UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería del Software

Aplicación web sobre las restricciones ciudadanas por el COVID-19 en el territorio español

Web Application on citizen restrictions by COVID-19 in Spain

Realizado por
Serafín Cortés Ramírez

Tutorizado por
Eduardo Guzmán De los Riscos

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, septiembre de 2021

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA DEL SOFTWARE

Aplicación web sobre las restricciones ciudadanas por el COVID-19 en el territorio español

Web Application on citizen restrictions by COVID-19 in Spain

Realizado por

Serafín Cortés Ramírez

Tutorizado por

Eduardo Guzmán De los Riscos

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2021

Fecha defensa: septiembre de 2021

Abstract

Due to the health situation that has been produced by COVID-19, it has been set different restrictions, limitations and guidelines throughout of the advance of the pandemic and the development of the cure in all countries, in order to maintain some levels of health in each of these. In the case of Spain, in each autonomous community and for each province there have been different norms and guidelines for action. This added to the number of media diverse and all the information available on the internet, make it difficult to follow all current restrictions and regulations. Therefore, an application is sought that concentrate all this information up to date and also inform the population and that all its inhabitants are aware of these and apply them. That each user is able to view these data in a comfortable and simple way, in addition to be able to offer this information openly to anyone who wants it use in other applications or data sources.

Today the situation with COVID-19 is better than when the idea was raised and this has led to various changes in the orientation of the project to cover future similar needs in the future. Therefore, it has ended up implementing a backend with a REST API structure developed in Django. A technology that allows us to create elements in the database quickly and a rapid development, to which with the use of Django REST Framework it is possible to offer endpoints with results in JSON format. In addition, a frontend with Angular that makes requests to it. This has made it possible to have two services, the first would offer an open data source on information and restrictions in a health emergency situation and can be used by any user both to visualize the data and to develop their own applications and make requests to the data. The second service would help the user "Common" to be able to more easily visualize said data and even show it in various ways. With this structure it is intended to promote the applications and data on information in possible pandemics and that everyone can be aware of, or even expand its use with other possible frontends developed by users.

Keywords: recommendations COVID-19, web application, Django,

REST, Angular

Resumen

Debido a la situación sanitaria que se ha producido por el COVID-19 ha sido necesario establecer diferentes restricciones, limitaciones y directrices a lo largo del avance de la pandemia y el desarrollo de la cura en todos los países, para poder mantener unos niveles de sanidad en cada uno de estos. En el caso de España, en cada comunidad autónoma y para cada provincia han existido diferentes normas y directrices de actuación. Esto sumado al número de medios de comunicación diversos y a toda la información disponible en internet, dificulta el poder seguir todas las restricciones y normativas actuales. Por ello, se busca una aplicación que concentre toda esta información con actualidad y además informar a la población y que todos sus habitantes sean conocedores de estas y las apliquen. Que cada usuario sea capaz de visualizar estos datos de forma cómoda y sencilla, además de poder ofrecer esta información de forma abierta para todo aquel que la quiera usar en otras aplicaciones o fuentes de datos.

A día de hoy la situación con el COVID-19 es mejor que cuando se planteó la idea y esto ha provocado diversos cambios en que la orientación del proyecto sea cubrir futuras necesidades similares en el futuro. Por ello, se ha acabado implementando un *backend* con estructura API REST desarrollado en Django. Una tecnología que nos permite la creación de elementos en la base de datos rápidamente y un rápido desarrollo, al que con el uso de Django REST Framework es posible ofrecer *endpoints* con resultados en formato JSON. Además también se ha desarrollado un *frontend* con Angular que realiza peticiones a este. Esto ha permitido disponer de dos servicios, el primero ofrecería una fuente de datos abierta sobre información y restricciones en una situación de emergencia sanitaria y puede ser utilizada por cualquier usuario tanto para visualizar los datos cómo para desarrollar sus propias aplicaciones y hacer peticiones a los datos. El segundo servicio ayudaría al usuario "común" a poder visualizar con mayor facilidad dichos datos e incluso mostrarlos de diversas formas. Con esta estructura se piensa promover las aplicaciones y datos sobre información en posibles pandemias y que todo el mundo pueda estar

al tanto, o incluso se amplíe su uso con otros posibles *frontends* desarrollados por otros usuarios.

Palabras clave: recomendaciones COVID-19, aplicación web, Django, REST, Angular

Índice

1 Introducción

1.1. Motivación

Desde que se declaró el COVID-19 como pandemia el 11 de marzo de 2020 por la Organización Mundial de la Salud, e incluso durante esta, cada país ha establecido diferentes restricciones para controlar la situación sanitaria y para también poder mantener un nivel mínimo en terminos económicos, sociales, turísticos, etc. Además, también se ha informado de las diferentes medidas e información para que todo el mundo pueda prevenir y reducir el contagio del virus. No obstante, en gran parte del tiempo no han habido suficientes herramientas con información actualizada, puesto que estas aún no estaban desarrolladas. Por eso, en el caso de España, si la población quería informarse sobre la situación sanitaria con el COVID-19 y las restricciones del momento la gran mayoría que se podía encontrar era desactualizada en periódicos, medios online, en los diferentes canales de televisión o en los diferentes Boletines Oficiales del Estado. Por tanto, en esta situación hubiera sido necesario disponer de alguna herramienta en la que se pueda publicar toda esta información y las personas pudieran acudir a ella para informarse con una fácil visualización, o incluso de disponer de esta misma como fuente de datos para otras herramientas visuales o manejo de datos.

1.2. Objetivos

El objetivo inicial de este Trabajo de Fin de Grado (TFG) es ofrecer una herramienta con la que las personas pudieran ver de una forma visual y fácil todas las restricciones de las diferentes regiones de España, ya que a lo largo del estado de alarma cada una de estas ha tenido una normativa distinta. Además, que también sea posible comparar las restricciones en caso de que se pudiera viajar e informarse de diferentes prácticas y actuación para que cada persona pueda protegerse y ayudar a reducir el contagio del COVID-19. Todo esto sería la parte visual de la

herramienta o *frontend*, que recibiría la información de la parte del servidor y manejo de datos o *backend*. Con ello, sería posible por un lado hacer uso de la parte visual para informarse, y por otro lado hacer uso de la parte del servidor para que otras personas puedan utilizar estos datos abiertos para crear otras herramientas visuales o de estadística. Pero, durante el desarrollo de este trabajo, la situación con el COVID-19 mejoró no solo en España sino en todo el mundo, por lo que la mayoría de las fuentes de datos abiertos del Gobierno de España y del Ministerio de Sanidad fueron desapareciendo, y actualmente con apenas restricciones en las distintas regiones del país el objetivo de este TFG es realizar esta misma idea para el COVID-19, pero que pueda reutilizarse en caso de que ocurriera otra pandemia en el futuro, o en algún país hubiera alguna emergencia sanitaria, como anteriormente también con el Ébola. Para ello, deberían de poderse almacenar datos en una Base de Datos y que estos puedan ser recibidos o devueltos por la parte *backend* y también recogidos por la parte *frontend* y ser visualizados. De esta forma, si ocurriera otra emergencia sanitaria, solo habría que cambiar levemente la aplicación para que esté en funcionamiento lo más pronto posible y no sea necesario desarrollar aplicaciones nuevas.

Por último, comentar que la parte proveedora de los datos (el *backend*) seguirá la arquitectura API REST [19]. De esta forma, cualquier usuario puede consultar los datos haciendo uso de la parte visual, o realizando peticiones a la parte del servidor. Este último devolvería los datos en formato JSON [12] y estaría abierta a cualquiera que quiera hacer uso de ello.

1.3. Metodología de Trabajo

Para el desarrollo de este TFG, se pondrá en práctica la metodología ágil [13] Prototipo [21]. Esta metodología consiste en producir un modelo temprano de un producto construido para evaluar un concepto o aprender de él. Es decir, tendríamos una construcción temprana de un producto funcional, de forma que nos permita realizar una demostración del mismo, y así poder recibir retroalimentación por parte del cliente, ya sean aspectos para refinar o nuevas ideas que añadirle. Se seguirán los siguientes pasos:

1. Determinar los objetivos
2. Desarrollo del prototipo
3. Demostración del prototipo y recibir retroalimentación

4. Pruebas

5. Implementación final

Los pasos 2 y 3 se repetirán hasta que se consiga un prototipo que cumpla los objetivos. El proceso se realizará de forma que se permita que durante la retroalimentación pueda haber cambios en los objetivos. El paso 1 ya se encontraría realizado en los apartados anteriores de Motivación y Objetivos. El paso dos a su vez estará compuesto por diferentes *sprints*[20], y estos tendrán las siguientes partes:

1. Documentación y diseño

2. Desarrollo

3. Pruebas

Es decir, que antes de que se tenga un prototipo listo para la siguiente retroalimentación, en la fase de Desarrollo se realizarán diferentes *sprints* en el que cada uno estará enfocado en la producción de cada parte del proyecto. Finalmente, pasados varios de estos *sprints*, ya estaría preparado el prototipo para la recepción de retroalimentación.

1.4. Tecnologías y Herramientas utilizadas

- **Figma**

  Es un editor de gráficos vectorial. Esta herramienta es una aplicación web de tipo software propietario. Es utilizada para el desarrollo de *mockups* de aplicaciones tanto web como móviles; aunque principalmente más orientado a este último. Esta herramienta ha sido utilizada para el desarrollo del prototipo/maqueta del *sprint* 0. Puede encontrarse en: https://www.figma.com.

- **IFMLedit**

  Es una herramienta web gratuita y *Open Source* pensada para el desarrollo de modelos utilizando el lenguaje IFML. Estas siglas vienen de: *Interaction Flow Modeling Language*; utilizado para la creación de modelos de interacción de usuario con una aplicación y el comportamiento del *frontend*. Esta tecnología ha sido utilizada para el desarrollo del diagrama IFML del *sprint* 0. Puede encontrarse en: https://www.ifmledit.org/.

- **MagicDraw**

  Es una herramienta de escritorio CASE utilizada para el desarrollo de modelos UML [9] y también para la generación de código a partir de estos. Desarrollada por No Magic y es un software propietario bajo licencia (para este trabajo se ha utilizado la licencia proporcionada por la Universidad de Málaga). Esta tecnología ha sido utilizada para el desarrollo de los modelos: "Diagrama de clases", "Diagrama de despliegue" y "Diagrama de estructura" del *sprint* 0.

- **PostgreSQL**

  Es un Sistema Gestor de Bases de Datos Relacional orientado a objetos y de código abierto. Desarrollada por PostgreSQL Global Development Group bajo la licencia "PostgreSQL License". Esta tecnología ha sido utilizada para el desarrollo de la Base de Datos de este TFG. Se indicará su uso con más precisión en el *sprint* 1. Puede encontrarse en: https://www.postgresql.org/.

- **pgModeler**

  pgModeler es una herramienta de modelado de bases de datos para PostgreSQL. Combina conceptos clásicos de relaciones de entidades con características propias de PostgreSQL. Es de código abierto con una licencia GPL. Esta herramienta ha sido utilizada para el diseño de la Base de datos antes de que esta sea generada con los modelos de Django. Puede encontrarse en: https://pgmodeler.io/.

- **PyCharm**

  PyCharm es un IDE de desarrollo específicamente para el lenguaje de programación Python. Actualmente se encuentra con dos versiones: *Community* y *Professional*. En este caso ha sido utilizada la primera versión. Ambas desarrolladas por Jetbrains, pero la primera es de código abierto y la segunda bajo licencia. Esta herramienta ha sido utilizada para el desarrollo del *backend* de la aplicación en el *sprint* 1. Puede encontrarse en: https://www.jetbrains.com/es-es/pycharm/.

- **Visual Studio Code**

  Visual Studio Code es un IDE de desarrollo multilenguaje bajo licencia MIT desarrollado por Microsoft. También posee diferentes herramientas integradas como control

de versiones con GIT[5], depuración, sintaxis, etc. Esta herramienta ha sido utilizada para el desarrollo del *frontend* en los *sprints* 1 y posteriores. Puede encontrarse en: https://code.visualstudio.com/.

- **Python**

  Python es un lenguaje de programación multiparadigma que está pensando en priorizar la legibilidad del código desarrollado. Es un lenguaje interpretado[8] y está desarrollado bajo una licencia "Python Software Foundation License". Esta tecnología ha sido utilizada para el desarrollo del *backend* de este TFG. Puede encontrarse en: https://www.python.org/.

- **Django**

  Django es un framework de desarrollo web para Python orientado a un rápido desarrollo, limpio y escalable sin dejar atrás la seguridad. Esta pensado para ser utilizado usando un patrón de diseño MVC[7]. Está desarrollado bajo la licencia de código abierto BSD. Esta tecnología ha sido utilizada para el desarrollo del *backend* en el *sprint* 1. Puede encontrarse en: https://www.djangoproject.com/.

- **Django REST Framework**

  Django REST Framework es una herramienta para ser utilizada junto a Django. Esta permite que las aplicaciones web desarrolladas tengan una estructura de API REST. Esta tecnología ha sido utilizada para el desarrollo del *backend* en el *sprint* 1. Puede encontrarse en: https://www.django-rest-framework.org/.

- **Gunicorn**

  Gunicorn es un servidor web para aplicaciones Python con soporte nativo para algunos frameworks como Django. Está desarrollado bajo licencia MIT. Esta tecnología es utilizada como servidor de aplicaciones para el *backend* de este TFG para que este sea alojado en el servicio PasS[4] Heroku. Puede encontrarse en: https://gunicorn.org/.

- **Angular**

  Angular es un framework de desarrollo de aplicaciones web. Pensado para que las aplicaciones desarrolladas sean "single-page applications"[6]. Desarrollado bajo una licencia MIT y está escrito en Typescript[14], y como hoja de estilos puede utilizar CSS[15] o

SASS[22]. Esta tecnología ha sido utilizada para el desarrollo del *frontend* en los *sprints* 1 y posteriores. Puede encontrarse en: https://angular.io/.

- **Angular Material**

  Angular Material es una libreria de componentes, diseño e infraestructura para Angular. Está desarrollado bajo una licencia MIT. Puede encontrarse en: https://material.angular.io/.

- **Leaflet**

  Leaflet es una librería escrita en JavaScript[16] para el uso de mapas interactivos en aplicaciones web. Puede ser importado dentro de un proyecto Angular y utilizado. Está desarrollado bajo una licencia BSD-2. Esta tecnología ha sido utilizada en una de las secciones del *frontend*. Puede encontrarse en: https://leafletjs.com/.

- **NPM**

  Como las siglas indican: "Node Package Manager". Es un gestor de paquetes de NodeJS y un entorno de ejecución de Javascript. Está desarrollado bajo licencia "Artistic License 2.0". Esta herramienta es utilizada debido a que es necesaria para el uso de Angular. Puede encontrarse en: https://www.npmjs.com/.

- **NodeJS**

  NodeJS es un entorno de ejecución multiplataforma basado en el lenguaje JavaScript. Orientado a la capa del servidor. Frameworks como Angular utilizan esta tecnología para su desarrollo. Está desarrollado bajo una licencia MIT. Puede encontrarse en: https://nodejs.org/es/.

- **Heroku**

  Heroku es un PaaS de computación en la nube que soporta múltiples lenguajes de programación. Su propietario es salesforce.com. Esta herramienta ha sido utilizada para poder desplegar tanto el *backend* como el *frontend* de este TFG en la nube y que pueda ser utilizado más allá de en un entorno local. Puede encontrarse en: https://www.heroku.com.

1.5. Estructura del documento

La estructura del documento es la siguiente:

- **Introducción:**

  En esta sección se habla de varios puntos. En el primero, Motivación, se habla acerca de cómo ha surgido la idea de este TFG. El segundo, Objetivos, se habla sobre cuál es la solución que se plantea al problema del punto anterior. El tercero, Metodología de Trabajo, se comenta cuál será la metodología de trabajo en el desarrollo de este TFG. Y el último punto, Tecnologías y Herramientas utilizadas, se describen brevemente las tecnologías utilizadas en este TFG.

- **Desarrollo:**

  Se describe a lo largo de todo el apartado el proceso de desarrollo de este TFG y las etapas que conlleva.

- **Conclusiones y Líneas Futuras:**

  En este apartado se comenta en primer lugar las conclusiones que han podido sacarse a raíz del desarrollo de este TFG. En el punto de Líneas Futuras se habla sobre los posibles cambios y mejoras que pueden recibir las herramientas desarrolladas para aumentar su funcionalidad y/o corregirla.

2 Desarrollo

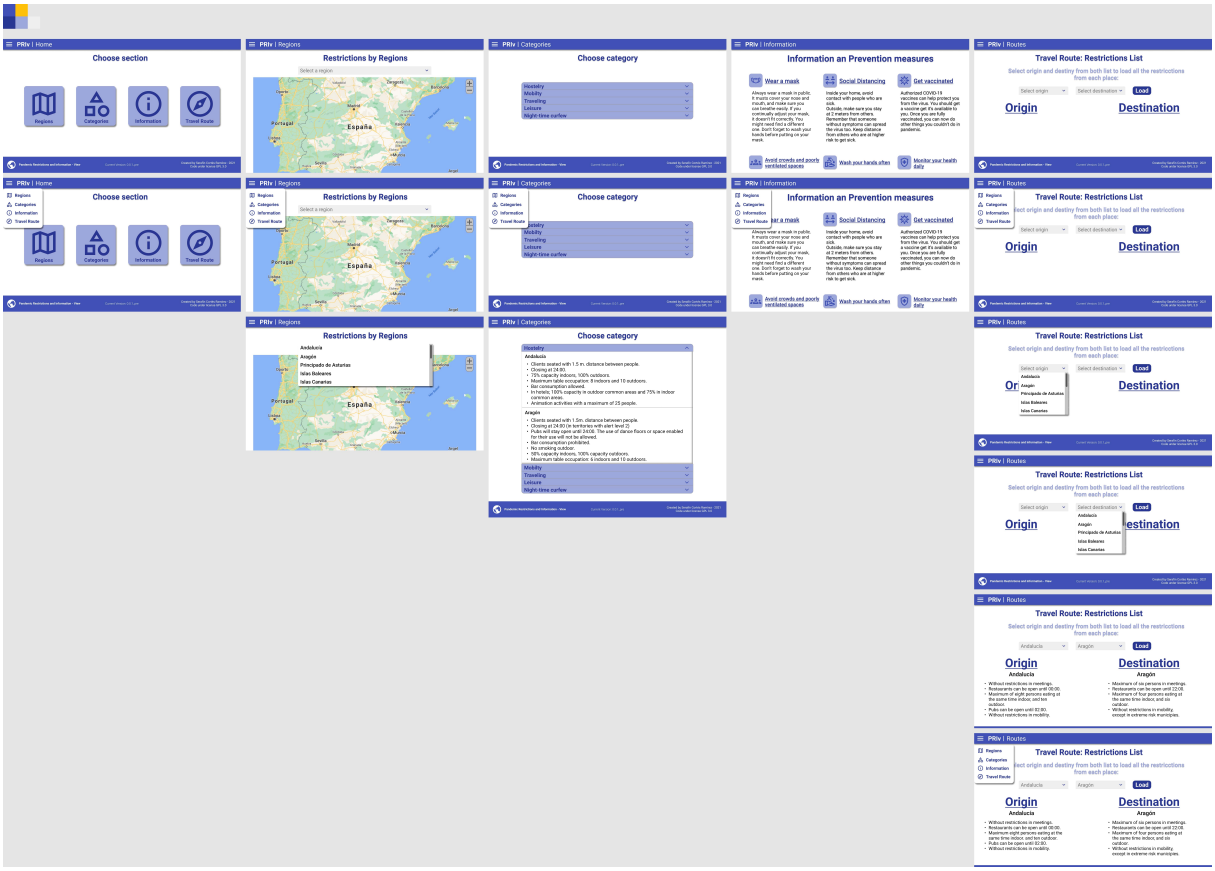
En apartados anteriores se ha comentado que el desarrollo se realizará en diferentes *sprints*. Por ello, se dividirá esta sección en cada uno de estos y se explicará lo que se ha realizado. Al ser así, en cada *sprint* lo que desarrollemos no tiene por qué estar sujeto a toda la documentación anterior como puede ocurrir con la metodología en cascada[17], sino que según se haga el desarrollo pueden modificarse tanto la documentación como el propio proyecto.
También es necesario recordar que a lo largo de este desarrollo se ha planteado que la aplicación sea la que contenga los datos, siendo estos almacenados en la base de datos del *backend*; a diferencia del planteamiento inicial en el que se pensaba más en la recogida de los datos desde páginas webs de datos abiertos como puede ser la página del Ministerio de Sanidad de España o `https://datos.gob.es/`.

2.1. Sprint 0 - Diseño y creación de diagramas iniciales

Como primer *sprint* primero se realizará el desarrollo de la idea. Esto se hará mediante el uso de diferentes diagramas y "maquetas" o "prototipos" que permitan representar un posible funcionamiento del proyecto.
En primer lugar, ha sido necesario desarrollar un *mockup*[18] que contenga la idea de la aplicación y que pueda representar un funcionamiento cercano al esperado. Este *mockup* demostraría, sin necesidad de programación, cómo quedaría la parte visual, es decir, el *frontend*. Aunque en los apartados anteriores se comentó que la aplicación tendría una parte visual y una parte de servidor (*backend*), y estos serían independientes (se podría utilizar el *frontend* para visualización de datos más cómoda, o directamente el *backend* para recibir la información con formato JSON). La realización de esta maqueta ayuda a poder imaginar cómo podría distribuirse la información y/o un posible funcionamiento. Para ello, se ha usado la herramienta online Figma. Esta aplicación web permite el desarrollo de *mockups* mediante el uso de gráficos vectoriales.

Se pueden crear todas las vistas mediante elementos como cajas, textos, imagenes, etc. y a cada uno de estos asignar un comportamiento. Tras todo ello, se puede probar el resultado y ver la maqueta de lo que sería la aplicación en un formato que recuerda al de las presentaciones. Para el caso de este TFG [1], se puede ver en la Figura 1 cuál es el resultado de todas las ventanas de la maqueta. En esta se ha colocado en cada columna cada una de las secciones de la vista

Figura 1: Visualización de todas las ventanas de la maqueta del *frontend* del proyecto



según la idea que se tenía en mente. Hacia abajo en cada columna se encuentra cada una de las distintas ventanas existentes de esa sección. Estas son:

- **Home:** Sirve para poder acceder rápidamente a cualquiera de las secciones. Se puede llegar a esta ventana pulsando en el texto de PRIv[2]. También es posible navegar entre las secciones usando el desplegable de la esquina superior izquierda.

[1]Enlace al mockup del proyecto en Figma: https://www.figma.com/file/3uh32KUJhqsQVvUu9IEjYK/PRIv_prototipo?node-id=0%3A1

[2]Este es el nombre que recibe la aplicación en su totalidad. Son las siglas de *Pandemic Restrictions and Information* y al ser la vista, se añade una "v" al final.

- **Regions:** En esta sección se ofrece la posibilidad de seleccionar una región en un desplegable y mostrar todas las restricciones en esta. Además, contiene un mapa de consulta del país en el que se esté centrando la aplicación en ese momento. Para este caso, las regiones son las comunidades autónomas de España.
- **Categories:** En esta sección se podrán abrir desplegables por cada una de las categorias disponibles en la aplicación. En estos se mostrarán todas las restricciones pertenecientes a esta categoría de cada una de las regiones.
- **Information:** Aquí podrá verse información relativa a sanidad, medidas preventivas y posibles acciones que se pudieran realizar durante el transcurso de la pandemia o la existencia de la enfermedad.
- **Routes:** En esta última sección, el usuario podrá formar una ruta de viaje seleccionando una región origen y una región destino. Finalmente, se mostrarán las restricciones de estas involucradas. De esta forma, el usuario sería capaz de poder visualizar diferencias entre las regiones para así poder planificar su viaje o ver si este fuera posible.

Nótese que este *mockup* se ha realizado en inglés por comodidad en su desarrollo, pero ello no tiene por qué implicar que la aplicación se encuentre en inglés o ser este el único idioma.

Con esto se encontraría plasmada la idea inicial de este Trabajo de Fin de Grado, no obstante, visualizando este diseño se puede ver que la disposición de los elementos y componentes en cada sección puede no ser la deseada o incluso llegar a ser algo incómoda para el usuario. Por ello, en posteriores *sprints* se valorará qué aspectos pueden quedarse tal y cómo se encuentran y/o mejorarse.

Una vez la idea ha pasado de ser un concepto a un elemento visual, es hora de formalizarla. Para ello, será necesario utilizar diferentes diagramas y documentos que nos ayudarán a imaginar cómo desarrollar el proyecto:

- **Requisitos funcionales y no funcionales**
  Los requisitos nos ayudarán a fijarnos en los aspectos que deberá tener el proyecto:
  - **Requisitos funcionales**

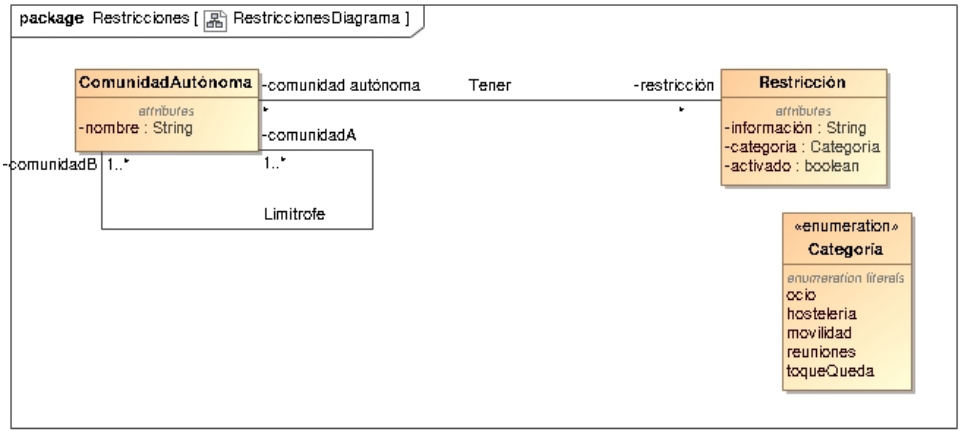
    - El *backend* dispone de una ventana de administración con los datos de la base de datos.
    - El *backend* dispone de credenciales para la ventana de administración.
    - El *backend* dispone en la ventana de administración opciones para crear, editar, eliminar y ver los datos de la base de datos.
    - Cualquier usuario puede realizar peticiones al *backend* de la aplicación y recibir un JSON con los datos solicitados desde una aplicación o desde el navegador.
    - El *frontend* dispone de una ventana inicial con botones al resto de las secciones.
    - El *frontend* dispone de una sección llamada "Regiones" donde será posible consultar las restricciones de una región seleccionada. Además dispondrá de un mapa de consulta.
    - El *frontend* dispone de una sección llamada "Categorías" donde será posible consultar todas las restricciones por cada categoría de restricción definida.
    - El *frontend* dispone de una sección llamada "Información" donde aparecerán datos sobre información sanitaria y prevención.
    - El *frontend* dispone de una sección llamada "Ruta viaje" donde un usuario podrá elegir una región de inicio e ir eligiendo las zonas limítrofes hasta una región final; para que finalmente te muestre todas las restricciones del camino elegido.

  - **Requisitos no funcionales**
    - Tanto el *backend* como el *frontend* se alojarán en Heroku.
    - Las credenciales de los usuarios de administración deben de ser seguras.
    - El *backend* y el *frontend* serán independientes entre sí.
    - La base de datos se alojará junto al *backend*.

- **Diagramas de clases**

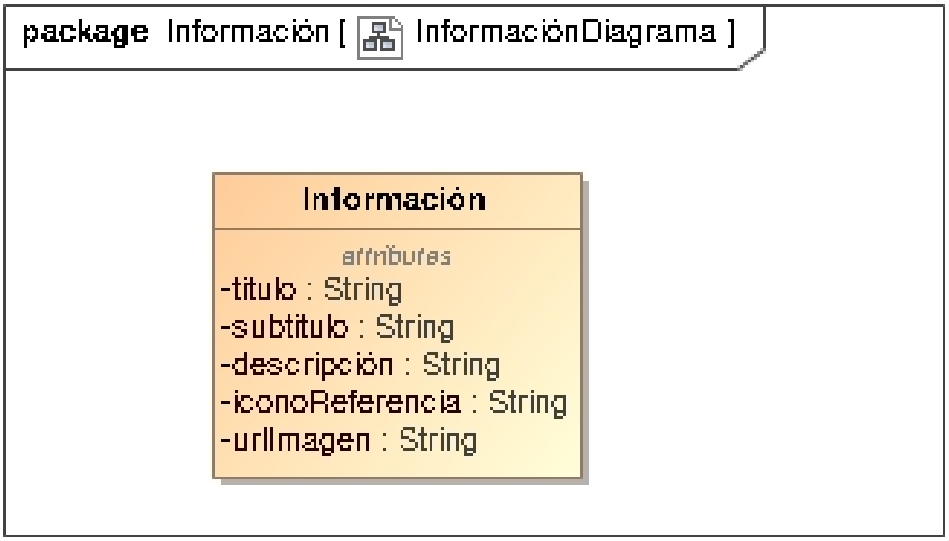
  Es necesario tener en cuenta cuáles son las entidades de la aplicación y de los datos que se van a manejar. El primer diagrama es el que se puede ver en la Figura 2. En este se

Figura 2: Diagrama de clases entre las comunidades autónomas y restricciones.



representan todas las clases que tienen que ver con respecto a las restricciones. Cada comunidad autónoma puede tener ninguna o varias restricciones asociadas, pero también una misma restricción puede aplicarse en varias comunidades autónomas. También la clase ComunidadAutónoma tiene una relación a sí misma de uno a muchos elementos. Esta representa que una comunidad autónoma es limítrofe con otra. Por último en este diagrama se puede observar el tipo enumerado "categoría" que representa los tipos de restricciones que se pueden contemplar.

Figura 3: Diagrama de clases con la clase "Información".



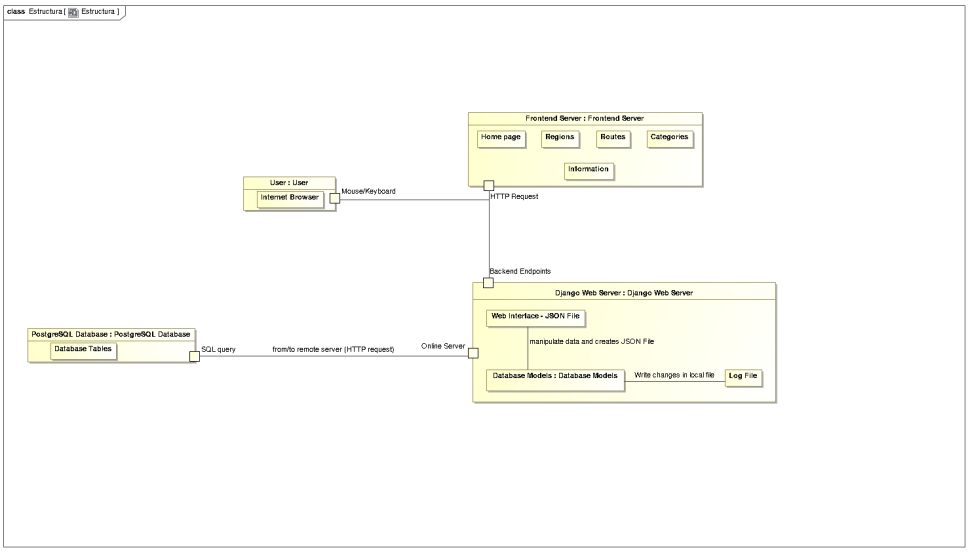
El segundo diagrama es el que puede verse en la Figura 3. En este simplemente se representa la clase "Información" que es la que contendrá las medidas de prevención e

información en una pandemia.

- **Diagrama de estructura**

  En la Figura 4 puede verse el diagrama de estructura que representa cómo se encuentra organizado este Trabajo de Fin de Grado y las distintas partes que lo componen y los componentes y/o elementos de cada una de estas.

Figura 4: Diagrama de Estructura



- **Diagrama de despliegue**

  En la Figura 5 puede verse este diagrama. Aunque puede ser similar al anterior, en este diagrama se representa el cómo y donde se encontrarían desplegadas cada una de las partes del proyecto para su uso o incluso la interacción de un usuario.

- **Diagrama IFML**

  En la Figura 6 puede verse el diagrama IFML. Este diagrama es útil para representar la interacción entre las diferentes secciones que pueden encontrarse en el *backend*.

Figura 5: Diagrama de Despliegue

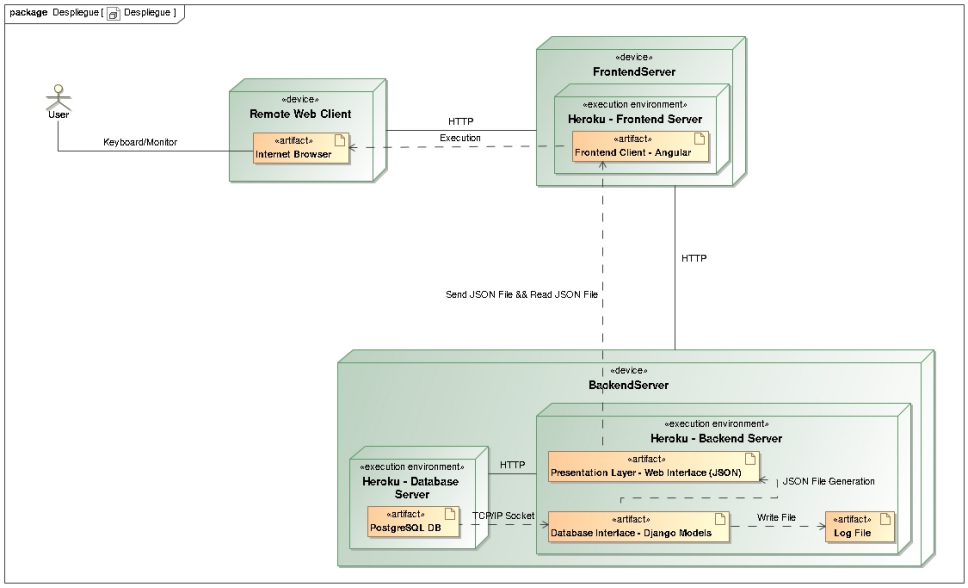
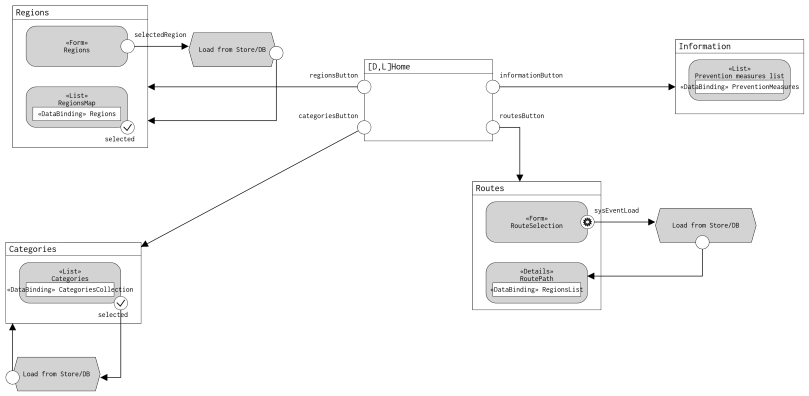


Figura 6: Diagrama IFML



Ahora que la idea se encuentra más formalizada, ya se puede pasar a realizar los *sprints* de desarrollo.

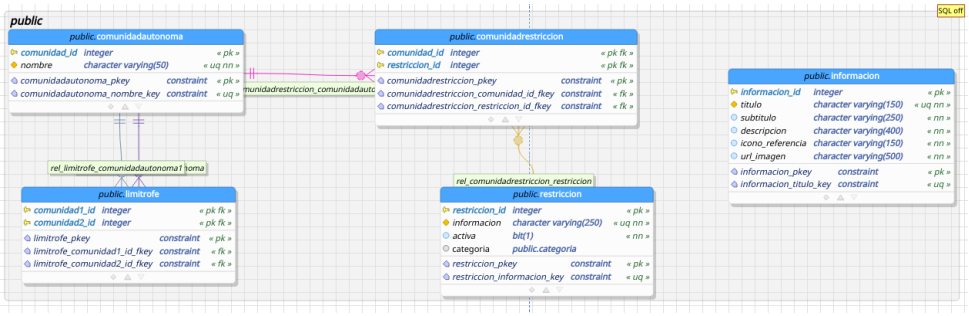
2.2. Sprint 1 - Desarrollo del *Backend* y conexión con la Base de Datos

En esta sección se van a desarrollar los siguientes requisitos (mencionados en el*sprint* 0):

- El *backend* dispone de una ventana de administración con los datos de la base de datos.
- El *backend* dispone de credenciales para la ventana de administración.
- El *backend* dispone en la ventana de administración opciones para crear, editar, eliminar y ver los datos de la base de datos.
- Cualquier usuario puede realizar peticiones al *backend* de la aplicación y recibir un JSON con los datos solicitados desde una aplicación o desde el navegador.
- Las credenciales de los usuarios de administración deben de ser seguras (esto lo hará Django automáticamente.

Para empezar con el desarrollo del *backend* antes es necesario "convertir" los Diagramas de Clases realizados en el *sprint* anterior a un modelo de Base de datos. La base de datos que se va a utilizar va a ser PostgreSQL, ya que es una base de datos relacional de código abierto y que además funciona bastante bien con Python y Django. Para dicho modelo se usará pgModeler.

Figura 7: Modelo de la Base de datos con PostgreSQL



En la Figura 7 puede verse cómo quedaría el modelo de la base de datos. La relación "Limítrofe" de muchos a muchos elementos y la relación "Comunidad-Restricción" también de muchos a muchos elementos ahora se representa como una tabla cada una. Esto es debido a que la única forma en base de datos relacionales de representar dicha relación de muchos a muchos elementos es con una tabla relacionada con las otras dos con relaciones de uno a muchos elementos.

Ahora lo normal sería crear un *script* de SQL[2] para crear estas tablas en nuestra base de datos local, pero gracias a las facilidades que aporta el framework de Python, Django, no es necesario que se haga este paso.

En la Figura 8 puede verse como referencia el caso de uso de añadir un elemento a la base de datos.

Figura 8: Caso de uso - Añadir un elemento a la tabla de Información

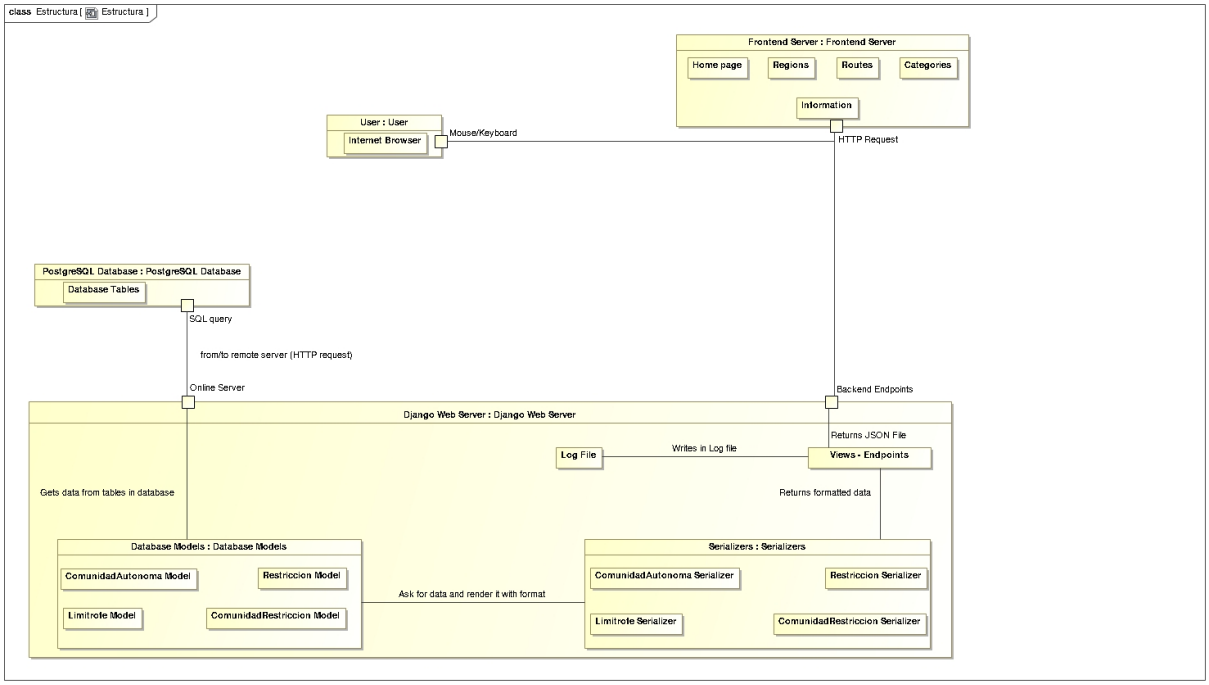
Caso de uso - Backend - Añadir información

Título	Añadir elemento de tipo Información
Descripción	El usuario administrador añade un nuevo dato a la tabla de Información desde la ventana de administración del *backend*.
Pre-condición	El usuario administrador ha iniciado sesión con sus credenciales en la ventana de administración del *backend*.
Post-condición	Se añade un nuevo elemento a la tabla de Información de la base de datos.
Prioridad	Alta.
Autor(es)	Serafín Cortés Ramírez
Escenario principal	
1. El usuario pulsa en la tabla "Información". 2. El sistema le devuelve una lista de todos los objetos que se encuentran en la tabla de "Información". 3. El usuario pulsa el botón "Add Informacion" 4. El sistema muestra un formulario con los siguientes campos: Titulo, Subtitulo, Descripción, Icono Referencia y Url Imagen. 5. El usuario rellena el formulario con los datos a introducir. 6. El usuario pulsa el botón "Save". 7. El sistema almacena en la base de datos el elemento de tipo "Información" creado. 8. El sistema muestra al usuario la lista de todos los elementos de la tabla.	
Escenario alternativo	
4.a El usuario pulsa el botón "atrás" del navegador. 4.a.1 El sistema no guarda los cambios y muestra la lista de elementos de la tabla "Información".	

Por último, antes de empezar a desarrollar el *backend* es necesario concretar las partes de este *sprint* en el Diagrama de Estructura realizado previamente para que así sea posible tener en cuenta las partes de este. Tras los cambios, el documento quedaría como puede verse en la Figura 9.

Utilizar Django para el *backend* proporciona múltiples ventajas en el desarrollo de este. Una de ellas es que no es necesario que se hagan los cambios directamente sobre la base de datos sino

Figura 9: Diagrama de Estructura - Cambios para el *backend*

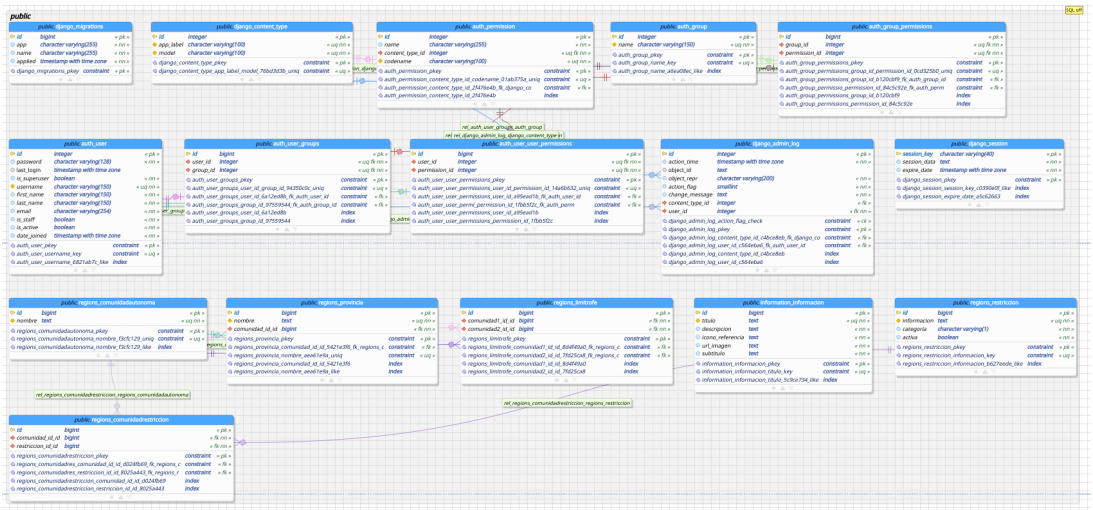


que definimos los modelos que utilizaremos y el framework se ocupa de ello automáticamente además de añadir otros elementos para su propio funcionamiento. En la Figura 10 puede verse cómo queda la base de datos tras la creación de los modelos.

Tras la definición de los modelos, se hará uso de *serializers*. Estos elementos son proporcionados por el framework: Django REST Framework y permiten obtener los datos de un modelo y convertirlos a un formato manejable por Python. Habrá un *serializer* por cada modelo de la base de datos. Después, definiremos los modelos de administración y se le indicará cuál es el modelo al que hace referencia. Esto permitirá tener una ventana de administración de Django (a la que accederemos con el *path*: */admin/*) y desde ahí poder añadir elementos a la base de datos sin necesidad de hacer consultas SQL; solo abriendo un formulario e introduciendo los datos. A esta ventana de administración se accederá con el usuario y contraseña que han sido definidos en la base de datos de PostgreSQL.

Finalmente, solo es necesario definir los *views* del proyecto. Aunque con el uso normal de Django estos se definirían con HTML y otras herramientas similares, con el uso de Django REST Framework es posible volver estas vistas en *endpoints* que al hacerles una petición

Figura 10: Estado de la base de datos tras la creación de los modelos

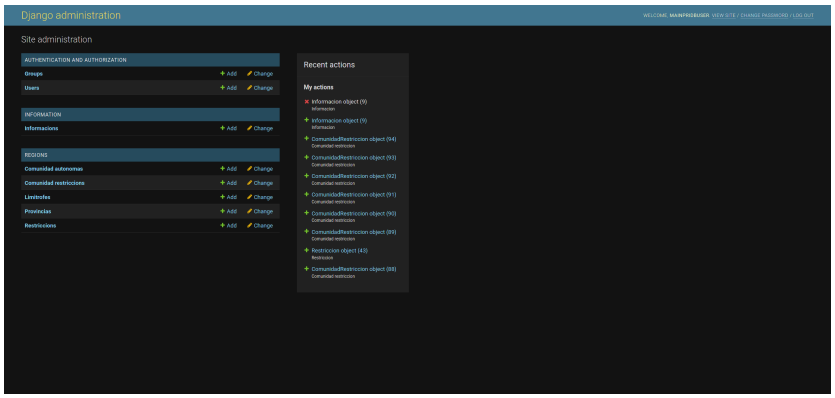


HTTP nos devolverán un fichero JSON con lo consultado.

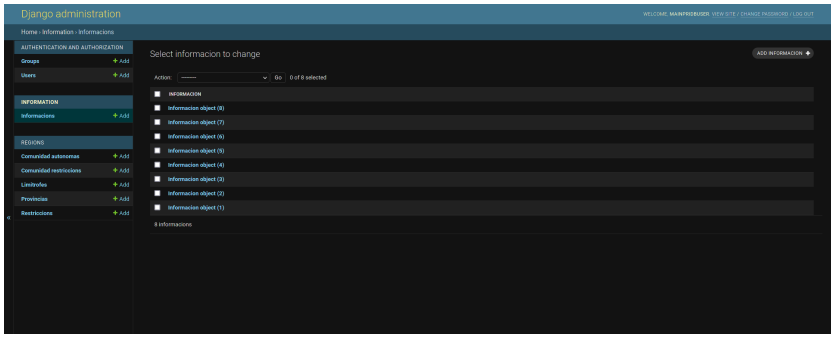
Los *endpoints* del proyecto son los siguientes:

- */regions/comunidadautonoma*
- */regions/limitrofes*
- */regions/restricciones*
- */regions/comunidadrestricciones*
- */information*
- */admin*

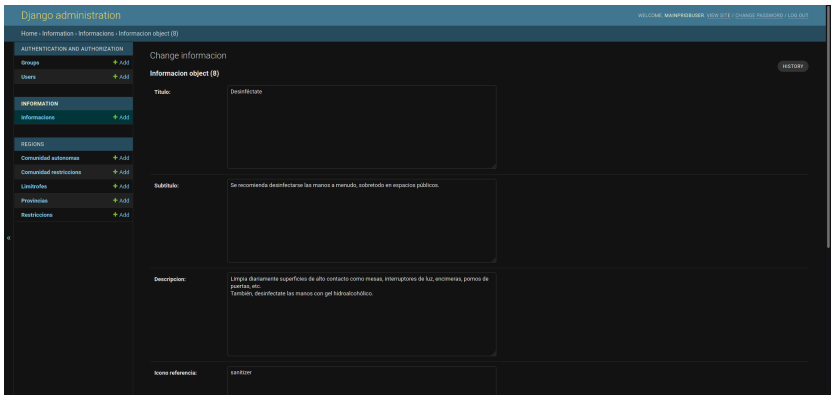
A continuación se muestran en las Figura 11 y 12 cómo queda el *backend* al terminar el *sprint*:



(a) Ventana de administración del *backend*

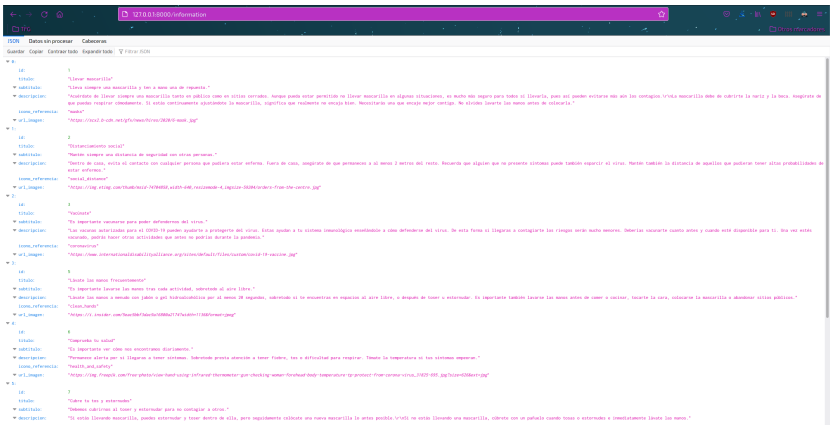


(b) Lista de los elementos de la tabla “Información”

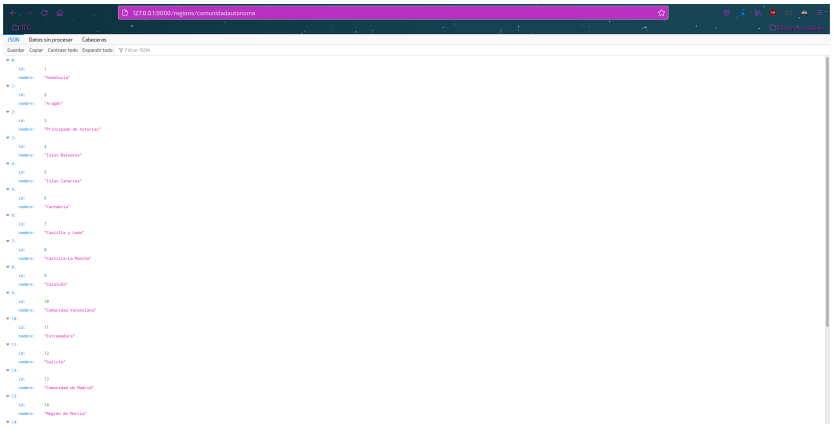


(c) Elemento de “Información” en la ventana de administración

Figura 11: Aspecto final del *backend* 1



(a) Resultado a una petición al *endpoint* de “Información”



(b) Resultado a una petición al *endpoint* de “ComunidadAutonoma”

Figura 12: Aspecto final del *backend* 2

2.3. Sprint 2 - Desarrollo del *Frontend*: Ventana Home

Ahora y en los *sprints* posteriores se desarrollará el *frontend*. Además, en cada uno se desarrollará uno de los requisitos funcionales propuestos en el *sprint* 0 correspondientes al *frontend*. Para ello se creará un proyecto de Angular. De entre los ficheros generados serán importantes los siguientes:

- app.module.ts: Aquí es donde se definirán todos los imports del proyecto de Angular.
- carpeta "app": Aquí es donde se encuentran todos los ficheros que se utilizarán para la aplicación y donde se colocarán los componentes desarrollados.

Primero hay que tener en cuenta cómo funciona Angular. Esta tecnología funciona a base de desarrollo de componentes. Estos elementos pueden ser propios de Angular, de la librería de componentes "Angular Material" que usaremos o que se hayan desarrollado. Si es el último caso, en este Trabajo de Fin de Grado los componentes tendrán 4 ficheros:

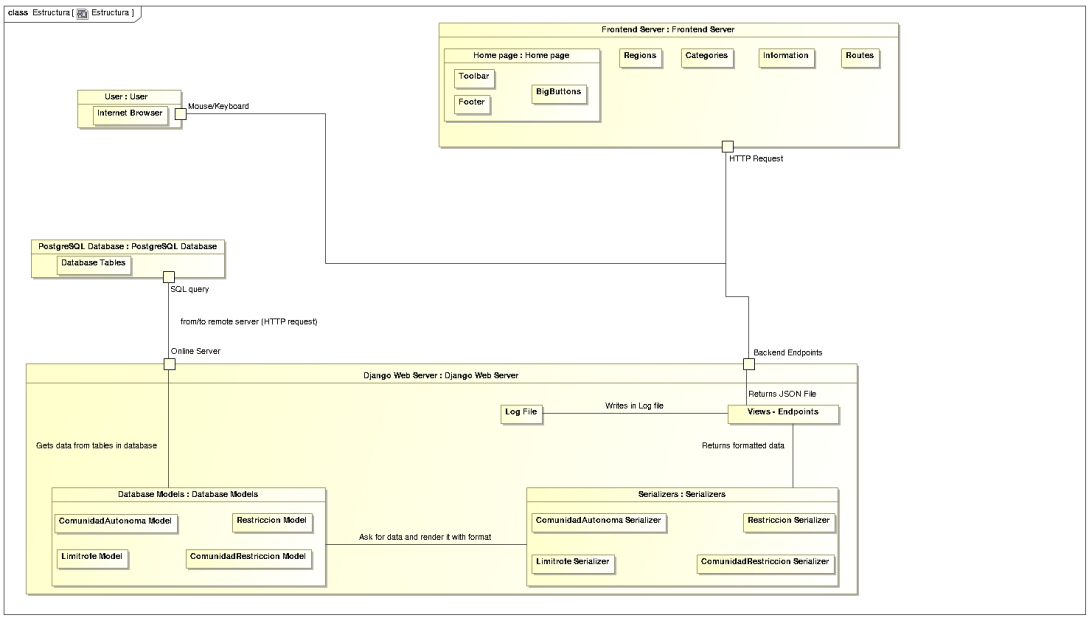
- .html: Fichero html donde se definirá la lógica visual del componente.
- .sass: Fichero de estilos que sustituye el posible fichero de estilos CSS. SASS es un lenguaje precompilado de CSS que nos permitirá tener mayor funcionalidad.
- .spec.ts: Ficheros de pruebas de los componentes.
- .ts: Fichero que define el control y el funcionamiento del componente además de obtener datos, definir clases, etc.

Ahora en el Diagrama de Estructura que se ha desarrollado en *sprints* anteriores se especificará los elementos desarrollados en este *sprint*.

En la Figura 13 puede verse que se han añadido tres componentes a la sección de "Home". Estos son:

- Toolbar
- Footer
- BigButtons

Figura 13: Diagrama de Estructura - Cambios para el *frontend*



Los dos primeros serán componentes que se desarrollarán para ser la barra superior e inferior de la página tal y como se pudo ver anteriormente en el *mockup* realizado. El último será un componente desarrollado para poder crear los botones grandes que aparecerán por cada sección de la página. Todo ello se desarrollará en un componente padre que se llamará “Home” que contendrá los componentes comentados anteriormente.

Tras todo desarrollado, puede verse en la Figura 14 el aspecto de esta parte del *frontend*.

Figura 14: Ventana principal (Home) del *frontend*

2.4. Sprint 3 - Desarrollo del *Frontend*: Sección Información

Este *sprint* se centrará en continuar el desarrollo del *frontend* por la sección de "Información". En la Figura 15 puede verse un caso de uso como referencia para esta sección.

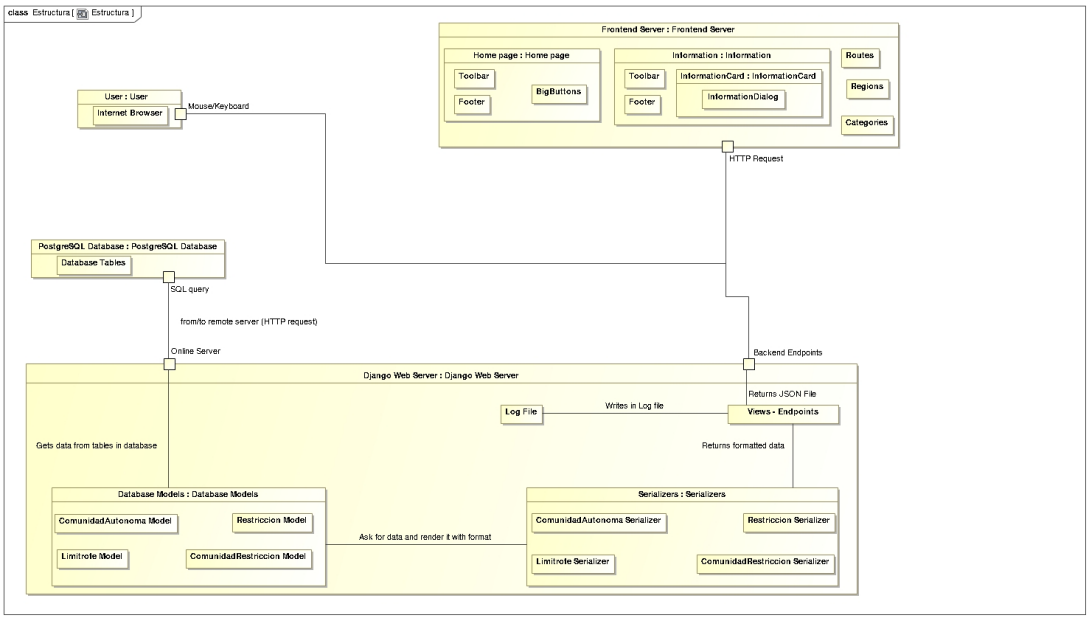
Figura 15: Caso de uso - consultar información

Caso de uso - Frontend - Consultar Información

Título	Consultar Información en la sección "Información"
Descripción	Un usuario accede a la sección de "Información" para consultar los datos sobre información y riesgo, y pulsa en una tarjeta.
Pre-condición	El usuario se encuentra con el *frontend* abierto en la ventana principal.
Post-condición	El sistema le muestra al usuario un *dialog* con más detalles de una tarjeta de información que ha pulsado.
Prioridad	Alta.
Autor(es)	Serafín Cortés Ramírez
Escenario principal	
1. El usuario pulsa el botón de la sección "Información". 2. El sistema le muestra una lista de tarjetas con todos los elementos de información y prevención de la base de datos. 3. El usuario pulsa en una de las tarjetas de información. 4. El sistema muestra un *dialog* con una descripción más detallada.	
Escenario alternativo	
3.a El usuario pulsa en el menú desplegable del toolbar y pulsa en la sección "Regiones". 3.a.2 El sistema le devuelve la sección "Regiones"	

También hay que tener en cuenta el diseño realizado en el *mockup* ya que aunque contenía una idea similar a lo que se buscaba, no poseía un diseño esperado. Por ello es necesario pensar en otro planteamiento. A continuación, se modificará el Diagrama de Estructura anterior para que nos ayude a desarrollar esta fase.

Figura 16: Diagrama de Estructura - Cambios para el *frontend*



En la figura 16 puede verse que a esta sección además de los componentes "toolbar" y "footer" creados en el *sprint* anterior se añadirá el componente "InformationCard". Este componente será el encargado de mostrar con un formato de tarjeta el título, iconoReferencia, subtitulo y una imagen de los datos de "Información" recogidos de la base de datos. También posee un componente hijo "InformationDialog" que abrirá un cuadro de texto al pulsar en la tarjeta que contendrá la descripción del objeto de información.

Para poder recibir los datos de la base de datos se creará un servicio de Angular[1] el cuál hará una petición HTTP al *endpoint*: */information*, recibirá los datos provenientes del archivo JSON y los devolverá en una función que será invocada en el componente que mantiene toda la sección. Al inicializarse este componente, se ejecutará dicha función y se cargarán los elementos asíncronamente. Por cada dato recibido se creará un componente InformationCard.

Finalmente puede verse en la Figura 17 el aspecto final de esta sección del *frontend* al final de este *sprint*.

(a) Ventana “Información” con la lista de elementos

(b) Ventana “Información” mostrando un *dialog*

Figura 17: Sección “Información” final

2.5. Sprint 4 - Desarrollo del *Frontend*: Sección Ruta Viaje

En este *sprint* se desarrollará la sección “Ruta Viaje” en la que se permitirá al usuario planificar una ruta de viaje de una comunidad autónoma a otra pasando por las comunidades limítrofes. En la Figura 18 puede verse un caso de uso de referencia a esta sección.

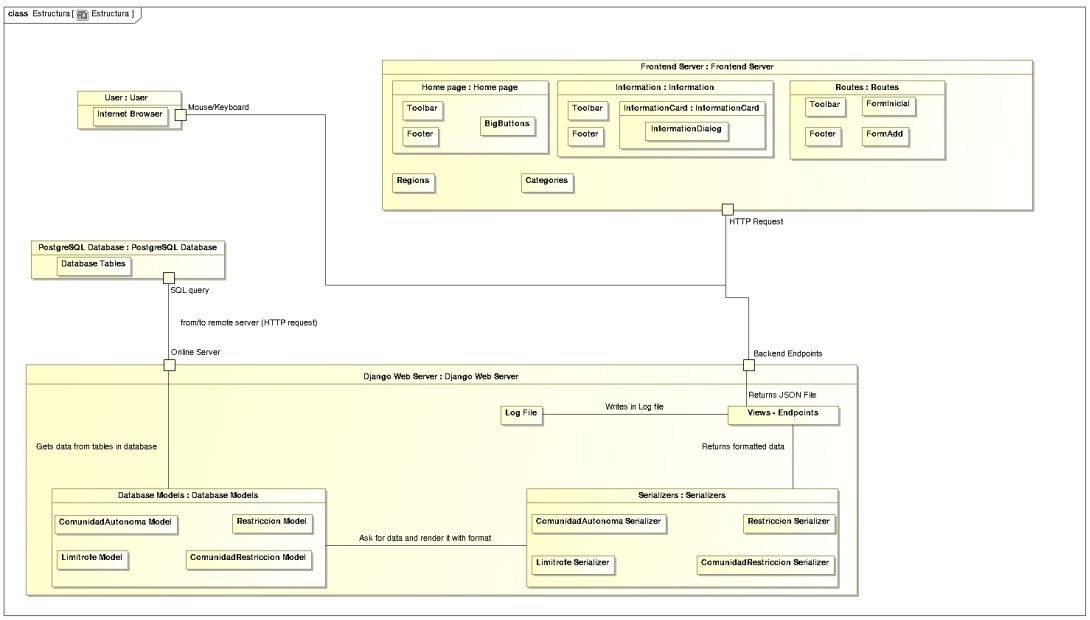
Figura 18: Caso de Uso - Planificación de Ruta

Caso de uso - Frontend - Ruta de Viaje

Título	Planificar una ruta de viaje
Descripción	Un usuario accede a la sección de “Ruta Viaje” para planificar un viaje en coche desde una comunidad autónoma a otra.
Pre-condición	El usuario se encuentra con el *frontend* abierto en la ventana principal.
Post-condición	El sistema le muestra al usuario todas las restricciones de las comunidades que ha elegido para hacer la ruta.
Prioridad	Alta.
Autor(es)	Serafín Cortés Ramírez
Escenario principal	
1. El usuario pulsa el botón de la sección “Ruta Viaje”. 2. El sistema le muestra la sección “Ruta Viaje” al usuario con un seleccionable de comunidad autónoma inicial. 3. El usuario selecciona una comunidad autónoma inicial y pulsa el botón “Empezar” 4. El sistema añade la comunidad autónoma a la lista de elegidas y muestra un seleccionable con las comunidades autónomas limítrofes a la anterior. 5. El usuario selecciona la siguiente comunidad autónoma y pulsa el botón “Añadir”. 6. El sistema añade la última comunidad autónoma a la lista de seleccionadas y carga todas las comunidades limítrofes a la anterior. 7. El usuario pulsa el botón “Finalizar”. 8. El sistema devuelve una lista con todas las restricciones de las comunidades seleccionadas.	
Escenario alternativo	
5.a El usuario pulsa el botón “Reiniciar”. 3.a.2 El sistema recarga todos los elementos y vuelve a mostrar un seleccionable con la comunidad autónoma inicial.	

Si se observa el *mockup* realizado aunque posee la idea de esta sección, el diseño quizá no es el adecuado. Por ello será necesario hacer cambios.

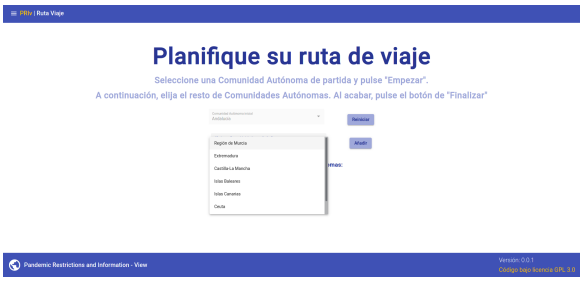
Figura 19: Diagrama de Estructura - Cambios para el *frontend*



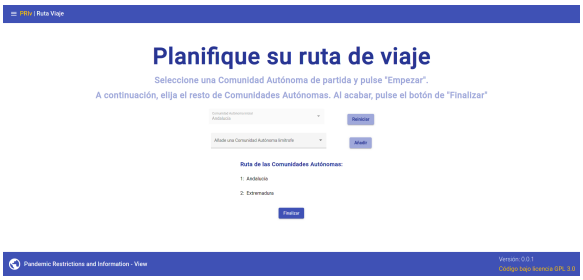
En la Figura 19 puede verse que esta sección contendrá el toolbar y el footer común para todo el *frontend* y además los componentes "formInicial" y "formAdd". Estos se añadirán al propio componente de esta sección y servirán para que el usuario pueda elegir una comunidad autónoma inicial y para que pueda ir añadiendo sucesivas a la lista de seleccionadas. Estos componentes son propios de Angular por lo que no será necesaria su creación. Para poder cargar los datos de comunidades autónomas, restricciones y comunidades limítrofes será necesario crear otro servicio de Angular para cargar asíncronamente todos los elementos.

En la Figura 20 se muestra el aspecto de esta sección al finalizar el *sprint*.

(a) Ventana “Ruta Viaje” con selección inicial



(b) Ventana “Ruta Viaje” con selección a añadir



(c) Ventana “Ruta Viaje” con selección añadida

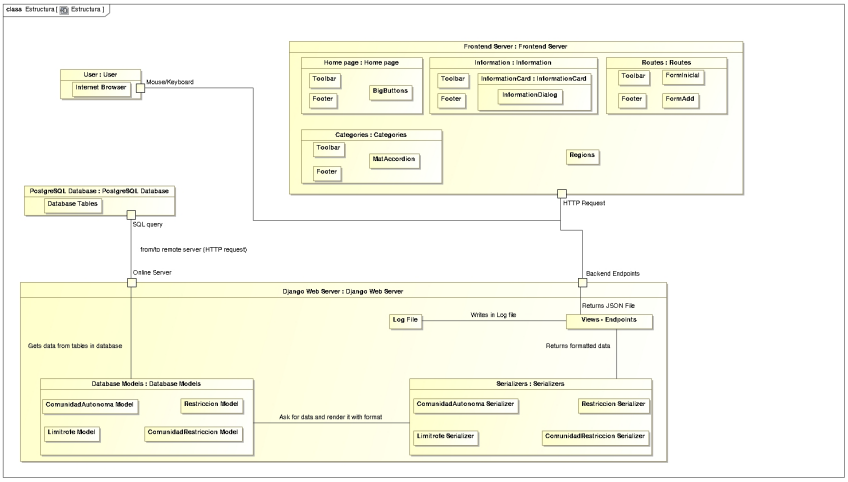
(d) Ventana “Ruta Viaje” con resultado

Figura 20: Sección “Ruta Viaje” final

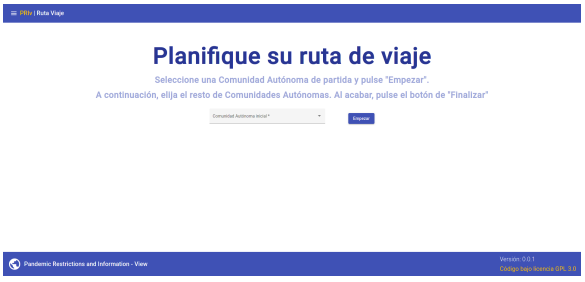
2.6. Sprint 5 - Desarrollo del *Frontend*: Sección Categorias

En este *sprint* se desarrollará la sección “Categorias”. En esta se intentará conservar el diseño y funcionalidad presentada en el *mockup* ya que se ajusta a la idea de esta sección.

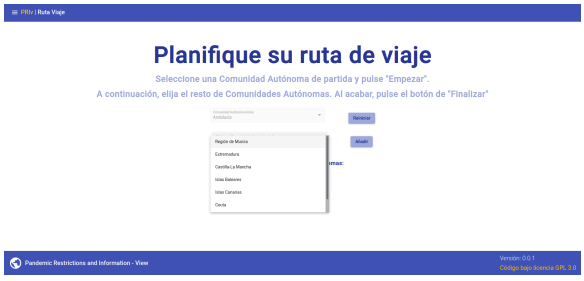
Figura 21: Diagrama de Estructura - Cambios para el *frontend*



En la Figura 21 puede verse los componentes que se encontrarán en esta sección. Aparte de los componentes “Toolbar” y “Footer” comunes se añadirá un componente llamado “MatAccordion”. Este componente es propio de Angular Material. El componente principal se encargará de mostrar un elemento dentro de “MatAccordion” según cuantas categorías reciba de la base de datos. Al pulsar en una de ellas se mostrarán todas las restricciones de dicha categoría. En la Figura 22 cómo se encuentra esta sección al finalizar el *sprint*.



(a) Ventana “Categorias” con cada categoría de las restricciones



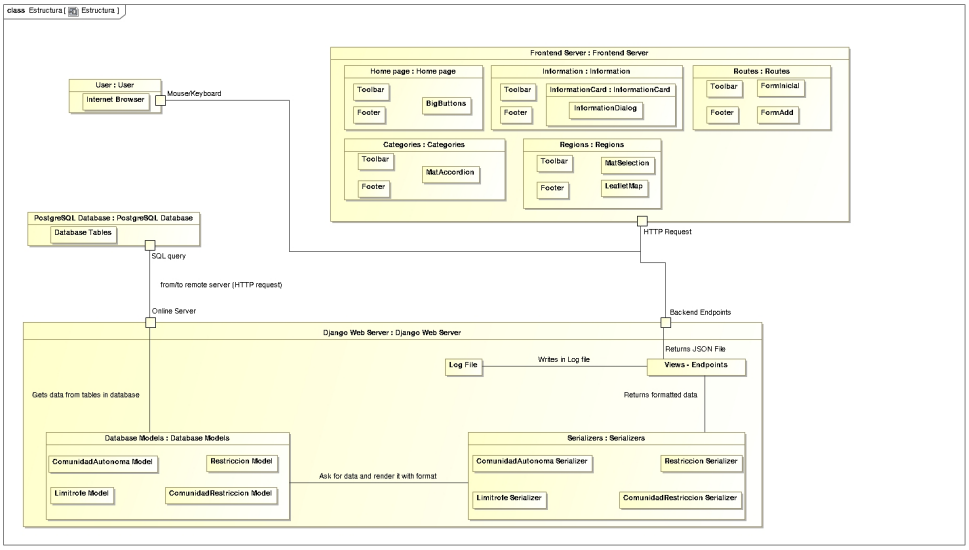
(b) Ventana “Categorias” con la categoría “Movilidad” abierta

Figura 22: Sección “Categorias” final

2.7. Sprint 5 - Desarrollo del *frontend*: Sección Regiones

En este *sprint* se desarrollará la última sección del *frontend* llamada "Regiones". Esta tendrá la funcionalidad de poder mostrar las restricciones de una comunidad autónoma seleccionada y mostrar un mapa de las comunidades autónomas para que si un usuario no es conocedor de estas pueda consultarlo.

Figura 23: Diagrama de Estructura - Cambios para el *frontend*



En la Figura 23 puede verse que los componentes utilizados serán "MatSelection" y "LeafletMap". El primero es un componente de un menú desplegable de selección de opción de Angular Material por lo que solo será necesario proporcionarle las opciones que se cargarán de base de datos. El segundo es un componente mapa se creará y en su interior se añadirá un elemento mapa de la tecnología Leaflet, que da la posibilidad de proporcionar mapas interactivos en entornos web.

Para mostrar en el mapa las comunidades autónomas será necesario obtener un *shapefile*[3] [11] con los contornos. Posteriormente este fichero es necesario cambiar su formato a GeoJson[10]. Una vez hecho esto, Leaflet nos permite proporcionar este archivo al mapa y lo añadirá automáticamente. Ahora solo es necesario pasarle al mapa una función privada que muestre el nombre de cada comunidad autónoma. En la Figura 24 puede verse cómo queda la sección al finalizar el *sprint*.

[3]Shapefile obtenido de https://rpubs.com/albtorval/595824

(a) Ventana “Regiones” sin comunidad elegida

(b) Ventana “Regiones” tras seleccionar comunidad

Figura 24: Sección “Regiones” final

2.8. Sprint 6 - Despliegue de la aplicación en Heroku

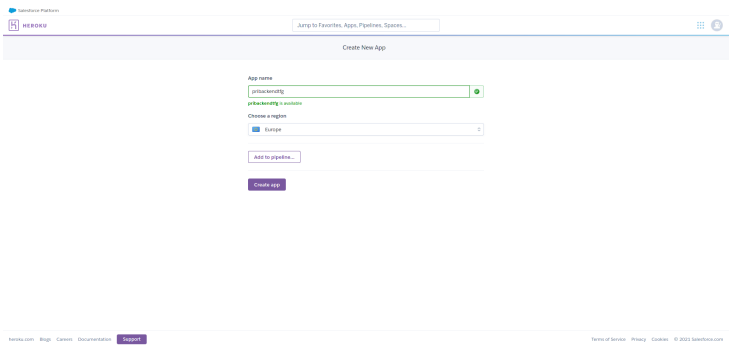
En esta sección se desarrollarán los siguientes requisitos:

- Tanto el *backend* como el *frontend* se alojarán en Heroku.
- La base de datos se alojará junto al *backend*.

2.8.1. *Backend*

Para poder subir el *backend* a Heroku, primero debemos crear crear una nueva app en una dentro de esta plataforma.

Figura 25: Creación de una app en Heroku



Para el siguiente paso hay disponibles dos opciones:

- Añadir un repositorio de GitHub[3]
- Inicializar un repositorio local Git y subir los ficheros a Heroku.

Para este Trabajo de Fin de Grado se utilizará la segunda opción. A continuación es necesario disponer de los ficheros del *backend* en otra carpeta (preferiblemente) y añadir los siguientes ficheros:

- **Procfile:** En este fichero se indica a Heroku que se tratará de un proyecto web, que debe utilizar el servidor de aplicaciones Gunicorn y ejecutar el proyecto según se indica en el fichero wsgi.py. Esto se ha hecho de la siguiente forma:

```
web: gunicorn prib.wsgi:application
```

- **runtime.txt:** En este fichero se indica a Heroku que versión de Python utilizar. Para este Trabajo de Fin de Grado se ha utilizado la versión 3.9.6.

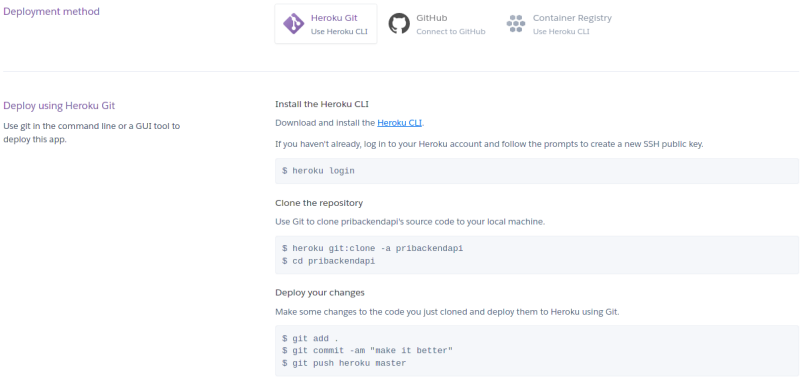
- **requirements.txt:** En este fichero se indica a Heroku las dependencias del proyecto. Se han indicado las siguientes dependencias:

```
asgiref==3.4.1
dj-database-url==0.5.0
Django==3.2.5
django-cors-headers==3.7.0
djangorestframework==3.12.4
gunicorn==20.1.0
psycopg2==2.9.1
pytz==2021.1
sqlparse==0.4.1
whitenoise==5.3.0
```

El siguiente paso es modificar el fichero settings.py del proyecto de Django y añadir a "ALLOWED_HOSTS" lo siguiente:

```
.herokuapp.com
```

Figura 26: Instrucciones de subida proporcionadas por Heroku



Para acabar, solo será necesario abrir un terminal de la computadora y seguir los pasos que se indican en la pestaña "Deploy" de la app de Heroku creada previamente. En la Figura 26 pueden verse los pasos proporcionados por la plataforma.

En cuanto en la terminal se indique que ha terminado la subida, el *backend* estaría listo para su uso en la dirección (por defecto): https://pribackendapi.herokuapp.com/

2.8.2. *Frontend*

Para poder subir el *frontend* a Heroku se seguirán los mismos pasos que para el *backend* con la diferencia que solo se añadirá el fichero Procfile con el siguiente contenido:

```
web: node server.js
```

En el que se indica que se ejecutará un proyecto web, utilizará el servidor de aplicaciones NodeJs y ejecutará el fichero server.js. Este último fichero es un *script* de JavaScript en el que se indica que se devolverá el fichero *index.html* del proyecto de Angular y se usará el puerto 5000 (por defecto para Heroku).

Para acabar, será necesario abrir un terminal de la computadora y seguir los mismos pasos indicados por Heroku que usamos para el *backend* solo que indicando el nombre de la app para el *frontend*. En cuanto termine todo el *frontend* estaría listo para su uso en la dirección (por defecto): https://privfrontend.herokuapp.com/

3

Conclusiones y Líneas Futuras

3.1. Conclusiones

La idea de este Trabajo de Fin de Grado es poder proporcionar herramientas que sirvan de información y prevención en caso de una pandemia o una enfermedad en un país porque con el COVID-19 y el inicio de la pandemia de 2020 no era posible encontrar ninguna herramienta que pudiera ayudar a combatir el virus y hasta pasados varios meses no salieron las primeras aplicaciones; pero mientras tanto la población mundial debía informarse de medidas de prevención, restricciones, información, etc. por los medios tradicionales. Por ello y para evitar esto en el futuro, es necesario que existan aplicaciones, sobre todo de código abierto, que permitan informar a la población no solo de un país sino que esa aplicación pueda prepararse y ajustarse en el menor tiempo posible y también puedan servir para múltiples países. Es por eso que se ha elegido como tecnologías principales Django y Angular, pues son tecnologías con mucha documentación y que permiten un desarrollo rápido, reutilizable y mantenible.

La decisión de que el *backend* sea con estructura API REST es para que, al ser código abierto, cualquiera pueda utilizar la API y obtener datos para sí mismo o incluso para ser utilizado con otros proyectos web *frontend* y, así con la colaboración y el código abierto, pueda darse información a la mayor cantidad de personas posible. Por otro lado, comentar que dichos proyectos web *frontend* servirían para que el usuario promedio pueda visualizar estos datos sin necesidad de hacer consultas a la API y proporcionarle diversas herramientas de consulta y filtrado de datos.

Por tanto, con este Trabajo de Fin de Grado se ha podido desarrollar lo que sería la primera versión de estas herramientas que pueden seguir siendo mantenidas en el futuro para mejorar su uso y reutilización en caso de una emergencia sanitaria. Además del desarrollo, este Trabajo de Fin de Grado también ha tenido un uso formativo en varios puntos pues ha servido para: poder ver la importancia de la Ingeniería de Software y sus procedimientos en el desarrollo de software ya que permite una producción estructurada y documentada; aprender nuevas tecnologías de uso actual y con importancia como Python, Angular, Django, SASS, PostgreSQL, etc. También para poder mostrar que el desarrollo web es un entorno muy amplio con muchas posibilidades y que también requiere de precisión y tiempo de desarrollo.

3.1.1. Problemas durante el desarrollo del Trabajo de Fin de Grado

Al igual que en cualquier proyecto de software, este Trabajo de Fin de Grado se ha visto afectado por diversos problemas que han afectado a su desarrollo o incluso al resultado final. A continuación se detallan dichos problemas encontrados y cómo han sido abordados:

- **Obtención de datos de restricciones e información sanitaria durante el COVID-19:**

  Para este Trabajo de Fin de Grado se plantea inicialmente el uso de la aplicación para mostrar información durante el COVID-19 en España y obtener la información de las páginas de datos abiertos del Ministerio de Salud de España o incluso del portal de datos.gob.es. No obstante durante el desarrollo comienzan las vacunaciones en España y en varios países por lo que estas fuentes de datos abiertos dejan de aparecer por lo que hace que no sea posible la obtención de datos directamente ni recibir cambios (como por ejemplo en restricciones) por el estado de la pandemia. Esto ha hecho que cambie la idea inicial ya que la solución que se le ha dado a esto es que los datos de la API puedan ser añadidos directamente desde una ventana de administración en el *backend* y al realizar una consulta se obtengan directamente de la base de datos. Esto permite que se añadan datos manualmente de diversas fuentes y puedan ser editados o borrados; además si se necesita en un futuro la utilización de datos abiertos, solo sería necesario la lectura de los datos e inserción en la base de datos utilizando las posibilidades que ofrecen Django y Django REST Framework, por lo que se conseguiría también que los datos manuales y los recibidos se muestren en el *frontend* de forma homogénea.

- **Adaptabilidad del *frontend* con respecto a los distintos tamaños de pantalla y ventana:**

  En la actualidad hay una gran cantidad de dispositivos con acceso a internet y con navegadores web como pueden ser ordenadores de mesa, portátiles, móviles, etc. Y esto hace que haya una variedad muy grande de tamaños de pantalla. Es por esto que ahora las aplicaciones webs deben de ser adaptivas a todos estos tamaños de pantalla o incluso a una reducción de la ventana del navegador. Para que una web sea completamente (o mínimamente) adaptable es necesario añadir a las aplicaciones webs multitud de elementos y funciones que permitan diversos tamaños de ventana y usualmente esto suele ser una tarea que puede llevar bastante más tiempo del esperado. Para el caso de este Trabajo de Fin de Grado, se ha conseguido que haya cierta Adaptabilidad en el *frontend* pero quizá pueda no ser la deseada en tamaños de pantalla muy pequeños o en dispositivos móviles. La solución que se ha dado a esto es utilizar la librería de código abierto llamada Flex-layout. Con esta librería es posible conseguir una gran Adaptabilidad ya que hace posible los cambios de tamaño en los componentes en el momento. Esta herramienta se encuentra dentro del proyecto, no obstante, utilizar las funciones que permiten una prácticamente completa Adaptabilidad en la aplicación web requieren bastante tiempo de desarrollo en incorporarlas y que para este Trabajo de Fin de Grado conllevarían demasiado uso de horas. Es por ello que se plantea añadir estas funciones en Líneas Futuras.

- **Eventos con el mapa de Leaflet y Angular:**

  En el planteamiento de la idea se quería que en la sección "Regiones" al pulsar en una comunidad autónoma en el mapa directamente cargara las restricciones. Pero al añadir la función de carga de restricciones al mapa de la herramienta Leaflet no era capaz de ejecutarla ya que se esta se ejecuta solo una vez durante la creación del mapa y este automáticamente mostrará lo pedido en el mapa, pero nunca podría ejecutar las funciones desarrolladas puesto que este objeto de Leaflet no es capaz de acceder al componente de la sección. Por ello, el siguiente intento para solucionar este problema fue generar un evento al pulsar en una comunidad autónoma en el mapa, pero estos eventos son los generados por Leaflet, así que desde Angular no es posible recogerlos ya que este framework utiliza otros métodos de recogida de eventos. También se probó con eventos de

JavaScript pero estos tampoco pudieron ser recogidos correctamente y ejecutar la función que muestra las restricciones. Tras investigar en diferentes páginas webs, artículos, documentación, etc. no se encontró nada sobre esto por lo que finalmente se decide solucionar esto dejando el mapa de Leaflet en el mismo lugar pero mostrando el nombre de la comunidad autónoma seleccionada, ya que aunque no cargue las restricciones puede ser necesario cómo consulta para los usuarios que no pertenezcan al país y/o no tengan conocimiento sobre las comunidades autónomas.

3.2. Líneas Futuras

Como líneas futuras de este Trabajo de Fin de Grado que permitan mejorar las herramientas producidas y crear nuevas versiones se propone lo siguiente:

- **Internacionalización en las herramientas para mostrar varios idiomas:**
  Es importante que tanto en el *backend* como en el *frontend* haya soporte de múltiples idiomas ya que no todos los usuarios de las herramientas pueden saber español, además de que estas pueden ser utilizadas en otros países.

- **Adaptabilidad más completa para pantallas más pequeñas y/o dispositivos móviles:**
  Tal y cómo se ha comentado en el apartado anterior de Conclusiones, es necesario que las aplicaciones web sean adaptivas al tamaño para poderse mostrar en la mayor cantidad de dispositivos posible. Gracias a que en el proyecto se ha incluido y hecho uso de la librería FlexLayout para Angular, solo sería necesario incluir a los distintos elementos del *frontend* funciones que recojan el tamaño de pantalla y devuelvan el tamaño de los componentes y también permitan reposicionarlos en la pantalla; ya que el tiempo de desarrollo que esto conlleva es elevado.

- **Componentes que permitan la generación de secciones más rápidamente:**
  Una línea futura que puede ayudar al desarrollo de estas aplicaciones presentadas es la creación de diversos componentes que reciban como argumento descripciones, títulos, configuraciones, etc. y permitan generar una sección en el *frontend* con mayor rapidez y facilidad aún ya que es importante disponer de estas secciones cuanto antes.

Referencias

[1] Angular. Introduction to services and dependency injection. URL https://angular.io/guide/architecture-services.

[2] A. Beaulieu. Learning sql. 2009.

[3] K. Brown. What is github, and what is it used for? URL https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/.

[4] B. Butler. Paas primer: What is a platform as a service and why does it matter, 2013. URL https://www.networkworld.com/article/2163430/paas-primer--what-is-platform-as-a-service-and-why-does-it-matter-.html.

[5] S. Chacon. Git. URL https://git-scm.com/.

[6] D. Flanagan. Javascript - the definitive guide. 2006.

[7] M. Fowler. Gui architectures. URL https://martinfowler.com/eaaDev/uiArchs.html.

[8] freeCodeCamp. https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/. URL https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/.

[9] I. J. Grady Booch, James Rumbaugh. Unified modeling language user guide. 2005.

[10] I. E. T. F. (IETF). The geojson format. URL https://datatracker.ietf.org/doc/html/rfc7946.

[11] E. S. R. I. Inc. Esri shapefile technical description. 1998.

[12] E. International. Standard ecma-404 - the json data interchange syntax. 2017.

[13] G. Larman. *Agile and Iterative Development: A Manager's Guide*. 2004.

[14] Microsoft. Typescript: Javascript with syntax for types.

[15] M. D. Network. Learn to style html using css, . URL https://developer.mozilla.org/en-US/docs/Learn/CSS.

[16] M. D. Network. Javascript, . URL https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[17] Oxagile. Waterfall software development model, 2014. URL https://www.oxagile.com/article/the-waterfall-model/.

[18] O. Pleten. What is a mockup and why do we need it, 2019. URL https://keenethics.com/blog/1521631041972-the-importance-of-mockups.

[19] RedHat. What is a rest api?, 2020. URL https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[20] M. Rehkopf. Sprints. URL https://www.atlassian.com/agile/scrum/sprints.

[21] J. Volchko. Prototyping methodology: Steps on how to use it correctly, 2017. URL https://www.lumitex.com/blog/prototyping-methodology.

[22] W3Schools. Sass introduction. URL https://www.w3schools.com/sass/sass_intro.asp.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA