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 /> ● <EventItem />	

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 /> ● <EventTags /> ● <EventFilter />	
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. Request Tab

<RequestsPage/>	
Sub-components: ● <RequestsForMe/> ● <RequestsByMe/> ● <RequestItemForMe/> ● <RequestItemByMe/>	
Responsibilities: - Enables user to view and interact with event requests that the user has sent or received	Collaborators: - Athul Vincent - Bharath Varma Chamathi - Xin Yi Hu

6. Bio Tab

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

Responsibilities:
- Enables user to display their:
  - Name
  - Age
  - Gender
	Collaborators:
	- Farhan Bin Faisal
	- Athul Vincent
	- Beatrice Lim-Kian-Siang
	- Jeremy Neilson

- Created Events
- Enables user to display and edit their:
  - Interests
  - Biography	

7. Chat Page

<ChatPage />	
Sub-components: ● <ChatMessageInput /> ● <ChatRoomList /> ● <ScrollableChatBox />	
Responsibilities: - Enables user to see their existing chat rooms. - Enables user to send messages to others in the chat rooms. - Enables user to in real time when their chat partner is typing	Collaborators: - TBA

8. Promoters Request Tab

<PromoterRequestsPage/>	
Sub-components: ● N/A	
Responsibilities: - Enables businesses to send requests to users to act as promoters for their events.	Collaborators: - Athul Vincent -

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)
- Mongoose (version 7.2.1)
- Nodemon (version 2.0.22)
- Socket.io (version 4.7.1)
- Multer (version 1.4.5)

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.).

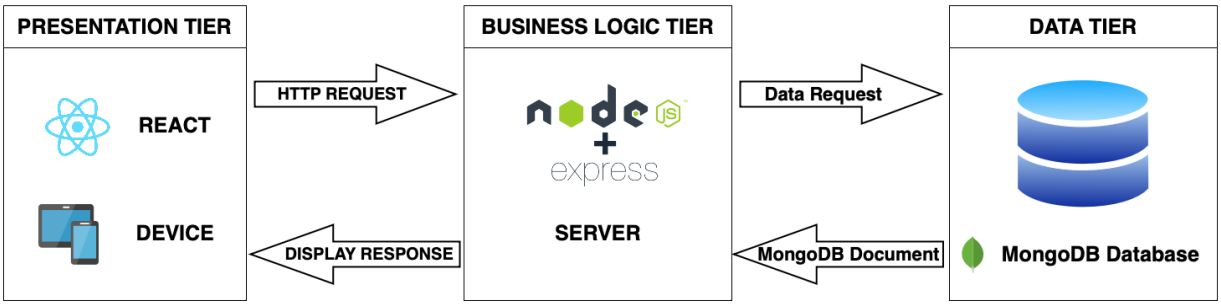

Figure 1: High-level overview of GoGo's software system architecture

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.

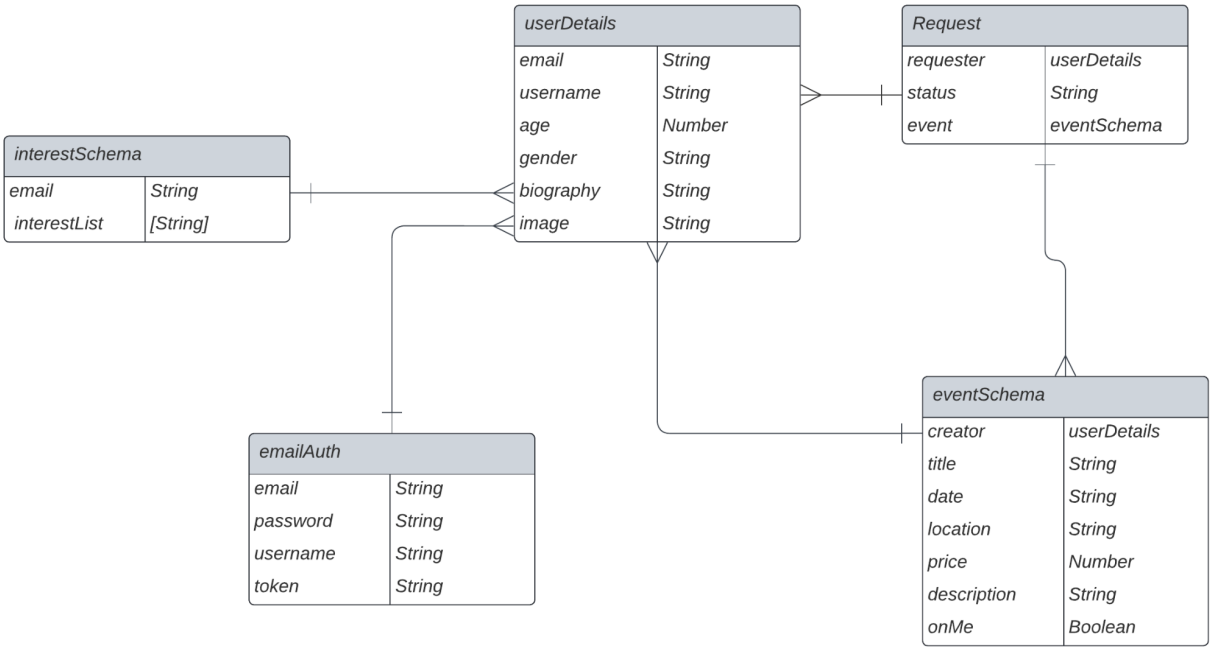

Figure 2: MongoDB schemas/documents currently used by GoGo's Data Tier

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.

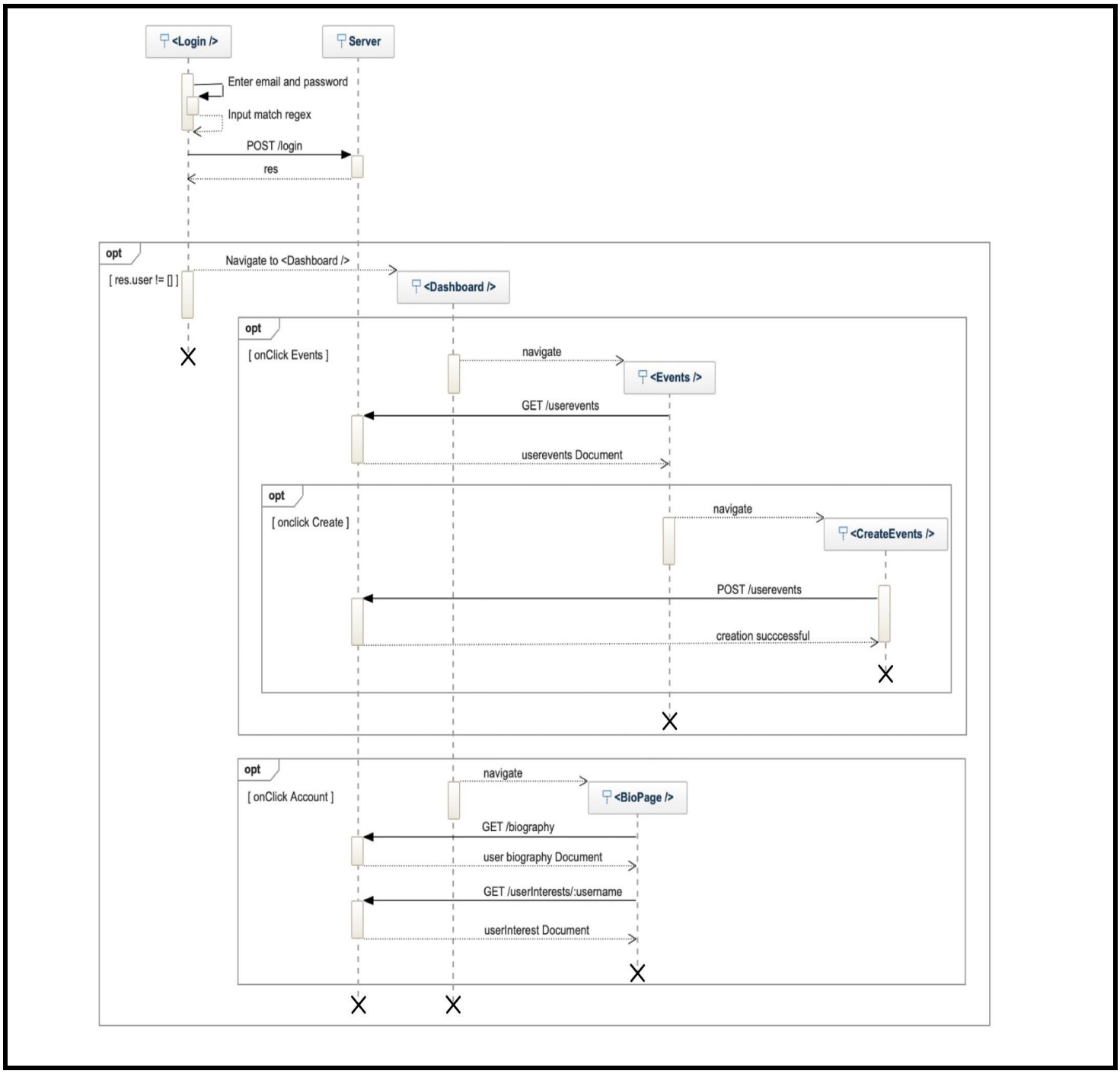

Figure 3: Sequence Diagram of GoGo indicating interactions between certain front-end components and server during an example application usage session.

E. GOGO API DOCUMENTATION

The API is available at http://localhost:3000/api [Inspired by simple-rent-api]

SIGNUP

SIGNUP WITH EMAIL, PASSWORD AND USERNAME

- Makes an email-auth object with the given information
- POST /email-auth/
- Example Body

```
{
        "email": “test@gmail.com”,
        "password": “12341234”,
        “username”: “test user”
}
```

-

Status Codes	
200 OK	Indicates a successful response
400 Bad Request	Indicates that the parameters provided are invalid.

LOGIN

LOGIN WITH EMAIL AND PASSWORD

- Generate session token
- POST /login/
- Example Body

```
{
        "email": “test@gmail.com”,
        "password": “12341234”
}
```

- Status codes

Status Code	
200 OK	Indicates a successful response
400 Bad Request	Indicates that the parameters provided are invalid.

VERIFY SESSION TOKEN

- Verify session token and return user information
- GET /login/
- Status codes

Satus Code	

200 OK	Indicates a successful response
400 Bad Request	Indicates that the parameters provided are invalid.

CHATS

GET ALL CHAT ROOM DOCUMENTS

- Returns all chat documents
- GET /chats/

- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No chat room document found

GET ALL CHAT ROOM DOCUMENTS OF A PARTICULAR USER

- Returns all chat documents of a particular user
- GET /chats/:email

- Parameters

Name	Type	In	Description
email	string	path	Specifies the chat rooms of a particular user

- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No chat room document found associated with the given email

GET ONE PARTICULAR CHAT ROOM

- Returns a chat documents associated with a particular roomID
- GET /chats/usingRoomID/:roomID
-
- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No chat room document found with the given roomID

CREATE A NEW CHAT ROOM

- POST /chats/
- Example Body

```
{
        "chatHistory": [],
        "participantsUsernames": ["bharath_chat", "farhan_chat"],
        "partipants": ["bharath_chat@gmail.com", "farhan_chat@gmail.com"],
        "roomID": "bharath_chat@gmail.comfarhan_chat@gmail.com"
}
```

- Status Codes

Satus Code	
201 Created	Indicates that the chat room has been created successfully.
400 Bad Request	Indicates that the parameters provided are invalid.

UPDATE CHAT HISTORY

- PATCH /chats/:roomID
- Example Body

```
{
   "currentChatHistory": [],
    "newMessage":   {
                      “senderEmail”: "bharath_chat@gmail.com",
                      “newMessage”:  "How are you?”
                    }
}
```

- Parameters

Name	Type	In	Description
roomID	string	path	Specifies the chat room id.

- Status Codes

Satus Code	
201 Created	Indicates that the updated chatroom has been posted successfully.
400 Bad Request	Indicates that the parameters provided are invalid.

DELETE A CHAT ROOM

- DELETE /chats/:roomID
- Parameters

Name	Type	In	Description

roomID	string	path	Specifies the chat room id.

- Status Codes

Satus Code	
204 No Content	Indicates that the chat room has been deleted successfully.
400 Bad Request	Indicates that the parameters provided are invalid.
404 Not found	Indicates that there is no chat room with the specified roomid

EVENTS

GET ALL CREATED EVENTS

- Returns all event documents
- GET /userevents/

- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No event document found

CREATE AN EVENT

- POST /userevents/
- Example Body

```
{
        "eventID": “533a36b8-e878-4e75-909a-be96af9d268b”,
        "creator": “”,
        "title": “Movie Night”,
        "date": "2023-07-30",
        "location": "20 Bloor Street East",
        "price": "0",
        "description": "Jeremy’s birthday movie night",
        "ticketLink": "N/A",
        "onMe": "True"
}
```

- Status Codes

Satus Code	
201 Created	Indicates that the event has been created successfully.
400 Bad Request	Indicates that the parameters provided are invalid.

GET ONE PARTICULAR EVENT

- Returns event documents associated with a particular roomID
- GET /userevents/:email
- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No event document found with the given creator

UPDATE NUMBER OF REQUESTS FOR A PARTICULAR EVENT

- POST /userevents/numRequests
- Status codes

Status Code	
200 OK	Indicates a successful response
404 Not Found	Event to update numRequests field is not found

PROFILE PICTURE

GET PROFILE PICTURE OF CURRENT USER IN SESSION

- Returns profile picture for the current user account.
- GET /image/:email

- Parameters

Name	Type	In	Description
email	string	path	Specifies the profile picture of a particular user

- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No profile picture found.

CREATE A PROFILE PICTURE FOR THE CURRENT USER IN SESSION

- POST /image/

- Status Codes

Satus Code	
200 OK	Indicates that profile picture has been successfully uploaded to the database.
404 Not Found	No user found.

INTERESTS

GET ALL USER-INTEREST DOCUMENTS

- Returns all userInterest documents
- GET /userInterests/

- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No userInterest document found

GET ALL userInterest DOCUMENTS OF A PARTICULAR USER

- Returns all chat documents of a particular user
- GET /userInterests/:email
- Parameters

Name	Type	In	Description
email	string	path	Specifies the userInterest doc of a particular user

- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No userInterest document found associated with the given email

CREATE A userInterest DOCUMENT

- POST /userInterests/
- Example Body

```
{
```

"useremail": “dmitri.farhan@gmail.com”,
“interestList”: [“soccer, “hockey”]
}

- Status Codes

Satus Code	
201 Created	Indicates that the userInterest doc has been created successfully.
400 Bad Request	Indicates that the parameters provided are invalid.

DELETE A userInterest DOCUMENT

- DELETE /interests/:email
- Parameters

Name	Type	In	Description
email	string	path	Specifies the userInterest id.

- Status Codes

Satus Code	
204 No Content	Indicates that the userInterest doc has been deleted successfully.
400 Bad Request	Indicates that the parameters provided are invalid.
404 Not found	Indicates that there is no userInterest doc with the specified roomid

REQUESTS

GET A LIST OF REQUESTS BY WHO CREATED THEM

- Returns all requests made by the specified requester
- GET /by/:requester

- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No request document found

GET A LIST OF ACCEPTED REQUESTS

- Returns all requests that the requester accepted
- GET /accepted/:requester

- Status codes

Satus Code	

200 OK	Indicates a successful response
404 Not Found	No request document found

GET A LIST OF PENDING REQUESTS OF A EVENT

- Returns all requests of a specific event that are pending
- GET /pending/:event

- Status codes

Satus Code	
202 OK	Indicates a successful response
400 Error	No request document found

PATCH ACCEPTED REQUEST

- Accept the specified event by changing its “status” to “accepted”
- PATCH /accept/:_id

- Status codes

Satus Code	
203 OK	Indicates a successful change
400 Error	No request document found

PATCH REJECTED REQUEST

- Reject the specified event by changing its “status” to “rejected”
- PATCH /reject/:_id

- Status codes

Satus Code	
203 OK	Indicates a successful change
400 Error	No request document found

PROMOTER REQUESTS

GET A LIST OF PROMOTERS REQUESTS BY EVENT

- Returns all promoter requests made for a event
- GET /promoter-requests/event/:event_id

- Status codes

Satus Code	

200 OK	Indicates a successful response
404 Not Found	No promoters request document found

POST A PROMOTER REQUEST

- POST /promoter-requests
- Example Body

```
{
        "requesteeEmail": “athul.vincent@gmail.com”,
}
```

- Status codes

Satus Code	
202 OK	Indicates a successful response
404 Error	No promoter request was issued
409	Promoter request already exists

BUSINESS ACCOUNTS

CREATE A BUSINESS ACCOUNT WITH EMAIL, PASSWORD AND BUSINESS NAME

- Creates an email-auth object and a businessDetails object for the business
- POST /business/create/
- Example Body

```
{
        "email": “bharath_business@gmail.com”,
        “password”: “business-password”,
        “businessName”: “bharath’s business”
}
```

- Status codes

Satus Code	
200 OK	Indicates a successful response
400 Error	Business account could not be set up, email is taken

GET BIOGRAPHY OF THE BUSINESS BY EMAIL

- Returns the biography of the business
- GET /business/biography/:email
- Status codes

Satus Code	

200 OK	Indicates a successful response
404 Not Found	No business account found

CHANGE BIOGRAPHY OF THE BUSINESS BY EMAIL

- Changes the biography of the given business account
- POST /business/biography/
- Example Body

```
{
        "useremail": “bharath_business@gmail.com”,
        “biography”: “this is bharath’s business”
}
```

- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No business account found

GET PROFILE PICTURE OF THE BUSINESS BY EMAIL

- Returns the profile picture of the business
- GET /business/image/:email
- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No business account found

CHANGE PROFILE PICTURE OF THE BUSINESS BY EMAIL

- Changes the profile picture of the given business account
- POST /business/image/
- Example Body

```
body: {
        "useremail": “bharath_business@gmail.com”,
}
file: <upload a new profile pic>
```

- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No business account found

PROMOTER INVITES

POST A PROMOTER REQUEST

- POST /promoter-invites
- Example Body

```
{
      “inviteeEmail”: invitee_email,
      “promoterEmail”: promoter_email,
      “event”: rand_event_id
  }
```

- Status codes

Satus Code	
202 OK	Indicates a successful response
404 Error	No promoter request was issued
409	Promoter request already exists

GET A LIST OF PROMOTERS INVITES BY INVITEE

- Returns all promoter invites sent to an invitee
- GET /promoter-invites/to/:invitee_id

- Status codes

Satus Code	
200 OK	Indicates a successful response
404 Not Found	No promoters request document found

GET A LIST OF PROMOTERS INVITES BY PROMOTER

- Returns all promoter invites sent by a promoter
- GET /promoter-invites/by/:promoter_id

- Status codes

Satus Code	
200 OK	Indicates a successful response

PATCH ACCEPTED PROMOTER INVITE

- Accept the specified invite by changing its “status” to “accepted”
- PATCH /accept/:_id

- Status codes

Satus Code	
203 OK	Indicates a successful change
400 Error	No request document found

PATCH REJECTED PROMOTER INVITE

- Reject the specified invite by changing its “status” to “rejected”
- PATCH /reject/:_id

- Status codes

Satus Code	
203 OK	Indicates a successful change
400 Error	No request document found

DELET PROMOTER INVITE

- delete the specified invite
- PATCH /delete/:_id

- Status codes

Satus Code	
203 OK	Indicates a successful change
400 Error	No request document found

FACEBOOK LOGIN ACCOUNTS

SIGN UP WITH FACEBOOK

- Creates an account on facebook
- POST auth/facebook/
- Callback URL redirects to /dashboard
- Status codes

Satus Code	
200 OK	Indicates a successful response

GOOGLE LOGIN ACCOUNTS

CHECK IF USER ALREADY HAS AN ACCOUNT

- Checks if an account with the given email already exists.
- If the account exists, generate a session token and return it to front-end
- POST login/google/check/
- Status codes

Satus Code	
200 OK	Indicates a successful response (the token is null if the user does not exist. An account with filled token is returned in case of successful fetch.)