**Sign up form (Auth Module)**

Single form layout for mentor and student roles.

Template driven (To be converted to reactive form)

Front end Validation:

**Note: All fields are required**

* Email validation for a pattern!
* First name and Last name fields should not be empty
* Same password entries in two input
* Username should be unique (should make backend queries to check to let user know)
* For mentor LinkedIn URL should match across a pattern
* Button should only be working if form is valid but that alone should not be only condition for form being valid. When it is clicked do check again for validation

Back end Validation:

* Same validation should be performed in backend too before saving into any respective collection

Passwords should be encrypted in backend before saving into collection. (bcrypt-nodejs)

**Note: Never store plain passwords in the database!**

Express middleware is used to implement this hashing logic before entry is saved in the database.

Once an entry is done, it’s time to log them in! We create JSON web token with payload as an object with id and role as a parameter. Random signature named brainees is used to create JWT along with payload, and this token is sent to client!

**JSON Web Token (JWT) Take a look at this section to understand JWT**

We will be creating a token that's encrypted and contains JSON data or a JWT.

Once we generate the token on backend, we will send that with the response to

frontend. Since its encrypted, the frontend will not know what's inside it but

it will trust that backend had sent it an authentication token, and the frontend

will save it so that the next time the browser loads, it will have that authentication

token, and the user won’t have to login again (Similar to cookie authentication)

To access any private resource, we send the token along with the secure request and when it arrives at backend the server decrypts it and realizes it’s the same user and grant them permission to make request.

Tokens consists of Payload, secret which is used to protect payload. In Production environ you may not hardcode the secret (2nd param of jwt.encode)

Decode payload at frontend JSON.parse(atob(encodedtoken.split('')[1]))

The atob() method decodes a base-64 encoded string

**Note: If no token is present with the request or token is modified the server would response with 401 unauthorized access and Angular Interceptor for response status 401 would redirect to login page! Angular Interceptor also is attaching token if present in LocalStorage to every request made to sever**

**NOTE:** Middleware name checkAuthenticated (**src/auth.js**) will send 401 if token is not present in header or if payload is incorrect.

**Backend Routes responsible for signing up either roles**

* **routes/mentorRoutes.js** (Usage of controllers, which uses module revealing pattern)
* **routes/studentRoutes.js** (Usage of controllers, which uses module revealing pattern)

Body Parser is used to parse body and whole communication is done passing **JSON**,so content-type(POST) and Accept(GET) could be set **application/json** via an interceptor

**Server status 503 is sent in case of unsuccessful entry of user in database, else {message,token}**

Also **note,** SignUpGuard is implemented on Signup Component and will never be displayed to user who is signed in, if tried to access it will redirect it to **/role/dashboard**

**Login Form (Auth Module)**

Single form layout for mentor and student roles.

Template driven (To be converted to reactive form)

Front end Validation:

* All fields must be filled!
* Button should only be enabled if form is valid

Backend Validation:

* Starts querying database in order Student, Mentor
* Uses bcrypt-nodejs to compare hashed password with input password
* In case of a valid match sends a response with token!
* In case of invalid match sends a response with status code **401**(Invalid id or password)

After a successful login user is redirected to **/role/dashboard**

Also **note,** LoginGuard is implemented on Login Component and will never be displayed to user who is signed in, if tried to access it will redirect it to **/role/dashboard**

**Backend Routes responsible for logging in a user:**

* **routes/loginRoutes.js**

**Student Module**

Components:

* Home/Dashboard
* Analytics
* My mentors
* Quizzes

**Home/Dashboard**

A student can see all the posts of a mentor he/she is subscribed to, can also like them and can view public profile of mentor by clicking on the name of mentor in posts.

**Frontend services involved:**

* getMyPosts() **ngOnInit**
* Queries Post collection to fetch the post and then Mentor model to fetch more data about who posted it

**Backend Routes involved:**

* /student/general/getmyposts

**Analytics**

//TODO

**My mentor**

Displays all the mentors to which the user is subscribed. Also displays all available mentors for more subscription.

**Frontend services involved**

* getMyMentors() **ngOnInit**
* Responsible for fetching all mentors subscribed by the user
* Queries MentorStudentSubscription to get subscribed data and then queries Mentor collection to fetch more data!
* browseMentors()
* Queries Mentor collection to fetch mentors which are not in MentorStudentSubscription collection
* subscribeMentor()
* Leaves a request (isApproved:false) with a mentor, and removes that mentor from browseMentor list
* unSubscribeMentor()
* Simply unsubscribe a mentor

**Backend routes involved**

* /student/general/getmymentors
* /student/general/browsementors
* /student/general/subscribementor
* /student/general/unsubscribementor/:id

**Quizzes**

**//TODO**

**Mentor Module**

Components:

* Home/Dashboard
* Set Quiz
* My students
* My courses
* New requesrs

**Home/Dashboard**

A mentor can post and see all the posts of himself/herself, can also like them and can view public profile of those who liked.

**Frontend services involved:**

* getMyPosts() **ngOnInit**
* Queries Post collection to fetch the post and then Mentor model to fetch more data about who posted it

**Backend Routes involved:**

* /student/general/getmyposts

**Set Quiz**

//TODO

**My students**

Displays all the students subscribed with the mentor. Mentor can also remove a student from here.

**Frontend services involved**

* getMyMentors() **ngOnInit**
* Responsible for fetching all mentors subscribed by the user
* Queries MentorStudentSubscription to get subscribed data and then queries Mentor collection to fetch more data!
* browseMentors()
* Queries Mentor collection to fetch mentors which are not in MentorStudentSubscription collection
* subscribeMentor()
* Leaves a request (isApproved:false) with a mentor, and removes that mentor from browseMentor list
* unSubscribeMentor()
* Simply unsubscribe a mentor

**Backend routes involved**

* /student/general/getmymentors
* /student/general/browsementors
* /student/general/subscribementor
* /student/general/unsubscribementor/:id

**My Courses**

**//TODO**

**New Requests**

**For Angular 4.3+:**

Import HttpClientModule

Requests using HttpClient class (Makes heavy usage of RxJS)

One of the nice things in working of responses with RxJS is number of method available with RxJS to manipulate the data(operators)

**New Syntaxes: Pipeable|Lettable operators**

Many of the method in the http client returns Observables(part of RxJS)

**HTTPS headers can be passed via requests in option parameters which takes object literal**

Successful POST should return 201 Created status code

Successful GET returns 200 OK

Successful update returns 204 No content

Successful delete returns 204 No Content

**You can prefetch data for the new component before activating the new route using resolvers**

**Error handling**

Handle and encapsulate errors in service

Don’t expose implementation details to the component

Use RxJS “catchError” operator for the same, It will allow you to return custom error to components

Use ErrorObservable to send custom error observable back to component

**Resolvers**

Fetch data before navigating

Prevent presentation of an empty component

Prevent routing to component with errors

Better user experience

Services that implement resolve interface

**Interceptors**

Services that implement the HttpInterceptor interface

Manipulate HTTP requests before they’re sent to server (Same with responses)