UniversidaddeValladolid

E.T.S Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Dashboard personal basado en widgets

Junio de 2017

Autor:
Jairo Velasco Martín
Tutor:
Yania Crespo González-Carvajal

RESUMEN

El objetivo de este proyecto es desarrollar una aplicación de tipo SaaS (*Software as a Service*), cuyo propósito es presentar información relevante para la organización y la rutina diaria de los usuarios, como pueden ser noticias, listas de tareas o la previsión del tiempo, de forma conjunta y personalizable.

La aplicación ha sido denominada *Dashpot*, y se trata de un *dashboard* personalizable, en el que el usuario elige qué *widgets* mostrar, así como su ubicación en la pantalla y su configuración.

El proyecto ha sido desarrollado empleando como marco la pila MEAN, junto a las tecnologías propias de proyectos web, como son HTML y CSS, siguiendo una metodología de desarrollo ágil, aplicando los principios de SCRUM y de BDD (*Behaviour Driven Development*).

ABSTRACT

The purpose of this project is to develop a SaaS (*Software as a Service*) application, which goal is to provide relevant information to the users daily planning and routine, such as news, to-do lists or the weather forecast, in a combined and customizable way.

The application has been named Dashpot, and it is a customizable dashboard, in which the user can choose the widgets to show, the position on the screen and their configuration.

The project has been developed using the MEAN stack as a framework, along with the technologies for web projects, such as HTML and CSS, following an agile development methodology and applying the principles of SCRUM and BDD (*Behaviour Driven Development*).

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

PARTE PRIMERA - INTRODUCCIÓN Y CONTEXTO

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS.

1.1 Introducción.

En los últimos tiempos han ido apareciendo herramientas y servicios de información que facilitan nuestro día a día, nos permiten organizarnos, recordar lo que tenemos que hacer y obtener datos. A menudo estas tecnologías se presentan de manera separada, de forma que, por ejemplo, hay que acceder a un servicio para consultar el tiempo y después a otro para utilizar una lista de tareas, perdiendo así el foco de atención. Es por ello que resulta interesante la creación de un panel que proporcione una visión conjunta y general de la información.

1.2 Motivación.

Este proyecto surgió a partir de la idea de que un usuario pueda disponer de un panel con múltiples herramientas en forma de widget, que aportasen valor durante el desarrollo de su rutina personal, ofreciendo una visión más global y resumida de la información relevante.

Dicha idea fue presentada a la profesora Yania Crespo González-Carvajal, que la consideró interesante, por lo que se comprometió a tutorizar el proyecto.

1.3 Objetivos.

El objetivo principal de este trabajo de fin de grado es el desarrollo de una aplicación SaaS (*Software as a Service*) empleando metodologías ágiles y *frameworks* potentes y novedosos. Por tanto, se consideran los siguientes subobjetivos:

- Planificación de un proyecto software mediante una metodología ágil como SCRUM, y un proceso de Desarrollo Basado en Comportamiento (BDD por sus siglas en inglés).
- Desarrollo de esquemas, bocetos y prototipos a lo largo de todo el desarrollo.
- Implementación de la aplicación empleando un framework *frontend* avanzado, como Angular 2, y un framework b*ackend,* como Node.js.

1.4 Organización de la memoria.

Esta memoria se organiza en tres partes principales, una primera parte introductoria para establecer el contexto y el marco de trabajo, una parte donde se exponen los detalles del proyecto y su planificación, y por último una parte donde se extraen conclusiones, de forma que se recogen los siguientes contenidos principales:

- **Marco teórico:** Se describen brevemente los métodos y técnicas empleados para plantear, planificar y desarrollar el proyecto, además de justificar su elección.
- **Marco tecnológico:** Se da una breve descripción de las herramientas y tecnologías empleadas para la planificación y el desarrollo del proyecto.
- **Proyecto software:** Se detalla la planificación, el desarrollo y el seguimiento del proyecto.
- **Conclusiones y mejoras futuras:** Se extraen conclusiones sobre el proyecto y se plantea una línea de posibles mejoras futuras.
- **Bibliografía:** Referencias bibliográficas consultadas durante el proyecto.
- **Anexos:** Elementos adicionales.

CAPÍTULO 2. MARCO TEÓRICO.

En esta sección se detallan las características de la metodología empleada para la realización del proyecto. Los métodos y procesos de desarrollo elegidos se corresponden con metodologías ágiles.

2.1 Desarrollo Ágil.

El desarrollo ágil hace referencia al uso de métodos de ingeniería del software basados en el desarrollo iterativo e incremental, permitiendo la evolución y adaptación de los requisitos a lo largo de la consecución del proyecto, manteniendo una relación continúa y directa con el cliente, que prima sobre la documentación.

El Manifiesto Ágil recoge cuatro valores principales y algunos principios para el desarrollo ágil de software, concretamente estos son los valores [1] que priman sobre los ligados a procesos de desarrollo más tradicionales:

- Individuos e interacciones sobre procesos y herramientas.
- Software que funciona sobre documentación comprensiva.
- Colaboración del cliente sobre negociación de contratos.
- Respuesta al cambio sobre el seguimiento de un plan.

2.2 Historias de usuario.

La especificación de requisitos en una metodología ágil típicamente se realiza empleando representaciones denominadas historias de usuario (*user stories*).

Una *user story* representa un requisito de la aplicación descrito en un lenguaje común y de forma breve, esto es, empleando una o dos frases.

Para lograr una gestión completa de los requisitos, las historias de usuario van acompañadas de discusiones con los usuarios y de pruebas de validación. De manera que, en el momento de implementar una historia, el desarrollador tenga la posibilidad de discutirla con el cliente, y que una vez implementada pueda ser probada y validada.

El estilo con que se escribe una historia de usuario puede ser libre, pero es recomendable seguir la siguiente estructura [2]:

Como (rol)
quiero (algo)
para poder (beneficio).

Dicha estructura permite responder a las preguntas de qué se quiere, quién se beneficia y cuál es el beneficio. Las *user stories* deben ser pequeñas, verificables, estimables, negociables, valoradas por usuarios o clientes e independientes unas de otras.

Acompañando a las historias de usuario se pueden definir los escenarios, que son los conjuntos de pasos que tiene que dar un usuario para obtener un resultado usando la aplicación. Dichos escenarios pueden ser de éxito o representar situaciones de fallo [3].

2.3 SCRUM.

SCRUM es un marco de desarrollo ágil para la gestión de proyectos, iterativo e incremental, que se caracteriza por el solapamiento de las distintas fases del desarrollo, en contraposición a los ciclos secuencial o de cascada [4].

A los períodos en que se desarrolla trabajo se los denomina *sprints*. La duración de un *sprint* es ajustable, pero se recomienda que no sea variable. Al final de cada *sprint* el equipo debe presentar los avances y el resultado será un producto potencialmente entregable.

El *product backlog* es un documento en el que se recogen todos los requisitos del proyecto, contiene descripciones genéricas de las funcionalidades deseadas, así como una estimación a grandes rasgos del esfuerzo requerido para lograr cada una; representa qué va a ser construido.

Al planificar un *sprint* se toma un subconjunto de tareas y se añade a otro documento llamado *sprint backlog*, este contiene los requisitos que se van a desarrollar en el *sprint*. Las tareas del *sprint backlog* no se asignan, sino que son los integrantes del equipo de desarrollo los que las van tomando.

En un desarrollo siguiendo SCRUM se dan típicamente los siguientes roles:

- ***Product Owner*:** Es el representante del cliente, se asegura de que el equipo trabaja de acuerdo con los intereses del negocio. Es también el encargado de escribir, priorizar y colocar en el *product backlog* las historias de usuario.
- ***Scrum Master*:** Realiza las labores de facilitador, es decir, su trabajo es eliminar los obstáculos que puedan impedir lograr el objetivo de un *sprint*.
- **Equipo:** Su trabajo es desarrollar el proyecto.

En un planteamiento de tipo SCRUM, aparecen las siguientes reuniones:

- ***Daily Scrum*:** Cada día del *sprint* las personas involucradas en el desarrollo del proyecto se reúnen durante un tiempo breve (típicamente inferior a 15 minutos) para contestar a tres cuestiones: qué se ha hecho ayer, qué se hará hoy y si se ha presentado algún problema que impida alcanzar un objetivo.
- ***Sprint planning*:** Al inicio de cada sprint se realiza una reunión en la que se selecciona qué trabajo realizar en este periodo, se prepara el *sprint backlog* y se comunica e identifica cuánto del trabajo es probable que se realice en el *sprint*.
- ***Sprint review*:** El objetivo de esta reunión es revisar el trabajo que fue completado y no completado y presentar el trabajo completado a los interesados.
- ***Sprint retrospective*:** En esta reunión los integrantes del equipo dejan sus impresiones sobre el *sprint* realizado con el propósito de lograr una mejora continua.

2.3.1 Beneficios de SCRUM.

Emplear un marco para el desarrollo ágil como SCRUM aporta beneficios sobre procesos de desarrollo más tradicionales como el desarrollo en cascada, algunos de ellos son los siguientes [5]:

- Mayor productividad debido a la capacidad del equipo para estructurarse y a una menor burocracia.
- Mayor flexibilidad ante cambios gracias a que el marco está diseñado para adaptarse a nuevos requisitos o la evolución de los existentes, que pueden darse por necesidades del cliente o la evolución del mercado.
- Reducción del *time to market*, debido a que el cliente puede empezar a usar características del producto antes de que esté totalmente completado.
- Reducción de riesgos, ya que a causa de desarrollar primero las funcionalidades más importantes y de saber la velocidad a la que avanza el equipo pueden despejarse algunos riesgos de forma anticipada.

2.4 Behaviour Driven Development (BDD).

El desarrollo guiado por comportamiento (conocido en inglés como *Behaviour Driven Development* o por sus siglas BDD) es un proceso de desarrollo de software que surgió a partir del desarrollo guiado por pruebas (*Test Driven Development* o TDD).

El principal impulsor de BDD, Dan North, describió la metodología en una conferencia en 2009 [6] de la siguiente manera:

BDD es una metodología ágil de segunda generación, de tipo afuera hacia adentro, de escala múltiple, y de alta automatización, que describe un ciclo de interacciones con producciones bien definidas, resultando en el envío de software funcional, probado y trascendente.

La característica principal de BDD es la utilización de un lenguaje de dominio (*Domain Specific Language* o DSL) común que permite unir la parte de negocio y la parte técnica, y desde el que arrancar el *testing* y el desarrollo. En BDD, las pruebas de validación se describen con las historias de usuario, a las que se añaden criterios de aceptación en términos de escenarios [7].

El ciclo básico de un proceso de desarrollo guiado por comportamiento es el siguiente:

1 - Escritura de las historias de usuario.
2 - Especificación de los escenarios.
3 - Automatización de las pruebas de aceptación.

Las pruebas de validación están compuestas por historias de usuario, escritas siguiendo la recomendación ágil ya mencionada: *"Como [rol] quiero [algo] para poder [beneficio]."*

A su vez, los criterios de validación que componen un escenario deben estar implementados como cláusulas y en términos de situaciones siguiendo el principio *Given-When-Then*, con el siguiente formato [8]:

"Dado que [contexto inicial],
cuando [ocurre el evento],
entonces [asegurar algunos resultados]".

2.4.1 Beneficios de BDD.

El empleo de *Behaviour Driven Development* en el proceso de desarrollo aporta beneficios como una mejor comunicación entre todas las partes interesadas en un proyecto, ya que mediante el uso de lenguajes específicos de dominio (*DSL* por sus siglas en inglés) comprensibles por los stakeholders (*Business Readable)*, todas las personas implicadas en un proyecto son capaces de entender y escribir fácilmente requisitos y funcionalidades de la aplicación, ya tengan perfil técnico o no, permitiendo así una importante mejora en la comunicación [9].

Además, se elimina el frecuente problema de que existan múltiples documentos expresando la misma información, por lo que cuando se produce un cambio en un requisito, sólo hay que hacer modificaciones en un único lugar. Por ejemplo, al modificar una historia de usuario a su vez se estará cambiando la especificación de la prueba de validación correspondiente.

En definitiva, gracias al uso de BDD, los analistas y diseñadores son capaces de especificar más fácilmente los requisitos y los desarrolladores pueden validar con menor dificultad las funcionalidades de la aplicación, reforzando los principios ágiles de comunicación y de adaptación a los cambios.

CAPÍTULO 3. MARCO TECNOLÓGICO.

En esta sección se detallan las tecnologías concretas elegidas para el desarrollo del proyecto, éstas se adaptan a la metodología y el proceso de desarrollo descritos en el Capítulo 2.

3.1 La pila MEAN.

MEAN es un acrónimo para denominar a la pila de tecnologías que, funcionando de forma conjunta, conforman un marco de trabajo *full stack* para desarrollo web, empleando JavaScript o alguna de sus variantes como lenguaje de programación.

Las tecnologías que forman la pila MEAN [10] son:

- **MongoDB:** Como base de datos no relacional, para almacenar ficheros JSON.
- **Express:** Como *framework backend*, se trata de un módulo de Node.js que facilita tareas como crear una API REST.
- **Angular:** Como *framework frontend*, para crear la parte visual del cliente de la aplicación, en forma de *Single Page Application* (SPA).
- **Node.js:** Como entorno *backend* con un modelo asíncrono, en el que corren módulos como Express.

3.2 MongoDB.

MongoDB es un sistema gestor de bases de datos que se caracteriza por ser de tipo no relacional, porque almacena datos empleando documentos al estilo JSON junto a esquemas, en lugar de tablas [11].

Concretamente, los datos son almacenados e intercambiados empleando un formato llamado BSON, que se trata de una representación binaria de JSON.

En MongoDB, los datos se almacenan en colecciones, los documentos de una misma colección no tienen por qué seguir exactamente el mismo esquema, así, por ejemplo, algunos tendrán menos campos que otros, pero sin introducir nulos como ocurriría en una base de datos de la familia SQL.

Algunas de las características de Mongo [12] son:

- **Indexación**: Cualquier campo de un documento puede ser indexado, también pueden crearse índices secundarios.
- **Consultas *ad-hoc***: Soporta la búsqueda por campos, consultas de rangos y expresiones regulares.
- **Balanceo de carga**: MongoDB puede escalar horizontalmente usando una técnica denominada *sharding*. Esta técnica consiste en realizar particiones de la base de datos, cada una de esas particiones se denomina *shard*, y puede ser alojada en una instancia separada del servidor de bases de datos, para lograr balancear la carga.
- **Agregación**: Permite emplear el modelo de programación *MapReduce* para procesar grandes lotes de datos y operaciones de agregación.
- **Ejecución de JavaScript del lado del servidor**: Se puede emplear código JavaScript para realizar consultas, operaciones de agregación o ser ejecutado directamente desde el gestor de bases de datos.

3.2.1 Mongoose.

Mongoose es un ODM (*Object Document Mapper*) para MongoDB, su labor es traducir los objetos a sus representaciones en la base de datos, convirtiendo de la notación de objetos a la notación de documentos.

Además, añade funcionalidades muy interesantes de cara al modelado de la aplicación, ya que permite la creación de *schemas*, que son construcciones que permiten definir los campos y tipos que representan a una entidad de la lógica de negocio; gracias a ello es posible realizar validaciones sobre los datos, como pueden ser de tipo, de campos requeridos o de integridad referencial, lo cual resulta muy valioso dado que MongoDB es una tecnología que por sí sola apenas incluye la posibilidad de realizar estas comprobaciones, dado que sus mecanismos son muy pobres [13].

A partir de los esquemas, se declaran modelos sobre los que pueden ejecutarse las operaciones CRUD (*Create-Read-Update-Delete*), gracias a las funciones que integra, puede encargarse una operación sobre datos y procesar el resultado mediante un *callback*. A continuación, se incluye un ejemplo de consulta de datos sobre un modelo, tomado de [14].

```
var Person = mongoose.model('Person', yourSchema);

// find each person with a last name matching 'Ghost', selecting the `name` and `occupation` fields
Person.findOne({ 'name.last': 'Ghost' }, 'name occupation', function (err, person) {
  if (err) return handleError(err);
  console.log('%s %s is a %s.', person.name.first, person.name.last, person.occupation)
})
```

3.3 Express.

Express.js es un *framework* diseñado para la construcción de APIs y aplicaciones web, fue lanzado en 2010 y hoy en día es el estándar de facto en cuanto a marcos de trabajo de servidor para Node.js [13].

Express tiene como fundamentos ser mínimo y flexible, a la vez que provee de un conjunto robusto de funcionalidades para facilitar el desarrollo de aplicaciones. Entre dichas funcionalidades se encuentran métodos y utilidades HTTP que facilitan la creación de APIs REST.

Algunas de las funcionalidades que provee Express son las siguientes:

- **Direccionamiento:** Permite definir fácilmente los *endpoints* de una API y cómo se responde a las solicitudes de cliente, pudiendo crear métodos de ruta y vías de acceso [14].
- **Funciones de *middleware*:** Se trata de funciones que tienen acceso a los objetos de solicitud, respuesta y a la siguiente función del ciclo de petición/respuesta, por lo que pueden realizar tareas como ejecutar modificaciones en los objetos de solicitud y respuesta, finalizar el ciclo o invocar el siguiente *middleware* en la pila [15].
- **Motores de plantilla:** También son soportados algunos motores de plantilla como Jade, pudiendo hacer uso de funciones de *renderización* [16].

3.4 Angular.

Angular es un framework JavaScript de código abierto, pensado para la creación de aplicaciones de una sola página (*Single Page Applications*) basadas en el navegador y con capacidad MVC (*Modelo-Vista-Controlador*). La primera versión fue lanzada en 2010 bajo el nombre de AngularJS. Posteriormente ha ido evolucionando hasta ser reinterpretada por completo, siendo actualmente mantenida por Google [17].

Angular se encuentra actualmente en desarrollo y bajo constante evolución; aunque mantiene algunos conceptos de AngularJS, es incompatible con dicha versión, ya que no comparten principios ni estructura.

La primera versión considerada estable del nuevo Angular, llamado Angular 2, fue lanzada en septiembre de 2016, y desde entonces se han ido incorporando nuevas funcionalidades. La salida de Angular 4 está prevista para diciembre de 2017; cabe mencionar que el equipo de desarrollo decidió dar un salto en la numeración, pasando de la versión 2 a la 4, puesto que habían empleado la 3 para desarrollar funcionalidad en paralelo [18].

Para el desarrollo del proyecto que nos ocupa, se ha escogido Angular 2, ya que a pesar de que AngularJS sigue siendo mantenido, resulta más conveniente adoptar las mejoras y principios que propone la nueva versión, aprendiendo, además, a usar un *framework* con mejor proyección de futuro.

3.4.1 Características y principios.

La característica más importante de Angular 2 es su arquitectura basada en componentes [19], que trata de facilitar los principios básicos de bajo acoplamiento y alta cohesión. Esta aproximación aporta ventajas [20] en cuanto a:

- **Reutilización:** En caso de necesitar exponer la misma funcionalidad en diferentes puntos de la aplicación, los componentes de Angular 2 permiten hacerlo fácilmente.

- **Facilidad para realizar pruebas:** Es más sencillo realizar pruebas unitarias sobre un componente con un pequeño conjunto de responsabilidades que sobre una aplicación grande no estructurada.

- **Legibilidad:** En general**,** resulta más fácil de leer una aplicación bien estructurada y encapsulada en paquetes, y Angular permite encapsular la funcionalidad de una manera lógica y entendible.

- **Mantenibilidad:** Si se logra un bajo acoplamiento gracias al uso de componentes bien diferenciados, el sistema será más fácil de mantener.

3.4.2 Arquitectura.

En la Ilustración 1 se muestra la arquitectura que presenta Angular, en cuanto a la naturaleza y funciones de sus componentes y las relaciones entre ellos.

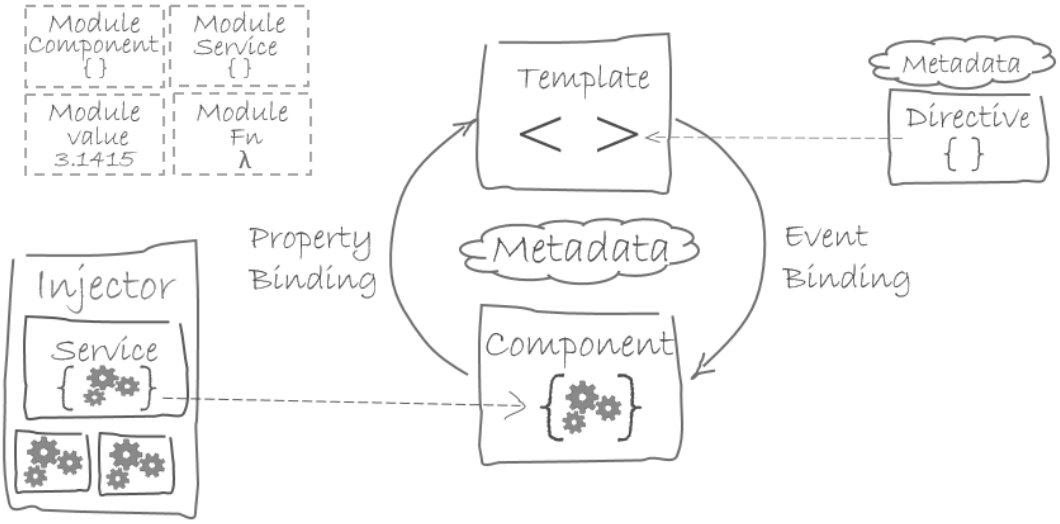

ILUSTRACIÓN 1 - ARQUITECTURA DE UNA APLICACIÓN ANGULAR 2. TOMADO DE [21].

- **Módulos:** Los módulos son componentes que se caracterizan por tener una función única, por lo que desempeñan una sola tarea. Todas las aplicaciones Angular tienen al menos un módulo, que es la propia aplicación, pero a su vez pueden estar compuestas de más módulos.

- **Componentes:** Los componentes son los bloques básicos de construcción. Un componente es una clase controladora junto con una plantilla; todos los elementos visuales en Angular son componentes, los cuales pertenecen a módulos, de los cuales pueden ser exportados. Un componente sabe cómo *renderizarse* a sí mismo y cómo configurar la inyección de dependencias, además, puede tener asociados estilos CSS.

- **Plantillas:** La vista de un componente se define usando una plantilla, que permite definir cómo mostrar la información. Las plantillas están compuestas por código HTML junto a anotaciones de Angular.

- **Servicios:** Los servicios son clases cuyo propósito es servir una funcionalidad, generalmente relacionada con un acceso a datos. Por ejemplo, un servicio puede encargarse de la autenticación de usuarios, o de obtener datos a través de una API. Los servicios son inyectados en los componentes mediante el mecanismo de inyección de dependencias.

- **Directivas:** Una directiva es una clase que representa metadatos, las directivas pueden ser de componente, de decorador o de plantilla.

- **Metadatos:** Los metadatos definen la manera en que se procesa una clase, esto se logra añadiendo *decorators* de TypeScript, por ejemplo, si se tiene una clase llamada *MyComponen*t esta será simplemente una clase, y no será interpretada como un componente a no ser que se le añada el decorador @Component.

Al configurar el decorador, se específica el selector del *component*, que nos permitirá incrustar el componente dentro de una plantilla, las rutas a la plantilla y la hoja de estilos asociada, y otros datos como puede ser la lista de proveedores de servicios de los que se nutre el componente, a continuación, se muestra un ejemplo.

```
@Component({
  selector: 'app-example',
  templateUrl: './example.component.html',
  styleUrls: ['./example.component.css'],
  providers: [ExampleDataService]
})
```

- ***Data Binding*:** El *Data Binding* es el proceso por el que se coordinan o sincronizan los datos entre las fuentes y las vistas, es decir, conecta elementos de las plantillas HTML con elementos de componentes. Existen las siguientes formas [22] de vinculación:

    - ***Interpolation*:** Muestra el valor de una propiedad de un componente en la vista.
    - ***Property binding*:** Permite vincular una propiedad de la vista con algún elemento de un componente. Por ejemplo, permite vincular la propiedad src de un elemento *<img>* de la vista con un valor de un componente.
    - ***Event binding*:** Conecta un evento de la vista con alguna propiedad de un componente, por ejemplo, permite ejecutar una función determinada o fijar una variable.
    - ***Two-way binding*:** Enlaza bidireccionalmente una propiedad de un componente con un elemento de la vista, de manera que si se modifica el valor desde la vista este también cambia en el componente y viceversa [23].

3.4.3 Inyección de dependencias.

La inyección de dependencias es un patrón de diseño software, que consiste en suministrar un objeto a una clase en lugar de ser la propia clase la que crea el objeto.

En Angular, el mecanismo de inyección crea o mantiene instancias de servicios para pasárselas a los componentes que las requieren, para evitar que sean estos los responsables de crear y gestionar sus propias instancias de servicios [24].

3.4.4 Modelo-Vista-Controlador.

Angular permite seguir el patrón MVC (Modelo-Vista-Controlador), de forma que quedan perfectamente diferenciadas y desacopladas las funciones del modelo y de la vista. En una aplicación Angular, el modelo corresponde a las entidades y los servicios, el controlador a los componentes y la vista a las plantillas HTML [25].

En la Ilustración 2, se muestra la aplicación del patrón Modelo Vista Controlador a la arquitectura de un widget representativo de los que se implementan en la aplicación. Así se representa cada parte del patrón:

- **Modelo:** Las clases correspondientes a servicios, factorías, o entidades representan el modelo.

- **Vista:** Las plantillas, formadas mediante HTML y directivas propias de Angular, conforman la vista.

- **Controlador:** Las clases correspondientes a componentes, suponen la parte del controlador, el cual hace de intermediario entre el modelo y la vista.

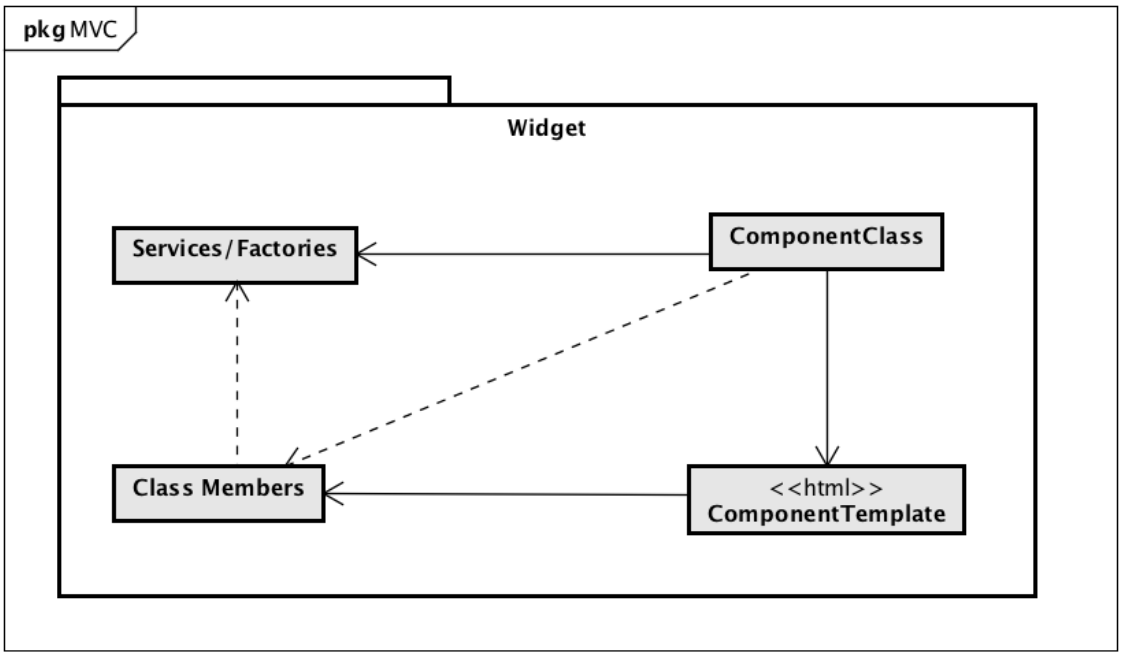

ILUSTRACIÓN 2 - APLICACIÓN DEL PATRÓN MVC EN ANGULAR 2

3.4.5 Mecanismo de detección de cambios.

El estado de la aplicación se puede ver reflejado en elementos como párrafos, formularios, enlaces o botones en la interfaz web, en general, cualquier elemento perteneciente al DOM (*Document Object Model*). Al proceso que plasma el estado de las estructuras de datos en elementos visuales se le denomina *rendering*.

Este proceso es relativamente sencillo cuando se realiza de forma estática, pero hacerlo en tiempo de ejecución requiere de un mecanismo de detección de cambios, que solucione los problemas de cómo detectar los cambios, qué modificar y cómo hacerlo.

Angular puede detectar los cambios de estado [26] producidos por:

- ***Eventos***: Como pueden ser *clicks*, envíos de formularios…
- ***XHR***: La obtención de datos del servidor mediante XHR (*XmlHttpRequest*).
- ***Timers***: Mediante funciones como *setTimeOut()*.

La tarea que desempeña el mecanismo de detección de cambios es básicamente la de monitorizar en todo momento el estado interno de la aplicación y hacer visibles, en la interfaz de usuario, todos aquellos cambios que se produzcan.

3.4.6 Ciclo de vida de los componentes.

Angular se encarga de crear componentes, renderizarlos, crear y renderizar a sus componentes hijos, comprobar cuando sus propiedades cambian y destruirlos antes de sacarlos del DOM. Para lograr esta gestión, los componentes tienen un ciclo de vida, y se nos ofrecen *hooks* (enganches) para poder intervenir en los momentos clave de la vida de un componente [27].

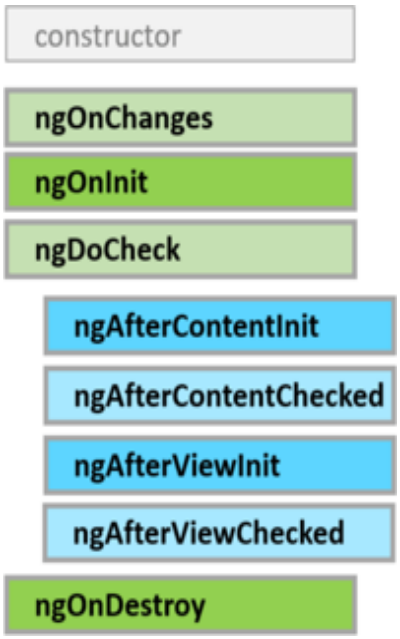

ILUSTRACIÓN 3 - CICLO DE VIDA DE UN COMPONENTE ANGULAR. TOMADO DE [27].

Para poder enganchar con estos eventos, hay que implementar el método correspondiente al evento, perteneciente a la interfaz del mismo nombre, pero sin el prefijo ng.

Por ejemplo, un componente puede implementar la interfaz *OnInit*, definiendo el método *ngOnInit()*, que se ejecutará cada vez que se inicialice el componente. Esto se puede lograr de la siguiente manera:

```
export class ComponentExample implements OnInit {
  constructor() { }

  // implement OnInit's ‘ngOnInit’ method
  ngOnInit() { console.log(‘OnInit executed’); }

}
```

La secuencia de vida se muestra en la Ilustración 3, a continuación, se detallan cada uno de los eventos del ciclo, y cómo se producen:

- ***ngOnChanges()*:** Ocurre cuando Angular actualiza las propiedades de entrada de datos vinculadas mediante *Data Binding*, se ejecuta antes que *ngInit()* y cada vez que se producen cambios.

- ***ngOnInit()*:** Inicializa el componente después de que Angular haya mostrado primero las propiedades vinculadas de datos, y fija las propiedades de entrada. Se ejecuta una única vez, después del primer *ngOnChanges()*.

- ***ngDoCheck()*:** Se emplea para detectar y actuar ante cambios que Angular no puede detectar por sí solo.

- ***ngAfterContentInit()*:** Responde después de que Angular proyecte en la vista un contenido externo, se ejecuta una vez tras el primer *ngDoCheck()*.

- ***ngAfterContentChecked()*:** Responde después de que Angular compruebe el contenido proyectado en el componente. Se ejecuta después de *ngAfterContentInit()* y de cada subsecuente *ngDoCheck()*.

- ***ngAfterViewInit()*:** Se ejecuta después de que Angular inicialice la vista de un componente y de sus vistas hijas. Se ejecuta una vez después del primer *ngAfterContentChecked()*.

- ***ngAfterViewChecked()*:** Responde después de que Angular compruebe la vista de un componente y sus vistas hijas. Se ejecuta después de *ngAfterViewInit()* y de cada subsecuente *ngAfterContentChecked()*.

- ***ngOnDestroy()*:** Se ejecuta justo antes de que Angular destruya el componente, eliminando la suscripción a los Observables y desacoplando los manejadores de eventos para evitar problemas de memoria.

3.5 Node.js.

Node.js es un entorno de ejecución multiplataforma, de código abierto, basado en ECMAScript y en el motor V8 (sobre el que se comenta más adelante), que se caracteriza por ser asíncrono, y presentar un modelo de E/S orientado a eventos, no bloqueante [28].

Resulta muy útil para la creación de programas altamente escalables como son los servidores web. Presenta un único hilo de ejecución, de forma que el orden de evaluación es secuencial, sin embargo, las operaciones relacionadas con entrada o salida de datos, se ejecutan de forma totalmente concurrente, gracias al obligado uso de *callbacks* [29].

En cuanto al motor V8, se trata de un entorno de ejecución para JavaScript, desarrollado por Google para utilizarlo en Google Chrome; es software libre y se caracteriza porque compila el código fuente JavaScript a código máquina, en lugar de interpretarlo.

Node.js incorpora una serie de módulos básicos, como pueden ser los relacionados con tareas de red, ficheros, *buffers*, temporizadores o *streams*.

Además, es posible añadir módulos de terceros que aporten funcionalidades nuevas, extendiendo o añadiendo niveles de abstracción al entorno. Para gestionar estos módulos, se emplea una herramienta denominada NPM (*Node Package Manager*); Express y Mongoose son ejemplos de estos módulos.

3.6 Protractor.

Una parte del ciclo básico de desarrollo BDD es la automatización de pruebas, es por ello necesario un *framework* que facilite esta tarea.

El *framework* elegido ha sido Protractor, por ser específico de aplicaciones Angular. Este permite realizar E2E (*End to End testing*), con la característica de que los test corren contra la aplicación en un navegador real, interactuando como lo haría un usuario [30].

3.7 Gherkin.

Gherkin es un lenguaje específico de dominio (*Domain Specific Languag*e, también conocido por sus siglas en inglés DSL) con la característica de ser comprensible por los stakeholders (*Business Readable*), sirviendo de vehículo entre el cliente (o sus representantes) y los desarrolladores.

Se trata de un lenguaje con el que describir las funcionalidades y definir el comportamiento del software, sin entrar en su implementación.

3.8 Cucumber.

Cucumber es un *framework* pensado para realizar BDD. Básicamente permite validar escenarios de la aplicación, escribiendo los test en un lenguaje plano y comprensible, como es Gherkin [31].

En Cucumber, las especificaciones se definen en ficheros con extensión *feature*, uno para cada característica; estos son los elementos que debe, o puede, incluir dicho fichero:

- **Característica (*Feature*):** Se trata del encabezado que define el marco de la prueba. Puede ser sólo el título de la característica o incluir también su descripción, por ejemplo, en forma de historia de usuario.

- **Escenario (*Scenario*):** Se trata de la lista de pasos que llevan a la consecución de un resultado, es decir, los pasos que daría un usuario para conseguir algún resultado valioso de la aplicación. Cada paso comienza con alguna de estas palabras clave:

  Given (Dado/a/os/as)
  When (Cuando)
  Then (Entonces)
  But (Pero) o And (Y)

- **Antecedentes (*Background)*:** Si todos los escenarios comparten una serie de precondiciones, es conveniente usar este apartado para especificarlas, evitando repetirlas en cada escenario.

- **Esquema del escenario (Scenario Outline):** Otra posibilidad es que los escenarios dependan de variables, por lo que es conveniente reutilizar un mismo escenario, para el que cambien dichas variables, en lugar de repetirlo muchas veces con distintos datos. Los *Scenario Outline* nos permiten hacer esto, ya que van junto a la tabla *Examples* donde definimos distintos valores de entrada.

A continuación, se muestra el ejemplo para la característica de registro:

```
  Feature: Register Example

Scenario Outline: The user is using the register form

  Given user is at the register page
  Given '<email>' is the user email in the register form
  Given '<password>' is the user password in the register form
  Given '<name>' is the user name in the register form
  When submitting the register form
  Then the register form is validated '<valid>'

  Examples:
    | email           | password                | name       | valid |
    | ""              | ""                      | ""         | false |
    | ""              | "thisisavalidpassword"  | "test"     | false |
    | "test.es"       | "thisisavalidpassword"  | "test"     | false |
    | "test@test.es"  | ""                      | "test"     | false |
    | "test@test.es"  | "thisisavalidpassword"  | "test"     | true  |
    | "test@test.es"  | "thisisavalidpassword"  | ""         | false |
```

3.9 Heroku.

Heroku es una plataforma de computación en la nube, que permite alojar servicios y aplicaciones web, ofreciendo interesantes capacidades de gestión y escalado.

Existe una modalidad gratuita que incluye los servicios básicos, pero si se desea escalar en cuanto a capacidades de cómputo o almacenamiento, es necesario adquirir un plan de pago.

La aplicación ha sido desarrollada y probada en un entorno local, por motivos de comodidad y velocidad, sin embargo, se han empleado los mecanismos de integración continua que ofrece Heroku, automatizando los despliegues, de forma que cada vez que se actualizaba el repositorio remoto, los cambios eran reflejados en el entorno de producción.

Si no se conecta un repositorio con Heroku y se activa el despliegue automático, es necesario ejecutar una serie de simples comandos para desplegar la aplicación haciendo uso de git, gracias a una herramienta llamada *Heroku Toolbelt*.

Además, Heroku cuenta con un dashboard desde el que gestionar la aplicación, pudiendo ver registros, cambios e incidencias, así como un gestor de *add-ons*. Estos *add-ons* permiten añadir fácilmente funcionalidades, como pueden ser de conexión con bases de datos, monitorización, alertas y notificaciones, mensajería, ejecución automática de tareas, seguridad, testing, almacenamiento en caché, etc. Por ejemplo, en este proyecto se ha empleado uno de estos add-ons, para obtener soporte para la base de datos MongoDB.

3.10 Pivotal Tracker.

Pivotal Tracker es una herramienta de tipo SaaS (*Software as a Service*) que permite gestionar la planificación de proyectos empleando metodologías ágiles, gracias al uso de los elementos ya comentados de historias de usuario y *sprints* [34].

Para ello se presenta una distribución por columnas, que se establece así:

- ***Icebox*:** Incluye las historias que, han sido definidas, pero no se ha previsto incluir en el sprint en curso ni en el siguiente.

- ***Backlog*:** Compuesto por las historias del siguiente sprint, ordenadas por prioridad.

- ***Current*:** Se trata de las historias que se están desarrollando en el sprint en curso, también ordenadas por prioridad.

- ***Done*:** Es el conjunto de historias ya implementadas a lo largo del desarrollo del proyecto.

- ***My Work*:** Son las historias asignadas a uno mismo, ya estén terminadas o no. Esto resulta útil si se trabaja en equipo.

- ***Epics*:** Se trata de historias más grandes, que engloban un conjunto de historias para poder añadir abstracción y tener una visión más amplia.

En Pivotal Tracker, existe un concepto llamado *velocity*, este nos permite medir la velocidad con que se completan las historias, en función del nivel de complejidad de la historia.

Por ejemplo, si tomamos una escala de 1 a 3, podemos añadir puntos de complejidad a las historias desde la menos compleja (1) a la más compleja (3), y Pivotal Tracker expresará la *velocity* en función de cuántos puntos

sumen las historias que se han completado en un sprint; de manera que se toma la cantidad media de puntos completados en los últimos *sprints* para tratar de predecir cómo irán los siguientes.

La información que nos aporta la *velocity* resulta muy útil a la hora de elegir qué tareas incluir en un *sprint*, estimar cuando se alcanzarán ciertos hitos, o evitar sobreestimar o subestimar las capacidades del equipo de desarrollo.

A la hora de definir historias, se nos ofrecen los siguientes tipos:

- ***Feature*:** Se trata de las características que aportan funcionalidad a nuestra aplicación. Es el tipo de historia más común, y se sigue el estilo ya comentado de "*Como (rol) quiero (algo) para poder (beneficio)*".

- ***Chore*:** Se trata de tareas de apoyo o más manuales, que no aportan una funcionalidad por si solas, como pueden ser configurar un repositorio, modificar un esquema de la base de datos, etc.

- ***Bug*:** Son las historias destinadas a corregir errores o *bugs* surgidos en la funcionalidad existente.

- ***Release*:** Este tipo se usa para marcar las tareas de despliegue de funcionalidad en producción.

3.11 Toggl.

Para controlar el tiempo dedicado a cada tarea, con el objetivo de realizar el seguimiento, se ha empleado una herramienta de tracking llamada Toggl [35].

Esta permite especificar qué tarea se va a monitorizar y a qué proyecto pertenece, pudiendo introducir a mano las horas empleadas o usar un temporizador, para, posteriormente obtener informes temporales con desglose de tareas.

Es multiplataforma, dado que está disponible para OS X, Windows, Linux, iOS y Android, y cuenta con una versión gratuita, que ha sido la empleada en este proyecto, aunque también son comercializadas versiones más avanzadas para empresas, que cuentan con más opciones de monitorización e informes más detallados.

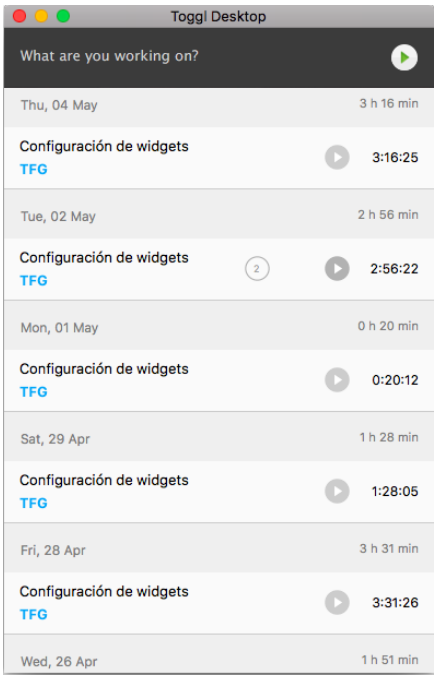

ILUSTRACIÓN 4 - INTERFAZ DE TOGGL

PARTE SEGUNDA - PROYECTO SOFTWARE.

CAPÍTULO 4. PLAN DE DESARROLLO SOFTWARE.

4.1 Planificación

Teniendo en cuenta que el Trabajo de Fin de Grado está valorado en 12 créditos ECTS (*European Credit Transfer and Accumulation System*) [32], y que un crédito equivale a 25 horas, se obtiene un total de 300 horas, por lo que se ha adaptado el alcance del proyecto para tratar de ajustarse a dicha duración.

Se han repartido las 300 horas estimadas del proyecto entre 20 semanas, esto resulta en 15 horas semanales. La duración de un *sprint* se ha fijado en dos semanas, por tanto, el proyecto se compone de 10 *sprints*, comenzando el 30 de enero, y terminando el 18 de junio.

Duración estimada: 300 horas

Duración del sprint: 2 semanas

Número de *sprints*: 10

Este es el desglose semanal, correspondiente a 15 horas/semana:

Semana	Fechas	Sprint
1	30 En - 5 Feb	1
2	6 Feb -12 Feb	
3	13 Feb - 19 Feb	2
4	20 Feb - 26 Feb	
5	27 Feb - 5 Mar	3
6	6 Mar - 12 Mar	
7	13 Mar - 19 Mar	4
8	20 Mar - 26 Mar	
9	27 Mar - 2 Abr	5
10	3 Abr - 9 Abr	
11	10 Abr - 16 Abr	6
12	17 Abr - 23 Abr	
13	24 Abr - 30 Abr	7
14	1 May - 7 May	
15	8 May - 14 May	8
16	15 May - 21 May	
17	22 May - 28May	9
18	29 May - 4 Jun	
19	5 Jun - 11 Jun	10
20	12 Jun - 18 Jun	

La fecha límite para la entrega del proyecto es el 28 de junio de 2017, por lo que se cuenta con una holgura de 10 días.

4.2 Product Backlog.

Para recoger la funcionalidad se han planteado historias de usuario, que conforman el *product backlog*. Se han especificado en total veinticuatro historias de usuario, que permiten recoger los requisitos de la aplicación de una forma adecuada con las metodologías ágiles.

En la tabla se muestra el contenido inicial del *product backlog*. En la sección 6.1 se detalla la descripción de cada una de las historias de usuario.

Product Backlog	
User Story	**Nombre**
US-001	Registrarse en la aplicación
US-002	Añadir *widget* al *dashboard*
US-003	Eliminar *widget* del *dashboard*
US-004	Modificar configuración de tamaño de un *widget*
US-005	Mover *widget* en el *dashboard*
US-006	Identificarse en la aplicación
US-007	Recuperar contraseña
US-008	Modificar perfil
US-009	Cambiar contraseña
US-010	Añadir una tarea a un *widget* de lista de tareas
US-011	Marcar como completada una tarea de un *widget* de lista de tareas
US-012	Marcar como no completada una tarea de un *widget* de lista de tareas
US-013	Eliminar tarea de un *widget* de lista de tareas
US-014	Leer noticia del *widget* de noticias RSS
US-015	Ver portada del *widget* de portadas
US-016	Añadir un *feed* al *widget* de noticias RSS
US-017	Eliminar un *feed* del *widget* de noticias RSS
US-018	Modificar el nombre de una lista de tareas
US-019	Modificar el nombre de un *widget* de noticias RSS
US-020	Modificar las unidades de un *widget* del tiempo
US-021	Modificar la ubicación sobre la que mostrar la previsión de un *widget* del tiempo
US-022	Consultar la previsión del tiempo
US-023	Leer una cita célebre
US-024	Ver la fecha actual

4.3 Desglose de Tareas.

A partir de las historias de usuario y de las tareas de planificación, instalación, configuración y documentación se ha detallado el siguiente desglose de tareas:

Desglose de tareas	
Tarea	**Descripción**
T-001	Especificar de las historias de usuario y sus escenarios
T-002	Establecer presupuesto y calendarización
T-003	Elaborar plan de riesgos
T-004	Configurar el entorno de desarrollo
T-005	Crear bocetos de la UI
T-006	Crear modelo de dominio
T-007	Plantear la arquitectura
T-008	Implementar la base del *dashboard* sobre la que disponer *widgets* (US-001, US-003, US-004, US-005)
T-009	Implementar la funcionalidad básica del widget de lista de tareas (US-010, US-011, US-012, US-013)
T-010	Implementar la funcionalidad básica del widget de noticias RSS (US-014)
T-011	Implementar la funcionalidad básica del widget de citas célebres (US-023)
T-012	Implementar la funcionalidad básica del widget de portadas (US-015)
T-013	Implementar la funcionalidad básica del widget del tiempo (US-022)
T-014	Implementar la funcionalidad básica del widget de bienvenida/fecha (US-024)
T-015	Implementar registro de usuarios (US-001)
T-016	Implementar login/logout (US-006)
T-017	Implementar recuperación de contraseña (US-007)
T-018	Implementar modo edición del dashboard
T-019	Añadir configuración al widget de lista de tareas (US-018)
T-020	Añadir configuración al widget de noticias RSS (US-016, US-017, US-019)

T-021	Añadir configuración al widget del tiempo (US-020, US-021)
T-022	Añadir configuración de tamaños a todos los widgets (US-004)
T-023	Implementar modificación de la configuración de perfil (US-008)
T-024	Implementar eliminación de widgets (US-003)
T-025	Implementar creación dinámica de widgets (US-002)
T-026	Mejoras y ajustes visuales
T-027	Mejorar la documentación interna
T-028	Revisar y realizar más pruebas
T-029	Documentar introducción, motivación y objetivos
T-030	Documentar el marco teórico y el tecnológico
T-031	Documentar el proyecto software
T-032	Documentar las conclusiones y futuras mejoras
T-033	Revisar, corregir y mejorar la documentación.

4.4 Plan de riesgos.

En este apartado se plantean posibles riesgos que pueden surgir a lo largo del desarrollo del proyecto, así como los planes de contingencia para cada uno de ellos.

Para hablar de riesgos, se tendrá en cuenta la siguiente clasificación:

- **Riesgos de proyecto:** Se trata de los relacionados con los miembros del equipo, los proveedores, la disponibilidad de los recursos y la calendarización.

- **Riesgos de producto o técnicos:** Son aquellos que se derivan de la falta de experiencia en el dominio del problema.

- **Riesgos de proceso:** Se trata de los relacionados con el proceso de desarrollo, como puede ser la documentación o la planificación.

Para analizar la exposición al riesgo se expresa la relación entre la probabilidad de ocurrencia de un riesgo y su impacto. Es conveniente representar esta clasificación en una matriz, en este caso se ha decidido que sea 3x3, aunque puede refinarse más en proyectos más complejos.

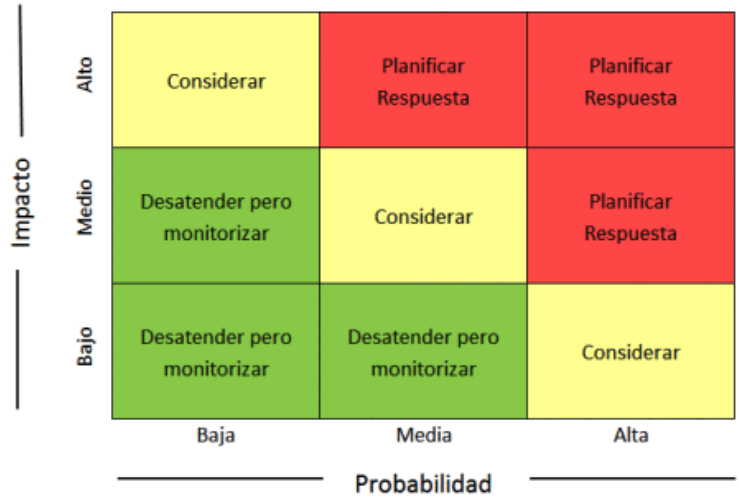

ILUSTRACIÓN 5 - MATRIZ DE ANÁLISIS DE EXPOSICIÓN AL RIESGO. TOMADO DE [47].

Para gestionar los riesgos, se establecen un plan de mitigación y un plan de contingencia [33]:

- **Plan de mitigación:** Se trata de lo que en el PMBOK (*Project Management Body of Knowledge*) se denomina Estrategia de Respuesta a los Riesgos, es decir, las acciones que intentan reducir la probabilidad de ocurrencia de un riesgo o el impacto en caso de que se produzca, por tanto, su objetivo es reducir la exposición al riesgo.

- **Plan de contingencia:** Se trata de las respuestas que se dan cuando efectivamente se produce un riesgo, el PMBOK define como respuesta de contingencia a aquellas respuestas que se utilizan solamente si ocurre efectivamente el riesgo.

RISK-001	Retraso en la planificación
Categoría	Proceso
Probabilidad	Alta
Impacto	Alto
Descripción	Pueden producirse retrasos debido a una mala estimación de la calendarización o de una mala planificación, lo que causaría no finalizar el proyecto en las fechas previstas.
Gestión del riesgo	
Estrategia	Reservar el riesgo.
Plan de mitigación	Se han previsto holguras en la planificación para que, en caso de que se produzcan retrasos, poder intentar finalizar el proyecto antes de la fecha límite.
Plan de contingencia	Incrementar el esfuerzo diario empleado.

TABLA 1 RIESGO DE RETRASO EN LA PLANIFICACIÓN

RISK-002	Falta de conocimiento de las tecnologías empleadas
Categoría	Producto
Probabilidad	Alta
Impacto	Alto
Descripción	Las tecnologías que se van a emplear, especialmente TypeScript y la pila MEAN, son mayormente desconocidas por el equipo de desarrollo.
Gestión del riesgo	
Estrategia	Investigar el riesgo.
Plan de mitigación	Leer documentación y realizar tutoriales y pruebas con las tecnologías antes y durante en el proyecto.
Plan de contingencia	Buscar información o pedir ayuda sobre las cuestiones que puedan surgir debido a la falta de conocimiento.

TABLA 2 RIESGO DE FALTA DE CONOCIMIENTO DE LAS TECNOLOGÍAS EMPLEADAS

RISK-003	Falta de madurez de Angular 2
Categoría	Producto
Probabilidad	Media
Impacto	Medio
Descripción	Angular es un *framework* en constante evolución, y la versión 2 ha sido lanzada hace poco, lo cual puede suponer que algunas funcionalidades no estén suficientemente desarrolladas o estén poco documentadas.
Gestión del riesgo	
Estrategia	Investigar el riesgo.
Plan de mitigación	Seguir de cerca la evolución del *framework*, analizando los cambios que se producen y cómo afectan a la versión empleada.
Plan de contingencia	Obtener ayuda experta en foros o apoyarse en JavaScript puro para implementar las funciones que resulten problemáticas usando Angular.

TABLA 3 RIESGO DE FALTA DE MADUREZ DE ANGULAR 2

RISK-004	Diseño pobre
Categoría	Proceso
Probabilidad	Baja
Impacto	Medio
Descripción	Un diseño pobre de la aplicación podría generar problemas de implementación.
Gestión del riesgo	
Estrategia	Protegerse del riesgo.
Plan de mitigación	Dar mucha importancia a las fases de diseño.
Plan de contingencia	Replantear el diseño.

TABLA 4 RIESGO DE DISEÑO POBRE

RISK-005	Cambios en los requisitos
Categoría	Proceso
Probabilidad	Media
Impacto	Medio
Descripción	Una modificación de los requisitos, es decir, de las historias de usuario, una vez empezada la construcción podría tener consecuencias para el proyecto.
Gestión del riesgo	
Estrategia	Protegerse del riesgo.
Plan de mitigación	Utilizar una metodología ágil que facilite una buena adaptación a los cambios.
Plan de contingencia	Realizar los cambios necesarios.

TABLA 5 RIESGO DE CAMBIOS EN LOS REQUISITOS

RISK-006	Problemas con los recursos hardware
Categoría	Proyecto
Probabilidad	Baja
Impacto	Bajo
Descripción	Un fallo en un equipo informático de los utilizados podría suponer retrasos en el proyecto.
Gestión del riesgo	
Estrategia	Protegerse del riesgo.
Plan de mitigación	Mantener a punto los equipos y realizar copias de seguridad del proyecto.
Plan de contingencia	Sustituir el equipo por otro o repararlo.

TABLA 6 RIESGO DE PROBLEMAS CON LOS RECURSOS HARDWARE

RISK-007	Arquitectura inadecuada para el propósito de la app
Categoría	Producto
Probabilidad	Baja
Impacto	Medio
Descripción	Una mala elección de la arquitectura de la aplicación puede implicar un sobreesfuerzo en el desarrollo, aumentando los costes.
Gestión del riesgo	
Estrategia	Investigar el riesgo.
Plan de mitigación	Plantear la arquitectura lo más pronto posible e investigar cómo evoluciona su implementación respecto al planteamiento inicial.
Plan de contingencia	Decidir si es más conveniente replantear una nueva arquitectura o asumir el sobrecoste y continuar con la misma.

TABLA 7 RIESGO DE ARQUITECTURA INADECUADA PARA EL PROPÓSITO DE LA APP

RISK-008	Falta de experiencia del equipo de desarrollo
Categoría	Proceso
Probabilidad	Media
Impacto	Alto
Descripción	La falta de experiencia de los recursos humanos del equipo de desarrollo puede suponer retrasos y sobrecostes.
Gestión del riesgo	
Estrategia	Protegerse del riesgo
Plan de mitigación	Incluir tiempo dedicado a la formación del equipo.
Plan de contingencia	Aumentar los recursos de tiempo o económicos destinados a formación.

TABLA 8 RIESGO DE FALTA DE EXPERIENCIA DEL EQUIPO DE DESARROLLO

RISK-009	Baja de un miembro del equipo de desarrollo
Categoría	Proyecto
Probabilidad	Baja
Impacto	Alto
Descripción	La baja de un miembro del equipo supondría sobrecostes y/o retrasos.
Gestión del riesgo	
Estrategia	Asumir el riesgo.
Plan de mitigación	Lo normal sería contar con recursos humanos de reserva, pero la naturaleza de este proyecto no lo permite por sus requisitos de autoría.
Plan de contingencia	Aceptar el retraso en la calendarización.

TABLA 9 RIESGO DE BAJA DE UN MIEMBRO DEL EQUIPO DE DESARROLLO

4.5 Cálculo del Presupuesto.

Es necesario establecer una estimación del coste de desarrollo antes de llevarlo a cabo, para después poder establecer una comparación con el coste real, observar y analizar las desviaciones.

Para estimar el coste salarial, se ha tomado como referencia el convenio que establece el Ministerio de Trabajo e Inmigración [34], que está en vigor desde 2009. En él se recoge que el salario por convenio de un Analista programador y Diseñador de página web es de 21.555,66€ anuales.

21555,66€ / 12 meses / 4 semanas/mes / 40 horas/semana = 11,23€ / hora.
11,23€/hora x 300 horas/hombre = 3369€

La razón por la que se escoge el perfil de Analista programador y Diseñador de página web, es que el proyecto involucra decisiones de análisis y diseño que escapan a las competencias de un Programador Senior, además debido a la metodología de trabajo empleada, no resulta factible diferenciar el coste de trabajadores con roles propios de cada fase, como podría ser el de Jefe de Proyecto o Administrador de test.

En cuanto al hardware, se emplea un *MacBook Air* de 13", cuyo coste de adquisición es de 1390€. En relación a la amortización de equipos informáticos, la Agencia Tributaria [35] establece un coeficiente lineal máximo del 25%, y un período máximo de 8 años.

1390€ x 25% = 347,5 €/año
Teniendo en cuenta que el proyecto se va a desarrollar durante medio año:
347,5 x 0,5 = 173,75 €

A continuación, se estiman los costes, sumando los ya detallados a otros como los de adquisición de dominios, software o licencias.

Estimación del coste
3369€ Salario del Analista Programador y Diseñador de página web
173,75€ Coste amortizado del equipo hardware empleado
12€ Coste anual de contratación de un dominio .com
7€/mes x 6 meses = 42€ Coste de contratar un servidor *dyno hobby* en Heroku
9€/mes x 6 meses = 54€ Coste de contratar los servicios privados de GitHub
Coste total: **3650,75€**

Para poder contar con un colchón de presupuesto con el que afrontar el riesgo, se toma un factor del 18% por el que multiplicar la cantidad estimada.

Factor multiplicativo
3650,75 x 1,18 = 4307,88€

Por tanto, la cifra final de presupuesto estimada para la consecución del proyecto es de 4307,88€.

Presupuesto final
4307,88€

CAPÍTULO 5. SEGUIMIENTO DEL PROYECTO.

5.1 Seguimiento temporal.

Para realizar el seguimiento del proyecto, se han empleado herramientas de monitorización del tiempo como Toggl, presentado en la sección 3.10.

En la Tabla 10 Seguimiento temporal, se detalla el seguimiento de los *sprints*.

Sprint	Tarea	Descripción Tarea	Estado	Complejidad	Horas/Hombre
EPIC 1		**Dashboard por defecto**			
1	T-001	Especificar de las historias de usuario y sus escenarios	Completado	2	5,5h
	T-002	Establecer presupuesto y calendarización	Completado	2	2,5h
	T-003	Elaborar plan de riesgos	Completado	2	3h
	T-005	Crear bocetos de la UI	Completado	2	5h
	T-004	Configurar el entorno de desarrollo	Completado	2	7h
	T-006	Crear modelo de dominio	Completado	1	2h
	T-007	Plantear la arquitectura	En progreso	2	1h
	T-008	Implementar la base del dashboard sobre la que disponer widgets	Pendiente	3	0h
2	T-007	Plantear la arquitectura	Completado	2	2,5h
	T-008	Implementar la base del dashboard sobre la que disponer widgets	Completado	3	16h
	T-009	Implementar la funcionalidad básica del widget de To-Dos	En progreso	3	11h
	T-010	Implementar la funcionalidad básica del widget de noticias RSS	Pendiente	3	0h
3	T-009	Implementar la funcionalidad básica del widget de To-Dos	Completado	3	5h
	T-010	Implementar la funcionalidad básica del widget de noticias RSS	Completado	3	14,5h
	T-011	Implementar la funcionalidad básica del widget de citas célebres	Completado	3	10h

4	T-012	Implementar la funcionalidad básica del widget de portadas	Completado	3	17h
	T-013	Implementar la funcionalidad básica del widget del tiempo	Completado	3	15,5h
	T-014	Implementar la funcionalidad básica del widget de bienvenida/fecha	Pendiente	2	0h
	T-015	Implementar registro de usuarios	Pendiente	3	0h
EPIC 2		**Dashboard con gestión de usuarios**			
5	T-014	Implementar la funcionalidad básica del widget de bienvenida/fecha	Completado	2	4,5h
	T-015	Implementar registro de usuarios	Completado	3	11h
	T-016	Implementar login/logout	Completado	3	12h
	T-017	Implementar recuperación de contraseña	En progreso	3	3,5h
6	T-017	Implementar recuperación de contraseña	Completado	3	8h
	T-018	Implementar modo edición del dashboard	Completado	2	6,5h
	T-019	Añadir configuración al widget de To-Dos	Completado	1	2,5h
	T-020	Añadir configuración al widget de noticias RSS	Completado	2	6,5h
	T-021	Añadir configuración al widget del tiempo	Completado	2	5,5h
EPIC 3		**Dashboard personalizable**			
7	T-022	Añadir configuración de tamaños a todos los widgets	Completado	2	8,5h
	T-023	Implementar modificación de la configuración de perfil	Completado	3	13h
	T-024	Implementar eliminación de widgets	Completado	2	7,5h
	T-025	Implementar creación dinámica de widgets	En progreso	3	2h
8	T-025	Implementar creación dinámica de widgets	Completado	3	16,5h
	T-026	Mejoras y ajustes visuales	Completado	3	10h
	T-027	Mejorar la documentación interna	Completado	2	3,5h

	T-028	Revisar y realizar más pruebas	Pendiente	3	0h
9	T-028	Revisar y realizar más pruebas	Completado	3	8h
	T-029	Documentar introducción, motivación y objetivos	Completado	1	2,5h
	T-030	Documentar el marco teórico y el tecnológico	Completado	3	14h
	T-031	Documentar el proyecto software	En progreso	3	14h
10	T-031	Documentar el proyecto software	Completado	3	21,5h
	T-032	Documentar las conclusiones y futuras mejoras	Completado	2	6h
	T-033	Revisar, corregir y mejorar la documentación.	Completado	2	5h
				Horas/ hombre totales	309,5h

TABLA 10 SEGUIMIENTO TEMPORAL

El proyecto se desarrolló estableciendo una *velocity* de valor 10, algunas tareas se alargaron en el tiempo más de lo previsto, pero gracias a un aumento del esfuerzo se ha podido seguir adecuadamente la calendarización, sin retrasos.

Horas/Hombre previstas
300h
Horas/Hombre reales
309,5h
Desviación
3%

El número total de horas empleadas ha sido de 309,5h, frente a las 300h previstas, esto supone una desviación del 3%, la cual resulta muy razonable y perfectamente asumible.

5.2 Costes reales.

En la sección 4.4, se detallaba la estimación inicial de los costes para el proyecto; a continuación, vamos a comparar los costes reales con los previstos en dicho apartado, y a obtener conclusiones sobre los resultados.

Teniendo en cuenta que se mantiene el coste por hora que se le paga al Analista Programador y Diseñador de página web, dado que ha habido una pequeña desviación hacia arriba en cuanto a las horas empleadas, obtenemos un sobrecoste por este lado.

11,23€/hora x 309,5 horas/hombre = 3475,69€ Coste salarial
Sobrecoste de 106,69€

El coste en cuanto a hardware se ha mantenido tal y como se había previsto, ya que no se han producido averías ni problemas que supusiesen aumentar el gasto previsto.

173,75€ Coste amortizado del equipo hardware empleado
Se ajusta a la previsión

Se había previsto obtener un dominio .com, pero no estaba disponible para el nombre elegido, así que finalmente se optó por un .co que resultó más barato.

7,5€ Coste anual de contratación de un dominio .co
Ahorro de 4,5€ sobre lo previsto

En cuanto a los servicios de Heroku, se decidió hacer uso de la versión gratuita, ya que proveía funcionalidad suficiente como para permitir sin problemas el desarrollo de la aplicación. Además, los servicios de GitHub pudieron ser obtenidos de manera gratuita gracias a su plan destinado a estudiantes, suponiendo también un ahorro.

0€/mes x 6 meses = 0€ Coste de contratar un servidor dyno free en Heroku
0€/mes x 6 meses = 0€ Coste de contratar los servicios privados de GitHub para estudiantes
Ahorro de 96€

El coste final del proyecto ha sido de 3656,94€, por tanto, ha sido muy similar al coste estimado, y se ha ajustado sin sobrecostes al presupuesto calculado de 4307,88€.

A pesar del pequeño sobrecoste salarial, también se consiguieron ahorros en cuanto a pagos de servicios, en cualquier caso, dado que se estableció un colchón de gasto al aplicar un factor multiplicativo, el presupuesto estaba adaptado para tolerar sobrecostes, como es natural que puedan darse en este tipo de proyectos.

Coste final:
3656,94€

CAPÍTULO 6. ESPECIFICACIÓN DE HISTORIAS DE USUARIO

Debido a la metodología escogida, los requisitos han sido recogidos en forma de historias de usuario. En este capítulo se detallan dichas historias junto a sus escenarios asociados. Las historias de usuario se describen usando Gherkin como parte de la aplicación de BDD.

6.1 Especificación de historias de usuario.

US-001	Registrarse en la aplicación
Definición	
Como usuario Quiero poder registrarme en la aplicación Para poder acceder a las funcionalidades	
Antecedentes	
Dado que estoy en la página de registro Dado que no estoy identificado	
Escenario:	Registro satisfactorio de un usuario
Dado que escribo mi email en el formulario Dado que introduzco mi contraseña en el formulario Dado que introduzco mi nombre en el formulario Cuando envío el formulario Entonces es comprobado como válido Y debería ser redirigido a mi dashboard	
Escenario:	Registro insatisfactorio cuando ya existe un usuario con el mismo email
Dado que escribo mi email en el formulario Dado que introduzco mi contraseña en el formulario Dado que introduzco mi nombre en el formulario Dado que ya existe un usuario con mi email en la aplicación Cuando envío el formulario Entonces es comprobado como inválido	
Escenario:	Registro insatisfactorio cuando la contraseña es demasiado corta
Dado que escribo mi email en el formulario Dado que introduzco una contraseña menor de 6 caracteres en el formulario Dado que introduzco mi nombre en el formulario Cuando envío el formulario Entonces es comprobado como inválido	

TABLA 11 - US REGISTRARSE EN LA APLICACIÓN

US-002	Añadir un widget al dashboard
Definición	
Como usuario registrado Quiero añadir un widget al dashboard Para poder visualizarlo y hacer uso de él	
Antecedentes	
Dado que estoy identificado en la aplicación Dado estoy en el modo edición del dashboard	
Escenario:	Añadir widget satisfactoriamente
Cuando selecciono añadir un nuevo widget Entonces deberían aparecer las opciones de configuración del widget E introduzco la configuración deseada Y el widget debería ser visible en el dashboard Y selecciono la opción guardar Entonces el widget debería permanecer en el dashboard	
Escenario:	Cancelar al añadir widget
Cuando selecciono añadir un nuevo widget Entonces deberían aparecer las opciones de configuración del widget E introduzco la configuración deseada Y pulso cancelar Entonces el widget no debería ser visible en el dashboard Y los cambios no deberían persistir	

TABLA 12 - US AÑADIR UN WIDGET AL DASHBOARD

US-003	Eliminar un widget del dashboard
Definición	
Como usuario registrado Quiero poder eliminar un widget del dashboard Para poder dejar de visualizarlo y borrar sus datos	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que el widget está en el dashboard	
Escenario:	Eliminar widget satisfactoriamente
Dado que estoy en el modo edición del dashboard Cuando selecciono borrar un widget Entonces debería desaparecer del dashboard	

TABLA 13 - US ELIMINAR UN WIDGET DEL DASHBOARD

US-004	Modificar configuración de un widget
Definición	
Como usuario registrado Quiero poder modificar un widget del dashboard Para poder cambiar la configuración de dicho widget	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que el widget está en el dashboard	
Escenario:	Modificar widget satisfactoriamente
Dado estoy en la página de edición del dashboard Dado que he seleccionado un widget a modificar Dado que he cambiado algún dato de su configuración Cuando guardo la nueva configuración Entonces los cambios deben mostrarse y persistir	
Escenario:	Modificar widget insatisfactoriamente
Dado estoy en la página de edición del dashboard Dado que he seleccionado un widget a modificar Dado que he cambiado algún dato de su configuración por otro de tipo inválido Cuando guardo la nueva configuración Entonces debería aparecer un mensaje informando del error	

TABLA 14 - US MODIFICAR CONFIGURACIÓN DE UN WIDGET

US-005	Mover un widget en el dashboard
Definición	
Como usuario registrado Quiero poder mover un widget en el dashboard Para poder cambiar la posición de dicho widget	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que el widget está en el dashboard Dado estoy en el modo edición del dashboard	
Escenario:	Mover un widget satisfactoriamente
Cuando arrastro un widget a otra posición dentro de la ventana del dashboard Entonces debería mostrarse cómo quedaría dispuesto el widget en dicha posición Y el widget permanecerá en la nueva posición	
Escenario:	Mover un widget insatisfactoriamente
Cuando arrastro un widget a otra posición fuera de la ventana del dashboard Entonces el widget debería permanecer en el mismo sitio	

TABLA 15 - US MOVER UN WIDGET EN EL DASHBOARD

US-006	Loguearse en la aplicación
Definición	
Como usuario Quiero poder loguearme en la aplicación Para poder ver mi configuración y mis datos	
Antecedentes	
Dado que no estoy identificado en la aplicación Dado que estoy en la página de login	
Escenario:	Hacer login satisfactoriamente
Dado que mi email está en el formulario de login Dado que mi contraseña está en el formulario de login Cuando envío el formulario Entonces es comprobado como válido	
Escenario:	Hacer login con email inexistente
Dado que un email no registrado está en el formulario de login Dado que mi contraseña está en el formulario de login Cuando envío el formulario Entonces es comprobado como inválido Y se muestra un mensaje informando del error	
Escenario:	Hacer login con contraseña incorrecta
Dado que mi email está en el formulario de login Dado que una contraseña que no es la mía está en el formulario de login Cuando envío el formulario Entonces es comprobado como inválido Y se muestra un mensaje informando del error	
Escenario:	Hacer login sin introducir email
Dado que mi contraseña está en el formulario de login Cuando envío el formulario Entonces es comprobado como inválido Y se muestra un mensaje informando del error	
Escenario:	Hacer login sin introducir contraseña
Dado que mi email está en el formulario de login Cuando envío el formulario Entonces es comprobado como inválido Y se muestra un mensaje informando del error	

TABLA 16 - US LOGUEARSE EN LA APLICACIÓN

US-007	Recuperar contraseña
Definición	
Como usuario Quiero poder recuperar mi contraseña Para poder recuperar el acceso a mi cuenta	
Antecedentes	
Dado que no estoy identificado en la aplicación Dado que estoy en la página de login	
Escenario:	Recuperar contraseña satisfactoriamente
Dado que he elegido recuperar contraseña Cuando introduzco mi email Entonces se me informa de que voy a recibir un correo para recuperar la contraseña	

TABLA 17 - US RECUPERAR CONTRASEÑA

US-008	Modificar perfil
Definición	
Como usuario Quiero poder modificar mi email o mi nombre en mi perfil Para poder usar unos datos distintos	
Antecedentes	
Dado que estoy identificado en la aplicación	
Escenario:	Modificar perfil satisfactoriamente
Dado que estoy en la página de configuración del perfil Cuando modifico mi email o mi nombre y guardo los cambios Entonces los cambios deberían hacerse efectivos y yo debería ser informado	

TABLA 18 - US MODIFICAR PERFIL

US-009	Cambiar contraseña desde la configuración
Definición	
Como usuario Quiero poder modificar mi contraseña Para emplear una contraseña distintita a la anterior	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en la página de configuración del perfil	
Escenario:	Cambiar contraseña satisfactoriamente
Dado que la nueva contraseña está en el formulario Dado que la verificación de contraseña está en el formulario Dado que las contraseñas coinciden Cuando envío el formulario Entonces la contraseña debería quedar modificada y yo ser informado	
Escenario:	Cambiar contraseña sin que coincida la verificación
Dado que la nueva contraseña está en el formulario Dado que la verificación de contraseña está en el formulario Dado que las contraseñas no coinciden Cuando envío el formulario Entonces debería mostrarse un mensaje de error	

TABLA 19 - CAMBIAR CONTRASEÑA DESDE LA CONFIGURACIÓN

US-010	Añadir tarea a un widget de lista de tareas
Definición	
Como usuario Quiero poder añadir una nueva tarea a un widget de lista de tareas Para que quede registrada	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el dashboard	
Escenario:	Añadir tarea satisfactoriamente
Dado que la nueva tarea está en el formulario del widget Cuando envío el formulario Entonces la tarea debería mostrarse en el listado widget	

TABLA 20 - US AÑADIR TAREA A UN WIDGET DE LISTA DE TAREAS

US-011	Marcar como completada tarea de un widget de lista de tareas
Definición	
Como usuario Quiero poder marcar como completada una tarea de un widget de lista de tareas Para que pueda recordar que está hecha	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el dashboard	
Escenario:	Marcar tarea completada satisfactoriamente
Dado que la tarea está en el listado del widget Dado que la tarea está marcada como no completada Cuando la marco como completada Entonces debería mostrarse su compleción	

TABLA 21 - US MARCAR COMO COMPLETADA TAREA DE UN WIDGET DE LISTA DE TAREAS

US-012	Marcar como no completada tarea de un widget de lista de tareas
Definición	
Como usuario Quiero poder marcar como no completada una tarea de un widget de lista de tareas Para que pueda recordar que no está hecha	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el dashboard	
Escenario:	Marcar tarea no completada satisfactoriamente
Dado que la tarea está en el listado del widget Dado que la tarea está marcada como completada Cuando la desmarco como completada Entonces debería dejar de mostrarse su compleción	

TABLA 22 - US MARCAR COMO NO COMPLETADA TAREA DE UN WIDGET DE LISTA DE TAREAS

US-013	Eliminar tarea de un widget de lista de tareas
Definición	
Como usuario Quiero poder eliminar una tarea de un widget de lista de tareas Para que desaparezca de forma definitiva	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el dashboard	
Escenario:	Eliminar tarea satisfactoriamente
Dado que la tarea está en el listado del widget Cuando selecciono eliminar Entonces debería desaparecer del widget	

TABLA 23 - US ELIMINAR TAREA DE UN WIDGET DE LISTA DE TAREAS

US-014	Leer noticia del widget de noticias RSS
Definición	
Como usuario Quiero poder leer una noticia de un widget de noticias RSS Para poder ver su contenido	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el dashboard	
Escenario:	Leer noticia satisfactoriamente
Dado que la noticia está en el widget de noticias RSS Cuando abro la noticia Entonces debería poder ver su contenido	

TABLA 24 - US LEER NOTICIA DEL WIDGET DE NOTICIAS RSS

US-015	Ver portada del widget de portadas
Definición	
Como usuario Quiero poder ver una portada de un widget de portadas Para poder ver su contenido	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el dashboard	
Escenario:	Ver portada satisfactoriamente
Dado que la portada está en el widget de noticias RSS Cuando abro la portada Entonces debería poder ver su contenido	

TABLA 25 - US VER PORTADA DEL WIDGET DE PORTADAS

US-016	Añadir un feed al widget de noticias RSS
Definición	
Como usuario Quiero poder añadir una fuente feed a un widget de noticias RSS Para poder recibir su contenido	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el modo edición del dashboard	
Escenario:	Añadir feed satisfactoriamente
Dado que estoy editando la configuración de un widget de noticias RSS Cuando añado la URL del feed Entonces debería quedar almacenada Y el contenido del feed debería mostrarse en el widget	
Escenario:	Añadir feed con URL inválida
Dado que estoy editando la configuración de un widget de noticias RSS Cuando añado una URL mal formada de un feed Entonces debería mostrarse un mensaje de error Y la URL no debería quedar almacenada	

TABLA 26 - US AÑADIR UN FEED AL WIDGET DE NOTICIAS RSS

US-017	Eliminar un feed de un widget de noticias RSS
Definición	
Como usuario Quiero poder añadir eliminar una fuente feed de un widget de noticias RSS Para dejar de recibir su contenido	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el modo edición del dashboard	
Escenario:	Añadir feed satisfactoriamente
Dado que estoy editando la configuración de un widget de noticias RSS Cuando marco eliminar la URL del feed Entonces la URL debería dejar de estar almacenada Y el contenido del feed no debería mostrarse en el widget	

TABLA 27 - US ELIMINAR UN FEED DE UN WIDGET DE NOTICIAS RSS

US-018	Modificar el nombre de un widget de lista de tareas
Definición	
Como usuario Quiero poder modificar el nombre de un widget de lista de tareas Para poder diferenciar más fácilmente unas listas de otras	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el modo edición del dashboard	
Escenario:	Añadir feed satisfactoriamente
Dado que estoy editando la configuración de un widget de lista de tareas Dado que el nombre nuevo está en el formulario Cuando envío el formulario Entonces el widget debería mostrar el nuevo nombre	

TABLA 28 - US MODIFICAR EL NOMBRE DE UN WIDGET DE LISTA DE TAREAS

US-019	Modificar el nombre de un widget de noticias RSS
Definición	
Como usuario Quiero poder modificar el nombre de un widget de noticias RSS Para poder diferenciar más fácilmente unas noticias de otras	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el modo edición del dashboard	
Escenario:	Añadir feed satisfactoriamente
Dado que estoy editando la configuración de un widget de noticias RSS Dado que el nombre nuevo está en el formulario Cuando envío el formulario Entonces el widget debería mostrar el nuevo nombre	

TABLA 29 - US MODIFICAR EL NOMBRE DE UN WIDGET DE NOTICIAS RSS

US-020	Modificar las unidades de un widget del tiempo
Definición	
Como usuario Quiero poder modificar las unidades de un widget del tiempo Para poder ver las temperaturas en mi sistema métrico preferido	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el modo edición del dashboard	
Escenario:	Seleccionar sistema imperial satisfactoriamente
Dado que estoy editando la configuración de un widget del tiempo Dado que el sistema actual es el sistema métrico Cuando selecciono el sistema imperial Entonces el widget debería mostrar las unidades en sistema imperial (Farenheit)	
Escenario:	Seleccionar sistema métrico satisfactoriamente
Dado que estoy editando la configuración de un widget del tiempo Dado que el sistema actual es el sistema imperial Cuando selecciono el sistema métrico Entonces el widget debería mostrar las unidades en sistema métrico (Celsius)	

TABLA 30 US - MODIFICAR LAS UNIDADES DE UN WIDGET DEL TIEMPO

US-021	Modificar la ubicación sobre la que mostrar la previsión de un widget del tiempo
Definición	
Como usuario Quiero poder modificar la ubicación sobre la que mostrar la previsión de un widget del tiempo Para poder ver la previsión para mi ubicación deseada	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el modo edición del dashboard	
Escenario:	Introducir ubicación satisfactoriamente
Dado que estoy editando la configuración de un widget del tiempo Dado que la longitud en coordenadas está en el formulario Dado que la latitud en coordenadas está en el formulario Cuando envío el formulario Entonces el widget debería mostrar la previsión para la nueva ubicación	
Escenario:	Introducir ubicación con coordenadas inválidas
Dado que estoy editando la configuración de un widget del tiempo Dado que una longitud inválida en coordenadas está en el formulario Dado que la latitud en coordenadas está en el formulario Cuando envío el formulario Entonces debería informarse del error y la ubicación no debería cambiar	

TABLA 31 US - MODIFICAR LA UBICACIÓN SOBRE LA QUE MOSTRAR LA PREVISIÓN DE UN WIDGET DEL TIEMPO

US-022	Consultar la previsión del tiempo
Definición	
Como usuario Quiero poder consultar la previsión del tiempo Para poder obtener información relevante	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el dashboard	
Escenario:	Añadir feed satisfactoriamente
Dado que tengo un widget de previsión del tiempo añadido y configurado Cuando entro en el dashboard Entonces el widget debería mostrar la previsión del tiempo	

TABLA 32 - US CONSULTAR PREVISIÓN DEL TIEMPO

US-023	Leer una cita célebre
Definición	
Como usuario Quiero poder leer una cita célebre Para poder obtener información relevante	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el dashboard	
Escenario:	Añadir feed satisfactoriamente
Dado que tengo un widget de citas célebres añadido Cuando entro en el dashboard Entonces el widget debería mostrar la cita célebre	

TABLA 33 - US LEER UNA CITA CÉLEBRE

US-024	Ver la fecha actual
Definición	
Como usuario Quiero poder modificar ver la fecha actual Para poder obtener información relevante	
Antecedentes	
Dado que estoy identificado en la aplicación Dado que estoy en el dashboard	
Escenario:	Añadir feed satisfactoriamente
Dado que tengo un widget de bienvenida añadido Cuando entro en el dashboard Entonces el widget debería mostrar la fecha	

TABLA 34 - US VER LA FECHA ACTUAL

6.2 Modelo de dominio.

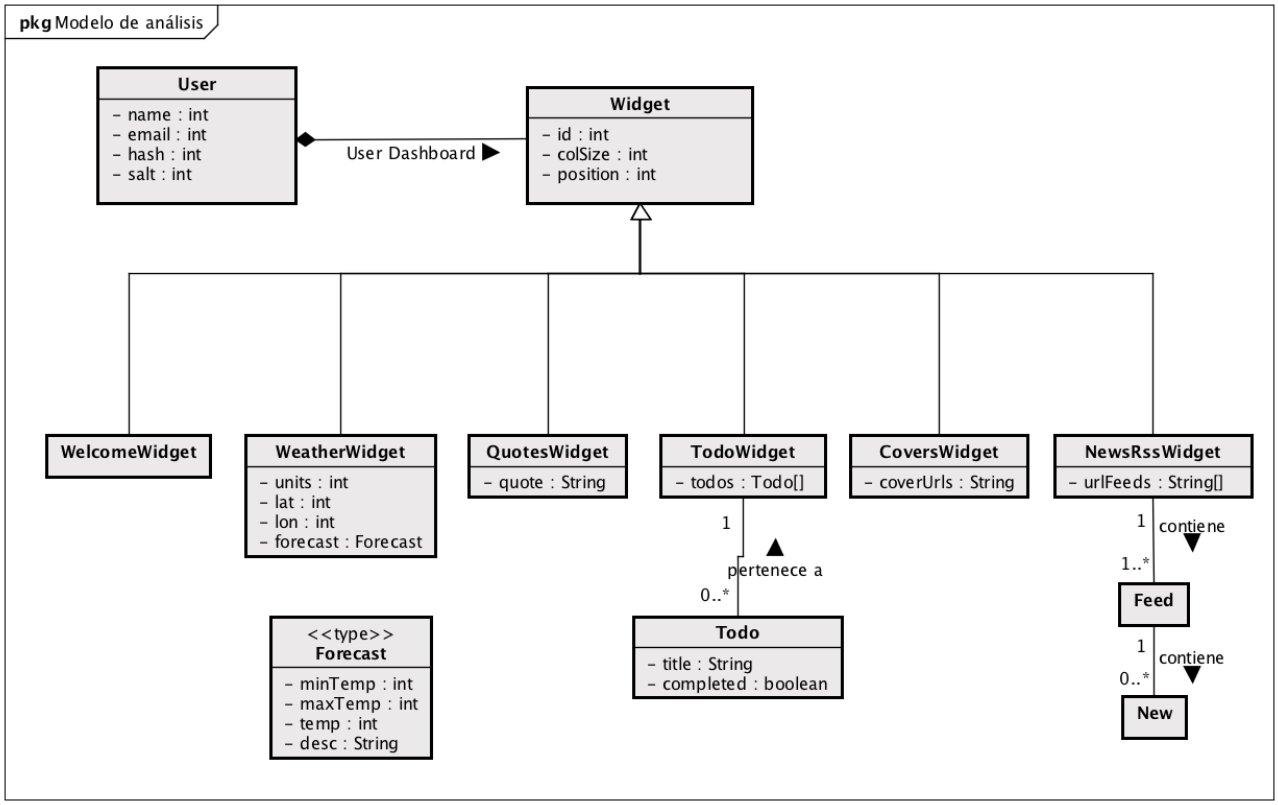

ILUSTRACIÓN 6 - MODELO DE DOMINIO

Se ha planteado el modelo que se muestra en la Ilustración 6, en el que se detallan los distintos *widgets* y su jerarquía de herencia.

La relación de un usuario con los widgets representa el *dashboard*, se trata de una composición fuerte dado que un componente (un *widget*) no puede ser compartido por varios compuestos (varios usuarios de *dashboard*).

Los diferentes tipos de *widget* tienen como padre una clase *Widget* que permite modelar los atributos comunes de todos.

CAPÍTULO 7 – DECISIONES DE DISEÑO.

7.1 Modelo arquitectónico.

La arquitectura de la aplicación es la propia de una aplicación web, de tipo cliente-servidor, pero tiene como diferenciación respecto a las webs clásicas que se trata de una SPA (*Single Page Application*).

La diferencia principal entre una SPA y una aplicación web clásica es dónde se realiza la lógica de negocio, ya que en las SPAs la implementación se encuentra casi íntegramente en el navegador, quedando el servidor relegado a realizar menos tareas y más básicas, especialmente de acceso a datos.

Uno de los principales problemas en cuanto a rendimiento de las aplicaciones web, hoy en día viene dado por el cuello de botella que suponen las latencias, y es que ya no es el ancho de banda lo que limita la experiencia de usuario [36]. Las SPAs tienen entre sus características la solución a este problema, dado que cuando hablamos de SPAs, el peso de la aplicación recae en el *frontend*, y todas las vistas están contenidas en la aplicación, realizando una única carga inicial, y posponiendo únicamente la carga de recursos pesados.

La ventaja de que la lógica se ejecute en el navegador del cliente es que se mejora la velocidad de carga, ya que, gracias a la eliminación de las latencias propias de la red, se obtienen tiempos de carga propios de aplicaciones locales. También se logra reducir el ancho de banda y la carga del servidor, por lo que la aplicación será mucho más escalable y a un menor coste.

El ancho de banda se reduce porque hay datos que se pueden mantener almacenados en el navegador, sin necesidad de estar transfiriéndolos y recuperándolos constantemente del servidor, a su vez la carga en cuanto a proceso por parte del servidor se ve muy reducida, porque gran parte de las rutinas de la lógica de negocio pueden ejecutarse independientemente en los clientes locales [37].

Concretamente, la arquitectura empleada en este proyecto es la propia de la pila MEAN (*Mongo Express Angular Node*). Esta pila se caracteriza por el empleo de Angular como *framework* en el *frontend*, mientras que el *backend* lo conforman Express.js, Node.js y MongoDB.

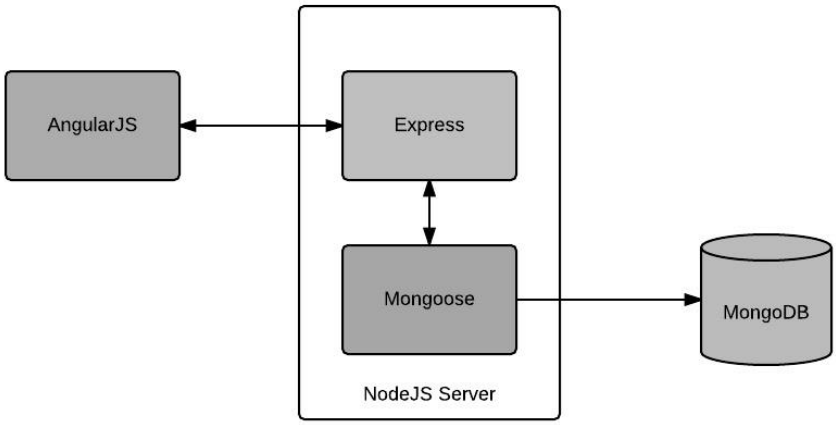

ILUSTRACIÓN 7 - ARQUITECTURA DE UNA APLICACIÓN MEAN. TOMADO DE [38].

7.2 Bocetos de la interfaz de usuario.

Se han planteado los bocetos de la interfaz de usuario mediante la herramienta *Balsamiq Mockups*, la cual permite seleccionar elementos visuales genéricos de entre una colección, para esbozar así las vistas de forma sencilla pero efectiva.

En la Ilustración 8, se muestra el diseño principal del *dashboard*. Por defecto se ha establecido una distribución por columnas, aunque el usuario será capaz de personalizar su propia vista como desee, eligiendo la posición y tamaño de los *widgets*.

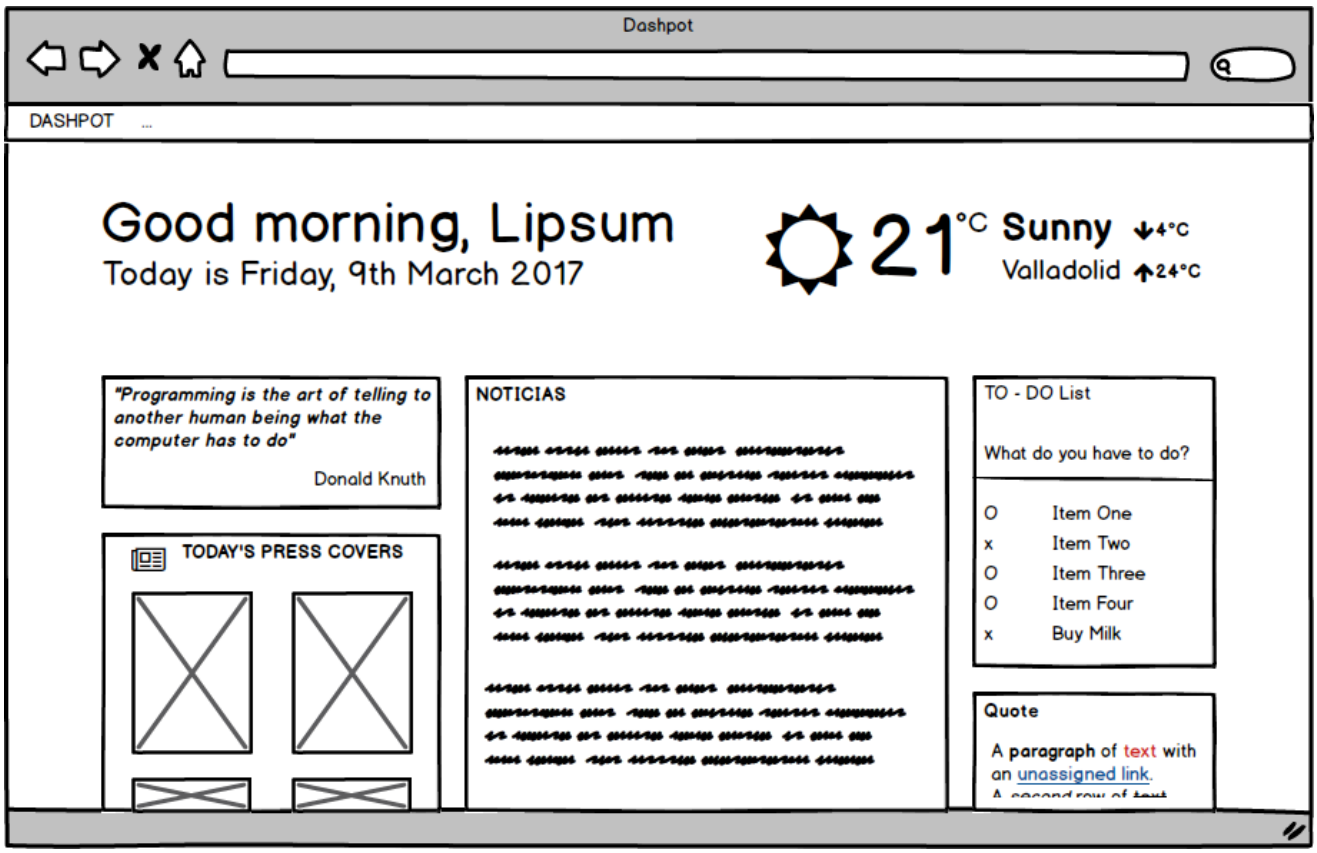

ILUSTRACIÓN 8 - BOCETO DE LA INTERFAZ DEL DASHBOARD

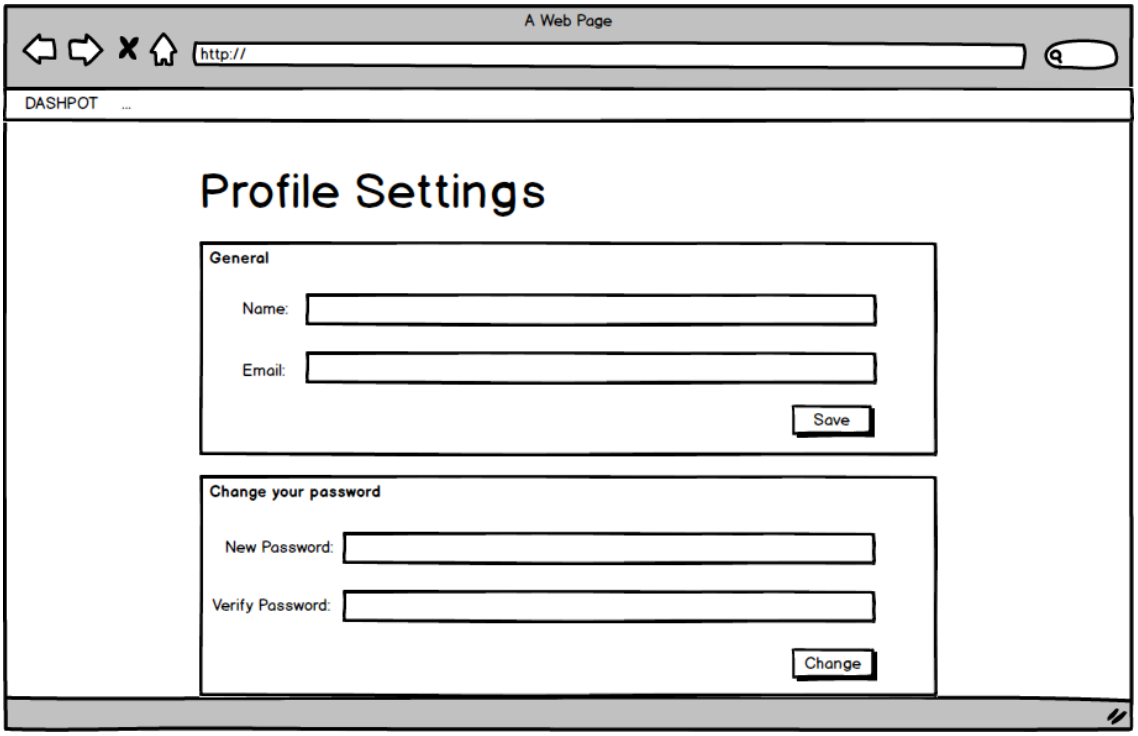

ILUSTRACIÓN 9 - BOCETO DE LA PÁGINA DE CONFIGURACIÓN DEL PERFIL

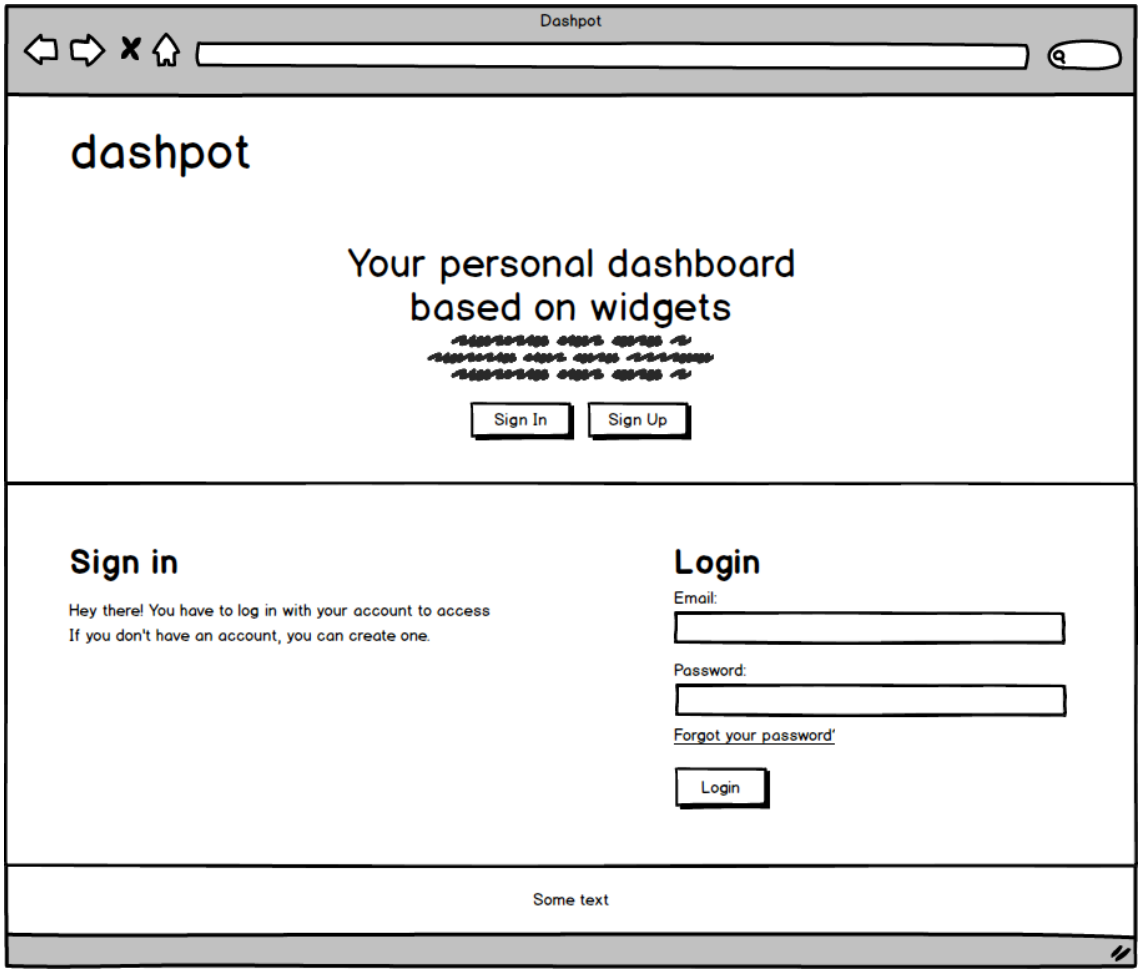

ILUSTRACIÓN 10 - BOCETO DE LA PÁGINA DE INICIO

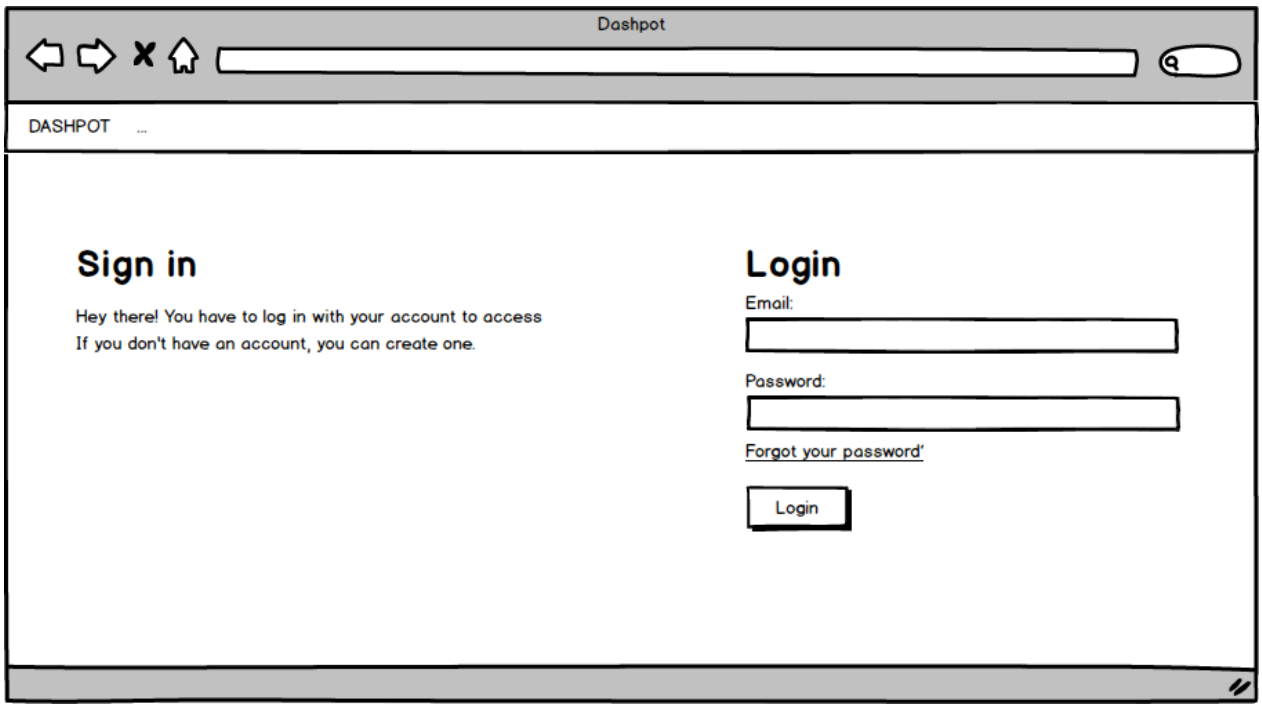

ILUSTRACIÓN 11 - BOCETO DE LA VISTA DE LA PÁGINA DE ACCESO

ILUSTRACIÓN 12 - BOCETO DE LA VISTA DE LA PÁGINA DE REGISTRO

7.3 Modelos de los widgets.

Para poder detallar la arquitectura interna de la aplicación, en este apartado se incluyen diagramas representativos del diseño referentes a la aplicación SPA del *frontend*, por tanto, todas las referencias al *dashboard* en esta sección, serán relativas a la parte del cliente.

7.3.1 Widget de listado de tareas.

En el diseño de todos los widgets se ha seguido el patrón MVC (Modelo-Vista-Controlador); en el caso del widget de listado de tareas, ver Ilustración 17, el controlador corresponde a la clase *TodoComponent*, la vista corresponde a la plantilla, escrita en HTML junto a los *bindings* propios de Angular, y el modelo está formado por el servicio *TodoDataService* y la clase *Todo*, la cual modela una tarea.

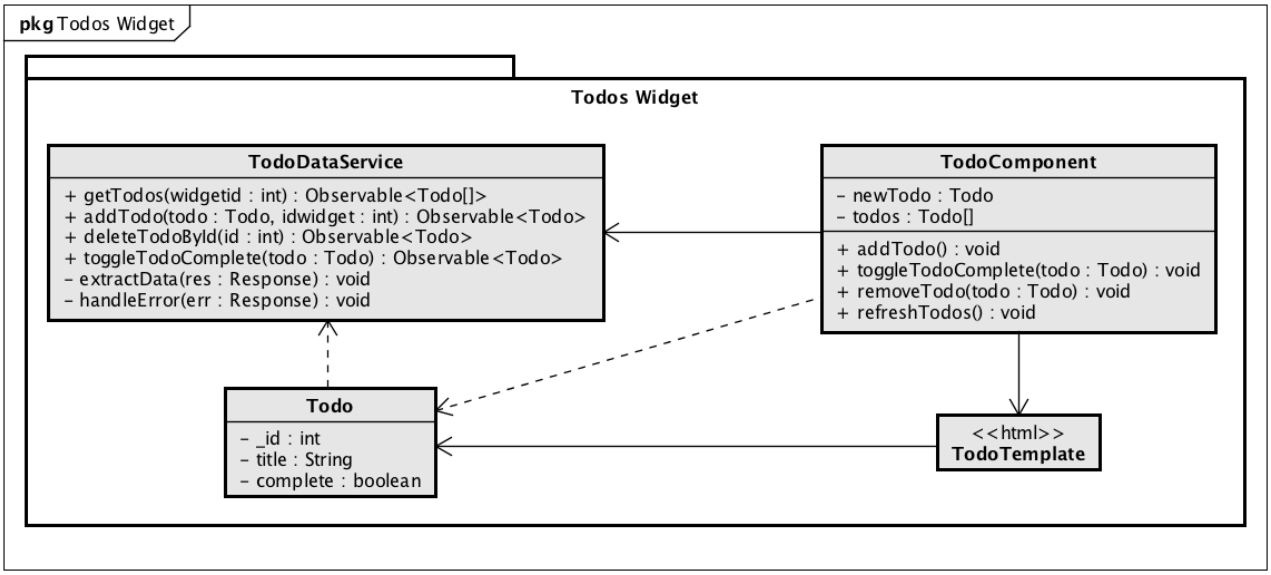

ILUSTRACIÓN 13 - ARQUITECTURA DEL WIDGET DE LISTA DE TAREAS

La clase controladora gestiona los eventos de añadir, marcar o desmarcar como completado, y borrar una tarea. Típicamente, lo que hace ante estos eventos es realizar una llamada al modelo, esto es la clase del servicio, suscribiéndose al *Observable*.

Cuando se modifican los datos de alguna de las propiedades del componente, en este caso, por ejemplo, cuando se modifica el *array* de todos que se encuentra como atributo en *TodoComponent*, el mecanismo de detección de cambios de Angular entra en acción y la vista vuelve a ser *renderizada* para mostrar los cambios.

7.3.2 Widget del tiempo.

De manera similar al widget de listado de tareas, el widget del tiempo tiene un controlador, *WeatherComponent*, que se asocia con la vista, *WeatherTemplate*, y el modelo, *WeatherService*. La clase *Forecast* es parte del modelo y permite representar una previsión del tiempo. La arquitectura del *widget* puede observarse en la Ilustración 14.

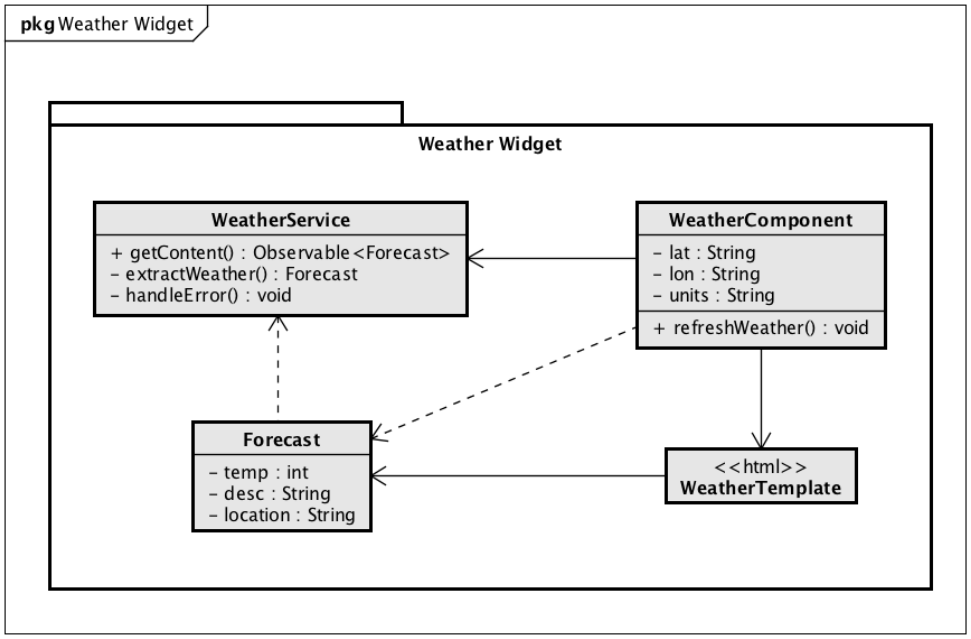

ILUSTRACIÓN 14 - ARQUITECTURA DEL WIDGET DEL TIEMPO

7.3.3 Widget de noticias.

El widget de noticias tiene una arquitectura muy similar a la de los widgets ya comentados, la peculiaridad que presenta es que aparece una clase *Feed* perteneciente al modelo, que es una composición de *FeedEntry*, de esta manera se logra que un mismo *widget* soporte múltiples *feeds* como fuentes, a la vez que se modelan las entradas de cada *feed*.

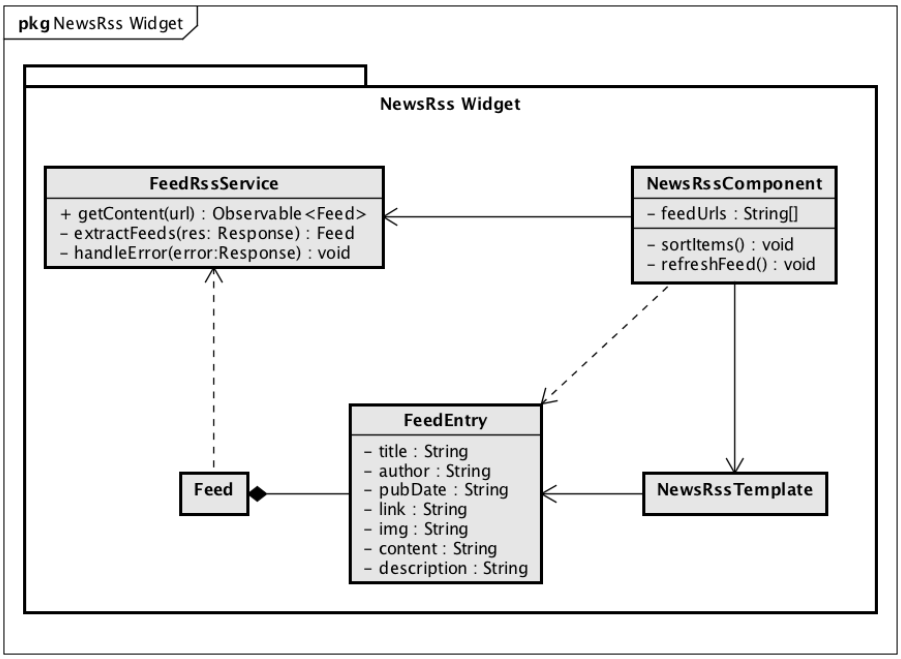

ILUSTRACIÓN 15 - ARQUITECTURA DEL WIDGET DE NOTICIAS

Los métodos privados *extractFeed()* y *handleError()* del servicio, se encargan respectivamente de extraer un objeto de tipo *Forecast* a partir del JSON contenido en la respuesta, y de gestionar los errores que puedan producirse.

La ordenación de las noticias la realiza el controlador, gracias al método *sortItems()*, que coloca los elementos comparando sus fechas, para mostrarlos de más reciente a menos reciente en cuanto a fecha de publicación.

7.3.4 Widget de citas célebres.

El widget de noticias es continuista con la arquitectura MVC que se ha ido mostrando a lo largo de los diagramas anteriores, representativos de los otros widgets. Observando la Ilustración 16, puede apreciarse que el controlador corresponde a la clase *QuotesComponent*, la vista a la clase *QuotesTemplate* y el modelo a las clases *QuotesService* y *Quote.*

Cuando el controlador recibe el evento de refrescar la cita, este llama al servicio y se suscribe al Observable, para modificar el valor de su atributo *quote*, y que Angular refresque la vista, volviéndola a renderizar cuando detecte el cambio.

Una cita, es decir, un objeto de la clase *Quote*, tendrá un texto y un autor, el controlador consigue construir uno de estos objetos llamando al método *getQuote()* del servicio del modelo, y lo hace en el momento de la creación del componente, gracias al hook *onInit()*, para que desde el momento en que cargue el *widget* ya haya una cita recuperada.

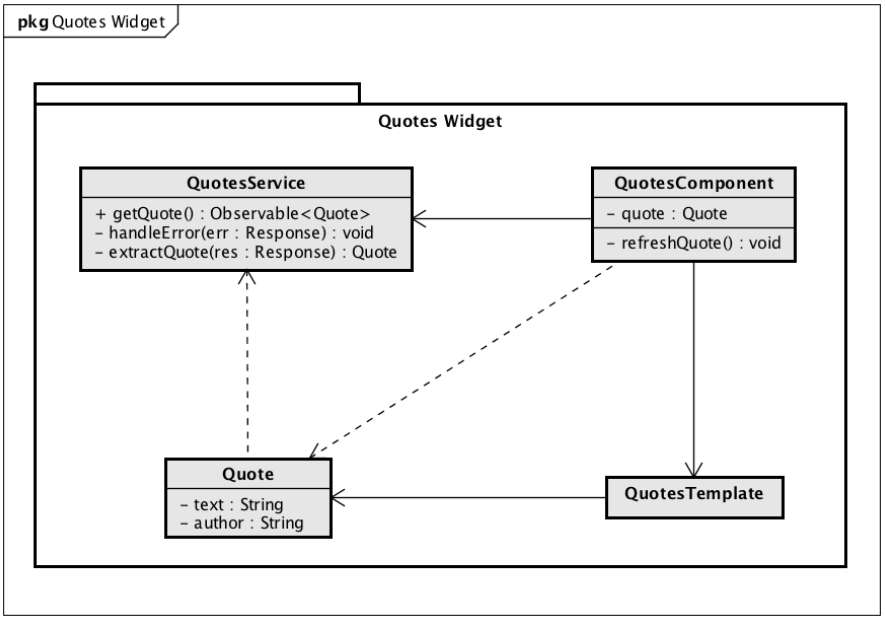

ILUSTRACIÓN 16: ARQUITECTURA DEL WIDGET DE CITAS

7.3.5 Widget de portadas.

En el widget de portadas, ver Ilustración 17, el controlador corresponde a *CoversComponent*, la vista a *CoversTemplate* y el modelo a la clase *Http*, que realiza las peticiones necesarias para comprobar la disponibilidad portadas.

El motivo por el que no se ha diseñado un servicio específico como el que aparece en otros widgets, es que las imágenes de las portadas se obtienen de un proveedor externo, y localmente sólo es necesario formar las *URLs* y comprobar la disponibilidad, por lo que se ha optado por una aproximación más simple al no necesitar entidades propias para modelar una portada.

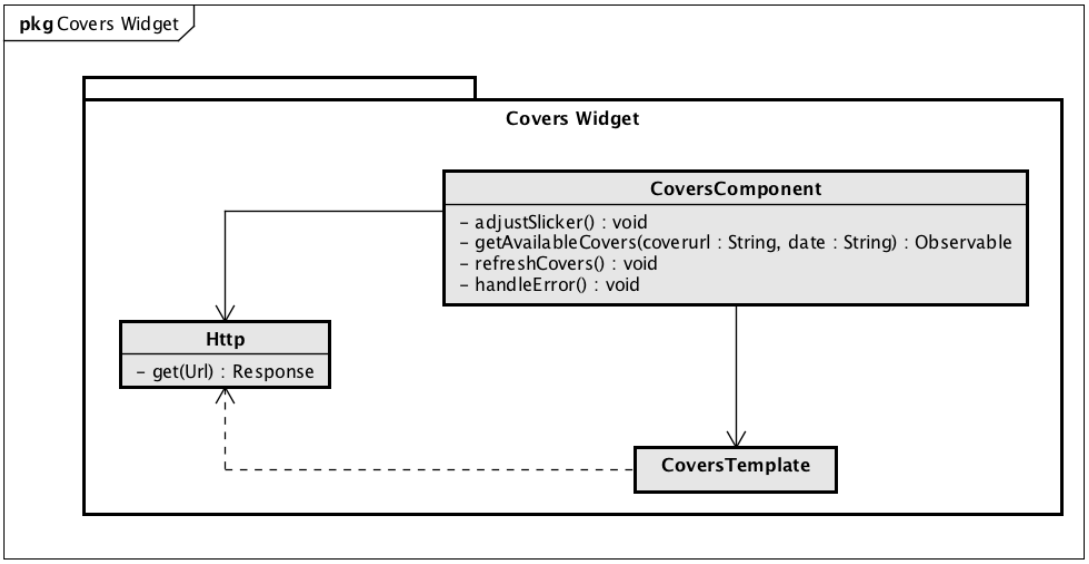

ILUSTRACIÓN 17 - ARQUITECTURA DEL WIDGET DE PORTADAS

7.3.6 Arquitectura del Dashboard.

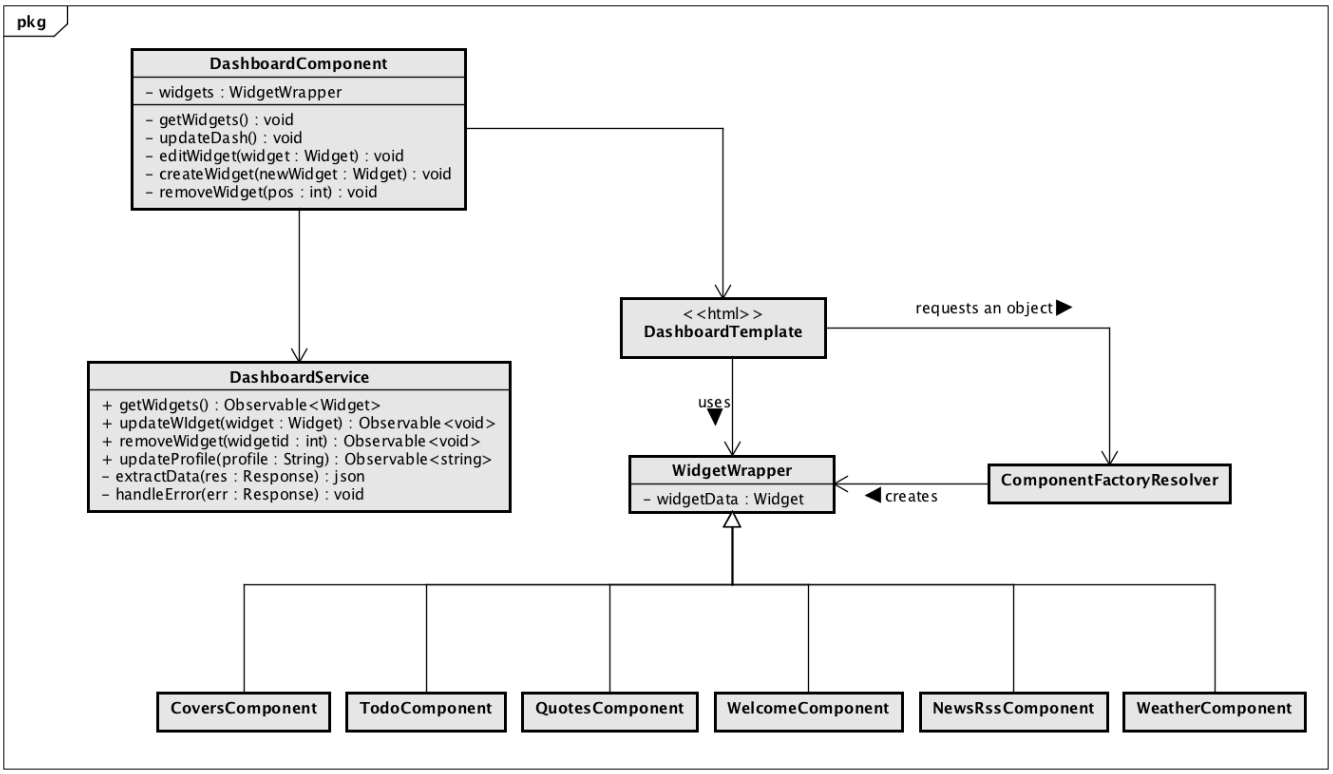

ILUSTRACIÓN 18 - ARQUITECTURA DEL DASHBOARD

El dashboard es un componente Angular, que contiene en su vista otros componentes Angular, que son los *widgets*. En la Ilustración 18, se modela la relación entre entre el dashboard y los *widgets*. Es importante recordar que se está detallando la aplicación SPA correspondiente al *frontend*, lo interesante de este diagrama es el uso del patrón factoría.

Una de las características del *dashboard* es que un usuario puede crear los *widgets* que desee, es por ello que no se conoce de antemano el tipo de los widgets asociados al *dashboard*, y es necesario un mecanismo para crearlos de forma dinámica. Para resolver esto, se ha utilizado el patrón software factoría abstracta, que permite fabricar los objetos del tipo adecuado bajo demanda, mediante encapsulación, permitiendo así construir *widgets* de forma dinámica, gracias a que la factoría *ComponentFactoryResolver* se encarga de construir los *widgets* del tipo correcto.

La clase *DashboardComponent* es la controladora, recibe los eventos de creación, modificación, actualización y borrado de los widgets, y realiza llamadas al modelo, es decir, a la clase *DashboardService*. La forma en que lo hace no se diferencia demasiado de la mostrada para los demás *widgets*, puesto que, recordemos que también se trata de un componente Angular.

7.3.7 Dependencias entre componentes y servicios.

Para tener una visión más global de las dependencias entre las clases *Component* (Controlador) y las clases *Service* (Modelo), se ha elaborado el diagrama de la Ilustración 19. En el que se muestran también las realizaciones de los *hooks* del ciclo de vida de Angular.

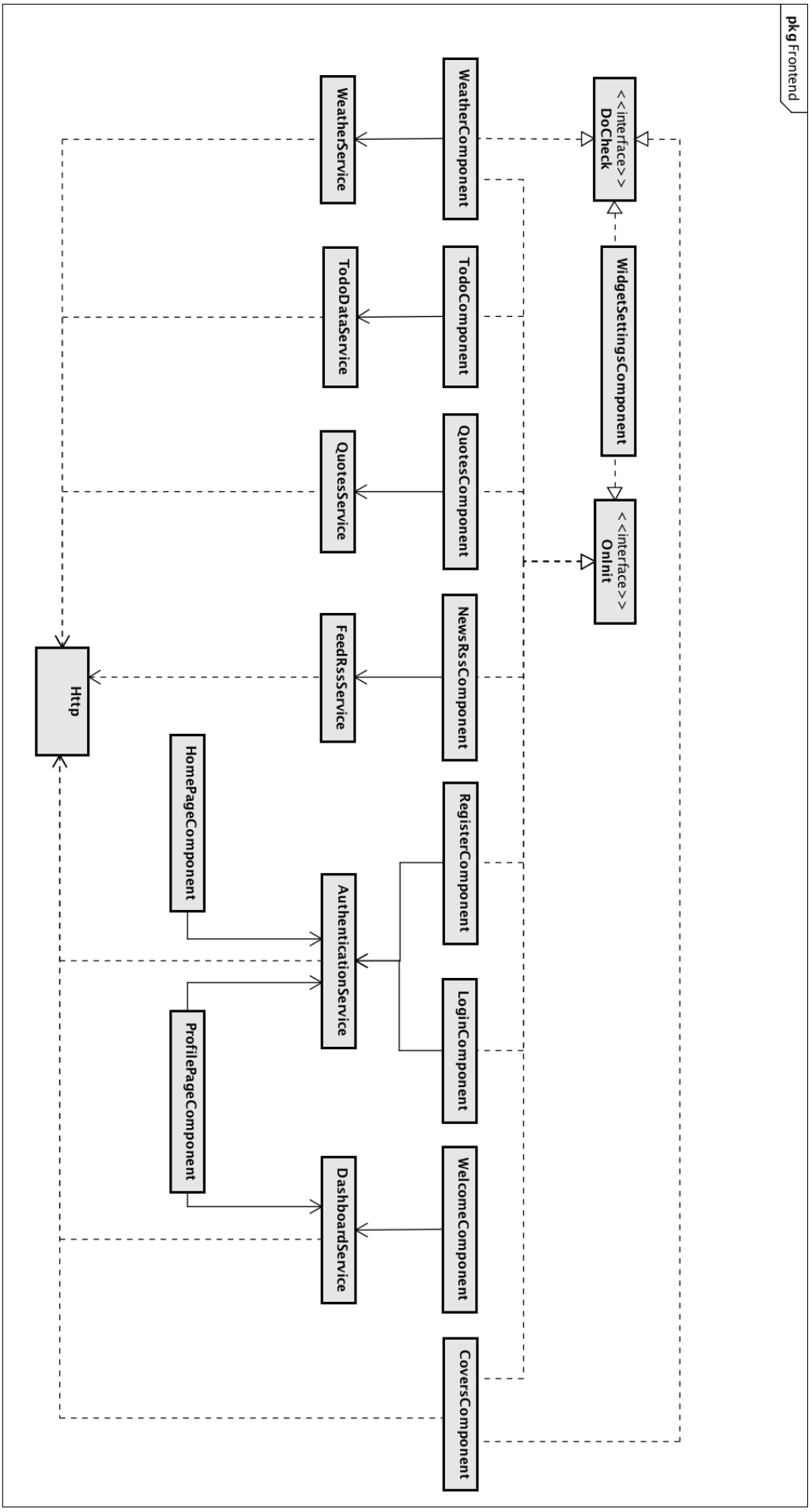

ILUSTRACIÓN 19 - DEPENDENCIAS ENTRE CLASES DEL DASHBOARD

7.4 Modelo del comportamiento.

En esta sección se incluyen algunos diagramas de secuencia, que permiten reflejar el comportamiento de la aplicación en cuanto al flujo de operaciones. En la Ilustración 20, se muestra el flujo para un caso genérico en que se quiere realizar una operación de datos CRUD cualquiera [39].

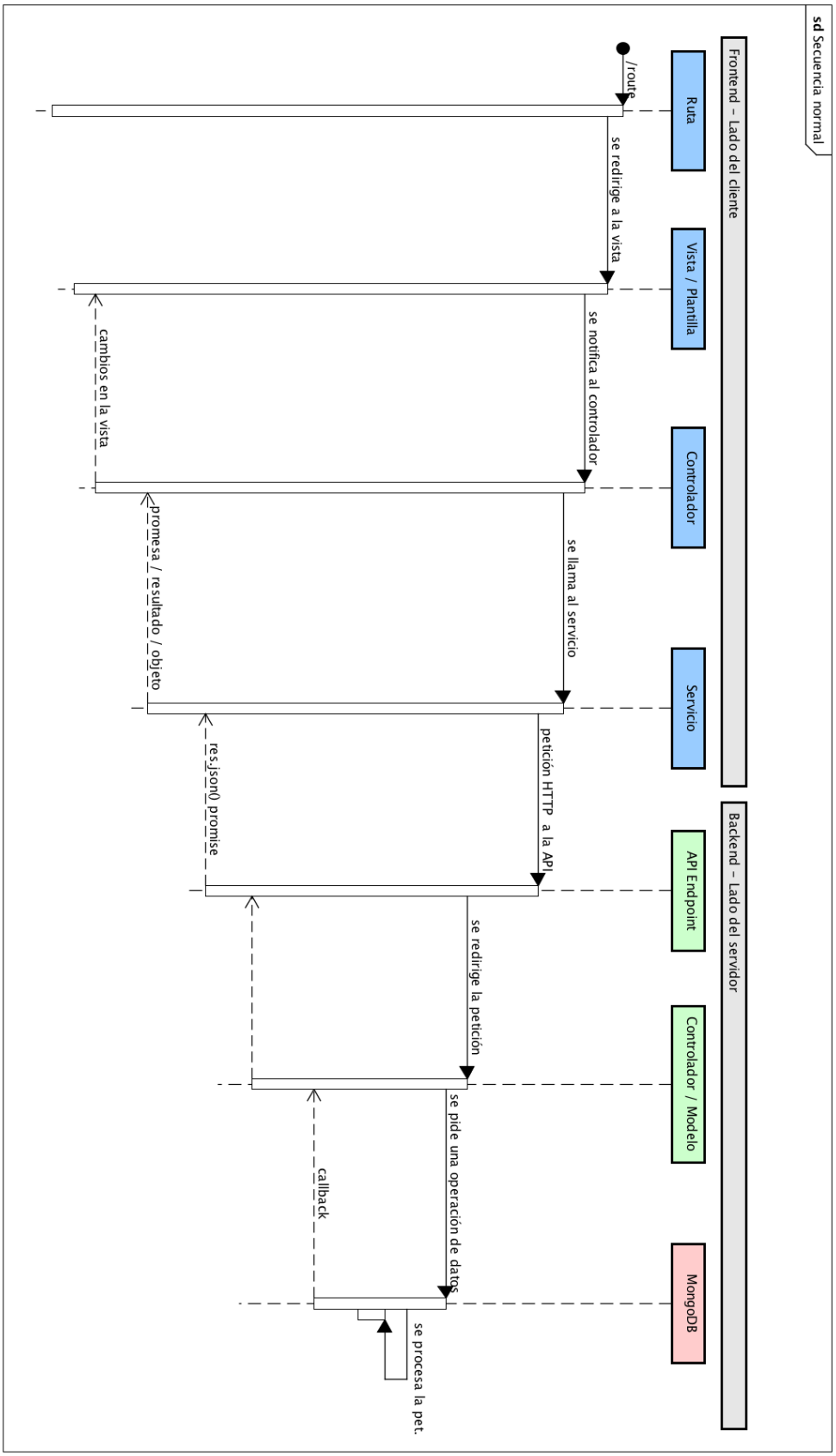

ILUSTRACIÓN 20 - DIAGRAMA DE SECUENCIA PARA UNA OPERACIÓN DE DATOS

El primer elemento que entra en juego es el router de Angular.js, para redirigir a la vista correcta. A continuación, la vista notifica al controlador de que requiere cambios, y éste se comunica con el servicio correspondiente, el cual realizará una petición HTTP a la API REST.

Cuando el servidor recibe una petición, ésta se dirige al controlador adecuado, el cual se encarga de requerir las operaciones necesarias a la base de datos y de responder a la petición, haciendo uso de los modelos, que se representan mediante *Schemas* de Mongoose.

Finalmente, el servicio del lado del cliente recibe respuesta del servidor, y a su vez responde al controlador mediante construcciones como *Observables* o promesas, para después actualizar la vista.

Para conseguir un mejor nivel de detalle, se han documentado los diagramas para los casos concretos de añadir una nueva tarea, correspodiente al widget de tareas, y obtener el el contenido de un *feed*, correspondiente al *widget* de noticias RSS.

El caso de la obtención del contenido de un *feed* se documenta en la Ilustración 18, este flujo de operaciones resulta de interés porque los datos se obtienen de una fuente externa, en lugar de obtenerlos del *backend* de la aplicación. Cuando se refrescan los *feeds*, se ejecuta un bucle que obtiene el contenido de cada uno de ellos, esto se logra gracias al servicio FeedRssService, que realiza peticiones HTTP al servicio externo mediante la clase Http, propia de Angular.

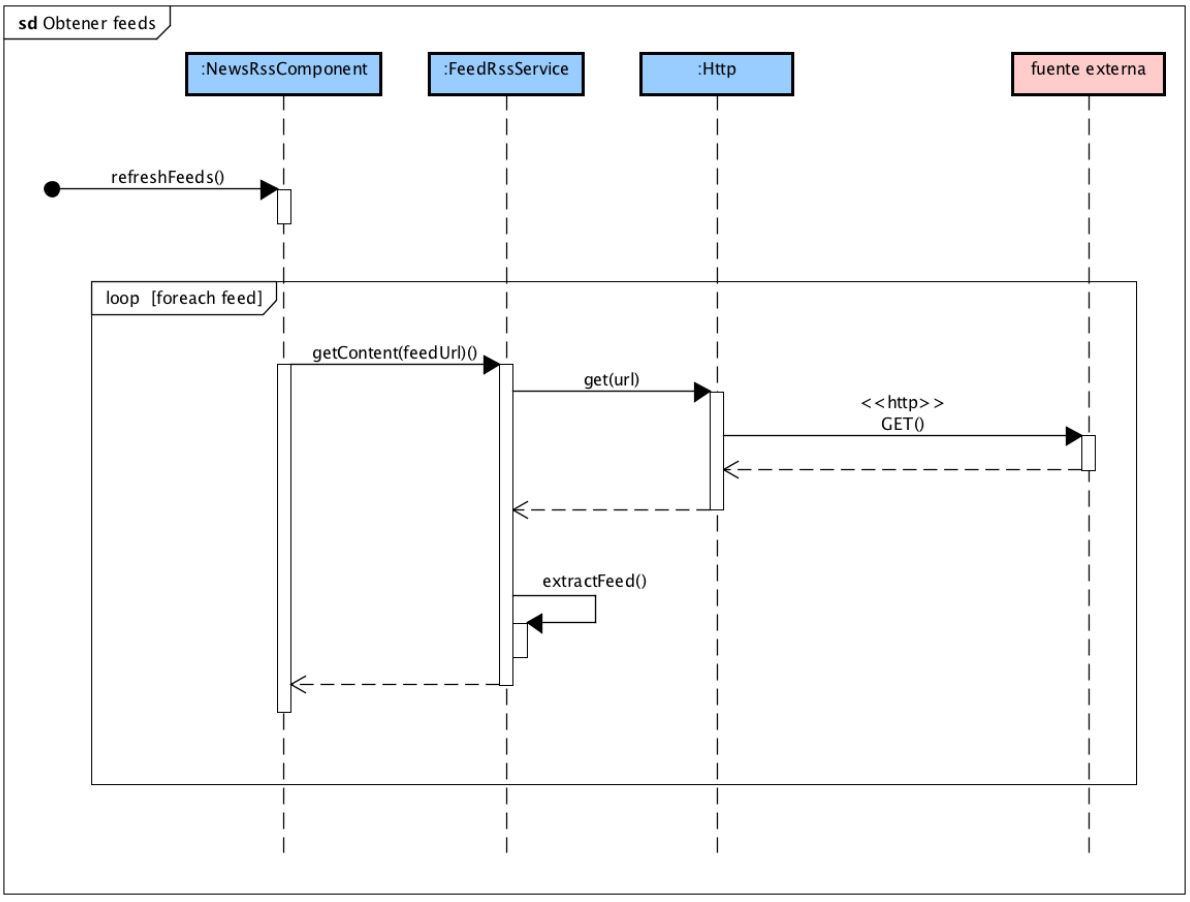

ILUSTRACIÓN 21 - DIAGRAMA DE SECUENCIA PARA LA OBTENCIÓN DE UN FEED

En el segundo caso de ejemplo, que se muestra en la Ilustración 22, se documenta la secuencia de operaciones que permiten la creación de una nueva tarea asociada a un widget de lista de tareas. Dicha secuencia es muy similar a la mostrada en la Ilustración 20, dado que se está realizando una operación de datos en la que interviene el *backend*.

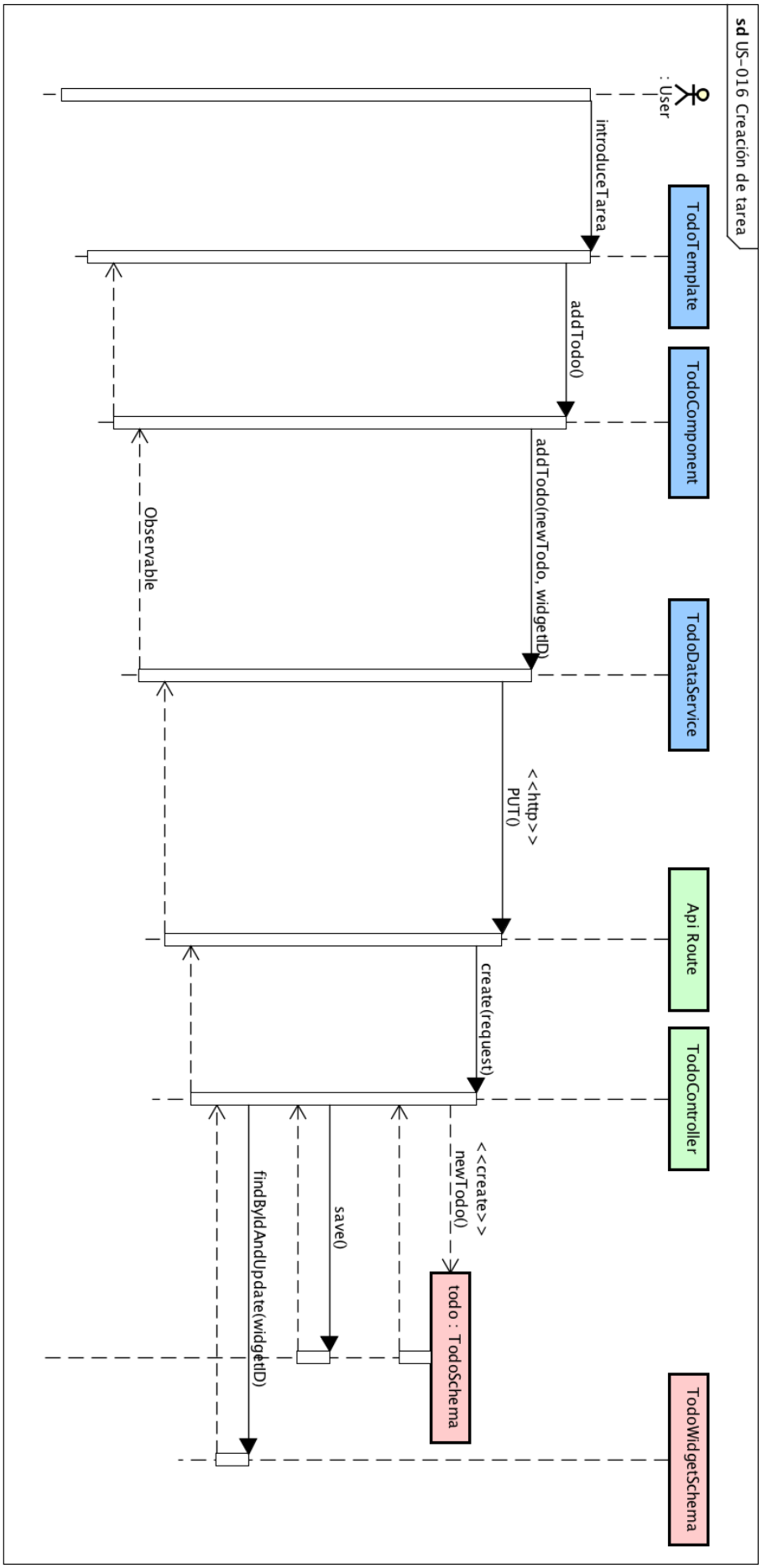

ILUSTRACIÓN 22 - SECUENCIA PARA LA CREACIÓN DE UNA TAREA

7.5 Diseño de los datos.

7.5.1 Modelo lógico.

Dado que la base de datos empleada es no relacional, para realizar un diagrama que modele su estructura lógica resulta más conveniente emplear un diagrama de clases que un diagrama entidad-relación, este se recoge en la Ilustración 23.

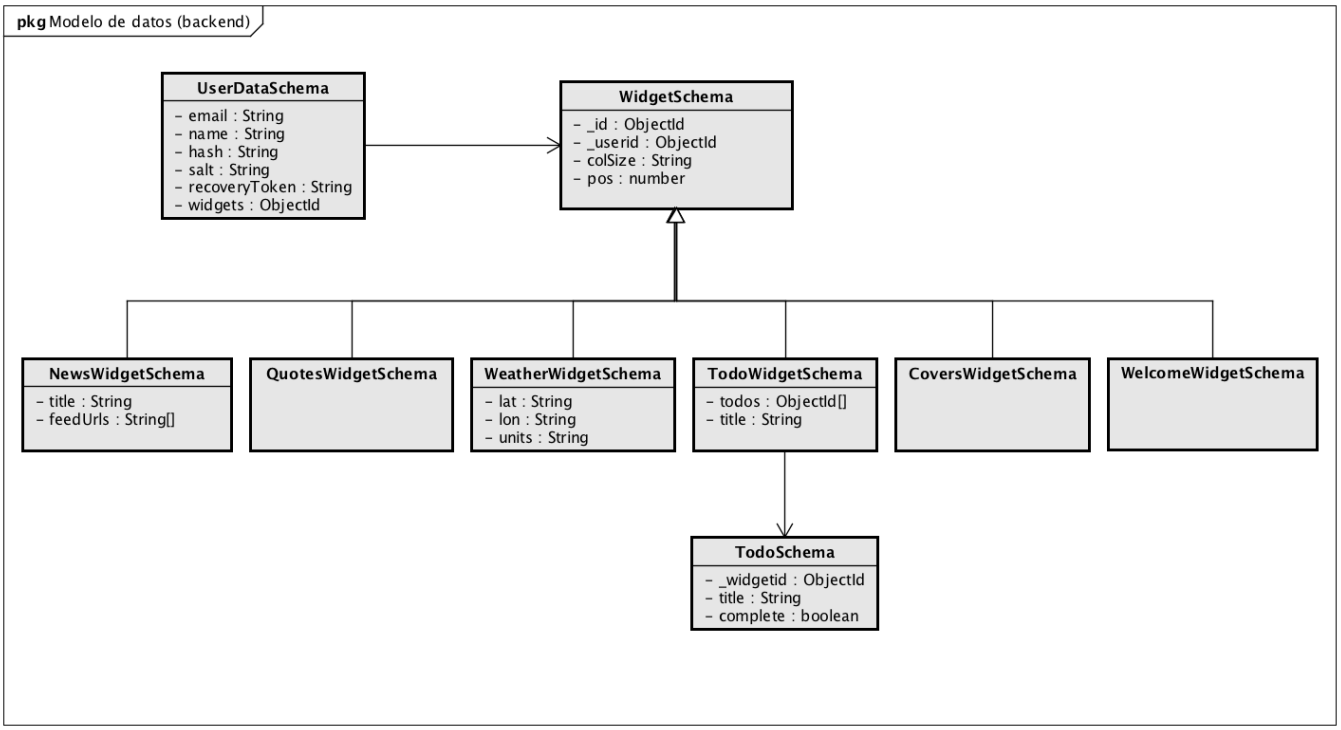

ILUSTRACIÓN 23 - MODELO LÓGICO DE DATOS

La estructura de la base de datos es bastante simple, sólo aparecen dos relaciones de tipo uno a muchos (1 a N), una de ellas es la que se da entre los widgets y un usuario, es decir, entre *UserDataSchema* y *WidgetSchema*, y la otra es la que aparece entre las tareas y un widget de lista de tareas, es decir, entre *TodoWidgetSchema* y *TodoSchema*.

Con respecto al modelo de dominio que se planteó en la Ilustración 6, cabe mencionar que las citas célebres (*quotes*), los *feeds* y la previsión del tiempo (*forecast*) no se ven reflejados en el modelo de datos porque se obtienen de servicios externos, y por tanto no persisten en la base de datos de la aplicación. Lo que se almacena en base de datos es la información relativa a usuarios, su configuración de *widgets* y sus listas de tareas.

7.5.2 Modelo físico.

A la hora de establecer el modelo físico de los datos resulta necesario implementar la herencia entre *widgets*, teniendo la clase *Widget* como padre, para poder tener un *schema* de Mongoose con los atributos comunes, y diferentes esquemas que extiendan al padre con sus propios atributos para cada tipo de *widget*.

La forma de implementar esto mediante Mongoose es el uso de *Discriminators*, los cuales permiten almacenar datos en *schemas* ligeramente distintos dentro de la misma colección.

De esta manera, se define un esquema base y junto a él los esquemas que lo extienden, y se usa la función *discriminator* para registrar los distintos modelos diferenciando por su tipo:

```
   function BaseSchema() {
     Schema.apply(this, arguments);

     this.add({
         _id : {
           type: ObjectId,
           required: true,
           unique: true
         },
         _userid : {
           type: ObjectId,
           ref: 'User'
         },
         colSize : {
           type: String,
           required: true
         },
         pos : {
           type: Number,
           required: true
         }
     });
   }
```

Para completar el ejemplo, así se definen los esquemas para los widgets del tiempo (*WeatherWidgetSchema*) y lista de tareas (*TodoWidgetSchema*):

```
var TodoWidgetSchema = new BaseSchema({
  todos: [{type : ObjectId, ref: 'Todo'}],
  title: { type: String }
});

var WeatherWidgetSchema = new BaseSchema({
  lat: {type: String},
  lon: {type: String},
  units: {type: String}
});

exports.TodoWidgetSchema = Widget.discriminator('TodoComponent', TodoWidgetSchema);
exports.WeatherWidgetSchema = Widget.discriminator('WeatherComponent', WeatherWidgetSchema);
```

7.6 Modelo de despliegue.

En la Ilustración 24, se muestra el modelo de despliegue de la aplicación, a lo largo de tres dispositivos: el cliente, el servidor web, y el servidor de bases de datos.

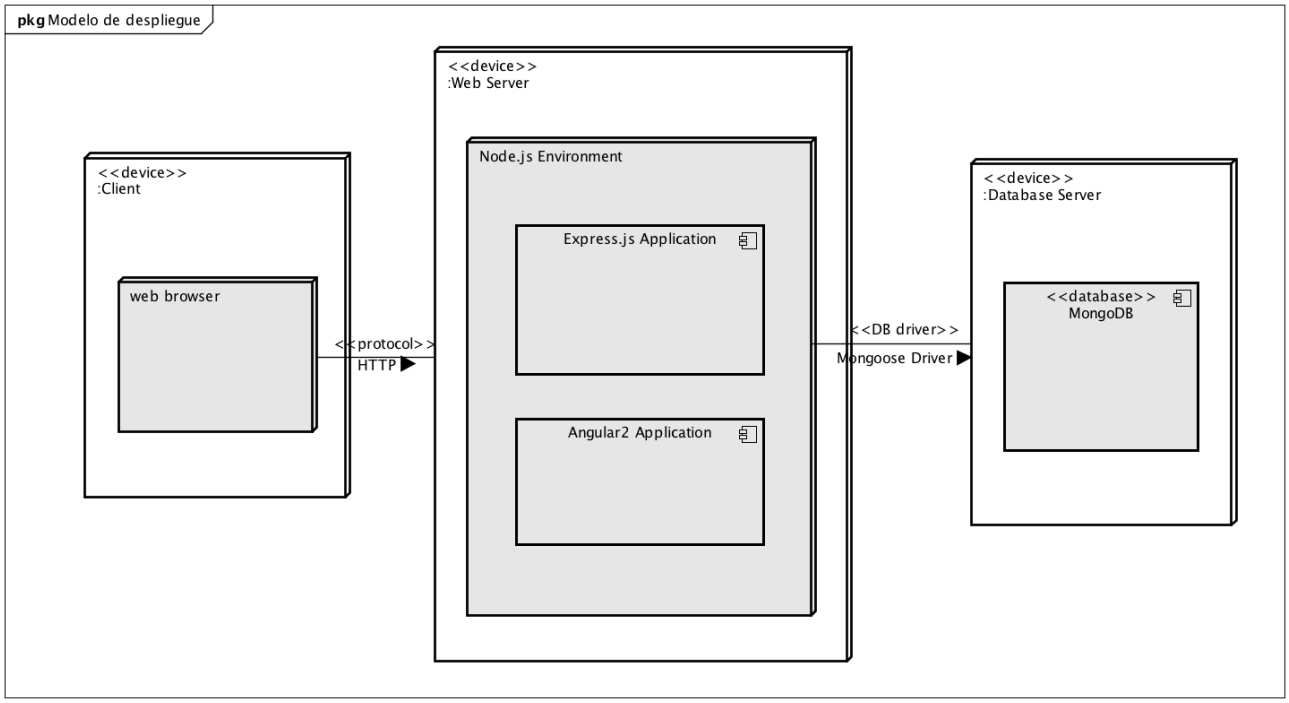

ILUSTRACIÓN 24 - DIAGRAMA DE DESPLIEGUE

Tanto el *frontend* como el *backend* son proveídos por un servidor web con soporte para Node.js, la plataforma elegida sobre la que desplegar ha sido Heroku.

El servidor web se conecta con el servidor de bases de datos de tipo MongoDB para lograr la persistencia y obtención de datos. El servidor de bases de datos está desplegado en la plataforma mLab, que provee bases de datos como servicios (*Database as a service*), y se integra con Heroku gracias a un *add-on*.

Toda la lógica de la aplicación que no requiere de escritura o lectura de la base de datos, se encuentra en el *frontend* y corre en el navegador del cliente. Cuando se requieren operaciones que necesitan de la intervención del servidor, éstas se realizan mediante llamadas a la API REST, empleando el protocolo HTTP y el formato JSON para la representación de los datos transferidos.

CAPÍTULO 8 - IMPLEMENTACIÓN.

En este capítulo se concretan algunos de los detalles de implementación de la aplicación.

8.1 API REST.

El *frontend* y el *backend* se comunican mediante una API de tipo REST (*REpresentational State Transfer*), dicha arquitectura se caracteriza por ser sin estado, ya que el servidor no mantiene información sobre el estado a lo largo de las operaciones [40].

La API facilita la comunicación entre el lado del cliente y el del servidor, para poder obtener y persistir información relativa al usuario, la configuración de widgets y las listas de tareas. Para implementar la API se ha utilizado el *framework* Express, que provee un conjunto robusto de funcionalidades para facilitar el desarrollo de APIs.

En Express, definir un *endpoint* es tan sencillo como especificar en el *router* la URL correspondiente, junto a la función *callback* encargada de procesar la petición:

```
//API Widget Endpoints
router.get('/widgets', auth, widgetsController.index);
router.post('/widget', auth, widgetsController.updateWidget);
router.delete('/widget/:id', auth, widgetsController.removeWidget);
```

En el ejemplo anterior se muestra la declaración de los *endpoint*s para las operaciones asociadas a gestión de widgets, como son la obtención de todos los widgets, y la actualización o borrado de uno en concreto. Cabe destacar que al hacer esta declaración también se puede especificar si es necesaria la autenticación del usuario, que en el caso de ejemplo mostrado lo es.

Las URIs de la API están formadas por la URL de la *webapp* seguidas de /api, los *endpoints* que conforman la API se pueden consultar en la Tabla 35 Endpoints de la API REST.

Método HTTP	URL	Descripción	Autenticación
GET	/todos/:idwidget	Obtiene las tareas asociadas a un determinado widget de listado de tareas	Si
POST	/todos	Dada una tarea, la actualiza con los nuevos datos	Si
PUT	/todos/:idwidget	Crea una nueva tarea asociada a un widget de listado de tareas	Si
DELETE	/todos/:id	Elimina la tarea que viene determinada por su id	Si
POST	/password	Cambia la contraseña por una nueva	Si
GET	/profile	Obtiene los datos relativos al perfil de un usuario	Si
POST	/profile	Actualiza un perfil de usuario con nuevos datos	Si
POST	/login	Identifica a un usuario para hacer login	No
PUT	/register	Crea un nuevo usuario	No
POST	/recoverpass	Permite activar el mecanismo de recuperación de contraseña (se genera un *token* y se manda un mail de recuperación)	No
POST	/resetpass	Verifica la validez del token y cambia la contraseña	No
GET	/weather	Obtiene la previsión del tiempo	No
GET	/coverproxy/:coverurl	Obtiene la disponibilidad de una portada dada su URL	No
GET	/quote	Obtiene una cita célebre	No
GET	/widgets	Obtiene todos los widgets del dashboard del usuario	Si
POST	/widget	Actualiza la configuración del widget	Si
DELETE	/widget/:id	Elimina un widget dado su ID	Si

TABLA 35 ENDPOINTS DE LA API REST

8.2 Autenticación y seguridad.

Para poder garantizar la seguridad de que un usuario sólo pueda realizar peticiones a la API referentes a sus datos, se ha usado autenticación empleando *JSON Web Tokens*.

Cuando un usuario se registra, se le solicita una contraseña de al menos 6 caracteres, entonces se genera un *salt*, es decir, una cadena aleatoria, y se guarda en base de datos el *hash* generado a partir de la contraseña y el *salt*. De esta manera se consigue almacenar las contraseñas de forma segura, puesto que no están en plano y además el *salt* aporta protección ante ataques por diccionario, especialmente en los que se empleen tablas *rainbow* precomputadas.

Después del login o del registro de un usuario se genera un *JSON Web Token* en el *backend* de la aplicación, este token se le pasa al *frontend*, y es almacenado en el almacenamiento local del navegador. Cada vez que se realice una petición a la API, se le pasará el *token* en el campo *Authorization*, y se comprobará en el lado del servidor que dicho *token* es válido y no ha caducado. En caso de invalidez, el usuario será redirigido a la página de *login* [43].

En el diagrama de actividad de la Ilustración 25 se muestra el flujo de autenticación, en el cual un usuario hace login, obtiene un *token*, y lo utiliza para realizar una petición de datos a la API.

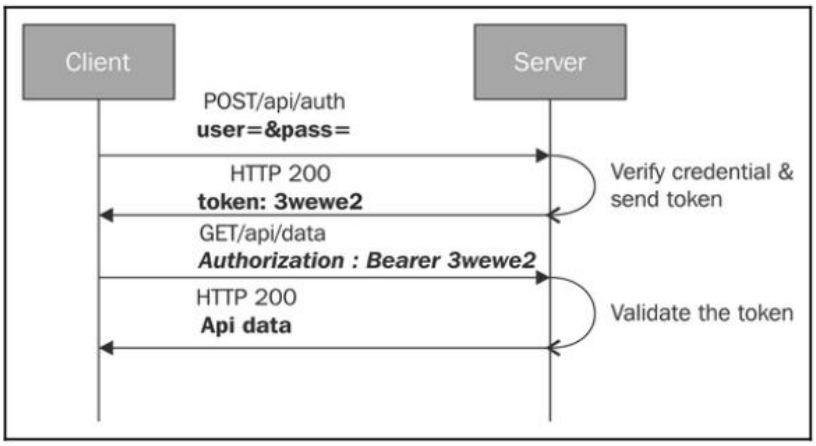

ILUSTRACIÓN 25 - AUTENTICACIÓN MEDIANTE JSON WEB TOKENS. TOMADO DE [44].

Para facilitar la gestión de autenticación se ha empleado un módulo para Node.js llamado Passport.js, este permite definir distintas estrategias de autenticación, las cuales se empaquetan en forma de módulos, lo que permite no incluir dependencias innecesarias.

8.4 Otros detalles de implementación.

8.4.1 Posición personalizable de los widgets

Uno de los objetivos es que el *dashboard* sea altamente personalizable, es por ello que resultaba conveniente que los widgets pudieran ubicarse como quisiese el usuario. Para lograr esto, se guarda la posición de los widgets de un usuario, mediante una variable de tipo entero incremental, cuando se mueve un widget se actualizan las nuevas posiciones de todos los demás.

Para permitir la recolocación de los widgets de una forma intuitiva, se ha optado por un mecanismo de drag and drop (arrastrar y soltar), empleando la librería Dragula [47] para facilitar esta tarea. Cuando un usuario entra en modo edición, los widgets se cargan dentro de elementos * * que gracias a la librería mencionada pueden ser arrastrados a posiciones distintas, y mediante la directiva [dragulaModel] se vincula un array representativo del estado.

8.4.2 Responsivo y tamaño variable de widgets.

Para facilitar la adaptabilidad a dispositivos móviles y el posicionamiento en la pantalla de los distintos widgets se ha empleado Bootstrap.

Bootstrap es un *framework* HTML, JavaScript y CSS para el desarrollo frontal de aplicaciones web, cuyo objetivo es facilitar la implementación de la parte visual de una web, con especial atención en las características responsivas de adaptabilidad a diferentes dispositivos [48].

Además de ofrecer elementos visuales como botones, barras y alertas, también introduce una forma sencilla de tratar la disposición espacial de los elementos, empleando contenedores y columnas.

Un contenedor puede ser fluido o de ancho fijo, en cualquier caso, su espacio horizontal se divide en 12 columnas; se determinan filas en las que introducir los elementos, y para cada uno de ellos se establece una columna y su ancho. Esto se logra mediante clases CSS, los prefijos col-xs, col-sm, col-md y col-ld permiten tratar con dispositivos muy pequeños, pequeños, medianos y grandes.

En el caso que nos ocupa, se ha establecido una relación de tamaños con valores de anchura de columna de Bootstrap, para permitir a los usuarios elegir el tamaño de un widget, dicha relación se ha realizado tomando múltiplos de dos y puede consultarse en la Tabla 36.

Tamaño	Equivalencia Bootstrap
Extra Small	col-md-2
Small	col-md-4
Medium	col-md-6
Large	col-md-8
Extra Large	col-md-10
Full Large	col-md-12

TABLA 36 EQUIVALENCIA DE TAMAÑOS CON BOOTSTRAP

Puede observarse que se ha establecido la división para dispositivos medianos y grandes, dado que en los pequeños resulta más conveniente que los widgets colapsen y se muestren en una única columna.

8.4.3 Proxy en la obtención de covers.

Inicialmente se planteó descargar las portadas de periódicos una vez al día, ayudándose de una tarea programada, y almacenarlas en el servidor. Pero debido a las peculiaridades de Heroku en cuanto a la obtención dinámica de *assets* en su versión gratuita, esto resultó imposible.

La solución propuesta pasaba por obtener las imágenes de las portadas directamente desde el proveedor, para lo que hubo que plantear respuesta a dos problemas: que se desconoce el momento en que las portadas del día están disponibles, y que hacer esta averiguación desde el *frontend* suponía un error en los navegadores modernos, ya que al contar con el mecanismo de Cross-origin resource sharing *(CORS)* no permiten realizar las peticiones necesarias.

La solución al primer problema fue sencilla, si la portada de hoy no está disponible aún, se solicita la de ayer. Para el segundo problema se planteó que era el *backend* el que debía encargarse de comprobar la disponibilidad de las portadas, de esta manera el *frontend* hace una llamada a la API para realizar esta averiguación, evitando así el error de *cross-origin* al actuar el ser**v**idor como *proxy* entre el cliente y el proveedor.

CAPÍTULO 9 - PRUEBAS.

9.1 Pruebas de validación.

Para la realización de pruebas, se han empleado las tecnologías expuestas en el capítulo 3 (ver secciones 3.7, 3.8 y 3.9). Debido a que en la metodología ágil empleada no hay una fase específica dedicada a pruebas, éstas se han ido realizando a medida que se implementaba funcionalidad.

Los ficheros en los que están implementadas las pruebas de validación se encuentran en el directorio e2e, que cuelga de la raíz del proyecto. Los archivos con extensión *feauture* contienen la especificación del test mediante Gherkin, los ficheros *page* sirven para desacoplar los elementos de la vista, de forma que, al ser empleados para conectar funcionalidad con elementos del DOM, si alguno se modifica sólo habrá que hacer el cambio en un sitio. Finalmente, los *step* contienen las acciones cuyo propósito es simular el paso que da un usuario durante la realización de una historia de usuario.

Aquí se recogen de forma representativa los resultados de las pruebas de validación de *login* y registro, la primera de ellas corresponde al momento en que se implementó la funcionalidad necesaria para que la prueba pasase como válida, la segunda corresponde al momento en que la prueba sirvió para detectar como algunos cambios en el código habían hecho que dejase de funcionar la funcionalidad ya existente.

9.1.1 Prueba satisfactoria de login.

A continuación, se documenta la prueba de validación del login, en un caso satisfactorio. Esta es la especificación tal y como aparece en el fichero *feature:*

```
Feature: Login


  Scenario Outline: The user is using the login form


    Given user is at the login page
    Given '<email>' is the user email in the login form
    Given '<password>' is the user password in the login form
    When submitting the login form
    Then the login form is validated '<valid>'


    Examples:
      | email           | password                | valid |
      | ""              | ""                      | false |
      | ""              | "thisisavalidpassword"  | false |
      | "test.es"       | "thisisavalidpassword"  | false |
      | "test@test.es"  | ""                      | false |
      | "test@test.es"  | "thisisavalidpassword"  | true  |
```

A modo de ejemplo, se detalla también la especificación del primer paso de la prueba, que estaría dentro del correspondiente fichero *step*. Se ha empleado la librería *cucumber-tsflow* para poder utilizar la sintaxis de anotaciones mostrada.

```
@given(/^user is at the login page$/)
public givenUserIsAtLoginPage (callback): void {
  this.loginPageObject.get();
  expect(this.loginPageObject.getTitle()).to.eventually.equal('Login');
  callback();
}
```

Para las aserciones se ha empleado la librería *chai-as-expected*.

Feature:	Login (US-006)
Escenario:	Hacer login satisfactoriamente
Pasos	
✔ Given user is at the login page	
✔ Given '"test@test.es"' is the user email in the login form	
✔ Given '"thisisavalidpassword"' is the user password in the login form	
✔ When submitting the login form	
✔ Then the login form is validated 'true'	

TABLA 37 PRUEBA DE LOGIN SATISFACTORIO

Feature:	Login (US-006)
Escenario:	Hacer login con email inexistente
Pasos	
✔ Given user is at the login page	
✔ Given '"nonregistered@email.es"' is the user email in the login form	
✔ Given '"thisisavalidpassword"' is the user password in the login form	
✔ When submitting the login form	
✔ Then the login form is validated 'false'	

TABLA 38 PRUEBA DE LOGIN CON EMAIL NO REGISTRADO

Feature:	Login (US-006)
Escenario:	Hacer login con formulario vacío
Pasos	
✔ Given user is at the login page	
✔ Given '""' is the user email in the login form	
✔ Given '""' is the user password in the login form	
✔ When submitting the login form	
✔ Then the login form is validated 'false'	

TABLA 39 PRUEBA DE LOGIN CON CAMPOS VACÍOS

Feature:	Login (US-006)
Escenario:	Hacer login con email vacío
Pasos	
✔ Given user is at the login page ✔ Given "" is the user email in the login form ✔ Given "'thisisavalidpassword"' is the user password in the login form ✔ When submitting the login form ✔ Then the login form is validated 'false'	

TABLA 40 PRUEBA DE LOGIN CON EMAIL VACÍO

Feature:	Login (US-006)
Escenario:	Hacer login con contraseña vacía
Pasos	
✔ Given user is at the login page ✔ Given "'test@test.es"' is the user email in the login form ✔ Given "" is the user password in the login form ✔ When submitting the login form ✔ Then the login form is validated 'false'	

TABLA 41 PRUEBA DE LOGIN CON CONTRASEÑA VACÍA

Feature:	Login (US-006)
Escenario:	Hacer login con email inválido
Pasos	
✔ Given user is at the login page ✔ Given "'test.es"' is the user email in the login form ✔ Given "'thisisavalidpassword"' is the user password in the login form ✔ When submitting the login form ✔ Then the login form is validated 'false'	

TABLA 42 PRUEBA DE LOGIN CON EMAIL INVÁLIDO

9.1.2 Prueba no satisfactoria de registro.

```
  Feature: Register

Scenario Outline: The user is using the register form

  Given user is at the register page
  Given '<email>' is the user email in the register form
  Given '<password>' is the user password in the register form
  Given '<name>' is the user name in the register form
  When submitting the register form
  Then the register form is validated '<valid>'

  Examples:
    | email           | password                | name       | valid |
    | ""              | ""                      | ""         | false |
    | ""              | "thisisavalidpassword"  | "test"     | false |
    | "test.es"       | "thisisavalidpassword"  | "test"     | false |
    | "test@test.es"  | ""                      | "test"     | false |
    | "test@test.es"  | "thisisavalidpassword"  | "test"     | true  |
    | "test@test.es"  | "thisisavalidpassword"  | ""         | false |
```

Feature:	Registro (US-001)
Escenario:	Registro con campos vacíos
Pasos	
✔ Given user is at the register page	
✔ Given "" is the user email in the register form	
✔ Given "" is the user password in the register form	
✔ Given "" is the user name in the register form	
✔ When submitting the register form	
✔ Then the register form is validated 'false'	

TABLA 43 PRUEBA DE REGISTRO CON CAMPOS VACÍOS

Feature:	Registro (US-001)
Escenario:	Registro con email vacío
Pasos	
✔ Given user is at the register page	
✔ Given "" is the user email in the register form	
✔ Given "'thisisavalidpassword'" is the user password in the register form	
✔ Given "'test'" is the user name in the register form	
✔ When submitting the register form	
✔ Then the register form is validated 'false'	

TABLA 44 PRUEBA DE REGISTRO CON EMAIL VACÍO

Feature:	Registro (US-001)
Escenario:	Registro con email inválido
Pasos	
✔ Given user is at the register page	
✔ Given '"test.es"' is the user email in the register form	
✔ Given '"thisisavalidpassword"' is the user password in the register form	
✔ Given '"test"' is the user name in the register form	
✔ When submitting the register form	
✔ Then the register form is validated 'false'	

TABLA 45 PRUEBA DE REGISTRO CON EMAIL INVÁLIDO

Feature:	Registro (US-001)
Escenario:	Registro con contraseña vacía
Pasos	
✔ Given user is at the register page	
✔ Given '"test@test.es"' is the user email in the register form	
✔ Given '""' is the user password in the register form	
✔ Given '"test"' is the user name in the register form	
✔ When submitting the register form	
✔ Then the register form is validated 'false'	

TABLA 46 PRUEBA DE REGISTRO CON CONTRASEÑA VACÍA

Feature:	Registro (US-001)
Escenario:	Registro con nombre vacío
Pasos	
✔ Given user is at the register page	
✔ Given '"test@test.es"' is the user email in the register form	
✔ Given '"thisisavalidpassword"' is the user password in the register form	
✔ Given '""' is the user name in the register form	
✔ When submitting the register form	
✔ Then the register form is validated 'false'	

TABLA 47 PRUEBA DE REGISTRO CON NOMBRE VACÍO

Feature:	Registro (US-001)
Escenario:	Registro con email ya existente
Pasos	
✔ Given user is at the register page	
✔ Given 'emailthatexists@test.es"' is the user email in the register form	
✔ Given "'thisisavalidpassword"' is the user password in the register form	
✔ Given "'test"' is the user name in the register form	
✔ When submitting the register form	
✔ Then the register form is validated 'false'	

TABLA 48 PRUEBA DE REGISTRO CON EMAIL YA EXISTENTE

Feature:	Registro (US-001)
Escenario:	Registro satisfactorio
Pasos	
✔ Given user is at the register page	
✔ Given "'test@test.es"' is the user email in the register form	
✔ Given "'thisisavalidpassword"' is the user password in the register form	
✔ Given "'test"' is the user name in the register form	
✔ When submitting the register form	
✗ Then the register form is validated 'true'	

TABLA 49 PRUEBA DE REGISTRO SATISFACTORIO

La ejecución mostrada de esta prueba sirvió para detectar que la funcionalidad del registro dejó de funcionar, las comprobaciones relativas a validez de datos del formulario se realizaban correctamente, el fallo se producía precisamente cuando los datos de registro eran válidos.

Una vez hecha la traza, se determinó que el causante era la rutina encargada de enviar un email a los nuevos usuarios cuando estos se registran, ya que la configuración de email estaba mal introducida, por lo que el fallo se producía únicamente cuando el usuario se registraba correctamente y se trataba de mandar el correo.

Una vez corregida la incidencia, la prueba de validación del registro para el escenario satisfactorio volvió a pasar correctamente.

Feature:	Registro (US-001)
Escenario:	Registro satisfactorio
Versión:	Correctora
Pasos	
✔ Given user is at the register page	
✔ Given "'test@test.es"' is the user email in the register form	
✔ Given "'thisisavalidpassword"' is the user password in the register form	
✔ Given "'test"' is the user name in the register form	
✔ When submitting the register form	
✔ Then the register form is validated 'true'	

TABLA 50 PRUEBA DE REGISTRO SATISFACTORIO TRAS CORREGIR LA INCIDENCIA

9.2 Pruebas unitarias.

Las pruebas unitarias resultan especialmente interesantes de realizar sobre las clases que implementan servicios, para comprobar que las operaciones sobre datos funcionan correctamente, obteniendo los resultados esperados.

Angular emplea un entorno de pruebas denominado Karma, se trata de una herramienta que despliega un servidor que ejecuta el código a testear contra uno o varios navegadores conectados, también tiene la capacidad de observar ficheros para realizar automáticamente test cuando se detecten cambios.

Como *framework* se emplea Jasmine, el cual se integra a la perfección con Karma, ofreciendo una sintaxis sencilla e intuitiva.

A modo de ejemplo, se documentan aquí las pruebas unitarias realizadas sobre el servicio asociado al *widget* de lista de tareas, esto es, la clase *TodoDataService*. Por convención, los ficheros destinados a realizar pruebas unitarias en Angular llevan el infijo *spec* en su nombre, por tanto las pruebas se especifican en el fichero *todo-data-service.spec.ts*. Esta es la prueba que permite comprobar que una tarea se está añadiendo correctamente a un *widge*t de listado de tareas:

```
 describe('#add(todo, widgetid)', () => {

    it('should add a todo to the widget', inject([TodoDataService],
     (service: TodoDataService) => {
      let todo = new Todo({title: 'Test', complete: false});
      service.addTodo(todo,'1');
      expect(service.getTodos('1')).toEqual([todo]);
    }));

  });
```

Para ejecutar manualmente las pruebas, ha de usarse el comando *ng test*:

```
$ ng test

INFO [karma]: Karma v0.12.31 server started at http://localhost:9876/

INFO [launcher]: Starting browser Chrome

INFO [Chrome 42.0.2311 (Mac OS X 10.10.3)]: Connected on socket 2absOkNfa1asasaX0fCJ with id 71338229

Chrome 42.0.2311 (Mac OS X 10.10.3): Executed 10 of 10 (10 SUCCESS) (14.075 secs)
```

Las pruebas unitarias siempre resultan muy útiles para asegurarse del funcionamiento correcto de la aplicación, pudiendo probar a un nivel de detalle más bajo que el que ofrecen las pruebas de validación.

PARTE TERCERA - CONCLUSIONES.

CAPÍTULO 10 - CONCLUSIONES Y FUTURAS MEJORAS.

10.1 Conclusiones.

Tras la realización del proyecto se puede asegurar que se han conseguido los objetivos marcados, se ha desarrollado una aplicación SaaS consistente en un *dashboard*, que cumple los requisitos especificados en forma de historias de usuario, y el proyecto se ha planificado y completado empleando metodologías ágiles.

En cuanto a la metodología empleada, he podido afianzar mis conocimientos sobre SCRUM, y aprender cómo se planifica un proyecto desde el inicio hasta su compleción siendo el responsable en todo el proceso.

Partía de un desconocimiento total de la pila MEAN, y he logrado adquirir los conocimientos necesarios para completar una aplicación compleja, dinámica y personalizable.

La elección de Angular 2 como *framework* para el *frontend* ha tenido como ventajas poder usar tecnologías y técnicas de reciente y futura adopción, como los componentes web o el uso de TypeScript. También ha tenido como desventajas que se trata de un *framework* muy reciente y en constante evolución, por lo que la documentación a menudo era escasa e incompleta, y que se producían cambios destructivos entre versiones.

Por otro lado Node.js resultó ser una tecnología mucho más madura, de la que puede encontrarse una base sólida de documentación y ejemplos, por lo que fue mucho más fácil de aprender y resultó menos problemático de utilizar. También he podido comprobar como Express facilita enormemente la creación de APIs.

En cuanto a MongoDB, ha resultado muy interesante trabajar con una base de datos no relacional, pudiendo comparar la forma en que se tratan los datos respecto al paradigma más tradicional de las bases de datos relacionales, que me resultaba mucho más familiar.

En general desarrollar este proyecto me ha servido para adquirir más conocimientos sobre gestión de proyectos y desarrollo web, y afianzar los ya existentes; dado que he podido aprender nuevas tecnologías como las de la ya mencionada pila MEAN, y practicar otras como HTML, CSS, JavaScript o UML.

10.2 Mejoras futuras.

Aunque el proyecto está terminado, pueden realizarse algunas mejoras y expandir funcionalidades, estas son algunas de ellas:

- **Soporte para múltiples idiomas:** Actualmente la aplicación está implementada en inglés, sería conveniente adaptarla para que soporte múltiples idiomas, con el fin de llegar a más usuarios finales. Esta adaptación presenta complejidades como tener que lidiar con múltiples servicios proveedores, ya que, por ejemplo, actualmente el de citas célebres sólo aporta contenido en inglés.

- **Geoubicación para el widget del tiempo:** Resultaría muy cómodo para el usuario ver en todo momento la previsión del tiempo del lugar en el que se encuentra, de forma que si se desplaza no tenga que cambiar la configuración, para ello el widget del tiempo podría usar la integración de la geoubicación que aportan los navegadores modernos para localizar el sitio para el que mostrar el tiempo.

- **Selección de personalizable de diarios en el widget de portadas:** Sería conveniente que un usuario pudiese elegir cuáles son los diarios o periódicos para los que le interesa ver portadas. Actualmente esta selección viene dada por defecto y no es personalizable, debido a que no se ha podido encontrar un proveedor de portadas suficientemente abierto o que provea una API.

- **Limitar la forma en que pueden colocarse los widgets:** La implementación actual de la colocación de los widgets permite dejar el *dashboard* en estados muy poco atractivos visualmente. Limitar las zonas donde puede colocarse un determinado widget no permitiría al usuario hacer combinaciones poco usables.

- **Añadir autenticación con servicios como Facebook o Twitter:** Actualmente es muy cómodo poder emplear mecanismos de autenticación 0-Auth que nos permitan usar aplicaciones sin tener que registrarnos en una nueva plataforma, por ello sería conveniente poder usar cuentas de servicios como Facebook o Twitter para *loguearse* en la aplicación.

- **Implementar más widgets:** El *dashboard* actual está implementado para que pueda expandirse y sirve como base para implementar tantos widgets nuevos como se desee, algunos de ellos podrían ser relativos al estado del tráfico, la integración con redes sociales o la visualización de datos financieros.

- **Refrescar el contenido de forma automatizada:** Podría establecerse un intervalo de tiempo para recargar de forma automática la información de widgets como el de noticias, el tiempo o portadas para que el usuario visualice nuevos datos sin tener que refrescar la página.

BIBLIOGRAFÍA

[1] M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, A. Hunt, J. Kern y R. Jeffries, «Principles of the Agile Manifesto,» 2001 febrero 17. [En línea]. Available: http://agilemanifesto.org/iso/es/manifesto.html. [Último acceso: 2017 junio 14].

[2] P. Venkata Gayatri, «Agile User Stories,» 2013. [En línea]. Available: https://www.scrumalliance.org/community/articles/2013/september/agile-user-stories . [Último acceso: 18 junio 2017].

[3] D. Steinberg y D. W. Palmer, Extreme Software Engineering, Pearson Education, 2004.

[4] Colaborativo, «SCRUM (Software Development),» 2017. [En línea]. Available: https://en.wikipedia.org/wiki/Scrum_(software_development). [Último acceso: 18 junio 2017].

[5] Softeng, 2014. [En línea]. Available: https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum.html. [Último acceso: 18 junio 2017].

[6] D. North, «How to sell BDD to business,» 2009. [En línea]. Available: https://skillsmatter.com/skillscasts/923-how-to-sell-bdd-to-the-business. [Último acceso: 18 junio 2017].

[7] D. North, «Introducing BDD,» 2006. [En línea]. Available: https://dannorth.net/introducing-bdd/. [Último acceso: 2017 junio 16].

[8] M. Fowler, «Given, When, Then,» 2013. [En línea]. Available: https://martinfowler.com/bliki/GivenWhenThen.html. [Último acceso: 18 junio 2017].

[9] L. Karam, «The Benefits of Behavior-Driven Development,» 2017. [En línea]. Available: https://cucumber.io/docs/reference. [Último acceso: 18 junio 2017].

[10] Mean.io, «Mean.io,» 2016. [En línea]. Available: http://mean.io/. [Último acceso: 18 junio 2017].

[11] MongoDB, «What is MongoDB,» 2017. [En línea]. Available: https://www.mongodb.com/what-is-mongodb. [Último acceso: 18 junio 2017].

[12] MongoDB , «Introduction to MongoDB,» 2016. [En línea]. Available: https://docs.mongodb.com/manual/introduction/. [Último acceso: 2017 junio 18].

[13] G. Voyeur-Perrault, «MongoDB: What is the idea behind an ORM/ODM for a NoSQL database?,» 2012. [En línea]. Available: https://www.quora.com/MongoDB-What-is-the-idea-behind-an-ORM-ODM-for-a-NoSQL-database. [Último acceso: 18 junio 2017].

[14] Mongoose, «Mongoose Docs: Queries,» 2016. [En línea]. Available: http://mongoosejs.com/docs/queries.html. [Último acceso: 18 junio 2017].

[15] A. Mardan, «Foreword,» de *Pro Express.js*, Apress, p. xix.

[16] Express, «Express.js Routing,» 2017. [En línea]. Available: http://expressjs.com/en/guide/routing.html. [Último acceso: 18 junio 2017].

[17] Express, «Using middleware,» 2017. [En línea]. Available: http://expressjs.com/en/guide/using-middleware.html. [Último acceso: 18 junio 2017].

[18] Express, «Using template engines,» 2017. [En línea]. Available: http://expressjs.com/en/guide/using-template-engines.html. [Último acceso: 18 junio 2017].

[19] Colaborativo, «AngularJS,» 2017. [En línea]. Available: https://en.wikipedia.org/wiki/AngularJS. [Último acceso: 17 junio 2017].

[20] I. Minar, «It's going to be Angular 4.0, or just Angular,» 2016. [En línea]. Available: http://angularjs.blogspot.com.es/2016/12/ok-let-me-explain-its-going-to-be.html#Why_not_version_3_then_62. [Último acceso: 17 junio 2017].

[21] R. Shresth, «Angular 2: A Component-Based MVC Framework,» 2016. [En línea]. Available: https://dzone.com/articles/angular-2-a-component-based-mvc-framework. [Último acceso: 17 junio 2017].

[22] D. Scott y M. Mundi, «6 Reasons for Employing Component-based UI Development,» 2016. [En línea]. Available: https://www.tandemseven.com/technology/6-reasons-component-based-ui-development/. [Último acceso: 18 junio 2017].

[23] Google Inc., «Angular Architecture Overview,» 2016. [En línea]. Available: https://angular.io/guide/architecture. [Último acceso: 17 junio 2017].

[24] Google Inc., «Angular Docs: Displaying Data,» 2017. [En línea]. Available: https://angular.io/guide/displaying-data. [Último acceso: 18 junio 2017].

[25] P. Precht, «Two-way Data Binding in Angular,» 2016. [En línea]. Available: https://blog.thoughtram.io/angular/2016/10/13/two-way-data-binding-in-angular-2.html. [Último acceso: 2017 junio 19].

[26] Google Inc., «Angular Docs: Dependency Injection,» 2017. [En línea]. Available: https://angular.io/guide/dependency-injection. [Último acceso: 18 junio 2017].

[27] Colaborativo, «How MVC pattern can be explained in Angular 2?,» 2016. [En línea]. Available: https://stackoverflow.com/questions/38844771/how-mvc-pattern-can-be-explained-in-angular-2. [Último acceso: 17 junio 2017].

[28] P. Precht, «Angular change detection explained,» 2016. [En línea]. Available: https://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html. [Último acceso: 14 junio 2017].

[29] Google Inc., «Angular Docs: Lifecycle Hooks,» 2016. [En línea]. Available: https://angular.io/guide/lifecycle-hooks. [Último acceso: 2017 junio 15].

[30] Node, «Node.js Home,» 2015. [En línea]. Available: https://nodejs.org/es/. [Último acceso: 15 junio 2017].

[31] Colaborativo, «What is Node.js?,» 2009. [En línea]. Available: https://stackoverflow.com/questions/1884724/what-is-node-js. [Último acceso: 13 junio 2017].

[32] Protractor, «Protractor Website,» 2017. [En línea]. Available: http://www.protractortest.org/. [Último acceso: 14 junio 2017].

[33] Cucumber, «Cucumber Reference,» 2017. [En línea]. Available: https://cucumber.io/docs/reference. [Último acceso: 18 junio 2017].

[34] P. Tracker, «Pivotal Tracker Reference Quick Start,» 2015. [En línea]. Available: https://www.pivotaltracker.com/help/articles/quick_start/. [Último acceso: 18 junio 2017].

[35] Toggl, «Toggl Website,» 2017. [En línea]. Available: https://toggl.com/. [Último acceso: 18 junio 2017].

[36] T. -. G. d. Alumno, «Web de la escuela de ingeniería informática,» 2017. [En línea]. Available: https://www.inf.uva.es/wp-content/uploads/2013/01/00-GuiaAlumnoTFG_2017.pdf. [Último acceso: 17 junio 2017].

[37] N. Figuerola, «Riesgos: Plan de Mitigación vs Plan de Contingencia vs Fallback Plan,» 2015. [En línea]. Available: https://articulospm.files.wordpress.com/2015/06/riesgos-plan-mitigacion-vs-plan-contingencia-vs-fallback-plan.pdf. [Último acceso: 10 junio 2017].

[38] Ministerio de Trabajo e Inmigración, «BOE - XVI Convenio Colectivo Estatal de empresas de consultoría y estudios de mercados y la opinión pública,» 2009. [En línea]. Available: https://www.boe.es/boe/dias/2009/04/04/pdfs/BOE-A-2009-5688.pdf. [Último acceso: 2017 6 17].

[39] Agencia Tributaria , «Tabla de coeficientes de amortización lineal.,» 2016. [En línea]. Available: http://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml. [Último acceso: 14 junio 2017].

[40] G. Ilya, «Latency: The new web performance bottleneck,» 2012. [En línea]. Available: https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/. [Último acceso: 2017 junio 16].

[41] J. Tejero, «SPA, un paradigma de arquitectura de aplicaciones Web en auge,» 2013. [En línea]. Available: https://cink.es/blog/2013/10/07/spa-un-paradigma-de-arquitectura-de-aplicaciones-web-en-auge/. [Último acceso: 18 junio 2017].

[42] A. Kumar, «MEAN Stack Apps explained for Java Developers,» 2016. [En línea]. Available: https://dzone.com/articles/mean-stack-apps-explained-for-java-developers. [Último acceso: 16 junio 2017].

[43] G. Stafford, «Calling Third-Party HTTP-based RESTful APIs from the MEAN Stack,» 2015. [En línea]. Available: https://programmaticponderings.com/2015/01/06/calling-third-party-web-apis-from-the-mean-stack/. [Último acceso: 2017 junio 18].

[44] M. Asier, «Conceptos sobre APIs REST,» 2012. [En línea]. Available: http://asiermarques.com/2013/conceptos-sobre-apis-rest/ . [Último acceso: 17 junio 2017].

[45] S. Holmes, «User Authentication with the MEAN Stack,» 2016. [En línea]. Available: https://www.sitepoint.com/user-authentication-mean-stack/. [Último acceso: 15 junio 2017].

[46] A. Chandermani y K. Henessey, Angular 2 By Example, Packt, 2016.

[47] Colaborativo, «Repositorio de Dragula,» 2017. [En línea]. Available: https://github.com/bevacqua/dragula. [Último acceso: 18 junio 2017].

[48] Bootstrap, «Sitio web de Bootstrap,» 2017. [En línea]. Available: http://getbootstrap.com/. [Último acceso: 18 junio 2017].

[49] Google Inc., «Angular Docs: Testing,» 2016. [En línea]. Available: https://angular.io/guide/testing. [Último acceso: 2017 junio 15].

[50] J. Hanson, «PassportJS Docs,» 2016. [En línea]. Available: http://passportjs.org/docs. [Último acceso: 2017 junio 15].

[51] M. Mooney, «Getting started with Protractor and Cucumber,» 2016. [En línea]. Available: https://semaphoreci.com/community/tutorials/getting-started-with-protractor-and-cucumber . [Último acceso: 17 junio 2017].

[52] D. Patterson y A. Fox, Engineering Software As a Service: An Agile Approach Using Cloud Computing, Primera ed., Strawberry Canyon, 2013.

[53] A. Goicochea, «Análisis de los riesgos de un proyecto,» 2012. [En línea]. Available: https://anibalgoicochea.com/2012/11/23/analisis-de-los-riesgos-de-un-proyecto/. [Último acceso: 18 junio 2017].

ANEXOS

Suponiendo que se parte del código fuente, lo primero es instalar las dependencias, que se encuentran definidas en el fichero *package.json*, mediante el comando:

```
$ npm install
```

Una vez hecho esto, se ha de construir la aplicación Angular 2, usando AngularCLI, mediante el siguiente comando:

```
$ ng build
```

Para correr el servidor, hay que ejecutar el fichero server.js con Node.js:

```
$ node server.js
```

Una vez que el servidor está en ejecución, la aplicación es accesible mediante el navegador, a través del puerto 3000.

Despliegue automático en Heroku:

1. Hacer login en el servicio

```
$ heroku login
```

2. Clonar el repositorio

```
$ heroku git:clone dash-app
```

3. Desplegar la aplicación

```
$ git push heroku master
```

Esto nos permite subir los cambios que se ha anotado en el repositorio, realizando el despliegue de forma automática, gracias a las tareas definidas en el fichero *package.json*. Una vez realizados los dos primeros pasos, ya sólo será necesario realizar el tercero cada vez que se quiera desplegar, o permitir el acceso al repositorio desde el panel de Heroku, para que se desplieguen los cambios cada vez que se haga un *commit* sin necesidad de ejecutar el comando.

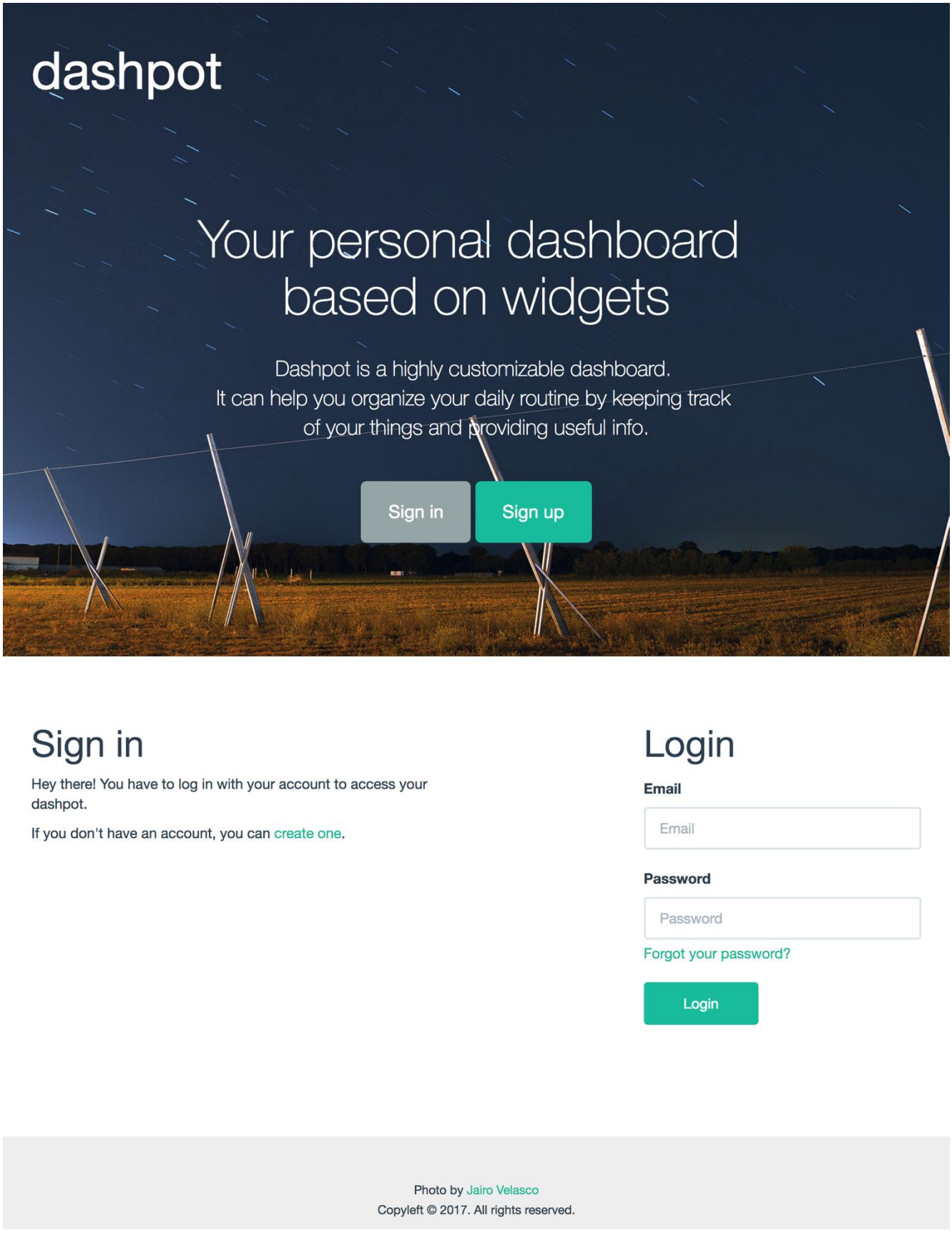

ILUSTRACIÓN 26 – VISTA DE LA PÁGINA DE INICIO

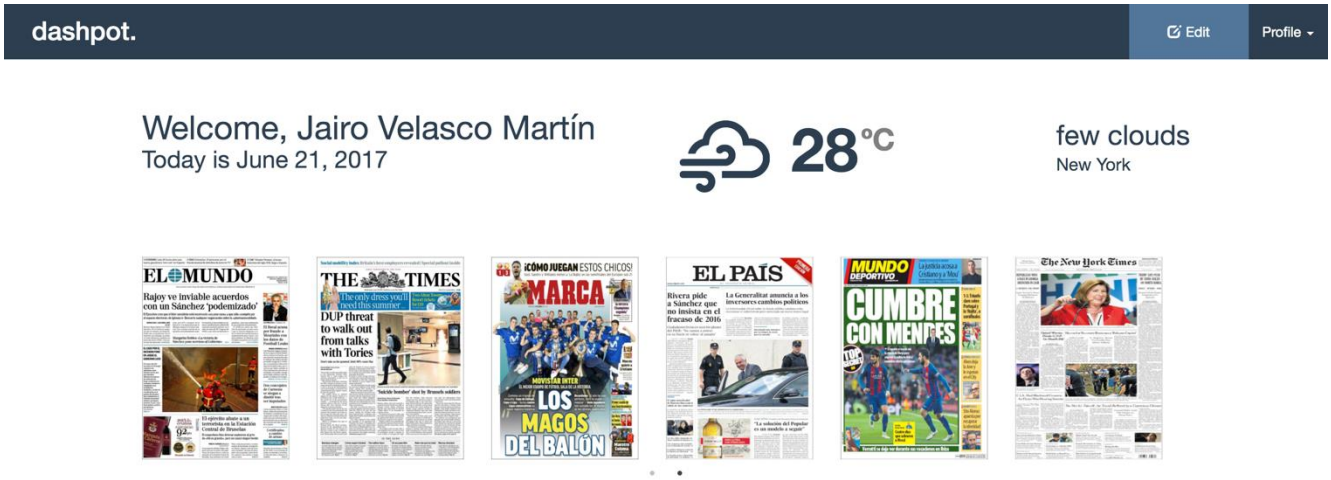

ILUSTRACIÓN 27 - VISTA DEL DASHBOARD CON LA CONFIGURACIÓN POR DEFECTO

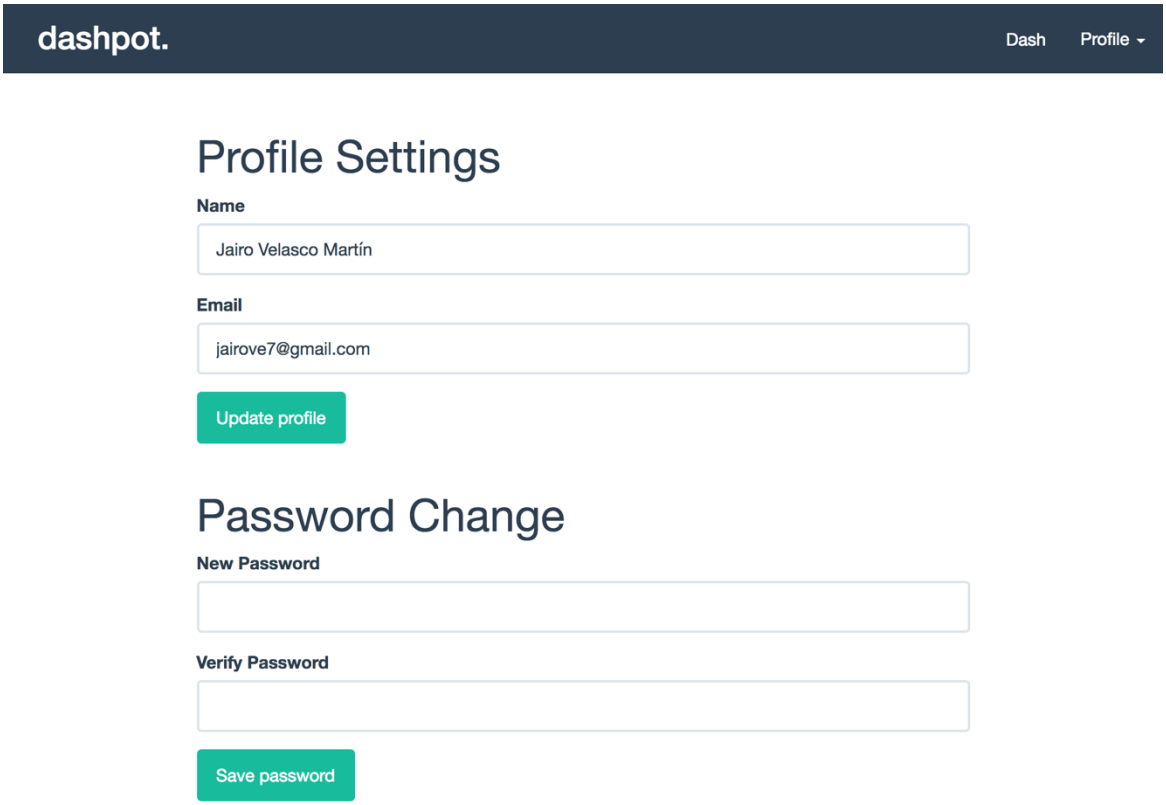

ILUSTRACIÓN 28 - VISTA DE LA CONFIGURACIÓN DE PERFIL

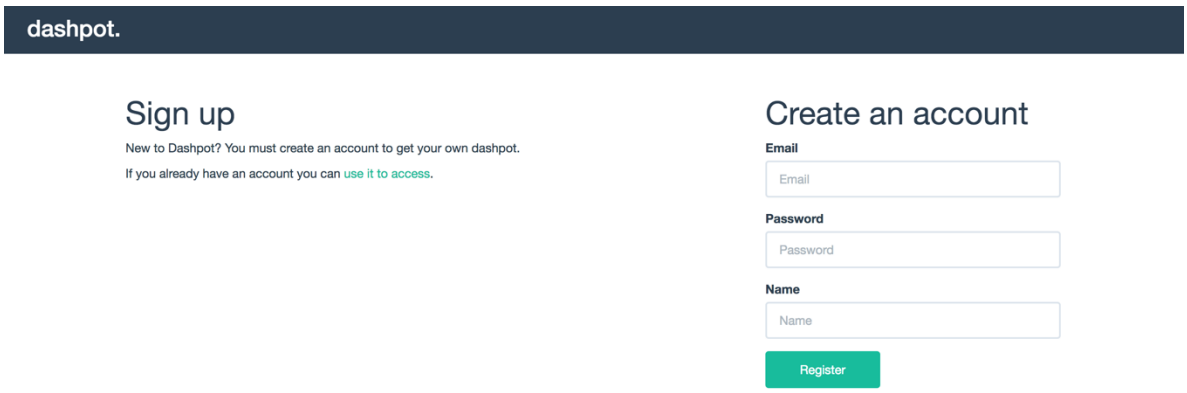

ILUSTRACIÓN 29 - VISTA DE LA PÁGINA DE REGISTRO

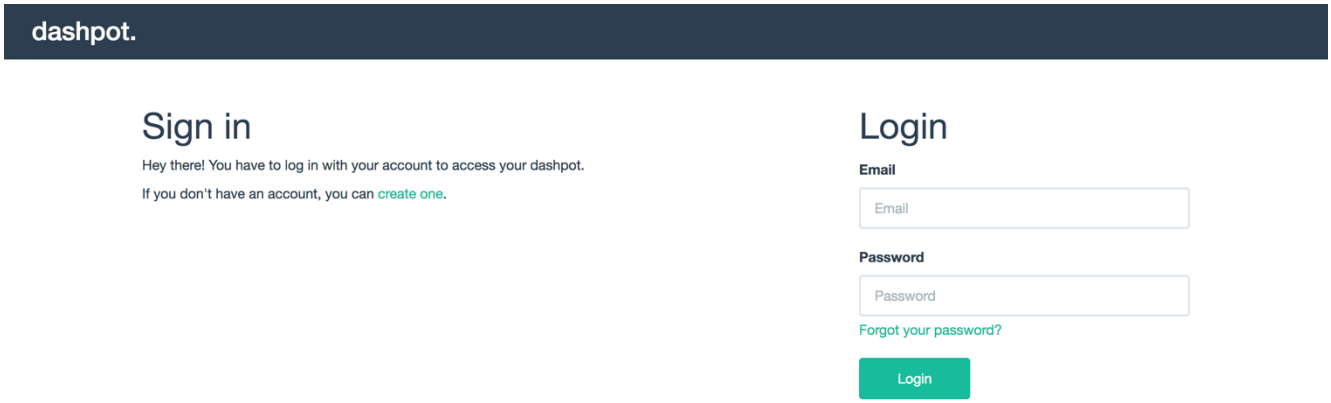

ILUSTRACIÓN 30 - VISTA DE LA PÁGINA DE ACCESO

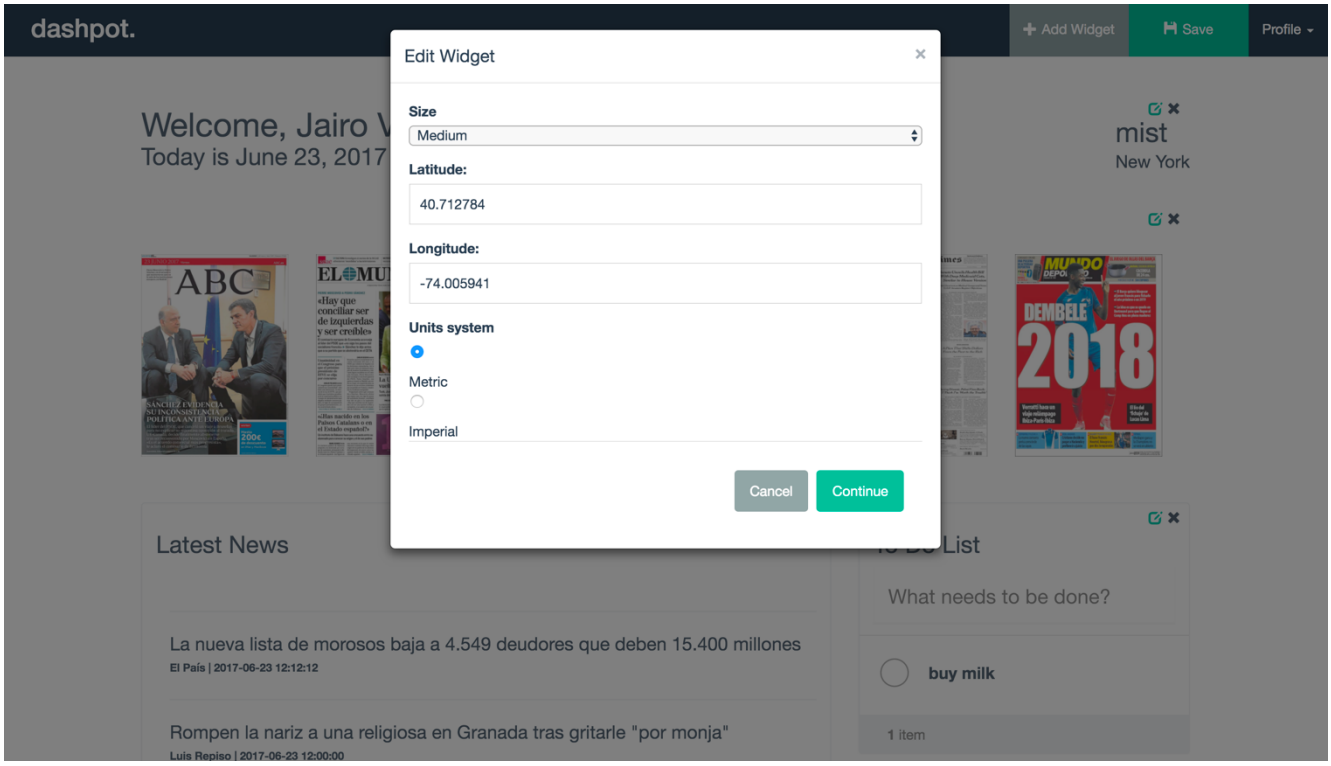

ILUSTRACIÓN 31 - VISTA DE LA EDICIÓN DE UN WIDGET

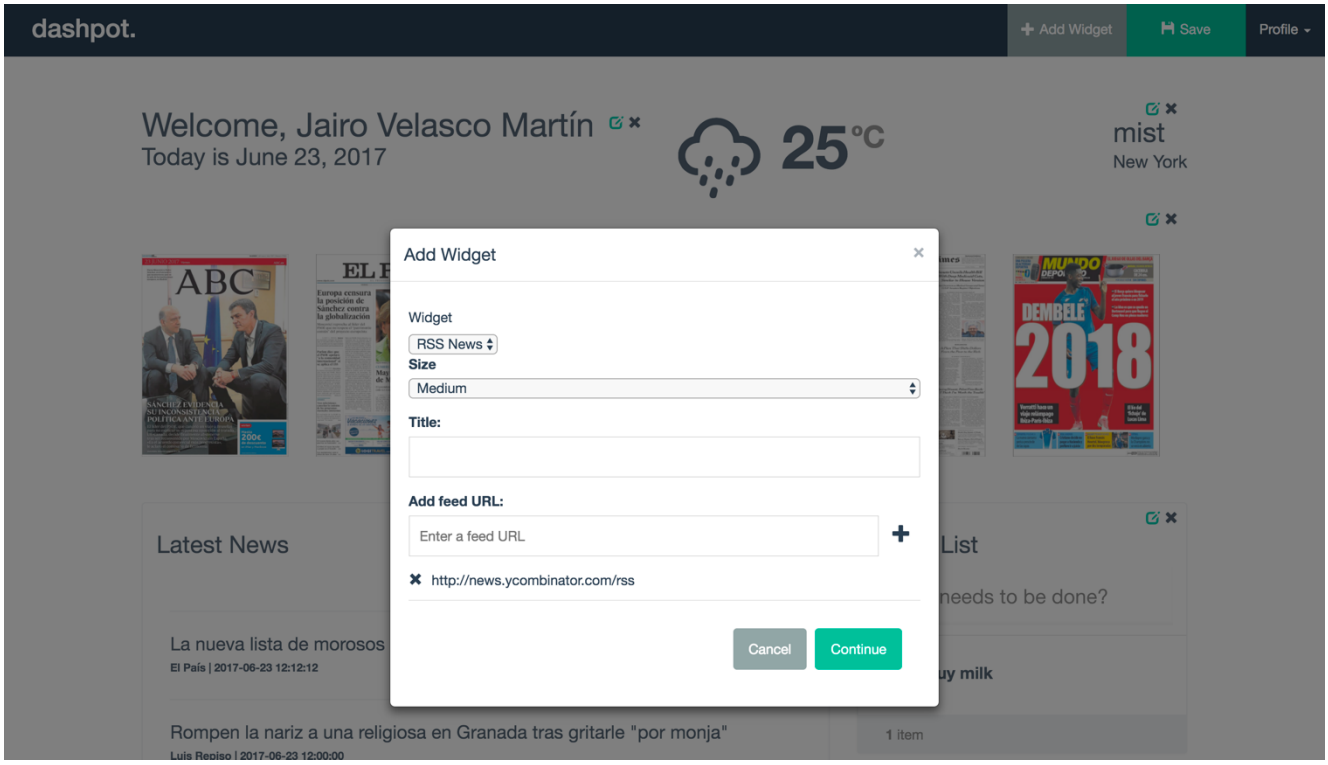

ILUSTRACIÓN 32 - VISTA DE AÑADIR WIDGET