Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO **F**IN DE **G**RADO

Grado en Ingeniería Informática
Mención en Tecnologías de la Información

Diseño e implantación de un entorno web con NodeJS

Autor:
D. José Miguel Fernández Sáenz de Navarrete

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO **F**IN DE **G**RADO

Grado en Ingeniería Informática
Mención en Tecnologías de la Información

Diseño e implantación de un entorno web con NodeJS

Autor:
D. José Miguel Fernández Sáenz de Navarrete
Tutores:
Dr. Joaquín Adiego Rodríguez
Dra. Natalia Martín Cruz

Índice general

1 Introducción

1.1 Objetivos del proyecto

En este apartado se detallaran los objetivos principales que se han establecido para este proyecto:

- Mejorar el rendimiento y la usabilidad ya que la anterior plataforma estaba en desuso debido a su complejidad y bajo rendimiento
- Desarrollo de una interfaz gráfica moderna y eficiente que se adapte a las tecnologías actuales y pueda visualizar se forma correcta en dispositivos de escritorio y móviles
- Desarrollo de una interfaz a través de la cual se puedan extraer gráficos a partir de los datos de los proyectos que serán seleccionados por el usuario para su posterior análisis.
- Desarrollo de una interfaz que permita la insercción y edición de los datos de los proyectos
- Desarrollo de una interfaz que permita la insercción y edición de los datos de los usuarios
- Realizar la migración de los datos del proyecto antiguo al nuevo, adaptando las clases al nuevo sistema de persistencia
- Compatibilidad de la plataforma web con los navegadores actuales más utilizados
- Seguimiento y documentación técnica del desarrollo de todo el proyecto

1.2 Estructura de la memoria

La estructura de la memoria se basa en los siguientes pilares.

- **Introducción**, en este apartado se describirán de forma breve las motivaciones de la realización del TFG, además de una breve descripción de los objetivos generales y una comparativa entre soluciones. Este apartado está formado por:
  - Estructura de la memoria
  - Descripción del escenario
  - Análisis de la solución existente
  - Propuesta de solución alternativa
- **Tecnologías utilizadas**, se describirán las herramientas y tecnologías utilizadas, así como porqué se han decidido utilizar dichas herramientas. Además, se compararán con las herramientas habituales de desarrollo web
  - React

  - GraphQL
  - Node.JS
  - MongoDB
  - Npm

- **Planteamiento de la solución**, se describirán las funcionalidades que debe tener la aplicación, empezando por el análisis de requisitos, los casos de uso y los diferentes diagramas de la aplicación. Este apartado está formado por:
  - Modelo de casos de uso
  - Análisis de requisitos
  - Diagrama de clases
  - Arquitectura de la solución

- **Planificación**, se describirá el método empleado para la planificación del proyecto y como este se llevó a cabo.
  - Introducción
  - Product Backlog
  - Sprint Backlog

- **Ejecución del proyecto**, se detallará la estructura del proyecto, como se organizó la información y como se desarrolló.
  - Estructura del proyecto
  - Organización de la información
  - Migración de la BBDD
  - Desarrollo de la solución

- **Conclusiones y trabajo futuro**

- **Bibliografía**

1.3 Descripción del escenario

El objetivo principal de este trabajo es reconstruir una plataforma web a través de la cual se puedan analizar datos relacionados con la cooperación internacional para el desarrollo. Esta reconstrucción se ha llevado a cabo dado que la anterior plataforma ya ha quedado obsoleta y los usuarios actuales de la misma buscaban una modernización y simplificación de la misma. El proyecto anterior lo realizó Antonio Fuentes Pérez como proyecto de fin de carrera con el título ”Plataforma UVa-AECID: Eficiencia en la Cooperación Internacional para el Desarrollo”

Este trabajo comenzó a realizarse en Octubre del año 2018, los tutores que orientaron y revisaron este trabajo son:

- **Joaquín Adiego Rodríguez**, doctor perteneciente al Departamento de Informática encargado de la Arquitectura y Tecnología de Computadores, Ciencias de la Computación e Inteligencia Artificial en la Escuela de Ingeniería Informática de Valladolid.
- **Natalia Martín Cruz**, profesora perteneciente al Departamento de Organización de Empresas y Comercialización e Investigación de Mercados en la Facultad de Ciencias Económicas y Empresariales de la Universidad de Valladolid.

La principal motivación para este trabajo es que actualmente estoy aprendiendo a desarrollar webs con *frameworks* de Javascript, esto es debido a que durante estos últimos 8 años este lenguaje está en la vanguardia del desarrollo web por su versatilidad, al poderlo utilizar tanto en la parte del cliente como en la del servidor, además tras el desarrollo de los *frameworks* más usados hay grandes empresas y usuarios que velan por su desarrollo de forma que actualmente la mayoría de navegadores web utilizados soportan su ejecución.

1.4 Análisis de la solución existente

La aplicación en la que se basa este trabajo se desarrolló en el año 2009, su arquitectura está basada en el modelo Cliente-Servidor como la mayoría de las plataformas web. La aplicación tiene un perfil multitarea y concurrente de forma que puede atender varias peticiones a la vez, los nodos con los que cuenta son los siguientes:

- Cliente, puede ser cualquier dispositivo que disponga de un navegador web, no requiere un hardware específico, pero sí un mínimo que a día de hoy cualquier dispositivo posee.
- Servidor, se trata de una máquina física o virtual con el sistema operativo Ubuntu Server, dispone del servidor web Http Apache2 para atender las peticiones web y el sistema gestor de base de datos MySQL.

Según la arquitectura descrita el diagrama de despliegue de la aplicación es el siguiente:

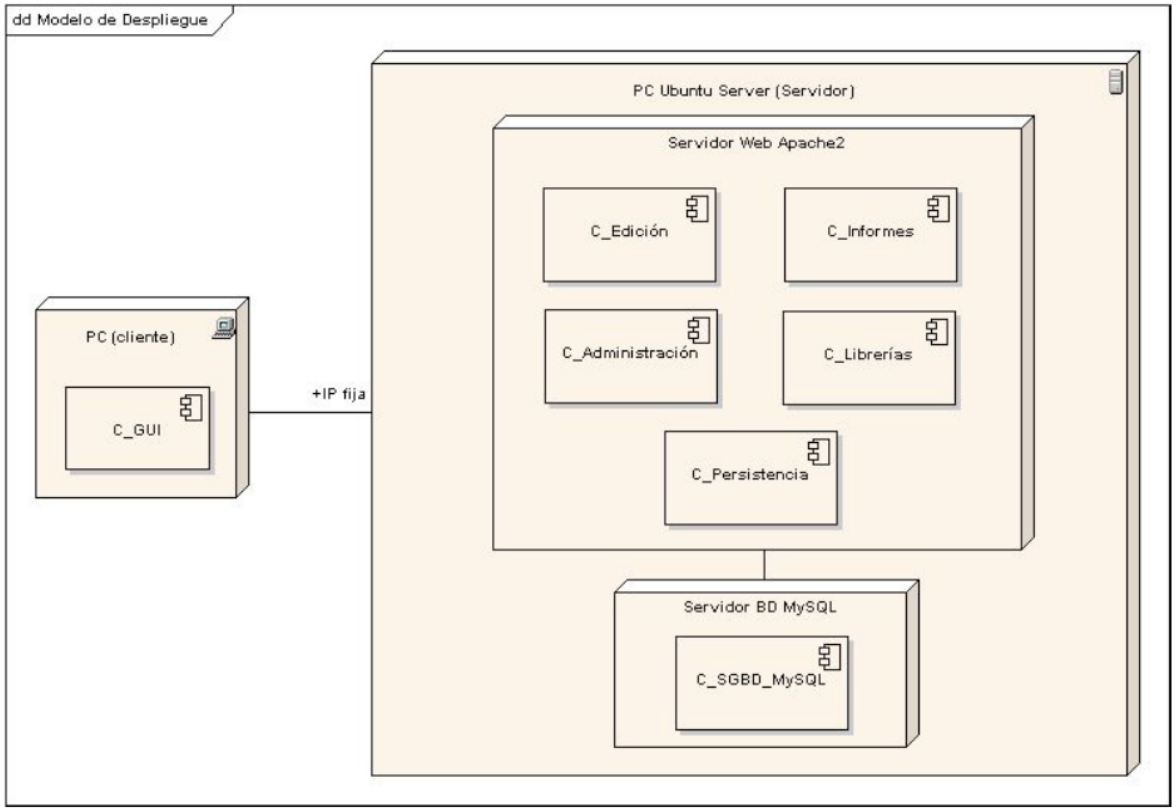


Figura 1.1: Diagrama de despliegue. Pérez [2009]

Como se puede observar en la figura anterior, a través del servidor web Apache y del servidor de MySQL se maneja toda la lógica y persistencia de datos. En la aplicación rediseñada se aplicará un enfoque más distributivo de forma que se intentarán minimizar los riesgos que existen en la arquitectura actual.

El riesgo más inminente que posee esta arquitectura es que no existe ningún tipo de redundancia en la persistencia de datos, de forma que si el servidor se viera comprometido podrían perderse todos los datos almacenados.

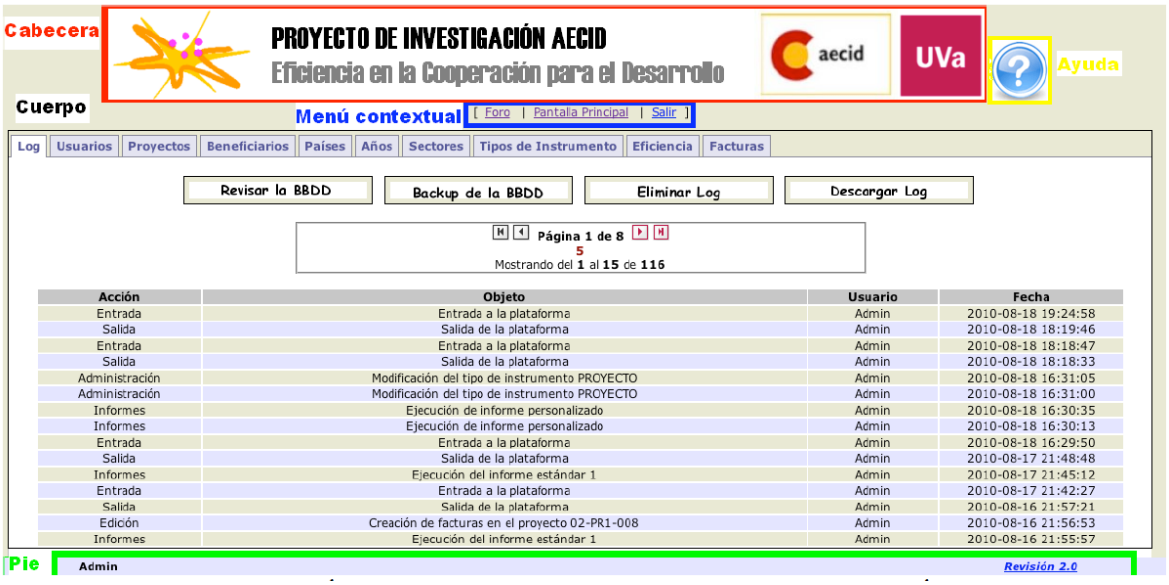
1.5 Propuesta de solución alternativa

En la solución propuesta se cambia parcialmente el enfoque de la arquitectura, ya que al basarlo en tecnologías actuales el enfoque Cliente-Servidor no es suficiente para dividir la complejidad de la solución. Esto es debido a que en la solución existente toda la lógica recae sobre el servidor, y el cliente únicamente actúa como una mera interfaz gráfica que expone los datos al usuario. Si llevamos este enfoque al modelo de capas, podríamos considerar que en el cliente se encuentra una capa de presentación sin lógica y el servidor absorbe toda la lógica y persistencia de datos.

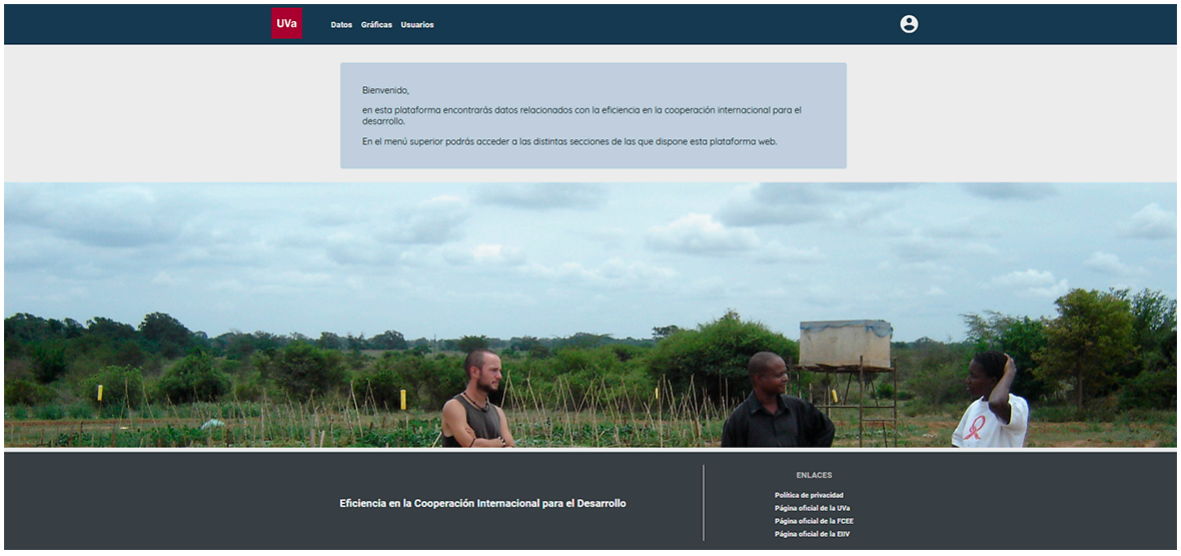
En la aplicación que se presenta aquí se va a proceder a dividir las responsabilidades entre diferentes actores, de esta forma el sistema adquirirá una mayor confiabilidad, flexibilidad y capacidad de crecimiento. Cabe destacar que con las tecnologías actuales se está creando una tendencia en la cual se invierte la responsabilidad de cada actor, de forma que los clientes cada vez obtienen más independencia del servidor.

Los objetivos fundamentales que se pretenden alcanzar con el desarrollo de esta plataforma son los siguientes:

- Simplificación de interfaces, la aplicación existente posee una lógica demasiado compleja que se traslada a la interfaz del usuario, por ello en la solución propuesta se abstraerá la lógica de forma que el usuario utilice la aplicación de una forma más intuitiva y amigable. En las figuras 1.2a y 1.2b puede observarse el cambio.
- Mejora de rendimiento, actualmente toda la persistencia de datos se encuentra en formato SQL, tras revisar el modelo y el tipo de consultas realizadas se ha tomado la decisión de migrarlo a un sistema NoSQL. De esta forma se simplificarán las consultas y se mejorará el rendimiento de la plataforma.
- Modernización de la infraestructura, con el avance que tiene el desarrollo web hoy en día es inevitable la necesidad de actualizar las plataformas, en esta área se han producido grandes avances sobre todo enfocados en el lenguaje JavaScript ya que ahora es posible codificar prácticamente toda la plataforma bajo un mismo lenguaje de programación. En apartados posteriores se mostrarán las ventajas e inconvenientes de este enfoque.



(a) Interfaz antigua. Pérez [2009]



(b) Interfaz nueva

Figura 1.2: Comparativa de interfaces

2 Tecnologías utilizadas

2.1 React

React es una biblioteca Javascript que permite crear interfaces de usuario interactivas basadas en componentes reutilizables.

Fue **creada por Facebook** en 2011, su creación se produjo como necesidad de resolver una serie de problemas que existían en la red social, entre ellos una mejora del rendimiento, ya que, hasta ese momento la cantidad de conexiones que se debían realizar entre las vistas y los datos era excesiva ralentizando el rendimiento de la red social.

Además del rendimiento esta herramienta facilitaba enormemente el desarrollo ágil de componentes de interfaces de usuario reutilizables, por lo que se ha convertido en uno de los frameworks **open-source** con más proyección de futuro hoy en día. En 2013, su código fue liberado, convirtiéndose en una de las librerías más utilizadas por empresas entre las que se encuentran Instagram, Airbnb, Netflix o Dropbox.

Los componentes de React implementan el método de **renderizado, o render**, que toma una serie de parámetros de entrada y genera la salida correspondiente en el navegador. Además de la propia lógica del componente, el resultado del render depende, fundamentalmente, de dos tipos de valores:

- **Propiedades**: Conocidas como **props**. Son aquellos valores que se proporcionan al componente en el momento de su creación y cuyo valor es utilizado para el posterior renderizado del mismo. Su valor es inmutable a lo largo de todo el ciclo de vida del componente.

- **Estado**: Conocido como **state**. Son aquellos valores que representan el estado interno de un componente en un momento determinado del ciclo de vida. Un cambio en el estado del componente provocará necesariamente un renderizado del mismo.

Dentro del proceso de carga de una página, la tarea más pesada es la de actualizar la parte visible del navegador. React utiliza un DOM virtual, sobre el que realiza las actualizaciones correspondientes a los renderizados, para posteriormente compararlo con el DOM original y realizar únicamente los cambios mínimos necesarios en la página, en lugar de recargarla por completo.

2.1.1 Babel

React se desarrolla mediante una extensión de sintaxis similar a XML para ECMAScript. Además, al estar basado en ECMAScript es **sencillo de utilizar si se conoce JavasScript** ya que utiliza la mayoría de sus funciones y permite la inserción de código HTML facilitando la creación y legibilidad de los componentes. Sin embargo, no es una sintaxis destinada a ser interpretada por los navegadores web, sino que **debe ser preprocesada a través de transpiladores** que lo transforman en lenguajes que sean interpretables por el navegador.

En este contexto aparece **Babel**, un transpilador de código que transforma el JSX de React, en código Javascript . Además de ello, se encarga de transformar las funcionalidades de Javascript más modernas, cuya compatibilidad no es total, al estándar ECMAScript5, compatible con todos los navegadores actuales.

Es posible que, con el paso de los años y las distintas actualizaciones de los navegadores, el uso de los transpiladores no sea necesario pero, por el momento, no existe otra alternativa para poder utilizar React.

2.1.2 Webpack

Webpack se define como un empaquetador de módulos o *bundler*, aunque podríamos definirlo como la herramienta de compilación utilizada por React para generar los archivos definitivos que se utilizarán en el servidor web. A grandes rasgos, es un task runner que se ocupa de procesar unos archivos de entrada y generar los correspondientes archivos de salida (como se puede observar en la figura 2.1).

Mediante un archivo JSON, se definen una serie de reglas que se aplicarán al código fuente para obtener el resultado definitivo. Entre ellas, las más comunes son:

- Transpilación del código Javascript ES6+ y JSX a Javascript ES5 utilizando Babel.
- Gestión de dependencias.
- Concatenación del código.
- Minimización y ofuscación del código.

Esta herramienta es muy útil cuando se quiere desarrollar aplicaciones web con una filosofía modular, es decir, distribuyendo el código en módulos que a su vez son dependencias de otros módulos.

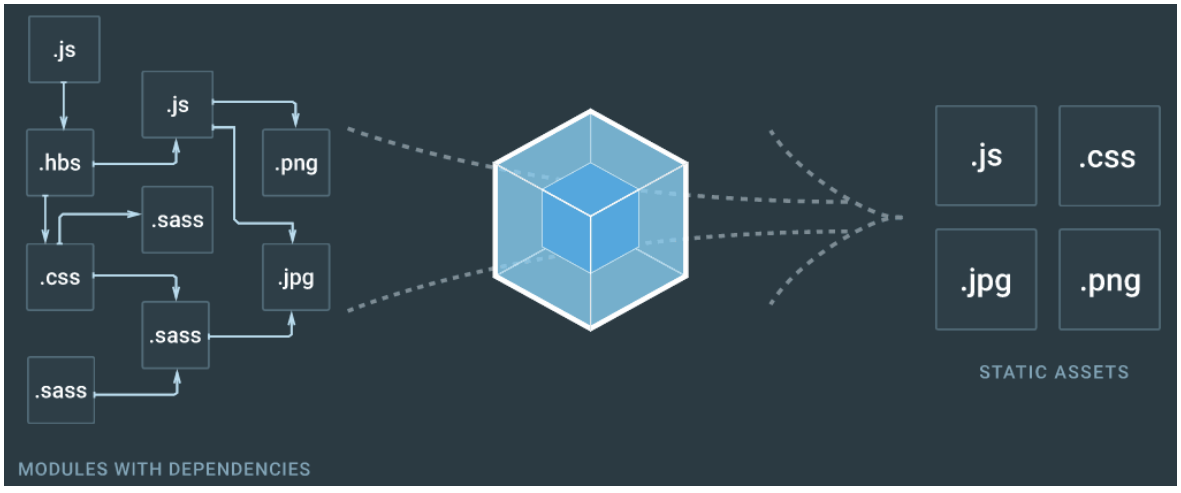


Figura 2.1: Diagrama funcionamiento Webpack. Alarcón [2017]

2.1.3 React router

React Router es una librería que permite la gestión dinámica de rutas para convertir aplicaciones React en Single Page Applications.

Utilizando esta librería, se aprovecha el potencial de React ya que únicamente se realiza una petición HTTP al servidor desde el navegador la primera vez que necesitas cargar la página o bien cuando la actualices manualmente, y el código que se envía como respuesta es siempre el mismo independientemente de la ruta a la que se acceda. A partir de ese momento, todos los cambios en la aplicación o modificaciones en la ruta de la aplicación son gestionadas desde cliente, modificando únicamente aquellos componentes que varían de una ruta a otra.

Esta es una de las características que permiten un incremento muy notable en el rendimiento de las aplicaciones React puesto que se elimina por completo el tiempo de transferencia HTTP de los servidores tradicionales. Además, al delegar gran parte de la lógica al lado del cliente, la escalabilidad es mucho mayor.

Sin embargo para la carga de datos desde la BBDD aún se necesita realizar peticiones a servidor para ello se utiliza una API mediante la cual el servidor retornará los datos necesarios.

2.2 GraphQL

La API utilizada para la aplicación es GraphQL la cual se diferencia bastante de las API tradicionales. Las API RESTFull tradicionales generan una llamada a un método con unos parámetros determinados, posteriormente el servidor genera una respuesta única. En sistemas donde se maneja una gran cantidad de información, esto puede generar dos problemas:

- Exceso de información: En API demasiado genéricas, es habitual encontrar consultas que devuelven gran cantidad de datos para adaptarse, a la vez, a múltiples tipos de consulta, proporcionando información innecesaria. Esto hace que se aumente el tiempo de respuesta y el tamaño de la misma (un factor muy importante, sobre todo en aplicaciones móviles).

- Exceso de métodos: Para corregir el problema del punto anterior, se suele recurrir a crear nuevas llamadas, más específicas, que proporcionen la parte necesaria de los datos en lugar de la totalidad de ellos. Esto puede generar API demasiado extensas y difíciles de mantener.

GraphQl fue creado en 2012 por Facebook, con el objetivo de dar respuesta a estos problemas, permitiendo crear una API más flexible, mantenible y ligera. Su principal diferencia es que, dentro de cada petición, además de incluir el valor de los parámetros de entrada, también se permite seleccionar qué **campos de la respuesta se desea obtener**, en lugar de obtener siempre los mismos.

Además de poseer una **respuesta flexible** en función de los datos que se le pidan, la entrada de datos también puede ser flexible, es decir en función de los datos enviados se pueden obtener diferentes tipos de datos. Esto hace que mediante un solo método podamos recopilar mucha más información que con las API tradicionales (figura 2.2)

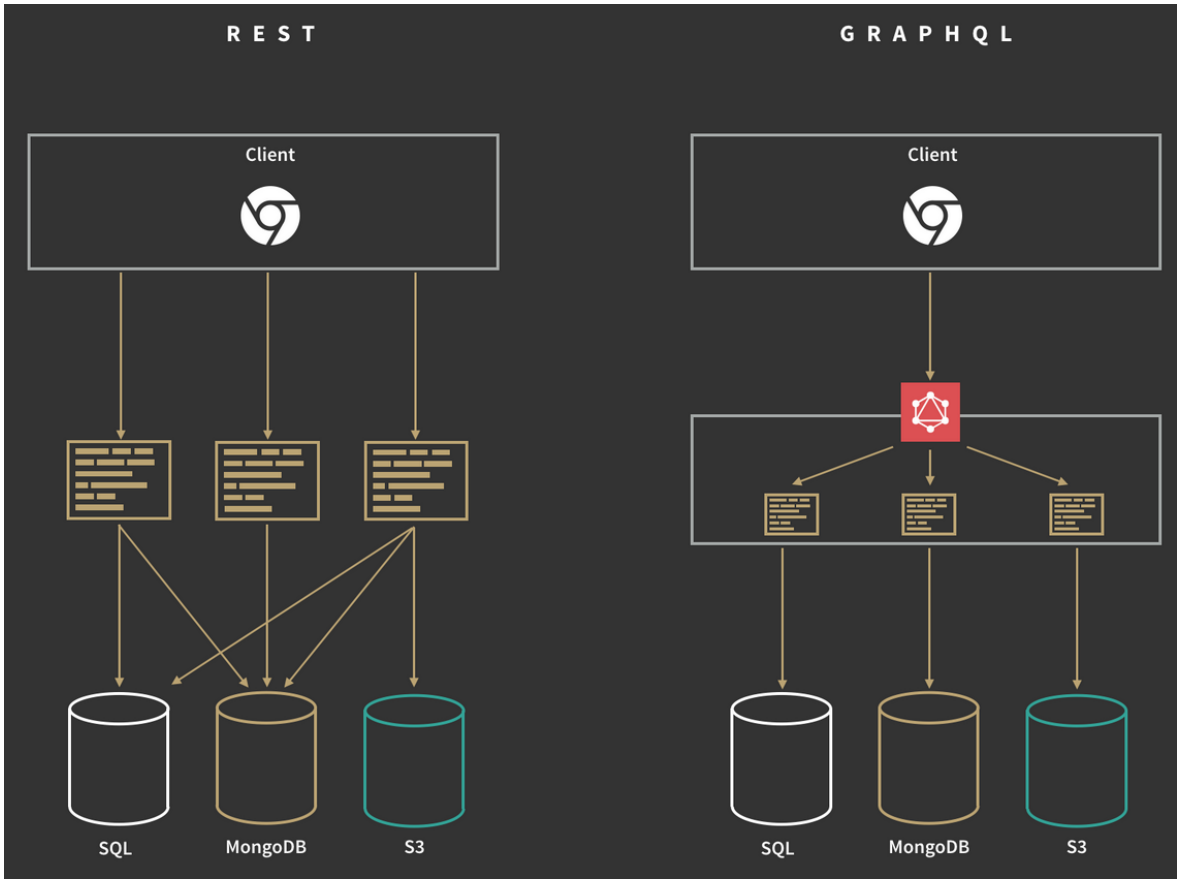


Figura 2.2: Diagrama REST/GRAPHQL. gra

2.3 Node.JS

Node.JS es un entorno en tiempo de ejecución multiplataforma que suele usarse en la capa del servidor (pero no limitada a esta) y que está basado en ECMAScript. Fue creado con el objetivo principal de poder construir plataformas web con una **alta escalabilidad** ya que los entornos tradicionales sufrían cuando se veían sometidos a una alta concurrencia de usuarios. Foundation [2009]

Al estar basado en ECMAScript el lenguaje en el que se codifica es casi idéntico a JavaScript ya que utiliza el motor de código V8, al igual que el navegador Google Chrome. Las principales ventajas de este entorno son las siguientes:

- Se trata de un **entorno asíncrono**, por lo que permite procesar una gran cantidad de solicitudes sin que su rendimiento se vea afectado. La forma en la que atiende las peticiones no se asemeja a los entornos tradicionales en los que para cada petición se asigna un hilo del SO, si no que cada conexión **ejecuta un evento dentro del entorno** que se traduce en un hilo del SO que se ejecuta en bucle, esto permite ahorrar mucho tiempo ya que por la propia naturaleza del motor, este no permite bloqueos y porque no se bloquea directamente cuando realizas una llamada de E/S.

- Orientado a eventos, la arquitectura está **dirigida por eventos** ya que por un lado es más útil que utilizar la orientada a objetos en este tipo de entornos. Esto es debido a que el funcionamiento del servidor se basa en solicitudes que requieren una respuesta, es

decir, se realiza una conexión, se reciben ciertos datos y se devuelve la respuesta, estas acciones son mucho más cercanas a eventos que a un sistema puramente interactivo. Además, esta orientación permite que las operaciones no bloqueantes informen de que han finalizado mediante señales.

- Mayor tasa de transferencia, este entorno permite unas tasas de transferencia o *throughput* mucho mayores a los entornos tradicionales, la desventaja en este caso es que la **latencia puede verse incrementada**, por lo que depende de la orientación que tenga la plataforma web. Sin embargo, cuando la concurrencia crece, esta latencia suele tener una **tendencia lineal** como se puede observar en la figura 2.3

- Modular, al igual que la mayoría de librerías y *frameworks* tiene una estructura modular que permite la **carga y descarga de diferentes módulos** que amplían la funcionalidad base y permiten unas gestión y actualización mucho más eficiente en comparación con otros entornos en los que la funcionalidad viene incluida en el *core*.

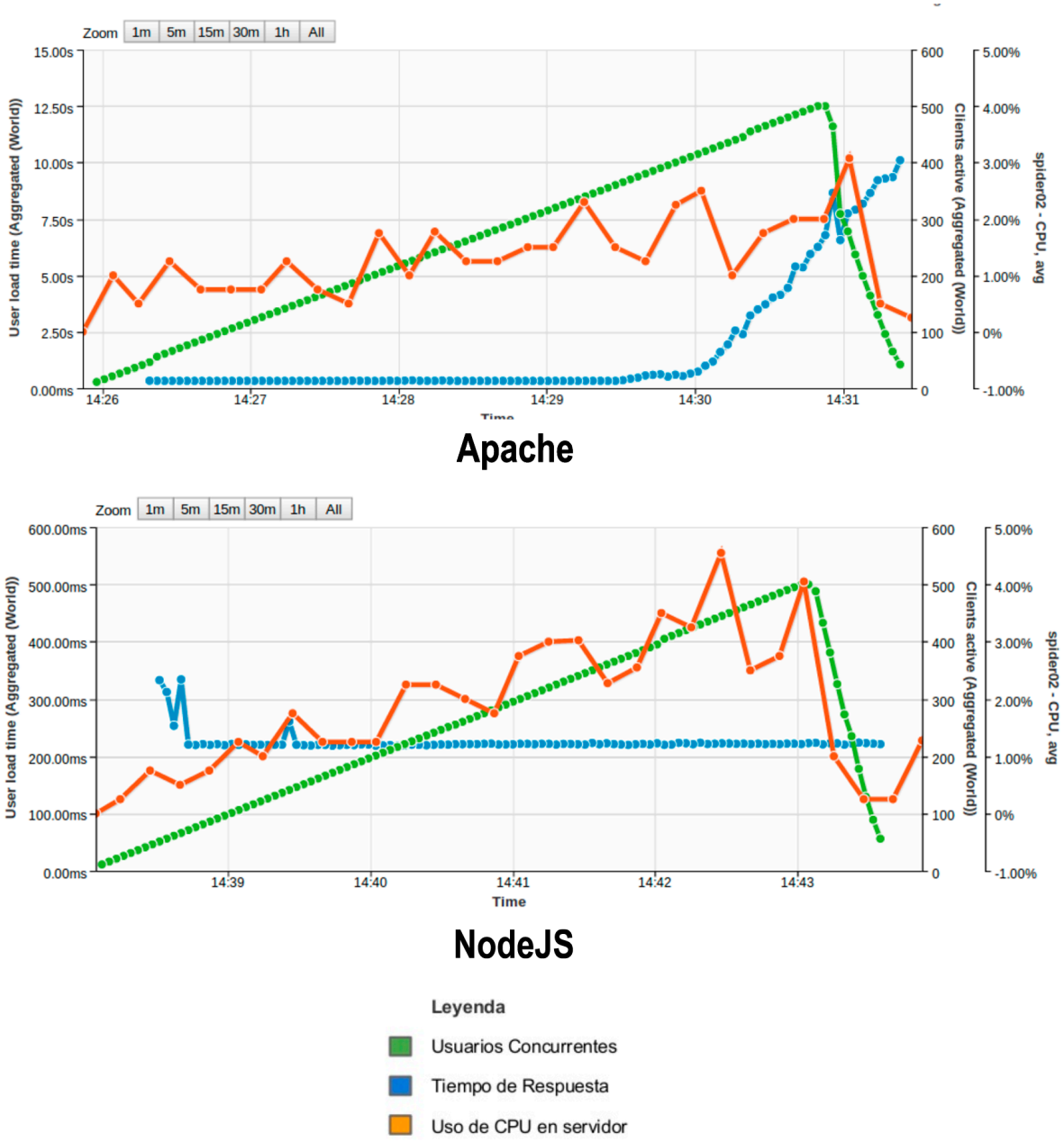


Figura 2.3: Comparativa rendimiento Node.JS vs Apache. Impact [2013]

2.3.1 Express.JS

Express se trata del **framework más utilizado** de Node, y es la librería que permite el funcionamiento de otros frameworks ya que proporciona utilidades base para el **desarrollo web**, podría decirse que actúa como un programa que acepta las peticiones web y las maneja, entre estas utilidades se encuentran:

- Manejador de **peticiones HTTP**, que es el encargado de recibir y responder a las peticiones realizadas desde los clientes, acepta la gran mayoría de tipos de peticiones HTTP.

- Ajuste de aplicaciones web, permite **realizar ajustes** en la aplicación web como esta-

blecer el puerto de escucha, búsqueda de la ruta de las plantillas que se utilizan para responder a las peticiones.

- Actua de *middleware*, esta es una de las utilidades más importantes ya que permite **preprocesar las solicitudes** a través de todos los *middlewares* necesarios ya sean propios o módulos de terceros. Esto es de utilidad para desestructurar las peticiones, clasificarlas en función del método utilizado, manejar los diferentes errores (tal y cómo se puede observar en la figura)

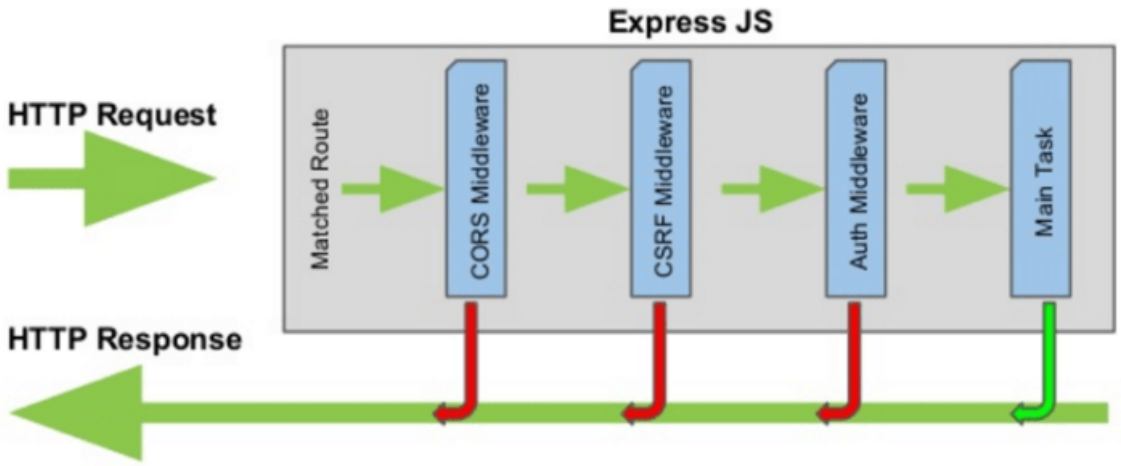


Figura 2.4: Diagrama procesado Express. Kononenko

2.4 MongoDB

MongoDB se trata de una base de datos multiplataforma NoSQL orientado a documentos, esto se traduce en que el formato que tiene no se basa en tablas como en las BBDD SQL. Además está escrito en C++ lo cual le confiere bastante rapidez a la hora de ejecutar las tareas. Las principales características que posee este sistema son:

- Almacena los datos en **documentos en formato JSON** flexibles, con lo cual cada documento registrado puede contener una estructura diferente con campos diferentes que se pueden modificar posteriormente. Esto puede resultar ventajoso ya que en ocasiones objetos que a priori poseen los mismos campos pueden tener diferentes propiedades, cabe destacar que desde el lado servidor se pueden **fijar modelos flexibles** para que su estructuración sea más sencilla. Esto hace que la integración de los datos sea más fácil y rápida.

- Es una **base de datos distribuida**, por lo que proporciona una elevada disponibilidad, escalabilidad horizontal y distribución. Además esto le confiere gran solvencia a la hora de balancear la carga.

- Infraestructura en la nube, mediante esta solución se ahorran muchos costes ya que se pueden registrar una gran cantidad de datos ofreciendo una **gran disponibilidad y flexibilidad** mediante la fragmentación y distribución en varios servidores.

- Obtención de informes en tiempo real, esto permite reunir datos de diferentes fuentes y obtener una vista instantánea de los datos para realizar un análisis sobre ellos.

Aunque normalmente las bases de datos NoSQL tienen un ámbito reducido cada día se están mejorando para proporcionar las ventajas de las que disponene los sistemas clásicos SQL. La principal característica por la que se ha escogido este sistema es por la necesidad de tener **datos semi estructurados** ya que los informes registrados no siempre poseen los mismos datos.

No suele recomendarse en sistemas que requieran operaciones tales como JOIN ya que este sistema no lo permite y requiere realizar varias consultas, sin embargo al **disponer de apuntadores** a otros documento,s si se estructura bien, se puede prescindir de estas operaciones que a menudo resultan muy costosas si se maneja una elevada cantidad de datos.

En la siguiente figura 2.5 se puede observar una comparación entre las propiedades de una base de datos SQL y una basada en MongoDB

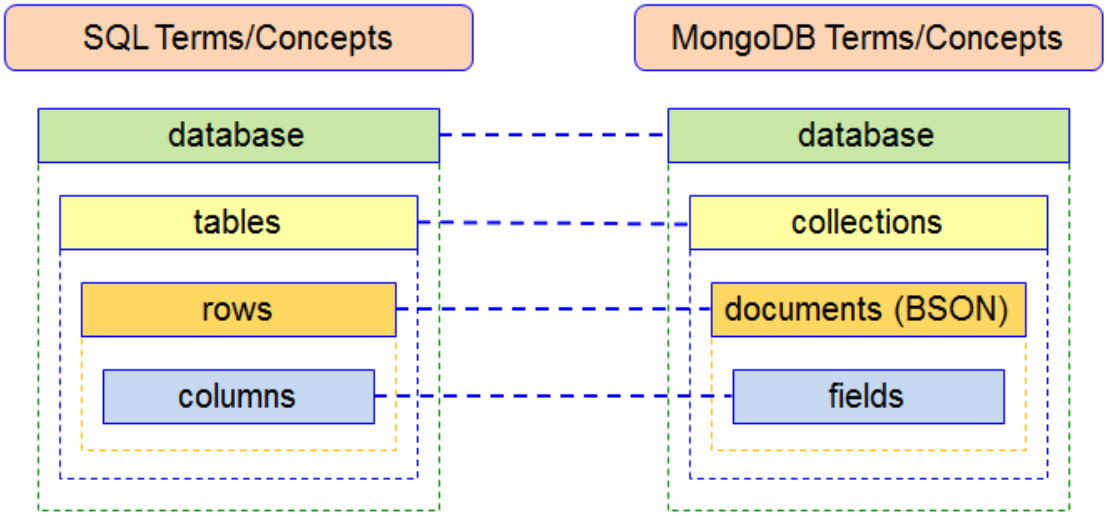


Figura 2.5: Diagrama comparativo entre SQL y MongoDB. Malaya [2013]

2.4.1 Mongoose

Mongoose se trata de un framework que **actua como un *mapper* (como se puede observar en la figura 2.6)** para NodeJS y que facilita enormemente las consultas realizadas a bases de datos basadas en MongoDB. Una de las mejores utilidades de las que dispone es que permite definir objetos con un esquema que se pueden asignar a un documento de MongoDB.

Aunque anteriormente se ha comentado que MongoDB no sigue un esquema fijo, es conveniente fijar la estructura de los documentos que se van a guardar para mantener una coherencia y facilitar la estructuración de los datos, además no se pierde flexibilidad ya que se pueden fijar los atributos del documento como obligatorios u opcionales en función de lo que se necesite.

Además de esto Mongoose contiene muchas funciones a mayores que permiten guardar, validar, eliminar y consultar los datos almacenados utilizando las funciones comunes de MongoDB

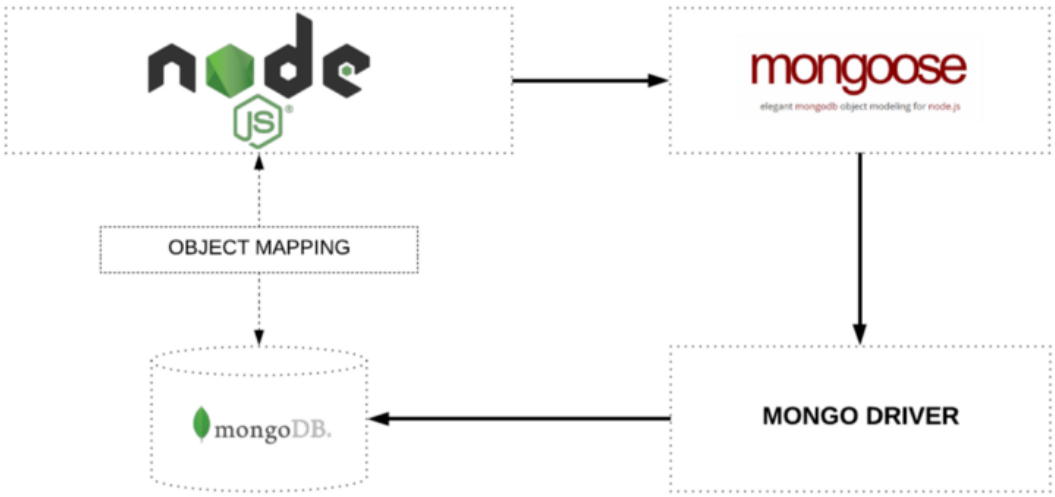


Figura 2.6: Diagrama funcionamiento Mongoose. Karnik [2018]

2.5 Npm

Npm es un gestor de paquetes y módulos que suele utilizarse junto con Node.JS. Consiste en un cliente de línea de comandos y un repositorio en línea de paquetes públicos y privados. Está completamente codificado en JavaScript. NPM [2013]

Las principales ventajas de utilizar este gestor son:

- Gestión **eficiente y centralizada** del código, permite buscar código empaquetado a través de su repositorio, al contrario que otros gestores en los que hay que buscar en varios repositorios.
- Gestión de dependencias, permite realizar las **instalaciones y actualizaciones de forma ordenada**, dando lugar a realizar despliegues del proyecto sin necesidad de mover todas las dependencias y con la ejecución de comandos sencillos.
- Soporte de la comunidad, al disponer de repositorios públicos y privados existe una **amplia comunidad** que se dedica al mantenimiento y revisión de los paquetes pudiendo elegir entre gran variedad.

3 Planteamiento de la solución

3.1 Modelo de casos de uso

En la siguiente figura 3.1 se muestra el diagrama de casos de uso del sistema.

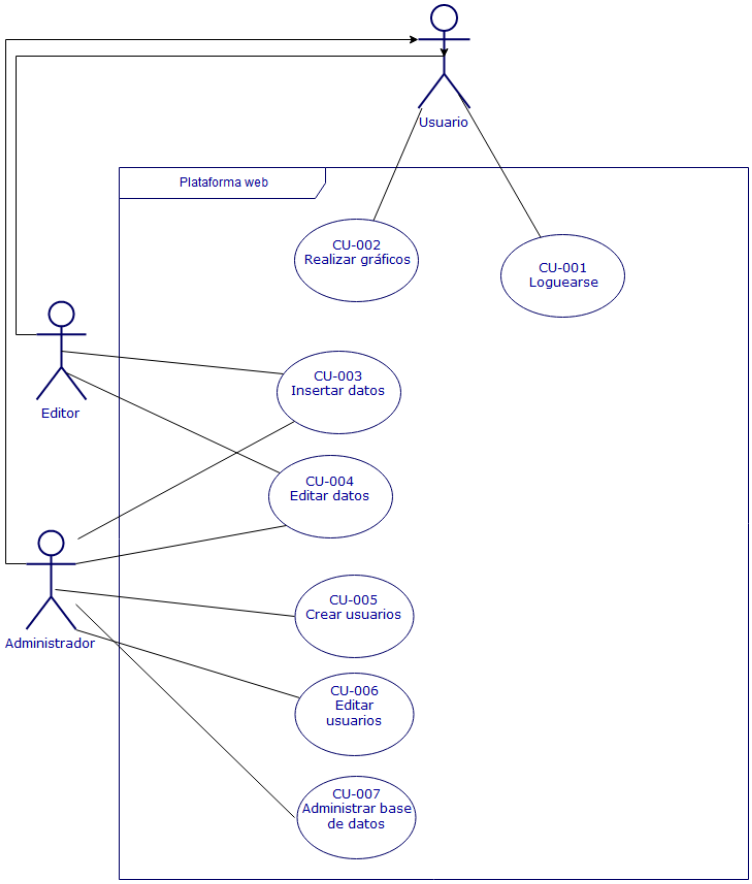


Figura 3.1: Diagrama de los casos de uso

Los actores que participan en el sistema son los siguientes:

- **Usuario**, es un actor principal que accede a la web y que únicamente tiene permisos para *loguearse* o realizar visualizaciones de gráficas.
- **Editor**, es un actor principal que tiene una cuenta registrada en el sistema y tiene permisos para crear y editar datos de los proyectos.
- **Administrador**, es un actor principal que tiene una cuenta registrada en el sistema y tiene permisos para crear y editar las cuentas de los usuarios.

3.2 Análisis de requisitos

En esta sección se definirán los requisitos, que son declaraciones de las funcionalidades que dispondrá el sistema y las restricciones que existirán sobre las mismas.

3.2.1 Requisitos funcionales

Identificación	*RF-01*
Descripción	*El sistema deberá permitir la inserción de datos de los proyectos y datos asociados a ellos*
Importancia	*Esencial*
Prioridad	*Muy alta*
Riesgo	*Bajo*
Caso de Uso	*CU-003*

Identificación	*RF-02*
Descripción	*El sistema deberá permitir la edición de datos de los proyectos y datos asociados a ellos que ya se encuentren registrados en el sistema*
Importancia	*Esencial*
Prioridad	*Muy alta*
Riesgo	*Bajo*
Caso de Uso	*CU-004*

Identificación	*RF-03*
Descripción	*El sistema deberá permitir la visualización de gráficas extraídas de los datos seleccionados por el usuario, el listado de los datos seleccionables puede encontrarse en la siguiente figura*
Importancia	*Esencial*
Riesgo	*Bajo*
Prioridad	*Muy alta*
Caso de Uso	*CU-002*

Identificación	*RF-04*
Descripción	*El sistema deberá proporcionar una interfaz gráfica a través de la cual el usuario puede acceder a las diferentes funcionalidades de la plataforma*
Importancia	*Esencial*
Prioridad	*Muy alta*
Riesgo	*Bajo*
Caso de Uso	*Todos*

Identificación	*RF-05*
Descripción	*El sistema organizará el acceso a la información a través de la interfaz dividiendo los apartados en:* • *Datos: apartado en el cual se insertarán datos referentes a los proyectos* • *Gráficos: apartado en el cual se mostrarán los gráficos extraídos de los datos* • *Usuarios: apartado para la gestión de usuarios del sistema*
Importancia	*Esencial*
Riesgo	*Bajo*
Prioridad	*Muy alta*
Caso de Uso	*CU-002, CU-003, CU-004, CU-005, CU-006*

Identificación	*RF-06*
Descripción	*El sistema controlará la inserción de los datos a través de formularios, asegurándose que los datos introducidos son validados*
Importancia	*Esencial*
Riesgo	*Medio*
Prioridad	*Media*
Caso de Uso	*CU-003, CU-005*

Identificación	*RF-07*
Descripción	*El sistema deberá tener un sistema de acceso a la base de datos en la cual podrán visualizarse todos los datos guardados en la plataforma.*
Importancia	*Esencial*
Riesgo	*Bajo*
Prioridad	*Alta*
Caso de Uso	*CU-007*

Identificación	*RF-08*
Descripción	*El sistema poseerá los siguientes roles de usuario:* • *Administrador: Tendrá acceso a todos los apartados del sistema* • *Editor: Tendrá acceso a todos los apartados excepto al control de usuarios.* • *Usuario no registrado: únicamente tiene acceso a la visualización de gráficas*
Importancia	*Esencial*
Riesgo	*Bajo*
Prioridad	*Alta*
Caso de Uso	*CU-002*

Identificación	*RF-09*
Descripción	*El sistema llevará un registro de los usuarios registrados y permitirá registrar a nuevos usuarios a través de un apartado designado para ello al que sólo podrán acceder los administradores*
Importancia	*Alta*
Riesgo	*Medio*
Prioridad	*Alta*
Caso de Uso	*CU-005, CU-006*

3.2.2 Requisitos no funcionales

Identificación	*RNF-01*
Descripción	*El sistema deberá estar instalado en un servidor de la universidad, al tratarse de un sistema multiplataforma el sistema operativo queda a elección del cliente.*
Importancia	*Esencial*
Prioridad	*Alta*
Riesgo	*Bajo*

Identificación	*RNF-02*
Descripción	*El sistema deberá ser accesible de forma pública desde Internet, aunque existan apartados que estén cerrados a la gente que no esté registrada en la plataforma*
Importancia	*Esencial*
Prioridad	*Alta*
Riesgo	*Bajo*

Identificación	*RNF-03*
Descripción	*El sistema deberá estar adaptado para poderse acceder desde plataformas de escritorio y móviles*
Importancia	*Esencial*
Prioridad	*Media*
Riesgo	*Bajo*

Identificación	*RNF-04*
Descripción	*El sistema no deberá exceder unos tiempos máximos de carga de 3 segundos para todas las interfaces y de 5 para la generación de gráficas*
Importancia	*Esencial*
Prioridad	*Media*
Riesgo	*Bajo*

Identificación	*RNF-05*
Descripción	*El sistema no deberá solicitar una confirmación para cualquier borrado de datos que se produzca en la base de datos*
Importancia	*Esencial*
Prioridad	*Alta*
Riesgo	*Alto*

3.3 Diagrama de clases

En la siguiente figura 3.14 se muestra el diagrama de clases, el cual representa una estructura estática del sistema mostrando las clases y sus atributos y operaciones, así como las relaciones entre los objetos.

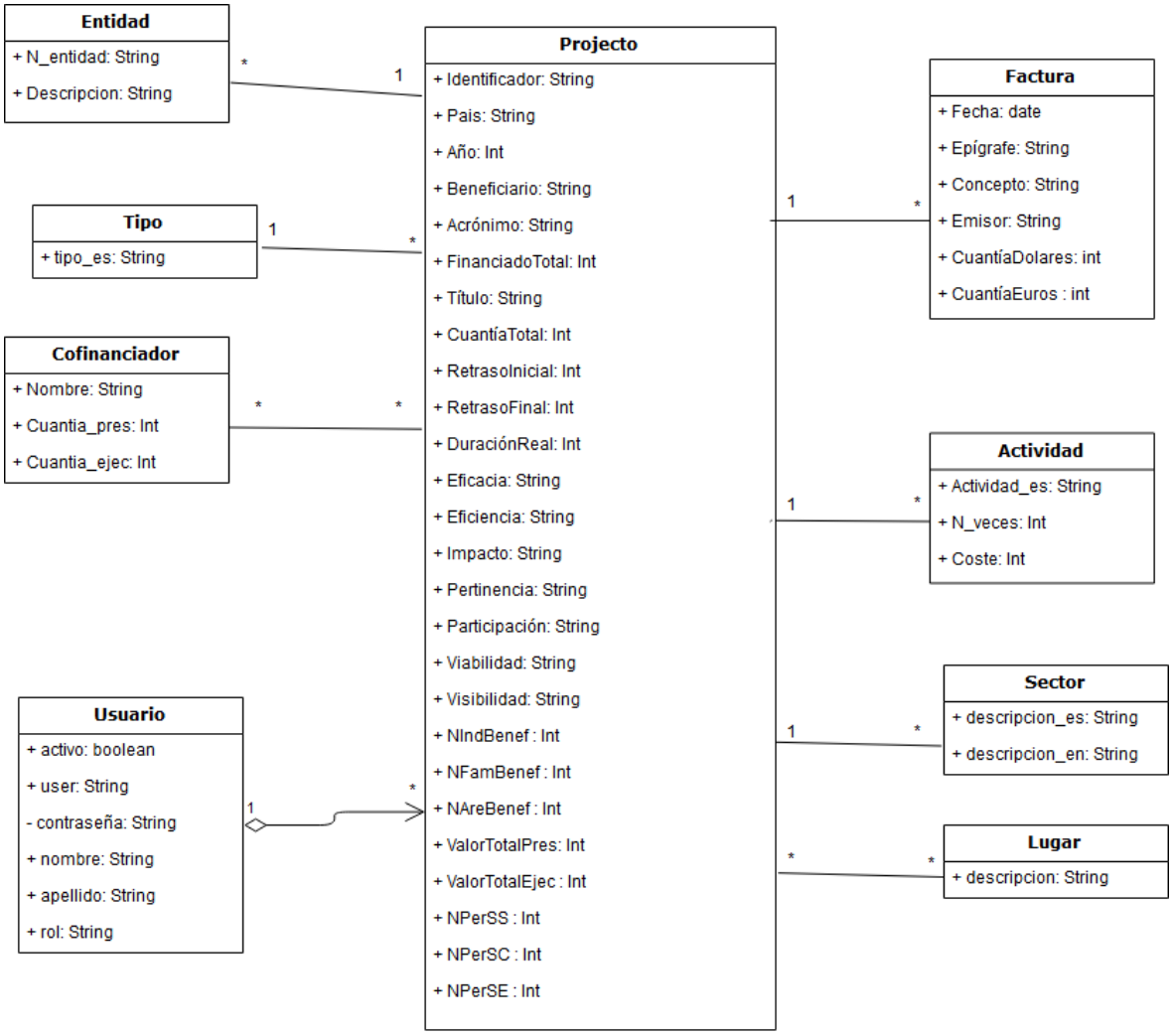


Figura 3.14: Diagrama de clases

3.4 Diagrama de actividad

En este apartado se detallara el diagrama de actividad 3.15 de una de las funcionalidades más importantes de la plataforma, la creación de un gráfico, a través de este diagrama se modelará el comportamiento del sistema en el caso de uso CU-002

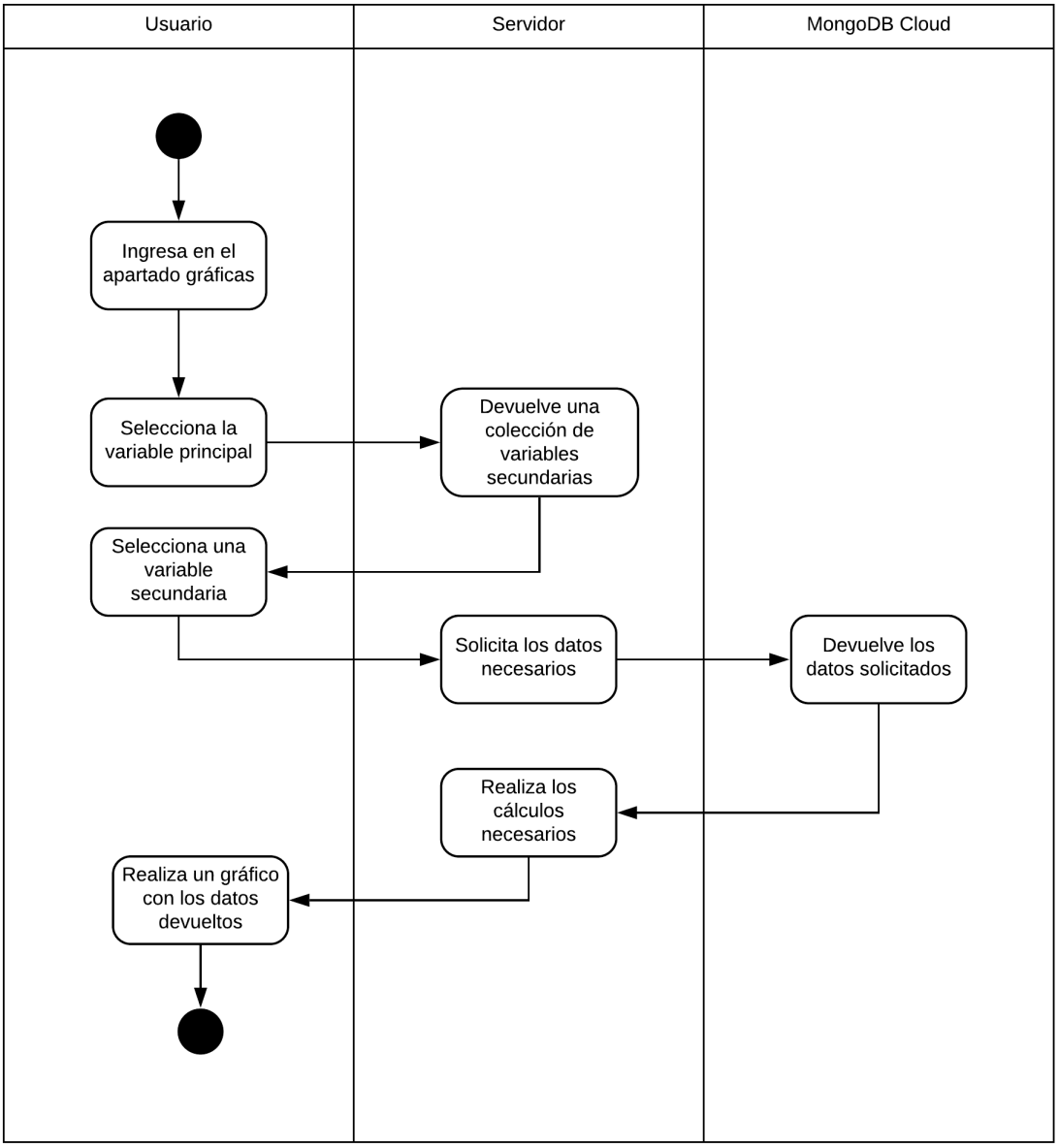


Figura 3.15: Diagrama de actividad del CU-002

3.5 Arquitectura de la solución

En este apartado se describe la arquitectura lógica y física que se planificó para el sistema. El proyecto en el que se basa poseía una arquitectura lógica Cliente-Servidor, en la cual el cliente tomaba la lógica de presentación y de aplicación, mientras que el servidor tomada la

lógica de acceso a datos e incluso la propia base de datos.

Al plantear una **renovación de la aplicación**, la arquitectura propuesta se basa en una arquitectura dividida principalmente en 4 capas. [ver Richards, 2015, Cap. 1]

- **Capa de presentación**, referente a la interacción entre el usuario y el sistema, compuesta por la interfaz de usuario y la lógica asociada a la misma. Su principal responsabilidad es mostrar la información al usuario.
- **Capa de negocio**, es la encargada de recibir las peticiones del usuario, procesar la información de entrada y realizar las funciones necesarias para tratar la información.
- **Capa de servicios**, que consiste en la lógica que realiza las funciones principales de la aplicación: procesamiento de datos, implementación de funciones de negocio
- **Capa de base de datos**, encargada de la persistencia de los datos.

En la figura 3.16 se muestra el diagrama de despliegue de la solución, a través del cual estará modelada la arquitectura en tiempo de ejecución del sistema.

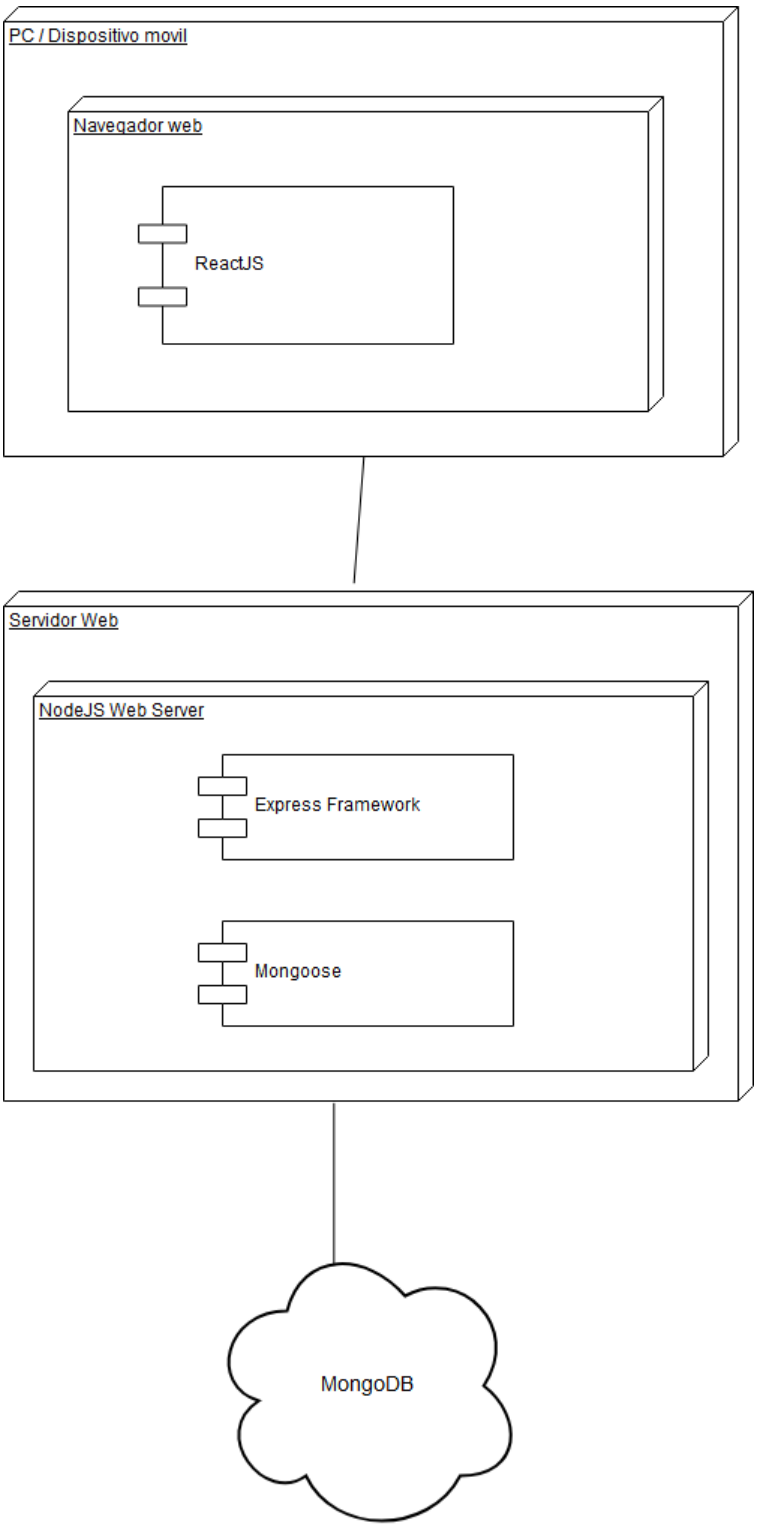


Figura 3.16: Diagrama de despliegue

4 Planificación

4.1 Introducción

El objetivo de la planificación software es recopilar toda la información necesaria para llevar a cabo la gestión, control y seguimiento del proyecto. En él se describe el plan global que será seguido durante el desarrollo de la plataforma.

Para realizar la estimación de costes y tiempos asociados a este proyecto se debe tener en cuenta el carácter formativo de este proyecto y la existencia de un único desarrollador, estudiante de Ingeniería Informática, por lo que los costes del esfuerzo se consideran nulos ya que no va a recibir ninguna retribución económica por su trabajo. Pérez [2009]

De la misma forma, el resto de costes (indirectos, de hardware y de software) no son aplicables en este proyecto ya que se aprovechan las instalaciones y materiales de la ETS de Ingeniería Informática puestos a disposición del alumno por el tutor principal del proyecto. Pérez [2009]

La planificación de este proyecto se ha realizado tomando como marco de trabajo Scrum. Este marco de trabajo se basa en el trabajo en equipo, por ello no se ha aplicado de manera literal, sino que se ha adaptado a los actores involucrados en el mismo, el alumno y los dos tutores. Puede carecer de lógica adaptar este marco, sin embargo, al estar enfocado al desarrollo ágil de software basado en ciclos temporales de duración fija, lo hace ideal ya que las diferentes fases del proyecto se repiten en cada iteración permitiendo una retroalimentación constante entre los diferentes participantes del proyecto.

La opción alternativa se hubiera basado en una gestión clásica, basada en el Proceso Unificado de Desarrollo Software, este marco de trabajo se caracteriza por estar basado en casos de uso y en la arquitectura del proyecto, además de ser iterativo e incremental. Podría dividirse en cuatro fases para entender mejor su desarrollo.

- **Inicio**, en esta fase de analizan varios factores principales como el alcance del proyecto, su viabilidad, las metas que se desean alcanzar, los plazos y los costos.
- **Elaboración**, en esta fase se realiza una mejora de la visión del proyecto, se empieza a implementar la funcionalidad básica del proyecto, se acotan los riesgos de forma más concisa y se ajustan las estimaciones realizadas.
- **Construcción**, abarca el desarrollo del producto hasta que incluya unos requisitos mínimos.
- **Transición**, durante esta fase el producto debe estar preparado para ser probado y utilizado por un cliente sin problemas, una vez finalice esta fase se podrá comenzar a pensar en nuevos desarrollos que mejoren su funcionamiento.

En la figura 4.1 se puede observar que estas fases se dividen en distintas iteraciones, este enfoque tiene la principal desventaja que cada fase depende de su iteración por lo que no dispone de tanta flexibilidad y retroalimentación como los marcos de trabajo ágiles.

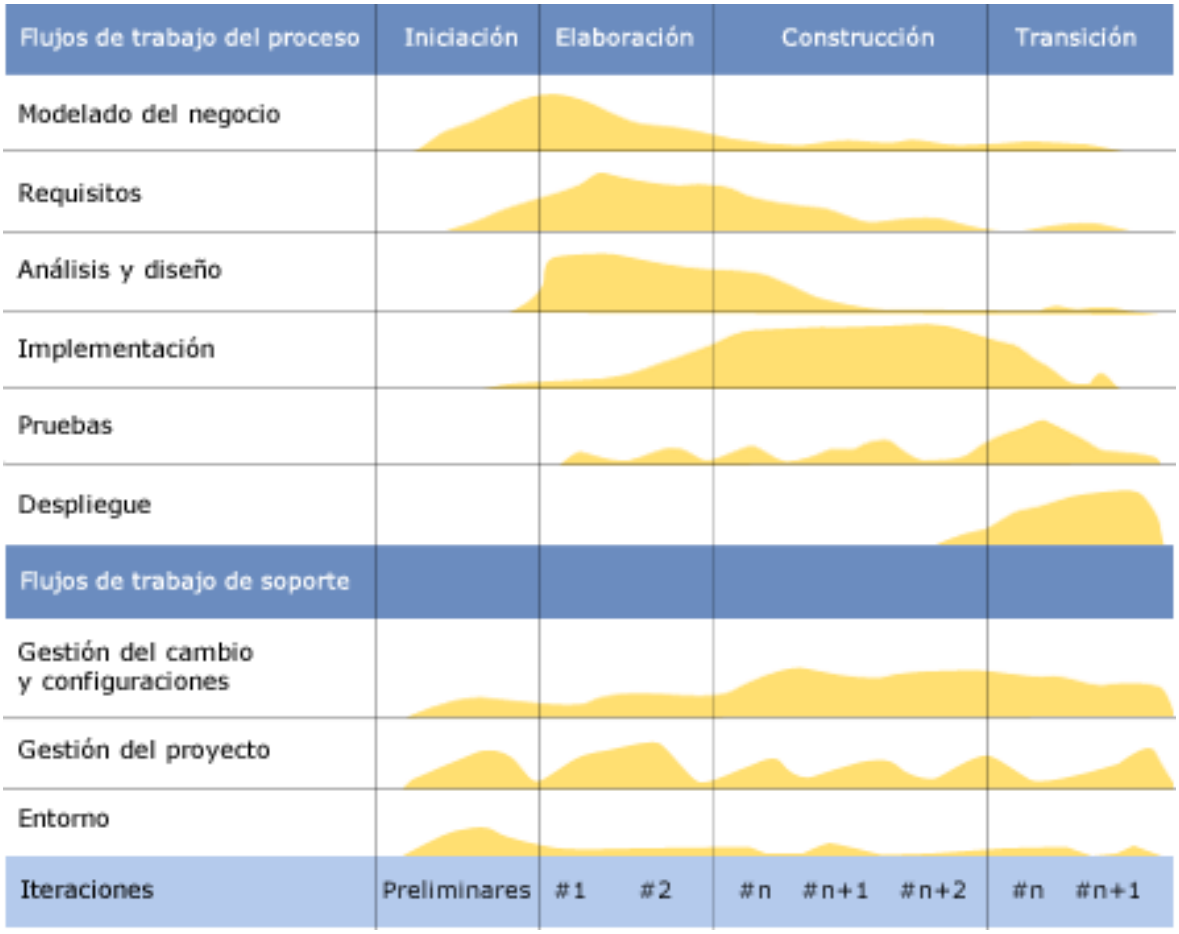


Figura 4.1: Diagrama fases RUP. Commons [2008]

Ahora se analizarán las principales ventajas de adoptar el marco de trabajo SCRUM con respecto a los marcos tradicionales de gestión predictiva.

- **Mejor adaptación**, esto se traduce en una respuesta rápida a los cambios, al tratarse de un proceso evolutivo se pueden realizar cambios de manera más fácil durante las iteraciones, no es necesario esperar hasta el final para corregir los fallos.

- **Mejor retroalimentación**, las entregas parciales permiten una optimización de los recursos y facilitan las labores de seguimiento. De esta forma el producto que se presente al final será el conjunto de varios desarrollos parciales que han sido revisados.

- **Priorización de tareas**, al poder priorizar las tareas en cada iteración, se puede saber con certeza cuales tienen una mayor relevancia para el proyecto de forma que se pueden centralizar los esfuerzos y unificar los criterios de actuación sobre las mismas.

Sin embargo, también se ha de tener en cuenta los inconvenientes que tiene con respecto otros marcos. Existe una mayor dependencia de los actores responsables en el proyecto, esto es debido a que al existir un mayor número de reuniones y evaluaciones sean los líderes del proyecto sobre los que recaiga una mayor responsabilidad. Además, por lo general suele haber una falta de documentación debido a que las metodologías ágiles no plantean como tal alternativas a la recopilación de información de los proyectos.

Por ello, en este caso lo que se realizará es una planificación basada en SCRUM, sin embargo, se paliarán los principales defectos de este marco ya que la elaboración de documentación es fundamental para un trabajo de este tipo y respecto a los responsables del proyecto al ser un número menor que en proyecto tradicional la responsabilidad recaería de la misma forma que con una planificación basada en un marco de trabajo tradicional.

Para la estimación y desarrollo de las tareas se ha de tener en cuenta que este proyecto tiene un **objetivo formativo** y únicamente existe un desarrollador por ello se ha adaptado este sistema

4.2 Product Backlog

En este apartado se definirá el product backlog, es decir, la lista ordenada de todas las tareas que haya que realizar en el proyecto. Se enumeraran los requisitos y las tareas y la estimación en horas realizada para cada una de ellas.

Aunque el product backlog suele ser un elemento flexible para la mayoría de los proyectos en los que se aplica una planificación tipo SCRUM, en este caso, se ha optado por una planificación creada desde el principio ya que **se conoce a priori el alcance** del proyecto por lo que puede completarse de una forma más exacta.

En la figura 4.2 se muestra el product backlog con la lista de requisitos, sus tareas asociadas y el estado y el tiempo estimado. Esta captura se realizó durante la **primera semana** de desarrollo del proyecto. Se ha optado por captarlo en este momento para ver el conjunto completo de estados disponibles.

Requisito	Tarea	Estado	Nº de horas estimadas
Análisis de la solución anterior	Revisión de la documentación	Completado	10
	Identificación de los requisitos	No iniciado	36
	Planificación de la migración	No iniciado	12
Implantación y estabilización de la plataforma	Diseño de la BBDD	No iniciado	36
	Diseño de las interfaces	No iniciado	24
	Codificación básica del servidor en el backend	No iniciado	36
Codificación	Migración de los datos	No iniciado	16
	Autenticación	No iniciado	8
	Diseño e implementación de las APIS	No iniciado	24
	Implementación de las interfaces	No iniciado	24
Documentación	Realización de la memoria	En proceso	48
Pruebas	Realización de pruebas y pequeñas implementaciones	No iniciado	8
Defensa	Realización de la presentación y defensa del proyecto	No iniciado	8
		Nº total de horas estimadas	290

Figura 4.2: Product Backlog (1a semana)

4.3 Sprint Backlog

El Sprint Backlog es el listado de tareas que proviene del desglose de las Historias de Usuario que conforman el Product Backlog. Como en este caso la planificación no se está realizando puramente estilo SCRUM, no se dispone de historias de usuario como tal si no que se utilizan requisitos.

Además en el Sprint Backlog se realiza un desglose de las horas realizadas en cada tarea, de forma que, se pueden comparar las horas estimadas con las horas invertidas reales. Con esta comparativa se puede ver el progreso realizado y realizar un gráfico de avance *burnout* que puede definirse como un gráfico que muestra el avance que se va produciendo en el proyecto y nos permite evaluar si existen problemas en las estimaciones.

La planificación se ha estructurado en un **periodo temporal de dos meses**, los cuales han sido divididos en un sprint por semana dando lugar a un total de 9 sprints aproximadamente. Para la planificación de horas se ha estipulado una **estimación diaria de 4 horas**

a lo largo de 6 sprints, otra estimación de 6 horas diarias a lo largo de 2 sprints y el sprint restante es dinámico por lo que se ajustará en función de como evolucione el trabajo.

En la figura 4.3 se puede observar el progreso realizado durante la primera semana, durante la cual se finalizó la tarea de revisión del proyecto anterior con un saldo de dos horas en negativo. Esto indica que la **estimación inicial fue infravalorada** para esa tarea, esto no es un problema debido a que esta planificación permite reajustar las horas y planificar que tareas realizar en cada sprint al final del mismo.

Tareas	Tiempo estimado	Tiempo real	Tiempo restante	1	2	3	4	5	6	7
Revisión de la documentación	10	**12**	-2	3	4		3	2		
Identificación de los requisitos	36	**36**	35							1
Planificación de la migración	12	**12**	12							
Diseño de la BBDD	36	**36**	36							
Diseño de las interfaces	24	**24**	24							
Codificación básica del servidor en el backend	36	**36**	36							
Migración de los datos	16	**16**	16							
Autenticación	8	**8**	8							
Diseño e implementación de las APIS	24	**24**	24							
Implementación de las interfaces	24	**24**	24							
Realización de la memoria	48	**48**	39	2	1		4			2
Realización de pruebas y pequeñas implementaciones	8	**8**	8							
Realización de la presentación y defensa del proyecto	8	**8**	8							
TOTAL	290	292	268	5	5	0	7	2	0	3
Daily burnout	0			4	4	4	4	4	4	4
Tiempo total restante (estimado)			290	286	282	278	274	270	266	262
Tiempo total restante (real)			292	287	282	282	275	273	273	270

Figura 4.3: Sprint Backlog (1a semana)

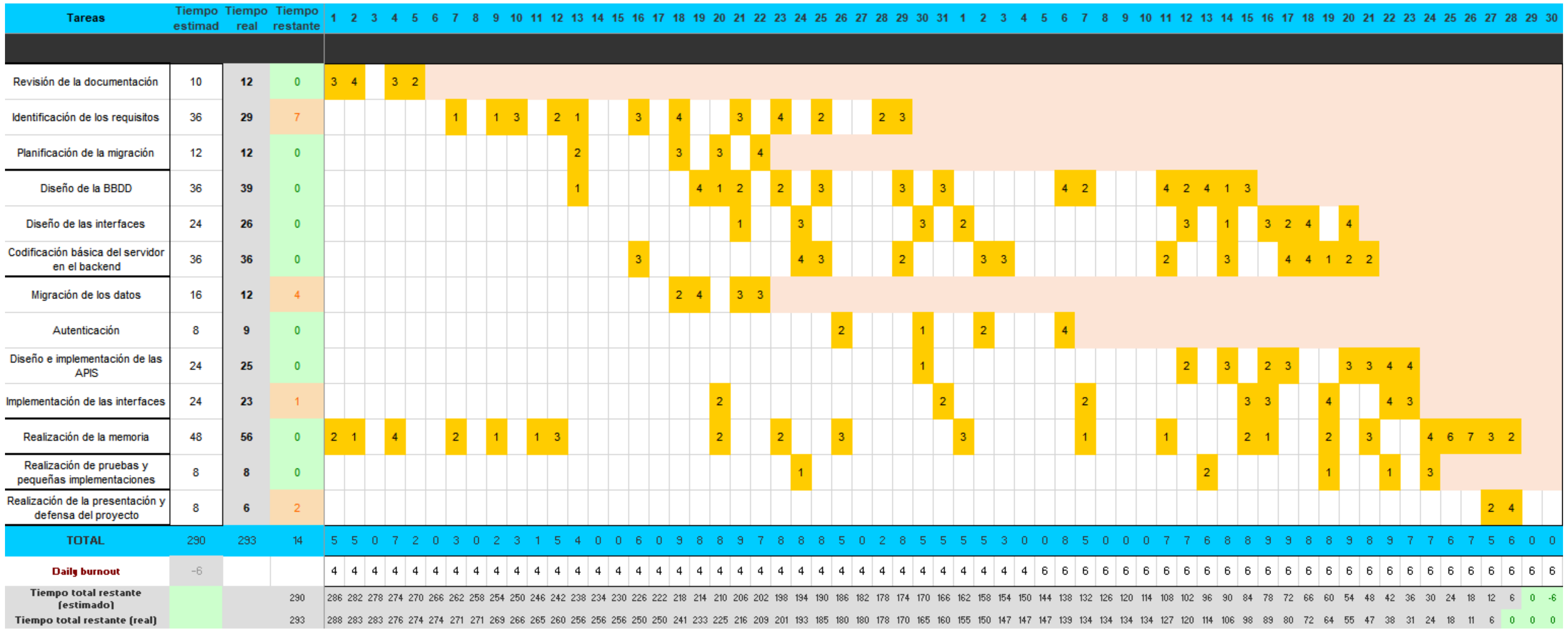
Sin embargo, en la figura FINAL se puede observar cómo se tuvieron que **reajustar las horas de trabajo** de cada día, esto se produjo debido a que hubo días en los que no se

trabajó y una serie de tareas que se infravaloraron mientras que otras se sobreestimaron. Esto produjo un **saldo positivo de 3 horas**, que indica que hubo que trabajar 3 horas más de las estimadas en al principio.

A través de estos datos se puede elaborar el gráfico de avance o *burnout* (figura 4.5), este gráfico muestra el progreso realizado comparándolo con la planificación ideal, por lo que, se puede detectar si las estimaciones fueron incorrectas y si se está realizando de forma correcta el trabajo planificado.

En la gráfica la línea azul muestra el **avance lineal ideal** marcado por la planificación que se realizó al principio del proyecto, mientras que la línea roja indica el **avance real**. Si la línea roja sobrepasa a la azul, significa que se ha producido un error en la estimación y que habrá que realizar un sobreesfuerzo o replanificar las horas para poder alcanzar la fecha deseada. Sin embargo en los momentos en los que la línea azul sobrepasa a la roja, significa que el avance real es mejor que el estimado.

En la figura 4.4 se puede observar el sprint backlog al finalizar el proyecto, en esta figura se puede ver los fallos de planificación respecto varias tareas ya que algunas se sobreestimaron y otras se infravaloraron. Para paliar este problema se ha realizado un **ajuste de horas** en ciertos días trabajados, es decir, se aumentó el número de horas trabajadas algunos días mientras que otros se descansó. También se muestra el número total de horas invertidas en el proyecto y a través de estos datos se ha generado el gráfico de Burndown.



Tareas	Tiempo estimad	Tiempo real	Tiempo restante	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Revisión de la documentación	10	12	0	3	4		3	2																										
Identificación de los requisitos	36	29	7							1		1	3		2	1			3		4			3		4		2			2	3		
Planificación de la migración	12	12	0													2					3		3		4									
Diseño de la BBDD	36	39	0													1						4	1	2		2		3				3		3
Diseño de las interfaces	24	26	0																					1			3						3	
Codificación básica del servidor en el backend	36	36	0																3								4	3				2		
Migración de los datos	16	12	4																		2	4		3	3									
Autenticación	8	9	0																										2				1	
Diseño e implementación de las APIS	24	25	0																														1	
Implementación de las interfaces	24	23	1																				2											2
Realización de la memoria	48	56	0	2	1		4			2		1		1	3								2			2			3					
Realización de pruebas y pequeñas implementaciones	8	8	0																								1							
Realización de la presentación y defensa del proyecto	8	6	2																															
TOTAL	290	293	14	5	5	0	7	2	0	3	0	2	3	1	5	4	0	0	6	0	9	8	8	9	7	8	8	8	5	0	2	8	5	5
Daily burnout	-6			4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Tiempo total restante (estimado)			290	286	282	278	274	270	266	262	258	254	250	246	242	238	234	230	226	222	218	214	210	206	202	198	194	190	186	182	178	174	170	166
Tiempo total restante (real)			293	288	283	283	276	274	274	271	271	269	266	265	260	256	256	256	250	250	241	233	225	216	209	201	193	185	180	180	178	170	165	160

Tareas	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Revisión de la documentación																														
Identificación de los requisitos																														
Planificación de la migración																														
Diseño de la BBDD						4	2				4	2	4	1	3															
Diseño de las interfaces	2											3		1		3	2	4		4										
Codificación básica del servidor en el backend		3	3								2			3			4	4	1	2	2									
Migración de los datos																														
Autenticación		2				4																								
Diseño e implementación de las APIS												2		3		2	3			3	3	4	4							
Implementación de las interfaces							2								3	3			4			4	3							
Realización de la memoria	3						1				1				2	1			2		3			4	6	7	3	2		
Realización de pruebas y pequeñas implementaciones													2						1			1		3						
Realización de la presentación y defensa del proyecto																											2	4		
TOTAL	5	5	3	0	0	8	5	0	0	0	7	7	6	8	8	9	9	8	8	9	8	9	7	7	6	7	5	6	0	0
Daily burnout	4	4	4	4	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
Tiempo total restante (estimado)	162	158	154	150	144	138	132	126	120	114	108	102	96	90	84	78	72	66	60	54	48	42	36	30	24	18	12	6	0	-6
Tiempo total restante (real)	155	150	147	147	147	139	134	134	134	134	127	120	114	106	98	89	80	72	64	55	47	38	31	24	18	11	6	0	0	0

Figura 4.4: Sprint Backlog (Final)

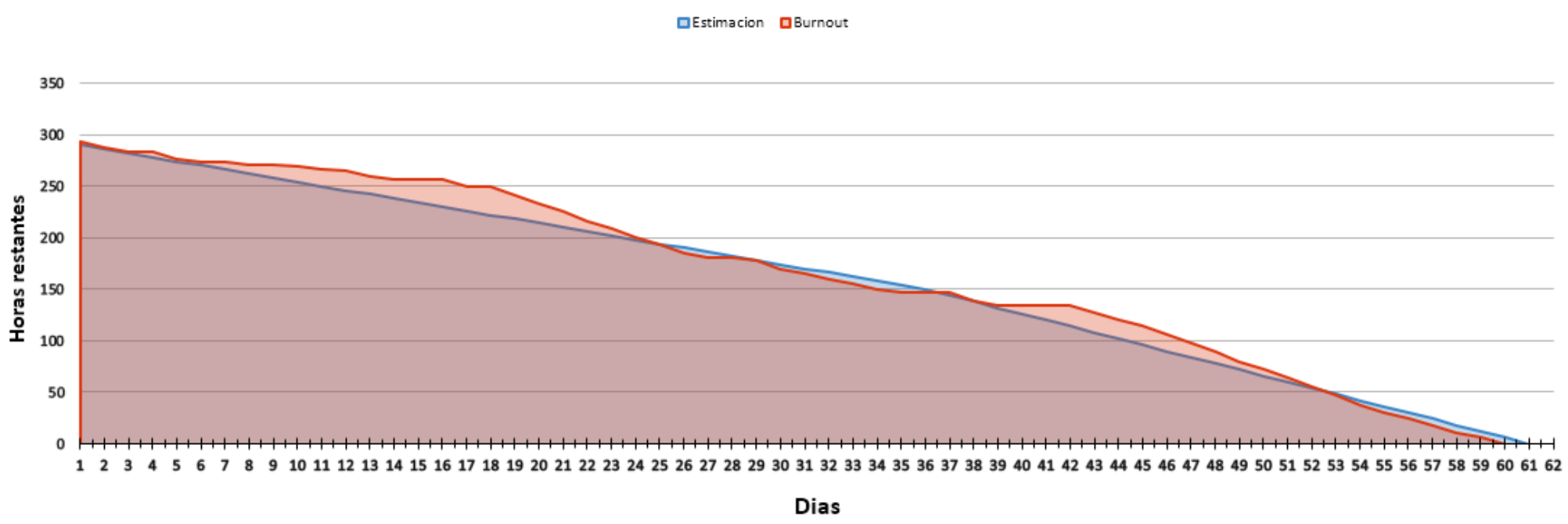


Figura 4.5: Burndown chart

5 Ejecución del proyecto

5.1 Estructura del proyecto

En este apartado se detallará la estructuración (figura 5.1) que se ha realizado a nivel de código en el proyecto. Todo el código que se ejecuta tanto en el lado cliente como en el lado servidor se encuentra en una carpeta que se subdivide en:

- **Public**, en esta carpeta se encuentran los únicos recursos que son **accesibles de forma pública** desde el navegador web del cliente, sin necesidad de estar autenticado o registrado. En este caso se encuentran las imágenes estáticas que se cargan en la web, el código Javascript compilado, que es el encargado de mostrar las interfaces y ofrecer la funcionalidad necesaria y el HTML que utiliza el código Javascript como plantilla.
- **Scripts**, en esta carpeta se encuentra el código y los datos necesario para realizar la migración de la base de datos en SQL a la versión NoSQL. Todos los datos se encuentra en formato JSON que ha sido exportado directamente desde el gestor de MySQL.
- **Src**, en esta carpeta se encuentra todo el código fuente que se ejecuta tanto en cliente como en el servidor. Se encuentra sin compilar y se distribuye a su vez en varias subcarpetas.
  - App, en esta carpeta se encuentra el código que se ejecuta en el navegador del cliente (ReactJS y GraphQL), es decir, toda la lógica de validaciones en cliente, interfaces y llamadas a API en el servidor.
  - Models, en esta carpeta se encuentran los modelos de objetos utilizados por el servidor, tal y como se detallaron en el diagrama de clases
  - Server, en esta carpeta se encuentra todo el código que se ejecuta en el servidor (NodeJS, Express, GraphQL y Mongoose), el cual es el encargado de procesar las peticiones de los clientes y solicitar los datos a la base de datos localizada en la nube
- Resto de archivos, el resto de archivos son archivos de configuración de los diferentes frameworks que se ejecutan en el sistema, el que tiene **más importancia es el package.json**, ya que es el encargado de recopilar todos los paquetes y las dependencias de Npm.

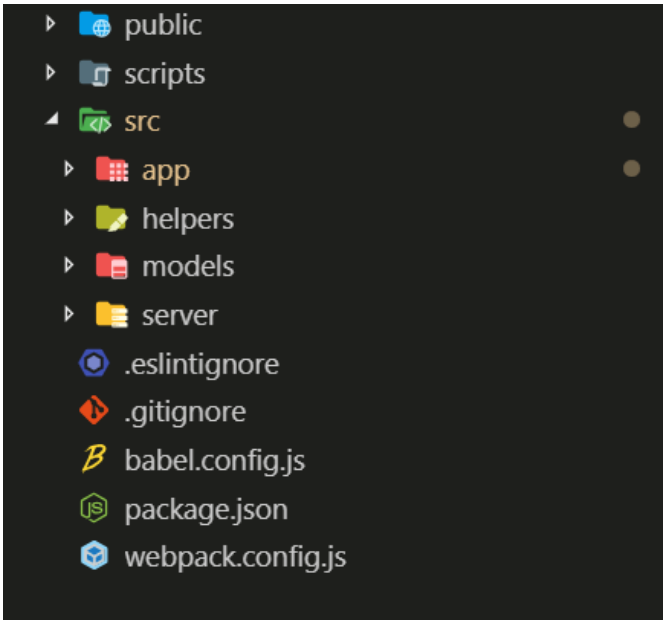


Figura 5.1: Estructura general del proyecto

5.2 Organización de la información

En este apartado se detallará de una forma más concisa como se organizan las funciones en React, que decisiones de diseño se tomaron y como funciona su **sistema de componentes**.

Una aplicación en React puede basarse en una **aplicación de una sola página**, es decir, existe un único archivo JavaScript que maneja toda la lógica de las interfaces del cliente, o bien una **aplicación de múltiples páginas** en la cual se pueden dividir en diferentes archivos los múltiples apartados de la página y que cada uno gestione su lógica.

En este caso se ha utilizado el modelo de una sola aplicación ya que por el alcance y las propiedades que ha de tener el sistema es mejor debido a:

- **Mejor velocidad**, debido a que se trata de una sola aplicación no se requiere actualizar toda la página para cambiar a otro apartado y solicitar otro archivo JavaScript por lo que mejora significativamente la velocidad de uso del sistema.
- **Mejor funcionamiento en móviles**, relacionado con lo anterior, en los dispositivos móviles es esencial el tiempo de carga y que la aplicación esté optimizada para estos dispositivos. Con un sólo archivo es mucho más sencillo que el rendimiento sea mejor.

La principal connotación negativa de este enfoque es que es peor para un **posicionamiento *SEO***, debido a que la mayoría de la página se carga dinámicamente y los buscadores no indexan muchas de las palabras que se generan de esta forma. Sin embargo, en este tipo de aplicación no es algo importante ya que no posee muchos textos y en principio no está enfocado a promocionarse de esta forma.

5.2.1 Estructura de un componente

El funcionamiento de una aplicación en React se basa en un archivo HTML que carga un archivo JavaScript con el código compilado. Este a su vez tiene un punto de entrada encargado de montar los diferentes componentes requeridos, como puede observarse en la figura 5.8.

Una vez que se ha cargado el archivo compilado, este a su vez arranca el código que hay en App.js, este archivo suele contener una librería denominada **Router React**, que es la encargada de resolver las rutas de la aplicación web. Un vez se resuelve la ruta, se cargan los componentes que estén asignados a la misma y estos **contienen la lógica y la interfaz** que se ha de mostrar al cliente.

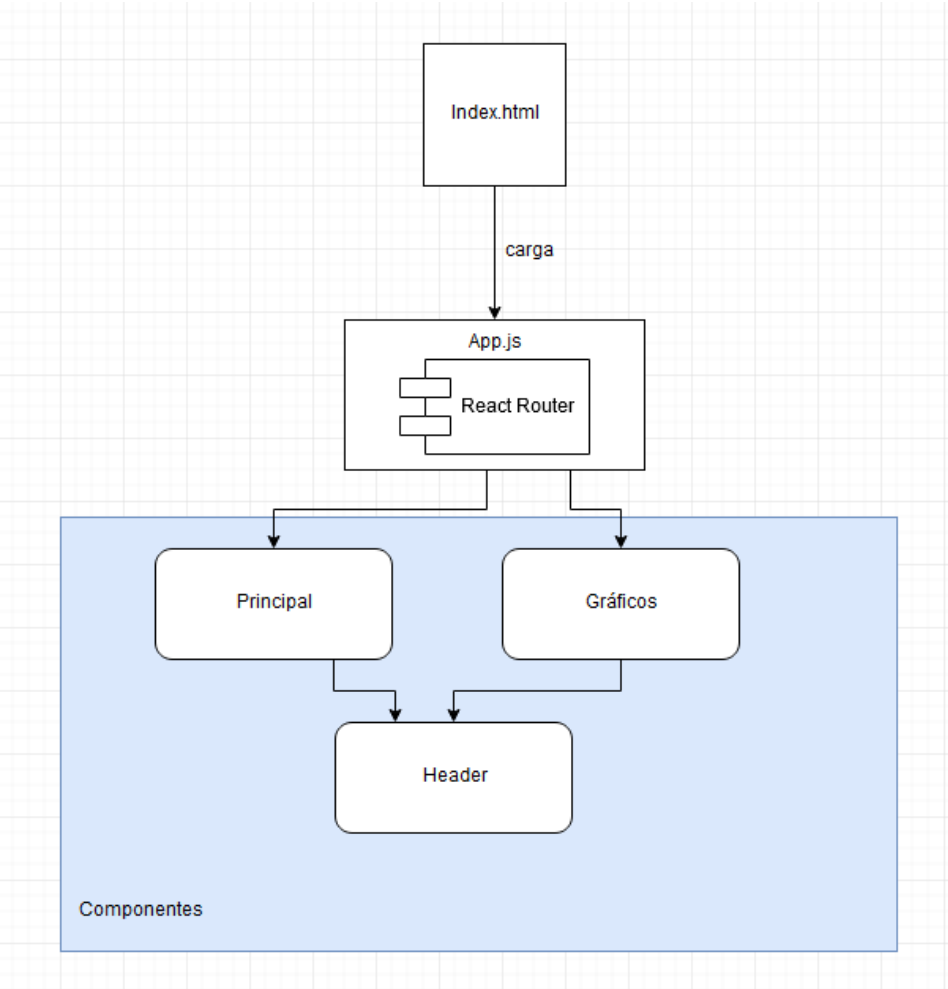


Figura 5.2: Diagrama funcionamiento app React

A su vez cada componente está conformado **principalmente por funciones** y variables, la función más importante que es la que suele llevar el nombre del propio componente es la encargada de devolver los datos que se le requieren o la interfaz solicitada.

5.3 Migración de la BBDD

Para la migración de la base de datos de MySQL se ha utilizado el gestor MySQL Workbench, el cual da una **serie de funcionalidades** para facilitar este proceso, te permite

realizar un diagrama de clases basado en las tablas y en los campos que conforman la base de datos. De esta forma se puede planificar de una mejor forma el nuevo modelo de objetos que se requiere.

Como se puede observar en la figura 5.3, el diagrama de clases era bastante confuso, y para la migración se optó por traspasar los datos más importantes. Una vez seleccionadas las tablas que se querían exportar, se utilizó otra funcionalidad que te permite exportar en formato JSON la estructura de las tablas junto con su contenido.

El principal problema de este modelo y que se ha tratado de solucionar mediante su migración a un modelo documental en NoSQL es la serie de relaciones entre tablas que provocaba tiempos largos de espera debido a la cantidad de operaciones JOIN que se debían ejecutar.

El cambio de modelo de datos fue principalmente en la clase proyecto, ya que esta es la que tenía más relaciones. Mediante su estructuración en un documento, se permite que tenga sus datos propios mejor estructurados en una única clase y **apuntadores a otros documentos**. De esta forma las operaciones JOIN dejan de cobrar sentido ya que es el propio documento el que tiene las direcciones de su información relativa a otras clases.

En la figura 5.4 se observa que hay datos en los cuales únicamente se muestra un identificador, esto es un apuntador a una referencia de otra clase que directamente posee esos datos.

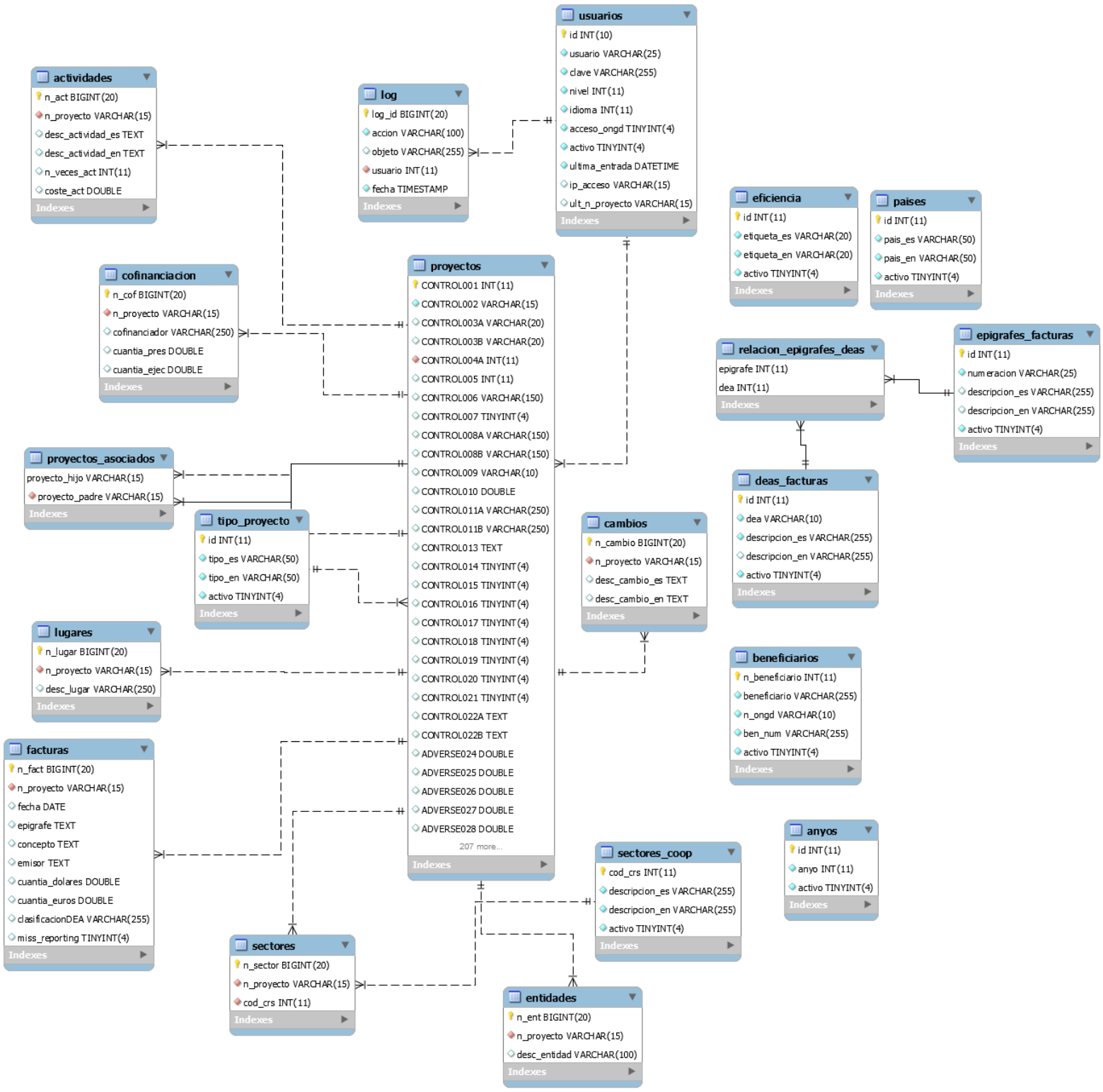


Figura 5.3: Diagrama clases MySQL

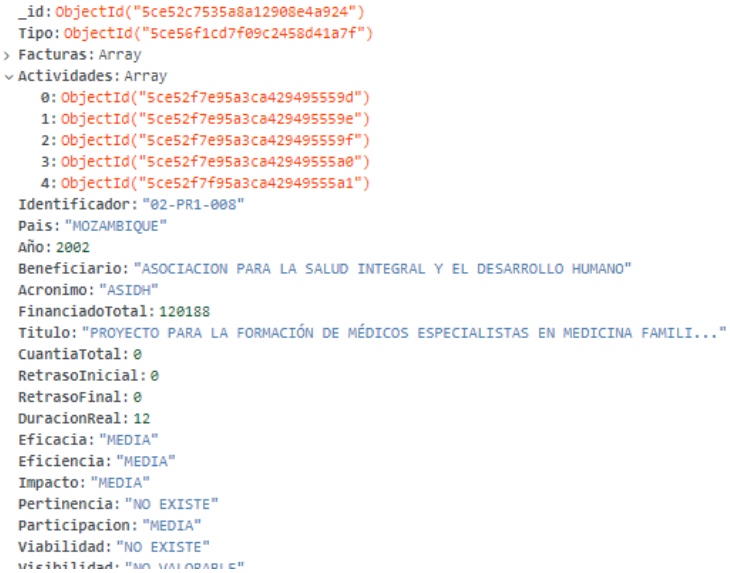


Figura 5.4: Estructura de la clase Proyecto en MongoDB

Una vez se extrajeron los datos en formato JSON, se codificó un script que los adaptaba con los nombres de los atributos escogidos para el modelo de este proyecto, para su posterior insercción en la base de datos NoSQL.

5.4 Interfaces gráficas

En este apartado se mostrarán algunas de las interfaces gráficas de las que dispone el sistema, tanto en su versión de escritorio como en su versión móvil.

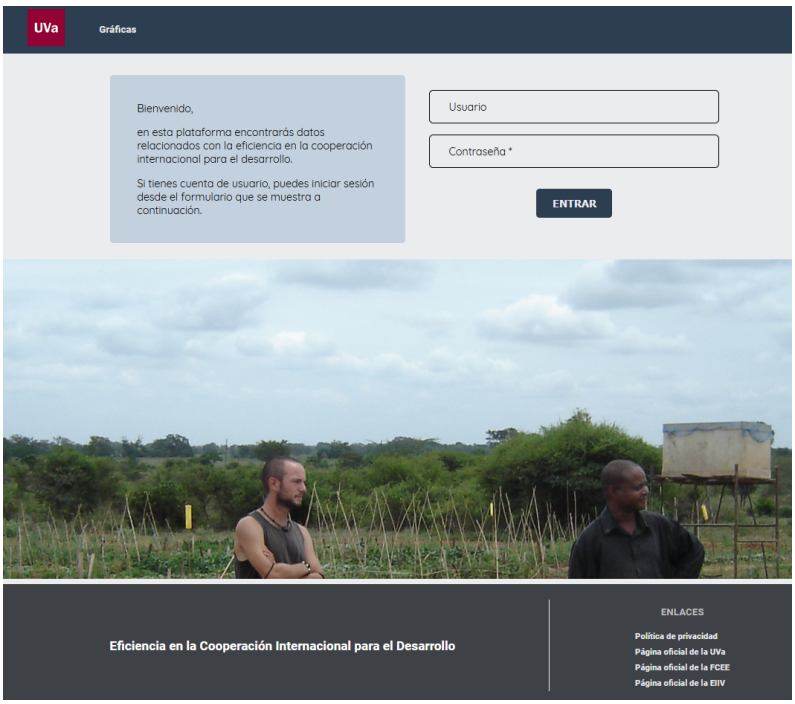


Figura 5.5: Interfaz página principal escritorio

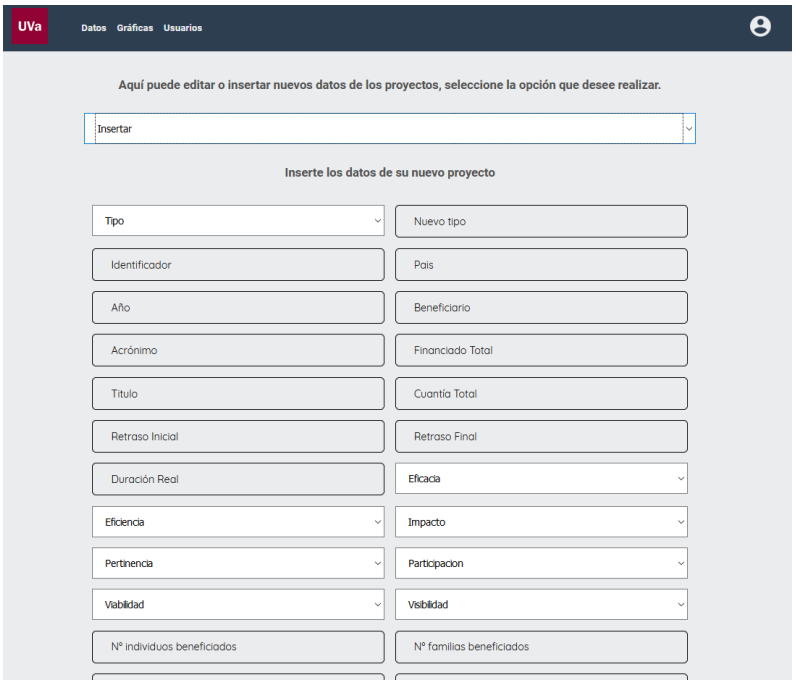


Figura 5.6: Interfaz inserción datos escritorio

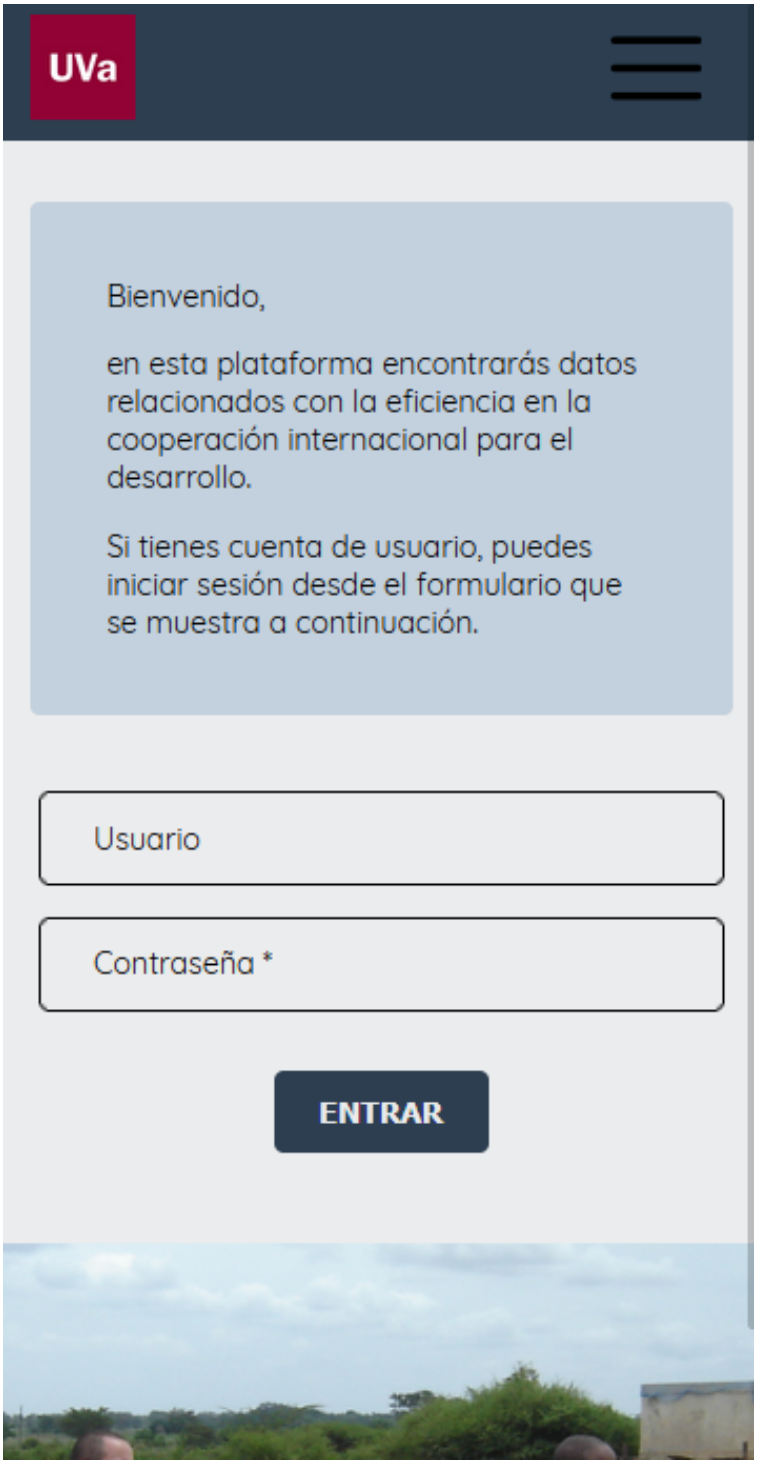


Figura 5.7: Interfaz página principal movil

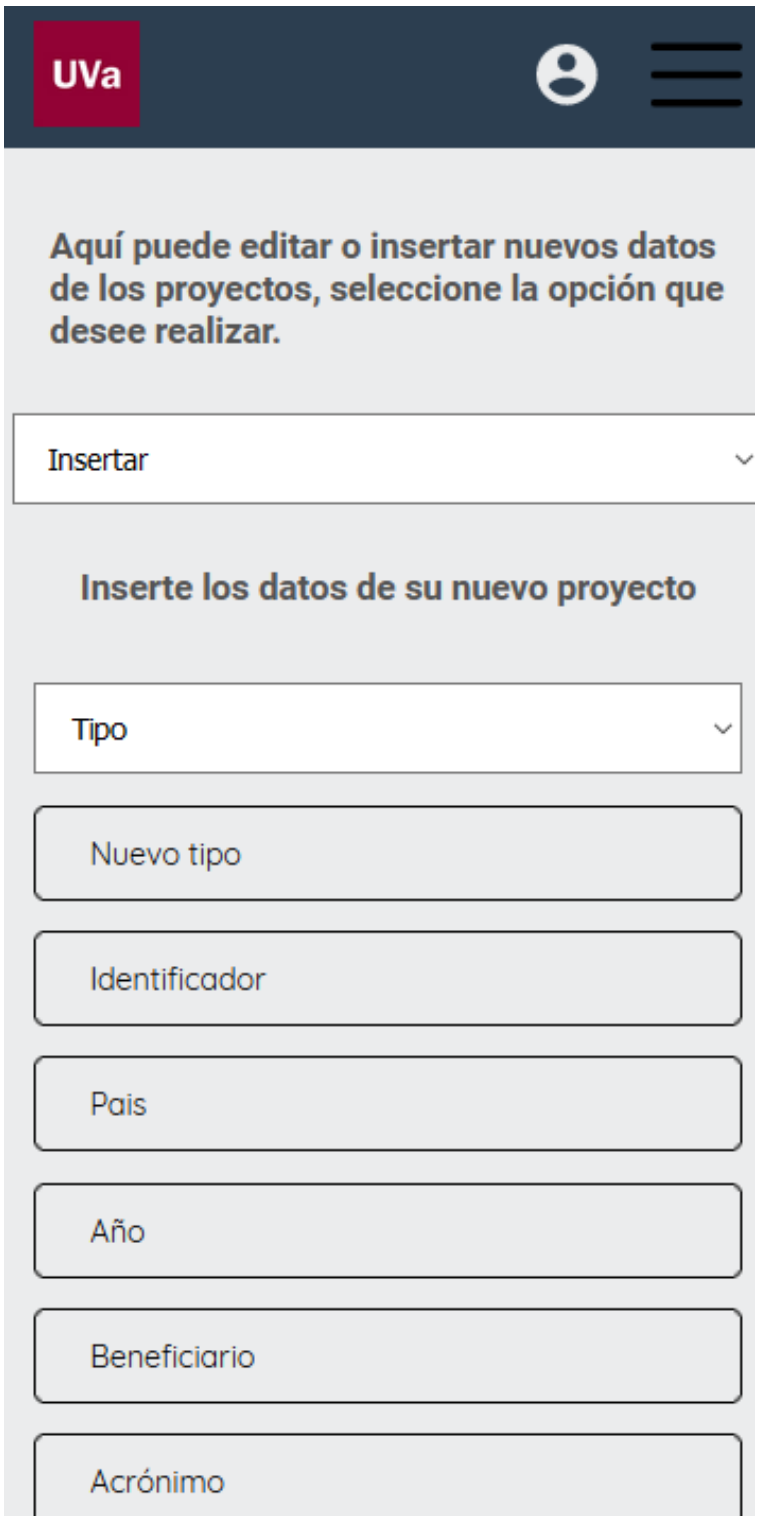


Figura 5.8: Interfaz insercción datos movil

6 Conclusiones

En este apartado se detallarán las conclusiones que se han llevado a cabo a lo largo del desarrollo del proyecto. Es importante destacar que todo el proyecto se basaba en un proyecto de fin de carrera bastante amplio.

El objetivo principal era **el rediseño de la plataforma** para que dentro de las funcionalidades descritas fuera realmente utilizado por el cliente, Natalia Cruz. En ese sentido el cliente está contento con el trabajo realizado por lo que puede considerarse que a pesar de tener un alcance limitado se ha cumplido el objetivo más inmediato.

Como se comentó anteriormente la principal motivación para realizar este trabajo era la **libertad** que me ofrecieron los tutores **a la hora de elegir** las tecnologías para desarrollarlo. Dándome la oportunidad de desarrollarlo en tecnologías basadas en JavaScript que actualmente están teniendo un crecimiento bastante notable.

Como principal obstáculo me encontré con que el **modelo de datos del proyecto anterior era demasiado amplio** y tuvimos que prescindir de datos, sin embargo no fue un inconveniente mayor ya que no ofrecían valor a las gráficas que eran una de las funcionalidades principales del sistema.

Respecto a la gestión también encontré ciertos problemas a la hora de planificar el proyecto ya que es difícil estimar el número de horas que te puede llevar cada tarea teniendo en cuenta el marco en el cual se desarrolla este proyecto. Sin embargo, debido a la metodología adaptada y aunque no se aplicó como tal SCRUM, sí que se pudieron asumir los riesgos de una mala planificación inicial.

Por la parte de desarrollo, es muy cómodo codificar todo el sistema en prácticamente un lenguaje de programación, esto agilizó bastante ciertas partes que a priori pueden parecer más complejas si se hubiera optado por otras tecnologías. Además los componentes reutilizables permiten un ahorro de trabajo considerable y un posterior mantenimiento más eficiente.

Además de esto se pudo comprobar la eficiencia del sistema a la hora de generar gráficos lo cual fue bastante satisfactorio ya que en el proyecto anterior generar ciertos informes requería de un tiempo bastante excesivo, en este sistema, la generación de gráficas no supera los máximos establecidos para ninguno de los parámetros escogidos, a pesar de la cantidad de datos que debe consultar.

6.0.1 Trabajo futuro

Por último conviene añadir que en un principio tanto los tutores como yo considerábamos un mayor número de funcionalidades en el sistema que por tiempo no se han podido implementar. Se considerará **mejorar ciertos aspectos** de la aplicación de cara a la presentación de la misma, implementando un **cifrado** en la aplicación y una mejor **gestión de los usuarios** ya que en este momento se debe hacer desde la base de datos.

7 Glosario

- Framework, es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.
- Open-source, modelo de desarrollo de software basado en la colaboración abierta.
- HTTP, o protocolo de transferencia de hipertexto (en inglés: Hypertext Transfer Protocol) es el protocolo de comunicación que permite las transferencias de información en Internet
- XML, eXtensible Markup Language, traducido como ”Lenguaje de Marcado Extensible” o ”Lenguaje de Marcas Extensible”, es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.
- JSON, acrónimo de JavaScript Object Notation, «notación de objeto de JavaScript» es un formato de texto sencillo para el intercambio de datos.
- API, es una interfaz de programación de aplicaciones (del inglés API: Application Programming Interface). Es un conjunto de rutinas que provee acceso a funciones de un determinado software.
- SEO, optimización en motores de búsqueda (del inglés search engine optimization), es un conjunto de acciones orientadas a mejorar el posicionamiento de un sitio web en la lista de resultados de los buscadores de Internet

Bibliografía

Antonio Fuentes Pérez. *Plataforma UVa-AECID:Eficiencia en la Cooperación Internacional para el Desarrollo*. Universidad de Valladolid, 2009.

José Manuel Alarcón. Gráfico webpack, 2017. URL `https://www.campusmvp.es/recursos/post/webpack-que-es-para-que-sirve-y-sus-ventajas-e-inconvenientes.aspx`.

Diagrama graphql/rest. URL `https://miro.medium.com/max/1838/1*fKhxPG5PQ55kDika_UVukw.png`.

Node.js Foundation. Documentación de nodejs, 2009. URL `https://nodejs.org/es/`.

Load Impact. Gráfica comparativa apache y nodejs, 2013. URL `https://blog.loadimpact.com/blog/node-js-vs-php-using-load-impact-to-visualize-node-js-efficency/`.

Kevin Kononenko. Diagrama express.js. URL `https://i2.wp.com/blog.codeanalogies.com/wp-content/uploads/2017/11/92ebf-08hizvtx-da3c26uv.png?w=730&ssl=1`.

Victoria Malaya. Gráfica comparativa modelos sql y nosql, 2013. URL `http://sql-vs-nosql.blogspot.com/2013/11/indexes-comparison-mongodb-vs-mssqlserver.html`.

Nick Karnik. Diagrama mongoose, 2018. URL `https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/`.

NPM. Documentación de npm y el manual de usuario, 2013. URL `https://www.npmjs.com/package/forever`.

Mark Richards. *Software Architecture Patterns*. O'reilly, 2015.

Wikimedia Commons. Diagrama rup, 2008. URL `https://commons.wikimedia.org/w/index.php?curid=4277122`.

MongoDB Inc. Documentación de mongodb y mongodb cloud, 2009. URL `https://www.mongodb.com/`.

Grady; RUMBAUGH James. JACOBSON, Ivar; BOOCH. *El Proceso Unificado de Desarrollo de Software*. Pearson Addisson-Wesley, 2000.

Vishal Rambhiya. Principales diferencias entre php y nodejs, 2018. URL `https://www.websoptimization.com/blog/what-is-the-difference-between-php-and-node-js/`.

Michael Abernethy. Documentación genérica acerca de nodejs, 2011. URL `https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/index.html`.