Escuela Politécnica Superior

Diseño e Implementación de un Tablón de Anuncios para Venta de Segunda Mano Adaptado al Usuario

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Francisco Javier Galiana Cano

Tutor/es:

Estela Saquete Boró

Septiembre 2017

1. JUSTIFICACIÓN Y OBJETIVOS

El objetivo principal es analizar, diseñar y desarrollar una aplicación que consta de un tablón de anuncios para la venta de segunda mano y que este sea adaptado al usuario.

Para ello, vamos a apoyarnos en una API REST en el lado del servidor parte fundamental del proyecto donde realizar llamadas a través de la aplicación cliente que nos ayudará también a fortalecer dicha aplicación cliente. Profundizaremos en el aspecto teórico-práctico de la personalización.

Cuando diseñamos una aplicación intentamos lograr unos objetivos, el principal sería satisfacer una necesidad de un grupo de usuarios o facilitar una acción común a un grupo de estos, por otro lado en el marco propio de los creadores de ésta, la monetización de esta para lograr unos beneficios. Pero todos ellos evolucionarán favorablemente siempre que vayan en la misma dirección y esta es gracias a la ayuda de los usuarios finales (o clientes) que consumen dicha aplicación. [1]

Si decimos que la consecución de los objetivos perseguidos está condicionada por la satisfacción del usuario final debemos comprender que los factores o atributos de calidad de una aplicación que influirán en dicha satisfacción estarían relacionados con: calidad y utilidad de los contenidos, calidad del servicio, calidad del diseño de dicha aplicación y además la adaptabilidad de la aplicación a cada usuario específico.

Este último concepto, sin menospreciar al resto, la adaptabilidad de la aplicación a cada usuario específico es el que nos hace tomar la decisión de realizar dicho proyecto puesto que tenemos en el usuario final un producto estrella que nos muestra estadísticas que servirán resultados que nos ofrecerán si se están llevando a cabo tanto objetivos comunes como personales, así como una gran ayuda a la hora de ofrecer un mejor servicio a cada tipo de usuario de la forma más personalizada posible.

2. AGRADECIMIENTOS

Agradezco a mi tutora Estela Saquete su ayuda, ideas, conocimientos y apoyo durante todo el proyecto para hacer frente a cualquier tipo de dificultad, a mis profesores en estos años de universidad que me han brindado cada uno de los conocimientos suficientes para enfrentarme a cualquier problema del grado que quería estudiar, a los compañeros que han estado en esta etapa y a mi familia y amigos que me han apoyado en cualquier momento.

3. ÍNDICE

ÍNDICE DE CONTENIDOS

1. Justificación, 2
2. Agradecimientos, 3
3. Índice, 4
   - 3.1 Índice de contenidos, 4
   - 3.2 Índice de figuras, 6

4 Introducción, 8

5 Marco teórico o estado del arte, 9

6 Objetivos, 11

7 Metodología, 12
   - 7.1 Tecnologías y herramientas empleadas, 13
     - 7.1.1 ¿Por qué utilizamos NodeJS?, 15
     - 7.1.2 ¿Por qué utilizamos ReactJS?, 16
   - 7.2 Tamaño de muestra, 17
   - 7.3 Técnicas y herramientas de almacenamiento de datos, 17
     - 7.3.1 ¿Por qué utilizamos MongoDB?, 17

8 Cuerpo del trabajo, 19
   - 8.1 Planificación, 19
   - 8.2 Análisis funcional, 20
   - 8.3 Metodología de desarrollo software, 24
   - 8.4 Implementación, 25
     - 8.4.1 Patrón MVC, 26
     - 8.4.2 Arquitectura, 27
     - 8.4.3 Personalización, 30
     - 8.4.4 Implementación del Back-End, 32
     - 8.4.5 Seguridad, 36
     - 8.4.6 Realización de pruebas, 48
     - 8.4.7 Implementación del Front-End, 50
   - 8.5 Despliegue del proyecto, 55
   - 8.6 Casos de estudio, 56

8.7 Branding, 59

8.8 Problemas, 60

8.9 Mejoras, 61

9 Conclusiones63

10 Bibliografía, 65

ÍNDICE DE FIGURAS

Figura 1 – Esquema de la economía Colaborativa por IEBS [2], 9

Figura 2 – Distribución de las diferentes partes del proyecto, 19

Figura 3 – Descripción tablero Planificación TFG, 20

Figura 4 – Diagrama Caso de Uso, 22

Figura 5 – Proyecto en Git, 25

Figura 6 – Esquema modelo MVC por Código Facilito, 26

Figura 7 – Estructura 2 partes de nuestro proyecto [19], 27

Figura 8 – Estructura de nuestro proyecto, 28

Figura 9 – Descripción del modelo Entidad-Relación, 34

Figura 10 – Función de Registro en el controlador user.js, 35

Figura 11 – Peticiones http en el directorio index.js de routes, 36

Figura 12 – Función isAuth en el directorio middleware, 36

Figura 13 – Ejemplo de JWT [16], 37

Figura 14 – Funciones createToken y decodeToken, 38

Figura 15 – Método signIn de controlador user.js, 39

Figura 16 – Método pre-guardado en el modelo User.js, 40

Figura 17 – Método que implementa obtención producto especificado, 41

Figura 18 – Método que implementa obtención todos los productos, 42

Figura 19 – Método que muestra estructura de locations, 43

Figura 20 – Método que implementa obtención productos cercanos, 44

Figura 21 – Método que implementa obtención productos de un usuario, 45

Figura 22 – Método guarda un producto, 46

Figura 23 – Método actualiza un producto, 46

Figura 24 – Método borra un producto, 47

Figura 25 – Cuerpo de la petición http en JSON, 48

Figura 26 – Tres ejemplos testing con Mocha, 49

Figura 27 – Estructura Front-End de Thingy, 50

Figura 28 – Modelo DOM [24], 51

Figura 29 – Ficheros contenidos en “js”, 53

Figura 30 – Parte del archivo App.js, 53

Figura 31 – Parte Componente Product.js, 55

Figura 32 – Interfaz Login y Registro, 56

Figura 33 – Interfaz “Todos los productos”, 57

Figura 34 – Interfaz “Información detallada”, 58

Figura 35 – Interfaz “En mi zona”, 58

Figura 36 – Interfaz “Mi perfil“, 58

Figura 37 – Interfaz “Mis Productos”, 59

Figura 38 – Logo de Thingy, 60

4. INTRODUCCIÓN

Dicho proyecto consiste en analizar, implementar y desarrollar un tablón de anuncios para la venta de segunda mano adaptado al usuario, como correctamente definimos se quiere adaptar al usuario, pero ¿por qué le damos tanta importancia a esto y qué sucede en la actualidad?

Cualquier aplicación web está condicionada por la satisfacción del usuario final, por tanto existirán unos factores que influirán en dicha satisfacción entre ellos la calidad del servicio y de sus contenidos pero sobre todo aquel donde situamos este proyecto, en la calidad del diseño de la aplicación en especial adaptada al usuario.

Sabemos por estudios que en 2016 (*Informe por Ditrendia*) los españoles preferían comprar desde el móvil a través del sitio web. Sumado a esto, cuando realizan una compra el 77,7% la realiza a través del ordenador. Además este año se vio duplicado el tiempo que los internautas dedicaban a realizar búsquedas y compras online, unido a otro dato importante, y es que es destacable que al realizar una compra móvil, la usabilidad (permitir encontrar fácilmente la información o navegar de forma rápida) puede más que la marca del producto ya que estos usuarios tienen claras sus preferencias por lo que quieren comprar rápidamente de manera sencilla y cómoda. Justo de esta serie de datos es de donde surge la finalidad de nuestro proyecto, que no es más que adaptar lo más posible una aplicación de este sector para que el usuario logre su objetivo lo antes posible quedando así satisfecho.

5. MARCO TEÓRICO O ESTADO DEL ARTE

En cuanto a la base de la investigación el objetivo es sencillo y es trasladar el modelo de los clasificados de la prensa. Dentro de los tiempos que corren, encontramos un número elevado de aplicaciones que pueden ir en una dirección similar hacia la que nosotros investigamos, centrando el desarrollo en la búsqueda de conexión entre individuos. A continuación encontramos un esquema de esta práctica y sus ventajas que denominamos economía colaborativa (promueve el intercambio de bienes o servicios a partir de un enfoque de aprovechamiento de los recursos y por tanto beneficio y ahorro para todas las partes), puesta en contacto para que ellos ya determinen como vender los productos que unos compradores ofertan. [3]

Figura 1 - Esquema de la economía Colaborativa por IEBS [2]

Intentan simplificar la aplicación al máximo para mejorar la usabilidad y lograr el objetivo mencionado de la forma más ágil posible. Tras realizar el estudio previo hemos de mencionar las 4 aplicaciones que más nos pueden llamar la atención:

- Wallapop [4], (originada en 2013) intentan explotar las funciones que nos da el móvil, como hemos comentado, presente siempre en cada momento, ya sea en el bolsillo o en el bolso además de que la gran mayoría suele estar conectado a

internet. Por ello, utilizan la geolocalización de un teléfono inteligente para actuar de filtro en las búsquedas en dicha aplicación, esto hace que tenga gran implicación para la experiencia del usuario. Explotando así la relevancia frente a la importancia de una forma divertida.

- LetGo [5], (originada en 2015), similar a Wallapop pero intentando agilizar el proceso de publicación de un anuncio identificando el título automáticamente a partir de la foto, ayudando al usuario a comunicarse con mensajes predeterminados además de la navegación utilizando gestos con los dedos a partir de la ley de Fitts para calcular el tiempo que un usuario tarda en llegar hasta el target deseado.
- Milanuncios [6] y Vibbo [7] (originadas en 2005 y 1978 con el nombre del portal web Segundamano) son similares pero no se centran tanto en la rapidez por añadir un anuncio sino que tienen un gran abanico de artículos de cualquier categoría y ahí reside su potencial.

Sin embargo, aunque encontramos aplicaciones similares en estas aún no se ha potenciado en gran medida el uso de datos de los usuarios para favorecer el filtrado o mejorar el diseño de una aplicación adaptándola a este de forma personalizada que es lo que intentamos con nuestro proyecto.

6. OBJETIVOS

Como ya hemos descrito, el principal objetivo de este proyecto es la creación de una aplicación denominada, Thingy, llevando a cabo las tareas para la realización de un muro de venta de segunda mano lo más personalizado a cada usuario.

Además, el objetivo que tiene Thingy es facilitar al usuario la tarea de publicar un anuncio reduciendo el número de pasos así como que la búsqueda de un producto sea lo más cercana y exitosa posible para el usuario en cuestión. Por ello, recolectamos datos que guardaremos en la base de datos de forma agrupada y que nos harán servir la información al usuario de forma más precisa y correcta por medio de búsquedas anteriores, por su localización, búsquedas de los usuarios de su zona, facilidad a la hora de acceder a las opciones, etc.

Para lograr dichos objetivos, hemos marcado una serie de pautas a seguir:

- Estudio sobre el funcionamiento de las aplicaciones compraventa
- Estudio de la forma de operar de dicha idea
- Planificación del proyecto
- Realización del prototipo
- Definición del modelo de negocio
- Realización de estudio de mercado
- Implementación de la aplicación

7. METODOLOGÍA

Thingy se trata de una aplicación web dedicada a la compraventa de segunda mano y para adaptarla al usuario es necesario la continua recolección de datos, por tanto vamos a trabajar con grandes volúmenes de datos de distintos usuarios. Esta información será almacenada, posteriormente se filtrará y procesará para para una presentación al usuario de la forma más personalizada posible. Debido a la expansión futura de dicha aplicación, recolectaremos toda la información aunque en una primera entrega descartaremos parte de la información para futuras mejoras.

¿Cómo hemos seguido dicha investigación?

Hemos utilizado lo que denominan Método Tecnológico, descrito por Mario Bunge, 2007:

- Formularemos el problema
- Generamos una hipótesis, un artefacto capaz de solucionar el problema
- Realizamos prueba o contrastación de hipótesis, en ella:
    - Adquirimos el conocimiento antecedente necesario o sobre otros competidores
    - Invención de reglas técnicas y del artefacto en esquema
    - Realizamos una descripción detallada del plan
    - Realizamos pruebas analizando la información recogida y evaluamos
    - Corregimos buscando soluciones para mejorar este y optimizándolo.
- Llegamos con las conclusiones de esto al artefacto

Tras la investigación pasamos a explicar cómo se ha desarrollado el proyecto en el siguiente apartado.

7.1 TECNOLOGÍAS Y HERRAMIENTAS EMPLEADAS

Ubuntu

Ubuntu será nuestro sistema operativo basado en GNU/Linux y que se distribuye como software libre, el cual incluye su propio entorno de escritorio denominado Unity. Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario. Está compuesto de múltiple software normalmente distribuido bajo una licencia libre o de código abierto.

Trello [8]

Lo utilizaremos para la planificación, trata de un software de administración de proyectos con interfaz web, cliente para iOS y Android para organizar proyectos. Empleando el sistema Kanban, para el registro de actividades con tarjetas virtuales organiza tareas, permite agregar listas, adjuntar archivos, etiquetar eventos, agregar comentarios y compartir tableros.

Visio [9]

Software de dibujo vectorial para Microsoft Windows. Las herramientas que lo componen permiten realizar diagramas de oficinas, diagramas de bases de datos, diagramas de flujo de programas, UML, etc. Utilizaremos para la especificación de la Base de datos.

GitHub [10]

Se trata de una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Es donde almacenamos el API REST y la aplicación cliente. El enlace es el siguiente: https://github.com/frangaliana/rest-api-thingy

Atom [11]

Editor de código fuente de código abierto para macOS, Linux, y Windows con soporte para plug-ins escritos en Node.js y control de versiones Git integrado, desarrollado por GitHub.

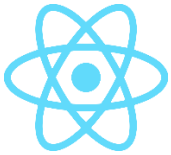
MongoDB [12]

(De la palabra en inglés "humongous" que significa enorme) es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto.

MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En lugar de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos similares a JSON con un esquema dinámico (MongoDB utiliza una especificación llamada BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Node JS [13]

Es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Lo utilizamos para la creación del API Rest junto al framework Express con las consiguientes librerías.



React [14]

React (también llamada React.js o ReactJS) es una biblioteca Javascript de código abierto para crear interfaces de usuario con el objetivo de animar al desarrollo de aplicaciones en una sola página. Es mantenido por Facebook, Instagram y una comunidad de desarrolladores independientes y compañías. Con ella realizaremos la interfaz del cliente.

Aquí hemos descrito brevemente cada uno de estos, pero para el desarrollo de la parte de software nos centraremos en 3 pilares: NodeJS, MongoDB (con Mongoose para ORM) y React

7.1.1 ¿Por qué utilizamos NodeJS?

La razón primordial es porque su lenguaje es ECMAScript (Javascript) y por lo tanto servidor como cliente utilizarán el mismo lenguaje, pero sobre todo por temas de rendimiento, curva rápida de aprendizaje e innovación.

- Es un servidor más veloz que otros
- Tiene una tecnología más joven con una gran comunidad detrás

- Implementación más cercana (curva de aprendizaje más fácil)
- Al realizar peticiones de muchos recursos a la misma vez, como no bloquea hilos, nos permite atender a todas las peticiones de forma óptima, lo que es un punto a favor para la escalabilidad.

Sumado a ello que utilizamos Express, un framework web bastante ligero con el que obtenemos características para aplicaciones web y móviles sólidas entre ellos: Routing, con el que asociamos de forma sencilla un método HTTP y una URL con el código a ejecutar; se tiene un método más flexible que write para generar el contenido de respuesta y una integración de motores de plantillas para aplicaciones web "clásicas" que devuelven HTML/CSS al cliente en lugar de datos "en crudo" en JSON. Por todo ello y por el uso principal de estas dos nos servirán para desarrollar nuestra API REST.

7.1.2 ¿Por qué utilizamos ReactJS?

Como el resto de componentes a desarrollar desarrolla aplicaciones con ECMAScript 5 o 6. La principal característica es que actualiza las vistas para reflejar cambios en los datos de la aplicación, para esto ocupa un DOM Virtual que aumenta la velocidad en la que se pueden modificar los elementos de la página por ello ganamos en tiempo. Tiene una curva de aprendizaje sencilla. Los componentes que se desarrollan son más fáciles de mantener individualmente. Ayudamos al cliente que es nuestro jefe, con la característica anterior, la capacidad de rápido re-renderizado mejora la experiencia del usuario. Aunque no ha sido utilizado en este proyecto, Flux es una alternativa muy buena a MVC puesto que el flujo de datos en una sola dirección ayuda a la mantenibilidad de la aplicación y mantiene los datos de los elementos del DOM consistentes.

7.2 Tamaño de muestra

De momento el volumen de datos a volcar será ínfimo con el pensado para un futuro además nos servirá para mostrar de forma más detallada aquellas operaciones que realizará nuestra aplicación o el prototipo de esta.

7.3 Técnicas y herramientas de almacenamiento de datos

7.3.1 ¿Por qué utilizamos MongoDB?

Algunas de las características que nos hicieron decantarnos por estas eran que MongoDB es de código abierto, no suponía costo alguno. También como hemos dicho, la diferencia respecto a por ejemplo una base de datos relacional como MySQL es que en lugar de tablas los datos se almacenan en collections, los documentos son cada uno de los elementos almacenados en estas colecciones y son en formato BSON sustituyendo a las líneas y definen los campos de las columnas de una tabla SQL. Van siempre en un par clave-valor representado este valor una cifra, palabra o texto.

¿Qué lo hace realmente especial? Que mientras en MySQL todas las filas de una tabla tienen la misma estructura en MongoDB los documentos no están sujetos a un orden fijo. Con ello podemos crear nuevos campos con cualquier valor, mientras que en una base de datos relacional se necesita una reestructuración completa. La clave, única dentro de un documento, es posible que aparezca en otros documentos, algo no posible en MySQL tampoco, donde son creadas en forma de documentos incorporados o referencias.

Otra de las grandes diferencias y dato positivo a nuestro favor es el enfoque es la recuperación de datos. MongoDB no utiliza SQL como lenguaje de consulta, sino de JavaScript para la comunicación entre MongoDB y el cliente con métodos específicos de dicho lenguaje de programación valiéndose de drivers (librerías).

Por ello, buena opción para un proyecto web como el nuestro donde podemos tener grandes conjuntos de datos sin estructurar puesto que cada vez podemos necesitar guardar más información para ganar en precisión en cuanto a la personalización. Además gestionaremos un gran número de datos diferentes que deban ser almacenados y procesados rápidamente. Además la escalabilidad horizontal de este sistema de base de datos es casi ilimitada, pue estas se pueden distribuir la carga de datos en diferentes hosts a media que esta aumenta sin comprometer la seguridad.

MongoDB también facilita la creación de copias de la totalidad de datos y las pone a disposición en diferentes servidores.

Finalmente, también juega un papel importante a la hora de resumir grandes cantidades de datos de una o más fuentes. Resumiendo para este proyecto que se caracterizará ahora o en un futuro por:

- Escalabilidad: proyecto en crecimiento, aumentando número de visitas y solicitudes, por tanto demandará una mayor capacidad de respuesta por parte de las bases de datos.
- Disponibilidad: deberá de estar disponible en todo momento, incluso en caso de fallos en el servidor.
- Flexibilidad: dicho proyecto debe ajustarse en cualquier momento dinámicamente.

Para el futuro, si se distribuyese en múltiples servidores se produciría también una diferencia crucial entre MongoDB y las bases de datos relacionales. Debido a que al almacenar datos en una MongoDB presenta un pequeño intervalo de tiempo en el que el acceso de lectura solo está permitido para los datos más antiguos (denominado Consistencia Eventual). Este modelo es utilizado por MySQL, MariaDB, etc. Es decir, deriva de MongoDB.

8. CUERPO DEL TRABAJO

En este apartado describiremos el desarrollo del proyecto continuando después de la citada investigación previa, análisis funcional, describiendo la planificación previa, el modelo de negocio asociado, la metodología de desarrollo software que vamos a seguir y la consiguiente descripción técnica del sistema desarrollado, con sus características importantes, funcionamiento y posibles mejoras.

8.1 PLANIFICACIÓN

El primer paso para el desarrollo del software es la planificación de dicho proyecto (Sommerville, 2000). Sin la correcta planificación el riesgo aumenta y las posibilidades de entregar con éxito el proyecto disminuyen. La planificación es crucial para evitar fallos. Determinamos las tareas a realizar, los tiempos estimados, los recursos y el orden de estos según importancia de la funcionalidad.

Por ello para realizar esta planificación de tareas se ha utilizado como hemos comentado antes la aplicación Trello. En ella hemos dividido primero en 3 tableros como muestra la ilustración (El Backend, el FrontEnd y el desarrollo completo del proyecto en cada uno estudiaremos todas sus innovaciones).

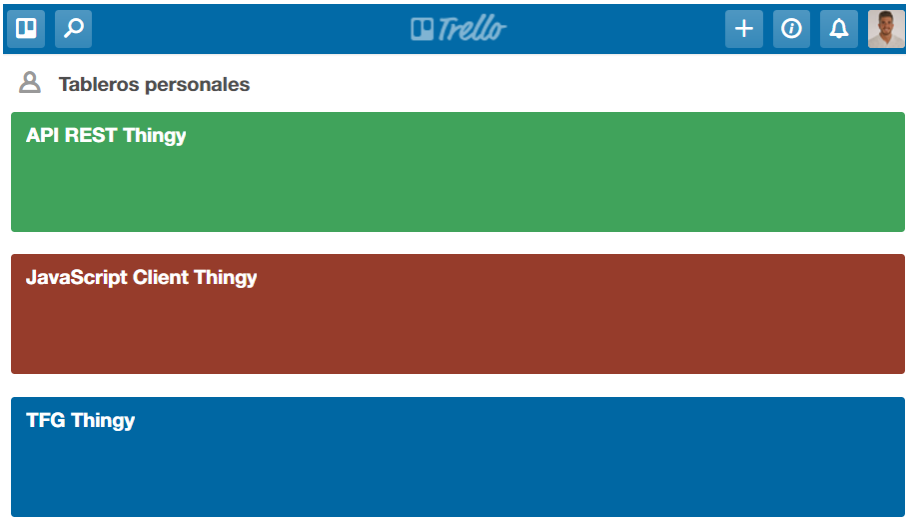


Figura 2 - Distribución de las diferentes partes del proyecto

Dentro de cada tablero podemos encontrar en una serie de categorías (utilizamos una metodología ágil con un tablero de planificación pero al estar

compuesto en este momento el equipo de una persona, reducimos las categorías también) como “Main Tasks”, “Developing” y “Finished” como podemos ver en la siguiente ilustración.

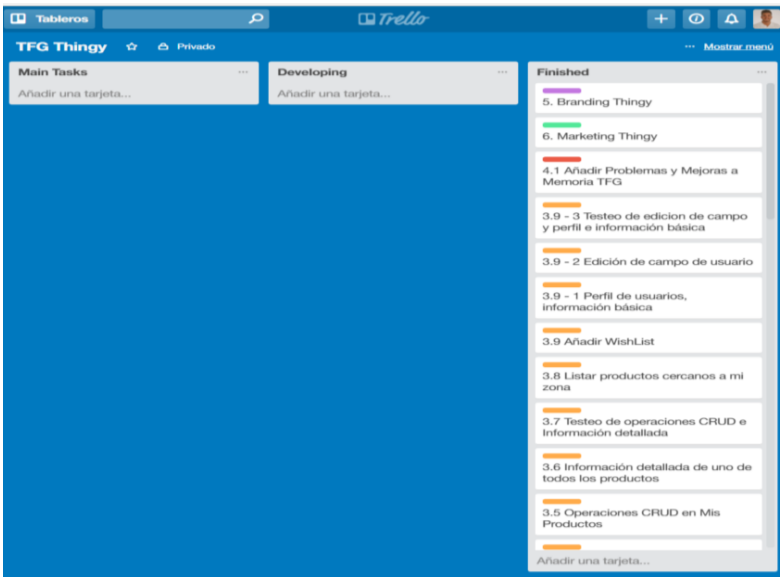


Figura 3 - Descripción tablero Planificación TFG

En él nos encontramos con una tarea principal que describe cada funcionalidad o en este caso cada sección del proyecto, dentro de cada una de estas tareas hay una serie de subtareas ordenadas por el color de la tarea principal. Además estas están ordenadas por prioridad según más importancia tenga hacia el cliente, tenemos un WorkInProgress (número máximo de tareas en cada categoría) en la que no trabajaremos más de 2 tareas y si alguna de ellas es más grande, haremos una subdivisión de ella. En la categoría “developing” limitado de forma indirecta (por sprint)

¿Qué ventaja nos proporciona esto?

Ritmo ágil de desarrollo de la aplicación, trabajando día a día, semana tras semana regulando la carga de trabajo evitando problemas que puedan surgir y retrasos de la finalización

8.2 ANÁLISIS FUNCIONAL

Una vez después de la planificación se trata de la primera fase de cualquier proyecto, empezaría con la reunión con el cliente, en este caso nuestra reunión

con la tutora. Tras lo investigado y consultado por ambas partes es el lugar donde especificamos de forma concreta y detallada qué tiene que hacer la aplicación y cómo tiene que hacerlo.

¿Qué nos permite dicho análisis?

- Evaluar viabilidad técnica y económica
- Minimizar desigualdades entre estética y funcionalidad
- Explicar de forma detallada y precisa los diferentes requisitos funcionales
- Analizar, controlar y supervisar el desarrollo funcional de la aplicación
- Diseñar diferentes metodologías con las que testear el buen funcionamiento de la aplicación
- Garantizar que el equipo de mantenimiento conozca la funcionalidad de la aplicación al finalizar el desarrollo.

Los requisitos funcionales son declaraciones de los servicios que proveerán dicho sistema, cómo reaccionará a las peticiones, además de lo que no debe hacer. Habitualmente este es un punto problemático para muchos desarrolladores ya que dan requisitos ambiguos y el cliente no desea esto. Si hay que pulir estos y desglosarlos en nuevos, realizando los cambios pertinentes al sistema y retrasando su entrega e incrementando el costo siempre y cuando esto sea meditado con el cliente.

Hemos desarrollado un diagrama de casos de uso para definir los requisitos funcionales presentes en nuestra aplicación de los cuales desarrollaremos por prioridades pero no todos puesto que esto desarrollaremos ciertas versiones.
Como podemos ver en el siguiente esquema.

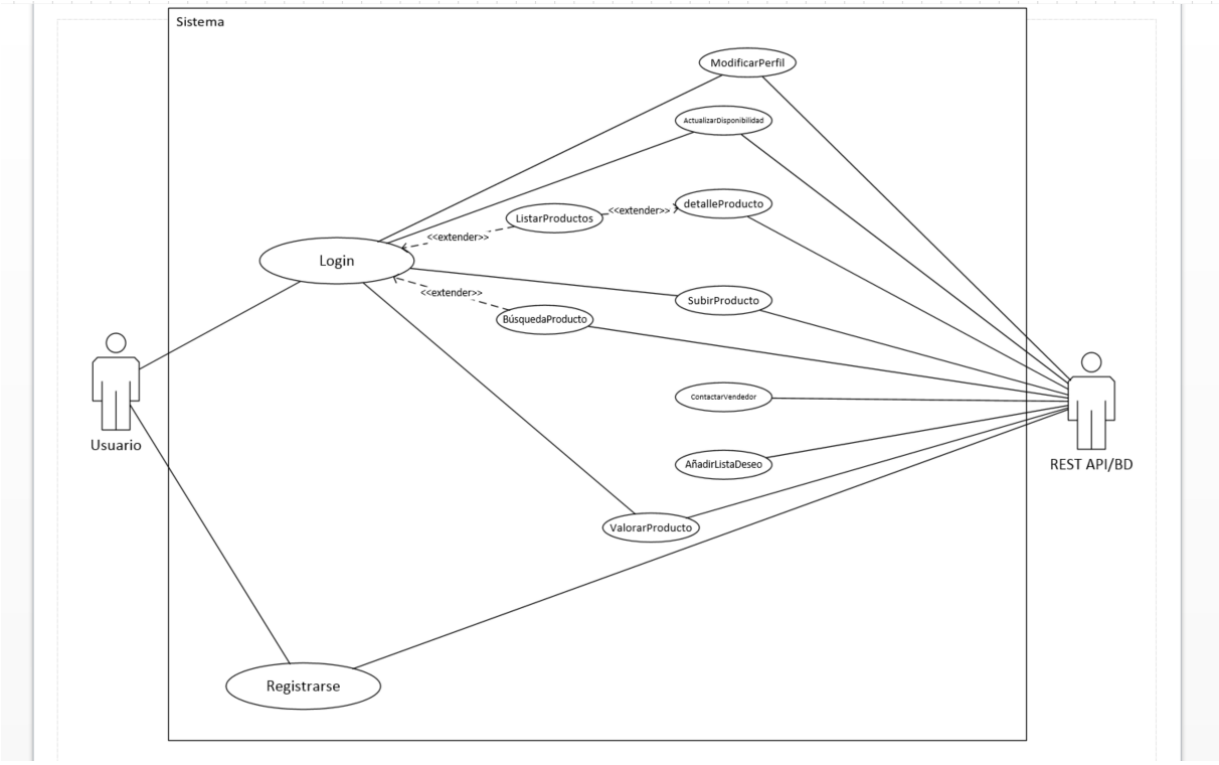


Figura 4 – Diagrama de Casos de Uso

En Thingy estos requisitos funcionales pueden dividirse en grupos por los casos de uso dispuestos en el anterior diagrama y desglosarse en los siguientes:

- El usuario que desee entrar en Thingy deberá estar registrado. Por tanto:
    - Sistema registrará información de los usuarios, por medio del respectivo formulario.
- Para mostrar las distintas funciones el usuario debe estar logueado. Por tanto:
    - Sistema debe permitir que los usuarios registrados puedan identificarse mediante sus credenciales
- Una vez logueados podrían llevar a cabo las funciones a las que da acceso dicha aplicación y que corresponderían el resto de requisitos funcionales:
    - El sistema debe permitir mostrar todos los productos existentes.
    - El sistema debe permitir mostrar los productos cercanos a la zona del usuario
    - El sistema de permitir mostrar la información detallada de un producto

- El sistema debe permitir la búsqueda de un producto por una serie de filtros
- El sistema debe permitir la subida de un producto
- El sistema debe permitir borrar un producto
- El sistema debe permitir editar un producto
- El sistema debe permitir añadir a la lista de deseos un producto
- El sistema debe permitir contactar al comprador con el vendedor a base de mensajes
- El sistema debe dar valoración del producto, comprador y vendedor
- El sistema debe permitir modificar los campos del perfil del usuario

Más tarde describiremos las funcionalidades y explicaremos con detalle aquellas que pertenecen a la personalización, el grueso del proyecto.

En cuanto a los requisitos funcionales son aquellos que no se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades emergentes de éste como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y la representación de datos que se utiliza en la interfaz del sistema. Estos surgen de la necesidad del usuario, políticas de organización, interoperabilidad con otros sistemas de software o hardware. Algunos de ellos serán los siguientes:

- El sistema se visualizará correctamente en navegadores Google Chrome y Mozilla Firefox
- El sistema debe cumplir las disposiciones recogidas en la Ley Orgánica de Datos Personales y en el Reglamento de medidas de seguridad
- El sistema no debe tardar más de cinco segundos en mostrar los resultados de una lista de productos si se supera dicho plazo, detendrá la búsqueda lanzando un mensaje al usuario

Estos son algunos de los requisitos aunque podremos ir actualizándolos conforme se desarrolle el software o evolucione dicha aplicación

8.3 METODOLOGÍA DE DESARROLLO SOFTWARE

Trabajaremos aunque no tenemos un grupo de desarrolladores sino que simplemente somos uno implementaremos un proceso Scrum dentro de las metodologías ágiles por lo que buscamos en un entorno donde se necesita obtener resultados lo más pronto posibles, existen requisitos cambiantes o poco definidos y debemos buscar innovación, flexibilidad y productividad para aumentar la competitividad con el resto de competidores. Por ello lo que haremos será ejecutar bloques temporales cortos y fijos (iteraciones que no durarán más de 2 semanas, para un feedback y reflexión por parte del integrante). Cada iteración proporcionará un resultado completo, es decir, un prototipo, que será un incremento del producto final a ser entregado al cliente si él lo solicita.

Dentro de la lista de tareas descritas en la planificación que ya han sido priorizadas con ayuda del cliente (la tutora) hemos priorizado los objetivos eligiendo aquellas tareas que aportan mayor valor al producto.

Cada día respondíamos a las tres preguntas:

- ¿Qué he hecho desde la última revisión?
- ¿Qué voy a hacer a partir de ahora?
- ¿Qué impedimentos tengo o puedo tener?

Realmente no llega a ser SCRUM ya que faltan mucho de los integrantes del equipo SCRUM pero a rasgos generales trabaja la misma filosofía, englosada dentro de la metodología ágil. Finalmente al acabar una iteración, su prototipo ha sido demostrado al cliente (tutor) junto a una restrospectiva, cómo ha sido su manera de trabajar y cuáles podrían ser los problemas que le impidiesen progresar adecuadamente, mejorando continuamente nuestra productividad.

Todo el trabajo que se llevará acabo estará almacenado en GitHub el sistema de Control de Versiones Git. Adjunto un enlace en la bibliografía además de la ilustración que muestra los archivos contenidos dentro del proyecto.

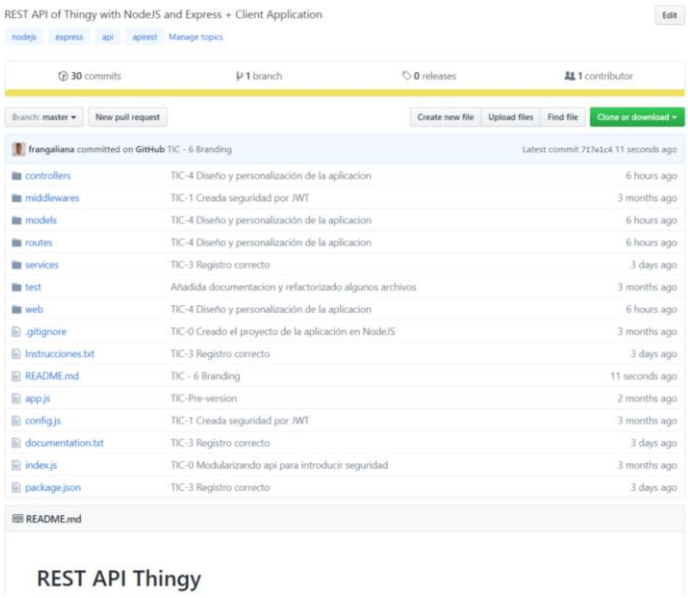


Figura 5 – Proyecto en Git

8.4 IMPLEMENTACIÓN

Como hemos descrito antes Thingy es una aplicación móvil que consta de 2 partes claramente diferenciables, Back-end y Front-end, para separar la API REST y así poder generar una interfaz en cualquier tipo de plataforma. La teoría derivaría del patrón MVC (Modelo-Vista-Controlador) aunque sólo en aspectos generales para ilustrar cada una de las partes y hacer más descriptiva la constitución de nuestra arquitectura aunque como hemos comentado e indagaremos más adelante cada una es independiente, pero sí entraremos ahora en contexto.

8.4.1 PATRÓN MVC

El MVC o Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Modelos y Controladores) separa la lógica de la aplicación de la lógica de la vista en una aplicación. Es una arquitectura importante puesto que se utiliza tanto en componentes gráficos básicos hasta sistemas empresariales; la mayoría de los frameworks modernos utilizan MVC (o alguna adaptación del MVC) para la arquitectura, entre ellos podemos mencionar a Ruby on Rails, AngularJS, nuestro proyecto con NodeJS y React además de muchos otros más. Para verlo mejor lo plasmaremos gráficamente con la siguiente ilustración.

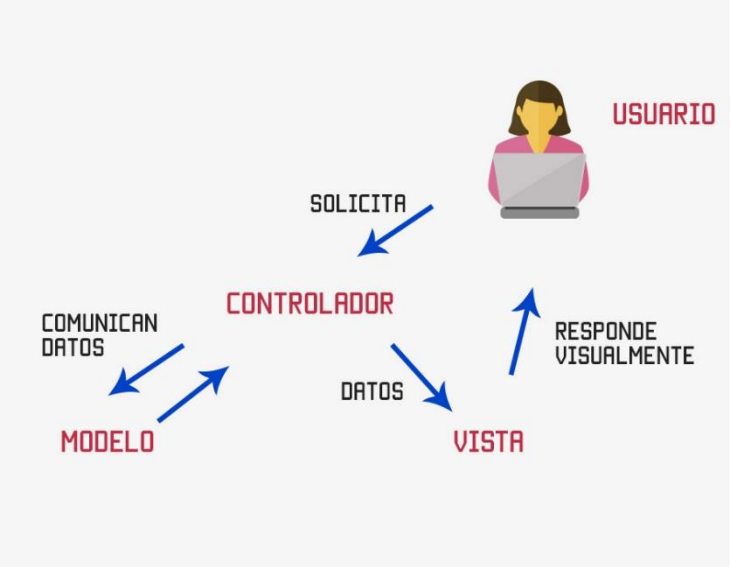


Figura 6 – Esquema modelo MVC por Código Facilito [15]

El modelo, se encarga de los datos, generalmente (pero no obligatoriamente) consultando la base de datos. Actualizaciones, consultas, búsquedas, etc. todo eso va aquí, en el modelo.

El controlador, se encarga de controlar, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.

La vista, son la representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica va aquí. Ni el modelo ni el controlador se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la vista.

8.4.2 ARQUITECTURA

Como hemos comentado tras el diseño utilizando el patrón MVC podemos ver 2 partes claramente diferenciables: Front-End y Back-End:

- Front-end, parte de la aplicación a la que tiene acceso el usuario, es decir, la parte visible del proyecto. Aquí definimos lo que se conoce como Arquitectura de la Información (AI). Ella se centra en la organización y estructuración del contenido de forma que sea usable para el usuario. Para ello tomaremos diferentes métodos y procesos basados en la Experiencia de usuario (UX) cómo por ejemplo, test de usabilidad, diagramas de flujo, etc. La repercusión de esta se verá reflejada en la tasa de engagment, reputación de la empresa, aumento de conversiones, etc. Lograr el mejor rendimiento aquí hará adquirir muchas estadísticas sobre la aplicación y mucha recomendación de los propios usuarios.
- Back-end, parte que permanece oculta al usuario y a la que tiene acceso el cliente y administrador. Se trabaja la parte lógica de una aplicación, es decir, que todo funcione correctamente.

La arquitectura de nuestra aplicación tendría la siguiente forma representada en esta imagen.

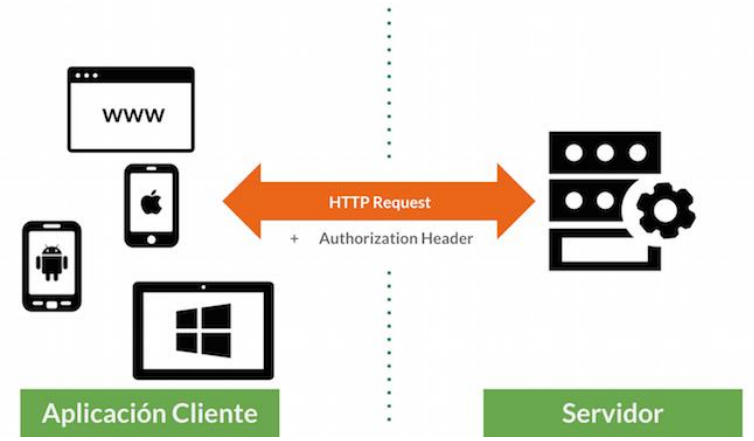


Figura 7 – Estructura 2 partes de nuestro proyecto [19]

El directorio de nuestro proyecto sería el siguiente.

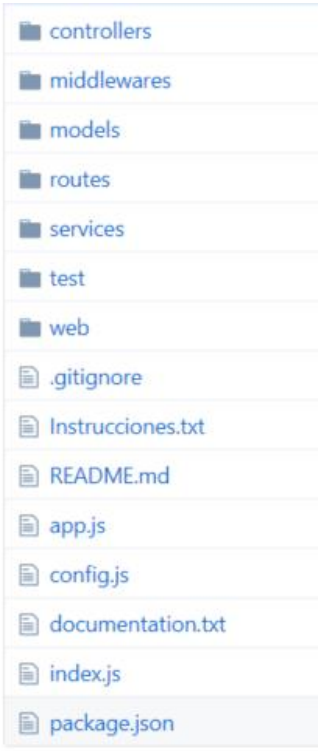


Figura 8 – Estructura de nuestro proyecto

Como podemos observar las 2 partes están presentes el Back-end sería todo el proyecto a excepción de la carpeta web, mientras que el Front-end sería justo dicha carpeta.

Al igual que con la "comparación" de MVC tendríamos la Vista englobada en la carpeta "Web". Por otra parte, el Controlador en "controllers" que se ayudarían de "middleware" para realizar sus operaciones y realizar las peticiones a través de "routes" y finalmente el Modelo en la carpeta "models".

Los proyectos de NodeJS tienen en común el package.json, está situado en la raíz del proyecto y es un manifiesto donde quedará reflejada la configuración del proyecto con información como la siguiente: nombre del proyecto, autor, versión, dependencias, scripts, repositorio git, motor de node, etc.

El resto lo creamos para utilizar el framework Express de una manera útil, ordenada y fluida. Además de la carpeta test que realiza pruebas sobre los componentes desarollados antes de la fase de producción.

¿Qué podemos encontrar en cada una de estas carpetas?

En "controllers", todos y cada uno de los controladores que definen cada una de las operaciones de los modelos, como hemos descrito de la API-REST.

En "middlewares", son funciones que se sitúan antes de las funciones que se llaman en los controladores y permiten la administración y comunicación de datos.

En "models", todos y cada uno de los modelos que representan las colecciones en la base de datos y que Mongoose los llama Schema (reducen código a la hora de realizar modificar y realizar consultas sobre los datos).

En "routes", se encuentra un archivo índice donde recogemos el listado de peticiones Http asociados con las funciones que trabajan o realizan estas y el middleware necesario.

En "services", encontramos 2 funciones auxiliares que tratan la seguridad (hablaremos de ellas posteriormente).

En "test", pruebas sobre la aplicación (hablaremos también posteriormente).

Dentro de "web", como hemos comentado, la carpeta de la aplicación cliente, encontramos 4 carpetas: "css", la cual tiene las hojas de estilo para la aplicación y la librería bootstrap; "fonts", las fuentes necesarias para el texto; "js" los componentes React que visualizarán los datos y el Bundle que los arrancará (hablaremos de ello posteriormente); "public" donde se guardan las imágenes de la aplicación y finalmente "index.html" que cargará todas las vistas y "users.html" la vista del perfil desarrollada de una forma distinta a la que mencionaremos para los componentes React. Por otro lado, el archivo *app.js* se encontrará la llamada a express e indicaremos que se usará "/api" para utilizar la api con la url y "/web" para utilizar el cliente a través de la carpeta "web".

Por otro lado, en *config.js* encontramos las variables para cambiar sin ir uno por uno el puerto de la aplicación, que está en el 3000, la dirección de la base de datos *'mongodb://localhost:27017/thingydb'* o el código secreto pasado al token. Para llegar a index.js que es donde conectaríamos Mongoose a la base

de datos, exportaríamos los archivos *app* y *config* anteriormente citados y realizamos la conexión escuchando en el puerto indicado en la variable. Ahora estará accesible la url: http://localhost:3000/

Para toda la edición añadiremos en package.json a la llamada npm start con la que utilizaremos la herramienta Nodemon [25] que se encarga de recompilar el código cada vez que se guarda.

8.4.3 PERSONALIZACIÓN

Esta es la parte a estudiar por dicho proyecto, en Thingy se ha querido poner atención a como la aplicación va a aprender del usuario para cada vez mostrar la información más filtrada para cada uno de estos usuarios y ¿cómo es capaz de realizar dicha acción?

Implementaremos una serie de funcionalidades que procesaran y almacenarán información para que con esta la visualización de la información o carga de datos sea más precisa para el usuario que se ha logueado. Pasaremos a definirlas arrojando información interesante sobre ellas.

Primeramente el uso de MongoDB es una de las herramientas como hemos comentado anteriormente, puesto que nos dejará almacenar volúmenes de información que pueden ser modificados posteriormente sin necesidad de realizar una reestructuración completa. Cuando evolucione la aplicación y crezca la información gracias a esta base de datos se procesarán y almacenarán datos rápidamente, además de que podremos ubicar información en distintos servidores como medida de seguridad o la mencionada escalabilidad de la aplicación. Unida a dicha base de datos, las funcionalidades desarrolladas serían las siguientes para las primeras versiones:

- Vista de información detallada del producto con los campos más relevantes, aquellos que dan una información completa al cliente además de saber en qué estado está el producto, entre ellos: imágenes que son un factor importante a la hora de vender o decantarse por un producto[16], la fecha de publicación del usuario con la que se muestra la antigüedad del anuncio, título que expresa

en pocas palabras y de forma clara y concisa qué producto se está vendiendo, una descripción realizada por el usuario para detallar de la forma más precisa posible el producto que está vendiendo que unido a la categoría hace un filtrado entre diferentes productos, contador de visitas que mostrarán al usuario si es un producto en el que la gente puede estar interesado lo que hará adelantarse a otros compradores, localización del producto, además de acceso a la página del vendedor y opciones rápidas de un solo click para guardar en la lista de deseos un producto interesante (se describirá a continuación) o contactar con el vendedor para realizar la compra.

- Añadir productos a la lista de deseos, con ello el usuario tendrá en un solo click aquellos productos que considera que merecen una revisión regular puesto que le pueden haber llamado la atención para futura compra. Con los productos que guarde en la lista de deseos (WishList) podremos saber para más tarde filtrar los datos en su tablón:
    - Zona en la que está interesada
    - Categoría de más interés en dicho momento y productos relacionados o similares
    - Precio por el que podría comprar según categoría

- Búsqueda de los productos cercanos a su zona. Gracias a permitir la ubicación cuando un usuario se registra se añade un filtro de búsqueda adicional al usuario, y este es la autobúsqueda a un click y conseguir tener a su alcance la información de todos aquellos productos con un vendedor dentro de su radio de localización. Un ejemplo útil sería para aquella persona que busca un producto pero no dispone de un medio como para llegar al vendedor por lo que obtendrá productos en la zona donde se encuentra.

- Subida de un anuncio en tan solo dos clicks. No solo es fácil comprar sino también vender ya que desde la sección "Mis productos" solo es necesario rellenar varios campos y el producto rápidamente aparecerá en el listado de todos los productos.
- Búsqueda de productos ya sea por nombre, categoría, fecha de publicación lo que hará filtrar aún más los resultados que arrojará.
- Contactar con el vendedor mediante mensajes, una forma rápida de quedar o recoger información para un envío y su posterior compra. Thingy conecta a dos personas para llevar a cabo una compra en cuestión de varios clicks.
- Valoración sobre el producto/vendedor/comprador. La tecnología hace posible un proceso de auditoría siempre abierto, las opiniones o valoraciones bien utilizadas son una herramienta que repercute a nivel de visibilidad y rentabilidad tanto para el vendedor como para la aplicación. Según el TripBarometer 2016 de TripAdvisor, el 80% de los usuarios lee entre 6 y 12 opiniones antes de reservar una habitación de hotel, puesto que podría aplicarse a este proceso de selección también [17]. Por ello, cuando se finaliza un matching y se realiza la compra existe la opción de valorar a la otra persona para que el resto de usuarios tengan una información de fiabilidad sobre ella.

8.4.4 IMPLEMENTACIÓN BACK-END

El Back-end es aquella capa que hemos mencionado anteriormente que permanece oculta al usuario pero a la que tiene acceso el cliente y administrador. Se trabaja la parte lógica de una aplicación, es decir, que todo funcione correctamente, el lado del servidor. En esta parte mediante el lenguaje de programación JavaScript, el entorno de ejecución Node.js y el framework Express, que nos facilitará mediante métodos de utilidad HTTP y middleware a su disposición, crearemos una REST API a la que cualquier interfaz mediante el uso de HTTP obtendrá datos y generará operaciones sobre esos datos.

Estos datos están almacenados usando persistencia con base de datos, esta es en este caso MongoDB. Como hemos comentado anteriormente además de las ventajas que hacen que nos decantemos por ella, se trata de un sistema de base de datos NoSQL orientado a documentos. En lugar de guardar los datos en tablas como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos en documentos similares a JSON (BSON) con un esquema dinámico.

Para conectar MongoDB con NodeJs utilizamos la librería llamada Mongoose [18] que utiliza el modelo de mapeo relacional en el que creando en el proyecto una serie de modelos que implementan cada una de las entidades descritas reduciendo la cantidad de código necesario para lograr la persistencia de los objetos. Trabaja en armonía con MVC, donde el modelo corresponderá a dicho objeto o colección de MongoDB. A continuación presentamos una "Tabla de datos relacionales guía" propuesta para en la siguiente ilustración para describir la estructura de datos de nuestra aplicación.

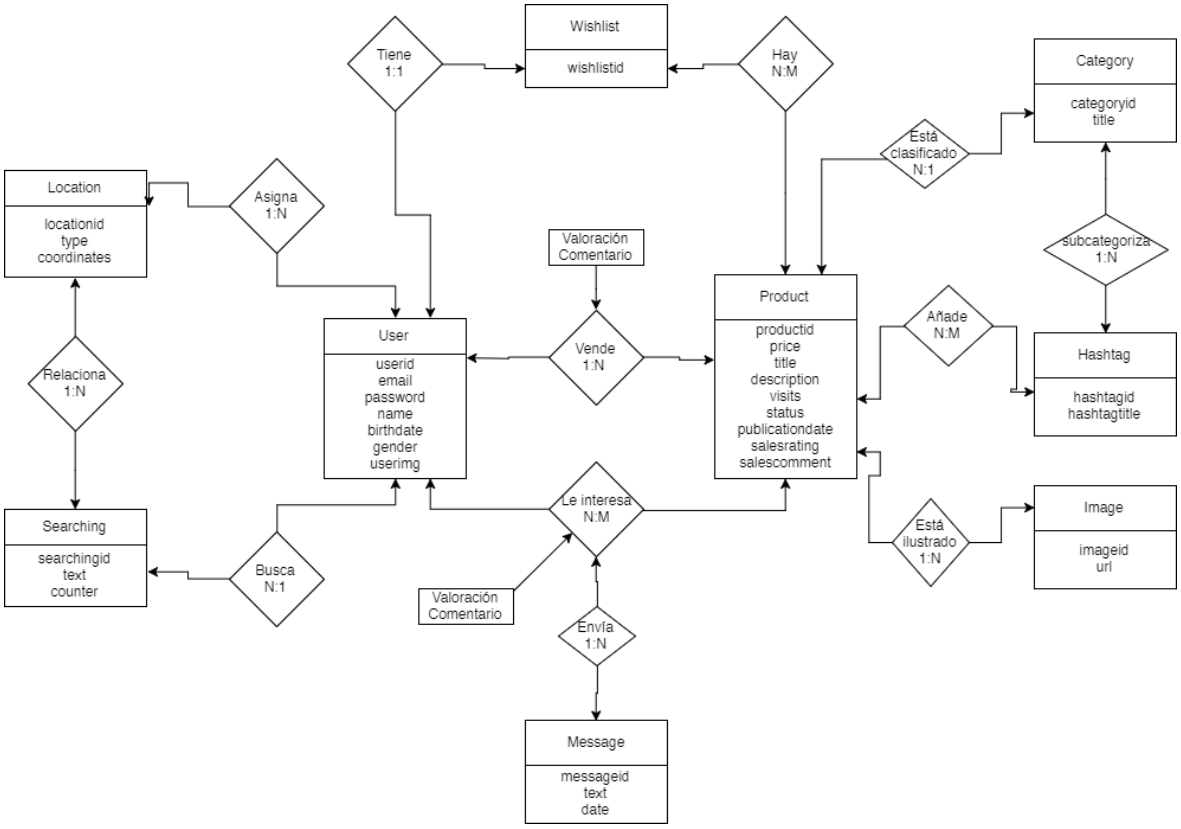


Figura 9 – Descripción del modelo entidad-relación

En el lado del controlador creamos las operaciones que tratan con los datos de cada modelo (Usuarios, Productos, etc…), son las siguientes para ilustrar el ejemplo en el lado del cliente:

- function signUp(req, res)
- function signIn(req, res)
- function getUsers(req, res)
- function getUser(req, res)
- function updateUser(req,res)
- function deleteUser(req,res)

Una función como las anteriormente citadas tendría la siguiente forma:

```
const mongoose = require('mongoose');
const User = require('../models/user');
const Location = require('../models/location')
const service = require('../services');
const bcrypt = require('bcrypt-nodejs');
const moment = require('moment')

function signUp(req, res) {

    let idLocation = "59a9a4011bd0197083c9e201";

    if(req.body.location != undefined){
      const location = new Location({
        type: req.body.location.type,
        coordinates: req.body.location.coordinates
      })

     location.save(function(err, result){
      idLocation = result.id;
     });
   }

      const user = new User({
        email: req.body.email,
        name: req.body.name,
        password: req.body.password,
        gender: req.body.gender,
        birthdate: moment(req.body.birthdate, 'DD/MM/YYYY'),
        location: idLocation
      });

      user.userimg = user.gravatar()

      user.save((err) => {
        if (err) res.status(500).send({ message: `Error al registrar el usuario ${err}` });

        return res.status(201).send({
          message: 'Te has registrado correctamente',
          token: service.createToken(user),
          user_id: user.id,
          links: {
            products: 'localhost:3000/api/products',
            self: 'localhost:3000/api/users/'+user.id,
            user_products: 'localhost:3000/api/users/'+user.id+'/products'
          }
        });
      });
}.
```

Figura 10 – Función de Registro en el controlador user.js

Añadimos en las rutas las peticiones a los recursos unido a las operaciones que muestran estos datos, y si lo deseamos podemos incorporar funciones middleware para interactuar o comunicarse con la función del controlador. En este caso utilizamos isAuth para comprobar que los usuarios están autentificados a la hora de acceder a dicho recurso. Mostramos una ilustración ejemplo de estos dos elementos.

```
//Rutas de users
api.post('/signup', userCtrl.signUp);
api.post('/signin', userCtrl.signIn);
api.get('/users', userCtrl.getUsers);
api.get('/users/:userId', userCtrl.getUser);
api.put('/users/:userId', auth, userCtrl.updateUser);
api.delete('/users/:userId', auth, userCtrl.deleteUser);

module.exports = api;
```

Figura 11 – Peticiones http en el directorio index.js de routes

```
'use strict';

const services = require('../services');

function isAuth(req, res, next) {
  if (!req.headers.authorization) {
    return res.status(403).send({ message: 'No tienes autorización' });
  }

  const token = req.headers.authorization.split(' ')[1];

  services.decodeToken(token)
    .then(response => {
      req.user = response;
      next();
    })
    .catch(response => {
      res.status(response.status);
    });
};

module.exports = isAuth;
```

Figura 12 – Función isAuth en el directorio middleware

8.4.5 SEGURIDAD

Algo que en muchas aplicaciones suele pasar inadvertido pero que es pilar para una buena aplicación es de lo que hablamos en este apartado: la seguridad de la aplicación.

Por una parte para que no entren visitantes maliciosos y también para que cada usuario tenga su información a salvo es necesaria la autenticación.

Para que dicha API REST pudiese ser escalable para cualquier tipo de Front-End, como una aplicación web o aplicación móvil, ya que el servidor se encuentra separado del Front-End como hemos descrito anteriormente, la mejor opción y por la que nos hemos decantado es por una Autenticación basada en Token.

El usuario autentica nuestra aplicación enviando al servidor un código cifrado (Token), bien sea con un par usuario/contraseña (como es nuestro caso), o a través de un proveedor como puede ser Twitter, Facebook, Google, etc. El servidor lo descifra, observa si el usuario está registrado y las opciones que

tiene permitidas. A partir de entonces, cada petición HTTP que realiza el usuario va acompañada de este Token en la cabecera.

Este Token se trata de una firma cifrada que permite a nuestra API identificar al usuario. Dicho Token no es almacenado en el servidor sino en el lado del cliente (como en nuestra aplicación en localStorage, explicaremos más tarde) y el API se encarga de descifrar ese Token y redirigir el flujo de la aplicación en un sentido u otro. Al estar aquí es por ello lo que hemos comentado sobre permitir escalar a la API REST. Entonces si están en el lado del cliente, ¿cómo que el servidor (API REST) trabaja con ellos autorizando a realizar operaciones a ciertos usuarios sobre los datos y a otros no autorizándolos?

Utilizamos el estándar de este tipo de autenticación que se trata de utilizar JSON Web Tokens (JWT). Al igual que el API trabajamos con JSON, como puedes ver estamos uniformizando la base lo más posible para dotar de robustez, velocidad y garantías, ya que este formato es agnóstico y podemos utilizar el que queramos, en nuestro caso Node.Js.

¿De qué se compone JWT?

Está compuesto por 3 strings separados por un punto conforme la ilustración siguiente:

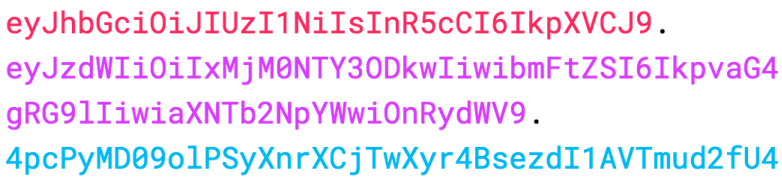


Figura 13 – Ejemplo de JWT [16]

El 1er string (color rojo) sería el Header, cabecera del token que a su vez tiene dos partes, el tipo, que en este caso es JWT y la codificación utilizada HMAC SHA256.

El 2ndo string (color lila) sería el Payload, compuesto con JWT Claims donde se colocan los atributos que definen nuestro token. Existen varios aunque nosotros hemos colocado: “sub”, identifica el sujeto del token con su id de su modelo, “iat”, identifica la fecha de creación del token, válido para ponerle

fecha de caducidad (en formato de tiempo Unix) y "exp", que identifica a la fecha de expiración del token. Podemos calcularla a partir del iat. En formato de tiempo UNIX.

Finalmente el string de color azul que está formada por los anteriores componentes cifrados en Base64 con clave secreta (la almacenamos en nuestro backend, justo en nuestro archivo de configuración por si se quiere cambiar posteriormente). Así sirve de Hash para comprobar que todo está correcto.

Una vez explicado correctamente el estándar de autentificación utilizado crearemos dos funciones que estarán alojadas en la carpeta "services" donde hemos comentado que teníamos las funciones que nos ayudarían a realizar operaciones, las podemos ver en la siguiente ilustración.

```
'use strict';

const jwt = require('jwt-simple');
const moment = require('moment');
const config = require('../config');

function createToken(user) {
  const payload = {
    sub: user._id,
    iat: moment().unix(),
    exp: moment().add(1, 'hour').unix(), //Antes 14, 'days'
  };

  return jwt.encode(payload, config.SECRET_TOKEN);
};

function decodeToken(token) {
  const decoded = new Promise((resolve, reject) => {
    try {
      const payload = jwt.decode(token, config.SECRET_TOKEN);

      if (payload.exp <= moment().unix()) {
        reject({
          status: 401,
          message: 'El token ha expirado',
        });
      };

      resolve(payload.sub);
    } catch (err) {
      reject({
        status: 500,
        message: 'Invalid token',
      });
    };
  });

  return decoded;
};

module.exports = {
  createToken,
  decodeToken,
};
```

Figura 14 – Funciones createToken y decodeToken

CreateToken es la función que estará dentro del controlador de User y en la función que realiza el registro, lo que hará será crear el JWT para el usuario. Mientras que decodeToken se utiliza en isAuth, que reside en la carpeta middleware y se muestra en la figura 11, para decodificar el JWT que le pasa el cliente y comprobar si es un Token inválido o ha expirado.

Por otro lado, esta aplicación dispone de otra forma de autenticación más sencilla denominada Autenticación Basic, esta es un método diseñado para permitir a un navegador web u otro programa cliente, proveer dichas credenciales del usuario en forma de usuario y contraseña al solicitar una página al servidor y está codificada en Base 64. Puede apreciarse en la siguiente figura en el método *signIn*. Si realizamos una petición con terminación: *signin?type=basic*

```
unction signIn(req, res) {

 var email = req.body.email;
 var password = req.body.password;

 //Cuando hagamos signin?type=basic
 if(req.query.type) {
   if(req.query.type === 'basic') {
     //Obtiene el login y la contraseña
      var auth = new Buffer(req.get('Authorization').substring(6), 'base64').toString('ascii')
      var split = auth.split(":")
      email = split[0]
      password = split[1]
      console.log('Peticion de login -> ' + email + ':' + password)
   }
 }
      if (email && password) {
         User.findOne({ email: email}, (err, user) => {  //en vez de login: req.body.email
           if (err) return res.status(500).send({ message: err });
           if (!user) return res.status(404).send({ message: 'Usuario no encontrado' });

           bcrypt.compare(password, user.password, function(err, comparePassword) { //en vez de password: req.body.password
             if (err) return res.status(500).send({ message: err });
             if (comparePassword == false) return res.status(403).send({ message: 'Datos incorrectos' });

             res.status(200).send({
                 message: 'Te has logueado correctamente',
                 user_id: user.id,
                 token: service.createToken(user),
                 links: {
                   products: 'localhost:3000/api/products',
                   self: 'localhost:3000/api/users/'+user.id,
                   user_products: 'localhost:3000/api/users/'+user.id+'/products'
                 }
             });
           });
         });
     } else {
       res.status(400);
                  //res.header('WWW-Authenticate', 'Basic realm="ADI"')
                  res.send('Login incorrecto');
                  console.log('Login incorrecto');
     }
;
```

Figura 15 – Método signIn de controlador user.js

Un aspecto más de seguridad a apuntar que hemos desarrollado es la codificación de la contraseña a la hora de almacenarla en la base de datos, puesto que sin codificación ante la sustracción indebida de información de la base de datos quedarían las contraseñas en limpio. Utilizaremos la funcionalidad de Mongoose para ejecutar una función antes de que el modelo se guarde en la base de datos denominada *pre* donde ejecutaremos la siguiente función para encriptar la contraseña que introduzca el usuario. Se trata de un callback con parámetro *next* para que pueda pasar al siguiente middleware, teniendo el usuario comprobaremos si no ha modificado su contraseña, si es que sí utilizaremos Bcrypt [21], se trata de una función que hará que hasheará la contraseña con el *salt* generado, que se trata de un fragmento aleatorio que se usará para generar el hash. Todo ello controlando errores, si no ha habido errores la contraseña será entonces el hash creado como podemos ver en la siguiente ilustración.

```
UserSchema.pre('save', function(next) {
  let user = this;
  if (!user.isModified('password')) return next();

  bcrypt.genSalt(10, (err, salt) => {
    if (err) return next(err);

    bcrypt.hash(user.password, salt, null, (err, hash) => {
      if (err) return next(err);

      user.password = hash;
      next();
    });
  });
});
```

Figura 16 – Método pre-guardado en el modelo User.js

Así se evita que dos contraseñas tengan el mismo hash y los problemas que conlleva, por ejemplo, ataques a fuerza bruta o Rainbow table, que son asociaciones entre textos y su hash asociado, para evitar su cálculo y acelerar la búsqueda de la password. Con el salt, se añade un grado de complejidad que evita que el hash asociado a una password sea único.

Por otro lado, para la autenticación necesitaríamos comprobar que la password introducida sea correcta. Al contrario que en otros sistemas aquí no desencriptamos la password, sino que la comparamos con la introducida por el usuario con la *función compare* como podemos ver en la Figura 22.

Volviendo a la implementación de los casos de uso aunque hemos descrito muchas de las formas en las que trabajamos indicaremos a continuación los casos de uso desarrollados y los describiremos. A la hora de describir un anuncio en el proyecto lo hemos denominado producto para que se entienda la relación

¿Qué casos de uso hemos realizado?

- Obtener un producto específico

```
'use strict';

//Colocar .. para salir de Controller e ir a Models
const Product = require('../models/product');
const User = require('../models/user');
const Location = require('../models/location');


//Aquel al que se le pase el middleware antes tendrá req.user con el id del user logueado
function getProduct(req, res) {
  let productId = req.params.productId;

  Product.findById(productId, (err, product) => {
    User.populate(product, {path: "user", select: '-password'}, function(err, product) {
      res.setHeader('Content-Type', 'application/json');
      if (err) return res.status(500).send({ message: `Error al realizar la petición ${err}` });
      if (!product) return res.status(404).send({ message: `No existe el producto con el id = ${productId}` });

      //Al mandar un objeto del mismo nombre clave-valor, se puede reducir así product:product
      var result = {
        product: product,
        links: {
          self: 'localhost:3000/api/products/'+product.id,
          user: 'localhost:3000/api/users/'+product.user.id
        }
      }
        res.status(200).send(result);
      });
  });
}
```

Figura 17 – Método que implementa obtención de un producto específico

Como podemos observar recuperamos el id del producto para realizar una búsqueda en la base de datos utilizando Mongoose con la función *findById (productId, function)* donde el id sea el de dicho producto, usamos la función *populate* para según el "path" seleccionado y evitamos en la búsqueda que muestre la contraseña. Tras realizar la

búsqueda con el resultado enviaremos una respuesta con su estado y cuerpo del mensaje.

- Obtener todos los productos

```
anction getProducts(req, res) {
var limit;

if(req.query.limit) {
  limit = parseInt(req.query.limit)
  if(isNaN(limit)){
    return next(new Error())
  }
} else {
  limit = 5;
}

var query = {};
//Para obtener la página anterior a un id
if (req.query.before) {
  query = {"_id" : {$lt: req.query.before}};
//Para obtener la página posterior a un id
} else if (req.query.after) {
  query = {"_id": {$gt: req.query.after}};
}

Product.find(query)

  .limit(limit)
  .exec((err, products) => {
    User.populate(products, {path: "user", select: '-password'}, function(err, products) {
      res.setHeader('Content-Type', 'application/json');
      if (err) return res.status(500).send({ message: `Error al realizar la petición ${err}` });
      if(products.length > 0){
        if(req.query.before){
          products.reverse();
        }
        var result = {
              data: products,
              paging: {
                cursors: {
                  before: products[0].id,
                  after: products[products.length-1].id
                },
                previous: 'localhost:3000/api/products?before='+products[0].id,
                next: 'localhost:3000/api/products?after='+products[products.length-1].id,
              },
              links: {
                self: 'localhost:3000/api/products',
                users: 'localhost:3000/api/users'
              }
            }
        } else {
            var result = {
                  data: products,
                  paging: {
                  cursors: {
                    before:undefined,
                    after:undefined
                    },
                    previous: undefined,
                    next: undefined
                  },
                  links: {
                    self: 'localhost:3000/api/products',
                    users: 'localhost:3000/api/users'
                  }
                }
      `
```

Figura 18 – Método que implementa la obtención de todos los productos

En este método estableceremos una variable que ejercerá de límite el cual trata de dar el resultado total agrupado en porciones con un número máximo de elementos establecido por el límite, el resto pasará a la siguiente página por lo que implementaremos paginación por medio de cursores que serán los definidos en el resultado como "before" and "after".

Para ello crearemos distintas consultas según se indique en la url una dirección u otra. Finalmente mediante esta vez *find (query)* y añadiendo las funciones para filtrar de *limit (limit)* y *exec (function)* para poder arrojar los datos y según el tamaño de estos arrojar un resultado u otro.

- Obtener productos cercanos a una localización

Antes de mostrar la función que ejecuta dicho proceso pasaremos a explicar cómo trabajamos con la localización. En la colección *locations* en MongoDB almacenamos todas las localizaciones hasta el momento registradas. Cada documento en la colección tiene la siguiente estructura:

```
'use strict';

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const LocationSchema = new Schema({
  type: {
    type: String,
    default: "Point"
  },
  coordinates: {
    type: [Number],
    index: "2dsphere",
    default: [38.280153, -0.712901]
  }
})

module.exports = mongoose.model('Location', LocationSchema);
```

Figura 19 – Modelo que muestra la estructura de locations

Guardamos la localización como un documento GeoJSON, es un estándar abierto diseñado para representar elementos geográficos sencillos, junto con sus atributos no espaciales. Tiene la forma descrita arriba compuesta por el tipo y las coordenadas que almacenan la latitud y altitud con un índice denominado 2dsphere para representar puntos situados en esfera.

```
function getNearbyProducts(req, res) {
  let resultFinal = [];
  var limit;
  let userId = req.user;

  User.findById(userId, {password:0}, (err, user) => {
    Location.populate(user, {path: "location"}, function(err, result){
      Location.geoNear(
        {type:'Point', coordinates: [parseFloat(result.location.coordinates[0]),parseFloat(result.location.coordinates[1])]},
        {maxDistance:100000, spherical: true}
      ).then(function(locations){

        for(var i = locations.length - 1; i >= 0; i--){
          var locationId = locations[i].obj.id;

          User.find({"location": locationId})
           .exec((err, result) =>{
            if(result.length > 0){

              for(var j = result.length - 1; j >= 0; j--){
                //Control para no sacar resultados de tu propio id
                 var userId = result[j].id;

                Product.find(userId, (err,products) =>{
                  if(products.length > 0){

                    resultFinal = {
                          data: products,
                          paging: {
                            cursors: {
                              before: products[0].id,
                              after: products[products.length-1].id
                            },
                            previous: 'localhost:3000/api/products?before='+products[0].id,
                            next: 'localhost:3000/api/products?after='+products[products.length-1].id,
                          },
                          links: {
                            self: 'localhost:3000/api/products',
                            users: 'localhost:3000/api/users'
                          }
                        }
                  } else {
                    resultFinal = {
                          data: products,
                          paging: {
                          cursors: {
                            before:undefined,
                            after:undefined
                            },
                            previous: undefined,
                            next: undefined
                          },
                          links: {
                            self: 'localhost:3000/api/products',
                            users: 'localhost:3000/api/users'
                          }
                        }
                  }
                  res.setHeader('Content-Type', 'application/json');
                  res.status(200).send(resultFinal);

                 })
               }
             }
           })
         }
       })
     })
   })
}
```

Figura 20 – Método que implementa la obtención de productos cercanos

Quizá el más complejo de todos, como hemos citado anteriormente trabaja también con paginación por cursores. Recoge el id del usuario y realizará una búsqueda *findById (userId, function)*, donde la función interior utiliza *populate* para poder mostrar las localizaciones de cada usuario con el parámero path de "location". Ahora utiliza una función especial de Mongoose denominada *geoNear ()* donde pasamos la localización a buscar cercanas como primer parámetro, la distancia máxima unido a un atributo denominado "spherical" a true, importante si hablamos de una coordenada localizada en una esfera. Añadimos una promesa que recogerá las localizaciones. Con el array de localizaciones utilizaremos un bucle for para recoger el id y hacer una búsqueda de usuarios con la condición de que exista el id recogido

utilizando *find ()*. Ahora utilizaremos la misma ejecución pero para de los usuarios que ha encontrado buscar los productos que tienen cada uno y así mostrar los resultados que devuelve teniendo en cuenta que no puede estar presente el del propio usuario.

- Obtener productos de un usuario

```
function getProductsUser(req, res) {
  var userId = req.params.userId;

  var limit;
  if(req.query.limit) {
    limit = parseInt(req.query.limit)
    if(isNaN(limit)){
      return next(new Error())
    }
  } else {
    limit = 5;
  }

  var query = {"user": userId};
  //Para obtener la página anterior a un id
  if (req.query.before) {
    query = {"_id" : {$lt: req.query.before}};
  //Para obtener la página posterior a un id
  } else if (req.query.after) {
    query = {"_id": {$gt: req.query.after}};
  }

  //La llamada populate hará que en la ruta "user" lo popule con los datos del modelo Autor, quedando una respuest
  //Si dentro de populate ponemos: select: '...' indicaremos aquellos que queremos mostrar
  Product.find(query)
    .limit(limit)
    .exec((err, products) =>{
        res.setHeader('Content-Type', 'application/json');
        if (err) return res.status(500).send({ message: `Error al realizar la petición ${err}` });
          if(products.length > 0){
            if(req.query.before){
              products.reverse();
            }
            var result = {
                  data: products,
                  paging: {
                    cursors: {
                      before: products[0].id,
                      after: products[products.length-1].id
                    },
                    previous: 'localhost:3000/api/users/'+userId+'/products?before='+products[0].id,
                    next: 'localhost:3000/api/users/'+userId+'/products?after='+products[products.length-1].id,
                  },
                  links: {
                    self: 'localhost:3000/api/users/'+userId+'/products',
                    user: 'localhost:3000/api/users/'+userId,
                    users: 'localhost:3000/api/users'
                  }
                }
            } else {
                var result = {
                    data: products,
                    paging: {
                    cursors: {
                      before:undefined,
                      after:undefined
                      },
                      previous: undefined,
                      next: undefined
                    },
                    links: {
                      self: 'localhost:3000/api/products',
                      user: 'localhost:3000/api/'+userId,
                      users: 'localhost:3000/api/users'
                    }
                  }
            }

          res.status(200).send(result);
```

Figura 21 – Método que implementa la obtención de productos de usuario

En dicho método ejecutamos lo anteriormente descrito tanto para paginación como utilizando el método *find ()* y sus funciones asociadas. Esto arrojará el resultado pertinente.

- Crear/Editar/Borrar un producto

```
function saveProduct(req, res) {
  console.log('POST /api/products');
  console.log(req.body);

  let product = new Product();
  product.title = req.body.title;
  product.price = req.body.price;
  product.user = req.user;
  product.categoryproduct = req.body.categoryproduct;
  product.description = req.body.description;
  product.visits = req.body.visits;
  product.status = req.body.status;
  product.publicationdate = req.body.publicationdate;
  product.salesrating = req.body.salesrating;
  product.salescomment = req.body.salescomment;

  product.save((err, productStored) => {
    if (err) { res.status(500).send({ message: `Error al guardar en base de datos: ${err}` })
    } else {
      res.setHeader('Content-Type', 'application/json');
      res.header('Location','http://localhost:3000/api/products/'+product.id)

      var result = {
        product: productStored,
        links: {
          self: 'localhost:3000/api/products/'+product.id,
          user_products: 'localhost:3000/api/users/'+product.user+'/products'
        }
      }
      res.status(201).send(result);
    }

  });
}
```

Figura 22 – Método que guarda un producto

En dicho método definimos una instancia de Producto que más tarde ejecutamos la función *save (function)*, esta función guardará el producto y con la función alojada en su interior arrojaremos la respuesta como siempre definida por su cabecera y cuerpo en el que devolveremos el producto creado.

```
function updateProduct(req, res)  {
  let productId = req.params.productId;
  let update = req.body;

  Product.findByIdAndUpdate(productId, update, (err, productUpdated) => {
    res.setHeader('Content-Type', 'application/json');

    if(productUpdated.user == req.user){
      if (err) res.status(500).send({ message: `Error al actualizar el producto: ${err}` });
      else {
        var result = {
          product: productUpdated,
          links: {
            self: 'localhost:3000/api/products/'+productUpdated.id,
            user_products: 'localhost:3000/api/users/'+productUpdated.user+'/products'
          }
        }
        res.status(200).send(result);
      }
    } else {
      res.status(400).send({ message: 'Actualización no autorizada: producto de otro usuario'})
    }
  });
}
```

Figura 23 – Método que actualiza un producto

En esta función pasamos los parámetros del usuario y con el id del usuario utilizamos la función *findByIdAndUpdate (productid, update,*

función) después devolveremos el resultado como en los métodos anteriores.

```
function deleteProduct(req, res) {
  let productId = req.params.productId;

  Product.findById(productId, (err, product) => {
    if(product.user == req.user){

      if (err) res.status(500).send({ message: `Error al borrar el producto: ${err}` });

      product.remove(err => {
        if (err) res.status(500).send({ message: `Error al buscar el producto: ${err}` });
        else {
          var result = {
            message: `Producto eliminado correctamente, id: ${productId}`,
            links: {
              user_product: 'localhost:3000/api/users/'+product.user+'/products',
              products: 'localhost:3000/api/products'
            }
          }
          res.status(200).send(result);
        }
      });
    } else {
      res.status(400).send({ message: 'Borrado no autorizado: producto de otro usuario'})
    }
  });
}
```

Figura 24 – Método que borra un producto

Mediante la id del producto que es pasada por la url utilizamos *findById* para encontrar el usuario alojado en la base de datos y ejecutar *remove (function)* sobre el producto con su consiguiente respuesta.

El resto de métodos se implementan de la misma forma pero hemos querido hacer hincapié en aquellos métodos que conforman la personalización. Puesto que Añadir/Borrar/Listar favoritos es igual que los anteriores pero utilizando el modelo de WishedProducts.

- Obtener un usuario
- Obtener todos los usuarios
- Editar/Eliminar un usuario
- Login/Registro de usuario (utilizando autentificación JWT o Basic, según se realice la petición)
- Añadir/Eliminar producto de la lista de deseos

Como hemos descrito, dichas peticiones realizarían respuestas http con su cabecera y el cuerpo que estaría en JSON como podemos ver en la siguiente fotografía.

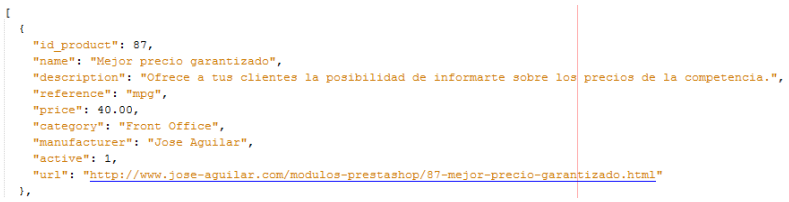


Figura 25 – Cuerpo de la petición http en JSON

8.4.6 REALIZACIÓN DE PRUEBAS

Además una vez realizada una funcionalidad esta antes de pasar en planificación a "Finished" era testeada para su correcto funcionamiento. De dicho bug se generaba en Git una rama paralela al proyecto para el arreglo de dicha funcionalidad para luego incorporarla dentro del proyecto. El testing se ha compuesto de pruebas de caja blanca y posterior debugging de la API REST se ha realizado utilizando Mocha [22], trata de un framework para NodeJS donde los tests puede ser ejecutados desde el terminal o el navegador. Este tiene la misma finalidad que JUnit en Java, la diferencia es que en mocha realizamos tests de código asíncrono de forma bastante sencilla. Aquí se adjunta un ejemplo del desarrollo.

```
describe('API REST Test:', function(){

  //Test POST /api/signup
  it(' Registro realizado de forma correcta', function(done){
      //Al objeto supertest le pasamos la app de Express
     server
          //Hacemos una petición HTTP
          .post('/api/signup').send({ email: 'fjgc9@alu.ua.es', name: 'Francisco Javier', password: '1234'})
          .expect(201)
          .end(function(err, res) {
            assert.equal('Te has registrado correctamente',res.body.message);
            done();
          });
  });

  //Test POST /api/signin?type=basic
  it(' Login con datos incorrectos, devuelve estado 404', function(done) {
    var auth = new Buffer('fjgc9@alu.ua.es:12345').toString('base64');
    //Al objeto supertest le pasamos la app de Express
    server
      //Hacemos una petición HTTP
      .post('/api/signin?type=basic').set('Authorization', 'Basic ' + auth)
      .expect(403)
      .end(done);
  });

  //Test POST /api/signin?type=basic
  it(' Login con datos correctos, mediante Basic', function(done) {
    var auth = new Buffer('fjgc9@alu.ua.es:1234').toString('base64');
    //Al objeto supertest le pasamos la app de Express
      server
        //Hacemos una petición HTTP
        .post('/api/signin?type=basic').set('Authorization', 'Basic ' + auth)
        .expect(200)
        .end(function(err, res) {
            assert(res.body.token);
            access_token = res.body.token
            done();
        })
  });
```

Figura 26 – Tres ejemplos de testing con Mocha

Como hemos comentado son similares a JUnit, le pasamos unos datos de prueba y realizamos la petición, expresamos la salida esperada y añadimos una función con un *assert.equal* el cual comprueba si se ha llevado a cabo correctamente y viendo en este caso en la consola que se han pasado. Por otro lado, las pruebas realizadas con supertest proveen un alto nivel de abstracción para probar servicios http. Utilizando por debajo superagent como cliente http. Este lo que proporciona son aserciones pudiendo verificar igual el código de estado de la respuesta, el body, cabecera string, etc. Para los componentes de React es utilizado Jest que es utilizado por Facebook para testear el código JavaScript incluyendo aplicaciones React. Tiene una ventaja y es que sus tests corren en paralelo de forma distribuida por lo que maximiza el rendimiento. Es capaz de realizar mocks.

Además se han realizado pruebas emergentes para verificar el correcto funcionamiento con pruebas de carga pero no se puede validar directamente

desde internet al no estar alojada allí por lo que se ha utilizado un Add-On de Mozilla Firefox como es Html Validator.

8.4.7 IMPLEMENTACIÓN FRONT-END

Ahora desarrollamos la parte visible del proyecto, en ella se centra la organización y estructuración del contenido (recursos devueltos o enviados al servidor) de forma que sea usable para el usuario, se trata de la parte Front-end.

Esta servirá de cliente al API REST. En ella utilizamos React JS [14], se trata de una biblioteca JavaScript de código abierto que crea interfaces con el objetivo de animar al desarrollo de aplicaciones Single Page Application. Este mantiene un virtual DOM propio, en lugar de confiar solamente en el DOM del navegador. Esto deja a la biblioteca determinar qué partes del DOM han cambiado comparando contenidos entre la versión nueva y la almacenada en el virtual DOM, y utilizando el resultado para determinar cómo actualizar eficientemente el DOM del navegador ganando en rendimiento. Incluimos las librerías de Bootstrap y jQuery, además de trabajar con Ajax para realizar las peticiones desde el servidor. Esto nos muestra las siguientes ilustraciones que vienen a continuación.

La estructura de la parte del Front-end vendría a ser de la siguiente forma.

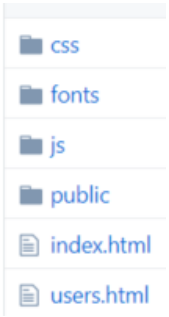


Figura 27 – Estructura del Front-End de Thingy

Para que pueda funcionar la parte de Front-End debíamos hacer que además de responder a la API, sirviese archivos estáticos (HTML/CSS/JS). Por ello creamos una carpeta web en el proyecto y colocamos dichos archivos, añadiendo un *index.html* de prueba.

En el servidor añadiremos la función de Express llamada *express.static* para que se sirva los archivos de la carpeta "web" anteriormente citada.

Para acceder a la web como hemos dicho al principio sería con el siguiente enlace: http://localhost:3000/web

Antes de nada… ¿Qué es DOM (Document Object Model)?

Se trata de una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML, XHTM y XML, un modelo sobre cómo pueden combinarse estos objetos y una interfaz estándar para acceder a ellos y manipularlos. [23]

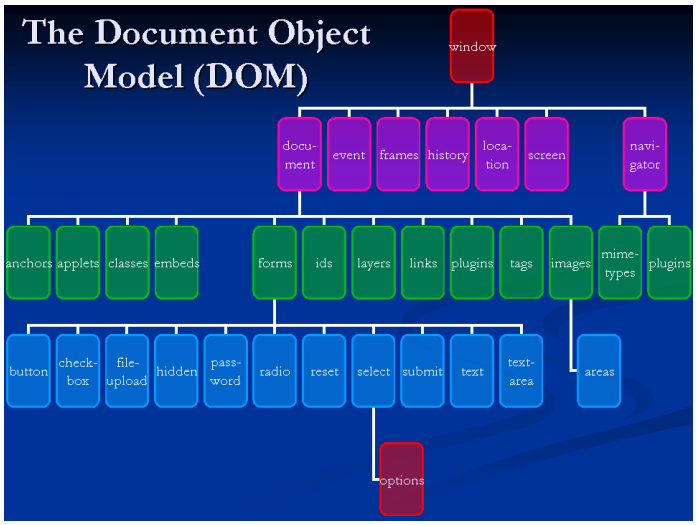


Figura 28 – Modelo DOM [24]

En el apartado de *Tecnologías y herramientas utilizadas* explicamos porque usamos React pero no describimos, además de la gran ventaja de que tenga un DOM Virtual para la actualización de datos antes del DOM Original, como vamos a trabajar. Partimos desde *index.html*, en él uniremos todo nuestro código JavaScript, las vistas que mediante las llamadas a la API nos devolverán los datos y nosotros nos encargaremos de representarlos.

Si hay una característica que mejor puede definir lo que representa ReactJS es el concepto SPA o Simple Page Application. ¿Qué es una SPA?
Una SPA es una aplicación web que se implementa sobre una única página web. A diferencia de otros frameworks, no navega de una página a otra para cambiar la estructura de la página. Ejecuta funciones que toman las actualizaciones de estado de la página y que se traducen en una representación virtual de la página resultante de haber añadido, modificado o borrado el cambio. Siempre que es informado de un cambio de estado vuelve a ejecutar esas funciones para determinar una nueva representación virtual de la página y a continuación se traduce automáticamente en los cambios del DOM necesarios para reflejar esta nueva página. Es rápido porque utiliza un DOM Virtual que hace selectivamente subárboles de los nodos sobre la base de cambios de estado, desarrollándolo con la menor cantidad posible de manipulación DOM.

Por ello, mejora significativamente la experiencia de usuario.

Para construir esto vamos a generar unos componentes en lenguaje JSX [26], se trata de un pseudolenguaje que facilita dicho desarrollo, estos estarán escritos en este lenguaje por medio de etiquetas y sintaxis muy parecida a la de HTML pero que se compila a JavaScript. Al no poder utilizar directamente este lenguaje debemos usar un precompilador basado en NodeJS que nos genera JavaScript nativo y evita que el JSX tenga que ser compilado por el usuario. ¿Cómo realizamos esto? Pues nosotros hemos delegado en package.json la orden *npm run watch* para utilizar Watchify [26] y Babelify [27] para la traducción a ES5 que es la versión que implementan los navegadores y así generar el bundle.js con main.js que será el desencadenante de renderizar todo el DOM de React que está ubicado en una página padre que hemos denominado "App.js". Veamos los elementos que existen dentro de la carpeta "js" y después veremos una ilustración de "App.js" que pasaremos a describir.

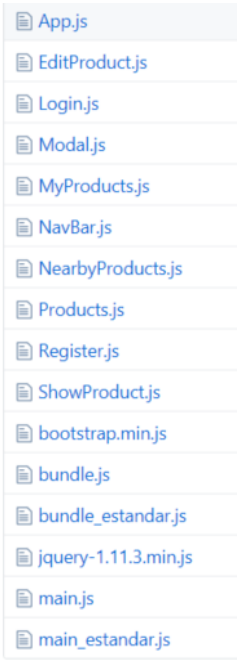


Figura 29 – Ficheros contenidos en "js"

```
var App = React.createClass({
        getInitialState: function() {
        return {
            logueado: (localStorage.getItem('token') != undefined),
                                                myProducts: false,
                                                nearbyProducts: false,
                                                allProducts: true,
                                                registered: false,
            mensaje: '',
            data: [],
        }
    },
    //Metodos para que otros componentes puedan cambiar el estado del principal
    login() {
       this.setState({logueado: true, mensaje: ''});
    },
    logout() {
       this.setState({logueado: false, mensaje: 'Has cerrado la sesión'});
    },
          OnMyProducts() {
                  this.setState({myProducts: true, allProducts: false, nearbyProducts: false})
          },
          OnAllProducts() {
                  this.setState({allProducts: true, myProducts: false, nearbyProducts: false})
          },
          OnNearbyProducts() {
                  this.setState({nearbyProducts: true, myProducts: false, allProducts: false})
          },
    setMensaje(m) {
           this.setState({mensaje: m});
    },
    mensajeLeido() {
           this.setState({mensaje: ''});
    },
    setData(data) {
       this.setState({data: data});
    },

          registerUser() {
                  this.setState({registered: true})
          },

          createUserSuccess() {
                  this.registerCerrar();
          },

          registerCerrar() {
                  this.setState({registered: false})
          },

    render: function() {
          let container;
          let modal;
                  let registerModal;

                  //Si el usuario está logueado muestran los productos
                 if (this.state.logueado && this.state.myProducts && !this.state.allProducts && !this.state.nearbyProducts) {
                        container = (
```

Figura 30 – Parte del archivo App.js

Dentro de este fichero "padre" es donde se llama a cada uno de los componentes que se cargarán según las *states* almacenadas. Estas se tratan de valores de los que está compuesto el componente interno y controlado por este.

¿Cómo se actualizan? Mediante la función *this.setState ({atributo: valor})* donde "this" sería el objeto. Cada vez que una de estas cambia su estado se llama a render () que actualiza la nueva salida dependiendo en nuestro caso de los valores que hacen elegir un componente u otro como son (*myProducts, nearbyProducts, allProducts, registered)* y con ello mostrar o todos los Productos, los Productos cercanos, Mis Productos o la Lista de Deseos. Además existe una función que describe el estado inicial del componente como es *getInitialState* () y una serie de funciones como *componentDidMount ()* realiza render la primera vez si existe. *ComponentWillUpdate (nextProps, nextStates), componentDidUpdate (prevsProps, prevsStates), render ()* que se encargan de actualizar el DOM cada vez que un estado se actualiza. Además es posible crear en el funciones definidas por nosotros que manejaran los datos contenidos como pueden ser *login (), OnMyProducts (), OnAllProducts (), OnNearbyProducts (), etc.* Si algún componente que derive de este su función necesita un *state* del componente padre este se incluye en los parámetros del componente importado como podemos ver en Render, por ejemplo con "Products" y la visión de este componente sería la siguiente.

Figura 31 – Parte del Componente Products.js

Como podemos apreciar no nos encontramos *states*, pero sí *props* que son una especie de referencia hacia una instancia del componente padre. Además también podemos apreciar como hay código Bootstrap que importamos desde la librería condicionada para ello, en la que también los elementos son componentes React.

8.5 DESPLIEGUE DEL PROYECTO

Antes de nada debemos tener instalado MongoDB, NodeJS y ReactJS.

Debemos crear una base de datos en MongoDB cuyo nombre sea ThingyDB. Para ello: entramos en mongo con el comando *mongo* y creamos una *use thingydb*. En el archivo de configuración donde la variable MONGODB veremos: *mongodb://localhost:27017/thingydb*

Ahora nada más tendremos que abrir 3 terminales en los que ejecutemos:

npm start, *npm run watch* y *npm run watch_estandar* que arrancarán tanto el API REST como la aplicación cliente del proyecto.

8.6 CASOS DE ESTUDIO

Nos centraremos en este apartado en detallar gráfica y textualmente el funcionamiento de la aplicación.

Esto se podría ver como dos modos, desde el modo comprador y desde el modo vendedor. Comencemos por el modo comprador.

Como es habitual nada más entrar nos encontraremos con la interfaz de Login (si no se introducen los campos correctos este usuario no se logueará, esto es un ejemplo para mostrar las validaciones).

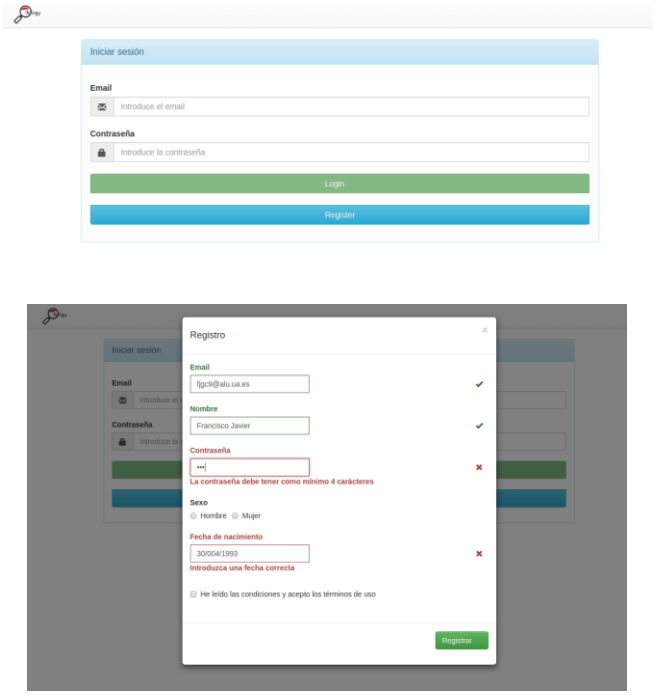


Figura 32 – Interfaz de Login y Registro

Como podemos observar arriba tenemos una Barra de Navegación en la que no se muestra nada hasta el momento. Justo bajo aparece un recuadro donde nos permite 2 opciones:

- Iniciar sesión, introduciendo email y contraseña (ambos campos están controlados por validaciones para un correo correcto y contraseña correcta)
- Registrarse, donde se abrirá un modal en el que antes (si aún no se ha permitido) lanzará una notificación el navegador pidiendo

permiso para recoger la localización del usuario. Más tarde podremos rellenar los campos que nos pide la aplicación.

Tras acceder logueandonos deberemos pulsar Login y si ha sido mediante Registro, el navegador nos redirigirá automáticamente. Ahora nos encontramos en la interfaz Home donde se encuentran Todos los Productos.

Como podemos ver en la parte superior del navegador podremos acceder a otras interfaces como son la de Mis Productos, En mi zona, Mi lista de deseos, Mi Perfil o simplemente hacer Logout. Además de realizar una búsqueda.

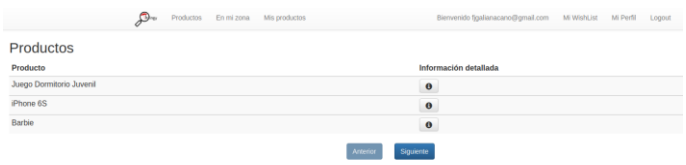


Figura 33 – Interfaz de "Todos los productos"

Si queremos saber más información acerca de un producto ("Me ha llamado la atención el título", "¿Encajará en mi descripción?", "Está muy visitado…, ¡voy a darme prisa!", "¡Es un producto nuevo!", "Este usuario tiene cosas curiosas") tan solo deberemos pulsar en el botón de información detallada, se nos abrirá un modal en el que aparecerá la información de dicho producto, del usuario donde verás otros productos suyos y unas opciones las cuales se pueden pinchar, "Añadir a WishList" que enviará el producto a tu "Lista de deseos" donde se podrán eliminar o borrar los productos que hayan contenido en ella. Una vista como la de "Todos los productos" o pinchar en "Lo quiero" y se facilitará la información para contactar con el vendedor.

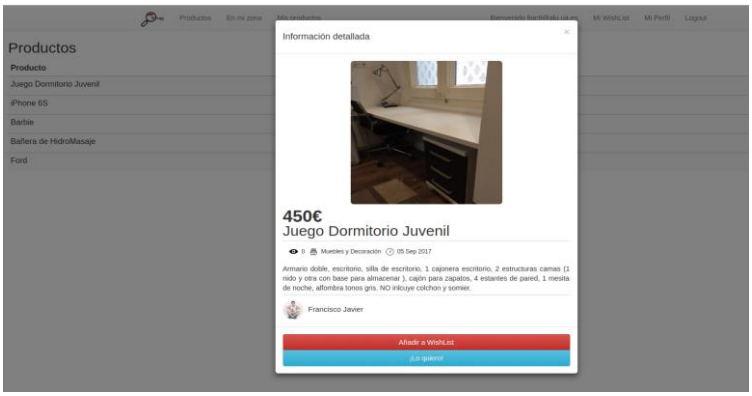


Figura 34 – Interfaz de "Información detallada"

Si sólo queremos salir, bastará con pinchar en la cruz que se sitúa en la parte superior derecha.

También al usuario le puede interesar el catálogo de productos que pueden haber cerca de su zona, así que para ello pinchamos "En mi zona", esta página nos mostrará los productos más cercanos a nuestra zona así como ver su información detallada, agregar a su lista de deseos o pinchar en "Lo quiero!".

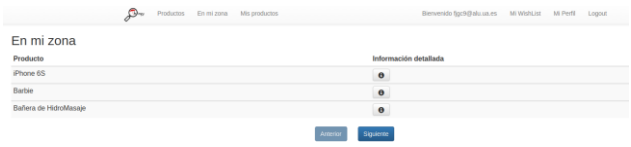


Figura 35 – Interfaz de "En mi zona"

Además podrá acceder a su perfil y cambiar su información personal o darte de Baja puesto que aparecerán tus datos personales y dos opciones para editar estas o para darte de baja.

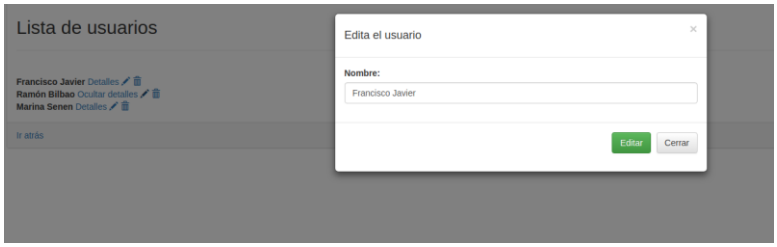


Figura 36 – Interfaz de "Mi Perfil"

Finalmente en el modo vendedor, estando logueado podemos acceder a "Mis Productos", donde tendremos la posibilidad de acceder a tus productos "En venta" pudiendo realizar acciones sobre ellos como editar su información pulsando el icono del lápiz o borrar un producto pulsando el icono de la papelera.

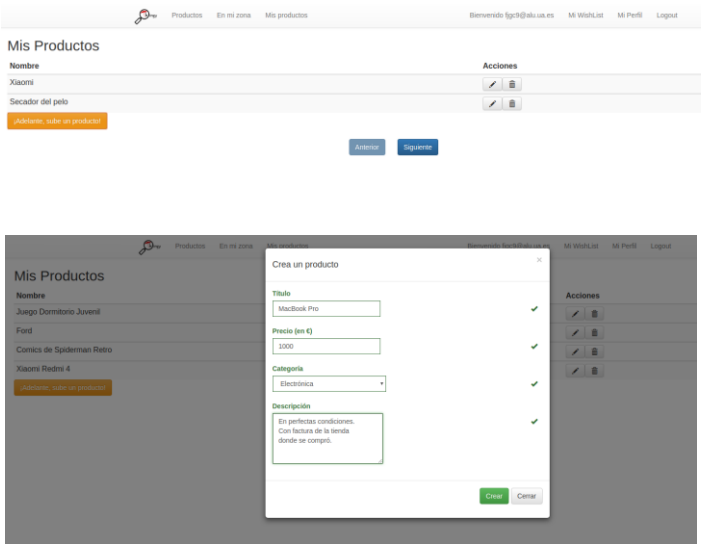


Figura 37 – Interfaz de "Mis Productos"

8.7 BRANDING

Hemos querido realizar Branding, se dice del proceso que se encarga de hacer y construir una marca mediante la administración estratégica del conjunto total de activos vinculados de forma directa o indirecta al nombre. Con ello hemos generado un logo mediante el nombre del proyecto.

En él hemos querido relacionar 3 conceptos:

- El nombre, quiere decir cosita en inglés que es lo que hay dentro de este proyecto, la gente sube "cosas".
- El icono de la lupa, sombrilla y silla, además de formar parte del nombre estamos dando a ver conocer que al entrar en la web podrás disfrutar de una serie de productos donde los que podrás buscar y encontrar el que se ajusta a tu medida.

- Finalmente, la sombrilla y la silla no es más que nada, una comparación en la que la búsqueda y navegación por nuestra aplicación será en tu tiempo libre y tranquilamente relajado.

  Uniendo todos ellos definimos el logo como el que aparece en la siguiente ilustración.

Figura 38 – Logo de Thingy

8.8 PROBLEMAS

El principal problema que se ha detectado ha sido el desconocimiento del lenguaje de programación sumado a los frameworks y variaciones con respecto al lenguaje de programación habitualmente utilizado.

En el apartado de Front-End cambiar la inercia a HTML sobre React debido a ser tan similar a HTML pero no serlo. Es difícil al principio pero la curva de aprendizaje es favorable con práctica.

Tener claro muy claro el concepto de API para definir el servidor y la Aplicación Cliente para definir el cliente puesto que los errores no se muestran

todos en la misma consola, si no unos en la consola del navegador y otros en la consola de la máquina donde se ejecute el servidor.

Otra problemática ha sido la carga de la función de "En mi zona" que muestra los productos cercanos puesto que realiza muchas búsquedas y es necesario refactorizar para encontrar la función mínima posible.

Inconveniente también la continua puesta en marcha del compilador que autocompila a JavaScript los componentes React unidos a cambios en el servidor lo que hacía volver a reiniciar en cada cambio hasta que se pone fin añadiendo Nodemon.

La planificación es otro de los puntos débiles a tratar, no como error sino como aspecto a mejorar puesto que aun marcando un correcto WorkInProgress en el tablón de planificación, a veces faltaba pulir el desglose de alguna funcionalidad a desarrollar.

8.9 MEJORAS

Es posible alargar los proyectos el tiempo deseado puesto que las tecnologías como las personas evolucionan y es posible que aparezcan componentes anticuados con el paso del tiempo, mejoras o ampliaciones de la misma, aquí van las más destacadas:

- Podernos dar de alta mediante cuentas de Google, Facebook, etc. Mejorará la experiencia de usuario puesto que no deberá de rellenar ningún tipo de campo. Por ejemplo, mediante PassportJS.

- Añadir a la publicación del anuncio/producto una nube de hashtags que formen parte de una categoría que relacionen palabras y así poder realizar búsquedas más poderosas o mejorar el panel de Home. Por ejemplo, mediante algún tipo de Api de lingüística mediante inteligencia artificial.

- Añadir notificaciones en la barra de navegación para notificar de las acciones del momento más importantes o interacciones hacia dicho usuario

- Ampliar la búsqueda "En mi zona" para devolver apartados como por ejemplo: "Lo que más se busca en tu zona", "Qué han subido recientemente", etc.

- Mejorar servicio de feedback por valoraciones tanto de un comprador como de un vendedor

- Añadir un servicio de mensajería interna para contactar entre comprador-vendedor fácilmente

- Tras mejorar el API REST, base de la aplicación, llevar la aplicación cliente al resto de dispositivos, puesto que se está produciendo un auge en la compra por smartphone

- Subir el API Rest y la aplicación Cliente a un servidor de internet, y tras esto:
  - Buscar formas de monetización, ya sea:
    - Lograr una buena tasa de conversión mediante mejora del diseño de la aplicación cliente haciéndola más adaptable, refactorizar para mejorar su posicionamiento en los buscadores o desarrollar Email Marketing para generar fieles que propaguen la aplicación por la red.
    - Crear una opción Premium que incluya poder mostrarte arriba del todo siendo "Anuncio relevante" pagando una cierta cantidad de dinero.
    - Iniciar campañas de marketing para dar publicidad al producto una vez desarrollado por completo

9. CONCLUSIONES

Thingy no se trata simplemente de una aplicación. Su base reside en su API Rest que sumado a las mejoras y ampliaciones pertinentes podrá llegar, mediante el correcto desarrollo de sus distintas formas de aplicación cliente a cualquier usuario, y como mencionábamos al principio facilitar así a los usuarios la acción de matching y compra de un producto mientras se recaban datos.

Gracias a los usuarios finales que consuman dicha aplicación, ésta evolucionará favorablemente el proyecto con la consecuente obtención de beneficios.

En este proyecto hemos intentado acércanos lo más posible a un ambiente profesional de una empresa.

Por ello hemos llevado a cabo el "Método tecnológico" junto a una planificación donde imperaba la metodología ágil basada en Scrum unido a la utilización de herramientas innovadoras de programación y frameworks como NodeJS, ExpressJS, ReactJS. Eligiendo MongoDB en la capa de persistencia, una tecnología puntera en el manejo de grandes volúmenes de información, BigData.

Además de herramientas de planificación como Trello simulando tablero Scrum, control de versiones con Git simulando un proyecto real y utilizando herramientas para la generación de diagramas, librerías, etc.

Destacar la posibilidad de dar opción a que los desarrolladores pudiesen crear sus propias aplicaciones clientes conectándose a la API para tratar con los grandes volúmenes de datos dispuestos a manejar en Thingy, así como potenciar el desarrollo de una aplicación móvil ya que como describíamos en el informe DiTrendia de 2016, se ha incrementado tres veces más el uso de estos que el e-commerce tradicional.

Desde el punto de vista propio, mencionar que ha sido un gran desafío debido a no poseer la suficiente madurez en un gran número de estas tecnologías puesto que no tenía conocimiento previo de ellas. Me llevo una experiencia cercana en un "ecosistema

profesional generado" además de haber mejorado mis habilidades como desarrollador que aún están por pulir.

Cabe destacar que es un producto que desde que pude ver que la propuesta había sido aceptada me ilusionó, no he podido desarrollar de la mejor forma y todo lo bien que hubiese querido por ello me gustaría continuar hasta poder ésta a PlayStore en forma de aplicación móvil y por qué no intentar competir con el resto de competidores.

10. BIBLIOGRAFÍA Y REFERENCIAS

[1] Wikipedia. (2017). Diseño Centrado en el Usuario. Accedida 10 diciembre 2016, desde https://es.wikipedia.org/wiki/Dise%C3%B1o_centrado_en_el_usuario

Martín, David. (2010). Diseño de aplicaciones adaptativas. Accedida 10 diciembre 2016, desde http://www.nosolousabilidad.com/articulos/aplicaciones_adaptativas.htm

[2] Ditrendia. (2016). Informe ditrendia 2016: Mobile en España y en el Mundo. Accedida 3 febrero 2017, desde http://www.amic.media/media/files/file_352_1050.pdf

[3] Fuente, Oscar (2015). Qué es la economía colaborativa: Ejemplos, ventajas y datos más relevantes. Accedida 3 febrero 2017, desde http://comunidad.iebschool.com/iebs/emprendedores-y-gestion-empresarial/economia-colaborativa-consumo

[4] Wallapop. (2017). Accedida 5 octubre 2016, desde http://www.wallapop.com

[4] Archanco, Eduardo. (2016). Estrategia de Wallapop: todas las razones de su éxito. Accedida 5 febrero 2017, desde http://elespectadordigital.com/estrategia-de-wallapop-exito

[5] LetGo. (2017). Accedida 5 octubre 2016, desde https://es.letgo.com/es

[5] Wikipedia. (2017). Letgo. Accedida 5 febrero 2017, desde https://en.wikipedia.org/wiki/Letgo

[5] Llensa, Emma. (2014). La ley de Fitts aplicada a dispositivos móviles. Accedida 4 febrero 2017, desde http://www.ubicuostudio.com/es/diseno-grafico/ley-fitts-aplicada-diseno-apps

[6] MilAnuncios. (2017). Accedida 5 octubre 2016, desde https://www.milanuncios.es

[7] Vibbo. (2017). Accedida 5 octubre 2016, desde https://www.vibbo.com

Llensa, Emma. (2014). La ley de Fitts aplicada a dispositivos móviles. Accedida 4 febrero 2017, desde http://www.ubicuostudio.com/es/diseno-grafico/ley-fitts-aplicada-diseno-apps

OBS Business School. (2017). Los 4 factores que más influyen en el éxito de un proyecto. Accedida 10 marzo 2017, desde http://www.obs-edu.com/es/blog-project-management/actualidad-project-management/los-4-factores-que-mas-influyen-en-el-exito-de-un-proyecto

[7] Ubuntu. (2017). Ubuntu. Accedida 10 diciembre 2017, desde https://www.ubuntu.com

[8] Trello. (2017). Trello.com. Accedida 1 febrero 2017, desde https://trello.com

[9] Visio. (2017). Visio. Accedida 2 febrero 2017, desde https://es.wikipedia.org/wiki/Microsoft_Visio

[10] Github. (2017). Accedida 20 mayo 2017, desde https://github.com

[11] Atom. (2017). Accedida 20 mayo 2017, desde https://atom.io

MDN. (2017). JavaScript. Accedida 10 abril 2017, desde https://developer.mozilla.org/es/docs/Web/JavaScript

[12] MongoDB. Accedida 10 abril 2017, desde https://docs.mongodb.com/?_ga=2.241804866.1101103380.1504480291-2037111583.1504480291

[12] MongoDB. Bases de Datos NoSQL. Accedida 10 abril 2017, desde https://es.wikipedia.org/wiki/NoSQL

[12] MongoDB. Bases de Datos Documentales. Accedida 10 abril 2017, desde https://es.wikipedia.org/wiki/Base_de_datos_documental

MongoDB. (2017). Presentación y comparación con MySQL. Accedida 10 abril 2017, desde https://www.1and1.es/digitalguide/paginas-web/desarrollo-web/mongodb-presentacion-y-comparacion-con-mysql

MongoDB. (2017). MongoDB vs MySQL. Accedida 12 abril 2017, desde https://www.gpsos.es/2017/03/mysql-vs-mongodb

[13] NodeJS. Accedida 14 abril 2017, desde https://nodejs.org/es

[13] Sopa de Bits. (2013). Accedida 14 abril 2017, desde https://www.sopadebits.sdweb.es/novas/2013-11/nodejs-pros-contras-y-algunas-notas

FHIOS. (2017). Qué es y para qué sirve un análisis funcional. Accedida 10 abril 2017, desde https://www.fhios.es/para-que-sirve-un-analisis-funcional

Metodología Gestión de Requerimientos. (2017). Accedida 10 abril 2017, desde https://sites.google.com/site/metodologiareq/capitulo-ii/tecnicas-para-identificar-requisitos-funcionales-y-no-funcionales

Javier J. (2017). Requisitos. Accedida 10 abril 2017, desde http://www.lsi.us.es/~javierj/cursos_ficheros/02.%20Un%20ejemplo%20de%20requisitos.pdf

Francisco Javier Galiana Cano. (2017). Repositorio Thing. Accedido desde https://github.com/frangaliana/rest-api-thingy

[15] Uriel Hernández. (2014). MVC (Model, View, Controller). Accedido 14 abril 2017, desde https://codigofacilito.com/articulos/mvc-model-view-controller-explicado

[16] Equipo IngenioInc. (2013). La importancia de la imagen a la hora de vender. Accedido 14 abril 2017 desde https://www.ingenioinc.com/la-importancia-de-la-imagen-a-la-hora-de-vender

[17] Hosteltur. (2017). El poder de las opiniones en la intención de compra del cliente. Accedido 16 abril 2017 desde https://www.hosteltur.com/116585_poder-opiniones-intencion-compra-cliente.html

[18] Mongoose. (2017). Mongoose Docs. Accedida 15 abril 2017, desde http://mongoosejs.com/docs/guide.html

Colomina, Otto. (2016). REST con Node y Express. Accedida 17 abril, desde https://moodle2016-17.ua.es/moodle/pluginfile.php/79721/mod_resource/content/2/REST_con_Node_y_Express.html

[19] Azaustre, Carlos. (2016). Autenticación con JWT. Accedida 17 abril 2017, desde https://carlosazaustre.es/que-es-la-autenticacion-con-token

Azaustre, Carlos. (2016). Curso NodeJS y MongoDB. Accedida 17 abril 2017, desde https://www.youtube.com/watch?v=JwZKcm3zWcA&list=PLUdlARNXMVkk7E88zOrphPyGdS50Tadlr

[20] JWT. (2017). Introduction to JSON Web Token. Accedida 17 abril 2017, desde https://jwt.io/introduction

[25] Nodemon. (2017). Accedida 17 abril, desde https://nodemon.io

[21] Vicente, David. (2017). Encriptación de password en NodeJS y MongoDB: bcrypt. Accedida 17 abril 2017, desde https://solidgeargroup.com/password-nodejs-mongodb-bcrypt?lang=es

Stackoverflow.com. (2017). Stack Overflow. Accedida 21 abril 2017, desde http://stackoverflow.com

[22] XurxoDev. (2017). Testeando un Api Rest con mocha. Accedido 25 mayo 2017, desde http://xurxodev.com/testeando-un-api-rest-con-mocha

React. (2017). React vs Angular2. Accedida 17 mayo, desde http://www.enrique7mc.com/2016/01/comparacion-react-js-vs-angular-2

[14]React. (2017). React Docs. Accedida 20 mayo 2017, desde https://facebook.github.io/react/docs/hello-world.html

Es.stackoverflow.com. (2017). Stack Overflow en español. Accedida 1 junio 2017, desde http://es.stackoverflow.com

[23] Wikipedia. (2017). DOM. Accedida 15 julio 2017, desde https://es.wikipedia.org/wiki/Document_Object_Model

[24] Daniel Smith's Personal Blog. (2017). Accedida 15 julio 2017, desde https://dsmith77.wordpress.com/2008/07/20/the-document-object-model-dom

[26] Watchify. (2017). Accedida 21 julio 2017, desde https://www.npmjs.com/package/watchify

[27] Babelify. (2017). Accedida 21 julio 2017, desde https://www.npmjs.com/package/babelify

React-Bootstrap. (2017). React-Bootstrap. Accedida 20 agosto 2017, desde https://react-bootstrap.github.io/components.html