ENUNCIAT PAC 2

Programació reactiva:

RxJS + NgRX

Desenvolupament front-end avançat

Máster universitari en desenvolupament de llocs i aplicacions web

Universitat Oberta de Catalunya

Contingut

- Format d'entrega
- Enunciat
- Puntuació

Format d'entrega

S'entregarà un fitxer comprimit en format **.zip** del projecte **blog-uoc-project-front** que **resol la pràctica**.

També un document de text comentant qualsevol aspecte que sigui rellevant per a tindre en compte, per exemple, algunes decisions de disseny, **alguna part que no funcioni o hagi quedat pendent**, ... En definitiva, aspectes que jo com a "corrector" hagi de tenir en compte. Per altra banda, estaria bé afegir alguna captura de pantalla del **Redux dev-tools** per a tenir una imatge visual del vostre magatzem de dades.

- **Aquesta pràctica es farà a partir de la solució de la pràctica 1**
- Recordeu de no influir el directori **"node_modules"** als projectes

Enunciat

Exercici 1 – De promeses a observables

En aquest primer exercici **partirem de la solució oficial de la pràctica 1** i haurem d'aplicar-li uns canvis.

Primer haurem d'estudiar la teoria del document **Teoría_PEC2_RXJS_es.pdf**, i, un cop estudiada, tindrem que **transformar tots els serveis que fan les crides a la api, que ara estan implementats amb promeses, a observables, i adaptar la resta del projecte per a que funcioni correctament.**

Fixeu-vos que, al documento de teoria, a les pàgines 4 i 5 teniu un exemple senzill de com fer-ho.

Bàsicament, tindreu que anar passant a observables totes les crides de tots els serveis del projecte i poc a poc, a mesura ho aneu fent, tindreu que anar adaptant el codi als controladors per a que tot continuï funcionant de la mateixa manera.

¡Consell! ¡No modifiqueu tots els serveis a la vegada! Modifiqueu un servei, busqueu al projecte allí on s'utilitza i adapteu el codi del controlador. Valideu que funciona y a partir d'aquí modifiqueu el següent servei. Intenteu anar pas a pas per tal de garantir la integritat i la consistència del que aneu implementant. Si toqueu moltes coses i després tenim errors, és més complicat trobar-los.

Concretament, haureu de passar a observables els serveis següents:

- **AuthService**: login
- **CategoryService**: getCategoriesByUserId, createCategory, getCategoryById, updateCategory, deleteCategory
- **PostService**: getPosts, getPostsByUserId, createPost, getPostById, updatePost, likePost, dislikePost, deletePost
- **UserService**: register, updateUser, getUSerById

Penseu quan adapteu el codi dels controladors que canviarà el comportament, és a dir, en lloc d'utilitzar el **async/await** ens subscriurem a l'observable i dintre de la "subscripció" haurem d'implementar el codi que toqui.

Exercici 2 – Pla de proves

En aquest segon exercici haurem de crear un pla de proves per validar que el funcionament de **tots els casos d'ús** segueix funcionant correctament després de passar els serveis de promeses a observables.

Es proposa crear un Excel que inclogui per a cada cas d'ús les següents **columnes**:

- **Cas d'ús** (nom curt)
- **Descripció** (descripció més detallada del cas d'ús)
- **Entrada** (condicions d'entrada que s'han de complir per poder fer la prova)
- **Sortida esperada** (resultat teòric esperat de la prova)
- **Resultat real amb transformació a Observables** (resultat real d'executar la prova determinada un cop passat el servei a Observables)
- **Intents** (si detecteu un error, es corregeix i s'indica el nombre d'iteracions necessàries)
- **Observacions** (per indicar si es detecta un error i es corregeix, quines accions s'han implementat per resoldre'l)

Cal plantejar una **fila** d'aquest Excel per a cada cas d'ús. Hem de cobrir tots els possibles casos per garantir la integritat de la nostra aplicació.

Exemples:

- login amb dades correctes
- login amb dades incorrectes
- editar un post (resultat esperat, s'haurà de validar el canvi, que la resta de dades es mantinguin, …)
- alta de categoria
- edició de categoria
- ...

Exercici 3 – Aplicar REDUX al nostre projecte

Al exercici 1 hem transformat totes les crides a la api dels nostres serveis de promeses a observables. **Partint d'aquest projecte de l'exercici 1**, haurem de transformar la gestió de l'usuari autenticat per a que funcioni amb el patró **Redux**.

Es a dir, en comptes d'utilitzar el **localStorage** per emmagatzemar el **userId** i **access_token** i fer les corresponents gestions, utilitzarem **Redux**. També deixarem d'utilitzar el servei **HeaderMenuService** ja que passarem a gestionar-ho via **Redux**. Per tant, al finalitzar l'exercici podrem eliminar els serveis **LocalStorageService** i **HeaderMenuService** i totes les seves referències.

Si analitzem la solució de la pràctica 1, veiem on s'utilitza **userId** i el **access_token**:

- Al **login.component.ts** quan fem **login**, obteníem la resposta de la api amb el **userId** i el **access_token** i ho guardàvem al **localStorage** per a consultar-lo posteriorment quan fos necessari.
    - Això quedarà substituït pel **dispatch** de l'acció de **login** de l'estat de **Auth**, de manera que en aquest estat de **Redux** tindrem guardat tant **userId** com **acces_token** per a fer la subscripció i consultar aquestes dades quan sigui necessari
- Un cop autenticats i tenint al **localStorage** l'**userId** i l'**access_token**, aquests s'utilitzen en diferents parts del projecte:
    - **categories-list.component.ts** per recuperar l'**userId** i passar-li a la crida **getCategoriesByUserId**
    - **category-form.component.ts** per recuperar l'**userId** i utilitzar-lo per crear o actualitzar una categoria
    - **posts-list.component.ts** per recuperar l'**userId** i passar-li a la crida **getPostsByUserId**
    - **post-form.component.ts** per recuperar l'**userId** i utilitzar-lo per crear o actualitzar un post i per a recuperar les categories de l'usuari autenticat
    - **profile.component.ts** per recuperar l'**userId** i utilitzar-lo per a recuperar les dades del perfil i poder fer actualitzacions
    - Per a tots aquests casos anteriors extraurem l'**userId** de l'estat de **Auth** en lloc del **localStorage**
- **home.component.ts** per a recuperar l'**userId** i si existeix mostrar els botons de **like** i **dislike**
    - Aquí consultarem l'**userId** de l'estat de **auth**
- **header.component.ts** subscripció al servei **headerMenuService** per saber si mostrar el menú públic o privat i al mètode de **logout** elimina del **localStorge**

l'**userId** i l'**acces_token** i fa un **next** al servei **headerMenuService** per indicar que no estem autenticats.

  - Això quedarà substituït per una subscripció al estat **Auth** i al **logout** fent la crida del **dispatch** de l'acció de **logout** de l'estat de **Auth**

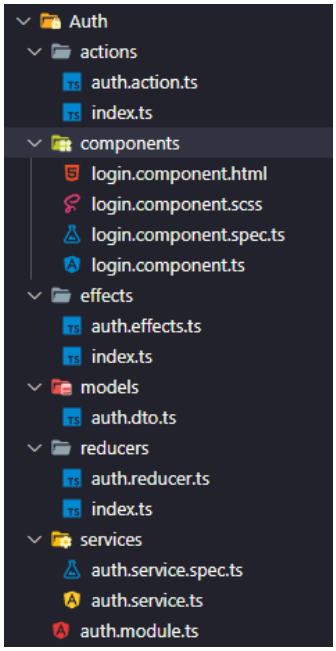
- **auth.guard.ts** recupera l'**access_token**, si el troba habilita el poder accedir a la ruta determinada

  - Aquí consultarem l'**access_token** de l'estat de **Auth**

- **auth-interceptor.service.ts** recupera l'**access_token**, si el troba l'afegeix als **headers** de la petició en curs a la api

  - Aquí consultarem l'**access_token** de l'estat de **Auth**

La proposta per estructurar el mòdul **Auth**:



La proposta del estat per gestionar la autenticació:

```
export interface AuthState {
  credentials: AuthDTO;
  loading: boolean;
  loaded: boolean;
  error: any;
}
```

Per a resoldre aquest exercici ajudeu-vos del **Redux dev-tools** i afegiu alguna captura de pantalla al document que adjunteu a la entrega per tal de tenir la “foto” de tot el vostre projecte.

Exercici 4 – Pla de proves

En aquest quart i últim exercici afegirem una nova columna a l'Excel del pla de proves de l'exercici 2, el nom de la qual serà **Resultat real amb transformació a Redux**, i tornarem a executar tot el pla de proves.

D'aquesta manera, en tornar a provar tots els casos d'ús, podrem validar que la implementació a **Redux** no "trenca" altres funcionalitats.

Puntuació

A continuació, mostrem quan puntuen cada exercici:

- Exercici 1 **[4 punts]**
- Exercici 2 **[1 punts]**
- Exercici 3 **[4 punts]**
- Exercici 4 **[1 punts]**