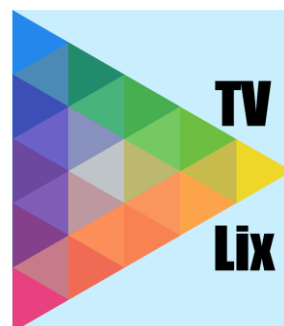


TV-LIX App

Documentación técnica y

Manual de Usuario

Miriam Gaviria



Índice

INTRODUCCIÓN	3
Objetivo del documento.....	3
Objetivo de la aplicación.	3
1. ARQUITECTURA DE LA APLICACIÓN.....	4
2. API REST – EPISODATE	5
3.- BASE DE DATOS.....	6
4.- DESARROLLO DE BACKEND	7
4.1.- Herramientas y tecnologías empleadas	7
4.2.- Microservicio TVLixServices	9
Entities.....	9
UserEntity.....	9
TvShowEntity	10
UserTvShowEntity	11
OpinionEntity	11
Controllers.....	12
UsersController	12
TvShowsController	12
UserTvShowsController.....	13
OpinionsController	14
Services.....	15
UsersService.....	15
TvShowsService.....	15
UserTvShowsService	15
OpinionsService	16
ServicesImpl	17
UsersServiceImpl.....	17
TvShowsServiceImpl.....	18
UserTvShowsServiceImpl	19
OpinionsServiceImpl	21
Repository	22
UsersRepository	22
TvShowsRepository	22
UserTvShowsRepository	22
OpinionsRepository.....	22

5. DESARROLLO FRONTEND.....	23
5.1.- Herramientas y tecnologías utilizadas	23
5.2.- Guía de estilos	24
Estructura	24
Logotipo.....	24
Estilos	25
Colores.....	25
Tipografía.....	25
Botones	25
Cards.....	26
Valoraciones	27
Alertas	27
Mensajes de error	28
5.3.- Estructura de la parte de frontend.....	29
6.- MANUAL DEL USUARIO	31
6.1.- Pantalla principal.....	31
6.2.- Login/Logout - Crear cuenta	32
6.3.- Pantalla principal del usuario logeado	33
6.4.- Listados de series - Formularios.....	34
Series que quiero ver.....	34
Formulario.....	35
Series viendo	35
Formulario.....	36
Series vistas	36
Formulario.....	37
Errores	37
6.5.- Pantalla de detalle de una serie.....	38
6.6.- Valoraciones APP – Formulario	39
6.7.- Perfil – Actualizar datos de la cuenta de usuario.....	40
7.- BIBLIOGRAFIA	42

Introducción

Objetivo del documento.

El objetivo del presente documento es definir y detallar tanto el funcionamiento de la aplicación *TVLix*, como su desarrollo web, divididos en diferentes apartados:

1. Diseño de la aplicación
2. Fuentes de información externa
3. Persistencia de los datos
4. Parte trasera de la aplicación o back end
5. Parte visual o front end
6. Manual de uso
7. Bibliografía

Objetivo de la aplicación.

En la actualidad, las series de TV son una forma de ocio y muchos de sus seguidores las ven en varias plataformas, por lo que considero de gran utilidad crear una aplicación web que nos permita gestionar nuestra colección de series que estamos viendo, queremos ver, hemos visto, etc. Así como el poder obtener información básica sobre su contenido, duración, etc. Incluye además características como en qué punto de la visualización se encuentra de aquellas que actualmente está viendo y dejar un histórico de aquellas que ya ha dejado de ver con una puntuación para cada una de ellas.

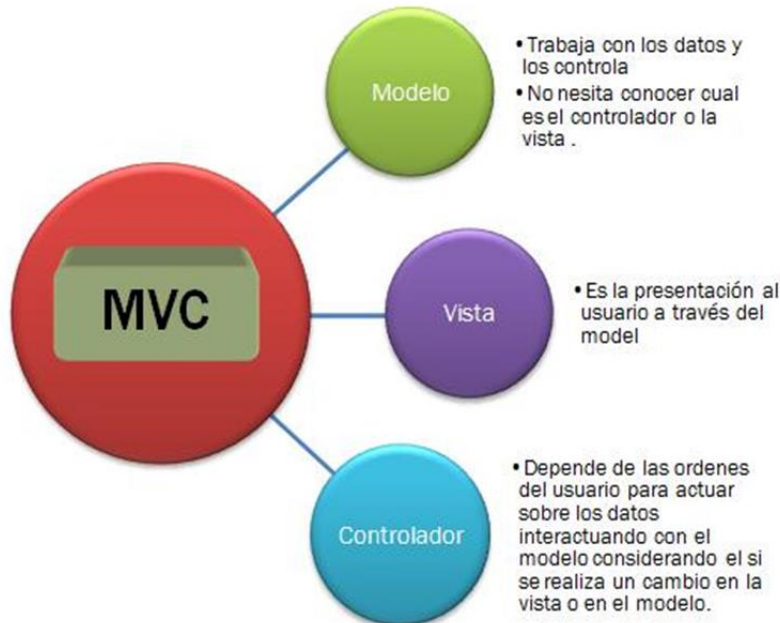
TVLix es una aplicación cuyo objetivo es gestionar el seguimiento de las diferentes series de televisión que el usuario quiere ver y por qué, en qué punto de la visualización se encuentra de aquellas que actualmente está viendo y dejar un histórico de aquellas que ya ha dejado de ver con una puntuación para cada una de ellas.

Además, también sirve como fuente de información de series de TV ya que recoge información de una fuente externa como es la API Episodate.

1. Arquitectura de la aplicación

Para el desarrollo de esta aplicación, se ha utilizado el patrón de **diseño MVC, Modelo Vista Controlador (MVC)*** (3 capas), donde se separan la interfaz de usuario, la lógica de control y los datos en tres componentes distintos:

- El **Modelo** con la representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.



- La **Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos de interacción con éste.
- El **Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ambos y las transformaciones para adaptar los datos a las necesidades de cada uno.

El desarrollo de la aplicación se puede dividir en cuatro apartados:

- Persistencia: he creado una base de datos MySQL
- Backend: desarrollado en Java
- Frontend: desarrollado en TypeScript
- API Rest: la fuente de información de la que bebe TVLix es la API Episodate.

Para poder desarrollar el proyecto, se ha montado un entorno de desarrollo con el servidor **XAMPP***, que consiste principalmente en la base de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script: PHP y Perl. Además, Xampp ha permitido instalar Apache permite probar los avances del desarrollo sin necesidad de tener que acceder a internet.

Además, también se ha realizado un control de versiones gracias al sistema **Git***, que me ha permitido tener una copia de seguridad del proyecto y ver la evolución del mismo, gracias a los diferentes commits realizados. Para trabajar con git, se ha utilizado la herramienta de entorno gráfico de Fork.

2. API REST – Episodate

Se suele considerar a una **API RESTFUL*** como un contrato entre un proveedor de información y un usuario, donde se establece el contenido que se requiere del consumidor (la llamada) y el que necesita el productor (la respuesta).

En este caso, se ha necesitado trabajar con una API para conseguir información de las series de TV. En concreto, y después de una tarea de investigación, se decidió consumir información de **Episodate***, debido a su sencillez de implementación, así como su variedad de datos disponibles.

De los diferentes endpoints que expone esta API, se han utilizado los siguientes:

- Most Popular TV Shows: una lista con las series más populares. El resultado de esta consulta contiene información básica y está ordenada de forma descendente:
 - **URL:** /api/most-popular?page=:page
 - **Example:** <https://www.episodate.com/api/most-popular?page=1>
- Search: busca a través de todas las series de la base de datos de la API. La llamada se realiza a través de un término, que es el título de la serie, y el resultado se ordena por orden alfabético por el nombre de la serie.
 - **URL:** /api/search?q=:search&page=:page
 - **Example:** <https://www.episodate.com/api/search?q=arrow&page=1>
- Show details: devuelve información completa sobre una serie determinada. La búsqueda se realiza a través del id que la serie tiene en la API.
 - **URL:** /api/show-details?q=:show
 - **Example:** <https://www.episodate.com/api/show-details?q=29560>

3.- Base de datos

Para la capa de persistencia se ha utilizado una base de **datos relacional***. Este tipo de bases de datos están formadas por un conjunto de tablas, similares a las tablas de una hoja de cálculo, formadas por filas (registros) y columnas (campos). Los registros representan cada uno de los objetos descritos en la tabla y los campos los atributos (variables de cualquier tipo) de los objetos. En el modelo relacional de base de datos, las tablas comparten algún campo entre ellas. Estos campos compartidos van a servir para establecer relaciones entre las tablas que permitan consultas complejas.

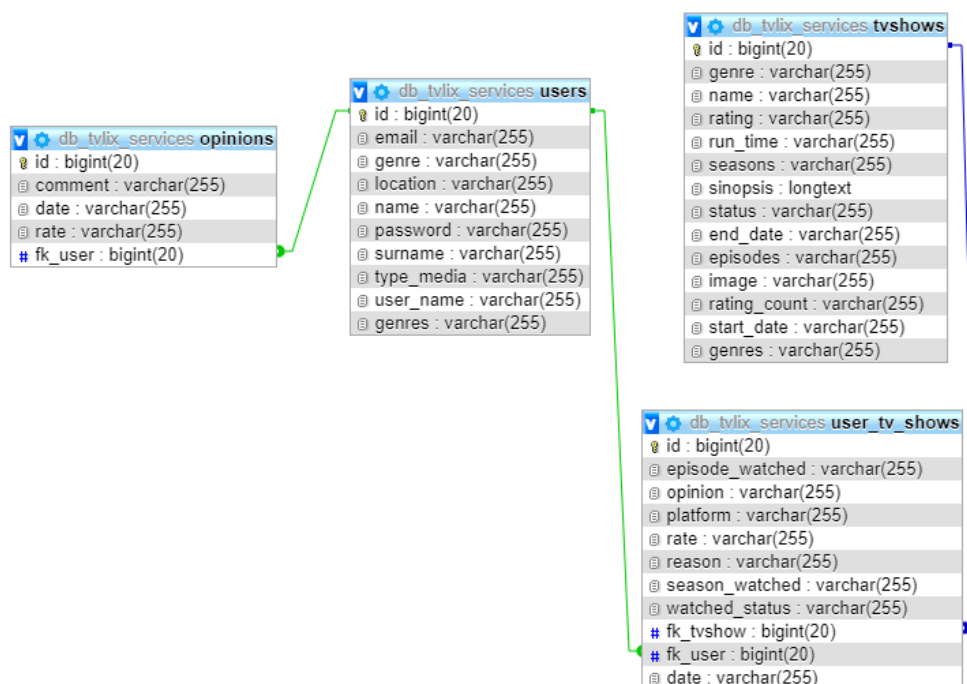
Para realizar las consultas, se utiliza el lenguaje de **SQL*** (Lenguaje Estructurado de Consultas) en el que las órdenes especifican cual debe ser el resultado. Al ser declarativo es muy sistemático, sencillo y con una curva de aprendizaje muy agradable.

Y para trabajar con la base de datos, se ha utilizado el sistema de gestión **MySQL***, que sirve para almacenar toda la información que se desee en bases de datos relacionales, como también para administrar todos estos datos sin apenas complicaciones gracias a su interfaz visual y a todas las opciones y herramientas de las que dispone. Es algo esencial, sobre todo en webs que cuentan con la opción de registrar usuarios para que inicien sesión.

Para trabajar con MySQL, se ha utilizado la herramienta de PHPMyAdmin que provee XAMPP.

En la aplicación de TVLix, la base de datos está formada por cuatro tablas:

- User, cuya clave primaria es su id, generada automáticamente
- TvShows: cuya clave primaria es la id que viene de la API
- UserTvShows: es la table que pone en relación la tabla de User y TvShow, de ahí que tiene dos claves foráneas que son las primarias de las otras dos tablas.
- Opinions: que guarda relación con la tabla de User



4.- Desarrollo de Backend

La parte de backend de TVLix se ha desarrollado en el lenguaje de programación Java y utilizando el framework de SpringBoot con la herramienta Spring tool suite.

4.1.- Herramientas y tecnologías empleadas

Spring boot* es una herramienta que es facilitada por el framework de desarrollo spring, que permite crear aplicaciones de manera rápida sólo configurando algunas dependencias y los parámetros requeridos en los archivos de configuración (properties). Su objetivo es simplificar la declaración de dependencias de un proyecto y el despliegue del mismo en un particular server, permitiendo así forma centrarse en la lógica de negocio y por tanto, sus ventajas son:

- Rápido setup de aplicación y servidor.
- Buen soporte en el sitio de spring.io y en su comunidad (gran cantidad de recursos para consultar en Internet).
- Se puede obtener una aplicación funcionando en verdaderamente poco tiempo.
- Los archivos donde se declaran las dependencias tienden a ser más cortos, ya que sólo es necesario declarar unas pocas dependencias de spring boot y luego estas mismas se encargarán de descargar todos los jars necesarios.

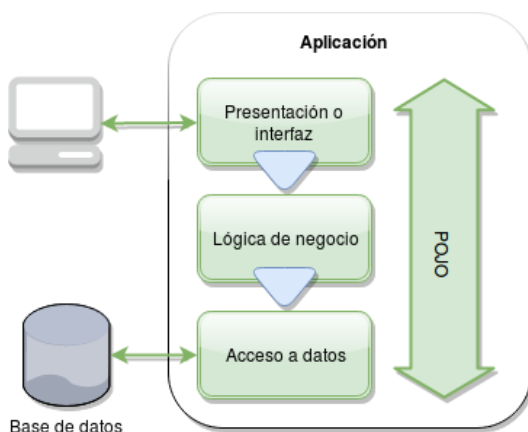
Junto a Spring boot se ha utilizado la herramienta open source **Maven*** que simplifica los procesos de build (compilar y generar ejecutables a partir del código fuente).

Como complemento, también se ha recurrido a **Java Hibernate***, una herramienta de mapeo objeto-relacional (ORM) bajo licencia GNU LGPL, que facilita el mapeo de atributos en una base de datos tradicional, y el modelo de objetos de una aplicación mediante archivos declarativos o anotaciones en los beans de las entidades que permiten establecer estas relaciones. Todo esto, permite agilizar la relación entre la aplicación y la base de datos SQL.



TvLix.postman_collection.json

Además, a lo largo del desarrollo de la parte de backend, se ha empleado la herramienta **Postman*** para testear, monitorizar, mockear y simular cada uno de los métodos que he necesitado crear en mi aplicación. Adjunto colección de las diferentes request desarrolladas.



El desarrollo de la parte de backend se ha realizado a través de **microservicios***, en concreto con uno, donde se gestiona tanto la información recibida desde la API, como la que se trabaja con la base de datos creada expreso.

Un microservicio permite separar responsabilidades al constar de varios niveles:

1. Interactuar con el cliente del servicio (capa interfaz o presentación). Aquí agrupamos las clases y métodos que conforman el **API** del servicio web.

2. Ejecutar procesamientos y aplicar reglas de negocio. Estas son clases que tienen la «sabiduría» o conocimiento sobre la lógica de lo que hace el servicio web. Algunos llaman a estas clases, **servicios**
3. Acceso a datos. Son las clases que se encargan de almacenar y extraer la información de un repositorio. Algunos las llaman *Repository* o *Data Access Object (DAO)*

4.2.- Microservicio TVLixServices

El microservicio está formado por estos apartados:

- Controller: supone la puerta de entrada y de salida de los datos que introduce el usuario (vista) o de la información de la base de datos (modelo)
 - Service: interfaz con los métodos con los que se realizan las diferentes llamadas y mapeos necesario para conectar el modelo con la vista o viceversa.
 - Implementación del servicio: con la lógica de las funciones anteriores.
 - Repository: donde se realiza la query a la base de datos. En este caso, he utilizado Hibernate para facilitar esta labor gracias a los métodos ya embebidos en esta herramienta.
- Entity: se trabaja con cuatro entities que corresponden a las cuatro tablas contenidas en la base de datos: user, tvShow, userTvShow y opinions.

Entities

UserEntity

```
@Entity
@Table(name = "users")
@Data
@ToString(includeFieldNames = true)
@JsonInclude(Include.NON_NULL)

public class User implements Serializable{

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long id;
    @Column
    private String genre;
    @Column
    private String location;
    @Column
    private String name;
    @Column
    private String email;
    @Column
    @NotNull
    private String password;
    @Column
    private String surname;
    @Column
    private String typeMedia;
    @Column
    @NotNull
    private String userName;
}
```

Modelo User con la información que introduce el propio usuario cuando crea una cuenta en la aplicación.

Por una parte, contiene información sobre la cuenta: email, password, username e id; y por otra, contiene información sobre el usuario: nombre, apellido, localidad y preferencias sobre el tipo de contenido multimedia (serie de TV, película, serie...) y el género (drama, comedia...).

Estos dos últimos campos, typeMedia y genre, se podrían utilizar en una versión posterior de TVLix, por ejemplo, para mandar información personalizada al usuario sobre diferentes series de TV.

TvShowEntity

```

@Entity
@Table (name = "tvshows")
@Data
@ToString(includeFieldNames = true)
@JsonInclude(Include.NON_NULL)

public class TvShow implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column
    private long id;

    @Column
    private String name;

    @Column
    private String runTime;

    @Column
    private String genre;

    @Column
    private String seasons;

    @Column
    private String episodes;

    @Column
    private String rating;

    @Column
    private String rating_count;

    @Column
    private String status;

    @Column
    private String sinopsis;

    @Column
    private String image;

    @Column
    private String start_date;

    @Column
    private String end_date;
}

```

Esta entidad corresponde a la información que recibimos de la API y se guarda en la base de datos para no tener que realizar tantas llamadas a la API.

De toda la información que se recibe de Episodate, se decidió persistir estos parámetros que son los necesarios para que le usuario gestione la visualización de las televisiones.

UserTvShowEntity

```

@Entity
@Table (name = "userTvShows")
@Data
@ToString(includeFieldNames = true)
@JsonInclude(Include.NON_NULL)

public class UserTvShow implements Serializable{

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long id;

    @ManyToOne
    @JoinColumn(name = "FK_USER", nullable = false, updatable = false)
    private User user;

    @ManyToOne
    @NotNull
    @JoinColumn(name = "FK_TVSHOW", nullable = false, updatable = false)
    private TvShow tvShow;

    @Column
    @NotNull
    public String watchedStatus;

    @Column
    private String rate;

    @Column
    private String opinion;

    @Column
    private String date;

    @Column
    private String seasonWatched;

    @Column
    private String episodeWatched;

    @Column
    private String reason;

    @Column
    private String platform;
}

```

Esta entidad es la encargada de relacionar los usuarios de la aplicación con las series de la base datos. De ahí, que tenga dos foreign keys:

- User, del tipo User
- tvShow, del tipo TvShow

Además, contiene otros parámetros que son los que el usuario puede ir modificando para gestionar las series.

OpinionEntity

```

@Entity
@Table (name = "opinions")
@Data
@ToString(includeFieldNames = true)
@JsonInclude(Include.NON_NULL)

public class Opinion implements Serializable{

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long id;

    @Column
    private String date;

    @Column
    private String comment;

    @Column
    @NotNull
    private String rate;

    @ManyToOne
    @JoinColumn(name = "FK_USER", nullable = false, updatable = false)
    private User user;
}

```

Esta entidad corresponde a las opiniones y valoraciones que el usuario que puede dejar sobre la aplicación.

Esto permite que las siguientes versiones de TVLix, pueda tener en cuenta estas opiniones de los usuarios.

Controllers

UserController

```
@CrossOrigin(origins= {"https://tvlix.dendarii.es"})
@RestController
@RequestMapping ("/api/users")
public class UsersController {

    @Autowired
    private UsersService usersService;

    @PostMapping ("/isUser")
    public int checkIsUser (@RequestBody @Valid User user) {

        return usersService.checkIsUser(user);
    }

    @GetMapping("/userName/{userName}")
    public User getUserByUsername(@PathVariable (required = true) @Valid String userName) {

        return usersService.getUserByUsername(userName);
    }

    @GetMapping("/{id}")
    public User getUserById(@PathVariable (required = true) @Valid long id) {

        return usersService.getUserById(id);
    }

    @PostMapping
    public boolean postUser(@RequestBody @Valid User user) {

        return usersService.saveUser(user);
    }

    @PutMapping
    public boolean putUser(@RequestBody @Valid User user) {

        return usersService.updateUser(user);
    }

    @DeleteMapping("/{id}")
    public void deleteUserById(@PathVariable (required = true) @Valid long id) {

        usersService.deleteUserById(id);
    }
}
```

Este controller es la puerta de entrada y salida de los datos relacionados con los usuarios de la aplicación. Gracias a él, se pueden crear usuarios, actualizarlos, eliminarlos y recibir información sobre ellos gracias a los métodos `getUserById` y `getUserByld`.

TvShowsController

Este controller es el encargado de manejar la información sobre las series que se recibe de la API en la base de datos. Con este controller el objetivo es no realizar tantas llamadas a Episodate y evitar posibles problemas si la API tiene algún tipo de problema

```

import org.springframework.beans.factory.annotation.Autowired;

@CrossOrigin(origins= {"https://tvlix.dendarii.es"})
@RestController
@RequestMapping ("/api/tvShows")
public class TvShowsController {

    @Autowired
    private TvShowsService tvShowsService;

    @GetMapping
    public List <TvShow> getTvShows() {

        return tvShowsService.getAllTvShows();
    }

    @GetMapping("/{id}")
    public TvShow getTvShowById(@PathVariable (required = true) @Valid long id) {
        return tvShowsService.getTvShowById(id);
    }

    @PostMapping
    public void postTvShow(@RequestBody @Valid TvShow tvShow) {

        tvShowsService.saveTvShow(tvShow);
    }

    @PutMapping
    public void putTvShow(@RequestBody @Valid TvShow tvShow) {

        tvShowsService.updateTvShow(tvShow);
    }

    @DeleteMapping("/{id}")
    public boolean deleteTvShowById(@PathVariable (required = true) @Valid long id) {
        return tvShowsService.deleteTvShowById(id);
    }
}

```

En esta ocasión, el usuario no puede eliminar ninguna serie, sino que es el administrador-desarrollador de la aplicación, quien, bajo su opinión, considere que ya no es necesario que una serie siga ocupando espacio en la base de datos. Para eliminar la serie, puede realizar a través de una llamada con Postman.

UserTvShowsController

```

@CrossOrigin(origins= {"https://tvlix.dendarii.es"})
@RestController
@RequestMapping ("/api/user_tv_shows")
public class UserTvShowsController {

    @Autowired
    private UserTvShowsService userTvShowsService;

    @GetMapping
    public List <UserTvShow> getUserTvShows() {

        return userTvShowsService.getAllUserTvShows();
    }

    @GetMapping("/{userId}")
    public List <UserTvShow> getUserTvShowsBy(@PathVariable (required = true) @Valid long userId) {

        return userTvShowsService.getUserTvShows(userId);
    }

    @GetMapping("/{userId}/{watchedStatus}")
    public List <UserTvShow> getUserTvShowsByStatus(@PathVariable (required = true) @Valid long userId, @PathVariable (required = true) @Valid String watchedStatus) {

        return userTvShowsService.getUserTvShowsByStatus(userId, watchedStatus);
    }

    @PostMapping
    public void postUserTvShow(@RequestBody @Valid UserTvShow userTvShow) {

        userTvShowsService.saveUserTvShow(userTvShow);
    }

    @PutMapping
    public void putUserTvShow(@RequestBody @Valid UserTvShow userTvShow) {

        userTvShowsService.updateUserTvShow(userTvShow);
    }

    @DeleteMapping("/{id}")
    public void deleteUserTvShowById(@PathVariable (required = true) @Valid long id) {

        userTvShowsService.deleteTvShowUserById(id);
    }
}

```

Este controller se encarga de relacionar la información de la serie de TV con el usuario. Es el controller que se utiliza para mostrar la información de las diferentes listas del usuario en la aplicación y de los formularios que puede rellenar.

OpinionsController

```
@CrossOrigin(origins= {"https://tvlix.dendarii.es"})
@RestController
@RequestMapping ("/api/opinions")
public class OpinionsController {

    @Autowired
    private OpinionsService opinionsService;

    @GetMapping
    public List <Opinion> getAllOpinions() {

        return opinionsService.getAllOpinions();
    }

    @PostMapping
    public void postOpinion (@RequestBody @Valid Opinion opinion) {

        opinionsService.saveOpinion(opinion);
    }

    // Method to probe
    @PutMapping
    public void putOpinion (@RequestBody @Valid Opinion opinion) {

        opinionsService.updateOpinion(opinion);
    }

    @DeleteMapping("/{id}")
    public void deleteOpinion (@PathVariable (required = true) @Valid Long id) {

        opinionsService.deleteOpinion(id);
    }
}
```

Este controller gestiona las opiniones que crea el usuario.

Como en controllers anteriores, en este caso, el usuario no puede ni modificar, ni eliminar una opinión. Es el desarrollador quien lo hace, si por ejemplo, se trata de una opinión que ofende a alguien.

Services

Diferentes interfaces con las que se modelan y se transfieren los datos.

UserService

```
public interface UserService {  
    public abstract int checkIsUser(User user);  
    public abstract User getUserByUsername(String userName);  
    public abstract User getUserById(long id);  
    public abstract boolean saveUser(User user);  
    public abstract boolean updateUser(User user);  
    public abstract void deleteUserById(long id);  
}
```

TvShowsService

```
package com.gaviros.tvlix.service;  
  
import java.util.List;  
  
public interface TvShowsService {  
    public abstract List<TvShow> getAllTvShows();  
    public abstract TvShow getTvShowById(long id);  
    public abstract void saveTvShow(TvShow tvShow);  
    public abstract void updateTvShow(TvShow tvShow);  
    public abstract boolean deleteTvShowById(long id);  
}
```

UserTvShowsService

```
package com.gaviros.tvlix.service;  
  
import java.util.List;  
  
public interface UserTvShowsService {  
    public abstract List<UserTvShow> getAllUserTvShows();  
    public abstract List<UserTvShow> getUserTvShowsByStatus(@Valid long userId, @Valid String watchedStatus);  
    public abstract List<UserTvShow> getUserTvShows(@Valid long userId);  
    public abstract void saveUserTvShow(@Valid UserTvShow userTvShow);  
    public abstract void updateUserTvShow(@Valid UserTvShow userTvShow);  
    public abstract void deleteTvShowUserById(@Valid long id);  
}
```


OpinionsService

```
package com.gaviros.tvlix.service;

import java.util.List;

public interface OpinionsService {

    public abstract List <Opinion> getAllOpinions ();

    public abstract void saveOpinion (Opinion opinion);

    public abstract void updateOpinion(@Valid Opinion opinion);

    public abstract void deleteOpinion (long id);
```

ServiceImpl

En este apartado, se implementan los diferentes métodos de las interfaces de los servicios.

Muchos de estos métodos, casi no tienen lógica, así que se procede a explicar aquellos que tienen una parte de lógica.

UserServiceImpl

checkIsUser()

Este método sirve para comprobar si el usuario que se quiere logear en la aplicación está en la base de datos y si los datos introducidos, coinciden con los que hay.

El método devuelve 0 si el usuario que se ha enviado a través del body de la request se encuentra en la base de datos o si ha habido algún problema con la petición. Si está en la tabla de user, devuelve 1 si el userName y la password introducidas, no coinciden con lo guardado, y 2 si los datos coinciden.

```
@Override
public int checkIsUser(@Valid User user) {
    try {
        User userRecovered = usersRepository.findByUserName(user.getUserName());

        if(userRecovered == null || userRecovered.getUserName().isEmpty()) {
            Log.info("the user isn't registred on database");
            return 0;
        } else {
            if (user.getUserName().equals(userRecovered.getUserName()) && user.getPassword().equals(userRecovered.getPassword())) {
                Log.info("the user name and the password are corrected");
                return 2;
            } else {
                Log.info ("the password is not corrected");
                return 1;
            }
        }
    } catch (Exception e) {
        Log.error("The user couldn't login " + e.getMessage());
        return 0;
    }
}
```

SaveUser()

Este método comprueba que el objeto user que se envía en el body, no se encuentra en la base de datos, y si es así, el recibido por parámetro se guarda en la tabla User.

Se trata de un método void, por lo que no devuelve nada.

```

public boolean saveUser(User user) {
    try {

        User userRecovered = new User();

        userRecovered = usersRepository.findByUserName(user.getUserName());

        if(userRecovered != null && userRecovered.getUserName().equals(user.getUserName())) {

            Log.info ("user not saved because it is already saved in database");

            return false;

        } else {

            usersRepository.save(user);

            return true;

        }

    } catch (Exception e) {

        return false;

    }

}

```

UpdateUser()

Este boolean método recupera el usuario de la base de datos por el id del usuario que se envía, y se comprueba si este objeto recuperado es igual al que se envía. Si es así, devuelve false y si es diferente, guarda el usuario y devuelve true.

```

@Override
public boolean updateUser(User user) {
    try {

        User userRecovered = usersRepository.findById(user.getId());

        if(userRecovered.equals(user)) {

            Log.info ("There is no changes");

            return false;

        } else {

            usersRepository.save(user);

            return true;

        }

    } catch (Exception e) {

        Log.error("Couldn't update the user " + e.getMessage());

        return false;

    }

}

```

TvShowsServiceImpl

UpdateUser()

Este método, comprueba que los episodios y el status de la serie es igual que estos parámetros del objeto que se envía a través del body. Si es así, no se hace nada, pero si son diferentes, se actualiza la serie.

```

@Override
public void updateTvShow(TvShow tvShow) {
    try {
        TvShow tvShowRecovered = tvShowsRepository.findById(tvShow.getId());
        if (tvShow.getEpisodes().equals(tvShowRecovered.getEpisodes()) && tvShow.getStatus().equals(tvShowRecovered.getStatus())) {
        } else {
            tvShowsRepository.save(tvShow);
        }
    } catch (Exception e) {
        Log.error("The tvShow couldn't been updated " + e.getMessage());
    }
}

```

UserTvShowsServiceImpl

getUserTvShowsByStatus()

Este método devuelve un listado del tipo UsersTvShows según el valor del parámetro watchedStatus enviado en la request. En esta ocasión, los parámetros de la llamada, se envían a través de la url y son el id del usuario y el string correspondiente al watchedStatus

```

@Override
public List<UserTvShow> getUserTvShowsByStatus(@Valid long userId, @Valid String watchedStatus) {
    try {
        List<UserTvShow> userTvShows = getUserTvShows(userId);
        List<UserTvShow> userTvShowsByStatus = userTvShows.stream().filter(tvShow -> tvShow.watchedStatus.equals(watchedStatus)).collect(Collectors.toList());
        return userTvShowsByStatus;
    } catch (Exception e) {
        Log.error("Couldn't get the user userTvShows by watchedStatus " + e.getMessage());
        return null;
    }
}

```

SaveUserTvShowsByStatus()

Este método sirve para guardar las series de un usuario.

Lo primero que se hace, es añadir la fecha del momento en el que se guarda la serie y luego se comprueba si esa serie, que se transfiere a través del body de la request, está en la tabla de userTvShow, es decir, si en esa tabla hay un registro cuyas foreign keys, coincide con los objetos de user y tvShows incluidos en el objeto userTvShow. Si no es así, guarda la serie en la tabla de TvShows y la userTvShow en su correspondiente tabla.

Se trata de un método void.

```

@Override
public void saveUserTvShow(@Valid UserTvShow userTvShow) {

    String userTvShowDateAsString = getCurrentDate();
    userTvShow.setDate(userTvShowDateAsString);

    try {

        TvShow tvShowRecovered = tvShowsRepository.findById(userTvShow.getTvShow().getId());

        if (tvShowRecovered == null) {

            tvShowsRepository.save(userTvShow.getTvShow());
            userTvShowRepository.save(userTvShow);

            System.out.println("tvShow saved on tvShows table and userTvShows table" );

        } else {

            UserTvShow userTvShowRecovered = getUserTvShowByTvShowId(userTvShow);

            if (userTvShowRecovered == null) {

                userTvShowRepository.save(userTvShow);

                System.out.println("userTvShow save on userTvShows table" );

            }

        }

    } catch (Exception e) {

        Log.error("Couldn't get the user userTvShows by watchedStatus " + e.getMessage());

    }

}

```

UpdateUserTvShowsByStatus()

Este método sirve para actualizar las series de un usuario. Como en el método anterior, también es un método void.

Lo primero que se hace, es añadir la fecha del momento en el que se guarda la serie. Después, se recupera la userTvShow que coincide con la que se pasa por el body. Si estos objetos son iguales, no se hace nada, pero si son diferentes, se actualiza la userTvShow existen con el nuevo objeto.

```

@Override
public void updateUserTvShow(@Valid UserTvShow userTvShow) {

    String userTvShowDateAsString = getCurrentDate();
    userTvShow.setDate(userTvShowDateAsString);

    UserTvShow userTvShowRecovered = getUserTvShowByTvShowId(userTvShow);

    try {

        if (userTvShow.equals(userTvShowRecovered)) {

            System.out.println("userTvShow not saved because it is already saved in database");

        } else {

            userTvShowRepository.save(userTvShow);

            System.out.println("usrTvShow updated" );

        }

    } catch (Exception e) {

        Log.error("Couldn't update the userTvShow " + e.getMessage());

    }

}

```

GetUserTvShowByIdTvShowId()

Este es un método privado que se utiliza únicamente en la implementación del userTvShowsService y que se llama desde los métodos de SaveUserTvShowsByStatus y updateUserTvShow.

El objetivo es conseguir la userTvShow que coincide con la que se pasa por el parámetro.

```
private UserTvShow getUserTvShowByTvShowId(UserTvShow userTvShow) {
    try {
        List<UserTvShow> userTvShowListRecovered = userTvShowRepository.findByUser(userTvShow.getUser());
        UserTvShow userTvShowById = null;
        for (UserTvShow userTvShowRecovered : userTvShowListRecovered) {
            if (userTvShowRecovered.getTvShow().getId() == userTvShow.getTvShow().getId()) {
                userTvShowById = userTvShowRecovered;
            }
        }
        return userTvShowById;
    } catch (Exception e) {
        Log.error("Couldn't get userTvShowByTvShowId " + e.getMessage());
        return null;
    }
}
```

OpinionsServiceImpl

SaveOpinion()

```
@Override
public void saveOpinion (Opinion opinion) {
    Date currentDate = new Date();
    String strDateFormat = "yyyy-MM-dd HH:mm";
    SimpleDateFormat opinionDate = new SimpleDateFormat(strDateFormat);
    String opinionDateAsString = opinionDate.format(currentDate);
    opinion.setDate(opinionDateAsString);
    try {
        opinionsRepository.save(opinion);
    } catch (Exception e) {
        Log.error("Couldn't post the opinion " + e.getMessage());
    }
}
```

Este método void, tiene como finalidad guardar en base de datos, la opinión que ha creado el usuario a través de un formulario. Antes de guardar la opinión, se añade una fecha con hora del momento en el que se ha ejecutado la petición para que luego la lista de opiniones se pinten ordenadas por este dato.

Repository

En este apartado es donde se realizan las diferentes queries para conectar con la base de datos. En esta ocasión no ha sido necesario desarrollar ninguna custom query, ya que los métodos que facilita Java Hibernate han sido suficientes.

UsersRepository

```
public interface UsersRepository extends CrudRepository<User, Long>{  
    public abstract User findByUserName(String userName);  
    public abstract User findById(long id);  
}
```

TvShowsRepository

```
import org.springframework.data.repository.CrudRepository;[]  
public interface TvShowsRepository extends CrudRepository<TvShow, Long> {  
    public abstract TvShow findById(long id);  
}
```

UserTvShowsRepository

```
public interface UserTvShowRepository extends CrudRepository<UserTvShow, Long> {  
    List<UserTvShow> findByUser(User user);  
    UserTvShow findById(TvShow tvShow);  
}
```

OpinionsRepository

```
package com.gaviros.tvlix.repository;  
import org.springframework.data.repository.CrudRepository;[]  
public interface OpinionsRepository extends CrudRepository<Opinion, Long>{  
}
```

5. Desarrollo Frontend

El desarrollo de la parte frontend se ha realizado en el lenguaje de TypeScript, utilizando el framework de Angular a través del IDE de **Visual Studio Code***, que gracias a sus funcionalidades y plugins que he instalado (como Prettier), entre otras cosas, ha permitido escribir código manteniendo las mismas características de formato.

5.1.- Herramientas y tecnologías utilizadas

Angular* separa el frontend y el backend en la aplicación que permite mantener todo más ordenado gracias al patrón MVC (Modelo-Vista-Controlador). Además, facilita el trabajo cuando he tenido que modificar alguna de las partes, gracias a que favorece la escalabilidad.

Para que la aplicación fuera responsive, se ha utilizado **Bootstrap***, una librería con diferentes componentes, como cards o botones, con los que se ha maquetado la página web.

Otras herramientas empleadas son:

- **Sweet Alert***, con la que se han creado notificaciones y alertas de un modo más visual. Además, gracias a este plugin de JQuery, se han configurado estas notificaciones según las necesidades de cada momento: notificación de éxito, error...
- **Ngx-pagination***: para la paginación de la tabla de opiniones sobre la app

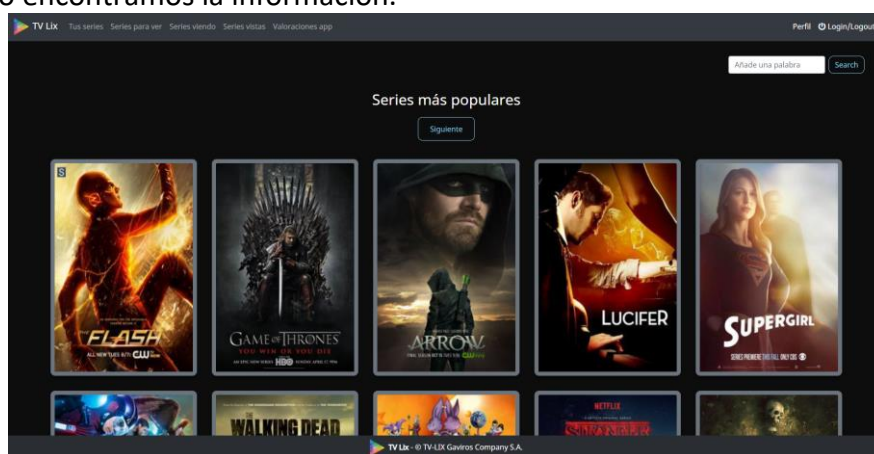
5.2.- Guía de estilos

El objetivo de este pequeño manual de estilos es normalizar la estructura de los contenidos y el diseño de la página web. Se han establecido las pautas que se han mantenido en el desarrollo de la web y que, a su vez, permiten mantener la coherencia ante posibles modificaciones.

Estructura

Todas las pantallas guardan una misma estructura:

- 1.- Zona de navegación: la parte de la izquierda de este menú contiene el logo y los enlaces a las diferentes pantallas que muestran series de televisión; mientras que, en la derecha, están los botones que nos envían a las páginas relacionadas con el usuario.
- 2.- Contenido: en este apartado encontramos la información.
- 3.- Pie: donde se puede ver el nombre de la “empresa” que ha desarrollado esta aplicación. Además, en el archivo Header.html, está comentada una línea de este pie con los diferentes botones que podrían llevar a las diferentes redes sociales de esta empresa (Facebook, Twitter, Instagram...)



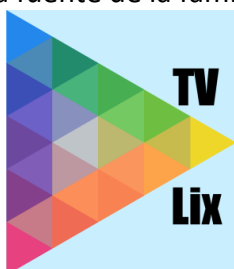
Mapa de navegación. Se decidió elaborar una estructura reticular del sitio web con secciones bien diferenciadas, pero entrelazadas entre sí a través de diferentes links.

Logotipo

El logotipo de TV Lix se ha diseñado combinando dos elementos que guardan relación con las series y el mundo de la televisión:

- La forma que tiene el símbolo del play.
- La gama cromática utilizada, es similar a la existente en la antigua pantalla de carta de ajuste que existía en España cuando se dejaba de emitir por televisión.

Completo: aparece en el formulario de login. Compuesto por el símbolo del play sobre un fondo azul (skyblue), que es el corporativo de la app. Y con la fuente de la familia Impact.



Reducido: únicamente el símbolo del play. Utilizado como favicon y en el pie de página



Estilos

Cada uno de los componentes de TV Lix tiene su correspondiente hoja de estilos, con SASS. Pero, además, en el archivo styles.css hay diferentes clases que son comunes en toda la aplicación. Gracias a este archivo, no he tenido que repetir algunos estilos.

Colores

Durante toda la aplicación son varios los colores utilizados:

- Negro (#0d0d0d): para el fondo del body
- Blanco(#FFFFFF): para el color de las letras
- Gris (#a2a2a2): para la barra del menú y el pie de página
- Skyblue (#87ceeb): para los botones, hover de enlaces y demás.
- #095370: usado en el listado de opiniones para los enlaces de página siguiente o anterior.

Tipografía

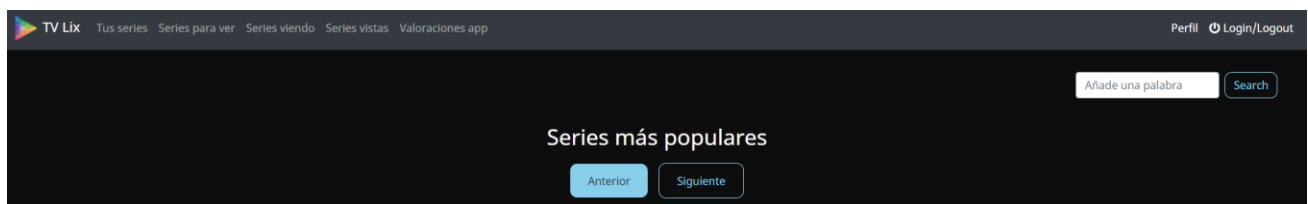
A lo largo de la aplicación se han utilizado varios tipos de fuentes:

- Noto-Sans regular: fuente base de todas las páginas. Con un tamaño de 1rem
- Para las cards de los listados de la cuenta del usuario, he utilizado: sans-serif, Poppins y FontAwesome, con la que se han incluido varios iconos.

Botones

Los estilos de los botones están contenidos en el archivo de styles.css, ya que se han empleado en diferentes pantallas.

Estos botones tienen la fuente y el borde de color skyblue, y en el momento en el que se hace hover sobre ellos, los colores se intercambian y el fondo pasa a ser de color skyblue y la fuente negra. Además, el texto está centrado, tanto vertical como horizontalmente, en un contenedor con los bordes redondeados. Y todos ellos están centrado con respecto al body de la pantalla.



Hay unos botones distintos de los anteriores: se trata de los enlaces para pasar de página en el listado de valoraciones de la app. El número de la página que se está viendo en ese momento es como un botón en hover “standard”, pero cuando hacemos hover se cambia el color por #095370.

Valoraciones de la app

Fecha	Usuario	Comentario	Valoración
2021-05-19 19:13	pau	Considero que se trata de una aplicación muy útil y sencilla de utilizar	★★★★★
2021-05-03 19:27	maria	Es una aplicación que podría mostrar más información, como la de los actores o los directores. Si me gusta, cómo gestionar mis listas de series.	★★★★★
2021-05-03 19:26	jon	Su uso es muy sencillo, por lo que cualquiera puede utilizarla	★★★★★
2021-05-03 19:25	Cristina	Me encanta esta aplicación porque tiene la información exacta que necesito y me permite tener controlados aquellos capítulos que he visto y los que no.	★★★★★
2021-04-14 20:41	pau	La interfaz es limpia y agradable	★★★★★
2021-04-08 18:30	ana	Se trata de una aplicación con buenas intenciones, pero que tiene que mejorar mucho	★★★★★

« Anterior 1 2 Siguiente »

Deja tu opinión

Cards

Durante el desarrollo se han diseñado dos tipos de cards:

- Las que aparecen en los listados de la cuenta de usuario: series para ver, viendo y vistas. Estas cards están divididas en dos partes: a la izquierda, la imagen de la serie con enlaces, que se hace

Listado de series que estás viendo actualmente

Busca una serie en este listado



La he visto

Más información

Eliminar de mi perfil

The Flash

Episodio visto: 4 -- Temporada viendo: 1

Barry Allen is a Central City police forensic scientist with a reasonably happy life, despite the childhood trauma of a mysterious red and yellow being killing his mother and framing his father. All that changes when a massive particle accelerator accident

Estado de la serie: **grabando capítulos**

[Modifica los datos](#)



Outlander

Episodio visto: 1 -- Temporada viendo: 3

Outlander follows the story of Claire Randall, a married combat nurse from 1945 who is mysteriously swept back in time to 1743, where she is immediately thrown into an unknown world where her life is threatened. When she is forced to marry Jamie, a chivalro...

Estado de la serie: **grabando capítulos**

[Modifica los datos](#)

visibles cuando se hace hover sobre ella; y a la derecha, con diferentes datos.

- Las que están en las pantallas de tvShows y de búsqueda. Estas cards son la respuesta de llamadas a Episodate. En ellas, no hay información, destaca la imagen de la serie (por lo que se la reconoce) y los diferentes enlaces para pasar esta serie a los diferentes listados de la cuenta del usuario. Este tipo de card también es la que se puede ver en la pantalla a la que es redirigido el usuario cuando hace login en la aplicación

Series que quieres ver



Bones


La estoy viendo

La he visto

Eliminar la serie

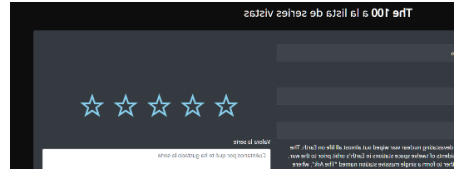
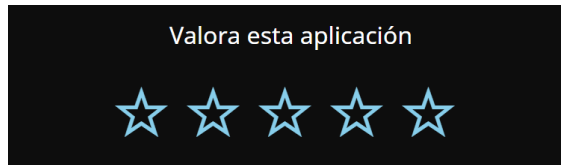
Más información

Series que estás viendo



Valoraciones

Tanto las valoraciones que realiza el usuario sobre la aplicación, como las que hace sobre una serie que ha visto, utilizo un sistema de estrellas de color skyblue que van del 1 al 5, donde el 5 es la máxima valoración.

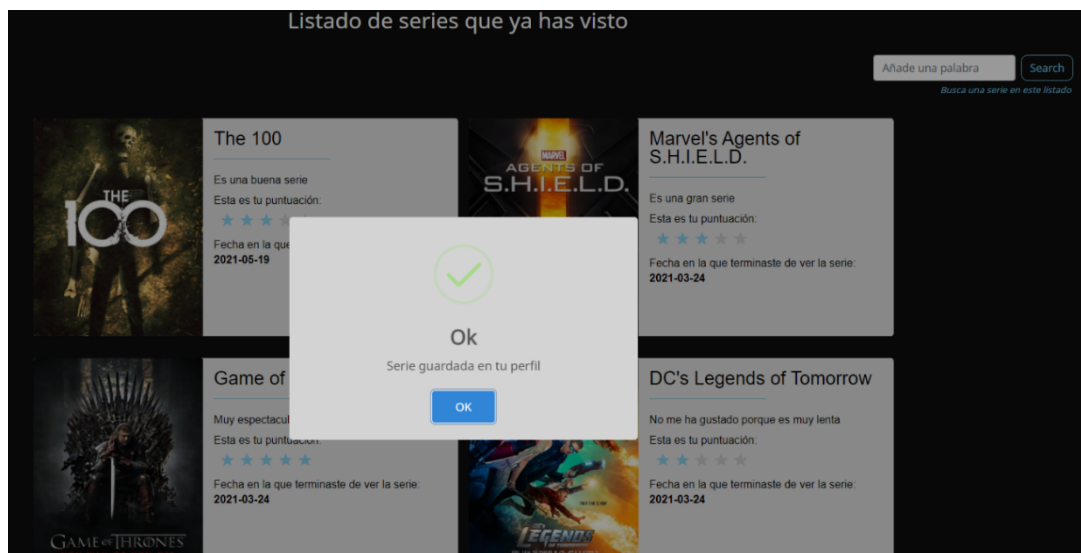


Alertas

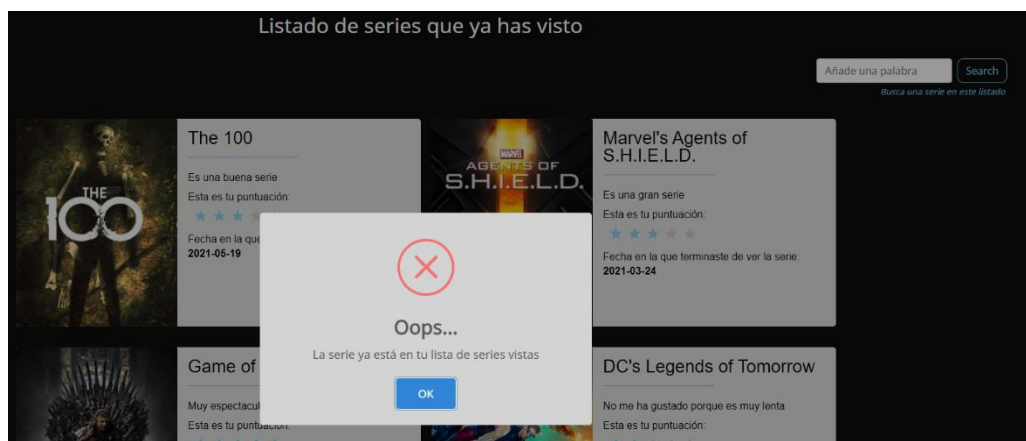
Gracias a la librería de SweetAlert2 se han personalizado las diferentes alertas que le pueden aparecer al usuario cuando realiza alguna acción.

Hay dos tipos de alertas:

- Éxito:



- Error:



Mensajes de error

Los formularios contenidos en la aplicación requieren que todos los campos sean rellenados por el usuario. Si alguno de ellos no se rellena, aparece un mensaje de error, que en todos guarda el mismo estilo y que está en el archivo `styles.css`

Supernatural en la lista de series viendo

Géneros	Drama, action, supernatural
Duración de cada capítulo	60 minutos
Temporadas	15
Episodios	326
Puntuación	9.6540
Síntesis	<p>Supernatural is an American fantasy horror television series created by Eric Kripke. It was first broadcast on September 13, 2005, on The WB and subsequently became part of successor The CW's lineup. Starting Jared Padalecki as Sam Winchester and Jensen Ackles as Dean Winchester, the series follows the two brothers as they hunt demons, ghosts, monsters, and other supernatural beings in the world. The series is produced by Warner Bros. Television, in association with Wonderland Sound and Vision. Along with Kripke, executive producers have been McG, Robert Singer, Phil Sgriccia, Sera Gamble, Jeremy Carver, John Shiban, Ben Edlund and Adam Glass. Former executive producer and director Kim Manners died of lung cancer during production of the fourth season.
The series is filmed in Vancouver, British Columbia and surrounding areas and was in development for nearly ten years, as creator Kripke spent several years unsuccessfully pitching it. The pilot was viewed by an</p>

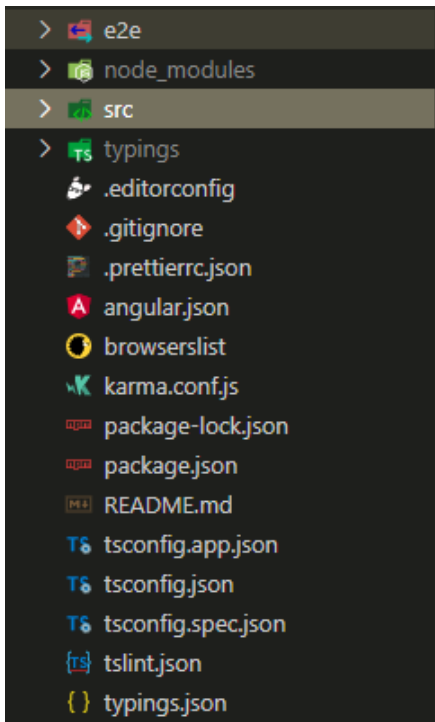
Temporada que estás viendo

Episodio que ya has visto

Todos los campos son requeridos

Guardar en series viendo

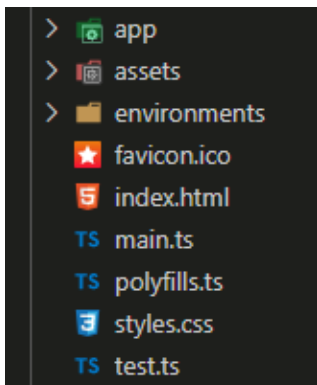
5.3.- Estructura de la parte de frontend



Para estructurar los archivos del frontend, se ha seguido la estructura que se genera al crear un proyecto en Angular*.

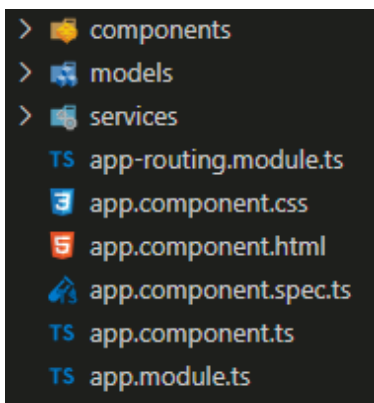
Algunos de los elementos más importantes que se pueden encontrar en esta imagen son:

- Carpeta node_modules.
- .prettierrc.json: archivo en el que se configura el formateo del código
- Package.json: es uno de los archivos de “configuración” del proyecto, donde se incluyen las dependencias y los scripts.



Dentro de la carpeta de src está el grueso del código de la parte de frontend:

- Carpeta app con los componentes, módulos y archivos de las rutas.
- Carpeta assets: guarda los diferentes recursos de la aplicación: imágenes, iconos, tipos de fuentes...
- Index.html: primera página de la aplicación (contiene la etiqueta <app-root>)
- Styles.css: el archivo general de los estilos, con diferentes clases que se utilizan en toda la aplicación.



Como ya se ha indicado, en la carpeta de app, se encuentra la parte más importante y específica de la aplicación:

- Carpeta components
- Carpeta models
- Carpeta services
- App-routing.module.ts: donde se configuran las diferentes rutas de los componentes de la aplicación
- App.component.html: donde se define la vista. En este caso está formado por las etiquetas: <app-header>, <router-outlet>, <app-footer>.
- App.module.ts: donde se ubican los diferentes módulos, servicios, importaciones y providers.

```

> finished-tv-shows
> finished-tv-shows-form
> footer
> found-tv-shows
> header
> login
> opinions
> tv-show-detail
> tvShows
> user
> user-tv-shows
> watching-tv-shows
> watching-tv-shows-form
> wished-tv-shows
> wished-tv-shows-form

```

Dentro de la carpeta componentes se incluyen todos los componentes creados para la aplicación. De todos estos, están siempre visibles el header y el footer, por eso están incluidos en el archivo `app.component.html`. El resto, varía según la ruta en la que se encuentre el usuario.

```

TS episodes.model.ts
TS opinion.model.ts
TS tvShowApi.model.ts
TS tvShowDetail.model.ts
TS tvShowDTO.model.ts
TS tvShows.model.ts
TS tvShowsList.model.ts
TS user.model.ts
TS userTvShowDTO.model.ts

```

Los modelos de la aplicación, son los archivos que trabajan con los datos. En este caso, hay modelos que reciben la información de la API y otros de la base de datos:

- `tvShowApi`: modelo con la información que llega de la API. Está compuesto por un objeto de tipo `tvShowDetail`, formado a su vez por campos, entre ellos, otro objeto de tipo `episodes`. Este modelo es el utilizado para ver la información de la serie
- `tvShowsList`: también contiene información de la API. Entre sus campos, está un listado de objetos `tvShows`. Es utilizado para los listado de las series más populares y el resultado de la búsqueda

- `UserTvShowDTO`: este modelo corresponde con el modelo de back `userTvShow`
- `TvShowDTO`: modelo que corresponde con el modelo de back `tvShow`
- `User`: con la información sobre el perfil del usuario. Este modelo se utiliza para crear, editar y eliminar un usuario, además de para mostrar su perfil.
- `Opinion`: es el modelo que contiene los campos de las valoraciones de la app.

```

TS login.service.ts
TS opinion.service.ts
TS tvShows.service.ts
TS user.service.ts
TS userTvShows.service.ts

```

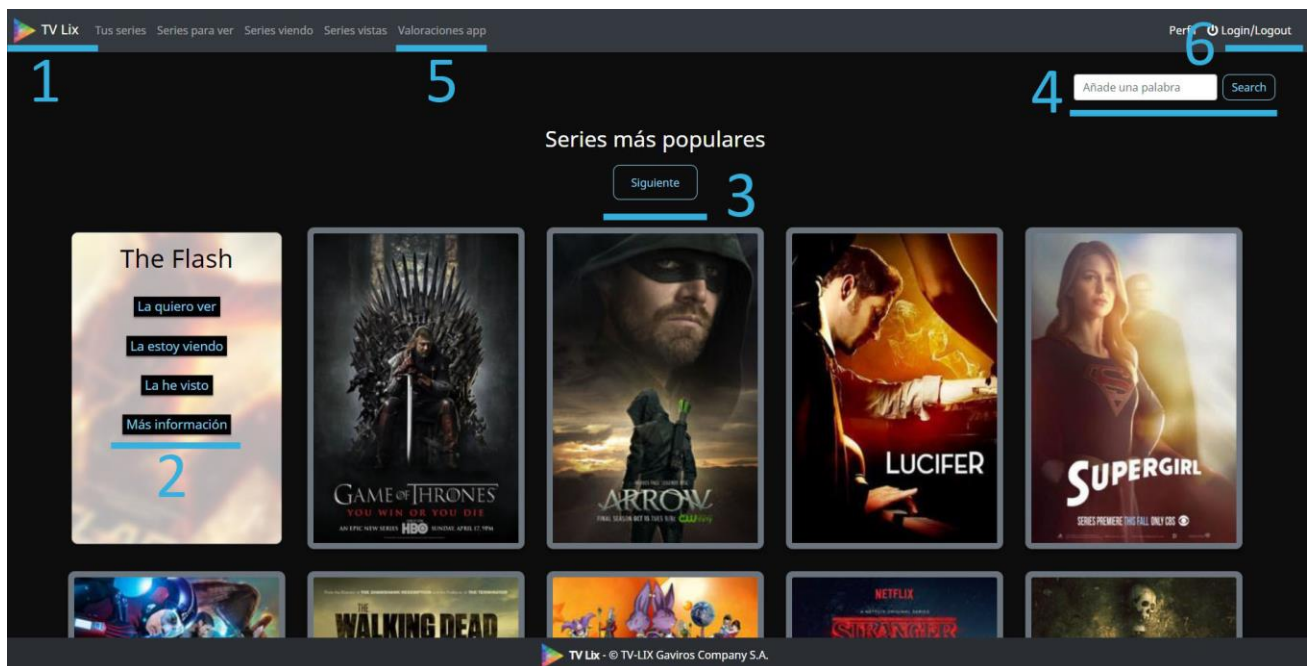
Los services son los archivos en los que se encuentran las diferentes llamadas que ponen en comunicación el front con back. En este caso hay cinco, que prácticamente coinciden con los controladores de back. La única diferencia es que en front está separado el servicio de login para acceder a la app y el de user, que contiene las llamadas relacionadas con el perfil del usuario.

6.- Manual del usuario

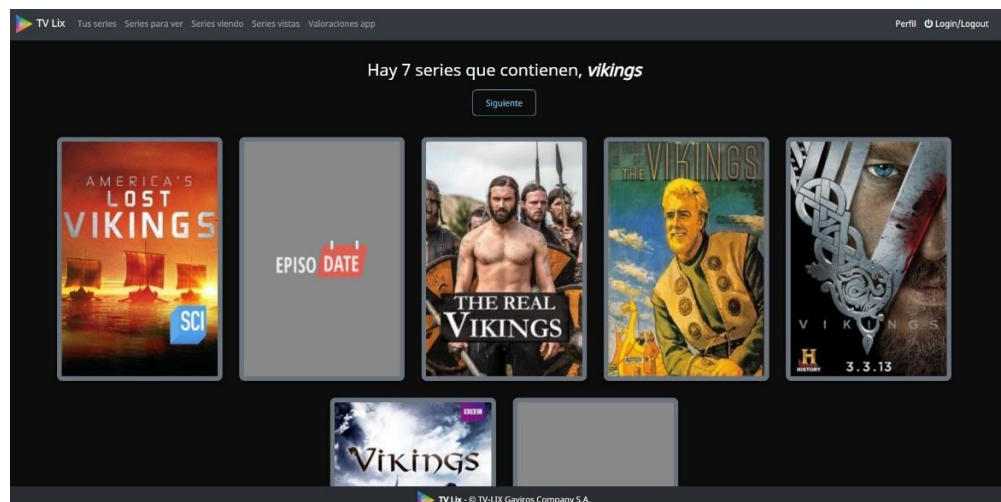
La aplicación TVLix está pensada y diseñada para facilitar a los usuarios información sobre series de TV y para facilitarles una gestión sobre el seguimiento de aquellas series en las que están interesados.

Cuando el navegador accede a esa dirección, lo primero que se ve es la pantalla de inicio, con el listado de las series más populares y en la que cualquier usuario, esté o no esté registrado, puede interactuar de diferentes maneras.

6.1.- Pantalla principal



- 1.- Botón TVLix: redirecciona a la pantalla principal
- 2.- Este botón redirige a otra pantalla en con información detallada de la serie
- 3.- Con este botón el usuario pasa a la siguiente página.
- 4.- Buscador: en este input el usuario puede añadir el título de una serie que quiere buscar y al pinchar, se abre un listado con el resultado que devuelve la API, muy similar a la pantalla actual.



5.- Este enlace lleva a la pantalla con el listado de las opiniones que han dejado los usuarios sobre la app. Cualquier usuario puede verlas, pero sólo los registrados pueden dejar sus opiniones.

6.- Al pinchar en el enlace de Login/Logout, el usuario puede acceder a la pantalla de login, si no está logeado, o salir de la sesión si lo está.

6.2.- Login/Logout - Crear cuenta

En esta pantalla, si el usuario quiere acceder a la aplicación relleno correctamente el formulario de acceso.

Además, si quisiera crear una cuenta nueva, podría acceder al formulario a través del botón: Crear cuenta.

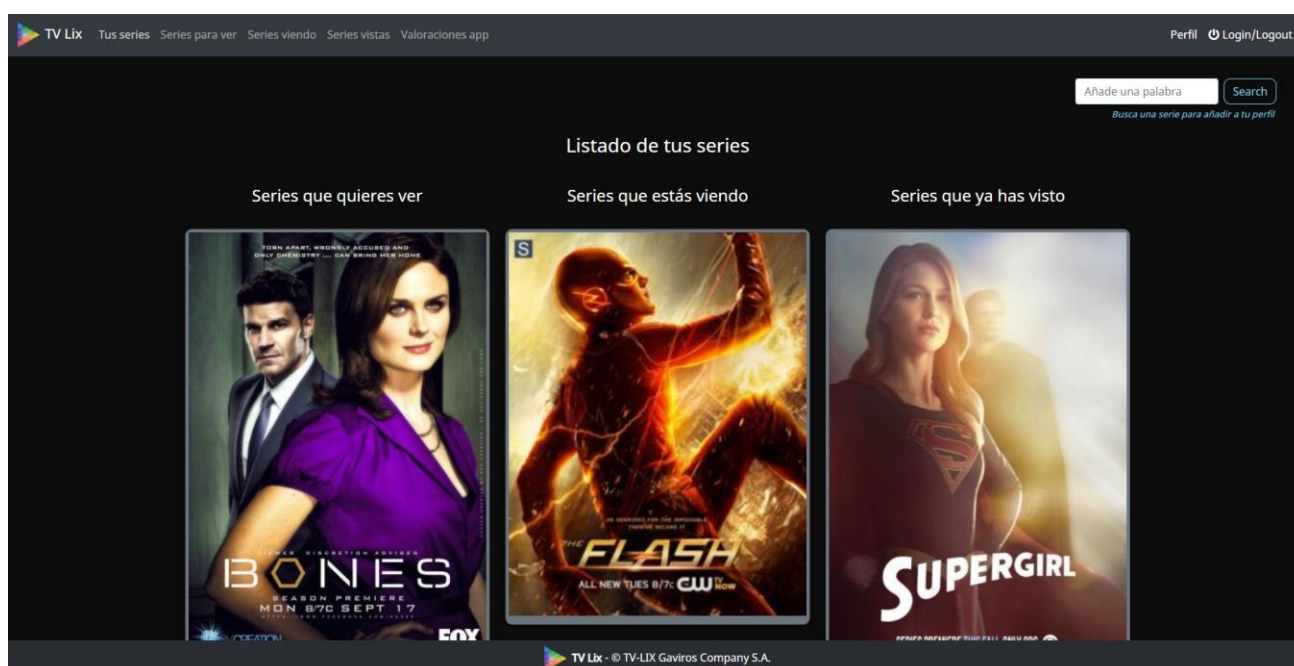
Este es el formulario que debe rellenar un usuario para poder registrarse en TVLix y poder así, utilizar completamente la aplicación. Como indica el texto, el usuario debe cumplimentar todos los campos porque si no, el sistema no le permitirá crear la cuenta.

6.3.- Pantalla principal del usuario logeado

Una vez que el usuario ha hecho login, entra en lo que se podría denominar su pantalla principal. Una pantalla en la que aparece un “resumen” de las tres listas de series de las que se compone esta aplicación:

- Series que quiere ver
- Series que actualmente está viendo
- Series vistas.

Si en alguna de estas tres listas, hay más de una serie en esa lista, aparece un botón en la parte inferior para redirigir al listado completo.



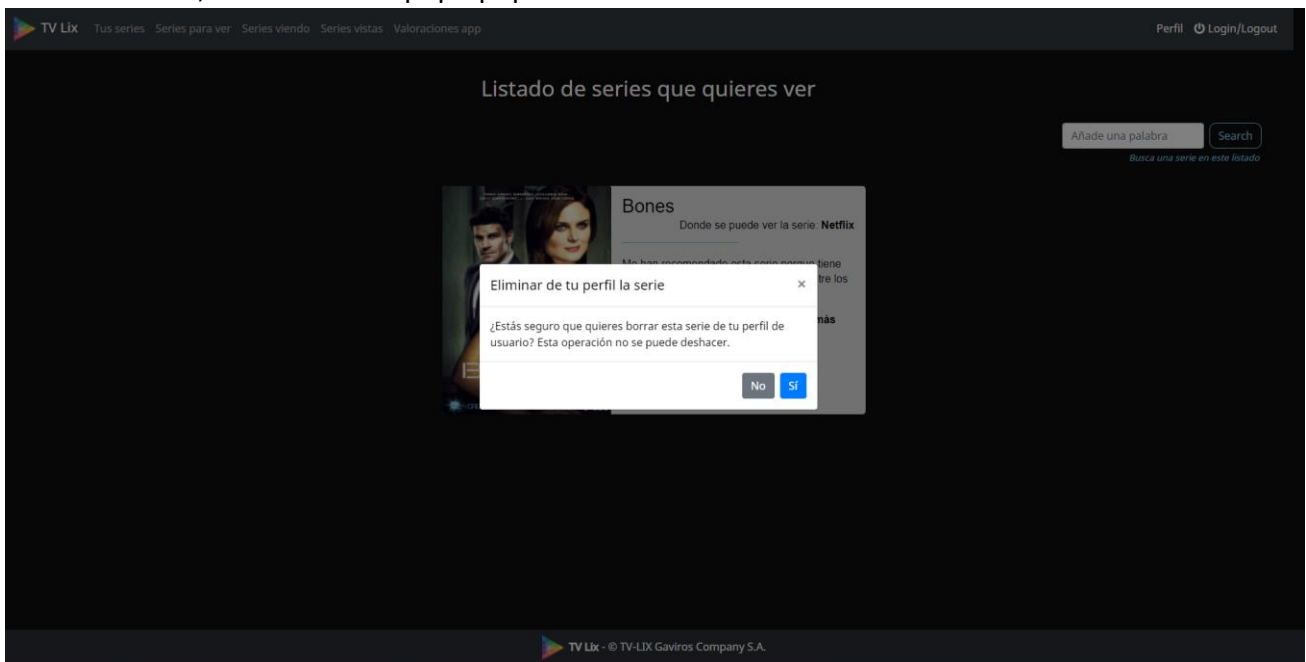
Además, ahora todos los enlaces del header redirigen a diferentes pantallas, y no como antes, que lo hacían a la de login (salvo TV Lix y Valoraciones app, que como he comentado anteriormente, son pantallas públicas):

- Tus series: con esta especie de resumen de los listados
- Series para ver: listado completo de las series que quiere ver el usuario en un futuro
- Series viendo: pantalla con aquellas series que el usuario que está viendo ahora
- Series vistas: listado con las series que ya ha visto

Las cards de cada serie, similares a las de la pantalla principal (enlace de TV Lix) contienen varios botones:

- La quiero ver
- La estoy viendo
- Ya la he visto
- Estos tres botones redirigen a su correspondiente formulario con el que el usuario incluirá la serie en este listado, siempre y cuando, no esté en alguno de los otros listados.

- Eliminar serie: elimina la serie de tu cuenta. Para ello, se requiere la confirmación del usuario, a través de un pop up que salta cuando se clicca en el botón.

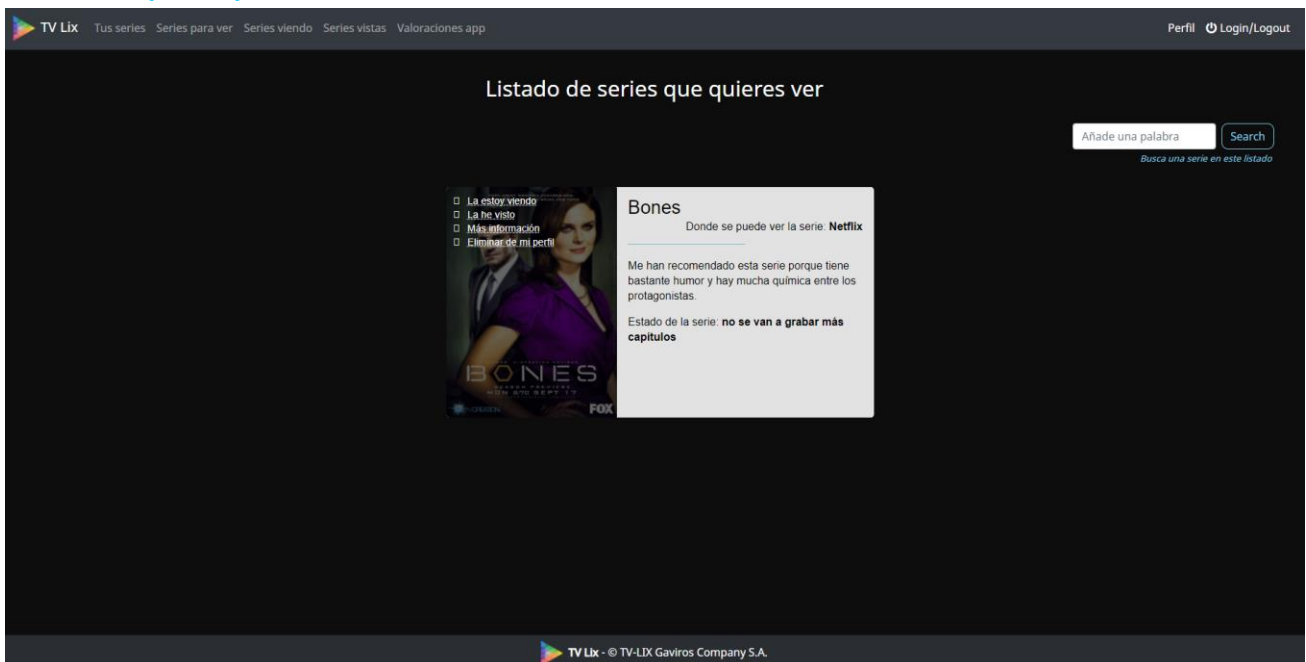


- Más información: redirige a una pantalla en la que se puede ver la información completa de la serie

6.4.- Listados de series - Formularios

En las siguientes tres pantallas, se muestran los listados de series con los que el usuario ya interactuado con anterioridad

Series que quiero ver



Este listado muestra aquellas series que el usuario quiere ver. En las cards, aparece la información que ha rellenado en el formulario correspondiente: por qué quiere verla y dónde la puede ver.

Formulario

The screenshot shows the 'Bones a la lista de series para ver' form in the TV-LIX app. The form is divided into two main sections: a left sidebar with series details and a main content area for user input.

Géneros	Crime, drama, medical
Duración de cada capítulo	60 minutos
Temporadas	12
Episodios	246
Puntuación	9.0698
Síntesis	The premise of the show is an alliance between forensic anthropologist Dr. Temperance "Bones" Brennan and FBI Special Agent Seeley Booth. Brennan is the central character and team leader of the fictional Jeffersonian Institute Medico-Legal Lab, a federal institution that collaborates with the FBI, mirroring the real-life relationship between the FBI and the Smithsonian Institution. Set in Washington, D.C., the show revolves around solving Federal legal cases by examining the human remains of possible murder victims. Dr. Brennan and her team provide scientific expertise and Special Agent Booth provides FBI criminal investigation technique. In addition to the prospective murder cases featured in each episode, the series explores the backgrounds and relationships of its characters. An important ongoing dynamic between Brennan and Booth is their disagreement about science and faith. Brennan argues for science, evidence, and atheism. Booth argues for faith, God, and the unproven. The series is also known for its dark comedic undertones to, in essence, lighten the gravity of the show's intense subject matter.

¿En qué plataforma puedes ver la serie?

☒ Netflix ☐ HBO ☐ Amazon prime video ☐ Disney Plus ☐ Otros

Netflix

¿Por qué quieres ver esta serie?

Me han recomendado esta serie porque es muy dinámica. Además, tiene bastantes golpes de humor, y un poquito de ciencia.

Guardar en series para ver

Además, puede interactuar con la serie, ya que la puede enviar a la lista de series que está viendo o a la de vistas, encontrar más información o eliminar la serie de su cuenta.

Series viendo

En este listado se pueden gestionar aquellas series que el usuario está viendo en la actualidad. En las cards, aparecen varios datos informativos: la última temporada y capítulo que ya ha visto y las primeras líneas del resumen de la misma.

The screenshot shows the 'Listado de series que estás viendo actualmente' in the TV-LIX app. The list contains two series cards: 'The Flash' and 'Outlander'.

The Flash
Episodio visto: 4 -- Temporada viendo: 1
Barry Allen is a Central City police forensic scientist with a reasonably happy life, despite the childhood trauma of a mysterious red and yellow being killing his mother and framing his father. All that changes when a massive particle accelerator accident
Estado de la serie: **grabando capítulos**
[Modifica los datos](#)

Outlander
Episodio visto: 1 -- Temporada viendo: 3
Outlander follows the story of Claire Randall, a married combat nurse from 1945 who is mysteriously swept back in time to 1743, where she is immediately thrown into an unknown world where her life is threatened. When she is forced to marry Jamie, a chivalrous...
Estado de la serie: **grabando capítulos**
[Modifica los datos](#)

Además, el usuario tiene a su disposición cuatro botones con los que trabajar:

- La he visto: para enviar esta serie al listado de series vistas, siempre y cuando no esté ya
- Más información
- Eliminar serie
- Modifica los datos: con el que aparecerá el formulario para actualizar la información la serie: la temporada y el capítulo que ha visto.

Formulario

The screenshot shows the 'The Flash' series editing form in the TV-LIX app. The form is titled 'The Flash en la lista de series viendo'. It contains a table with series details and two input fields for user selection.

Géneros	Drama
Duración de cada capítulo	60 minutos
Temporadas	7
Episodios	134
Puntuación	9.3720
Sinopsis	Barry Allen is a Central City police forensic scientist with a reasonably happy life, despite the childhood trauma of a mysterious red and yellow being killing his mother and framing his father. All that changes when a massive particle accelerator accident

Temporada que estás viendo:

Episodio que ya has visto:

Guardar en series viendo

TV Lix - © TV-LIX Gaviros Company S.A.

Series vistas

Este listado incluye las series que ya ha visto el usuario. En esta ocasión, el usuario sólo puede eliminar la serie o encontrar más información.

The screenshot shows the 'Listado de series que ya has visto' in the TV-LIX app. It displays a grid of series cards with their posters, titles, and brief descriptions.

Search bar: Search
Busca una serie en este listado

Series	Descripción
Marvel's Agents of S.H.I.E.L.D.	Es una gran serie. Esta es tu puntuación: ★★★★★. Fecha en la que terminaste de ver la serie: 2021-03-24.
Game of Thrones	Muy espectacular. Esta es tu puntuación: ★★★★★. Fecha en la que terminaste de ver la serie: 2021-03-24.
DC's Legends of Tomorrow	No me ha gustado porque es muy lenta. Esta es tu puntuación: ★★★★★. Fecha en la que terminaste de ver la serie: 2021-03-24.
Sherlock	Es una gran serie, con grandes actores. Las novelas están fielmente reflejadas en cada uno de los capítulos. Esta es tu puntuación: ★★★★★. Fecha en la que terminaste de ver la serie: 2021-03-17.

TV Lix - © TV-LIX Gaviros Company S.A.

Además, en las diferentes cards, el usuario podrá dos tipos de información que tuvo que rellenar en el formulario para incluir la serie en esta lista: una valoración del 1 al 5 mediante unas estrellas y las razones por las que le ha gustado o no.

Formulario

The Flash a la lista de series vistas

Géneros	Drama
Duración de cada capítulo	60 minutos
Temporadas	7
Episodios	134
Puntuación	9.3720
Síntesis	Barry Allen is a Central City police forensic scientist with a reasonably happy life, despite the childhood trauma of a mysterious red and yellow being killing his mother and framing his father. All that changes when a massive particle accelerator accident

Valora la serie

Cuéntanos por qué te ha gustado la serie

Guardar en series vistas

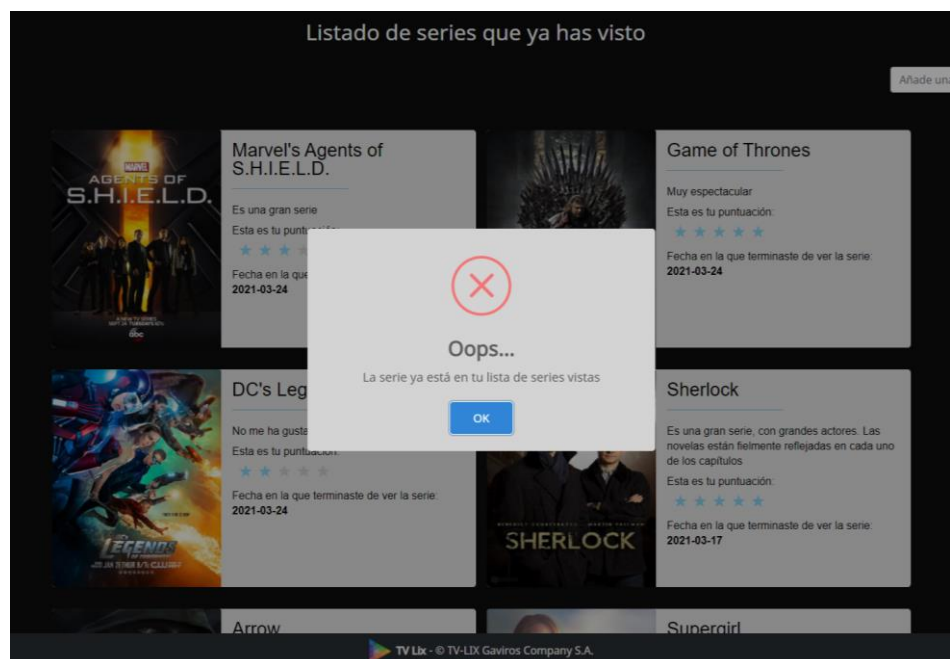
TV Lix - © TV-LIX Gaviros Company S.A.

Errores

Cuando el usuario quiere mover una serie de una lista a otra, le pueden aparecer dos tipos de error:

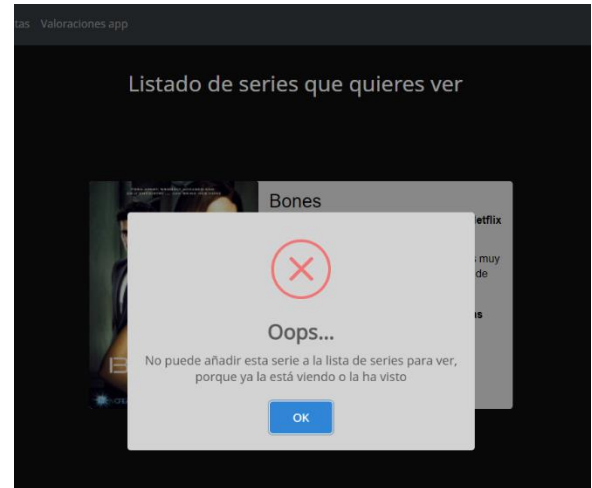
Cuando el usuario quiere pasar una serie al listado de series vistas o al de series que quiere ver, le saltará una alerta, avisándole que esa operación no se puede realizar si ya está en esa lista.

No le saltará ninguna alerta si le da al botón de 'La estoy viendo', ya que puede modificar los datos sobre la temporada y el episodio visto.



Por otra parte, también puede saltar otra alerta si el usuario quiere añadir una serie a la lista de series que quiere ver, si esta ya está en alguna de las otras dos listas.

La idea es que una serie pase por las tres listas en orden lógico: series que se quieren ver, series que se están viendo y series vistas. Puede empezar por cualquier lista, pero, por lógica, en ningún caso, se puede ir hacia “atrás”: una serie vista, no puede pasar a viendo o se quiere ver, ni una que se está viendo, puede pasar a la lista se quiere ver.



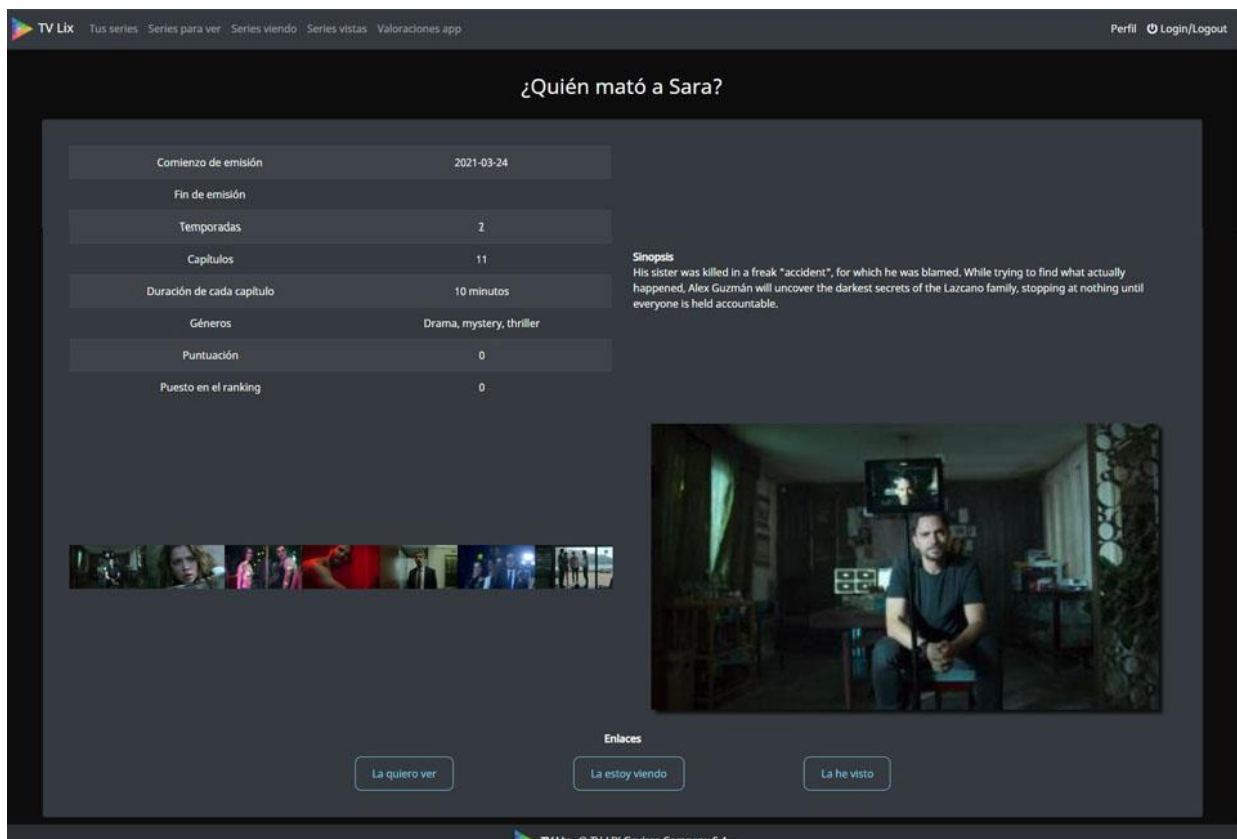
6.5.- Pantalla de detalle de una serie

Aquí se muestra la información que se recibe de la API de Episodate.

Puede haber un enlace (no en esta imagen) que abre otra ventana con más información sobre la serie. El enlace está sobre la foto principal.

Además, en la parte inferior de la pantalla, hay tres botones:

- Para mandar la serie a series que se quieren ver
- Para mandar la serie a la lista de series que se ven en la actualidad
- Para enviar la serie al listado de series vistas.



6.6.- Valoraciones APP – Formulario

Como se ha apuntado en anteriores, la pantalla en la que se pueden ver las diferentes opiniones que han dejado los usuarios sobre la aplicación, puede ser visionada por cualquier persona, haya o no haya hecho login:

The screenshot shows the 'Valoraciones de la app' (App Reviews) page. At the top, there's a navigation bar with links: 'Tus series', 'Series para ver', 'Series viendo', 'Series vistas', and 'Valoraciones app'. On the right, there's a 'Perfil' link and a 'Login/Logout' button. The main title is 'Valoraciones de la app'. Below it is a table with four columns: 'Fecha', 'Usuario', 'Comentario', and 'Valoración'.

Fecha	Usuario	Comentario	Valoración
2021-05-03 19:27	maria	Es una aplicación que podría mostrar más información, como la de los actores o los directores. Sí me gusta, cómo gestionar mis listas de series.	★★★★★
2021-05-03 19:26	jon	Su uso es muy sencillo, por lo que cualquiera puede utilizarla	★★★★★
2021-05-03 19:25	Cristina	Me encanta esta aplicación porque tiene la información exacta que necesito y me permite tener controlados aquellos capítulos que he visto y los que no.	★★★★★
2021-04-14 20:41	pau	La interfaz es limpia y agradable	★★★★★
2021-04-08 18:30	ana	Se trata de una aplicación con buenas intenciones, pero que tiene que mejorar mucho	★★★★★
2021-03-22 19:25	pau	fgdgd	★★★★★

Below the table, there's a pagination control: « Anterior 1 2 3 Siguiete ». A button labeled 'Deja tu opinión' is centered below the pagination. At the bottom, there's a footer: 'TV Lix - © TV-LIX Gaviros Company S.A.'.

Para dejar una opinión, se debe ser un usuario registrado, y debe haber hecho login en la aplicación, porque si no, esta al pinchar en el botón de Dejar tu opinión, redirigirá al usuario a la pantalla de login.

El formulario para opinar sobre TVLix es muy sencillo ya que únicamente pide una calificación del 1 al 5 y una valoración sobre lo que gusta o no de la app. Ambos campos son requeridos, por lo que, si no se rellenan, el sistema no permite crear la opinión.

The screenshot shows the 'Valora esta aplicación' (Rate this app) form. At the top, there's a navigation bar with links: 'Tus series', 'Series para ver', 'Series viendo', 'Series vistas', and 'Valoraciones app'. On the right, there's a 'Perfil' link and a 'Login/Logout' button. The main title is 'Valora esta aplicación'. Below it are five empty star icons for rating. Underneath the stars is a text input field with the placeholder 'Escribe un comentario' and 'Cuéntanos qué te parece la aplicación'. Below the input field, there's a note: 'Todos los campos son requeridos'. A button labeled 'Guardar opinión' is centered below the input field. At the bottom, there's a footer: 'TV Lix - © TV-LIX Gaviros Company S.A.'.

6.7.- Perfil – Actualizar datos de la cuenta de usuario

El Perfil es el último apartado al que puede acceder un usuario que ha hecho login, y en él, puede comprobar los datos que están guardados en la base de datos sobre su cuenta. Son todos los campos que debió rellenar para registrarse en TVLix.

TV Lix Tus series Series para ver Series viendo Series vistas Valoraciones app Perfil Login/Logout

Perfil de usuario

Nombre	Maria
Apellidos	Maria
Username	maria
Correo electrónico	maria@gmail.com
Localidad	Sangüesa
Tipo de programa	Películas
Género del programa	Aventuras

Actualiza tus datos

Elimina tu usuario

TV Lix - © TV-LIX Gavros Company S.A.

Si en algún momento, quiere actualizarlos, puede hacerlo pinchando en el botón de Actualizar datos. El formulario es muy similar al de crear cuenta. Con la salvedad, que aquí, el formulario ya está relleno por defecto, con los datos obtenidos de la base de datos.

TV Lix Tus series Series para ver Series viendo Series vistas Valoraciones app Perfil Login/Logout

Actualiza tus datos

Todos los campos son requeridos

Nombre: Maria Apellidos: Maria

Correo electrónico: maria@gmail.com Password: *****

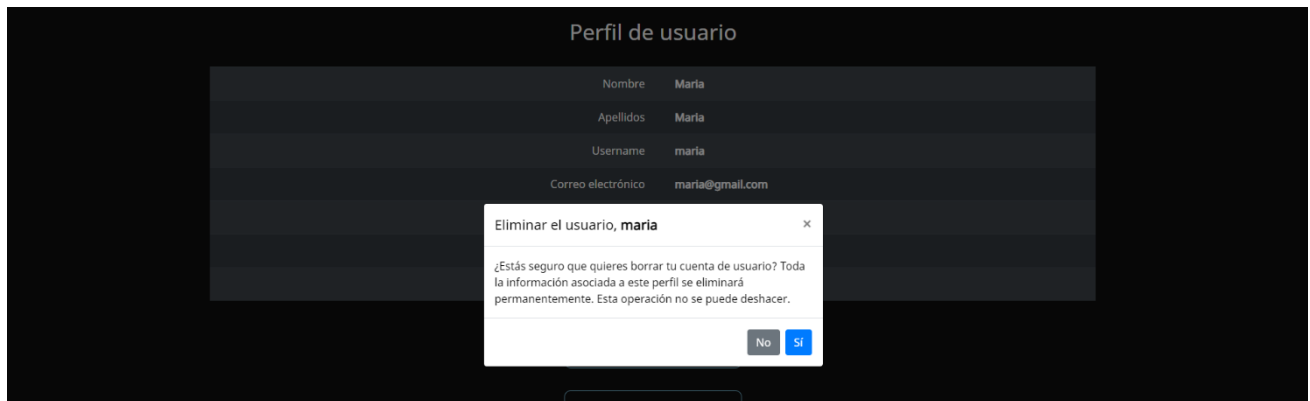
Username: maria Localidad: Sangüesa

¿Qué programa prefieres? Películas ¿Qué género te gusta más? Aventuras

Guardar datos

localhost:4200/tv/shows TV Lix - © TV-LIX Gavros Company S.A.

Si, por el contrario, quiere eliminar su cuenta de usuario, puede hacerlo pinchando en el botón de Elimina tu usuario. Si realiza esta acción, se mostrará un pop up de confirmación por si el usuario se lo quiere pensar.



Una vez eliminado el usuario, el sistema redirige al usuario ya eliminado de la base de datos a la pantalla de login, con la esperanza de que se vuelva a registrar en TVLix.

7.- Bibliografia

ARQUITECTURA

- MVC: <http://brianhernandezg.blogspot.com/p/metodologia-de-trabajos-con-mvc.html>
- XAMMP: <https://sites.google.com/site/portafoliovicenciosr/poll>
- Git: <https://www.hostinger.es/tutoriales/que-es-github#Que-es-Git>

API REST

- API Rest: <https://www.redhat.com/es/topics/api/what-is-a-rest-api>
- Episodate: <https://www.episodate.com/api>

BASE DE DATOS

- Base de datos relacional: https://www.um.es/geograf/sigmur/temariohtml/node63_mn.html
- Lenguaje SQL: https://www.um.es/geograf/sigmur/temariohtml/node63_mn.html
- Motor MySQL: <https://neoattack.com/neowiki/mysql/>

BACKEND

- Sprint tool suite:
- SpringBoot: <https://www.dataart.com.ar/news/como-desplegar-una-aplicacion-springboot-utilizando-docker/>
- Maven: <https://www.javiergarzas.com/2014/06/maven-en-10-min.html>
- Hibernate: <https://ifgeekthen.everis.com/es/que-es-java-hibernate-por-que-usarlo.s.f>
- Postman: <https://openwebinars.net/blog/que-es-postman/>
- Microservicios: <http://sinbugs.com/como-crear-un-microservicio-o-servicio-web-rest-con-spring-boot-1/>

FRONTEND

- Visual Studio Code: <https://www.softzone.es/programas/utilidades/visual-studio-code/>
- Angular: <https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/>
- Bootstrap: <https://raiolanetworks.es/blog/bootstrap/#:~:text=Bootstrap%20es%20un%20kit%20de,%2C%20cuadros%2C%20botones%2C%20formularios%E2%80%A6>
- SweetAlert: <https://blog.endeos.com/notificaciones-al-usuario-con-jquery-sweet-alert/#:~:text=Con%20Sweet%20Alert%20conseguimos%20dar,plugin%20de%20muchas%20formas%20diferentes>
- Ngx-pagination: <http://michaelbromley.github.io/ngx-pagination/#/>
- Estructura Angular: <https://lacasadelprogramador.com/estructura-proyecto-angular/>