UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

DESARROLLO DE APLICACIÓN WEB PARA PROYECTO BEETOOL

Trabajo Fin de Grado

Grado en Tecnología Digital y Multimedia

AUTOR/A: López Conejero, Jose

Tutor/a: Ballester Merelo, Francisco José

Cotutor/a: Ramos Peinado, Germán

Cotutor/a externo: PONS PUIG, AURELIO-JOSE

CURSO ACADÉMICO: 2022/2023

Resumen

Este trabajo final de grado aborda el desarrollo de una aplicación web completa, su implementación dentro de un contenedor, y su posterior integración dentro del proyecto BEETool (Blade Erosion Evaluation Tool) del Grupo TecEner de la universidad CEU (Centro de Estudios Universitarios). El proyecto aborda el desarrollo de una herramienta computacional con la que se puede evaluar y predecir la forma en la que se comportan distintos materiales utilizados en la construcción de las palas de las turbinas eólicas con el objetivo de mejorar la eficiencia en la producción de energía y minimizar los costes de mantenimientos.

Se trata de un TFG colaborativo en el que cada uno de los dos miembros se ha encargado de algunas de sus partes: aplicación web y contenerización e integración.

Este trabajo se centra en el desarrollo de la aplicación web desde la que poder introducir todos los datos, lanzar las simulaciones y visualizar resultados para poder actuar como panel de control. El trabajo aborda una primera etapa de análisis de requisitos, necesidades de conexiones externas hacia la herramienta de simulación y acceso a las bases de datos. Posteriormente se trata el propio desarrollo de la aplicación empleando lenguajes como PHP, HTML, CSS y JavaScript para resolver cada una de las partes, identificando qué resultados mostrar al usuario final y la forma más adecuada.

La herramienta creada permite introducir al usuario todos los datos necesarios, lanzar las simulaciones, y mostrar en texto y de forma gráfica los resultados de las mismas en un informe o en la propia web.

El resultado de la aplicación web junto a la posterior contenerización e integración y la aplicación móvil, da al proyecto BEETool una herramienta global con la que poder predecir los resultados de eficiencia y mantenimiento de palas de turbinas eólicas. Los resultados finales son totalmente operativos y están siendo ya ofertados a empresas del sector energético.

Resum

Aquest treball final de grau aborda el desenvolupament d'una aplicació web completa, la seua implementació dins d'un contenidor, i la seua posterior integració dins del projecte BEETool (Blade Erosion Evaluation Tool) del Grup TecEner de la universitat CEU (Centre d'Estudis Universitaris). El projecte aborda el desenvolupament d'una eina computacional amb la qual es pot avaluar i predir la forma en la qual es comporten diferents materials utilitzats en la construcció de les pales de les turbines eòliques amb l'objectiu de millorar l'eficiència en la producció d'energia i minimitzar els costos de manteniments.

Es tracta d'un TFG collaboratiu en el qual cadascun dels dos membres s'ha encarregat d'algunes de les seues parts: aplicació web i contenerització i integració.

Aquest treball se centra en el desenvolupament de l'aplicació web des de la qual poder introduir totes les dades, llançar les simulacions i visualitzar resultats per a poder actuar com a panell de control. El treball aborda una primera etapa d'anàlisi de requisits, necessitats de connexions externes cap a l'eina de simulació i accés a les bases de dades. Posteriorment es tracta el propi desenvolupament de l'aplicació emprant llenguatges com PHP, HTML, CSS i JavaScript per a resoldre cadascuna de les parts, identificant quins resultats mostrar a l'usuari final i la forma més adequada.

L'eina creada permet introduir a l'usuari totes les dades necessàries, llançar les simulacions, i mostrar en text i de manera gràfica els resultats de les mateixes en un informe o en la pròpia web.

El resultat de l'aplicació web al costat de la posterior contenerització i integració i l'aplicació mòbil, dona al projecte BEETool una eina global amb la qual poder predir els resultats d'eficiència i manteniment de pales de turbines eòliques. Els resultats finals són totalment operatius i estan sent ja oferits a empreses del sector energètic.

Abstract

This final degree project addresses the development of a complete web application, its implementation within a container, and its subsequent integration in BEETool (Blade Erosion Evaluation Tool) project of the TecEner Group of CEU University (Center for University Studies). BEETool addresses the development of a computational tool which allows to evaluate the way different materials used in the construction of wind turbine blades behave with the aim of improving energy production efficiency and minimizing maintenance costs.

It is a collaborative TFG in which each of the two members has been responsible for some of its parts: web application and containerization and integration.

This work focuses on the development of the web application which makes it possible to enter all the data, run the simulations and visualize results to be able to act as a control panel. The work addresses a first stage of requirements analysis, needs for external connections to the simulation tool and access to databases. Subsequently, the development of the application itself is addressed using languages such as PHP, HTML, CSS and JavaScript to solve each of the parts, identifying what results should be shown to the end user and the most appropriate way.

The tool created enable the user to enter all the necessary data, run the simulations, and display in text and graphically the results of them in a report or on the website itself.

The result of the web application combined with the subsequent containerization and integration and the mobile application, gives the BEETool project a global tool in which the efficiency and maintenance results of wind turbine blades can be predicted. The results are fully operational and them are already being offered to companies in the energy sector.

A mis padres y hermanos.

A Jose Antonio, mi compañero de TFG.

A mi tutor, Germán, por su paciencia y seguimiento constante y a Aurelio, por la oportunidad.

Índice general

I Memoria

Listado de siglas empleadas

API Application Programming Interfaces.

AVI Agencia Valenciana de la Innovación.

BEETool Blade Erosion Evaluation Tool.

FEDER Fondo Europeo de Desarrollo Regional.

ORM Object Relational Mapping.

PHP Hypertext Preprocessor.

TecEner Grupo de Investigación y Desarrollo de Tecnologías en Aplicaciones Energéticas.

TFG Trabajo Final de Grado.

UCH CEU Centro de Estudios Universitarios Cardenal Herrera.

Parte I

Memoria

Capítulo 1

Introducción y Objetivos

1.1. Introducción

Para entender los objetivos de este trabajo, primero es necesario comprender el proyecto BEETool, cómo surge, con qué finalidad, qué problemas intenta solucionar, cómo los resuelve, etc.

En primer lugar y respondiendo a las preguntes anteriores, el nombre de este proyecto proviene de Blade Erosion Evaluation Tool [1], es decir, a grandes rasgos, una herramienta que es capaz de evaluar la erosión en las palas de las turbinas eólicas. Cuenta con la colaboración de la Agencia Valenciana de la Innovación (AVI) de la Generalitat Valenciana, y está cofinanciado por la Unión Europea a través del Programa Operativo del Fondo Europeo de Desarrollo Regional (FEDER) de la Comunitat Valenciana 2014-2020.

Está dirigido por Fernando Sánchez López, dentro de un grupo de investigadores llamado TecEner (Grupo de Investigación y Desarrollo de Tecnologías en Aplicaciones Energéticas) del Centro de Estudios Universitarios Cardenal Herrera (UCH CEU). Este grupo centra sus investigaciones en dos grandes ámbitos: por un lado en la optimización energética de tecnologías de hidrógeno y solar en aplicaciones de movilidad y edificación, habiendo participando en varios proyectos relacionados; y por otro lado, en el Desarrollo de tecnologías de nuevos materiales para la energía eólica en aplicaciones de palas de aerogeneradores, donde se enmarca el proyecto BEETool que se ha mencionado anterior-

mente.

En este sentido, cabe destacar el grave problema que supone la erosión y el desgaste de los materiales en los aerogeneradores sometidos a condiciones climáticas desfavorables, como puede ser lo más alto de una montaña o incluso alta mar. El principal agente de esta erosión y desgaste son las gotas de lluvia que impactan en la superficie a grandes velocidades causando perdida de eficiencia en las palas. Además, el hecho de cambiar una pala cuando está dañada se vuelve un proceso muy complejo, largo y costoso debido a la altura de los generadores o el tamaño de las propias palas, que pueden llegar a medir ochenta y cinco metros de largo. A esto se debe sumar el reto de reciclar las palas cuando terminen su vida útil, ya que al estar compuestas de diversos materiales, y tan complejos, se vuelve muy complicado intentar no contaminar el medio ambiente.

Para poder adelantarse a estos problemas, existe una única máquina en Europa , que es capaz de simular el desgaste y las condiciones al las que se someten los materiales de las palas de los aerogeneradores y es aquí donde el Proyecto BEETool cobra importancia.

El profesor e investigador Fernando Sanchez, junto con su grupo, ha desarrollado una herramienta computacional que es capaz de simular las condiciones climáticas que se han mencionado y de esta forma evaluar el rendimiento de diferentes materiales sometidos a ellas. Con ello, se ataca el problema en la etapa más temprana del desarrollo de los aerogeneradores, mejorando así, de manera muy significativa, la eficiencia de la producción de energía eólica.

Para complementar esta herramienta computacional que han diseñado, necesitaban el desarrollo de una aplicación web, a modo de interfaz de usuario, en la que poder introducir diferentes datos y generar informes con la evaluación de las simulaciones. Esta última parte es lo que aborda este Trabajo Final de Grado (TFG). En la figura 1.1 se explica de manera muy general como funciona el proyecto.

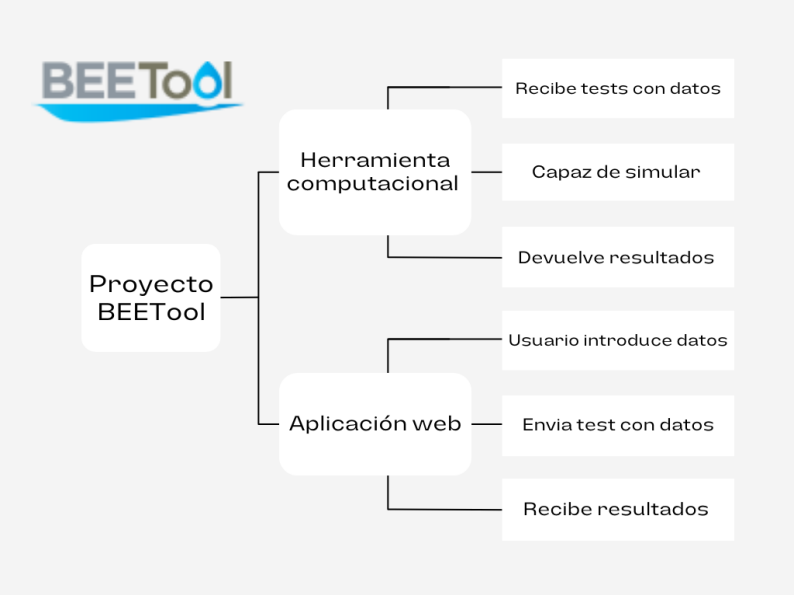


Figura 1.1: Esquema general del funcionamiento del proyecto BEETool

Se trata de un TFG colaborativo, es decir, en él han trabajado de forma conjunta dos personas, diferenciando de forma clara dos partes en el desarrollo del proyecto: Desarrollo de aplicación web para proyecto BEETool, contenido en esta memoria y por otro lado Contenerización, integración de sistemas y migración de aplicación web con aplicación móvil complementaria. Esta parte está realizada por Jose Antonio Téllez Mérida.

Se ha desarrollado en su totalidad durante la estancia en un periodo de prácticas de empresa en A3P Interacción Digital S.L. y en colaboración y revisión continua por parte de los miembros del proyecto BEETool.

En la siguiente figura, 1.2, se explica a grandes rasgos como funcionaría la aplicación web, y sus componentes básicos, que se explicarán en detalle en esta memoria.

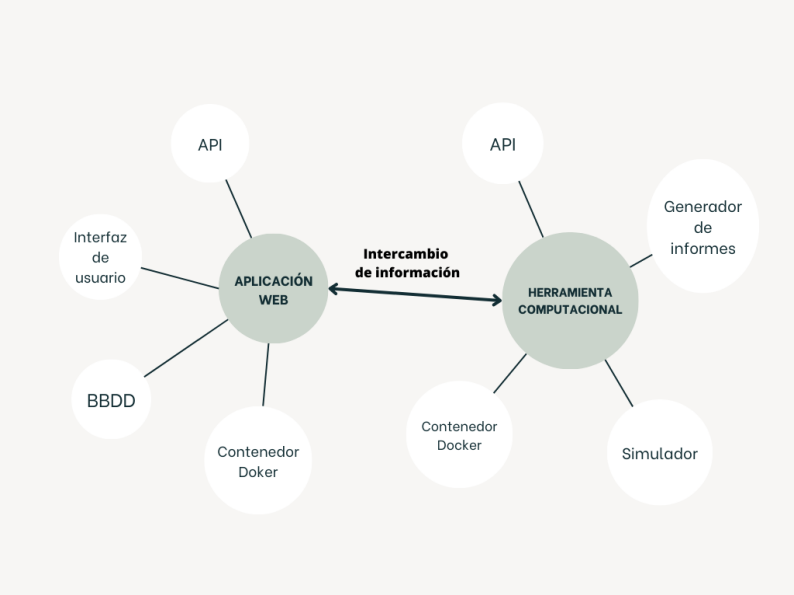


Figura 1.2: Esquema general del funcionamiento de la aplicación web junto con la herramienta computacional

1.2. Objetivos

En cuanto a objetivos del trabajo podemos hablar de desarrollar un panel de control, en el que, siguiendo las fases determinadas de una simulación que se detallarán posteriormente, poder introducir los datos necesarios en cada una de estas fases, guardarlos en una base de datos y poder acceder a ellos fácilmente. Uno de los retos del panel será que admita cualquier tipo de dato multimedia y sea capaz de procesarlo correctamente, ya sean números (decimales, negativos, etc.), textos o imágenes.

Recopilando todos los datos que el usuario introduzca, ya sea manualmente o mediante cualquier tipo de ficheros, el panel deberá de ser capaz de enviarlos en formato ’JSON’[2] a la herramienta computacional de la que se hablaba anteriormente. Una vez realizado este proceso de comunicación, el panel mostrará un informe detallado con los resultados de la simulación. Este informe y su contenido dependerá de lo que el proyecto BEETool requiera por lo que, como objetivo de este trabajo se presentará un informe de ejemplo.

Además, cabe mencionar que se necesitaba una herramienta lo más funcional y sencilla posible, descrita en la figura 1.3 , por lo que no era necesario centrar demasiados esfuerzos

en procesos como el diseño, si no más bien en esta funcionalidad mencionada.

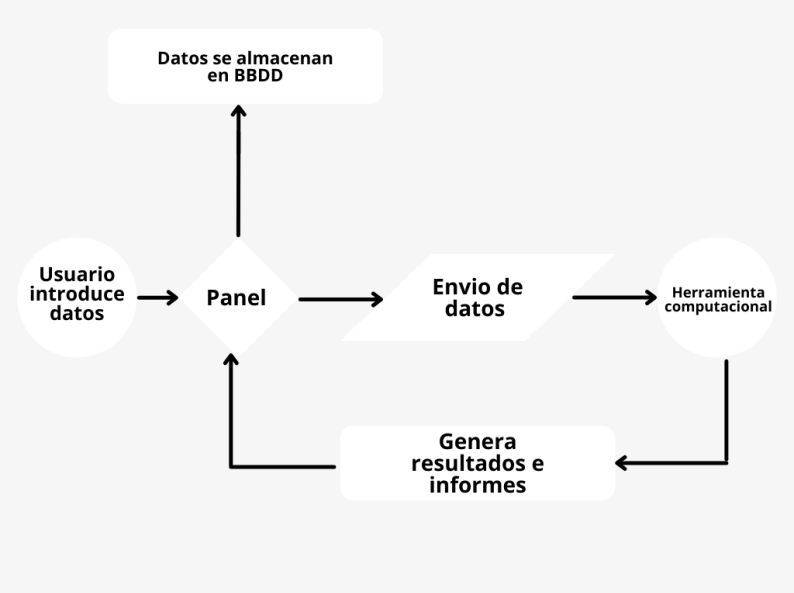


Figura 1.3: Esquema general del funcionamiento de la aplicación web

Capítulo 2

Metodología

En cuanto a la metodología de trabajo se ha seguido la metodología ’Pair Programming’ [3] o trabajo en parejas. Es una práctica de desarrollo de software en la que dos programadores trabajan juntos en una estación de trabajo compartida para completar una tarea o resolver un problema. En esta metodología, uno de los programadores es el ’conductor’ y escribe el código activamente, mientras que el otro es el ’observador’ y proporciona comentarios, revisa el código y ayuda en la resolución de problemas. De esta forma, ambos desarrolladores han participado en el desarrollo de las dos partes del proyecto y conocen su funcionamiento.

El Pair Programming se basa en la colaboración estrecha y constante entre los dos programadores para mejorar la calidad del código, fomentar el intercambio de conocimientos, reducir los errores y mejorar la productividad. Ambos programadores han trabajado en conjunto, discutiendo y tomando decisiones conjuntas en tiempo real. Esto permite una revisión continua del código y facilita la detección temprana de errores o malas prácticas.

En cuanto a la planificación de objetivos, se ha utilizado la herramienta Trello. Trello es una herramienta de gestión de proyectos basada en tableros en línea. Proporciona una interfaz visualmente intuitiva que permite organizar y dar seguimiento a las tareas de un proyecto de manera eficiente. Trello se basa en el concepto de tableros, listas y tarjetas, que representan diferentes etapas o elementos del proyecto. Para este proyecto se creó un tablero en el que, como se puede ver en la figura 2.1 se han creado diferentes tarjetas con

los objetivos o con las diferentes tareas a realizar. En la figura aparecen en la columna 'Hecho', pero han ido pasando por las diferentes columnas según en la fase de desarrollo en la que se encontraban.

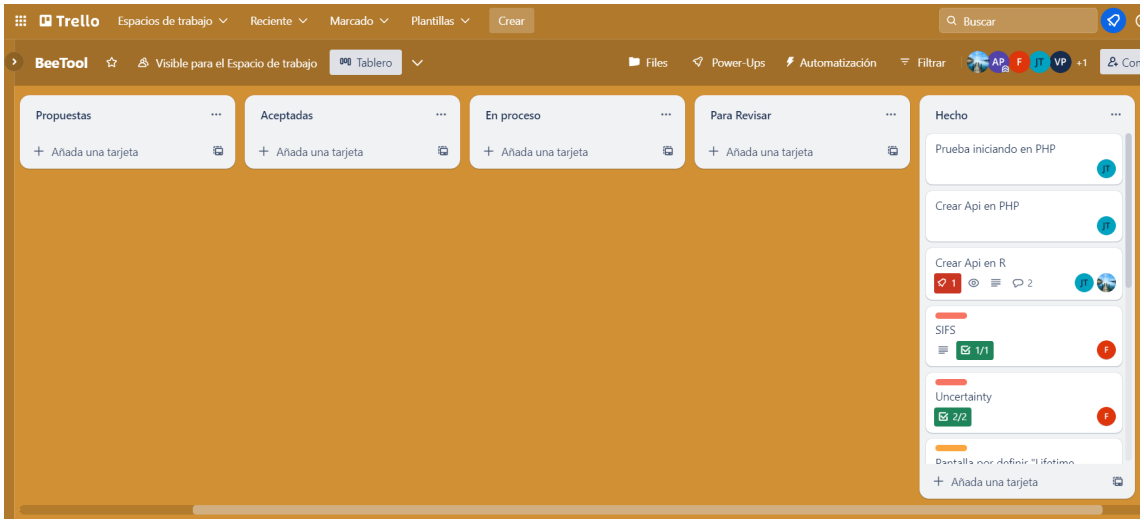


Figura 2.1: Captura de tablero BEETool en Trello

Además de esto, se realizaban reuniones semanales con el tutor de las prácticas de empresa y con los miembros del proyecto BEETool en las que se revisaba el trabajo realizado y se proponían nuevos objetivos a cumplir.

En cuanto al desarrollo de la aplicación web, se ha trabajado de forma local, realizando actualizaciones constantes en un repositorio de GitHub [4]. De esta forma, ambos miembros del equipo tenían en su ordenador el proyecto con la última versión.

Finalmente, y descrito en detalle en la otra parte del TFG colaborativo se realizo la subida del proyecto a un servidor real, pudiendo acceder desde la ruta [5] 'https://panel.beetool.es/admin'

Capítulo 3

Selección de Tecnologías

En este cápitulo se van a explicar los diferentes lenguajes de programación, frameworks usados en el proyecto, así como los diferentes softwares que han permitido el correcto funcionamiento del mismo.

3.1. PHP

PHP[6], acrónimo de ”Hypertext Preprocessor”, es un lenguaje de programación ampliamente utilizado en el desarrollo de aplicaciones web dinámicas. Fue creado originalmente por Rasmus Lerdorf en 1994 y desde entonces ha experimentado un crecimiento significativo en popularidad.

PHP es un lenguaje de código abierto, lo que significa que su código fuente es accesible y puede ser modificado y distribuido libremente. Esto lo convierte en una opción atractiva para los desarrolladores, ya que no requiere la adquisición de una licencia o el pago de derechos de autor para utilizarlo.

Una de las principales características de PHP es su capacidad para integrarse fácilmente con HTML, por lo que, como se va a explicar posteriormente, en la parte de front-end del panel web se ha usado este lenguaje, entre otros. PHP se ejecuta en el lado del servidor, lo que significa que el código PHP se procesa en el servidor antes de que se envíe el resultado final al navegador web del usuario.

PHP ofrece una amplia gama de funcionalidades y características que lo hacen una de las mejores opciones para este trabajo, como la manipulación de formularios web, el acceso a bases de datos, la generación de contenido dinámico, la gestión de sesiones y cookies, entre otras. También es compatible con una variedad de bases de datos populares, como MySQL, la cual usaremos a través de la herramienta phpMyAdmin.

Además, PHP cuenta con una extensa comunidad de desarrolladores que contribuyen con bibliotecas, frameworks, que se explicaran posteriormente y recursos adicionales que facilitan el desarrollo de aplicaciones web de forma más eficiente.

En resumen, PHP es un lenguaje de programación de código abierto utilizado para el desarrollo de aplicaciones web dinámicas. Su naturaleza de código abierto lo hace accesible para su uso sin restricciones de derechos de autor por lo que es una opción bastante acertada para el desarrollo de la aplicación web en el proyecto BEETool.

3.2. Slim

Habiendo escogido el lenguaje principal del desarrollo, es necesario el uso de un framework con el que trabajar:

Slim Framework[7] es un microframework de código abierto para el desarrollo de aplicaciones web en PHP. Está diseñado para ser ligero y eficiente y se enfoca en proporcionar las funcionalidades básicas para construir aplicaciones web sin agregar una sobrecarga innecesaria. Esto hace que se vuelva la mejor opción, ya que como se ha explicado en el apartado anterior, dentro de los objetivos de la aplicación, se encuentra un resultado sencillo a la vez que funcional.

Una característica destacada de Slim Framework es su enfoque en la simplicidad y la elegancia. A diferencia de otros frameworks más completos, Slim se centra en proveer solo las herramientas esenciales para el enrutamiento de URLs, la gestión de solicitudes y respuestas HTTP, y la manipulación de datos.

Además, Slim es altamente extensible gracias a su arquitectura modular. Es posible, como se explicará posteriormente, agregar fácilmente nuevos componentes o utilizar componen-

tes de terceros para ampliar las capacidades de Slim según las necesidades específicas de cada proyecto.

A pesar de ser un microframework, Slim ofrece una serie de características útiles, como el enrutamiento flexible y dinámico, la inyección de dependencias, el manejo de errores y excepciones, la creación de APIs RESTful y la renderización de plantillas. Un API REST permite que dos sistemas se comuniquen entre sí a través de Internet de manera eficiente y confiable.

Estas características hacen de Slim una buena opción para el desarrollo rápido y eficiente de aplicaciones web como la necesaria en el proyecto.

3.3. RedBeanPHP

Gracias a esta capacidad de integrar nuevos componentes con Slim, cabe destacar el uso de RedBeanPHP en el proyecto:

RedBeanPHP[8] es una biblioteca ORM (Object Relational Mapping) de código abierto para PHP. Su objetivo principal es simplificar y agilizar el proceso de almacenamiento y recuperación de objetos en una base de datos, eliminando la necesidad de escribir consultas SQL manualmente.

Una de las principales características de RedBeanPHP es su capacidad para gestionar automáticamente la estructura de la base de datos. A través de técnicas de ”mapeo de objetos”, RedBeanPHP crea y ajusta automáticamente las tablas y columnas necesarias en la base de datos según la estructura de los objetos utilizados en la aplicación. Esto evita la necesidad de escribir y mantener consultas de creación y alteración de tablas manualmente. Debido a esto, se va a convertir en la herramienta que nos va a dar acceso desde el panel, a la base de datos, cuando sea necesario en el desarrollo del mismo. Permitirá realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) y manipular objetos directamente, sin la necesidad de lidiar con consultas SQL, lo que simplifica enormemente el desarrollo y mantenimiento del código.

Además es compatible con la base de datos que se va a utilizar en el proyecto, MySQL.

3.4. Docker

Por último, en la parte de backend, es importante el uso de la herramienta Docker. No es el objetivo principal del proyecto, pero toda la aplicación web esta alojada en un contenedor Docker, lo que mejora su escalabilidad, su eficacia y su rendimiento entre otras cosas. Toda la parte de contenerización con Docker se aborda en la otra parte del TFG colaborativo.

3.5. HTML, CSS y JavaScript

Por otro lado, en la parte de front-end, gracias a la compatibilidad con PHP Y Slim Framework, se van a usar los lenguajes HTML, CSS y JavaScript además del motor de plantillas Twig.

HTML (HyperText Markup Language), CSS (Cascading Style Sheets) y JavaScript son tres tecnologías fundamentales en el desarrollo web. Cada una de ellas tiene un propósito específico y se utilizan en conjunto para crear sitios web interactivos y visualmente atractivos. HTML se utiliza para definir la estructura y el contenido de una página web, CSS se utiliza para dar estilo y diseño a la página, y JavaScript se utiliza para agregar interactividad y funcionalidad dinámica

3.6. Twig

Twig [9] es un motor de plantillas de código abierto diseñado para PHP. Proporciona una forma eficiente y segura de separar la lógica de presentación de una aplicación web.

El objetivo principal de Twig es mejorar la legibilidad y mantenibilidad del código al permitir una separación clara entre la lógica del backend y la presentación del frontend. En lugar de mezclar código PHP con HTML directamente, Twig utiliza archivos de plantilla que contienen marcadores y expresiones especiales para generar el contenido final.

Utiliza una sintaxis clara y sencilla y además permite algo tan importante como es la utilización y la herencia de plantillas, es decir, la reutilización de código. Como se explicará en la parte de desarrollo, esta última característica ha servido en gran parte para darle forma

y estructura al panel final.

Además Twig está diseñado para ser rápido y eficiente. Utiliza un sistema de caché que almacena en memoria las plantillas compiladas para un rendimiento óptimo en ejecuciones posteriores.

3.7. GitHub

En relación al control de versiones, se ha usado la herramienta Git y GitHub, donde, en un repositorio del proyecto, se ha ido actualizando periódicamente el proceso de desarrollo del mismo.

3.8. Lagan

Lagan[10] es una versión de un CMS (Content Management System), basado en la flexibilidad. Un CMS o Sistema de Gestión de Contenidos es un sistema online que permite poner en marcha una página web de forma sencilla y rápida. Se trata de un software que ayuda a administrar contenidos dinámicos, y funciona precisamente con las tecnologías seleccionadas en este proyecto, por lo que se seleccionó para el desarrollo del mismo.

Funciona a partir de modelos de datos, o en programación, clases predefinidas con diferentes propiedades que pueden ser modificadas según la necesidades. Estos modelos son simplemente combinaciones simples de 'arrays' (tipo de dato que almacena conjuntos de datos homogéneos) que se pueden crear y modificar fácilmente y en combinación con las plantillas de Twig, mostrarse por pantalla de forma sencilla.

En el caso de este proyecto, es necesario un modelo de datos diferente por cada pantalla de la aplicación web. Cada modelo de dato se explica detalladamente más adelante en esta memoria pero, por ejemplo, en el caso de materiales, se necesita una clase con un campo de texto para el nombre del material, quince campos numéricos para las características propias de ese material y por último otro campo de texto por si fuera necesario un comentario acerca de ese material.

Cada apartado de la web necesita sus propios campos, por lo que Lagan, es una buena opción ya que estos apartados (materiales, condiciones climáticas, tests simulados...) se pueden programar como un modelo de lagan cada uno.

Lagan viene con tipos de datos predefinidos, pero también será necesario crear tipos de datos específicos para cubrir las necesidades solicitadas en el proyecto.

3.9. Bootstrap y jQuery

Bootstrap[11] es un framework de código abierto desarrollado por Twitter. Proporciona una colección de herramientas y componentes predefinidos, como botones, barras de navegación, cuadros modales, rejillas de diseño responsivas, entre otros. Bootstrap facilita el diseño y la creación de interfaces de usuario atractivas y responsivas, sin necesidad de escribir mucho código personalizado. También ofrece estilos CSS predefinidos y un conjunto de plugins opcionales para agregar características adicionales, como carruseles, ventanas emergentes y validación de formularios.

jQuery[12] es una biblioteca de JavaScript rápida, compacta y de uso general. Está diseñada para simplificar la manipulación y el manejo de eventos en el lado del cliente. jQuery proporciona una amplia gama de funcionalidades, como la manipulación del DOM (Document Object Model), la animación, la gestión de eventos, las solicitudes AJAX, etc. Con jQuery, es más fácil y rápido realizar tareas comunes en JavaScript, ya que ofrece una sintaxis simplificada y abstracciones que funcionan en múltiples navegadores.

Al incluir Bootstrap y jQuery en el directorio assets del proyecto, se pueden utilizar en las páginas HTML para aprovechar sus funcionalidades y estilos. Esto implica enlazar los archivos CSS y JavaScript correspondientes en el encabezado o pie de página de la página HTML, de manera similar a como se vinculan otros archivos CSS y JavaScript.

3.10. JSON

JSON (JavaScript Object Notation) es un formato de datos ligero y legible por humanos utilizado para el intercambio de datos en aplicaciones web. Se basa en la sintaxis de los objetos JavaScript y se utiliza comúnmente como un formato de datos para la comunicación entre el cliente y el servidor. Es el formato elegido para intercambiar la información entre la aplicación web y la herramienta computacional del proyecto BEETool.

Se caracteriza principalmente porque representa los datos en una estructura de pares clave-valor. Los datos se organizan en objetos y matrices, y cada elemento se define mediante una clave y un valor. Los valores pueden ser de diferentes tipos, como cadenas de texto, números, booleanos, objetos o matrices anidadas. En la siguiente fragmento de código se muestra un ejemplo sencillo:

```
{
  "nombre": "Juan",
  "edad": 30,
  "hobbies": ["correr", "leer", "viajar"],
  "direccion": {
    "calle": "Calle Principal",
    "ciudad": "Ciudad Ejemplo",
    "pais": "Ejemplo"
  }
}
```

Capítulo 4

Desarrollo de la solución

En cuanto al desarrollo de la aplicación web se diferencian dos grandes fases:

4.1. Fase 1: Diseño

Esta fase describe la primera parte del desarrollo, antes de escribir el código, qué se debía de conseguir, qué estructura tenía que tener, etc.

En cuanto a las necesidades que debía cubrir, se propuso desde el proyecto BEETool que tenía que ser un panel funcional y sencillo, que fuera capaz de recoger una serie de datos y características de un test y enviarlas en formato JSON a su herramienta. Después, que fuera capaz de recibir una serie de datos también en formato json a modo de respuesta y mostrar un informe en formato PDF al usuario.

En primer lugar, antes de explicar el resultado final, se va a explicar las primeras ideas que surgieron en el inicio del proyecto para desarrollar la aplicación.

4.1.1. Primeras ideas

Las siguientes figuras muestran los primeros bocetos iniciales con los que se plasmó las primeras ideas sobre el panel:

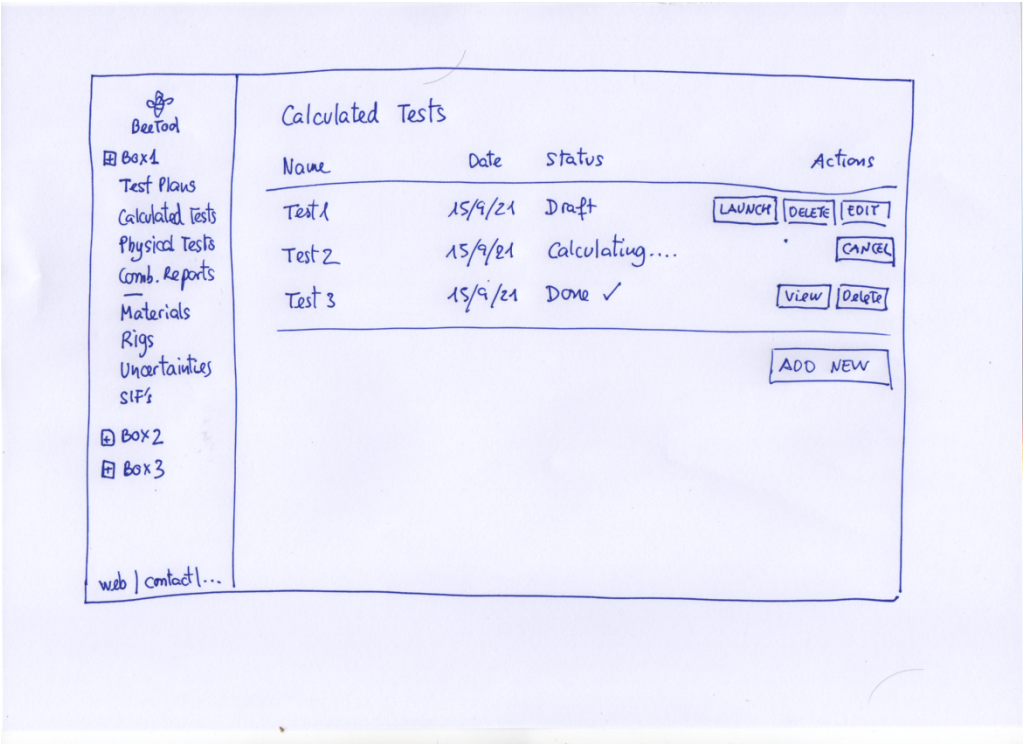


Figura 4.1: Boceto de pantalla Calculated Tests

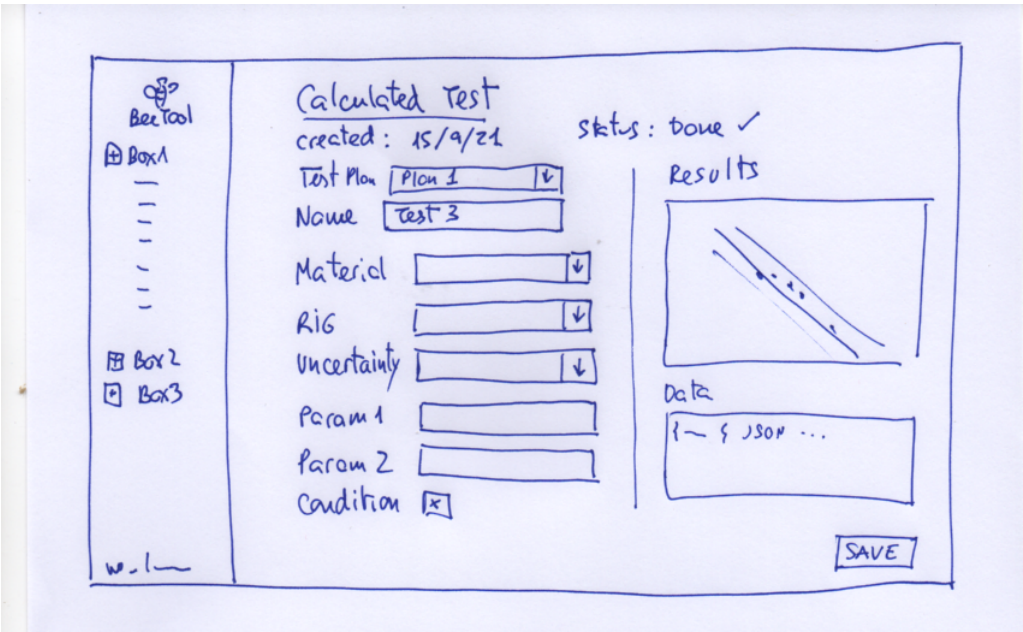


Figura 4.2: Boceto de pantalla para añadir Calculated Test

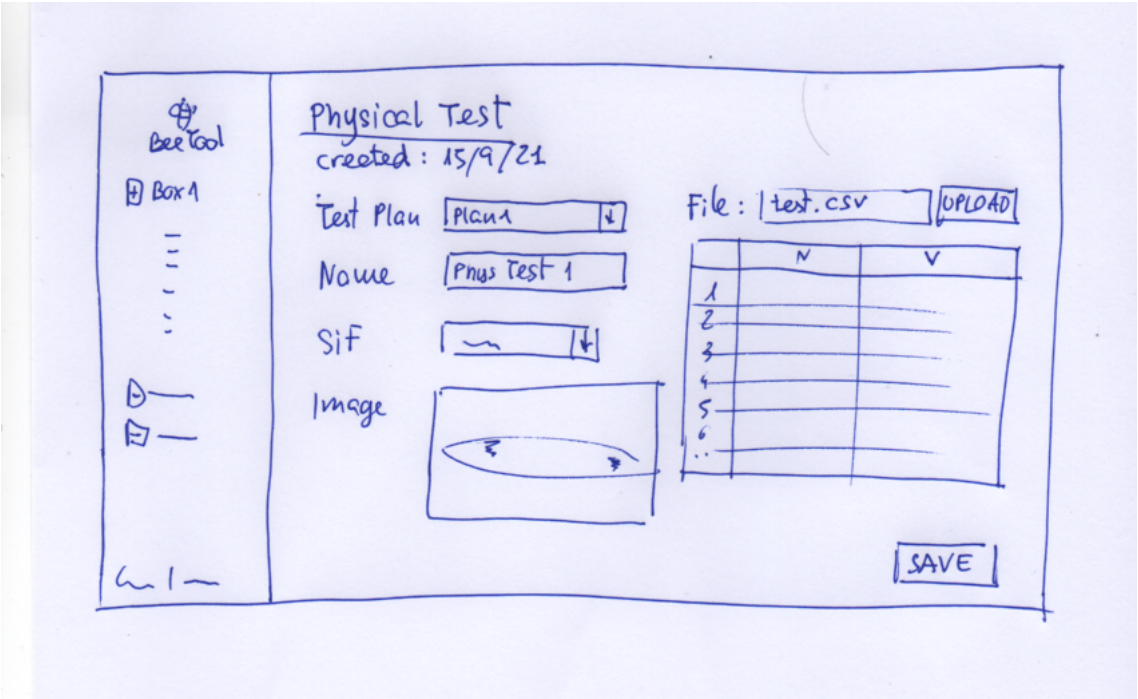


Figura 4.3: Boceto pantalla Physical Test

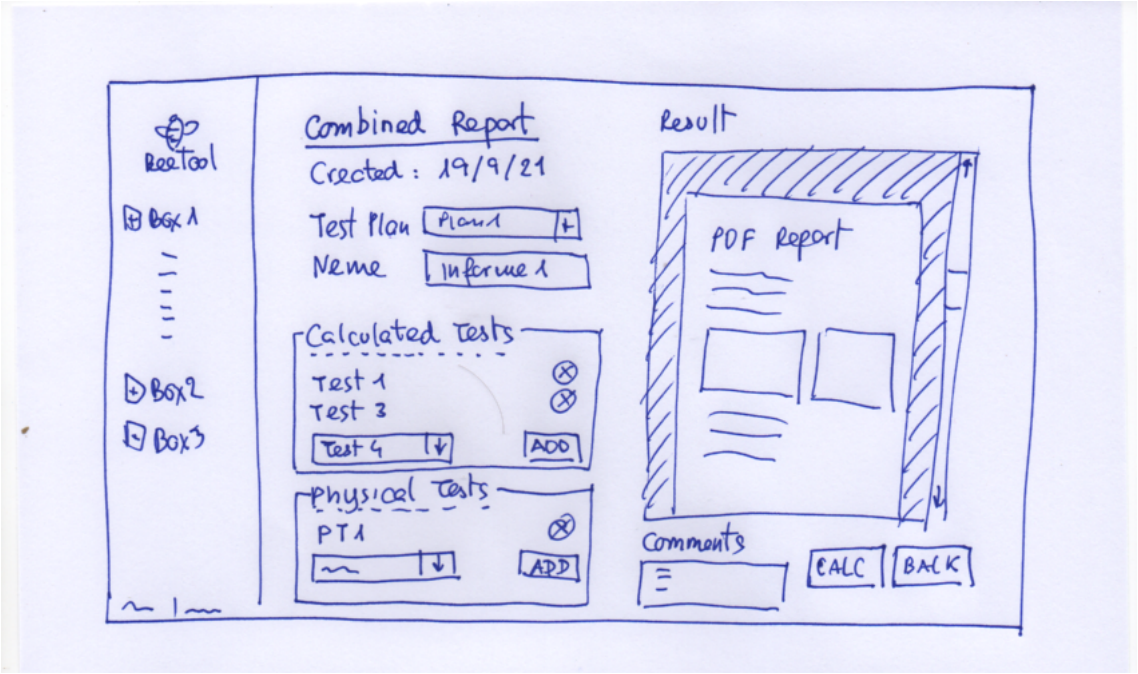


Figura 4.4: Boceto pantalla Combined Report

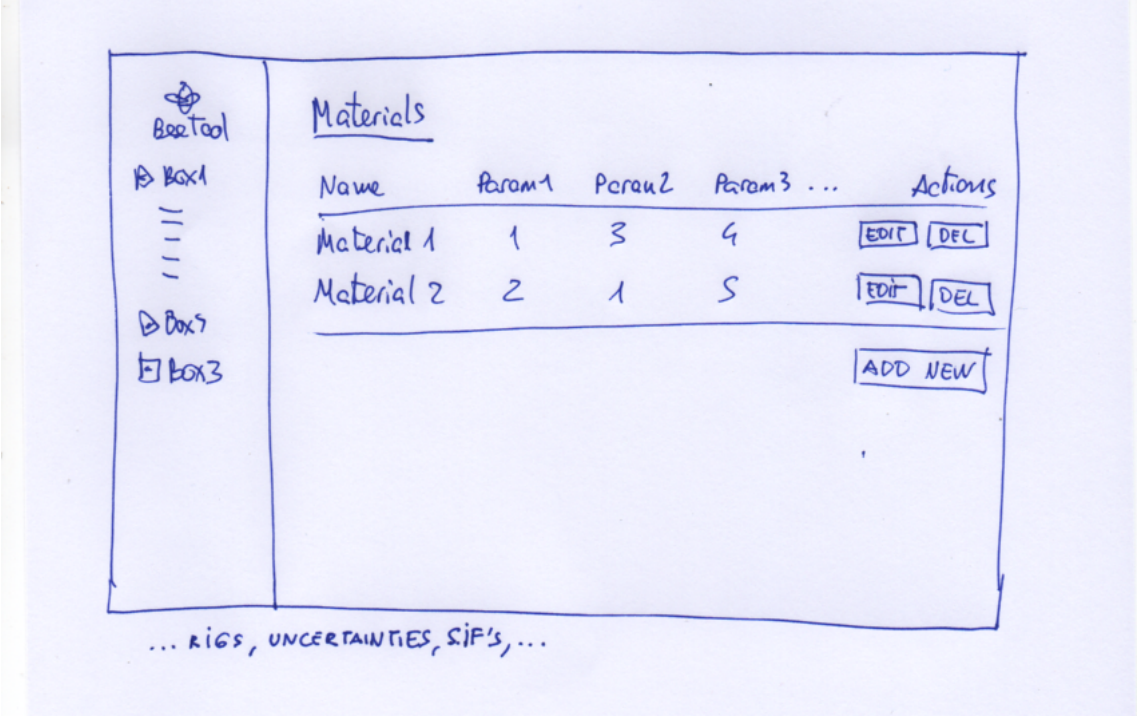


Figura 4.5: Boceto de pantalla Materials

En un principio, el panel se iba a componer de una barra lateral siempre activa que actuaría de menú, desde el que se podría acceder a una serie de pantallas: 'test plans', que serían simplemente agrupaciones de tests y de simulaciones; 'calculated tests'(figura 4.1, 4.2), que serían los tests que se simularían con la herramienta computacional del proyecto BEETool; 'Physical tests', que serían tests que han sido ya simulados pero de forma física, por lo que se tendría ya los resultados (figura 4.3) y 'Combined Reports', una pantalla donde mostrar los resultados y los informes de las simulaciones de los tests (figura 4.4) Además de estas pantallas, también habrían varias como la figura 4.5 donde mostrar los listados de materiales, condiciones climáticas, etc.

Estas últimas, estarían compuestas de un listado, a modo de base de datos, de todos los elementos creados, con diferentes botones que permitan acciones como puede ser borrar o editar, por ejemplo. A su vez, cada una dispondría de un botón para añadir nuevos elementos al listado, como se puede ver en la figura 4.5 el recuadro con el texto 'ADD NEW'.

Cuando se habla de test, se hace referencia a una serie de condiciones ambientales, tipos de materiales y características determinadas que necesita la herramienta computacional del

proyecto BEETool para poder realizar todos los cálculos y poder generar un informe al respecto. Todos estos datos también tendrían sus propias pantallas, como las mencionadas anteriormente, con un listado y capacidad de añadir nuevos elementos.

Estos nombres y pantallas fueron simplemente unos primeros bocetos con los que empezar a darle forma al panel pero no fueron los definitivos.

4.1.2. Diseño final

Más tarde, partiendo de esta idea inicial, se decidió estructurar el menú de una forma más ordenada, pero manteniendo la idea que se acaba de describir: todos los elementos de cada apartado en forma de lista y un botón con el que se podrán añadir nuevos elementos.

Estas decisiones eran tomadas en consenso entre los desarrolladores que posteriormente han realizado el TFG colaborativo y los miembros del proyecto BEETool.

Esta nueva estructura estaría dividida en cuatro partes: *Pre-processing*, donde se puede acceder a toda la parte de antes de simular un test, como pueden ser los materiales, o sus condiciones climáticas; *Test data*, en la que encontraríamos las pantallas de los tests fisicos mencionados anteriormente, que ya habían sido simulados sin la herramienta del proyecto, junto con una serie de condiciones determinadas; *Results*, en los que aparecerían los tests que, o se van a simular o ya están simulados con la herramienta del proyecto y la pantalla de test plans, donde simplemente se podrían agrupar diferentes tests de cualquier tipo. Por último, en la última parte tendríamos *Others*, que hace referencia a dos datos en concreto, que son opcionales para los tests. Se observa en la siguiente figura, 4.21

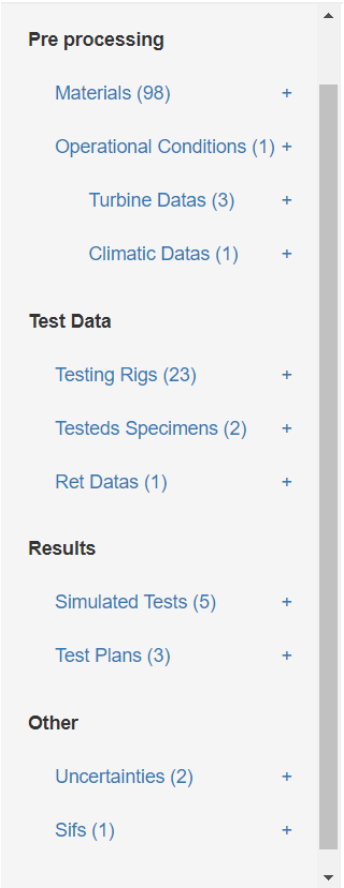


Figura 4.6: Estructura del panel

Para entender mejor la estructura, se diseñó también un diagrama simplificado, la figura 4.7, que describe el flujo que podría seguir un posible usuario de la aplicación web al utilizarla. En el no se muestran todos los apartados de la web, solo los más importantes ya que funcionan de la misma manera.

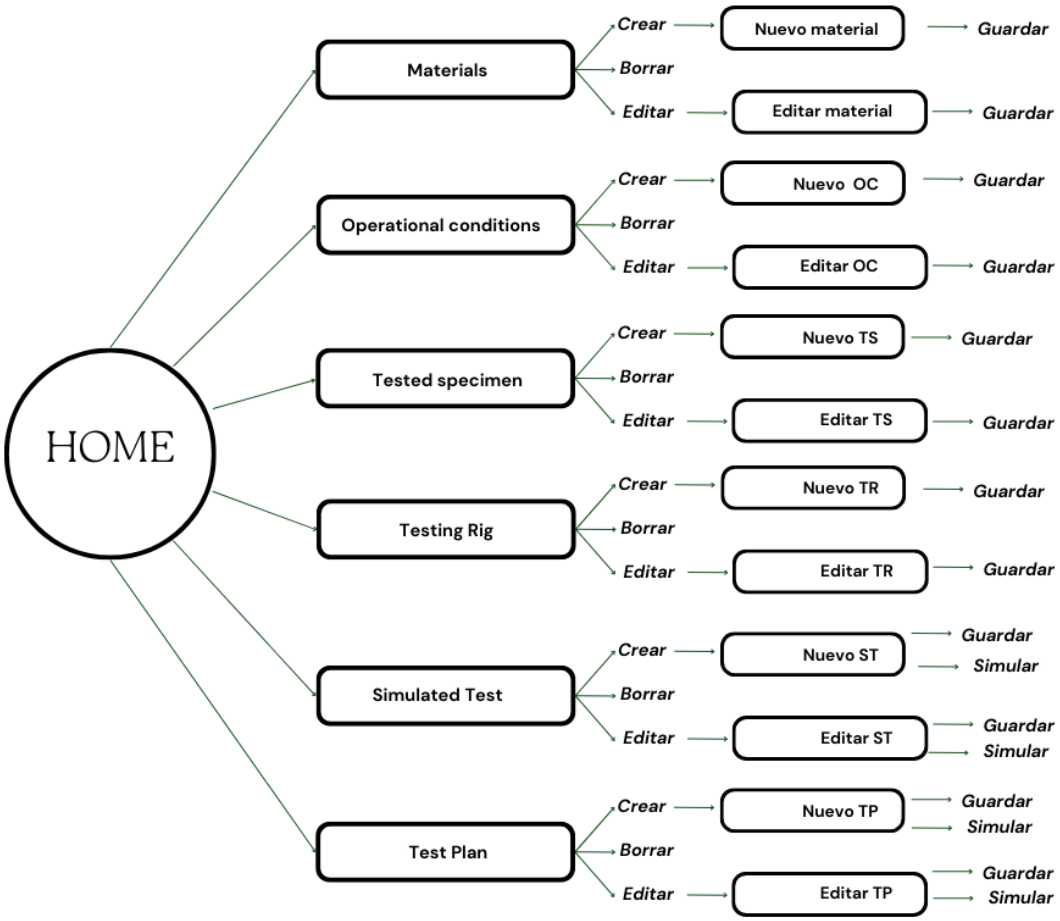


Figura 4.7: Diagrama de flujo simplificado

4.2. Fase 2: implementación

Antes de empezar a explicar cada parte del desarrollo es importante definir que, como se ha nombrado en el capítulo anterior, la aplicación web está alojada en un contenedor docker, pero esta parte no es objetivo de este TFG. Debido a esto, toda la fase de inicialización de un proyecto Slim con PHP, comandos comandos a ejecutar y paquetes necesarios no será explicado en profundidad pero es necesario comprenderlo.

Por lo tanto después de la fase de diseño del panel, y sin entrar en detalle en la contenerización e inicialización del proyecto, pasaríamos a simplemente ejecutar el contenedor con el comando `docker compose up -d`. Con esto tendríamos una aplicación web va-

cía, únicamente con un archivo index.php y los archivos de configuración, ejecutándose en el puerto 80 de nuestro servidor local.

En este punto la aplicación estaría compuesta por una carpeta llamada vendor, que contiene todos los paquetes con sus respectivos archivos para que una aplicación slim funcione, un par de archivos de configuración y un archivo index.php en el que básicamente tendríamos estas pocas líneas de código necesarias para que slim funcione.

```
$app = new \Slim\App(["settings" => [
        'displayErrorDetails' => ERROR_REPORTING
]]);

$app->run();
```

4.2.1. Carpeta vendor, composer.lock y composer.json

A partir de aquí se comienza con toda la fase de desarrollo partiendo de la idea que se tenía en la fase del diseño.

En primer lugar, la carpeta vendor que se mencionaba anteriormente se crea automaticamente al inicializar un proyecto slim. En ella encontramos todos los paquetes y bibliotecas necesarias para que funcione correctamente la aplicación, como puede ser la biblioteca twig, explicada en el capítulo tres, o la biblioteca lagan. Cabe destacar este último, ya que va a ser el que va a darle forma y estructura al proyecto. Investigando sobre esta biblioteca, se encontró un proyecto en GitHub que nos pudo servir como referencia para el nuestro.[13]

Además de la carpeta vendor, existen otros dos ficheros que se generan automaticamente: composer.lock y composer.json.

Composer es una herramienta que hemos utilizado para la gestión de dependencias en el proyecto, ya que es de las más populares en el desarrollo software con PHP.

El archivo composer.json es un archivo en formato JSON que contiene la configuración y metadatos del proyecto. En este archivo se definen las dependencias requeridas por el

proyecto, así como otras configuraciones importantes, como el nombre del proyecto, el autor, las rutas de autoloading y los scripts personalizados. El composer.json especifica las bibliotecas y paquetes que se deben instalar para que el proyecto funcione correctamente.

Por otro lado, el archivo *composer.lock* se genera automáticamente por Composer cuando se ejecuta el comando `composer install o composer update`. Este archivo se utiliza para bloquear las versiones exactas de las dependencias y sus dependencias transitivas en un momento específico. Contiene información detallada sobre las versiones exactas de las dependencias instaladas, incluyendo sus subdependencias y las restricciones de versión especificadas en el composer.json. El propósito principal del composer.lock es garantizar que todas las personas que trabajan en el proyecto utilicen exactamente las mismas versiones de las dependencias, evitando así posibles problemas de compatibilidad y asegurando la reproducibilidad del entorno de desarrollo.

4.2.2. Config.php y Setup.php

Una vez se tiene la configuración inicial y teniendo a index.php como archivo principal se deben crear dos archivos nuevos que añadiremos a index con los comandos que se muestran a continuación: config.php y setup.php.

```
require __DIR__ . '/../config.php';

require __DIR__ . '/../setup.php';
```

4.2.2.1. Config.php

El archivo config.php va a ser un archivo de configuración que va a contener variables y constantes globlaes del proyecto entre las que se encuentran, por ejemplo, el nombre del proyecto o la ruta donde ese alojará el proyecto, que en este caso será `'http://localhost:800/public/admin'` ya que se ha trabajado en local. También están definidas variables de configuración de la base de datos como el nombre de usuario y contraseña, los usuarios y las contraseñas que pueden acceder al panel con los diferentes permisos que tiene cada

uno (se explicará más adelante en que consiste esto) o diferentes rutas de archivos que serán necesarias posteriormente para el desarrollo.

Como se ha explicado anteriormente el archivo config.php se incluye en el archivo index.php que es el principal, lo que permite centralizar y mantener la configuración en un solo lugar para facilitar los cambios y la gestión de la aplicación.

Destacar que, para definir la ruta de la API (Application Programming Interfaces) de la aplicacion, que se utilizará cuando sea necesario enviar un archivo a la herramienta computacional de BEETool, se utilizan los siguientes comandos para obtener la dirección IP del ordenador en el que se ha trabajado. Se obtiene desde el archivo '.env' que se explica en el siguiente apartado.

```
$IP = getenv('IP');
define ("API_URL", "http://".$IP.":8000");
```

4.2.2.2. Setup.php

El archivo setup.php es un archivo que se utiliza durante el proceso de instalación o configuración inicial de una aplicación o sistema. Este archivo se ejecuta una sola vez y realiza tareas relacionadas con la configuración inicial, como mostrar los errores en pantalla, por ejemplo, para hacer más eficiente el proceso de desarrollo.

Además en este archivo se encuentran las inicializaciones de RedBean, haciendo uso de las variables globales mencionadas en el archivo anterior, de Lagan, para poder cargar los modelos con los que se va a trabajar y de Composer autoloader. Este último es una funcionalidad muy útil que tiene Composer ya que se encarga de cargar automáticamente las clases y archivos necesarios en el proyecto, sin la necesidad de requerirlos manualmente en cada archivo.

4.2.3. Configuración inicial

Además de los archivos mencionados anteriormente, existen una serie de archivos de configuración necesarios para que todo funcione correctamente, que se explican a continua-

ción:

4.2.3.1. .env

Al haber trabajado en pareja en el desarrollo, se han usado diferentes ordenadores, por lo que la dirección IP era distinta en cada uno. Esto suponía un problema ya que se tenía que cambiar en distintos archivos cada vez que se actualizaba el proyecto. Para poder solucionarlo se creó este archivo.

Un archivo .env es un archivo de configuración para almacenar variables de entorno muy común en el desarrollo de aplicaciones web. Estas variables son utilizadas para configurar y personalizar el comportamiento de una aplicación en diferentes entornos.

El formato del archivo .env puede variar dependiendo del lenguaje o framework utilizado, pero generalmente sigue una estructura simple de pares de clave-valor. Cada línea del archivo contiene una variable y su respectivo valor, separados por un signo igual (=).

El propósito principal de utilizar un archivo .env es separar la configuración sensible de una aplicación del código fuente, lo que brinda una mayor flexibilidad y seguridad. En el caso de este proyecto solo se ha incluido como variable de entorno la dirección IP de los desarrolladores del proyecto.

Al ser información confidencial, este archivo no se incluía en el sistema de control de versiones.

4.2.3.2. php.ini

El archivo php.ini es un archivo de configuración utilizado por PHP que contiene una serie de directivas que controlan el comportamiento y las características de PHP en el servidor.

El propósito principal del archivo php.ini es permitir la personalización de la configuración de PHP según las necesidades específicas de la aplicación como puede ser, en este caso la configuración de directivas de PHP que controlan diversos aspectos como la configuración de la memoria o los límites de tiempo de ejecución.

También encontramos alguna configuración de seguridad como configurar los informes

de errores o configurar el límite de tamaño de carga de archivos:

```
memory_limit = 256M
post_max_size = 256M
upload_max_filesize = 256M
display_errors= On
log_errors = On
error_log =/dev/stderr
```

4.2.3.3. apache2.conf y .htaccess

Por último, el archivo 'apache2.conf' y el archivo '.htaccess' son dos archivos de configuración utilizados en el servidor web Apache. Para desarrollar una aplicación web, es necesario disponer de un servidor web, cuyo propósito principal es recibir las solicitudes de los clientes (navegadores web) y responder enviando los archivos y recursos solicitados. Por tanto, estos dos archivos son los encargados de la configuración de Apache como servidor web, el primero globalmente, y el segundo del sitio web en concreto de este proyecto.

4.2.4. Estructura de carpetas

Una vez explicados todos los archivos de configuración es importante hablar de la estructura de carpetas del proyecto, figura 4.8 :

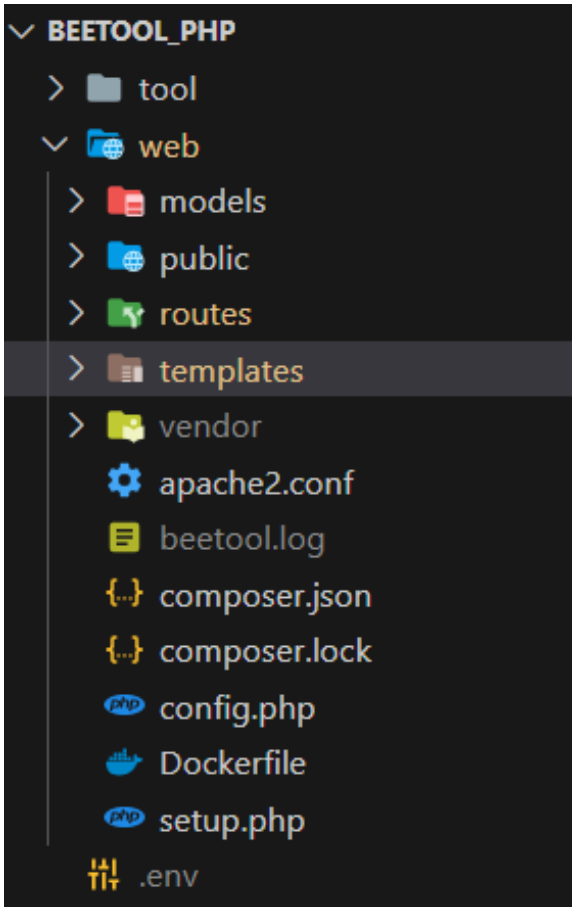


Figura 4.8: Estructura de carpetas de la aplicación

En un principio está dividido en dos grandes carpetas *tool y web*. En el caso de este TFG solo se ha trabajado en la segunda. La carpeta tool es un réplica de lo que tendría que tener el proyecto BEETool para poder recibir los datos que se envíen desde la aplicación web pero no se va a explicar en profundidad ya que no está dentro de los objetivos de este proyecto. Está descrita en profundidad en la otra parte de este TFG colaborativo.

Dentro de la carpeta web, encontramos una serie de subcarpetas en las que se ha dividido el panel web, además de los archivos de configuración que se acaban de describir. Las subcarpetas están formadas por: 'Models', donde podemos encontrar cada uno de los modelos que forman la aplicación y de los que se ha hablado anteriormente; 'Public', que es la carpeta a la que un usuario va a tener acceso y la que contiene el fichero principal 'index.php', además de las configuraciones específicas del panel como son 'php.ini' y '.htaccess'; 'Routes', que va a contener todas las comunicaciones internas de la página

web además de un fichero con funciones determinadas; 'Templates', donde se encuentran las plantillas Twig con el código HTML que se va a mostrar en pantalla y por último la carpeta vendor explicada anteriormente.

Una vez entendida la estructura de los archivos del proyecto, se comienza a darle forma partiendo del archivo principal mencionado varias veces, index.php. En el añadimos los archivos de configuración necesarios, como se ha explicado en anteriores apartados y en caso de que haya errores también activamos que se muestren por pantalla. Después, además de inicializar Slim, se va a configurar Twig, para que sea capaz de acceder a la carpeta 'Templates'.

Importante destacar la creacion de la variable 'view', que se va a utilizar para renderizar las pantallas oportunas en cada momento.

Por otro lado, también se van a configurar las variables con los archivos de las rutas del panel que están dentro de la carpeta 'Routes'. En resumen, dentro del archivo index, se consigue centralizar la mayoría del proyecto y es el archivo que sirve de control de todos los demás.

Por último, en relación con la seguridad del panel, se decidió establecer un login sencillo como el siguiente que también se gestiona desde este archivo:

```
$app->add(new \Slim\Middleware\HttpBasicAuthentication([
        'path' => array('/admin'),
        'secure' => FORCE_SSL,
        'relaxed' => ['localhost'],
        'users' => $panel_users
]));
```

Este login básico se genera añadiendo a la variable 'app', que es la variable principal del proyecto, una instancia de la clase 'HttpBasicAuthentication' que se encargará de verificar las credenciales proporcionadas por el cliente en la cabecera HTTP 'Authorization'. Si estas credenciales son válidas, es decir, coinciden con las establecidas en el archivo config.php, se podrá acceder al panel, de lo contrario, se volverán a solicitar.

4.2.5. Modelos de Lagan

Se ha mencionado varias veces los modelos con los que trabaja Lagan y en este apartado se van a explicar en detalle.

Un modelo, en general en Lagan, es una clase en la que utilizando un constructor se inicializa y se configura el estado inicial de un objeto. Dentro del cuerpo del constructor se van a configurar diferentes parámetros suiguiendo la estructura `this->clave = 'valor'` donde clave serán las diferentes propiedades del modelo y valor, como su nombre indica, aquello que está asociado a dicha propiedad, ya sea un nombre, un número o también un nuevo array de propiedades. En los comandos siguientes se puede observar un ejemplo sencillo del modelo 'Uncertainty':

```
<?php
namespace Lagan\Model;

class Uncertainty extends \Lagan\Lagan {
    function __construct() {

        $this->type = 'uncertainty';
                $this->plural = 'uncertainties';

        // Description in admin interface
        $this->description = 'Uncertainty parameters';
        $this->list_columns = 8;
        $this->properties = [
            [
                'name' => 'title',
                'description' => 'Uncertainty Name',
                'required' => true,
                'searchable' => true,
                'type' => '\Lagan\Property\Str',
                'input' => 'text',
```

```
                    'column' =>'pre',
                ]
            ];
        }
}
?>
```

Todos los modelos siguen la misma estructura: una clase, con un contructor que asigna un tipo, un plural, una descripción del modelo y un array de propiedades. Estos datos se utilizarán a la hora de mostrarse por pantalla como podemos ver en la figura.

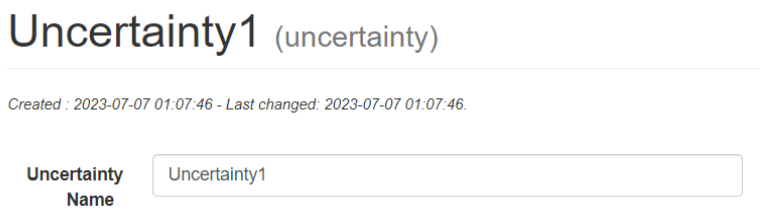


Figura 4.9: Ejemplo de modelo sencillo de LAGAN

Este array de propiedades que se ha mencionado, va a estar compuesto por tantos elementos, como campos de datos sean necesarios. En este caso de ejemplo, solo existe un elemento, pero dependiendo del modelo, existirán un número determinado de campos para que el usuario pueda introducir datos.

Cada elemento del array de propiedades tiene, a su vez, otro conjunto de propiedades y en todos los casos tendrá la misma estructura: un nombre, una descripción, dos propiedades booleanas (true/false) para definir si pueden ser buscadas o si es obligatorio ese campo, un tipo y un campo input que definirá la plantilla que utilizará el campo de datos en concreto. Esta última propiedad es la más importante, ya que Lagan tiene algunos campos predeterminados.

Estos campos predeterminados se encuentran en la carpeta 'Property-templates' y se crean al inicializar el proyecto. Para este proyecto no era suficiente con los que Lagan tiene por defecto, por lo que posteriormente se explicará como se ha creado, por ejemplo, un nuevo campo de datos para mostrar un archivo con formato json.

En cuanto a los tipos, se han utilizado únicamente tres de los que Lagan tiene por defecto: 'Str', para la mayoría de campos, ya que permite trabajar con strings (cadenas de caracteres); 'Many to one', en el caso de que sea necesario establecer relaciones de campos 'muchos a uno' con otros modelos; o para relaciones 'muchos a muchos' se utilizara el tipo 'Many to many'.

Estas relaciones de campos ocurrirán en los casos en los que en un modelo se necesite introducir como un campo, un modelo previo, es decir, para simular un test, va a ser necesario introducir un campo material, que este a su vez es un modelo con diferentes campos para las propiedades de cada material. Por lo tanto, si existen varios materiales creados, se aplicará una relación de 'muchos a uno', y el campo 'Many to one' permitirá escoger un material a partir de todos los que haya creados previamente en la base de datos.

4.2.6. Modelos del panel

Como se ha explicado anteriormente, los modelos están separados en cuatro categorías:

4.2.6.1. Pre-processing

En esta categoría encontramos los primeros modelos previos a realizar cualquier test.

Materials: está formado por un campo de texto para el nombre del material en concreto. En todos los modelos es obligatorio que, como mínimo, en el array de propiedades haya un campo para el nombre del modelo cuyo valor tiene que ser title, ya que, se usará como título en la pantalla de cada modelo. Además, se necesitan quince campos numéricos para introducir características del material, como pueden ser la resistencia o la densidad y un último campo de texto en el que añadir un comentario en el caso de que se necesite.

Para los campos numéricos, se pidieron unas características especiales desde el proyecto BEETool, por lo que no era suficiente la plantilla con la que cuenta Lagan por defecto para estos casos. Por tanto, dentro de la carpeta 'Public' se creó una subcarpeta 'Property-templates-a3p' en la que se alojarán todas las plantillas personalizadas.

En este caso y como se observa en las siguientes lineas de código, se pidió que los campos

numéricos, además de los valores que se han descrito, tuvieran una pequeña ayuda cuando se acercase el ratón a dicho campo (’tooltip’), un valor máximo y mínimo y su unidad de medida. Se establecieron valores predeterminados a modo de ejemplo:

```
[
    'name' => "young22",
    "description"=> "Young Modulus 22",
    'tooltip' => "Introduce ejemplo",
    "max"=> 100000, "min"=> 0, 'step'=>0.000000000001,
    'unit'=>'m/s',
    'searchable' => false,
    'required' => false,
    'type' => '\Lagan\Property\Str',
    'input' => 'a3p_number',
    'column' =>'half',
],
```

Por último, la clave ’column’ se utiliza únicamente en los campos que se van a mostrar en primer lugar, con el valor ’pre’, los que serán el primer campo de la segunda columna, con el valor ’half’, y en los campos que se mostrarán en último lugar, con el valor ’post’.

En este caso, podemos observar en la figura 4.10 que, este campo en concreto se mostrará como primer campo de la segunda columna, al tener el valor ’half’.

De esta forma se consigue que todos los campos estén ordenados de manera específica según las necesidades del modelo, ya que es totalmente personalizable.

Una vez establecidos los valores mencionados, se deben mostrar por pantalla. Para ello, dentro de la carpeta ’Property-templates-a3p’, se genera una nueva que va a contener un fichero HTML como el siguiente:

```
<div class="form-group{% if property.required %}
    required{% endif %}">

        <label for="{{ property.name }}" class="col-sm-2
        control-label">{{ property.description }}
```

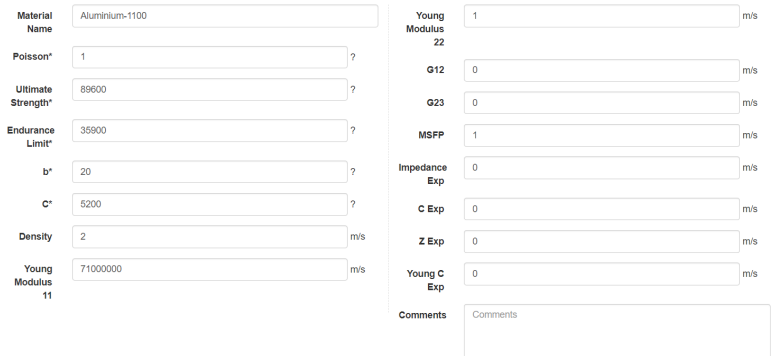


Figura 4.10: Campos del modelo material

```
{% if property.required %}*{% endif %}
</label>

<div class="col-sm-10">
        <input
        style="width:90%;display:inline-block;"
        type="number"
        class="form-control"
        id="{{ property.name }}"
        name="{{ property.name }}"
        placeholder="{{ property.description }}"
        value="{{ bean[property.name] }}"
        title="{{ property.tooltip }}"

    {% if property.required %}
        required="required"
    {% endif %}

        {% if property.max %}
        max="{{ property.max }}"
    {% endif %}
```

```
                {% if property.min %}
                min="{{ property.min }}"
            {% endif %}

                {% if property.step %}
                step="{{ property.step }}"
            {% endif %}>

                {% if property.unit %}
                {{ property.unit }}
            {% endif %}

        </div>
</div>
```

Este archivo actúa como plantilla para todos los campos numéricos de la aplicación web y esta formado por dos elementos 'div' principales, uno anidado dentro de otro.

En HTML, 'div' es un elemento de bloque que se utiliza para crear una sección o contenedor genérico en una página web. El elemento 'div' no tiene un significado semántico específico, sino que se utiliza como una estructura genérica para agrupar y organizar otros elementos y contenido dentro de una página.

El uso principal de 'div' es proporcionar un contenedor para otros elementos y permitir aplicar estilos y manipular su contenido con CSS y JavaScript. Al agregar una o varias etiquetas 'div' en el código HTML, se pueden crear divisiones lógicas y estructurales en la página para facilitar la presentación y organización del contenido.

En el primero de ellos, se crea un elemento 'label' que servirá como etiqueta para el nombre de la propiedad que se solicite, ya sea densidad, resistencia, etc. En el caso de que esta propiedad sea obligatoria, se escribirá un asterisco después del nombre.

En HTML, 'label' es un elemento utilizado para asociar un texto descriptivo con un elemento de formulario, como un campo de entrada (<input>) . El elemento 'label' propor-

ciona una etiqueta legible por humanos que describe el propósito o la función del elemento de formulario.

El uso de ’label’ mejora la usabilidad y accesibilidad de los formularios, ya que permite que los usuarios hagan click en el texto de la etiqueta para seleccionar o enfocar el elemento de formulario asociado.

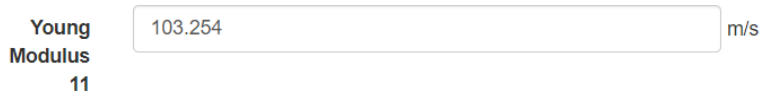


Figura 4.11: Ejemplo de campo numérico

Otro aspecto importante es el uso de la herramienta Twig para el acceso a las variables. Se describe en detalle en la siguiente sección, pero como se observaba anteriormente, para acceder a una variable en concreto se utiliza la sintaxis `{{ variable.propiedad }}`. De esta forma, se consigue una plantilla genérica que hace mucho más eficiente el código de la aplicación.

Dentro del ’div’ principal, además del elemento ’label’ que describe la propiedad, se encuentra un elemento ’input’ con diferentes atributos, anidado dentro de otro elemento ’div’. En HTML ’input’ es un elemento utilizado para crear campos de entrada en formularios web. Es uno de los elementos más utilizados y versátiles para recopilar datos ingresados por los usuarios. Este puede tener varios tipos dependiendo del dato que se espera recibir por parte del usuario, siendo en este caso de tipo númerico ya que todas las propiedades de los materiales serán números.

Este elemento ’input’ contiene varios atributos específicos como, por ejemplo, ’placeholder’, que escribirá un valor por defecto antes de que el usuario introduzca nada, en este caso la descripción de la que se habla en el anterior apartado.

Además, contiene los atributos para los valores máximos y mínimos permitidos, así como para la ayuda al acercar el ratón al campo en el atributo ’title’, como se ha mencionado anteriormente. A estas propiedades, se accede con la sintaxis propia de Twig.

Por último, se escribirá en pantalla la unidad de medida del campo, en caso de que exista.

Operational Conditions: este modelo está formado por un campo de texto para el nombre (este campo es obligatorio para todos los modelos), dos campos 'Many to one' para seleccionar los datos de la turbina y las condiciones climáticas y un area de texto para escribir el comentario que desee el usuario si fuese necesario.

Para los campos 'Many to one', Lagan tiene por defecto una plantilla que permite acceder a otros modelos y mostrar en pantalla un listado seleccionable de todos los elementos que se hayan creado en ese modelo, es decir, en este caso, se creará un desplegable con las turbinas y otro con las condiciones climáticas que se hayan creado previamente, como se puede observar en la figura 4.12.

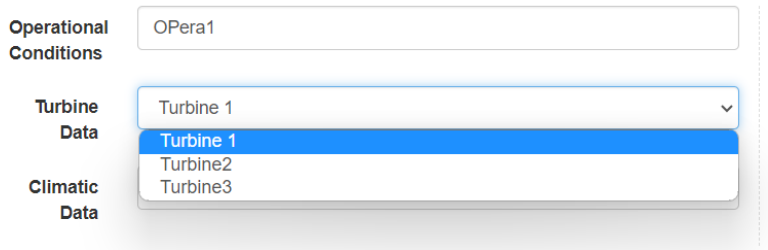


Figura 4.12: Ejemplo de campo Many to One

Climatic Data y Turbine Data: en estos dos casos, encontramos una estructura similar a la que tenemos en el modelo 'Metrials'. Un campo para el nombre, un área de texto para los comentarios y distintos campos numéricos para las características específicas de cada uno, como puede ser el diámetro de la turbina, su velocidad o la intensidad de la lluvia en el caso de las condiciones climáticas.

Destacar la importancia del uso de las plantillas, ya que para estos casos se utiliza la misma que se había creado en el modelo anterior, aprovechando el mismo código. Esto mejora la eficiencia en el desarrollo y reduce la duplicación de código. Además, facilita la actualización y personalización del diseño en el proyecto.

Al tener una plantilla centralizada, cualquier cambio en el diseño se puede realizar de forma rápida y sencilla, y se reflejará en todas las páginas que utilizan esa plantilla. Esto permite mantener un diseño actualizado y adaptarlo a las necesidades del proyecto.

Figura 4.13: Modelo Climatic Data

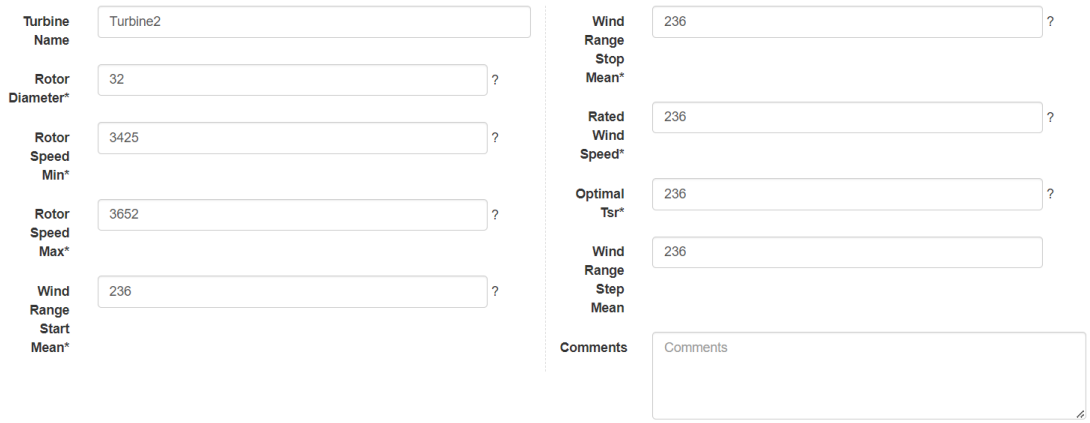


Figura 4.14: Modelo Turbine Data

4.2.6.2. Test data

Dentro de este apartado encontramos tres modelos distintos:

Testing rigs: este modelo esta compuesto por las condiciones específicas de un test, que son distintas a las que encontraríamos en la parte de 'Pre-procesing como puede ser el diámetro de la gota de agua o su velocidad. En total doce nuevos campos numéricos como los descritos anteriormente.

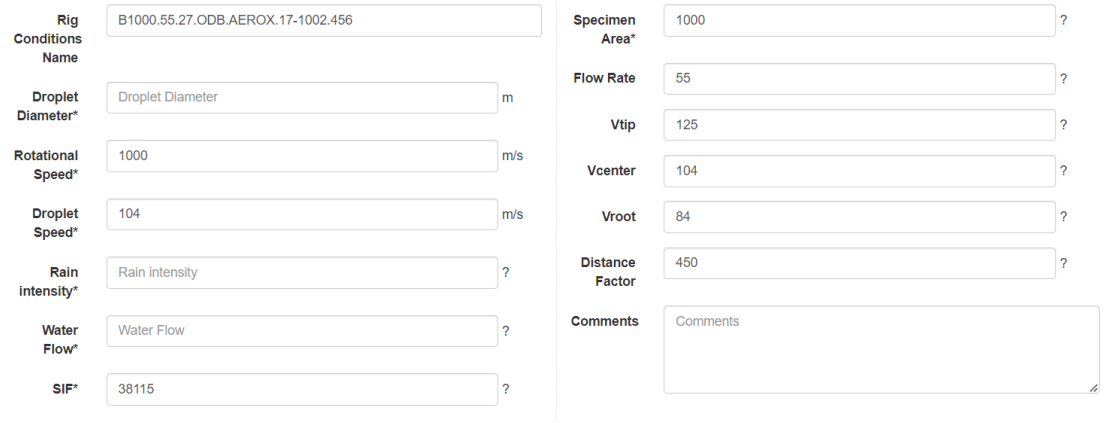


Figura 4.15: Modelo Testing Rig

Tested Specimens: como se ha explicado en apartados anteriores, los tested specimens son tests que ya han sido simulados previamente, que tambíen podrán enviarse a la herramienta computacional del proyecto BEETool para generar un informe sobre ellos. De esta forma comparar con otros test simulados y poder sacar conclusiones.

Están formados por el campo nombre, tres campos 'Many to one' para seleccionar tres tipos de materiales distintos, un campo numérico para el grosor de la capa, dos nuevos campos 'Mnay to one' para seleccionar 'Operational conditions' y 'Ret Data' (este último se explica a continuación) y el área de texto de comentarios.

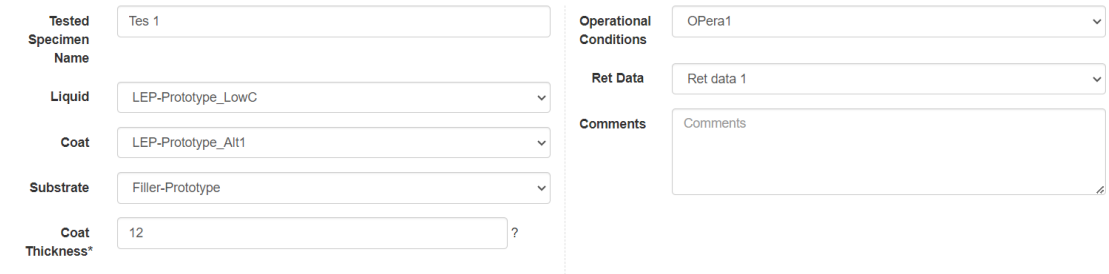


Figura 4.16: Modelo Tested Specimens

Ret Data: en el caso de este modelo se encuentran diferentes campos que pueden ser añadidos a los 'Tested Specimens'. Un campo para el nombre,un campo para los comentarios, un campo numérico, un campo para añadir imágenes que puedan ser necesarias o descriptivas del test, un campo a modo de tabla en el que se puedan añadir comentarios para las

imágenes añadidas en el campo anterior y por último, uno en el que poder subir a la página web un fichero en formato csv.

Estos tres últimos campos no son predeterminados de Lagan, por lo que fueron proporcionados por la empresa donde se realizó el proyecto.

Cabe destacar que se realizaron pequeñas modificaciones en las plantillas proporcionadas para que quedasen conforme se requería en el panel como por ejemplo el nombre que apaerece debajo de las imágenes o las columnas con los encabezados del fichero csv que se muestra en pantalla.

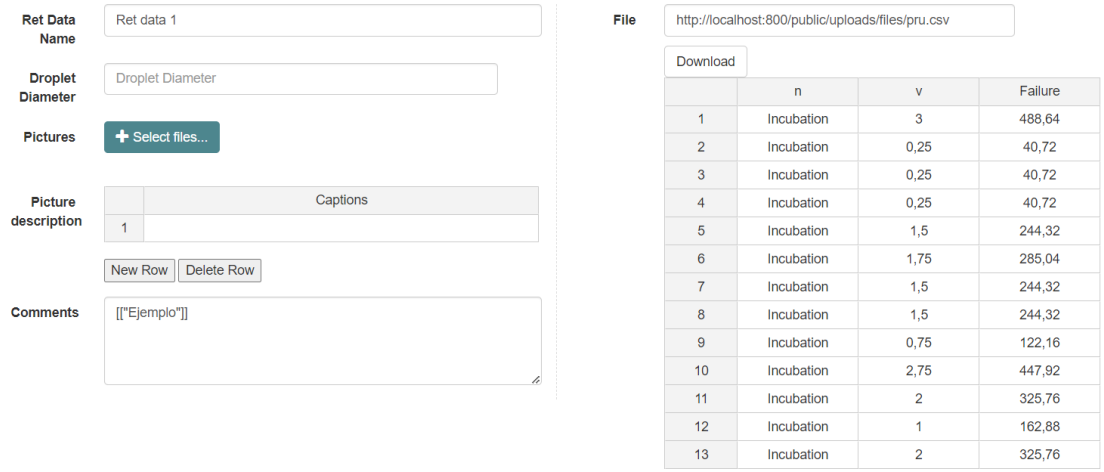


Figura 4.17: Modelo Ret Data

4.2.6.3. Results

En tercer lugar se encuentra la parte de resultados, donde existen dos nuevos modelos:

Simulated tests: este modelo es el encargado de agrupar todos los modelos anteriores necesarios para simular un test con la herramienta computacional del proyecto BEETool. Está compuesto por el campo nombre, seis campos 'Many to one' con los que seleccionar tres tipos de materiales, las condiciones operacionales del test, el 'Testing Rig' y la 'Uncertainty', un campo numérico y el aéa de texto para comentarios.

Por otro lado se han añadido tres campos 'especiales' que se irán rellenando automáticamente conforme se avance en el proceso de simular un test explicado en el apartado

de resultados. Estos campos especiales mostrarán un resumen de los datos introducidos en el test en formato json llamado 'In', un resumen de los resultados obtenidos también en formato json junto con un visor en PDF del formulario obtenido después de realizar la simulación, llamado 'OUT', y un campo que mostrará el estado del test en concreto, pudiendo ser pendiente, calculando o finalizado.

El PDF se muestra en pantalla mediante la siguiente línea de código. En ella simplemente se utiliza una etiqueta especial de HTML, 'embed', de tipo 'application/pdf'.

```
<embed
    src="{{app_url}}/reports/pdf/{{report}}"
    type="application/pdf"
    width="100%"
    height="300px"
    />
```

Se muestran en la siguiente figura 4.18:

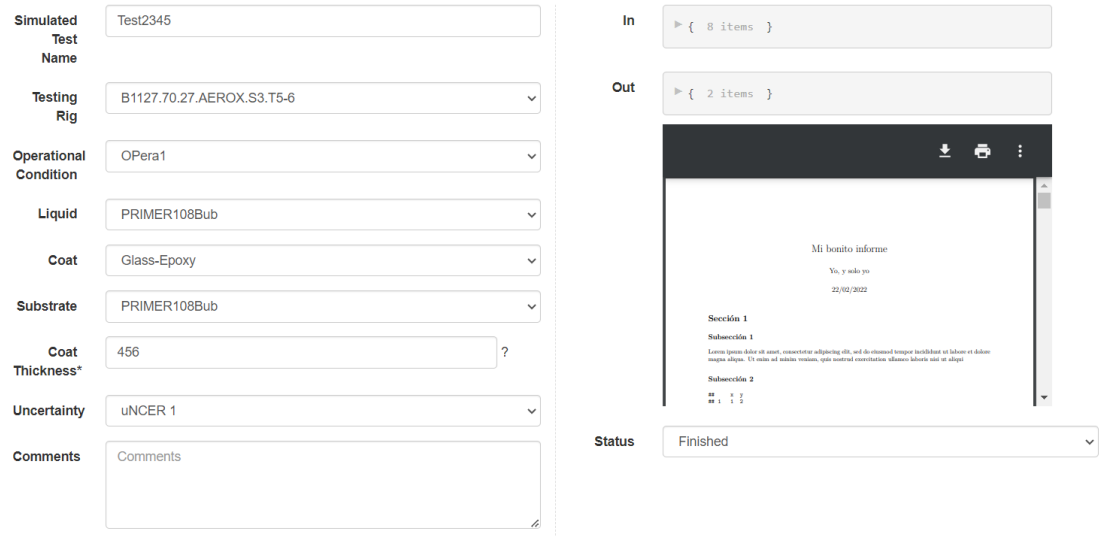


Figura 4.18: Modelo Simulated Test

Test plans: en este apartado es donde se pueden agrupar los dos distintos tipos de tests y enviarlos también a la herramienta computacional para simularse. Encontramos un campo nombre y uno de comentarios como en todos los modelos anteriores y los tres campos

especiales que se han explicado en el modelo 'Simulated tests' ya que la forma de enviarlos y de recibir los resultados va a ser idéntica.

Por otro lado, para poder seleccionar los distintos tests que se desean simular era necesario un campo del tipo 'Many to Many', ya que pueden existir 'muchos' tests y se podran seleccionar más de uno para agruparse en un 'Test Plan'.

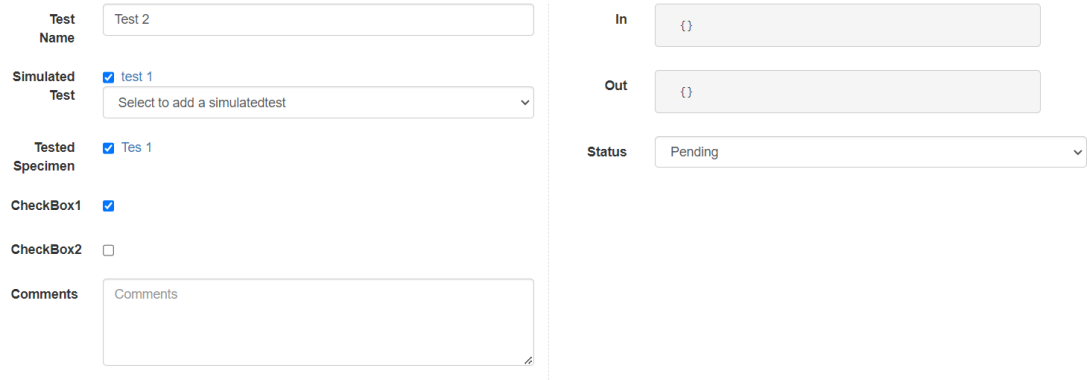


Figura 4.19: Modelo Test Plans

En el último apartado de la aplicación se encuentran dos últmios modelos que simplemente muestran un campo nombre a modo de descripción. Se trata de los modelos 'Uncertainty' y 'Sif'.

4.2.7. Carpeta templates

En esta carpeta se encuentra todas las plantillas que se han usado para la aplicación web, con todo el código HTML, CSS y JavaScript necesario para su correcta visualización. Además, como se ha explicado en esta memoria, se hace uso de la herramienta Twig.

El funcionamiento básico de Twig se basa en tres conceptos principales: plantillas, variables y filtros. Las plantillas de Twig son archivos que contienen código HTML mezclado con instrucciones Twig. Estas plantillas se utilizan para generar la salida final del contenido dinámico. Dentro de las plantillas encontramos las variables y los filtros.

En Twig, las variables se utilizan para almacenar y mostrar datos dinámicos en las plantillas. Pueden representar valores como texto, números, arreglos u objetos. Para mostrar

el contenido de una variable en una plantilla, se utiliza la doble llave {{ }}. Los filtros de Twig se utilizan para manipular o modificar datos antes de mostrarlos en la plantilla. Los filtros se aplican a una variable utilizando el carácter de tubería (|).

La sintaxis de Twig se destaca por su legibilidad y su enfoque en la separación de preocupaciones. Al utilizar estructuras de control, como if o for se puede aplicar lógica condicional y bucles en las plantillas sin mezclar la lógica del proyecto con el código HTML de la siguiente forma:

```
{% if usuario %}
    <p>Bienvenido, {{ usuario.nombre }}</p>
{% else %}
    <p>Inicia sesion para continuar</p>
{% endif %}
```

Podemos observar en el ejemplo sencillo anterior, que para introducir esta lógica se usa también una sintaxis especifica propia de Twig. En este ejemplo, en el caso de que exista la variable usuario, mostrará por pantalla un párrafo con el nombre, y en caso contrario, un párrafo distinto.

Para el caso de los bucles, funciona de la misma forma. Como vemos a continuación, en este bucle, se recorre una lista de nombres y se muestran por pantalla cada uno:

```
  {% for nombre in nombres %}
    <li>{{ nombre }}</li>
  {% endfor %}
```

En cuanto a la estructura de las plantillas se han creado tres principales: 'Base', que va a contener el esqueleto de la aplicación, la barra lateral a modo de menú y la barra superior con los logos del proyecto, además de actuar como archivo principal conteniendo las etiquetas 'head' y 'body' propias de cualquier archivo HTML; 'Beans', que mostrará un listado a modo de tabla con todos los elementos de cada modelo y 'Bean', que es la plantilla que muestra un elemento en concreto. Se puede ver en la figura 4.20 un esquema simplificado del funcionamiento.

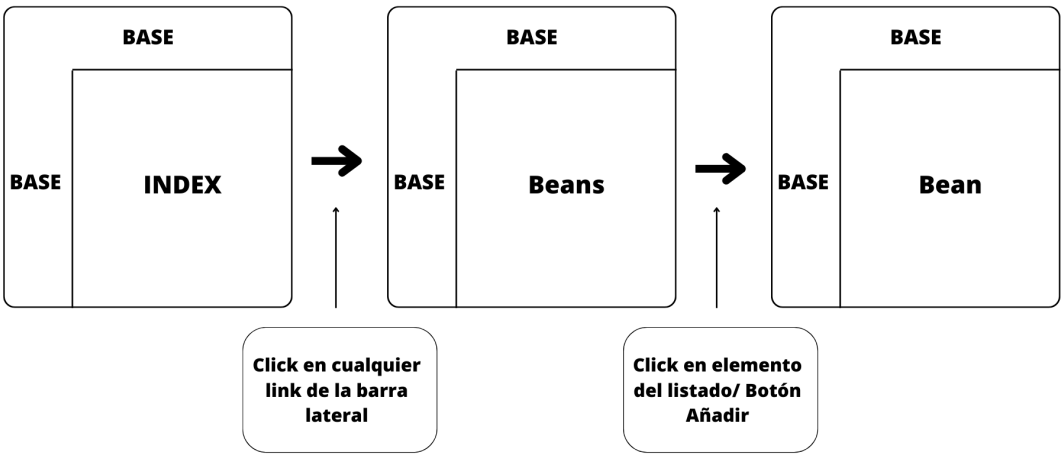


Figura 4.20: Esquema de la estructura de plantillas

Con otras palabras, en pantalla siempre se mostrarán las barras superiores y laterales (Base) y por ejemplo, si pulsamos en materiales saldrá un listado con los materiales que se hayan creado (Beans).

Para ver en detalle alguno de ellos, pulsaremos en el nombre y aparecerán todos los campos del material (Bean), explicados cada uno en el apartado de modelos.

4.2.7.1. Base

Este archivo actúa como el archivo principal en cuanto a interfaz de usuario se refiere. En todas las pantallas posibles que se puedan generar, siempre se va a mostrar, ya que estas pantallas son formadas a partir del código de este archivo.

En él, encontramos el bloque 'head', que contiene todos los metadatos de la interfaz, como puede ser el título, o diferentes enlaces a hojas de estilo o a scripts contenidos en la carpeta 'assets'.

Dentro de esta carpeta encontramos todo lo relacionado con el diseño de la página web, los archivos CSS y Javascript de las librerías de Bootstrap y JQuery o las imágenes que

se van a mostar en la aplicación web.

En este bloque también se encuentran algunos estilos que se le han dado a la web como pueden ser los márgenes o los colores de la barra lateral, como se puede ver en las siguientes líneas:

```
.nav-sidebar {
    margin-right: -21px; /* 20px padding + 1px border */
    margin-bottom: 20px;
    margin-left: -20px;
}

.nav-sidebar > li > a {
    padding-right: 20px;
    padding-left: 20px;
}

.nav-sidebar > .active > a,
.nav-sidebar > .active > a:hover,
.nav-sidebar > .active > a:focus {
    color: #fff;
    background-color: #000;
}
```

Como se ha explicado en los objetivos, con el panel web se buscaba que fuese funcional por lo que no se ha profundizado en exceso en la apariencia y se ha dejado en un segundo plano. Es por esto, que no se va a explicar en detalle.

Después del bloque 'head', tenemos el bloque body, en el que se encuentra toda la parte visual. Como se ha mencionado, dentro de este bloque se encuentra la barra de navegación superior, haciendo uso de las etiquetas 'nav' y 'div' que actuan como contenedores para los diferentes logos. Mencionar que el logo de la izquierda actúa como un link que al ser pulsado llevará al usuario a la página inicial.

Por otro lado, haciendo uso de las etiquetas 'ul' y 'li', utilizados en HTML para generar

listas sin un orden preestablecido como podría ser numérico, y en nuevos contenedores 'divs', se crea la barra de navegación lateral que va a actuar como un menú.

```
{% for beantype in beantypes %}

{% if beantype=='retdata' or beantype=='testingrig' or
 beantype=='testedspecimen' %}

<li>
    <a href="{{ path_for('listbeans', { 'beantype':
    beantype})}}?limit=50"
    class="capitalize"
    style="padding-left: 40px;">
    {{plurals[beantype] }}</a>

    <a href="{{ path_for('addbean', { 'beantype':
    beantype }) }}"
    class="addbean">+</a>
</li>

{% endif %}
{% endfor %}
```

Como vemos en el ejemplo del código, se recorre una lista con todos los modelos existentes llamada 'beantypes'. En el siguiente apartado se explica como se genera esta lista y las demás variables del proyecto.

En este caso (es el ejemplo de la parte de 'Test Data'), si los modelos son 'retdata', 'testing rig', or 'testedspecimen', se mostrará un elemento nuevo de la lista (etiqueta 'li'), que va a contener un enlace al listado de ese modelo, y un enlace con un símbolo '+' que permitirá añadir un elemento nuevo a ese modelo.

Se hace lo mismo con todos los demás obteniendo lo que podemos ver en la siguiente figura, 4.21:

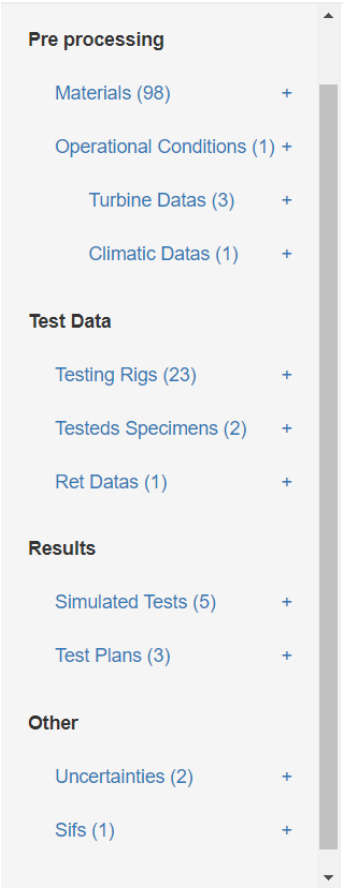


Figura 4.21: Parte del panel que genera el fichero base

Por último, y para que este archivo sirva como plantilla principal, se va a abrir un 'nuevo bloque de contenido'. Gracias a Twig, en este nuevo bloque se va a poder insertar lo que requiera el proyecto, ya sea la parte del listado de elementos o la de detalles de un elemento.

```
{% block content %}

{# Content from child template #}

{% endblock content %}
```

Como muestra el comentario de la segunda línea, la plantilla 'hija' se insertará justo en ese bloque. Para que esto ocurra, estas plantillas 'hijas' deberán contener un bloque como el anterior ('block content'), y la siguiente linea:

```
{% extends "admin/base.html" %}
```

4.2.7.2. Beans

Este archivo contiene la parte del código que se mostrará, junto con el de base, cuando el usuario seleccione alguno de los links de la barra lateral.

El bloque de código esta formado primeramente por un título, el del modelo en cuestión, y por debajo su descripción, un pequeño formulario que permite buscar un elemento en concreto y un botón idéntico al símbolo '+' de la barra lateral pero con otro aspecto, que permite añadir un elemento a la lista.

Después, con las etiquetas propias de HTML que permiten crear una tabla ('th', 'tr', etc.) se crea la primera fila con los encabezados: ID, que será simplemente un número identificativo de cada elemento, el nombre del elemento, sus tres primeras propiedades y dos botones que permitrán editar o borrar. Cabe destacar que a todos estos datos se accede gracias a los modelos explicados previamente. Por ejemplo, en el caso de las propiedades se hace referencia a ese array de elementos que contenían los modelos.

Una vez que se tiene el encabezado, se recorrerá con un bucle todos los elementos existentes y por cada uno se creará una nueva fila en la tabla, mostrando los datos que se han descrito en la parte del encabezado.

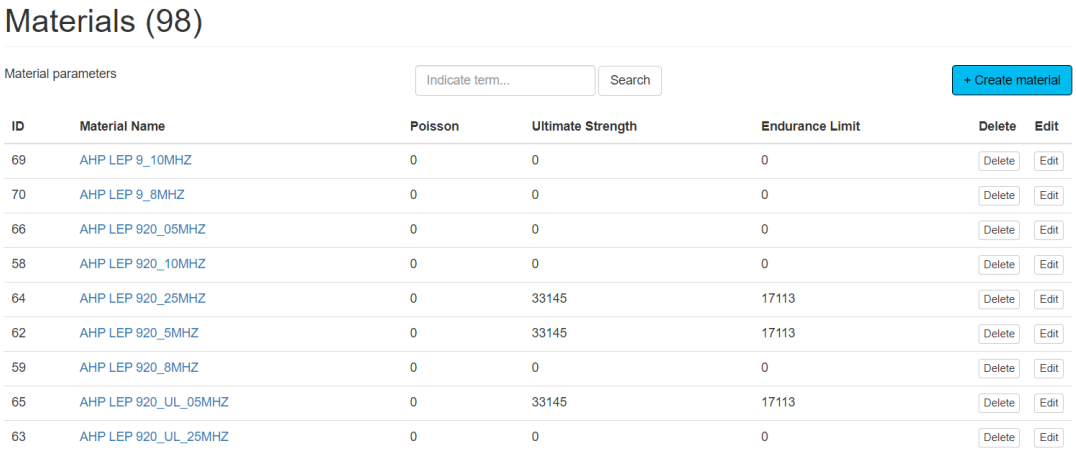


Figura 4.22: Ejemplo de la plantilla Beans

4.2.7.3. Bean

En este archivo se realiza el desarrollo de la pantalla que sirve para mostrar el detalle de un elemento en concreto o para añadir un elemento ya que la pantalla es muy similar. Se utiliza la misma forma explicada en el archivo anterior para que funcione como 'hija' de la plantilla principal.

Para este caso tendremos un título que será el nombre del elemento, o si estamos añadiendo uno, la palabra add, seguido de la fecha en la que ha sido creado, o modificado. Después, se mostrarán todos los campos que se han descrito en el apartado modelos, que son concretamente cada elemento del array de propiedades mencionado también en el archivo anterior. Estos campos se dispondrán en pantalla según su propiedad 'column'.

En la parte inferior, se dispondrán tres botones para borrar, guardar y vover a la pantalla anterior (Beans) o simplemente guardar.

Cabe destacar, que en el caso de los modelos 'Simulated test' y 'Test plan' era necesario añadir un nuevo botón de enviar, ya que en estás pantalla el usuario tiene que ser capaz de enviar a la herramienta computacional del proyecto BEETool los tests que desea que sean simulados. Para ello, se ha seguido el campo de estado del test, que podía tener los valores pendiente, calculando y finalizado. Según este valor, se mostrará el nuevo botón o no para evitar simular dos veces el mismo test, o un botón que permita recargar la página si se está calculando el test.

El funcionamiento de este botón es distinto al de los demás, ya que al tener que comunicarse con la herramienta computacional para enviarle el test y pasarle los datos, se utilizó el siguiente código JavaScript:

```
$("[name='enviar']").click(function(e){
        e.preventDefault();
        $.ajax({
            type: "POST",
            url: 'http://localhost:800/
            public/public_api/enviar_test_simulado',
            data: {
```

```
                id:"{{ bean.id }}",
                type:"{{ beantype }}"
            },
            dataType: 'json',
            success: function(e){
                console.log(e);
                setTimeout(function()
                {location.reload();},1000);
            },
            error: function(e){
                console.log(e);
                alert('fallo');
            }
        });
    });
```

Simplificando el código, lo que hace es una petición Ajax al servidor pasandole la id del test a simular y su tipo, 'simulatedtest' o 'testplan'. Una petición Ajax (Asynchronous JavaScript and XML) es una técnica de programación que permite enviar y recibir datos de forma asíncrona entre un servidor y una página web, sin tener que recargar la página completa.

En una petición Ajax, se envía una solicitud al servidor a través de JavaScript, en este caso una petición POST, ya que se deben enviar datos. El servidor procesará la solicitud y enviará una respuesta en formato JSON. Este procesamiento se realiza en el archivo 'public api' explicado en detalle en el siguiente apartado.

Una vez que la respuesta se ha recibido, la página web realizará una recarga.

En lo relacionado con las peticiones, en este proyecto se utilizan cuatro tipos de solicitudes HTTP (GET, POST, PUT, DELETE) que representan las operaciones básicas CRUD (Crear, Leer, Actualizar, Eliminar). La solicitud GET se utiliza para recuperar datos del servidor. Esta solicitud no realiza cambios en el servidor y solo obtiene datos.

La solicitud POST se utiliza para enviar datos al servidor para su procesamiento o almacenamiento. Al realizar una solicitud POST, los parámetros se envían en el cuerpo de la solicitud y no son visibles en la URL.

La solicitud PUT se utiliza para enviar datos al servidor para crear o actualizar un recurso específico. A diferencia de POST, la solicitud PUT generalmente se utiliza para actualizar un recurso existente identificado por su URL. El cuerpo de la solicitud contiene los datos que se deben agregar o modificar en el recurso.

La solicitud DELETE se utiliza para eliminar un recurso específico en el servidor. Al realizar una solicitud DELETE, se identifica el recurso a eliminar mediante su URL.

Estas operaciones serán importantes en la parte de rutas ya que, dependiendo del tipo de petición, se mostrará en pantalla una plantilla u otra.

4.2.7.4. Index.html

Además de estos archivos, era necesaria una página inicial, a modo de 'home', en la que, también siendo hija de 'base' para contener las barras laterales y superiores, se decidió que mostrase en pequeñas tablas, un pequeño resumen de cada modelo.

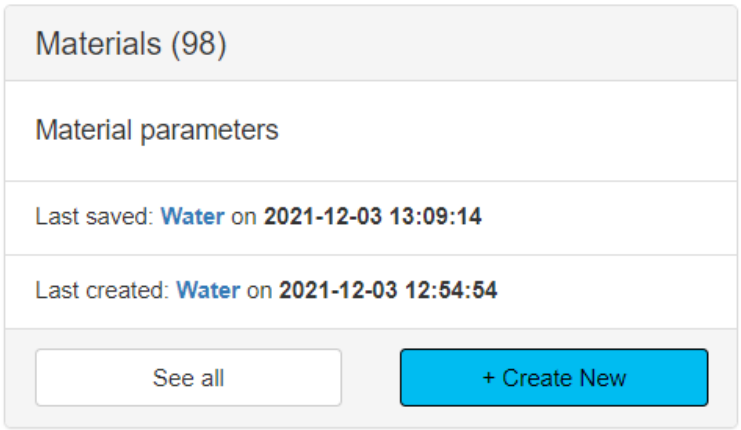


Figura 4.23: Resumen del modelo que se muestra en la pantalla index.php

4.2.8. Carpeta Routes

Esta carpeta contiene los ficheros que gestionan toda la parte de rutas de la aplicación, es decir, qué sucede cuando el usuario pulse en un botón o en cualquier link, qué página se muestra, qué variables se deben manipular, etc.

4.2.8.1. Functions

Antes de describir las rutas posibles del panel es importante describir el archivo functions.php.

En el se han creado una serie de funciones necesarias para el correcto funcionamiento. De esta forma, se agrupan en un mismo archivo todos los bloques de código que podremos reutilizar, mejorando la eficiencia en el desarrollo.

Algunas de las funciones más destacadas son 'getBeantypes', que devuelve un array con el nombre de todos los modelos que se han creado, 'getBeanTitles' o 'getBeanPlurals' que realizan lo mismo pero devolviendo los títulos y los plurales de cada modelo. En el caso de los nombres en plural, son utilizados en pantalla en la barra lateral o en los títulos de las pantallas ya que normalmente se tendrá mas de un elemento de cada modelo.

Por otro lado encontramos las funciones 'setResponse' y 'enviar post' que serán utilizadas en las comunicaciones con la herramienta computacional de BEETool, la primera devolverá una respuesta a cualquier petición que se haga e la aplicación web, y la segunda enviará una petición de tipo POST a la URL de la herramienta computacional utilizando la biblioteca Curl. En ella se enviarán todos los datos del test o de los tests que se quieran simular.

Curl es una biblioteca muy utilizada en el desarrollo de aplicaciones web que permite realizar solicitudes HTTP entre otras muchas cosas.

Por último destacar la función 'beetool-log' que sirvió para poder depurar todo el código. Esta función escribe en un archivo de texto plano los posibles errores que se tenían a la hora de desarrollar, o en el caso de que no hubiera errores se utilizaba para acceder al contenido de diferentes variables y ver en detalle como se modificaban en cada momento de ejecución.

4.2.8.2. Admin

En la siguiente figura 4.24 se muestra un esquema de las rutas a las que se iría accediendo conforme el usuario navegue por el panel.

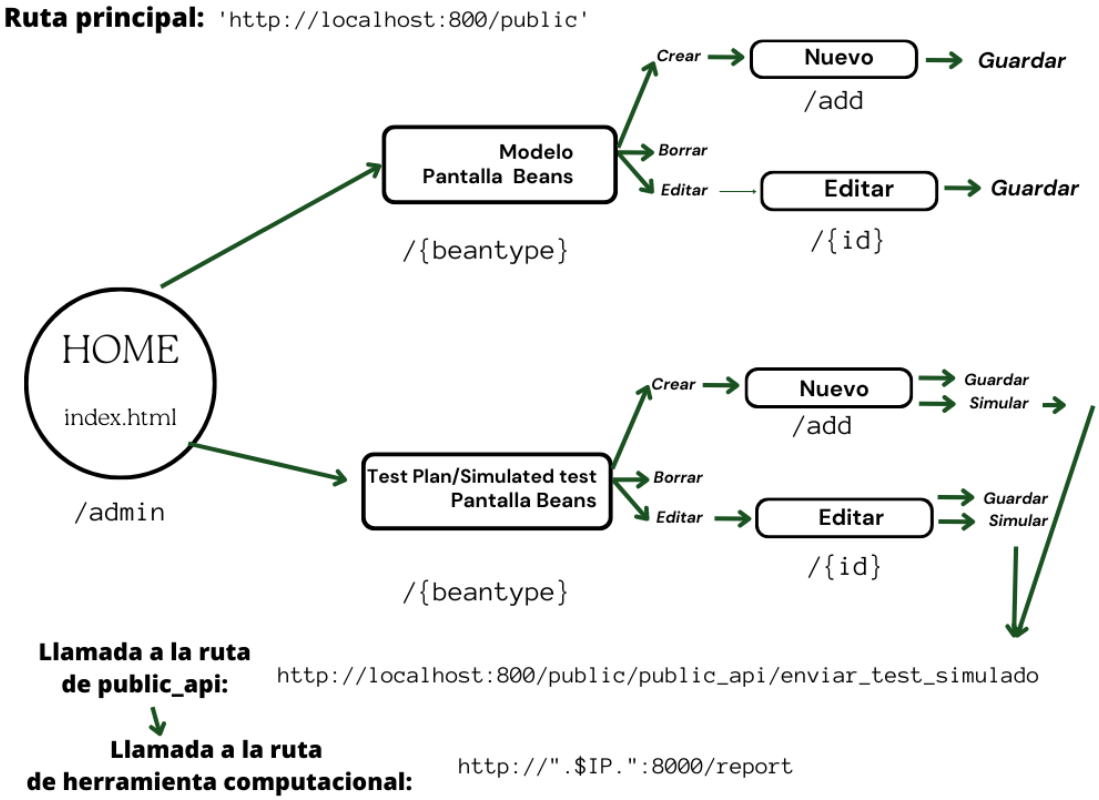


Figura 4.24: Diagrama de flujo de las rutas del panel

Como se puede ver, este archivo está dividido en una serie de funciones según la ruta a la que se acceda desde el panel. Todas ellas se encuentran dentro de la ruta `'http://localhost:800/public/admin'`. En los casos de las rutas entre llaves, se hace referencia a una variable, como puede ser el tipo del modelo (materials, climatic data, turbine data), o a la ID de un elemento en especifico.

Para acceder a una determinada ruta, el usuario simplemente tiene que hacer click sobre los links y botones del panel. Las plantillas HTML serán las encargadas de que cuando esto pase, se acceda a la ruta correcta, descrita en este apartado.

En todos los casos las funciones tienen la misma estructura: se accede a los datos mediante

las funciones explicadas previamente, se modifican las variables según la ruta a la que se haya accedido y con la siguiente línea se muestra en pantalla la plantilla que sea necesaria:

```
return $this->view->render(
$response, 'admin/beans.html', $data
);
```

En este caso se mostrará en pantalla la plantilla 'Beans' y se le pasará la variable 'data' que contiene todo lo necesario para que la plantilla, como se ha visto anteriormente, acceda al modelo y a todos sus datos. De esta forma mostrarlos en pantalla correctamente.

Por tanto, este archivo contiene todas las funciones para todas las rutas posibles del panel, mostradas en la figura anterior, para poder mostrar la plantilla adecuada en cada caso.

4.2.8.3. Public API

Este archivo tiene una estructura parecida al anterior pero en este caso contiene dos rutas que no mostrarán ninguna plantilla.

La primera, '/enviar-test-simulado', es la ruta que se utiliza cuando el usuario pulsa el botón de enviar una test. Como se ha explicado anteriormente, este botón realizaba una petición POST y enviaba la ID del test y su tipo (simulatedtest o testplan).

Con estos dos datos, esta función accede a la base de datos y genera un archivo json que contendrá todos los datos que requiere la simulación de un test, desde las propiedades de los materiales, las condiciones climaticas, hasta las características de la turbina o todos los comentarios que se hayan podido añadir.

Este archivo en formato json se envía con la función 'enviar-post' descrita en el apartado anterior y en caso de que la respuesta sea buena, se cambiará el estado del test de 'pendiente' a 'calculando'.

En esta función se accede a la base de datos como se ha explicado, pero cabe destacar que se realiza de forma sencilla gracias al uso de ReadBean. Este framework permite el uso de comando sencillos que realizan consultas en la base de datos como los siguientes:

```
$testplan = R::findOne('testplan', 'id =?',[$id]);
```

```
$testplan -> estado = 'Calculating';
R::store($testplan);
```

La primera linea busca en la base de datos la tabla testplan, y en ella, la fila del elemento cuya ID sea igual a la ID que se ha enviado al pulsar el botón de simular.

En las dos últimas cambia la propiedad estado de la tabla testplan a 'Calculando' y con el comando R::store lo guarda en la base de datos.

Por otro lado, en este archivo encontramos otra función que recibe los datos que se envían desde la herramienta computacional de BEETool cuando se ha terminado la simulación. Si todo ha funcionabo de forma correcta el estado del test se cambiará a finalizado.

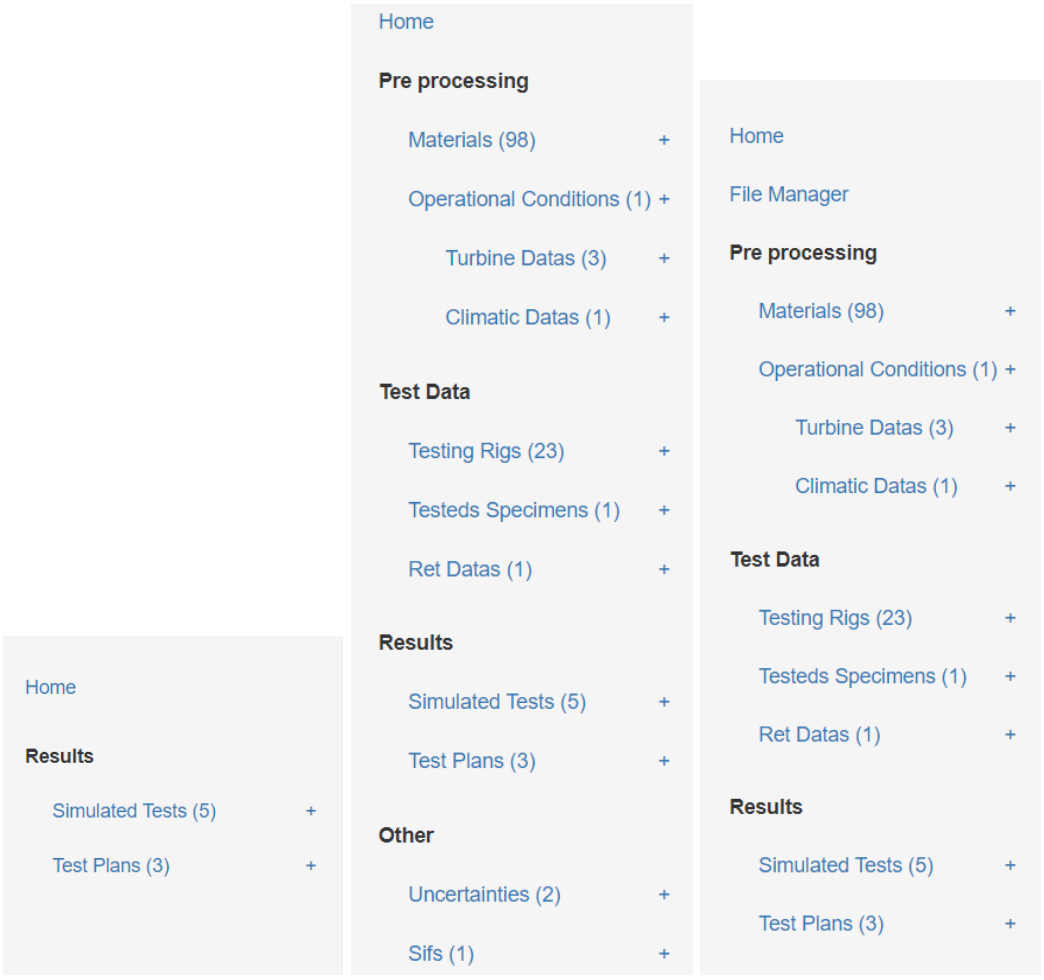
4.2.9. Últimas funcionalidades

Una vez desarrollado todo el panel, se nos pidió desde el proyecto que para acceder al panel deberían de existir tres permisos diferentes: un 'admin' que sea capaz de visualizar, editar y modificar todos los archivos, un 'editor' que pueda acceder a todo lo descrito hasta ahora en esta memoria, y un posible cliente de una empresa que pueda tener el panel, al que solo se le permita acceder a la parte de resultados de las simulaciones.

Para ello, se crearon las variables globales en el archivo config.php con los diferentes permisos y la función autenticar que devuelve el usuario que ha accedido al panel. Por otro lado en la plantilla principal, mediante bloques condicionales se modificó la barra de navegación lateral, mostrando únicamente lo que se había solicitado según el permiso del usuario.

Con esto, no se conseguía del todo las restricciones de cada permiso, ya que el usuario podía acceder a las diferentes pantallas escribiendo la url en el navegador. Para solucionar este problema, en el archivo admin.php, que gestiona todas las rutas, se hizo uso de la función autenticar menionada anteriormente.

En cada función, es decir en cada ruta posible del panel, se comprueba que el usuario tenga los permisos adecuados, y dependiendo de esto, mostrar una plantilla u otra.



(a) Permiso Cliente (b) Permiso editor (c) Permiso admin

Figura 4.25: Diferencias de los diferentes permisos del panel

También fue necesario crear una página de inicio nueva, para el caso de que el usuario fuese un cliente, ya que en caso contrario podría acceder al resumen de todos los datos que se muestran en la pantalla de inicio que se creó en un principio.

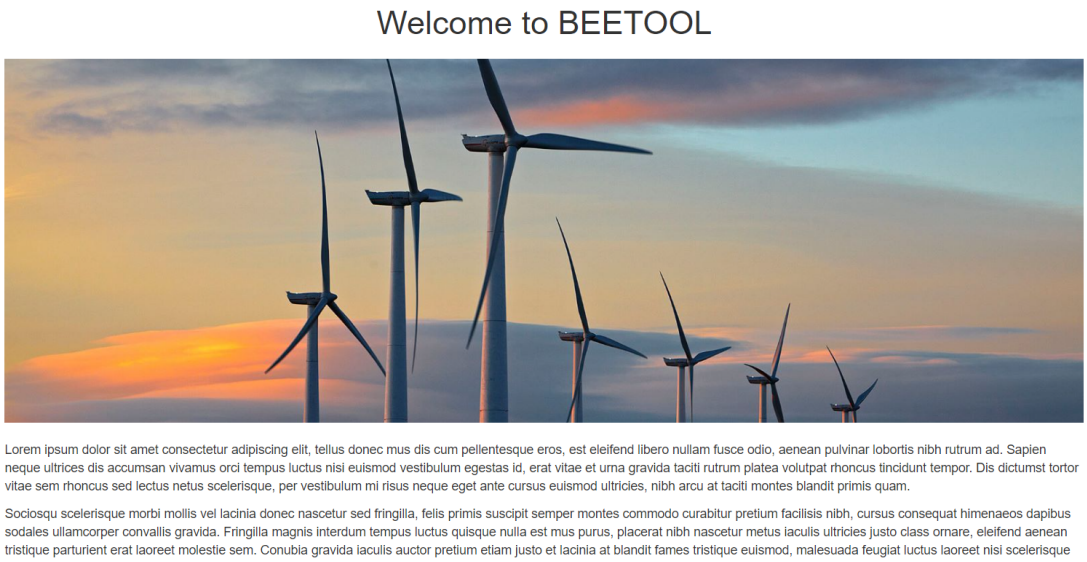


Figura 4.26: Inicio del permiso cliente

Destacar también, que el usuario con permiso de ’admin’, tiene acceso desde la web a todos los archivos mediante un gestor de ficheros llamado ’Filemanager’. Este gestor de archivos fue proporcionado por el proyecto BEETool y se puede ver en la figura 4.27

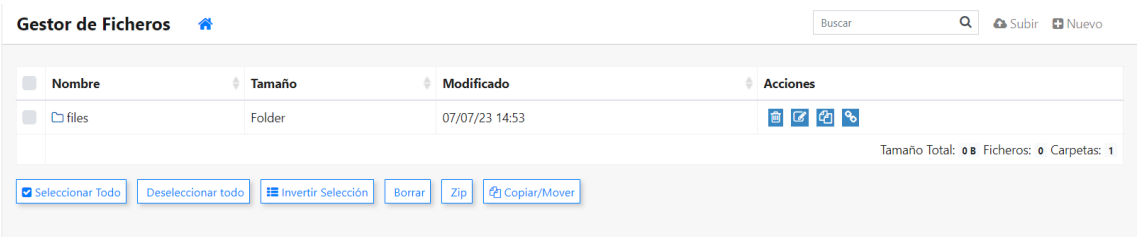


Figura 4.27: Pantalla del gestor de archivos

Capítulo 5

Resultados

Por tanto, una vez descrita la parte de desarrollo y comprobado que todo funciona correctamente, se realizó una migración a un servidor web real. Este proceso está descrito en la otra parte del TFG colaborativo.

Como resultado del proyecto se obtuvo una aplicación web, funcionando en un servidor web real, en la que, después de loguearse, un posible usuario puede introducir diferentes datos. La aplicación web es capaz de recopilar todos estos datos, generar un archivo json con ellos, y enviarlo a un simulador. También es capaz de recibir los resultados y el informe que envíe el simulador y mostrarlos en pantalla.

En el apartado de Anexos se muestran todas las pantallas que componen la aplicación a modo de demostración.

Capítulo 6

Conclusiones y lineas futuras

En cuanto a las conclusiones a las que se ha llegado con el desarrollo del proyecto podemos hablar de que se han cumplido los objetivos que se propusieron desde el principio. Gracias a la aplicación web, el proyecto BEETool cuenta con una herramienta que a modo de interfaz de usuario, es capaz de comunicarse con el simulador que han desarrollado y complementarlo.

De esta forma, se ofrece una solución para resolver el problema del que se hablaba en la introducción de esta memoria, la degradación y el desgaste que se produce en las palas de las turbinas eólicas y que supone numerosas pérdidas en las empresas del sector de la producción de energía eólica.

Además, se ofrece una solución en forma de software, igual de válida que un simulador físico, pero con la ventaja de no tener que desplazar ningún elemento, con lo que esto supone.

Se puede hablar también de la relación del proyecto con los Objetivos de Desarrollo Sostenible [14], una serie de objetivos a nivel global que se adoptaron en el año 2015 como parte de una nueva agenda de desarrollo sostenible, con el fin de acabar con la pobreza y proteger el planeta en los siguientes quince años. Entre los objetivos se encuentran algunos como ’Energía Asequible y No contaminante’ o ’Ciudades y Comunidades Sostenibles’.

Este TFG, junto con el proyecto BEETool, impulsa la producción de energía eólica, un tipo de energía que no contamina, es decir, no libera emisiones de gases de efecto invernadero

ni residuos. Además, el uso de aerogeneradores en las ciudades hace que estas se vuelvan más sostnibles en cuanto a producción de energía se refiere.

Con todo esto, podemos hablar de que todo el proyecto se puede ofrecer a distintas empresas, realizando una copia de la aplicación web para cada una. Como consecuencia, en un futuro, cada empresa podrá disponer de su propia máquina en la que realizar pruebas y simulaciones con con sus propios materiales, condiciones, etc. pero de forma virtual.

Si cada empresa tiene su propia aplicación que le permite realizar sus propias simulaciones, se consigue también el factor de confidencialidad, indispensable también en la carrera por conseguir el mejor producto que ofrecer al mercado de la producción de energía eólica.

Bibliografía

[1] UCH CEU. *Innovación para mejorar la eficiencia de la energía eólica*. URL: https://medios.uchceu.es/actualidad-ceu/innovacion-para-mejorar-la-eficiencia-de-la-energia-eolica/.

[2] JSON Org. *JSON*. URL: https://www.json.org/json-es.html.

[3] Wikipedia. *Pair proggramming*. URL: https://en.wikipedia.org/wiki/Pair_programming#:~:text=Pair%20programming%20is%20a%20software,two%20programmers%20switch%20roles%20frequently..

[4] Jose Antonio Tellez. *Beetool$_p$hp*. URL: https://github.com/tellez11/beetool_php.

[5] Aurelio Pons. *Beetool*. URL: https://panel.beetool.es/admin..

[6] PHP. *¿Que es PHP?* URL: https://www.php.net/manual/es/intro-whatis.php.

[7] Slim Framework Team. *Slim*. URL: https://www.slimframework.com/.

[8] RedBeanPHP community. *RedBeanPHP*. URL: https://www.redbeanphp.com/index.php.

[9] Symfony y Twig. *Twig*. URL: https://twig.symfony.com/.

[10] Lútsen Stellingwerff. *Lagan*. URL: https://www.laganphp.com/#property-type-controllers.

[11] the Bootstrap team. *bootstrap*. URL: https://getbootstrap.com/.

[12] OpenJS Foundation y jQuery contributors. *jQuery*. URL: https://jquery.com/.

[13] Lútsen Stellingwerff. *Lagan*. URL: https://github.com/lutsen/lagan.

[14] Naciones Unidas. *Objetivos de desarrrollo sostenible*. URL: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/.

[15] Jose Antonio Tellez y Jose López. *Beetool*. URL: https://github.com/tellez11/beetool_php.

Parte II

Anexos

El código del proyecto en su totalidad se encuentra en el siguiente enlace [15]

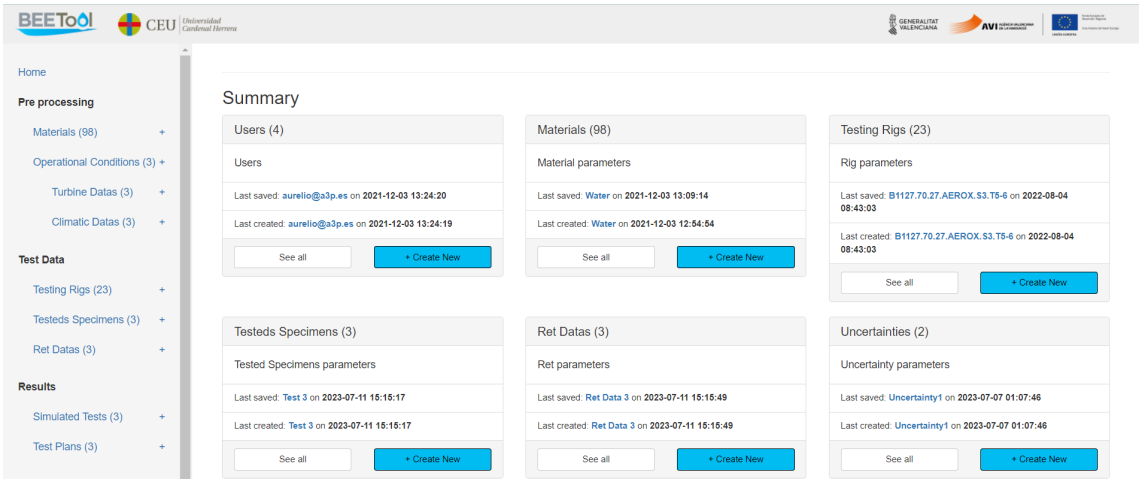


Figura 1: Pantalla Home

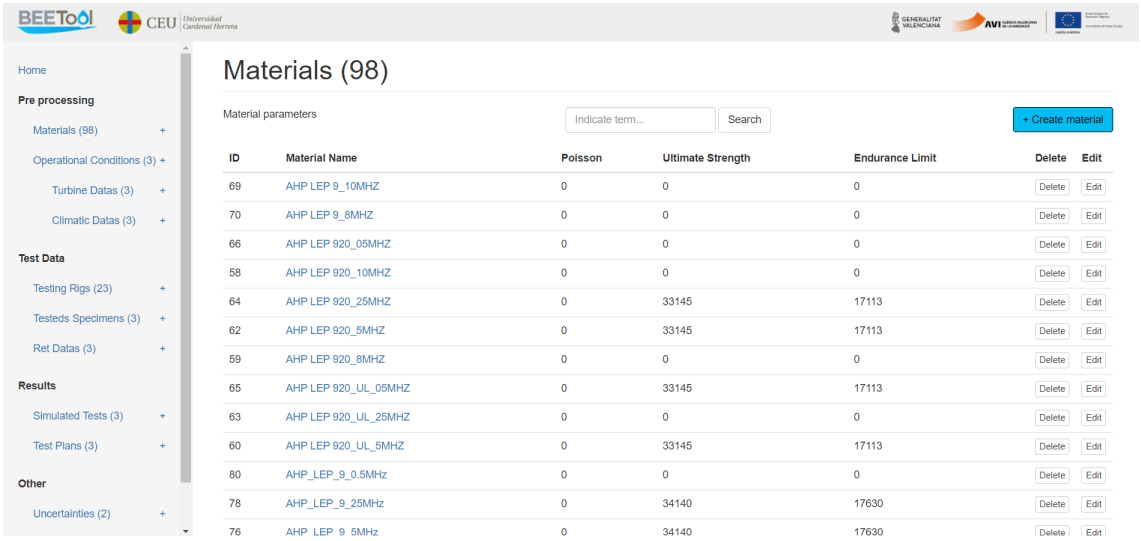


Figura 2: Pantalla listado Materiales

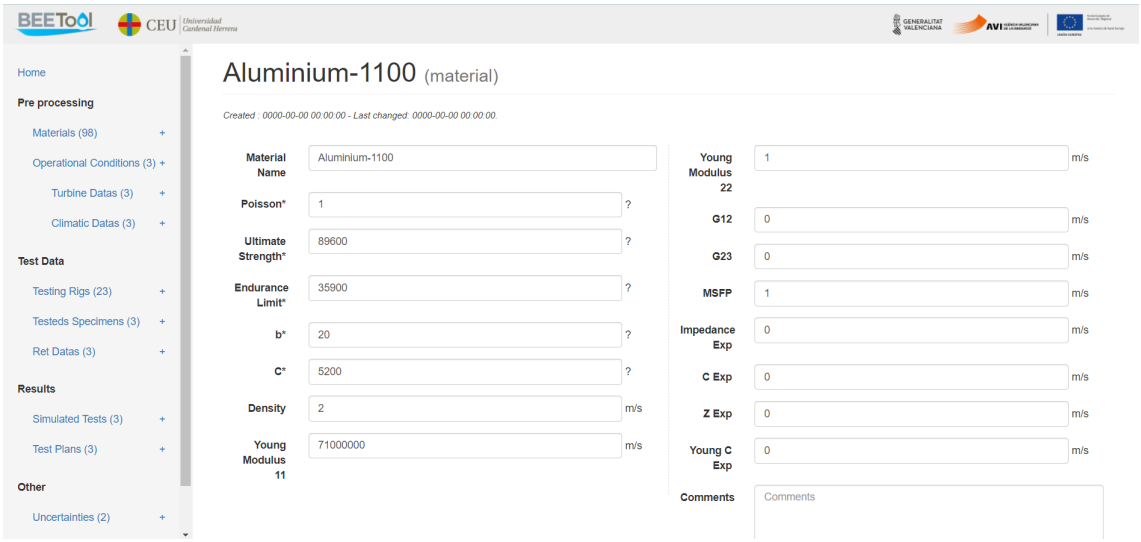


Figura 3: Pantalla campos de material

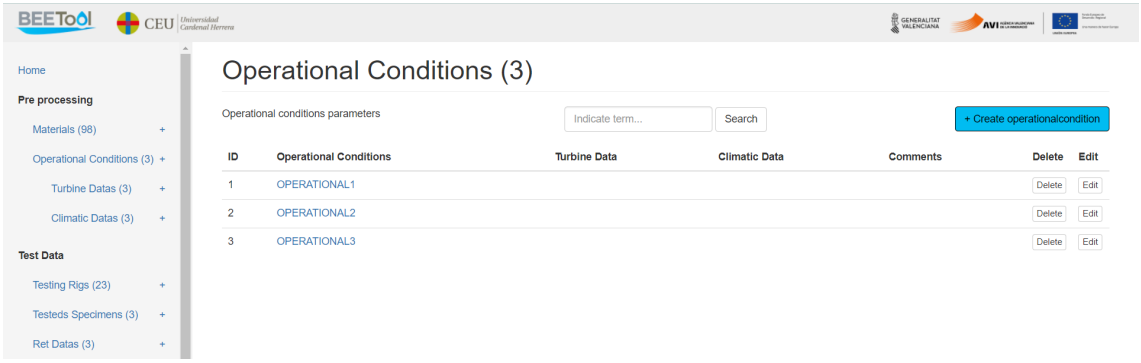


Figura 4: Pantalla listado Operational Conditions

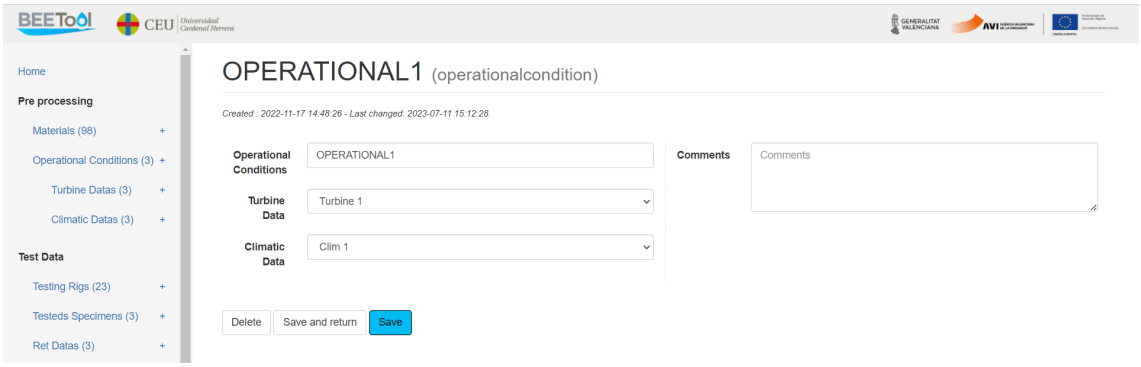


Figura 5: Pantalla campos de Operational Conditions

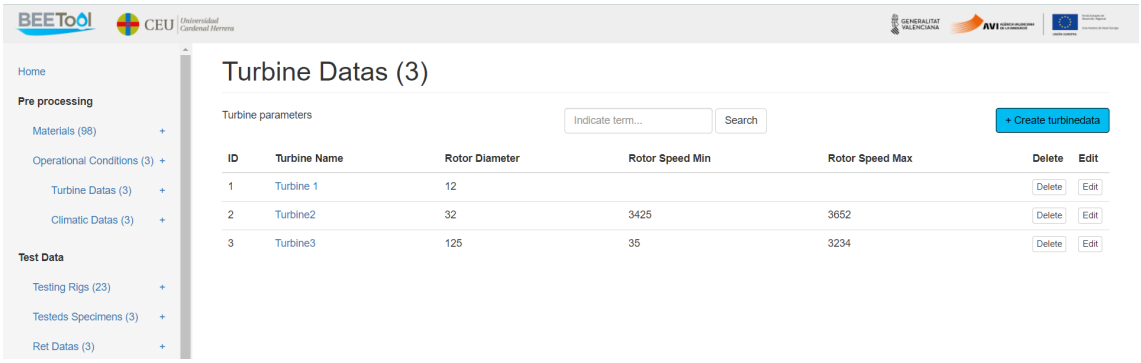


Figura 6: Pantalla listado Turbine Data

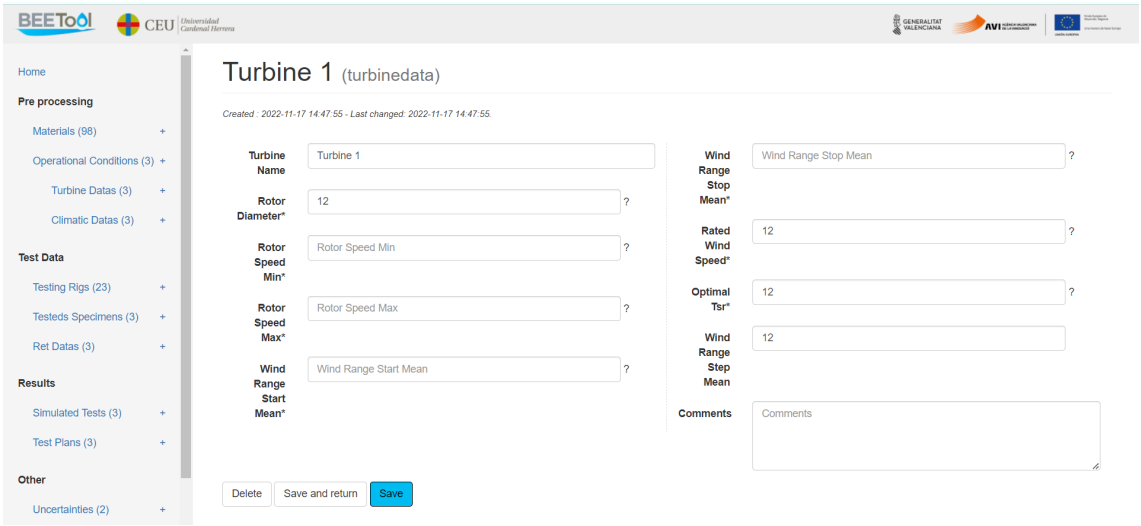


Figura 7: Pantalla campos Turbine Data

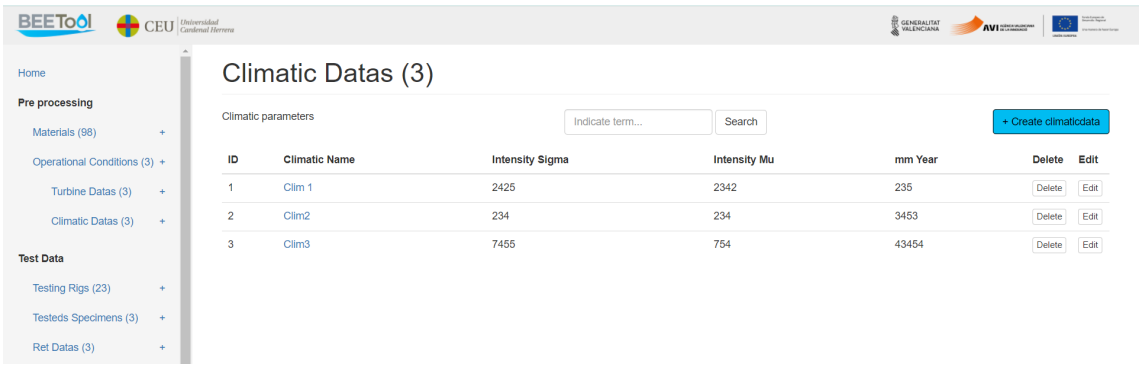


Figura 8: Pantalla listado Climatic Datas

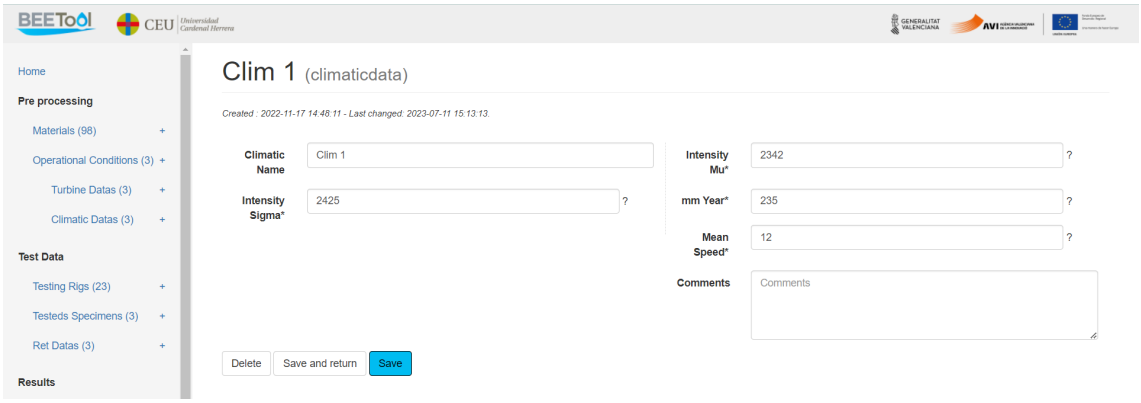


Figura 9: Pantalla campos Climatic Data

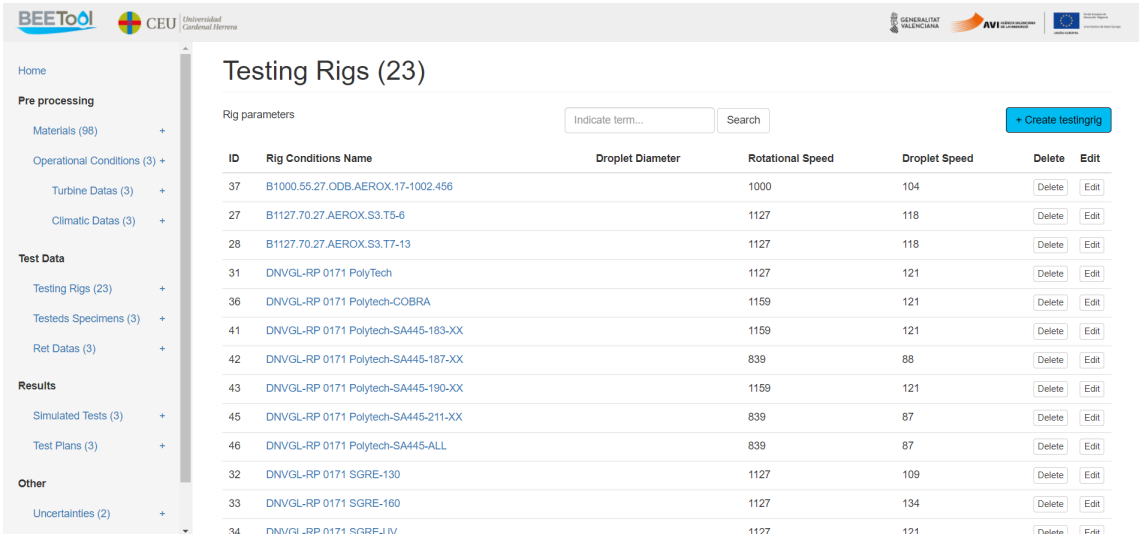


Figura 10: Pantalla listado Testing Rigs

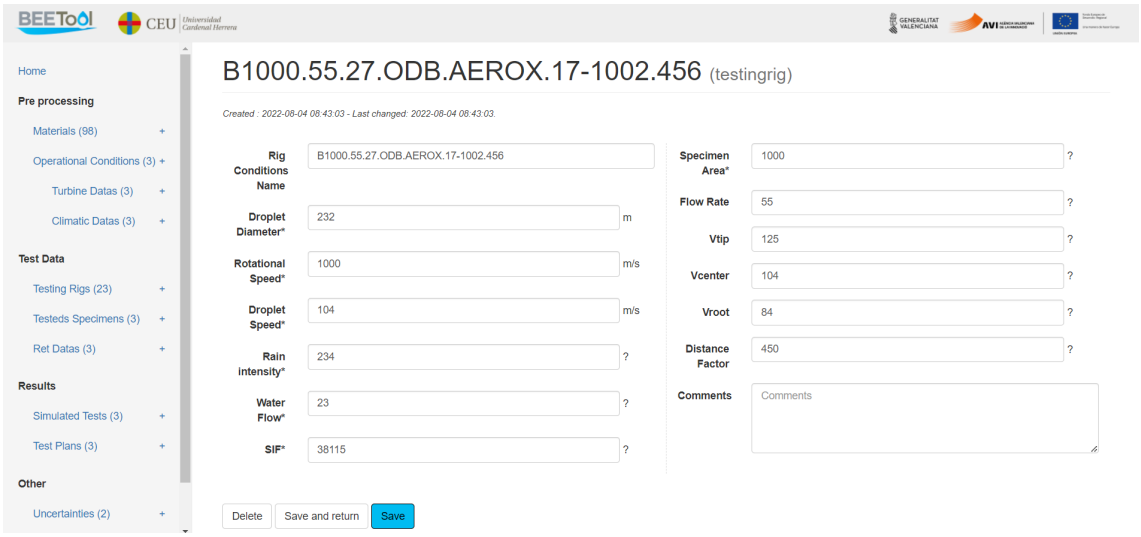


Figura 11: Pantalla campos Testing Rig

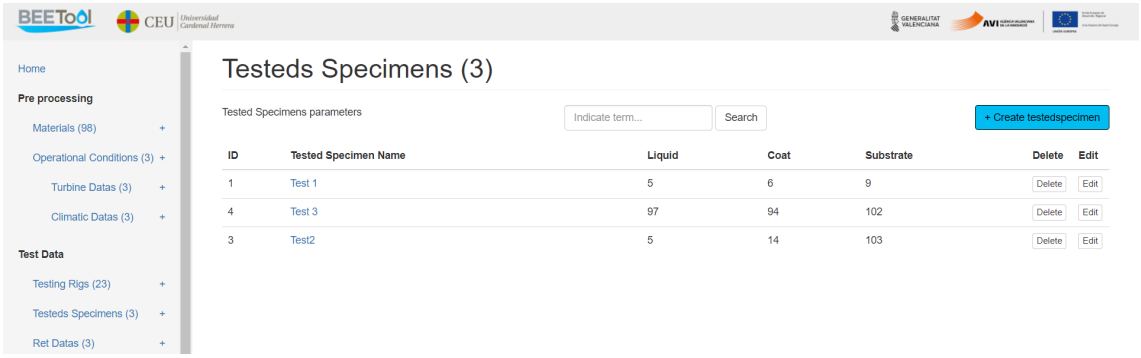


Figura 12: Pantalla listado Testeds Specimens

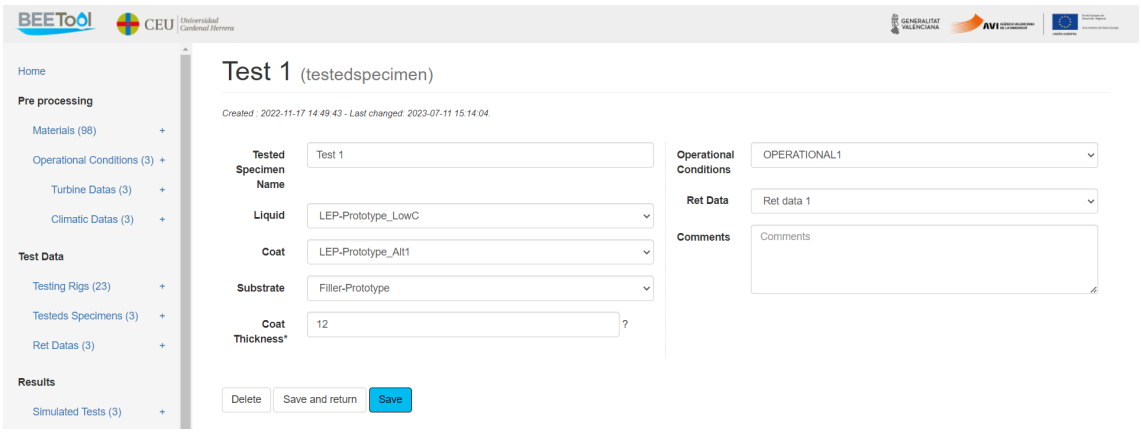


Figura 13: Pantalla campos Tested Specimen

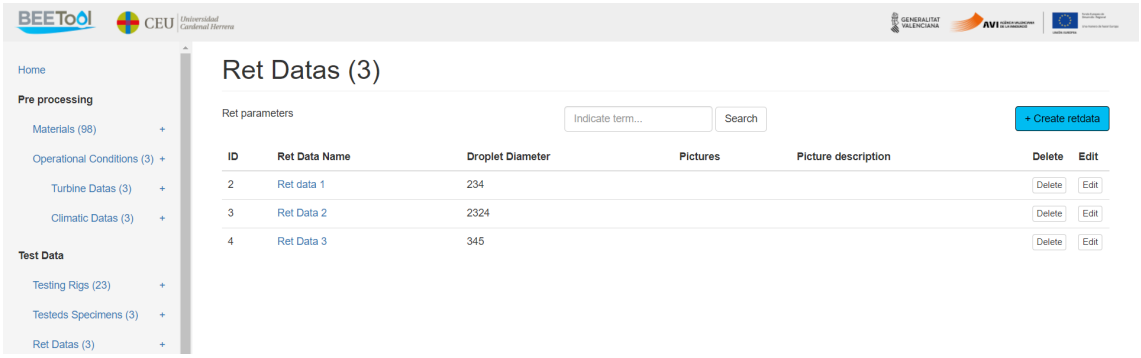


Figura 14: Pantalla listado Ret Datas

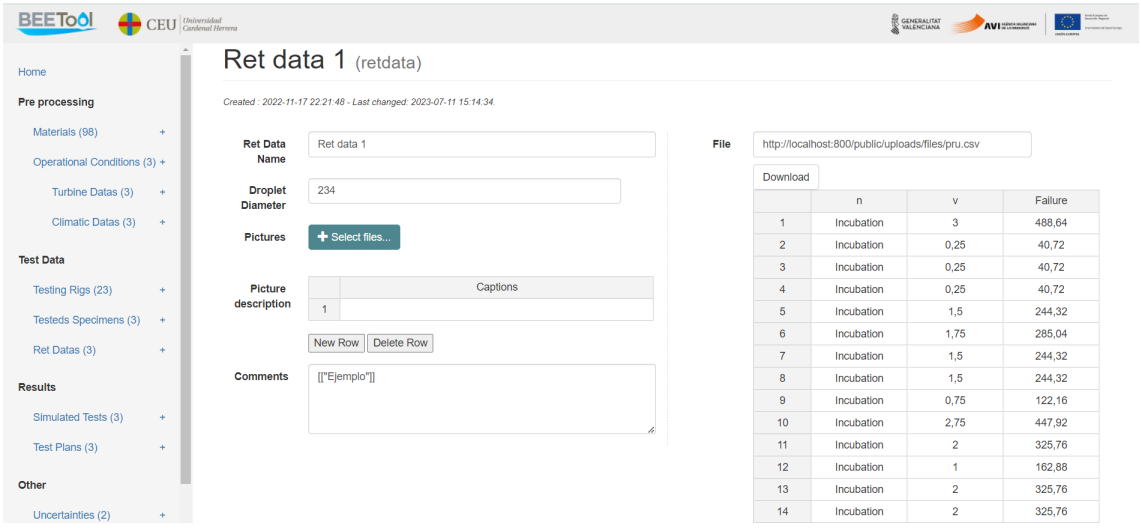


Figura 15: Pantalla campos Ret Data

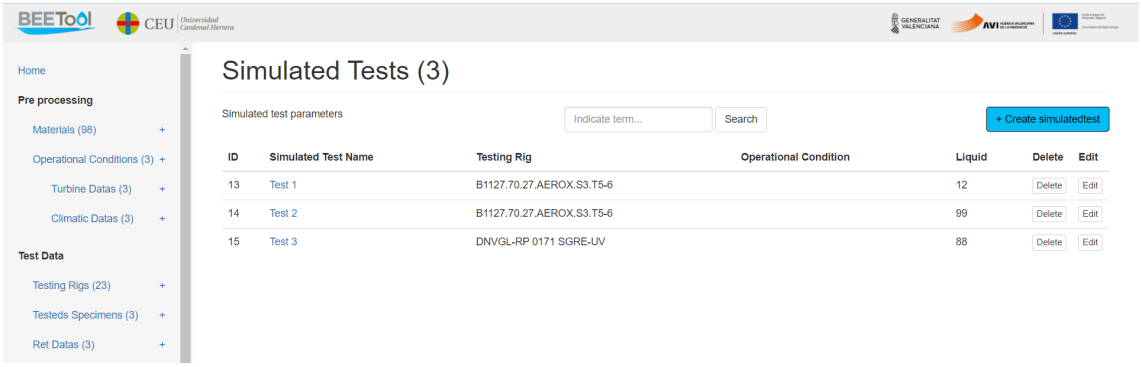


Figura 16: Pantalla listado Simulated Tests

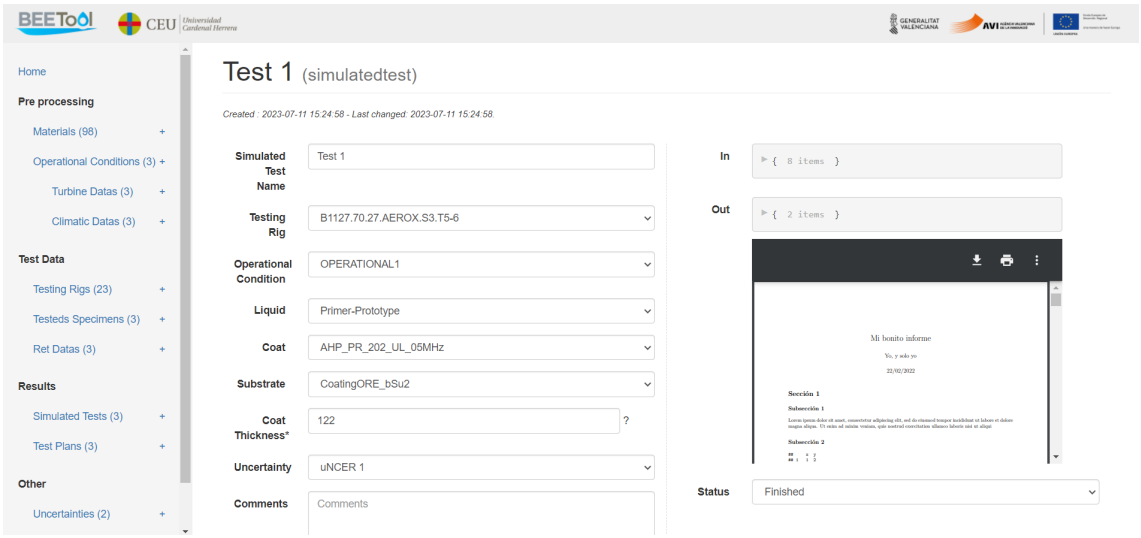


Figura 17: Pantalla campos Simulated Test

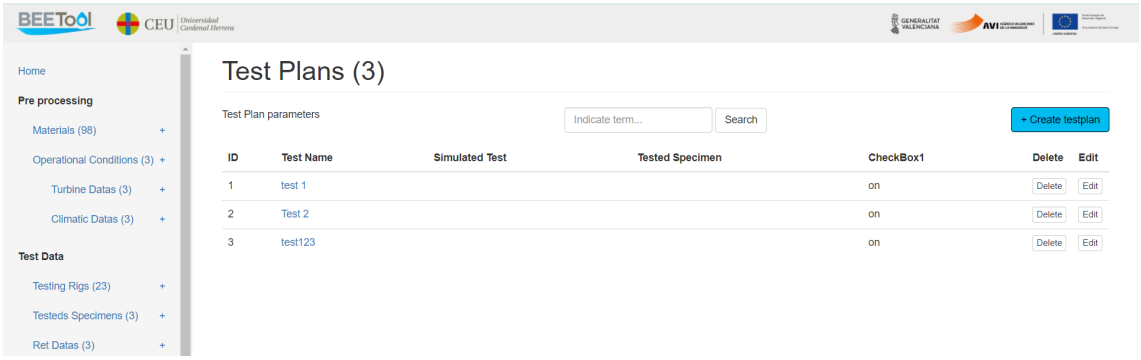


Figura 18: Pantalla listado Test plans

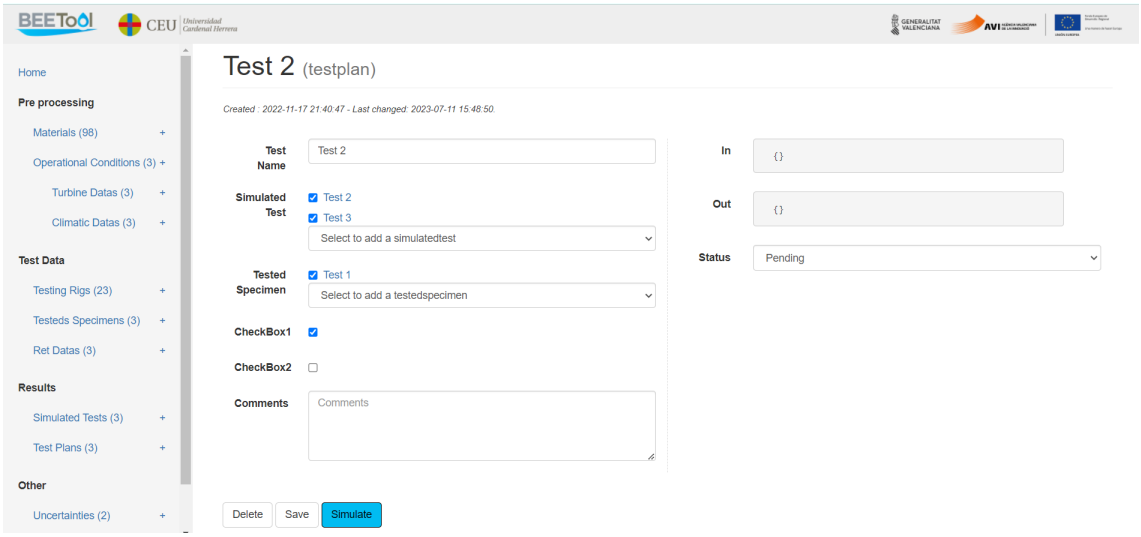


Figura 19: Pantalla campos Test plan