



TÉCNICO SUPERIOR EN DESARROLLO DE
APLICACIONES WEB
Departamento de Informática



MANUAL TÉCNICO

Autor: Manuel Guillén Gallardo
Curso Académico: 2021/2022

ÍNDICE

Introducción	3
Arquitectura de la aplicación	4
2.2. Backend	4
2.3. Frontend	5
Documentación técnica	6
3.1. Análisis	6
3.1.1. Requisitos funcionales	6
3.1.2. Requisitos no funcionales	8
3.2. Desarrollo	8
3.2.1. Diseño de la BBDD	8
Modelo Entidad / Relación de la BBDD	9
Modelo relacional de la BBDD	10
3.2.2. Diagrama de casos de uso	11
3.3. Pruebas realizadas	12
Proceso de despliegue	16
4.1. Introducción	16
4.1. Vercel para Angular	16
4.2. Heroku para Spring Boot	20
Propuestas de mejora	21
5.1. Requisitos funcionales pendientes	21
5.2. Requisitos técnicos	21
5.2.1 Backend	21
5.2.2 Frontend	21
6. Webgrafía:	23
6.1. Cursos online	23
6.2. Webs de referencia	23
6.3. Artículos de referencia	23
6.3.1. Definición de API Rest	23
6.3.2.. Modelo Vista Controlador	24

1. Introducción

Toggle es una red social para centros educativos de Formación Profesional (FP en adelante) en Extremadura. Su finalidad es ofrecer un canal de comunicación entre los distintos ciclos formativos, teniendo como premisa fundamental la elaboración de proyectos colaborativos.

A continuación expondremos qué objetivos se pretenden alcanzar con esta aplicación y por qué:

- Definir una **arquitectura escalable** que permita enriquecer la funcionalidad de la aplicación para adaptarse a posibles nuevas necesidades del mercado y los usuarios. Es por este motivo que se ha trabajado con algunas de las tecnologías y diseños en boga actualmente.
- Gran parte de las aplicaciones relacionadas con el ámbito educativo y que basan su funcionalidad en el intercambio o generación de contenidos, están dirigidas al profesorado. Con nuestra propuesta, establecemos a **los centros educativos como raíz estructural** sin que por ello el profesorado pierda autonomía. Este hecho amplía las posibilidades funcionales de la aplicación (en principio limitada al ámbito FP y al territorio extremeño) y mejora la gestión y organización de contenidos.
- Ofrecer una **experiencia de usuario de calidad**. Para ello se implementan interfaces intuitivas, de corte minimalista y adaptables a todo tipo de dispositivos.

2. Arquitectura de la aplicación

2.1. Introducción

En cuanto a las herramientas empleadas para el desarrollo del proyecto se han empleado las siguientes:

- Un equipo con el sistema operativo de Windows 10.
- Los entornos de desarrollo:
 - Eclipse > v2022-03(4.23.0).
 - Visual Studio Code > v1.68.0.
- Aplicaciones de escritorio:
 - MySQL Server > v8.0.
 - Workbench > v8.0.
 - pgAdmin > v4 6.5
 - Postman > v7.0.9.
 - Github Desktop > v3.0.2.
- Aplicaciones en la nube:
 - lucid.app
 - app.diagrams.net
 - [Spring initializr](https://spring.io/init)
 - [Json Web Token](https://jwt.io)¹
 - [Heroku](https://heroku.com)
 - [Vercel](https://vercel.com)
- Aplicaciones en consola de comandos:
 - Heroku CLI > v7.53.0
 - Angular CLI >
 - NodeJs > v16.14.2
 - npm > v8.5.0

En los siguientes apartados veremos cuáles han sido las tecnologías y patrones de diseño empleados para el desarrollo del proyecto.

2.2. Backend

Se han tomado como referencia diversos conceptos a la hora de implementar su arquitectura. Partiendo del precepto de separar en capas la gestión de los procesos que vinculan a la parte cliente y servidor, se ha tenido en cuenta:

¹ Ofrece un estándar de creación de tokens para el intercambio de información entre el navegador y el servidor. Entre sus ventajas: permite el transporte de metadatos y no está ligado a estados.

- Para la estructuración del código se ha optado por implementar el patrón de diseño Modelo Vista Controlador.
- Para la comunicación entre servidor y cliente de nuestra API se han tomado los límites de arquitectura definidos por el estilo REST (Representational State Transfer).
- Acorde a la división en capas de la parte backend, a fin de desacoplar el modelo de nuestra vista, se ha hecho uso del patrón DTO (Data Transfer Object).

Para la implementación de estos conceptos, se ha utilizado el lenguaje de programación *Java* y el framework *Spring* con su extensión *Spring Boot* en su versión 2.6.6. La utilidad del mismo excede lo dicho hasta el momento. Para mostrar esto, paso a exponer las dependencias implementadas en este proyecto:

- Spring Boot DevTools: permite reiniciar automáticamente la aplicación con cada cambio que se produzca en el código.
- Spring Web: para la creación de aplicaciones RESTful utilizando Spring MVC..
- Spring Security: controlador de accesos y autenticación.
- Spring Data JPA: facilita el trabajo con las capas de acceso a datos y su persistencia.
- MySQL Driver: habilita la conexión con la base de datos..
- Validation: permite establecer definiciones en nuestras clases y sus atributos.
- Lombok v1.18.22: generador de métodos recurrentes en Java.

2.3. Frontend

Para el desarrollo de la parte cliente, se ha trabajado con los siguientes lenguajes web:

- JavaScript.
- TypeScript.
- HTML.
- CSS.

Por otro lado, se ha implementado el framework Angular en su versión 13, acompañado de las siguientes librerías y módulos:

- SweetAlert2, JQuery, Toastr y Material.
- LOCALE_ID: permite configurar localización para el manejo de horas.
- Ng2SearchPipeModule: permite realizar búsquedas por filtro.
- DataTablesModule: permite elaborar tablas dinámicas.
- ReactiveFormsModule: permite elaborar formularios reactivos a la interacción del usuario.

La implementación de este framework está respaldada por la arquitectura que se ha usado en la parte servidora. De este modo, se ha tratado de llevar la estructura de carpetas y la definición de los elementos a la parte frontal.

3. Documentación técnica

3.1. Análisis

A continuación exponemos los requisitos funcionales y no funcionales implementados adecuadamente en la aplicación². Para organizar el contenido, tomamos como referencia los roles existentes en la aplicación.

3.1.1. Requisitos funcionales

USUARIO SIN CUENTA

- Acceder a la página principal de la aplicación.
- Acceder al formulario de creación de perfil.
- Crear perfil como usuario con rol *ROLE_MANAGER_CENTER*.
- Iniciar sesión.

ROLE_MANAGER_CENTER

- Acceder a su perfil. Compuesto por:
 - ◆ Datos del centro: Nombre del centro, Ciudad, Descripción.
 - ◆ Profesores asociados al centro.
 - ◆ Ciclos Formativos que se imparten en el centro.
 - ◆ Datos del administrador del centro: Nombre, Nombre de Usuario, Email y Contraseña.
- Opciones de creación:
 - ◆ Creación de profesor asociado a centro: Nombre y Apellidos, Nombre de Usuario, Contraseña y Email.
 - ◆ Asociar Ciclo Formativo a centro.
- Opciones de listado:
 - ◆ Visualización de profesores asociados.

² A la hora de exponer las posibles mejoras, se tendrán en cuenta, entre otros factores, aquellos requisitos funcionales expuesto en la propuesta de proyecto y que no han sido implementados.

- ◆ Visualización de Ciclos Formativos que imparte.
- Opciones de borrado:
 - ◆ Eliminación de profesores asociados.
 - ◆ Eliminación de Ciclos Formativos que imparte.
- Opciones de actualización:
 - ◆ Actualización de datos del centro.
 - ◆ Actualización de datos del administrador del centro: Nombre, Email y Password.
- Cerrar sesión.

ROLE_TEACHER_CENTER

- Iniciar sesión.
- Acceder a su perfil. Compuesto por:
 - ◆ Datos personales: Nombre, Email, Centro asociado y Contraseña.
 - ◆ Ciclos asociados.
 - ◆ Proyectos: Proyectos como creador y Proyectos como colaborador.
- Acceder a búsqueda de proyectos creados por otros profesores.
- Opciones de creación:
 - ◆ Asociar ciclo a su perfil.
 - ◆ Crear proyecto: Título y Descripción.
 - ◆ Crear solicitud de colaboración en un proyecto.
- Opciones de listado:
 - ◆ Visualizar Ciclos Formativos asociados a su perfil.
 - ◆ Visualizar listado de proyectos creados por él.
 - ◆ Visualizar el detalle de un proyecto: Datos de proyecto, Colaboradores y Solicitudes de colaboración.
 - ◆ Visualizar listado de proyectos en los que colabora.
- Opciones de borrado:
 - ◆ Borrar proyecto creado por él.
 - ◆ Borrar ciclo asociado a él.
- Opciones de actualización:
 - ◆ Actualizar Datos Personales: Nombre, Email y Password.

- ◆ Actualizar Datos de Proyecto: Título, Descripción, Ciclos Asociados y Aceptar o Rechazar solicitudes de colaboración de otros profesores en proyectos creados por él..

→ Cerrar sesión.

3.1.2. Requisitos no funcionales

- Diseño responsivo.
- Experiencia de usuario caracterizada por ser intuitiva y amigable.
- Garantías de seguridad mediante el uso de tecnologías que certifican la autenticación de los usuarios.
- Diseño modular que permita aumentar el ciclo de vida y desarrollo de la aplicación de la forma más eficiente.

3.2. Desarrollo

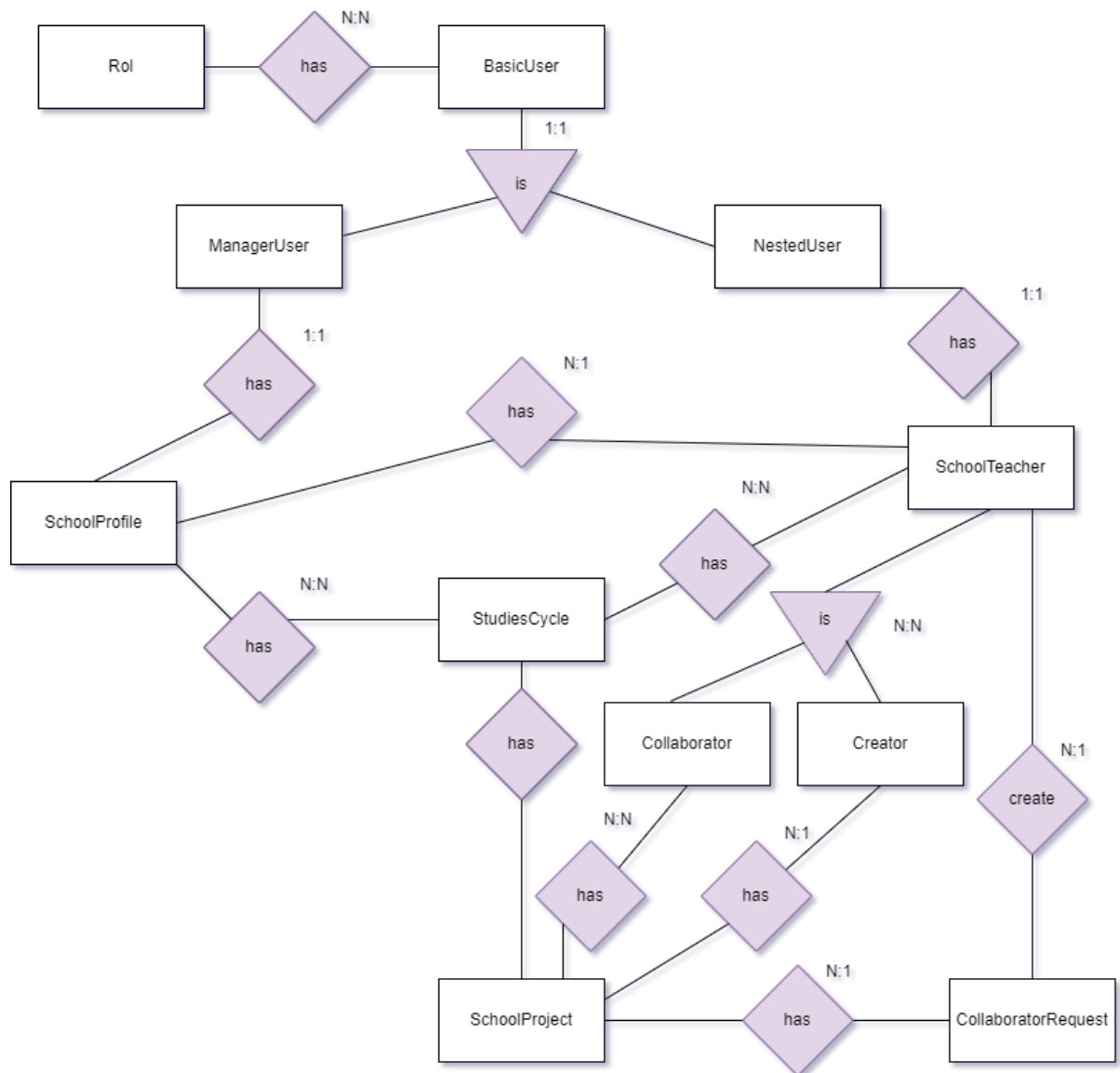
En este apartado comparto los diagramas utilizados para el desarrollo del proyecto. Algunos puntos deberán ser tratados en profundidad a fin de exponer claramente las dependencias existentes.

3.2.1. Diseño de la BBDD

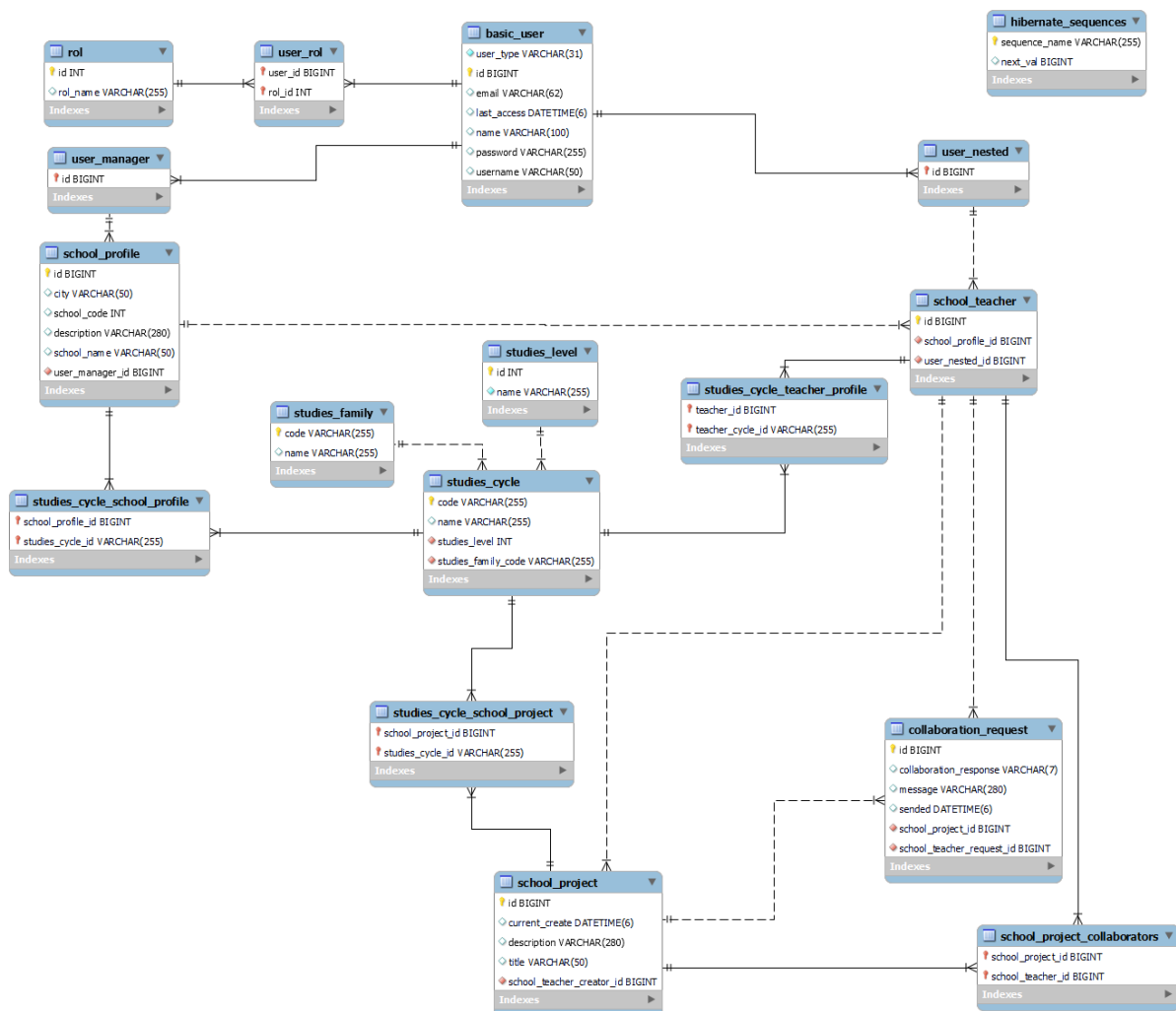
La creación de la base de datos se realiza mediante directivas JPA implementadas en nuestra aplicación. La descripción de dichas directivas serán ofrecidas mediante Javadoc.

Por otra parte, debido a la amplitud de las imágenes, serán compartidas en una carpeta adjunta a los documentos entregados junto al proyecto.

Modelo Entidad / Relación de la BBDD



Modelo relacional de la BBDD



En cuanto al diseño de la BBDD me gustaría responder a la siguiente pregunta:

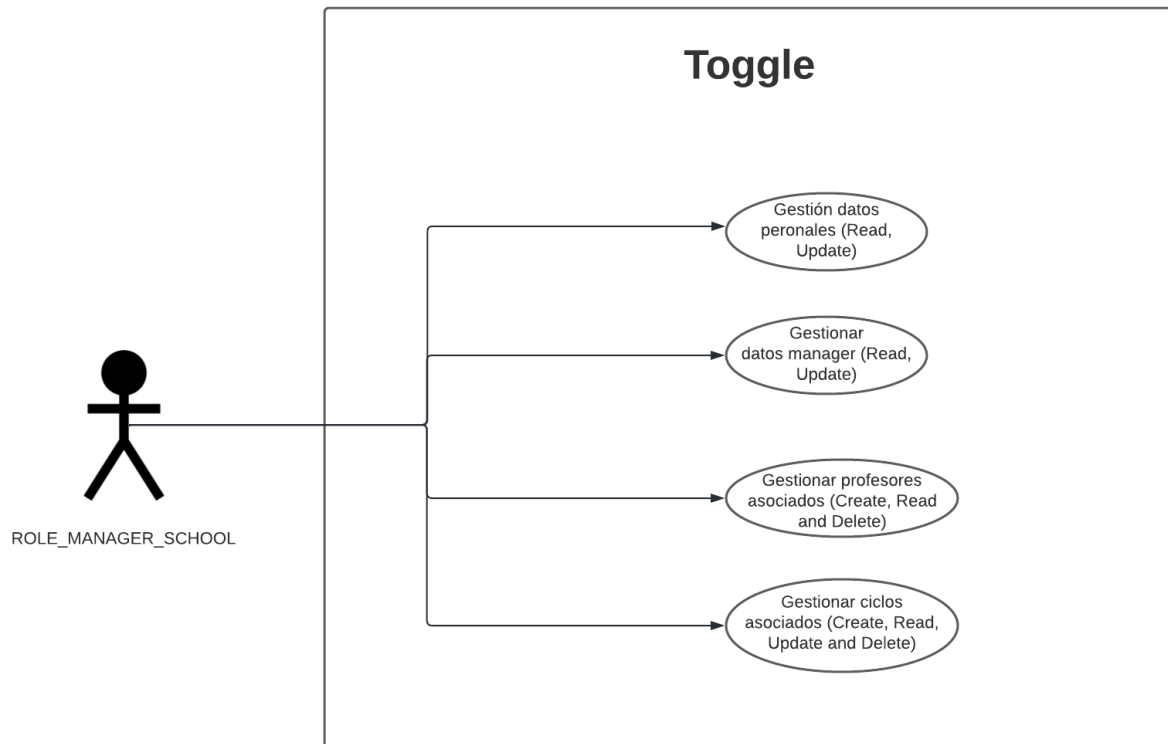
¿Por qué las dos relaciones generalizadas que pueden verse en el modelo E/R se resuelven de un modo distinto en el modelo relacional?

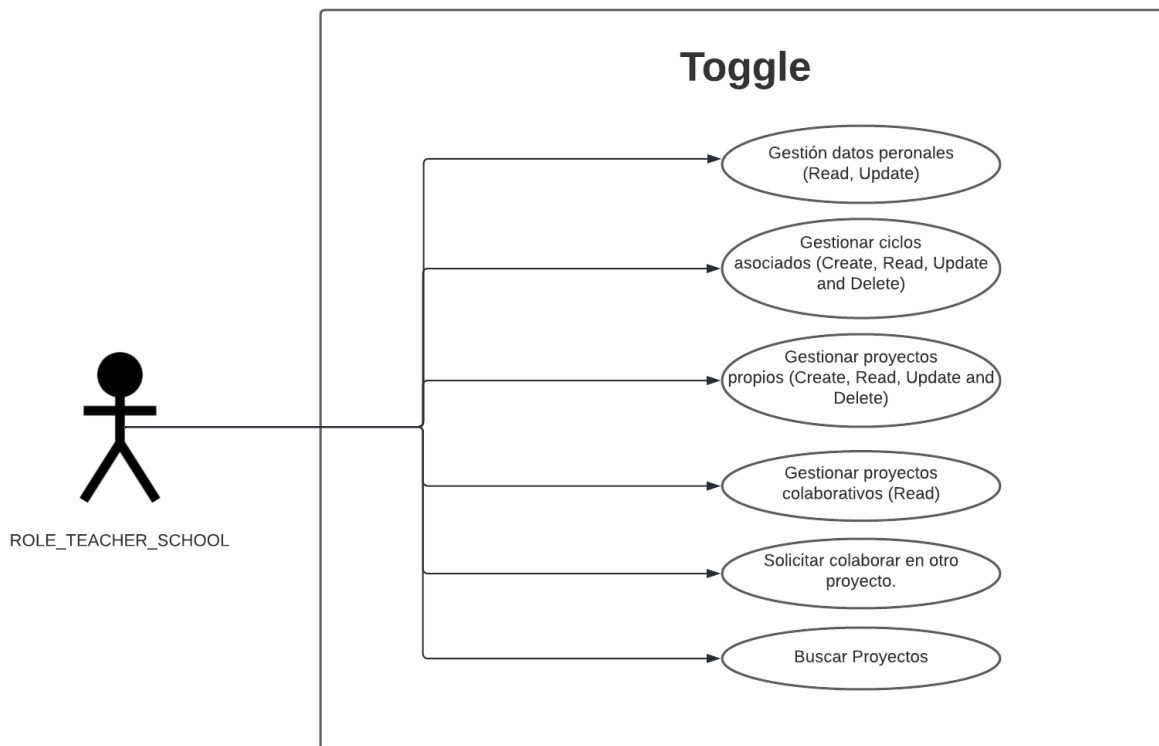
Resulta interesante apreciar que las relaciones generalizadas, BasicUser (UserNested || UserManager) y SchoolTeacher (Collaborator || Creator), se hayan resuelto en el modelo relacional de forma distinta. Esto se debe a la cardinalidad de las relaciones. Veamos:

- **BasicUser** se resuelve como una relación de herencia debido a la cardinalidad 1:1 en ambos resultados de la generalización.

- **SchoolTeacher** se resuelve o bien con tabla intermedia (Collaborator+SchoolProject), o bien como un atributo de una de las tablas, donde Creator es un atributo de SchoolProject.

3.2.2. Diagrama de casos de uso





3.3. Pruebas realizadas³

Para la realización de pruebas de la *API* servida por el backend se ha usado la aplicación Postman, la cual nos ofrece una serie de herramientas que nos facilitan la creación y validación de servicios, así como la generación de la documentación de la misma.

En el caso de nuestra aplicación, ha sido especialmente útil contar con Postman, debido a la implementación de *JWT*. El proceso de desarrollo podría resumirse en implementación de un servicio con los requisitos pertinentes y prueba del mismo.

Veamos el desarrollo de un ejemplo en el que se realizan dos peticiones sin autenticación y una en la que es un requisito estar autenticado. Todas estas peticiones están trabajándose con el mismo servicio, dado que realizaremos validaciones de autenticación únicas en nuestra aplicación. Para ello hemos necesitado de la interacción de la librería Spring Security con el manejo de tokens de autenticación.

³ Por falta de tiempo y problemas técnicos se dejan en el tintero las pruebas realizadas mediante **Heroku CLI** y el gestor de BBDD **pgAdmin**.

CREACIÓN DE CUENTA DE USUARIO (ROL_MANAGER_CENTER)

Realizamos una petición *POST* a nuestro servicio introduciendo los datos requeridos en formato *JSON*. Si la aplicación se encuentra desplegada y los datos cumplen con el formato y los requisitos definidos en la petición, recibiremos una respuesta *HTTP* informándonos de que se ha realizado con éxito.

The screenshot displays a REST client interface with the following details:

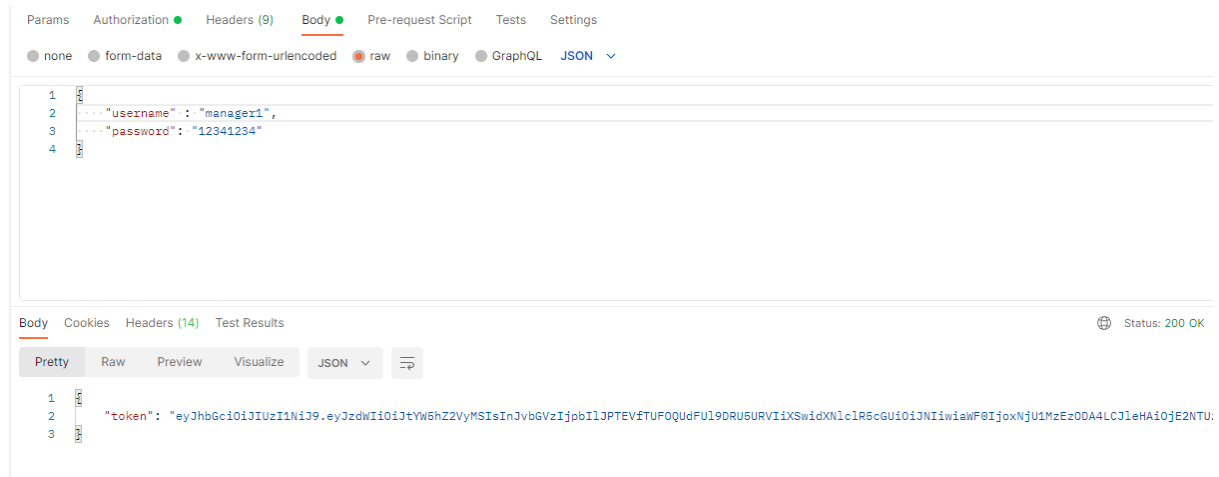
- Method:** POST
- URL:** http://localhost:8080/toggle/auth/create-account
- Body Tab:** Selected, showing a JSON payload:

```
1 {
2   "name": "Manager School",
3   "username": "manager1",
4   "email": "mschool@gmail.com",
5   "password": "12341234",
6   "schoolProfile": {
7     "name": "I.E.S. Santiago Apóstol",
8     "code": "11111113"
9   }
10 }
11 }
```
- Response Tab:** Selected, showing the response body in JSON format:

```
1 {
2   "mensaje": "Usuario guardado"
3 }
```

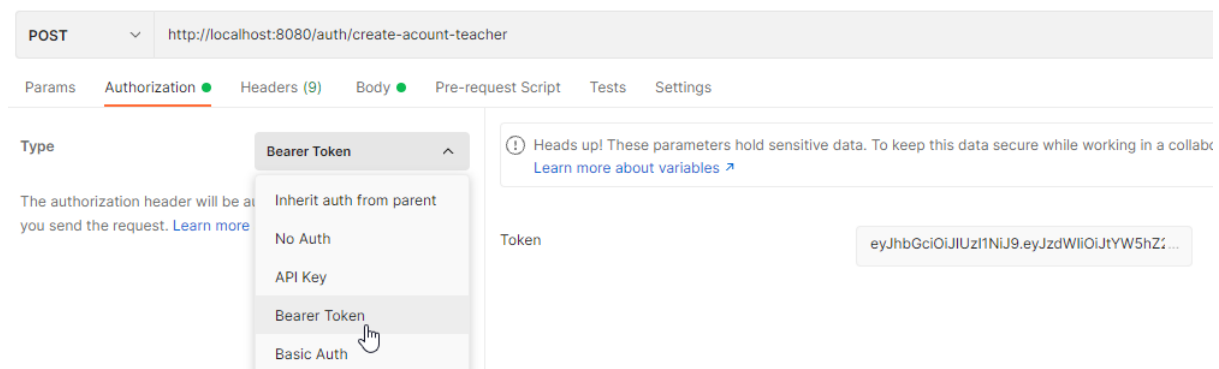
LOGIN CON USUARIO CREADO (ROL_MANAGER_CENTER)

La siguiente petición consume el servicio destinado a inicio de sesión, el cual implementa la lógica necesaria para la autenticación de los datos introducidos. En este caso debemos comprobar que nuestra petición recibe por respuesta un token que será la llave del usuario para navegar de forma segura y autenticada por nuestra aplicación.

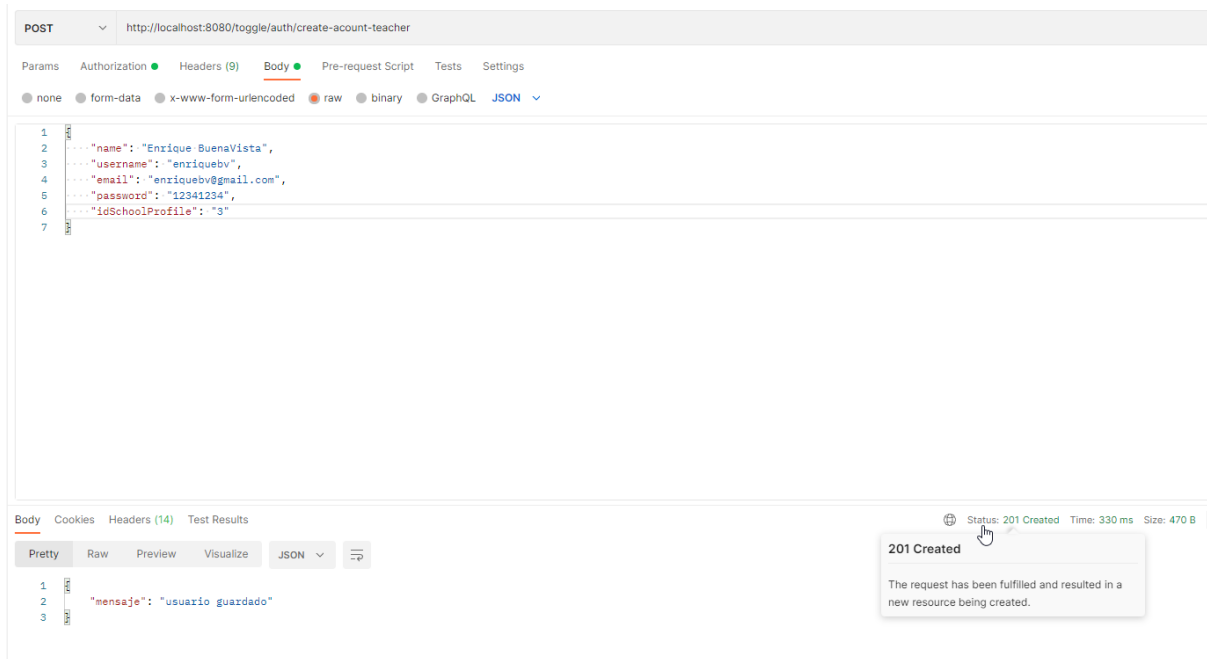


CREACIÓN DE USUARIO POR OTRO USUARIO (ROL_MANAGER_CENTER)

Como usuarios con `ROL_MANAGER_CENTER` podemos crear usuarios asociados al centro del cual somos administradores. Para realizar esta operación necesitamos contar con un token de autenticación que hemos recibido al loguearnos. Si bien en los anteriores casos ha bastado con introducir los datos adecuados en el cuerpo de nuestra petición, en este caso, necesitamos enviar también el token que valida nuestra autenticación.



Una vez hemos seleccionado el tipo de autenticación (Bearer Token) y hemos copiado el token que recibimos en la anterior petición. Es momento de introducir los datos en el cuerpo de la petición y realizar la solicitud.



Como se ha mencionado anteriormente, se ha realizado una documentación para exponer las especificaciones de los servicios que ofrece la aplicación. A dicha documentación se puede acceder desde el siguiente enlace: [Toggle REST API \(getpostman.com\)](https://getpostman.com).

4. Proceso de despliegue

4.1. Introducción

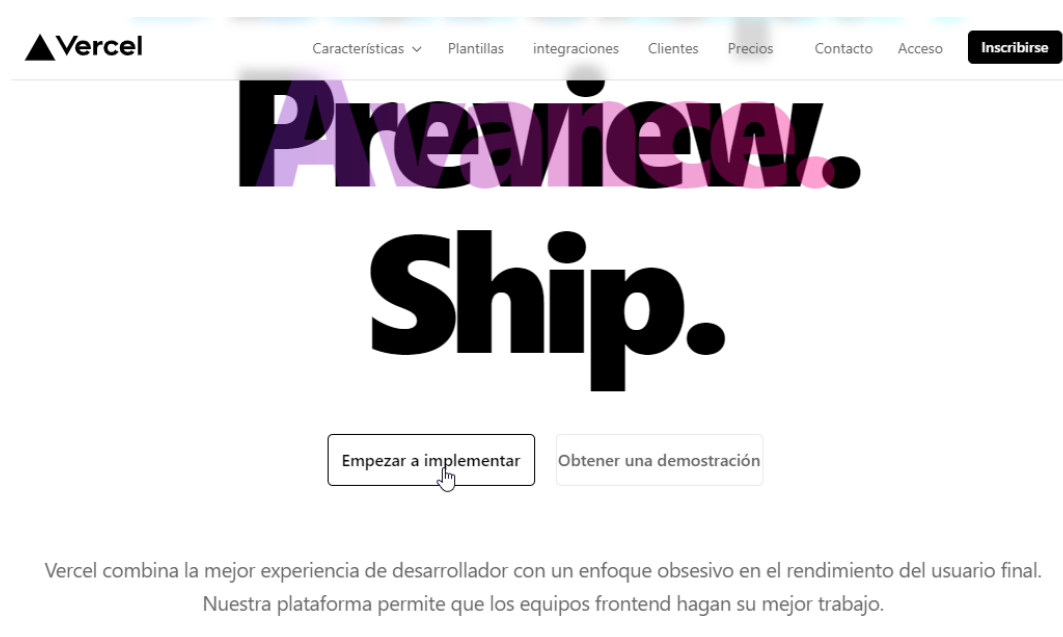
Cada una de nuestras aplicaciones será desplegada con una herramienta. Para nuestra aplicación en frontend desarrollada en Angular vamos a usar vercel.com y para nuestra aplicación en backend desarrollada en Java y SpringBoot vamos a utilizar heroku.com.

He de señalar que he intentado desplegar mis aplicaciones mediante distintas herramientas. La combinación señalada es la que más se ha ajustado a mis necesidades (o lo que es lo mismo, es la única que he conseguido hacer funcionar), aunque para realizar este despliegue he debido cambiar el gestor de BBDD pasando de MySQL a Postgre. Si bien esto último ha resultado ser un inconveniente de última hora, será interesante aportar en la descripción lo aprendido en el proceso.

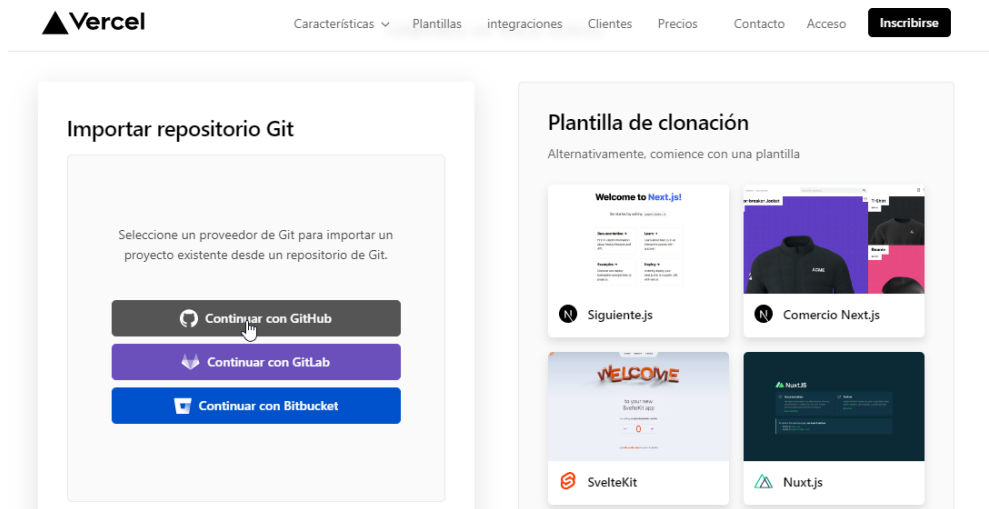
Por último, nos ahorraremos un paso advirtiéndolo que para ambos despliegues será necesario contar con una cuenta de GitHub y con los respectivos repositorios clonados.

4.1. Vercel para Angular

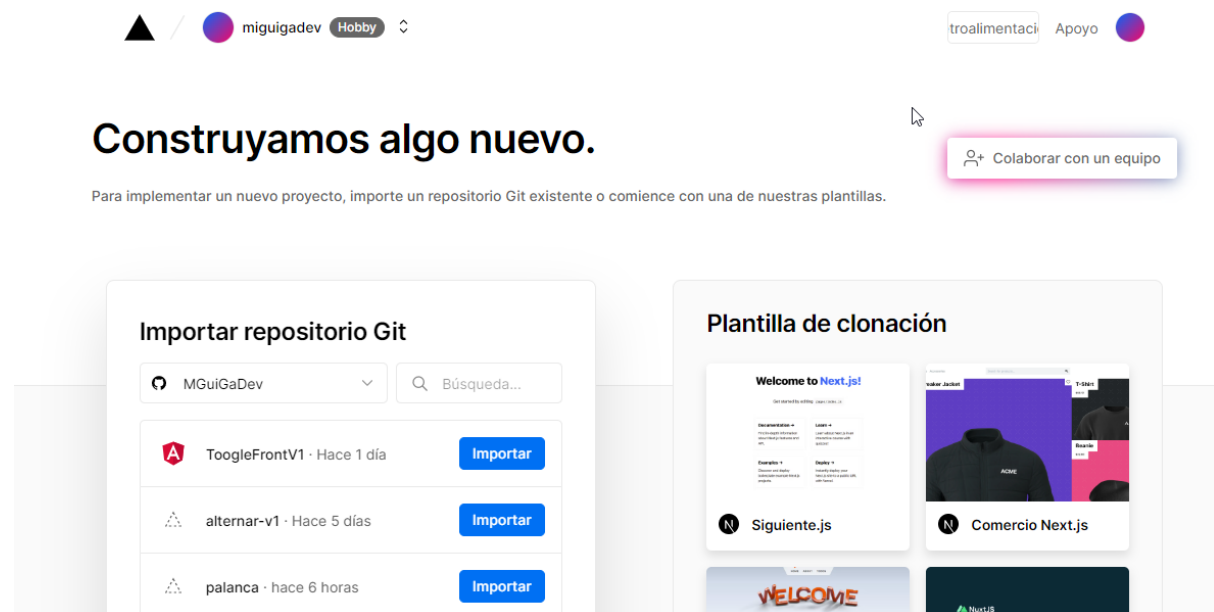
Accedemos a la página oficial ofrecida en el anterior apartado:



Al hacer clic en implementar, somos redirigidos al menú de opciones disponibles para desplegar nuestro repositorio. En esta vista, seleccionamos “Continuar con GitHub”.



Como sucede cada vez que tratemos de enlazar algún software a nuestra cuenta de GitHub, se nos solicita iniciar sesión. Iniciamos sesión y accedemos a nuestro “dashboard” de despliegue.





Se puede apreciar en la imagen que nuestra cuenta de GitHub se encuentra enlazada, dado que aparecen nuestros repositorios. Seleccione la primera opción y configuro el despliegue:

Configurar proyecto

NOMBRE DEL PROYECTO

MARCO PREESTABLECIDO

 Angular 

DIRECTORIO RAÍZ

Editar

► Configuración de compilación y salida

► Variables de entorno

Desplegar

Tras clicar en “Desplegar” comenzará la implementación de nuestra aplicación:

Desplegar

 Implementación en cola...hace 5s...

► Edificio	5s 
► Ejecución de comprobaciones	
► Asignación de dominios	

 pie de página - f6c214

Cancelar implementación

Una vez finalizado el despliegue podremos dirigirnos a nuestro panel de control:



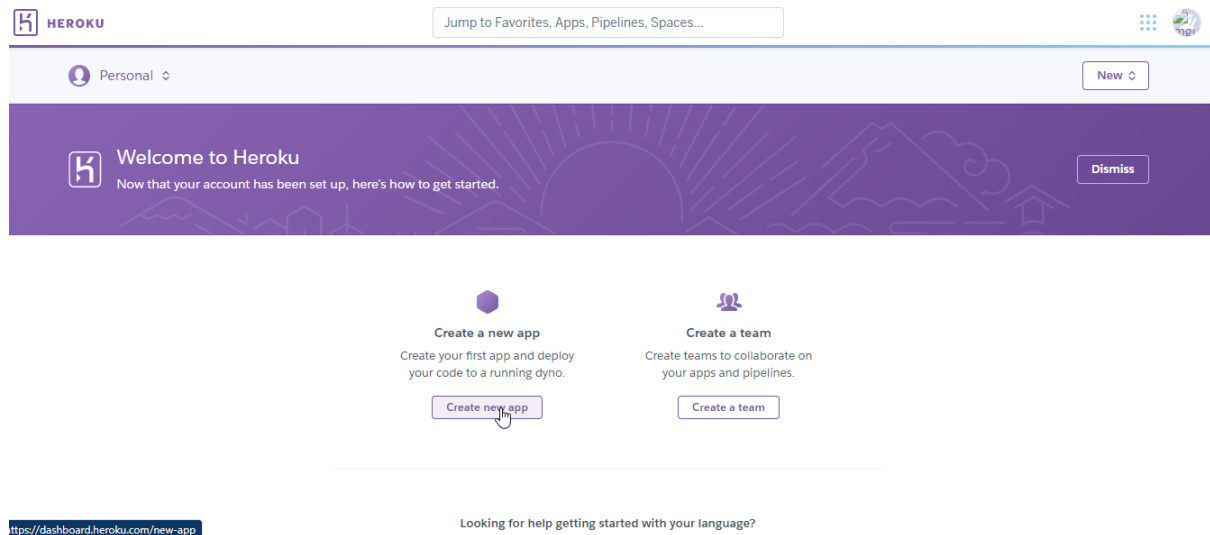
En nuestro panel de control tenemos la información de nuestro despliegue. Entre los datos, debemos prestar atención a la URL que usaremos para acceder a nuestro proyecto desplegado y al campo RAMA, en el cual se especifica que versión se está desplegando:



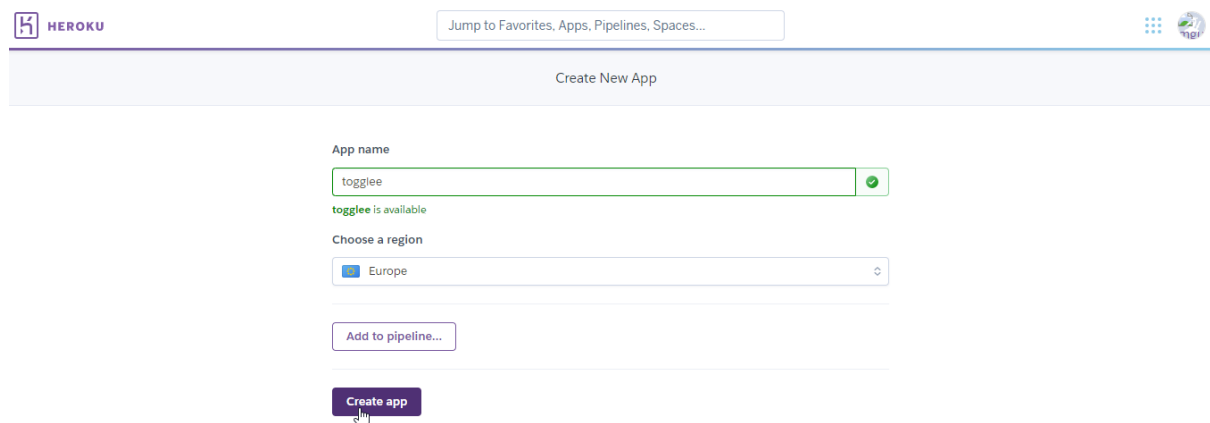
Este tipo de despliegue tiene dos grandes beneficios: es muy fácil de ejecutar y está integrado a GitHub, por lo que cada vez que actualicemos el repositorio, tenemos la posibilidad de actualizar el despliegue.

4.2. Heroku para Spring Boot⁴

En primer lugar, accederemos al link aportado en el apartado anterior. Crearemos un usuario rellenando el formulario al efecto y una vez cumplimentado, accederemos a nuestra vista de perfil:

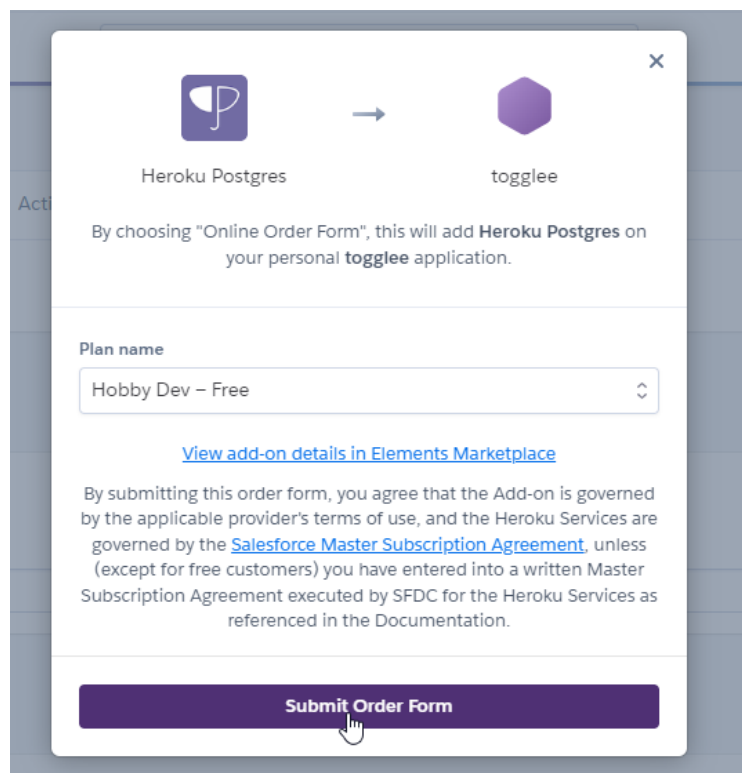
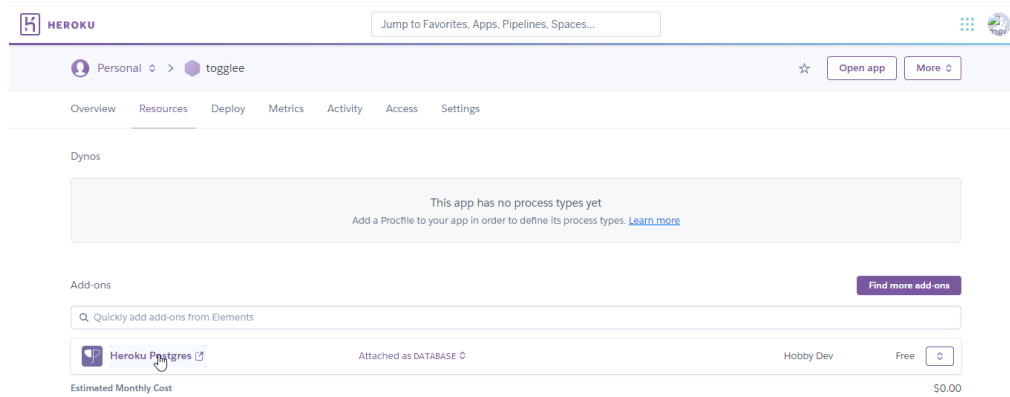


Haciendo clic en “Crear una app” comenzaremos este proceso de despliegue. Para ello, damos nombre a nuestra app y seleccionamos una región.

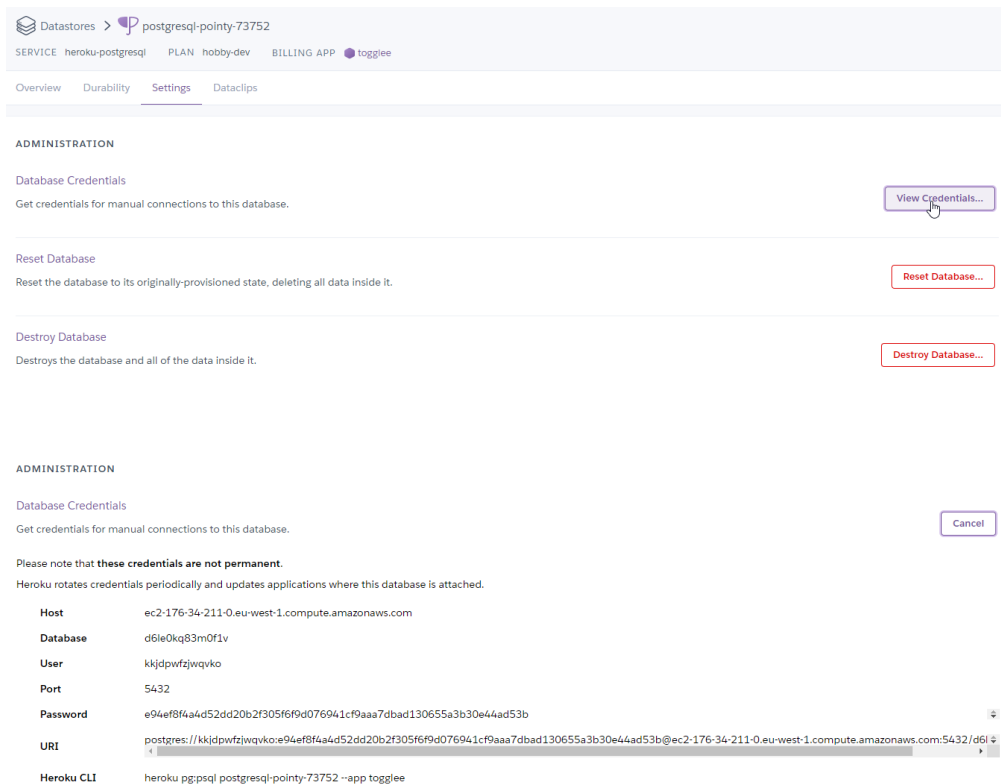


Una vez hemos creado la aplicación, se nos redirecciona a nuestro dashboard para configurarla. En primer lugar, seleccionamos en “Resources” a “Heroku Postgres”, debido a que MySQL requiere de uso de tarjeta de crédito o débito. Esto va a condicionar nuestros pasos en adelante.

⁴ Durante todo el día de hoy (16/06/2022) la plataforma Heroku.com ha presentado problemas con la API Dashboard de usuario. Esto ha impedido que pueda iniciar sesión desde consola. Es una lástima que no pueda añadir lo aprendido acerca de Heroku CLI, puesto que ha sido la clave para que pueda desplegar mi proyecto. Gracias a la visualización de los **log** he podido visualizar pormenorizadamente los problemas que han surgido en su despliegue y solventarlos.



Una vez hemos añadido nuestro gestor, descargamos la aplicación *pgAdmin* para poder hacer comprobaciones y gestionar la base de datos que se nos sirve desde la aplicación. Para ello, accedemos a la BBDD generada en nuestra aplicación y vemos las credenciales que son necesarias añadir tanto a nuestra aplicación de escritorio para gestionar Postgre, como en los distintos ficheros de nuestra aplicación backend.



Como se aprecia en la imagen anterior, se nos proporciona un nombre de usuario, un puerto un host para la conexión y una contraseña. Estos datos deberán ser incluidos en el fichero *application.properties* de nuestra aplicación. Recordemos que teníamos la configuración para conectar con MySQL y, por tanto, debemos hacer más cambios. A continuación se comparten capturas del archivo antes mencionado y del archivo *pom.xml*.

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
```

En el archivo *pom.xml* debemos eliminar la dependencia con MySQL y añadirla a Postgre, como se aprecia en la imagen. En el archivo *application.properties* debemos introducir los datos antes señalados:

```
spring.datasource.url=jdbc:postgresql://ec2-63-32-248-14.eu-west-1.compute.amazonaws.com/d1la1iip3h1jjv
spring.datasource.password=a3d523623aa4843606f43053cf49fb5d754a56cd7ad8756ed03643e785ecb968
spring.datasource.username=nywdlcjnmkvzt
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL10Dialect
spring.jpa.properties.hibernate.format_sql=true
jwt.secret=${JWT_SECRET}
jwt.expiration=${JWT_EXPIRATION}
```

Debemos prestar especial atención a las variables declaradas en este archivo. Estas son necesarias debido a mi implementación de JWT y ahora son responsabilidad de Heroku.

Además, debemos realizar otros cambios en el directorio raíz de nuestra aplicación. Añadimos un archivo *.gitignore* configurado para ignorar los siguientes archivos:

```
target/
.settings/
.mvn/
```

Por último, será necesario añadir un archivo *Procfile* para definir cómo queremos que se comporte nuestra aplicación por defecto al ser desplegada:

```
web: java -Dspring.profiles.active=default -Dserver.port=$PORT -jar target/*.war
```

Finalmente, elegimos la opción de despliegue conectando con nuestra cuenta de GitHub. Seleccionamos el repositorio que deseamos utilizar y desplegamos. Al terminar el despliegue, se nos comparte una URL que es donde está desplegada la aplicación.

5. Propuestas de mejora

En cuanto a las propuestas de mejora, he decidido dividir las mismas en apartados dependiendo del ámbito afectado. Me gustaría puntualizar que el desarrollo de esta aplicación ha ido paralelo al aprendizaje de las tecnologías, herramientas y conceptos utilizados.

5.1. Requisitos funcionales pendientes

En el tintero han quedado los siguientes requisitos:

- Creación de un usuario con rol de administrador.
 - ◆ Vistas relativas a las funciones que puede realizar.
 - ◆ Eliminar usuarios de tipo ROL_MANAGER_CENTER y ROL_TEACHER_CENTER.
- Ciertas funcionalidades para ROL_MANAGER_CENTER.
 - ◆ Vista de detalle de perfil.
 - ◆ Vista de búsqueda de otros perfiles.
 - ◆ Vista de búsqueda de proyectos de profesores no asociados al mismo.
 - ◆ Implementación de datos relacionados al centro: Intereses y Redes Sociales.

5.2. Requisitos técnicos

5.2.1 Backend

En cuanto a algunas de las mejoras aplicables a este ámbito, señalo:

- Reorganización de los servicios así como del resto de paquetes relacionados con los mismos: Repositories, Services y Controllers.
- Buenas prácticas para la implementación de dichos servicios:
 - ◆ No todas las peticiones cuentan con respuestas ofrecidas por el servidor acerca del estado de las mismas.
 - ◆ Si bien la mayoría de servicios hacen uso de DTOs para el manejo de datos con el cliente, la definición de los mismos es mejorable.
 - ◆ Si bien no siempre se hace uso de las herramientas de automatización ofrecidas por la librería SPRING DATA para el manejo de sentencias SQL, sería conveniente hacer un uso personalizado de las mismas a fin de interactuar con mayor eficiencia sobre la BBDD.

5.2.2 Frontend

En cuanto a esta parte, apunto algunas de las mejoras:

- Es evidente la necesidad de una mejora de los estilos aplicados a las vistas.
- Reorganización de la estructura de paquetes.
- Replanteamiento de los modelos/clases que protagonizan la transferencia de datos.
- Generación de componentes específicos de modo que permitan su modularización de un modo más eficiente y ordenado.
- Reajuste de los servicios utilizados.
- Implementación de sistema de notificaciones pendientes para usuarios.
- Implementación de librería para edición de texto que permita realizar descripciones más elaboradas en la creación de proyectos.

6. Webgrafía:

6.1. Cursos online

- Luigi Code (14 abril 2020). [CRUD básico Angular 8 + Spring-Boot + MySQL](#)
- Luigi Code (14 abril 2020). [Autenticación JWT con Spring Boot y Angular](#)
- Benedettelli, Rafael (29 octubre 2020). Curso de microservicios con Java y Spring Boot. [Escuela It.](#)
- Pérez Cano, Antonio (16 mayo 2020). Estructuración de un proyecto en Angular. [OpenWebinars.](#)

6.2. Webs de referencia

- [LogicBig.](#)
- [Java2s.](#)
- [Stackoverflow.](#)
- [RedHat.](#)

6.3. Artículos de referencia

6.3.1. Definición de API Rest

- [RedHat.](#)
- <https://juanda.gitbooks.io/webapps/content/api/arquitectura-api-rest.html>.

6.3.2.. Modelo Vista Controlador

- [DesarrolloWeb.](#)
- [FreeCodeCamp.](#)
- [SoftwareEngineering.](#)