StudyTogether - System Design Document - CSC301

Collaborators : Mohamed Issa, Maor, Daniel Laufer, Milind, John

Table of Contents:

CRC Cards Description

Front-end Modules/Pages:

Class Name: App	
Parent Class: N/A (root class) **Subclasses:** *Everything*	
Responsibilities: ● Responsible for conditionally rendering each page on the website. ● Provides access to the redux store and chakra ui theme to all subclasses.	**Collaborators:** ● src/modules/* ● src/actions/Auth.js ● src/reducers/root

Class Name: LandingPage	
Parent Class: App **Subclasses:** *None*	
Responsibilities: ● The first page that a user sees when loading up StudyTogether ● Displays our logo, description of the app, features, etc.	**Collaborators:** ● src/modules/* ● src/actions/Auth.js ● src/reducers/root

Class Name: Login	
Parent Class: App **Subclasses:** GreenButton	
Responsibilities: ● Handles authenticating users ● On successful login, a JWT token is returned to the client and is stored in a redux store. (optionally saved to localStorage if a user requests to be remembered).	**Collaborators:** ● src/reducers/Auth.js ● src/actions/Auth.js

Class Name: ForgotPassword

Parent Class: App **Subclasses:** None	
Responsibilities: ● Allows user to send their email to the backend to get an email to reset password	**Collaborators:** ● None

Class Name: ResetPassword	
Parent Class: App **Subclasses:** None	
Responsibilities: ● Reach this page after clicking the link sent from ForgotPassword page ● Allows user to change password if the link from the email is used	**Collaborators:** ● None

Class Name: Groups	
Parent Class: App **Subclasses:** SecondGroup, Group	
Responsibilities: ● Displays all the groups to the authenticated user ● It shows all the groups in both the ways for demo	**Collaborators:** ● None

Class Name: EmailSent	
Parent Class: None **Subclasses:** None	
Responsibilities: ● Let's user know that email to change password is sent	**Collaborators:** ● None

Class Name: Register	

Parent Class: None **Subclasses:** GreenButton	
Responsibilities: ● Handles registering users (ie creating new accounts) and subsequently authenticating users on success ● On successful registration, a JWT token is returned to the client and is stored in a redux store. (optionally saved to localStorage if a user requests to be remembered).	**Collaborators:** ● src/reducers/Auth.js ● src/actions/Auth.js

Class Name: GroupCreator	
Parent Class: App **Subclasses:** Map, GreenButton	
Responsibilities: ● Allows users to create new groups by specifying details about it (ex. Whether or not it is recurring, it's title, current attendee count, etc) ● User the Map component to allow users to specify a location for the study group.	**Collaborators:** ● src/components/Map.js

Class Name: AccountInfo	
Parent Class: App **Subclasses:** None	
Responsibilities: ● Allows users to view/edit their account information ● Allows users to view other people's account info while restricting them from updating other's account details ● Updates localStorage when user changes their name ● Displays study groups a user is a part of	**Collaborators:** ● src/reducers/Auth.js ● src/actions/Auth.js

Class Name: NotFoundPage
Parent Class: App **Subclasses:** None

Responsibilities: ● A page that is displayed once the user tries to navigate to a url that our front-end doesn't handle.	**Collaborators:** ● N/A

Front-end Custom Components:

Class Name: Group	
Parent Class: Any (i.e this is intended to be rendered/used by many classes) **Subclasses:** None	
Responsibilities: ● Displays a group as in the Figma design ● On click links to the group page ● Also displays group status	**Collaborators:** ● None

Class Name: GreenButton	
Parent Class: Any (i.e this is intended to be rendered/used by many classes) **Subclasses:** None	
Responsibilities: ● A reusable Chakra-ui button with our custom styling and settings	**Collaborators:** ● None

Class Name: CustomSpinner	
Parent Class: Any (i.e this is intended to be rendered/used by many classes) **Subclasses:** None	
Responsibilities: ● A reusable Chakra-ui spinner with our custom styling and settings ● Displayed in multiple places of our app whenever we make a HTTP request to the back-end.	**Collaborators:** ● None

Class Name: DetailedGroup
Parent Class: Any (i.e this is intended to be rendered/used by many classes) **Subclasses:** None

Responsibilities: ● Displays a group as in the second Figma design ● Display can vary as different sizes are given as props ● On click links to the group page ● Displays the group’s status	**Collaborators:** ● None

Class Name: Map	
Parent Class: Any (i.e this is intended to be rendered/used by many classes) **Subclasses:** None	
Responsibilities: ● An interactive google map ● Users can click on the map to set markers.	**Collaborators:** ● None

Class Name: Navbar	
Parent Class: App **Subclasses:** GreenButton	
Responsibilities: ● A component that contains several clickable items that will navigate the user to its corresponding page in the app.	**Collaborators:** None

BACKEND

Backend - App

Class Name: app.js	
Parent Class: None **Subclasses:** None	
Responsibilities: ● Holds the main configurations : defined routes, mongoDB client, middleware. ● Run and build the express server	**Collaborators:** ● routes/user.js ● routes/forgot.js ● routes/studygroup.js

Backend - Routes

Class Name: routes/user.js	
Parent Class: None **Subclasses:** None	
Responsibilities: ● Handles user authentication and authorization ○ Uses JWT token ● Update non-sensitive user information	**Collaborators:** ● models/user.model.js ● helper/helperUser.js ● app.js

Class Name: routes/studygroup.js	
Parent Class: None **Subclasses:** None	
Responsibilities: ● Handle all CRUD operations for study groups	**Collaborators:** ● models/studygroup.models.js ● helper/helperUser.js ● models/studygroupseries.model.js ● app.js

Class Name: routes/forgot.js	
Parent Class: None **Subclasses:** None	
Responsibilities: ● Recover password in case user forgot it (i.e update old password to a new one) ○ Recover process involves tokenization to add a layer of security ○ Sends reset link to the user's email	**Collaborators:** ● models/user.model.js ● models/token.model.js ● app.js

Backend - helpers

Class Name: helpers/helperUser.js	
Parent Class: None **Subclasses:** None	
Responsibilities: ● Encapsulates all the helper methods related to the user model. ○ verifyToken() ○ respondJWT()	**Collaborators:** ● routes/user.js ● routes/studygroup.js

Backend - ODM

Class Name: models/user.model.js	
Parent Class: None **Subclasses:** None	
Responsibilities: ● A mongoose document model (ODM) that is used to communicate with the users collections.	**Collaborators:** ● routes/user.js ● routes/studygroup.js ● routes/forgot.js

Class Name: models/studygroup.model.js	
Parent Class: None **Subclasses:** None	
Responsibilities: ● A mongoose document model (ODM) that is used to communicate with the study group collections.	**Collaborators:** ● routes/studygroup.js

Class Name: models/token.model.js	
Parent Class: None **Subclasses:** None	
Responsibilities: ● A mongoose document model (ODM) that is used to communicate with the tokens collections.	**Collaborators:** ● routes/forgot.js

Class Name: models/studygroupseries.model.js	
Parent Class: None **Subclasses:** None	
Responsibilities: ● A mongoose document model (ODM) that is used to communicate with the Study-group series collection	**Collaborators:** ● routes/studygroup.js

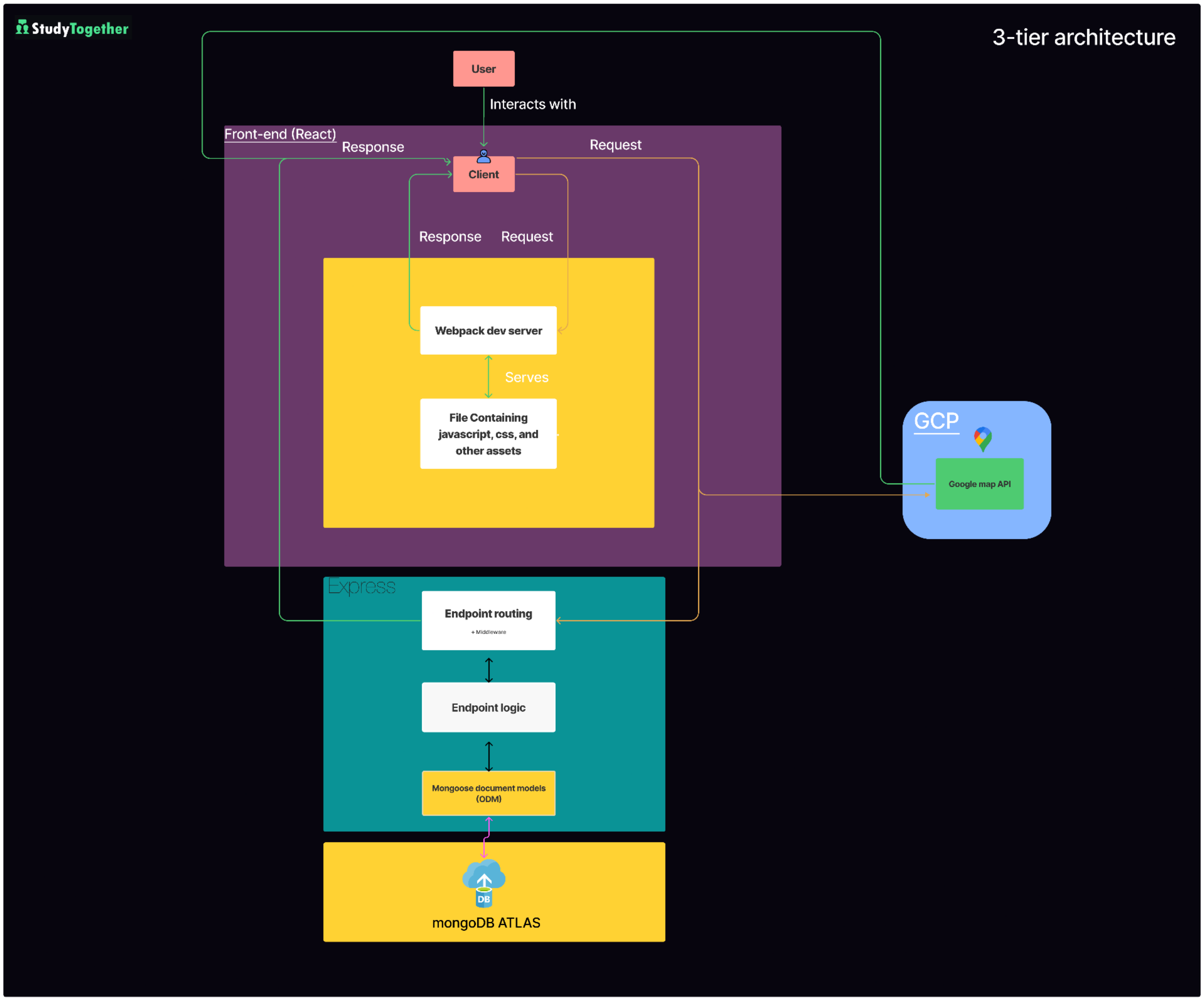
System decomposition

When running the app locally *(we will update this document when we setup a permanent hosting solution for the app)*, when the user makes a request to view the web app, the webpack dev server bundles all of the javascript, css, and other assets into one file and sends it to the user. Once the user receives this content, the React code will dynamically render content in the client's browser. As the user interacts with the app, many http requests will need to be sent to the back-end. This is done through axios, a javascript library made for sending http requests. Additionally, the front-end uses the Google Maps API for integrating interactive map components in the app. We use a javascript library called *react-google-maps/ap*i that handles making requests to google's servers, all we need to do is provide it an API key. The API key for this service is found and configured in the StudyTogether project Google Cloud Platform.

The express framework will handle routing this request to the appropriate class and execute the defined operations, there operations could include user authentication and authorization, create a new study-group, reset password, etc. We also have custom middleware to ensure certain checks are made before the request is processed. We currently have a middleware that verifies JWT tokens embedded in incoming requests. For any operation that involves DB communication, it will use an ODM such as models/user.model.js to update stored data as desired. One thing to note is that we are not using a local mongoDB server, but instead a cloud instance through mongoDB Atlas.

In the case where user input is invalid, we incorporate both frontend and backend validation that will ensure no operation is executed with uncertain data, we also log the error. Front-end validation will give the user real time feedback on the issue but making sure not to send anything to the backend until validated. If a request with invalid data does make it to the backend, we make sure to include a validation error handler on every endpoint that will stop executing the request and return the appropriate status code and error message. For any other reason that the request to the back-end may fail (invalid credentials, network failure, etc), the front-end handles it and provides the user with informative feedback on how to resolve the issue they are facing.

We used a three-tier system architecture described here: https://www.linuxjournal.com/article/3508


StudyTogether
3-tier architecture
User
Interacts with
Front-end (React)
Response
Request
Client
Response
Request
Webpack dev server
Serves
File Containing javascript, css, and other assets
GCP
Google map API
Express
Endpoint routing
+ Middleware
Endpoint logic
Mongoose document models (ODM)
DB
mongoDB ATLAS