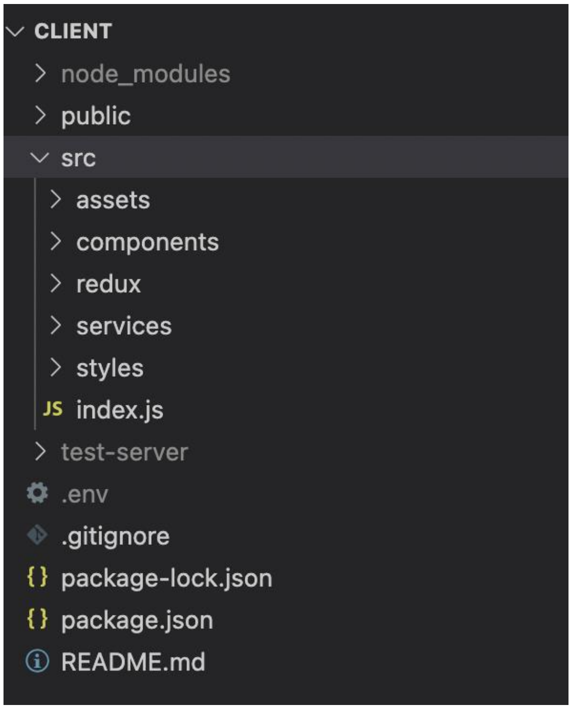
Goally Documentation

Frontend Technical Document

Tech Stack used in the app include:

React.js, JavaScript, SASS, Redux-Toolkit

Directory Structure



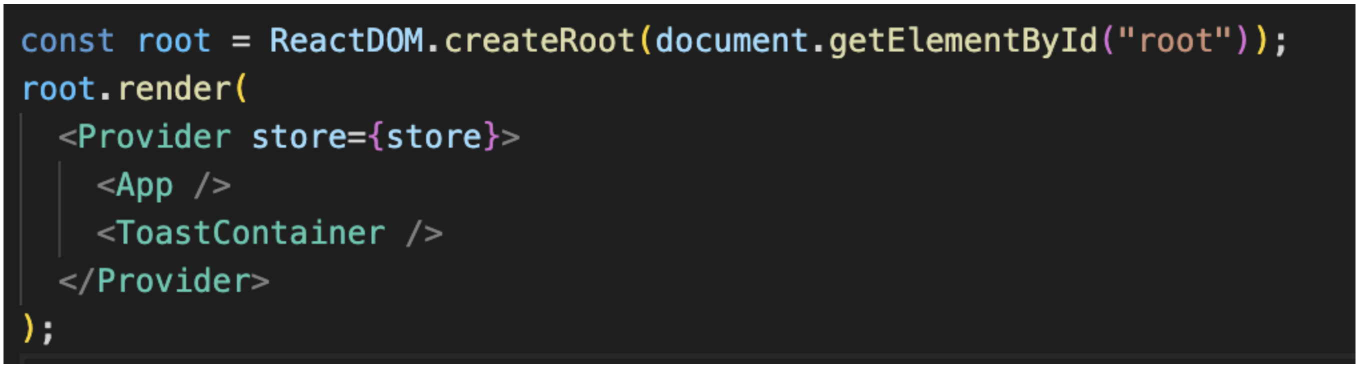
```
∨ CLIENT
  > node_modules
  > public
  ∨ src
    > assets
    > components
    > redux
    > services
    > styles
    JS index.js
  > test-server
  .env
  .gitignore
  {} package-lock.json
  {} package.json
  ⓘ README.md
```

All the source files lie inside the /src folder.
Within /src:

a. /assets contains all the images and icons.
b. /components contains all the web components made in react being used throughout the webapp.
c. /redux contains the codefiles including /slices and store.js to enable app level state management in the entire application.
d. /services contain the service logic and api specifications for the entire app. Put simply, they contain all the API call methods.
e. /styles contain the entire SASS files used inside the application. To note, no styles files were produced inside of the components. It’s a design choice we went forward with keeping all the styles in one place.
f. Index.js marks the main entry point of the entire application. Also mentioned in the package.json.
g. Package.json lists all the packages (third party) being used in the entire application.

Entry Point

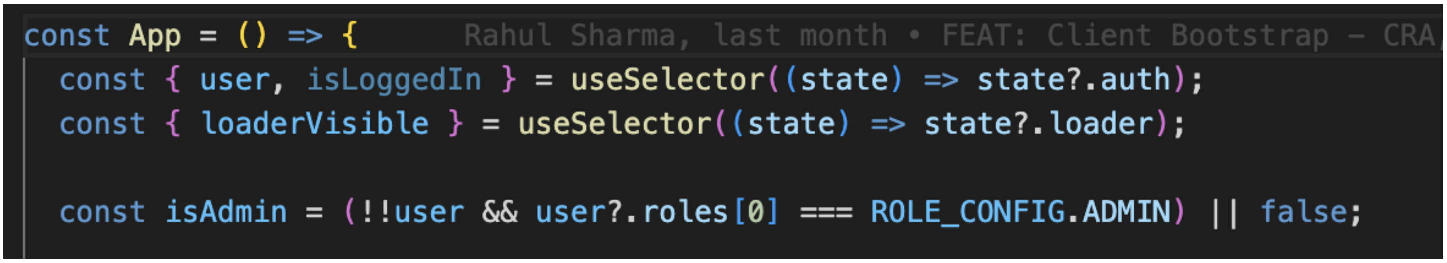
The main entry point for the app is /src/index.js



```
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <Provider store={store}>
    <App />
    <ToastContainer />
  </Provider>
);
```

Inside this file, we have enabled an app level state management using Redux store. Also notice the <ToastContainer /> catering to all the toasts (alert messages) being triggered throughout the application at different times.

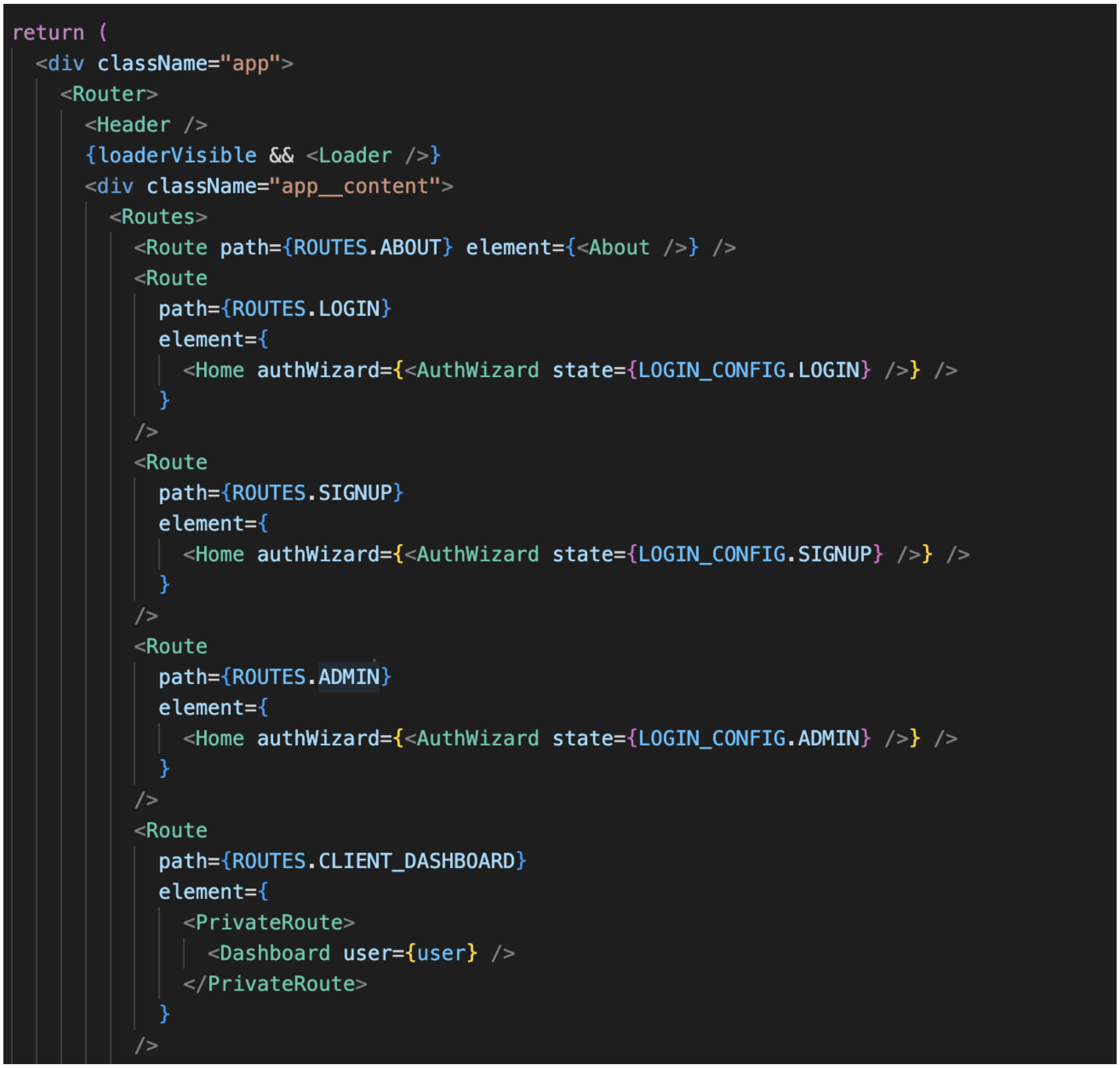
Within app.js



```
const App = () => {      Rahul Sharma, last month • FEAT: Client Bootstrap - CRA,
  const { user, isLoggedIn } = useSelector((state) => state?.auth);
  const { loaderVisible } = useSelector((state) => state?.loader);

  const isAdmin = (!!user && user?.roles[0] === ROLE_CONFIG.ADMIN) || false;
```

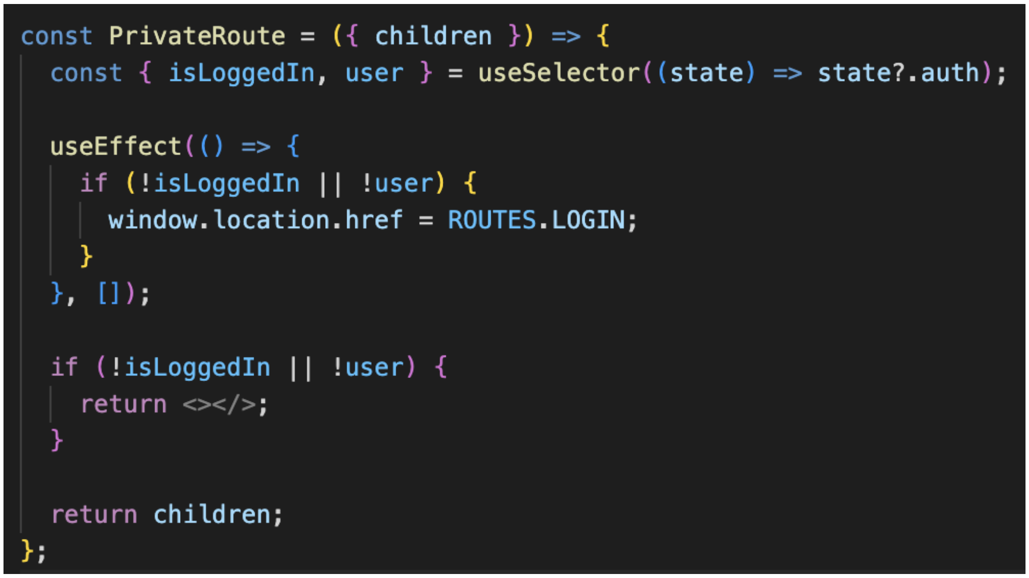
A quick look inside app.js shows we fetch user from the central store and then check if the user is admin or not. isAdmin flag is used to enable the admin level privileges across the application.

Note, the user object that we fetch from the central store is initially {} and is later populated upon login. (check authSlice.js)



```
return (
  <div className="app">
    <Router>
      <Header />
      {loaderVisible && <Loader />}
      <div className="app__content">
        <Routes>
          <Route path={ROUTES.ABOUT} element={<About />} />
          <Route
            path={ROUTES.LOGIN}
            element={
              <Home authWizard={<AuthWizard state={LOGIN_CONFIG.LOGIN} />} />
            }
          />
          <Route
            path={ROUTES.SIGNUP}
            element={
              <Home authWizard={<AuthWizard state={LOGIN_CONFIG.SIGNUP} />} />
            }
          />
          <Route
            path={ROUTES.ADMIN}
            element={
              <Home authWizard={<AuthWizard state={LOGIN_CONFIG.ADMIN} />} />
            }
          />
          <Route
            path={ROUTES.CLIENT_DASHBOARD}
            element={
              <PrivateRoute>
                <Dashboard user={user} />
              </PrivateRoute>
            }
          />
```

Within app, we have configured the router layer including both public and private routes. Public routes can be accessed by the user who is not logged it whereas the private route components like Dashboard and Survey Forms will not be accessible by the user if he is not logged in.

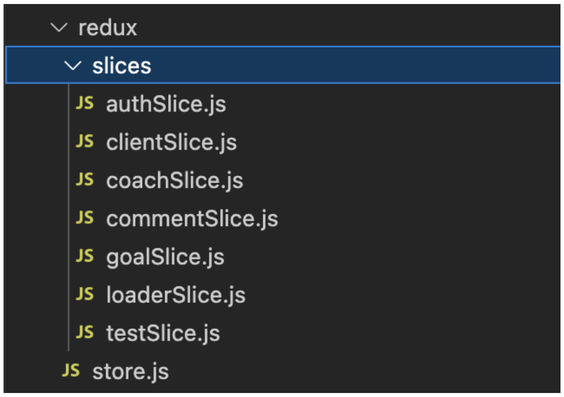


```
const PrivateRoute = ({ children }) => {
  const { isLoggedIn, user } = useSelector((state) => state?.auth);

  useEffect(() => {
    if (!isLoggedIn || !user) {
      window.location.href = ROUTES.LOGIN;
    }
  }, []);

  if (!isLoggedIn || !user) {
    return <></>;
  }

  return children;
};
```

PrivateRoute component as you can see is just an HOC that returns <></> (nothing!) when the user is not logged in or the user object itself is null. Note, the isLoggedIn flag will also be populated when the user signs in on the platform (check authSlice.js).

If the user object exists on the app level state (redux) and the user is logged in then we simply return the component that was passed to the <PrivateRoute /> component.

Data flow through Redux

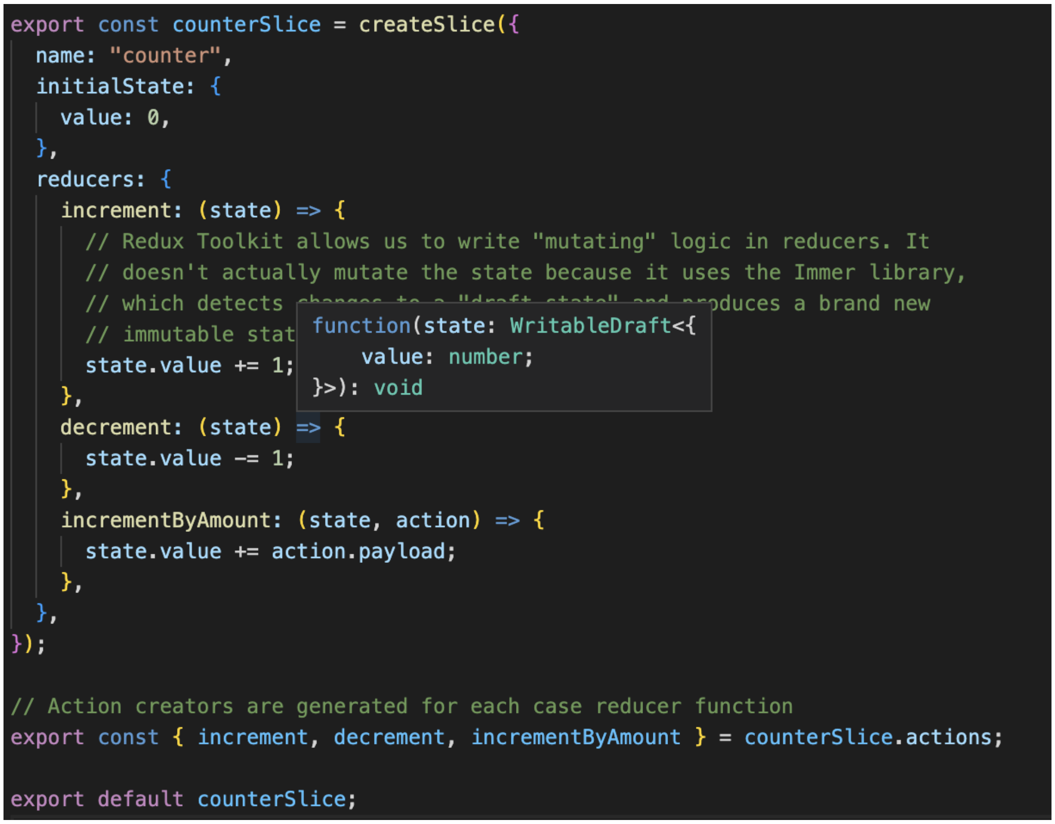


```
redux
  slices
    authSlice.js
    clientSlice.js
    coachSlice.js
    commentSlice.js
    goalSlice.js
    loaderSlice.js
    testSlice.js
  store.js
```

Within redux, we have slices and store. Slices contain the (reducer, state, and actions), all encapsulated inside them. Also, it enables calling APIs within an action creator as they create an async thunk for the same.

Test slice code walkthrough

We have a testSlice setup which can be used as a template for any new slice we create.



```
export const counterSlice = createSlice({
  name: "counter",
  initialState: {
    value: 0,
  },
  reducers: {
    increment: (state) => {
      // Redux Toolkit allows us to write "mutating" logic in reducers. It
      // doesn't actually mutate the state because it uses the Immer library,
      // which detects changes to a "draft state" and produces a brand new
      // immutable stat
      state.value += 1;
    },
    decrement: (state) => {
      state.value -= 1;
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload;
    },
  },
});

// Action creators are generated for each case reducer function
export const { increment, decrement, incrementByAmount } = counterSlice.actions;

export default counterSlice;
```

```
function(state: WritableDraft<{
    value: number;
}>): void
```

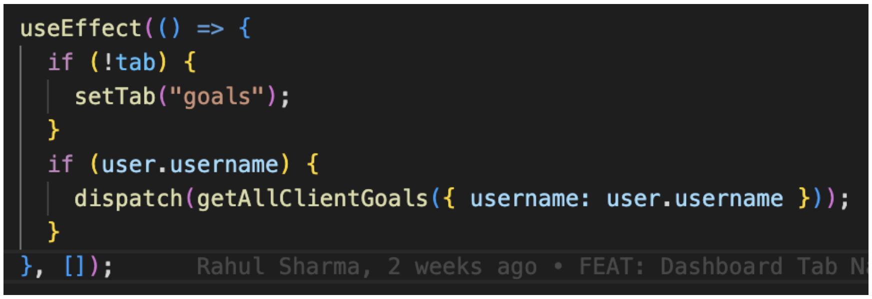
Within the test slice, we have a state entity, named counter whose initial value is set to 0. Furthermore, we have some reducers like increment and decrement which changes the state counter’s value.

This reducers are also exported as actions from the slice itself which then can be called/dispatched using the useDispatch hook provided by react-redux.

API Request Flow

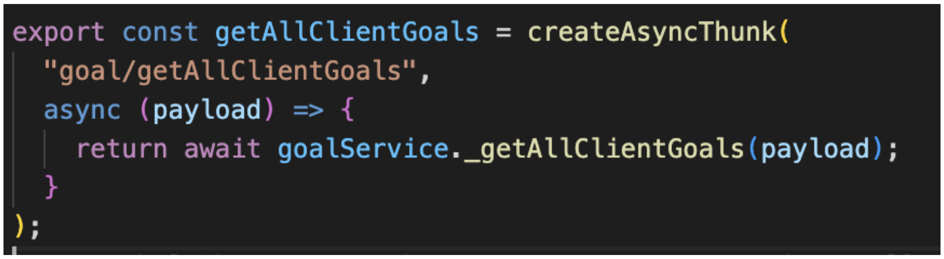
A flow where we fetch all the goals of a client. This walkthrough aims to enable the developer to understand how we use the useDispatch, slices, actions and reducers combined in action to fetch all the goals associated with a client.

Within src/components/dashboard/index.js, on initial component mount, we dispatch (like explained before) an action, getAllClientGoals() and pass the client's username as payload.



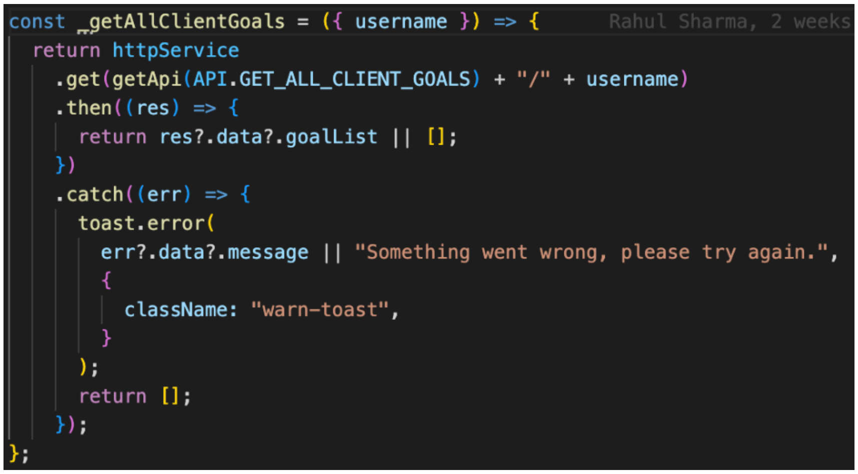
```
useEffect(() => {
  if (!tab) {
    setTab("goals");
  }
  if (user.username) {
    dispatch(getAllClientGoals({ username: user.username }));
  }
}, []);      Rahul Sharma, 2 weeks ago • FEAT: Dashboard Tab N
```

This method is nothing but an action creator method inside goalSlice.js



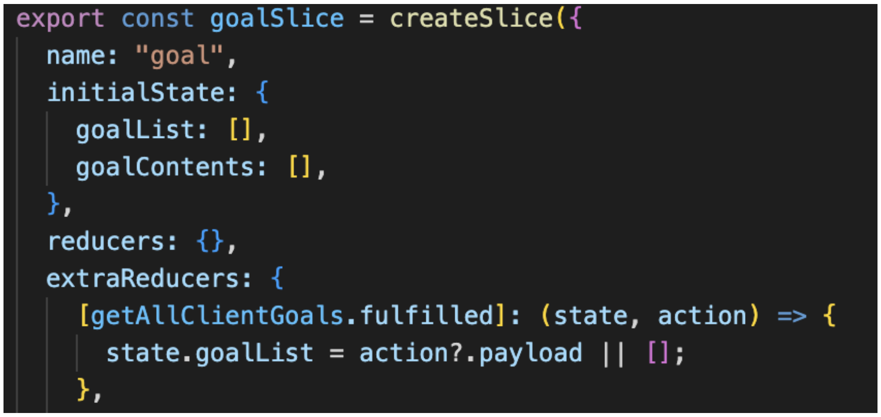
```
export const getAllClientGoals = createAsyncThunk(
  "goal/getAllClientGoals",
  async (payload) => {
    return await goalService._getAllClientGoals(payload);
  }
);
```

This calls the _getAllClientGoals(payload) method of the goalService.js – This method inside the goalService.js is the exact place where the API call lies.



```
const _getAllClientGoals = ({ username }) => {      Rahul Sharma, 2 weeks
  return httpService
    .get(getApi(API.GET_ALL_CLIENT_GOALS) + "/" + username)
    .then((res) => {
      return res?.data?.goalList || [];
    })
    .catch((err) => {
      toast.error(
        err?.data?.message || "Something went wrong, please try again.",
        {
          className: "warn-toast",
        }
      );
      return [];
    });
};
```

Once the API call succeeds, the res?.data?.goalList [] array returned from the backend is then passed to the reducer of the goal slice which updates the clientGoals [] key inside the central/app level goal state for the client (using Redux store).

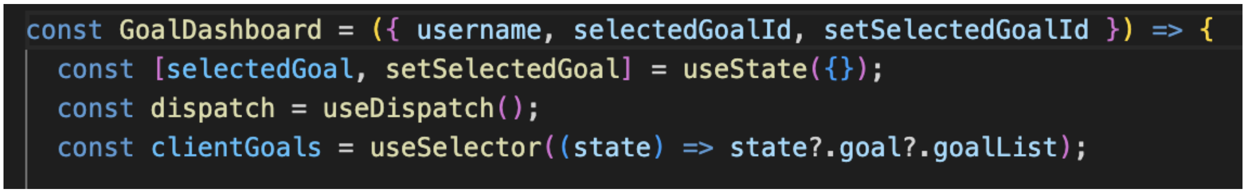


```
export const goalSlice = createSlice({
  name: "goal",
  initialState: {
    goalList: [],
    goalContents: [],
  },
  reducers: {},
  extraReducers: {
    [getAllClientGoals.fulfilled]: (state, action) => {
      state.goalList = action?.payload || [];
    },
```

To note, these methods which call the API service fall under the extraReducers part of the goalSlice and their completion or success is identified by the key - *.fulfilled – This is a function which expects the state and action as its arguments. The action.payload will contain the same list that we returned from the goalService's method.*

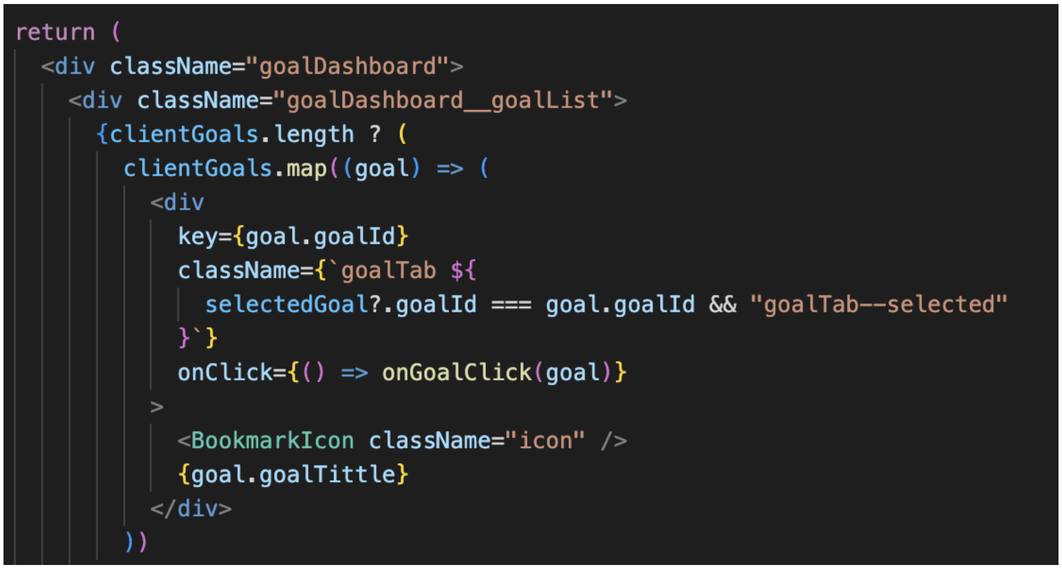
Now when the central/app state level goal list populates. It can be easily accessed anywhere across the app using the hook useSelector provided by react-redux.

For e.g. inside GoalDashboard component, see how we select the goalList into the clientGoals.



```
const GoalDashboard = ({ username, selectedGoalId, setSelectedGoalId }) => {
  const [selectedGoal, setSelectedGoal] = useState({});
  const dispatch = useDispatch();
  const clientGoals = useSelector((state) => state?.goal?.goalList);
```

Now once the clientGoals are updated from the redux, it triggers an update on the GoalDashboard component, which calls its render method and updates the view where clientGoals is being rendered.



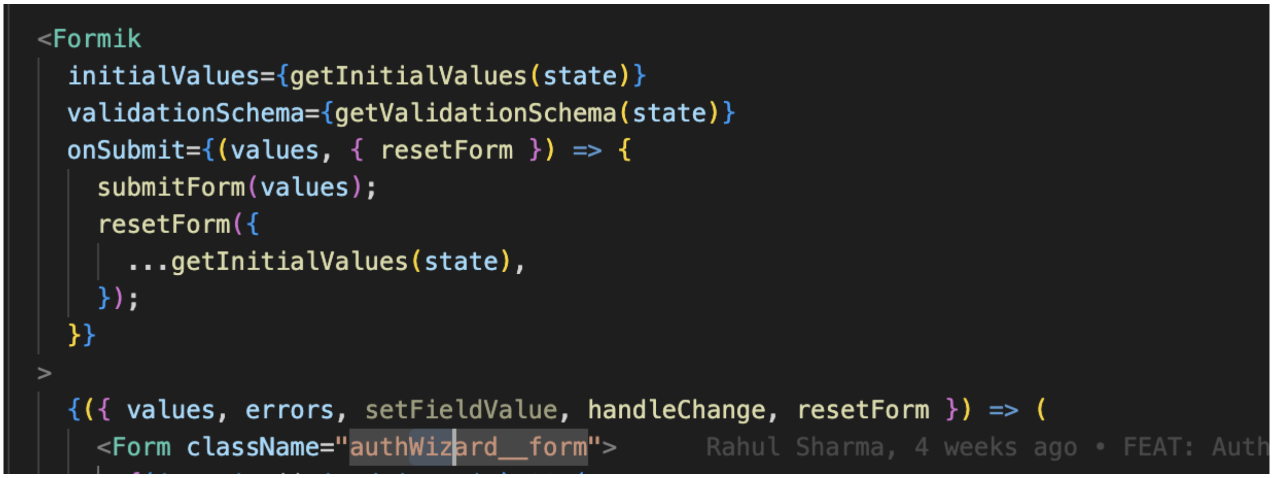
```
return (
  <div className="goalDashboard">
    <div className="goalDashboard__goalList">
      {clientGoals.length ? (
        clientGoals.map((goal) => (
          <div
            key={goal.goalId}
            className={`goalTab ${
              selectedGoal?.goalId === goal.goalId && "goalTab--selected"
            }`}
            onClick={() => onGoalClick(goal)}
          >
            <BookmarkIcon className="icon" />
            {goal.goalTittle}
          </div>
        ))
```

With this update, all the client goals are made visible on the goal dashboard component.

Authentication Flow

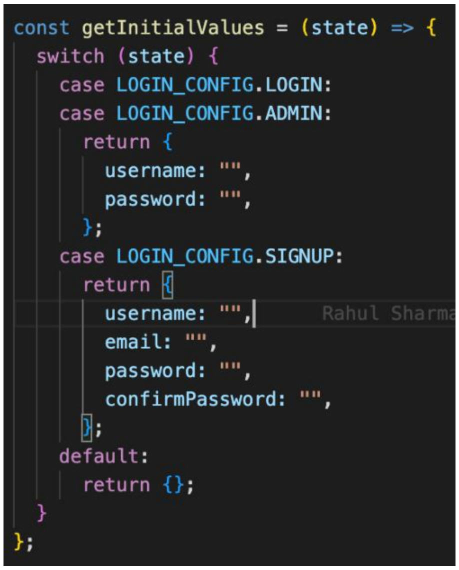
Initially when the user lands on the platform, he sees the landing page with the login and signup wizard, we call it the AuthWizard on the component level.

AuthWizard is a generic component which uses Formik to enable form handling including validation and field handling.



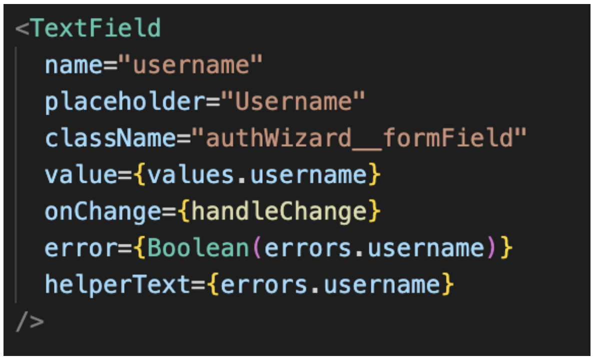
```
<Formik
  initialValues={getInitialValues(state)}
  validationSchema={getValidationSchema(state)}
  onSubmit={(values, { resetForm }) => {
    submitForm(values);
    resetForm({
      ...getInitialValues(state),
    });
  }}
>
  {({ values, errors, setFieldValue, handleChange, resetForm }) => (
    <Form className="authWizard__form">      Rahul Sharma, 4 weeks ago • FEAT: Auth
```

If we check the render method of the AuthWizard application, we see that the Formik expects some initial state and a validation schema. Now since we have 3 users (client, coach and admin) on the platform and one AuthWizard for all of them to enable them to have access to the portal, we provide custom init values and validation schema based on which user it is and what action he is performing (state).



```
const getInitialValues = (state) => {
  switch (state) {
    case LOGIN_CONFIG.LOGIN:
    case LOGIN_CONFIG.ADMIN:
      return {
        username: "",
        password: "",
      };
    case LOGIN_CONFIG.SIGNUP:
      return {
        username: "",      Rahul Sharma
        email: "",
        password: "",
        confirmPassword: "",
      };
    default:
      return {};
  }
};
```

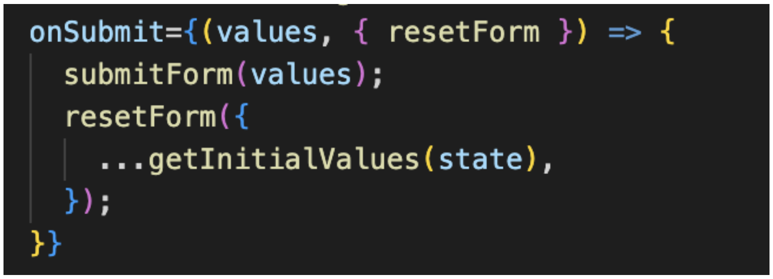
For e.g., it the user is a client/coach who is trying to login into the portal, we return an object with only username and password.



```
<TextField
  name="username"
  placeholder="Username"
  className="authWizard__formField"
  value={values.username}
  onChange={handleChange}
  error={Boolean(errors.username)}
  helperText={errors.username}
/>
```

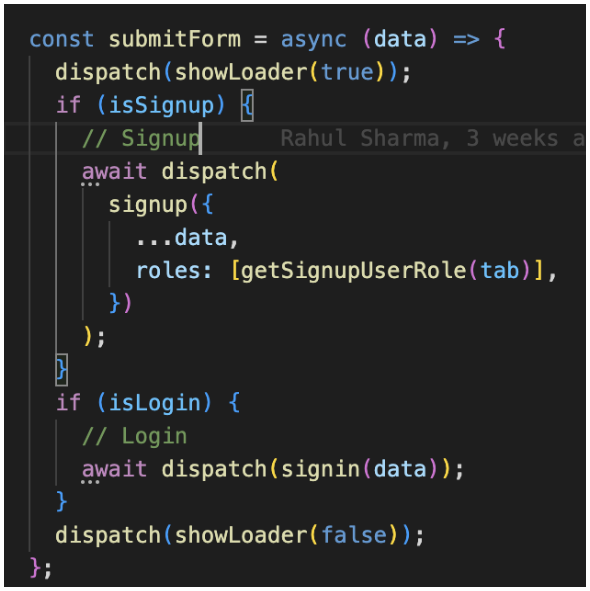
Within Formik, we have used <TextField /> component for different fields in the login/signup form. A quick look in the code above shows the initial value for the username is mapped to the username key in the values object that is inputted into Formik (which initially has our INIT_VALUES).

For each field in the forms, we use in Formik, (also all other forms), we have a handleChange handler which updates the value of the field into the values object.



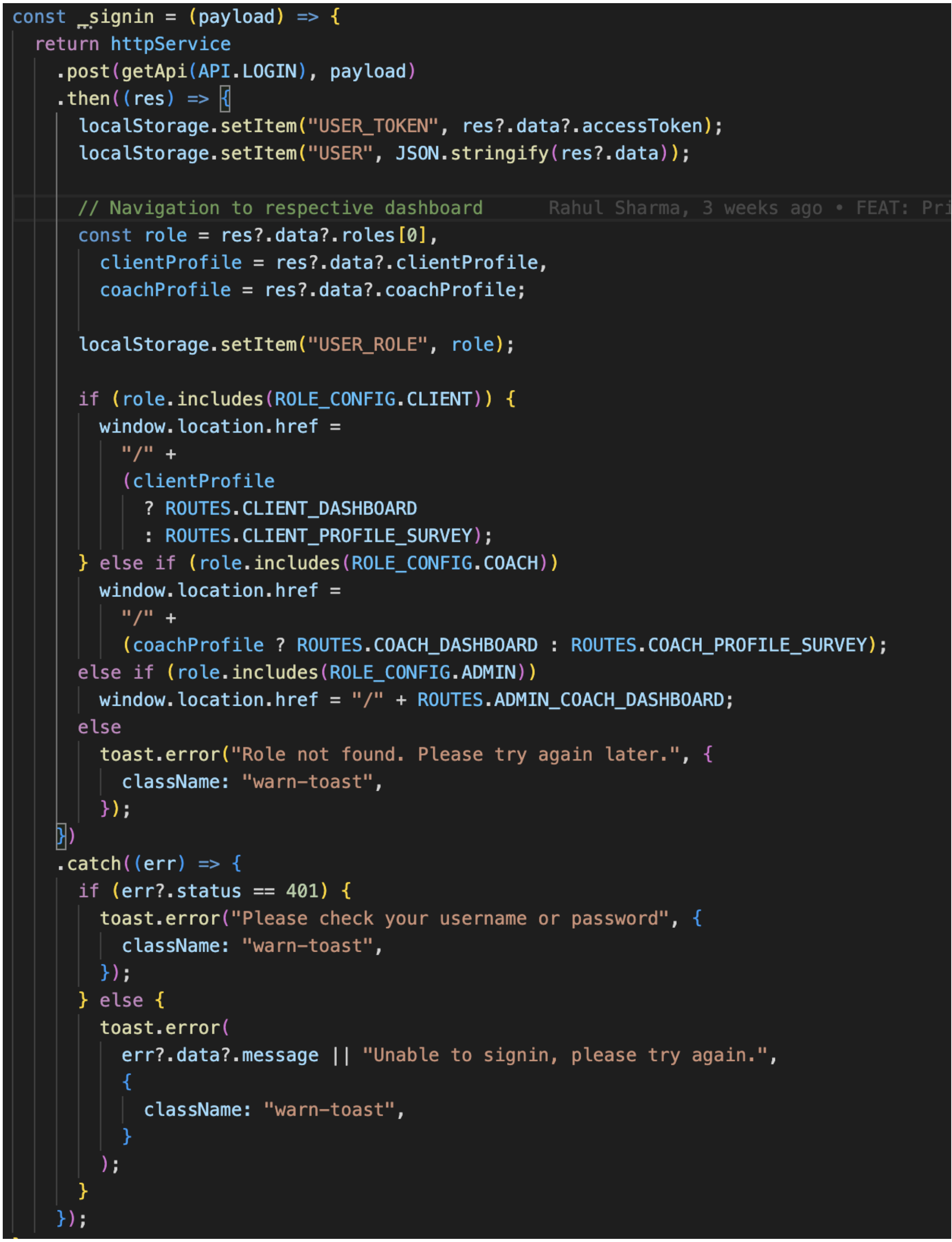
```
onSubmit={(values, { resetForm }) => {
  submitForm(values);
  resetForm({
    ...getInitialValues(state),
  });
}}
```

On a successful form submit, we call the submitForm method with our values object and then reset the form to its initial state for a seamless user experience.



```
const submitForm = async (data) => {
  dispatch(showLoader(true));
  if (isSignup) {
    // Signup        Rahul Sharma, 3 weeks a
    await dispatch(
      signup({
        ...data,
        roles: [getSignupUserRole(tab)],
      })
    );
  }
  if (isLogin) {
    // Login
    await dispatch(signin(data));
  }
  dispatch(showLoader(false));
};
```

Within the submit form method, we display the loader and based on the state being received in the AuthWizard we either dispatch the signup or signin action along with the values payload and appropriate role (in case of signup).



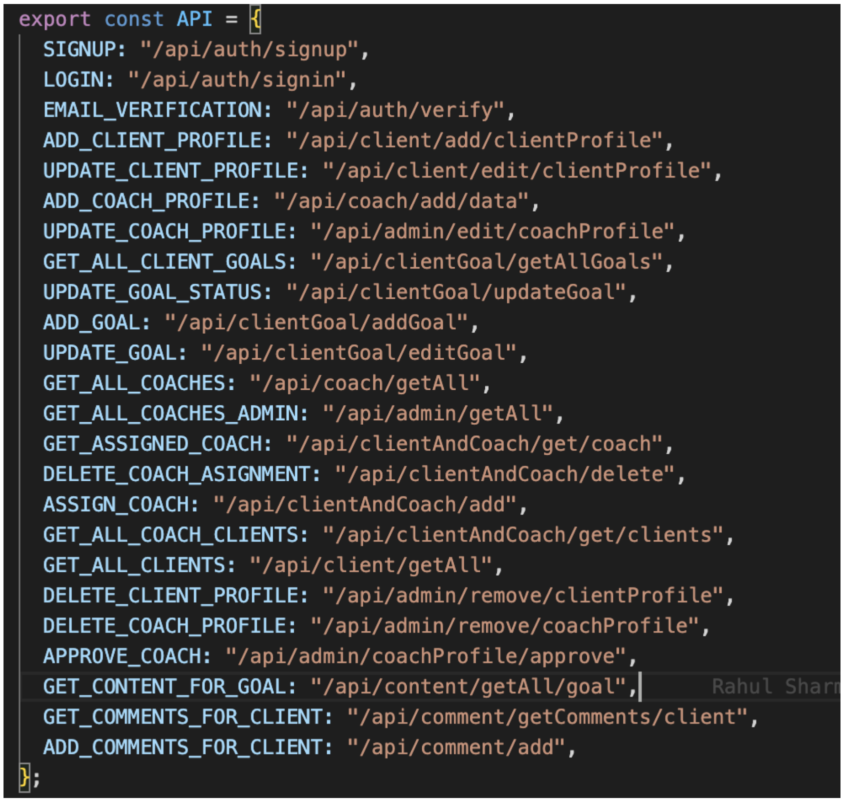
```
const _signin = (payload) => {
  return httpService
    .post(getApi(API.LOGIN), payload)
    .then((res) => {
      localStorage.setItem("USER_TOKEN", res?.data?.accessToken);
      localStorage.setItem("USER", JSON.stringify(res?.data));

      // Navigation to respective dashboard      Rahul Sharma, 3 weeks ago • FEAT: Pri
      const role = res?.data?.roles[0],
        clientProfile = res?.data?.clientProfile,
        coachProfile = res?.data?.coachProfile;

      localStorage.setItem("USER_ROLE", role);

      if (role.includes(ROLE_CONFIG.CLIENT)) {
        window.location.href =
          "/" +
          (clientProfile
            ? ROUTES.CLIENT_DASHBOARD
            : ROUTES.CLIENT_PROFILE_SURVEY);
      } else if (role.includes(ROLE_CONFIG.COACH))
        window.location.href =
          "/" +
          (coachProfile ? ROUTES.COACH_DASHBOARD : ROUTES.COACH_PROFILE_SURVEY);
      else if (role.includes(ROLE_CONFIG.ADMIN))
        window.location.href = "/" + ROUTES.ADMIN_COACH_DASHBOARD;
      else
        toast.error("Role not found. Please try again later.", {
          className: "warn-toast",
        });
    })
    .catch((err) => {
      if (err?.status == 401) {
        toast.error("Please check your username or password", {
          className: "warn-toast",
        });
      } else {
        toast.error(
          err?.data?.message || "Unable to signin, please try again.",
          {
            className: "warn-toast",
          }
        );
      }
    });
```

,

Upon signin, within the authService.js, the user is navigated to different routes based on his/her role in the application. Also, we save the user object within the local storage of the browser as well.

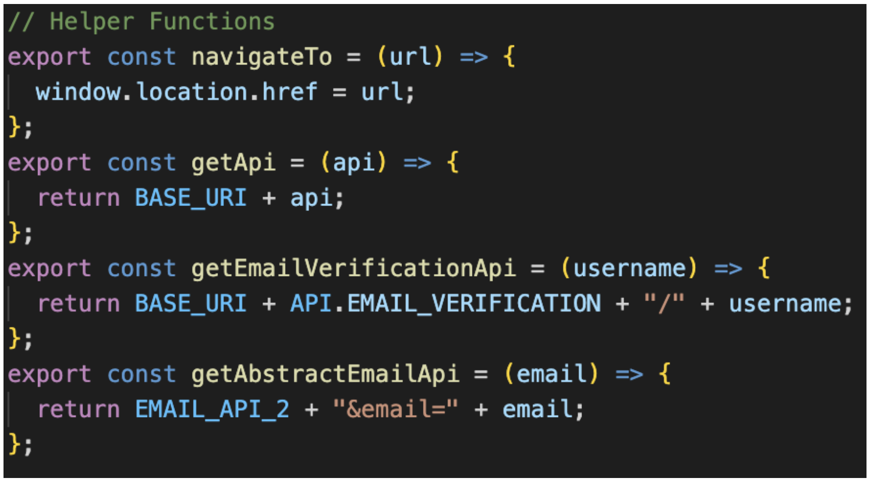
HTTP Service / Axios Request-Response Interceptors

All the service/API related stuff lies inside the /src/services directory. The apiConfig.js file houses all the APIs currently being used in the application.



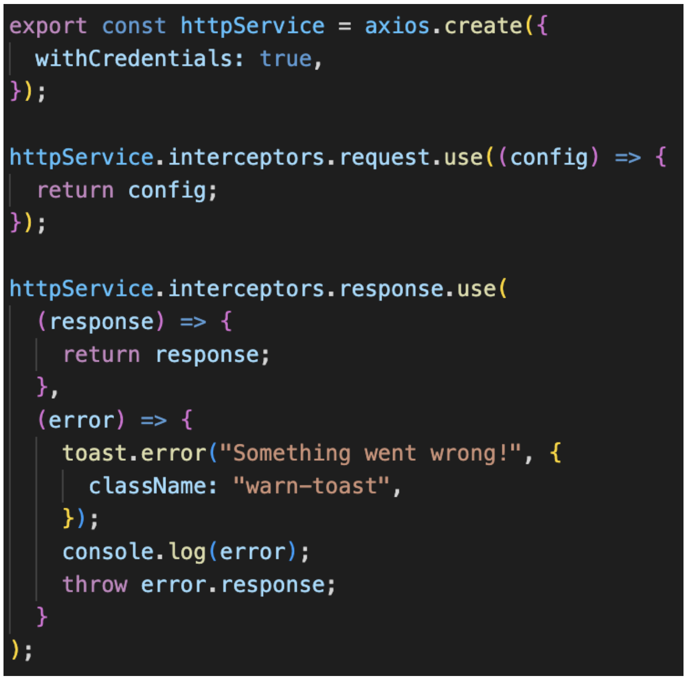
```
export const API = {
  SIGNUP: "/api/auth/signup",
  LOGIN: "/api/auth/signin",
  EMAIL_VERIFICATION: "/api/auth/verify",
  ADD_CLIENT_PROFILE: "/api/client/add/clientProfile",
  UPDATE_CLIENT_PROFILE: "/api/client/edit/clientProfile",
  ADD_COACH_PROFILE: "/api/coach/add/data",
  UPDATE_COACH_PROFILE: "/api/admin/edit/coachProfile",
  GET_ALL_CLIENT_GOALS: "/api/clientGoal/getAllGoals",
  UPDATE_GOAL_STATUS: "/api/clientGoal/updateGoal",
  ADD_GOAL: "/api/clientGoal/addGoal",
  UPDATE_GOAL: "/api/clientGoal/editGoal",
  GET_ALL_COACHES: "/api/coach/getAll",
  GET_ALL_COACHES_ADMIN: "/api/admin/getAll",
  GET_ASSIGNED_COACH: "/api/clientAndCoach/get/coach",
  DELETE_COACH_ASIGNMENT: "/api/clientAndCoach/delete",
  ASSIGN_COACH: "/api/clientAndCoach/add",
  GET_ALL_COACH_CLIENTS: "/api/clientAndCoach/get/clients",
  GET_ALL_CLIENTS: "/api/client/getAll",
  DELETE_CLIENT_PROFILE: "/api/admin/remove/clientProfile",
  DELETE_COACH_PROFILE: "/api/admin/remove/coachProfile",
  APPROVE_COACH: "/api/admin/coachProfile/approve",
  GET_CONTENT_FOR_GOAL: "/api/content/getAll/goal",      Rahul Sharm
  GET_COMMENTS_FOR_CLIENT: "/api/comment/getComments/client",
  ADD_COMMENTS_FOR_CLIENT: "/api/comment/add",
};
```

It also includes some helper functions to return the exact endpoint for the API.



```
// Helper Functions
export const navigateTo = (url) => {
  window.location.href = url;
};
export const getApi = (api) => {
  return BASE_URI + api;
};
export const getEmailVerificationApi = (username) => {
  return BASE_URI + API.EMAIL_VERIFICATION + "/" + username;
};
export const getAbstractEmailApi = (email) => {
  return EMAIL_API_2 + "&email=" + email;
};
```

The httpService.js, exports the httpService object which is an axios instance and contains the interceptors for both the API request and response. Any common code, for e.g. common API headers, must be placed inside the request/response interceptors.



```
export const httpService = axios.create({
  withCredentials: true,
});

httpService.interceptors.request.use((config) => {
  return config;
});

httpService.interceptors.response.use(
  (response) => {
    return response;
  },
  (error) => {
    toast.error("Something went wrong!", {
      className: "warn-toast",
    });
    console.log(error);
    throw error.response;
  }
);
```

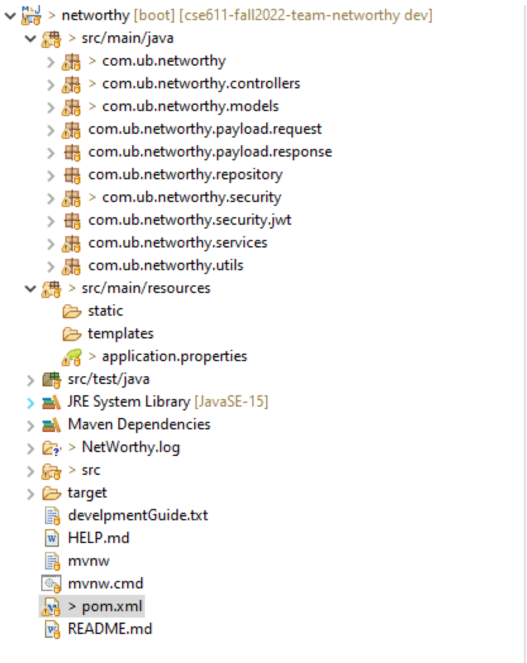
Backend Technical Document

Tech Stack

1. Spring-Boot 2.0
    a. Spring Security
    b. JWT (0.9.1)
    c. Spring Mail (2.5.6)
    d. SpringFox – SwaggerUI (3.0.0)
2. MongoDB (6.0.3)

SpringBoot

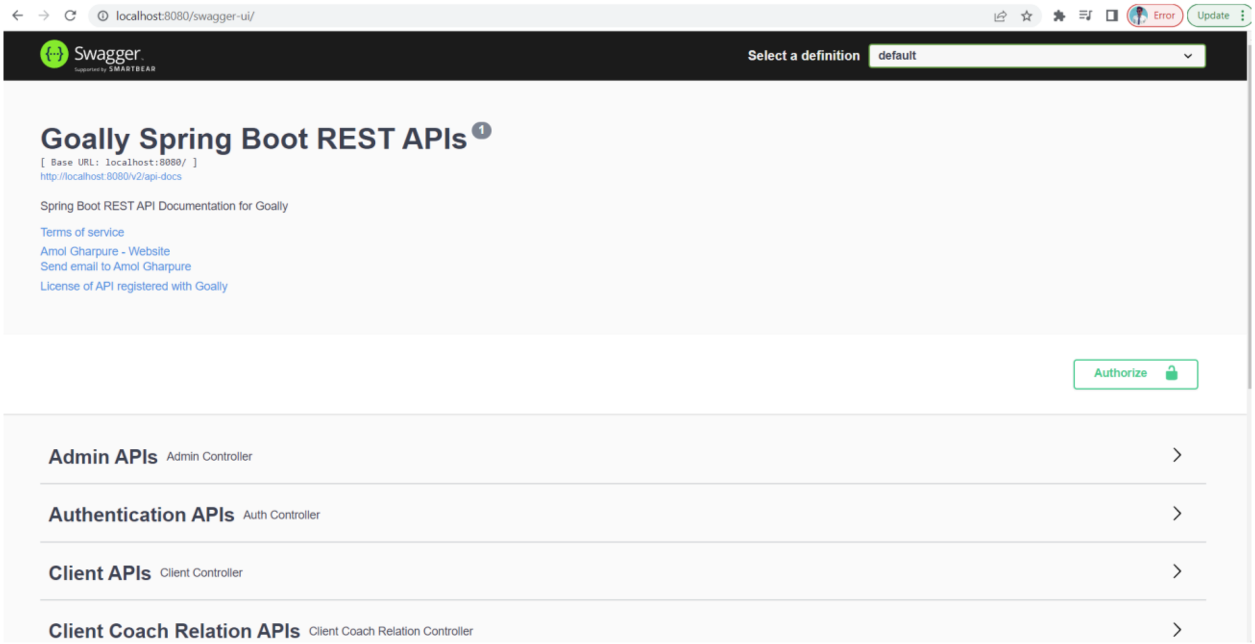
1. Folder Structure
    a. Model – Pojos for entities
    b. Controller – REST controller for API access
    c. Services – service classes for different functional controllers
    d. Repository – Mongo Repositories for entities
    e. Security – Spring security code
    f. Request – custom request pojos
    g. Response – custom response pojos
    h. Utils – common util classes



```
> networthy [boot] [cse611-fall2022-team-networthy dev]
  > src/main/java
    > com.ub.networthy
    > com.ub.networthy.controllers
    > com.ub.networthy.models
    com.ub.networthy.payload.request
    com.ub.networthy.payload.response
    com.ub.networthy.repository
    > com.ub.networthy.security
    com.ub.networthy.security.jwt
    com.ub.networthy.services
    com.ub.networthy.utils
  > src/main/resources
    static
    templates
    > application.properties
  src/test/java
  JRE System Library [JavaSE-15]
  Maven Dependencies
  > NetWorthy.log
  > src
  target
  develpmentGuide.txt
  HELP.md
  mvnw
  mvnw.cmd
  > pom.xml
  README.md
```

2. API Documentation

    To make API documentation simple and central we have used **Swagger.**
    You can access the swagger resource using the below urls
    a. Launching spring app from local machine :- http://localhost:8080/swagger-ui/
    b. From AWS EC2 instance :- http://{ec2-address}:8080/swagger-ui/



localhost:8080/swagger-ui/

Swagger

Select a definition: default

Goally Spring Boot REST APIs

[Base URL: localhost:8080/]

http://localhost:8080/v2/api-docs

Spring Boot REST API Documentation for Goally

Terms of service

Amol Gharpure - Website

Send email to Amol Gharpure

License of API registered with Goally

Authorize

Admin APIs Admin Controller

Authentication APIs Auth Controller

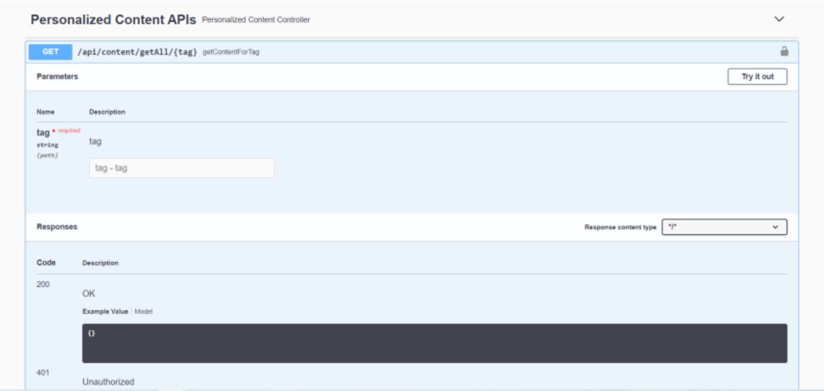
Client APIs Client Controller

Client Coach Relation APIs Client Coach Relation Controller

3. Swagger Testing

   To use swagger follow the below steps

   a. This is the API definition :-



Personalized Content APIs Personalized Content Controller

GET /api/content/getAll/{tag} getContentForTag

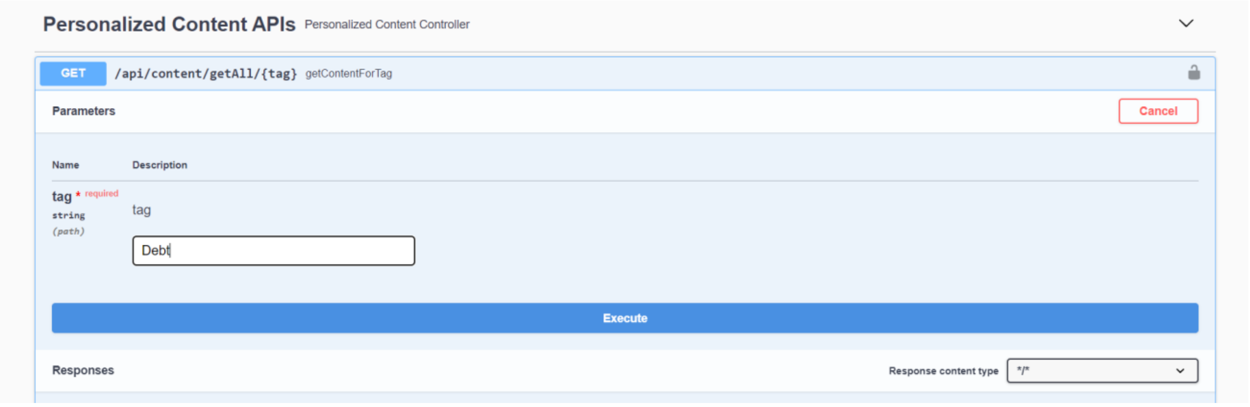
Parameters — Try it out

Name	Description
tag * required string (path)	tag tag - tag

Responses — Response content type: */*

Code	Description
200	OK Example Value | Model {}
401	Unauthorized

   b. Click on “Try it Out” and input your value in the field:-



Personalized Content APIs Personalized Content Controller

GET /api/content/getAll/{tag} getContentForTag

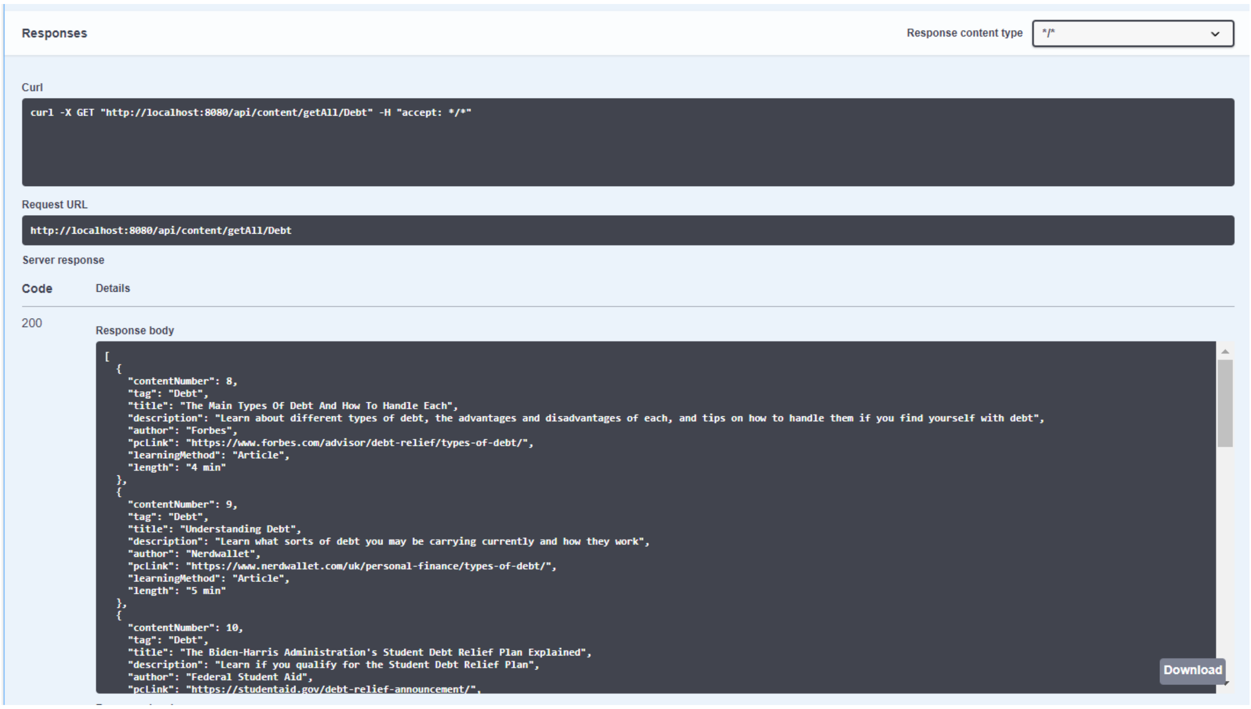
Parameters — Cancel

Name	Description
tag * required string (path)	tag Debt

Execute

Responses — Response content type: */*

   c. Click on execute to execute the API. You can see the expected output, or you will get the appropriate error: -



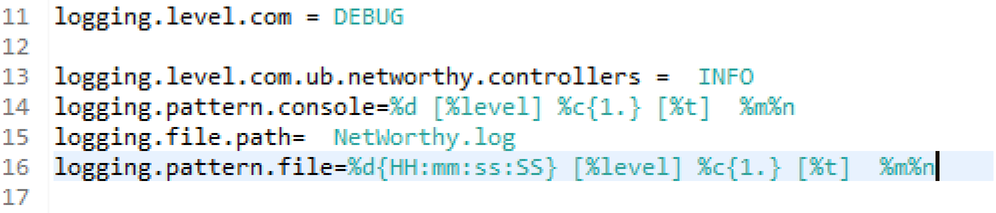
```
curl -X GET "http://localhost:8080/api/content/getAll/Debt" -H "accept: */*"
```

```
http://localhost:8080/api/content/getAll/Debt
```

```
[
  {
    "contentNumber": 8,
    "tag": "Debt",
    "title": "The Main Types Of Debt And How To Handle Each",
    "description": "Learn about different types of debt, the advantages and disadvantages of each, and tips on how to handle them if you find yourself with debt",
    "author": "Forbes",
    "pcLink": "https://www.forbes.com/advisor/debt-relief/types-of-debt/",
    "learningMethod": "Article",
    "length": "4 min"
  },
  {
    "contentNumber": 9,
    "tag": "Debt",
    "title": "Understanding Debt",
    "description": "Learn what sorts of debt you may be carrying currently and how they work",
    "author": "Nerdwallet",
    "pcLink": "https://www.nerdwallet.com/uk/personal-finance/types-of-debt/",
    "learningMethod": "Article",
    "length": "5 min"
  },
  {
    "contentNumber": 10,
    "tag": "Debt",
    "title": "The Biden-Harris Administration's Student Debt Relief Plan Explained",
    "description": "Learn if you qualify for the Student Debt Relief Plan",
    "author": "Federal Student Aid",
    "pcLink": "https://studentaid.gov/debt-relief-announcement/",
```

4. Logging :-

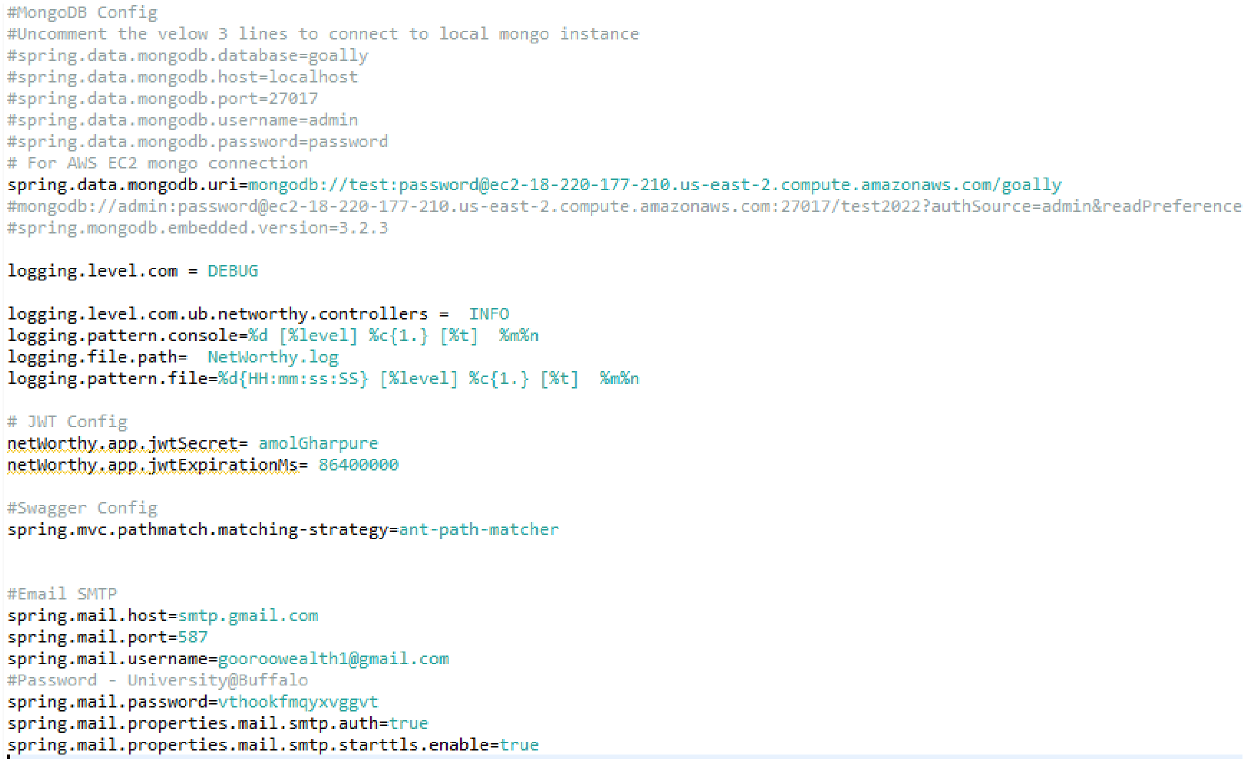
   For application logging we have used SLF4J logging.
   A different logging level and pattern can be set as per requirements.
   “Application.properties”



```
logging.level.com = DEBUG

logging.level.com.ub.networthy.controllers =  INFO
logging.pattern.console=%d [%level] %c{1.} [%t]  %m%n
logging.file.path=  NetWorthy.log
logging.pattern.file=%d{HH:mm:ss:SS} [%level] %c{1.} [%t]  %m%n

```

5. Application Properties

   Comments are added for each config property.



```
#MongoDB Config
#Uncomment the velow 3 lines to connect to local mongo instance
#spring.data.mongodb.database=goally
#spring.data.mongodb.host=localhost
#spring.data.mongodb.port=27017
#spring.data.mongodb.username=admin
#spring.data.mongodb.password=password
# For AWS EC2 mongo connection
spring.data.mongodb.uri=mongodb://test:password@ec2-18-220-177-210.us-east-2.compute.amazonaws.com/goally
#mongodb://admin:password@ec2-18-220-177-210.us-east-2.compute.amazonaws.com:27017/test2022?authSource=admin&readPreference
#spring.mongodb.embedded.version=3.2.3

logging.level.com = DEBUG

logging.level.com.ub.networthy.controllers =  INFO
logging.pattern.console=%d [%level] %c{1.} [%t]  %m%n
logging.file.path=  NetWorthy.log
logging.pattern.file=%d{HH:mm:ss:SS} [%level] %c{1.} [%t]  %m%n

# JWT Config
netWorthy.app.jwtSecret= amolGharpure
netWorthy.app.jwtExpirationMs= 86400000

#Swagger Config
spring.mvc.pathmatch.matching-strategy=ant-path-matcher


#Email SMTP
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=gooroowealth1@gmail.com
#Password - University@Buffalo
spring.mail.password=vthookfmqyxvggvt
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

MongoDB

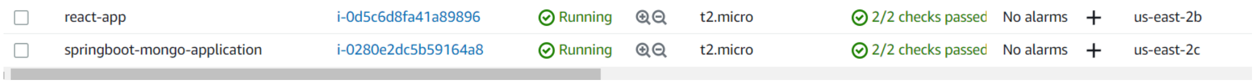
We have used MongoDb 6.0.3 for the project.
To setup on local machine.

1. Install mongo on your local machine. (default port is 27017)
2. Create a database goally – “use goally”
3. We need to create 3 static collections and import the data from excel files.
    a. “roles” – roles.csv
    b. “content” - Personalized Content.csv
    c. “tags” - Tags.csv
4. You can refer the “Database – Steps to Add Collection.docs” for more information
5. Rest all required collections will be created dynamically by SpringBoot.

Deployment Documentation

For deploying the application we have used 2 EC2 instances due to the system memory issue.

1. “springboot-mongo-application”:- SpringBoot and MongoDb
2. “react-app: - ReactJS application

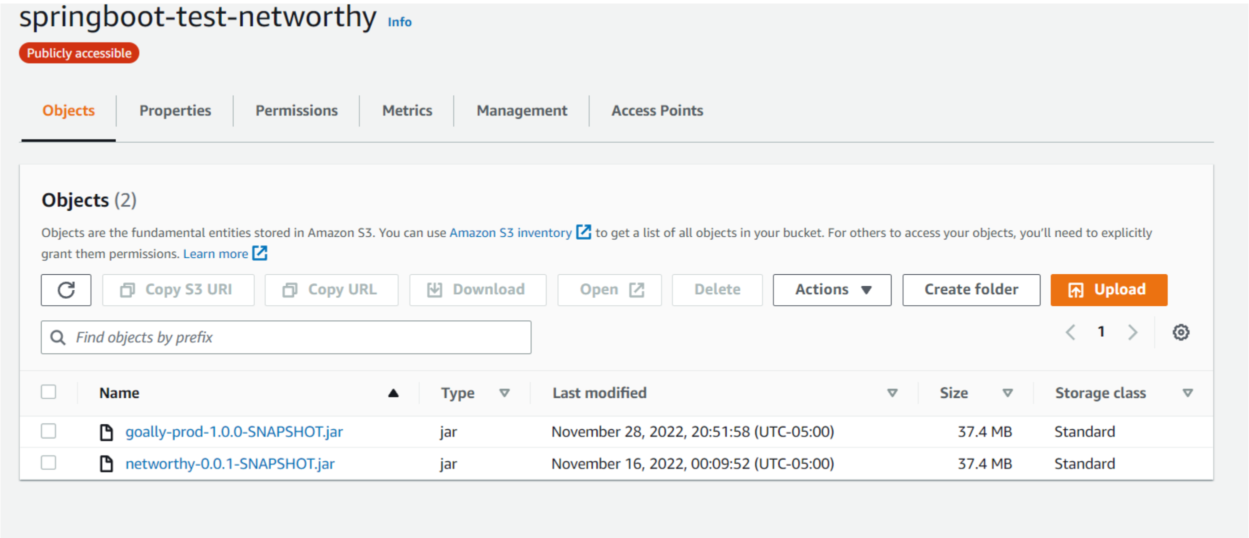


□	react-app	i-0d5c6d8fa41a89896	✓ Running		t2.micro	✓ 2/2 checks passed	No alarms	+	us-east-2b
□	springboot-mongo-application	i-0280e2dc5b59164a8	✓ Running		t2.micro	✓ 2/2 checks passed	No alarms	+	us-east-2c

User needs to SSH or use putty into the correct EC2 instance. “.ppk” keys can be downloaded from the AWS console to connect to these instances.

SpringBoot Deployment

1. Upload the latest jar file to your S3 bucket.



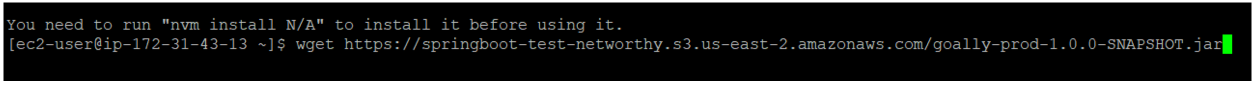
springboot-test-networthy Info

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name ▲	Type	Last modified	Size	Storage class
□	goally-prod-1.0.0-SNAPSHOT.jar	jar	November 28, 2022, 20:51:58 (UTC-05:00)	37.4 MB	Standard
□	networthy-0.0.1-SNAPSHOT.jar	jar	November 16, 2022, 00:09:52 (UTC-05:00)	37.4 MB	Standard

2. Copy the url for the jar file and go to your EC2 instance ssh/putty screen.
   Use “wget {URL}” to download the latest jar file in your EC2 instance.

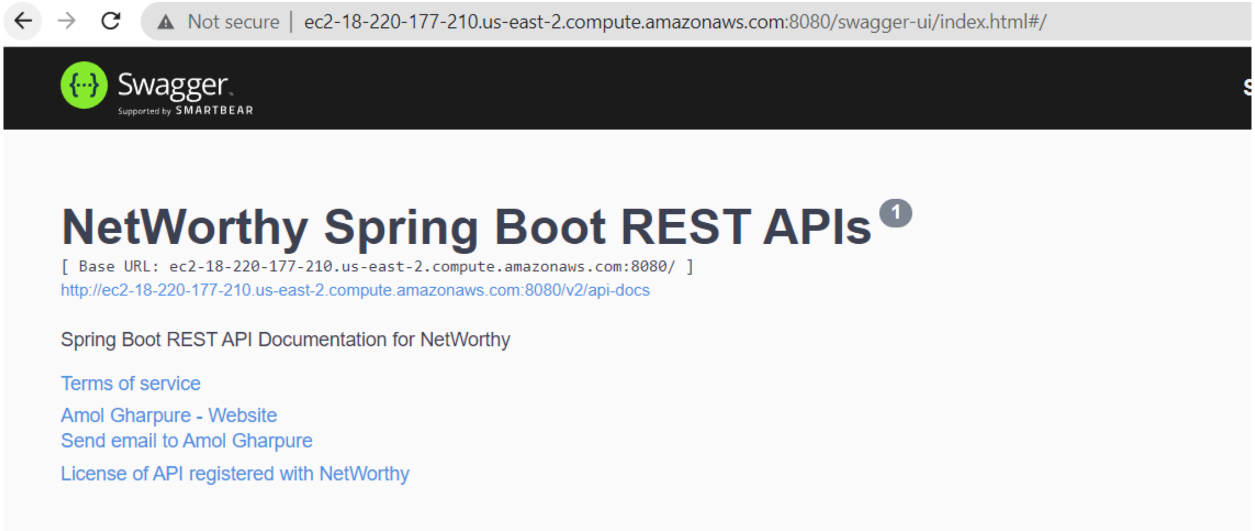


```
You need to run "nvm install N/A" to install it before using it.
[ec2-user@ip-172-31-43-13 ~]$ wget https://springboot-test-networthy.s3.us-east-2.amazonaws.com/goally-prod-1.0.0-SNAPSHOT.jar
```

3. Go to the correct directory and use the below command to run the springboot app as a background task.

   “nohup java -jar goally-prod-1.0.0-SNAPSHOT.jar”

4. Go to your swagger and check if the Spring app is up and running:-


Not secure | ec2-18-220-177-210.us-east-2.compute.amazonaws.com:8080/swagger-ui/index.html#/
Swagger
Supported by SMARTBEAR

NetWorthy Spring Boot REST APIs 1

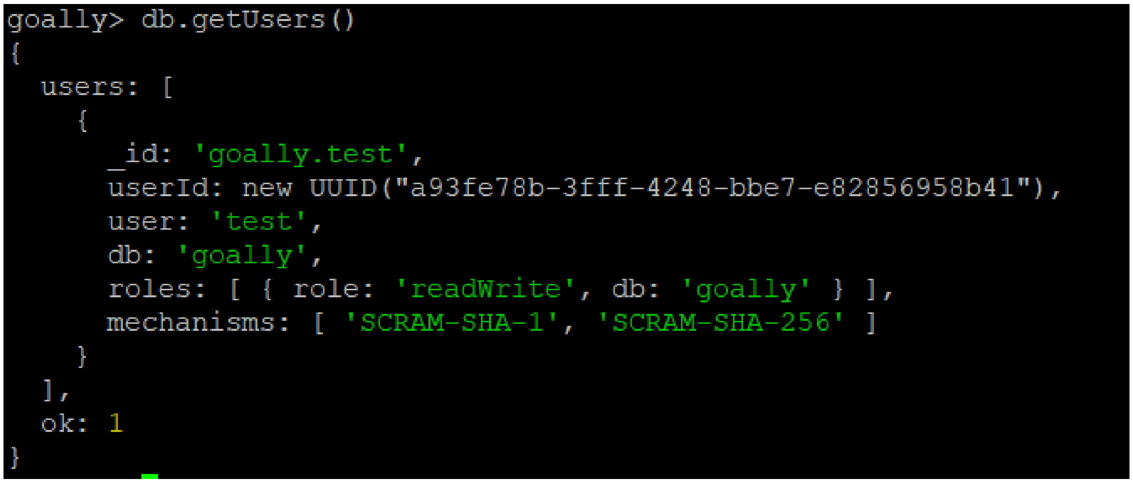
[Base URL: ec2-18-220-177-210.us-east-2.compute.amazonaws.com:8080/]
http://ec2-18-220-177-210.us-east-2.compute.amazonaws.com:8080/v2/api-docs

Spring Boot REST API Documentation for NetWorthy

Terms of service
Amol Gharpure - Website
Send email to Amol Gharpure
License of API registered with NetWorthy

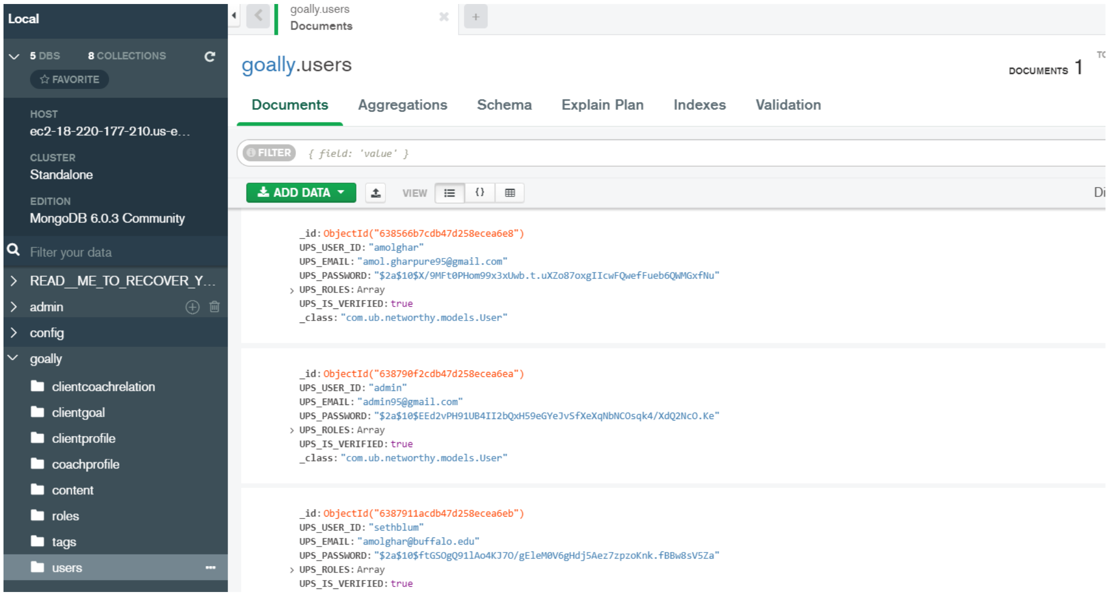
MongoDB Deployment

1. On the same instance use “mongosh -u admin -p password” to access the mongo shell with admin privileges.
2. In case you add a new database. There will be read and write permissions required for a user that will login into that database used in your SpringBoot sonfig.



```
goally> db.getUsers()
{
  users: [
    {
      _id: 'goally.test',
      userId: new UUID("a93fe78b-3fff-4248-bbe7-e82856958b41"),
      user: 'test',
      db: 'goally',
      roles: [ { role: 'readWrite', db: 'goally' } ],
      mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ]
    }
  ],
  ok: 1
}
```

3. You can now connect this Mongo remote instance with your local Mongo compass :-
URL :- “mongodb://admin:password@ec2-18-220-177-210.us-east-2.compute.amazonaws.com:27017/test2022?authSource=admin&readPreference=primary&appname=MongoDB%20Compass&directConnection=true&ssl=false”
You can see all the collections for the DB “Goally”. To add the static tables roles, content and tags refer to the MongoDB section in Backend Technical Document.



HOST
ec2-18-220-177-210.us-e...
CLUSTER
Standalone
EDITION
MongoDB 6.0.3 Community

```
{ field: 'value' }
```

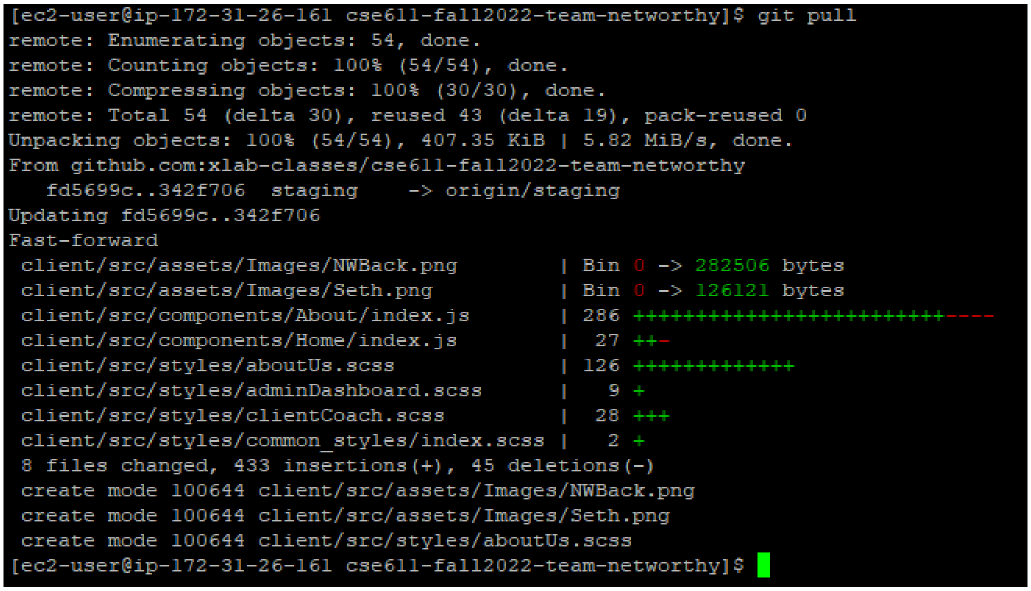
```
_id: ObjectId("638566b7cdb47d258ecea6e8")
UPS_USER_ID: "amolghar"
UPS_EMAIL: "amol.gharpure95@gmail.com"
UPS_PASSWORD: "$2a$10$X/9MFt0PHom99x3xUwb.t.uXZo87oxgIIcwFQwefFueb6QWMGxfNu"
> UPS_ROLES: Array
UPS_IS_VERIFIED: true
_class: "com.ub.networthy.models.User"

_id: ObjectId("638790f2cdb47d258ecea6ea")
UPS_USER_ID: "admin"
UPS_EMAIL: "admin95@gmail.com"
UPS_PASSWORD: "$2a$10$EEd2vPH91UB4II2bQxH59eGYeJvSfXeXqNbNCOsqk4/XdQ2NcO.Ke"
> UPS_ROLES: Array
UPS_IS_VERIFIED: true
_class: "com.ub.networthy.models.User"

_id: ObjectId("6387911acdb47d258ecea6eb")
UPS_USER_ID: "sethblum"
UPS_EMAIL: "amolghar@buffalo.edu"
UPS_PASSWORD: "$2a$10$ftGSOgQ91lAo4KJ7O/gEleM0V6gHdj5Aez7zpzoKnk.fBBw8sV5Za"
> UPS_ROLES: Array
UPS_IS_VERIFIED: true
```

React Application Deployment

SSH into the react-app EC2 instance.

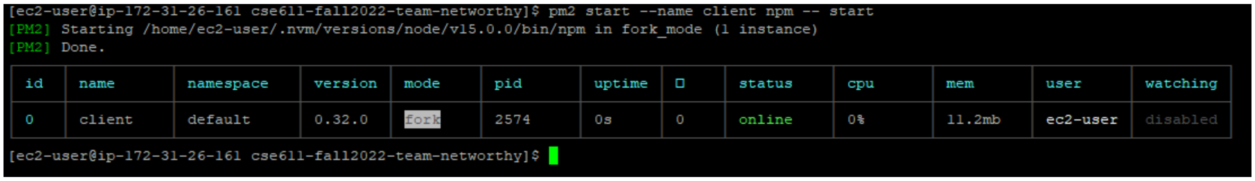
We have linked the react github with this instance. The user just needs to pull the latest changes and run the app.

1. To pull the latest changes, go to the react app directory and run “git pull”



```
[ec2-user@ip-172-31-26-161 cse611-fall2022-team-networthy]$ git pull
remote: Enumerating objects: 54, done.
remote: Counting objects: 100% (54/54), done.
remote: Compressing objects: 100% (30/30), done.
remote: Total 54 (delta 30), reused 43 (delta 19), pack-reused 0
Unpacking objects: 100% (54/54), 407.35 KiB | 5.82 MiB/s, done.
From github.com:xlab-classes/cse611-fall2022-team-networthy
   fd5699c..342f706  staging    -> origin/staging
Updating fd5699c..342f706
Fast-forward
 client/src/assets/Images/NWBack.png         | Bin 0 -> 282506 bytes
 client/src/assets/Images/Seth.png           | Bin 0 -> 126121 bytes
 client/src/components/About/index.js        | 286 ++++++++++++++++++++++++----
 client/src/components/Home/index.js         |  27 ++-
 client/src/styles/aboutUs.scss              | 126 +++++++++++++
 client/src/styles/adminDashboard.scss       |   9 +
 client/src/styles/clientCoach.scss          |  28 +++
 client/src/styles/common_styles/index.scss  |   2 +
 8 files changed, 433 insertions(+), 45 deletions(-)
 create mode 100644 client/src/assets/Images/NWBack.png
 create mode 100644 client/src/assets/Images/Seth.png
 create mode 100644 client/src/styles/aboutUs.scss
[ec2-user@ip-172-31-26-161 cse611-fall2022-team-networthy]$
```

2. We are running this app on PM2. Use the following command to start the app “pm2 start --name client npm -- start”



```
[ec2-user@ip-172-31-26-161 cse611-fall2022-team-networthy]$ pm2 start --name client npm -- start
[PM2] Starting /home/ec2-user/.nvm/versions/node/v15.0.0/bin/npm in fork_mode (1 instance)
[PM2] Done.
┌────┬──────────┬───────────┬─────────┬───────┬──────┬────────┬───┬────────┬─────┬────────┬──────────┬──────────┐
│ id │ name     │ namespace │ version │ mode  │ pid  │ uptime │ □ │ status │ cpu │ mem    │ user     │ watching │
├────┼──────────┼───────────┼─────────┼───────┼──────┼────────┼───┼────────┼─────┼────────┼──────────┼──────────┤
│ 0  │ client   │ default   │ 0.32.0  │ fork  │ 2574 │ 0s     │ 0 │ online │ 0%  │ 11.2mb │ ec2-user │ disabled │
└────┴──────────┴───────────┴─────────┴───────┴──────┴────────┴───┴────────┴─────┴────────┴──────────┴──────────┘
[ec2-user@ip-172-31-26-161 cse611-fall2022-team-networthy]$
```

3. The Application is now up and running.