UNIT 2

Angular

Final exercise

Client-side Web Development
2nd course – DAW
IES San Vicente 2024/2025
Author: Arturo Bernal Mayordomo

Index

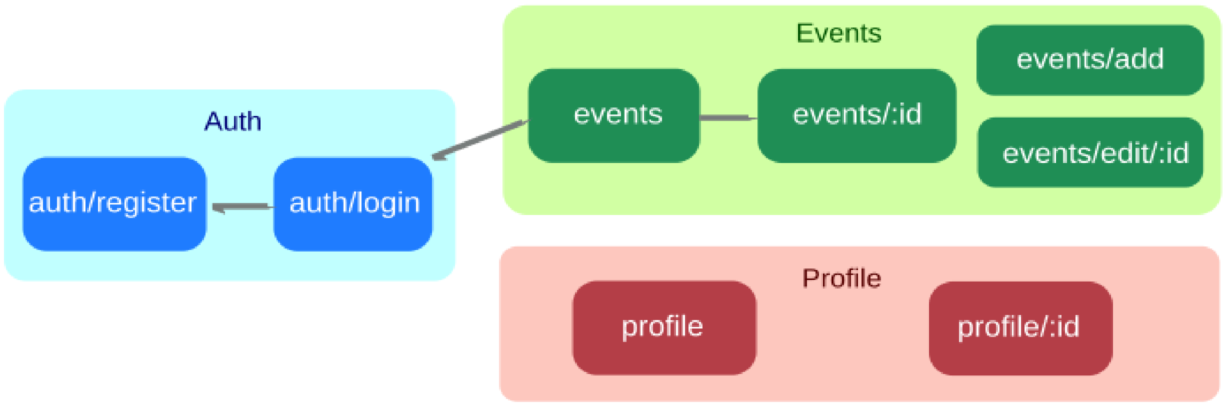
SVTickets (Angular Version)

This final exercise will be an expansion of the exercises we've been doing during this unit. Use week 11 exercise as a starting point.

Web services are the same as in unit 1, with some additions like Google or Facebook login -> https://github.com/arturober/svtickets-services

Routes

The application will have the following routes:



Not authenticated routes

auth/register

This page will contain a form for the user to register. Create the same form used in unit 1 project and **validate it** in the client side: all fields are required, email fields must be of type email, and password must have at least 4 characters .

Also, create a validator that validates that both emails are equal. Put the error message for this validator below the "repeat email" input (with the corresponding css class for that input). You can create a group validator to check both values, or just create a normal validator that you put on the second email field and receives the first email field as the input value.

Don't forget to geolocate the user and send the coordinates to the server.

auth/login

This page will contain a page with a form to log in (email, password) and the Google and Facebook buttons for login/register. Validate the form (all fields required).

Also geolocate the user here and send the coordinates with the login information (lat, lng). This includes the normal login and also Google/Facebook login.

Authenticated routes

events

This page will show all posts (like in previous exercises). Use the same card you used in unit 1 project.

The button to attend an event (**/events/:id/attend**) is now mandatory and must work updating the number of people attending (and the attend boolean), and also the text color and the icon.

In the card (**important** → if the event is yours):

- Put the button to delete the post (this will show a confirm dialog asking if you're sure you want to delete it using ngBootstrap o ngx-sweetalert2).
- Put a button to edit the post (below the description for example). The edit button will go to **/event/:id/edit**. It can be a link with the btn classes. Example: <a class="btn btn-primary" ...>Edit</a>.

Filtering the events

You must filter and order the events like you did in the TypeScript project. You can choose 2 ways of doing this:

Traditional

Search and ordering, and next page will be activated by click events. The events will call methods that change the search, page, or order value, and then call another method to load events based on these values

Reactive (+0,25 points)

- The search input's value will be stored in a signal. Use **debounceTime** (600ms) and **toSignal** to update the value like we've seen in class.
    - Reactive forms (FormControl)

    ◦ Template forms (ngForm)
- The order value (distance, price, date) will be in a signal also (change it with the corresponding buttons.
- The page to load will be also be a signal
- Use the effect function and create dependencies reading the values of the 3 signals above. After that call the corresponding service to load the events based on these search params.

In both approaches, if the page === 1 replace the events array with the result, and if the page is > 1, concatenate the events returned from the server to the array.

Don't use the new Angular 19 rxResource API, because it can't concatenate the results (page > 1) to the existing events array. It will always replace it.

events/:id

Like in unit 1 project, this page will show the details of a event (:id). Deleting the event from here will redirect to **/events.**

Show the map inside a card. The card header will be the event's address.

It's now mandatory to show the list of users attending the event. When the user clicks the button and changes his/her attending choice, the list will be downloaded again and replaced.

It's also mandatory to implement the possibility to comment an event and showing the comments in this page. The HTML for the card with the list of comments and the comment template is commented in the first project (event-detail.html). Add this CSS:

```css
.user-info {
  width: 8rem;
  .avatar {
    width: 4rem;
  }
}

.comment {
  white-space: pre-wrap;
}
```

Also show a form to create a new comment. This form will only have an input for the comment and a button to send it (POST → **/events/:id/comments**) with this format:

```
{ comment: "User comment" }
```

The form can be something like this:

```html
<form class="mt-4">
  <div class="form-group">
    <textarea class="form-control" name="comment" placeholder="Write a comment"></textarea>
  </div>
  <button type="submit" class="btn btn-primary mt-3">Send</button>
</form>
```

The server will return the inserted comment (test the service in Postman). Add this comment to the list without reloading it.

events/add

This page will contain the form to create a new event. Like in exercise 4, validate the form and don't let the user send it until it's valid.

Also validate the form and don't let the user send it until it's valid. **Important**: show feedback to the user (colors, messages) so he/she knows what's valid or not. Like in the unit 1 project, show an input to search for an address and the map with the coordinates where the event will take place.

posts/:id/edit

Will edit a event. You must reuse the component to add an event (**event-form**). For example, if you don't receive an id, add a new event. But, if you receive an id, edit the event (showing the current info in the inputs). Also change the submit button text.

Don't worry if the current address doesn't appear in the map's search input. You can show the current address in a paragraph above the map, for example.

profile/me – profile/:id

Like in unit 1 project, this page will show an user's profile information. If you don't receive an id, show the current logged user, otherwise show the user with the id. Both routes will reuse the same component (**profile-page**).

Show also a map (and a marker) with the user coordinates.

Show the edit (image, profile, password) buttons only if the profile is yours (logged user).

In this component, put also 2 links that will show the following:

- Events this user has created → Will go to the **/events** page but sending a query param named **creator** → Example: /events?creator=49
- Events this user attends to → Will go to the **/events** page but sending a query param named **attending** → Example: /events?attending=49

To send query parameters to a route in a link, use the queryParams attribute:

```
<a [routerLink]="['/events']" [queryParams]="{ creator: user.id }">Created events</a>
```

Use **input** (one for each query param) in the events-page component to get the value. Keep in mind that this value is optional (maybe it's not present). In the effect function (constructor) where you load the events, include these 2 values and call the corresponding method to load the events based on these options.

Add the corresponding text that indicates what the user is seeing at below the search/order bar. Examples:

- Events created by Pepito. Filtered by party. Ordered by price.
- Events attended by Juana. Filtered by birth. Ordered by distance.

In order to get and show the user's name (you only have the id), you can call the service that returns the user profile information.

Resolvers

Event detail, event edit and **profile** pages must get the basic data with a Resolver before showing the page (you've already done that with the event detail page in previous exercises).

Interceptors

You must use an interceptor to insert the authentication token in every HTTP request to the server (See here for more details), and another to set the base url for the server (already done in past exercises).

Services

Auth service → This service will manage all operations related to login / register.

Events service → All operations related with events (including attending and comments).

Profile service → Operations related with users (profile).

Authentication

- **AuthService** → This service will perform the login (storing the authentication token) and logout (removing the token) actions, and will contain the following attributes and methods:
    - **#logged: WritableSignal<boolean>** → By default false. Will indicate if the user is logged in or not. Create a getter that returns this signal in read-only mode.
    - **login(data: UserLogin): Observable<void>** → Will check the login against the server. If login goes ok, in the **map** function, save the token in the **Local Storage** and set **logged** to true.
    - **Logout(): void** → This method will remove the token from the **Local Storage**, set **this.logged** to false.
    - **isLogged(): Observable<boolean>.**
        - If the **this.logged** property is false and there's no token in Local Storage, return Observable<false> → **of(false)**. Import the of function from 'rxjs', it returns an observable with that value.
        - If the **this.logged** property is true, return Observable<true> → **of(true)**
        - But if it's false and there's a token, return the call to the **auth/validate** service (Observable). Inside the pipe method:
            - If there's no error (**map** function), change **this.#logged** to true and return **true**
            - If there's an error (**catchError** function), remove the token from local storage (not valid), and return **of(false)**. The catchError function must return the value inside an observable.
    - **Other methods** → Implement other methods for user registration, login with Google (send credentials with lat and lng) and Facebook (send accessToken with lat and lng). If you think you need to implement anything else, do it.

Showing/hiding menu when login/logut

In the Menu component, show only the menu links when the user is not logged. Create a **computed signal** with the same value as the logged signal in the AuthService service.

The logout functionality is also handled by this component. Call **AuthService.logout()** to remove the token.

Guards

Create 2 guards for controlling authenticated routes:

- **LoginActivateGuard** → Use it in every route except for auth routes. Will return the call to **AuthService.isLogged()**. In the map function:
  - if the user is not logged (false), return a redirection (urlTree) to '**/auth/login**' page.
  - If the user is logged (true), return true. You can go to the route.
- **LogoutActivateGuard** → Use it only for the **auth/** routes. Will return the call to **AuthService.isLogged()**. In the map function:
  - if the user is logged (true), return a redirection (urlTree) to '**/posts**'.
  - If the user is not logged (false), return true. You can go to the route.

Also, use the leavePageGuard with 'posts/add', 'posts/edit/:id' and '/auth/register' routes. Only ask the user if the he/she has changed something in the form (dirty), or if the data has not been saved yet. Use ngBootstrap modal to ask the user.

Marks

Compulsory part

- Register and login, including Google and Facebook login (2 points)
- Showing, filtering and ordering events, including the option to show only events that a user has created *'/posts?creator=24'* or attends *'/posts?attending=24'* (2 points)
- Event cards are correct, updates the attend button (and number), and have the delete and edit buttons when necessary, and work (0,5 points).
- Event details, including attending users and comments (2 points)
- Creating / updating an event (1 points)
- Show and edit profile (1,5 points)
- Code, structure, good programming practices, reusing components

(event-card, event-form, ...). Routes, interceptors, resolvers and guards work ok. Using signals API and reactive functions and values (computed, effect, ...) (1 point)

  - You must use modern Angular features
  - Icons should use angular-fontawesome, not the JavaScript library.

Optional

This parts add extra points. Only valid when the base mark is at least **7.5**. Maximum mark in the project is 12.

- Using Reactive Forms instead of Template Forms (0.5 points).
- Updating product list (order, page, search, creator, attending) in a reactive way (using effect) (0.25 points)
- Load attending people and comments (event-detail) using rxResource (0.25 points)
- Buttons for adding a new event, deleting an event, register or login will show an animation when processing the request (use the load-button component we saw in content projection and update it if necessary -> to enable type="submit" for example). (0.25 points)
- Using a library to show modals like ngBootstrap, and use it for feedback in forms, deleting an event, etc. (0.25 points)
- Create a server-side rendered app that works (no errors or warnings) with client hydration, cookies, etc.. (up to 0.5 points)
  - Login and register must be prerendered routes
  - Everything else must be rendered in the server
- Add animations, including transitions between routes with Angular Animations (up to 0.5 points)
- Use de ngx-image-cropper library to crop images before sending them to the server.
  - Like in unit 1 project the event's image will have an aspect ratio of 16/10 with a width of 1024px, and the user avatar image’s aspect ratio is 1 and its width should be 200px. Send the image in jpg format. (0.5 points)