Module 4

In this task, you will create an app routing, authorization mechanism and glue all the app with help of services.

Criteria (10 points max)

1. [1 points] Implement `app-routing.module.ts` with such lazy load modules:

- Login page (`/login` route)
- Registration page (`/registration` route)
- [default / fallback url] Courses page with a list of courses (`/courses` route)
- Add page for adding a new course (`/courses/add` route)
- Add page for showing an existing course (`/courses/:id` route, where `id` – course id)
- Add page for editing an existing course (`/courses/edit/:id` route, where `id` – course id)

2. [2 points] Implement authors API & store services:

- Generate `authors.service.ts` & `authors-store.service.ts` with help of Angular CLI in `/src/app/services` folder.
- According to Swagger of backend app, implement `getAll` and `addAuthor` methods for `authors.service.ts` which should make API calls.
- Implement mechanism of storing authors and reflecting loading state in `authos-store.service.ts` by creating `getAll` and `addAuthor` methods and using `authors.service.ts` inside them. Also, implement private properties `isLoading$$` & `authors$$` which should be Behavior subjects, in addition to provide access to those variables and to limit external control, make public properties `isLoading$` & `authors$` which should be just Observables (and could be got by calling `.asObservable` on earlier created Behaviour subjects).

3. [2 points] Implement courses API & store services:

- Generate `courses.service.ts` & `courses-store.service.ts` with help of Angular CLI in `/src/app/services` folder.
- According to Swagger of backend app, implement following methods for `authors.service.ts` which should make API calls:
  - `getAll`
  - `createCourse`
  - `editCourse`
  - `getCourse`
  - `deleteCourse`
- Implement mechanism of storing courses and reflecting loading state in `courses-store.service.ts` by creating following methods and using `courses.service.ts` inside them:
  - `getAll`
  - `createCourse`
  - `editCourse`
  - `getCourse`
  - `deleteCourse`

- Implement private properties `isLoading$$` & `courses$$` which should be Behavior subjects, in addition to provide access to those variables and to limit external control, make public properties `isLoading$` & `courses$` which should be just Observables (and could be got by calling `.asObservable` on earlier created Behaviour subjects).
- Implement method(s) for searching courses (in case of using those method(s) the data in `courses$` variable should reflect current state)

4. [2 points] Implement authorization by creating `auth` module:

- Generate `auth` module with help of Angular CLI in `src/app` folder.
- Implement `session-storage.service.ts`:
  - Generate `session-storage.service.ts` with help of Angular CLI in `/src/app/auth/services` folder.
  - Inject Window with help of `@Inject` decorator inside service.
  - Implement `setToken(token: string)`, `getToken` and `deleteToken` methods which should use sessionStorage.
- Implement `auth.service.ts`:
  - Generate `auth.service.ts` with help of Angular CLI in `/src/app/auth/services` folder.
  - According to Swagger of backend app implement `login`, `logout` and `register` methods which should make API calls.
  - Create a pair of properties (private Behaviour subject `isAuthorized$$` + public Observable `isAuthorized$`) for `isAuthorised` state.
  - `session-storage.service.ts` should be used inside `auth.service.ts` methods for operations with token.
- Implement guards:
  - Generate `authorized.guard.ts` & `not-authorized.guard.ts` with help of Angular CLI in `/src/app/auth/guards` folder.
  - Guards should use `auth.service.ts` for checking if user is authorized.
  - `authorized.guard.ts` should implement `canLoad` interface. In case if user is authorized guard returns `true`, if not - redirects to `/login` with help of `UrlTree`.
  - `not-authorized.guard.ts` should implement `canActivate` interface. In case if user is not authorized guard returns `true`, if not - redirects to `/courses` with help of `UrlTree`.
- Add guards usage:
  - Use `authorized` guard for `/courses`, `/course/add`, `/course/edit/:id` and `/course/:id` routes.
  - Use `not-authorized` guard for `/login` and `/registration` routes.
- Implement `token.interceptor.ts` for adding token to requests in `/src/app/auth/interceptors` folder. In case when request has `401` error, logout and redirect to `/login`.

5. [1 point] Implement user module:

- Generate `user.module.ts` with help of Angular CLI in `/src/app/user` folder.
- Implement `user.service.ts` & `user-store.service.ts`
  - Generate `user.service.ts` & `user-store.service.ts` with help of Angular CLI in `/src/app/user/services` folder.
  - According to Swagger of backend app, implement `getUser` method for `user.service.ts` which should make API call.

    - Implement mechanism of storing authors and reflecting `isAdmin` state in `user-store.service.ts` by creating `getUser` method and using `user.service.ts` inside it. Also, implement private properties `name$$` & `isAdmin$$` which should be Behavior subjects, in addition to provide access to those variables and to limit external control, make public properties `name$` & `isAdmin$` which should be just Observables (and could be got by calling `.asObservable` on earlier created Behaviour subjects).
- Implement `admin.guard.ts` which should follow `CanActivate` interface and should return `true` in case if user is admin and redirect to `/courses` in case if user is not admin with help of `UrlTree`.
- Add `admin.guard.ts` for `/course/add` and `/course/edit/:id` routes.

6. [2 points] Integrate earlier created components with pages and services:

- Remove all earlier mocked data and replace it with the created services.
- Implement search for courses page with help of `courses.services.ts`.
- Show controls for editing/removing course only in case if user has `isAdmin=true`.
- In case when user clicks on remove course button - show confirm modal window and delete course only in case of confirmation.
- In case when user clicks on edit course button - redirect to edit course page of exact course.
- In case when user clicks on show course button - redirect to course page of exact course.
- Add course in case of submitting form on add new course page.
- Update course in case of submitting form on edit course page.