Web API de Gestión de Inventarios

Solución web de gestión empresarial desarrollada en AngularJS consumiendo un servicio REST.

Sergio Arranz Sobrino
Desarrollo de Aplicaciones Multiplataforma
U-Tad
Madrid, España
sergioas1996@gmail.com

***Abstracción*—En este proyecto se presenta la propuesta y desarrollo de una aplicación web como solución para negocios y proyectos orientados a la gestión de inventarios. El principal objetivo es proveer al cliente de una herramienta con una serie de utilidades para manejar sus datos, gestionarlos y administrarlos de forma cómoda, dinámica y sencilla haciendo uso de las últimas tecnologías y componentes web.**

***Índice de términos*—AngularJS, JSON, Web API, gestión empresarial, REST.**

I. Introducción

En la industria actual, las web APIs informáticas de gestión empresarial son muy importantes puesto que son el pilar fundamental sobre el que se apoya una empresa para desarrollar correctamente las actividades de la misma. Gracias a ellas, destacarán en la toma de decisiones y objetivos al contar con un tratamiento de información rápido, fluido y mucho más cómodo de manejar.

Son una de las grandes revoluciones recientes de la tecnología informática para la gestión de datos en la nube. En este ámbito, las APIs (Application Programming Interface) son las piezas clave de código que permiten compatibilidad y coexistencia para que todo funcione. En muchos casos, es el facilitador para que los usuarios remotos puedan consumir los servicios de un determinado proveedor.

Existen variedades de soluciones web informáticas en el ámbito empresarial, desde las más orientadas a soporte como pueda ser un Helpdesk, CAU o Bug Tracking; hasta aquellas más orientadas a gestión de páginas web y foros como los CMS (sistemas de Gestión de Contenidos), muy usados para crear y administrar contenidos por parte de administradores y editores.

La aplicación planteada pretende gestionar inventarios a través de tablas con datos, de modo que ha sido concebida para clientes más específicos como podría ser una empresa de logística. Más allá de su funcionalidad, gran parte del desarrollo ha sido centrado en transmitir una ilustración clara y concisa al receptor puesto que en muchos casos formará parte del personal del negocio en sí, como podría ser un administrativo o un funcionario que registre y actualize datos constantemente.

Este último aspecto (Experiencia de Usuario) ha sido uno de los principales problemas del caso inicial para el que se comenzó a desarrollar la solución y por tanto, la app es una alternativa que pretende dar una experiencia totalmente distinta incorporando las siguientes funcionalidades:

- Páginas anidadas y subniveles utilizando los menores recursos posibles, de modo que sea totalmente adaptable a la estructura que pida el cliente.
- Reutilizable, diseñada de tal forma que se puedan añadir o borrar secciones con datos sin complicación.
- Experiencia de usuario única, incorporando un mix de las nuevas tendencias web que están utilizando las actuales aplicaciones web de gestión empresarial.

II. Motivación

La tecnología móvil (smartphones, tablets, smartwatches..) es la más demandada actualmente, sin embargo los distintos entornos y sistemas operativos obligan a desarrollar una app para cada uno de ellos, por tanto en este aspecto ganan una gran ventaja las aplicaciones web, ya que cualquier dispositivo que incorpore un navegador será capaz de ejecutarlas, con la única preocupación por parte del desarrollador de que tenga un diseño responsive, de modo que se adapte correctamente a todos los cambios de resolución.

Todo esto, sumado a la popularidad de JavaScript como sustitución de PHP y otros lenguajes del lado servidor por su liberación de trabajo, hace muy atractiva la idea de crear una app basada en este lenguaje.

La idea de este proyecto nace del caso real de un cliente para el cual he diseñado, desarrollado e implantado una aplicación de gestión completamente nueva puesto que la estructura de la anterior no fué correctamente diseñada y las tecnologías usadas quedaron anticuadas, lo cual acabó limitando sus funciones y haciendo tedioso realizar cambios e incorporar mejoras. Gracias a ello puse énfasis en muchas de las necesidades generales de cualquier aplicación web de gestión y especialmente en la capacidad de adaptación, de modo que ese fuese el propósito principal de la aplicación, ofrecer soluciones para diversos negocios a través de una interfaz única.

III. REQUERIMIENTOS DE HARDWARE

En cuanto a los requerimientos de hardware utilizados a lo largo del desarrollo de la aplicación destacamos la parte de testing ya que AngularJS ofrece una gran variedad de APIs, plugins y soluciones online para ello y por tanto se decidió alquilar un servidor gratuito de mocking llamado Mockable IO con varios dominios para realizar todas las llamadas y peticiones necesarias al servicio REST.

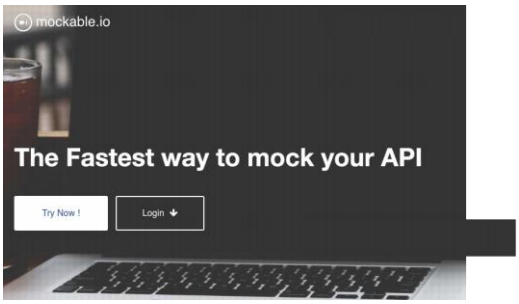

Fig 1. Servicio de mocking utilizado para las llamadas REST [1]

En cuanto a los requerimientos de hardware que deben utilizar los usuarios finales, simplemente será necesario un dispositivo que permita establecer conexión a Internet.

IV. REQUERIMIENTOS DE SOFTWARE

Para el desarrollo de la parte servidor se ha utilizado NodeJS (8.11.2 LTS) y el gestor de paquetes NPM que viene integrado en su instalación, el cual ha sido necesario para lanzar la app e instalar nuevos módulos de forma global.
Adicionalmente se ha instalado Bower, un gestor de dependencias que ha permitido la instalación de nuevos módulos para el proyecto.

El IDE utilizado tanto para la parte lógica como visual ha sido WebStorm, creado por JetBrains y diseñado para trabajar con Javascript, con soporte nativo para muchos lenguajes web y frameworks como AngularJS o React.

En cuanto a los requerimientos de software que deben utilizar los usuarios finales, será necesario un browser (navegador) de Internet para acceder a la web API, preferiblemente Google Chrome (testeado en la v 67.0.3396.79).

V. DESARROLLO

El desarrollo del proyecto se lleva a cabo en cuatro partes fundamentales divididas en las siguientes categorías:

La primera parte es la elección del framework, que consiste en la búsqueda y selección de tecnologías argumentando los motivos para llevar a cabo todo el proceso de desarrollo.

La segunda parte es la propuesta de Experiencia de Usuario y Diseño, donde se detallan y argumentan las fases generales de UX y se muestra parte del diseño visual de la app.

La tercera parte es el desarrollo de la lógica de la aplicación, donde se razona y se explica todo el desarrollo a nivel interno explicando las distintas funcionalidades.

Por último, Testing es la última fase en la que se explica todo lo relacionado con pruebas y mocking prototipado.

A.Primera parte; Elección de Framework

Hay una gran variedad de frameworks para construir aplicaciones web, para escogerlo he decidido basarme en los principales objetivos que quería lograr en la aplicación:

- Mayor fluidez posible en UX
- Uso de patrón MVC (Modelo, Vista, Controlador)
- Diseño de elementos organizado y reutilizable

Aquí es donde nace la idea de utilizar AngularJS como framework ya que la comunicación entre cliente y servidor se realiza de forma transparente al usuario, de modo que se logra que este tenga la sensación de no abandonar nunca la página principal de la app.

Además, provee una estructura organizativa separando la capa de presentación, lógica y componentes y esta última es la que permite que cada elemento tenga su propia lógica (a través del uso de directivas) logrando que se componga de módulos independientes y se eviten problemas de estado compartido.

Fig 2. AngularJS, Framework JS Open Source usado para la app [2]

Hablando en términos generales, AngularJS es una tecnología del lado del cliente, un framework de JavaScript de código abierto, mantenido por Google y utilizado para crear y mantener SPA's (aplicaciones web de una sola página).

Permite construir aplicaciones web modernas e interactivas mediante el aumento de nivel de abstracción entre el desarrollador y las tareas de desarrollo más comunes en una aplicación web. Es compatible con todos los navegadores de última generación (Chrome, Firefox, Safari, Opera, Webkits, IE9+) y además se encarga de funciones avanzadas de apps modernas tales como: separación de lógica, modelo de datos y vistas; servicios Ajax, inyección de dependencias, historial de navegador, testing, etc.

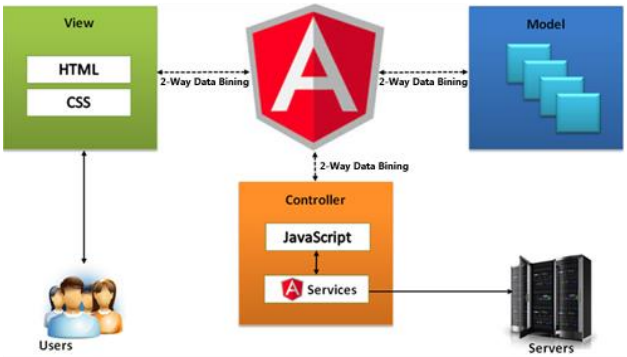

Fig 3. Diagrama orientativo del funcionamiento de AngularJS [3]

B. Segunda parte; Propuesta de UX/UI y Diseño

Esta sección comprende una parte muy importante en el desarrollo del proyecto ya que es uno de los puntos más fuertes de AngularJS tal y como se mencionó anteriormente, de modo que condicionará la fluidez y usabilidad de la app.

Para llevar a cabo el punto se plantea una Propuesta de Diseño UX dividida en las siguientes fases:

1. **Contexto**. Se parte del planteamiento de webs con mala disposición de elementos y donde no se esclarece donde accede el usuario, de modo que la idea es cambiar esa visión por completo.
2. **Visión**. Gracias al framework y la composición del proyecto se da la sensación al usuario de permanecer en la misma página, por lo que los elementos expuestos a continuación serán el pilar de la app.
3. **Propuesta.** Donde se incluyen las siguientes fases:

   a. Estrategia. Se opta por buscar errores comunes y mejoras a nivel global en varias Web APIs existentes, integrando nuevos conceptos y enfoques de la arquitectura de información y diseño de interacción.
   b. Metodología. El proceso se lleva a cabo incluyendo temporalmente las funcionalidades y para sustentar el argumento se prueban con datos reales, herramientas de testing como Mock-ups.
   c. Herramientas a desarrollar. Fundamentalmente se hará uso del plug-in de Jquery DataTables y de tabs (pestañas) ligado al enrutamiento de la app.
   d. Objetivos. Finalmente se integran las siguientes mejoras en los siguientes elementos:

   - Barra de navegación
     - Distintivo en cada sección/nivel/subnivel en el que se hace clic, para esclarecer el acceso.
     - Botón para ocultar/mostrar barra, de modo que no siempre tenga porqué ocupar el área de trabajo.
     - Esconder el resto de secciones cada vez que se pulsa una, de modo que el usuario siempre tenga una visión clara de donde accede.
   - Tablas de datos
     - Opciones de exportación de los datos de la tabla a distintos formatos (CSV, Excel, PDF, etc)
     - Búsqueda inmediata de filas usando cualquiera de los campos como filtro sin necesidad de ejecutar consultas innecesarias a BBDD.
     - Visualización de información clara, posibilidad de orden alfabético de columnas, etc.
   - Navegación entre secciones
     - Organizada a través de un sistema de pestañas para una disposición clara y organizativa de los subniveles con posibilidad de acceso a una subsección a través del sistema de pestañas o a través de un subnivel de la barra de navegación.

4. **Tiempos**. La planificación de todas las fases generales de la propuesta UX se ha llevado a cabo en un periodo aproximado de tres meses, en el cual se pueden observar con más detalle las tareas y fechas estimativas a través de un Diagrama Gantt:

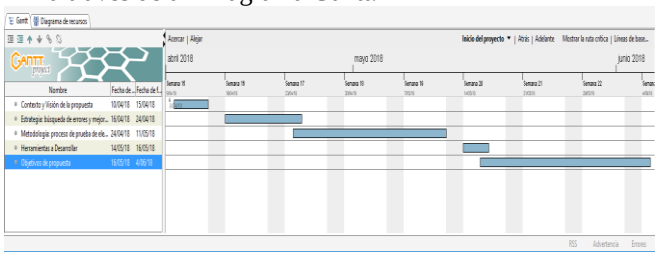

Fig 4. Diagrama Gantt utilizado en la propuesta de diseño UX [4]

En cuanto al diseño visual de la app, se ha escogido utilizar como plantilla AdminBSB – Material Design, un template gratuito de administración desarrollado con Bootstrap y Google Material Design.
Integra muchos de los elementos más actualizados del mercado, ejemplos de usos organizados por secciones y ha sido de gran utilidad para el proyecto ya que muchos de ellos han sido reutilizados en componentes y módulos independientes.

Fig 5. Template utilizado para el diseño visual de la app [5]

C. Tercera parte; Desarrollo de la lógica de la aplicación

Tras la elección de tecnologías y diseño junto con los resultados obtenidos, se procede con el desarrollo de la parte lógica del proyecto. Esta a su vez se divide en tres partes, una primera en la que se explican los primeros pasos del proyecto, la segunda en la que se explica su estructura y una última en la que se desarrollan los componentes y funcionalidades principales.

1) Creación de proyecto y sistema de enrutamiento

Para comenzar con el proyecto en AngularJS, he escogido la opción de utilizar esta plantilla general para proyectos facilitada en la documentación oficial de Angular, que no es mas que un esqueleto con una estructura de proyecto formada con un par de módulos de ejemplo. Una vez instalada y ejecutada con los comandos correspondientes, se comienza a modificar por completo.

AngularJS está especialmente indicado para hacer aplicaciones de una sola página (concepto conocido como SPA), y este concepto unido al sistema de enrutamiento nos permite la posibilidad de representar URLs distintas simulando un sistema de navegación en una única página, sin salirnos nunca de la página incial.

Esto entre otras cosas permite memorizar rutas profundas dentro de la app y tener enlaces que lleven a partes internas (deeplinks), sin necesidad de acceder desde la página inicial. Facilita también el uso de favoritos y del historial del navegador, ya que gracias a las rutas internas, se puede guardar un estado concreto de la app y navegar hacia delante y atrás con los botones del navegador. Además, mantiene las vistas en archivos independientes, por lo que reduce su complejidad y administra los controladores que facilitarán el procesamiento en ellas.

Partiendo de este concepto, una de las primeras cuestiones que surgió en el planteamiento de la aplicación fué el sistema de enrutamiento, el cual varía en función de la complejidad de las vistas que se quiera gestionar para la página.

Hay dos módulos principales para gestionar el enrutamiento en AngularJS:

- **ngRoute**. Es parte del core oficial del framework AngularJS y es un módulo potente y adecuado para vistas y escenarios básicos.
- **ui-router**. Adecuada para vistas anidadas, aplicaciones complejas y desarrollada por la comunidad (Third Party), además de soportar lo mismo que ngRoute junto con muchas funciones extra.

A simple vista podemos deducir que la opción más recomendable es ui-router, sin embargo leyendo documentación y viendo opiniones deja claro que realmente depende del tipo de aplicación que queramos desarrollar.

En el caso de este proyecto se desean cargar vistas complejas de forma simultanea en una misma página, por tanto ngRoute queda limitado. Se han realizado librerias de terceros como angular-route-segment para cubrir dicha limitación con ngRoute, sin embargo he preferido hacerlo de forma nativa con ui-router y además por los siguientes motivos:

- ✓ Permite tener enlaces en función de los nombres de estado, de modo que con cambiar el nombre de uno se actualizan todos los enlaces.
- ✓ Los decorators permiten crear rutas dinámicas basadas en la URL a la que se trata de acceder.
- ✓ Los estados ayudan a mapear y acceder a diferente información sobre distintos estados a los que se le puede pasar facilmente información a través de parámetros.
- ✓ Permite determinar facilmente si estás en un estado o en el padre de un estado para ajustar un elemento de la interfaz en las vistas.

Ui-router utiliza el patrón composite view que es uno de los clásicos a la hora de desarrollar la capa de presentación y es el que define que la vista puede tener varias subvistas que se vayan actualizando de forma independiente.

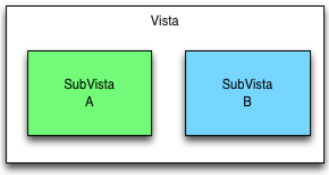

Fig 6. Funcionamiento de ui-router y composite view [6]

2) Estructura del proyecto

Tras escoger el sistema de enrutamiento, descargar su módulo correspondiente con el gestor de paquetes Bower e inyectarlo en el proyecto, se realiza la estructura del proyecto creando los módulos correspondientes y para enrutarlos entre sí con ui-router se usan los llamados states, generando el siguiente diagrama:

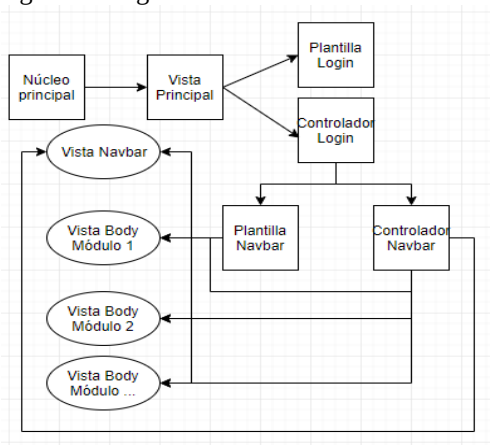

Fig 7. Diagrama de composición de la aplicación [7]

Explicando el diagrama con más detalle, la estructura del proyecto parte de un controlador/núcleo principal (app.js) y una vista principal (index.html), desde los que se carga el primer módulo (Login) que llama al segundo (Barra de navegación) que será el que pase asumir el control del resto de módulos de la app, de modo que además de cargar la vista de la barra de navegación lateral, cargará la parte "body" de cada módulo correspondiente.

Tras crear el esqueleto de la app este sería el resultado:

Fig 8. Ejemplo de página de inicio de la web api [8]

3) Componentes y funcionalidades

En esta sección se expondrán muchas de las principales funciones llevadas a cabo explicando previamente los elementos propios de AngularJS utilizados y el desarrollo de la misma.

Scopes. Son el núcleo fundamental de las apps de Angular, en ellos se guarda la información de los modelos que se representan en la vista y también los atributos que se usan para manejar la lógica de la misma. Se manejan principalmente desde los controladores y las directivas.

RootScope. Es un contexto de trabajo global para toda la app, cuyos datos se guardan en todo momento mientras se está ejecutando la app.

Niveles y subniveles de navegación: se decide guardar estos valores a nivel de SessionStorage para la persistencia de datos mientras dura la sesión de navegación, para ello se establecen métodos getters y setters a nivel de RootScope en el núcleo principal de la app y se inyectan en cada módulo haciendo uso de scopes y Contextos que serán pasados posteriormente en la vista, de modo que compruebe constantemente el nivel/subnivel en el que se encuentra y aplique los estilos correspondientes.

Una vez diseñada la funcionalidad del sistema, se establece el patrón de colorear y esclarecer el la sección, nivel y subnivel tal y como se puede observar en un ejemplo a continuación.

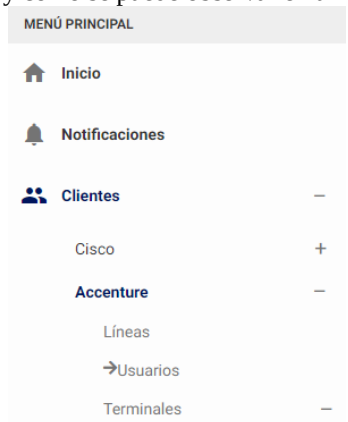

Fig 9. Ejemplo de sistema de niveles y subniveles de la app [9]

Filtros. Permiten modificar la forma en la que se representa la información al usuario final. AngularJS tiene muchos filtros nativos creados para uso directo y una gran facilidad para crear filtros propios.

Soporte Multilenguaje: se ha utilizado la funcionalidad de los filtros para comenzar a crear un soporte multilenguaje en Español/Inglés, el cual se guarda a nivel de SessionStorage y es aplicado a la vista de la barra de navegación que forma parte de una directiva. Las traducciones y los parámetros quedan guardados en un archivo properties que recoge el núcleo de la app y lo aplica a través del Contexto.

Controladores. Son los encargados de inicializar y modificar la información que tienen los Scopes según las necesidades. Se pueden crear acciones personalizadas que se pueden llamar a través de las vistas, de modo que se pueda llamar a funciones de $scope como si se referenciase una variable de datos.

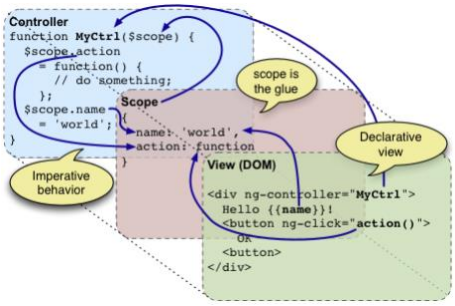

Fig 10. Interacción entre Controlador, Vista y Scope en Angular [10]

Módulos. Cada sección de la app tiene uno propio con una vista y un controlador, el cual recoge los datos contenidos por el $scope para definir la lógica.

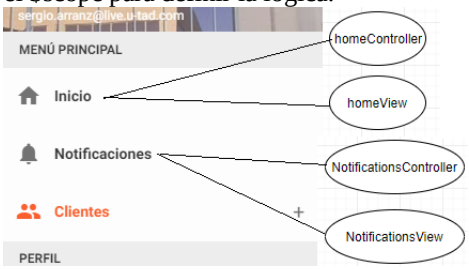

Fig 11. Ejemplo de módulos con su Vista y Controlador asignado [11]

Directivas. Son marcas en los elementos del árbol DOM, en los nodos del HTML, que son las que indican al compilador de Angular que debe añadir un comportamiento determinado a un elemento o transformarlo como corresponda. Es el único sitio recomendable donde debemos manipular el árbol DOM para que entre dentro del ciclo de vida de compilación, binding y renderización del HTML.

Son la técnica al fin y al cabo que permite crear componentes visuales propios encapsulando posibles complejidades en implementación, normalizando y parametrizando según necesidades. Estas pueden ser creadas a nivel de elemento, atributo, clase y comentario.

A continuación se explican las directivas principales y su implementación en la app, tanto las nativas del propio AngularJS como las propias para determinadas tareas:

ng-repeat. Es muy útil para representar en la vista colecciones de elementos. Dentro del elemento del DOM en el que se encuentra la directiva, y por tanto el cual se repetirá, se puede hacer referencia al objeto y realizar la estructura que se quiere repetir, así como los datos a mostrar en cada objeto del listado.

Colecciones de elementos: la directiva nativa ng-repeat ha sido utilizada en la mayor parte de elementos de la app como secciones, niveles, filas de datos de tablas,etc.

Secciones: además de utilizar ng-repeat para mostrar la colección de elementos de secciones, niveles y subniveles, se

ha creado una directiva propia en la que se le pasa un dato como parámetro para tener siempre la referencia de la sección, de modo que una vez recogida en la estructura a través del servicio REST se puedan añadir o quitar niveles/subniveles de forma dinámica y gestionar sus datos de forma independiente a pesar de estar en el mismo módulo.

Tablas de datos: es la directiva propia más completa e importante de la app, a la cual se le pasan los siguientes parámetros a nivel de scope:

- Identificador: es un nombre único para identificar cada tabla y que posteriormente pueda ser referenciada cuando se accede a ella a través de un tab/pestaña.
- Headers: son las cabeceras que contendrá la tabla con referencia, en la cual se recogerán del servicio REST a través del controlador y se pintarán en la vista a través de la directiva.
- Data: son los datos que contendrán las filas de la tabla con referencia, en la que se recogerán del servicio REST a través del controlador y se pintarán según cada condición a través de la directiva.
- Config: parámetro adicional para generar las condiciones necesarias para pintar una tabla de determinada manera.

Tras recoger estos parámetros y generar las colecciones de datos con la directiva ng-repeat con la referencia del cliente, obtendriamos en la app una tabla Jquery inteligente con los datos correspondientes pintados.

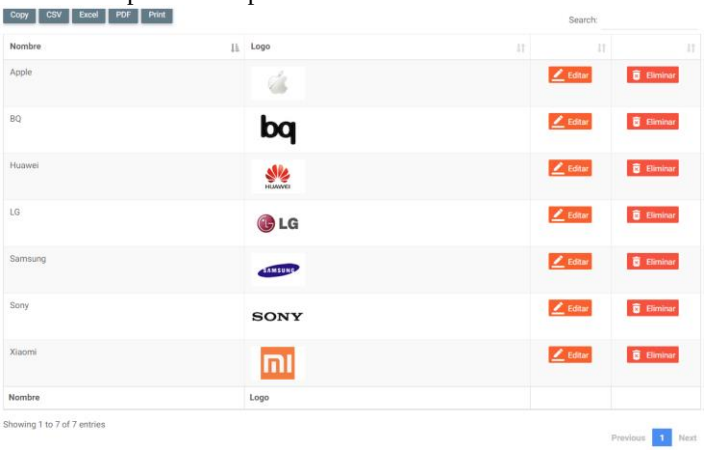

Fig 12. Ejemplo de Tabla de la app de marcas de terminales [12]

Para entender mejor el funcionamiento de la directiva, a continuación se especifican los parámetros para tres ejemplos de tablas de la app, en los que todos los datos de la tabla (cabeceras, datos, config) están formados por estructuras de datos que son solicitadas al servicio REST.

Parámetros Directiva Tablas			
identificador	***headers***	***data***	***config***
branchesTable	headersBranches	dataBranches	configBranches
modelsTable	headersModels	dataModels	configModels
storesTable	headersStores	dataStores	configStores

ng-if. Muestra el código del elemento del DOM en el que se encuentra si la condición de la directiva se cumple, pero a diferencia de ng-show, no se llega a generar el código en el DOM que interpreta el navegador en caso de no cumplirse la condición.

Condiciones Tablas: es muy importante destacar que a pesar de tener una directiva con una plantilla general donde pasamos unos parámetros para generar las tablas con unos datos específicos, es posible que un campo de una tabla queramos representarlo por ejemplo con una imágen y en otra con un input de tipo checkbox.

Aquí es donde entra en juego la directiva nativa de AngularJS ng-if, con la cual podemos hacer múltiples condiciones preguntando por objetos específicos y generando distintas tablas con la misma plantilla tal y como se puede observar en la imágen a continuación.

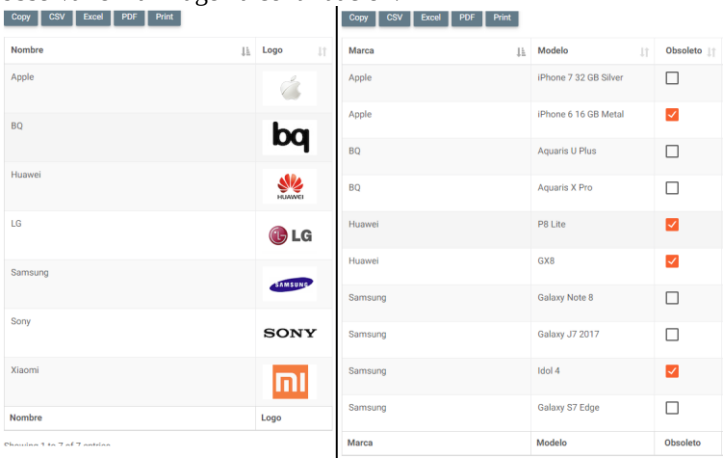

Fig 13. Ejemplos de distintas tablas generadas con una directiva [13]

ng-click. Esta directiva nativa de AngularJS permite asignar la llamada a una función del controlador que se encuentre en dicho momento activo. Solo se puede incluir en los elementos del DOM "clickables", como botones o enlaces. Además de especificar la función que se quiere llamar en la directiva, se incluirán los parámetros a la función, para que el controlador sepa sobre que objeto de todo el listado debe realizar la función.

Botones y enlaces: utilizando la directiva ng-click pasando item como parámetro y ng-if para condicionar cada botón, se ejecuta un evento específico en el controlador que pasa a ser un cuadro modal en el caso de las tablas como el siguiente:

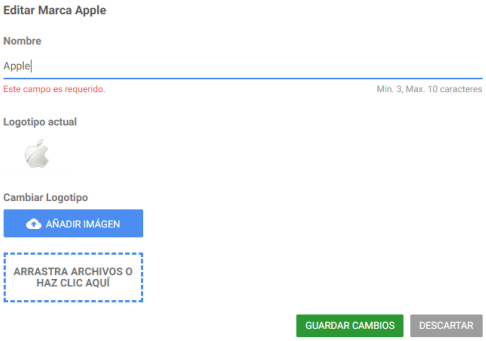

Fig 14. Ejemplo de cuadro modal condicional con ng-click [14]

Promesas. Es un método para resolver el valor de una variable de forma asíncrona. Son objetos que representan el valor de retorno o la excepción que devuelve una función, y son de gran utilidad cuando se trabaja con objetos remotos como es el caso de esta app.

Tradicionalmente en Javascript se hace uso de las llamadas "callbacks", pero dificultan el debugging, y sin embargo, con las promesas se pueden tratar con los datos de forma que ya hayan sido retornados, sin depender de que el callback se dispare, además de no disponer de una garantía a la hora de esperar la llamada.

Llamadas al servicio REST: para devolver en la app datos reales (nombres de secciones, propiedades, datos de tablas, etc) se hace uso de un servicio al que se le pasa una URL y con las promesas se devuelve una respuesta o una excepción.

D.Cuarta parte; Testing

Para llevar a cabo la parte de testing inicialmente se optó por utilizar varias estructuras de datos en los controladores de la aplicación, pero para hacerla más funcional y contrastable a las nuevas tendencias web, finalmente se recogen todos los datos haciendo llamadas a un servicio REST.

REST es cualquier interfaz entre sistemas que utilice el prtocolo HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es la alternativa más actual y en crecimiento a otros protocolos estándar de intercambio de datos como SOAP, que dispone de gran capacidad pero con mucha complejidad innecesaria cuando es preferible una manipulación de datos más sencilla con REST.

Estas pruebas se realizan a través de un servicio de mocking, que no es mas que pruebas de prototipado para servicios que permite acelerar el desarrollo de software, bien por dependencias de terceros o porque las líneas de tiempo no se cruzen y la gestión de pre-requisitos se convierta en un caos.

Para llevar a cabo la parte de testing inicialmente se optó por utilizar varias estructuras de datos en los controladores de la aplicación, pero para hacerla más funcional y contrastable a las nuevas tendencias web, finalmente se recogen todos los datos haciendo llamadas a un servicio REST.

Para esta aplicación se ha utilizado la plataforma online de mocking Mockable IO para simular las llamadas REST API con posibilidad de agregar varios dominios y configurando las peticiones de forma personalizada.

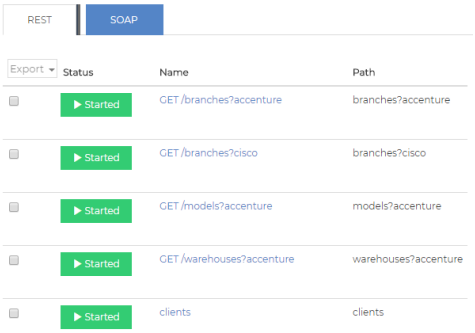

Fig 15. Ej. Dominios servicio Mocking para REST API Service [15]

Como se observa en la imágen, hay cinco dominios de ejemplo con dos clientes referenciados para llevar a cabo las pruebas. Cada una de estas URLs generadas en protocolo HTTP y HTTPS son pasadas por los servicios de la app que a través de las promesas devolverán una respuesta o una excepción.

El formato de las mismas es representado en una estructura JSON en la cual se pueden añadir, modificar o borrar datos, de modo que un ejemplo de uso sería el siguiente:

Fig 16. Ej. Estructura JSON para secciones en Mockable.io [16]

Tras devolver la estructura de secciones en el formato adecuado, el controlador gestiona los datos y ordena a la vista pintar las secciones tal y como se aprecia en la imágen:

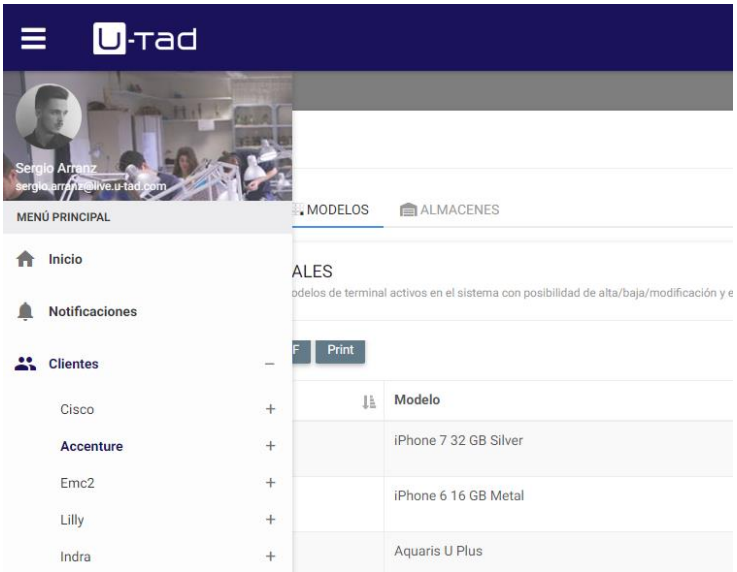

Fig 17. Ej. secciones pintadas tras recibir los datos del REST [17]

Cabe destacar que también existen muchas otras plataformas de testing y mocking para AngularJS, e incluso librerias y dependencias creadas por terceros que se inyectan directamente en el proyecto, en el caso de esta aplicación he elegido este servicio online ya que ofrece facilidad para hacer las pruebas, es muy personalizable y el plan gratuito ofrece suficientes recursos.

VI. ISSUES/DIFICULTADES

Una de las mayores dificultades en el desarrollo ha sido pensar, dibujar, llevar a cabo e implementar la estructura inicial del proyecto y conectar los distintos módulos a través del sistema de enrutamiento ya que hay muchas vias y buenas formas para llevarlo a cabo.

Otra de las dificultades ha sido entender el paso de parámetros en estados y recogida de los mismos en los controladores para pasar las referencias a través de URL y que cada módulo tenga unos datos distintos en función de estas.

Por último, el uso de una directiva y una única plantilla para todas las tablas supuso inicialmente gran complejidad para manejar distinta información y tipos de objeto en cada fila de la tabla en función de los requerimientos.

VII. FUTUROS TRABAJOS

La idea de hacer el proyecto con componentes reutilizables y escalables es que continue creciendo e incorporando funciones, por ello se continuará con el desarrollo y una de las cosas por completar es el soporte multilenguaje, el cual por el momento solo está adaptado en una parte de la aplicación y la idea es cubrir la funcionalidad por completo, e incluso añadir botones de cambio de idioma y detección de región para mostrar la página por defecto en un idioma u otro.

Para la parte de autenticación, quedaron realizados varios estados posibles sin llegar a implementarse para el caso de que un usuario no recordase sus credenciales o hubiese que redirigirle a una página intermedia de creación de nueva contraseña tras recibir la URL a través del mail.

En el caso de que hubiese varios perfiles para entrar a administrar la aplicación (Gestor, Moderador, etc) sería necesaria la gestión de roles para mostrar/esconder información en función de las necesidades del cliente.

Otro de los puntos a mejorar en el futuro es toda la parte de comercialización para la que estaba pensado el proyecto en un inicio puesto que finalmente ha quedado orientado a gestión de inventarios y no cubre otras grandes funcionalidades que pueda tener un CMS o un gran panel de administración como pueden ser gráficos, stats de tablas, historial de accesos, notificaciones y un área personal de edición de perfíl.

Fig 18. Módulo oficial de Firebase para AngularJS [15]

VIII. AGRADECIMIENTOS

Quiero dar las gracias en este proyecto a mi familia y amigos por el apoyo en todo momento, a mi profesor Jonathan Bar-Magen Numhauser por su empeño y dedicación explicando un framework al que posteriormente me dedicaría por completo para el proyecto (Angular) y al personal de mi empresa de prácticas de Grado Superior (Telcomunity), en especial a Alejandro Tirado por todo su esfuerzo, dedicación y paciencia ayudándome con el proyecto en general.

IX. REFERENCIAS

[1] Güray Yarar, desarrollador del template de Google Material Design utilizado para el diseño visual de la aplicación.
https://github.com/gurayyarar/AdminBSBMaterialDesign
[2] Angular, por la estructura general de proyecto “angular-seed” para comenzar con el desarrollo.
https://github.com/angular/angular-seed
[3] Documentación oficial de AngularJS para el uso y referencias de muchos de los elementos del lenguaje.
https://docs.angularjs.org/api
[4] Atraura, blog de tecnología para la explicación teórica de la elección de framework.
https://www.atraura.com/angularjs/
[5] AyerViernes, blog de Experiencia de Usuario con las fases de preparación de una propuesta de diseño UX.
https://www.ayerviernes.com/blog/pasos-para-preparar-una-propuesta-comercial-de-diseno-ux
[6] GanttProject, por su herramienta para la administración de proyectos utilizada en la última fase de la propuesta UX.
https://www.ganttproject.biz/
[6] GanttProject, por su herramienta para la administración de proyectos utilizada en la última fase de la propuesta UX.
https://www.ganttproject.biz/
[7] StackOverflow, por la ayuda en gran parte del desarrollo y en la elección y explicación del sistema de enrutamiento
https://stackoverflow.com.
[8] Draw IO, por su herramienta para dibujar el diagrama de la estructura del proyecto.
https://www.draw.io/
[9] TFG de Fco Javier Avilés López, para la explicación de muchos de los componentes de AngularJS.
http://repositorio.upct.es/bitstream/handle/10317/4411/tfg396.pdf
[10] DataTables, por su plug-in JQuery para todas las tablas.
https://datatables.net/
[11] Mockable IO, por su plataforma online de mocking para conectar la aplicación a un servicio REST.
https://www.mockable.io/