Universidad del País Vasco
Euskal Herriko Unibertsitatea

Facultad de Informática

Grado de Ingeniería Informática

▪ Proyecto Fin de Grado ▪

Ingeniería del Software

PROPUESTA PARA EL DESARROLLO COLABORATIVO DE TFGS INSPIRADO EN LA REALIZACIÓN DE UNA APLICACIÓN OPEN SOURCE MÓVIL PARA LA GESTIÓN PERSONALIZADA DE HORARIOS DE ESTUDIANTES DE LA UPV-EHU

Autor: Joseba Carral Del Castillo

Director: José Miguel Blanco

Junio 2018

informatika fakultatea
facultad de informática

Agradecimientos

Antes de empezar, quiero aprovechar las siguientes líneas para agradecer el apoyo recibido por las personas de mi entorno durante mi etapa en la universidad, sin el que probablemente este trabajo no hubiera sido posible.

Primero, quiero agradecer a mi tutor, José Miguel Blanco, por la confianza mostrada desde el principio del proyecto. Tengo que agradecerle toda la ayuda que me ha ofrecido y la paciencia estos meses. Gracias también por lo que me has enseñado estos años, no solo en el ámbito académico. No puedo olvidarme del apoyo que nos diste, a mi y a mis compañeros, cuando intentamos formar un grupo de trabajo, ABJ o CoDevs.

También tengo que agradecer a MagnaSIS, empresa junior de la Universidad del País Vasco (UPV-EHU), que me ofreció hacer las prácticas de empresa hace 2 años. Gracias a esa colaboración surgió la idea de este proyecto.

A mis amigos y familia, por el apoyo y paciencia que han tenido durante estos años, por la ayuda que me han ofrecido.

Tengo que agradecer en especial a Borja y Olatz, quienes además me han ayudado con el desarrollo del proyecto. Por sus consejos y por participar convirtiéndose en colaboradores de la aplicación.

En general, gracias a todos que me han apoyado para sacar este trabajo adelante.

Resumen

Este documento contiene la memoria del trabajo de fin de grado que trata el desarrollo de un sistema para la extracción de datos de la web de la UPV-EHU y el uso de estos para generar una herramienta orientada al estudiante.

Esta herramienta se presenta como una aplicación móvil, disponible en Android e iOS, donde el alumno puede consultar información sobre la universidad y crear de forma automática horarios basándose en sus preferencias.

El objetivo principal de este proyecto es realizar el desarrollo siguiendo un modelo *open source* del que finalmente se pueda obtener una aproximación para establecer las bases de una metodología para el desarrollo de futuros trabajos de fin de grado (TFGs) utilizando repositorios abiertos alojados en GitHub fomentando así la colaboración.

Como resultado de este trabajo se han generado una colección de repositorios, accesibles a través de GitHub, donde se encuentra tanto la documentación como el código desarrollado, además de la propuesta de modelo *open source* para futuros trabajos de fin de grado.

Índice

Lista de Figuras

Lista de tablas

Lista de programas

1

Introducción

A continuación, se describe de forma breve el trabajo de fin de grado al que hace referencia este documento. En este capítulo se mencionan los antecedentes, los objetivos y los resultados obtenidos. Además, se describen los apartados del documento.

1.1. Introducción

La idea de este proyecto nace en el verano del 2016 durante las prácticas de empresa realizadas en Magna SIS. Por aquel entonces, se propuso realizar una herramienta con la que se obtuvieran los horarios de las asignaturas de la Facultad de Informática de San Sebastián (FISS) y que permitiese a los usuarios poder crear horarios personalizados con las asignaturas en las que se encontraban matriculados. Desafortunadamente, no se lograron los objetivos propuestos. A pesar de no cumplir con lo especificado se logró generar una versión simplificada de una librería para obtener datos de la web de la universidad.

Con la experiencia descrita en mente y después de una reunión con José Miguel Blanco, tutor del proyecto y supervisor de aquellas prácticas, se decide retomar aquella fallida idea con la idea de mejorarla. Puesto que el modelo de desarrollo *open source* ha servido como sistema para mi formación complementaria, se decide realizar el desarrollo siguiendo este modelo y estudiar la posibilidad de generar una metodología *open source* aplicable a futuros trabajos de fin de grado de la FISS.

Con todo esto, se marcan dos objetivos principales para este proyecto. El primero es realizar el desarrollo siguiendo un modelo *open source* para crear un sistema para la extracción de datos de la web de la UPV-EHU y usar esta información mediante un cliente móvil. La aplicación desarrollada debe permitir al estudiante poder consultar la información de la universidad y generar un horario personalizado de forma automática. El segundo objetivo es realizar una primera aproximación de una propuesta de desarrollo *open source* aplicable a futuros trabajos de fin de grado de la FISS, de forma que se fomente la colaboración y el acceso a los materiales generados durante estos trabajos.

Como resultado de este proyecto, se obtienen los entregables propuestos, disponibles en una colección de repositorios en GitHub. Por un lado, se han conseguido generar tres módulos independientes donde se encuentra toda la implementación del sistema. El primero es una librería responsable de extraer los datos de la web de la UPV-EHU. El segundo paquete es el servidor, donde está parte de la lógica de negocio y gestión de la base de datos. El último repositorio, es el que contiene el cliente móvil que permite al estudiante gestionar la información de la universidad.

Este desarrollo se ha realizado de forma open source, por lo tanto, todos los repositorios tienen su propia documentación además de una wiki común donde se detalla todo el sistema, desde cómo se ha implementado hasta cómo se configura. En base al trabajo realizado se ha podido generar una propuesta de modelo para utilizarla en futuros TFG.

En la documentación se describe todo el proceso que se ha seguido durante el desarrollo de este trabajo. Esta memoria está compuesta de 10 capítulos, siendo la introducción el primero de ellos.

En el capítulo de antecedentes se describe el contexto del proyecto, se introduce el modelo *open source* y la aplicación a desarrollar. A continuación, en los objetivos, se detalla cual es el alcance que se define para esta etapa del proyecto, las exclusiones y los objetivos generales.

Tras estos tres primeros apartados donde se introduce el proyecto, llega el capítulo en el que se describen los cuatro pilares de los proyectos open source. Después, en el capítulo quinto se realiza un análisis del producto donde se muestra la arquitectura, los casos de uso a implementar y el modelo de datos.

El sexto punto del documento es el que muestra cuál ha sido la implementación. Para ello se empieza primero explicando cuales son las tecnologías utilizadas y luego las implementaciones de cada módulo.

En el siguiente apartado, el séptimo, se describe el uso o soporte del proyecto en GitHub. Aquí se muestra cómo se ha utilizado esta plataforma para documentar los repositorios con el código.

Tras haber realizado la implementación y la documentación de esta, se realiza la propuesta de un modelo aplicable a trabajos de fin de grado. Esta propuesta esta basada en los puntos anteriores.

A continuación, se explica la gestión del proyecto. Aquí se muestra la gestión del alcance, tiempo, comunicaciones y riesgos.

Por último, en el décimo capítulo, se tratan las conclusiones obtenidas con el desarrollo de este trabajo y las posibles mejoras futuras.

2

Antecedentes

En este segundo apartado del documento se describe el contexto del proyecto. Está dividido en dos bloques, los antecedentes relacionados con el modelo de desarrollo open source y los que están vinculados a la aplicación para la gestión de los horarios de la universidad.

2.1. OPEN SOURCE

El software *open source* es aquel que tiene su código fuente público, disponible para cualquier usuario. Permite su copia, modificación y distribución cumpliendo los términos especificados en las licencias de uso.

Es un modelo de trabajo especialmente aplicado en el desarrollo software pero que se puede extrapolar a otras áreas como por ejemplo la investigación en ciencia.

Este modelo de trabajo tiene beneficios tanto para desarrolladores como para las empresas que lo adoptan.

En el caso de los desarrolladores es una forma de mejorar la forma de trabajar. Escribir código que otros desarrolladores van a leer obliga a mejorar la forma de estructurar los proyectos y hacer el código más legible. La posibilidad de trabajar con personas de diferentes partes del mundo permite fortalecer una competencia como es el trabajo en equipo y la comunicación. Es una forma de mejorar como desarrollador aprendiendo de la experiencia de otros usuarios.

En cuanto a las empresas, los beneficios que aporta este modelo de trabajo son múltiples. Como ejemplo ser la importancia de la reducción de costes, ya que generalmente son aplicaciones gratuitas que eliminan gastos que pueden provenir del uso de software propietario como el pago por mejoras o mantenimiento. Otros aspectos positivos son la seguridad que ofrece el uso de software totalmente auditable o el mantenimiento, que es independiente del propietario. La disponibilidad del código permite modificar y personalizar las aplicaciones para adaptarlas a las necesidades de la empresa.

Es conocida la fundación linux por ser una de las impulsoras del uso de tecnologías *open source.* Es un movimiento cada vez más extendido pues grandes compañías como Microsoft, Facebook, Google o Amazon utilizan plataformas como GitHub para publicar sus productos de forma pública.

Y es que GitHub es una de las grandes razones del aumento de proyectos open source. Se trata de un gestor de repositorios de proyectos que utilizan Git, un software de control de versiones, que conecta a millones de desarrolladores, independientemente de lenguajes de programación o tecnologías que utilicen.

Según datos publicados por la propia organización, durante el 2017 hubo 26 millones de repositorios con actividad. Entre estos repositorios, los más activos son los de empresas como Facebook y Google, que gracias al aporte de miles de usuarios de todo el mundo han lanzado y mejorado sus productos.

2.1.1. GitPoint

Dentro de GitHub hay almacenados millones de repositorios con proyectos colaborativos. Escoger un único ejemplo para utilizarlo como referencia de buenas prácticas acaba siendo una decisión completamente arbitraria pues la cantidad de candidatos que cumplen con los mínimos requeridos es interminable. Para este trabajo se utiliza como ejemplo la aplicación *GitPoint* en la que se ha colaborado traduciéndola al euskera.

Figura 1: Logo de GitPoint

GitPoint es una aplicación móvil desarrollada utilizando *React Native* y que está disponible para los sistemas operativos más importantes.
El objetivo de este proyecto es desarrollar un cliente móvil de la plataforma GitHub para que el usuario pueda gestionar su actividad en cualquier dispositivo móvil.
Tiene una comunidad de hasta 80 personas que han colaborado en el desarrollo de la aplicación. El que haya un número de contribuidores tan elevado es debido en gran parte a la documentación y a los responsables de mantener el proyecto.
Todo el código está disponible en su repositorio de GitHub que contiene la documentación donde se describe el proceso para colaborar e introduce al desarrollador el funcionamiento del código. Además, también dispone de una página web orientada al usuario final de la aplicación.
Gracias a esta documentación, y después de un tiempo siendo usuario de la aplicación, decidí colaborar traduciendo el contenido al euskera.

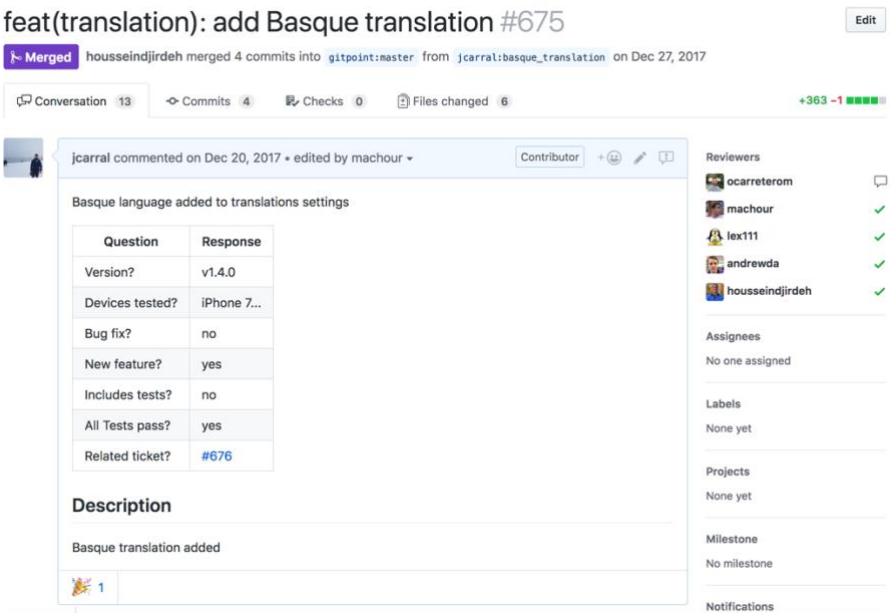


Figura 2: Issue abierto con la traducción de GitPoint

2.2 APLICACIÓN UNIVERSIDAD

La UPV-EHU cuenta con diversas alternativas para la consulta de los horarios de las asignaturas: hacer uso de la web general mediante GAUR o con la herramienta WebUntis. Por lo general el alumno lo que busca es una forma rápida, sencilla y con posibilidad de personalización.

Aunque con las aplicaciones actuales y con herramientas de terceros se pueden generar calendarios personalizados y realizar consultas puntuales sobre las asignaturas, el proceso para llevarlo a cabo es demasiado largo.

Con el fin de mejorar este problema, en verano de 2016 durante el periodo de prácticas abierto por MagnaSis se propuso generar una solución. Los meses de mayo, junio y julio un grupo de 8 alumnos del que formaba parte trabajó a diario para diseñar una solución.

Aquel proyecto quedó como un prototipo, no se obtuvo un producto, aunque se llegó a generar una versión de prueba. El resultado fue una web que solo permitía comprobar el horario de una asignatura.

Han pasado dos años y la consulta de horarios de asignaturas o tutorías sigue siendo una de las acciones más recurrentes por los alumnos.

Ante la falta de una aplicación o sistema que facilite este trabajo se muestra una oportunidad de generar una aplicación de código abierto que mejore las alternativas actuales gracias a la colaboración de los propios estudiantes, ya sea con el aporte de ideas como en la implementación de la propia herramienta.

3

Objetivos

En este capítulo se describen los objetivos principales del proyecto. Estos son dos, crear una aplicación para la gestión de los horarios de la universidad y obtener un modelo de desarrollo open source aplicable en los trabajos de fin de grado. El contenido está dividido en cuatro apartados. Primero se exponen los objetivos del proyecto. A continuación, se describe el alcance del proyecto, explicando cuales son los requisitos funcionales y no funcionales. Luego, se enumeran las exclusiones del proyecto. Por último, se muestra el EDT con los paquetes de trabajo del TFG.

3.1 Objetivos del proyecto

Los objetivos del proyecto son varios: por un lado, se encuentran los relacionados con el desarrollo del sistema para la generación de calendarios, *EHUCalendarGenerator*, y por el otro el del modelo *open source, TFG-OpenSourceModel*.

El primer objetivo es realizar el desarrollo de un conjunto de herramientas siguiendo un modelo open source, estando disponibles en GitHub todos los repositorios. Partiendo de un diseño inicial de la librería *ehu-scraping*, se busca adaptarla y mejorarla para poder extraer la información de la web de la UPV-EHU.
Una vez la herramienta de *scraping* está implementada, el trabajo consiste en hacer uso de esta librería para desarrollar una aplicación móvil que permita a los alumnos consultar la información sobre las asignaturas y profesores de la UPV-EHU.
La aplicación debe permitir que los usuarios puedan suscribirse tanto a asignaturas como a profesores. Así podrá generarse un calendario personalizado con las clases a las que acuden.
La navegación debe ser fluida e intuitiva para mejorar las alternativas ya existentes.
Dada la cantidad de grados, asignaturas y profesores, resulta imprescindible que la obtención de datos sea un trabajo automático. El sistema es quien debe obtener la información que se encuentra pública en la web de la universidad. Además, cualquier modificación debe también verse reflejada en la aplicación. De esta forma, se reducen las responsabilidades del administrador.
El desarrollo debe ser modular, de forma que la aplicación móvil pueda ser reutilizable en implementaciones para otras universidades, no limitándose exclusivamente a la UPV-EHU.

El segundo objetivo importante de este trabajo es la realización de una primera aproximación de una propuesta de modelo open source para trabajos de fin de grado. Se trata de un proyecto con visión de futuro que busca fomentar la colaboración y el acceso a los materiales generados en los trabajos.

Como resultado de los objetivos anteriores, se busca generar una colección de repositorios en GitHub donde se encuentren de forma pública todos los paquetes desarrollados, con su código y documentación, y un repositorio con la propuesta de modelo. Además del código se generará también una versión funcional de la aplicación.

3.2 Alcance

El alcance de este proyecto va más allá de los plazos para este trabajo de fin de grado y por eso está limitado. De la misma forma que la idea de generar una aplicación para la universidad viene de antes, el futuro no se queda en lo que se desarrolle a lo largo de este TFG. El objetivo es que una vez finalizado este periodo, se mantenga un proyecto abierto donde el resto de los estudiantes puedan colaborar en el desarrollo de la aplicación o en la mejora del modelo de trabajo de fin de grado open source.

De aquí en adelante, todo lo relacionado con el alcance del proyecto se refiere únicamente a la etapa del trabajo de fin de grado.

3.2.1 Requisitos funcionales

- El sistema debe permitir a los usuarios registrarse en la aplicación.
- El sistema debe restringir las acciones en función del nivel de autorización del usuario.
- El usuario debe poder consultar y guardar tanto asignaturas como profesores.
- El usuario puede consultar su horario personal.
- El sistema debe permitir que el usuario modifique su perfil. Debe poder modificar desde los datos personales hasta las credenciales de acceso.
- El sistema debe permitir actualizar el contenido de la base de datos desde la propia aplicación.
- El sistema debe permitir cambiar el idioma de la interfaz.

3.2.2 Requisitos no funcionales

En este apartado se describen los requisitos no funcionales de la aplicación y de la propuesta de modelo open source para los TFGs.

- Todos los repositorios generados deben ser públicos.
- Las tecnologías utilizadas vienen dadas desde el inicio del proyecto y por lo tanto no se necesita de un análisis de alternativas.
- Los repositorios están abiertos a propuestas y mejoras de terceras personas desde el inicio del proyecto.
- La aplicación debe estar disponible para los dos principales sistemas operativos móviles, Android e iOS.

3.3 Exclusiones

Dado que se trata de un proyecto a largo plazo y que el resultado de este trabajo no es más que un prototipo de una versión funcional quedan excluidos del trabajo los siguientes aspectos:

- Estudio del impacto del Reglamento General de Protección de Datos (RGPD).
- Publicación de la aplicación en las tiendas oficiales.
- Gestión de adquisiciones.
- Monitorización y análisis de rendimiento
- Traducción de la documentación de los repositorios al euskera e inglés.
- Se excluyen de la gestión del tiempo las horas que se vayan a invertir para la revisión de la memoria y la preparación de la defensa del TFG.

3.4 EDT

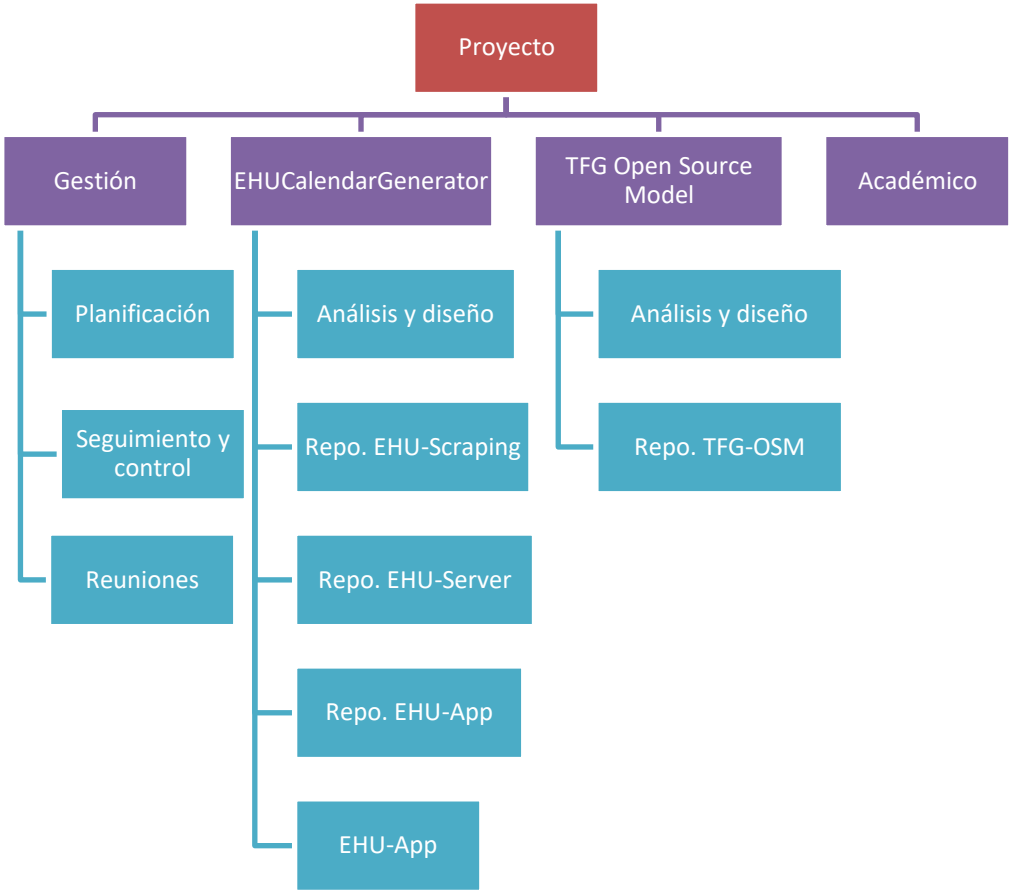


Figura 3: EDT del proyecto.

4

Open Source

En este capítulo se describen los aspectos más importantes de un proyecto *open source*.
El contenido está estructurado en cuatro bloques que hacen referencia a cada uno de los pilares más importantes para el desarrollo de un proyecto de código abierto.

4.1 Control de versiones

La gestión de versiones es una tarea vital en cualquier proyecto de desarrollo software. Independientemente de si se trata de un proyecto individual o grupal, se necesita mantener diferentes versiones del código. Para esta tarea existen diferentes sistemas que nos facilitan el trabajo como son *Git*, *CVS*, *SourceSafe* o *Subversion*. Las principales ventajas de este tipo de herramientas son:

- Mantener diferentes estados del código.
- Capacidad para recuperar versiones antiguas.
- Utilizar diferentes ramas.
- Fusionar cambios entre las diferentes ramas de trabajo.
- Mantener un registro de las personas que han colaborado en el proyecto.

4.1.1 Git

Git es la herramienta de control de versiones de software más popular de todas, fue diseñada por Linus Torvalds.
La principal característica de Git es que es distribuido y no necesita un servidor central puesto que cada desarrollador tiene una copia local donde trabaja y desde donde puede propagar los cambios. Con esto se consigue una evidente mejora en el rendimiento en proyectos grandes, consiguiendo así búsquedas más eficaces para detectar los cambios.

La otra gran ventaja es la flexibilidad en la gestión de las ramas o *branches*. Estas se pueden crear de forma local, independientes del repositorio centralizado donde cada una tiene uso diferente, permitiendo así crear diferentes flujos de trabajo.

4.1.2 Flujo de trabajo

Al no existir una forma de trabajo predeterminada, a la hora de trabajar con las ramas son los propietarios los que han de escoger cual es el flujo de trabajo que se va a seguir. Existen diferentes alternativas en la comunidad de desarrolladores.

Una de ellas es el modelo centralizado, el más simple. Se trata de una única rama *master* en el repositorio remoto sobre la que todos los colaboradores realizan sus *commits*. A pesar de ser la menos compleja, es la que más problemas puede generar a la hora de fusionar el trabajo y mantener un orden.

Otra opción, una de las más extendidas, diseñada por Vincent Driessen y recomendada por Atlassian es la que se trabaja con una serie de ramas en torno a los lanzamientos del proyecto. Por un lado, están las ramas *master*, *develop* y *releases*.

- **Releases** es la que tiene el proyecto con el código preparado para publicar.
- **Master** es el paso previo. El software ya en producción estará utilizando la última versión de la rama *releases* aunque el código es perfectamente funcional.
- **Develop**, la tercera de estas ramas es sobre la que se trabaja en la fase de desarrollo, el código es inestable y no puede considerarse una versión funcional del producto.

Además de estas tres ramas principales existen dos grupos de ramas secundarias:

- Las ramas para **funcionalidades** son las que partiendo de la última versión de *develop,* se empieza a trabajar de forma independiente para añadir una nueva utilidad. Cada funcionalidad tiene su propia rama y una vez que esta se termina de implementar pasa a fusionarse con *develop*.
- El otro grupo es el de los **errores**. Cuando se detecta algún tipo de problema en el código y con el fin de solventarlo, se realiza una copia de la rama *master* donde se ha encontrado el fallo y se trabaja sobre el duplicado para corregir cualquier incidencia. Una vez está resuelto el problema se vuelve a fusionar la copia con la rama *develop* para volver a empezar con el proceso de publicación.

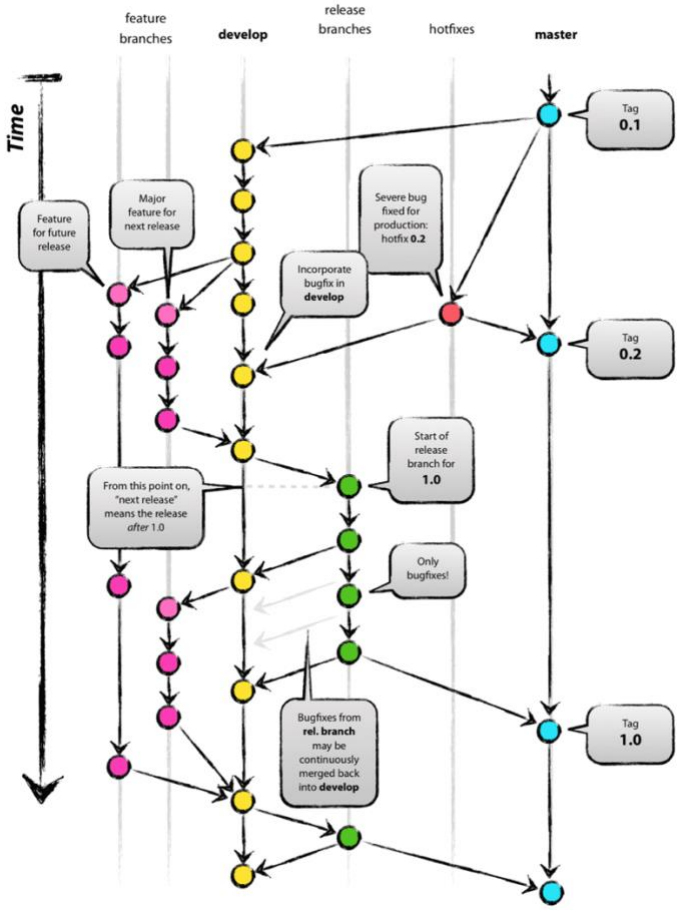


Figura 4: Workflow propuesto por atlassian.

A pesar de esto, la decisión de definir el modelo a seguir sigue siendo de los responsables del proyecto y son estos quienes deben definir un flujo de trabajo para mantener un orden, pero adaptándose a las necesidades y requisitos del proyecto. Es importante tener en cuenta que cuanto más complejo sea el sistema, aunque ofrezca más beneficios, también requerirá un mayor aprendizaje de la herramienta y una mayor disciplina de trabajo.

El diseño de las ramas no es el único aspecto a decidir cuando se trata de diseñar un flujo de trabajo, también hay que especificar la nomenclatura de éstas y la forma de fusionarlas.
En cuanto a nombrar las ramas, es una buena práctica el uso de prefijos para cada tipo de etiqueta, así pues, una rama para añadir una funcionalidad puede empezar llamándose *feature-* o *feature/*, para los errores utilizar el prefijo *hotfix-* o *bug-*.

En la fusión de ramas existen dos variantes. La primera es la que está configurada por defecto y la que cuando se juntan dos ramas se mezclan también todos los *commits* que se han hecho en éstas. Esta opción es problemática, pues si se quiere volver a alguna versión antigua obliga al desarrollador a revisar los *commits* de todas las ramas y deshacerlos.
La segunda opción, la más común, es la que se conoce como *fast-forward* y en la que, a diferencia de la anterior, cuando se fusiona una rama sobre otra lo que hace es añadir el último *commit*, sin perder la rama secundaria con los cambios.

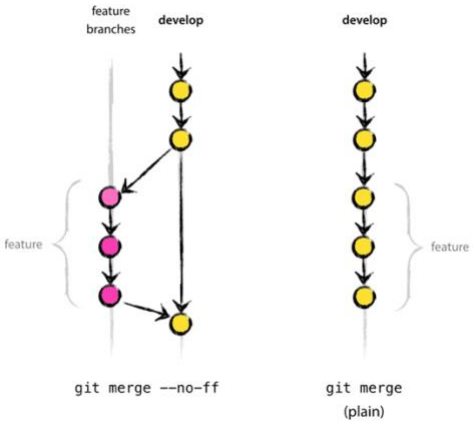


Figura 5: Diferencias entre los tipos de fusiones

4.2 Documentación

Uno de los aspectos esenciales para conseguir que un trabajo sea atractivo, e invite a la colaboración, es la documentación. Todo proyecto debe tener una guía explicando sus funcionalidades y otra explicando cuales son los pasos que realizar y las normas para trabajar en el mismo proyecto.

4.2.1 Formato de la documentación

En cuanto al formato, existen diferentes alternativas para mostrar la documentación que introduciremos, brevemente, a continuación.
La primera y más habitual es la de almacenar los ficheros dentro del propio repositorio en formato *markdown*.
Todos los gestores de repositorios permiten visualizar estos documentos. Cuando se trata de trabajos más pequeños esta es la mejor opción.

Dentro de esta alternativa se pueden guardar todos los archivos en el mismo repositorio, organizándolo en directorios, o aprovechar las wikis, funcionalidad que también implementan gestores como GitHub o GitLab y que permite ordenar todo de una forma más estructurada.

La segunda opción es la de utilizar una página web o blog independiente al repositorio. La elección de utilizar un sitio personal facilita al usuario la visualización de la documentación. Esta solución suele estar enfocada a proyectos más grandes.

La última opción es la de combinar las dos alternativas, mantener la documentación en el repositorio y con los mismos ficheros generar un sitio web. GitHub tiene una funcionalidad llamada GitHub Pages que convierte un repositorio en una web utilizando html o directamente convirtiendo los archivos markdown gracias a otras herramientas como GitBook. Esta elección es la que siguen compañías como Facebook, IBM, Twitter o Microsoft.

4.2.2 Guía de colaboración

Esta guía es el conjunto de ficheros que deben facilitar al colaborador trabajar con la herramienta, explicando desde cómo debe instalar y configurar el software para empezar a trabajar en él, hasta los pasos que tiene que seguir para contribuir en el proyecto. Además, se deben especificar las peculiaridades del proyecto, cómo es el estilo de código o las normas de conducta que hay que seguir.

Toda guía debe estar compuesta principalmente por cuatro ficheros que se detallarán a continuación: README.md, CONTRIBUTING.md, CODE_OF_CONDUCT.md y LICENSE.md.

4.2.2.1 README.md

Es el fichero principal de toda documentación, el archivo que se ve cuando se abre el repositorio. Se trata de la primera impresión que se lleva el usuario al entrar en el proyecto. Este documento no debe entrar en detalles, es introductorio y debe ayudar al desarrollador a entender el proyecto. Para que un documento se considere bueno debe cubrir al menos los siguientes aspectos:

- ¿Qué hace el proyecto?
- ¿Cómo se instala?
- Ejemplo de uso
- Licencia
- Información del autor.

Es importante que los potenciales colaboradores sean capaces de entender en que consiste y cuáles son sus objetivos y motivaciones. Para ello hay que explicar de forma breve qué y por qué lo hace.

Además, se debe especificar de forma sencilla, los pasos a seguir para realizar la instalación, añadiendo los comandos para que el proceso sea lo más simple posible. También es recomendable presentar uno o varios ejemplos de uso con código para llamar la atención y que el usuario sea capaz de entender en funcionamiento.

Se debe añadir el tipo de licencia que utiliza el proyecto, o al menos mencionarla. Además, es aconsejable añadir información sobre el autor y la forma de contacto con él o con los responsables del proyecto.
Todas estas cuestiones deben responderse de forma resumida y deben vincularse los documentos donde se especifica de forma más precisa cada uno de los puntos.

Por último, también se puede responder a otras preguntas como quién ha trabajado en el proyecto, el estado en el que se encuentra o las funcionalidades básicas. Todo lo que ayude al usuario a comprender cómo funciona y que consiga llamar la atención para que colabore o trabaje con él.

4.2.2.2 Contributing.md

Este es un documento que detalla la forma de trabajar y colaborar en el proyecto. Este fichero debe ayudar al usuario a participar correctamente y así mantener una coherencia a lo largo de todo el proyecto.

Toda guía debe describir cómo colaborar para reportar un error, sugerir una nueva funcionalidad, añadir un test o realizar una mejora. Además de definir los pasos a seguir hay que especificar algunas características, como las etiquetas y su modo de empleo, o el estilo del código. Es recomendable que el fichero principal se llame CONTRIBUTING.md pues a la hora de crear una *issue* o *pull request* GitHub y otros gestores, automáticamente enlazan este documento para que el usuario esté al corriente de que hay que cumplir unos requisitos.

4.2.2.2.1 Issues

Una *issue*, es una incidencia o un error en el software. En proyectos *open source* los colaboradores pueden abrir *issues* para alertar errores. Cómo crear una es uno de los apartados de la documentación, pues, aunque por definición se entiende que una *issue* es para reportar errores, se utiliza como una herramienta de comunicación y hay que especificar su uso.

Más allá de fallos, pueden abrirse *issues* para discutir sobre algún tema referente al repositorio, proponer una nueva funcionalidad o cambio.
También para facilitar el trabajo de los contribuyentes, se pueden crear *templates* de los *issues*, de forma que cuando un usuario vaya a abrir una nueva, se le muestra un mensaje por defecto con la plantilla que tiene que seguir.
Estas plantillas deben llevar el nombre ISSUE_TEMPLATE.md y es recomendable almacenarlas en la raíz del proyecto o en la carpeta *.github/* para ser detectada automáticamente. Una buena plantilla debe ayudar al usuario a abrir una incidencia y a que el resto de los usuarios sean capaces de entenderla.

```
| Pregunta          | Respuesta    |
| ----------------- | ------------ |
| ¿Versión?          | vX.X.X       |
| Dispositivo  | iPhone 7... |

## ¿Qué sucede?
<!--
  Describe el comportamiento
-->

## ¿Qué debería suceder?
<!--
  Describe los pasos.
-->

## ¿Qué pasos he seguido?
<!--
  Describe los pasos que has seguido para llegar al error
-->
```

Tabla 1: Ejemplo de plantilla para issues.

4.2.2.2.2 Pull requests

El otro apartado que debe describirse es el de los *pull requests*, esto es, una propuesta de cambio ya implementada o con vistas a ello. Puede ser un único cambio, como la solución de un error, o el inicio de una nueva funcionalidad en la que van a trabajar varias personas.
Cuando se abre un *pull request* es importante apuntar a la *issue* a la que hace referencia, ya que de esta forma se puede mantener un canal de comunicación a través de esta incidencia. Para vincularlo basta con escribir en el mensaje principal una almohadilla seguida del número de la *issue*.

Todas las colaboraciones deben mantener el mismo estilo de código especificado y deben ir acompañadas de pruebas para validar el funcionamiento del software de forma automática. Para esta validación se suele hacer uso de herramientas de integración continua que ejecutan una serie de *tests* sobre el código y comprueban la validez de forma automática, facilitando el trabajo de los responsables de mantener el código.

Además, se debe añadir cualquier contenido que ayude a comprender el *pull request*. Por ejemplo, cuando se propone un cambio de interfaz es una buena práctica añadir imágenes del antes y el después.

Por último, se deben organizar las *pull request* para poder entender cual es el objetivo de cada una de ellas y agilizar la búsqueda. Para ello se pueden utilizar las etiquetas o *labels* que ofrecen todos los gestores de repositorios o también escribiendo en el título de cada una de ellas el motivo.

```
[FEATURE] Nuevo traducción al francés
[BUGFIX] Error al cambiar de idioma
[DOCS] Guia de uso de la app
```

Tabla 2: Ejemplos de *commits*.

4.2.2.3 CODEOFCONDUCT.md

El código de conducta es un documento que establece una serie de normas de comportamiento que deben seguir todos los participantes en el proyecto. El objetivo de este documento es el de mantener un ambiente de trabajo positivo. Las reglas que se establecen no deben ser discriminatorias, por ejemplo respecto a sexo, edad o religión.
En el código de conducta se debe definir el alcance, es decir, donde y a quién se le debe aplicar las consecuencias de no cumplirlo y la forma de reportar en caso de que alguien no lo siga.
Todo este contenido debe estar descrito en un fichero con el nombre CODE_OF_CONDUCT.md, que se debe almacenar en la raíz del proyecto, para que las plataformas de desarrollo colaborativo puedan detectarlo y de esta forma vincularlo para que cuando alguien intente abrir una incidencia o subir un cambio, le recuerde que debe seguir las normas establecidas por la organización.

Existen diferentes estándares para proyectos de código abierto, como el *Django Code of Conduct*, el *Citizen Code of Conduct* o el que ofrece Google, pero el más popular es el *Contributor Covenant* que siguen proyectos como Rails, Swift o Atom.
En éste se definen unos estándares de uso, el alcance, las responsabilidades y la aplicación. Entre los ejemplos de conductas adecuadas se mencionan los siguientes:

- Uso de lenguaje amable e inclusivo
- Respeto a diferentes puntos de vista y experiencias
- Aceptación de críticas constructivas
- Enfocarse en lo que es mejor para la comunidad
- Mostrar empatía a otros miembros de la comunidad

El uso de este código de conducta es libre y está disponible en diferentes idiomas y formatos para poder añadirlo a cualquier proyecto.

4.2.2.4 License.md

La licencia es el permiso de uso que da el propietario del proyecto al resto de usuarios, donde estos se comprometen a cumplirla desde el momento que hacen uso del software. En este documento se establecen tanto los derechos como las obligaciones que tiene el usuario del producto.
El uso de una licencia permite que el proyecto esté protegido ante ciertas cuestiones legales, pero ofrecen al resto de usuarios la posibilidad de uso y modificación del software en función de unas condiciones y limitaciones que el propietario debe establecer.
Existe una gran variedad de licencias destinadas a proyectos software. Todas ellas se pueden incluir en el repositorio. La decisión sobre qué licencia escoger es uno de los puntos importantes de cualquier proyecto abierto, porque una vez establecida, sólo se puede modificar con el consentimiento de todos los que hayan colaborado en el desarrollo.
Para escoger una licencia hay que entender primero el término *copyleft*. Este concepto jurídico señala que cualquier software libre debe seguir siéndolo en obras derivadas, de forma que nadie se pueda apropiar de un proyecto de software abierto.

Dentro de las licencias hay dos grupos: las permisivas que son las que mantienen el concepto *coyleft* y las restrictivas, que son las que no lo siguen. A continuación, se describen las licencias más comunes de ambos grupos.

- General Public License (GPL): El usuario de software bajo esta licencia es libre de usar, modificar y distribuir el contenido siempre y cuando lo haga bajo la misma licencia. Mantiene el concepto *copyleft*.
- Apache License: Creada por la *Apache Software Foundation*, permite el libre uso del software siempre y cuando se reconozca al propietario y no se modifique la licencia.
- MIT: Licencia que surge en el Instituto Tecnológico de Massachusetts. Permite que el usuario use, modifique, distribuya, cambie la licencia o venda el software siempre y cuando se incluya la nota de copyright y derechos.
- BSD: No es una licencia *copyleft* puesto que permite que el desarrollador reutilice el código para el desarrollo de software de código libre o propietario.

De forma alternativa a las licencias comunes, se pueden utilizar licencias *Creative Commons*. En este caso habría que incluirla en el fichero README.md. No es una práctica muy común pero es una alternativa sencilla y flexible.

4.2.3 Guía de uso

Otro apartado de la documentación es la guía de uso. Este contenido está dirigido no solo al colaborador, sino también al desarrollador o usuario que quiere consumir el producto. Es la especificación de cómo utilizar el software: el libro de instrucciones que muestra la estructura de las funcionalidades y el uso e interacción con éstas.

Una guía debe estar compuesta por los siguientes apartados:

1. Índice: El índice de la guía debe permitir al usuario navegar entre todos los documentos de la guía sin necesidad de tener que ir de uno en uno.

2. Requisitos: En este apartado se deben describir los requisitos del entorno donde será instalado el software y pueden ser tanto de hardware como de software.

3. Instalación: En este documento se debe especificar cómo instalar el software y los pasos que hay que seguir para ponerlo en marcha. Es una buena práctica añadir los comandos que tiene que introducir el usuario para facilitarle la tarea.

4. Configuración: Se trata de un documento donde se describe cómo configurar el proyecto para poder utilizarlo.

5. Primeros pasos: En este apartado se deben añadir algunos casos de uso introductorios para que el usuario pueda ver de forma rápida el funcionamiento del software.

6. Explicación de funcionalidades: En este punto es donde se deben describir todas las funcionalidades de la aplicación. Esto se debe hacer siguiendo un orden e intentando acompañar cada caso con un ejemplo de uso.

7. Ejemplos: Uno de los apartados recomendables es el proporcionar ejemplos reales de uso de la aplicación para que el usuario pueda ver una implementación completa.

4.3 Usuarios

La base de todo proyecto son los usuarios donde existe una jerarquía. A continuación, se describen los cuatro tipos de usuario: *Propietario, responsable, Colaborador* y *Consumidor*.

4.3.1 Propietario

A la hora de gestionar los proyectos uno de los aspectos a decidir es la ubicación del código generado, si está pensado como un proyecto con posibilidad de ampliación en un futuro o si tiene una fecha de caducidad como puede ser la entrega de este trabajo. Este aspecto determina el tipo de propietario del proyecto. Puede ser de propiedad individual o de una organización.

4.3.1.1 Propietario individual

Esta primera variante es la más común de todas, en donde la propiedad del proyecto es del usuario que crea el repositorio. El contenido en este repositorio puede estar disponible de forma pública o privada. En este último caso sólo tendrán acceso al contenido los usuarios a los que el propietario haya otorgado los permisos.
Para dar continuidad basta con hacer *fork* del proyecto, realizar una copia del contenido, y trabajar de forma independiente a el mismo.

4.3.1.2 Organización propietaria

Cuando un proyecto no pertenece a una persona y se le quiere dar continuidad independientemente de quién sea el propietario, la mejor alternativa es la de almacenar el proyecto en un repositorio dentro de una organización en GitHub.

En este caso es la organización la que da los permisos de administrador del proyecto a un usuario y también quien revoca ese privilegio, de esta forma si este usuario no continúa al frente del proyecto basta con cambiar el administrador.
Estas organizaciones que se crean en GitHub no sólo permiten gestionar los repositorios mediante permisos individuales, sino que también ofrecen la posibilidad de crear equipos y subequipos de trabajo para organizar, en función del grado de autorización de cada uno de ellos o del rol que tengan. Además, cada grupo interno tiene su propio canal de comunicación donde se pueden abrir discusiones. Esta funcionalidad sólo está disponible en las organizaciones.

4.3.2 Responsable

El responsable del proyecto puede ser uno o varios usuarios del proyecto. También se le conoce como *maintainer*. Este grupo de usuarios son aquellos a los que el administrador ha otorgado permisos de escritura y tienen la responsabilidad de mantener actualizado el proyecto atendiendo las incidencias o *pull requests* que se realizan.

4.3.3 Colaborador

El colaborador es todo aquel usuario que ha contribuido en el proyecto de alguna manera, ya sea reportando un error, añadiendo una mejora o participando activamente en las discusiones.
Los colaboradores no tienen permisos especiales, solo de lectura. Para poder modificar el código del proyecto deben hacerlo mediante *pull request* que deben ser aceptadas por los responsables. Con esto se evita que cualquiera ponga en juego la integridad del proyecto.

4.3.4 Consumidor

Este último grupo está formado por toda aquella persona que hace uso del software generado. Los consumidores son potenciales colaboradores del proyecto y el objetivo es que sea lo suficientemente atractivo para conseguir que aporten de alguna manera.

4.4 Código

Este aspecto del proyecto es vital para que los usuarios aporten en él. Si el código no es claro, no está bien presentado y no existe coherencia, no será lo suficiente atractivo como para conseguir colaboradores. Para que esto no pase, todo repositorio debe cumplir unos requisitos mínimos y así facilitar el trabajo de los desarrolladores.

El código tiene que ser coherente, se deben establecer inicialmente unas directrices, como el idioma o la nomenclatura de las variables y funciones. Esto se tiene que mantener a lo largo del proyecto y deben seguirlas todos los usuarios que tomen parte de forma activa en el desarrollo.

En cuanto al estilo es importante que se mantengan en todo el proyecto las mismas propiedades y que todos trabajen basándose en él. Cuántos espacios usar para tabular, los saltos de línea o el tipo de comillas son algunas de las características que hay que definir. Para mantener un estilo unificado es recomendable seguir los estándares más populares de cada lenguaje y aprovecharse de herramientas *linter* que se encargan de realizar análisis estáticos del código para notificar al programador sobre si sigue el estándar definido.

Por último, el código debe ser testeado, deben ejecutarse pruebas que verifiquen la lógica de la aplicación antes de lanzarla a producción. Es importante que cualquier aporte o modificación que se haga, sea validado ejecutando una serie de *tests* sobre cada funcionalidad. El hecho de utilizar test protege al desarrollador frente a la regresión, de esta forma si en un momento del ciclo de vida de la aplicación se realiza alguna modificación que no pasa las pruebas, detectar el error es mucho más sencillo. Para que un test sea correcto debe cumplir una serie de requisitos:

- **Atomicidad**, es decir, que solo compruebe una única funcionalidad.
- **Rápido y claro**, que el desarrollador entienda qué hace solo con leer el nombre.
- La función debe ser **reutilizable y pura**. Esto quiere decir que bajo las mismas condiciones responderá siempre igual.

La metodología a seguir es algo que los responsables del proyecto deben decidir al inicio de este. Una buena metodología es la *Test Driven Development* que responde a las siglas TDD donde lo primero se definen los requisitos, luego se desarrollan las pruebas y, finalmente, el código que debe ser verificado por estas pruebas.

5

Análisis y diseño

En este capítulo se describe la lógica del sistema EHUCalendarGenerator. La sección está dividida en tres apartados: el diseño de la arquitectura, los casos de uso implementados y el modelo de datos.

5.1 Arquitectura

Como se puede apreciar en la figura 6, la arquitectura sigue un modelo cliente-servidor, donde el cliente es en este caso la aplicación móvil.

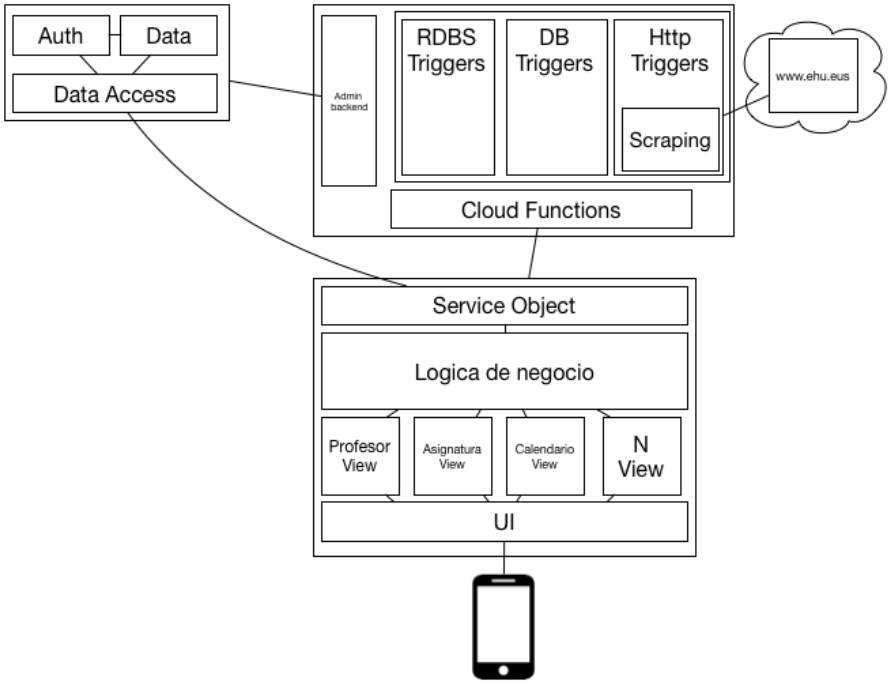


Figura 6: Arquitectura de la aplicación.

Es en el servidor donde se centraliza toda la gestión de datos y se gestionan todos los recursos. El servidor está separado en diferentes bloques. Por un lado, el acceso a los datos y usuarios, que se realiza a través de los módulos de Firebase, FirebaseAuth y Firebase Realtime Database. La interacción del cliente con estos servicios es directa gracias a la *api* que ofrece la herramienta. La conexión con los datos y usuarios es inmediata. Por otro lado, se encuentra el módulo de Firebase, Firebase Functions, desde el que se gestionan estos datos y hace de servidor, activándose las funciones en base a eventos. Para esta aplicación se utilizan los eventos de la base de datos, de autenticación y las funciones http. El bloque de funciones http es el que utiliza la librería de *scraping* para acceder a los datos de la UPV-EHU y a continuación almacenarlos en la base de datos. Este paquete es el responsable de la lógica de negocio.

El cliente, la aplicación móvil, interactúa con el servidor mediante peticiones http. Cada función pública del servidor tiene su *url* asociada a la que se realizan las peticiones. La aplicación se encarga de *renderizar*, mostrar en pantalla, las vistas y de la lógica asociada a estas, ya que toda la lógica está implementada en el *backend*.

5.2 Casos de uso

Este apartado está dividido en tres bloques donde se describen los casos de uso más importantes para cada actor de la aplicación. En el Anexo C se pueden ver las interfaces que van asociadas a cada caso de uso.

5.2.1 Usuario anónimo

En este primer bloque se describen los casos de uso relacionados con el usuario que aún no está registrado en la aplicación.

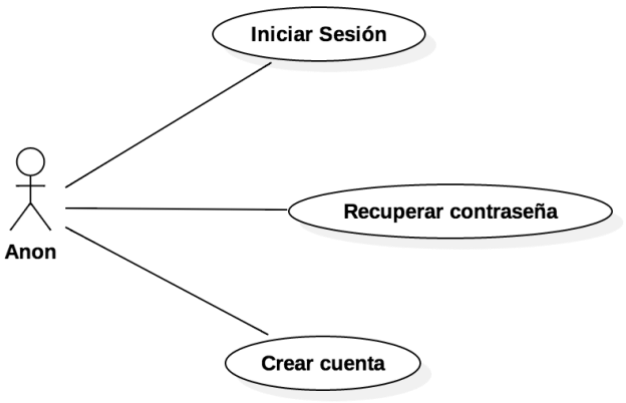


Figura 7: Casos de uso del usuario anónimo.

El primer caso de uso es el de **iniciar sesión**. Se trata de la primera vista del usuario no registrado, que debe rellenar un formulario con sus credenciales para acceder a la aplicación. La validación de los datos se realiza a través del servicio FirebaseAuth, y es este quien en caso afirmativo responde con un *token* que se guarda en el almacenamiento local para evitar repetir este proceso cada vez que se inicia la aplicación.

El segundo caso de uso, también para usuarios no registrados, es el **de crear una cuenta nueva**. La interacción se realiza de la misma forma, el cliente se conecta a través de la librería FirebaseAuth. Con los datos que envía el cliente y, si son correctos, se crea un nuevo usuario, activando un *trigger* en el servidor que vincula este nuevo usuario con un perfil en la base de datos. El servidor también envía un correo a la cuenta del usuario para que la confirme. Al finalizar todo el proceso, el sistema responde con el *token* de usuario para que el cliente inicie sesión.

El tercer caso de uso de un usuario anónimo es el de **recuperar su contraseña.** Para ello el usuario debe navegar hasta la ventana de recuperación, donde después de añadir el correo electrónico, el sistema a través de los servicios de Firebase, se encarga de enviar un correo de recuperación con las instrucciones necesarias.

5.2.2 Usuarios registrados

Los casos de uso relacionados con los usuarios ya registrados más relevantes son las búsquedas, las vistas de fichas, la suscripción a asignaturas o la vista de calendarios.

Figura 8: Casos de uso del usuario registrado.

Las **búsquedas**, tanto para asignaturas, profesores o grados, se realizan a través de las funciones http. El usuario introduce las palabras a buscar y la aplicación envía una petición con los datos a la *url* asociada a esta funcionalidad. El servidor, con los datos proporcionados, conecta con la base de datos de Firebase usando la librería Firebase Realtime Database (RDB) y filtra entre todos los que encajen con la petición recibida. Estos datos son la respuesta que recibe la aplicación y los que muestra en pantalla. La información es almacenada por la aplicación en un objeto de estado, de forma que, si el usuario quiere volver a buscar el mismo contenido, no sea necesario realizar una nueva petición.

Cuando se muestran los resultados, el usuario puede **acceder a la ficha** de cualquiera de estos. Para ello, el cliente utilizando Firebase, accede directamente a la base de datos a través de la api de Firebase ya que no se requiere de un tratamiento especial de los datos.

El siguiente caso, el de **suscripción**, se realiza dentro de la ficha de una asignatura o profesor. El usuario selecciona la opción de *seguir*, o *dejar de seguir*, y se modifica la base de datos añadiendo o suprimiendo el elemento a seguir dentro de la lista de subscripciones del perfil de usuario. Si el estudiante intenta seguir una asignatura, se le pide que escoja un grupo entre todos los que hay, de esta forma se le asocia el horario de ese grupo y no el de todo el conjunto.

Los usuarios también tienen la posibilidad de acceder a sus calendarios con las asignaturas a las que está suscrito. Para que esto sea posible la aplicación descarga todos los horarios de las asignaturas a las que está suscrito el alumno y los calendarios de la universidad y grados a los que pertenece. De esta forma, se suprimen los días festivos y periodos no lectivos.

Además, el estudiante puede configurar su perfil de forma que puede cambiar de idioma, editar sus detalles personales o cambiar su contraseña. El primer caso, el de modificar el idioma, no requiere comunicación con el exterior, puesto que se trata únicamente de una funcionalidad de la aplicación. En los otros dos casos, para modificar datos del perfil, cuando el estudiante actualiza sus datos en la aplicación, la aplicación, a través de la *api* de Firebase, modifica la base de datos.

5.2.3 Usuario administrador

El tercer bloque de casos de uso es el relacionado con la figura del administrador, quien tiene acceso a todas las funcionalidades de la aplicación y además añade las que se describen a continuación.

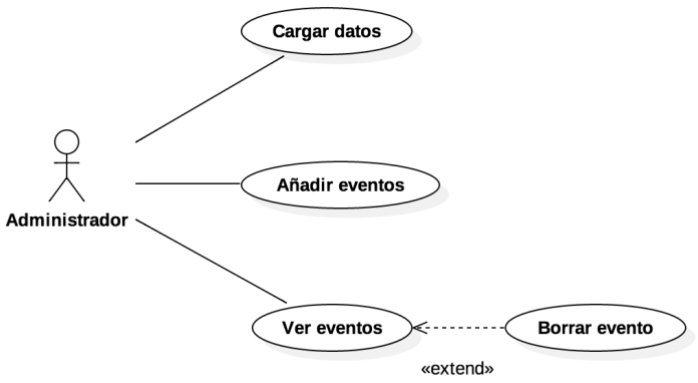


Figura 9: Casos de uso del administrador.

Para poder generar los horarios de los estudiantes, se debe primero crear un calendario de la universidad y de los grados. De esta forma se especifican los días no lectivos y las semanas, junto a su identificador, puesto que la UPV-EHU muestra los horarios en rangos semanales.

El administrador, a través de la interfaz de la aplicación, tiene la posibilidad de añadir tanto días festivos para el calendario general como para cualquiera de los grados disponibles. Además, a los grados se le deben añadir el día de inicio de cada semana. Cuando el administrador introduce estos datos se modifica automáticamente la base de datos con las nuevas fechas.

Si se equivoca, o por algún motivo se desea eliminar una fecha, existe el caso de uso 'borrar evento' que permite seleccionar evento y suprimir de la lista total esta fecha.

La última funcionalidad del administrador es la de cargar los datos, ya que tiene la posibilidad de, seleccionando un grado, asignatura o profesor, activar la actualización los datos de forma manual.

5.3 Modelo de datos

En este apartado se describe el modelo de datos utilizado en el proyecto. La información está almacenada en una base de datos de un único documento en formato JSON. Toda la información del sistema se encuentra en el mismo fichero que se divide en nodos para organizarla. En el Anexo E se pueden consultar los datos que contiene la base de datos.

5.3.1 Nodo EHU

Por un lado, se encuentra el nodo **ehu**: en él se guarda toda la información sobre la universidad. Está compuesto por cinco subnodos: **grades, teachers, subjects, calendars** y **searchs.**

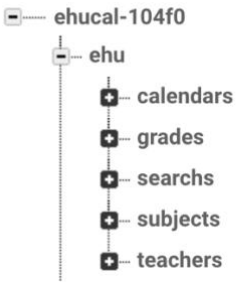


Tabla 3: Estructura del nodo ehu

Los tres primeros tienen la información completa obtenida de la web de la UPV-EHU. El acceso a los grados se hace a través del código de la facultad, el campus y el identificador del grado, manteniéndose así el mismo modelo que utiliza la UPV-EHU.

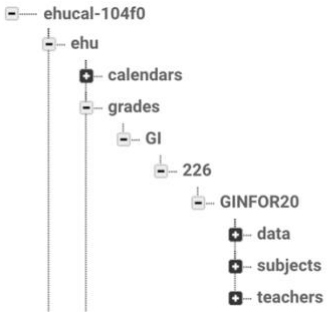


Tabla 4: Estructura de un nodo de grado.

Cada grado está compuesto por un objeto con la información resumida y dos objetos más que almacenan todos los profesores y asignaturas dentro de este grado. Estos dos objetos funcionan como un diccionario donde la clave es el código del profesor o asignatura y el valor el nombre.

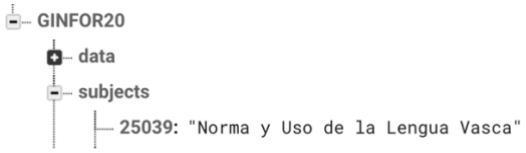


Tabla 5: Lista de asignaturas dentro de un grado.

Para las asignaturas solo se requiere el código de estas y el del grado al que pertenecen. Así se evita la necesidad de conocer el campus y facultad en las que se encuentran. Todas las asignaturas están dentro del mismo nodo agrupadas y se identifican por la clave con formato *CODIGOASIGNATURA_CODIGOGRADO*.

Tabla 6: Estructura del nodo asignatura.

Una asignatura está dividida en tres objetos: ***detail***, que almacena la información ofrecida por la guía docente, ***summary,*** con los datos básicos de la asignatura y ***schedule*** con el horario.

Los profesores, de la misma forma que las asignaturas, están almacenados todos por su identificador y el del grado al que pertenecen. Como existe la posibilidad de que una misma persona ejerza en diferentes centros, la información de cada subnodo solo muestra la referente al grado por el que se le busca.

Tabla 7: Estructura del nodo profesores.

En el módulo profesor se guarda directamente toda la información en él, donde también se encuentra otro objeto bajo el atributo ***schedule*** que contiene el horario de todas las tutorías que ofrece.

El cuarto, el nodo ***searchs***, está formado por otros tres subnodos con profesores, grados y asignaturas, donde cada uno está compuesto por una lista clave-valor. La clave sigue el formato *CODIGOELEMENTO_CODIGOGRADO* y el valor es el nombre del elemento. La función del nodo *search* es agilizar las búsquedas de forma que no sea necesario obtener todos los datos y se acceda de forma más rápida.

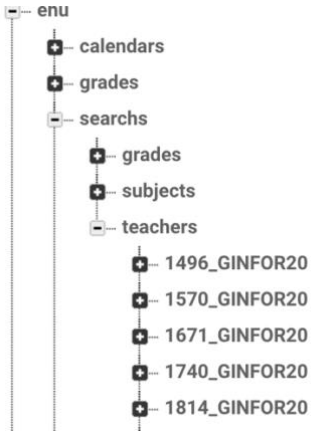


Tabla 8: Estructura del nodo searchs.

El último bloque dentro del nodo *ehu* es el de los calendarios. En él se almacena tanto el calendario general de la universidad como los específicos de cada grado. Dentro de cada calendario se identifican las fechas no lectivas y las que indican el inicio de una semana.

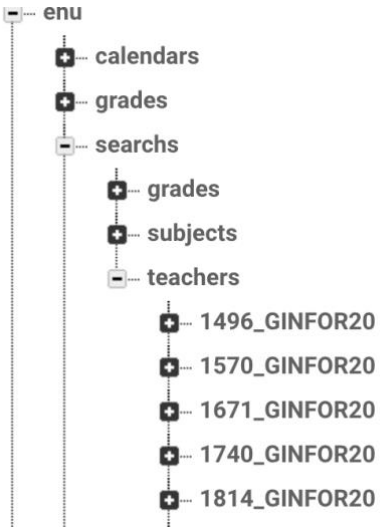


Tabla 9: Vista detallada de un nodo de búsqueda.

5.3.2 Nodo users

Otro conjunto de datos es el de usuarios, aquí se almacena toda la información de los perfiles. Cuando se registra un usuario, se añade un elemento a este nodo con la información básica del usuario. La contraseña, email y nombre no se almacenan aquí, quedan registrados en el módulo FirebaseAuth por motivos de seguridad. Cada perfil de usuario tiene por un lado los datos básicos en el objeto *data,* como son su nombre, apellido o avatar. Además, dentro del mismo perfil, pero en objetos diferentes, se almacenan los elementos a los que está suscrito.

Un elemento importante del perfil de usuario es el rol que se asigna inicialmente como "regular" y que determina el grado de acceso a las funcionalidades de la aplicación.

Tabla 10: Estuctura del nodo users

6

Implementación

En el apartado de implementación se explica como ha sido desarrollada el sistema EHUCalendarGenerator al completo. En este capítulo se explica cuales son las tecnologías utilizadas en todo el proyecto. El resto de los puntos del capítulo se enfocan en los aspectos comunes, como el uso de Git, y en cada una de las tres implementaciones realizadas: la librería EHUScraping, el servidor EHUServer y el cliente EHUApp.

6.1 Tecnologías

El lenguaje principal de la aplicación es JavaScript. Se han implementado todos los paquetes utilizando las últimas versiones a partir de 2015 (ES6, ES7 y ES8). A pesar de utilizar un lenguaje común, todos los módulos utilizan tecnologías diferentes que se detallan a continuación.

Para la librería de *scraping* se utiliza el entorno de ejecución multiplataforma NodeJS junto al gestor de paquetes NPM. El servidor está implementado usando los servicios de la plataforma Firebase. El cliente se ha desarrollando usando el *framework* React Native junto a la librería de Javascript: React. Por último, todas las implementaciones utilizan herramientas comunes como el servicio de integración continua Travis CI o el empaquetador de módulos Webpack.

6.1.1 Tecnologías de la librería

Para el desarrollo de la librería se requiere una tecnología que se pueda implementar tanto en el servidor como en el cliente. Para ello y como el resto de los paquetes están desarrollados usando implementaciones en JavaScript, la elección es NodeJS.

6.1.1.1 NodeJS

NodeJS es un entorno de ejecución multiplataforma y de código abierto que está basado en JavaScript y orientado al desarrollo en el servidor, aunque no limitado a ello. Esta tecnología surgió en el año 2009 a manos de Ryan Dahl, basada en el motor de JavaScript V8 de Google. Permite una programación dirigida por eventos, ayudando a generar servicios más rápidos y escalables.

Los aspectos técnicos más relevantes de esta tecnología son los siguientes:

- **Concurrencia**: Trabaja sobre un único hilo mediante peticiones de E/S no bloqueantes. Gracias a esto soporta cientos de conexiones simultáneas y permite crear aplicaciones concurrentes.
- **Motor V8**: Motor de JavaScript desarrollado por Google para su navegador en el año 2008 y de carácter *open source*. Éste compila el código JavaScript a lenguaje nativo en vez de interpretarlo en tiempo real.
- **Bucle de eventos**: Es una cola de funciones que realiza todas las operaciones de E/S de forma asíncrona. Con esto se evita que se bloqueen las operaciones.
- **Módulos**: NodeJS ofrece una serie de módulos por defecto que no requieren de instalación y además también ofrece la posibilidad de usar otros desarrollados por terceros.

6.1.1.2 NPM

NPM es un gestor de paquetes para JavaScript que viene preinstalado con NodeJS. Su objetivo es la instalación y gestión de módulos desarrolladas por terceros. Funciona como un repositorio de librerías donde un programador sube su trabajo para que el resto de los desarrolladores puedan utilizarlo en sus proyectos.

Los paquetes se pueden instalar de forma local como global y estos quedan almacenados en directorios con el nombre *node_modules*.

Cuando se trata de un proyecto software se instala de forma local en él, quedando reflejadas las dependencias en el fichero *package.json*

Este fichero determina qué librerías hay y de qué tipo, pues pueden ser tanto para uso en producción de forma que se instalan en todos los casos o para uso en fase de desarrollo. Para generar un fichero *package.json* se hace mediante el siguiente comando.

```
$ npm init
```

Programa 1: Comando para iniciar paquete npm.

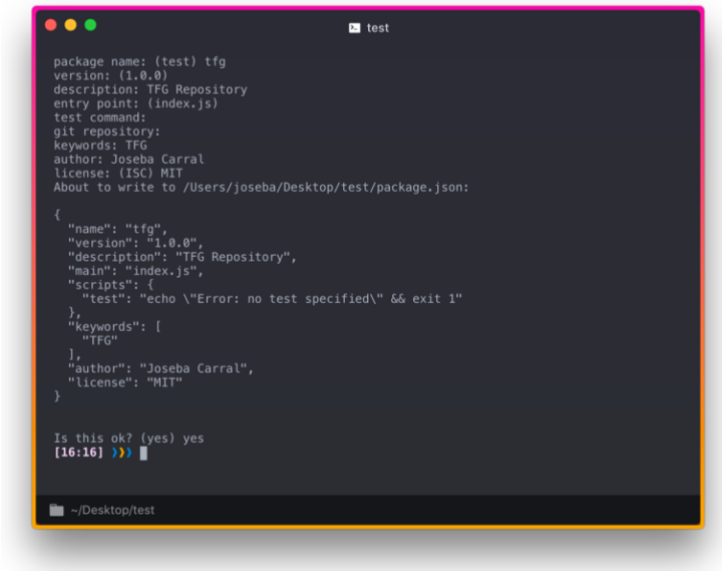


Figura 10: Consola con el resumen del comando npm init

A través de la misma consola, la herramienta realiza una serie de preguntas para generar el archivo inicial.

La instalación de paquetes, también mediante la línea de comandos, se puede hacer de varias formas. La primera es de forma global y se instala en la raíz de la máquina. Esta operación suele requerir permisos de administrador. La otra alternativa es la local, la más habitual. Cuando se instala un paquete de forma local se debe especificar que registre en el *package.json* utilizando el flag *–save* o *–save-dev* si se quiere hacer solo para desarrollo. Si no se guardase aunque la librería se instalara, no quedaría registrada y se perdería en caso de que otro desarrollador fuera a descargarse el proyecto o cuando llegara la hora de subirlo a producción.

Los comandos más comunes son los siguientes:

```
//Iniciar un proyecto
$ npm init
//Instalar todos los paquetes que hay registrados en el package.json
$ npm install
//Actualizar los paquetes
$ npm update
```

```
// Instalar paquetes de forma global
$ npm install -g [NOMBRE_PAQUETE]°
//Instalar y guardar registro del paquete en proyecto local
$ npm install --save [NOMBRE_PAQUETE]
//Instalar y guardar el paquete en modo desarrollo
$ npm install --save-dev [NOMBRE_PAQUETE]
```

Programa 2: Comandos comunes npm.

De la misma forma que se pueden descargar, también se pueden subir librerías y ponerlas disponibles para el resto de la comunidad de desarrolladores.

El primer requisito es tener una cuenta en la web de NPM. El registro es gratuito, aunque cuenta con una opción de pago para poder subir los proyectos de forma privada pues por defecto son públicos.

El segundo requisito es que el proyecto tenga un fichero *package.json*. Estos ficheros están compuestos por una serie de atributos, la mayoría de ellos opcionales. Entre todos ellos destacan los siguientes:

- **Name**: Atributo obligatorio. Es el nombre que se muestra en la web. Debe ser único, en minúsculas y sin espacios. Este nombre es el que se utiliza para descargar la librería.
- **Version**: Atributo obligatorio. Determina la versión en la que se encuentra el proyecto. Debe ser una secuencia ordenada para que sea interpretable. Se recomienda utilizar un estándar como *semver*.
- **Main**: Punto de entrada del programa.
- **Description**: Descripción del módulo. Se mostrará en la web.
- **License**: Licencia del software. Por defecto se ofrece la licencia ISC.
- **Repository**: Enlace al repositorio donde se encuentra el proyecto.
- **Author**: Datos sobre el autor.
- **Files**: Lista de archivos que se deben subir al repositorio. Por defecto se suben todos salvo los que se omiten a través de los *.gitignore* o *.npmignore*.
- **Scripts**: Un objeto con los comandos que se ejecutan.
- **Dependencies**: Objeto con las dependencias instaladas junto a la versión de estas.
- **devDependencies**: Dependencias instaladas para la fase de desarrollo.
- **Engines**: Objeto para especificar las versiones de NodeJS sobre las que funciona el software.

Para publicar la librería se hace a través de la línea de comandos y utilizando la cuenta de usuario de npmjs.

```
//Iniciar sesión en NPMJS. Pedirá credenciales
$ npm login
//Subir el proyecto
$ npm publish
```

Programa 3: Comandos para publicar paquete en npm.

Si no hay problemas, el contenido estará disponible instantáneamente para poder descargar o visitar en la web.

6.1.2 Tecnologías del servidor

En este apartado se describen las tecnologías utilizadas para el desarrollo de la lógica de negocio del servidor. Estas tienen que cumplir los siguientes requisitos:

- Gestión de usuarios
- Almacenamiento de datos NoSql
- Escalable
- Almacenamiento seguro
- Conexiones seguras

6.1.2.1 Firebase

Firebase es una plataforma de Google que ofrece los servicios necesarios para el desarrollo del proyecto cumpliendo los requisitos especificados.

Aúna un conjunto de servicios disponibles para diferentes plataformas móviles gracias a sus API integradas en SDKs individuales para web, Android e iOS. Se divide en tres bloques de servicios: desarrollo, crecimiento y monetización.

Más allá del abanico de herramientas del que está compuesto, ofrece ciertas ventajas frente a opciones más tradicionales. La primera ventaja es la oferta de planes, desde uno gratuito suficiente para la fase de desarrollo hasta planes personalizados en función del consumo tanto de ancho de banda como de almacenamiento. Esto hace que sea fácilmente escalable. También es importante su fácil configuración y que no requiere de infraestructuras adicionales, ahorrando así una tarea tediosa para el desarrollador. A pesar de ser ya bastante completa, el crecimiento es continuo pues Google está constantemente añadiendo funcionalidades y mejorando la plataforma. No solo la empresa trabaja en ello, además al tener *SDKs* públicas existe una gran comunidad de desarrolladores trabajando en generar librerías adicionales.

6.1.2.1.1 Servicios utilizados

Para este proyecto no se requiere el uso de todos los servicios de Firebase. A continuación, se especifican los utilizados: Firebase Auth, Firebase RealTime Database y Firebase Functions.

Firebase Auth: Este servicio es el responsable de la base de usuarios. Ofrece una serie de funciones para crear y gestionar los usuarios de la aplicación de forma simple y segura.

No requiere de configuración más allá de determinar cuales son los proveedores permitidos, porque además de ofertar la posibilidad de añadir cuentas con el correo y contraseña, ofrece la posibilidad de utilizar proveedores como Google, Facebook, GitHub, Twitter o usando el número de telefono.

Firebase Realtime Database: Para el almacenamiento de datos se utiliza este servicio y aunque actualmente se ofrece otra alternativa, FireStore, esta aún se encuentra en fase de desarrollo y no es lo suficientemente estable.

Firebase Realtime Database es una base de datos no relacional que destaca por ofrecer información en tiempo real. El contenido está almacenado en un único documento JSON que se estructura con nodos.

Permite trabajar sin conexión en las aplicaciones porque el SDK hace que los datos persistan en el disco. El acceso se puede hacer de forma directa desde la aplicación móvil o web sin requerir de un servidor. Esto se puede hacer gracias a las reglas de seguridad.

Lo que denominan Security Rules, es un lenguaje de reglas flexible que permite estructurar y limitar el acceso, tanto en modo escritura como lectura, a los diferentes nodos de la base de datos en función de ciertos parámetros establecidos por el desarrollador. Los permisos de estas reglas funcionan en cascada y por lo tanto si se le concede a un nodo todos los hijos tendrán al menos el mismo nivel, nunca menos.

```
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "auth.uid == $uid",
        ".write": "auth.uid == $uid",
      }
    }
  }
}
```

Programa 4: Reglas de seguridad en firebase.

Firebase Functions: Este servicio, disponible desde 2017, se encuentra en Mayo del 2018 en fase de desarrollo, pero es lo suficientemente estable como para hacer uso de él. Esta funcionalidad permite crear funciones JavaScript en el entorno NodeJS que se invocan mediante eventos. Las funciones permiten dividir el backend en microservicios, siendo cada una de ellas encargada de una tarea diferente.

Hay varios posibles disparadores, eventos que provocan la ejecución de la función.

- HTTP Triggers: Funciona en base a peticiones http a urls asociadas a las funciones.
- Realtime Database Triggers: Se activan cuando se realiza cualquier operación CRUD (Crear, leer, actualizar o borrar) sobre la base de datos.
- Authentication Triggers: Provocan la ejecución de las funciones al crear o borrar una cuenta de usuario.
- Analytics Triggers: Se activa cuando un usuario genera un evento de conversión.
- Crashlytics Triggers: Se ejecutan junto al servicio Crashlytics.
- Cloud Storage Triggers: Cuando se sube, descarga o actualiza algún elementos del cloud storage se activan estas funciones.
- Cloud Pub/Sub Triggers: Se activa junto a cualquier evento del servicio Pub/Sub.
- Firestore Trigger: Se ejecuta cuando se realiza alguna operación sobre el servicio Firestore.

6.1.2.1.1 Servicios no utilizados

A pesar de no utilizar la totalidad de los servicios ofrecidos, hay que destacar otros, no incluidos, que merecen ser presentados de forma sucinta.

- Firebase Storage: Se trata de un servicio de almacenamiento de ficheros de forma ordenada siguiendo una estructura de nodos sobre la que se le puede aplicar una serie de reglas de seguridad de la misma forma que sobre las bases de datos en tiempo real.
- Hosting: Servicio que permite alojar contenido estático de forma sencilla y segura.

- **Firebase Google Analytics**: Ofrece informes con analíticas de la aplicación desarrollada. Solo disponible para implementaciones móviles.
- **Firebase Cloud Messaging**: Servicio para enviar notificaciones a los usuarios de todas las plataformas sin coste alguno.
- **Predicitions**: Aplica *machine learning* a las analíticas de la aplicación para gestionar los usuarios en base al uso que hagan de la aplicación.
- **Firebase Firestore**: Servicio de almacenamiento de datos alternativo que extiende las funcionalidades de las bases de datos en tiempo real. Se encuentra aún en fase Beta. Permite organizar el contenido en colecciones y documentos. Además, también mejora las *queries* respecto al actual.
- **Remote config**: Servicio para modificar el comportamiento y apariencia de las aplicaciones móviles de forma remota.
- **Invites**: Servicio para generar invitaciones e incorporar usuarios gracias a ellas.
- **Admob**: Permite añadir publicidad y generar ingresos mediante la aplicación.
- **Adwords**: Servicio para publicitar el proyecto en la plataforma Admob.
- **App Indexing**: Ayuda a vincular la aplicación con el servicio de autocompletado de Google.

6.1.3 Tecnologías del cliente.

Este apartado detalla las tecnologías escogidas y requisitos que debe cumplir el lado del cliente. Para este proyecto se utiliza React Native como framework de desarrollo de aplicaciones móviles junto a la librería Redux. Además para los elementos visuales se utiliza la librería React Native Elements.

Los requisitos que se deben cumplir son los siguientes:

- Multiplataforma.
- Interfaz intuitiva.
- Fácil de integrar con Firebase y NPM.
- Permitir varios idiomas.
- Fácil de modificar.

6.1.3.1 React Native

React Native es una tecnología creada por Facebook que permite crear aplicaciones móviles nativas para Android e iOS pero desarrollando de forma híbrida. Hasta la fecha las aplicaciones móviles se producían de forma separada, una versión independiente para cada plataforma. Como alternativa surgieron las aplicaciones híbridas. Se desarrollaba para web y se incrustaba en una *web view*. Esta segunda opción permitía agilizar el desarrollo, pero no ofrece las mismas posibilidades que las aplicaciones nativas ya que el acceso al hardware y a las APIs nativas está limitado. Como solución a esta limitación se presenta React Native que propone un desarrollo web, pero con una experiencia nativa.
Se requiere tener instalado NodeJS en la máquina sobre la que se trabaja y además un emulador o un dispositivo físico para ver el progreso.

Utiliza JavaScript y la librería React junto a los componentes nativos de cada plataforma. De esta manera se trabaja de la misma forma que si se hiciera una web, con el mismo conjunto de tecnologías. El resultado se compila automáticamente a código nativo, un solo desarrollo genera dos aplicaciones.

6.1.3.2 React

Figura 11: Logo React

React es también una librería desarrollada por Facebook para aplicaciones web. Está orientada al *frontend* aunque se puede implementar en lado del servidor, de esta forma permite crear aplicaciones isomorfas. Su uso no se limita a la web, aunque fue concebida para ello. También se utiliza para generar aplicaciones móviles desarrolladas de forma híbrida.

El desarrollo se basa en componentes, pequeños elementos visuales diferenciados por tener comportamientos diferentes. El uso de estos componentes permite reutilizar código e incluso importar otros componentes desarrollados por terceros.

6.1.3.3 Redux

Figura 12: Logo de Redux.

Redux es una librería independiente para utilizar la arquitectura Flux.

Esta arquitectura, desarrollada por Facebook para el manejo de datos, es la sustituta del patrón MVC. Con *flux,* el flujo de datos es unidireccional, parten de la vista, llegan a un store mediante acciones y es el store quien actualiza las vistas de nuevo. Una arquitectura flux está compuesta por tres elementos:

- **Store**: Es el elemento que mantiene el estado de la aplicación, donde se almacenan los datos. Puede haber varios *stores*.
- **Actions**: Es un objeto JavaScript que indica una intención. Está compuesto por un atributo que indica el tipo de acción que se va a realizar y, si se necesitase, también de los datos que se modificarán en el estado.
- **Dispatcher**: Es el intermediario entre una acción y el *store*. Es el que propaga la acción hacia el store.

Redux sigue este patrón modificando algunos aspectos de él. Para empezar, solo hay un *store*, es decir, el estado de la aplicación se almacena en un único objeto JavaScript. Este estado no se puede modificar, es solo de lectura y, si se quiere actualizar algún atributo del árbol, se debe hacer mediante acciones.

Para trabajar con el estado se añade un nuevo elemento al modelo, el ***reducer***. El *reducer* es una función pura que recibe dos parámetros: el estado actual de la aplicación y la acción. Con estos dos parámetros genera una copia del estado que modifica en función de la acción

recibida y la retorna para que la reciba el *store*. Así se mantiene el principio de *inmutabilidad* del estado: no se modifica, se crea uno nuevo. Esto facilita a la hora de hacer un seguimiento de errores más exhaustivo. Una aplicación puede tener varios *reducers*, cada uno de ellos responsable de una parte del estado, manteniendo una aplicación más modular.

6.1.3.4 React Native Elements

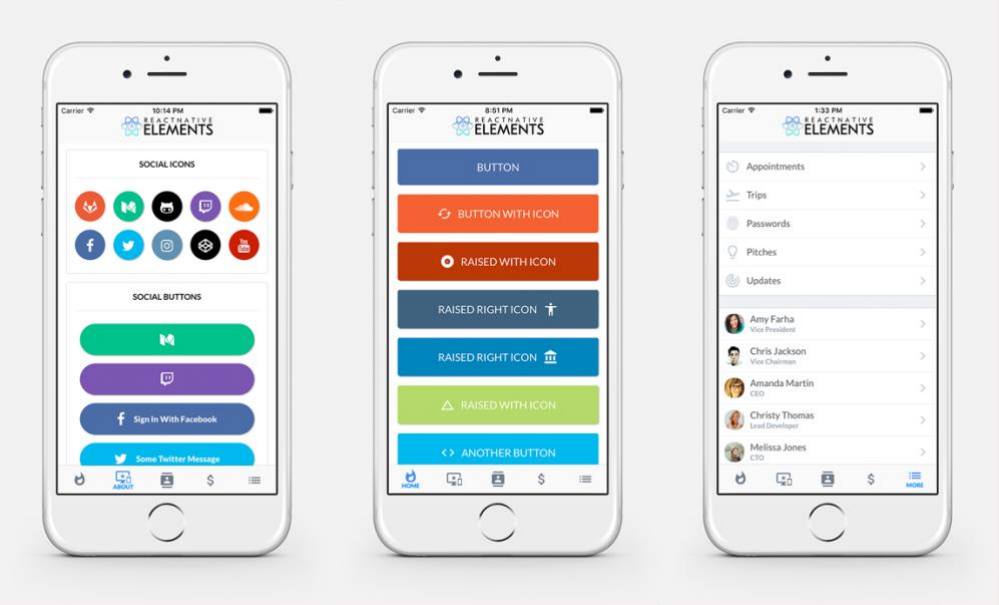


Figura 13: Componentes de React Native Elements.

React Native Elements es una librería open source que ofrece una gran variedad de componentes visuales para React Native. Estos componentes se adaptan perfectamente a las dos plataformas móviles. Además de los ofrecidos ya por defecto por React Native, con esta librería se puede extender el comportamiento de muchos ellos y añadir algunos que no están implementados por Facebook. No requiere de una configuración más allá de la instalación a través de NPM y por lo tanto hace de ella una herramienta muy sencilla y cómoda de utilizar.

6.1.4 Tecnologías comunes

Además de las tecnologías orientadas a los diferentes bloques del proyecto, hay otras que se pueden implementar en todos los proyectos porque no están limitadas a un tipo de producto.

6.1.4.1 Travis CI

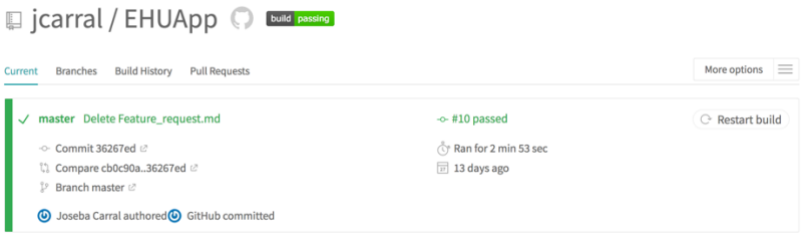


Figura 14: Vista de la interfaz de TravisCI.

Travis es una herramienta de integración continua con un plan gratuito que permite automatizar una gran cantidad de tareas únicamente con un archivo .travis.yml.

Esta herramienta se integra perfectamente con GitHub y acepta múltiples entornos de trabajo. Esto permite probar el proyecto en diferentes configuraciones sin necesidad de instalarlas en una máquina local.

Entre las tareas que se pueden realizar, está la de ejecutar pruebas y validar si el software las pasa, subir el contenido automáticamente a producción o simplemente llevar un control de calidad del proyecto.

6.1.4.2 Webpack

Webpack es lo que se conoce como un *bundler* o empaquetador de módulos. Esta herramienta está orientada al desarrollo de aplicaciones web pero se puede utilizar en cualquier proyecto, ya que tiene más funcionalidades. Está pensada para gestionar las dependencias, ejecutar tareas en fase de desarrollo o crear un servidor.

Se considera una herramienta de compilación, pues a partir de los ficheros del proyecto genera unos nuevos más eficientes y funcionales para trabajar en producción.

6.2 Uso de GIT

Git se ha utilizado como herramienta de gestión de versiones en todas las implementaciones de este TFG. Para organizar el flujo de trabajo se ha seguido una versión simplificada de la mencionada en el apartado 4.1.2 de esta memoria.
A diferencia de la versión recomendada por Atlassian, se omite el uso de la rama *releases* para los módulos EHUServer y EHUApp. Si se mantiene esta rama en el paquete EHUScraping porque al utilizar TravisCI se automatiza el despliegue a NPM desde esta rama.

A la hora de añadir una funcionalidad nueva, se utilizan ramas que empiezan por *feature/*, seguidas del nombre de la nueva característica que se implementa. Todas estas ramas parten del *branch develop* y cuando se terminan de implementar se fusionan de nuevo con este, nunca directamente con la versión *master*.
Los errores o bugs se solucionan en ramas que empiezan con el prefijo *hotfix/* seguida de la versión que se está corrigiendo. Por ejemplo, si la versión 1.0.0 de la librería tiene algún fallo, se crea una rama con el nombre *hotfix/v1.0.0* que soluciona este error.

En la siguiente figura se puede apreciar de forma gráfica un fragmento de la gestión de ramas de Git.

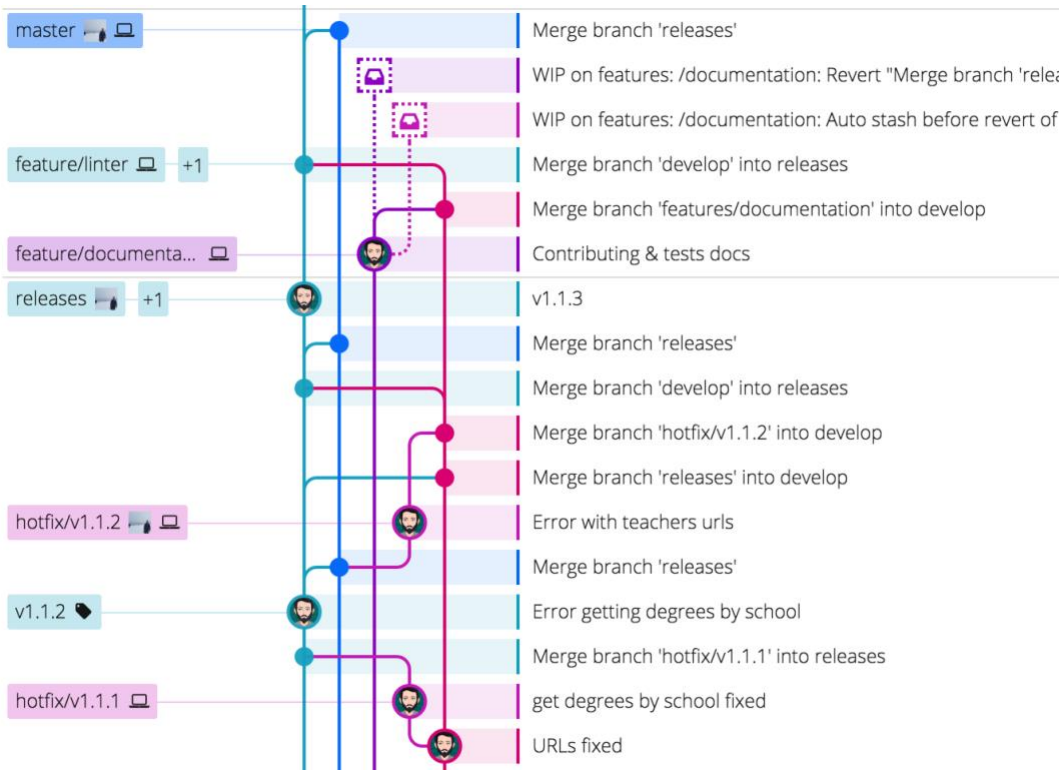


Figura 15: Vista de las ramas utilizando la aplicación GitKraken.

6.3 Implementación general

A pesar de ser tres paquetes diferentes, implementados con distintas tecnologías en los tres casos se han utilizado tecnologías comunes con configuraciones similares. Estas herramientas están orientadas al desarrollo y no al producto final. Las dos primeras, editorconfig y linter, son las que se responsabilizan de mantener el mismo estilo de código. A continuación se encuentra la configuración del servicio de integración continua TravisCI. Por último, el archivo *.gitignore* con los ficheros que se deben omitir a la hora de subir el contenido a los repositorios.

6.3.1 Editorconfig

Con el fin de mantener el mismo estilo de código, todos los proyectos comparten el fichero *.editorconfig*. En él se establecen las reglas básicas, como que todos los archivos tienen una tabulación de dos espacios y termina con un salto de línea para evitar posibles errores con ciertos editores.

```
root=true

[*]
indent_style = space
tab_width = 2
end_of_line = lf
charset = utf-8
insert_final_newline = true
```

Programa 5: Archivo .editorconfig.

6.3.2 Linter

Para seguir un estilo de código común se utiliza un fichero .eslintr.json donde se especifica el estándar a seguir. Para este proyecto se sigue el propuesto por AirBnb, que tiene gran aceptación dentro de la comunidad de desarrolladores de JavaScript. A pesar de ello, se sobrescriben algunas reglas y con el fin de evitar inconvenientes con alguna implementación.

```
{
  "env": {
    "browser": true,
    "node": true
  },
  "extends": "airbnb",
  "rules": {
    "react/jsx-filename-extension": 0,
    "func-names": 0
  }
}
```

Programa 6: Fichero .eslintr.json utilizado

Las reglas que se desactivan son diferentes en todos los proyectos y se hace para evitar conflictos con las tecnologías utilizadas.

6.3.3 .travis.yml

A pesar de que este archivo es diferente en todos los casos, sus diferentes versiones, tienen determinadas propiedades en común. Para empezar, se utiliza el mismo lenguaje node_js que no es más que la versión JavaScript para el servidor, usando la última versión estable.

Todos deben ejecutar el comando *npm run lint* que comprueba que el proyecto sigue el estándar antes especificado.

```
sudo: false
language: node_js
node_js:
- stable

script:
- npm run lint
```

Programa 7: Ejemplo de fichero .travis.yml

Como resultado del uso de TravisCI en el Anexo G se puede consultar los logs resultantes.

6.3.4 .gitignore

Este fichero tiene la responsabilidad de evitar subir archivos confidenciales a GitHub. Estos son algunos de los ficheros que se debe evitar subir. No son los únicos, ya que cada tecnología añade a esta lista otros archivos que se recomienda no estén disponibles para todos.

```
node_modules
package-lock.json
yarn.lock
yarn-error.log
.vscode
.DS_Store
...
```

Programa 8: Fichero .gitignore con los principales archivos a evitar.

6.4 Implementación de la librería.

La librería EHUScraping es la encargada de obtener los datos de la web de la universidad. Está implementada utilizando JavaScript y está disponible para utilizar a través de NPM. En el AnexoD.1 se puede consultar parte del código utilizado.

La estructura del proyecto es la siguiente:

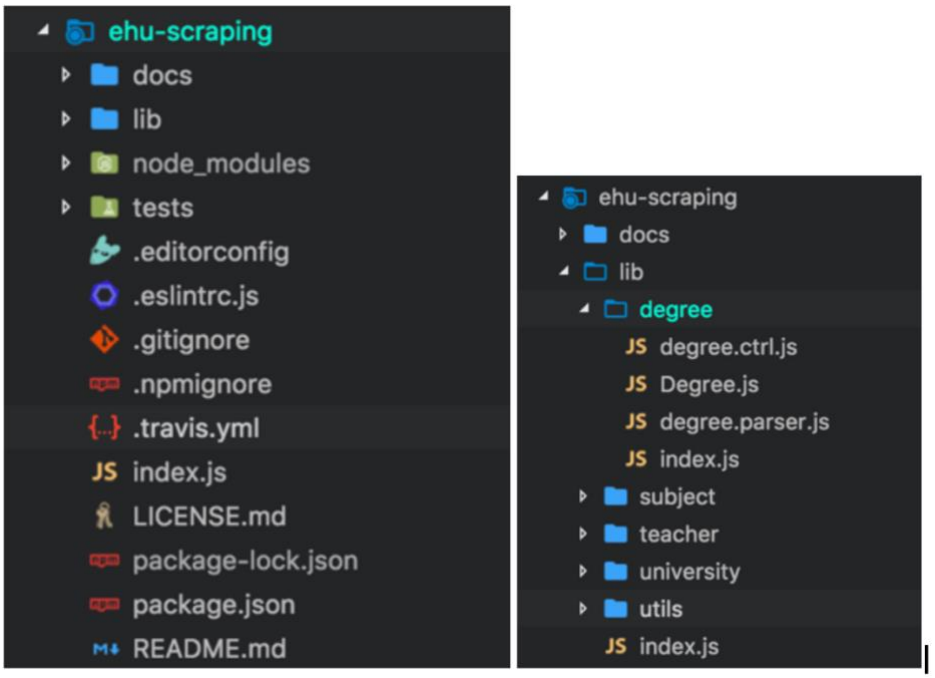


Tabla 11: Estructura de la libreria.

El punto de partida de la aplicación es el *index.js,* que únicamente carga las clases del paquete.

La librería está formada por cuatro módulos: Universidad, Grados, Profesores y Asignaturas. Cada uno de ellos realiza una serie de funciones públicas que se pueden utilizar a través de las clases.

A su vez, cada carpeta está formada por cuatro ficheros. El principal es el *index.js* que es el punto de entrada y que llama al archivo con el nombre de la clase. En él se implementan las funciones que luego el cliente llegará a usar. Estas funciones pueden ser estáticas o requerir de una instancia de la clase.

Los otros dos ficheros son el *parser* y el *controlador*. El primero es el que convierte los datos al formato especificado y el segundo es el que contiene las funciones responsables de realizar peticiones y gestionar el funcionamiento de los métodos principales.

6.4.1 Configuración

Para la configuración de la librería hay que añadir un fichero *.npmignore* que, de la misma forma que el *.gitignore* antes descrito, evita que se suban ciertos ficheros, en este caso a NPM. En caso de no existir este archivo, por defecto se utiliza *.gitignore* pero como en la mayoría de situaciones hay contenidos que sí se deben subir a un sitio pero al no otro, se utilizan ambos ficheros. En esta ocasión, además de los elementos omitidos en Git, se añade el directorio *docs/* ya que así cuando el usuario se descarga la librería para usarla no necesita instalar la documentación.

```
tests
node_modules
package-lock.json
yarn.lock
yarn-error.log
.vscode
docs
```

Programa 9: Fichero .npmignore de la librería.

Para agilizar el trabajo de despliegue se utiliza el fichero *.travis.yml* que, además de validar el código, también permite automatizar tareas. En este caso se especifica que si la aplicación pasa todas las pruebas en la rama *releases* entonces se suba la nueva versión de forma automática a NPM. Para ello se utiliza el correo y un *token* seguro.

```
language: node_js
node_js:
- stable
deploy:
  provider: npm
  email: jcarraldc@gmail.com
  api_key:
    secure:
nHAIWB3LEcfGe9SjQ7JZ//BMKnhtV0f6C1BZ8Xl/WVV5l1ptVO9zlKjaDsMsEJLQq4onQtO+L1d
l9hwjSD0A=
  on:
    branch: releases
    tags: on
```

Programa 10: Configuración de Travis en la libreria.

6.4.2 Módulo University

Este módulo implementa las funciones referentes a la obtención de datos de la universidad. Todos los métodos son estáticos y permiten la obtención de la lista de grados de la universidad pudiendo filtrarse por el campus o facultad en la que se encuentran.
Las funciones disponibles son las siguientes:

```
class University {
  static getGradesList(data){}
  static getGradesURL(){}
  static getCampus(){}
}
```

Programa 11: Clase University de la librería.

6.4.3 Módulo Grades

Esta clase es la responsable de ofrecer la información de un grado. Para crear un objeto se requiere como mínimo del código del grado ya que el de la facultad es opcional. Da la posibilidad de obtener la lista de asignaturas, profesores o el resumen de un grado. Además, también se pueden utilizar las funciones estáticas para consultar el nombre, código o facultad a la que pertenece una carrera.

```
class Grade {
  constructor(code, school){}
  getSummary(){}
  getSubjects(course){}
  getTeachers(){}
  getURL() {}
  get code(){}
  get school(){}
  static getName(code){}
  static getCode(codeName){}
  static getSchool(code){}
}
```

Programa 12: Clase Grade de la librería.

6.4.4 Módulo Subject

El módulo asignaturas permite obtener el contenido de cada materia. Para ello es necesario especificar el código de la asignatura, el del grado y el curso en el que está. La clave del centro es opcional.

```
class Subject {
  constructor(subject, grade, course, school){}
  getSummary(){}
  getDetail(){}
  getSchedule(){}
}
```

Programa 13: Clase Subject de la librería.

6.4.5 Módulo Teacher

Este último bloque es el que implementa las funcionalidades de los profesores. Solo dispone de un único método, el que obtiene las tutorías de los profesores y donde junto a ellas se encuentra un breve resumen del docente, con el nombre, apellido, correo y despacho.

```
class Teacher{
  constructor(id, grade){}
  getTutorships(){}
}
```

Programa 14: Clase Teacher de la librería.

6.4.5 Módulo con las utilidades

Con el fin de mantener el código estructurado hay un quinto bloque para las funciones internas que necesita la aplicación.

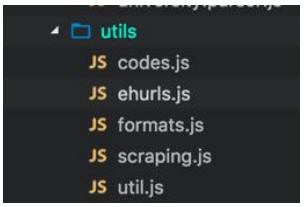


Tabla 12: Estructura del módulo utils de la librería.

Por un lado, el fichero ***codes.js*** contiene dos objetos: uno es un diccionario donde la clave es el código del grado y el valor el nombre de este y en el otro objeto el código del grado devuelve el centro al que pertenece. Esto permite obtener los datos de forma más ágil, evitando acceder a la web de la universidad para tener que consultarlos.

```
module.exports = {
  degrees: {
    'GADEDE30': 'doble-grado-administracion-direccion-empresas-y-derecho',
…,
    'GTRADU10': 'grado-traduccion-e-interpretacion'
  },
  schools: {
    'GADEDE30': '321',
…,
    'GTRADU10': '130'
  }
};
```

Programa 15: Fragmento de código del fichero code.js

Ehurls.js es un fichero que devuelve una clase con todas las funciones necesarias para la obtención de los enlaces de la universidad. Todos los métodos son estáticos y por lo tanto no hay que instanciar la clase.

```
class EHUrls {
  constructor(){}
  static getAllDegrees(){}
  static getDegree(degree, school){}
  static getSubjectsPage(degree, school){}
  static getSubject(subject, school, degree, course){}
  static getTeacherList(gradeName){}
  static getTeacherProfile(degree, id){}
  static getCode(str, url){}
}
```

Programa 16: Clase EHUrls de la librería.

Formats.js implementa una serie de funciones para formatear elementos, como pueden ser los códigos de los campus, las fechas o el valor de los tipos de sesiones de clase.

```
const _campusValue = (str) => {};
const _courseValue = (str) => str[str.length - 1];
const _numToLanguage = num => {};
const _valueToType = str => { };
const _getMonthValue = str => {};
```

Programa 17: Funciones disponibles en archivo formats.js.

Scraping.js es el fichero que realiza las llamadas a la web de la universidad. Hace uso de la librería *request* y tiene la responsabilidad de enviar la petición http y devolver los datos que recibe de la web o un error en caso de fallo.

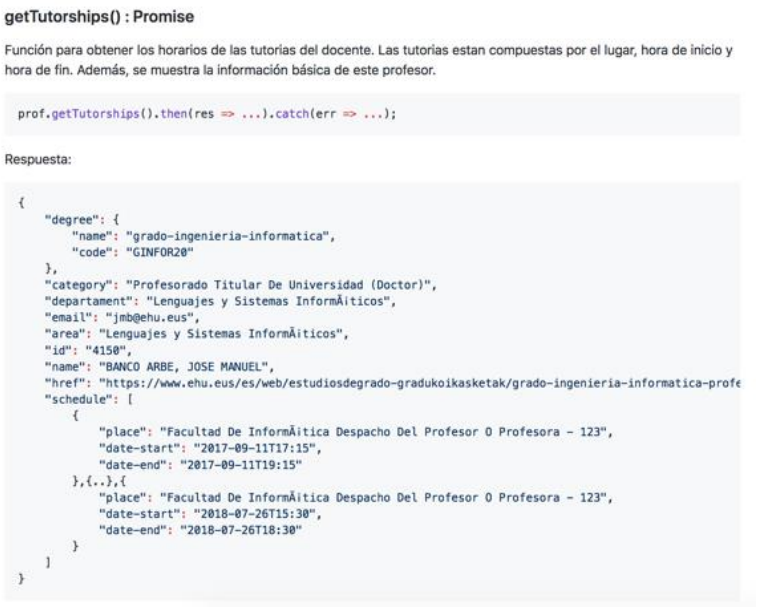
```
const getDataFromWeb = (url) => new Promise((resolve, reject) => {
  if (url.length === 0)
    return reject('Error: Invalid url. [getDataFromWeb]');
  request({
    uri: url,
    encoding: 'binary'
  }, (err, res, body) => {
    if (err)
      return reject(`Err: ${err}. [getDataFromWeb]`);
    return resolve(body);
  });
});
```

Programa 18: Función responsable de extraer el contenido de la web.

6.4.6 Documentación

Otro apartado de la librería es la documentación. Se encuentra dentro del directorio */docs* y en él están tanto los ficheros con la información sobre la api como la guía de colaboración.

Todas las funciones publicas están detalladas dentro de la carpeta /api. Cada clase tiene un documento en formato *markdown* describiendo el uso de estas. En estos documentos se detalla el uso de forma descriptiva, con ejemplos de implementación y con las respuestas de cada una de las funciones.



getTutorships() : Promise

Función para obtener los horarios de las tutorias del docente. Las tutorias estan compuestas por el lugar, hora de inicio y hora de fin. Además, se muestra la información básica de este profesor.

```
prof.getTutorships().then(res => ...).catch(err => ...);
```

Respuesta:


```
{
    "degree": {
        "name": "grado-ingenieria-informatica",
        "code": "GINFOR20"
    },
    "category": "Profesorado Titular De Universidad (Doctor)",
    "departament": "Lenguajes y Sistemas InformÃ¡ticos",
    "email": "jmb@ehu.eus",
    "area": "Lenguajes y Sistemas InformÃ¡ticos",
    "id": "4150",
    "name": "BANCO ARBE, JOSE MANUEL",
    "href": "https://www.ehu.eus/es/web/estudiosdegrado-gradukoikasketak/grado-ingenieria-informatica-profe
    "schedule": [
        {
            "place": "Facultad De InformÃ¡tica Despacho Del Profesor O Profesora - 123",
            "date-start": "2017-09-11T17:15",
            "date-end": "2017-09-11T19:15"
        },{..},{
            "place": "Facultad De InformÃ¡tica Despacho Del Profesor O Profesora - 123",
            "date-start": "2018-07-26T15:30",
            "date-end": "2018-07-26T18:30"
        }
    ]
}
```


Figura 16: Ejemplo de presentación de una función en la api de la librería.

6.4.7 Tests

Para validar la calidad de la librería se han realizado una serie de pruebas unitarias de las funciones más relevantes. Para ello se han utilizado las librerías mocha y chai.

Cada una de las clases exportables ha sido testeada con diferentes casos de uso. Todas las funciones probadas tienen algún caso de uso que comprueba los sucesos exitosos y los fallidos.

Dentro del directorio *test/* están todos los ficheros con las pruebas, uno por cada clase, y dentro de los mismos se ejecutan los casos para todas las funciones.

Un ejemplo es el de prueba de la clase *University*. En el siguiente bloque de código se puede comprobar como para la función *getGradesList* se valida el caso de error y las diferentes opciones válidas en función de los parámetros. Lo que se comprueba no es la ausencia de errores, sino que las respuestas concuerden con el modelo especificado.

```
describe('getGradesList:', function () {
    it('should return an error, wrong campus code', function () {
             return    expect(University.getGradesList({    campus:    'R2D2'
})).to.be.rejected;
    });

    it('should return an object with all the attributes', () => {

        return University.getGradesList({campus: 'BI'})
            .then(res => {
                expect(res).to.be.an('array');
                if (res.length > 0) {
                    const school = res[0];
                    expect(school).to.be.an('object');
                    expect(school).to.have.property('name');
                    expect(school).to.have.property('code');
                    expect(school).to.have.property('campus');
                    expect(school).to.have.property('grades');
                    expect(school.degrees).to.be.an('array');
                    if (school.degrees.length > 0) {
                        const grade = school.degrees[0];
                        expect(grade).to.be.an('object');
                        expect(grade).to.have.property('name');
                        expect(grade).to.have.property('href');
                        expect(grade).to.have.property('code');
                    }
                }
            });
    });

    it('should return all the grades', () => {});
    it('should return all the grades filtered by school', () => {});
});
```

Programa 19: Test para probar la función getGradesList de la librería.

6.5 Implementación del servidor

Este apartado describe cómo está implementado el servidor, EHUServer, y el uso de Firebase en él. Además, se detalla la estructura de directorios utilizada durante la fase de desarrollo y cómo se genera el código final que se sube a producción. Parte del código utilizado se puede consultar en el AnexoD.2 y, además, en el Anexo F se pueden ver los logs de Firebase y el funcionamiento de los servicios desde el cliente web.

6.5.1 Estructura del servidor

Este proyecto solo hace uso de la autenticación, base de datos en tiempo real y funciones de Firebase. Los dos primeros servicios no requieren de una implementación especial, ya que se realizan los ajustes mediante el cliente web. Para las funciones, como hay que implementarlas, sí que se necesita de un repositorio donde presentar el código.
Con el fin de mantener un orden se divide el contenido en función del tipo de disparador que provoca la ejecución de una función. Por este motivo, dentro del directorio principal hay tres carpetas donde están todas las funciones: *database*, *http* y *auth*.

Tabla 13: Estructura del servidor.

Todas las funciones trabajan de una forma u otra con la base de datos de Firebase, para escribir, leer o actualizar datos. Para interactuar con Firebase se crea una clase DB propia que encapsula los objetos firebase y firebase-admin. Es la responsable de inicializar la aplicación y conectar con el resto de los servicios de Firebase.

```
class DB{
  constructor(){
    if(admin.apps.length === 0){
      if(process.env.NODE_ENV === 'development')
      admin.initializeApp({
        credential: admin.credential.cert(firebase.credentials),
        databaseURL: firebase.database,
      });
      else{
        admin.initializeApp(functions.config().firebase);
      }
    }
    this._db = admin.database();
    this._firestore = admin.firestore();
  }

  get db(){
    return this._db;
  }

  ref(path){
    return this._db.ref(path);
  }

  collection(name){
    return this._firestore.collection(name);
  }

  batch(){
    return this._firestore.batch();
  }
}
```

Programa 20: Clase DB del servidor.

El otro fichero importante es el que vincula las funciones implementadas con los disparadores de Firebase. Este archivo es el *index.js,* que se encuentra en la raíz y que importa todas las funciones para luego vincularlas con su *trigger*.

```
const {
  loadData,
  searchByName,
} = require('./http');

const { onSignUp, } = require('./auth');
const { onRemoveSub, onSubscribe } = require('./database');

//HTTP Triggers
//exports.load = functions.https.onRequest(load);
exports.searchByName = functions.https.onRequest(searchByName);

//Database Triggers
exports.onSubscribeSubject=
functions.database.ref('/users/{userID}/subjects')
       .onWrite((event) => onSubscribe(event, 'subjects'));

exports.onSubscribeTeacher=
functions.database.ref('/users/{userID}/teachers')
       .onWrite((event) => onSubscribe(event, 'teachers'));

//Authentication triggers
exports.createNewUser = functions.auth.user().onCreate(onSignUp);
```

Programa 21: Fichero index.js del servidor.

6.5.2 Puesta en producción

Hasta ahora toda la implementación ha sido realizada con la última versión de JavaScript y no es compatible con la versión de Firebase que utiliza para la ejecución, por eso hay que generar código interpretable.
Para esta tarea se utiliza *babel*, un compilador que transforma el código para que sea aceptado por Firebase. Cuando se crea o modifica una función y se desea subir hay que transformarla, para eso se ejecuta un comando que está definido en el package.json y que es el responsable de llamar a babel.

```
$ npm run build
```

Programa 22: Comando para transformar el código a una versión compatible.

```
"scripts": {
    "build": "rm -rf functions && babel src --out-dir functions --copy-files
--ignore src/node_modules && cp src/.eslintrc.json functions/",
    "functions": "firebase deploy --only functions"
  },
```

Programa 23: Fragmento del fichero package.json con el script para compilar.

Con esto se consigue generar la carpeta *functions/* que se subirá a producción ya en formato correcto.
A la hora de subir el contenido se puede hacer de dos formas. La primera es enviando todas las funciones, lo que provocará que se actualice todo el servidor. Esto se hace utilizando el comando:

```
$ firebase deploy --only functions
```

Programa 24: Comando para subir todas las funciones a producción.

La alternativa consiste en desplegar únicamente el método que ha sido modificado y así agilizar la tarea.

```
$ firebase deploy --only functions:nombreDeLaFuncion
```

Programa 25: Comando para subir una función a producción.

6.5.3 Reglas de seguridad

En este último apartado se muestran las reglas de seguridad utilizadas para restringir el acceso a la base de datos. La condición principal es que ningún usuario que no este registrado en el sistema pueda consultar los datos. Además, los datos relacionados con el nodo ehu solo pueden ser modificados por el sistema o por un usuario con el rol de administrador. Por último, al perfil de un usuario solo puede acceder el mismo usuario o un administrador. No podrá consultar los datos personales un usuario que no cumpla con esos requisitos, protegiendo así los datos personales de los usuarios del sistema. Además del control de acceso, también se utilizan las reglas de seguridad para indexar los datos de la base de datos, en este caso para las asignaturas se especifica que se haga a través del atributo *code* de los grupos.
A continuación se muestra el objeto con las reglas de seguridad.

```
{
    "rules": {
        "users": {
            "$uid": {
                ".read": "$uid === auth.uid || root.child('users/' + $uid +
'/role/') === 'admin'",
                ".write": "$uid === auth.uid || root.child('users/' + $uid +
'/role/') === 'admin'"
            }
        },
        "ehu": {
            ".read": "$uid === auth.uid",
            ".write": "root.child('users/' + $uid + '/role/') === 'admin'",
            "subjects": {
                "$key": {
                    "schedule": {
                        "groups": {
                            ".indexOn": [
                                "code"
                            ]
                        }
                    }
                }
            }
        }
    }
}
```

Programa 26: Reglas de seguridad de Firebase.

6.6 Implementación del cliente

En este apartado se detalla la implementación del cliente móvil, EHUApp. Primero se describe la estructura de carpetas utilizada. A continuación, se explica el uso de Redux, la navegación y, por último, cómo se implementa Firebase en este proyecto. En el Anexo D.3 se puede consultar el código para la implementación de EHUApp.

6.6.1 Estructura de carpetas

Como se ha descrito anteriormente, en este proyecto se utiliza la librería Redux para el flujo de datos y el control del estado de la aplicación. Por ello, y como se trata de una aplicación de un tamaño considerable, es importante organizar los directorios de forma que se pueda seguir escalando la aplicación sin que se convierta en algo insostenible.
En la siguiente figura se puede ver una captura del árbol de directorios.

Tabla 14: Estructura de la aplicación.

El punto de partida de la aplicación es el fichero *app.js* que está en la raíz del proyecto. A partir de ahí, todo el contenido está almacenado en el directorio *app/*.

Las carpetas pueden ser de dos tipos. En uno se almacenan ficheros de configuración o funciones reutilizables y en el otro se incluyen módulos con componentes de React y Redux.

El primer grupo lo forman los siguientes directorios:

- Lib: Conjunto de ficheros con funciones sueltas que se reutilizan a lo largo de todo el proyecto. Aquí se encuentran las funciones responsables de utilizar Firebase, normalizar datos o validar campos.
- Config: Carpeta que contiene todos los ficheros que se encargan de configurar el proyecto. Entre estos ficheros destacan los responsables de gestionar las rutas, configurar el *store* y el que fusiona todos los *reducers*.
- Assets: Directorio donde se almacenan las imágenes y estilos que se reutilizan a lo largo de toda la aplicación.
- Components: En esta carpeta se guardan todos los componentes puros de React que se utilizan en diferentes módulos. Estos componentes no tienen ningún comportamiento y son solo elementos visuales.
- Locale: Lista de funciones responsables de la traducción de la aplicación. Aquí se encuentran todos los ficheros de traducción y los que configuran la aplicación para que esta pueda estar en múltiples idiomas.

En el siguiente bloque de directorios se encuentran todos los módulos con componentes de React y Redux. Están agrupados todos por funcionalidad.

- Auth: Almacenan todas las vistas responsables de la autenticación o registro de usuario.
- Admin: Componentes que se encargan de gestionar todas las funcionalidades del administrador.
- Calendar: Vistas que generan y visualizan los calendarios.
- Grade: Componentes para gestionar las fichas detalladas del grado y todas las relacionadas con este.
- Search: Conjunto de componentes responsables del apartado de búsquedas.
- Settings: Elementos visuales responsables de los ajustes de la aplicación.
- Subject: Vistas de las fichas de las asignaturas.
- Teacher: En este directorio se almacenan todos los componentes que se encargan de visualizar las vistas de un profesor.
- User: Componentes responsables de la visualización y edición del perfil del usuario.

6.6.2 Redux

Siguiendo la estructura explicada en el punto anterior, todos los directorios con componentes funcionales de la aplicación tienen asociados un *reducer* y un fichero con acciones.
Las acciones son las que modifican ese *reducer* y se pueden ejecutar desde cualquier punto de la aplicación, pero estas son las únicas que pueden editar el estado de su módulo.

De la misma forma que un componente puede importar una acción de otro directorio, también puede ver el valor de cualquier atributo del estado. Esto se hace gracias a la función *connect* que ofrece la librería react-redux.

```
import { connect } from 'react-redux';

class AdminDatesList extends Component {
 render() {
   const { gradesCalendar, ehuCalendar, navigation } = this.props;
   const selectedCalendarName = navigation.state.params.selectedCalendar;
   return (<AdminDatesListScreen
     handleDeleteItem={this.handleDeleteItem}
     dates={
       selectedCalendarName==='ehu'
       ?ehuCalendar:gradesCalendar[selectedCalendarName]}
   />);
 }
}

//Función que mapea las propiedades del estado
const mapStateToProps = (state, action) => ({
 loading: state.calendar.fetching,
 gradesCalendar: state.calendar.grades,
 ehuCalendar: state.calendar.ehu,
});


//Función para mapear las acciones
const mapDispatchToProps = dispatch => ({
 deleteDateAction: (calendar, type, id) =>
       dispatch(deleteDate(calendar, type, id)),
});

export const AdminDatesList = connect(mapStateToProps,
mapDispatchToProps)(AdminDatesListContainer);
```

Programa 27: Ejemplo de uso de Redux en la aplicación.

Connect es una función *currificada* que por un lado recibe dos funciones para el mapeo de propiedades y por otro el componente que se quiere vincular.
La primera función, *mapStateToProps,* es la responsable de seleccionar los atributos del estado global que el componente recibirá como propiedades. La segunda, *mapDispatchToProps*, es la que permite acceder a las acciones desde las propiedades de la clase. Con esto se consigue que un componente funcional pueda acceder a los datos de cualquier subestado.

Aunque solo existe un único estado, al haber varios *reducers*, se puede decir que este estado está dividido en pequeños subestados. Esto permite tener una organización más adecuada.
En la siguiente figura se puede ver como sería el estado global, un objeto donde los nodos del primer nivel son también objetos, uno por cada módulo.

```
{
  auth: {
    isAuthenticated: false,
    isLoggingIn: false,
    isLoggingOut: false,
    accessToken: null,
    user: {},
    hasInitialUser: false,
    locale: Translate.getLocale(),
    error: '',
  },
  calendar: {
    ehu: {},
    grades: {},
    loadedCalendars: [],
    loadedSchedules: [],
    schedules: {},
    fetching: false,
    error: null,
  },
  grade: {
    grade: {},
    error: false,
    searching: true,
  },
  search: {
    searchSubjectText: '',
    searchTeacherText: '',
    searchGradeText: '',
    selectedIndex: 0,
    searching: false,
    subjects: [],
    teachers: [],
    grades: [],
  },
  settings: {
    locale: Translate.getLocale(),
  },
  subject: {
    subject: {},
    searching: true,
    error: false,
  },
  teacher: {
    teacher: {},
    searching: true,
    teacherData: {},
    error: false,
  },
  profile: {
    data: {},
    subjects: {},
    teachers: {},
    grade: '',
    error: '',
    fetching: false,
  },
}
```

Figura 17: Representación del estado de la aplicación.

6.6.3 Navegación

El siguiente apartado de la implementación, es la navegación: lo que permite moverse entre diferentes componentes de la aplicación en función del nivel de acceso del usuario y de la ruta. Este es el componente principal de la aplicación, el primero en ser llamado cuando se inicia y que se ejecuta cada vez que el usuario intenta navegar entre ventanas.

La navegación está implementada en el fichero routes.js de la carpeta *config/*. Está desarrollado utilizando la librería react-navigation, que proporciona una serie de funciones para crear diferentes tipos de navegadores.
Para organizar las rutas se diferencia entre tres tipos de usuarios: anónimos, aquellos que han iniciado sesión y los administradores. Cada uno de ellos tiene diferente grado de acceso a las rutas y es por ello por lo que se necesitan diferentes objetos de navegación.

React-Navigation ofrece diferentes tipos de navegadores, pero los más utilizados (también en este proyecto) son *StackNavigator* y *TabNavigator*. El primero, *StackNavigator*, agrupa los componentes en una pila y para cambiar de ventana solo hay que invocarla por el nombre que se le asigna. El segundo tipo, *TabNavigator*, es un navegador que utiliza una barra de navegación en el margen inferior y permite cambiar de componente al seleccionarlo en el menú inferior.

El navegador principal es uno que está compuesto por los tres citados anteriormente y por la pantalla inicial conocida como *Splash Screen*. Esta ventana es la primera de la lista y por lo tanto la primera en mostrarse. Desde ella se decide qué navegador se muestra a continuación en función del tipo de usuario.

```
const EHUApp = StackNavigator({
 Splash: {
   screen: Splash,
   navigationOptions: {
     header: null,
   },
 },
 AnonNavigator: {
   screen: AnonNavigator,
   navigationOptions: {
     header: null,
   },
 },
 UserNavigator: {
   screen: UserNavigator,
   navigationOptions: {
     header: null,
   },
 },
 AdminNavigator: {
   screen: AdminNavigator,
   navigationOptions: {
     header: null,
   },
 },
});
```

Programa 28: Fragmento con el navegador principal.

Cuando la ventana *Splash* notifica algún cambio en el estado de usuario, decide a que tipo de navegador se debe cambiar o si se debe mantener.

```
class SplashContainer extends Component {
...
 componentWillReceiveProps = async (newProps) => {
   const {
     isAuthenticated, navigation, user, fetchProfileAction, fetching,
profile,
   } = newProps;
   await wait(500);
   if ((!profile || Object.keys(profile).length === 0) && !fetching) {
     await fetchProfileAction();
   }
   if (isAuthenticated && user.role !== 'admin')
navigateTo('UserNavigator', navigation);
   else if (isAuthenticated) navigateTo('AdminNavigator', navigation);
   else navigateTo('AnonNavigator', navigation);
 }
...
}
```

Programa 29: Implementación de la SplashScreen.

Los navegadores de administrador y usuario son del tipo *TabNavigator* y están compuestos por las mismas vistas, a excepción del administrador, que añade el componente AdminPage donde están disponibles las funciones para usuarios con privilegios.
El navegador para usuarios anónimos es del tipo pila porque no requiere de un menú inferior al estar compuesto únicamente por tres ventanas: Inicio de sesión, registro y recuperación de credenciales.

```
const AdminNavigator = TabNavigator(
 {
   ...sharedTabs, //Objeto con las vistas compartidas
   Admin: {
     screen: AdminStackNavigator,
     navigationOptions: {
       header: null,
       tabBarIcon: ({ tintColor }) => (
         <Icon
           containerStyle={{ justifyContent: 'center', alignItems: 'center'
}}
           color={tintColor}
           name="unlock-alt"
           type="font-awesome"
           size={33}
         />
       ),
     },
   },
 },
 {
   tabBarPosition: 'bottom',
   tabBarOptions: {
     showIcon: true,
     showLabel: false,
     activeTintColor: colors.white,
     inactiveTintColor: colors.grey,
     style: {
       backgroundColor: colors.black,
     },
   },
);
```

Programa 30: Ejemplo de navegador para un administrador.

6.6.4 Firebase

El último elemento de la implementación que se debe mencionar es el de la integración de Firebase con el proyecto.
Para trabajar se debe configurar la aplicación para vincularlo con el proyecto Firebase. Para ello se inicializa el objeto firebase con las credenciales proporcionadas desde la interfaz web en el fichero app.js que es el punto de partida.

```
firebase.initializeApp(config.credentials);
```

Programa 31: Inicialización de firebase.

De forma que todo el contenido esté organizado, la interacción con los servicios se hace a través de una serie de funciones implementadas en un único fichero *firebase.js* que se encuentra en la carpeta lib. Esto quiere decir que no se llama en ningún momento a los servicios de Firebase desde otro lugar que no sea este archivo.
En él se implementan distintas funciones que, mediante el mismo objeto firebase o a través de librerías para realizar peticiones http, se consigue integrar la aplicación con el servidor.

```
//Funcion que consulta los datos mediante peticiones http
export const searchByName = async (name, type) => {
 let query;
 switch (type) {
   case T_SUBJECTS:
     query = 'subject=null';
     break;
   case T_TEACHERS:
     query = 'teacher=null';
     break;
   default:
     query = '';
 }
 const results = await
axios.get(`${urls.firebase}/searchByName?name=${name.toLowerCase()}&${query
}`);
 return results.data;
};

// Función que obtiene los datos directamente de la base de datos.
export const getTeacherFromFirebase = async (teacher) => {
 const ref =
firebase.database().ref(`/ehu/teachers/${teacher.code}_${teacher.grade}`);
 const data = await ref.once('value');
 const res = data.val();
 return res;
};
```

Programa 32: Función para realizar consultas a Firebase usando el servicio Firebase Functions.

6.6.5 Tests

Para probar el código de la aplicación se han realizado una serie de pruebas unitarias que verifiquen el funcionamiento de los componentes de React. Para ejecutar estas pruebas se han utilizados las herramientas Jest y Enzyme, desarrolladas por Facebook y AirBnB respectivamente. La primera librería, Jest, se puede utilizar con cualquier proyecto de JavaScript, sin embargo, Enzyme sirve únicamente para validar los componentes de React.

Las pruebas se han realizado en diferentes ficheros, uno por cada componente testeado, dentro de la carpeta *__tests__* . El nombre de la carpeta para pruebas se ha modificado pues Jest por defecto reconoce el directorio *__tests__* y ejecuta todos los ficheros que se encuentren en él.

En el directorio *__tests__* se encuentra únicamente el fichero jestsetup.js que se encarga de configurar las variables globales para las pruebas.

```
import Enzyme, { shallow, render, mount } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';

// React 16 Enzyme adapter
Enzyme.configure({ adapter: new Adapter() });

// Make Enzyme functions available in all test files without importing
global.shallow = shallow;
global.render = render;
global.mount = mount;
```

Programa 33: Configuración de JEST.

Los componentes que se han probado se encuentran dentro del directorio *components*, allí cada uno tiene su fichero. Para cada componente se comprueba el funcionamiento en todos los escenarios posibles. A continuación se muestra un fragmento de código de las pruebas del componente LoadingScreen.

```
import React from 'react';
import { shallow } from 'enzyme';
import { LoadingScreen } from '../../app/components';

describe('<LoadingScreen />', () => {
  it('should render the LoadingScreen', () => {
    const wrapper = shallow(
      <LoadingScreen />
    );
    expect(wrapper).toMatchSnapshot();
  });
});
```

Programa 34: Ejemplo de prueba del componente LoadingScreen.

7

Soporte del proyecto en GitHub

Una vez descrita tanto la implementación como el análisis y diseño de la aplicación, en este capítulo se expone el uso de la plataforma GitHub para este proyecto. A continuación, se explicará como están organizados los repositorios, los documentos que hay en cada uno de ellos y la organización de la wiki con toda la documentación del proyecto. Por último, se describe el proceso de una contribución real al proyecto.

7.1 Repositorios

Como se detalla en el apartado de diseño y análisis de este documento, el proyecto está dividido principalmente en tres módulos diferentes: EHUScraping, EHUServer y EHUApp. A la hora de almacenar estos paquetes surgen varias alternativas: utilizar un único repositorio, dos repositorios o cada módulo en un repositorio independiente.

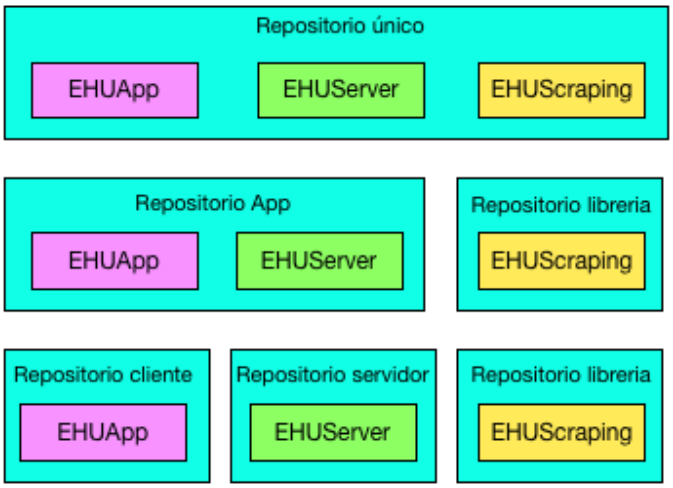


Figura 18: Figura representativa de las tres posibles organizaciones de repositorios.

La primera opción, centralizar todo el código en único proyecto, mantendría un orden de directorios donde cada uno tendría su paquete. Esta alternativa permitiría tener una única documentación, no se repetirían documentos como la licencia, el código de conducta o la guía de contribución. Además, facilita la colaboración ya que solo hay un punto de entrada, de forma que los nuevos colaboradores no tendrían que estar navegando entre diferentes sitios. Como desventaja principal está el hecho de que aumentaría el número de *commits* y por lo tanto la complejidad para la gestión de estos, además de que mantener un orden a la hora de abrir incidencias o *pull request* sería una tarea más costosa.

La segunda opción es agrupar en un repositorio el cliente móvil y el servidor, dejando a un lado la librería de *scraping*. Esta alternativa permitiría poder trabajar la librería de forma completamente independiente ya que es un módulo que no está vinculado, porque, aunque el servidor tenga entre sus dependencias la librería de *scraping,* esta puede ser reutilizada en otros proyectos diferentes.

La última alternativa, utilizar un único repositorio por proyecto, es la que se ha escogido. Se descarta la primera alternativa porque la librería de *scraping* es independiente al proyecto global, aunque haya surgido como una necesidad para este proyecto el alcance es mayor que la aplicación móvil. La segunda opción, también se desecha puesto que cliente y servidor están implementados con tecnologías diferentes y el desarrollo también es independiente ya que la única vinculación que existe entre ambas implementaciones es el de algunas funciones https.
A pesar de todo, los tres repositorios siguen las mismas normas y, por lo tanto, no obliga al desarrollador a tener que aprenderse el funcionamiento de cada una de ellas. Con este diseño

donde se independizan las tres implementaciones, además, se abre la posibilidad de reutilizar la aplicación o el servidor para proyectos de otras universidades donde únicamente se modifique el sistema de extracción de datos.

7.2 Documentación de los repositorios

Como se explica en el punto anterior, aunque los módulos se encuentren en repositorios diferentes, comparten la estructura de la documentación. En este apartado se describen los diferentes escritos de cada una de las documentaciones.

7.2.1 README

Este fichero se encuentra en cada uno de los repositorios, es el punto de partida, aquí se resume brevemente el funcionamiento de cada proyecto. Todos mantienen la misma estructura ya que están compuestos por los siguientes apartados:

- Introducción: Breve descripción del proyecto, acompañado de las etiquetas que indican el estado de cada una de las implementaciones.
- Índice: Lista que enlaza los distintos puntos del documento inicial.
- Instalación: Breve introducción a la instalación del software.
- Funcionalidades: Lista con las funcionalidades más representativas de cada una de las implementaciones
- Tecnologías: Listado con las tecnologías que se han necesitado para desarrollar ese proyecto.
- Contribución: Enlace al documento donde se describe la forma de contribuir al proyecto y a la wiki.
- Licencia: Referencia a la licencia que sigue el proyecto.
- Autores: Mención a los colaboradores de cada proyecto.

7.2.2 CONTRIBUTING

El proceso de colaboración en cualquiera de los módulos del proyecto es el mismo. En cualquier caso, todos los repositorios en el fichero *CONTRIBUTING.md* o en la wiki detallan cómo trabajar, ya que a pesar de seguir unas reglas comunes hay variaciones debido a la naturaleza de cada módulo.

La guía de contribución está estructurada la forma que se describen los siguientes puntos:

- Requisitos: Explicación de los requisitos que debe cumplir una persona para poder colaborar en el proyecto.
- ¿Cómo empezar?: Primeros pasos a la hora de cooperar en el proyecto, aquí se explica cuál es el proceso para empezar a colaborar.
- Reportar un error: En este apartado se expone el proceso para reportar un error en el código, explicando los diferentes tipos de fallos y el proceso a seguir en cada uno de ellos.

- **Propuesta de una nueva funcionalidad:** En este punto se describen los pasos a seguir para realizar una propuesta de mejora o modificación en la aplicación.
- **Versiones**: Descripción del proceso de versionado del software. Para este proyecto todos los módulos siguen el estándar *Semver*.
- **Estilo de código**: Guía con las convenciones para escribir el código. Todos los repositorios siguen el estándar de programación JavaScript propuesto por *AirBnB,* aunque con algunas modificaciones para facilitar el trabajo con cada una de las tecnologías. Todas las peculiaridades están descritas en los documentos de cada proyecto.
- Etiquetas: Listado de etiquetas o *labels* que se utilizan en las incidencias que se abren en GitHub. Hay que organizarlas de forma que se facilite la búsqueda.

Como complemento, y con el fin de facilitar al desarrollador la tarea de seguir con el proceso de colaboración, se añaden varias plantillas que se mostraran por defecto cuando se abran *issues* o *pull requests*. Además, el uso de estas ayudas al resto de usuarios a entender de forma más clara de qué trata.

7.2.2.1 Plantilla para las incidencias

Esta plantilla es utilizada cuando se abre una nueva incidencia, tanto de error como alguna propuesta de mejora. Para poder mostrarla, cuando un usuario abre una nueva incidencia se almacena con el nombre de *ISSUE_TEMPLATE.md* en la carpeta *.github/* de cada proyecto. Está compuesta por una serie de preguntas que debe responder el usuario para ser aceptada.

- **Título**: Título descriptivo de la incidencia. Debe ir precedido por las etiquetas [FEATURE] o [BUG] para poder diferenciar el motivo.
- **Comportamiento esperado:** Si se trata de un error, se debe explicar qué debería suceder si todo funcionase correctamente, en el caso de un cambio debe describir cómo debe funcionar la nueva modificación.
- **Comportamiento actual:** En caso de error, explicar cuál es y cómo sucede. Para una propuesta de cambio explicar las diferencias con el estado actual.
- **Solución:** Este punto es opcional. Se puede proponer una solución al error o cómo implementar cualquier cambio.
- **Pasos seguidos**: Este punto es solo en caso de ser una incidencia de error. Deben especificarse los pasos seguidos hasta encontrase con el error.
- **Contexto**: El desarrollador debe aclarar en qué le afecta esta incidencia o cual es el objetivo de esta.
- **Información del entorno**: Detalles con la información del entorno en que se ha detectado la incidencia. Especialmente importante si se trata de un error.

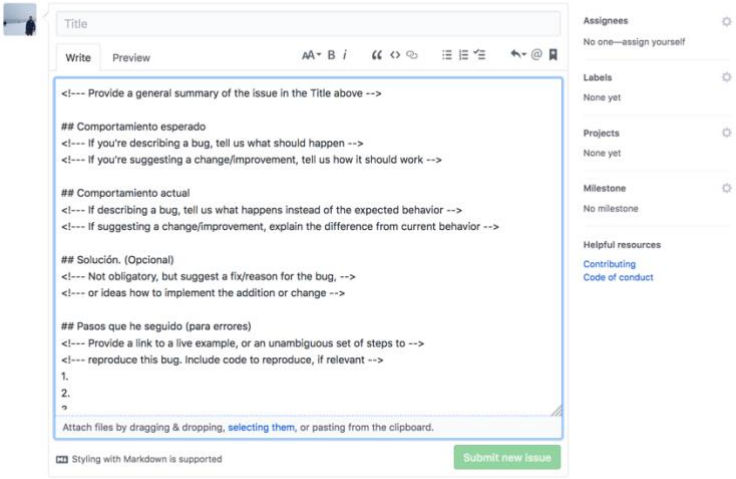


Figura 19: Vista de la plantilla de incidencia en GitHub

7.2.2.2 Plantilla para abrir un Pull Request

Cuando se trata de abrir una nueva *pull request* se utiliza el documento *PULL_REQUEST_TEMPLATE.md,* donde el usuario debe responder a una serie de puntos:

- Descripción: Descripción de los cambios de forma detallada.
- Referencia a la incidencia: Enlace a la *issue* que se ha abierto previamente.
- Motivación y contexto: El desarrollador debe explicar la razón de esta nueva propuesta y qué problema resuelve.
- Capturas de pantalla: Este punto es opcional pero recomendable si se trata de algún cambio en la interfaz.
- Tipo de cambio. En este punto se debe marcar la opción que corresponda al tipo de cambio realizado.
- Checklist: Lista con varios puntos que debe marcar en caso de que sus cambios cumplan con lo que se pide.

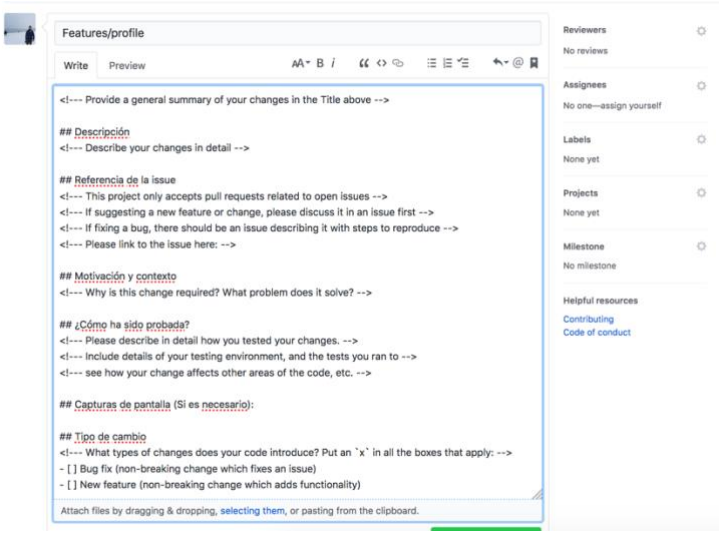


Figura 20: Vista de la plantilla para abrir un pull request en GitHub.

7.2.3 Licencia

Figura 21: Repositorio en GitHub con licencia MIT.

Todos los paquetes software utilizan el mismo tipo de licencia para proyectos *open source*. Se ha escogida la licencia MIT, que se originó en el Instituto Tecnológico de Massachusetts. Esta licencia es los suficientemente permisiva y por lo tanto impone pocas restricciones a la hora de utilizar el código generado.

Las principales características de esta licencia son las siguientes:

- No hay restricciones a la hora de utilizar, modificar o copiar
- La única condición es la de añadir la nota de copyright y el apartado de derechos en todas las copias.
- El software bajo este tipo de licencias se ofrece sin ningún tipo de garantía, limitando así las responsabilidades del propietario.

7.2.4 Código de conducta

El código de conducta de todos los módulos del proyecto es el mismo. Se sigue el estándar de Contributor Covenant utilizando una versión traducida al castellano que se puede encontrar en todos los repositorios con el nombre *CODE_OF_CONDUCT.md*.
Al aceptar este código de conducta, los contribuyentes se están comprometiendo a generar una comunidad sin discriminar a ningún tipo de persona y donde no se aceptan ataques, ni publicación de datos personales, contenido de carácter sexual o una conducta deshonesta. Los responsables del proyecto rechazarán cualquier aporte que vaya en contra de este código.

7.3 Wiki

Además de los documentos expuestos en los puntos anteriores, el proyecto está documentado de forma que se explica de forma detallada el funcionamiento de la aplicación con el objetivo de ayudar a posibles contribuyentes a colaborar y a los usuarios de la aplicación a entender el funcionamiento. Para organizar todo el contenido se utilizan las wikis de GitHub.

7.3.1 GitHub Wikis

Cada repositorio de GitHub tiene una sección que se conoce como Wiki. Este es un apartado que permite organizar toda la documentación del proyecto. Está pensado para proyectos en los que con el *Readme.md* la documentación se queda corta y se necesita generar una base de conocimiento más amplia para los desarrolladores. También es una buena alternativa

cuando se trata de software que tiene más de una funcionalidad y requiere de una explicación más detallada o de proyectos con múltiples fuentes y por lo tanto se necesita centralizar todo en un único sitio.

Una wiki no es más que otro repositorio interno compuesto por documentos de texto que acepta diferentes formatos (Pod, Textile, RDoc, MediaWiki, Markdown, etc) y en la que, por defecto, se usa *markdown*. Como cualquier repositorio permite editar el contenido mediante la interfaz web o de forma local para luego subirlo usando los mismos comandos de Git.

Una wiki está compuesta principalmente por los siguientes cuatro elementos:

- **Página**: Este es documento principal, donde se encuentra el contenido de la documentación. No hay un límite de páginas por wiki.
- **Sidebar**: Se trata de un menú lateral opcional que permite al desarrollador crear una tabla de contenidos que facilite al usuario navegar entre las diferentes páginas.
- **Header**: Cabecera que se repite en todas las páginas de la documentación.
- **Footer**: De la misma forma que la cabecera, este elemento permite generar un pie de página que se repetirá en todas las páginas de la wiki.

7.3.2 Estructura de la wiki

Para este proyecto se ha utilizado una única wiki donde se centra toda la documentación de los tres módulos. Esta wiki se encuentra dentro del repositorio del cliente móvil, aunque desde el resto de los repositorios se enlaza a la wiki principal.

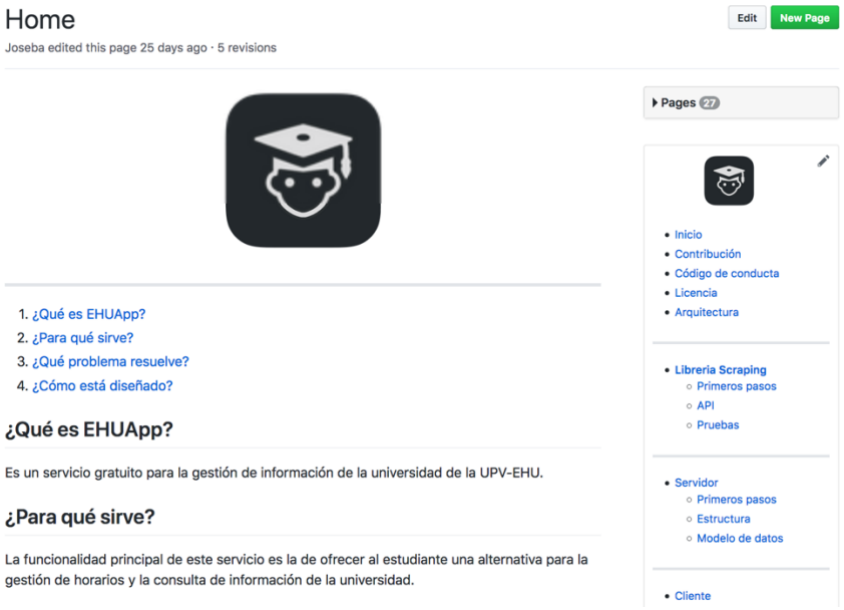


Figura 22: Página inicial de la wiki.

El punto de partida de esta wiki es el fichero *Home,* donde se explica de forma breve el proyecto. En este documento se responde de forma breve a las siguientes preguntas:

- ¿Qué es EHUApp?
- ¿Para qué sirve?
- ¿Qué problema resuelve?
- ¿Cómo está diseñada?

Para organizar la tabla de contenidos se divide en cuatro secciones, a las que se accede a través del menú lateral. El apartado principal es donde está toda la documentación común y en lo que, como se ha especificado anteriormente, se trata de los documentos de contribución, licencia y código de conducta. A estos se les añade un apartado con la arquitectura global del proyecto donde se explica el diseño seguido.

A continuación, se describe el contenido de cada uno de los subapartados del proyecto.

7.3.2.1 Wiki de la librería

En este apartado se describe todo lo referente al uso y desarrollo de la libreria de *scraping*. El documento inicial, de la misma forma que la página de inicio, responde a una serie de preguntas para entender mejor el fin de esta herramienta.

- ¿Qué es EHUScraping?
- ¿Para qué sirve EHUScraping ?
- ¿Qué problemas resuelve?
- ¿Cómo está desarrollada?

Además de este documento se incluyen cuatro más: por un lado, están los orientados al usuario y por otro al desarrollador.
Los documentos orientados al desarrollador son los siguientes:

- Primeros pasos: Guía con los comandos necesarios para poder instalar la librería y poder empezar a trabajar sobre ella.
- Estructura: Documento detallando la estructura de directorios utilizada para organizar la librería.
- Tests: Apartado donde se describe el formato de *tests* que se han ejecutado y cómo realizarlos para poder implementar una funcionalidad nueva.

Finalmente, el usuario que desea hacer uso de la librería también tiene un apartado donde poder consultar toda la información de esta. Este documento es una copia del que está disponible en el repositorio original de la librería. El apartado API es el que describe todas las funcionalidades disponibles. Se detalla desde cómo instalarla para su uso hasta cómo utilizar cada una de las funciones disponibles y están todas acompañadas por ejemplos reales. No se trata de un único documento, son diferentes, pero se accede a través del índice de contenidos disponible en este apartado API.

7.3.2.2 Wiki del servidor

El apartado del servidor, de forma análoga al de la librería, está organizado con un documento inicial que responde a las mismas preguntas y de los mismos documentos enfocados al desarrollador. Además, se incluye un documento con el modelo de datos de la aplicación donde se describe la estructura de datos para que el desarrollador pueda trabajar con la información.

Este modelo de datos explica la forma en la que se almacenan los datos en el servidor, en este caso en único documento JSON en Firebase. Se describen todos los nodos que tiene este fichero, qué tipo de datos contiene cada uno de ellos y cómo se accede a ellos.

7.3.2.4 Wiki del cliente

Por último, el cliente móvil. Este punto sigue el mismo modelo que los anteriores, añadiendo cuatro apartados más: contribución, traducción, casos de uso y generar binarios.

Aunque ya existe un punto entre los documentos comunes donde se especifica la forma de contribuir al proyecto, dentro del subapartado del cliente se vuelve a incluir extendiendo el caso general con las particularidades correspondientes a la implementación.

En el documento con la traducción se describe la forma con la que está hecha la transcripción de la aplicación y cuáles son los pasos que hay que seguir para añadir un nuevo idioma. Está explicado con el fin de que cualquier usuario pueda colaborar en la traducción de la aplicación independientemente de sus conocimientos de la tecnología.

En el documento de casos de uso se encuentran explicadas todas las funcionalidades existentes. Los casos de uso están separados en bloques en función de los privilegios del actor, cada uno de estos viene acompañado por un diagrama UML y una explicación de cada funcionalidad.

En el último punto se explica como generar los binarios necesarios para poder utilizar la aplicación en cualquier dispositivo, Android o iOS. En este apartado se describen todos los pasos que hay que seguir para generar las versiones con todos los certificados, o sin ellos si es para realizar pruebas.

7.4 Publicaciones

Para hacer un seguimiento de las versiones y ofrecer de una forma accesible el software generado se hace uso del apartado *releases* de Github.

Para acceder a las *releases* se hace a través de la página principal del repositorio, donde bajo el nombre se puede ver una etiqueta junto al número de publicaciones que hay disponibles.

Figura 23: Acceso a las releases de EHUApp.

Ya en la página de las *releases* se muestra una lista con todas las que hay disponibles. Una publicación se caracteriza por tres cosas: la versión, la descripción y los adjuntos.

Las versiones de las *releases* siguen el mismo formato que el propio software, en este caso el estándar *semver*. Además pueden ir acompañadas de tres sufijos: *-alpha*, *-beta* y *-rc*.

- ***Alpha*** es el sufijo que se utiliza cuando una funcionalidad está incompleta.
- ***Beta*** se utiliza cuando la funcionalidad está completa pero el software aún no está disponible para ser puesto en producción.
- ***rc*** se utiliza cuando es una versión disponible para ser puesta en ejecución a no ser que haya algún error de última hora.

En la descripción se muestran las funcionalidades que se han añadido o los errores que se han solucionado respecto a la publicación anterior. Si los cambios realizados vienen motivados por *issues* o *pull request,* entonces se deben referenciar en las notas.

Por último se encuentran los adjuntos de cada versión, donde por defecto se añade el código fuente en un fichero *zip* o *tar.gz,* pero también permite añadir otros ficheros. La única limitación para estos archivos es que deben tener un tamaño inferior a 2GB. Para EHUApp en cada publicación se adjunta una apk con la aplicación lista para ser utilizada en dispositivos Android.

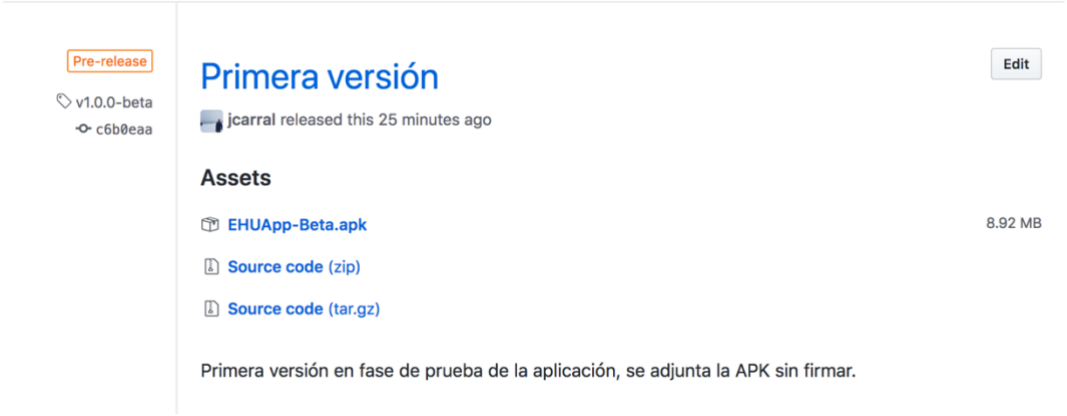


Figura 24: Primera release de EHUApp.

7.5 Caso de uso real

A continuación se describe el proceso de una colaboración realizada por dos personas ajenas al proyecto.

El primer paso es abrir una incidencia, para eso el usuario devBorja creo una nueva siguiendo la plantilla. En esta *issue* pedía que la aplicación estuviera disponible también traducida al euskera.

Figura 25: Pull Request para traducir la app.

Como respuesta, el responsable añade las etiquetas para identificar la incidencia abierta.

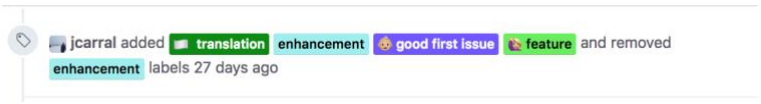


Figura 26: Etiquetas de la incidencia.

Tras varios días, una tercera persona, olatzromeo, sube un *pull request* donde se propone una modificación de la traducción. El mensaje sigue la plantilla propuesta donde se describe el tipo de cambios que ha realizado.

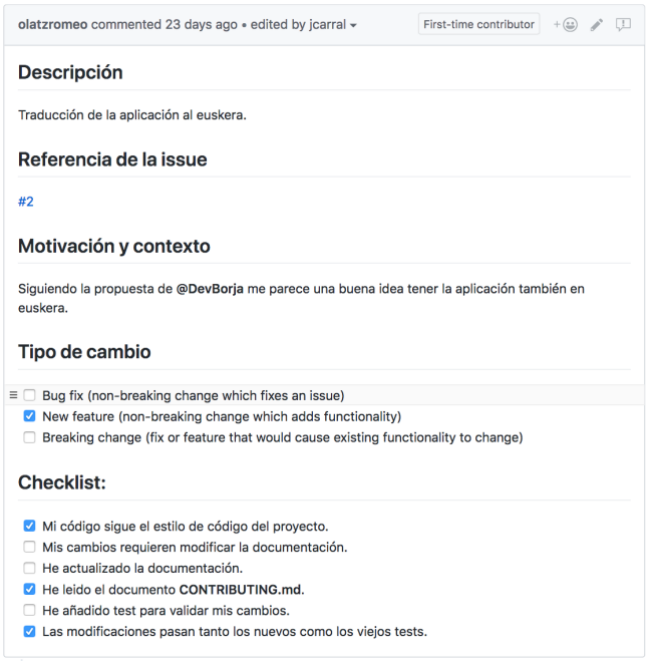


Figura 27: Pull Request abierto por olatzromeo.

Al estar el repositorio vinculado a TravisCI con su correspondiente archivo de configuración, automáticamente se comprueba la validez del código propuesto. El resultado es que no pasa todas las pruebas, ya que no sigue correctamente la guía de estilos.

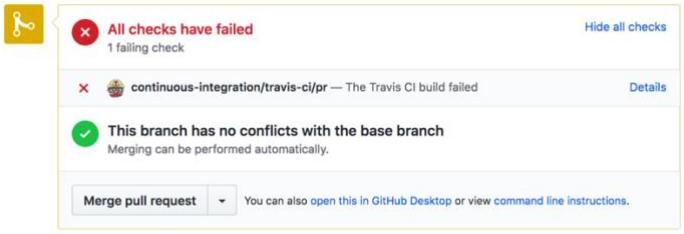


Figura 28: Respuesta de TravisCI a la propuesta.

Finalmente, se le agradece a la usuaria el aporte, pero se le pide que arregle el código. De esta forma se consigue controlar todos los aportes de los usuarios de forma más ágil.

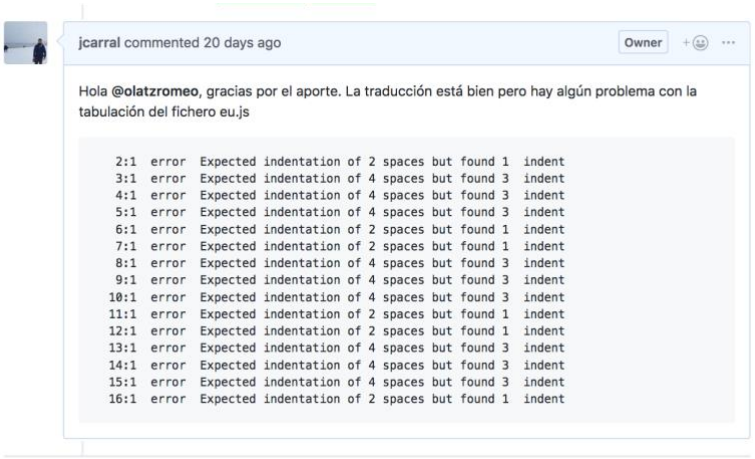


Figura 29: Respuesta del responsable a la propuesta de olatzromeo.

8

Propuesta para la gestión de trabajos de fin de grado soportados por GitHub con un modelo open source

En este capítulo se expone la propuesta de modelo para la gestión de trabajos de fin de grado de la Universidad del País Vasco en la plataforma GitHub.

El capítulo está dividido en tres bloques. Primero se explica la propuesta para los repositorios, seguida de las wikis. Para terminar se propone un arquetipo de repositorio con todos los documentos necesarios para poder implementarlo.

8.1 Modelo de repositorio

Durante el transcurso de este documento se ha ido explicando las consideradas buenas prácticas para el desarrollo de un proyecto *open source* y, en base a las mismas, se ha desarrollado un proyecto que cumple esas pautas. Como resultado de todo eso, a continuación, se propone un modelo de repositorio para cualquier proyecto de código abierto.

Como los trabajos de fin de grado están abiertos al uso de cualquier lenguaje de programación, no es posible definir un modelo único, ya que cada una de estas tecnologías tiene sus estándares. A pesar de eso se pueden establecer una serie de reglas a seguir en todos los proyectos, como la estructura principal de directorios, los principales documentos y la forma de colaborar.

8.1.1 Directorios

Para tener una organización en cualquier repositorio hay que crear una estructura de carpetas que siga un orden. Por esta razón se propone el uso de la siguiente estructura.

```
|
|- .github/
|    |- ISSUE_TEMPLATE.md
|    |- PULL_REQUEST.md
|- docs/
|- tests/
|- examples/
|- src/
|- dist/
|- bin/
|- README.md
|- LICENSE.md
|- CODE_OF_CONDUCT.md
|- CONTRIBUTING.md
```

Tabla 15: Propuesta de estructura

.github/ es la carpeta utilizada para almacenar las plantillas que se mostrarán cuando un usuario vaya a abrir un *pull request* o un *issue*.

Como en algunos casos no es necesario utilizar una wiki para la documentación, ya sea porque el proyecto no es muy complejo o porque el responsable no ve la necesidad de ello, la carpeta *docs/* es la alternativa donde almacenar toda la documentación del proyecto. Dentro de este directorio no se guarda la licencia, el código de conducta o la guía de contribución, ya que está orientada a documentos relacionados con el producto software.

Es recomendable añadir al repositorio algún ejemplo de implementación del producto y para eso se utiliza la carpeta examples/. Se deben añadir ejemplos básicos de uso y al menos una implementación avanzada donde se pueda ver el funcionamiento de todos los casos de uso.

De la misma forma que un buen proyecto no se entiende sin ejemplos de uso tampoco se entendería sin las pruebas, para eso está la carpeta test/, que se debe colocar sobre la raíz del repositorio. En algunas tecnologías las pruebas van incluidas en el código fuente o no se pueden realizar fuera del directorio donde está el código, en ese caso la ruta puede ser

diferente, pero se debe mantener el nombre del directorio y se debe especificar en el fichero *README.md* el lugar de las pruebas.

El código del repositorio se debe mantener dentro de la carpeta ***src/*** y en ella seguir la estructura de carpetas recomendada para cada tecnología. Igual que hay excepciones para las pruebas también las hay para los archivos con código. En la raíz del repositorio se pueden colocar archivos con código como los ficheros índices.

El código sobre el que se desarrolla y el que se genera para enviar a producción habitualmente no es el mismo, y por eso debe haber otra carpeta ***dist/*** donde colocar ese contenido.

Para los binarios generados, y siguiendo el estándar de los sistemas UNIX, se utiliza la carpeta ***bin/***.

8.1.2 Documentos

Además de mantener un orden con los directorios hay que seguirlo también con los ficheros generados, tanto de documentación como los del software que se desarrolle.

Como ya se ha explicado en capítulos anteriores, un proyecto Open Source debe tener al menos el *README.md*, la guía de contribución, el código de conducta y la licencia. Todos estos documentos tienen que estar en la raíz del repositorio. A continuación, se describe el contenido de cada uno de ellos.

8.1.2.1 README

Al ser el primer fichero se considera el más importante y por ello es el que tiene que tener el contenido mejor estructurado y más claro. Con este fin, y siguiendo el ejemplo del proyecto desarrollado, se propone seguir la siguiente estructura:

- Título y descripción: El encabezado debe ir seguido de una breve descripción que introduzca al usuario el software. Además, se recomienda añadir placas o *badges* que muestren el estado del proyecto.
- Tabla de contenidos: En algunos casos este fichero suele ser bastante extenso y por eso acompañarlo con un índice que enlace todos los apartados mejora la experiencia de usuario.
- Primeros pasos: En este apartado se debe explicar de forma breve cuales son los prerrequisitos para usar el software, la instalación y la configuración necesaria.
- Funcionalidades: En la sección de funcionalidades se deben mostrar los casos de uso más representativos de la aplicación.
- Tecnologías: Aquí se deben listar las tecnologías utilizadas, al menos las más importantes para el desarrollo del proyecto. Cada una de estas debe enlazar a su documentación para que en caso de necesitar consultarla sea más sencilla encontrarla.
- Contribución: Aunque exista un archivo dedicado únicamente a ello, también debe añadirse una sección en el *README.md* que enlace a este o haga una breve introducción a la forma de colaborar.

- Licencia: De la misma forma que en el punto anterior, se debe mencionar la licencia utilizada y enlazar al documento que contenga la información de esta.
- Autores: En este apartado se debe añadir al menos una forma de contacto con el propietario y mencionar a los principales responsables del proyecto. Si el número de colaboradores fuese grande podría enlazarse el documento con la lista completa de estos.

8.1.2.2 Contribución

El documento de contribución es al que accederán los colaboradores para empezar a trabajar en el proyecto y, por lo tanto, debe detallar el proceso a seguir. Para esto, se recomienda que esté compuesto de los siguientes apartados:

- Tabla de contenidos: Facilita la navegación entre los diferentes puntos del fichero, que habitualmente es bastante extenso.
- Prerrequisitos: En este apartado se deben especificar cuáles son los requisitos previos para empezar a trabajar como mencionar el código de conducta o el sistema de comunicación.
- Como empezar: En este punto se debe describir el proceso que hay que seguir para comenzar a trabajar en algún cambio y cómo se tiene que abrir un *pull request*.
- Reportar un error: Aquí se tiene que describir como gestionar los errores que se encuentren en la aplicación. Cuando se trata de una vulnerabilidad de la seguridad es importante mencionar al usuario alguna forma de contacto alternativa a las incidencias.
- Propuestas de mejora: De la misma forma que se hace con los errores, hay que describir los pasos a seguir para proponer una funcionalidad nueva o cambio.
- Versionado: Apartado donde se describe el sistema de versiones utilizado. Se recomienda seguir el conocido como *Semver* o Semantic Versioning en el cual las versiones están compuestas por tres elementos en el formato `Major.Minor.Patch` donde:
    - Major: Modificación que no es compatible con versiones anteriores del código.
    - Minor: Cuando se añade alguna nueva característica o se realiza algún cambio sin dejar de ser compatible con una versión anterior.
    - Patch: Correcciones que siguen siendo compatibles.

- Estilo de código: Apartado que menciona cual es el estilo de código que hay que seguir y cómo trabajar con las herramientas que se encargan de verificarlo.
- Etiquetas: Cuando se abre una incidencia o un *pull request*, un responsable puede añadir una etiqueta en función del contenido. En este apartado se mostrará una lista de todas las disponibles, junto a una descripción de cada una de ellas.
- Contacto: Añadir una forma de contacto para que cualquier persona pueda consultar las dudas acerca de este guía.

Estos son los puntos mínimos que debe contener la guía, cualquier apartado extra que se incluya para facilitar el proceso es aceptado. Si el documento se extiende demasiado se pueden enlazar otros ficheros donde se especifique cada apartado en detalle, pero dejando en este, al menos, el resumen de cada una de las secciones.

8.1.2.3 Código de conducta

La elección del código de conducta determina el tipo de comunidad que se quiere crear, los principios y valores del proyecto. A continuación, se exponen los dos códigos que se recomiendan, quedando en manos del responsable de cada trabajo el seleccionar el que considera más adecuado.

- Code Covenant: Este es el escogido para el proyecto desarrollado en este trabajo. Está orientado a cualquier tipo de proyecto, independientemente del tamaño y donde se intenta crear una comunidad positiva sin discriminaciones a ninguna persona por su condición.
- Citizen Code of conduct: Este código de conducta está orientado a proyectos con una comunidad de usuarios grande donde se busca crear un ambiente inclusivo para la mayor parte de participantes independientemente del contexto.

Además de estos dos comentados, que también son los recomendados por GitHub, existen diferentes organizaciones como Twitter, Django o Google que tienen sus propias versiones y que están disponibles también para el uso público.

8.1.2.4 Licencia

Al igual que sucede con los códigos de conducta, la elección de la licencia queda en manos del responsable, pues depende del alcance de cada proyecto. A continuación, se proponen una serie de licencias tanto *copyleft* como permisivas.

- GPL. Conocida como GNU GPL. Es una licencia del tipo *copyleft* que permite la libre distribución del software siempre que cualquier derivado sea bajo la misma licencia.
- MIT. Permite la libre distribución y modificación del software siempre y cuando en los derivados de este se mantenga la nota de copyright.
- Apache. Licencia permisiva que permite la modificación de la licencia en cualquier derivado siempre que se especifique que se está utilizando software con licencia Apache.
- BSD. Similar a la licencia MIT que permite la libre distribución sin restricciones más allá de la nota de Copyright. Permite que el código fuente derivado sea en software privativo.

8.2 Modelo de wiki

Para complementar la documentación de los repositorios se propone el uso del apartado wiki de GitHub para mostrar en detalle la guía del proyecto.
En este apartado se exponen algunos de los documentos necesarios para completar la documentación del proyecto.

- Inicio: Documento principal de la sección, es el equivalente al README y es donde se debe introducir el proyecto intentando responder a una serie de preguntas.
    - ¿Qué es el proyecto?
    - ¿Para qué sirve?
    - ¿Qué problema resuelve?

- Primeros pasos: Documento con los pasos para empezar a trabajar en el proyecto. En este apartado se describe el proceso que hay que seguir para poder trabajar con el software. Aquí se debe especificar cuáles son los prerrequisitos necesarios, como las herramientas adicionales que se necesitan para trabajar, y los pasos para instalarlo.
- Configuración: Si se necesita de una configuración adicional se debe documentar en este apartado.
- Contribución: Documento con la guía de contribución siguiendo el modelo en el apartado anterior.
- Licencia: En este apartado se debe colocar la licencia para el software. Así como en el archivo que se encuentra en la raíz solo se incluye la licencia original, en este se pueden añadir las traducciones y explicaciones necesarias.
- Código de conducta: Como en el punto anterior, en la wiki también se añade el código de conducta del proyecto, aquí también son aceptadas traducciones y explicaciones.
- Análisis y diseño: Documento con la introducción al análisis y diseño realizado para el proyecto. Este documento debe enlazar los ficheros donde se explique con detalle cada unos de los apartados del análisis realizado.
- Arquitectura: En este apartado se expone la arquitectura que se ha diseñado para la implementación del proyecto.
- Casos de uso: Un documento o varios donde se exponen los casos de usos que se han implementado acompañados de los diagramas UML necesarios.
- Modelos de datos: Se trata de otro apartado del análisis y diseño donde se describe el modelo de datos que se ha utilizado.
- Implementación: Sección que debe describir la implementación de la aplicación con cada uno de los apartados que la forman.
- Estructura: La estructura debe describir la organización de directorios que se ha utilizado y el objetivo de cada una de estos.
- Pruebas: Documento donde se explican el tipo de pruebas que se han realizado y las reglas que tienen que cumplir. Además, se debe describir el proceso para añadir más tests y ejecutarlos.
- Funcionalidades: Lista de funcionalidades implementadas y como hacer uso de ellas si fuera necesario, acompañadas de ejemplos de uso y de imágenes que puedan ayudar al usuario a entender el funcionamiento.

8.3 Repositorio

Como resultado de la búsqueda de una metodología para el desarrollo de proyectos open source orientados a los trabajos de fin de grado de la universidad se ha generado un repositorio en GitHub con toda la documentación necesaria para empezar a trabajar. Estos documentos han sido producidos en base a los utilizados para los repositorios que forman el paquete *EHUCalendarGenerator* y es por eso por lo que en algunos de los ejemplos se hace referencia al proyecto original.

Este repositorio contiene las plantillas necesarias para generar la documentación, tanto para los repositorios como para la wiki de estos.
Como no deja de ser otro proyecto open source, está abierto a colaboraciones siguiendo el mismo modelo que propone, por lo que además de plantillas, contiene ficheros que documentan el funcionamiento del repositorio.

Toda la documentación se encuentra inicialmente en castellano, pero el contenido está organizado con la previsión de futuras traducciones.

```
|
|- repos/
|    |- es /* Plantillas en castellano */
|    |    |- .github/
|    |    |     |- ISSUE_TEMPLATE.md
|    |    |     |- PULL_REQUEST.md
|    |    |- licenses/
|    |    |     |- MIT.md
|    |    |     |- GPL.md
|    |    |     |- Apache.md
|    |    |     |- BSD.md
|    |    |- codesofconduct/
|    |    |     |- CITIZEN.md
|    |    |     |- CONTRIBUTORCOVENANT.md
|    |    |- README.md
|    |    |- CONTRIBUTING.md
|    |- en
|    |- eu
|- wiki/
|- examples/
|- templates/
|       |- EXAMPLES-ES.md
|- README.md
|- LICENSE.md
|- CODE_OF_CONDUCT.md
|- CONTRIBUTING.md
```

Tabla 16: Estructura del repositorio con el modelo.

Además, contiene la carpeta *examples* donde se almacenan ejemplos de TFGs que siguen un modelo *open source.* El objetivo de este directorio es ofrecer un lugar al que puedan ir los estudiantes que deseen seguir este modelo para encontrar referencias o como un punto al que acudir a la hora de buscar un trabajo en el que poder colaborar.

9

Gestión

Este capítulo se divide en cuatro bloques, en los que se describen los apartados más relevantes de la gestión del proyecto.
Por un lado, se expone la gestión del alcance y las variaciones sufridas respecto al inicial. A continuación se presentan tanto la gestión del tiempo como la de las comunicaciones. Por último, se explican los riesgos que se han identificado y la forma de mitigarlos.

9.1 Gestión del alcance

Al ser un proyecto pensado para darle continuidad una vez finalice el periodo del trabajo de fin de grado, se tuvieron que limitar los objetivos iniciales y definir una fecha para finalizar esta fase.
El alcance definido inicialmente para esta fase no ha sufrido variaciones significativas. El conocimiento de las tecnologías utilizadas para la implementación de los tres módulos desarrollados ha favorecido el cumplir los objetivos establecidos. Aún así, se desechó la idea de probar la aplicación en usuarios reales de la facultad. Y de esta forma poder invertir ese tiempo en el análisis del desarrollo del modelo open source. El hecho de necesitar de una formación en este apartado y no contar con referencias de este modelo de desarrollo en el ámbito académico ha provocado un desvío del tiempo estimado. A pesar de todo, se han cumplido con los objetivos iniciales, a excepción de la prueba con estudiantes.

9.2 Gestión del tiempo

En este apartado se describe la gestión del proyecto a lo largo del ciclo de vida del trabajo de fin de grado. Se empieza con la exposición de las horas planificadas, las que finalmente se han invertido en cada área y los desvíos más significativos que se han producido. A continuación se describen los periodos de trabajo, desde el inicio hasta la entrega final con los repartos de tareas. Para finalizar, se muestran los hitos más relevantes del TFG.

Para la gestión del tiempo hay que tener en cuenta los antecedentes del proyecto, pues como se ha especificado en puntos anteriores se parte de la idea que se concibió en el verano de 2016, aunque no es hasta septiembre de 2017 cuando comienza realmente el proyecto. Además, se trata de un proyecto al que se le tiene pensado dar continuidad una vez termine el trabajo de fin de grado y por lo tanto, este apartado solamente incluye la gestión para esta etapa.

En la siguiente tabla se muestran las horas estimadas al inicio del proyecto, las que finalmente se han invertido y el desvío que ha habido.

		Estimado	Invertido	Desvio
EHUCalendarGenerator	Analisis y diseño	50	40	-10
	Implementacion EHUScraping	20	40	20
	Implementacion EHUServer	20	15	-5
	Implementacion EHUApp	100	80	-20
TFG-OSModel	Formacion	25	30	5
	Analisis y diseño	20	30	10
	Documentación producto	30	40	10
	Desarrollo modelo	30	35	5
Gestion	Planificacion inicial	30	35	5
	Seguimiento y control	15	20	5
	Reuniones	10	12	2
	Memoria	60	70	10
	Total	410	447	37

Tabla 17: Tabla con las horas estimadas, invertidas y las desviaciones de estas.

Como se puede ver se distinguen tres bloques: EHUCalendarGenerator, TFG-Open Source Model y gestión. La previsión inicial de horas a invertir es superior a las 300 y a pesar de eso, la cantidad real de tiempo empleado es superior a lo planificado. Este desvío se debe principalmente a que se trata de un proyecto para el que se carecía de experiencia en cuanto a la metodología *open source* se refiere. Tampoco se conocían referentes en el ámbito académico y por lo tanto ha necesitado de un tiempo añadido para obtener una solución completamente nueva. Si bien es cierto que el bloque del producto es el que tiene el menor desvío, la implementación de la librería ha requerido del doble del tiempo estimado debido a los múltiples cambios que ha sufrido la web de la universidad, que han obligado a realizar diferentes versiones de la librería.

A continuación se describen los intervalos de tiempo. Para ello, además de un diagrama de Gantt se aprovechan las herramientas que proporciona GitHub para hacer un seguimiento.

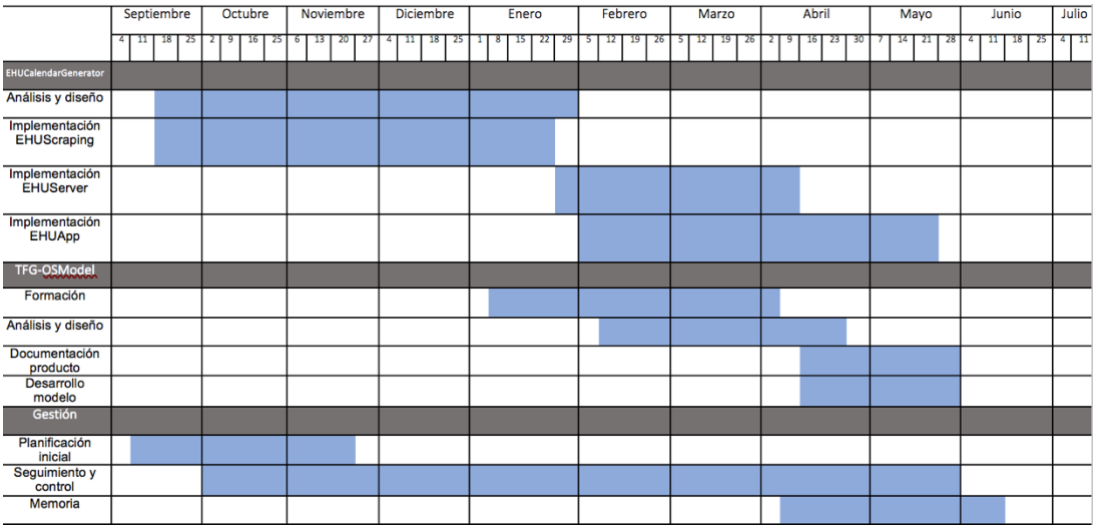


Tabla 18: Diagrama Gantt.

Como se puede ver en el diagrama anterior, el proyecto comenzó en septiembre de 2017 cuando se hizo la propuesta al tutor, José Miguel Blanco. Los primeros meses del proyecto estuvieron enfocados a la planificación y la adaptación del trabajo realizado para MagnaSis. El bloque de gestión tiene presencia a lo largo de todo el ciclo de vida, siendo el inicio y final cuando mayor impacto tiene.

El apartado para la metodología comienza en enero de 2018 con la formación puesto que uno de los requisitos del producto es que siga un modelo de código abierto. Aunque el comienzo es a principios de año, no es hasta la última parte del proyecto cuando aumenta la importancia de este bloque ya que es cuando tras haber generado el producto se procede a realizar un análisis para generar un modelo.

Por último, en cuanto al bloque EHUCalendarGenerator, es en febrero cuando se puede ver el aumento de la dedicación a pesar de que comenzó con en septiembre con la librería. Para analizar mejor este apartado se utilizan los diagramas proporcionados por GitHub. Estos gráficos se generan automáticamente y muestran la frecuencia con la que se han realizado los *commits* en cada repositorio. Gracias a esto se puede hacer un seguimiento, ya que están disponibles de forma pública a través de la interfaz web de la plataforma dentro del apartado *Insights*.

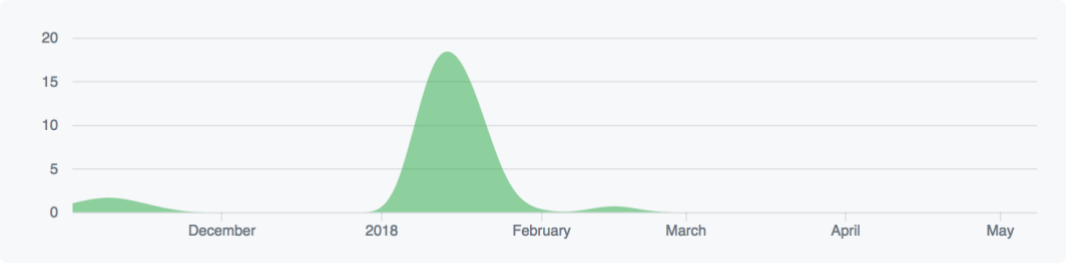


Tabla 19: Periodo de los commits de la librería.

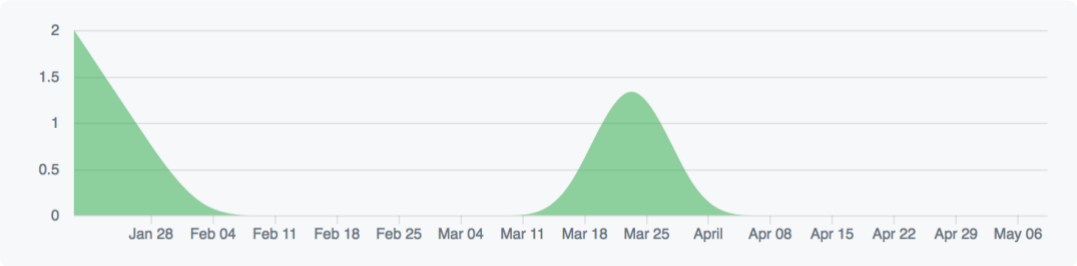


Tabla 20: Periodo de los commits del servidor.

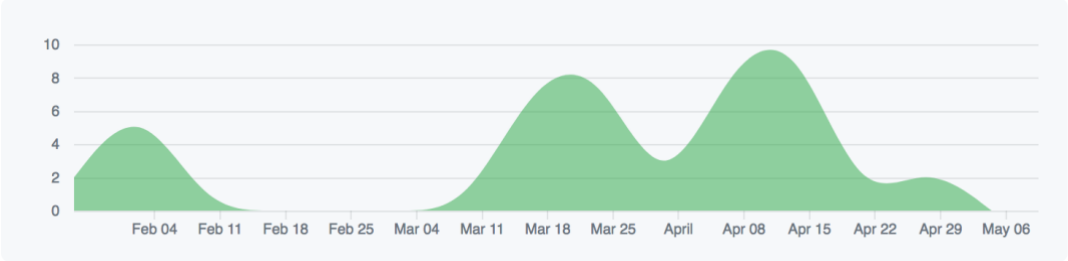


Tabla 21: Periodo de commits de la aplicación.

Como se puede ver, la librería es la primera sobre la que se trabaja, siendo el mes de enero cuando mayor frecuencia de trabajo hay. Al mismo tiempo que se termina con esta, empieza el proceso para el desarrollo del servidor, al que le sigue el cliente, y que continúan durante los meses de febrero, marzo y abril.

Para finalizar con el apartado de gestión de tiempo se muestran los hitos a lo largo del proyecto. Como se ha mencionado anteriormente, el primer hito, en septiembre, es el que da comienzo a este trabajo, cuando se acepta la propuesta para el desarrollo de este.
Hay que destacar un hito a mitad de proyecto, cuando se le expone al tutor y principal interesado el modelo de gestión de proyectos *open source*.
Finalmente, también se deben mencionar los hitos relacionados con el cierre de las implementaciones que ocurren durante el mes de abril. Estas fechas se marcaron como claves con el fin de dejar tiempo suficiente para generar un modelo reutilizable para futuros trabajos de fin de grado.

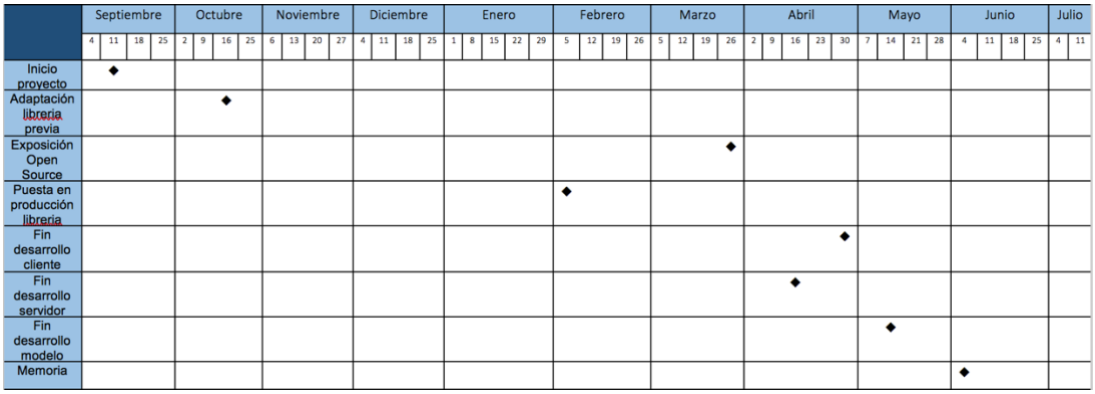


Tabla 22: Diagrama de hitos.

9.3 Gestión de las comunicaciones

Uno de los apartados a tener en cuenta en este proyecto ha sido la gestión de las comunicaciones, no solo con el tutor y principal interesado del proyecto, sino que con los colaboradores del proyecto.

La comunicación con el tutor ha sido de forma presencial, acudiendo a su despacho para realizar las reuniones de seguimiento del proyecto o la consulta de dudas. Durante la primera fase del proyecto, desde septiembre a diciembre, no hubo reuniones más allá de la de comienzo. Esto se debe a que durante este periodo se compaginaban las clases y el proyecto, por lo que la dedicación era parcial y estaba más enfocada a la adaptación del trabajo previo y planificación. No es hasta la segunda reunión cuando las reuniones empiezan a tener una frecuencia mayor, aproximadamente cada 3 semanas. En el Anexo A se encuentran los ordenes del día de todas las reuniones.

Como se trata de un proyecto donde se utilizan herramientas que no se trabajan mucho en la universidad, se han buscado colaboradores con los conocimientos suficientes para que pudieran contribuir sin la necesidad de formarse expresamente para este trabajo. Para ello, a principio del proyecto se les preguntó, tanto a Borja Diez como a Olatz Romeo (profesionales y graduados en ingeniería informática), si estaban interesados en colaborar realizando una prueba de aportación en un proyecto *open source*. Esta consulta se realizó de forma presencial. Se recibió una respuesta positiva en ambos casos y se les pidió que intentaran realizar un aporte siguiendo la documentación disponible en el repositorio del cliente. Todo el proceso de comunicación se realizó de forma personal o mediante la plataforma de mensajería Telegram donde consultaban las dudas. El resultado es el que se describe en el apartado 7.4 de esta documentación.

9.4 Gestión de riesgos

A continuación se describen los principales riesgos para el proyecto y las decisiones tomadas para mitigarlos.

Aunque por experiencias anteriores ya se conocía la posibilidad de que la web de la UPV-EHU se modificase, no se tuvo en cuenta el impacto de un posible cambio, pero gracias a la arquitectura modular solo afecta de forma directa a la librería de *scraping*. Aunque se trata de un riesgo bastante probable, se ha tomado la decisión de mantener una copia de seguridad parcial en la base de datos a la que se accedería en caso de fallo.

El otro riesgo relacionado con el producto, y también con la librería, es el que implica la gran cantidad de peticiones que se necesitan para obtener todos los datos de la web. Realizar una carga completa de los tres campus requeriría más de 50.000 peticiones http lo que podría causar un grave problema a los servidores de la UPV-EHU si esto no se gestiona correctamente. Para ello, y como la aplicación no está pensada para ser lanzada al mercado aún, se limita únicamente a los grados de Ingeniería Informática de Guipúzcoa, Ingeniería Química del campus de Vizcaya y el de Ingeniería en Automoción del campus de Álava. De esta forma se puede probar con un grado de cada sede. Además, también se limita la frecuencia de las peticiones por cada volcado de datos, es decir, cuando se carga un grado, en vez de realizar todas las peticiones simultáneamente, se hacen secuencialmente con un segundo de diferencia.

Al tratarse de un proyecto *open source* y estar todo el código accesible en Github existe el riesgo de publicar cualquier dato sensible como contraseñas o datos personales. Para el primer caso, contraseñas y claves para *apis*, esta información se añaden en ficheros que posteriormente se ignorarán mediante el archivo *.gitignore*. Para datos personales, como los nombres de profesores, que se utilizan para los ejemplos se ha decidido utilizar nombres ficticios. Por lo tanto, ninguna de las personas que aparece en las capturas de pantallas o ejemplos que se utilizan en la documentación es real.

En cuanto a los riesgos de gestión, el principal es la inexistencia de un modelo referente para la gestión de proyectos *open source* en trabajos de fin de grado y por lo tanto el tener que explorar la viabilidad de uno sin tener un precedente. Para mitigar este riesgo se han marcado unas fechas para el cierre de las implementaciones con el suficiente margen como para requerir de tiempo extra y que los desvíos no influyan en la viabilidad del trabajo de fin de grado.

10

Conclusiones

En este capítulo se exponen las conclusiones obtenidas tras el desarrollo del trabajo. Por un lado, se describen las conclusiones, tanto personales como las de los diferentes bloques del trabajo de fin de grado y, por otro, se analizan las posibles líneas futuras del proyecto.

10. 1 Conclusiones del proyecto.

En cuanto a las conclusiones obtenidas gracias a este proyecto, habría que dividirlas también en tres bloques: relacionadas con el desarrollo, con el modelo de trabajo open source y de tipo general.

Primero se encuentra la implementación de la aplicación. A pesar de tener una formación previa en las tecnologías no he dejado de adquirir nuevos conocimientos, al ser este proyecto un primer acercamiento al desarrollo de una aplicación real. El tratarse de un proyecto relativamente grande ha servido de aprendizaje para entender la importancia de realizar un análisis previo y de una planificación del desarrollo. Como resultado de todo esto se ha obtenido una aplicación que es perfectamente funcional para ser utilizada en un futuro por la UPV-EHU.

El desarrollo de un modelo *open source* ha supuesto un auténtico reto, pues la formación en este aspecto no era la deseada. Con el aprendizaje en este tema se han obtenido varias conclusiones. Primero, la importancia de una documentación de calidad. Cuando se buscan proyectos el desarrollador tiende a desecharlos si no hay una documentación que le introduzca y ayude a trabajar, en ocasiones el código pasa a quedar en un segundo plano. Por lo tanto, debe ser un requisito indispensable para cualquier trabajo donde se busca la colaboración. Segundo, y más importante, es la fuente de conocimientos que supone. Trabajar en proyectos *open source* te enseña muchas cosas que no se aprenden en el ámbito académico. Aprender a comunicarte con gente muy diversa y de la que puedes adquirir conocimientos de todo tipo. Exponer tu trabajo a los demás, que esté abierto a críticas y aportes de terceras personas es otra forma de mejorar como desarrollador. Pero no hace falta colaborar en un repositorio para aprender, basta con explorar entre los millones de repositorios que hay y encontrar uno que seguir, e ir analizando cómo se trabaja esa tecnología, las metodologías que usan o el sistema de comunicación que mantienen. Con eso ya se obtiene una información muy valiosa. Gracias a la gran comunidad de desarrolladores que hay, este modelo se ha convertido en una alternativa de aprendizaje tan interesante como las convencionales.

Por último, las conclusiones generales. La importancia de una buena gestión del tiempo ha sido una de las lecciones aprendidas más importantes. Habitualmente se tiende a intentar generar un producto con un alcance muy amplio, sin tener en cuenta el tiempo disponible, y el resultado suele ser algo incompleto. Por eso el definir unos objetivos iniciales limitados en función del tiempo ayuda a obtener una solución más valiosa.

El conocimiento adquirido gracias a este trabajo no se queda en las tecnologías o en la metodología *open source*. La necesidad de explorar nuevas alternativas, realizar una buena planificación, una gestión de todos los aspectos que abarcan el proyecto, el análisis o el diseño, todo en conjunto, me ha ayudado a entender mejor lo que es realmente la ingeniería del software.

10.2 Líneas futuras

Si bien es cierto que con la entrega del trabajo de fin de grado termina una etapa importante de este proyecto, su propia esencia está orientada a que pueda tener continuidad en el futuro. Las líneas de continuación son tan amplias como las que las propias personas que se acerquen a los repositorios quieran explorar, aun siendo así se concretan a continuación algunas posibilidades que, por su inmediatez, puedan ser abordadas en un futuro cercano.

Por un lado, se refieren al código desarrollado, la librería, el servidor y cliente que forman la herramienta para la gestión de calendarios y por el otro lado se encuentra la búsqueda de un modelo *open source*.

10.2.1 Herramienta para la gestión de horarios - EHUCalendarGenerator

En cuanto a los tres módulos que forman el producto software se tienen identificadas una serie de funcionalidades que fueron excluidas cuando se realizó la planificación.

- Obtener los datos de la universidad en los tres idiomas disponibles: castellano, inglés y euskera.
- Mejorar la eficiencia de la librería.
- Mejorar la interfaz de la aplicación móvil.
- Permitir descargar los calendarios o compartirlos a través de la aplicación.
- Añadir la posibilidad de interactuar entre profesores y alumnos de forma directa a través de la aplicación.
- Añadir nuevos idiomas a la aplicación.
- Desarrollo de aplicación adaptándose a la RGPD.

10.2.2 Modelo open source – TFG-OpenSourceModel

Al tratarse únicamente de una propuesta de modelo, esta es la que está más abierto a posibles mejoras o modificaciones y por lo tanto el tiene un futuro más abierto. El objetivo es lograr en un futuro implantar el modelo de trabajo de fin de grado *open source* en la facultad. Con este fin en mente se va a continuar estudiando y realizando mejoras. Entre las modificaciones que se prevén está la adaptación de los contenidos a diferentes idiomas, empezando por el euskera.

Anexo A: Órdenes del día de las reuniones

Fecha	14/09/2017 – 15:30
Lugar	Despacho del tutor, FISS.
Asistentes	Joseba Carral y José Miguel Blanco.

Orden del día:

- Inicio del proyecto
- Planificación primera parte

Fecha	21/12/2017 – 15:30
Lugar	Despacho del tutor, FISS.
Asistentes	Joseba Carral y José Miguel Blanco.

Orden del día:

- Definir el alcance y marco del proyecto.
- Organización de los antecedentes

Fecha	16/01/2018 – 15:30
Lugar	Despacho del tutor, FISS.
Asistentes	Joseba Carral y José Miguel Blanco.

Orden del día:

- Revisión de la planificación
- Organización de los antecedentes
- Determinar un MVP
- Antecedentes en proyectos open source. Mostrar el pull request en GitPoint.

Fecha	08/02/2018 – 15:30
Lugar	Despacho del tutor, FISS.
Asistentes	Joseba Carral y José Miguel Blanco.

Orden del día:

- Revisión de los objetivos para la reunión.
- Gestión del volcado de datos desde la web.
- Definir los repositorios en GitHub del proyecto.

Fecha	26/03/2018 – 16:30
Lugar	Despacho del tutor, FISS.
Asistentes	Joseba Carral y José Miguel Blanco.

Orden del día:

- Exposición de un proyecto open source
- Revisión de los próximos hitos: Implementación y documentación de los repositorios.
- Licencias para los repositorios.

Fecha	23/04/2018
Lugar	Despacho del tutor, FISS.
Asistentes	Joseba Carral y José Miguel Blanco.

Orden del día:

- Pull Request de terceras personas.
- Uso de la wiki como sistema para documentar los proyectos.
- Propuesta de meta repositorio.

Fecha	14/05/2018 – 15:30
Lugar	Despacho del tutor, FISS.
Asistentes	Joseba Carral y José Miguel Blanco.

Orden del día:

- Dudas con la estructura de la memoria.
- Revisión de los hitos.

Anexo B: Entregables

Los cuatro entregables resultantes de este proyecto están disponibles públicamente a través de la plataforma GitHub, organizados en repositorios.

La propuesta de repositorio para proyectos de fin de grado usando un modelo open source se encuentra disponible en la siguiente url:

- https://github.com/jcarral/TFG-OpenSourceModel

Figura 30: Anexo B: Vista principal repositorio modelo

La librería para realizar el scraping está disponible en un repositorio GitHub y en la plataforma NPM mediante los siguientes enlaces:

- https://github.com/jcarral/ehu-scraping
- https://www.npmjs.com/package/ehu-scraping

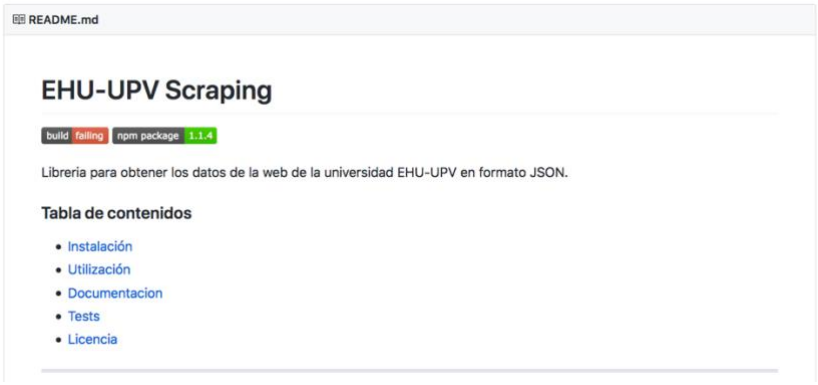


Figura 31: Anexo B: Vista principal repositorio libreria

Los repositorios con el servidor y la aplicación móvil son también accesibles a través de los siguientes enlaces:

- [Cliente] https://github.com/jcarral/EHUApp
- [Servidor] https://github.com/jcarral/EHUServer

En el mismo repositorio del cliente, EHUApp, se encuentra el apartado con las últimas versiones publicadas donde se adjuntan las APKs generadas.

- https://github.com/jcarral/EHUApp/releases

Por último, la wiki, está disponible en el repositorio del cliente o se puede acceder directamente siguiendo el siguiente enlace:

- https://github.com/jcarral/EHUApp/wiki

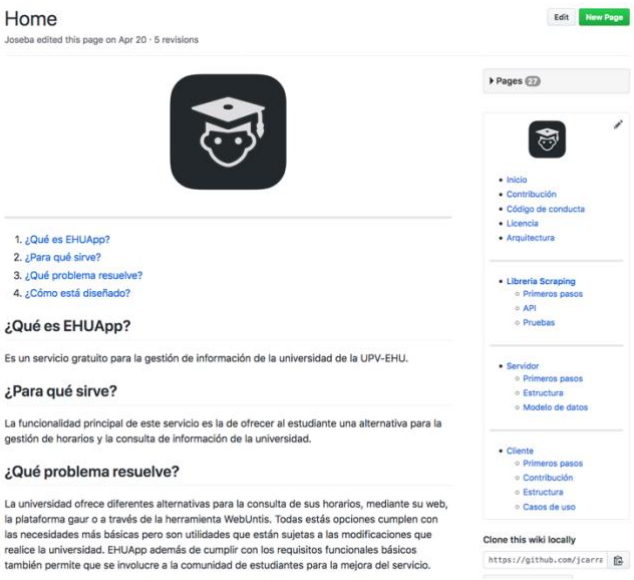
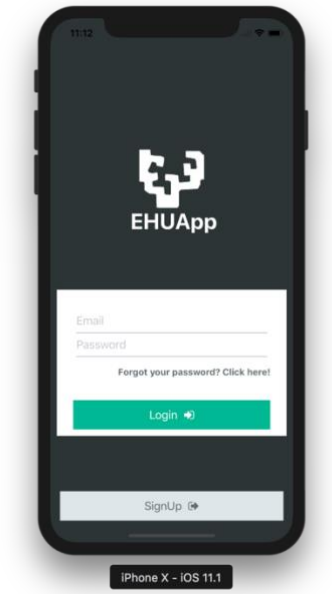


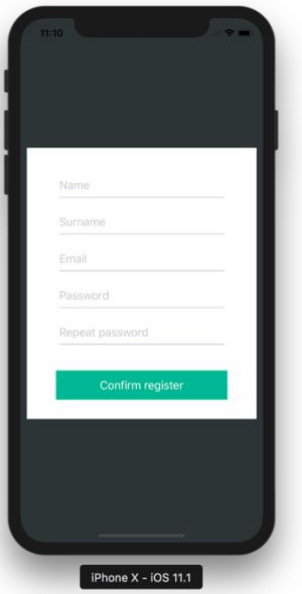
Figura 32: Anexo B: Vista principal de la wiki

Anexo C: Interfaces de la aplicación

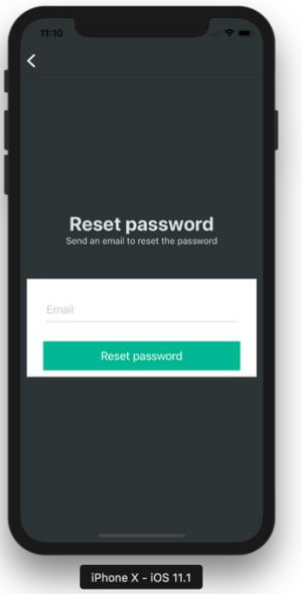
En este apartado se muestran todas las interfaces de la aplicación. Para estos ejemplos se ha utilizado la versión de la aplicación para iOS 11.1 y el dispositivo es un iPhone X.



Vista inicio de sesión

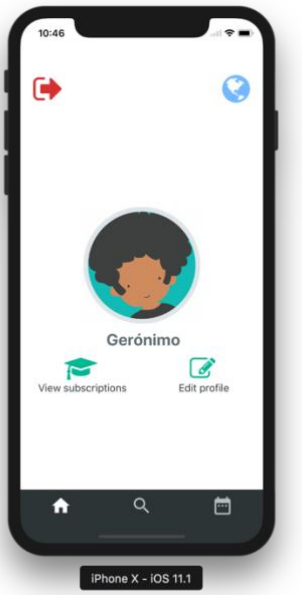


Vista registro

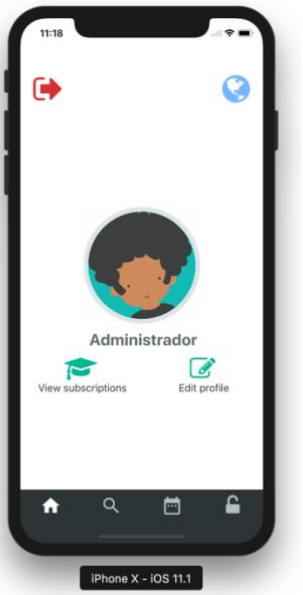


Vista recuperar contraseña

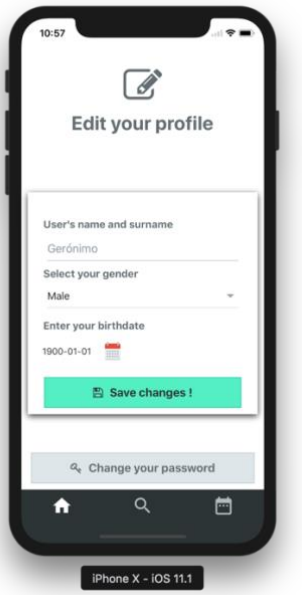
SplashScreen



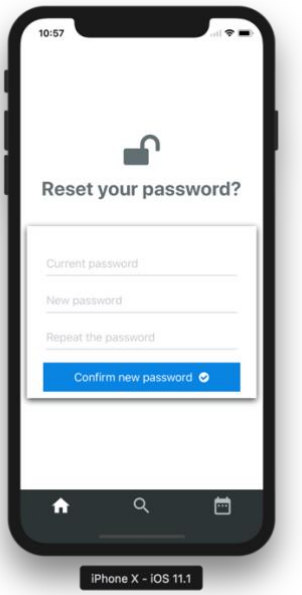
Perfil de usuario normal



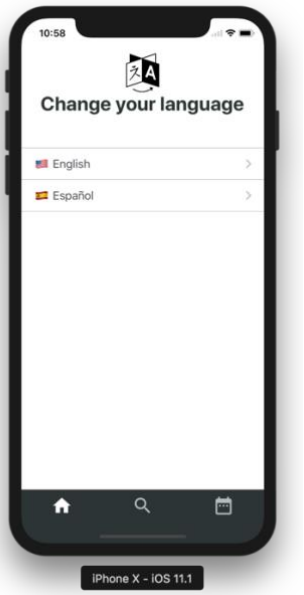
Perfil de usuario administrador



Vista editar perfil



Vista editar contraseña



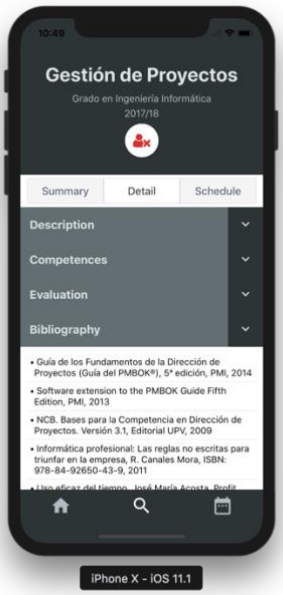
Vista editar idioma

Vista suscripciones

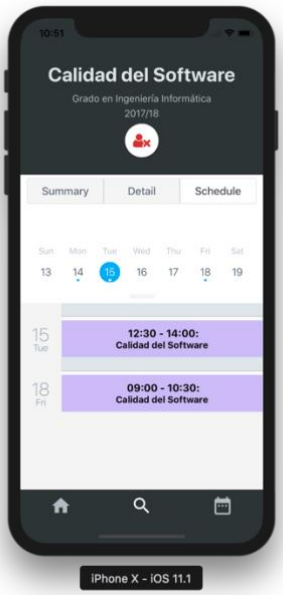
Vista búsqueda vacía

Vista búsqueda

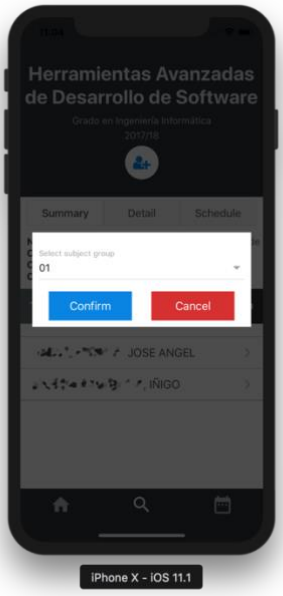
Vista principal asignatura



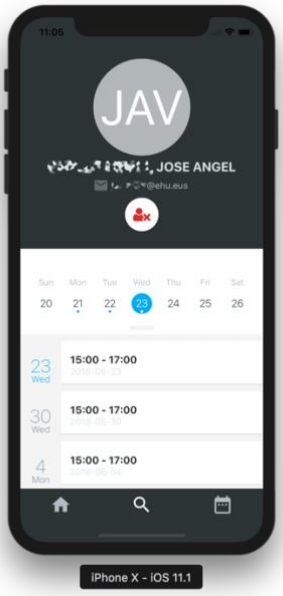
Vista detalle asignatura



Vista horario asignatura



Vista suscripción asignatura

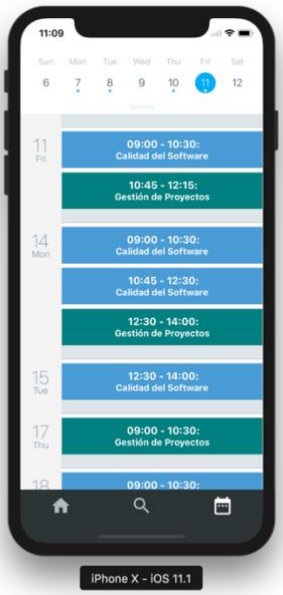


Vista principal profesor

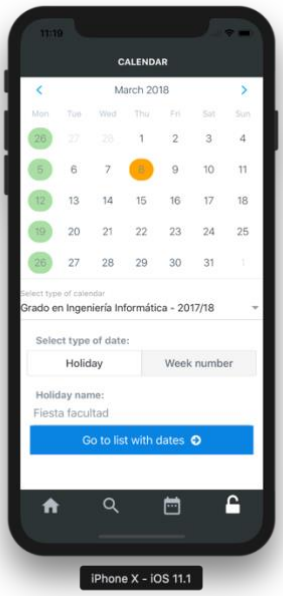
Vista principal grado

Vista lista asignaturas de grado

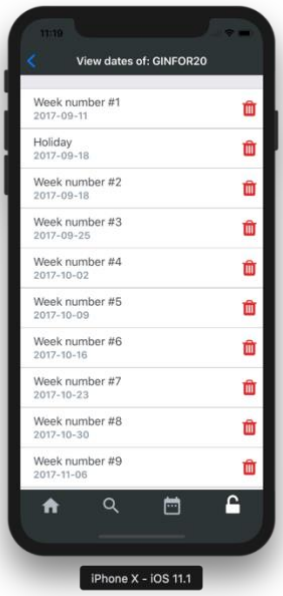
Vista lista de profesores de grado



Vista calendario personal



Vista calendario festivo del administrador



Vista lista festivos del administrador

Anexo D: Código

D.1 Ejemplos de código de EHUScraping

El siguiente fragmento de código muestra la función responsable de obtener el resumen de una asignatura a partir del contenido de la web. Está función se encuentra en el fichero *parser* del módulo asignatura.

```
const _parseSubjectSummary = (data, school, degree) => new Promise((resolve, reject) => {
          if (!data) return reject('Can\'t parse empty data');
          else if(!school) return reject('Error: can\'t parse subject summary without school code
[parseSubjectSummary]');
          else if(!degree) return reject('Error: can\'t parse subject summary without degree code
[parseSubjectSummary]');

          const $ = cheerio.load(data);
          let subject = {};
          const container = $('#contenedor');
          const title = getTitleAndCode($);

          subject.name = title.name;
          subject.code = title.code;
          subject.school = {};
          subject.degree = {};
          subject.teachers = [];
          subject.languages = [];
          subject.school.code = school;
          subject.degree.code = degree;

          $(container).find('ul').each(function(index){
                    getSummaryBasic($, this, subject, index);
          });

          $(container).find('.asig').each(function(){
                    const currentTeacher = getTeacher($, this);
                    pushTeacher(currentTeacher, subject.teachers);
                    currentTeacher.languages.forEach((language) => pushLanguage(language, subject.languages));
          });

          subject.href = EHUrls.getSubject(subject.code, subject.school.code, subject.degree.code,
subject.course);

          return resolve(subject);
});
```

Programa 35: Anexo D: Función del *parser* de la asignatura.

En el siguiente trozo de código se puede ver como a partir de una *url* se llama a la función que se encarga de realizar la petición a la web [Capítulo 6.4.5] para que finalmente se invoque a la función anterior con el contenido de la web. Este fragmento hace referencia al fichero subject.ctrl.js.

```
const getSubjectSummary = (subject, school, degree, course) => {
          if (!subject || typeof subject !== typeof 'string') return Promise.reject('Error: Subject code is
required. [getSubjectSummary]');
          else if (!school || typeof school !== typeof 'string') return Promise.reject('Error: School code is
required. [getSubjectSummary]');
          else if (!degree || typeof degree !== typeof 'string') return Promise.reject('Error: Degree code is
required. [getSubjectSummary]');
          else if (!course || typeof course !== typeof 'string') return Promise.reject('Error: Course is
required. [getSubjectSummary]');

          const url = EHUrls.getSubject(subject, school, degree, course);
          return getDataFromWeb(url)
                    .then(data => parseSubjectSummary(data, school, degree));
};
```

Programa 36: Anexo D: Función del controlador de la asignatura.

Para obtener la *url* de la asignatura se hace gracias a una de las funciones de la clase EHUrls que con los códigos de asignatura, centro, grado y curso genera en enlace correcto.

```
class EHUrls {

          static getSubject(subject, school, degree, course){
                    if (!subject) throw new Error('Error: Subject code is required. [EHUrls.getSubject]');
                    else if (!school) throw new Error('Error: School code is required. [EHUrls.getSubject]');
                    else if (!degree) throw new Error('Error: Degree code is required. [EHUrls.getSubject]');
                    else if (!course) throw new Error('Error: Course code is required. [EHUrls.getSubject]');
                    else return `http://gestion-
servicios.ehu.es/pls/entrada/plew0040.htm_asignatura_next?p_sesion=&p_cod_idioma=CAS&p_en_portal=N&p_cod_centro=$
{school}&p_cod_plan=${degree}&p_anyoAcad=act&p_pestanya=3&p_menu=principal&p_cod_asig=${subject}&p_ciclo=X&p_curs
o=${course}&p_vengo_de=asig_cursos`;
}
}
```

Programa 37: Anexo D: Función para obtener la *url* de la asignatura.

Finalmente, la clase Subject llama a la función controladora, descrita anteriormente, para poder devolver al usuario los datos del resumen de la asignatura.

```
class Subject {
          constructor(subject, degree, course, school){
                    this._subject = subject;
                    this._school = school||Degree.getSchool(degree);
                    this._degree = degree;
                    this._course = course;
          }

          /**
           * Returns the summary of the subject
           */
          getSummary(){
                    return getSubjectSummary(this._subject, this._school, this._degree, this._course);
          }
}
```

Programa 38: Anexo D: Fragmento de la clase Subject.

D.2 Ejemplos de código de EHUServer

Dentro del servidor para el servicio *Firebase Function* se han organizado en diferentes bloques los tres tipos de funciones: *http*, *database* y *auth*. Cada

Dentro del bloque http se encuentran las responsables de la búsqueda en la base de datos y las de cargar la información de la web. A continuación se muestra un fragmento de la carga de datos:

```
const { University, Grade, Subject, Teacher } = require('ehu-scraping');
const db = require('../../db');
const database = new db();

//Variable global para agrupar todas las transacciones
let startMs;

/**
 * Función para cargar los datos de la universidad en la base de datos de firebaese
 * @param {*} req
 * @param {*} res
 * @param {*} next
 */
const loadData = async (req, res, next) => {
          //Datos para cargar solo el grado de ingenieria informática.
          const { campus, grade, school } = req.body;

          startMs = new Date().getTime();

          let gradeInfo = {};
          try{
                    gradeInfo = await loadGrade(grade, school, campus);
                    console.log('Grade loaded');
                    await loadSubjects(gradeInfo.courses, grade, school, campus);
                    console.log('Subjects loaded');
                    await loadTeachers(gradeInfo.teachers, grade, school, campus);
                    console.log('Teachers loaded');
                    console.log(`Tiempo necesario para cargar los datos del grado ${grade}: ${((new
Date().getTime() - startMs)) / 1000} segundos`);
                    res.send('OK!');
          }catch(e){
                    console.log(e);
                    return res.status(400).send('Error', err)
          }

};

module.exports = {
          loadData
};


/**
 * Función para cargar de forma secuencial todas las asignaturas del grado
 */
const loadSubjects = async (courses, grade, school, campus) => {
          //Espera un segundo para no saturar el servidor despues de haber cargado desde la página de grado.
          await wait(1500);
          const subjects = parseListOfSubjects(courses);
          for (const subject of subjects) {
                    //Cargar asignaturas de una en una, no todas de golpe
                    await loadSubject(subject, grade, school, campus);
                    //Esperar 1 segundo por cada asignatura para no sobrecargar
                    await wait(1000);
          }
};



/**
 * Carga en la base de datos la información referente a una asignatura.
 */
const loadSubject = async (subjectObj, grade, school, campus) => {

          const subjectRef = database.ref(`/ehu/subjects/${subjectObj.code}_${grade}/`);
          const subject = new Subject(subjectObj.code, grade, subjectObj.course, school);
          console.log('Cargando: ', grade, subjectObj.code, subjectObj.course, (startMs - new
Date().getTime())/1000);

          let subjectInfo = {};

          subjectInfo.summary = await subject.getSummary();
          await wait(1000);
          subjectInfo.detail = await subject.getDetail();
          await wait(1000);
          subjectInfo.schedule = await subject.getSchedule();
          await subjectRef.set(subjectInfo);
};
```

Programa 39: Anexo D: Fragmento para el volcado desde datos de la web.

Para realizar la búsqueda desde la aplicación de forma más efectiva se ha implementado una función responsable de esa tarea.

```
const db = require('../../db');
const database = new db();

const searchByName = async (req, res, next) => {
	try {
		let data;
		if (req.query.subject) {
			data = await filterBySubjectName(req.query.name);
		} else if (req.query.teacher) {
			data = await filterByTeacherName(req.query.name);
		} else {
			data = await filterByGradeName(req.query.name);
		}
		console.log('aa')
		return res.send(data);
	} catch (err) {
		console.log(err);
		return res.status(500).send(err);
	}
};


const filterByGradeName = async (name) => {
	const ref = database.ref('/ehu/searchs/grades/');
	const snap = await ref.once('value');
	let results = [];
	const snapObj = snap.val();
	Object.keys(snapObj).forEach(grade => {
		if(snapObj[grade].name.toLowerCase().includes(name)){
			results.push(Object.assign({}, snapObj[grade], {code: grade}));
		}
	});
	return results;
};
```

Programa 40: Anexo D: Función responsable de realizar la búsqueda de información en la base de datos.

En el siguiente bloque, *database*, se gestionan las funciones que se disparan sobre cualquier evento de la base de datos.

```
const onSubscribe = async (event, type) => {
	const { userID } = event.params;
	const subscription = event.data.val();
	console.log('subscription', subscription);
	let code;
	if (type === 'subject') {
		code = subscription.name;
	} else if (type === 'teacher') {
		code = Object.keys(subscription).find(a => a);
	}
	return database.ref('subscriptions').child(type).child(code).push(userID);
};
```

Programa 41: Anexo D: Función para manejar las subscripciones en las asignaturas.

El último bloque del servidor es el de las funciones que se ejecutan con las acciones de autenticación en la aplicación.

```
const db = require('../db');
const database = new db();

const onSignUp = async (event) => {
	const { uid, email, displayName, } = event.data;

	const ref = database.ref('users').child(uid);
	await ref.set({
		data: {
			email: email,
			name: displayName || '',
		},
		role: 'regular',
	});
};

const onDelete = async (event) => {
	const { uid } = event.data;
	const ref = database.ref('users').child(uid);
	await ref.remove();
};
```

Programa 42: Anexo D: Funciones para crear y borrar un perfil de la base de datos.

D.3 Ejemplos de código de EHUApp

El punto de partida de la aplicación es el fichero App.js.

```
import React, { Component } from 'react';
import { Provider } from 'react-redux';
import * as firebase from 'firebase';

import { firebase as config } from './app/config';
import store from './app/config/store';
import EHUApp from './app/config/routes';
import { Calendar } from './app/lib';

firebase.initializeApp(config.credentials);

const App = () => (
  <Provider store={store}>
    <EHUApp />
  </Provider>
);

export default App;
```

Programa 43: Anexo D: App.js de EHUApp.

El componente principal es el *router* [Capítulo 6.6.3], ahí es donde se gestionan todas las vistas. Para el uso de *Redux* se ha organizado la aplicación de forma que cada módulo tiene sus propias acciones y un *reducer*. A continuación se muestra el código para el *reducer* y las acciones de las asignaturas.

```
import { START_LOADING, SUBJECT_ERROR, SUBJECT_FETCH } from './subject.types';

const initialState = {
  subject: {},
  searching: true,
  error: false,
};

export const subjectReducer = (state = initialState, action) => {
  switch (action.type) {
    case START_LOADING:
      return {
        ...state,
        searching: true,
        error: false,
      };
    case SUBJECT_ERROR:
      return {
        ...state,
        error: true,
        searching: false,
      };
    case SUBJECT_FETCH:
      return {
        ...state,
        searching: false,
        error: false,
        subject: action.payload,
      };
    default:
      return state;
  }
};
```

Programa 44: Anexo D: Reducer del módulo Subject.

```
import { START_LOADING, SUBJECT_ERROR, SUBJECT_FETCH } from './subject.types';
import { getSubjectFromFirebase } from '../lib';

export const getSubject = subject => async (dispatch) => {
  try {
    dispatch({
      type: START_LOADING,
    });
    const result = await getSubjectFromFirebase(subject);
    return dispatch({
      type: SUBJECT_FETCH,
      payload: result,
    });
  } catch (err) {
    return dispatch({
      type: SUBJECT_ERROR,
    });
  }
};

export const startSearching = () => ({
  type: START_LOADING,
});
```

Programa 45: Anexo D: Acciones del módulo Subject.

```
export const START_LOADING = 'START_LOADING_SUBJECT';
export const START_LOADING_SUBJECT = 'START_LOADING_SUBJECT';
export const SUBJECT_FETCH = 'SUBJECT_FETCH';
export const SUBJECT_ERROR = 'SUBJECT_ERROR';
```

Programa 46: Anexo D: Constantes del modulo Subject. subject.types.js

Como Redux no permite tener varios reducers se deben juntar todos en un único objeto, eso se hace en el fichero index.reducer.js del directorio config.

```
import { combineReducers } from 'redux';
import { authReducer } from '../auth/auth.reducer';
import { searchReducer } from '../search/index';
import { teacherReducer } from '../teacher/teacher.reducer';
import { subjectReducer } from '../subject/subject.reducer';
import { gradeReducer } from '../grade/grade.reducer';
import { settingsReducer } from '../settings/settings.reducer';
import { userReducer } from '../user/user.reducer';
import { calendarReducer } from '../calendar';

export default combineReducers({
  auth: authReducer,
  search: searchReducer,
  teacher: teacherReducer,
  subject: subjectReducer,
  grade: gradeReducer,
  settings: settingsReducer,
  profile: userReducer,
  calendar: calendarReducer,
});
```

Programa 47: Anexo D: *Reducer* principal de la aplicación.

Este reducer se vincula al store principal de la aplicación en el fichero store.js del mismo directorio.

```
import { applyMiddleware, createStore } from 'redux';
import thunk from 'redux-thunk';
import promise from 'redux-promise-middleware';
import { createLogger } from 'redux-logger';
import reducers from './index.reducer';

const middleware = applyMiddleware(promise(), thunk, createLogger());
export default createStore(reducers, middleware);
```

Programa 48: Anexo D: Store de la aplicación EHUApp.

La traducción de la aplicación se ha hecho con la librería I18n de React. Para ello, en el directorio locale se crea un archivo index.js donde se inicia la configuración.

```
import I18n from 'react-native-i18n';
import { common } from '../config/';

import {
  es,
  en,
} from './languages';

I18n.fallbacks = true;
I18n.defaultLocale = common.defaultLocale;

I18n.translations = {
  es,
  en,
};

export default I18n;
```

Programa 49: Anexo D: Configuración de la traducción.

Esta configuración hace que el lenguaje por defecto sea el inglés y que en caso de que se esté utilizando otro idioma y uno de los textos tenga traducción se utilice la versión inglesa. Los textos se almacenan en un objeto JSON dentro del subdirectorio languages. A continuación se muestra un fragmento de la versión en castellano.

```
export const es = {
  auth: {
    resetTitle: 'Restaurar contraseña',
    resetSub: 'Enviar un correo para restaurar la contraseña',
    btnReset: 'Restaurar contraseña',
    resetPlaceholder: 'Email',
  },
  components: {
    emptyList: 'No hay elementos disponibles',
  },
  grade: {
    profile: {
      teachers: 'Profesores',
      subjects: 'Asignaturas',
      info: 'Información',
      emptyTeachersList: 'No hay profesores disponibles',
      emptySubjectsList: 'No hay asignaturas disponibles',
      summary: 'Resumen',
      contact: 'Contacto',
      openModal: 'Abrir resumen',
      cardTitle: 'Resumen',
      closeBtn: 'Cerrar resumen',
    },
  },
  teacher: {
    profile: {
      noTutorships: 'No hay tutorias para este día',
      noMore: 'No hay más tutorias',
      nextTutorship: 'Siguiente tutoría',
    },
  },
  settings: {
    languageTitle: 'Cambia tu idioma',
  },
  user: {
    btnPassword: 'Cambia tu contraseña',
    btnLogout: 'Salir',
    btnSubs: 'Ver subscripciones',
    btnEdit: 'Editar perfil',
    logoutAlert: '¿ Estás seguro de los cambios ?',
    password: {
      current: 'Contraseña actual',
      new: 'Nueva contraseña',
      repeat: 'Repite la contraseña',
      btnConfirm: 'Confirmar',
      alertSamePassword: 'Las contraseñas deben coincidir',
      alertShortPassword: 'La contraseña es demasiado corta',
      alertWrongPassword: 'La contraseña vieja es erronea',
      alertTitle: 'Opps, error al actualizar',
      title: 'Modifica tu contraseña',
    },
  },
  admin: {
  },
  calendar: {
    empty: 'El horario no está disponibles.',
    holiday: 'Vacaciones',
  },
};
```

Programa 50: Anexo D: Traducción en castellano.

Para que la traducción sea visible en los textos de la aplicación se han creado una serie de funciones que se encargan de esta tarea.

```
import { AsyncStorage } from 'react-native';
import I18n from '../locale';
import { common } from '../config';

export class Translate {
  static t = key => I18n.t(key, I18n.locale);

  static getLocale = () => {
    const locale = (I18n.locale && I18n.locale.toLowerCase()) || common.defaultLocale;
    return locale;
  };
  static getCurrentLocale = async () => {
    const deviceLocale = Translate.getLocale();
    const storageLocale = await AsyncStorage.getItem('locale');
    return storageLocale || deviceLocale;
  };

  static configureLocale = (locale) => {
    I18n.locale = locale;
  };

  static saveLocale = async (locale) => {
    await AsyncStorage.setItem('locale', locale);
    return true;
  };
}
```

Programa 51: Anexo D: Clase Translate de EHUApp.

Para traducir basta con llamar a la función t, este es el nombre que utiliza el método por defecto de la librería, dentro de cualquier componente.

```
<View>
        <CategoryDivider iconName="contacts" title={Translate.t('grade.profile.contact')} />
        <ContactView contact={data.contact} />
</View>
```

Programa 52: Anexo D: Traducción de un fragmento de la interfaz.

La interacción con los servicios de Firebase no se hace directamente desde los métodos de cada componente, para ello se ha creado un fichero firebase.js en el directorio lib donde se han definido todas las funciones necesarias para trabajar con Firebase.

```
export const addSubscriptionOnFirebase = async (path, data) => {
  if (firebase.auth().currentUser === null) throw new Error('You must be logged in');
  const { uid } = firebase.auth().currentUser;
  const ref = firebase.database().ref('users').child(uid).child(path);
  return ref.update(data);
};

export const deleteSubscriptionOnFirebase = async (path, key) => {
  if (firebase.auth().currentUser === null) throw new Error('You must be logged in');
  const { uid } = firebase.auth().currentUser;
  const ref = firebase.database().ref('users').child(uid).child(path)
    .child(key);
  return ref.remove();
};
```

Programa 53: Anexo D: Fragmento de código del fichero firebase.js.

A continuación una de las acciones que hace uso de las funciones anteriores.

```
export const addNewSub = (subsType, data) => async (dispatch) => {
  try {
    dispatch({
      type: START_NEW_SUBSCRIPTION,
    });
    let actionType;
    if (subsType === 'subjects') actionType = SUCCESS_NEW_SUBSCRIPTION_SUBJECTS;
    else if (subsType === 'teachers') actionType = SUCCESS_NEW_SUBSCRIPTION_TEACHERS;
    else {
      return dispatch({
        type: ERROR_NEW_SUBSCRIPTION,
        payload: 'Invalid subscription type',
      });
    }
    await addSubscriptionOnFirebase(subsType, data);

    return dispatch({
      type: actionType,
      payload: data,
    });
  } catch (error) {
    return dispatch({
      type: ERROR_NEW_SUBSCRIPTION,
      payload: error,
    });
  }
};
```

Programa 54: Anexo D: Fragmento de código de las acciones del módulo user.

Por último, los componentes. Todos los módulos están formados por componentes basados en clases o los funcionales. Los primeros son los que tienen toda la lógica, en el siguiente fragmento se puede ver el de la ficha de profesor.

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { TeacherProfileScreen } from './screens';
import { getTeacher } from '.';

import { addNewSub, deleteSubscription } from '../user';

class TeacherProfileContainer extends Component {
  state = {
    following: false,
    teacherCode: '',
  };

  componentWillMount = () => {
    const { params } = this.props.navigation.state;
    const { getTeacherAction } = this.props;
    const teacherCode = `${params.params.code}_${params.params.grade}`;
    this.setState({ teacherCode });
    getTeacherAction(params.params);
  }

  componentWillReceiveProps = (nextProps) => {
    const { userTeachers = {} } = nextProps;
    const { following, teacherCode } = this.state;

    this.setState({ following: Object.keys(userTeachers).includes(teacherCode) });
  }

  handleToggleSubscription = async () => {
    const {
      addNewSubAction,
      deleteSubscriptionAction,
      teacher,
      getTeacherAction,
    } = this.props;
    const { params } = this.props.navigation.state;
    const { following, teacherCode } = this.state;
    const tmpSub = {};
    if (!following) {
      tmpSub[teacherCode] = teacher.name;
      await addNewSubAction('teachers', tmpSub);
    } else {
      await deleteSubscriptionAction('teachers', teacherCode);
    }
  }

  render() {
    const { following } = this.state;
    return (<TeacherProfileScreen
      searching={this.props.searching}
      data={this.props.teacher}
      error={this.props.error}
      handleToggleSubscription={this.handleToggleSubscription}
      following={following}
    />);
  }
}

const mapStateToProps = (state, action) => ({
  searching: state.teacher.searching,
  teacher: state.teacher.teacherData,
  userTeachers: state.profile.teachers,
  error: state.teacher.error,
});

const mapDispatchToProps = dispatch => ({
  getTeacherAction: params => dispatch(getTeacher(params)),
  addNewSubAction: (type, data) => dispatch(addNewSub(type, data)),
  deleteSubscriptionAction: (type, key) => dispatch(deleteSubscription(type, key)),
});

export const TeacherProfile = connect(mapStateToProps, mapDispatchToProps)(TeacherProfileContainer);
```

Programa 55: Anexo D: Componente para la ficha de profesor.

Como se puede ver en la función *render*, no se implementa aquí la interfaz, eso se hace en los componentes funcionales que se encuentran dentro de la carpeta *screens* de cada módulo. A continuación, se muestra el fragmento correspondiente a la vista de la ficha de profesor.

```
import React from 'react';
import { View, StyleSheet, SafeAreaView, ActivityIndicator, FlatList } from 'react-native';
import { Icon, Text, Avatar, List, ListItem, Button } from 'react-native-elements';
import { Agenda } from 'react-native-calendars';

import { EmptyList } from '../../components';
import { colors } from '../../config';
import { sortByDate, Translate, Dates } from '../../lib';

const styles = StyleSheet.create({
  …
});

export const TeacherProfileScreen = ({
  searching,
  data = {},
  error,
  handleToggleSubscription,
  following,
}) => {
  const parsedSchedule = (data && data.schedule) ? Dates.scheduleToCalendar(data.schedule) : {};
  return (
    <SafeAreaView style={styles.safe}>
      <View style={styles.container}>
        {
          searching &&
          <View>
            <ActivityIndicator size="large" color={colors.black} />
          </View>
        }
        {
          !searching
          && error
          && !data
          && (
            <View>
              <Text> Error </Text>
            </View>)
        }
        {
          !searching
          && !error
          && data
          && <TeacherView
            data={data}
            handleToggleSubscription={handleToggleSubscription}
            following={following}
          />
        }
      </View>
      <Agenda
        items={parsedSchedule}
        renderItem={(item, first) => <CalendarRow item={item} />}
        renderEmptyDate={() => <View />}
        rowHasChanged={(r1, r2) => r1.start !== r2.start}
        renderEmptyData={() => <EmptyData />}
      />

    </SafeAreaView>
  );
};
```

Programa 56: Anexo D: Vista de la ficha de profesor.

Anexo E: Base de datos

Como se ha descrito en el capítulo 5.3 la base de datos del proyecto se basa en nodos dentro de un único fichero JSON. A continuación se muestran algunos de los datos almacenados en el documento de la base de datos.


```
{
"NTH7ulsRoSgOP4hootWaX54znXo1" : {
    "data" : {
      "birthdate" : "1900-01-01",
      "displayName" : "Administrador",
      "gender" : "other"
    },
    "role" : "admin",
    "subjects" : {
      "25987_GINFOR20" : {
        "group" : "01",
        "name" : "Gestión de Proyectos"
      }
    }
  },
  "PrPtoCd4ERbJKFG5BtKODANOM1k1" : {
    "data" : {
      "birthdate" : "1932-04-30",
      "displayName" : "Manolo",
      "email" : "test@test.es",
      "gender" : "female",
      "name" : ""
    },
    "role" : "regular"
  },
  "lh931TLxQrWERQ2hrN0Ow1zqGyv2" : {
    "data" : {
      "birthdate" : "1900-01-01",
      "displayName" : "Joseba",
      "email" : "joseba@test.es",
      "gender" : "male",
      "name" : ""
    },
    "role" : "admin",
    "subjects" : {
      "25987_GINFOR20" : {
        "group" : "31",
        "name" : "Gestión de Proyectos"
      },
      "26009_GINFOR20" : {
        "group" : "31",
        "name" : "Análisis Matemático"
      },
      "26022_GINFOR20" : {
        "group" : "01",
        "name" : "Introducción a los Sistemas Operativos"
      },
      "26026_GINFOR20" : {
        "group" : "01",
        "name" : "Diseño de Bases de Datos"
      },
      "26030_GINFOR20" : {
        "group" : "16",
        "name" : "Administración de Bases de Datos"
      },
      "26231_GINFOR20" : {
        "group" : "01",
        "name" : "Herramientas Avanzadas de Desarrollo de Software"
      },
      "26239_GINFOR20" : {
        "group" : "01",
        "name" : "Calidad del Software"
      },
    },
    "teachers" : {
      "1496_GINFOR20" : "GARCIA GOMEZ, JOSE MARIA",
      "1671_GINFOR20" : "GARCIA GOMEZ, OSCAR",
      "1740_GINFOR20" : "GARCIA GOMEZ, TOMAS ANTONIO",
      "1814_GINFOR20" : "GARCIA GOMEZ, JESUS",
      "1815_GINFOR20" : "GARCIA GOMEZ, MARIA LUISA",
      "1823_GINFOR20" : "GARCIA GOMEZ, ANA ROSA",
      "1997_GINFOR20" : "GARCIA GOMEZ, MARIA ISABEL",
      "288194_GINFOR20" : "GARCIA GOMEZ, RODRIGO",
      "317643_GINFOR20" : "GARCIA GOMEZ, JOSU",
      "3711_GINFOR20" : "GARCIA GOMEZ, JOSE ANGEL",
      "5307_GINFOR20" : "GARCIA GOMEZ, BLANCA ROSA"
    }
  }
}
```


Programa 57: Anexo E: Nodo *users* de la base de datos.

Dentro del nodo *ehu*, hay cinco subnodos que debido al gran cantidad de datos que se almacenan en estos solo se muestra un fragmento de cada uno de ellos.


```
{
  "GINFOR20" : {
    "holidays" : {

      "-L9uomk6miWtD2Ahozv2" : {
        "dateString" : "2018-03-19",
        "day" : 19,
        "month" : 3,
        "timestamp" : 1521417600000,
        "weekStart" : 23,
        "year" : 2018
      },
      "-L9uomu0BFfp-MvJs05l" : {
        "dateString" : "2018-03-26",
        "day" : 26,
        "month" : 3,
        "timestamp" : 1522022400000,
        "weekStart" : 24,
        "year" : 2018
      },
      "-L9uonjK8L6jvRHgWenL" : {
        "dateString" : "2018-04-09",
        "day" : 9,
        "month" : 4,
        "timestamp" : 1523232000000,
        "weekStart" : 25,
        "year" : 2018
      },
      "-L9uonsattFaUVOW9CI-" : {
        "dateString" : "2018-04-16",
        "day" : 16,
        "month" : 4,
        "timestamp" : 1523836800000,
        "weekStart" : 26,
        "year" : 2018
      },
      "-L9uoo1b9WRXSQp1WSEn" : {
        "dateString" : "2018-04-23",
        "day" : 23,
        "month" : 4,
        "timestamp" : 1524441600000,
        "weekStart" : 27,
        "year" : 2018
      },
      "-L9uooBdfRtEdvTwMGHC" : {
        "dateString" : "2018-04-30",
        "day" : 30,
        "month" : 4,
        "timestamp" : 1525046400000,
        "weekStart" : 28,
        "year" : 2018
      },
      "-L9uopaU2PE_rPxfoocT" : {
        "dateString" : "2018-05-07",
        "day" : 7,
        "month" : 5,
        "timestamp" : 1525651200000,
        "weekStart" : 29,
        "year" : 2018
      },
    }
  },
  "ehu" : {
    "holidays" : {
      "-L9lzGyEvIUIwXHa32rM" : {
        "dateString" : "2018-03-19",
        "day" : 26,
        "month" : 4,
        "name" : "San José",
        "timestamp" : 1524700800000,
        "year" : 2018
      }
    },
    "weekStart" : {
      "-L9lzJDKPYbDRAN37EKd" : {
        "dateString" : "2018-04-27",
        "day" : 27,
        "month" : 4,
        "timestamp" : 1524787200000,
        "weekStart" : 2,
        "year" : 2018
      }
    }
  }
}
```


Programa 58: Anexo E: Información del nodo calendars.

En el siguiente fragmento se muestra parte de la información almacenada en un grado de la UPV-EHU, en este caso, el grado en ingeniería informática.


```
{
"data" : {
    "contact" : {
      "address" : "",
      "email" : " correo@ehu.eus",
      "name" : " NEREA GARCIA GOMEZ",
      "phone" : " 943018029"
    },
    "href" : "http://gestion-
servicios.ehu.es/pls/entrada/plew0040.htm_siguiente?p_sesion=&p_cod_idioma=CAS&p_en_portal=N&p_anyoAcad=act&p_cod
_centro=226&p_cod_plan=GINFOR20&p_menu=intro",
    "minimum-grade" : "9,83",
    "name" : "Grado en Ingeniería Informática - 2017/18",
    "summary" : "El Grado en Ingeniería Informática te ofrece un inmenso campo para el estudio, la investigación
y la innovación. Las Nuevas Tecnologías de la Información y la Comunicación están hoy en día presentes en todos
los ámbitos de nuestra vida cotidiana. En este grado podrás profundizar en áreas tan apasionantes como la
inteligencia artificial, la robótica, el procesamiento del lenguaje natural, el procesamiento digital de imagen y
sonido, las comunicaciones multimedia, etc. Todo esto lo podrás averiguar en el Grado de Ingeniería Informática.
Y además podrás especializarte en Computación, Ingeniería de Computadores o Ingeniería del Software. También, te
formarás para trabajar de manera autónoma o integrada en equipos multidisciplinares."
  },
  "subjects" : {
    "25039" : "Norma y Uso de la Lengua Vasca",
    "25972" : "Álgebra",
    "25973" : "Métodos Estadísticos de la Ingeniería",
    "25987" : "Gestión de Proyectos",
    "25989" : "Economía y Administración de Empresas",
    "26009" : "Análisis Matemático",
    "26011" : "Matemática Discreta",
    "26012" : "Fundamentos de Tecnología de Computadores",
    "26013" : "Metodología de la Programación",
    "26014" : "Principios de Diseño de Sistemas Digitales",
    "26015" : "Estructura de Computadores",
    "26016" : "Estructuras de Datos y Algoritmos",
    "26017" : "Ingeniería del Software",
    "26018" : "Arquitectura de Computadores",
    "26019" : "Introducción a las Redes de Computadores",
    "26020" : "Bases de Datos",
    "26021" : "Lenguajes, Computación y Sistemas Inteligentes",
    "26022" : "Introducción a los Sistemas Operativos",
    "26023" : "Investigación Operativa",
    "26025" : "Sistemas de Gestión de Seguridad de Sistemas de Información",
    "26026" : "Diseño de Bases de Datos",
    "26029" : "Sistemas Web",
    "26030" : "Administración de Bases de Datos",
    "26031" : "Programación Básica",
    "26209" : "Programación Modular y Orientación a Objetos",
    "26210" : "Servicios y Aplicaciones en Red",
    "26211" : "Compilación",
    "26212" : "Diseño de Algoritmos",
    "26213" : "Modelos Abstractos de Cómputo",
    "26214" : "Inteligencia Artificial",
    "26215" : "Ingeniería del Software II",
    "26216" : "Computación Científica",
    "26217" : "Gráficos por Computador",
    "26239" : "Calidad del Software",
    "26240" : "Desarrollo Industrial del Software",
    "26241" : "Gestión Avanzada de Información",
    "26244" : "Comunicaciones Móviles y Multimedia",
    "26245" : "Diseño y Proyectos de Redes",
    "26248" : "Diseño y Construcción de Sistemas Digitales",
    "26249" : "Procesadores de Alto Rendimiento",
    "26250" : "Sistemas Operativos",
    "26251" : "Diseño de Sistemas Empotrados",
    "26252" : "Sistemas de Cómputo Paralelo",
    "26253" : "Evaluación del Rendimiento de Sistemas Informáticos",
    "26254" : "Ingeniería de Control",
    "26256" : "Sistemas Distribuidos",
    "26257" : "Seguridad, Rendimiento y Disponibilidad de Servicios e Infraestructuras",
    "26258" : "Electrónica Aplicada al Tratamiento de Datos",
    "26259" : "Diseño de Sistemas Operativos y Tiempo Real **",
    "26260" : "Procesado Digital de Sonido e Imagen",
    "26261" : "Robótica, Sensores y Actuadores",
    "27812" : "Cálculo",
    "27813" : "Programación Concurrente",
    "27814" : "Visualización y Entornos Virtuales",
    "27815" : "Interfaces Inteligentes y Accesibles",
    "27816" : "Trabajo Fin de Grado"
  },
  "teachers" : {
    "1496" : "GARCIA GOMEZ, JOSE MARIA",
    "1570" : "GARCIA GOMEZ, FRANCISCA",
    "1671" : "GARCIA GOMEZ, OSCAR",
    "1740" : "GARCIA GOMEZ, TOMAS ANTONIO",
    "1814" : "GARCIA GOMEZ, JESUS",
    "1815" : "GARCIA GOMEZ, MARIA LUISA"

  }
}
```


Programa 59: Anexo E: Información del nodo ehu/degrees/GI/226/GINFOR20

A continuación, un trozo del contenido almacenado sobre la asignatura Gestión Avanzada de la Información del Grado en Ingeniería Informática.


```
{
"detail" : {
    "bibliography" : [ "XML in a nutshell. E.R. Harold eta W.S Means. 2004, O'Reilly.", "XSLT & XPATH. A guide to
transformations. J.R. Gardner eta Z.L. Rendon. 2002, Prentice-Hall.", ],
    "competences" : "A continuación se esbozan los temas por tratar. Intercambio de documentos entre
organizaciones: el mundo XML y estándares de descripción...",
    "content" : "1. Estándares y tecnologías XML para intercambio de documentos (XPath, XML Schema, Schematron,
XQuery, xSQL)...",
    "course" : "2017/18",
    "description" : "“Gestión Avanzada de la Información” (GAI) es una asignatura obligatoria dentro de la
Especialidad de Ingeniería del Software. Esta asignatura descansa sobre aspectos impartidos en  Sistemas Web y
Bases de Datos. ...",
    "extraordinary_announcement" : "- Examen final: 85% El examen consistirá en una parte de conceptos básicos y
ejercicios escritos de carácter práctico. - Trabajo práctico: 15%. Para presentarse al examen final de la
evaluación de conjunto (en la convocatoria ordinaria o extraordinaria), se deberá haber entregado el trabajo del
proyecto XML. ...",
    "grade" : "Grado en Ingeniería Informática",
    "name" : "Gestión Avanzada de Información",
    "observation" : "No hay información disponible sobre esta sección",
    "ordinary_announcement" : "La asignatura tiene dos modos de evaluación: la evaluación final (o de conjunto) y
la evaluación continua.EVALUACIÓN CONTINUALa evaluación continua, a la que el alumnado podrá acogerse
voluntariamente, ... ",
    "school" : "Facultad de Informática",
    "year" : "Indiferente"
  },
  "schedule" : {
    "groups" : [ {
      "code" : "01",
      "schedule" : [ {
        "monday" : [ {
          "hours" : "09:00-10:30",
          "type" : "GL"
        }, {
          "hours" : "10:45-12:15",
          "type" : "GL"
        } ],
        "tuesday" : [ {
          "hours" : "10:45-12:15",
          "type" : "M"
        } ],
        "wednesday" : [ {
          "hours" : "12:30-14:00",
          "type" : "M"
        } ],
        "weeks" : "16-30"
      } ],
      "teachers" : [ {
        "code_area" : "0570",
        "code_grade" : "GINFOR20",
        "code_school" : "226",
        "dep_teacher" : "141",
        "id_teacher" : "1671",
        "name" : "GARCIA GOMEZ, OSCAR"
      } ]
    },
 ]
  },
  "summary" : {
    "code" : "26241",
    "course" : "2017/18",
    "credits" : "6",
    "departament" : "\n\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t",
    "grade" : {
      "code" : "GINFOR20",
      "name" : "Grado en Ingeniería Informática"
    },
    "href" : "http://gestion-
servicios.ehu.es/pls/entrada/plew0040.htm_asignatura_next?p_sesion=&p_cod_idioma=CAS&p_en_portal=N&p_cod_centro=2
26&p_cod_plan=GINFOR20&p_anyoAcad=act&p_pestanya=3&p_menu=principal&p_cod_asig=26241&p_ciclo=X&p_curso=2017/18&p_
vengo_de=asig_cursos",
    "name" : "Gestión Avanzada de Información",
    "school" : {
      "code" : "226",
      "name" : "Facultad de Informática"
    },
    "teachers" : [ {
      "code_area" : "0570",
      "code_grade" : "GINFOR20",
      "code_school" : "226",
      "dep_teacher" : "141",
      "id_teacher" : "1671",
      "name" : "GARCIA GOMEZ, OSCAR"
    },
],
    "year" : "Indiferente"
  }
}
```


Programa 60: Anexo E: Información del nodo ehu/subjects/26241_GINFOR20

El siguiente fragmento muestra la información de un/a profesor/a del departamento de Lenguajes y Sistemas Informáticos.


```
{
  "area" : "Lenguajes y Sistemas InformÃ¡ticos",
  "category" : "Profesorado Titular De Universidad (Doctora)",
  "departament" : "Lenguajes y Sistemas InformÃ¡ticos",
  "email" : "m.gomez@ehu.eus",
  "href" : "https://www.ehu.eus/es/web/estudiosdegrado-gradukoikasketak/grado-ingenieria-informatica-
profesorado?p_redirect=consultaTutorias&p_anyo_acad=20170&p_idp=3805",
  "name" : "GOMEZ GARCIA, MARIANO",
  "schedule" : [ {
    "date-end" : "2017-09-12T13:00",
    "date-start" : "2017-09-12T11:00",
    "place" : "Facultad De InformÃ¡tica Despacho Del Profesor O Profesora"
  }, {
    "date-end" : "2017-09-13T13:1515:30",
    "date-start" : "2017-09-13T12:15",
    "place" : "12:15 - 13:15Facultad De InformÃ¡tica Despacho Del Profesor O Profesora15:30 - 16:30Facultad De
InformÃ¡tica Despacho Del Profesor O Profesora"
  },  ]
}
```


Programa 61: Anexo E: Información del nodo ehu/teachers/3805_GINFOR20

Por último, el nodo *searchs* donde se encuentra la información reducida para agilizar las búsquedas dentro de la aplicación.


```
{
  "grades" : {
    "GINFOR20" : {
      "campus" : "GI",
      "name" : "Grado en Ingeniería Informática - 2017/18",
      "school" : {
        "code" : "226",
        "name" : "Facultad de Informática"
      }
    }
  },
  "subjects" : {
    "25039_GINFOR20" : "Norma y Uso de la Lengua Vasca",
    "25972_GINFOR20" : "Álgebra",
    "25973_GINFOR20" : "Métodos Estadísticos de la Ingeniería",
    "25987_GINFOR20" : "Gestión de Proyectos",
    "25989_GINFOR20" : "Economía y Administración de Empresas",
    "26009_GINFOR20" : "Análisis Matemático",
    "26011_GINFOR20" : "Matemática Discreta",
    "26012_GINFOR20" : "Fundamentos de Tecnología de Computadores",
    "26013_GINFOR20" : "Metodología de la Programación",
    "26014_GINFOR20" : "Principios de Diseño de Sistemas Digitales",
    "26015_GINFOR20" : "Estructura de Computadores",
    "26016_GINFOR20" : "Estructuras de Datos y Algoritmos",
    "26017_GINFOR20" : "Ingeniería del Software",
    "26018_GINFOR20" : "Arquitectura de Computadores",
    "26019_GINFOR20" : "Introducción a las Redes de Computadores",
    "26020_GINFOR20" : "Bases de Datos",
    "26021_GINFOR20" : "Lenguajes, Computación y Sistemas Inteligentes",
    "26022_GINFOR20" : "Introducción a los Sistemas Operativos",
    "26023_GINFOR20" : "Investigación Operativa",
    "26025_GINFOR20" : "Sistemas de Gestión de Seguridad de Sistemas de Información",
    "26221_GINFOR20" : "Robótica y Control Inteligente",
    "26222_GINFOR20" : "Heurísticos de Búsqueda",
    "26223_GINFOR20" : "Aprendizaje Automático y Redes Neuronales",
    "26224_GINFOR20" : "Visión por Computador",
    "26225_GINFOR20" : "Técnicas Avanzadas del Inteligencia Artificial",
    "26257_GINFOR20" : "Seguridad, Rendimiento y Disponibilidad de Servicios e Infraestructuras",
    "26258_GINFOR20" : "Electrónica Aplicada al Tratamiento de Datos",
    "26259_GINFOR20" : "Diseño de Sistemas Operativos y Tiempo Real **",
    "26260_GINFOR20" : "Procesado Digital de Sonido e Imagen",
    "26261_GINFOR20" : "Robótica, Sensores y Actuadores",
    "27812_GINFOR20" : "Cálculo",
    "27813_GINFOR20" : "Programación Concurrente",
    "27814_GINFOR20" : "Visualización y Entornos Virtuales",
    "27815_GINFOR20" : "Interfaces Inteligentes y Accesibles",
    "27816_GINFOR20" : "Trabajo Fin de Grado"
  },
  "teachers" : {
    "1496_GINFOR20" : "GARCIA GOMEZ, JOSE MARIA",
    "1570_GINFOR20" : "LUCIO CARRASCO, FRANCISCA",
    "1671_GINFOR20" : "GARCIA GOMEZ, OSCAR",
    "1740_GINFOR20" : "GARCIA GOMEZ, TOMAS ANTONIO",
    "1814_GINFOR20" : "GARCIA GOMEZ, JESUS",
    "1815_GINFOR20" : "GARCIA GOMEZ, MARIA LUISA",
    "1823_GINFOR20" : "GARCIA GOMEZ, ANA ROSA",
    "1879_GINFOR20" : "GARCIA GOMEZ, JULIO",
    "1997_GINFOR20" : "GARCIA GOMEZ, MARIA ISABEL",
    "2235_GINFOR20" : "GARCIA GOMEZ, LUIS MARIA",
    "25805_GINFOR20" : "GARCIA GOMEZ, IBAI",
    "262818_GINFOR20" : "GARCIA GOMEZ, BORJA"
  }
}
```


Programa 62: Anexo E: Información del nodo ehu/searchs

Anexo F: Firebase

A continuación se muestran capturas con el estado de los servicios de Firebase para está aplicación. Se muestra la gestión de los usuarios, el uso de la base de datos o los *logs* de las funciones.

Servicio Firebase Auth con los usuarios registrado durante el desarrollo de la aplicación. La siguiente figura muestra la interfaz web para la gestión de los usuarios.

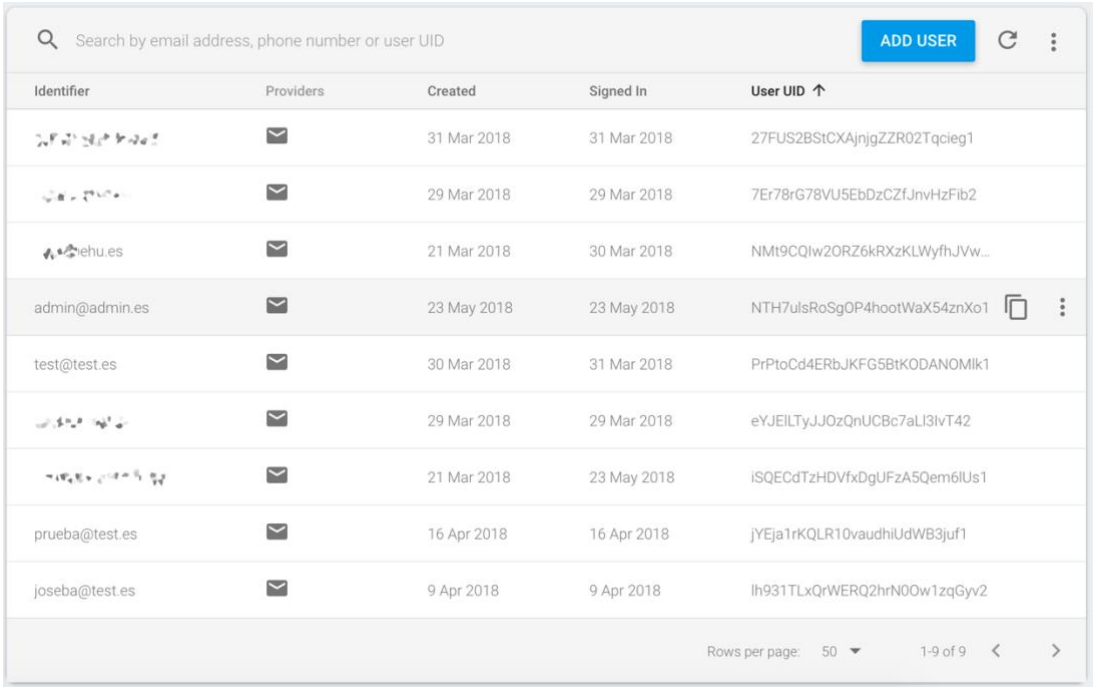


Figura 33: Anexo F: Vista de la interfaz del servicio FIrebase Auth.

En la siguiente imagen se pueden ver las funciones que hay en ejecuciones en Firebase y el uso de estas en las últimas horas.

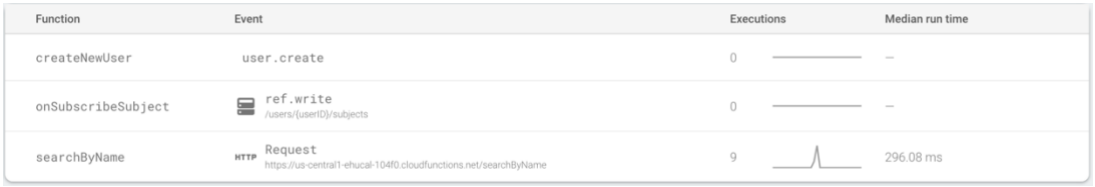


Figura 34: Anexo F: Vista de las funciones en ejecución en Firebase.

Por último, los *logs* de las ejecuciones de las funciones que se muestran a través de la consola de Firebase disponible en el sitio web.

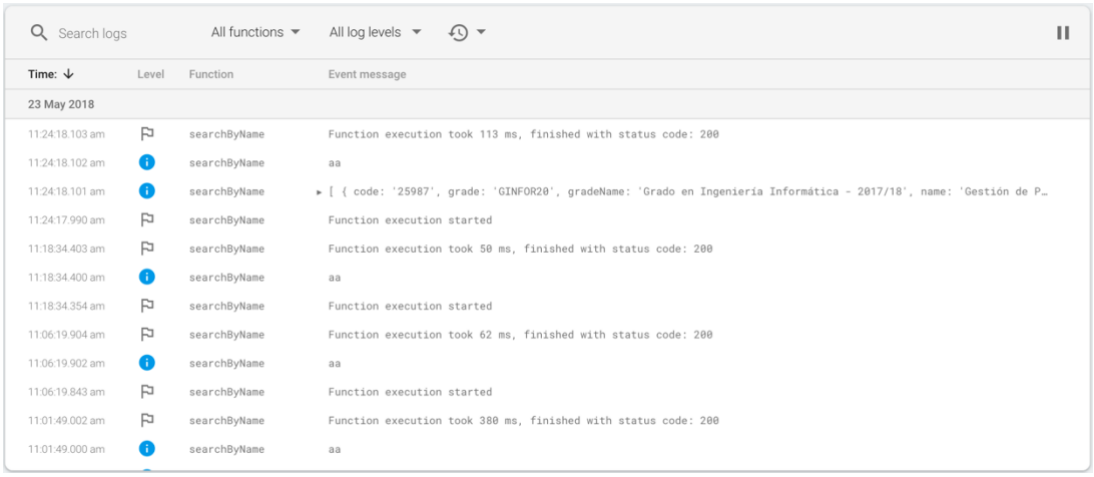


Figura 35: Anexo F: Vista en la consola de Firebase de los logs de las funciones ejecutadas.

Anexo G: TravisCI

TravisCI es la herramienta de integración continua que se ha utilizado en todas las implementaciones del proyecto. En este apartado se muestra la interfaz web donde se ofrecen los *logs* de todas las ejecuciones.

En esta primera captura se puede ver como el módulo EHUApp pasa las pruebas especificadas en el fichero *.travis.yml* y por lo tanto es válido.

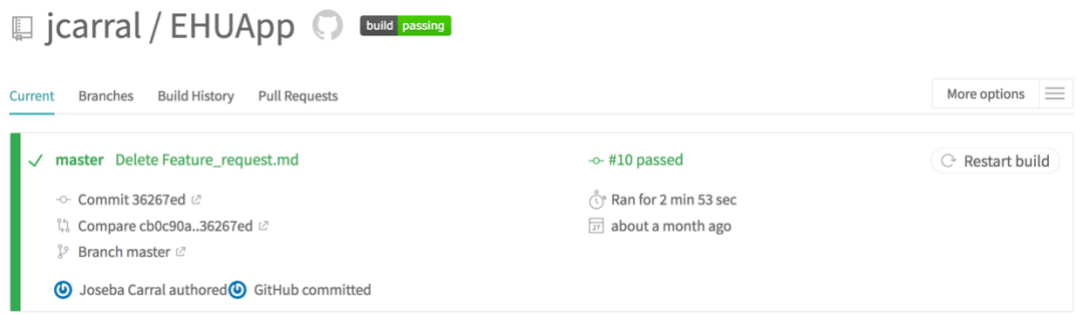


Figura 36: Anexo G: Última versión ejecutada en TravisCI.

Aunque la última versión es válida no todas las versiones que ha tenido la aplicación a lo largo del proyecto lo han sido. TravisCI comprueba cada versión de la rama subida a Github.

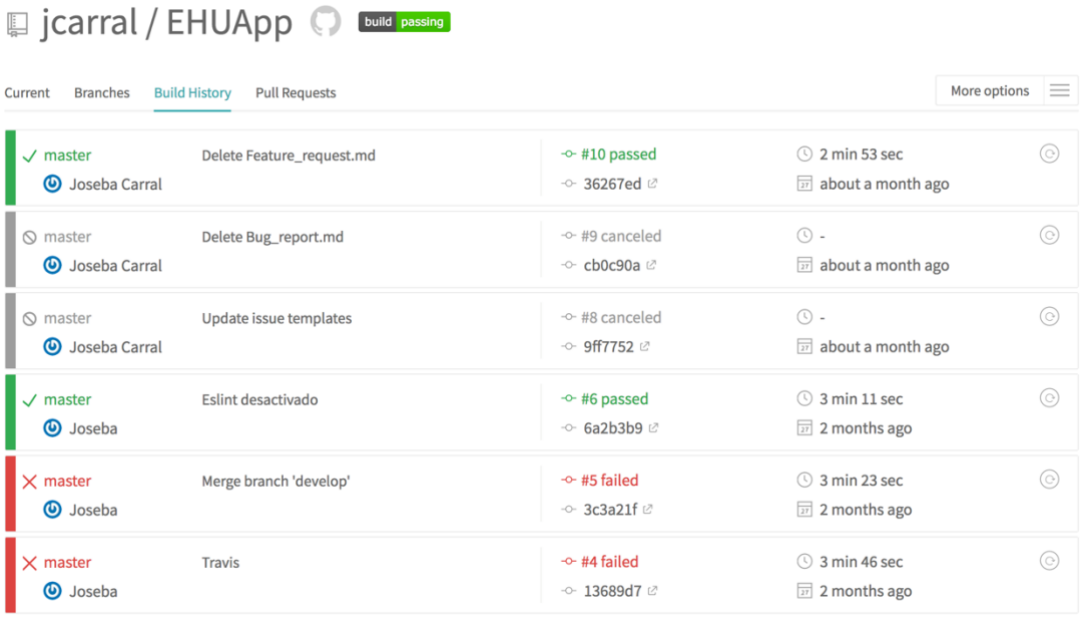


Figura 37: Anexo G: Historial de versiones ejecutadas en TravisCI.

Por último, también verifica si los *pull requests* son correctos o no. Además de mostrarlo en la página de Github también se puede comprobar a través de su cliente web. En la siguiente imagen se puede ver como el *pull request* realizado por *@olatzromeo* falla y por qué.

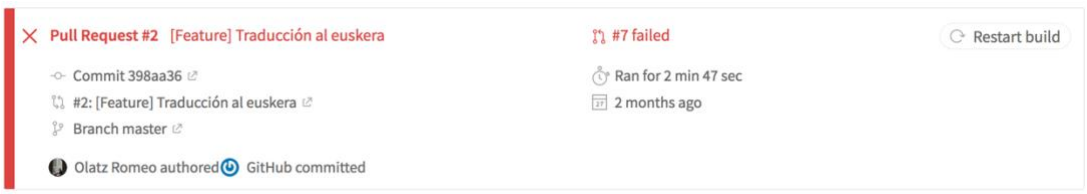


Figura 38: Anexo G: Error en el *pull request* realizado por olatzromeo.

Figura 39: Anexo G: *Logs* del error del *pull request*.