SYSTEMS DESIGN DOCUMENT

GOGO - BETTER WITH YOU

TABLE OF CONTENTS

A. HIGH-LEVEL DESCRIPTION OF THE MAJOR FRONT-END COMPONENTS

The following tables elaborate on our application's major front-end components, responsibilities, and interactions with other front-end components.

1. Login Page

<Login />	
Sub-components: ● <FacebookLogin/> ● <GoogleLogin/>	
Responsibilities: - Enables login with Email - Enables login with Facebook - Enables login with Google	Collaborators: - Xin Yi Hu - TBA

2. Signup Page

<SignupHub />	
Sub-components: ● <AccountSetup /> ● <Signup /> ● <FacebookAuth /> ● <GoogleAuth />	
Responsibilities: - Enables registration with Email - Enables registration with Facebook - Enables registration with Google - Enables user to input age and gender during registration	Collaborators: - Bharath Varma Chamathi - TBA

3. Dashboard Page

<Dashboard />	
Sub-components: ● <Sidebar />	

Responsibilities:
- Enables navigation to the following:
  - Events tab
  - Events creation page
  - Bio Tab
  - Invites tab
	Collaborators:
	- Beatrice Lim-Kian-Siang
	- TBA

4. Events Tab

<EventsTab />	
Sub-components: ● <Events /> ● <CreateEvents />	
Responsibilities: - Enables users to view and interact with upcoming events posted on the app. - Enables users to post events on the app.	Collaborators: - Jeremy Neilson - TBA

5. Invitations Tab

<InvitationsTab/>	
Sub-components: ● <Invitation/>	
Responsibilities: - Enables user to view and interact with event invitations that the user has sent or received	Collaborators: - TBA

6. Bio Tab

<BioPage />	
Sub-components: ● <ProfilePicture/> ● <NameAgeGender/> ● <Interests/> ● <UserBio/>	

Responsibilities:
- Enables user to display their:
  - Name
  - Age
  - Gender
- Enables user to display and edit their:
  - Interests
  - Biography
	Collaborators:
	- Farhan Bin Faisal
	- Athul Vincent
	- TBA

B. ASSUMPTIONS/DEPENDENCIES ABOUT THE OPERATING ENVIRONMENT

1. Supported Operating Systems:

- Windows
- MacOS
- Linux

2. Databases:

- Supports MongoDB Atlas (a NoSQL database)

3. Dependencies:

- Requires npm (Node.js v18.16.0) version 9.5.1 to be installed
- Requires the following packages:
  - ExpressJS (version 4.18.2)
  - Passport (version 0.6.0)
  - Mongoose (version 7.2.1)
  - Nodemon (version 2.0.22)

4. Network configurations:

- Need to ensure no other application is running on port 5000
- Client runs on port 3000. Ensure no other application is running on this port.
- For MacOS, need to turn on *Airplay receiver* as it runs on port 5000 by default

C. SYSTEM ARCHITECTURE

We decided to go forward with a three-tiered architecture which is composed of the following:

- Presentation Tier (ReactJS)
- Business Logic Tier (NodeJS and ExpressJS)
- Data Tier (MongoDB)

Briefly, the presentation tier consists of the UI user directly interacts with. The business logic tier maps the user's actions at the presentation tier to the underlying functionality of the application. This includes getting and posting data to the database (e.g., when the user presses a certain button). Finally, the data tier consists of the database, which stores the application's required information (user, events, interest information, etc.).

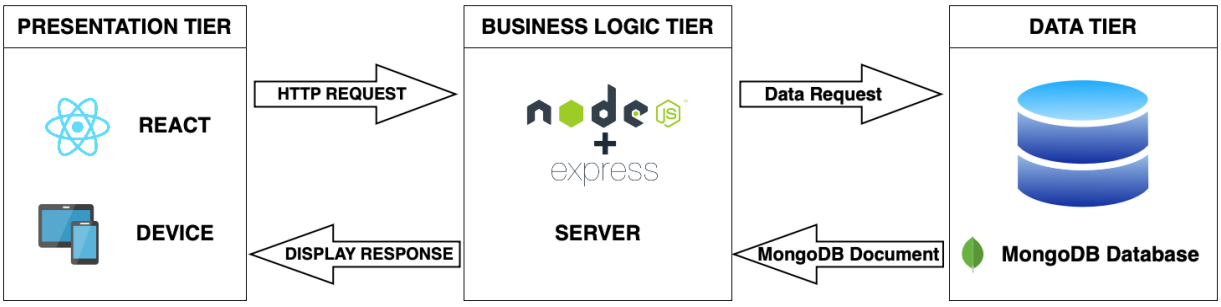

Figure 1: High-level overview of GoGo's software system architecture

This architecture was chosen because it is easy to scale horizontally, fits well with the MERN stack, and can handle a huge data stream. Alongside this, we have a small team and a relatively small codebase. Therefore, we decided to keep it simple and go forward with a monolithic architecture for our *Logic Tier.*

D. SYSTEM DECOMPOSITION

1. Database Design

As mentioned in the previous section, we are using MongoDB atlas for our application's database. Currently, we are following a normalized data model where all schemas are linked using the username. The currently implemented schemas in GoGo are shown in the Figure below.

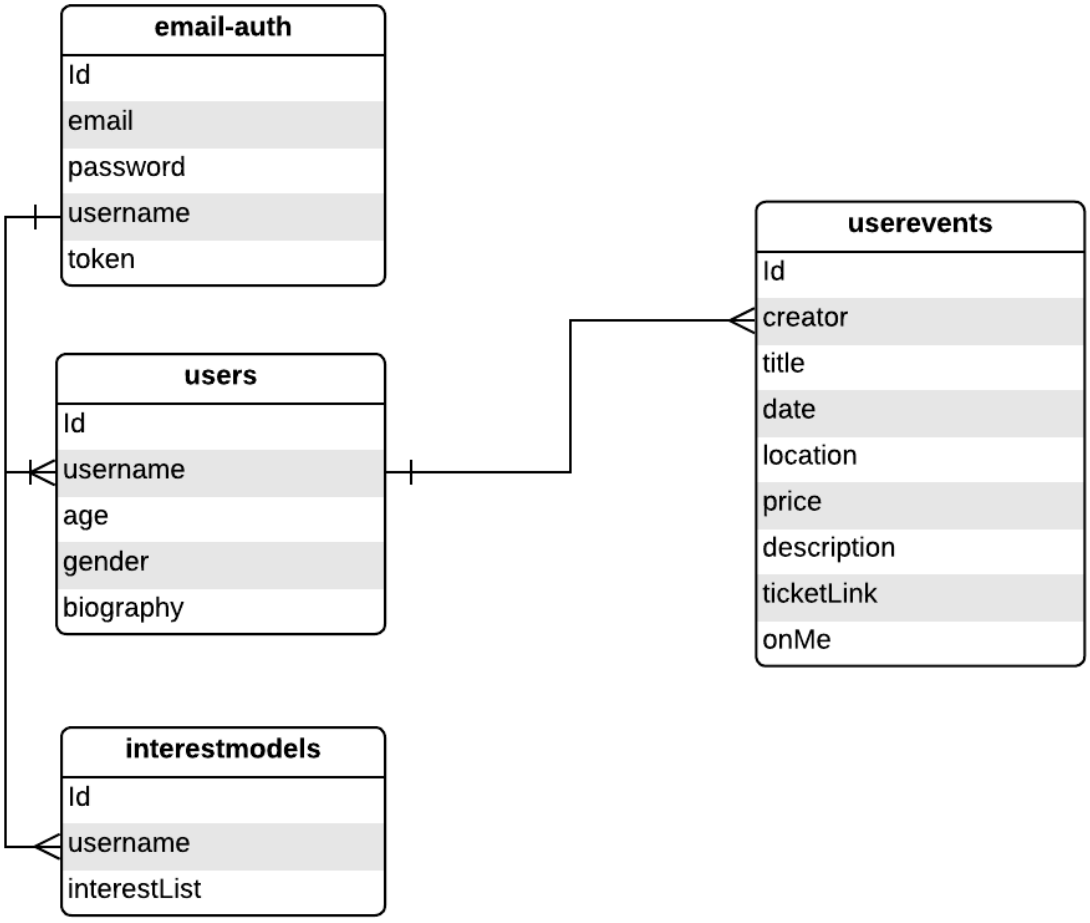

Figure 2: MongoDB schemas/documents currently used by GoGo's Data Tier

2. Relating system architecture to detailed design

In this section, we will elaborate on how the front-end components **(Section A)** interact with the server and the database. In doing so, we will refer to the **System Architecture diagram (Figure 1)** and the **Database Design diagram (Figure 2)** when appropriate.

Broadly, when a ReactJS component needs to retrieve information from the database, it sends HTTP requests to the server at the appropriate end-point using methods from the **Axios Library**. This request is processed by the **ExpressJS router (Figure 1),** which interacts with the **MongoDB database (Figure 1)**. After interaction with the database, the router sends back a response to the front-end component. This response would be composed of status codes (series 200 for success and series 400 for errors/failures) and JSON documents (depending on the nature of the request). In case of errors (status codes of series 400), they will be handled appropriately by the frontend logic.

Interactions between a subset of GoGo's front-end components with the server and database are described below

a. <Login />

This component makes a **POST** request to the server at the **/login** endpoint (with the input email and password as body parameters). This request is processed by an ExpressJS router, which interacts with the MongoDB database **(Figure 1).**

If an **email-auth** document (**see Figure 2**) with a matching email and password is found in the database:

- The server generates a JSON Web Token(JWT) and stores it in the **email-auth** document.
- The updated document is saved in the database.
- A response (status code: 200) with the **email-auth** document is sent back to the <Login /> component.
- User is navigated to the dashboard page

If an **email-auth** document is not found in the database

- The server throws an error with an appropriate message (status code: 400)
- This error is handled by the front-end logic, which displays that the user needs to input a registered email-password combination

b. <CreateEvents />

After the user submits all necessary information (creator name, ticket price, date, etc.) of the event and clicks the "**Create Event**" button, a **POST** request is made to the server at the **/userevents** end-point (with a JSON body containing the event information). This request is processed by an ExpressJS router, which interacts with the MongoDB database **(Figure 1).**

If the **userevents document (Figure 2)** is successfully inserted into the database, a response with status code 200 is returned to the front end. Subsequently, an alert message is shown to the user indicating success. In terms of errors, we are currently not expecting any during this action.

c. <Interests />

Upon navigation to BioPage, the **<Interest />** component makes a **GET** request at the **/userInterests/:username** endpoint to get the current interests array of the user.

If a userInterests document is found:

- A response (status code 200) with the **userInterests document (Figure 2)** is sent back to the front-end.

If the **userInterests** document is not found in the database

- The server throws an error with an appropriate message (status code of 400)
- This error is handled by the front-end logic which interprets the error as an empty interestList and hence does not display userInterests at that time

If the user edits their interest and saves the changes, a **POST** request is made to the server at the **/userInterests** endpoint (with a new interestList as a body parameter). This request is processed by an **ExpressJS router** that interacts with the **MongoDB database**.

If the new userInterests document with the updated interestList is successfully posted in the database:

- A response (status code 200) is sent back by the server
- A GET request is made by the component again at the /userInterests/:username endpoint to retrieve and display the updated set of interests

Currently we are not expecting any errors to be generated in this process

The next page shows a UML Sequence Diagram that portrays some interactions between a subset of front-end components and the server during an example application usage session.

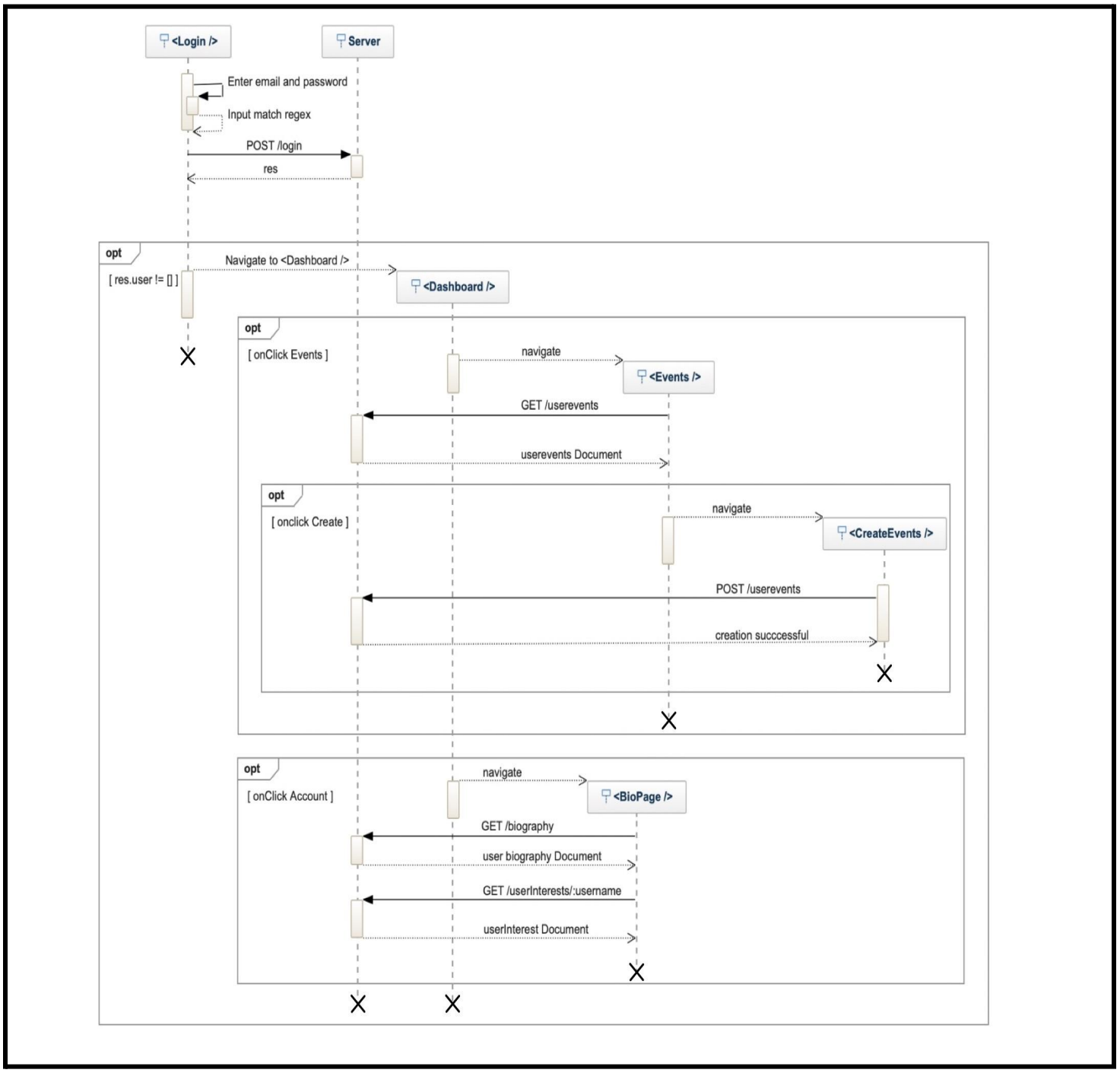

Figure 3: Sequence Diagram of GoGo indicating interactions between certain front-end components and server during an example application usage session.