Grado en Ingeniería Informática

Trabajo Final de Grado

Aplicación móvil con React Native sobre noticias y puntos de interés en la provincia de Castellón

Autor:
Adrián Enríquez Ballester

Supervisor:
Sergio Aguado González
Tutor académico:
Reyes Grangel Seguer

Fecha de lectura: 21 de Marzo de 2018
Curso académico 2017/2018

Resumen

Este documento describe la gestión y el desarrollo del Trabajo de Fin de Grado de Ingeniería del *Software* realizado por su autor durante su estancia en prácticas en la empresa Cuatroochenta. Nomepierdoniuna, una *webzine* independiente de Castellón de la Plana, pretende renovar su aplicación móvil. Esta se ofrece únicamente para el sistema iOS y ha quedado desfasada con el paso de los años. El proyecto se ha llevado a cabo en dicha estancia y su objetivo ha sido el desarrollo de una aplicación móvil con el *framework* React Native, con la finalidad de ofrecer un producto más actual para Android e iOS. La nueva aplicación mantiene la funcionalidad de la anterior, orientada a mostrar el contenido publicado en Nomepierdoniuna, además de ampliaciones en cuanto al contenido que se muestra y nuevas funcionalidades para las secciones en las que se ha observado más tráfico en su correspondiente versión web. Su desarrollo ha estado caracterizado por el uso de tecnologías recientes, como lo es React Native, y metodologías ágiles como Scrum.

Palabras clave

Aplicaciones móviles, React Native, Android, iOS, Scrum, *webzine*

Keywords

Mobile app, React Native, Android, iOS, Scrum, webzine

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El proyecto expuesto en este documento trata sobre el desarrollo de la aplicación móvil de Nomepierdoniuna, una *webzine* que ofrece información práctica sobre la escena de música y espectáculos en Castellón y comarca. Ha sido llevado a cabo en la empresa Cuatroochenta, como proyecto de una estancia en prácticas.

A continuación, para introducir el contexto en el que se ha realizado, se describen dicha *webzine* y dicha empresa. La sección finaliza con el establecimiento del alcance del proyecto.

1.1.1. Nomepierdoniuna

Nomepierdoniuna es una *webzine* independiente que ofrece información de lo que ocurre en Castellón y La Plana a nivel musical y cultural [19]. Es un sitio de referencia y punto de encuentro reconocido por público, artistas, programadores e instituciones.

En cuanto a su audiencia y difusión, según Google Analytics posee cerca de 20.000 usuarios. Su web tiene un tráfico de 40.000 visitas al mes. También publican contenido en YouTube, en el que acumulan 400.000 visitas y 280 subscriptores, y en las redes sociales, siendo en Facebook con 10.000 seguidores dónde más presencia tienen.

El perfil genérico de usuario de Nomepierdoniuna, es el de residente en la provincia de Castellón, de entre 25 y 34 años, interesado en los eventos musicales y culturales que se producen en su entorno.

Nomepierdoniuna tiene una estrecha relación con la empresa Cuatroochenta, puesto que es dirigida por algunos miembros del departamento de *marketing* y comunicación de esta.

1.1.2. Cuatroochenta S.L.

Cuatroochenta es una empresa especializada en el desarrollo integral de aplicaciones para smartphones y tablets y programación avanzada a medida para mejorar procesos de trabajo [18]. En esta empresa cooperan profesionales en los campos de la programación, desarrollo, diseño gráfico, *marketing* y comunicación, con tal de llevar a cabo productos competentes en su concepto, diseño, implementación y explotación posterior.

La filosofía que sigue la empresa promueve que cada proyecto sea creado con una metodología de colaboración real y una altísima implicación por parte de los miembros del equipo. De forma simplificada, el desarrollo integral que proponen conlleva:

- **Creación de concepto o idea de aplicación:** estudio de los objetivos del cliente, características del usuario y asesoramiento sobre la funcionalidad y estructura de la aplicación en base a su modelo de negocio u objetivos y a la usabilidad de la aplicación. Ayuda en la definición de ideas para desarrollar aplicaciones viables y eficaces.
- **Diseño de interfaz y experiencia de usuario app:** obtención del máximo rendimiento que ofrecen las diferentes plataformas para desarrollar interfaces amigables que ofrezcan una óptima experiencia de usuario.
- **Implementación y desarrollo de las aplicaciones móviles y parte servidora:** transformación de la idea en realidad, testeando y asegurando un funcionamiento óptimo antes de su lanzamiento.
- **Lanzamiento, promoción y explotación de la aplicación:** lanzamiento de la aplicación y promoción a través de los medios más adecuados según la estrategia y objetivos del cliente y las posibilidades de las diferentes plataformas (e.g. App Store, Google Play, Windows Market Place).

Recientemente, la empresa ha decidido comenzar un proceso de pivotaje del desarrollo nativo de aplicaciones móviles, hacia el desarrollo híbrido utilizando React Native. Para ello, ha creado un nuevo departamento dedicado a esta tecnología y ha trasladado a algunos miembros de departamentos de desarrollo nativo iOS y Android a este nuevo. Con esto se pretende poder desarrollar aplicaciones móviles para ambos sistemas de manera única.

La implantación de esta forma de desarrollo está llevando tiempo y esfuerzo, dado que requiere formación y además se trata de una tecnología que aún está en constante evolución. Sin embargo, la empresa ya ha obtenido buenos resultados tras publicar algunas aplicaciones desarrolladas con React Native que están funcionando en el mercado.

1.1.3. Alcance del proyecto

Este proyecto consiste en desarrollar una aplicación móvil para iOS y Android que muestre contenido ofrecido por Nomepierdoniuna, además de algunas funcionalidades que faciliten el acceso y distribución de este.

Actualmente existe una aplicación móvil nativa para iOS con el mismo propósito, pero ha quedado obsoleta con el paso de los años. Por ello, el producto que se ha realizado cuenta con la misma funcionalidad salvo mejoras que se han propuesto durante su desarrollo, resultando ser una renovación de la versión anterior. Esta nueva aplicación aporta la mejora de servir tanto para iOS como para Android, dado que anteriormente estaba dedicada únicamente al primero. Además hace uso de tecnologías recientes, ofreciendo más mantenibilidad y longevidad al producto.

La aplicación permite consultar las noticias, eventos y puntos de interés que se publican en Nomepierdoniuna, mostrando estos últimos situados en un mapa. También permite seleccionar eventos como favoritos para poder tener posteriormente un acceso más rápido a ellos. Por último, los usuarios pueden enviar eventos como sugerencia a través de la aplicación para que sean publicados en la *webzine*. Al igual que en la web, no se ha requerido ningún sistema de autenticación o cuentas de usuario.

Además de la funcionalidad básica, constituida por la del sistema antiguo y algunas mejoras ya decididas previamente, tras entrevistar al cliente para realizar la elicitación de requisitos, se ha decidido añadir las siguientes funcionalidades:

- Permitir la búsqueda de eventos mediante filtros.
- Poder compartir eventos mediante aplicaciones externas (e.g. Whatsapp) generando una tarjeta en formato imagen.
- Añadir las secciones de TV y canción de la semana, ofrecidas en la web y que no se ofrecían en la antigua aplicación móvil.
- Cambiar la forma de presentar la publicidad para que tenga más impacto.

Las siguientes se han añadido como resultado del *feedback* obtenido del cliente, al probar versiones del producto incremental:

- Permitir también la búsqueda de noticias mediante filtros.
- Poder realizar búsquedas por título de noticias.
- Poder ver en el mapa la ubicación del usuario, y enlazar con aplicaciones externas (e.g. Google Maps) para ver cómo llegar a los puntos de interés.

Por último, las siguientes han sido añadidas durante el diseño de la interfaz de usuario:

- Permitir también la búsqueda por título de eventos y puntos de interés, con sugerencias de búsqueda.
- Poder consultar los eventos que se van a realizar en un punto de interés.
- Poder compartir también noticias.
- Permitir seleccionar también noticias favoritas.

El alcance de este sistema no incluye ninguna gestión sobre la *webzine* Nomepierdoniuna, ya que esta es independiente y con respecto al proyecto constituye solo una fuente de información, que será accedida a través de servicios web.

Los requisitos de la aplicación han sido acordado con David Hernández Beltrán, coordinador de Nomepierdoniuna y miembro del departamento de comunicación de la empresa, quién toma el rol de cliente en este proyecto. Los servicios web necesarios han sido proporcionados por Sergio Aguado González, CTO de Cuatroochenta y supervisor del proyecto. Por otra parte, los diseños de la aplicación han sido ofrecidos por el departamento de diseño, y su posterior lanzamiento y explotación también implicarán a otros departamentos como los de *marketing* y comunicación.

Como se ha explicado en la sección de Nomepierdoniuna, los usuarios potenciales de esta aplicación son los residentes de la provincia de Castellón, interesados en los eventos musicales y culturales que se producen en su entorno.

1.2. Objetivos del proyecto

El principal objetivo de este proyecto es proporcionar un acceso más cómodo a la información publicada en Nomepierdoniuna desde dispositivos móviles. Para lograrlo, se ha desarrollado una aplicación móvil con este propósito que reemplaza a la antigua versión, junto con algunas mejoras derivadas de la experiencia obtenida. Los datos utilizados para valorar esta experiencia fueron obtenidos a través de Google Analytics, constituyendo una fuente de información de uso directa.

Uno de los datos conocidos tras analizar el sistema anterior es que, aún accediendo desde el navegador web, había más usuarios accediendo mediante dispositivos móviles que desde escritorio. Es posible que se deba a que la versión móvil no estaba disponible para los usuarios con sistema Android. Este hecho ha motivado que la nueva aplicación se ofrezca para ambos sistemas operativos, iOS y Android, ampliando el número de usuarios que podrán acceder desde el móvil de manera más cómoda, en lugar de acceder a la versión web a través de un navegador.

Otro de los datos obtenidos indica que la sección de agenda es, con una gran diferencia, la más accedida del sistema. Por ello, se ha dado prioridad a las mejoras en esta sección. Estas mejoras consisten en poder filtrar la lista de eventos de la misma manera que se permite en la web, y poder compartir de manera sencilla entradas de la agenda en forma de tarjeta a través de cualquier otra aplicación de mensajería o red social.

En resumen, el objetivo principal está dividido en los siguientes subobjetivos:

- Desarrollar una nueva aplicación con la misma funcionalidad base que la antigua.
- Conseguir que los usuarios que acceden desde dispositivos móviles lo hagan a través de la aplicación, puesto que ahora se ofrece también para Android.
- Mejorar el sistema anterior ampliando su funcionalidad. Se ofrecen nuevas secciones que ya se ofrecían en la web y se mejoran las ya presentes en la aplicación.

- De entre las nuevas funcionalidades, dar prioridad a las que mejoran la sección más utilizada, es decir, la agenda de eventos.

1.3. Estructura de la memoria

Este documento consta de seis capítulos. El primero, en el que se incluye esta sección, es un capítulo introductorio. Se ofrece un contexto del entorno del proyecto, y una definición y delimitación de este.

El segundo capítulo describe la situación inicial del sistema que se va a mejorar y desarrollar, seguida de las tecnologías que han sido utilizadas para llevar a cabo el proyecto en su totalidad.

En el tercer capítulo, se explica la metodología que se ha seguido, junto con la correspondiente planificación de tareas y seguimiento del proyecto. También se muestra la estimación de recursos y costes. En general, el capítulo trata sobre la gestión del proyecto.

El cuarto capítulo corresponde al análisis del sistema. Se muestran los requisitos funcionales y su proceso de elicitación, el modelo de datos y la arquitectura del sistema. También se comentan los diseños de la interfaz de usuario y se muestra un subconjunto de estos.

El quinto capítulo corresponde a la implementación de la aplicación. En él se muestra con detalle este proceso, y también se explican las técnicas de verificación y validación empleadas.

Por último, el sexto capítulo recopila las conclusiones adquiridas y enuncia cuál va a ser el futuro del proyecto.

Capítulo 2

Descripción del proyecto

2.1. Situación inicial

La webzine Nomepierdoniuna es gestionada por sus editores a través de WordPress, el cual en su momento fue configurado para satisfacer sus necesidades. Se nos ha dado acceso para poder comprobar cómo editan y publican ellos la información. Esto nos permite saber en qué formato enviarles los datos para que después les sea más cómodo introducirlos, cosa que hacemos por ejemplo cuando la aplicación les envía correos de eventos sugeridos por usuarios. En la Figura 2.1 se puede ver un ejemplo de formulario con el que publican contenido.

Como se ha detallado en el capítulo anterior, el proyecto parte de un sistema previo. Se trata de una aplicación iOS que ya no está operativa debido a la antigüedad y falta de mantenimiento. No se ha podido probar, recurso que habría sido útil para tener una idea fiel de cómo era, pero se van a mantener sus funcionalidades y por ello ya está establecida una gran parte de los requisitos funcionales. Las estadísticas obtenidas del uso del sistema también han dado indicaciones valiosas sobre en qué aspectos centrar las mejoras.

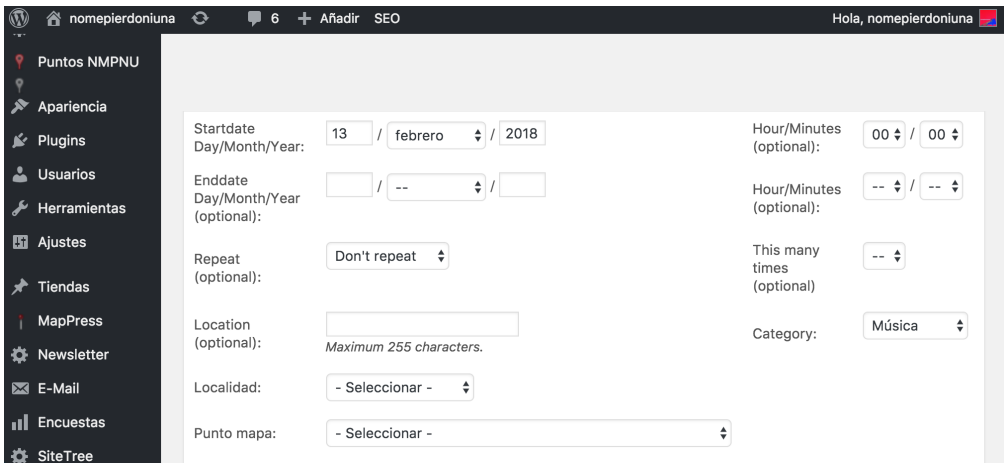


Figura 2.1: *Formulario en el Wordpress de Nomepierdoniuna para publicar eventos.*

Un problema que se observa, y del que se explica su resolución más adelante, es que en su modelo de datos los eventos de la agenda no identifican unívocamente al punto del mapa al que corresponden. Esto ha complicado dos de las funcionalidades extra que se quería añadir: poder acceder desde un evento al lugar donde se va a realizar, y poder ver los eventos que se van a realizar en un lugar.

La arquitectura del sistema de Nomepierdoniuna está detallada de forma complementaria con la del nuevo sistema en el Capítulo 4, dentro del apartado de diseño de la arquitectura del sistema.

Al ser una aplicación completamente nueva, salvo por el conocimiento de los requisitos funcionales básicos, el desarrollo parte desde cero. En este documento se trata solo la parte cliente (i.e. la aplicación móvil) y no la parte servidora, la cual consiste en una serie de servicios web que se han ido solicitando según necesidad.

2.2. Tecnologías

Este proyecto se ha realizado con el *framework* React Native, orientado al desarrollo de aplicaciones móviles. Para explicar en qué consiste, antes se introduce brevemente la librería React, el pilar más importante sobre el que se asienta. Después, se explican las principales funcionalidades que ofrece, cómo lleva a cabo la gestión de dependencias y el montaje del proyecto, y quiénes desarrollan el *framework* y con qué filosofía.

Una de las librerías que se ha utilizado, Redux, merece un trato destacado puesto que va a tener impacto sobre la forma en que la se reparte la lógica del código. Además, el sistema utiliza tecnologías de notificación *push* del servidor a los clientes. Por lo tanto se dedica parte de la sección a explicar estos temas.

Para terminar, se describen el entorno de desarrollo y aspectos como el sistema de gestión y publicación de versiones. También se explica el uso de JIRA como herramienta de soporte en la gestión de proyectos.

2.2.1. React

React es una librería JavaScript para la implementación de interfaces de usuario [14]. Su principal característica es la encapsulación de los elementos de la interfaz en bloques separados, llamados componentes. Esto, además de facilitar la modularidad y reutilización de código, se debe a que React se basa en un diseño declarativo de la interfaz de usuario. En este contexto se refiere a que cada componente especifica cómo debe construirse.

Los componentes mantienen su propio estado y tienen un ciclo de vida, es decir, una serie de eventos a los que pueden responder, como los momentos previos y posteriores a su construcción o desmontado. También se puede especificar cuándo deben actualizarse o no, permitiendo hacer más eficiente la actualización de la parte de la interfaz que se encuentra en pantalla cuando se debe responder a algún cambio.

Internamente hace uso del llamado VirtualDom, una abstracción a más nivel de la representación en memoria de los elementos de la interfaz de usuario [21]. Es común que las interfaces de usuario se implementen utilizando estructura de árbol, y en el contexto al que se orienta React, la manipulación directa de esta estructura es innecesariamente costosa. Como idea general y sin entrar en detalles, para ofrecer un mejor rendimiento primero se actualiza el VirtualDom, se compara el DOM que resultaría de este con el DOM actual y se realiza el mínimo cambio posible.

React se suele utilizar junto con JSX (JavaScript Syntax eXtension). Se trata de una extensión sintáctica para JavaScript que permite definir cómo se renderizan los componentes de manera parecida a como se haría en un lenguaje de marcado [7]. Su uso no es indispensable, pero sí se ha utilizado en este proyecto debido a la comodidad que conlleva. React utiliza una herramienta llamada Babel para realizar posteriormente la transpilación a JavaScript.

Esta librería está orientada al desarrollo de aplicaciones de una sola página, una tendencia reciente en las aplicaciones web que se suele conseguir jugando con la visibilidad de los elementos de la interfaz según el estado de la aplicación. Con ello, se consigue una experiencia más fluida de cara al usuario. También hay casos en los que se mapea la url con un estado concreto de la aplicación para dar la sensación de que se navega de la misma manera que se hacía en las páginas web tradicionales. Aún así, a diferencia de lo que ocurría en estas, la página no se está recargando.

React ha sido desarrollado por Facebook e Instagram. Es un producto de código abierto [13] y está dispuesto para ser extendido por desarrolladores o grupos independientes, hecho que demuestra la cantidad de librerías de este tipo que se suelen utilizar de forma complementaria a React en la mayoría de proyectos.

2.2.2. React Native

El *framework* React Native permite desarrollar aplicaciones móviles, tanto para iOS como para Android, con JavaScript y creando la interfaz de usuario de la misma manera que se hace en React [17]. Además, ofrece una serie de componentes nativos, es decir, que se corresponden con componentes en su correspondiente plataforma, que permiten un aspecto visual y rendimiento similar al de las aplicaciones nativas.

Está construido sobre Node para realizar, entre otras cosas, el montaje del proyecto y la gestión de dependencias con el gestor de paquetes npm. Node es un entorno de ejecución para JavaScript, cuyo ecosistema de paquetes, npm, es el ecosistema mas grande de librerías de código abierto en el mundo [11]. React Native también ofrece un servidor de desarrollo con compilación incremental que permite probar la aplicación en poco tiempo después de realizar cambios.

Por lo anterior, se requiere tener instalado Node. También hay que instalar Watchman si se quiere utilizar de forma cómoda el servidor de desarrollo. Se trata de una herramienta de Facebook que está atenta a los cambios que se producen en los ficheros que corresponden al proyecto, lanzando eventos como respuesta [22]. En este caso, realiza la recompilación incremental cuando se han producido cambios en el código.

Este *framework* se ofrece distribuido en dos paquetes npm. Uno de ellos contiene el código de React Native y ha de ser instalado localmente en el proyecto. El segundo se ha de instalar de forma global y consiste en una serie de comandos útiles para el uso de este *framework* (e.g. para inicializar proyectos, iniciar el servidor de desarrollo o configurar módulos) [15].

Los módulos que introducen componentes o código nativo traen consigo los ficheros y configuraciones que hacen falta para su funcionamiento. Se pueden instalar manualmente, o utilizando uno de los comandos mencionados antes para este propósito. Algunas librerías pueden tener detalles que es necesario aplicar de forma manual, y por ello todas suelen incluir una guía de instalación. Como es habitual en los *frameworks* de desarrollo multiplataforma, se ofrecen formas de crear código específico de plataforma, es decir, código que solo se va a ejecutar en la plataforma indicada.

Al igual que React, ha sido desarrollado por Facebook y también es de código abierto [16]. Sigue la misma filosofía en cuanto a extensibilidad.

2.2.3. Redux

Redux es una implementación del patrón de diseño de software Flux. No es la única que hay, pero observando en GitHub a día de hoy el número de descargas al mes y el mantenimiento que tiene frente a otras, se puede ver que es la más utilizada. Es ofrecida para varios *frameworks*, entre ellos React Native.

Se trata de un patrón para crear la parte cliente de aplicaciones, motivado principalmente por el surgimiento de aplicaciones de una sola página [5]. Por su naturaleza, en ellas se favorece el crecimiento excesivo de complejidad del código a medida que la aplicación crece, con los problemas que esto conlleva (e.g. difícil de mantener, depurar y ampliar). Para resolver esto, Flux toma principalmente dos medidas.

La primera de ellas es tener controlado el estado de la aplicación. En lugar de encontrarse disperso por todo el código, se tiene centralizado en una sección concreta y con una estructura organizada. Esta parte es la que en el patrón se denomina *store*. Colateralmente, esto también hace posible tomar *snapshots* de los estados que va tomando la aplicación. De esto último se puede sacar provecho por ejemplo para mantener un histórico de estados que se puedan deshacer y rehacer, o poder guardar un estado para retomarlo en otro momento.

La segunda medida es restringir y tener controlada la manera en la que el estado puede cambiar. Para ello, las acciones que pueden hacer que el estado cambie, llamadas *actions*, están definidas y solo pueden ser ejecutadas por un *dispatcher*. Tras la ejecución de una acción, el *store* comunica a la *vista* el estado actualizado. El estado además ha de ser inmutable, y el que resulta tras llevar a cabo una acción es uno nuevo, no una modificación del anterior. En la Figura 2.2 se puede ver un diagrama de este patrón, donde se aprecia también en qué sentido fluyen los eventos.

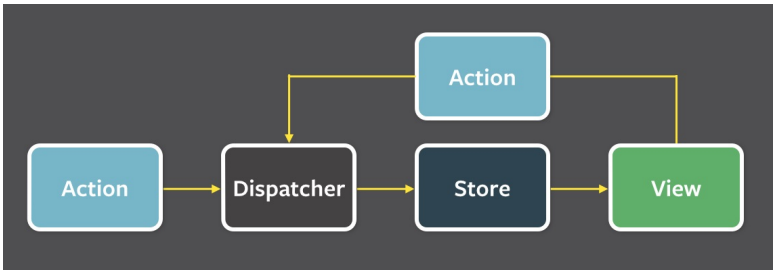


Figura 2.2: *Diagrama de Flux [6].*

En Redux aparece un elemento más. Cuando un *dispatcher* ejecuta una acción, el *store* envía esa acción junto con el estado actual a los *reducers*. Estos últimos son los que se encargan de construir el nuevo estado y notificar de los cambios a la vista [9]. En la Figura 2.3 se muestra el diagrama que resulta del anterior para constituir Redux.

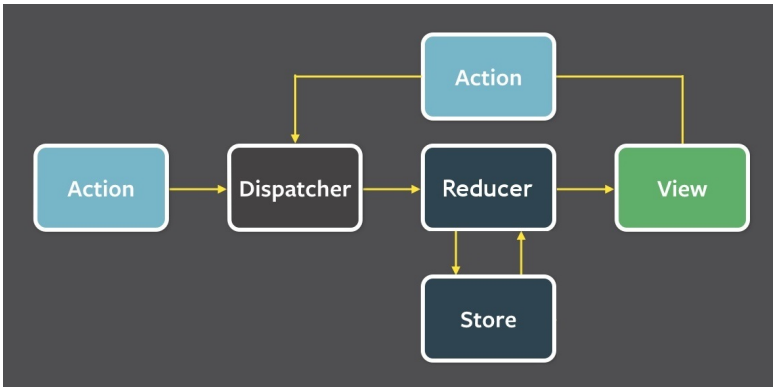


Figura 2.3: *Diagrama de Redux [6].*

Este patrón tiene cierto parecido con el conocido MVC, del cual también existe una familia de patrones que surgen de su variación (e.g. MVP y MVVM). La característica de Redux frente al MVC y sus otros derivados, es que en este primero los eventos fluyen en una sola dirección. Con esto, se está tomando otra medida para reducir la complejidad del código, puesto que que el comportamiento del sistema es más previsible.

2.2.4. Notificaciones *push*

Las notificaciones push son mensajes que se envían directamente desde un servidor a dispositivos móviles con sistema operativo iOS y Android, entre otros [12]. Estas pueden ser recibidas incluso cuando la aplicación a la que corresponden no está en ejecución, mostrando al usuario un acceso directo a alguna funcionalidad de esta o al lanzamiento de alguna acción.

OneSignal ofrece de forma gratuita una plataforma para gestionar estas notificaciones [20]. También hay disponible un paquete npm que facilita el uso de este sistema en React Native.

Por medidas de seguridad, en iOS es necesario que en el montaje firmado de la aplicación se especifique si se hace uso de este tipo de notificaciones.

En este proyecto, las notificaciones no se han correspondido con ningún requisito funcional. Su integración ha sido un requisito de la empresa, por motivos de formación y para poder darles uso en un futuro.

2.2.5. Entorno de desarrollo

Con React Native se está desarrollando para dos plataformas distintas, cada una con sus distintas versiones y dispositivos, y estos con distintos tamaños de pantalla. Es imprescindible probar la aplicación por lo menos en dispositivos que representen a un conjunto con características parecidas, para poder encontrar problemas o detalles a cambiar.

Para poder utilizar el emulador de iPhone, el proyecto se ha desarrollado desde Mac OS. En concreto, desde un portátil MacBook Pro retina 13". Se ha utilizado WebStorm como IDE, un producto de JetBrains enfocado al desarrollo en el ecosistema JavaScript actual, que está dando cobertura a la mayoría de *frameworks* populares en este ámbito [23].

En ocasiones ha habido que realizar modificaciones en los proyectos generados para Android e iOS. Para cada uno de ellos se ha usado el correspondiente IDE por excelencia: Android Studio para Android y Xcode para iOS. También se ha utilizado el emulador de Android, y para poder realizar pruebas de forma física, la empresa tiene a disposición una gran variedad de dispositivos de ambos sistemas, tanto móviles como tablets.

Para dar soporte a la administración y seguimiento de tareas, se ha a utilizado la herramienta JIRA. Esta ofrece funcionalidad para proyectos llevados a cabo con Scrum y además es la herramienta utilizada por Cuatroochenta para el registro de horas de trabajo.

La gestión y publicación de versiones también tiene su papel en el entorno de desarrollo, pero se explica a continuación en su propio apartado.

2.2.6. Gestión y publicación de versiones

Como control de versiones se ha utilizado Git, teniendo el repositorio del proyecto alojado en la cuenta de Bitbucket de Cuatroochenta. Este último es un producto de Altassian que ofrece servicio de repositorios Git de forma gratuita, tanto públicos como privados. Es una alternativa factible a GitHub, dado que este requiere pagar para el alojamiento de repositorios privados. Bitbucket también ofrece desde hace poco servicios de integración continua, pero la empresa utiliza otra plataforma para este propósito.

Las versiones se publican a través de Visual Studio App Center. Esta plataforma ofrece servicios de integración continua, tras realizar la vinculación con un repositorio. Es compatible con Bitbucket y ampliamente configurable. En este caso, dada una rama del control de versiones correspondiente a una versión de la aplicación (e.g. una versión beta), se ha configurado para

que realice el montaje, firma y publicación cada vez que se realice un *push* (i.e. adición de cambios realizados) en esta rama.

La firma de versiones en Android es más sencilla de realizar. Es posible crear los certificados desde Android Studio y configurar fácilmente el proceso de montaje con Gradle para que haga la firma. En el caso de iOS, el procedimiento no es tan trivial. Primero hay que registrar la aplicación en la cuenta de Apple de la empresa, en este caso Cuatroochenta. Una vez registrada, se deben crear perfiles de desarrollo y distribución, los cuales se pueden exportar como ficheros firmados. Una vez obtenidos estos ficheros, hay que suministrar al App Center el fichero del perfil de distribución, junto con el certificado correspondiente para firmar la aplicación durante su montaje.

Una vez se han publicado las versiones para ambos dispositivos, pasan a estar disponibles para su distribución. Esta puede hacerse de manera pública, o restringida solo a unos dispositivos concretos para realizar pruebas.

Capítulo 3

Planificación del proyecto

Este capítulo detalla la metodología que se ha elegido para el desarrollo del proyecto y su correspondiente planificación, teniendo en cuenta que se ha seguido un modelo ágil.

En la primera sección, se justifica el uso de la metodología Scrum. También se explica en qué consiste y algunas adaptaciones que se han hecho para este caso. Seguidamente, se muestra la planificación de las tareas que se han llevado a cabo durante la estancia en prácticas.

Para finalizar el capítulo, se muestran las estimaciones desglosadas de recursos y costes, junto con el seguimiento del proyecto y cómo se han tratado las desviaciones.

3.1. Metodología

La metodología que se ha utilizado en este proyecto es una versión simplificada de Scrum. Simplificada porque, como se explica después de detallar la metodología, en este caso se han omitido los aspectos relacionados con la coordinación del equipo.

Esta metodología consiste en un marco de desarrollo ágil en el que las tareas por hacer, definidas como historias de usuario en la pila del producto, son agrupadas en paquetes a realizar durante un período de tiempo llamado *sprint*, dando lugar a versiones incrementales del producto. Se puede ver un diagrama explicativo en la Figura 3.1.

Los roles principales que intervienen en Scrum son:

- Propietario del producto. Ayuda a escribir las historias de usuario, establecer sus prioridades y validar el progreso del producto.
- Scrum Master. Se encarga de facilitar el trabajo o eliminar obstáculos que se le presenten al equipo.
- Equipo Scrum. Consiste en el equipo que lleva a cabo el producto.

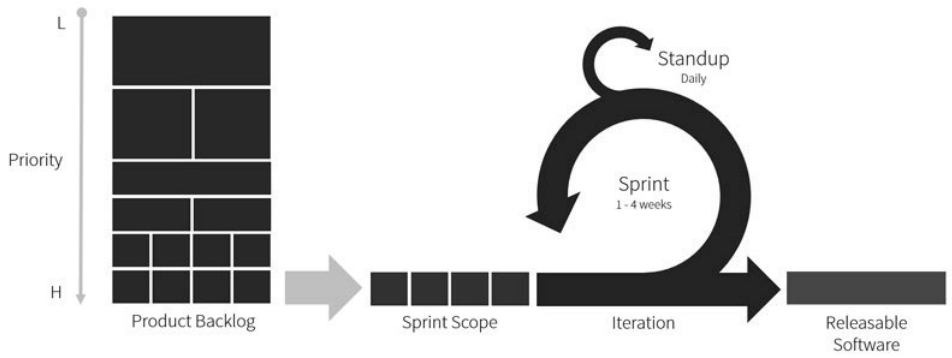


Figura 3.1: *Diagrama de Scrum [2].*

Al ser una metodología ágil, responde bien ante cambios que se decidan hacer en pleno desarrollo. La pila del producto está abierta a modificaciones o ampliaciones y se promueve una comunicación continua con el cliente a través del producto incremental para planificar el *sprint* siguiente. En el caso de este proyecto, el producto incremental se ha ofrecido en forma de versiones beta publicadas en la empresa de forma interna.

Uno de los beneficios de Scrum es la auto-organización del equipo a través de reuniones diarias, de inicio y de fin del *sprint*. Esta parte ha sido omitida en este proyecto, dado que el equipo ha estado constituido tan solo de un miembro. No obstante se ha elegido esta metodología por la organización dinámica de tareas en constante comunicación con el cliente, al no tratarse de un proyecto grande ni fuera de lo común, en el que hay aspectos funcionales abiertos a cambios desde su inicio.

Para poder realizar la planificación de tareas, esta metodología se basa en la asignación de puntos de historia a las historias de usuario. Estos puntos son una medida relativa de su complejidad, independientes del tiempo que puede llevar realizarlas. Por otra parte, son dependientes del equipo de desarrollo, ya que al ser una medida relativa, no coincidirá el valor, numérico o no, que asignaría un equipo a una historia de usuario, frente al que asignaría otro. Como sistema de puntuación, se suele utilizar la serie de Fibonacci para no dar a pensar que hay una precisión matemática detrás, omitiendo los elementos 1 y 2 dada su cercanía con el 3 [25].

En base al establecimiento de estos puntos, se estima cuántos de ellos es capaz de realizar el equipo en un *sprint*. Se eligen historias de usuario que en puntos sumen aproximadamente esta cantidad, y se reestima esta cantidad según los resultados de los *sprints*. Lo recomendable es realizar la reestimación cada 3 *sprints*, pero en este caso, se ha decidido realizarla según necesidad, si no se han podido terminar todas las historias, o si se han terminado antes de lo previsto. La justificación de porqué utilizar esta medida en lugar de horas, es porque se trata de una estimación de tamaño en lugar de tiempo [24]. Al no requerir tanta precisión, supone menos esfuerzo realizar una planificación de este tipo. Además, permite flexibilidad ante imprevistos, a la vez que a la larga permite establecer la velocidad a la que es capaz de trabajar el equipo.

La herramienta JIRA, mencionada en la Sección 2.2.5, entorno de desarrollo, se ha utilizado para dar soporte a la administración y seguimiento de tareas [8]. Esta ofrece funcionalidad

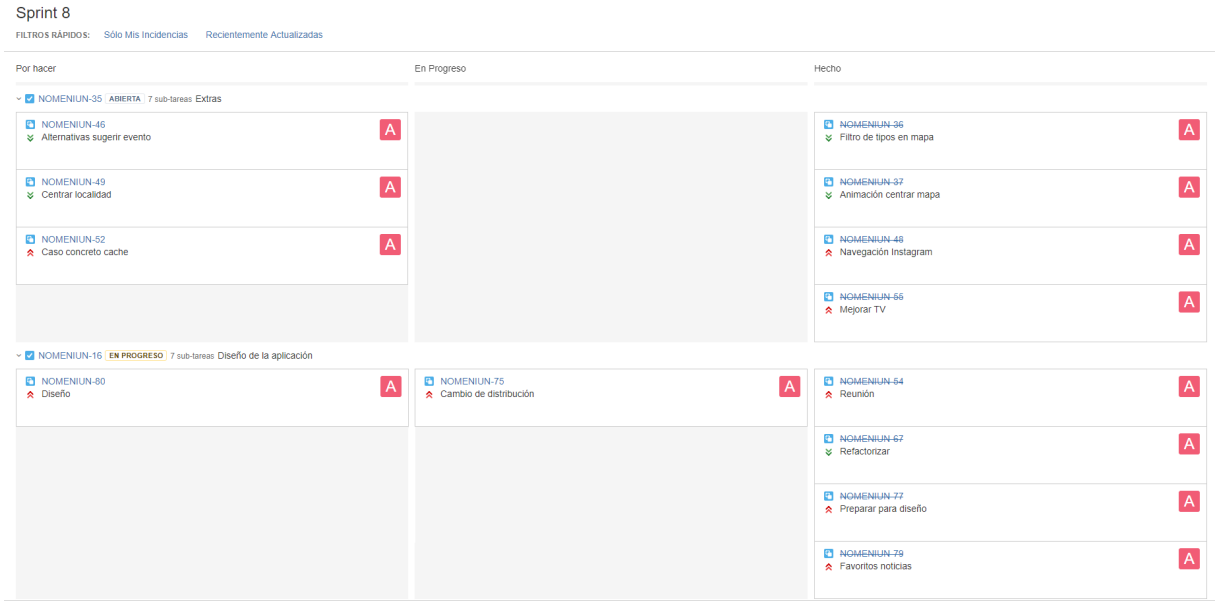


Figura 3.2: *Pizarra Kanban de un sprint.*

específica para proyectos llevados a cabo con Scrum. En ella se ha gestionado tanto la pila del producto como la organización y desarrollo de los *sprints*. Estos últimos se han gestionado además con el uso de pizarras Kanban como la que se muestra en la Figura 3.2, donde las historias de usuario del *sprint* se pueden clasificar según en qué estado se encuentran (e.g. pendiente, en progreso y cerrada). También permite generar una serie de informes vinculados a esta metodología, tanto de realización de tareas en un *sprint* frente a lo estimado, como de rendimiento general del equipo.

Uno de los informes más relevantes en esta metodología, es el *Burndown Chart*. Este diagrama, aplicado a un *sprint*, muestra los puntos de historia que quedan por realizar en función del tiempo. El resultado final es parecido a una recta con pendiente negativa, que se cruza idealmente con el eje horizontal antes del fin del *sprint*. Este diagrama ofrece información sobre el trabajo realizado, el que queda por hacer, y con la pendiente resultante en un momento dado, da una idea de cuándo parece que se va a completar todo el trabajo [1].

3.2. Planificación

Antes de entrar en aspectos de planificación relacionados con la metodología, a continuación se describe la planificación de la estancia en prácticas completa, donde se incluye la realización del proyecto junto con otras tareas. Para ello, se ha considerado oportuno crear un diagrama de Gantt, mostrado en la Figura 3.3, donde se reflejan también las tareas de inicio del proyecto.

Se trata de tareas tales como la formación en las tecnologías utilizadas, reuniones con miembros involucrados y el desarrollo de una versión simple de la aplicación con la funcionalidad básica. Esto último ha servido como punto de partida del desarrollo y como un primer contacto con estas tecnologías tras la formación.

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1		**NMPNU**	**300 horas**	**lun 20/11/17**	**vie 16/02/18**
2		**Inicio del proyecto**	**65 horas**	**lun 20/11/17**	**lun 11/12/17**
3		**Formación**	**30 horas**	**lun 20/11/17**	**mar 28/11/1**
4		Formación en React Native	20 horas	lun 20/11/17	vie 24/11/17
5		Formación en Redux	10 horas	vie 24/11/17	mar 28/11/17
6		Réplica básica del sistema anetrior	25 horas	mar 28/11/17	mar 05/12/17
7		**Definición del proyecto**	**10 horas**	**mar 28/11/17**	**jue 30/11/17**
8		**Reuniones**	**5,5 horas**	**mar 28/11/1**	**mié 29/11/1**
9		Reunión inicial con el cliente	1 hora	mar 28/11/17	mar 28/11/17
10		Reunión con el encargado del backend	0,5 horas	mar 28/11/17	mar 28/11/17
11		Análisis de viabilidad	3,5 horas	mar 28/11/17	mié 29/11/17
12		Runión de validación con el cliente	0,5 horas	mié 29/11/17	mié 29/11/17
13		Establecimiento del alcance y objetivos	2,5 horas	mié 29/11/17	mié 29/11/17
14		DCU	2 horas	mié 29/11/1	jue 30/11/17
15		**Planificación y documentación**	**10 horas**	**mar 05/12/17**	**lun 11/12/17**
16		Definición de tareas y estimaciones	2,5 horas	mar 05/12/17	mar 05/12/17
17		Diagrama de Gantt	2,5 horas	mar 05/12/17	jue 07/12/17
18		Documentación de la propuesta técnica	5 horas	jue 07/12/17	lun 11/12/17
19		Entrega de la propuesta técnica	0 horas	lun 11/12/17	lun 11/12/17
20		**Desarrollo técnico del proyecto**	**215 horas**	**lun 11/12/17**	**lun 12/02/18**
21		**Sprints**	**215 horas**	**lun 11/12/17**	**lun 12/02/18**
22		Sprint 1	25 horas	lun 11/12/17	lun 18/12/17
23		Sprint 2	25 horas	lun 18/12/17	mar 26/12/1
24		Sprint 3	20 horas	mar 26/12/1	mar 02/01/1
25		Sprint 4	20 horas	mar 02/01/1	lun 08/01/18
26		Sprint 5	25 horas	lun 08/01/18	lun 15/01/18
27		Sprint 6	25 horas	lun 15/01/18	lun 22/01/18
28		Sprint 7	25 horas	lun 22/01/18	lun 29/01/18
29		Sprint 8	25 horas	lun 29/01/18	lun 05/02/18
30		Sprint 9	25 horas	lun 05/02/18	lun 12/02/18
31		**Web services**	**25 horas**	**lun 11/12/17**	**lun 18/12/17**
32		Solicitar y esperar nuevos web services necesarios	0 horas	lun 11/12/17	lun 11/12/17
33		Añadir tarea de integración de web services	0 días	lun 18/12/17	lun 18/12/17
34		**Diseño de UI**	**20 horas**	**mar 26/12/1**	**mar 02/01/1**
35		Solicitar diseños de UI	0 horas	mar 26/12/1	mar 26/12/1
36		Añadir tarea de implementción de diseños de UI	0 días	mar 02/01/18	mar 02/01/18
37		**Testing**	**25 horas**	**lun 22/01/18**	**lun 29/01/18**
38		Solicitar testing	0 horas	lun 22/01/18	lun 22/01/18
39		Añadir tareas resultado del testing	0 días	lun 29/01/18	lun 29/01/18
40		**Fin del proyecto**	**20 horas**	**lun 12/02/18**	**vie 16/02/18**
41		Cerrar el estado actual del producto	15 horas	lun 12/02/18	jue 15/02/18
42		Evaluación de futura puesta en marcha y lanzamiento	5 horas	jue 15/02/18	vie 16/02/18
43		Entrega	0 días	vie 16/02/18	vie 16/02/18

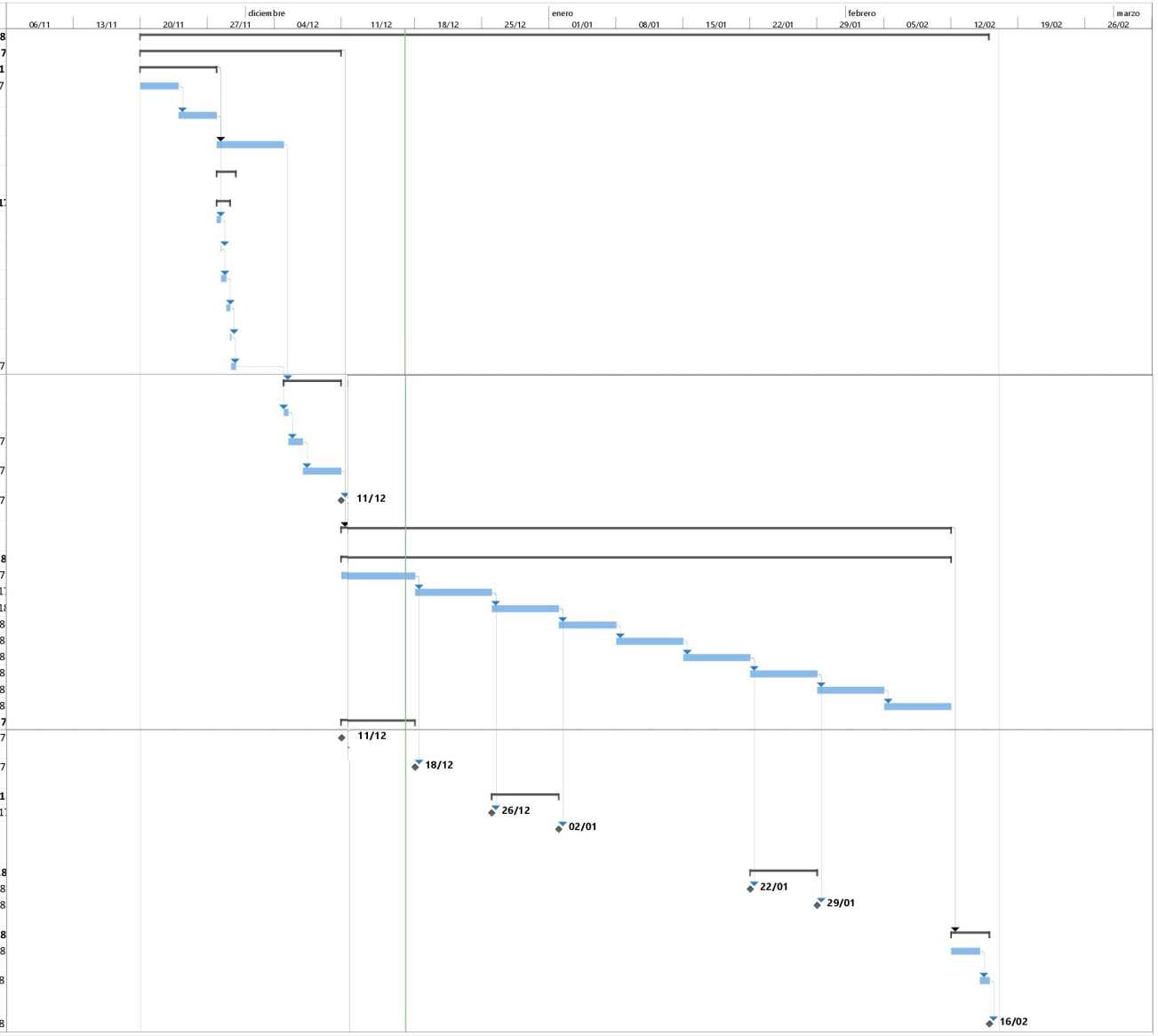


Figura 3.3: *Diagrama de Gantt de la estancia en prácticas.*

En este diagrama, también se pueden ver los *sprints* planificados con duración de una semana, junto con hitos a realizar al llevar terminados un cierto número de *sprints*. La solicitud de los primeros servicios web ha sido planificada para el principio, después de haber realizado el análisis de requisitos y definido el proyecto. Un poco más adelante, los diseños de interfaz de usuario y por el final, un paso del producto por el departamento de *testing*. Hay que tener en cuenta que se trata de un proyecto realizado en una estancia en prácticas, hecho que implica una restricción temporal a las 300 horas que dura esta.

En un proyecto real, no es habitual que los diseños de interfaz se den prácticamente al final del desarrollo. En este caso, al tratarse de un proyecto de prácticas, no ha tenido prioridad suficiente como para tenerlas ya preparadas al inicio. Se ha tenido en cuenta la necesidad de adaptar la aplicación a los diseños cuando estos estuviesen preparados. Por ello, mientras tanto se ha avanzado en el proyecto con una interfaz de usuario simple y que pueda ser fácilmente modificable.

La asignación de tareas de la pila del producto a los *sprints* se ha hecho de forma dinámica, dando prioridad a las mejoras en la sección de la agenda. La planificación ha estado abierta a la valoración de ampliaciones y modificaciones funcionales. Dada esta incertidumbre, se han programado *sprints* hasta poco antes de finalizar las 300 horas de la estancia, momento en el que se debe valorar el estado del proyecto.

Cuadro 3.1: *Asignación de puntos de historia.*

ID	**Rol**	**Funcionalidad**	**PH**
HU01	Como usuario de la aplicación necesito	consultar los eventos de la agenda.	3
HU02		sugerir eventos para que sean publicados.	5
HU03		ver en un mapa dónde se va a realizar un evento.	3
HU04		consultar las noticias publicadas.	3
HU05		poder ver el *post* de una noticia, vídeo o canción.	5
HU06		compartir eventos en forma de imagen.	3
HU07		compartir noticias como enlaces a su *post* en la web.	1
HU08		consultar en un mapa los puntos de interés publicados, viendo además mi ubicación.	5
HU09		ver cómo llegar a un punto de interés desde mi ubicación.	3
HU10		ver los eventos que se van a realizar en un punto de interés.	3
HU11		filtrar los eventos, noticias y puntos de interes según una serie de criterios.	8
HU12		buscar por nombre entre los eventos, noticias y puntos del mapa, viendo además sugerencias de búsqueda.	8
HU13		consultar las canciones de la semana y vídeos publicados.	5
HU14		añadir eventos y noticias como favoritos.	5
HU15		ver los eventos y noticias que he guardado como favoritos.	5

Como sistema de puntuación de historia de usuario, se ha utilizado el explicado como ejemplo en la Sección de metodología. El 3 corresponde a las historias de usuario más sencillas, el 5 a las de dificultad media y el 8 a las de mayor dificultad. No se han utilizado valores mayores. En el Cuadro 3.1 se puede ver la asignación de puntos a cada historia de usuario. Estas se encuentran detalladas en la Sección 4.1.1. No hay que confundir esta tabla de asignación de puntos de historia con la pila del producto inicial, dado que aquí se han incluido también las que han aparecido durante el desarrollo.

3.3. Estimación de recursos y costes del proyecto

En la Figura 3.4, se puede ver el desglose de los costes del proyecto incluyendo personal *hardware* y *software*, junto con el cálculo total. Se ha tenido también en cuenta la contratación por horas del diseñador gráfico encargado de este proyecto. Como desarrollador, he dedicado un porcentaje de las horas a realizar un papel de analista de sistemas, otro al de diseñador de sistemas y otro al de programador, como se ve reflejado en la tabla de la imagen. La cantidad total asciende a 5.133,59 €.

Recurso		Costo			Acumular
		Fijo	Variable (por hora)	Total	
Desarrollador frontend	Analista de sistemas	0,00 €	14,00 €	840,00 €	Prorrateo
	Diseñador de sistemas	0,00 €	14,00 €	840,00 €	Prorrateo
	Programador	0,00 €	8,00 €	1.440,00 €	Prorrateo
Diseñador gráfico		0,00 €	14,00 €	280,00 €	Prorrateo
Licencia Sketchapp		99,00 €	0,00 €	99,00 €	Prorrateo
WebStorm license		129,00 €	0,00 €	129,00 €	Prorrateo
MacBook Pro		1.505,59 €	0,00 €	1.505,59 €	Prorrateo
Tiempo porcentual estimado		Horas total		3.400,00 €	factor humano
Analista de sistema	20,00%	60		1.733,59 €	material
Diseñador de sistemas	20,00%	60		5.133,59 €	TOTAL
Programador	60,00%	180			
Diseñador gráfico		20			
Total horas trabajadas	300				

Figura 3.4: *Costes del proyecto.*

3.4. Seguimiento del proyecto

En esta sección, se detalla el seguimiento del proyecto a partir de los *sprints* realizados. Las tareas anteriores al inicio del Scrum se han realizado según lo previsto. Estas eran las relacionadas con la formación en las tecnologías necesarias, reuniones con el cliente y los miembros de la empresa involucrados, y el comienzo de la implementación del proyecto con una parte de la funcionalidad básica.

Al inicio de la sección de los *sprints*, se muestra la pila del producto inicial, mientras que en la sección de cierre del proyecto, se muestra el estado final de esta.

3.4.1. *Sprints*

La pila inicial al comienzo del Scrum, se muestra en el Cuadro 3.2. Las historias no coinciden exactamente con las que se han mostrado en la asignación de puntos de la panificación. Esto se debe a que, al inicio del proyecto, aún no se habían realizado ciertos cambios que se verán durante el seguimiento. También hay historias de usuario correspondientes a la funcionalidad básica que ya se han implementado antes del Scrum, para poner en práctica lo aprendido en la formación, como se ve en la tarea 6 del diagrama de Gantt. Estas son las de poder consultar los eventos de la agenda, poder consultar las noticias publicadas y poder consultar los puntos de interés de la aplicación.

Cuadro 3.2: *Pila del producto inicial.*

ID	**Funcionalidad**	**PH**
F01	Nuevo sistema de publicidad.	3
F02	Sugerir eventos.	5
F03	Sección de Instagram embebido.	1
F04	Filtrado de eventos.	3
F05	Eventos favoritos.	3
F06	Compartir eventos.	3
F07	Almacenamiento local intermedio.	3
F08	Mejorar rendimiento del mapa.	1
F09	Caché *offline*.	5
F10	Sección TV.	3
F11	Sección canción de la semana.	3

Los identificadores utilizados en este cuadro no coinciden con los utilizados para las historias de usuario de la planificación y el análisis de requisitos funcionales. De esta manera se pueden ir presentando en el orden en el que aparecieron, y juntarlos con los otros requisitos funcionales. En este cuadro también se incluyen estos últimos, de los cuales se pueden ver sus puntos de historia asignados. A continuación, se detalla el seguimiento *sprint* a *sprint*. Se recuerda que estos han sido realizados con una duración de una semana cada uno.

Primer *sprint*

En este *sprint*, se implementó el nuevo sistema de publicidad, se añadió la sección de Instagram y se empezó a desarrollar el filtrado en la agenda de eventos, sin llegar a finalizar esta.

De los siete puntos de historia que tenía asignados, se realizaron los cuatro correspondientes al sistema de publicidad y a la sección de Instagram, quedando por acabar los tres del filtrado de eventos.

Segundo *sprint*

En este *sprint*, se terminó el filtrado de eventos, y se añadió la sección de favoritos, en este punto solo de eventos. Se completaron los seis puntos que habían sido asignados al *sprint*.

El filtrado de eventos aún no estaba integrado con los servicios web, y se utilizaba un mock para ello. La pila del producto tras finalizar este *sprint* se puede ver en el Cuadro 3.3, con la historia de integrar los servicios web añadida, puesto que había que esperar a que estuviesen preparados para integrarlos.

Cuadro 3.3: *Pila del producto al finalizar el segundo sprint.*

ID	**Funcionalidad**	**PH**
F02	Sugerir eventos.	5
F06	Compartir eventos.	3
F07	Almacenamiento local intermedio.	3
F08	Mejorar rendimiento del mapa.	1
F09	Caché *offline*.	5
F10	Sección TV.	3
F11	Sección canción de la semana.	3
F12	Integrar servicio web de filtrado de eventos.	1

Tercer *sprint*

Durante este *sprint*, se implementó el sistema de almacenamiento intermedio sobre el almacenamiento local del dispositivo. También se añadió la funcionalidad de compartir eventos en forma de imagen. De nuevo, los seis puntos asignados al *sprint* fueron completados. Esto indica que en este punto se estaba teniendo una velocidad de trabajo de seis puntos de historia por *sprint*.

Cuarto *sprint*

Este *sprint* se dedicó a añadir la funcionalidad de sugerir eventos, e integrar el servicio web de búsqueda. Hasta ahora se utilizaba un mock para desarrollar la interfaz de usuario correspondiente a la búsqueda de eventos. De nuevo, los seis puntos de historia asignados fueron completados.

Quinto *sprint*

Durante este *sprint*, se implementó el sistema de caché *offline* sobre los datos que se reciben de los servicios web. También se dedicó tiempo a arreglar asuntos del mapa para tener un mejor rendimiento. La velocidad de trabajo se mantuvo constante en seis puntos de historia por *sprint*.

En el siguiente *sprint* se pretendía publicar una versión beta, para mostrar al cliente el estado actual de la aplicación. La pila del producto tras finalizar este *sprint* se puede ver en el Cuadro 3.4, con la historia de publicar la versión beta añadida. Esta vez, la publicación tiene un coste mayor, dado que hay que realizar una serie de preparativos, pero las publicaciones posteriores no requerirán este esfuerzo.

Cuadro 3.4: *Pila del producto al finalizar el quinto sprint.*

ID	**Funcionalidad**	**PH**
F10	Sección TV.	3
F11	Sección canción de la semana.	3
F13	Publicar versión beta.	3

Sexto *sprint*

En el sexto *sprint*, se llevó a cabo el desarrollo de la sección TV. Para finalizar el *sprint*, se realizaron los preparativos para publicar una versión beta, y se publicó esta para ser ofrecida al cliente. De nuevo, seis puntos fueron asignados y completados.

Reunión del diseño de la interfaz de usuario

Los resultados de la reunión de diseño se pueden dividir en dos aspectos. En primer lugar, el *feedback* del cliente tras probar la versión beta. Se trata de añadir una serie de requisitos funcionales que no se habían solicitado en un principio, bien porque había considerado obvios o se han pensado más adelante.

Estos requisitos son los de poder filtrar también noticias, poder ir de un evento al punto del mapa donde se realiza, que el mapa muestre también la ubicación del usuario, poder consultar cómo llegar a un punto de interés desde esta ubicación, y poder ver los *posts* completos de noticias, canciones y vídeos. También se ha descartado la funcionalidad de mostrar el perfil de Instagram de Nomepierdoniuna en la aplicación, dado que no era lo que el cliente se imaginaba. En el Cuadro 3.5 se puede ver la pila del producto actualizada.

Cuadro 3.5: *Pila del producto tras la reunión del diseño de la interfaz.*

ID	**Funcionalidad**	**PH**
F11	Sección canción de la semana.	3
F14	Filtrado de noticias.	3
F15	Ver evento en el mapa.	3
F16	Ver ubicación del usuario y cómo llegar a un punto.	1
F17	Ver *posts* completos.	3

El segundo resultado de la reunión, es que durante las semanas siguientes, el departamento de diseño se va a encargar de realizar los diseños de interfaz gráfica de la aplicación, intentando terminarlos antes de que finalice la estancia en prácticas.

Séptimo *sprint*

En el séptimo *sprint*, se añadieron los filtros de noticias, poder ver la ubicación del usuario en el mapa, poder consultar cómo llegar a un punto de interés desde esta ubicación y poder ir de un evento a su punto del mapa. Se completaron siete de los siete puntos de historia asignados al *sprint*. Se recuerda que los puntos de historia no son una medida exacta, además de que el desarrollador está adquiriendo más destreza con las tecnologías que está utilizando.

Octavo *sprint*

Este *sprint* se dedicó a modificar la secciones de noticias y TV, para poder ver *posts* completos, extraídos directamente de la web de Nomepierdoniuna. De esta manera, estos *posts* que se leen en la aplicación generan visitas a sus correspondientes en la web. Esta implementación también se aplicará a la sección de la canción de la semana cuando llegue el momento. También se dedicó a mejorar la forma en la que se cargan los vídeos de la TV, ya que tal y como se había preparado podía ser lento en algunos dispositivos. También se había asignado la implementación de la canción de la semana a este *sprint*, pero no pudo ser realizada. Se completaron tres puntos de historia, correspondientes a la funcionalidad de ver *posts* completos, de los seis asignados, dado que se dedicó tiempo del *sprint* a mejorar una sección que ya se había implementado.

Hasta el momento, todas las librerías del proyecto estaban configuradas de manera que a la hora de resolver dependencias se buscase la versión más reciente. Esto dio lugar a que la versión montada en el ordenador de desarrollo a veces tuviera comportamientos diferentes en comparación con las versiones montadas en integración continua. Se descubrió cuando apareció algún *bug* causado por este asunto. Por ello, antes de terminar el proyecto, se debían dejar las dependencias con un número fijo de versión. En el Cuadro 3.6 se puede ver la pila del producto actualizada y con esta historia añadida.

Cuadro 3.6: *Pila del producto tras la reunión del diseño de la interfaz.*

ID	**Funcionalidad**	**PH**
F11	Sección canción de la semana.	3
F18	Actualizar y fijar versiones de las dependencias.	1

Noveno *sprint*

Durante este *sprint*, se implementó la sección de la canción de la semana. También se actualizó la versión de React Native del proyecto, y se actualizaron y fijaron las versiones del resto de dependencias. Una nueva versión de la librería de mapas solucionó bastantes de los problemas que tenía la anterior. A este *sprint* se le habían asignado los cuatro puntos de historia que había en la pila del producto, a la espera de tener los diseños de la interfaz de usuario.

Décimo *sprint*

En la última semana, se recibieron los diseños de interfaz de usuario de la aplicación. Por ello, se decidió hacer un sprint más para llegar hasta donde se pudiese. Los diseños están muy bien hechos, tanto en el sentido de moda de las aplicaciones como en el de usabilidad. El problema es que en ellos aparecen requisitos funcionales que no habían sido planeados hasta el momento.

Estos nuevos requisitos funcionales son, poder realizar búsquedas de eventos, noticias y puntos de interés por nombre, con sugerencias de búsqueda incluidas, poder tener también noticias favoritas, y poder ver los eventos correspondientes a un punto del mapa.

Se adaptaron las secciones de eventos, noticias, favoritos y mapa según los diseños. También se adaptó la imagen de compartir evento y los filtros de búsqueda. Se implementaron las nuevas funcionalidades de búsqueda de eventos y puntos del mapa por nombre, dejando el código preparado para, juntamente con la búsqueda de noticias, poder implementar más adelante la funcionalidad de ver sugerencias a medida que se escribe. También se implementó la funcionalidad de noticias favoritas. Por último, se implementó la funcionalidad de ver los eventos correspondientes a un punto del mapa, encontrando un problema de la librería de mapas en Android, que seguramente va a requerir una pequeña modificación del diseño de la interfaz que corresponde a esta sección. Para poder evaluar el estado del proyecto, en este punto se publicó otra versión beta.

En este *sprint* no se va a hablar de puntos de historia, dado que no ha habido tiempo de realizar planificaciones y se ha tratado de hacer lo máximo posible en poco tiempo. En el Capítulo 6, sobre las conclusiones del proyecto, se va a detallar esta situación. A continuación, en la sección de cierre del proyecto, se muestra también la pila del producto final.

3.4.2. Cierre del proyecto

La pila del proyecto al finalizar los *sprints* planificados se puede ver en el Cuadro 3.7. Habiendo terminado la estancia en prácticas, aún quedan historias por hacer antes de considerar terminada la aplicación. Parece que seguiré en la empresa, y se va a intentar que en un futuro cercano, la aplicación se termine y salga a producción.

Cuadro 3.7: *Pila del producto final.*

ID	**Funcionalidad**	**PH**
F19	Adaptar diseño de la TV.	3
F20	Adaptar diseño de la canción de la semana.	3
F21	Adaptar diseño de sugerir evento.	3
F22	Sugerencias de búsqueda por texto.	5

Capítulo 4

Análisis y diseño del sistema

Este capítulo corresponde al análisis de requisitos del proyecto y su diseño. En la primera sección se exponen los requisitos funcionales, con la derivación directa de historias de usuario a partir de casos de uso. También se explica cómo se han obtenido estos y se dedica un subapartado a mencionar otro tipo de requisitos del sistema. Para finalizar la sección, se muestra el modelo de datos de la aplicación.

La segunda sección ofrece un plano general de la arquitectura del sistema completo, donde se delimita y detalla la parte que corresponde a este proyecto. La última parte trata sobre la interfaz de usuario de la aplicación, describiendo su navegación y diseño.

4.1. Análisis del sistema

4.1.1. Requisitos funcionales

En la Figura 4.1 se pueden ver los casos de uso de la aplicación. Una parte de estos son herencia del sistema aniguo, otro subconjunto se ha obtenido a partir de entrevistas con el cliente al inicio del proyecto, y otros han ido apareciendo durante el desarrollo.

Los que estaban establecidos antes del inicio, considerados como básicos, son los de sugerencia de eventos, eventos favoritos, y consulta de la agenda, noticias y puntos de interés. En la primera reunión, se añadió el filtrado de eventos de la agenda, poder compartir eventos en forma de imagen y añadir las secciones de TV y canción de la semana. También se habló de añadir una sección con el perfil de Instagram de Nomepierdoniuna, pero finalmente quedó descartado.

En la segunda reunión, se decidió permitir también el filtrado de noticias y el resto de ampliaciones que corresponden al mapa. Por último, también se añadieron las de poder filtrar puntos del mapa, compartir noticias y poder añadir noticias como favoritas.

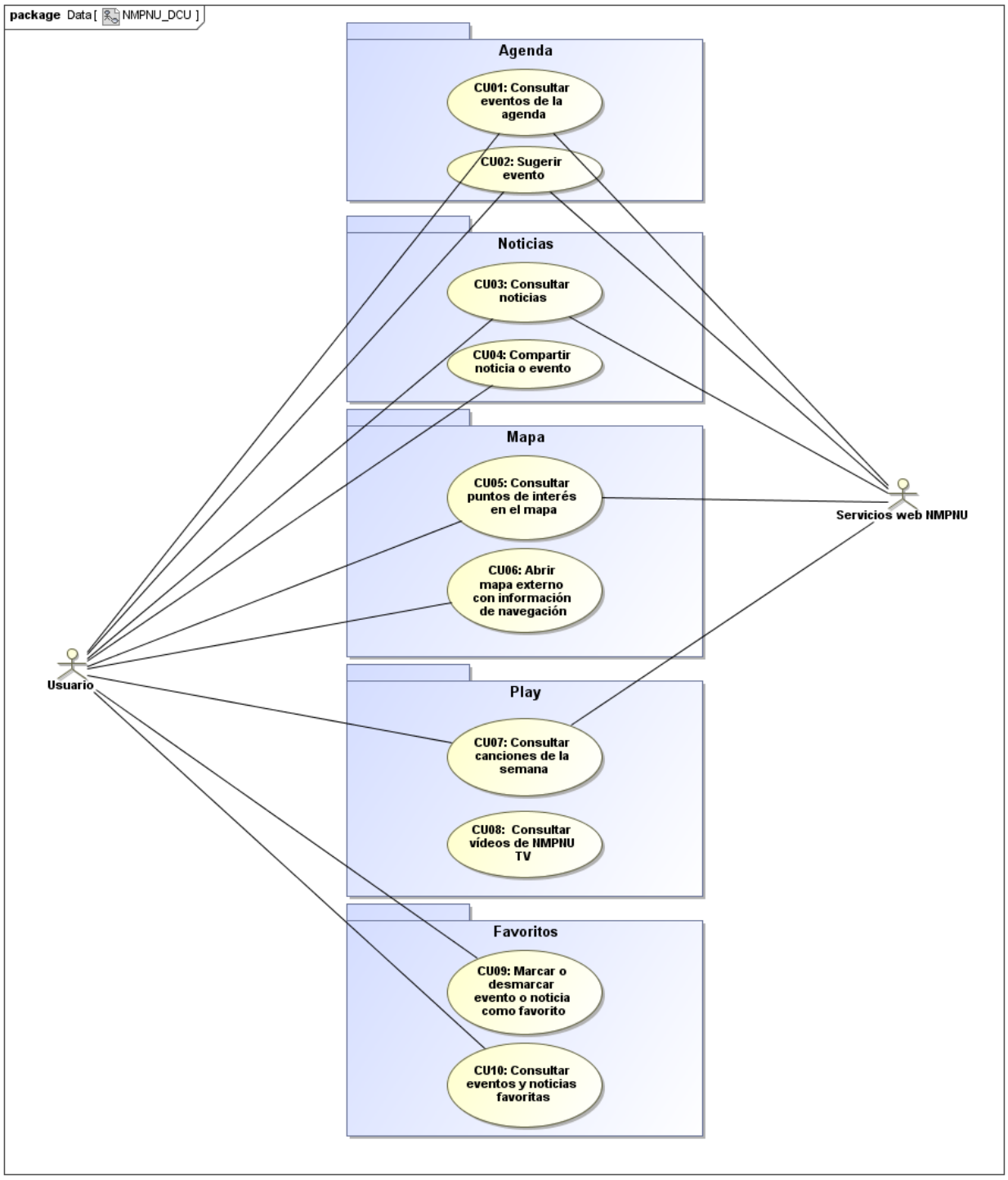


Figura 4.1: *Diagrama de casos de uso del proyecto.*

El diagrama de casos de uso se ha utilizado para ofrecer una idea visual de la funcionalidad de la aplicación. Como se ha seguido la metodología Scrum, en lugar de detallar estos, a continuación están definidas las historias de usuario, las cuales muestran la funcionalidad del sistema con más detalle.

Las historias de usuario resultantes se muestran en el Cuadro 4.1. En este caso solo hay un rol, el de usuario de la aplicación. Es necesaria la enumeración de estas, puesto que tienen un papel fundamental en la planificación de Scrum, la metodología que se ha utilizado. A parte, la definición de historias de usuario es una técnica habitual en el análisis de requisitos de proyectos de *software*, debido a que se pueden establecer y validar de manera natural y comprensible junto con el cliente.

Cuadro 4.1: *Historias de usuario.*

ID	**Rol**	**Funcionalidad**
HU01	Como usuario de la aplicación necesito	consultar los eventos de la agenda.
HU02		sugerir eventos para que sean publicados.
HU03		ver en un mapa dónde se va a realizar un evento.
HU04		consultar las noticias publicadas.
HU05		poder ver el *post* de una noticia, vídeo o canción.
HU06		compartir eventos en forma de imagen.
HU07		compartir noticias como enlaces a su *post* en la web.
HU08		consultar en un mapa los puntos de interés publicados, viendo además mi ubicación.
HU09		ver cómo llegar a un punto de interés desde mi ubicación.
HU10		ver los eventos que se van a realizar en un punto de interés.
HU11		filtrar los eventos, noticias y puntos de interes según una serie de criterios.
HU12		buscar por nombre entre los eventos, noticias y puntos del mapa, viendo además sugerencias de búsqueda.
HU13		consultar las canciones de la semana y vídeos publicados.
HU14		añadir eventos y noticias como favoritos.
HU15		ver los eventos y noticias que he guardado como favoritos.

A continuación, se dan algunos detalles a tener en cuenta de cada una de estas historias de usuario:

- **HU01:** solo se han de mostrar eventos que aún no han pasado.
- **HU02:** el usuario ha de introducir los datos del evento. Este se ha de enviar por correo a la administración de Nomepierdoniuna para su validación y publicación.
- **HU03:** no todos los eventos se realizan en un punto de interés registrado en el sistema. Los que lo tienen, lo referencian por el nombre y no escrito exactamente igual, dado que hasta el momento se han publicado a mano. Cuando no tienen, se ha de dar un aviso al usuario.
- **HU04:** a diferencia de los eventos, las noticias no tienen una fecha de expiración.
- **HU05:** las noticias, vídeos y canciones tienen un *post* asociado, al cual se ha de permitir acceder desde la aplicación. Este acceso ha de generar una visita al *post* correspondiente.

- **HU06:** los eventos no tienen ningún *post* ni vista asociada. Se han de poder compartir en forma de una imagen que simula ser una tarjeta, con la información del evento y el logo de Nomepierdoniuna.
- **HU07:** de las noticias, se ha de poder compartir el enlace a su *post* en la web.
- **HU08:** los puntos de interés publicados se han de poder consultar en un mapa. En este se ha de permitir ver la ubicación del usuario.
- **HU09:** se ha de permitir ver cómo llegar al punto utilizando una aplicación externa (e.g. Google Maps).
- **HU10:** misma situación que en HU03, se trata de la consulta inversa.
- **HU11:** los eventos se han de poder filtrar por fecha de inicio, fecha de fin, categoría, localidad y solo gratuitos. Las noticias se han de poder filtrar por categoría, con selección múltiple, y los puntos del mapa por tipo, también con selección múltiple.
- **HU12:** las búsquedas han de mostrar un listado de resultados a medida que se escribe. En el caso del mapa y las noticias, estas sugerencias han de llevar al punto en el mapa o al *post* de la noticia respectivamente. En el caso de los eventos, simplemente realizar la búsqueda.
- **HU13:** sin necesidad de ver el *post* correspondiente, se ha de permitir escuchar la canción de la semana o ver el vídeo en un reproductor.
- **HU14:** se ha de permitir seleccionar y deseleccionar eventos y noticias como favoritos, para permitir un acceso más rápido a estos.
- **HU15:** las noticias y eventos favoritos se han de poder consultar desde la sección de favoritos. Los eventos se han de eliminar cuando pasa su fecha.

4.1.2. Otros requisitos

Además de los requisitos dados por las necesidades funcionales de la aplicación, también se han definido otros con diversas motivaciones.

El primero de ellos, es poder utilizar la aplicación de forma *offline* en la medida de lo posible. Para ello, el dispositivo ha de mantener los últimos datos que reciba estando *online*, y funcionar con ellos en caso de que no haya conexión a internet. Las funcionalidades que en principio quedan fuera de este requisito son las de sugerir eventos, escuchar canciones y ver vídeos.

Dado que, exceptuando lo anterior, la aplicación no mantiene mucha información en el almacenamiento local, se ha visto viable implementar un sistema de almacenamiento intermedio para ofrecer una experiencia de uso más fluida.

Por último, la empresa ha requerido que la aplicación esté preparada para recibir notificaciones *push*, explicadas en la Sección 2.2.4. Con estas, se podrá notificar en cualquier momento a los usuarios con el mensaje que se decida para la situación.

4.1.3. Modelo de datos

La representación de los datos que requiere utilizar la aplicación se ha definido considerando un modelo no relacional. Como se ve en la sección siguiente, es cierto que los datos de Nomepierdoniuna están internamente organizados con un esquema de datos relacional. Sin embargo, la parte cliente trabaja solo con un subconjunto de las entidades del modelo de datos de la web, con posibles ampliaciones durante el desarrollo. Por ello, se ha considerado que sean tratados de esta manera para facilitar modificaciones durante el transcurso del proyecto. El esquema que se ha definido puede verse en la Figura 4.2.

```
"evento" : {
    "id",
    "título",
    "inicio",
    "fin",
    "lugar",
    "enlace",
    "idCategoría",
    "localidad",
    "precio"
}

"evento_sugerido": {
    "nombreRemitente",
    "emailRemitente",
    "título",
    "descripción",
    "fecha",
    "hora",
    "localidad",
    "idCategoría",
    "mapa": {
        "nombre",
        "latitud",
        "longitud"
    },
    "precio":
}

"noticia" : {
    "id",
    "título",
    "resumen",
    "fecha",
    "idCategoría",
    "enlace",
    "esPortada",
    "imagen"
}

"canción" : {
    "id",
    "título",
    "canción",
    "enlace",
    "imagen"
}

"vídeo" : {
    "id",
    "título",
    "vídeo",
    "enlace"
}

"punto_mapa" : {
    "id",
    "título",
    "descripción",
    "latitud",
    "longitud",
    "idCategoría"
}

"localidad" : {
    "id",
    "nombre"
}

"categoría_eventos" : {
    "id",
    "nombre"
}

"categoría_mapa" : {
    "id",
    "nombre"
}

"categoría_noticia" : {
    "id",
    "nombre"
}
```

Figura 4.2: *Esquema del modelo de datos.*

Algunas de las entidades de este modelo o sus atributos se han incorporado tras decidir añadir funcionalidades nuevas durante el desarrollo, como en el caso de las categorías de noticias. No ha sido posible incluir en la entidad evento un atributo que identifique de forma única el punto del mapa donde se va a realizar, puesto que el modelo de datos de Nomepierdoniuna no tiene establecida esta relación, y se ha considerado costoso añadirla.

En este esquema no aparece ningún elemento relacionado con los *posts* de noticias, canciones o vídeos. Esto se debe a que se extraen directamente de la web a partir del enlace, generando visitas a estos tal y como ha sido requerido.

Como último comentario, la entidad de evento sugerido está estructurada de manera que a los editores les sea directo validar y añadir la información de un evento, según el formulario con

el que realizan estas publicaciones.

4.2. Diseño de la arquitectura *software* del sistema

Para ofrecer una visión general del sistema, a continuación se explica la arquitectura *software* de Nomepierdoniuna. También se explica la parte servidora que da soporte a la aplicación que se ha desarrollado. No obstante, hay que recordar que este proyecto está restringido al desarrollo de la parte cliente de la aplicación, de la que se dan también detalles en esta sección. En la Figura 4.3 se pueden ver estos componentes y su relación. Se puede apreciar también que no se trata de una arquitectura con mucha complejidad.

La web de Nomepierdoniuna es un Wordpress, configurado para permitir a los editores publicar el contenido de la revista. Este utiliza internamente MySQL como sistema de gestión de bases de datos, para permitir su persistencia.

La parte servidora de esta aplicación consiste en un conjunto de servicios web implementados en PHP. Para acceder a los datos de Nomepierdoniuna, utilizan la librería MySQLi. Estos son los que ofrecen a la aplicación móvil la mayor parte de los datos que requiere, a excepción de los *posts* de noticias, canciones y vídeos, que son accedidos directamente de la web.

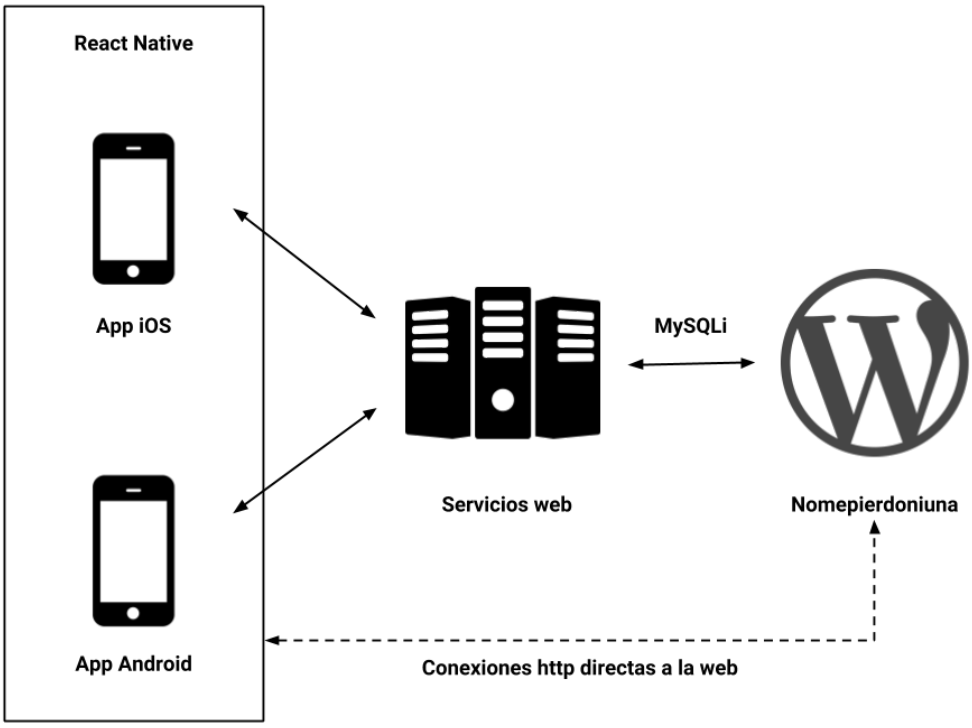


Figura 4.3: *Arquitectura software del sistema.*

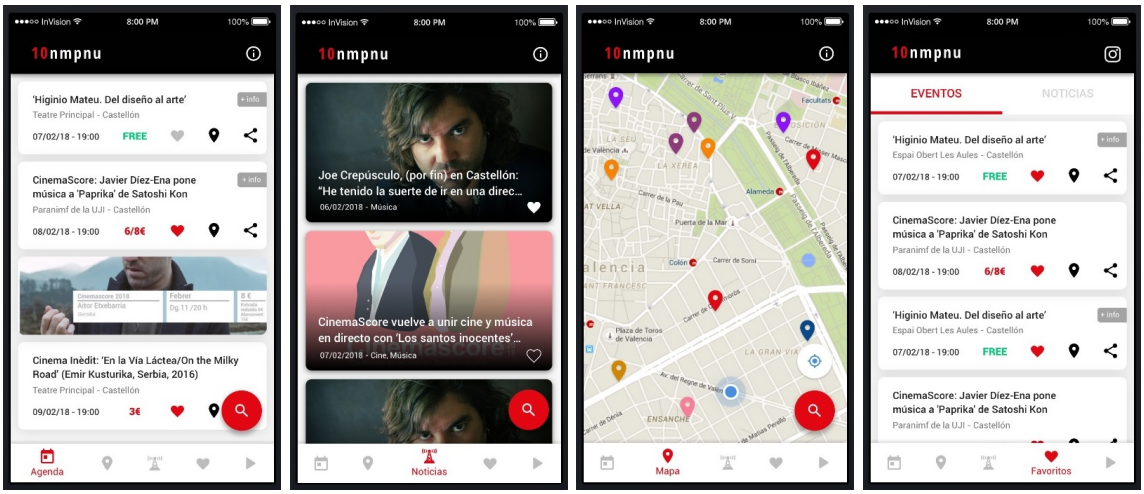
A pesar de que se muestran las aplicaciones como distintas para cada sistema, al utilizar React Native, estas han sido generadas a partir de un mismo código. Al no utilizarse cuentas

de usuario ni ningún sistema de autenticación, la gestión de favoritos se ha realizado con el uso del almacenamiento local del cliente (i.e. del dispositivo móvil).

4.3. Diseño de la interfaz de usuario

El diseño de la interfaz ha sido realizado por el equipo de diseño de Cuatroochenta. La navegación general se basa en un menú inferior desde donde se puede acceder a las principales secciones. En este documento se muestran solo las vistas más representativas.

La Figura 4.4 muestra los diseños de las secciones más importantes. En estos, también se puede ver el menú de navegación inferior como elemento más determinante en la navegación de la aplicación. La sección de favoritos consiste en dos subsecciones, navegables a través de un menú de pestañas superiores, para distinguir entre eventos y noticias. En la cabecera de todas ellas, se observa el nuevo logo de Nomepierdoniuna, con un 10 indicando que con esta aplicación se celebra su décimo aniversario.



(a) *Agenda* (b) *Noticias* (c) *Mapa* (d) *Favoritos*

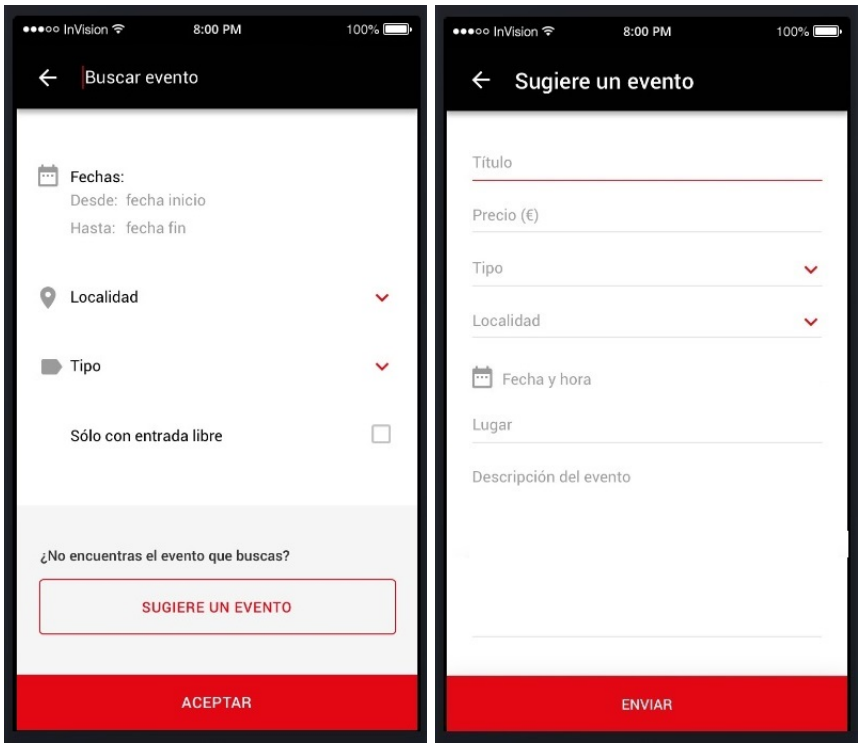
Figura 4.4: *Diseños de algunas secciones de la aplicación móvil.*

La imagen con apariencia de tarjeta que genera la aplicación para compartir eventos se muestra en la Figura 4.5. Esta proporciona la información más relevante del evento, acompañada del logo de Nomepierdoniuna.

En la Figura 4.6, se puede ver el diseño de la vista desde donde se introducen los filtros de la agenda de eventos. Las correspondientes para filtrar noticias y puntos del mapa tienen un diseño similar, y tienen en la cabecera un campo para realizar la búsqueda por texto. También se puede ver el formulario para sugerir eventos. Este representa la única parte de la aplicación que requiere entrada de datos por parte del usuario, sin tener en cuenta la búsqueda por texto.

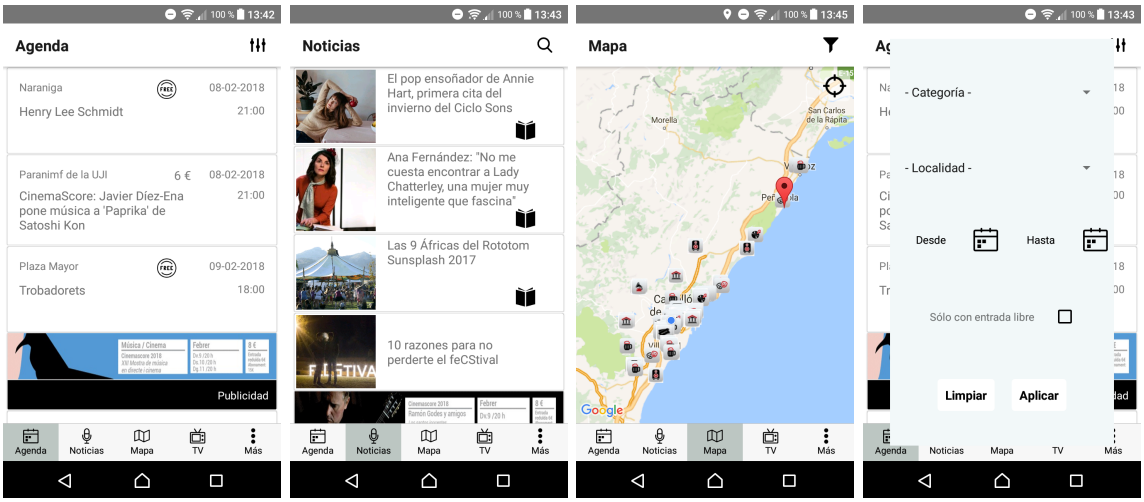
Mientras estos diseños se estaban preparando, el desarrollo se ha realizado con unos diseños más básicos, preparados para poder ser adaptados más adelante. En las Figuras 4.7 y 4.8 se pueden ver algunos de estos diseños provisionales.

Figura 4.5: *Imagen de evento para compartir.*



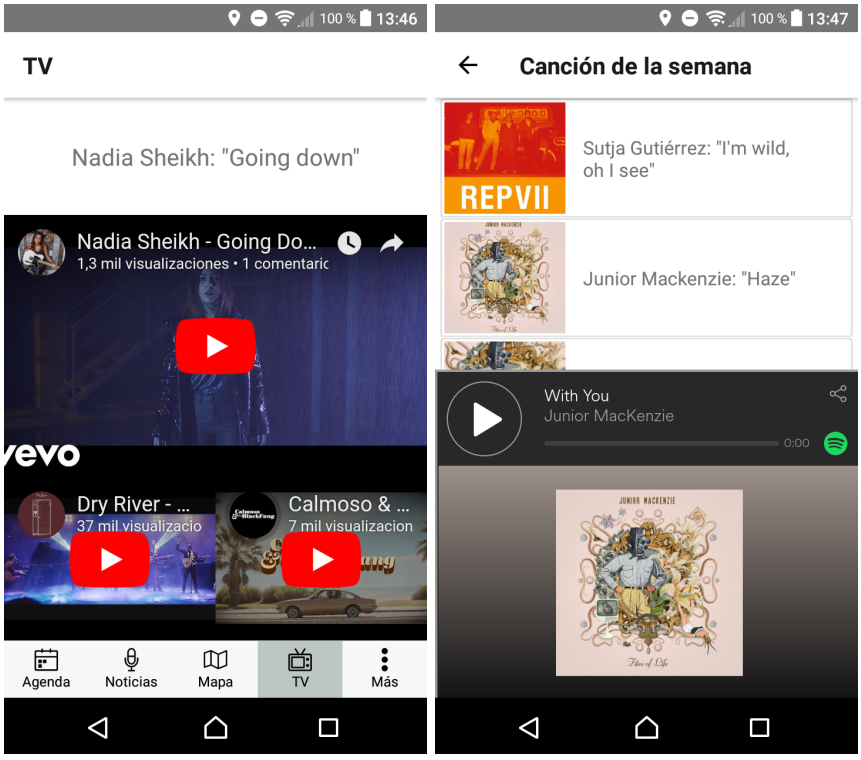
(a) *Filtros de la agenda* (b) *Formulario de sugerir evento*

Figura 4.6: *Otros diseños de la aplicación móvil.*



(a) *Agenda* (b) *Noticias* (c) *Mapa* (d) *Filtros de la agenda*

Figura 4.7: *Diseños durante el desarrollo de algunas secciones de la aplicación móvil.*



(a) *TV* (b) *Canción de la semana*

Figura 4.8: *Otros diseños durante el desarrollo de la aplicación móvil.*

Capítulo 5

Implementación y pruebas

En la mayor parte de este capítulo se detalla la implementación del sistema. En general, se trata de un cliente desarrollado en JavaScript con React Native, siguiendo el patrón de diseño que proporciona Redux.

Para empezar, se introduce la inicialización del proyecto y la organización de su estructura de directorios, detallando los que contienen el código fuente. A continuación, separados en subsecciones, se describen los elementos que constituyen el patrón de Redux (i.e. vista, *reducers* y *actions*). La vista está constituida por los componentes React, y también incluye la parte relacionada con la navegación de la interfaz de usuario. Para terminar, se explica también la gestión del almacenamiento local en el dispositivo, incluyendo los sistemas de caché implementados.

Durante este capítulo, se muestran algunos ejemplos de código, simplificados en la medida de lo posible y siempre para ilustrar cómo se ha resuelto un problema en concreto o alguna particularidad. Sobre todo, para los lectores familiarizados con JavaScript, ofrecen una idea de cómo es la sintaxis involucrada en un proyecto desarrollado con React Native.

La última parte del capítulo, más breve que la anterior, detalla las técnicas de verificación y validación que han sido utilizadas.

5.1. Detalles de implementación

Es necesario mencionar que JavaScript es un lenguaje débilmente tipado. Esto impide disponer de interfaces de programación, a pesar de que incluye soporte para clases y objetos. En el código desarrollado en este proyecto, han sido utilizados algunos patrones de diseño de software que son típicos en la programación orientada a objetos, pero que implementados en este sistema parecen no tener mucha fortaleza, dada la ausencia de interfaces y control de tipos. No obstante, su utilidad es la misma si el desarrollador tiene claro dónde y para qué los ha aplicado.

5.1.1. Estructura del directorio del proyecto

En este caso, el proyecto se ha inicializado creando un proyecto React Native desde Webstorm. También podría haberse hecho por línea de comandos utilizando el módulo de npm **react-native-cli**. La estructura de ficheros resultante, omitiendo algunos ficheros de configuración, es la siguiente:

```
nmpnu-reactnative/
|-- __tests__
|-- android
|-- app.json
|-- bitbucket-pipelines.yml
|-- index.js
|-- ios
|-- node_modules
|-- package-lock.json
|-- package.json
|-- src
```

Los directorios **android** e **ios** contienen los respectivos proyectos nativos de la aplicación, y el directorio **src** es el que contiene el código fuente de este proyecto. El directorio **__tests__** contiene los tests, explicados en la Sección de verificación y validación 5.2.

El directorio **node_modules** almacena los módulos npm que utiliza la aplicación, y la gestión y configuración general de estos está declarada en el fichero **package.json**. Cada vez que se añada un módulos npm al proyecto, también ha de ser añadido a este fichero con tal de dejar constancia de su uso.

Como se ha explicado en el apartado de tecnologías, React Native, al igual que React, se basa en la declaración de componentes. Para constituir la aplicación, tiene que haber un componente raíz desde el que surgen todos los demás. El fichero **index.js** es el encargado de registrar este componente como una aplicación, y lo hace de la siguiente manera, siendo App el componente mencionado:

```
1 import { AppRegistry } from 'react-native';
2 import App from './src/App';
3
4 AppRegistry.registerComponent('NMPNU', () => App);
```

La organización dentro del directorio **src** está determinada por el uso de Redux. Como se ha explicado en el apartado de tecnologías, esta librería consiste en ayudas para la implementación de un patrón de diseño, y como tal, este ha influido fuertemente en la organización del código de esta aplicación. El contenido del directorio src ha sido estructurado de la siguiente manera:

```
nmpnu-reactnative/src/
|-- App.js
```

```
|-- Config.js
|-- actions
|-- components
|-- notifications
|-- reducers
|-- res
|-- router
|-- storage
|-- util
```

El fichero **App.js** declara el componente raíz de la aplicación, mientras que el fichero **Config.js** contiene ciertas variables globales que, como se verá, configuran algunos aspectos de su funcionamiento.

Los directorios **actions** y **reducers** corresponden a los elementos *actions* y *reducers* de Redux. La vista está constituida por componentes declarados en el directorio **components**.

El directorio **router** también contiene algunos componentes, pero están separados de los anteriores porque se encargan de la navegación de la aplicación. El de **notifications** se explica más adelante, pero el nombre ya indica su propósito.

El directorio **util** contiene métodos útiles de uso común, mientras que **res** contiene recursos tales como colores, textos, direcciones url e iconos. En **storage** se encuentran los módulos para la gestión del almacenamiento de datos de la aplicación en el dispositivo.

5.1.2. Vista

Hablando de los componentes desde el punto de vista de la estructura de árbol que forman, antes de describir el componente raíz se explican los componentes que cuelgan directamente de él: el Router de navegación y la *Splash screen*.

Navegación

La navegación de la aplicación se ha implementado utilizando el módulo **react-native-router-flux**. Como muestra ilustrativa, a continuación se puede ver el comando de npm que instala el módulo y lo inscribe en el fichero **package.json**:

```
npm i react-native-router-flux --save
```

El uso de este módulo consiste en la implementación de un componente en el que se declaran las distintas escenas que tiene la aplicación, cada una asociada al componente que debe mostrar, y en el que se indica cómo ha de ser la navegación. En este caso, ha sido definido como una escena con vista de pestañas que contiene las escenas de las secciones principales. También se

han ido declarado en él el resto de escenas de la aplicación, pero la primera que se ve al montarse el componente es la escena con vista de pestañas.

Tras realizar varias refactorizaciones, finalmente ha quedado como un componente en el que, dadas las escenas que deben aparecer en la vista de pestañas y las que no, las sitúa en el lugar que les corresponde. Esto permite cambiar la distribución de las secciones de la aplicación de forma muy sencilla. También se ha refactorizado parte del código de la declaración de escenas, declarando una escena genérica de la aplicación.

A continuación, se muestra el código del componente dedicado a este propósito, omitiendo algunas partes que no son releventes en este apartado. De paso, se pueden ver los nombres de las escenas que han quedado finalmente en la aplicación, y parte de la sintaxis que involucra la creación de componentes. El método *render* es el que define cómo es el componente, mientras que los elementos Tabs, Router y Scene forman parte del módulo mencionado que se está utilizando.

```
import React, {Component} from 'react';
import {Router, Scene, Tabs} from 'react-native-router-flux';
// imports of scenes
// ...

export default class RouterNMUNU extends Component {

    scenes = [
        FiltersScene, PostDetailScene, PointAgendaScene, SuggestEventScene,
        AboutScene, WeeklySongScene, TVScene
    ];

    scenesInTabs = [
        AgendaScene, MapScene, NewsScene, FavouritesScene, PlayScene
    ];

    render() {
        const {routerProps, tabViewProps} = this.props;
        return (
            <Router
                {...routerProps}
            >
                <Scene>
                    <Tabs
                        {...tabViewProps}
                    >
                        {this.renderTabsScenes()}
                    </Tabs>
                    {this.renderScenes()}
                </Scene>
            </Router>
        );
    }
```

```
34
35      renderScenes = () => this.scenes.map(scene => scene());
36
37      renderTabsScenes = () =>
38          this.scenesInTabs.map((scene, index) => scene({initial: !index}));
39
40  }
```

La escena de las pestañas que está situada en primer lugar es la que se marca como inicial, siendo la primera que se muestra al iniciar la aplicación una vez se ha desvanecido la pantalla de carga. Actualmente se trata de la sección de la agenda.

En la Figura 5.1, se muestra un diagrama de los componentes que constituyen el Router. A efectos de este, todas las escenas, tanto las que se sitúan en la vista de pestañas como las que no, son tratadas como escenas genéricas.

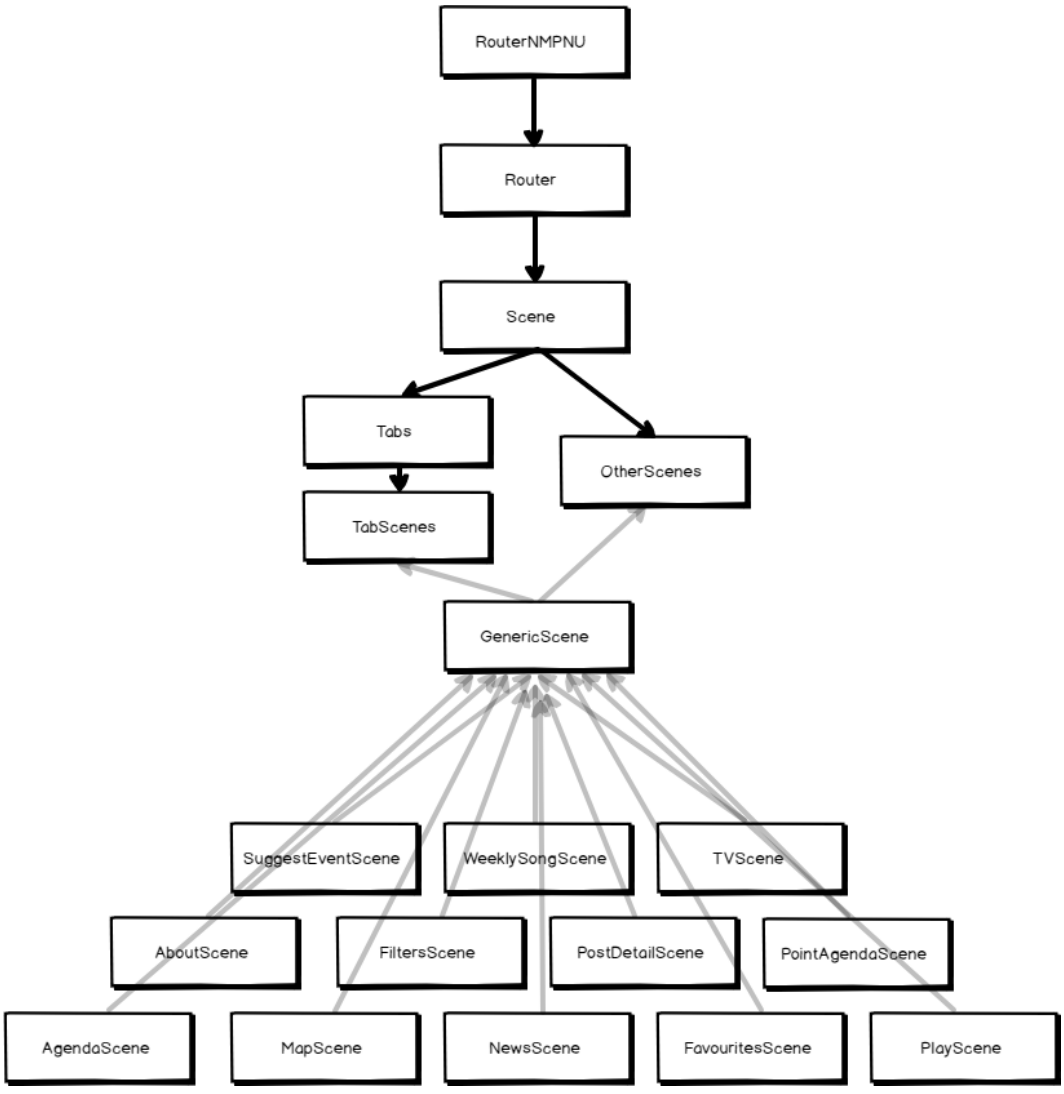


Figura 5.1: *Diagrama de los componentes que forman el Router.*

Splash screen

La *splash screen* es la pantalla inicial de la aplicación. Consiste en una pantalla en blanco con el logo de Nomepierdoniuna en el centro. Esta se aprovecha también para esperar a que se realicen ciertas cargas antes de mostrar la aplicación. Una vez estas cargas se han completado, la pantalla se desvanece con una transición a transparente, dando lugar a la vista de pestañas con la sección inicial.

La manera en la que este componente se da cuenta de que se han completado las cargas, es a través de Redux. No obstante, no se muestra esta lógica, dado que no es un ejemplo adecuado para mostrar el funcionamiento de Redux por primera vez. Se muestra un ejemplo de esto más adelante, aprovechando otro componente más simple.

La transición a transparente se ha implementado utilizando la API Animated de React Native. El siguiente método, toma el atributo opacidad del componente y lo hace variar gradualmente hasta 0, siguiendo una progresión lineal durante 500 segundos. Al finalizar, indica al componente que la transición ha terminado.

```
animatedHideSplashScreen() {
    Animated.timing(this.state.visibility, {
        toValue: 0,
        easing: Easing.linear,
        duration: 500
    }).start(() => this.setState({finished: true}));
}
```

Componente raíz

El componente raíz de esta aplicación está compuesto por los dos componentes mencionados antes: la *splash screen* y el Router de navegación. También se inicializan los elementos necesarios para utilizar Redux. Para ello, los componentes contenidos en este componente raíz han de estar envueltos en otro componente del paquete Redux, llamado Provider. A este se le ha de pasar como *property* (i.e. parámetro), una instancia de Store, objeto del paquete encargado de almacenar el estado de la aplicación, al que a su vez hay que pasarle como parámetro los *reducers* que se han implementado, explicados más adelante en la sección correspondiente.

Los componentes React tienen un ciclo de vida, que permite responder a eventos como el montado o desmontado del componente. La inicialización del Store se realiza en el método que corresponde al evento previo al montado del componente. Se aprovecha también para cargar la publicidad de la aplicación, en el caso de que esta opción se encuentre activada. Este tipo de configuraciones se permiten modificar en un fichero del proyecto, llamado **Config.js**.

A continuación se muestra el código de este componente, simplificado en cuanto a algunos aspectos que en este punto no han sido mencionados:

```
// React and redux imports
```

```
// ...
import {ADVERTISING_ENABLED} from './Config';
import Router from './router/Router';
import SplashScreen from './components/SplashScreen';

export default class App extends Component {
    componentWillMount() {
        this.store = createStore(reducers);

        if (ADVERTISING_ENABLED)
            advertsFetch()(this.store.dispatch);
    }

    render() {
        return (
            <Provider store={this.store}>
                <View>
                    <Router advertisingEnabled={ADVERTISING_ENABLED}/>
                    <SplashScreen/>
                </View>
            </Provider>
        );
    }
}
```

Publicidad

La manera en la que la aplicación antigua mostraba la publicidad de Nomepierdoniuna era similar a la que se utiliza en la web, mostrando un *banner* que está en todo momento en la parte superior de la pantalla. Para hacerlo de una manera más acorde con las aplicaciones actuales, se ha decidido mostrar la publicidad entre las listas de la aplicación.

La mayoría de secciones de esta aplicación consisten en una *scroll view* en la que se muestra un listado de elementos (e.g. la agenda de eventos o la sección de noticias). Para adoptar este nuevo sistema de publicidad, se ha implementado una extensión del componente de listas que ofrece React Native, que inyecta entre los elementos de esta bloques publicitarios. En ella se puede configurar cada cuántos elementos debe poner publicidad.

Este componente ejecuta el siguiente método sobre los datos que tiene que mostrar, de manera que introduce entre ellos la publicidad aleatoriamente, repitiéndose lo menos posible y espaciada según se le indica con un parámetro:

```
injectAdvertsOnData({data, spaceBetween}) {
    return _.reduce(data,
        this.injectAdvertsReducer(this.adverts, spaceBetween), []);
}
```

```
5
6  injectAdvertsReducer(adverts, spaceBetween) {
7      let i = -1;
8      let shuffledAdverts = [];
9
10     return (accumulatedItems, current) => {
11         if (shuffledAdverts.length === 0)
12             shuffledAdverts = _.shuffle(adverts);
13
14         if (i === spaceBetween - 1)
15             accumulatedItems.push({
16                 ...shuffledAdverts.shift(),
17                 id: 'advert' + accumulatedItems.length
18             });
19
20         accumulatedItems.push(current);
21         i = (i + 1) % (spaceBetween);
22         return accumulatedItems;
23     };
24 }
```

Agenda de eventos

Sobre el componente anterior, se ha implementado el componente de la agenda de eventos, cuyo estado es mantenido por Redux. Este solicita al montarse los eventos al servidor, a través de la *action* correspondiente, y cuando los recibe los muestra en la lista con el formato adecuado. Es actualizable deslizando el dedo hacia abajo, lo que produce que se vuelva a realizar la petición de eventos de la agenda al servidor. Uno de los botones de cada evento en la lista, es el que permite compartirlo en forma de imagen.

Para poder crear la imagen, se ha utilizado el módulo **react-native-view-shot**. Este permite hacer capturas de componentes React Native. Al pulsar en el botón de compartir, se oscurece la aplicación y se muestra un modal con la información del evento en él. Este modal tiene el aspecto de la imagen que se tiene que compartir, y con el módulo mencionado antes se realiza una captura sobre él.

Una vez obtenida la imagen, esta se comparte mediante aplicaciones externas gracias al módulo **react-native-share**, que permite enviar imágenes en formato binario base 64. También permite enviar texto junto con la imagen, y esto se ha aprovechado para compartirla junto con el enlace de descarga de la aplicación, que provisionalmente es el de la versión beta.

A continuación se muestra una versión simplificada del código de este componente, donde se ve por una parte el uso del componente de lista con publicidad, y por otra la sintaxis que involucra la declaración de componentes conectados con Redux:

```
1  import {connect} from 'react-redux';
2  // Imports from React, React Native, subcomponents and actions
```

```
// ...

class AgendaList extends Component {
    componentWillMount() {
        this.props.agendaCategoriesFetch();
        this.props.agendaLocalitiesFetch();
    }

    render() {
        const {refreshingData, agendaData, agendaFetch} = this.props;

        return (
            <View style={{flex: 1}}>
                <AdvertisingRefreshList
                    refreshing={refreshingData}
                    data={agendaData}
                    fetch={agendaFetch}
                    spaceBetween={3}
                {/*other list props ...*/}
                />
                <AgendaFiltersButton/>
            </View>
        );
    }
}

const mapStateToProps = ({agenda}) => ({
    agendaData: agenda.agendaData,
    refreshingData: agenda.refreshingData
});

export default connect(mapStateToProps,
    {agendaFetch, agendaCategoriesFetch, agendaLocalitiesFetch})(AgendaList);
```

Noticias

Al igual que el componente de la agenda de eventos, el de noticias también se ha implementado sobre la lista con publicidad. También de la misma manera, este componente solicita al servidor las noticias de Nomepierdoniuna, y una vez las recibe las muestra, permitiendo actualizar deslizando el dedo hacia abajo.

El diseño de los elementos de la lista de noticias requería que la imagen de la noticia estuviese de fondo, con el título por encima en la parte inferior, como se puede ver en la subfigura *b* de la Figura 4.4. Sin hacer nada adicional, el texto puede no verse bien según los colores de fondo de la imagen. Una técnica habitual para solucionar esto consiste en poner el texto sobre una capa oscura transparente. Para disimularlo, se suele hacer que esta capa tenga una transparencia gradual, que tiene un valor pequeño en la parte inferior de la imagen y se vuelve completamente

transparente al llegar aproximadamente a la mitad de esta.

React Native no ofrece la posibilidad de crear transparencias u otros atributos en forma degradada, como sí es habitual ver en el css de diseño web. Para simular esto, se ha implementado un componente que crea una serie de vistas apiladas, cada una más transparente que la anterior. Cuantas más vistas utilizadas, más aparenta ser un degradado continuo. Experimentalmente se ha observado que 40 es un buen valor para este caso. El código de este componente que simula transparencia degradada es el siguiente:

```
// React and React Native imports
// ...

export const GradientOverlay = ({
    startOpacity = 0.6,
    nodes = 40,
    color = 'black',
    height = 70}) => (
        (Array.apply(null, Array(nodes))).map((per, i) => <View key={i} style={{
            backgroundColor: color,
            borderWidth: 0,
            borderColor: 'transparent',
            opacity: startOpacity * (1 - ((i + 1) * (1 / nodes))),
            position: 'absolute',
            bottom: `${i * height / nodes}%`,
            left: 0,
            width: '100%',
            height: `${height / nodes}%`
        }}/>)
    );
```

Mapa

El mapa de puntos de interés se ha implementado con el módulo **react-native-maps**. Un problema que tiene actualmente es que hay muchos puntos de interés, que provocan que la interfaz gráfica se quede congelada un instante cuando se renderizan estos en el mapa. Más adelante se debería estudiar cómo mejorar esto, pero por ahora se ha puesto énfasis en que el filtrado de puntos del mapa se haga de manera eficiente.

Dada la forma en la que funciona este mapa, el enfoque más cómodo al realizar el renderizado de puntos de interés, era volver a renderizar el mapa completo, con solo los puntos seleccionados por el filtro. Una mejora consistiría en mantener el mapa, borrar todos los puntos y volver a renderizarlos, dejando solo los seleccionados por el filtro. La mejor opción, aunque más complicada de implementar, y la utilizada en la aplicación actualmente, consiste en borrar solo los puntos que no se deban mostrar, y renderizar solo los que antes no estaban seleccionados pero ahora sí.

Este módulo ha dado bastantes problemas en cuanto a comportarse distinto en un sistema

u otro, y por ello es el que ha requerido más código específico de plataforma (e.g. al realizar la transición para centrar la ubicación, o al mostrar programáticamente el *callout* de un punto). A continuación se muestra un ejemplo de código específico de plataforma para iOS:

```
import {Platform} from 'react-native';

// ...

// solving programmatically press problem
if (Platform.OS === 'ios' && this.programmaticallyPressedMarker)
    this.registeredMarkers[this.programmaticallyPressedMarker]
        .hideCallout();
// ...
```

Favoritos

El componente de favoritos es una vista con dos pestañas, una para los favoritos de la agenda de eventos y otra para las noticias favoritas. Cada una de estas reutiliza el mismo componente utilizado para las secciones de agenda y noticias, pero solo mostrando los elementos que son favoritos, y sin mostrar publicidad entre ellos. Es decir, se ha implementado reciclando otros componentes de la aplicación, que ya se habían implementado de manera configurable para esto.

Un problema encontrado, es que la sección ya se encuentra dentro de una pestaña de la aplicación, y la librería de navegación utilizada funciona mal si se intentan anidar pestañas. Para implementar esta sección, se ha utilizado otro módulo específico que ofrece una vista de pestañas simple: **react-native-tab-view**.

Sugerir evento

Esta sección es la única, a excepción de las búsquedas por texto, que tiene que tratar con entradas de datos del usuario. El componente es un formulario, diseñado para no permitir introducir datos inválidos en algunos campos (e.g. uso de *date picker* para introducir fechas) y validadores en otros (e.g. email).

Un problema que se ha encontrado durante el desarrollo, es que el teclado de iOS en los componentes de entrada de texto de React Native no esconde el teclado automáticamente al tocar fuera del campo de texto, como suele ser habitual. Por ello, se ha tenido que dedicar tiempo a conseguir que este componente se comporte de una manera normal, desde el punto de vista de la usabilidad.

Posts

Para poder mostrar de la mejor manera posible los *posts* de noticias, canciones de la semana y vídeos, al mismo tiempo que se generan visitas a sus correspondientes en la web, estos se han implementado con un componente web, que muestra el *post* original.

Para ello, se descarga el código html del *post* en la web a partir de su enlace, y se transforma para mostrar solo el artículo. A continuación se muestra el código que adapta el html de la web a esta aplicación:

```
export const postExtractor = (html) => html
    // Removes header
    .replace(/<div id="header1">[\s\S]*<div id="main"/,
        '<div id=\"main\"')
    // Removes a section division bar
    .replace(/<div id="header_barra_subvideos[\s\S]*<div id="primary"/,
        '<div id=\"primary\"')
    // Removes padding top
    .replace(/id="page" class="hfeed" style="padding-top:20px;"/,
        'id=\"page\"')
    // Removes secondary sections
    .replace(/<!-- #prim[\s\S]*<!-- #sec/, '<!-- #sec')
    // Removes footer
    .replace(/<div id="pie[\s\S]*<script/, '<script')
    // Removes adds panel
    .replace(/<ins class="adsby[\s\S]*<div id="comments/,
        '<div id="comments')
    // Removes comments
    .replace(/<div id="comments"[\s\S]*<!-- \.entry-content/,
        '<\/div><!-- .entry-content')
    // Removes nav state title
    .replace(/"page-title"[\s\S]*<article/,
        '\"page-title\"></h1><article')
    // Removes 'show in map' (If any)
    .replace(/<div id="entry-map-position[\s\S]*<div id="entry-social/,
        '<div id=\"entry-social\"')
    // Removes related news, social and tags
    // (This must be the last step or will fail when comments)
    .replace(/<div id="entry-social[\s\S]*<div id="entry_col2"/,
        '<\/div><div id=\"entry_col2\"');
```

Filtros y búsqueda

Las secciones de agenda, noticias y mapa permiten filtrar su contenido según unos criterios. El componente que se ha implementado para esto se ha refactorizado en la medida de lo posible para poder ser compartido por las tres secciones. A la vista de filtros se accede desde cada

sección a través de un botón flotante, y según desde la sección desde la que se haga, el panel de filtros que se muestra es el que corresponde. Hablando en términos de Redux, al introducir los criterios de filtrado y pulsar el botón de aceptar, se ejecuta una acción que hace cambiar el estado de la aplicación para que los datos se muestren filtrados.

En la parte superior de la vista de filtros, hay un campo de texto que permite realizar búsquedas por texto. Su implementación se ha realizado con una idea similar a la del filtrado, pero aún no se ha desarrollado la parte que muestra sugerencias de búsqueda a medida que se escribe. Sin embargo, se ha dejado el código preparado para que sea sencillo de implementar.

Notificaciones *push*

Para que la aplicación pueda recibir notificaciones *push*, se ha utilizado el módulo **react-native-onesignal**. Este ofrece integración con los servicios de notificaciones de OneSignal. Además, se ha reciclado código proporcionado de un proyecto hecho anteriormente en Cuatroochenta. Con este, simplemente basta con realizar su inicialización antes de que se monte el componente raíz.

Más adelante, si se quiere programar reacciones concretas a ciertas notificaciones, el código correspondiente debe escribirse en uno de los métodos del proyecto. Por ahora no ha sido necesario implementar ninguna.

5.1.3. *Actions*

Como se ha explicado en la Sección 2.2.3 sobre Redux, siguiendo este patrón, las *actions* son las acciones que pueden hacer que el estado de la aplicación cambie, y realmente las únicas con las que se ha de poder hacer. De otra forma, se estaría actuando en contra de lo que el patrón pretende lograr.

Para evitar la aparición de errores al escribir el identificador de las acciones, en primer lugar se han definido las variables que identifican a cada una de ellas en el fichero **types.js**, dentro del subdirectorio dedicado a las *actions*. En este caso, se ha asignado a cada una su propio nombre como identificador. Así quedan algunas de ellas:

```
1 export const NEWS_FETCH = 'news_fetch';
2 export const NEWS_FETCH_SUCCESS = 'news_fetch_success';
3 export const SUGGEST_EVENT_SEND = 'suggest_event_send';
4 export const SUGGEST_EVENT_CLEAR_DATA = 'suggest_event_clear_data';
```

Las dos primeras corresponden a las acciones de solicitar los datos de las noticias. Al ser un evento asíncrono, una para realizar la petición y otra para cuando se hayan recibido los datos. La siguiente corresponde a la acción de enviar un evento sugerido a partir de los datos del formulario, y la última, a la limpieza del contenido de este formulario tras enviar o cancelar.

Las acciones son un par con el identificador de la acción, que indica qué se debe hacer, y una *payload*, que ha de contener lo que se necesite para realizar la acción. En Redux, estas se crean a partir de *function creators*, funciones que devuelven acciones. Por medio de un *dispatcher*, las acciones son enviadas a los *reducers* que hay implementados, siendo alguno de ellos el que tomará la acción y sabrá qué hacer con ella.

La parte correspondiente a los *dispatchers* es transparente al desarrollador, puesto que tras conectar un componente con Redux y ofrecerle los *function creators* correspondientes, como se ha visto en la sección anterior, bastará con llamar a la función desde el mismo componente. A continuación, se muestra la implementación de dos de ellas, una muy simple y otra menos simple:

```
1  import {SUGGEST_EVENT_CLEAR_DATA} from './types';
2
3  export const suggestEventClearData = () => ({type: SUGGEST_EVENT_CLEAR_DATA});
4
5  export const searchByText = (action, text) => ({type: action, payload: text});
```

La primera de ellas, indica que se ha de limpiar el contenido del formulario de sugerir evento, y no requiere *payload*. La segunda de ellas es para realizar las búsquedas por texto. Realizará la búsqueda en una sección de la aplicación u otra en función del parámetro *action*, con el texto *text* que se incluye como *payload*.

Para poder delegar a las *actions* la lógica de solicitar datos de forma asíncrona, se necesita agregar un módulo específico, llamado Redux Thunk. Este ha de ser añadido al Store durante su inicialización, y permite declarar *function creators* que ejecuten acciones de forma asíncrona, como respuesta a la espera de algún evento.

Este es el caso de las peticiones de datos a los servicios web (e.g. al pedir las noticias de Nompierdoniuna). Tras realizar una refactorización de código, puesto que se han definido bastantes de estas acciones con código común, el código ha quedado de la siguiente manera, permitiendo crear cualquier acción de este tipo a partir de los nombres de la acción de inicio y fin, el nombre del endpoint, y opcionalmente una función preproceso a aplicar a los datos recibidos:

```
1  // Utils and networking imports ...
2  // ...
3  import {NEWS_CATEGORIES_FETCH_SUCCESS} from './types';
4  import {NEWS_CATEGORIES_ENDPOINT} from '../res/endpoints';
5  import {cacheStorage} from '../storage';
6
7  export const newsCategoriesFetch = () =>
8      fetchDispatcher(null, NEWS_CATEGORIES_FETCH_SUCCESS, NEWS_CATEGORIES_ENDPOINT);
9
10 // Other fetch action creators like newsCategoriesFetch
11 // ...
12
```

```
13 const fetchDispatcher = (action, successAction, endpoint, preprocess = idFunction) =>
14     (dispatch) => {
15         dispatchStartingFetchSignal(action, dispatch);
16         offlineCaseFetch(successAction, endpoint, dispatch, preprocess);
17         makeFetch(successAction, endpoint, dispatch, preprocess)
18     };
19
20 const dispatchStartingFetchSignal = (action, dispatch) =>
21     action ? dispatch({type: action}) : null;
22
23 const offlineCaseFetch = (successAction, endpoint, dispatch, preprocess) =>
24     isNetworkConnected().then(isConnected =>
25             !isConnected ?
26                 fetchFromCache(successAction, endpoint, dispatch, preprocess)
27             : null
28     );
29
30 const fetchFromCache = (successAction, endpoint, dispatch, preprocess) =>
31     cacheStorage.getCache(endpoint, data => {
32         if (!_.size(data))
33             toastMessage(NETWORK_ERROR_MESSAGE);
34         dispatch({type: successAction, payload: preprocess(data)});
35     });
36
37 const makeFetch = (successAction, endpoint, dispatch, preprocess) =>
38     axios.get(endpoint + queryJSON)
39     .then(onFetchResponse(successAction, endpoint, dispatch, preprocess))
40     .catch((err) => console.log(err));
41
42 const onFetchResponse = (successAction, endpoint, dispatch, preprocess) =>
43     ({data}) => {
44         dispatch({type: successAction, payload: preprocess(data)});
45         cacheStorage.setCache(endpoint, data);
46     };
```

En este ejemplo, el *action creator* en cuestión es el método *newsCategoriesFetch*. Gracias a la refactorización realizada, todos los demás action creators de petición de datos en la aplicación se han declarado en una sola instrucción, de manera similar a este.

En el código se puede ver involucrada la caché *offline*, aunque esto se explica más adelante, en la sección sobre el almacenamiento local.

5.1.4. *Reducers*

Otro tipo de elementos en este patrón, los *reducers*, son los encargados de crear el nuevo estado de la aplicación, a partir del anterior y la solicitud de realizar una acción, y enviar este

nuevo a la vista. Para ello, los componentes que necesitan conocer algo del estado mantenido por Redux, son inicializados específicamente para recibir estos datos en sus *properties*. En la sección sobre la vista, se ha visto algún ejemplo de la sintaxis correspondiente a esto al mostrar la declaración de componentes conectados con Redux.

El componente en cuestión, recibe el nuevo estado proporcionado por los *reducers*, y es actualizado cuando esto ocurre, siempre que el estado recibido sea distinto de como era anteriormente. En el siguiente código se muestra un ejemplo de *reducer*:

```
1  import {MAP_FETCH_SUCCESS, MARKERS_TEXT_SEARCH} from '../actions';
2  import {byTitleSearch} from '../util';
3
4  const INITIAL_STATE = {
5      mapFullPoints: [],
6      visiblePoints: [],
7      mapCategories: [],
8      searchText: ''
9  };
10
11 export default MapReducer = (state = INITIAL_STATE, action) => {
12     const {type, payload} = action;
13
14     switch (type) {
15         case MAP_FETCH_SUCCESS:
16             const {categories, points} = payload;
17             return {
18                 mapFullPoints: points,
19                 visiblePoints: points,
20                 mapCategories: categories,
21                 searchText: ''
22             };
23         case MARKERS_TEXT_SEARCH:
24             return {
25                 ...state,
26                 visiblePoints: byTitleSearch(state.mapFullPoints, payload, 'titulo'),
27                 searchText: payload
28             };
29         default:
30             return state;
31     }
32 }
```

Se trata del *reducer* correspondiente a los puntos de interés de Nomepierdoniuna. En este, el estado consiste en la lista completa de puntos, la lista de puntos visibles, la lista de categorías de puntos y un texto de búsqueda. Inicialmente, son *arrays* vacíos o cadena vacías, según corresponda. Una de las acciones a las que responde, es a la de haber recibido datos del servidor. El nuevo estado consistirá en los puntos y categorías recibidos en la *payload*. Otra, es la de

búsqueda por texto de entre los puntos de interés. En este caso, el nuevo estado consistirá en los puntos que tengan un título parecido al texto de búsqueda, *payload*, manteniendo los puntos totales almacenados para no impedir realizar otra búsqueda después.

Lo habitual es tener varios *reducers*, cada uno con responsabilidad sobre una parte del estado concreta. Estos se pueden combinar, utilizando la API de Redux, en un único *reducer* que los agrupa. Este último, cuyo código simplificado se muestra a continuación, es el que hay que proporcionar al Store de Redux durante su inicialización, como se ha mostrado en la sección del componente raíz:

```
// Import reducers
// ...

export default combineReducers({
    mapContent: MapReducer,
    // Other reducers
    // ...
});
```

5.1.5. Almacenamiento local

En la primera parte de esta sección, se explica el almacenamiento de datos de la aplicación propiamente dicho. La segunda corresponde también al almacenamiento, pero con otro propósito: crear un sistema de caché *offline*.

Por último, se explica un sistema intermedio de almacenamiento que ha sido implementado con el objetivo de tener una mayor fluidez en la experiencia del uso de la aplicación.

Almacenamiento de datos

La persistencia de eventos y noticias favoritas se ha implementado en el almacenamiento local del dispositivo. La API de React Native permite gestionar esta de forma sencilla, a través del objeto AsyncStorage. Para ello, dado un identificador que se haya elegido para almacenar unos datos concretos, permite realizar sobre él una serie de operaciones básicas de tipo *get*, *set* y *remove*, devolviendo además el resultado de forma asíncrona a través de un objeto Promise. También ofrece algunos métodos más específicos como *merge*, cuyo propósito es fusionar los elementos de un par clave valor con los de otro almacenado. Ante las claves duplicadas, se mantienen las del nuevo. A continuación se puede ver un ejemplo de operación *get*, en un método que se ha definido para obtener los eventos y noticias favoritas según el valor de la variable STORAGE_LABEL:

```
const getFavourites = (STORAGE_LABEL) => (then) => {
    AsyncStorage.getItem(STORAGE_LABEL, (err, result) => {
        if (err)
```

```
4               console.log(err);
5           else
6               then(result ? JSON.parse(result) : Object.create(null));
7       });
8   };
```

La forma de almacenar estos datos se ha realizado con una idea distinta según si se trata de noticias o eventos. En el caso de las noticias, simplemente se almacena el objeto entero, y siempre se mantiene en memoria hasta que se desmarca como favorito. El caso de los eventos de la agenda tiene una particularidad, y es que están constantemente actualizándose en la web, sobre todo por que una vez ha pasado su fecha, han de ser eliminados.

Para ello, se ha optado por almacenar de los eventos favoritos solo el identificador y, cada vez que se reciben datos actualizados de los eventos, estos son marcados según los identificadores almacenados. A pesar de que no tendría ningún efecto visible, con el paso del tiempo habría una gran cantidad de identificadores de eventos pasados en la memoria local. Por eficiencia, al recibir datos actualizados, se sustituye el contenido del almacén de eventos favoritos por la intersección de identificadores de entre los eventos recibidos y los favoritos. A continuación, se muestra el pseudocódigo de este proceso:

Data: ids de eventos favoritos, lista de eventos actualizada
Result: nuevos ids de eventos favoritos, lista de eventos marcada

```
1 for evento en lista de eventos actualizada do
2 |   if id de evento en ids de eventos favoritos then
3 |   |   marcar evento como favorito;
4 |   |   añadir id a nuevos ids de eventos favoritos;
5 |   end
6 end
```

Este proceso se realiza cada vez que se reciben datos actualizados de la agenda de eventos, y permite dos cosas. Por una parte, sincronizar el almacenamiento local de eventos favoritos con los eventos que hay en Nomepierdoniuna. Por la otra, añadir a los datos externos recibidos la información sobre si son favoritos o no. En la implementación, la comprobación de si un id pertenece a los ids de eventos favoritos tiene un coste $\mathcal{O}(1)$, y por ello el algoritmo tiene un coste temporal $\mathcal{O}(n)$.

Caché *offline*

Como se ha visto en el apartado *Actions*, una gran parte de los datos que utiliza la aplicación se obtienen con peticiones a servicios web, cada uno con su *endpoint*. Para permitir disponer de datos cuando no hay conexión a internet, se ha implementado un sistema caché de estos en el almacenamiento local del dispositivo.

Cada vez que se reciben datos actualizados de algún servicio, se almacena en este sistema un par con el nombre del *endpoint* como clave, y los datos recibidos como valor. Esto permite que, en el caso de no disponer de conexión a internet, las peticiones se hagan a este sistema

caché, y no al *endpoint* original. NetInfo un objeto de la API de React Native, es el que permite conocer el estado de la conexión a internet. Ha dado algunos problemas, dado que, por lo menos actualmente, no ofrece un comportamiento similar en iOS y Android. Esto ha causado confusión y aparición de *bugs* durante el desarrollo, pero finalmente ha sido solucionado con código específico de plataforma. La idea general es la siguiente:

```
   Data: endpoint
   Result: datos
 1 if hay conexión a internet then
 2 |  solicitar datos al endpoint real;
 3 |  almacenar datos en la caché de este endpoint y devolverlos;
 4 else
 5 |  if hay datos en caché then
 6 |  |  devolver datos de la caché;
 7 |  else
 8 |  |  avisar de que no hay datos;
 9 |  end
10 end
```

A pesar de la aparente sencillez, hay que tener en cuenta que en la realidad tanto la comprobación de la conexión como la petición de datos al servidor son eventos asíncronos. La solución que se ha dado en este caso, ha sido lanzar al mismo tiempo la petición sobre el estado de la red y la petición de datos al servidor. Una vez se resuelve la del estado de la red, si hay conexión no se hace nada, y si no la hay se solicitan los datos a la caché. En el caso de que no haya, la resolución de la petición al servidor simplemente devolverá un error y tampoco se hará nada, pero la resolución de la otra solicitará o incluso ya habrá solicitado las de caché.

Proxy de almacenamiento local

Dado que la aplicación no necesita almacenar muchos datos, se ha implementado también un sistema sobre el acceso al almacenamiento local, que mantiene una copia de los datos en la memoria principal y ofrece un tiempo de respuesta más rápido. Este se sincroniza después de manera asíncrona con el almacenamiento real del dispositivo. Al ser pocos datos, no supone un problema mantenerlos en memoria principal mientras se les está dando uso.

Este sistema ha sido preparado para poder ser desactivado cambiando la variable correspondiente en el fichero **Config.js**. De esta manera, se pueden realizar pruebas de la aplicación sin el efecto de este sistema de almacenamiento intermedio, para propósitos de depuración y para ver si realmente tiene un efecto visible. Haciendo estas pruebas, se puede notar a simple vista la mejora en la fluidez de la aplicación cuando hay involucradas operaciones de almacenamiento, incluso sin tener que realizar mediciones precisas.

Para la implementación de este sistema se ha utilizado el patrón de programación Proxy. En la Figura 5.2 se puede ver el diagrama de este patrón, aplicado al almacenamiento de eventos favoritos. En este patrón, el cliente se comunica con el objeto de almacenamiento, sin saber que se trata de un intermediario entre él y el objeto de almacenamiento real.

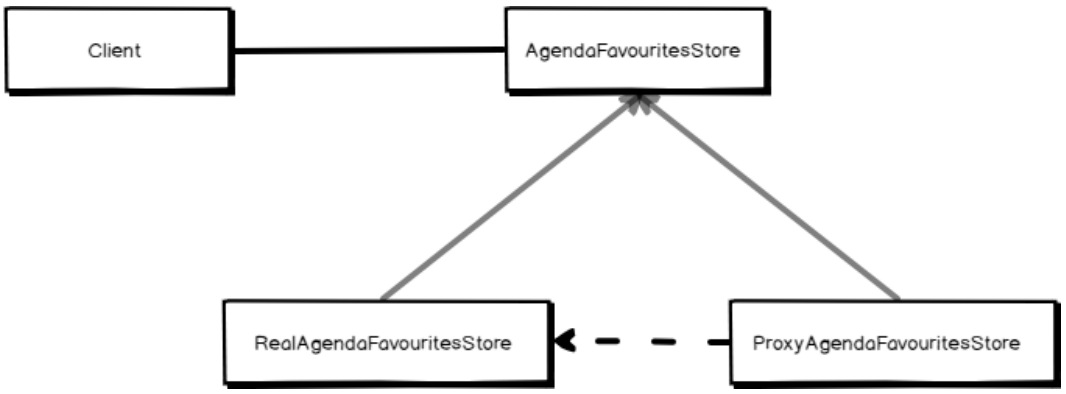


Figura 5.2: *Diagrama del patrón Proxy en el almacén de eventos favoritos.*

A continuación se muestra un ejemplo del código, incluyendo solo el método *getFavourites*, el cual responde con los eventos favoritos almacenados, a no ser que aún no haya sido utilizado y tenga que pedírselo al almacenamiento real:

```
// ProxyFavouritesStore
export default (realFavouritesStore) => {
    const PROXY = {data: null};

    const getFavourites = (then) => {
        if (PROXY.data && then)
            then(PROXY.data);
        else
            requestData(() => getFavourites(then));
    };

    // Other methods
    // ...

    const requestData = (then) => {
        realFavouritesStore.getFavourites((favourites) => {
            PROXY.data = favourites;
            if (then)
                then();
        })
    };

    return {getFavourites, /*Other methods ...*/};
};
```

En esta implementación, para que su funcionamiento mantenga integridad de datos, no se debe mezclar el uso del objeto Proxy con el real. Esto ya se ha tenido en cuenta en la exportación del módulo, exportando solo uno u otro en función del fichero de configuración mencionado antes.

5.2. Verificación y validación

Realizar un proceso de test exhaustivo sobre este tipo de sistemas es complejo. Por una parte, está el inconveniente de que han de funcionar sobre una variedad inmensa de dispositivos distintos, y en dos sistemas operativos distintos también. Por otra, una de las partes más importantes de este proyecto es la interfaz de usuario, y hay que lograr que se adapte bien a los distintos tamaños de pantalla sobre los que se va a mostrar.

Existen técnicas para realizar tests sobre componentes de React Native, tests unitarios en este entorno y tests sobre los elementos de Redux. Algunos se han podido utilizar, y otros no por falta de tiempo y complicaciones que han surgido en su intento.

Los tests que se han utilizado, explicados en esta sección, son por un lado tests unitarios, y por otro sobre componentes de React Native simples basados en *snapshots*. Se ha utilizado además la integración continua de Bitbucket, la cual ejecuta los tests cada vez que se suben cambios al repositorio. En la Figura 5.3, se puede ver una captura de Bitbucket, ejecutando estos tests tras añadir cambios en el proyecto.

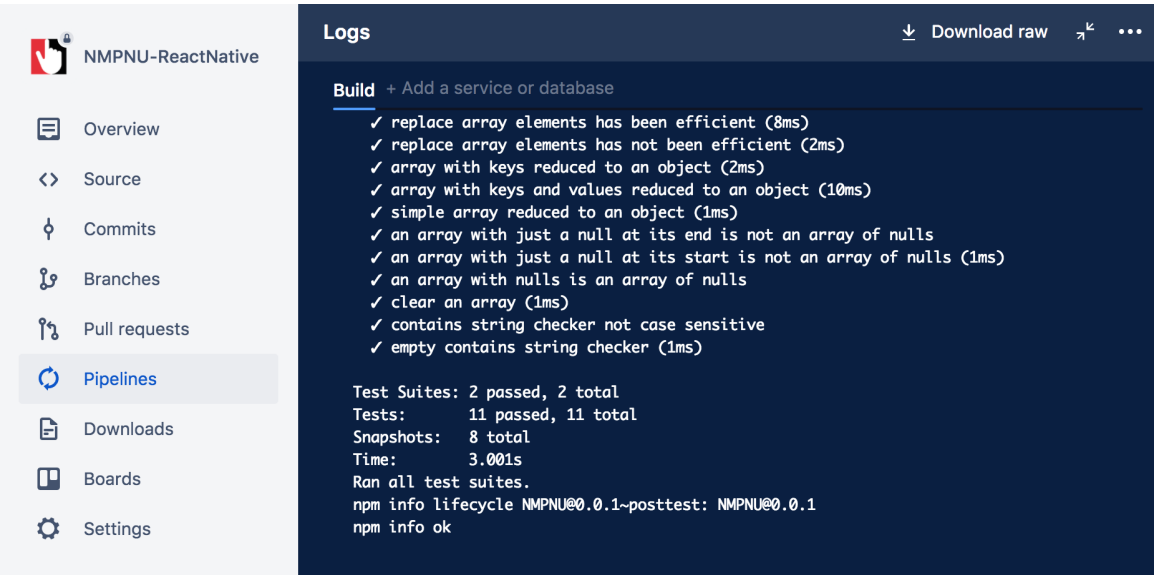


Figura 5.3: *Tests en la integración continua de Bitbucket.*

El sistema de publicación de versiones utilizado, Visual Studio App Center, ofrece servicios para que, cada vez que se monte la versión, esta sea probada sobre dispositivos reales. Sin embargo, este tipo de tests no han sido utilizados en el proyecto, dado que la empresa dispone de departamento de *testing* y no suele utilizar este servicio.

La empresa dispone de una serie bastante amplia de dispositivos, móviles y tablets, para realizar pruebas. Una práctica que se ha seguido en el desarrollo de este proyecto, es probar en los simuladores de Android e iOS la aplicación y todas sus funcionalidades desarrolladas hasta el momento de forma constante. De forma habitual, también se ha ido probando sobre distintos dispositivos reales.

5.2.1. Tests unitarios

Para la realización de tests unitarios, se ha utilizado el módulo Jest. Este permite declarar tests en JavaScript de manera bastante elegante. Se declaran con el método *test*, con un nombre y una función que corresponde al test. En esta última, se utilizan los métodos de la API según convenga. En el siguiente código se puede ver un ejemplo de la sintaxis de estos tests, donde se han utilizado los métodos *expect(expr).toBeFalsy* y *expect(expr).toBe*:

```
import {
    arrayToObjReducer, clearArray,
    containsStringChecker, isArrayOfNulls
} from '../../src/util';

test('an array with just a null at its end is not an array of nulls',
    () => expect(isArrayOfNulls(['A', 'AB', null])).toBeFalsy());

test('clear an array', () => {
    let a = [1,34,12];
    clearArray(a);
    expect(a.length).toBe(0);
});

test('empty contains string checker', () => {
    let checker = containsStringChecker([]);
    expect(checker('player')).toBeFalsy();
    expect(checker('notIn')).toBeFalsy();
});
```

Los tests unitarios se han utilizado para la mayoría de métodos útiles de la aplicación, con la idea de tests de caja negra, probando con representantes de clases de entradas posibles, incluyendo condiciones de frontera.

Jest también permite comprobar en sus tests si su ejecución ha mostrado algo por consola (e.g. mensajes de *logging*). En el ejemplo siguiente, se puede ver un uso de esto, dado un método que avisa por consola cuando las entradas que está recibiendo no le permiten trabajar eficientemente:

```
import {replaceOn} from '../../src/util';

test('replace array elements has not been efficient', () => {
    let a1 = /*Initialization ...*/
    let a2 = /*Initialization ...*/
    let outputData = "";
    let storeLog = inputs => (outputData += inputs);
    console["log"] = jest.fn(storeLog);
    replaceOn(a1, a2, (elem) => elem.id);
```

```
    expect(outputData).toMatch(/[\S\s]+optimization/);
});
```

5.2.2. Tests de *snapshot*

Otra técnica utilizada para realizar tests sobre React Native, consiste en realizar un *snapshot* de la renderización de un componente (i.e. representar su aspecto visual en un fichero con JSX). La primera vez que se ejecuta el test, se crea este fichero. A partir de entonces, cada vez que se ejecute, se comprobará por medio de un test de Jest que el *snapshot* resultante coincide con el del fichero.

Esto permite alertar sobre si un componente ha cambiado inesperadamente de aspecto. Cuando el cambio sea intencionado, el desarrollador tiene que actualizar el *snapshot* que hay en el fichero. Para realizar este *snapshot*, se utiliza un renderizador mock de React, disponible en el módulo react-test-renderer.

Cuando un subcomponente del componente que se está probando es demasiado complejo y se puede obviar su buen funcionamiento para el test, hay métodos disponibles en el API de Jest para reemplazarlos por versiones mock. En el siguiente ejemplo se puede ver la sintaxis de este tipo de tests:

```
// React, AgendaCardSection and renderer imports ...
// ...

jest.mock('react-native-view-shot', () => 'ViewShot');

test('agenda card section renders', () => expect(renderer.create(
    <AgendaCardSection
        item={/*example object ...*/}
    />
).toJSON()).toMatchSnapshot());
```

Un problema que se ha detectado en uno de los tests, ha consistido en que el servidor donde se ejecuta la integración continua de Bitbucket, se encuentra en una zona horaria distinta a la nuestra. Este hecho provocaba que los tests sobre componentes que mostraban fechas y horas de eventos, fallasen al representar la hora de forma humanamente legible, dado que el *snapshot* contra el que se probaban había sido creado en el entorno local. La solución ha consistido en fijar la zona horaria de la aplicación, independientemente de la del sistema. Se trata de una medida permisible, dado que la aplicación actualmente es útil tan solo en la provincia de Castellón, España, y no es probable que se expanda internacionalmente.

5.2.3. Tests más completos

Los tests que se han realizado en este proyecto sobre componentes React Native, no han sido realizados de manera tan completa como se podría. Hay otras técnicas y módulos, como Enzyme [3], que permiten realizar tests sobre componentes más complejos, incluyendo la manipulación de su estado para ver resultados.

Estos se han estudiado e intentado aplicar al proyecto, pero debido a complicaciones con las versiones de librerías y documentación desfasada, no se ha logrado con éxito. Con más tiempo, en otros proyectos sería interesante verlo con más profundidad y aplicar alguna de estas técnicas.

Capítulo 6

Conclusiones

6.1. Académicas

La mayoría de tecnologías involucradas en el proyecto no han sido vistas en el Grado en Ingeniería Informática, concretamente en el itinerario de Ingeniería del Software. Por ejemplo, no se ha visto el lenguaje de programación utilizado, ni nada relacionado con el desarrollo móvil. Sin embargo, es cierto que la función de este Grado es proporcionar una base sólida para después tener la capacidad de aprender tecnologías nuevas, dado que se trata de un campo muy amplio y en constante evolución. Realmente, no ha sido complicado aprender lo necesario siguiendo los cursos de formación al principio de la estancia.

En cuanto a la metodología utilizada, sí que se ha recibido formación en el Grado sobre esta. En los cursos más tempranos se profundiza en metodologías tradicionales, mientras que en el último curso se imparte una asignatura dedicada a las metodologías ágiles, de forma teórica y además aplicadas a un proyecto académico pero real. En esta asignatura se ha visto la metodología utilizada para el proyecto: Scrum.

6.2. Sobre React Native

React Native me ha dado muy buena impresión en unos aspectos, y no tan buena en otros. Por una parte, se trata de un *framework* relativamente reciente y se observa cómo ha aprendido en muchos aspectos de otros predecesores. Por otra parte, esta relativa novedad hace más difícil encontrar en la comunidad de desarrolladores soluciones a problemas que puedan surgir.

Otra de las desventajas que tiene, es la falta de madurez en algunas de las librerías secundarias dedicadas a esta tecnología, y sobre todo el rápido crecimiento de estas, el cual provoca que en poco tiempo unas hayan quedado obsoletas, y otras aún estén constantemente actualizándose.

El hecho de permitir el desarrollo simultáneo para Android e iOS es una ventaja, pero hay que tener en cuenta que pueden, y lo más seguro es que lo hagan, surgir comportamientos

inesperados de un sistema a otro. Por ello, es importante estar constantemente probando la aplicación en ambos sistemas para encontrar y resolver pronto problemas puntuales.

También me ha gustado pasar bastantes horas desarrollando en JavaScript, dado que hoy en día se trata de uno de los lenguajes de programación más importantes y aún no había podido profundizar en él. Por otra parte, en futuros proyectos con esta tecnología preferiría utilizar TypeScript, un superconjunto de JavaScript que permite disponer de tipado e interfaces de programación.

React native pertenece a la categoría de *frameworks* de desarrollo móvil nativo multiplataforma, y dentro de esta ha ganado terreno a otros como Native Script [10], con soporte para Angular. Sin embargo, recientemente ha aparecido uno nuevo que tiene papeles para dejar atrás a React Native. Se trata de Flutter [4], y es desarrollado por Google, hecho que da a pensar que puede estar muy bien, tras ver el buen trabajo que han hecho con Android.

6.3. Sobre Redux

Redux me ha parecido indispensable para proyectos como este. Se trata de un MVC actualizado para dar apoyo a las *single page aplications*. Salvo detalles concretos de la implementación, la manera general en la que se ofrece su API me parece insuperable, haciendo transparente al desarrollador todo lo que ha podido, con muy poco código *boilerplate* necesario y con la idea de utilizar ideas del paradigma de programación funcional. Además, ofrece todo su potencial en un módulo de apenas 2kB.

6.4. Sobre el proyecto

El resultado final del proyecto es una versión con toda la funcionalidad requerida, sin tener en cuenta las funcionalidades de última hora, pero sin tener los diseños de la interfaz de usuario adaptados. No hubo tiempo de adaptar toda la aplicación a los diseños, ni de terminar algunas de las funcionalidades que surgieron junto a estos.

El único problema notable que ha habido en este proyecto, es la tardía entrega de los diseños de la interfaz de usuario. Las funcionalidades nuevas que iban surgiendo durante el desarrollo aún podían introducirse en la planificación sin problemas, pero la gran cantidad de ellas surgidas en la última semana fueron imposibles de planificar y realizar como es debido. En proyectos reales, los diseños se obtienen prácticamente al inicio, pero en este, es comprensible que no pudo ser posible, al no tener prioridad suficiente siendo un proyecto de estancia en prácticas.

Como ya experimenté en otros proyectos con asuntos similares, hasta que han estado disponibles los servicios web, ha sido útil utilizar mocks que simulan su comportamiento. De esta manera he podido desarrollar funcionalidades que requerían de estos servicios web, aún en desarrollo por entonces. Una vez han estado listos no ha sido complicado integrarlos en el sistema.

En cuanto al futuro del proyecto, voy a quedarme a trabajar en la empresa y lo primero

que se me asignará es terminar esta aplicación, incluso con la ayuda de otros miembros del departamento de desarrollo móvil con React Native. La aplicación tiene además un motivo especial, y es que se trata de la celebración del décimo aniversario de Nomepierdoniuna. Es por ello que los logos de la cabecera de la aplicación y la *splash screen* incluyen un número 10.

Bibliografía

[1] Burndown Chart. https://confluence.atlassian.com/jirasoftwarecloud/burndown-chart-777002653.html. [Consulta: 27 de Febrero de 2018].

[2] Diagrama de Scrum. https://dbcms.s3.amazonaws.com/devbridgecom/bcms/image/16ded07f8edc4a0c911ae16c511ef241/ScrumBacklog.jpg. [Consulta: 27 de Febrero de 2018].

[3] Enzyme. https://github.com/airbnb/enzyme. [Consulta: 2 de Marzo de 2018].

[4] Flutter. https://flutter.io. [Consulta: 4 de Marzo de 2018].

[5] Flux architecture in depth. https://facebook.github.io/flux/docs/in-depth-overview.html#content. [Consulta: 7 de Febrero de 2018].

[6] Flux diagram. https://facebook.github.io/flux/img/flux-simple-f8-diagram-with-client-action-1300w.png. [Consulta: 11 de Febrero de 2018].

[7] Introduciendo JSX. https://reactjs.org/docs/introducing-jsx.html. [Consulta: 2 de Febrero de 2018].

[8] Jira. https://es.atlassian.com/software/jira. [Consulta: 1 de Marzo de 2018].

[9] Motivación de Redux. https://redux.js.org/docs/introduction/Motivation.html. [Consulta: 2 de Febrero de 2018].

[10] Native Script. https://www.nativescript.org/. [Consulta: 4 de Marzo de 2018].

[11] Node. https://nodejs.org/es. [Consulta: 28 de Febrero de 2018].

[12] Notificaciones push. https://es.wikipedia.org/wiki/Notificaci%C3%B3n_push. [Consulta: 11 de Febrero de 2018].

[13] React en Github. https://github.com/facebook/react. [Consulta: 30 de Enero de 2018].

[14] React JS library. https://reactjs.org. [Consulta: 30 de Enero de 2018].

[15] React Native CLI, npm package. https://www.npmjs.com/package/react-native-cli. [Consulta: 10 de Febrero de 2018].

[16] React Native en GitHub. https://github.com/facebook/react-native. [Consulta: 10 de Febrero de 2018].

[17] React Native Framework. https://facebook.github.io/react-native. [Consulta: 30 de Enero de 2018].

[18] Sobre Cuatroochenta. https://www.cuatroochenta.com/sobre-cuatroochenta. [Consulta: 30 de Enero de 2018].

[19] Sobre Nomepierdoniuna webzine. http://www.nomepierdoniuna.net/about. [Consulta: 30 de Enero de 2018].

[20] Sobre OneSignal. https://onesignal.com/about. [Consulta: 11 de Febrero de 2018].

[21] VirtualDom. https://reactjs.org/docs/faq-internals.html. [Consulta: 2 de Febrero de 2018].

[22] Watchman. https://facebook.github.io/watchman. [Consulta: 7 de Febrero de 2018].

[23] Webstorm. https://www.jetbrains.com/webstorm. [Consulta: 2 de Febrero de 2018].

[24] Javier Garzas. ¿Por qué utilizamos Puntos Historia para estimar y no horas? . http://www.javiergarzas.com/2015/06/puntos-historia-para-estimar-y-no-horas.html. [Consulta: 27 de Febrero de 2018].

[25] Julián Gómez. Método de Estimación Ágil: Puntos de Historia. http://www.laboratorioti.com/2013/02/21/metodo-de-estimacion-agil-puntos-de-historia. [Consulta: 27 de Febrero de 2018].