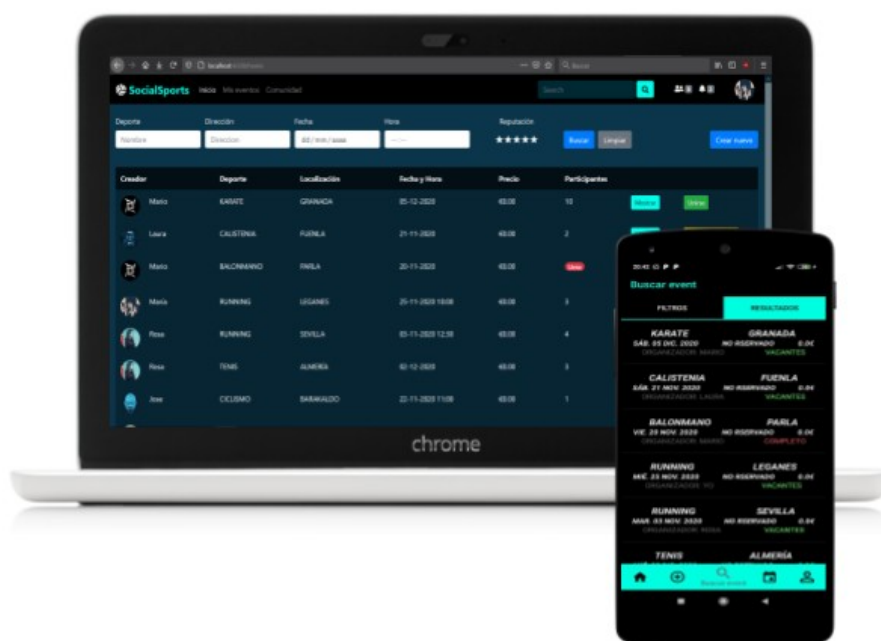


SOCIALSPORTS 2.0

Desarrollo de Aplicaciones Multiplataforma
IES Gaspar Melchor de Jovellanos



Autora:
Laura Patón Campos

Profesora coordinadora:
Nina Castro Cintas

Índice

| | |
|--|----|
| • Resumen | 3 |
| • Abstract | 3 |
| • Datos técnicos | 4 |
| ◦ Nombre de la aplicación | 4 |
| ◦ Lenguajes de programación | 4 |
| ◦ Manual de instalación | 4 |
| • Funcionalidad | 6 |
| ◦ Diagrama E/R | 6 |
| ◦ Diagramas de clases | 7 |
| • Usabilidad | 8 |
| ◦ Diseño de interfaces | 8 |
| • Tecnologías | 13 |
| ◦ Servicio web RESTful | 13 |
| ◦ Spring | 13 |
| ▪ Introducción | 13 |
| ▪ Inversión de control e Inyección de dependencias | 14 |
| ▪ Spring Boot | 14 |
| ▪ JPA / Hibernate / Spring Data JPA | 15 |
| • JPA | 15 |
| • Hibernate | 15 |
| • Spring Data JPA | 15 |
| • Entidades | 16 |
| • Repositorios | 17 |
| ▪ Servicios | 17 |
| ▪ Controladores | 17 |
| ▪ Spring Validation | 18 |
| ◦ JWT | 18 |
| ◦ Angular | 19 |
| ◦ Bootstrap | 20 |
| • Despliegue | 20 |
| ◦ Heroku | 20 |
| ◦ Firebase Hosting | 21 |
| • Bibliografía | 22 |

RESUMEN

El presente proyecto es una continuación de SocialSports, aplicación móvil orientada a la realización de eventos deportivos.

En esta ocasión contaremos con tres componentes principales; una API REST, una aplicación móvil y una aplicación web, en donde móvil y web se comunicarán entre sí gracias a la API.

Para ello, se ha cambiado en su totalidad la API, dado que la anterior tenía algunos errores y era complicada de mantener, y además, se ha creado una aplicación web con Angular.

ABSTRACT

The present project is a continuation of SocialSports, a mobile app aimed to realize sport events.

This time, we will have three principal components; an API REST, a mobile app and a website, where app and web will communicate each other thanks to the API.

For this, the API have completely changed, since the previous had some errors and it was complicated to maintain, and also, it has been created a webpage with Angular.

DATOS TÉCNICOS

Nombre de la aplicación

SocialSports

Lenguajes de programación

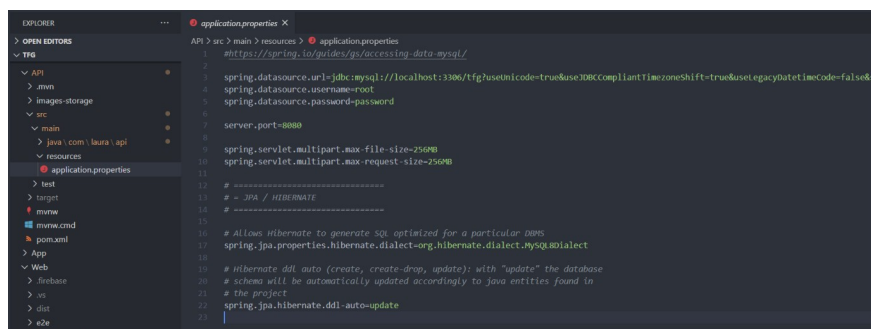
Java y TypeScript

Manual de instalación

Para que sea más fácil, se ha desplegado tanto el servicio RESTful como la aplicación web en hostings gratuitos. Aún así, si se quiere probar en local haría falta lo siguiente:

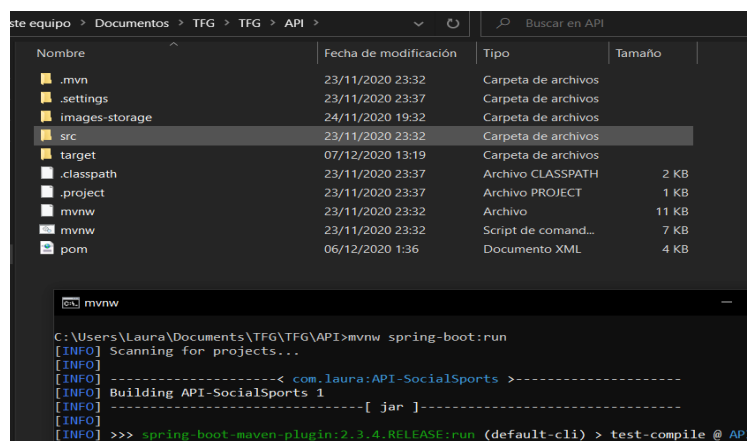
API REST

Primero habría que cambiar la configuración de la base de datos en el archivo `application.properties` (y si se quiere una base de datos relacional distinta a Mysql, también habría que poner la dependencia del driver en el `pom.xml`):



```
1 #application.properties
2 #https://spring.io/guides/gs/accessing-data-mysql/
3
4 spring.datasource.url=jdbc:mysql://localhost:3306/tfg?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&useSSL=true
5 spring.datasource.username=root
6 spring.datasource.password=password
7
8 server.port=8080
9
10 spring.servlet.multipart.max-file-size=250MB
11 spring.servlet.multipart.max-request-size=250MB
12
13 # =====
14 # = JPA / HIBERNATE
15 # =====
16 # Allows Hibernate to generate SQL optimized for a particular DBMS
17 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
18
19 # Hibernate ddl auto (create, create-drop, update): with "update" the database
20 # schema will be automatically updated accordingly to java entities found in
21 # the project
22 spring.jpa.hibernate.ddl-auto=update
23
```

Y una vez configurado esto, para poner todo en funcionamiento solo hace falta irte a la carpeta del proyecto y ejecutar el archivo `mvnw` con el comando: **`mvnw spring-boot:run`**



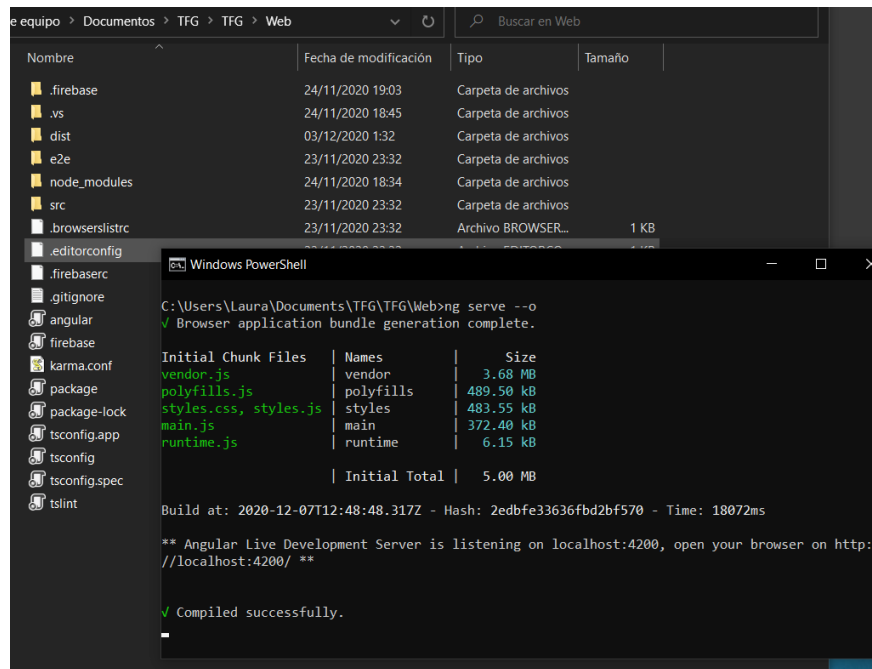
| Nombre | Fecha de modificación | Tipo | Tamaño |
|----------------|-----------------------|---------------------|--------|
| .mvn | 23/11/2020 23:32 | Carpeta de archivos | |
| .settings | 23/11/2020 23:37 | Carpeta de archivos | |
| images-storage | 24/11/2020 19:32 | Carpeta de archivos | |
| src | 23/11/2020 23:32 | Carpeta de archivos | |
| target | 07/12/2020 13:19 | Carpeta de archivos | |
| .classpath | 23/11/2020 23:37 | Archivo CLASSPATH | 2 KB |
| .project | 23/11/2020 23:37 | Archivo PROJECT | 1 KB |
| mvnw | 23/11/2020 23:32 | Archivo | 11 KB |
| mvnw.cmd | 23/11/2020 23:32 | Script de comand... | 7 KB |
| pom.xml | 06/12/2020 1:36 | Documento XML | 4 KB |

```
C:\Users\Laura\Documents\TFG\TFG\API>mvnw spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.laura:API-SocialSports >-----
[INFO] Building API-SocialSports 1
[INFO]
[INFO] -----[ jar ]-----
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.3.4.RELEASE:run (default-cli) > test-compile @ API
```

Aplicación Web

Haría falta tener Angular CLI instalado.

Después en la carpeta del proyecto, éste se ejecutaría con el siguiente comando: **ng serve**



The screenshot shows a Windows File Explorer window with the path `C:\Users\Laura\Documents\TFG\TFG\Web`. The file list includes `.firebase`, `.vs`, `dist`, `e2e`, `node_modules`, `src`, `.browserslistrc`, `.editorconfig`, `.firebaserc`, `.gitignore`, `angular`, `firebase`, `karma.conf`, `package`, `package-lock`, `tsconfig.app`, `tsconfig`, `tsconfig.spec`, and `tslint`.

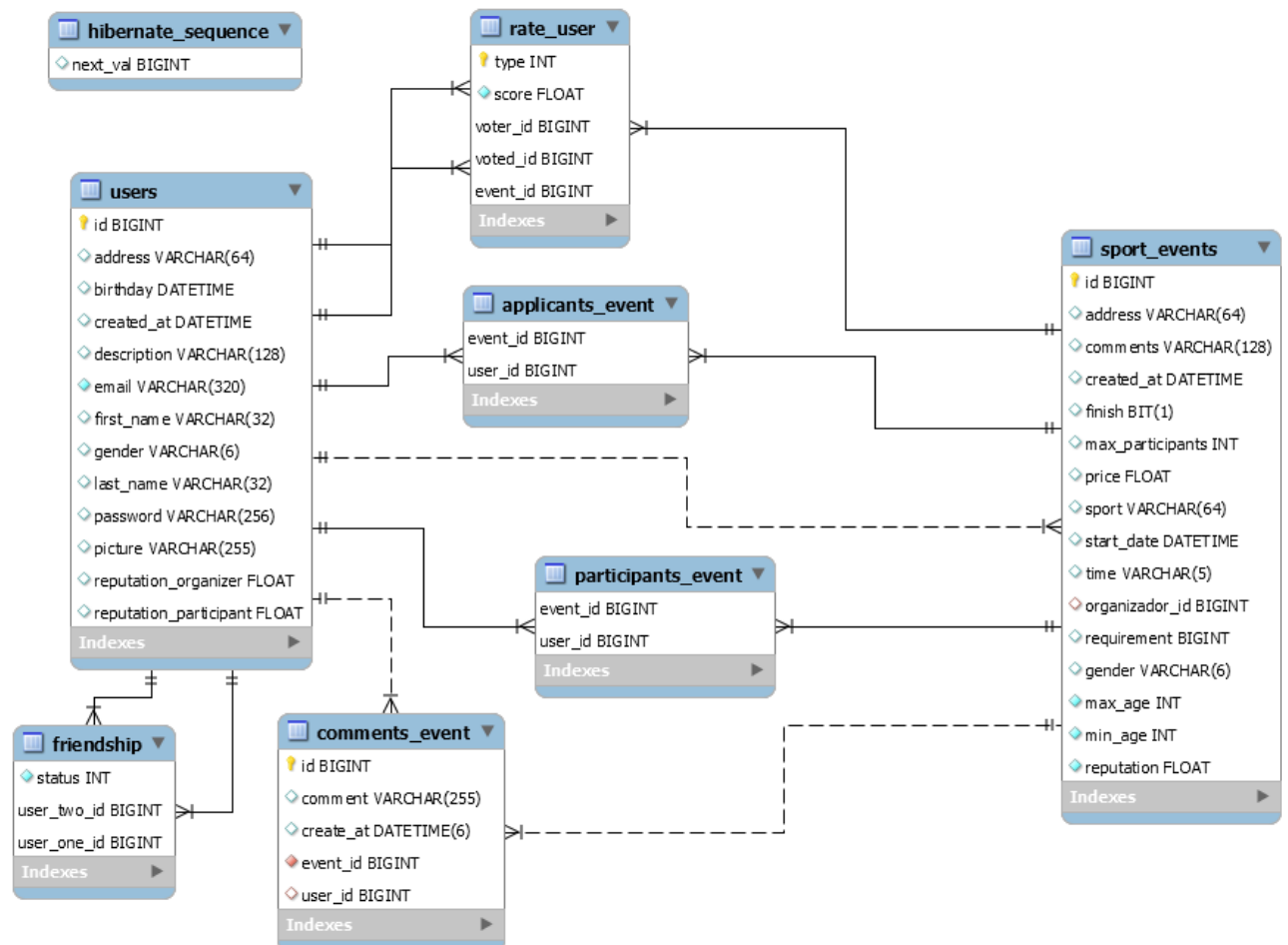
Overlaid on the File Explorer is a Windows PowerShell terminal window. The command `ng serve --o` has been executed. The output shows that the browser application bundle generation is complete. It then displays a table of initial chunk files:

| Initial Chunk Files | Names | Size |
|------------------------------------|------------------------|-----------|
| <code>vendor.js</code> | <code>vendor</code> | 3.68 MB |
| <code>polyfills.js</code> | <code>polyfills</code> | 489.50 kB |
| <code>styles.css, styles.js</code> | <code>styles</code> | 483.55 kB |
| <code>main.js</code> | <code>main</code> | 372.40 kB |
| <code>runtime.js</code> | <code>runtime</code> | 6.15 kB |
| Initial Total | | 5.00 MB |

Below the table, the build information is shown: `Build at: 2020-12-07T12:48:48.317Z - Hash: 2edbf33636fbd2bf570 - Time: 18072ms`. A message indicates that the Angular Live Development Server is listening on `localhost:4200`. The terminal concludes with `Compiled successfully.`

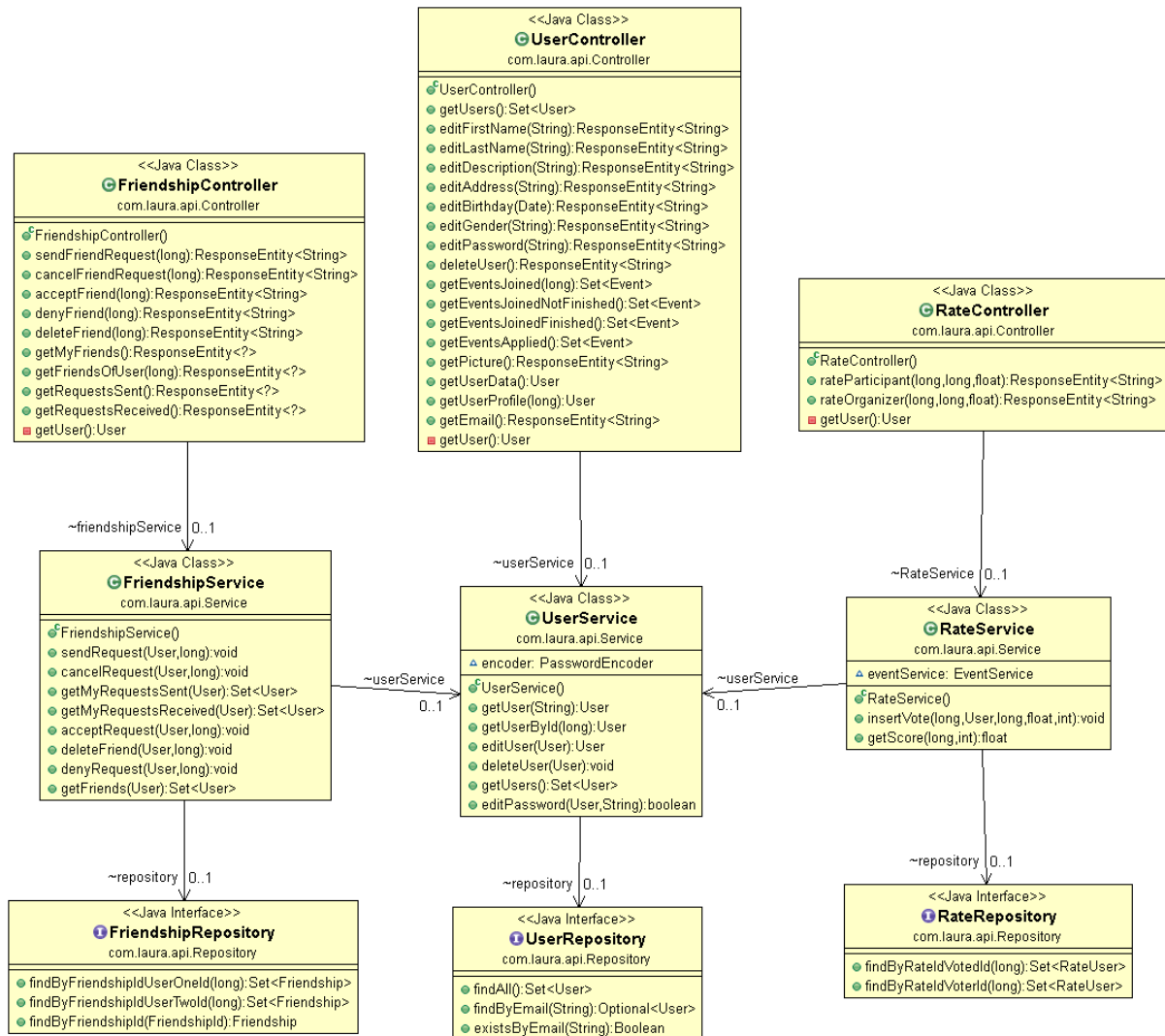
FUNCIONALIDAD

1. Diagrama relacional

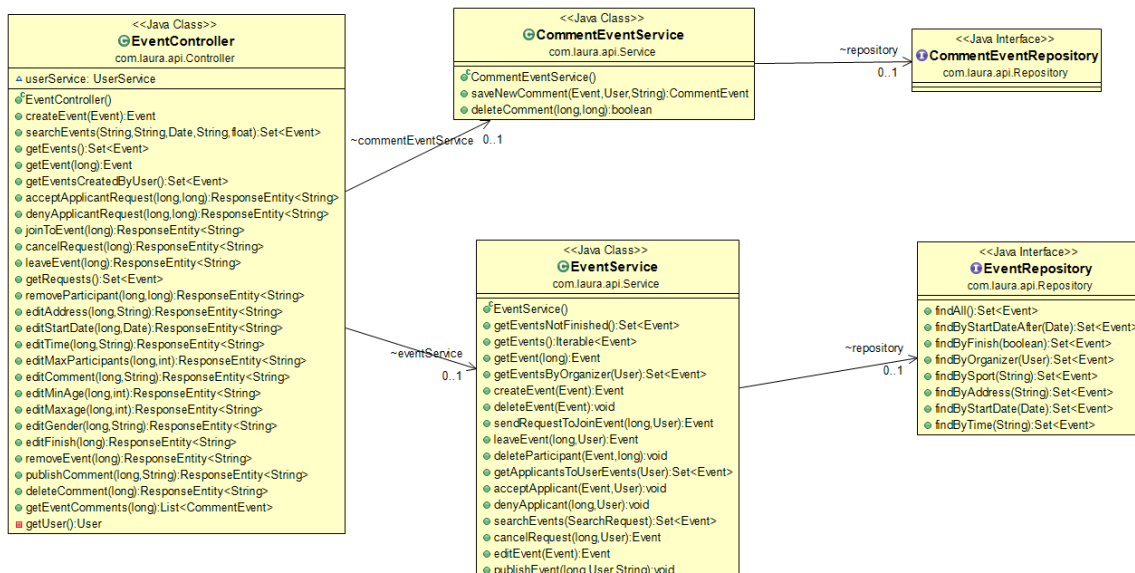


2. Diagramas de clases

2.1 Clases de gestión de usuarios, votación de usuarios y amistad



2.1 Clases de gestión de eventos y comentarios



USABILIDAD

1. Diseño de interfaces

1.1 Inicio de sesión y registro:

The image displays two screenshots of a web application interface, likely for a basketball-related platform, showing the Login and Signup pages.

Login Page:

- Header: Login
- Form fields: Email, Password
- Buttons: Entrar, Registrarse

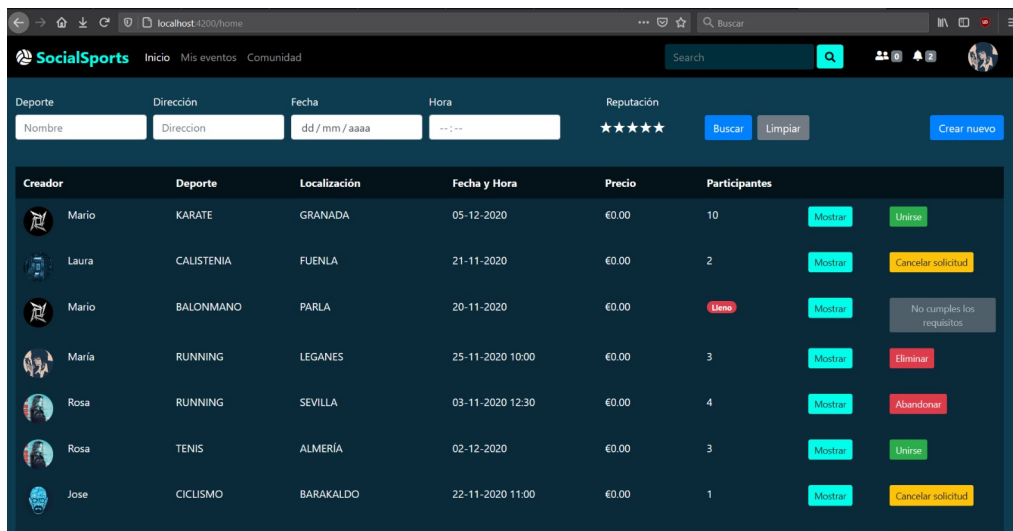
Signup Page:

- Header: Signup
- Form fields: Nombre, Apellidos, Email, Password, Repeat password, Fecha de cumpleaños (dd / mm / aaaa)
- Gender selection: ☐ Mujer ☐ Hombre ☒ Otro
- Buttons: Registrarse, Login

La primera página que aparecerá al abrir la aplicación web será la de inicio de sesión, y si se intentase acceder a otras rutas diferentes sin estar logeado te redireccionaría a esa página otra vez. Igualmente, sin un token válido no se podrá acceder a la mayoría de las uris de la API.

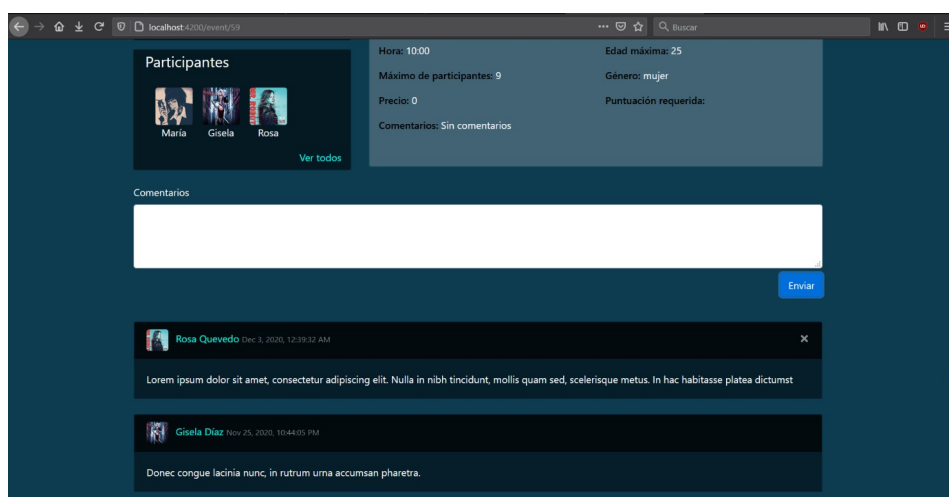
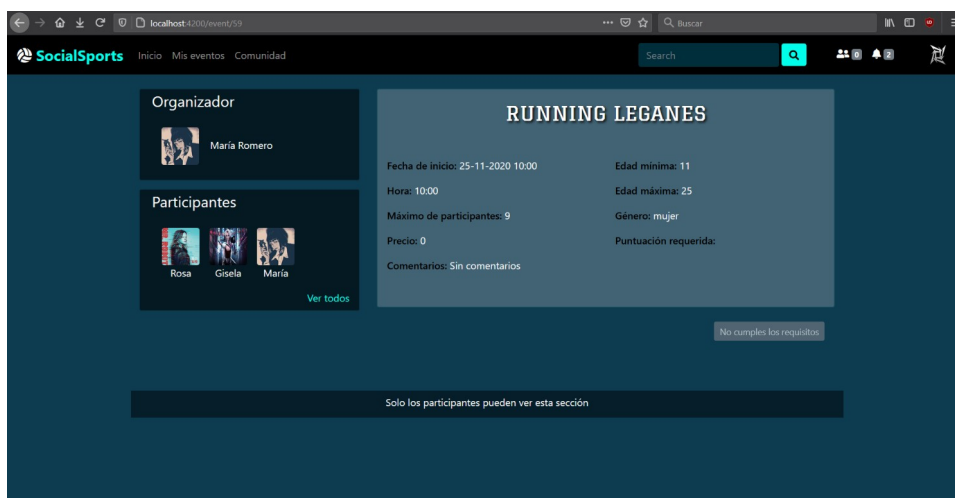
Por otra parte, el login y el registro mostrarán distintos mensajes de error si no se rellenan los campos correctamente.

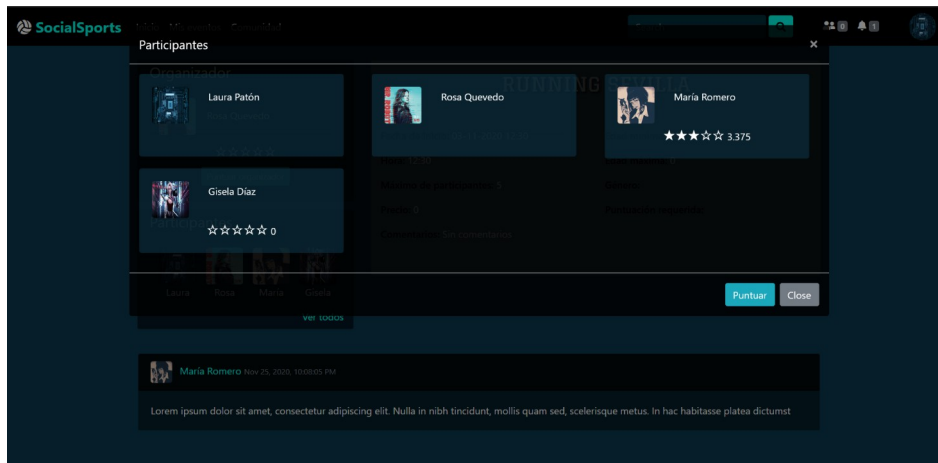
1.2. Eventos



La página de inicio correspondería con el menú de “buscar” de la aplicación móvil. Desde esta página aparecerán todos los eventos existentes no finalizados, en la cuál se podrán realizar las búsquedas y realizar acciones sobre los eventos, como unirse, eliminar un evento que sea tuyo, abandonar, etc..

1.3. Mostrar evento

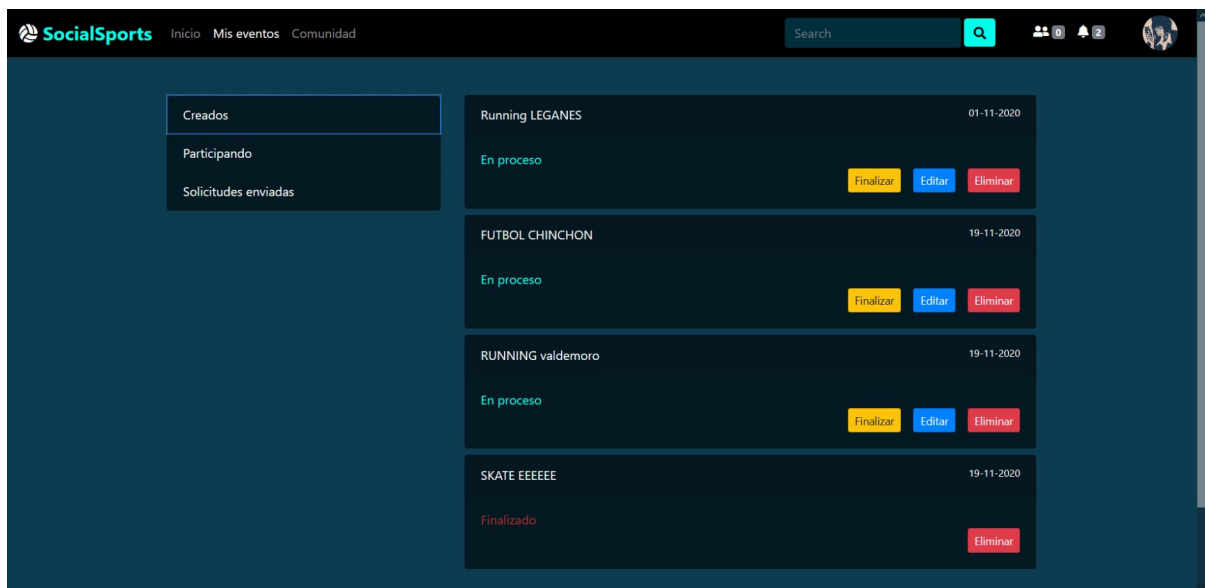




En la página del evento se mostrarán los datos del evento, sus participantes, su creador y los participantes podrán hacer comentarios. Pueden darse varios casos:

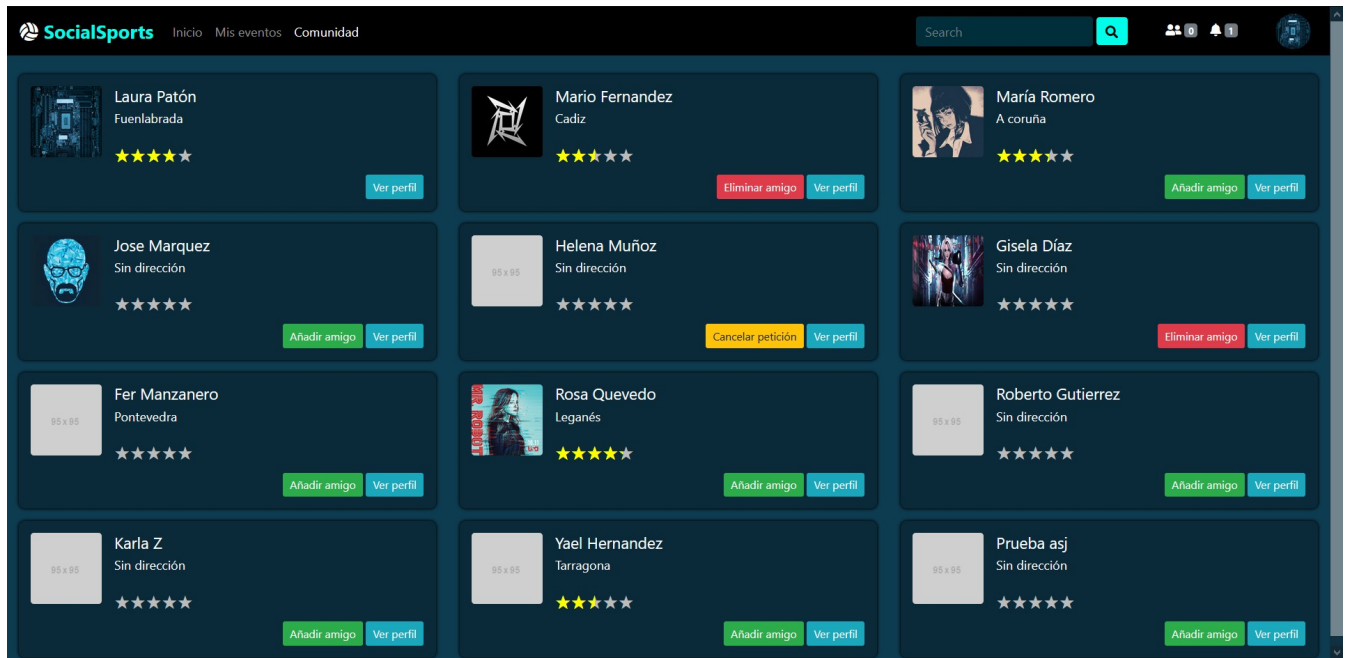
- 1- Si eres el creador del evento también te da la opción de editar los datos de ese evento y poder eliminar participantes.
- 2- Si no estás unido a ese evento no podrás ver los comentarios ni hacerlos.
- 3- Una vez que el evento finaliza, se podrá votar a los participantes y al organizador. (Se puede votar varias veces lo mismo, simplemente se modificaría el voto anterior).

1.4. Mis eventos



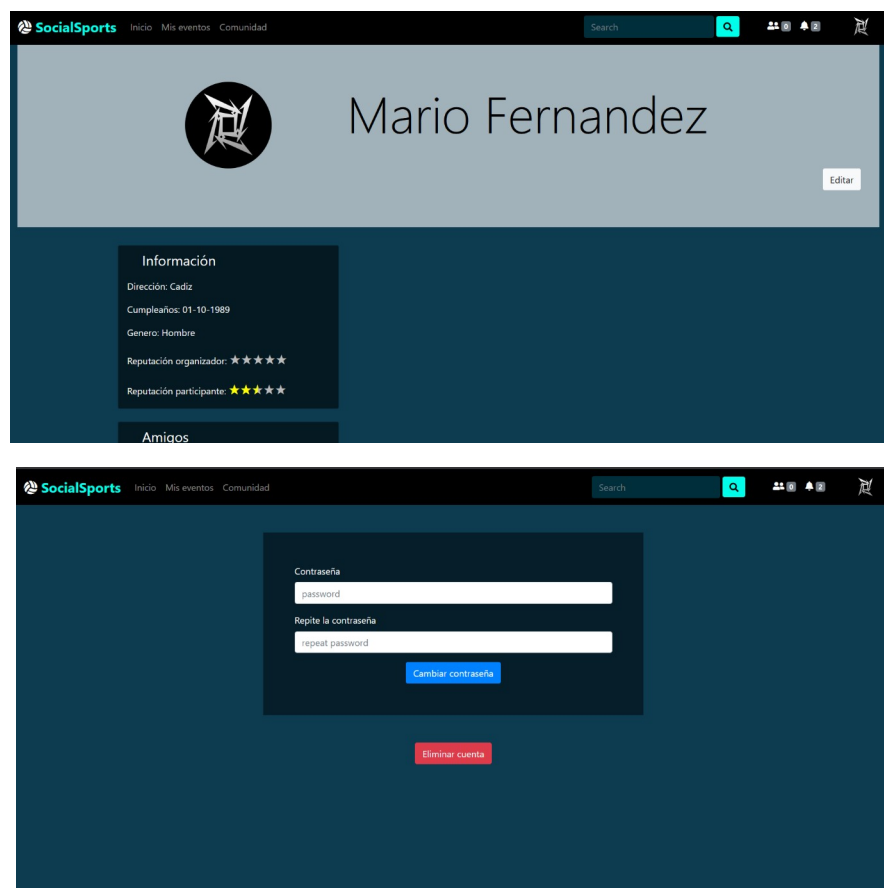
En esta página aparecerán los eventos creados del usuario, en los que está participando y las solicitudes enviadas. Igualmente se podrá mostrar, finalizar, eliminar, abandonar, etc.. dependiendo del evento.

1.5. Comunidad



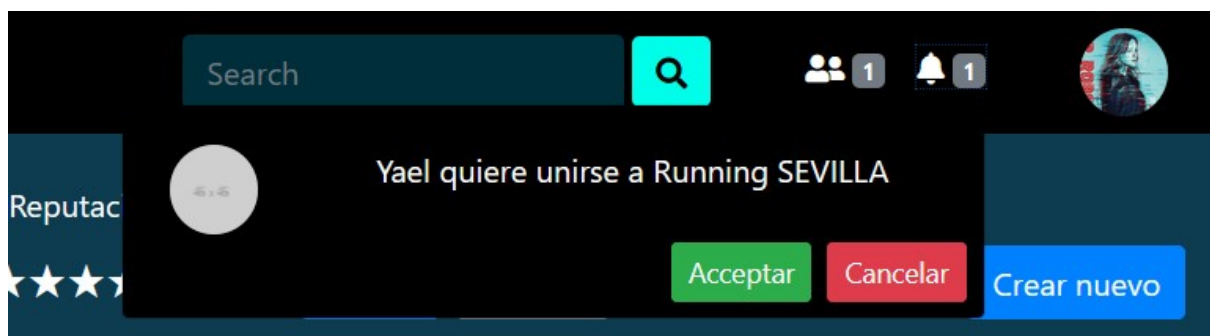
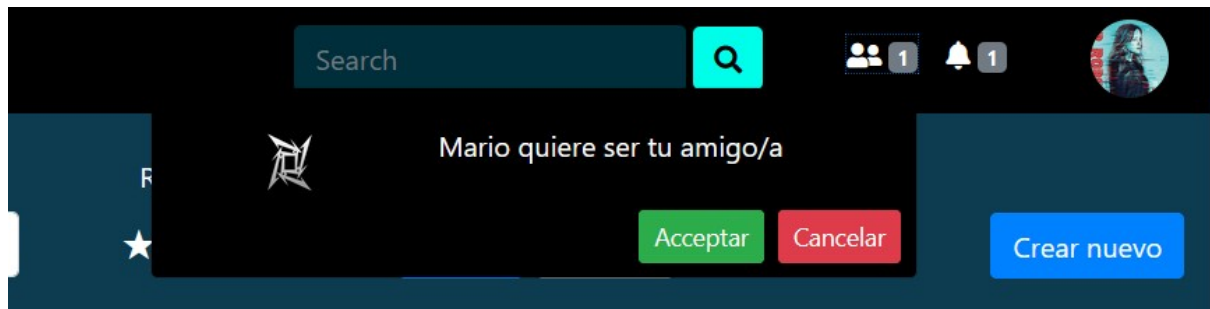
Aquí aparecerán todos los usuarios de la aplicación y se les podrá agregar, aceptar o eliminar en la lista de amigos y ver sus perfiles.

1.6. Perfil y configuración



En el perfil aparecerá la información del usuario y su reputación tanto como participante como organizador además de sus amigos. Si es tu propio perfil también te permitirá modificar tus datos. Por último en la configuración de la cuenta se puede cambiar la contraseña o eliminar la cuenta definitivamente.

1.7 Notificaciones



En los dos iconos de la barra de navegación aparecerán las notificaciones de peticiones de amistad y peticiones de usuarios que quieren unirse a algún evento tuyo.

TECNOLOGÍAS

1. Servicio Web RESTful

REST (Representational State Transfer) es un estilo de arquitectura que proporciona estándares para facilitar la comunicación entre sistemas utilizando el protocolo HTTP.

Los servicios web basados en esta arquitectura se denominan servicios web RESTful.

Las llamadas a la API se hacen como peticiones HTTP, en las que:

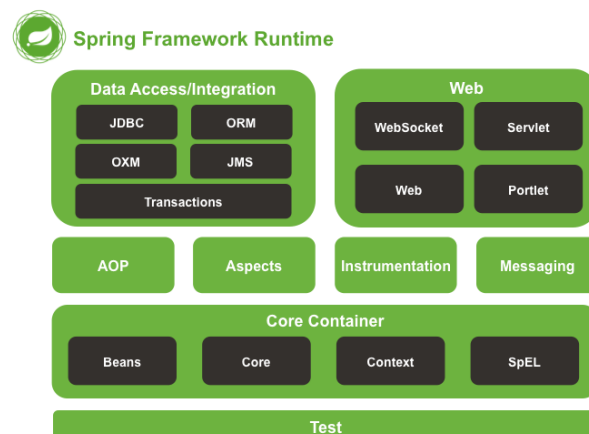
- La URI representa el recurso que se quiere manejar:
http://localhost:8080/api/event/list
- Los métodos indican el tipo de operación a realizar:
 - **GET** (Leer)
 - **POST** (Crear)
 - **PUT** (Modificar)
 - **DELETE** (Borrar)
- Códigos de respuesta:
 - **1xx**: Respuesta informativa
 - **2xx**: Éxito
 - **3xx**: Redirecciones
 - **4xx**: Errores del cliente
 - **5xx**: Errores del servidor

Además HTTP es un protocolo sin estado, es decir, cada petición es independiente y no tiene relación con las solicitudes anteriores. Al ser sin estado, no se recuerda la sesión de un usuario, y hace falta un mecanismo para autenticar al usuario en cada petición. Para ello, una vez el usuario haya iniciado sesión con sus credenciales, recibirá un token que se enviará en las siguientes peticiones.

2. Spring Framework

Spring es un framework de código abierto para el desarrollo de aplicaciones Java.

En la siguiente imagen podemos ver los diferentes módulos que conforman a día de hoy Spring:



Dos conceptos fundamentales en Spring son: la IoC (inversión de control) y la inyección de dependencias.

Inversión de control

La inversión de control (IoC) es un principio de ingeniería de software mediante el cual el control de los objetos se transfiere a un contenedor. Éste será el encargado de controlar el flujo de la aplicación. El contenedor de Spring es responsable de crear instancias y configurar objetos conocidos como beans, así como de administrar su ciclo de vida.

Inyección de dependencias

Es un patrón de diseño mediante el cual se implementa IoC. En lugar de que cada clase tenga que instanciar los objetos que necesite, será Spring el que inyecte esos objetos.

2.1 Spring Boot

Spring boot facilita la creación de aplicaciones basadas en Spring evitando que tengas que pararte en la configuración de sus módulos. Aplica el concepto de Convention over configuration (CoC).

Con Spring inicializarse puede crear fácilmente la estructura que tendrá un proyecto de Spring Boot: <https://start.spring.io/>

Para la configuración de este proyecto se ha escogido un proyecto de tipo Maven, con la versión 2.4 de Spring Boot y Java 8.

Las dependencias que se han usado son las siguientes:

- **Spring Data JPA:** Nos permite simplificar el acceso y persistencia de los datos.
- **Spring Security:** Se encarga de proporcionar autenticación, autorización y protección contra ataques comunes. También proporciona un mecanismo muy sencillo para hashear las contraseñas (en este proyecto se está usando Bcrypt).
- **Spring Validation:** Para validar datos.
- **Spring Web:** Para crear servicios web RESTful (trae un tomcat embebido por defecto).
- **Spring Devtools:** Para no tener que reiniciar la aplicación cuando se hace algún cambio en desarrollo.
- **JWT:** Java Json Web Token
- **Swagger:** Para la documentación.
- El driver de Mysql

1.1.1 JPA / HIBERNATE / SPRING DATA JPA

- **JPA**

JPA (API de persistencia de Java) proporciona la especificación para el mapeo objeto-relacional. Establece una interface común que es implementada por un proveedor de persistencia.

- **HIBERNATE**

Hibernate es un ORM (Herramienta de mapeo objeto-relacional), que te permite mapear las estructuras de una base de datos sobre una estructura lógica de entidades. Spring usa Hibernate como proveedor JPA por defecto.

- **SPRING DATA JPA**

Spring Data JPA proporciona [interfaces de repositorios](#) que automatizan las consultas básicas.

Lo único que será necesario tener en nuestro proyecto serán las dependencias de starter-spring-data-jpa en el pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

También hará falta el driver de la base de datos que vayamos a usar:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

Por último se añade la configuración en el archivo de application.properties.

```
application.properties X
API > src > main > resources > application.properties
1 server.port=8080
2
3 spring.servlet.multipart.max-file-size=100MB
4 spring.servlet.multipart.max-request-size=100MB
5
6 #https://spring.io/guides/gs/accessing-data-mysql/
7
8 spring.datasource.url=jdbc:mysql://localhost:3306/tfg?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
9 spring.datasource.username=root
10 spring.datasource.password=password
11
12 # =====
13 # = JPA / HIBERNATE
14 # =====
15
16 # Allows Hibernate to generate SQL optimized for a particular DBMS
17 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
18
19 # Hibernate ddl auto (create, create-drop, update): with "update" the database
20 # schema will be automatically updated accordingly to java entities found in
21 # the project
22 spring.jpa.hibernate.ddl-auto= update
23
```

1.1.1.1 Entidades

Lo principal cuando utilizamos JPA es la definición de las entidades. Las dos anotaciones

principales que debe tener una entidad son:

@Entity para definir que esta clase es una entidad, e **@Id** para indicar cual es su clave primaria.

Otras anotaciones:

@GeneratedValue: Para generar un id autoincremental.

@OneToOne, **@OneToMany**, **@ManyToOne** y **@ManyToMany** para indicar la relación entre dos entidades.

@EmbeddedId permite usar una clase anotada con **@Embeddable** como una clave primaria compuesta.

```
@Entity
public class RateUser {

    @EmbeddedId RateId rateId;

    private float score;

    public RateUser(){

    }

}
```

```
@Embeddable
public class RateId implements Serializable{

    private static final long serialVersionUID = 1L;

    @ManyToOne
    private User voted;

    @ManyToOne
    private User voter;

    @ManyToOne
    private Event event;

    private int type;

    public RateId(){

    }

}
```

1.1.1.2 Repositorios

Con Spring Data JPA definimos una interfaz de repositorio para cada entidad. Un repositorio contiene métodos para realizar operaciones CRUD típicas. Opcionalmente se pueden añadir más operaciones siguiendo una convención de nomenclatura. Por ejemplo, si se quiere encontrar un usuario por el Email la nomenclatura sería: `findByEmail()`

Se anotan con **@Repository** y en este caso se ha heredado de `CrudRepository` (contiene métodos CRUD básicos como: **findById()**, **findAll()**, **save()**, **delete()**, **count()**, etc..)

```
@Repository
public interface RateRepository extends CrudRepository<RateUser,RateId>{

    Set<RateUser> findByRateIdVotedId(long id);

    Set<RateUser> findByRateIdVoterId(long id);

}
```


1.1.4 Servicios

Se anotan con **@Service** y representan la capa de negocio de la aplicación y desde ella se accederá a la base de datos a través de los repositorios.

```
@Service
public class FriendshipService {

    @Autowired
    FriendshipRepository repository;

    @Autowired
    UserService userService;

    public Set<User> getMyRequestsReceived(User user){
        Set<Friendship> friendships = repository.findByFriendshipIdUserTwoId(user.getId());
        Set<User> requestsReceived = new HashSet<>();
        if(friendships != null){
            for(Friendship fs : friendships){
                if(fs.getStatus() == 0){
                    User sender = fs.getFriendshipId().getUserOne();
                    requestsReceived.add(sender);
                }
            }
        }
        return requestsReceived;
    }
}
```

1.1.5 Controladores

Atienden a las peticiones HTTP. En el caso de un servicio web basado en REST se anotan con **@RestController**

Los métodos de un controlador admiten distintos tipos de parámetros:

@RequestParam: /user?id=1

@PathVariable: /user/1

@RequestBody: En el cuerpo de una petición

Por defecto el formato para consumir y producir los datos es JSON.

1.3. Spring Validation

Proporciona validaciones para los objetos.

```
SignupRequest.java X
API > src > main > java > com > laura > api > payload > SignupRequest.java
1 package com.laura.api.payload;
2
3 import java.util.Date;
4
5 import javax.validation.constraints.Email;
6 import javax.validation.constraints.NotBlank;
7 import javax.validation.constraints.Size;
8
9 public class SignupRequest {
10     @NotBlank
11     @Size(max = 320)
12     @Email
13     private String email;
14
15     @Size(max = 32)
16     private String firstname;
17
18     @Size(max = 32)
19     private String lastname;
20
21     private Date birthday;
22
23     @NotBlank
24     @Size(min = 6)
25     private String password;
```

@Size: Para indicar el tamaño

@NotBlank: Que no esté vacío ni sea nulo

@Email: Para validar emails (aunque por defecto permite emails sin punto)

La anotación **@Valid** en el controlador será la que se encargue de validar los datos enviados, y con **BindingResult** se puede obtener la información de los errores.

```
@PostMapping("/signup")
public ResponseEntity<?> registerUser(@Valid @RequestBody SignupRequest signUpRequest, BindingResult bindingResult) {
    if (bindingResult.hasErrors()) {
        List<FieldError> errors = bindingResult.getFieldErrors();
```

2. JWT

Java JSON Web Token, una de las librerías de código abierto más usadas para la creación de JWT en Java.

Un JWT (JSON Web Token) es un estándar abierto (RFC-7519) basado en JSON para crear un token que sirva para enviar datos entre aplicaciones o servicios y garantizar que sean válidos y seguros. El caso más común es para manejar la autenticación en aplicaciones móviles o web.

La estructura del token es la siguiente: **header.payload.signature**

-Header: El encabezado donde se indicará el algoritmo (HS256) y el tipo de token (JWT).

-Payload: Datos de usuario, privilegios y otro tipo de datos que queramos añadir.

-Signature: La firma que nos permite verificar si el token es válido.

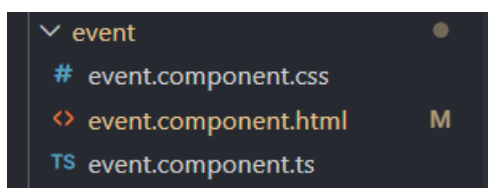
3 Angular

Angular es un framework de código abierto mantenido por Google que se usa para la creación de aplicaciones web.

3.1 Conceptos generales

-Modulos: Agrupa un conjunto de elementos (componentes, directivas, servicios...).

-Componentes: Una aplicación Angular se construirá en base a un árbol de componentes. Tienen una parte visual con HTML y CSS y una parte lógica con TypeScript.



El archivo `app.component.html` de esta aplicación contiene la etiqueta `<router-outlet></router-outlet>` que se encargará de mostrar un componente u otro en función de la ruta.

-**Rutas**: Cada ruta de la aplicación tendrá asignado un componente. Por ejemplo, las rutas de esta aplicación están definidas en el archivo `"app-routing.module.ts"` de la siguiente manera:

```
{ path: 'home', component: EventsComponent, canActivate: [AuthGuard]},
```

`path`: Especifica la ruta.

`component`: Asigna un componente a esa ruta.

`CanActivate`: Decide si se puede ir a esa ruta o no. Esto se hace a través de un guard que devolverá `true` o `false` para permitir o no el paso.

-**Servicios**: Definen un grupo de funcionalidades reutilizables. Por ejemplo, las llamadas a la API.

3.2 HttpClient

Es el mecanismo que tiene Angular para comunicarse con un servidor remoto a través del protocolo HTTP.

Todos los métodos de `HttpClient` devuelven un observable al cual habrá que subscribirse para recibir los datos o los errores.

```
getListEvents(): Observable<Event[]> {  
  let headers = new HttpHeaders({ 'Authorization': 'bearer ' + this.tokenStorage.getToken() });  
  return this.http.get<Event[]>(EVENT_API + '/list', { headers: headers });  
}
```

```
getAllEvents() {  
  this.eventService.getListEvents().subscribe(data => {  
    this.events = data;  
  });  
}
```

4. Bootstrap

Bootstrap es un framework de código abierto cuyo objetivo principal es facilitar el diseño web al desarrollador. Permite crear webs con un diseño "responsive", adaptable a cualquier tipo de dispositivo y tamaño de pantalla.

DESPLIEGUE

1. Heroku

El despliegue de la API se ha hecho con Heroku. Heroku es una plataforma como servicio (PaaS) que permite a los desarrolladores crear, ejecutar y operar aplicaciones completamente en la nube.

Tiene un plan gratuito para aplicaciones no comerciales con un límite de horas de uso, si se consumen todas esas horas la aplicación “se duerme” durante el resto del mes.

El despliegue en Heroku se hace a través de git.

1. Se crea una aplicación en Heroku
2. Se añade el nuevo remoto en nuestra aplicación con git
3. Se hace un commit de nuestros cambios
4. Con “git push heroku master” se desplegaría el proyecto en Heroku.

2. Firebase Hosting

El despliegue de la aplicación web se ha hecho con Firebase Hosting, que también ofrece hosting gratuito además de SSL sin tener que configurar nada.

Su despliegue también es muy sencillo:

1. Con **firebase init** se inicializa el proyecto.
2. **ng build –prod**, construye nuestra aplicación Angular para producción. Este comando sustituiría lo está en el archivo “environment.ts” por lo hay en “environment.prod.ts” (en este caso sustituye la variable “baseUrl” que contiene la url de la API en localhost, por otra que contiene la url de la API que está subida en Heroku).
3. Con **firebase deploy** se despliega nuestra aplicación.

BIBLIOGRAFÍA

<https://www.baeldung.com>

<https://spring.io/>

<https://angular.io/docs>

<https://devcenter.heroku.com/articles/free-dyno-hours>

<https://medium.com/@aleemuddin13/how-to-host-static-website-on-firebase-hosting-for-free-9de8917bebf2>

<https://openwebinars.net/blog/que-es-json-web-token-y-como-funciona/>

<https://www.baeldung.com/spring-security-registration-password-encoding-bcrypt>