# CSCI 5409 Advanced Topics in Cloud Computing

**Project Design Study Report:** Tourism app
**Instructor:** Dr. Saurabh Dey
**Date of Submission:** 21 February 2020
**Submitted By:**

Arunachalam Hari (B00832143)
Gamidi Vamsi (B00834696)
Kase Raviteja (B00823644)
Sharma Anuj (B00825885)

Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia

## Table of Contents

Table of Figures

## Project Requirements

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:

- Search
  - Users 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
  - Users 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.

## System Architecture

The application needs to be deployed on the cloud and also utilize the various benefits that the cloud has to offer. As previously discussed, we can split the application into three modules, namely

1. Front-end: Angular for web applications/ Android application for mobile applications
2. Back-end: Python flask server to build the API and handle the logic for our application.
3. Database: MySQL database to store the persistent data.

The **architecture** and **services** we are planning to use for this deployment are:

**AWS EC2**

The AWS instance where the entire application will be hosted. The standalone EC2 instance will hold the front-end Angular server and the relational MySQL database. We aim to use a single EC2 instance to hold both the servers, to remain within the budget constraints and give more leeway to use multiple cloud services for the python API server [7].

**Docker container**

We are going to be using a Docker container to hold the backend server image. A container is a virtual environment that bundle the application code and required libraries into a single package. This can be then deployed on any host OS as it has all the dependencies required to run the application [1]. The advantages of using a docker container are:

- *Portability*: A container can be moved be from one from environment to another, without affecting the application it contains.
- *Modularity*: A container can be used to hold different types of application.
- *Security:* Each container is isolated from other containers and hence brings with better security
- *Scalability*: The container takes as much as space as required and can scale, if the application becomes bigger.

**AWS ECS**

Amazon EC2 Container Service (Amazon ECS) is a highly scalable, high-performance container management service that supports Docker containers and allows you to run applications easily on a managed cluster of EC2 instances. The ECS service scheduler places tasks onto container instances in the cluster, monitors their performance and health, and restarts failed tasks as needed.[2] Since we are going to be using a docker container, ECS seems like the way to go.

An option available to us is using ECS with Fargate, but we have chosen ECS with EC2 as it provides the features needed for this project but at a cheaper price point. The downside to ECS is that it involves more configurations, but this would serve as a good learning experience [6].

The ECS has four parts to it and we will be using different AWS services to do them.

**Elastic container repository (ECR)**:

AWS ECR is a fully managed Docker container registry that can be used to store, manage, and deploy Docker container images. It enables us to deploy docker images without worrying about out container repositories or scaling the underlying infrastructure[3]. ECR by default holds the data in a highly scalable infrastructure, thus helping us to create a scalable application.

**Clusters**:

An Amazon ECS cluster is a logical grouping of tasks or services [4]. Since we are using the EC2 launch type, cluster here refers to a grouping of container instances. Clusters can contain multiple different container instance types, but each container instance may only be registered to one cluster at a time.

**Tasks**:

This defines how containers should be created and how they should be run. We can define the various parameters that determine how the Docker container should be run. These parameters include [5], but not limited to:

- The docker image associated with each container in the task.
- Specify how much CPU and memory to use for each container.
- The Docker networking mode
- The bootstrap commands for the containers
- The IAM roles the task must use.

**Services**:

The containers that will be running and will be auto scaled, that is to the tasks that should be executed and maintained simultaneously in a n AWS cluster is called a service. If any of the tasks fails, the AWS ECS service scheduler launches another instance of the task and helps maintain the desired count of task definitions associated with the service.

**Load balancing**:

We will be using the network load balancer as it is what recommended by AWS based on the fact that it allows for TCP traffic to pass through without decrypting and encrypting again.[2]. This is important as we are planning to encrypt the data. This will involve the creation of an Amazon Virtual private cloud (VPC).

**Encryption**: Encryption is an important part of data transaction. To secure the data, one of the most common and powerful methods is to use a SSL certificate. We will use the SSL certificate provided by AWS to secure the data and transfer data, where required through HTTPS.

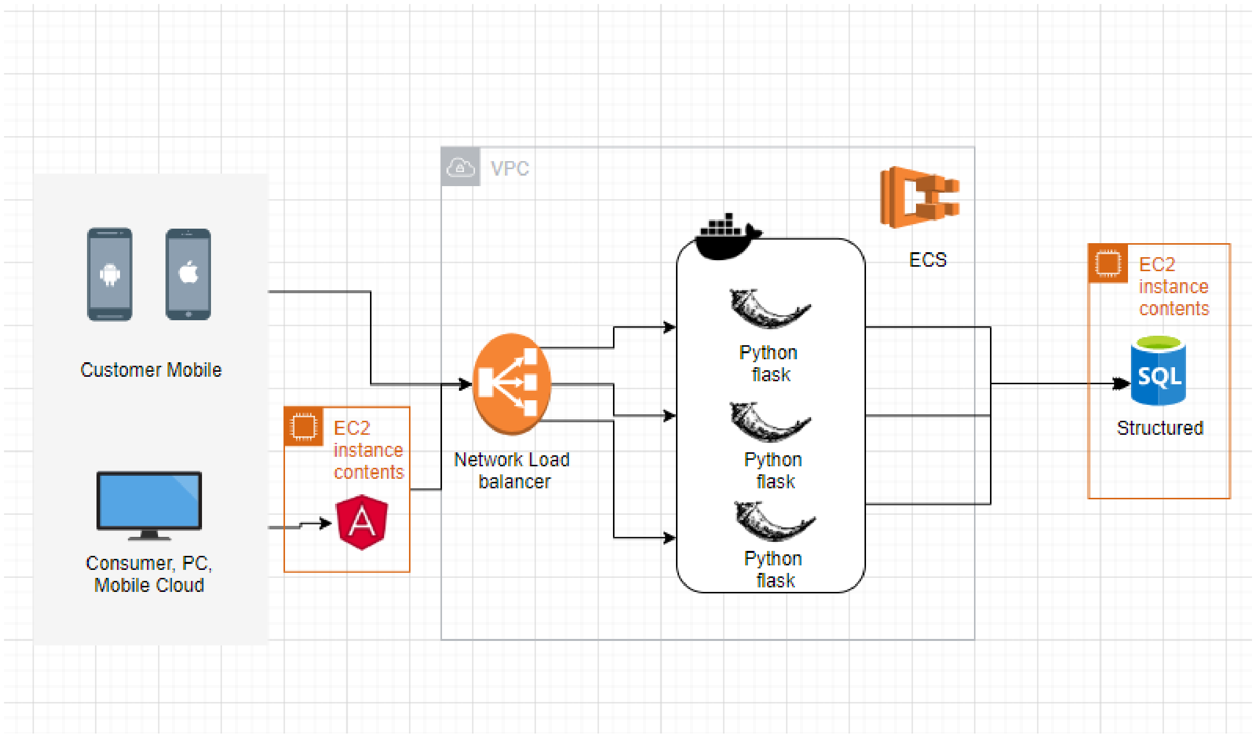


*Figure 1: System Architecture (Drawn using Draw.io)*

## Sitemap

The sitemap defines the flow of website or android application through pages and screens. Figure2 depicts the sitemap for the website and android application. The user visits the home page and will be able to see the search and filter section, where users can be able to search and filter various tourist destinations in Canada and based on the search criteria a list of options is displayed. Users can be able to select a particular tourist destination. After user clicks on a particular destination, the user will be redirected to the Tourist destination page which contains the highlight features of the destination and Book ticket functionality. Once user clicks on Book ticket functionality, the user will be redirected to the Ticket booking page. Ticket booking page contains the functionality to book tickets, once user clicks on book ticket functionality user will be prompted to log in, if user is not a member user will be prompted to create an account. Once user logins, the user will be redirected to the payment gateway where the user will enter the details and confirm the booking and after booking is done email will be sent to the user with generated ticket

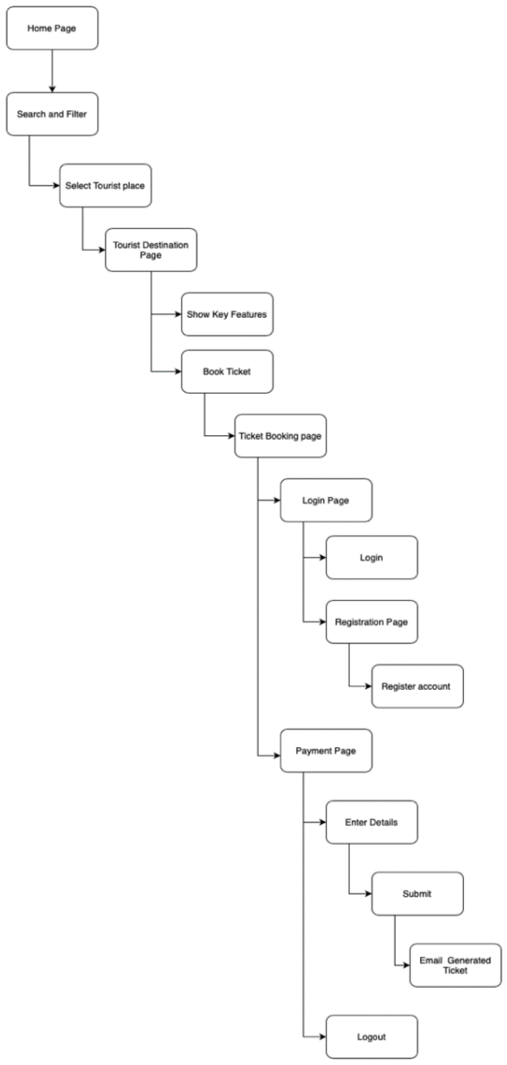
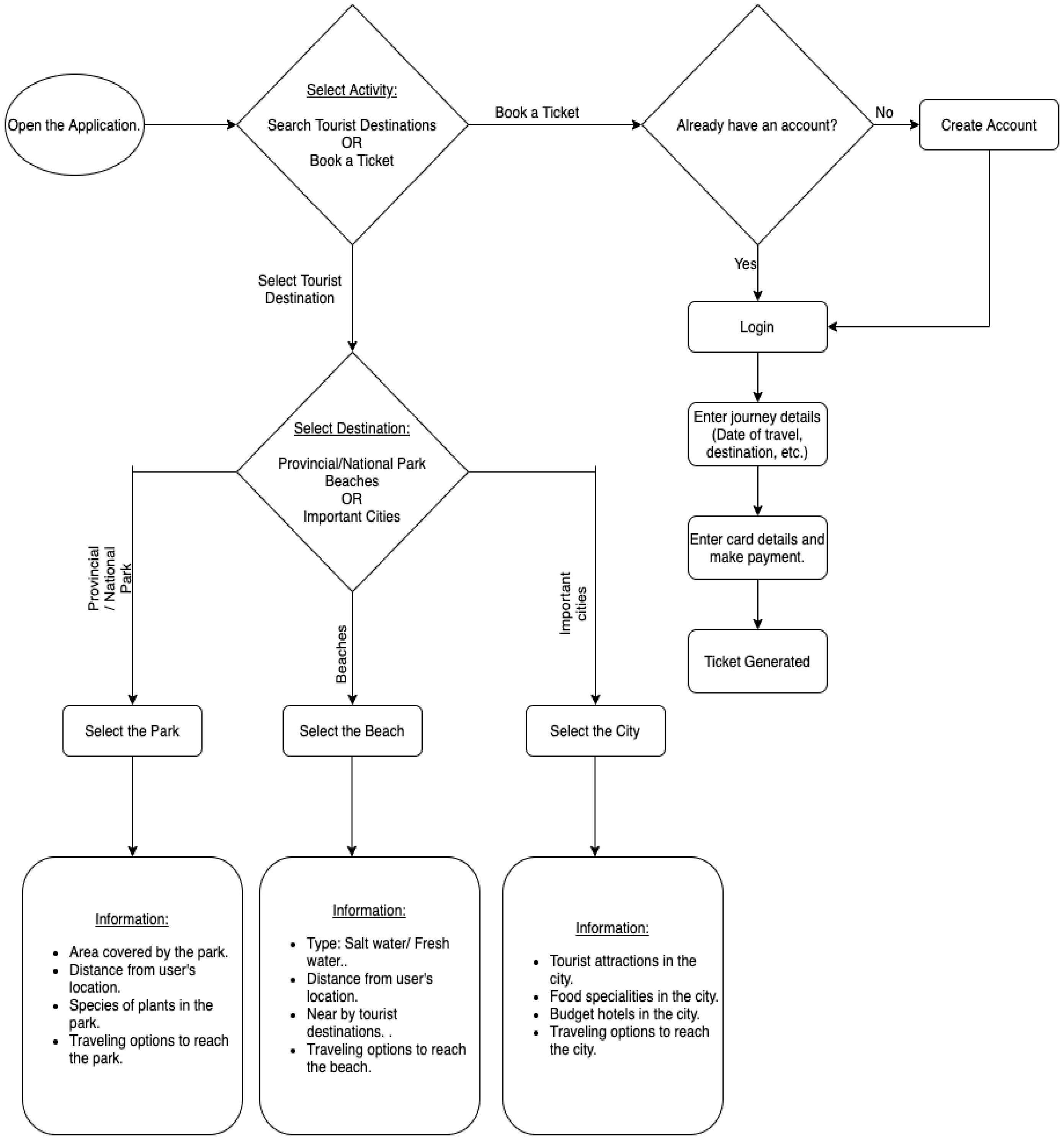


*Figure 2: Sitemap for web and Android application(Drawn using draw.io)*

## Activity Diagram



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 Activity diagram for the proposed application flow. [8] (created using draw.io)*

Once the user opens the application, they will either have the option to select a tourist destination or book a ticket.

- **Select a tourist destination:** They can select 3 types of tourist destinations i.e., Provincial/National Parks, Beaches, and Important cities. Based on their choice, the application will provide the desired information about the location (e.g.: specialties of the location, travel options to reach the destination, etc.).

- **Book a Ticket:** If the user wants to book a ticket, he/she has to log into the application to do so. The application will ask the user if they have a registered account on the application. If they don't have an account, users have to create an account to proceed to the login page. If they have an account already, they will be asked to enter the credentials to log in. Once they are logged in, the application will prompt the user to enter the journey details like date of travel, destination, etc. Once this is done, they will make the payment by entering their credit/debit card details. After the payment is done, the users will get a digital copy of their ticket.

## Data Modelling

We have identified the following entities in our application:

**User:** User entity holds the information of the user. The attributes of this class are User_Id, First_Name, Last_Name, Gender, Email_Id, Phone_No.

**Ticket:** Ticket entity stores the generated ticket data. Ticket_Id, Package_Id, Booking_User_Id, Ticket_Price, Discount_Applied are the attributes of this entity. Package_Id and Booking_User_Id are the foreign keys referencing User entity and Package entity respectively.

**Package:** This entity holds the details of all the available packages with attributes Source, Destination, Package_Id, Price, Transport_Mode.

**Transaction:** Transaction entity stores all the transaction details including Payment_Method, Generated_Ticket_Id. Generated_Ticket_Id is the foreign key referencing Ticket entity.

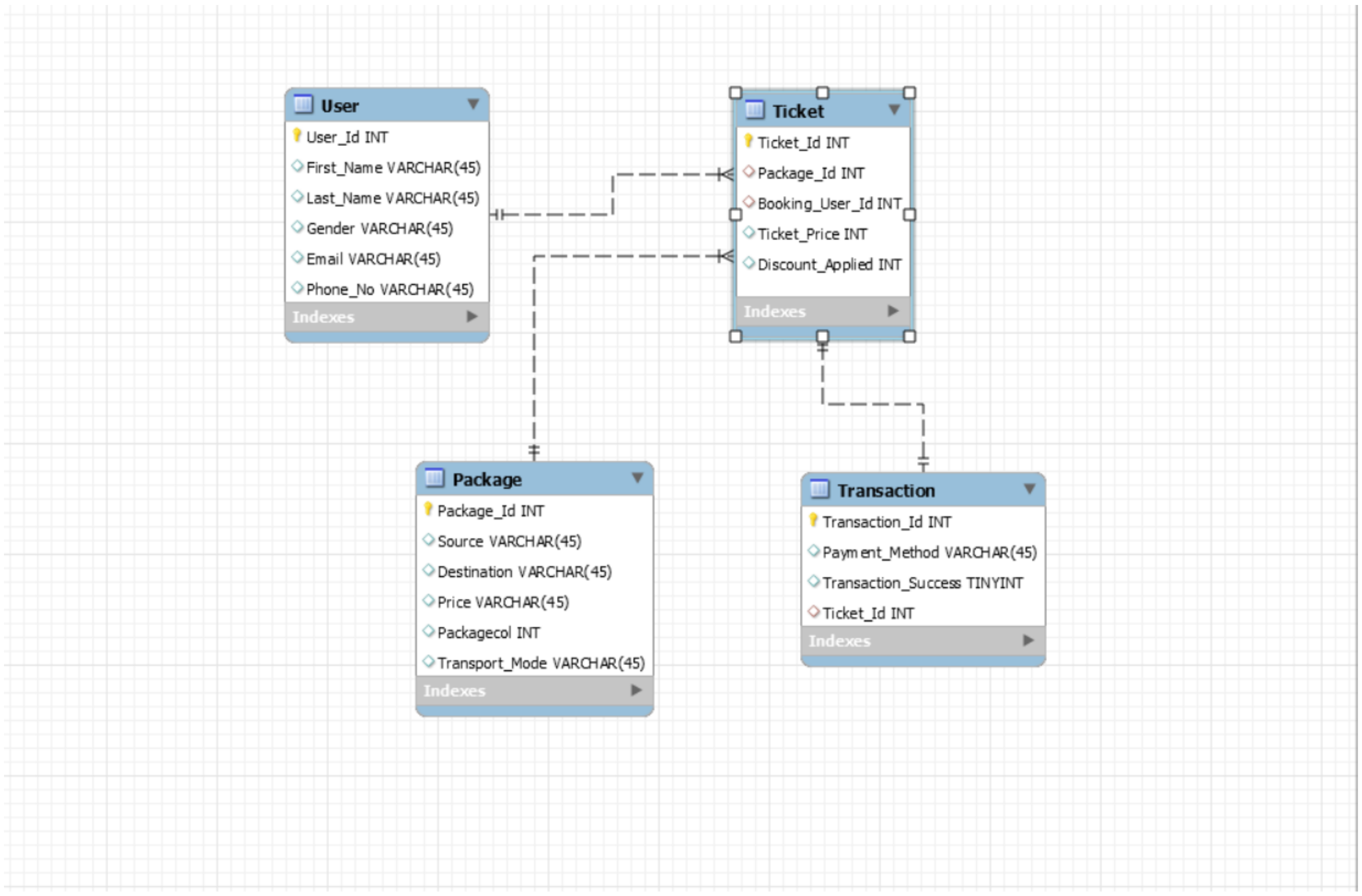


*Figure 4 Entity Relationship Diagram(Drawn using MySQL workbench)*

## Rest Services

Rest APIs are created and consumed as services in both android application and angular website. The following are the rest APIs will be created which will be placed in Elastic container service which acts as source of data for the client-side and communicates with database for the data.

Search API: Uses GET verb which retrieves all the tourist destinations with the searched keyword being passed to it.

TouristDestination API: Uses the GET verb to fetch the selected tourist destination by the user from the database along with the key features of the selected tourist destination.

LoginAPI: Uses the GET verb to fetch the user status in the database by validating against the database.

RegisterAPI: Uses the PUT verb to create an account for the user in the database if the user doesn’t exist in the database.

BookTicketAPI: Uses POST verb with user details and the destination for bus ticket booking.

GenerateTicketAPI: Uses GET verb to generate the ticket with details from BookTicketAPI and the generated ticket is mailed to the user.

These are the Rest APIs considered as of now and in the further development process, APIs will be created in needed.

These APIs will be running independently on Elastic Container Service thereby achieving the granularity and in a loosely coupled environment, and as these services are running independently, there will be high security and will be free from injection attacks.

## References

**[1]** Joyce Lin's 'Deploying a scalable web application with docker and kubernetes' [Online]

Available: https://medium.com/better-practices/deploying-a-scalable-web-application-with-docker-and-kubernetes-a5000a06c4e9 [Accessed on Feb 19, 2020]

[2] Xiaoyun Yang's 'A complete guide to deploying your web app to aws' [Online]

Available: https://codeburst.io/a-complete-guide-to-deploying-your-web-app-to-amazon-web-service-2854ff6bc399 [Accessed on Feb 19, 2020]

[3] AWS's documentation on clusters [Online]

Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/clusters.html [Accessed on Feb 20, 2020]

[4] AWS's documentation on task definitions [Online]

Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definitions.html [Accessed on Feb 20, 2020]

[5] AWS's documentation on rds [Online]

Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_SettingUp.html [Accessed on Feb 20, 2020]

[6] AWS's documentation on Amazon elastic container service [Online]

Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html [Accessed on Feb 20, 2020]

[7] AWS's documentation on Amazon EC2 [Online]

Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html [Accessed on Feb 20, 2020]

[8] "Flowchart Maker & Online Diagram Software", Draw.io, 2020. [Online].

Available: https://www.draw.io/. [Accessed on Feb 17, 2020].