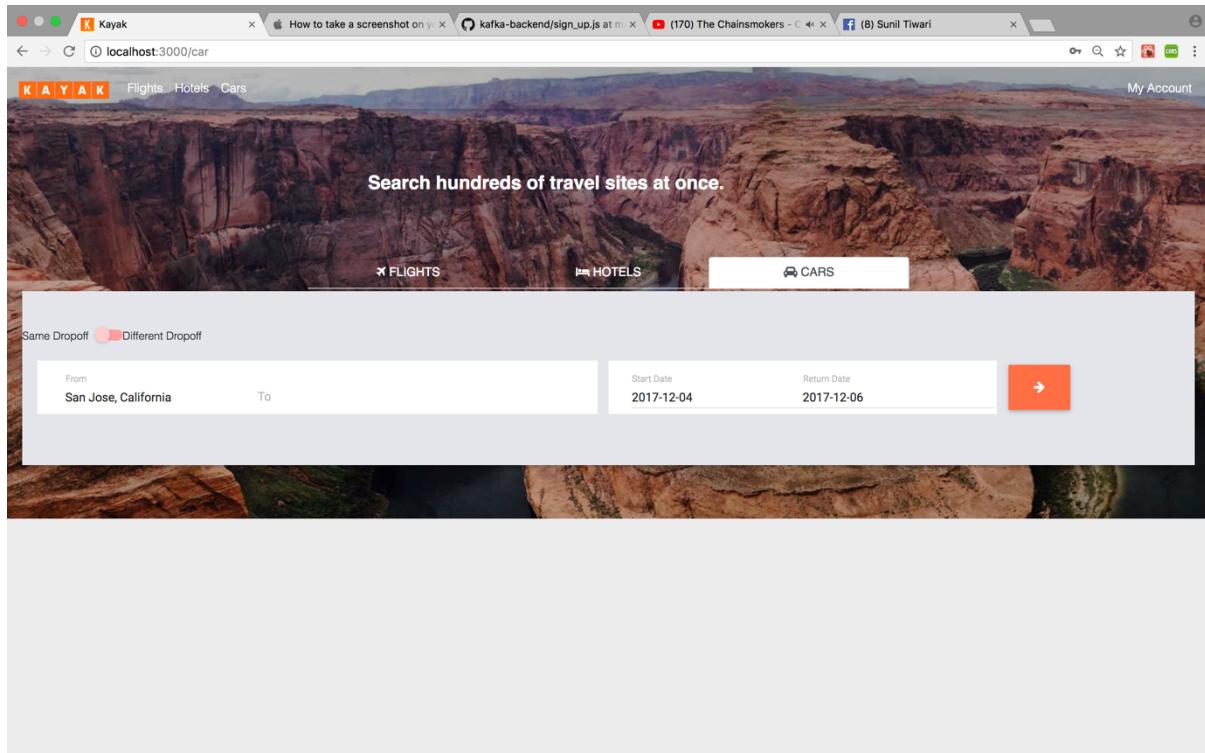
    console.log('Session is -----');
    console.log(req.session);

    var reqObject = {
        email : req.session.email,
        booking : req.body.booking,
        credit_card : req.body.credit_card
    }
    kafka.make_request(queueName, reqObject, function(err,results){

        if(err){
            console.log('Returning Error ----' + err);
            res.send({ 'status': err.code, 'message' : err.message });
        }
        else {
            console.log('Returning results ----' + results);
            if(results.code == "200"){
                res.send({ 'status': results.code , 'api_results' : results.value });
            }
            else {
                res.send({ 'status': results.code , 'api_results' : results.value });
            }
        }
    });
});

module.exports = router;
```

Car Search Page (UI):



A screenshot of the Kayak car search results page. The title bar says "localhost:3000/carlist". The header is identical to the search page. On the left, there are filters for "Price" (a slider from Low:\$90 to High:\$230) and "Type" (checkboxes for Small, Medium, Large, SUV, Luxury, Van, Pickup Truck, and Convertible). The main content area displays three car listings in cards:

- Audi**
Pickup : 101 E San Fernando St., San Jose, California
Dropoff : 101 E San Fernando St., San Jose, California
[GET DIRECTION](#)


\$90
Luxury
[BOOK](#)
- BMW**
Pickup : 101 E San Fernando St., San Jose, California
Dropoff : 101 E San Fernando St., San Jose, California
[GET DIRECTION](#)


\$100
Small
[BOOK](#)
- BMW**
Pickup : 101 E San Fernando St., San Jose, California
Dropoff : 101 E San Fernando St., San Jose, California
[GET DIRECTION](#)


\$150
SUV
[BOOK](#)

Kayak Flights Hotels Cars My Account

localhost:3000/carlist

Price

Low:\$90 High:\$230

Type

- Small
- Medium
- Large
- SUV
- Luxury
- Van
- Pickup Truck
- Convertible

Audi
Pickup : 101 E San Fernando St., San Jose, California
Dropoff : 101 E San Fernando St., San Jose, California

\$90
Luxury

BOOK

GET DIRECTION



Kayak Flights Hotels Cars My Account

localhost:3000/carlist

Price

Low:\$159 High:\$230

Type

- Small
- Medium
- Large
- SUV
- Luxury
- Van
- Pickup Truck
- Convertible

Audi
Pickup : 101 E San Fernando St., San Jose, California
Dropoff : 101 E San Fernando St., San Jose, California

\$200
Medium

BOOK

GET DIRECTION



Car Search Page (Node Backend):

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "Kayak" and contains "kafka-front-end" and "kafka-back-end". The "node_modules" folder is highlighted.
- Code Editor:** The file "hotel.js" is open, showing Node.js code for handling a car search request. It includes logic to check the response code and send appropriate JSON back to the client.
- Terminal:** The terminal shows several log entries related to car search requests, including successful responses (status 200) and errors (status 401).
- Logs:** The event log at the bottom right indicates 1:1 LF: UTF-8: Git: master.

Car Search Page (Kafka Backend):

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "Kayak" and contains "kafka-front-end" and "kafka-back-end". The "node_modules" folder is highlighted.
- Code Editor:** The file "hotel.js" is open, showing Kafka backend code for a car search endpoint. It prints the car list to the console.
- Terminal:** The terminal shows the printed car list, which includes two car objects: one for a "Lexus" and one for a "Van".
- Logs:** The event log at the bottom right indicates 1:1 LF: UTF-8: Git: master.

Car Search (Business Logic):

```
var express = require('express');
var kafka = require('../kafka/client');
var search = require('../search');
var router = express.Router();

/* GET car list */
router.get('/cars', function (req, res) {
  var data = {'searchtype':'car', 'searchquery':req.query};
  search.searchFromApi(data, function (err, results) {
    if(err || !results){
      res.send({'status': 401});
    }
    res.send({'cars': results, 'status': 201});
  });
});

router.get('/getcars', function (req, res) {
  var searchcriteria=JSON.parse(req.query.data);
  var vendor = req.query.vendor ;
  var data={'vendor':vendor, 'searchcriteria':searchcriteria};

  kafka.make_request(vendor,data, function(err,results){
    if(err){
      res.send({'status': 401});
    }
    else {
      if(results.code == "200"){
        res.send({'status': 200 , 'api_results' : results.value});
      } else {
        res.send({'status': 401});
      }
    }
  });
});

router.post('/book', function (req, res) {
  //console.log(req.body);
  var queueName = "BookCar";
  var reqObject = {
```

```
else {
  if(results.code == "200"){
    res.send({'status': 200 , 'api_results' : results.value});
  } else {
    res.send({'status': 401});
  }
}

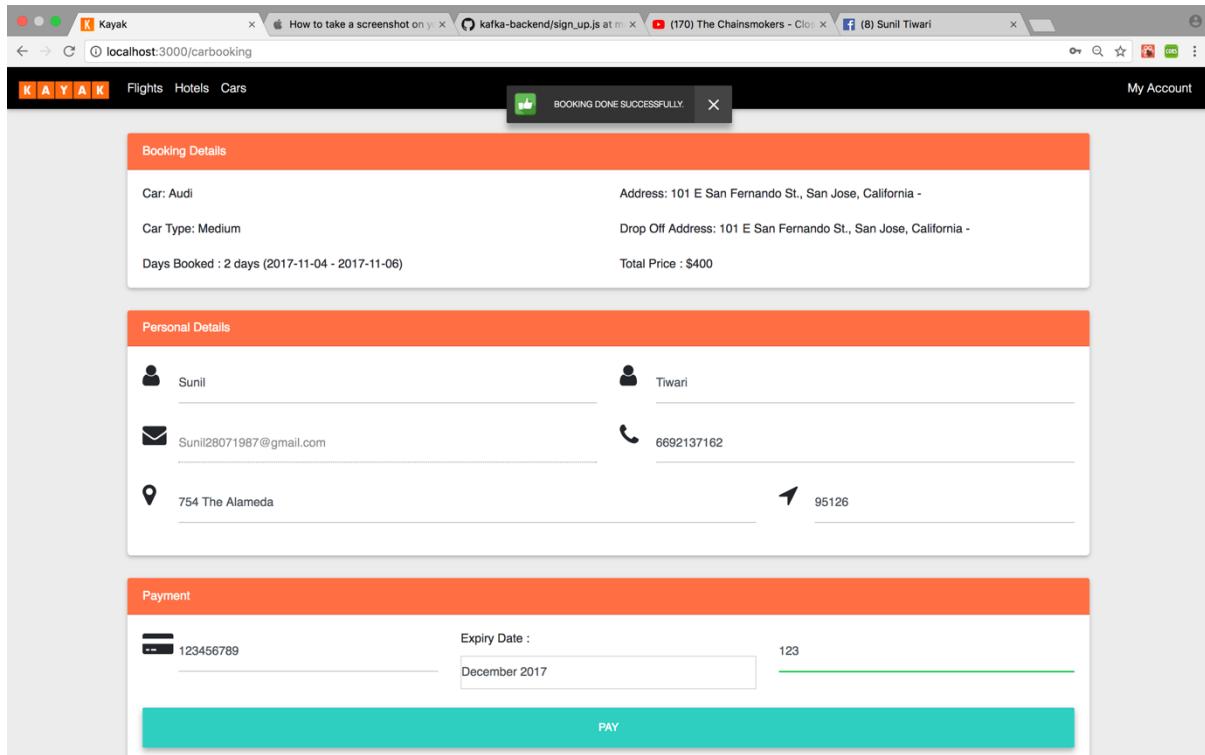
router.post('/book', function (req, res) {
  //console.log(req.body);
  var queueName = "BookCar";

  var reqObject = {
    email : req.session.email,
    booking : req.body.booking,
    credit_card : req.body.credit_card
  };

  //console.log(req.session.email);
  kafka.make_request(queueName, reqObject, function(err,results) {
    if (err) {
      console.log('Returning Error ----' + err);
      res.send({'status': err.code, 'message': err.message});
    }
    else {
      console.log('Returning results ----' + results);
      if (results.code == '200') {
        res.send({'status': results.code, 'api_results': results.value});
      }
      else {
        res.send({'status': results.code, 'api_results': results.value});
      }
    }
  });
});

module.exports = router;
```

Car Booking Page (UI):



Car Booking Page (Node Backend):

The screenshot shows the Eclipse IDE interface with a Node.js project named 'Kayak'. The 'Project Explorer' view shows files like mysql.js, server.js, SampleData.js, App.js, Welcome.js, nav.js, profile.js, flightsearch.js, flight.js, and hotel.js. The 'Terminal' view displays log messages from the Kafka backend, including a successful response to a 'book' request and a log entry for 'getcars' with vendor information. The code editor shows a portion of the 'hotel.js' file with logic for handling book requests.

```
if(results.code == "200"){
    res.send({'status': 200 , 'api_results' : results.value});
}
else {
    res.send({'status': 401});
}

router.post('/book', function (req, res) {
    console.log(req.body);
    var queueName = "BookHotel";
    console.log('Request is -----');
    console.log(req.body);
    console.log('Session is -----');

    if(results.code == "200"){
        res.send({'status': 200 , 'api_results' : results.value});
    }
    else {
        res.send({'status': 401});
    }
});
```

```
msg received
response { code: '200',
  value:
   [ { _id: '5a252e9c0cbbedeb45942269',
       carId: 'MMT101',
       cartype: 'Small',
       carmodel: 'BMW',
       imageurl: 'BMW.png',
       specification: 'automatic',
       pickupaddress: [Object],
       dropoffaddress: [Object],
       dailyrent: 100 },
     { _id: '5a252e9d0cbbedeb4594226c',
       carId: 'MMT102',
       cartype: 'SUV',
       carmodel: 'BMW',
       specification: 'automatic',
       imageurl: 'BMW.png',
       pickupaddress: [Object],
       dropoffaddress: [Object],
       dailyrent: 150 } ] }

GET /car/getcars?vendor=MmtCars&data=%7B%22pickupcity%22:%22San+Jose%22,%22pickupstate%22:%22California%22,%22dropoffcity%22:%22San+Jose%22,%22dropoffstate%22:%22California%22%7D 200 8.633 ms -
762
in make request from the queueCleartripCars
...in kafkarpc...
in response
```

Car Booking Page (Kafka Backend):

The screenshot shows an IDE interface with the following details:

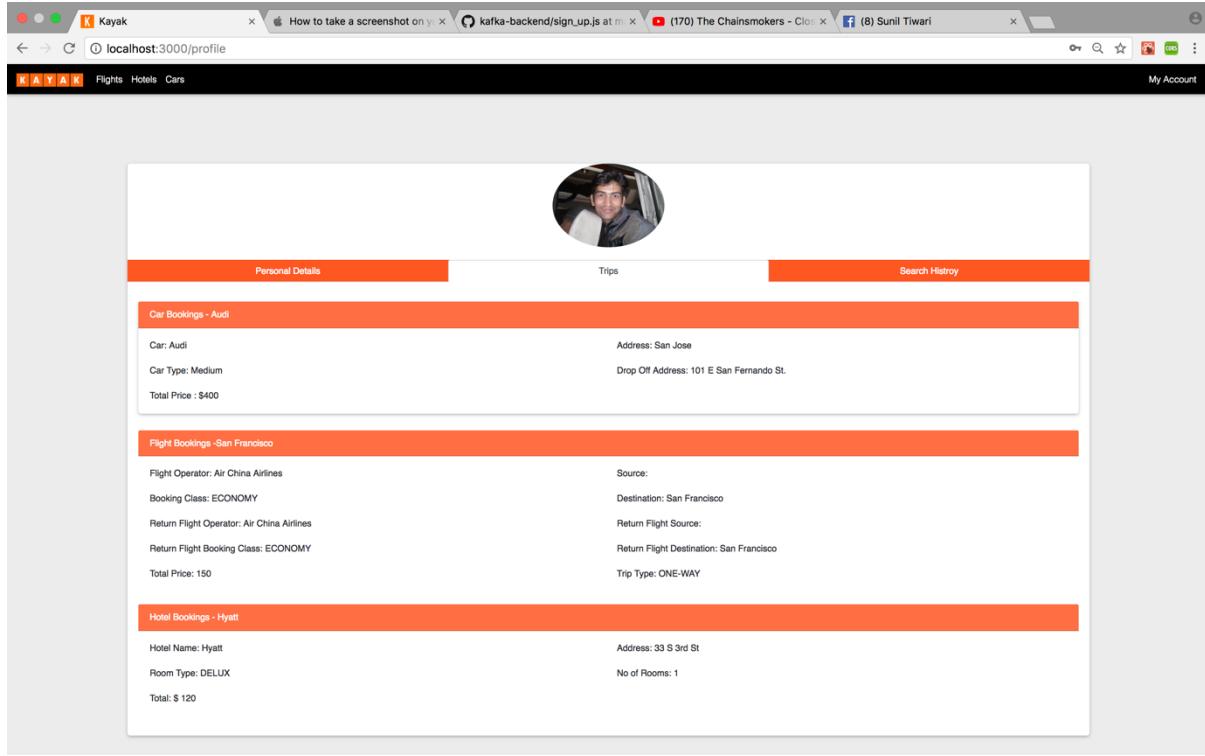
- Project Structure:** The project is named "Kayak" and contains "kafka-front-end" and "kafka-back-end". The "kafka-front-end" folder includes "routes" and "node_modules".
- Code Editor:** The file "hotel.js" is open, showing Node.js code for a Kafka backend. It includes a POST route for booking a car. The code logs the request and response, checks the result code, and sends a response back to the client.
- Terminal:** The terminal shows a JSON object representing a car booking request. It includes fields like _id, carId, carType, carModel, imageurl, specification, pickupaddress, and dropoffaddress. It also shows a list of topics and their correlation IDs.
- Bottom Status Bar:** Shows version control (Git master), encoding (UTF-8), and other system information.

Car Booking (Business Logic):

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "Kayak" and contains "kafka-front-end" and "kafka-back-end". The "kafka-front-end" folder includes "routes" and "node_modules".
- Code Editor:** The file "car.js" is open, showing Node.js code for business logic. It includes a POST route for booking a car. The code logs the request, creates a request object with session email, booking details, and credit card, and then makes a Kafka request using the "make_request" function.
- Bottom Status Bar:** Shows version control (Git master), encoding (UTF-8), and other system information.

User Booking History Page (UI):



User Booking History Page (Node Backend):

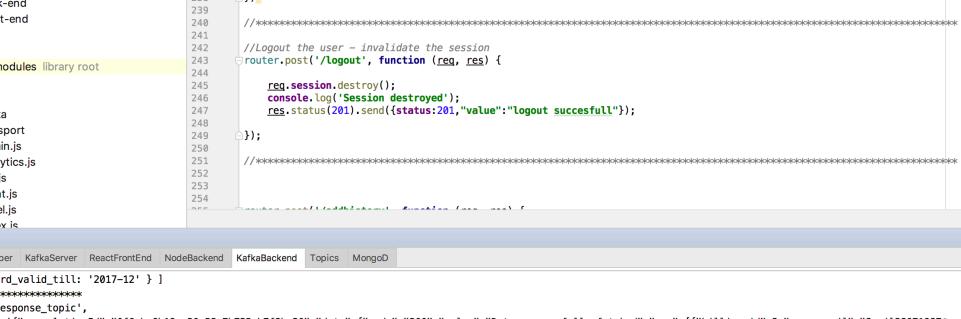
A screenshot of the Eclipse IDE showing a Node.js project structure and code editor. The project tree on the left shows a hierarchy of files and folders, including 'kafka-front-end', 'routes', 'user.js', 'node_modules', 'public', 'routes', 'kafka', 'passport', 'admin.js', 'analytics.js', 'car.js', 'flight.js', 'hotel.js', and 'index.js'. The code editor window displays a portion of the 'user.js' file, specifically the 'logout' function:

```
    //Logout the user - invalidate the session
    router.post('/logout', function (req, res) {
        req.session.destroy();
        console.log('Session destroyed');
        res.status(201).send({status:201,value:"logout successfull"});
    });
    //*****
```

The terminal window at the bottom shows a successful HTTP request:

```
i am here
Everything successfull
GET /user/bookinghistory 304 13.081 ms -
```

User Booking History Page (Kafka Backend):



The screenshot shows a Java IDE interface with multiple tabs open. The current tab is 'user.js'. In the code editor, the cursor is positioned at the start of a method call 'req.session.destroy();'. A tooltip from the IDE's code completion feature is displayed above the cursor, showing the full method signature: 'req.session.destroy()'.

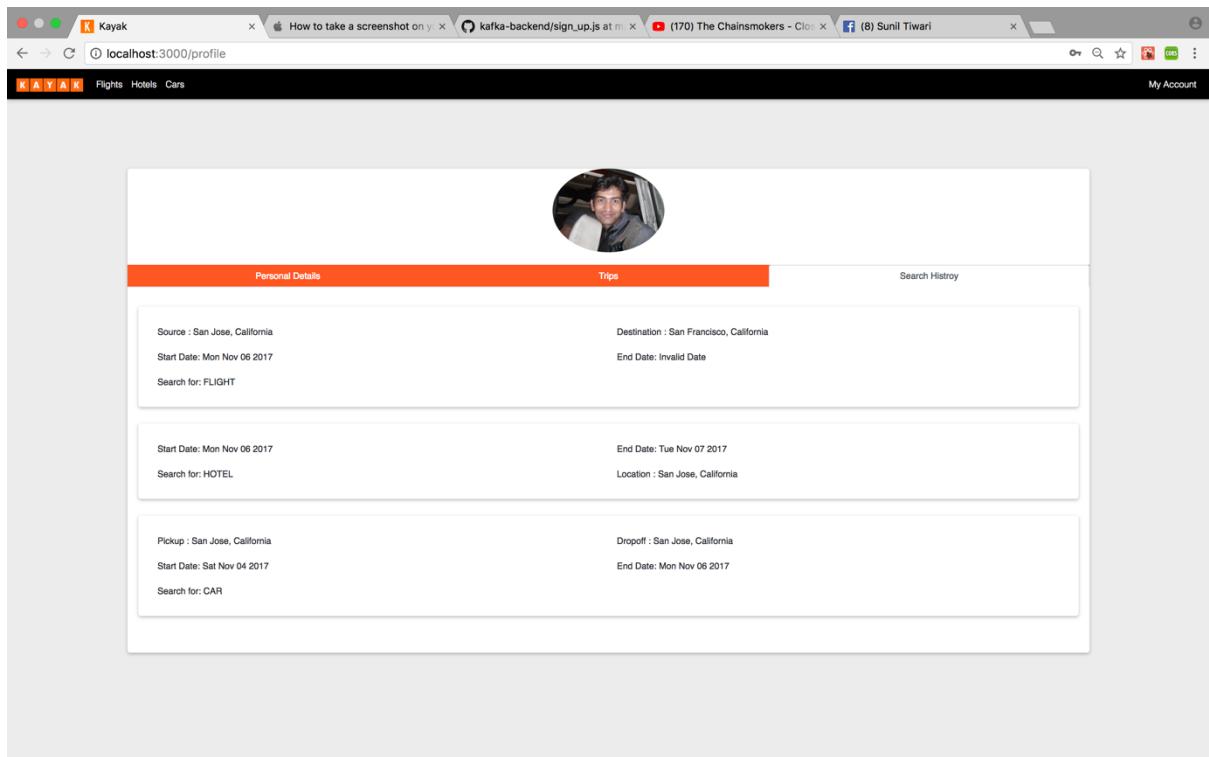
```
router.post('/Logout', function (req, res) {
    req.session.destroy();
    console.log('Session destroyed');
    res.status(201).send({status:201,value:"logout succesfull"});
});
```

User Booking History Page (Business Logic):

The screenshot shows a Java IDE interface with multiple tabs open. The current tab is 'user.js' under the 'routes' directory. The code in the file is as follows:

```
90     }
91   }
92 }
93 }
94 }
95 } else {
96   res.send({ "status": 401 });
97 });
98 //*****
99 router.get('/bookinghistory', function (req, res) {
100   // var reqEmail = req.body.email;
101   // var reqPassword = req.body.password;
102   // console.log("Inside history path");
103   var email = ("email": req.session.email);
104   kafka.make_request('bookings', email, function (err, results) {
105     if(err){
106       console.log("Error occurred");
107       res.send({ "status": 401, "data": results });
108     } else{
109       console.log("i am here");
110       if(results.code==200){
111         console.log("Everything successfull");
112         res.send({ "status": 201, "data": results });
113       } else {
114         res.send({ "status": 401, "data": results });
115       }
116     }
117   });
118 }
119 })
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 })
128 //*****
129 router.post('/register',function (req,res) {
130   console.log("____");
131   console.log(req.body);
132   console.log("____");
133   kafka.make_request('register', req.body ,function(err,results){
134     if(err){
135       console.log("After kafka response");
136     }
137   });
138 }
139 }
140 }
```

User Search History Page (UI):



User Search History Page (Node Backend):

A screenshot of a code editor showing the `user.js` file from the Kayak Node.js backend. The code defines a POST route for logging out users. It uses Express.js middleware to destroy the session and log a message. The code then handles errors and returns a success response. Below the code editor, the terminal shows several command-line entries related to the application's configuration and environment.

User Search History Page (Kafka Backend):

The screenshot shows a code editor with the file `user.js` open. The code handles a POST request for logging out a user. It uses `req.session.destroy()` to invalidate the session and logs a message to the console. A comment indicates the response status for a successful logout.

```
236
237
238
239
240
241
242 //Logout the user - invalidate the session
243 router.post('/logout', function (req, res) {
244   req.session.destroy();
245   console.log('Session destroyed');
246   res.status(201).end('Logout successful');
247});
```

The left sidebar shows the project structure with files like `App.js`, `Welcome.js`, `nav.js`, `profile.js`, `flightsearch.js`, `flight.js`, `hotel.js`, `car.js`, and `user.js`. The `node_modules` folder is highlighted.

User Search History Page (Business Logic):

The screenshot shows a code editor with the file `search.js` open. This script performs a search operation. It requires the `kafka` module, `axios` for making HTTP requests, and `node-async-loop` for asynchronous operations. The `searchFromApi` function sends a GET request to a Kafka topic endpoint with the provided search parameters. It then processes the results and pushes them into an array. Finally, it calls a callback function with the results or an error.

```
1 var kafka = require('./kafka/client');
2 var axios = require('axios');
3 var asyncLoop = require('node-async-loop');
4
5
6 function searchFromApi(data, callback){
7   console.log("search from api data");
8   console.log("*****");
9   console.log(data);
10  console.log("*****");
11  console.log(data.searchtype);
12  kafka.make_request('getapi', data.searchtype, function(err, results){
13
14    if(err){
15      callback(err, null);
16    }
17    else{
18      var search_results = [];
19
20      if(results.code == "200"){
21
22        var vendorArr=results.value;
23
24        asyncLoop(vendorArr, function (vendor, next)
25        {
26          axios.get(vendor.vendorapi,
27            {
28              params: {
29                'data': data.searchquery
30              }
31            }).then(function(response) {
32              search_results.push.apply(search_results,response.data.api_results);
33            })
34        });
35
36        next();
37
38      }
39
40      }, catch(function(error) {
41        callback(error, null);
42      });
43
44    }, function (err)
45    {
46      if (err)
47      {
48        callback(err, null);
49      }
50    }
51  });
52};
```

The left sidebar shows the project structure with files like `App.js`, `Welcome.js`, `nav.js`, `profile.js`, `flightsearch.js`, `flight.js`, `hotel.js`, `car.js`, `user.js`, `index.js`, `mail.js`, `passport.js`, `search.js`, and `user.js`. The `node_modules` folder is highlighted.

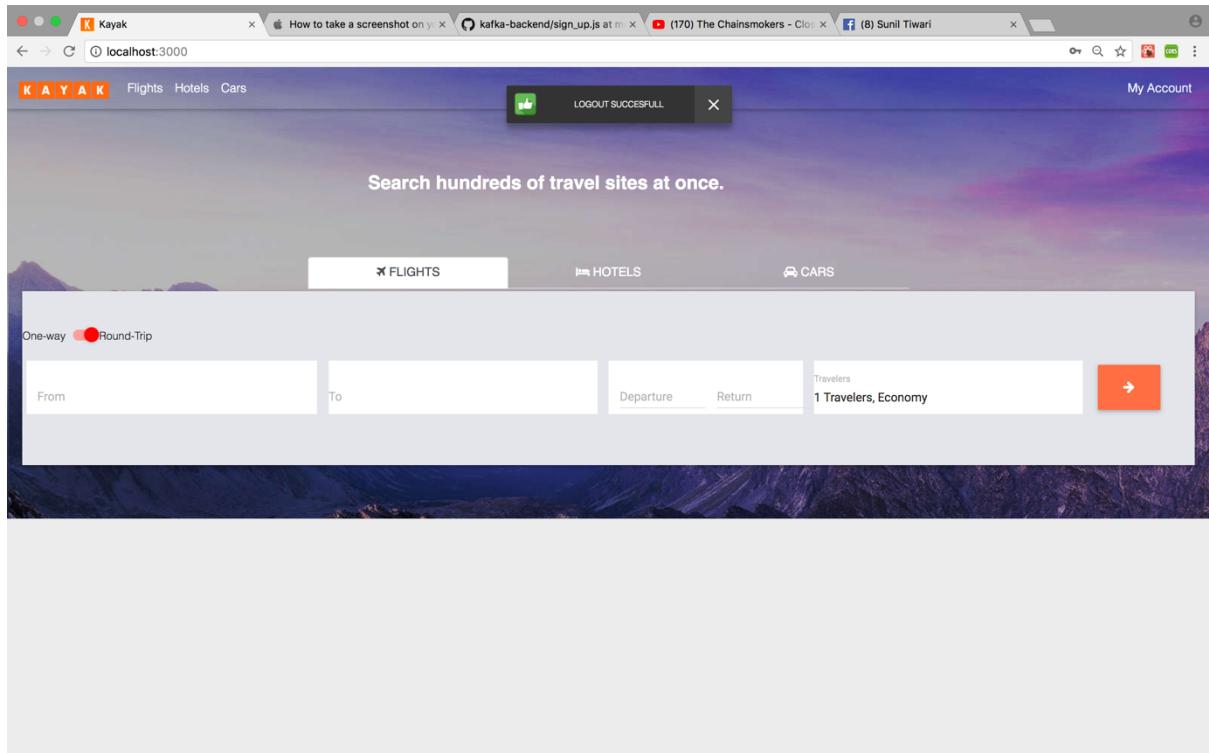
The screenshot shows a code editor interface with the following details:

- Project Structure:** The left sidebar displays the project structure under "Kayak". Key components include "kafka-front-end", "routes", and "search.js".
- Code Editor:** The main area shows the content of `search.js`. The code uses ES6 syntax, including `async` and `await` keywords.
- Code Snippet:**

```
if(results.code == "200"){
    var vendorArr=results.value;
    asyncLoop(vendorArr, function (vendor, next)
    {
        axios.get(vendor.vendorapi,
            {
                params: {
                    "data": data.searchquery
                }
            }).then(function(response) {
        search_results.push.apply(search_results,response.data.api_results);
        // console.log("car results after push",search_results)
        next();
    }).catch(function(err) {
        callback(err, null);
    });
}, function (err)
{
    if (err)
    {
        callback(err, null);
    }
    else {
        callback(null, search_results);
    }
});
}
else {
    callback(null, null);
}

});
```
- Toolbars and Status Bar:** The bottom of the screen features a toolbar with icons for Version Control, Terminal, and TODO. The status bar at the bottom right shows "Event Log", "1:1 LF: UTF-8: Git: master", and other system information.

Sign out Page (UI):



Sign out Page (Node Backend):

A screenshot of an IDE (Eclipse) showing the 'car.js' file from a project named 'kafka-front-end'. The code handles a POST request to '/book'. It checks if the response code is 200; if so, it sends a success message with 'status: 200' and 'api_results: results.value'. If not, it sends a failure message with 'status: 401'. Below the code, a terminal window shows a successful API call with a response object containing user data and a session destroy message. The bottom status bar indicates the file is 1:1, LF, UTF-8, and part of the master branch.

Sign out Page (Kafka Backend):

The screenshot shows an IDE interface with the project structure for 'kafka-front-end' selected. The file 'car.js' is open in the editor. The code handles a POST request to '/book'. It checks the response code, sends a success or error message to the client, and then logs the request body. A database query is shown in the terminal, returning a user's profile information. The response topic includes correlation ID, user data, and a success message.

```
else {
    if(results.code == "200"){
        res.send({status: 200 , 'api_results' : results.value});
    }
    else {
        res.send({status: 401});
    }
}

router.post('/book', function (req, res) {
    //console.log(req.body);
    var queueName = "BookCar";
    var reqObject = {
        email : req.session.email,
        ...
    };
    ...
});
```

```
DB Results:--  
inside the else of the get data  
*****  
[ RowDataPacket {  
  first_name: 'Sunil',  
  last_name: 'Tiwari',  
  user_role: 'USER',  
  city: '',  
  state: '',  
  zip_code: '95126',  
  profile_image_path: 'Sunil28071987@gmail.com386468_10150432773574298_212500349_n.jpg',  
  email: 'Sunil28071987@gmail.com',  
  phone: '6692137162',  
  street_address: '754 The Alameda',  
  credit_card_number: '123456789' } ]  
*****  
{ topic: 'response_topic',  
  messages: [{correlationId:'7c3bb03f424aa0ee6f425941226358d', "data": {"code":"200", "value": "user data success fully fetched", "data": [{"first_name": "Sunil", "last_name": "Tiwari", "user_role": "USER", "city": "", "state": "", "zip_code": "95126", "profile_image_path": "Sunil28071987@gmail.com386468_10150432773574298_212500349_n.jpg", "email": "Sunil28071987@gmail.com", "phone": "6692137162", "street_address": "754 The Alameda", "credit_card_number": "123456789"}]}},  
  partition: 0 } ]  
*****  
{ response_topic: { '0': 319 } }
```

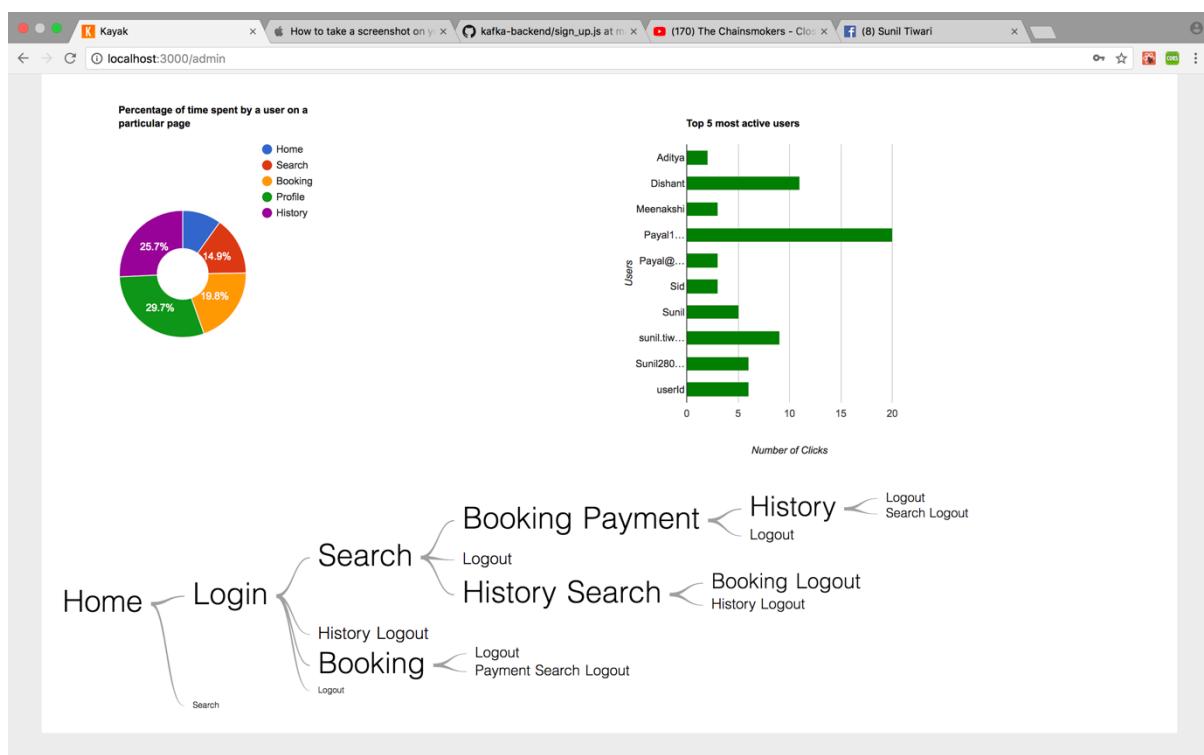
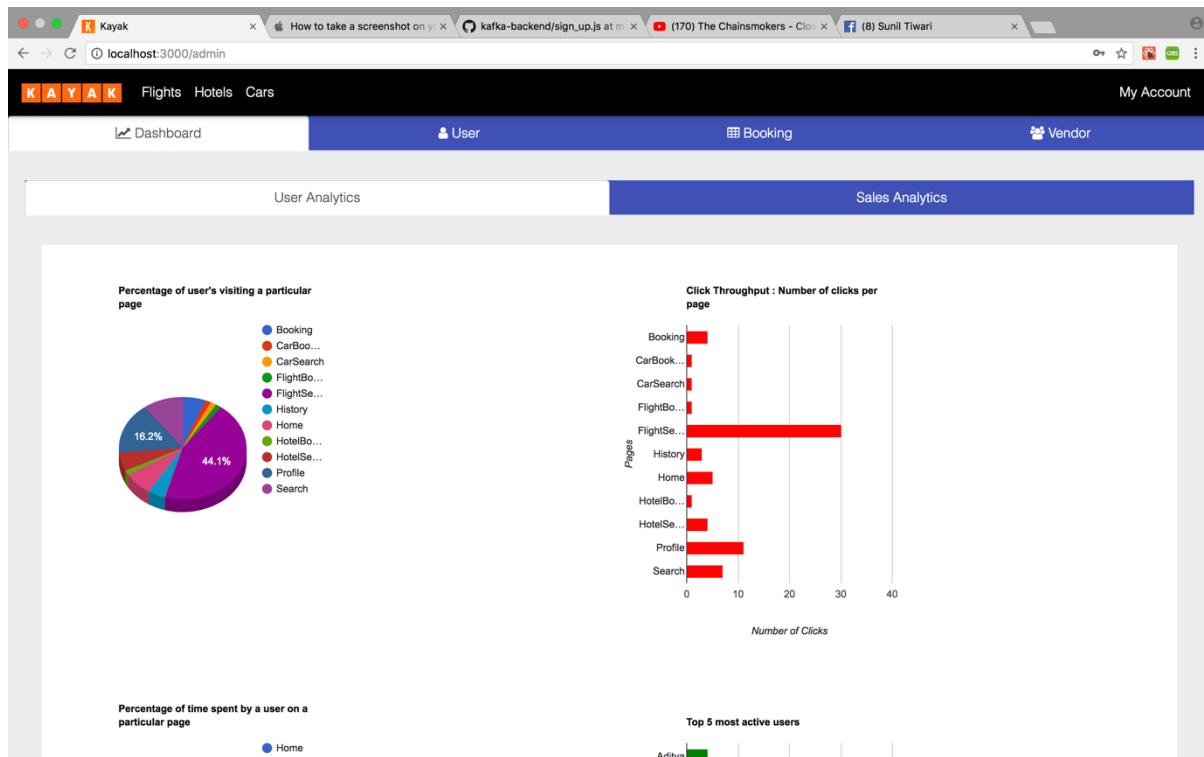
Sign out Page (Business Logic):

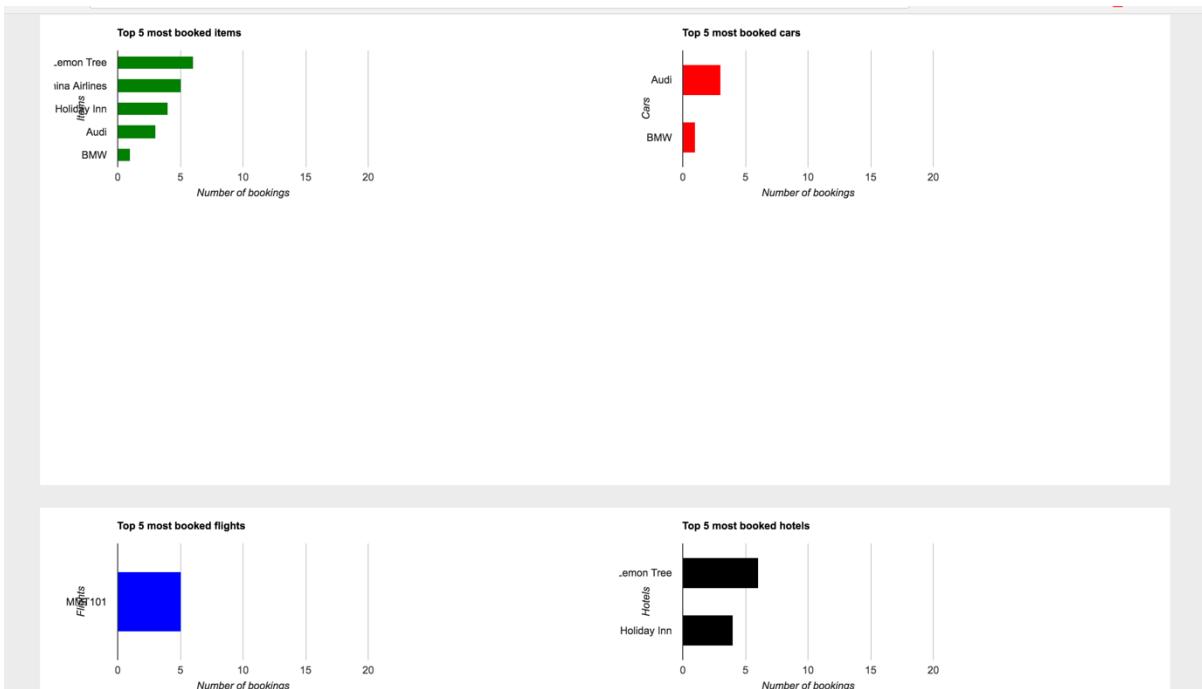
The screenshot shows an IDE interface with the project structure for 'kafka-front-end' selected. The file 'user.js' is open in the editor. The code handles a POST request to '/logout'. It invalidates the session and sends a success message. A database query is shown in the terminal, returning a user's booking history. The response includes the user's name and a success message.

```
//Logout the user - invalidate the session
router.post('/logout', function (req, res) {
    req.session.destroy();
    console.log('Session destroyed');
    res.status(201).send({status:201,"value": "logout successfull"});
});
```

```
i am here  
Everything successfull  
GET /user/bookinghistory 200 16.614 ms - 3307
```

Admin Dashboard Page (UI):





Admin Dashboard Page (Node Backend):

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "Kayak" and contains "kafka-front-end" and "kafka-back-end" sub-projects. "kafka-front-end" includes "node_modules" (library root), "public", and "routes" (containing "kafka", "passport", "admin.js", "analytics.js", "car.js", "flight.js", "hotel.js", "index.js", "mail.js", "passport.js", "search.js", and "user.js").
- Code Editor:** The "user.js" file is open, showing code related to user authentication and booking history retrieval.
- Terminal:** The terminal shows three POST requests to "/analytics/getchart" with response codes 200 and execution times (131.042 ms, 85.508 ms, 90.092 ms).
- Event Log:** Shows a warning about using 'var' instead of 'let' or 'const'.

Admin Dashboard Page (Kafka Backend):

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "Kayak" and contains "kafka-front-end" and "kafka-back-end" sub-projects. "kafka-front-end" includes "node_modules" (library root), "public", and "routes" (containing "kafka", "passport", "admin.js", "analytics.js", "car.js", "flight.js", "hotel.js", "index.js", "mail.js", "passport.js", "search.js", and "user.js").
- Code Editor:** The "user.js" file is open, showing code related to user authentication and booking history retrieval.
- Terminal:** The terminal shows a connection error ("err is null") and a successful DB query response ("DB Results").
- Event Log:** Shows a warning about using 'var' instead of 'let' or 'const'.

Admin Dashboard Page (Business Logic):

The screenshot shows a code editor interface with the following details:

- Project Structure:** The project is named "Kayak" and contains subfolders like "kafka-front-end", "routes", and "analytics.js".
- Code Editor:** The main pane displays the content of the "analytics.js" file.
- Code Content:** The file contains two main sections of code, each enclosed in a multi-line comment block. The first section handles a POST request to "/clicktracker" and the second handles a POST request to "/getchart". Both sections include logging of the request body, sending a Kafka message, and then returning a response based on the Kafka result.
- Toolbars and Status:** The bottom of the screen shows standard developer tools like Version Control, Terminal, and TODO, along with status information like "Event Log", "1:1 LF", "UTF-8", and "Git: master".

```
//*****  
router.post('/clicktracker', function (req, res) {  
    // var reqEmail = req.body.email;  
    // var reqPassword = req.body.password;  
  
    console.log("Inside app.js")  
    console.log("Request Body : ",req.body)  
    kafka.make_request('click_tracker_req',req.body, function(err,results){  
        if(err){  
            console.log(err);  
            res.status(400).json({status:400});  
        }  
        else {  
            if(results.code == 200){  
                res.status(200).json({status:200,result:results.value});  
            }  
            else {  
                res.status(400).json({status:400,result:results.value});  
            }  
        }  
    });  
    //*****  
}  
  
router.post('/getchart', function (req, res) {  
    //*****  
    console.log("Inside app.js")  
    console.log("Request Body : ",req.body)  
    kafka.make_request("get_chart_req",req.body, function(err,results){  
        if(err){  
            console.log(err);  
        }  
        else {  
            if(results.code == 200){  
                console.log("=====");  
                console.log(results.results);  
                console.log("=====");  
                res.status(200).json({status:200,data:results.results});  
            }  
            else {  
                res.status(400).json({status:400,data:results.results});  
            }  
        }  
    });  
    //*****  
}
```

Admin User Page (UI):

The screenshot shows a web browser window with multiple tabs open. The active tab displays an admin interface for managing users. The page has a header with the Kayak logo and navigation links for Flights, Hotels, Cars, User, Booking, and Vendor. A 'My Account' link is also present. Below the header is a table listing two users:

User Id	Name	Email	Phone Number	Address	Pincode
1	Payal	Payal1@gmail.com	6692137162	754 The Alameda	95126
2	Sunil	Sunil28071987@gmail.com	6692137162	754 The Alameda	95126

Each row in the table includes a small trash can icon for deleting the user.

Admin User Page (Node Backend):

The screenshot shows a code editor interface with several files open in tabs at the top: Welcome.js, nav.js, profile.js, flightsearch.js, flight.js, hotel.js, car.js, user.js, analytics.js, and search.js. The main editor area contains the 'analytics.js' file, which includes the following code:

```
router.post('/clicktracker', function (req, res) {
  // var reqEmail = req.body.email;
  // var reqPassword = req.body.password;
  console.log("Inside app.js");
  console.log("Request Body : ",req.body);
  kafka.makeRequest('click_tracker_req',req.body, function(err,results){
    if(err)
      console.log(err);
      res.status(400).json({status:400});
    else
      {
        if(results.code == 200){
          res.status(200).json({status:200,result:results.value});
        }
      }
  });
});
```

Below the code editor is a terminal window showing the following command and its output:

```
data:
[ { first_name: '',
  last_name: '',
  email: '1@1@gmail.com',
  phone: '',
  street_address: '',
  zip_code: '' },
{ first_name: 'Payal',
  last_name: 'Tiwari',
  email: 'Payal1@gmail.com',
  phone: '6692137162',
  street_address: '754 The Alameda',
  zip_code: '95126' },
{ first_name: '',
  last_name: '',
  email: 'Payal1@gmail.com',
  phone: '',
  street_address: '',
  zip_code: '' },
{ first_name: 'Sunil',
  last_name: 'Tiwari',
  email: 'sunil.tiwari@sjtu.edu',
  phone: '6692137162',
  street_address: '754 The Alameda',
  zip_code: '95126' },
```

The terminal also displays some linting errors: 'var used instead of let or const'.

Admin User Page (Kafka Backend):

The screenshot shows an IDE interface with the following details:

- Project Explorer:** Shows the project structure under "kafka-front-end" with files like Welcome.js, nav.js, profile.js, flightsearch.js, flight.js, hotel.js, car.js, user.js, analytics.js, and search.js.
- Code Editor:** Displays a file named "analytics.js". The code handles a POST request to "/clicktracker". It logs the request body, makes a Kafka request, and then sends a response based on the result. A SQL query is also present in the code.
- Terminal:** Shows the output of a command, likely a Kafka consumer or producer log, displaying messages from a topic named "response_topic". The log includes fields like correlationId, offset, partition, key, and value.
- Status Bar:** Shows the event log, file encoding (UTF-8), and git status (master).

Admin User Page (Business Logic):

The screenshot shows an IDE interface with the following details:

- Project Explorer:** Shows the project structure under "kafka-front-end" with files like nav.js, profile.js, flightsearch.js, flight.js, hotel.js, car.js, user.js, admin.js, analytics.js, and search.js.
- Code Editor:** Displays a file named "admin.js". It contains logic for handling GET and POST requests to "/getallusers" and "/addvendor" respectively. It uses the express framework and interacts with a Kafka client to perform operations.
- Status Bar:** Shows the event log, file encoding (UTF-8), and git status (master).

Admin Bookings Page (UI):

Booking					
Booking Id	Booking Type	Price	User	Credit Card	Date
248	FLIGHT	\$200	adi@gmail.com	1234432112344321	2017-12-04
249	FLIGHT	\$200	adi@gmail.com	2999999999999999	2017-12-04
250	HOTEL	\$6160	adi@gmail.com	2999999999999999	2017-12-04
251	CAR	\$1710	adi@gmail.com	2999999999999999	2017-12-04
252	FLIGHT	\$200	adi@gmail.com	2131313131313131	2017-12-04
253	HOTEL	\$6160	adi@gmail.com	1241241241241241	2017-12-04
254	FLIGHT	\$200	adi@gmail.com	1241241241241241	2017-12-04
255	HOTEL	\$36000	adi@gmail.com	1241241241241241	2017-12-04
256	HOTEL	\$480	adi@gmail.com	1241241241241241	2017-12-04

Admin Bookings Page (Node Backend):

The screenshot shows a Java project structure in the Project Explorer on the left, with a file named `user.js` open in the code editor. The code is a Node.js script for a Kafka consumer. The code editor has syntax highlighting for JavaScript and includes annotations from a static code analysis tool, such as 'var' used instead of 'let' or 'const'. The terminal below shows the execution of the application, which returns a JSON object and ends with a 200 status code and a response time of 26.085 ms.

```
Project ~ /Desktop/KayakFinal/Kayak
  - Kayak
    - database
    - kafka-back-end
    - kafka-front-end
    - .idea
    - bin
    - node_modules library root
      - routes
        - kafka
        - passport
          - admin.js
          - analytics.js
          - car.js
          - flight.js
          - hotel.js
          - index.js
    - public
    - routes
    - search.js

Welcome.js nav.js profile.js flightsearch.js flight.js hotel.js car.js user.js analytics.js

157 //*****
158 //*****
159 //*****
160 //*****
161 router.put('/update',function (req,res) {
162   console.log("Inside the update path");
163   console.log("*****");
164   console.log(req.body);
165   console.log("*****");
166   var payload = req.body;
167   payload['user_email']=req.session.email;
168   console.log(payload.user_email);
169   kafka.make_request('update', payload ,function(err,results){
170     if(err){
171       console.log("After kafka response");
172       done(err,{});
173       res.send({"status":401})
174     }
175   })
176 }

Terminal
Local Zookeeper KafkaServer ReactFrontEnd NodeBackend KafkaBackend Topics MongoD
+ source_street: '101 E San Fernando St.',
  source_country: 'USA',
  source_zipcode: '',
  destination_city: 'San Jose',
  destination_state: 'California',
  destination_street: '101 E San Fernando St.',
  destination_country: 'USA',
  destination_zipcode: '',
  source_airport: '',
  destination_airport: '',
  return_source_airport: '',
  return_destination_airport: '',
  booking_start_date: '2017-11-04T07:00:00.000Z',
  booking_end_date: '2017-11-06T08:00:00.000Z',
  return_booking_start_date: null,
  return_booking_end_date: null,
  target_count: null,
  booking_class: null,
  credit_card_type: 'undefined',
  credit_card_number: '123456789',
  credit_card_holder_name: 'undefined',
  credit_card_valid_from: 'undefined',
  credit_card_valid_till: '2017-12' ] }

Returning results ----[object Object]
GET /admin/bills 200 26.085 ms - 3232
Event Log
```

Admin Bookings Page (Kafka Backend):

The screenshot shows a Java IDE interface with the following details:

- Project Explorer:** Shows the project structure under "Kayak".
- Terminal:** Displays the command `credit_card_valid_from` followed by several lines of JSON-like configuration data.
- Code Editor:** The file `user.js` is open, showing a Node.js script. It includes a UML-style sequence diagram for a Kafka update request. The code handles errors and sends a response with status 401 if an error occurs during the Kafka request.
- Search Bar:** At the top, there is a search bar with the placeholder "Search" and a magnifying glass icon.

Admin Bookings Page (Business Logic):

Admin Vendor Page (UI):

The screenshot displays the Admin Vendor Page interface. At the top, there's a navigation bar with tabs for Dashboard, User, Booking, and Vendor. The Vendor tab is active. On the left, a table lists vendors with their names, service types, and API endpoints. On the right, a form allows adding new vendors with fields for Vendor Name, Vendor EndPoint, Vendor Email, Vendor Schema, and Type (Flight selected). Below this, a large form for adding flights includes fields for Flight Operator, Flight Id, Economy Price, First Class Price, Business Price, arrival and departure dates/times, and origin/destination fields. A prominent blue "ADD FLIGHT" button is at the bottom.

Vendor Name	Type of Service	Vendor API
MMT	car	http://localhost:3001/car/getcars?vendor=MMTCars
Cleartrip	car	http://localhost:3001/car/getcars?vendor=CleartripCars
Alamo	car	http://localhost:3001/car/getcars?vendor=AlamoCars
MMT	hotel	http://localhost:3001/hotel/gethotels?vendor=MMTHotels
Cleartrip	hotel	http://localhost:3001/hotel/gethotels?vendor=CleartripHotels
TripAdvisor	hotel	http://localhost:3001/hotel/gethotels?vendor=TripAdvisorHotels
MMT	flight	http://localhost:3001/flight/getflights?vendor=MMTFlights
Cleartrip	flight	http://localhost:3001/flight/getflights?vendor=CleartripFlights
Expedia	flight	http://localhost:3001/flight/getflights?vendor=ExpediaFlights

Add Vendor

Vendor Name:

Vendor EndPoint:

Vendor Email:

Vendor Schema:

Type:

Flight

ADD

Flight **Car** **Hotel**

Flight Operator **Flight Id**

Economy Price **First Class Price** **Business Price**

Arrival Day: **Departure:**

Arrival : **Departure:**

From **To** **From** **To**

ADD FLIGHT

Vendor Details

No.	Flight ID	Operator
1	MMT100	Lufthansa Airlines
2	MMT101	Air China Airlines

Admin Vendor Page (Node Backend):

```

router.post('/deletevendor',function (req,res) {
  console.log(req.body);
  kafka.make_request('deletevendor', req.body, function(err,results){
    res.send({'status': results.code});
  });
});
  
```

The terminal shows the execution of the code:

```

email: 'CleartripHotels' },
{ vendorid: 6,
  vendorname: 'TripAdvisor',
  servicetype: 'hotel',
  vendorapi: 'http://localhost:3001/hotel/gethotels?vendor=TripAdvisorHotels',
  model: 'tria@gmail.com',
  email: 'TripAdvisorHotels' },
{ vendorid: 7,
  vendorname: 'MMT',
  servicetype: 'flight',
  vendorapi: 'http://localhost:3001/flight/getflights?vendor=MmtFlights',
  model: 'mmt@gmail.com',
  email: 'MmtFlights' },
{ vendorid: 8,
  vendorname: 'Cleartrip',
  servicetype: 'flight',
  vendorapi: 'http://localhost:3001/flight/getflights?vendor=CleartripFlights',
  model: 'cct@gmail.com',
  email: 'CleartripFlights' },
{ vendorid: 9,
  vendorname: 'Expedia',
  servicetype: 'flight',
  vendorapi: 'http://localhost:3001/flight/getflights?vendor=ExpediaFlights',
  model: 'expedia@gmail.com',
  email: 'ExpediaFlights' } ]
Returning results ----[object Object]
GET /admin/vendors 200 27.804 ms - 1635
  
```

Admin Vendor Page (Kafka Backend):

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "Kayak" and contains "kafka-front-end" and "kafka-back-end". The "kafka-back-end" folder includes "routes" and "admin.js".
- Code Editor:** The "admin.js" file is open. It contains logic for handling a POST request to "/deletevendor". The code uses the "kafka" module to make a request to "deletevendor".
- Terminal:** The terminal tab shows a command-line session with several vendor data entries. Each entry consists of a vendor ID, name, service type, vendor API URL, and email.
- Event Log:** The event log shows a message indicating 364 responses.
- Bottom Status Bar:** Shows "1:1 LF: UTF-8 Git: master".

```
const kafka = require('node-kafka');
const express = require('express');
const router = express.Router();
const bodyParser = require('body-parser');

router.post('/deletevendor', function (req, res) {
    console.log(req.body);
    kafka.make_request('deletevendor', req.body, function(err, results){
        if(err)
            res.send({status: err.code, message: err.message});
        else
            if(results.code == "200")
                res.send({status: results.code});
            else
                res.send({status: results.code});
    });
});

module.exports = router;
```

Admin Vendor Page (Business Logic):

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "Kayak" and contains "kafka-front-end" and "kafka-back-end". The "kafka-back-end" folder includes "routes" and "admin.js".
- Code Editor:** The "admin.js" file is open. It contains logic for handling POST and GET requests to "/deletevendor" and "/vendors". The code uses the "kafka" module to make requests to "deletevendor" and "getvendors".
- Terminal:** The terminal tab shows a command-line session with several vendor data entries. Each entry consists of a vendor ID, name, service type, vendor API URL, and email.
- Event Log:** The event log shows a message indicating 364 responses.
- Bottom Status Bar:** Shows "1:1 LF: UTF-8 Git: master".

```
const kafka = require('node-kafka');
const express = require('express');
const router = express.Router();
const bodyParser = require('body-parser');

router.post('/deletevendor', function (req, res) {
    console.log(req.body);
    kafka.make_request('deletevendor', req.body, function(err, results){
        if(err)
            res.send({status: err.code, message: err.message});
        else
            if(results.code == "200")
                res.send({status: results.code});
            else
                res.send({status: results.code});
    });
});

router.get('/vendors', function (req, res) {
    kafka.make_request('getvendors', "", function(err, results){
        if(err)
            res.send({status: err.code, message: err.message});
        else
            if(results.code == "200")
                res.send({status: results.code, vendors: results.value});
            else
                res.send({status: results.code});
    });
});

module.exports = router;
```

Performance (Test Cases):

Performance with Kafka queueing is excellent. Kafka is a distributed and parallel processing architecture provides seamless asynchronous API handling experience to provide a faster performance.

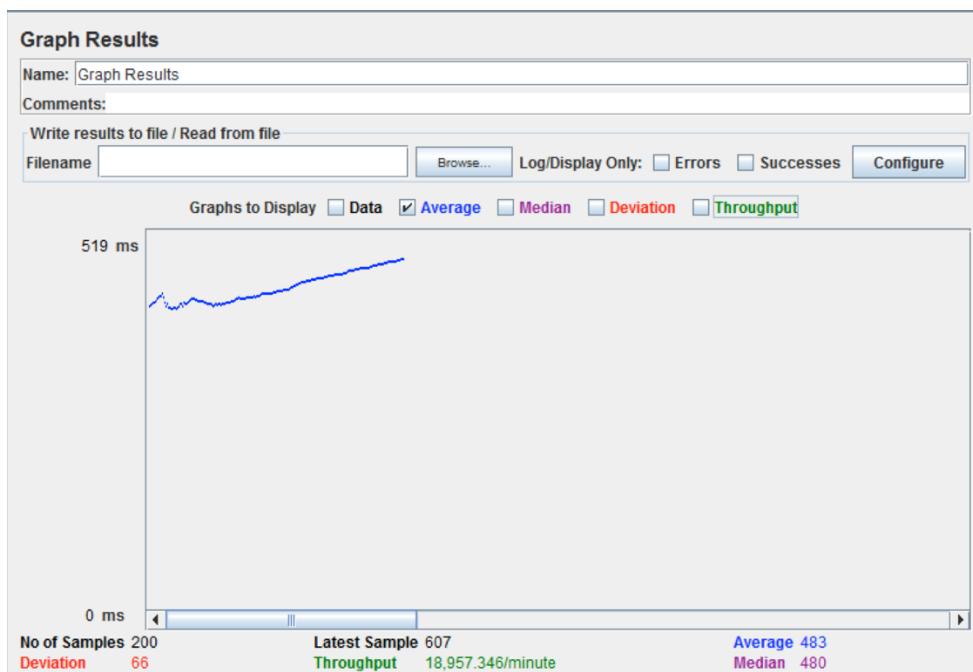
Analysis:

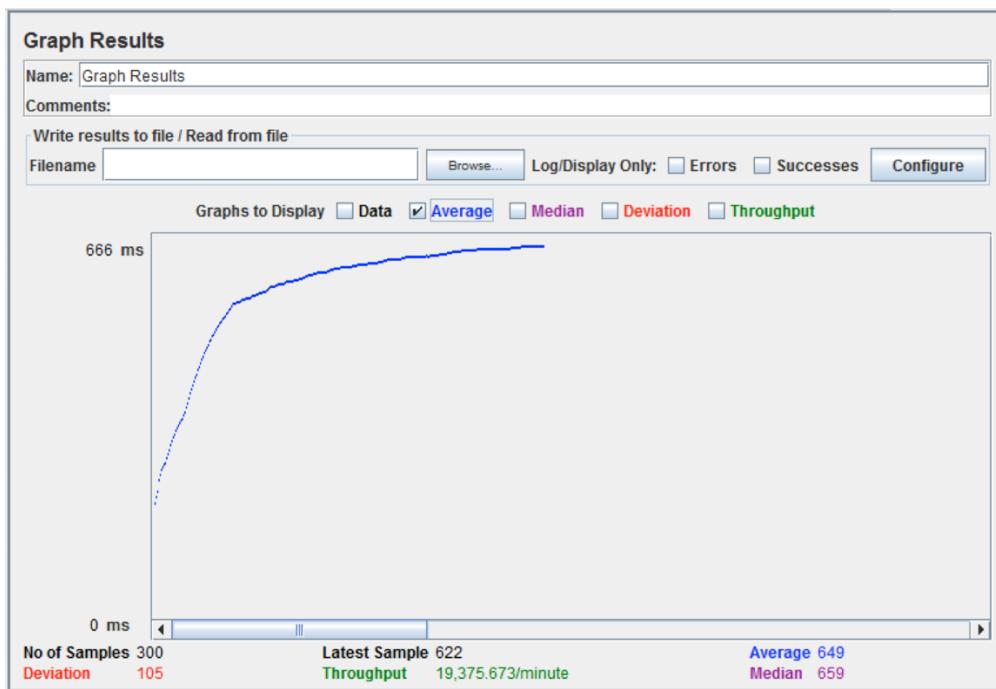
Kafka's performances analysis as per the following testing tells us that Kafka is good at handling large number of requests. Initially, when the requests increase time for service increases, but soon as the request become as large as thousands, it stabilizes. This tells us that our architecture is designed to handle large number of requests with same throughput.

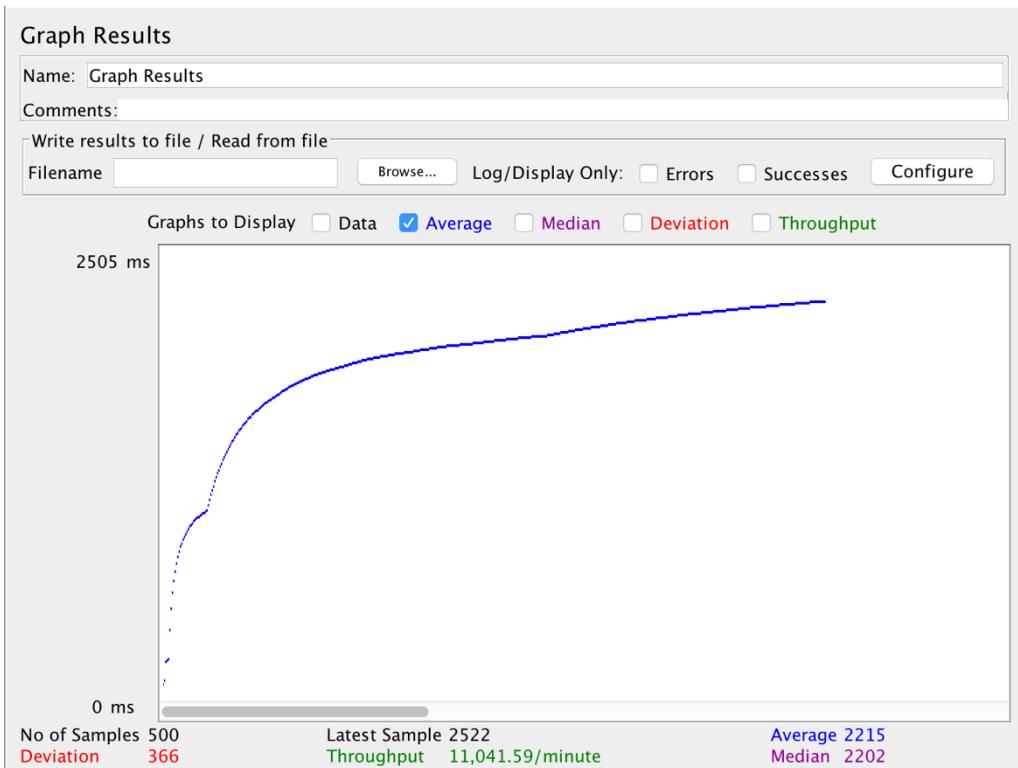
We could have achieved better performance by

- More number of producer partitions
- Increased number of consumers in a consumer group

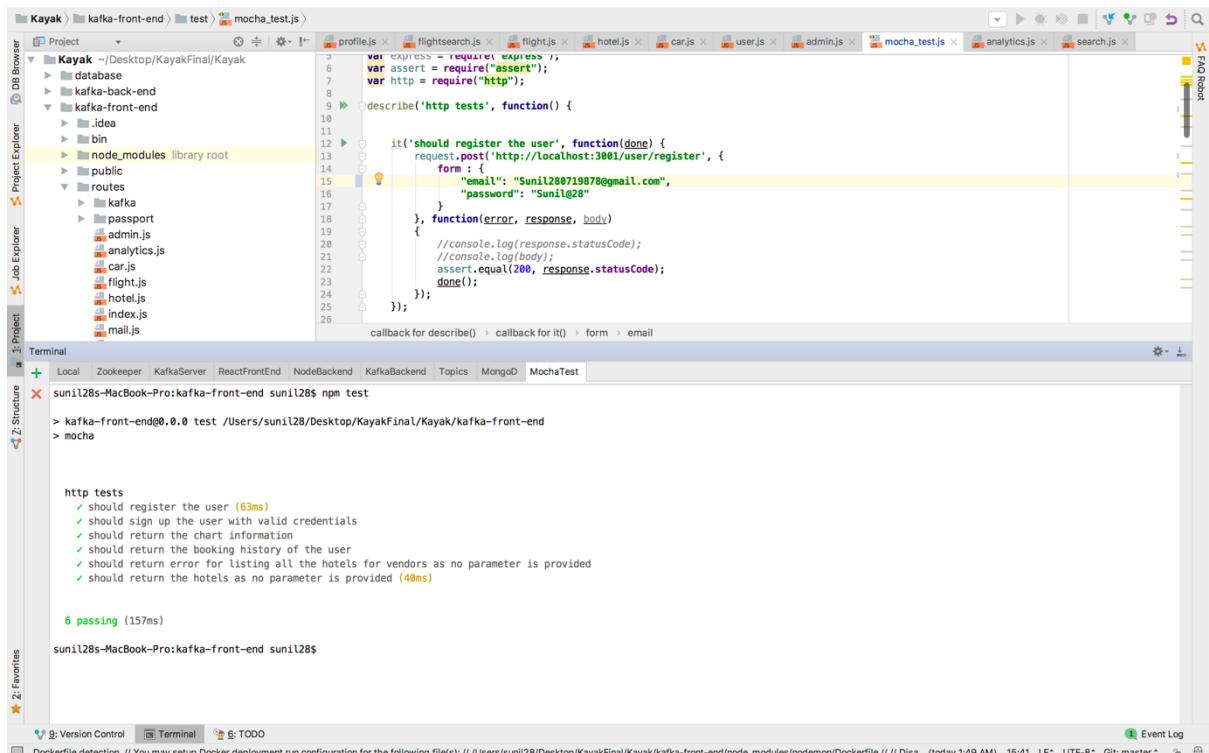
JMeter Test Case







Mocha Test Cases



```
var express = require('express');
var assert = require("assert");
var http = require("http");

describe('http tests', function() {
    it('should register the user', function(done) {
        request.post('http://localhost:3001/user/register', {
            form : {
                "email": "Sunil28071987@gmail.com",
                "password": "Sunil@28"
            }
        }, function(error, response, body) {
            //console.log(response.statusCode);
            //console.log(body);
            assert.equal(200, response.statusCode);
            done();
        });
    });
});

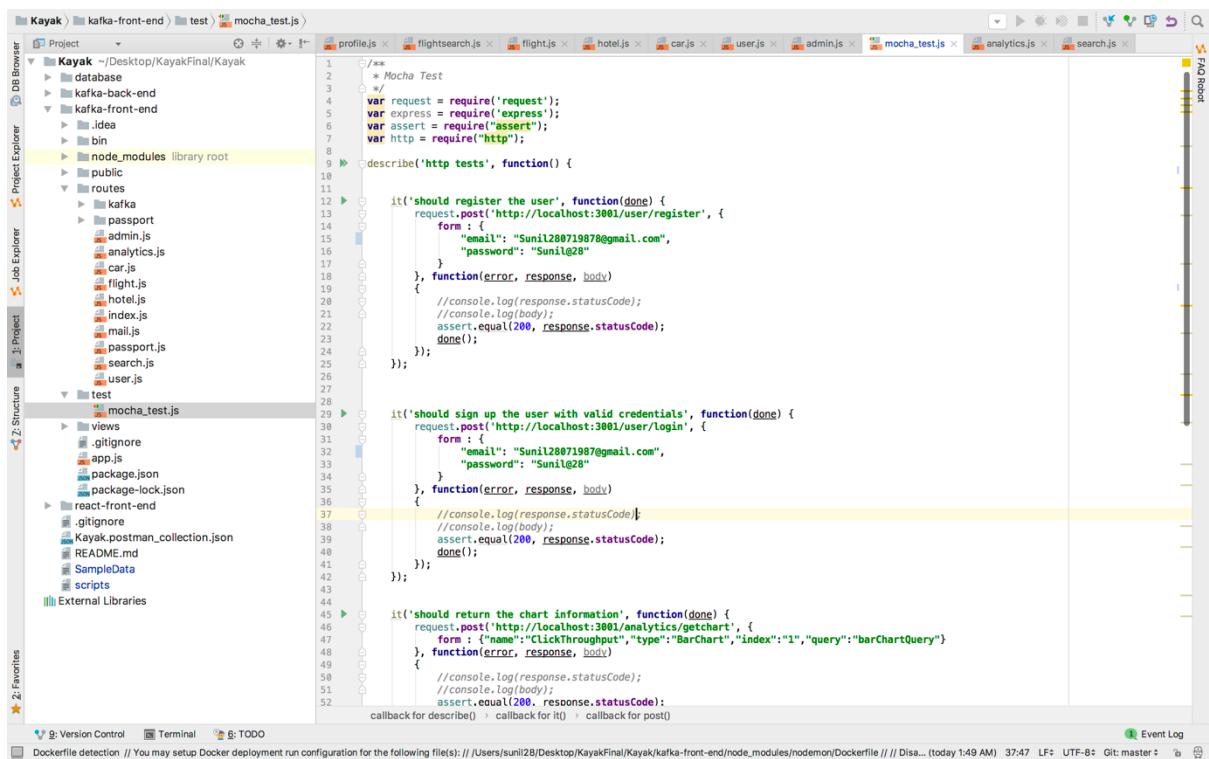
callback for describe() > callback for it() > form > email
```

```
sunil28s-MacBook-Pro:kafka-front-end sunil28$ npm test
> kafka-front-end@0.0.0 test /Users/sunil28/Desktop/KayakFinal/Kayak/kafka-front-end
> mocha

http tests
  ✓ should register the user (63ms)
  ✓ should sign up the user with valid credentials
  ✓ should return the chart information
  ✓ should return the booking history of the user
  ✓ should return error for listing all the hotels for vendors as no parameter is provided
  ✓ should return the hotels as no parameter is provided (40ms)

  6 passing (157ms)

sunil28s-MacBook-Pro:kafka-front-end sunil28$
```



```
/* Mocha Test */
var request = require('request');
var express = require('express');
var assert = require("assert");
var http = require("http");

describe('http tests', function() {
    it('should register the user', function(done) {
        request.post('http://localhost:3001/user/register', {
            form : {
                "email": "Sunil28071987@gmail.com",
                "password": "Sunil@28"
            }
        }, function(error, response, body) {
            //console.log(response.statusCode);
            //console.log(body);
            assert.equal(200, response.statusCode);
            done();
        });
    });

    it('should sign up the user with valid credentials', function(done) {
        request.post('http://localhost:3001/user/login', {
            form : {
                "email": "Sunil28071987@gmail.com",
                "password": "Sunil@28"
            }
        }, function(error, response, body) {
            //console.log(response.statusCode);
            //console.log(body);
            assert.equal(200, response.statusCode);
            done();
        });
    });

    it('should return the chart information', function(done) {
        request.post('http://localhost:3001/analytics/getchart', {
            form : { "name": "Clickthrough", "type": "BarChart", "index": "1", "query": "barChartQuery" }
        }, function(error, response, body) {
            //console.log(response.statusCode);
            //console.log(body);
            assert.equal(200, response.statusCode);
            done();
        });
    });
});

callback for describe() > callback for it() > callback for post()
```

```
Dockerfile detection // You may setup Docker deployment run configuration for the following file(s): // /Users/sunil28/Desktop/KayakFinal/Kayak/kafka-front-end/node_modules/nodemon/Dockerfile // // Disa... (today 1:49 AM) 37:47 LF+ UTF-8+ Git: master
```

```
41     });
42   });
43   });
44   });
45   });
46   });
47   });
48   });
49   });
50   });
51   });
52   });
53   });
54   });
55   });
56   });
57   });
58   });
59   });
60   });
61   });
62   });
63   });
64   });
65   });
66   });
67   });
68   });
69   });
70   });
71   });
72   });
73   });
74   });
75   });
76   });
77   });
78   });
79   });
80   });
81   });
82   });
83   });
84   });
85   });
86   });
87   });

callback for describe() -> callback for it() -> callback for post()
```

Observations & Lessons:

It was first time for the entire team when we created an entire full stack end to end application. Kayak was a large system and we learned following lessons.

1. Designing user interface for a large system like Kayak.
2. Session & State management for large system like Kayak.
3. Designing a large system with various small sub components (React, Redux, Database, Kafka, API's) and handling interaction among them.
4. Creating and managing large number of API calls in NodeJS.
5. Creating, sending and receiving and handling distributed messaging between clients and server.
6. Handling secure user authentication using various strategies with the help of PassportJS libraries.
7. Testing the REST API's using various testing tools such as JMeter and MochaJS.
8. Caching SQL queries to improve the performance and throughput of the system.
9. Tracking the user activity through the UI and create a logging mechanism to save all that information in the database.
10. Drawing the meaningful insights from the logged information and make decisions based on user actions.