CSCI 5409 Advanced Topics in Cloud Computing

Final Project Report

Instructor: Dr. Saurabh Dey
Date of Submission: 29 March 2020
Submitted By:

Arunachalam Hari (B00832143)

Gamidi Vamsi (B00834696)

Kase Raviteja (B00823644)

Sharma Anuj (B00825885)

Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia

Contents

1.Project Requirement:
This project involves the development of a secure web application and a mobile application for Canada tourism. The functionalities are the same for both the web application and the mobile application. The key requirements for the project include the home page which contains a search page. Users can search for the national parks, beaches, hill stations, amusement parks, ski resorts, holiday destinations all over Canada. They do not need to login to query the destinations. After selecting a holiday destination, the user will be taken to the booking page. To book tickets, the user will have to create an account and login. A ticket will be generated after the completion of the payment.

The following are the main modules in the project:[1]

- Search
  - The user can search for various places all over Canada. By using the GPS of the mobile in which the application is used, locations will be sorted based on the location.
- Signup
  - For the booking of tickets, the user can create an account using this module.
- Login
  - After creating an account, the user can log in to book the tickets.
- Booking
  - The user can book the tickets after logging into the account.
- Payment
  - The payment module handles the payments through credit/debit cards.
- Ticket Generation
  - A ticket will be generated after the payment is done.

2.Job role:
Arunachalam Hari: Flask API development, Swagger implementation, Docker implementation, Cloud deployment using Elastic container service

Kase Raviteja: Angular application development and deployed and auto-scaled using Elastic BeanStalk service. Load and performance testing using JMeter.

Gamidi Vamsi: Android application development: Login, Signup, Two-Factor Authentication, Displaying Location details, Searching and Filtering the Locations.

Sharma Anuj: Mobile application: Fetching data from packages API (http://52.90.211.209:81/package/), implementing ticket booking functionality, generation of bus ticket after the booking is successful in android application, writing and executing the test cases for mobile application, proofreading the project report.

3.Knowledge Sharing:
This project has given industry exposure and gave an idea of how the actual industry works and scales according to user traffic. Our team has used Gitlab to achieve continuous integration and timely reviews are done on the code pushed and merging the code with reviewal of merge requests id done.

We were skeptical at the beginning, considering challenges of the time frame and with tons of services provided by AWS. After spending a good amount of time in research our team has decided to use ECS since it has the best pricing model when compared to other services provided by AWS for deploying services.

Since we will be using the AWS Educate account and considering the resources offered to us, we stick to ECS and which is the best option.

Due to the unforeseen situation that happened because of the COVID 19 pandemic, most of our interactions have happened through Microsoft Teams. But in a way, it has provided the exposure of how things work in the actual industry. Most of our conversations and knowledge sharing has happened in Microsoft Teams. As the conversations can be recorded, if anyone gets stuck at some point, they can always go through the conversation and can be able to solve the issue. Apart from this, everyone in the team has gained a wonderful experience and knew how the usage of the cloud can help industries on a large scale. As graduate students, we have gained the confidence to pitch with an idea and invest a small amount of money to get starting our startup.

4. Final design:

The final architecture we implemented for this project is:

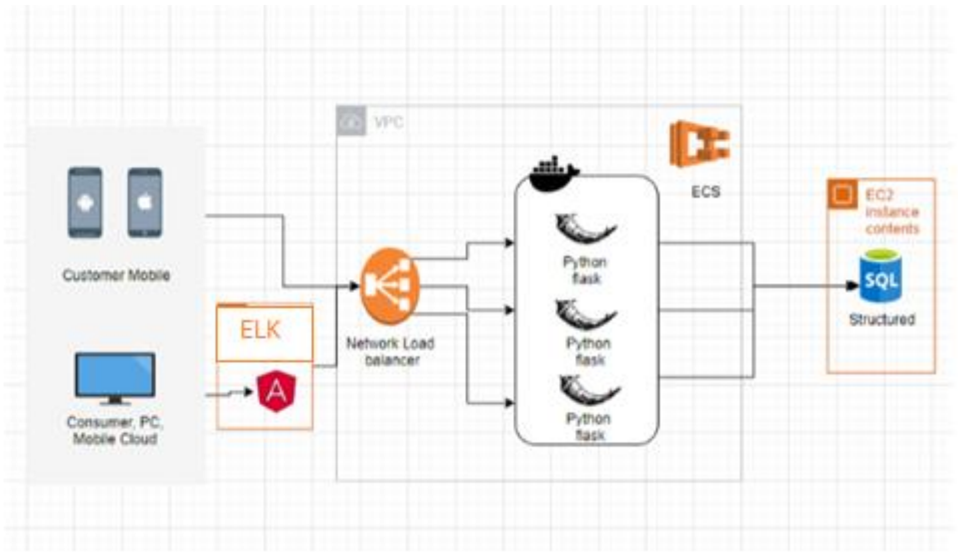


Figure 1:Architecture

The architecture that is implemented is almost identical to the one proposed in the design documents with the minus of autoscaling and the addition of swagger API documentation.

The mobile application is installed on the client application. The Angular application is hosted on an EC2 instance along with the MySql server. The core of the cloud application, the restful APIs are hosted on multiple clusters using AWS' Elastic container services.

4.1 Sitemap

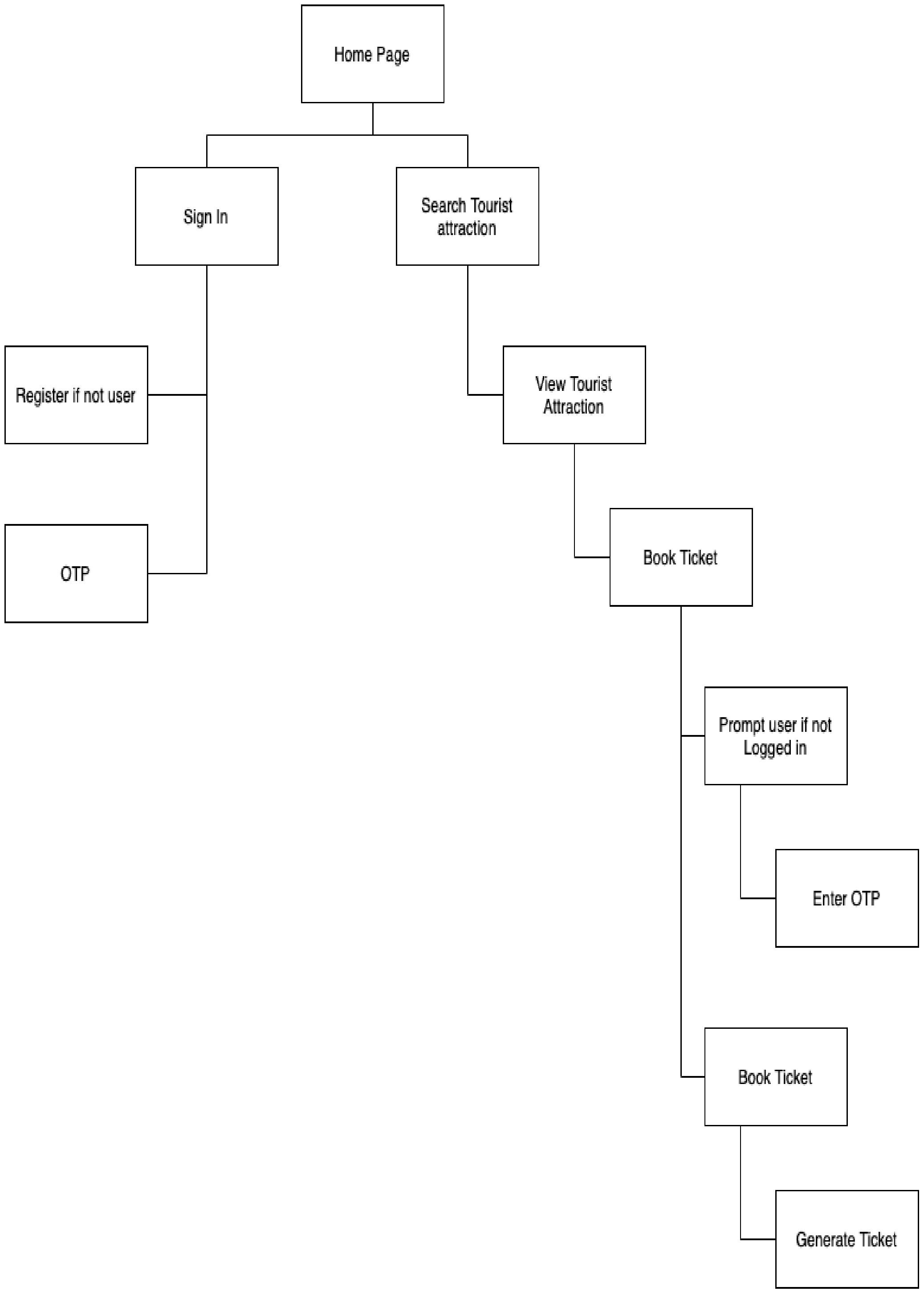


Figure 2:Sitemap[2]

4.2 Flowchart

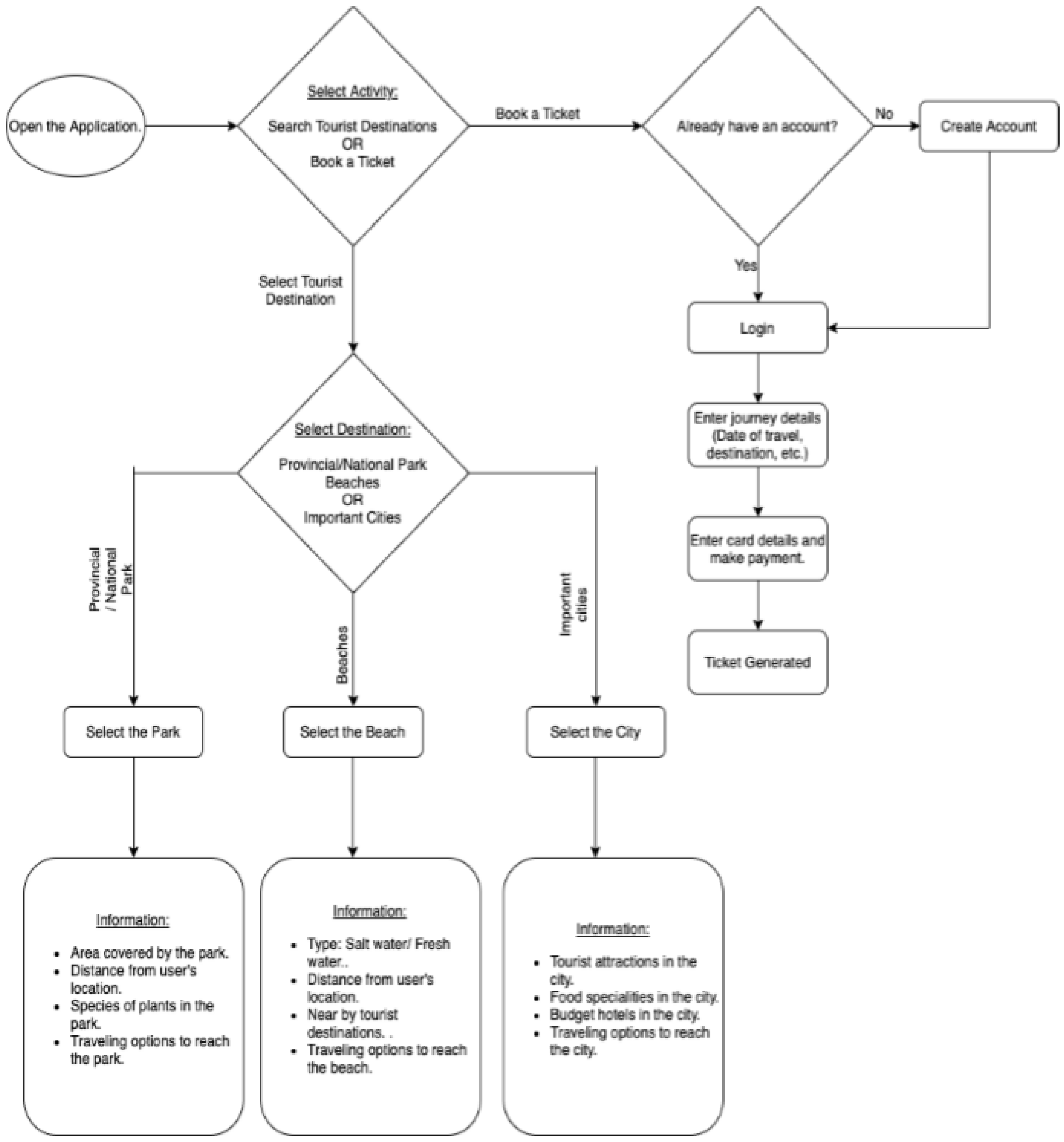

Open the Application.
Select Activity:
Search Tourist Destinations
OR
Book a Ticket
Book a Ticket
Already have an account?
No
Create Account
Yes
Login
Enter journey details (Date of travel, destination, etc.)
Enter card details and make payment.
Ticket Generated
Select Tourist Destination
Select Destination:
Provincial/National Park
Beaches
OR
Important Cities
Provincial / National Park
Beaches
Important cities
Select the Park
Select the Beach
Select the City
Information:
Area covered by the park.
Distance from user's location.
Species of plants in the park.
Traveling options to reach the park.
Information:
Type: Salt water/ Fresh water..
Distance from user's location.
Near by tourist destinations. .
Traveling options to reach the beach.
Information:
Tourist attractions in the city.
Food specialities in the city.
Budget hotels in the city.
Traveling options to reach the city.

Figure 3: Flow chart [2]

5.Completion Report

5.1 Web application

Angular version 8 is used to build the website. The flow of the website is followed as per the requirements mentioned in the project specification document. Such a built angular application is deployed in AWS EC2. APIs developed are consumed in the angular website, which makes it more robust and secure as the major functionalities work over the server-side.

The flow of the website is as follows:

1. User will be redirected to Home page
2. On the Home page, the list of tourist attractions is displayed with search bar enabled.
3. User can search particular tourist attraction, as the user starts searching, the tourist attraction is loaded asynchronously from the database with the consumption of API.
4. Users can be able to click on the desired tourist attraction.
5. Once the user selects the desired tourist attraction, the user will be redirected to a particular tourist destination which contains key features.
6. If the user wants to Book Ticket to a particular destination, the user clicks on Book Ticket.
7. Once the user clicks on Book Ticket, if the user is already logged, the user will be redirected to the Ticket booking page or else the user will be prompted to log in.
8. User registers, if not already a user or user login by providing email and password.
9. If the user entered details are correct, the user will be sent an OTP to email, to authenticate the login.
10. Once the user enters OTP, the user will be redirected to the Ticket Booking page.
11. The user enters all the details, necessary for booking the ticket and confirms payment.
12. If payment is successful, the user will be redirected to the ticket confirmation page with all the necessary details.
13. An email will be generated with ticket details that will be sent to the user.

The flow of the website is captured in the below snapshots:

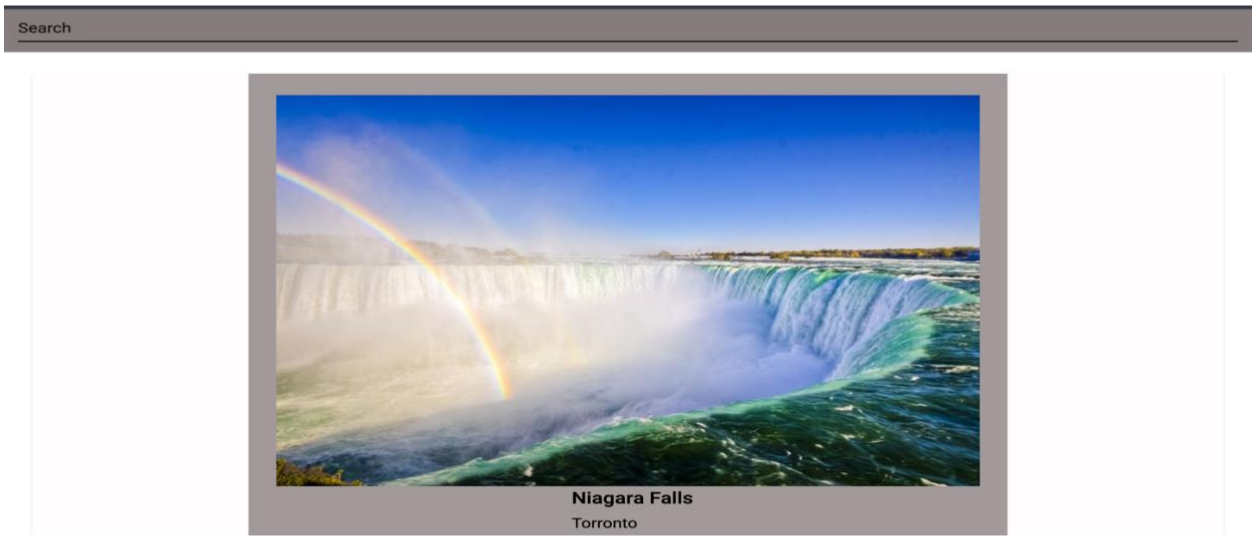


Figure 4:Home page

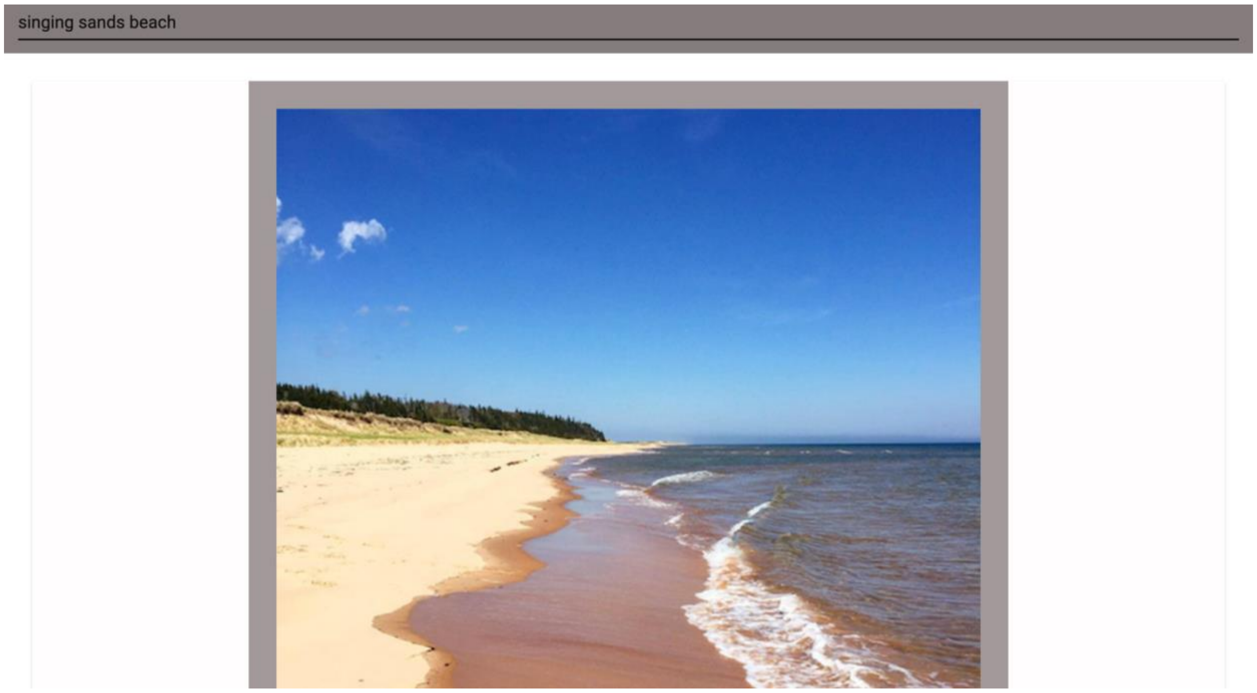
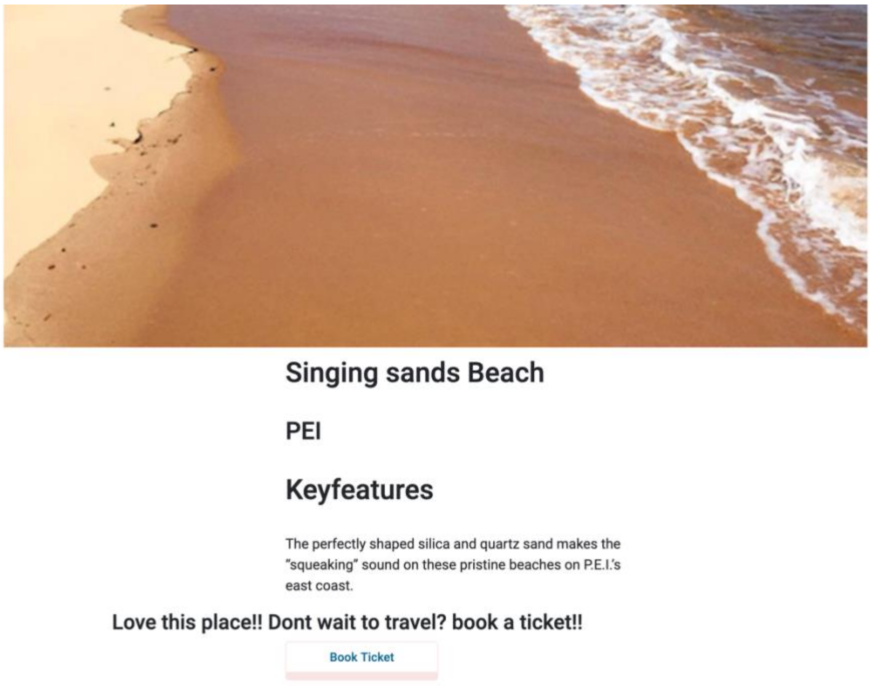


Figure 5:Search Functionality



Singing sands Beach

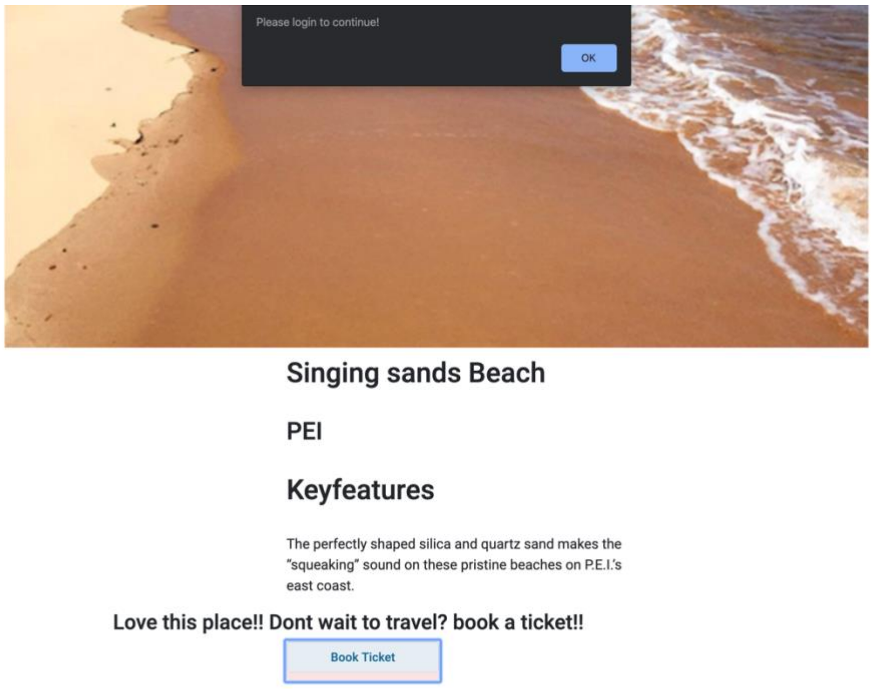
PEI

Keyfeatures

The perfectly shaped silica and quartz sand makes the "squeaking" sound on these pristine beaches on P.E.I.'s east coast.

Love this place!! Dont wait to travel? book a ticket!!

Figure 6:Key Features of Tourist attraction


Please login to continue!

Singing sands Beach

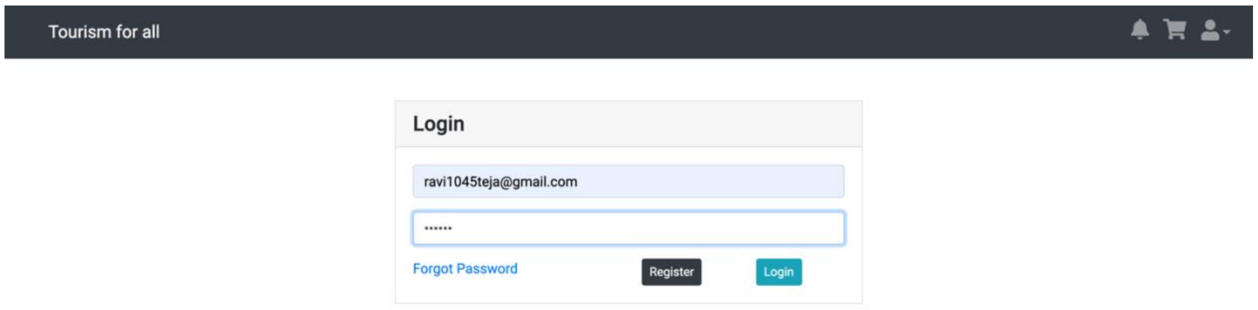
PEI

Keyfeatures

The perfectly shaped silica and quartz sand makes the "squeaking" sound on these pristine beaches on P.E.I.'s east coast.

Love this place!! Dont wait to travel? book a ticket!!

Figure 7:Login Prompt once the user clicks Book Ticket

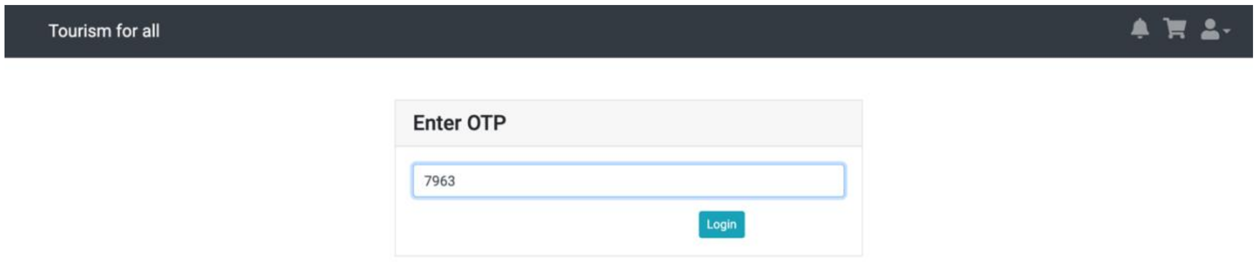

Tourism for all

Login

ravi1045teja@gmail.com

••••••

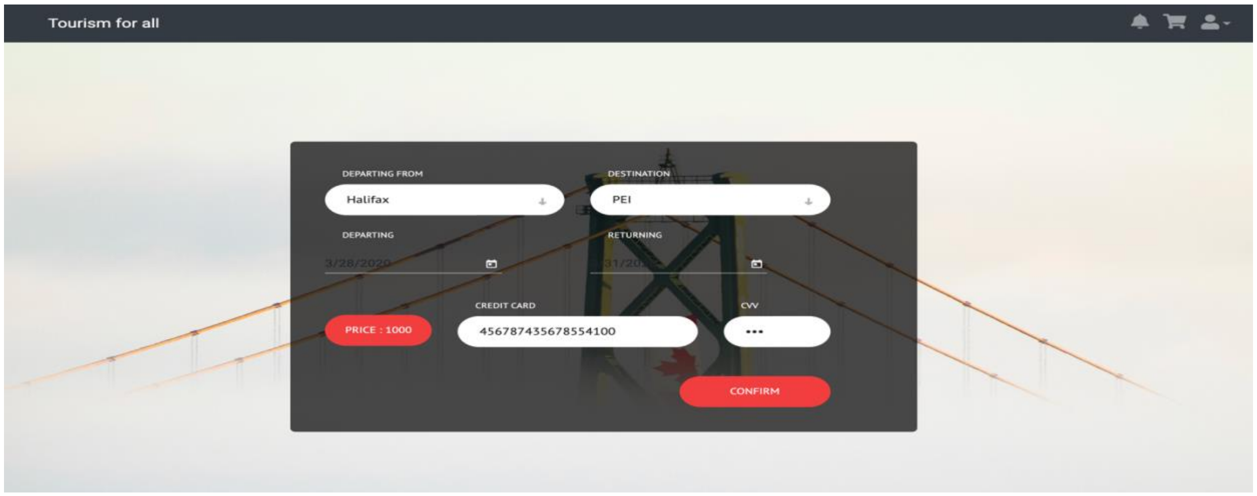
Figure 8:Login Page


Tourism for all

Enter OTP

7963

Figure 9:OTP Authentication


Tourism for all

DEPARTING FROM: Halifax

DESTINATION: PEI

DEPARTING: 3/28/2020

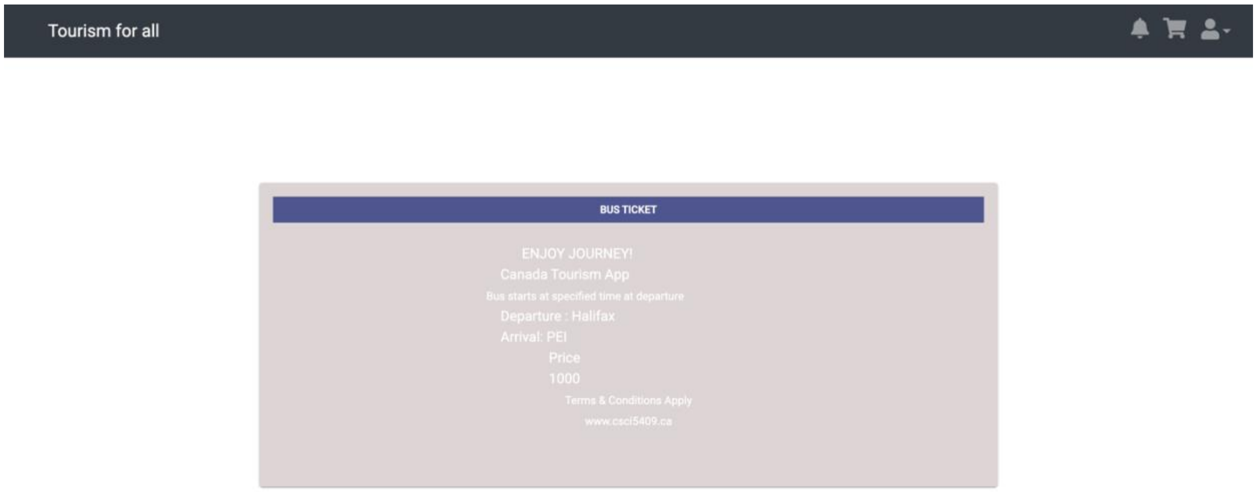
RETURNING: 31/20

PRICE : 1000

CREDIT CARD: 456787435678554100

CVV: •••

Figure 10:Ticket Booking page


Tourism for all

BUS TICKET

ENJOY JOURNEY!
Canada Tourism App
Bus starts at specified time at departure
Departure : Halifax
Arrival: PEI
Price
1000
Terms & Conditions Apply
www.csci5409.ca

Figure 11:Ticket Confirmation page

5.2 Android application

An android application was created with almost similar functionality as the web application. The application consumes the API's to get and post the data as required. The user will be able to see all the locations on the home page without logging in. There is a search option using which the user can filter the places. A location from the home page can be selected which will take the user to the next page that shows complete details of the location. The user can click on book ticket button to book the ticket to the selected destination. To book a ticket, the user has to log in if there is an existing account or can sign up if the user is new. To provide better security, two-factor authentication was implemented by sending an OTP (One-time password) to the email provided by the user. After logging in, the user can select the source and destination from the list of packages available by providing the credit card details. Once the ticket is booked, the user will get the ticket on the application itself.

A recycler view was used to display the location details on the home page. If the user enters a text in the search bar on the top of the screen, text changed event will be triggered which filters the locations list by comparing the search input given by the user. A variety of validations were used for signup activity to increase the security by urging the user to select a strong password.

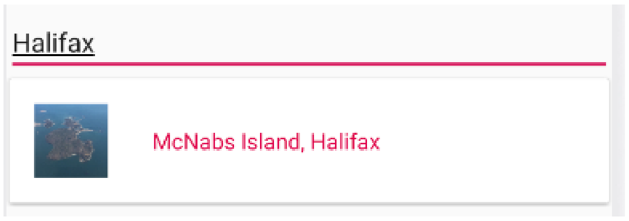


Figure 12: Home page with search functionality

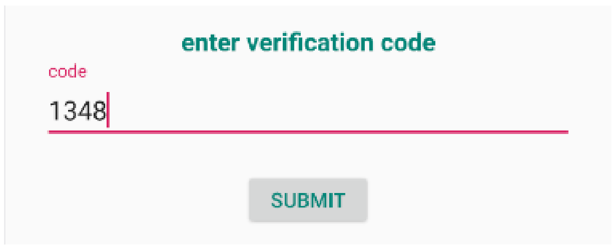


Figure 13: Two-Factor Authentication

Once the user selects the location, he/she is redirected to the ticket booking page, where they have to enter the desired information like source of the journey, destination (it is pre-selected based on the user's location selection from the previous activity), departure date, returning date. Based on the source and destination, the price text box is populated automatically by fetching the price from the API (http://52.90.211.209:81/package). After that, the user has to enter their card details (credit/debit card number, and CVV number) and clicks the "Submit" button. If all the details are valid, a sample ticket is generated for the user with relevant details like source, destination, ticket number, travel date, and price.

The following components are used for ticket booking page:

- The spinner variable in android [5] is used for populating the source and destination from the API. The data from the API is stored in JSON array objects and later on stored in the Array list which is populated in the Spinner object.
- The departure and returning date are fetched using Date Picker dialog object [7] in android and it is displayed in the "mm/dd/yyyy" format.
- For all the other fields, the details are entered in the text box by the user.

Following validations are implemented in the Ticket booking page:

- The source and destination cannot be the same.
- The return date cannot be before the departure date.
- The credit/debit card number should be a 16-digit number.
- The CVV number should be a 3-digit number.

5.3 API

For API, we have used the python Flask framework to build them. They are restful microservices. To maintain simplicity and understanding, we have split the API into two main parts, the Authentication and user management API, which can be accessed through the following link:

http://100.26.195.40:82/.

These APIs take care of User registration, returning user details, the login, logout and user validation based on JWT.

The application uses a separate MySQL database for authentication, to maintain security.

For login, we have implemented **2-factor Authentication**. Once the user enters his email and password, he receives an email with the OTP. Using this OTP, he/she can log in to the site. To highlight our work for session management in a cloud environment, we used **Java Web Tokens** to authenticate the client. Using the token, we can ensure it's a valid user and get the details of the user using the token. However, using this we can't authenticate the server. We can consider this, a drawback.

For encryption, we have implemented **hashing of password** at the **database level**.

The second set of APIs are related to the tourism site as such and can be accessed through the following link:

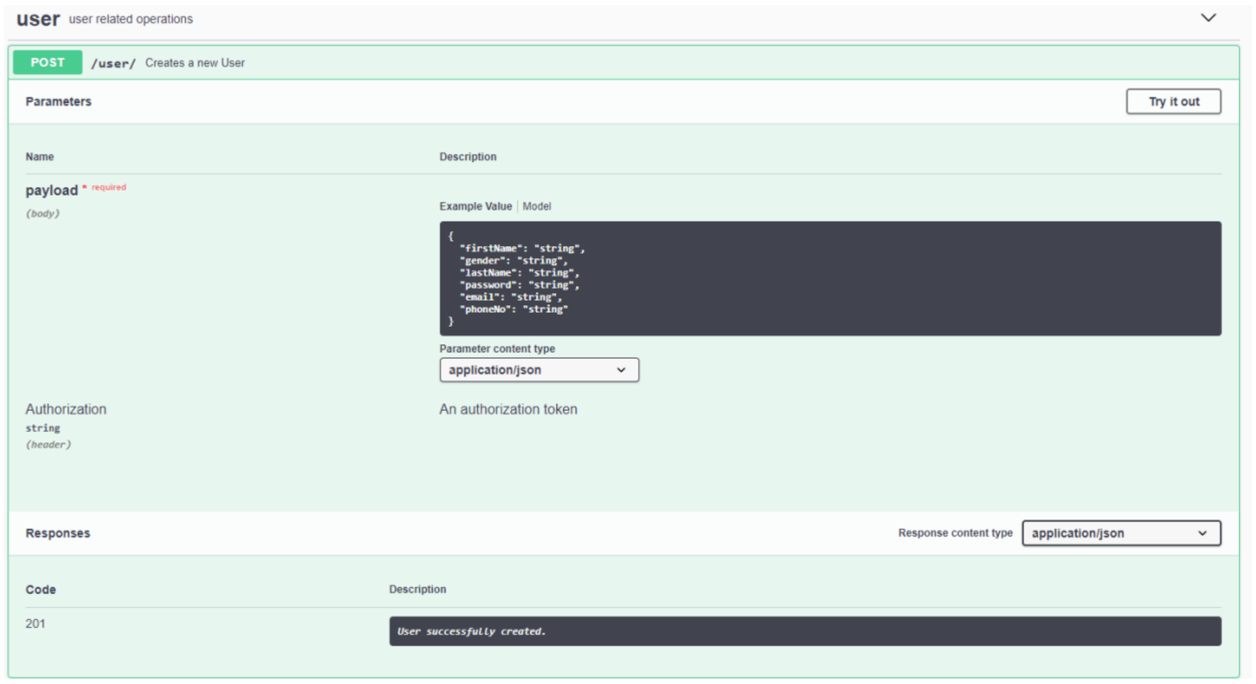
http://52.90.211.209:81/.

These APIs take care of the business logic of the site such as location and packages listing, location and package details, ticket booking, and payment. Here the payment and ticket booking API need authorization, as they need a valid JWT to be called and return a proper response. All logic for the functionalities for the website is handled at the API side.

Documentation and Testing

We believe documentation is vital for APIs, as this would enable the API to be accessible for all, beyond the scope of this application. So, we have implemented Swagger UI. Swagger is the largest framework for designing APIs using a common language and enabling the development across the whole API lifecycle, including documentation, design, testing, and deployment [3].

Apart from the documentation, we can use Swagger to test our API directly. It makes the entire process simpler by giving us the sample response and request. This ensures that even a non-technical person can test our APIs



payload * required
(body)

Example Value | Model

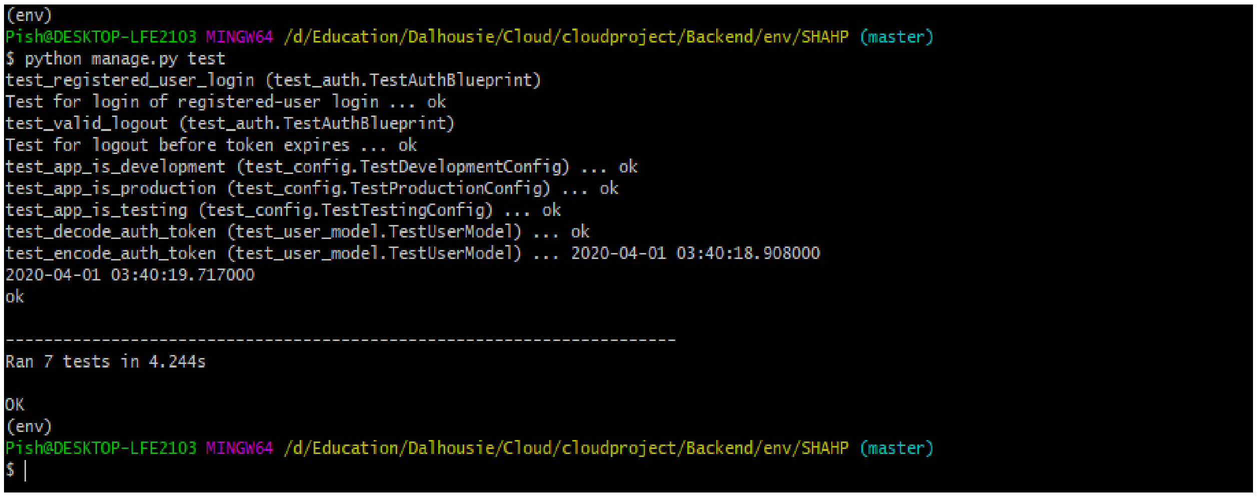
```
{
  "firstName": "string",
  "gender": "string",
  "lastName": "string",
  "password": "string",
  "email": "string",
  "phoneNo": "string"
}
```

Parameter content type: application/json

Authorization
string
(header)
An authorization token

201 User successfully created.

Apart from this manual testing, we have also implemented automation test cases in the flask for Auth and User APIs. These test cases can be used as part of CI/CD if it was implemented.



```
(env)
Pish@DESKTOP-LFE2103 MINGW64 /d/Education/Dalhousie/Cloud/cloudproject/Backend/env/SHAHP (master)
$ python manage.py test
test_registered_user_login (test_auth.TestAuthBlueprint)
Test for login of registered-user login ... ok
test_valid_logout (test_auth.TestAuthBlueprint)
Test for logout before token expires ... ok
test_app_is_development (test_config.TestDevelopmentConfig) ... ok
test_app_is_production (test_config.TestProductionConfig) ... ok
test_app_is_testing (test_config.TestTestingConfig) ... ok
test_decode_auth_token (test_user_model.TestUserModel) ... ok
test_encode_auth_token (test_user_model.TestUserModel) ... 2020-04-01 03:40:18.908000
2020-04-01 03:40:19.717000
ok

----------------------------------------------------------------------
Ran 7 tests in 4.244s

OK
(env)
Pish@DESKTOP-LFE2103 MINGW64 /d/Education/Dalhousie/Cloud/cloudproject/Backend/env/SHAHP (master)
$ |
```

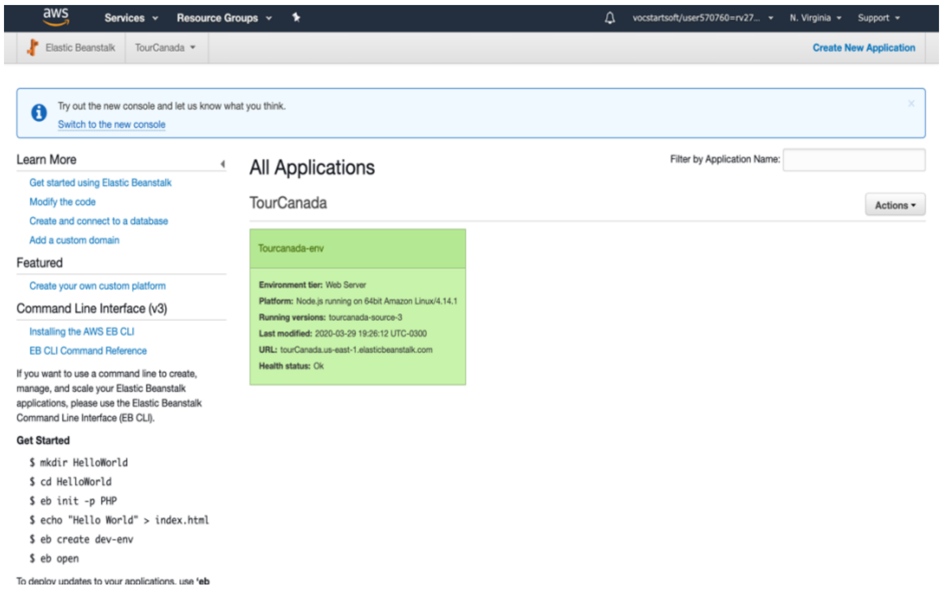
5.4 Cloud deployment

Web Application:

Angular website is deployed using **Elastic BeanStalk** service and auto-scaled to 3 instances and load balanced using **application load balancer** [4]. The angular application developed locally is built and a server file is created which invokes the deployed application at 8081. The “dist” folder created after built and the server file are zipped and uploaded to the created environment of Elastic Bean Stalk.

Application is available at the link:

http://tourcanada.us-east-1.elasticbeanstalk.com/homepage



Try out the new console and let us know what you think.
Switch to the new console

Learn More
Get started using Elastic Beanstalk
Modify the code
Create and connect to a database
Add a custom domain

Featured
Create your own custom platform

Command Line Interface (v3)
Installing the AWS EB CLI
EB CLI Command Reference

If you want to use a command line to create, manage, and scale your Elastic Beanstalk applications, please use the Elastic Beanstalk Command Line Interface (EB CLI).

Get Started

```
$ mkdir HelloWorld
$ cd HelloWorld
$ eb init -p PHP
$ echo "Hello World" > index.html
$ eb create dev-env
$ eb open
```

To deploy updates to your applications, use 'eb

All Applications

Filter by Application Name:

TourCanada

Actions

Tourcanada-env

Environment tier: Web Server
Platform: Node.js running on 64bit Amazon Linux/4.14.1
Running versions: tourcanada-source-3
Last modified: 2020-03-29 19:26:12 UTC-0300
URL: tourCanada.us-east-1.elasticbeanstalk.com
Health status: Ok

Figure 12:Angular Website Deployed using Elastic BeanStalk

API:

The two flask applications are dockerized. This allows us to deploy the containers in any environment without any worries about the dependencies. To facilitate the building of the docker containers, we developed the applications in python's Virtual environment. This allowed us to get the required dependencies easily. This also helped us understand that a docker is a virtual environment at its core.

The next step in our deployment was to set up our cluster and deploy the dockers there. As discussed in our design, we decided to use AWS's Elastic container service (ECS). For this, we need to store the docker images in AWS's Elastic container repository (ECR). Here, we hit our first roadblock. To push our images to ECR, we need to use AWS CLI. The credentials for the AWS educate 50$ account didn't work. We ended up spending a lot of time, trying to find a workaround for this. We ended up using the 100$ account.

Using the CLI, we pushed the docker images to ECR.We login into our ecr using

"aws ecr get-login-password --region us-east-1 | sudo docker login --username AWS --password-stdin 579529920164.dkr.ecr.us-east-1.amazonaws.com/cloud-project"

The docker images must be tagged using

"docker tag auth-app:latest 579529920164.dkr.ecr.us-east-1.amazonaws.com/cloud-project"

and then pushed to ECR using

"sudo docker push 579529920164.dkr.ecr.us-east-1.amazonaws.com/cloud-project:latest"

The next step is to create the cluster in which the dockers should be run. We can use the AWS UI to create. Here we define the number and type of EC2 instances that should be run and AMI roles that are associated with this cluster. Here, we faced an issue where EC2 instances weren't launched. We found out that we have assigned '*AmazonEC2ContainerServiceforEC2Role*' policy to our AMI role. The AWS documentation is lacking in this area.

After this, we will have to define an **ECS task**. This is where we choose the docker image that needs to run in a cluster. We define the ports that need to be opened for the docker containers. We also define the memory and CPU required for the task.

Now, we must implement a **network load balancer** according to our design. For this, we need to set up listeners, which are internet-facing and with ports that are required for us. In our case TCP ports, 80 and 81 were required. Then we choose the availability zones according to our needs and create the network load balancer.

The final step in our deployment is to link all of that we have created. We can do this by creating a service in the cluster. In the service, we choose our task and load balancer. Then we launch the service.

We were unable to do autoscaling with ECS because of account access issues. We couldn't find a work-around for this, so we used the Elastic bean stack as an example for autoscaling for the web application.

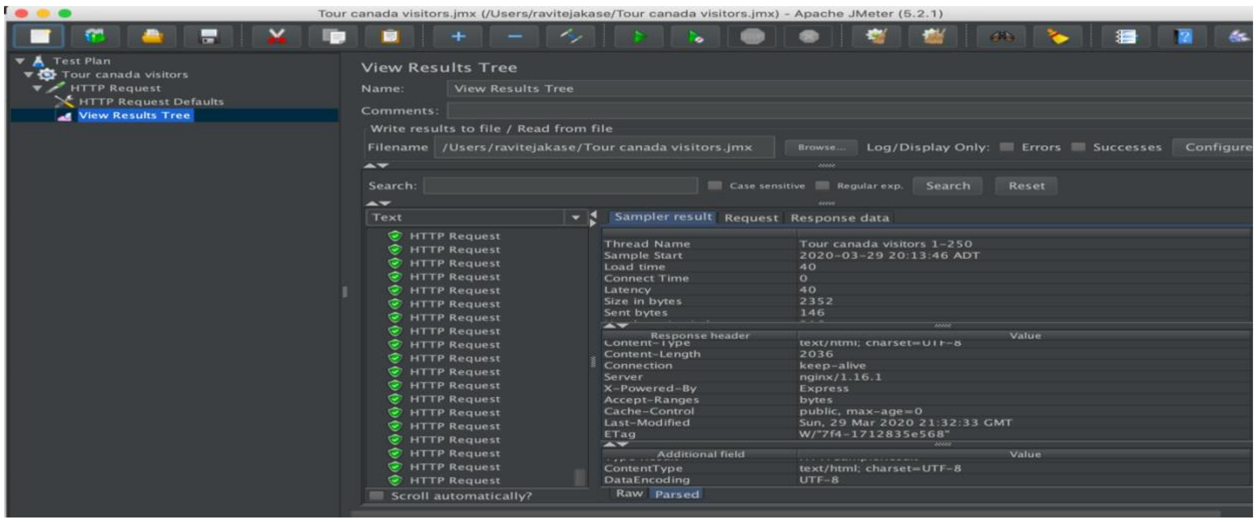
Another issue we faced was that we couldn't create namespaces. We need namespaces for two services with-in the same cluster to communicate with each other. In the end, we used different clusters to overcome this problem.

Android Application:

The application is available as an apk and can be installed on client mobile phones

5.5. Testing (Load and performance)

Load testing is done using Apache JMeter [5] with 500 threads twice and due to the load balancing done, requests were served perfectly. Metrics of testing done were shown in AWS Console and the test requests parsed are fetched in JMeter as well.


Tour canada visitors.jmx (/Users/ravitejakase/Tour canada visitors.jmx) - Apache JMeter (5.2.1)

Test Plan
Tour canada visitors
HTTP Request
HTTP Request Defaults
View Results Tree

View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename /Users/ravitejakase/Tour canada visitors.jmx
Log/Display Only: Errors Successes

Search: Case sensitive Regular exp.

Text

HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request

Scroll automatically?

Sampler result | Request | Response data

Thread Name	Tour canada visitors 1-250
Sample Start	2020-03-29 20:13:46 ADT
Load time	40
Connect Time	0
Latency	40
Size in bytes	2352
Sent bytes	146

Response header	Value
Content-Type	text/html; charset=UTF-8
Content-Length	2036
Connection	keep-alive
Server	nginx/1.16.1
X-Powered-By	Express
Accept-Ranges	bytes
Cache-Control	public, max-age=0
Last-Modified	Sun, 29 Mar 2020 21:32:33 GMT
ETag	W/"7f4-1712835e568"

Additional field	Value
ContentType	text/html; charset=UTF-8
DataEncoding	UTF-8

Raw Parsed

Figure 13: Http Requests from JMeter

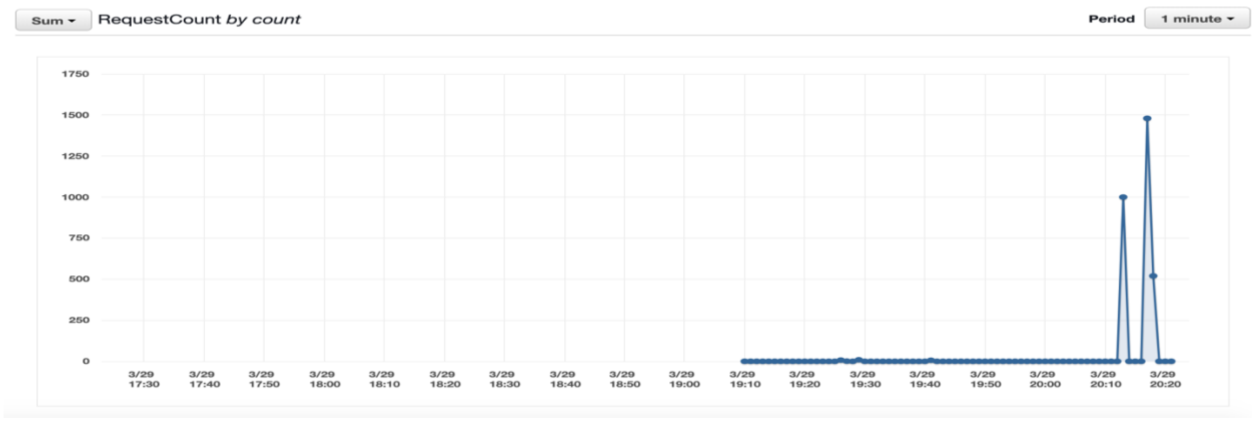


Figure 14: Requests sent from JMeter are shown in the AWS Console

So, the evidence shows that the application deployed is load balanced and auto-scaling occurs, if more number of requests hit the website.

6.Future work

We would love to work on CI/CD as it is an important part of cloud applications. We were unable to do this due to the lack of time. Also, we want to explore autoscaling in ECS.

We would also like to use a certificate to encrypt our communications so that we can use HTTPs

7.Repositories

Web Application and API's: https://git.cs.dal.ca/haria/cloudproject.git

Android Application: https://git.cs.dal.ca/gamidi/tourism.git

8.References

[1] A. Hari, G. Vamsi, S. Anuj, K. Raviteja "Group_11_Project_Feasibility_Report.pdf" Dalhousie University, [online document], 2020. [Accessed 25-Mar-2020]

[2] [diagrams.net - free flowchart maker and diagrams online, "Flowchart Maker & Online Diagram Software," *Diagrams.net*, 2020. [Online]. Available: https://app.diagrams.net/. [Accessed: 26-Mar-2020]

[3] DEV Community, "Why use Swagger for creating and documenting APIs," The DEV Community, 09-Jul-2018. [Online]. Available: https://dev.to/dianamaltseva8/why-use-swagger-for-creating-and-documenting-apis-115l. [Accessed: 29-Mar-2020]

[4] J. Pablo, "Deploying Angular WebApps with Nodejs to Amazon Web Services' Elastic Beanstalk," *Medium*, 20-Feb-2019. [Online]. Available: https://medium.com/@diuke/deploying-angular-webapps-with-nodejs-to-amazon-web-services-elastic-beanstalk-fa271a4542ea. [Accessed: 29-Mar-2020]
[5] Ratna Emani, "How to install and run Apache JMeter in 8 easy steps," *Agiletrailblazers.com*, 2016. [Online]. Available: https://www.agiletrailblazers.com/blog/how-to-install-apache-jmeter-and-how-to-run-jmeter. [Accessed: 29-Mar-2020]

Code references:

[1] freeCodeCamp.org, “How to structure a Flask-RESTPlus web service for production builds,” freeCodeCamp.org, 15-Apr-2018. [Online]. Available: https://www.freecodecamp.org/news/structuring-a-flask-restplus-web-service-for-production-builds-c2ec676de563/. [Accessed: 29-Mar-2020]

[2] “Docker - Deploying Flask app to ECS - 2020,” Bogotobogo.com, 2020. [Online]. Available: https://www.bogotobogo.com/DevOps/Docker/Docker-Flask-ALB-ECS.php. [Accessed: 29-Mar-2020]

[3] “Getting Started with Network Load Balancers - Elastic Load Balancing,” Amazon.com, 2020. [Online]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/network/network-load-balancer-getting-started.html. [Accessed: 29-Mar-2020]

[4] Faruk Ahmed, “Volley Get and Post Json object request”, 31-Oct-2018 [Online] Available: https://medium.com/techpin/https-medium-com-s2purno-volley-get-and-post-jsonobject-request-46fb8a46f799 [Accessed: 15-Mar-2020]

[5]"Android spinner (drop down list) example – Mkyong.com", Mkyong.com, 2020. [Online]. Available: https://mkyong.com/android/android-spinner-drop-down-list-example. [Accessed: 25- Mar- 2020].

[6] Alif, “Android Volley Tutorial – Making HTTP GET, POST, PUT”, 2013 [Online]. Available: https://www.itsalif.info/content/android-volley-tutorial-http-get-post-put [Accessed: 15-Mar-2020]

[7] "DatePicker Tutorial With Example In Android Studio | Abhi Android", Abhiandroid.com, 2020. [Online]. Available: https://abhiandroid.com/ui/datepicker. [Accessed: 28- Mar- 2020].