Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

ESTUDIO Y DESARROLLO DE UN PROTOTIPO DE APLICACIÓN WEB PARA LA GESTIÓN DE CITAS EN UNA CLÍNICA DENTAL

Alumno: María Martínez López

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

Septiembre, 2018

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don JOSÉ RAMÓN BALSAS ALMAGRO, tutor del Trabajo Fin de Grado titulado: ESTUDIO Y DESARROLLO DE UN PROTOTIPO DE APLICACIÓN WEB PARA LA GESTIÓN DE CITAS EN UNA CLÍNICA DENTAL, que presenta MARÍA MARTÍNEZ LÓPEZ, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, SEPTIEMBRE de 2018

El alumno: Los tutores:

MARÍA MARTÍNEZ LÓPEZ JOSÉ RAMÓN BALSAS ALMAGRO

Índice ilustraciones

Índice tablas

1. INTRODUCCIÓN

1.1. Motivación

Por lo general no existen muchas aplicaciones web para facilitar tanto al odontólogo como al cliente la solicitud de cita online. Se suelen implementar aplicaciones de escritorio orientadas al odontólogo, es decir, que la puedan utilizar únicamente en la clínica dental, y no suele haber aplicaciones web para la gestión de citas.

Tras estar hablando con mi hermana, estudiante de 4º curso de odontología, me ayudó a decidirme a realizar esta aplicación web, ya que me informó de cómo estaba actualmente este sector, además de que me gustaría que la usara para su futura clínica.

El hecho también de esta elección es porque en un futuro cercano me gustaría dedicarme a trabajar en el desarrollo de aplicaciones web y me gustaría profundizar en aprender aún más sobre este tema.

1.2. Objetivos

Los objetivos que inicialmente se plantearon para este proyecto son los siguientes:

- Realizar un estudio de necesidades y soluciones existentes para el prototipo de una aplicación web para la gestión de citas en una clínica dental.
- Diseñar un prototipo de aplicación web especialmente orientado a la utilización del *framework* Spring, utilizando los módulos Spring MVC y Spring Security, usando el patrón MVC para el desarrollo de este prototipo y siguiendo la metodología *SCRUM*.
- Documentar la memoria del proceso seguido con las fases principales de análisis, diseño e implementación del proyecto.

1.3. Metodología

Para comenzar cualquier proyecto es necesario definir cuál será la metodología de trabajo, seleccionando aquella que mejor se adapte a las características específicas del proyecto y del equipo de trabajo. Por ello hemos optado por seguir una de las diferentes metodologías ágiles que hay, *SCRUM.*

SCRUM [1], se trata de un proceso ágil que nos permite centrarnos en ofrecer el más alto valor de negocio en el menor tiempo. Permite inspeccionar rápida y repetidamente el trabajo real en el software (cada 2/4 semanas) avanzando mediante *Sprint (*análogo a las iteraciones*)*.

Esta metodología tiene una serie de características destacables tales como son los roles en los que se pueden distinguir; *Product owner*, *Scrum Master* y *Team*. En nuestro caso, el equipo de trabajo estará únicamente formado por mi persona, que a su vez realizo el papel de *Scrum Master*.

Otras características importantes antes de comenzar a hablar de ella, es conocer los artefactos de esta metodología:

- *Product Backlog,* se trata del bloque de trabajo, es decir, los requisitos del sistema o historias de usuario. Repriorizada al comienzo de cada *Sprint*.
- *Tareas* consiste en la división de las historias de usuario.
- Diagrama Burn-down indica la cantidad de tiempo estimada hasta la fecha de entrega.

Para entender mejor cómo funciona esta metodología visualizaremos la siguiente ilustración 1.

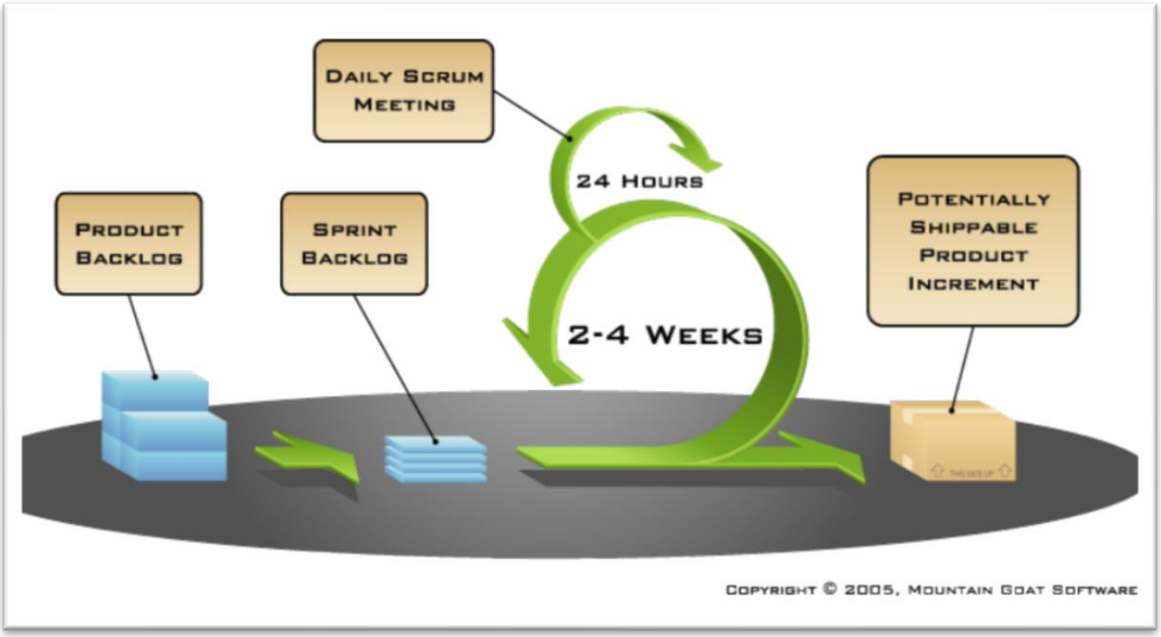


Ilustración 1. Proceso metodología Scrum.

1. Primeramente, se toman los requisitos que nos da el cliente, creando para cada requisito principal un bloque de trabajo, también llamado historia de usuario.
2. Posteriormente, el cliente ordena las historias de usuario en una pila de producto según la prioridad de cada una de ellas.
3. Después, se escogen una serie de historias de usuario y se trabajan con ella en un Sprint.
4. Tras finalizar cada Sprint, se le enseña al usuario el resultado. Volvemos al segundo paso, hasta finalizar el producto.

1.4. Estructura del documento

El presente documento se ha organizado con la siguiente distribución:

En el apartado 2, se encuentra el capítulo de análisis donde realizaremos un análisis para obtener los usuarios objetivos para las aplicaciones de clínicas dentales, además de conocer cómo está el estado del arte respecto a esta temática y los *frameworks* utilizados para aplicaciones web. Se estudiarán las historias de usuario, la planificación inicial del proyecto, su estudio de viabilidad y para finalizar este bloque mostraremos el modelo de dominio de la aplicación.

En el tercer apartado, diseño, realizaremos los diagramas para su posterior implementación tras obtener el análisis. Los diagramas que se indicarán son: el diagrama de clases, el diagrama de entidad – relación, diagramas de secuencia y, por último, diagrama de la interfaz.

En el cuarto apartado, se documentará la parte de implementación, explicando los diagramas de arquitectura de la aplicación, y entrando en los detalles más importantes de la parte de implementación.

En el apartado quinto, se resumirán las pruebas realizadas de las historias de usuario realizadas, usando criterios de aceptación.

En el apartado sexto, se concluirá con las conclusiones y trabajos futuros del trabajo fin de grado realizado.

En el séptimo apartado, bibliografía, se mostrarán las referencias bibliográficas.

Por último, se incluirán dos anexos, el anexo de los contenidos suministrados y el anexo de manual de usuario.

2. ANÁLISIS

2.1. Análisis preliminar

En primer lugar, estudiaremos las características del usuario objetivo para conocer quién es el usuario que va a utilizar nuestra aplicación, luego veremos el estado del arte actual tanto en las aplicaciones de gestión para la odontología como los *frameworks* y tecnologías usadas para este tipo de aplicaciones y por último, veremos las tecnologías que finalmente hemos decidido escoger para desarrollar nuestro proyecto.

2.1.1. Usuario objetivo

Actualmente, la mayoría de las personas están utilizando internet para su día a día y no hay sitio en el que no veamos a alguien con un aparato electrónico en la mano usando alguna aplicación web.

Este concepto ya no tiene edad, últimamente hay un aumento de gente mayor que se adapta con mayor facilidad al manejo de un móvil para no quedarse atrás y poder comunicarse con sus familiares. Además, no solamente en estos casos está incrementando su uso. En el caso de las empresas es habitual que tengan una página web para darse a conocer, para comunicarse o vender con sus clientes, por ello podemos decir que la utilización de la red es casi obligatoria para poder estar conectados con los demás teniendo la comodidad de hacerlo a través de nuestro teléfono móvil o *tablet*.

En la ilustración 2 se muestra un estudio sobre el uso de internet en el año 2017 en España, el 84,6% [2] de la población de entre 16 y 74 años utilizaron internet, esto supone que, como hemos dicho anteriormente, se confirma que más de la mitad de los españoles lo usan en su vida cotidiana.

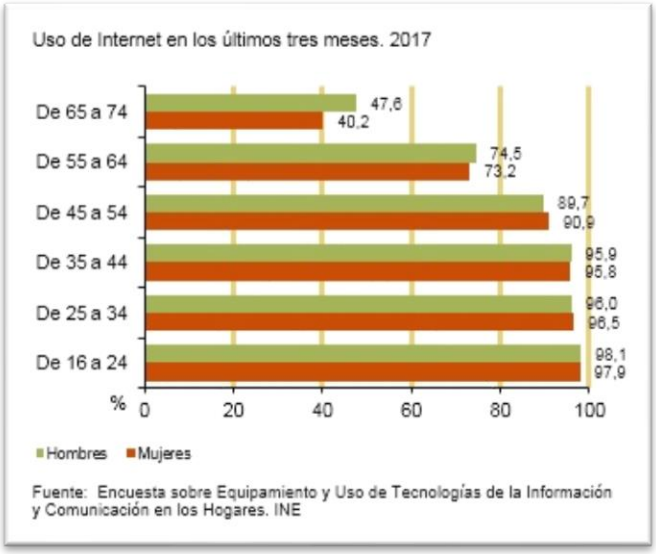


Ilustración 2. Gráfica del uso de internet en la población española en 2017.

En el caso de la odontología, también podemos decir que es uno de los sectores que cada vez está adquiriendo más importancia, ya que hay una gran mayoría de usuarios que se están animando no solo a mejorar su salud sino también su aspecto físico.

Según una encuesta [3] en la ilustración 3 se muestra que el 6% ha tenido problemas para reír/sonreír debido a la apariencia de los dientes.

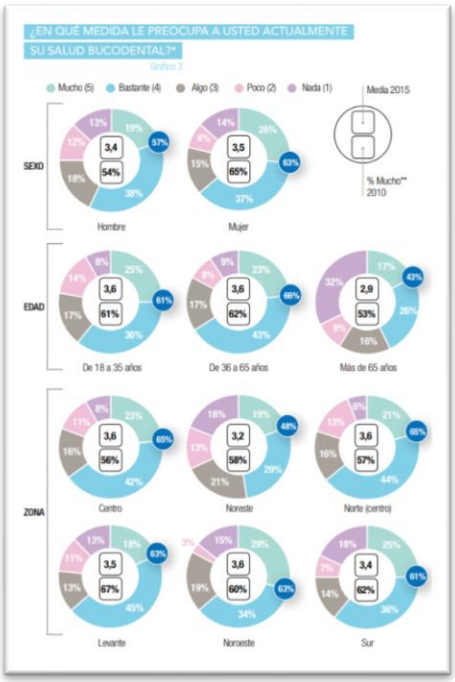


Ilustración 3. Comparativa de la preocupación por la salud bucodental.

Teniendo en cuenta todo lo dicho anteriormente y que tenemos que aprovechar nuestro tiempo, deberíamos tener más herramientas para facilitar el trato cliente – odontólogo. Como ocurre en otros sectores, cada vez más las propias clínicas suelen trabajar con un ordenador delante en el que vuelcan todos los datos de los pacientes. También a la hora de promocionarse a través de las redes o una página propia. Así pues, hay gran variedad de personas que buscan un dentista por internet para hacerse algún tratamiento o para comparar precio en diferentes sitios. Sin ir más lejos en las redes sociales hay chicas y chicos llamados *influencers* a los que siguen e imitan mucha gente, ellos se unen a la hora de cada vez más de promocionar los tratamientos dentales, que los odontólogos les ofrecen, para promocionar la clínica dental, y así ganar más clientes.

Veamos un ejemplo del crecimiento significativo a la hora de buscar un odontólogo a través de internet en la ilustración 4 [4].

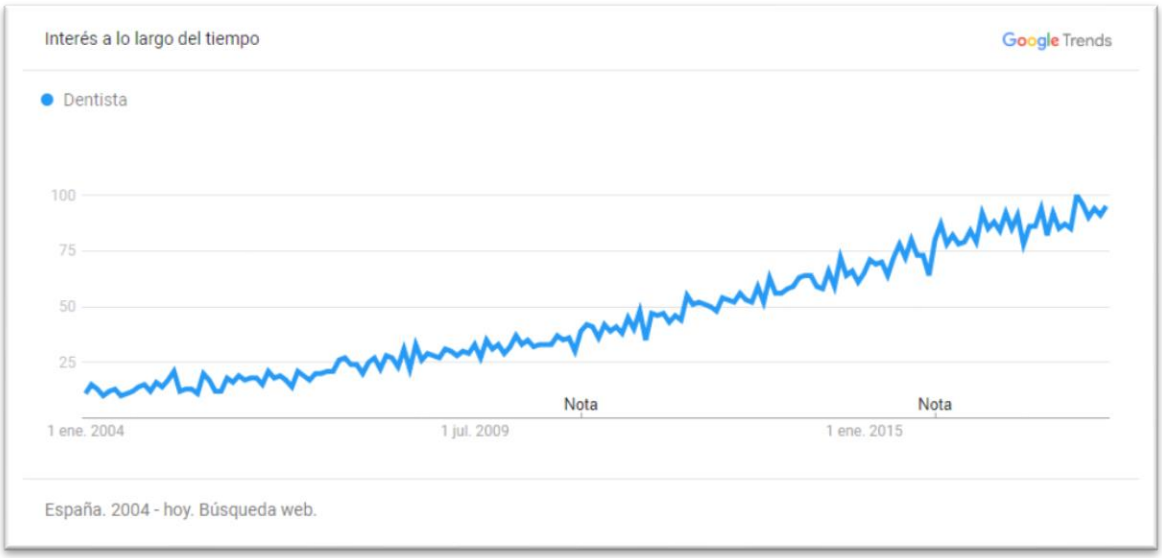


Ilustración 4.Crecimiento de búsquedas en Google de la palabra dentista.

Por ello, hoy en día hemos evolucionado cambiando la petición de cita a través de llamada telefónica o bien presentándose personalmente, a poder hacerlo también a través de internet, véase un ejemplo en la ilustración 5.

Ilustración 5. Ejemplo de solicitación de cita online en una clínica dental.

2.1.2. Estado del arte

En este apartado, lo dividiremos en dos partes, en primer lugar, hablaremos de las aplicaciones de gestión para la odontología que actualmente utilizan algunos dentistas, para saber que hay en el mercado, y, en segundo lugar, las tecnologías y *frameworks* que son tendencia hoy en día, para informarnos con cuál es mejor trabajar.

2.1.2.1. Aplicaciones de gestión para odontología

En este ámbito, lo más normal en una clínica es utilizar alguna herramienta software que nos facilite nuestra labor. Actualmente, como hemos indicado anteriormente, hay sitios web para promocionarse, pero pocas en las que puedas pedir cita online o tener tu propio perfil para ver cuándo es tu próxima cita. También existen herramientas con la finalidad de proporcionarles a los profesionales una mayor comodidad con la que trabajar con los datos recogidos. Un ejemplo es una herramienta que utilizan los alumnos del grado de odontología de la Universidad de Granada, se llama ***Natura+***, es una aplicación informática para la gestión de las historias clínicas de los pacientes, por temas de confidencialidad no se puede acceder, a no ser que sea el alumnado o el profesorado de esta facultad.

Otro ejemplo es ***OrisDent evo***[1], es una aplicación de escritorio centrada para clínicas dentales, está soportado tanto por Windows como Mac, la cual tiene una serie de características como son gestión de historias clínicas, de facturación y de citas, entre otras funcionalidades, desde la clínica dental.

En cambio, una aplicación web tiene una serie de ventajas como son: nos permite a los pacientes acceder para solicitar cita o consultar sus tratamientos desde casa o a través del móvil. Además, de a los propios odontólogos también pueden acceder a historiales, agendas sin necesidad de estar en la clínica.

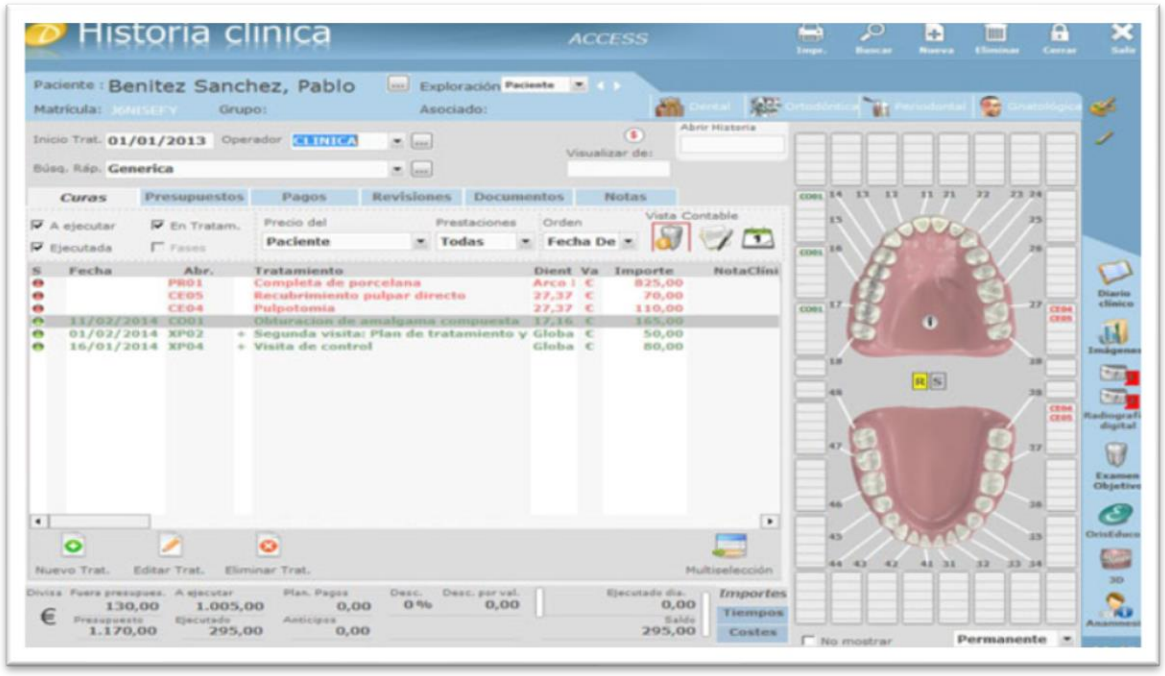


Ilustración 6. Ejemplo de historia clínica en OrisDent evo.

En la ilustración 6 vemos una de las funcionalidades nombradas anteriormente: un ejemplo de una historia clínica de un paciente, en la cual podemos observar los datos del paciente como es el nombre y apellidos, los tratamientos que se han hecho y el precio de ellos.

En la ilustración 4 del apartado anterior también vimos un ejemplo de página web en la que pedir cita online, en la que el usuario primero tiene que rellenar un formulario para encontrar la clínica dental más cercana de esa compañía, posteriormente éste elige la clínica, por último, el usuario debe identificarse con

[1] https://www.orisline.com/es/orisdent-evo-software-para-dentistas/ampliacion-orisdent-evo/

su usuario y contraseña si estuviera registrado, sino tendría que dar una serie de datos personales para finalmente en ambos casos acceder al sistema para pedir cita el día y hora deseada.

2.1.2.2. Tecnologías y framework

Hoy en día, existe una gran diversidad en cuanto a *frameworks* o herramientas para poder desarrollar una aplicación web con las diferentes opciones que busquemos.

En la ilustración 7 se puede observar algunos de los *framework* más conocidos que hay actualmente, y como han ido cambiando según su popularidad desde finales de 2014 hasta ahora. La media [5] se ha hecho con el número de estrellas que tiene el repositorio de un determinado *framework* en *GitHub* y el número de preguntas en *Stack Overflow* quedando como vemos en la ilustración 6, también hay que decir que en general estos *frameworks* se clasifican en *frameworks* para aplicaciones web clásicas basadas en el servidor, como es el caso de *Spring MVC*, *Ruby on Rails*, o bien en aplicaciones web modernas centradas en el cliente, ya sea *AngularJS*, *React*,etc.

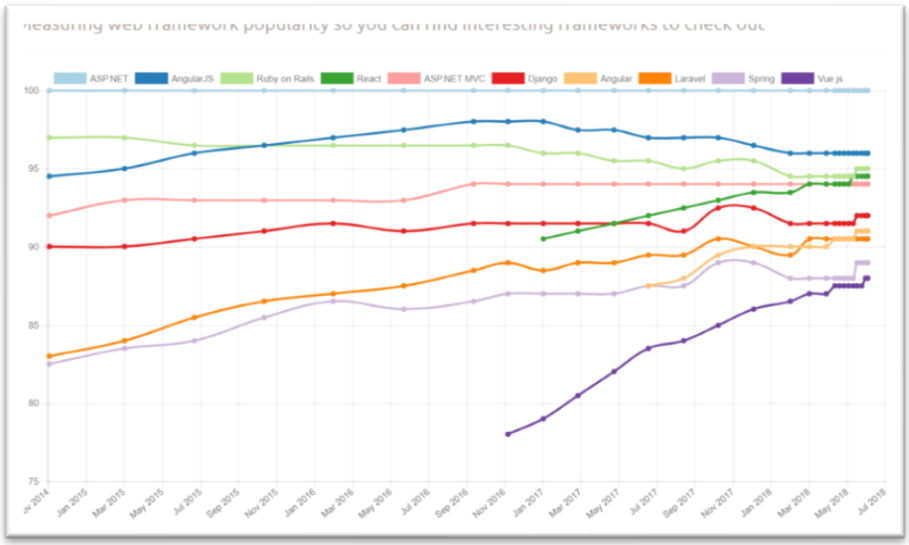


Ilustración 7. Ranking de puntos de diferentes framework.

Dicho esto, hablaremos de algunos de ellos.

Ilustración 8. Logo AngularJS

AngularJS[2] es un framework de JavaScript permitiendo el desarrollo en el lado del cliente, sigue la arquitectura MVC, además de permitir crear aplicaciones de una única página (SPA). Está mantenido por Google, nació en 2009 pero en 2012 se empezó hacer más popular.

Éste es distribuido como un archivo JavaScript y puede ser añadido en la página web con un script tag. Actualmente, su última versión estable es la 1.6.8.

Ilustración 9. Logo Ruby on Rails

Ruby on Rails[3], es un *framework OpenSource* escrito en el lenguaje Ruby siguiendo el patrón MVC. Éste combina Ruby con HTML, CSS Y JavaScript para crear aplicaciones web. Este *framework* tiene una mayor facilidad que otros *frameworks* porque no necesita configuraciones en archivos como ocurre en Java. Su última versión estable es 5.1.4. Algunos ejemplos de páginas web hechas con Ruby on Rails son Airbnb, twitch, github, entre otras.

Ilustración 10. Logo Spring Framework

Spring Framework[4], es el *framework* más popular de Java, también *OpenSource.* Inicialmente fue escrito por Rod Johnson, actualmente es un proyecto de la fundación SpringIO. Su última versión es la 5.

Este *framework* tiene alrededor de 20 módulos de los cuales se agrupan en 6 áreas, este modularidad deja a libre elección el uso de la parte del *framework* que más nos interese usar, sin necesidad de incluir todo el *framework*, por ejemplo, Spring MVC.

[2] https://angularjs.org/
[3] https://rubyonrails.org/
[4] https://spring.io/

2.2. Propuesta de solución

Después de hacer un análisis preliminar, tenemos claro que vamos a realizar un prototipo de aplicación web en la que diferentes usuarios puedan acceder para gestionar sus citas de forma online y el personal de la clínica que tenga lo necesario de manera más sencilla.

Como veremos en posteriores capítulos usaremos el patrón MVC[5], que es un patrón arquitectónico software, en la que se separan en tres partes diferenciadas el modelo, la vista y el controlador.

Seguiremos la metodología SCRUM como vimos en el apartado 1.2.

2.2.1. Tecnologías usadas

Tal y como hemos comentado en el apartado anterior, hay una gran variedad de *frameworks* para utilizar en nuestro proyecto, por ello vamos a hablar de las tecnologías o *frameworks* que vemos como buena elección en aplicación:

En este lugar los vamos dividir en dos apartados, por un lado, el lado de *front-end* y por otro el de *back-end*:

Así pues, vamos empezar por el lado del cliente, *el front-end*:

Usaremos Bootstrap porque es uno de los *framework* CSS que nos permite una facilidad y comodidad de uso por su compatibilidad con el diseño *responsive*.

También por supuesto el uso de HTML5, que se trata de la última versión de HTML.

Y para el lado del servidor, el *back-end*:

El uso del *Framework* Spring, comentado en el capítulo anterior, es el más conocido del lenguaje java, también con mucha importancia por su uso para

[5] MVC: MODELO-VISTA-CONTROLADOR

aplicaciones empresariales, este *framework* no se queda corto para su uso. Se usarán algunos de los módulos de éste como puede ser el de Spring MVC y Spring Security para el lado de la autenticación. Aunque su proceso de aprendizaje es elevado, como se ha visto durante asignaturas de la carrera, su elección facilitará en gran medida reducir el tiempo necesario para llevar a cabo el proyecto.

Para la base de datos cualquier SGBD relacionales, aunque para el desarrollo del prototipo se utilizará por su simplicidad Apache, y el servidor Apache Tomcat.

La elección no ha sido fácil, aunque ayudó el hecho de haberlas usado bastante en algunas asignaturas de la carrera, por ello prefiero utilizar tecnologías conocidas para tener la oportunidad de profundizar en ellas, evitando retrasos innecesarios en el aprendizaje, puesto que, en este caso, se adaptan perfectamente para desarrollar el sistema propuesto.

Además, tras lo dicho en el apartado anterior el *framework* elegido es Spring para la implementación de esta aplicación, usaremos algunos de sus módulos y algunos *framework* como son Bootstrap, JQuery, CSS, entre otros, para la parte del diseño de las vistas. Todo ello se detallará en el apartado de Implementación del presente documento.

2.3. Historias de usuario

Como estamos siguiendo una metodología ágil, SCRUM, debemos determinar las historias de usuario correspondientes.

Las historias de usuario [6] son un instrumento muy utilizado en las metodologías ágiles para obtener los requisitos de la aplicación a partir de la información obtenida de los usuarios, permitiendo su evolución y cambio a lo largo del proyecto.

Para ello seguiremos el modelo INVEST [6][7] para la descripción de las historias de usuario, se trata de una guía para la correcta escritura de las historias de usuario. Para el caso de la estimación se trata de una estimación de coste a nivel de desarrollo, para ello, se pueden usar o bien los puntos de historia o el tiempo ideal. Para este proyecto usaremos los puntos de historia [7] que son una unidad de medida para expresar el tamaño de trabajo.

ID	Título	Rol	Descripción	Estimación
1	Iniciar sesión	Administrador, Personal y Cliente (Usuario)	Como usuario quiero poder iniciar sesión para acceder a la plataforma.	5
2	Cerrar sesión	Administrador, Personal y Cliente (Usuario)	Como usuario quiero poder salir de la aplicación cuando desee.	2
3	Dar de alta cliente	Personal	Como personal quiero poder añadir clientes en su primera visita.	5
4	Dar de alta personal	Administrador	Como administrador quiero poder añadir el nuevo personal de la clínica.	5
5	Cambiar contraseña	Administrador, Personal, Cliente (Usuario)	Como usuario quiero poder modificar mi contraseña.	4
6	Editar datos personales	Administrador	Como administrador quiero poder editar los datos personales de los	2

[6] https://agileforall.com/new-to-agile-invest-in-good-user-stories/
[7] Independiente, Negociable, Estimable, Pequeña (Small), y Testeable.

			clientes o personal por si están equivocados	
7	Añadir cita	Administrador, Cliente	Como administrador o cliente quiero poder obtener una nueva cita.	5
8	Anular cita	Administrador, Cliente	Como administrador o cliente quiero poder borrar una cita cuando el cliente diga.	2
9	Añadir observación	Personal	Como personal quiero poder incluir observaciones cuando venga el cliente a su cita.	5
10	Ver observaciones	Personal	Como personal quiero poder ver las observaciones que se le han ido tomando al cliente en sus citas.	2
11	Visualizar listado de clientes	Administrador, Personal	Como administrador o personal quiero poder ver el listado de clientes.	2
12	Visualizar listado de personal	Administrador	Como administrador quiero poder ver el listado de personal.	2
13	Visualizar citas propias	Personal, Cliente	Como personal o cliente quiero poder ver mis citas.	2
14	Visualizar listado de citas	Administrador, Personal	Como personal o administrador quiero poder visualizar las citas que hay.	2

15	Crear odontograma de cada cliente	Personal	Como personal quiero añadir un odontograma a cada cliente en su primera cita.	4
16	Editar odontograma	Personal	Como personal quiero editar cada pieza de dientes de cada cliente cuando sea necesario.	4
17	Visualizar odontograma	Personal	Como personal quiero ver el odontograma de cada cliente cuando asista a consulta.	4

Tabla 1. Historias de usuario.

2.4. Planificación inicial de tareas

Ya que tenemos las historias de usuario y su estimación podemos realizar una planificación inicial de la descomposición de tareas para crear la pila del producto donde se encuentran las historias de usuario para cada Sprint.

Como hemos dicho en el apartado anterior, para la estimación hemos utilizado puntos de historia, por ello seguiremos una técnica de estimación [7] como es la técnica *Planning Poker* [8] propuesta por *Grenning* combina el juicio experto, la estimación por analogía y la desagregación en un enfoque que consigue estimaciones rápidas pero fiables, y será mediante una escala creciente como puede ser la serie de *Fibonacci*: 1, 2, 3, 5, 8, 13. Además tomaremos qué 1 punto de historia es igual a una jornada de trabajo de 6 horas.

Añadiremos a la planificación inicial una tarea principal descompuesta en 3 subtareas, llamada configuración del entorno.

ID	Historia de usuario	Tareas	Rol / Duración

[8] https://www.planningpoker.com/

	Configuración del entorno	1. Creación de proyecto MAVEN y Configuración del framework. 2. Conexión con apache-tomcat y Creación base de datos. 3. Documentación	Programador 2ph Programador 4ph Analista 13ph
1	Iniciar sesión	1. Creación vista login. 2. Creación del controlador login. 3. Configuración Spring security.	Programador 1ph Programador 1ph Programador 3ph
2	Cerrar sesión	1. Creación del controlador deslogueo.	Programador 2ph
3	Dar de alta cliente	1. Creación vista crear cliente. 2. Creación clase Cliente. 3. Creación DAO Cliente. 4. Creación del método creación cliente en el controlador registro.	Programador 1ph Programador 1ph Programador 2ph Programador 1ph
4	Dar de alta personal	1. Creación vista crear personal. 2. Creación clase Personal. 3. Creación DAO Personal. 4. Creación del método creación personal en el controlador registro.	Programador 1ph Programador 1ph Programador 2ph Programador 1ph
5	Cambio de contraseña	1. Creación vista cambiar contraseña. 2. Creación del método cambio de contraseña en el controlador registro v.	Programador 2ph Programador 2ph
6	Editar datos personales	1. Creación vista expediente. 2. Creación del método editar datos personales en el controlador administrador.	Programador 1ph Programador 1ph

7	Añadir cita	1. Creación vista crear cita. 2. Creación clase Cita. 3. Creación DAO Cita. 4. Creación del método crear cita en el controlador registro.	Programador 1ph Programador 1ph Programador 2ph Programador 1ph
8	Anular cita	1. Creación vista anular cita. 2. Creación del método anular cita en el controlador registro.	Programador 1ph Programador 1ph
9	Añadir observación	1. Creación vista añadir observación. 2. Creación clase Observacion. 3. Creación DAO Observacion. 4. Creación del método añadir observación en el controlador personal.	Programador 1ph Programador 1ph Programador 2ph Programador 1ph
10	Ver observación	1. Creación vista ver observación. 2. Creación del método ver observación en el controlador personal.	Programador 1ph Programador 1ph
11	Visualizar listado clientes	1. Creación vista listado clientes. 2. Creación del método listado clientes en el controlador clientes.	Programador 1ph Programador 1ph
12	Visualizar listado personal	1. Creación vista listado personal. 2. Creación del método listado personal en el controlador personal.	Programador 1ph Programador 1ph
13	Visualizar citas propias	1. Creación vista citas. 2. Creación del método citas en el controlador clientes y personal.	Programador 1ph Programador 1ph

14	Visualizar listado citas	1. Creación vista listado citas. 2. Creación del método citas en el controlador administrador.	Programador 1ph Programador 1ph
15	Crear odontograma	1. Creación vista crear odontograma. 2. Creación clase Odontograma. 3. Creación del DAO Odontograma. 4. Creación del método crear odontograma en el controlador registro.	Programador 1ph Programador 1ph Programador 1ph Programador 1ph
16	Editar odontograma	1. Creación vista editar odontograma. 2. Creación del método editar odontograma en el controlador cliente.	Programador 2ph Programador 2ph
17	Visualizar odontograma	1. Creación vista visualizar odontograma. 2. Creación del método visualizar odontograma en el controlador clientes.	Programador 2ph Programador 2ph

Tabla 2. Descomposición en tareas de historias de usuario.

Con un total de 76 puntos de historia, ahora se realizará una tabla en la que se indicarán los *Sprints*, la fecha de inicio y fin de cada Sprint, los puntos de historia para cada Sprint, y la velocidad de cada Sprint [7] que es una medida de ratio de avance del equipo.

Las tres últimas historias de usuario no se han podido realizar porque la estimación inicial no fue adecuada y algunas historias más prioritarias llevaron más tiempo del estimado. Al recalcular los puntos de historia nos quedaría un total de 64.

Para calcular el número de puntos de historia por Sprint, se divide el total de puntos de historia (64) entre el número de Sprint (7), aproximando (10) y dejando para el último Sprint (4) un mayor margen.

Empezamos en enero y terminamos en julio, teniendo en cuenta que en Scrum la duración de cada Sprint es de 2 a 4 semanas, estimaremos 4 semanas x 5 días semanales nos quedarían 20 puntos de historia por Sprint, como hemos indicado queda un margen para cambios. Para la velocidad de cada Sprint la obtendremos con el porcentaje de 10ph * 100 / 20ph.

Sprint	**Fecha inicio**	**Fecha fin**	**Puntos de historia**	**Velocidad**
1	8 enero 2018	31 enero 2018	10	50%
2	1 febrero 2018	28 febrero 2018	10	50%
3	1 marzo 2018	30 marzo 2018	10	50%
4	2 abril 2018	30 abril 2018	10	50%
5	1 mayo 2018	31 mayo 2018	10	50%
6	1 junio 2018	29 junio 2018	10	50%
7	2 Julio 2018	31 Julio 2018	4	20%

Tabla 3. Estimación de la velocidad por Sprint.

Para el seguimiento del plan de entrega se suele hacer un diagrama de evolución (burn down chart) como vemos en la ilustración 11, en el que no ha habido cambios respecto lo planeado.

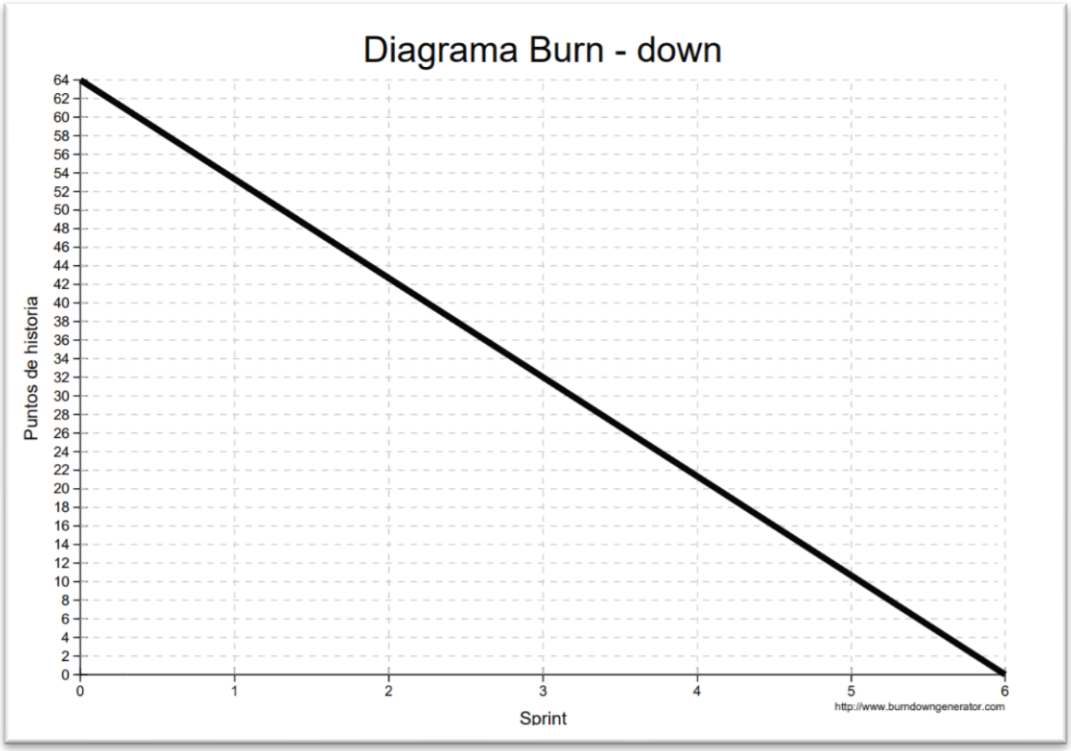


Ilustración 11. Diagrama burnDown

2.5. Estudio de viabilidad

En este apartado comentaremos si nuestro proyecto es viable. En cuanto a coste tenemos que saber los siguientes datos:

En la primera tabla veremos el software necesario para realizar el proyecto, y lo que nos costaría.

Licencia software	
Netbeans IDE 8.2	Gratuito
Spring Framework	Gratuito
Bootstrap	Gratuito
JQuery	Gratuito
Apache Derby	Gratuito
Apache Tomcat	Gratuito
Visual Paradigm	Gratuito
Axure	Gratuito

Tabla 4. Coste licencias software.

Lo siguiente necesario para realizar el coste sería el material hardware que se va a utilizar para que salga adelante nuestra aplicación, en este caso solo estamos hablando de un portátil.

Equipo	
Asus R510V	Según el presupuesto que hay actualmente en el mercado, este portátil tiene un valor de 895 euros y como su vida media es de 4 a 5 años y lo hemos utilizado durante 5 meses, su valor se reduciría a 82,87 euros.

Tabla 5. Coste equipo informático.

Ahora vamos a comentar los últimos puntos para realizar el coste, el precio del internet utilizado durante los siguientes meses, las nóminas del programador y el analista.

- **Acceso a internet** → El precio de acceso a internet es aproximadamente 56 euros mensuales, tras utilizarlo durante 5 meses esto ascendería a 280 euros.

- **Nóminas** → Según el convenio colectivo de Telefónica [8] publicado en BOE en la resolución del 15 de marzo de 2016, se puede conocer el sueldo base de un analista y un programador. Así podemos estimar con los datos que tenemos del apartado anterior lo que les correspondería de nómina a cada uno.

En el apartado anterior, hemos podido hacer una estimación de las horas que realizarían tanto el analista como el programador, por un lado, el analista son 8 puntos de historia que serían 13 jornadas de 6 horas lo que quedaría un total de 78 horas. Y, por otro lado, el programador sería lo restante de 64 puntos de historia – 13 puntos de historia del analista, lo daría como resultado un total de 51 puntos de historia, y haciendo lo mismo que para el analista 51 x 6 serían 306 horas totales.

	Horas totales	Nómina
Analista	78 horas	1050,4 euros
Programador	306 horas	4120,80 euros

Tabla 6. Nóminas programador y analista.

	Coste
Licencia software	0 euros
Equipo	82,87 euros
Acceso internet	280 euros
Nóminas	5171.20 euros
Coste	5534.07 euros
Coste total con margen beneficio	6000 euros

Tabla 7.Coste total estudio viabilidad.

2.6. Modelo de dominio

Una vez conocida la viabilidad del producto, pasaremos a estudiar la información que nuestro sistema va a gestionar. Para realizar el modelo de dominio o diagrama de clases conceptuales utilizaremos la notación UML.

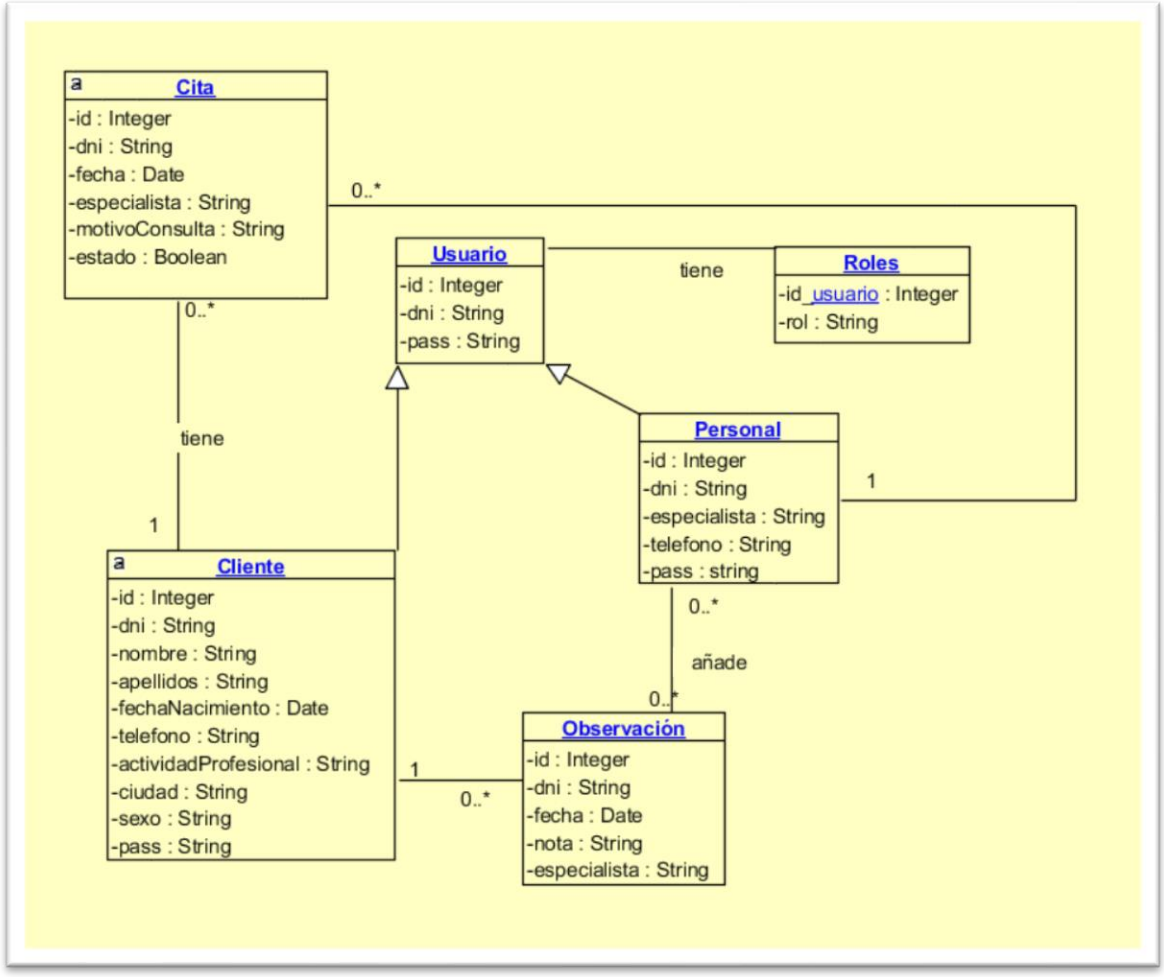


Ilustración 12. Diagrama modelo de dominio.

En la ilustración 12, podemos ver el modelo de dominio en el que se pueden distinguir claramente las clases que vamos a tener en nuestro sistema.

Primero tenemos la clase Usuario en la que tenemos el identificador de usuario, el DNI y la contraseña. De este heredan las clases Personal y Cliente con las claves en comunes, identificador y DNI. El cliente y personal son totalmente diferentes con respecto a sus atributos y también respecto al rol que los distingue el uno del otro. La clase Cita es la entidad en la que se almacena y gestiona las citas de los pacientes con sus respectivos especialistas (Personal), en ella hay un booleano que es el estado de la cita que puede ser pendiente o finalizada. Y por último tenemos la clase observación en la que el personal puede ir añadiendo después de cada cita con el paciente una nota de lo que se le ha hecho.

3. DISEÑO

En este capítulo veremos nuestra parte de diseño. Esta es una de las fases que componen el ciclo de vida de un software. Tras realizar un análisis de las funcionalidades definidas mediante las historias de usuario estas serán transformadas en una representación software. Para ello vamos a realizar una serie de diagramas como son: diagrama entidad - relación, diagrama de clases, diagramas de secuencia y por último diagrama de interfaz.

3.1. Diagrama Entidad – Relación

A continuación, veremos las tablas que forman nuestra base de datos, con el nombre del atributo, el tipo, la clave y la descripción de cada una de ellas.

- **Tabla Usuarios:**

Nombre del atributo	Tipo	Clave	Descripción
id	integer	Primary key	Identicador usuario.
dni	varchar		Número de identificador personal para acceso a la aplicación.
pass	varchar		Contraseña personal para acceso a la aplicación.

Tabla 8. Tabla usuarios. Entidad - Relación.

- **Tabla Roles:**

Nombre del atributo	Tipo	Clave	Descripción
Id_usuario	integer	Primary key	Identificador usuario.
rol	varchar		Tipo de usuario, distintos privilegios.

Tabla 9. Tabla Roles. Entidad - Relación.

- **Tabla Clientes:**

Nombre del atributo	Tipo	Clave	Descripción
id	integer		Identificador del cliente.
dni	varchar	primary key	Número de identificador personal del cliente.
nombre	varchar		Nombre del cliente.
apellidos	varchar		Apellidos del cliente.
fechanacimiento	date		Fecha de nacimiento del cliente.
actividadProfesional	varchar		Profesión del cliente.
sexo	varchar		Sexualidad del cliente.
ciudad	varchar		Ciudad donde vive el cliente.
telefono	varchar		Número de teléfono del cliente.
pass	varchar		Contraseña del cliente.

Tabla 10. Tabla Clientes. Entidad - Relación.

- **Tabla Personal:**

Nombre del atributo	Tipo	Clave	Descripción
id	integer		Identificador del personal de la clínica.
dni	varchar		Número de identificador personal del personal de la clínica.
especialista	varchar	Primary key	Nombre y apellidos del personal de la clínica.
telefono	varchar		Número de teléfono del personal de la clínica.
pass	varchar		Contraseña del personal de la clínica.

Tabla 11. Tabla Personal. Entidad - Relación.

- **Tabla Cita:**

Nombre del atributo	Tipo	Clave	Descripción
id	integer	Primary key	Identificador de la cita.
dni	varchar		Número de identificador personal del cliente.
fecha	timestamp		Fecha y hora de la cita.
consulta	varchar		Motivo de la consulta de la cita.
especialista	varchar		Nombre y apellidos del especialista.

Tabla 12. Tabla Cita. Entidad - Relación.

- **Tabla Observacion:**

Nombre del atributo	Tipo	Clave	Descripción
id	integer	Primary key	Identificador de la observación.
dni	varchar		Número de identificador personal del cliente.
fecha	date		Fecha de la observación.
nota	varchar		Tratamiento propuesto, nota del personal de la clínica
especialista	varchar		Nombre y apellidos del especialista.

Tabla 13. Tabla Observación. Entidad - Relación.

Después de ver todas las tablas que forman nuestra BBDD, vamos a ver cómo se relacionan entre ellas, ilustración 13:

Tras observar este diagrama, ilustración 13, podemos aclarar algunos aspectos relevantes que nos pueden surgir:

- En gran parte de las tablas se ha utilizado como clave primaria el id, ya que por ejemplo en el caso de la tabla cita no podemos usar el DNI como clave primaria, al poder tener un mismo cliente varias citas con distintos profesionales, esto nos facilita este tipo de casos. En otros casos, hemos tenido que utilizar otras claves primarias como son el DNI como podemos ver en la tabla cliente.
- Otro punto a destacar es que hemos utilizado una tabla llamada roles en la que tenemos el rol de cada usuario con su id_usuario. Esta tabla nos ayuda a la hora de buscar el rol de cada usuario para poder acceder a la plataforma.

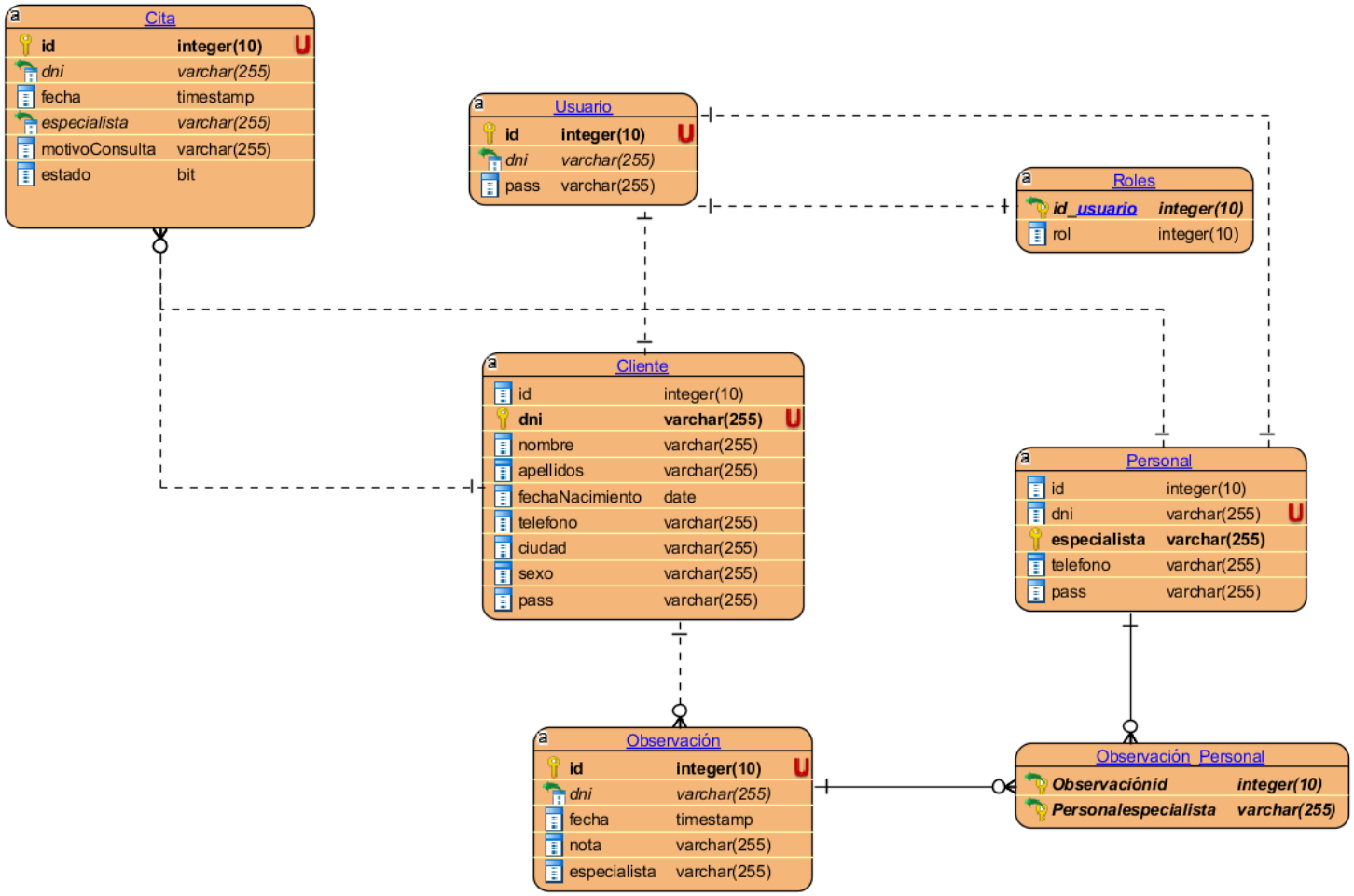


Ilustración 13. Diagrama Entidad - Relación.

3.2. Diagrama de Clases

Ahora veremos cómo están distribuidas nuestras clases utilizando el patrón modelo-vista-controlador, en el que se pueden ver las vistas totalmente diferenciadas de los controladores y del modelo. Para visualizarlos mejor se realizarán en paquetes como veremos en las siguientes ilustraciones.

Primero, en la ilustración 12, se puede observar la parte del modelo, en el que se engloban tanto las entidades como los DAO (Data Access Object).

En el último apartado del capítulo anterior pudimos ver el modelo de dominio, en el que pudimos observar las entidades que había en nuestro sistema, ahora lo podemos ver en el paquete modelo en el que no hay cambios respecto con el modelo de dominio.

También podemos observar en esta ilustración 14, los DAO: PersonalDAO, ClienteDAO, CitaDAO y ObservacionDAO, que se relacionan con la base de datos a través del DAOJDBC respectivo.

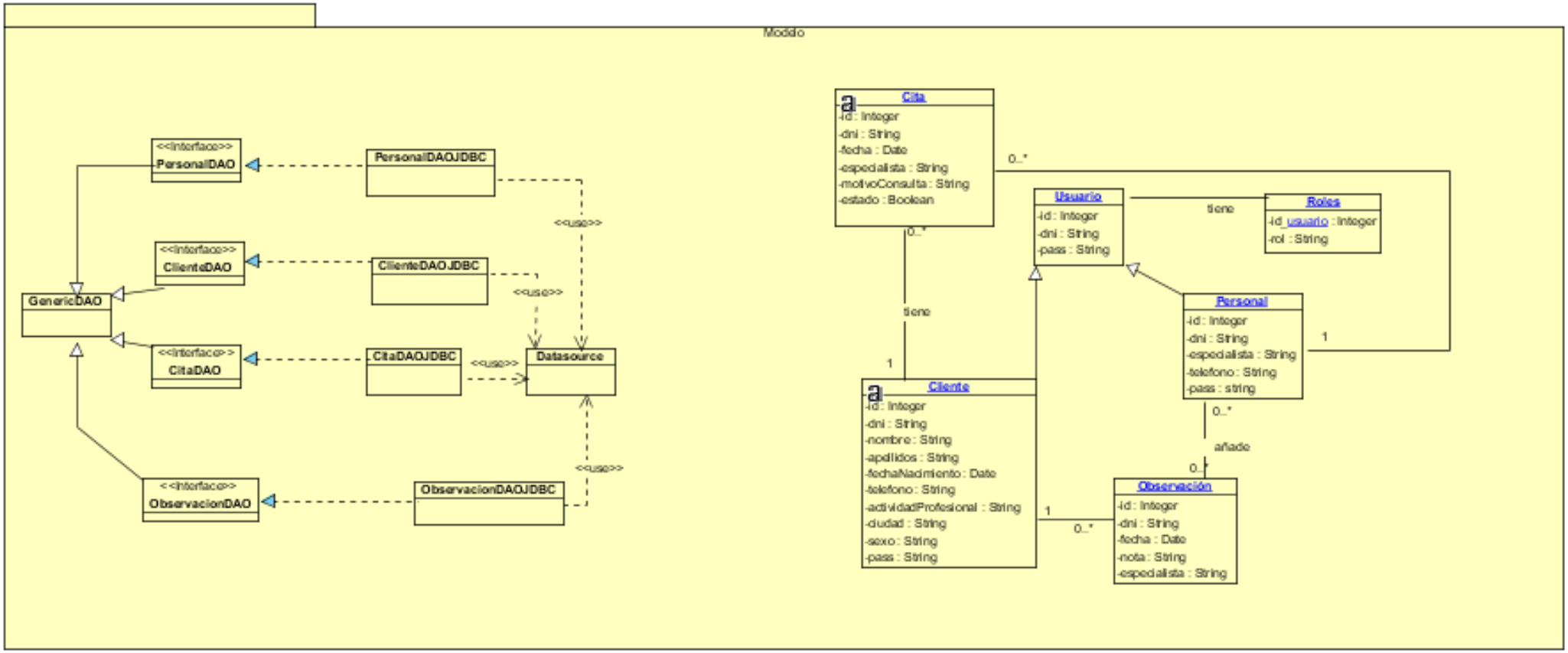


Ilustración 14. Modelo, diagrama de clases.

En la siguiente ilustración 15, se ve el paquete vistas con las que se mostrarán los resultados al usuario, en ella no se han incluido todas las vistas, aunque se han aportado a modo de ejemplo algunas de las vistas correspondientes al controlador RegistroController encargadas de la creación de citas, de usuarios. El detalle de todas ellas puede consultarse en las fuentes suministradas del proyecto realizado con Visual Paradigm.

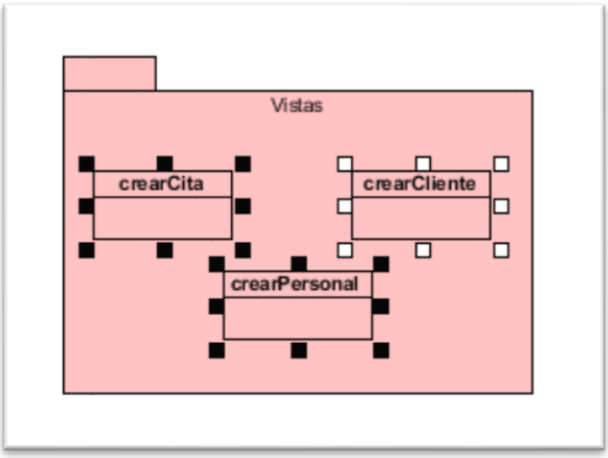


Ilustración 15. Vistas, diagrama de clases.

La parte de los controladores lo mostraremos antes de mostrar el esquema general de nuestro diagrama de clases final en la ilustración 16. Este paquete controlador está compuesto por los 5 controladores que forman nuestro sistema.

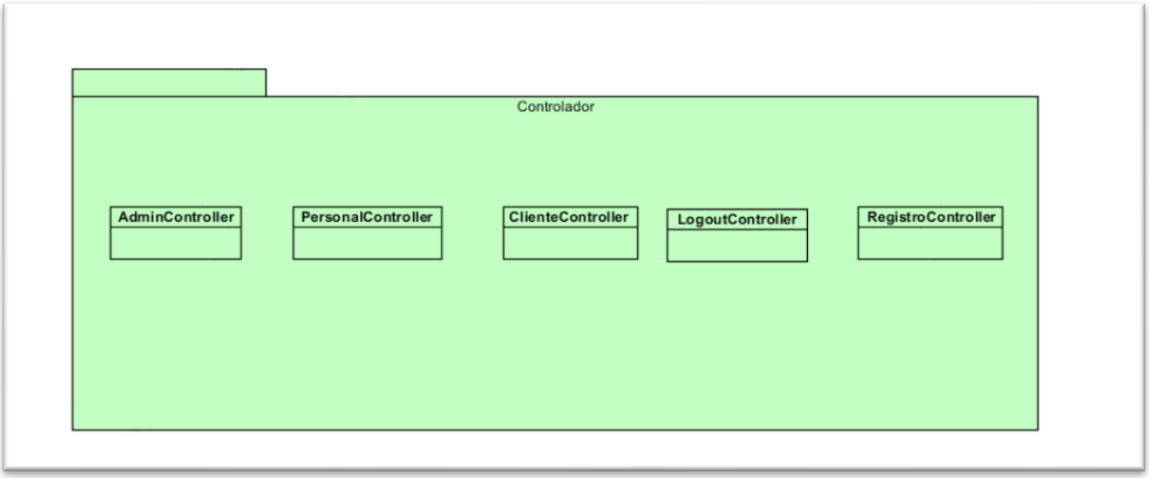


Ilustración 16. Controlador, diagrama de clases.

Y por último tenemos como hemos dicho el diagrama de clases final con todas sus relaciones, utilizando diferentes colores para que se pueda distinguir claramente los tres paquetes, ilustración 17.

Un ejemplo sería el controlador RegistroController, el cual dispone de los DAOs, PersonalDAO, CitaDAO, ClienteDAO y ObservacionDAO, principalmente para la creación. Éste los utilizará para intercambiar datos entre las vistas, Cita, Observacion, Personal, Cliente.

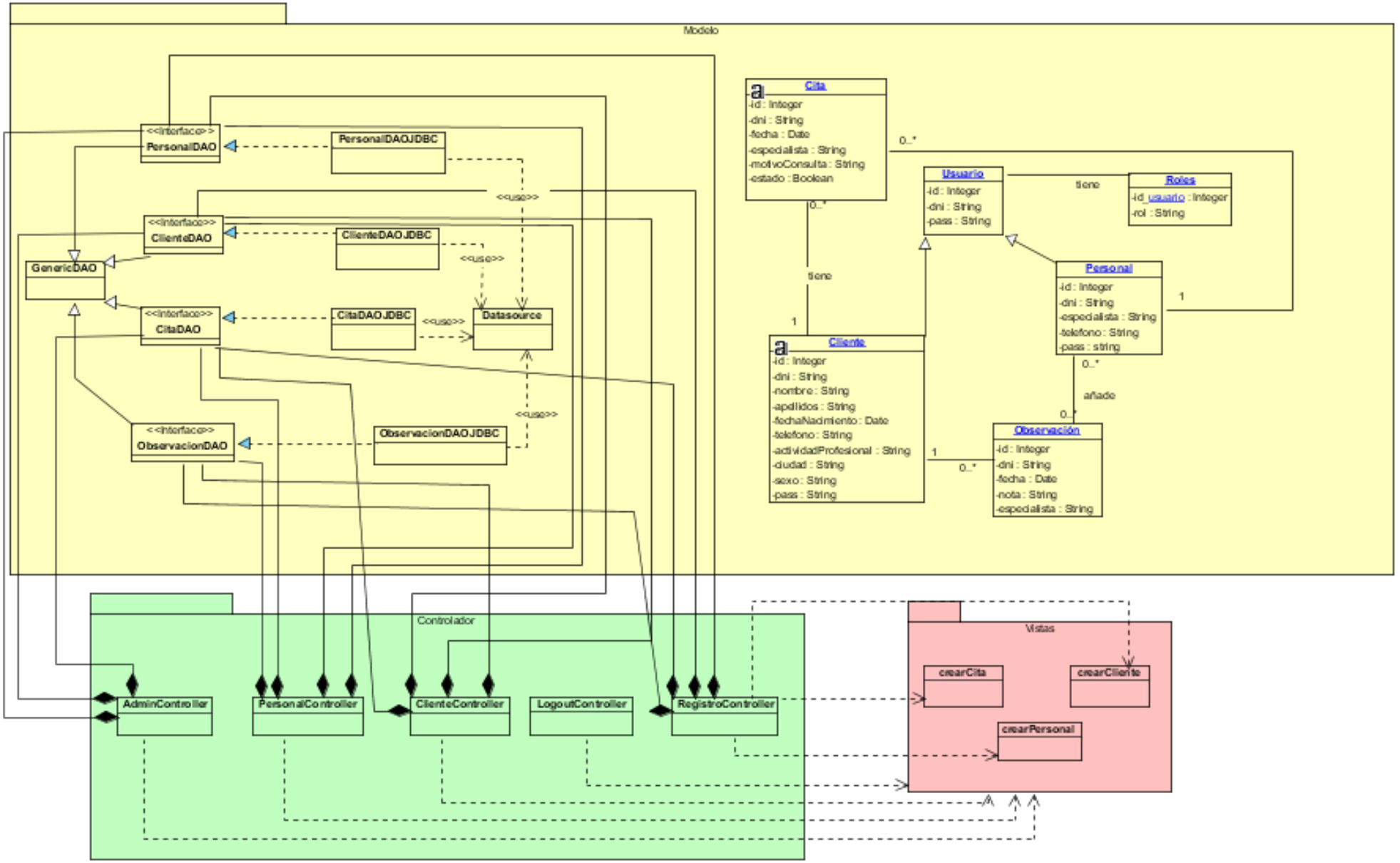


Ilustración 17.Diagrama de clases.

3.3. Diagrama de secuencia

En este apartado mostraremos los diagramas de secuencia que son un tipo de diagrama utilizado para la interacción de una serie de objetos a través de un tiempo. Se realizarán de los apartados más representativos de la aplicación, utilizando el patrón MVC. Utilizaremos colores para representar los elementos de la vista (azul claro), controladores (verde claro) y modelo (naranja).

3.3.1. Diagrama de secuencia Iniciar sesión

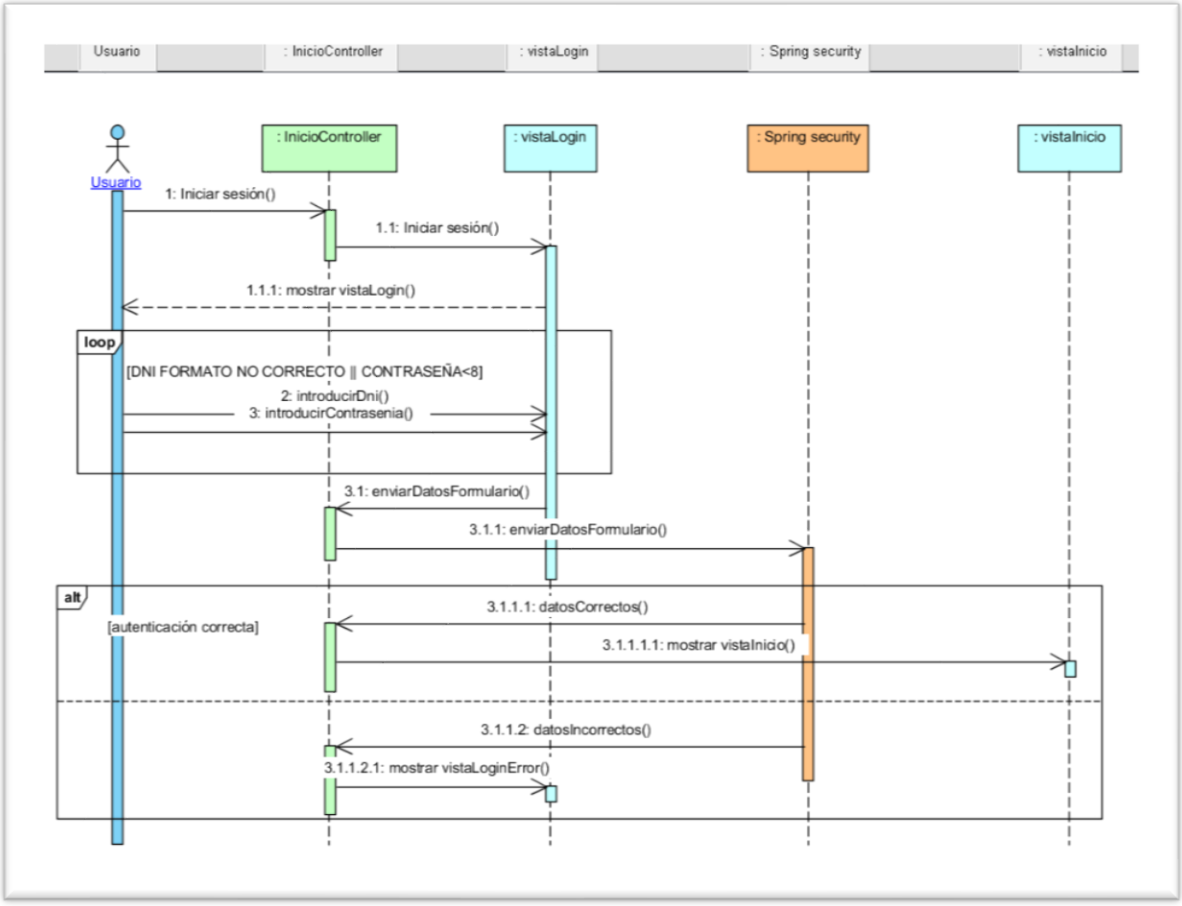


Ilustración 18. Diagrama secuencia iniciar sesión.

En la ilustración 18 podemos ver un diagrama en el que tendremos las vistas Login.jsp e Inicio.jsp, el controlador de inicio y autenticación Spring Security. Primero el usuario, que puede ser cliente, personal o administrador introducirá sus credenciales como son su número de identificación personal y su contraseña personal, los cuales deben ser válidos antes de mandar el formulario. Para finalizar, solamente dará un error si el usuario no está registrado en el sistema o si la contraseña no coincide con los datos guardados en la base de datos, si no es así el usuario entrará a su página de inicio personal.

3.3.2. Diagrama de secuencia añadir cita

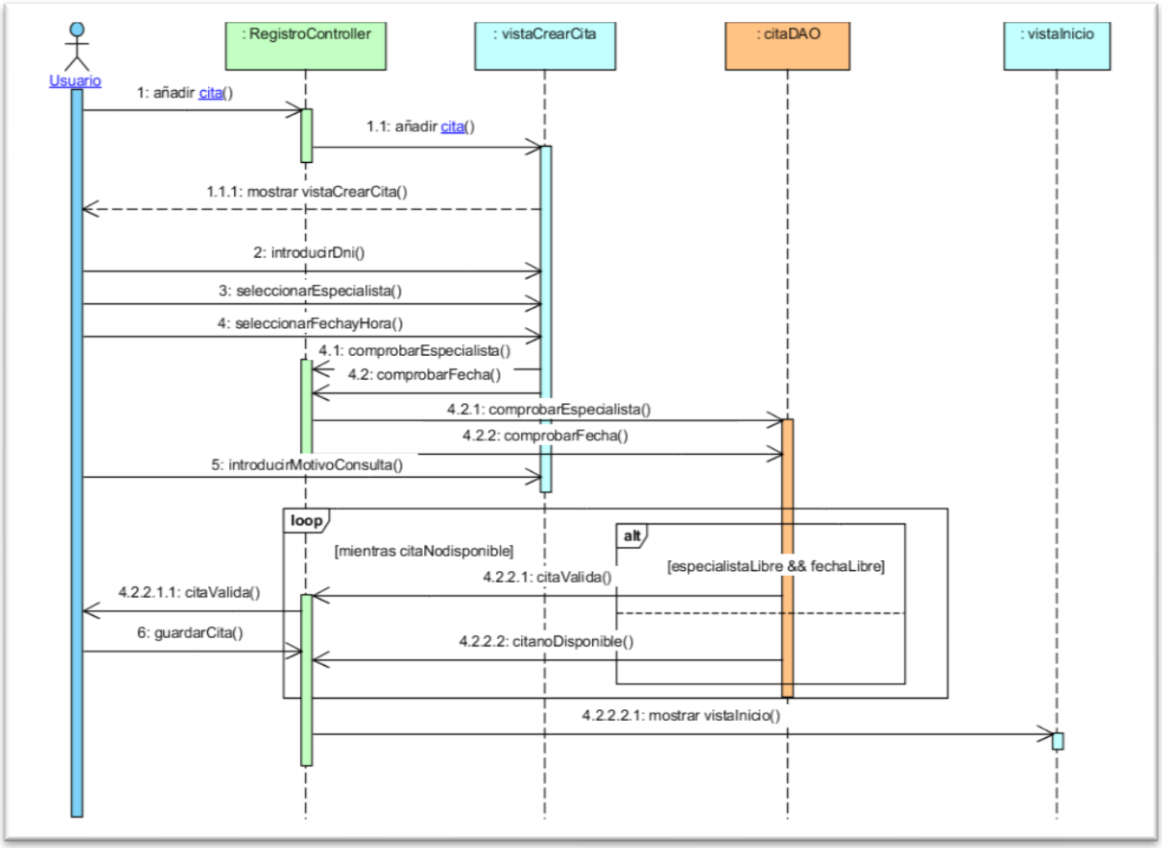


Ilustración 19. Diagrama de secuencia Añadir cita.

En la ilustración 19 trata la funcionalidad de añadir cita, lo puede realizar tanto el cliente como el administrador, por ello se tiene el actor “Usuario” aunque sea para la misma función.

El usuario tiene que rellenar los datos del formulario de crear cita, primero introduciendo su número de identificación personal, posteriormente

seleccionando el especialista que desea escoger, eligiendo la fecha y hora, y por último introduciendo el motivo de la consulta.

Cuando se introduce la fecha y el especialista, se valida si el especialista tiene libre esa cita en su agenda mostrando en rojo cuando no lo está, esperando hasta que el usuario ponga una correcta.

3.4. Diseño de la Interfaz

En este último apartado de la parte de diseño vamos a realizar una serie de bocetos de la aplicación, la opción elegida para ello es ***Axure RP***[9], se trata de una herramienta sencilla para el prototipado.

Primero para indicar cómo se pasan de unas vistas a otras tenemos un diagrama de estados, en el solo se verán las vistas del lado del administrador, ilustración 20.

[9] https://www.axure.com/

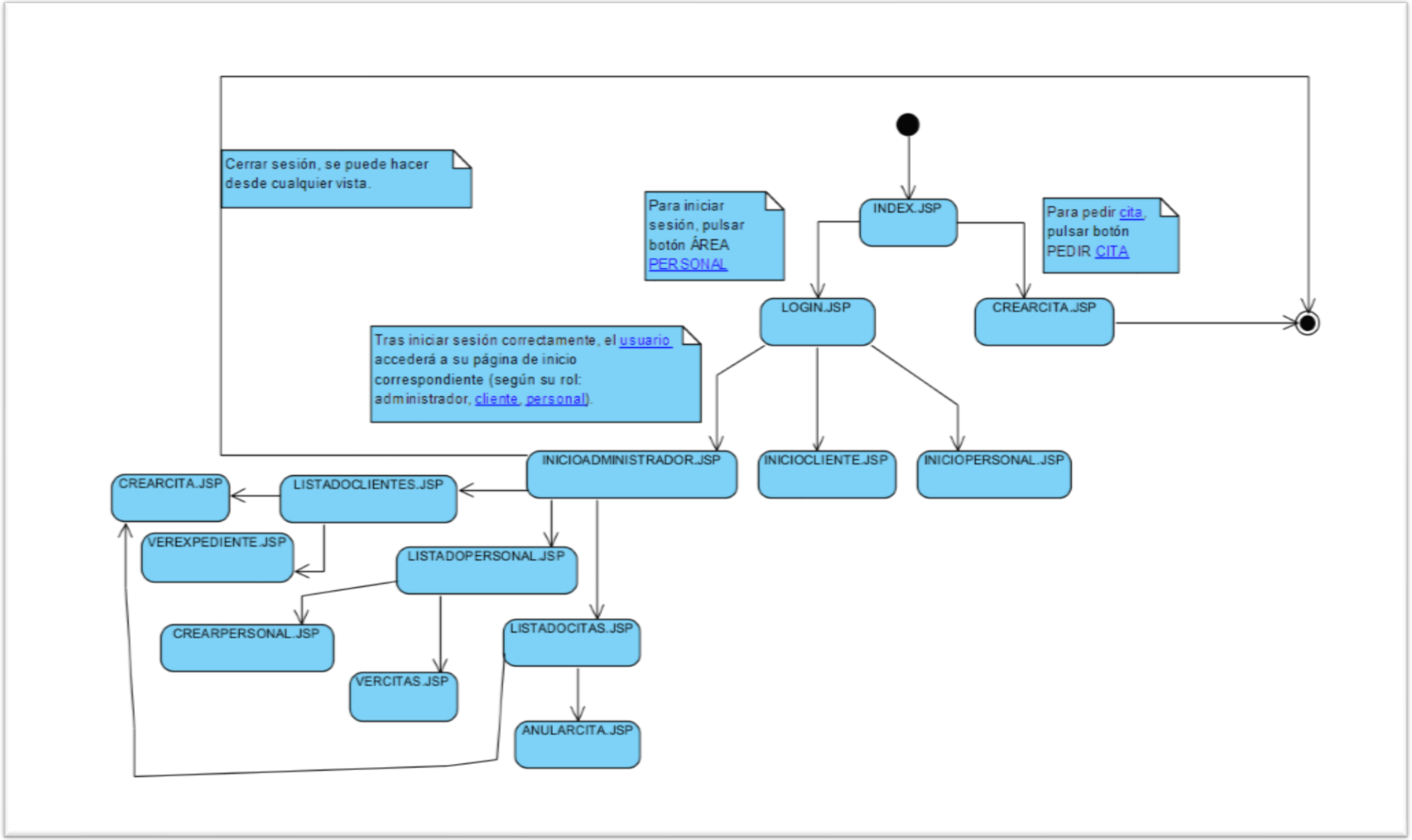


Ilustración 20. Diagrama de estados.

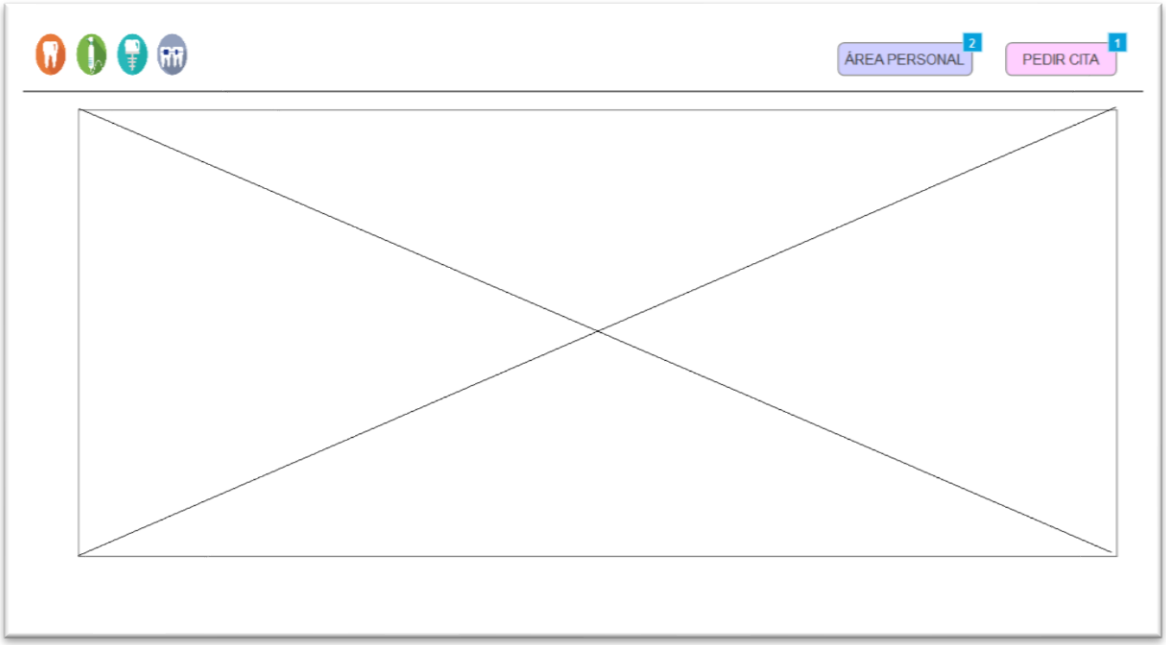


Ilustración 21. Boceto Página inicio

En la ilustración 21, se ve la interfaz de la página de inicio, en ella hay una serie de objetos, como pueden ser los dos botones principales, área personal y pedir cita en la parte superior derecha de nuestra interfaz. En el botón área personal, el usuario es redirigido a la página de iniciar sesión que veremos en la siguiente ilustración.

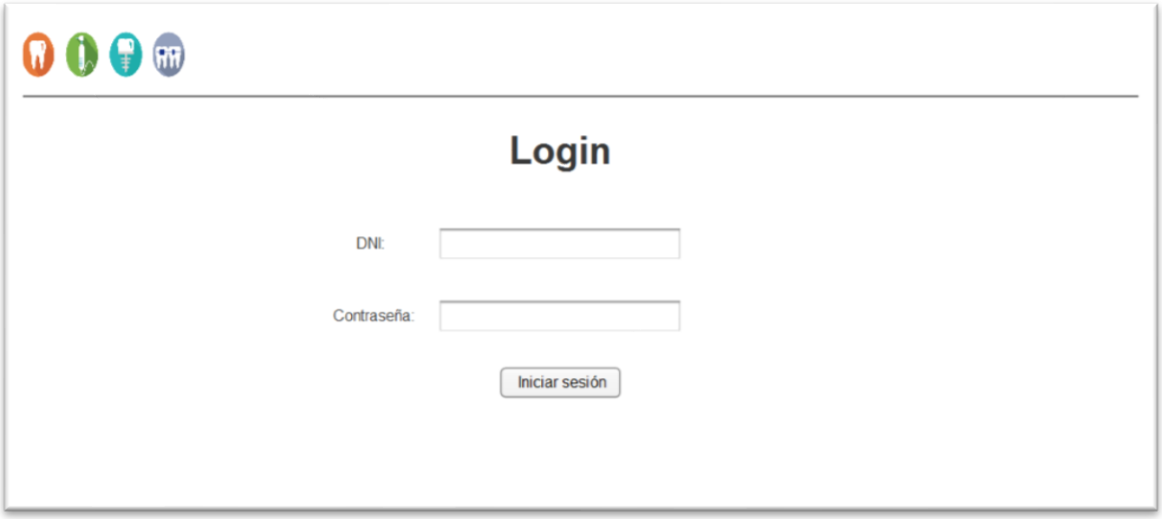


Ilustración 22. Boceto Iniciar sesión

Ahora estamos situados en la ilustración 22 donde el usuario tiene que acceder mediante su número de identificación personal y su contraseña.

Ilustración 23. Boceto pedir cita

El siguiente boceto es pedir cita, como podemos ver en la ilustración 23, en la que el usuario puede rellenar con una serie de datos como son su dni, fecha y hora a elección, el especialista que se prefiera y el motivo de su consulta.

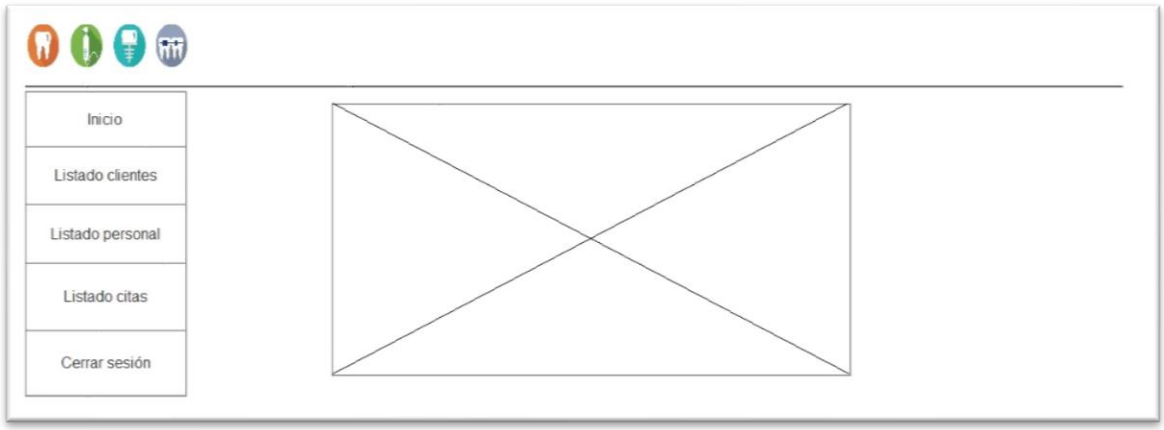


Ilustración 24. Boceto inicio administrador

Tras iniciar sesión, cada usuario verá su página de inicio dependiendo del rol que tenga, en la ilustración 24 sería el menú principal del administrador. En este boceto se puede observar la serie de datos a las que tiene acceso el administrador como son el listado de clientes, personal y citas, en posteriores bocetos veremos a donde nos llevan estos botones.

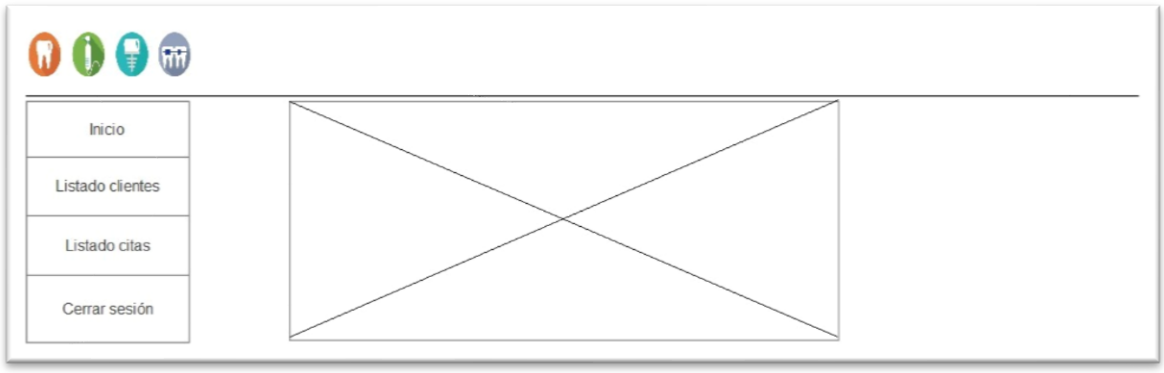


Ilustración 25. Boceto inicio personal.

En esta ilustración 25, se muestra la página de inicio del personal de la clínica, en la que podemos observar que es bastante parecida a la anterior ilustración, pero eliminando el listado de personal.

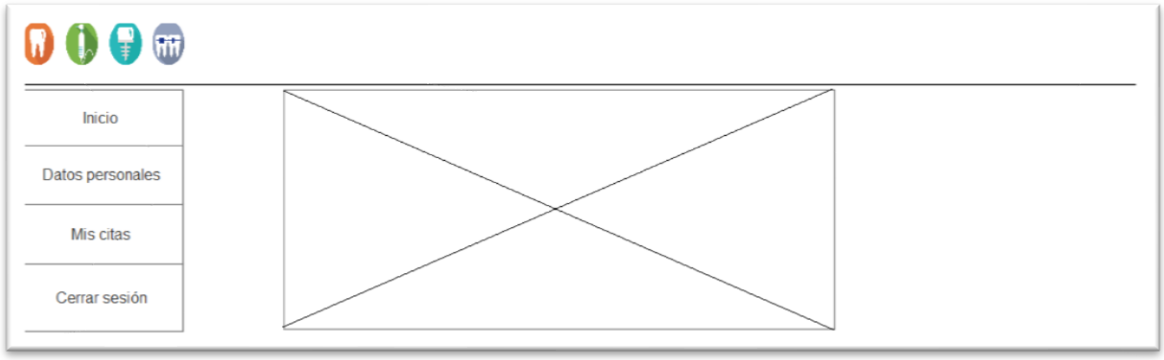


Ilustración 26. Boceto inicio cliente.

Y ahora tenemos la ilustración 26, que nos quedaba por enseñar el inicio de un cliente. También han cambiado algunos objetos como pueden ser los datos personales y mis citas.

Ilustración 27. Boceto listado clientes administrador.

En la ilustración 27, se puede observar un prototipo de la vista listado de clientes en la parte del administrador, en ella salen datos personales como Nombre y Apellidos, DNI, sexo y fecha de nacimiento, también se puede ver el expediente completo pulsando en el botón ver expediente y añadir una nueva cita para el cliente que se nos redirigirá a la ilustración 23, explicada anteriormente.
En la parte del personal de la clínica como hemos dicho en bocetos anteriores también se puede ver el listado de los clientes, pero éste tiene otras botones en la parte de opciones de la vista, en la siguiente ilustración 28, lo veremos.

Ilustración 28. Boceto listado clientes personal.

En esta ilustración 27, vemos como diferencia con respecto a la ilustración 26, la opción ver observaciones y el botón nuevo cliente. Solo el personal de la clínica, el dentista, en la primera consulta del cliente tomará sus datos y lo creará como nuevo cliente.

En la siguiente ilustración 29, tenemos el listado personal del lado del administrador, es una vista similar a los listados anteriores, donde tenemos el nombre y apellidos del especialista, su DNI y tenemos la opción de ver sus citas. También únicamente el administrador tendrá la opción de añadir personal a la base de datos.

Ilustración 29. Boceto listado personal administrador.

Y por último veremos el boceto de listado de citas del lado del administrador, la ilustración 30, donde aparecerá el DNI del cliente que tiene esa cita, su fecha y hora, el motivo de la consulta, el especialista que seleccionó para que le atendiera y por último la opción de anular esa cita, esta opción la tiene únicamente el administrador y el cliente.

Ilustración 30. Boceto listado citas administrador.

4. IMPLEMENTACIÓN

En este capítulo se hablará de los detalles más importantes de la implementación y uso de las tecnologías empleadas, además de ver los diagramas arquitectónicos de la aplicación. Estos apartados nos ayudarán a entender en profundidad cómo funciona nuestra aplicación.

4.1. Arquitectura

Explicaremos nuestra arquitectura a través de ilustraciones, primero en la ilustración 31 se muestra un diagrama arquitectónico de las tecnologías y lenguajes que hemos elegido para esta aplicación, como indicamos en la fase de análisis en la que estuvimos comentando los fundamentos de cada uno de estos *frameworks* y bibliotecas, aunque en este apartado se entrará en detalle los aspectos de uso en el contexto del código de la aplicación, diferenciándolos entre cliente y servidor.

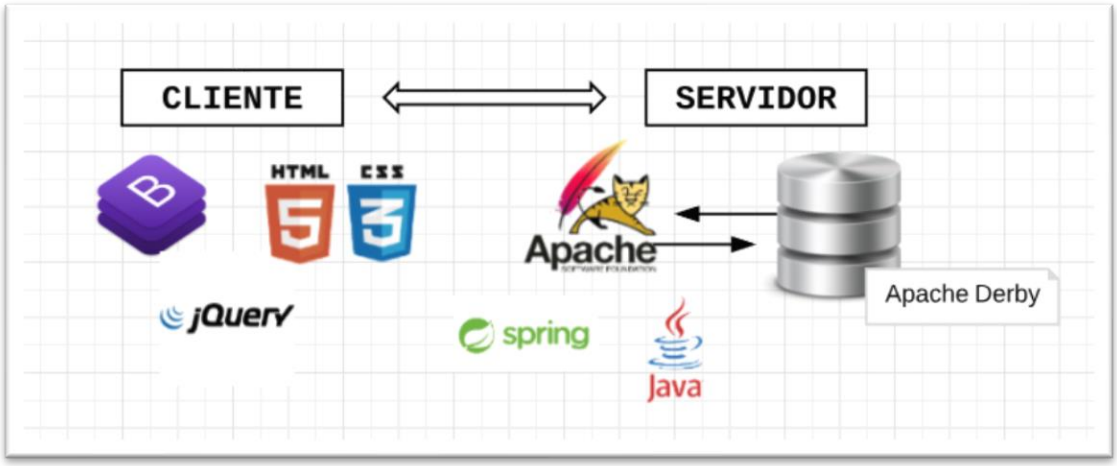


Ilustración 31. Diagrama arquitectónico de tecnologías usadas.

Viendo la ilustración 31 podemos observar que usaremos en la parte del servidor Spring Framework, centrándonos en Spring MVC y Spring Security, para la base de datos usaremos Apache Derby y como servidor Apache Tomcat, por no olvidarnos del lenguaje de programación Java.

En la parte del cliente usaremos un *framework* como Bootstrap para un buen diseño HTML5 y CSS, y JQuery para facilitar algunas funcionalidades JQuery,

como han sido para crear un calendario con fecha y hora en pedir cita, con el uso de un *datetimepicker* de JQuery.

En segundo lugar, veremos en la siguiente ilustración 32 otro diagrama, pero en este caso de la arquitectura MVC**:**

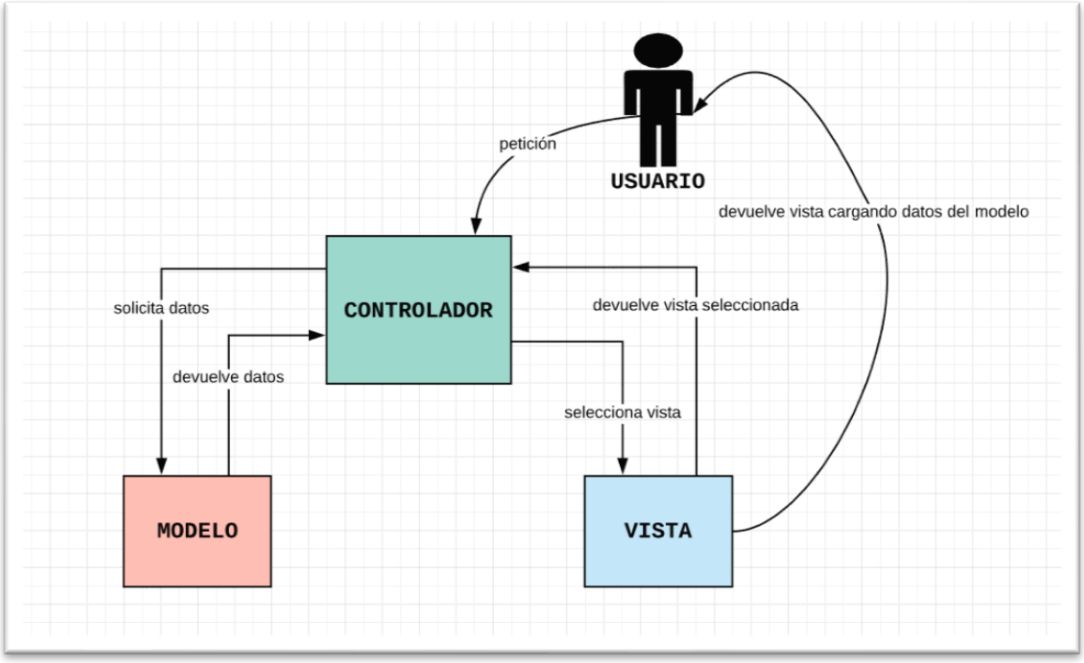


Ilustración 32. Diagrama arquitectónico MVC.

Este patrón, como hemos nombrado en ocasiones anteriores, es el que se utiliza para nuestro *framework* elegido, Spring MVC [9], y organiza el código de la aplicación en tres grandes grupos: el controlador, el modelo y las vistas. Primero el usuario hace una petición al controlador, éste solicita los datos al modelo y el modelo se los devuelve, posteriormente, el controlador selecciona la vista y devuelve la vista seleccionada, el usuario puede ver los datos a través de la vista.

4.2. Detalles sobre implementación

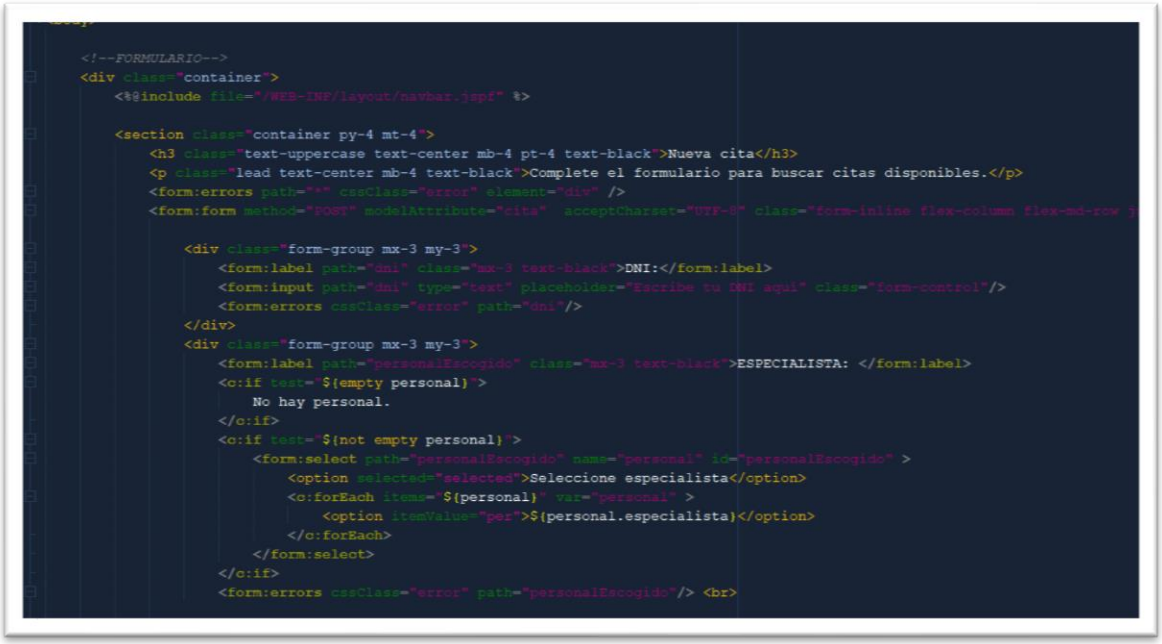
En este último apartado del capítulo 4 trataremos sobre los detalles más relevantes de la parte de implementación de nuestra aplicación entrando en profundidad en algunos casos.

Bootstrap

Primero, en el lado del cliente, tal y como hemos nombrado anteriormente, las vistas destacan por el uso de Bootstrap 4. Éste nos ha facilitado hacer una interfaz elegante a la vista con sus múltiples funcionalidades, para utilizarlo se puede realizar de varias maneras. Vamos a hablar de dos de ellas, en el primer caso se descargan los archivos JS y CSS, los añadimos a nuestro proyecto e incluimos la carpeta donde se haya guardado, en el head de cada vista, esta opción garantiza que tengamos los archivos siempre disponibles. En el segundo caso, se puede incluir el enlace directamente en nuestras vistas, esta última se puede utilizar únicamente cuando tengamos internet, esta evita consumir ancho de banda del servidor, veremos un ejemplo en la siguiente ilustración 33.

Ilustración 33. Ejemplo de incluir Bootstrap.

Un ejemplo de uso de Bootstrap podemos verlo en la ilustración 34, en un ejemplo de código, en la que aparece el sistema de *grid* de Bootstrap, es decir, la disposición de los elementos en nuestra pantalla.



```
<!--FORMULARIO-->
<div class="container">
    <%@include file="/WEB-INF/layout/navbar.jspf" %>

    <section class="container py-4 mt-4">
        <h3 class="text-uppercase text-center mb-4 pt-4 text-black">Nueva cita</h3>
        <p class="lead text-center mb-4 text-black">Complete el formulario para buscar citas disponibles.</p>
        <form:errors path="*" cssClass="error" element="div" />
        <form:form method="POST" modelAttribute="cita"  acceptCharset="UTF-8" class="form-inline flex-column flex-md-row

            <div class="form-group mx-3 my-3">
                <form:label path="dni" class="mx-3 text-black">DNI:</form:label>
                <form:input path="dni" type="text" placeholder="Escribe tu DNI aquí" class="form-control"/>
                <form:errors cssClass="error" path="dni"/>
            </div>
            <div class="form-group mx-3 my-3">
                <form:label path="personalEscogido" class="mx-3 text-black">ESPECIALISTA: </form:label>
                <c:if test="${empty personal}">
                    No hay personal.
                </c:if>
                <c:if test="${not empty personal}">
                    <form:select path="personalEscogido" name="personal" id="personalEscogido" >
                        <option selected="selected">Seleccione especialista</option>
                        <c:forEach items="${personal}" var="personal" >
                            <option itemValue="per">${personal.especialista}</option>
                        </c:forEach>
                    </form:select>
                </c:if>
                <form:errors cssClass="error" path="personalEscogido"/> <br>
```

Ilustración 34. Sistema grid Bootstrap.

Con JQuery ocurre igual que con Bootstrap, se puede hacer de la misma manera a la hora de incluir las dependencias.

En este caso, JQuery, como hemos comentado en el apartado anterior, lo hemos usado para un calendario de fecha y hora (*Datetimepicker*) para que el usuario pueda escoger la cita que quiera. En la ilustración 35, podemos ver cómo está implementado este *datetimepicker* con JQuery.

```
$.datetimepicker.setLocale('es');
$('#fecha').datetimepicker({
    disabledWeekDays: [0],
    allowTimes: ['10:00', '10:15', '10:30', '10:45',
        '11:00', '11:15', '11:30', '11:45',
        '12:00', '12:15', '12:30', '12:45',
        '13:00', '13:15', '13:30', '13:45',
        '17:00', '17:15', '17:30', '17:45',
        '18:00', '18:15', '18:30', '18:45',
        '19:00', '19:15', '19:30', '19:45',
        '20:00', '20:15', '20:30', '20:45'],
    minDate: 0,
    step: 15,
    onShow: festivos,
    onChangeDateTime: function (dp, $input) {
        $('.fecha').html($input.val());
        console.log($input.val());

        var obj_citas = JSON.parse(citas);
        for (var i=0; i < obj_citas.citas.length; i++) {
            // Cargar fecha y adaptar al formato de DatePicker.
            cita_fecha = obj_citas.citas[i].fecha;
            cita_fecha = cita_fecha.substring(0, cita_fecha.length - 5);
            cita_fecha = cita_fecha.replace(/-/g , "/");

            if ($input.val() === cita_fecha && valorEspecialista === obj_citas.citas[i].especialista) {

                jQuery("#fecha").css("border", "3px solid red");
                jQuery(".msgerror1").css("visibility", "visible");
            }else{
                jQuery("#fecha").css("border", "1px solid ");
                jQuery(".msgerror1").css("visibility", "hidden");
            }
```

Ilustración 35. Datetimepicker JQuery.

Spring MVC

Respecto a las tecnologías empleadas en el lado del servidor, empezaremos hablando de Spring MVC [9]. Se trata de una parte del *framework* Spring, basado en el uso del patrón MVC. Este modelo, además, se basa en un controlador que gestiona todas peticiones, llamado *DispatcherServlet*. Veremos un ejemplo en la ilustración 36, del registro del controlador frontal de Spring utilizando una inicialización programática en Java, que sería el equivalente a la versión declarativa clásica en el fichero web.xml. Esta clase es detectada automáticamente por el contenedor del servlet.

```java
public class DentalWebApplicationInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    public void onStartup(ServletContext container) {

        // Create the 'root' Spring application context
        AnnotationConfigWebApplicationContext rootContext
                = new AnnotationConfigWebApplicationContext();
        rootContext.register(SpringMvcConfig.class);

        // Manage the lifecycle of the root application context
        container.addListener(new ContextLoaderListener(rootContext));

        // Register and map the Spring dispatcher servlet
        ServletRegistration.Dynamic dispatcher
                = container.addServlet("springMvcFrontDispatcher", new DispatcherServlet(rootContext));
        dispatcher.setLoadOnStartup(1);
        dispatcher.addMapping("/main/*");

        // UTF-8 request param fix
        FilterRegistration.Dynamic fr = container.addFilter("encodingFilter",
                new CharacterEncodingFilter());
        fr.setInitParameter("encoding", "UTF-8");
        fr.setInitParameter("forceEncoding", "true");
        fr.addMappingForUrlPatterns(null, true, "/*");


    }
```

Ilustración 36. Ejemplo de configuración para registro e inicialización de Dispatcher Servlet.

DispatcherServletApplicationInitializer[10] es una interfaz proporcionada por Spring MVC que garantiza que su configuración en código sea detectada y automáticamente usada para inicializar cualquier *Servlet*.

Veremos en la siguiente ilustración 37 la forma de crear el *DataSource* que usaremos para los DAOs y Spring Security, estará ubicado en el fichero SpringMvcConfig.

```java
@Bean(name="dataSource")
   public DataSource dataSource(){
      DriverManagerDataSource dataSource = new DriverManagerDataSource();
      dataSource.setDriverClassName("org.apache.derby.jdbc.ClientDriver");
      dataSource.setUrl("jdbc:derby://localhost:1527/dental");
      dataSource.setUsername( "root" );
      dataSource.setPassword( "root" );
      return dataSource;
   }
```

Ilustración 37. Creación DataSource

[10] https://bit.ly/2JU1qa5

A continuación, hablaremos de los controladores[11] que interpretan la entrada del usuario y la transforman en un modelo que la vista representa al usuario. Spring MVC proporciona el modelo de programación basado en anotaciones para los controladores de MVC que utiliza anotaciones como *@RequestMapping*, *@RequestParam*, *@ModelAttribute*, etc. En la ilustración 38 se indicará lo comentado con un ejemplo de código.

```
@Controller
@RequestMapping("/registro")

public class RegistroController {

    @Autowired
    @Qualifier("clienteDAOJDBCBean")
    ClienteDAOJDBC clienteDAO;

    @Autowired
    @Qualifier("citaDAOJDBCBean")
    CitaDAOJDBC citaDAO;

    @Autowired
    @Qualifier("personalDAOJDBCBean")
    PersonalDAOJDBC personalDAO;
```

Ilustración 38. Ejemplo un controlador.

@Controller nos indica que esta clase con la función de controlador y *@RequestMapping*, esta anotación nos permite saber que URL está asignada a este controlador. No debemos olvidarnos de los DAO y DAOJDBC, para llamarlos debemos usar las anotaciones *@Autowired* y *@Qualifier* para la inyección de dependencias, ilustración 38.

[11] https://bit.ly/2JU1qa5

A continuación, un ejemplo de código de dos métodos GET y POST para visualizar cómo se generan y procesan los datos, ilustración 39.

```
@RequestMapping(value = "/crearCita", method = RequestMethod.GET)
public String crearCita(ModelMap model) {

    List<Personal> personalList = personalDAO.searchAll();

    model.addAttribute("personal", personalList);
    model.addAttribute("cita", new Cita());
    return "cita/crearCita";
}

@RequestMapping(value = "/crearCita", method = RequestMethod.POST)
public String creaCita(
        @ModelAttribute("cita") @Valid Cita c,
        BindingResult result, ModelMap model) {
    String view = "redirect:crearCita";
    if (!result.hasErrors()) {
        citaDAO.create(c);
        model.clear();
    } else {
        view = "cita/citaError";
    }
    return view;
}
```

Ilustración 39. Ejemplo GET y POST.

Bean Validation

Bean Validation[12] es una especificación de Java, que nos permite expresar restricciones en modelos de objetos a través de anotaciones y validaciones en formularios.

Las restricciones[13] pueden ser integradas o definidas por el usuario. Las restricciones definidas por el usuario se llaman restricciones personalizadas. Varias restricciones incorporadas están disponibles en el paquete javax.validation.constraints .

[12] https://beanvalidation.org/
[13] https://docs.oracle.com/javaee/6/tutorial/doc/gircz.html

Veremos un ejemplo de código en la ilustración 40, usando algunas de las muchas anotaciones que existen.

```
private int id;
@Pattern(regexp="\\d{7,8}(-?[a-zA-Z])?")
private String dni;

@NotEmpty
@Size(min=3,max=50)
private String nombre;

@NotEmpty
@Size(min=5,max=50)
private String apellidos;

@DateTimeFormat(pattern = "yyyy-MM-dd")
private Date fechaNacimiento;
```

Ilustración 40. Bean Validation.

Spring Security

En lo referente al apartado de seguridad se ha utilizado como hemos nombrado anteriormente un módulo de Spring, Spring Security [10], que se usa para la autenticación y autorización del usuario. En las siguientes ilustraciones observamos la configuración necesaria para que funcione correctamente.

Esta configuración se puede hacer de forma declarativa mediante el uso de archivos XML, o de forma programática utilizando código Java como ha sido en nuestro caso.

En la ilustración 41 vemos cómo en primer lugar se añaden las dependencias necesarias en el pom.xml para el correcto funcionamiento.

```xml
<!--Spring security-->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>4.2.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>4.2.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>4.2.3.RELEASE</version>
</dependency>
```

Ilustración 41. Dependencias en el pom.xml de Spring Security.

Tras esto, se crea una nueva clase llamada SecurityConfig.java, en la que tendremos la conexión a la base de datos en nuestro caso, ya que los usuarios y roles están allí ubicados para la autenticación, no podemos olvidar que hay que añadir las anotaciones siguientes que se observan mejor en la siguiente ilustración 42.

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private DataSource dataSource;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.jdbcAuthentication().dataSource(dataSource)
        .usersByUsernameQuery(
                "SELECT DNI AS USERNAME ,PASS AS PASSWORD ,ENABLED AS ENABLED FROM USUARIOS WHERE DNI=?")
        .authoritiesByUsernameQuery(
                "SELECT USUARIOS.DNI AS USERNAME, ROLES.ROL AS AUTHORITY "
              + "FROM USUARIOS INNER JOIN ROLES ON USUARIOS.ID = ROLES.ID_USUARIO "
              + "WHERE USUARIOS.DNI = ?");
    }
```

Ilustración 42. Autenticación.

En la ilustración 43, tenemos las reglas de control de acceso (autorización), a que *urls* pueden acceder cada usuario dependiendo del rol, estas reglas también están dentro del archivo SecurityConfig.java.

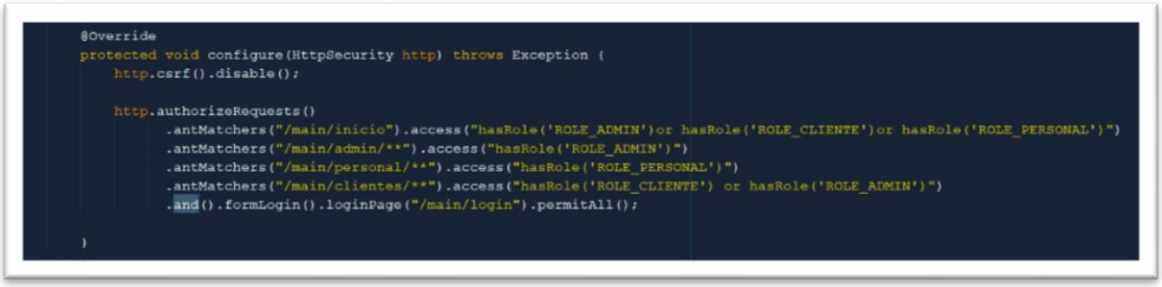


Ilustración 43. Autorización.

Ahora se configura la clase SpringMvcConfig.java, en la que se importará la clase de configuración vista en la ilustración anterior, en esta ocasión nos conectaremos con la base de datos como anteriormente vimos en la parte de configuración de Spring MVC, ilustración 37.

A continuación, crearemos una clase abstracta llamada SpringSecurityInitializer.java, en la ilustración 44, con esto automáticamente se registrará el filtro springSecurityFilterChain para cada URL en nuestra aplicación.

Ilustración 44. Configuración springSecurityFilterChain.

Después editar la clase que creamos al inicio de nuestro proyecto, que es la que se encarga del inicializarlo, DentalWebApplicationInitializer.java, que también se hará abstracta y se añadirán los siguientes métodos. Entre ellos, se encargan de

añadir un ContextLoaderListener que carga SecurityConfig.java, esta clase será la encargada de cargar todo para que funcione, ilustración 45.

```java
@Override
protected Class<?>[] getRootConfigClasses() {
    return new Class[] { SpringMvcConfig.class };
}

@Override
protected Class<?>[] getServletConfigClasses() {
    return null;
}

@Override
protected String[] getServletMappings() {
    return new String[] { "/" };
}
```

Ilustración 45. DentalWebApplicationInitializer.java

Con estos pasos, nuestra tendremos la configuración necesaria para Spring Security autenticación con roles.

Conexión y gestión de la base de datos

En este caso, el gestor de base de datos que hemos decidido elegir es Apache Derby.

Para conectarnos con la base de datos utilizaremos un pool de conexiones, ya que la creación y cierre de una conexión es lento y costoso. Por el contrario, un pool tiene como ventajas que permite reutilizar conexiones con la BD entre diferentes peticiones a la aplicación y no olvidarnos de las credenciales únicas para todas las conexiones.

En el fichero context.xml que se encuentra en el interior de la carpeta META-INF, tal y como se muestra en la siguiente ilustración 46, podemos ver el driver

(driverClassName) que queremos utilizar, la cadena de conexión (name), entre otros datos de información para la configuración del pool de conexiones con tomcat.

```
<Resource auth="Container"
          driverClassName="org.apache.derby.jdbc.ClientDriver"
          url="jdbc:derby://localhost:1527/dental"
          type="javax.sql.DataSource"
          name="jdbc/dental"
          username="root"
          password="root"
          maxTotal="25"
/>
```

Ilustración 46. Context.xml, conexión base de datos.

A continuación, ilustración 47, tendremos el Bean asociado al pool de conexiones, que hablamos de él en ocasiones anteriores para la configuración de Spring Security, este ejemplo de código se situaría en el archivo SpringMvcConfig.java, en el caso de que se usara en lugar de Java config se utilizara XML, se situaría en el applicationContext.xml.

```
@Bean(name="dataSource")
   public DataSource dataSource(){
      DriverManagerDataSource dataSource = new DriverManagerDataSource();
      dataSource.setDriverClassName("org.apache.derby.jdbc.ClientDriver");
      dataSource.setUrl("jdbc:derby://localhost:1527/dental");
      dataSource.setUsername( "root" );
      dataSource.setPassword( "root" );
      return dataSource;
   }
```

Ilustración 47. Ejemplo de Bean en SpringMvcConfig.java

5. PRUEBAS

En este apartado se mostrarán los criterios de aceptación de las historias de usuario del apartado 2.3, estas pruebas se realizaron después de cada Sprint para validar las historias de usuario implementadas.

ID	HISTORIA DE USUARIO	CRITERIOS DE ACEPTACIÓN
1	INICIAR SESIÓN	1. Inicio de sesión exitoso: El usuario accederá a la aplicación poniendo correctamente los datos requeridos. 2. Inicio de sesión fallido: El usuario no introduce correctamente su DNI o contraseña, o ambos casos. En este caso el usuario deberá rellenar de nuevo el formulario de inicio de sesión correctamente. 3. Sesión expiró: El usuario deberá acceder de nuevo al sistema poniendo de nuevo sus credenciales.
2	DAR DE ALTA CLIENTE	1. Creación exitosa: El usuario se crea correctamente al rellenar el formulario. 2. Creación fallida: El usuario no puede crearse, ya que pueden faltar datos necesarios para su creación, rellenar datos incorrectos, o intentar crear usuario ya existente.
5	CAMBIAR CONTRASEÑA	1. Cambio exitoso: El usuario cambia la contraseña correctamente (entre 8 y 20 caracteres). 2. Cambio fallido: El usuario no cumple con el cambio de contraseña.
7	AÑADIR CITA	1. Creación exitosa: El usuario completa correctamente los datos del formulario.

		2. Creación fallida por cita no disponible: El usuario debe rellenar de nuevo el formulario, escogiendo otro especialista ya que puede estar ocupado en esa fecha y hora escogida.
9	AÑADIR OBSERVACIÓN	1. Creación exitosa: El usuario completa correctamente los datos del formulario. 2. Creación fallida: El usuario debe rellenar de nuevo el formulario, ya que los datos proporcionados no cumplen con los requeridos.

Tabla 14. Criterios de aceptación de las historias de usuario.

Tras realizar esta serie de criterios vemos que las funcionalidades de nuestro sistema funcionan correctamente.

6. CONCLUSIONES

Para concluir hemos de decir que hemos cumplido con los objetivos propuestos al inicio del proyecto. Primeramente, se ha hecho un estudio las necesidades y soluciones existentes que hay actualmente en el mercado. Tras esto, se pasó a la hora de la implementación, en la cual hemos utilizado la tecnología Spring MVC y Spring Security, siendo estas de un gran avance con respecto antes del empezar el proyecto respecto a conocimientos adquiridos, ya que he aprendido bastante de esta tecnología. Además de que se ha seguido con la metodología SCRUM tal y como propusimos. Y el patrón MVC, contemplado en todas las partes del proyecto. En la parte de la documentación se ha intentado dejar lo más claro posible como se ha realizado cada parte de este proyecto.

En mi opinión, he de decir que el proyecto ha sido gratificante, ya que he aprendido mucho de las tecnologías y metodologías usadas. Además de que son tecnologías que no pasan de moda y están a la hora del día.

El proyecto ha sido un gran reto, aunque fueran conocimientos conocidos por otras asignaturas a lo largo de la carrera, ya que han surgido algunos errores, pero se han solucionado por la gran motivación que me llevó hacer este TFG, gracias a la ayuda de mi tutor.

Para finalizar me gustaría aprender aun más de este proyecto, e intentar tenerlo en un futuro para la clínica de mi hermana, manteniéndolo y mejorándolo cada vez más.

6.1. MEJORAS Y TRABAJOS FUTUROS

En este último apartado de conclusiones diremos que tras no poder realizar tres de las historias de usuario planeadas en el capítulo 2, apartado de historias de usuario (2.3). Las cuales son las siguientes crear odontograma, visualizar odontograma y editar odontograma.

En un futuro se tendrán en cuenta para poder implementarlas y mejorar la aplicación, ya que son funcionalidades que aportan gran interés para este sector.

Gracias a la arquitectura y tecnologías usadas será de gran facilidad poder añadir en un futuro esas historias de usuario indicadas, además de otras de gran ayuda para este oficio, odontología. Cómo pueden ser notificaciones a dispositivos móviles de las próximas citas a los clientes, seguimiento de expedientes para actuaciones que impliquen a varios profesionales, añadir presupuestos de los clientes, etc.

7. BIBLIOGRAFÍA

[1] *A. Álvarez, R. De las Heras, C. Lasa. Métodos ágiles y Scrum (Manuales imprescindibles).* Anaya multimedia, 2011.

[2] Población que usa Internet (en los últimos tres meses). Tipo de actividades realizadas por Internet, Disponible en: https://bit.ly/1PYwho6

[3] *Revista RCOE (junio 2016) Vol. 21 Suplemento 1. Encuesta poblacional la salud bucodental en España 2015.* Disponible en: https://bit.ly/2v3Tqhv

[4] Crecimiento de búsquedas en Google de la palabra dentista. Disponible en: https://bit.ly/2mKxTqq

[5] Ranking de puntos de diferentes *framework*. Disponible en: https://hotframeworks.com/

[6] *M. Cohn. User stories applied: For Agile Software Development. Addison Wesley, 2004*

[7] *Mike Cohn. Agile Estimating and Planning. Pretince Hall, 2006.*

[8] Convenio telefónica. Disponible en: https://www.boe.es/boe/dias/2016/03/15/pdfs/BOE-A-2016-2624.pdf

[9] Spring Web MVC. Docs.spring.io. https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc

[10] Spring Security. projects.io. http://projects.spring.io/springsecurity/

APÉNDICE I. MANUAL DE USUARIO

En este primer apéndice se mostrará el manual de usuario de esta aplicación.

En primer lugar, tenemos la página de inicio de la página web, ilustración 48. En la que podemos ver una serie de botones en la parte superior derecha, estos botones son los más relevantes de esta página de inicio. En ellos, podemos tanto entrar con nuestro DNI y contraseña, o bien, pedir cita online rellenando un formulario. Estas dos funcionalidades lo veremos después de esta vista.

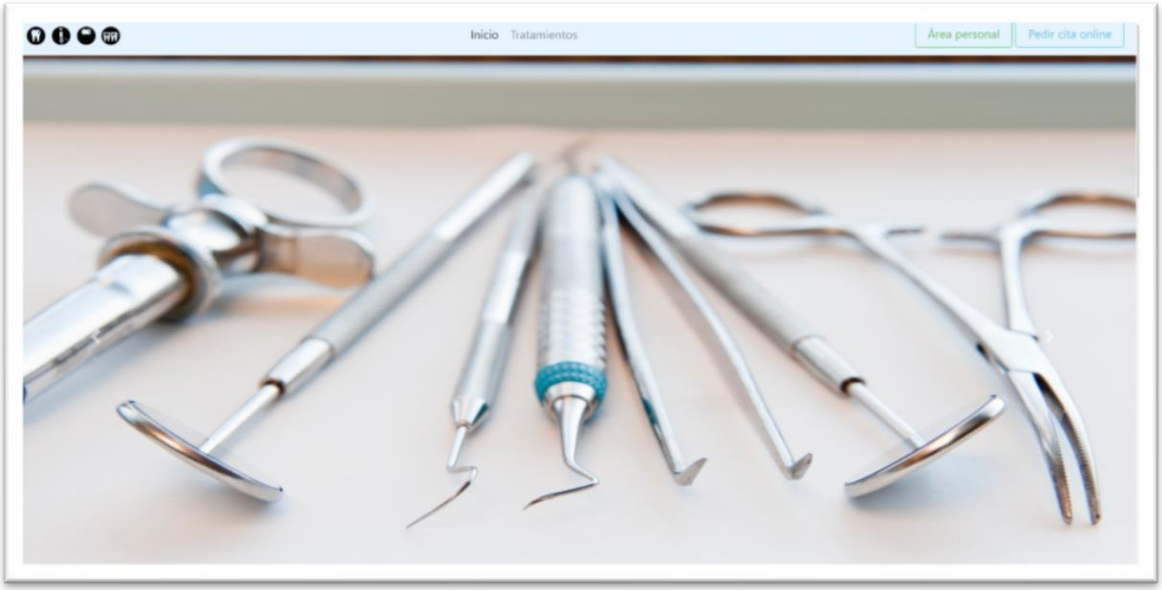


Ilustración 48. Página inicio. Manual de usuario.

En segundo lugar, tenemos la página de logueo, ilustración 49. Para acceder a ella, hemos de darle al botón área personal de la ilustración 48.

En esta vista debemos rellenar los dos campos con nuestro DNI y contraseña (para acceder debe anteriormente añadirlo a la base de datos el personal de la clínica en la primera cita).

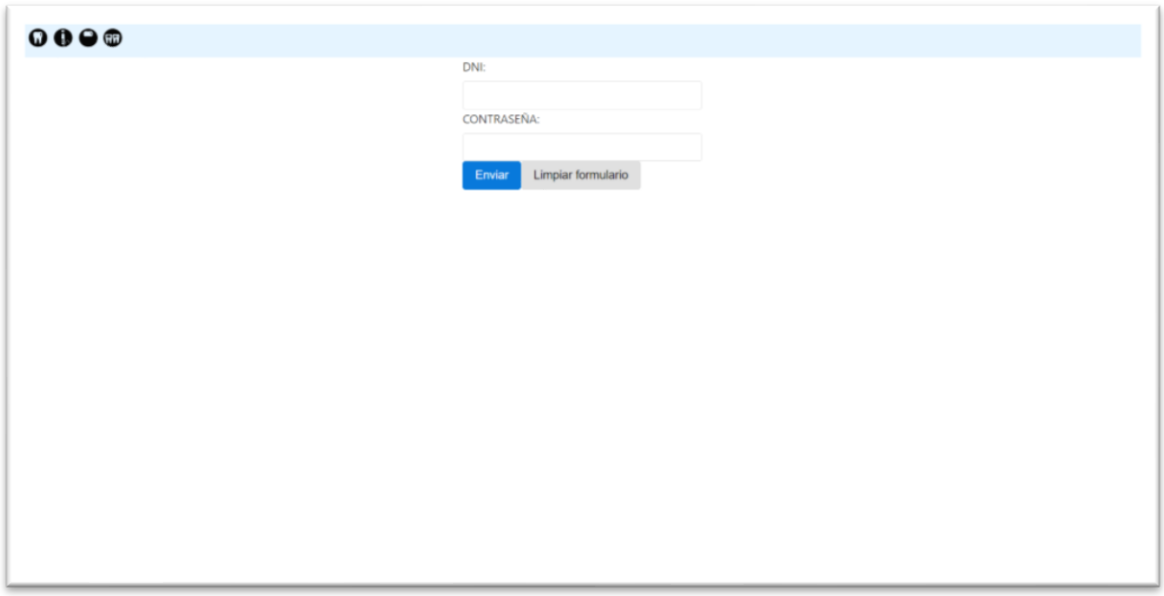


Ilustración 49. Página login. Manual de usuario.

En tercer lugar, como hemos nombrado anteriormente, desde la página de inicio, ilustración 48, se puede acceder a pedir cita online. Esto nos lleva a esta vista que vemos en la ilustración 50.

En ella hay que rellenar un formulario para poder pedir cita y ver si el especialista y la fecha que queremos están libres.

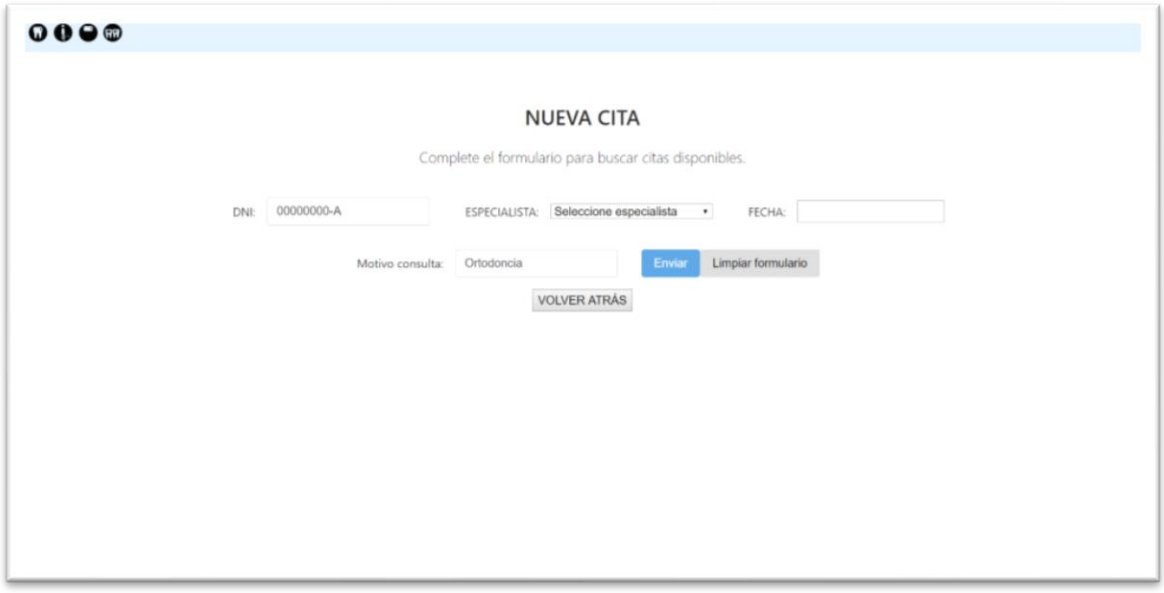


Ilustración 50. Pedir cita online. Manual de usuario.

Tras acceder desde el login, nos encontramos con diferentes inicios dependiendo del rol que tengamos como hemos comentado en ocasiones anteriores.

Panel de usuario con rol cliente de la clínica.

En este caso hemos entrado con el rol de cliente, ilustración 51.

En esta ilustración 51, observamos que tenemos un navbar en la parte izquierda de arriba abajo, con una serie de accesos a distintas opciones, en las siguientes ilustraciones veremos a donde nos lleva cada una.

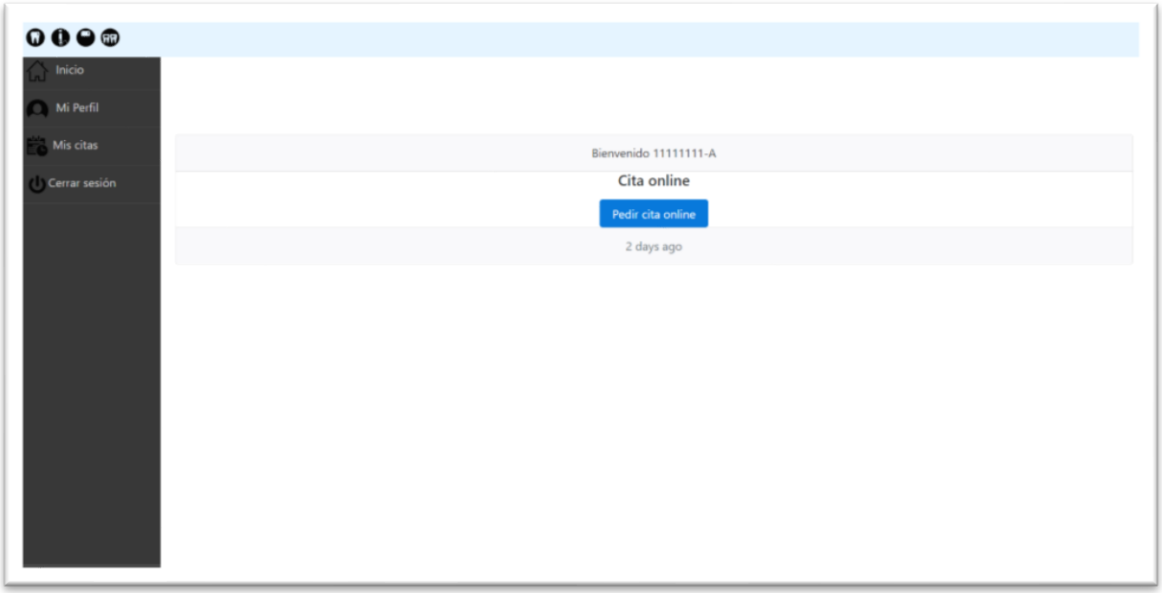


Ilustración 51. Página inicio cliente. Manual de usuario.

En el caso de darle a la opción mi perfil, ilustración 52, obtenemos esta vista en la cual salen los datos personales del usuario y un botón “Cambiar contraseña”.

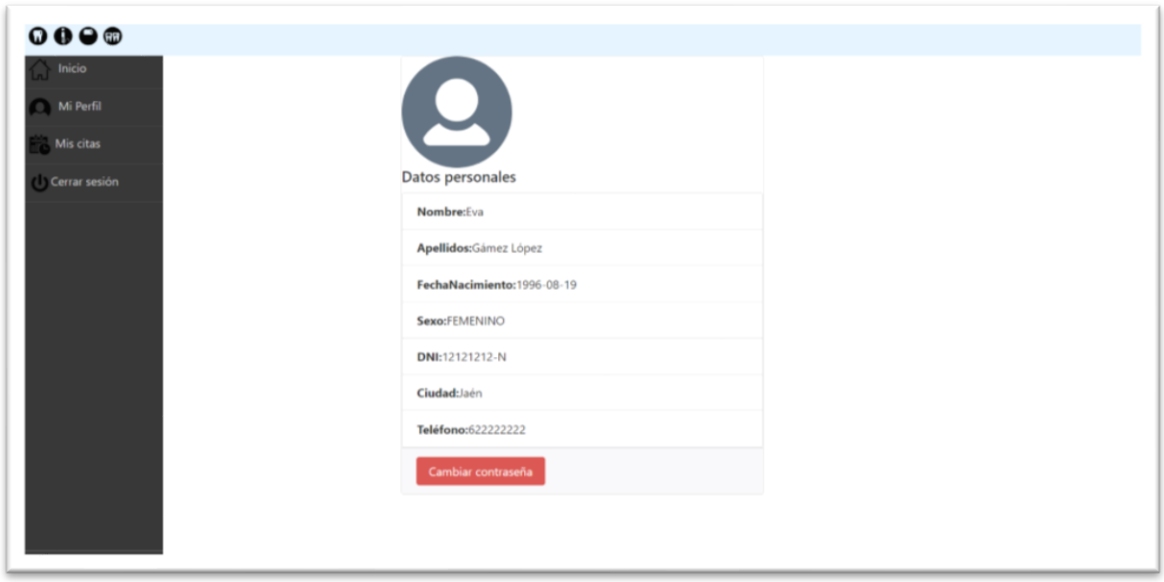


Ilustración 52. Mi perfil. Manual de usuario.

Al darle al botón cambiar contraseña nos aparece la siguiente ilustración 53.

En este caso, podemos cambiar la contraseña, y poner una nueva.

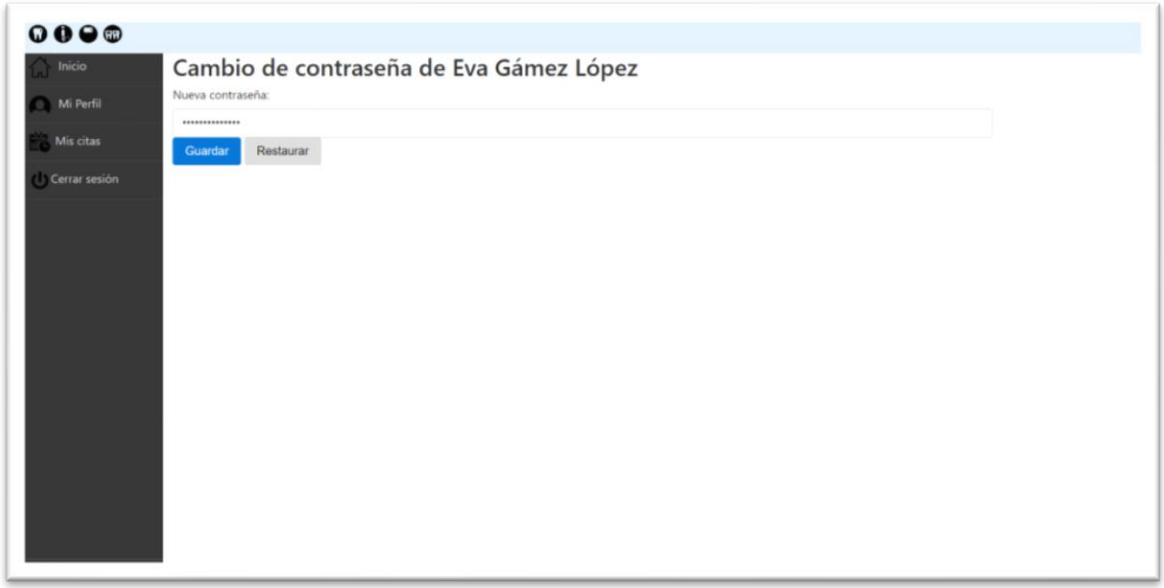


Ilustración 53. Cambiar contraseña. Manual de usuario.

Pasamos a la opción mis citas, en la cual veremos las citas pendientes que tenemos actualmente. Ilustración 54.

Además de ver que citas tenemos, podemos obtener una nueva cita en la parte superior derecha, botón “Nueva cita”, o bien, anular la cita que teníamos como pendiente.

Al pulsar el botón “Nueva cita” obtendremos la ilustración 50 vista anteriormente.

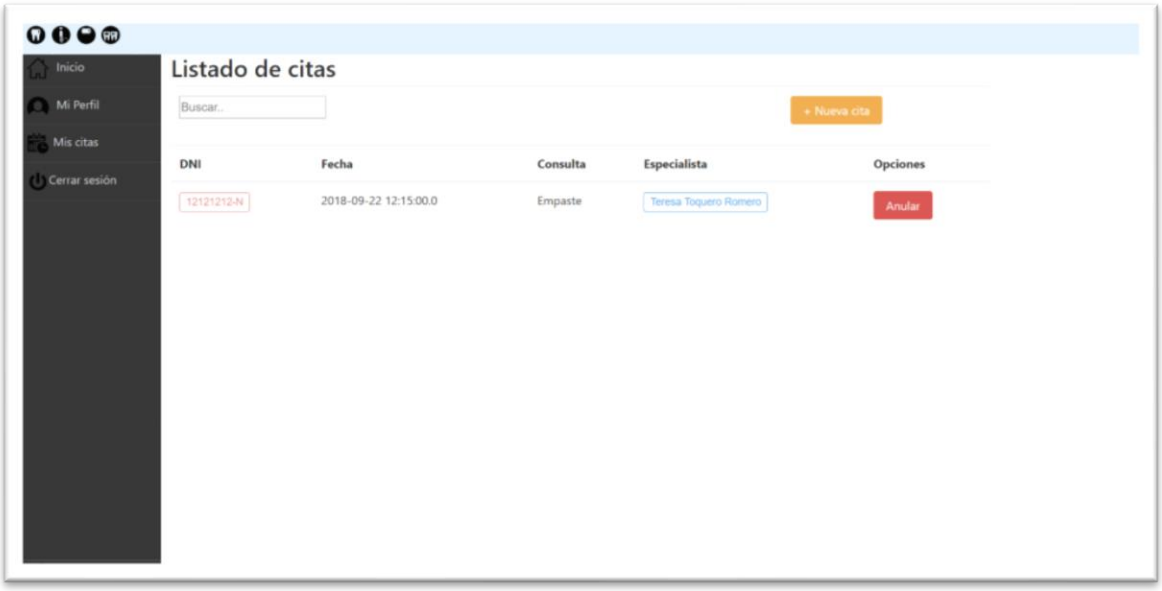


Ilustración 54. Mis citas. Manual de usuario.

Panel de usuario con rol personal de la clínica (dentistas)

En esta primera vista del lado del dentista, ilustración 55, podemos ver el listado de clientes, donde tiene diferentes opciones tales como crear un nuevo cliente (en la primera visita a la clínica, el dentista es el encargado de crear al nuevo usuario). Y el botón observaciones en las cuales se seguirá el proceso de evolución del cliente mediante anotaciones que añadirá el dentista.

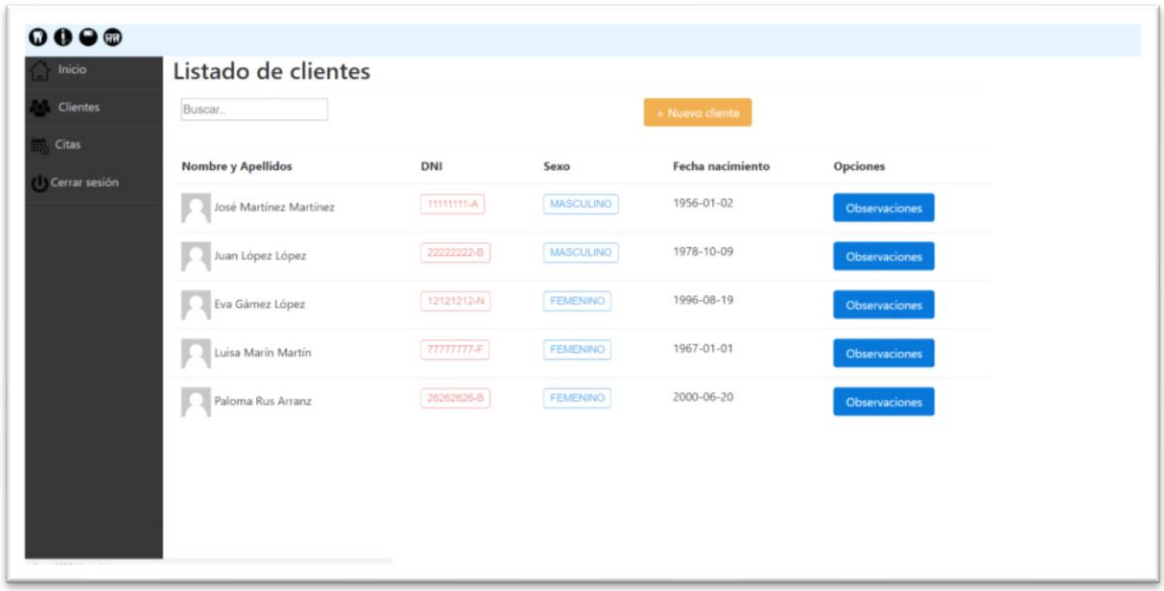


Ilustración 55. Listado clientes. Manual de usuario.

En la siguiente ilustración 56, tenemos las observaciones del cliente, a la que hemos accedido a través de la vista anterior, ilustración 55. En ella hay una serie de datos que tendrá que rellenar el personal para añadirla al cliente.

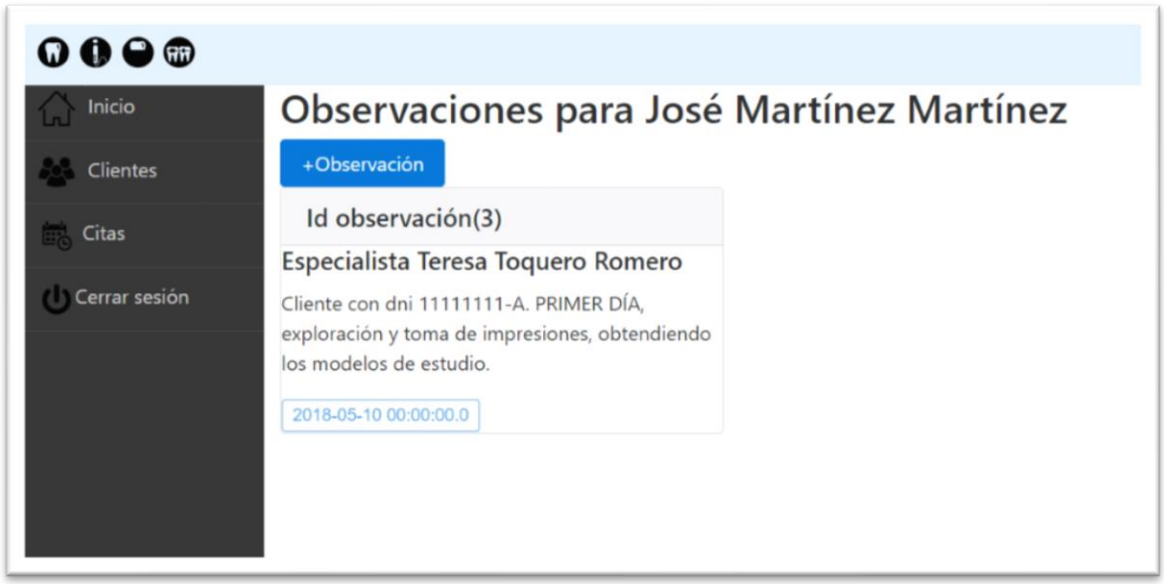


Ilustración 56. Observaciones. Manual de usuario.

Lo siguiente es el listado de citas en la que veremos únicamente las citas pendientes que hay, ilustración 57.

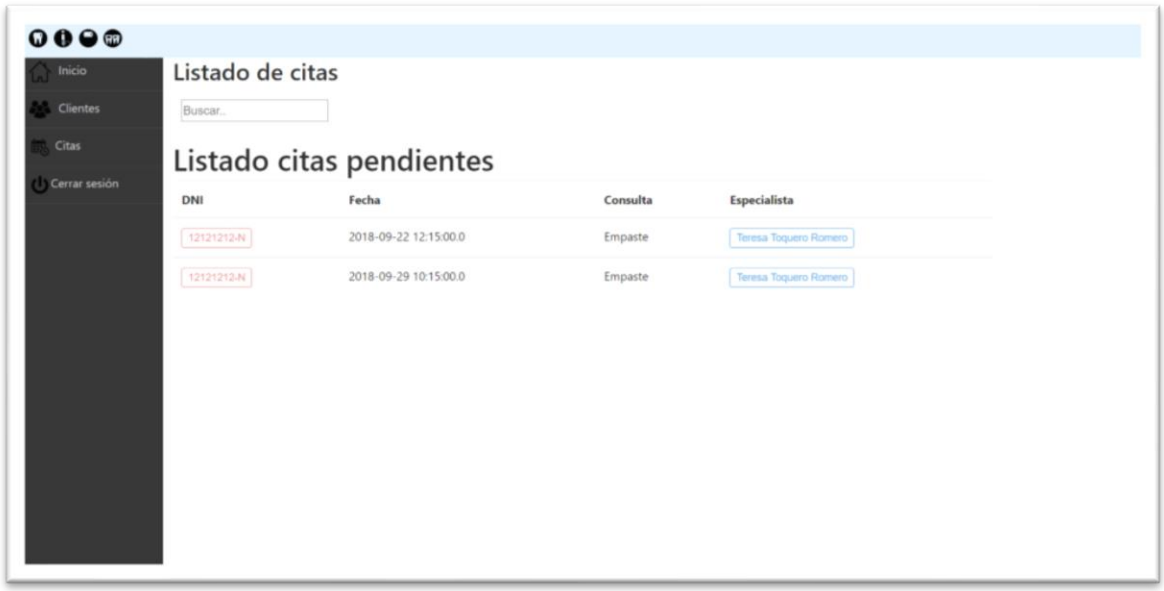


Ilustración 57. Citas pendientes. Manual de usuario.

Panel de usuario con rol administrador de la clínica (administrativo)

En este caso, la ilustración 58, tenemos el listado de clientes. En los cuales podemos ver sus expedientes (datos personales que puede modificar el administrador únicamente) o añadir cita.

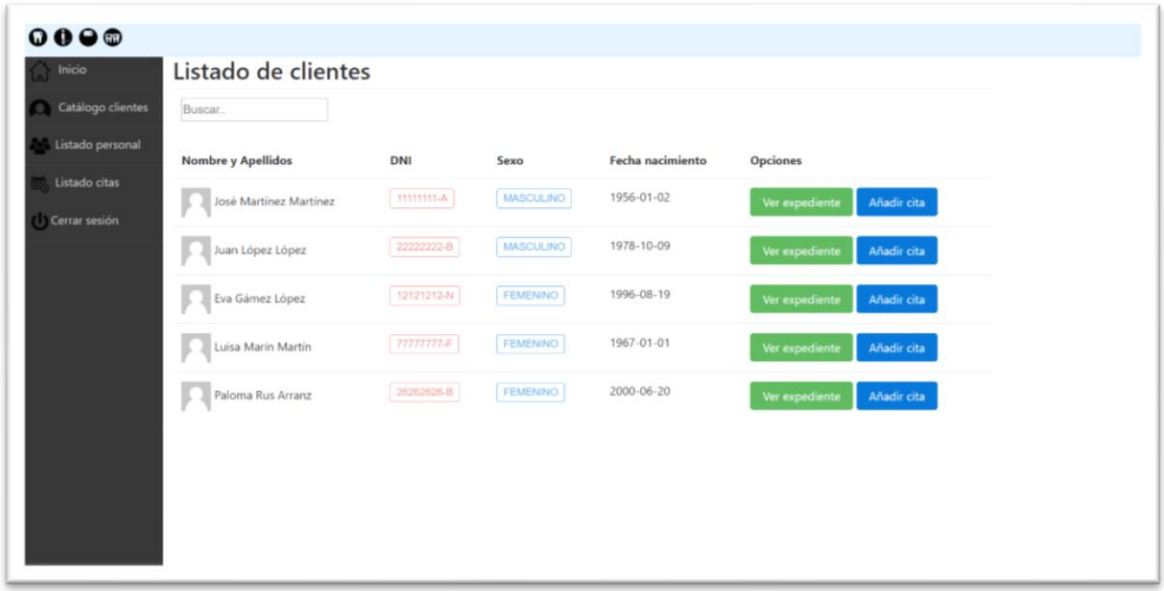


Ilustración 58. Listado clientes administrador. Manual de usuario.

Como hemos comentado, en la siguiente ilustración 59, tenemos los datos personales del cliente, los cuales se pueden modificar.

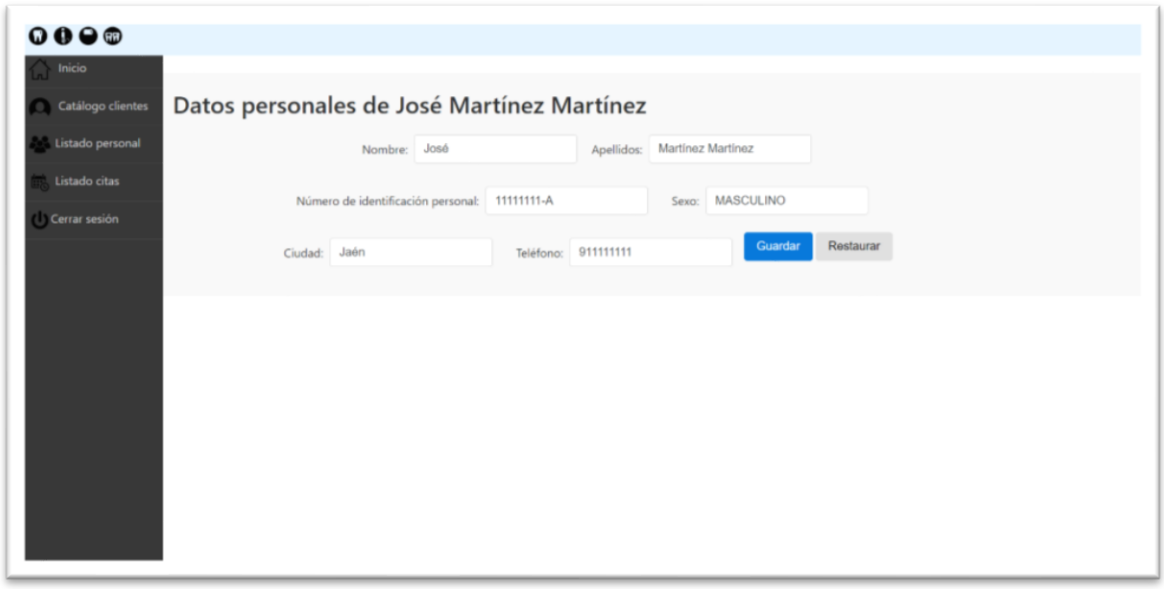


Ilustración 59. Datos personal cliente. Manual de usuario.

A continuación, pasamos a añadir cita, ilustración 60, esta vista es prácticamente igual que la vista en ocasiones anteriores, nada mas que con un detalle cambiado, el DNI, es el DNI del cliente que hemos visto en la lista anterior.

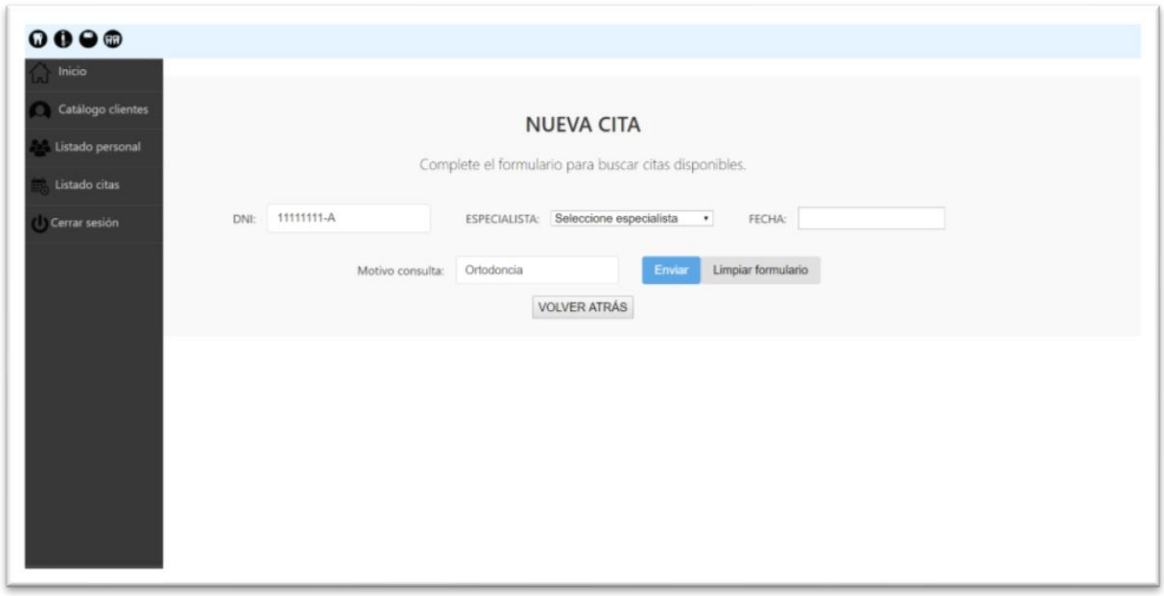


Ilustración 60. Añadir cita cliente. Manual de usuario.

Esta ilustración 61, es muy parecida a las anteriores, añadiendo ver las citas de cada personal y poder añadir un personal nuevo a la clínica.

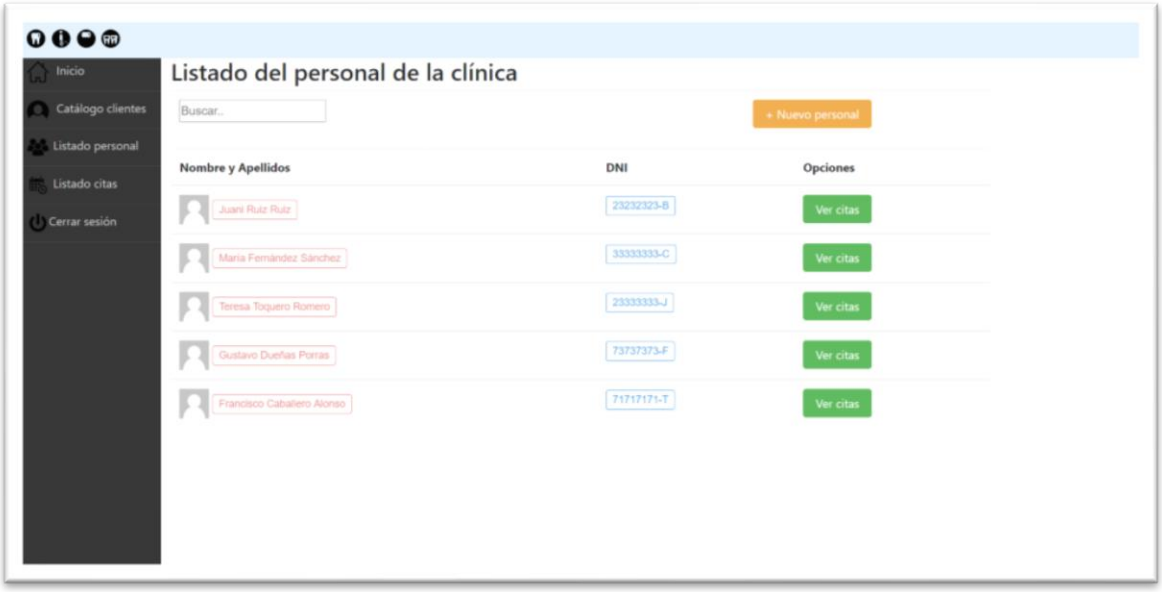


Ilustración 61. Listado personal. Manual de usuario.

Como podemos ver en la ilustración 62, las citas por personal de la clínica.

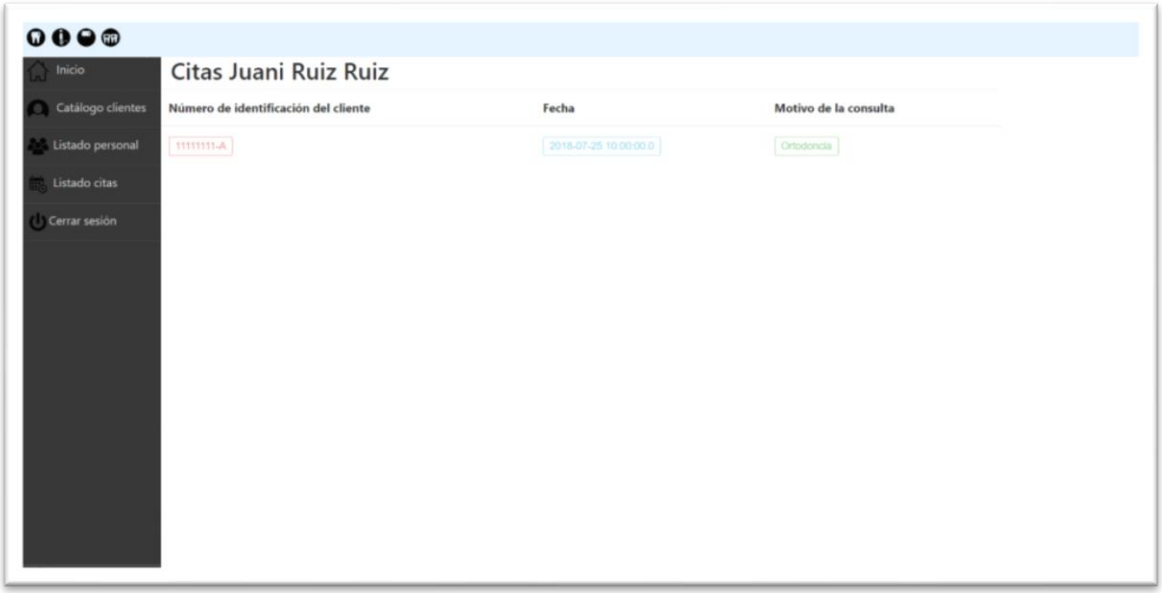


Ilustración 62. Citas personal. Manual de usuario.

Vemos en la siguiente ilustración 63, la creación del personal de la clínica, rellenando un formulario.

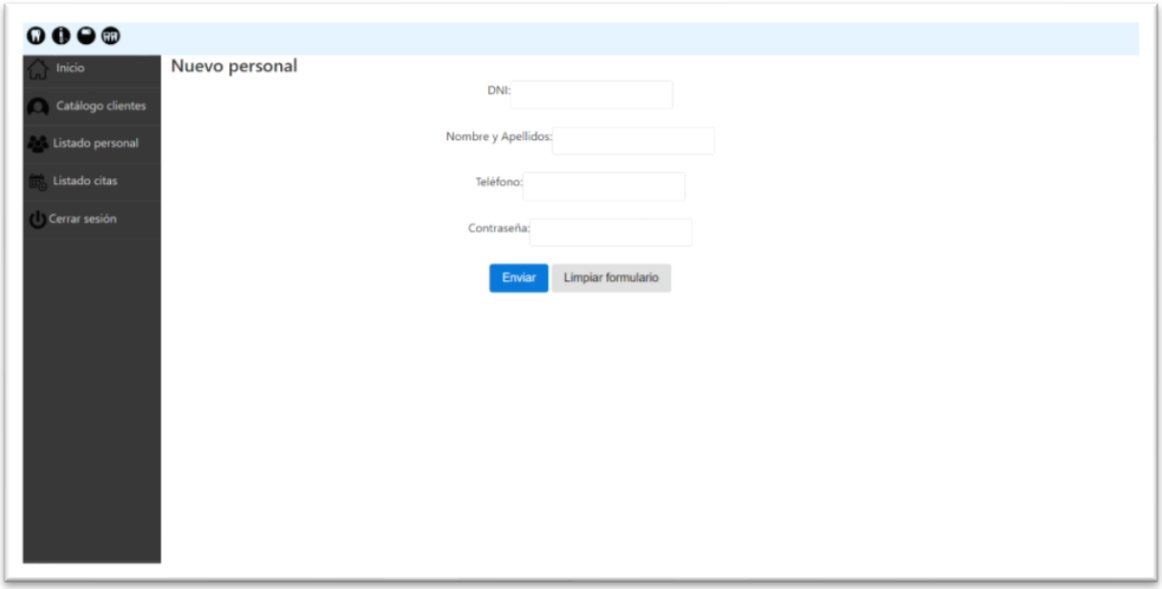


Ilustración 63. Nuevo personal. Manual de usuario.

Y por ultimo el listado de las citas pendientes y finalizadas. Las pendientes únicamente se pueden anular. Ilustración 64.

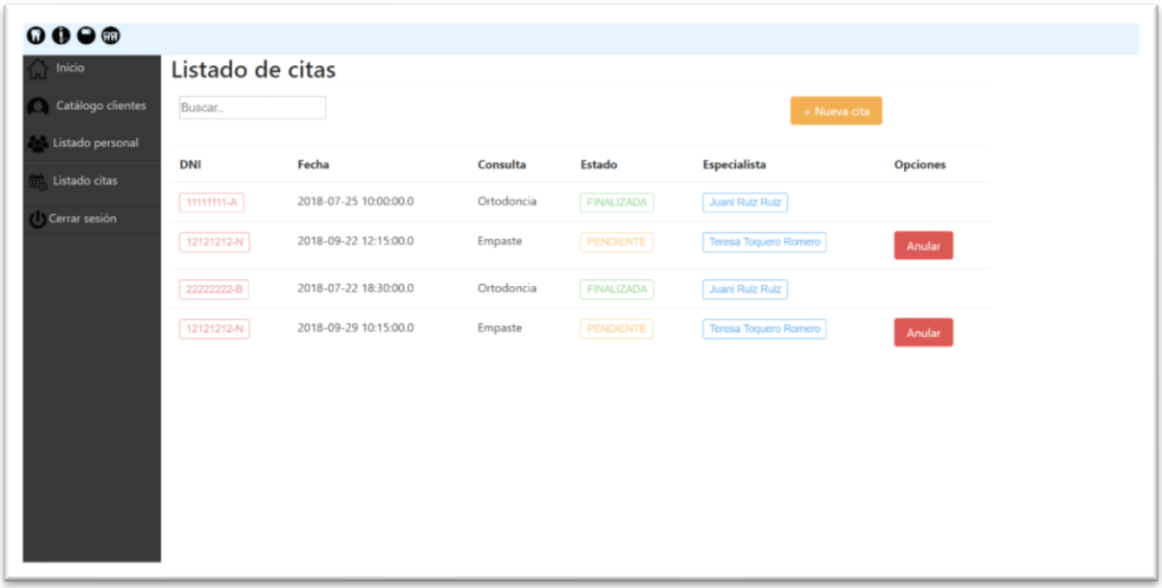


Ilustración 64. Listado citas. Manual de usuario.

APÉNDICE II. DESCRIPCIÓN DE CONTENIDOS SUMINISTRADOS

El CD suministrado cuenta con los siguientes archivos:

- Memoria_Martínez_López_María.pdf, documento memoria.
- Codigo_Martínez_López_María.zip, código fuente del proyecto.
- Disenio_Martínez_López_María.vpp, proyecto Visual Paradigm.

APÉNDICE III. MANUAL DE INSTALACIÓN

Para ejecutar el proyecto es necesario ejecutarlo en ***Netbeans*** y tener lo siguiente.

- **Servidores de Aplicaciones y Servicios Java necesarios para esta aplicación.**
    - Tomcat 8, http://tomcat.apache.org
    - DerbyDB, (incluido en Java EE 7 SDK) para crear la base de datos.
        - En Netbeans, primero debemos darle a "***services"*** y darle al botón derecho ***"New conection".***
        - Posteriormente, poner ***Java DB***, como vemos en la siguiente ilustración 65.

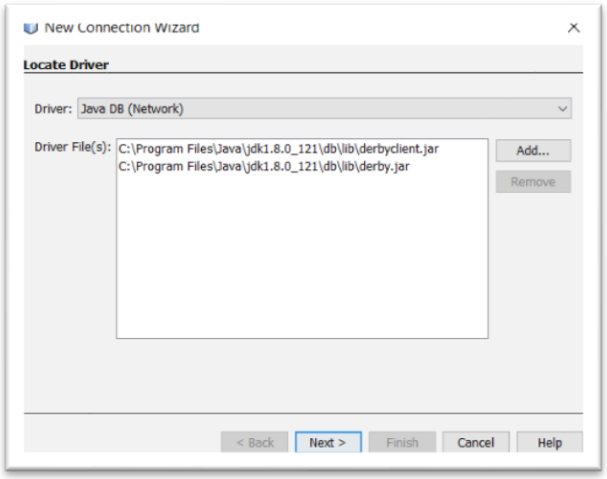


Ilustración 65. Manual de instalación 1.

        - El último paso es rellenar con los siguientes datos:

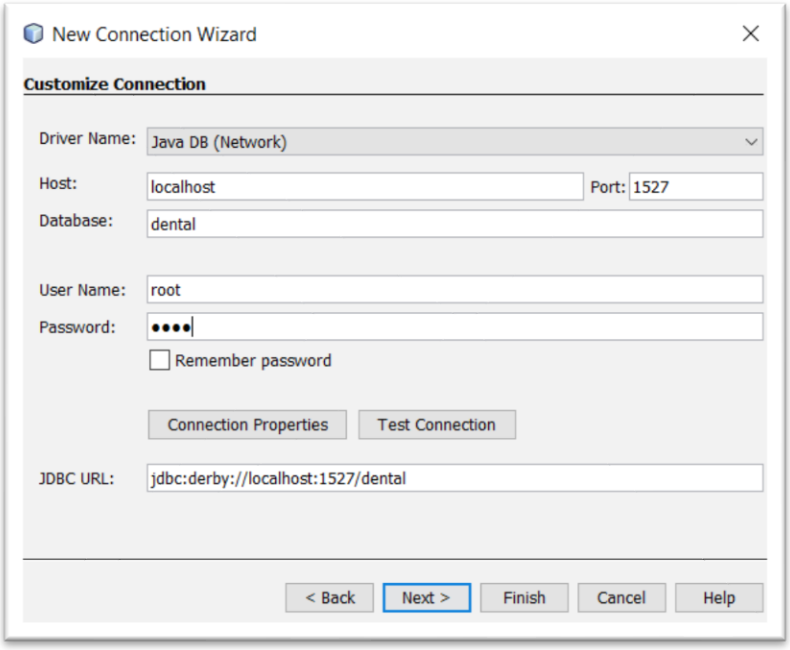


Ilustración 66. Manual de instalación 2.

Tras seguir esta serie de pasos, hay que ejecutar un archivo SQL llamado DBInitScript.sql.