Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Trabajo Fin de Grado

ESTUDIO Y DESARROLLO DE UNA APLICACIÓN WEB PARA LA GESTIÓN DE RESERVAS DE UNA PELUQUERÍA

Alumno: Miguel Reche Moreno

Tutor: Prof. D. José Ramón Balsas Almagro
Dpto: Informática

Septiembre, 2018

Universidad de Jaén

Escuela Politécnica Superior de Jaén
Departamento de Informática

Don José Ramón Balsas Almagro, tutor del Proyecto Fin de grado titulado: Estudio y desarrollo de una aplicación web para la gestión de reservas de una peluquería, que presenta Miguel Reche Moreno, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2018

El alumno: Los tutores:

Miguel Reche Moreno José Ramón Balsas Almagro

Índice

1. Introducción

Introducción al proyecto

Para terminar la carrera de grado en ingeniería en informática he decidido basar mi trabajo en la realización de una aplicación móvil para la gestión de citas de una peluquería.

Este tipo de aplicación se encargará de gestionar de manera automática la posibilidad de ver las citas disponibles, la solicitud y aceptación de las citas por parte de los clientes e historial de citas, todo ello también será posible por parte del administrador de la peluquería.

La idea es que el cliente pueda desde su móvil, ya sea con un sistema operativo Android o Apple iOS ver las citas disponibles en cada momento y solicitar la que desee.

El cliente mediante un sistema de identificación, puede ver las citas disponibles, ordenadas por horas cada día de la semana, las disponibles aparecerán en verde y las ocupadas como no disponibles en rojo, seleccionando la que desee. También desde el móvil puede ver el historial de citas. El administrador podrá gestionar los usuarios que tienen acceso a la gestión de citas, pudiendo reservar directamente a un usuario, cancelar una cita si considera oportuno y buscar al usuario que desee.

Esta comunicación cliente-reserva facilita la gestión de reservas, dando al usuario total libertad para la consulta continuada de las fechas disponibles.

Motivación

Realizar la reserva de citas para una peluquería desde el dispositivo móvil de cada usuario debería ser un acto corriente para cualquier ciudadano. Hoy en días las barberías gozan de gran popularidad gracias a las tendencias actuales, entre el público masculino, siendo este establecimiento con clara orientación a los jóvenes resulta muy ventajoso el desarrollo de una aplicación.
Por ello vamos a realizar el desarrollo de una aplicación de gestión de una barbería para la solicitud de citas, que contendrá una aplicación móvil para los clientes y una aplicación web para el dueño de la barbería, con la que pueda realizar la gestión pertinente.

Esta barbería se encuentra ubicada en la localidad de La Carolina, por ella pasan la mayoría de los hombres de la localidad, sobre todo los jóvenes, por ello, el dueño de la barbería nos solicitó la implementación de esta solución

adaptada a sus necesidades, siendo considerado cliente a lo largo de la memoria.

Según *ditrendia[1]*, que es una conocida marca de estrategia de marketing y ventas digitales, España lidera el ranking mundial de usuarios de móvil con acceso a internet con un 88% y en el mundo un usuario pasa una media de 170 minutos al día mirando el móvil, de todos ellos un 100% de los jóvenes son móviles mientras que un 99% accede a diario a través de internet con él.
Por ello el desarrollo de una aplicación de gestión es una solución acertada para este establecimiento, ya que los clientes para los que está orientado este sector emplean gran parte de su tiempo libre en la interactuación con aplicaciones de dispositivo móvil.

El uso de las aplicaciones ha aumentado un 11% respecto al último año, un 80% busca información comercial, así como un 70% lo utiliza para comparar precios o buscar opiniones, todos estos datos nos hacen pensar en la ventaja competitiva de tener una aplicación de gestión.

Por otra parte, según un *informe del centro criptológico nacional[2]*, en España 9 de cada 10 móviles tienen el sistema operativo Android. Ya que nuestra aplicación está orientada a un público centrado en la ciudad se tuvieron en cuenta las posibilidades de la realización de la aplicación en Android nativo o bien optar por una tecnología híbrida. Fue esta última opción la opción la que salió victoriosa gracias a que se puede llegar al 100% de los potenciales usuarios.

Una vez decidido que se iba a utilizar una tecnología híbrida para el desarrollo de la aplicación, se eligió utilizar el framework *Ionic* [1] ya que podemos crear aplicaciones móviles en diferentes plataformas que pueden instalarse en teléfonos con Windows, Android e iOS.
Ionic utiliza HTML5, lo que facilita la creación de aplicaciones, además que el uso de *Angular JS*[2] y *Apache Cordova*[3] lo hace muy interactivo y cuenta con una gran gama de métodos ya que es un framework de Javascript de código abierto mantenido por Google, lo que nos da bastante seguridad en lo que a compatibilidad y mantenimiento del framework se refiere.

En la realización de este proyecto se van a poner en práctica conocimientos adquiridos en asignaturas como, desarrollo de aplicaciones web, bases de datos, estructura de datos, desarrollo ágil, desarrollo de aplicaciones empresariales, ingeniería de requisitos, etc… cada una desde su fase de desarrollo, ayudará a la finalización del proyecto y la puesta en marcha de un caso práctico real.

[1] https://ionicframework.com/
[2] https://angularjs.org/
[3] https://cordova.apache.org/

Objetivos

El objetivo principal de este proyecto será dotar de una herramienta de gestión de citas para los clientes de una peluquería, esta aplicación será compatible con cualquier sistema operativo móvil. Por otra parte, el administrador tendrá un gestor en el cual podrá consultar a los usuarios, así como reservar o cancelar horarios de citas.

Objetivos específicos

Para realizar este proyecto, vamos a marcar unos objetivos que nos ayudaran a ir completando las distintas fases de su desarrollo y logar el objetivo final.
Los objetivos específicos para lograr el objetivo general son los siguientes:

- Extraer los requisitos del producto software que vamos a desarrollar, en el que obtendremos lo que debe de ser capaz de realizar el sistema y obtendremos datos para la etapa de diseño de la solución. Para ello se realizará un estudio de soluciones existentes además de consultar las necesidades específicas del cliente.
- Diseño general de la arquitectura del proyecto software, este objetivo, desarrollaremos una representación técnica, en el que incluirán un análisis y diseño completo del proyecto software que vamos a desarrollar.
- Desarrollar una aplicación móvil en el framework de tecnología híbrida *Ionic* para la gestión de reservas por parte de los clientes de una peluquería.
- Desarrollar una aplicación web en Spring MVC en el que se desarrollará un servicio REST que cuente con un sistema gestor de bases de datos para que el administrador sea capaz de gestionar usuarios y las reservas en su comercio.

Metodologías

Para la realización de este proyecto se va a seguir una metodología de trabajo ágil[3], ya que esta utiliza un proceso de acercamiento iterativo al producto final, pudiendo ir entregando intervalos de desarrollo al cliente, aparte de tener una mayor libertad de realizar cambios en el proyecto a medida que se va realizando.

Esto nos permite ir realizando el proyecto en pequeñas tareas las cuales se pueden ir desarrollando de forma simultánea. El cliente puede intervenir y las pruebas de software se realizan mientras se lleva a cabo del desarrollo del mismo, con lo cual se consigue una mejora continua y una adecuación constante del producto a las necesidades del cliente.
El término ágil fue aplicado al desarrollo de software en una reunión celebrada en 2001, en esta reunión su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software

rápidamente y respondiendo a los cambios que surjan a lo largo del proyecto [4].
Firmaron el 'Manifiesto por el desarrollo ágil del software' [4], en el que destacaron los siguientes principios [5]:

- Individuos e interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.
- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio, mejor que acogerse a un plan.
-

Dando importancia a los elementos de la derecha, aunque con más importancia aun a los de la izquierda, surgió un nuevo paradigma.
Dado que al comienzo del desarrollo de la aplicación el cliente no tenía claras sus prioridades, hace que una metodología ágil sea especialmente útil, ya que la comunicación continua con el cliente conseguirá que se vayan alcanzando los objetivos.
Otra de las ventajas de utilizar una metodología ágil es que, por medio de reuniones continuas con el cliente, llamadas *sprints*, se irán clasificando las tareas en orden de prioridad y se podrá llegar a completar el sprint semanal.
Por todo ello se ha decidido emplear una metodología ágil en el desarrollo del proyecto. Entre todos los tipos de métodos ágiles que podemos emplear para el desarrollo de software, hemos decidido emplear la metodología Scrum[6].

- **Diferencias entre metodología tradicional y metodología ágil**

En este punto vamos a comparar las dos metodologías que hemos estudiado para la gestión de proyectos, en nuestro caso vamos a adoptar una gestión del proyecto mediante una metodología ágil, en base a los beneficios que aporta esta metodología en contra de una tradicional.
Con la gestión de proyectos mediante un enfoque tradicional lo que tratamos es de controlar la incertidumbre del proyecto definiendo el proyecto al completo desde el principio, estableciendo un plan detallado con unos tiempos y costes controlados, así como la priorización de tareas.
Las metodologías ágiles proponen, como como se observa en la ilustración 2, un modelo más adaptativo, el que se contempla la refactorización, ir añadiendo funcionalidad al proyecto.
En lugar de tener un plan cerrado desde el principio, las metodologías ágiles buscan tener un contacto continuado con el cliente, por tanto, los cambios se contemplan como algo normal en el proyecto [7].

Ilustración 1: diferencias entre metodologías tradiciones y ágiles

Fuente: ://excelencemanagement.wordpress.com/2017/05/29/los-beneficios-de-implementar-la-metodologia-agil/

Los beneficios de optar por una metodología ágil son los siguientes [8]:

- **cambios**: se pueden realizar cambios en requerimientos generados por necesidades del cliente rápidamente, ya que este tipo de metodología está diseñada para adecuarse a nuevas exigencias.
- **calidad**: después de cada sprint, se espera obtener una versión del trabajo funcional, lo que hace desarrollar un software de calidad.
- **productividad**: el equipo de desarrollo puede estructurar su trabajo de manera autónoma, lo que favorece la motivación.
- **tiempos**: es posible estimar de manera fácil el tiempo de desarrollo que se va a emplear en una funcionalidad y cuándo se podrá hacer uso de ella.
- **riesgo**: al desarrollar primero las tareas con más alta prioridad y conocer la estimación de tiempos, nos podemos adelantar a riesgos que sucedan durante el desarrollo.

- **SCRUM**

  Dentro de las metodologías ágiles de desarrollo encontramos varios tipos de métodos, uno de ellos es el *Scrum*, que es el que vamos a utilizar en el proceso de desarrollo de la solución.

  Scrum es una metodología ágil que ayuda a trabajar activamente con el cliente durante el desarrollo del proyecto, con ello se pueden tener nuevas funcionalidades que no han sido incluidas desde un inicio o tener un punto de vista diferente.

  Esta metodología nos permite adaptar las expectativas del cliente y priorizarlas en base a unas estimaciones y necesidades finales de usuarios. Cada entrega que produzca el sprint se incluirán nuevas funcionalidades que cubran las necesidades especificadas por el cliente, esta nueva funcionalidad se puede probar con usuarios y adaptar cambios o integrar una nueva funcionalidad que no han sido definidas con anterioridad.

Cada funcionalidad que sea requerida por el cliente se transformará en una tarea con un orden de prioridad en función de su valor y unas estimaciones de tiempo que formarán parte del sprint [6].

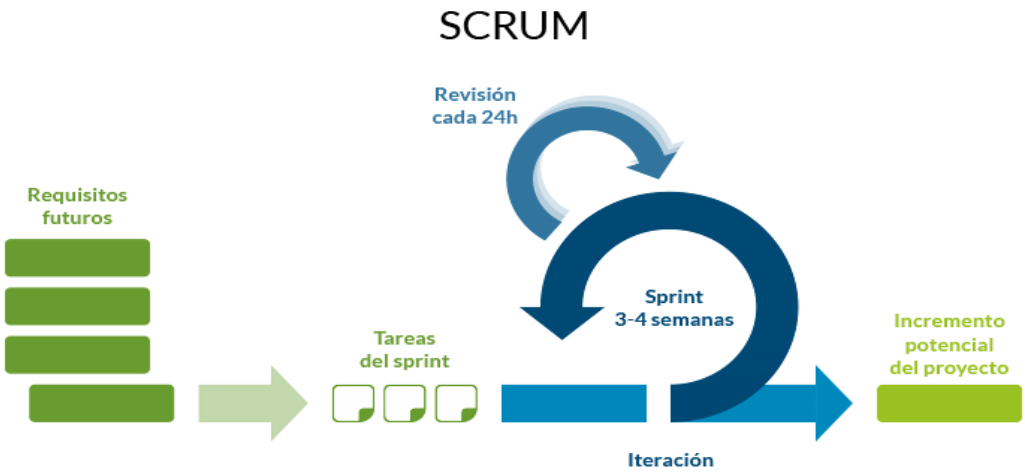

Ilustración 2: metodología scrum Fuente: https://lorbada.com/es/diferentes-metodologias-agiles

Adaptación de SCRUM al proyecto.

Una vez vista la definición de esta metodología, vamos a ver los procesos que vamos a llevar a cabo con esta en el desarrollo de la solución.

Para adaptar scrum al proyecto se va a realizar el siguiente flujo de trabajo [9]:

- **Sprint**: periodo de tiempo en el que se lleva a cabo el desarrollo, la duración del sprint será constante durante todo el desarrollo del proyecto, se comenzará con una duración de 2 semanas, el cual se podrá ir ajustando en función al ritmo de desarrollo. Al final de cada sprint, se presenta el producto que podrá ser entregado al cliente.
- **Planificación**: se procederá a una reunión con el cliente en la que se detallarán como y cuáles van a ser las funcionalidades que van a ser incluidas dentro del sprint. Una vez las tareas sean definidas se podrá estimar su prioridad, así como los tiempos en función de las tareas incluidas y cuando se entregará el objetivo del sprint.
- **scrum diario:** reunión diaria en la que se sincroniza el trabajo y se establece el plan para el desarrollo diario del proyecto. En esta reunión se pondrá en un panel las tareas a realizar y el proceso que se lleva a cabo en el sprint.
- **revisión**: Una vez se finalice el sprint, se comprobará el incremento con el cliente, él decide las funcionalidades que están completadas y las que no. Según las características del sprint se puede incluir algún tipo de documentación.
- **retrospectiva**: una vez concluida la revisión del sprint se analizarán problemas y aspectos a mejorar en el desarrollo de la aplicación.

Scrum tiene diferentes roles en el equipo de desarrollo, pero en mi caso he tenido que hacerme cargo de varios roles en el proceso de desarrollo [10]:

- **Product owner:** define las funcionalidades principales de la aplicación, establece una priorización de ellas y evalúa los entregables, este rol será para el dueño de la peluquería.
- **Scrum master-equipo de desarrollo:** el rol de scrum master se encarga de liderar al equipo de desarrollo, gestiona la reducción de impedimentos y trabaja con el product owner, por otra parte, el equipo de desarrollo tiene los conocimientos técnicos necesarios para el desarrollo de la aplicación. Al estar solo al cargo del desarrollo estos dos perfiles de desarrollo los llevaría individualmente.

Por otra parte, las reuniones se realizarán de manera individual, planificando antes de llevar a cabo del desarrollo.

Estructura del documento

El documento se estructura de la siguiente forma:

- En el apartado número 2, se encuentra el análisis de la solución, en este punto se van a estudiar un marco de trabajo que nos permita hacer estimaciones razonables de recursos y planificación temporal para el desarrollo con los objetivos marcados por el cliente.
- En el apartado número 3 se define el diseño de la aplicación, en este punto se estudiarán las técnicas que nos van a proporcionar unos principios para definir la realización de la solución.
- En el apartado número 4 está la implementación de la aplicación, en este punto se encuentra la implementación que se ha llevado a cabo a lo largo del proceso de producción, se indican las tecnologías que se han utilizado al igual que imágenes de la implementación.
- El apartado número 5 es el relacionado con las pruebas a las que hemos sometido a la aplicación para llegar a los criterios de satisfacción marcados por el cliente en el análisis de requisitos.
- El apartado número 6 es para las conclusiones que hemos obtenido durante el desarrollo del proyecto.
- El apartado número 7 se encuentra la bibliografía relacionada con la documentación del proyecto, en ella podremos acceder a las fuentes estudiadas.
- El apartado número 8 es para el índice de las ilustraciones utilizadas en esta documentación.
- En el apartado número 9 nos encontramos el índice de tablas del documento.
- En el apartado 10 está el manual de instalación, dentro de este apartado tendremos la información necesaria para poder poner en funcionamiento el sistema.

- En el apartado 11 es para el manual de usuario, en esta sección indicaremos como utilizar todas las funcionalidades de tiene el sistema, a parte, de una breve explicación con imágenes de cada vista.
- En el apartado 12 es para indicar el contenido suministrado para la realización de este proyecto.

2. Análisis

Esta fase es la más importante en el desarrollo de una aplicación. Es imprescindible un buen análisis para crear una buena base para el proyecto. En ella se analiza el problema detalladamente, los requisitos que debe cumplir la aplicación para un correcto funcionamiento y la solución escogida.
Usualmente el cliente/usuario tiene una visión incompleta/inexacta de lo que necesita y es necesario ayudarle para obtener la visión completa de los requisitos. El contenido de comunicación en esta etapa es muy intenso ya que el objetivo es eliminar la ambigüedad en la medida de lo posible.
El análisis de requisitos permitirá que al desarrollar la aplicación se especifiquen las funcionalidades que tendrá, así como la interfaz de usuario y las restricciones que debe tener el software, todo esto sujeto a posibles cambios que pueda haber.

Análisis preliminar

Actualmente el centro cuenta con una gestión de citas mediante una agenda clásica en la que, una vez que el cliente se persona físicamente al establecimiento, allí el gestor apunta el nombre y teléfono del cliente, lo que presenta algunas desventajas:

- El cliente debe de personarse el cliente por sí mismo en el establecimiento en horario de apertura.
- Tener que llamar en horario de apertura al establecimiento y no tener la línea ocupada por otro cliente.
- La gestión de un comercio en base a la escritura a mano es una práctica poco eficiente, ya que requiere que un empleado debe dedicar su tiempo a consultar y actualizar la agenda y a atender al cliente, lo que evita que pueda desarrollar labores más productivas.
- Por otra parte, también es una práctica muy insegura ya que un deterioro continuo o una pérdida del cuaderno pueden dar lugar a la pérdida de información básica para el funcionamiento del local.

Ilustración 3: diferencias entre método tradicional de reserva de citas y aplicación de reservas.

Independientemente del tamaño de la empresa, la tecnología entre otras cosas te permite simplificar el trabajo, agilizar procesos, reducir errores y sobretodo fortalecer la toma de decisiones de tus equipos en campo.

Un término intermedio en el uso de tecnología de gestión serían las agendas electrónicas o documento electrónico, en este caso la información sobre la

gestión de citas y datos de usuario estarían almacenados en un sistema de gestión, pero tendrían los mismos problemas de eficiencia en el trabajo que una hoja de cálculo tradicional, ya que un empleado debería de atender al cliente durante todo el proceso. La implementación de sistemas de gestión de reservas de citas por parte de los clientes mediante una tecnología móvil, permite de la independencia en la gestión de citas, dándole al usuario total disponibilidad para acceder al horario de reservas y poder realizar una reserva sin tener que presentarse físicamente en local, ni tener que realizar una llamada telefónica en horario laboral.

El sistema de gestión de reservas que proponemos en el siguiente trabajo permitirá agilizar el proceso de reservas, rentabilizarlo al máximo y tener estos beneficios:

- **Disminuye el riesgo de errores**: lo que evita que haya pérdidas de información por culpa de un error de escritura, poca legibilidad o deterioro del cuaderno.
- **Permite la administración integral de la compañía:** Esto te permitirá delegar la acción de gestión de reservas en los clientes con toda la confianza posible.
- **Hace más eficiente la gestión de reservas:** El sistema de gestión tendrá información detallada de cada usuario, así como historial para consultarlo cuando se desee.
- **Se trata de un software especializado**: Que incluye mejoras constantes y que se trabaja y diseña en función de las necesidades definidas por el cliente
- **Permite el análisis de información en tiempo real:** La consulta de reservas estará actualizada en tiempo real a disposición de cualquier cliente que desee consultarlo.
- **Automatización de las acciones:** Al contrario que en una versión manual, el sistema permitirá la automatización de tareas como la gestión de citas o consulta de cliente.
- **Impulsa a crecer ordenadamente:** Si el establecimiento crece en lo que a recursos humanos se refiere, el sistema se adaptará rápidamente facilitando el incremento de personal.

En el desarrollo de la aplicación servidor hemos decidido realizar la implementación con una tecnología web, ya que una aplicación web nos va a permite trabajar desde cualquier sitio ya que los datos de la aplicación (y puede que la propia aplicación) estarán almacenados en un servidor de Internet y podremos acceder a ellos simplemente teniendo una conexión a la red [11].

- **Aplicaciones web:** es una aplicación o herramienta informática accesible desde cualquier navegador, bien sea a través de internet (lo habitual) o bien a través de una red local. Lo que las hace accesibles desde cualquier dispositivo que esté conectado a la red, no requiriendo ningún tipo de instalación. Este dato aporta una gran ventaja en nuestro caso, ya que el administrador puede tener acceso a los

recursos del sistema en cualquier momento.

Ilustración 4: funcionamiento de aplicaciones web. Fuente: https://www.neosoft.es/blog/que-es-una-aplicacion-web/

- **Aplicaciones de escritorio:** Será un programa el encargado de realizar la funcionalidad del software implementado que instalaremos en cada puesto de trabajo y se conectará a través de Internet con la base de datos. Una gran desventaja para la realización de la aplicación de administración en escritorio es su escasa portabilidad ya que, si lo implementamos para un entorno Windows, solo en equipos de ese tipo funcionará y no podremos usarla en una tablet o un teléfono.

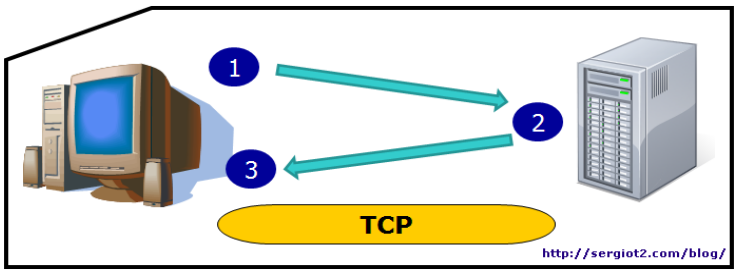

Ilustración 5: funcionamiento de aplicaciones escritorio. Fuente: https://geeks.ms/sergiotarrillo/2009/01/14/aplicaciones-de-escritorio-vs-aplicaciones-web-hay-diferencia-en-el-desarrollo/

Para el desarrollo del sistema se va a tener en cuenta las diferentes opciones que tenemos para su desarrollo y elegir la que más se adecue a las necesidades del establecimiento [12]:

- **Aplicación nativa:** La aplicación nativa está desarrollada y optimizada específicamente para el sistema operativo determinado y la plataforma de desarrollo del fabricante, como por ejemplo *Objective C* o *Swift* para *iOS*, *Java* para *Android* y *.Net* para *Windows Phone*. Esto requiere la necesidad de desarrollar una aplicación diferente para cada tipo de plataforma. Este tipo de aplicaciones se adapta al 100% con las funcionalidades y características del dispositivo obteniendo así una mejor experiencia de uso y un mayor rendimiento frente a las aplicaciones híbridas.

Trabajar en modo nativo nos permite tener disponibles las últimas funcionalidades que se hayan añadido a los sistemas operativos, así como poder acceder a recursos del dispositivo.

Ilustración 6: diferencias entre aplicaciones nativas, híbridas y webapps Fuente: https://obux.wordpress.com/2017/02/14/apps-nativas-vs-apps-hibridas-vs-apps-generadas-ventajas-y-desventajas/

- **Aplicación web**: La aplicación web es la opción más sencilla y económica de crear aplicaciones, puesto que al desarrollar una única aplicación se reducen al máximo los costes de desarrollo. Asimismo, en este tipo de aplicaciones, puede utilizarse el "responsive web design", creando así una única aplicación adaptada para todo tipo de dispositivos. Por el contrario, la aplicación web ofrece una peor experiencia de uso, puesto que ignora las características del dispositivo y una menor seguridad ya que depende de la seguridad que ofrezca el propio navegador.

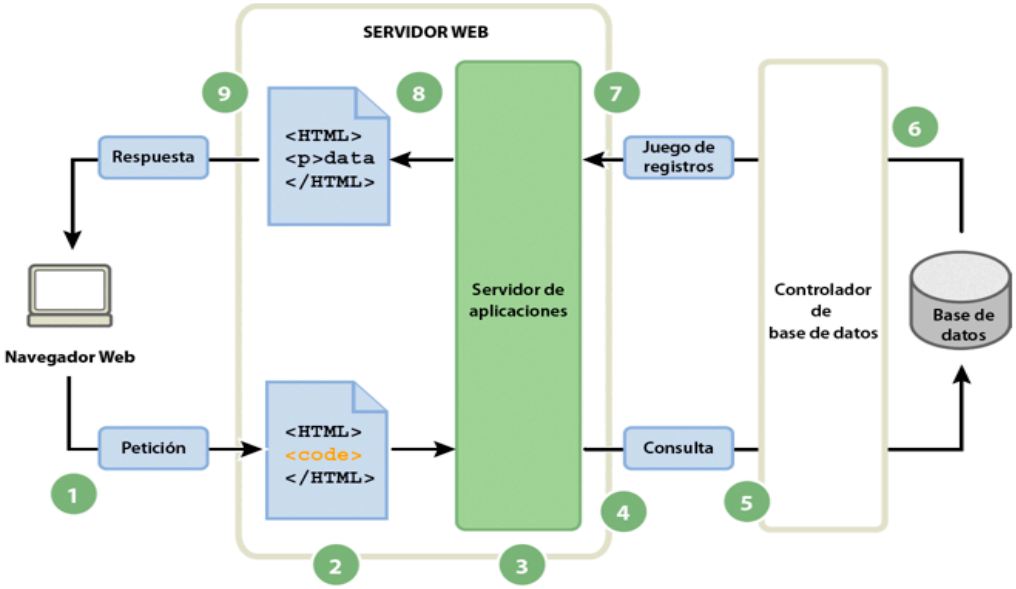

Ilustración 7: estructura de aplicaciones web. Fuente: http://sitioswebqueretaro.com/plan-aplicaciones-web.php

- **Aplicación híbrida:** Son aplicaciones desarrolladas usando *HTML5*, *CSS* y *JavaScript*[4], desplegadas dentro de un contenedor nativo como *Cordova*[5] el cual brinda acceso a las capacidades del dispositivo de una forma totalmente neutral respecto al sistema operativo. Aprovecha al máximo la versatilidad de un desarrollo web y tiene la capacidad de adaptación al dispositivo como una app nativa.
  Además, comporta un menor coste que una aplicación nativa y una mejor experiencia de uso que una aplicación web. Sin embargo, tiene un rendimiento ligeramente inferior al de una aplicación nativa debido a que cada página debe ser renderizada desde el servidor y supone una mayor dificultad de desarrollo.

  Las aplicaciones híbridas llegan a cualquier tipo de usuario independientemente del dispositivo que esté usando.

[4] https://www.w3.org/html/
[5] https://cordova.apache.org/

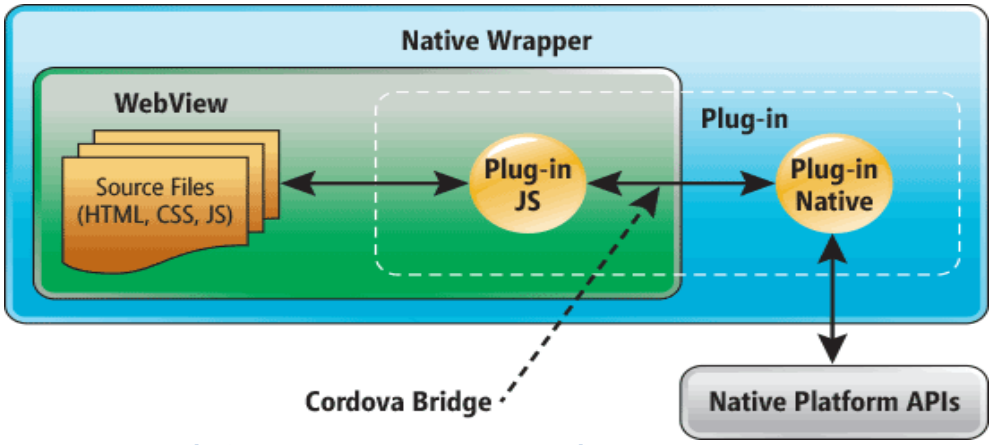

Ilustración 8: estructura de aplicaciones híbridas. Fuente: https://msdn.microsoft.com/en-us/magazine/dn879349.aspx

- **Generadas**: Son aplicaciones desarrolladas usando herramientas como *Xamarin*[6] o *Genexus*[7] (entre muchas otras), en donde el desarrollo se realiza usando técnicas y lenguajes específicos de la herramienta y luego se genera la App en el lenguaje de la plataforma destino para ser compilada con las herramientas nativas. Son lo que se denomina falsas nativas [13]. La ventaja de esta tecnología es que, como las aplicaciones híbridas, utilizan un lenguaje base para todas las plataformas. Por otra parte, es una tecnología nueva y aún está en desarrollo lo que hace que los accesos a recursos del sistema donde esté instalada la aplicación necesiten de un módulo nativo para su funcionamiento.

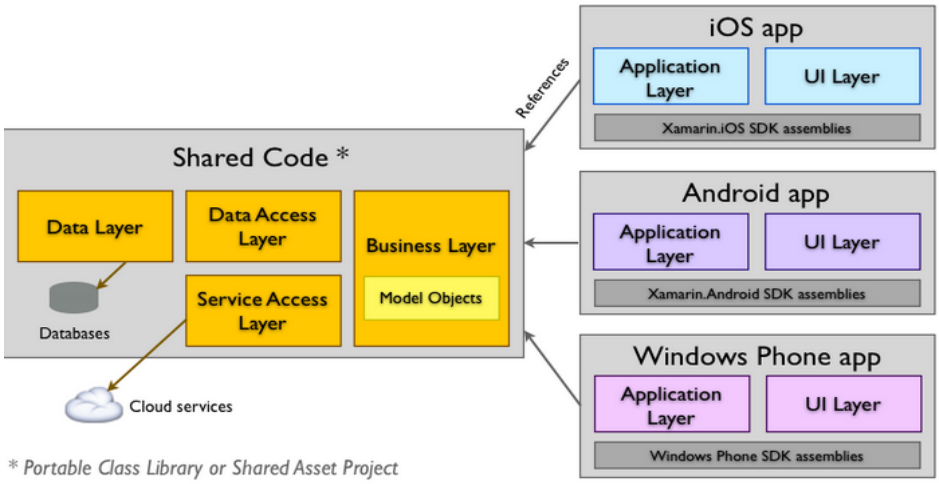

Ilustración 9: proyecto de tecnología híbrida. Fuente: https://obux.wordpress.com/2017/02/14/apps-nativas-vs-apps-hibridas-vs-apps-generadas-ventajas-y-desventajas/

Una vez realizada la comparativa entre todas las tecnologías con las que podemos emprender el desarrollo se ha elegido por el desarrollo de la aplicación en una tecnología híbrida, ya que con ello llegaremos al 100% de los clientes y no perderemos rendimiento, ya que nuestra

[6] https://docs.microsoft.com/es-es/xamarin/
[7] https://www.genexus.com/es/global

aplicación no hará uso de componentes del dispositivo, a todo ello hay que sumarle los bajos costes de desarrollo de la aplicación.

Estado del arte

En este punto vamos a estudiar las soluciones similares que podemos encontrar en el mercado, con este estudio, obtendremos unas ideas de las funcionalidades que posee cada aplicación y podemos hacer un estudio relativo a las funcionalidades más repetidas, así como las exclusivas de cada aplicación por si pueden ser agregadas a nuestra solución.

Tener una aplicación móvil te puede ayudar a conseguir más clientes para tu peluquería o centro de estética, tener más citas y aumentar los ingresos. Para muchas pequeñas y medianas empresas, una aplicación móvil es una nueva y una gran manera de conectar con sus clientes y abrir nuevos canales de marketing. Pero para las peluquerías, en particular, las aplicaciones móviles ofrecen un montón de grandes características para impulsar el negocio de manera rentable

Ilustración 10: imagen de una barbería. Fuente: http://miguelvisbarbershop.com/

Propuesta de solución

Desarrollar una aplicación a medida del local, donde los clientes puedan hacer uso de la aplicación gestionando las reservas, reservando o cancelando las ya realizadas y obteniendo información sobre su historial de reservas. La aplicación será desarrollada en el framework de tecnología híbrida móvil Ionic ya que podemos llegar a todos los clientes potenciales del local con un solo desarrollo.

Por otra parte, se desarrollará un servidor utilizando el *framework* de desarrollo Java *Spring*[14] que proporciona una *API REST*[15]para la comunicación con la aplicación de los clientes. Por otro lado, toda la gestión de la información se realizará por medio de una aplicación web, desarrollada mediante el módulo Spring MVC, a la que sólo tendrá acceso el administrador principal, es decir, el dueño del local. En esta aplicación se podrá consultar en tiempo real el estado de las reservas, el cliente que ha realizado cada reserva, pudiendo conceder nuevas reservas a usuarios que no han sido dados de alta en la aplicación o bien cancelar una reserva realizada por cualquier cliente. Otra funcionalidad que tendrá la aplicación web será la búsqueda de usuarios en la que se podrá conocer el historial de cada uno.

La aplicación almacenará toda la información en un sistema de gestión de bases de datos relacional, donde se encontrarán datos del usuario y las reservas que se han realizado. Para ello utilizaremos *JPA*[8] para mapear las clases de persistencia (que queramos guardar en la base de datos) haciendo uso de *DAOs*[9] para acceder a la base de datos. Como base de datos se utilizará *MySQL*[10].

Estudio de aplicaciones similares

En la actualidad existen buscadores de peluquerías por localidad o tipo de tratamiento que nos pueden ahorrar el tiempo de búsqueda.
Uno de ellos se trata de *treatwell*[11], este motor de búsqueda te permite filtrar resultados por localidad, tipo de trabajo y horario, pero en Jaén por el momento no hay resultados.

- Ventajas:
  - mayor visibilidad
  - grupo de clientes ya registrado
  - sistema multiplataforma
  - Buscador por localización

- Inconvenientes:
  - aplicación cerrada, no permite modificaciones por local
  - competitividad entre locales
  - no tiene la posibilidad de dar opinión sobre el local
  - mayor visibilidad mediante pago al buscador

[8] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
[9] https://es.wikipedia.org/wiki/Objeto_de_acceso_a_datos
[10] https://www.mysql.com/
[11] https://www.treatwell.es/

Ilustración 11: Imagen de la aplición web treatwell

Otra página que podemos solicitar es *Shortcuts*[12], en esta página podemos solicitar tener una aplicación software para nuestro negocio, este sistema software implementa las mismas funcionalidades que vamos a desarrollar para nuestro negocio, pero con el inconveniente de que este sistema será distribuido por toda aquella empresa que lo solicite, lo que hace que el software no se adecue a las necesidades específicas de cada local.

- Ventajas:
  - Software de gestión por establecimiento
  - Separación de aplicaciones de negocios
  - sistema multiplataforma
  - interfaz muy llamativa

- Inconvenientes:
  - aplicación cerrada, no permite modificaciones por local
  - no permite integración entre varios sistemas de gestión

Ilustración 12: Imagen de la aplición web Shortcuts

[12] https://www.shortcuts.es/

Por último, vamos a hablar de *barberapp*[13], se trata de una aplicación web, con la que el administrador del local tiene una gestión completa de las citas y los clientes pueden hacer la reserva de citas con la aplicación mediante su dispositivo móvil.

- Ventajas:
  - software de gestión por establecimiento
  - separación de aplicaciones de negocio
  - sistema multiplataforma
  - interfaz muy llamativa

- Inconvenientes:
  - No permite integración con varios sistemas de gestión
  - Solo dispone de método de reservas

Ilustración 13: Imagen de la aplicacion web barberApp

Historias de usuario

Una historia de usuario[14] es una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario. Estas historias de usuario se llevan a cabo para la especificación de requisitos en el uso de metodologías ágiles de desarrollo.

Al decidir realizar el proyecto en base a la metodología ágil SCRUM, las Historias de Usuario pueden ser modificadas en su declaración, en su objetivo o en sus criterios de aceptación. Pueden ser priorizadas u ordenadas por nuevos parámetros que le surjan al cliente. O pueden ser sacadas del Product Backlog al modificar el alcance o las tareas a desarrollar.
En el desarrollo de aplicaciones en base a metodologías ágiles, se llama Product Backlog a la lista de tareas correspondientes a la realización del proyecto, en nuestro caso al utilizar Scrum, El Product Backlog es una lista de

[13] https://www.barberapp.es/

características que han sido priorizadas, y contiene descripciones breves sobre todo lo que se va a desarrollar [15].
Las historias de usuario que vamos a describir serán numeradas como identificador, además de tener unos criterios de satisfacción que describen las acciones que hay que realizar una vez acabada la historia de usuario para validarla.

- **Aplicación móvil para clientes**

ID	TÍTULO	DESCRIPCIÓN
1	Registro en la aplicación	Como administrador quiero que cualquier usuario podrá registrarse en el sistema introduciendo en el registro los datos requeridos por la aplicación
2	Login en la aplicación	Como cliente quiero que una vez esté dado de alta en la aplicación, pueda entrar a las funcionalidades a través de introducir mi nombre de usuario y contraseña
3	Información del local	Como cliente quiero tener acceso a información del local, horario de apertura, contacto, ubicación, etc.
4	Recorrido al local	Como cliente quiero poder tener la opción de que desde la aplicación pueda ser guiado hasta la ubicación del local
5	Consulta de reservas	Como cliente quiero poder ver el horario de reservas del local, estando diferenciadas las citas ya reservadas de las disponibles.
6	Disponibilidad de reservas	Como administrador quiero que cada cliente solo pueda ver el horario de reservas en un intervalo de fechas definido
7	Reserva de citas	Como cliente quiero poder realizar una petición de cita disponible que desee, siempre que no tenga una reserva ya activa
8	Cancelación de citas	Como cliente quiero poder cancelar la cita activa y reservar una nueva
9	Límite de reservas	Como administrador quiero que cada usuario solo pueda tener activa una reserva a la vez, evitando así la mala gestión por parte de los clientes o un uso abusivo.
10	Consulta de historial	Como cliente quiero tener acceso a la información sobre su historial de reservas en la peluquería
11	Consulta de perfil	Como cliente quiero tener una sección de perfil dentro de la aplicación donde podrá consultar los datos almacenados sobre su perfil en el servidor
12	Gestión de perfil	Como cliente quiero tener una sección de perfil dentro de la aplicación donde podrá modificar los datos almacenados

		sobre su perfil en el servidor
13	Pago de la reserva	Como cliente quiero poder pagar la reserva una vez se haya completado el servicio
14	Bonificaciones	Como administrador quiero que cada cliente obtenga unos puntos de fidelidad que serán cambiados por premios o descuentos
15	Catálogo de productos	Como administrador quiero que cada cliente pueda ver el catálogo de productos disponibles en la tienda.

Tabla 1: Requisitos de la aplicación móvil para clientes

- **Aplicación de gestión administrativa**

ID	TÍTULO	DESCRIPCIÓN
1	Acceso al sistema	Como administrador del sistema solo yo puedo tener acceso a la aplicación
2	Persistencia de datos	Como administrador quiero que los datos de los clientes y las reservas realizadas se almacene en una base de datos
3	Consulta de clientes	Como administrador quiero poder consultar las reservas realizadas por cada cliente, así como el historial de reservas de cada cliente
4	Consulta de reservas	Como administrador quiero poder consultar el horario semanal de reservas
5	Reserva de citas	Como administrador quiero poder realizar una reserva ya sea con usuario registrado en la base de datos como sino
6	Aceptación de citas	Como administrador quiero poder aceptar las citas solicitadas por los clientes
7	Cancelación de reservas	Como administrador quiero poder cancelar una reserva en una determinada hora, independientemente del cliente que la haya reservado
8	Gestión de usuarios	Como administrador quiero poder consultar y tener control sobre los usuarios que acceden a la aplicación, denegando el acceso si se considera oportuno
9	Gestión de productos	Como administrador quiero poder consultar y editar los productos que van a poder verse dentro de la aplicación

Tabla 2: Requisitos de la aplicación para el administrador

Criterios de satisfacción

Los criterios de satisfacción[16] nos sirven para dar como válida una historia de usuario, es esta tabla podemos ver los que tiene cada historia de usuario de cada aplicación. Estos criterios de satisfacción comprenden las funcionalidades que debe tener cada requisito generado por el cliente y serán comprobados al término de cada tarea por el equipo de trabajo y al finalizar el sprint por el cliente.

- **Aplicación cliente**

ID	TÍTULO	DESCRIPCIÓN
1	Registro en la aplicación	• introducir datos del cliente en el formulario de registro y comprobar errores • inserción de datos del registro y comprobar registro con éxito
2	Login en la aplicación	• introducir datos del usuario en el formulario de login y comprobar errores • comprobación de la existencia del cliente y dar acceso a la aplicació
3	Información del local	• correcta visualización de los datos del local por parte del cliente
4	Recorrido al local	• Enlazar ubicación del cliente con la del local a través de la aplicación Google Maps
5	Consulta de reservas	• visualización del horario de reservas del local, los clientes solo tendrán acceso a 5 días del horario.
6	Disponibilidad de reservas	• visualización del estado de cada cita en el horario de reservas
7	Reserva de citas	• introducir datos del cliente en el formulario de reservas y comprobar errores • inserción de datos del registro y comprobar solicitud con éxito
8	Cancelación de citas	• selección correcta de la cita a cancelar • eliminado de datos y comprobación de éxito en la operación
9	Límite de reservas	• comprobación de la existencia de una única reserva activa por cliente
10	Consulta de historial	• visualización de las reservas realizadas por el cliente
11	Consulta de perfil	• visualización de los datos del cliente
12	Gestión de perfil	• edición de los datos del cliente

13	Pago de la reserva	• pagar factura de la reserva mediante la aplicación paypal
14	Bonificaciones	• sumar un punto a la cuenta de cada cliente y comprobación de éxito en la operación
15	Catálogo de productos	• visualización de catálogo de productos en venta.

Tabla 3: Criterios de satisfacción para la aplicación cliente

- **Aplicación servidor**

ID	TÍTULO	DESCRIPCIÓN
1	Acceso al sistema	• introducir datos del administrador en el formulario de login y comprobar errores • comprobación de usuario administrador y salida de errores
2	Persistencia de datos	• almacenar los datos en un sistema de almacenamiento de base de datos diseñado para este proyecto
3	Consulta de clientes	• introducir nombre del cliente en el formulario de búsqueda y comprobar errores • buscar cliente en la base de datos • visualización de los datos del cliente introducido
4	Consulta de reservas	• búsqueda de datos sobre las reservas realizadas para la semana actual • visualización correcta de datos
5	Reserva de citas	• introducción del nombre del cliente en el formulario de reservas y comprobar datos de entrada • comprobar existencia de cliente y creación de cliente nuevo en caso de no estarlo • introducir reserva del cliente y comprobación de éxito en la operación
6	Aceptación de citas	• seleccionar cita que tiene una solicitud de reserva • realizar aceptacion/cancelación sobre la cita seleccionada • comprobación de éxito en la operación
7	Cancelación de reservas	• introducción del nombre del cliente en el formulario de reservas y comprobar datos de entrada • introducir reserva del cliente y comprobación de éxito en la operación
8	Gestión de	• introducción del nombre del cliente en el formulario de

	usuarios	reservas y comprobar datos de entrada • bloquear acceso de usuario en la aplicación
9	Gestión de productos	• Consulta de productos activos para la venta en la aplicación

Tabla 4: Criterios de satisfacción para la aplicación del administrador

Planificación temporal

Una vez determinado el alcance el proyecto a través de las historias de usuario, el siguiente paso es determinar la viabilidad del mismo estableciendo los recursos necesarios para su desarrollo, tanto desde el punto de vista temporal como económico. En este apartado estudiaremos la planificación que se va a llevar a cabo para la realización del desarrollo de las historias de usuario, lo que conlleva la estimación, priorización y cálculo de velocidad.

A continuación, se van a estimar cada historia de usuario, esta estimación será un número de 1-10, siguiendo la serie de *Fibonacci [17],* un orden en escala, utilizaremos este método para medir una estimación sobre la complejidad relativa a cada historia de usuario.

Con esta estimación nos podremos hacer una idea de la dificultad de realizar cada historia de usuario, cada historia de usuario a su vez tendrá una estimación de horas de desarrollo [19], no hay una relación directa entre puntos de historia y horas de desarrollo, es decir no existe una regla p=punto de historia, h=horas de desarrollo x=p * h. Sin embargo, es probable que, al término del proyecto, el promedio muestre una relación parecida, pero nada evita, que una historia de 2 puntos de historia nos cueste 10 horas de desarrollo y otra también de 2 puntos nos tome 20 horas para implementarla.

Las horas de desarrollo se han basado en los puntos de historia de usuario, la experiencia del equipo de desarrollo y lo crítica que sea la tarea en cuestión.

En la tabla se descompone cada historia de usuario en tareas. Esta descomposición se debería realizar al sacar la historia de usuario del *Backlog*. Además de reflejar las horas de trabajo realizadas con cada perfil.

- **Aplicación cliente:**

Id	**Historia de usuario**	**Asignado a**
1	**Registro en la aplicación**	**4 ptos.**
1.1	Obtención de requisitos	Analista 4h.
1.2	Implementación de servicio REST	Programador 5h.

1.3	Implementación de la vista de usuario	Programador 5h.
1.4	Pruebas de funcionalidad	Programador 2h
2	**Login en la aplicación**	**5 ptos.**
2.1	Obtención de requisitos	Analista 4h.
2.2	Implementación de servicio REST	Programador 5h.
2.3	Implementación de la vista de usuario	Programador 5h.
2.4	Pruebas de funcionalidad	Programador 2h.
3	**Información del local**	**3 ptos.**
3.1	Obtención de requisitos	Analista 2h.
3.2	Diseño de la vista de usuario	Programador 2h.
3.3	Implementación de la vista de usuario	Programador 2h.
3.4	Pruebas de funcionalidad	Programador 1h
4	**Recorrido al local**	**7 ptos.**
4.1	Obtención de requisitos	Analista 3h.
4.2	Investigación sobre la API de google maps	Programador 6h.
4.3	Implementación de la conexión con la API	Programador 5h.
4.4	Pruebas de funcionalidad	Programador 4h.
5	**Consulta de reservas**	**6 ptos.**
5.1	Obtención de requisitos	Analista 4h.
5.2	Implementación de servicio REST	Programador 5h.
5.3	Implementación de estructura de almacenamiento	Programador 7h.
5.4	Diseño de la vista	Programador 4h.
5.5	Implementación de la vista	Programador 5h.
5.6	Pruebas de funcionalidad	Programador 2h.
6	**Disponibilidad de reservas**	**4 ptos**
6.1	método de selección de reservas	Programador 3h.
6.2	Diseño de la vista	Programador 2h.
6.3	Implementación de la vista	Programador 3h.

6.4	Pruebas de funcionalidad	Programador 2h.
7	**Reserva de citas**	**6 ptos.**
7.1	Obtención de requisitos	Analista 3h.
7.2	método de selección de reserva	Programador 2h.
7.3	Implementación de servicio REST	Programador 3h.
7.4	Diseño de la vista	Programador 2h.
7.5	Implementación de la vista	Programador 3h.
7.6	Pruebas de funcionalidad	Programador 2h.
8	**Cancelación de citas**	**5 ptos**
8.1	Obtención de requisitos	Analista 2h.
8.2	método de selección de reserva	Programador 2h.
8.3	Implementación de servicio REST	Programador 3h.
8.4	Diseño de la vista	Programador 2h.
8.5	Implementación de la vista	Programador 3h.
8.6	Pruebas de funcionalidad	Programador 2h.
9	**Límite de reservas**	**3 ptos.**
9.1	Obtención de requisitos	Analista 2h.
9.2	método de comprobación de reserva activa	Programador 2h.
9.3	Implementación de servicio REST	Programador 3h.
9.4	Pruebas de funcionalidad	Programador 2h.
10	**Consulta de historial**	**4 ptos.**
10.1	Obtención de requisitos	Analista 2h.
10.2	Implementación de servicio REST	Programador 3h.
10.3	Implementar estructura de almacenamiento	Programador 3h.
10.4	Diseño de la vista	Programador 2h.
10.5	Implementación de la vista	Programador 3h.
10.6	Pruebas de funcionalidad	Programador 2h.
11	**Consulta de perfil**	**4 ptos.**

11.1	Obtención de requisitos	Analista 2h.
11.2	Implementación de servicio REST	Programador 3h.
11.3	Implementar estructura de almacenamiento	Programador 2h.
11.4	Diseño de la vista	Programador 3h.
11.5	Implementación de la vista	Programador 3h.
11.6	Pruebas de funcionalidad	Programador 2h.
12	**Gestión de perfil**	**4 ptos.**
12.1	Obtención de requisitos	Analista 2h.
12.2	Almacenado de nuevos datos	Programador 2h.
12.3	Implementación de servicio REST	Programador 3h.
12.4	Pruebas de funcionalidad	Programador 2h.
13	**Pago de la reserva**	**9 ptos.**
13.1	Obtención de requisitos	Analista 3h.
13.2	Obtención de datos del cliente y cuenta	Programador 4h.
13.3	Establecer conexión con la playpal	Programador 5h.
13.4	Pruebas de funcionalidad	Programador 3h.
14	**Bonificaciones**	**5 ptos.**
14.1	Obtención de requisitos	Analista 2h.
14.2	Implementación de servicio REST	Programador 5h.
14.3	Implementación de envío de notificación al cliente	Programador 5h.
14.4	Pruebas de funcionalidad	Programador 2h
15	**Catálogo de productos**	**4 ptos.**
15.1	Obtención de requisitos	Analista 4h.
15.2	Diseño de la vista	Programador 4h.
15.3	Implementación de la vista	Programador 4h.
15.4	Pruebas de funcionalidad	Programador 2h.

Tabla 5: Asignación de tareas de la aplicación del cliente

- **Aplicación servidor**

Id	Historia de usuario	Asignado a
1	**Acceso al sistema**	**5 ptos.**
1.1	Obtención de requisitos	Analista 4h.
1.2	Implementación de servicio REST	Programador 5h.
1.3	Implementación de la vista de usuario	Programador 5h.
1.4	Pruebas de funcionalidad	Programador 2h.
2	**Persistencia de datos**	**7 ptos.**
2.1	Obtención de requisitos	Analista 4h.
2.2	Diseño de base de datos	Programador 5h.
2.3	Implementación de base de datos	Programador 5h.
2.4	Pruebas de funcionalidad	Programador 3h.
3	**Consulta de clientes**	**5 ptos.**
3.1	Obtención de requisitos	Analista 3h.
3.2	método de selección de cliente	Programador 2h.
3.3	Implementación de servicio REST	Programador 3h.
3.4	Diseño de la vista	Programador 2h.
3.5	Implementación de la vista	Programador 3h.
3.6	Pruebas de funcionalidad	Programador 2h.
4	**Consulta de reservas**	**4 ptos.**
4.1	Obtención de requisitos	Analista 2h.
4.2	Implementación de servicio REST	Programador 3h.
4.3	Implementar estructura de almacenamiento	Programador 3h.
4.4	Diseño de la vista	Programador 2h.
4.5	Implementación de la vista	Programador 3h.
4.6	Pruebas de funcionalidad	Programador 2h.
5	**Reserva de citas**	**5 ptos.**

5.1	Obtención de requisitos	Analista 2h.
5.2	Implementar método de selección de reserva y cliente	Programador 3h.
5.3	Diseño de la vista	Programador 3h.
5.4	Implementación de la vista	Programador 4h.
5.5	Pruebas de funcionalidad	Programador 2h.
6	**Aceptación de reserva**	**5 ptos.**
6.1	Obtención de requisitos	Analista 2h.
6.2	Diseñar vista de solicitud de reserva	Programador 3h.
6.3	Implementar método de aceptación de cita	Programador 2h.
6.4	Pruebas de funcionalidad	Programador 2h.
7	**Cancelación de reservas**	**6 ptos.**
7.1	Obtención de requisitos	Analista 2h.
7.2	Implementar método de selección de reserva y cliente	Programador 3h.
7.3	Implementación de servicio REST	Programador 4h.
7.4	Diseño de la vista	Programador 3h.
7.5	Implementación de la vista	Programador 4h.
7.6	Pruebas de funcionalidad	Programador 2h.
8	**Gestión de usuarios**	**7 ptos.**
8.1	Obtención de requisitos	Analista 4h.
8.2	Implementar método de selección de cliente	Programador 3h.
8.3	Implementación de servicio REST	Programador 4h.
8.4	Diseño de la vista	Programador 2h.
8.5	Implementación de la vista	Programador 3h.
8.6	Pruebas de funcionalidad	Programador 2h.
9	**Gestión de productos**	**4 ptos.**
9.1	Obtención de requisitos	Analista 4h.
9.2	Diseño de la vista	Programador 2h.
9.3	Implementación de la vista	Programador 3h.

9.4	Pruebas de funcionalidad	Programador 2h.

Tabla 6: Asignación de tareas de la aplicación del administrador

A continuación, se mostrará una tabla resumen de los tiempos totales para llevar a cabo la solución del proyecto, además de una separación de horas de trabajo por roles.

Trabajador	**Horas**
Analista	66 horas
Programador	270 horas
	330 horas

Tabla 7: Resumen de asignación de tareas por roles

En el siguiente punto se desglosa la organización del proyecto, adaptándose al tiempo disponible para el desarrollo del proyecto.

Priorización de historias de usuario

En las metodologías ágiles la planificación se realiza constantemente a lo largo del producto. De esta forma se adapta al cambio, en vez de seguir un plan definido. Durante la gestión de requisitos de un desarrollo de un producto o servicio, además de realizar una correcta recogida y escritura de los requisitos, adquiere una gran importancia realizar una priorización de los requisitos. Esta tarea no es nada sencilla, debido a que podemos encontrar intereses diferentes, o incluso opuestos.

La priorización de requisitos se hace para analizar las áreas claves que se tienen en cuenta antes de tomar una decisión importante, dentro de ello hay una serie de técnicas que se emplean para esta priorización.

Una de las técnicas más utilizadas en la priorización de requisitos es la conocida como técnica MoSCoW [20]. Es una técnica de priorización de requisitos basada en el hecho de que, aunque todos los requisitos se consideren importantes, es fundamental destacar aquellos requisitos vitales que aportan un mayor valor al negocio y que son considerados obligatorios

- **M – Must**: Requisitos que debe tener la solución. Son los requisitos mínimos que hacen usable la solución.
    - Aplicación cliente

Id	Nombre	Puntos de historia

1	Registro en la aplicación	4 ptos.
2	Login en la aplicación	5 ptos.
3	Información del local	3 ptos.
5	Consulta de reservas	6 ptos.
6	Disponibilidad de reservas	4 ptos.
7	Reserva de citas	6 ptos.
8	Cancelación de citas	5 ptos.
9	Límite de reservas	3 ptos.

Tabla 8: Requisitos imprescindibles de la aplicación del cliente

- Aplicación servidor

Id	Nombre	Puntos de historia
1	Acceso al sistema	5 ptos.
2	Persistencia de datos	7 ptos.
3	Consulta de clientes	5 ptos.
4	Consulta de reservas	4 ptos.

Tabla 9: Requisitos imprescindibles de la aplicación del administrador

- **S – Should**: Requisitos que debería incluir la solución. Requisitos importantes, pero no obligatoriamente necesarios.

  - Aplicación cliente

Id	Nombre	Puntos de historia
10	Consulta de historial	4 ptos.
11	Consulta de perfil	4 ptos.
12	Gestión de perfil	4 ptos.

Tabla 10: Requisitos que debe tener la aplicación del cliente

- Aplicación servidor

Id	Nombre	Puntos de historia
5	Reserva de citas	5 ptos.
6	Cancelación de reservas	6 ptos.
9	Gestión de productos	4 ptos.

Tabla 11: Requisitos que debe tener la aplicación del administrador

- **C – Could**: Requisitos que podría incluir la solución. Estos requisitos mejorarían el rendimiento del servicio, pero podrían ser eliminados fácilmente.

  - Aplicación cliente

Id	Nombre	Puntos de historia
4	Recorrido al local	7 ptos.
15	Catálogo de productos	4 ptos.

Tabla 12: Requisitos que podría tener la aplicación del cliente

  - Aplicación servidor

Id	Nombre	Puntos de historia
8	Gestión de productos	4 ptos.

Tabla 13: Requisitos que podría tener la aplicación del administrador

- **W – Won't**: Requisitos que no se van a hacer (por el momento), pero que en un futuro podrían incorporarse, ya que tras la estimación de puntos de historia se ha comprobado que no es viable su planificación dentro de los límites establecidos para el TFG.

  - Aplicación cliente

Id	Nombre	Puntos de historia
13	Pago de la reserva	9 ptos.
14	Bonificaciones	5 ptos.

Tabla 14: Requisitos que no tendrá la aplicación del cliente

  - Aplicación servidor

Id	Nombre	Puntos de historia
7	Gestión de usuarios	7 ptos.

Tabla 15: Requisitos que no tendrá la aplicación del administrador

Estimación de velocidad

Para estimar el tiempo necesario que necesitaremos para desarrollar las historias de usuario vamos a utilizar el concepto físico de velocidad como la magnitud determinada por la cantidad de trabajo realizada en un periodo de tiempo. Velocidad en scrum[21], es la cantidad de puntos de la historia que el equipo de desarrollo puede completar durante un Sprint.

La estimación de la velocidad se realiza para obtener una guía del trabajo que se va a realizar en el próximo sprint, otro factor también por el que se realiza esta estimación, es para tener una idea aproximada de la fecha de finalización de la aplicación, se considera algo orientativo ya que puede haber cambios en el proyecto durante su realización o incluir nuevos requisitos.

Se van a tener en cuenta todas las historias de usuario:

- La aplicación cliente cuenta con 72 puntos de historia.
- La aplicación servidor cuenta con 48 puntos de historia

La fecha de comienzo será el 26 de diciembre al 7 de Julio, 27 semanas
La primera semana será invertida en investigación y toma de contacto de la tecnología ionic. Durante esta investigación se procedió a la instalación de componentes necesarios[22] para su correcto funcionamiento, este framework utiliza Cordova para compilar e implementar como una aplicación nativa.

La mayoría de las herramientas en la CLI se basan en Node y se administran a través de npm. La forma más rápida de instalar Node y NPM[14] en su máquina es a través del instalador NodeJS[15]. Una vez configurado, procedemos con la instalación del cliente de ionic y Cordova.

Las últimas dos semanas serán invertidas en la realización de la memoria de la aplicación, lo que nos deja 24 semanas para la realización de la aplicación.

Como tenemos que desarrollar una aplicación cliente y un servidor, vamos a realizar primero la aplicación servidor, para esta aplicación vamos a utilizar 10 semanas, con una estimación de 2 semanas por sprint, tenemos 5 sprint.

[14] https://www.npmjs.com/
[15] https://nodejs.org/en/

La velocidad será de: 48 puntos/5 = 9,6 puntos de historia por sprint, que vamos a redondear a 10 puntos de historia por sprint.
En la siguiente tabla se expone la duración del desarrollo de la aplicación por sprint.

Sprint	Inicio	Final	Puntos	Velocidad	Historia
1	02/01/2018	12/01/2018	12	120%	1,2
2	15/01/2018	26/01/2018	9	90%	3,4
3	29/01/2018	09/02/2018	10	100%	5,6
4	12/02/2018	23/02/2018	10	100%	7,9
5	26/02/2018	09/03/2018	7	70%	8

Tabla 16: Duración del desarrollo de la aplicación del administrador

Una vez terminada la aplicación servidor ya podemos desarrollar la aplicación cliente, para ello se va a evaluar en la siguiente tabla el desarrollo de los sprints, como tenemos 14 semanas para el desarrollo de la aplicación, vamos a dividir los sprint en 2 semanas como en la aplicación servidor, lo que nos llevaría a una velocidad de: 72/7 = 10,4 puntos de historia por sprint, que redondeamos a 11 puntos de historia por sprint.

Sprint	Inicio	Final	Puntos	Velocidad	Historia
1	12/03/2018	23/03/2018	9	80%	1,2
2	26/03/2018	06/04/2018	10	90%	5,6
3	09/04/2018	20/04/2018	11	100%	3,10,11
4	23/04/2018	04/05/2018	10	90%	7,12
5	07/05/2018	18/05/2018	11	100%	4,7
6	21/05/2018	01/06/2018	12	110%	8,13
7	04/06/2018	15/06/2018	9	90%	14,15

Tabla 17: Duración del desarrollo de la aplicación del cliente

Hay sprint en los que se supera la velocidad estimada, esto nos indica que habrá que trabajar más para llevar a cabo estos sprint.

Evolución de estimaciones

Al término del primer sprint, no se pudieron entregar los puntos se historia que se habían planteado, solo se pudo entregar la historia 2 de usuario, esto nos hizo recapacitar y volver a estructurar los sprint para el desarrollo de la aplicación.

En el siguiente gráfico podemos ver como la planificación principal fue alterada por la planificación definitiva.

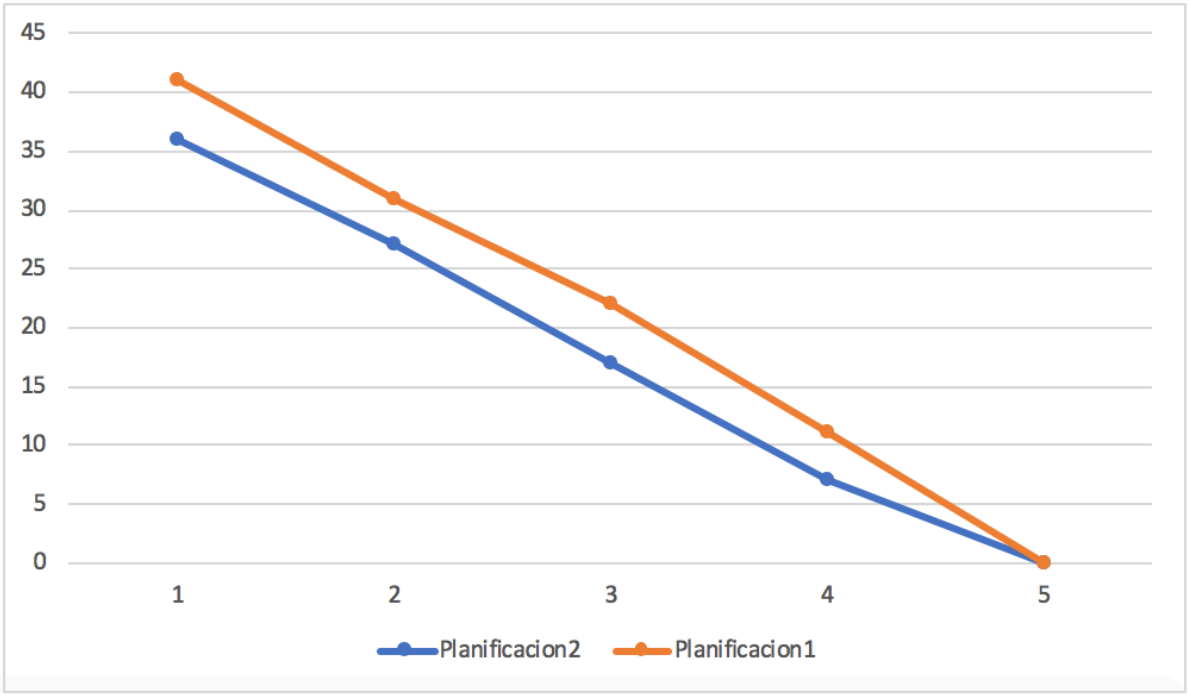

Ilustración 14: diagrama burn down de la aplicación servidor

Para ajustar el desarrollo del proyecto, se acordó con el cliente no desarrollar las historias de usuario que se encuentran en el wont. Si le quitamos la primera historia con el primer sprint concluido, ahora la suma de puntos de historia para la aplicación administrador será de 41, la velocidad por lo tanto es de 41/4=11 puntos de historia por sprint.
Así quedaría la tabla para el desarrollo de la aplicación administrador:

Sprint	Inicio	Final	Puntos	Velocidad	Historia
1	02/01/2018	12/01/2018	7	70%	2
2	15/01/2018	26/01/2018	10	90%	1,3
3	29/01/2018	09/02/2018	9	90%	4,5
4	12/02/2018	23/02/2018	11	100%	6,7
5	26/02/2018	09/03/2018	11	100%	8,9

Tabla 18: Duración del desarrollo ajustado de la aplicación del administrador

Por otra parte, la aplicación cliente, tiene unas entregas de sprint según lo previsto, hasta la entrega 6, ahí hay un retraso y no se entrega en su totalidad, por lo que se decide con el cliente no implementar las historias de usuario que se encuentran en el wont. Con lo que nos quedarían 9 puntos de historia por completar, la velocidad en este caso es de 9/2 = 5 puntos de historia por sprint.

Mediante el diagrama de Burndown, veremos el tiempo que falta para terminar el trabajo.
Se ha desarrollado un gráfico, al igual que con la aplicación servidor, donde podemos ver cómo afectó este hecho a la planificación principal y la planificación definitiva.

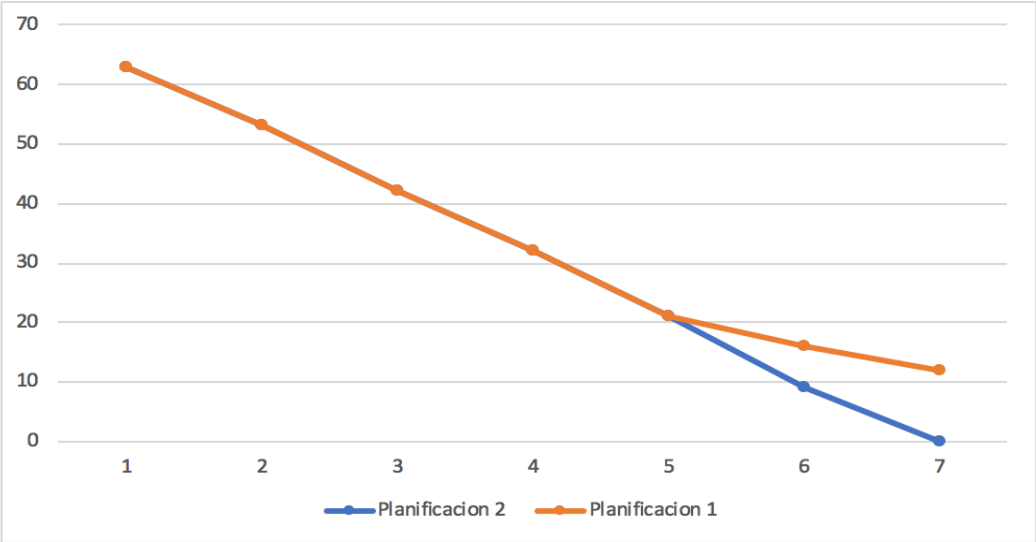

Ilustración 15: diagrama burn down de la aplicación cliente

Sprint	Inicio	Final	Puntos	Velocidad	Historia
1	12/03/2018	23/03/2018	9	80%	1,2
2	26/03/2018	06/04/2018	10	90%	5,6
3	09/04/2018	20/04/2018	11	100%	3,10,11
4	23/04/2018	04/05/2018	10	90%	7,12
5	07/05/2018	18/05/2018	11	100%	4,7
6	21/05/2018	01/06/2018	5	100%	8
7	04/06/2018	15/06/2018	4	90%	16

Tabla 19: Duración del desarrollo ajustado de la aplicación del cliente

Estudio de viabilidad económica

La viabilidad es la medida del beneficio obtenido en una organización gracias al desarrollo de un sistema de información, con este estudio podremos conocer si el proyecto aportará los beneficios que esperamos[23].
El estudio de viabilidad realiza una estimación de si las necesidades de usuario identificadas que se pueden satisfacer utilizando las tecnologías

software y hardware actuales. Este estudio a su vez, nos ayudará a evaluar la probabilidad de éxito antes del comienzo del proyecto y minimizar los riesgos.
A continuación, se detalla en una tabla el coste por unidad y total de software y hardware que se ha requerido para la realización del proyecto
El coste total del proyecto será aumentado un 20% para obtener beneficios.
El software empleado en su mayoría es gratuito.

Software	**Precio**	**Tiempo de uso**	**Precio de uso**
Netbeans 8.2	0€	6 meses	0€
Postman	0€	6 meses	0€
Visual Studio Code	0€	6 meses	0€
Visual Paradigm Community Edition	0€	6 meses	0€
Google Chrome	0€	6 mese	0€
Microsoft Office 365 personal	7€/mes	6 meses	42€
Total			42€

Tabla 20: Coste de software en el desarrollo de la aplicación

La estimación del Hardware se ha hecho en base a su tiempo de vida útil, dividido por el tiempo de uso al que le vamos a someter durante el desarrollo del proyecto:

Hardware	**Coste**	**vida**	**Precio**
MacBook Pro	1300€	7 años	92,85€
Huawei P8 Lite	150€	2 años	37,5€
			130,35€

Tabla 21: Coste de hardware en el desarrollo de la aplicación

El coste relativo al salario del personal será calculado en la siguiente tabla, en ella están marcadas las horas de trabajo totales para el desarrollo de la aplicación y los datos del salario que hemos obtenido para los dos perfiles de empleados que hay en el proyecto: para el Analista[24] y para el programador[25]

Cargo	**Salario mes**	**Salario hora**	**horas día**	**total horas**	**Coste**
Analista	2050€	12,8125	5	66	845,625€
Programador	1330€	8,3125	5	285	2369,625€
					3214,6875€

Tabla 22: Total coste de mano de obra

Una vez calculado los apartados anteriores vamos a calcular el coste total del proyecto:

Coste	**Precio**
Software	42€
Hardware	130,35€
Mano de obra	3214,6875€
Total, sin beneficio	3387,0375€
Beneficio 20%	677,3375€
Total	**4064,025€**

Tabla 23: Coste total de la aplicación

Modelo de dominio

Una vez determinada la viabilidad del proyecto, lo siguiente será realizar un modelo de dominio de la solución. Un modelo de dominio[26] en la resolución de problemas e ingeniería de software, es un modelo conceptual de todos los temas relacionados con un problema específico. En él se describen las distintas entidades, sus atributos, papeles y relaciones, además de las restricciones que rigen el dominio del problema.
A partir de este modelo se comenzará el estudio del diseño del proyecto.

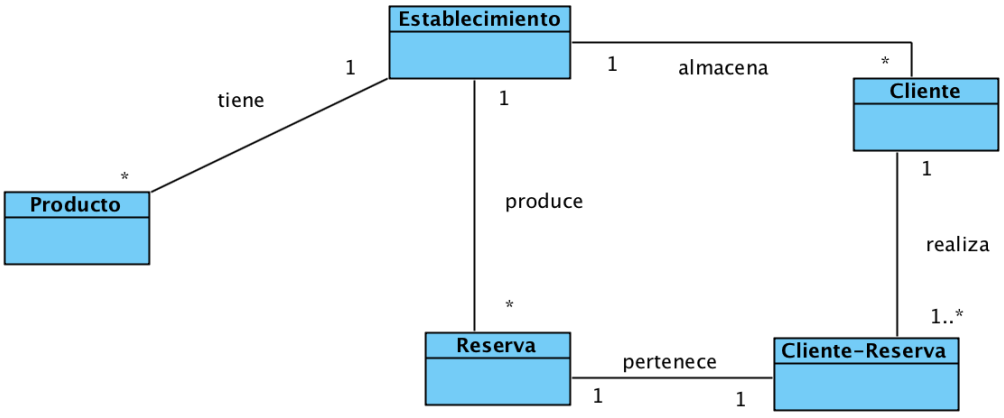

Ilustración 16: modelo de dominio del proyecto

Se ha establecido la entidad Establecimiento para almacenar la información suministrada por el cliente de la aplicación y poder aumentar de números de

locales por el cliente en futuras actualizaciones, los productos están asociados al establecimiento que los tiene disponibles a la venta. Por otra parte, las reservas de un establecimiento solo pueden estar asociadas a un cliente que esté almacenado en la entidad cliente, con ello tendremos un control de las reservas asociadas a cada cliente.

3. Diseño

El diseño de software se define como el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistemas con los suficientes detalles como para permitir su realización física. El objetivo del diseñador es producir un modelo o representación de una entidad que será construida más adelante, el diseño del Software tiene un impacto directo sobre la capacidad del sistema para cumplir o no el total de requerimientos establecidos. Un error de diseño en esta fase puede acarrear problemas en todo el proyecto y provocar múltiples cambios durante la implementación.

Esto nos permite poder crear varios modelos del sistema que forman parte del plan de desarrollo de la solución, estos diseños pueden ser evaluados en relación a su calidad y mejorarse antes de comenzar a generar código. El diseño es definido como: "*El proceso de definición de la arquitectura, componentes, interfaces y otras características de un sistema o componente que resulta de este proceso*"[27].

En este apartado se desarrollarán los diagramas de entidad-relación, así como los de clases y secuencia, además de las interfaces que se van a implementar en el proyecto.

Para el desarrollo de la solución se empleará una metodología orientada a objetos, en las que los objetos se crearán mediante clases relacionadas entre sí. Por lo tanto, se utilizarán los diagramas de clases que proporciona la metodología UML para establecer las relaciones existentes entre las clases que componen el sistema. Estos diagramas ofrecen un estándar en lenguaje gráfico para describir el modelo del sistema, en él se incluirán también los métodos que se utilizarán en el sistema.

En este punto se definirán en primer lugar el modelo de datos sobre un sistema de gestión de bases de datos relacional, para ello realizaremos un modelo entidad-relación de la solución, luego se realizará el diagrama de clases en el que se definirá la estructura del sistema.

Luego se desarrollará el diagrama de secuencia, que es el diagrama que es usado para modelar interacción entre objetos mediante el sistema UML [28].
Por último, estarán los diseños de las interfaces, tanto de la aplicación del administrador como de la aplicación del cliente, en este punto se incluirán el diagrama de estados de la aplicación y las vistas de las interfaces de las aplicaciones.

Modelo de entidad-relacion

El modelo de entidad relación fueron creados por Peter Chen. Estos modelos han sido la base para diversas metodologías sobre análisis y diseño de sistemas, herramientas de ingeniería de software asistida por computador (CASE) y repositorios de sistemas.

Este modelo, está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados entidades, y de relaciones entre esos objetos [29], ya que es una herramienta para el modelado de datos que permite representar las entidades relevantes de un sistema de información, así como sus interrelaciones y propiedades.

Los elementos que van a componer el modelo de entidad-relación son los siguientes [30]:

- Entidades: representan los objetos que van a tener persistencia de datos.
- Atributos: definen características que posee cada entidad, una entidad puede tener varios atributos.
- relaciones: son vínculos que nos permiten definir dependencias entre entidades, estas relaciones pueden ser de distintos tipos según su relación, en nuestra solución tendremos relaciones de los siguientes tipos:
    - Uno a uno: una entidad se relaciona únicamente con otra.
    - Uno a muchos: determina que un registro de una entidad puede estar relacionado con varios de otra entidad.

En nuestro caso, como hemos utilizado el framework Spring, éste admite la integración con diversos sistemas de persistencia como Hibernate, Java Persistence API (JPA), Java Data Objects (JDO) e iBATIS SQL Maps para la administración de recursos, implementaciones de objetos de acceso a datos (DAO) y estrategias de transacción.

El mapeado objeto-relacional, definido por la especificación JPA, proporciona persistencia en una base de datos relacional a una aplicación orientada a objetos usando una interfaz de alto nivel, ya que tanto el mapeo de los objetos a las tablas relacionales como la generación del código SQL para las operaciones de persistencia es automático. Vamos a utilizar este mapeado de clases porque es un código muy sencillo, legible además de no tener que utilizar código SQL y su uso lo hemos estudiado en la asignatura desarrollo de aplicaciones empresariales.

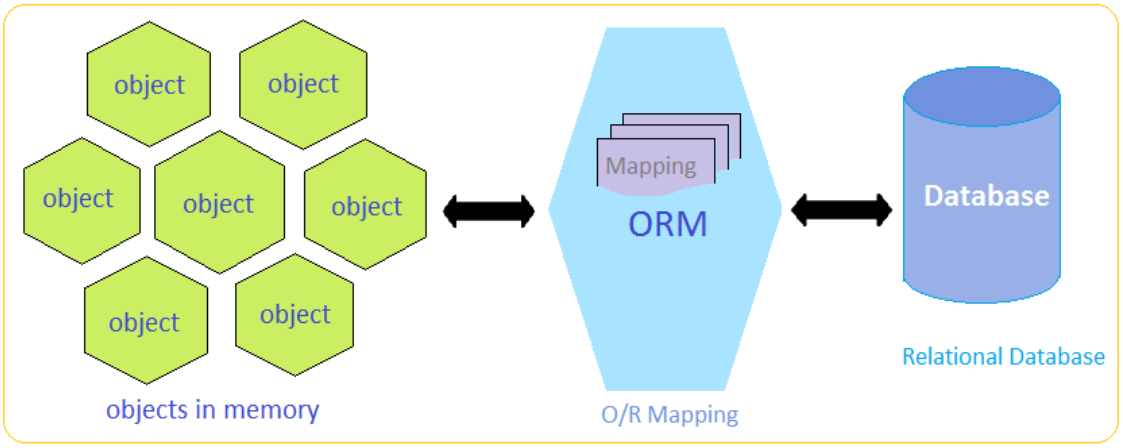

Ilustración 17: mapeo relacional de objetos. Fuente: https://hub.packtpub.com/welcome-spring-framework/

Cuando se ejecute la aplicación de Spring, las clases con las anotaciones respectivas que requieran persistencia, se crearán automáticamente en la base de datos, al igual que el diagrama de entidad-relación que será generado automáticamente por JPA a partir de la información proporcionada por el modelo de dominio.

Como podemos observar las tablas están en 3a forma normal, porque están en 2FN (los atributos no clave depende de toda la clave principal) y no existe ninguna dependencia transitiva entre los atributos que no son clave.

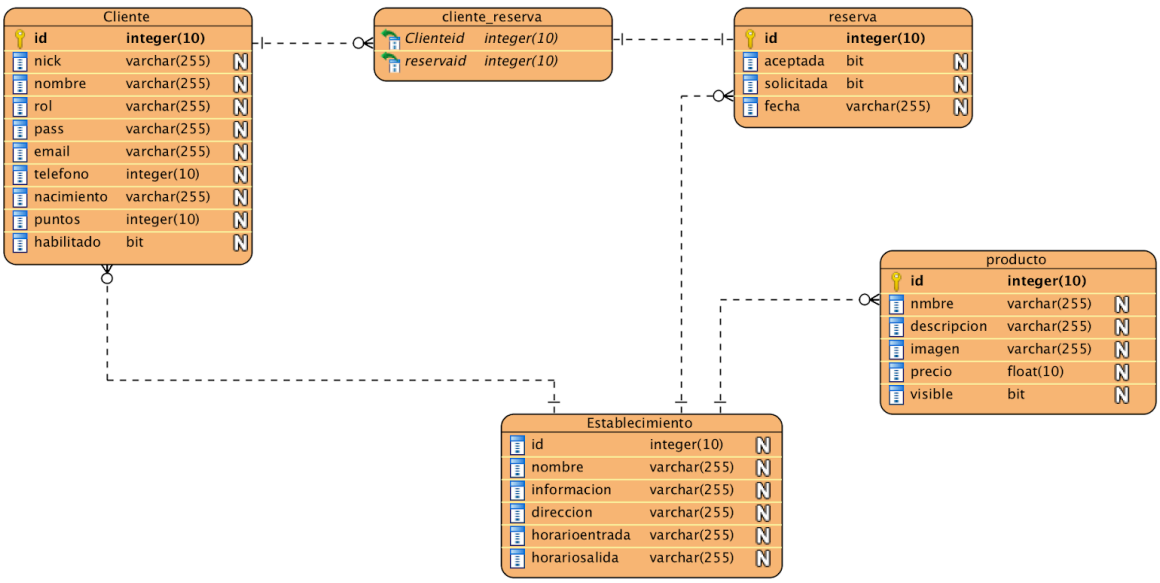

Ilustración 18: diagrama de bases de datos

Diagrama de clases

Un diagrama de clases[31] es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos.

Como el sistema propuesto utilizará internet, se utilizará una arquitectura cliente-servidor para el intercambio de datos entre la aplicación del cliente, la aplicación del administrador y la parte servidor.

Las dos aplicaciones, se conectan con el servidor para obtener los datos actualizados del establecimiento, a continuación, se describe en el siguiente diagrama la relación que se establecen entre ellos.

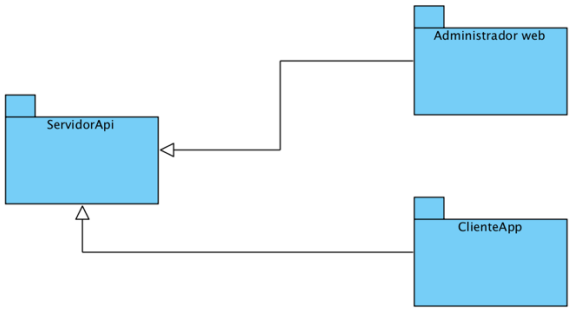

Ilustración 19: diagrama de relación de las aplicaciones del proyecto

Por tal motivo, se tratará el diseño de cada una de las partes de forma independiente en los siguientes apartados.

- **Aplicación servidor**

La aplicación la vamos a desarrollar con Spring, por lo que la estructura del servidor va a seguir el esquema de *Spring Boot*[16] que es un empaquetado para crear aplicaciones JEE de manera rápida y con todo incluído. Es una de las tecnologías dentro del mundo de Spring que nos permite desarrollar una aplicación que contiene los servicios *RESTFUL* del proyecto, este servicio define unos principios arquitectónicos por los cuales diseñan unos servicios con los que podemos obtener recursos del sistema.

La Transferencia de Estado Representacional (REST - Representational State Transfer) fue ganando amplia adopción en toda la web como una alternativa más simple a SOAP*5* y a los servicios web basados en el Lenguaje de Descripción de Servicios Web (Web Services Description Language - WSDL)[32].

[16] https://spring.io/projects/spring-boot

REST define un set de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP.
Una implementación de un servicio REST sigue estos cuatro principios fundamentales:

- utiliza los métodos HTTP de manera explícita
- no mantiene estado
- expone URIs con forma de directorios
- transfiere XML, JavaScript Object Notation (*JSON*), o ambos

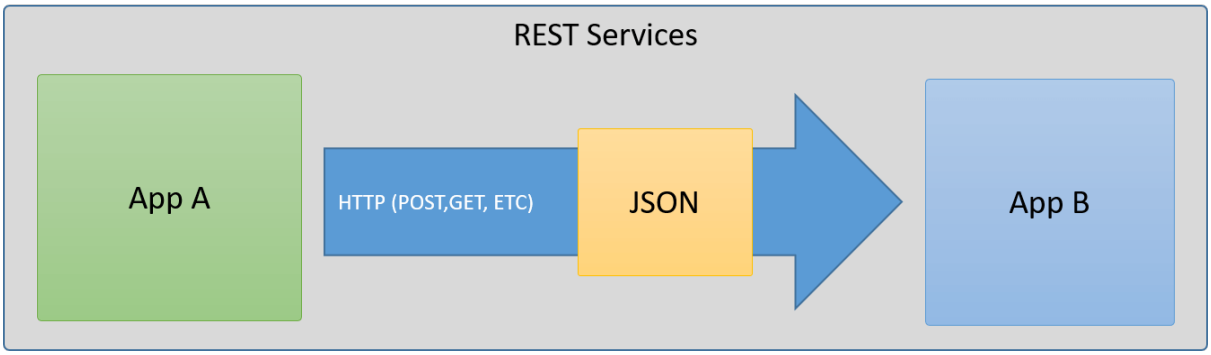

Ilustración 20: servicios rest
Fuente:https://www.oscarblancarteblog.com/2017/03/06/soap-vs-rest-2/

Este servicio RESTFUL hace uso de unas clases DAO(data Access objetc)[33] ya que es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos. La ventaja de usar estos componentes es que no requiere conocimiento directo del destino final de la información que manipula.

El servicio RESTFUL hará uso de los DAO de cada objeto mapeado en la persistencia de datos para la gestión de los datos almacenados en memoria.

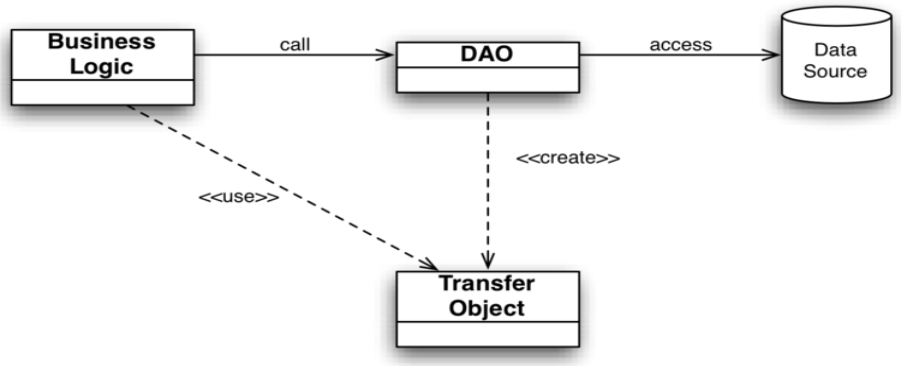

Ilustración 21: patron DAO Fuente:https://gl.wikipedia.org/wiki/Data_access_object

Cuando tenemos que devolver información por medio del servicio RESTFUL a la aplicación utilizamos las clases DTOS, estas clases son objetos que definen cómo se van a enviar los datos a través de la red. Esta información se

encapsula en el objeto que contenga los datos que queremos enviar y se transmite al cliente.

Este es el diagrama final de clases de la aplicación servidor

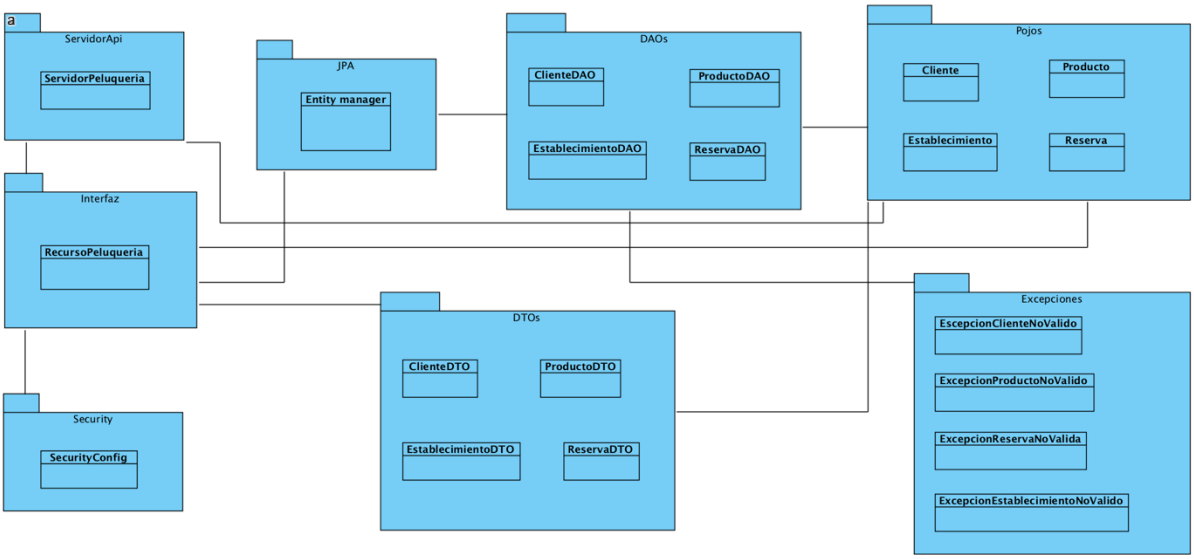

Ilustración 22: diagrama de clases servidor

En el diagrama podemos observar como el servicio API recoge la llamada y conecta con la interfaz, esta interfaz es el sistema que tiene los métodos de llamada al *entity manager*, que a su vez posee los *DAOs* para la recogida de datos seleccionados. Una vez recogidos los datos, la interfaz crea los DTOs con los que se envían los datos al cliente del servicio.

- **Aplicación administrador**

Para desarrollar la aplicación web del administrador hemos utilizado Spring mvc, este framework tiene sus propios módulos que conforman un framework para el desarrollo de aplicaciones Java basadas en web, que sigue el patrón de diseño MVC.

En esta aplicación se encuentran las vistas que serán utilizadas por el administrador del establecimiento, estas vistas se comunican con el controlador mediante la etiqueta @RequestMapping, en esta etiqueta introducimos el valor de la llamada asignada en la vista destinada al administrador, con la que se conecta al controlador.

En el modelo Spring MVC las peticiones llegan al controlador y este realiza las funciones necesarias asignadas a la petición solicitada por el administrador y de ahí hace uso del modelo para realizar las peticiones a los servicios web implementados en la aplicación servidor por medio de *restTemplate*[17].

[17] https://www.baeldung.com/rest-template

A continuación, se muestra un diagrama de la estructura que seguirá el proyecto en base a la arquitectura MVC que implementa Spring.

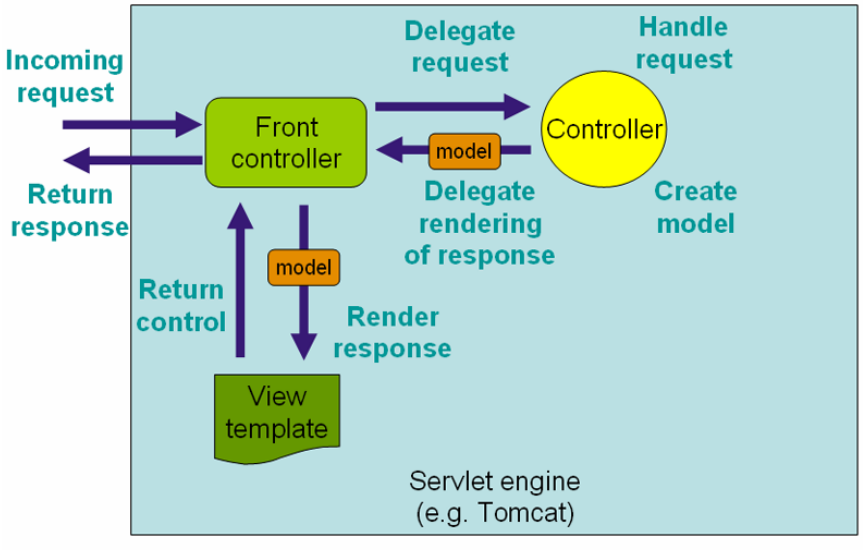

Ilustración 23: estructura aplicacion web spring mvc Fuente: http://websystique.com/spring-4-mvc-tutorial/

Spring nos provee de un cliente Rest, RestTemplate, esta plantilla se encarga de realizar la conexión http, lo único que hace falta es pasarle la url del servicio del administrador con el que conectar. El mismo RestTemplate gestiona sus propios 'messageConverters', con los que parsear los datos enviados y recibidos de/a json, por ejemplo. Solo es necesario añadir en nuestro proyecto la dependencia necesaria (APPLICATION_JSON, en nuestro caso).

En esta imagen se muestra la comunicación con la aplicación del servidor.

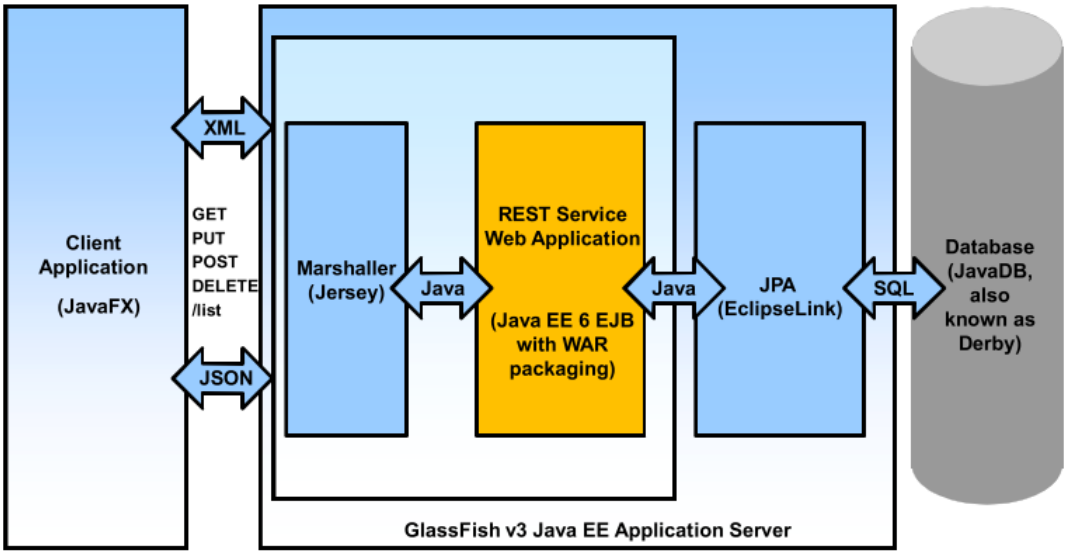

Ilustración 24: esquema comunicación adminsitrador, servidor Fuente: https://www.accesa.eu/resources/javafx-and-restful-web-services-communication

A continuación, se muestra el diagrama de clases de la aplicación del administrador, en él podemos ver la estructura de la aplicación.

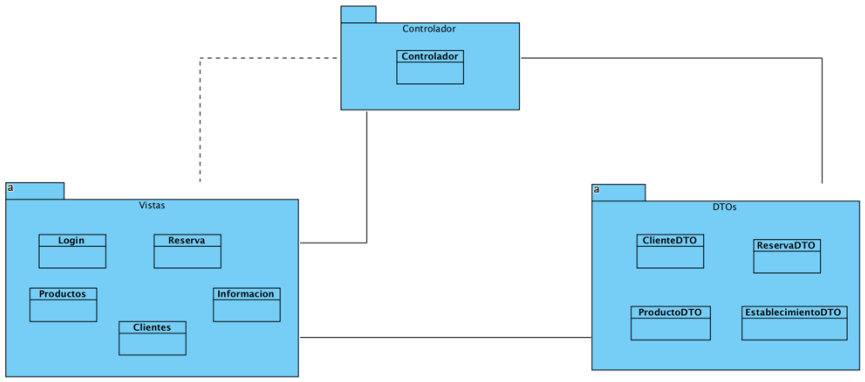

Ilustración 25: diagrama de clases aplicación administrador

El controlador gestiona el evento que llega de la acción provocada por el cliente y envía la vista, esta vista envía de vuelta al controlador la llamada solicitada, el controlador obtiene los DTOs correspondientes y los envía de vuelta a la vista.

- **Aplicación cliente**

En el cliente hemos utilizado el framework de tecnología híbrida ionic que está desarrollado sobre AngularJS y Apache Cordova.

La estructura del framework utiliza el patrón MVVM (Model View ViewModel) [34] con la que tratamos de desacoplar la interfaz de usuario de la lógica de la aplicación.

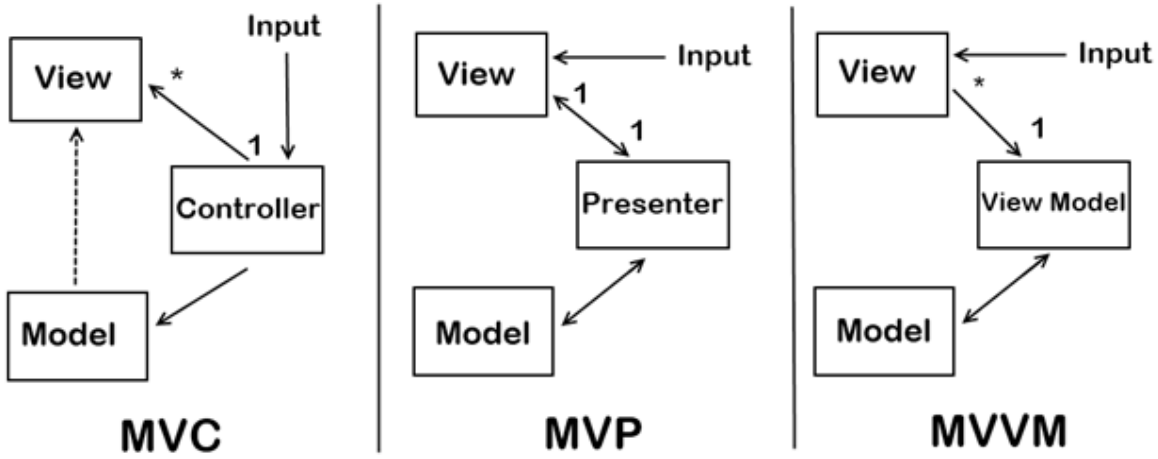

Ilustración 26: mvc vs mvp vs mvvm
Fuente:http://geekswithblogs.net/dlussier/archive/2009/11/21/136454.aspx

En nuestro caso, la utilización de este patrón tiene las siguientes ventajas:

- La realización de pruebas unitarias para la vista y el modelo se pueden realizar por separado, sin hacer uso de la vista.

- Durante el proceso de desarrollo, se puede realizar el trabajo de diseño y desarrollo en paralelo, esto nos proporciona mucha independencia a la hora de desarrollar la solución final.

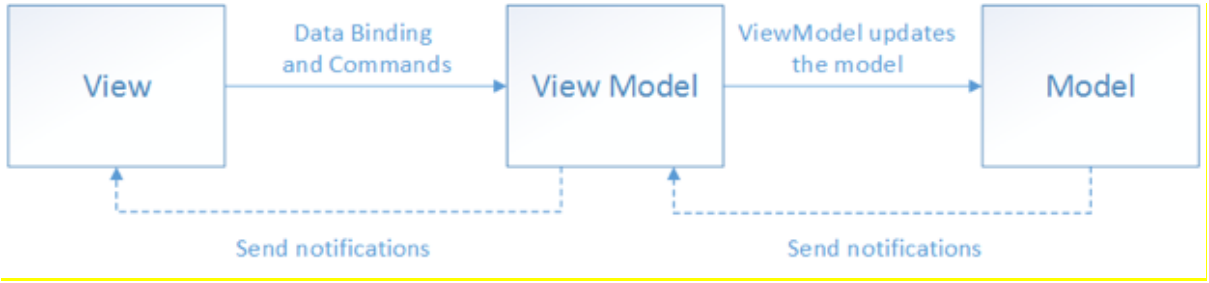

Ilustración 27: patrón Model-View-ViewModel
Fuente:http://geekswithblogs.net/dlussier/archive/2009/11/21/136454.aspx

Como la vista está comunicada con el modelo a través de un viewModel si hay un cambio en el modelo, éste se refleja en la vista y viceversa.

A continuación, se muestra el diagrama de clases de la aplicación del cliente, en él podemos ver la estructura de la aplicación.

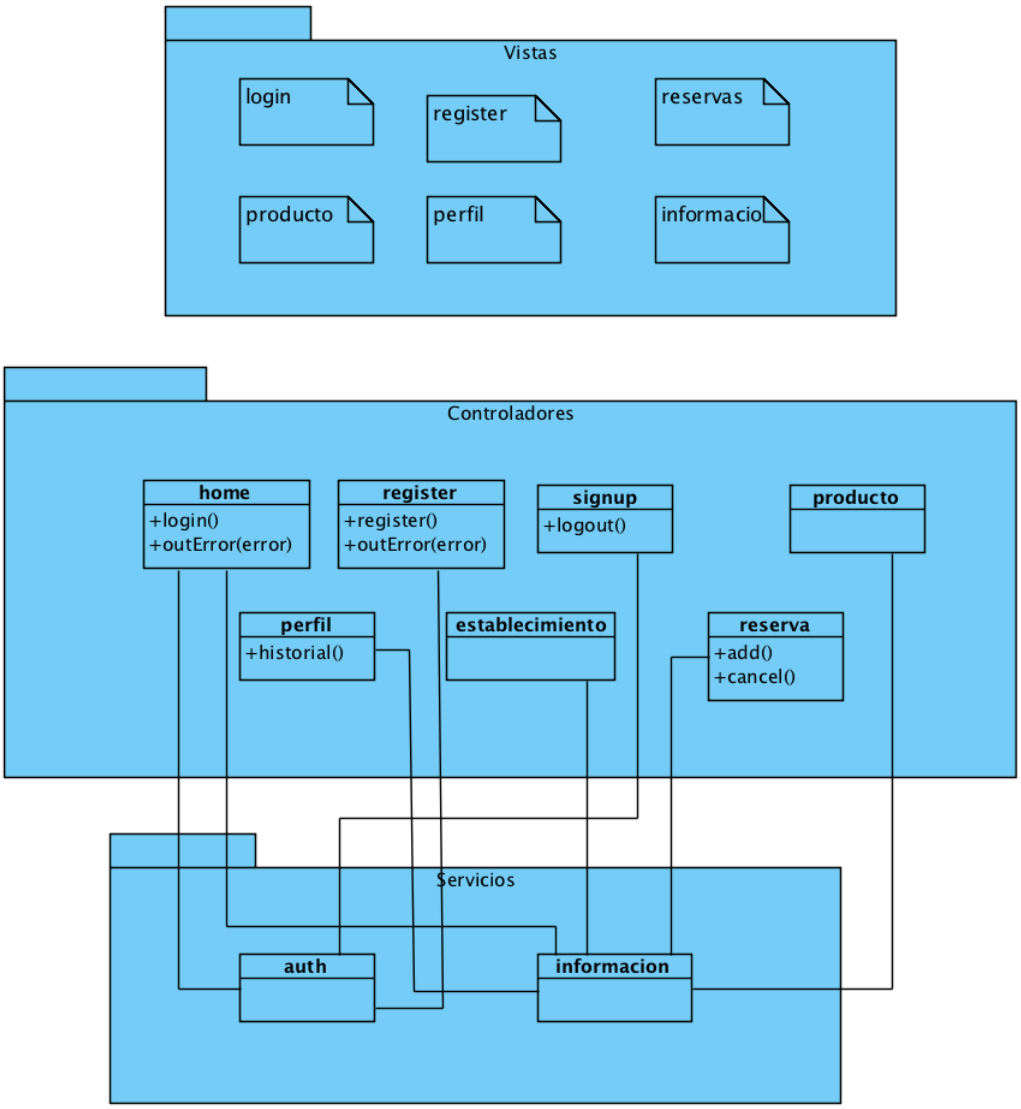

Ilustración 28: Diagrama de clases de la aplicación cliente

En el diagrama se han simplificado las conexiones de la vista por su visibilidad, se puede comprobar como los controladores de las vistas, captan el evento requerido y enlazan con los servicios para su obtención de datos.

Diagrama de secuencia

A continuación, se muestra los diagramas de secuencia más representativos del sistema, donde se puede ver la interacción de objetos a través del tiempo, el diagrama de secuencia*1* es un tipo de diagrama usado para modelar interacción entre objetos en un sistema según UML.

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. En nuestro caso, puesto que hacemos uso de una arquitectura MVC tanto en la aplicación cliente como en la parte del servidor, los patrones de interacción

suelen ser similares para las diferentes funcionalidades implementadas. Por tal motivo, vamos a mostrar los diagramas de secuencia que van a ser más representativos de la solución, puesto que el resto de funcionalidades tienen un comportamiento muy similar como se ha justificado. Como tenemos dos aplicaciones, mostraremos en ambos casos los diagramas de secuencia más interesantes de cada aplicación.

- **Aplicación Servidor**

  - Aceptar reserva:

    Los clientes realizan una petición de cita a una hora deseada, esta petición, queda almacenada asociada a la hora elegida.
    El administrador puede ver estas peticiones en el horario del establecimiento y aceptar la solicitud.

    El servicio envía los datos a la API, ésta llama al DAO de reservas, que comprueba la existencia de la solicitud y la acepta.
    Una vez realizada la aceptación de la solicitud, el administrador podrá ver el cambio de estado de la reserva, al igual que el cliente.

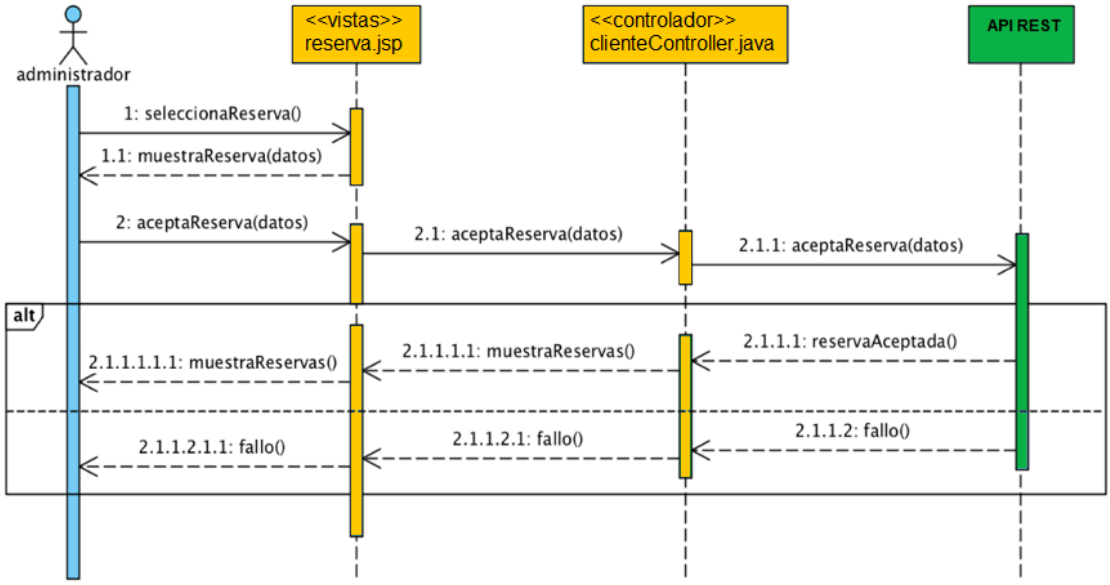

Ilustración 29: diagrama de secuencia aceptar Reserva

Por otra parte, en el servidor se reciben los datos de aceptación de reserva en el controlador de la API, después se utiliza el DAO de reservas para editar el estado de la solicitud de la reserva y se envía una respuesta al cliente según si hubo o no éxito.

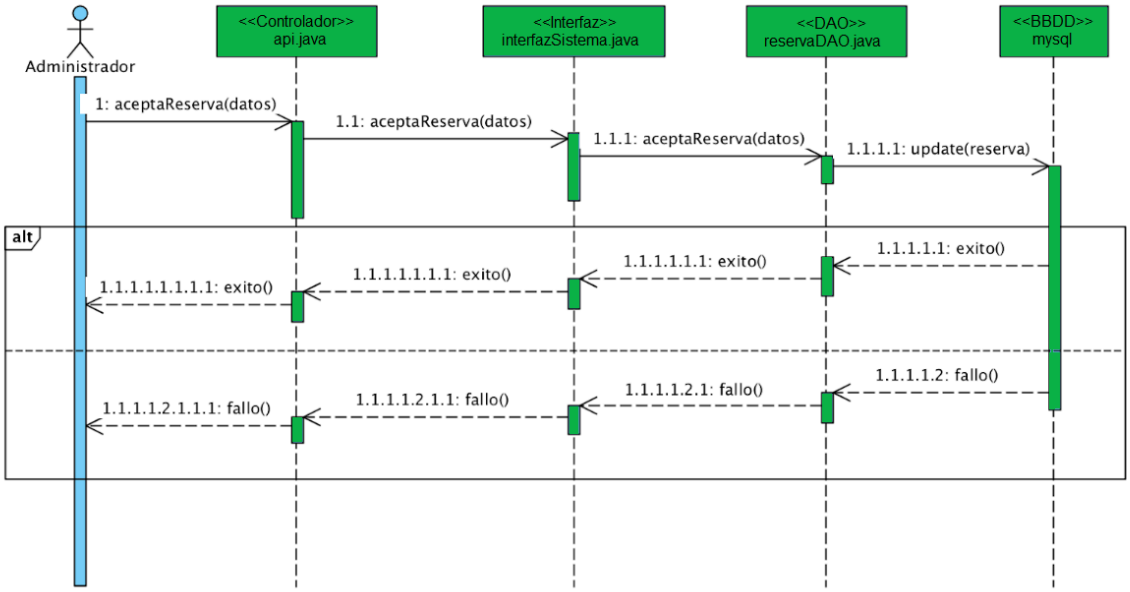

Ilustración 30: diagrama de secuencia aceptar Reserva servidor

- Crear productos:

  El administrador puede consultar los productos que van a ser vistos por la aplicación cliente. El administrador tiene acceso a un formulario donde podrá insertar un producto nuevo en el sistema.
  El servicio envía los datos a la API, ésta llama al DAO de productos, que inserta el nuevo producto.
  Una vez realizada la inserción, el administrador podrá ver el nuevo producto en la lista de productos del sistema.

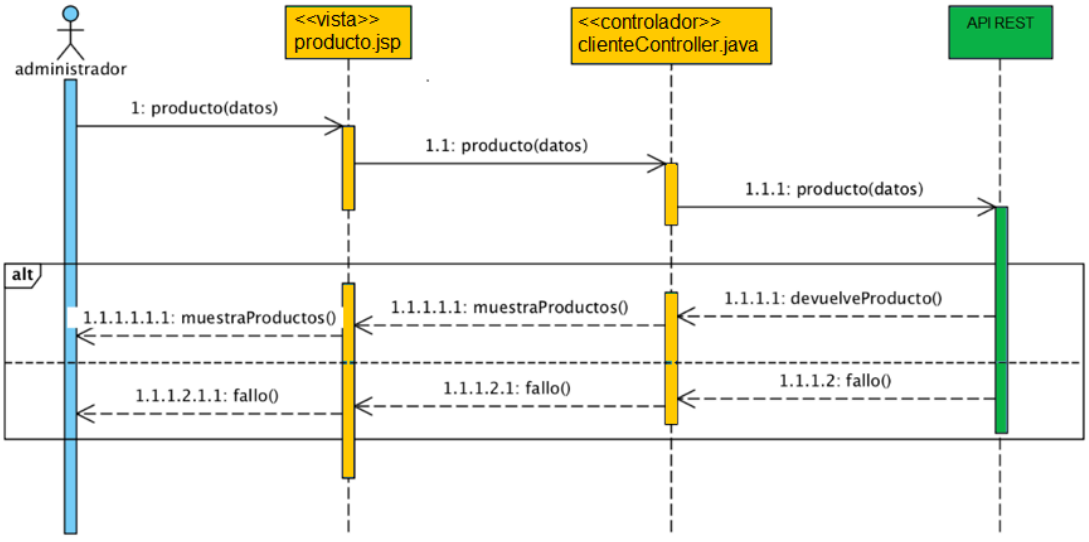

Ilustración 31: diagrama de secuencia crear Producto

En el servidor se reciben la petición de inserción de un nuevo producto en el controlador de la API, después se utiliza el DAO de productos para la inserción del producto y se envía una respuesta al cliente según si hubo o no éxito.

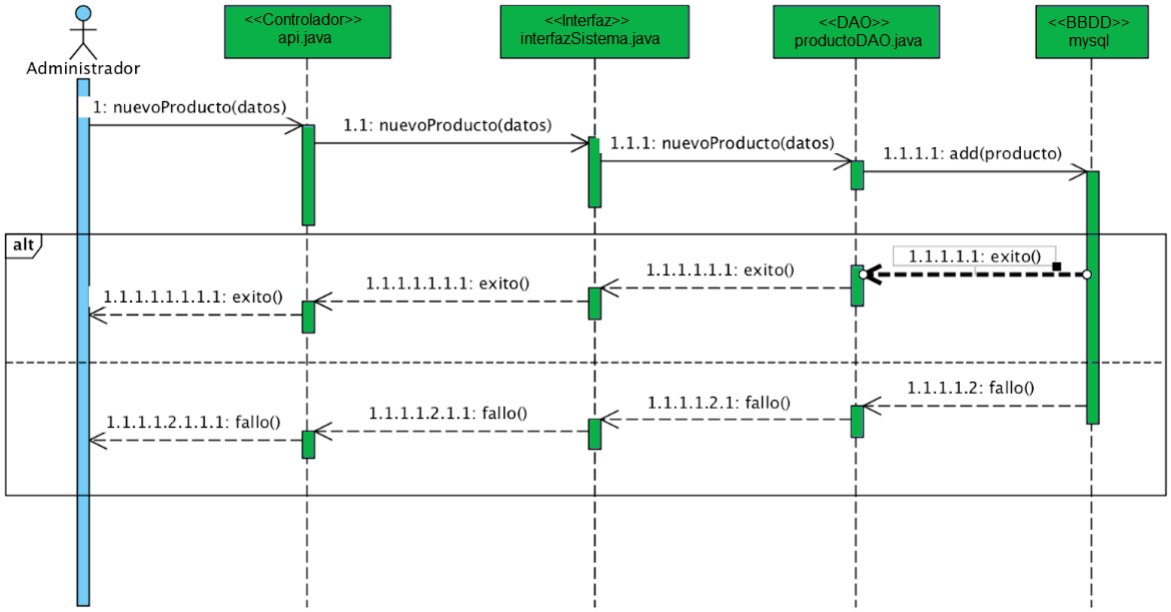

Ilustración 32: diagrama de secuencia crear Producto servidor

- Consulta de clientes:

  El administrador tiene acceso a los clientes que están registrados en el sistema, pudiendo consultar su perfil.
  El servicio envía el nombre del cliente a la API, ésta llama al DAO de clientes, que comprueba y obtiene los datos del cliente solicitado.
  Una vez realizada la consulta, el administrador podrá ver los datos del perfil del cliente solicitado.

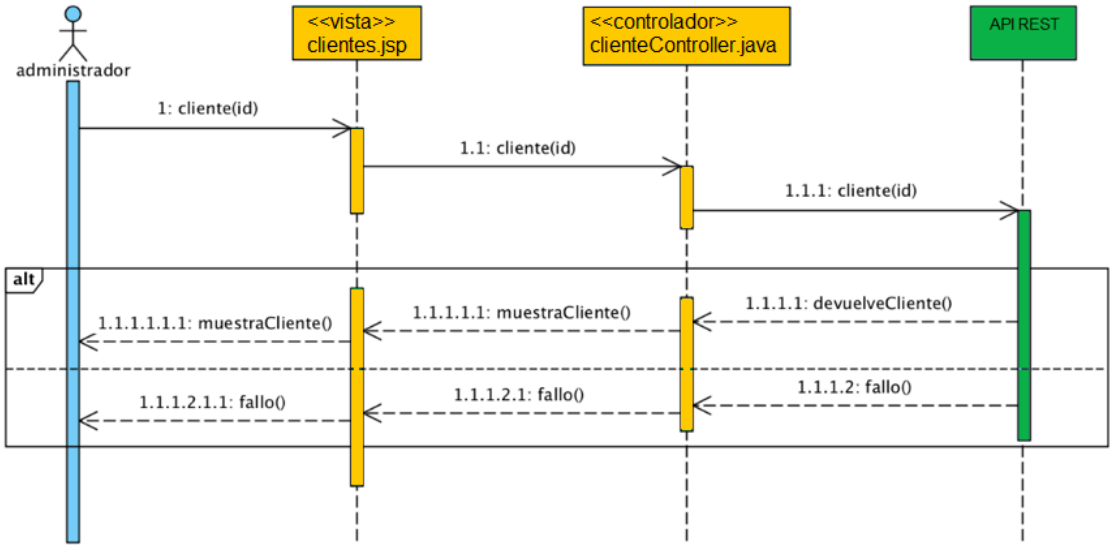

Ilustración 33: diagrama de secuencia consultar cliente

El servidor se recibe la petición de consulta de cliente en el controlador de la API, después se utiliza el DAO de clientes para la consulta del cliente y se envía el cliente al controlador para su visualización.

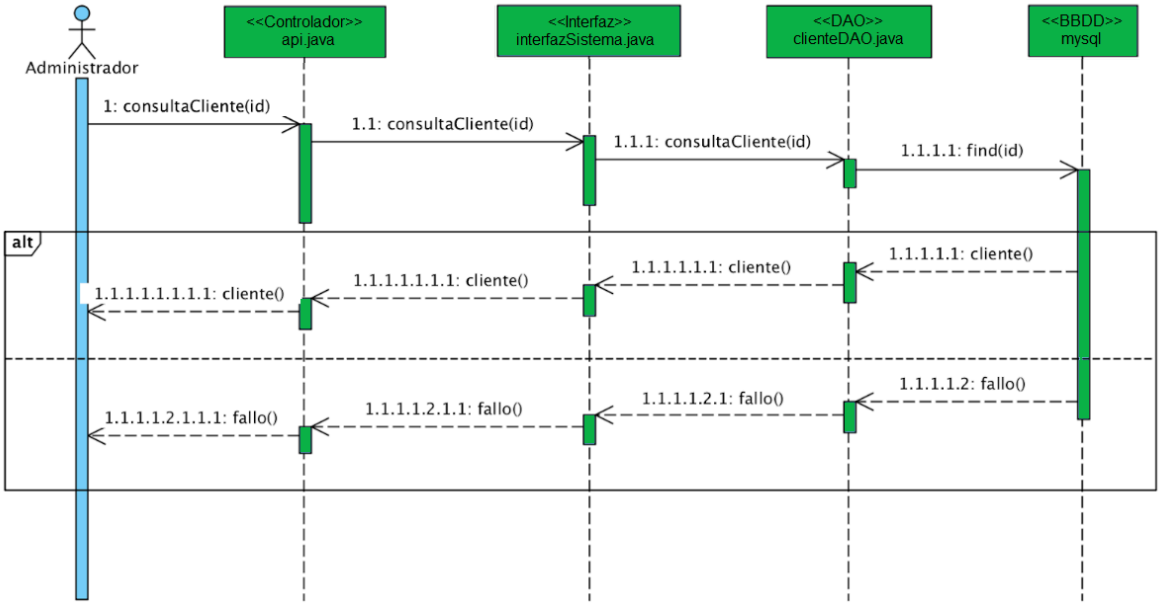

Ilustración 34: diagrama de secuencia consultar cliente servidor

- **Aplicación Cliente**

  - Registro

    Si el cliente no tiene registro en el sistema, puede acceder mediante el formulario que está habilitado para ello, el cliente introduce los datos

que llegan al controlador y este los envía al servicio llamado *auth.ts* que se comunica con la API.
Una vez realizado el proceso de registro, el servidor devuelve una respuesta de éxito o fracaso y se envía al cliente a la siguiente vista.

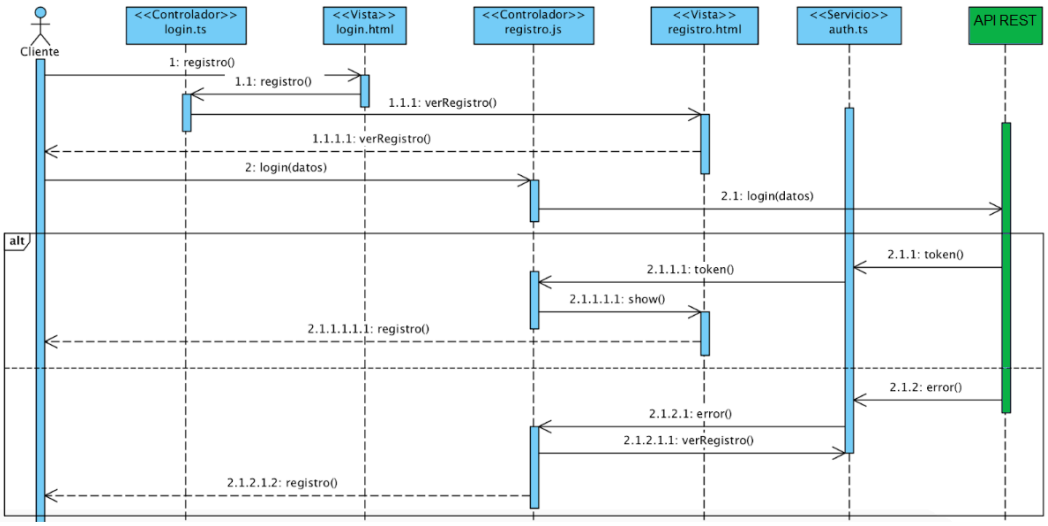

Ilustración 35: diagrama de secuencia registro cliente

El servidor se recibe la petición de registro de cliente en el controlador de la API, después se utiliza el DAO de clientes para la consulta del cliente y se envía como respuesta el *token* y si no se ha registrado con éxito un error.

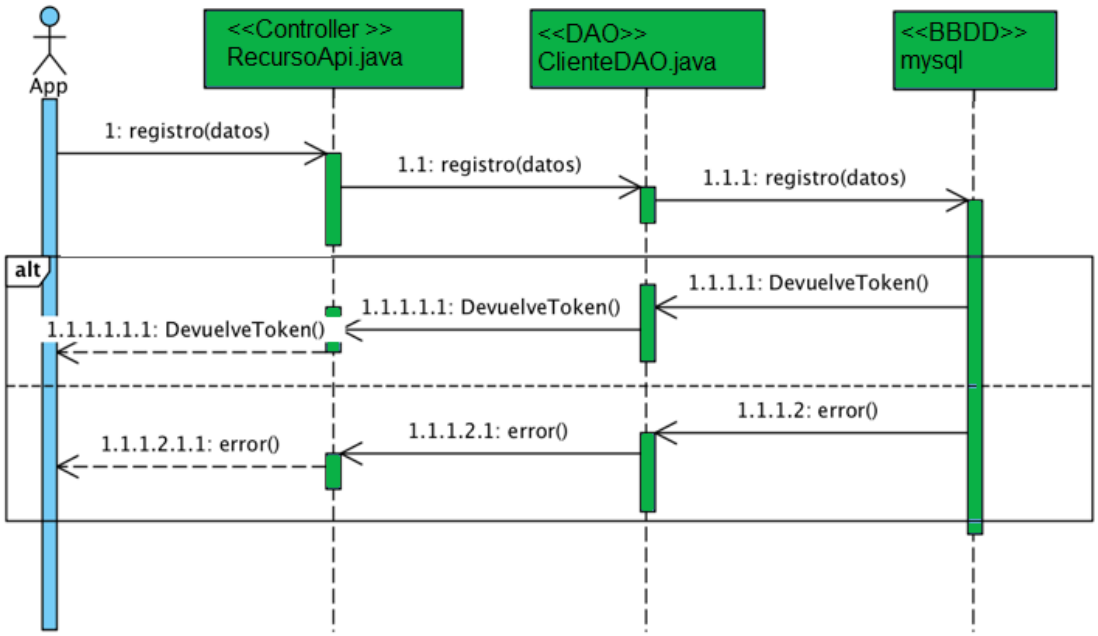

Ilustración 36: diagrama de secuencia registro cliente servidor

- Login

El cliente tiene acceso a la aplicación por medio de un formulario de logueo, este formulario envía los datos de entrada al controlador para posteriormente llegar al servicio que se encarga de la autenticación.
El servicio envía los datos del cliente a la API, ésta llama al DAO de clientes, que comprueba su existencia y obtiene un *token*[35] de entrada al sistema.

El token es una cadena de caracteres que tiene un significado coherente para el sistema, este token se basa en una serie de caracteres que son almacenados junto al cliente en base de datos, que el cliente envía al servidor, éste comprueba que se trata del mismo código almacenado en base de datos y verifica la entrada del cliente.

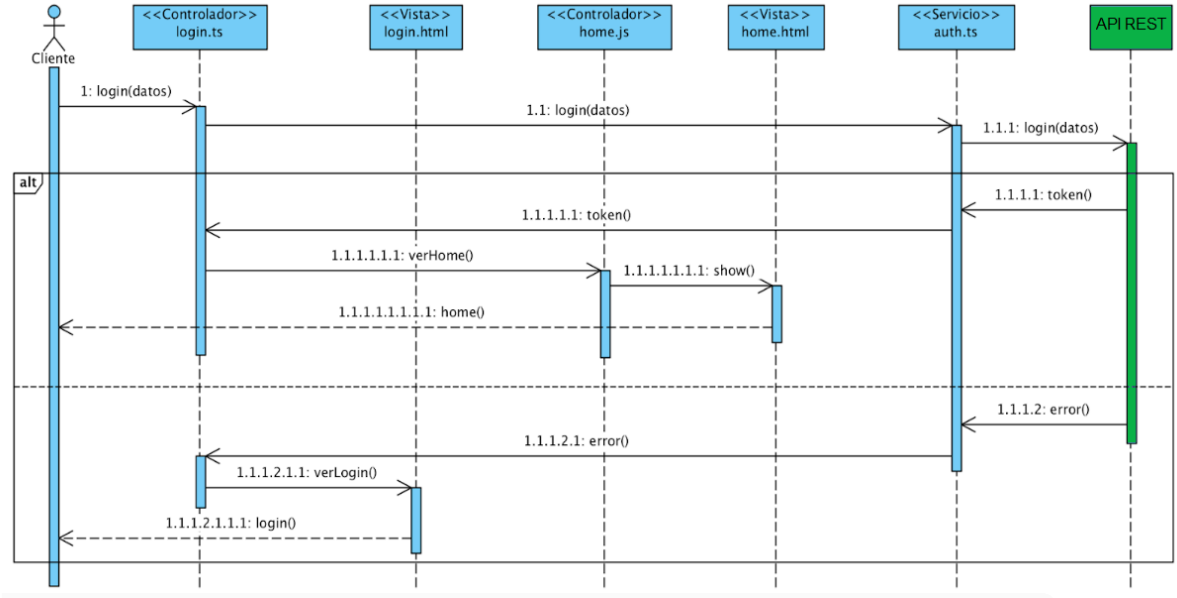

Ilustración 37: diagrama de secuencia login cliente

El servidor se recibe la petición de *login* de cliente en el controlador de la API, después se utiliza el DAO de clientes para la consulta del cliente y se envía como respuesta el *token* y si no se encuentra en el sistema se devuelve un error.

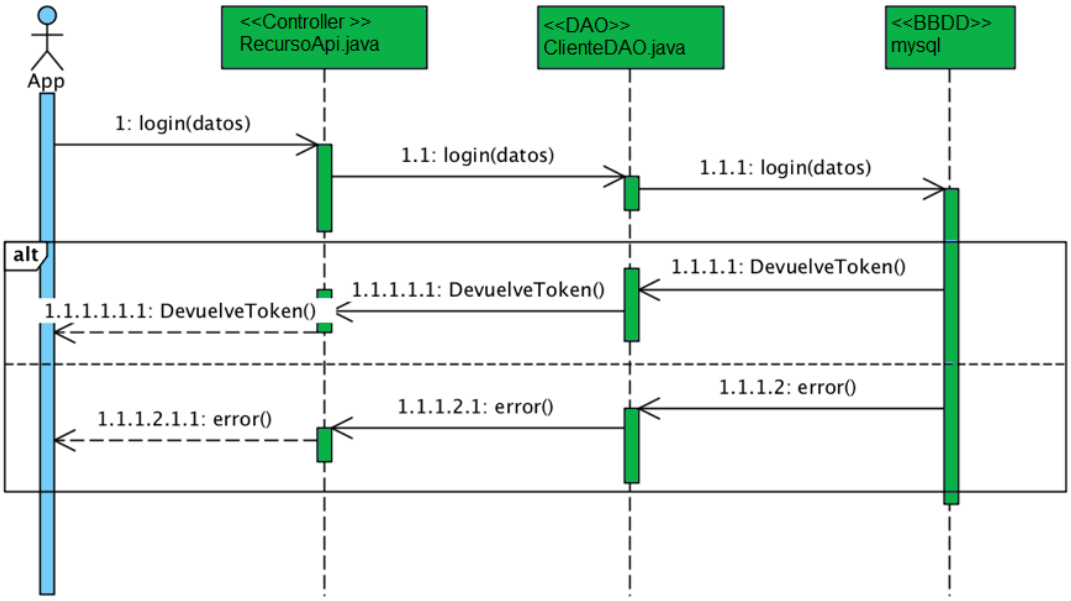

Ilustración 38: diagrama de secuencia login cliente servidor

- Reserva

  El cliente puede ver el horario de reservas del establecimiento, donde se podrá realizar una reserva a las horas que estén habilitadas para ello, el servicio envía la fecha y el nombre del cliente a la API, ésta llama al DAO de reservas, que comprueba al cliente y se procede a realizar la reserva asociada al cliente dado.
  Una vez realizada la reserva, el cliente podrá ver el estado de solicitud de reserva, que deberá ser contestado por el administrador del sistema.

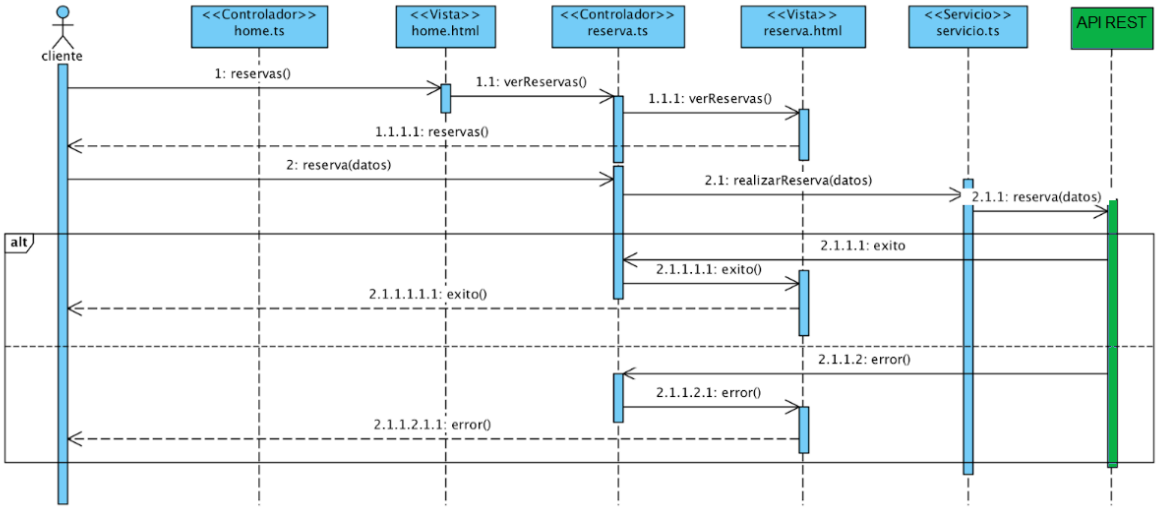

Ilustración 39: diagrama de secuencia reserva cliente

El servidor se recibe la petición de inserción de reserva en el controlador de la API, después se utiliza el DAO de reservas para la inserción de la reserva y se envía el cliente la reserva para la visualización del estado de la reserva.

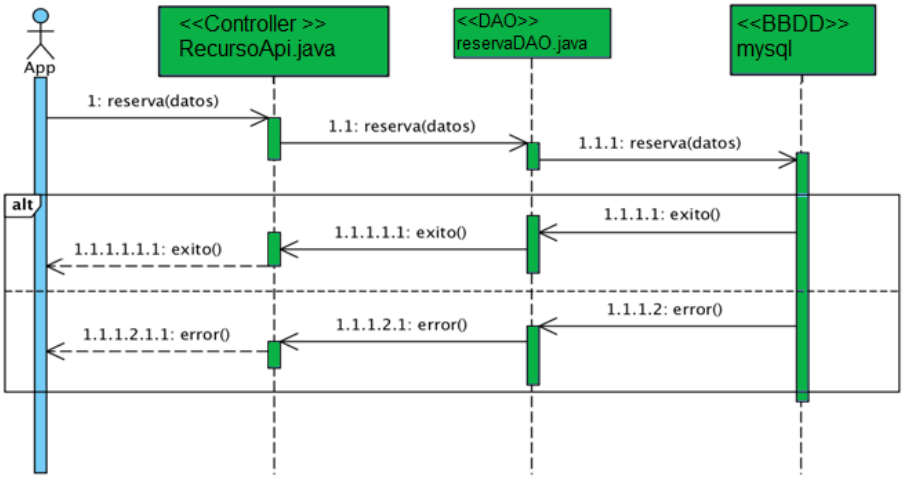

Ilustración 40: diagrama de secuencia reserva cliente servidor

Diseño de la interfaz

En este punto en primer lugar se estudiará el diagrama de estados de la aplicación cliente, mediante una representación gráfica en la que podremos ver las rutas o caminos que puede tomar un movimiento de información luego de ejecutarse cada proceso.

Una vez visto el diagrama de estados se mostrarán las interfaces de la aplicación servidor de reserva, producto y cliente, al igual que los bocetos de las interfaces de la aplicación del cliente.

Diagrama de estados de la aplicación cliente

El diagrama de estados muestra las transiciones que se desarrollan en el flujo de ejecución de la aplicación cliente.
Inicialmente, el cliente tendrá una ventana de *login* con la que puede acceder al sistema mediante un formulario de acceso, si el cliente no tiene acceso a la aplicación puede hacerlo mediante la vista de registro, en la que el usuario quedará registrado en el sistema y puede hacer *login* una vez completado este procedimiento.
Una vez completado el proceso de autenticación, se redirige al usuario a la vista de reservas, donde podrá ver el horario del establecimiento con las reservas que están disponibles y podrá solicitar la reserva de la hora deseada.
Además, el cliente podrá consultar en la vista de productos, los productos disponibles para comprar en el establecimiento, así como información del establecimiento y los datos almacenados de su perfil, cada uno en una vista diferente.

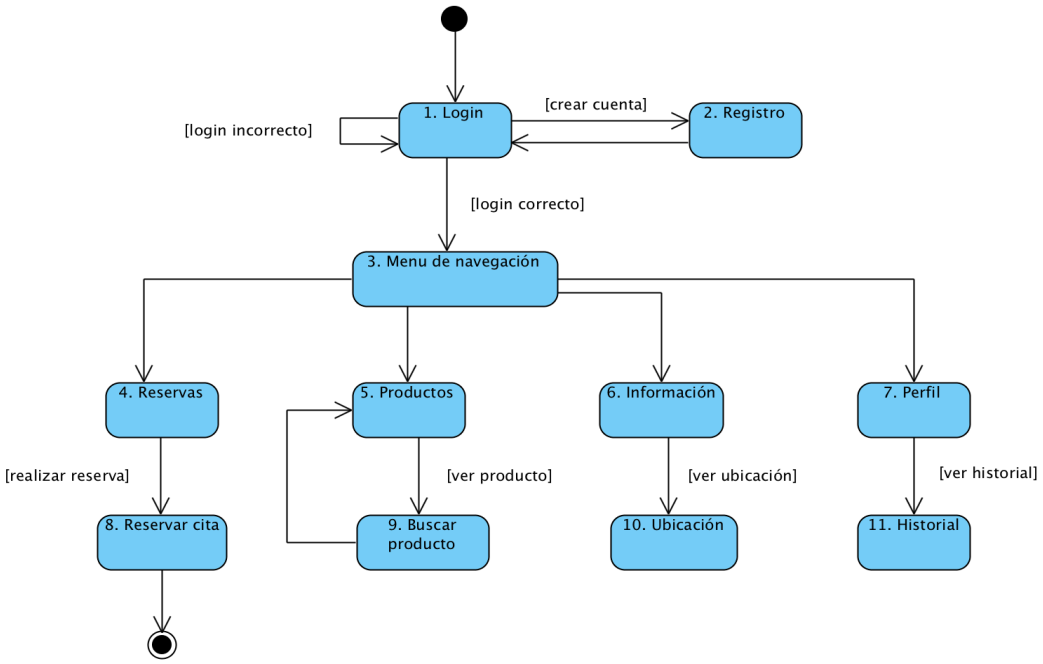

Ilustración 41: diagrama de estados de la aplicación cliente

Interfaces de la aplicación servidor

A continuación, se muestran los bocetos de varias de las interfaces que están a disposición del administrador, estas vistas han sido diseñadas mediante la aplicación quickMockup[18]:

- reservas: en este primer boceto podremos ver el horario de reservas del establecimiento, estarán organizados en una tabla donde de cada

[18] https://jdittrich.github.io/quickMockup/

día partirán las horas de trabajo, donde cada una es una reserva que tiene diferentes estados, cada estado estará identificado por un color diferente: verde disponible, naranja solicitada, rojo reservada. Al pulsar sobre una reserva, depende del estado en el que se encuentre se mostrará un cuadro de opciones que podemos aplicar sobre la reserva pulsada.

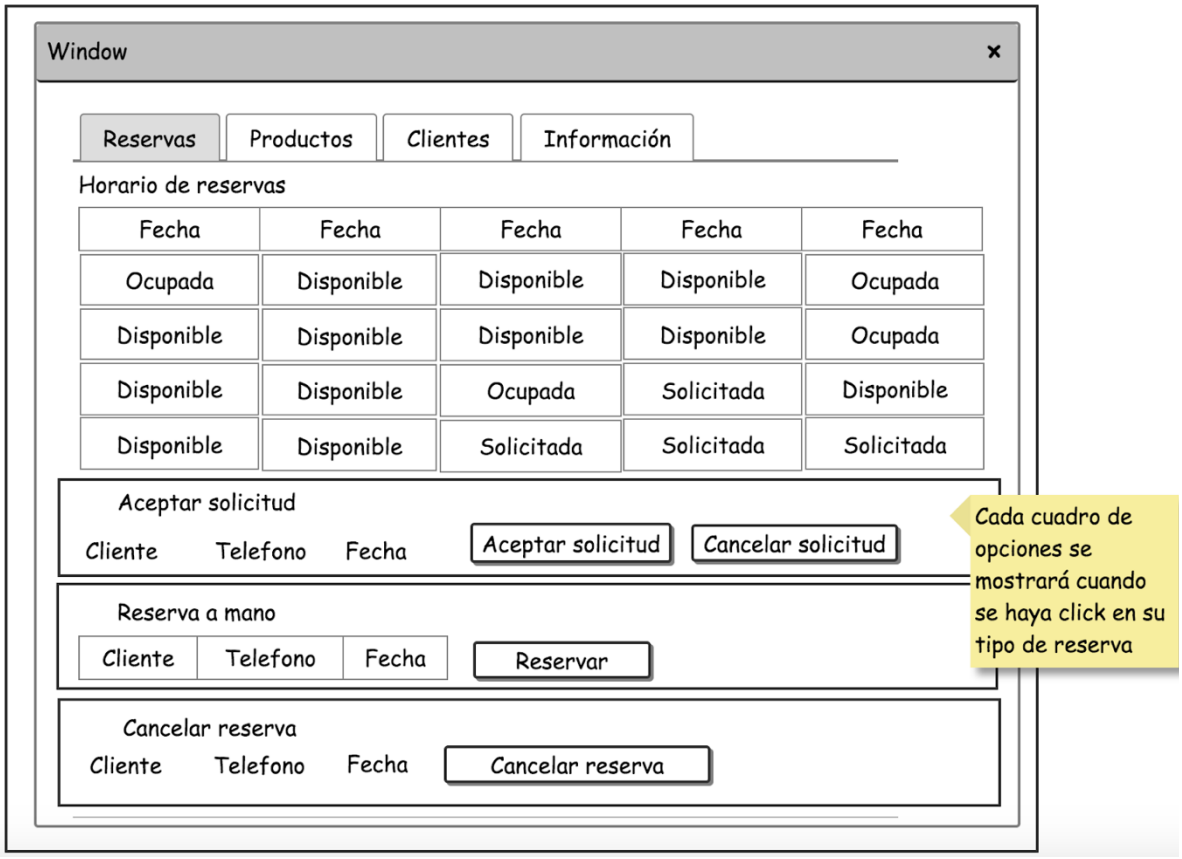

Ilustración 42: interfaz de reservas

- productos: en esta vista se podrán ver los productos almacenados en el sistema, cada producto podrá ser editado por medio de un formulario que será accesible al pinchar sobre un producto y un formulario de inserción para nuevos productos.

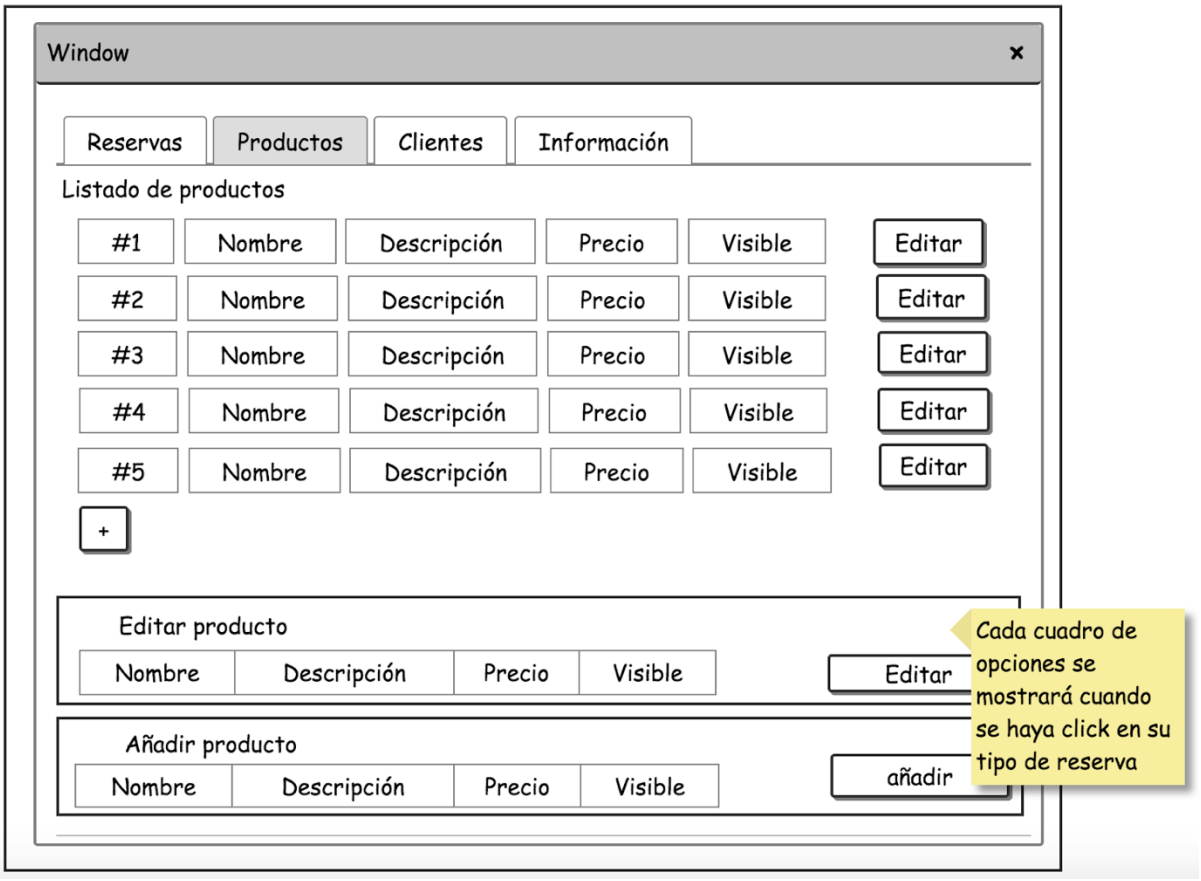

Ilustración 43: interfaz de productos

- clientes: en la vista de clientes se podrá hacer una búsqueda del cliente por medio de nick y nombre de usuario, una vez consultado el cliente se podrá ver el histórico de reservas del cliente, al igual que un formulario donde podremos habilitar/deshabilitar la entrada del cliente en la aplicación.

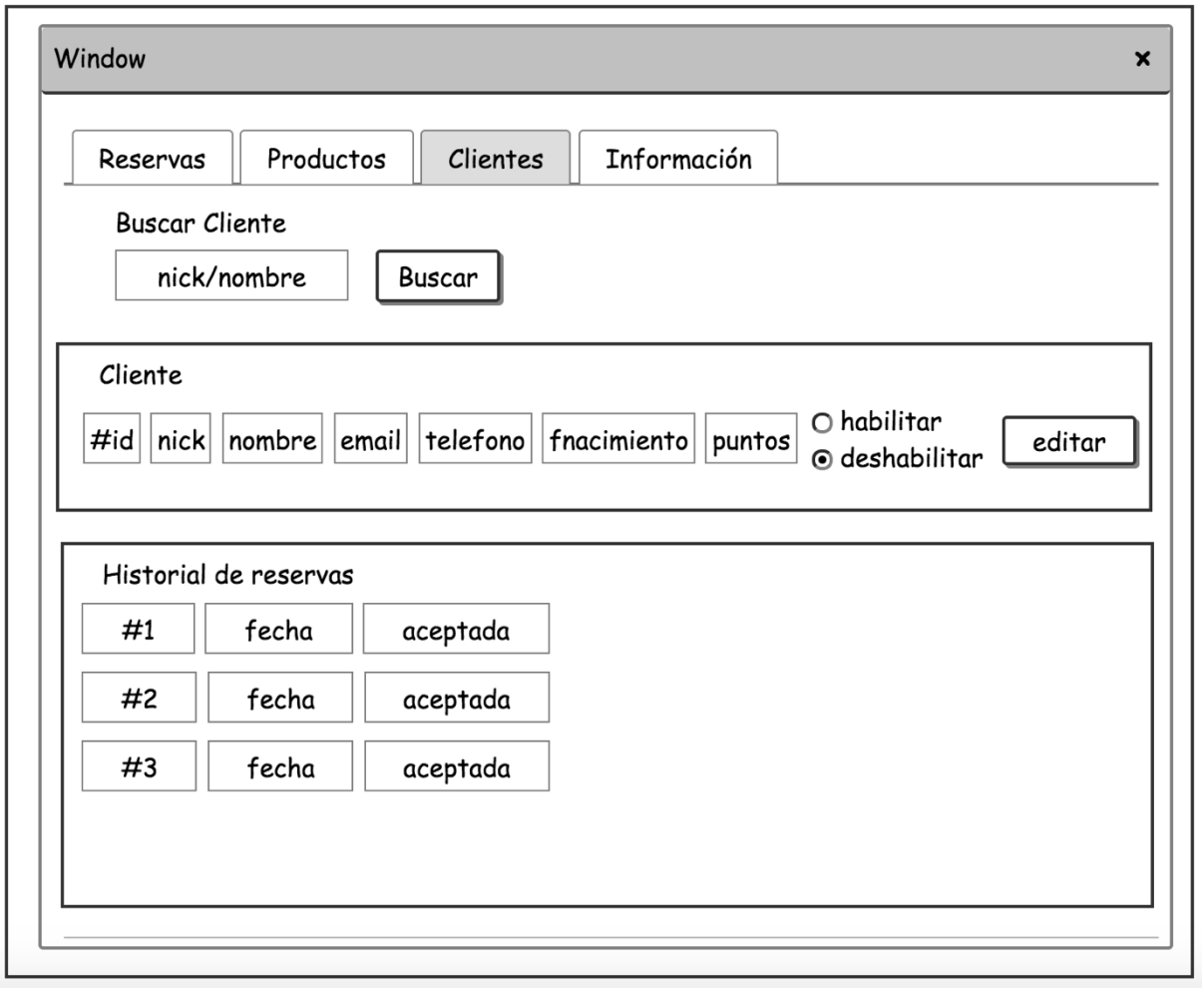

Ilustración 44: interfaz de clientes

Interfaces de la aplicación cliente

En este apartado se muestran los bocetos de varias de las interfaces que están a disposición del cliente, estas vistas han sido diseñadas mediante la aplicación que hemos mencionado anteriormente.

- identificación: A continuación, se mostrarán las interfaces de acceso al sistema que están disponibles en la aplicación del cliente. En primer lugar, se encuentra el *login,* donde se introduce el nick en el sistema y contraseña para acceder en caso de estar ya registrado, por otra parte, si no está registrado, tiene un formulario de registro al sistema, donde una vez rellenados todos los campos y validados podrá tener acceso al sistema.

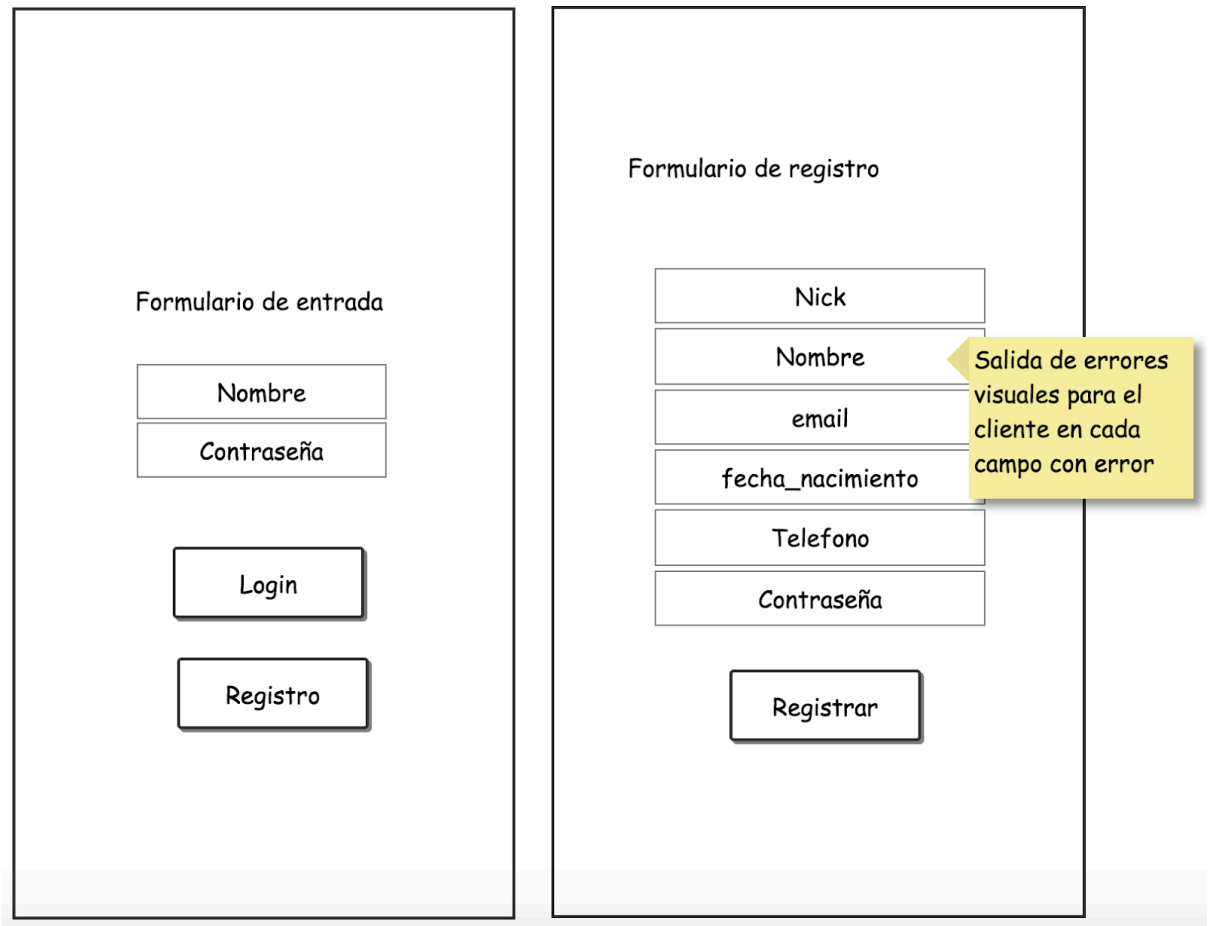

Ilustración 45: interfaz de acceso al sistema

- reserva: En esta vista se puede ver cómo se realiza una reserva en la aplicación del cliente, para ello se dispone del horario de reservas del establecimiento, donde se visualizarán de un color distinto las reservas según el estado en el que se encuentren: verde disponible, naranja solicitada, rojo reservada.

Ilustración 46: interfaz de reservas de la aplicación cliente

- perfil: En esta vista el cliente podrá ver una lista con las reservas realizadas en el establecimiento, además de un formulario donde el cliente puede consultar sus datos almacenados en el sistema y editarlos.

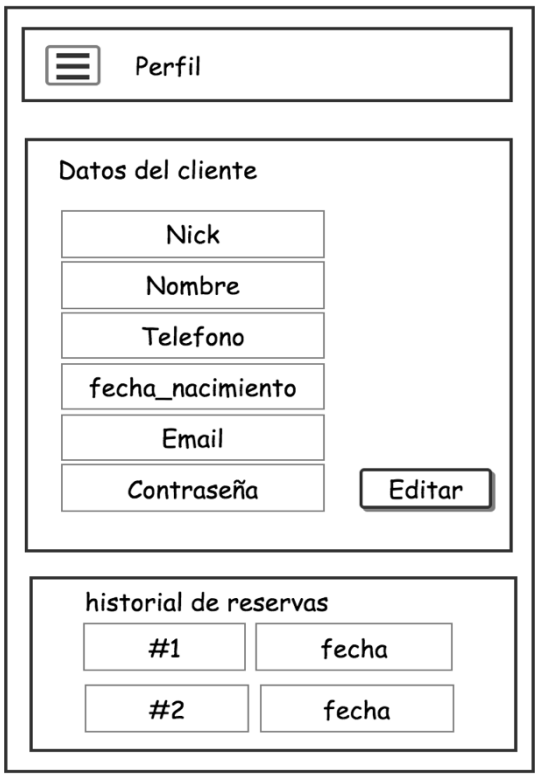

Ilustración 47: interfaz de perfil del cliente

4. Implementación

A continuación, se detalla la arquitectura de la aplicación y los componentes más relevantes que se han usado para el desarrollo de la solución, para ello mostraremos fragmentos de código de la aplicación.

Arquitectura de la aplicación

Para la implementación de la solución se va a hacer uso de la arquitectura cliente-servidor, el cliente realizará una serie de peticiones que serán respondidas por el servidor.
La arquitectura cliente-servidor es una red de comunicaciones en la cual los clientes están conectados a un servidor, en el que se centralizan los diversos recursos y aplicaciones que se ponen a disposición de los clientes cada vez que estos son solicitados. Esto significa que todas las gestiones que se realizan se concentran en el servidor, de manera que en él se disponen los requerimientos provenientes de los clientes que tienen prioridad, los archivos que son de uso público y los que son de uso restringido[36].

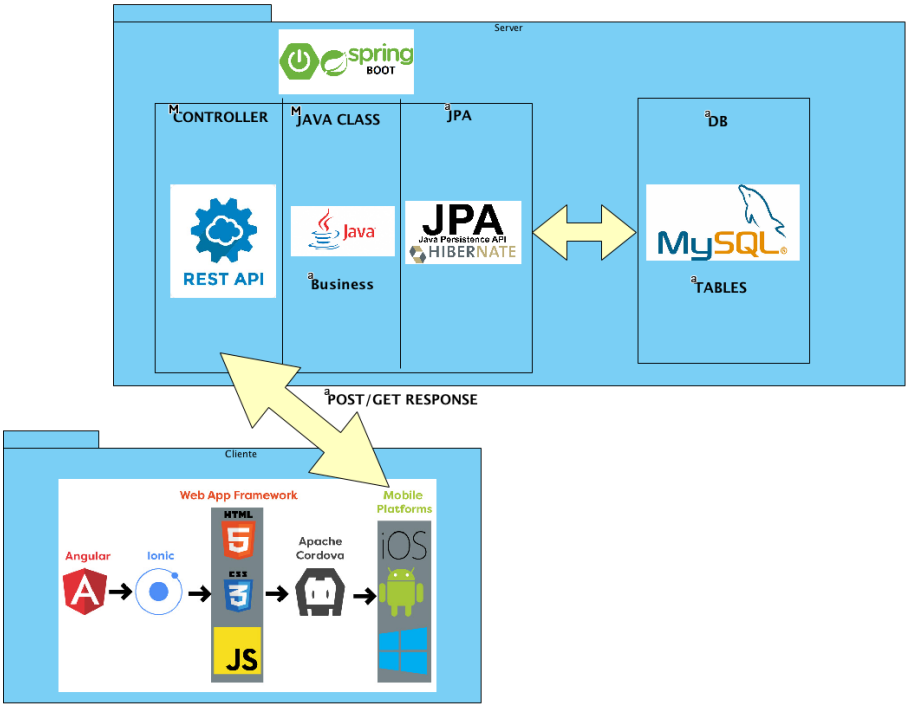

Ilustración 48: arquitectura cliente-servidor

La comunicación entre el cliente y el servidor se realiza por medio de peticiones, estas peticiones se atienden en el servidor por medio de una API.

Una API REST[37] es una interfaz de comunicación entre sistemas que utilicen el protocolo HTTP[19], para ser una API REST se necesitan peticiones GET/POST/PUT/DELETE.
Una vez se ha realizado la comunicación cliente-servidor, para obtener datos o realizar operaciones sobre los datos, en cualquier formato XML[20] que es un lenguaje de marcas utilizado para almacenar datos en forma legible, JSON[21], sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes.
En la siguiente imagen se muestra el esquema de un servicio de API REST.

Ilustración 49: servicio API REST Fuente: https://ingcarlosperez.co/blog/tecnologias-web/buenas-practicas-en-el-diseno-de-peticiones-api-rest.html

En nuestro caso, el cliente hará uso del framework ionic y para el servidor Spring. El cliente Ionic se comunica con la API que proporciona el servidor por medio de peticiones GET y POST. Estas peticiones son recogidas por el servidor que se conecta a la base de datos por medio de JPA y se almacenan los datos en unas clases JAVA que forman parte del modelo de negocio del proyecto.

Detalles de la implementación

En este punto analizaremos los detalles de cada una de las aplicaciones que hemos desarrollado. A continuación, se describe la aplicación servidor, en ella detallamos como implementamos el servicio de persistencia, como también las llamadas a los servicios disponibles en la API. Para la aplicación del cliente podemos ver la estructura utilizada y la implementación de las vistas.

Para la realización de la aplicación del administrador se ha utilizado NetBeans 8.0.2 y Postman 5.5.2. Con NetBeans se han creado dos proyectos Java Spring MVC donde uno es el servidor que contiene los JPA y la API del proyecto y el otro proyecto contiene las vistas para la administración del establecimiento por parte del administrador.

[19] https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto
[20] https://es.wikipedia.org/wiki/Extensible_Markup_Language
[21] https://www.json.org/

- **Aplicación servidor**

Una vez creada la aplicación incluiremos los paquetes que nos van a hacer falta para la implementación del servidor. Lo primero que vamos a implementar van a ser las clases que contendrán los JPA que forman parte del modelo de dominio de la solución.
A continuación, se muestra un ejemplo de implementación JPA en la clase Reserva.Java

```
@Entity
@Table(name="Reserva")
@XmlRootElement(name="reserva")
public class Reserva implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    private int id;
    private boolean estado;
    private boolean aceptada;
    private String fecha;
    @OneToOne
    private Cliente cliente;
    @ManyToOne
    @JoinColumn(name="codEstablecimiento")
    private Establecimiento establecimiento;
```

Ilustración 50: implementación JPA en Reserva.Java

Mediante la etiqueta @Entity se indica que esta será una clase que forme parte de la persistencia en JPA. En ellos se almacenarán los datos del sistema.
@Table nos sirve para indicar el nombre de la clase que tendrá en la persistencia y @XMLRootElement el nombre de tag que queramos que salga en el fichero XML, nuestro bean java necesita llevar unas anotaciones especiales que ayuden a convertir el Java bean[38] en XML.
Para identificar los objetos que Spring va a utilizar, se definen mediante Javabean, que una clase de Spring que se encargará de realizar la instanciación de los objetos cuando sea necesario y de establecer la relaciones entre ellos mediante inyección de dependencias, estos se definen en el archivo ApplicationContext.xml del proyecto.

```
<bean id="InterfazSistema" class="servidor.InterfazSistemaImpl">
    <property name="clienteDAO" ref="clienteDao" />
    <property name="reservaDAO" ref="reservaDao" />
    <property name="productoDAO" ref="productoDao" />
    <property name="establecimientoDAO" ref="establecimientoDao" />
</bean>

<bean id="clienteDao" class="dao.ClienteDAO" />
<bean id="reservaDao" class="dao.ReservaDAO" />
<bean id="productoDao" class="dao.ProductoDAO" />
<bean id="establecimientoDao" class="dao.EstablecimientoDAO" />

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="org.apache.derby.jdbc.ClientDriver" />
    <property name="url" value="jdbc:derby://localhost:1527/BBDDpeluqueria" />
    <property name="username" value="root" />
    <property name="password" value="root" />
</bean>
```

Ilustración 51: implementación de ApplicationContext.xml

Se define en un bean el nombre de la base de datos al igual que el valor de acceso.
En este fichero se definen también los DAOs que son los componentes que proporcionan comunicación entre la aplicación y el almacenamiento de datos.
Por lo que cuando la capa de lógica de negocio necesite interactuar con la base de datos, va a hacerlo a través del servicio que le ofrece los DAOs.
Las ventajas de utilizar Spring Framework para crear sus ORM DAO incluyen[39]:

- *Pruebas más fáciles.* El enfoque de IoC de Spring facilita el intercambio de implementaciones y ubicaciones de configuración de Hibernate,JDBC, gestores de transacciones e implementaciones de objetos mapeados. Esto a su vez hace que sea mucho más fácil probar cada parte del código relacionado con la persistencia de forma aislada.
- *Excepciones comunes de acceso a datos.* Spring puede ajustar excepciones de su herramienta ORM, convirtiéndolas de excepciones creadas por nosotros en tiempo de ejecución DataAccessException común. Esta característica le permite manejar la mayoría de las excepciones de persistencia.
- *Gestión general de recursos.* Esto hace que estos valores sean fáciles de administrar y cambiar. Spring ofrece un manejo eficiente, fácil y seguro de los recursos de persistencia.
- *Gestión integrada de transacciones.* Se puede generar un interceptor de método de declaración orientada a aspectos (AOP), ya sea a través de la etiqueta @Transactional o configurando explícitamente el aviso de AOP de transacción en un archivo de configuración XML.

En la siguiente imagen se muestra la implementación de la clase ProductoDAO.Java

```java
@Repository
@Transactional(propagation=Propagation.SUPPORTS, readOnly=true)
public class ProductoDAO {

    @PersistenceContext
    private EntityManager em;

    public ProductoDAO(){}

    @Transactional(propagation = Propagation.REQUIRED, readOnly = false, rollbackFor = ExcepcionProductoNoValido.cla
    public void add(Producto p){
        try{
            //Introducimos el usuario en la base de datos
            em.persist(p);
            em.flush();
        }catch(Exception e){
            throw new ExcepcionProductoNoValido();
        }
    }
```

Ilustración 52: implementación ProductoDAO.Java

Todo esto se envuelve en el bean del servidor, en este bean se encuentra la interfaz del sistema donde se definen todos los métodos que van a estar disponibles para llamarse desde la API.

```java
public interface InterfazSistema{

    public Cliente comprobarLogeo(String pdto, String token) throws ExcepcionClienteNoValido;
    public Reserva buscaReservaFecha(String nombre) throws ExcepcionReservaNoValida;
    public List<Reserva> buscaReservaList(String fecha1,String fecha2) throws ExcepcionReservaN
    public Cliente altaUser(ClienteDTO user) throws ExcepcionClienteNoValido;
    public Cliente login(String id, String pass) throws ExcepcionClienteNoValido;
    public Reserva creaReserva(String id, String fecha, boolean estado, boolean aceptada) throw
    public Reserva cancelaReserva(String id, String fecha) throws ExcepcionReservaNoValida;
    public Reserva aceptaReserva(String id, String fecha) throws ExcepcionReservaNoValida;
    public Reserva MuestraReserva(int id) throws ExcepcionReservaNoValida;
    public Cliente buscaPersonaPorNick(String nick) throws ExcepcionClienteNoValido;
    public Cliente buscaPersonaPorTelefono(int numero) throws ExcepcionClienteNoValido;
    public List<Reserva> mostrarReservas() throws ExcepcionReservaNoValida;
    public boolean comprobarNick(String nick) throws ExcepcionClienteNoValido;
    public List<Reserva> mostrarReservaCliente(String id) throws ExcepcionReservaNoValida;
```

Ilustración 53: implementación interfazSistema.Java

Estos métodos son llamados por los servicios que ofrece la API, en la siguiente imagen se muestra una llamada GET a la llamada al servicio para la obtención del historial del cliente.

```java
/**
 * @brief historial de reservas de un cliente dado por id
 * @param id id del cliente
 * @return lista con las reservas del cliente
 * @throws Excepciones.ExcepcionClienteNoValido
 */
@RequestMapping(value = "administrador/reservas/cliente", method = RequestMethod.GET)
public List<ReservaDTO> historialReservasCliente(@RequestParam("idusu") String id) throws ExcepcionClienteNoValic
    if (interfazSistema.comprobarNick(id)) {
        List<ReservaDTO> listReserDto = new ArrayList<>();
        for (Reserva r : interfazSistema.mostrarReservaCliente(id)) {
            listReserDto.add(new ReservaDTO(r));
        }
        return listReserDto;
    }else{
        throw new ExcepcionClienteNoValido();
    }
}
```

Ilustración 54: implementación controlador.Java

A continuación, se observa la información relevante de los end-points[40] REST de la aplicación:

Tipo	Detalle	url	RequestBody	Return	Error
GET	acceso al sistema	/administrador/login	Cliente	Cliente	null
GET	Acceso al sistema	/cliente/login	Ciente	Cliente	null

POST	nuevo cliente	/cliente/nuevo	Cliente	Cliente	null
POST	crear reserva	/administrador/reserva/nueva	Reserva	200	null
DELETE	cancelar reserva	/administrador/reserva/cancela	Reserva	200	404
PUT	aceptar reserva	/administrador/reserva/cliente/acepta	Reserva	200	404
POST	Crear reserva	/cliente/reserva/nueva	Reserva	200	404
DELETE	Cancelar reserva	/cliente/reserva/cancela	Reserva	200	404
GET	buscar cliente	/administrador/cliente	id	Cliente	null
GET	buscar producto	/administrador/producto	id	Producto	null
GET	historial de reservas del cliente	/administrador/reservas/cliente	id	List<Reservas>	null
GET	listado de productos	/administrador/productos	null	List<Productos>	null
GET	Listado de productos	/cliente/productos	Null	List<Productos>	null
GET	buscar reserva	administrador/reserva/{id}	id	Reserva	null

GET	listado de reservas	/administrador/reservas	null	List<Reservas>	null
GET	Listado de reservas	/cliente/reservas	Null	List<Reservas>	null
GET	información del local	/administrador/informacion	null	Establecimiento	null
POST	crear producto	/administrador/producto/nuevo	Producto	200	404
PUT	editar producto	/administrador/producto/edita	Producto	200	404
POST	editar cliente	/cliente/edita	Cliente	200	404

Tabla 24: tabla end-points

Para terminar, se ha incluido el paquete de Spring Security, ya que es un framework que permitirá gestionar todo lo relativo a la seguridad de nuestra aplicación web, desde el protocolo de seguridad, hasta los roles que necesitan los usuarios para acceder a los diferentes recursos de la aplicación. En nuestro caso vamos a gestionar las url de la api, que son utilizadas por el administrador, restringiendo el acceso solo a usuarios con autorización.

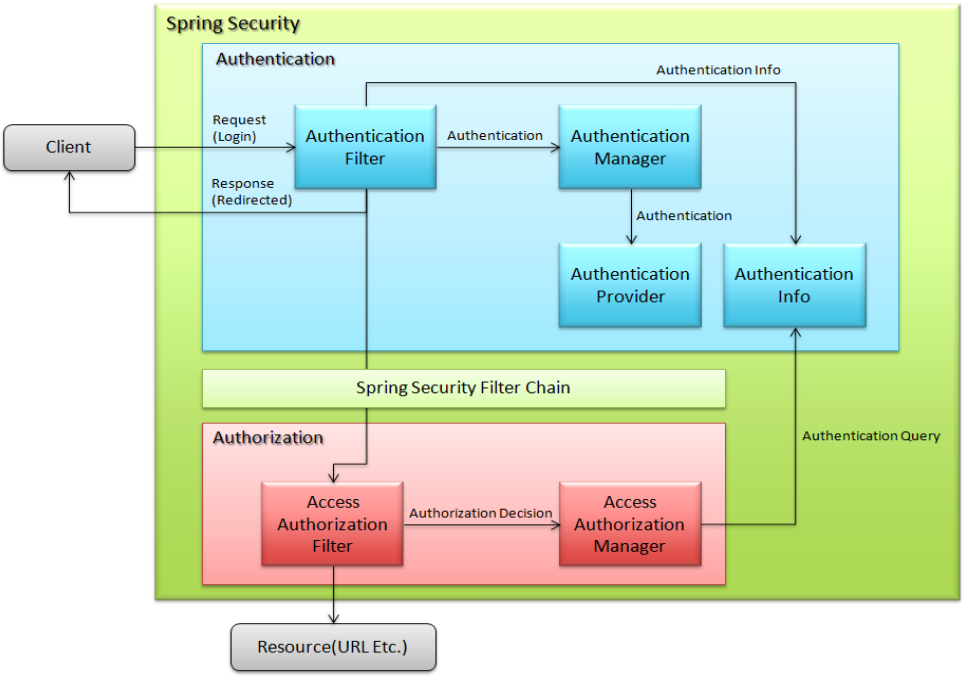

Ilustración 55: diagrama spring security

Para hacer uso de Spring Securiry, es necesario incluir en el fichero *web.xml* las urls a proteger.

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>
        org.springframework.web.filter.DelegatingFilterProxy
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/administrador/*</url-pattern>
</filter-mapping>
```

Ilustración 56: Spring securiry en fichero web.xml

Por otra parte, en el fichero applicationContext.xml se definen los usuarios que tienen autorización de entrada mediante su rol establecido.

```
<security:authentication-manager>
    <security:authentication-provider user-service-ref="proveedorUsuarios"/>
</security:authentication-manager>

<security:user-service id="proveedorUsuarios">
    <security:user name="admin" password="admin" authorities="ROLE_ADMIN"/>
</security:user-service>
```

Ilustración 57: Acceso de usuarios en el applicacionContext.xml

- **Aplicación administrador**

La aplicación del administrador se ha desarrollado en Spring MVC como hemos visto anteriormente, en esta sección vamos a profundizar más en la implementación de la aplicación.

Lo primero que tenemos que hacer es indicar en el archivo *dispatchet-servlet.xml* el componente donde se sitúa los componentes de las vistas, ya que en nuestro caso hemos utilizado el framework boostrap[22] para el diseño y la base del proyecto, formado por el controlador, modelo y DTOs del proyecto.

```
<context:component-scan base-package="cliente" />

<mvc:resources mapping="/img/**" location="/img/" />

<mvc:resources mapping="/css/**" location="/css/" />
```

Ilustración 58: archivo dispatcher-servlet.xml

[22] https://getbootstrap.com/

Bootstrap es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de Javascript adicionales con las que podremos realizar el diseño de las vistas mediante la inclusión de la clase de estilos deseada en las etiquetas que componen las vistas.

En estas vistas añadimos los enlaces que nos llevarán a los servicios implementados en el controlador. Éste formulario, contiene en su '*action*' la llamada implementada en el controlador, la llamada se lanzará cuando el administrador haga '*click*' en el botón de crear la reserva una vez rellenado por completo el formulario.

A continuación, se puede ver el ejemplo de un servicio implementado en la vista de reservas del administrador, en este caso, se trata del formulario para crear las reservas.

```
<div class="container">
    <div class='col-md-9 table-responsive'>
        <form class="table table-striped" id="formCrear" method="POST" action="creaReserva">
            <h4>Reserva a mano</h4>
            <ul class="ul-primero">
                <li class="li-primero">
                    <input class="form-control" id="nombre" name="nombre" placeholder="nombre"  type="text">
                </li>
                <li class="li-primero">
                    <input class="form-control" id="telefono" name="telefono" placeholder="telefono"  type="number">
                </li>
                <li class="li-primero">
                    <input class="form-control" id="fecha" name="fecha" placeholder="fecha de la reserva"  type="text"
                </li>
                <li class="li-primero">
                    <input class="btn btn-success" name="submit" id="submit" tabindex="-" value="crear reserva" type="
                </li>
            </ul>
            <hr>
        </form>
    </div>
</div>
```

Ilustración 59: formulario de creación de reservas

En el controlador, se captura esta llamada y se recogen los datos del formulario, por medio

```
/*
* Métodos de administrador
*/
@RequestMapping(value = "/creaReserva", method = RequestMethod.POST)
public String creaReserva(HttpServletRequest request, ModelMap model) throws ParseException {
    //obtener datos del formulario
    String nombre = request.getParameter("nombre");
    String telefono = request.getParameter("telefono");
    String fecha = request.getParameter("fecha");
    //realizar llamada al servicio del servidor
    restTemplate.getMessageConverters();
    HttpHeaders headers = new HttpHeaders();
    headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
```

Ilustración 60: método para crear reserva del controlador del administrador

Una vez se ha llegado al controlador, éste hace uso del @*RequestMapping* como se ha visto anteriormente para tramitar el servicio solicitado mediante la url del servidor y la extensión para el servicio REST.

Los datos se reciben del servidor mediante DTOs (data transfer object), estos objetos se utilizan para almacenar datos en comunicación mediante interfaces remotas, como es nuestro caso, no contienen ningún tipo de lógica de negocio que requiera pruebas generales.

Los datos son recogidos por el controlador del administrador mediante la clase *ResponseEntity* que implementa Spring.

```
ResponseEntity<ReservaDTO> respRdto = restTemplate.exchange(URL+urlReser, HttpMethod.POST, entity, ReservaDTO.clas
ReservaDTO rdto = respRdto.getBody();

List<Dia> horario = new ArrayList<>();
if(rdto != null){
    List<ReservaDTO> muestra = new ArrayList<>();

    String url = URL+"/administrador/reservas";
    muestra = getListReservas(url);

    horario = getHorario(muestra);
}

model.addAttribute("reservas",horario);

return "admin/perfilAdmin";
```

Ilustración 61: obtención de datos del servidor

Si la respuesta es afirmativa, los datos recogidos son enviados al método que crea el horario de reservas y se envían a la vista del cliente para que el administrador pueda gestionar esta nueva cita, sino es afirmativa no se creará el proceso de generación del horario.

- **Aplicación cliente**

La aplicación cliente se ha desarrollado en la tecnología híbrida ionic, para ello se ha usado 'visual studio code'[23], que es un editor de código desarrollado por windows, esta tecnología está desarrollada sobre Angular Js y Apache Cordova.

En sus inicios AngularJS basaba su estructura en MVC (Model View Controller), pero con el paso de los años se ha orientado más hacia MVVM (Model View ViewModel), este patrón se caracteriza por tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación.

La diferencia entre MVC y MVVM es que mientras MVC se separan los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos, cuando la lógica de negocio realiza un cambio, es necesario que ella sea la que actualiza la vista. Sin embargo, en MVVM se

[23] https://code.visualstudio.com/

separan los datos de la aplicación, la interfaz de usuario, pero en vez de controlar manualmente los cambios en la vista o en los datos, estos se actualizan directamente cuando sucede un cambio en ellos, por ejemplo, si la vista actualiza un dato que está presentando se actualiza el modelo automáticamente y viceversa.

A continuación, vamos a explicar cómo es la estructura de un proyecto ionic.

Lo primero que hacemos es crear un proyecto en blanco, al crear este proyecto se generan automáticamente una serie de estructuras de archivos y carpetas. Una vez se ha creado el proyecto en la carpeta 'src' es donde se ubica el contenido de la aplicación.

Principalmente, tiene esta estructura:

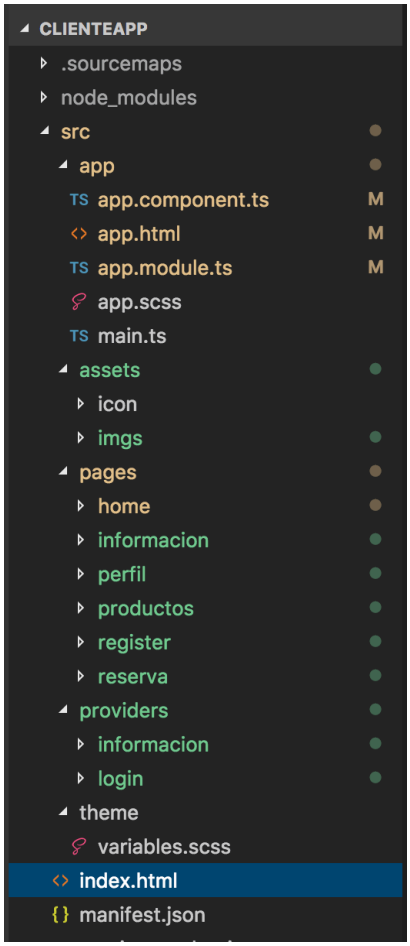

Ilustración 62: Estructura proyecto ionic

- En la carpeta *app* se encuentra el módulo principal del proyecto, es donde se definen las clases y componentes que vamos a utilizar. En primer lugar nos encontramos el fichero *app.component.ts* este fichero nos ha servido para la implementación del menú de navegación lateral, en él se definen los métodos de flujo de vistas.

```
@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage:any = HomePage;
  @ViewChild(Nav) nav:Nav;

  constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {
    platform.ready().then(() => {
      // Okay, so the platform is ready and our plugins are available.
      // Here you can do any higher level native things you might need.
      statusBar.styleDefault();
      splashScreen.hide();
    });
  }

  goReservas(){
    this.nav.setRoot(ReservaPage);
  }

  goProductos(){
    this.nav.setRoot(ProductosPage);
  }

  goInformacion(){
    this.nav.setRoot(InformacionPage);
  }

  goPerfil(){
    this.nav.setRoot(PerfilPage);
  }
```

Ilustración 63: Estructura fichero app.component.ts

Una vez definidos estos métodos, en el siguiente fichero *app.html* se muestra la vista que tendrá el menú de navegación y la llamada a los métodos anteriores.

```
<ion-menu [content]="content">
        <ion-header>
          <ion-toolbar>
            <ion-title>Menu</ion-title>
          </ion-toolbar>
        </ion-header>
        <ion-content>
          <ion-list>
            <button ion-item (click)="goReservas()">
              Reservas
            </button>
            <button ion-item (click)="goProductos()">
              Productos
            </button>
            <button ion-item (click)="goInformacion()">
              Informacion
            </button>
            <button ion-item (click)="goPerfil()">
              Perfil
            </button>
          </ion-list>
        </ion-content>
      </ion-menu>

      <ion-nav id="nav" #content [root]="rootPage"></ion-nav>
```

Ilustración 64: Estructura fichero app.html

Por último está el fichero *app.module.ts* este fichero es importante ya que en él se definen las vistas y clases que vamos a utilizar durante el desarrollo de la aplicación, en la siguiente imagen podemos ver su contenido.

```
@NgModule({
  declarations: [
    MyApp,
    HomePage,
    RegisterPage,
    ReservaPage,
    ProductosPage,
    InformacionPage,
    PerfilPage
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    HomePage,
    ReservaPage,
    RegisterPage,
    ReservaPage,
    ProductosPage,
    InformacionPage,
    PerfilPage
  ],
  providers: [
    StatusBar,
    SplashScreen,
    {provide: ErrorHandler, useClass: IonicErrorHandler},
    LoginProvider,
    InformacionProvider
  ]
```

Ilustración 65: Estructura fichero app.module.ts

Todas estas clases han sido previamente incluidas en el fichero en su cabecera.

- *assets* es la carpeta que contiene los recursos de la aplicación.

- En *pages* se encuentran todas las vistas de la aplicación, estas vistas se componen de 3 ficheros: un fichero *.html* donde se define la forma de la vista, otro fichero *.scss* donde se encuentran los estilos utilizados en la vista y un archivo *.ts* donde está la lógica de la vista. A continuación, se muestra una imagen con el desglose de la carpeta de la vista *home.*

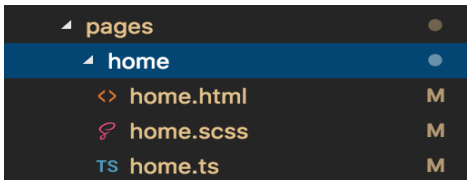

Ilustración 66: Estructura vista home

La comunicación entre vista y lógica se realiza mediante variables definidas en la lógica y mostradas en la vista, en las siguientes imágenes se muestra un ejemplo de uso.

En la vista para el perfil del cliente, definimos dos variables, que van a contener la información a mostrar, esta información puede estar almacenada de varios tipos y tenemos que adecuar el trato de la información al tipo recibido.

```
usuario: any;
historial: any;

constructor(public navCtrl: NavController, public navParams:
                    public menuCtrl: MenuController, private provider
}
```

Ilustración 67: Definición de variables en vista perfil

Estas variables son mostradas en la vista del perfil del cliente de la siguiente manera, al ser *historial* una lista tenemos que recorrerlo para mostrar su contenido:

```
<ion-content padding>
    <ion-title>Información del cliente</ion-title>

    <h6><p><strong> Nick: </strong>{{usuario?.nick}}</p></h6>
    <h6><p><strong> Nombre: </strong>{{usuario?.nombre}}</p></h6>
    <h6><p><strong> email: </strong>{{usuario?.email}}</p></h6>
    <h6><p><strong> Teléfono: </strong>{{usuario?.telefono}}</p></h6>
    <h6><p><strong> Fecha de nacimiento: </strong>{{usuario?.nacimiento}}</p></h6>
    <br>
    <ion-title>Historial de reservas</ion-title>
    <ion-list>

    <ion-item *ngFor="let h of historial">
      <div *ngIf="h?.aceptada == true then aceptada else cancelada"></div>
          <ng-template #aceptada> <p><strong>Fecha: </strong>{{ h?.fecha }} Aceptada</p></ng-template>
          <ng-template #cancelada><p><strong>Fecha: </strong>{{ h?.fecha }} Cancelada</p></ng-template>
    </ion-item>

  </ion-list>
```

Ilustración 68: Vista de variables del cliente

- *providers* contiene los servicios que conectan con la API de la aplicación servidor, estos servicios son utilizados por las vistas en su fichero *.ts*

  En la siguiente imágen se puede ver cómo se utilizan los servicios que proporciona la API para la gestión de información del establecimiento.

```
@Injectable()
export class InformacionProvider {

  path : string = 'http://localhost:8081/PeluServer/serRest/administrador/';

  constructor(public http: HttpClient) {
    console.log('Hello InformacionProvider Provider');
  }

  loadInformacion(){
    return this.http
    .get(this.path+'informacion');
  }

  loadProductos(){
    return this.http
    .get(this.path+'productos');
  }

  loadReservas(){
    return this.http
    .get(this.path+'reservas');
  }
```

Ilustración 69: Estructura de gestión de servicios de información

- Por último, *themes* suministra los estilos generales de la aplicación.

  Una vez terminado el desarrollo de la aplicación, podemos generar la aplicación tanto para el sistema Android como para IOs.
  Esto nos servirá para poder ejecutar nuestra aplicación y hacer uso de ella en cualquier dispositivo.
  Lo primero que tenemos que hacer es añadir las plataformas al proyecto mediante:

  - $ *ionic cordova platform add android*
  - *$ ionic cordova platform add ios*

  Cada plataforma contiene el soporte necesario para generar la aplicación en el sistema operativo que se está indicando.
  Una vez añadida la plataforma se genera la aplicación de la siguiente forma:
  - *$ ionic build android –release*
  - *$ ionic build ios --release*

  El archivo apk[24] se encuentra en Plataformas\Android\Build \Outputs\apk
  Para la instalación de esta apk en nuestro dispositivo hemos tenido que realizar los siguientes pasos:

[24] extensión de los paquetes de instalación de aplicaciones para Android

En nuestro dispositivo, tenemos que dar permiso de ejecución para aplicaciones que no vengan de la play store.

Una vez realizado esto, conectamos el dispositivo con el pc mediante un cable usb.

Cuando se haya conectado, enviamos la apk a nuestro dispositivo y la instalamos.

Una vez hayamos realizado estos pasos podremos utilizar la aplicación en el dispositivo.

5. Pruebas

En este punto se van a describir las pruebas que se han llevado a cabo para verificar y validar el desarrollo.

En primer lugar, se han realizado pruebas unitarias a nivel de software en los métodos de las clases para comprobar su correcto funcionamiento de manera aislada.

Luego se han realizado pruebas de integración entre los distintos componentes del sistema, con estas pruebas verificamos que los componentes de la aplicación funcionan correctamente actuando en conjunto.

Por último. pruebas del sistema, en ellas se corrobora el correcto funcionamiento del sistema completo para cada historia de usuario que haya sido definida, estas historias de usuario han sido un total de 15 historias de usuario para la aplicación del cliente y 9 para la aplicación del administrador.

Al término de cada sprint, se entregó un prototipo con el correcto funcionamiento de las historias de usuario descritas, las cuales fueron probadas y validadas por el cliente.

Una vez se iban implementando cada servicio disponible en la API del servidor, este servicio era probado mediante la aplicación Postman[25], con esta aplicación podemos saber el correcto funcionamiento del servicio, así como los datos que han sido devueltos en cada momento por los 20 servicios disponibles en el servidor.

A continuación, se puede ver un ejemplo de petición realiza con esta aplicación al servicio disponible en la aplicación servidor, en la primera imagen se solicita un producto mediante su *id*, el servicio retorna el producto con id solicitado al servidor, por otra parte, en la segunda imagen se puede ver la devolución de usuario cuando este no se encuentra almacenado.

[25] https://www.getpostman.com/

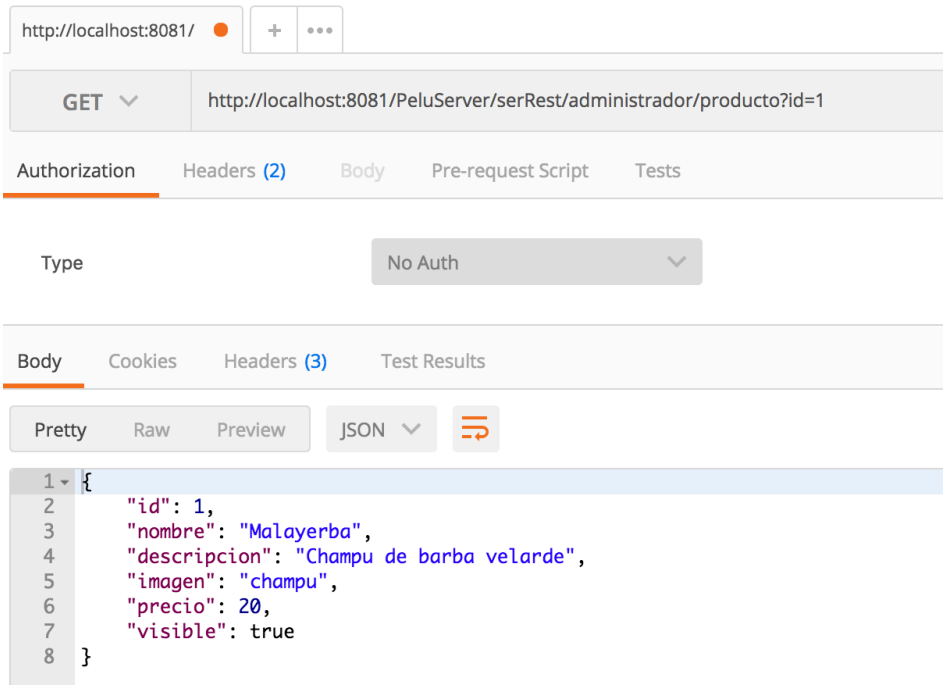

Ilustración 70: imagen de servicio REST disponible

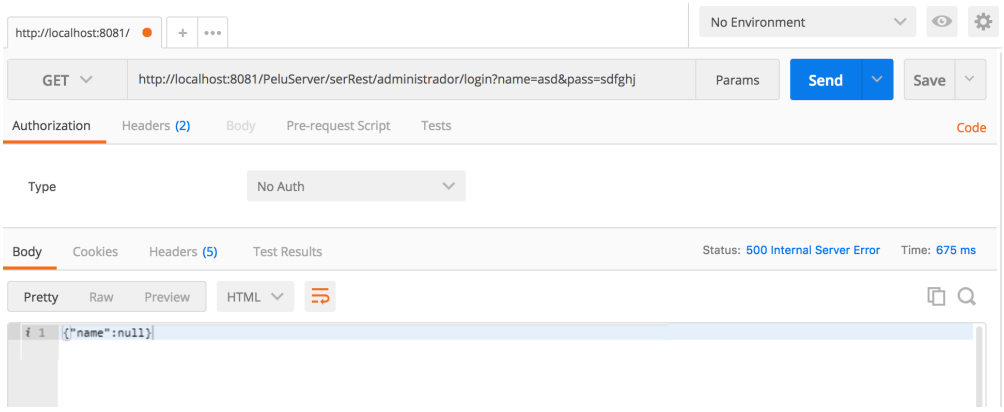

Ilustración 71: imagen de servicio REST usuario inválido

Finalmente, al término de la solución el cliente quedó satisfecho con el trabajo realizado y surgieron nuevas ideas que se irán añadiendo al proyecto en futuras actualizaciones, como poder tener varios establecimientos dentro de la misma aplicación.

En el apartado 6.1 se muestran todas estas posibles ideas surgidas para futuras actualizaciones de la aplicación.

6. Conclusiones

En la actualidad el crecimiento tecnológico va en aumento año tras año, eso hace que cada vez se puedan dar más servicios a la sociedad, estos servicios van centrados a automatizar procesos diarios de una manera sencilla, fiable y eficaz.

Con esta solución automatizamos el servicio de solicitud de reservas de citas aumentando la eficiencia del servicio.
En el transcurso de la implementación de la solución podemos concluir que se han cumplidos los objetivos que se plantearon y la priorización de las historias de usuario:

- Se ha desarrollado un estudio para el desarrollo de una solución que satisfaga la demanda de necesidades del cliente, extrayendo ventajas e inconvenientes de las opciones disponibles para su implementación. Además, se han mantenido reuniones con el cliente para obtener las necesidades del cliente y entregar prototipos a lo largo de la ejecución del problema hasta su solución.

- Se ha diseñado un sistema informático definiendo sus funcionalidades mediante historias de usuario creadas a partir de requisitos obtenidos mediante reuniones con el cliente y un estudio de viabilidad temporal y económica que dotarán de estabilidad al proyecto. Todo esto se ha realizado bajo el marco de la metodología de desarrollo ágil SCRUM.

- Se ha realizado la implementación de una aplicación móvil utilizando la tecnología híbrida ionic, esta tecnología utiliza Angular para generar aplicaciones disponibles para Android e iOs.

- Se ha desarrollado una aplicación web que será utilizada por el adminsitrador para la gestión de recursos del establecimiento, esta aplicación está desarrollada en Spring MVC, es una tecnología conocida ya que se ha estudiado en diferentes asignaturas del grado en ingeniería informática.

- Por último, se han realizado pruebas para conocer el grado de satisfacción de los requisitos planteados con el cliente, adaptándose la planificación a los imprevistos surgidos.

Desde el punto de vista personal, el desarrollo de esta aplicación me ha servido para conocer la importancia del trabajo para el que me he formado y que realizaré en los próximos años.

Al inicio del desarrollo, me enfrenté con problemas como diseño del modelo de datos que fue resuelto gracias a los conceptos aprendidos en mi formación.

Un concepto diferente ha sido el desarrollo de la aplicación al ser el framework *ionic* una tecnología que no hemos visto en el curso, para ello fui aprendiendo conceptos del lenguaje en foros especializados mientras iban surgiendo los problemas, una vez había adquirido conocimiento necesario pude avanzar más rápido con la aplicación.

Estas aplicaciones híbridas son una tecnología que está actualmente en vanguardia gracias a su soporte para varias plataformas, el poder utilizar un mismo código para ambas aplicaciones y el poder utilizar recursos del sistema hacen de esta tecnología una buena opción si queremos desarrollar aplicaciones móviles.

El uso de JPA para generar de manera automática las tablas de la base de datos me ha servido para conocer mejor su funcionamiento y generar una estructura sólida de desarrollo.
El uso de metodologías ágiles ha sido de gran ayuda para saber bien que tarea realizar en cada momento, como establecer un horario para el desarrollo de la solución.

La implementación de este proyecto me servirá en el futuro ya que se usan tecnologías que están actualmente entre las primeras opciones escogidas por las empresas más grandes del sector en sus proyectos, aparte, de que me ayudara en mi carrera profesional.

Trabajos futuros

En esta sección vamos a hablar de tareas que se han descartado para inclusión en esta primera versión de la aplicación, así como nuevas ideas para futuras actualizaciones del proyecto.

Para la aplicación del administrador se ha quedado sin desarrollar:

- Gestión de usuarios: esta tarea sería la que se incluiría en una próxima actualización del servidor, por otra parte, también sería necesaria una actualización del modelo de datos si se desea incluir el pago al establecimiento desde la aplicación del cliente.

En la aplicación del cliente encontramos las siguientes tareas:

- Pago de la reserva: esta tarea sería necesaria para una próxima actualización, también se pensaron en nuevos requisitos mientras se estaba haciendo el desarrollo de la solución, uno de ellos sería la reserva de productos al igual que la compra de productos desde la aplicación.
- Bonificaciones: una vez se descartó la inclusión de esta tarea en esta primera versión, se estudió un cambio para esta tarea, siendo un sistema de ofertas y bonificaciones una posible solución para esta tarea en la próxima actualización.

- Sistema de avisos y notificaciones cuando el administrador haya aceptado o cancelado nuestra solicitud de reserva de cita.

En el futuro si varios establecimientos quisieran tener este sistema en sus establecimientos, la idea sería poder registrarse en varios establecimientos y poder realizar las consultas a sus horarios de manera fluida dentro de la aplicación, todo esto llevaría a poder desarrollar una red social para este tipo de establecimientos donde cada cliente pudiera seguir a varias marcas y recibir ofertas en su dispositivo móvil.

Una propuesta más sería la inclusión de videos y fotos del trabajo realizado en el local, al igual que un sistema de comentarios donde poder dar el cliente su opinión sobre el servicio recibido.

Con todas estas ideas se podría crear una aplicación muy completa y con mucha probabilidad de éxito en el mercado.
Mediante el diseño, tecnologías utilizadas y experiencia en el desarrollo de este proyecto, no sería difícil la inclusión de estos nuevos requisitos en la aplicación.

7. Bibliografía

[1] *Informe Mobile en España y en el Mundo 2017.*ditrendia. [Online] Disponible en https://ditrendia.es/informe-mobile-espana-mundo-2017/ Acceso: Julio, 2018

[2]*Informe Anual 2017 Dispositivos y comunicaciones móviles.* Gobierno de España. [Online] https://www.ccn-cert.cni.es/informes/informes-ccn-cert-publicos/2826-ccn-cert-ia-10-18-informe-ciberamenazas-2017-y-tendencias-2018-dispositivos-moviles-dispositivos-y-comunicaciones-moviles/file.html. Acceso: Julio, 2018

[3] *desarrollo ágil de software*. wikipedia. [Online] Disponible en https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software. Acceso: Julio, 2018

[4] P. González, Apuntes de la asignatura Desarrollo Ágil, 2017.

[5] *Manifesto for Agile Software Development*. [Online] Disponible en: http://agilemanifesto.org/ Acceso: Jul, 2018

[6] *Qué es scrum*. proyectosagiles. [Online] Disponible en: https://proyectosagiles.org/que-es-scrum/ Acceso: Jul, 2018

[7] Juan Delgado (2012) *Ventajas e inconvenientes de metodologías Ágil y Predictiva* [Online] Disponible: http://www.itmplatform.com/es/blog/ventajas-e-inconvenientes-de-metodologias-agil-y-predictiva/ Acceso: Julio, 2018

[8] *Los beneficios de Implementar la Metodología Ágil.* Intelligence to business. [Online]. Disponible: http://www.i2btech.com/blog-i2b/tech-deployment/los-beneficios-de-implementar-la-metodologia-agil/. Acceso: Julio, 2018

[9] Walter Lara (2015) *Metodología Scrum: Cómo funciona la metodología de trabajo Scrum* [Online] Disponible: https://platzi.com/blog/metodologia-scrum-fases/ Acceso: Julio, 2018

[10] *Proceso y Roles de Scrum.* Softeng. [Online] Disponible en: https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum/proceso-roles-de-scrum.html. Acceso: Julio, 2018

[11] *Aplicaciones web vs aplicaciones escritorio.* webprogramacion. [Online] Disponible en: https://webprogramacion.com/356/blog-informatica-tecnologia/aplicaciones-web-vs-aplicaciones-de-escritorio.aspx. Acceso: Julio, 2018

[12] *Apps Híbridas vs Nativas vs Generadas. ¿Qué decisión tomar?.* InnovaAge [Online] Disponible en: https://www.innovaportal.com/innovaportal/v/696/1/innova.front/apps-hibridas-vs-nativas-vs-generadas-que-decision-tomar. Acceso: Julio, 2018

[13] Apps nativas vs Apps híbridas vs Apps generadas, ventajas y desventajas. Obux [Online] Disponible en: https://obux.wordpress.com/2017/02/14/apps-nativas-vs-apps-hibridas-vs-apps-generadas-ventajas-y-desventajas/ Acceso: Julio, 2018

[14] *Historias de usuario*. wikipedia. [Online] Disponible en https://es.wikipedia.org/wiki/Historias_de_usuario. Acceso: Julio, 2018

[15] *Scrum: ¿Qué es el Product Backlog?.* programacionymas. [Online] Disponible en: https://programacionymas.com/blog/scrum-product-backlog. Acceso: Julio, 2018

[16] Pitu Sabadí (2017) *HISTORIAS DE USUARIO PARA DEFINIR LOS REQUISITOS DE UN PROYECTO* [Online] Disponible: http://scrumizate.com/post/14/historias-de-usuario-para-definir-los-requisitos-de-un-proyecto. Acceso: Julio, 2018

[17] *Sucesión de Fibonacci*. wikipedia. [Online] Disponible en: https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci Acceso: Julio, 2018

[19] Lucho Salazar (2016) *Planear sprints en horas y no por puntos* [Online] Disponible en: http://www.gazafatonarioit.com/2016/11/planear-sprints-en-horas-y-no-por-puntos.html Acceso: Julio, 2018

[20] Pablo Soneira. *Técnica de priorización MoSCoW.* [Online] Disponible en: *http://www.laboratorioti.com/2016/09/26/tecnica-priorizacion-moscow/* Acceso: Julio, 2018

[21] Miguel. *LA VELOCIDAD EN EL SPRINT DE SCRUM.* [Online] Disponible: http://managementplaza.es/blog/la-velocidad-sprint-scrum/. Acceso: Julio, 2018

[22] *Installiing ionic* IonicFramwork. [Online] Disponible en: https://ionicframework.com/docs/intro/installation/ Acceso: Julio, 2018

[23] *Estudio de viabilidad de un proyecto.* [Online] Disponible en: https://www.obs-edu.com/es/blog-project-management/causas-de-fracaso-de-un-proyecto/estudio-de-viabilidad-de-un-proyecto-como-y-por-que-llevarlo-cabo Acceso: Julio, 2018

[24] *sueldo de analista en España* [Online] Disponible en: https://www.indeed.es/salaries/Analista-de-sistema-Salaries Acceso: Julio, 2018

[25] *sueldo de programador en España* [Online] Disponible en: https://www.indeed.es/salaries/Programador/a-junior-Salaries Acceso: Julio, 2018

[26] *modelo de dominio.* [Online] Disponible en: *https://repositorio.grial.eu/bitstream/grial/1153/1/8.%20Modelo%20de%20dominio.pdf* Acceso: Julio, 2018

[27] Miguel Angel Macas. *Diseño de software* *https://www.monografias.com/trabajos73/diseno-software/diseno-software.shtml* Acceso: Julio, 2018

[28] *diagrama de secuencia* [Online] Disponible en https://es.wikipedia.org/wiki/Diagrama_de_secuencia Acceso: Julio, 2018

[29] *modelo entidad relación.* [Online] Disponible en https://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n Acceso: Julio, 2018

[30] Leonardo De Seta. *introducción a los servicios web RESTful* *https://dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful* Acceso: Julio, 2018

[31] *Diagrama de clases* [Online] Disponible en https://es.wikipedia.org/wiki/Diagrama_de_secuencia Acceso: Julio, 2018

[32] *WSDL* [Online] Disponible en https://es.wikipedia.org/wiki/WSDL Acceso: Julio, 2018

[33] *data access objetc* [Online] Disponible en: https://es.wikipedia.org/wiki/Objeto_de_acceso_a_datos Acceso: Julio, 2018

[34] *Model View ViewModel* [Online] Disponible enhttps://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93modelo_de_vista Acceso: Julio, 2018

[35] token de seguridad: [Online] Disponible en: https://es.wikipedia.org/wiki/Token_(inform%C3%A1tica) Acceso: Julio, 2018

[35] *diagrama de estados* [Online] Disponible en: https://es.wikipedia.org/wiki/Diagrama_de_estado Acceso: Julio, 2018

[36] *Cliente-Servidor* [Online] Disponible en: https://es.wikipedia.org/wiki/Cliente-servidor Acceso: Julio, 2018

[37] *Transferencia de Estado Representacional* [Online] Disponible en: https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional Acceso: Julio, 2018

[38] *Java bean a XML con JAXB* [Online] Disponible en: http://chuwiki.chuidiang.org/index.php?title=Java_bean_a_XML_con_JAXB Acceso: Julio, 2018

[39] Yanina Muradas M. Conoce qué es Spring Framework y por qué usarlo qué usarlo. [Online] Disponible en https://openwebinars.net/blog/conoce-que-es-spring-framework-y-por-que-usarlo/ Acceso: Julio, 2018

[40] *APIs in a Few Words* [Online] Disponible en https://smartbear.com/learn/performance-monitoring/api-endpoints/ Acceso: Julio, 2018

8. Índice de ilustraciones

9. Índice de tablas

10. APÉNDICE I: Manual de instalación del sistema

Aplicación servidor

Inicialmente se requiere tener instalado un servidor de MYSQL Server, lo podemos instalar directamente desde la página web de mysql[26]. Los parámetros de la base de datos son:

spring.datasource.url: jdbc:mysql://localhost:1527/BBDDpeluqueria_db
spring.datasource.username: root

spring.datasource.password: root

Es necesario tener instalado Netbeans para probar la aplicación, ya que, en un entorno de producción, solo habría que desplegar el fichero .WAR generado por Netbeans en un servidor. El siguiente paso es pulsar sobre “Archivo”- “Importar proyecto” y abrir la carpeta llamada “AdministrdorBack” que contiene la aplicación del servidor que está en el USB entregado.

Después pulsar sobre “Ejecutar proyecto”.

Aplicación del administrador

Inicialmente tenemos que tener ejecutándose la aplicación servidor.

Una vez hayamos ejecutado el servidor, el siguiente paso es pulsar sobre “Archivo”- “Importar proyecto” y abrir la carpeta llamada “AdministrdorFront” que contiene la aplicación del servidor que está en el USB entregado.

Después pulsar sobre “Ejecutar proyecto”.

Aplicación móvil

Primero acceder con consola de comandos a la carpeta suministrada en el USB llamada “clienteApp”, después ejecutar: Ionic serve. Este comando abrirá el navegador con la aplicación.

Además, se incluye en la carpeta Platforms el proyecto compilado para Android o IOS.

[26] https://www.mysql.com/

11. APÉNDICE II: Manual de usuario

En este apartado vamos a realizar un manual de usuario de las aplicaciones que hemos desarrollado, tanto la aplicación del administrador como la aplicación del cliente.

Vamos a comenzar por la aplicación del servidor, donde entraremos con una cuenta de administrador y se observará el flujo de la aplicación, al igual que las distintas opciones que hay en cada vista.

Aplicación Administrador

Al ejecutar la aplicación, lo primero que podemos ver es un formulario de acceso, en él solo el usuario con rol de administrador podrá tener acceso a la aplicación.

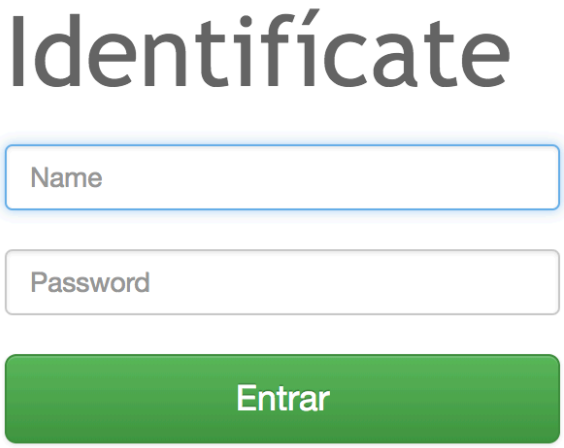

Ilustración 72: Vista de acceso a la aplicación administrador

Una vez se ha realizado el acceso a la aplicación, podremos ver el horario del establecimiento, este horario se ha definido por el cliente a 1h por cita y 5 días de visualización. Este horario es interactivo, cuando le pulsemos a cada cita del horario podremos ver un menú diferente según el estado en el que se encuentre la cita. Verde cita disponible, Rojo cita reservada, Naranja cita pendiente de aceptación

A continuación, se pueden ver las imágenes del horario en cada uno de los estados en los que el administrador se puede encontrar una cita.

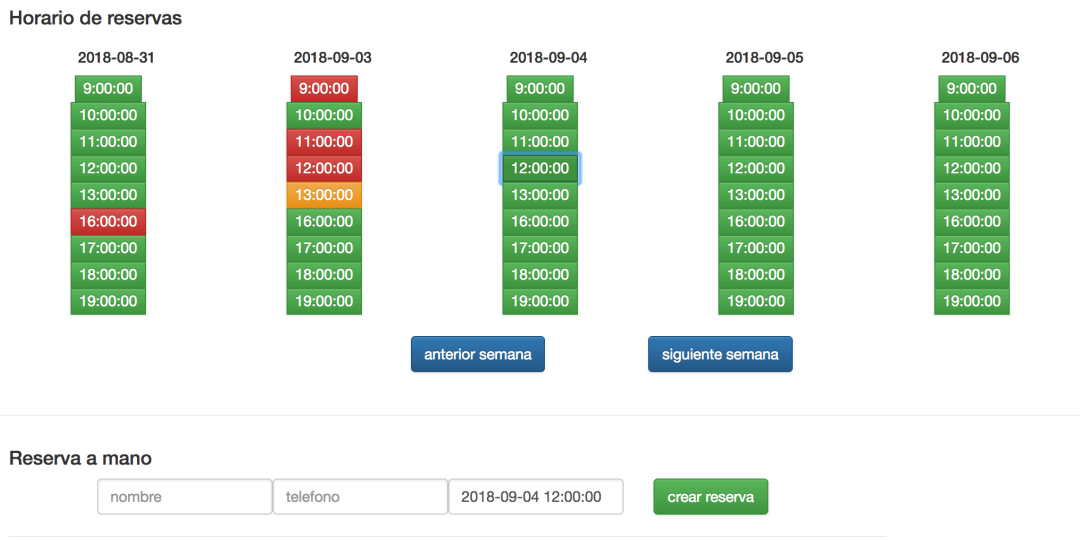

Ilustración 73: Vista de reservas de la aplicación administrador

La hora de la cita se coloca automáticamente dependiendo de la hora pulsada, en los botones de anterior semana, siguiente semana, podemos ver los siguientes 5 días laborales y su horario de citas asignado.

El administrador debe de introducir el nombre de usuario y teléfono para realizar una reserva, esta reserva será aceptada directamente y se asignará al usuario esté incluido en base de datos o no.

Ilustración 74: Vista 2 de reservas de la aplicación administrador

Al cancelar una reserva, esta hora se colocará en verde, siendo posible realizar otra reserva a la misma hora si lo considera oportuno.

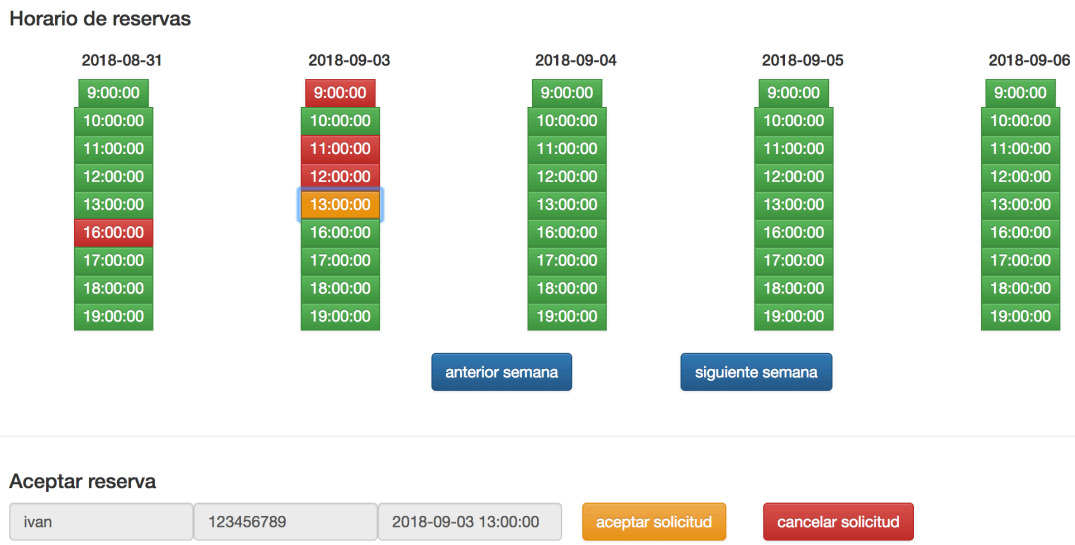

Ilustración 75: Vista 3 de reservas de la aplicación administrador

Cuando el cliente solicita una reserva, ésta le llega al administrador, en el que puede ver el nombre del cliente y su teléfono, el administrador puede elegir entre aceptar la reserva o cancelar la solicitud.

Una vez vistas todas las opciones que tenemos para la gestión de reservas, vamos a ver las opciones disponibles para la gestión de productos.
En esta vista tenemos un buscador de producto por id, un desglose de los productos almacenados en base de datos, donde tenemos la opción de editar cada producto y un botón de acceso al formulario de registro de producto.

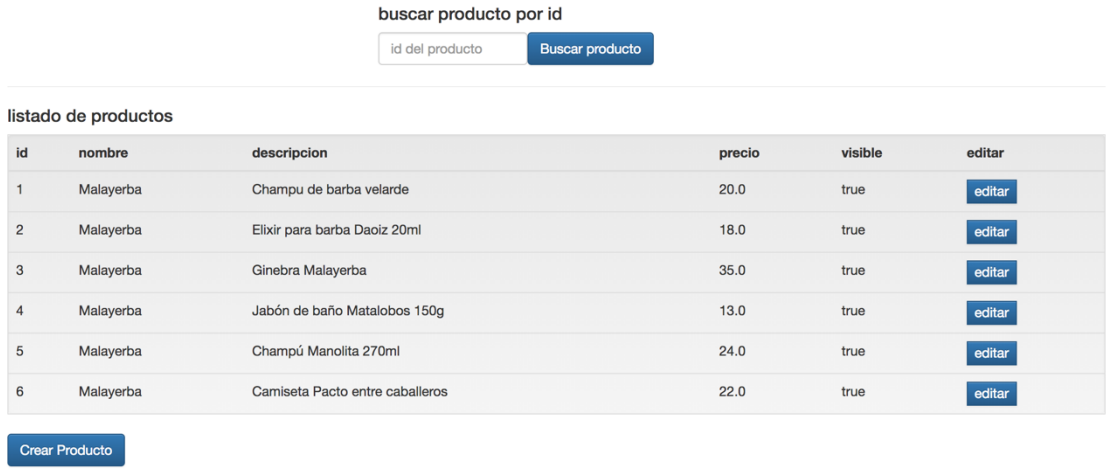

Ilustración 76: Vista de productos de la aplicación administrador

Cuando le pulsamos al botón editar de un producto, tenemos un formulario donde podemos editar o eliminar el producto.

Ilustración 77: Vista 2 de reservas de la aplicación administrador

Cuando pulsemos en el botón de crear producto, tendremos un formulario para realizar la inserción del producto en la base de datos, con el campo visible podemos gestionar la visibilidad o no del producto en la aplicación del cliente.

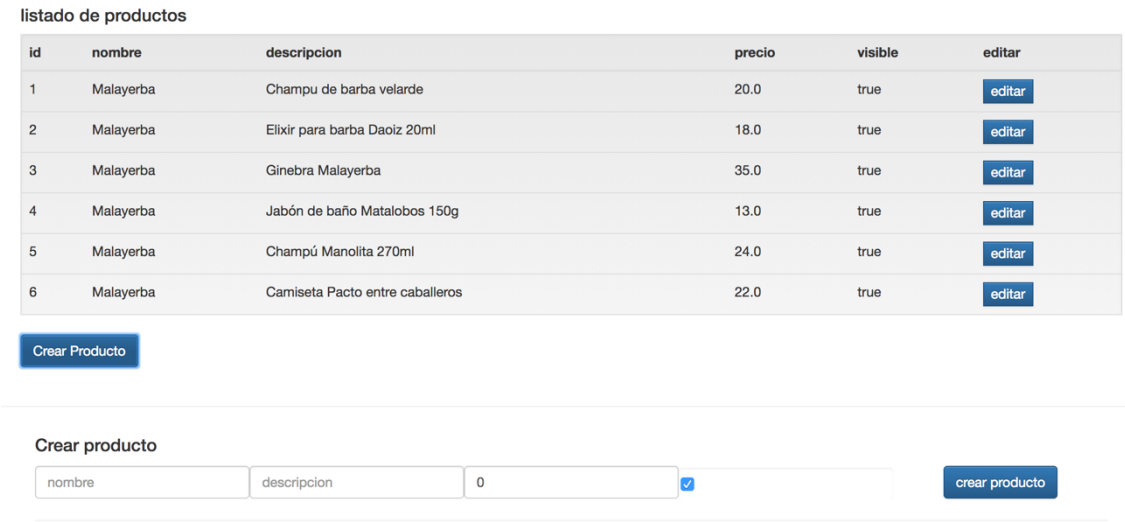

Ilustración 78: Vista 3 de reservas de la aplicación administrador

Una vez realizada la inserción del producto, podremos ver el producto en la lista de productos almacenados en base de datos.

Otra opción que tenemos disponible para la aplicación del administrador es la vista de clientes, donde tenemos a nuestra disposición un buscador de cliente por nombre. Cuando se busque un cliente, éste aparece desglosado y el historial de reservas del cliente, también podemos editar la habilitación del cliente en la aplicación, con esta funcionalidad, el administrador puede tener el control de los clientes que pueden tener acceso a la aplicación para la solicitud de reservas.

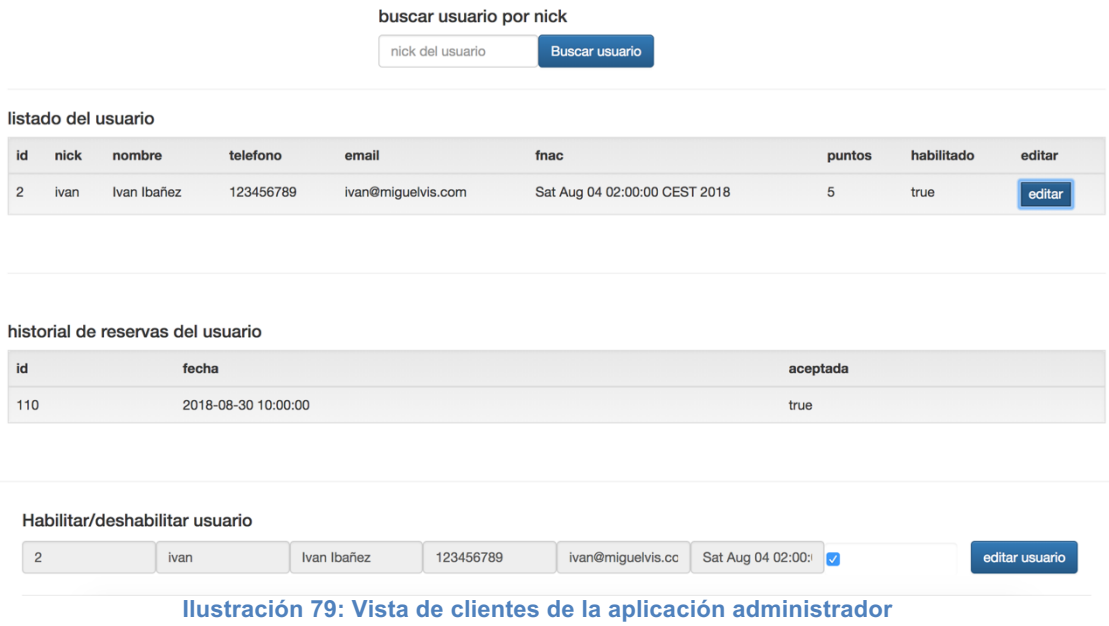

Ilustración 79: Vista de clientes de la aplicación administrador

La última opción que tenemos disponible en la aplicación, es la gestión de la información asignada al establecimiento que aparecerá en la aplicación del cliente, tenemos un formulario donde se podrán editar todos los campos de información.

Ilustración 80: Vista de información de la aplicación administrador

Una vez vistas todas las opciones que tenemos en la aplicación del administrador para la gestión del establecimiento, vamos a ver las vistas y el flujo disponible en la aplicación de los clientes.

Aplicación cliente

Esta aplicación está disponible tanto para Android como para IOs, ya que es una aplicación híbrida, las vistas serán iguales en cada uno de los distintos sistemas.
Lo primero que vemos en la aplicación es el formulario de acceso, si es un cliente ya registrado, introduce su Nick en el sistema y contraseña con el que entrará al menú de la aplicación.

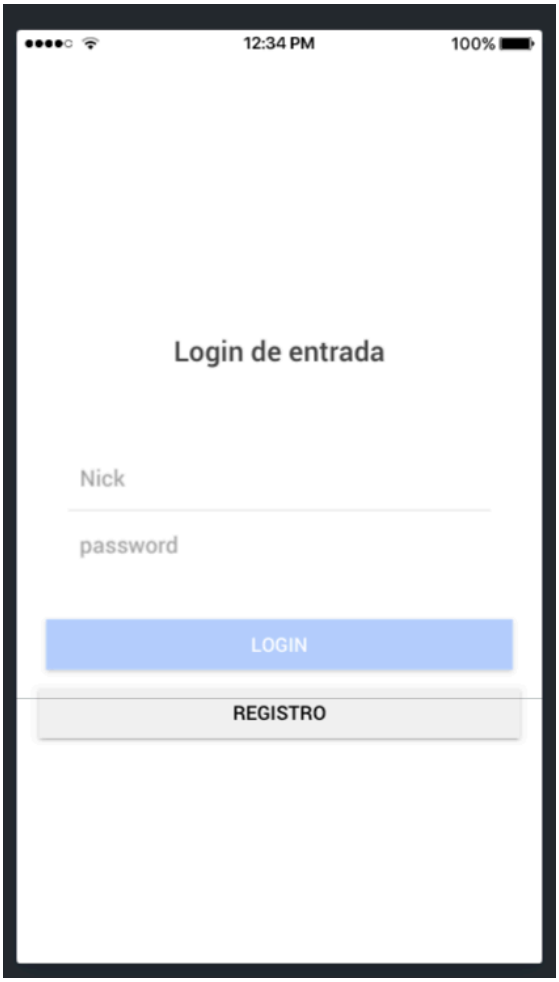

Ilustración 81: Vista de entrada a la aplicación cliente

Un cliente sin registro en el sistema podrá hacerlo mediante el formulario de registro que proporciona la aplicación.

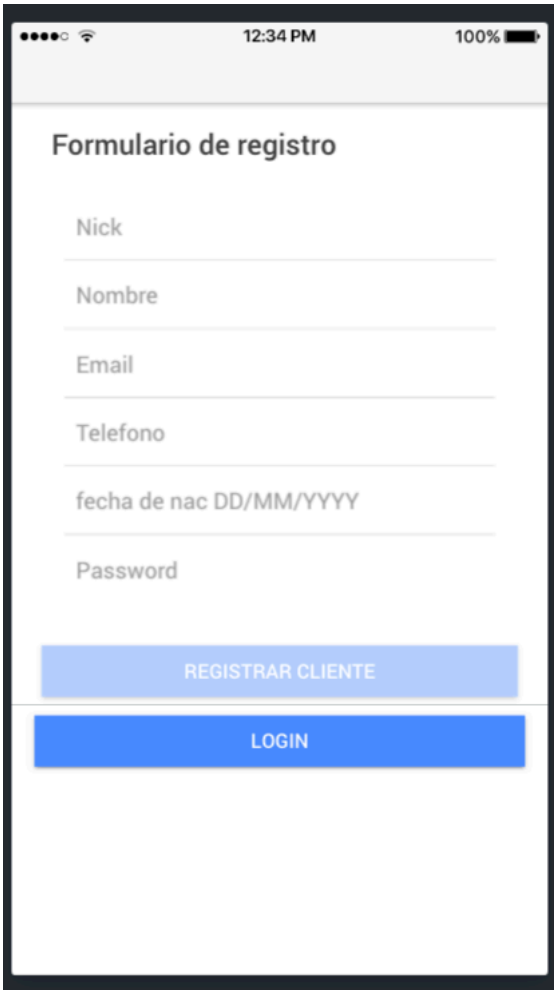

Ilustración 82: Vista de registro a la aplicación cliente

Una vez realizado el acceso a la aplicación, tenemos un menú donde tenemos acceso a las distintas vistas de la aplicación.

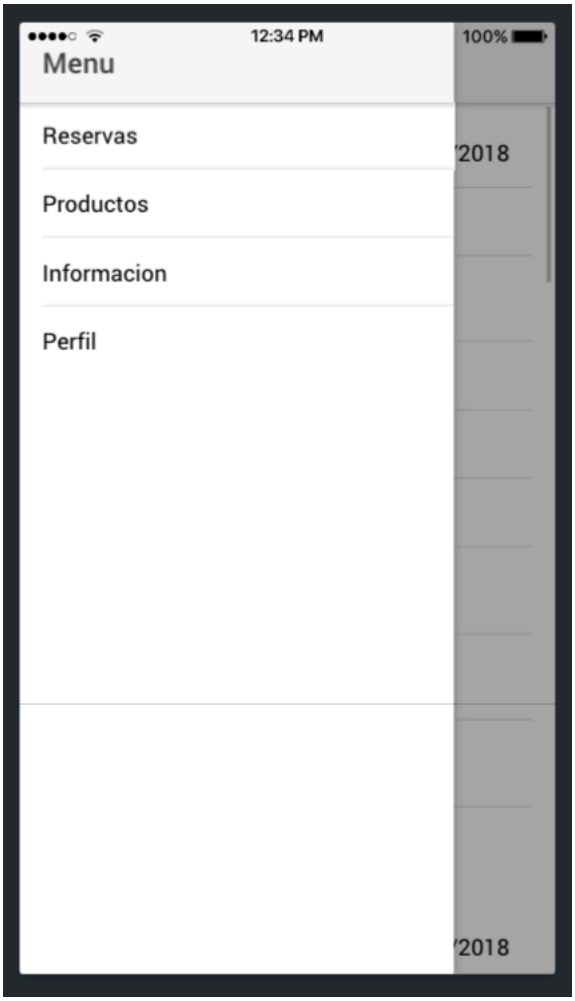

Ilustración 83: Menú lateral de la aplicación cliente

Mediante este menú lateral se tendrá acceso a todas las vistas de la aplicación.

La primera vista que vamos a ver es la vista de reservas, en esta vista se puede ver el horario de reservas del establecimiento en un intervalo de 5 días, este intervalo fue definido en los requisitos de la aplicación por el administrador, el horario de reservas es el mismo que puede ver el administrador en su aplicación, una reserva de cita no contestada, aparecerá como reservada para los demás usuarios de la aplicación.

Los clientes tienen a su disposición un botón verde para hacer efectiva la solicitud de reserva en las horas que están disponibles, un cliente solo podrá tener activa una reserva solicitada a la vez.

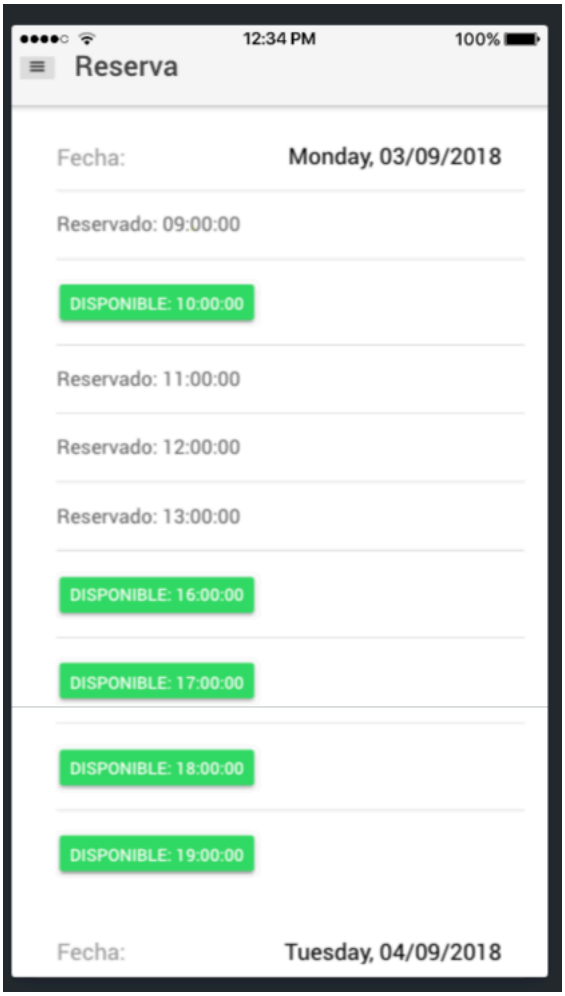

Ilustración 84: Vista de reservas de la aplicación cliente

Una vez realizada la reserva, que en nuestro caso será a las 17:00 podremos ver un mensaje de solicitud realizada con éxito.

Si navegamos hasta la vista de perfil, tenemos un formulario para la edición de los datos del cliente, la reserva que tenemos actualmente solicitada, que podemos cancelarla cuando lo desee el cliente, además de un historial con las reservas registradas en el sistema.

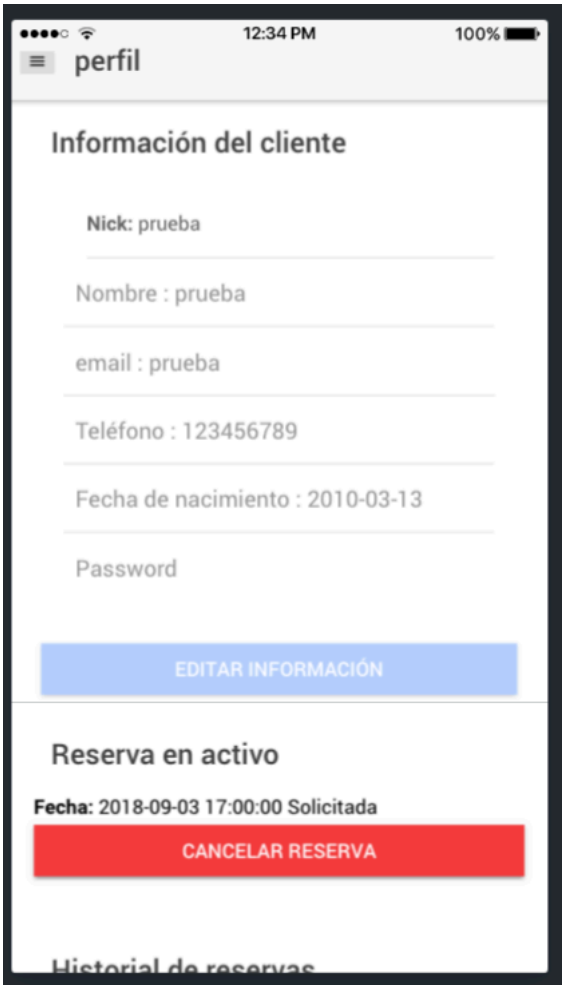

Ilustración 85: Vista del perfil de la aplicación cliente

La siguiente vista, es la vista de los productos que podemos comprar en el establecimiento, estos productos tienen información, además del precio y una imagen. Estos productos son los registrados por el administrador en el sistema.

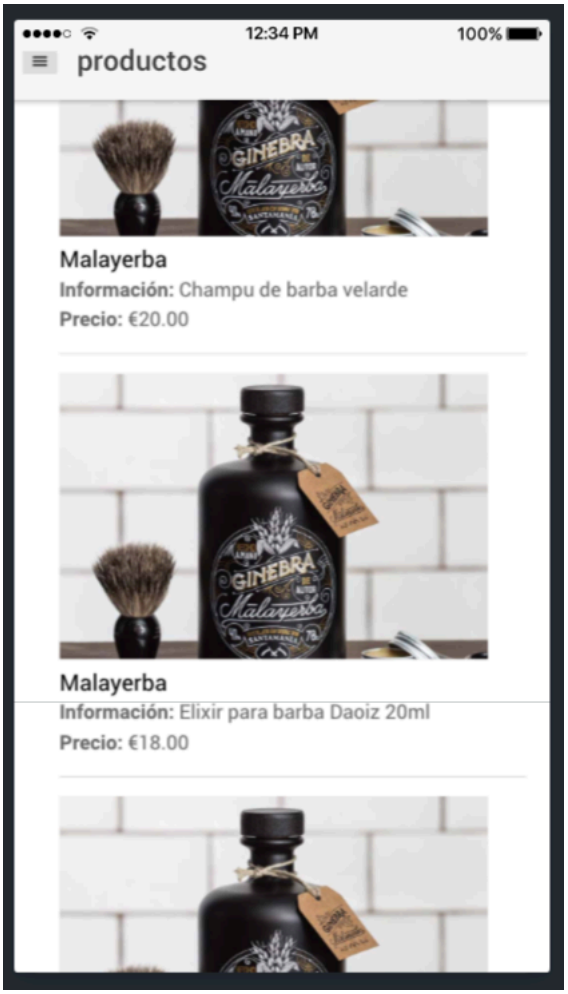

Ilustración 86: Vista del perfil de la aplicación cliente

Por último, encontramos la vista la información del establecimiento, en esta vista podemos ver información referente al establecimiento que administrador ha registrado en la base de datos, al igual que un botón donde nos llevará a *google maps* [27]con la ubicación del local.

[27] https://www.google.es/maps

Ilustración 87: Vista de información de la aplicación cliente

12. APÉNDICE III: Manual de contenidos suministrados

Se suministra un USB que contiene los siguientes archivos comprimidos:

- Aplicación del servidor llamada AdministrdorBack.zip
- Aplicación del administrador llamada AdministrdorFront.zip
- Aplicación del cliente llamado ClienteApp.zip
- Documentación final llamada TFG- Documentacion.pdf