UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Herramienta web para la gestión de informes automáticos

Autor:
Daniel Montero Castelao

Directores:
Manuel Lama Penín
Juan Carlos Vidal Aguiar
Victor José Gallego Fontenla

Grado en Ingeniería Informática

Febrero 2019

Trabajo de Fin de Grao presentado en la *Escola Técnica Superior de Enxeñaría* de la *Universidade de Santiago de Compostela* para la obtención del Grado en Ingeniería Informática.

D. Manuel Lama Penín, Profesor del Departamento de Electrónica y Computación de la Universidade de Santiago de Compostela, y **D. Juan Carlos Vidal Aguiar**, Profesor del Departamento de Electrónica y Computación de la Universidade de Santiago de Compostela, y **D. Victor José Gallego Fontenla**, Investigador Predoctoral FPU del Ministerio de Ciencia, Innovación y Universidades.

INFORMAN:

Que la presente memoria, titulada *Herramienta web para la gestión de informes automáticos*, presentada por **D. Daniel Montero Castelao** para superar los créditos correspondientes al Trabajo de Fin de Grao de la titulación de Grao en Enxeñaría Informática, se realizó bajo nuestra dirección en el Departamento de Electrónica y Computación de la Universidade de Santiago de Compostela.

Y para que así conste a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a 30 de enero de 2019:

El director,

Manuel Lama Penín

El codirector,

Juan Carlos Vidal Aguiar

El codirector,

Victor José Gallego Fontenla

El alumno,

Daniel Montero Castelao

Índice general

1. Introducción **1**
1.1. Objetivos 2
1.2. Estructura del documento 3

2. Gestión del proyecto **5**
2.1. Gestión del alcance 5
2.1.1. Descripción del alcance del producto 5
2.1.2. Criterios de aceptación del producto 6
2.1.3. Entregables del producto 7
2.1.4. Restricciones del proyecto 7
2.2. Gestión de riesgos 8
2.2.1. Activos 8
2.2.2. Amenazas 9
2.2.3. Especificación de riesgos 11
2.2.4. Identificación de riesgos 13
2.2.5. Tratamiento de riesgos 15
2.3. Metodología de desarrollo 18
2.4. Planificación 21
2.4.1. Estructura de descomposición del trabajo (EDT) 21
2.4.2. Cronograma (Planificación inicial) 24
2.4.3. Cronograma (Planificación real) 25
2.5. Gestión de la configuración 26
2.5.1. Elementos de configuración 27
2.5.2. Sistema de gestión de la configuración 27
2.5.3. Estructura del repositorio 28
2.5.4. Gestión de cambios 28
2.6. Gestión de costes 29
2.6.1. Costes directos 29
2.6.2. Costes indirectos 30
2.6.3. Costes totales 31
2.7. Gestión de las comunicaciones 31
2.7.1. Planificación de las comunicaciones 34

3. Análisis de requisitos **35**
3.1. Actores del sistema . . . 35
3.1.1. Intereses del usuario . . . 36
3.1.2. Perfil de conocimiento del usuario . . . 36
3.1.3. Factores de rechazo del usuario . . . 37
3.2. Especificación de requisitos . . . 37
3.2.1. Requisitos funcionales . . . 38
3.2.2. Requisitos no funcionales . . . 44
3.2.3. Requisitos de información . . . 45
3.2.4. Casos de uso . . . 47
3.2.5. Matrices de trazabilidad . . . 53

4. Análisis de tecnologías y herramientas **55**
4.1. Servicios web . . . 55
4.1.1. Jersey y Java . . . 56
4.1.2. Express.js y JavaScript . . . 57
4.1.3. Rocket y Rust . . . 57
4.1.4. Conclusión . . . 57
4.2. Tecnologías web estándar . . . 58
4.2.1. HTML5 . . . 58
4.2.2. CSS3 . . . 59
4.2.3. JavaScript . . . 59
4.2.4. JSON . . . 60
4.3. Frameworks y librerías . . . 60
4.3.1. TypeScript . . . 60
4.3.2. React . . . 61
4.3.3. Ant-Design . . . 62
4.3.4. Recharts.js . . . 62
4.3.5. Material icons . . . 63
4.4. Base de datos documental . . . 63
4.4.1. MongoDB y Mongoose . . . 64
4.5. Herramientas . . . 64
4.5.1. NPM . . . 64
4.5.2. Docker . . . 65
4.5.3. PhantomJS . . . 65
4.5.4. Git . . . 66
4.5.5. WebStorm . . . 66
4.5.6. TeXStudio . . . 66

5. Diseño e implementación del sistema **67**
5.1. Diseño de la arquitectura del sistema . . . 67
5.1.1. Esquema de la arquitectura del sistema . . . 67
5.1.2. Arquitectura global . . . 68

5.1.3. Arquitectura del cliente web 70
5.1.4. Arquitectura del servidor 72
5.2. Patrones arquitectónicos 74
5.2.1. Arquitectura orientada a servicios 74
5.2.2. Flux 74
5.3. Modelo de datos 75
5.4. Diseño de clases 77
5.4.1. Diagramas de paquetes 77
5.4.2. Diseño de clases del cliente web 80
5.4.3. Diseño de clases del servidor 93
5.5. Diagramas de interacción 95
5.5.1. Secuencia de llamadas para guardar una plantilla 95
5.5.2. Secuencia de llamadas para cargar una plantilla 96
5.5.3. Secuencia de llamadas para imprimir una plantilla a PDF . 98
5.6. Diseño de interfaz de usuario 100

6. Validación y pruebas **103**
6.1. Pruebas unitarias 103
6.2. Pruebas de integración 108
6.3. Validación de interfaz de usuario 111
6.3.1. Heurísticos de Nielsen 112
6.3.2. Cuestionario SUS 114
6.4. Matriz de trazabilidad 116

7. Conclusiones y ampliaciones **117**
7.1. Conclusiones 117
7.2. Ampliaciones 119

A. Manuales técnicos **121**
A.1. Manual de despliegue 121
A.2. Manual de configuración 122
A.2.1. Cambiar la URI de la capa de servicios 122
A.2.2. Cambiar la URI de la base de datos 123
A.2.3. Versiones utilizadas en el desarrollo 123

B. Manual de usuario **125**
B.1. Vista normal de la aplicación 125
B.2. Crear nuevos componentes 126
B.3. Cambiar las propiedades de la plantilla 127
B.4. Propiedades, tamaño y posición de un componente 128
B.4.1. Propiedades de componentes de texto 129
B.4.2. Propiedades del componente imagen 130
B.4.3. Propiedades del componente lista 131

B.4.4. Propiedades del componente malla 132
B.4.5. Propiedades del componente tabla 133
B.4.6. Propiedades del componente gráfica cartesiana 134
B.4.7. Propiedades del componente gráfica polar 135
B.4.8. Propiedades del componete gráfica de dispersión 137
B.5. Definición de variables 139
B.5.1. Sección de creación de variables 139
B.5.2. Tipos de variables 139
B.5.3. Mapeador de variables 141
B.5.4. Cargador de JSON 143
B.6. Uso de variables y fuentes de datos 144
B.7. Creación, guardado y borrado de plantillas 145
B.8. Impresión de plantilla 145

C. Licencia **147**

Bibliografía **149**

Índice de figuras

2.1. Primer nivel de la estructura de descomposición de trabajo 21
2.2. Descomposición del paquete de trabajo *Definición del proyecto* . . 22
2.3. Descomposición del paquete de trabajo *Aplicación web* 22
2.4. Descomposición del paquete de trabajo *Capa de servicios* 23
2.5. Descomposición del paquete de trabajo *Entorno y base de datos* . 23
2.6. Cronograma general del proyecto 24
2.7. Cronograma del paquete de trabajo *Definición del proyecto* 25
2.8. Cronograma del paquete de trabajo *Aplicación web* 25
2.9. Cronograma del paquete de trabajo *Capa de servicios* 25
2.10. Cronograma del paquete de trabajo *Entorno y base de datos* . . . 25
2.11. Planificación real del proyecto 26
2.12. Estructura del repositorio 28

5.1. Esquema de la arquitectura del sistema 68
5.2. Arquitectura de alto nivel del sistema 69
5.3. Arquitectura de la aplicación web 70
5.4. Arquitectura del servidor 72
5.5. Diagrama de paquetes del cliente web 78
5.6. Diagrama de paquetes del servidor 79
5.7. Diagrama de clases del núcleo de la aplicación 81
5.8. Diagrama de clases del paquete *ComponentsPanel* 83
5.9. Diagrama de clases del paquete *CRUDPanel* 84
5.10. Diagrama de clases del paquete *LayoutComponents* (1) 85
5.11. Diagrama de clases del paquete *LayoutComponents* (2) 86
5.12. Diagrama de clases del paquete *PropsPanels* (1) 88
5.13. Diagrama de clases del paquete *PropsPanels* (2) 89
5.14. Diagrama de clases del paquete *VarsSection* 91
5.15. Diagrama de clases del paquete *ReportLayout* 93
5.16. Diagrama de clases del servidor 94
5.17. Diagrama de interacción para **Guardar una plantilla** 96
5.18. Diagrama de interacción para **Cargar una plantilla** 98
5.19. Diagrama de interacción para **Imprimir a PDF** 99
5.20. Versión minimalista de la interfaz de la aplicación 100
5.21. Boceto de la disposición de los elementos en la interfaz completa . 101

5.22. Versión completa de la interfaz de la aplicación 102

B.1. Pantalla principal de la aplicación 125
B.2. Panel de componentes de la aplicación 126
B.3. Panel de propiedades de la plantilla 127
B.4. Panel de edición de un título 128
B.5. Panel de edición de un párrafo 129
B.6. Panel de edición de una imagen 130
B.7. Panel de edición de una lista 131
B.8. Panel de edición de una celda de una malla 132
B.9. Panel de edición de una tabla estática 133
B.10. Panel de edición de una gráfica cartesiana 134
B.11. Panel de edición de una gráfica de sectores 136
B.12. Panel de edición de una gráfica de radar 137
B.13. Panel de edición de una gráfica de dispersión 138
B.14. Panel de creación de variables 139
B.15. Mapeador de variables 141
B.16. Cargador de JSON 143

Índice de cuadros

2.2. Tabla de evaluación de interesados del proyecto 32
2.3. Tabla de clasificación de interesados del proyecto 32
2.1. Tabla de identificación de interesados del proyecto 33
2.4. Matriz de planificación de comunicaciones 34

3.1. Matriz de trazabilidad Caso de uso - Objetivos 53
3.2. Matriz de trazabilidad Caso de uso-Entregables 54

6.1. Matriz de trazabilidad Pruebas Unitarias - Casos de uso 116

Capítulo 1

Introducción

La toma de decisiones bien informada y a tiempo es vital para cualquier organización, independientemente del nivel ejecutivo y de las plataformas, aplicaciones o infraestructuras que se utilicen.

Los informes son una de las principales fuentes utilizadas para fundamentar estas decisiones. Sin embargo, su generación sigue siendo, a día de hoy, un proceso costoso y, en la mayoría de los casos, completamente o parcialmente manual. Además, la gran cantidad de datos manejados en estos informes hace que, en la actualidad, sea más necesario disponer de herramientas que permitan generar de forma automática informes.

Un ejemplo de esta situación en el ámbito médico es el proceso de rehabilitación cardíaca de pacientes que han sufrido un evento coronario. En este proceso, los médicos generan manualmente informes analizando la evolución del paciente tomando en cuenta cerca de 400 variables clínicas.

La situación anterior conlleva una serie de problemas que las tecnologías de la información y la comunicación deben de abarcar.

El primer problema es que la información que maneja el experto del negocio (en este caso al médico) cambia con el tiempo, pues aparecen nuevas variables, mientras que otras quedan obsoletas y deben de eliminarse, lo que conlleva a una volatilidad de datos.

El segundo problema es que las tecnologías de la información avanzan y abren nuevos frentes de posibilidades para representar o tratar la información, esto provoca que la generación de informes también cambie, pues es necesario adaptarlos a las nuevas situaciones que se avecinan y a la nueva información que se debe de mostrar.

Finalmente, los expertos del negocio pueden tener ciertas preferencias sobre como representar la información en un informe, estas preferencias pueden variar de un experto a otro o de una corporación a otra, lo que provoca que la misma información se pueda representar de muchas maneras distintas.

Es por ello que muchas aplicaciones o sistemas que se dedican a la generación de informes quedan obsoletos o presentan dificultades de ampliación o escalabilidad al estar construidos de forma monolítica o situacional, de modo que solo generan un tipo de informe o un conjunto pequeño de ellos cuya estructura e información no cambia. Esto último, provoca que no se puedan adaptar con la flexibilidad suficiente a los problemas anteriormente descritos lo que suele conllevar implementaciones adicionales del sistema.

En este trabajo fin de grado (TFG) se propondrá una solución genérica a lo anterior, de forma que dado un modelo de datos se puedan construir infinitas representaciones de información y que estas sean fácilmente alterables, tanto para añadir más datos, quitarlos o para cambiar la forma de mostrar la información.

Todo lo anterior se realizará mediante la creación de un sistema basado en una arquitectura orientada a servicios junto con una herramienta web que utilice dichos servicios.

La herramienta web proporcionará una interfaz que permita la creación plantillas de informes junto con la definición de la estructura de la información que contendrán, esto último incluye la definición de variables y fuentes de datos para vincular a un modelo de datos heterogéneo y no especificado de antemano que será insertado en la plantilla.

Esta herramienta utilizará los servicios web expuestos por el sistema para crear, guardar, cargar y modificar dichas plantillas, esto incluye tanto la estructura como la información (variables y fuentes de datos), adicionalmente el sistema proporcionará servicios para permitir la impresión del informe generado a formato PDF en caso de que el usuario lo necesite.

1.1. Objetivos

El objetivo principal del proyecto es la creación de un sistema que de soporte a la generación de informes automáticos basados en plantillas, en concreto se plantean los siguientes objetivos:

- **OBJ-1**: Desarrollo de una interfaz gráfica que permita a los usuarios crear plantillas de informes según sus preferencias o necesidades de la forma más

genérica posible. La interfaz web será una **SPA** [1] ejecutada sobre el navegador web, los datos que se representen en la plantilla serán cargados desde el almacenamiento local del cliente y substituidos por la propia aplicación.

- **OBJ-2**: Desarrollo de los componentes de una arquitectura orientada a servicios que permitan la gestión y mantenimiento de los informes creados por los usuarios de la aplicación web. Estos componentes serán utilizados por la herramienta web para guardar la estructura de las plantillas que genere.
- **OBJ-3**: Desarrollo de los componentes de una arquitectura orientada a servicios que faciliten la integración de fuentes de conocimiento externas. Estos servicios permitirán el guardado de modelo de datos en plantillas guardadas de antemano para su substitución en la aplicación web.

1.2. Estructura del documento

La presente memoria recoge toda la documentación relacionada con el desarrollo de este Trabajo fin de Grado (de ahora en adelante TFG). La estructura en la que está dividida la memoria es la siguiente:

- **Capítulo 1**: **Introducción**, descripción del contexto y motivación del trabajo a realizar, objetivos del proyecto y estructura del documento.
- **Capítulo 2**: **Gestión del proyecto**, incluye la planificación, alcance del proyecto, gestión de riesgos, gestión de la configuración, gestión de costes y gestión de las comunicaciones.
- **Capítulo 3**: **Análisis de requisitos**, incluye el análisis de los actores del sistema, análisis de los distintos factores de aceptación o rechazo e intereses de los actores del sistema y el análisis de los distintos requisitos del sistema junto los casos de uso.
- **Capítulo 4**: **Análisis de tecnologías y herramientas**, incluye el análisis de las distintas tecnologías y herramientas que se emplean para el desarrollo del proyecto junto con la justificación de su uso.
- **Capítulo 5**: **Diseño e implementación del sistema**, incluye los diagramas de arquitectura del sistema, diagramas de clases de los distintos módulos del sistema, diagramas de interacción de las funcionalidades más complejas del sistema y la sección de diseño de la interfaz gráfica de la aplicación.

[1] *Single Page Application*, aplicación web que interactúa directamente con el usuario mediante la reescritura de la página actual, evitando la sensación de interrupción entre carga de páginas.

- **Capítulo 6**: **Validación y pruebas**, incluye las pruebas unitarias, pruebas de integración, y validación de interfaz de usuario.
- **Capítulo 7**: **Conclusiones**, incluye las conclusiones obtenidas tras la realización del trabajo realizado y las posibles ampliaciones del mismo.
- **Apéndice A**: **Manual de despliegue**, incluye el manual de despliegue de la aplicación junto las opciones de configuración disponibles.
- **Apéndice B**: **Manual de usuario**, incluye el manual de usuario para la utilización de la aplicación.
- **Apéndice C**: **Licencia de uso**, incluye la licencia de uso del software y de este trabajo.

Capítulo 2

Gestión del proyecto

La parte de gestión del proyecto es una fase gradual que se realiza durante todo el transcurso de un proyecto cualquiera. En esta fase se define el cronograma de las actividades, la gestión de los riesgos, los costes y las comunicaciones con los diferentes interesados en el proyecto.

Puesto que este TFG es un proyecto informático, también será necesario especificar la gestión de los distintos elementos del software que se desarrollen durante el transcurso del proyecto, esto incluye, el modo de guardar la diferentes versiones que se generan así como el modo de documentar los cambios que se realizan en el código desarrollado.

2.1. Gestión del alcance

En esta sección se describe el alcance del producto, los criterios de aceptación del producto por parte del cliente, los entregables del producto junto con sus restricciones.

2.1.1. Descripción del alcance del producto

El producto en su conjunto consta de tres partes:

- **Aplicación web (Cliente)**
- **Capa de servicios y base de datos (Servidor)**
- **Documentación de la aplicación**

La aplicación web es una interfaz de usuario ejecutada sobre un navegador web moderno cualquiera. Esta interfaz permite la creación de plantillas de informes, el usuario podrá seleccionar un rango de componentes para formar la plantilla, dichos componentes van desde párrafos y títulos, hasta imágenes, listas, tablas y

gráficas.

Adicionalmente el usuario podrá modificar las propiedades de cada componente de la plantilla, incluyendo (pero no limitándose) el tamaño del componente, fuente, desplazamiento, decoración, etc.

La aplicación web permite también la definición de variables por parte del usuario, una variable puede ser una cadena de texto, ua fecha o una fuente de datos de un componente (como una tabla o una gráfica), el usuario definirá la variable y asociará la variable a uno o más componentes de la plantilla que esté editando. El usuario podrá después optar por cargar una archivo en formato JSON para darle valor a las variables y reemplazarlas de forma automática en la aplicación, generando de este modo un informe completo en base a la plantilla que ha especificado, obteniendo un PDF con las variables reemplazadas en la plantilla.

La aplicación permitirá también generar múltiples informes dada la carga de múltiples archivos JSON para una misma plantilla, adicionalmente permitirá crear, guardar, modificar y borrar plantillas de informes, dicha funcionalidad corresponderá al servidor que utilizará una base de datos NoSQL para almacenar estos datos.

El producto contendrá documentación, incluyendo esta los manuales de despliegue y uso del software.

2.1.2. Criterios de aceptación del producto

El producto entregado al cliente se considerará adecuado y aceptado cuando cumpla las siguientes condiciones:

- Permite la creación de componentes de texto (inclusión de párrafos y títulos) en las plantillas.
- Permite la inclusión de listas y tablas en las plantillas.
- Permite la inclusión de imágenes en las plantillas.
- Permite definir el tamaño y la disposición de cada elemento que forme la plantilla.
- Permite la inclusión de gráficas (barras, área, líneas, sectores, radar y dispersión en dos variables) en la plantilla.
- Permite la inclusión de variables en la plantilla.
- Permite la carga de fuentes de datos en las gráficas, tablas y listas de una plantilla.

- Permite modificar cualquier componente que forme parte de una plantilla.
- Permite crear, guardar, modificar y borrar plantillas.
- Permite generar informes en base a plantillas reemplazando las variables que estén situadas en la plantilla dado un archivo en formato JSON.
- Permite generar múltiples informes en base a una plantilla dados múltiples archivos en formato JSON.
- Los informes generados por la aplicación se descargan en formato PDF.

2.1.3. Entregables del producto

El producto se compondrá de un único entregable final con todas las funcionalidades que el cliente haya especificado para pasar sus criterios de aceptación, este entregable incluirá:

- **Aplicación web (Cliente)**
- **Capa de servicios y base de datos (Servidor)**
- **Documentación de la aplicación**

Sin embargo se liberará versiones parciales durante el transcurso del proyecto que se utilizarán para mostrar al cliente el estado de desarrollo de la aplicación, estas versiones parciales están sujetas a cambios tanto por parte del cliente como por parte del desarrollador según se estime conveniente para pasar los criterios de aceptación del producto. Se define un número mínimo de 3 versiones parciales al cliente para informarle del estado de desarrollo del producto.

2.1.4. Restricciones del proyecto

Las restricciones del proyecto son las siguientes:

- El proyecto debe de realizarse en un tiempo aproximado de 421 horas.
- La aplicación debe de estar desarrollada con tecnologías web estándar (HTML5, CSS y JavaScript) o cualquier framework o librería que se derive de estas tecnologías.
- La base de datos utilizada debe de ser NoSQL
- Debe de utilizarse una arquitectura orientada a servicios REST para implementar las funcionalidades del servidor.
- La aplicación debe de poder tratar los archivos de formato JSON.

2.2. Gestión de riesgos

En todo proyecto de cierta consideración es necesario realizar una gestión de los distintos riesgos que puedan aparecer y que por ello pueden afectar de alguna manera al transcurso del proyecto.

Los riesgos que afectan a un proyecto pueden variar desde:

- **Riesgos específicos del proyecto** (Cambio de tecnologías, requisitos no correctamente especificados, diseños incorrectos...)
- **Riesgos genéricos** (Cambios de legislación, desastres naturales, bajas del personal, cancelación del proyecto...)

Cualquier riesgo que pueda afectar al proyecto debe de ser identificado y en caso de ser viable su tratamiento, tener algún plan para controlar el riesgo en caso de que se produzca.

A continuación se detallan los riesgos más importantes que pueden afectar a este proyecto.

2.2.1. Activos

Antes de empezar a identificar riesgos vinculados al proyecto es necesario identificar aquellos activos que son importantes para la realización del proyecto.

La tabla que representa un activo está compuesta por los siguientes campos:

- **ID**: Código identificador único del activo
- **Nombre**: Nombre del activo
- **Descripción**: Descripción del activo

Los activos identificados son los siguientes:

ID	AC-001
Nombre	Estación de trabajo
Descripción	La estación de trabajo es crucial para el desarrollo del proyecto tanto para generar la documentación del proyecto como para realizar la implementación de la aplicación que se va a desarrollar durante el transcurso del mismo, solo se cuenta con una actualmente.

ID	AC-002
Nombre	Código de la aplicación
Descripción	El código de la aplicación contendrá absolutamente toda la implementación que se ha diseñado en este proyecto, esto implica el código ejecutable que tanto el cliente como el servidor ejecutarán en sus respectivas fronteras de desarrollo para llevar a cabo sus funcionalidades.

ID	AC-003
Nombre	Base de datos
Descripción	La base de datos contendrá toda la información de operaciones de manipulación de datos que la aplicación cliente solicite al servidor.

ID	AC-004
Nombre	Repositorio del proyecto
Descripción	El repositorio contendrá las sucesivas versiones del código y de la documentación que se vayan generando durante el transcurso del proyecto.

2.2.2. Amenazas

Ya identificados los activos se procede a identificar las diferentes amenazas que pueden producirse en el contexto del proyecto actual.

La tabla que representa una amenaza está compuesta por los siguientes campos:

- **ID**: Código identificador único de la amenaza
- **Nombre**: Nombre de la amenaza
- **Descripción**: Descripción de la amenaza

Las amenazas identificadas son las siguientes:

ID	AM-001
Nombre	Cambio de tecnologías de desarrollo
Descripción	Las tecnologías que se utilizan para desarrollar la aplicación están en constante evolución, puede darse el caso de que salgan nuevas versiones de estas tecnologías y que estás sean incompatibles con las versiones anteriores

ID	AM-002
Nombre	Actualizaciones del sistema operativo
Descripción	Si se produce una actualización del sistema operativo puede darse el caso de que ciertas tecnologías o programas que se utilizan para desarrollar el proyecto dejen de funcionar o funcionen parcialmente.

ID	AM-003
Nombre	Fallo de hardware
Descripción	La documentación o código generado pueden perderse completamente o parcialmente por un fallo de hardware de la estación de trabajo, imposibilitando la opción de recuperación.

ID	AM-004
Nombre	Cambios de requisitos
Descripción	Se pueden producir cambios de requisitos a lo largo del transcurso del proyecto.

ID	AM-005
Nombre	Planificación optimista
Descripción	Cuando se planifica el proyecto se realizan estimaciones demasiado optimistas de modo que se tiene la idea de que las tareas se acabarán antes de lo previsto cuando en realidad no va a ser así.

ID	AM-006
Nombre	Planificación pesimista
Descripción	Cuando se planifica el proyecto se realizan estimaciones demasiado pesimistas, de modo que se tiene la idea de que las tareas acabarán mucho después de lo previsto cuando en realidad no va a ser así.

ID	AM-007
Nombre	Falta de implicación por parte de los interesados del proyecto
Descripción	Los principales interesados del proyecto no se involucran lo suficiente en su desarrollo, faltando *feedback* sobre el estado actual del mismo.

ID	AM-008
Nombre	Uso de nuevas tecnologías
Descripción	El uso de nuevas tecnologías puede producir retrasos en la planificación si no se cuenta con suficiente formación para utilizarlas a la velocidad normal.

ID	AM-009
Nombre	Alcance del proyecto sobredimensionado
Descripción	El alcance del proyecto es demasiado excesivo para el tiempo y presupuestos acordados, por lo que no se puede realizar de forma completa

ID	AM-010
Nombre	Diseño poco escalable
Descripción	Un diseño poco escalable dificulta la posibilidad de ampliar la aplicación durante el desarrollo de la misma, produciendo retrasos en la planificación

ID	AM-011
Nombre	Mala usabilidad de la interfaz
Descripción	Desarrollar una interfaz con mala usabilidad puede provocar el rechazo absoluto de la aplicación por parte del cliente o del usuario final.

2.2.3. Especificación de riesgos

Una vez identificados los activos y las amenazas se procede a identificar los riesgos del proyecto, por lo tanto se relacionan las amenazas con los activos y se le da a cada relación de este tipo una escala de probabilidad e impacto extrayendo de este modo un riesgo para el proyecto.

La tabla que define un riesgo está compuesta por los siguientes campos:

- **ID**: Código identificador único del riesgo
- **Amenazas involucradas**: Lista de amenazas involucradas en el riesgo (ID de las amenazas)
- **Activos involucrados**: Lista de activos involucrados en el riesgo (ID de los activos)

- **Probabilidad**: Probabilidad del riesgo (Cuan probable es que suceda)
- **Impacto**: Impacto del riesgo (Si se produce cuando daño puede llegar a realizar)
- **Exposición**: Factor calculado entre la probabilidad y el riesgo

A continuación se indican las distintas escalas y opciones de tratamiento de riesgos contempladas y utilizadas en este proyecto.

Las escalas de probabilidad que se han utilizado para medir los riesgos son las siguientes:

- **Alta**: El riesgo tiene una probabilidad muy alta de que pueda llegar a manifestarse, la posibilidad de aparición oscila entre el 81 % y el 100 %, ambos inclusive.
- **Media**: El riesgo tiene una probabilidad media de que pueda llegar a manifestarse, la posibilidad de aparición oscila entre el 36 % y el 80 %, ambos inclusive.
- **Baja**: El riesgo tiene una probabilidad baja de que pueda llegar a manifestarse, la posibilidad de aparición oscila entre el 0 % y el 35 %, ambos inclusive.

Las escalas de impacto utilizadas para medir los riesgos son las siguientes:

- **Catastrófico**: El riesgo produce un gran impacto en el proyecto, pudiendo producir la cancelación del mismo si llegase a producirse.
- **Alto**: El riesgo produce un gran impacto en el proyecto, pudiendo alterar enormemente su curso o su planificación si llegase a producirse.
- **Medio**: El riesgo produce un impacto medio en el proyecto, pudiendo alterar puntualmente ciertas acciones de transcurso del mismo.
- **Bajo**: El riesgo produce un impacto bajo en el proyecto, pudiendo alterar levemente algunas opciones del mismo.
- **Insignificante**: El riesgo apenas produce impacto alguno en el proyecto, sus consecuencias pueden ser ignoradas.

Las posibles acciones que se pueden realizar para tratar un riesgo en este proyecto son las siguientes:

- **Asumir**: No se toma ninguna acción con respecto al riesgo, simplemente se asume que va a suceder y que tendrá consecuencias en el proyecto.

- **Prevención**: Se crea un plan para reducir la probabilidad de que el riesgo ocurra durante el transcurso del proyecto.
- **Minimización**: Se crea un plan para reducir el impacto que el riesgo produce en caso de manifestarse.
- **Contingencia**: Se crea un plan que se ejecuta una vez el riesgo se ha manifestado en el proyecto, con el objetivo de reducir los efectos del riesgo una vez se ha producido.

Para priorizar los riesgos según la exposición que producen, se utiliza la siguiente matriz en la que se representan las distintas escalas de impacto en las filas y las escalas de probabilidad en las columnas de la matriz.

El valor de la exposición se calcula como el producto del valor cuantitativo asignado a cada escala, indicado entre paréntesis en la matriz.

	Alta (1)	Media (0.65)	Baja (0.15)
Catastrófico (1)	1	0.65	0.15
Alto (0.8)	0.8	0.52	0.12
Medio (0.5)	0.5	0.325	0.075
Bajo (0.3)	0.3	0.195	0.045
Insignificante (0.1)	0.1	0.065	0.015

Los riesgos que presenten un valor de exposición más alto serán los que tengan la prioridad más alta para aplicarles medidas de tratamiento.

Se consideraron las siguientes escalas de exposición:

- **Alta**: Valor mayor o igual que 0.65.
- **Media**: Valor menor que 0.65 y mayor o igual que 0.15.
- **Baja**: Valor menor que 0.15.

2.2.4. Identificación de riesgos

A lo largo del proyecto se han identificado los siguientes riesgos:

ID	RIE-001
Amenazas involucradas	**AM-001**
Activos involucrados	*AC-002, AC-003, AC-004*
Probabilidad	Alta
Impacto	Alto
Exposición	Alta

ID	RIE-002
Amenazas involucradas	**AM-002**
Activos involucrados	*AC-001, AC-002, AC-003*
Probabilidad	Baja
Impacto	Medio
Exposición	Baja

ID	RIE-003
Amenazas involucradas	**AM-003**
Activos involucrados	*AC-001, AC-002, AC-003*
Probabilidad	Baja
Impacto	Catastrófico
Exposición	Media

ID	RIE-004
Amenazas involucradas	**AM-004**
Activos involucrados	*AC-002, AC-003*
Probabilidad	Alta
Impacto	Medio
Exposición	Media

ID	RIE-005
Amenazas involucradas	**AM-005**
Activos involucrados	*AC-002*
Probabilidad	Alta
Impacto	Alto
Exposición	Alta

ID	RIE-006
Amenazas involucradas	**AM-006**
Activos involucrados	*AC-002*
Probabilidad	Alta
Impacto	Bajo
Exposición	Baja

ID	RIE-007
Amenazas involucradas	**AM-007**
Activos involucrados	*AC-002*
Probabilidad	Media
Impacto	Catastrófico
Exposición	Alta

ID	RIE-008
Amenazas involucradas	**AM-008**
Activos involucrados	*AC-002, AC-003, AC-004*
Probabilidad	Alta
Impacto	Medio
Exposición	Media

ID	RIE-009
Amenazas involucradas	**AM-009**
Activos involucrados	*AC-002, AC-003*
Probabilidad	Media
Impacto	Alta
Exposición	Media

ID	RIE-010
Amenazas involucradas	**AM-010**
Activos involucrados	*AC-002*
Probabilidad	Media
Impacto	Alto
Exposición	Media

ID	RIE-011
Amenazas involucradas	**AM-011**
Activos involucrados	*AC-002*
Probabilidad	Alta
Impacto	Catastrófico
Exposición	Alta

2.2.5. Tratamiento de riesgos

Dados los riesgos contemplados en la sección anterior se ha procedido a elaborar los siguientes planes de tratamiento, priorizando los riesgos con mayor

exposición.

Los campos de la tabla de un tratamiento son los siguientes:

- **ID**: Código identificador único del tratamiento
- **Riesgos involucrados**: Lista de riesgos involucrados en el tratamiento (ID del riesgo)
- **Tipo de plan**: Define el tipo del tratamiento
- **Descripción**: Descripción del tratamiento
- **Probabilidad actual**: Probabilidad de suceso de los riesgos si se aplica el tratamiento
- **Impacto actual**: Impacto de los riesgos si se aplica el tratamiento
- **Exposición actual**: Exposición de los riesgos si se aplica el tratamiento

ID	TRA-001
Riesgo(s) involucrado	**RIE-001 y RIE-002**
Tipo de plan	Minimización
Descripción del plan	Para evitar incompatibilidades entre futuras versiones se procederá a descargar y conservar de forma integra versiones **LTS** (*Long Term Support*) de las distintas tecnologías que se usen, de modo que estas serán menos propensas a cambiar, o en caso de cambiar no aseguran incompatibilidades dentro de la misma versión.
Probabilidad actual	Alta
Impacto actual	Insignificante
Exposición actual	Baja

ID	TRA-002
Riesgo(s) involucrado	**RIE-005**
Tipo de plan	Minimizar
Descripción del plan	Para evitar estimaciones demasiado optimistas se añadirá un margen de tiempo de una semana a cada tarea que implique desarrollo en tecnologías inexpertas por parte del desarrollador.
Probabilidad actual	Alta
Impacto actual	Bajo
Exposición actual	Baja

ID	TRA-003
Riesgo(s) involucrado	**RIE-007**
Tipo de plan	Prevención y Minimización
Descripción del plan	Mantener reuniones de forma planificada con los interesados e informarse sobre su disponibilidad y horarios para maximizar la eficiencia de las mismas. Adicionalmente intentar esclarecer lo máximo posible los requisitos que se puedan derivar de la influencia de los interesados para en caso de no poder reunirse tener claro la funcionalidad a realizar.
Probabilidad actual	Baja
Impacto actual	Medio
Exposición actual	Baja

ID	TRA-004
Riesgo(s) involucrado	**RIE-011**
Tipo de plan	Prevención y Minimización
Descripción del plan	Creación de sucesivos prototipos de la interfaz tanto funcionales como no funcionales y enseñárselos al cliente o usuario para comprobar su usabilidad y si el diseño sigue una dirección correcta.
Probabilidad actual	Baja
Impacto actual	Medio
Exposición actual	Bajo

ID	TRA-005
Riesgo(s) involucrado	**RIE-003**
Tipo de plan	Minimización
Descripción del plan	Se guardará el código de la aplicación y de la memoria en un repositorio en remoto, adicionalmente se tendrá múltiples copias del trabajo en dispositivos de almacenamiento externos.
Probabilidad actual	Baja
Impacto actual	Insignificante
Exposición actual	Baja

ID	TRA-009
Riesgo(s) involucrado	**RIE-009**
Tipo de plan	Prevención
Descripción del plan	Se esclarecerá el alcance exacto del proyecto con el cliente dado el tiempo y el presupuesto disponible para de este modo evitar que el alcance exceda alguna de las dos limitaciones.
Probabilidad actual	Baja
Impacto actual	Alta
Exposición actual	Baja

ID	TRA-010
Riesgo(s) involucrado	**RIE-010**
Tipo de plan	Contingencia
Descripción del plan	Si se llega a producir el caso de que el diseño no se pueda escalar, se procede a volver a una versión anterior en la que el diseño no estaba realizado o aún no se había realizado de forma incorrecta, y se adapta para que sea escalable.
Probabilidad actual	Medio
Impacto actual	Bajo
Exposición actual	Baja

Los riesgos no contemplados en los tratamientos se asumen, de modo que no existe ningún plan para tratarlos en caso de que ocurran o puedan ocurrir.

2.3. Metodología de desarrollo

A la hora de desarrollar un proyecto, una de las fases más importantes y de la cual depende prácticamente su éxito o fracaso es la elección de una metodología, o ciclo de vida para desarrollar el proyecto.

Existen multitud de ciclos de vida que se pueden elegir para desarrollar un proyecto, cada uno de ellos presenta sus ventajas e inconvenientes y la elección de uno frente a otro depende siempre del contexto del proyecto al que se intente aplicar.

Puesto que la elección del ciclo de vida no debe de hacerse a la ligera, ya que condiciona fuertemente el transcurso del proyecto, se debe de realizar un previo análisis sobre el contexto del proyecto y a partir de este análisis decidir que ciclo de vida cuadra mejor para la situación expuesta.

Analizando el contexto del proyecto nos encontramos con los siguientes puntos:

- Los requisitos no están bien definidos en las fases iniciales y medias del proyecto, pues el cliente tiene una idea general de como solventar su problema pero hasta que esta no vaya tomando forma no se tendrá una especificación clara de los requisitos.
- Se utilizan tecnologías en la que el desarrollador no es experto, dichas tecnologías son útiles, avanzadas y modernas por lo que pueden ahorrar tiempo una vez que el desarrollador haya adquirido experiencia en su uso, sin embargo las etapas iniciales se dedicarán a formación y aunque esta se complete exitosamente existirán traspiés por falta de experiencia.
- El calendario del proyecto es cambiante y está lleno de otros compromisos, pues el único desarrollador del proyecto tiene otro tipo de compromisos pendientes que coinciden con el desarrollo del proyecto, por lo que se pueden producir retrasos en las planificaciones.
- Se necesita avanzar rápido en el desarrollo para poder enseñar al cliente el estado de avance del proyecto y así de este modo esclarecer mejor los requisitos que se van a definir.
- Solo se cuenta con un único recurso humano para desarrollar todo el proyecto.
- El cliente está disponible para reuniones, sin embargo no se pude saturar su agenda actual con exceso de reuniones, estas deben de realizarse de forma moderada y planificada.

En base a las características analizadas anteriormente se puede concluir que la mejor elección para desarrollar el proyecto es la elección de una ***metodología ágil***, descartando todas las metodologías *pesadas* o tradicionales, como cascada, incrementos o espiral.

El motivo de esta elección es que los requisitos no están bien definidos y probablemente se produzcan cambios bruscos en la planificación según estos se van definiendo, además es necesario liberar funcionalidades de forma periódica al cliente para obtener información sobre el estado de avance del proyecto.

Dentro de las metodologías ágiles existen dos posibilidades que encajan bien dentro del contexto del proyecto, estas posibilidades son: ***scrum*** y **programación extrema**.

Aunque las dos presentan grandes similitudes entre sí para este proyecto nos

decantaremos por **programación extrema**, el motivo de esta elección es que ***scrum*** delimita mucho el margen de maniobra en las iteraciones con los *sprints*, en cambio **programación extrema** permite más flexibilidad al utilizar iteraciones cortas orientadas a prototipos.

Adicionalmente, ***scrum*** requiere un gran número de reuniones con los clientes, y aunque es posible reunirse con ellos no conviene saturar su agenda con reuniones constantes ya que tienen otros compromisos que atender.

Por ello se selecciona como metodología de desarrollo **programación extrema**, esta metodología se orienta a desarrollar desde un enfoque incremental utilizando iteraciones cortas para liberar funcionalidades de forma continua al cliente o usuario. Esto último coincide con la necesidad de: 1) enseñar al cliente la evolución del proyecto, 2) los requisitos no están bien especificados en las fases iniciales y 3) la planificación tiende a cambiar según los requisitos cambian.

Además, **programación extrema** es una metodología orientada a pruebas y centrada en la refactorización del código que se desarrolle durante el transcurso del proyecto, de modo que primero se programan las pruebas que deben de pasar las funcionalidades y luego se programa el código para que pase las pruebas programadas sin importar cuantas veces se tenga que reescribir, mejorar o evolucionar dicho código. De modo que se satisfacen las necesidades de: 1) se utilizan tecnologías nuevas y no se cuenta con experiencia en su uso y 2) solo se cuenta con un recurso para desarrollar todo el proyecto.

Los incrementos que se contemplan en este proyecto son los siguientes:

- **Incremento 1 (*Aplicación web*)**: Constituye la realización de la aplicación web de edición y creación de informes y plantillas de informes con todas sus funcionalidades.

- **Incremento 2 (*Capa de servicios*)**: Consta de la implementación de la capa de servicios web que utilizará el cliente para enviar y recibir datos al servidor, en concreto almacenamiento y obtención de plantillas e informes creados en la aplicación web.

- **Incremento 3 (*Entorno y base de datos*)**: Creación de la base de datos para almacenar los datos recibidos en la capa de servicios junto con la puesta en marcha del entorno completo con la aplicación web y la capa de servicios.

2.4. Planificación

En esta sección se especifica la planificación de la duración total del proyecto, esta viene especificada por la distribución de las tareas que componen el desarrollo completo del proyecto junto con el número de horas estimadas para su realización.

La especificación de las tareas, o paquetes de trabajo, se detalla en la *Estructura de descomposición de trabajo* mientras que la duración de cada una de ellas y su distribución en el tiempo se especifica en el *Cronograma*.

2.4.1. Estructura de descomposición del trabajo (EDT)

Puesto que el proyecto incluye un amplio rango de tareas a realizar es preciso descomponer estas en paquetes de trabajo manejables y más pequeños que permitan estimar de forma efectiva su duración y trabajo requerido para realizarlos, además esta descomposición de tareas servirá también para definir de forma esquemática el trabajo que se realizará durante el transcurso del proyecto.

En la figura 2.1 se muestra el nivel más alto de la descomposición del trabajo a realizar, el proyecto en su conjunto está formado por 4 grandes paquetes de trabajo: la *definición del proyecto*, la *aplicación web*, la *capa de servicios* y el *entorno y la base de datos*.

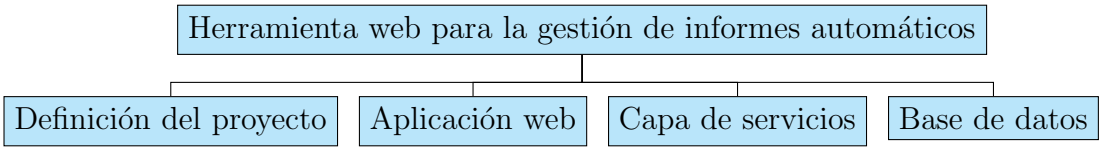

Figura 2.1: Primer nivel de la estructura de descomposición de trabajo

El paquete de trabajo *definición del proyecto* definido en la figura 2.2 se compone de las tareas relativas a la gestión del proyecto, análisis de tecnologías y análisis de requisitos, esto incluye la *gestión de riesgos* del proyecto, la *definición del alcance* del proyecto, el *análisis de requisitos* de la aplicación y el *análisis de tecnologías* que se utilizarán durante la fase de desarrollo del proyecto.

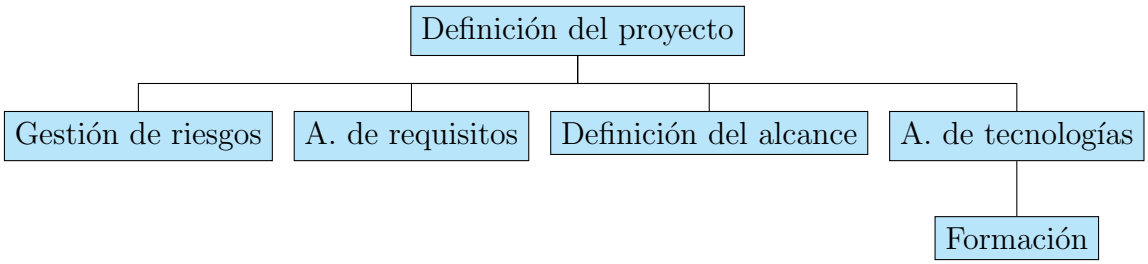

Figura 2.2: Descomposición del paquete de trabajo *Definición del proyecto*

El primer incremento corresponde a la *Aplicación web* y está definido en la figura 2.3.

Este incremento está compuesto de las tareas relativas a la realización de la interfaz web que dará soporte a la creación y gestión de informes automáticos, se prevé que esté sea el paquete de trabajo más complejo y por ello el que más tiempo necesite para ser completado.

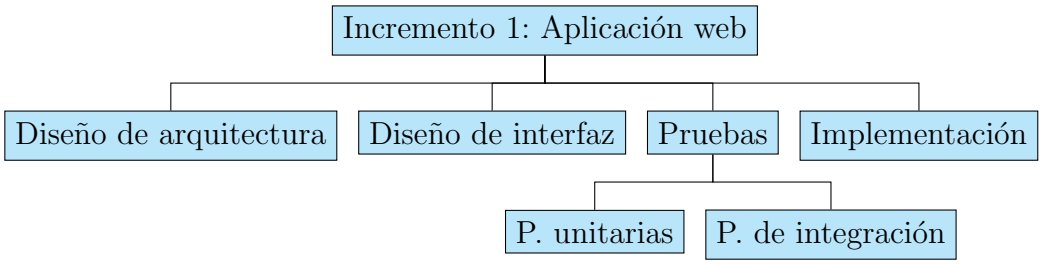

Figura 2.3: Descomposición del paquete de trabajo *Aplicación web*

En este incremento se engloban las tareas relativas a la elaboración del diseño de la aplicación, dividido en dos tareas, el *diseño de la arquitectura del sistema* que engloba el diseño interno y más funcional de la aplicación y el *diseño de la interfaz* que engloba el diseño visual que el usuario observa para utilizar la aplicación.

Además, se incluye en este paquete la tarea de *implementación del sistema* que se realizará una vez se hayan completado los paquetes de diseño, este paquete se corresponde con la implementación funcional de los requisitos y el enlace de dichas funcionalidades con la interfaz de usuario.

Finalmente está el paquete correspondiente a la fase de pruebas, dividida en dos tareas, las *pruebas unitarias*, que comprueban las funcionalidades de forma aislada y las *pruebas de integración* que comprueban que la aplicación de forma autónoma funciona correctamente.

El segundo incremento corresponde a la *Capa de servicios*, representado en la figura 2.4.

Este incremento comprende el desarrollo de las funcionalidades de almacenamiento e impresión de informes situadas en el servidor, esta tarea está subdividida en los paquetes de *diseño de la capa de servicios* en la que se diseñará de forma abstracta los servicios que contendrá el servidor y el modo de utilizarlos.

A continuación se realizará la *implementación de la capa de servicios* en la que se incluirán de forma funcional los diseños anteriores, finalizando en las pruebas para comprobar que la capa de servicios funciona de forma correcta, componiéndose esta última fase a las pruebas unitarias y de integración.

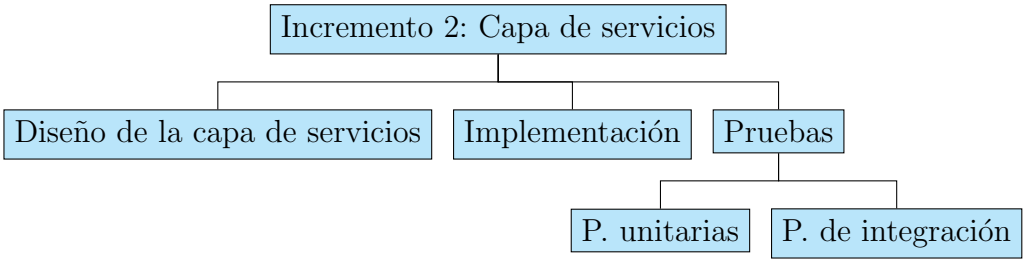

Figura 2.4: Descomposición del paquete de trabajo *Capa de servicios*

El tercer incremento es el *Entorno y base de datos* representado en al figura 2.5.

Este incremento engloba las tareas relativas a montar el entorno de despliegue de la aplicación junto con la base de datos utilizada por la capa de servicios, esta tarea se divide en los paquetes de trabajo: *Implementación del entorno* y *Implementación de la base de datos*.

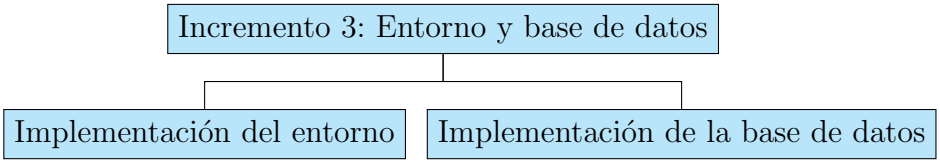

Figura 2.5: Descomposición del paquete de trabajo *Entorno y base de datos*

El primer paquete engloba las tareas relativas a instalar, configurar e iniciar el entorno de despliegue de las aplicaciones mientras que el segundo está destinado a las acciones relativas a la selección e implementación de una base de datos NoSQL en el entorno previamente instalado.

2.4.2. Cronograma (Planificación inicial)

La planificación que se muestra a continuación es la planificación **inicial** del proyecto, sin que se produjese ningún riesgo durante el desarrollo y atendiendo únicamente a las estimaciones iniciales.

Identificados y analizados los distintos paquetes de trabajo que componen el proyecto en su totalidad se procede a realizar la planificación o distribución temporal del proyecto, para ello se generaron los siguientes cronogramas indicando la duración estimada en horas de cada uno de los paquetes de trabajo y de las tareas que componen dichos paquetes de trabajo.

En la figura 2.6 se muestra el cronograma general del proyecto, la estimación incluye una duración total aproximada de 426 horas, el proyecto está compuesto de los paquetes de trabajo anteriormente identificados más la elaboración de la documentación del proyecto, esta última tarea se realiza en paralelo con los paquetes de trabajo durante todo el transcurso del proyecto.

Nombre	Trabajo	Inicio	Terminado
Herramienta web para la gestión de	426 horas	**9/03/18 8:00**	**22/06/18 13:00**
Definición del proyecto	32 horas	**9/03/18 8:00**	**19/03/18 17:00**
Definición del alcance del proyecto	8 horas	9/03/18 8:00	9/03/18 17:00
Análisis de requisitos	8 horas	12/03/18 8:00	13/03/18 17:00
Análisis de tecnologías	8 horas	14/03/18 8:00	15/03/18 17:00
Gestión de riesgos	8 horas	16/03/18 8:00	19/03/18 17:00
Aplicación web	338 horas	**20/03/18 8:00**	**4/06/18 17:00**
Diseño de arquitectura	12 horas	20/03/18 8:00	26/03/18 17:00
Diseño de la interfaz	12 horas	27/03/18 8:00	2/04/18 17:00
Implementación	290 horas	3/04/18 8:00	28/05/18 17:00
Pruebas unitarias	12 horas	29/05/18 8:00	31/05/18 17:00
Pruebas de integración	12 horas	1/06/18 8:00	4/06/18 17:00
Capa de servicios	36 horas	**5/06/18 8:00**	**18/06/18 17:00**
Diseño de la capa de servicios	12 horas	5/06/18 8:00	7/06/18 17:00
Implementación	18 horas	8/06/18 8:00	14/06/18 17:00
Pruebas unitarias	4 horas	15/06/18 8:00	15/06/18 17:00
Pruebas de integración	2 horas	18/06/18 8:00	18/06/18 17:00
Entorno y base de datos	20 horas	**19/06/18 8:00**	**22/06/18 13:00**
Implementación del entorno	16 horas	19/06/18 8:00	21/06/18 17:00
Implementación de la base de datos	4 horas	22/06/18 8:00	22/06/18 13:00
Documentación del proyecto	426 horas	9/03/18 8:00	22/06/18 17:00
Fin del proyecto	0 horas	22/06/18 17:...	22/06/18 17:00

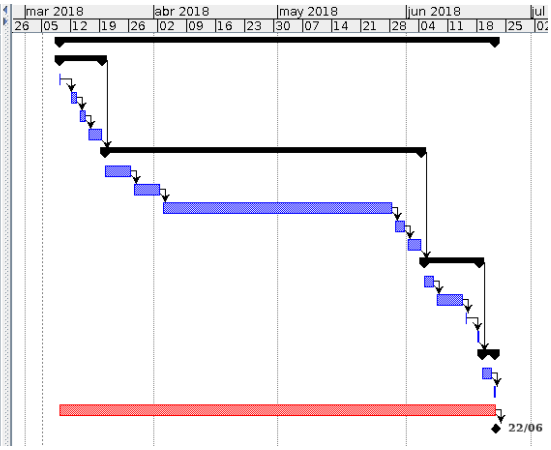

Figura 2.6: Cronograma general del proyecto

La primera fase del proyecto es la *definición del proyecto* (Figura 2.7), esta tarea tiene una duración aproximada de 32 horas, una vez finalizada esta tarea se procede a empezar con el paquete de trabajo más laborioso y complejo de todo el proyecto, la *aplicación web* (Figura 2.8), este paquete de trabajo tiene una duración aproximada de 338 horas y compondrá el núcleo del proyecto, su duración será más elevada que la de los demás paquetes debido a que implica mayor esfuerzo de desarrollo y diseño para poder completarse.

Una vez finalizada la implementación de la fase anterior se procede a realizar el paquete de trabajo *capa de servicios* (Figura 2.9), este paquete de trabajo tiene una duración aproximada de 36 horas debido a que las funcionalidades con-

templadas son pocas y fáciles de programar, adicionalmente se cuenta experiencia por parte del desarrollador en esta parte.

Finalmente, se procede a realizar el último paquete de trabajo, *entorno y base de datos* (Figura 2.10), este paquete tiene una duración estimada de 20 horas y consiste en implementar la base de datos y el entorno de despliegue fuera del entorno de desarrollo normal de la aplicación, al acabar este paquete de trabajo y la documentación asociada al proyecto se alcanzaría el hito de *Fin del proyecto* y este quedaría finalizado.

⊟Definición del proyecto	32 horas	**9/03/18 8:00**	**19/03/18 17:00**
Definición del alcance del proyecto	8 horas	9/03/18 8:00	9/03/18 17:00
Análisis de requisitos	8 horas	12/03/18 8:00	13/03/18 17:00
Análisis de tecnologías	8 horas	14/03/18 8:00	15/03/18 17:00
Gestión de riesgos	8 horas	16/03/18 8:00	19/03/18 17:00

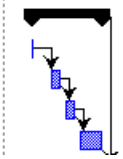

Figura 2.7: Cronograma del paquete de trabajo *Definición del proyecto*

⊟Aplicación web	338 horas	**20/03/18 8:00**	**4/06/18 17:00**
Diseño de arquitectura	12 horas	20/03/18 8:00	26/03/18 17:00
Diseño de la interfaz	12 horas	27/03/18 8:00	2/04/18 17:00
Implementación	290 horas	3/04/18 8:00	28/05/18 17:00
Pruebas unitarias	12 horas	29/05/18 8:00	31/05/18 17:00
Pruebas de integración	12 horas	1/06/18 8:00	4/06/18 17:00

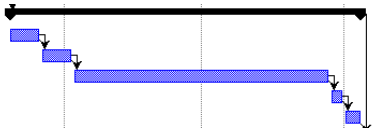

Figura 2.8: Cronograma del paquete de trabajo *Aplicación web*

⊟Capa de servicios	36 horas	**5/06/18 8:00**	**18/06/18 17:00**
Diseño de la capa de servicios	12 horas	5/06/18 8:00	7/06/18 17:00
Implementación	18 horas	8/06/18 8:00	14/06/18 17:00
Pruebas unitarias	4 horas	15/06/18 8:00	15/06/18 17:00
Pruebas de integración	2 horas	18/06/18 8:00	18/06/18 17:00

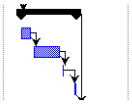

Figura 2.9: Cronograma del paquete de trabajo *Capa de servicios*

⊟Entorno y base de datos	20 horas	**19/06/18 8:00**	**22/06/18 13:00**
Implementación del entorno	16 horas	19/06/18 8:00	21/06/18 17:00
Implementación de la base de datos	4 horas	22/06/18 8:00	22/06/18 13:00

Figura 2.10: Cronograma del paquete de trabajo *Entorno y base de datos*

2.4.3. Cronograma (Planificación real)

Durante la ejecución del proyecto se produjeron dos variaciones significativas en el cronograma de la planificación inicial, provocando un desajuste en la planificación.

Los riesgos que se produjeron fueron dos:

- Duración estimada para una tarea demasiado optimista.
- Acumulación de tareas para el único recurso disponible.

El primer riesgo produjo un desajuste de 15h en la implementación de la aplicación web, debido al tratamiento de minimización aplicado ante estas situaciones de planificaciones optimistas el desajuste no fue demasiado importante y apenas impactó la planificación.

El segundo riesgo produjo un retraso de 1 mes en la planificación (el mes de Mayo), debido a una acumulación de tareas no relativas al proyecto por parte del desarrollador.

En este mes no se pudo avanzar en el desarrollo del proyecto, lo que produjo que no se continuara con el hasta principios de junio. Este riesgo no estaba tratado debido a que no existen medidas posibles para solventarlo de forma satisfactoria, por lo que se decidió asumir sus consecuencias.

La planificación real se muestra en la figura 2.11

⊟Herramienta web para la gestión de	**441 horas**	**9/03/18 8:00**	**20/07/18 13:00**
⊟Definición del proyecto	**32 horas**	**9/03/18 8:00**	**19/03/18 17:00**
Definición del alcance del proyecto	8 horas	9/03/18 8:00	9/03/18 17:00
Análisis de requisitos	8 horas	12/03/18 8:00	13/03/18 17:00
Análisis de tecnologías	8 horas	14/03/18 8:00	15/03/18 17:00
Gestión de riesgos	8 horas	16/03/18 8:00	19/03/18 17:00
⊟Aplicación web	**353 horas**	**20/03/18 8:00**	**2/07/18 17:00**
Diseño de arquitectura	12 horas	20/03/18 8:00	26/03/18 17:00
Diseño de la interfaz	12 horas	27/03/18 8:00	2/04/18 17:00
Implementación	305 horas	3/04/18 8:00	25/06/18 17:00
Pruebas unitarias	12 horas	26/06/18 8:00	28/06/18 17:00
Pruebas de integración	12 horas	29/06/18 8:00	2/07/18 17:00
⊟Capa de servicios	**36 horas**	**3/07/18 8:00**	**16/07/18 17:00**
Diseño de la capa de servicios	12 horas	3/07/18 8:00	5/07/18 17:00
Implementación	18 horas	6/07/18 8:00	12/07/18 17:00
Pruebas unitarias	4 horas	13/07/18 8:00	13/07/18 17:00
Pruebas de integración	2 horas	16/07/18 8:00	16/07/18 17:00
⊟Entorno y base de datos	**20 horas**	**17/07/18 8:00**	**20/07/18 13:00**
Implementación del entorno	16 horas	17/07/18 8:00	19/07/18 17:00
Implementación de la base de datos	4 horas	20/07/18 8:00	20/07/18 13:00
Documentación del proyecto	441 horas	9/03/18 8:00	23/07/18 9:00
Fin del proyecto	0 horas	23/07/18 9:00	23/07/18 9:00

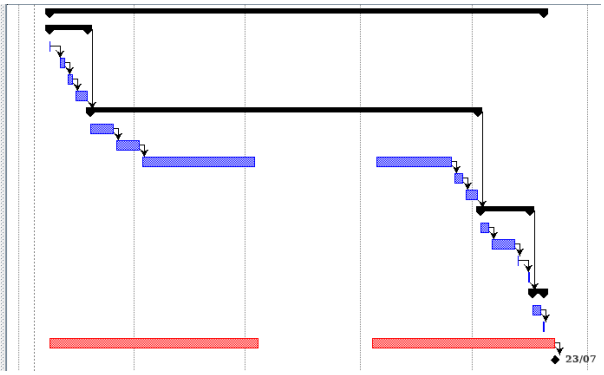

Figura 2.11: Planificación real del proyecto

2.5. Gestión de la configuración

Los diferentes elementos que constituyen el software que se va a construir en este proyecto deben de estar sometidos a una gestión de la configuración, esto incluye controlar las diferentes versiones del código o de los elementos del software que se generen durante el transcurso del proyecto, también se debe de asegurar la integridad de las versiones en caso de que sea necesario volver a alguna versión anterior del código generado.

Además, es necesario gestionar los cambios que se realizan durante la construcción

del código del software, el modo de aplicar estos cambios debe de estar gestionado por un proceso de gestión de cambios en el que se indica el modo de realizar los cambios.

En esta sección se indican las distintas herramientas o procesos que se utilizan para realizar la correcta gestión de la configuración de los elementos del software de este proyecto.

2.5.1. Elementos de configuración

Antes de proceder a definir el sistema de gestión de la configuración y el proceso de gestión de cambios es necesario identificar que elementos de configuración serán los gestionados por nuestro proceso de gestión de la configuración.

Los elementos de configuración que se han identificado en este proyecto son los siguientes:

- **Código de la aplicación cliente**
- **Código del servidor**
- **Memoria del proyecto**

Estos tres elementos son los que van a ser gestionados por el proceso de gestión de la configuración puesto que son propensos a sufrir cambios y van a generarse distintas versiones de cada uno de ellos a lo largo del transcurso del proyecto, adicionalmente es crucial mantener su integridad en caso de que sea necesario volver a una versión anterior de uno de ellos.

2.5.2. Sistema de gestión de la configuración

Para gestionar la integridad de las versiones junto con las distintas versiones del código generado se utiliza el software de control de versiones *Git*.

El repositorio de los archivos estará situado en la plataforma **Bitbucket** de *Atlassian* [1], que permite de forma gratuita la creación de repositorios privados o públicos según las necesidades del usuario.

En el repositorio se almacenará tanto el código del cliente y del servidor como la memoria del proyecto.

[1] `https://bitbucket.org/`

2.5.3. Estructura del repositorio

La estructura del repositorio que contendrá los elementos de la configuración es la siguiente:

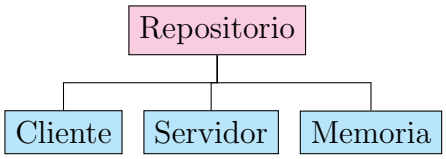

Figura 2.12: Estructura del repositorio

En la carpeta ***Cliente*** se incluye el código de la aplicación relativo a la interfaz web (*Front end*), junto con sus pruebas y archivos auxiliares (imágenes, iconos, librerías, etc.).

En la carpeta ***Servidor*** se incluye el código de la capa de servicios REST (*Back end*), adicionalmente se incluyen las librerías o archivos auxiliares que se necesiten para que el código sea funciona.

Finalmente en la carpeta ***Memoria*** se añaden los archivos relativos a la presente memoria, esto incluye el documento actual junto con las figuras y los diagramas que se incluyen en el.

Para el cliente y el servidor se utilizó adicionalmente NPM (*Node Package Manager*), de modo que existe una carpeta llamada *node_modules* que contiene los paquetes instalados por esta herramienta junto con el árbol de dependencias en un archivo llamado *package.json*, esto es utilizado por NPM para instalar y controlar las dependencias de la aplicación, de modo que solo es necesario subir al repositorio el código fuente base de la aplicación junto con el archivo que indica sus dependencias.

2.5.4. Gestión de cambios

En este proyecto solo existe un colaborador, por lo que no se puede dar la posibilidad de incompatibilidades entre cambios a diferentes versiones del código, por ello no será necesario aplicar un proceso de gestión de cambios complejo al proyecto ni será necesario la creación de un sistema de gestión de cambios completo.

En su lugar los cambios en el código se limitan a simples anotaciones escritas por el propio colaborador para indicar los cambios, estas etiquetas será registradas por el software de control de versiones, de modo que es posible seguir su

eliminación y añadido de forma sencilla.

Las anotaciones contempladas son las siguientes:

- **TODO**: Indica que la funcionalidad o cambio aun no se ha realizado y está pendiente de realizarse.
- **FIXME**: Indicia que hay un fallo en una funcionalidad o sección del código que no está corregida.
- **TESTME**: Indica que falta comprobar el funcionamiento de la funcionalidad ante casos de prueba.

2.6. Gestión de costes

En este apartado se procederá a realizar el análisis de los costes del proyecto, cabe mencionar que aunque este proyecto es de ámbito académico y solo será desarrollado por una persona es importante realizar este tipo de gestiones pues según *The Standish Group* [1], un 31 % de los proyectos de software fracasan y no se llegan a completar, mientras que un 52.7 % terminan con un 189 % de exceso de costes, quedando solo un 16 % de proyectos que terminan sin exceso de costes y en el tiempo asignado.

El análisis que se realizará en esta sección tendrá como objetivo contabilizar los costes del proyecto si este se desarrollase en un ámbito no académico.

Los tipos de costes contemplados para este proyecto son los siguientes:

- **Costes directos**: Aquellos fácilmente derivables de la realización del proyecto, son fáciles de calcular y de estimar puesto que están relacionados con el propio proyecto, como el salario de los recursos humanos.
- **Costes indirectos**: Son difíciles de calcular puesto que no están directamente relacionados con el proyecto y suelen ser tangenciales al mismo, como el consumo eléctrico o de agua.

2.6.1. Costes directos

Los costes directos atribuibles a este proyecto son los correspondientes al salario del desarrollador del mismo, los tutores del proyecto desempeñan el rol de clientes para este proyecto de modo que no se tienen en cuenta como posibles recursos humanos a la hora de calcular los costes del personal.

Basándonos en el estudio realizado por *Vitae Consultores* [2], tomando como

referencia la categoría de **Programador Front-end (HTML5, JavaScript)** sin experiencia (Junior 0-2), con un sueldo medio de **16.000 €** anuales y asumiendo un coste del 31 % sobre el salario bruto para la seguridad social se obtendrían los siguientes costes relativos al personal.

Se asumen 20 días laborables al mes y jornadas de 8 horas de trabajo.

Salario bruto	Seg. Social	Total anual	Coste mensual	Coste hora
16.000 €	4960 €	20.960 €	1746,66 €	10,91 €

Puesto que el proyecto tendrá una duración media de 4 meses, el coste del personal sería de **6986,64 €**. Adicionalmente es necesario incluir como costes directos 4 copias de la presente memoria y el CD con el código de la aplicación, estimando que la memoria tendrá una extensión de 150 páginas el coste de todo lo anterior de forma estimada sería de **120 €**.

Por ello el valor de los costes directos asciende en total a **7106,64 €**.

2.6.2. Costes indirectos

Como se mencionó anteriormente los costes indirectos son difíciles de estimar y de atribuir al proyecto puesto que son tangenciales al transcurso del mismo.

Como costes indirectos del proyecto tendremos los programas de software utilizados para el desarrollo del mismo, que aunque fueron obtenidos bajo licencias de prueba o licencias de uso académico se debe de incluir su coste real puesto que en un proyecto real sería necesario utilizar licencias de pago comerciales.

El coste de los diferentes programas utilizados para desarrollar el proyecto se incluye en el siguiente cuadro:

Nombre	Coste
Webstorm	49 €
StarUML 2	60 €
TexStudio	0 €
Ubuntu Linux	0 €
MongoDB	0 €
Docker	0 €
ReactJS	0 €
NodeJS	0 €
Ant Design	0 €

Además es necesario incluir el coste de la estación de trabajo, que tuvo un coste de adquisición de **485 €**, asumiendo una vida útil de 5 años se obtuvieron los siguientes costes:

Coste	Meses de vida útil	Meses de uso	Coste atribuible al proyecto
485 €	60	4	32,33 €

Finalmente es necesario incluir los costes relativos al uso de electricidad, agua, Internet, desplazamientos etc. Estos supondrán un **10 %** extra a los costes finales obtenidos.

En resumen, como costes indirectos derivados del software y de la estación de trabajo se obtiene la cantidad de **141,33 €** más el **10 %** extra a los costes finales como derivación de costes relativos a electricidad, agua, Internet, etc.

2.6.3. Costes totales

El coste total del proyecto se muestra en el siguiente cuadro:

	Coste
Costes directos	7106,64 €
Costes indirectos	141,33 €
Electricidad, Internet, ...	10 %
Total	7972,77 €

Por ello, el coste total del proyecto asciende a **7972,77 €**.

2.7. Gestión de las comunicaciones

A la hora de desarrollar cualquier proyecto siempre se deben de tener en cuenta los interesados, o *stakeholders*, pues su influencia en el proyecto puede ser notoria y puede forzar a que el proyecto fracase o triunfe dependiendo del tipo de influencia que el interesado pueda ejercer sobre el transcurso del mismo.

Por ello se deben de identificar los interesados que influyan de algún modo en el transcurso o desarrollo del proyecto, la identificación de los interesados para este proyecto se muestra en el cuadro 2.1, en el se indican los distintos interesados junto con sus datos de contacto, localización y rol de desempeñado en este proyecto.

Además de identificar los interesados es necesario analizar que tipo de objetivos poseen de cara al proyecto así como su nivel de influencia y función como

interesado, esto sería la evaluación de los interesados identificados. En el cuadro 2.2 se realiza la evaluación de los interesados del proyecto, indicando la función de los interesados, sus expectativas de cara al proyecto, su nivel de influencia y las fases de mayor interés.

Finalmente, es necesario clasificar los interesados anteriormente identificados, la clasificación tiene como objetivo analizar que tipo de apoyo muestran los interesados de cara al proyecto, es decir, si apoyan el proyecto, si son neutrales a su realización o se si oponen a su realización, dichas clasificación se encuentra recogida en el cuadro 2.3, donde se indica el nivel de apoyo del proyecto y el origen del mismo.

ID	Función	Expectativas	Influencia	Fase de mayor interés
I-001	Proporcionar tutorización y resolución de dudas relativas al proyecto	Finalización exitosa del proyecto	Media	Todas
I-002	Proporcionar tutorización y resolución de dudas relativas al proyecto	Finalización exitosa del proyecto	Media	Todas
I-003	Proporcionar tutorización y resolución de dudas relativas al proyecto	Finalización exitosa del proyecto	Alta	Todas
I-004	Desarrollar todas las fases del proyecto	Finalización exitosa del proyecto	Alta	Todas

Cuadro 2.2: Tabla de evaluación de interesados del proyecto

ID	Origen	Nivel de apoyo
I-001	Externo	Apoyo
I-002	Externo	Apoyo
I-003	Externo	Apoyo
I-004	Interno	Apoyo

Cuadro 2.3: Tabla de clasificación de interesados del proyecto

ID	Nombre	Rol desempeñado	Localización	Contacto
I-001	Manuel Lama Penín	Tutor del proyecto	Campus Vida, Santiago de Compostela	manuel.lama@usc.es
I-002	Juán C. Vidal Aguiar	Tutor del proyecto	Campus Vida, Santiago de Compostela	Pendiente
I-003	Victor J. Gallego Fontenla	Tutor del proyecto	Campus Vida, Santiago de Compostela	victor.gallego@usc.es
I-004	Daniel Montero Castelao	Desarrollador	Campus Vida, Santiago de Compostela	daniel.montero@rai.usc.es

Cuadro 2.1: Tabla de identificación de interesados del proyecto

2.7.1. Planificación de las comunicaciones

Con los interesados identificados, evaluados y clasificados se procede a realizar la planificación de las comunicaciones, de modo que en caso de tener que comunicarse con cualquiera de los interesados se haga de la forma más eficiente posible y que dicha comunicación no sea irrelevante.

Las reuniones con los interesados se acuerda en el lugar de trabajo de estos, en el *CiTIUS* situado en el Campus Vida en Santiago de Compostela. Dichas reuniones se realizan en el despacho de alguno de los interesados según se acordado previo aviso.

No se establece ningún intervalo para las reuniones, estas se irán realizando según las necesidades del desarrollador o por petición propia de alguno de los interesados, se considera que estas reuniones serán mucho más frecuentes al inicio del desarrollo del proyecto, en la fase de definición y análisis de requisitos y cuando se completen incrementos o iteraciones de desarrollo.

En el cuadro 2.4 se muestra la matriz de planificación de las comunicaciones, donde se indican los distintos medios para comunicarse con los interesados, idioma utilizado, el tipo de información que los interesados reciben junto con su nivel de detalle.

ID	Objetivo	Nivel de detalle	Tipo de información	Idioma	Formato
I-001	Conocer evolución del proyecto. Resolver dudas	Alto	Técnica, Descriptiva	Castellano, Gallego	Reuniones, Correo electrónico
I-002	Conocer evolución del proyecto. Resolver dudas	Alto	Técnica, Descriptiva	Castellano, Gallego	Reuniones, Correo electrónico
I-003	Conocer evolución del proyecto. Proporcionar requisitos. Resolver dudas.	Alto	Técnica, Descriptiva	Castellano, Gallego	Reuniones, Correo electrónico

Cuadro 2.4: Matriz de planificación de comunicaciones

Capítulo 3

Análisis de requisitos

La fase de análisis de requisitos es una de las más importantes en la ingeniería del software, pues en ella se condiciona enormemente la dirección del proyecto a desarrollar. Su incorrecta realización suele llevar a retrocesos en el proyecto debido a la inconformidad de los interesados, o en el peor de los casos puede llevar a la cancelación del mismo.

Por ello, es de obligado cumplimiento enfocar esta fase lo mejor posible para evitar este tipo de situaciones en el proyecto, para lograr esto, se realizará un análisis de los usuarios que utilizarán nuestra aplicación, se identificarán sus características, intereses, nivel de conocimiento general y factores de rechazo.

En base a los factores identificados en los usuarios que utilizarán la aplicación y a las reuniones con los clientes se extraerán los requisitos funcionales y no funcionales que deberá de cumplir la aplicación.

Dichos requisitos serán priorizados en base a las preferencias de los usuarios y los clientes, de modo que los de mayor prioridad serán los que antes se empiecen a diseñar e implementar.

3.1. Actores del sistema

Es importante, antes de pasar a la fase de especificación de requisitos, identificar los actores que intervendrán en nuestro sistema o aplicación, así como identificar los intereses de cada uno de ellos y aquellas situaciones que harían que rechazaran nuestra aplicación o se sintiesen incómodos utilizándola.

Adicionalmente, es necesario identificar el perfil de conocimiento de cada actor que utilice nuestra aplicación, este factor puede condicionar múltiples requisitos funcionales y no funcionales además de determinar el nivel de usabilidad que se

le conceda a la aplicación, también condiciona las comunicaciones con ellos en caso de que estos actores sean los clientes que encargan el proyecto.

Enfocando el análisis con una base sobre los usuarios que van a utilizar nuestra aplicación incrementa las probabilidades de éxito del proyecto además de disminuir los cambios posibles en los requisitos de la aplicación.

En caso de existir múltiples actores con distintos intereses, es de suma importancia intentar identificar conflictos de intereses entre los actores del sistema (en caso de que existan) que podrían provocar, o bien que los requisitos de la aplicación sufriesen cambios, o bien que apareciesen nuevos requisitos en momentos inesperados del proyecto.

En nuestra aplicación solo existe un actor, el cual llamaremos **Usuario**, por lo tanto no es necesario realizar un análisis de conflictos de intereses y además se podrá centrar el análisis del único actor de una forma más exhaustiva, los siguientes apartados se centrarán exclusivamente en analizar este actor.

3.1.1. Intereses del usuario

Los intereses principales del usuario general de la aplicación son los siguientes:

- Crear y editar plantillas de informes los más rápido y cómodamente posible.
- Reutilizar las plantillas de informes múltiples veces.
- Crear plantillas cuyo modelo de datos representado es fácilmente alterable y editable.
- Introducir variables y fuentes de datos de un modelo de datos en el contenido de la plantilla.
- Generar un informe completo en base a una plantilla de forma cómoda y simple.

Con estos intereses identificados se pueden identificar requisitos no funcionales más fácilmente, además de extraer algunos requisitos funcionales no pedidos explícitamente por el cliente.

3.1.2. Perfil de conocimiento del usuario

De media general, el usuario al que está destinada la aplicación, poseerá conocimientos altos en el sector de las TIC, además su nivel técnico es de nivel experto en cuanto a materias relacionadas con la generación y manipulación de informes de todo tipo.

Puesto que el usuario tiene un perfil alto de conocimientos en las TIC no va a ser necesario aplicar un nivel extremadamente alto de usabilidad.

Adicionalmente, es posible incluir toda clase de tecnicismos relacionados con la materia en la aplicación sin riesgo a que el usuario no los comprenda.

3.1.3. Factores de rechazo del usuario

Los factores que pueden provocar que el usuario rechace la aplicación de forma **absoluta** son los siguientes:

- Generar un informe dura mucho tiempo o directamente no generarlo.
- Editar el modelo de datos que representa la plantilla es extremadamente complejo.
- Falta de componentes que representen información estructurada como tablas o gráficas.
- Usar la aplicación es difícil o tedioso.
- No poder inyectar variables o fuentes de datos en la plantilla.

Con estos factores identificados, se pueden extraer una mayor cantidad de requisitos tanto funcionales como no funcionales y evitar posibles errores de diseño e implementación en el futuro.

3.2. Especificación de requisitos

Con los usuarios analizados se puede proceder a realizar un análisis de los posibles requisitos que debe de cumplir la aplicación, tanto desde un punto de vista funcional como de rendimiento, no funcional o de información.

Los requisitos que se identifiquen tendrán una prioridad, la prioridad dependerá del valor que el usuario o cliente conceda al requisito que se haya extraído.

En esta sección se ha utilizado la siguiente escala:

- **Crítica**: El requisito es indispensable, la aplicación no sirve si no se cumple.
- **Alta**: El requisito tiene un valor considerable para el cliente o usuario, su ausencia puede provocar el rechazo o inconformidad de uso.

- **Media**: El requisito tiene un valor medio para el cliente o usuario, su ausencia puede notarse aunque no implicaría a grandes rasgos una carencia grave.
- **Baja**: El requisito tiene una prioridad baja para el cliente, su ausencia no provoca efectos negativos para el cliente o usuario.

3.2.1. Requisitos funcionales

ID	RF-01
Nombre	Creación de plantillas nuevas
Descripción	La aplicación debe de permitir la creación de plantillas nuevas cuyo contenido por defecto será un folio de tamaño A4 en blanco.
Prioridad	Crítica

ID	RF-02
Nombre	Guardado de plantillas
Descripción	La aplicación debe de permitir el guardado de las plantillas que se hayan generado en la base de datos.
Prioridad	Crítica

ID	RF-03
Nombre	Carga de plantillas desde la base de datos
Descripción	La aplicación debe de permitir al usuario la carga de plantillas previamente creadas.
Prioridad	Crítica

ID	RF-04
Nombre	Borrado de plantillas
Descripción	La aplicación debe de permitir al usuario borrar aquellas plantillas que ya no considere necesarias.
Prioridad	Crítica

ID	RF-05
Nombre	Inserción de componentes de texto en las plantillas
Descripción	La aplicación permitirá al usuario la inserción o añadido de un número indeterminado de elementos de texto (títulos, párrafos y listas) a la plantilla que esté editando.
Prioridad	Crítica

ID	RF-06
Nombre	Inserción de imágenes en las plantillas
Descripción	La aplicación permitirá al usuario la inserción o añadido de un número indeterminado de imágenes cargadas desde el almacenamiento local del usuario en la plantilla que esté editando.
Prioridad	Alta

ID	RF-07
Nombre	Inserción de tablas en las plantillas
Descripción	La aplicación permitirá al usuario la inserción o añadido de un número indeterminado de tablas en la plantilla que esté editando.
Prioridad	Crítica

ID	RF-08
Nombre	Inserción de gráficas de datos en las plantillas
Descripción	La aplicación permitirá al usuario la inserción o añadido de un número indeterminado de gráficas de datos (barras, área, dispersión, sectores, líneas y radar) en la plantilla que esté editando.
Prioridad	Crítica

ID	RF-09
Nombre	Inserción de mallas de celdas en las plantillas
Descripción	La aplicación permitirá al usuario la inserción o añadido de un número indeterminado de mallas de celdas en la plantilla que esté editando.
Prioridad	Alta

ID	RF-10
Nombre	Definición de variables de contenido dinámico
Descripción	La aplicación permitirá al usuario la definición de un número indeterminado de variables a nivel de plantilla.
Prioridad	Crítica

ID	RF-11
Nombre	Mapeo de variables a rutas de JSON
Descripción	La aplicación permitirá al usuario realizar un mapeo de rutas del archivo en formato JSON que haya cargado en la aplicación en las variables que haya declarado.
Prioridad	Crítica

ID	RF-12
Nombre	Carga de archivos JSON a nivel de aplicación
Descripción	La aplicación permitirá al usuario la carga de archivos en formato JSON para su posterior mapeo de rutas en la sección de variables.
Prioridad	Crítica

ID	RF-13
Nombre	Inyección de variables en componentes de texto
Descripción	La aplicación permitirá al usuario la inyección de variables en componentes de texto mediante la sintaxis {{*nombre de la variable*}}.
Prioridad	Crítica

ID	RF-14
Nombre	Carga de fuente de datos en tablas
Descripción	La aplicación permitirá al usuario la carga de datos dinámicos desde una variable declarada, la fuente de datos será seleccionada como una propiedad de la tabla.
Prioridad	Crítica

ID	RF-15
Nombre	Carga de fuente de datos en gráficas
Descripción	La aplicación permitirá al usuario la carga de datos dinámicos desde una variable declarada, la fuente de datos será seleccionada como una propiedad de la gráfica.
Prioridad	Crítica

ID	RF-16
Nombre	Reemplazo y carga de variables
Descripción	La aplicación permitirá a voluntad del usuario reemplazar las variables que ha inyectado en los componentes de texto y cargar las fuentes de datos de las tablas, listas y gráficas que haya declarado en la plantilla.
Prioridad	Crítica

ID	RF-17
Nombre	Impresión a PDF de la plantilla con variables reemplazadas
Descripción	La aplicación permitirá al usuario la opción de generar un archivo en formato PDF a partir de la plantilla que ha generado substituyendo todas la variables y cargando todas las fuentes de datos declaradas por el usuario, el PDF generado tendrá un tamaño por página de un folio DIN A4 y podrá ser descargado por el usuario una vez generado.
Prioridad	Crítica

ID	RF-18
Nombre	Edición del tamaño y desplazamiento de un componente
Descripción	La aplicación permitirá al usuario editar el ancho y el desplazamiento hacia la derecha que ocupan los componentes que haya en la plantilla.
Prioridad	Alta

ID	RF-19
Nombre	Borrado de componentes y filas de componentes
Descripción	La aplicación permitirá al usuario borrar cualquier componente (o una fila entera de ellos) que haya añadido a la plantilla que esté editando.
Prioridad	Crítica

ID	RF-20
Nombre	Opciones de decoración, alineación y enumeración a los componentes de texto
Descripción	La aplicación permitirá al usuario definir la alineación del contenido de los componentes de texto (centrado, izquierda, derecha, justificado) junto con las opciones de decoración básicas (negrita, cursiva y subrayado), en caso de ser una lista se podrá definir si es ordenada o no ordenada.
Prioridad	Alta

ID	RF-21
Nombre	Edición de componentes
Descripción	La aplicación permitirá al usuario la edición del contenido de cualquier componente presente en la plantilla (Cambiar el contenido, opciones de decoración, añadir/quitar filas de datos a las tablas o series de datos a las gráficas).
Prioridad	Crítica

ID	RF-22
Nombre	Añadido de celdas en las mallas
Descripción	La aplicación permitirá al usuario el añadido de un número indeterminado de celdas en un componente de tipo malla.
Prioridad	Crítica

ID	RF-23
Nombre	Edición de propiedades de las celdas de una malla
Descripción	La aplicación permitirá al usuario definir la altura que ocupa una celda, así como su ancho, su posición dentro de la malla, así como su contenido textual, aplicándose las mismas propiedades que un componente de tipo texto, el usuario podrá decidir en cualquier momento si borrar dicha celda de la malla.
Prioridad	Crítica

ID	RF-24
Nombre	Inserción de componentes nuevos mediante *Drag and Drop*
Descripción	La aplicación permitirá al usuario la creación de nuevos componentes utilizando la técnica de *Drag and Drop* para arrastrarlos desde el menú de componentes a la posición deseada en la plantilla.
Prioridad	Crítica

ID	RF-25
Nombre	Alteración de la posición de componentes existentes mediante *Drag and Drop*
Descripción	La aplicación permitirá al usuario alterar la posición de componentes existentes en la plantilla arrastrándolos y soltándolos en cualquier posición disponible en la plantilla.
Prioridad	Crítica

ID	RF-26
Nombre	Creación de versiones de una plantilla
Descripción	La aplicación permitirá al usuario definir versiones para una plantilla determinada, de modo que estas se guardaran con el mismo nombre de plantilla en la base de datos, aunque su contenido será diferente.
Prioridad	Media

ID	RF-27
Nombre	Edición de versiones de una plantilla
Descripción	La aplicación permitirá al usuario cargar una versión que el seleccione de una plantilla determinada.
Prioridad	Media

ID	RF-28
Nombre	Borrado de una versión
Descripción	La aplicación permitirá al usuario borrar una versión determinada de una plantilla que el seleccione.
Prioridad	Media

ID	RF-29
Nombre	Carga de fuentes de datos en listas
Descripción	La aplicación permitirá al usuario la carga de datos dinámicos desde una variable declarada, la fuente de datos será seleccionada como una propiedad de la lista.
Prioridad	Media

ID	RF-30
Nombre	Carga de múltiples archivos JSON en la aplicación
Descripción	La aplicación permitirá al usuario la carga de múltiples archivos en formato JSON, de modo que el usuario podrá alternar entre uno o otro para su uso en la declaración de variables.
Prioridad	Crítica

ID	RF-31
Nombre	Impresión de múltiples informes
Descripción	La aplicación permitirá al usuario la impresión de múltiples informes de forma simultánea ante una plantilla determinada, de modo que la aplicación generará un informe por cada JSON cargado en la aplicación, reemplazando las variables por los valores de cada JSON.
Prioridad	Alta

3.2.2. Requisitos no funcionales

ID	RNF-01
Nombre	La aplicación debe de ser en Web
Descripción	La aplicación debe de ser web para asegurar mayor compatibilidad y portabilidad.
Prioridad	Crítico

ID	RNF-02
Nombre	La plantilla generada debe de guardarse en formato JSON
Descripción	La plantilla que genere el usuario se guardará como un objeto en formato JSON que represente la estructura y contenido de la plantilla.
Prioridad	Crítico

ID	RNF-03
Nombre	Utilización de servicios Web
Descripción	Para implementar el lado servidor que se encargará de proporcionar las funcionalidades de guardado, carga y borrado de plantillas es necesario implementarlas como servicios RESTful para garantizar compatibilidad con la aplicación web.
Prioridad	Crítico

ID	RNF-04
Nombre	Uso de tecnologías web
Descripción	Puesto que la aplicación será en web, se deberán de utilizar las tecnologías de desarrollo web como HTML5, CSS3 y JavaScript.
Prioridad	Crítico

ID	RNF-05
Nombre	Base de datos NoSQL Documental
Descripción	Para almacenar las plantillas en formato JSON se utilizará una base de datos documental.
Prioridad	Crítica

ID	RNF-06
Nombre	Uso de *Node.js*
Descripción	Para imprimir el PDF del informe es necesario utilizar *PhantomJS*, el cual requiere de la utilización de *Node.js* para ejecutarlo.
Prioridad	Crítica

ID	RNF-07
Nombre	El informe enviado al servicio de PDF debe estar en HTML
Descripción	El servicio de generación de PDF debe únicamente recibir un HTML a partir del cual generará el PDF.
Prioridad	Crítica

ID	RNF-08
Nombre	Uso de JSON a nivel interno
Descripción	La aplicación debe de tratar todos sus datos a nivel interno en JSON, por lo tanto en caso de recibir datos desde bases de datos relacionales o CSV estos deberán de ser traducidos a JSON.
Prioridad	Crítica

3.2.3. Requisitos de información

ID	RI-01
Nombre	Variable de texto
Descripción	El sistema debe de almacenar las variables de texto que declare el usuario junto con la configuración que el establezca.
Requisitos asociados	RF-10, RF-11
Datos específicos	- Nombre de la variable - Configuración de la variable - Ruta del JSON
Prioridad	Crítica

ID	RI-02
Nombre	Variable tipo Fuente de datos
Descripción	El sistema debe de almacenar las variables tipo fuente de datos que declare el usuario junto con su asociación a los elementos de una plantilla, además de la configuración personalizada que el establezca para dicha variable.
Requisitos asociados	RF-10, RF-11, RF-14, RF-15
Datos específicos	- Nombre de la variable - Configuración de la variable - Ruta del JSON - Componente(s) asociado(s)
Prioridad	Crítica

ID	RI-03
Nombre	Componente
Descripción	El sistema debe de almacenar la configuración de cada componente que pertenezca a una plantilla determinada.
Requisitos asociados	RF-05, RF-06, RF-07, RF-08, RF-09
Datos específicos	- Identificador del componente - Configuración del componente - Disposición del componente - Componentes hijos
Prioridad	Crítica

ID	RI-04
Nombre	Plantilla
Descripción	El sistema debe de almacenar la plantilla creada por un usuario.
Requisitos asociados	RF-01, RF-02, RF-03, RF-04, RF-26, RF-27, RF-28
Datos específicos	- Nombre de la plantilla - Versión de la plantilla - Componentes de la plantilla - Variables de la plantilla
Prioridad	Crítica

ID	RI-05
Nombre	Informe
Descripción	El sistema debe de almacenar el informe resultante creado por un usuario a partir de una plantilla.
Requisitos asociados	RF-017, RF-031
Datos específicos	- Nombre del informe - Versión del informe
Prioridad	Crítica

3.2.4. Casos de uso

ID	CU-01
Nombre	Creación de un componente
Requisitos asociados	RF-05, RF-06, RF-07, RF-08, RF-09, RF-24
Descripción	El sistema deberá de permitir al usuario la inclusión de cualquier componente definido por el sistema en sus plantillas.
Precondiciones	Existe una plantilla cargada en la aplicación.
Postcondiciones	Se crea el componente que el usuario haya seleccionado en la plantilla con la configuración por defecto.
Pasos	1 - El usuario selecciona un componente cualquiera en el panel de componentes. 2 - El usuario arrastra el componente seleccionado a una fila o columna vacía de la plantilla. 3 - El usuario suelta el componente en la posición que haya seleccionado.
Prioridad	Crítica
Criterio de validación	El usuario puede seleccionar cualquier componente e insertarlo en cualquier posición disponible en la plantilla.

ID	CU-02
Nombre	Modificación de un componente
Requisitos asociados	RF-18, RF-20, RF-21, RF-22, RF-23, RF-25
Descripción	El sistema deberá de permitir al usuario la modificación de las propiedades disponibles de cualquier componente en la plantilla (incluyéndose la posición y tamaño del elemento).
Precondiciones	Existe una plantilla cargada en la aplicación con al menos un componente.
Postcondiciones	Las propiedades del componente se modifican según el criterio especificado por el usuario.
Pasos	1 - El usuario selecciona el componente que quiere editar. 2 - Se despliega el menú de edición de propiedades del componente mostrando sus opciones y parámetros. 3 - El usuario modifica las propiedades del componente (alterando el tamaño o contenido del mismo). 4 - El componente identifica los cambios realizados en su estado y se actualiza para reflejarlos.
Prioridad	Crítica
Criterio de validación	El usuario puede seleccionar y editar cualquier componente presente en la plantilla, además los cambios realizados se aplican al componente de forma correcta.

ID	CU-03
Nombre	Borrado de componentes
Requisitos asociados	RF-19
Descripción	El sistema deberá de permitir al usuario el borrado de cualquier componente presente en la plantilla.
Precondiciones	Existe una plantilla cargada en la aplicación con al menos un componente
Postcondiciones	El componente seleccionado es borrado de la plantilla.
Pasos	1 - El usuario selecciona el componente que quiere borrar. 2 - Se despliega el menú de edición de propiedades del componente mostrando sus opciones y parámetros. 3 - El usuario selecciona la opción de borrar componente. 4 - El componente actualiza su estado y es borrado de la plantilla.
Prioridad	Crítica
Criterio de validación	El usuario puede seleccionar y borrar cualquier componente presente en la plantilla.

ID	CU-04
Nombre	Creación de plantillas
Requisitos asociados	RF-01
Descripción	El sistema deberá de permitir al usuario la creación de plantillas nuevas.
Precondiciones	Ninguna.
Postcondiciones	Se crea la plantilla con los criterios especificados por el usuario.
Pasos	1 -El usuario selecciona la opción de *Nueva* en la barra de acciones. 2 -Se despliega un cuadro que indica los campos que se deben de cubrir para crear la nueva plantilla. 3 - El usuario cubre los campos especificando el nombre y versión de la plantilla y pulsa en la opción de *Crear*. 4 - El sistema crea la plantilla y cambia la vista mostrando una plantilla en blanco.
Prioridad	Crítica
Criterio de validación	El usuario puede crear plantillas nuevas con el nombre y versión que el especifique.

ID	CU-05
Nombre	Guardado de plantillas
Requisitos asociados	RF-02
Descripción	El sistema deberá de permitir al usuario guardar las plantillas que el haya creado, junto con el estado de todos sus componentes.
Precondiciones	Existe una plantilla creada con antelación con algún componente.
Postcondiciones	Se guarda la plantilla en la base de datos con su nuevo estado.
Pasos	1 - El usuario realiza modificaciones a la plantilla actual. 2 - El usuario pulsa en la opción de *Guardar* en la barra de acciones. 3 - La aplicación envía la plantilla al servidor y este la guarda en la base de datos. 4 - El usuario es informado de que la plantilla ha sido guardada con éxito.
Prioridad	Crítica
Criterio de validación	El usuario puede guardar las plantillas a las que haya aplicado cualquier cambio.

ID	CU-06
Nombre	Carga de plantillas
Requisitos asociados	RF-03
Descripción	El sistema deberá de permitir al usuario la carga de cualquier plantilla que haya guardado con antelación en la base de datos.
Precondiciones	Existe una plantilla guardada en la base de datos
Postcondiciones	Se carga la plantilla que el usuario ha seleccionado en la aplicación
Pasos	1 - El usuario selecciona la opción de *Cargar* en la barra de acciones. 2 - Se despliega un menú en el que aparecen todas las versiones de las plantillas disponibles. 3 - El usuario selecciona una plantilla y pulsa en la opción de *Cargar*. 4 - El sistema obtiene la plantilla de la base de datos y cambia la vista para mostrarla.
Prioridad	Crítica
Criterio de validación	El usuario puede cargar las plantillas que haya guardado previamente en la aplicación.

ID	CU-07
Nombre	Borrado de plantillas
Requisitos asociados	RF-04
Descripción	El sistema deberá de permitir al usuario el borrado de cualquier plantilla almacenada en la base de datos
Precondiciones	Existe una plantilla guardada en la base de datos
Postcondiciones	Se borra la plantilla seleccionada por el usuario.
Pasos	1 - El usuario selecciona la opción de *Cargar* en la barra de acciones. 2 - Se despliega el menú en el que aparecen todas las versiones de las plantillas disponibles. 3 - El usuario selecciona una plantilla y pulsa en la opción de *Borrar*. 4 - El sistema borra la plantilla seleccionada de la base de datos
Prioridad	Crítica
Criterio de validación	El usuario puede borrar las plantillas que haya guardado previamente en la aplicación.

ID	CU-08
Nombre	Definición de variables
Requisitos asociados	RF-10, RF-011
Descripción	El sistema deberá de permitir al usuario la definición de variables de contenido dinámico.
Precondiciones	Existe una plantilla cargada en la aplicación
Postcondiciones	Se crea la variable con los criterios del usuario
Pasos	1 - El usuario selecciona la sección de *Variables* en el menú de navegación. 2 - La aplicación cambia a la vista de variables. 3 - El usuario introduce el nombre y el tipo de la variable y pulsa en *Crear*. 4 - El sistema crea la variable y la define en su espacio de nombres. 5 - El usuario define a continuación una ruta de JSON para el contenido de la variable junto con las demás propiedades de la variable.
Prioridad	Crítica
Criterio de validación	El usuario puede crear y modificar variables para una plantilla cualquiera.

ID	CU-09
Nombre	Inyección de variables en componentes
Requisitos asociados	RF-13, RF-14, RF-15, RF-29
Descripción	El sistema deberá de permitir al usuario la inyección de las variables creadas en los componentes de texto la plantilla o el uso de estas como fuente de datos dinámica para las listas, tablas o gráficas
Precondiciones	Existe una plantilla con un componente, además debe de existir una variable con una ruta de JSON definida
Postcondiciones	Se inyecta la variable en el componente
Pasos	1 - El usuario selecciona un componente. 2.a - (Si es un componente de texto) El usuario escribe el nombre de la variable entre {{}}. 2.b - (Si es una tabla, lista o gráfica) El usuario activa la opción *dinámica* y selecciona la variable. 3 - El componete actualiza su estado para reflejar los cambios.
Prioridad	Crítica
Criterio de validación	El usuario puede inyectar cualquier variable definida en cualquier componente dentro de los criterios especificados por el componente.

ID	CU-10
Nombre	Carga y reemplazo de variables
Requisitos asociados	RF-16
Descripción	El sistema deberá de permitir al usuario reemplazar las variables inyectadas por su valor.
Precondiciones	Existe una plantilla con un componente, este componente debe de tener inyectada una variable (como texto o como fuente de datos), la variable debe de estar definida y hay un archivo JSON cargado con datos.
Postcondiciones	Se reemplazan las variables de la plantilla por su valor.
Pasos	1 - El usuario selecciona la opción de *Reemplazar variables*. 2 - La aplicación busca las variables en los componentes y las reemplaza por su valor. 3 - La plantilla actualiza su estado para mostrar el contenido de todas las variables.
Prioridad	Crítica
Criterio de validación	El usuario puede reemplazar las variables definidas en la plantilla.

ID	CU-11
Nombre	Carga de archivos JSON a nivel de aplicación
Requisitos asociados	RF-12, RF-30
Descripción	El sistema deberá de permitir al usuario cargar múltiples archivos JSON en la aplicación para dar valor a las variables que el defina en la plantilla.
Precondiciones	Existe una plantilla creada
Postcondiciones	Se carga el archivo (o los archivos) JSON en la aplicación
Pasos	1 -El usuario selecciona la sección de *Variables* en el menú de navegación. 2 - La aplicación cambia a la vista de variables. 3 - El usuario selecciona la opción de *Cargar archivo JSON*. 4 - El usuario selecciona los archivos JSON de su almacenamiento local que desea cargar en la aplicación. 5 - La aplicación carga los archivos JSON seleccionados. 6 - El usuario selecciona uno de los archivos cargados como archivo actual para dar valor a las variables.
Prioridad	Crítica
Criterio de validación	El usuario pueda cargar un número indeterminado de archivos JSON en la aplicación.

ID	CU-12
Nombre	Impresión a PDF de informes
Requisitos asociados	RF-17, RF-31
Descripción	El sistema deberá de permitir al usuario la generación de un informe por cada archivo JSON cargado en la aplicación, substituyendo las variables definidas por los valores especificados en cada JSON.
Precondiciones	Existe una plantilla con componentes con variables inyectadas además existe un archivo JSON cargado en la aplicación.
Postcondiciones	Se crea un informe con las variables reemplazadas en PDF por cada archivo JSON cargado en la aplicación
Pasos	1 - El usuario selecciona la opción de *Imprimir todos* en la barra de acciones. 2 -La aplicación substituye las variables en la plantilla y pasa el informe a formato PDF (Este paso se repite por cada archivo JSON presente en la aplicación).
Prioridad	Crítica
Criterio de validación	El usuario pueda obtener un informe en formato PDF con las variables reemplazadas por cada archivo JSON cargado en la aplicación.

3.2.5. Matrices de trazabilidad

	OBJ-001	OBJ-002	OBJ-003
CU-01	X		
CU-02	X		
CU-03	X		
CU-04	X	X	X
CU-05	X	X	X
CU-06	X	X	
CU-07	X	X	
CU-08	X		X
CU-09	X		X
CU-10	X		X
CU-11	X		X
CU-12	X	X	X

Cuadro 3.1: Matriz de trazabilidad Caso de uso - Objetivos

	Entregable 1	Entregable 2	Entregable 3	Entregable Final
CU-01	X	X	X	X
CU-02	X	X	X	X
CU-03	X	X	X	X
CU-04			X	X
CU-05			X	X
CU-06			X	X
CU-07			X	X
CU-08		X	X	X
CU-09		X	X	X
CU-10		X	X	X
CU-11		X	X	X
CU-12				X

Cuadro 3.2: Matriz de trazabilidad Caso de uso-Entregables

Capítulo 4

Análisis de tecnologías y herramientas

A la hora de realizar la implementación o diseño de un sistema siempre existen distintas alternativas tecnológicas para hacerlo.

Es importante analizar este tipo de tecnologías o herramientas antes de decidir cual utilizar para desarrollar el sistema, pues eligiendo una o otra produce alteraciones substanciales en las fases de diseño e implementación y pruebas, lo que provoca que estas tengan que ser adaptadas a las decisiones tecnológicas que se realicen en esta fase.

Adicionalmente, destacar que la elección de una tecnología debe tener como criterio principal de selección permitir que el sistema a desarrollar cumpla con los requisitos contemplados en el proyecto, además de facilitar trabajo al desarrollador.

4.1. Servicios web

A la hora de transmitir información entre distintas aplicaciones distribuidas nos encontramos con distintos paradigmas (paso de mensajes, llamada de procedimiento remoto, sockets, etc.), la decisión de cual aplicar depende del contexto de la aplicación y del proyecto a desarrollar.

Uno de los paradigmas más utilizados en desarrollo de aplicaciones web actualmente son los **servicios web**, que consiste en acceder a métodos remotos a través de la web [3], generalmente utilizando HTTP y transmitiendo un mensaje de texto que debe de ser interpretado por el servidor para la realización de la operación que el cliente ha solicitado, adicionalmente los mensajes de texto pueden ser documentos en formato XML o JSON, así como texto plano o tramas de datos, el

formato que se utilice depende del servicio y del proveedor del mismo.

Gracias a esto se garantiza interoperatividad entre distintos lenguajes de programación o implementaciones de tecnologías, ya que todos ellos adoptarán un estándar para transmitir la información y entenderse.

Para este proyecto, el requisito no funcional **RNF-03** impone la restricción de que el paradigma a utilizar sean servicios web para cumplir las funcionalidades de almacenamiento y manipulación de plantillas generadas con la aplicación, por lo tanto el uso de este paradigma de computación distribuida es obligatorio, y aunque no lo fuese es recomendable al tratarse de una aplicación web.

Adicionalmente se establece en ese mismo requisito que los servicios web deben de estar implementados como servicios **RESTful**, por lo tanto otras alternativas como **SOAP** quedan descartadas para implementar los servicios web.

Para implementar una API que contenga los servicios web existen múltiples posibilidades o alternativas, debido a que existe desacoplamiento entre el cliente y el servidor se puede utilizar cualquier lenguaje de programación para hacerlos y actualmente existen múltiples *frameworks* o librerías para cada lenguaje que proporcionan facilidades para implementar servicios REST.

A continuación se muestran las distintas alternativas que se analizaron y cual se optó por utilizar en este proyecto.

4.1.1. Jersey y Java

Jersey [4] es un framework que permite el desarrollo de servicios web RESTful en Java. Permite programar los servicios como funciones Java estándar, las operaciones de Jersey están implementadas siguiendo las especificación de la API JAX-RS [5], por ello Jersey utiliza toda clase de anotaciones para describir como servicios web las funciones Java programadas.

Mediante el uso de la anotación *@Path* se definen las URIs de los servicios, adicionalmente las las anotaciones *@POST*, *@GET*, *@PUT* y *@DELETE* se define que tipo de operación HTTP utilizan los servicios para invocarse.

Se puede definir adicionalmente si el servicio produce o consume información mediante las anotaciones *@Consumes* y *@Produces*.

Actualmente, esta bajo licencia de uso doble **CDDL** versión 1.1 y **GPL** versión 2.

4.1.2. Express.js y JavaScript

Express.js [6] es un framework de desarrollo web para Node.js, permite de forma muy sencilla implementar servicios REST mediante el uso de funciones JavaScript específicas del framework.

Las funciones: express().post(), express().get(), express.put() y express().delete() permiten crear servicios con cada una de la operaciones HTTP permitidas para los servicios REST.

Adicionalmente, se integra perfectamente con otras funcionalidades de Node.js y existen multitud de paquetes y librerías para expandir su abanico de funcionalidades, en concreto, contiene paquetes para trabajar directamente con bases de datos documentales como MongoDB de forma sencilla.

Su licencia de uso es MIT.

4.1.3. Rocket y Rust

Rocket [7] es un framework para el reciente lenguaje **Rust** [8] que está siendo desarrollado por Mozilla, el cual es un lenguaje diseñado para ser seguro, concurrente y eficiente.

Su funcionamiento esta basado en los principios de Jersey, mediante el uso de anotaciones de la forma *#[]* se pueden definir las funciones programadas en Rust como servicios web.

En concreto se pueden utilizar las anotaciones: *#[get()]*,*#[post()]*, *#[put()]* y *#[delete()]* para definir el tipo de operaciones HTTP que se van a utilizar en la función programada en Rust.

Esta bajo licencia MIT y Apache 2.0.

4.1.4. Conclusión

Analizando las tres posibilidades anteriores se ha determinado que el framework y el lenguaje que se va a utilizar para implementar los servicios REST es **Express.js y JavaScript**.

Los motivos de esta decisión son los siguientes:

- Maneja de forma nativa el formato JSON utilizado por los servicios REST.
- Es extremadamente sencillo implementar servicios REST en el.

- Soportado en la mayoría de las plataformas puesto que corre en Node.js.
- Tiene toda clase de paquetes para trabajar con bases de datos NoSQL documentales con suma facilidad (**RNF-05**).
- Hay que usar *Node.js* para imprimir el PDF mediante el uso de *PhantomJS* (**RNF-06**).

4.2. Tecnologías web estándar

Los requisitos **RNF-01** y **RNF-04** establecen que la aplicación de interacción directa con el usuario (*Front end*) debe de estar implementada usando tecnologías web.

Las tecnologías web estándar actuales son : **HTML5**, **CSS3** y **JavaScript**.

A lo largo de este apartado se definen las tres además de su utilidad y funcionalidad en este proyecto.

4.2.1. HTML5

HTML es el lenguaje básico para representar los elementos de una aplicación web. Se trata de un lenguaje desarrollado y estandarizado por **W3C** [1] y **WHATWG** [2].

HTML define una página web como un conjunto de etiquetas o marcas, su anidamiento y orden en el documento definirá el contenido de la página web.

Actualmente HTML se encuentra en su quinta entrega, más conocida como HTML5, esta versión incluye nuevas etiquetas semánticas que facilitan el desarrollo de aplicaciones web, como por ejemplo: *main* para el contenido principal, *footer* para el pie de página, *header* para el encabezado de la página o *nav* para la navegación de la página.

Adicionalmente HTML dispone de nuevas APIs como *canvas* para el pintado de contenido en una página web o *Drag and Drop* para arrastrar y soltar elementos HTML presentes en el documento de la página web.

[1] `https://www.w3.org/`
[2] `https://whatwg.org/`

4.2.2. CSS3

La definición de la apariencia de un elemento definido en el lenguaje HTML se realiza mediante el uso de hojas de estilo en cascada, o más bien conocidas como *Cascading Style Sheets* (**CSS**).

CSS permite mediante la definición de reglas aplicar estilo a los elementos HTML incluidos en una página web, dichas propiedades abarcan desde el color de los elementos, tamaño de fuente, bordes de los elementos y márgenes a posición de los elementos, modo de visualización y tamaño de los mismos entre otras propiedades. Mediante el uso de CSS se programa la visualización de una página web de forma completa por parte del usuario final de la aplicación.

Actualmente la versión de CSS en la que se está trabajando es CSS3 [3], aunque su implementación no es absoluta en todos los navegadores si se ha alcanzado un gran progreso en las funcionalidades que contempla.

4.2.3. JavaScript

JavaScript es el lenguaje de programación de alto nivel utilizado a nivel web para dotar a las páginas web de contenido dinámico. Se trata de un lenguaje de tipado débil y dinámico basado en prototipos, además es multiparadigma ya que incorpora paradigmas de programación imperativa, funcional y orientada a objetos. Fue desarrollado por *Netscape Communications Corporation* y *Mozilla Foundation* [4] y estandarizado por ECMA [5].

JavaScript en conjunto con la API DOM permite acceder a los elementos HTML y CSS de la página web y modificarlos, añadiendo más elementos o eliminándolos si es necesario. De este modo se puede modificar el contenido de la página web de forma dinámica según el usuario va produciendo eventos de interacción, como pulsar un botón o rellenar un formulario.

Actualmente JavaScript está en la versión **ES8**, o conocido también como *ECMAScript 2017* [9], esta versión incorpora funcionalidades de programación adicionales como funciones asíncronas, rellenado de cadenas de texto o obtención de propiedades de objetos, esta versión está implementada actualmente en casi todos los navegadores modernos.

Adicionalmente, ES8 incluye ES6, una de las actualizaciones más importantes de JavaScript, en la que se incluyen lambdas y se refuerza el paradigma de pro-

[3] `https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3`
[4] `https://www.mozilla.org/en-US/foundation/`
[5] `https://www.ecma-international.org/`

gramación funcional del lenguaje.

Debido a su extenso uso existen multitud de librerías y *frameworks* que permiten agilizar el desarrollo en JavaScript de páginas web.

4.2.4. JSON

JSON [10] o *JavaScript Object Notation* es el formato de representación e intercambio de datos utilizado por excelencia en JavaScript.

La representación de un objeto en formato JSON se basa en la utilización de pares {clave: valor}, de modo que es posible representar objetos complejos mediante el uso de un formato sencillo, siendo este formato fácil de comprender por un humano y fácil de traducir para una máquina.

El formato JSON es utilizado en JavaScript para representar los objetos que manipula y crea, sin embargo el uso de JSON se ha propagado a algunas de las bases de datos documentales NoSQL, de modo que almacenan los documentos como un objeto en formato JSON, o en todo caso como un BSON (JSON en binario), y utilizan un sistema de consultas basadas en JSON.

4.3. Frameworks y librerías

Para agilizar el trabajo a realizar o poder realizar ciertas funcionalidades en un proyecto de software se pueden utilizar librerías o frameworks.

Puesto que esta aplicación se sitúa en el contexto web, existen multitud de frameworks y librerías de uso libre que se pueden utilizar para realizar este proyecto.

Debido a su enorme cantidad y a su diversidad no es posible definir todas las alternativas posibles en esta sección, por ello solo se van a mencionar aquellas que se seleccionaron junto a las razones que determinaron la decisión.

4.3.1. TypeScript

TypeScript [11] es un lenguaje de superconjunto de JavaScript desarrollado y mantenido por *Microsoft* y bajo licencia Apache 2.0.

El propósito de TypeScript es introducir tipado estático a JavaScript, de modo que el código programado se vuelve declarativo al introducir tipos de forma estática en el código JavaScript evitando errores comunes de programación en JavaScript.

Puesto que es un lenguaje de superconjunto, la escritura de código TypeScript incluye tanto lenguaje JavaScript como código no-JavaScript, por ello TypeScript utiliza un compilador propio que traduce estas directivas no-JavaScript a código equivalente JavaScript que puede ser interpretado por cualquier navegador.

Adicionalmente, TypeScript cuenta con un conjunto de funcionalidades adicionales como *enums*, interfaces, intersección de tipos, tipos genéricos, *namespaces*, símbolos, mapas, propiedades, etc. Además, se puede integrar con los principales frameworks de desarrollo web, como AngularJS, React, Express.js o Vue.js de forma sencilla.

Este lenguaje se ha seleccionado como tecnología de desarrollo debido a su compatibilidad con JavaScript normal, fácil integración con frameworks de desarrollo web y a las funcionalidades de programación orientada a objetos adicionales que proporciona.

También mencionar que la principal causa de esta elección es el tipado estático que introduce, de modo que se puede desarrollar código declarativo libre de errores en tiempo de compilación.

4.3.2. React

React [12] es una librería de JavaScript declarativa y basada en componentes desarrollada por *Facebook* bajo licencia MIT, que permite desarrollar interfaces de usuario de forma rápida, sencilla y escalable a costa de una curva considerable de aprendizaje para un desarrollador inexperto.

React engloba la visión de una página web como un conjunto de componentes pequeños que interactúan entre sí, cada uno de ellos tiene su propio estado, el cual se altera según el usuario va interactuando con la página web. En base a esto resalta la principal característica de React que es la eficiencia de actualización de los componentes mediante su motor de seguimiento, el cual mantiene un seguimiento de los distintos componentes que existan en la página web y solo actualiza aquellos cuyo estado interno cambia, sin necesidad de refrescar la página entera.

Al ser basado en componentes, React aplica el patrón de diseño *Composite Pattern* [13], en el cual los componentes se van englobando unos dentro de otros formando al final un único componente global, pero abstrayendo al programador de tener que realizar la composición global. Como consecuencia de lo anterior, es fácil escalar la aplicación y reutilizar componentes en distintas partes de la aplicación sin necesidad de volver a programarlos.

La principal elección de esta librería en comparación a otras alternativas como Angular.js o Vue.js es su eficiencia y rendimiento a la hora de que el usuario interactúe directamente con la página, adicionalmente al ser una librería se puede ampliar su funcionalidad integrándola con otras existentes en caso de que exista alguna carencia de funcionalidad.

4.3.3. Ant-Design

Ant-Design [14] es un framework de CSS para React y TypeScript desarrollado por la comunidad [6] y bajo licencia MIT.

Proporciona una serie de componentes de React ya creados para utilizar en todo tipo de contexto, los componentes abarcan desde botones a listas pasando por tablas, iconos y notificaciones.

También cuenta con un sistema propio de *layout* y *grid* que permite situar cualquier componente de forma cómoda en la pantalla al mismo tiempo que se mantiene la responsividad de la página.

Es fácilmente editable, por lo que pueden alterare las reglas de CSS que incluya el framework para adaptarlas a las necesidades de la aplicación que se esté desarrollando además de poder integrarle cualquier otro componente con suma facilidad.

Este framework se ha elegido porque es ideal para la situación actual de tecnologías de este proyecto, puesto que está pensado para ser utilizado con React y Typescript, adicionalmente es fácilmente editable y ampliable e incluye un conjunto amplio de todo tipo de componentes.

4.3.4. Recharts.js

Reachts.js [15] es una librería de componentes de React creada por *Recharts Team* bajo licencia MIT que permite la representación de un amplio rango de gráficas.

Esta basado en componentes de React por lo que puede crearse una gráfica de forma escalable y reutilizable, es fácilmente personalizable y presenta un renderizado eficiente y portable al representar las gráficas como una imagen de tipo SVG.

Incluye gráficas cartesianas como gráficas de lineas, barras, áreas y dispersión, además de permitir combinarlas para crear gráficas mixtas. También permite re-

[6] `https://github.com/ant-design/ant-design/blob/master/AUTHORS.txt`

presentar gráficas polares como las gráficas de sectores y las de radar.

La elección de esta librería viene dada por su sencillez de uso, extensibilidad, adaptabilidad y amplio rango de gráficas, además está pensada para ser utilizada con React.

4.3.5. Material icons

Material [16] es un framework de CSS desarrollado y mantenido por *Google* bajo licencia Apache 2.0.

Material tiene un amplio rango de iconos que pueden ser utilizados en cualquier aplicación web, su rango de iconos incluye aplicaciones para el móvil, aplicaciones de tipo GPS y aplicaciones de edición.

Puesto que su rango de iconos para aplicaciones de edición es grande y este proyecto tiene como intención crear un editor web basado en una SPA se ha decidido utilizar los iconos de Material para desarrollar la aplicación.

4.4. Base de datos documental

Basándonos en la referencia [17], una base de datos documental es aquella que almacena la información siguiendo un paradigma de orientación a documentos, en el que se almacena la información de la base de datos en documentos y no en tablas, como en las bases de datos relacionales tradicionales, es por esto que se consideran las bases de datos documentales como de tipo NoSQL (*Not Only SQL*) o No Solo SQL puesto que no garantizan transacciones **ACID** *(Atomicity, Consistency, Isolation, Durability)* como las bases de datos SQL tradicionales.

Las bases de datos NoSQL priorizan escalar mejor horizontalmente y en el uso de APIs orientadas a objetos para modificar el valor de los datos almacenados permitiendo de este modo mayor comodidad para escalar el contenido de las mismas a una situación de una arquitectura de *Big Data* en la que se generan grandes volúmenes de datos a diario [18].

El requisito **RNF-05** establece que la base de datos que se debe de utilizar para almacenar las plantillas debe de ser una base de datos documental, adicionalmente el requisito **RNF-08** indica que el formato de uso interno de la aplicación debe de ser JSON, por lo tanto es necesario utilizar una base de datos documental que utilice a nivel interno el formato JSON o BSON para almacenar los datos, para de este modo facilitar la integración de la aplicación con la base de datos.

4.4.1. MongoDB y Mongoose

MongoDB [19] es una base de datos documental NoSQL que almacena los documentos mediante una representación en formato BSON, su uso es extendido y popular entre las bases de datos NoSQL documentales y permite la realización de consultas en formato JSON para seleccionar datos de documentos. Adicionalmente, existen múltiples APIs para acceder a MongoDB desde distintos lenguajes como Java, C++, Python y JavaScript (*Node.js*).

Para acceder a MongoDB desde JavaScript se cuenta con un módulo programado para *Express.js*, conocido como *Mongoose* [20], que proporciona una API sencilla y fácilmente escalable para acceder y manipular datos de una base de datos MongoDB mediante el uso de esquemas.

Se ha decidido utilizar esta base de datos debido a su uso extendido, alta escalabilidad y a la disponibilidad de APIs sencillas para utilizar con *Node.js* y *Express.js*.

4.5. Herramientas

A continuación se detallan las herramientas adicionales que se han utilizado para desarrollar el proyecto.

4.5.1. NPM

NPM (*Node Package Manager*) [21] es un gestor de paquetes (o módulos) para el lenguaje JavaScript y el entorno de servidor *Node.js*.

Proporciona una interfaz de comandos para instalar, actualizar y borrar paquetes de *Node.js* de forma sencilla, los paquetes se actualizan y descargan desde el registro público de NPM.

Cuando se añaden paquetes o dependencias a un proyecto de JavaScript, NPM crea un archivo llamado *package.json* que contiene un árbol de dependencias entre módulos escrito en formato JSON.

Este archivo es el utilizado por NPM para descargar las dependencias o paquetes que no se encuentren en el proyecto, este proceso es similar al que realizan otros gestores de paquetes, como Maven con el archivo *pom.xml*.

Esta herramienta gestionará las dependencias de los módulos que se utilicen en el proyecto, puesto que el desarrollo del proyecto será completamente en JavaScript, tanto en lado servidor como cliente, el uso de este gestor es recomendable

para mantener actualizados los paquetes que se utilicen en ambas fronteras de desarrollo.

4.5.2. Docker

Docker [22] es una herramienta que proporciona virtualización de imágenes mediante el uso de contenedores.

Los contenedores de Docker contienen de forma íntegra software, como bases de datos, sistemas operativos o intérpretes de lado servidor, las imágenes de los contenedores se descargan desde un repositorio público de modo que es posible instalar en cuestión de minutos una imagen de un sistema operativo o una base de datos sin necesidad de pasar por procesos de instalación o virtualización como con las máquinas virtuales tradicionales.

Adicionalmente, Docker permite de forma sencilla instalar, crear, modificar y borrar contenedores sin una carga excesiva para el sistema operativo y eliminando todo rastro de la virtualización.

Docker será utilizado para almacenar en un contenedor la base de datos MongoDB, de modo que no será necesario instalarla en la capa nativa del sistema operativo.

4.5.3. PhantomJS

PhantomJS [23] es un navegador ligero sin interfaz gráfica de usuario ejecutado sobre el intérprete de JavaScript Node.js, que puede ser invocado mediante comandos para que realice acciones de navegación o administración de páginas web de forma automática mediante el paso de archivos de script escritos en JavaScript.

También incluye herramientas para imprimir la página web como un archivo PDF, PNG o JPG, para el caso del archivo PDF permite definir las dimensiones de la página, los márgenes, cabeceras del archivo y números de página.

Para Node.js existe el paquete *phantom* [24] que permite invocarlo y operar con el desde JavaScript mediante el uso de una API rápida y sencilla.

El uso de PhantomJS está destinado a imprimir el informe generado desde la aplicación a un archivo PDF.

4.5.4. Git

Git [7] es una herramienta de control de versiones de código abierto creada por Linus Torvalds y la comunidad.

Mediante el uso de repositorios y la linea de comandos se puede hacer un seguimiento de las distintas versiones del código que se generen durante el proyecto, de modo que es posible volver a una versión anterior del código generado en cualquier momento. Adicionalmente proporciona herramientas para trabajar en colaborativo, como la creación de ramas (*branches*) y unión de ramas (*merge*).

Git será utilizado para mantener un control de versiones del código que se genere durante el proyecto, proporcionando de este modo soporte para la gestión de la configuración del proyecto, de modo que en caso de que alguna versión del proyecto no sea válida se pueda volver a una versión anterior.

4.5.5. WebStorm

WebStorm [8] es un entorno de programación para JavaScript creado por *JetBrains*.

Cuenta con toda clase de opciones de autocompletado avanzado, optimización de código y pruebas. Adicionalmente cuenta con soporte para TypeScript y los principales frameworks y librerías de desarrollo web como React o AngularJS, lo cual lo hace el entorno de desarrollo ideal para este proyecto.

Actualmente, WebStorm solo se puede obtener bajo una licencia de pago, pero *JetBrains* cuenta con una oferta especial para estudiantes que permite obtener sus productos durante un año de forma gratuita y legal.

4.5.6. TeXStudio

TeXStudio [9] es un entorno integrado para crear documentos en LaTeX, proporciona toda clase de asistentes para facilitar la creación del documento tales como creación de tablas o ecuaciones, así como opciones de autocompletado o resaltado de errores de código o ortográficos.

Sumando esto junto a su baja curva de aprendizaje esta herramienta es ideal para crear la presente memoria del proyecto.

[7] `https://git-scm.com/`
[8] `https://www.jetbrains.com/webstorm/`
[9] `https://www.texstudio.org/`

Capítulo 5

Diseño e implementación del sistema

La fase de diseño constituye el siguiente eslabón en un proyecto informático, pues en esta fase es cuando se representan desde una visión de alto nivel los requisitos que se han recogido durante la fase de análisis.

La representación de estos requisitos pasa por distintos niveles de abstracción, de este modo se generan diagramas orientados a desarrollo, como los diagramas de clases, o diagramas orientados a ser enseñados al cliente, como los diagramas de arquitectura de alto nivel o los bocetos de las distintas vistas de la aplicación.

Sea cual sea el nivel de abstracción que se utilice la intención de los diagramas es dar una visión de alto nivel de como estará implementado el sistema o de como se representarán los requisitos que se hayan recogido.

Adicionalmente, los diagramas orientados a desarrollo también sirven para documentar el funcionamiento y disposición estructural del sistema, sirviendo de este modo como parte de la documentación para realizar futuras ampliaciones, modificaciones, pruebas o arreglo de fallos en el sistema que se represente en ellos.

A lo largo de esta sección se irán describiendo diagramas con distintos niveles de abstracción, incluyendo: arquitectura, clases, interacción e interfaz de usuario.

5.1. Diseño de la arquitectura del sistema

5.1.1. Esquema de la arquitectura del sistema

Antes de pasar a un nivel de detalle puramente técnico es necesario definir de forma esquemática como estará estructurado el sistema en su conjunto, el

esquema de la arquitectura completa del sistema se puede observar en la imagen 5.1

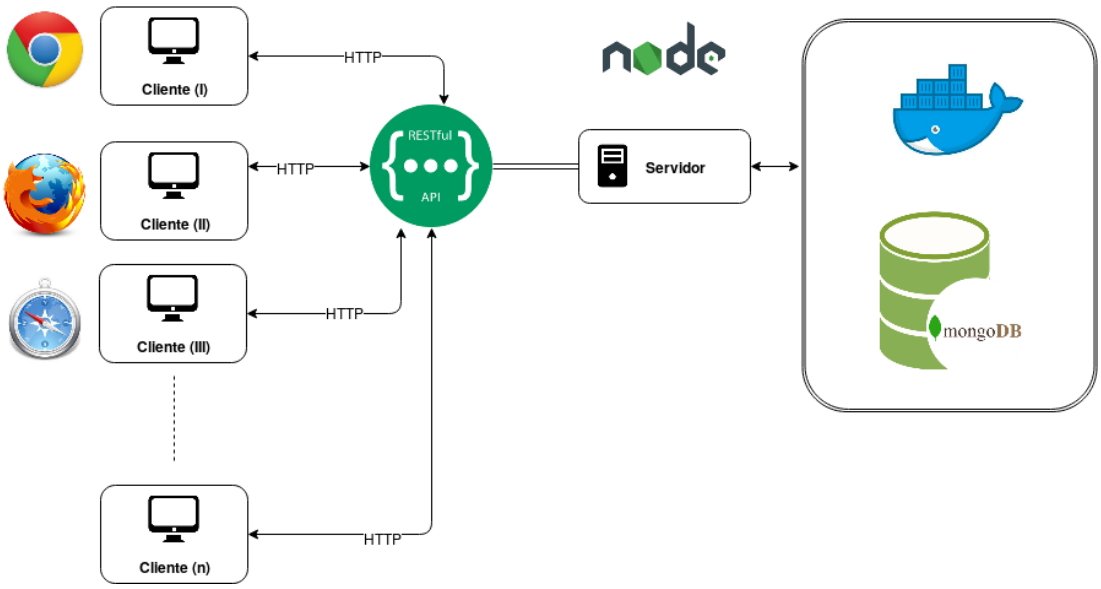

Figura 5.1: Esquema de la arquitectura del sistema

El sistema en su conjunto es una aplicación independiente de la plataforma ejecutada sobre un navegador web cualquiera (Google Chrome, Firefox, Safari...), el cliente web contendrá la interfaz web con que el usuario interactúa para la creación de plantillas e informes. Mediante esta opción se permite la conexión de un número indeterminado de clientes al servidor.

Ciertas acciones del cliente requerirán apoyo del servidor, estas acciones se invocan utilizando el protocolo de comunicación HTTP para invocar las acciones de una API REST que el servidor expone.

El servidor estará ejecutándose en una máquina independiente del cliente y estará programado en NodeJS, como se ha dicho antes el servidor expondrá una API REST con los servicios web de manipulación de plantillas o impresión de informes.

Finalmente el servidor se comunicará con un contenedor de Docker de una base de datos MongoDB para almacenar, extraer o manipular cualquier dato que los clientes necesiten.

5.1.2. Arquitectura global

En el nivel más alto de abstracción tenemos la arquitectura completa del sistema, este diseño permite obtener una visión rápida de los componentes principales

que forman el sistema sin necesidad de entrar en detalle de como está construido cada uno de ellos a nivel de implementación.

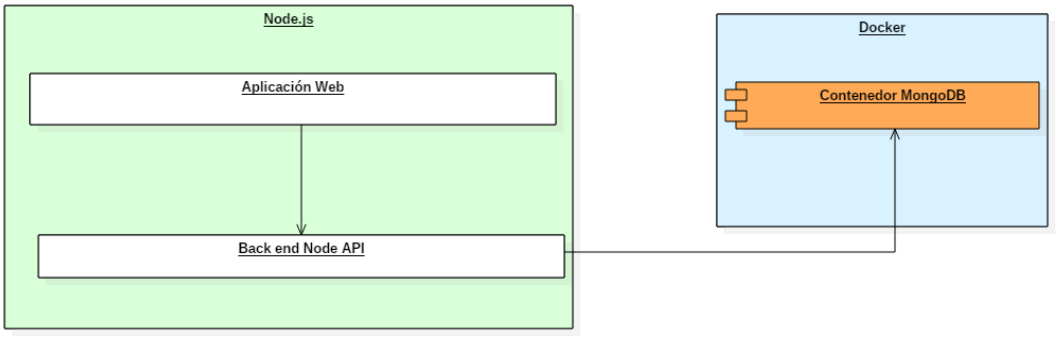

Figura 5.2: Arquitectura de alto nivel del sistema

En la figura 5.2 se muestra un diagrama de arquitectura de alto nivel, en el que se muestran los componentes de mayor alto nivel que forman la aplicación.

El requisito **RNF-01** establece que *la aplicación debe de ser Web*, por lo tanto uno de los componentes primigenios que formarán la aplicación será la interfaz de usuario, que estará programada como una aplicación web.

Adicionalmente el requisito **RNF-03** establece que es necesario la *utilización de servicios web* para implementar el lado servidor, por lo tanto se puede extraer que otro componente primigenio que formará la aplicación será el *Back end* (Capa de servicios) que estará programado como una API en *Node.js*.

La API del servidor expondrá servicios web que consumirá la aplicación web para cumplir con las funcionalidades de manipulación de plantillas que han sido contempladas en la fase de requisitos, como guardado, carga o borrado, adicionalmente proporcionará soporte para realizar la impresión de informes en formato PDF, tal como establece el requisito **RNF-06** que indica que el PDF se debe generar mediante el uso de *PhantomJS*.

En base a lo anterior y al requisito **RNF-05**, que indica que es necesario el *uso de una base de datos NoSQL documental*, se extrae el último componente primigenio de la aplicación, la *base de datos MongoDB*, esta base de datos será la utilizada por el servidor para guardar y extraer datos que la aplicación web demande con llamadas a servicios web.

Adicionalmente se indica que el entorno en el que se ejecutan tanto el servidor, como la aplicación web es *Node.js*, mientras que la base de datos MongoDB

estará contenida en un contenedor de Docker para facilitar la movilidad de la base de datos.

Visto los componentes primigenios, o de más alto nivel, se procede a analizar cada uno de ellos bajando el nivel de abstracción, detallando su composición interna en mayor detalle y las dependencias externas o internas que presenten en su composición.

5.1.3. Arquitectura del cliente web

Como se puede observar en la figura 5.3 que viene a continuación, la arquitectura del cliente web se compone de un paquete principal, *Layout Principal*, en este paquete se integran paquetes más pequeños y específicos que componen en su conjunto la aplicación.

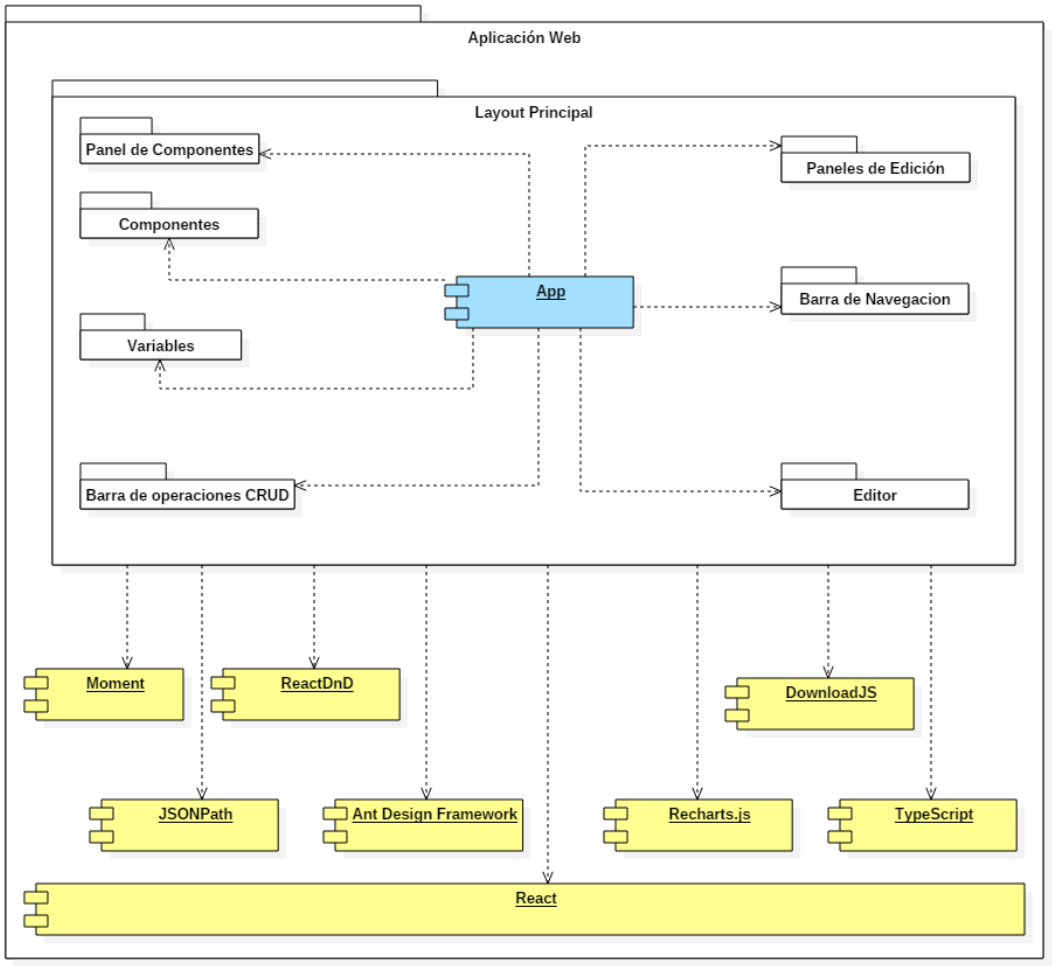

Figura 5.3: Arquitectura de la aplicación web

Los paquetes que integran el *Layout Principal* son:

- **Panel de componentes**: Contiene el panel de los componentes que se pueden añadir a la plantilla.
- **Componentes**: Contiene aquellos componentes funcionales que se añaden y modifican en la plantilla.
- **Variables**: Contiene todo lo relativo a la sección de variables y fuentes de datos que se cargan en el informe.
- **Barra de operaciones CRUD**: Contiene todo lo relativo a las acciones de creación, guardado, carga y borrado de plantillas.
- **Paneles de edición**: Contiene paneles de edición asociados a cada tipo de componente que se añada a la plantilla.
- **Barra de navegación**: Contiene las distintas secciones de la página.
- **Editor**: Contiene el layout donde está situada la plantilla que se está editando, contendrá todas las operaciones lógicas para organizar los componentes que se sitúen.

Todos estos componentes de menor tamaño se integran mediante composición en el componente principal *App*, de modo que al final quedará un único componente que integre todos los demás.

Adicionalmente, el paquete *Layout Principal* depende de librerías o componentes adicionales para realizar sus funcionalidades, dichas librerías o componentes están indicados en el diagrama mediante el color amarillo, los componentes externos son los siguientes:

- **React**: Librería de React, la aplicación basa su funcionamiento en esta librería.
- **Moment**: Librería utilizada para formatear fechas como cadenas de texto.
- **ReactDnD**: Librería para implementar *Drag and Drop* en React.
- **JSONPath**: Librería para hacer consultas de rutas a objetos en formato JSON.
- **Ant Desing Framework**: Framework de CSS para React, la aplicación utiliza sus componentes para dar funcionalidad a la interfaz.
- **Reacharts.js**: Librería para representación de gráficas.

- **DownloadJS**: Librería para ejecutar descargas de archivos en el lado cliente.
- **TypeScript**: Compilador de TypeScript, traduce el código escrito en TypeScript a código JavaScript ejecutable por el navegador.

Los componentes anteriores en conjunto con el paquete *Layout Principal* permiten el funcionamiento integral de la aplicación, siendo de este modo necesarios para que esta pueda funcionar de forma correcta.

5.1.4. Arquitectura del servidor

En la figura 5.4 situada a continuación se muestra la arquitectura del servidor, en su conjunto el servidor está formado por un único módulo, el *Módulo de servicios web*, que incluye todos los servicios web que la aplicación puede invocar para realizar sus funcionalidades.

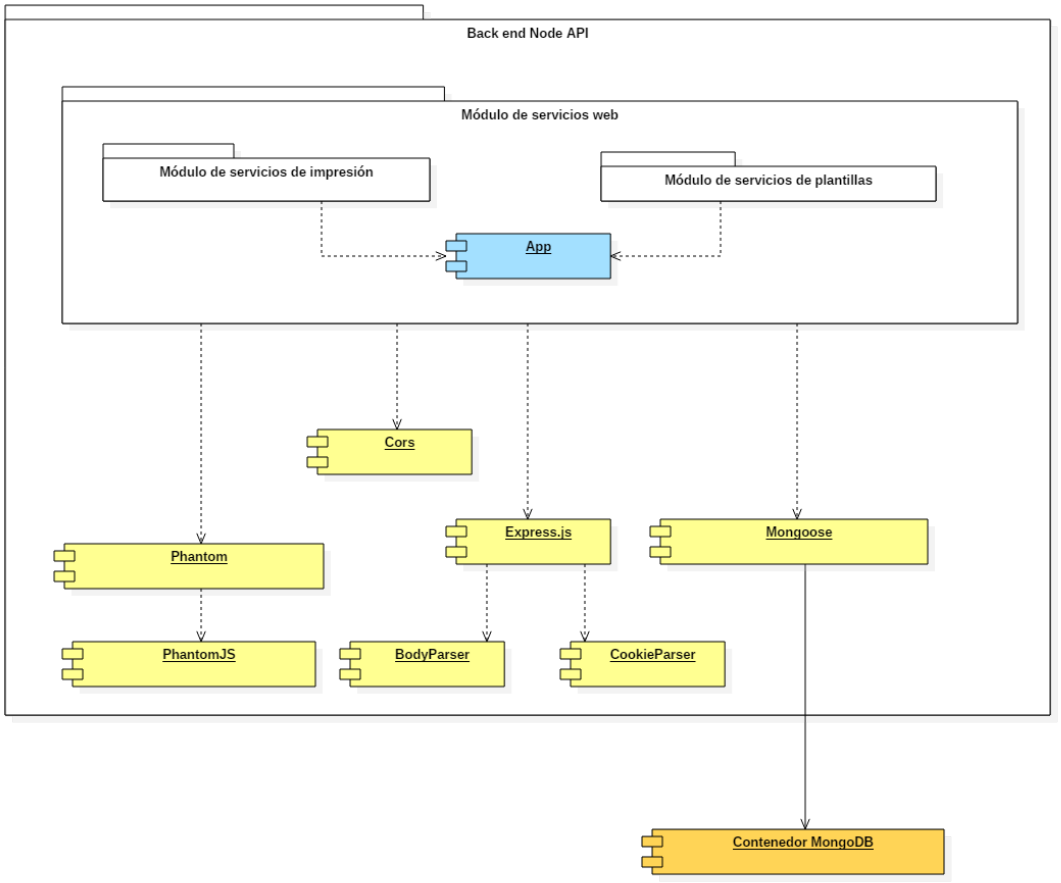

Figura 5.4: Arquitectura del servidor

El *Módulo de servicios web* a su vez está dividido en dos módulos, el *Módulo de servicios de impresión* y el *Módulo de servicios de plantillas*.

El *Módulo de impresión* contiene todos los servicios relativos a la impresión de archivos en formato PDF, de modo que será el encargado de realizar las llamadas a PhantomJS.

El *Módulo de servicios de plantillas* incluye los servicios para crear, obtener, modificar y borrar plantillas, el cliente hará uso de este módulo para guardar las plantillas, modificarlas o borrarlas, de modo que este módulo será el que interactúe con las base de datos MongoDB mediante el uso de la API de *Mongoose*.

Ambos módulos serán integrados mediante composición en el componente principal *App*, que básicamente será un enrutador de servicios, pues en el se colgarán distintos *endpoints* de servicios, de modo que será fácil añadir más servicios al lado servidor en caso de ser necesario.

Las dependencias del *Módulo de servicios web* son las siguientes:

- **Express.js**: Librería de Express.js, la API basa su funcionamiento en este framework, proporciona toda clase de funcionalidades y objetos para crear servicios.
- **Cors**: Librería que permite el uso de CORS (**Cross Origin Resource Sharing**), de modo que el servidor puede realizar llamadas a otros orígenes distintos para obtener recursos.
- **Mongoose**: Librería que proporciona operaciones para trabajar con la base de datos MongoDB desde JavaScript.
- **Phantom**: Librería que expone una API de uso sencillo para trabajar con PhantomJS desde JavaScript.
- **PhantomJS**: Navegador sin interfaz gráfica que puede ser ejecutado por el servidor para tareas de mantenimiento, impresión o navegación rápida.
- **BodyParser**: Usado por Express.js para traducir el cuerpo de peticiones HTTP.
- **CookieParser**: Usado por Express.js para traducir cookies de peticiones HTTP.

Las librerías anteriores en conjunto con los dos módulos de servicios forman el servidor al que el cliente hará peticiones mediante el uso de servicios web para invocar operaciones de manipulación de plantillas o impresión de informes.

5.2. Patrones arquitectónicos

En esta sección se indican los patrones arquitectónicos utilizados a la hora de desarrollar el sistema.

5.2.1. Arquitectura orientada a servicios

El sistema estará basada en una arquitectura orientada a servicios, esto implica que se dividirá la parte cliente (front end) de la parte servidor (back end), el servidor exportará un conjunto de funcionalidades como servicios web que el cliente utilizará como apoyo para proporcionar una mejor experiencia al usuario.

Esta arquitectura permite desacoplar el cliente del servidor, de modo que se aumenta la escalabilidad y mantenimiento del mismo, además permite utilizar lenguajes de programación distintos tanto en el cliente como en el servidor al utilizar el protocolo HTTP para realizar la comunicación entre ambas partes.

5.2.2. Flux

Flux es una alternativa al modelo **MVC** [1] tradicional utilizado en desarrollo web, este patrón arquitectónico es utilizado por React para estructurar los componentes de la aplicación y el modo en el que ellos interactúan.

Flux se basa en una restricción, y es que el e flujo de datos de la aplicación solo puede circular en una dirección, esto se logra mediante la definición de tres componentes principales: *store*, *view component* y *dispatcher*.

Cada *store* contiene determinados datos que representan la lógica de negocio de la aplicación, estos datos son representados por uno o varios *view component*, adicionalmente un *view component* puede extraer datos de varios *stores*, si un *view component* utiliza un store cualquiera se suscribe a el.

Cuando se ejecuta una acción en un *view component* que implica la modificación de los datos almacenados en uno o varios *stores*, este realiza una llamada al *dispatcher*, (solo existe un *dispatcher* en la aplicación) el dispatcher identifica la acción y modifica el o los *stores* afectados por el cambio.

Al cambiar un *store*, los *view component* que estaban suscritos a dichos *stores* sea actualizan para reflejar dichos cambios.

[1] **Modelo Vista Controlador**: Patrón arquitectónico que divide la aplicación en Modelos de datos que contienen la lógica de negocio, vistas que representan dichos datos y controladores que definen que vista se representa.

Esto permite que no existan complicaciones a la hora de escalar la aplicación ya que hace muy predecible el flujo de datos de la misma.

5.3. Modelo de datos

En esta sección se especifica la información que se almacena en la base de datos que utiliza el sistema, puesto que esta base de datos es NoSQL, el modelo de datos no estará constituido de tablas y de relaciones entre dichas tablas, si no de documentos, puesto que la base de datos elegida es MongoDB.

La información que se almacena en cada documento es un objeto *Plantilla*, dicho objeto está compuesto por los siguientes campos:

- **Nombre**: El nombre que el usuario ha dado a la plantilla (Tipo *string*).
- **Versión**: La versión que el usuario ha seleccionado para la plantilla (Tipo *string*).
- **Variables**: Conjunto de variables que forman parte de la plantilla (Tipo *Array<object>*)
- **Componentes**: Conjunto de componentes que forman la plantilla (Tipo *Array<object>*)

Las variables de una plantilla son un conjunto de objetos ordenados en un *array*, cada objeto del array representa una variable y sus campos varían dependiendo de si la variable es de tipo texto o si es una fuente de datos, los campos posibles que tiene una variable son los siguientes:

- **Nombre**: Nombre que el usuario ha dado a la variable (Tipo *string*)
- **Tipo**: Tipo de variable (Tipo *string* controlado por enumerado)
- **Ruta JSON**: Ruta de mapeo de JSON (Tipo *string* o *undefined*)
- **Propiedad de etiquetas (Columnas)**: Ruta del JSON para obtener las etiquetas de las columnas (Tipo *string* o *undefined*) [Solo en variable TABLE_DATA_SOURCE]
- **Unidad del eje X**: Define la unidad del eje X (Tipo *string* o *undefined*) [Solo en variable SCATTER_DATA_SOURCE]
- **Unidad del eje Y**: Define la unidad del eje Y (Tipo *string* o *undefined*) [Solo en variable SCATTER_DATA_SOURCE]
- **Nombre del eje X**: Define el nombre del eje X (Tipo *string* o *undefined*) [Solo en variable SCATTER_DATA_SOURCE]

- **Nombre del eje Y**: Define el nombre del eje Y (Tipo *string* o *undefined*) [Solo en variable SCATTER_DATA_SOURCE]
- **Opacidad de series**: Define la opacidad de cada serie (Tipo *Array<number>* o *undefined*) [Solo en variables POLAR_DATA_SOURCE]
- **Color de la series**: Define el color de cada serie (TIpo *Array<string>* o *undefined*) [Solo en variables CARTESIAN_DATA_SOURCE, POLAR_DATA_SOURCE y SCATTER_DATA_SOURCE]

Los distintos tipos de variables están controladas por el siguiente enumerado:

- **STRING**: La variable es una cadena de texto
- **NUMBER**: La variable es un valor numérico
- **DATE**: La variable es una fecha en formato HH:MM (Hora y minuto)
- **DATE_DD_MM_Y**: La variable es una fecha en formato DD/MM/AAAA (Día, mes, año)
- **DATE_MM_DD_Y**: La variable es una fecha en formato MM/DD/AAAA (Mes, día, año)
- **LIST_DATA_SOURCE**: La variable es una fuente de datos de una lista
- **TABLE_DATA_SOURCE**: La variable es una fuente de datos de una tabla
- **CARTESIAN_DATA_SOURCE**: La variable es una fuente de datos de una gráfica cartesiana
- **POLAR_DATA_SOURCE**: La variable es una fuente de datos de una gráfica polar
- **SCATTER_DATA_SOURCE**: La variable es una fuente de datos de una gráfica de dispersión

Los componentes de una plantilla están ordenados en un *array* de objetos, el orden en el que estén situados los componentes dictamina su orden de aparición en la plantilla, adicionalmente cada componente puede tener dentro otros componentes, esto se puede aplicar de forma recursiva.

Los campos de un componente de la plantilla son los siguientes:

- **Tipo**: Tipo de componente (Tipo *string* controlado por enumerado)
- **Padre**: Componente padre (Tipo *object* o *undefined*)

- **Hijos**: Componentes hijos (Tipo *Array<object>* o *undefined*)
- **Estado**: Estado del componente (Tipo *object*)

Los distintos tipos de componentes están controlados por el siguiente enumerado:

- **ROW**: Componente fila
- **COL**: Componente columna
- **TITLE**: Componente título
- **PARAGRAPH**: Componente parágrafo
- **TABLE**: Componente tabla
- **IMAGE**: Componente image
- **GRID**: Componente malla
- **LIST**: Componente lista
- **CARTESIAN_CHART**: Componente gráfica cartesiana
- **POLAR_CHART**: Componente gráfica polar
- **SCATTER_CHART**: Componente gráfica de dispersión

5.4. Diseño de clases

Los diagramas de clases ofrecen un nivel de abstracción menor que los diagramas de arquitectura al representar la estructuración y comunicación de los elementos de la aplicación que se encuentren descritos en el diagrama.

Es por ello que se suelen orientar más a los desarrolladores de la aplicación o a personas con mayor nivel técnico, también tienden a modificarse numerosas veces según la fase de desarrollo progresa, pues los detalles de implementación suelen variar el diseño de clases inicial.

Los diagramas de clases que se representan a continuación se asocian con los módulos contemplados en los diagramas de arquitectura anteriores.

5.4.1. Diagramas de paquetes

A continuación se detalla la estructuración de las clases a nivel de paquete tanto para el lado servidor como para el lado cliente.

Paquetes del cliente web

En la imagen 5.5 se muestran los distintos paquetes en los que se estructuran las clases de la aplicación web.

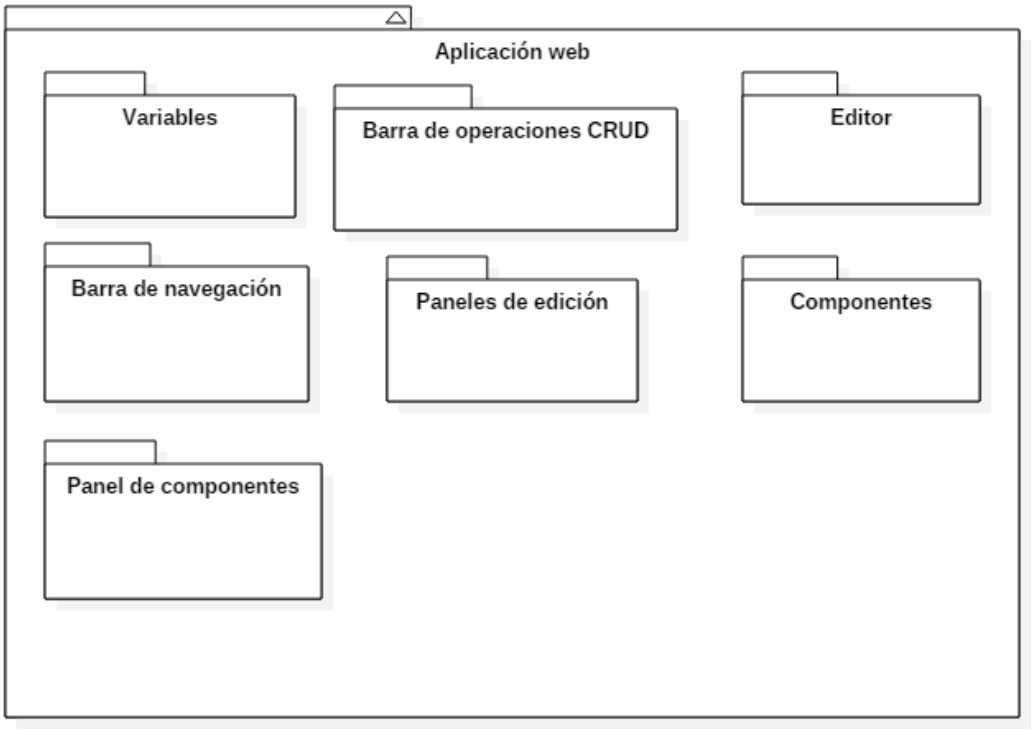

Figura 5.5: Diagrama de paquetes del cliente web

Los paquetes son los siguientes:

- **Variables**: Contiene las clases relativas a la sección de variables, incluyendo la creación, modificación y borrado de estas junto con la carga de JSON.
- **Barra de operaciones CRUD**: Contiene la clases que definen el comportamiento de la barra de acciones CRUD de la aplicación.
- **Editor**: Las clases que definen el comportamiento de la parte central de la aplicación web (esto es, donde se visualiza la plantilla) se incluyen en este paquete.
- **Barra de navegación**: Las clases que dan funcionalidad a la barra de navegación de la aplicación se sitúan en este paquete.
- **Paneles de edición**: Paquete que contiene todas las clases que definen paneles de edición a nivel de componente.

- **Componentes**: Contiene aquellas clases que definen un componente que se inserta en una plantilla.

- **Panel de componentes**: Contiene la clases que definen el comportamiento del panel de creación de componentes.

Paquetes del servidor

En la imagen 5.6 se muestran los distintos paquetes en los que se estructuran las clases del servidor.

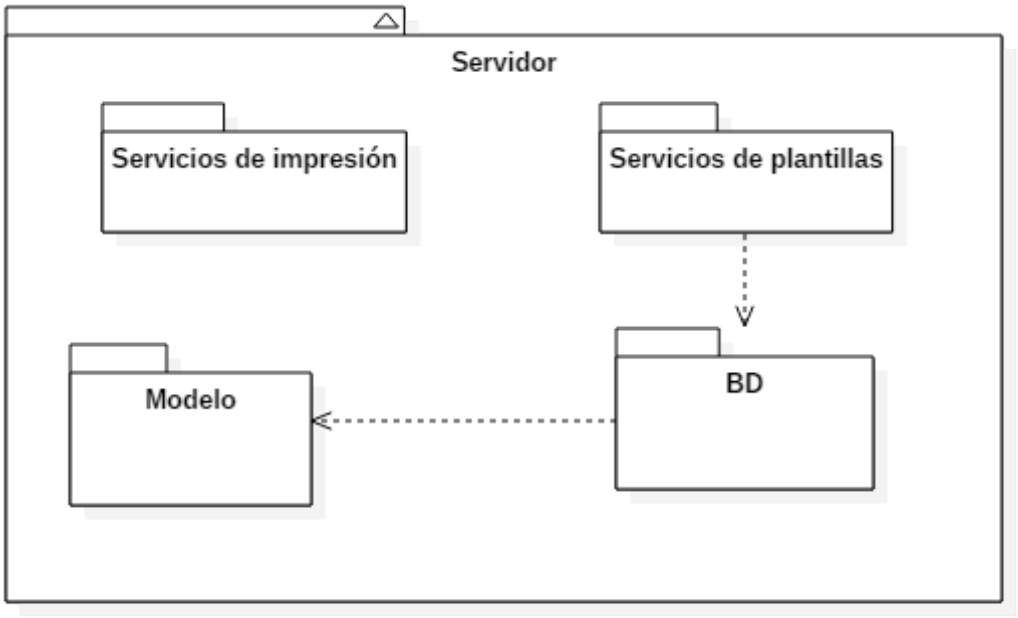

Figura 5.6: Diagrama de paquetes del servidor

Los paquetes son los siguientes:

- **Servicios de impresión**: Contiene las clases que definen la API REST para imprimir informes a formato PDF.

- **Servicios de plantillas**: Contiene las clases que definen la API REST para crear, modificar, eliminar, guardar y obtener plantillas.

- **BD**: Contiene las clases que operan con el driver de la base de datos MongoDB, el paquete *Servicios Plantillas* depende de este paquete para poder ejecutarse.

- **Modelo**: Contiene las clases que definen el modelo de la base de datos para facilitar el cometido del driver, el paquete *BD* necesita este paquete para poder ejecutarse.

5.4.2. Diseño de clases del cliente web

Es importante destacar que cada clase que se presente en los diagramas del cliente web hereda, directa o indirectamente, de la clase *React.Component*, esta clase es proporcionada por la librería de React y contiene una serie de métodos y atributos que permiten que sea manipulada por el motor de renderizado de React.

Además, cada componente puede implementar opcionalmente una interfaz para el estado (*state*) del componente y otras interfaz para sus propiedades (*props*). Las propiedades son inmutables y se utilizan como parámetros del constructor del componente mientras que el estado es mutable y privado a nivel de componente.

El motor de React renderiza los componentes principalmente ante dos situaciones, cuando el estado del componente se modifica o cuando se construye el componente por primera vez.

En los siguientes diagramas se muestran clases abstractas con el estereotipo *<<interface>>*, dichas clases son las interfaces que se utilizan para determinar el estado o las propiedades de un componente de React, de modo que aquellas interfaces que sigan el patrón *NombreComponente**Props*** serán las utilizadas para definir las propiedades mientras que aquellas que sigan el patrón *NombreComponente**State*** serán las utilizadas para definir el estado de un componente.

Mencionar que aquellas clases que no tengan una interfaz que defina su estado son componentes sin estado, o *stateless*, mientras que aquellas clases que no tengan interfaz que defina sus propiedades son componentes con constructor vacío.

Como último apunte, el enfoque seguido es una combinación entre programación orientada a objetos, programación declarativa y programación funcional.

Como consecuencia cuando tenemos una relación de componente hijo a componente padre, el componente hijo recibe como parámetros del constructor referencias a funciones privadas del componente padre, estas funciones serán invocadas por el componente hijo cuando este estime oportuno.

En caso de tener una relación de componente padre a componente hijo, el componente padre puede guardar la referencia al componente hijo e invocar sus métodos públicos o volver a construirla pasando parámetros distintos al constructor.

Núcleo de la aplicación

En la figura 5.7 situada a continuación se especifica el diagrama de clases del núcleo de la aplicación, o lo que es lo mismo, el componente *App* que compone todos los módulos.

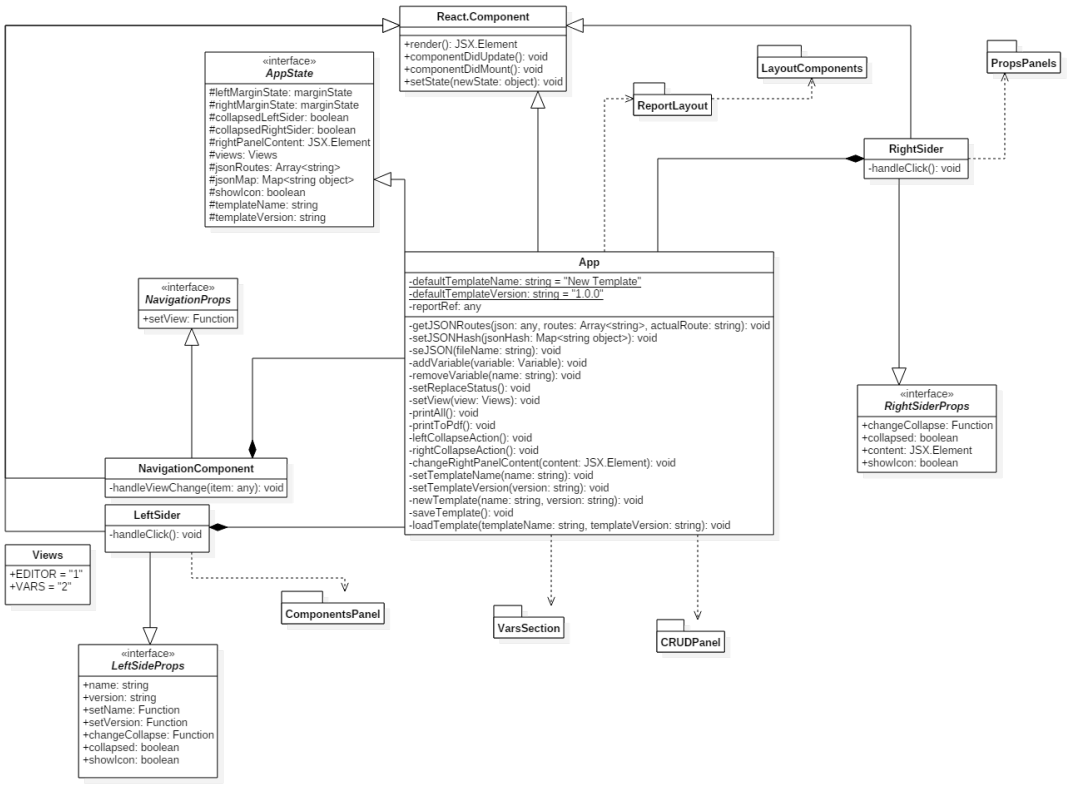

Figura 5.7: Diagrama de clases del núcleo de la aplicación

El componente *App* está compuesto de otros componentes que se corresponden con distintas partes del *Layout* de la aplicación, en concreto está compuesto de los siguientes componentes o paquetes:

- **NavigationComponent**: Componente que contiene la barra de navegación superior de la aplicación, además de todos los submenús de cada sección, por ello todas aquellas acciones que impliquen un cambio de vista son invocadas desde este componente.

- **LeftSider**: Componente que se corresponde con el menú plegable izquierdo de la aplicación, principalmente contendrá el panel de componentes que se pueden arrastrar a la parte central de la aplicación.

- **RightSider**: Componente que se corresponde con el menú plegable derecho de la aplicación, contiene todos los paneles de edición de componentes, su

contenido se modifica de forma bidireccional, de modo que las acciones realizadas desde este componente modifican el contenido de la parte central, además las acciones que se realicen desde la parte central modifican el contenido de este componente.

- **ReportLayout**: Paquete de componentes que se corresponde con la parte central de la aplicación, será donde los componentes que se generen desde el panel izquierdo se muestren, además la edición aplicada desde el panel derecho será renderizada en esta parte de la aplicación.

- **VarsSection**: Paquete de componentes que componen la sección de edición de variables y fuentes de datos, contiene la carga de JSON, creación de variables y asignación de rutas o parámetros adicionales a las variables.

- **CRUDPanel**: Paquete de componentes que componen la barra de creación, carga, guardado, borrado e impresión de plantillas.

El componente *App* es el núcleo de la aplicación, puesto que es necesario pasar por el para transmitir información desde un extremo de la aplicación a otro, por ejemplo desde el panel izquierdo (*LeftSider*) a la parte central (*ReportLayout*), por ello está clase cuenta con una amplia cantidad de métodos para realizar llamadas a cada componente principal que la integra que son pasados como parámetros a los componentes que la forman.

Paquete *ComponentsPanel*

El paquete *ComponentsPanel* es un panel plegable que se inserta en el menú izquierdo de la aplicación, en este panel se representan visualmente mediante iconos aquellos componentes que se pueden arrastrar a la parte central de la aplicación, en la figura 5.8 se puede observar las clases que forman este paquete de componentes.

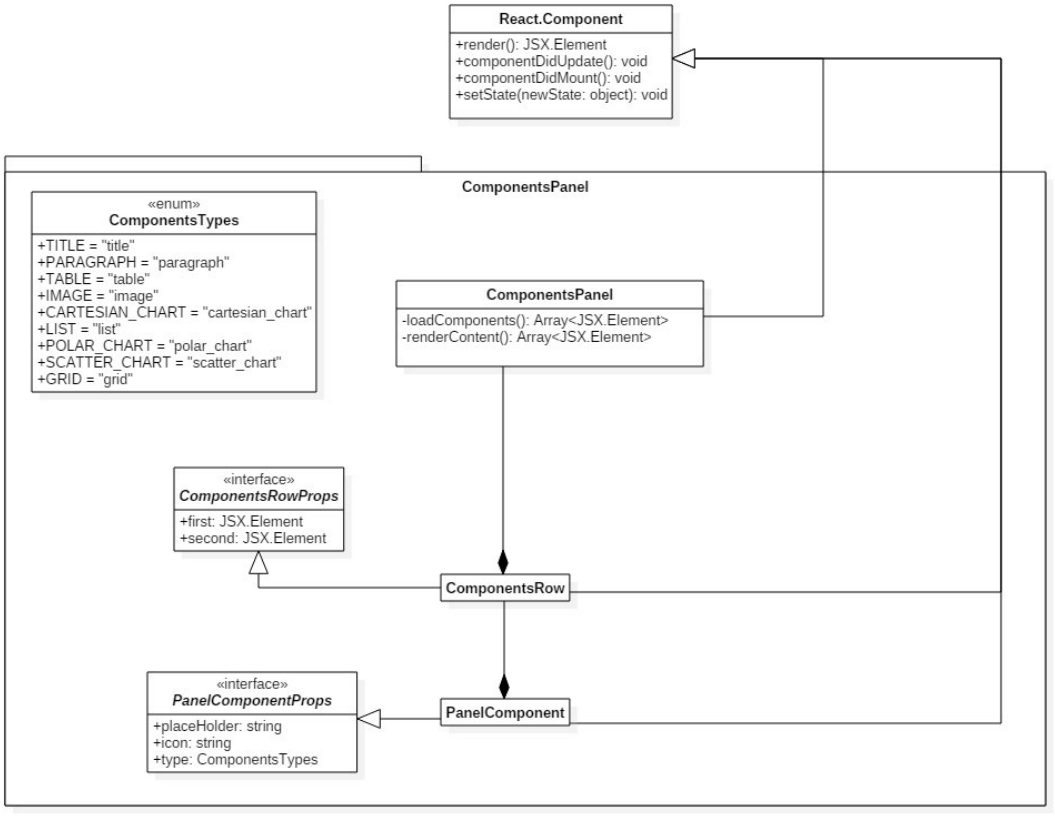

Figura 5.8: Diagrama de clases del paquete *ComponentsPanel*

El componente *ComponentsPanel* está compuesto de la clase *ComponentsRow* y esta a su vez por un *PanelComponent*, el motivo de esta disposición es proporcionar una manera escalable y sencilla de añadir componentes y al mismo tiempo formatear su visualización en filas de dos elementos.

Por ello, la clase *PanelComponent* contendrá un tipo de la clase *ComponentsTypes*, este tipo será el utilizado para saber que componente queremos crear cuando hagamos *Drag and Drop* en la parte central de la aplicación.

Los componentes *PanelComponent* se insertan de dos en dos en un *ComponentsRow* y estos a su vez se insertan en el componente principal *ComponentsPanel* que será la parte exportada por el paquete para su inserción en el menú izquierdo de la aplicación.

Paquete *CRUDPanel*

El paquete *CRUDPanel* es aquel destinado a contener todos los componentes que proporcionan apoyo a las acciones de creación, guardado, carga y borrado de

plantillas, además de las impresión de informes.

En la figura 5.9 se aprecia la presencia de la clase *CRUDComponent*, este componente es el que representará la barra de operaciones CRUD e impresión de plantillas, por ese motivo en su interfaz se incluyen funciones para convocar estas operaciones desde el componente *App*.

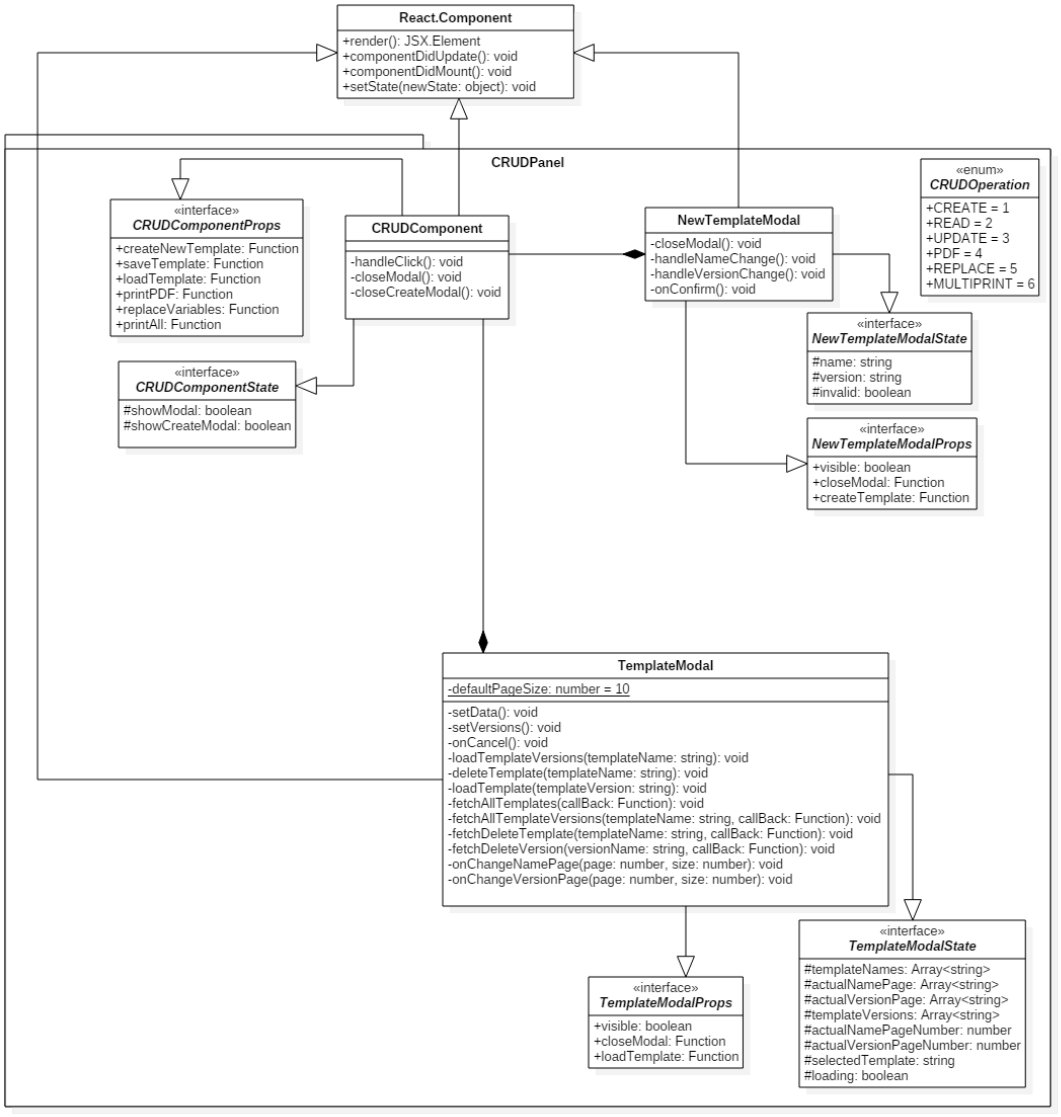

Figura 5.9: Diagrama de clases del paquete *CRUDPanel*

A su vez, la clase *CRUDComponent* está compuesta de otras dos clases, *TemplateModal* y *NewTemplateModal*, estas clases representan *Modals* que se muestran en la pantalla cuando el usuario ejecuta acciones en la barra de operaciones

CRUD.

La clase *NewTempalteModal* se corresponde con el Modal que se muestra cuando el usuario ejecuta la acción de crear una plantilla nueva.

La clase *TemplateModal* se corresponde con la lista de pantillas guardadas en la base de datos que el usuario puede cargar o borrar cuando ejecuta la acción de cargar plantilla existente.

Paquete *LayoutComponents*

El paquete *LayoutComponents* contiene los clases de los componentes que se compondrán un informe, este paquete es el más complejo y más grande de toda la aplicación puesto que incluye toda clase de casuísticas específicas para cada componente imposibilitando de modo extraer jerarquías estables de clases.

En la figura 5.10 y 5.11 se muestra la jerarquía más elemental que se puede extraer de todos los componentes, la clase *AbstractComponent* hereda de *React.Component* y proporciona un método abstracto llamado *setEditPanelContent*, el cual será utilizado por cada componente para crear su panel de edición en el menú derecho de la aplicación.

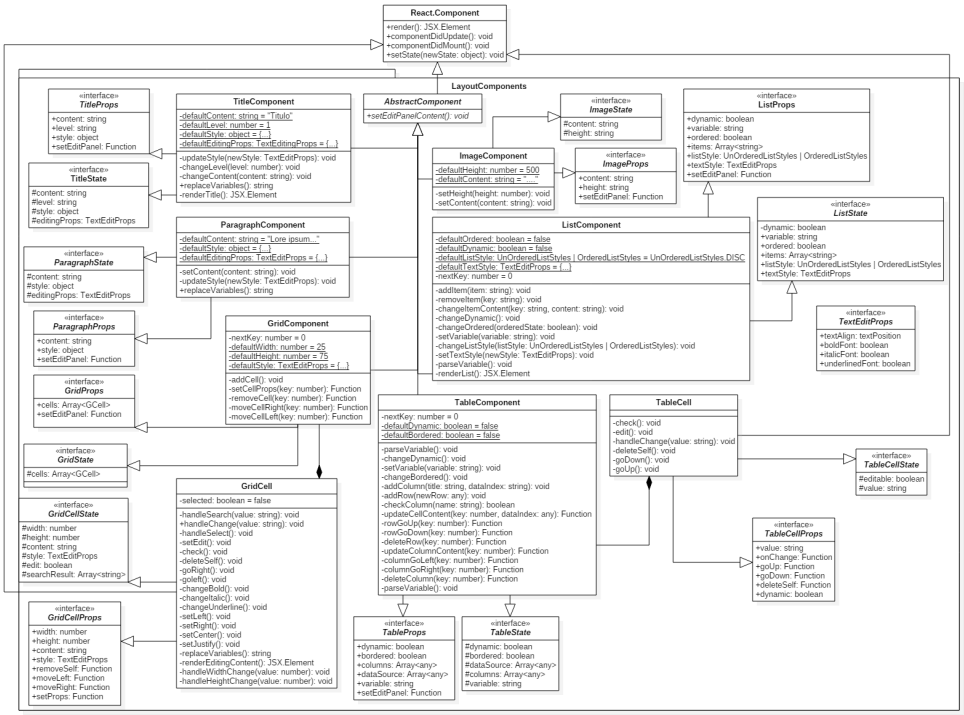

Figura 5.10: Diagrama de clases del paquete *LayoutComponents* (1)

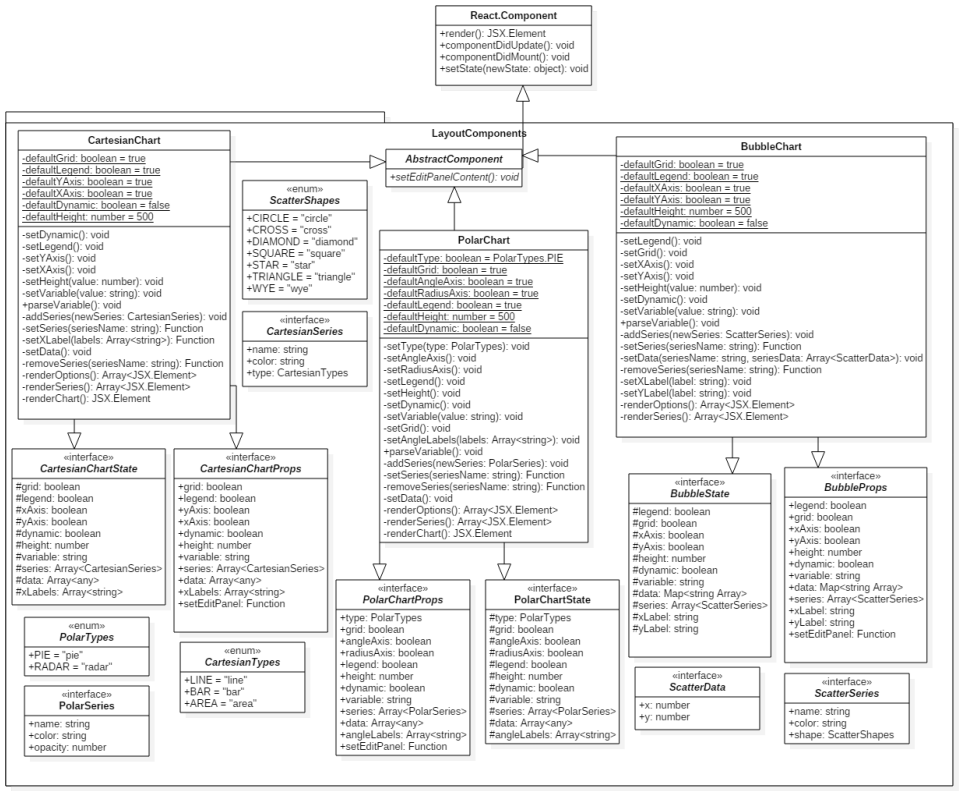

Figura 5.11: Diagrama de clases del paquete *LayoutComponents* (2)

Posteriormente cada componente implementa dos interfaces, una para el estado y otras para sus propiedades, parte de los atributos de las propiedades de un componente son comunes a los atributos de su estado, esto es debido a que las propiedades no se pueden mutar y el estado si, por lo tanto para crear un componente existente es necesario pasar todas las propiedades que permitan reconstruir su estado original.

Los componentes principales que integran este paquete son los siguientes:

- **TitleComponent**: Componente que representa un título o encabezado de documento, incluye opciones para editar su contenido, posición del texto, tipo de encabezado según las etiquetas de HTML H1 a H6, y opciones de fuente como negrita, subrayado y cursiva.
- **ParagraphComponent**: Componente que representa un párrafo, incluye opciones para editar su contenido, posición de texto y opciones de fuente como negrita, subrayado o cursiva.
- **ImageComponent**: Componente que representa una imagen en el documento, incluye opciones para subir una imagen desde almacenamiento local

a la aplicación, modificar la altura y modificar el ancho de la imagen.

- **GridComponent**: Componente que representa una malla de celdas no ordenadas, solo incluye opciones para añadir celdas (*GridCell*).
- **GridCell**: Componente que representa una celda de una malla de celdas no ordenadas, incluye opciones para editar el contenido de la celda, definir la posición del texto de la celda o opciones de fuente como negrita, subrayado y cursiva. Adicionalmente cuenta con opciones para modificar el tamaño, tanto alto como ancho de la celda además de cambiar el orden en el que se encuentra dicha celda.
- **ListComponent**: Componente que representa una lista de elementos, incluye opciones para añadir, quitar y modificar elementos de la lista a voluntad, además permite editar la posición del texto y cuenta con opciones de fuente como negrita, subrayado y cursiva. También cuenta con la opción de definir si la lista es ordenada o no ordenada, habilitando distintos tipos de encabezados para cada elemento de la lista. Finalmente cuenta con la opción de definir la lista como dinámica para generar su contenido desde una fuente de datos cargada desde JSON.
- **TableComponent**: Componente que representa una tabla de datos, incluye opciones para añadir, modificar y borrar columnas o filas, permite cambiar el orden de las filas y columnas a voluntad. Este componente incluye la opción de definirse como dinámico, de modo que su contenido se carga desde una fuente de datos.
- **CartesianChart**: Componente que representa una gráfica cartesiana mixta, de modo que permite la representación combinada de series como barras, áreas y lineas. Este componente permite añadir, modificar y borrar series de datos, adicionalmente cuenta con opciones adicionales como activar o desactivar la leyenda, ejes de coordenadas y la malla cartesiana. Cuenta con la opción de definirse como gráfica dinámica para cargar las series de datos desde una fuente de datos.
- **PolarChart**: Componente que representa una gráfica polar, permite alterar entre dos tipos de gráficas, la gráfica de sectores y la gráfica de radar o *spider-web*. Incluye opciones para añadir, modificar y borrar series de datos además de opciones adicionales para activar o desactivar la leyenda, malla polar y muestra del radio de la gráfica. Cuenta con la opción de definirse como gráfica dinámica para cargar series de datos desde una fuente de datos.
- **BubbleChart**: Componente que representa una gráfica de dispersión en dos variables, permite la creación, modificación y borrado de series de datos, incluye opciones para activar o desactivar la leyenda, malla cartesiana o ejes

de coordenadas. Cuenta con la opción de definirse como gráfica dinámica para cargar series de datos desde una fuente de datos.

Las acciones que modifican un componente se ejecutan desde su correspondiente panel de edición, situado en el paquete *PropsPanels*.

Paquete *PropsPanels*

El paquete *PropsPanels* contiene los componentes que representan los paneles de edición de cada tipo de componente que se pueda representar en la parte central de la aplicación.

En la figura 5.12 y 5.13 se muestran las distintas clases que conforman este paquete de componentes, como se puede observar existe un panel (una clase), por cada componente que se puede representar en la parte central de la aplicación.

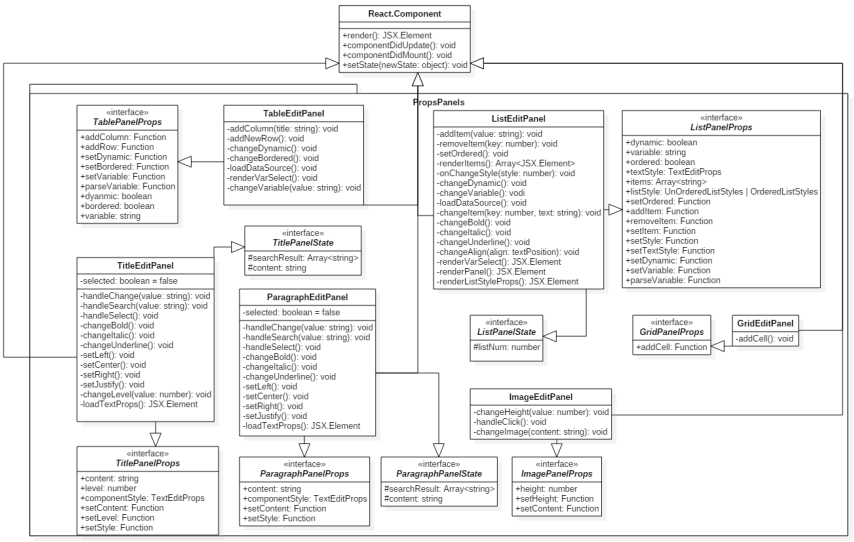

Figura 5.12: Diagrama de clases del paquete *PropsPanels* (1)

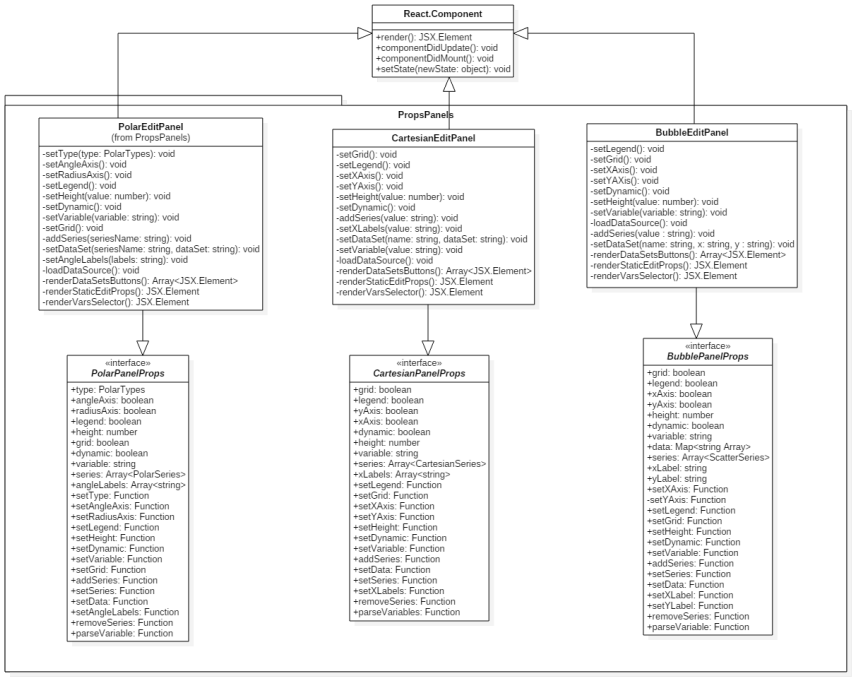

Figura 5.13: Diagrama de clases del paquete *PropsPanels* (2)

En resumen los componentes que forman este paquete son los siguientes:

- **TitleEditPanel**: Componente que representa el panel de edición de un título, contiene todas las acciones y llamadas a funciones para modificar el estado del componente título con el que está vinculado, incluyéndose las opciones de texto o contenido del mismo.

- **ParagraphEditPanel**: Componente que representa el panel de edición de un párrafo, contiene todas las acciones y llamadas a funciones para modificar el estado del componente párrafo con el que está vinculado, incluyéndose las opciones de texto o contenido del mismo.

- **ImagenEditPanel**: Componente que representa el panel de edición de una imagen, contiene las opciones para modificar el alto y ancho de la imagen además de subir una imagen desde almacenamiento local, todo esto mediante realización de llamadas a funciones del componente imagen con el que está vinculado.

- **GridEditPanel**: Componente que representa un panel de edición de una malla de celdas, contiene opciones para añadir más celdas a la malla.

- **ListEditPanel**: Componente que representa un panel de edición de una lista de elementos, contiene opciones para añadir, borrar y editar los elementos

de la lista vinculada, también permite definir el tipo de lista (ordenada o no ordenada) además de editar las propiedades de texto de los elementos y el encabezado de cada elemento de la lista, además permite definir la lista como dinámica seleccionando la fuente de datos desde la que se obtendrán los datos que representará la lista.

- **TableEditPanel**: Componente que representa un panel de edición de una tabla, contiene opciones para añadir filas y columnas nuevas, permite definir la tabla vinculada como dinámica y seleccionar la fuente de datos que se cargará en ella.

- **CartesianEditPanel**: Componente que representa un panel de edición de una gráfica cartesiana, contiene opciones para añadir y borrar series de datos, además de permitir editar su contenido y color, permite además definir la gráfica vinculada como dinámica y seleccionar la fuente de datos que se cargará en ella.

- **PolarEditPanel**: Componente que representa un panel de edición de una gráfica polar, contiene opciones para añadir y borrar series de datos, además de permitir editar su contenido, color y opacidad, permite además seleccionar el tipo de gráfica polar (sectores o radar) y definir la gráfica vinculada como dinámica seleccionado la fuente de datos que cargará los datos en ella.

- **BubbleEditPanel**: Componente que representa un panel de edición de una gráfica de dispersión, contiene opciones para añadir y borrar series de datos, además de permitir editar su contenido, color y forma de representación de los puntos, también permite definir la gráfica vinculada como dinámica seleccionado la fuente de datos desde la que se cargarán los datos.

Paquete *VarsSection*

El paquete *VarsSection* contiene los componentes que representan las sección de definición y manipulación de variables y fuentes de datos de la aplicación.

En la figura 5.14 se muestran las clases que forman el paquete, la construcción de esta sección se basa en la utilización del componente *VarsLayout* como núcleo de la sección, de modo que las distintas partes que componen esta sección se integran en el como un componente más ejecutando cada una sus funcionalidades de forma autónoma pero pasando siempre por el componente núcleo para realizar modificaciones en las otras partes de la aplicación.

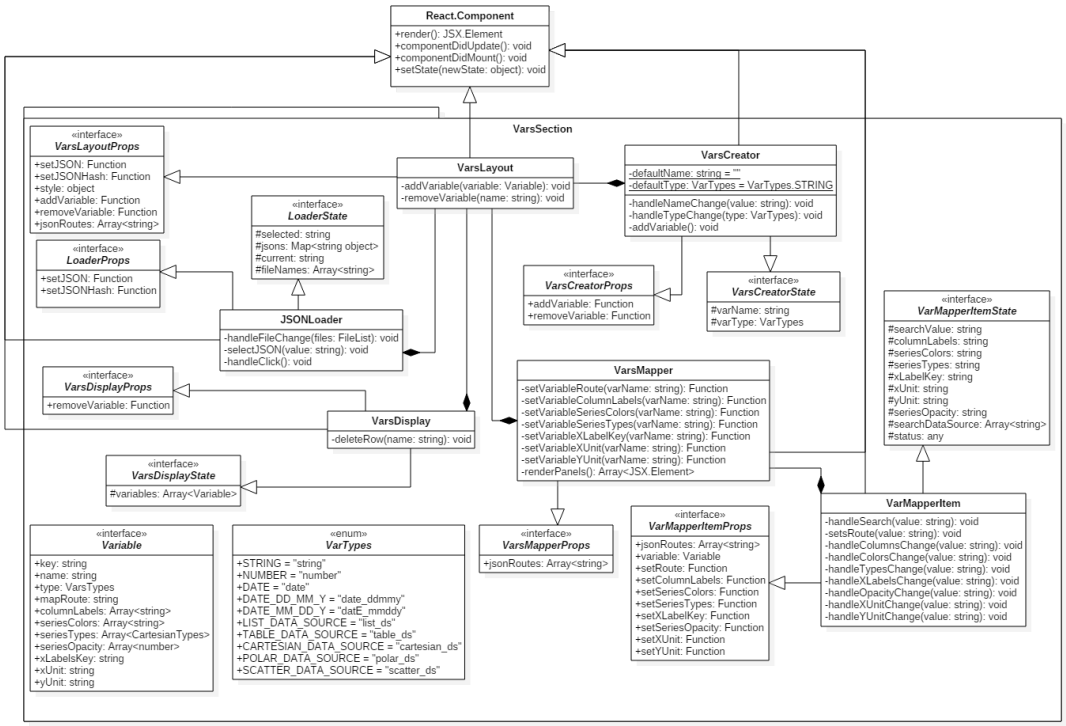

Figura 5.14: Diagrama de clases del paquete *VarsSection*

Este paquete reemplazará la sección central de la aplicación cuando el usuario cambie de sección, de modo que los paneles de edición y creación de componentes se ocultarán hasta que el usuario decida volver a la vista del editor.

Las clases que forman este paquete son las siguientes:

- **VarsLayout**: Componente núcleo de la sección, en ella se integran todos los demás componentes que forman la sección.
- **JSONLoader**: Componente que gestiona la carga de JSON en la aplicación, también gestiona que JSON se está utilizando en ese momento, en caso de que el usuario realice una carga múltiple de JSONs en la aplicación.
- **VarsDisplay**: Componente que muestra las variables ya declaradas en la aplicación, permitiendo eliminar cualquiera que ya no sea necesaria.
- **VarsCreator**: Componente que se encarga de crear nuevas variables a petición del usuario, permite definir el nombre y el tipo de la variable que se va a crear.
- **VarsMapper**: Componente que se encarga de realizar el mapeado de una ruta de JSON a una variable, además de permitir definir opciones adicionales dependiendo del tipo de variable que se esté editando, las distintas

variables que se representan en este componente están agrupadas en *VarsMapperItem* que son las encargadas de gestionar individualmente la variable que tengan vinculada.

- **Variable**: Clase que representa la estructura de datos de una variable, incluye toda clase de atributos para definir el nombre, ruta del JSON y tipo. Adicionalmente incluye toda clase de atributos opcionales que dependiendo del tipo que tenga la variable son cubiertos o no desde el *VarsMapper*.

- **VarsTypes**: Enum que indica los distintos tipos de variables que se pueden definir en el *VarsCreator*, dependiendo del tipo de variable que se defina se activaran unas opciones o otras en el *VarsMapper*.

Paquete *ReportLayout*

El paquete *ReportLayout* contiene los componentes que representan la parte central de la aplicación, aquella donde se representan los componentes del paquete *LayoutComponents*.

En la figura 5.15 se muestran las clases que conforman este paquete, el enfoque de construcción de la composición de este paquete viene dado por la representación tradicional de una tabla, de modo que se divide el layout en filas y estas a su vez en columnas, los componentes se introducen en las columnas y el tamaño que ocupan estará determinado por el tamaño de la columna.

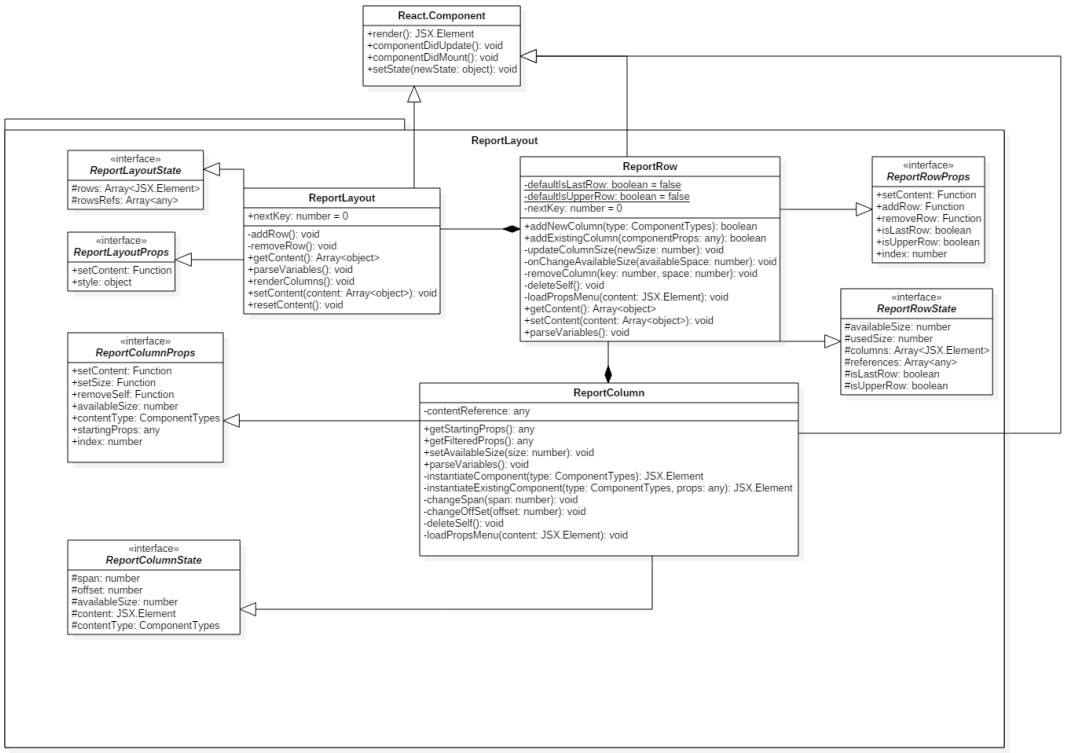

Figura 5.15: Diagrama de clases del paquete *ReportLayout*

La clase principal es *ReportLayout*, esta tiene como único cometido la creación y eliminación de filas de componentes en la plantilla que se está generando, las filas están representadas mediante la clase *ReportRow*.

La clase *ReportRow* se subdivide en columnas y se encarga de gestionar el tamaño que ocupa cada una de ellas, de modo que ninguna de ellas exceda el máximo tamaño que tiene asignada la fila, también gestiona la eliminación de columnas.

La clase *ReportColumn* es la que representa una columna en la plantilla, su cometido es guardar el componente que se desea mostrar en la plantilla, se modo que se encarga de construirlo cuando recibe un evento de la API de *Drag and Drop*. Este componente también puede pasarse a la API de *Drag and Drop* de modo que se puede arrastrar para cambiarlo de posición en la plantilla.

5.4.3. Diseño de clases del servidor

En diseño de las clases del servidor es más sencillo que el del cliente, pues en el simplemente se generan una serie de *endpoints* que invocan servicios REST, estos servicios utilizan por debajo funcionalidades ya implementadas de otros paquetes, como Mongoose para trabajar con la base de datos MongoDB o Phantom

para invocar a PhantomJS, por lo tanto la implementación y diseño de clases en esta frontera de desarrollo es mucho más sencilla y fácil de llevar a cabo.

En la figura 5.16, situada a continuación, se muestran las distintas clases que componen el servidor, la pieza central del servidor es la clase *App* en la que se integran todas las demás, esta clase está a su vez integrada en la clase *Server* que es la encargada de desplegar el servidor en un puerto de escucha determinado.

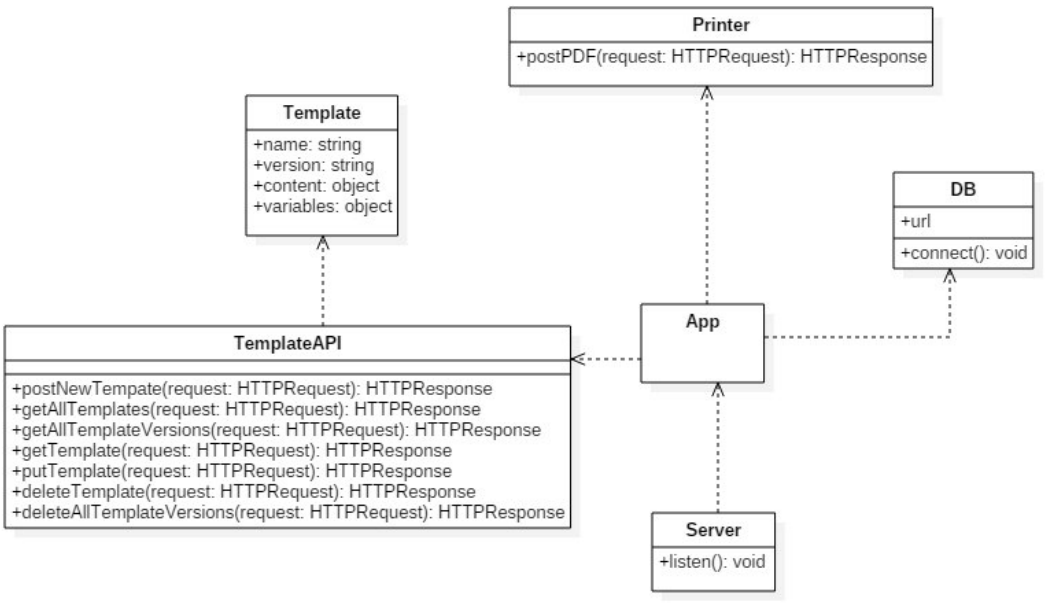

Figura 5.16: Diagrama de clases del servidor

La clase *TemplateAPI* contiene los servicios REST de manipulación de plantillas, esta clase utiliza la clase *Template* para representar un modelo de documento, una estructura de datos organizada y bien definida para guardar en MongoDB, que es la que se utiliza con Mongoose para guardar plantillas de forma homogénea en la base de datos documental MongoDB.

La clase *Printer* contiene el servicio para impresión de PDFs, al cual será necesario transmitirle el HTML que represente la plantilla para su impresión, este HTML será insertado en una página en blanco abierta por PhantomJS y será renderizada como un PDF que será enviado de vuelta al cliente para su posterior descarga.

La clase *DB* contiene la configuración de la base de datos MongoDB que se está utilizando, actúa como conector para la clase *TemplateAPI*.

5.5. Diagramas de interacción

En esta parte se muestran una serie de diagramas de interacción que representan el flujo de operaciones de las funcionalidades más complejas de la aplicación, dichas operaciones incluyen llamadas a componentes externos o llamadas desde clases del cliente a clases del servidor.

Existen otros diagramas de interacción pero se han omitido al ser transversales a las librerías de desarrollo utilizadas (React), a continuación solo se muestran los más relevantes.

5.5.1. Secuencia de llamadas para guardar una plantilla

En la figura 5.17 se muestra la secuencia de llamadas para realizar la funcionalidad de guardar una plantilla, dicha secuencia es la siguiente:

1. El usuario hace click en la opción de **Guardar** en la barra de operaciones CRUD, invocando la función *handleClick* con el parámetro **UPDATE**.

2. El componente *CRUDPanel* registra el evento y realiza una llamada a la función *saveTemplate* del componente núcleo *App*.

3. El componente *App* realiza una llamada al servicio de guardar plantilla mediante la invocación de la función *putTemplate* en el servidor mediante el uso de la URI especificada para invocar la función.

4. El servidor utiliza la función de guardado de datos de la librería Mongoose pasando los datos recibidos en la petición HTTP del cliente.

5. Mongoose invoca una operación de guardado de datos en la base de datos MongoDB.

6. Una vez se ha acabado la transacción en la base de datos se procede a enviar la petición de respuesta al cliente.

7. La respuesta se transmite en cascada en orden inverso de modo que el usuario recibe *feedback* de que la transacción se ha realizado con éxito y el guardado se ha completado.

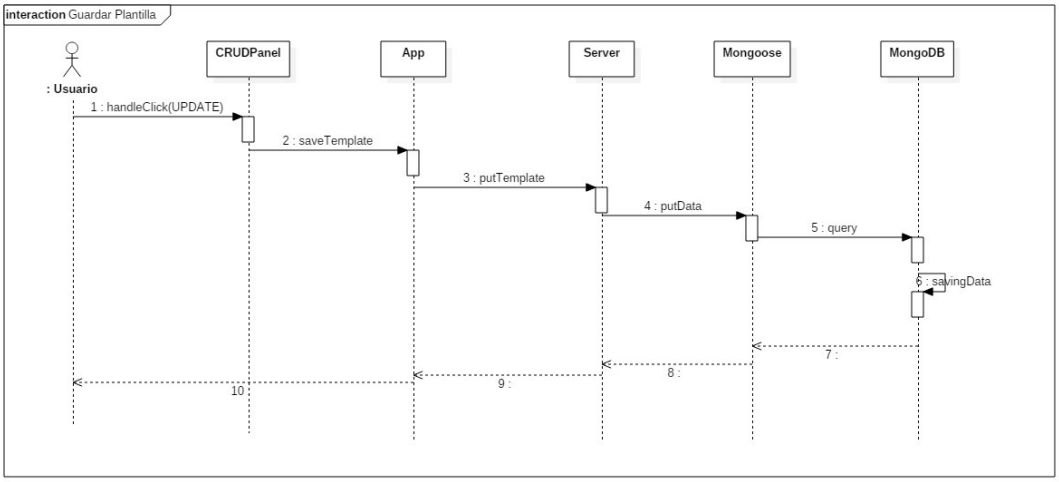

Figura 5.17: Diagrama de interacción para **Guardar una plantilla**

5.5.2. Secuencia de llamadas para cargar una plantilla

En la figura 5.18 se muestra la secuencia de llamadas para realizar la funcionalidad de cargar una plantilla, dicha secuencia es la siguiente:

1. El usuario hace click en la opción de **Cargar** en la barra de operaciones CRUD, invocando la función *handleClick* con el parámetro **READ**.

2. El componente *CRUDPanel* invoca la operación *openModal* del componente *TemplateModal* para abrir el modal de plantillas disponibles al usuario.

3. El componente *TemplateModal* llama al servicio de obtención de todos los nombres de las plantillas disponibles.

4. El servidor transmite la operación de obtención de datos a la librería de Mongoose.

5. Mongoose invoca la operación de obtención de datos en la base de datos MongoDB.

6. El servidor envía los datos de vuelta al cliente una vez que la base de datos devuelve la consulta.

7. El componente *TemplateModal* formatea los datos recibidos y los muestra al usuario.

8. El usuario hace click en la opción de editar de una de las plantillas que se muestran en el modal invocando la operación de *loadTemplateVersions* y pasando como parámetro a la función el nombre de la plantilla seleccionada.

9. El componente *TemplateModal* invoca el servicio de obtención de las distintas versiones de la plantilla seleccionada.

10. El servidor recibe la petición y transmite la operación a la librería de Mongoose.

11. Mongoose invoca la operación de obtención de datos en la base de datos MongoDB.

12. El servidor envía los datos de vuelta al cliente una vez que la base de datos devuelve la consulta.

13. El componente *TemplateModal* recibe las versiones de la plantilla y formatea los datos recibidos para mostrárselos al usuario.

14. El usuario selecciona una versión que se muestra en el modal, invocando el método *loadTemplate* y pasando como parámetros el nombre de la plantilla y su versión.

15. El componente *TemplateModal* invoca el servicio de obtención del contenido de una versión de una plantilla.

16. El servidor recibe la petición y transmite la operación a la librería de Mongoose.

17. Mongoose invoca la operación de obtención de datos en la base de datos MongoDB.

18. El servidor envía los datos de vuelta al cliente una vez que la base de datos devuelve la consulta.

19. El componente *TemplateModal* realiza una llamada a la función *loadTemplate* del componente núcleo *App* pasando la información de la plantilla que se va a cargar.

20. El componente *App* realiza la carga de la información recibida y se la muestra al usuario.

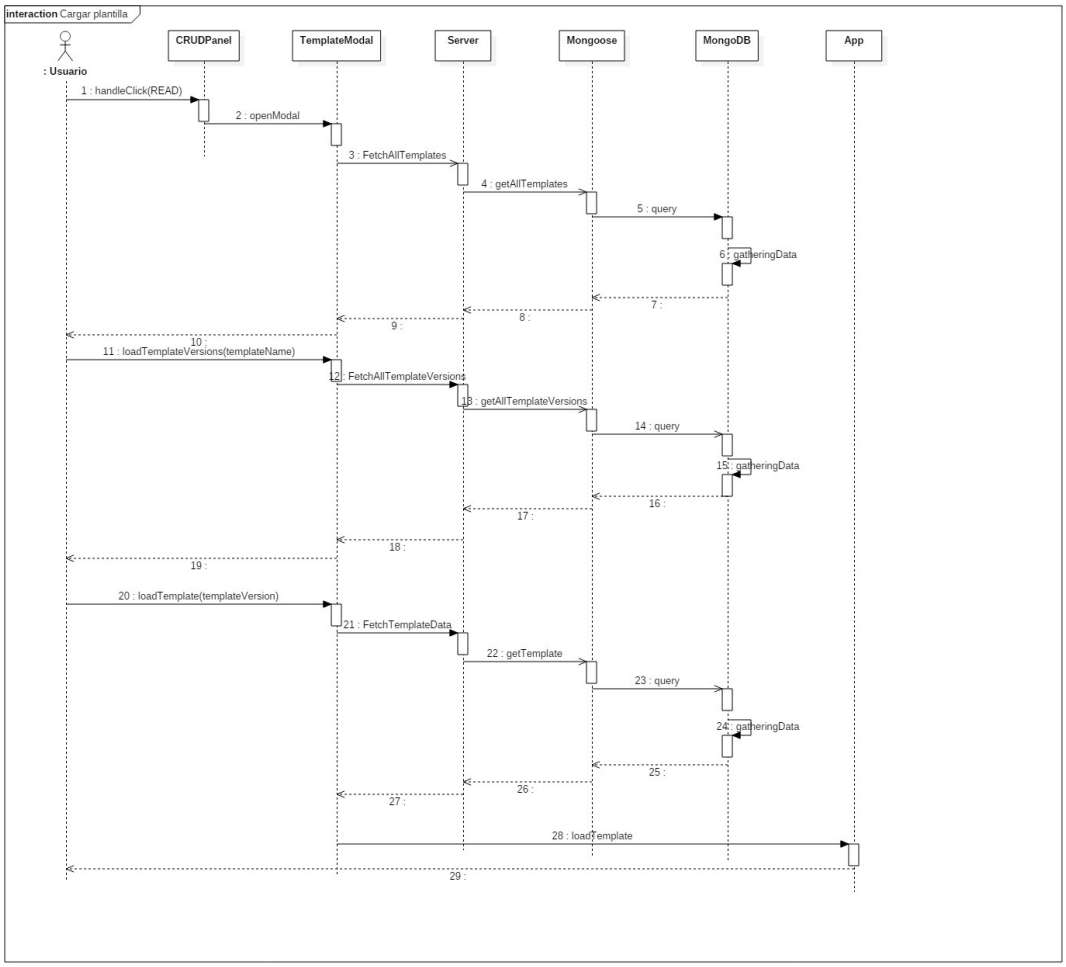

Figura 5.18: Diagrama de interacción para **Cargar una plantilla**

5.5.3. Secuencia de llamadas para imprimir una plantilla a PDF

En la figura 5.19 se muestra la secuencia de llamadas para realizar la funcionalidad de imprimir una plantilla a formato PDF, dicha secuencia es la siguiente:

1. El usuario hace click en la opción de **Imprimir a PDF** en la barra de operaciones CRUD, invocando la función *handleClick* con el parámetro **PDF**.

2. El componente *CRUDPanel* invoca la operación *printToPDF* del componente núcleo *App*.

3. El componente *App* invoca la función de *parseVariables* del componente *ReportLayout* para forzar un reemplazo de variables en el informe.

4. Una vez se ha reemplazado las variables el componente *App* realiza una llamada al servicio de impresión de PDFs del servidor, pasando como parámetros el CSS y el HTML del informe generado, además de dimensionarlo como un folio DIN-A4.

5. El servidor recibe la petición y realiza una llamada a *Phantom* para que abra una página en blanco.

6. Una vez se ha abierto la página en blanco, el servidor modifica el contenido de la misma transmitiéndole a Phantom el HTML y el CSS recibido en la petición HTTP.

7. El servidor ordena a Phantom que haga el renderizado de la página y lo guarde como PDF.

8. El servidor transmite la información del PDF como un *byteStream* al cliente.

9. El cliente recibe los datos e invoca la librería *DownloadJS* para forzar la descarga del archivo PDF a almacenamiento local en el ordenador del usuario.

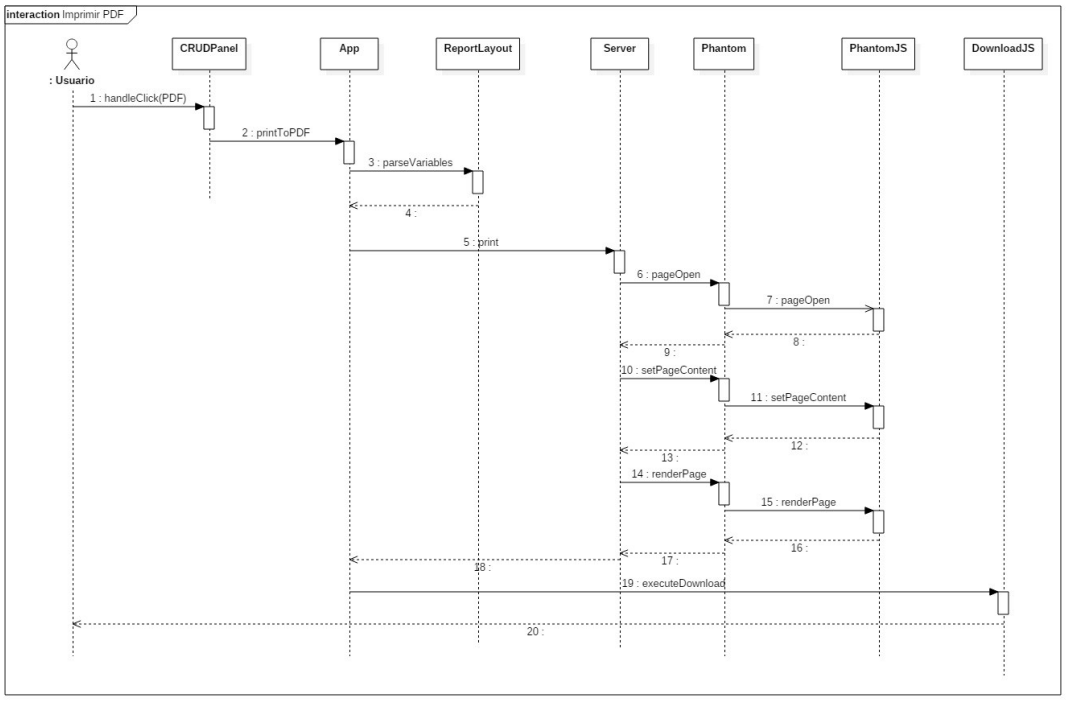

Figura 5.19: Diagrama de interacción para **Imprimir a PDF**

5.6. Diseño de interfaz de usuario

En esta sección se muestran los distintos diseños que se han realizado de la interfaz gráfica.

La realización de estos diseños tiene su relevancia para indicarle al cliente o usuario final de forma abstracta la apariencia de la aplicación junto con los elementos que la van a componer. Mediante la realización de estos diseños se puede obtener *feedback* por parte del usuario y del cliente y de este modo mejorar el diseño inicial, además en caso de que el diseño no satisfaga las demandas del cliente se puede rectificar con suficiente celeridad como para que el proyecto no fracase.

Los diseños iniciales de la interfaz se realizaron en HTML5 y CSS puesto que se cuenta con experiencia y conocimiento suficiente de ambas tecnologías como para crearlos de forma rápida, además se pueden reutilizar los componentes que se creen en futuras versiones de la aplicación.

Se crearon dos diseños de la interfaz, uno inicial muy básico que solo intentaba reflejar la idea general de la aplicación y uno completo en el que se representaban sin funcionalidad alguna los componentes que formarían la aplicación.

La versión de la interfaz representada en al figura 5.20 fue la primera creada para su evaluación por parte del cliente, la interfaz creada es minimalista y sencilla puesto que su único propósito era mostrar la idea de disposición de los componentes que forman la aplicación junto con el propósito de cada uno de ellos.

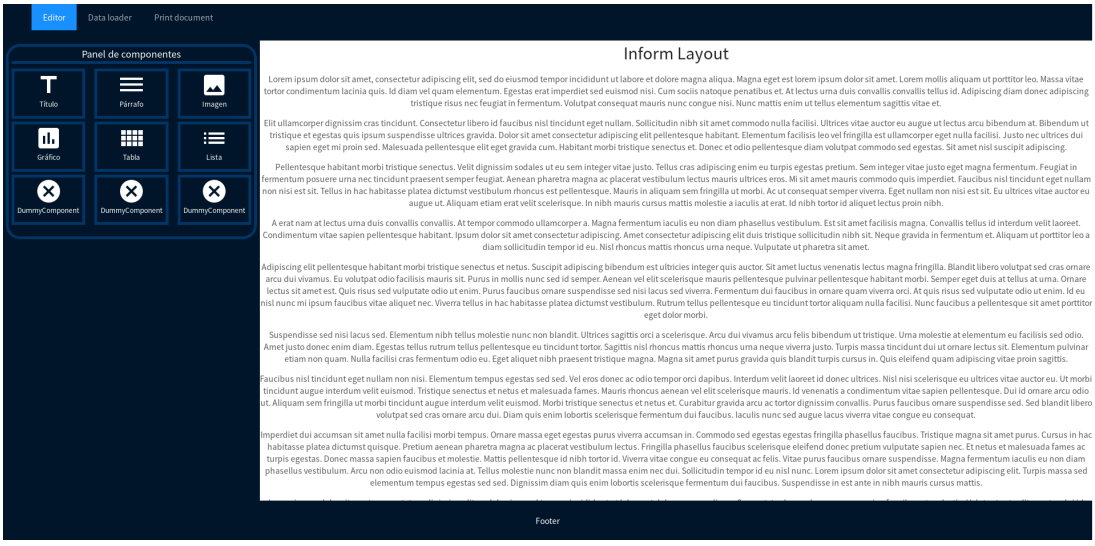

Figura 5.20: Versión minimalista de la interfaz de la aplicación

En este diseño inicial simplemente se mostraba una idea inicial y general de como sería la aplicación, con un panel de componentes en la parte izquierda y un menú de navegación en la parte superior para cambiar de sección, los elementos creados se mostrarían en la parte central de la aplicación y se editarían en esa misma parte.

En base a ese diseño se obtuvo *feedback* del cliente y se aplicaron mejoras y cambios al diseño minimalista, las ideas se han basado en el boceto de disposición de elementos representado en la figura 5.21. En base a este boceto se creó una versión nueva y más avanzada de la interfaz que incluía todos los componentes situados de la misma manera que en el boceto.

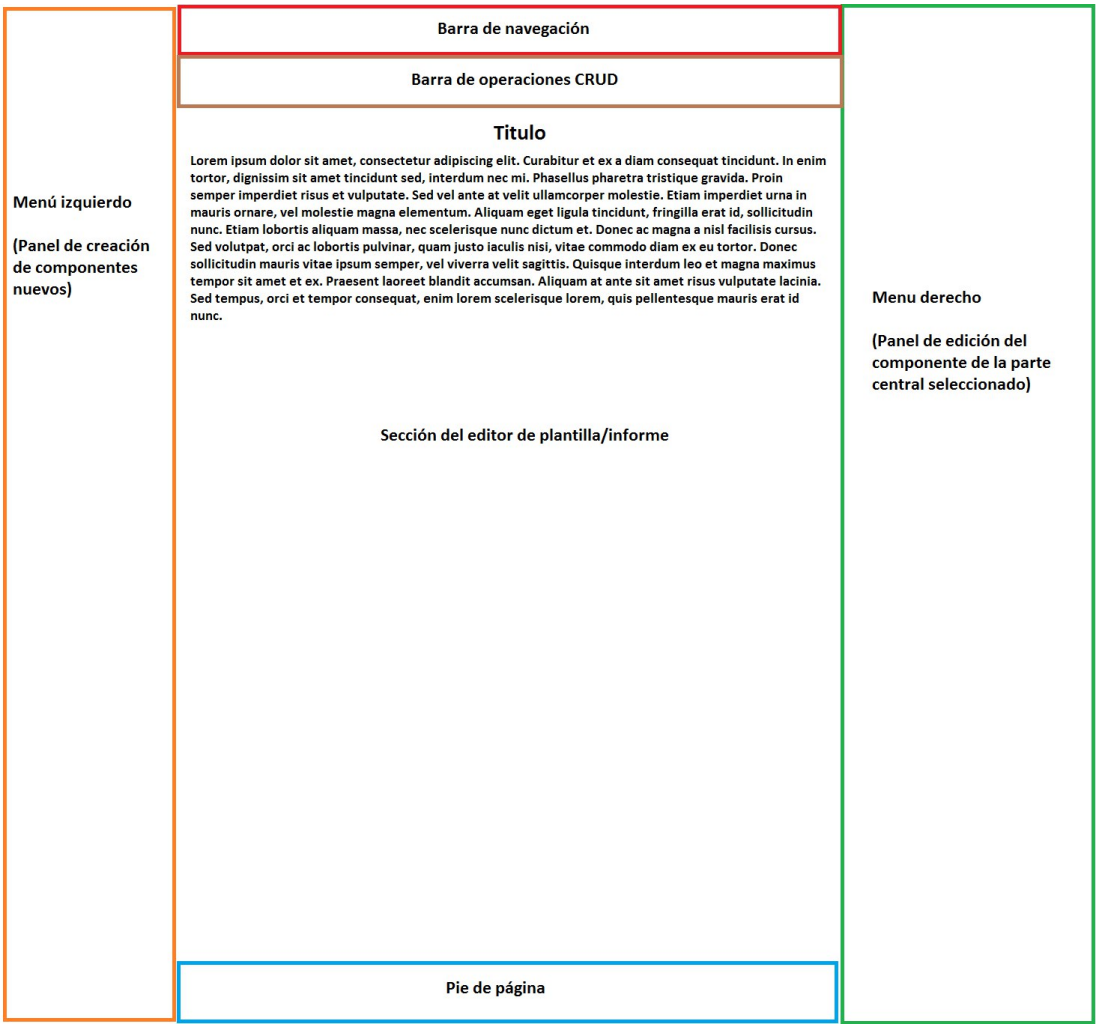

Figura 5.21: Boceto de la disposición de los elementos en la interfaz completa

En la siguiente versión de la interfaz se aplicaron mejoras de usabilidad al diseño inicial, como uso de colores más claros para el contenido general y colores

oscuros para contenido importante a resaltar, división de los menús en paneles plegables y traslado de las opciones de edición de los componentes al menú derecho en vez de editarlas en la parte central de la aplicación.

Como consecuencia de lo anterior se obtuvo el diseño de la interfaz presente en la figura 5.22.

Este diseño convenció más al cliente por lo que se procedió a desarrollar las funcionalidades de la aplicación en torno a este diseño de interfaz.

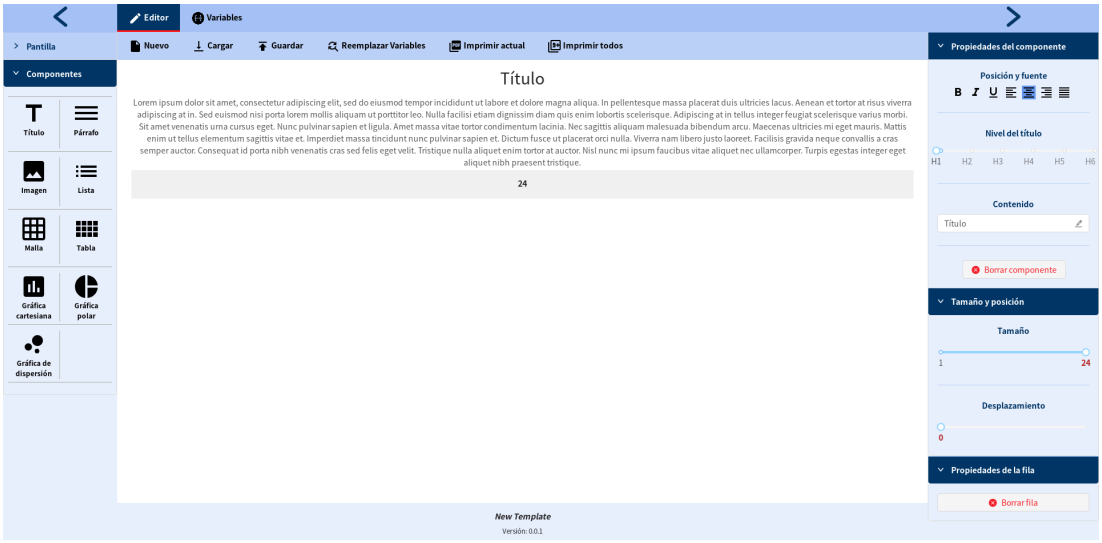

Figura 5.22: Versión completa de la interfaz de la aplicación

Capítulo 6

Validación y pruebas

En esta sección se indican las distintas pruebas que se han realizado contra el software que se ha desarrollado a lo largo de este proyecto.

Esta fase tiene dos objetivos principales:

- Comprobar que el software programado funciona correctamente ante las entradas y salidas contempladas y no contempladas en sus algoritmos y formularios, más conocido como fase de verificación.

- Comprobar que el software construido cumple con las expectativas del cliente, esto es, que el software resuelve el problema que el cliente ha planteado de forma correcta según sus criterios, también conocido como fase de validación.

A continuación se muestran los distintos tipos de pruebas que se han realizado junto con sus resultados.

6.1. Pruebas unitarias

Las pruebas unitarias tienen como objetivo comprobar que cada funcionalidad de la aplicación comprobada de forma independiente funciona según lo esperado ante parámetros válidos e inválidos, estas pruebas tienen que ser lo más completas posibles, por lo que tienen que probar la mayor cantidad de código posible y el mayor rango posible de parámetros de entrada.

ID	PU-01
Descripción	Se abre la interfaz web en la sección del editor y se selecciona la opción de *Nueva*, a continuación se indica el nombre de la plantilla y la versión y se confirma la creación.
Casos de uso asociados	**CU-04**
Criterio de aceptación	*La parte central de la aplicación cambia al contenido de una plantilla en blanco, además el nombre y versión de la plantilla aparecen correctamente en el pie de página*
Estado	**Superada**

ID	PU-02
Descripción	Se abre la interfaz web en la sección del editor y se selecciona la opción de *Guardar*.
Casos de uso asociados	**CU-05**
Criterio de aceptación	*El contenido que existe en la plantilla es convertido a formato JSON y "enviado al servidor".*
Estado	**Superada**

ID	PU-03
Descripción	Se abre la interfaz web en al sección del editor y se selecciona una versión de una plantilla para su carga.
Casos de uso asociados	**CU-06**
Criterio de aceptación	*El contenido de la parte central de la aplicación cambia para mostrar el contenido de la plantilla seleccionada, además el pie de página se actualiza con el nombre y versión de la plantilla seleccionada.*
Estado	**Superada**

ID	PU-04
Descripción	Se abre la interfaz web en la sección del editor y se selecciona la opción de borrado de una plantilla existente.
Casos de uso asociados	**CU-07**
Criterio de aceptación	*La plantilla es borrada y desaparece de la lista para su selección.*
Estado	**Superada**

ID	PU-05
Descripción	Se abre la interfaz web y se procede a arrastrar cualquier componente de texto (título, párrafo o lista) del panel de componentes a la parte central de la aplicación donde posteriormente se suelta en una fila con espacio disponible.
Casos de uso asociados	**CU-01**
Criterio de aceptación	*Se crea un componente de texto con contenido por defecto en la posición seleccionada.*
Estado	**Superada**

ID	PU-06
Descripción	En la sección de variables se procede a escribir el nombre de una variable y seleccionar cualquiera de los tipos disponible, posteriormente se pulsa la opción de añadir variable.
Casos de uso asociados	**CU-08**
Criterio de aceptación	*Se crea la variable con el nombre y tipo seleccionado y se añade a la tabla de variables.*
Estado	**Superada**

ID	PU-07
Descripción	En la sección de variables, se pulsa el botón de cargar JSON y se selecciona un archivo en formato JSON para su carga.
Casos de uso asociados	**CU-11**
Criterio de aceptación	*El JSON queda cargado en la aplicación y se muestra su contenido.*
Estado	**Superada**

ID	PU-08
Descripción	Se selecciona un componente ya creado en la parte central de la aplicación y se procede a modificar su tamaño y desplazamiento junto con sus opciones propias de contenido.
Casos de uso asociados	**CU-02**
Criterio de aceptación	*El tamaño y desplazamiento del componente aumenta o se reduce en base al espacio disponible y según el criterio indicado por el usuario, además el contenido del componente se modifica según el criterio del usuario.*
Estado	**Superada**

ID	PU-09
Descripción	Se selecciona un componente ya creado y se pulsa en la opción de borrar componente.
Casos de uso asociados	**CU-03**
Criterio de aceptación	*El componente es borrado completamente y queda su espacio como una columna vacía.*
Estado	**Superada**

ID	PU-10
Descripción	Se selecciona una componente lista y se pasa su tipo a dinámico, a continuación se selecciona una fuente de datos de las disponibles y se pulsa en el botón de cargar.
Casos de uso asociados	**CU-09**
Criterio de aceptación	*La lista refleja el contenido de la fuente de datos seleccionada.*
Estado	**Superada**

ID	PU-11
Descripción	En la sección de variables se pulsa sobre el botón de cargar JSON y se seleccionan múltiples archivos en formato JSON para su carga.
Casos de uso asociados	**CU-11**
Criterio de aceptación	*Se cargan en la aplicación todos los JSON válidos y se muestra al usuario la lista de cual está seleccionado actualmente.*
Estado	**Superada**

ID	PU-12
Descripción	Se selecciona un componente de texto cualquiera y se procede a insertar una variable mediante la sintaxis {{*nombre de la variable*}}.
Casos de uso asociados	**CU-09**
Criterio de aceptación	*La variable queda vinculada al texto, por lo que será reemplazada cuando el usuario seleccione la opción de reemplazar variables.*
Estado	**Superada**

ID	PU-13
Descripción	Se selecciona un componente tabla y se cambia su tipo a dinámico, a continuación se selecciona una fuente de datos y se pulsa el botón de cargar fuente de datos.
Casos de uso asociados	**CU-09**
Criterio de aceptación	*El contenido de la tabla se modifica para representar la fuente de datos que se carga.*
Estado	**Superada**

ID	PU-14
Descripción	Se selecciona un componente gráfica cualquiera y se cambia su tipo a dinámico, a continuación se selecciona una fuente de datos y se pulsa el botón de cargar fuente de datos
Casos de uso asociados	**CU-09**
Criterio de aceptación	*El contenido de la gráfica se adapta a la fuente de datos seleccionada.*
Estado	**Superada**

ID	PU-15
Descripción	Con variables variables vinculadas a componentes de texto y fuentes de datos seleccionadas en gráficas y tablas se pulsa el botón de *Reemplazar variables.*
Casos de uso asociados	**CU-10**
Criterio de aceptación	*Las variables vinculadas a componentes de texto se reemplazan correctamente y el contenido de las fuentes de datos se carga en los componentes correspondientes.*
Estado	**Superada**

ID	PU-16
Descripción	Con una plantilla con ciertos componentes creados se pulsa el botón de *Imprimir a PDF*.
Casos de uso asociados	**CU-12**
Criterio de aceptación	*Se recibe un PDF con el contenido de la plantilla.*
Estado	**Superada**

ID	PU-17
Descripción	Con una plantilla con ciertos componentes y múltiples JSON se pulsa el botón de *Imprimir todos*.
Casos de uso asociados	**CU-12**
Criterio de aceptación	*Se recibe un PDF por cada JSON reemplazando las variables para cada JSON.*
Estado	**Superada**

6.2. Pruebas de integración

Las pruebas de integración tienen como objetivo comprobar que la interacción entre los distintos módulos que forman la aplicación se realiza correctamente, de modo que tenemos la seguridad que la funcionalidad completa se realiza correctamente sin fallo alguno y que muestra los resultados correctos.

ID	PI-01
Descripción	Con una plantilla con contenido ya creada se procede a pulsar en el botón de *Guardar*
Criterio de aceptación	*El contenido es transmitido por HTTP al servidor, este lo recibe y lo guarda en la base de datos, finalmente el cliente recibe una respuesta HTTP 200 indiciando que la operación ha sido un éxito.*
Estado	**Superada**

ID	PI-02
Descripción	Se abre la interfaz de la aplicación y se pulsa en el menú de cargar para obtener todas las plantillas disponibles del servidor.
Criterio de aceptación	*Se transmite una petición HTTP GET al servidor para obtener todas las plantillas, este contesta con otra petición HTTP 200 con las plantillas obtenidas, a continuación el cliente las muestra todas en forma de lista paginada.*
Estado	**Superada**

ID	PI-03
Descripción	Con el menú de cargar plantillas abierto se selecciona una de las plantillas de la lista para obtener sus versiones.
Criterio de aceptación	*Se transmite una petición HTTP GET al servidor para obtener todas las versiones de la plantilla seleccionada, a continuación el servidor responde con una petición HTTP 200 con las versiones de la plantilla obtenidas, el cliente muestra estas versiones en forma de lista paginada.*
Estado	**Superada**

ID	PI-04
Descripción	Con el menú de cargar plantillas abierto se selecciona una de la versiones de una plantilla determinada y se pulsa en la opción de *editar*.
Criterio de aceptación	*Se transmite una petición HTTP GET al servidor para obtener el contenido de la versión de la plantilla seleccionada, el servidor responde con un HTTP 200 con el contenido de la versión, el cliente carga todos los datos recibidos y cambia la parte central de la aplicación para que represente el contenido recibido.*
Estado	**Superada**

ID	PI-05
Descripción	Con el menú de cargar plantillas abierto se selecciona la opción borrar de una versión cualquiera de una plantilla.
Criterio de aceptación	*Se envía una petición HTTP DELETE al servidor para borrar la versión seleccionada, el servidor borra la versión seleccionada y envía una petición HTTP 200 al cliente informando de que se ha borrado con éxito.*
Estado	**Superada**

ID	PI-06
Descripción	Con el menú de cargar plantillas abierto se selecciona la opción de borrar una plantilla cualquiera.
Criterio de aceptación	*Se envía una petición HTTP DELETE al servidor para que borre todas las versiones de la plantilla seleccionada, el servidor borra de la base de datos todas las versiones de la plantilla seleccionada y envía al cliente una respuesta HTTP 200 indicando que la operación se ha completado con éxito*
Estado	**Superada**

ID	PI-07
Descripción	Con una plantilla con contenido se selecciona la opción de *Imprimir PDF*.
Criterio de aceptación	*El cliente comprime el contenido de la plantilla para que se adecue a las dimensiones de un folio DIN A4, a continuación envía una petición HTTP POST al servidor con el código HTML y CSS de la plantilla junto con el nombre, versión y dimensiones del folio, el servidor recibe la petición y procede a crear el PDF invocando a* ***PhantomJS****, una vez creado el PDF se envía al cliente con un HTTP 200 como un blob de datos, el cliente recibe la petición e invoca a* ***DownloadJS*** *para que ejecute la descarga del contenido recibido.*
Estado	**Superada**

ID	PI-08
Descripción	Con una plantilla con variables y fuentes de datos y un JSON cargado en la aplicación se selecciona la petición de *Imprimir a PDF*.
Criterio de aceptación	*El cliente reemplaza todas las variables y fuentes de datos primero en la plantilla, comprime la plantilla para que se adecue a un folio DIN A4 y envía el código HTML y CSS al servidor junto con el nombre, versión y dimensiones del folio, el servidor recibe la petición HTTP POST del cliente y crea el PDF con* ***PhantomJS*** *a continuación envía el PDF al cliente con una respuesta HTTP 200 como blob de datos, el cliente invoca a* ***DownloadJS*** *para que ejecute la descarga del contenido recibido.*
Estado	**Superada**

ID	PI-09
Descripción	Con una plantilla con variables y fuentes de datos y múltiples JSONs cargados en la aplicación se selecciona la petición *Imprimir todos*.
Criterio de aceptación	*El cliente selecciona el primer JSON y reemplaza las variables y fuentes de datos, comprime la plantilla en DIN A4 y envía el código HTML y CSS al servidor junto con el nombre, versión y dimensiones del folio, el servidor recibe la petición HTTP POST del cliente y crea el PDF con* ***PhantomJS****, a continuación envía el PDF al cliente con una respuesta HTTP 200 con un blob de datos, el cliente invoca a* ***DownloadJS*** *para que ejecute la descarga. El cliente cambia de JSON y vuelve a repetir los pasos anteriores, así hasta haber seleccionado todos los JSONs que había cargados.*
Estado	**Superada**

6.3. Validación de interfaz de usuario

Para evaluar la usabilidad y la validez de la interfaz de usuario diseñada en el sistema se realizó un cuestionario SUS a distintos usuarios y se anotaron los resultados, adicionalmente se comprobó que la interfaz cumpliera con los heurísticos de ***Nielsen***.

6.3.1. Heurísticos de Nielsen

Se ha analizado que la interfaz cumpla en mayor o menor medida los heurísticos de Nielsen [25] siendo estos los siguientes:

1. **Visibilidad del estado del sistema**: El sistema debe de dar información de su estado al usuario/s dando retroalimentación adecuada dentro de un intervalo apropiado.

2. **Utilización del lenguaje de los usuarios**: El sistema debe de utilizar el lenguaje de los usuarios, en lugar de utilizar términos propios del sistema.

3. **Control y libertad del usuario**: Si el usuario selecciona una opción por error debe de contar con opciones para deshacer o rehacer dicha acción.

4. **Consistencia y estándares**: Los términos que se muestren en el sistema deben de seguir la convención de la plataforma en la que se desarrolla, de modo que el significado de las palabras o situaciones del sistema se mantienen consistentes.

5. **Prevención de errores**: El sistema debe de priorizar evitar los errores antes de que se produzcan y el usuario tenga que solucionarlos, o en todo caso preguntar confirmación al usuario.

6. **Minimización de la carga de la memoria del usuario**: El sistema debe de minimizar la información que el usuario necesita memorizar para utilizalo, las instrucciones de uso deben de estar visibles en todo momento.

7. **Flexibilidad y eficiencia de uso**: El sistema debe de permitir el uso de opciones avanzadas que agilicen su uso para usuario expertos.

8. **Diseño minimalista**: No conviene inundar la interfaz con información redundante o inútil.

9. **Diagnóstico y reconocimiento de errores**: El sistema deberá de proporcionar retroalimentación al usuario en caso de que se produzcan errores, ayudando a este a solucionarlos.

10. **Documentación**: La documentación de uso del sistema debe de ser sencilla y fácil de encontrar, además debe de cubrir la mayor cantidad de aspecto del sistema.

Los resultados del análisis se basan en los siguientes criterios:

- Se puntúa de 0 a 10 el cumplimiento del criterio heurístico de Nielsen, siendo 0 un cumplimiento nulo y 10 un cumplimiento absoluto.

- Se tiene en cuenta el contexto del programa, esto es el ambiente en el que va a ser utilizado.
- Se adjuntan las razones que definen la escala de cumplimiento.
- El criterio de puntuación del análisis es subjetivo y debe ser realizado desde un punto de vista autocrítico.
- La puntuación final es la media aritmética de la puntuación de cada término.
- Se considera que el sistema cumple los heurísticos de Nielsen si la puntuación es igual o superior a **7,5**.

Los resultados obtenidos fueron los siguientes:

- **Visibilidad del estado del sistema** [*10*]: El sistema informa a los usuarios en que sección se encuentran en todo momento, adicionalmente los cambios en la plantilla se reflejan correctamente y se muestra de forma adecuada que cambios producen las acciones en la plantilla, se indica correctamente cuando las variables están o no siendo reemplazadas además de indicar que se está(n) generando correctamente el/los PDF(s).
- **Utilización del lenguaje de los usuarios** [*8*]: En su conjunto el sistema utiliza un lenguaje común a los usuarios del mismo, sin embargo existen unos cuantos términos que son específicos del sistema, como las gráficas dinámicas y el tratamiento de las variables, estos términos no son intuitivos para el usuario.
- **Control y libertad del usuario** [*4*]: El sistema carece de opciones de deshacer y rehacer en su conjunto, aun así se puede guardar el estado de una plantilla en el servidor y cargarlo en caso de error de edición. Sin embargo esta solución no aporta un grado de cumplimiento adecuado del heurístico, existen restricciones en el uso general de los componentes y en el modo de situarlos en la pantalla, la flexibilidad de uso de los componentes es limitada.
- **Consistencia y estándares** [*10*]: El sistema mantiene la consistencia de sus términos en todo momento, puede apreciarse esto en el mantenimiento del significado de la palabra *variable* o *dinámica* en los distintos componentes, además de seguir los estándares de la plataforma en la que se desarrollo correctamente.
- **Prevención de errores** [*7*]: El sistema evita que se produzcan errores, como evitar la creación de una plantilla sin nombre o sin versión, sin embargo consiente que se produzcan otros, concretamente la introducción de valores erróneos en algunos formularios aunque se avisa del error en estos casos.

- **Minimización de la carga de la memoria del usuario** [*8*]: En su conjunto el sistema utiliza conceptos sencillos para realizar acciones, de modo que el usuario no tiene que memorizar grandes conceptos de uso, adicionalmente el sistema cuenta con opciones de autocompletado para la selección de variables y la edición de las mismas de modo que se sugiere al usuario el nombre de la variable en base a los criterios que está buscando. Sin embargo el usuario tiene que memorizar algunos conceptos de uso de las funcionalidades más avanzadas, como las gráficas o la configuración de una serie de datos desde variable.

- **Flexibilidad y eficiencia de uso** [*7*]: El sistema cuenta con opciones para usuarios avanzados, en concreto para la generación y edición de plantillas desde carga y reemplazo de variables, la opción de cargar múltiples JSON e imprimir múltiples PDF ayuda también en este criterio. Sin embargo faltan atajos de selección de componentes o de cambio de sección.

- **Diseño minimalista** [*10*]: La interfaz en su conjunto es minimalista, apenas contiene información y solo muestra la información necesaria para el usuario, se utilizan iconos o elementos interactivos para realizar la mayoría de las acciones eliminando de este modo contenido innecesario de la interfaz.

- **Diagnóstico y reconocimiento de errores** [*10*]: El sistema informa correctamente de los errores que se producen en el, adicionalmente las notificaciones de los errores son visibles e indican al usuario de la situación que ha ocurrido.

- **Documentación** [*8*]: Algunos aspectos no se cubren con la suficiente profundidad y falta algún que otro detalle menor en la documentación, sin embargo en su conjunto la documentación es sencilla de utilizar e informa de la mayoría de los casos de uso importantes.

La puntuación final es **8,2** de modo que se considera que el sistema cumple con los heurísticos de Nielsen.

6.3.2. Cuestionario SUS

El cuestionario SUS consiste en una serie de preguntas que se realizan contra una serie de usuarios tras probar la aplicación sin conocerla de antemano.

El cuestionario tiene los siguientes criterios:

- Consta de 10 preguntas

- Las preguntas se puntúan de 0 a 5, siendo 0 *Muy en desacuerdo* y 5 *Muy de acuerdo*.

- La puntuación de las preguntas pares es inversamente proporcional a la que da el usuario.

- Todas las preguntas deben de ser contestadas, no hay posibilidad de abstención en una pregunta.

- La puntuación final se calcula como el doble del sumatorio de la puntuación de cada pregunta.

Se considera que el sistema es usable si la puntuación es igual o superior a los **75** puntos.

Las preguntas que se adjunta en el cuestionario son las siguientes:

1. Encuentro el sistema sencillo de utilizar.

2. Considero que es necesario leerse un manual o documentación para poder utilizar el sistema correctamente.

3. No aprecio inconsistencias en los términos que utiliza el sistema.

4. Percibo que el sistema no me informa de los errores que ocurren en el.

5. Controlo en todo momento las acciones que estoy realizando en el sistema.

6. El uso del sistema me resulta excesivamente complejo.

7. No considero que sea necesario la ayuda de ningún especialista o de documentación para utilizar el sistema correctamente.

8. Me pierdo frecuentemente en las acciones que realizo en el sistema.

9. El sistema informa de forma clara y concisa del estado de las operaciones que se realizan en el.

10. Considero que es necesario mucha práctica para poder utilizar el sistema de forma correcta.

Este cuestionario se realizó contra 5 usuarios diferentes, los resultados obtenidos se reflejan en la siguiente tabla:

	Pregunta 1	Pregunta 2	Pregunta 3	Pregunta 4	Pregunta 5	Pregunta 6	Pregunta 7	Pregunta 8	Pregunta 9	Pregunta 10	Puntuación
Usuario 1	5	2	5	3	4	0	3	1	5	0	82/100
Usuario 2	4	1	4	3	5	1	4	2	4	1	76/100
Usuario 3	4	3	5	2	2	1	5	2	5	1	80/100
Usuario 4	5	2	5	0	4	3	2	3	4	3	72/100
Usuario 5	5	0	5	0	4	3	4	0	5	0	90/100

La media obtenida de la puntuación de los usuarios es de **80** puntos, por lo que se considera que la interfaz es usable en su conjunto debido a que es superior al límite de 75 puntos establecido.

6.4. Matriz de trazabilidad

	PU-01	PU-02	PU-03	PU-04	PU-05	PU-06	PU-07	PU-08	PU-09	PU-10	PU-11	PU-12	PU-13	PU-14	PU-15	PU-16	PU-17
CU-01					X												
CU-02								X									
CU-03									X								
CU-04	X																
CU-05		X															
CU-06			X														
CU-07				X													
CU-08						X											
CU-09										X		X	X	X			
CU-10															X		
CU-11							X				X						
CU-12																X	X

Cuadro 6.1: Matriz de trazabilidad Pruebas Unitarias - Casos de uso

Capítulo 7

Conclusiones y ampliaciones

En este apartado se comentan las conclusiones obtenidas tras la realización de este trabajo fin de grado, adicionalmente se indican las posibles ampliaciones que se le podrían aplicar para conseguir que la aplicación sea más funcional o presente mejoras de rendimiento.

7.1. Conclusiones

Como principal conclusión mencionar que el uso de las tecnologías de *Big Data* y en especial de *Business Intelligence* ha provocado un aumento sustancial de la información que tenemos a nuestra disposición, provocando que sea necesario la implementación de software adicional para mostrar dicha información de forma legible y directa para los usuarios, normalmente, en forma de informes.

Dichos programas presentan el problema de que son construidos a medida para un sistema de **BI** concreto, provocando que sea necesario adaptarlos cuando sea necesario modificar el modelo de datos a mostrar, en este trabajo se ha desarrollado un software que permite la creación de informes que no dependen de un modelo de datos concreto, proponiendo de este modo una solución al problema anterior, aunque este software no es perfecto ni aplicable a absolutamente todos los contextos, la idea general de construir de forma genérica vistas asociadas a modelos de datos que el usuario cargue, y todo lo anterior de la forma más sencilla posible, permite solucionar la falta de flexibilidad de los sistemas anteriormente mencionados.

Por una banda, destacar la importancia del uso de la computación distribuida en el mundo actual, especialmente el paradigma de arquitecturas orientadas a servicios. Esto es debido a que es posible la creación de aplicaciones cliente encargadas de realizar ciertas funcionalidades que antes solo se podían realizar en el servidor, de modo que es posible distribuir la carga que antes residía ple-

namente en los servidores entre el cliente y el servidor, ahorrando de este modo capacidad de cómputo. Gracias a esto es posible crear arquitecturas escalables aplicables al mundo de información volátil actual en el que es necesario continuas adaptaciones para mantener la escalabilidad de los sistemas, además de facilitar el desacoplamiento de funcionalidades y aumentar la seguridad de los servidores.

Adicionalmente, durante este TFG se ha obtenido conocimiento sobre un gran abanico de tecnologías y frameworks, destacando que el uso correcto de estas tecnologías puede agilizar enormemente el desarrollo de cualquier sistema software actual, esto es debido a que se evita "reinventar la rueda" y por ello se puede invertir el tiempo de desarrollo desarrollando funcionalidades nuevas en vez de recrear las que ya se han implementado anteriormente.

Por otra banda, destacar la importancia de la aplicación y uso de patrones de diseño en el desarrollo de software, pues gracias a ellos es posible la realización de aplicaciones altamente escalables, fáciles de mantener y cuyas partes pueden ser reutilizadas en otras aplicaciones distintas. En este trabajo se han aplicado distintos patrones de diseño, incluyendo composición, herencia y división modular (*module pattern*) los cuales han permitido agilizar el desarrollo de la aplicación además de facilitar la corrección de errores en fases más avanzadas del desarrollo sin necesidad de destruir gran parte del código.

Finalmente, mencionar la importancia de la generación de documentación asociada al uso de un sistema software, tecnología determinada o framework, pues ella es la que permite el uso correcto de todo lo anterior además de agilizar la curva de aprendizaje requerida para usar correctamente lo documentado.

Sin embargo, es necesario mencionar también las dificultades presentadas a la hora de desarrollar este proyecto.

La principal dificultad presentada en el proyecto es el uso de tecnologías nuevas por primera vez por parte del desarrollador, en concreto el uso del lenguaje JavaScript. Aunque JavaScript es un lenguaje sencillo de comprender inicialmente puede llegar a ser complejo de depurar debido a que no está fuertemente tipado a diferencia de otros lenguajes de programación como C++ o Java que tipifican de forma estática, sin embargo esto puede solventarse mediante el uso de lenguajes de superconjunto, como TypeScript, que añaden tipado estático a JavaScript.

Otra de las grandes dificultades fue el uso del framework React, este framework escrito en JavaScript presenta una curva de aprendizaje muy elevada al introducir conceptos nuevos como ciclos de vida de componentes, patrón de composición y renderizado que no son conceptos sencillos de comprender inicialmente. Esto ha provocado que fuese necesario esfuerzo adicional para empezar a utilizar este fra-

mework, además de aplicar múltiples refactorizaciones al código de la aplicación según el conocimiento que se poseía sobre el framework iba aumentando durante el transcurso del proyecto.

7.2. Ampliaciones

Aunque se alcanzaron los objetivos contemplados en el alcance del proyecto y los requisitos propuestos por el cliente, el software desarrollado presenta carencias y faltas en ciertos aspectos.

En primer lugar, el software solo permite cargar modelos de datos en formato JSON, una de las posibles ampliaciones sería generalizar la carga de modelos de datos para permitir de este modo la carga de modelos de datos en formato **XML**, **CSV** o incluso extraídos desde bases de datos relacionales.

Adicionalmente, el software cuenta con un conjunto limitado de opciones de edición, pues solo se ha contemplado opciones básicas y mínimas para dar legibilidad a un informe, otras de las posibles ampliaciones sería añadir más opciones de edición a los componentes de la aplicación, como por ejemplo cambiar el color de la fuente, cambiar el tamaño de la fuente, cambiar el tipo de fuente o el color de fondo del componente.

Otra de las posibles ampliaciones sería añadir otro tipo de componentes a la aplicación, como árboles, diagramas o añadir otros tipos de gráficas que no están contempladas en el sistema actual.

También es posible aumentar las opciones de impresión, permitiendo que el informe generado sea utilizado para distintos tipos de folios como DIN A3, DIN A2 o formato carta.

Como última posible ampliación propuesta estaría una mejora general del rendimiento de la aplicación, pues en caso de usar un informe con muchas gráficas y tablas el rendimiento de la aplicación se mina al provocar cambios de vista, principalmente la mejora de rendimiento implicaría el uso de transformaciones CSS3 y uso de GPU para calcular las transiciones y animaciones de la interfaz y de este modo mantener el rendimiento.

Apéndice A

Manuales técnicos

En esta sección se adjuntan los manuales relativos a la parte técnica del proyecto, incluyéndose el despliegue de la aplicación junto con las opciones de configuración que esta pueda presentar y el modo de configurarlas.

A.1. Manual de despliegue

Para desplegar la aplicación es necesario las siguientes aplicaciones/entornos, estos no se proporcionan en el CD del código de la aplicación y deben de ser instalados aparte:

- **Node.js** (*Versión 9.11.1*), es el entorno en el que se despliega la capa de servicios y la aplicación web, se puede descargar desde `https://nodejs.org/en/`.
- **NPM** (*Versión 6.1.0*), gestor de dependencias utilizado en las aplicaciones, se puede descargar desde `https://www.npmjs.com/get-npm`.
- **Docker** (Opcional) (*Versión 18.03.1-ce build 9ee9f40*), utilizado para desplegar los contenedores de la base de datos MongoDB actual, se puede descargar desde `https://www.docker.com/get-started`.

El uso de Docker es opcional si se opta por utilizar otro tipo de base de datos MongoDB en otra URI de acceso, el uso de Node.js y NPM es obligatorio si se desea desplegar la aplicación web y la capa de servicios puesto que ambas fueron desarrolladas dentro de ese entorno.

Para desplegar la aplicación web:

1. Descomprimir su contenido en un directorio.
2. Abrir una terminal del sistema operativo (bash, xterm, powershell, ...).

3. Situar la terminal en el directorio donde está la aplicación web y donde se sitúa el archivo *package.json* de NPM.

4. Ejecutar el comando *npm start*.

Para desplegar la capa de servicios:

1. Descomprimir su contenido en un directorio.

2. Abrir una terminal del sistema operativo (bash, xterm, powershell, ...).

3. Situar la terminal en el directorio donde está la capa de servicios y donde se sitúa el archivo *package.json* de NPM.

4. Ejecutar el comando *npm start*.

Para desplegar la base de datos mongoDB:

1. Abrir una terminal del sistema operativo (bash, xterm, powerhsell, ...).

2. Situar la terminal el directorio del archivo *tar* que contiene la imagen de la base de datos.

3. Ejecutar el comando *docker load -i* <***nombre del archivo tar***>.

4. Ejecutar el comando *docker images* para obtener el ID de la imagen que se ha cargado.

5. Ejecutar el comando *docker run –name* <***nombre del contenedor***> *-i -t* <***ID de la imagen***> para crear el contenedor e iniciarlo.

6. Ahora solo es necesario ejecutar el comando *docker start* <***nombre del contenedor***> para arrancar el contenedor anterior en caso de que se pare o se cierre la terminal anterior.

A.2. Manual de configuración

A.2.1. Cambiar la URI de la capa de servicios

En caso de que la capa de servicio se despliegue en una URI diferente a *localhost* es necesario modificar el archivo **App.tsx** o **App.js** de la aplicación web.

Si se opta por modificar el archivo **App.tsx** es necesario utilizar el compilador de TypeScript para compilar el archivo a JavaScript interpretable por el navegador, la URI está especificada en la línea *export const __globalAPI__: string =* "***URI de capa de servicios***" del archivo **App.tsx**.

Si no interesa compilarlo con TypeScript y se decide cambiar la URI de forma más directa, cambiar la línea *exports.__globalAPI__ = "**URI de capa de servicios**"* en el archivo **App.js**.

A.2.2. Cambiar la URI de la base de datos

En caso de utilizar otra base de datos MongoDB localizada en otra URI es necesario modificar el archivo *db.js* de la capa de servicios, en concreto la línea *var url = **URI a la base de datos***.

A.2.3. Versiones utilizadas en el desarrollo

En caso de necesitar modificar el código de la aplicación o actualizar los entornos indicar que este proyecto se ha realizado bajo las siguientes versiones de entornos y lenguajes:

- **Node.js** *(Versión 9.11.1)*
- **NPM** *(Versión 6.1.0)*
- **Docker** *(Versión 18.03.1-ce build 9ee9f40)*
- **TypeScript** (*Versión 2.9.2*)
- **React** (*Versión 16.3*)
- **JavaScript** (*ES2017*)
- **Navegadores** (*Firefox v60 y Google Chrome v64*)

En caso de utilizar versiones superiores o inferiores no se garantiza que el código pueda funcionar debido a la eliminación de funcionalidades o cambios en las APIs de los entornos, se recomienda utilizar estas versiones para maximizar la compatibilidad.

En referencia a los navegadores, se ha probado la aplicación en Firefox v60 y Google Chrome v64 puesto que son actualmente los navegadores más utilizados y más estables en términos de rendimiento. Probar con versiones anteriores no garantiza que el software funcione correctamente debido a faltas de funcionalidades de JavaScript o CSS.

Apéndice B

Manual de usuario

En esta sección se adjunta el manual de usuario de la aplicación, en el se indican las opciones que contiene y el modo de usarlas así como sus limitaciones.

B.1. Vista normal de la aplicación

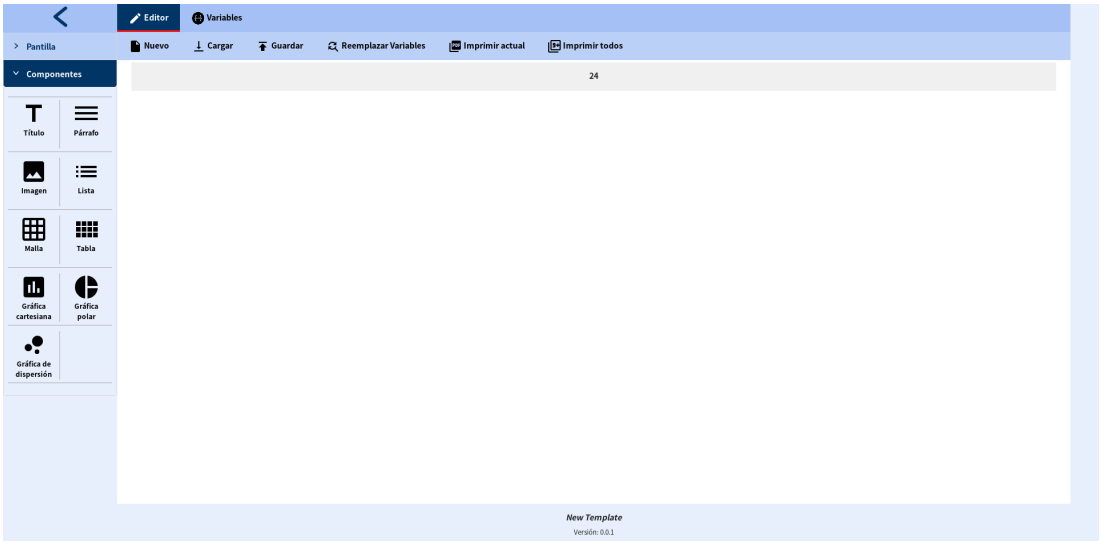

Figura B.1: Pantalla principal de la aplicación

Cuando se accede a la aplicación se muestra la interfaz de la figura B.1, la aplicación por defecto crea una plantilla vacía cuyo nombre y versión es *New Template* y *0.0.1* respectivamente, la plantilla está en blanco y solo contendrá una fila vacía para crear contenido.

B.2. Crear nuevos componentes

Para crear componentes nuevos se utiliza el panel situado en la parte izquierda de la aplicación, desplegado bajo la pestaña de **Componentes**, dicho panel se puede observar en la figura B.2.

Figura B.2: Panel de componentes de la aplicación

Cada componente que se puede crear en la plantilla está situado en este panel y cada uno está identificado por un icono y su respectivo nombre.

Para crear componentes nuevos se utiliza *Drag and Drop*, es decir basta con situar el cursor encima del icono del componente que se quiere crear, posteriormente se mantendrá pulsado el botón izquierdo del ratón y se moverá el cursor hasta una fila que tenga espacio disponible, una vez situado el cursor en la fila se puede proceder a soltar para que el componente arrastrado se cree con contenido por defecto en la posición indicada.

B.3. Cambiar las propiedades de la plantilla

Para cambiar las propiedades de la plantilla, esto es, el nombre y la versión de la misma que se muestran en el pie de página, se utiliza el panel desplegable **Plantilla** situado en el panel izquierdo de la aplicación, dicho panel se puede observar en la figura B.3.

Figura B.3: Panel de propiedades de la plantilla

En el campo **Nombre** se indica el nombre de la plantilla, mientras que en el campo **Versión** se indica la versión de la plantilla.

El usuario puede editar cualquiera de los dos campos a voluntad, no se aplican restricciones ni a los caracteres ni a los formatos que se pueden introducir en ambos campos, cuando el usuario edita cualquiera de los dos campos los cambios son guardados automáticamente y estos son reflejados automáticamente cambiando el pie de página.

B.4. Propiedades, tamaño y posición de un componente

Para editar las propiedades de un componente cualquiera, este debe de ser seleccionado previamente por el usuario, esta selección se realiza pulsando con el botón izquierdo del ratón sobre el componente que se quiera editar. Cuando esta acción sea realizada el panel derecho cambiará de contenido o se mostrará en caso de que esté oculto.

En la figura B.4 se puede observar el panel de edición de un componente de tipo *Título*.

Figura B.4: Panel de edición de un título

El panel está a su vez dividido en tres paneles plegables, el panel de **Propie-**

dades de la fila es común a todos los componentes y permite borrar la fila en la que el componente está situado, esto incluye todos los otros componentes que compartan fila con este componente.

El panel de **Tamaño y posición** también es común a cualquier componente y permite editar el tamaño que ocupa el componente a lo largo de la pantalla junto con la definición de su desplazamiento. El tamaño máximo que puede ocupar un componente es de 24 columnas. Cuando se reduce el tamaño que ocupa un componente se deja espacio libre en la fila que está ocupando, este espacio se indica con la aparición de una celda gris con un número de 1 a 24 que indica el espacio disponible.

Si existe espacio disponible, se puede aumentar el desplazamiento para desplazar el componente de izquierda a derecha en la fila en la que se sitúe, o puede optarse por crear otro componente nuevo en el espacio libre ocupándolo por completo.

El panel **Propiedades del componente** varía de un componente a otro y es en este panel donde se indican las propiedades específicas de un componente cualquiera.

B.4.1. Propiedades de componentes de texto

Los componentes de texto contemplados son **Título** y **Párrafo**, estos componentes tienen como único objetivo contener texto introducido por el usuario, el panel del título se puede observar en la figura anteriormente mencionada B.4, mientras que el panel de edición de un párrafo se puede observar en la figura B.5.

Figura B.5: Panel de edición de un párrafo

Ambos permiten editar la posición del texto, con las opciones tradicionales de cualquier editor de texto, adicionalmente se permite editar la decoración del texto con tres opciones posibles, letra en negrita (representado por una **B**), letra cursiva (representado por una *I*), y subrayado (representado por una U). Cualquiera de las opciones anteriores se puede activar simplemente haciendo click en ellas con el botón izquierdo del ratón, las opciones de decoración de texto se pueden combinar mientras que las opciones de posición solo permiten una a la vez.

Adicionalmente el componente **Título** permite editar el nivel de encabezamiento, este se cambia utilizando el *slider* que permite alterar entre los distintos niveles, desde H1 a H6.

El contenido del componente de texto se puede editar directamente en la entrada de formulario habilitada en el panel de edición, los cambios se verán reflejados según el usuario edita el texto de la entrada del formulario.

B.4.2. Propiedades del componente imagen

El componente imagen reflejado en la figura B.6 contiene simplemente dos opciones, una entrada de formulario numérica que permite definir el tamaño de la altura de la imagen, los valores de este formulario están restringidos a números y no puede tomar valores negativos.

Las imágenes son cargadas desde el disco duro del usuario, para ello se utiliza el botón de **Cargar imagen** que desplegará una ventana dependiente del sistema operativo que permitirá la selección del archivo en cuestión que el usuario seleccione. La carga de imágenes no restringe el formato del archivo, pero si se selecciona una archivo con formato inválido la imagen no se cargará correctamente y no se mostrará.

Figura B.6: Panel de edición de una imagen

B.4.3. Propiedades del componente lista

El componete lista permite la creación de múltiples elementos ordenados o no ordenados, el panel de edición de este componente se muestra en la figura B.7.

En este panel se reflejan las mismas propiedades que en un componente de texto, salvo que se añade una opción adicional que permite definir la lista como ordenada, es decir, el encabezamiento de los elementos pasará a ser un número o letra en vez de un símbolo.

Figura B.7: Panel de edición de una lista

El panel proporciona un *slider* que permite definir el elemento de encabezamiento de un ítem de la lista, el usuario podrá elegir entre los disponibles,

contando con distintos encabezamientos para las listas ordenadas y para las no ordenadas.

Si se desea añadir un elemento nuevo basta con introducir el contenido del elemento en la entrada de formulario inferior y pulsar *Enter* para crear el elemento, al realizar esta acción el elemento será mostrado en la lista y se creará una entrada de formulario para ese elemento en la sección **Editar elementos actuales**.

En la sección **Editar elementos actuales** se muestra una entrada de formulario por cada elemento presente en la lista, modificando la entrada del formulario aplica cambios directamente al componente de la lista. Para eliminar el elemento pulsar en el botón rojo situado al final del formulario.

Las opciones de edición anteriores solo están disponibles cuando la lista es estática, es decir, su contenido no se obtiene desde una fuente de datos.

B.4.4. Propiedades del componente malla

Una malla permite la creación de celdas independientes con dimensiones personalizadas por el usuario, el panel de edición de este componente solo cuenta con un botón para añadir más celdas a la malla con dimensiones predefinidas, dichas celdas solo contienen componentes de texto simples.

Para editar una celda situar el cursor encima de ella y pulsar en el botón de edición que se muestra, posteriormente se activará el panel de edición de la celda que se ha seleccionado mostrado en la figura B.8.

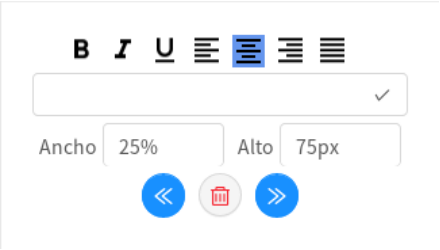

Figura B.8: Panel de edición de una celda de una malla

El panel permite editar el contenido de la celda y las propiedades del texto, adicionalmente se adjuntan entradas de formulario para editar el ancho y alto que ocupa la celda, el alto se mide el pixeles y solo puede tomar valores superiores a 10 pixeles, el ancho se mide en % y toma valores superiores al 10 % y como máximo 100 %. Adicionalmente se adjuntan dos botones que permiten desplazar la celda de izquierda a derecha en la malla o borrarla. Para confirmar los cambios

de edición se debe de pulsar en el botón de *check* situado al final de la entrada del formulario de edición.

B.4.5. Propiedades del componente tabla

El panel de edición de una **tabla estática** se corresponde con la figura B.9, en este panel se permite al usuario añadir filas vacías a la tabla y crear columnas del mismo modo que un elemento de una lista, introduciendo el nombre de la columna en la entrada del formulario y pulsando *enter*.

Figura B.9: Panel de edición de una tabla estática

Cuando se añade una columna o fila a la tabla desde este panel, este se verá reflejado inmediatamente en la tabla vinculada, para editar el contenido de la fila o columna se sigue el mismo procedimiento que en una celda de una malla, se sitúa el cursor encima de la columna o fila pertinente y se selecciona la opción de editar, posteriormente se introduce el valor deseado en la entrada del formulario que se habilita para la edición y se pulsa *enter* o el botón de *check* para finalizar la edición.

Adicionalmente se permite al usuario cambiar el orden de las filas y columnas que añada a la tabla mediante el uso de las flechas de dirección que se habilitan cuando se sitúa el cursor encima de una fila o columna.

Las opciones de edición anteriores solo están disponibles cuando la tabla es estática, es decir su contenido no es cargado desde una fuente de datos.

B.4.6. Propiedades del componente gráfica cartesiana

El componente **gráfica cartesiana** permite representar gráficas de líneas, barras y áreas, todas ellas combinadas según el usuario defina en las series del componente, el panel de este componente se refleja en la figura B.10.

Figura B.10: Panel de edición de una gráfica cartesiana

En este panel se provee al usuario de un conjunto de opciones que permiten

activar o desactivar elementos visuales de la gráfica, como la malla cartesiana, ejes de coordenadas o la leyenda, estas opciones se encuentran en la sección de **Propiedades de la gráfica**. Adicionalmente se permite editar el tamaño de la altura de la gráfica en la entrada numérica del formulario de la sección **Altura de la gráfica**.

Si se quiere añadir nuevas series de datos basta con introducir el nombre de la serie en la entrada del formulario de la sección **Añadir nuevas series de datos** y pulsas *enter* para crear la serie, al hacer esto se crea un botón en la sección **Editar conjuntos de datos** con el nombre de la serie que se acaba de introducir.

Para editar el contenido de una serie basta con pulsar el botón de *Editar serie <nombre de la serie>* situada en la sección **Editar conjuntos de datos**, al hacer esto se desplegará el menú de edición de esa serie, este panel está compuesto por una entrada de formulario donde se introducen los datos de la serie.

Los datos de la serie deben de estar introducidos entre los corchetes y separados por comas, solo admitiéndose números para componer los valores de la serie. Un ejemplo de valores de una serie sería *[1,2,3,4,5,6,7,8,9]*.

Adicionalmente el panel de edición permite definir el tipo de gráfica que representarán los datos además del color de la serie.

Los tipos de gráficas disponibles son : *línea*, *área* y *barras*. Los colores deben de ser introducidos en formato hexadecimal RGB, un ejemplo de color válido sería **#ff0000**.

Adicionalmente la gráfica cartesiana permite editar las etiquetas del eje X, introduciendo cadenas de texto a voluntad del usuario. Para editarlas pulsar sobre el botón **Editar etiquetas del eje X**, e introducir en los corchetes las etiquetas separadas por comas y entre comillas dobles, un ejemplo válido de etiquetas sería *["Etiqueta 1", "Etiqueta2"]*.

B.4.7. Propiedades del componente gráfica polar

El componente **gráfica polar** permite representar gráficas de sectores o de radar, el panel de edición de este componente está reflejado en la figura B.11 para una gráfica de sectores y en la figura B.12 para una gráfica de radar.

Figura B.11: Panel de edición de una gráfica de sectores

Ambos paneles funcionan del mismo modo que una gráfica cartesiana, permitiendo el añadido de series de la misma manera y permitiendo la personalización de opciones visuales en la misma localización que las gráficas cartesianas, además de permitir cambiar el tamaño de la altura de la gráfica.

El panel polar proporciona además de lo anterior la opción de seleccionar que tipo de gráfica se desea representar, siendo los posibles valores *sectores* o *radar*.

Cabe destacar que existen diferencias en como se tratan los datos a nivel interno en cada uno de los dos tipos de gráficas disponibles. Las gráficas de sectores calculan su valor de representación como ***el sumatorio de los valores de la serie*** de modo que la serie *[5,5]* es lo mismo que *[10]* en una gráfica de sectores.

Sin embargo en una gráfica de radar, los valores de representación se calculan como cualquier otra gráfica de puntos, como una lista de valores, de modo que *[5,5]* no es equivalente a *[10]* en una gráfica de sectores.

Figura B.12: Panel de edición de una gráfica de radar

Adicionalmente la gráfica de sectores permite editar las etiquetas de los vértices del radar de representación, mientras que las gráficas de sectores no permiten este tipo de opción. Adicionalmente la gráficas polares permiten editar la opacidad del color de las series.

B.4.8. Propiedades del componete gráfica de dispersión

El componente **gráfica de dispersión** permite representar gráficas de burbujas o de dispersión en dos variables, el panel de edición está representado en la figura B.13.

El panel de edición de una gráfica de dispersión sigue los mismos procedimientos que una gráfica cartesiana, contando con opciones de personalización visuales de la gráfica y de edición y añadido de series junto el tamaño de la gráfica.

La diferencia radica en que la gráfica de dispersión permite editar las etiquetas del eje X y del eje Y pulsando en los botones de *Editar etiquetas del eje X* y *Editar etiquetas del eje Y* respectivamente, el formato de las etiquetas es el mismo que el de las gráficas cartesianas, introduciendo las etiquetas entre comillas dobles y separadas por comas.

Adicionalmente las series de datos de una gráfica de dispersión cuentan con dos conjuntos de datos por serie, el primer conjunto de datos representa los valores que toma el punto en el eje X mientras que el segundo conjunto de datos representa los valores que toma el punto en el eje Y.

Figura B.13: Panel de edición de una gráfica de dispersión

B.5. Definición de variables

La aplicación posee opciones para declarar e insertar variables en componentes existentes en la plantilla, el contenido de una variable puede ser una cadena de texto, un número o un conjunto de datos, para configurar las variables que se pueden utilizar en la plantilla que se esté editando basta con acceder a la sección de **Variables** en la barra de navegación. La sección de variables se subdivide en secciones más pequeñas, cada una de ellas con una funcionalidad concreta, dichas secciones se explican a continuación.

B.5.1. Sección de creación de variables

La primera sección es la sección de creación de variables, la cual se encuentra bajo el título de **Variables**, dicha sección se puede observar en la figura B.14.

Figura B.14: Panel de creación de variables

En esta sección se muestran las variables que hay definidas a nivel de plantilla en una tabla, en dicha tabla se indica el nombre de la variable el tipo de variable y se permite la opción de borrar la variable. Adicionalmente se permite crear variables nuevas introduciendo el nombre de la variable en la entrada del formulario inferior y seleccionando el tipo que la variable ocupará durante su estancia.

B.5.2. Tipos de variables

Las variables que se definen a nivel de plantilla tienen un tipo asignado, dependiendo del tipo su uso estará permitido en unos componentes y prohibido en otros, adicionalmente el tipo define las opciones de traducción de la variable de modo que es crucial definir el tipo adecuadamente o esta podría no mostrarse en los componentes.

Los tipos de variables disponibles son:

- **Cadena de texto**: El contenido de la variable es una cadena de texto y se traducirá literalmente su valor, de modo que se mostrará exactamente el valor obtenido de la ruta del JSON especificada.

- **Número**: El contenido de la variable es un número, se traducirá literalmente su valor pero esta traducción fallará si existen letras o si el número está entre comillas como una cadena texto.

- **Fecha y hora**: El contenido de la variable es una fecha y una hora, la traducción de este tipo de variable se realiza tomando los milisegundos desde el *epoch* de los sistemas UNIX, de modo que el valor recibido será un número que represente estos milisegundos, en caso contrario se mostrará *Invalid Date* al traducir la fecha.

- **Fecha (DD/MM/AAAA)**: El contenido de la variable es una fecha en formato día mes y año, la traducción de este tipo de variable se realiza tomando los milisegundos desde el *epoch* de los sistemas UNIX, de modo que el valor recibido será un número que represente estos milisegundos, en caso contrario se mostrará *Invalid Date* al traducir la fecha.

- **Fecha (MM/DD/AAAA)**: El contenido de la variable es una fecha en formato mes día y año, la traducción de este tipo de variable se realiza tomando los milisegundos desde el *epoch* de los sistemas UNIX, de modo que el valor recibido será un número que represente estos milisegundos, en caso contrario se mostrará *Invalid Date* al traducir la fecha.

- **F. Datos (Lista)**: El contenido de la variable es una fuente de datos para un componente tipo lista, este tipo de variable espera un *array* de objetos de los cuales solo mostrará una propiedad que el usuario seleccione en el *Mapeador de variables*.

- **F. Datos (Tabla)**: El contenido de la variable es una fuente de datos para un componente tipo tabla, este tipo de variable espera un *array* de objetos, la tabla creará una columna por cada propiedad del objeto genérico del *array* y una fila por cada objeto presente en el *array*.

- **F. Datos (Cartesiana)**: El contenido de la variable es una fuente de datos para un componente gráfica cartesiana, este tipo de variable espera un *array* de objetos, la gráfica cartesiana creará una serie por cada propiedad presente en el objeto genérico del *array*, los valores de la serie serán los valores de dicha propiedad en todos los objetos que componen el *array* recibido como parámetro.

- **F. Datos (Polar)**: El contenido de la variable es una fuente de datos para un componente gráfica polar, este tipo de variable espera un *array*

de objetos, la gráfica polar creará una serie por cada propiedad presente en el objeto genérico del *array*, los valores de la serie serán los valores de dicha propiedad en todos los objetos que componen el *array* recibido como parámetro.

- **F. Datos (Dispersión)**: El contenido de la variable es una fuente de datos para un componente gráfica de dispersión, este tipo de variable espera un mapa de *arrays* de objetos, la gráfica de dispersión creará una serie por cada clave del mapa, los valores de cada serie son los *arrays* de la claves, dichos *arrays* deben de contener objetos que tengan dos propiedades, **x** para definir el valor del punto en el eje x e **y** para definir el valor del punto en el eje y.

B.5.3. Mapeador de variables

Figura B.15: Mapeador de variables

La siguiente sección es el mapeador de variables, este se encuentra bajo el título **Mapeo de variables**, una posible representación del panel se puede ob-

servar en la figura B.15.

Cada vez que se define una variable nueva se crea un panel plegable en esta sección, haciendo click en el panel despliega las opciones de mapeo de la variables seleccionada, estas opciones varían dependiendo del tipo que la variable seleccionada tenga asociado, los posibles valores de mapeo son los siguientes:

- **Ruta**: Especifica la ruta del JSON con la que se asociará la variable, esta ruta es una cadena de texto que representa dicha ruta, se incluye implícito el puntero *this* para referenciar el objeto padre que engloba el JSON, por ejemplo un valor válido sería *people.0.name*.
- **Propiedad a mostrar en la lista (solo lista)**: Especifica cual de las propiedades del objeto genérico del **array** al que apunte la variable será mostrada en la lista, el valor es una cadena de texto que se corresponde con una de las propiedades del objeto genérico, por ejemplo si el objeto genérico de la lista es del tipo `{name: "Foo", age: 42}` pasa mostrar en una lista los nombres escribiríamos en este campo *name*.
- **Nombre de las columnas (solo tablas)**: Especifica cual sería el título de las columnas de la tabla que cargará esta fuente de datos, este parámetro es opcional de modo que si se deja en blanco el título de la columna será el nombre de la propiedad del objeto genérico que reciba la tabla, los valores que se especifiquen deben de ir entre comillas dobles y separados por comas, además todos estos valores deben de estar englobados dentro de `[]`. Un valor válido para este campo sería `["Nombre","Edad"]`.
- **Colores de las series (solo gráficas)**: Especifica cual es el color de cada serie que se añadirá a la gráfica, el parámetro es opcional, si no se especifica se toma siempre el mismo color por defecto, los colores deben de estar especificados en formato hexadecimal RGB, deben de ir entre comillas dobles y estar separados por comas, además todos estos valores deben de ir englobados dentro de un `[]`. Un valor válido para este campo sería `["#ff0000","#f0a2ff","#00ffee"]`.
- **Tipos de las series (solo gráfica cartesiana)**: Especifica el tipo de cada serie de la fuente de datos de la variable, no especificar el tipo de una serie provoca que el valor por defecto sea línea. Los tipos deben de especificarse entre comillas y separados por comas, adicionalmente es necesario englobarlos todos dentro de `[]`, los tipos disponibles son *line*, *area* y *bar*. Un valor válido para este campo sería `["line","area","bar","area"]`
- **Propiedad de las etiquetas del eje X (solo gráfica cartesiana)**: Especifica que propiedad del objeto genérico de la fuente de datos será seleccionada para cubrir las etiquetas del eje X de la gráfica, por ejemplo si el objeto

genérico de la fuente de datos es del tipo `{s1: 5, s2: 10, date: "12-05-10"}` y se quiere usar *date* para indicar las etiquetas del eje X se introduciría en este campo el valor *date*.

- **Opacidad de las series (solo gráfica polar)**: Especifica la opacidad de cada serie de la fuente de datos de la variable, no especificar este campo provoca que todas las series de la gráfica polar tengan opacidad 1, los valores que admite son números entre 0 y 1, estos valores deben de ir separados por comas y entre `[]`. Un valor válido para este campo sería `[0.6,0.3,0.2,1]`.
- **Propiedad de las etiquetas de los ángulos (solo gráfica polar)**: Especifica que propiedad del objeto genérico de la fuente de datos será seleccionado para cubrir las etiquetas de los ejes de la gráfica de tipo radar, por ejemplo si el objeto genérico de la fuente de datos es del tipo `{dh: 15, dv: 10, dy: 28, subject: "Math"}` y se quiere usar *subject* para representar las etiquetas de los vértices el valor del campo sería *subject*.
- **Unidad del eje X (solo gráfica de dispersión)**: Especifica el nombre de la unidad que se corresponde con el eje X, dicho valor es una cadena de texto introducida por el propio usuario. Un valor válido para este campo sería *kg*.
- **Unidad del eje Y (solo gráfica de dispersión)**: Especifica el nombre de la unidad que se corresponde con el eje Y, dicho valor es una cadena de texto introducida por el propio usuario. Un valor válido para este campo sería *cm*.

B.5.4. Cargador de JSON

La última sección es el cargador de json, que está bajo el título de **JSON** en la sección de variables, esta sección se ve reflejada en la figura B.16.

Figura B.16: Cargador de JSON

En esta parte se permite cargar cualquier archivo en formato JSON en la aplicación, la carga se realiza pulsando en el botón de *Cargar JSON* y seleccionando

el archivo o los archivos en formato JSON.

Una vez cargado el archivo este aparecerá a la derecha del cuadro de texto, el usuario podrá seleccionar cual quiere que se use actualmente en la aplicación pulsando en el botón asignado para cada JSON, o en caso de no necesitar usar ninguno pulsar sobre el botón *ninguno*.

B.6. Uso de variables y fuentes de datos

Las variables declaradas en la sección de variables pueden utilizarse en los componentes que se inserten en la plantilla, sin embargo existen ciertas limitaciones sobre donde usar cada variable que se defina por parte del usuario.

Las variables que tengan los tipos *cadena de texto*, *número* y *fecha* pueden insertarse en cualquier componente de texto como contenido usando la sintaxis `{{nombre de la variable}}`.

Para cargar una fuente de datos es necesario utilizar un componente con contenido dinámico, los componentes que soportan contenido dinámico cuentan con una opción llamada *Dinámico* en el panel de edición. ***Activar o desactivar esta opción restablece el componente a sus valores por defecto por lo que debe de tenerse especial cuidado si se quiere cambiar una componente estático a dinámico ya que el contenido que se borra no se puede recuperar***.

Activar la opción de *Dinámico* cambia el formato del panel de edición, de modo que muestra una entrada de formulario donde aparecen las posibles variables que representan una fuente de datos compatible con el tipo de componente que se está editando. El usuario puede seleccionar una de estas variables y pulsar sobre el botón de **Cargar fuente de datos**, esto provocará que la aplicación obtenga la fuente de datos de la ruta de json especificada en la variable asignada e intente traducirla para representarla en el componente.

Para reemplazar todas las variables embebidas en componentes de texto y cargar todas las fuentes de datos a la vez basta con utilizar el botón de *Reemplazar variables* de la barra de operaciones, este botón activará el reemplazo de variables en todos los componentes hasta que el usuario decida desactivarla.

B.7. Creación, guardado y borrado de plantillas

Las plantillas que se generen pueden ser guardadas y borradas cuando el usuario lo solicite, este tipo de operaciones se realizan mediante el uso de los botones de la barra de operaciones CRUD situada debajo de la barra de navegación de secciones.

Para crear una plantilla nueva se utiliza el botón de **Nuevo**, esto creará un cuadro de diálogo adicional donde el usuario podrá introducir el nombre y versión de la nueva plantilla a crear, esto reemplazará la vista actual por una plantilla nueva vacía con el nombre y versión que el usuario seleccionase, ***esta plantilla aún no está guardada en la base de datos, se guardará cuando el usuario pulse sobre el botón de guardar***.

Para guardar una plantilla se utiliza el botón de **Guardar**, cuando se guarda una plantilla esta se guarda bajo el nombre y versión especificada en el pie de página, ***este guardado reemplaza cualquier otra plantilla existente bajo ese nombre y esa versión***. Ejecutar un guardado guarda los componentes, sus propiedades y disposición junto con las variables y las propiedades de mapeo de estas, ***no se guardan las variables reemplazadas ni los JSON cargados en la aplicación***.

Para cargar una plantilla se utiliza el botón de **Cargar**, al realizar esta acción aparece una lista de las distintas plantillas que existen en la base de datos, seleccionar el botón de editar reemplaza la vista con una lista de las distintas versiones de la plantilla seleccionada si se procede a seleccionar el botón de editar, la versión seleccionada es cargada en la aplicación.

Adicionalmente en el panel anterior se proporciona una opción de borrar, seleccionar esta opción borrara una versión o todas las versiones de una plantilla seleccionada.

B.8. Impresión de plantilla

Para imprimir una plantilla se pulsa sobre el botón **Imprimir actual**, esto provocará un reemplazo de variables forzado con el JSON actual en la plantilla y generará un PDF que representa la plantilla, dicho PDF será enviado como una descarga de un archivo al usuario.

Si se utiliza la opción **Imprimir todos**, se generará un PDF por cada JSON cargado en la aplicación y se enviarán como archivos individuales al usuario.

Apéndice C

Licencia

El software y documentación asociado al mismo proporcionados en este trabajo fin de grado esta bajo licencia **MIT** [26], siendo esta la siguiente.

Se concede permiso, libre de cargos, a cualquier persona que obtenga una copia de este software y de los archivos de documentación asociados (el "Software"), para utilizar el Software sin restricción, incluyendo sin limitación los derechos a usar, copiar, modificar, fusionar, publicar, distribuir, sublicenciar, y/o vender copias del Software, y a permitir a las personas a las que se les proporcione el Software a hacer lo mismo, sujeto a las siguientes condiciones:

El aviso de copyright anterior y este aviso de permiso se incluirán en todas las copias o partes sustanciales del Software.

EL SOFTWARE SE PROPORCIONA "TAL CUAL", SIN GARANTÍA DE NINGÚN TIPO, EXPRESA O IMPLÍCITA, INCLUYENDO PERO NO LIMITADA A GARANTÍAS DE COMERCIALIZACIÓN, IDONEIDAD PARA UN PROPÓSITO PARTICULAR Y NO INFRACCIÓN. EN NINGÚN CASO LOS AUTORES O PROPIETARIOS DE LOS DERECHOS DE AUTOR SERÁN RESPONSABLES DE NINGUNA RECLAMACIÓN, DAÑOS U OTRAS RESPONSABILIDADES, YA SEA EN UNA ACCIÓN DE CONTRATO, AGRAVIO O CUALQUIER OTRO MOTIVO, DERIVADAS DE, FUERA DE O EN CONEXIÓN CON EL SOFTWARE O SU USO U OTRO TIPO DE ACCIONES EN EL SOFTWARE.

Bibliografía

[1] *The Standish Group Report: CHAOS*. Disponible en *https://www.projectsmart.co.uk/white-papers/chaos-report.pdf* [Consultado el 31/09/2018]

[2] *Vitae: Estudio salarial del sector TIC en Galicia 2015-2016*. Disponible en *https://www.scribd.com/document/288511179/Guia-Salarial-Sector-TI-Galicia-2015-2016* [Consultado el 31/09/2018]

[3] *What is Web services in simple terms*. Pregunta de Stack Overflow, respuesta del usuario *Noon Silk*. Disponible en *https://stackoverflow.com/questions/1353688/what-is-web-services-in-simple-terms* [Consultado el 31/09/2018]

[4] *Jersey RESTful Web Services in java*. Disponible en *https://jersey.github.io/* [Consultado el 31/09/2018]

[5] *JAX-RS: Java API for Restful Web Services Specification*. Disponible en *https://github.com/jax-rs/spec/blob/master/spec.pdf*. [Consultado el 31/09/2018]

[6] *Express.js Node Web Application Framework*. Disponible en *https://expressjs.com/* [Consultado el 31/09/2018]

[7] *Rocket- Simple, Fast, Type-safe Framework for Rust*. Disponible en *https://rocket.rs/* [Consultado el 31/09/2018]

[8] *The Rust programming language*. Disponible en *https://www.rust-lang.org/* [Consultado el 31/09/2018]

[9] *ECMAScript 2017 Language Specification ECMA-268*. Disponible en *https://www.ecma-international.org/ecma-262/8.0/* [Consultado el 31/09/2018]

[10] *ECMA-404 The JSON Data Interchange Standard*. Disponible en *https://www.json.org/* [Consultado el 31/09/2018]

[11] *TypeScript - JavaScript that scales.* Disponible en *https://www.typescriptlang.org/* [Consultado el 31/09/2018]

[12] *React - A JavaScript library for building user interfaces.* Disponible en *https://reactjs.org/* [Consultado el 31/09/2018]

[13] *Composite Pattern.* Artículo de la wikipedia. Disponible en *https://en.wikipedia.org/wiki/Composite_pattern* [Consultado el 30/09/2018]

[14] *Ant Design - A UI Design Language.* Disponible en *https://ant.design/* [Consultado el 30/09/2018]

[15] *Recharts.* Disponible en *http://recharts.org/* [Consultado el 30/09/2018]

[16] *Material Design.* Disponible en *https://material.io/* [Consultado el 30/09/2018]

[17] *Document-oriented database.* Artículo de la wikipedia. Disponible en *https://en.wikipedia.org/wiki/Document-oriented_database* [Consultado el 30/09/2018]

[18] *NoSQL Databases explained* por MongoDB Inc. Disponible en *https://www.mongodb.com/nosql-explained* [Consultado el 30/09/2018]

[19] *MongoDB for GIANT ideas — MongoDB.* Disponible en *https://www.mongodb.com/* [Consultado el 30/09/2018]

[20] *Mongoose, elegant mongodb object modeling for node.js.* Disponible en *http://mongoosejs.com/* [Consultado el 30/09/2018]

[21] *NPM, Node package manager.* Disponible en *https://www.npmjs.com/* [Consultado el 30/09/2018]

[22] *Docker, Build, Ship, and run Any App, Anywhere.* Disponible en *https://www.docker.com/* [Consultado el 30/09/2018]

[23] *PhantomJS - Scriptable Headless Browser.* Disponible en *http://phantomjs.org/* [Consultado el 30/09/2018]

[24] *Phantom - Fast NodeJS API for PhantomJS.* Disponible en *https://www.npmjs.com/package/phantom* [Consultado el 30/09/2018]

[25] *10 Usability Heuristics for User Interface Design* por Jakob Nielsen, 1 de enero de 1995. Disponible en *https://www.nngroup.com/articles/ten-usability-heuristics/* [Consultado el 31/09/2018]

[26] *The MIT License.* Disponible en *https://opensource.org/licenses/MIT* [Consultado el 31/09/2018]