Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

INSTACOMPONENT: APLICACIÓN WEB PARA REPARTO DE COMPONENTES TECNOLÓGICOS A DOMICILIO

Alumno: Álvaro Herrero Sanz

Tutor: Prof. D. José Ramón Balsas Almagro

Dpto: Departamento de Informática

Septiembre, 2021

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don José Ramón Balsas Almagro, tutor del Trabajo Fin de Grado titulado: **InstaComponent: aplicación web para reparto de componentes tecnológicos a domicilio**, que presenta Álvaro Herrero Sanz, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, SEPTIEMBRE de 2021

El alumno: Los tutores:

Álvaro Herrero Sanz José Ramón Balsas Almagro

ÍNDICE

Tabla de ilustraciones

1. Introducción

El presente trabajo pretende aportar una solución informática de calidad que sirva para dar respuesta a las necesidades específicas de cualquier empresa que requiera de un sistema de venta de productos y su posterior entrega, caracterizándose esta por realizarse en el momento y de una manera rápida y eficaz. Una analogía del servicio propuesto podrían ser las empresas de reparto de comida a domicilio, muy de moda hoy en día. Para dicho cometido vamos a plantear la empresa ficticia InstaComponent S.A.L., que será la empresa para la que se desarrollará el sistema en cuestión.

Figura 1. Logo InstaComponent S.A.L.

InstaComponent es una empresa de reparto al más propio estilo Uber Eats o Just Eat. La diferencia es que, en este caso, no se trabaja con el sector gastronómico, sino con el sector tecnológico: InstaComponent te llevará a casa el artículo tecnológico que desees en un tiempo récord.

InstaComponent nace en una época complicada y excepcional para todo el mundo como es la pandemia provocada por la COVID-19 lo que nos servirá para proporcionar un contexto a esta oportunidad de negocio. Según un estudio realizado por IAB, una de las mayores asociaciones del mundo de comunicación, marketing digital y publicidad: “Una de cada dos personas entrevistadas aumentó la frecuencia

de las cibercompras, y un 45% ha empezado a comprar online productos físicos, algo que no hacía antes de la pandemia" [1]. En relación al sector tecnológico, DE-CIX nos muestra en uno de sus estudios, como el uso de las tecnologías ha crecido exponencialmente: el uso de herramientas *online* colaborativas se ha multiplicado por 5, el *gaming online* ha crecido hasta un 30% y las plataformas de vídeo en *streaming* han llegado a doblar sus usuarios en esta crisis. Por otra parte, en relación al envío de comida a domicilio, que se asemeja bastante al servicio que nuestra empresa va a ofrecer, podemos encontrar el siguiente dato:

"Los datos apuntalan lo que se percibe en el sector: el envío de comida a domicilio ha pasado de suponer un 3-4% de las ventas a duplicarse hasta el 8% del sector de la restauración desde marzo, según la consultora NPD, que recuerda que durante el confinamiento muchos locales optaron por esta vía como única alternativa al cierre." [2]

Puede que tras situaciones como esta, y pese a que el virus sea superado, la población, hasta pasado un período de tiempo, se mantenga un poco reacia a acudir a tiendas físicas para evitar masificaciones, por lo que prefiera acudir a este servicio para adquirir cualquier compra del sector. También hay que tener en cuenta, y es un factor que puede servir para ganar clientes, la comodidad que supone el uso de este servicio; a un usuario de cualquiera de las ciudades en las que se va a implantar esta idea de negocio (grandes ciudades españolas), usar dicho servicio, por norma general, le va a proporcionar una experiencia infinitamente mejor que desplazarse por ellos mismos a los establecimientos de compra, o esperar la entrega de dicho artículo si se ha comprado de manera on-line (dejando a un lado a los competidores directos del servicio propuesto).

Otro factor que habría que tener en cuenta sería la postura de los propios comercios a la hora de adoptar este modelo de negocio, ¿por qué un comercio preferiría contratar los servicios de InstaComponent en lugar de utilizar su propia web de ventas o canal de distribución? De acuerdo, pues, en primer lugar, hay que dejar claro que ambos servicios no son excluyentes: recuerdo que atendiendo al plan de negocio de InstaComponent, la idea no es que el servicio se implante en todas las

ciudades (solamente en grandes urbes a priori), y por tanto debe de existir un servicio de venta y reparto propio para llegar allá donde no lleguen los servicios que aquí se proponen. Dejando eso a un lado, se podría tener en cuenta que la “adhesión” a InstaComponent por parte de las empresas, puede beneficiar a estas a la hora de ahorrar en los costes derivados del mantenimiento de sus propios sitios web o de la contratación de personal de reparto.

Este negocio no está enfocado a un nicho de mercado en concreto, sino que es capaz de adaptarse a las necesidades de diferentes grupos de personas. Por una parte, puede ser de gran utilidad para personas jóvenes y estudiantes, por la naturaleza del negocio (este grupo de personas está cada vez más estrechamente ligado a las tecnologías) sumado al sistema de códigos y promociones, que será exprimido al máximo por este público. Por otra parte, inevitablemente está ligado al mundo laboral dado que puede ser de gran utilidad para empresas en situaciones concretas de urgencia, debido a la celeridad del reparto. También puede servir de ayuda al otro frente del mundo laboral, los trabajadores: el servicio está activo durante una ventana horaria que abarca toda la jornada y que permite su uso desde primera hora de la mañana hasta prácticamente la noche para que todo el mundo pueda acceder en algún momento a él. Y, por último, no se debe olvidar que InstaComponent también está enfocado a clientes con movilidad reducida, que les suponga un esfuerzo extra acudir a un establecimiento a comprar y que, de esta manera, se le pueda la vida un poco más fácil a este grupo de personas.

1.1. Motivación

Tal y cómo acabamos de comentar, el proyecto va a consistir en diseñar y desarrollar una aplicación web para la empresa anteriormente explicada. Aunque pueda parecer obvio, el motivo por el que vamos a desarrollar esta aplicación es debido a que esta empresa y modelo de negocio, necesitan de un sistema para sustentarse y funcionar, a la vez, que será el punto de conexión entre la misma y los clientes, mediante el cual, estos podrán disfrutar de los servicios proporcionados por InstaComponent. Una aplicación web es accesible para prácticamente la totalidad de los usuarios, a la vez que es sencilla de utilizar. También permite abaratar en gran

medida los costes de desarrollo debido a la naturaleza de la misma, mientras que el proceso de soporte y mantenimiento de la misma es fácil de llevar a cabo.

Como ya sabemos, la aplicación va dirigida a un amplio abanico de usuarios por lo que uno de los pilares fundamentales de la misma es que sea entendible y fácil de utilizar para todos los públicos, como ya hemos comentado anteriormente. Además, al ser una aplicación que siempre va a estar en contacto directo con el usuario, se desea que presente una vistosa interfaz a la par que fluida, sin dejar de lado un funcionamiento robusto, que no dé lugar a ningún tipo de error.

Personalmente hablando, el propósito de este proyecto es muy claro: adquirir las tecnologías/metodologías que se utilizan hoy en día en el marco profesional, por lo que llevar adelante este trabajo puede ser la preparación idónea para la posterior incorporación al mundo laboral, puesto que se va a trabajar para cubrir necesidades que podrían ser completamente reales para empresas/usuarios.

1.2. Objetivos

En el planteamiento del presente TFG se establecieron los siguientes objetivos a desarrollar para obtener los resultados esperados:

- Realizar un estudio de necesidades y soluciones existentes en el contexto de los servicios de reparto a domicilio. En el actual proyecto se van a combinar los servicios que ofrecen las llamadas aplicaciones *food delivery* con el envío a domicilio convencional que pueden ofertar las tiendas especializadas en el sector informático/tecnológico.
- Diseñar un sistema informático especialmente orientado a la utilización de arquitecturas, principios de diseño y metodologías de desarrollo web.
- Seleccionar un entorno de desarrollo web e implementar un prototipo de aplicación web que satisfaga las necesidades que se hayan determinado.

1.3. Metodología

A la hora de llevar a cabo este proyecto, la metodología a seguir está clara desde un primer momento: se utilizará alguna metodología ágil, como por ejemplo SCRUM. Sin embargo, tenemos que tener en cuenta que, para utilizarla, existe la principal limitación de que el equipo está formado únicamente por un miembro, además de no existir un *stakeholder* o cliente real con el que intercambiar *feedback* del producto, por lo que esta situación podría denominarse coloquialmente como “ScrumBut”, que se trata de una situación en la cual se desea aplicar SCRUM, pero ciertas limitaciones lo impiden, como pasa en nuestro caso, por lo que el modo de trabajo se adapta a la condiciones presentes.

Para comenzar, no existirán los roles, o más bien, todos serán asignados a la misma persona. Cuando sea necesario adquirir los requisitos del producto, seremos el *Product Owner*, que simulará el trato con el cliente para así poder constituir la pila de producto además de “recibir un *feedback*” del producto entregado al final de cada iteración. Adquiriremos el papel de *SCRUM Master* cuando sea necesario aplicar las políticas y las pautas de SCRUM durante el desarrollo del proyecto. Finalmente seremos desarrolladores del equipo SCRUM cuando haga falta ponerse manos a la obra con la aplicación.

A la hora de realizar estimaciones acerca del tamaño de las historias de usuario propuestas, se usará la técnica “*Planning Poker*” (una vez más, con un solo usuario), ajustada ésta a la serie de valores presentes en la secuencia de Fibonacci, pero de una manera redondeada para facilitar la estimación.

Figura 2. Técnica Planning Poker con secuencia de Fibonacci redondeada. [3]

Con respecto a los *Sprints* o iteraciones, éstos tendrán una duración de dos semanas.

Finalmente, veamos cómo vamos a tratar las reuniones comunes de esta metodología. Obviamente, por los motivos anteriormente comentados, las reuniones no son posibles, aunque sí que sería posible realizar una simulación de las mismas. Para simular los *Stand-up meeting* se dedicarán 5 minutos cada día antes de empezar a trabajar para reflexionar acerca de las 3 preguntas fundamentales:

- ¿Qué hice ayer para ayudar al equipo a finalizar el *Sprint*?
- ¿Qué voy a hacer hoy para ayudar al equipo a finalizar el *Sprint*?
- ¿Qué obstáculos se están interponiendo en mi camino para finalizar el *Sprint*?

A la hora de imitar un *Sprint Review*, se dedicará una hora al final de cada Sprint para revisar todas las funcionalidades presentes en el entregable y valorarlas de una manera objetiva con la mayor abstracción posible; también existirá la probabilidad de modificar el *Product Backlog*, refinando las funcionalidades presentes o añadiendo nuevas tal y cómo se le ocurrirían a un cliente. Es complicado estar en todos los papeles al mismo tiempo, pero se realizará un esfuerzo extra para ver las cosas desde una perspectiva distinta dependiendo del rol que nos corresponda en cada momento. Y para acabar, con respecto al *Sprint Retrospective*, se reflexionará durante una última media hora, acerca de qué y cómo se puede mejorar a la hora de trabajar en el proyecto haciendo un fuerte ejercicio de autocrítica, analizando el proceso y pensando cómo podemos mejorar el trabajo de una manera inmediata.

Finalmente, se ha de recalcar que, aunque se utilizará un proceso iterativo, la memoria se organizará de una forma secuencial para facilitar la lectura, comprensión y localización rápida de todos los detalles y decisiones tomadas en cada uno de los aspectos principales del ciclo de vida del software.

1.4. Estructura del documento

El presente documento ha sido organizado de la siguiente manera:

El apartado número 2 ha sido completamente dedicado al Análisis. En primer lugar, se ha realizado un análisis y se ha estudiado las soluciones ya existentes similares a nuestro proyecto debido a que esto sería una referencia o punto de partida bastante válido, a partir del cual comenzar a modelar qué era lo que queríamos en nuestro sistema y cómo lo queríamos. Tras ello, este apartado viene proseguido una fase en la cual se han estudiado las posibles opciones y diferentes alternativas a tomar a nivel tecnológico. Tras ello, se encuentran relatadas la serie de Historias de Usuario que conforman nuestra pila de producto, acompañadas de una priorización MoSCoW y una planificación dividida en los *Sprints* que durará el desarrollo. La evolución de esta planificación previamente comentada ha sido descrita a continuación y se ha incluido un estudio detallado de los costes que conllevaría llevar a cabo este proyecto. Para finalizar se presenta el modelo de dominio, donde se empieza a esbozar la forma que va a tener el prototipo de nuestro sistema.

El apartado 3 de esta memoria está dedicado al Diseño del sistema. En él, se han utilizado diferentes diagramas: el diagrama entidad-relación, que refleja cómo las clases o entidades se relacionan entre sí, unas con otras; el diagrama de clases, en el cual se separarán específicamente las partes correspondientes al cliente y al servidor; o algunos diagramas de secuencia, mediante los cuales se representará el funcionamiento de los procesos más relevantes que se suceden en nuestro sistema. También se incluirán un subapartado dedicado a la interfaz del sistema y otro en el que se detalla el plan de pruebas con el que se validarán y testearán las funcionalidades implementadas en el sistema.

El apartado 4, dedicado a Implementación, está dividido en dos subapartados. En el primero de ellos se habla acerca de la arquitectura del sistema y cómo están organizados los diferentes componentes del mismo. En el segundo, dedicado a la implementación, se adjuntan fragmentos de código relevantes para el funcionamiento de la aplicación y se explica concretamente cuál es la función de los mismos.

El apartado 5 está enfocado a comentar los resultados de las pruebas propuestas anteriormente en el correspondiente plan de pruebas. El apartado 6 será dedicado a plantear las conclusiones obtenidas del desarrollo de este trabajo,

añadiendo, además, una serie de propuestas futuras y mejoras que se podrían realizar más adelante en nuestro proyecto.

Finalmente, se incluye una bibliografía con todo el contenido que se ha utilizado como referencia y unos apéndices con el manual de usuario y la descripción de los contenidos suministrados.

2. Análisis

2.1. Análisis Preliminar

Como ya sabemos, aunque nuestro modelo de negocio posea cierto componente innovador, al tratarse del reparto de artículos tecnológicos, se trata de un estilo de aplicación que está ciertamente estandarizado gracias a la serie de aplicaciones que ofrecen este servicio para la entrega de alimentos. Por ello vamos a tomar como referencia el cómo se está trabajando hoy en día en el desarrollo de dichas aplicaciones similares, pero modificando lo necesario con el objetivo de que la aplicación se adapte a la perfección a lo que nuestros clientes necesitan. También cabe destacar algo, y es que nos ceñiremos exclusivamente al diseño y desarrollo de software, por lo que no entraremos en temas administrativos, de organización o del funcionamiento de la empresa en sí (como pueden ser las negociaciones con franquicias/tiendas, dirección de los trabajadores o establecimiento de cuotas/precios), ya que estos exceden el alcance del presente trabajo.

Las pautas a la hora de desarrollar una aplicación estilo "*Delivery*" son claras:

"El diseño del sitio es funcional y discreto, en la medida de lo posible cumple con los objetivos del proyecto:

- Alta velocidad de descarga.
- Fácil navegación
- Cómoda visualización en pantallas de diferentes tamaños (diseño adaptativo).
- Nada distrae a los visitantes del propósito del servicio: la elección de la comida.

Puede acceder al sitio desde una computadora de escritorio, computadora portátil, tableta o teléfono móvil. Y en todas partes se mostrará correctamente, los elementos serán de tamaño fácil de usar e información importante a la vista." [4]

Por otra parte, también queda muy clara la importancia de la sincronización entre las acciones que realiza cada uno de los roles presentes y de la existencia de una API[1] para el funcionamiento:

[1] API: *Application Programming Interfaces* (https://red.ht/3jTebY1)

“Para organizar el trabajo de tales proyectos, se utiliza una arquitectura cliente-servidor y se desarrolla una API para intercambiar los datos.

También se requerirá API al organizar el intercambio de datos con los sitios y los programas de contabilidad de los restaurantes para actualizar los menús y los precios. Si un lugar se niega a ser automatizado, la actualización oportuna de los precios se convierte en su área de responsabilidad.

La disposición reflexiva del intercambio de datos entre las aplicaciones, los programas y el servidor es importante para el trabajo de todo el proyecto. Recomendamos que confíe el desarrollo de la parte del servidor del proyecto (*back-end*) y la parte del usuario (*front-end*) a un desarrollador profesional. Permitirá evitar múltiples problemas que puedan surgir en el proceso de configuración, así como escapar de posibles errores relacionados con el aspecto del factor humano (falta de interacción entre los desarrolladores).” [4]

2.2. Estudio de soluciones existentes

En este apartado vamos a echar un vistazo a las soluciones existentes ya creadas, es decir, a las aplicaciones con funcionalidades similares de las que sacar ideas que puedan ser relevantes para nuestro cometido, o ideas que no lo sean pero que puedan ser mejoradas mediante nuestro trabajo.

En primer lugar, vamos con la aplicación Uber Eats. Personalmente creo que su interfaz puede servir como referencia para nosotros ya que muestra una apariencia muy minimalista y simplificada, llegando a ser atractiva a la vista. Iconos simplificados sobre fondo blanco, menú desplegable a la izquierda cerrado por defecto, sección de sugerencias y opciones de filtrado intuitivas y limpias, acompañadas de una muestra de resultados claro y simétrico; todos estos elementos pueden servirnos de inspiración y guía a la hora de encauzar nuestro trabajo.

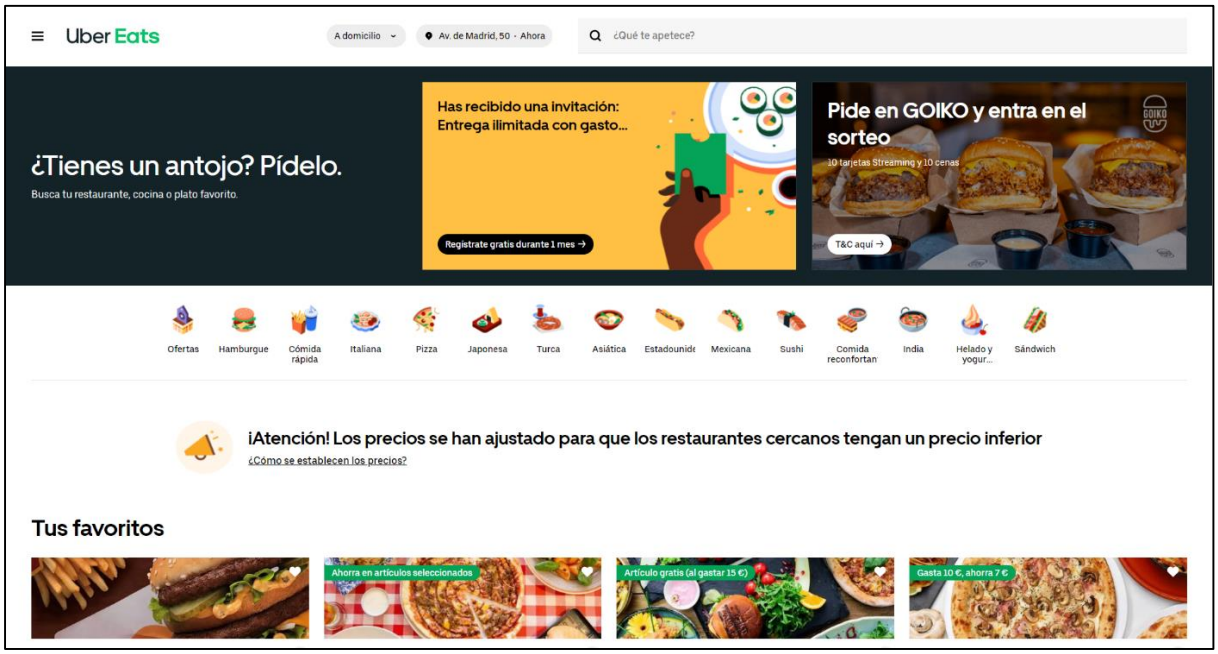


Figura 3. Página de inicio de Uber Eats: buscador, menú desplegable y categorías.

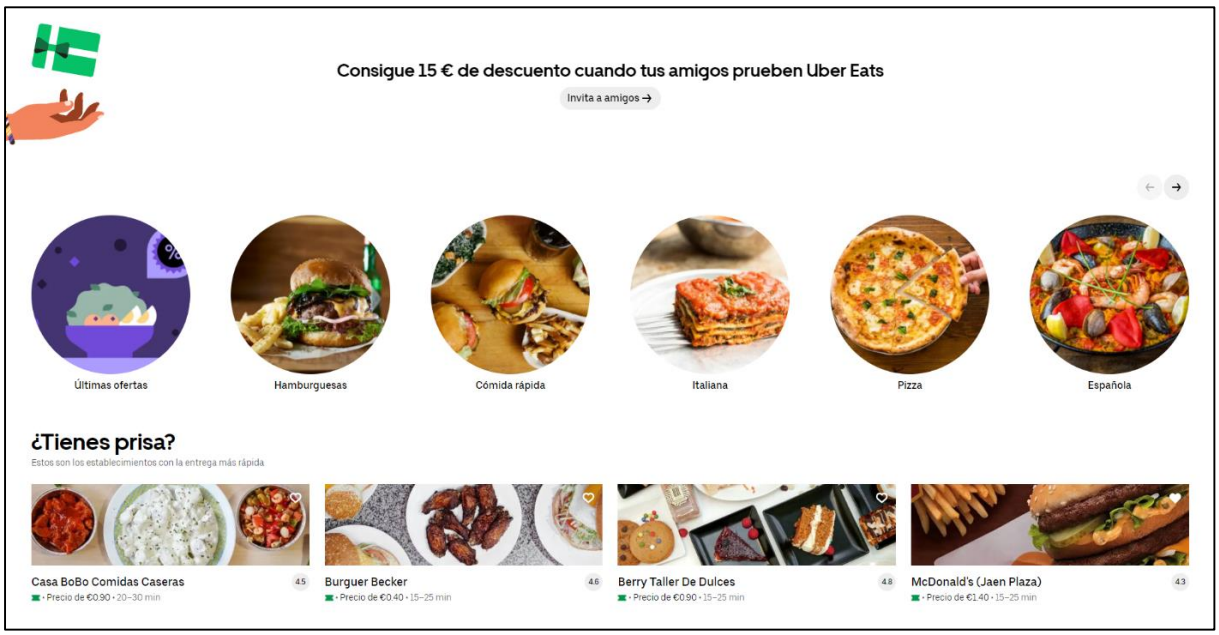


Figura 4. Página de inicio de Uber Eats: sugerencias.

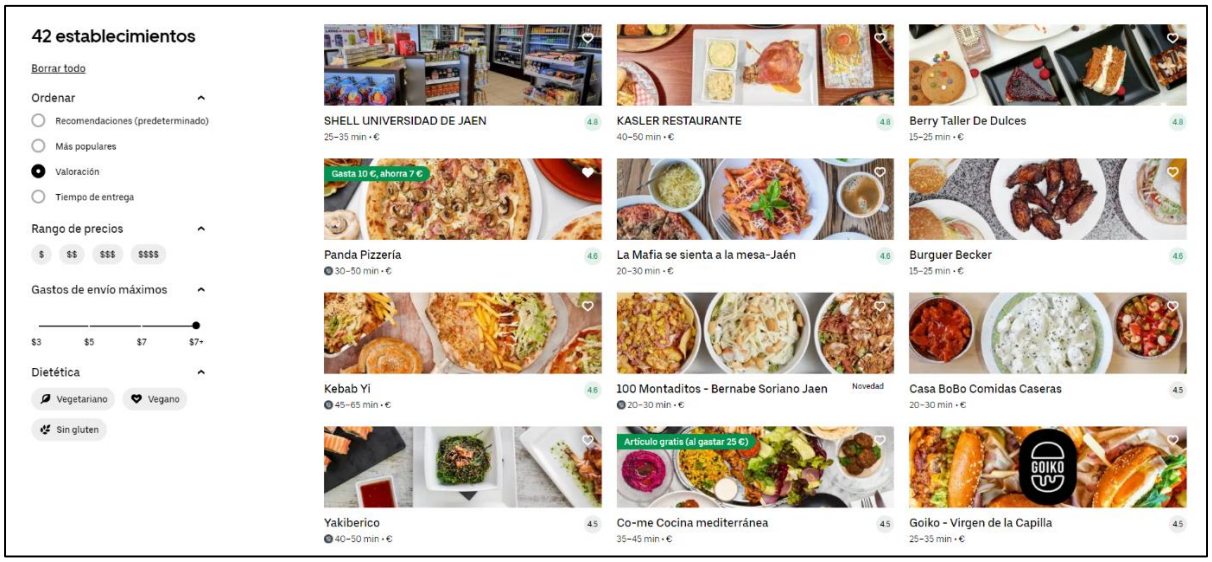


Figura 5. Buscador de Uber Eats.

Para testear a fondo su funcionamiento, se ha simulado el realizar un pedido de comida a McDonald's. Al elegir un establecimiento, aparece la carta de artículos disponibles, al igual que nosotros querremos que aparezca el catálogo del comercio seleccionado. Continúa el diseño minimalista y limpio, ordenado por categorías, además de incluir imagen y descripción en el caso de cada uno de los artículos a la venta. Al seleccionar una hamburguesa concretamente, nos ofrece incluirla en menú o modificar sus ingredientes; nosotros no trabajamos con comida, pero podríamos extender esta función a los artículos con los que trabajaremos (E.g. un usuario selecciona una Memoria USB 3.0 Sandisk 64 GB, y en las opciones disponibles aparece la versión correspondiente de 128 GB, con el incremento de precio correspondiente).

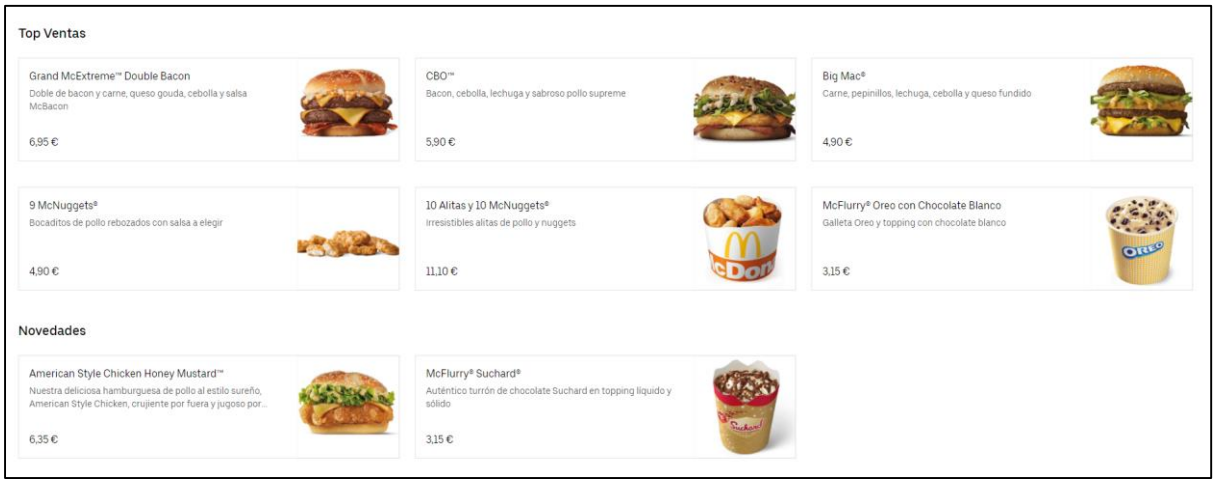


Figura 6. Pedido Uber Eats: selección de artículos.

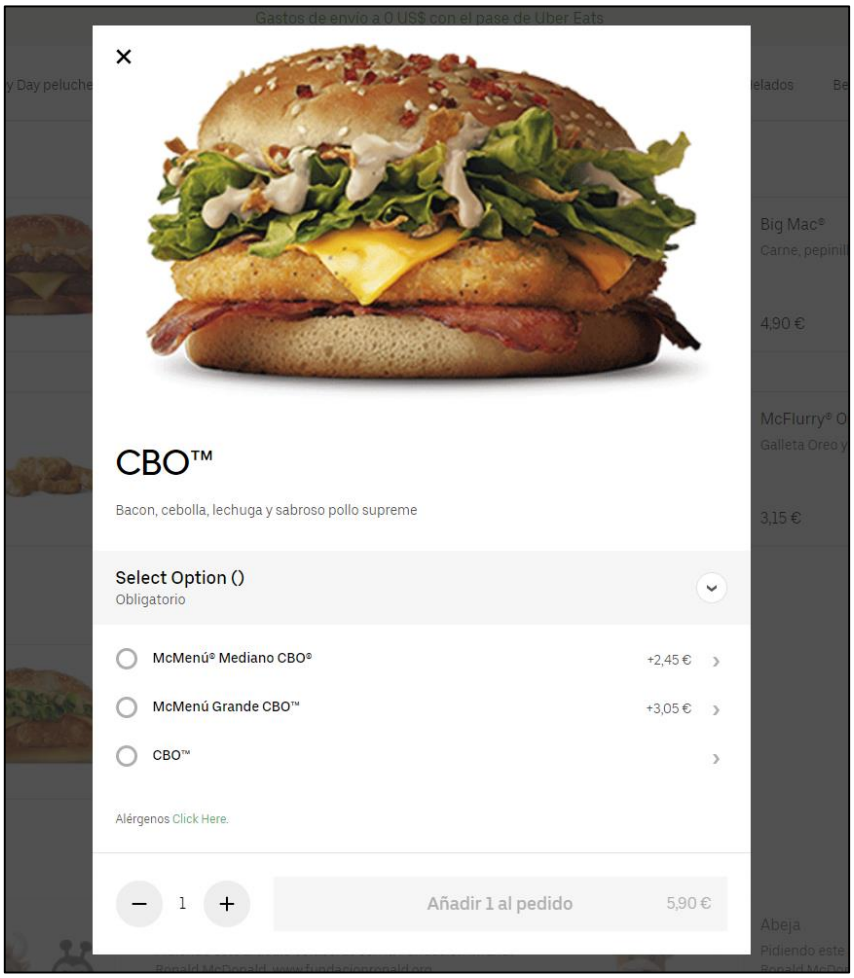


Figura 7. Pedido de Uber Eats: selección de artículo específico.

Una vez comenzado el pedido, la aplicación dispone de un botón en la esquina superior derecha de la pantalla, mediante el cual se despliega una vista previa del pedido provisional, incluyendo los artículos existentes en el mismo junto con los precios de cada uno y la suma total. Si pasamos a finalizar el pedido tenemos una pantalla final de confirmación del pedido en el que se pueden modificar las opciones de envío o de pago, entre otras. También se pueden introducir códigos promocionales para obtener descuentos en el pedido.

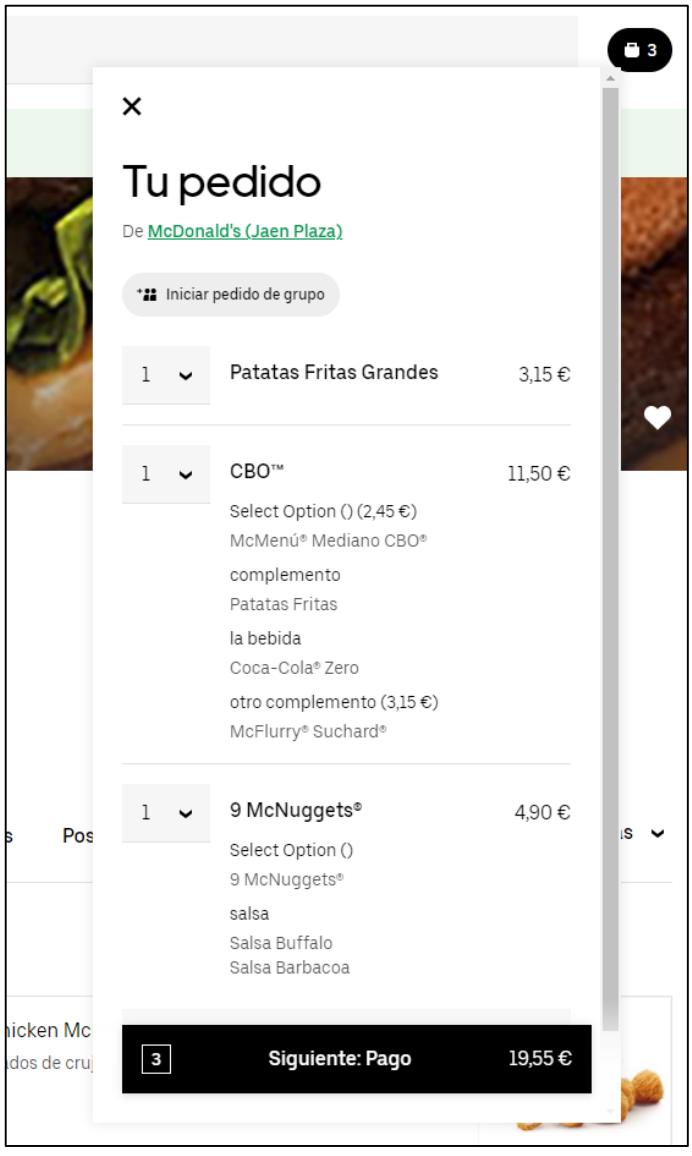


Figura 8. Pedido de Uber Eats: cesta provisional.

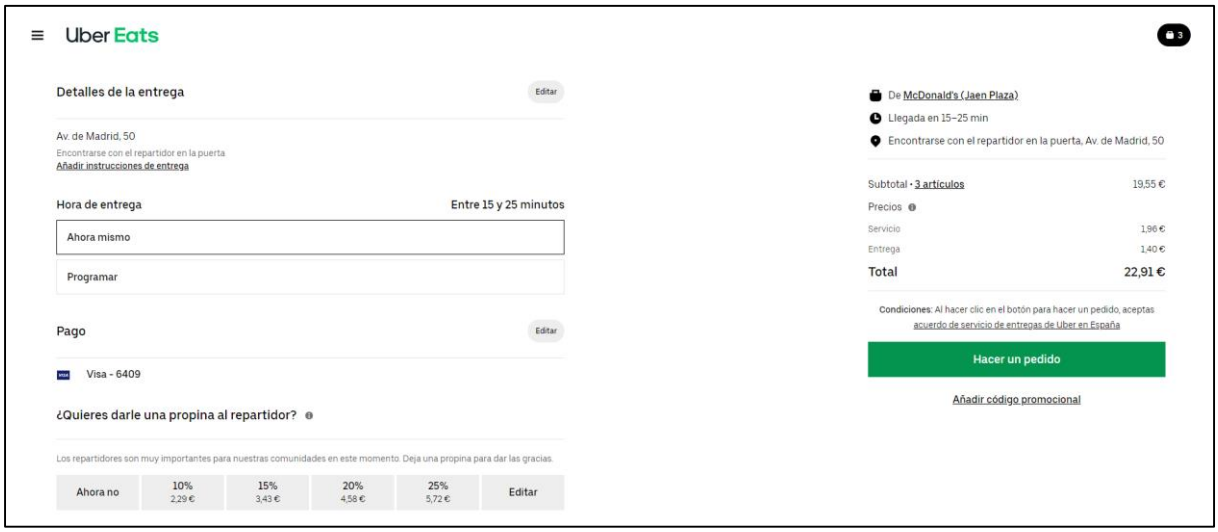


Figura 9. Pedido de Uber Eats: finalizar pedido.

Figura 10. Pedido de Uber Eats: introducción de códigos promocionales.

Tras analizar la aplicación de Uber, tenemos un gran abanico de ideas y referencias en las que basarnos; aun así, realizaremos un vistazo rápido también a la aplicación de Just Eat y Glovo.

Just Eat tiene una apariencia bastante similar a Uber: una interfaz sencilla y limpia, muy agradable a la vista, esta vez con el pedido provisional plasmado de manera perenne en la parte derecha de la pantalla. Por otra parte, Glovo presenta una interfaz mucho más atrevida, en la que predominan unos colores más llamativos, las formas redondeadas y una menor cantidad de texto en pantalla.

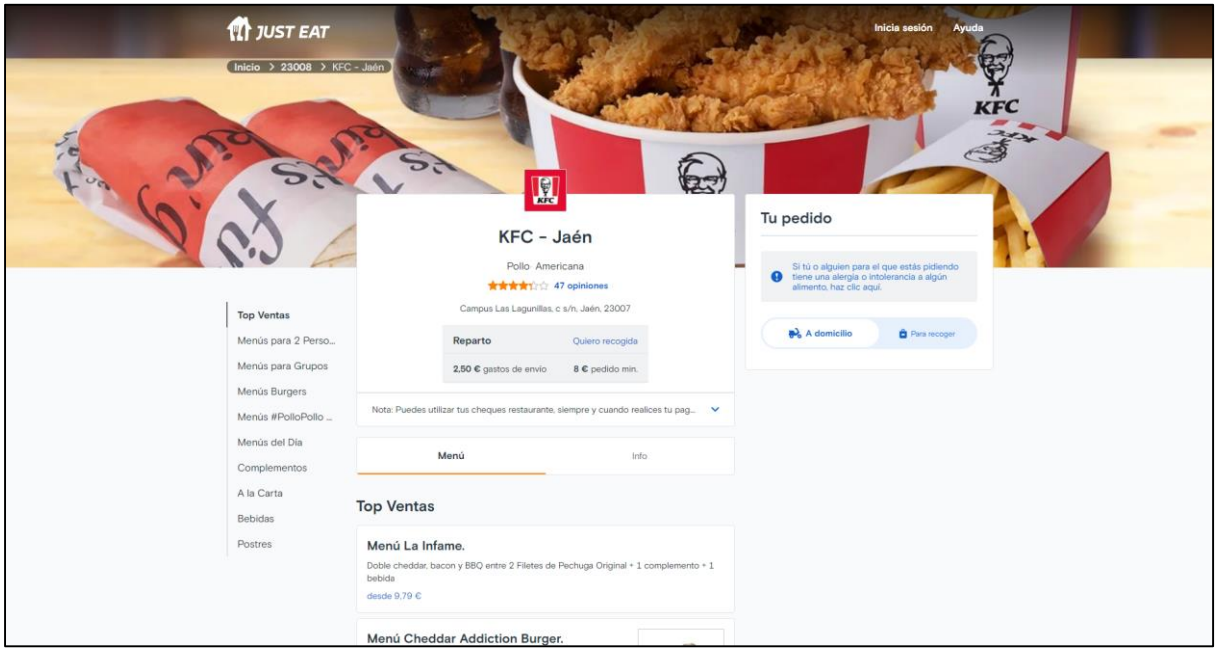


Figura 11. Pedido en Just Eat: selección de restaurante.

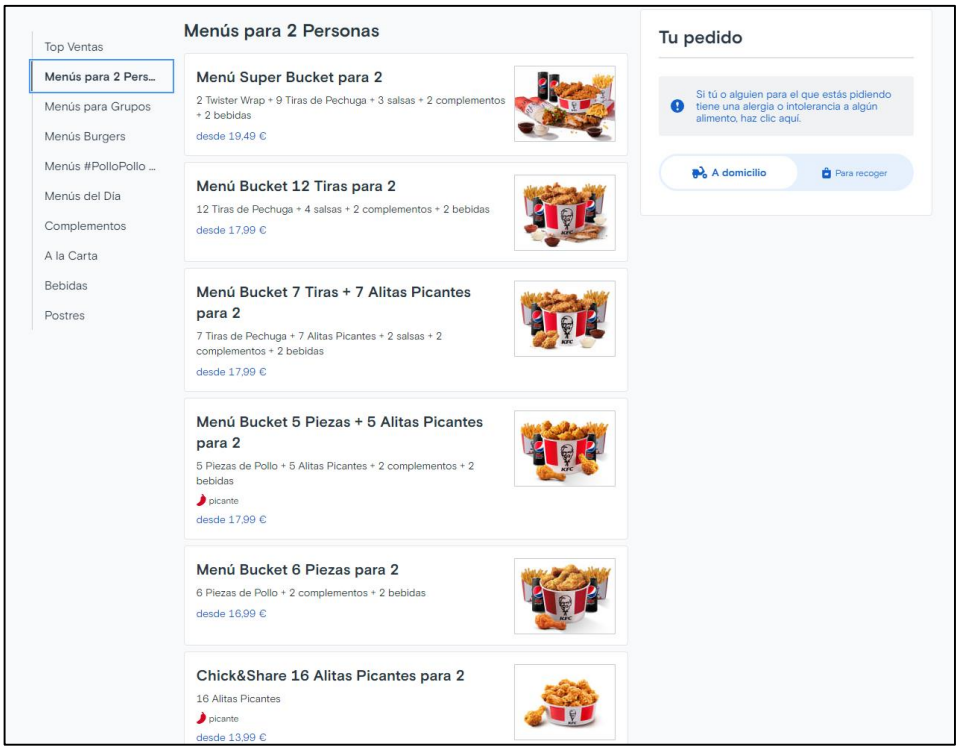


Figura 12. Pedido en Just Eat: selección de menú.

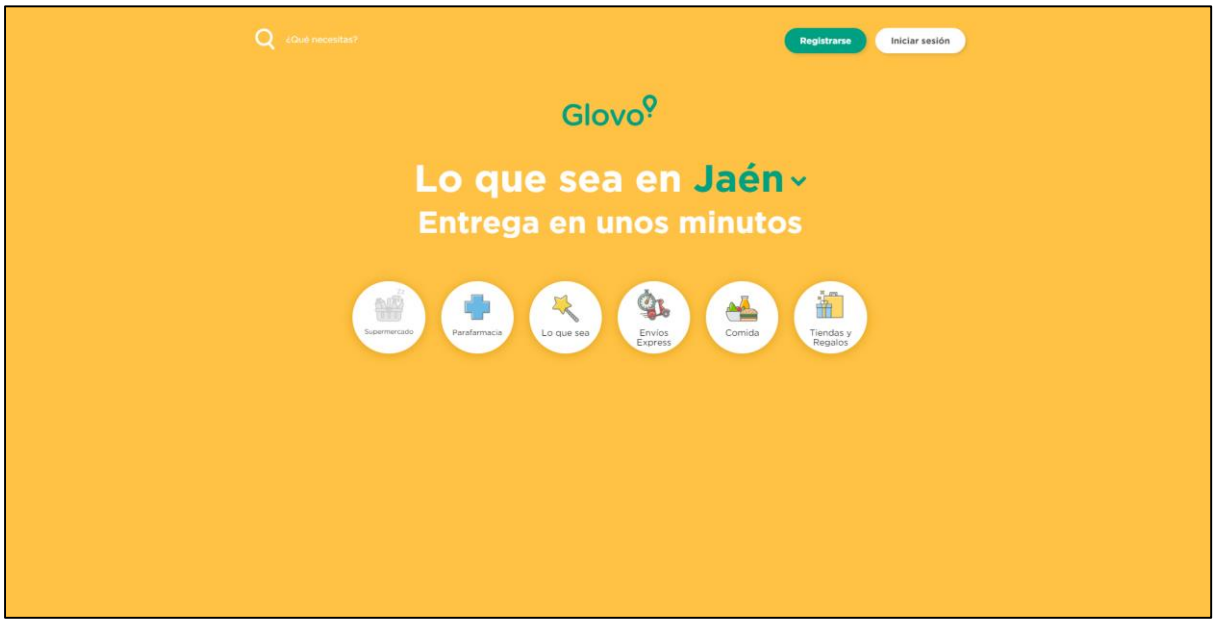


Figura 13. Página de inicio de Glovo.

Figura 14. Pedido en Glovo: selección de categoría en artículos de parafarmacia.

2.3. Estudio de posibles tecnologías

Para comenzar este apartado, debemos partir de la idea inicial de que existen dos filosofías posibles de desarrollo web: desarrollo centrado en cliente o en servidor. Dicho lo cual, descartamos la segunda ya que necesitamos una aplicación RIA (*rich Internet applications*) con interfaz intuitiva, respuesta rápida, adaptadas a dispositivos móviles, lo que lleva a centrarnos en desarrollo web en cliente con acceso a un API centralizada basada en REST como estándar de facto hoy en día para este tipo de aplicaciones.

Por tanto, con respecto a las tecnologías que se van a utilizar en el desarrollo del proyecto podemos destacar dos *frameworks* principalmente, como son Spring y Angular, los cuales vamos a ampliar a continuación.

2.3.1. Spring

En el caso del primero, Spring [5] se trata de un *framework* cuyo objetivo es simplificar el desarrollo de aplicaciones Java; este código fuente más simplificado junto con una menor dificultad en los ajustes a realizar son sus mayores ventajas. Podemos resaltar 3 pilares o principios en los que se apoya:

- Inyección de dependencias, con el uso de los componentes JavaBeans, que hacen las veces de contenedor para la transmisión de datos.
- Programación orientada a aspectos, con el objetivo de ofrecer una mayor modularidad a las aplicaciones orientadas a objetos.
- Plantillas, que hacen más sencilla la interacción con la API, haciendo que los recursos se gestionen automáticamente.

Todo esto ocurre de manera independiente a que se trate de aplicaciones web comunes o sin conexión web, con lo que estemos trabajando.

Podemos fechar su nacimiento en 2003, además de recalcar que posee una licencia de código abierto:

“En 2003 es presentado por primera vez de la mano de Rod Johnson, quien fuera su principal desarrollador, bajo una licencia basada en Apache 2.0. La idea principal es que Spring sirviera como una plataforma para desarrollar en Java de código abierto y gracias a esto su uso comenzó a extenderse hasta convertirse en el *framework* más popular para Java en un ámbito empresarial, para crear código de alto rendimiento, liviano y reutilizable.” [6]

Wikipedia nos amplía algo más la información acerca de su nacimiento e historia [7]:

“El primer gran lanzamiento fue la versión 1.0, que apareció en marzo de 2004 y fue seguida por otros hitos en septiembre de 2004 y marzo de 2005. La versión 1.2.6 de Spring Framework obtuvo reconocimientos Jolt Awards y Jax Innovation Awards en 2006. Spring Framework 2.0 fue lanzada en 2006, la versión 2.5 en noviembre de 2007, Spring 3.0 en diciembre de 2009 y Spring 3.1 dos años más tarde. El inicio del

desarrollo de la versión 4.0 fue anunciado en enero de 2013. La versión actual es la 5.1.6.”

2.3.2. Angular

Primero de todo, veamos de qué trata Angular [8]:

“Angular es un *framework opensource* desarrollado por Google para facilitar la creación y programación de aplicaciones web de una sola página, las webs SPA (*Single Page Application*).

Angular separa completamente el *frontend* y el *backend* en la aplicación, evita escribir código repetitivo y mantiene todo más ordenado gracias a su patrón MVC (Modelo-Vista-Controlador) asegurando los desarrollos con rapidez, a la vez que posibilita modificaciones y actualizaciones.” [9]

Otra de las características principales que este *framework* presenta es que es completamente modular y escalable, por lo que se puede adaptar más fácilmente a las necesidades de cualquier desarrollo. Utiliza principalmente el lenguaje de programación Typescript. También proporciona una amplia adaptabilidad con herramientas de *testing* y con Ionic, lo que les facilitará la adaptación a dispositivos móviles.

Al fin y al cabo, la elección de Angular responde a las corrientes de trabajo hoy en día: cada vez se le da menos protagonismo al modelo de trabajo en el cual el peso recae prácticamente por completo en el servidor, como sucede al trabajar con la tecnología JSF, por ejemplo.

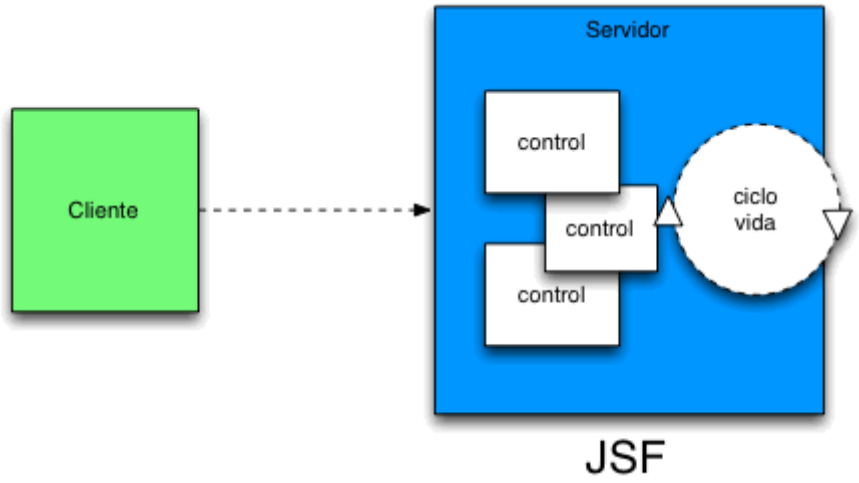


Figura 15. Esquema de funcionamiento de aplicaciones centradas en el servidor. [10]

En contraposición a lo que acabamos de ver, la tendencia actualmente es quitarle cada vez más carga y peso al protagonismo del servidor, haciendo que este peso recaiga sobre el cliente, como pasa en Angular, el *framework* elegido por nosotros para proceder al desarrollo de este proyecto.

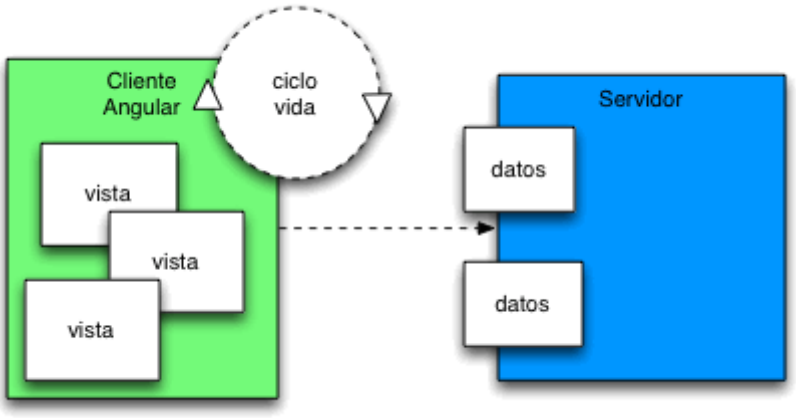


Figura 16. Esquema de funcionamiento de los frameworks MVC centrados en el cliente. [10]

Para hablar acerca de su historia, es inevitable mencionar a AngularJS, ya que es el punto de partida del proyecto que queremos tratar.

“AngularJS comenzó a ser desarrollado en 2009 por Miško Hevery originalmente era un servicio de almacenamiento online de archivos JSON donde el cobro dependía del peso en megabytes de cada archivo. Tiempo después abandonó el proyecto y relanzó angular como un proyecto *open-source*.” [11]

“AngularJS 2 fue anunciado en la ng-Europe conference 2014, causando un revuelo entre los desarrolladores ya que fue rediseñado por completo, trayendo bastantes mejoras. El 14 de septiembre de 2016, fue finalmente lanzada la primera versión oficial estable de Angular 2.” [11]

Finalmente, hay que dejar claro que en la elección del *framework*, también se han barajado otras opciones como son React o Vue, pero, ¿por qué finalmente nos hemos acabado decantando por Angular? Personalmente, no tengo demasiados conocimientos acerca de ninguno de ellos por lo que parten los tres en igualdad de condiciones; eso sí, deseo empezar por aprender el que más posibilidades ofrezca, de ahí viene la decisión.

Para empezar, es el más robusto y completo, el que nos permite hacer las aplicaciones a nivel empresarial más escalables a largo plazo, ofrece prácticamente todo lo existente posible. Esto está relacionado con el siguiente dato: tras una búsqueda en diferentes portales de empleo como son InfoJobs, Tecnoempleo o LinkedIn, el *framework* que cuenta con una mayor cantidad de ofertas de trabajo es Angular, seguido por React, y dejando en último lugar a Vue, que tiene una popularidad mucho menor a los dos nombrados anteriormente. Angular tiene un enorme apoyo y soporte por parte de la comunidad, y es cierto, que puede que sea el que posea una la curva de aprendizaje más elevada de los tres, pero esto está compensado por el resto de ventajas.

2.4. Propuesta de solución

La intención de este proyecto es diseñar y crear un sistema de un carácter bastante convencional. Se desea construir una aplicación con una apariencia vistosa, de fácil uso y adaptable a diferentes dispositivos como ordenadores, móviles o tabletas; para este cometido se recurrirá al *framework* previamente comentado, Angular.

Por otra parte, se utilizará Spring para el *backend*, mediante el cual se trabajará en crear un sistema robusto, que proporcione garantía y solvencia a la hora de trabajar con usuarios, cantidades de dinero o diferentes informaciones en relación con los pedidos. Al tratarse de un prototipo, no nos vamos a complicar en exceso a la hora de tratar la base de datos, por lo que nos vamos a apoyar en la que nos proporciona nuestro IDE[2], Netbeans, concretamente Derby.

Con respecto al cometido general de este proyecto, se desea diseñar una aplicación orientada al usuario, mediante la cual, los propios clientes del servicio podrán seleccionar artículos de un catálogo de productos de empresas asociadas de la misma ciudad para adquirirlos y recibirlos en su domicilio, en un muy corto período de tiempo.

2.5. Historias de usuario

A continuación, se pasan a presentar las historias de usuario que han sido pensadas y concebidas a priori para el desarrollo del proyecto. Como ya sabemos, estamos trabajando con una metodología iterativa que utiliza algunas técnicas ágiles empleadas en SCRUM, por lo que están sujetas a cambios y modificaciones al final de cada una de las iteraciones y tras los pertinentes análisis tanto de cómo se ha trabajado como de la evolución del prototipo.

Además de plasmar las historias de usuario en este apartado, para el desarrollo de este proyecto nos apoyaremos adicionalmente en la herramienta Pivotal Tracker[3],

[2] IDE: *Integrated Development Environment* (https://www.redhat.com/es/topics/middleware/what-is-ide)
[3] https://www.pivotaltracker.com/

que nos será de gran ayuda a la hora de gestionar tanto las historias como la planificación de los *Sprints* y nuestro progreso.

Las historias de usuario presentes a continuación han sido clasificadas atendiendo al tipo de usuario a los que afectan:

2.5.1. Cliente

HU01 - REGISTRO
Como cliente quiero registrarme para tener acceso a la aplicación y a las funcionalidades que ofrece.
Puntos de Historia: 20
Criterios de aceptación: - El correo proporcionado debe pertenecer a un dominio válido. - La contraseña proporcionada debe contar al menos con una minúscula, una mayúscula y un dígito, teniendo al menos 8 caracteres, pero no más de 16. - La contraseña debe repetirse por cuestión de seguridad. - Se debe proporcionar un número de teléfono. - Se mandará un email al usuario confirmando su registro.
Descomposición en tareas: (primera historia de usuario a realizar, empezando desde cero) - Creación del proyecto y configuración inicial de Spring. - Creación de un funcionamiento básico de iniciación en Spring. - Iniciación, configuración inicial y sincronización con el servidor de la página de aterrizaje en Angular. - Creación e iniciación de la base de datos local con la que vamos a trabajar. - Creación de primeras clases y métodos necesarios para llevar a cabo el registro en el servidor, añadiendo persistencia. - Creación de las vistas relacionadas con el proceso de registro en Angular y enlace con servidor.

HU18 - LOGIN
Como cliente quiero loguearme para acceder a los servicios que la aplicación ofrece.
Puntos de Historia: 8
Criterios de aceptación: - Se debe avisar al usuario si la configuración de entrada de teclado está en mayúsculas, a la hora de introducir la contraseña.
Descomposición en tareas: - Implementación en servicio REST del login en el servidor por parte de un cliente del sistema. - Diseño e implementación del componente de vista para identificación del usuario/login mediante el uso del framework Angular. - Implementación de funcionalidades en servicio de vista para verificación reactiva introducidos a la hora de loguearse. - Estudio de implementación de una autenticación de dos factores para ingresar a la aplicación si el usuario así lo desea para una mayor seguridad de su cuenta.

HU02 - LISTADO DE NEGOCIOS
Como cliente quiero ver el listado de negocios para decidir en cuál quiero comprar.
Puntos de Historia: 5
Criterios de aceptación: - Se debe mostrar un listado de todos los negocios disponibles en la ciudad en la que el cliente se encuentra.
Descomposición en tareas: - Implementación de la recuperación del listado de negocios desde la base de datos.

- Creación de componentes de vista para el listado e interacción con negocios mediante los cuales se mostrará al usuario el listado de dichos negocios (Incluyendo imágenes, descripción y demás detalles).

HU03 - CATÁLOGO DE PRODUCTOS
Como cliente quiero consultar el catálogo de productos de un negocio para elegir los productos que deseo adquirir.
Puntos de Historia: 8
Criterios de aceptación: - Se deben mostrar todos los productos en el catálogo de la tienda seleccionada. - Se indicará si algún producto se encuentra fuera de stock o no hay disponibilidad del mismo. - Los productos estarán clasificados según categorías. - Cada producto incluirá una descripción en la que incluya sus especificaciones, además de su precio.
Descomposición en tareas: - Implementación en servicio REST de la recuperación del listado de productos del negocio desde la base de datos. - Creación de componentes de vista para el listado e interacción con productos mediante los cuales se mostrará al usuario el listado de estos (Incluyendo imágenes, descripción y demás detalles).

HU04 - CESTA DE COMPRA
Como cliente quiero un carrito para acumular los artículos que deseo comprar.
Puntos de Historia: 5
Criterios de aceptación: - En los productos del catálogo existirá la opción “Añadir al carrito”. - Se podrá acceder al carrito en cualquier momento para visualizarlo.

- Se podrá eliminar cualquier artículo del carrito.
Descomposición en tareas: - Implementación en el servicio REST las funciones, mediante la cuales el usuario pueda gestionar su cesta de la compra, almacenando o modificando los artículos deseados. - Creación de la función Añadir al carrito disponible en todos los artículos del catálogo que tengan stock. - Implementación del componente vista para que el usuario pueda interaccionar con el propio carrito.

HU06 - INTRODUCCIÓN DE CÓDIGOS PROMOCIONALES
Como cliente quiero introducir códigos de promoción para tener descuento en mis próximos pedidos.
Puntos de Historia: 3
Criterios de aceptación: - Tras introducir el código, se indicará si es válido o en el caso de que no, se especificará el motivo (promoción caducada, cuenta no apta, etc.).
Descomposición en tareas: - Implementación en el servicio REST de las funciones necesarias para que el cliente tenga la oportunidad de introducir códigos promocionales junto a su pertinente comprobación y aplicación de la promoción. - Diseño e implementación del componente vista para llevar a cabo la tarea mencionada.

HU07 - PUNTUACIÓN DE ARTÍCULOS COMPRADOS
Como cliente quiero puntuar los artículos comprados para ayudar a futuros compradores.
Puntos de Historia: 13
Criterios de aceptación:

- El artículo recibirá una puntuación entre 1 y 5 estrellas. - El artículo recibirá una reseña por escrito de al menos 30 caracteres. - Se podrán modificar la valoración y la reseña una vez realizadas.
Descomposición en tareas: - Implementación en el servicio REST de la función necesaria para puntuar artículos una vez hayan sido adquiridos por el usuario. - Diseño e implementación mediante el framework Angular del componente vista necesario para llevar a cabo este proceso.

HU13 - INVITACIONES
Como cliente quiero invitar a otros usuarios para obtener descuento en mis próximos pedidos.
Puntos de Historia: 5
Criterios de aceptación: - La aplicación tendrá una sección específica para ello. - El sistema asignará un código aleatorio al usuario, que será el código de descuento de los futuros usuarios.
Descomposición en tareas: - Implementación en servicio REST de la función necesaria para generar un código de promoción aleatorio que pase a ser válido y pueda ser compartido con otros usuarios no registrados. - Diseño e implementación de los componentes vista necesarios para que el cliente pueda llevar a cabo esta tarea.

HU14 - COMPARTIR ARTÍCULO
Como cliente quiero compartir un artículo para mostrarlo a otros usuarios.
Puntos de Historia: 5
Criterios de aceptación:

- En la página del artículo se desplegará un menú en el que se permitirá compartir el artículo en diferentes redes sociales, generando un enlace o redirigiendo a otros sitios web/aplicaciones.
Descomposición en tareas: - Modificación del componente vista, incluyendo junto a cada artículo la función necesaria para que el cliente pueda compartir el artículo en diferentes redes sociales o apps de mensajería.

HU16 - CARTERA
Como cliente quiero añadir un método de pago para llevar a cabo el pago de los pedidos realizados.
Puntos de Historia: 5
Criterios de aceptación: - Se validará si la tarjeta de crédito es válida. - Se indicará si se trata de una tarjeta VISA o Mastercard.
Descomposición en tareas: - Implementación en servicio REST de las funciones pertinentes para el almacenamiento de tarjeta/tarjetas de crédito por parte de un usuario. - Diseño e implementación de los componentes vista mediante los cuales, el cliente pueda llevar a cabo el proceso.

HU17 - LOCALIZACIÓN ACTUAL
Como cliente quiero añadir mi localización actual para descubrir qué establecimientos prestan servicio en mi zona/localidad.
Puntos de Historia: 13
Criterios de aceptación: - Se proporcionará al usuario un formulario en el que indicará cuál es su ubicación.

Descomposición en tareas: - Implementación en el servicio REST de la función necesaria para almacenar la ubicación de un usuario y presentar los negocios que se encuentran en su ciudad. - Diseño e implementación de los componentes vista necesarios en Angular para representar dicho proceso.

2.5.2. Administrador

HU08 - MODIFICACIÓN LISTADO DE NEGOCIOS
Como administrador quiero acceder al listado de negocios para poder modificarlo.
Puntos de Historia: 20
Criterios de aceptación: - Se debe mostrar un listado de todos los negocios disponibles en la ciudad que se desee. - Será posible añadir o eliminar un negocio. - Será posible modificar la información de un negocio ya existente.
Descomposición en tareas: - Implementación en servicio REST de las funciones necesarias para completar esta historia de usuario (básicamente se trata de un CRUD en relación al listado de negocios disponible en una de las localizaciones de la empresa). - Creación de los componentes vista necesarios en Angular para representar cada acción desde el mostrado del listado de tiendas hasta el formulario al añadir un negocio nuevo a la lista.

HU09 - MODIFICACIÓN CATÁLOGO DE NEGOCIO

Como administrador quiero acceder al catálogo de cada negocio para poder modificarlo.
Puntos de Historia: 13
Criterios de aceptación: - Se deben mostrar todos los productos en el catálogo de la tienda seleccionada. - Será posible añadir o eliminar un artículo. - Será posible modificar la información de un artículo ya existente.
Descomposición en tareas: - Implementación en servicio REST de las funciones necesarias para completar esta historia de usuario (básicamente se trata de un CRUD en relación al stock de artículos de una de las tiendas asociadas a nuestra empresa). - Creación de los componentes vista necesarios para representar cada acción: desde el mostrado del listado de artículos, hasta el formulario al añadir uno nuevo a la lista.

HU15 - LISTADO DE PEDIDOS
Como administrador quiero acceder al listado de pedidos realizados para ver sus detalles y su estado.
Puntos de Historia: 8
Criterios de aceptación: - Se deben mostrar todos los pedidos realizados ordenados por fecha y hora. - Será posible filtrar la lista de pedidos realizados en función de una serie de parámetros.
Descomposición en tareas: - Implementación en servicio REST de las funciones necesarias en el servidor para completar esta historia de usuario.

- Creación del componente vista necesario en Angular para mostrar dicho listado otorgando la posibilidad al administrador de ser filtrado este según los parámetros requeridos.

2.5.3. Repartidor

[*Epic*][4] Como repartidor quiero conocer los detalles del pedido que tengo que entregar para llevarlo a cabo.

HU10 - VISUALIZACIÓN ESTABLECIMIENTO DEL PEDIDO
Como repartidor quiero conocer el establecimiento en el que se ha realizado el pedido y su localización para dirigirme hacia allí.
Puntos de Historia: 8
Criterios de aceptación: - Se mostrará en pantalla la información del negocio/establecimiento. - Se mostrará un mapa en el que estará localizado el negocio al que ir.
Descomposición en tareas: - Implementación en servicio REST de las funciones necesarias para mostrar la información del establecimiento de compra al repartidor. - Creación de los componentes vista necesarios para acompañar este proceso.

HU11 - VISUALIZACIÓN ARTÍCULOS DEL PEDIDO
Como repartidor quiero ver los productos solicitados para recogerlos del establecimiento.
Puntos de Historia: 8
Criterios de aceptación: - Se mostrará el producto o listado de productos a modo de factura.
Descomposición en tareas:

[4] https://www.scrummanager.net/bok/index.php?title=Epic

- Implementación en servicio REST de las funciones necesarias para mostrar la información de los productos que conforman el pedido al repartidor. - Creación de los componentes vista necesarios para acompañar este proceso.

HU12 - VISUALIZACIÓN DOMICILIO DEL CLIENTE
Como repartidor quiero conocer el domicilio del cliente para entregar el pedido solicitado.
Puntos de Historia: 13
Criterios de aceptación: - Se mostrará en pantalla la información del domicilio del cliente. - Se mostrará un mapa en el que estará localizado el domicilio al que ir.
Descomposición en tareas: - Implementación en servicio REST de las funciones necesarias para mostrar la información del domicilio del cliente al repartidor. - Creación de los componentes vista necesarios para acompañar este proceso.

A continuación, podemos ver una captura de pantalla de la pila de producto compuesta por todas las historias de usuario que, a priori, estarán presentes en nuestra aplicación, en la herramienta utilizada para una más ágil gestión de estas:

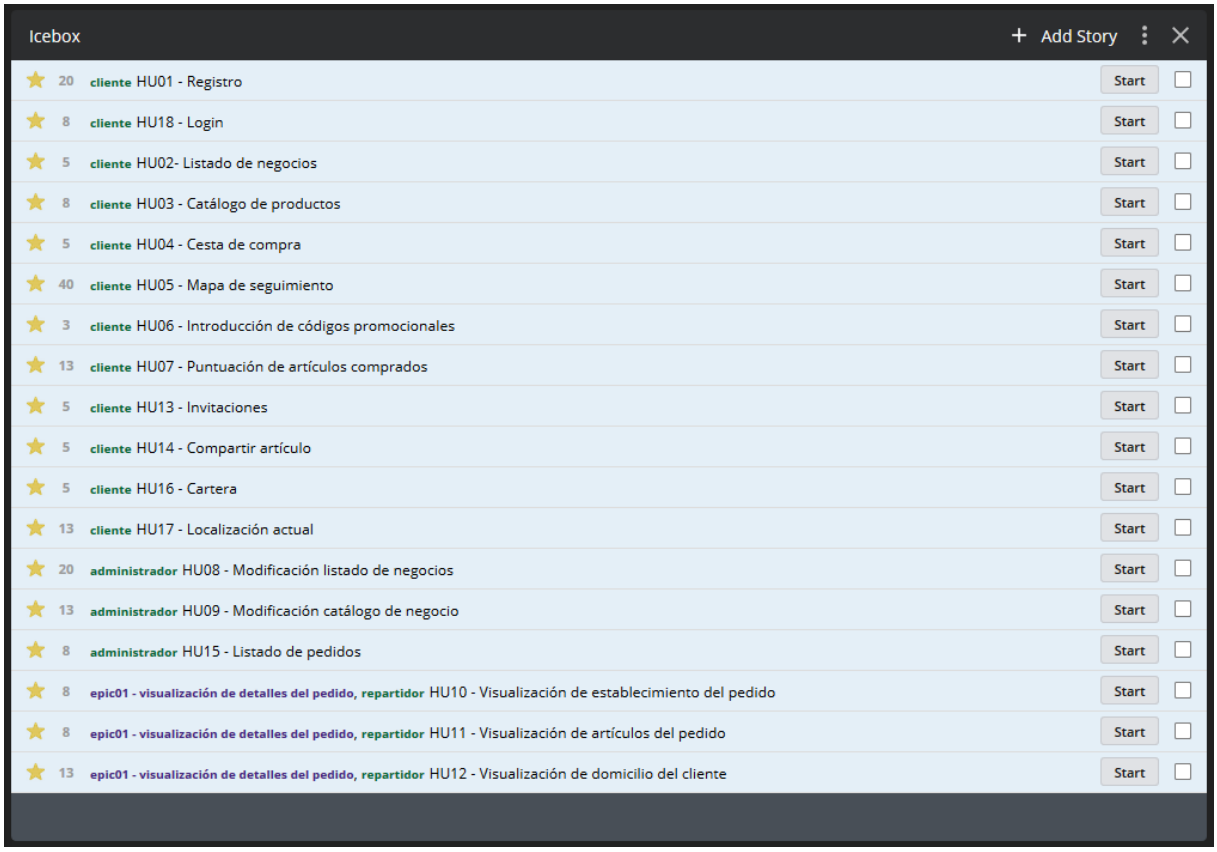


Figura 17. Product Backlog de nuestra aplicación.

2.6. Planificación inicial de tareas

Como ya se mencionó en el apartado Metodología, los *Sprints* durante el desarrollo tendrán una duración de dos semanas. Se asumirán 6 horas de trabajo diario de lunes a viernes durante las 10 semanas que durará la ejecución del proyecto posterior al pre-estudio, análisis y diseño, lo que sumará una cifra de más de 300 horas de trabajo, ya que las horas de desarrollo se suman a las ya empleadas para las preparaciones preliminares. También hay que tener en cuenta que las 300 horas repartidas en 5 iteraciones, no solamente corresponderán a la compleción de las historias de usuario, sino que también serán dedicadas a la actualización de memoria, diagramas, “reuniones” o elaboración de la presentación final; la realización de estas tareas vendrá reflejada correspondientemente en la memoria pese a no estar incluidas en la pila de producto. Por tanto, la propuesta inicial de iteraciones será de la siguiente manera.

Sprint 1	- Semana 1: 19 de abril – 23 de abril - Semana 2: 26 de abril – 30 de abril Sprint Review/Sprint Retrospective: 30 de abril
Sprint 2	- Semana 3: 3 de mayo – 7 de mayo - Semana 4: 10 de mayo – 14 de mayo Sprint Review/Sprint Retrospective: 14 de mayo
Sprint 3	- Semana 5: 17 de mayo – 21 de mayo - Semana 6: 24 de mayo – 28 de mayo Sprint Review/Sprint Retrospective: 28 de mayo
Sprint 4	- Semana 7: 31 de mayo – 4 de junio - Semana 8: 7 de junio – 11 de junio Sprint Review/Sprint Retrospective: 11 de junio
Sprint 5	- Semana 9: 14 de junio – 18 de junio - Semana 10: 21 de junio – 25 de junio Sprint Review/Sprint Retrospective: 25 de junio

Las historias de usuario, y tareas a completar en cada uno de ellos, se definirán antes de comenzar cada uno de los Sprints, ya que, debido a la naturaleza de la metodología ágil que se va a seguir, es probable y común que existan cambios y reajustes de las estimaciones durante el desarrollo; estos aspectos se estudiarán en el *Sprint Planning*, al comienzo de cada iteración.

Para continuar este apartado de planificación de tareas, recurriremos al conocido método de priorización de las mismas: MoSCoW. Es prácticamente indispensable su uso, ya que la cantidad de historias de usuario que poseemos es elevada y, por tanto, necesitamos de la priorización que MoSCoW nos proporciona para poder componer de una manera más sencilla el *Sprint Backlog* a la hora de comenzar cada una de estas iteraciones de nuestro proyecto.

Dicho lo cual, en la siguiente tabla procedemos a clasificar las 18 historias de usuario que componen el *Product Backlog*, en las 4 categorías que MoSCoW propone, atendiendo a su prioridad:

Must have	HU01 - Registro
	HU18 - Login
	HU02 - Listado de negocios
	HU03 - Catálogo de productos
	HU04 - Cesta de compra
	HU16 - Cartera
	HU17 - Localización actual
	HU08 - Modificación listado de negocios
	HU09 - Modificación catálogo de negocio
	HU15 - Listado de pedidos
	HU10 - Visualización de establecimiento del pedido
	HU11 - Visualización de artículos del pedido
	HU12 - Visualización de domicilio del cliente
Should have	HU06 - Introducción de códigos promocionales
	HU07 - Puntuación de artículos comprados
Could have	HU13 - Invitaciones
	HU14 - Compartir artículo
Won't have	HU05 - Mapa de seguimiento

Tabla 1 - Priorización MoSCoW de Historias de Usuario

2.7. Evolución de la planificación inicial

Tal y cómo hemos visto en apartados anteriores, estamos trabajando con metodologías ágiles, lo que nos aporta una flexibilidad en la planificación de nuestro proyecto que las metodologías clásicas y tradicionales no son capaz de ofrecernos.

Dicho lo cual, en el desarrollo de nuestro proyecto, ha habido algunos cambios en la planificación inicial del mismo que pasan a detallarse a continuación. En primer lugar, voy a comentar el que ha sido el mayor impedimento a la hora de desarrollar nuestro trabajo con fluidez: el aprendizaje de nuevas tecnologías. Hablaremos de ello con más detalle en el apartado final de conclusiones, pero ya podemos ir adelantando algunos aspectos. En la estimación de tiempos no se ha contado de manera correcta con el tiempo adicional que iba a ser necesario para enfrentarnos a la curva de aprendizaje del *framework* Angular, ya que nunca había trabajado con él, ni tan siquiera con el lenguaje Typescript. Con Spring es cierto que ya había trabajado en alguna ocasión, pero para refrescar, actualizar y dominar sus contenidos para abordar este trabajo también se tenía que haber contado con algo más de tiempo adicional.

Estas dificultades con las tecnologías seleccionadas se han traducido en dificultades para completar la mayoría de las historias de usuario propuestas, de modo que tras reflexionar acerca de ello en los primeros *Sprint review* y *Spring retrospective*, se optó por finalizar primero las historias de usuario que conforman las funciones CRUD en el sistema para que, valga la redundancia, este fuese capaz de presentar al menos una funcionalidad básica con la que funcionar.

Haciendo referencia al apartado anterior en el cual hemos presentado el método de priorización MoSCoW, del primer *tier* de priorización se han llevado a cabo completamente las siguientes historias:

- HU01 – Registro
- HU18 – Login
- HU02 – Listado de negocios
- HU02 – Catálogo de productos

- HU04 – Cesta de la compra
- HU08 – Modificación listado de negocios
- HU09 – Modificación catálogo de negocio

Además de estas historias, en la segunda iteración, se añadió una nueva historia de usuario no concebida en un principio, orientada a la gestión de los repartidores del sistema, la HU19.

HU19 - VISUALIZACIÓN/MODIFICACIÓN LISTADO DE REPARTIDORES
Como administrador quiero acceder al listado de repartidores de la empresa para poder consultarlo o modificarlo.
Puntos de Historia: 20
Criterios de aceptación: - Se debe mostrar un listado de todos los repartidores disponibles en la ciudad que se desee. - Será posible añadir o eliminar un repartidor. - Será posible modificar la información de un repartidor ya existente.
Descomposición en tareas: - Implementación en servicio REST de las funciones necesarias para completar esta historia de usuario (básicamente se trata de un CRUD en relación al listado de repartidores disponible en una de las localizaciones de la empresa). - Creación de los componentes vista necesarios en Angular para representar cada acción desde el mostrado del listado de repartidores hasta el formulario al añadir uno nuevo a la lista.

Esta historia también fue finalizada, de manera que el total de puntos de historia completados hasta la fecha suponían un total de 99 de los 172 que se estimaron en un inicio. Cabe resaltar que la historia HU05 - Mapa de seguimiento, que poseía un

valor total de 40 puntos de historia fue descartada completamente a lo largo del desarrollo del prototipo debido a su excesiva complejidad. Por otra parte, la estimación de los puntos de historia de la HU04 - Cesta de la compra, fue de 5 puntos; dicha cantidad fue excesivamente baja tras haberme enfrentado a ella y haberla implementado, por lo que hubo que realizar un reajuste de este peso llevando el valor hasta 20 en mi opinión.

En cuanto a las demás, la historia HU06 – Introducción de códigos promocionales se incluyó en el *Sprint Backlog* de la iteración final y empezó a implementarse, pero no acabó terminándose. Las historias pertenecientes al Epic, no se consideraron tan fundamentales y prioritarias como en un principio a la hora de llevar a cabo el prototipo, por lo que su tipo de priorización debería haber sido *Should have* en este caso.

2.8. Estudio de viabilidad

En este apartado analizaremos y desglosaremos parte por parte los gastos necesarios para afrontar el proyecto en el que nos disponemos a trabajar.

Para comenzar, veamos los gastos materiales, que se corresponden a gastos tanto de hardware como de software:

Conexión a Internet	36,95 €/mes; asumiendo una duración aproximada del desarrollo del prototipo de unos 2 meses, 73,90 €.
PC de sobremesa (incluyendo periféricos)	1.300 €; estimaremos una vida útil del producto de aproximadamente 36 meses. 1.300 €/36 meses = 36,11 €/mes. Asumiendo una duración aproximada del desarrollo del prototipo de unos 2 meses, 72,22 €.
Microsoft Office 365	Al estar trabajando una persona sola se asumirá el coste de una suscripción de carácter personal con un total de 2 meses de duración al paquete de ofimática de Microsoft.

	7,00 €/mes; asumiendo una duración aproximada del desarrollo del prototipo de unos 2 meses, 14,00 €.
Pivotal Tracker	Al estar trabajando una sola persona, nos bastará con el paquete gratuito que esta herramienta ofrece, con la limitación de un número máximo de 5 colaboradores en el proyecto. 0, 00€.
Visual Paradigm	Gracias a las licencias de software que la Universidad de Jaén, nos ofrece, el uso de esta herramienta es gratuito al tener disponible una licencia para estudiantes, que nos da la oportunidad de usar este recurso de manera ilimitada. En el caso de tratarse de un desarrollador independiente, habría que adquirir una suscripción del paquete Standard de la misma con una duración total de dos meses. 15,79 €/mes; al tratarse de dos meses, el uso de la herramienta sumaría 31,58 €.
Apache Netbeans 12.0	0,00 €
Visual Studio Code	0,00 €
SourceTree	0,00 €

A continuación, nos dispondremos a desglosar los gastos correspondientes a nóminas. Para ello tomaremos de referencia la información del portal de empleo Indeed[5], el cual nos asegura que el salario promedio de analista programador/a en España es de 27.668 € por año.

Analista programador/a	- Salario anual: 27.668 €. - Salario mensual: 2.305,66 € - Salario hora (asumiendo jornada laboral de 8 horas de lunes a viernes, 22 días, 176 horas/mes): 13,11 €.

[5] https://es.indeed.com/career/analista-programador/salaries

	- Coste de las 300 horas de trabajo que corresponden al desarrollo del prototipo: 3.931,81 €.

Para finalizar, mostraremos la suma de todos los costes para obtener el total:

Costes materiales	191,70 €
Costes personal	3.931,81 €
Total	4.123,51 €

2.9. Modelo de dominio

Para finalizar el apartado de análisis, presentamos a continuación el modelo de dominio. Precisamente como seguimos en el apartado de análisis, no entraremos demasiado en términos de diseño, que ya se estudiarán posteriormente; Pensaremos y trataremos con conceptos más cercanos al mundo real que a una hipotética implementación, clases o tipos de relaciones. Este modelo de dominio nos ayudará a vislumbrar que estructura irá teniendo la implementación de nuestra aplicación.

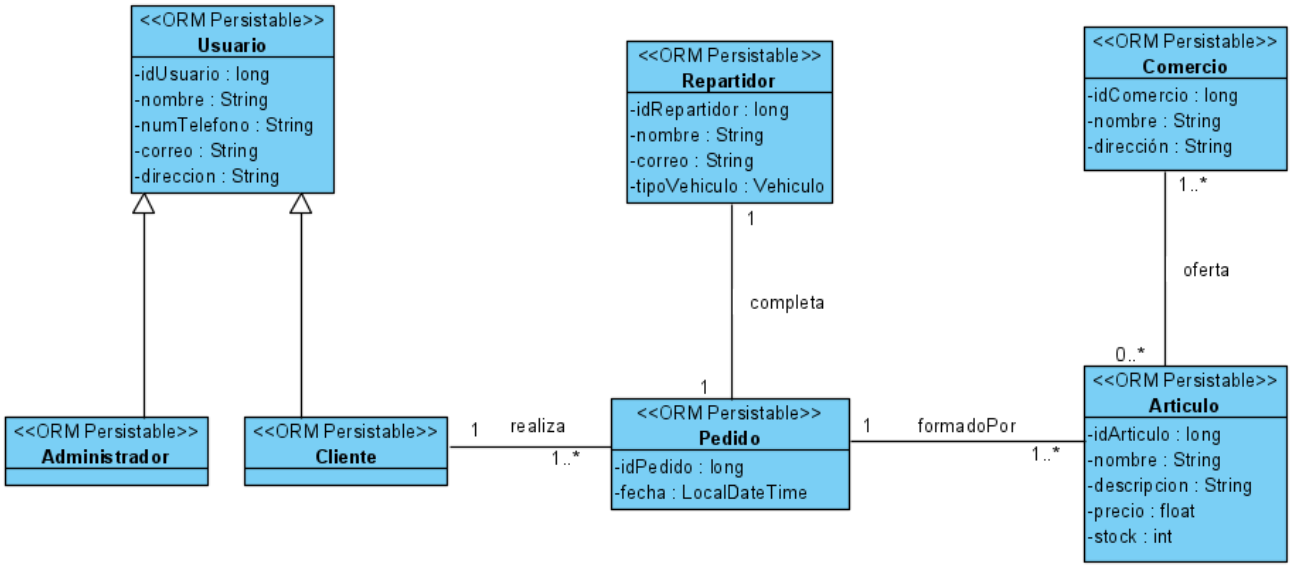


Figura 18. Modelo de dominio.

De este modelo dominio que acabamos de ver, podemos destacar algunos aspectos como que, por ejemplo, no existe una relación expresa entre Pedido y Comercio porque, en un principio, en un mismo pedido podrían ir incluidos artículos de diferentes comercios. Otro detalle, es que, en un principio, solamente se concibió la idea de que existiese la clase Usuario, que haría referencia al Cliente que se nos viene a la cabeza cuando pensamos a quien está dirigido el uso del sistema. Posteriormente, se realizó un pequeño reajuste mediante el cual se introducía una relación de generalización o herencia, que nos posibilitaría el discernir entre el Cliente al que va dirigida la aplicación, y el rol Administrador, que cobrará cierta importancia a la hora de manejar la gestión de los contenidos que se vayan añadiendo al sistema.

3. Diseño

3.1. Diagrama Entidad-Relación

Para inaugurar el apartado de diseño, comenzaremos viendo el modelo entidad-relación que ha sido generado a partir del modelo de dominio visto en el apartado anterior. A la hora de la implementación, se trabajará con el ORM[6] JPA[7], que se encargará de generar "automáticamente" las tablas, relaciones y claves necesarias para representar este esquema. Pese a ello, el modelo entidad-relación que a continuación se presenta, está normalizado en Tercera Forma Normal, basándonos en el concepto de dependencia funcional, de manera que cada atributo depende exclusivamente de la clave principal.

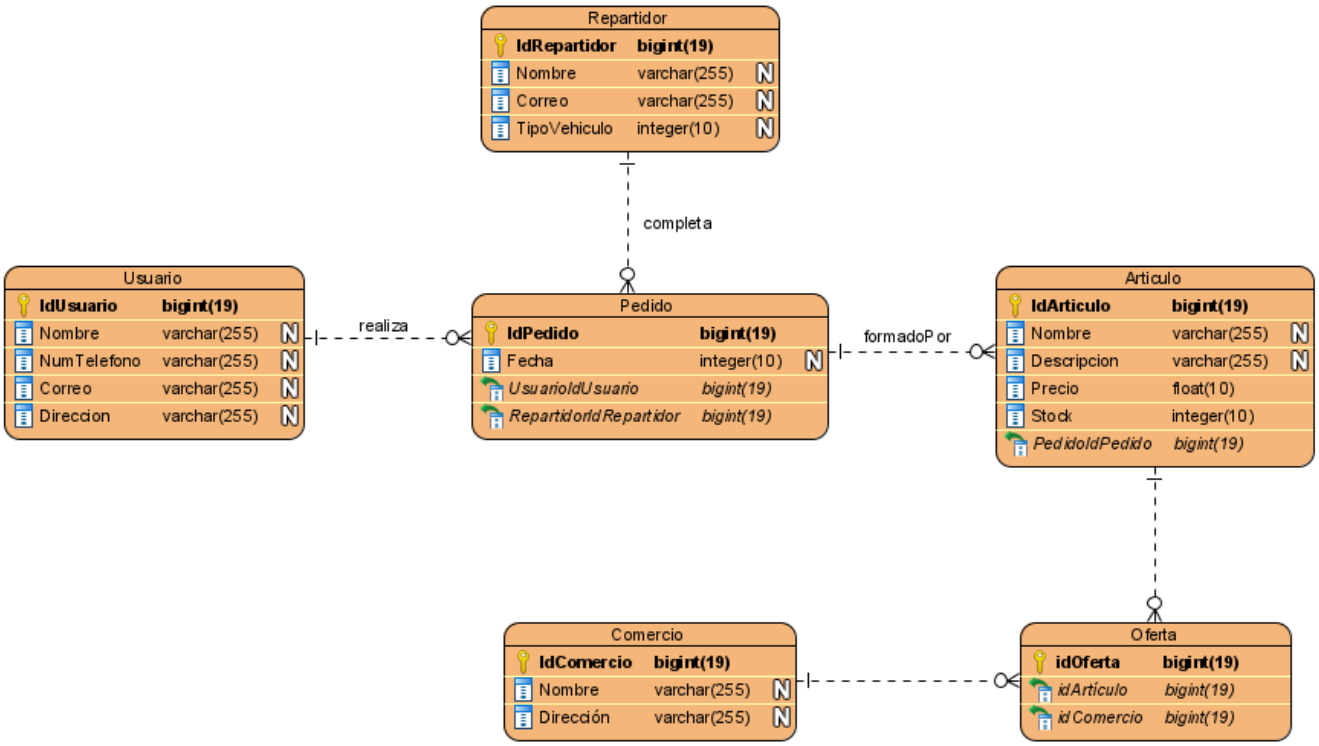


Figura 19. Diagrama entidad-relación.

[6] ORM: *Object Relational Mapping* (https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html)
[7] JPA: *Java Persistence API* (https://www.ibm.com/docs/es/was-liberty/nd?topic=overview-java-persistence-api-jpa)

3.2. Diagrama de Clases

Con respecto al diagrama de clases, al tratarse de un sistema cliente-servidor, se han separado ambas partes, dando lugar al siguiente diseño.

3.2.1. Cliente

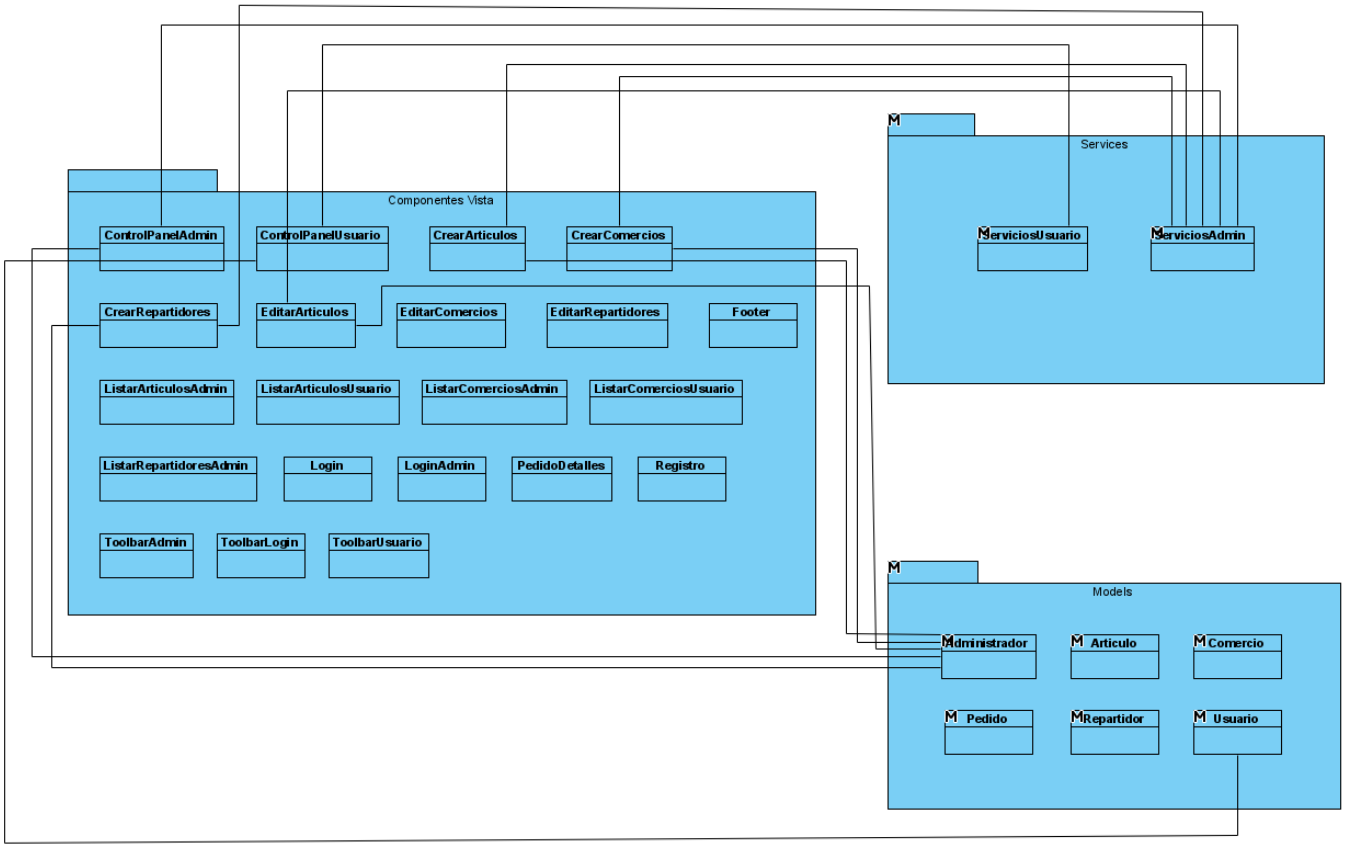


Figura 20. Diagrama de clases perteneciente al cliente, vistazo general.

Antes de entrar en el detalle del diseño propuesto, hay que aclarar que este ha sido condicionado por el modelo de desarrollo de Angular basado en componentes. ¿Y qué es un componente?

“Los componentes son el bloque de construcción más básico de un UI en una aplicación Angular. Una aplicación Angular es un árbol de componentes Angular. Los componentes Angular son un sub-conjunto de directivas. A diferencia de las directivas, los componentes siempre tienen una plantilla y solo un componente puede ser instanciado por un elemento en una plantilla.” [12]

Con respecto al diagrama presente en la figura 20, se ha representado un vistazo general de la parte perteneciente al cliente de la aplicación. En ella se puede ver que existen un total de tres paquetes: Componentes Vista, Services y Models. Solamente se han representado algunas de las relaciones para mejorar la legibilidad de dicho diagrama, por lo que las demás relaciones seguirían la misma dinámica.

Siguiendo en este lado, en la figura 21, se han detallado más concretamente los paquetes Services y Models, en los cuales se representan tanto los dos servicios que interactúan con el API REST, como las clases o entidades con las que trabajamos en el sistema.

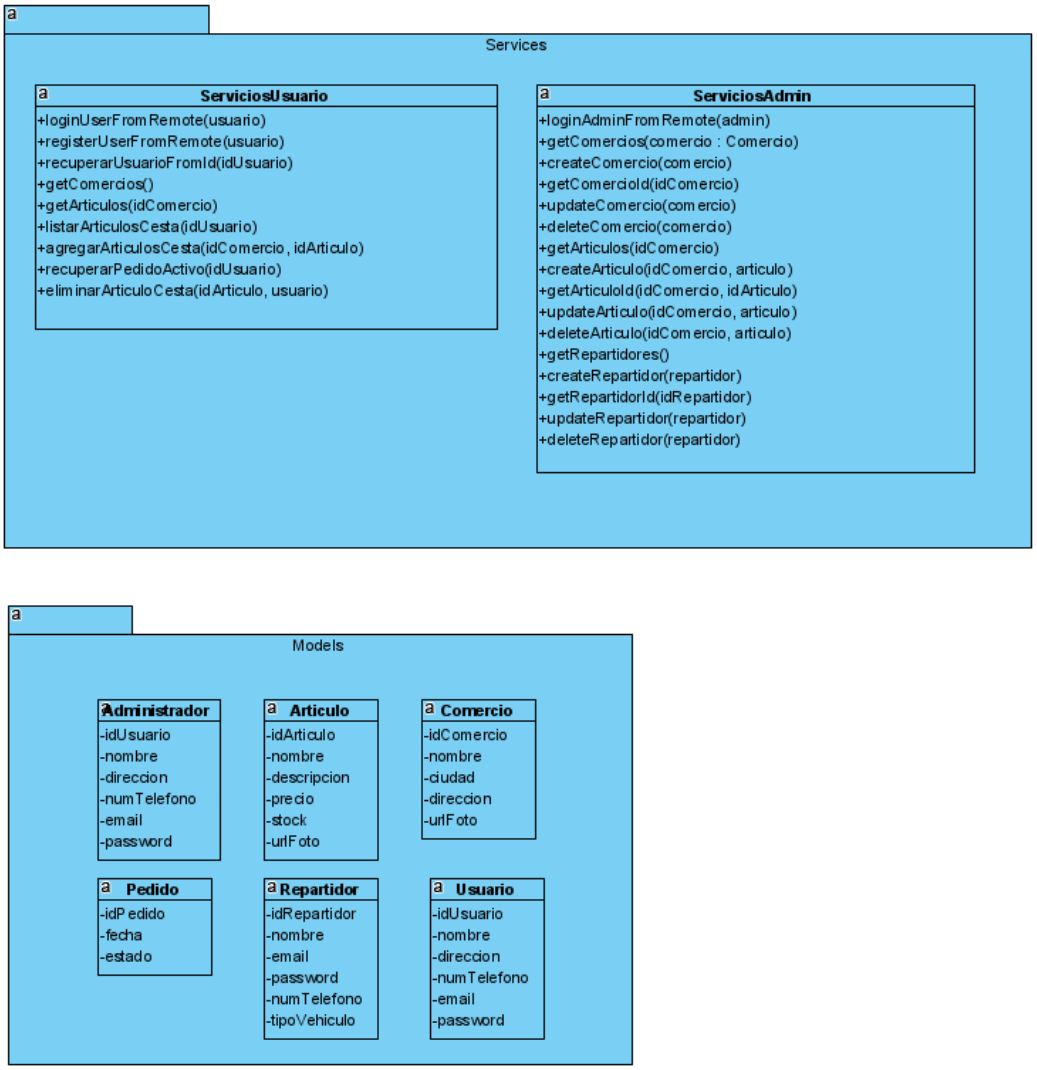


Figura 21. Diagrama de clases perteneciente al cliente, paquetes Service y Models

3.2.2. Servidor

Por otra parte, en este lado el diseño está basado en el modelo MVC clásico que establece el propio framework SpringMVC. Disponemos de un total de 4 paquetes principales como se muestra en la figura 22. El más llamativo se corresponde con el paquete Repository que, como veremos más adelante en esta memoria en el apartado de detalles de implementación, está compuesto por interfaces que heredan de JpaRepository. Ya que es la manera mediante la cual las clases del paquete Service se comunican e interactúan directamente con la base de datos. El paquete entidades es muy similar al visto en el lado del cliente, al igual que las funciones existentes en los controladores, están estrechamente relacionadas con las vistas en el paquete service de la figura 21.

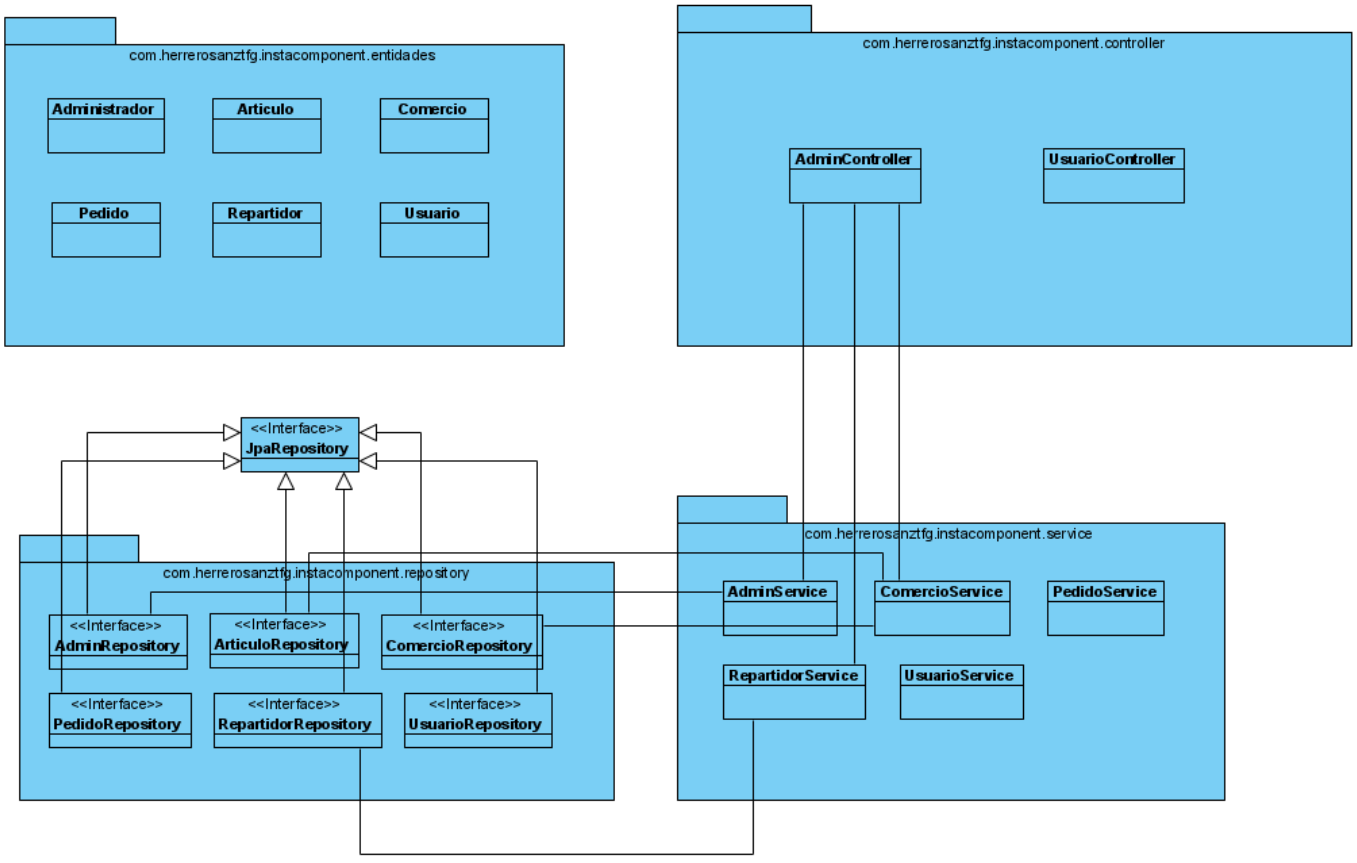


Figura 22. Diagrama de clases perteneciente al servidor.

3.3. Diagramas de secuencia

A continuación, vamos a ver los 4 diagramas de secuencia que se han realizado pertenecientes a la aplicación. Se ha decidido representar los procesos o *features* más relevantes que existen en el sistema y su funcionamiento, y que representan de una manera adecuada el flujo de información que se da gracias a la arquitectura planteada (que posteriormente será analizada con mayor profundidad).

3.3.1. Registro

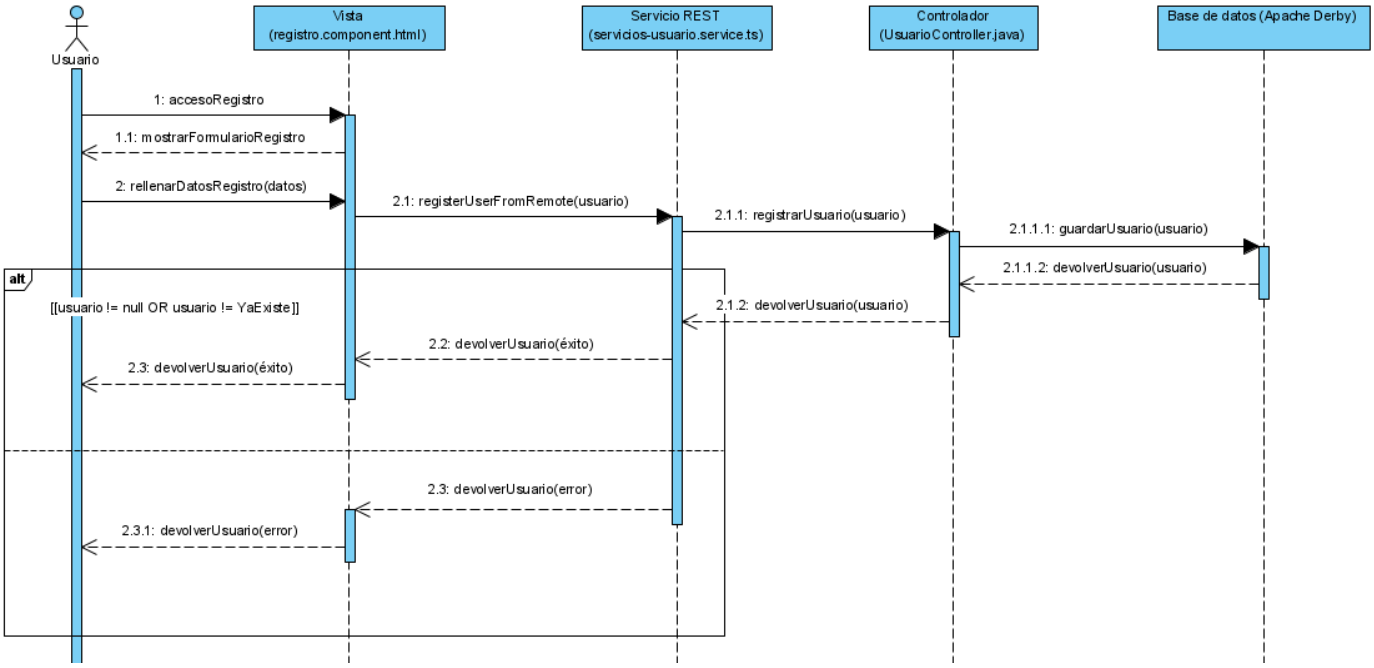


Figura 23. Diagrama de secuencia: Registro.

En este primer diagrama de secuencia podemos ver cómo se desarrollaría el registro de un usuario en nuestro sistema. El flujo parte desde el usuario, que, a través de la vista, se comunica con el servicio REST. De esta manera, y gracias a las peticiones HTTP, este servicio es capaz de establecer contacto con el controlador de nuestro servidor de Spring, el cual a su vez se comunica con la base de datos Derby para recuperar la información aquí almacenada. Al final del proceso, tendríamos un

fragmento combinado, el cual diferenciaría cuando el registro del usuario es válido de cuando no lo es (ya sea porque el usuario existe desde antes en el sistema o porque existe algún problema con los datos del mismo).

3.3.2. Login

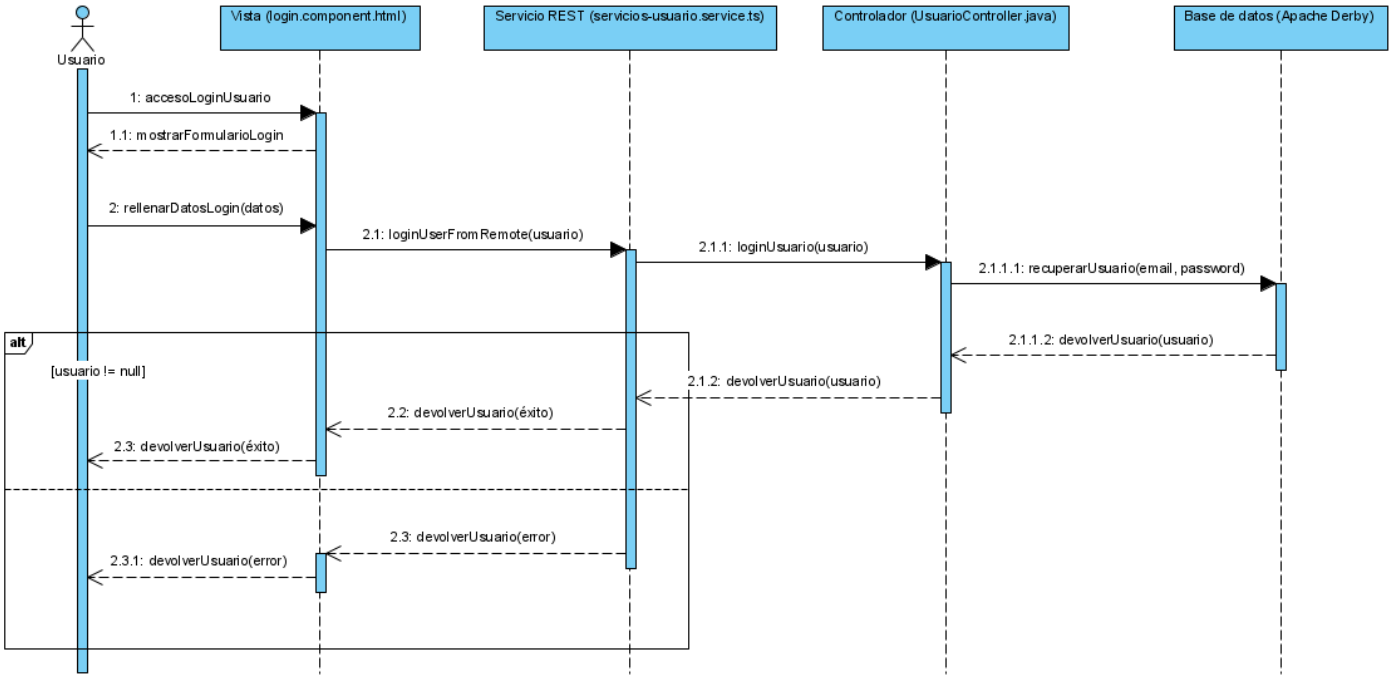


Figura 24. Diagrama de secuencia: Login.

El diagrama de secuencia que representa el logueo de un usuario en el sistema, es similar al anteriormente mostrado. Sigue el mismo proceso y de igual manera, al final, ha de comprobarse si los datos del usuario que quiere iniciar sesión son correctos, o no lo son/no existe el usuario en el sistema, lo que impediría su acceso.

3.3.3. Ver listado de artículos de un comercio

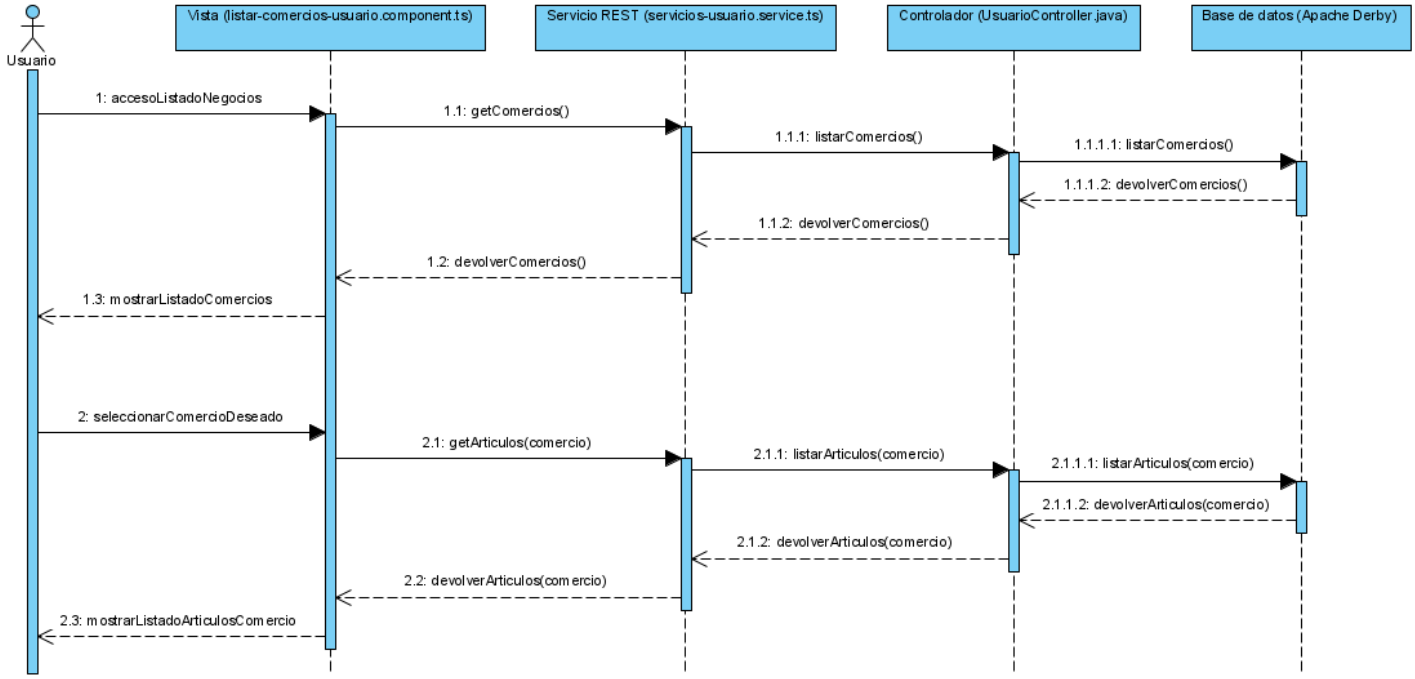


Figura 25. Diagrama de secuencia: Ver listado de artículos de un comercio.

En este diagrama de secuencia se ve representado el proceso mediante el cual un cliente puede acceder a los diferentes negocios disponibles y posteriormente consultar su stock, en el que podrá añadir los productos al carrito, añadirlos a su lista de artículos deseados, etc.

3.3.4. Modificar listado de negocios

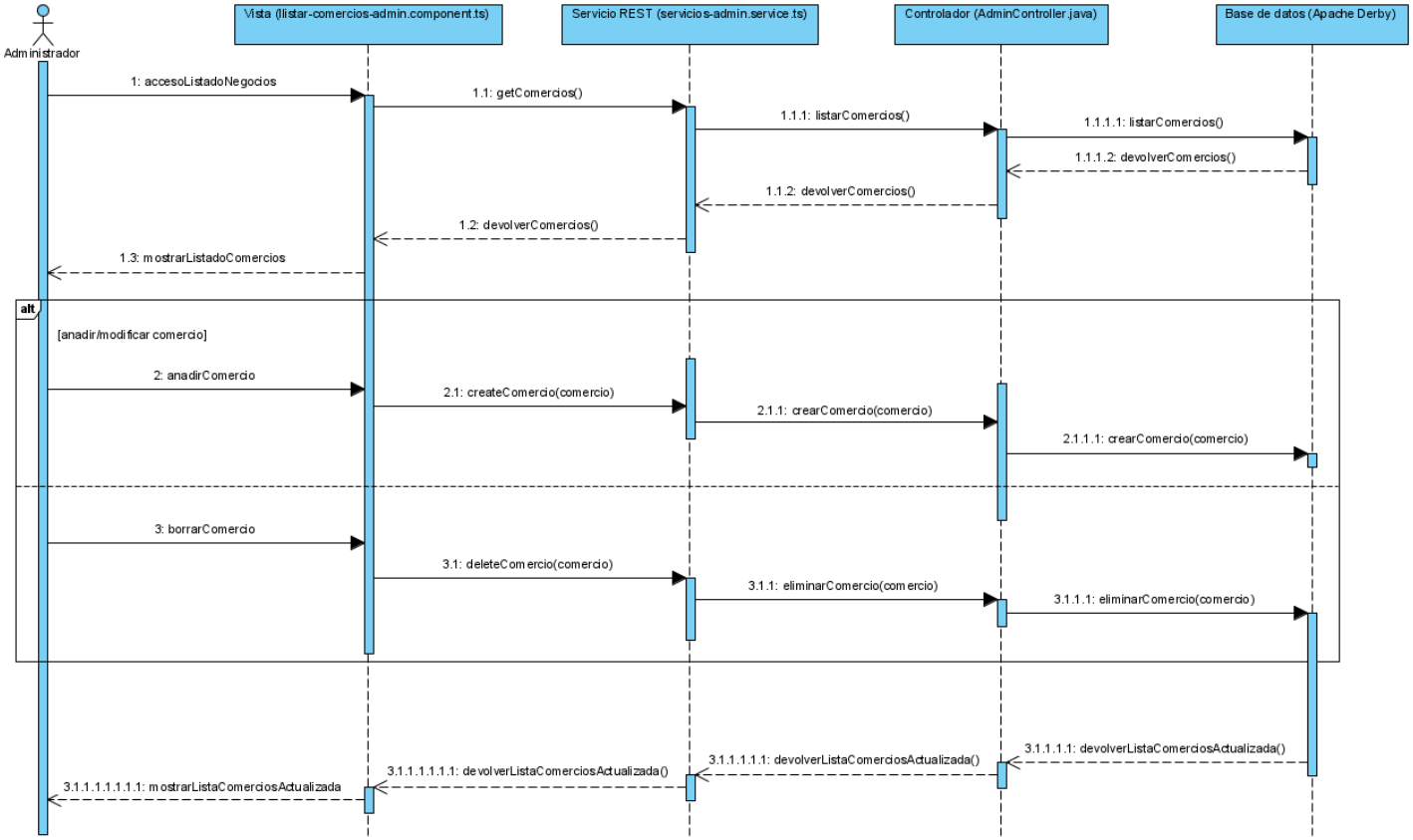


Figura 26. Diagrama de secuencia: Modificar listado de negocios.

La principal diferencia que se nos presenta en este último diagrama de secuencia es que cambia el punto de vista desde el que observamos el proceso. En este caso, no se trata del Cliente el que realiza la acción, sino del rol Administrador, ya que se representa la manera de la que éste modifica el listado de negocios existente en el sistema. Al listado de negocios se accede de la misma manera que un cliente, pero a continuación, entra en escena un fragmento combinado que diferencia cuando se quiere crear o modificar un negocio ya existente, de cuando se desea eliminar uno de estos.

3.4. Diseño de la Interfaz

En primer lugar, se debe aclarar que al tratarse de una aplicación cliente-servidor el diseño debe acometerse para la interfaz de ambas aplicaciones. Respecto a la interfaz gráfica del cliente, al no existir un cliente real no había unos requisitos específicos y por lo tanto en el prototipo se han ido desarrollando las diferentes vistas a la vez que se iban implementando las historias de usuario. Además, se utilizará el framework CSS Material Design, lo que garantizará una estética adecuada del diseño.

En el caso del API REST, el diseño sí que ha sido necesario con antelación, por lo que se detalla en el apartado siguiente. Para ello, nos hemos servido de la herramienta OpenAPI-GUI v3[8].

3.4.1. Diseño API REST

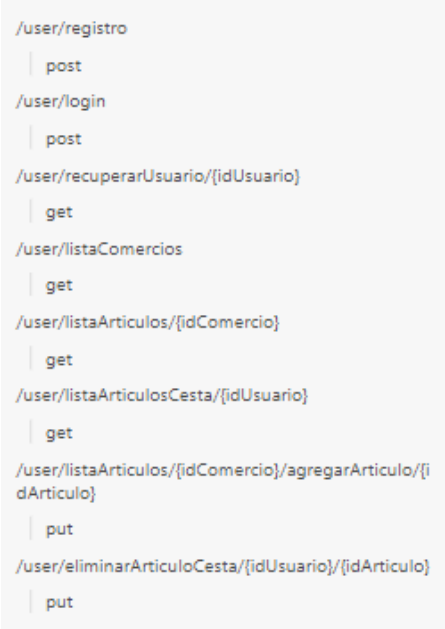


Figura 27. Visión esquemática de las distintas rutas presentes en UsuarioController.

[8] https://mermade.github.io/openapi-gui/

En la figura 27, se puede observar una visión esquemática de las URLs presentes en el controlador de Usuario. Al igual que en las figuras 28 y 29, se muestran los métodos existentes en el controlador de Usuario.

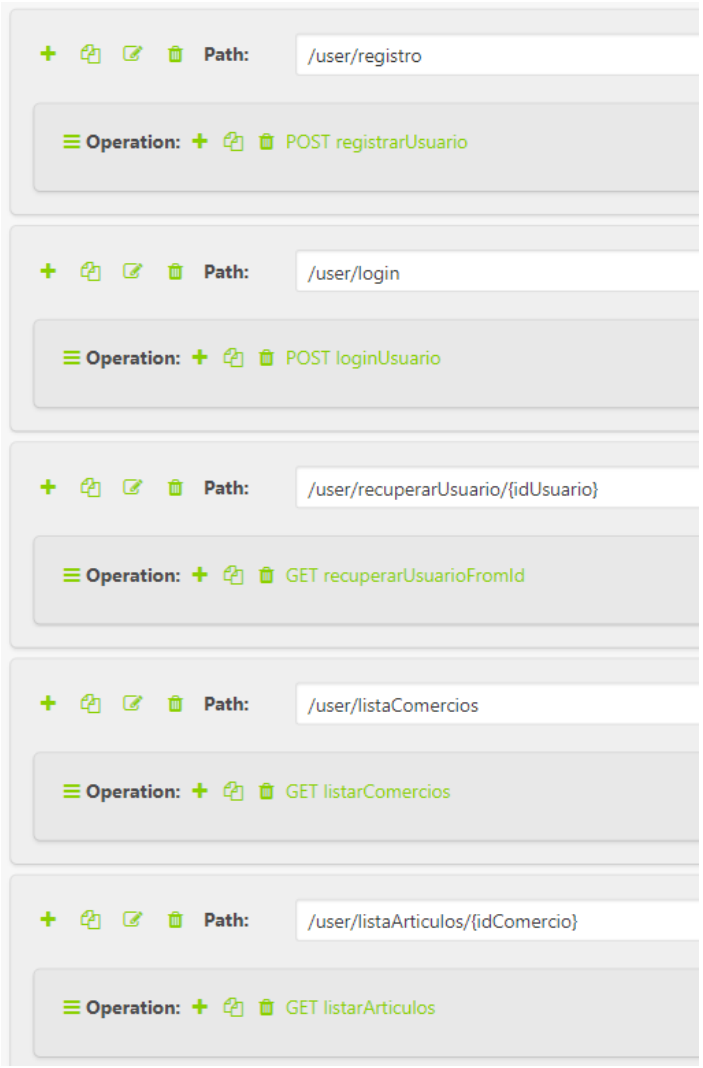


Figura 28. Operaciones API REST UsuarioController (1).

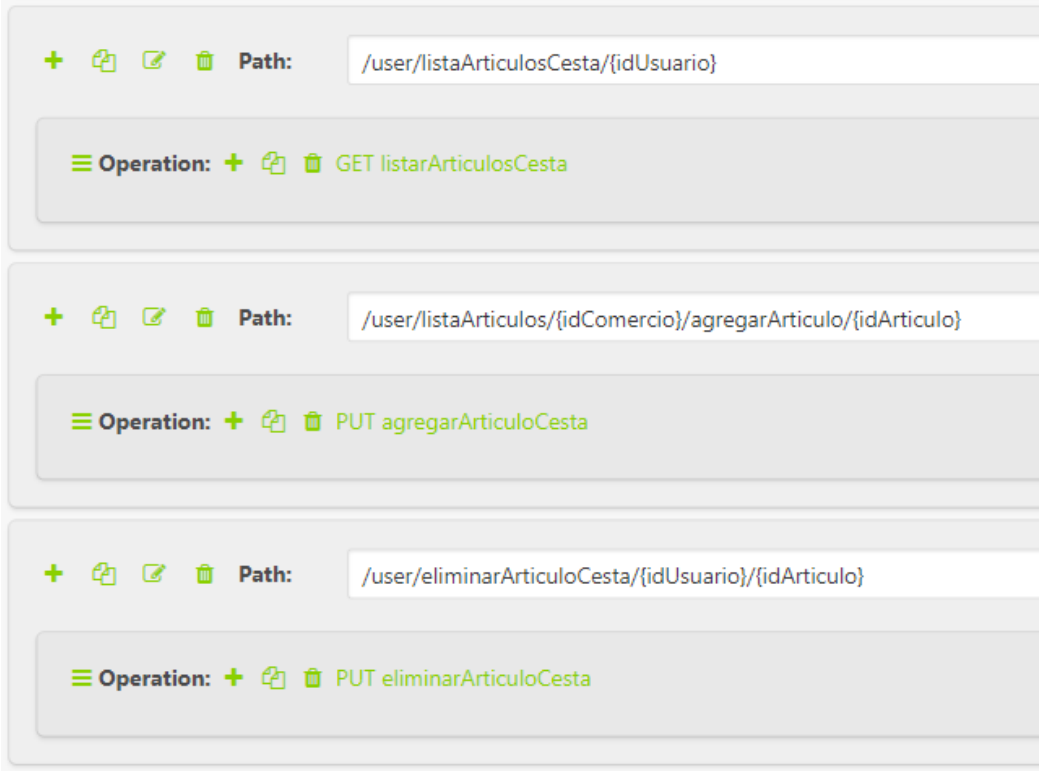


Figura 29. Operaciones API REST UsuarioController (2).

De la misma manera, en las sucesivas figuras, se muestra una visión esquemática de los verbos del controlador de Administrador, junto con los métodos de acceso HTTP incluidos en este.

Figura 30. Visión esquemática de las distintas rutas presentes en AdminController.

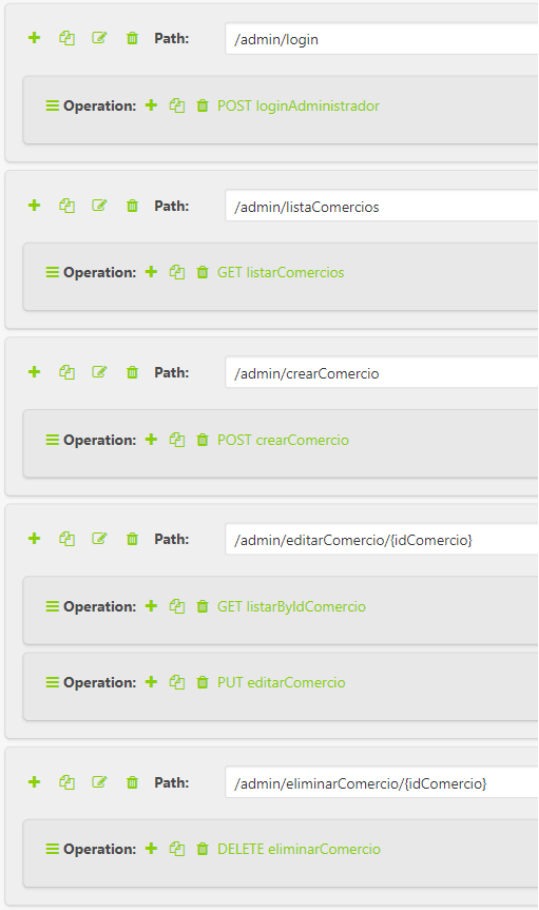


Figura 31. Operaciones API REST AdminController (1).

Figura 32. Operaciones API REST AdminController (2).

Figura 33. Operaciones API REST AdminController (3).

3.5. Plan de pruebas

Cuando se concibió este proyecto, la primera idea que se tuvo fue aplicar herramientas automatizadas para realizar *testing*, como por ejemplo JUnit[9], pero debido a los límites de tiempo, y especialmente, de conocimiento (ya que personalmente apenas había trabajado con TDD[10]), fue finalmente descartada. En su lugar, se plantearon otros dos tipos de pruebas a realizar en el proyecto.

En primer lugar, se propone un primer tipo de pruebas de caja negra sobre el API REST que se irán poniendo en práctica durante la implementación del prototipo. Estas consisten en el uso de la herramienta Postman[11]. Mediante Postman podremos comprobar tanto que las peticiones REST con datos válidos funcionan de manera correcta, como que las que tengan algún dato inválido devuelven los códigos de error correspondientes.

Por otra parte, también realizaremos pruebas sobre la interfaz gráfica continuamente durante la implementación, ya que mediante estas pruebas manuales podremos comprobar que se cumplen los criterios de aceptación de cada una de las historias de usuario planeadas. De la misma manera que antes, comprobaremos tanto el comportamiento válido esperado como las respuestas frente a posibles errores, como pueden ser las entradas erróneas de datos en formularios.

[9] https://junit.org/junit5/
[10] TDD: *Test Drive Development* (https://bit.ly/3jQy164)
[11] https://www.postman.com/

4. Implementación

A continuación, vamos a observar los detalles correspondientes a la implementación de nuestro sistema. Por una parte, estudiaremos la arquitectura que conforma el proyecto ayudándonos de algunos esquemas, lo que nos ayudará a obtener una visión más clara del mismo; y, por otra parte, veremos ejemplos y detalles concretos de código, que nos ayuden a entender cómo ha sido puesto en marcha el funcionamiento de la aplicación.

4.1. Arquitectura

Figura 34. Vista general de las tecnologías que conforman la arquitectura de nuestro prototipo: Spring Boot, Angular y Apache Derby.

En este punto realmente, no se presentan excesivas novedades, ya que en la fase de análisis se habló de una manera extensa acerca de las tecnologías con las cuales íbamos a trabajar: Spring Boot a la hora de trabajar con el servidor, el *framework* Angular para desarrollar el cliente, y la base de datos Apache Derby que nos proporciona nuestro IDE Netbeans. De igual manera, en los diagramas de secuencia, ya se iba dejando ver cuál era el flujo de trabajo a la hora de que los componentes del sistema interaccionasen entre sí; no obstante, vamos a observarlo con más detalle.

Nuestro sistema se divide en tres partes principales: el cliente web, que es el elemento que permite al usuario interactuar con el sistema y comunicarse con el

servidor; el propio servidor, que sirve de enlace entre el cliente web y nuestra base de datos; y esta última cuya función, como ya sabemos es almacenar toda la información que posibilita el funcionamiento de la aplicación.

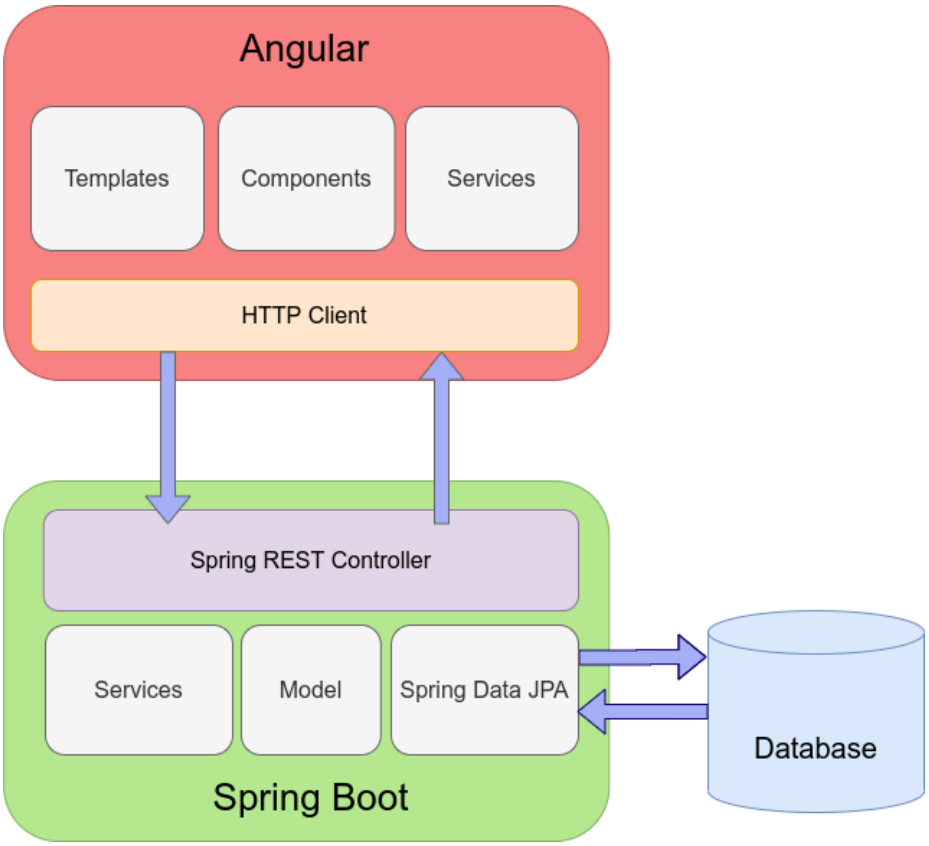


Figura 35. Diagrama arquitectónico del prototipo de aplicación desarrollado. **[13]**

En este primer diagrama podemos ver de una manera muy esquemática y clara cómo se distribuyen los elementos del sistema. Con respecto al cliente web, se pueden destacar los componentes y los servicios, que son las piezas fundamentales sobre las que recae el correcto funcionamiento del *framework* Angular. A continuación, entraría en escena el protocolo HTTP, que veremos con más detalle en el siguiente apartado, pero que ya se puede ir observando que actúa de enlace con el controlador REST. Finalmente, en Spring Boot poseemos modelos, que son los cuales nos ayudan

a manejar los objetos de negocio; servicios, que nos ayudarán a trabajar con la base de datos; y finalmente el componente fundamental JPA, que es de vital importancia a la hora de realizar el mapeo de las clases, relaciones y posterior almacenamiento de los datos.

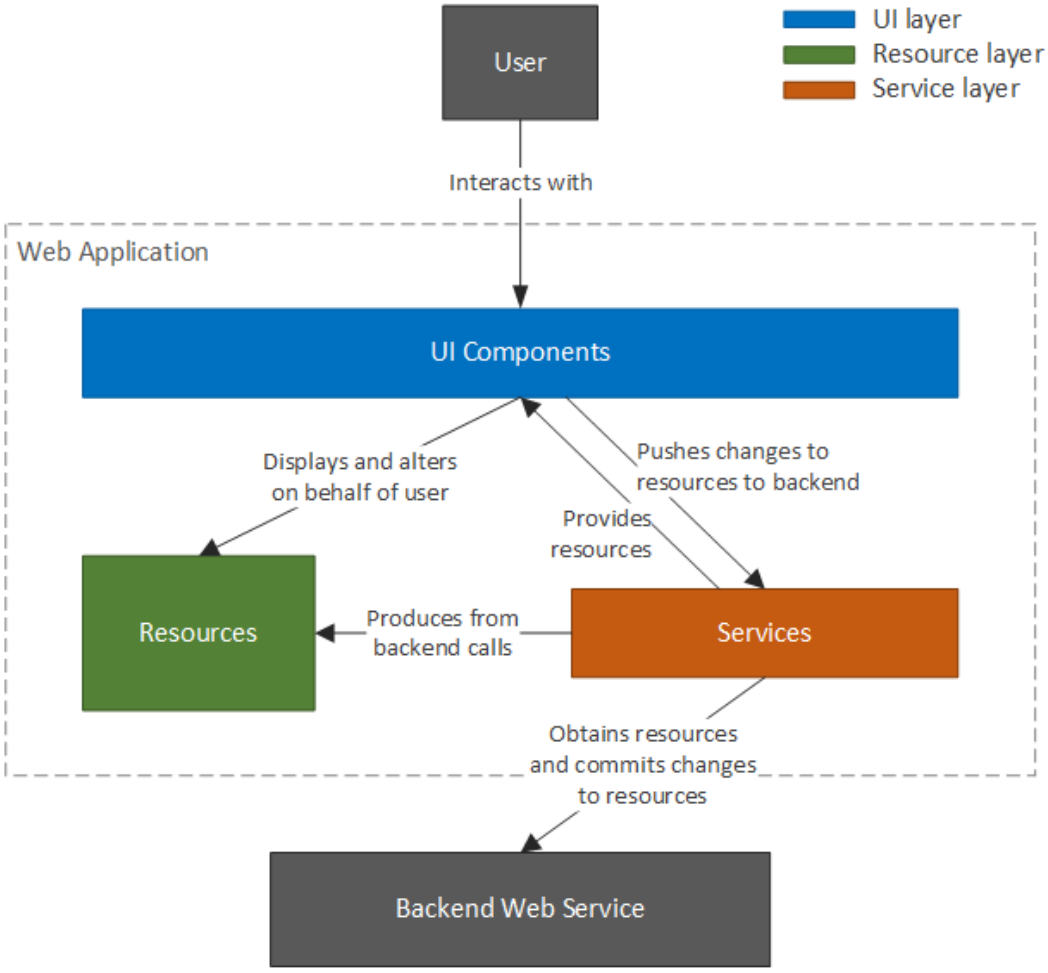


Figura 36. Diagrama arquitectónico perteneciente al cliente web. [12]

Para finalizar, aquí podemos ver un diagrama que hace referencia solamente a la parte del cliente web, y que nos ayuda a ver con algo más de detalle cómo actúan entre sí, los elementos que pertenecen a este “paquete”.

[12] https://bit.ly/3BOjtdb

4.2. Detalles sobre implementación

4.2.1. Servidor Spring Boot

Para comenzar este apartado trataremos el primer aspecto que tuvimos que abordar a la hora de poner en marcha nuestra aplicación, que fue habilitar las peticiones CORS[13]. De esta manera, las peticiones HTTP podrían funcionar sin ningún tipo de inconveniente a la hora de acceder a los recursos del servidor, por ejemplo, desde el navegador web de nuestro equipo, lo cual, por cuestiones de seguridad, permanece deshabilitado por defecto.

WebConfig.java

Source History

```java
package com.herrerosanztfg.instacomponent;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

/* Fichero necesario para habilitar las peticiones CORS hacia cualquier destino */

@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**");
    }
}
```

Figura 37. Habilitación de las peticiones CORS, fichero WebConfig.

```java
@RestController
@CrossOrigin(origins = "*", methods = {RequestMethod.GET, RequestMethod.POST, RequestMethod.PUT, RequestMethod.DELETE})
@RequestMapping("/user")
public class UsuarioController {
```

Figura 38. Habilitación de las peticiones CORS, anotación en controlador.

Para el correcto funcionamiento de la aplicación se han realizado dos modificaciones. Por una parte, se ha incluido en el proyecto el fichero WebConfig, pero

[13] CORS: Intercambio de recursos de origen cruzado (https://developer.mozilla.org/es/docs/Web/HTTP/CORS)

como aún con este seguían existiendo problemas al trabajar con peticiones PUT y DELETE, se añadió la anotación @CrossOrigins en los controladores de la aplicación para asegurarnos un funcionamiento sin problemas de las mismas.

En la figura 39, podemos observar unas de las piezas fundamentales de Spring, que es el uso de la anotación @Autowired, para producir la inyección de dependencias de las correspondientes clases.

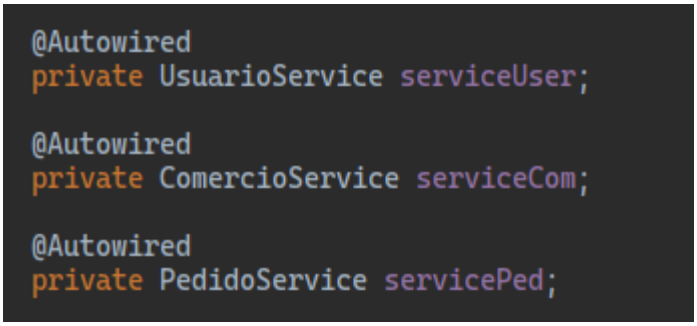


Figura 39. Inyección de dependencias en los controladores.

A continuación en la figura 40, vamos a ver de una manera más concreta y específica que antes, como ha sido implementado, por ejemplo, el controlador de Usuario, concretamente las funciones que hacen referencia a la gestión del carrito de la compra. Podemos observar cómo mediante una petición *Get* se obtienen los artículos de la cesta; cómo mediante una petición *Put*, el estado de la cesta es actualizado, siendo agregado a ella un nuevo producto; o como finalmente, mediante otra petición *Put*, se actualiza el estado de la cesta de nuevo, pero en este caso, para eliminar un artículo de ella.

```java
/* Gestión de la cesta de compra */

@GetMapping("/listaArticulosCesta/{idUsuario}")
public List<Articulo> listarArticulosCesta(@PathVariable("idUsuario") long idUsuario) {
    Usuario usuarioObj = serviceUser.listarByIdUsuario(idUsuario);
    Pedido pedidoObj = usuarioObj.getPedidosRealizados().get(usuarioObj.getPedidosRealizados().size() - 1);
    long idPedido = pedidoObj.getIdPedido();

    return servicePed.listarArticulosCesta(idPedido);
}

@PutMapping("/listaArticulos/{idComercio}/agregarArticulo/{idArticulo}")
public Articulo agregarArticuloCesta(@PathVariable("idComercio") long idComercio, @PathVariable("idArticulo") long idArticulo, @RequestBody Usuario usuario) {

    String tempEmail = usuario.getEmail();
    Usuario usuarioObj = serviceUser.recuperarUsuarioByEmail(tempEmail);
    Articulo articuloObj = serviceCom.listarByIdArticulo(idArticulo);

    Pedido pedidoObj = usuarioObj.getPedidosRealizados().get(usuarioObj.getPedidosRealizados().size() - 1);

    articuloObj.agregarPedidoPerteneciente(pedidoObj);
    return serviceCom.agregarArticuloCarrito(articuloObj);
}

@PutMapping("/eliminarArticuloCesta/{idUsuario}/{idArticulo}")
public Articulo eliminarArticuloCesta(@PathVariable("idUsuario") long idUsuario, @PathVariable("idArticulo") long idArticulo, @RequestBody Usuario usuario) {

    String tempEmail = usuario.getEmail();
    Usuario usuarioObj = serviceUser.recuperarUsuarioByEmail(tempEmail);
    Articulo articuloObj = serviceCom.listarByIdArticulo(idArticulo);

    Pedido pedidoObj = usuarioObj.getPedidosRealizados().get(usuarioObj.getPedidosRealizados().size() - 1);

    articuloObj.eliminarPedidoPerteneciente(pedidoObj);
    return serviceCom.eliminarArticuloCarrito(articuloObj);
}
```

Figura 40. Gestión cesta de la compra, controlador de usuario.

En la figura 41, podemos ver otra de las piezas fundamentales de la aplicación en *Spring*, y es como gracias a JPA, una clase es mapeada automáticamente de manera que se verá reflejada en la base de datos: tanto los atributos que conforman la misma, como la relación en este caso existente. Es imprescindible la anotación *@Entity*, mediante la cual indicamos que esta clase será una entidad, al igual que la anotación *@Id*, que nos indica cuál será la clave primaria de la tabla. Se ha optado por una estrategia de generación automática, la cual irá otorgando valores correlativos a los comercios, en este caso, que se irán añadiendo. También se han omitido algunas otras anotaciones como *@Column*, que cambiaría el nombre de la columna de la tabla, debido a que no queremos que sea diferente del nombre del atributo. Finalmente, en esta clase existe una relación con la clase Articulo del tipo *@OneToMany*, la cual está definida del tipo *EAGER*, ya que al tratarse de un prototipo y no tener una amplia cantidad de datos, no es necesario trabajar con una carga perezosa, la cual se definiría como *LAZY*.

```java
@Entity
public class Comercio {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long idComercio;
    private String nombre;
    private String ciudad;
    private String direccion;
    private String urlFoto;

    @OneToMany(mappedBy = "comercioOfertante", fetch = FetchType.EAGER)
    @JsonIgnoreProperties("comercioOfertante")
    List<Articulo> articulosOfertados;
```

Figura 41. Mapeo de la clase comercio.

Finalmente, con lo que a la parte del servidor respecta, en la figuras 42 y 43 se muestran el servicio Usuario, definido con la anotación *@Service*, que a su vez está relacionado con el repositorio Usuario, el cuál es una interfaz capaz de interaccionar directamente con la base de datos ya que hereda de *JpaRepository*.

```java
@Service
public class UsuarioService {

    @Autowired
    private UsuarioRepository repoUser;

    @Autowired
    private PedidoRepository repoPed;

    public Usuario guardarUsuario(Usuario usuario) {
        return repoUser.save(usuario);
    }

    public Usuario recuperarUsuarioByEmail(String email) {
        return repoUser.findByEmail(email);
    }

    public Usuario recuperarUsuarioByEmailAndPassword(String email, String password) {
        return repoUser.findByEmailAndPassword(email, password);
    }

    public void crearNuevoPedido(Pedido pedido){
        repoPed.save(pedido);
    }
```

Figura 42. Servicio Usuario.

```java
public interface UsuarioRepository extends JpaRepository<Usuario, Long> {

    public Usuario findByEmail(String email);

    public Usuario findByEmailAndPassword(String email, String password);

    public Usuario findByIdUsuario(long idUsuario);
}
```

Figura 43. Repositorio Usuario.

4.2.2. Cliente web Angular

Para comenzar la sección que al cliente web respecta, vamos a hablar del uso que le hemos dado a Angular Material[14]. Angular Material es una librería de estilos similar a la conocida Bootstrap[15], pero al estar desarrollada por el propio Angular, se encuentra más centrada en un estilo *material design*[16] además de contar con una perfecta integración con el propio *framework*. Todas las dependencias necesarias para el uso de esta librería han sido extraídas a un solo fichero como se observa en la figura 44.

angular-material.module.ts

src > app > angular-material.module.ts > materialModules

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { PortalModule } from '@angular/cdk/portal';
import { MatAutocompleteModule } from '@angular/material/autocomplete';
import { MatButtonModule } from '@angular/material/button';
import { MatButtonToggleModule } from '@angular/material/button-toggle';
import { MatCardModule } from '@angular/material/card';
import { MatCheckboxModule } from '@angular/material/checkbox';
import { MatChipsModule } from '@angular/material/chips';
import { MatRippleModule } from '@angular/material/core';
import { MatDividerModule } from '@angular/material/divider';
import { MatExpansionModule } from '@angular/material/expansion';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatIconModule } from '@angular/material/icon';
import { MatInputModule } from '@angular/material/input';
import { MatListModule } from '@angular/material/list';
import { MatMenuModule } from '@angular/material/menu';
import { MatPaginatorModule } from '@angular/material/paginator';
import { MatProgressSpinnerModule } from '@angular/material/progress-spinner';
import { MatSelectModule } from '@angular/material/select';
import { MatSidenavModule } from '@angular/material/sidenav';
import { MatSnackBarModule } from '@angular/material/snack-bar';
import { MatSortModule } from '@angular/material/sort';
import { MatTableModule } from '@angular/material/table';
import { MatTabsModule } from '@angular/material/tabs';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatTreeModule } from '@angular/material/tree';
import { MatBadgeModule } from '@angular/material/badge';
import { MatGridListModule } from '@angular/material/grid-list';
import { MatRadioModule } from '@angular/material/radio';
import { MatDatepickerModule } from '@angular/material/datepicker';
import { MatTooltipModule } from '@angular/material/tooltip';
```

Figura 44. Dependencias de Angular Material.

[14] https://material.angular.io/
[15] https://getbootstrap.com/
[16] https://material.io/design

Para continuar, en la figura 45 tenemos una sección del servicio REST ServiciosUsuario, en la cual podemos ver algunas de las funciones mediante las cuales el cliente web establece conexión con el servidor de Spring, para poder, entre otras, acceder al sistema mediante logueo, registrarse en la aplicación u obtener los comercios o artículos existentes en la base de datos, tanto pertenecientes a los propios comercios o a la cesta de la compra del usuario.

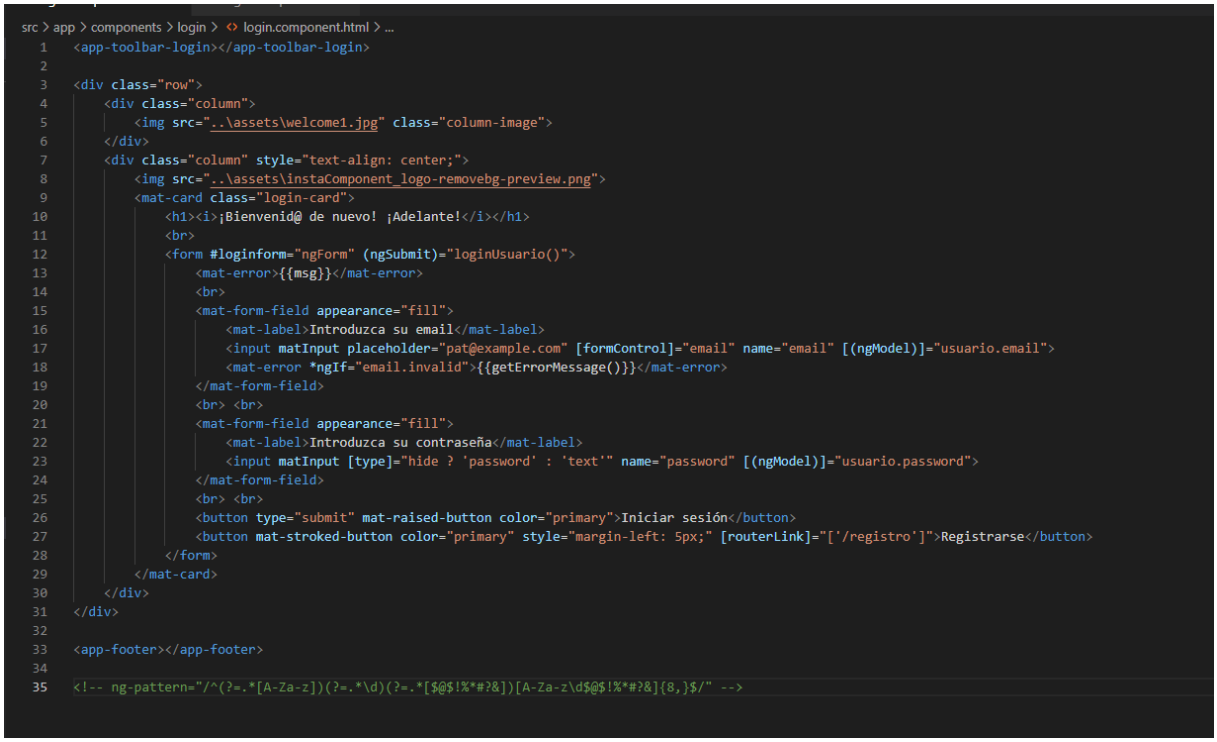
```
export class ServiciosUsuarioService {

  constructor(private http: HttpClient) { }

  /* Registro y login de usuarios */

  public loginUserFromRemote(usuario: Usuario): Observable<any> {
    return this.http.post<any>("http://localhost:8080/user/login", usuario);
  }

  public registerUserFromRemote(usuario: Usuario): Observable<any> {
    return this.http.post<any>("http://localhost:8080/user/registro", usuario);
  }

  public recuperarUsuarioFromId(idUsuario: number){
    return this.http.get<Usuario>("http://localhost:8080/user/recuperarUsuario" + "/" + idUsuario);
  }

  /* Visualización de listados de comercios y artículos */

  getComercios() {
    return this.http.get<Comercio[]>("http://localhost:8080/user/listaComercios");
  }

  getArticulos(idComercio: number) {
    return this.http.get<Articulo[]>("http://localhost:8080/user/listaArticulos" + "/" + idComercio);
  }

  /* Visualización y gestión de cesta de compra */

  public listarArticulosCesta(idUsuario: number){
    return this.http.get<Articulo[]>("http://localhost:8080/user/listaArticulosCesta" + "/" + idUsuario);
  }
```

Figura 45. Servicio REST ServiciosUsuario.

A continuación, en las figuras 46 y 47, podemos observar dos fragmentos de código del componente login perteneciente al cliente web, tanto el fichero .ts como el fichero .html.

```
export class LoginComponent implements OnInit {

  usuario = new Usuario();
  usuarioObj = new Usuario();
  msg = '';
  email = new FormControl('', [Validators.required, Validators.email]);
  hide = true;

  constructor(private service: ServiciosUsuarioService, private router: Router) { }

  ngOnInit(): void {
  }

  loginUsuario() {
    this.service.loginUserFromRemote(this.usuario).subscribe(
      data => {
        this.usuarioObj = data;
        console.log("response received");
        this.router.navigate(['/listaComercios']);
        localStorage.setItem("idUsuario", this.usuarioObj.idUsuario.toString());
        localStorage.setItem("nombreUsuario", this.usuarioObj.nombre.toString());
        localStorage.setItem('usuarioSesion', JSON.stringify(this.usuarioObj));
      },
      error => {
        console.log("Excepción lanzada");
        this.msg = "Credenciales incorrectas. Introduzca un email y contraseña válidas, por favor."
      }
    );
  }

  getErrorMessage() {
    if (this.email.hasError('required')) {
      return 'Debe introducir un email';
    }

    return this.email.hasError('email') ? 'Email inválido' : '';
  }
```

Figura 46. Fichero login.component.ts.



```
src > app > components > login > <> login.component.html > ...
<app-toolbar-login></app-toolbar-login>

<div class="row">
    <div class="column">
        <img src="..\assets\welcome1.jpg" class="column-image">
    </div>
    <div class="column" style="text-align: center;">
        <img src="..\assets\instaComponent_logo-removebg-preview.png">
        <mat-card class="login-card">
            <h1><i>¡Bienvenid@ de nuevo! ¡Adelante!</i></h1>
            <br>
            <form #loginform="ngForm" (ngSubmit)="loginUsuario()">
                <mat-error>{{msg}}</mat-error>
                <br>
                <mat-form-field appearance="fill">
                    <mat-label>Introduzca su email</mat-label>
                    <input matInput placeholder="pat@example.com" [formControl]="email" name="email" [(ngModel)]="usuario.email">
                    <mat-error *ngIf="email.invalid">{{getErrorMessage()}}</mat-error>
                </mat-form-field>
                <br> <br>
                <mat-form-field appearance="fill">
                    <mat-label>Introduzca su contraseña</mat-label>
                    <input matInput [type]="hide ? 'password' : 'text'" name="password" [(ngModel)]="usuario.password">
                </mat-form-field>
                <br> <br>
                <button type="submit" mat-raised-button color="primary">Iniciar sesión</button>
                <button mat-stroked-button color="primary" style="margin-left: 5px;" [routerLink]="['/registro']">Registrarse</button>
            </form>
        </mat-card>
    </div>
</div>

<app-footer></app-footer>

<!-- ng-pattern="/^(?=.*[A-Za-z])(?=.*\d)(?=.*[$@$!%*#?&])[A-Za-z\d$@$!%*#?&]{8,}$/" -->
```

Figura 47. Fichero login.component.html.

Finalmente, y para acabar, se presenta un fragmento de código ciertamente interesante. Pese a no ser una labor sencilla, en algunas de las vistas se ha trabajado de manera que la interfaz se comportase lo mayor *responsive* posible. Aquí se muestra cómo se ha implementado de manera *responsive* la presentación de los artículos de un comercio a un usuario. Tenemos 3 distinciones: la primera, está dirigida a pantallas de grandes resoluciones como ordenadores de sobremesa, en la cual, podemos observar de 5 a 6 elementos por fila; la segunda, está orientada a dispositivos de una resolución menor como puede ser una Tablet o un iPad colocado en formato *landscape*: de esta manera podremos ver 3 elementos por fila. Y, por último, se ha contemplado la visualización de dichos elementos en un dispositivo móvil, de manera que cada fila solamente tiene un elemento, ya que está pensado para que se vea el contenido con un simple *scroll* vertical, que es como estamos acostumbrados, y la manera más cómoda de visualizar información en uno de estos dispositivos.

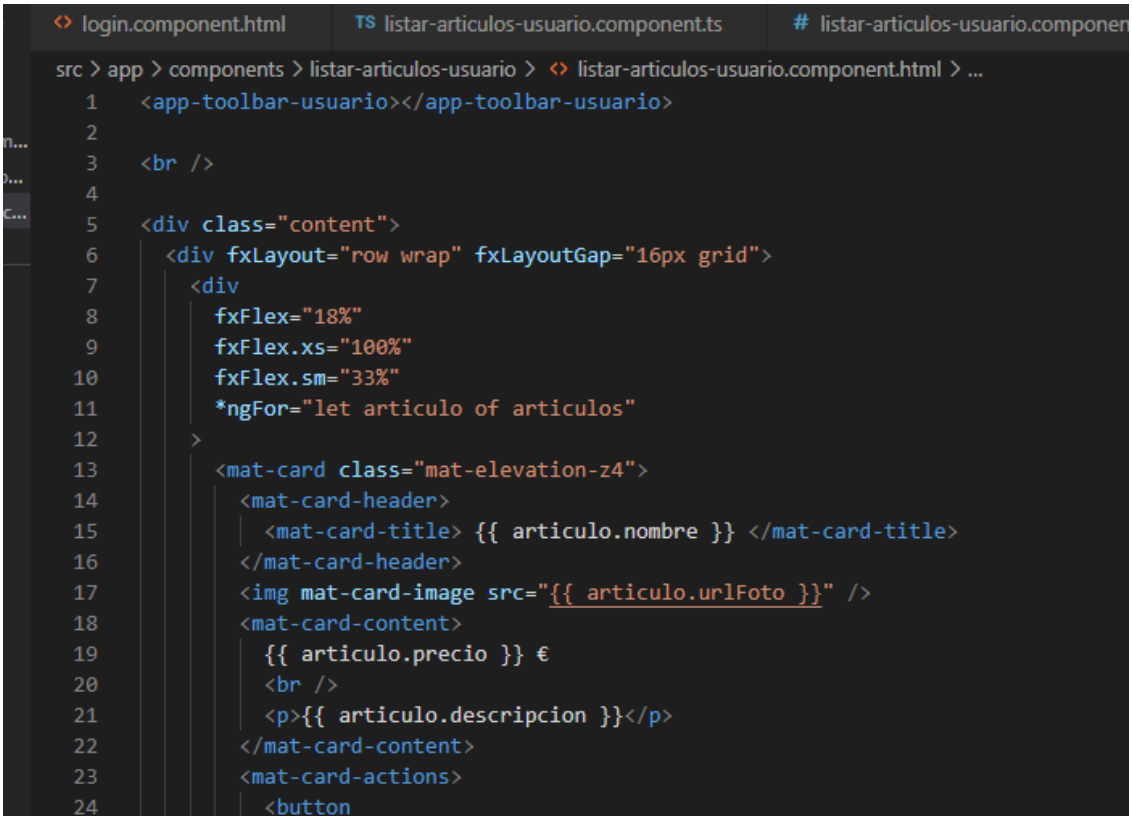


Figura 48. Fichero listar-articulos-usuario.component.html adaptación responsive.

5. Pruebas

Este apartado no es más que una extensión de la sección 3.5 en el cual se ha detallado el plan de pruebas que se va a llevar a cabo para testear la aplicación.

Por una parte, y en relación a las pruebas de la interfaz gráfica que se han ido realizando de manera manual, se puede comentar que han sido realizadas de manera continua a la vez que el prototipo iba avanzando, y de esta manera se ha verificado que los criterios de aceptación de las historias de usuario finalizadas se cumplen de manera correcta. También se ha verificado que el comportamiento de la app es el esperado ante posibles errores que puedan resultar de su funcionamiento.

De la misma manera, las pruebas de caja negra sobre el API REST han ido dándose lugar continuamente conforme la implementación del prototipo seguía su curso, verificando éstas el correcto funcionamiento de las peticiones REST tanto con datos válidos como inválidos.

6. Conclusiones

Para acabar esta memoria vamos a pasar a presentar unas conclusiones finales acerca del trabajo realizado.

En primer lugar, vamos a justificar el nivel de cumplimiento de los objetivos inicialmente propuestos. Desde mi punto de vista y en líneas generales se han cubierto los objetivos planteados en un principio en este proyecto. La temática elegida para la aplicación ha sido investigada a fondo, de manera que hemos tenido unas claras referencias con respecto a los objetivos que deseábamos acabar consiguiendo: se ha realizado un estudio profundamente elaborado tanto del ámbito de la aplicación a desarrollar como de las tecnologías óptimas para llevar a cabo el proceso. Se han seleccionado tecnologías punteras a día de hoy, por lo que no nos hemos limitado a realizar un simple desarrollo, sino que ha sido un proceso de formación para el futuro, como explicaremos con algo más de detalle, más adelante. Por otra parte, se han aplicado ampliamente muchos de los conceptos y conocimientos a lo largo del grado y en especial de la rama cursada por mí, acerca de metodologías ágiles, con las historias de usuario como punta de lanza, acompañadas de una planificación flexible y adaptada a los tiempos que corren.

La parte del proyecto que en mi opinión no se ha desarrollado al nivel inicialmente esperado ha sido la implementación del prototipo del sistema. Me hubiera gustado ahondar y pulir muchísimo más la aplicación, pero he sentido que me he encontrado con algunos inconvenientes, como ha sido la dificultad inicial de aprendizaje del *framework* Angular. Prácticamente nunca antes había trabajado con programación web por lo que me costó un tiempo de adaptación y comenzar a funcionar la aplicación, así como coger algo de soltura con el manejo del lenguaje *Typescript*, la gestión de componentes, servicios, etc. Con Spring sí que había trabajado antes, no obstante, no deja de tener cierta complejidad, que para una sola persona trabajando se puede hacer en ocasiones cuesta arriba, a la hora de solucionar los problemas e inconvenientes que vayan surgiendo a lo largo del desarrollo.

Pese a ello y, en mi opinión, el diseño del sistema tiene unas bases robustas las cuales, junto a la elección de Angular, hacen de él un proyecto con posibilidades de ser muy escalable si se desea en un futuro. Por lo tanto, me hace sentirme ciertamente satisfecho acerca del trabajo realizado, ya que no olvidemos que no deja de tratarse de un prototipo. Este ha sido uno de los puntos positivos a la hora de haber elegido este framework. Sí es cierto, como hemos comentado justo antes, que el desarrollo se ha tornado en una tarea cuesta arriba debido a la novedad que suponía para mi persona. Sin embargo, esto que hemos comentado, sumado a los conocimientos que considero que me ofrece para mi futuro en el mundo laboral, compensan con creces dichas dificultades. Valoro la tarea como una fase de aprendizaje altamente fructífera para mi persona.

Por comentar los aspectos más relevantes del proyecto, además de todo el tema del framework que ya ha sido dicho, creo que el trabajar con una metodología ágil, también puede ser fundamental para mí en el futuro debido a los puntos positivos que esta es capaz de ofrecerme a la hora de adaptarme al mundo laboral.

Sinceramente, estoy muy satisfecho de haber elegido este Trabajo de Fin de Grado, y lo volvería a elegir las veces que hiciera falta. Creo que es la guinda del pastel perfecta para poner un punto y final al grado, en el que he adquirido infinidad de conocimientos los cuales he tenido la oportunidad de aplicar aquí.

6.1. Mejoras y trabajos futuros

Como ya hemos hablado en el apartado anterior, uno de los adjetivos que pueden describir a nuestro sistema es la infinita escalabilidad que posee, por lo que en este apartado podríamos imaginar una enorme cantidad de mejoras y progresos partiendo de nuestra base, y que nunca nos llegasen a parecer suficientes.

En primer lugar, me encantaría en el futuro poder finalizar todas las historias de usuario inicialmente propuestas en la pila de producto y que finalmente no se han podido concluir por problema de tiempo. De la misma manera, se podrían añadir

muchas más que dotasen a la aplicación de una mayor complejidad y que, por tanto, harían crecer a la aplicación de una manera exponencial. Me gustaría también, mejorar mucho el aspecto de la interfaz de la aplicación, una vez mejorase mis dotes a la hora de manejar las tecnologías propuestas.

Y como ya hemos dicho, esta propuesta de aplicación, incluso podría llegar a materializarse pasando de ser un simple prototipo, a ser aplicada en el mundo de la empresa, quién sabe si haciendo real InstaComponent S.A.L.

7. Bibliografía

[1] M. Prieto, «Expansión,» 2020. [En línea]. Available: https://www.expansion.com/economia-digital/2020/08/20/5f3d852f468aeb11628b45c3.html.

[2] elEconomista.es, «elEconomista.es,» 2020. [En línea]. Available: https://www.eleconomista.es/empresas-finanzas/noticias/10833195/10/20/El-negocio-del-delivery-se-dispara-por-la-pandemia-y-moldea-la-ultima-fiebre-del-oro-en-Espana.html.

[3] Proagilist, «Proagilist: the professional agilist,» 2016. [En línea]. Available: https://proagilist.es/blog/agilidad-y-gestion-agil/backlog-de-producto-poker-de-planificacion/.

[4] W. Sandoval, «Pixelgrafía,» 2020. [En línea]. Available: http://www.pixelgrafia.com/post/114_como-crear-una-aplicacion-de-entrega-de-alimentos-delivery-similar-a-ubereats.

[5] G. L. Turnquist, Learning Spring Boot 2.0, 2014.

[6] Equipo Geek, «IfgeekthenEveris,» 2018. [En línea]. Available: https://ifgeekthen.everis.com/es/spring-framework.

[7] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Spring_Framework.

[8] F. C. C. T. A. L. Nate Murray, Ng-book 2: The Complete Guide to Angular 2, 2016.

[9] Quality Devs, «Quality Devs,» 2019. [En línea]. Available: https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/.

[10] C. Álvarez Caules, «Arquitectura Java,» 2015. [En línea]. Available: https://www.arquitecturajava.com/tiene-futuro-jsf/#JSF_y_Servidor.

[11] J. C. Valerio Barreto, «Medium,» 2016. [En línea]. Available: https://medium.com/@jc.valerio.b/angular-2-historia-caracter%C3%ADsticas-y-m%C3%A9todos-de-instalaci%C3%B3n-11492ea67e2b#:~:text=Un%20poco%20de%20Historia,como%20un%20proyecto%20open%2Dsource.

[12] R. Agasthyan, «TutsPlus,» 2018. [En línea]. Available: https://code.tutsplus.com/es/tutorials/beginners-guide-to-angular-4-components--cms-29674.

[13] FrontBackend, «FrontBackend,» 2021. [En línea]. Available: https://frontbackend.com/spring-boot/angular-11-spring-boot-2-postgresql.

[14] Angular Material, «Angular Material,» 2021. [En línea]. Available: https://material.angular.io/.

Apéndice I. Manual de Usuario

Para comenzar, en la figura 49 se puede observar un login convencional para iniciar sesión en el sistema. En la figura 50 tenemos el formulario de registro, para que un usuario previamente no registrado, pueda pasar a formar parte del sistema.

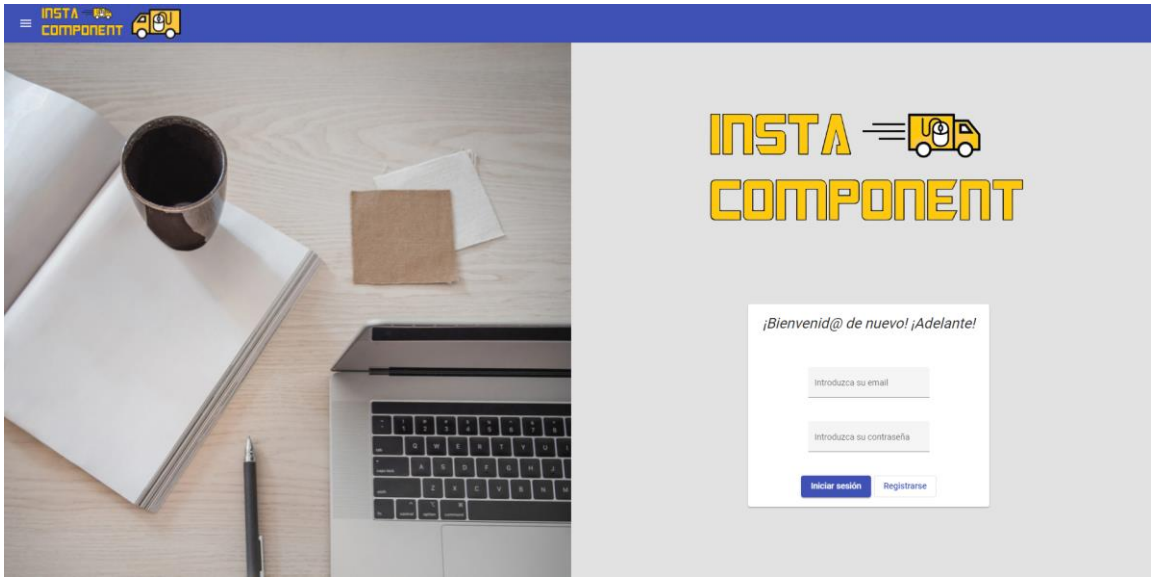


Figura 49. Vista de inicio y login.

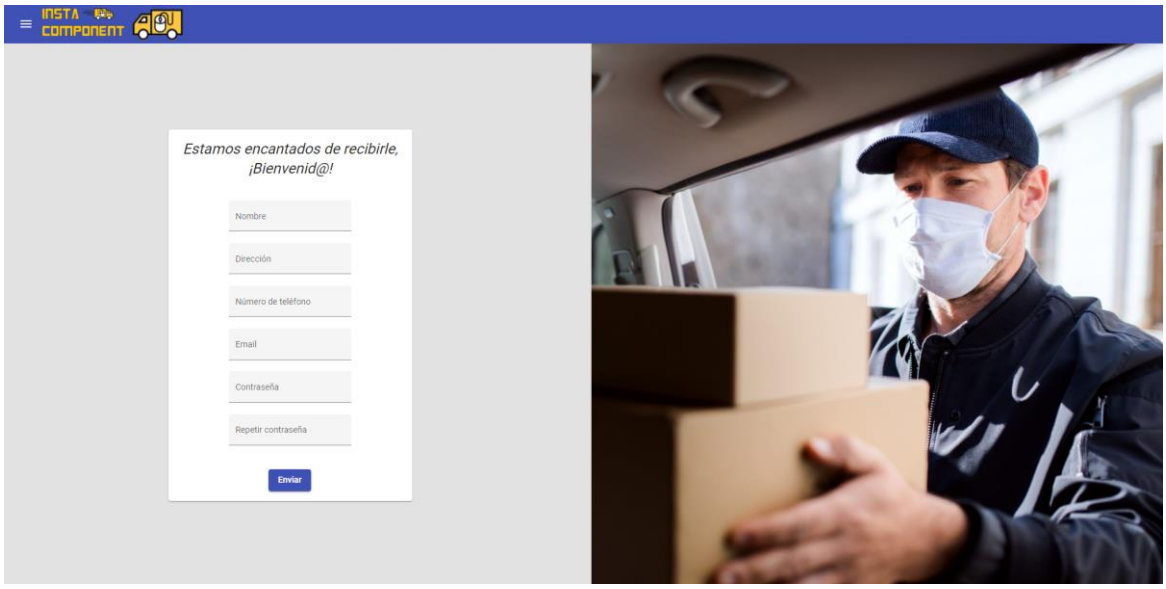


Figura 50. Vista de registro de usuarios.

Entrando al sistema como un usuario registrado, podemos ver el listado de comercios disponibles tal y como se muestra en la figura 51. Si seleccionamos uno de ellos, se puede ver el listado de los productos que ofrece, como se ve en la figura 52.

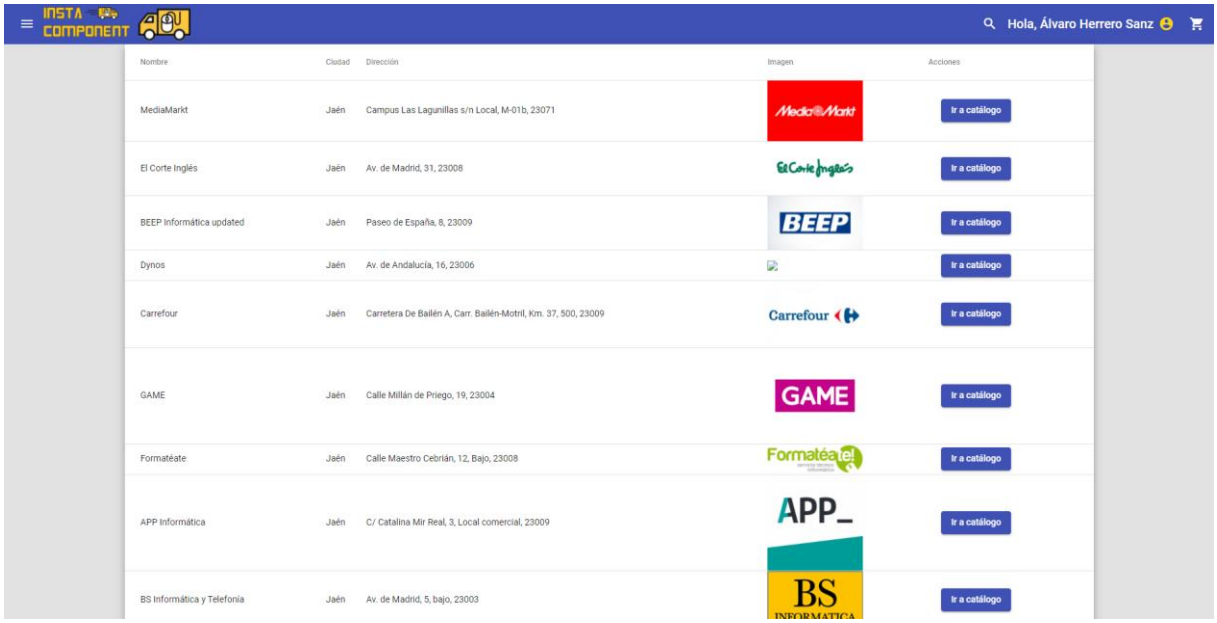


Figura 51. Listado de comercios.

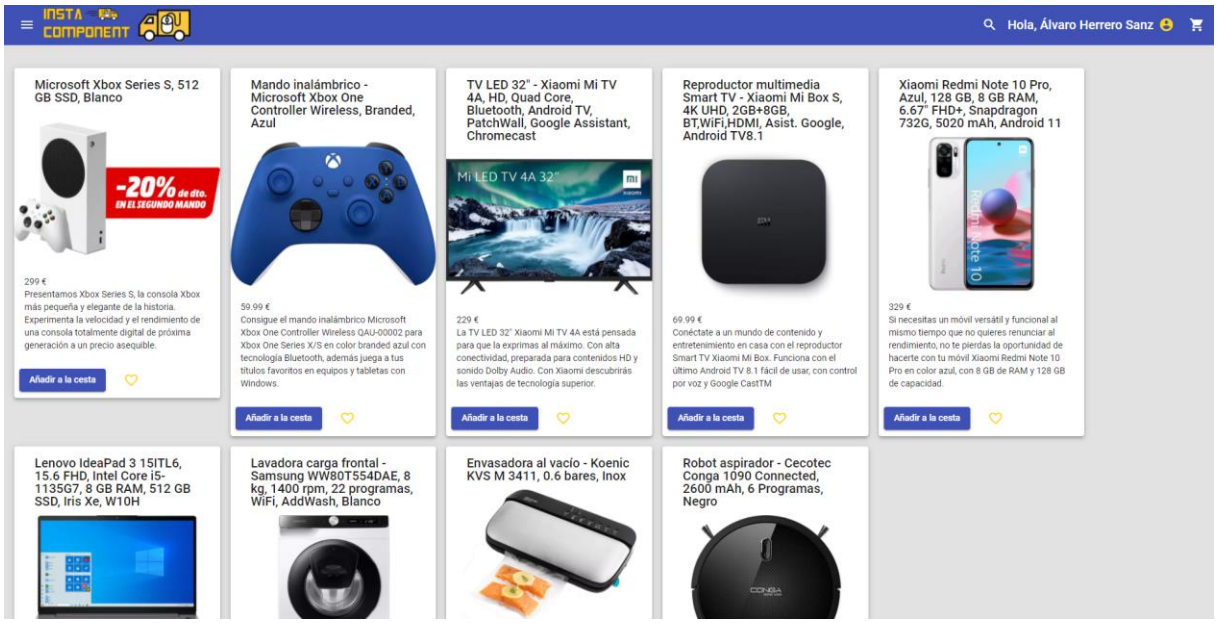


Figura 52. Listado de artículos de un comercio.

Si se pulsa el botón de añadir a la cesta, los artículos deseados se añaden al carrito, y pasan a formar parte de él, como se muestra en la figura 53.

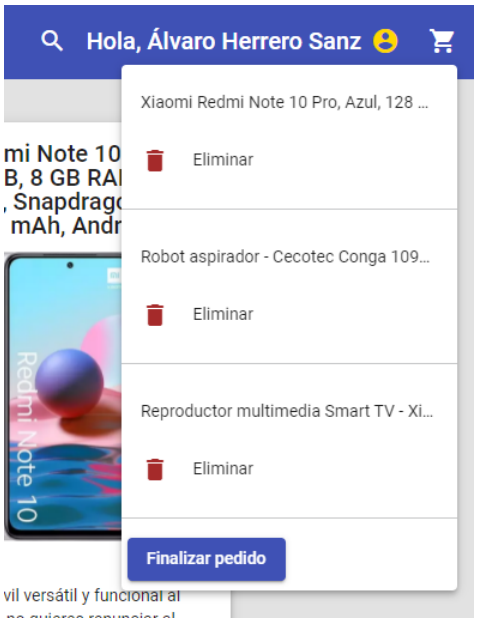


Figura 53. Cesta de la compra.

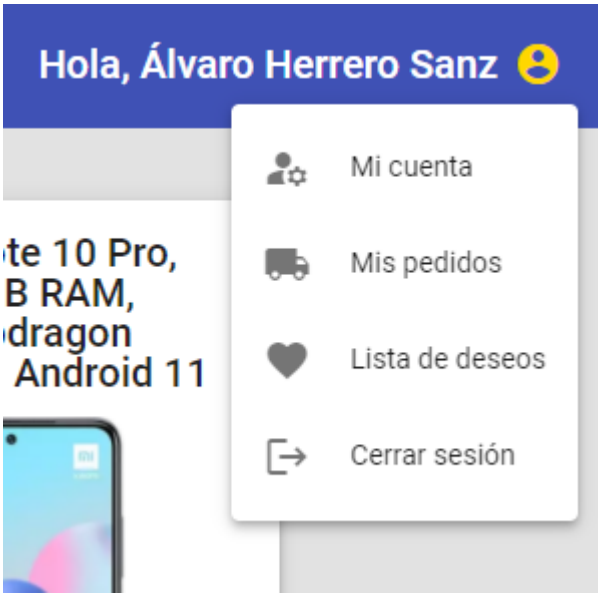


Figura 54. Sección Mi Usuario.

Si presionamos sobre el botón “Mi cuenta” podremos acceder a un panel de control específico de cada usuario (figura 55).

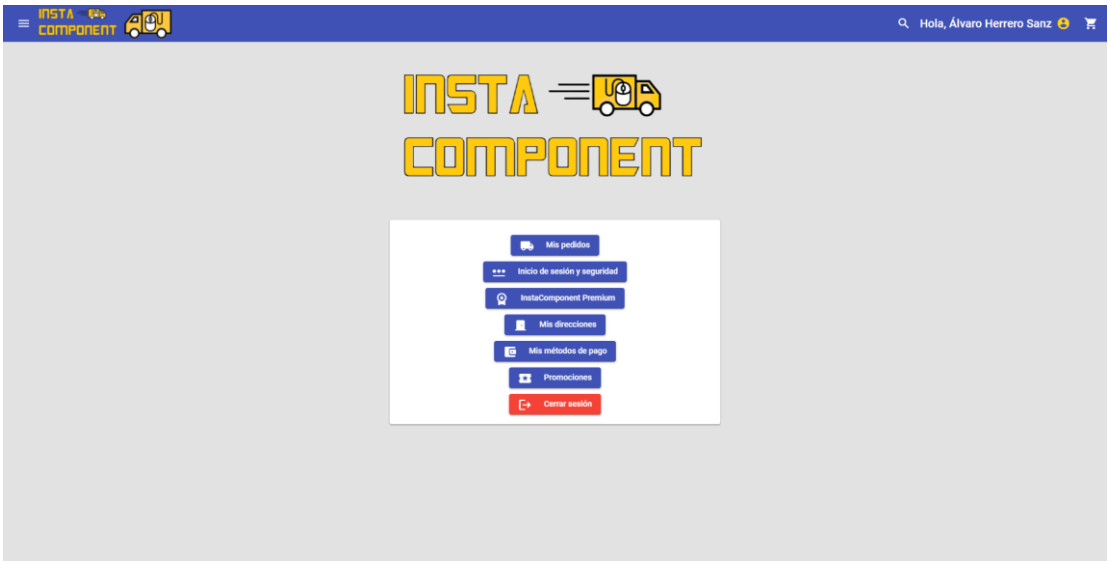


Figura 55. Panel de control usuario.

De igual manera que para clientes, existe una distinta página de login para el rol de administradores.

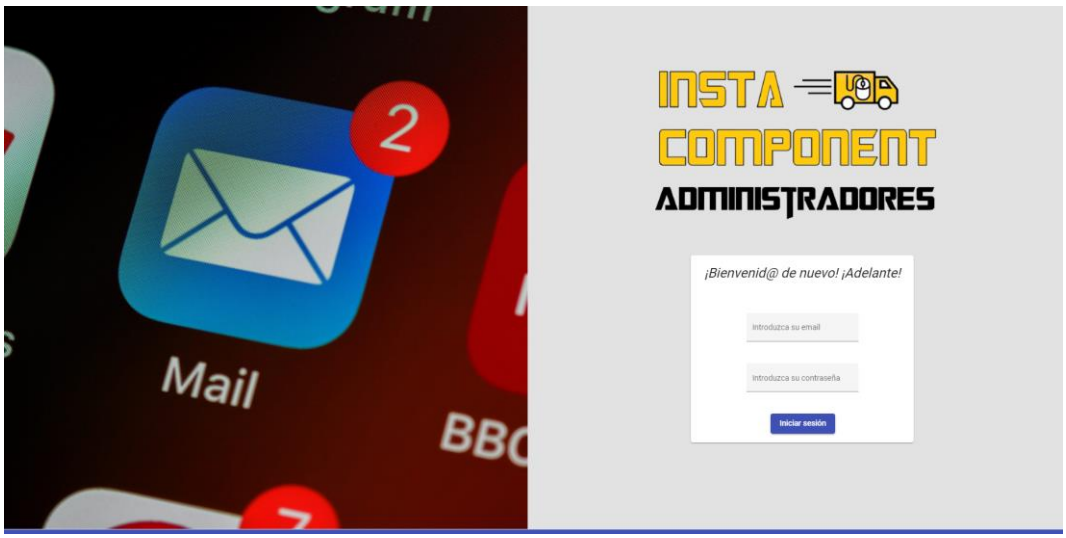


Figura 56. Vista de login administradores.

De forma similar a entrar como un cliente, podemos ver el listado de los negocios existentes, con la salvedad, de que, en este caso, al estar logueados mediante el rol administrador, tenemos a nuestra disposición las funcionalidades CRUD para operar con ellos. Sucede igual con los artículos de cada comercio y con los repartidores de la empresa, cuyo listado se puede observar en la figura 59.

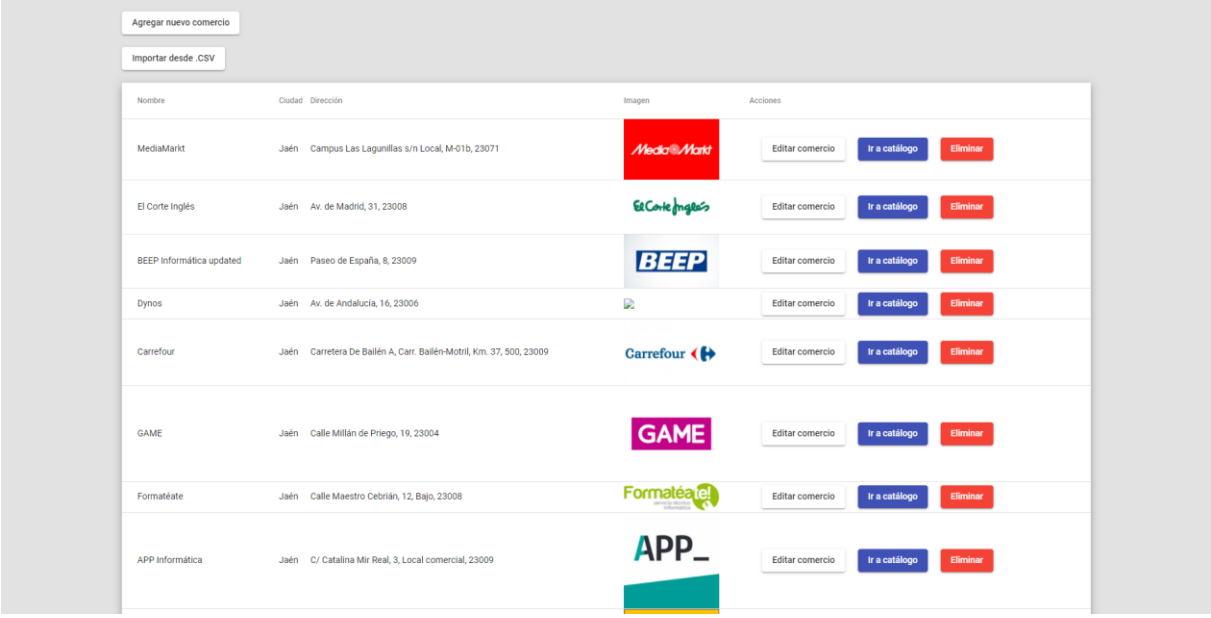


Figura 57. Listado de comercios para administradores.

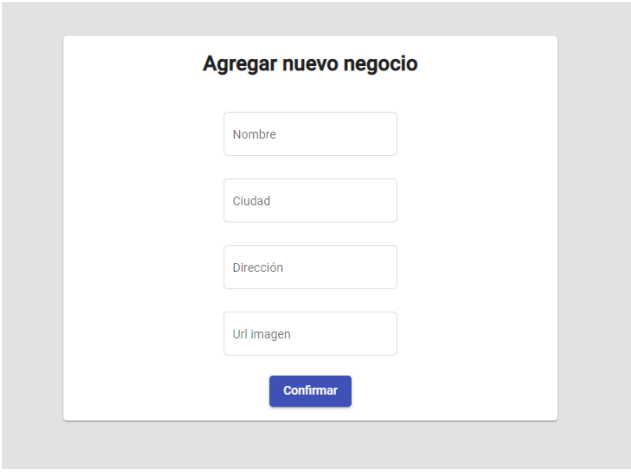


Figura 58. Formulario para agregar un nuevo negocio.

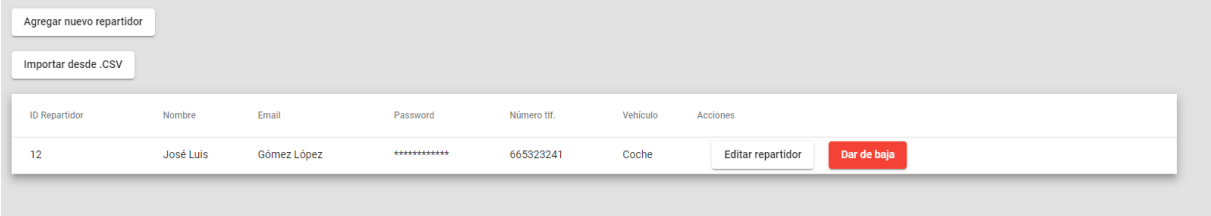


Figura 59. Listado de repartidores.

Apéndice II. Descripción de los contenidos suministrados

El fichero comprimido .zip que se entrega, además de contener esta memoria, posee los siguientes contenidos:

- InstaComponentSpring: El código fuente perteneciente al servidor de la aplicación.
- InstaComponentAngular: El código fuente perteneciente al cliente de la aplicación.
- Un fichero .vpp que contiene los diferentes diagramas incluidos en la memoria, creados mediante el software Visual Paradigm.
- Un fichero .sql que contiene un pequeño script útil para ejecutar una vez se ha creado la base de datos y esta se encuentra vacía, de modo que contaremos con algunas entradas de datos para cada una de las entidades existentes.