Grado en Ingeniería Informática

Trabajo Final de Grado

Herramienta de modelado y procesamiento gráfico de datos

Autor:
Oussama Alouat

Supervisor:
Jose Manuel Borrego
Tutor académico:
David Llorens Piñana

Fecha de lectura: 20 de julio de 2018
Curso académico 2017/2018

Resumen

Este documento explica el procedimiento seguido para desarrollar una herramienta de modelado y procesamiento gráfico de datos.

El proyecto se ha llevado a cabo mediante una metodología ágil, scrum.

Se trata de un sistema de arquitectura cliente/servidor, que ha sido implementado con JavaScript. Concretamente Vuejs en la parte *front-end* y Nodejs para la parte del *back-end*. Asimismo cuenta con dos bases de datos para albergar la información. La primera de ellas es MongoDB, que sirve para guardar la estructura de los datos. La segunda es PostgreSQL, que almacena los datos. Esta última conecta con el visualizador de los datos (Superset), el cual lee los datos y los representa en gráficas.

Palabras clave

Herramienta, procesamiento gráfico, JavaScript, Nodejs, Vuejs, Axios, Scrum, metodología ágil,dashboard, API, HTTP

Keywords

Tool,graphic processing, JavaScript, Nodejs, Vuejs, Axios, Scrum, Agile methodology, dashboard, API, HTTP

Índice general

Índice de figuras

Índice de cuadros

Listings

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

En este apartado se describe la empresa donde se ha realizado el proyecto, el contexto de dicho proyecto y la motivación que impulsó su realización

1.1.1. Descripción de la empresa

La empresa donde se está realizando el presente proyecto es 720tec, está ubicada en el edificio CEEI Castellón. Se trata de una empresa de carácter tecnológico que inició su actividad empresarial en diciembre de 2014. Por muy joven que parezca la empresa, está respaldada por un gran equipo de profesionales.

Su principal línea de negocio son los servicios tecnológicos, outsourcing, consultoría estratégica y diseño organizativo. Los servicios ofrecidos son variados, dependiendo del ámbito tecnológico. Los cuales son: diseño, provisión y mantenimiento de sistemas informáticos; desarrollo de proyectos especiales y soluciones para entornos específicos.

Actualmente el ámbito que más cambios sufre y con diferencia, es el referente a la industria, ya sea porque aparecen nuevas tecnologías o porque las actuales se mejoran. Por esta razón, 720tec es pionera creando soluciones para este entorno. Las soluciones que realiza la empresa van desde llevar el mantenimiento de equipos industriales, hasta desarrollar hardware especifico para dar solución a algún problema. Cosa que implica la implementación del software correspondiente. En numerosas ocasiones, la empresa solo se encarga de programar el software para sacarle mejor partido al aparato industrial.

1.1.2. Motivación del proyecto

Cuando se desea realizar alguna acción de manera repetitiva, lo más eficiente es automatizar esa tarea si existe la posibilidad. Para las empresas es muy importante tener tareas automatizadas, ya que supone un ahorro porque no se le está pagando a una persona para que haga la misma acción de manera reiterada, pero lo más importante es que supone una seguridad debido a que una persona puede cometer fallos, un sistema automatizado es muy complicado que los cometa.

La empresa 720tec, quiere mostrar gráficamente información contenida en bases de datos (dashboards) a través de una serie de ficheros. Esto es, el sistema tiene que permitir el almacenamiento de datos, el modelado de los mismos, su visualización y su análisis, mediante una interfaz que ofrezca una serie de elementos gráficos para ayudar con la interpretación de los mismos.

Esta necesidad surge porque la empresa quiere realizar análisis periódicos sobre las llamadas que recibe en una centralita de llamadas, 3cx, para conocer las estadísticas de cuántas llamadas recibe, cuántas ha atendido, cuántas no han podido ser atendidas, etc.

Asimismo la empresa quiere en un futuro extraer este producto como un servicio para sus clientes.

1.2. Objetivos del proyecto

El objetivo general del proyecto actual es la creación de una herramienta que permita procesar una serie de datos y representarlos gráficamente. Pero además tiene que permitir la automatización de este proceso.

Este objetivo general puede ser desglosado en:

- Recolección de datos: La herramienta debe de ser capaz de importar los datos desde diferentes fuentes de la empresa.
- Modelado de datos: La herramienta ha de ser capaz de modelar los datos según decida el usuario. Esto es, el usuario tiene que tener toda libertad para modelar los datos según le convenga.
- La herramienta debe de permitir la automatización de todo el proceso de modelado y almacenamiento de los datos.

La representación de datos se realizará con una tecnología, la cual se elegirá al inicio del proyecto. Dicha herramienta será configurada para que al leer una serie de datos pueda representarlos en un conjunto de dashboards con gráficas. Las cuales serán configurados por el equipo que desarrolla el proyecto.

La tecnología deberá de permitir realizar filtrados de los datos. Los menús de filtrado serán ajustados por el encargado de crear el dashboard.

Esto le permite al usuario visualizar los datos que ha introducido en forma de gráficas, sacar sus análisis y poder tomar decisiones en base a estos.

1.3. Estructura de la memoria

Esta memoria recoge la documentación del desarrollo del proyecto. La estructura de esta, comienza con el capítulo 2, dónde se realiza una descripción del proyecto y de las tecnologías empleadas para implementarlo. A continuación, en el capítulo 3, se describe la metodología empleada y la planificación realizada para llevarlo a cabo, también se desglosan los recursos y los costes que conllevan realizar el proyecto. Seguidamente, en el capítulo 4, se realiza un análisis del sistema y se describe el diseño de la arquitectura empleada por el sistema y el de las interfaces que conforman la herramienta. Por consiguiente, en el capítulo 5 se ha detallado la implementación de las funcionalidades más relevantes para la aplicación y las pruebas realizadas para comprobar el correcto funcionamiento de las mismas. Finalmente el capítulo 6 contiene las conclusiones del proyecto, esto es una valoración a cerca de los objetivos alcanzados, posibles extensiones para el futuro y una breve opinión personal sobre la experiencia de realizar un proyecto de estas características.

Capítulo 2

Descripción del proyecto

Según se ha comentado en el capítulo 1, la empresa 720tec necesita desarrollar una aplicación que permita la lectura de una serie de datos, el procesamiento y filtrado de los mismos y finalmente, su visualización. La aplicación tiene un fin general y se demostrará utilizandola para visualizar una centralita de llamadas, 3CX.

Inicialmente se ha tenido que realizar un estudio a cerca de la tecnología que se va ha emplear como tecnología base del proyecto, tal y como se puede apreciar en la sección 2.1, Estudio del estado del arte de las tecnologías

El presente proyecto se puede dividir en las siguientes partes :

Lectura y tratado de los datos: El sistema lee un conjunto de datos, como se puede ver en la Figura 2.1, que se le transmiten a través de un fichero con extensión csv.

Una vez leídos los datos, el sistema le permite al usuario decidir cómo los desea separar. Esto es así porque no todos los ficheros csv se separan igual, los hay que se separan con "," otros con ";" , etc.

Como en la mayoría de de ficheros csv en las primeras líneas contienen información no relevante del contenido. El sistema ofrece la posibilidad de realizar una primera limpieza de los valores ya separados, es decir, permite seleccionar y eliminar las filas que no se quieran tener entre los datos, según se muestra en la Figura 2.2.

Por consiguiente, como se observa en la Figura 2.3. El sistema permite decidir de qué tipo son los datos conservados tras la primera limpieza: texto, numéricos (números con decimales, números enteros), temporales (horas, minutos, segundos) o fechas (fechas simples, fechas con precisión horaria).

Asimismo la herramienta deja aplicar distintas acciones sobre los datos leídos. Las cuales pueden ser:

1. Eliminar caracteres de un determinado dato: Una vez que se ha seleccionado el dato sobre el que se desea aplicar esta acción, el sistema permite realizar la eliminación de tres formas:
   a) Indicando las posiciones inicio y fin.
   b) Indicando un carácter a partir del que empezar a borrar y otro hasta el que llegar.
   c) La combinación de las opciones anteriores.
2. Extraer caracteres de un dato seleccionado: El procedimiento es el mismo que el descrito en el apartado anterior.
3. Reemplazar una cadena de caracteres de un dato concreto: El procedimiento inicial, de encontrar el dato a reemplazar, es el mismo que se ha descrito en el punto primero. Tras eso, el usuario introduce una palabra o un carácter o incluso un conjunto de caracteres. Los caracteres seleccionados son eliminados y en su lugar se introducen los dados por el usuario.

En la Figura 2.4 se puede ver el siguiente ejemplo. Dado el dato "D-Alex:964255047 (964255047)", aplica la acción de reemplazar desde el carácter "D" hasta el carácter ":" ambos inclusive, por la palabra "Ejemplo" y se obtiene como resultado "Ejemplo 964255047 (964255047)".

No se ha mencionado anteriormente, pero los datos se leen por líneas. Cada línea es almacenada en un array y cada posición de ese array representa una columna de esa línea. Cada columna está nombrada por su correspondiente de la cabecera que el usuario ha seleccionado.

Una de las razones por las que se ha decidido realizar esto de esta manera, es porque, al aplicar un formato de texto sobre la columna 1, la columna 1 de todas las filas leídas será de tipo texto.

Del mismo modo, al aplicar una acción sobre cualquier dato, la acción se aplica a todos los datos que tengan la misma columna que el dato inicial.

Almacenamiento de los datos: El encargado de recoger los datos que ya han sido procesados y almacenarlos es un servidor. La tecnología empleada para desarrollarlo es Nodejs, debido a que en la empresa se emplea esta tecnología.

El servidor publica una API para que el cliente pueda comunicarse con él y transmitir los datos que desea almacenar. Posteriormente el servidor se conecta a la base de datos y los almacena.

El almacenamiento se divide en dos servicios. Por una parte se almacenan los datos que han sido tratados, y por otra, los componentes generados durante el proceso de tratamiento.

Para realizar el almacenamiento de los datos tratados inicialmente se planteó emplear una base de datos MySQL. No obstante, la tecnología que se ha seleccionado para la visualización de los datos presentaba problemas de conectividad con este tipo de bases de datos. Por ello se decidió utilizar PostgreSQL.

Como se ha mencionado en el apartado de los objetivos del capítulo 1, una de las finalidades de este proyecto es la automatización del proceso de transformación de los datos.

Figura 2.1: Carga de un fichero

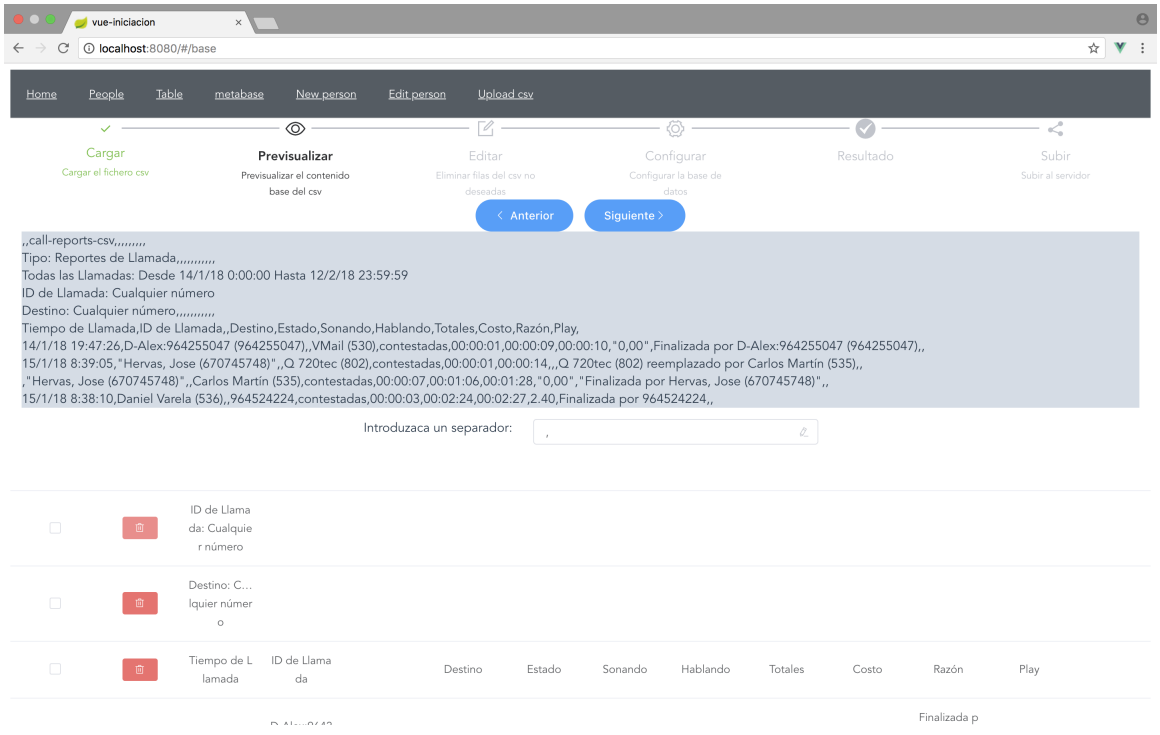

Figura 2.2: Eliminación de las líneas no deseadas y selección de la cabecera de los datos

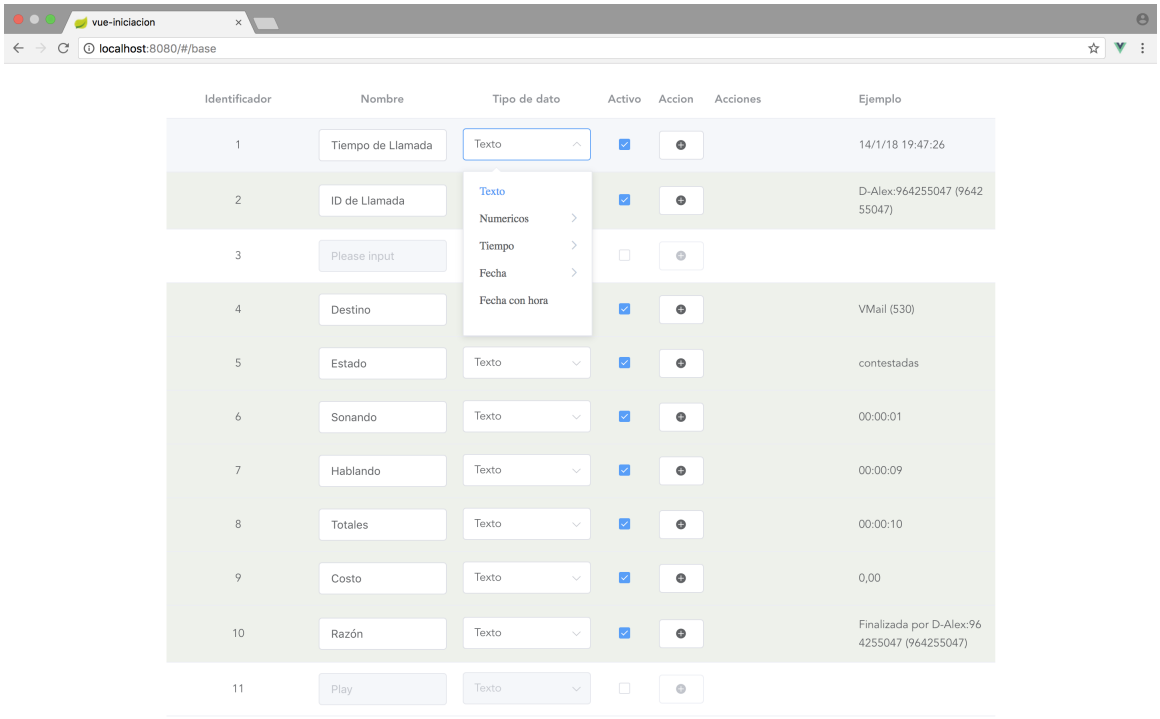

Figura 2.3: Tipos de datos aplicables a la información leída

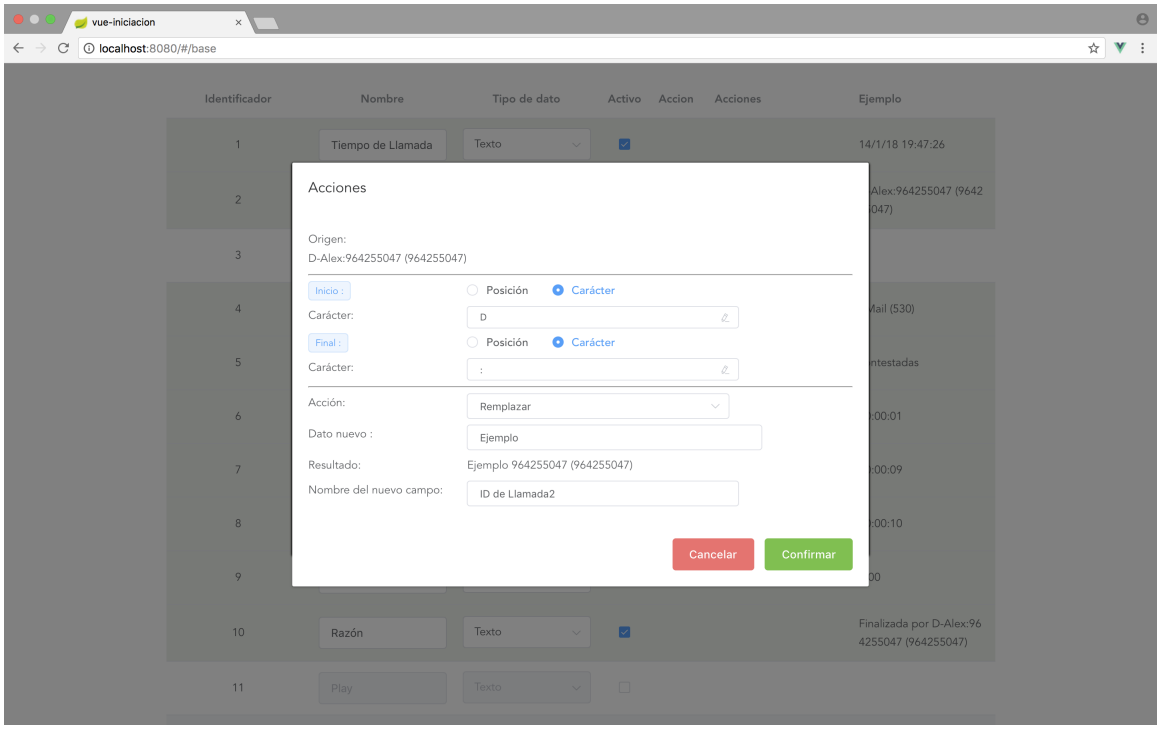

Figura 2.4: Ejemplo de la aplicación de una acción sobre un determinado dato

Para ello se tienen que almacenar los componentes generados a lo largo del proceso de tratamiento de datos. Las configuraciones que se han aplicado a los datos mientras estos eran procesados.

La base de datos seleccionada para guardar esta información es MongoDB, una base de datos no relacional. Se ha decidido emplear esta base de datos, por la sencilla razón de que tal y como están estructurados los datos resulta más eficiente almacenarlos en una base de datos no relacional.


```
[
    {
        "_id": "5b0d967e0c9e3e0c937b8679",
        "date": "2018-05-29T00:00:00.000Z",
        "name": "intento1",
        "fileName": "Todaslasllamadasn110000_2805_zT5diSRtVProE9qH6eNV.csv",
        "separator": ",",
        "__v": 0,
        "unwantedIndexes": [
            0,
            1,
            2,
            3,
            4,
            5
        ],
        "components": [
            {
                "id": "1",
                "name": "Tiempo de Llamada",
                "dataType": [
                    "timestamp"
                ],
                "index": 0,
                "order": 0,
                "isActive": true,
                "actions": [],
                "children": [],
                "phather": true,
                "count": 2
            },
```


Figura 2.5: Imagen 5: Estructura de los componentes a guardar, realizada en JSON.

Como señala la Figura 2.5, para poder automatizar el proceso del modelado, es necesario guardar una serie de datos. En este caso, se guarda el nombre del modelo, el nombre del fichero de donde provienen los datos, el separador empleado para dividir los distintos componentes, los índices de las filas que no se desean tener porque no son relevantes para el propósito y los componentes en sí. De cada componente se almacena el identificador del componente (id), el nombre de la columna a la que hace referencia (en este caso Tiempo de Llamda), el tipo de datos que alberga la columna referenciada, el índice de dicha columna, si dicha columna está o no activa y las acciones que se hayan realizado sobre dicha columna.

Visualización de los datos: La tecnología seleccionada para la representación de estos datos es Superset, en el siguiente apartado se puede ver todo el estudio del arte realizado.

Esta tecnología recoge los datos de una base de datos y los muestra en las gráficas que se le han definido.

En esta parte del proyecto apenas se ha tenido que programar, simplemente se han tenido que declarar las consultas SQL para que Superset busque los datos especificados y configurar los dashboard que se querían visualizar.

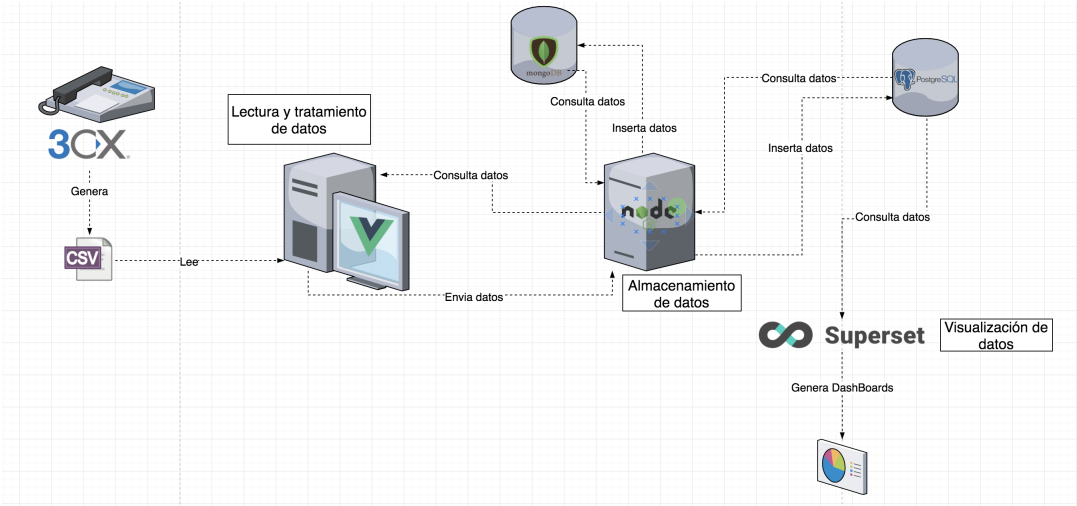

Figura 2.6: Diagrama del sistema

En resumen, al igual que se puede ver en la Figura 2.6, la centralita 3cx genera un fichero csv con la información de todas las llamadas que se han recibido y/o se han realizado. La parte cliente lee este fichero. Posteriormente se comunica con el servidor para transferir los datos ya tratados, éste a su vez los comunica a las bases de datos.

Por consiguiente, Superset lee los datos directamente de la base de datos PostgreSQL y los visualiza en las dashboards que han sido preconfiguradas para mostrar las gráficas que sean oportunas.

El procedimiento de lectura de datos de la parte del cliente únicamente se realiza la primera vez para configurar los componentes del fichero leído. Posteriormente es el propio servidor, el que realiza la lectura de los datos y lo procesa según la configuración que se ha establecido inicialmente.

2.1. Estudio del estado del arte de las tecnologías

Para poder realizar el presente proyecto es necesario seleccionar una tecnología base para la visualización de los datos.

La tecnología buscada debe de ser capaz de realizar representaciones gráficas tales como tablas, gráficos de barras, gráficos de líneas, gráficos de sectores y mapas, a partir de un conjunto

de datos. Además tiene que tener la capacidad de realizar cálculos, como por ejemplo calcular la media, el máximo, el mínimo, la mediana, etc. de los datos introducidos.

Es imprescindible que sea "open-source". Esto se debe a que en caso de ser necesario poder consultar el código fuente, ya sea por saber cómo está implementado algún componente o para solventar algún problema que pueda aparecer.

Asimismo tiene que ser accesible desde cualquier otro sistema, en otras palabras , debe de poseer una API para poder comunicar con ella. Igualmente es necesario que se integre o ejecute en un entorno web.

Finalmente debe de ofrecer la posibilidad de realizar filtrado de los datos que se visualizan en tiempo real. Y todo lo comentado tiene que ir acompañado de un diseño elegante y atractivo, de tal modo que sea simple y sencillo para facilitar la comprensión de todas las gráficas que se muestran.

Inicialmente el estudio tomó un enfoque de tecnología dashboard. En este enfoque se estudiaron las siguientes tecnologías.

- Razorflow [20]: implementada en HTML 5. Ofrece la posibilidad de conectar con dos APIs, una en JavaScript y la otra en php, pero no posee lógica alguna, es decir, no permite realizar cálculos con los datos que lee. Del mismo modo tampoco facilita la realización de un filtrado de datos en tiempo real. Cuenta con un diseño muy simple pero por contra es muy poco atractivo.
- Dashing [6]: permite usar widgets prefabricados y incluso fabricar unos propios empleando SCSS, HTML y coffeescript. Tiene una API que funciona bajo el protocolo HTTP. Pero por contra presenta los mismo problemas que Razorflow.
- StashBoard [22]: programada en en Python y alojada en Google Engine. Tiene una API Rest Full y para ejecutarla hay que hacerlo desde una cuenta de Google Engine. El principal problema de esta tecnología es la falta de información. Además de que el diseño es muy poco atractivo.
- FreeBoard [8]: Desarrollada en JavaScript. Tiene una API que funciona bajo protocolo HTTP y cuenta con un diseño simple y atractivo. Pero no permite realizar filtrado de datos en tiempo real ni tampoco tiene lógica añadida.
- Mozaik [15]: Basada en Nodejs, react , d3 y stylus. Por ello ofrece la posibilidad de ser extendida por módulos, es decir, para extender funcionalidades simplemente hay que instalar el módulo necesario. Pero presenta los mismos problemas que Razorflow y Dashing.
- DashBuilder [5]: Publicada bajo la licencia de apache. Presenta una API con protocolo HTTP, da la posibilidad de exportar los datos en formato excel y csv. Pero al igual que mozaik, no tiene un buen diseño, ni lógica alguna y tampoco permite realizar filtrado de los datos en vivo.
- Grafana [9]: Implementada en JavaScript. Existe la posibilidad de ser ejecutada en los propios hosts de Grafana o bajo cualquier otra plataforma. Tiene una API con protocolo HTTP, posee un diseño atractivo y simple para el usuario, asimismo permite ser extendida

a través de plugins. Incorpora una lógica añadida, es decir, permite aplicar operaciones sobre los datos que se le pasan. Pero no permite la inserción de filtros en vivo.

Tras concluir esta primera fase de estudio la empresa se dió cuenta de que el enfoque de tomado no era del todo correcto. Por ello se decidió cambiar el rumbo a las tecnologías de *Business Intelligence*.

A continuación se muestran las tecnologías estudiadas en esta nuevo enfoque.

- Visualizejs [25]: Es una librería de JavaScript. Se emplean llamadas REST para la comunicación y permite crear bonitos dashboards utilizando HTML5. Además tiene un diseño atractivo y simple. Pero no tiene lógica añadida ni tampoco permite filtrar en vivo los datos que se muestran.
- Metabase [12] Utiliza una API con protocolo HTTP. En el caso de no querer emplear la API, se puede hacer que conecte con una de las siguientes bases de datos: PostgreSQL, MySQLl, Druid, SQLServer, RedShit, MongoDB, Google BigQuerry, SQLite, H2, CrateDB, Oracle, Vertica y Presto. Es una tecnología un tanto diferente de las ya estudiadas, debido a que se ejecuta como un servicio en JAR, Docker o Mac Application. Tiene lógica, esto es, puede aplicar operaciones sobre los datos leídos y tiene un diseño simple y sencillo. Pero, no permite realizar filtros en vivo de los datos.
- Qclik Sense [19]: Posee una API HTTP. Cuenta con un diseño atractivo y elegante. Asimismo, permite aplicar filtros en vivo. Pero no es open-source, se presenta como una tecnología que se ejecuta en la nube empleando sus propios servicios. Además tampoco permite aplicar acciones sobre los datos leídos.
- DashBoardFree [4]: No dispone de una API. Tiene que conectarse a una base de datos para poder obtener la información. Su diseño no es muy atractivo, aunque su gran problema es la falta de documentación, cierto es que tiene una guía de como se debe configurar por primera vez, pero no se especifica en ningún lado por ejemplo las base de datos que están soportadas, ni tampoco las operaciones que permite realizar con los datos leídos, etc.

Tras finalizar este segundo intento y ver que no se ha dado con la tecnología adecuada aún, uno de los miembros de la empresa propuso que se estudiara “Superset”.

Superset [23] es una aplicación web de exploración y visualización de datos. Con una interfaz amigable e intuitiva permitiendo que el usuario pueda emplearla sin complicación alguna.

La aplicación representa la información leída en dashboards mediante gráficas, las cuales previamente son predefinidas y configuradas. La información puede ser leída a partir de una gran variedad de bases de datos, tales como PostgreSQL, MySQL, Oracle , etc. , permitiendo de esta forma una mayor flexibilidad para proporcionar los datos.

Permite realizar operaciones con los datos, como por ejemplo sacar el máximo, la media o la mediana de un conjunto de estos. Si las operaciones que se desean realizar no se encuentran entre las ofrecidas por la propia aplicación, existe la posibilidad de definir esa operación en

lenguaje SQL, es decir, realizar una consulta a la base de datos aplicando la operación que se necesite.

Del mismo modo también permite aplicar filtrado en vivo de los datos que se muestran en las tablas, de tal manera que las gráficas varían en función del filtrado que se aplica.

Una de las principales ventajas de esta aplicación es que es un proyecto activo. Posee un repositorio público en GitHub dónde los usuarios y los propios creadores de Superset, no paran de realizar *commits* con mejoras o con soluciones a posibles errores encontrados.

Uno de los inconvenientes es que en su gran mayoría está programada en Python, lo que hace muy complicado la legibilidad, la comprensión y la depuración del código. Esto es porque Python es un lenguaje interpretado que ofrece tipos de datos dinámicos, las variables pueden cambiar de tipo a lo largo de todo el código, cosa que dificulta su lectura.

Por último concluir recapitulado que se trata de un proyecto consolidado y avalado por Apache Software Foundation (ASF), ya que se encuentra en la Incubadora de Apache.

A continuación se muestra la Cuadro 2.1 dónde se comparan todas las tecnologías analizadas. En ella se puede apreciar que la única tecnología que cumple con todos los requisitos es Superset, como ya se ha comentado en los párrafos anteriores.

	open-source	Lógica	API	Entorno web	Diseño elegante	Filtros de selección
Razorflow	✓	✗	✓	✓	✗	✗
Dashing	✓	✗	✓	✓	✗	✗
Stashboard	✓	✗	✓	✓	✗	✗
Freeboard	✓	✗	✓	✓	✓	✗
Mozaik	✓	✗	✓	✓	✗	✗
Dashbuilder	✓	✗	✓	✓	✗	✗
Grafana	✓	✓	✓	✓	✓	✗
VisualiceJs	✓	✗	✓	✓	✓	✗
Metabase	✓	✓	✓	✓	✓	✗
Qclik Sense	✗	✗	✓	✓	✓	✓
Dashboardfree	✓	✗	✗	✓	✗	✗
Superset	✓	✓	✓	✓	✓	✓

Cuadro 2.1: Comparación entre las distintas tecnologías analizadas

2.2. Tecnologías

En esta sección se van a describir las tecnologías que se han empleado para realizar el presente proyecto.

Visual Studio Code [24] . Se trata de una herramienta desarrollada por Microsoft. Es multiplataforma y en comparación con Visual Studio es más ligero ya que las funcionalidades se le añaden mediante extensiones a medida que se van necesitando. Según los datos proporcionados

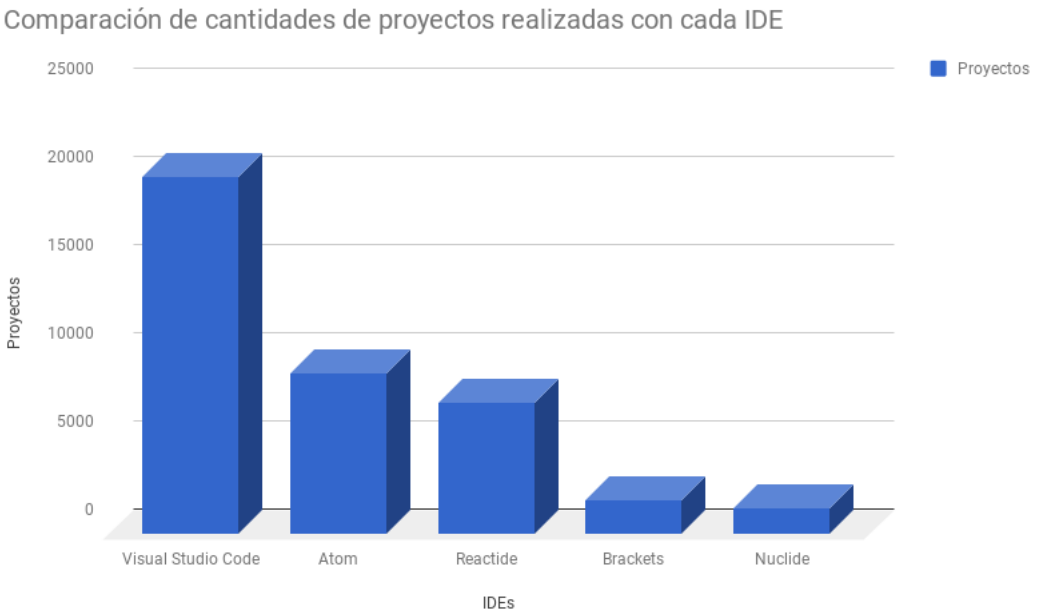

Figura 2.7: Comparación de IDEs empleadas durante el año 2017

por Risingstars [21] se ha podido elaborar un gráfico de barras comparando Visual Code con otros IDEs.

Tal y como se muestra en el gráfico anterior, la cantidad de proyectos realizados con Visual Studio Code es más del doble que los realizados con Atom que está en segundo puesto, mientras que es casi veinte veces la cantidad realizada con Nuclide. Esto se debe al gran soporte que ofrece Visual Studio Code y a la gran cantidad de actualizaciones que lanzan para corregir errores y para mejoras que facilitan la programación.

JavaScript [11]: Es un lenguaje de programación ligero e interpretado, orientado a objetos con funciones de primera clase. Puede ser aplicado a documentos HTML y empleado para agregar la interactividad dinámica de los sitios web. Pero también se emplea en entornos que no requieren navegador, como por ejemplo Nodejs.

Vue.js [27]: Es un *framework* de JavaScript destinada a construir interfaces. Está enfocado única y exclusivamente a la capa de visualización. Ofrece mucha facilidad para la integración con otras bibliotecas.

Element.io [7]: Es una biblioteca de componentes visuales basada en Vuejs 2.0. En el proyecto actual la mayoría de componentes visuales son de esta librería.

Nodejs [16]: Es un motor de construcción de JavaScript, impulsado por eventos asíncronos, es decir, cuando un cliente realiza una petición a un servidor, este continúa con sus tareas mientras el servidor le va respondiendo. Está enfocado para la creación de aplicaciones escalables.

PostgreSQL [17]: Es un sistema de bases de datos relacional de objetos abierto, que emplea

y amplía el lenguaje SQL. Se ejecuta en todos los principales sistemas operativos. Posee una reputación muy alta gracias a su arquitectura comprobada, su integridad de datos y a su robusto conjunto de funciones entre otras cosas.

MongoDB [14] Es una base de datos no relacional, esto es, que los datos no se guardan en tablas si no en ficheros flexibles similares a los JSON. Esto le da una gran fuerza y una gran flexibilidad al sistema que lo emplea ya que los ficheros pueden variar de estructura. Además es una base de datos distribuida en su núcleo, por lo tanto la alta disponibilidad, la escala horizontal y la distribución geográfica están integradas. Es gratuito y "open source", está publicado bajo la licencia "*GNU Affero General Public License*".

HTML [10]: Lenguaje de Marcado para Hipertextos (HyperText Markup Language). Es la pieza de creación más simple de una página web. Emplea etiquetas para definir los bloques. Este lenguaje no aporta funcionalidad a una página web, simplemente aporta el esqueleto y la representación visual de esta.

CSS [3]: Hojas de Estilo en Cascada (Cascading Style Sheets). Es el lenguaje gastado para detallar la representación de documentos HTML o XML. Casi todos los navegadores de hoy en día y un gran número de aplicaciones admiten CSS.

Vue.js devtools [26]: Es una herramienta que sirve para facilitar la detección de errores en Vuejs. Permite revisar el entorno de desarrollo, facilitando por ejemplo la visualización del valor de una determinada variable en un instante concreto o el componente que ha reaccionado ante un determinado evento, etc. Además su uso es bien sencillo ya que se emplea en tiempo real desde Google Chrome.

Postman[18]: Es un entorno de desarrollo para APIs. Tienen muchas funcionalidades, algunas de las cuales son, crear colecciones de las APIs de una aplicación, ejeuctar scripts que testean dichas colecciones, crear y ejecutar llamadas a las APIs, etc.

Para concluir este apartado se podría decir que todas estas tecnologías tienen que ver con el mundo web, cada una de ellas aporta un granito para hacer que este mundo sea más dinámico, visual, entretenido y aporte información.

2.3. Control de versiones

En cuanto a lo referente al control de versiones, se ha empleado GitLab. Una herramienta muy parecida a GitHub, que permite realizar pruebas o lanzamiento de paquetes, servicios, configurar monitores de seguimiento, etc. En este caso, la empresa 720tec tiene un GitLab propio configurado para sus servicios y es el que se ha empleado en el presente proyecto.

Para proyecto actual se ha creado un repositorio para la parte front-end y otro para la parte del back-end.

En ambos casos los repositorios tienen dos ramas, uno de desarrollo y otro principal.

La metodología empleada es la siguiente, inicialmente todos los cambios que se realizan se suben a la rama de desarrollo, una vez que se ha probado su correcto funcionamiento, se hace un "merge" en la rama principal y esos cambios pasan a estar en dicha rama.

Visual Studio code, posee su propia herramienta para trabajar con repositorios git. Pero en el caso del presente proyecto se ha optado por trabajar utilizando ordenes en línea de comandos.

En cualquier proyecto que abarque algún tipo de programación es indispensable el uso de una herramienta para el control de versiones para tener la posibilidad de trabajar desde cualquier otro ordenador, volver a versiones anteriores y guardar el proyecto de manera segura.

Capítulo 3

Planificación del proyecto

En este capítulo se va ha explicar la metodología empleada para desarrollar el proyecto y los componentes que tiene dicha metodología. Así como los recursos necesarios de este y el coste de los mismo. Asimismo, también se va ha detallar el seguimiento realizado.

3.1. Metodología

Tal y como se indica en [29] el 92 % de los proyectos en Francia implican una metodología ágil en comparación con un promedio del 96 % a nivel mundial ("92 % of all projects in France involve an agile practice as opposed to an average of 96 % worldwide")

Según lo citado anteriormente, alrededor del 96 % de los proyectos informáticos que se realizan a nivel mundial se realizan empleando metodologías ágiles. Esto se debe a las grandes ventajas que éstas ofrecen, entre los que destacan:

- Respuesta rápida a los cambios. Al ser evolutivo el proceso de desarrollo, el equipo de trabajo puede apreciar los errores a medida que va evolucionando en el proyecto, añadir funcionalidades o quitar alguna de las ya programadas.
- El cliente está involucrado de manera activa en cada una de las etapas del desarrollo, pudiendo aportar ideas, explicaciones, opinar sobre los resultados que se van obteniendo, etc.
- Se realizan entregas parciales pero funcionales del proyecto. El cliente puede ir viendo la funcionalidad que el proyecto va adquiriendo en cada una de las entregas.

3.1.1. ¿ Por qué emplear una metodología ágil para el presente proyecto ?

La principal razón es una de las ventajas que se ha comentado en la sección anterior, la rápida respuesta a los cambios. El presente proyecto es algo novedoso para la empresa, núnca se

han enfrentado a un proyecto de estas características con tantas tecnologías nuevas y teniendo que tratar con una tecnología de visualización a parte.

Esto posibilita la aparición de la necesidad de realizar cambios durante el proceso de desarrollo del proyecto. Estos cambios no tienen que afectar al flujo del proyecto, es decir, no tienen que retrasarlo, ni tampoco tienen que paralizarlo, simplemente se tienen que realizar y continuar con el proyecto.

Otra razón es que la empresa 720tec emplea metodologías ágiles para el desarrollo de todos sus proyectos, lo que significa que tiene un equipo muy cualificado en este tipo de metodologías que puede aportar mucho a la hora de realizar el presente proyecto.

3.1.2. Metodología empleada en el presente proyecto

Existe un gran número de metodologías ágiles (Extreme Programming o XP, Lean Software Development, Dynamic Systems Development Method o DSDM, Crystal Clear, XBreed, Adaptative Software Development o ASD, Agile Unified Process o AUP, etc.).

Dadas las características de este proyecto se ha elegido Scrum. Las razones de la selección de esta metodología son:

- Flexibilidad ante los cambios.
- Reducción del Time to Market, el cliente puede observar y emplear algunas funciones del producto antes de que esté finalizado.
- Los miembros de la empresa han trabajado con esta metodología y tienen gran experiencia en ella.

En los siguientes apartados se van a explicar ciertas cuestiones básicas e importantes a cerca de las metodologías ágiles que son: las historias de usuario, los puntos de historia y la pila del producto.

3.1.3. Historias de usuario

En las metodologías ágiles se emplean las historias de usuario para realizar la especificación de los requisitos del sistema. Cada historia de usuario describe una necesidad del cliente y qué funcionalidad se tiene que implementar para cubrirla.

Según los apuntes de la asignatura *"EI1050 Métodos ágiles"* una historia de usuario debe de ser:

- Independent: Las historias de usuario tienen que ser independientes unas de otras. De tal modo que si en un futuro hay alguna modificación en alguna historia de usuario, que no afecte al resto.

- Negociable: Una historia de usuario tiene que estar abierta a posibles modificaciones, no es inmutable.

- Valuable: Una historia de usuario tiene que formar parte del valor final del proyecto, es decir, le tiene que aportar valor.

- Estimable: Se tiene que poder estimar el tiempo que se va ha tardar en realizar, si no, no se puede planificar su desarrollo.

- Small: La definición de la historia de usuario debe de tener un nivel de complejidad para que permita su planificación, priorización, etc.

- Testable: Una historia de usuario se tiene que poder testear desde distintas perspectivas para verificar que se cumplen los requisitos especificados.

Estas seis características conforman el concepto "INVEST" creado por Bill Wake

También se recomienda que la historia de usuario tenga un nivel de complejidad en su definición.

La estructura de una historia de usuario está compuesta por:

- Id: identificador de la historia de usuario

- Título: nombre de la historia, suele ser una corta descripción de la misma

- Descripción: explicación de una determinada necesidad que el programa a implementar debe de cubrir. Está escrita en lenguaje normal. Su forma es: Como [rol], quiero [objetivo], para [beneficio]

- Prioridad: valor que le da el cliente a esta parte del software teniendo en cuenta las aportaciones que esta historia le da al sistema.

- Puntos de historia: Estimación de la complejidad de la historia de usuario

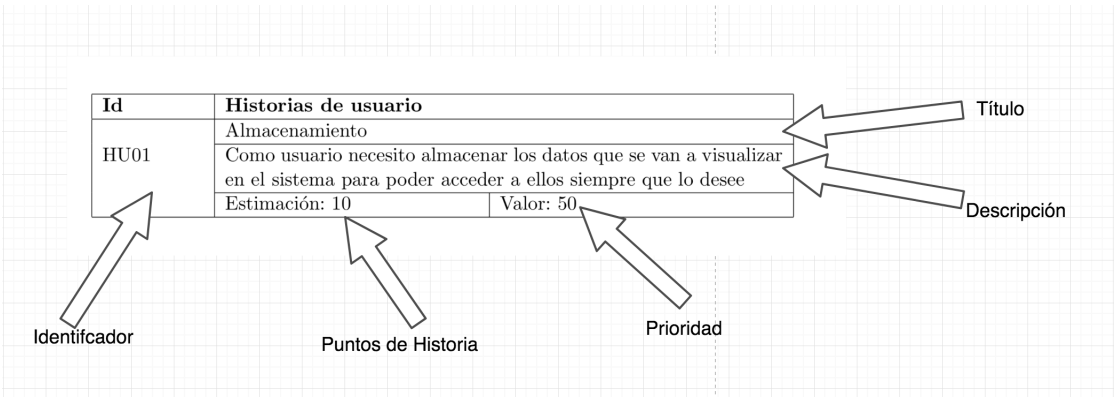

Figura 3.1: Estructura de una historia de usuario

3.1.4. Puntos de historia

Los puntos de historia son unas medidas asociadas a las historias de usuario que sirven para medir la complejidad de dicha historio. En este caso, 1 punto de historia de usuario equivale a 2 horas de trabajo, esto se ha decidido así debido a que 2 es divisor de 10 y es más sencillo realizar luego las cuentas.

3.1.5. Pila del producto

La pila del producto es un conjunto ordenado de todas las historias de usuario necesarias para desarrollar el proyecto.Esta pila no es inmutable, puede variar teniendo en cuenta todos los posibles cambios que surjan a lo largo del proyecto.

La pila del producto surge a partir de una serie de pasos que hay que realizar inicialmente:

- Crear un listado de todos los requisitos del cliente, a partir de una reunión con el mismo. En el presente proyecto no se ha realizado, al ser un proyecto de una estancia en prácticas, la empresa se ha encargado de realizar esta parte y se la ha entregado posteriormente al estudiante.
- A partir del listado, crear las correspondientes historias de usuario.
- Establecer para cada historia de usuario sus puntos de historia.
- Fijar la prioridad de cada historia de usuario por parte del cliente.

3.2. Planificación

En este apartado se va ha describir la planificación que se ha llevado a cabo para el proyecto presente.

3.2.1. Pila del producto inicial del proyecto

En principio se planteó la pila tal y como se ve en la Cuadro 3.1, pero en el caso del presente proyecto no es del todo correcta debido a que en este proyecto existen dos tipos de usuarios, los que configuran los dashboards, y los que consultan la información generada por los dashboards. En este documento, a los primeros se les llamará Generadores y a los segundos Visualizadores. Según se puede visualizar en los cuadros 3.2 y 3.3 respectivamente.

La estimación de los puntos de historia necesarios para desarrollar el proyecto sale a 112 puntos de historia, lo que equivale a unas 224 horas, esto es así porque se han dedicado unas 75 horas (3 semanas) a la selección de la tecnología base. Hay que clarificar que las estimaciones se han calculado teniendo en cuenta que el alumno no conoce los lenguajes de programación con

Id	Historias de usuario	
HU01	Almacenamiento	
	Como usuario necesito almacenar los datos que se van a visualizar en el sistema para poder acceder a ellos siempre que lo desee	
	Estimación: 25	Valor: 80
HU02	Selección del tipo de datos	
	Como usuario necesito poder seleccionar de qué tipo van ha ser los datos que se van a almacenar para que se genere la tabla en la base de datos de modo automático.	
	Estimación: 10	Valor: 60
HU03	Elección de la estructura de datos	
	Como usuario necesito poder elegir la estructura de la base de datos que se va ha emplear para almacenar los datos.	
	Estimación: 12	Valor: 70
HU04	Modelado de datos	
	Como usuario necesito poder modelar a mi gusto los datos que se van a representar, para mostrarlos según me convenga	
	Estimación: 40	Valor: 90
HU05	Selección de datos	
	Como usuario necesito poder seleccionar los datos sobre los que deseo realizar una gráfica para poder analizar esos datos en concreto	
	Estimación: 5	Valor: 40
HU06	Selección del tipo de gráfico	
	Como usuario necesito poder seleccionar el tipo de gráfico con que representar un determinado conjunto de datos	
	Estimación: 6	Valor: 40
HU07	Realización de operaciones sobre los datos	
	Como usuario necesito poder realizar operaciones matemáticas sobre los datos a representar como puede ser la media, el máximo, el mínimo, para poder representarlos en las gráficas	
	Estimación: 6	Valor: 40
HU08	Filtrado de datos	
	Como usuario visualizador de datos necesito poder realizar un filtrado de los datos que se muestran para poder elaborar un mejor análisis de estos .	
	Estimación: 4	Valor: 20
HU09	Gráfico de barras	
	Como usuario visualizador de datos necesito poder ver un gráfico de barras que resuma la información de los datos tratados para analizar.	
	Estimación: 4	Valor: 20

Cuadro 3.1: Pila del producto inicial del proyecto

Id	**Historias de usuario**	
HU01	Almacenamiento	
	Como usuario generador necesito almacenar los datos que se van a visualizar en el sistema para poder acceder a ellos siempre que lo desee.	
	Estimación: 25	Valor: 40
HU02	Selección del tipo de datos	
	Como usuario generador necesito poder seleccionar de qué tipo van ha ser los datos que se van a almacenar para que se genere la tabla en la base de datos de modo automático.	
	Estimación: 10	Valor: 40
HU03	Elección de la estructura de datos	
	Como usuario generador necesito poder elegir la estructura de la base de datos que se va ha emplear para almacenar los datos.	
	Estimación: 12	Valor: 40
HU04	Modelado de datos	
	Como usuario generador necesito poder modelar a mi gusto los datos que se van a representar, para mostrarlos según me convenga.	
	Estimación: 40	Valor: 60
HU05	Selección de datos	
	Como usuario generador necesito poder seleccionar los datos sobre los que deseo realizar una gráfica para poder analizar esos datos en concreto	
	Estimación: 6	Valor : 40
HU06	Selección del tipo de gráfico	
	Como usuario generador necesito poder seleccionar el tipo de gráfico con que representar un determinado conjunto de datos	
	Estimación: 6	Valor: 20
HU07	Realización de operaciones sobre los datos	
	Como usuario generador necesito poder realizar operaciones matemáticas sobre los datos a representar como puede ser la media, el máximo, el mínimo, para poder representarlos en las gráficas.	
	Estimación: 5	Valor: 20

Cuadro 3.2: Pila del producto inicial del proyecto(usuarios generadores)

Id	Historias de usuario	
HU08	Filtrado de datos	
	Como usuario visualizador de datos necesito poder realizar un filtrado de los datos que se muestran para poder elaborar un mejor análisis de estos .	
	Estimación: 4	Valor: 20
HU09	Gráfico de barras	
	Como usuario visualizador de datos necesito poder ver un gráfico de barras que resuma la información de los datos tratados para analizar.	
	Estimación: 4	Valor: 20

Cuadro 3.3: Pila del producto inicial del proyecto(usuarios visualizadores)

los que se van a desarrollar el proyecto, ni tampoco conoce las tecnologías necesarias. Por ello las historias de usuario tienen una estimación tan variada, ya que el alumno va comprendiendo mejor las tecnologías y los lenguajes de programación.

3.3. Estimación de recursos y costes del proyecto

En este apartado se va ha desglosar los recursos que han sido necesarios para desarrollar el proyecto y el coste que ha conllevado.

3.3.1. Recursos

- Humanos
  - Programador
  - Tester
  - Analista
  - Responsable
  - Cliente
- Hardware
  - Ordenador Asus: Procesador Intel Core i5-4310 2,70GHz , memoria 8GB y 500GB de almacenamiento.(Aportado por la empresa)
  - Auriculares: Aportado por el alumno para ver los cursos de formación
  - Ratón: Aportado por la empresa
  - Teclado: Aportado por la empresa
  - Pantalla adicional: Para poder programar y testear a la vez. En la pantalla integrada del ordenador portátil se tiene el navegador el navegador abierto con la ruta de la aplicación para probarla y en la pantalla adicional se tiene el código de la aplicación web. (Aportada por la empresa)

- Software
    - Windows 10: Software base de la empresa, todos los ordenadores de la empresa están conectados en red, empleando un dominio creado en Windows.
    - Linux Server: Se emplea para ejecutar todo lo necesario para el proyecto que requiera estar en linux, como por ejemplo las bases de datos o Superset.
    - Visual Studio Code: Editor empleado para programar la aplicación.
    - JavaScript,Nodejs, PostgreSQL, Vuejs, MongoDB, postman, etc.

3.3.2. Costes

Según las historias de usuario se ha estimado que van a ser necesarias unas 300 horas para realizar el proyecto. Esto supone tener a una persona programando ese número de horas. Según la empresa, un programador junior cobra al rededor de 1200€ al mes.

En un mes se suele trabajar alrededor de unos 21 días, y una jornada laboral tiene 8 horas. Entonces al hacer los cálculos sale que un programador junior cobra alrededor de unos 7,2€ por hora.

El programador de este proyecto supone un coste de 2160€ ya que 300 horas * 7,2€/hora = 2160€. Pero a este precio hay que sumarle un 30 % de su total, porque aún hace falta pagar los costes de seguridad social y demás costes del trabajador, 2160 * 1,3 = 2808. Lo que supone que el programador cuesta 9,36€ la hora.

En cuanto al hardware necesario para el desarrollo del proyecto. El ordenador portátil cuesta 800€, el ratón 10€, el teclado 20€ y la pantalla adicional 200€, lo que supone un total de 1030€.

En cuanto al coste del software necesario para el proyecto sólo hay que tener en cuenta la licencia de Windows 10 que se ha comprado, ya que es el único software de pago empleado, todo el resto son gratuitos. Esta licencia cuesta 259€ según se indica en la página oficial de *Microssoft Windows* [13].

Por último, hay que añadir un 20 % del total acumulado para los gastos de los recursos básicos necesarios para el funcionamiento de la empresa y de los trabajadores, como son la luz, el agua e internet.

En el Cuadro 3.4 se muestra un resumen de todo lo que se ha mencionado anteriormente.

3.4. Seguimiento del proyecto

En el apartado actual se van a mostrar todos los sprints realizados a lo largo del proyecto, las modificaciones realizadas en las historias de usuario y las actualizaciones que se han realizado en la pila del producto.

Las fechas de los sprints son:

Tipo de recurso	Recurso	Coste	Total
Humanos	Programador Junior	300h x 9,36€/h	2808€
Hardware	Ordenador portatil	1 x 800€	800€
	Pantalla Adicional	1 x 200€	200€
	Teclado	1 x 20€	20€
	Ratón	1 x 10€	10€
Softaware	Windows 10 Pro	1 x 259€	259€
Recursos	Luz, agua, internet	0,3 x 4097€	1229,1,4 €
Total			**5326,1€**

Cuadro 3.4: Resumen de costes del proyecto.

- Sprint 1 : del 12/02/2018 al 19/02/2018
- Sprint 2: del 19/02/2018 al 26/02/2018
- Sprint 3: del 26/02/2018 al 13/03/2018
- Sprint 4: del 13/03/2018 al 22/03/2018
- Sprint 5: del 22/03/2018 al 27/03/2018
- Sprint 6: del 27/03/2018 al 10/04/2018
- Sprint 7: del 10/04/2018 al 17/04/2018
- Sprint 8: del 17/04/2018 al 24/07/2018
- Sprint 9: del 24/07/2018 al 8/05/2018
- Sprint 10: del 8/05/2018 al 17/05/2018
- Sprint 11: del 17/05/2018 al 28/05/2018

3.4.1. Sprint 1

En este primer Sprint, inicialmente se ha definido la propuesta técnica, posteriormente se ha dedicado todo el tiempo a la búsqueda de la tecnología base para el proyecto.

Para realizar la búsqueda de la tecnología base, inicialmente se ha hecho una búsqueda en la cual se han encontrado muchas tecnologías. Pero no todas ellas eran del todo adecuadas para el proyecto. Se ha hecho una primera selección donde se han eliminado todas aquellas que no eran adecuadas para el proyecto a simple vista y finalmente se han realizado pruebas con las restantes.

Aunque este sprint no estaba ligado a ninguna historia de usuario, es crucial realizarlo porque para que el proyecto se pueda realizar inicialmente se tiene que saber con qué tecnología se va ha trabajar. Por ello se han definido una serie de tareas:

Tarea	Subtarea	Completada al finalizar el sprint
Redacción de la propuesta técnica	Definición del alcance	Si
	Objetivos del proyecto	Si
	Planificación a emplear en el proyecto	Si
	Descripción del proyecto	Si
Realizar pruebas de grafana	Transferir datos	
	Crear gráfico de barras	Si
	Crear gráfico de sectores	Si
	Aplicar acciones sobre los datos	No
	Crear filtros en vivo	No
Realizar pruebas de FreeBoard	Transferir datos	Si
	Crear gráfico de barras	Si
	Crear gráfico de sectores	No
	Aplicar acciones sobre los datos	No
	Crear filtros en vivo	No

Cuadro 3.5: *Backlog* sprint 1

En las actividades de la tarea de "Realizar pruebas de Grafana" no se han podido realizar las de "aplicar acciones sobre los datos" y "crear un filtrado en vivo", porque la tecnología no lo permite y por ello se ha descartado para el proyecto.

En el caso de las pruebas de FreeBoard, tampoco se han completado debido a las mismas razones que en el caso de Grafana, el sistema de FreeBoard no permite realizar esas acciones, con lo cual también ha sido descartada del proyecto.

3.4.2. Sprint 2

En este sprint se ha continuado con la búsqueda de la tecnología base ya que en el sprint anterior no se ha logrado encontrar ninguna que cubra los requisitos que la empresa ha propuesto.

Tareas del sprint 2.

- Probar DashboardFree
- Probar Qclik sense
- Probar Visualicejs
- Probar PowerBi
- Probar Metabase (incompleta)

En el Cuadro 3.6 solo aparece el desglose de una de las 5 tareas que hay que realizar, eso

Tarea	Subtarea	Completada al finalizar el sprint
Probar DashboardFree	Transferir datos	Si
	Crear gráfico de barras	Si
	Crear gráfico de sectores	Si
	Aplicar acciones sobre los datos	No
	Crear filtros en vivo	No

Cuadro 3.6: Parte del *Backlog* sprint 2

es así porque todas tienen las mismas subtareas a realizar. Por ello es preferible explicarlo a continuación.

Dashboardfree queda descartada del proyecto porque no puede aplicar acciones sobre los datos ni tampoco crear filtros en vivo. Qclik Sense también queda descartada pero en este caso porque es un servicio que procesa los datos en la nube, pero eso no interesa, porque no se tiene control de los procesos que se realzan. PowerBi se descarta porque es de pago y no es *open-source*. Visualicejs es una biblioteca para JavaScript, se descarta porque no es lo que se busca. Por último metabase queda incompleta, se pasará al próximo sprint (sprint 3).

3.4.3. Sprint 3

En este sprint se ha continuado con las pruebas de metabase hasta la primera mitad del sprint y luego se ha empezado un tutorial de Vuejs, para que el alumno en prácticas aprenda los conceptos básicos de esta librería.

Metabase se ha descartado porque no permite el filtrado en vivo de los datos. Por lo tanto el proyecto sigue sin tener tecnología base.

3.4.4. Sprint 4

En este sprint han aparecido dos nuevas historias de usuario que no fueron contempladas inicialmente. Las historias de usuario reciben los identificadores HU10 y HU11 respectivamente.

"Como usuario generador necesito poder leer los datos de un fichero csv para poder procesarlos después"

"Como usuario generador quiero poder separar los datos leídos empleando ',' para poder tratar los datos después de manera más fácil"

En el Cuadro 3.8 se muestra el *Backlog* del sprint 4. En este caso se puede dividir el sprint en dos fases. La primera fase contiene las dos primeras tareas y la segunda fase contiene la tarea restante.

Id	Historias de usuario	
HU10	Leer un fichero con extensión csv	
	Como usuario generador necesito poder leer los datos de un fichero csv para poder procesarlos después	
	Estimación: 4	Valor: 100
HU11	Separación del texto empleando ","	
	Como usuario generador quiero poder separar los datos leídos empleando ","para poder tratar los datos después de manera más fácil	
	Estimación: 6	Valor: 90

Cuadro 3.7: Dos nuevas historias de usuario(HU10 Y HU11)

En la primera fase se ha tardado en realizar 18 horas, lo que equivale a tres días y tres horas de trabajo:

La tarea *Leer un fichero con extensión csv* (HU10) ha costado un total de 7 horas de realizar.

- Crear la pantalla de lectura de fichero : 3 horas
- Crear la lógica que permite leer el fichero: 4 horas

La tarea *Separación del texto empleando ","* (HU11) ha costado un total de 11 horas de completar.

- Crear la funcionalidad de separar las líneas leídas empleando como delimitador ",": 3 horas
- Crear la funcionalidad de guardar los datos separados: 3 horas
- Crear la interfaz para visualizar los datos formateados: 5 horas

En la segunda fase se ha estado probando a fondo Superset, no es una historia de usuario en sí pero hay que encontrar una tecnología base para poder desarrollar el proyecto.

- Instalación de Superset en el linux server: 30 minutos
- Creación de una dashboards con gráficas a partir de los datos ejemplo que proporciona el propio Superset: 1 hora
- Creación de una base de datos de pruebas para Superset: 1hora
- Conectar Superset con la base de datos : 1 hora
- Crear un gráfico a partir de los datos leídos: 30 minutos
- Creación de un gráfico de sectores: 30 minutos
- Creación de un filtro en vivo para los datos: 30min

Id	Descripción	Subtarea	Descripción	Completada al finalizar sprint
HU10	Leer un fichero con extensión csv	T01	Crear la pantalla de lectura del fichero	Si
		T02	Crear la lógica que permite leer el fichero	Si
HU 11	Darle formato a los datos leídos	T3	Crear la funcionalidad de separar las líneas leídas empleando como delimitador ”,”	Si
		T4	Crear la funcionalidad de guardar los datos separados	Si
		T5	Crear la interfaz para visualizar los datos formateados	Si
-	Testear Superset	T6	Instalación de Superset en el linux server	Si
		T7	Creación de una dashboards con gráficas a partir de los datos ejemplo que proporciona el propio Superset	Si
		T8	Creación de una base de datos de pruebas para Superset	Si
		T9	Conectar Superset con la base de datos	Si
		T10	Crear un gráfico a partir de los datos leídos	Si
		T11	Creación de un gráfico de sectores	Si
		T12	Creación de un filtro en vivo para los datos	Si
		T13	Filtrar por fecha	No

Cuadro 3.8: *Backlog* sprint 4

- Filtrar por fecha: 2 horas (No completada)

En esta segunda parte del sprint todo se ha realizado de manera correcta, exceptuando el filtrado por fechas, que no se realiza de manera correcta. Durante esas dos horas que duró la actividad, la primera media hora fue de intentar realizarlo, pero al ver que no se podía, la siguiente hora y media se dedicó a averiguar las razones. Pero no se encontró respuesta alguna.

3.4.5. Sprint 5

En este sprint nuevamente aparecieron dos nuevas historia de usuario que no se había contemplado inicialmente:

"Como usuario generador de dashboards necesito poder eliminar las filas no relevantes del fichero leído para reducir la cantidad de datos a tratar"

"Como usuario generador de dashboards necesito poder seleccionar una fila como cabecera para poder generar las columnas de la tabla de la base de datos con los nombres adecuados"

Id	**Historias de usuario**	
HU12	Eliminar las filas no relevantes del fichero	
	Como usuario generador de dashboards necesito poder eliminar las filas no relevantes del fichero leído para reducir la cantidad de datos a tratar	
	Estimación: 3	Valor: 100
HU13	Seleccionar una fila como cabecera	
	Como usuario generador de dashboards necesito poder seleccionar una fila como cabecera para poder generar las columnas de la tabla de la base de datos con los nombres adecuados	
	Estimación: 8	Valor: 90

Cuadro 3.9: Dos nuevas historias de usuario en sprint 5 (HU12 y HU13)

Id	Descripción	Subtarea	Descripción	Completada al finalizar sprint
HU04	Modelado de datos	T14	Realizar acciones sobre los datos	No
		T15	Crear dos campos para la acción de reemplazar	No
		T16	Encadenar cambios (acciones) realizados en un campo	No
HU12	Eliminar filas no relevantes	T17	Crear la funcionalidad de eliminado	Si
		T18	Crear el botón de eliminar	Si

Cuadro 3.10: *Backlog* sprint 5

En el Cuadro 3.10, se visualiza que en este caso solo contiene dos historias de usuario, debido a que gran parte del sprint, por no decir el 90 % , se ha dedicado a intentar solventar el error que ha aparecido en el pasado sprint con Superset. En este caso solo se ha podido completar la historia de usuario *Eliminar filas no relevantes HU12*. En la tarea *Crear funcion de eliminado T17* se han invertido 4 horas y en la tarea *Crear botón de eliminar T18* se ha invertido una hora de trabajo. La historia de usuario *Modelado de datos HU04*, no se ha podido realizar y se deja para el siguiente sprint.

3.4.6. Sprint 6

En este sprint se han realizado todas aquellas actividades que no se pudieron hacer en el pasado sprint. Además de las actividades referentes a la historia de usuario HU13 *Seleccionar una fila como cabecera*. Asimismo, han aparecido varios cambios a destacar. Uno de ellos es

que la actividad *T14, Realizar acciones sobre los datos*, se ha dividido en cinco actividades distintas ya que esta acción en sí era demasiado genérica. El otro es la incorporación de una nueva actividad a la historia de usuario HU04, *T19 Crear una interfaz conjunta para todas las acciones.* En el Cuadro 3.11 se puede apreciar cómo queda el *backlog* del sprint 6.

Id	Descripción	Subtarea	Descripción	Completada al finalizar sprint
HU04	Modelado de datos	T14-1	Acción de añadir un texto a un dato	Si
		T14-2	Acción de reemplazar un dato por otro	Si
		T14-3	Acción que permita extraer una cadena de un dato	Si
		T14-4	Acción que permita borrar un dato	Si
		T14-5	Acción que permita seleccionar un dato como fecha un dato	Si
		T15	Crear dos campos para la acción de reemplazar	Si
		T16	Encadenar cambios (acciones) realizados en un campo	Si
		T19	Crear una interfaz conjunta para todas las acciones	Si
HU13	Seleccionar una fila como cabecera	T20	Crear la funcionalidad de selección de cabecera	Si
		T21	Crear la interfaz que permita la selección de la cabecera	Si
		T22	Crear la función que permita utilizar la cabecera para dar nombre a las columnas	Si
		T23	Crear los inputs que permitan modificar los nombres dados	Si

Cuadro 3.11: *Backlog* sprint 6

La duración del sprint ha sido de 15 días, lo que supone un total de 50 horas de trabajo, a continuación se muestra el desglose del tiempo empleado para realizar cada una de las actividades.

La tarea de modelado de datos (HU04) ha costado un total de 33 horas y 15 minutos de realizar.

- T14-1 *Acción de añadir un texto a un dato* : 2 horas
- T14-2 *Acción de reemplazar un dato por otro* : 2 horas
- T14-3 *Acción que permita extraer una cadena de un dato* : 1 hora y 30 minutos

- T14-4 *Acción que permita borrar un dato* : 2 horas
- T14-5*Acción que permita seleccionar un dato como fecha un dato* : 2 horas
- T15 *Crear dos campos para la acción de reemplazar*: 15 minutos
- T16 *Encadenar los cambios (acciones) realizadas en un campo*: 11 horas y 30 minutos
- T19 *Crear una interfaz conjunta para todas las acciones*: 12 horas

La tarea Seleccionar una fila como cabecera (HU13) se ha realizado en 16 horas y 45 minutos

- T120 *Crear la funcionalidad de de selección de cabecera*: 5 horas
- T21 *Crear la interfaz que permita la selección de la cabecera*: 7 horas
- T22 *Crear la función que permita utilizar la cabecera para dar nombre a las columnas* : 4 horas y 30 minutos
- T23 *Crear inputs que permitan modificar los nombres dados*: 15 minutos

3.4.7. Sprint 7

En este sprint han aparecido nuevas actividades a realizar en torno a la historia de usuario HU04 *Modelado de datos*, y se ha abordado la historia de usuario HU03 *Elección de la estructura de la base de datos*. Todo esto se puede consultar en el Cuadro 3.12.

En este caso el sprint dura una semana, lo que equivale a 25 horas. En las subtareas de la historia de usuario HU03 *Elección de la estructura de la base de datos*, se ha tardado más en realizar debido a que el alumno en prácticas no conocía los fundamentos de MongoBD y ha tenido que aprender a utilizarlo. A continuación se detalla el desglose de las horas necesitadas para cada tarea:

La tarea modelado ha necesitado 9 horas para realizarse

- T24 *En el modal de realizar acción, ver el origen y el resultado*: 1 hora y 30 minutos
- T25 *En la acción añadir poder seleccionar ...*: 3 horas
- T26 *En extraer mantener el origen* : 1 hora y 30 minutos
- T27 *Crear un selector que permita ....* : 3 horas

La tarea de elección de la estructura de la base de datos ha durado una suma de 16 horas

- T28 *Obtener la estructura a partir de los datos leídos*: 4 horas y 30 minutos
- T29 *Guardar dicha estructura en la base de datos*: 4 horas y 30 minutos
- T30 *Recuperar dicha estructura de la base de datos*: 4 horas
- T31 *Borrar dicha estructura de la base de datos*: 3 hora y 30 minutos

Id	Descripción	Subtarea	Descripción	Completada al finalizar sprint
HU04	Modelado de datos	T24	En el modal de realizar acción, ver el origen y el resultado	Si
		T25	En la acción de añadir poder seleccionar dónde insertar el dato(delante, detrás, o entremedias por posición absoluta y relativa en comodín)	Si
		T26	En extraer mantener el origen	Si
		T27	Crear un selector que permita seleccionar las posiciones citadas en T26	Si
HU03	Elección de la estructura de la base de datos	T28	Obtener la estructura a partir de los datos leídos	Si
		T29	Guardar dicha estructura en la base de datos	Si
		T30	Recuperar dicha estructura de la base de datos	Si
		T31	Borrar dicha estructura de la base de datos	Si

Cuadro 3.12: *Backlog* sprint 7

3.4.8. Sprint 8

En este sprint también se han añadido tareas a realizar de la historia de usuario HU04 *Modelado de datos*. Además se ha realizado la historia de usuario HU02 *Selección del tipo de datos*. Así como se puede contemplar en el Cuadro 3.13

Este sprint tiene una duración de una semana, es decir de 25 horas de trabajo. En este caso se ha completado todo el sprint. La tarea T32 y T36 son costosas debido que en la tarea T32 se ha tenido que programar de cero ese filtro y además se ha tenido que hacer la parte gráfica de esa función, por otra parte la tarea T36 ha costado de implementar debido a que requería un gran trabajo de análisis y optimización del código existente para que a la hora de simplificar el modal no se produjeran fallos en ninguna otra función ni componente. A continuación se desglosa de manera detallada:

El modelado de datos se ha hecho en 27 horas

- T32 *Crear un filtro automático para seleccionar la posición donde realizar las acciones* : 10 horas
- T33 *En la acción extraer indicar campo resultado*: 1 hora
- T34 *En la acción de reemplazar desactivar el campo origen y activar el nuevo* :1 hora
- T35 *A nivel general las acciones añadir un nuevo campo para el nombre de la columna*: 1

Id	Descripción	Subtarea	Descripción	Completada al finalizar sprint
HU04	Modelado de datos	T32	Crear un filtro automático para seleccionar la posición donde realizar las acciones	Si
		T33	En la acción extraer indicar campo resultado	Si
		T34	En la acción reemplazar desactivar el campo origen y activar el nuevo	Si
		T35	A nivel general en las acciones añadir un nuevo campo para el nombre de la columna	Si
		T36	A nivel general en las acciones simplificar el modal	Si
HU02	Selección del tipo de datos	T37	Crear la funcionalidad que permita seleccionar el tipo de datos	Si
		T38	Crear el selector que permita realizar la selección del tipo de datos	Si

Cuadro 3.13: *Backlog* sprint 8

hora

- T36 *A nivel general en las acciones simplificar el modal*: 8 horas

La selección del tipo de datos ha sido completada en 4 horas

- T37 *Crear la funcionalidad que permita seleccionar el tipo de datos*:3 horas
- T38 *Crear el selector que permita realizar la selección del tipo de datos*: 1 hora

3.4.9. Sprint 9

En este sprint se han realizado más tareas de mejora de la apariencia modal de las acciones a aplicar, es decir, tareas referentes a la historia de usuario HU04 *Modelado de datos* y ha aparecido una nueva historia de usuario (*"Como usuario generador quiero indicar el tipo de separador que quiero aplicar para separar el texto de tal modo que eso facilite su modelado posteriormente"*), que se podría decir que es la evolución de otra que ya apareció en la sección 3.4.4 Sprint 4 (*"Como usuario generador quiero poder separar los datos leídos empleando ',' para poder tratarlos después de manera más fácil"*).

Esto ha sucedido porque en un principio sólo se había pensado en datos que estuvieran separados por " ,". Pero eso limita mucho la aplicación, dado que si el usuario quiere procesar un

Id	Historias de usuario	
HU14	Indicar el tipo de separador del texto	
	Como usuario generador quiero indicar el tipo de separador que quiero aplicar para separar el texto de tal modo que eso facilite su modelado posteriormente	
	Estimación: 5	Valor: 90

Cuadro 3.14: Especificación de la historia de usuario HU14 *Indicar el tipo de separador del texto*

fichero que no esté separado por ",", debería de cambiar de manera manual todos los separadores por ",". Como eso no es viable, es preferible hacer que la aplicación deje elegir el separador y de este modo pueda leer cualquier fichero csv. En el cuadro 3.15 se puede apreciar el *Backlog* del sprint 9, dónde la primera tarea a realizar es la que se acaba de describir.

Este sprint contiene más tareas porque es de 15 días, el doble de lo habitual.Debido a esto, se ha decidido poner en él historias de usuario muy costosas. Las tareas que más han costado han sido: *T43 Refrescar los campos en cada acción*, debido a que en este caso se han tenido que realizar varias funciones, conectarlas entre sí y crear una manera de sincronización para controlar cuando se deben ejecutar y cuando no, *T45 Restablecer los valores de los datos después de cada acción* por las mismas razones que la tarea T43, porque no es algo inmediato, *T47-T50* que son las tareas referentes al tema de la base de datos, en este caso se ha tenido que buscar un poco la manera de realizar las conexiones con la base de datos y de enviarle datos a esta. A continuación se desglosa el tiempo empleado para realizar cada una de las tareas:

La tarea de indicar el separador del texto, ha costado 9 horas en total

- T39 *Desarrollar la función que lea el texto plano sin aplicar ningún separador*: 1 horas
- T40 *Crear la función que permita elegir el separador*: 3 horas
- T41 *Crear la vista que permita visualizar los resultados del separador*: 5 horas

La tarea de modelado de datos se ha realizado en 11 horas

- T42 *Justificar los campos del modal de acciones*: 1 hora
- T43 *Refrescar los campos en cada acción*: 3 horas y 30 minutos
- T44 *Origen y resultado no pueden ser un input*: 15 minutos
- T45 *Restablecer los valores de los datos después de cada acción*: 2 horas y 15 minutos
- T46 *El nombre de los campos tienen que ser incrementares*:4 horas

La tarea de almacenamiento ha sido desarrollada en 30 horas

- T47 *Crear la función que permita conectar el servidor con la base de datos*: 5 horas

Id	Descripción	Subtarea	Descripción	Completada al finalizar sprint
HU14	Indicar el tipo de separador del texto	T39	Desarrollar la función que lea el texto plano, sin aplicar ningún separador	Si
		T40	Crear la función que permita elegir el separador	Si
		T41	Crear la vista que permita visualizar los resultados del separador	Si
HU04	Modelado de datos	T42	Justificar los campos del modal de acciones	Si
		T43	Refrescar los campos en cada acción	Si
		T44	Origen y resultado no pueden ser un input	Si
		T45	Restablecer los valores de los datos después de cada acción	Si
		T46	El nombre de los campos tienen que ser incrementares	Si
HU01	Almacenamiento	T47	Crear la función que permita conectar el servidor con la base de datos	Si
		T48	Crear la función que permita insertar los datos en la base de datos desde el servidor	Si
		T49	Crear la función que le permita a la parte cliente conectar con el servidor	Si
		50	Crear la función que le permita al cliente realizar una petición para almacenar los datos	Si

Cuadro 3.15: *Backlog* sprint 9

- T48 *Crear la función que permita insertar los datos en la base de datos desde el servidor*: 11 horas
- T49 *Crear la función que le permita a la parte cliente conectar con el servidor*: 4 horas
- T50 *Crear la función que le permita al cliente realizar una petición para almacenar los datos* : 10 horas

3.4.10. Sprint 10

Este sprint, según se puede visualizar en el Cuadro 3.17, se ha empleado para realizar mejoras tanto en la historia de usuario *HU04 Modelado de datos*, como en las interfaces que implican a las historias de usuario *HU14 Indicar el tipo de separador del texto* , *HU13 Selección de una fila como cabecera* y *HU12 Eliminar las filas no relevantes para el fichero*, para mejorar la eficiencia y facilitar la realización de dichas operaciones. Asimismo se ha añadido una nueva historia de usuario (*HU15 "Como usuario generador necesito poder automatizar todo el procesamiento de datos para no tener que repetirlo cada vez que quiera añadir nueva información a la base de datos"*).

Id	**Historias de usuario**	
HU15	Automatizar el procesamiento de los datos	
	Como usuario generador necesito poder automatizar todo el procesamiento de datos para no tener que repetirlo cada vez que quiera añadir nueva información a la base de datos	
	Estimación: 10	Valor: 100

Cuadro 3.16: Especificación de la historia de usuario HU15 *Automatizar el procesamiento de los datos*

El tiempo empleado para la realización de este sprint ha sido de una semana, al igual que la mayoría de ellos, 25 horas en total. En cuanto a las tareas, se han logrado realizar todas menos la última, *T64 (Crear la función para cargar el fichero y realizar las operaciones)*, debido a la falta de tiempo para abordarlo todo ya que durante la tarea *T53 (Crear una función para que se puedan anidar los hijos a medida que que se vayan generando)*, surgió un error, que no permitió continuar con la implementación de las tareas hasta que fue solventado.

Además el tema de la historia de usuario *HU15 (Automatizar el procesamiento de los datos)* ha resultado complicado debido a que hubo dificultades con el tema de las funciones *asíncronas*, esto es, el servidor trabaja a su marcha y cuando tiene el resultado, lo devuelve, pero el cliente no se espera a que le devuelva los datos, si no que continua con su ejecución. Entonces eso había que controlarlo para que no diera errores y en un principio se tardó bastante en comprender la lógica y el funcionamiento de estas funciones. En el capítulo 5, se hablará de estas funciones y se describirán con más detalle.

A continuación se puede observar el desglose del tiempo empleado en cada una de las tareas que había que realizar para este sprint:

La tarea de modelado de datos se ha realizado en 10 horas

Id	Descripción	Subtarea	Descripción	Completada al finalizar sprint
HU04	Modelado de datos	T51	En el modal de acciones, el selector por posición tiene que empezar en 1	Si
		T52	Crear una función para que cuando se borre la acción se puedan borrar sus hijos	Si
		T53	Crear una función para que se puedan anidar los hijos a medida que se vayan generando	Si
		T54	Crear el botón de borrado	Si
		T55	Crear un *dialog* de información acerca de donde proviene el hijo	Si
HU12, HU13,HU14	Eliminar las filas no relevantes para el fichero, Selección de una fila como cabecera, Indicar el tipo de separador del texto	T56	Unificar las dos primeras interfaces en una mejor para mejorar la usabilidad	Si
		T57	Juntar el *check button* para seleccionar la cabecera y el botón de borrar líneas	Si
HU15	Automatización del procesamiento	T58	Extraer las peticiones de la base de datos a un fichero único	Si
		T59	Exportar la función de tratado de líneas de csv iniciales al servidor	Si
		T60	Exportar las funciones de formateo de los datos al servidor	Si
		T61	Exportar las funciones de aplicación de acciones al servidor	Si
		T62	Crear los *end-point* para que la parte cliente pueda realizar peticiones a las funciones exportadas al servidor	Si
		T63	Cambiar la lógica del cliente para que pueda realizar peticiones al servidor para procesar los datos que necesite	Si
		T64	Crear la función para cargar el fichero y realizar las operaciones	No

Cuadro 3.17: *Backlog* sprint 10

- T51 *En el modal de las acciones el selector por posición tiene que empezar en 1*: 30 minutos
- T52 *Crear una función para que cuando se borre la acción se puedan borrar sus hijos*: 3 horas
- T53 *Crear una función para que se puedan anidar los hijos a medida que se vayan generando*: 6 horas
- T54 *Crear el botón de borrado*: 15 minutos
- T55 *Crear un dialog de información acerca de donde proviene el hijo*: 15 minutos

La tarea de unificar las dos primeras interfaces (HU12, HU13,HU14) ha costado 2 horas y 45 minutos

- T56 *Unificar las dos primeras interfaces en una mejor para mejorar la usabilidad*: 2 horas y 30 minutos
- T57 *Juntar el check button para seleccionar la cabecera y el botón de borrar líneas* : 15 minutos

La tarea de automatización del procesamiento ha supuesto una duración de 12 horas y 15 minutos

- T58 *Extraer las peticiones de la base de datos a un fichero único*: 15 minutos
- T59 *Exportar la función de tratado de líneas de csv iniciales al servidor*: 2 horas y 30 minutos
- T60 *Exportar las funciones de formateo de los datos al servidor*: 2 horas
- T61 *Exportar las funciones de aplicación de acciones al servidor*: 2 horas
- T62 *Crear los end-point para que la parte cliente pueda realizar peticiones a las funciones exportadas al servidor*: 1 hora y 30 minutos
- T63 *Cambiar la lógica del cliente para que pueda realizar peticiones al servidor para procesar los datos que necesite*: 4 horas

3.4.11. Sprint 11

En este sprint, según se puede apreciar en el Cuadro 3.18, se ha continuado con la tarea T64 *Crear la función para cargar el fichero y realizar las operaciones*, del sprint anterior. Asimismo se han definido ciertas tareas referentes a las historias de usuario HU04 *Modelado de datos* y HU03 *Elección de la estructura de datos.* Finalmente se han realizado las historias de usuario HU05 *Selección de datos*, HU06 *Selección del tipo de gráfico* y HU07 *Realización de las operaciones sobre los datos.*

Id	Descripción	Subtarea	Descripción	Completada al finalizar sprint
HU15	Automatización del procesamiento	T64	Crear la función para cargar el fichero y realizar las operaciones	Si
H04	Modelado de datos	T65	Resolver el bug: Al borrar el último hijo, si se realiza una nueva inserción, ésta no se realiza en el índice adecuado	Si
		T66	Resolver el bug: El índice de donde se va ha realizar la acción no puede ser superior a la longitud del dato	Si
		T67	Resolver el bug: Las acciones por carácter no deben de ser excluyentes	Si
		T68	Las filas deshabilitadas mostrarlas en gris	Si
		T69	Añadir modal para confirmar en todas las acciones de borrado	Si
HU03	Elección de la estructura de datos	T70	Crear la función que permita obtener de la base de datos todas las estructuras (modales)	Si
		T71	Crear la vista que permita mostrar un listado con las estructuras que se han obtenido	Si
		T72	Crear la función para obtener los datos de una determinada estructura	Si
		T73	Crear la interfaz para visualizar la información de una determinada estructura	Si
HU05	Selección de datos	T74	Crear las consultas SQL para obtener los datos desde Superset	Si
HU06	Selección del tipo de gráfico	T75	Configurar los tipos de gráficos necesarios en Superset	Si
		T76	A cada gráfico asignarle una serie de datos para que se representen	Si
HU07	Realizar operaciones sobre los datos	T77	Configurar Superset para que ante un determinado grupo de datos aplique una determinada acción	Si

Cuadro 3.18: *Backlog* sprint 11

La duración de este sprint ha sido de un poco más que una semana, alrededor de 9 días, lo que equivale a 35 hora de trabajo. Todas las tareas han sido completadas con éxito. A continuación se desglosan los tiempos empleados en cada una de ellas:

La tarea de automatización del proyecto ha costado 8 horas de realizar.

- T64 *Crear la función para cargar el fichero y realizar las operaciones*: 8 horas

La tarea del modelado de datos ha necesitado un total de 17 horas para ser completada.

- T65 *Resolver el bug: Al borrar el último hijo, si se realiza una nueva inserción, ésta no se realiza en el índice adecuado*: 7 horas
- T66 *Resolver el bug: El índice de donde se va ha realizar la acción no puede ser superior a la longitud del dato*: 6 horas
- T67 *Resolver el bug: Las acciones por carácter no deben de ser excluyentes*: 3 horas y 15 minutos
- T68 *Las filas deshabilitadas mostrarlas en gris*: 15 minutos
- T69 *Añadir modal para confirmar en todas las acciones de borrado*: 30 minutos

La tarea de elección de la estructura de datos ha durado 4 horas

- T70 *Crear la función que permita obtener de la base de datos todas las estructuras (modales)*: 1 hora
- T71 *Crear la vista que permita mostrar un listado con las estructuras que se han obtenido*: 2 horas y 30 minutos
- T72 *Crear la función para obtener los datos de una determinada estructura*: 1 hora y 15 minutos
- T73 *Crear la interfaz para visualizar la información de una determinada estructura*: 2 horas y 30 minutos

La tarea de selección de datos se ha realizado en 1 hora y 15 minutos

- T74 *Crear las consultas SQL para obtener los datos desde Superset*: 1 hora y 15 minutos

La tarea de selección del tipo de gráfico ha sido completada en 45 minutos

- T75 *Configurar los tipos de gráficos necesarios en Superset*: 30 minutos
- T76 *A cada gráfico asignarle una serie de datos para que se represente*: 15 minutos

La tarea de realizar operaciones sobre los datos ha requerido 15 minutos para completarse

- T77 *Configurar Superset para que ante un determinado grupo de datos aplique una determinada acción* : 15 minutos

3.4.12. Conclusión del seguimiento

Las historias de usuario se han abordado todas. Unas de manera directa y otras de manera indirecta. Aquellas que tienen que ver con el usuario generador se han abordado de manera directa, en cambio aquellas que tiene que ver con el usuario visor, se han logrado de manera indirecta, esto es, las historias de usuario visor se cubren sin necesidad de hacer ninguna tarea explicita, si no completando las historias de usuario HU05, HUO6, HU07 del usuario generador. Debido a que en estas historias de usuario se configuran y generan las gráficas que el usuario visor quiere consultar.

A continuación se va ha mostrar como queda la pila del producto al final del proceso:

Id	**Historias de usuario**	
HU01	Almacenamiento	
	Como usuario generador necesito almacenar los datos que se van a visualizar en el sistema para poder acceder a ellos siempre que lo desee.	
	Estimación: 15	Valor: 80
HU02	Selección del tipo de datos	
	Como usuario generador necesito poder seleccionar de qué tipo van ha ser los datos que se van a almacenar.	
	Estimación: 3	Valor: 60
HU03	Elección de la estructura de datos	
	Como usuario generador necesito poder elegir la estructura de la base de datos que se va ha emplear para almacenar los datos.	
	Estimación: 10	Valor: 70
HU04	Modelado de datos	
	Como usuario generador necesito poder modelar a mi gusto los datos que se van a representar, para mostrarlos según me convenga.	
	Estimación: 40	Valor: 80
HU05	Selección de datos	
	Como usuario generador necesito poder seleccionar los datos sobre los que deseo realizar una gráfica para poder analizar esos datos en concreto	
	Estimación: 2	Valor : 40
HU06	Selección del tipo de gráfico	
	Como usuario generador necesito poder seleccionar el tipo de gráfico con que representar un determinado conjunto de datos	
	Estimación: 1	Valor: 40

Sigue en la página siguiente.

Id	Historias de usuario	
HU07	Realización de operaciones sobre los datos	
	Como usuario generador necesito poder realizar operaciones matemáticas sobre los datos a representar como puede ser la media, el máximo, el mínimo, para poder representarlos en las gráficas.	
	Estimación: 1	Valor: 40
HU08	Filtrado de datos	
	Como usuario visualizador de datos necesito poder realizar un filtrado de los datos que se muestran para poder elaborar un mejor análisis de estos .	
	Estimación: 1	Valor: 20
HU09	Gráfico de barras	
	Como usuario visualizador de datos necesito poder ver un gráfico de barras que resuma la información de los datos tratados para analizar.	
	Estimación: 1	Valor: 20
HU10	Leer un fichero con extensión csv	
	Como usuario generador necesito poder leer los datos de un fichero csv	
	Estimación: 4	Valor: 100
HU11	Separación del texto empleando ”,”	
	Como usuario generador quiero poder separar los datos leídos empleando ”,”para poder tratar los datos después de manera más fácil	
	Estimación: 6	Valor: 90
HU12	Eliminar las filas no relevantes del fichero	
	Como usuario generador de dashboards necesito poder eliminar las filas no relevantes del fichero leído para reducir la cantidad de datos a tratar	
	Estimación: 3	Valor: 100
HU13	Seleccionar una fila como cabecera	
	Como usuario generador de dashboards necesito poder seleccionar una fila como cabecera para poder generar las columnas de la tabla de la base de datos con los nombres adecuados	
	Estimación: 8	Valor: 90
HU14	Indicar el tipo de separador del texto	
	Como usuario generador quiero indicar el tipo de separador que quiero aplicar para separar el texto de tal modo que eso facilite su modelado posteriormente	
	Estimación: 5	Valor: 90
HU15	Automatizar el procesamiento de los datos	
	Como usuario generador necesito poder automatizar todo el procesamiento de datos para no tener que repetirlo cada vez que quiera añadir nueva información a la base de datos	
	Estimación: 10	Valor: 100

Cuadro 3.19: Pila del producto final del proyecto

Tal y como se puede apreciar en el Cuador 3.19, la pila del producto del proyecto ha cambiado con respecto a la definición inicial. Se han incluido 6 historias de usuario que han ido apareciendo a medida que se ha ido avanzando en los sprints. Desde la HU10 hasta la HU15. Además la estimación ha cambiado con respecto a la estimación inicial. Esto demuestra que a lo largo del desarrollo de un proyecto pueden aparecer cambios, que pueden hacer que algunas historias de usuario se atrasen o que se adelanten. Además, a medida que el proyecto avanzaba el alumno en prácticas ha ido adquiriendo mayor conocimiento acerca de las tecnologías empleadas, cosa que ha hecho que tarde menos en realizar ciertas tareas.

Capítulo 4

Análisis y diseño del sistema

En este capítulo se va ha describir la fase de diseño y análisis de la herramienta. Se comenzará con una explicación detallada de su diseño de datos. Posteriormente se continuará con la arquitectura y el estilo arquitectónico que ésta sigue. Finalmente se concluirá con el diseño de las interfaces, realizando una comparativa de cómo planificaron en un principio y como resultaron ser al final.

4.1. Diseño de datos o clases

El diseño intenta presentar las entidades que serán necesarias y cómo estas se relacionan entre ellas para que su desarrollo satisfaga las necesidades del cliente. En la Figura 4.1 se muestra el correspondiente diagrama de clases de la aplicación.

En el diagrama no se representa la estructura de la base de datos, aunque en el caso de la presente aplicación son necesarias dos bases de datos para albergar toda la información necesaria. Por una parte se almacenan los datos en PostgreSQL y por otra la estructura de estos se guarda en MongoDB.

En PostgreSQL siempre hay una tabla que no cambia y otra que varía en función de la estructura elegida en la aplicación. De la segunda tabla, tanto el nombre de las columnas como el tipo de los datos que estas albergan son seleccionados en la aplicación.

La primera tabla guarda las extensiones de teléfono, las coordenadas y el nombre de las provincias de España. En el listado 4.1 se muestra su diseño físico.

Figura 4.1: Diagrama de clases de la aplicación

```
1 CREATE table extensiones (
2    municipio varchar,
3    numero varchar,
4    coordenadaX DECIMAL,
5    coordenadaY DECIMAL,
6    PRIMARY KEY (extension)
7    );
```

Listing 4.1: Diseño físico de la tabla extensiones

En MongoDB se almacena la estructura de todo el modelado de datos, esto es, de que fichero provienen los datos, el separador se ha empleado para dividirlos, el nombre que recibe ese modelado, la fecha de creación, los indices que no son necesarios y los componentes que conforman en sí el modelo. En la Figura 4.2 se puede visualizar el contenido de la base de datos mogoDB, así como la estructura de los datos albergados.

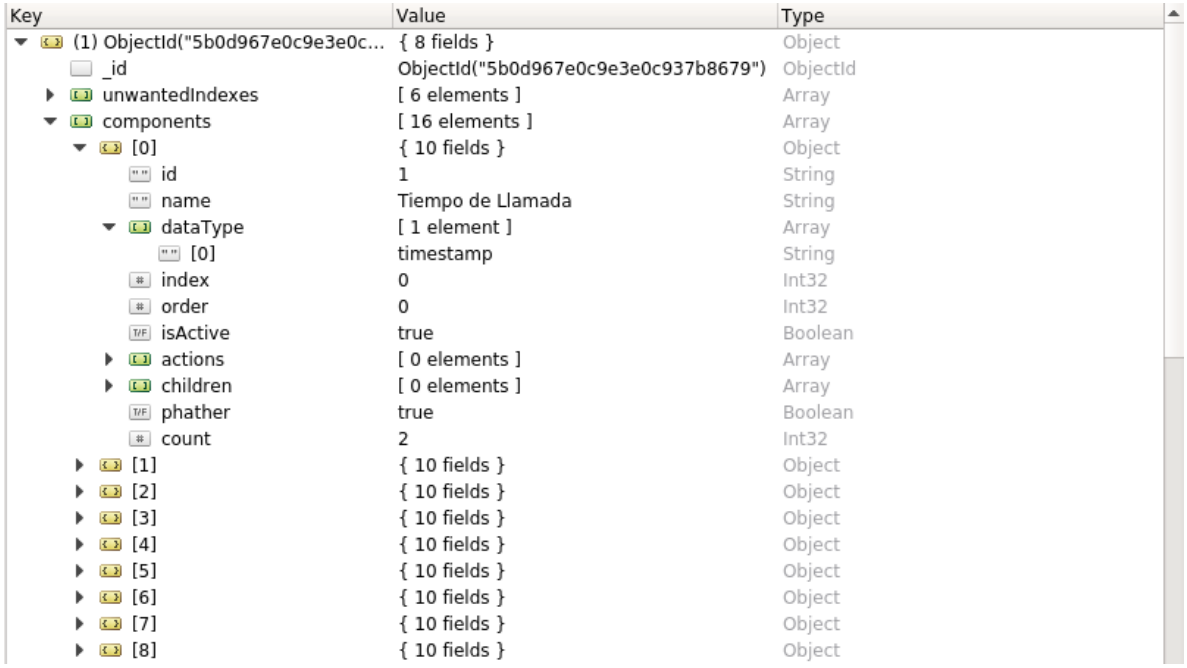

Figura 4.2: Contenido de MongoBD

4.2. Diseño de la arquitectura del sistema

La arquitectura de un sistema informático describe la estructura de los datos y componentes de la aplicación necesarios para desarrollarla. Esto es, constituye el estilo que poseerá el sistema, la estructura , las características de los componentes que lo forman y la relación entre estos.

En suma, la arquitectura de un sistema es una vista de la estructura de alto nivel que determina el estilo o la combinación de estilos para una concreta solución.

Por ello, este proceso es vital para el éxito de un proyecto.

La importancia del diseño de la arquitectura erradica en los siguientes tres puntos:

- Las representaciones de la arquitectura favorecen la comunicación entre todos los implicados en el desarrollo del proyecto.

- Resalta las decisiones prematuras que tendrán un profundo impacto en el proceso de desarrollo y que finalmente determina el éxito del sistema.

- Utilizar un diseño de la arquitectura de proporciones "reducidas" favorece en la comprensión de la estructura del sistema, permitiendo ver los componentes que conforman el sistema, su funcionamiento y el trabajo desempeñado por todos estos, ya sea de manera independiente o conjunta.

4.2.1. Estilo arquitectónico

Existe una gran variedad de estilos arquitectónicos, entre los que destacan:

- Centrada en los datos: Esta construida al rededor de un almacén y los clientes operan de manera independiente.

- Orientada a objetos: Los componentes encapsulan tanto los datos como las operaciones que se emplean para tratarlos.

- Cliente/servidor: El cliente realiza una petición al servidor y éste le responde con los datos solicitados

- Flujo de datos: Los datos de entrada necesitan transformarse a través de componentes que los modifican.

Normalmente las arquitecturas de los sistemas suelen estar compuestas de varios estilos arquitectónicos. En este caso la arquitectura del proyecto se basa en la combinación de dos estilos arquitectónicos, por una parte cliente/servidor y por otra flujo de datos.

El cliente realiza peticiones al servidor empleando el protocolo **HTTP**. Las peticiones van desde aplicar acciones a solicitar datos, es decir, el cliente puede realiza peticiones para aplicar alguna acción sobre un conjunto de datos, el servidor recoge estos datos, les aplica las acciones solicitadas y se los devuelve al cliente. Por ejemplo, el cliente pide que a un conjunto de datos se les aplique el formato, el servidor aplica el correspondiente formato a cada dato que el cliente le ha transferido y se los envía. En cuanto a la solicitud de datos, el cliente puede solicitar los datos almacenados en la base de datos, entonces el servidor realiza una petición a la base de datos y los envía.

En este proyecto los datos de entrada no están conforme se necesitan, por lo cual, son transformados a través de los distintos componentes que conforman la aplicación. Asimismo éstos son independientes entre sí, es decir, que sólo necesitan de los datos para poder trabajar.

4.2.2. Arquitectura del sistema

El sistema lo conforman 3 subsistemas claramente diferenciados como se comenta en el capítulo 2 (Descripción del proyecto). Se puede ver representado claramente en la Figura 2.6, donde se percibe el sistema completo como un conjunto de subsistemas que interactúan entre ellos para alcanzar una meta común, la satisfacción del usuario.

A continuación se va a explicar cada una de estas partes:

- **Lectura de datos**: Esta parte se encarga de leer y procesar los datos. Estos se leen a través de un fichero con extensión csv y son tratados mediante acciones que el cliente delimita. Las acciones las aplica el servidor, por lo tanto, el cliente realiza una petición dónde le solicita al servidor que realice alguna acción sobre una serie de datos que le pasa. El servidor realiza esas acciones y el resultado se lo envía al cliente.
- **Almacenamiento de datos** o **servidor**: Este subcomponente recibe las peticiones que realiza el cliente y devuelve el resultado. Además se encarga de realizar el almacenamiento y la recuperación de los datos.
- **Visualización de datos**: Lee los datos que el servidor almacena en la base de datos y los visualiza en sus dashboards. Se trata de un componente que no ha sido desarrollado por el alumno de la estancia en prácticas. Es una tecnología que se configura para la visualización de los datos.

Si se desea, se puede acudir al capítulo 2 (Descripción del proyecto), dónde se explica más a fondo cómo están conectados estos subsistemas y cómo funcionan.

4.3. Diseño de la interfaz

Las interfaces de esta aplicación se han ido realizando a medida que se avanzaban los *sprints*. Inicialmente se tenía un diseño básico para tener una idea de cómo era la estructura básica de esa interfaz. Posteriormente en cada iteración de los sprints, se refinan más y más, añadiendo funcionalidades gráficas, añadiendo botones, cambiando de posición ciertos componentes gráficos o incluso fusionando ciertas interfaces dejando solo una como resultante.

4.3.1. Diseño iniciales

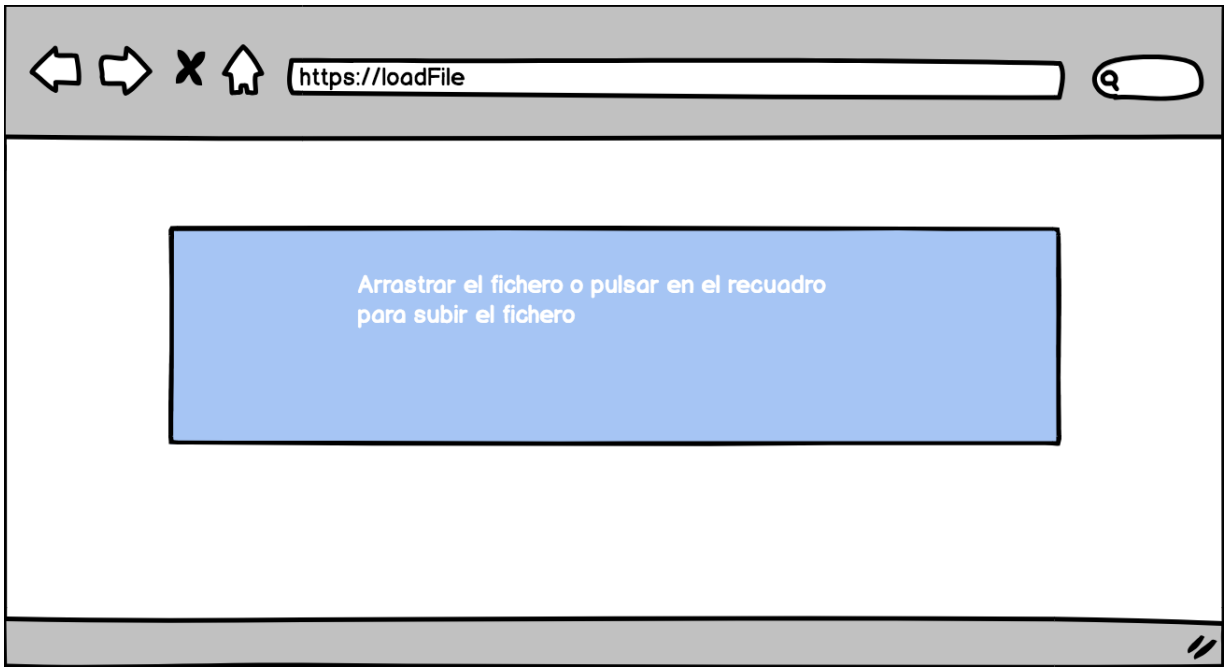

Figura 4.3: Diseño de la interfaz de carga del fichero

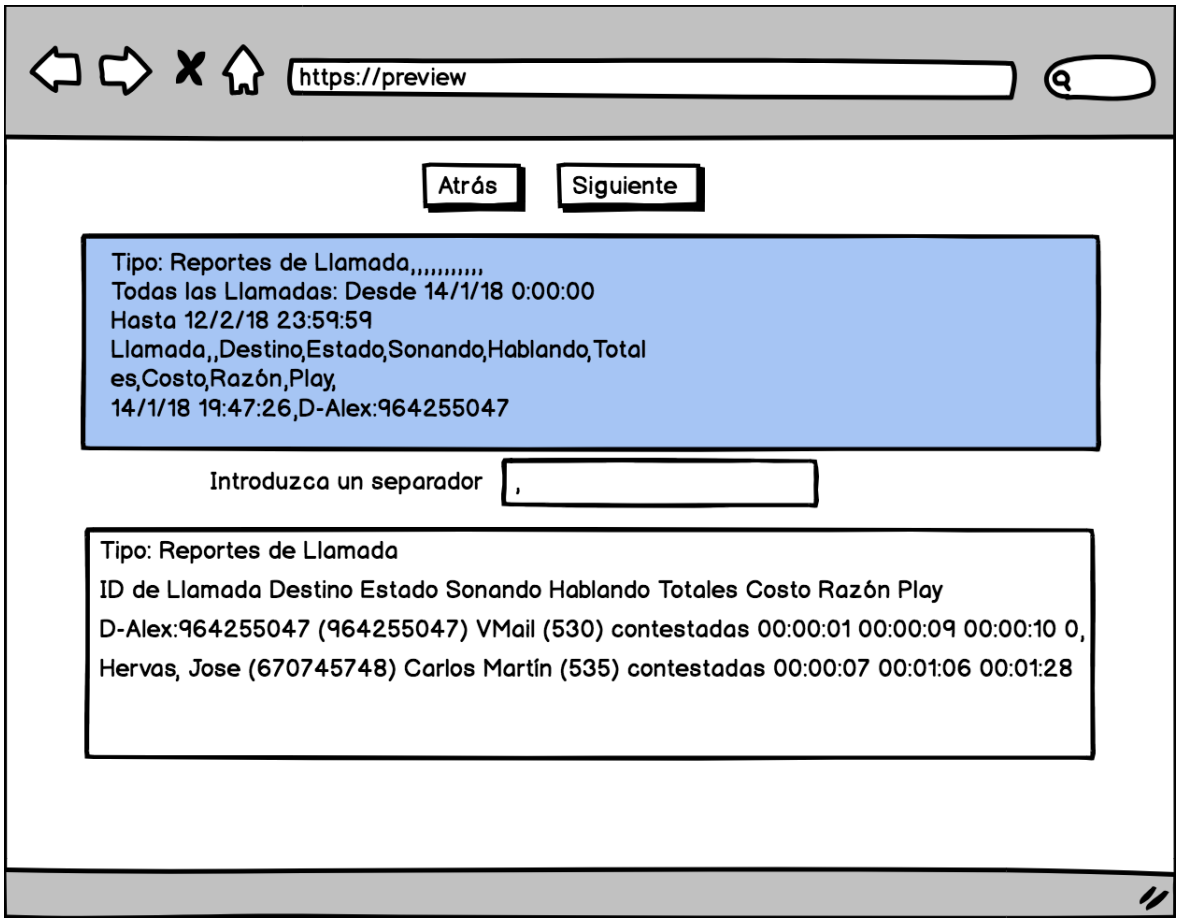

Figura 4.4: Diseño de la interfaz de previsualización del contenido del fichero

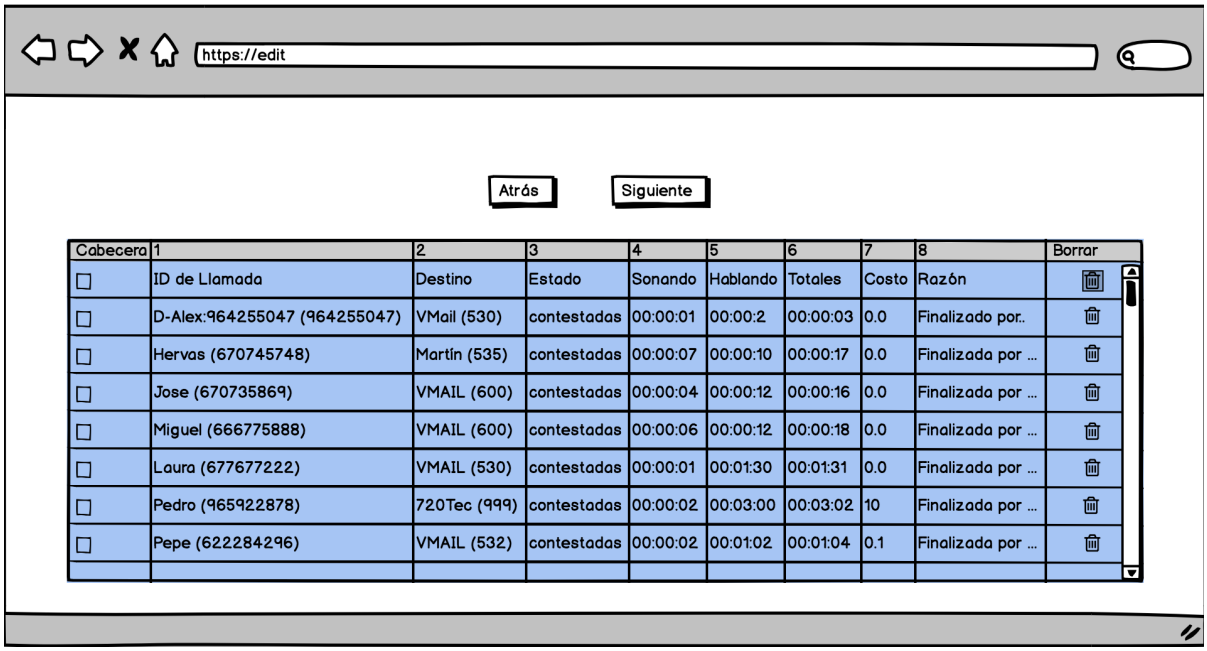

Figura 4.5: Diseño de la interfaz del filtrado del contenido

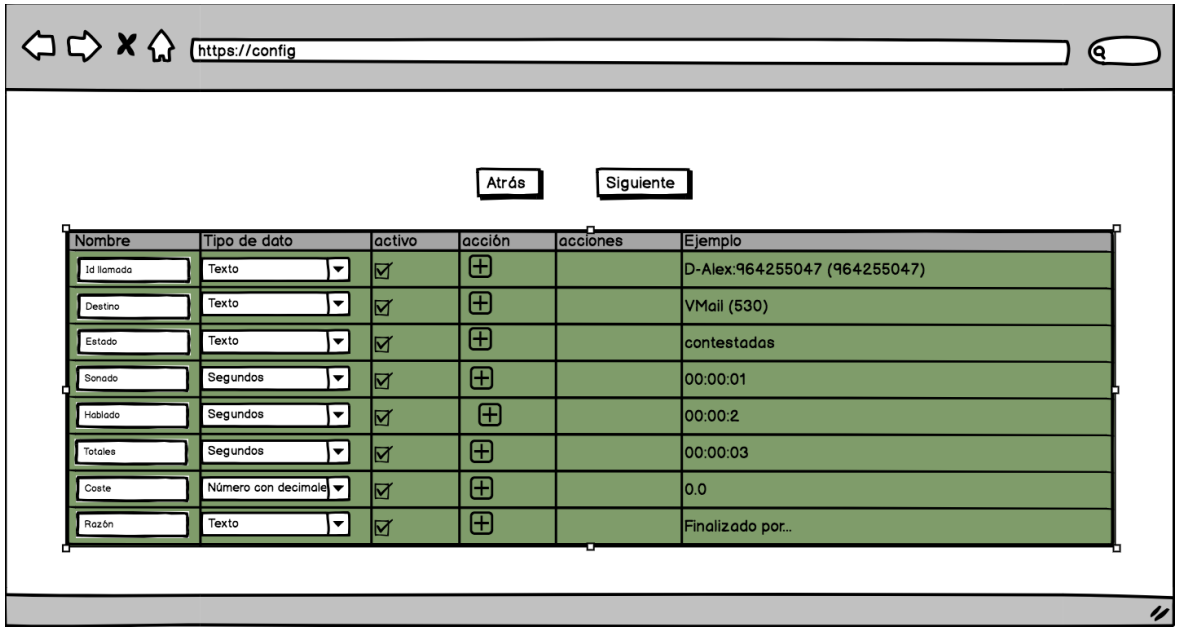

Figura 4.6: Diseño de la interfaz de modelado de los datos

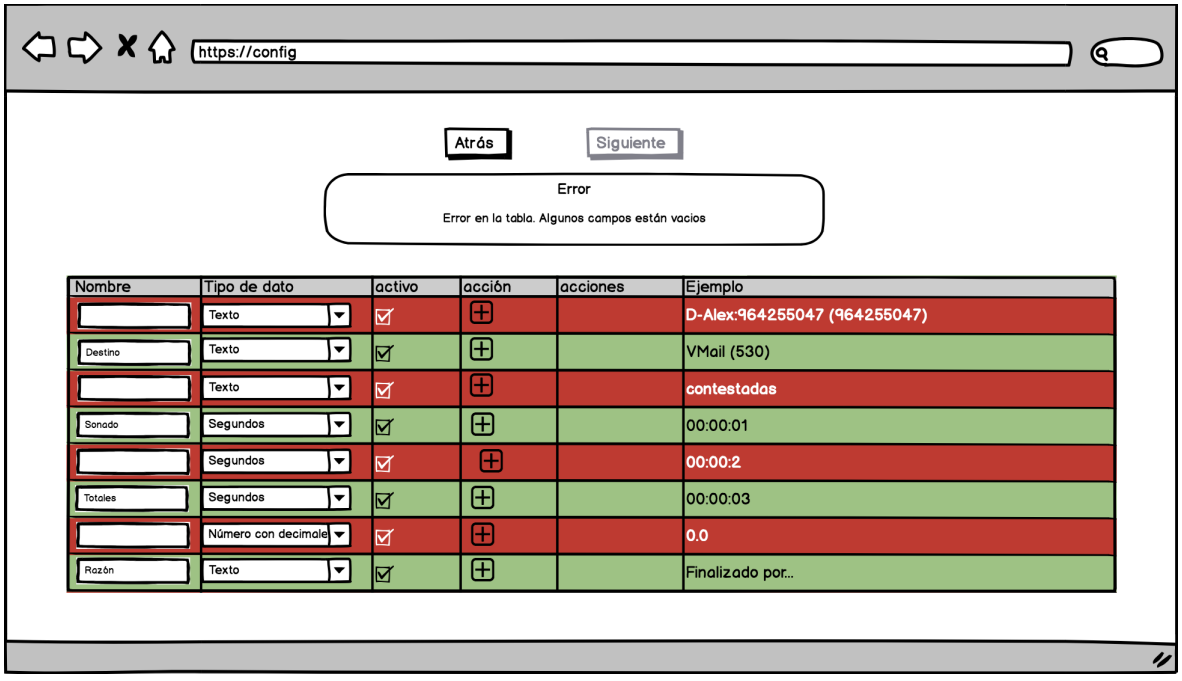

Figura 4.7: Diseño de la interfaz de modelado de los datos con errores

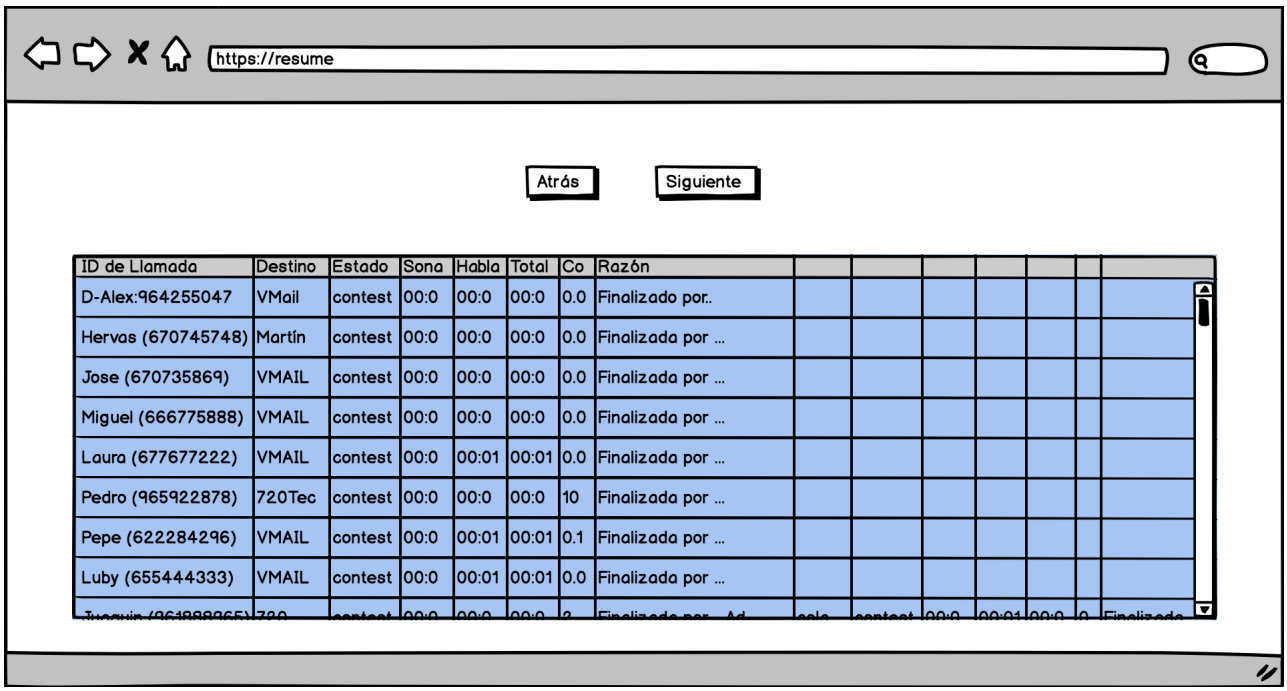

Figura 4.8: Diseño de la interfaz de resumen de los datos ya tratados

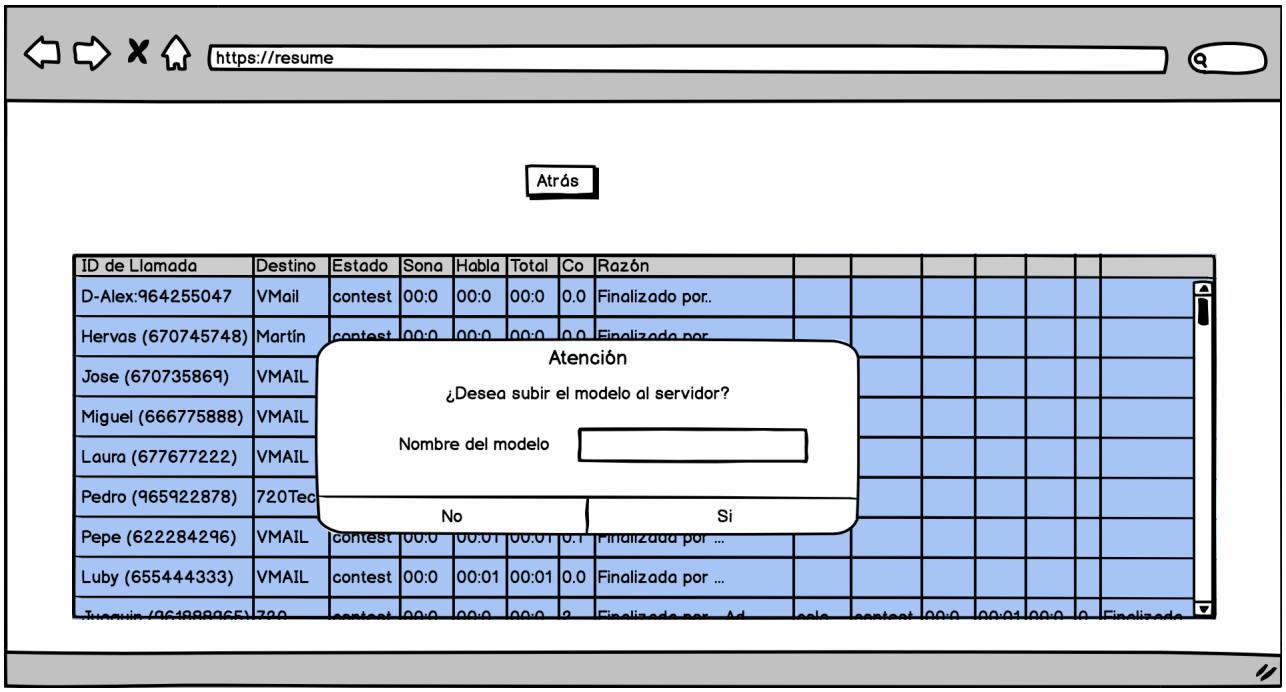

Figura 4.9: Diseño de la interfaz de subida de los datos al servidor

4.3.2. Interfaces finales

Como ya se puede apreciar las interfaces finales siguen un poco la estructura que se acordó en un principio, pero presentan bastantes cambios en torno a los estilos y a pequeños detalles de diseño. La diferencia más clara entre las interfaces iniciales y finales, es la fusión de las interfaces de previsualización (Figura 4.4) del contenido y filtrado del contenido (Figura 4.5) que da como resultado la interfaz de previsualización del contenido (Figura 4.14).

Otra diferencia notable es el orden de los elementos en la interfaz de modelado de datos.

En un principio estaba organizada de tal modo que en primer lugar se encontraba el nombre, en el segundo el selector del tipo de datos, en el tercero un check de si esa fila esta activa o no, en el cuarto un botón para abrir el modal de aplicar acciones, en el quinto un resumen de las acciones aplicadas y finalmente un ejemplo del dato tratado conforme indican todos los componentes anteriores. Según se percibe en la Figura 4.6

Finalmente se organizó de la siguiente manera. En primera posición se ha introducido un identificador, algo nuevo, que no se había tenido en cuenta en un principio. A continuación puso el check para saber si esa fila esta activa o no. Posteriormente se situó el nombre. Por consiguiente se introdujo el selector de los tipos de datos. A posteriori se incorporó el botón para abrir el modal de las acciones. Después se puso el resumen de las acciones aplicadas, que en este caso se ha cambiado el nombre de la columna, en vez de llamarla acciones, se le ha llamado Modelado. Seguidamente se agregó la columna que muestra un ejemplo de los datos con todas las acciones aplicadas, en este caso se le ha cambiado el nombre a "Dato tratado". Finalmente se muestra un botón de borrar, en este caso solo aparece en aquellas filas que se han generado a través de una acción, en aquellas que son originales del propio csv no aparece. Todo lo mencionado se puede apreciar en la Figura 4.16

También se han introducido dos nuevas interfaces. La primera (*Listado de los modelos almacenados en el servidor* Figura 4.10), como su nombre indica es una interfaz que muestra un listado de todos los modelos que se han guardado en la base de datos. La segunda, *Información de un modelo* (Figura 4.11), según indica su nombre muestra información de un determinado modelo.

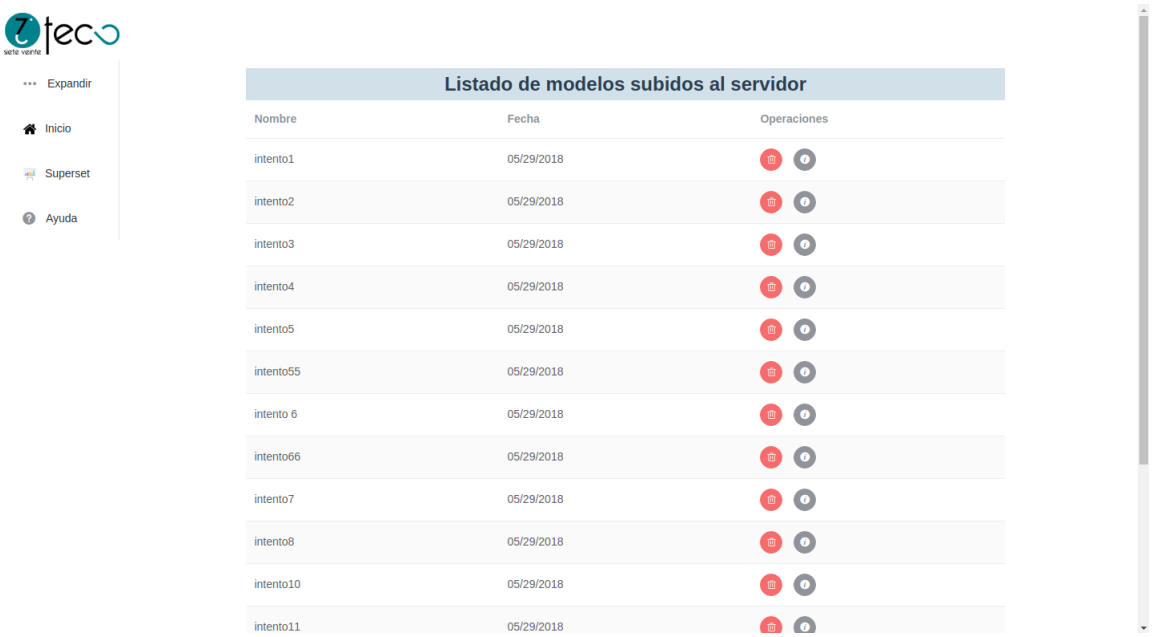

Figura 4.10: Interfaz del listado de modelos que hay en la base de datos

Figura 4.11: Interfaz de la información de un determinado modelo

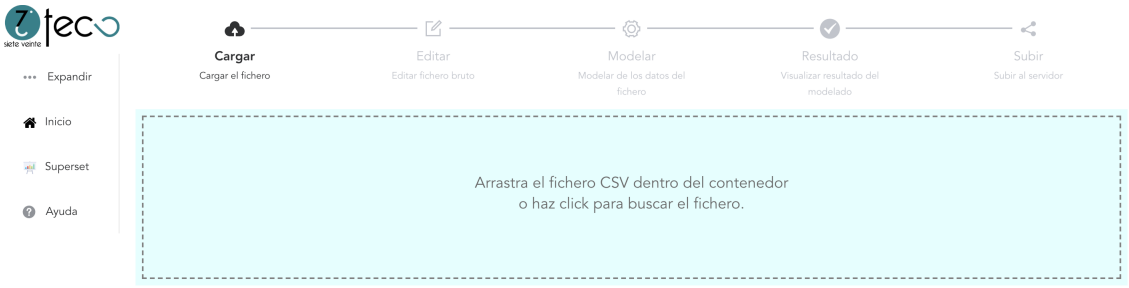

Figura 4.12: Interfaz de subida de los datos

Figura 4.13: Interfaz de previsualización del contenido

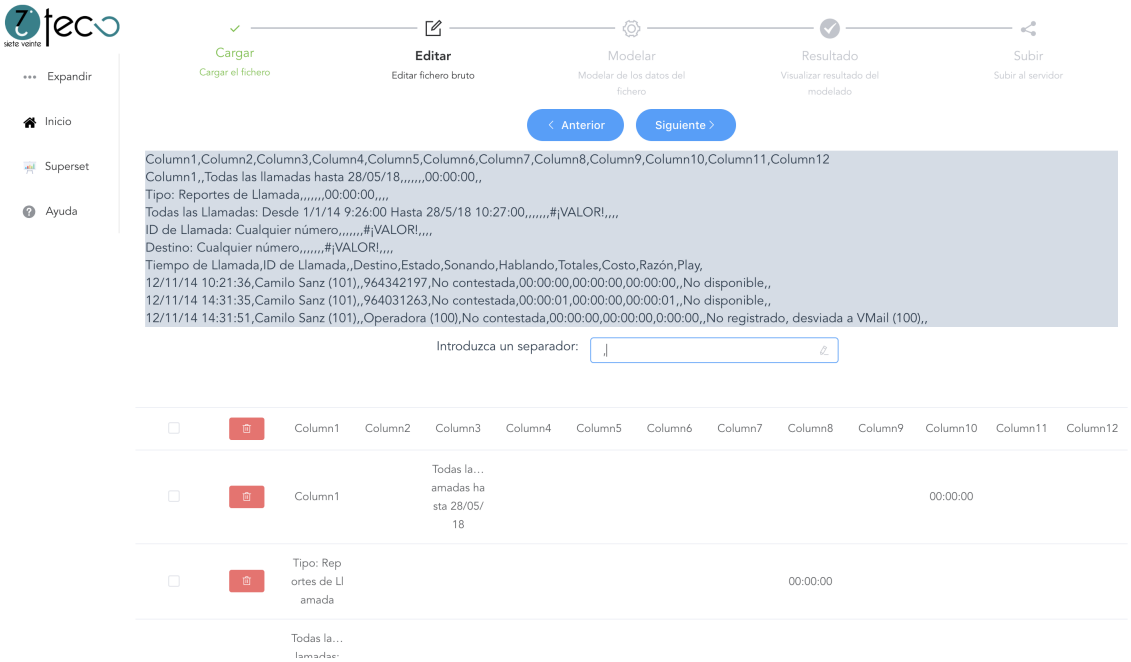

Figura 4.14: Interfaz de previsualización del contenido (Aplicando el separador)

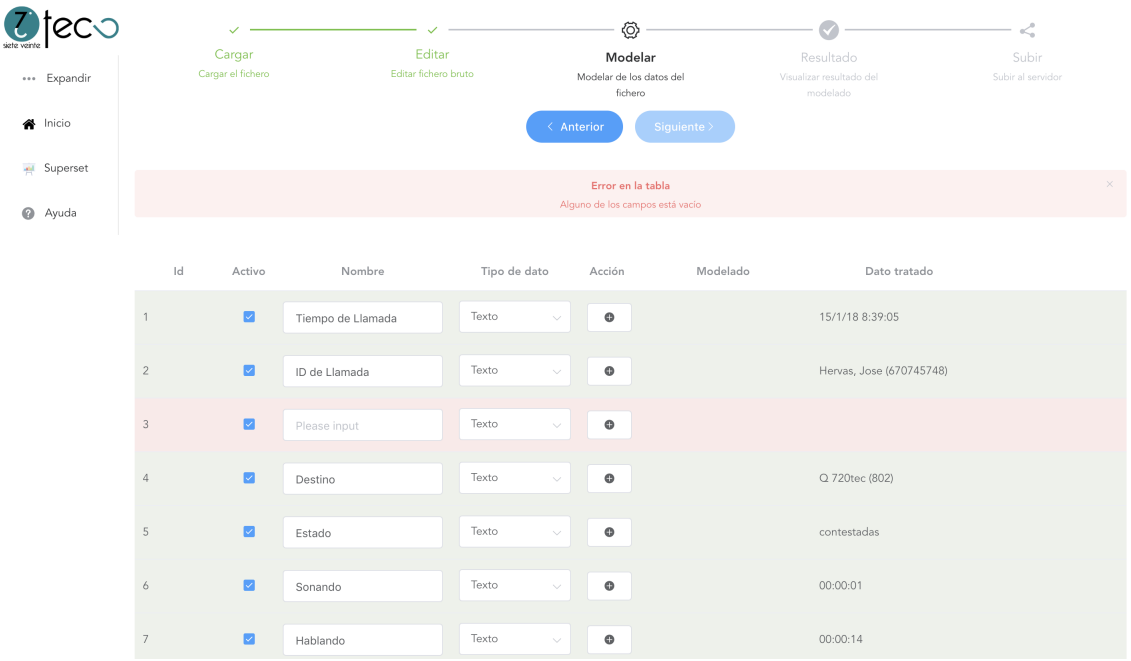

Figura 4.15: Interfaz de modelado de datos, inicial con errores

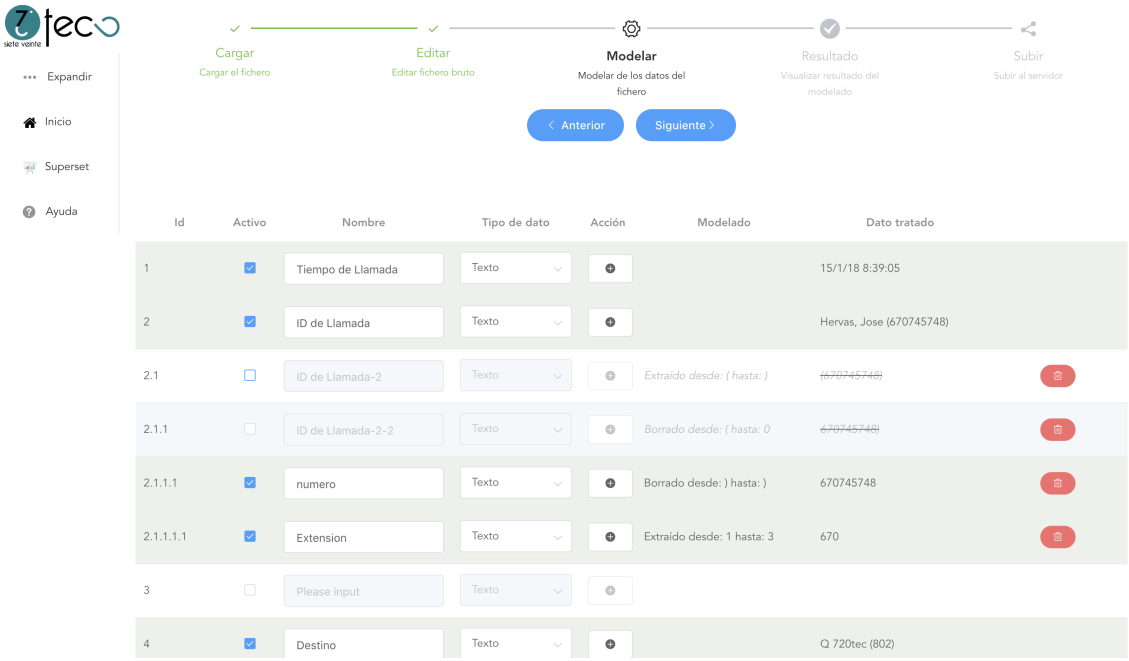

Figura 4.16: Interfaz de modelado de datos

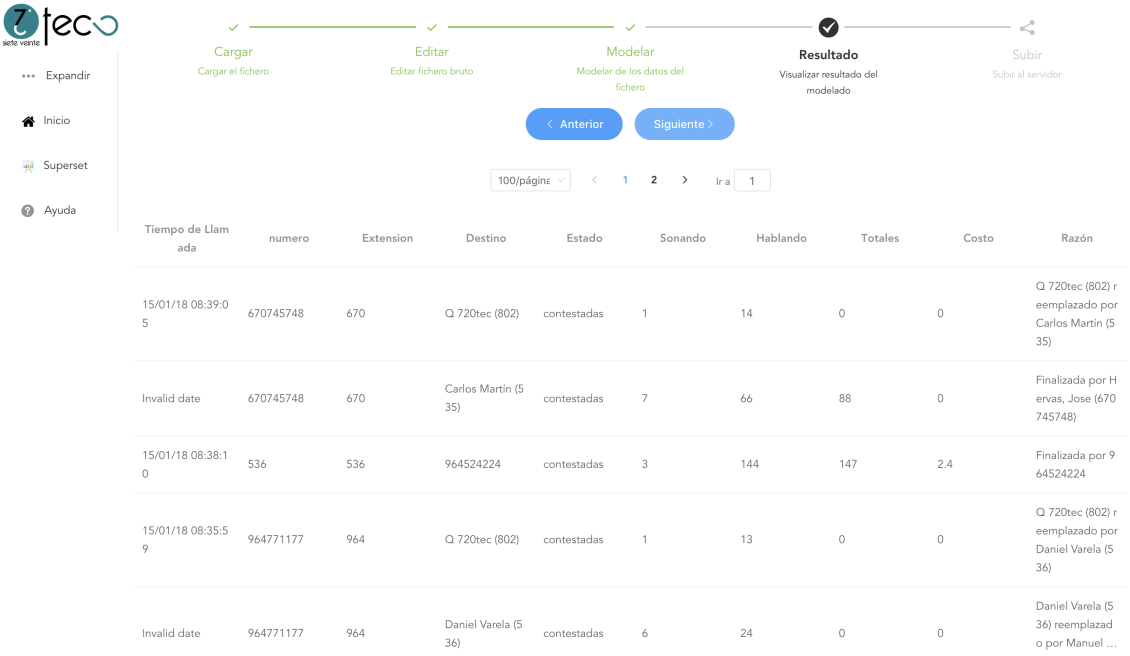

Figura 4.17: Interfaz del resumen de datos ya tratados

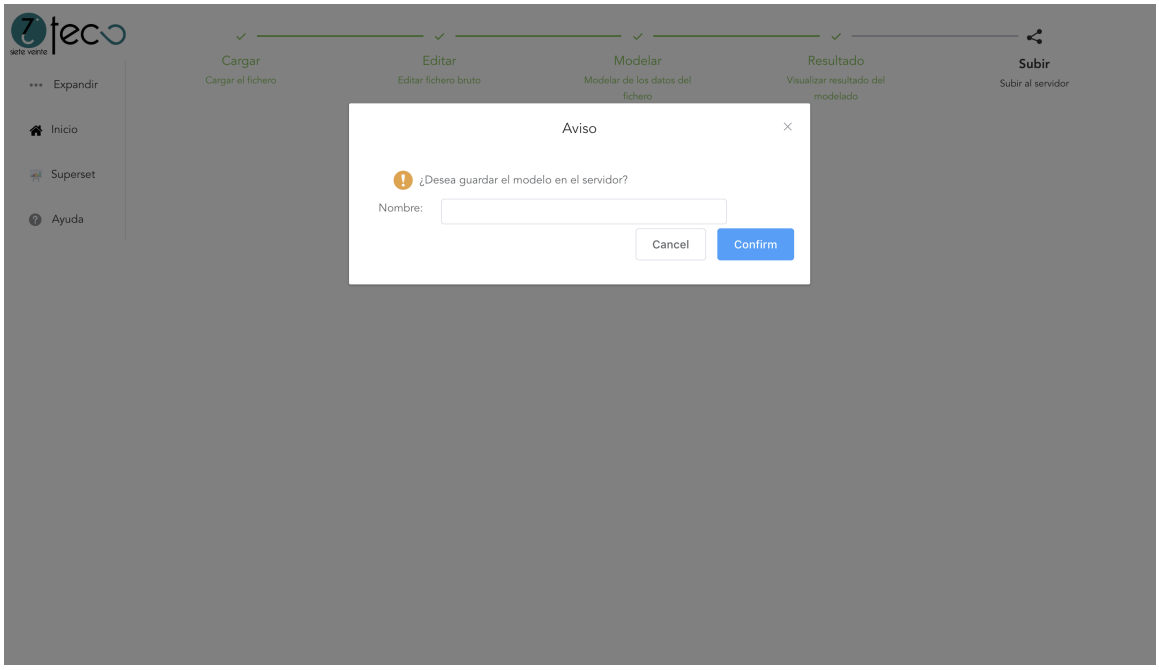

Figura 4.18: Interfaz de subida de los datos al servidor

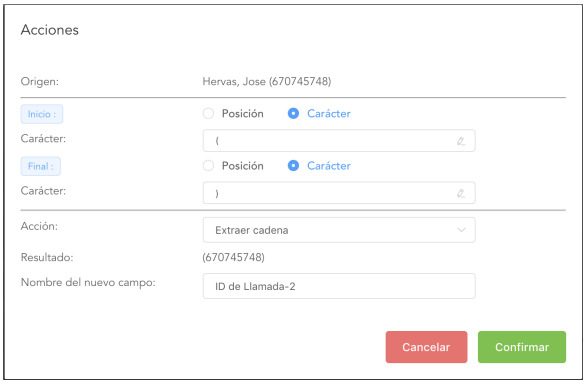

Figura 4.19: Interfaz de la realización de una acción

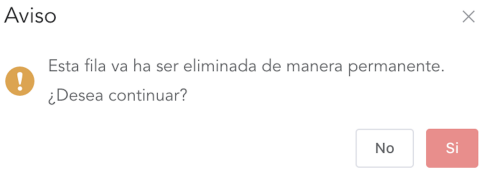

Figura 4.20: Interfaz de confirmación de borrado

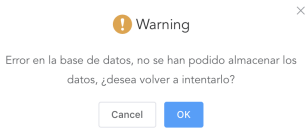

Figura 4.21: Interfaz de error producido en la base de datos

Figura 4.22: Tipos de datos que se pueden seleccionar

4.3.3. Superset

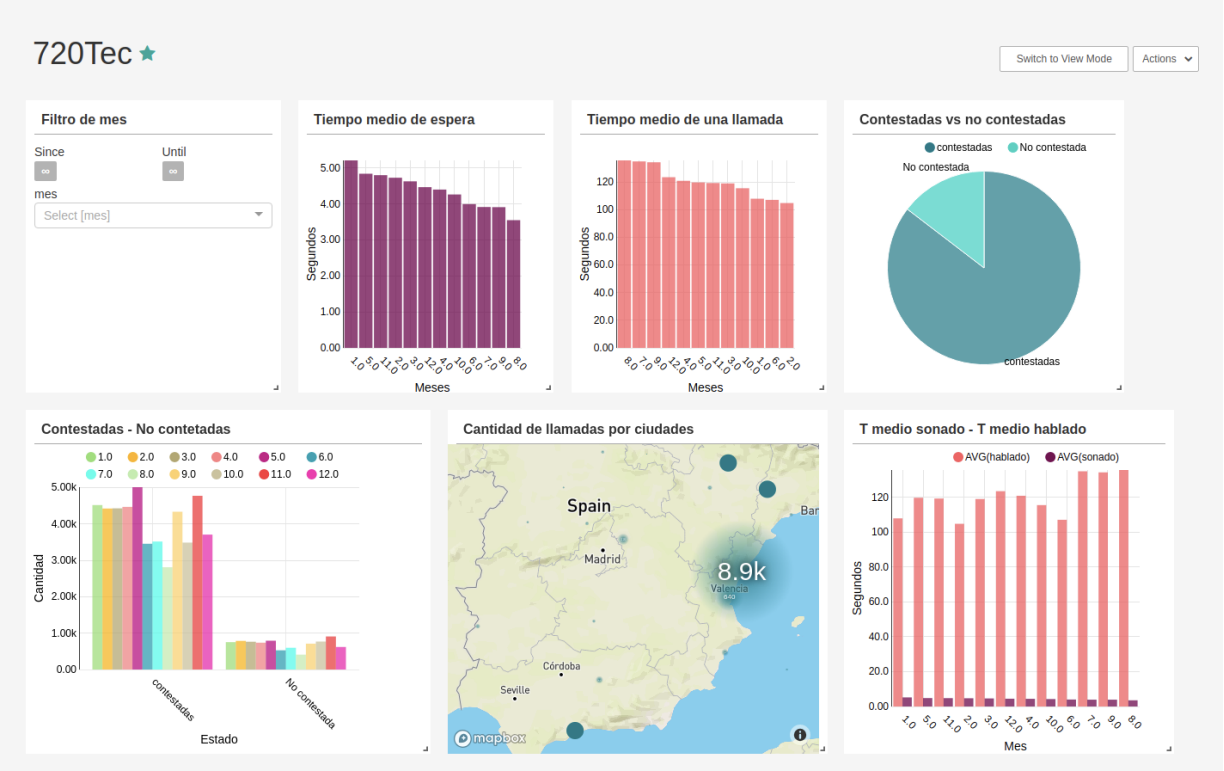

Figura 4.23: Dashboard completo de Superset

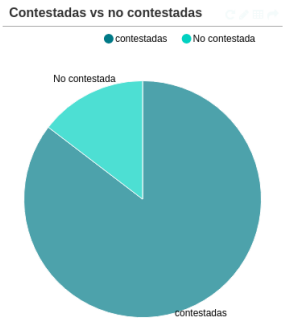

Figura 4.24: Comparación de llamadas contestadas y no contestadas

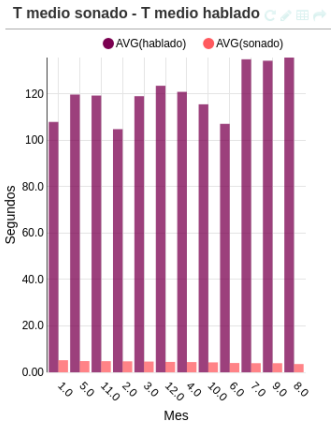

Figura 4.25: Comparación del tiempo medio sonado y hablado en una llamada

Figura 4.26: Filtro por meses

Figura 4.27: Mapa con las provincias de donde se han recibido o realizado llamadas

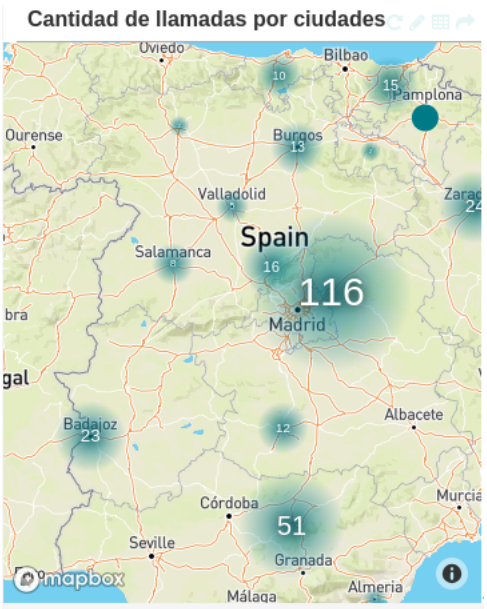

Figura 4.28: Mapa con las provincias de donde se han recibido o realizado llamadas (Detallado)

Capítulo 5

Implementación y pruebas

En este capítulo se describe la implementación que se ha llevado a cabo para desarrollar el sistema en cuestión. Asimismo, también se detallan las medidas y herramientas empleadas para validar y verificar el correcto funcionamiento del código programado.

5.1. Detalles de la implementación

En este apartado se detallan todas las características de la implementación así como la estructura que se ha seguido para organizar los distintos componentes del proyecto.

Hay que recordar que este proyecto sigue una arquitectura cliente/servidor, por lo tanto el proyecto en sí está formado por dos subproyectos que se comunican empleando el protocolo **HTTP**.

Asimismo, tampoco hay que olvidar la existencia de una tercera parte que le da la visualización al proyecto. Esta parte, al contrario que las otras dos, no se comunica de manera directa con ninguna de ellas, si no que lo hace con a la base de datos PostgreSQL.

En la Figura 5.1 se detalla cómo interactúan los distintos componentes entre sí. El cliente web está conformado por Vuejs y el store (subsección 5.1.1 Cliente), emplea *Axios* [1] para realizar las correspondientes peticiones al servidor. El servidor está implementado con Nodejs y recibe las peticiones del cliente a través de Express [2] (subsección 5.1.2 Servidor). Éste conecta con dos bases de datos. MongoDB sirve para almacenar las acciones realizadas sobre los componentes, mientas que PostgreSQL alberga los datos tratados. Superset lee los datos de la base de datos PostgreSQL y los representa mediante gráficas en dashboards (subsección 5.1.3 Superset).

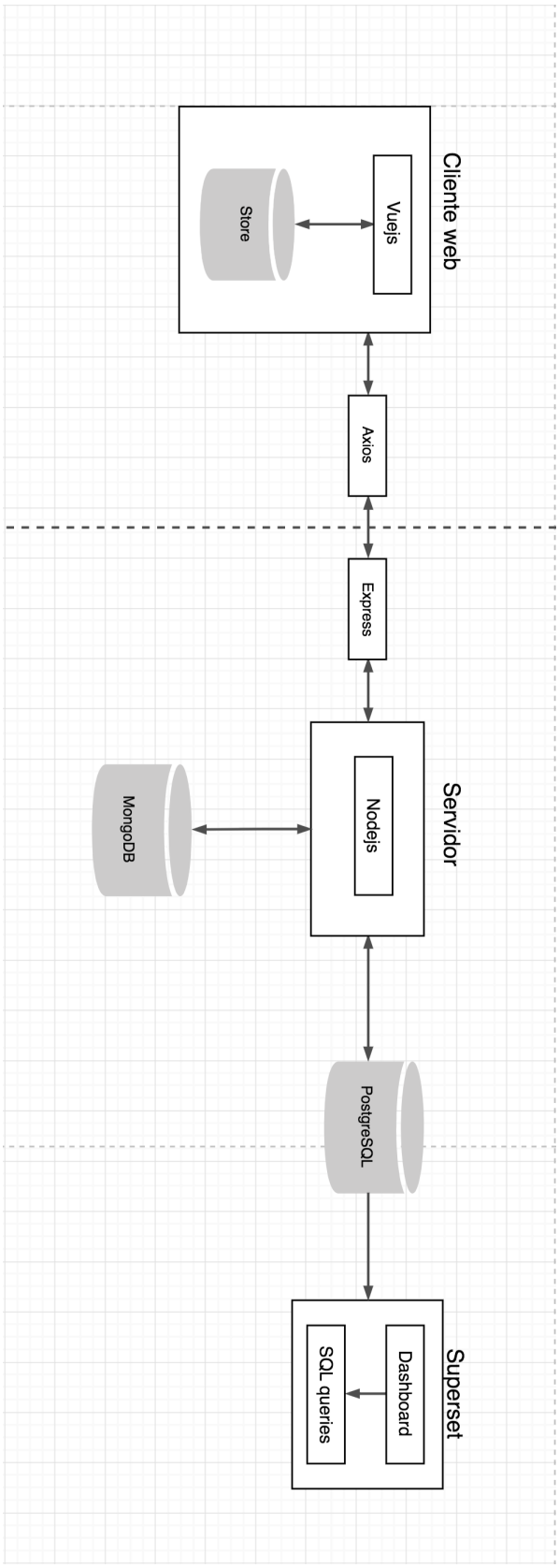

Figura 5.1: Diagrama del proyecto

5.1.1. Cliente

El cliente se ha desarrollado empleando la librería (framework) Vuejs, que ofrece una estructura propia para proyectos web.

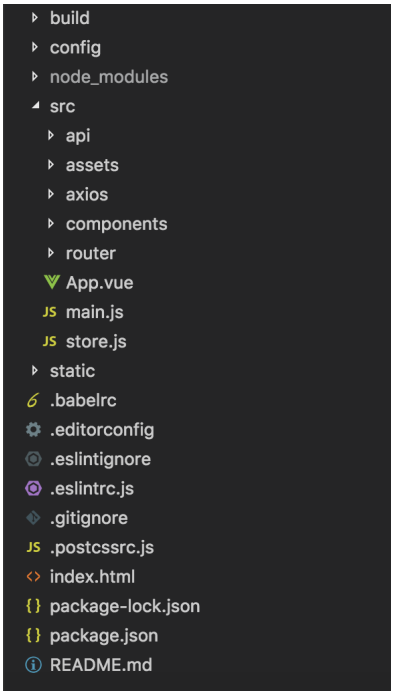

Figura 5.2: Estructura del proyecto

En la Figura 5.2 se puede apreciar como está estructurado el proyecto. En la carpeta *src* se encuentra todo lo referente a los componentes de la aplicación. Dentro de este directorio se encuentran:

- *api*: abarca una serie de ficheros JavaScript que sirven para enviar las peticiones al servidor. Este directorio no viene por defecto, si no que ha sido creado por el alumno en prácticas. Esto es así porque de esta manera se evita tener un fuerte "acoplamiento" con las llamadas al servidor, si por algún motivo alguna llamada cambia, simplemente hay que cambiarlo en un fichero y no en toda la aplicación.
- *assets*: Contiene las dependencias externas necesarias para el proyecto. En este caso contiene la imágen del logotipo de la empresa.
- *axios*: Incluye un fichero de configuración de *Axios*, un cliente HTTP de Vuejs. Axios es muy importante para el proyecto, debido a que es el encargado de realizar todos las peticiones que el cliente le realiza al servidor.
- *components*: Abarca los componentes de la aplicación. Se trata de un directorio predefinido por Vuejs.
- *router*: Contiene el fichero *index.js*, el cuál referencia todos los componentes, asignándoles una ruta.

- *App*: Este componente contiene todos los estilos, funciones, subcomponentes que se quieren gastar en toda la aplicación. En este caso contiene el menú principal de la aplicación, que es el mismo en todas las interfaces.
- *main*: Es el fichero en el que se incluyen todas las dependencias comunes necesarias para el proyecto. Aunque cada componente puede definir sus propias dependencias.
- *store*: Este fichero es el "almacén" de los datos que se procesan en la aplicación.

De todo lo que se ha citado anteriormente, tanto *Axios* como *store* son de vital importancia para el proyecto.

Axios, es una biblioteca que está muy integrada con Vuejs para hacer e interceptar peticiones *XMLHttpRequest*[1], en otras palabras peticiones Ajax[2] para la consecutiva transformación y uso de los datos en formato JSON.

```
29  static remove (name) {
30    axios.delete(`/table/${name}`)
31      .then(function (response) {
32        console.log(response.data)
33      })
34      .catch(function (error) {
35        console.log(error)
36      })
37  }
```

Figura 5.3: Función de borrado de una tabla de la base de datos

La función que se puede visualizar en la Figura 5.3, es un método que engloba a una función de *Axios*. Este método se encarga de borrar una determinada tabla de la base de datos. Para ello, llama a la función *delete* de Axios pasandole la ruta de la tabla que quiere borrar. Se trata de una función asíncrona, con lo cual, la respuesta la devuelve en un *callback* en forma de función. Si la petición se ha realizado de manera correcta se ejecuta la parte del *"then(function (response)"*, donde la variable *response* contiene la respuesta a la petición realizada. En cambio si ha habido algún fallo salta la parte del código *"catch(function(error))"*, en este caso la variable *error* contiene la información del error que se ha podido producir durante la petición.

Los métodos principales de Axios son:

- get: Método empleado para obtener los datos, puede recibir parámetros en el cuerpo de la llamada.
- post: Método para enviar los datos, en este caso es obligatorio pasarle los datos para poder realizar la petición. Normalmente suele usarse para actualizar los datos.
- put: Método que se emplea para enviar los datos también, en este caso suelen ser datos que se van a insertar por primera vez.

[1] Página oficial: https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest

[2] *Asynchronous JavaScript and XML* (JavaScript y XML asíncrono), técnica empleada para el desarrollo de aplicaciones interactivas.

- delete: Método para realizar borrado de datos, en este caso éste método no acepta datos en el cuerpo de la petición, si se tiene que pasar un dato tiene que ser a través de la cabecera de la petición.

Store es un fichero que implementa *Vuex* [28]. Vuex es una **biblioteca** que ayuda a concentrar los estados de las aplicaciones en un único lugar. Permitiendo mantener una complejidad simple en el sistema debido a la disminución de las interacciones entre los distintos componentes y a la unificación de los estados en un único lugar (**patrón de gestión de estado**). Sirve como un almacén centralizado para todos los componentes de la aplicación, con reglas que aseguran que el estado solo se puede mutar de manera predecible.

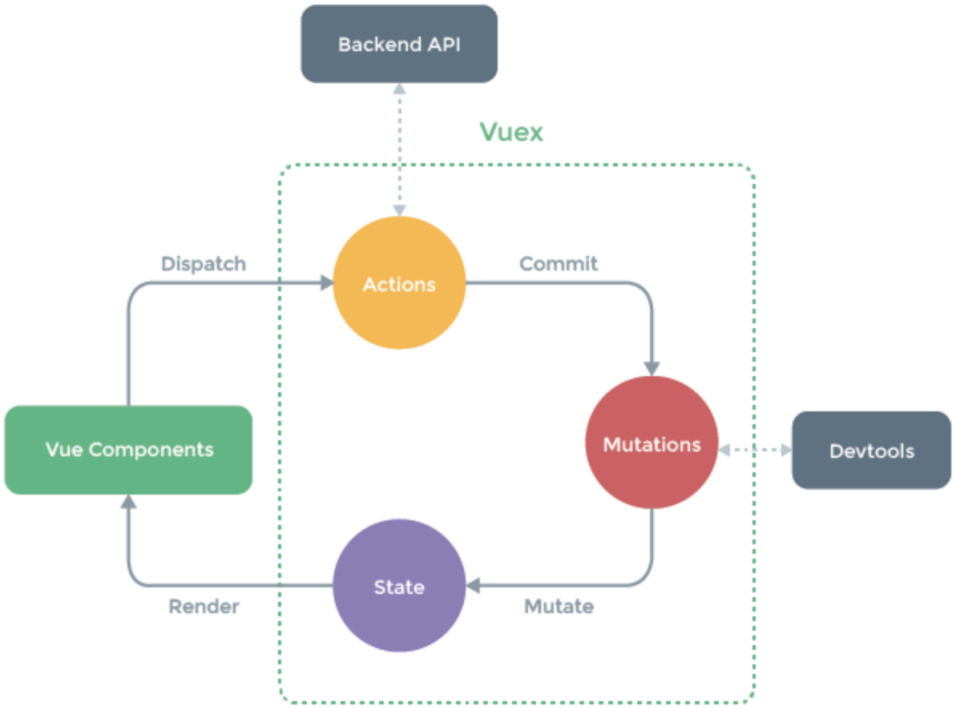

Figura 5.4: Representación de Vuex, extraida de la página oficial de Vuex.

En la Figura 5.4 se muestra el funcionamiento de Vuex. Cuando un componente quiere realizar un cambio realiza un "commit" indicando que desea cambiar. Ello produce un cambio ("mutation") y cambia el "state". De esta manera cualquier componente que fuera a consultar la variable donde se ha realizado el cambio, lo percibirá con la modificación hecha.

La estructura de un fichero que implenta Vuex se divide en:

State, estado del almacén. Es la sección donde se declaran los datos. Si una variable no se encuentra declarada en el *state* no puede ser obtenida ni modificada. La Figura 5.5 es un extracto del código del *state* empleado en la aplicación. Como se puede visualizar contiene variables para

almacenar la información del fichero leído. La variable "file" contiene la información referente al fichero, como son la extensión del mismo, el tamaño de este, etc. Por ejemplo la variable "processedElements" contiene los elementos ya procesados.

```
state: {
  filename: '',
  file: '', // Fichero cargado
  separator: '', // Separador empleado para dividir el fichero
  actions: [], // Acciones que se realizan sobre cada columna
  actionIndex: 0, // Índice de la columna sobre las que se realiza la columna
  dialogActive: false, // Dialogo de las acciones
  stepActive: 0, // Contador de los pasos de la aplicación
  headVector: [], // Vector con los elemtos de la cabecera
  indexHeader: -1, // Índice de la cabera
  processedElements: [], // Datos ya procesados
  modelData: [], // Estructura modal de los datos y las acciones aplicadas
  validados: [], // Valida que las estructuras modales son correctas
  localData: [], // Datos que se visualizan en configData
  fileProcessed: false, // Control de si los datos ya han sido leídos previamente
  unwantedIndexes: []
},
```

Figura 5.5: Sección state de la aplicación

Mutations, es la parte donde se encuentran los métodos encargados de realizar cambios en las variables definidas en el state. En la Figura 5.6 se visualiza un fragmento del código referente a la sección de mutations con tres métodos que se han empleado en este proyecto para realizar modificaciones en variables del state. El primero método (*setFile (state, f)*) inserta el fichero en la variable *file*, este método recibe dos parámetros, uno es el valor a insertar (f) y el otro es el estado donde se encuentra esa variable (state).

```
mutations: {
  // Setter de load
  setFile (state, f) {
    if (state.fileProcessed) {
      store.commit('resetVariables')
    }
    state.file = f
  },
  setFileName (state, name) {
    state.filename = name
  },
  setSeparator (state, data) {
    state.separator = data
  },
```

Figura 5.6: Sección mutations de la aplicación

Getters, esta sección contiene los métodos para obtener los datos del state. La Figura 5.7 es una captura de parte del código de la sección de los getters. En ella se pueden ver algunos métodos, el primero de ellos es *getFile* que devuelve el valor del fichero que se ha leído, el segundo es *getFileName* que devuelve el nombre del fichero leído y el tercero *getFileProcessed* que devuelve si los datos del fichero ya han sido o no procesados. Un detalle importante es que

todos los métodos reciben el parámetro state, ese atributo es el estado en el que está definida el atributo que se ha solicitado con ese método.

```
25    getters: {
26      getFile (state) {
27        return state.file
28      },
29      getFileName (state) {
30        return state.filename
31      },
32      getFileProcessed (state) {
33        return state.fileProcessed
34      },
35      getSeparator (state) {
36        return state.separator
37      },
```

Figura 5.7: Sección getters de la aplicación

Se ha decido emplear Vuex porque ayuda a desacoplar los componentes entre sí. Todos los componentes tienen acceso a las variables que necesitan, si un componente cambia su lógica los demás no se enteran. Esto también favorece la disminución del paso de parámetros entre distintos componentes. La cual cosa ayuda a tener un menor consumo de memoria y una mejor eficiencia.

En cuanto al tema de los componentes principales de la aplicación, son ficheros con extensión vue. En ellos se encuentra el HTML para visualizar el contenido, el JavaScript que le da funcionalidad a la página y el CSS que le proporciona estilo a la misma.

Un claro ejemplo de ello es el componente que carga y lee los datos del fichero, el cual se puede consultar en el Listado 1. Inicialmente desde la línea 1 hasta línea 28 se encuentra el fragmento de código HTML, que comienza con la etiqueta <*template*> y finaliza con la etiqueta <*template*> . Este fragmento se encarga de la visualización de la interfaz.

Por consiguiente desde la línea 29 hasta la línea 117 se encuentra el fragmento de código JavaScript, comienza con la etiqueta <*script*> y finaliza con la etiqueta <*script*>. En este fragmento se encuentran todas las funciones que tratan los datos para facilitar su uso, se podría decir que es la lógica de todo el componente. La función *save(formData)* se encarga de guardar tanto el fichero como el nombre de este. Es una función asíncrona y por ello tiene dos fragmentos. Si todo va bien, se ejecuta la parte del *then* (lineas 84-88), que devuelve una función que se encarga de guardar el fichero en el "store" (línea 85) y de incrementar el estado de la aplicación (línea 86). En cambio, si algo sale mal, se ejecuta la parte del *catch* (líneas 88-92), que devuelve otra función asíncrona que se encarga de imprimir el error por consola, cambiar el estado de la aplicación a fallo y guarda el error para mostrarlo en la interfaz de usuario.

Finalmente desde la línea 118 hasta la línea 161 se encuentra el fragmento de código CSS. En este fragmento se encuentran todas las clases de estilos aplicadas al HTML para darle un diseño amigable, bonito y atractivo.

Cuando se quiere aplicar un estilo en el HTML se usa la palabra reservada *class*, como por

ejemplo <*div class="dropbox"*> (linea 5). Esta clase se encuentra entre las líneas 131 y 140, le da formato al recuadro dónde hay que hacer click para subir el fichero. *outline: 2px dashed grey;* (línea 132) indica que alrededor de este recuadro se va a pintar unas líneas discontinuas de color gris y con un grosor de 2 píxeles. *outline-offset: -10px* (línea 133) indica que la línea que se ha pintado de color gris va ha tener un desplazamiento de 10 píxeles hacia dentro del recuadro. *background: lightcyan*(línea 134) indica que el fondo es de color azul cían. *padding: 10px 10px* (línea 136) indica que el recuadro está 10 píxeles por encima del centro y 10 por debajo, es decir, que de alto tiene 20 píxeles. *min-height: 200px* (lína 137) indica que como mínimo tiene 200 píxeles de ancho. *position: relative* (línea 138) indica que la posición es relativa, es decir que el recuadro se adecua al tamaño del navegador.

```
 8 <template>
 9   <div>
10     <template>
11       <form enctype="multipart/form-data" novalidate v-if="isInitial || isSaving
12         <div class="dropbox">
13           <input type="file" multiple :name="uploadFieldName"
14             :disabled="isSaving"
15             @change="filesChange($event.target.name, $event.target.files)
16             fileCount = $event.target.files.length"
17             accept=".csv" class="input-file">
18           <p v-if="isInitial">
19             Arrastra el fichero CSV dentro del contenedor<br> o haz click para b
20           </p>
21           <p v-else-if="isSaving">
22             Cargando archivo ...
23           </p>
24         </div>
25       </form>
26     </template>
27     <template v-if="isFailed">
28       <el-alert
29         title="¡Error en el fichero leído, intentelo de nuevo!"
30         type="error"
31         show-icon>
32       </el-alert>
33     </template>
34   </div>
35 </template>
36 <script>
37 import axios from '@/axios/axiosConfig.js'
38 import { mapGetters } from 'vuex'
39 const STATUS_INITIAL = 0
40 const STATUS_SAVING = 1
41 const STATUS_SUCCESS = 2
42 const STATUS_FAILED = 3
43 export default {
44   name: 'Load',
45   data () {
```

```
46      return {
47        currentStatus: null,
48        uploadedFiles: [],
49        uploadFieldName: 'csv'
50      }
51    },
52    computed: {
53      ...mapGetters({
54        fileName: 'getFileName'
55      }),
56      page_location: function () {
57        return window.location.origin
58      },
59      isInitial () {
60        return this.currentStatus === STATUS_INITIAL
61      },
62      isSuccess () {
63        return this.currentStatus === STATUS_SUCCESS
64      },
65      isFailed () {
66        return this.currentStatus === STATUS_FAILED
67      },
68      isSaving () {
69        return this.currentStatus === STATUS_SAVING
70      }
71    },
72    methods: {
73      // Metodo llamado por el boton, este méodo recoge el fichero
74      filesChange (fieldName, fileList) {
75        const formData = new FormData()
76        if (!fileList.length) return
77        Array
78          .from(Array(fileList.length).keys())
79          .map(x => {
80            formData.append(fieldName, fileList[x], fileList[x].name)
81          })
82        axios.post('/fileUpload', formData, {headers: {'Content-Type': 'multipart/
83          .then(function (response) {
84            console.log(response)
85          })
86        this.save(formData)
87      },
88      save (formData) {
89        var self = this
90        this.upload(formData)
91          .then(x => {
92            this.$store.commit('setFile', x)
93            this.$store.commit('incrementStepActive')
94            self.currentStatus = STATUS_SUCCESS
```

```
95          }).catch(err => {
96            console.log(err)
97            self.uploadError = err.response
98            self.currentStatus = STATUS_FAILED
99          })
100       },
101       // Obtiene los datos, carga el fichero
102       upload (formData) {
103         return new Promise((resolve, reject) => {
104           const csv = formData.getAll('csv')[0]
105           const fReader = new FileReader()
106           this.$store.commit('setFileName', csv.name)
107           fReader.onload = () => {
108             resolve(fReader.result)
109           }
110           fReader.readAsText(csv)
111         })
112       },
113       reset () {
114         // reset form to initial state
115         this.currentStatus = STATUS_INITIAL
116         this.uploadedFiles = []
117         this.uploadError = null
118       }
119     },
120     mounted () {
121       this.reset()
122     }
123   }
124   </script>
125   <style>
126     .crm {
127       color: rgb(92, 104, 119) !important;
128       background: white;
129       border: none;
130       font-family: OpenSans-Bold !important;
131       font-size: 14px;
132       border-radius: 0px;
133     }
134     .crm:hover {
135       background-color: #b1f0ff !important;
136       opacity: 1;
137     }
138     .dropbox {
139       outline: 2px dashed grey; /* the dash box */
140       outline-offset: -10px;
141       background: lightcyan;
142       color: dimgray;
143       padding: 10px 10px;
```

```
144       min-height: 200px; /* minimum height */
145       position: relative;
146       /* cursor: pointer; */
147     }
148     .input-file {
149       opacity: 0; /* invisible but it's there! */
150       width: 100%;
151       height: 200px;
152       position: absolute;
153       cursor: pointer;
154       right: 0;
155     }
156     .dropbox:hover {
157       background: lightblue; /* when mouse over to the drop zone, change color */
158     }
159     .dropbox p {
160       font-size: 1.2em;
161       text-align: center;
162       padding: 50px 0;
163     }
164     .invalid{
165       color: red;
166       border: 2px solid red;
167      }
168 </style>
```

Listing 5.1: Código del componente que realiza la subida del fichero y carga de datos (*Load.vue*)

5.1.2. Servidor

El servidor se ha desarrollado empleando Nodejs, que ofrece una estructura para *back-end*, es decir, servidores.

En la Figura 5.8 se puede observar cómo está estructurada esta parte. Dentro del directorio principal se encuentran:

- *config* : en este directorio se encuentra el fichero donde se declaran las variables de entorno del servidor.
- *models*: alberga todos los modelos de datos que MongoDB guarda.
- *routes*: Son los *end-point* empleados para que el cliente pueda comunicarse con el servidor.
- *uploads*: almacena los ficheros que se suben al servidor para ser tratados.
- *actionsProcessor.js*: contiene las funciones para aplicar las acciones que procesan los datos.
- *dataFormater.js*: incluye las funciones para seleccionar el tipo de cada dato.

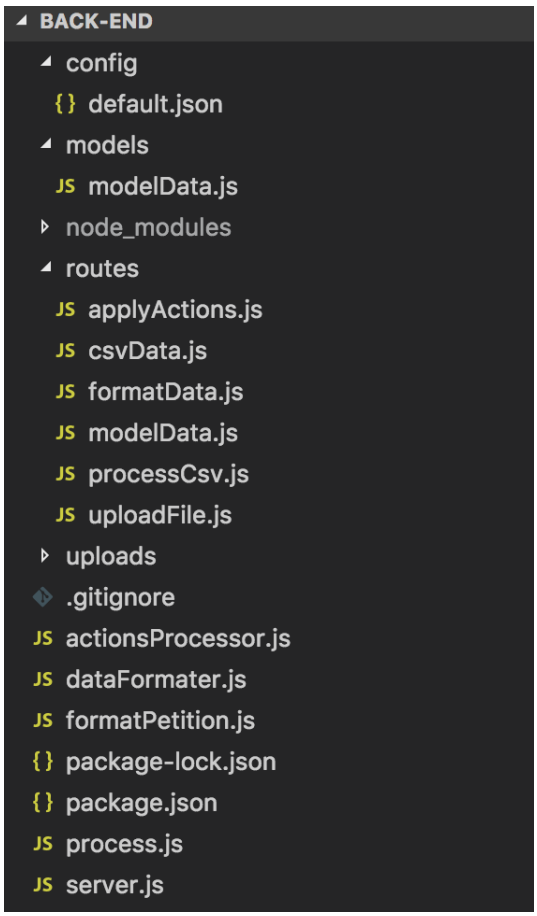

Figura 5.8: Estructura del back-end (servidor)

- *formatPetition.js*: formatea algunas peticiones para que los *end-points* puedan acceder a ellas y así no repetir código en cada *end-poin*
- *process.js*: automatiza todo el proceso de aplicar todas las acciones y formatos definidos en el modelado de datos.
- *server.js*: es el fichero principal de la parte del servidor, es el que ejecuta todo el resto, se podría decir que es el main.

A continuación se detalla más el contenido de los directorios que se acaban de citar.

En el directorio *config* se encuentra la variable de entorno de la base de datos que se emplea para albergar los datos (*"POSTGRESQL_URL": "postgres://tec:123@10.100.100.130:5432/superset"*), de este modo cuando se quiera hacer referencia a la base de datos para almacenar o obtener datos de ella, basta con poner *POSTGRESQL_URL*.

En la carpeta modules, tal y como se ha mencionado antes se encuentran los modelos de datos que MongoDB alberga. Esto es necesario para que MongoDB sepa interpretar la estructura que deben de tener esos datos. En la Figura 5.9 se puede percibir la estructura de los datos que guarda MongoDB en el caso de esta aplicación. Este modelo contiene la fecha para saber

en que momento se creó ese componente, el nombre del componente, el nombre del fichero al que referencia el componente, el separador empleado para tratar el texto, los índices que se desean eliminar del fichero y los componentes en si que se han generado durante el proceso de modelado.

```
var mongoose = require('mongoose'),
    Schema   = mongoose.Schema;
var modelDataSchema = new Schema({
    date: {type:Date},
    name:    { type: String },
    fileName: {type: String},
    separator: {type: String},
    unwantedIndexes: {type: Array},
    components: { type: Array }});
module.exports = mongoose.model('ModelData', modelDataSchema)
```

Figura 5.9: Estructura de MongoDB de la aplicación

routes contiene ficheros con las funciones que escuchan las llamadas que realiza el cliente. Estas funciones son conocidas como *end-poin* o también *API*. En el Listado 5.2 se puede apreciar el contenido de *modelData.js*, que se encarga de escuchar las peticiones a cerca del modelo de datos que guarda MongoDB. En las primeras líneas se declaran aquellas librerías necesarias para que el código pueda funcionar. En la línea 1, se importa el modelo que usa MongoDB para interpretar la estructura de los datos.

Donde más claramente se ve el uso de este modelo es en la función post (línea 6-23), que se encarga de insertar los datos en la base de datos. '/post' es la ruta empleada por el cliente para realizar peticiones a esta función. Se trata de una función asincrona que recibe una función con dos parámetros, req (contiene los valores necesarios para poder hacer la consulta) y res (es donde la función va ha enviar la respuesta).

Entre las líneas 8 y 15 es donde se genera un objeto de tipo ModelData, en este caso a cada atributo del modelo se le asigna su correspondiere variable que se encuentra en req. Para guardar este objeto se emplea la función *save*(línea 16), que también es otra función asíncrona que recibe una función con dos parámetros err y data. En caso de que haya un error, en la variable req se devuelve el error, en caso contrario en req se insertan los datos para notificar que todo ha ido bien.

```
1  var express=require("express")
2  var router = express.Router()
3  var ModelData =require("../models/modelData")
4  /*********************************************/
5  /***********Consultas de MongoDB**************/
6  router.post('/post',function(req,res){
7      var date = new Date().toJSON().slice(0,10)
8      var model=new ModelData({
9          date: date,
10         name: req.body.name,
11         components: req.body.components,
12         fileName: req.body.fileName,
```

```
13          separator: req.body.separator,
14          unwantedIndexes: req.body.unwantedIndexes,
15      })
16      model.save(function(err,data){
17          if(err){
18              res.json(err)
19          }else{
20              res.json(data)
21          }
22      })
23  })
24  router.get('/get',function(req,res){
25      ModelData.find({},function(err,data){
26          if(err){
27              res.json(err)
28          }else{
29              res.json(data)
30          }
31      })
32  })
33  router.delete('/remove',function(req,res){
34      ModelData.remove({},(err) => {
35          if (err) {
36              return res.status(500).send(err)
37          }else{
38              return res.json({msg: "Dato borrada"})
39          }
40      })
41  })
42  router.delete('/remove/:id', function(req,res){
43      ModelData.remove({_id: req.params.id},(err, data) => {
44          if (err) {
45              return res.status(500).send(err)
46          }else{
47              return res.json({msg: "Dato borrado", data: data})
48          }
49      })
50  })
51
52  module.exports = router
```

Listing 5.2: Código de la api de escucha referente a las funciones de mongoDB

actionsProcessor.js, es un fichero que contiene una serie de funciones que son llamadas por los *end-point*, para que procesen una serie de datos que estas les pasan. En el Listado 5.3 se puede distinguir la disposición que tiene y las funciones que contiene. La función principal es *applyActions* (línea 1-21) que recibe los datos a procesar y la estructura que estos tienen. La estructura entre otras cosas contiene las acciones que son necesarias aplicar para cada uno de los datos. Es la única función a la que los *end-point* pueden llamar. Debido a que es la única

que se exporta, tal y como se puede apreciar desde la línea 86 hasta la línea 88.

El trabajo de esta función es muy simple, para cada uno de los datos, consulta las acciones que este tiene. Para cada una de estas, mira si han sido aplicadas o no, en caso de no haber sido aplicadas, se calcula el indice donde será incorporado el dato resultante de aplicar la acción. Por consiguiente se llama la función *processAction*, que se encarga de ver que tipo de acción se va a realizar y llamarla.

Finalmente, la función que procesa el dato y le aplica la acción (desde la línea 49 hasta la línea 67), devuelve el dato y *applyActions* lo inserta en el vector de datos.

```
1  function appyActions (processedElements, modelData) {
2    var dato
3    for (var index = 0; index < processedElements.length; index++) {
4      for (var indice = 0; indice < modelData.length; indice++) {
5        for (var i = 0; i < modelData[indice].actions.length; i++) {
6          if (modelData[indice].actions[i].apply === false) {
7            var id = modelData[indice].id.split('.')
8            id.pop()
9            var id_final = getIndexById(id, modelData)
10           dato = processAction({action: modelData[indice].actions[i].action,
11           start: modelData[indice].actions[i].start,
12           end: modelData[indice].actions[i].end,
13           data: modelData[indice].actions[i].data},
14            processedElements[index][id_final])
15           processedElements[index].splice(indice, 0, dato)
16         }
17       }
18     }
19   }
20   return processedElements
21 }
22 function getIndexById (id, modelData) {
23   for (var j = 0; j< modelData.length; j++) {
24     id_phather = modelData[j].id.split('.')
25     if (id_phather.length === id.length) {
26       var count = 0
27       for (var k = 0; k < id.length; k++) {
28         if (id_phather[k] === id[k]) {
29           count++
30         }
31       }
32       if (count === id.length) {
33         return j
34       }
35     }
36   }
37
```

```
38 }
39 function processAction (options, data) {
40     switch (options.action) {
41       case 'remove':
42         return remove(data, options.start, options.end)
43       case 'replace':
44         return replace(data, options.data, options.start, options.end)
45       case 'extract':
46         return extract(data, options.start, options.end)
47     }
48   }
49 // Realiza la funcion de extraer
50 function extract(data, start, end) {
51   var position = getDataToApplyActions(start, end, data)
52   return data.slice(position[0], position[1])
53 }
54 // Realiza la funcion de reemplazar
55 function replace (data, replaceData, start, end) {
56   var position = getDataToApplyActions(start, end, data)
57   var firstHalf = data.slice(0, position[0])
58   var secodHalf = data.slice(position[1], data.length)
59   return firstHalf + replaceData + secodHalf
60 }
61
62 function remove (data, start, end) {
63   var position = getDataToApplyActions(start, end, data)
64   var firstHalf = data.slice(0, position[0])
65   var secodHalf = data.slice(position[1], data.length)
66   return firstHalf + secodHalf
67   }
68 function getDataToApplyActions (start, end, data) {
69   var positionVector = new Array(2)
70   if (start.type === 0) {
71     positionVector[0] = parseInt(start.position) - 1
72     if (end.type === 0) {
73       positionVector[1] = parseInt(end.position)
74     } else {
75       let dato = data.slice(parseInt(start.position) - 1, data.length)
76       positionVector[1] = dato.indexOf(end.data) + (parseInt(start.position))
77     }
78   } else {
79     var index = data.indexOf(start.data)
80     positionVector[0] = index
81     if (end.type === 0) {
82       positionVector[1] = parseInt(end.position) + index + 1
83     } else {
84       let d = data.slice(index, data.length)
85       positionVector[1] = d.indexOf(end.data) + index + 1
86     }
```

```
87    }
88    return positionVector
89  }
90  module.exports = {
91    appyActions: appyActions
92  }
```

Listing 5.3: Código contenido en el fichero actionsProcessor.js

process.js se encarga de automatizar todo el proceso de modelado de los datos. Este proyecto sin esta parte no tendría mucho sentido porque le exige al usuario tener que estar modelando en todo momento los datos, cuando a lo mejor no quiere realizar ningún cambio en ellos.

En el Listado 5.4 se puede apreciar el código de la función principal que se encarga de realizar las llamadas en orden a otras funciones para que procesen los datos. *processCsv*, recopila los datos leyendo el fichero que se le pasa como parámetro, a ese fichero le aplica un separador (línea 8) obteniendo unos datos que los guarda en un vector. Posteriormente ese vector se divide en objetos empleando la función *divideObjects* (línea 14) para facilitar el trabajo con estos. A continuación se eliminan las líneas que no se desean tener entre los datos (*removeUnwantedRows* línea 16). Después se aplican las acciones (*actions.applyActions* línea 18) y se le da el formato a los datos (*formater.applyDataFormat* línea 20). Por último se insertan los datos y se devuelve la respuesta de la inserción de esos datos, para saber si el proceso ha ido bien o no.

```
1  function processCsv(modelData, file) {
2    readFile(file.filename, function (err, data){
3      if (err) {
4        console.log(err)
5      } else {
6      var separator = modelData.separator
7      var unwantedElements = modelData.unwantedIndexes
8      var lines = splitInLines(data)
9      // Dividimos en un array según el separador
10     var porcessLines = []
11     for (let line in lines) {
12       porcessLines[line] = splitInFields(lines[line], separator)
13     }
14     var rawElements = divideObjects(porcessLines)
15     // Se eliminan las líneas no deseadas
16     rawElements = removeUnwantedRows(rawElements, unwantedElements)
17     // Se llama a la funcion que va ha aplicar las acciones
18     rawElements = actions.appyActions(rawElements, modelData.components)
19     // Dar formato a los valores
20     rawElements = formater.applyDataFormat(modelData.components, rawElements)
21     var resp = formater.insert(rawElements, modelData.name, modelData.components
22     return resp
23     }
24   })
```

```
25  }
```

Listing 5.4: Código contenido en el fichero actionsProcessor.js

server.js: Tal y como se ha mencionado anteriormente, se trata del main de la aplicación. En él se encuentran todas las bibliotecas necesarias para la ejecución del servidor. Una de las bibliotecas más importantes del servidor es *Express*. Esta biblioteca permite recibir y contestar las peticiones que realiza el cliente al servidor.

5.1.3. Superset

Para poder visualizar el contenido en las gráficas de Superset, se han realizado dos tareas.

La primera de ellas consiste en varias subtareas. Inicialmente se ha configurado la conexión con la base de datos para que pueda leer los datos. Posteriormente se ha configurado el dashboard, para tener un área donde representar esos datos. Finalmente se ha verificado que la conexión con la base de datos es correcta, durante esta tarea apareció el problema de que Superset no lee el formato de la base de datos, *utf-8*, es decir, no permite realizar consultas a la base de datos. Para resolver este error, se añadieron las líneas que se pueden ver en el Listado 5.5, al fichero config.py, que se encuentra dentro del *path* de Superset. Esto se ha tenido que realizar debido a que Superset está desarrollada en Python 2.7 y éste no sabe como interpretar el formato de *utf-8* de manera automática, por ello hay que indicarselo de manera explicita.

```
1 mport sys  # import sys package, if not already imported
2 reload(sys)
3 sys.setdefaultencoding('utf-8')
```

Listing 5.5: Codigo añadido a config.py

La segunda tarea consta de dos subtareas. Por una parte, se ha elegido el tipo de gráficas que se quieren adherir al dashboard, esto se realiza en un entorno gráfico. Por otra se han realizado las consultas SQL, para obtener los datos que se van a mostrar en las gráficas. A continuación se va ha explicar de manera más detallada algunas de las consultas SQL que se han realizado.

La primera consulta obtiene los meses y las medias del tiempo que las llamadas han estado en espera (sonando), el tiempo de conversación de cada llamada y el tiempo total de una llamada, es decir una suma de las dos anteriores. Así como se puede apreciar en el listado 5.6

```
1  SELECT sub.mes, AVG(sub.sonando) as sonado,
2    AVG(sub.hablando) as hablado,
3    AVG(sub.totales) as total
4    FROM ( SELECT tiempo_de_llamada, numero,extension, estado,sonando,
5          hablando, totales, costo,razón,
6          EXTRACT(MONTH FROM tiempo_de_llamada) as mes
7          FROM demofinal) AS sub
8  WHERE NOT (sub.mes IS NULL)
9  GROUP BY sub.mes
10 ORDER BY sub.mes;
```

Listing 5.6: Primera consulta SQL

La segunda consulta muestra las coordenadas de las provincias de donde se han realizado y recibido llamada. Esta consulta es importante porque permite crear un mapa con estas posiciones marcadas, mostrando un mapa geográfico del alcance de la actividad de la empresa. Tal como se puede ver en el listado 5.7

```
1 SELECT demo.tiempo_de_llamada, demo.numero,
2   demo.destino, demo.estado, demo.sonando, demo.hablando,
3   demo.totales, demo.costo, demo.razón as razon,
4   ex.municipio, ex.coordenaday, ex.coordenadax, ex.municipio
5 FROM demofinal as demo JOIN extensiones as ex
6   ON demo.extension = ex.numero;
```

Listing 5.7: Segunda consulta SQL

La tercera y última consulta consigue todas las llamadas, pero asignándoles un tipo, es decir, si las llamadas son recibidas o realizadas. En esta consulta la distinción se ha realizado empleando el número telefónico de la empresa (línea 6), de tal modo que si una llamada en el origen contiene ese número es que ha sido realizada, en caso contrario significa que ha sido recibida. Según se puede contemplar en el listado 5.8.

```
1 SELECT pr.codigo ,pr.tiempo_de_llamada, pr.numero,
2   pr.extension, pr.destino, pr.estado, pr.sonando,
3   pr.hablando, pr.totales, pr.costo, pr.razón as razon,
4   CASE
5     WHEN not (pr.numero='964066964') THEN 'Recibida'
6     WHEN (pr.numero='964066964') THEN 'Realizada'
7   END AS tipo
8   FROM demofinal as pr join extensiones as ext
9     ON ext.numero = pr.extension;
```

Listing 5.8: Tercera consulta SQL

5.2. Verificación y validación

La verificación y validación de la aplicación se ha ido realizando durante el desarrollo de cada componente y también durante las revisiones de los sprints.

Durante el desarrollo de los componentes tanto del cliente como del servidor se han empleado distintas herramientas que han facilitado la detección de errores.

En la parte del cliente, se ha empleado *devtools* de Vuejs. Tal y como se explica en la sección de tecnologías del capítulo 2, se trata de una herramienta que sirve para ayudar en la detección de errores. En la Figura 5.10 se puede apreciar dicha herramienta en funcionamiento. Esta herramienta permite ver en vivo todos los cambios que sufren las variables de la aplicación y en que componente sufren dicho cambio. Además se integra muy bien con *Vuex*, con lo cual permite ver que funciones getter y mutations se han ejecutado del store en tiempo real. Es decir, en el

momento que una función que esté dentro de mutations realice alguna modificación en alguna variable se muestra, tal y como se aprecia en la Figura 5.11.

Además, ofrece la funcionalidad de detección de los eventos que se han producido, esto es, detectar todos los clicks de ratón, o todos las inputs que se han introducido, todas las veces que se ha pulsado la tecla "enter", etc. Esto mismo se puede visualizar en la Figura 5.12.

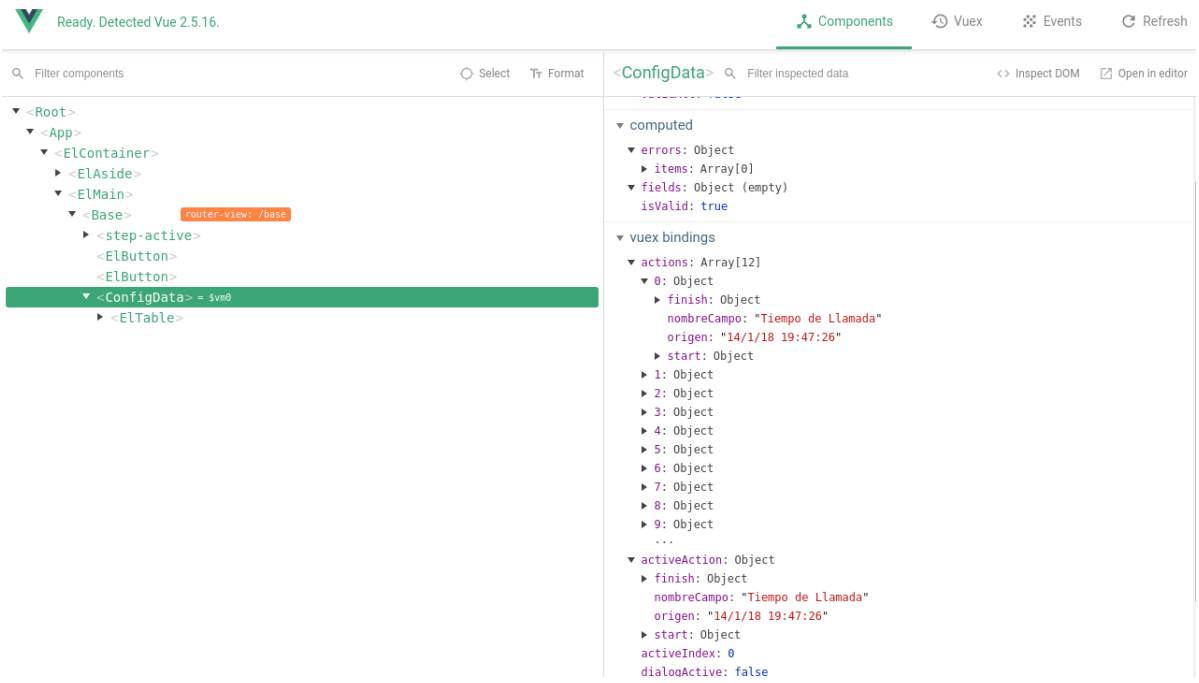

Figura 5.10: DevTools, cambios sufridos en las variables de cada componente

A parte del uso de esta herramienta, también se han realizado pruebas unitarias para verificar que las funciones programadas se comportan como se esperaba.

En cuanto al servidor se ha verificado de dos maneras. Por una parte las funciones que procesan los datos, se han testeado con tests unitarior para comprobar que procesan de manera correcta los datos. Por otra parte los *end-poin* se han testeado empleando Postman, una aplicación que permite enviar peticiones a APIs y ver la respuesta ante esas peticiones. Esto nos permite ver que datos se les pasa a los *end-point* y que datos devuelven. En la Figura 5.13 se observa una petición al end-point (*/modelData/get*) que sirve para obtener todos los modelos que hay en el servidor. En la parte baja de la figura, se puede apreciar la respuesta ofrecida a esta petición.

Por otra parte durante las reuniones de los sprints, los componentes han sido testeados manualmente por el equipo de desarrollo que está llevando este proyecto.

Otra cuestión importante que se ha tenido en cuenta es la validación de los datos que se han insertado en las bases de datos. Para MongoDB, se ha empleado robomongo, una herramienta que permite visualizar tanto la estructura de la base de datos MongoDB (Figura 5.14), como el contenido de la misma (Figura 4.2).

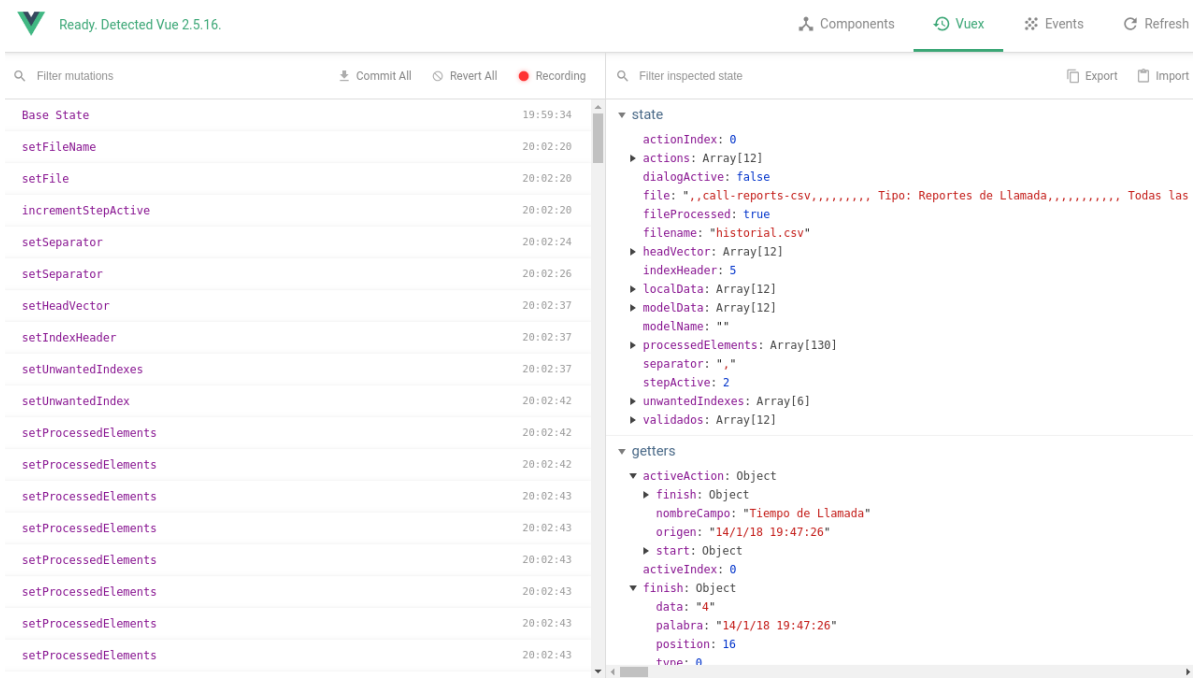

Figura 5.11: DevTools, cambios sufridos en las variables del store

Respecto a la validación de la interfaz, no se han realizad pruebas debido a la gran complejidad de estas y debido a que se validaban en los sprints por el equipo de desarrollo de este proyecto. De esta manera se ha llevado un desarrollo de interfaces progresivo, es decir a que a medida que transcurrían los sprints, las interfaces se probaban una a una y se mejoraban.

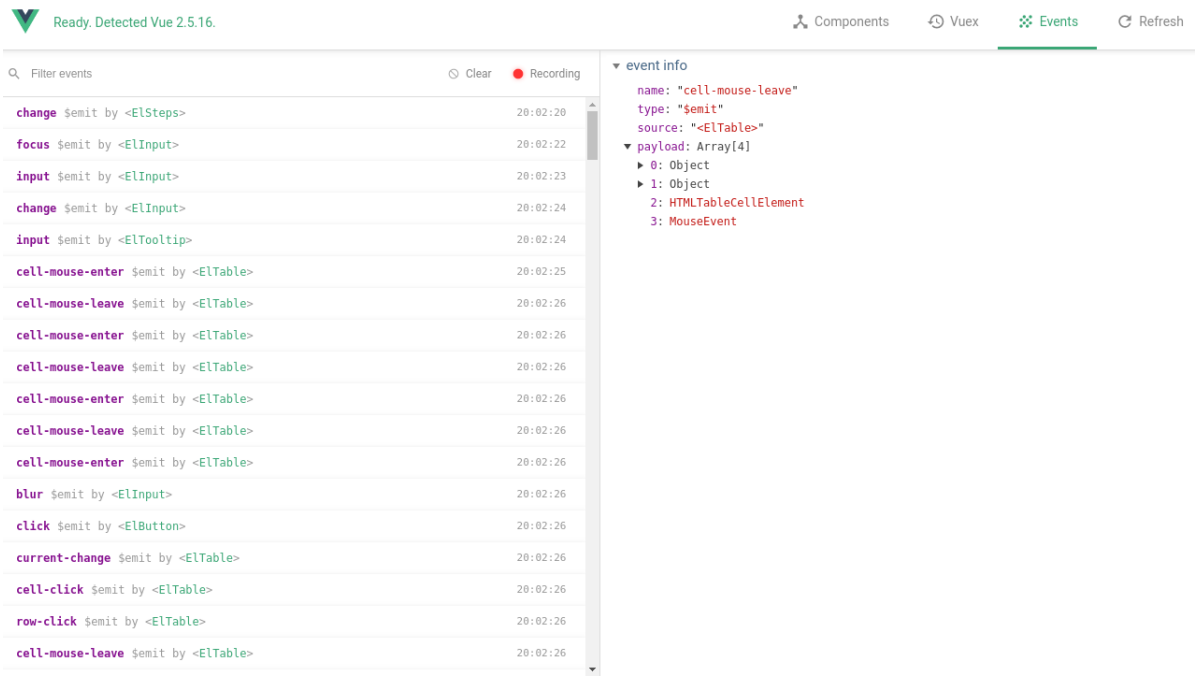

Figura 5.12: DevTools, detección de eventos

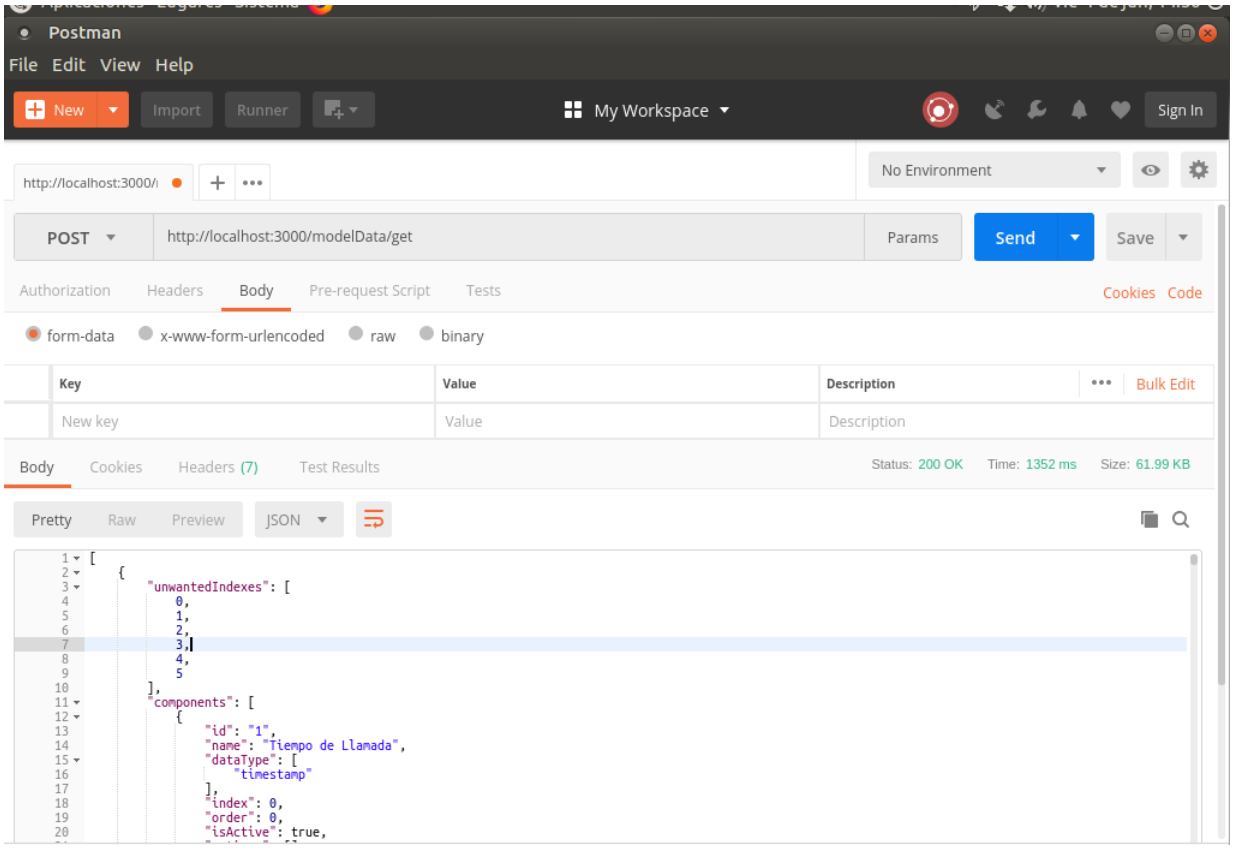

Figura 5.13: Postman, testeo de las APIs del servidor

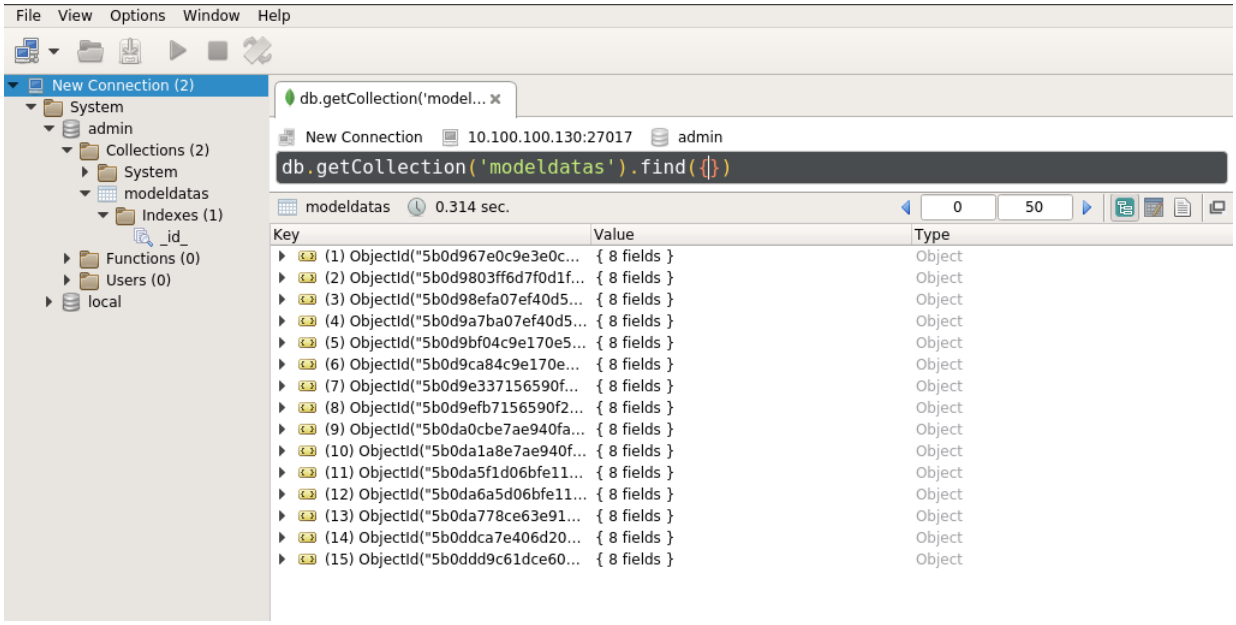

Figura 5.14: Estructura de MongoBD

Capítulo 6

Conclusiones

En este capitulo se van a desarrollar las conclusiones que se han tenido a cerca del proyecto. En un sentido personal, profesional y además se comentarán aspectos respecto a la viabilidad del proyecto.

6.1. Conclusión personal

A nivel personal, este proyecto en un principio me pareció un poco complicado de realizar, debido a que integraba muchas tecnologías de las que solo conocía el nombre. Esto me hizo entrar con un poco de miedo al proyecto, pero a medida que el proyecto avanzaba fui cogiendo confianza y realizando las tareas que se me solicitaban para completarlo.

Realmente a lo largo del proyecto ha habido momentos de estrés, momentos de risa y momentos de querer dejarlo todo. El estrés aparecía debido a la incertidumbre, a no saber como tratar con ciertas tecnologías o como resolver ciertos errores que nunca había visto. La alegría como ya se lo pueden imaginar, viene cuando se solventan todos esos problemas. Las ganas de dejarlo todo, surgió una vez a lo largo de todo el proyecto y fue debido a que estuve una semana intentando resolver un problema con Superset y no lo conseguí. Después una semana y media y fue cuando dí con la solución del problema.

6.2. Conclusión profesional

Este proyecto ha servido para abrirme los ojos respecto a como es el ámbito profesional de la ingeniería informática. No siempre conoces todos las tecnologías para desarrollar un proyecto, ello implica que tienes que estar siempre formándote a cerca de las últimas tecnologías para poder cubrir los nuevos proyectos que surgen.

He ganado una gran experiencia trabajando con el gran equipo que posee esta empresa. Aprendiendo mejor como se trabaja empleando la metodología scrum. Como se enfoca un pro-

yecto desde cero, con todas las fases necesarias.

6.3. Viabilidad del proyecto

Ya al principio, en el capítulo 1, se comentó que este proyecto nace como un proyecto interno pero que se quería extrapolar a los clientes de esta empresa. Pero el resultado ha sido que el proyecto se suspende, debido a la gran complejidad de Superset.

La persona que configure Superset para los clientes, tiene que ser una persona entendida en el ámbito de la informática, ya que para realizar las consultas a las bases de datos, tiene que escribirlas en lenguaje SQL porque las consultas son demasiado complejas como para hacerlas empleando el entorno visual de Superset.

Hay que decir que este problema es un problema de Superset y no del alumno que ha realizado las prácticas, porque Superset está bastante limitado respeto al entorno visual. Todo aquello que se quiera hacer que tenga algo de complejidad tiene que ser empleando el lenguaje SQL de la base de datos con la que esté conectada, en este caso PostgreSQL

Bibliografía

[1] Axios. https://vuejs.org/v2/cookbook/using-axios-to-consume-apis.html. [Consulta: 20 de Febrero de 2018].

[2] Axios. https://expressjs.com/. [Consulta: 20 de Febrero de 2018].

[3] Css. https://developer.mozilla.org/es/docs/Web/CSS. [Consulta: 1 de Mayo de 2018].

[4] Dashboardfree. http://www.dashboardfree.com/. [Consulta: 12 de Febrero de 2018].

[5] Dashbuilder. http://dashbuilder.org/about.html. [Consulta: 12 de Febrero de 2018].

[6] Dashing. http://dashing.io/. [Consulta: 12 de Febrero de 2018].

[7] Element io. http://element.eleme.io/#/es. [Consulta: 15 de Febrero de 2018].

[8] Freeboard. https://freeboard.io/. [Consulta: 12 de Febrero de 2018].

[9] Grafana. https://grafana.com/. [Consulta: 12 de Febrero de 2018].

[10] Html. https://developer.mozilla.org/es/docs/Web/HTML. [Consulta: 1 de Mayo de 2018].

[11] Javascript. https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/JavaScript_basics. [Consulta: 1 de Mayo de 2018].

[12] Metabase. https://www.metabase.com/. [Consulta: 12 de Febrero de 2018].

[13] Microssoft windows. https://www.microsoft.com/es-es/store/b/windows. [Consulta: 15 de Febrero de 2018].

[14] Mongodb. https://www.mongodb.com/what-is-mongodb. [Consulta: 1 de Mayo de 2018].

[15] Mozaik. http://mozaik.rocks/. [Consulta: 12 de Febrero de 2018].

[16] Nodejs. https://nodejs.org/en/. [Consulta: 15 de Febrero de 2018].

[17] Postgresql. https://www.postgresql.org/. [Consulta: 15 de Febrero de 2018].

[18] Postman. https://www.getpostman.com/. [Consulta: 1 de Mayo de 2018].

[19] Qclicksense. https://www.qlik.com/us/products/qlik-sense. [Consulta: 12 de Febrero de 2018].

[20] Razorflow. https://razorflow.com/. [Consulta: 12 de Febrero de 2018].

[21] Risingstars. https://risingstars.js.org/2017/en/#section-ide. [Consulta: 15 de Febrero de 2018].

[22] Stashboard. http://www.stashboard.org/. [Consulta: 12 de Febrero de 2018].

[23] Superset. https://superset.incubator.apache.org/. [Consulta: 12 de Febrero de 2018].

[24] Visual studio code. https://code.visualstudio.com/. [Consulta: 1 de Mayo de 2018].

[25] Visualizejs. https://community.jaspersoft.com/project/visualizejs. [Consulta: 12 de Febrero de 2018].

[26] Vue-devtools. https://github.com/vuejs/vue-devtools. [Consulta: 15 de Febrero de 2018].

[27] Vue.js. https://vuejs.org/. [Consulta: 15 de Febrero de 2018].

[28] Vuex. https://vuex.vuejs.org/. [Consulta: 20 de Febrero de 2018].

[29] Remi Caudwell. World quality report 2017-18 ninth edition. [Consulta: 10 de Mayo de 2018].