CMPE 273

ENTERPRISE DISTRIBUTIVE SYSTEMS

TEAM PROJECT

KAYAK PROTOTYPE

TEAM 11

https://github.com/asbharadiya/kayak

AMAN OJHA

ANKIT BHARADIYA

MAULIK BHATT

PALASH HEDAU

PRATEEK SHARMA

Contents

ABOUT

KAYAK

This is a prototype of the website, Kayak.com which is quite popular for all the travel needs of a customer. The website can be used to book flight, car or hotels from other service providers. Kayak acts as an aggregator in between and can also be used for comparing rates on different websites.

KAYAK was founded in 2004 by Steve Hafner and Paul M. English. Being available in over 18 languages, this website is widely used across many countries. It is particularly used as an meta search engine for finding flight, cars and hotel bookings.

In our application, the front end is created using React, HTML5, Bootstrap and CSS. After logging in, the user gets various options like search cars, flight and hotels along with booking and payment options. All these requests are sent to the server where various RESTful APIs are present to handle the requests. The backend is designed using NodeJS and ExpressJS.

From the RESTful api, the requests are further sent to the Messaging Queue backend where the exact logic for data handling is present. For our application, we are using Kafka messaging queue.

CONTRIBUTION

1. AMAN OJHA
    - Developed APIs for profile page in frontend as well as backend.
    - Flight listing and API development for admin section.
    - Report creation.
    - JMeter load testing along with Mocha testing.
    - Bug fixing and UI testing.

2. ANKIT BHARADIYA
    - Set up git repository and base project structure
    - Set up Kafka queue and topics
    - Defining API endpoints and main flow of data of entire project
    - Handling common CSS classes and responsiveness of webpages
    - Developed user Home module with search bar, payment module and common modals
    - Connection pooling implementation

3. MAULIK BHATT
    - Hotel listing and API development for admin section.
    - Analytics charts and user tracking in admin section.
    - Listing and API development of Billing section in admin side.
    - Research and implementation of Mongoose ODM.
    - Result sorting in each type of listings on user side.

4. PALASH HEDAU
    - Car listing and API development for admin section.
    - Admin's user listing module with search.
    - Search and Filters functionality across client and admin module.
    - Handling complex queries which involved nested updating for billing and booking module.
    - Cars booking frontend and billing interface. User past booking module
    - Redis setup and integration.

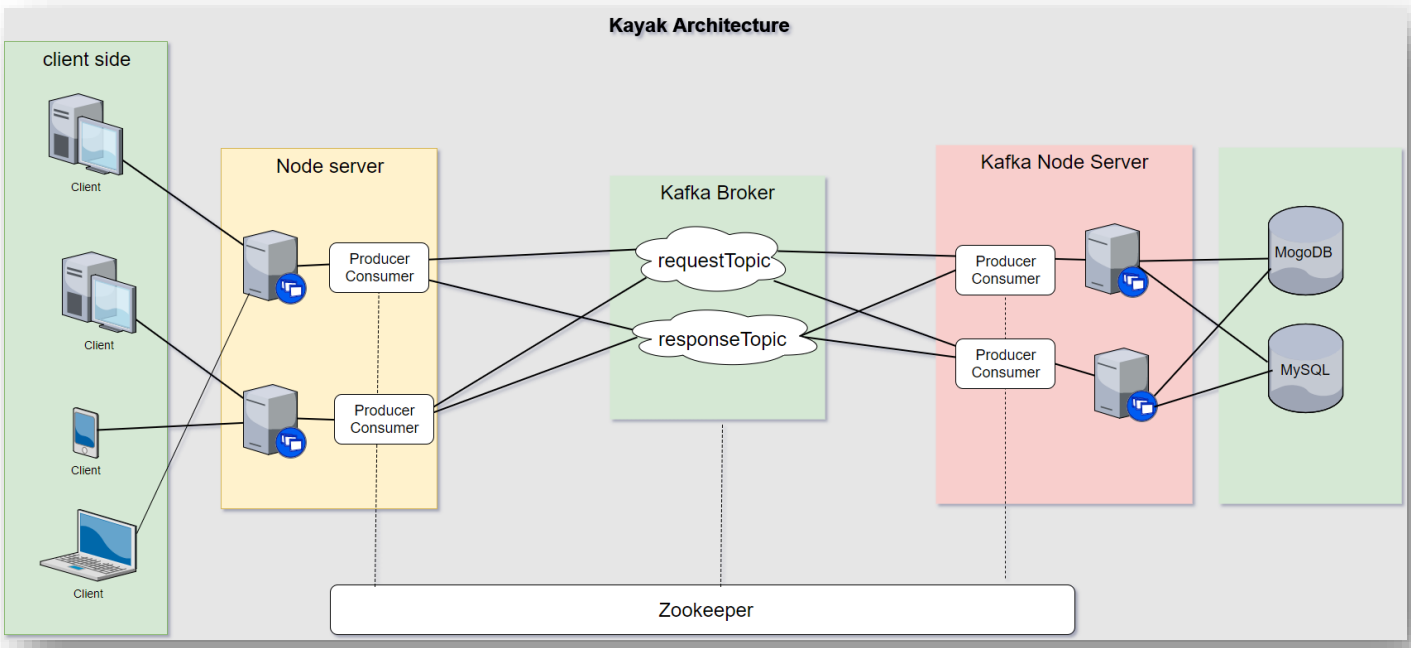
5. PRATEEK SHARMA
    - Passport authentication setup for admin as well as clients.
    - Hotel listing and API development for client website.
    - Implementation of user analytics to be displayed on client side.
    - Developing CRON jobs and data logging which was later used in analytics section.
    - Bug fixing and performance improvement on the go.

ABOUT THE PROJECT

1. Object Management Policy

   Kayak is a vast project in its own with various functionalities like managing listing of flights, cars and hotels. It also handles booking information, managing user's information, billing etc. Because of the multi offerings, we made the project modular.

   The project's architecture is as follows:



Admin Frontend:

The first module is admin frontend which can be only accessed with admin privileges. It is a centralized console from where an admin can control all the data to be displayed on the website. The admin will have rights to add, delete, modify data of flights, cars and hotels. While adding car, flight or hotel admin can add different details of each category i.e. name, address, city, source, destination, room availability, booking dates etc and upload pictures also.

The admin will also have access to an analytics page which will provide various analysis of gathered data of how the website is being used. Through analytics report and charts, admin can see share of each category hotel, flight and car in total revenue and orders. Admin can also view highest revenue generating companies in last year and last month, and city wise revenue. Admin also access track of user and most viewed or clicked listing.

The admin will also have access to listing of all the users registered on the website and can delete them if needed. It will also have information regarding billings.

Client Frontend:

The client frontend is the actual web application which will be used by the end user. After landing on the home page, the user will have options to search for cars, hotels and flights. While searching we provide suggestions of the cities of USA according to entered string in input box. The results will be displayed according to the user's selection criteria. User can change filter criteria on the fly and new results will be displayed. User can also sort the results (ascending/descending) based on price, rating, stars, departure time. Also, if more than 20 results come than load more option becomes visible to access results on next page.

After selecting a option, the user will be redirected to the booking page where he has to fill in all the information required according to listing category. Once all the information is validated, user will proceed to checkout. On successful payment, billing will be done.

User can also see his/her profile, past bookings and register payment methods. User also can update his/her person details and can also upload profile picture if want to. User also can register multiple payment methods which will be provided next while billing.

Node backend:

The node backend server is responsible for handling all the requests made by the above two front end services. It is the first point of contact which routes all the api calls to its respective handler. The first thing after receiving a request from the front end is to check whether the request is from a validated user or not. All these requests are handled by passport. After validation, the requests are passed on to the kafka node backend using kafka topics.

Kafka server:

The kafka server maintains topics which are also known as messaging queues. All the requests are passed via the request topic and received via the response topic. All the requests sent in the topic are associated with a unique id for identification purpose. These requests are further sent to the kafka node back end where the actual logic for data handling is present.

Kafka node backend server:

Once a request is received by the kafka back end, we perform the desired operation or fetch the required data from databases such as MongoDB or MYSQL. The required data is then further sent to the node back end via kafka server in response topic. Once the node back end receives the data, it sends the data back to the front end where the data is displayed to the user.

2. "Heavyweight" Resource Handling

As it is known that performing queries on the database requires costly resources, it is very essential to limit these kinds of activities. We managed our resources in the following way to reduce data fetch time, keeping in mind the necessary security measures.

- All the user's login information is stored in MySQL as it is more secured.

- Various other information like data related to cars, flights and hotels along with booking, billing and analytics data is stored in MongoDB as it supports faster data retrieval and easy to store data in modular format with no relation.
- All the images are upload on the amazon server and only the links are stored in the database. This helps in avoiding unnecessary data dumping in the database which could have hampered retrieval time.
- We used Redis to implement cashing, so we need not to perform actual DB operation if we already found same in key in our cache. It gives better performance and less resource utilization.
- We implemented connection pooling, so form beginning we keep fix number of connection open so when heavy load comes we need not to open and close many connections. We just take one connection from pool, use it and then return it back to pool for reuse. It helps us to perform DB operation faster.
- For implementation of kafka messaging queue, we created two topics: kayak and response_topic. First one is for sending message from node server to kafka backend server and later one is for return flow to get response.

3. Policy of writing data into the Database

As we discussed above, that performing queries on the database requires costly resources, it is very essential to limit these kinds of activities. We followed a few policies which helped us keeping our databases optimized.

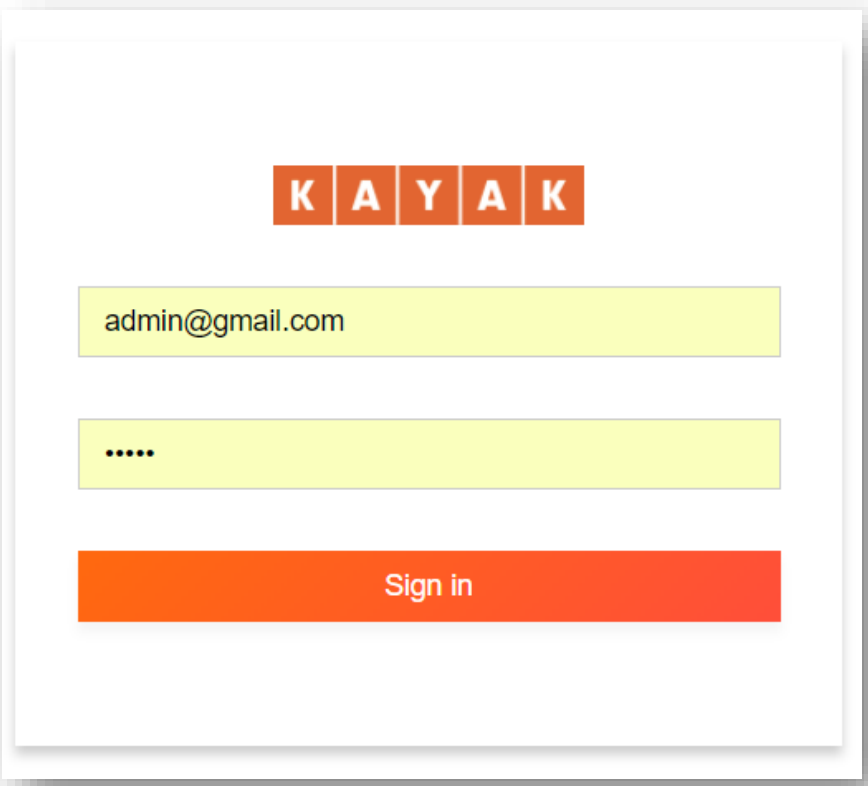
- No unnecessary data writing in the database. We only updated the database when a attribute is changed.
- All the images are upload on the amazon server and only the links are stored in the database. This helps in avoiding unnecessary data dumping in the database which could have hampered retrieval time.
- We used SQL caching which helped us retrieving data faster.
- We have created different api for different kind operations on each resource like GET, PUT, POST, DELETE. Because of this it helps us keep data integrity.
- We used MySQL for maintaining user login information as MySQL is more secured.
- Other data of the user as well as flights, hotels and cars with billing and booking information is stored in MongoDb because MongoDb is faster for data retrieval and security is not the actual need in this case.
- We also have many heavyweight images for user profile, hotel images, car images which we are directly saving on the cloud. This is done to avoid excessive data to be stored in the database which will also increase the hamper the performance.

ADMIN APPLICATION

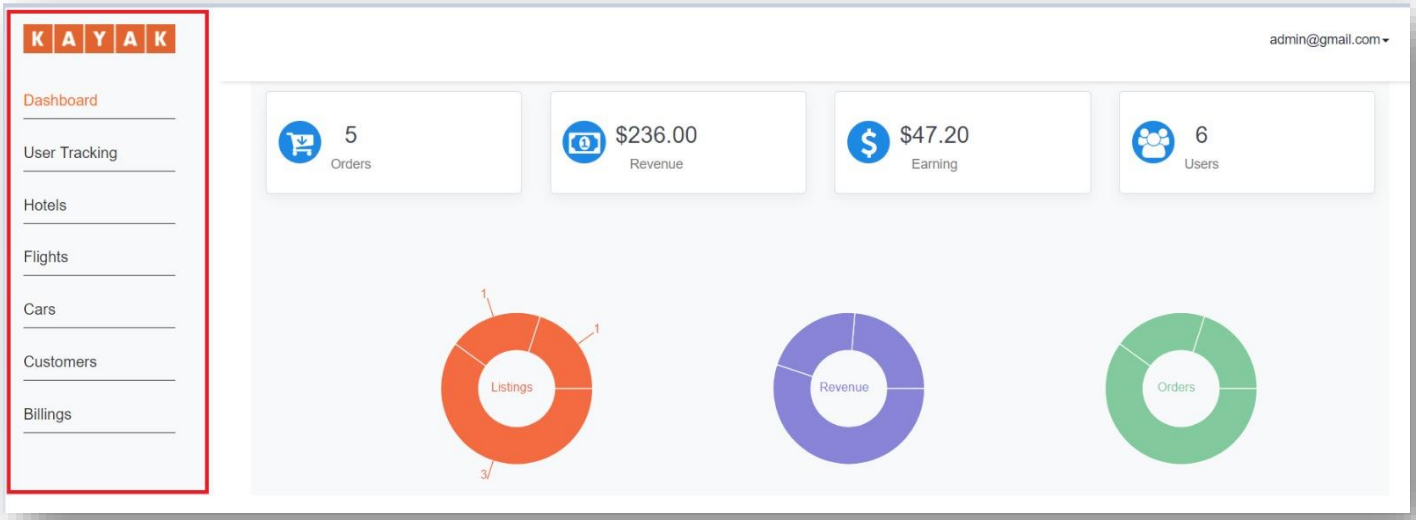
Admin Section

An admin application is made which can be only accessed by the admin credentials. Admin application has all the special privileges of adding, deleting, editing a flight, hotel or car. All the data added by the admin will be stored in MongoDB and will be displayed in the main customer application based on the search results.

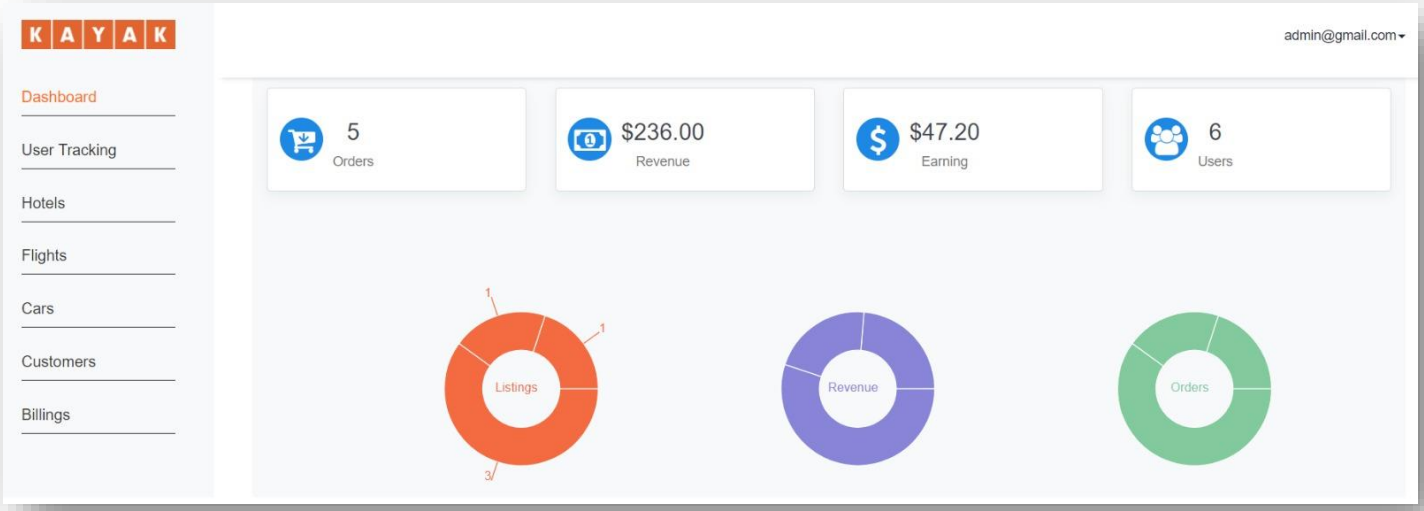
A customer will not have any option to add, delete or modify the flights, hotels or cars data.



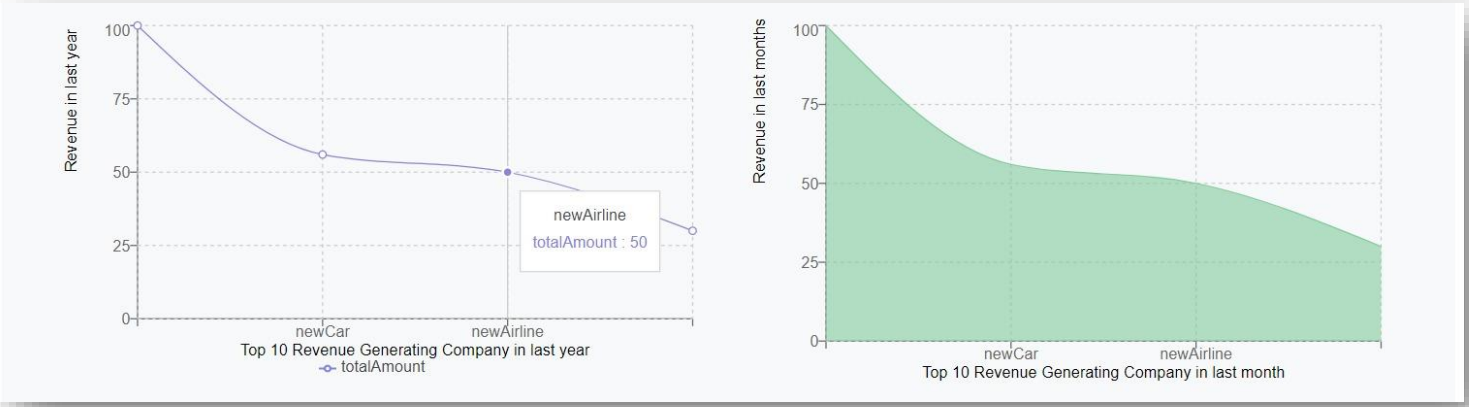
After login, a admin will get various options in the side bar (marked in red), Admin can use the links to go to different tabs and access data accordingly.



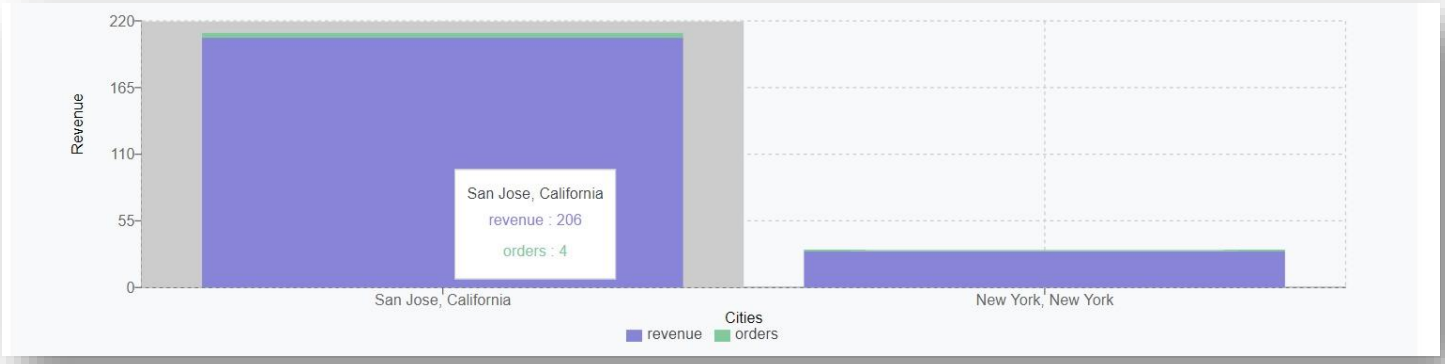
On the dashboard and User Tracking page, the admin can see various interactive charts and figures which can be used for data analysis. The charts can be used to analyze the total revenue, number of users registered, number of visitors, total earning, total orders etc. Using pie charts, admin can see share of each category in revenue, orders, and total listings.



Admin can see the highest revenue generating listings in last year and last month in two different graphs shown below. On hover of each point in line chart we can see value of revenue value and name of listing at that point. This data is generated from data of past year and past moth billings.



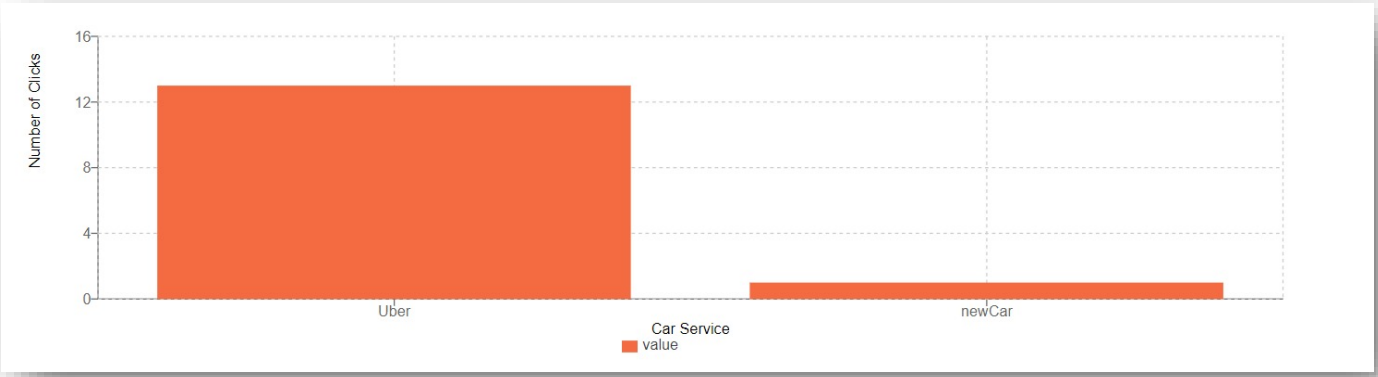
Admin also can see top revenue generating cities in bar graph below. On hover of each bar admin can see the revenue amount and number of orders from that city. On x-axis the name of cities is given. For example, we have taken San Jose and New York here.

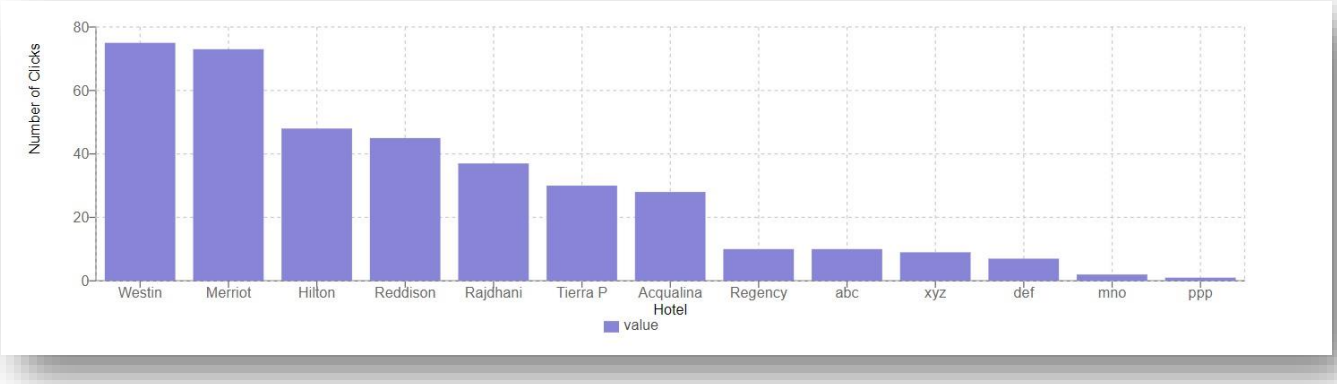


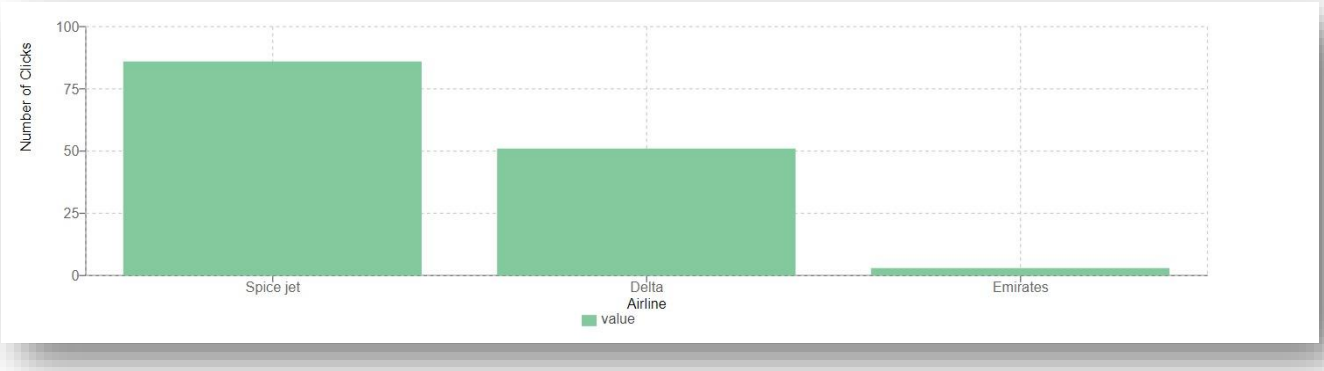
We have created click tracking services that firs on click of different html elements and loges data in log file. A crone job is set up and performs operations on log file and fill data in analytics collection in DB from this file. Later we use this data to perform various analytical operations and generate attractive graphs.

Using this click count data we can show most popular hotel, flight and car services on our user tracking page. We also showed which pages are accessed most by user in form of bar graph like below.

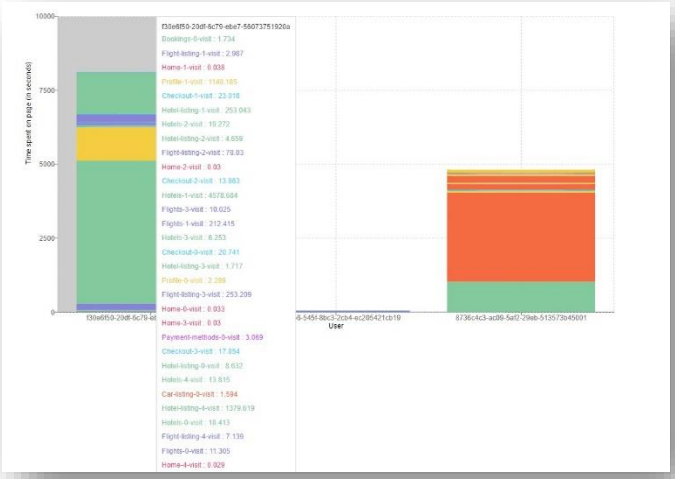
As we have click count data for each listing we are showing top car services which are popular among users. Same way we are showing top airlines and top hotels. On hover of each bar admin can see click counts for that listing. The name of listing are shown on x-axis.



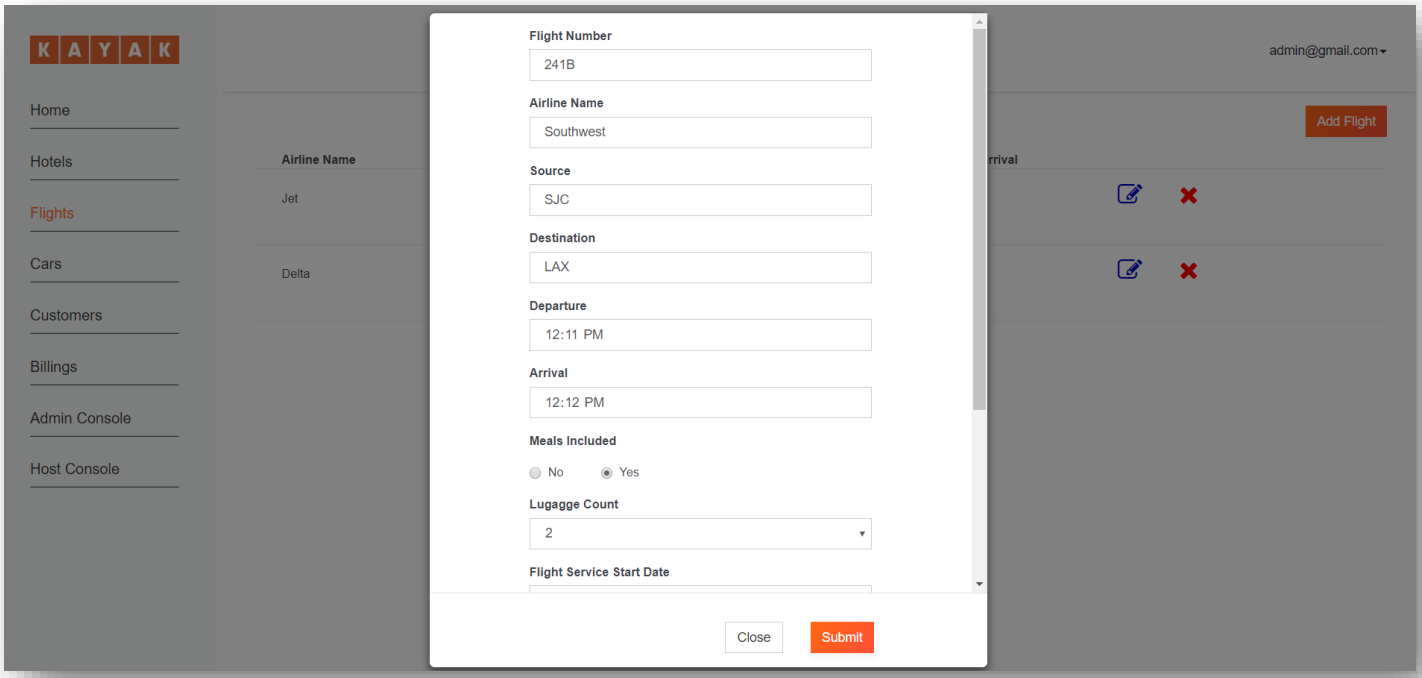




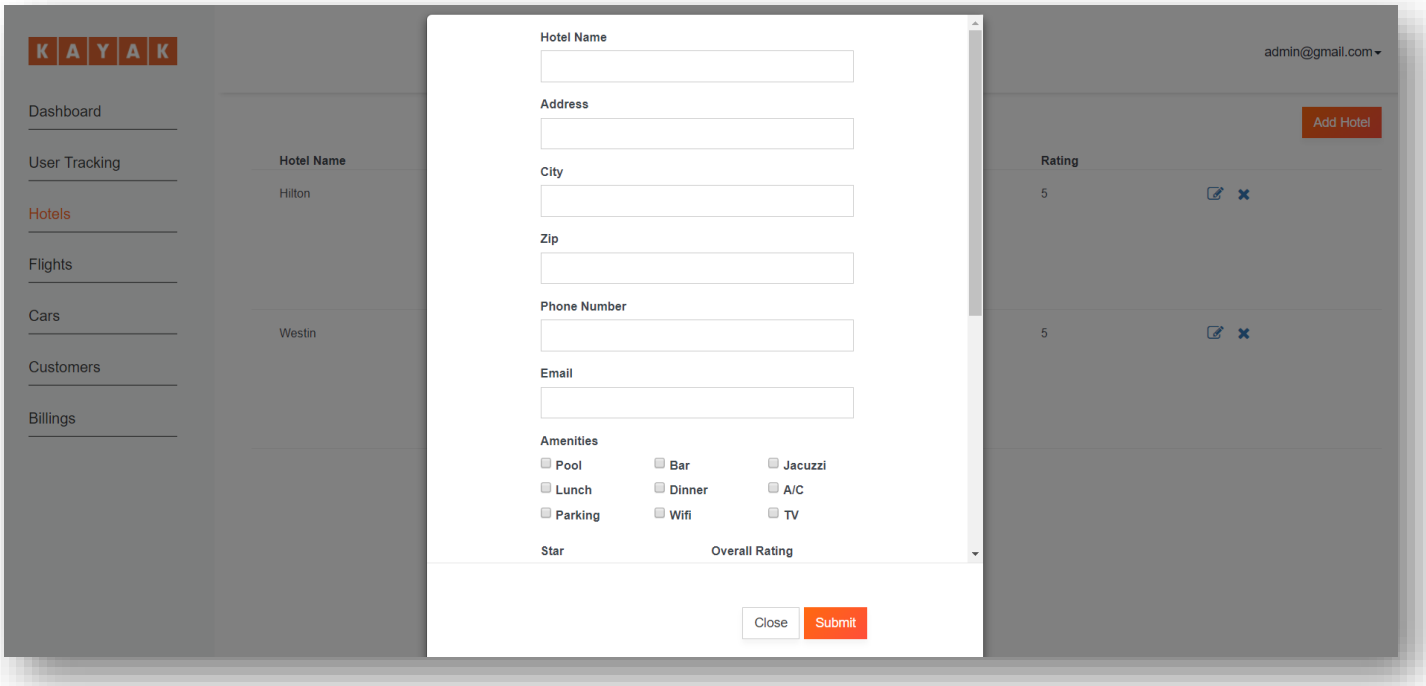
For user tracking purpose we used bar graph to show amount of time user spent on each page in chronological order, starting from bottom of bar. On hover of bar whole track of user access is show like below screenshot. Id of user is shown on x-axis to maintain anonymity. If user accessed same page for multiple times by navigating from different pages, each visit is mark with a number and time spent in each visit, that is also visible in screenshot. This way we can track any user very easily.



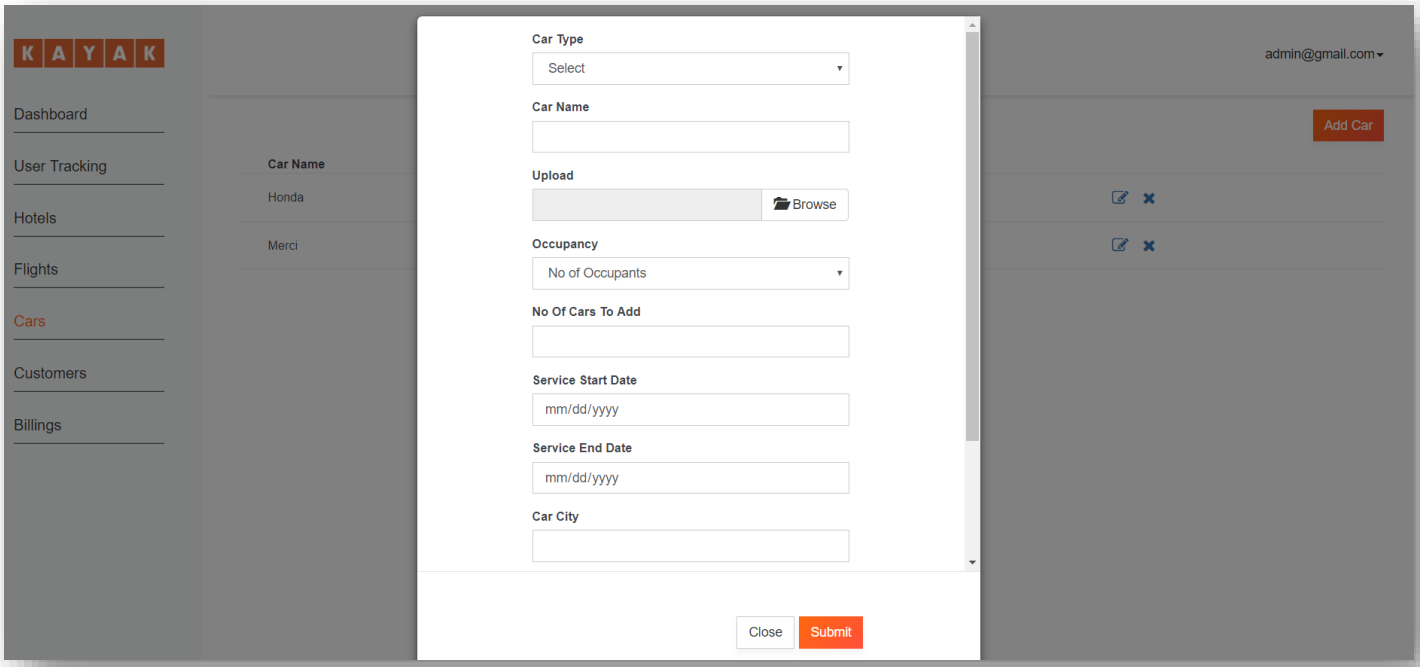
All the pages have options to add data, below is an example to add a flight details.



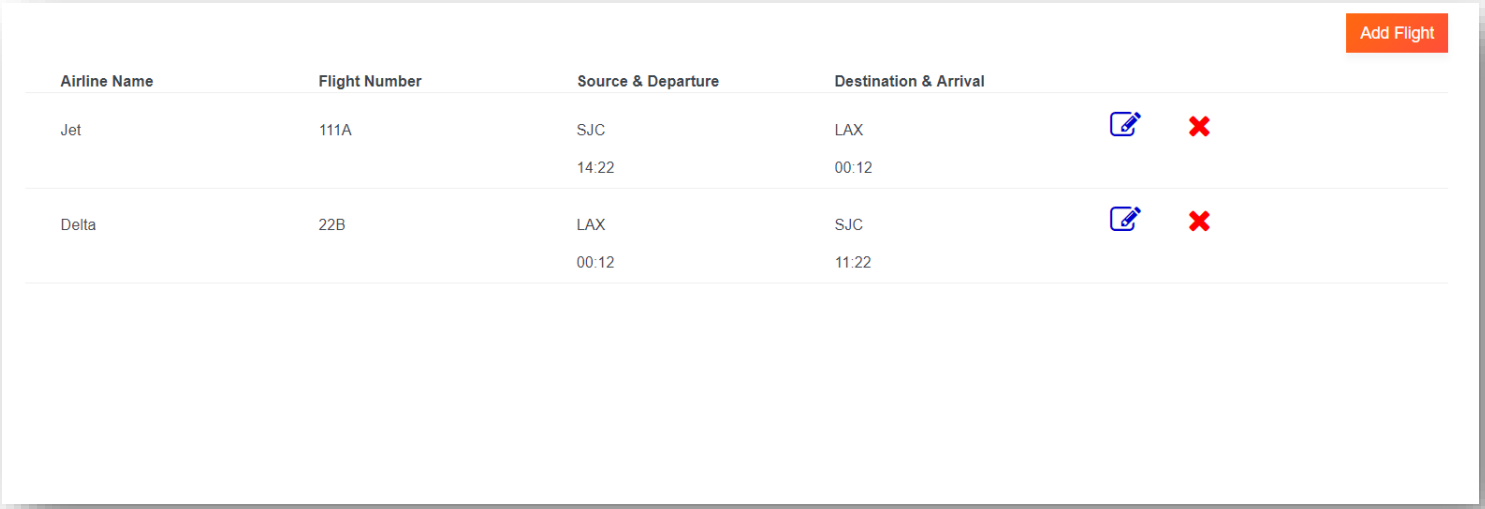
Option to add Hotel



Option to add Hotel



After adding data, the details are shown as below. Admin will also have an option to edit or delete any of the entries.



Hotel entries in admin section.

Add Hotel

Hotel Name	Address	Contact Info	Rating
Hilton	150 ABC Street,San Jose,CA,95112	9891121122,hilton@hilton.com	4
Westin	200 Bay,San Jose,CA,97667	6556656565,westin@westin.com	5

Car entries in admin section.

Add Car

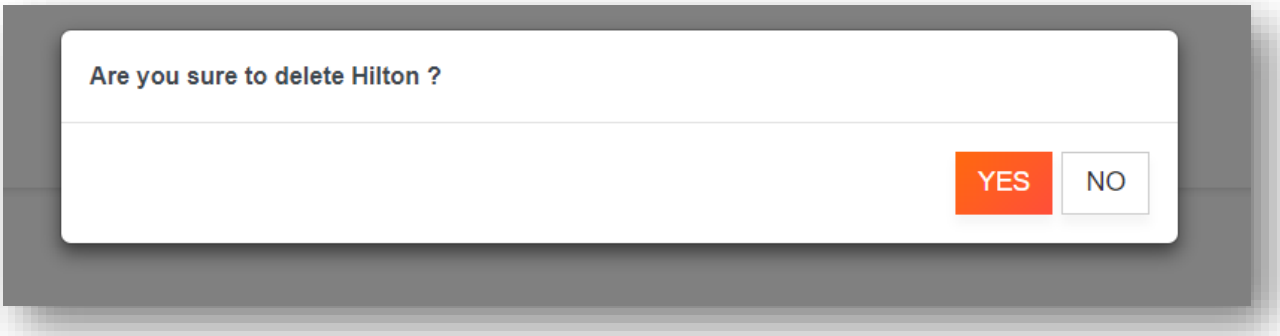
Car Name	Type	# of Cars/day	Rental Value
Honda12	Standard	21	22

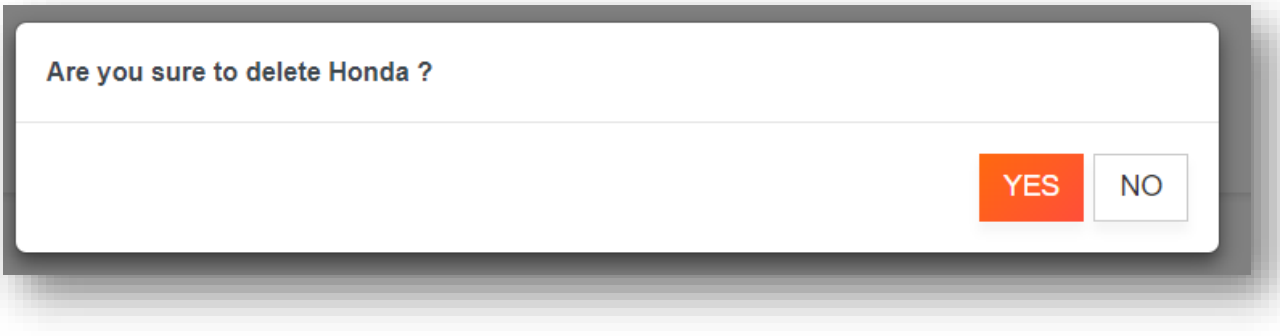
Registered User info in admin section. The admin will also have option to search users using the search box.

Search

	Name	Address	Contact Info
	Aman Ojha	0	aa@aa.comnull

Delete Options





Billing section in admin. All the past billing information can be found in the billing section.

Bill Id	Booking Id	Listing Type	Total Amount
5a245b967a70d4820427d684	5a245b967a70d4820427d683	flights	1
5a245c077a70d4820427d687	5a245c077a70d4820427d686	cars	25

After clicking on a bill, Admin will further see details of a bill.

Billing Information

Bill Id	5a245b967a70d4820427d684
Billing Date	2017-12-03T20:16:22.274Z
User Name	Aman Ojha
Total Amount	1

Booking Information

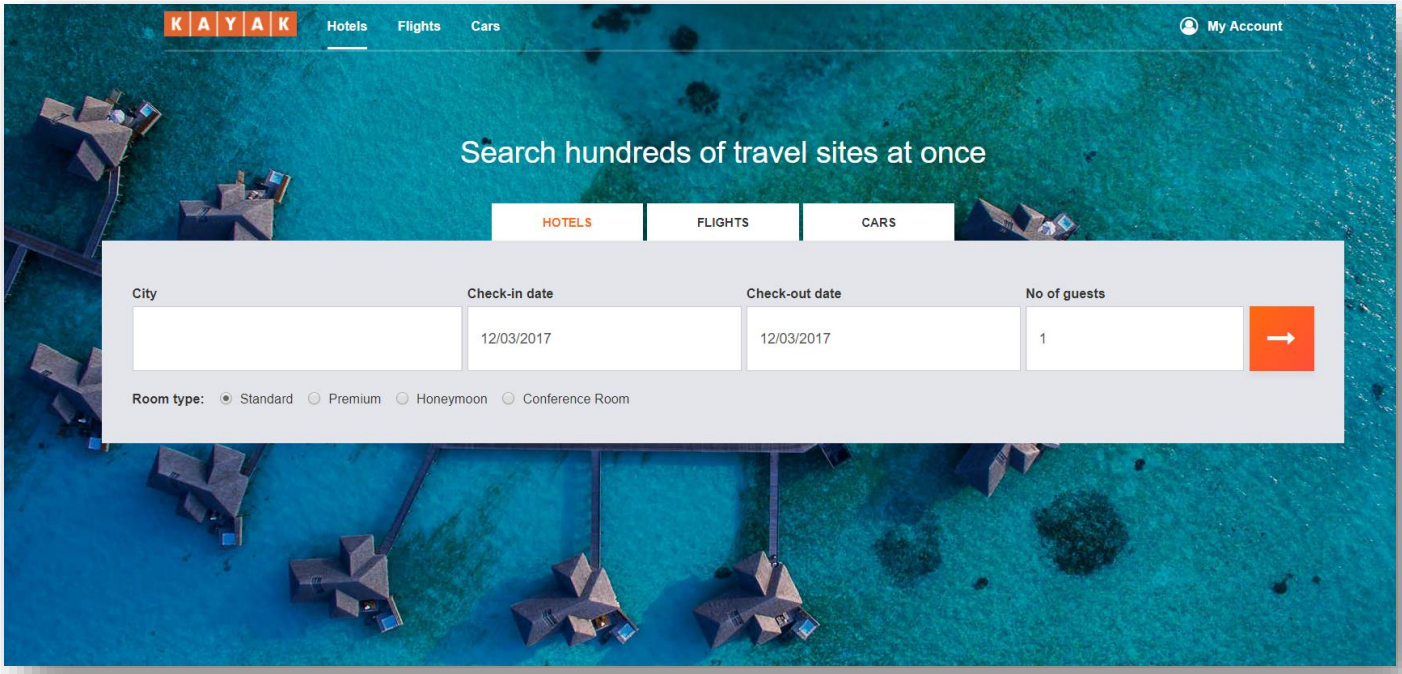
Booking Type	flights
Passengers's Email	aa@aa.com
Passengers's Contact Number	1212121212
Flight Source	San Jose, California
Flight Destination	Los Angeles, California
Flight Cabin	Economy
No of Passengers	1
Journey Date	12-05-2017

Close

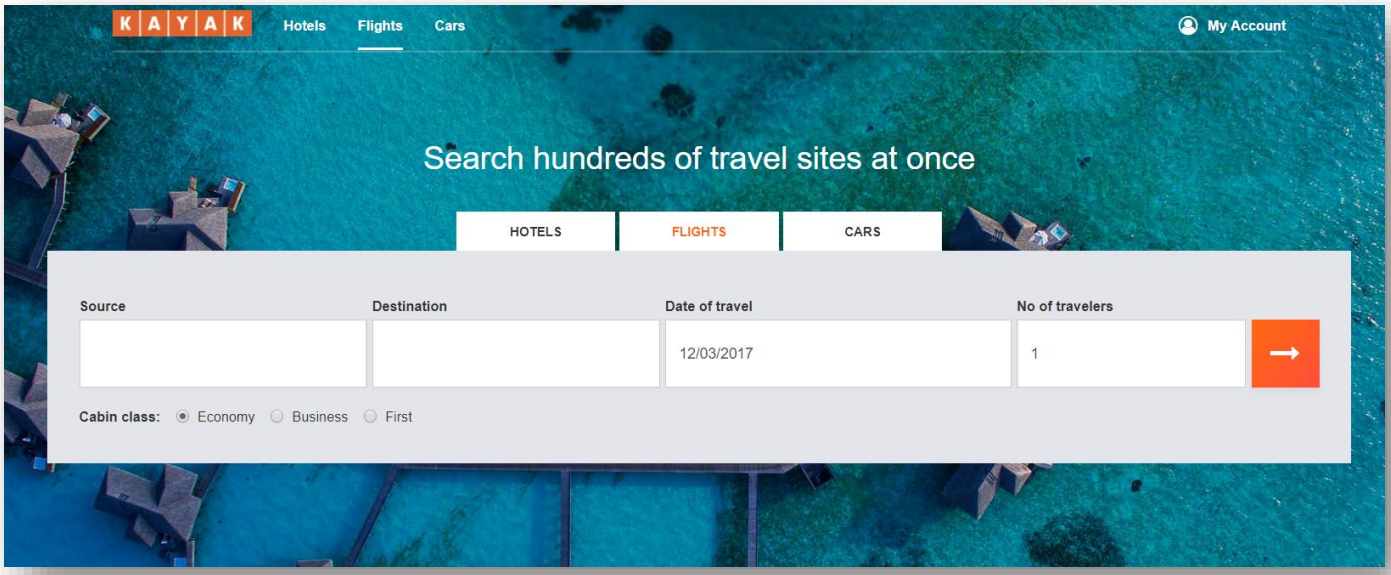
CUSTOMER APPLICATION

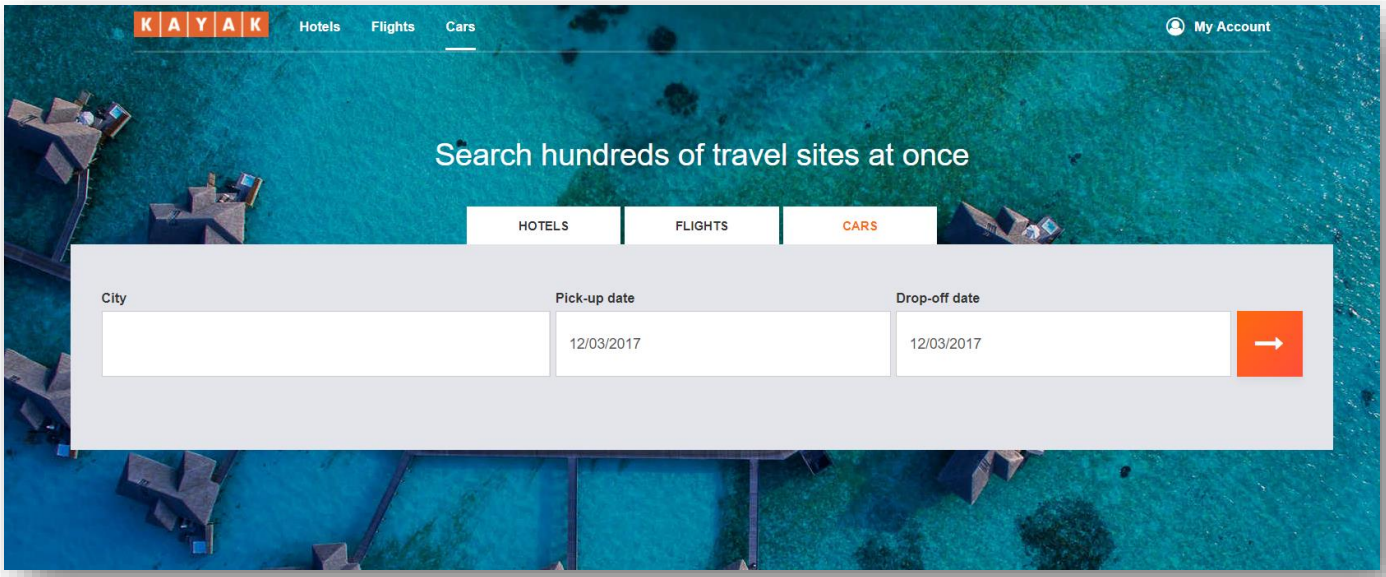
Client Section

The first page all the clients will be greeted with is shown below.

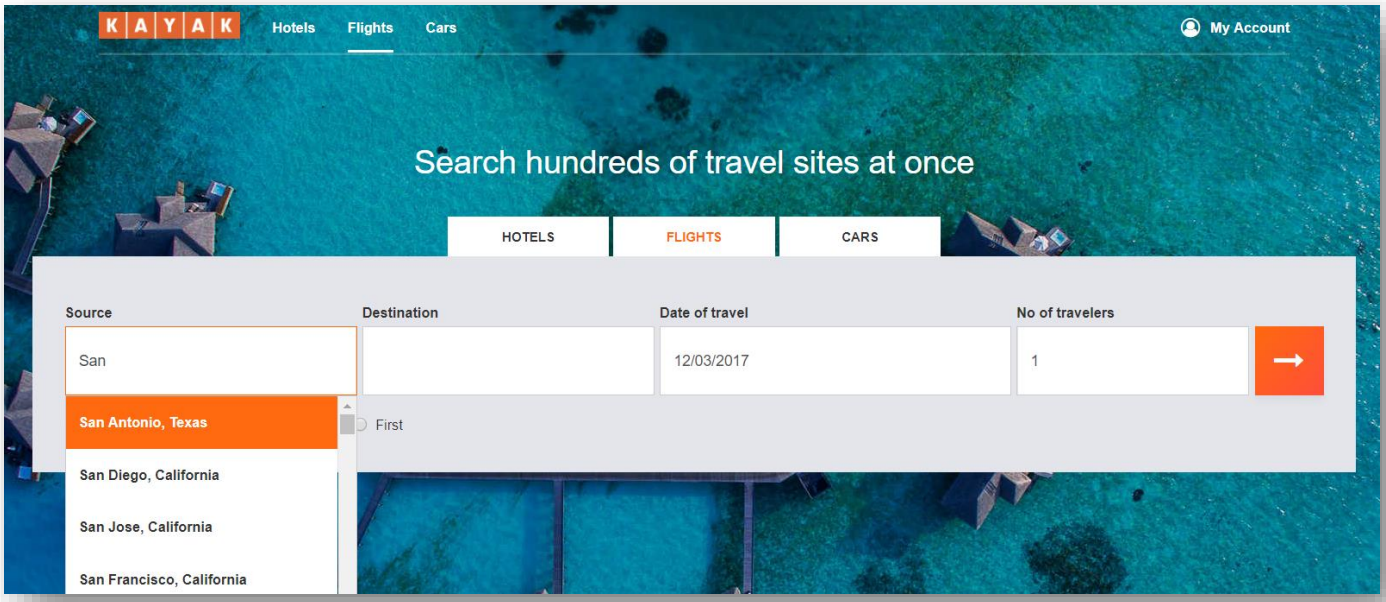


All the users will be able to book hotels, flights or cars.

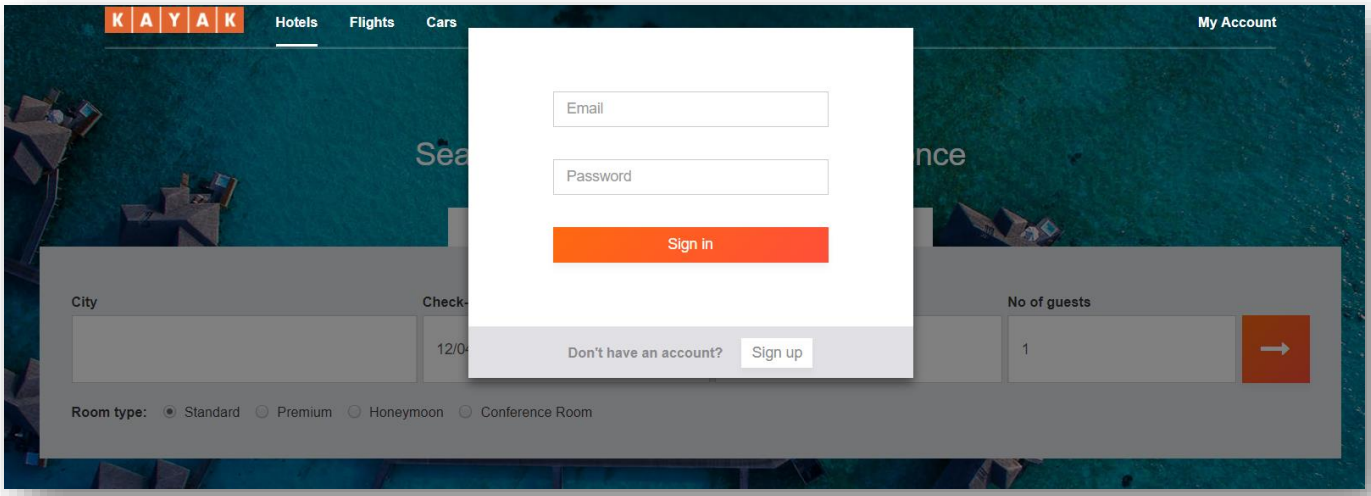


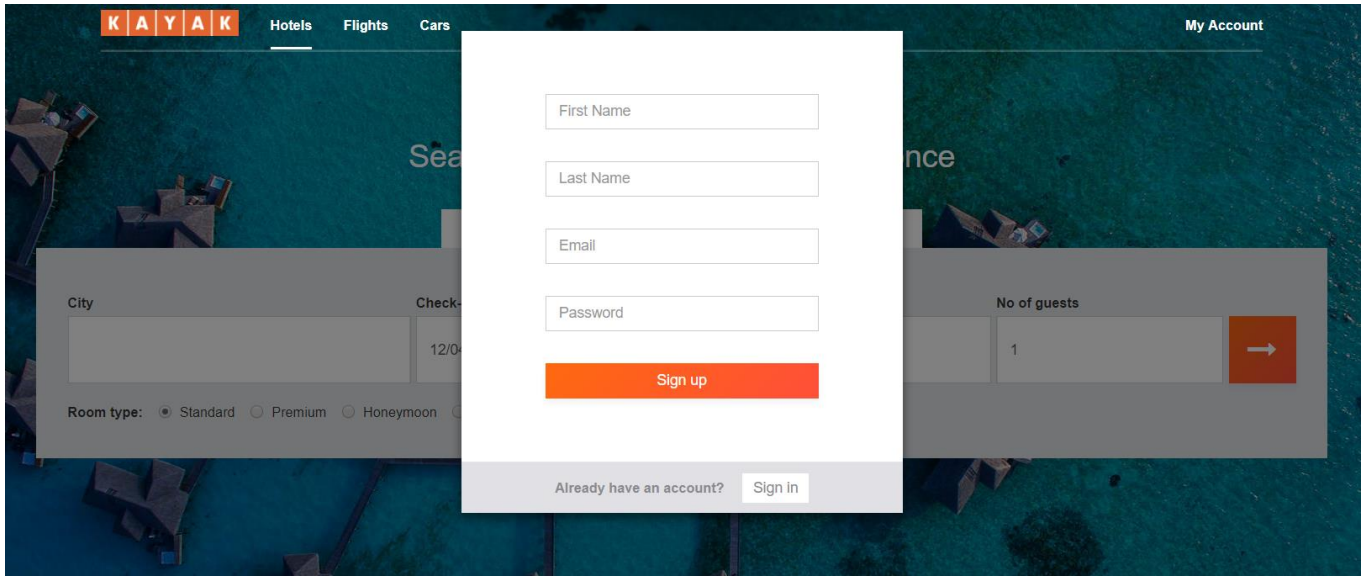


Also, search suggestions will be provided to the users to help them select the desired city easily.

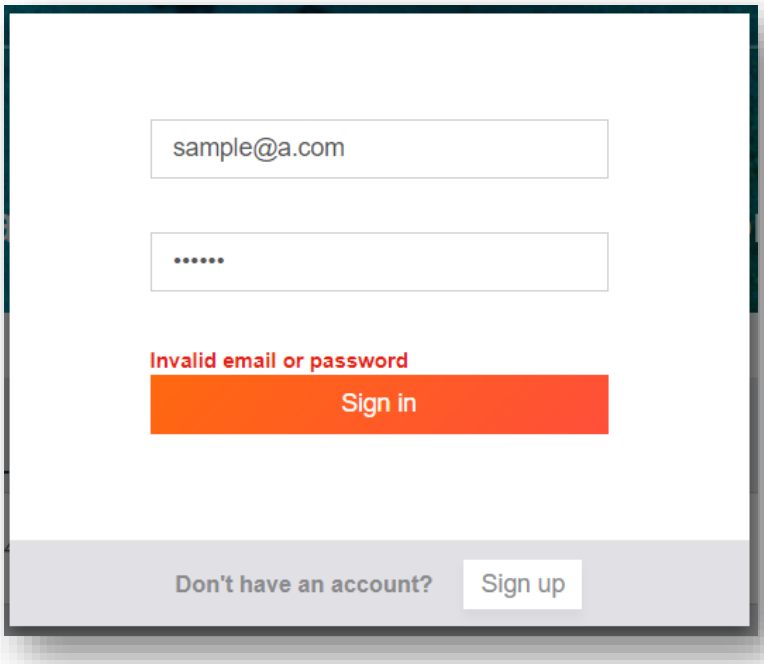


A user will also get an option to login or signup.

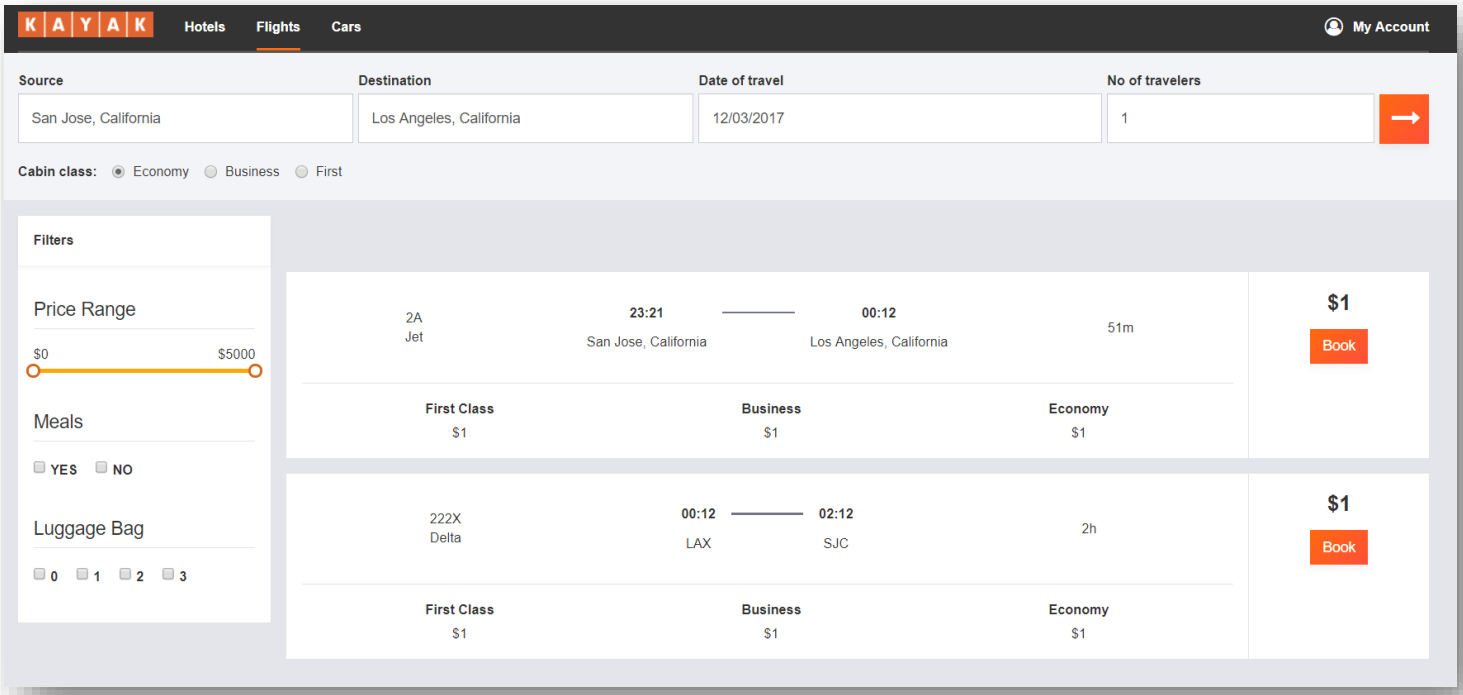




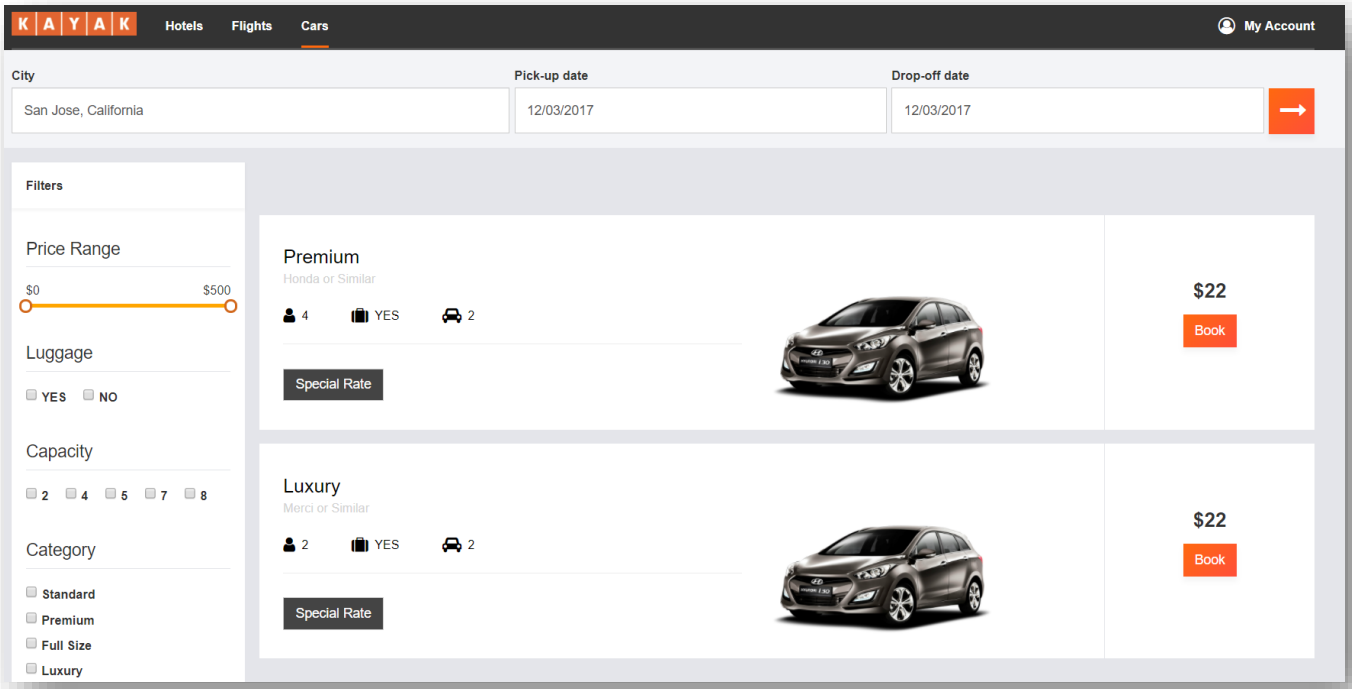
Error on wrong login credentials



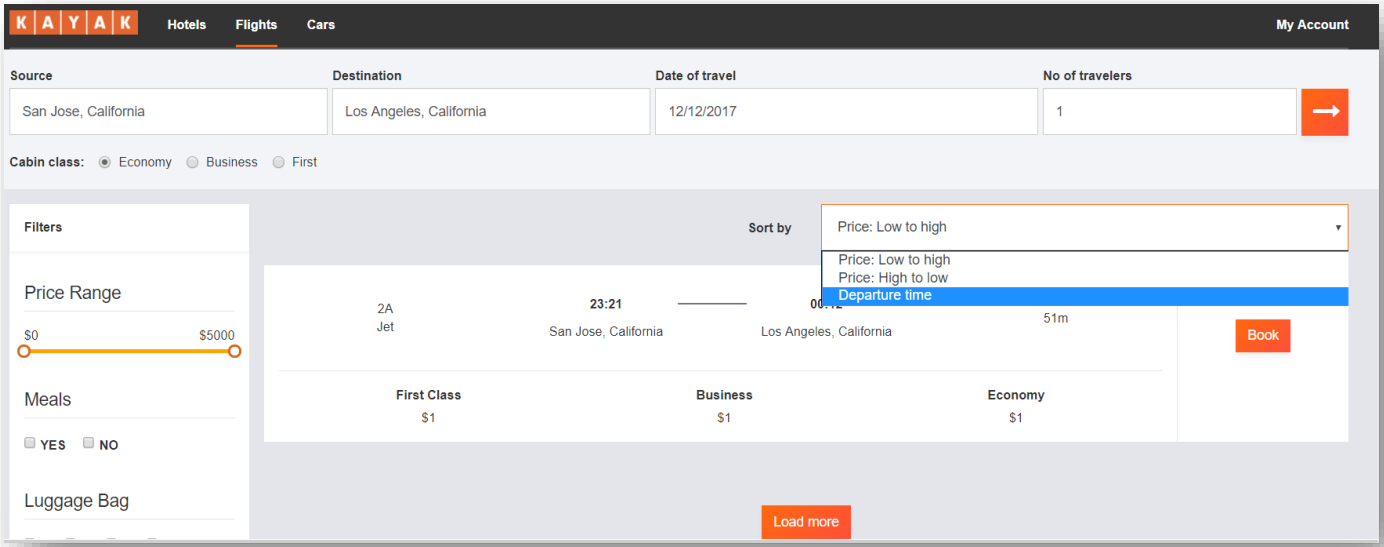
After selecting the desired cities and date of travel, the users will be shown the available options they have for booking.



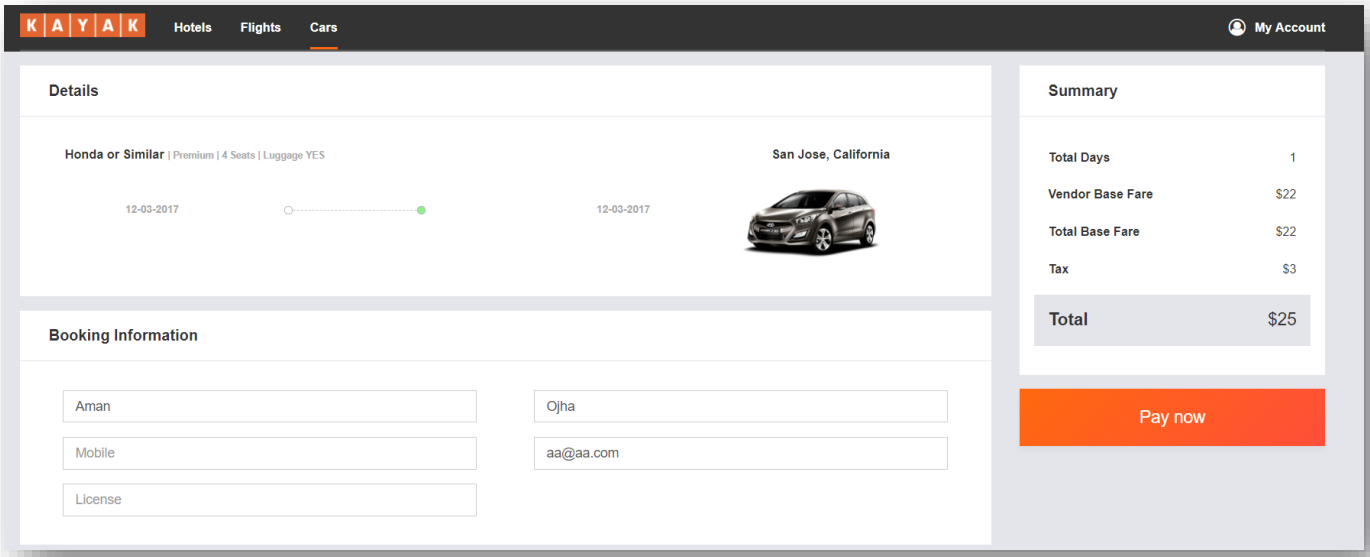
All the users will further have option to filter their search based on price range, or various other amenities available in the hotel, flight or car.



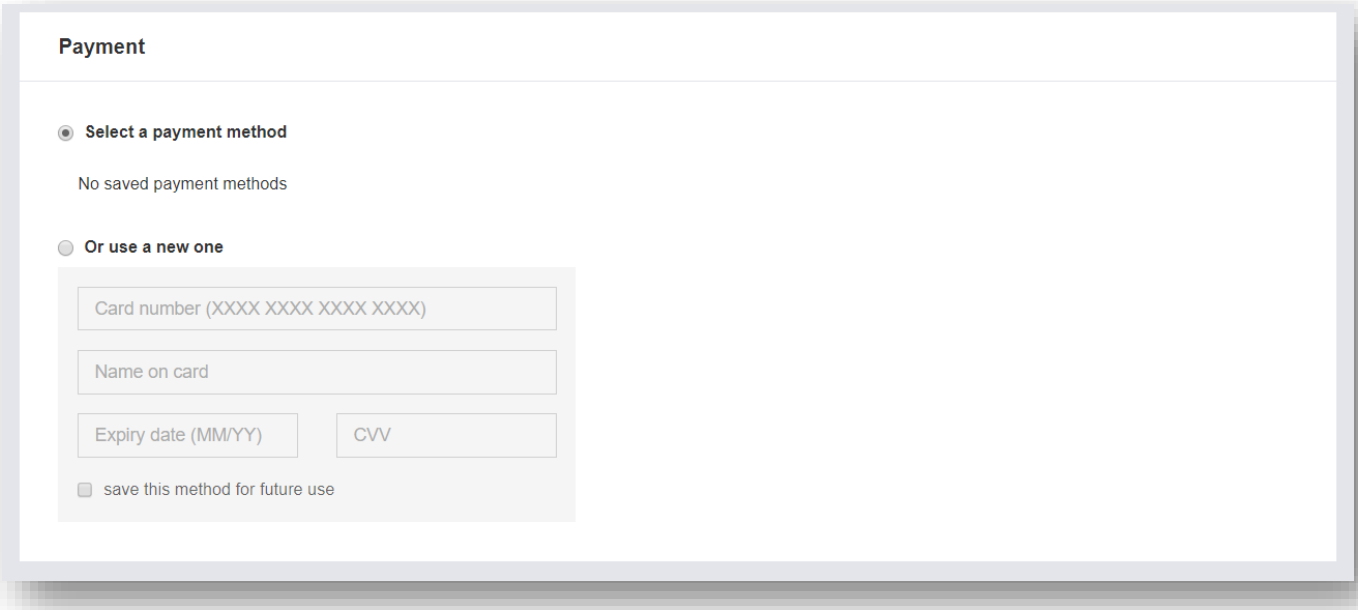
A user will also get an option to sort the available listing based on multiple parameters. Also, when the available listings are more than 20, we will get an option of 'Load More' to see more available options.



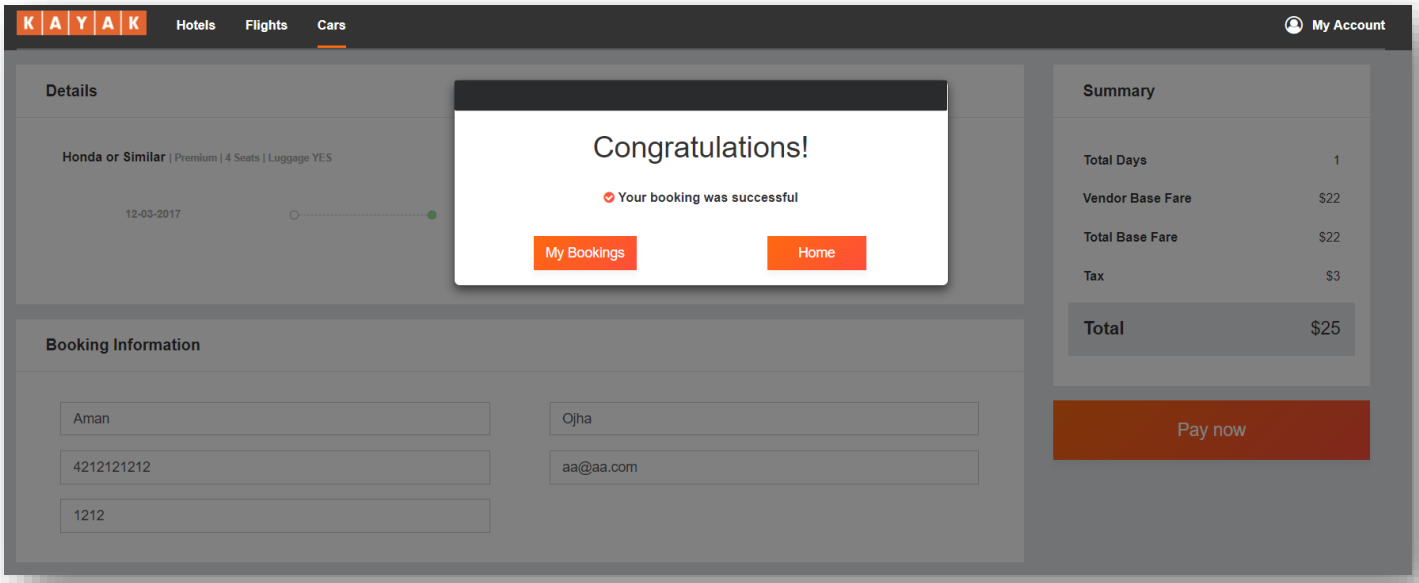
The user can select any of the available option and can further proceed for billing.



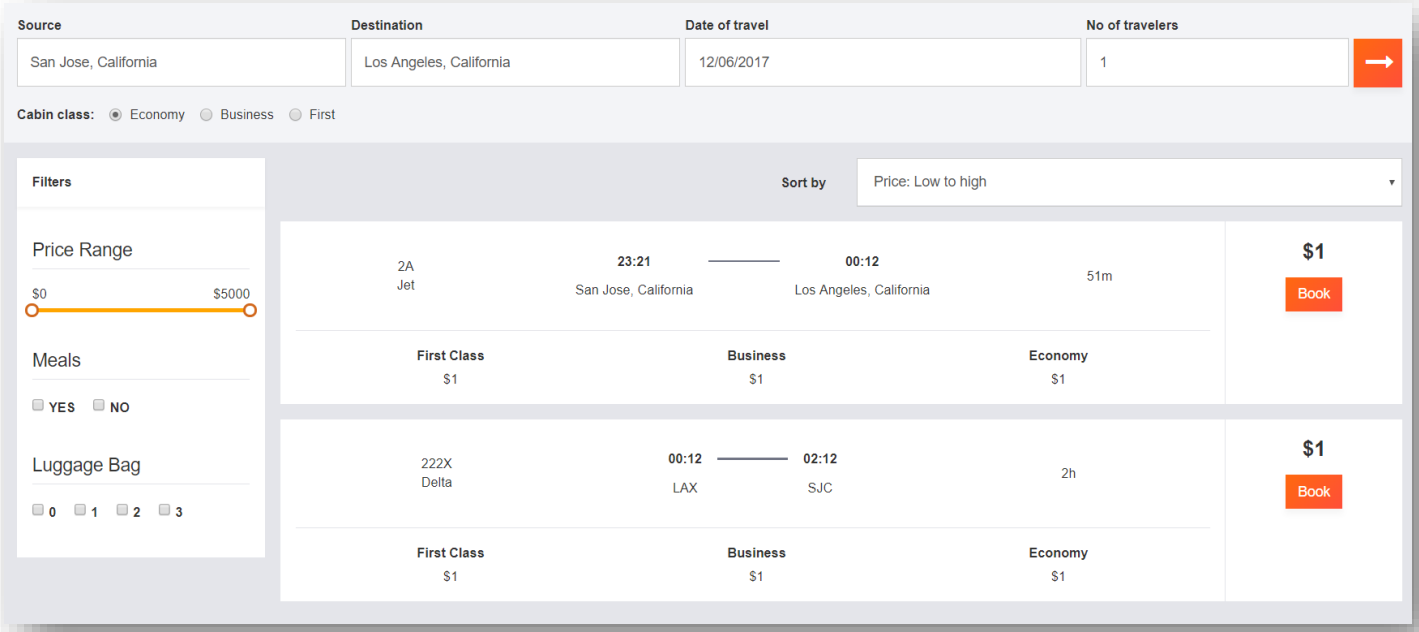
Payment Options



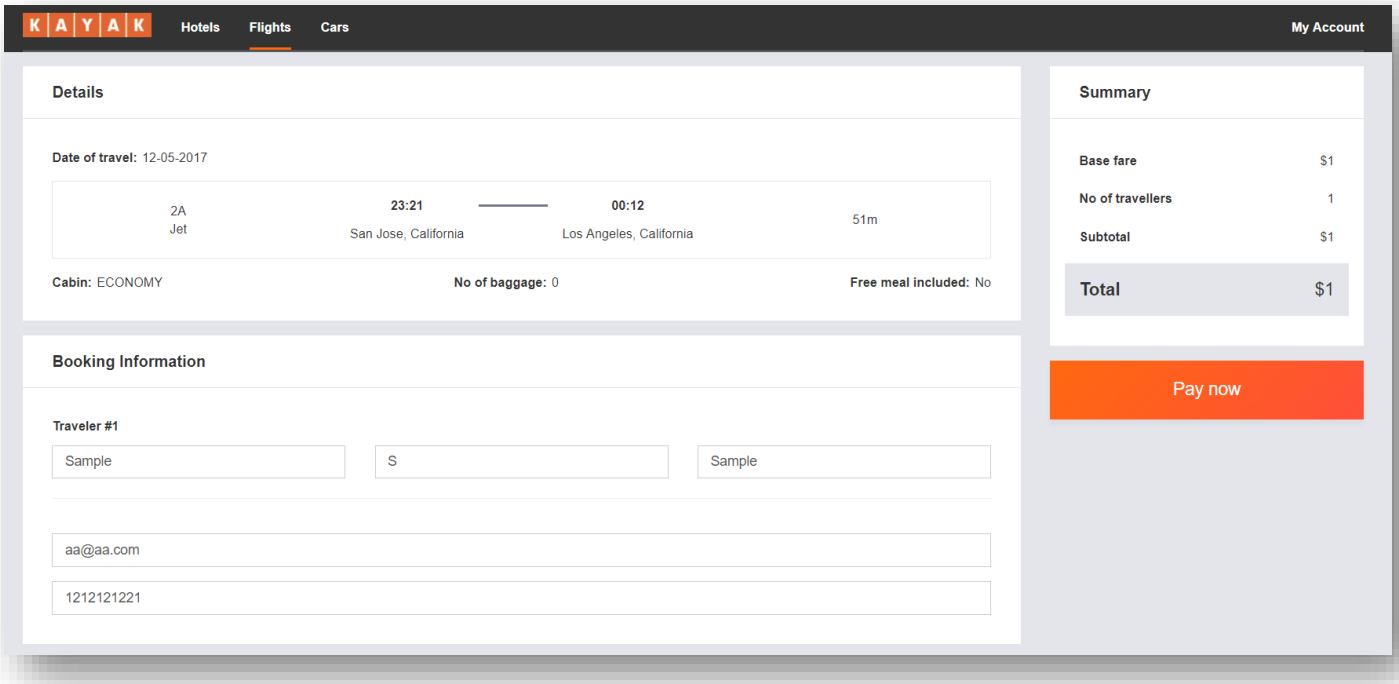
After filling the form with all correct information and selecting the pay option, the users selected option is booked and billed.



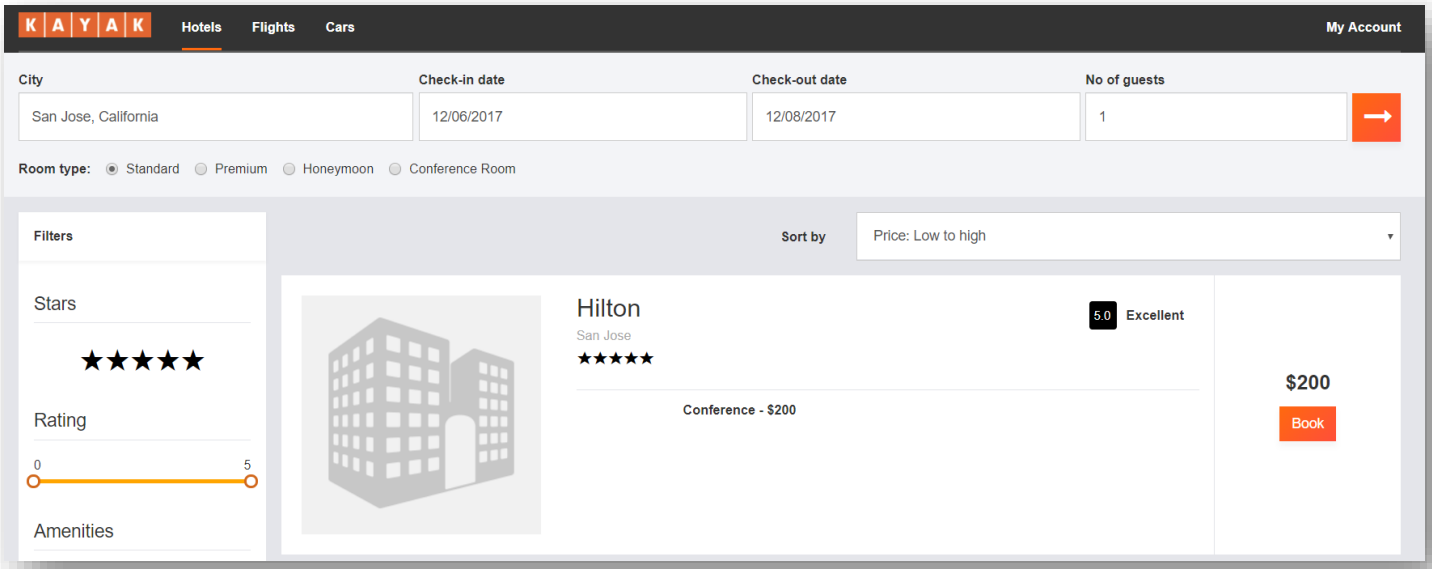
Flight Listing page



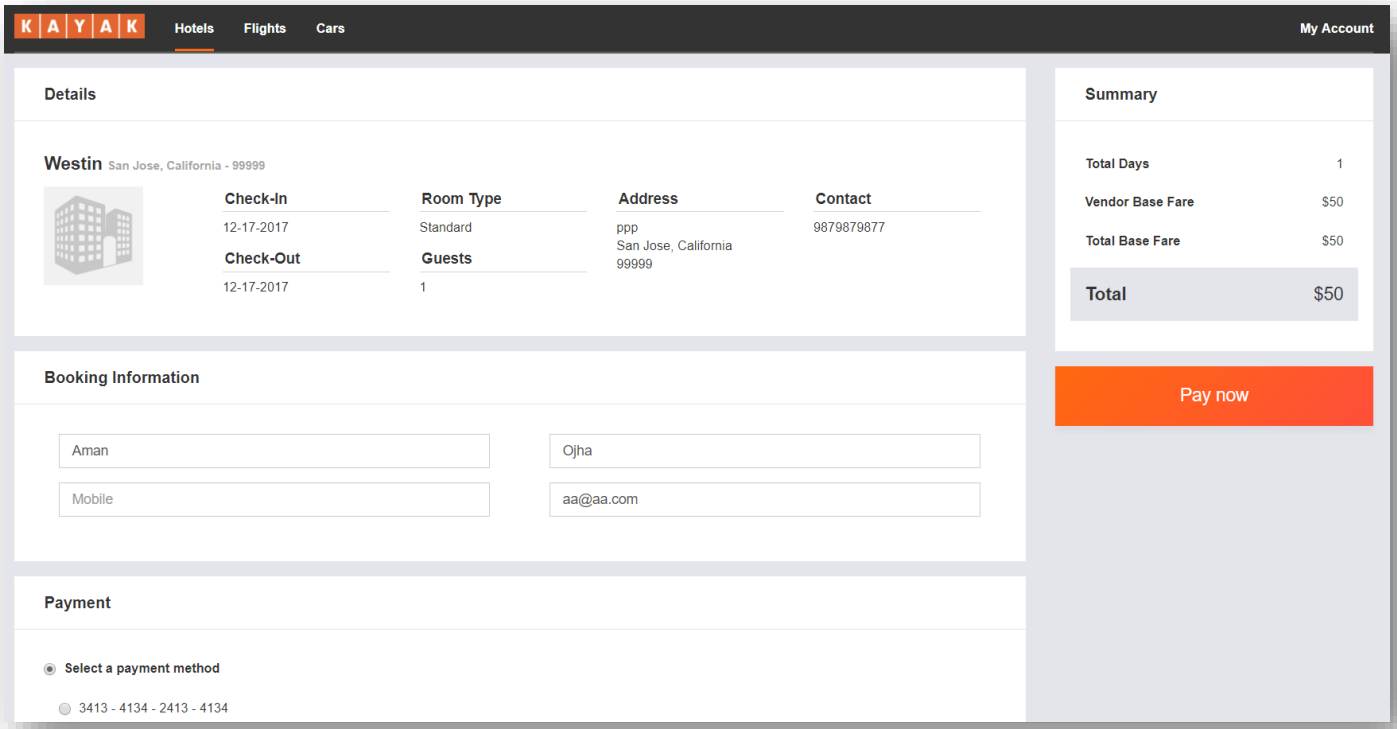
Flight Booking Page



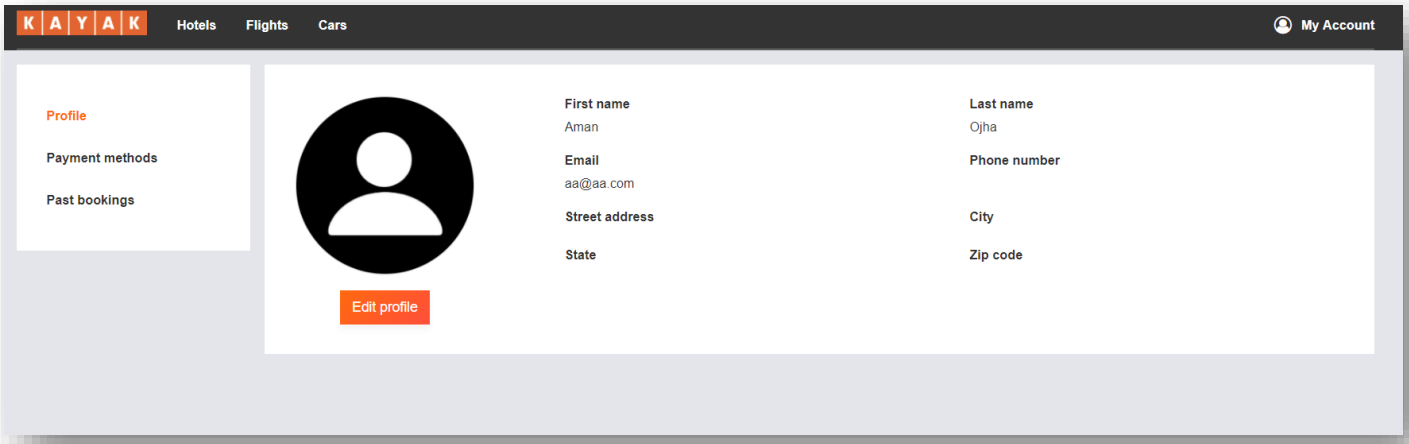
Hotel Listing Page



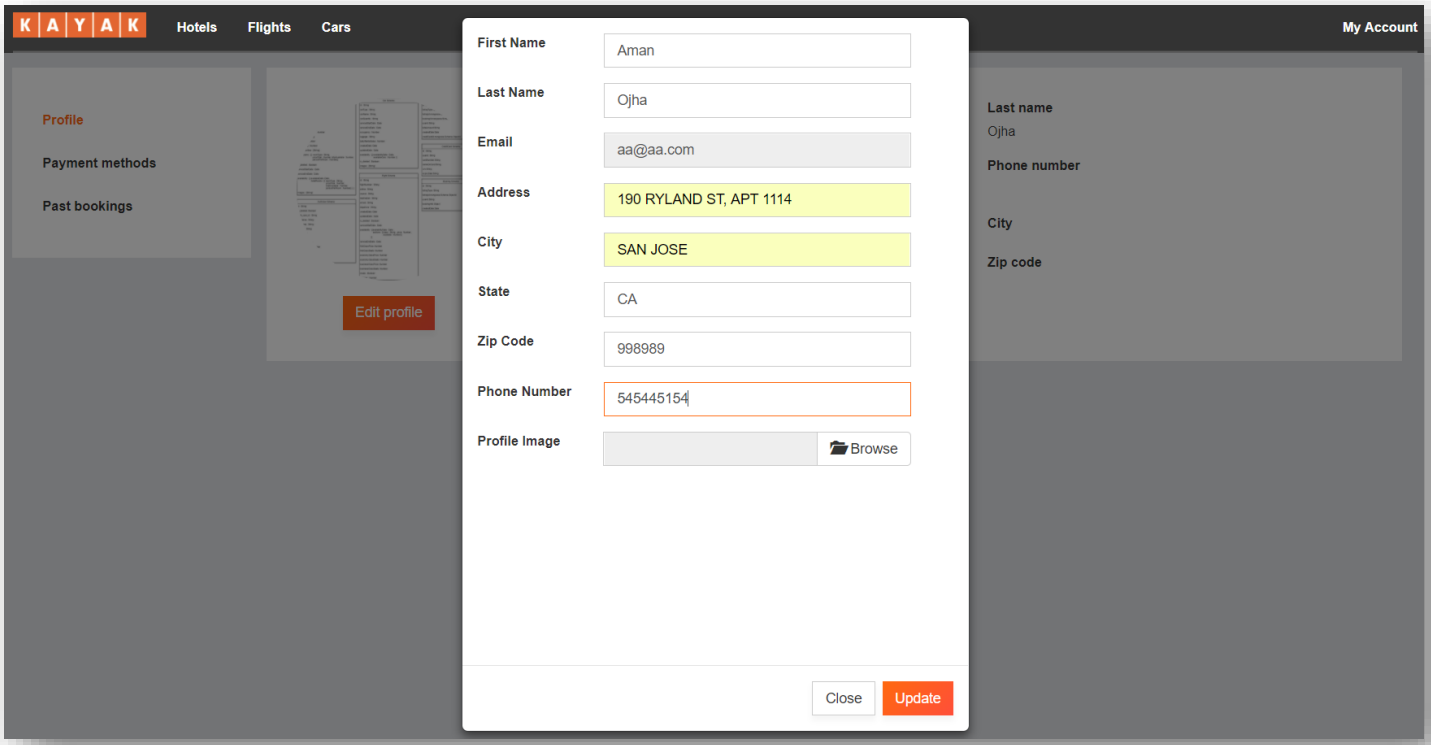
Hotel Booking Page



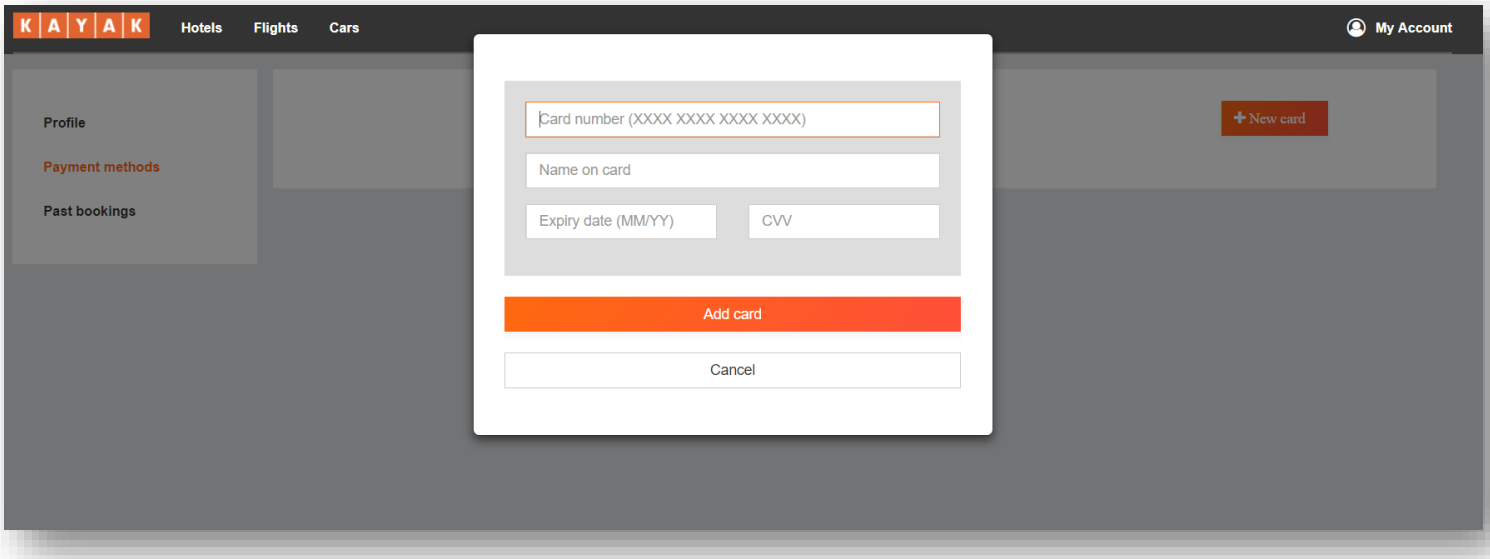
The user will also have option to check and update his profile.



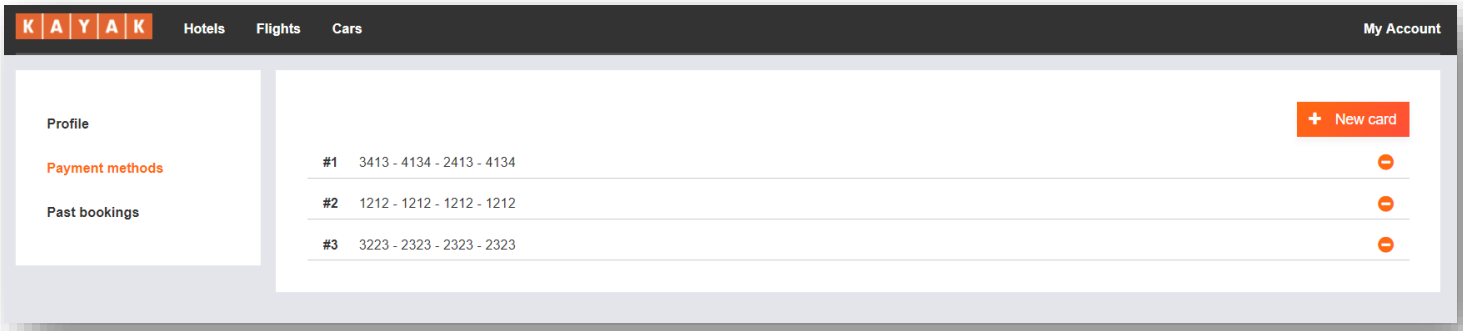
A User will also get option to edit his profile



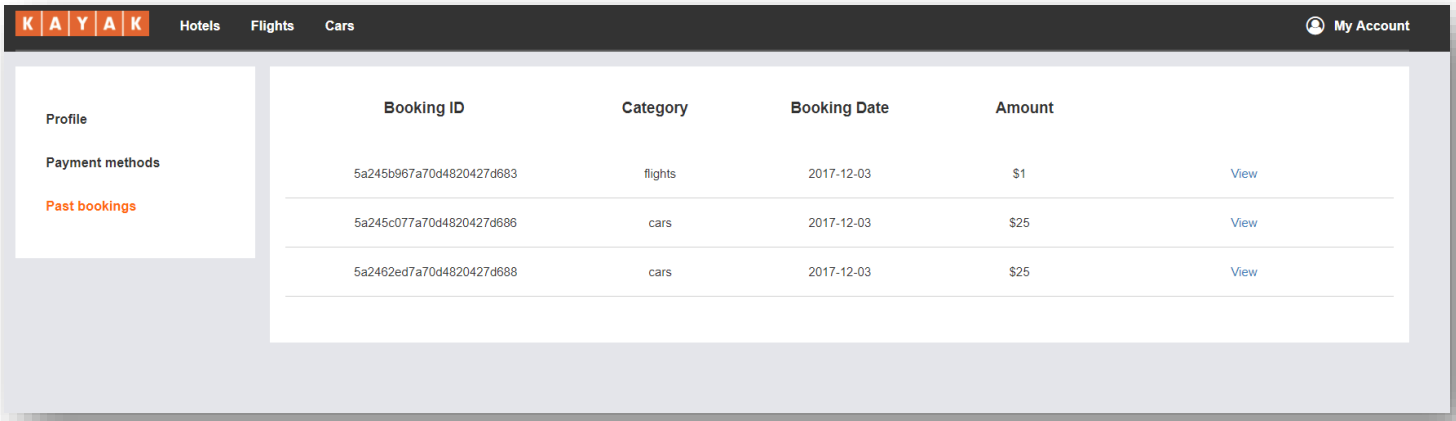
A user will also have option to add different payment methods.



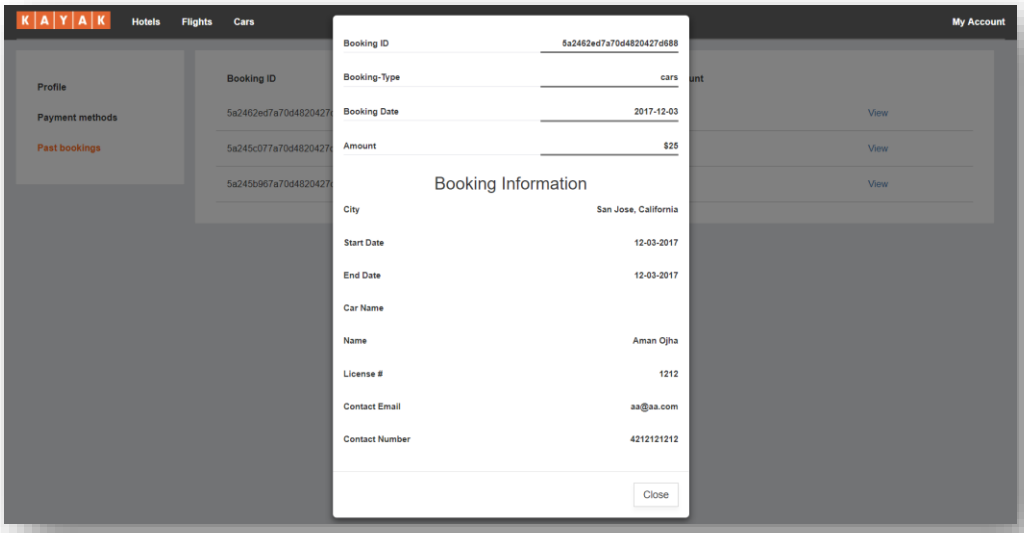
Listing of all added payment options



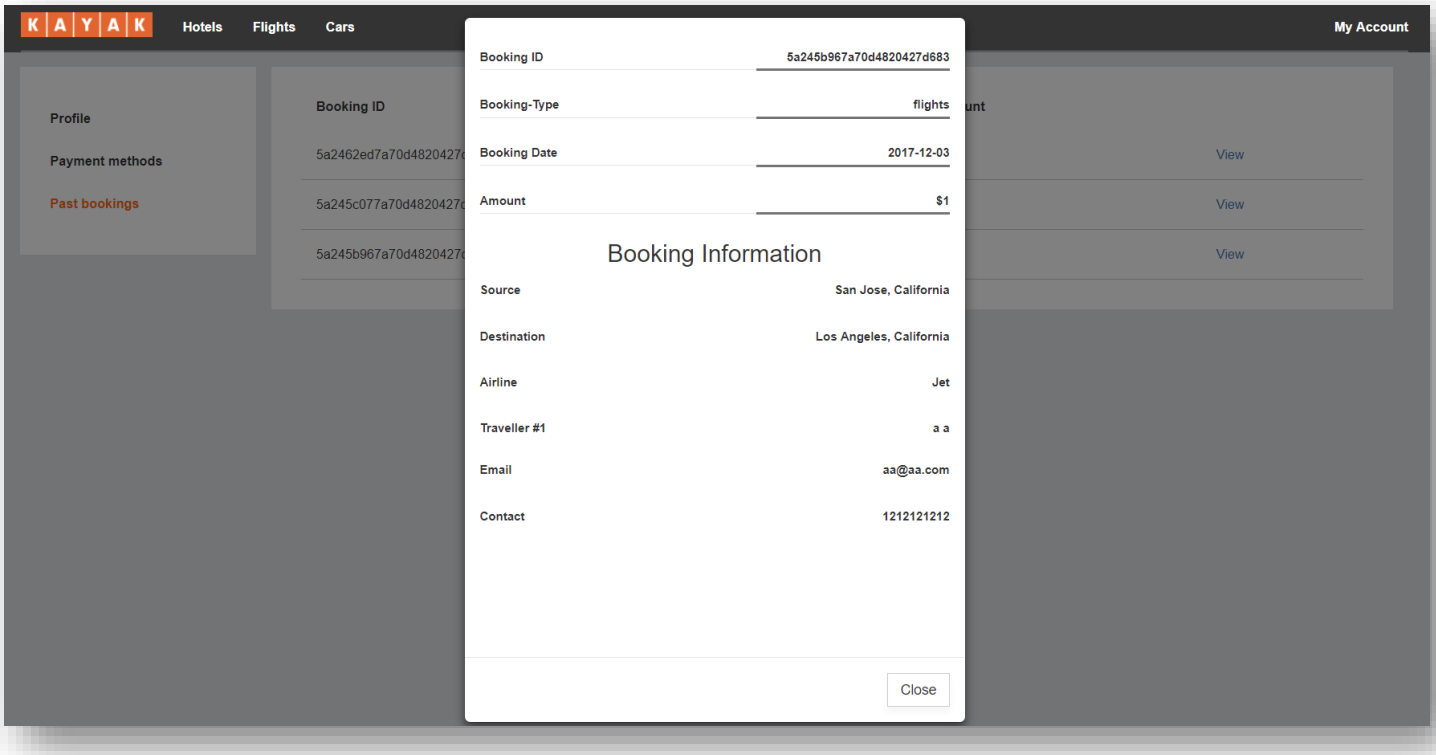
All the previous booking history can be viewed on the Past Bookings tab.



Car Past Booking receipts



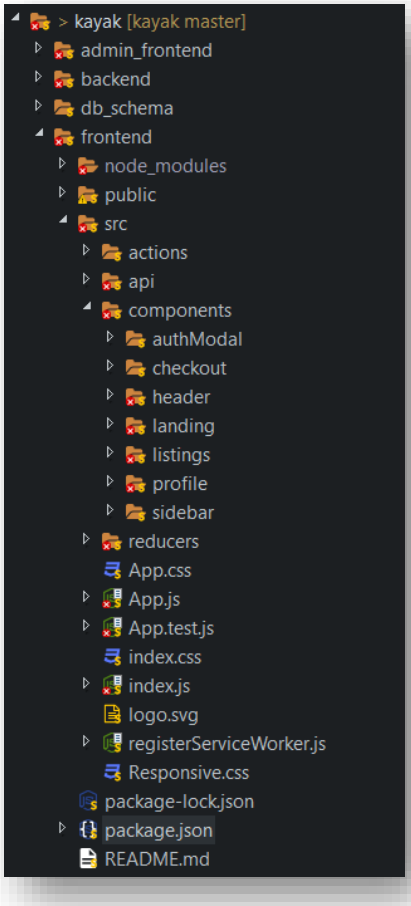
Flight Past booking receipt



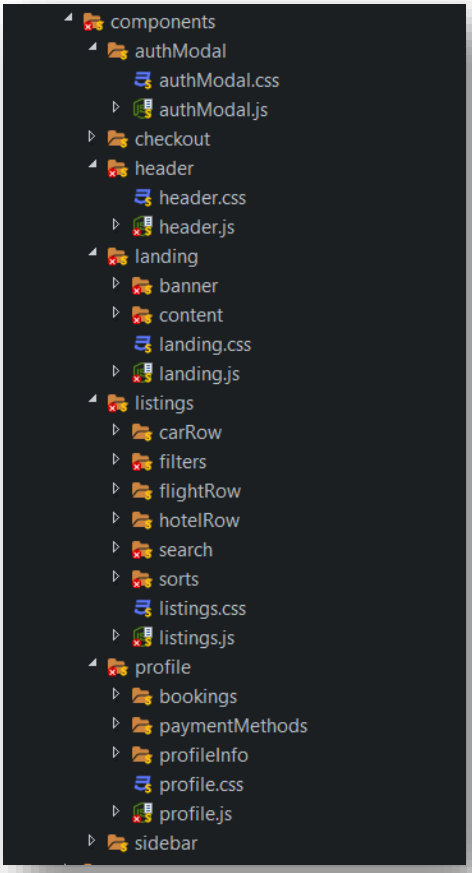
CODE LISTING

Client Frontend

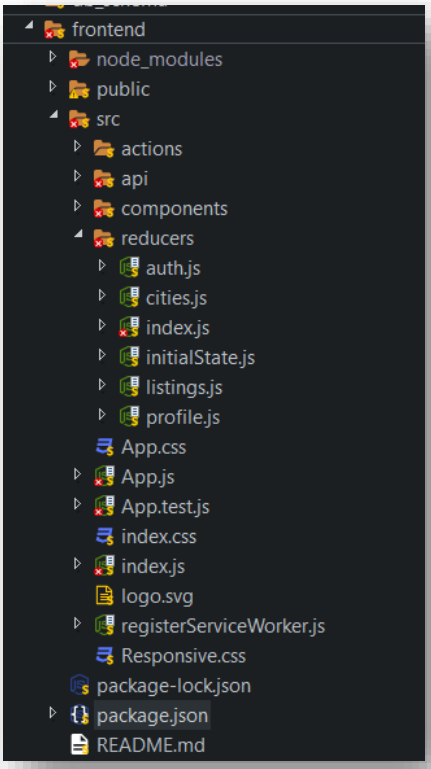
Base directory structure



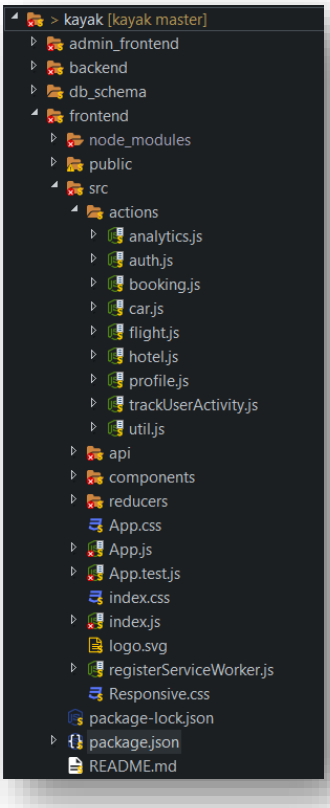
Components in frontend



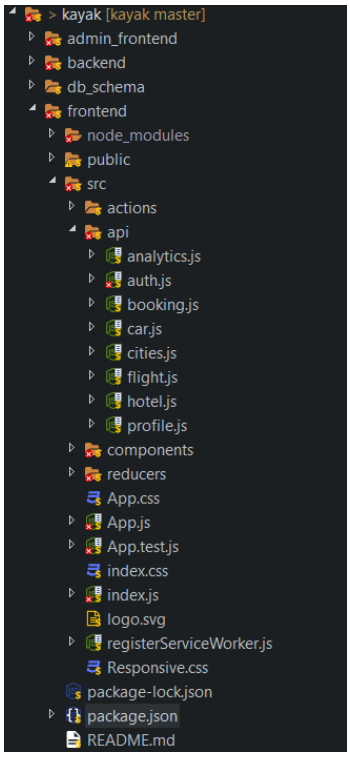
Reducer



Action Functions

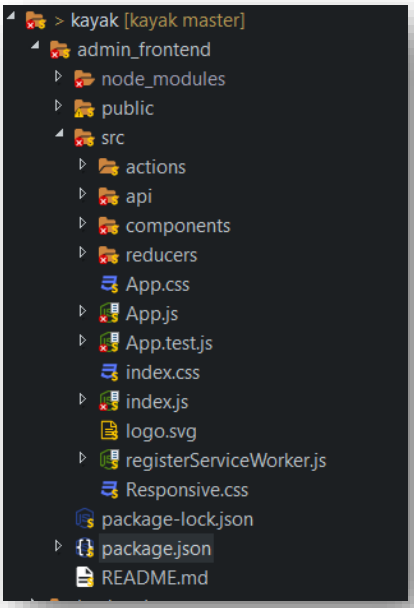


APIs

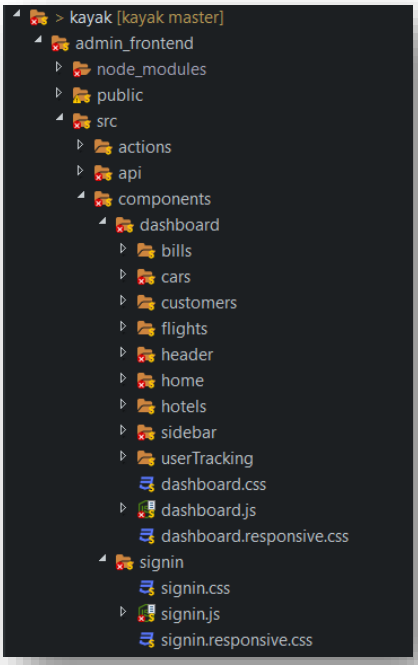


Admin Frontend

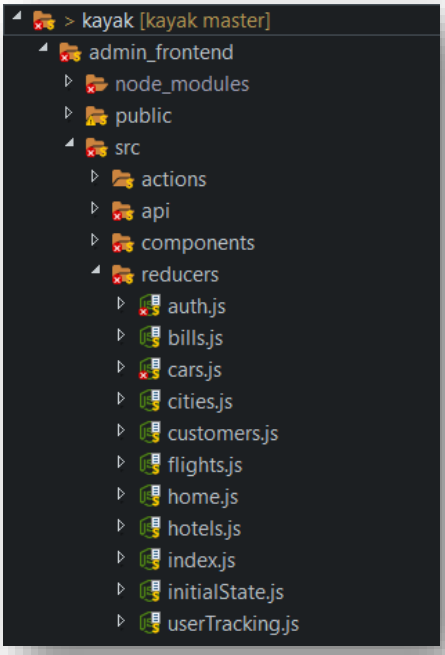
Base directory structure



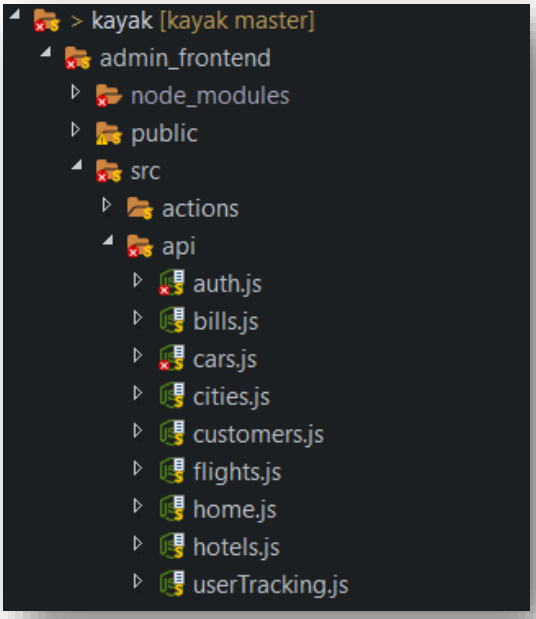
Components in admin



Reducer

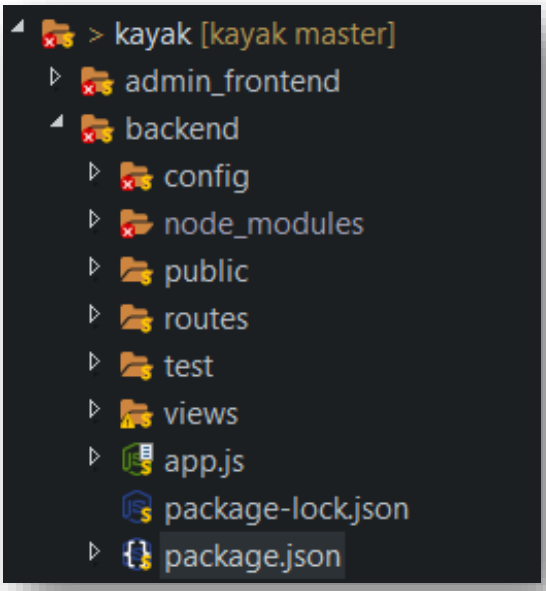


APIs

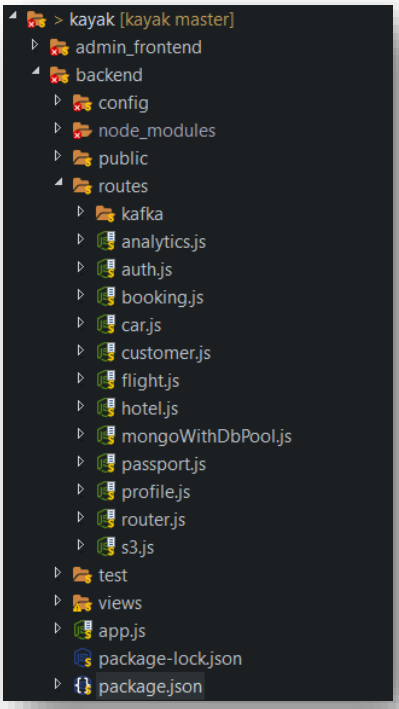


Node Backend

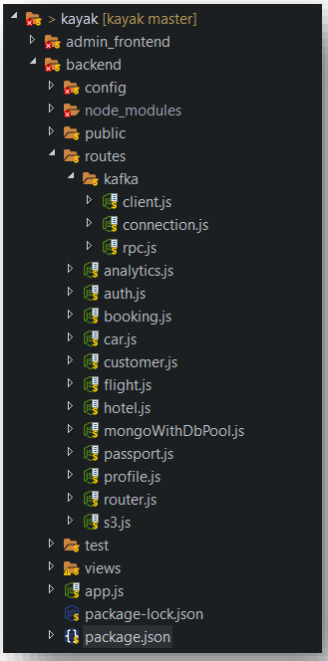
Base directory structure



Routes

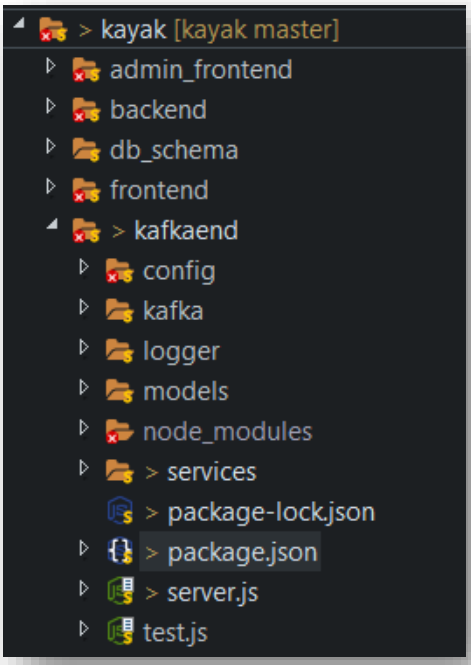


Kafka routes

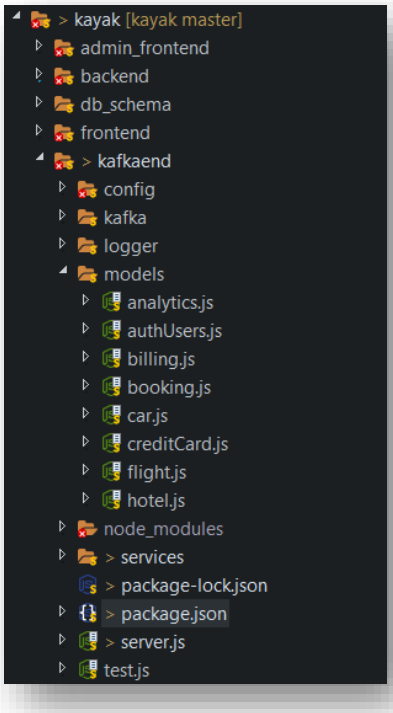


Kafka Backend

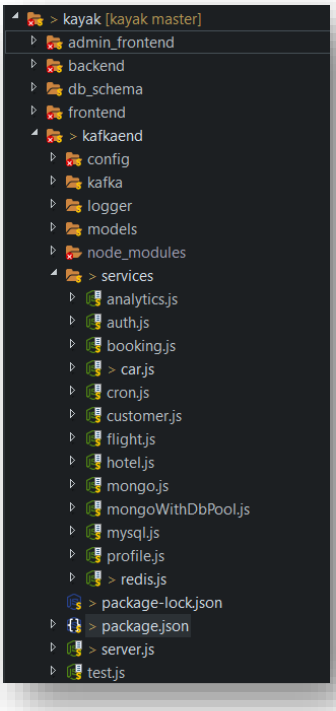
Base directory structure



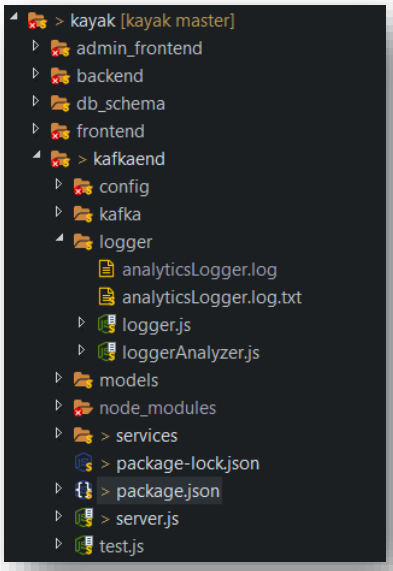
Models



Kafka services

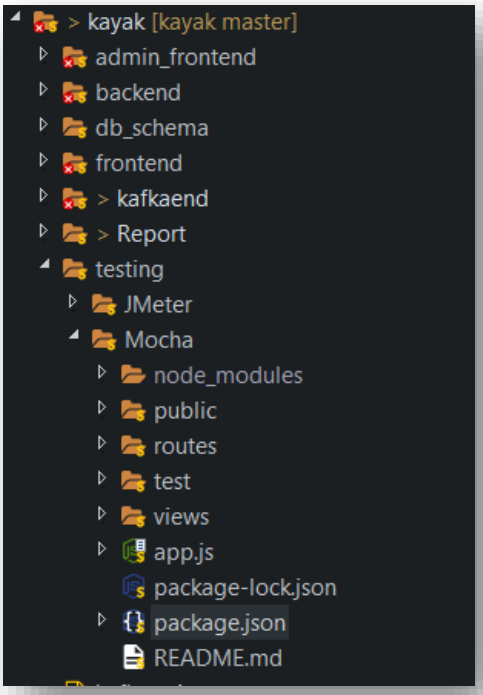


Analytics logging

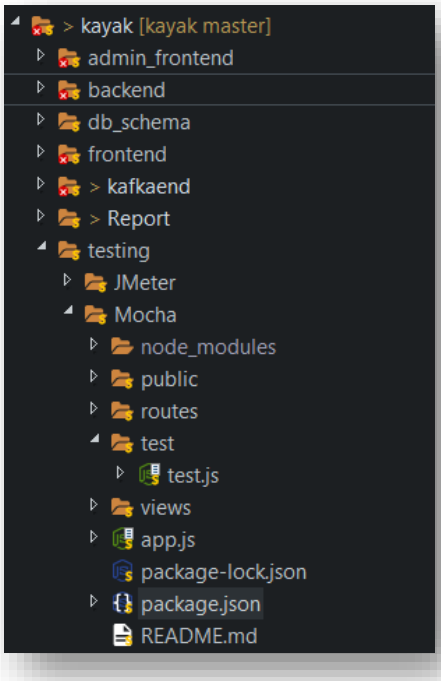


Mocha Testing

Base directory structure



Test Cases

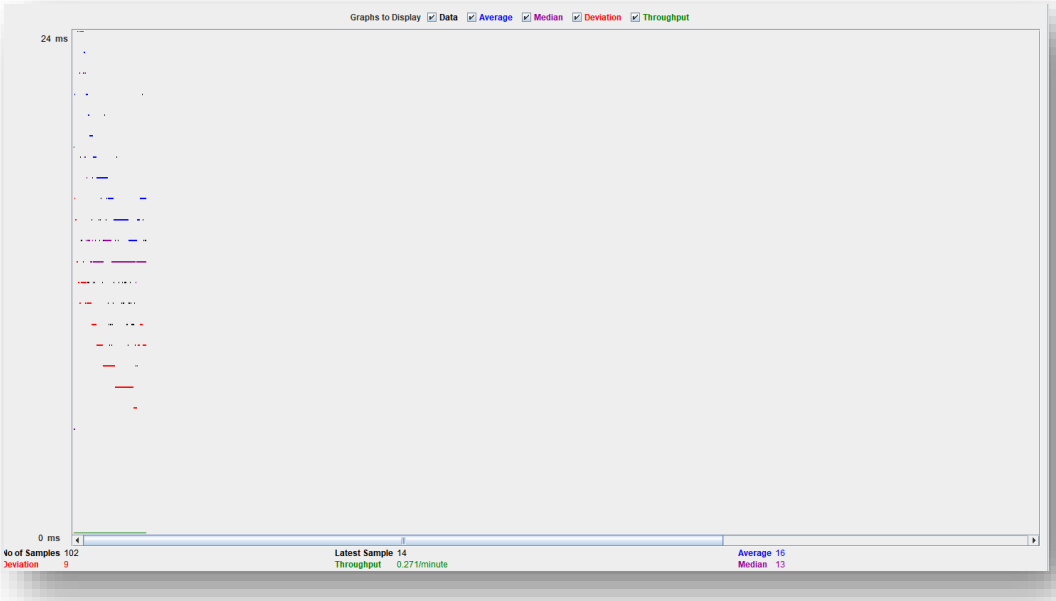


JMETER TESTING

Apache JMeter is a testing utility which is commonly used as load testing tool for analyzing and measuring various aspects of a project. It is primarily focused on testing web applications. JMeter offers a variety of connection policies like HTTP, LDAP, JDBC etc.

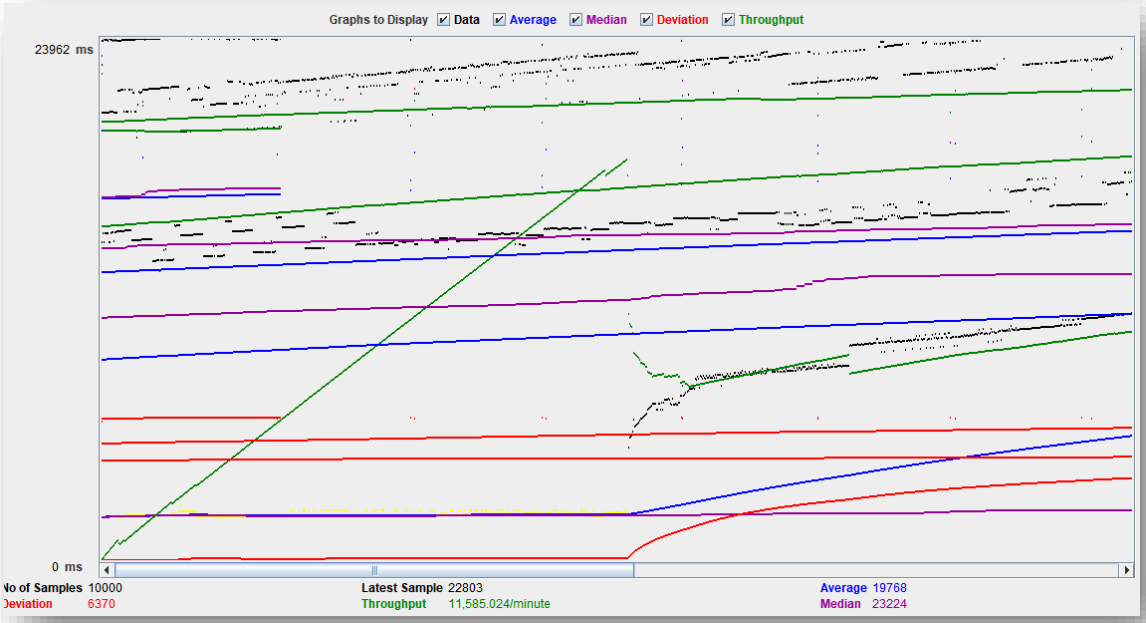
1. Listing

100 Users with average time of 16ms.



1000 User with average time of 3542 ms

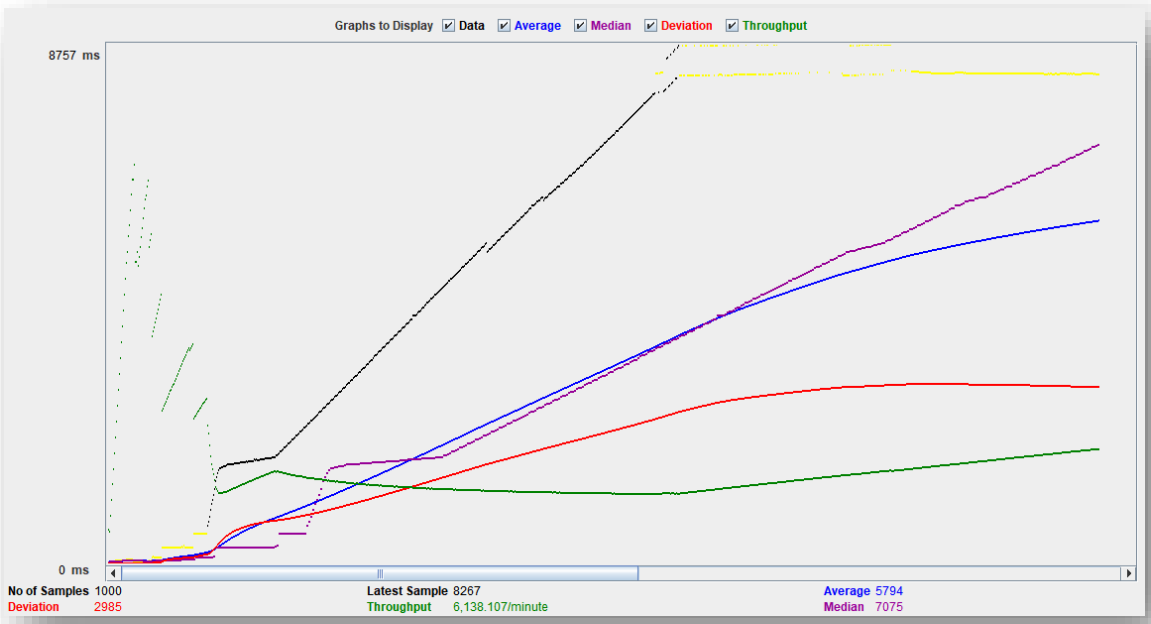
10,000 User with average time of 19768 ms



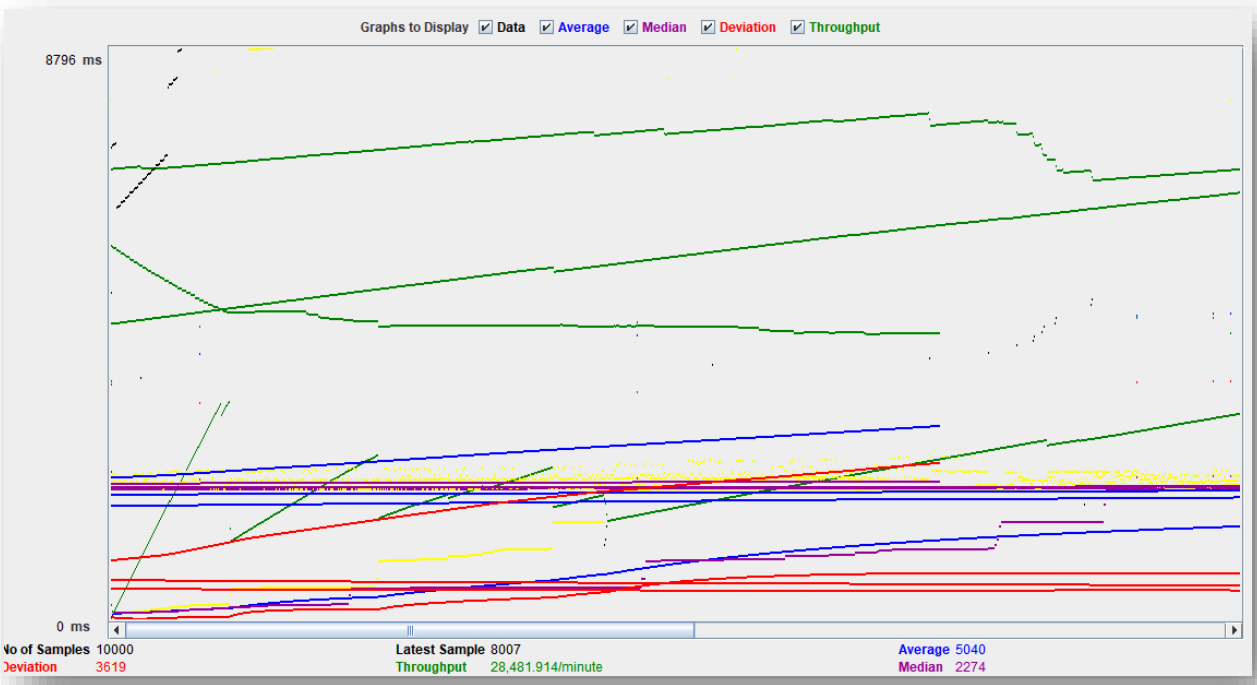
2. User Registration

100 Users with average time of 518 ms

1000 Users with average time of 5794 ms

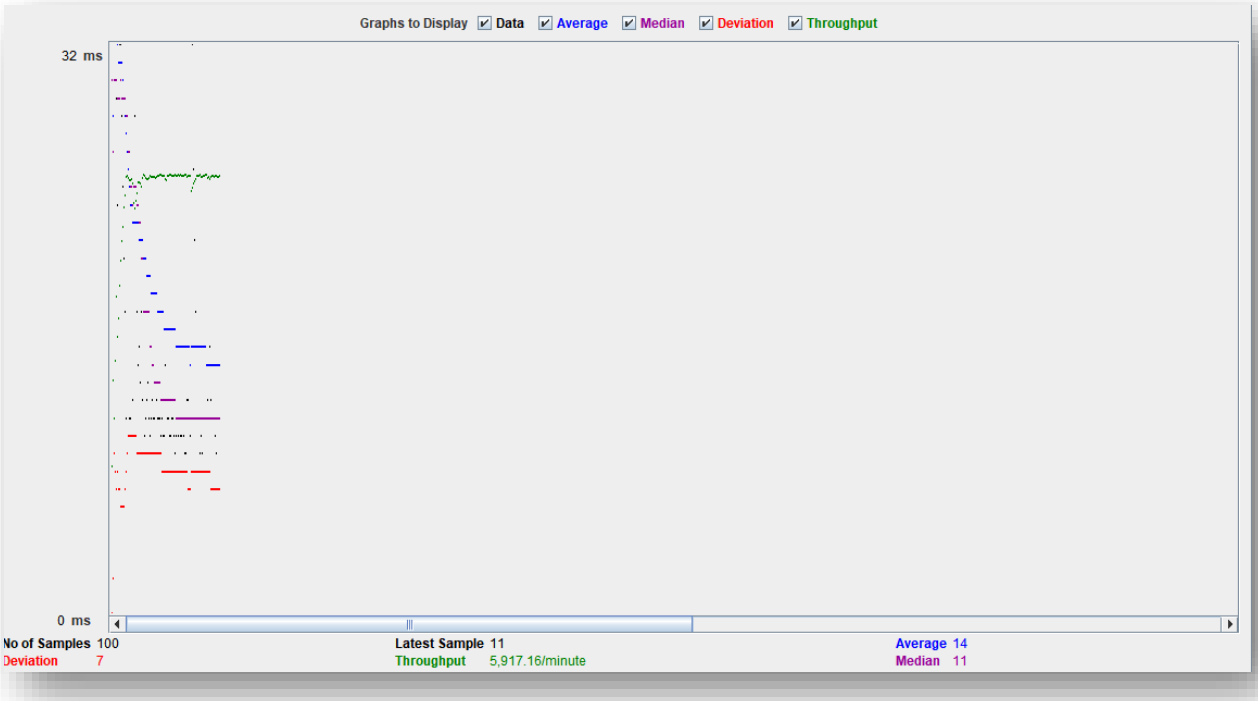


10,000 Users with average time of 5040 ms

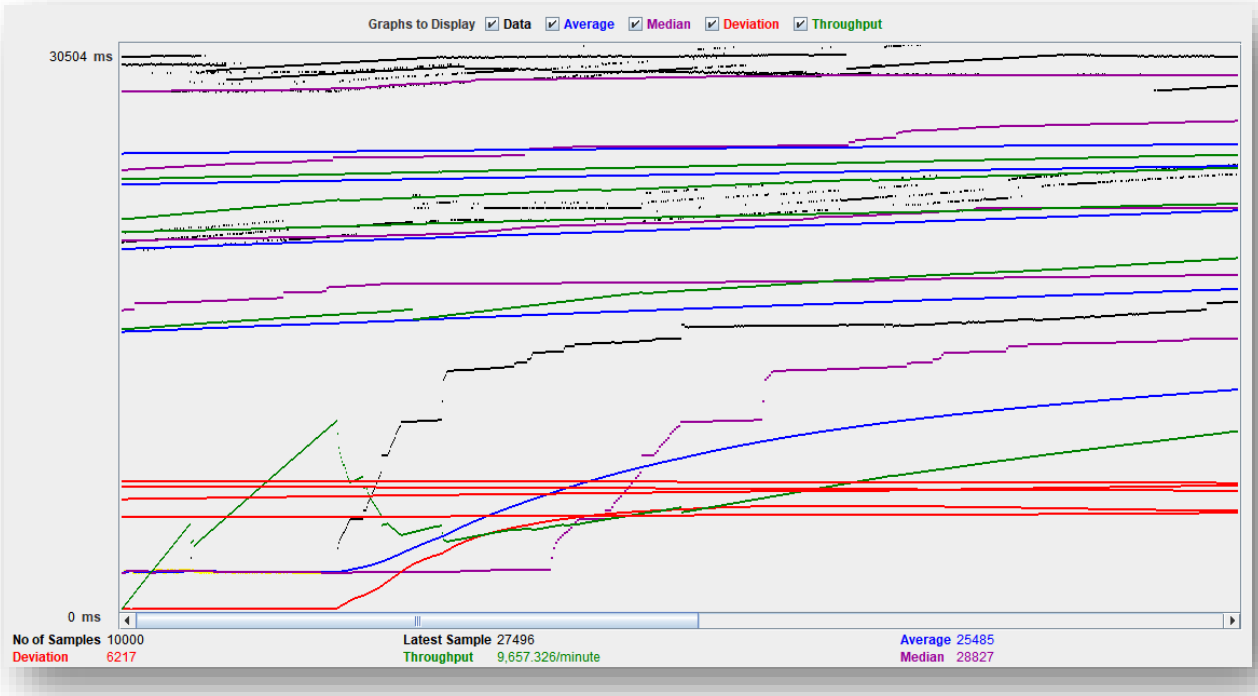


3. Bookings

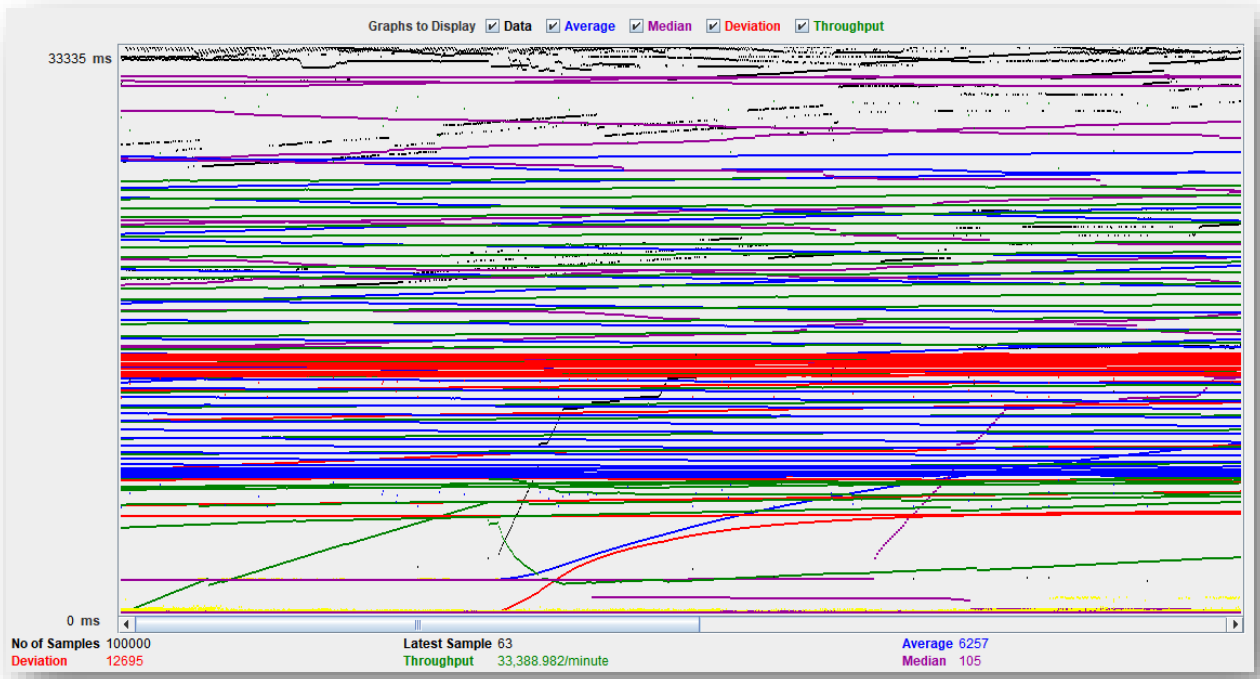
100 Users with average time of 14 ms



10,00 Users with average time of 25485 ms



100,000 Users with average time of 6257 ms



4. JMeter conclusion

Connection Pooling:

A connection pool is a cache of database connections maintained so that the connections can be reused when future requests to the database are required. Connection pools are used to enhance the performance of executing commands on a database. In connection pooling, after a connection is created, it is placed in the pool and it is used again so that a new connection does not have to be established. If all the connections are being used, a new connection is made and is added to the pool. Connection pooling also cuts down on the amount of time a user must wait to establish a connection to the database.

We observed that with connection pooling in place, the average time is reduced. This helps in improving the performance of our application.

Performance:

Performance is very important for a system. We must check our systems performance to check our applications responsibility and stability under workload. It also helps us to analyze various attributes like scalability, reliability and resource usage.

In the graphs above, for each application, we can see the average time taken by the server api to serve the request. We performed various activity like 100 requests by a single user, 1000 requests, 10,000 requests by a user and so on. It replicates a real-life scenario where a web app has to serve multiple requests at once which affects the performance of a system.

MOCHA TESTING

Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser. APIs were tested using Mocha successfully. Few API test cases are shown below:

```
var assert = require('assert');
var express = require('express');
var app = express();
var request = require('supertest');
var assert = require('chai').assert;

let chai = require('chai');
let chaiHttp = require('chai-http');

let should = chai.should();

chai.use(chaiHttp);

describe('http tests without auth', function(){

    before(function() {

    });

    after(function() {

    });

    it('ADMIN should be able to login with correct details', function (done)  {
        chai.request('http://localhost:3002/api/v1/a')
            .post('/signin')
            .send({
                email:"admin@gmail.com",
                password:"ankit"
            })
            .end(function(err, res) {
```

```
    it('ADMIN should not be able to login with wrong details', function (done)  {
        chai.request('http://localhost:3002/api/v1/a')
            .post('/signin')
            .send({
                email:"admin@gmail.com",
                password:"abc"
            })
            .end(function(err, res) {

                assert.equal(res.status, 400);
                done();
            });
    });

    it('CLIENT should be able to login with correct details', function (done)  {
        chai.request('http://localhost:3002/api/v1/c')
            .post('/signin')
            .send({
                email:"aa@aa.com",
                password:"abc@123"
            })
            .end(function(err, res) {

                assert.equal(res.status, 200);
                done();
            });
    });
```

```
    it('CLIENT should not be able to login with wrong details', function (done)  {
        chai.request('http://localhost:3002/api/v1/a')
            .post('/signin')
            .send({
                email:"aa@aa.com",
                password:"abc"
            })
            .end(function(err, res) {

                assert.equal(res.status, 400);
                done();
            });
    });

    it('CLIENT should get error while signup with wrong data', function (done)  {
        chai.request('http://localhost:3002/api/v1/c')
            .post('/signup')
            .send({
                email:"aa@aa.com",
                password:"abc"
            })
            .end(function(err, res) {

                assert.equal(res.status, 400);
                done();
            });
    });

```

Output:

```
C:\SJSU\GitHub\kayak\testing\Mocha (Aman_5)
λ npm test

> kayak_server@0.0.0 test C:\SJSU\GitHub\kayak\testing\Mocha
>  mocha


  http tests without auth
    √ ADMIN should be able to login with correct details (169ms)
    √ ADMIN should not be able to login with wrong details
    √ CLIENT should be able to login with correct details (89ms)
    √ CLIENT should not be able to login with wrong details
    √ CLIENT should get error while signup with wrong data
    √ CLIENT should not get error while signup with correct data (88ms)
    √ CLIENT/ADMIN should logout

  ADMIN http tests with auth
Cookie : connect.sid=s%3Agr1DI1Fdxp4Ih8UsIXj2aIrXlkGPhCOp.NpCEyhgu2mhejCt8d6tpBrH%2FyPP9zf81k6mh%2B7gAsyk
    √ ADMIN LOGIN (99ms)
    √ ADMIN Passport check session
    √ ADMIN should get listing of all cars
    √ ADMIN should get listing of all flights
    √ ADMIN should get listing of all hotels
    √ ADMIN should get listing of all customers
    √ ADMIN should get revenue based on type
    √ ADMIN should get revenue based on Top Companies
    √ ADMIN should get revenue based on City
    √ ADMIN should get User Analytics data

  CLIENT http tests with auth
Cookie : connect.sid=s%3A68qdJr4LFKqF7j96YhhnBq5-sdtWh7-x.nKa1V78Gv16z5%2B3IcpxEihoQCk5dzzw2d8PUFu0eQR4
    √ CLIENT LOGIN (87ms)
    √ CLIENT should get All Credit card details
    √ CLIENT should get Hotel listings


  20 passing (750ms)
```

DATABASE SCHEMA

Hotel Schema

```
id : String
hotelName : String
hotelAddress : String
hotelCity : String
hotelState : String
hotelZip : Number
hotelPhoneNumber : Number
hotelEmail : String
hotelStar : Number
hotelRating : Number
hotelAmenities : [String],
hotelRooms : [{  roomType : String,
                 priceTotal : Number, totalAvailable : Number,
                 personPerRoom : Number}]
is_deleted : Boolean
serviceStartDate : Date
serviceEndDate : Date
availability : [{ availableDate: Date,
                  hotelRooms : [{ roomType : String,
                                  priceTotal : Number,
                                  totalAvailable : Number,
                                  personPerRoom : Number}] } ]
images : [String]
```

AuthUser Schema

```
id : String
is_deleted: Boolean
auth_user_id : String
firstName : String
lastName : String
address: String
city: String
state: String
zip_code: String
phone_number: Number
profile_image: String
role: String
email: String
```

Car Schema

```
id : String
carType : String
carName : String
carQuantity : String
serviceStartDate : Date
serviceEndDate : Date
occupancy : Number
luggage : String
dailyRentalValue : Number
createdDate: Date
updatedDate : Date
availability : [{ availabilityDate : Date,
                  availableCars : Number }]
is_deleted : Boolean
images : [String]
```

Flight Schema

```
id : String
flightNumber : String
airline : String
source : String
destination : String
arrival : String
departure : String
createdDate: Date
updatedDate : Date
is_deleted : Boolean
serviceStartDate : Date
availability : [{availabilityDate : Date  ,
                 sections : [{class : String , price : Number ,
                              available : Number}]
                }]
serviceEndDate : Date
firstClassPrice: Number
firstClassSeats: Number
economyClassPrice: Number
economyClassSeats: Number
businessClassPrice: Number
businessClassSeats: Number
meals : Boolean
luggage : Number
```

Billing Schema

```
id : String
listingType: String
listingId:mongoose.Schema.ObjectId
bookingId:mongoose.Schema.ObjectId
userId:String
totalAmount:String
createdDate:Date
creditCardId:mongoose.Schema.ObjectId
```

CreditCard Schema

```
id : String
userId: String
cardNumber:String
nameOnCard:String
cvv:String
expiryDate:String
```

Booking Schema

```
id : String
listingType: String
listingId:mongoose.Schema.ObjectId
userId:String
bookingInfo:Object
createdDate:Date
```

1. MongoDB:

MongoDB is a document-oriented database. Instead of storing your data in tables made from individual rows, like a relational database does, it stores your data in collections made of individual documents. In MongoDB, a document is a big JSON blob with no particular format or schema.
MongoDB should be used in the following cases:

- High Write Load
  MongoDB by default prefers high insert rate over transaction safety. If you need to load tons of data lines with a low business value for each one, MongoDB should fit.
- High Availability in an Unreliable Environment
- Data is Location Based
  MongoDB has built in special functions, so finding relevant data from specific locations is fast and accurate.

In our application, MongoDB can be used to store files and user activity log data on the database.

2. MySQL:

MySQL is a free, open-source database management system. A DBMS is a system that manages databases and connects them to software. MySQL is a powerful, free open-source database management system that has been around for years. It is very stable and has a big community that helps maintain, debug and upgrade it.
MongoDB should be used in the following cases:

- When we need to define relations between data present in different tables.
- Data to be stored is not large.
- Large Scalability is not required.

OBSERVATIONS & LESSONS LEARNED

Observations

The implementation of the whole Kayak website along with the analytics part was a large scale project which needed proper planning and coordination. Few of the points which we observed are as follows:

- Defining all the APIs before starting the project helped us in understanding the whole skeleton of the project.
- We divided the project into different modules internally which helped us to develop different sections parallelly. This resulted in a lot of time saving because one module did not affect the others.
- While developing the UI, we made various components like sidebars etc which was reused in various web pages. This reduced the number of lines of code we had to write.
- The analytics page helped us gaining data from end user, which can be later used to improve the services.
- We used Mangoose ODM for managing data in MongoDB by pre-defining the data models. It helped in reducing the code to perform CRUD operations.
- We followed Agile methodology in our project development phase. The daily standup meeting benefited us with keeping track of everyone's tasks and planning ahead for the remaining tasks.

Lessons Learned

Below are few of the points we think could have been improved.

- Develop code keeping the analytics page in mind. This would have helped in developing the analytics page faster.
- JMeter helped us in testing of scalability and load balancing. After testing, we improved our code to make our website more stable.
- We used mocha to test various APIs with random data. It helped us in killing a few bugs as well.
- Feedback is an important part of project development. While deciding solutions to various functionalities, we kept a brainstorming session where everyone came up with suggestions and inputs.