Facultad de Ingeniería

Bernard Wand Polak

Obligatorio 1

Diseño de aplicaciones 2

Lucas Castro - Nº 218709

Ricardo Poladura - Nº 238052

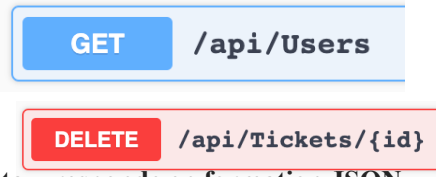
Grupo: N5A

Docentes: Ignacio Valle – Nicolás Fierro – Nicolás Blanco

EVIDENCIA DEL DISEÑO Y ESPECIFICACIÓN DE LA API

Criterios seguidos para asegurar que la API cumple con los criterios REST:

1. **Se usan sustantivos ante verbos:**



2. **Acepta y responde en formation JSON**

Request:

```
{
  "performerType": "string",
  "userId": 0,
  "startYear": "string",
  "genre": 0,
  "membersIds": [
    0
  ]
}
```

Respuesta:

```
{
    "message": "User successfully registered",
    "resultCode": "SUCCESS"
}
```

3. **Manejo de errores acorde a codigos de respuesta Http y errores descriptivos.**

Ejemplos de la API:

- **403**: Forbidden para aquellos recursos a los que el usuario no tiene permiso

```
{
    "statusCode": 403,
    "statusDescription": "Forbidden",
    "message": "Email or password incorrect."
}
```

- **400:** Bad Request para errores del cliente

```
{
    "statusCode": 400,
    "statusDescription": "BadRequest",
    "message": "The Password field is required."
}
```

- **401:** Unauthorized para cuando el no provee se identifica contra el sistema y por ende no esta autorizado.

- **200:** Ok Para aquellas requests que se llevaron a cabo correctamente. Por ejemplo : **GET** /genres y **POST** /users

4. **<u>Enpoints para acciones</u>**

Ejemplo de endpoint para accion de comprar ticket para un evento determinado.

Descripción del mecanismo de autenticación de requests:

Para autenticar a los usuarios del sistema y proteger recursos de ser accedido en bases a autorización de roles se decidio implementar la autenticación de requests mediante tokens de acceso particularmente utilizando la librería de JWT authentication (Json Web Tokens).

JWT especifica un metodo compacto y autocontenido para comunicar información como objetos JSON entre dos partes. Esta firmado y en base a esto puede ser verificado y confiable. Para esto utilizamos lo que se llama “secret” para generar un Hash (que es generado mediante el algoritmo HMAC)

Ejemplo en la API:

Basicamente se verifica la identificación del usuario mediante la adquisición de sus credenciales y usandolas para confirmar la identidad de dicho usuario. El proceso de autorización comienza una vez que las credenciales son legitimas. De ahí se procede a chequear permisos en base a roles.

Para autenticar se utiliza el header Authorization de la request para extraer el Bearer Token que utiliza el usuario una vez que quiere acceder a un recurso protegido. Pero en caso de querer loguearse contra el sistema, ahí se le genera un token de acceso.

A modo de ejemplo mostramos la implementación de un filter que es ejecutado como parte del pipeline de una request:

```
public async Task OnAuthorizationAsync(AuthorizationFilterContext context)
{

    var token = context.HttpContext.Request.Headers["Authorization"]
        .FirstOrDefault()?.Split(" ").Last();
    this.userService = context.HttpContext.RequestServices.GetService(typeof(IUserService)) as IUserService;
    var user = await userService.RetrieveUserFromToken(token);

    if (user == null)
    {
        var unauthorizedError = new UnauthorizedError("Access denied.");
        context.Result = new ObjectResult(unauthorizedError)
        {
            StatusCode = unauthorizedError.StatusCode
        };
    }
    else
    {
        if (!args.Contains(user.Role))
        {
            var forbidden = new ForbiddenError("No required authorization to access resource.");
            context.Result = new ObjectResult(forbidden)
            {
                StatusCode = forbidden.StatusCode
            };
        }
        else
        {
            var json = JsonConvert.SerializeObject(user);
            context.HttpContext.Session.SetString("user", json);
        }
    }
}
```

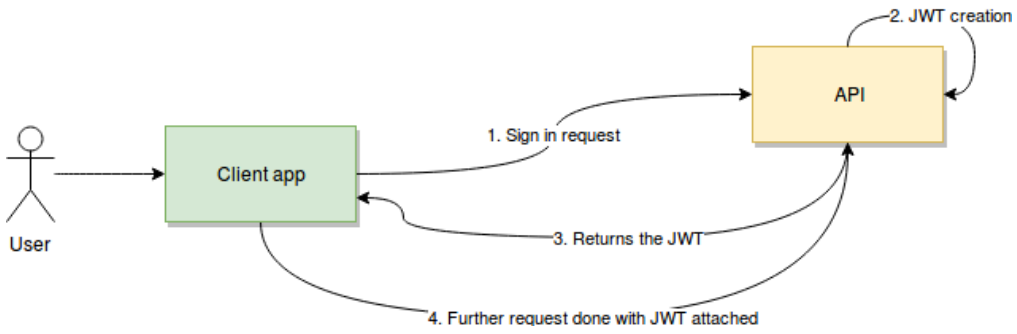
Aquí cuando el ususario se loguea y se le genera un token de acceso en caso de ser autenticado correctamente:

```
public async Task<User> Login(AuthenticationRequest model)
{
    var found = await repository.Get(u => u.Email.Equals(model.Email));
    if (found == null)
    {
        return null;
    }
    if (found.Password != null && !BC.Verify(model.Password, found.Password))
    {
        return null;
    }
    if (found.ActiveAccount)
    {
        var jwtService = this.factory.GetService(typeof(IJwtService)) as IJwtService;

        var token = jwtService.GenerateJwtToken(appSettings.JwtSecret, "id", found.Id.ToString());
        var user = mapper.Map<User>(found);
        user.Token = token;

        return user;
    }
    return null;
}
```

Aquí un breve diagrama de lo explicado anteriormente:



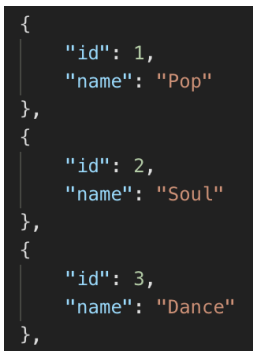
Descripción general de códigos de error (1xx, 2xx, 4xx, 3xx, 5xx)

- **403**: Forbidden para aquellos recursos a los que el usuario no tiene permiso

- **400:** Bad Request para errores del cliente

- **401:** Unauthorized para cuando el no provee se identifica contra el sistema y por ende no esta autorizado.

- **200:** Ok Para aquellas requests que se llevaron a cabo correctamente. Por ejemplo : **GET** /genres y **POST** /users



Descripción de los resources de la API

-> URL base: http://localhost:5000/api

Conciertos:

Endpoint + Metodo	POST /api/Events
Descripción	Registrar concierto
Parametros	-
Respuestas	200 - 401 - 403
Header	Authorization
Body	

Endpoint + Metodo	GET /api/Events
Descripción	obtener conciertos
Parametros	- type: string - newest: boolean - startDate: string - endDate: string - artistName: string - tourName: string - performerId
Respuestas	200
Header	-
Body	-

Endpoint + Metodo	GET /api/Events/{id}
Descripción	obtener concierto por id
Parametros	-
Respuestas	200
Header	-
Body	-

Endpoint + Metodo	PUT /api/Events/{id}
Descripción	Actualizar concierto
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization

Body	{
	"date": "2022-06-16T23:06:12.525Z",
	"ticketPrice": 0,
	"currencyType": "string",
	"eventType": "string",
	"tourName": "string",
	"location": "string",
	"address": "string",
	"country": "string",
	"artistsIds": [
	0
	]
	}

Endpoint + Metodo	DELETE /api/Events/{id}
Descripción	Borrar concierto
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization
Body	

Endpoint + Metodo	GET /api/Events/performer
Descripción	obtener conciertos
Parametros	- type: string - newest: boolean - startDate: string - endDate: string - artistName: string - tourName: string - performerId
Respuestas	200
Header	-
Body	-

Generos

Endpoint + Metodo	POST /api/Genres
Descripción	Registrar genero
Parametros	-
Respuestas	200 - 401 - 403
Header	Authorization

Body	{
	"name": "string"
	}

Endpoint + Metodo	GET /api/Genres
Descripción	obtener generos
Parametros	-
Respuestas	200
Header	-
Body	-

Endpoint + Metodo	DELETE /api/Genres/{id}
Descripción	Borrar concierto
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization
Body	

Endpoint + Metodo	GET /api/Genres/{id}
Descripción	obtener genero por id
Parametros	-
Respuestas	200
Header	-
Body	-

Endpoint + Metodo	PUT /api/Genres/{id}
Descripción	Actualizar genero
Parametros	-
Respuestas	200 – 403 - 401
Header	Authorization

Body	{
	"name": "string"
	}

Performers

Endpoint + Metodo	POST /api/Performers
Descripción	Registrar performer
Parametros	-
Respuestas	200 - 401 - 403
Header	Authorization

Body	{
	"performerType": "string",
	"userId": 0,
	"startYear": "string",
	"genre": 0,
	"membersIds": [
	0
	]
	}

Endpoint + Metodo	GET /api/Performers
Descripción	obtener performers
Parametros	-
Respuestas	200
Header	-
Body	-

Endpoint + Metodo	PUT /api/Performers/{id}
Descripción	Actualizar performer
Parametros	-
Respuestas	200 – 403 - 401
Header	Authorization

Body	{
	"performerType": "string",
	"startYear": "string",
	"genreId": 0,
	"artistsIds": [
	0
	]
	}

Endpoint + Metodo	DELETE /api/Performers/{id}
Descripción	Borrar performer
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization
Body	

Endpoint + Metodo	GET /api/Performers/{id}
Descripción	obtener performer por id
Parametros	-
Respuestas	200 – 401 - 403
Header	-
Body	-

Tickets

Endpoint + Metodo	POST /api/Tickets/purchase/{eventId}
Descripción	Comprar ticket
Parametros	- eventId: string
Respuestas	200 - 401 - 403
Header	Authorization (si es usuario registrado solo mandar Bearer Token)
Body	{ "firstName": "string", "lastName": "string", "email": "string" }

Endpoint + Metodo	PUT /api/Performers/{id}
Descripción	Actualizar Ticket status
Parametros	-
Respuestas	200 – 403 - 401
Header	Authorization
Body	{ "code": "string", "status": "string" }

Endpoint + Metodo	DELETE /api/Tickets/{id}
Descripción	Borrar ticket
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization
Body	

Endpoint + Metodo	GET /api/Tickets/{id}
Descripción	obtener ticket por id
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization
Body	-

Endpoint + Metodo	GET /api/Tickets
Descripción	obtener tickets
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization
Body	-

Endpoint + Metodo	GET /api/Tickets/code/{code}
Descripción	obtener codigo de ticket
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization
Body	-

Users

Endpoint + Metodo	POST /api/Users/login
Descripción	Login usuario
Parametros	-
Respuestas	200 - 401 - 403
Header	Authorization
Body	

Endpoint + Metodo	GET /api/Users
Descripción	obtener usuarios
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization
Body	-

Endpoint + Metodo	POST /api/Users/login
Descripción	Registrar usuario
Parametros	-
Respuestas	200 - 401 - 403
Header	Authorization

Body	{
	"firstname": "string",
	"lastname": "string",
	"email": "user@example.com",
	"password": "string",
	"role": "string"
	}

Endpoint + Metodo	GET /api/Users/{id}
Descripción	obtener usuario por id
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization
Body	-

Endpoint + Metodo	PUT /api/Users/{id}
Descripción	Actualizar usuario
Parametros	-
Respuestas	200 – 403 - 401
Header	Authorization

Body	{
	"firstname": "string",
	"lastname": "string",
	"password": "string",
	"email": "string",
	"role": "string",
	"activeAccount": true
	}

Endpoint + Metodo	DELETE /api/Users/{id}
Descripción	Borrar usuario
Parametros	-
Respuestas	200 – 401 - 403
Header	Authorization
Body	

Se crearon 6 usuarios diferentes para poder realizar las pruebas con la API. Dichos usuarios son contienen diferentes roles para los cuales se puede probar las diferentes funcionalidades del sistema:

Id: 1 **Nombre**: Lucas **Apellido**: Castro **Email**: lucas@example.com **Contraseña**: lucas1 **Rol:** ADMIN	**Id**: 4 **Nombre**: Seller **Apellido**: Test **Email**: seller@example.com **Contraseña**: seller1 **Rol**: SELLER
Id: 2 **Nombre**: Ricardo **Apellido**: Poladura **Email**: ricardo@example.com **Contraseña**: ricardo1 **Rol:** ADMIN	**Id**: 5 **Nombre** = Supervisor **Apellido**: Test **Email**: supervisor@example.com **Contraseña**: supervisor1 **Role**: SUPERVISOR
Id: 3 **Nombre**: Spectator **Apellido**: Test **Email**: spectator@example.com **Contraseña**: spectator1 **Rol**: SPECTATOR	**Id**: 6 **Nombre**: Artist **Apellido**: Test **Email**: artist@example.com **Contraseña**: artist1 **Rol**: ARTISTA