UNIDAD 2

Angular

Ejercicio final

Desarrollo web del lado del cliente
2º curso - DAW IES
San Vicente 2024/2025
Autor: Arturo Bernal Mayordomo

Índice

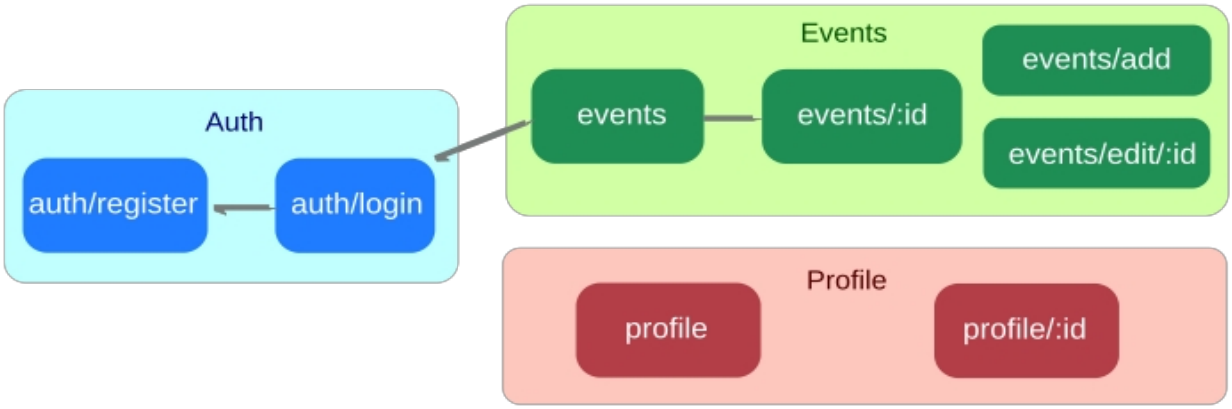
SVTickets (Versión Angular)

Este ejercicio final será una ampliación de los ejercicios que hemos estado haciendo durante esta unidad. Utiliza el ejercicio de la semana 11 como punto de partida.

Los servicios web son los mismos que en la unidad 1, con algunos añadidos como el inicio de sesión en Google o Facebook -> https://github.com/arturober/svtickets-services

Rutas

La aplicación tendrá las siguientes rutas:



Rutas no autenticadas

auth/register

Esta página contendrá un formulario para que el usuario se registre. Cree el mismo formulario utilizado en el proyecto de la unidad 1 y **valídelo** en el lado del cliente: todos los campos son obligatorios, los campos de correo electrónico deben ser de tipo correo electrónico, y la contraseña debe tener al menos 4 caracteres .

Además, crea un validador que valide que ambos emails son iguales. Pon el mensaje de error para este validador debajo de la entrada "repetir email" (con la clase css correspondiente para esa entrada). Puedes crear un validador de grupo para comprobar ambos valores, o simplemente crear un validador normal que pongas en el segundo campo de email y que reciba el primer campo de email como valor de entrada.

No olvides geolocalizar al usuario y enviar las coordenadas al servidor.

auth/login

Esta página contendrá una página con un formulario para iniciar sesión (correo electrónico, contraseña) y

los botones de Google y Facebook para iniciar sesión/registrarse. Valida el formulario (todos los campos son obligatorios).

También geolocalizar al usuario aquí y enviar las coordenadas con la información de inicio de sesión (lat, lng). Esto incluye el inicio de sesión normal y también el inicio de sesión de Google/Facebook.

Rutas autenticadas

eventos

Esta página mostrará todos los mensajes (como en ejercicios anteriores). Utilice la misma tarjeta que utilizó en el proyecto de la unidad 1.

El botón para asistir a un evento (**/events/:id/attend**) es ahora obligatorio y debe funcionar actualizando el número de asistentes (y el booleano attend), así como el color del texto y el icono.

En la tarjeta (**importante**→ si el evento es suyo):

- Pon el botón para borrar el post (esto mostrará un diálogo de confirmación preguntando si estás seguro de querer borrarlo usando ngBootstrap o ngx-sweetalert2).

- Pon un botón para editar el post (debajo de la descripción por ejemplo). El botón de edición irá a **/event/:id/edit**. Puede ser un enlace con las clases btn. Ejemplo: Editar.

Filtrar los eventos

Debes filtrar y ordenar los eventos como hiciste en el proyecto TypeScript.
Puedes elegir 2 formas de hacerlo:

Tradicional

La búsqueda y el pedido, y la página siguiente se activarán mediante eventos de clic. Los eventos llamarán a métodos que cambian el valor de búsqueda, página o pedido, y luego llamarán a otro método para cargar eventos basados en estos valores

Reactivo (+0,25 puntos)

- El valor de la entrada de búsqueda se almacenará en una señal. Utilice **debounceTime** (600ms) y **toSignal** para actualizar el valor como hemos visto en clase.
    - Formularios reactivos (FormControl)

- Plantillas de formularios (ngForm)

- El valor de la orden (distancia, precio, fecha) estará también en una señal (con los botones correspondientes.
- La página a cargar será también una señal
- Utilice la función de efecto y cree dependencias leyendo los valores del 3 señales anteriores. Después se llama al servicio correspondiente para cargar los eventos basados en estos parámetros de búsqueda.

En ambos enfoques, si la página=== 1 sustituye el array de eventos por el resultado, y si la página es> 1, concatena los eventos devueltos desde el servidor al array.

No uses la nueva API rxResource de Angular 19, porque no puede concatenar los resultados (página> 1) al array de eventos existente. Siempre lo reemplazará.

eventos/:id

Al igual que en el proyecto de la unidad 1, esta página mostrará los detalles de un evento (:id).
Borrar el evento desde aquí redirigirá a **/events.**

Mostrar el mapa dentro de una tarjeta. La cabecera de la tarjeta será la dirección del evento.

Ahora es obligatorio mostrar la lista de usuarios asistentes al evento. Cuando el usuario haga clic en el botón y cambie su elección de asistente, la lista se descargará de nuevo y se sustituirá.

También es obligatorio implementar la posibilidad de comentar un evento y mostrar los comentarios en esta página. El HTML para la tarjeta con la lista de comentarios y la plantilla de comentarios está comentado en el primer proyecto (event- detail.html). Añade este CSS:

```
.user-info {
   ancho: 8rem;
   .avatar {
     anchura: 4rem;
   }
}

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

También mostrar un formulario para crear un nuevo comentario. Este formulario sólo tendrá un
entrada para el comentario y un botón para enviarlo (POST→ **/events/:id/comments**)

con este formato:

```
{ comentario: " del usuario" }
```

El formulario puede ser algo así

```
<formulario clase= "mt-4">
  <div class= "grupo-forma">
    <textarea class= "form-control" name= "comment" placeholder= "Escribe un comentario"></textarea>
  </div>
  <tipo de botón= "submit" clase= "btn btn-primary mt-3"> Enviar</button>
</form>
```

El servidor devolverá el comentario insertado (prueba el servicio en Postman). Añade este comentario a la lista sin .

eventos/añadir

Esta página contendrá el formulario para crear un nuevo evento. Como en el ejercicio 4, valida el formulario y no dejes que el usuario lo envíe hasta que sea válido.

Valida también el formulario y no dejes que el usuario lo envíe hasta que sea válido. **Importante**: muestra feedback al usuario (colores, mensajes) para que sepa es válido o no. Como en el proyecto de la unidad 1, muestra una entrada para buscar una dirección y el mapa con las coordenadas donde tendrá el evento.

posts/:id/edit

Editará un . Debe reutilizar el componente para añadir un evento (**evento-formulario**). Por ejemplo, si no recibes un id, añade un nuevo . Pero, recibes un id, edita el evento (mostrando la información actual en los inputs). También cambia el texto del botón de enviar.

No se preocupe si la dirección actual no aparece en la entrada de búsqueda del mapa. Puedes mostrar la dirección actual en un párrafo encima del , por ejemplo.

perfil/me - perfil/:id

Como en el proyecto de la unidad 1, esta página mostrará la información del perfil de un . Si no recibe un id, mostrará el usuario actual registrado, de lo contrario mostrará el usuario con el id. Ambas rutas reutilizarán el mismo componente (**profile-page**).

Mostrar también un mapa (y un marcador) con las coordenadas del usuario.

Mostrar los botones de edición (imagen, perfil, contraseña) sólo si el perfil es tuyo (usuario registrado).

En este , ponga también 2 enlaces que mostrarán lo siguiente:

- Eventos que este usuario ha creado→ Irá a la página **/eventos** pero enviando un parámetro de consulta denominado **creador**→ Ejemplo: /eventos?creador=49
- Eventos a los que asiste este usuario a→ Irá a la página **/eventos** pero enviando un parámetro de consulta denominado **asistentes**→ Ejemplo: /eventos?asistentes=49

Para enviar parámetros de consulta a una ruta en un enlace, utilice el atributo queryParams:

```
<a [routerLink]="['/eventos']" [queryParams]="{ creador: user.id }"> Eventos creados</a>
```

Utilice la **entrada (**una para cada parámetro de consulta) en el componente events-page para obtener el . Ten en cuenta que este valor es opcional (puede que no esté presente). En la función effect (constructor) donde cargas los eventos, incluye estos 2 valores y llama al método correspondiente para cargar los eventos en base a estas .

Añada el texto correspondiente que indica lo que el usuario está viendo debajo de la barra de búsqueda/pedido. Ejemplos:

- Eventos creados por Pepito. Filtrados por partido. Ordenados por precio.
- Eventos a los que ha asistido Juana. Filtrados por nacimiento. Ordenados por distancia.

Para obtener y mostrar el nombre del usuario (sólo tienes el id), puedes llamar al servicio que devuelve la información del perfil del usuario.

Resolvedores

Las páginas de detalles del **evento, edición del evento** y **perfil** deben obtener los datos básicos con un Resolver antes de mostrar la página (ya lo has hecho con la página de detalles del evento en ejercicios anteriores).

Interceptores

Debes usar un interceptor para insertar el token de autenticación en cada petición HTTP al servidor (Ver aquí para más detalles), y otro para establecer la url base para el servidor (ya hecho en ejercicios anteriores).

Servicios

Auth service→ Este servicio gestionará todas las operaciones relacionadas con el inicio de sesión / registro.

Servicio de eventos→ Todas las operaciones relacionadas con eventos (incluyendo asistencia y comentarios).

Servicio de perfil→ Operaciones relacionadas con los usuarios (perfil).

Autenticación

- **AuthService** → Este servicio se realiza el inicio de sesión (almacenando la dirección autenticación) y logout (eliminación del token) y contendrá los siguientes atributos y métodos:

  - **#logged: WritableSignal<boolean>**→ Por defecto false. Indicará si el usuario está logueado o no. Crea un getter que devuelva esta señal en modo solo lectura.
  - **login(data: UserLogin): Observable<void>** → Comprobará el login contra el servidor. Si el login va bien, en la función **map**, guarda el token **en el Local Storage** y establece **logged** a true.
  - **Cerrar sesión():** ← Este método eliminará el token del **Local Storage**, establece **this.logged** como false. **void**
  - **isLogged(): Observable<boolean>.**
    - Si la propiedad **this.logged** es false y no hay ningún token en Local Storage, devuelve Observable<false>→ **of(false)**. Importa la función of de <rxjs>, devuelve un observable con ese .
    - Si la propiedad **this.logged** es true, devuelve Observable<true>. → **de(verdadero)**
    - Pero si es falso y hay a ficha, devuelve la llamada ala página servicio **auth/validate** (Observable). Dentro del método pipe:
      - Si no hay error (función **map**), cambia **this.#logged** a true y devuelve **true**
      - Si hay un error (función **catchError**), elimina el token del almacenamiento local (no válido), y devuelve **of(false)**. La función catchError debe devolver el valor dentro de un observable.
  - **Otros métodos**→ Implementar otros métodos para el registro de usuarios, login con Google (enviar credenciales con lat y lng) y Facebook (enviar accessToken con lat y lng). Si crees que necesitas implementar algo más, .

Mostrar/ocultar menú al iniciar sesión/cierre de sesión

En el componente Menú, muestre sólo los enlaces del menú cuando el usuario no haya iniciado sesión. Cree una **señal computada** con el mismo valor que la señal registrada en el servicio AuthService.

La página gestiona cierre de sesión en funcionalidad este componente es también componente. Llame a **AuthService.logout()** para eliminar el token.

Guardias

Crear 2 guardias para controlar las rutas autenticadas:

- **LoginActivateGuard** → Utilícelo en todas las rutas excepto en rutas de autenticación. Will devuelve la llamada a **AuthService.isLogged()**. En la función map:
  - si el usuario no ha iniciado sesión (false), devuelve una redirección (urlTree) a la página '**/auth/login**'.
  - Si el usuario está registrado (true), devuelve true. Puede ir a la ruta.
- **LogoutActivateGuard**→ Úsalo sólo para las rutas **auth/**. Devolverá la llamada a **AuthService.isLogged()**. En la función map:
  - si el usuario está registrado (true), devuelve una redirección (urlTree) a ' **/posts**'.
  - Si el usuario no está registrado (false), devuelve true. Puede ir a la ruta.

Utiliza también el leavePageGuard con las rutas 'posts/add', 'posts/edit/:id' y '/auth/register'. Sólo pregunta al usuario si ha cambiado algo en el formulario (dirty), o si los datos no se han guardado todavía. Usa ngBootstrap modal para preguntar al usuario.

Marcas

Parte obligatoria

- Registro e inicio de sesión, incluido el inicio de sesión en Google y Facebook (2 puntos)
- Mostrar, filtrar y ordenar eventos, incluyendo la opción de mostrar sólo los eventos que un usuario ha creado *'/posts?creator=24'* o a los que asiste */posts? attending=24'* (2 puntos)
- Las tarjetas de eventos son correctas, actualizan el botón de asistir (y el número), y tienen los botones de borrar y editar cuando es necesario, y funcionan (0,5 puntos).
- Detalles del acto, incluidos los usuarios asistentes y sus comentarios (2 puntos)
- Crear / actualizar un evento (1 puntos)
- Mostrar y editar perfil (1,5 puntos)
- Código, estructura, buena programación programación, reutilización de componentes en componentes

(event-card, event-form, ...). Routes, interceptors, resolvers y guards funcionan bien. Uso de señales API y funciones reactivas y valores (computed, effect, ...) (1 punto)

- Debe utilizar funciones modernas de Angular
- Los iconos deben usar angular-fontawesome, no la librería JavaScript.

Opcional

Estas piezas añaden puntos extra. Sólo es válido cuando la nota base es de al menos **7,5**. La nota máxima del proyecto es 12.

- Uso de formularios reactivos en lugar de formularios de plantilla (0,5 puntos).
- Actualización de la lista de productos (pedido, página, búsqueda, creador, asistente) de forma reactiva (mediante efecto) (0,25 puntos)
- Cargar personas asistentes y comentarios (evento-detalle) usando rxResource (0.25 puntos)
- botones para añadir un nuevo evento, borrar un , registrarse o iniciar sesión mostrarán una animación al procesar la petición (usa el componente load-button que vimos en la proyección de contenido y actualízalo si es necesario -> para habilitar type="submit" por ejemplo). (0.25 puntos)
- Usar una librería para mostrar modals como ngBootstrap, y usarla para feedback en formularios, borrar un evento, etc. (0.25 puntos)
- Crear una aplicación renderizada del lado del servidor que funcione (sin errores ni advertencias) con hidratación del cliente, cookies, etc. (hasta 0,5 puntos)
  - El inicio de sesión y el registro deben ser rutas prerrenderizadas
  - Todo lo demás debe renderizarse en el servidor
- Añade animaciones, incluyendo transiciones entre rutas con Angular Animations (hasta 0.5 puntos)
- Utilice la biblioteca ngx-image-cropper para recortar imágenes antes de enviarlas al servidor.
  - Al igual que en el proyecto de la unidad 1, la imagen del evento tendrá una relación de aspecto de 16/10 con una anchura de 1024px, y la relación de aspecto de la imagen del avatar del usuario es 1 y su anchura debe ser de 200px. Envíe la imagen en formato jpg. (0,5 puntos)