

Event Management Web Application
University of Southern California
EE 547 Project

Rajath Ramegowda
rramegow@usc.edu

Nisha Jacob
nejacob@usc.edu

Muskaan Parmar
muskaanp@usc.edu

December 14, 2022

Contents

1	Summary and Description	3
2	Architecture and Implementation	3
2.1	Technologies and Frameworks	3
2.2	Cloud Provider	4
2.3	Unique Features	4
2.4	Deployment	5
2.4.1	Instance Types	5
2.5	External Dependencies	5
2.6	Database Description	6
3	End-User Experience	7
3.0.1	Login Page	7
3.0.2	Dashboard	9
3.0.3	Student Dashboard:	9
3.0.4	Profile Page:	9
3.0.5	Event Search Page:	9
3.0.6	Search Results Page:	10
3.0.7	Bookings Page:	10
3.0.8	Past Bookings Page:	10
4	Planned work	11
5	Timeline and Milestones	11
6	Contribution	11
7	Challenges	12
8	Conclusion	12
8.1	Document API	13

1 Summary and Description

In this project, we propose to build a web application for managing and coordinating various events at USC. The objective is to build a tool that helps users organize and manage events such as conferences, workshops, and meetings within the university. The ultimate aim is to provide a common platform to host all university events, thereby providing a better reach. We decided to build this application since we faced multiple event management issues at USC events, such as each organization having different ways to check-in at events, having to physically collect coupons to be able to access amenities and so on. These issues were isolated and we try to solve them through our project. Users are categorized into two, namely students and administrators. Students will be able to register for events (and be given a QR code to be used at the time of check-in), view/edit their booking history, view profile settings and authenticate their login. Administrators have added responsibilities of creating and deleting events, managing event details and controlling event check-in and facility management.

The students are the target audience for our project. By using an event management web application, event organizers can streamline the planning process and ensure that their events run smoothly and successfully. It also helps in saving a lot of time as well as provides a better reach to the participants by having a check-in QR code of the event.

2 Architecture and Implementation

2.1 Technologies and Frameworks

For the front-end, we make use of Angular, HTML, CSS and TypeScript to create a webpage (**MEAN** Architecture). MEAN stack is a collection of JavaScript based technologies used to develop web applications. **M** stands for MongoDB, which is a database manager that implements a NoSQL structure. **E** corresponds to Express.js, a framework that supports and is used to host Node.js projects. **A** stands for Angular.js, that is a framework for building apps. It builds upon the classic html framework style and extends it to web applications. **N** corresponds to Node.js, a runtime environment that runs server side web applications. Figure 1 shows the MEAN System Architecture.

Angular, HTML, CSS, and TypeScript can be used together to create modern, dynamic websites. HTML is used to define the structure and content of a web page, CSS is used to add styling, and TypeScript (extended version of JavaScript with static type definitions) is used to add interactive elements to the webpage. CSS, or Cascading Style Sheets, is a stylesheet language that is used to define the look and feel of a website.

We make use of MongoDB to store user, event and registration related data in the database. It is a document-oriented database, which means that it stores data in documents that are organized into collections. This makes it easy to store and query data, as well as to update and modify it. Node.js and Express are used at back-end. Node.js is a runtime environment for JavaScript that allows developers to run JavaScript on the server side, outside of a web browser. This makes it possible to build highly scalable, performance-oriented web applications. Express is a web framework for Node.js that makes it easy to build APIs and web applications. It provides a range of features, such as routing and middleware, that make it easy to handle HTTP requests and responses.

GraphQL is mainly used to handle API calls. GraphQL and RESTful are two ways to make API calls. GraphQL allows clients to specify exactly what data they need from an API, while RESTful APIs use fixed operations to manipulate resources on a server. RESTful APIs use a fixed set of operations, such as GET, POST, and DELETE and are typically organized around resources that clients can access using URLs.

We use GraphQL to handle the various types of queries and other create or modify requests in relation to users, events and registrations. REST endpoints are used for ping, to check the general health of the server. It is also used in the final step, to bind the frontend and backend together.

Javascript, TypeScript(interactive elements), HTML, CSS(webpage style and structure) and GraphQL(query language) are the languages used to build our project. Frameworks used include Angular, Express.js and Node.js. Bootstrap 5 library is used.

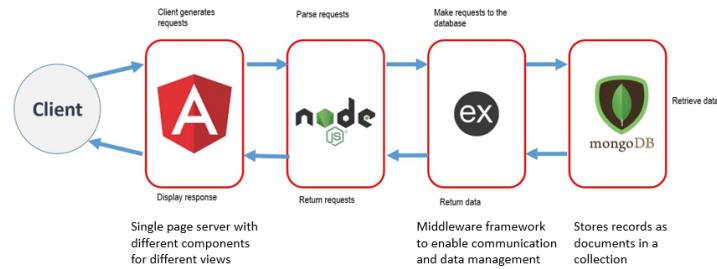


Figure 1: MEAN System Architecture

2.2 Cloud Provider

We use Amazon Web Services (AWS) as a cloud provider. We are making use of two services of AWS: AWS Amplify and AWS Elastic Beanstalk.

Development for the serverless QR code reader is done on AWS Amplify, that lets developers host full-stack applications on AWS. This functionality is triggered when a user tries to scan their QR code during event check-in, allowing the application to take appropriate action. This would ultimately be integrated to the main project. We use our own hosting to test this feature. We use AWS Amplify to test the individual module, qrcode reader.

We make use of a General Purpose compute instance (AWS Elastic Beanstalk) that provides a balance of compute, memory and networking resources and can be used for different workload types. After uploading our application code, Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

2.3 Unique Features

We have added certain features in our project which would allow users to have a smooth process when they want to register for an event:

- **Generating JWT for client authentication during login:** When the user visits the login page, he has to enter the login credentials like username and password. After this, the password goes through an encrypting process/hashing and is stored in the database in hashed form. JSON Web Tokens (JWTs) are used for password authentication by following the steps below:
 - user provides their username and password to the server.
 - server verifies the user's credentials and, if they are valid, creates a JWT that contains information about the user, such as their username and password.
 - server sends the JWT to the client.
 - client stores the JWT and includes it in the Authorization header of subsequent requests to the server.
 - when the server receives a request with a JWT, it verifies the token's signature to ensure that it has not been tampered with. If the signature is valid, the server can trust the information contained in the JWT and use it to authenticate the user.

When the user is logging in for second time, the same hashing occurs and the hashed password is compared with the value in the database to check if the user exists.

Code for issuing JWT Token for login:

```

    JWT_SECRET = <complex string with alphanumeric and symbols>
app.post('/api/login', async (req, res) => {
  const { username, password } = req.body
  const user = await Login.findOne({ username }).lean()
  if (!user) {
    return res.json({ status: 'error', error: 'Invalid username/password' })
  }
  if (await bcrypt.compare(password, user.password)) {
    // the username, password combination is successful
    const token = jwt.sign(
      {
        id: user._id,
        username: user.username
      },
      JWT_SECRET
    )
    return res.json({ status: 'ok', data: token })
  }
  res.json({ status: 'error', error: 'Invalid username/password' })
})

```

- **View specific information about each event:** Users can filter for events based on categories such as location, type and(or) organizer. Users also have the ability to view specific information about each event, including exact location on a map and register for events they like.
- **Register for events with a provision of QR code for check-in:** When a user registers for an event, they can view the registered events via the My Bookings Tab. This tab contains necessary information about each event and contains an event-specific QR code that the user can use to check-in.
- **Ability to create/delete events:** Admins have the extended ability to create upcoming events or delete cancelled events.

2.4 Deployment

2.4.1 Instance Types

AWS Elastic Beanstalk is a fully managed service for deploying and scaling web applications and services. We make use of AWS Elastic Beanstalk to deploy and run our Node.js application on the cloud.

Elastic Beanstalk makes it easy for developers to quickly deploy and manage their applications without worrying about the underlying infrastructure. It automatically scales and provisions the necessary resources to support the incoming traffic to the application. It also provides various tools and integration mechanisms to monitor and troubleshoot the application, such as health and performance metrics, logs, and alarms. This makes it easier for developers to quickly identify and fix issues with their applications.

Overall, Elastic Beanstalk provides a simple and cost-effective way for developers to deploy applications in the cloud.

2.5 External Dependencies

For our project, there are some external libraries, frameworks, APIs, or other components that are needed for the project to run properly. We have used four such dependencies:

QRcode.js: QRcode.js is a JavaScript library for generating and rendering QR codes on the web. It allows us to easily create and display QR codes in web applications, using a simple and

intuitive API. It uses canvas element to render QR codes on the page, and provides a number of customization options for controlling the appearance and behavior of the QR code.

bcryptjs: bcryptjs is a JavaScript library for hashing passwords using the bcrypt algorithm. When a user enters their password, we use bcryptjs to generate a secure hash of the password, and then store the hash in the database. When the user attempts to log in, we use bcryptjs to compare the provided password with the stored password hash, and verify whether the user has entered the correct password or not.

Mongoose: Mongoose was used to work with data in MongoDB in a more object-oriented fashion. It is a popular Object Document Mapper (ODM) library for Node.js and MongoDB. It allows us to define objects with a strongly-typed schema that maps to your MongoDB collections. This makes it easier to enforce data integrity, manipulate data, and perform CRUD operations on the MongoDB data.

Axios: We have use the JavaScript library, Axios for making HTTP requests. It helped us to easily send HTTP requests to REST endpoints and perform CRUD operations. Axios is promise-based, which means that it uses promises to handle asynchronous behavior and to make it easier to work with HTTP requests. It provides a simple, intuitive API for sending and receiving HTTP requests, and supports a wide range of request and response types.

We also make use of external APIs to improve the functionality of our application.

- IPinfo - We use IPinfo to fetch information about a client visiting the website. Currently, we collect the country and IP data from a user. This could later be used to provide geographical analytics of users visiting the application.
- Google Geolocation API - This is used to accurately indicate where an event is taking place (on a map), making it easier for users to reach the location.

2.6 Database Description

MongoDB uses a document-oriented model that allows us to store and manage data in the form of JSON-like documents. This makes it easier to work with data that has complex structures. It allows us to easily query and index your data to support a wide range of applications. Thus, in our case MongoDB is a good choice as it can handle large amounts of data as well as support high levels of concurrency. Collections are the basic unit of organization and data management in MongoDB. For our project, we are creating 4 collections- Login, User, Event and Reservations. The Collection Login stores information related to the login details of the user. The Collection User stores information related to the personal details of every user. The Collection Reservation stores information related to the events for which the user has signed up. The Collection Event stores information related to the events being help in present or future. Figure 2 shows the MongoDB Schema with four collections.

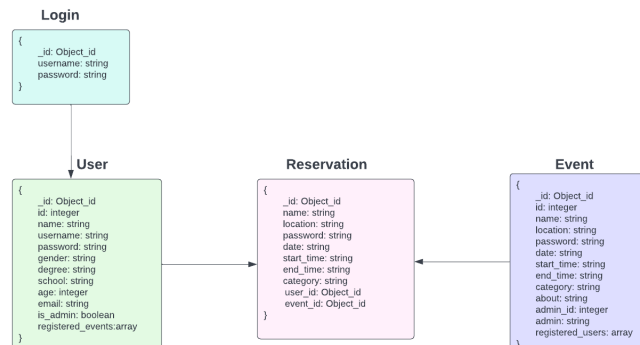


Figure 2: MongoDB Schema

3 End-User Experience

There are two types of users for our website: Privileged and Non privileged users.

Non-privileged Users (Students): These users can sign-up or log-in to their accounts. Upon successful authentication, the user can view four available option on the dashboard: Profile, Search, Bookings, Past Bookings. The profile page lets the user view and edit his/her details – entered during sign-up. The search page enables the user to search for events based on either Location or Organizer or the Type of event. In the Bookings page, the user can view upcoming event registrations along with a QR code they can use to check-in at the event time. Past Bookings lets the user view past events attended by the user.

Privileged Users (Admins): Admins have the added responsibility of creating and deleting events. An admin is not allowed to register to the event he/she has created.

The Figure 3 shows the webpage workflow. The section below explains the webpage features.

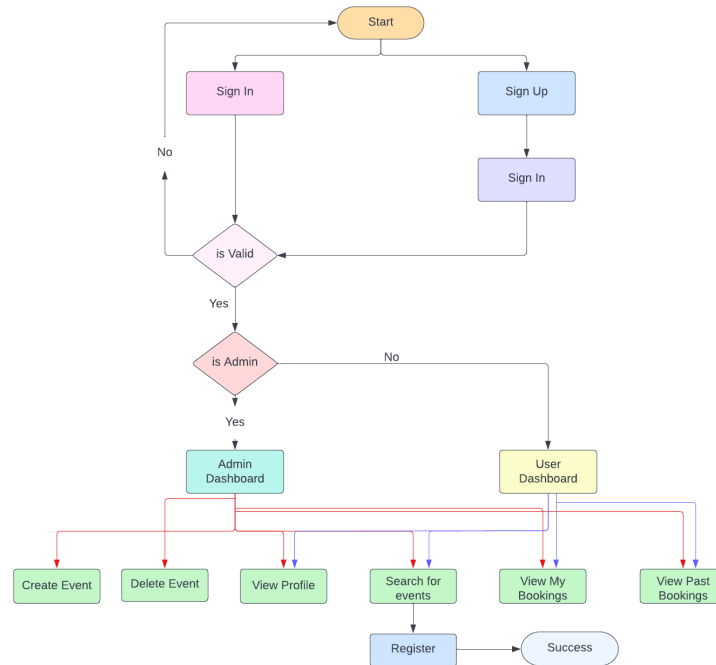


Figure 3: Webpage Workflow

3.0.1 Login Page

The homepage shows the user two options:

- **Sign in:** If the user already has an account, he/she simply has to enter username and password in the fields provided and click the "Sign in" button to access their personal account. Figure 4 shows the login page.
- **Sign up:** If the user doesn't have an account yet, he/she can click the "Sign up" button to create one. The user needs to provide some basic information such as your name, username, email address, gender, degree, school, student or admin and a password to get started. After this, they can directly sign in when they visit the webpage the next time. Figure 5 shows the sign up page.

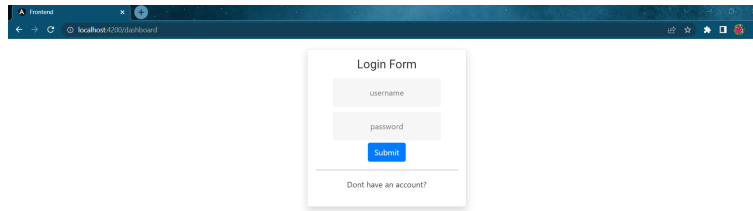


Figure 4: Login page

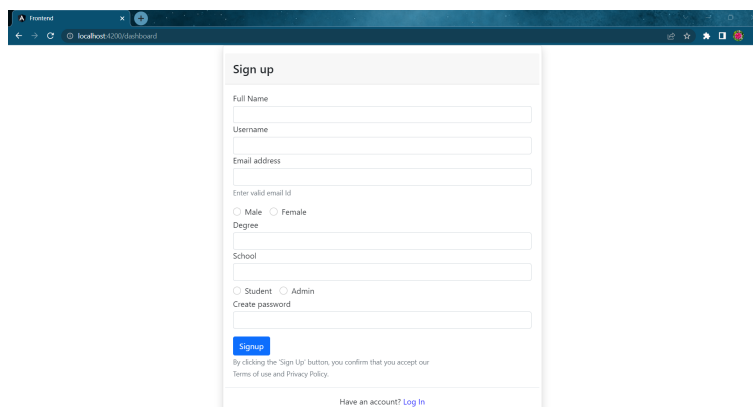


Figure 5: Sign up page

3.0.2 Dashboard

After the login, as per the value of isAdmin variable, the user is shown the dashboard for admin or student

Admin Dashboard: This dashboard is only meant for admin. They have the ability to create and delete events along with registering and viewing of events. An admin cannot register for an event that he/she has created.

3.0.3 Student Dashboard:

This dashboard is only meant for students. From here, students can register for an event.

3.0.4 Profile Page:

This section displays information of the user that is entered at sign up. The user has the option to edit details here. Figure 6 shows the profile page.

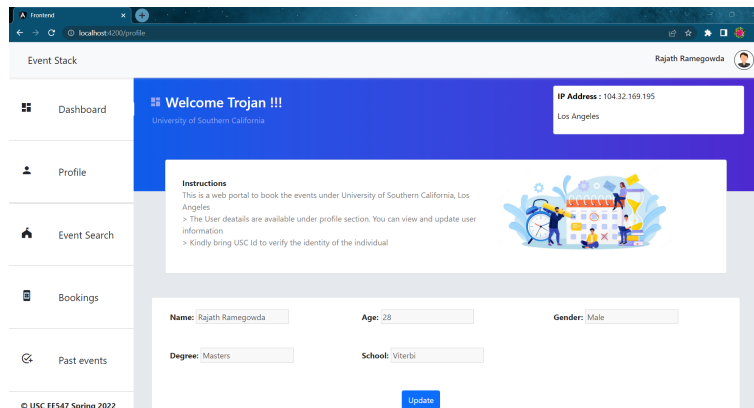


Figure 6: Profile page

3.0.5 Event Search Page:

User can filter event search based on event location, event organizer and event type. They will be able to view all future events satisfying the criteria. Figure 7 shows the event search webpage.

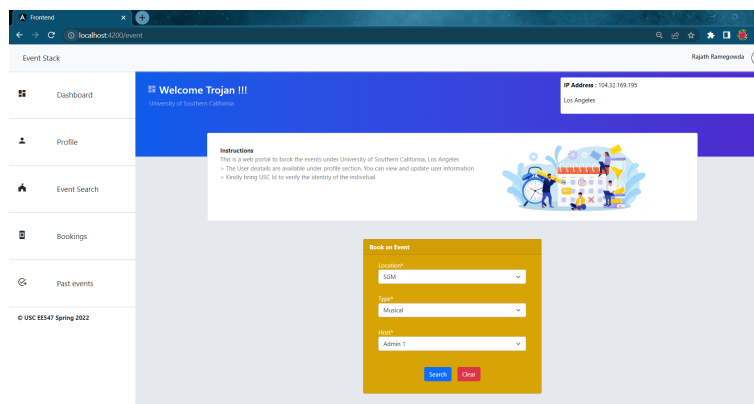


Figure 7: Event Search page

3.0.6 Search Results Page:

This page shows all events satisfying the search criteria. User can proceed to register for an event, thereby reducing the number of available seats in the event by 1. Figure 8 shows the search results webpage.

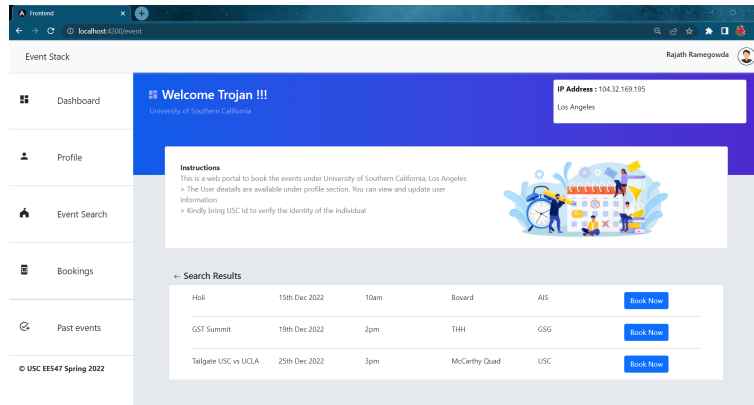


Figure 8: Search Results Page

3.0.7 Bookings Page:

This page shows all the events that the user has registered for along with a QR code that can be used at the time of check-in. It highlights event details as well as precise event location using Google Geolocation API. Figure 9 shows the bookings webpage.

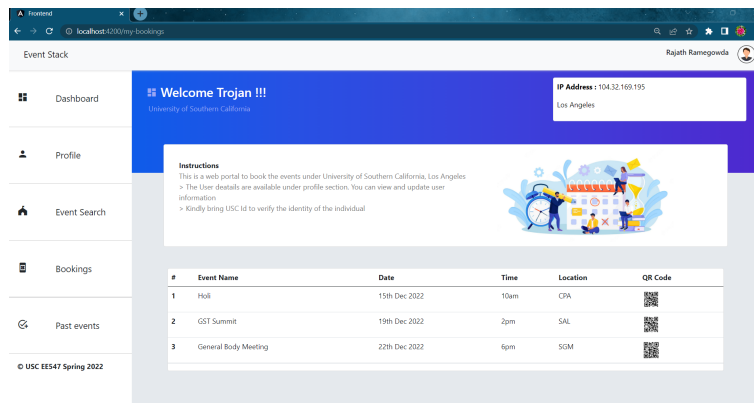


Figure 9: Bookings page

3.0.8 Past Bookings Page:

This page shows all past events that the user has attended along with a status showing 'Completed'. Figure 10 shows the past bookings webpage.

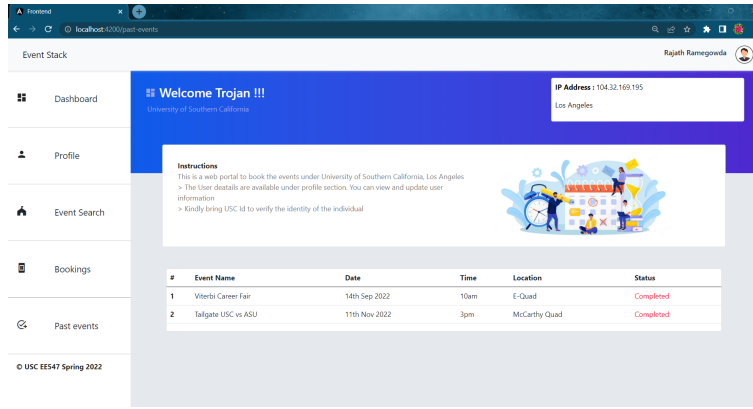


Figure 10: Past Bookings Page

4 Planned work

- Integrating the QR code scanner into the web application. This is currently a work in progress. It is currently hosted on AWS Amplify at <https://main.d2nvlg8gzu2b0r.amplifyapp.com/>. Integrating this into the application requires additional helper functions.
- Building UI for the Admin homepage

5 Timeline and Milestones

High Level Timeline	Backend	Frontend
12-01-2022	Implementation of GraphQL queries and mutations for users	Implementation of structure in the project and creating multiple components
12-05-2022	Implementation of GraphQL queries and mutations for admins. Connection with MongoDB is setup	Writing style sheets, implementing libraries and writing TypeScript to connect all components
12-09-2022	Implementation of GraphQL queries for events and authentication of user at sign-in	Implementation of Angular Data Services and writing RESTful APIs along with Apollo Angular GraphQL client side queries
12-12-2022	Implementation of backend queries for check-in QR code	Connecting to backend and data binding from the json response
12-13-2022	Merge both front end and back end. Hosting the website on AWS	Implementation of QR code reader

Table 1: Timeline in Retrospective

6 Contribution

The following is the rough breakdown of roles and responsibilities we plan for our team:

- **Rajath:** Client UX and UI. Built client side interface for API and create rough pages with dynamic server content". Assisted with design and styling. Also worked on introducing JWT

during login and QRcode for event registration.

- **Nisha:** Backend server design and API. Implemented GraphQL resolvers. Assisted with some layout elements.
- **Muskaan:** Database access and HTML page design. Handled overall “look and feel” as well as MongoDB operations. Coordinated the API integration between Nisha and Rajath.

All team members will work on the final presentation, slides, and report.

7 Challenges

We were conceptually clear with our project topic and how to go about implementing it, however faced a couple of challenges during implementation.

- Angular was chosen for our front end since it is a single page web application, however some roadblocks were faced while integrating with GraphQL. We got over this by using Apollo Angular client.
- Angular data services was not a very easy task to navigate through.
- QRCode generator and scanner required the usage of external libraries and research of a certain depth.
- We also faced a couple of issues using MongoDB but were able to overcome these while establishing connections through mongoose.

8 Conclusion

We have been able to build an Event Management System that serves as a common platform for all event organisers to host their events and gauge event response. The application also enables the smooth check-in of users on reaching the event location. Students and Admins can keep a track of events they have attended and number of users attending an event respectively.

We misunderstood the ease with which Angular could be integrated with the rest of the frameworks. Based on this assumption, Angular was chosen to be our front end language and we eventually faced a lot of issues while integrating with GraphQL. Ultimately, we figured out that the smoothest way to integrate both was using Apollo Angular Client, which is an ultra-flexible GraphQL client for Angular.

References

- [1] Node, Express, Angular 7, GraphQL and MongoDB CRUD Web App, <https://www.djamware.com/post/5c75d68880aca754f7a9d1ed/node-express-angular-7-graphql-and-mongodb-crud-web-app>
- [2] MongoDB-Atlas-Nodejs, <https://hevodata.com/learn/mongodb-atlas-nodejs/>
- [3] Stackoverflow, <https://stackoverflow.com/>
- [4] Mongoose, <https://www.npmjs.com/package/mongoose>
- [5] Bootstrap 5.2.3, <https://blog.getbootstrap.com/>
- [6] Reactive Pattern for Data Services in Angular, <https://medium.com/tkssharma/reactive-pattern-for-data-services-in-angular-859e4ca39099>
- [7] Apollo Angular, <https://the-guild.dev/graphql/apollo-angular/docs>

[8] How to use the Fetch API in Angular, <https://stuarttottle.medium.com/use-the-fetch-api-in-angular-1acafa67bbf2>

Appendix

8.1 Document API

User Schema:

```
const userType = new GraphQLObjectType({
  name: 'User',
  description: 'This shows all users (admins and non-admins)',
  fields: () => ({
    id: { type: GraphQLNonNull(GraphQLInt) },
    name: { type: GraphQLNonNull(GraphQLString) },
    username: { type: GraphQLNonNull(GraphQLString) },
    password: { type: GraphQLNonNull(GraphQLString) },
    gender: { type: GraphQLNonNull(GraphQLString) },
    degree: { type: GraphQLNonNull(GraphQLString) },
    school: { type: GraphQLString },
    age: { type: GraphQLInt },
    email: { type: GraphQLNonNull(GraphQLString) },
    isAdmin: { type: GraphQLNonNull(GraphQLBoolean) },
    registeredEvts: {
      type: new GraphQLList(GraphQLInt),
    }
  })
})
```

Event Schema:

```
const eventType = new GraphQLObjectType({
  name: 'Event',
  description: 'This shows the events',
  fields: () => ({
    id: { type: GraphQLNonNull(GraphQLInt) },
    name: { type: GraphQLNonNull(GraphQLString) },
    location: { type: GraphQLNonNull(GraphQLString) },
    date: { type: GraphQLNonNull(GraphQLString) },
    start_time: { type: GraphQLNonNull(GraphQLString) },
    end_time: { type: GraphQLString },
    category: { type: GraphQLNonNull(GraphQLString) },
    about: { type: GraphQLString },
    adminId: { type: GraphQLNonNull(GraphQLInt) },
    registeredUsers: {
      type: new GraphQLList(GraphQLInt)
    }
  })
})
```

Reservation Schema:

```
const reservationType = new GraphQLObjectType({
  name: 'Reservation',
```

```
fields:() => ({
  registrationId: {
    type: GraphQLNonNull(GraphQLString)
  },
  eventName: {
    type: GraphQLString
  },
  adminId: {
    type: GraphQLInt
  },
  location: {
    type: GraphQLString
  },
  start_time: {
    type: GraphQLString
  },
  eventCategory: {
    type: GraphQLString
  }
})
});
```