

# MUSIKOS

Músicos es la versión beta de una App pensada para que los músicos puedan crear un perfil y buscar otros músicos de su zona en función a instrumentos, intereses...

Este documento se va a dividir en cinco partes.

## PARTE 1: DIAGRAMAS

Diagrama ER:

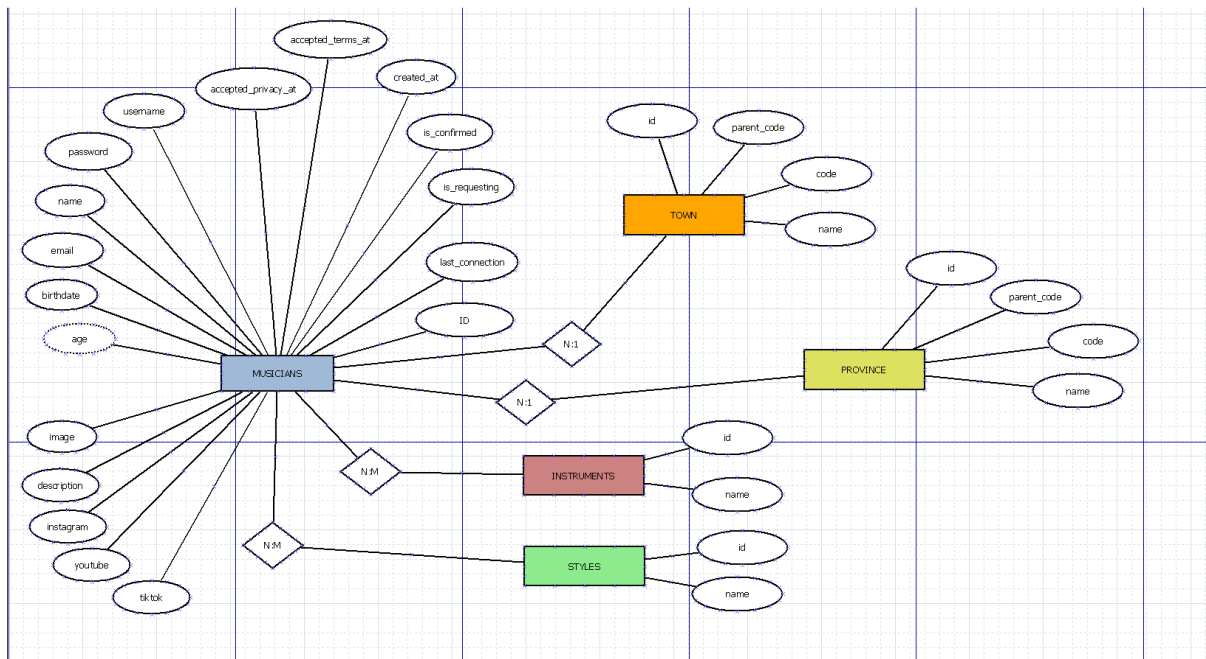


Diagrama de secuencia:

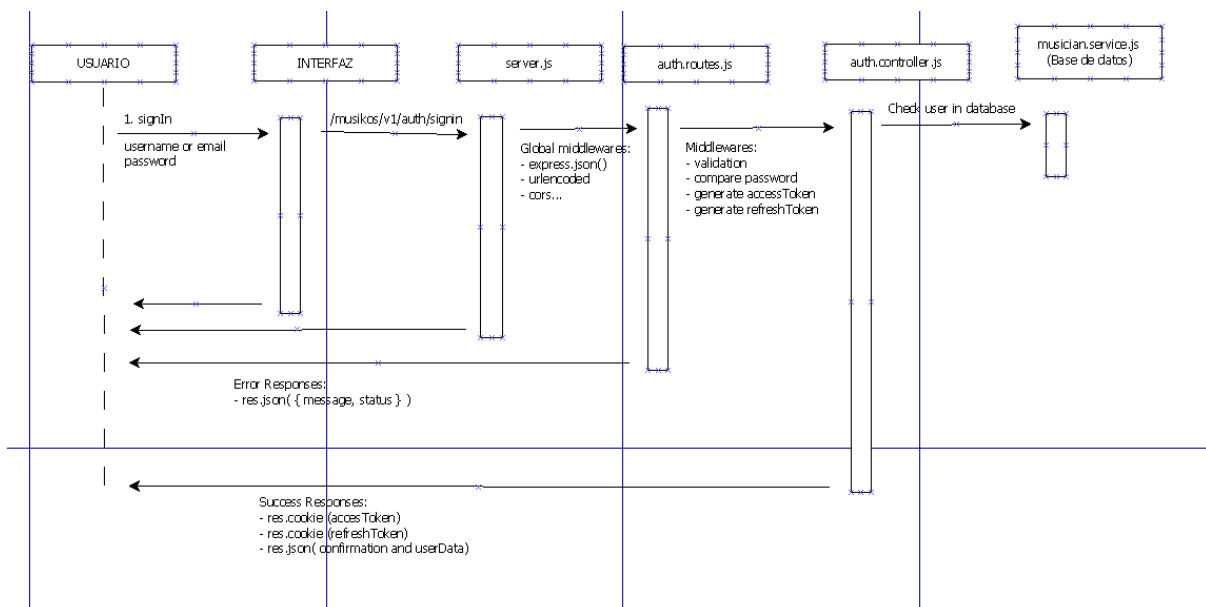
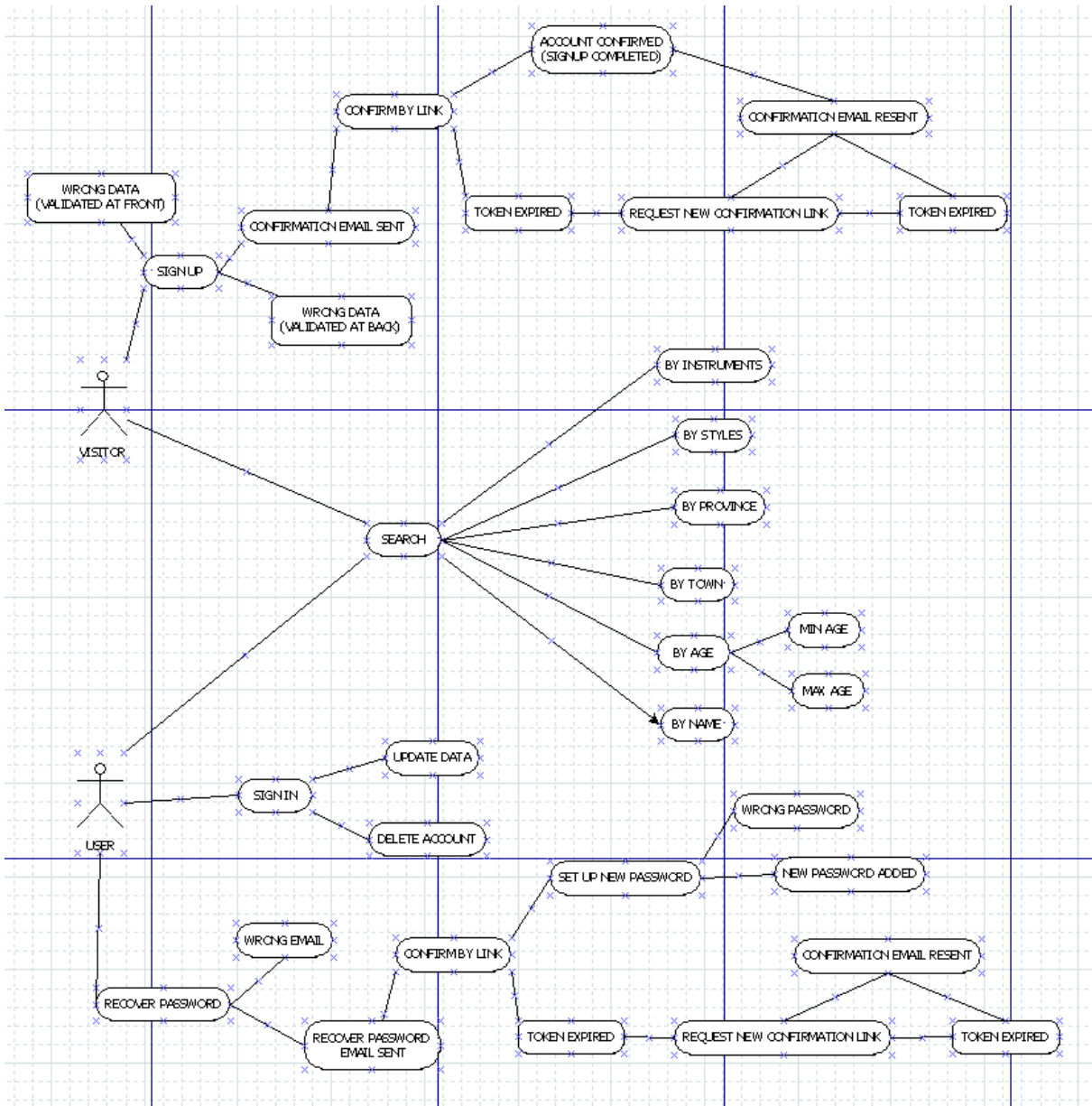


Diagrama de casos:



## PARTE 2: FUNCIONALIDADES

### 1. RESUMEN DE FUNCIONALIDADES PRINCIPALES

- Cualquier visitante puede acceder a la app y buscar otros músicos.
- Cualquier visitante puede crear una cuenta (sign up) a través de un correo electrónico.
- Cualquier usuario puede acceder a su cuenta (sign in) a través de su correo electrónico o username.
- Cualquier usuario puede ser encontrado por otros músicos.
- Cualquier usuario puede modificar ciertos datos de su cuenta en el “dashboard”.
- Cualquier usuario puede eliminar su cuenta.

### 2. PROCESO DE REGISTRO

El proceso de registro es sencillo y seguro. Para llevarlo a cabo se necesita tener acceso al email utilizado en el formulario. Estos son los pasos:

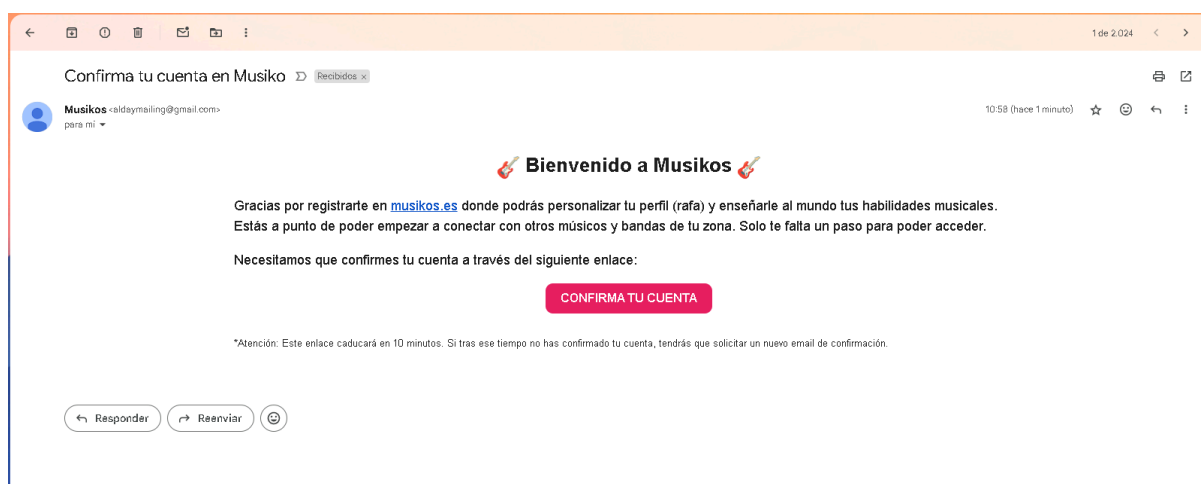
- El usuario rellena el formulario
  - Validación online: Los datos se validan según el usuario va escribiendo en los diferentes inputs.
  - Validación al enviar: Los datos se validan antes de enviar los datos al servidor.
- Recepción de datos en el back
  - Validación desde el backend.
  - Encriptación de la contraseña (hash)
  - Generación del token de confirmación
  - Generación y envío de email de confirmación (con token)
  - Creación del usuario
  - Respuesta al front indicando que tiene un correo
- Confirmación vía email (el usuario pulsa con el token en el email)
  - Validación del token
  - Cambio del campo “is\_confirmed” en el usuario (tabla “musicians”)
  - Redirección al front con respuesta exitosa o error (ej. token caducado)
  - Si el token ha caducado, en el front se nos mostrará un enlace con el que podremos mandar un nuevo email de confirmación.

Como en la presentación, en principio, no se podrá usar el servicio de email, dejo a continuación capturas de pantalla de ambos mails, los cuales están generados con html y css.

1. Al rellenar todos los campos correctamente nos aparecerá este mensaje

The screenshot shows the 'ZONA DE ACCESO' (Access Zone) of the Musikos website. It features two main sections: 'ACCEDE A TU CUENTA' (Log in to your account) and 'REGÍSTRATE AHORA' (Register now). The registration form includes fields for Email, Contraseña (Password), Fecha de nacimiento (Date of birth) with sub-fields for Día, Mes, and Año, and checkboxes for 'Acepto los Términos y condiciones' and 'Acepto la Política de privacidad'. A 'REGISTRARSE' button is at the bottom. A pink modal overlay with a close button (X) is displayed in the center, containing the text: '¡CONFIRMA TU CUENTA! Te hemos enviado un link de confirmación a rafaldaysparejo@gmail.com. Por favor, revisa tu correo y sigue el enlace'. The footer shows 'MUSIKOS © RAFA ALDAY'.

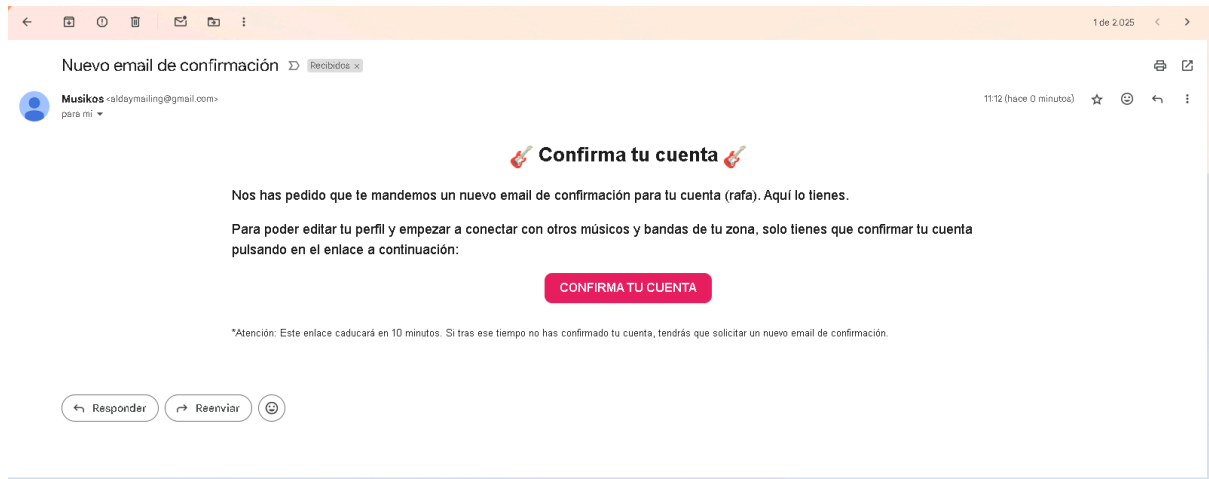
2. Este es el email que recibiremos



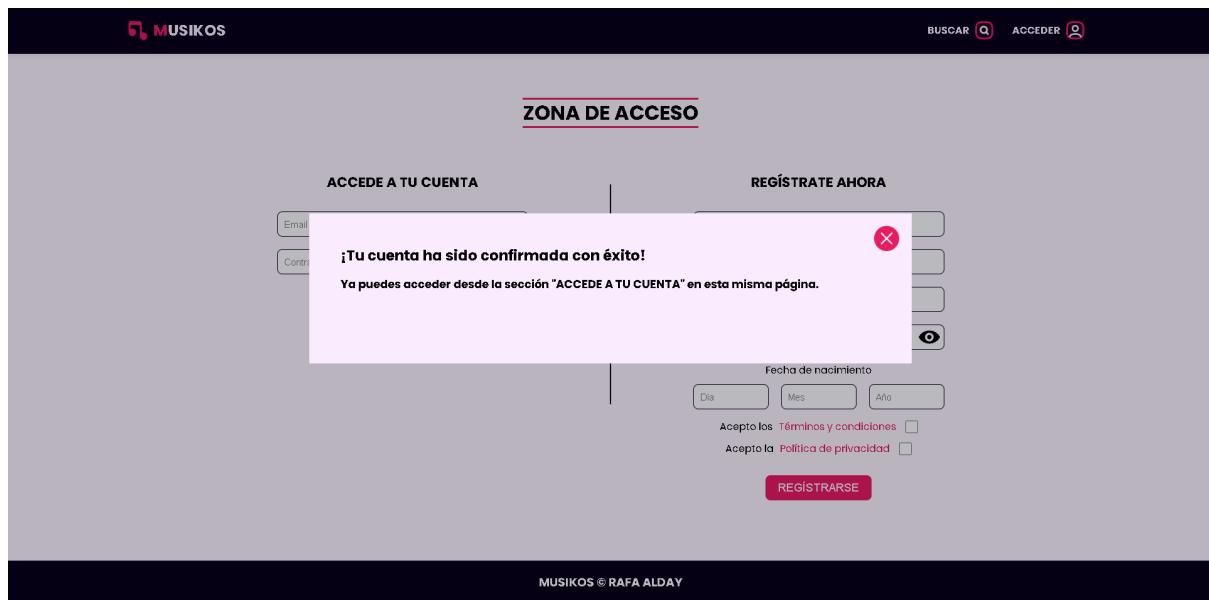
3. Si el token está caducado, nos redirigirá a esta página

This screenshot shows the same registration page as before, but with a different modal overlay. The modal, titled 'Enlace caducado', contains the text: 'El enlace de confirmación ha caducado. Si necesitas que te mandemos un nuevo email de confirmación pulsa en el siguiente enlace:'. Below the text is a pink button labeled 'REENVIAR EMAIL DE CONFIRMACIÓN'. The background registration form and footer are identical to the previous screenshot.

4. Si pulsamos en “reenviar email de confirmación”, se enviará un nuevo email:



5. Si todo es correcto en cualquiera de los email, nos redirigirá a la app con en mensaje de confirmación:



### 3. TOKENS DE REGISTRO Y RECUPERACIÓN

Se podría decir que hay dos tipos de tokens. Por una parte los tokens de acceso y recuperación, los cuales se envían como parámetro en los email de confirmación de registro o cambio de contraseña. Por otro lado tenemos tokens de acceso (accessToken y refreshToken), de los que hablaré más adelante.

Los tokens de acceso y recuperación tienen una duración muy corta (10 minutos) y en ellos intento incluir la menos cantidad de información posible. Simplemente sirven para comprobar que el usuario realmente tiene acceso al email.

## 4. PROCESO DE ACCESO

Una vez el usuario se ha registrado y quiere acceder a la aplicación, simplemente deberá introducir, bien su “username”, bien su “email” y después la contraseña. Ocurrirán varias cosas:

- Validación de los datos introducidos
- Comparación de la contraseña introducida y el hash
- Generación del “accessToken” y “refreshToken”
- Envío de los tokens generados por cookies (httpOnly)
- Envío de respuesta en json
- Si hay algún error, se enviará la respuesta con status correspondiente

Si el proceso anterior es correcto, en el front ocurrirán también ciertos procesos:

- Se recibirán los token en las cookies
- La respuesta recibida se incluirá en “sessionStorage.auth”
- Se actualizará el estado “isLoggedIn” a true.
- La barra de navegación añadirá las pestañas de “tu cuenta” (username) y “salir” (para cerrar sesión).

## 5. SISTEMA DE TOKENS DE ACCESO

Como hemos comentado antes, los token que se generan en el “signIn”, servirán para mantener al usuario autenticado de manera segura. Cuando se hagan solicitudes protegidas, se habilitará la opción credentials: “Include” en los fetch, la cual nos permitirá enviar el token vía http de manera segura al backend sin exponerlo, para allí poder comprobarlo. Esto da un mayor seguridad.

- “accessToken”: Este token tiene una duración más corta (1 hora) y es el que se utiliza para verificar si el usuario está logueado o no. Es el token que se comprueba en las rutas protegidas.
- “refreshToken”: Este token tiene una duración más larga (7 días). El proceso es el siguiente:
  - Si el “accessToken” caduca, el front solicitará la renovación del mismo a través del “refreshToken”.
  - Si el “refreshToken” es válido el back generará un nuevo “accessToken” y lo enviará al front.
  - Si el “refreshToken” también está caducado o es invalido, no se permitirá el acceso y se cerrarán sesión al usuario si estaba abierta.

## 6. IS\_REQUESTING FIELD

Comentar en este punto que he añadido un campo llamado “is\_requesting” a los usuarios. Dicho campo es un booleano y se utiliza para limitar las peticiones al usuario si ya está realizando alguna. Por ejemplo, si un usuario solicita la renovación de su contraseña, en el momento que se envía el email de recuperación su estado pasa a “is\_requesting: true”. Cuando el token del email de recuperación expire, el estado pasará automáticamente a “is\_requesting: false”. De esta manera se limita que el usuario no pueda hacer infinitas solicitudes si ya las tiene en marcha.

## 7. RECUPERACIÓN DE LA CONTRASEÑA

Tenemos la funcionalidad de recuperar la contraseña si el usuario la ha olvidado. Esto se puede hacer a través de un enlace justo debajo del formulario de “signIn”. El proceso es el siguiente:

- El usuario introduce email o username para recibir un email de recuperación.
- El back hace las validaciones pertinentes y, si todo está correcto, enviará un email de recuperación que incluye un token para dicho fin.
- Una vez el usuario pulse el enlace del email, si el token no ha expirado, se le redirigirá al front donde tendrá un formulario en el que podrá introducir su nueva contraseña. Antes de poder enviarla, dicha contraseña se validará desde el front para que tenga el formato correcto.
- Recibida la contraseña se validará en el back, se encriptará, se actualizará en el usuario y finalmente se responderá al front.
- Si hubiera cualquier error en este proceso, se da la respuesta adecuada con el status correspondiente.

\*\*\* Los emails y mensajes tanto de error como de confirmación son muy similares a los de el proceso de confirmación de creación de la cuenta. No muestro capturas para no extenderme demasiado.

## 8. ZONA DE USUARIO

Una vez el usuario ha hecho login, podrá acceder a su cuenta a través de la pestaña que habrá aparecido en la barra de navegación con su “username”. Esta sección está dividida en tres partes:

- “EDITA TU CUENTA”: Para editar datos de la cuenta como tal (email y contraseña).
- “EDITA TU PERFIL”: Aquí pretendo que el usuario pueda editar todo su perfil de músico, pero en esta versión beta solo me ha dado tiempo a editar el username.
- “COMUNICACIONES”: Aquí el usuario también podrá ver sus comunicaciones (al crear una cuenta se genera por defecto una de bienvenida).

## 9. BÚSQUEDAS

Cualquier visitante de la web puede buscar músicos. He creado lo que creo que es un buscador bastante completo y complejo para poder filtrar los resultados de muchas maneras.

- Filtrar por estilo/estilos
- Filtrar por instrumento/instrumentos
- Filtrar por provincia
- Filtrar por región (solo si se ha indicado provincia)
- Filtrar por edad (edad mínima y edad máxima)
- Filtrar por nombre (usando el método iLike de sequelize que busca coincidencias incluso parciales)
- Podemos combinar todos los filtros anteriores de la manera que queramos.

## PARTE 3: DOCUMENTACIÓN TÉCNICA

En esta sección voy a indicar brevemente que gestionan los distintos directorios y ficheros de cada repositorio (server y client).

### 1. SERVER

Antes de empezar, creo que es interesante definir las 5 grandes categorías que tiene el proyecto ya que, se repetirán en controladores, rutas y servicios

- **auth:** Proceso de registro autenticación del usuario
- **comm:** Comunicaciones
- **generic:** Datos estáticos como listado de estilos, instrumentos, entre otros.
- **legal:** Política de privacidad y condiciones de uso.
- **musicians:** Todo lo referente a los perfiles de músicos como tal.

Una vez comentado esto, repasemos los directorios:

- **/config** → Contiene diferentes ficheros de configuración de dependencias como las bbdd o variables de entorno entre otros.
- **/controllers** → Contiene todos los controladores del proyecto
- **/databases** → Contiene las conexiones y seeds a las bases de datos de MongoDB y MySQL.
- **/docs** → Ficheros de documentación de swagger, jdoc (vistas) y archivos.
- **/logs** → Contiene los archivos .log que se generan con winston.
- **/middlewares** → Diferentes middlewares reutilizables como por ejemplos de validación o tokens. Especial importancia a "error.middleware.js" que gestionará todos los errores.
- **/models** → Todos los modelos de MongoDB (mongoose) y MySQL (sequelize). En el caso de MySQL también tendremos las asociaciones.
- **/routes** → Ficheros con rutas divididas en las cinco categorías mencionadas antes.
- **/services** → En services están todas las operaciones relacionadas con las bases de datos, separadas en función a modelos.
- **/test** → Aquí tendremos archivos referentes a jmeter y jest
- **/utils** → Diferentes funciones reutilizables como por ejemplo "mailing.js" o "validate.js" entre otras. Especial importancia al directorio errors, donde se gestionan las clases para errores y logs.
- **/views** → Aquí hay ficheros con html/css con las vistas tanto de email como de la información legal.

### 2. CLIENT

- **/cypress** → Archivos referentes a pruebas con cypress
- **/public** → De momento solo está el icono de la app
- **/selenium** → Archivos referentes a pruebas con selenium
- **/src**
  - **/assets** → imágenes, fuentes...
  - **/components** → Componentes individuales y reutilizables
  - **/context** → AuthContext y SearchContext
  - **/Pages** → Home, DashBoard, Login y Search. Cada una con sus respectivos componentes y css.
  - **/utils** → Funciones reutilizables como useFetch.jsx o validate.js.
  - Resto de archivos → App.jsx, main.jsx...
- **Resto de archivos** → cypress.config.js, index.html, package.json...



## PARTE 4: TESTING Y DOCUMENTACIÓN

### 1. JEST EN EL BACKEND

Para realizar las pruebas con jest debemos introducir “npm run test” una nueva terminal.

```
PS C:\Users\Tardes\Desktop\DAW\DAW_musikos\musikos_server> npm run test

> musikos_server@1.0.0 test
(node:17064) ExperimentalWarning: VM Modules is an experimental feature and might change at any time
(node:17064) ExperimentalWarning: VM Modules is an experimental feature and might change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
PASS test/regex.test.js
PASS test/validatePassword.test.js
(node:10352) ExperimentalWarning: VM Modules is an experimental feature and might change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
PASS test/encryptPassword.test.js

Test Suites: 3 passed, 3 total
Tests: 15 passed, 15 total
Snapshots: 0 total
Time: 0.961 s
Ran all test suites.
npm notice
npm notice New major version of npm available! 10.9.0 -> 11.2.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.2.0
npm notice To update run: npm install -g npm@11.2.0
npm notice
PS C:\Users\Tardes\Desktop\DAW\DAW_musikos\musikos_server> |
```

### 2. JMETER EN EL BACKEND

Para realizar el test de autenticación de 50 usuarios con JMeter, he dejado una serie de ficheros en el directorio /test/jmeter donde ya tenemos el archivo users.csv así como el archivo del grupo de hilos en .jmx exportado.

Primero abriremos jmeter y el archivo .jmx mencionado. Nos aseguramos de que el archivo csv este importado correctamente:

Configuración del CSV Data Set	
Nombre:	Configuración del CSV Data Set
Comentarios	
Configura el Data Source de CSV	
Nombre de Archivo:	C:/Users/Tardes/Desktop/DAW/DAW_musikos/musikos_server/test/jmeter/users.csv
Codificación del fichero:	
Nombres de Variable (delimitados por coma):	usuario, pwd

Simplemente debemos ir a “Ver árbol de resultados” e inicializar la prueba (botón de play)

Resultados: Tendremos dos tipos, los usuarios que si existen y los que no. Las respuestas que recibiremos en cada caso son:

- Existe: {"verified":true,"user":{"id":3,"username":"pepitogrillo","email":"[pepito@grillo.com](mailto:pepito@grillo.com)"}}
- No existe: {"message":"Solicitud incorrecta. Verifica los datos enviados.","status":400}



### 3. JSDOC EN EL BACKEND

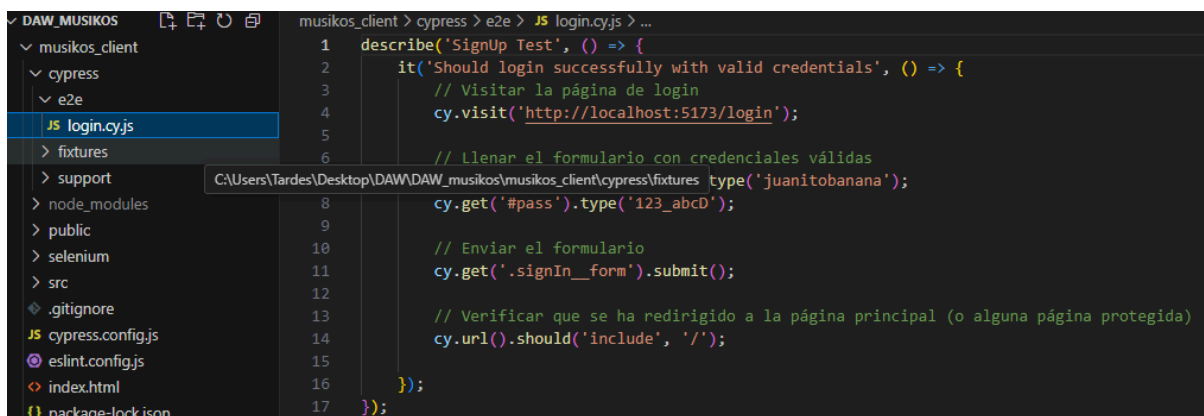
Para poder ver la UI de la documentación de funciones con jsDoc, debemos ir a “/docs/jsdoc” y abrir con live server el archivo index.html

Las funciones que he incluido son “comparePassword” y “encryptPassword”, ambas en “/utils/bcrypt.js”

En el futuro agregaré más funciones a esta documentación.

### 4. CYPRESS EN EL FRONTEND

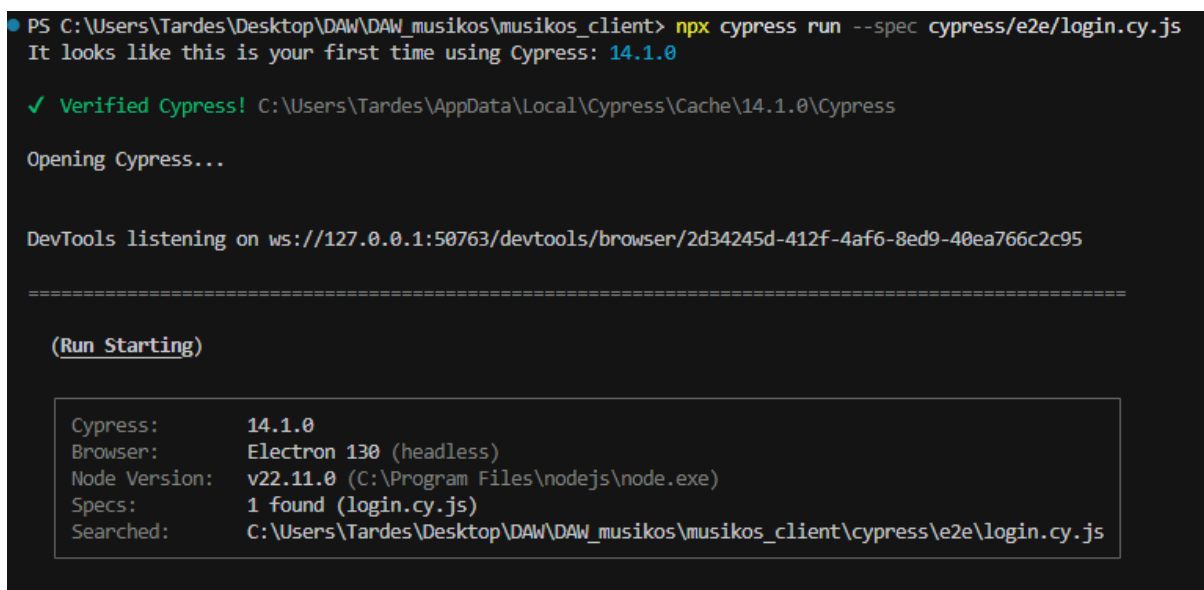
He probado el login con cypress. El archivo clave está en “/crypress/e2e/login.cy.js”



```
1 describe('SignUp Test', () => {
2   it('Should login successfully with valid credentials', () => {
3     // Visitar la página de login
4     cy.visit('http://localhost:5173/login');
5
6     // Llenar el formulario con credenciales válidas
7     cy.type('username', 'juanitobanana');
8     cy.get('#pass').type('123_abcd');
9
10    // Enviar el formulario
11    cy.get('#sign_in_form').submit();
12
13    // Verificar que se ha redirigido a la página principal (o alguna página protegida)
14    cy.url().should('include', '/');
15  });
16 }
17 });
```

Básicamente lo que hace es abrir la página del login e introducir los datos de un usuario.

Para inicializar la prueba debemos ejecutar “npx cypress run --spec cypress/e2e/login.cy.js” en una nueva terminal, teniendo arrancado vite. Nos aparecerá lo siguiente:



```
PS C:\Users\Tardes\Desktop\DAW\DAW_musikos\musikos_client> npx cypress run --spec cypress/e2e/login.cy.js
It looks like this is your first time using Cypress: 14.1.0

✓ Verified Cypress! C:\Users\Tardes\AppData\Local\Cypress\Cache\14.1.0\Cypress

Opening Cypress...

DevTools listening on ws://127.0.0.1:50763/devtools/browser/2d34245d-412f-4af6-8ed9-40ea766c2c95

=====

(Run Starting)

Cypress:      14.1.0
Browser:      Electron 130 (headless)
Node Version: v22.11.0 (C:\Program Files\nodejs\node.exe)
Specs:        1 found (login.cy.js)
Searched:     C:\Users\Tardes\Desktop\DAW\DAW_musikos\musikos_client\cypress\e2e\login.cy.js
```

Running: login.cy.js

(1 of 1)

### SignUp Test

✓ Should login successfully with valid credentials (1321ms)

1 passing (3s)

### (Results)

```
Tests:      1
Passing:    1
Failing:    0
Pending:    0
Skipped:    0
Screenshots: 0
Video:      false
Duration:   2 seconds
Spec Ran:   login.cy.js
```

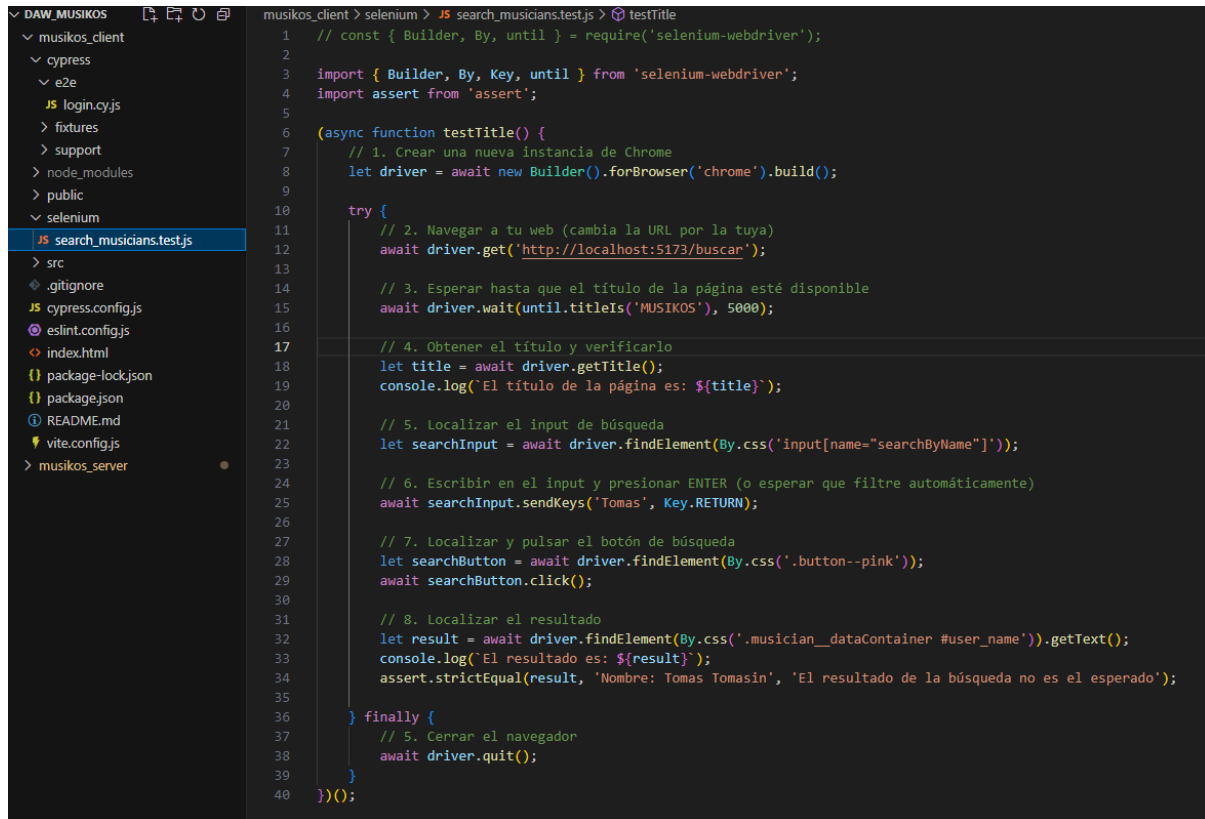
### (Run Finished)

Spec		Tests	Passing	Failing	Pending	Skipped
✓ login.cy.js	00:02	1	1	-	-	-
✓ All specs passed!	00:02	1	1	-	-	-

PS C:\Users\Tardes\Desktop\DAW\DAW\_musikos\musikos\_client> █

## 5. SELENIUM EN EL FRONTEND

He probado la función de búsqueda con selenium. El archivo clave está en “/selenium/search\_musicians.test.js”

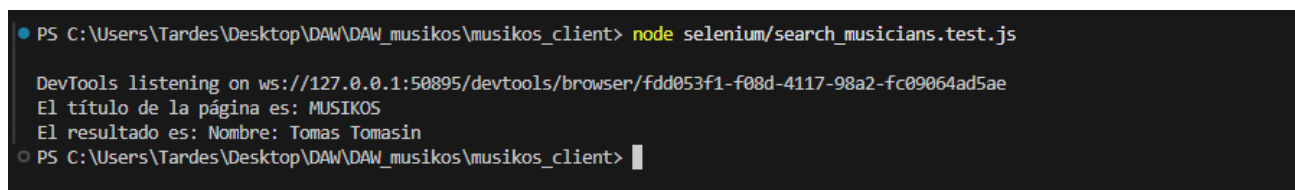


```
1 // const { Builder, By, until } = require('selenium-webdriver');
2
3 import { Builder, By, Key, until } from 'selenium-webdriver';
4 import assert from 'assert';
5
6 (async function testTitle() {
7   // 1. Crear una nueva instancia de Chrome
8   let driver = await new Builder().forBrowser('chrome').build();
9
10  try {
11    // 2. Navegar a tu web (cambia la URL por la tuya)
12    await driver.get('http://localhost:5173/buscar');
13
14    // 3. Esperar hasta que el título de la página esté disponible
15    await driver.wait(until.titleIs('MUSIKOS'), 5000);
16
17    // 4. Obtener el título y verificarlo
18    let title = await driver.getTitle();
19    console.log("El título de la página es: ${title}");
20
21    // 5. Localizar el input de búsqueda
22    let searchInput = await driver.findElement(By.css('input[name="searchByName"]'));
23
24    // 6. Escribir en el input y presionar ENTER (o esperar que filtre automáticamente)
25    await searchInput.sendKeys('Tomas', Key.RETURN);
26
27    // 7. Localizar y pulsar el botón de búsqueda
28    let searchButton = await driver.findElement(By.css('.button--pink'));
29    await searchButton.click();
30
31    // 8. Localizar el resultado
32    let result = await driver.findElement(By.css('.musician__dataContainer #user_name')).getText();
33    console.log("El resultado es: ${result}");
34    assert.strictEqual(result, 'Nombre: Tomas Tomasin', 'El resultado de la búsqueda no es el esperado');
35  } finally {
36    // 5. Cerrar el navegador
37    await driver.quit();
38  }
39 }
40 )();
```

Básicamente lo que hace es buscar un usuario por nombre (Tomás) y debería encontrar a “Tomas Tomasín”.

Para inicializar la prueba debemos ejecutar “node selenium/search\_musicians.test.js” en una nueva terminal, teniendo vite arrancado.

Nos aparecerá este mensaje:



```
PS C:\Users\Tardes\Desktop\DAW\DAW_musikos\musikos_client> node selenium/search_musicians.test.js
DevTools listening on ws://127.0.0.1:50895/devtools/browser/fdd053f1-f08d-4117-98a2-fc09064ad5ae
El título de la página es: MUSIKOS
El resultado es: Nombre: Tomas Tomasin
PS C:\Users\Tardes\Desktop\DAW\DAW_musikos\musikos_client> |
```

## PARTE 5: DIARIO DE SPRINTS

He organizado todo el trabajo en sprints. En cada uno, he ido anotando objetivos y procedimientos que he realizado. En este “diario de a bordo” se puede ver el proceso de cómo he ido trabajando.

Todo esto se puede encontrar en el “README.md”, pero me ha parecido interesante añadirlo a esta documentación también:

### 1. SPRINTS EN EL FRONTEND

#### ## 5o SPRINT (FRONTEND)

##### ### RESUMEN DE OBJETIVOS:

Mi intención en este sprint es tener las funcionalidades mínimas que se piden en el trabajo, para asegurar así cumplir todos los puntos.

1. Realización de pruebas con cypress para el login.  
(`npx cypress run --spec cypress/e2e/login.cy.js`)
2. Realización de pruebas con selenium para el buscador.  
(`node selenium/search_musicians.test.js`)
3. DashboardComms.jsx (componente)
  - Nos permite acceder y mostrar los comunicados al usuario
4. Gestión de eliminación de usuario

#### ## 4o SPRINT (FRONTEND)

##### ### RESUMEN DE OBJETIVOS:

He creado la página de Dashboard, donde el usuario podrá modificar sus datos. Debido a la falta de tiempo, he decidido poner solo algunas funciones básicas e intentar optimizar para cumplir con los objetivos que se piden en el trabajo, dejando funcionalidades para la versión 2.

1. Diseño de Dashboard.jsx (página)
  - \* Contenedor para DashboardAccount y DashboardProfile.
  - \* Al cargarse en la página, recibirá los datos del usuario.
2. DashboardAccount.jsx (componente)
  - \* Aquí se podrán modificar los datos de la cuenta como tal (email y contraseña)
3. DashboardProfile.jsx (componente)
  - \* Aquí se podrán modificar los datos del perfil de músico. De momento solo he implementado actualizar el "username" por lo indicado en el resumen.

#### ## 3er SPRINT (FRONTEND)

### ### RESUMEN DE OBJETIVOS:

He pre diseñado la página de búsqueda de músicos (Search.jsx). Se divide en dos componentes: SearchForm y SearchList.

#### 1. Pre diseño de Search.jsx (página)

- \* Contenedor para SearchForm y SearchList.
- \* Se apoya en un contexto (searchContext) utilizado tanto para obtener los listados generales de instrumentos, estilos, provincias... que se usarán para los inputs como para gestionar el listado final de usuario encontrados o posibles errores.

#### 2. Pre diseño SearchForm.jsx

- \* Incluye inputs totalmente creados por mí con css y acordes a la estética general de la aplicación.
- \* Se puede filtrar por instrumentos, estilos, provincia, ciudad, edad mínima y máxima y nombre.
- \* Custom hook useSearch utilizado para manejar la información de dichos filtros y finalmente conformar el query param final que se incluye en el fetch para obtener los datos del back.

#### 3. Pre diseño de SearchList.jsx

- \* Nada más cargarse el componente solicita al back un listado completo (sin filtrar), que es lo que aparece en la búsqueda. Este listado se guarda en sessionStorage para no hacer la búsqueda cada vez que se actualiza la página.
- \* Mostrará un listado con los siguientes datos por usuario: username, nombre, edad, estilos, instrumentos, provincia y ciudad (si la hay).

4. Queda pendiente: Pulir los estilos de todos los componentes y elementos, así como posiblemente añadir el dato "género" del usuario si finalmente lo pongo.

### ## 2o SPRINT (FRONTEND)

### ### RESUMEN DE OBJETIVOS:

He creado todo lo necesario para el signín en el front, así como comprobado la funcionalidad con el back. He ajustado componentes para que sean más reutilizables.

#### 1. Diseño del componente Signin.

(Acceso a cuenta)

#### 2. Diseño del componente RecoverPassModal.

(Modal para reestablecer la contraseña)

#### 3. Reajuste de componentes errorModal y succesModal.

(Errores y respuestas de solicitudes http)

#### 4. Revisión de lógica en la página "Login".

(Gestión de "query params" y ventanas modales)

#### 5. Creación de fichero "errorMessages" reutilizables.

(Traducir errores provenientes de "query params")

#### 6. Reajuste de "useFetch" para gestionar mejor los errores.

(Errores provenientes del back, errores del navegador/conexión)

7. Aplicación de estilos.

## ## 1er SPRINT (FRONTEND)

### ### RESUMEN DE OBJETIVOS:

Al igual que en el back, en el primer sprint del front me he centrado en configuraciones y disposiciones globales que agilizarán mucho los siguientes sprints, realizando solo la funcionalidad de signUp, que queda operativa y sincronizada con el back.

1. Inicialización del proyecto (react/vite).
2. Diseño de estética general (colores, logo, tipografía...)
3. Disposición inicial de las páginas y componentes reutilizables.
4. Función de signUp sincronizada con backend.
5. Diseño de ErrorModal (componente) reutilizable.
6. Diseño de SuccessModal (componente) reutilizable.
7. Diseño de LegalModal (componente) para mostrar los términos y la política de privacidad.
8. Creación de useFetch para las peticiones.
9. Creación de customFetch para peticiones desde js puro.
10. Creación de fichero validate.js para validaciones.
11. Diseño y creación de spinner de carga (img + modal).

## 2. SPRINTS EN EL BACKEND

### ## 5o SPRINT (BACKEND)

#### ### RESUMEN DE OBJETIVOS:

Mi intención en este sprint es tener las funcionalidades mínimas que se piden en el trabajo, para asegurar así cumplir todos los puntos.

1. Realizadas pruebas unitarias con Jest con dos funciones (encryptPassword y validatePassword)
2. Endpoints "comms" para comunicaciones -> Podemos enviar comunicados al usuario. Al crear una cuenta, por defecto se crea un comunicado de bienvenida.
  - comm.routes.js
  - comm.controller.js
  - comm.service.js
3. Endpoint "delete-account" para eliminar la cuenta

### ## 4o SPRINT (BACKEND)

#### ### RESUMEN DE OBJETIVOS:



En este sprint mi objetivo principal en el back era crear todos los endpoints para la modificación de datos por partes del usuario en su perfil y cuenta. He implementado solo algunas ya que, debido a la falta de tiempo, creo que es mejor priorizar en completar los requisitos que se piden para el trabajo.

1. Endpoint y controlador para "musicians/restricted-data" -> Nos proporciona toda la información del usuario. Para acceder, se debe verificar el token de las cookies ya que devolverá TODA la información del usuario.
2. Endpoint y controlador para "musicians/:username" -> Nos proporciona información pública del usuario (pensado para que el usuario pueda ver una página más completa de cada músico, pero probablemente no me de tiempo)
3. Endpoint, controlador y servicios para "musicians/update" -> Nos permite actualizar diferentes datos del usuario. He configurado username, contraseña e email.
4. Pruebas para actualizar imagen -> He intentado implementar la funcionalidad de cambiar la imagen del usuario pero, al estar gastando mucho tiempo en ello, he preferido dejarlo a un lado.
5. Pruebas y configuración inicial de jsDoc para que, llegado el momento, simplemente tenga que poner los comentarios.

## ## 3er SPRINT (BACKEND)

### ### RESUMEN DE OBJETIVOS:

He creado los endpoints correspondientes a la búsqueda de músicos. Por una parte tenemos la búsqueda inicial (todos los músicos con paginación) y la búsqueda filtrada. Esto supone varios controladores y acciones en los servicios (búsquedas en las bdd).

1. Creación del router generic.routes.js
  - \* Alberga las rutas para obtener listado de estilos, instrumentos...
  - \* Se asocia con generic.controller.js
  - \* Se asocia con generic.services.js
  - \* En resumen, todas las acciones para obtener información estática de la BD.
2. Endpoint y controlador '/musicians'
  - \* Permite obtener un listado completo de músicos.
  - \* Queda pendiente aplicar paginación. Si da tiempo lo haré.
3. Endpoint y controlador '/musician/search'
  - \* Permite, a través de query params, filtrar una búsqueda
  - \* Igualmente queda pendiente paginación si da tiempo
4. Servicios correspondientes (acciones en la bd)
5. Queda pendiente swagger y limpiar código para futuros sprints.

## ## 2o SPRINT (BACKEND)

### ### RESUMEN DE OBJETIVOS:

He trabajado toda la parte referente al signin, así como la recuperación de la contraseña. He creado el sistema de tokens que tendrá la app (con un accessToken y un refreshToken). He reorganizado los endpoints para que tanto signup como signin se accedan con "/auth" y tengan su router y controlador como "auth".

1. Endpoint, controlador y middlewares para "/auth/signin".  
(Acceder a cuenta)
2. Endpoint, controlador y middlewares para "/auth/verify-access-token".  
(Verificar el accessToken al iniciar la app o solicitar rutas protegidas)
3. Endpoint, controlador y middlewares para "/new-access-token".  
(Generar un nuevo accessToken a través del refreshToken)
4. Endpoint y controlador para "/clear-cookies".  
(Cerrar sesión)
5. Endpoint, controlador y middlewares para "/password-recover-email".  
(Enviar email para recuperar y cambiar contraseña)
6. Endpoint y controlador para "/confirm-password-recover".  
(Generar recoverPassToken al pulsar el enlace del email)
7. Endpoint, controlador y middlewares para "/password-recover".  
(Cambiar la contraseña)
8. Actualización de swagger con los nuevos endpoints.
9. Pruebas de seguridad y manejo de errores

## ## 1er SPRINT (BACKEND)

### ### RESUMEN DE OBJETIVOS:

He trabajado intensamente en todo el diseño inicial del backend, realizando solo una funcionalidad (signup). Aunque me ha llevado tiempo, todo lo dispuesto en este sprint agilizará mucho el trabajo en los siguientes.

1. Diseño del backend (directorios, middlewares, conexiones...).
2. Diseño de la estructura de datos (sequelize y mongoose).
3. Funcionalidad de signup sincronizada con frontend.
4. Configuración de nodemailer.
5. Diseño de la clase Email y sus métodos.
6. Diseño de la clase resError, sus métodos y el objeto resErrors.
7. Configuración de winston.
8. Diseño de la clase logError, sus métodos y el objeto logErrors.
9. Configuración de swagger.
10. Configuración de variables de entorno (dotenv).
11. Configuración de jwt y bcrypt.