Faculty of Computers and Information
"Helwan University"

Time For Travel Project Report

SUBMITTED BY:

Shenouda Farouk
Lamees mohamed
Ahmed Hassanein
Marina Saleh
Eman Mohamed
Maryan Fathy

SUBMITTED TO:

Dr/ Laia Abd EL-Hameed

TimeTravel

Project Contribution:

1. Lamees Mohamed

- Front-end for whole admin module with functionality to add, edit and view hotels, flights, cars(listings and bookings), users, hosts with filtering functionality
- Backend for admin module with functionality to add, edit, view hotels, flights, cars (listings and bookings), , users, hosts with filtering functionality
- Admin module validations

2. Shenouda Farouk

- Frontend for hotel listing, flight listing, car listing with filtering functionality for hotels, flights, cars based on appropriate inputs.
- Backend for hotel listing, flight listing, car listing with filtering functionality for hotels, flights, cars based on appropriate inputs.
- Redis caching
- JMeter testing

3. Marina Saleh $ Maryan Fathy

- Database design for MySQL and MongoDB Models.
- Admin dashboard analytics charts using user activity logs and processed them using elastic search.
- Frontend for admin dashboard
- Backend for admin dashboard
- Validations for Admin module

4. Ahmed Hassanein

- Frontend for hotel detail page, user profile with view, edit functionality for preferences, managing credit card details and trip history.
- Backend hotel detail page, user profile with view,edit functionality for preferences, managing credit card details and trip history.

5. Eman Mohamed

- Frontend for hotel - flight - car booking with respective payment and billing details, sign in, sign up.
- Backend for hotel - flight - car booking with respective payment and billing details, sign in, sign up.

Introduction

The objective of the project is to develop 'KAYAK' prototype. KAYAK is a travel metasearch engine, which enables users to search and book hotels, flights, and cars. KAYAK's other services include packages, rentals, cruises, guides, trains, flight tracker, routes, and deals. The application compares the prices with other websites and views the best deal.

Users are able to create (sign-up) an account. In order to view/manage their booking they must login and click account preferences under the Account tab. They have the privilege to set up their own profile image which appears at the top-right corner. Users can view and edit all their personal details under the preferences tab. This information include first name, last name, email, gender, address (street, city, state, zip code) , contact info and date of birth.

One can add and view all the credit/debit cards linked to their account. They may also view all of their past bookings (hotel | flights | cars).

We have included validations on every user input of the application. Server is tested for myriad concurrent users at a time. We have used Jmeter as a load testing tool for analyzing and measuring performance. Mocha, a javascript enabled framework serves the purpose testing the REST APIs.

Refer the requirement section for more info on implementation.

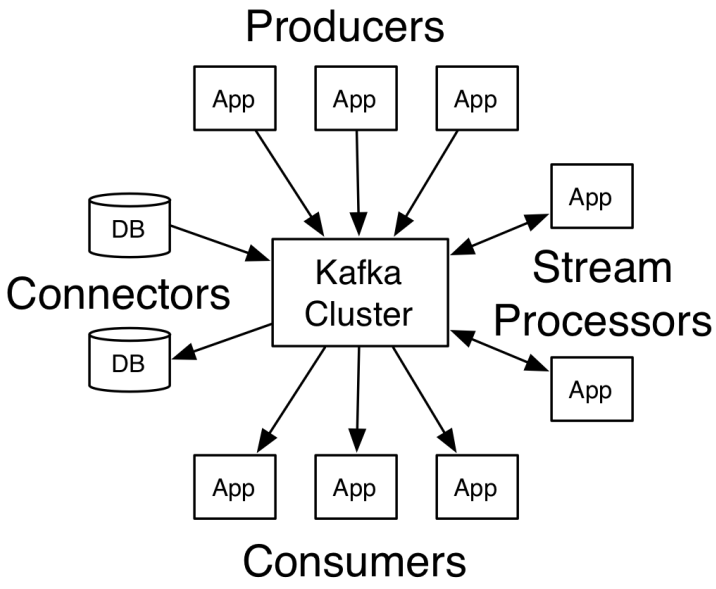
Purpose:

develop an online travel reservation system and use distributed publish-subscribe messaging system like Kafka to develop scalable, durable, reliable, and high-throughput distributed system. Moreover, we have used Redis for cache management.

System Design:

Technologies used for development of KAYAK prototype

Part of application	Technologies used
Front end	React.js We use React, the JS library for building our application's user interface. It enables us to create large webapps that change overtime w/o reloading page.
Back end	Node.js We have used Node.js for server-side JS execution and Express.js as a web application framework for Node.
Markup, UI/UX	HTML5, CSS3, bootstrap, react-strap Just like bootstrap, we have used React's in-built front-end component library, popularly known as react-strap. This is an open source toolkit for developing front-end along with other markup languages..

Database	SQL Structured Query Language is used to manage transactional data like 1 user credentials 2 user profile 3 booking details 4 traveller information 5 payment information 6 billing information Mongo MongoDB is used as a database to store all other records like 1 hotel details 2 flight details 3 car details 4 sessions
Password hashing	Bcrypt We have used Bcrypt as a password hashing function. It is one of the efficient encryption algorithms and resistant to brute force attacks
Authentication middleware	Passport.js We make use of an express-compatible middleware called Passport whose sole purpose is to authenticate requests which is accomplished by strategies.
Log generation	Winston We use versatile logging library named Winston to log messages to console.
Stream processing	Kafka We use Kafka as a streaming platform to build real-time streaming data pipelines that reliably get data between applications. 
Data caching	Redis Redis is an in-memory data structure store used for cache. Depending on your use case, you can persist it either by dumping the dataset to disk every once in

	awhile, or by appending each command to a log
Scripting language	Javascript

Connection pooling

Large amount of time is spent in waiting to get the connection back from mysql, since we re-use a single connection again and again. This degrades the performance. Using a connection pool, we can reuse existing connections avoiding the cost of initiating a connection, parsing SQL etc. This helps server handle requests efficiently and send responses faster.

Validations

Validations are performed at various user inputs. These include

1. While hotel-booking:
    a. Enter appropriate city (must not be *blank*)
    b. Check-in date should not be after check-out date
    c. Enter valid number of people (must not be *blank*)
    d. Enter valid number of rooms (must not be *blank*)

2. While flight-booking:
    a. Selecting valid to & from airports (must not be *blank*)
    b. Mandatory selection of 'departure-date' for return booking
    c. Enter valid number of persons traveling
    d. Mandatory selection of flight class

3. While car-booing:
    a. Enter appropriate city (must not be *blank*)
    b. Pick-up date should not be afte drop-off date
    c. None of the two date fields must be blank

4. Pin code validation (length must *not be more than 5*)

"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:**

Respective SQL schemas are listed one-by-one.

USER.sql

```
DROP TABLE IF EXISTS `user`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `user` (
  `username` varchar(50) NOT NULL,
  `password` varchar(500) NOT NULL,
  `accessInd` varchar(10) NOT NULL,
  PRIMARY KEY (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

USERPROFILE.sql

```
DROP TABLE IF EXISTS `userprofile`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `userprofile` (
  `username` varchar(50) NOT NULL,
  `firstName` varchar(50) NOT NULL,
  `lastName` varchar(50) NOT NULL,
  `street` varchar(50) DEFAULT NULL,
  `city` varchar(50) DEFAULT NULL,
  `state` varchar(50) DEFAULT NULL,
  `zipCode` varchar(50) DEFAULT NULL,
  `phoneNumber` varchar(45) DEFAULT NULL,
  `profileImage` varchar(100) DEFAULT NULL,
  `dateofbirth` date DEFAULT NULL,
  `gender` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`username`),
  CONSTRAINT `username` FOREIGN KEY (`username`)
REFERENCES `user` (`username`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

HOTELBOOKING.sql

```
DROP TABLE IF EXISTS `hotelbooking`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `hotelbooking` (
  `bookingId` mediumint(10) NOT NULL AUTO_INCREMENT,
  `hotelId` varchar(100) NOT NULL,
  `noOfPeople` int(1) NOT NULL,
  `roomType` varchar(10) NOT NULL,
  `fromDate` date NOT NULL,
  `toDate` date NOT NULL,
  `ticketPrice` float NOT NULL,
  `totalAmount` float NOT NULL,
  `username` varchar(50) NOT NULL,
  `hostId` mediumint(6) NOT NULL,
  `bill_day` varchar(45) DEFAULT NULL,
  `bill_month` varchar(45) DEFAULT NULL,
  `bill_year` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`bookingId`),
  KEY `username_idx` (`username`),
  CONSTRAINT `hotelbooking_ibfk_1` FOREIGN KEY (`username`)
REFERENCES `user` (`username`) ON DELETE NO ACTION ON
UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=16 DEFAULT
CHARSET=utf8;
```

FLIGHTBOOKING.js

```
DROP TABLE IF EXISTS `flightbooking`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `flightbooking` (
```

```
  `bookingId` mediumint(10) NOT NULL AUTO_INCREMENT,
  `flightId` varchar(100) NOT NULL,
  `noOfPassengers` int(3) NOT NULL,
  `flightClass` varchar(50) NOT NULL,
  `tripType` varchar(10) NOT NULL,
  `fromDate` date NOT NULL,
  `toDate` date DEFAULT NULL,
  `ticketPrice` float NOT NULL,
  `totalAmount` float NOT NULL,
  `username` varchar(50) NOT NULL,
  `hostId` mediumint(6) NOT NULL,
  `bill_day` varchar(45) DEFAULT NULL,
  `bill_month` varchar(45) DEFAULT NULL,
  `bill_year` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`bookingId`),
  KEY `username_idx` (`username`),
  CONSTRAINT `flightbooking_ibfk_1` FOREIGN KEY (`username`)
REFERENCES `user` (`username`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=26 DEFAULT
CHARSET=utf8;
```

CARBOOKING.sql

```
DROP TABLE IF EXISTS `carbooking`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `carbooking` (
  `bookingId` mediumint(10) NOT NULL AUTO_INCREMENT,
  `carId` varchar(100) NOT NULL,
  `noOfDays` int(10) NOT NULL,
  `fromDate` date NOT NULL,
  `toDate` date NOT NULL,
  `ticketPrice` float NOT NULL,
  `totalAmount` float NOT NULL,
  `username` varchar(50) NOT NULL,
  `hostId` mediumint(6) NOT NULL,
  `bill_day` varchar(45) DEFAULT NULL,
```

```
  `bill_month` varchar(45) DEFAULT NULL,
  `bill_year` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`bookingId`),
  KEY `username_idx` (`username`),
  CONSTRAINT `username_hotelbooking` FOREIGN KEY
(`username`) REFERENCES `user` (`username`) ON DELETE NO
ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT
CHARSET=utf8;
```

TRAVELERDETAILS.sql

```
DROP TABLE IF EXISTS `travelerdetails`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `travelerdetails` (
  `travelerId` mediumint(10) NOT NULL AUTO_INCREMENT,
  `bookingtype` varchar(20) NOT NULL,
  `bookingId` mediumint(10) DEFAULT NULL,
  `firstname` varchar(50) DEFAULT NULL,
  `lastname` varchar(50) DEFAULT NULL,
  `email` varchar(50) DEFAULT NULL,
  `phonenumber` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`travelerId`)
) ENGINE=InnoDB AUTO_INCREMENT=50 DEFAULT
CHARSET=utf8;
```

PAYMENTDETAILS.sql

```
DROP TABLE IF EXISTS `paymentdetails`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `paymentdetails` (
  `username` varchar(50) NOT NULL,
```

```
  `nameoncard` varchar(50) NOT NULL,
  `creditCardNumber` varchar(45) NOT NULL,
  `validThrough` varchar(45) NOT NULL,
  `cvv` varchar(3) NOT NULL,
  PRIMARY KEY (`creditCardNumber`),
  KEY `username_idx` (`username`),
  CONSTRAINT `paymentdetails_ibfk_1` FOREIGN KEY (`username`)
REFERENCES `user` (`username`) ON DELETE CASCADE ON
UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

BILLINGADDRESS.sql

```
DROP TABLE IF EXISTS `billingaddress`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `billingaddress` (
  `username` varchar(50) NOT NULL,
  `street1` varchar(30) NOT NULL,
  `street2` varchar(30) DEFAULT NULL,
  `postalcode` varchar(10) NOT NULL,
  `city` varchar(20) DEFAULT NULL,
  `state` varchar(20) DEFAULT NULL,
  `country` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`username`,`street1`,`postalcode`),
  CONSTRAINT `billingaddress_ibfk_1` FOREIGN KEY (`username`)
REFERENCES `user` (`username`) ON DELETE NO ACTION ON
UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

HOST.sql

```
DROP TABLE IF EXISTS `host`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
```

```
CREATE TABLE `host` (
  `hostId` mediumint(9) NOT NULL AUTO_INCREMENT,
  `hostName` varchar(100) NOT NULL,
  `serviceType` enum('flight','hotel','car') NOT NULL,
  PRIMARY KEY (`hostId`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT
CHARSET=utf8;
```

We have managed the following information for the USER entity.

1 User ID
2 First Name
3 Last Name
4 Address
5 City
6 State
7 Zip Code
8 Phone number
9 Email (is the username of the user)
10 Trip ID (username maintained in respective booking tables)
11 Profile Image
12 Credit Card details

Similarly, the listing for all the three objects were maintained properly.

HOTELS	FLIGHTS	CARS
_id	_id	_id
hostId	flightNo	hostId
hotelName	hostId	carName
hotelAddress	flightOperator	carType
city	departureDate	carMake
state	arrivalDate	carModel
zipcode	departureTime	capacity

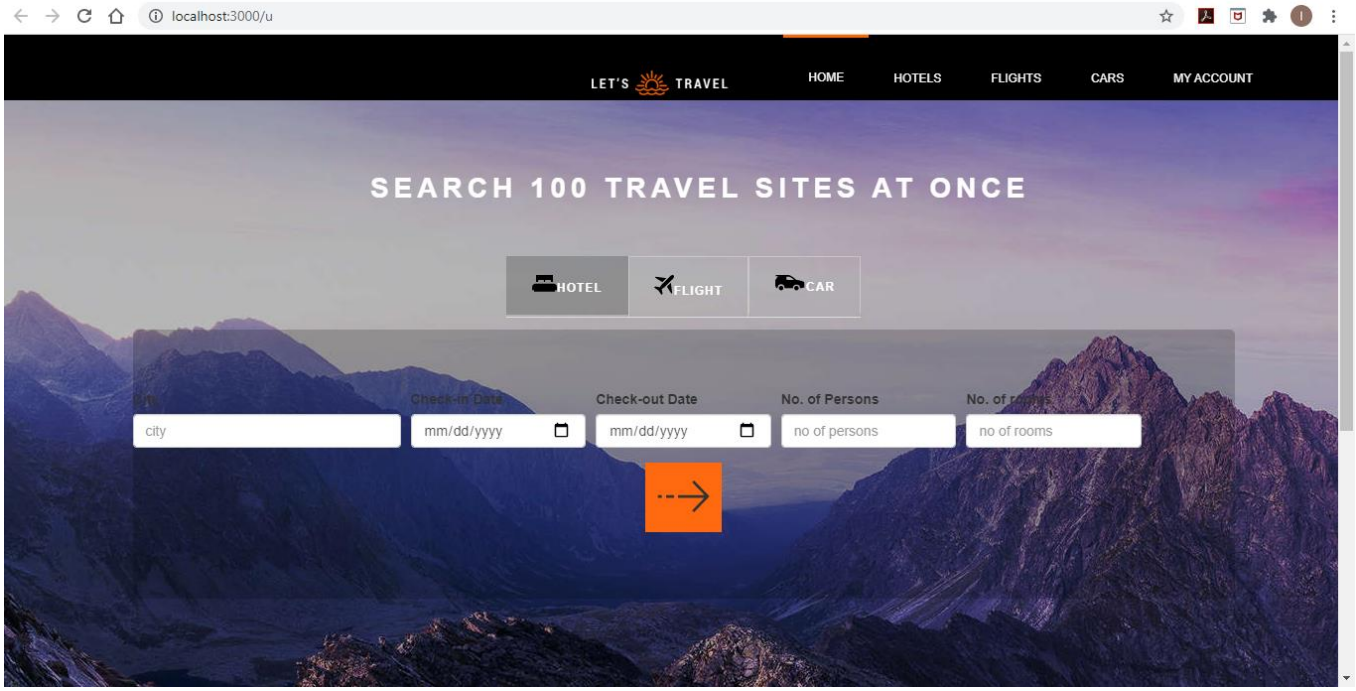
stars	arrivalTime	city
rooms _id noOfRooms roomPrice roomCapacity roomType	classes classType price noOfSeats	state
	duration	zipcode
	origin	price
	destination	

However, for the billing entity, we have maintained 3 different tables to record most important information.

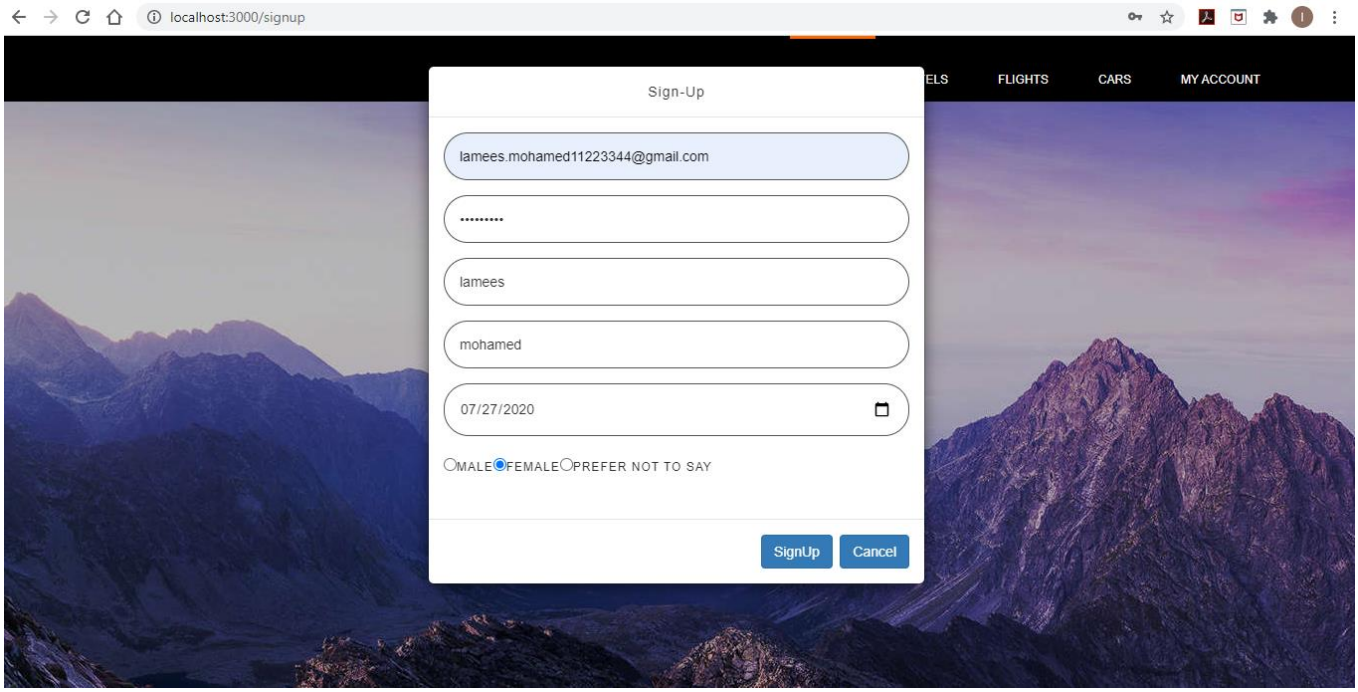
TRAVELERDETAILS	PAYMENTDETAILS	BILLINGADDRESS
travelerID	username	username
bookingType	nameoncard	street1
bookingId	creditCardNumber	street2
firstname	validThrough	postalcode
username	cvv	city
lastname		state
email		country
phonenumber		

Screenshots:

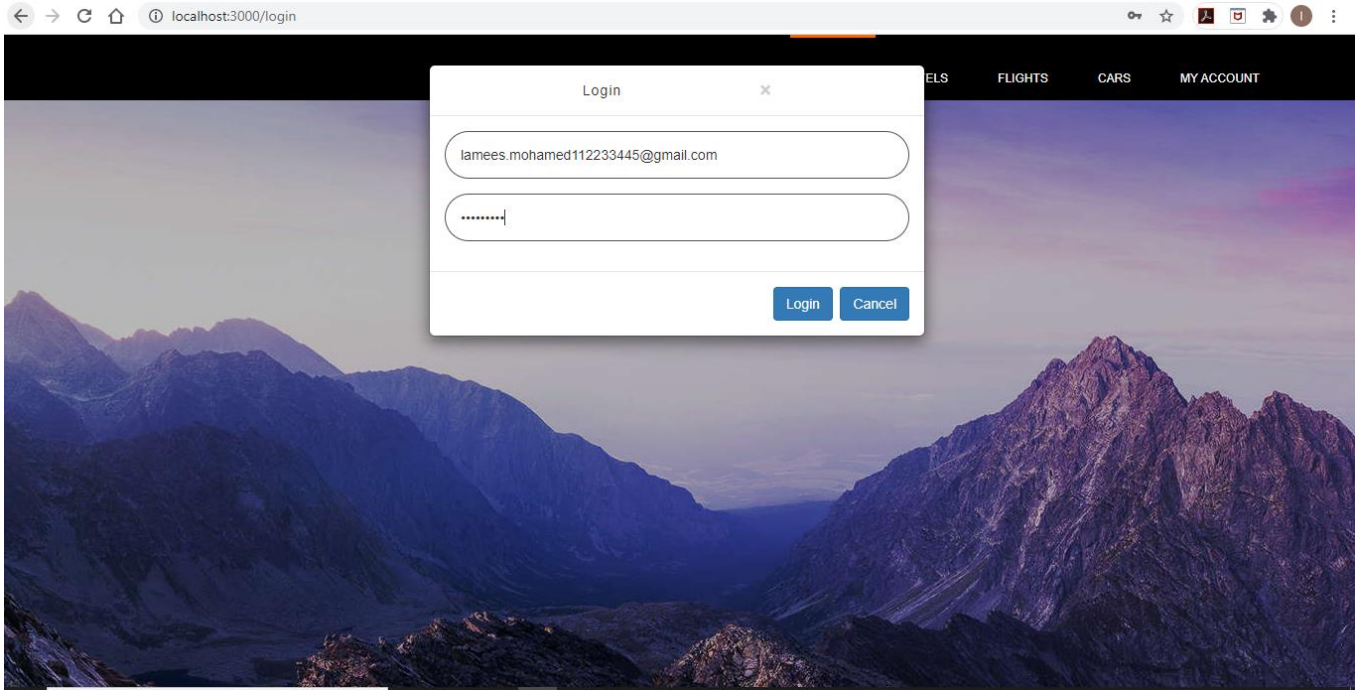
Home Page (UI):



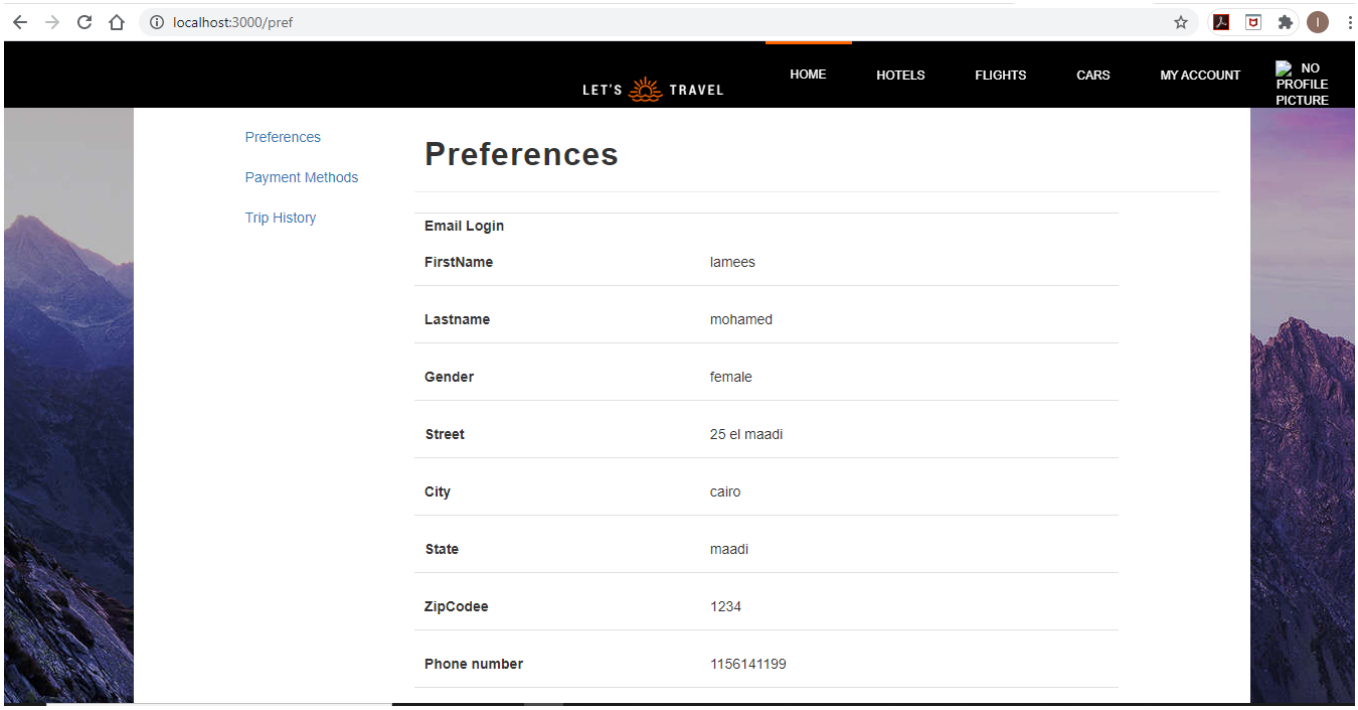
Signup Page (UI):



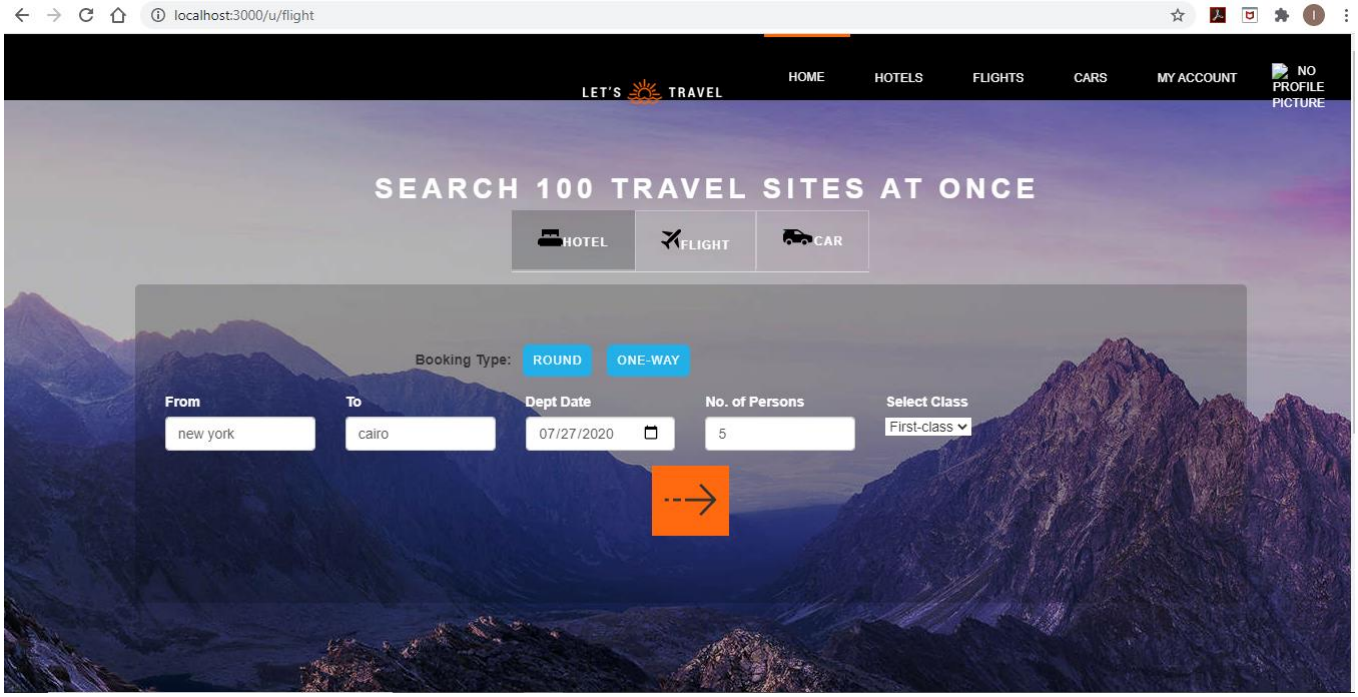
Sign in Page (UI):

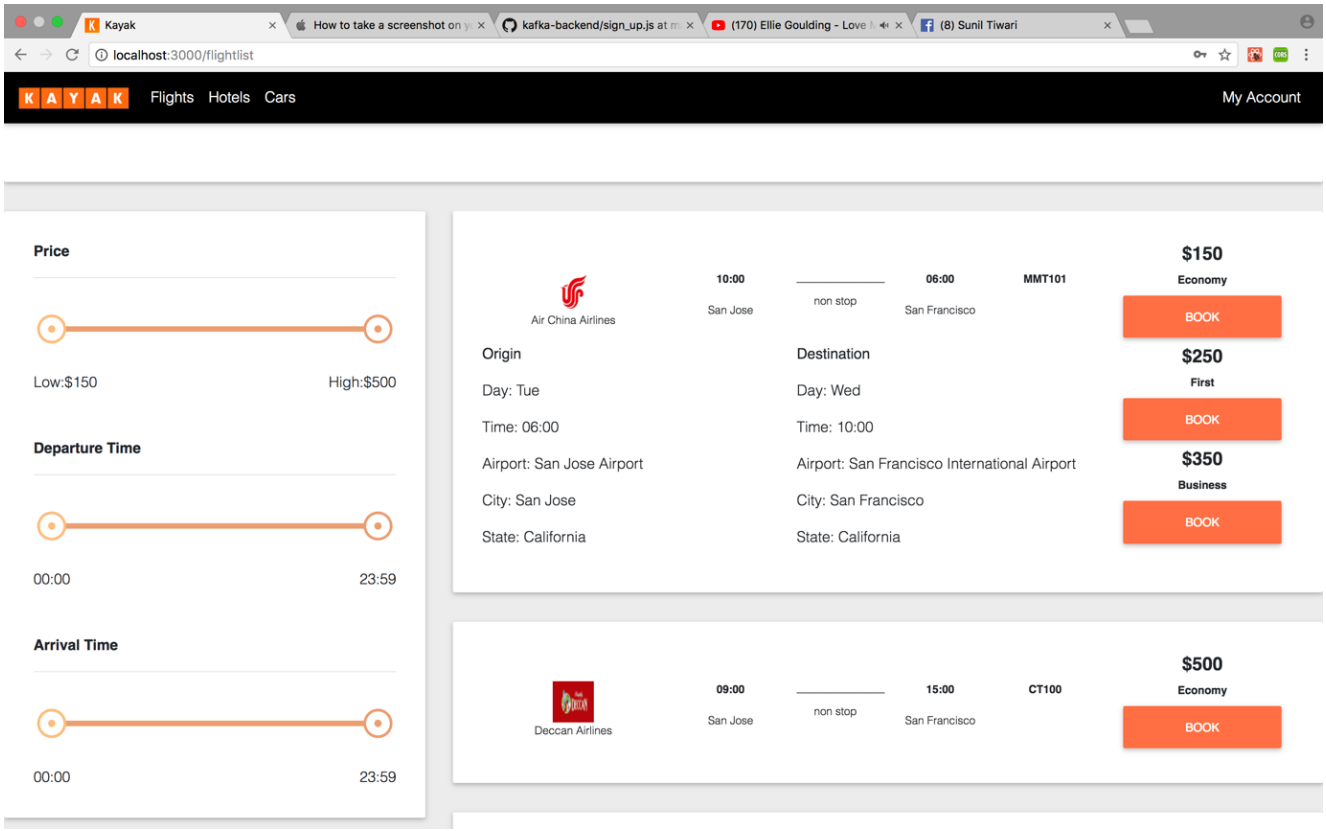


User Profile Page (UI):

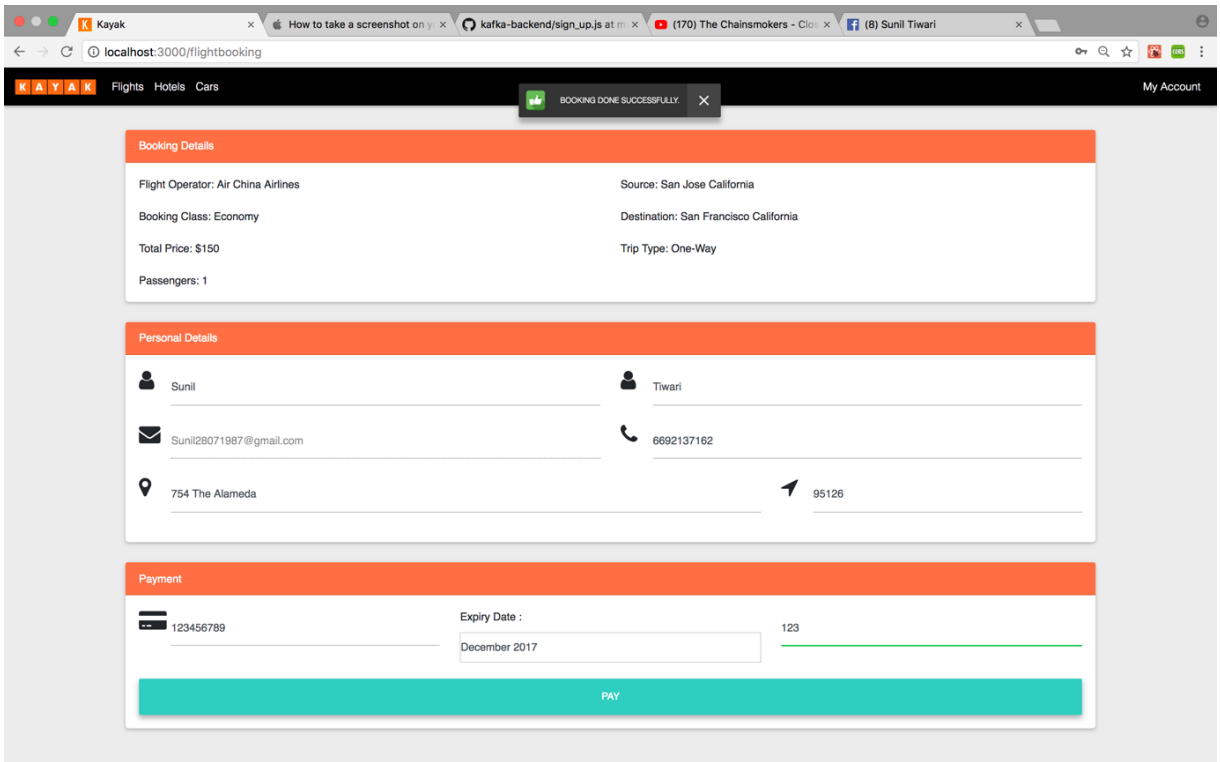


Flight Search Page (UI):

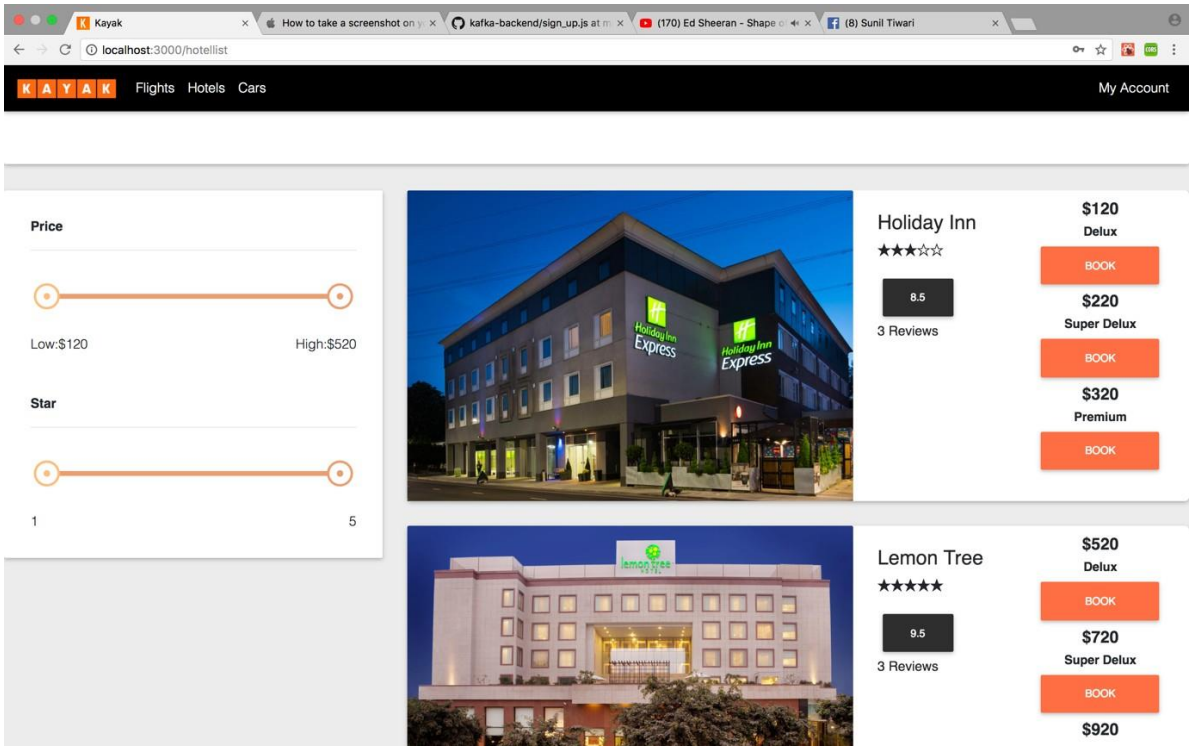


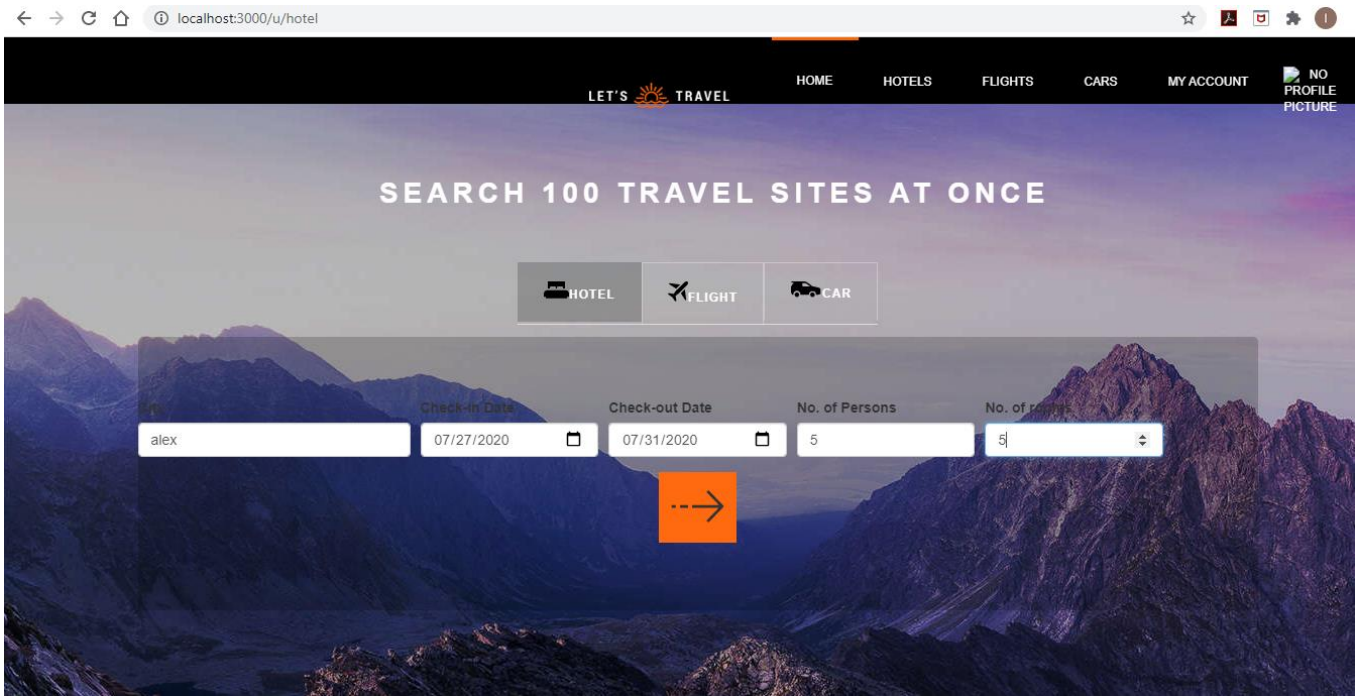


Flight Booking Page (UI):

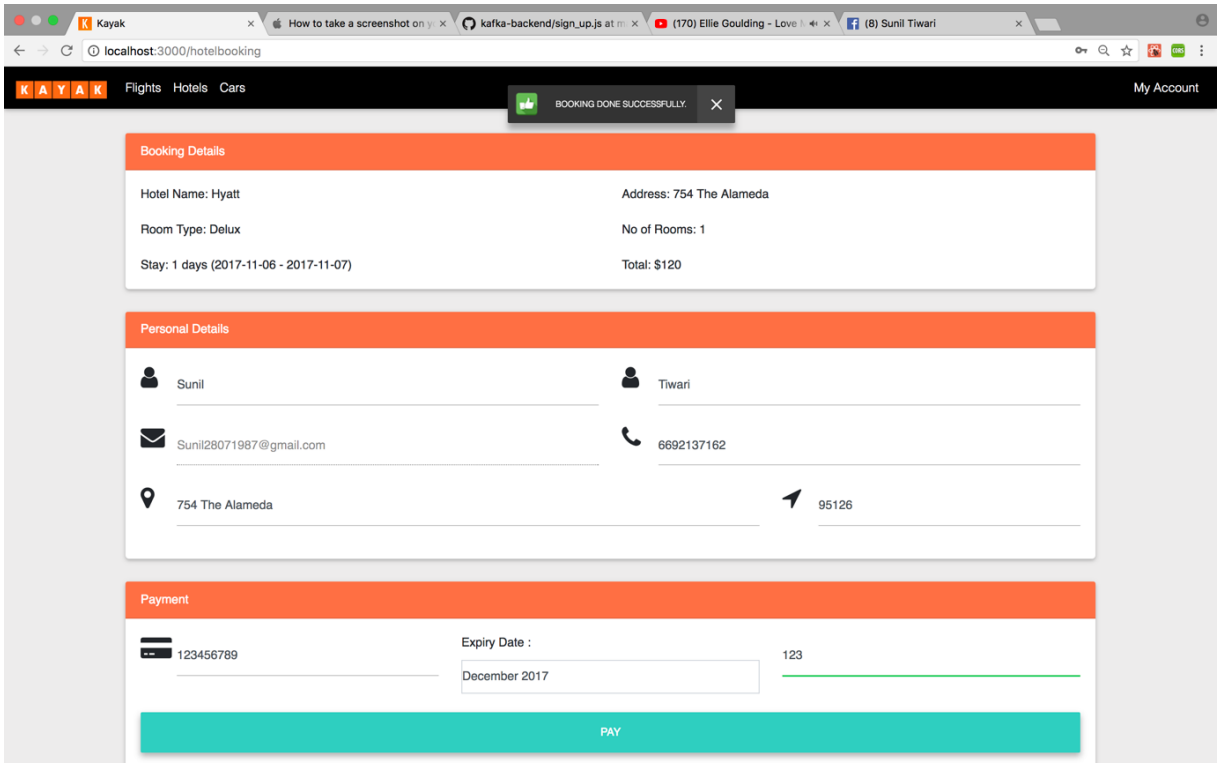


Hotel Search Page (UI)

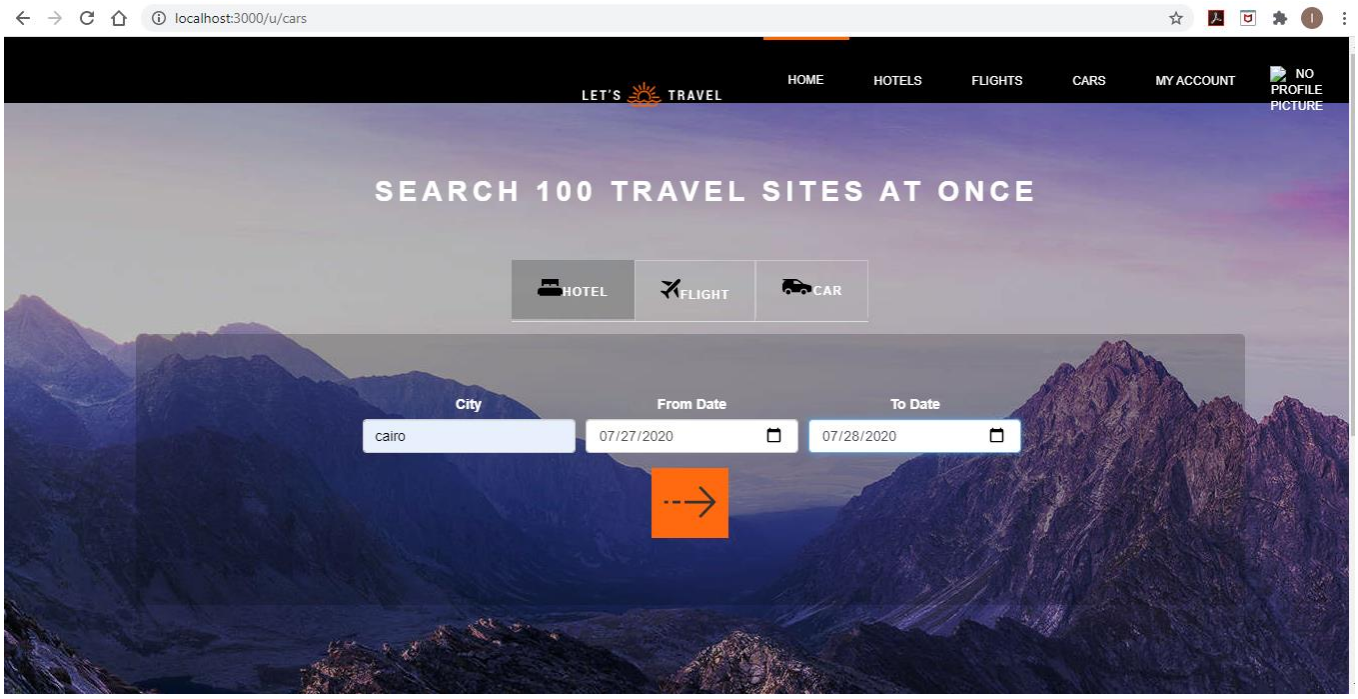


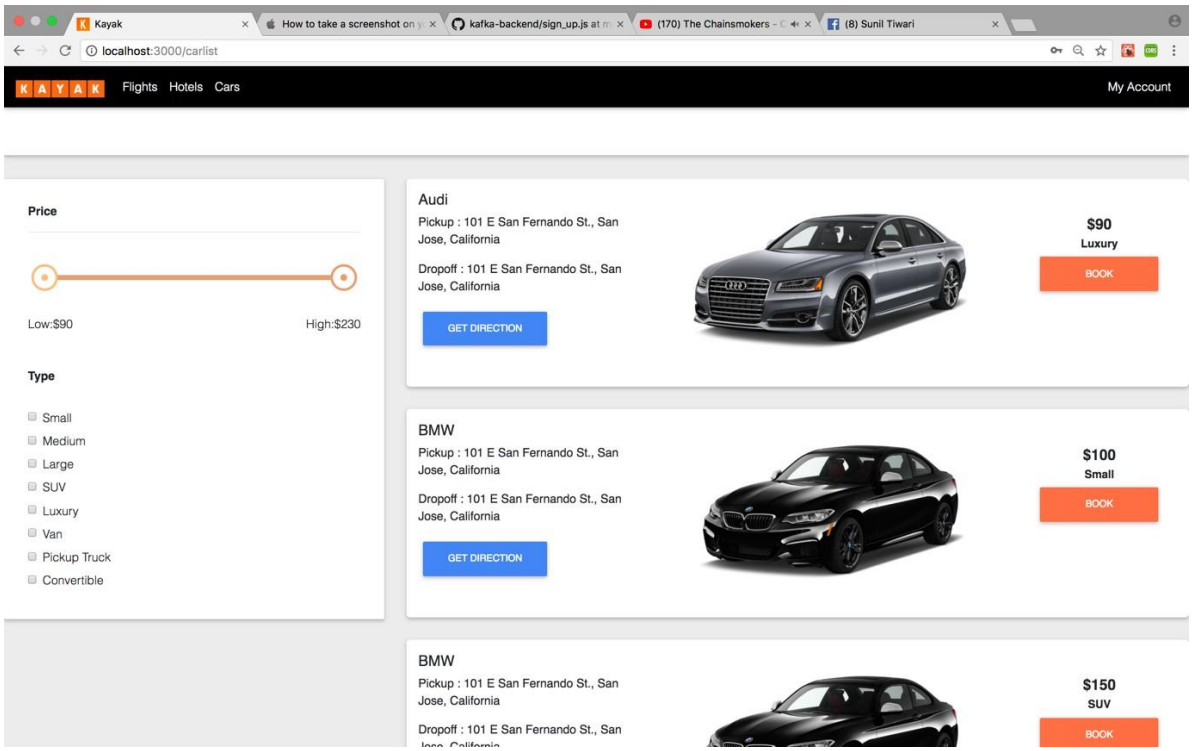


Hotel Booking Page (UI):

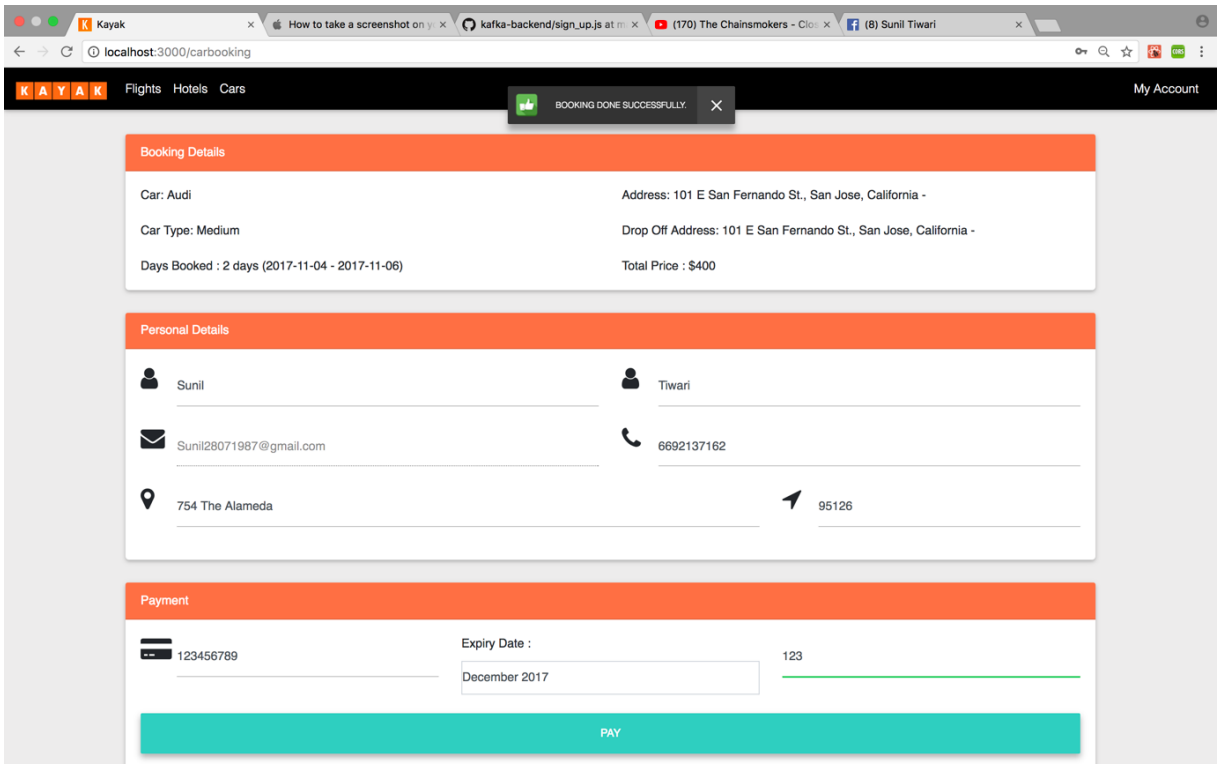


Car Search Page (UI):

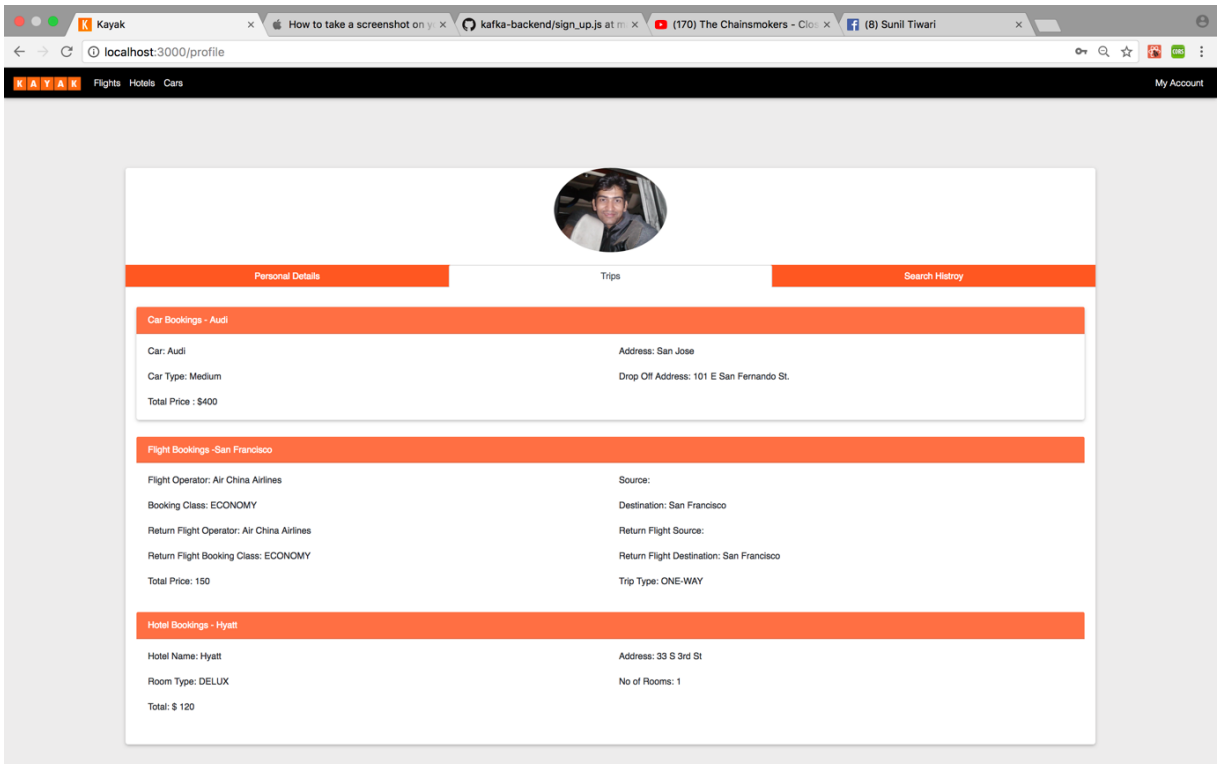




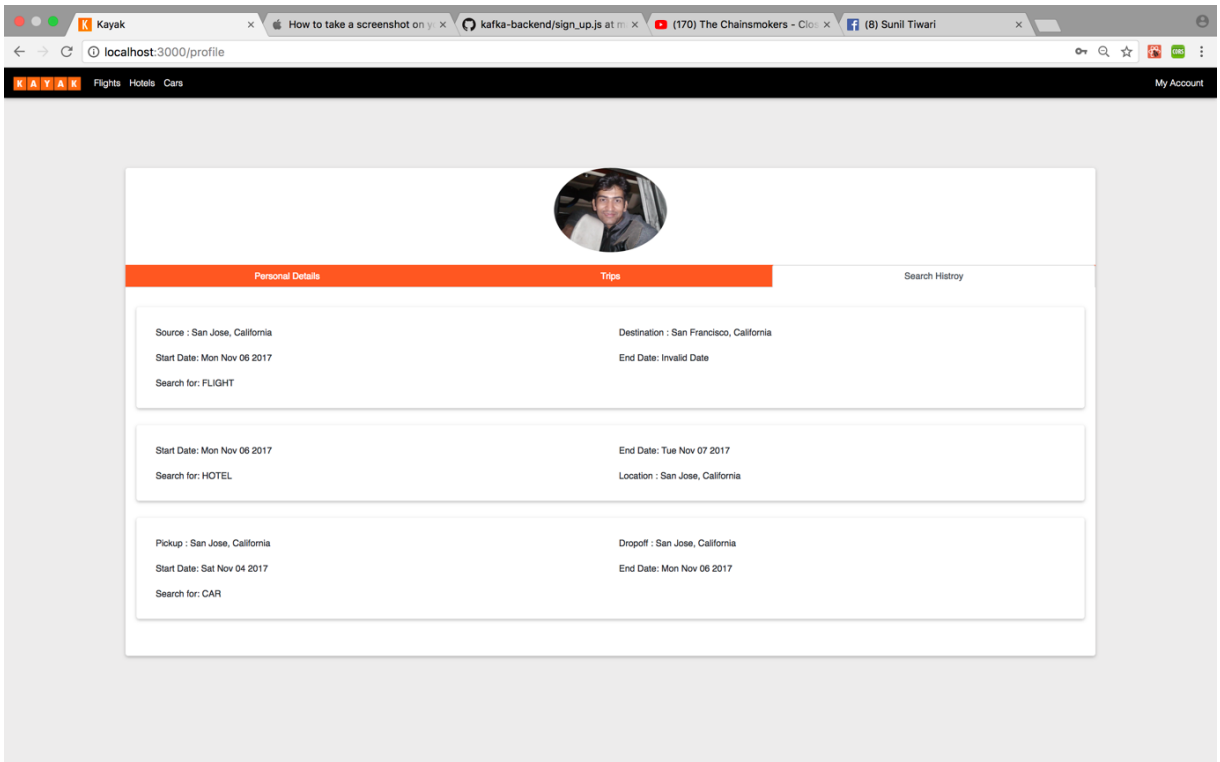
Car Booking Page (UI):



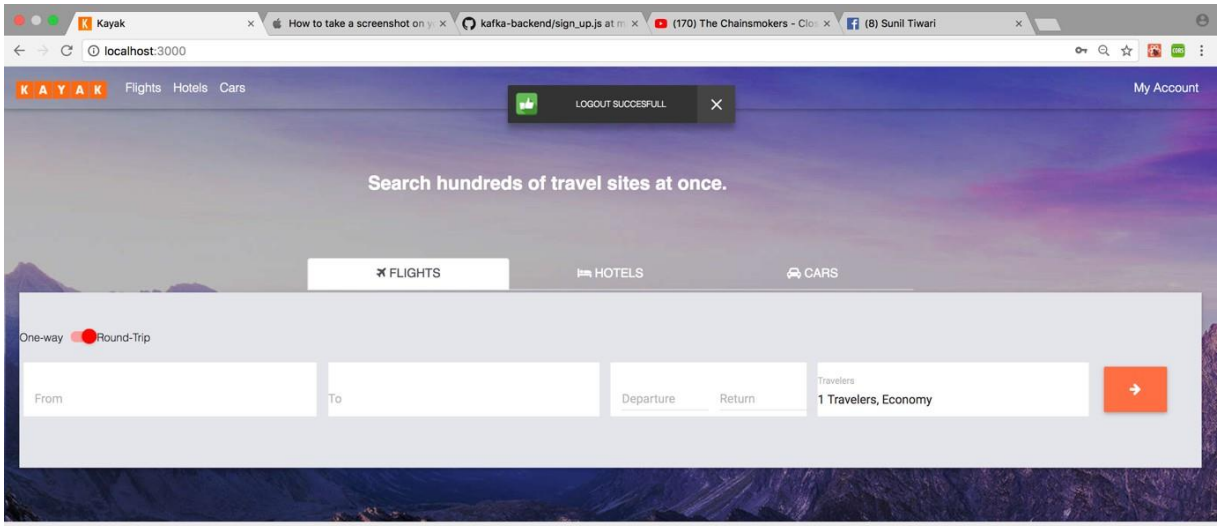
User Booking History Page (UI):



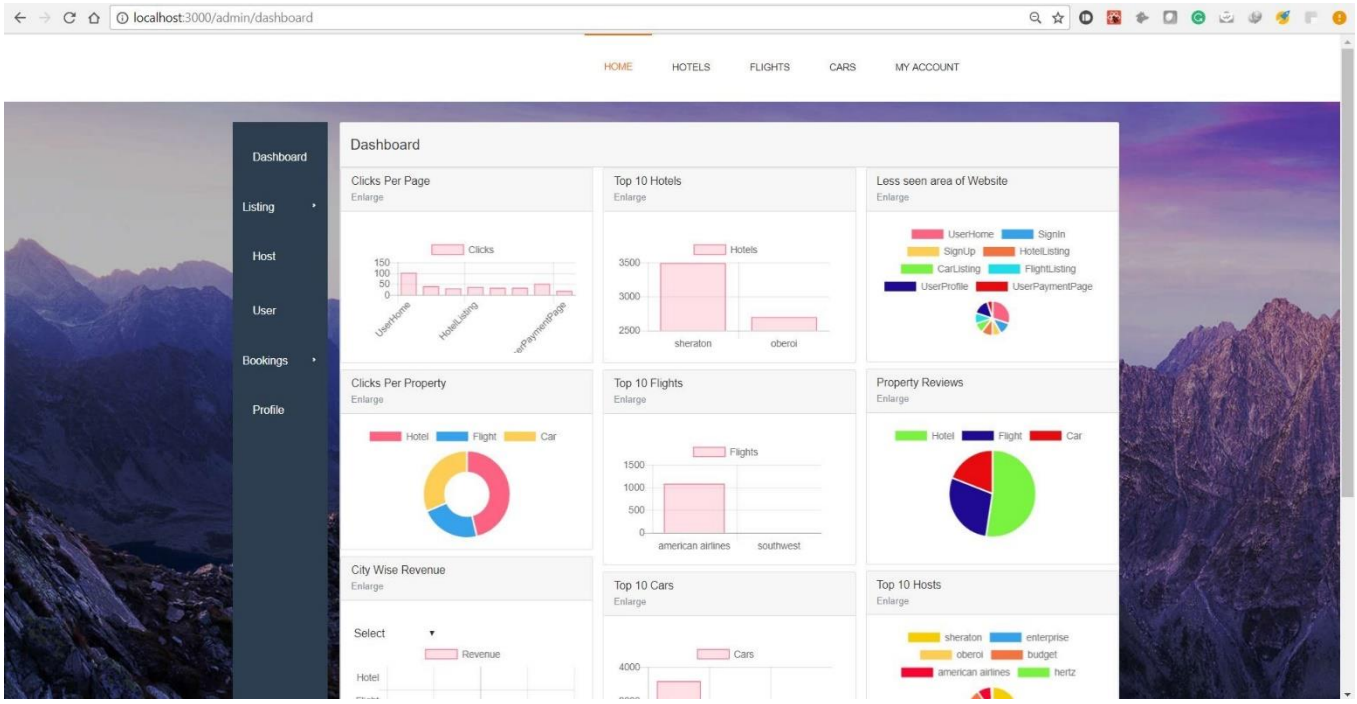
User Search History Page (UI):



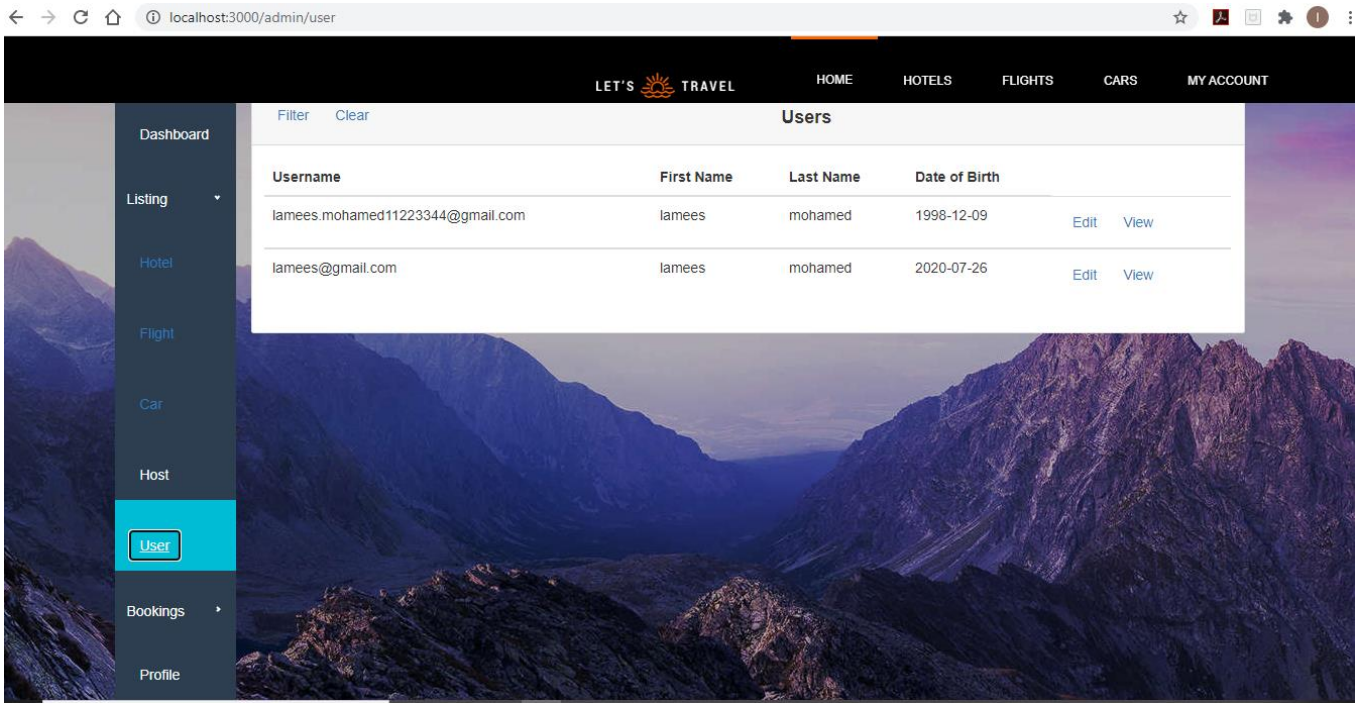
Sign out Page (UI):



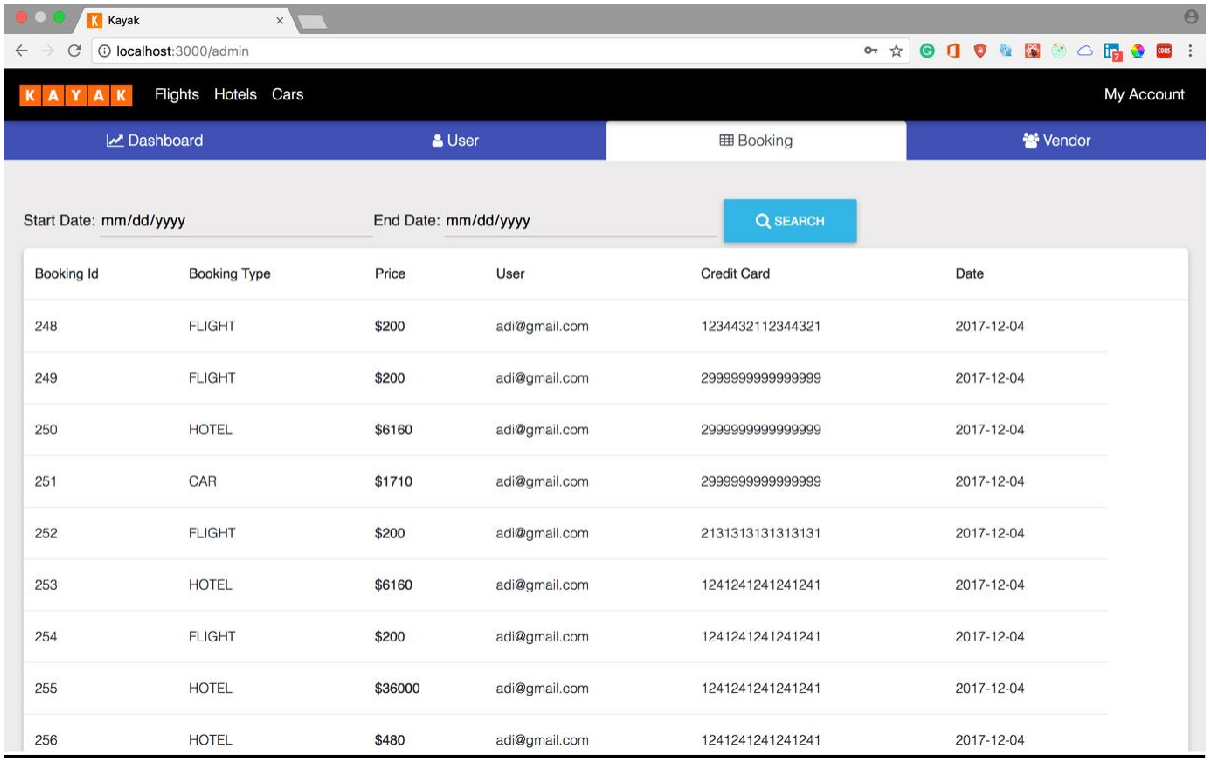
Admin Dashboard Page (UI):



Admin User Page (UI):



Admin Bookings Page (UI):



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

Performance (Test Cases):

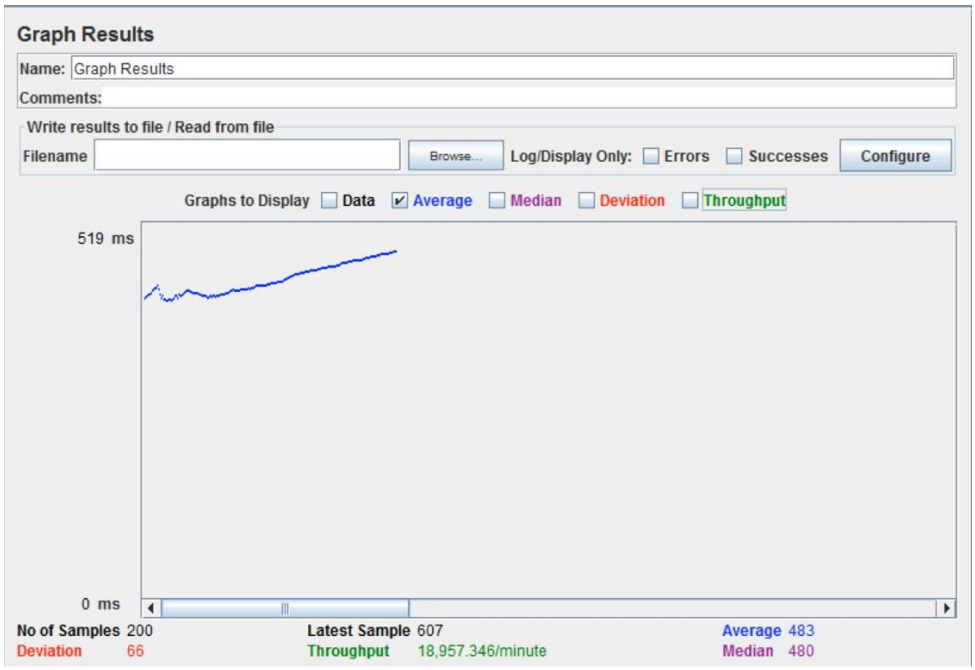
Performance with Kafka queueing is excellent. Kafka is a distributed and parallel processing architecture provides seamless asynchronous API handling experience to provide a faster performance.

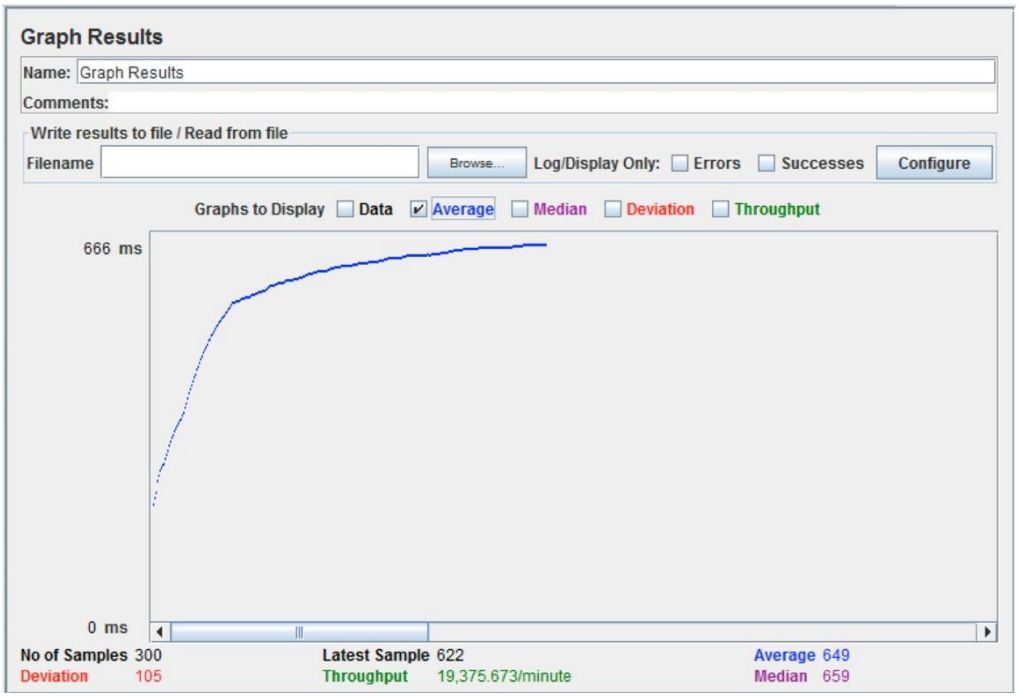
Analysis:
Kafka's performances analysis as per the following testing tells us that Kafka is good at handling large number of requests. Initially, when the requests increase time for service increases, but soon as the request become as large as thousands, it stabilizes. This tells us that our architecture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

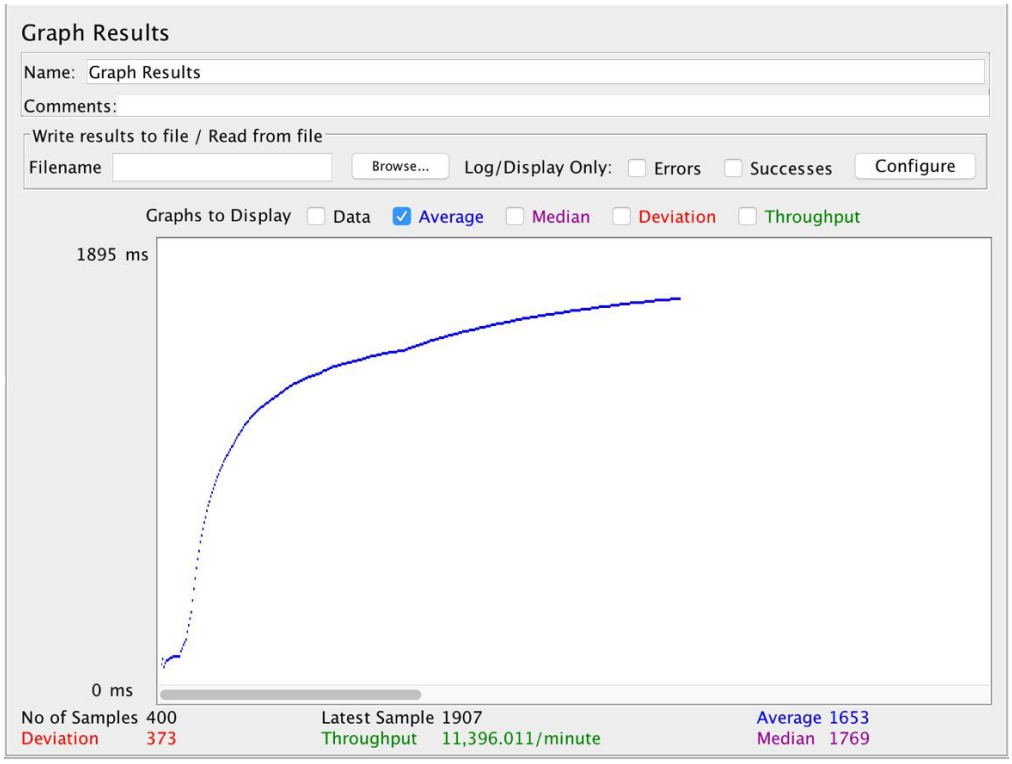
We could have achieved better performance by

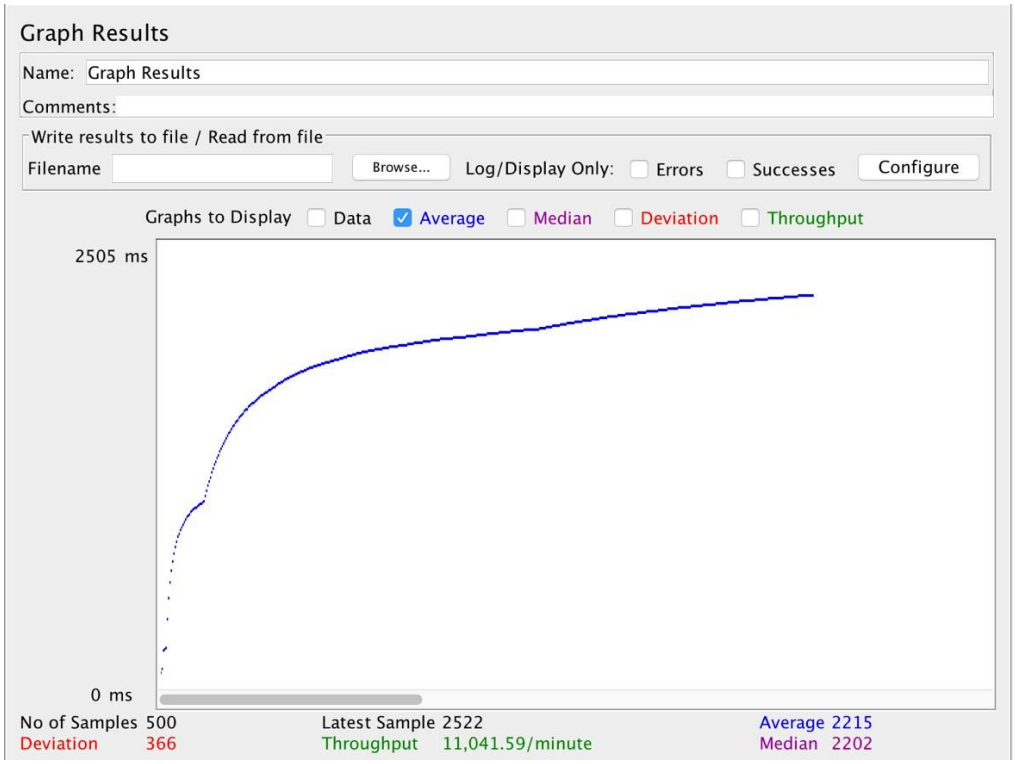
- More number of producer partitions
- Increased number of consumers in a consumer group

JMeter Test Case

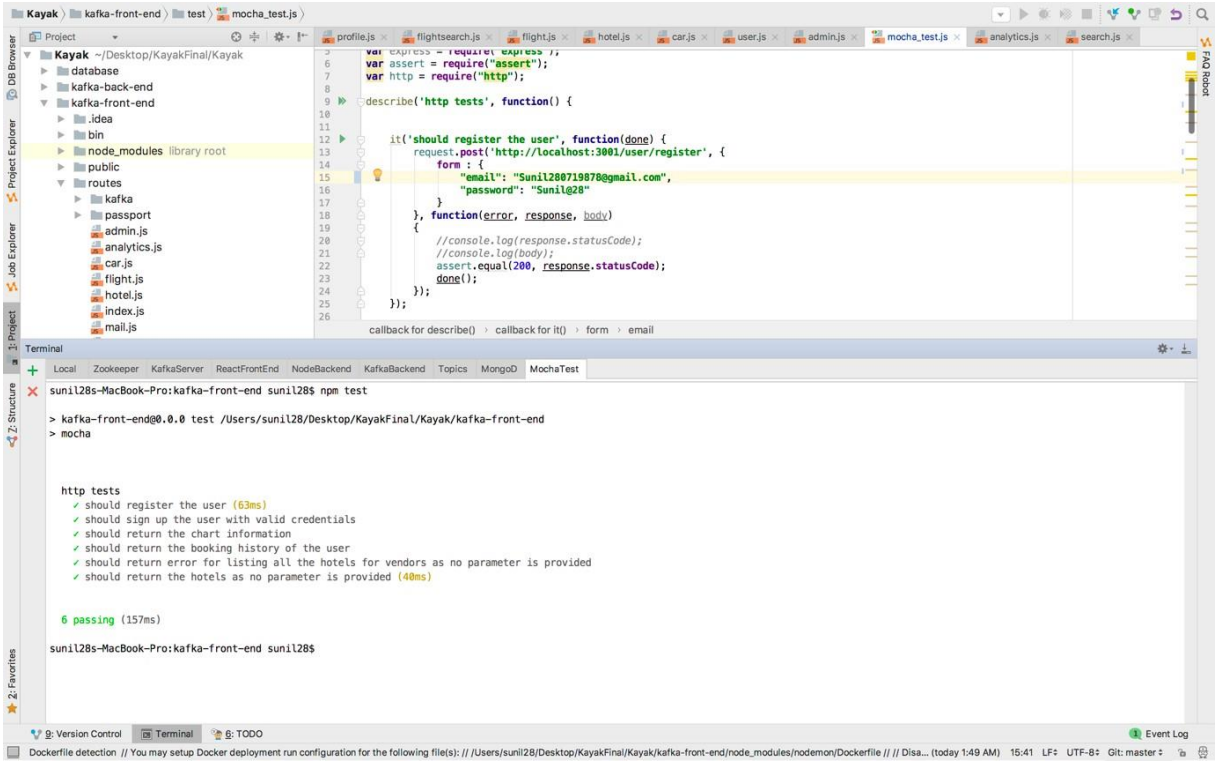


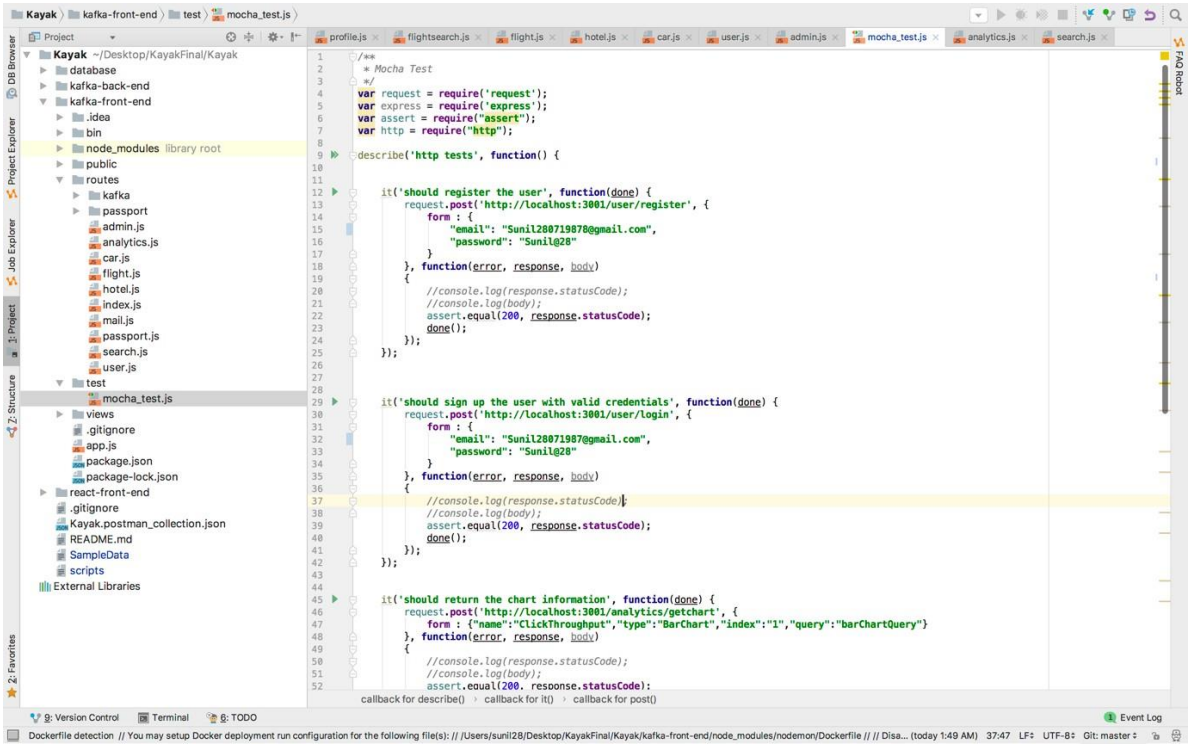

Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes Configure
Graphs to Display Data Average Median Deviation Throughput
666 ms
0 ms
No of Samples 300
Latest Sample 622
Average 649
Deviation 105
Throughput 19,375.673/minute
Median 659


Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes Configure
Graphs to Display Data Average Median Deviation Throughput
1895 ms
0 ms
No of Samples 400
Latest Sample 1907
Average 1653
Deviation 373
Throughput 11,396.011/minute
Median 1769


Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only:
Errors
Successes
Configure
Graphs to Display
Data
Average
Median
Deviation
Throughput
2505 ms
0 ms
No of Samples 500
Latest Sample 2522
Average 2215
Deviation 366
Throughput 11,041.59/minute
Median 2202

Mocha Test Cases





Kayak 〉 kafka-front-end 〉 test 〉 mocha_test.js 〉

Project

- Kayak ~/Desktop/KayakFinal/Kayak
  - database
  - kafka-back-end
  - kafka-front-end
    - .idea
    - bin
    - node_modules library root
    - public
    - routes
      - kafka
      - passport
      - admin.js
      - analytics.js
      - car.js
      - flight.js
      - hotel.js
      - index.js
      - mail.js
      - passport.js
      - search.js
      - user.js
    - test
      - mocha_test.js
    - views
    - .gitignore
    - app.js
    - package.json
    - package-lock.json
  - react-front-end
  - .gitignore
  - Kayak.postman_collection.json
  - README.md
  - SampleData
  - scripts
- External Libraries

profile.js | flightsearch.js | flight.js | hotel.js | car.js | user.js | admin.js | mocha_test.js | analytics.js | search.js

```
41          });
42      });
43
44
45      it('should return the chart information', function(done) {
46          request.post('http://localhost:3001/analytics/getchart', {
47              form : {"name":"ClickThroughput","type":"BarChart","index":"1","query":"barChartQuery"}
48          }, function(error, response, body)
49          {
50              //console.log(response.statusCode);
51              //console.log(body);
52              assert.equal(200, response.statusCode);
53              done();
54          });
55      });
56
57
58      it('should return the booking history of the user', function(done) {
59          this.timeout(5000);
60          http.get('http://localhost:3001/user/bookinghistory', function(res) {
61              assert.equal(200, res.statusCode);
62              done();
63          })
64      });
65
66
67      it('should return error for listing all the hotels for vendors as no parameter is provided', function(done) {
68          this.timeout(5000);
69          http.get('http://localhost:3001/hotel/gethotels', function(res) {
70              assert.equal(500, res.statusCode);
71              done();
72          })
73      });
74
75
76
77      it('should return the hotels as no parameter is provided', function(done) {
78          this.timeout(5000);
79          http://localhost:3001/hotel/hotels', function(res) {
80              assert.equal(200, res.statusCode);
81              done();
82          })
83      });
84
85
86
87  });
```

callback for describe() 〉 callback for it() 〉 callback for post()

9: Version Control | Terminal | 6: TODO | Event Log

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

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 act