Grado en Ingeniería Informática

Trabajo de Final de Grado

Aplicación de mapeo de atributos, DAPIgen

Autor:
Jesús Tur Blanco

Supervisor:
Rafael Goterris Perales
Tutor académico:
Carlos Granell Canut

Fecha de lectura: 11 de Julio de 2021
Curso académico 2020/2021

Resumen

En este documento se describe el desarrollo de la aplicación DAPIgen, una aplicación que permite el mapeo de atributos a partir de recursos generados desde una base de datos, con el fin de generar un proyecto java a partir de los recursos generados.

Para ello, se ha realizado un análisis, diseño y desarrollo siguiendo una metodología ágil scrum, es decir, dividiendo el proyecto en varias historias de usuario que se convertirían en épicas y tareas, organizadas en sprints para lograr implementar la funcionalidad deseada de la aplicación. Resultando en una aplicación web funcional, capaz de generar recursos y mapear sus atributos a partir de una conexión con la base de datos.

Esta aplicación ha sido desarrollada durante la estancia en prácticas en *Cloudappi*.

Palabras clave

Reactividad, Web, API, Ágil, Recursos.

Keywords

Reactivity, Web, API, Agile, Resources.

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

La empresa en la que se han desarrollado las prácticas se llama *Cloudappi*, una empresa centrada en los microservicios, desarrollo de APIs, desarrollo web, aplicaciones híbridas, IoT y transformación digital. Con sedes en Madrid, Castellón, Ciudad de México y Lima.

El proyecto se desarrolla en el área de I+D de la empresa, con la principal motivación de adaptar una aplicación anterior, Madrid Digital, a las actuales necesidades de la empresa, desarrollando una aplicación de mapeo de atributos con funcionalidad modificada y extendida, que permita al usuario un uso más rápido de esta. Pues en procesos complejos, habían procesos muy repetitivos y otros que no se podían deshacer que se pretenden simplificar en el desarrollo de esta aplicación.

Madrid digital era una aplicación de mapeo de atributos, con la que conectándose a una base de datos, se podían seleccionar ciertas entidades de la base de datos y verbos (Get, Post, Put, Delete), generando recursos con sus respectivas operaciones a partir de los cuales se podía generar un proyecto java dónde cada entidad se convertiría en una clase java y cada recurso en una clase controller relacionada a una entidad en la que aparecería cada operación como método con los parámetros necesarios para realizar llamadas a una API. Sin embargo presentaba algunos problemas de usabilidad, por ejemplo, al conectarse con la base de datos el puerto y el tipo de base de datos debía de escribirse a mano cuando normalmente solo se usan 3 tipos de bases de datos (POSTRGREESQL, ORACLE y MYSQL); también al crear recursos no aparecía marcada ningún verbo y se debía seleccionar uno a uno cuando es más cómodo que de base estén todos seleccionados y quitar los que no se necesiten; aunque el principal problema es que al crear un recurso, en caso de equivocarse al escoger las operaciones no se podían eliminar ni añadir de nuevos. Por eso, este proyecto se enfocó en, además de adaptar el proyecto al nuevo backend, solventar estos problemas de usabilidad.

1.2. Objetivos del proyecto

El objetivo de este proyecto es modificar y adaptar la aplicación Madrid Digital para que se acople a las nuevas necesidades de la empresa. Además se pretende extender la funcionalidad que implementaba, de forma que resulte más fácil la usabilidad por parte del usuario.

1.2.1. Alcance funcional

Desde el punto de vista funcional los principales objetivos son los descritos a continuación:

- Permitir que el usuario pueda conectarse a la base de datos, añadiendo el schema y puerto que necesite.
- Permitir que al crear un recurso (un controller relacionado a una entidad que contiene operaciones que permiten comunicar con una API rest), el usuario pueda escoger las tablas y operaciones de la base que necesite, precargando por defecto todas las operaciones.
- Permitir que el usuario, una vez cree un recurso, pueda eliminar las operaciones que no necesite del controlador asociado.
- Permitir que el usuario, una vez se cree un recurso, pueda añadir las operaciones que falten a un controlador.
- Permitir al usuario, una vez se creen los recursos, que pueda visualizar para cada operación de un controlador los parámetros que necesita si los necesita; los atributos que requiere cuando se haga una petición y los atributos que devolverá en la respuesta, pudiendo modificar estos atributos y mapearlos.

1.2.2. Alcance organizativo

Respecto al alcance organizativo, el sistema interactuará con las áreas de desarrollo frontend en java, quienes se encargarán de usar la aplicación para generar proyectos java a partir de los recursos que creen.

1.2.3. Alcance informático

Este sistema se comunicará con una base de datos Oracle a través de una API Rest, mantenida en un servidor de Amazon, de donde sacará los datos necesarios para construir para cada recurso, los *controllers*, sus respectivas operaciones y los parámetros que se necesitarán en cada operación.

1.3. Descripción del proyecto

Cuando un desarrollador java necesita crear una aplicación, normalmente debe crear a mano las clases necesarias del proyecto y nombrarlas una a una de forma manual con el entorno de desarrollo, ya sea Intellij, Eclipse o Visual Studio Code. Con el fin de agilizar este proceso manual, se desarrolló una aplicación de mapeo de atributos llamada Madrid Digital, con la que a partir de un fichero json o creando un recurso a partir de la base de datos, se crean diversos *controllers*, cada uno dotado de las operaciones básicas de una API Rest que se hayan seleccionado, en los cuales se pueden mapear y editar los atributos que contiene pudiendo cambiar su nombre, entidad a la que se relaciona, tipo y si debe tener alguna validación.

No obstante, esta aplicación presentaba algunos problemas en la usabilidad, que hacían su uso algo lento, por lo que a partir de la aplicación original, se decidió adaptar su funcionalidad y crear **DAPIgen**, una aplicación web con reactividad, desarrollada con el framework Vue 2, usando Typescript como lenguaje principal de programación y Jest para desarrollar los tests. La aplicación DAPIgen presentará varios cambios importantes respecto a la aplicación original, pues tanto la base de datos como la API Rest a la que se conecta son diferentes, por lo que la *store* donde se almacenan los datos de la aplicación, es decir, el modelo, deberá ser cambiado y adaptado para encajar con los nuevos datos que devolvían las llamadas de la API.

Además, este proyecto busca mejorar la usabilidad para el cliente como principal objetivo, de forma que se implementará funcionalidad adicional que permita al usuario utilizar de forma más sencilla la aplicación.

1.4. Estructura de la memoria

Sobre la estructura de esta memoria, cada capítulo describirá una fase del proyecto como se muestra a continuación:

En el capítulo 2, *Planificación del proyecto*, se describirá la metodología seguida en este proyecto, la planificación de este, los costes y recursos necesarios para desarrollarlo y que control del proyecto habrá.

En el capítulo 3, *Análisis y diseño del sistema*, se detallan los requisitos del proyecto y el modelado del sistema, así como los componentes del sistema y su interrelación junto con los criterios escogidos para el desarrollo de la interfaz gráfica.

En el capítulo 4, *Implementación y pruebas*, se explicará el trabajo de programación realizado, describiendo los patrones, estrategias y decisiones tomadas, además de detallar las pruebas que se realizaron para asegurar el correcto funcionamiento de la aplicación.

Finalmente el capítulo 5 se dedica a las conclusiones del proyecto.

Capítulo 2

Planificación del proyecto

En este capítulo se introducirá la planificación del proyecto, empezando por la metodología usada para la planificación y cómo se realizó, la estimación de los recursos y costes del proyecto y el seguimiento del proyecto.

2.1. Metodología

El proyecto ha sido desarrollado siguiendo una metodología ágil scrum adaptada a la empresa. Esta es una metodología ágil, basada en iteraciones donde se realizan uno o varios sprints, con reuniones diarias llamadas daily meetings y reuniones finales para cerrar cada sprint, donde se muestra al cliente las tareas desarrolladas e implementadas con el fin de poder mostrarle un producto con las funcionalidades trabajadas durante ese sprint.

Durante el periodo de estancias en prácticas, solo se realizó un único sprint, donde primeramente se realizó un pequeño backlog de tareas a completar. Con el tiempo, durante los daily meetings se fueron introduciendo nuevas tareas a medidas que se completaban las anteriores con el fin de implementar la funcionalidad necesaria en el periodo de tiempo establecido.

2.2. Planificación

La realización del proyecto se planificó siguiendo la metodología ágil scrum adaptada a la empresa, tal como se explica en el apartado anterior, teniendo en cuenta el límite de 300 horas, de las cuales 85 horas se destinaron al aprendizaje de las tecnologías que se iban a usar, Vuejs, typescript, apigen y jest, mediante cursos en udemy y tutoriales online.

Las 215 horas restantes fueron destinadas a completar las tareas del sprint y desarrollar la aplicación como tal. De forma que las horas quedaron repartidas de la siguiente forma:

- **Crear esqueleto de la app de mapeo**. Esta tarea consiste en crear la estructura de carpetas y ficheros de la aplicación. Esta tarea se planificó con una duración de 5 horas.
- **Crear maquetación de la página principal**. Esta tarea consiste en crear la estructura de la vista de la página principal, constando está de una cabecera, subcabecera, cuerpo y pie. Esta tarea se planificó con una duración de 5 horas.
- **Seleccionar las tablas y asociar los verbos**. Esta tarea consiste en crear un modal con un combo multiselección en el que se puedan escoger las tablas de la BBDD junto con las operaciones que van a tener. Esta tarea se planificó con una duración de 5 horas.
- **Añadir el tipo en la sección de conexión con la BBDD**. Esta tarea consiste en añadir un select en la sección de conexión con la BBDD de forma que se pueda seleccionar el tipo de la BBDD, MySQL, PostgreSQL u Oracle. Esta tarea se planificó con una duración de 5 horas.
- **Añadir el campo schema en la sección de conexión con la BBDD**. Esta tarea consiste en añadir un campo de texto a la sección de conexión con la BBDD, indicando el schema que se usará. Esta tarea se planificó con una duración de 2 horas.
- **Precargar el puerto al seleccionar un tipo de BBDD en la sección de conexión con la BBDD**. Esta tarea consiste en que una vez se seleccione el tipo de BBDD, se precargue un puerto por defecto dependiendo de la BBDD, también precargará un schema por defecto al seleccionar. Esta tarea se planificó con una duración de 3 horas.
- **Obtener de la conexión con la BBDD**. Esta tarea consiste, en que una vez se rellenen todos los campos de la sección de conexión con la BBDD, pulsando un botón, se pueda establecer la conexión con la BBDD. Esta tarea se planificó con una duración de 8 horas.
- **Realizar la llamada para crear un recurso**. Esta tarea consiste, en que una vez se seleccionen las tablas y verbos del modal descrito en la tarea *Seleccionar las tablas y asociar los verbos*, se llame a la API para que cree un recurso con las tablas y operaciones especificadas. Esta tarea se planificó con una duración de 3 horas.
- **Crear recursos**. Esta tarea consiste en que una vez hecha la llamada a crear un recurso, con los datos obtenidos de la API, se almacenen en la store y sean visualizados en la vista, mostrando los controllers con sus respectivas operaciones y dentro de estas, los atributos mapeados, pudiendo se estos editados por el usuario. Esta tarea se planificó con una duración de 86 horas.
- **Añadir operaciones a un recurso**. Esta tarea consiste en la creación de un modal que permita en un controller, añadir una operación básica que falte. Esta tarea se planificó con una duración de 33 horas.
- **Eliminar una operación de un recurso**. Esta tarea consiste en la creación de un botón al lado de cada operación, que al pulsarlo, permita eliminar la operación seleccionada, mostrando un modal para que el usuario pueda confirmar si desea borrarlo o no. Esta tarea se planificó con una duración de 15 horas.
- **Realizar tests para añadir operación**. Esta tarea consiste en crear tests para comprobar que la funcionalidad de la tarea *Recursos - añadir operación* funciona correctamente. Esta tarea se planificó con una duración de 40 horas.

- **Realizar cumplimentación del fichero README.md**. Esta tarea consiste en crear un fichero README.md que sirva como documentación del proyecto. Esta tarea se planificó con una duración de 3 horas.
- **Revisión y refactorización lógica**. Esta tarea consiste en refactorizar el código de forma que sea más legible. Se planificó con una duración de 2 horas.

2.3. Estimación de recursos y costes del proyecto

Respecto a los recursos necesarios para desarrollar el proyecto, se pueden estimar los siguientes:

- **Recursos humanos**. Este proyecto se planificó para ser realizado por un estudiante que realizará toda la parte de programación acompañado de un project manager que se encargará de crear las tareas y asegurar un seguimiento correcto del proyecto. El estudiante trabajará sin remuneración económica durante las 300 horas que dure su estadía en prácticas, mientras que el project manager cobrará unos 2000 euros al mes.
- **Herramientas de software**. Para la realización de este proyecto, se utilizará Visual Studio Code como entorno de desarrollo, TypeScript como lenguaje de programación y Vuejs 2 como framework para desarrollar el frontend. Todas estas herramientas son totalmente gratuitas y no suponen un coste adicional para el proyecto.
- **Herramientas de hardware**. Para desarrollar este proyecto, es necesario de un portátil con el que poder realizar la programación para desarrollar el proyecto, con un coste de 599 euros; un servidor activo donde se almacena la base de datos y se encuentra la API rest que comunicará el frontend y el backend, el cuál tiene un coste monetario de unos 1600 euros.

Suponiendo que el proyecto se desarrolla en un periodo aproximado de tres meses, el coste total del proyecto sería de unos 6000 euros para poder pagar los recursos humanos, más 599 del portátil, más 1600 del servidor, sumado a los gatos de luz, internet, mantenimiento y alquiler de la oficina, arroja un coste total de unos 10559 euros.

2.4. Riesgos del proyecto

Respecto a los principales riesgos del proyecto cabría destacar los siguientes:

- **Inexperiencia del desarrollador**. Como el desarrollador de la aplicación nunca antes trabajó con Typescript y Vue, era posible que surguieran problemas y atrasos. Para solventar este problema se optó por poner a disposición del desarrollador un profesional con experiencia que pudiera ayudarle cuando se quedara estancado.

- **Un solo desarrollador**. El desarrollo del frontend cayó en manos de un único desarrollador, por lo que era posible que no se completara toda la funcionalidad deseada, para ello se decidió por recortar parte de la funcionalidad para que el proyecto pudiera acabarse en la estancia en prácticas, centrándose en la conexión con la base de datos y generar los recursos.
- **Circunstancias imprevistas**. Al contar con un solo desarrollador, en caso de que cayera enfermo o sufriera un accidente, era posible que surgieran atrasos en el proyecto. Por eso, se decidió dejar dos días de margen al planificar la estancia en prácticas para poder tener un margen de tiempo con el que trabajar en caso de que surgiera un imprevisto.

2.5. Seguimiento del proyecto

Para el correcto seguimiento del proyecto, se realizaron reuniones diarias sobre las 10:15 de la mañana, llamadas daily meetings, entre el project manager y el estudiante. En ellas se describieron todos los progresos realizados el día anterior junto con el plan para el día. Finalmente, se realizó una reunión de cierre, donde se vio si se cumplieron los objetivos previstos.

En caso de que hubiere una desviación de la planificación, el proyecto podría haber sido retrasado, ya que al ser un proyecto interno de la empresa, perteneciente al I+D, no supondría demasiadas pérdidas o en el peor de los casos, mover a un profesional al proyecto para que ayude a su realización.

Capítulo 3

Análisis y diseño del sistema

En este capítulo se introducirá el análisis y el diseño del sistema, presentando el diagrama de casos de uso, las historias de usuario, el diagrama de clases del proyecto, el diseño de la arquitectura, el diseño navegacional y el diseño de la interfaz.

3.1. Análisis del sistema

3.1.1. Diagrama de casos de uso

En la *figura 3.1*, se puede apreciar el diagrama de casos de uso correspondiente a este proyecto, el cuál muestra las principales funcionalidades del proyecto. El único actor que interactúa con el sistema es el usuario, el cual representa a todo aquel que usará la aplicación.

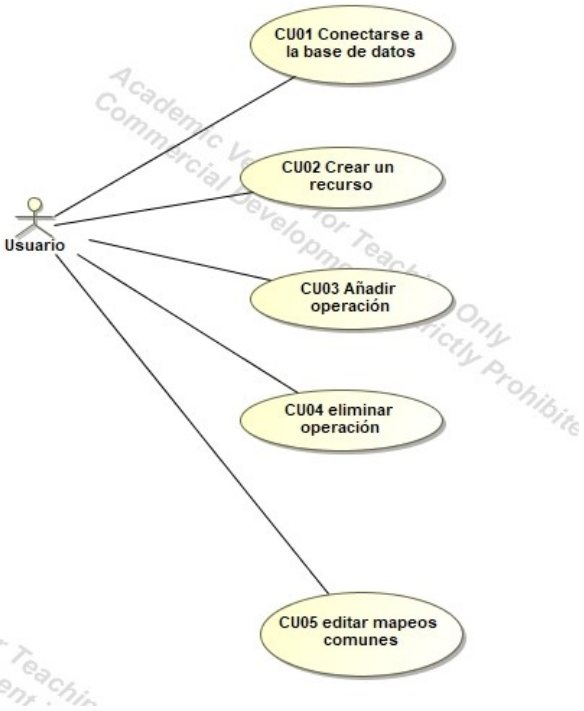

Figura 3.1: Diagrama de casos de uso

3.1.2. Historias de usuario

Las historias de usuario del proyecto pueden dividirse en las siguientes descritas a continuación:

- ***HU01 Conectarse a la BBDD***

  Como usuario de la aplicación,

  Quiero poder conectarme a la base de datos

  Para empezar a utilizar la aplicación.

- ***HU02 Crear recursos***

  Como usuario de la aplicación,

  Quiero poder crear un recurso a partir de las tablas y operaciones seleccionadas en la base de datos

  Para poder trabajar con los controllers del recurso.

- ***HU03 añadir operación***

  Como usuario de la aplicación,

  Quiero poder añadir una operación a un controlador

  Para que al controller se les añadan las operaciones escogidas que le falten.

- ***HU04 Eliminar operación***

Como usuario de la aplicación,

Quiero poder eliminar operaciones de un controller

Para quitar del controller las operaciones que no necesite.

- ***HU05 Gestionar mapeos***

  Como usuario de la aplicación,

  Quiero poder visualizar y editar los atributos en las zonas de los mapeos de cada operación de un controller

  Para que esos cambios queden guardados.

3.1.3. Diagrama de clases

En la *figura 3.2*, se puede apreciar el diagrama de clases de este proyecto, en el que se pueden apreciar las clases que forman parte del sistema junto con los respectivos atributos que las compone. Las principales clases que componen el diagrama serían la clase *Controller*, *Endpoint* y *Entity*.

Empezando por la clase *Controller*, es la que almacena cada recurso, conteniendo en él las entidades, las operaciones y el mapeo del recurso. Siguiendo por la clase *Entity*, es la clase que almacena las entidades de la aplicación, compuestas por el nombre de la entidad; el nombre de la tabla que contenía la entidad en la base de datos; y sus atributos. Finalmente la clase *Endpoint* almacena las operaciones de cada recurso, almacenando el nombre de la operación; el método de la operación, es decir, si es get, post, put, delete, getall o postgetall; los parámetros necesarios para llamar a la operación; el cuerpo de la request con la que llamar a la operación en caso de que requiera algún dato que no se pueda pasar por parámetros; el cuerpo de la respuesta, lo que devolverá la operación tras ser llamada; y el nombre de la entidad relacionada.

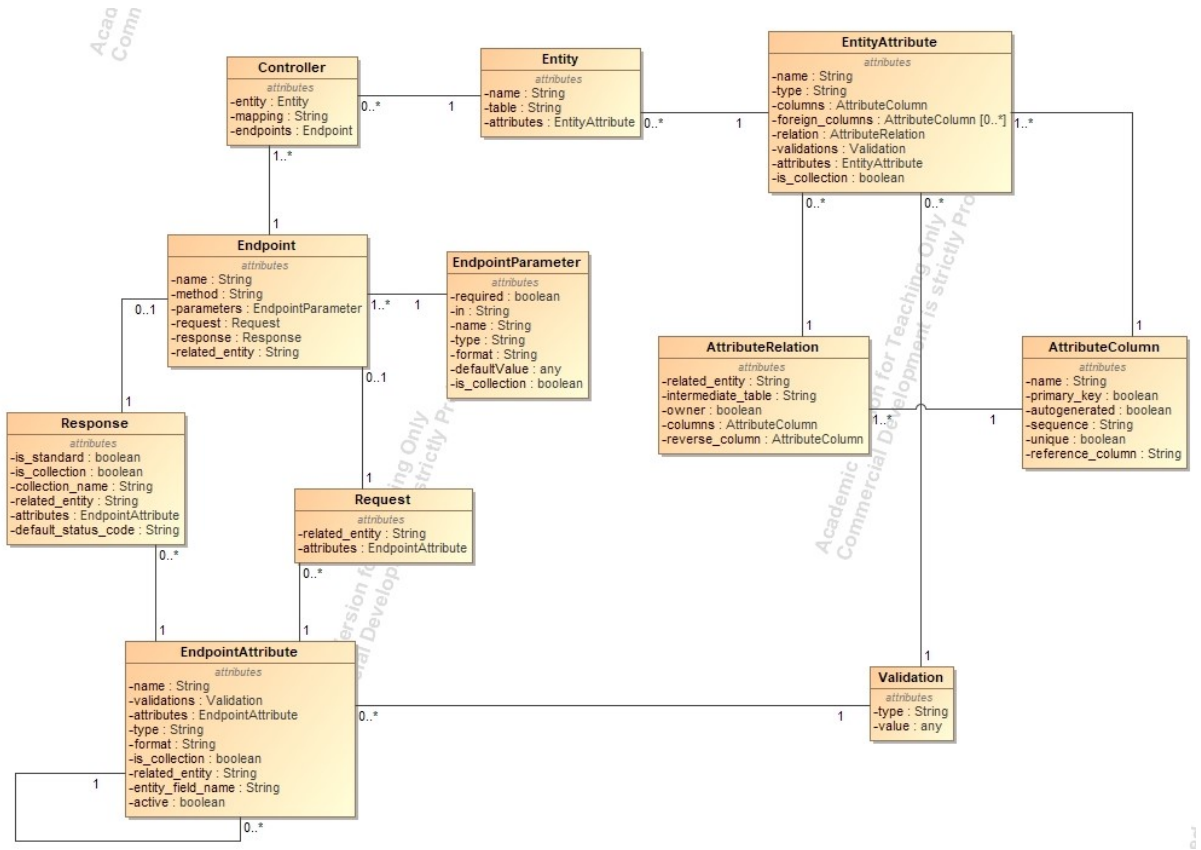

Figura 3.2: Diagrama de casos de uso

3.2. Diseño de la arquitectura sistema

En la *figura 3.3*, se puede apreciar la arquitectura principal del sistema, siguiendo una arquitectura básica de cliente servidor. Donde el cliente, la parte del frontend, realiza peticiones a una APIrest alojada en un servidor, que a su vez se comunica con una base de datos, de dónde saca la información necesaria para responder al cliente mediante una respuesta HTTP, que el frontend procesa para poder mostrar esa información de forma clara al usuario.

3.3. Diseño de la interfaz

Para realizar la interfaz de usuario, la empresa decidió que visualmente se pareciera a la aplicación que tomaba de referencia, de forma que los colores, fuente de letra y estructura básica la interfaz gráficas ya estaban decididas. Además, también se decidieron que los principales criterios a seguir fueran la uniformidad y la usabilidad, de forma que resultara más sencilla de usar para el cliente que la aplicación de la que partía.

En la *figura 3.4*, se puede observar el diseño de la página principal, donde se establecerá la conexión con la base de datos. Al introducir los datos correctos y darle a conectar, la parte de

Figura 3.3: Arquitectura del sistema

Datos de la conexión cambiará quedando como se muestra en la *figura 3.5* habilitando la opción de crear recursos.

Al pulsar el botón *Crear recursos*, aparecerá un modal como el que se muestra en la *figura 3.6*, donde podremos escoger las tablas que queremos cargar y sus respectivas operaciones, teniendo todos seleccionados por defecto para mayor usabilidad. Una vez se han seleccionado, al darle al botón generar, se crearían tantos recursos como tablas seleccionadas, en los que se podrían ver el nombre de estos, el nombre de estos, la entidad relacionada y sus operaciones, tal y como se pueden observar en las *figuras 3.7 y 3.8*.

Al presionar el botón con forma de más que se muestra en la *figura 3.8*, aparecerá un modal como el mostrado en la *figura 3.9*, dónde se podrán añadir las operaciones que falten a un recurso, mostrándose las que ya forman parte del recurso, sin la cruz que permite eliminar una operación de la selección.

En cambio, si se pulsa en el botón en forma de cruz dentro de un círculo rojo al lado de cada operación que se ve en la *figura 3.8*, aparecerá un modal como el que se ve en la *figura 3.10*, dónde si le damos a *Aceptar*, se eliminará la operación del recurso.

Finalmente, si decidimos expandir una operación, se mostrará el contenido que aparece en las *figuras 3.11, 3.12 y 3.13*, dónde se pueden ver los atributos que la operación necesitará como parámetros, los que necesitará en la request y los que devolverá como respuesta.

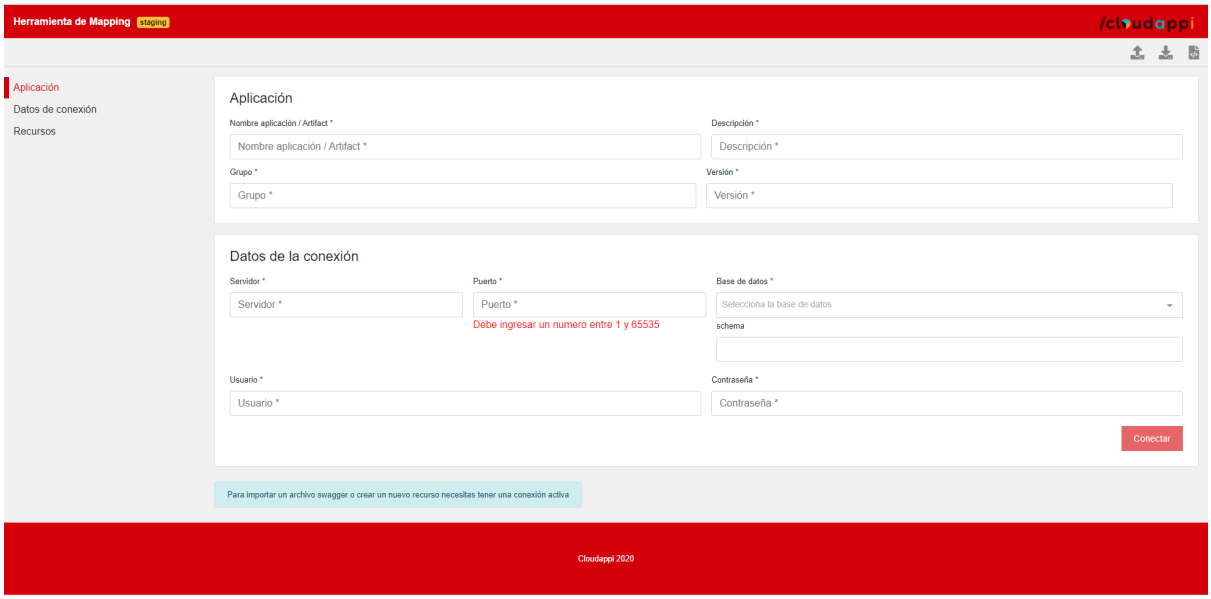

Figura 3.4: Página principal

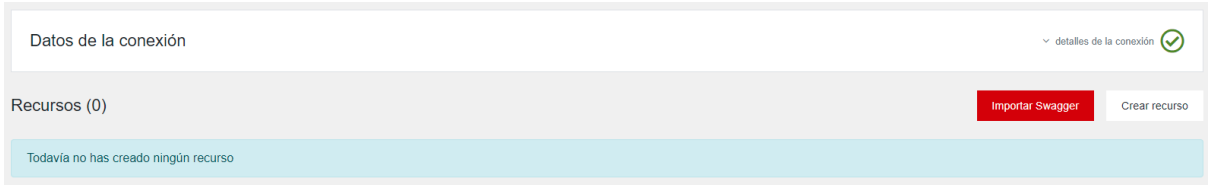

Figura 3.5: Conexión establecida

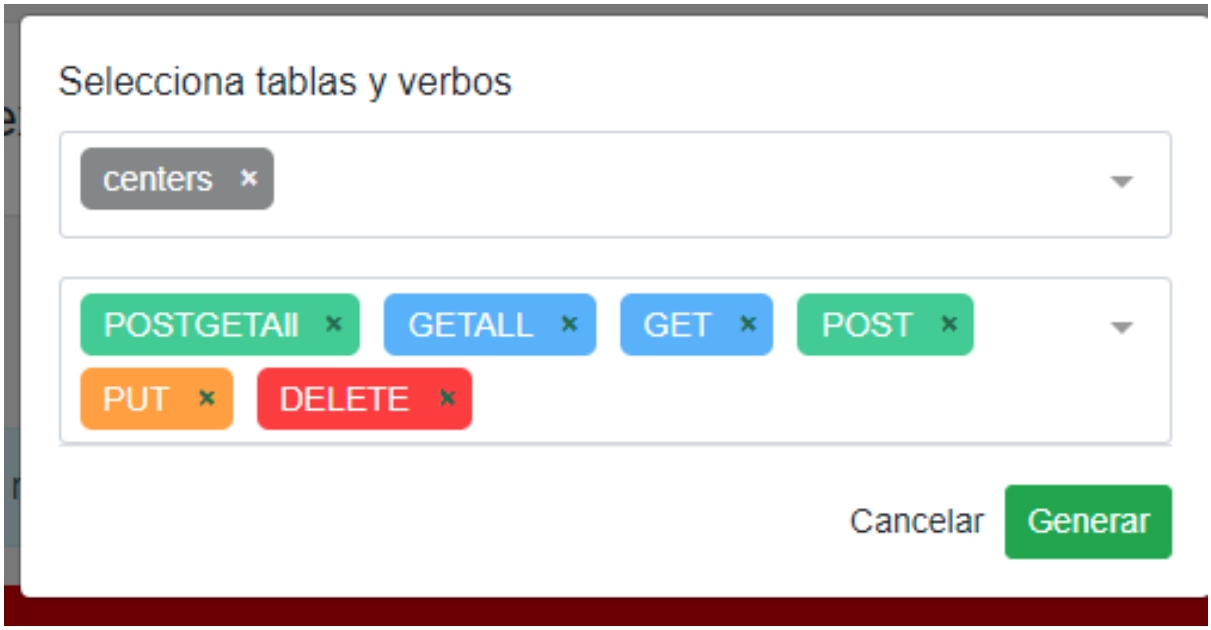

Figura 3.6: Seleccionar tablas y operaciones

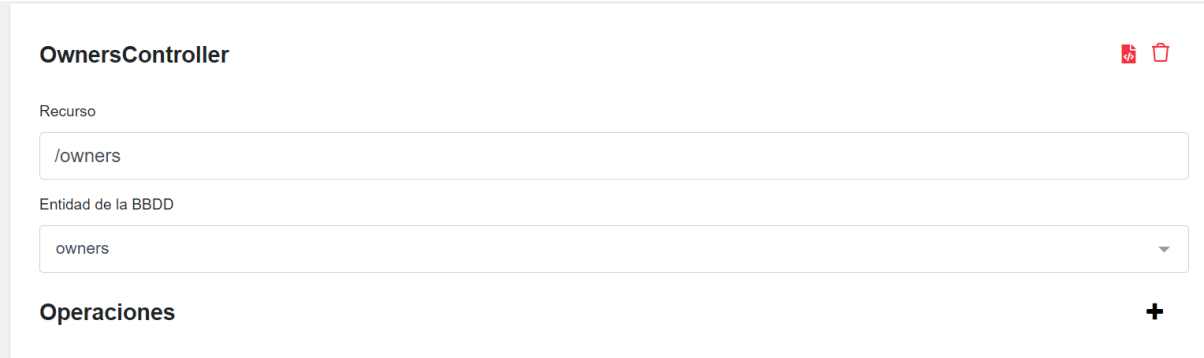

Figura 3.7: Título y entidad del controller

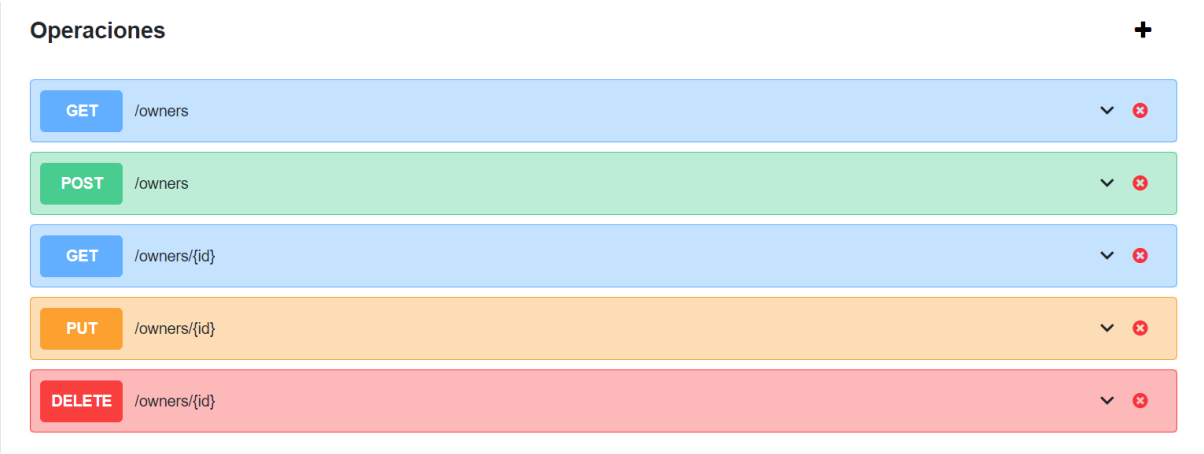

Figura 3.8: Operaciones del controller

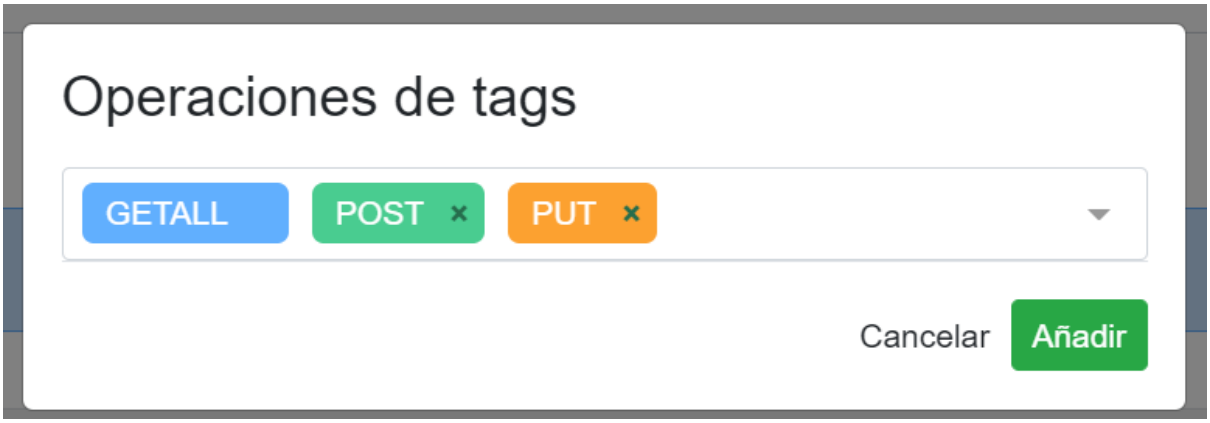

Figura 3.9: Añadir operaciones

Figura 3.10: Eliminar operación

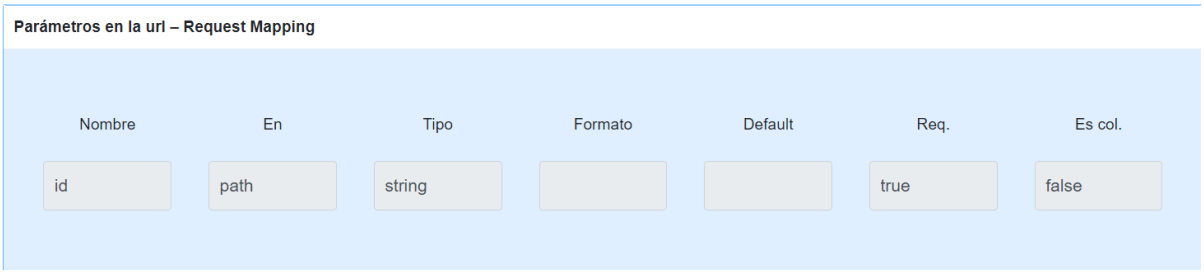

Figura 3.11: Parámetros

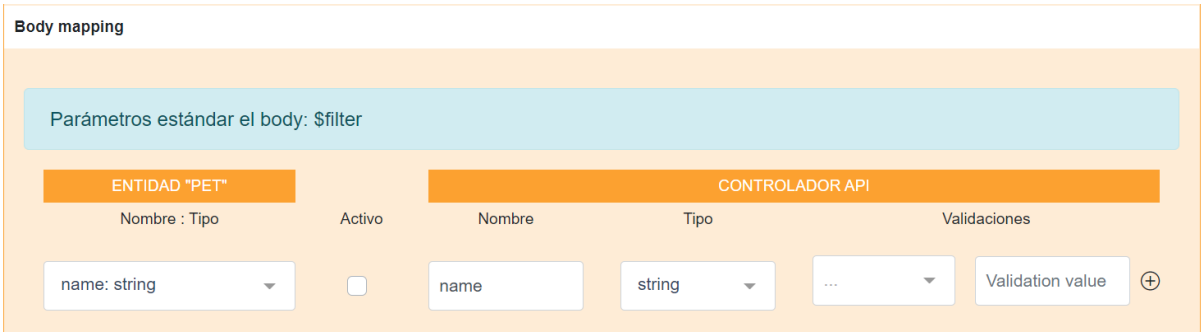

Figura 3.12: Request

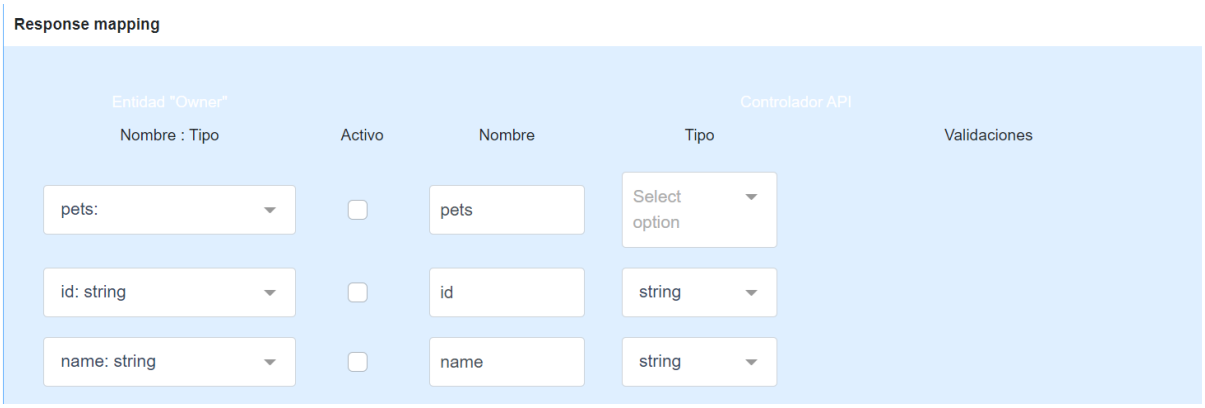

Figura 3.13: Respuesta

Capítulo 4

Implementación y pruebas

En este capítulo se detalla como ha sido la implementación del proyecto, describiendo el trabajo realizado junto a las pruebas realizadas para asegurar el correcto funcionamiento de la aplicación.

4.1. Detalles de implementación

Este proyecto se realizó utilizando el framework *Vuejs*, utilizando typescript como lenguaje principal de programación, usando además de *Vuejs*, la librería *Vuex* para tener un mejor manejo de los datos, almacenándolos en la *store*, la cuál está formada por distintas "clases" con atributos definidos como se muestra en el diagrama de clases mostrado anteriormente en la *figura 3.2*.

Además de Vuex, se usó la librería *Axios*, la cuál permite realizar peticiones HTTP, siendo el uso de esta, la forma más eficiente de comunicarse con la API rest del backend, trabajando con las respuestas obtenidas [3].

También cabe destacar, el uso de *nanoid*, una librería que permite la generación de claves únicas con las que guardar en diccionarios, objetos que no poseen ningún atributo identificador que los diferencie de otros de la misma clase [2].

Para la realización de la vista, se usó *pug*, un motor de plantilla que permite escribir código HTML de una forma más sencilla y directa [1], usando sangrado en lugar de los símbolos < y /> anidadamente para indicar cuando un componente forma parte de otro. También, para ciertos componentes se usó *bootstrap*, ya que permite usar al desarrollador componentes ya hechos, los cuáles suelen ser mucho más estéticos que componentes HTML planos. Complementando a *bootstrap*, para los iconos se usó *fontawesome*, ya que el proyecto del que partía DAPIgen también lo usaba y permitía una mayor uniformidad en la vista.

Para los estilos, se usó *scss*, una extensión de *css*, con una sintaxis mejorada respecto a *css* [5] a la hora de definir los estilos, los cuales en este proyecto se almacenan en una carpeta aparte y se pueden llamar desde la vista directamente, sin necesidad de declarar dichos estilos en el

bloque *style* de cada componente *.vue*, que es dónde normalmente se definen los estilos.

En cuanto al principal patrón usado en este proyecto, cabe remarcar el *MVVM*, model-view-viewmodel. Un patrón donde la vista, ubicada en el bloque *template* de cada componente *.vue*, se comunica con el modelo, ubicado en la store, a través del modelo de vista, ubicado en el bloque *script* de cada componente *.vue*, el cuál contiene además la lógica de la aplicación, dictando como se comportará la interfaz ante los eventos que surjan [4].

Finalmente, respecto a los problemas surgidos durante el desarrollo, la mayoría fueron problemas de reactividad, dados por cambiar algunas variables de forma no reactiva. Estos problemas fueron solventados al ganar experiencia y al ir aprendiendo a cómo se deben tocar las variables en caso de querer actualizarlas. En este desarrollo no fue necesario el uso de ningún algoritmo complejo, pues las tareas más costosas de realizar, en cuanto a tiempo de ejecución, fueron delegadas al backend.

Librerías usadas

nombre	versión	uso en el proyecto
Axios	0.21.1	Se usó para las comunicaciones HTTP con la API rest del backend
Nanoid	3.1.23	Se usó para generar claves únicas automáticamente
Pug	3.0.2	Se usó como substituto de HTML permitiendo un desarrollo más ágil
Bootstrap	4.6.0	Se usó para complementar la vista con componentes más estéticos
Fontawesome	5.15.2	Se usó para algunos iconos de la vista

4.2. Verificación y validación

En cuanto a las pruebas para comprobar el correcto funcionamiento de la aplicación se realizaron varias pruebas manuales además de un test unitario para comprobar el correcto funcionamiento de la funcionalidad *Añadir operación*.

Respecto a las pruebas manuales, las primeras que se hicieron fueron relacionadas a la conexión con el backend, comprobando que al rellenar los campos y establecer la conexión, la aplicación permitía comenzar a crear recursos. Las segundas pruebas manuales realizadas estuvieron relacionadas con la generación de los recursos, para comprobar, al darle al botón *Crear recurso*, que aparecía el modal con todos los verbos precargados y se podían escoger las tablas deseadas de entre todas las tablas almacenadas en la base de datos. Tras comprobar que se generaban y el backend enviaba la información correcta, se pasó a comprobar el correcto funcionamiento de la funcionalidad *Eliminar una operación*, apretando a la cruz envuelta en un círculo rojo de cada operación, comprobando que efectivamente se eliminaba la operación del recurso. Finalmente, se comprobó que las operaciones de los recursos creados tenían los atributos correctos, aunque para no depender del backend, se implementó un servidor mockup que enviaba una respuesta en formato json con los datos necesarios para construir varios recursos, algunos con atributos compuestos en sus operaciones.

Finalmente, las pruebas unitarias se centraron en comprobar el correcto funcionamiento del modal *Añadir operación*, usando la función *mount()* de *jest*, para asegurarse de que el HTML generado fuera el correcto, probando dos escenarios: Uno dónde el recurso no tenía operaciones y otro dónde el recurso ya tenía operaciones, comprobando que en el select del primer caso dejaría escoger entre todos los verbos mientras que en el segundo solo de los verbos que no estaban en el recurso.

Capítulo 5

Conclusiones

Se pueden presentar conclusiones en varios aspectos: en el ámbito formativo (sobre lo que has aprendido), en el ámbito profesional (sobre la experiencia en la empresa) y en el ámbito personal (sobre tu experiencia personal).

En el ámbito formativo, considero que este proyecto ha sido bastante productivo, pues me ha permitido aprender sobre tecnologías que nunca antes había usado como es el caso de *Vuejs*, *typescript* y *jest*. Además me permitió aprender sobre un nuevo paradigma de programación, la programación reactiva, y me ayudó a profundizar mis conocimientos sobre scrum.

En cuanto al ámbito profesional, creo que ha sido muy fructífero, pues aprendí un poco más del mundo laboral y obtuve un poco de experiencia en este ámbito.

Finalmente, sobre mi experiencia personal, me siento satisfecho con lo conseguido en este proyecto, pues a pesar de los momentos de frustración cuando surgía algún error, conseguí seguir adelante y hacer una aplicación web funcional.

Capítulo 6

Bibliografía

[1] Alfonso Serrano. Introducción y primeros pasos con pug. https://www.silocreativo.com/introduccion-primeros-pasos-pug/. [Consulta: 09 de Julio de 2021].

[2] Andrey Sitnik. Nano id. https://www.npmjs.com/package/nanoid. [Consulta: 09 de Julio de 2021].

[3] Brian Anglin. Axios. https://github.com/axios/axios. [Consulta: 09 de Julio de 2021].

[4] Jeremy Likness. Model-view-viewmodel (mvvm) explained. https://www.wintellect.com/model-view-viewmodel-mvvm-explained/. [Consulta: 10 de Julio de 2021].

[5] Supun Kavinda. The definitive guide to scss. https://blog.logrocket.com/the-definitive-guide-to-scss/#:~:text=Officially%20described%20as%20%E2%80%9CCSS%20with,run%20them%20in%20the%20browser. [Consulta: 10 de Julio de 2021].