Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO DE FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Aplicación web para la gestión de asistencia social a domicilio en ayuntamientos

Estudante: Manuel Ruiz Vale
Dirección: Manuel Álvarez Díaz

A Coruña, febreiro de 2021.

A quien permaneció a mi lado y a quien le habría gustado pero no pudo estarlo.

Agradecimientos

A mis padres y a Lorena por el amor y apoyo incondicional, uno de mis pilares más importantes. A Teresa por estar desde el minuto cero, dándome el impulso que necesito, siempre a mi lado. A todos mis amigos y compañeros. Gracias.

Quiero agradecer, por supuesto, a mi tutor Manuel Álvarez Díaz por la ayuda y el tiempo prestado para hacer posible la realización de este proyecto.

Resumen

Los servicios de ayuda social a domicilio prestados por ayuntamientos presentan características específicas, lo que complica la utilización de un soporte software genérico para ayudar en su gestión. Por este motivo, la actual organización de los turnos de trabajadores, en algunos ayuntamientos, se realiza de forma manual por parte del gerente, ayudándose de hojas de cálculo. El procedimiento se lleva a cabo utilizando distintas fuentes de datos, lo que provoca inconsistencia entre los mismos. Además, si algún trabajador requiere información sobre horarios, tareas o características de los pacientes, solo puede comunicarse con la gerencia en horario de mañana, lo que dificulta la obtención de información durante otro momento del día.

El objetivo del proyecto es proporcionar un sistema ajustado al dominio, que evite las inconsistencias de las diversas fuentes de información. De esta manera el trabajo de los gerentes del ayuntamiento se ve facilitado, ya que la información de los pacientes y trabajadores se encuentra en una misma aplicación. La consulta de la información de los pacientes y tareas asignadas por parte de los trabajadores se vuelve más cómoda, posibilitando su obtención en cualquier momento del día. Además la aplicación proporciona un tipo de usuario utilizado por los pacientes y/o sus familiares, que les permite ver qué tareas les han sido programadas y quien se encargará de realizarlas.

El sistema es diseñado e implementado como una aplicación web, utilizando *SpringBoot/JPA* en el backend y una aplicación de tipo SPA (*Single Page Application*) utilizando *Javascript* (*React + Redux*) en el frontend.

Abstract

Home social assistance services provided by municipalities have specific characteristics, what complicates the use of generic software support to assist in their management.For this reason, the current organization of workers' shifts in some city councils is done manually by the manager, using spreadsheets. The procedure is carried out using different data sources, what causes inconsistency between them. In adittion, if any worker requires information about schedules, tasks or patient characteristics, they can only communicate with management in the morning, this makes it difficult to obtain the information during another time of the day. The purpouse of the project is to provide a system adjusted to the domain, which avoids the inconsistences of the various sources of information. In this way the work of the city manager is facilitated, since the information of patients and workers is in the same application. Consultation of patients information and assigned tasks by workers becomes more

comfortable, making it possible to obtain them at any time of the day. Furthermore, the application provides a type of user used by patients and/or their families, that allows them to see which tasks have been programmed and who will be in charge of performing them. The system will be designed and implemented as a web application, using *SpringBoot/JPA* in the backend and a SPA (*Single Page Application*) using *Javascript* (*React + Redux*) in the frontend.

Palabras clave:

- Ayuntamiento
- Horarios
- Tareas
- Aplicación web
- React
- Redux
- Springboot
- Hibernate
- SPA

Keywords:

- Council
- Schedule
- Tasks
- Web application
- React
- Redux
- Springboot
- Hibernate
- SPA

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La población española tiene, estadísticamente, una avanzada edad. Si se pone el foco en Galicia, el número de personas de una edad superior a los 70 años con respecto al total de la población es aún mayor. Por tanto, es normal que suba la demanda del servicio de asistencia a domicilio proporcionado por los ayuntamientos de cada zona, ya que cuando se alcanza una cierta edad o se posee alguna diversidad funcional, la realización de las actividades básicas de la vida diaria se ve dificultada y es necesario cierto nivel de asistencia.

La organización de los turnos de trabajadores del sector no es una tarea trivial, ya que en usuarios de avanzada edad o niveles de dependencia altos, un cambio de trabajador supone un gran desconcierto. Además, según la ley 39/2006 [1] y su última reforma en el decreto-ley 20/2012[2], se establecen diferentes grados de dependencia, que determinan el nivel de prioridad entre los usuarios del servicio a la hora de recibirlos por parte del ayuntamiento. Por ejemplo, ante la petición de un mismo horario por dos pacientes, se debería dar preferencia al que menos facilidades tiene para realizar tareas básicas por si mismo o puede disponer de ayuda en su hogar por parte de sus familiares. Por estas razones y casuísticas tan específicas que es necesario tener en cuenta, no suelen automatizarse la creación de horarios y la asignación de trabajadores a pacientes.

1.1 Determinación de la situación actual

Para la realización de este trabajo, se ha contactado con el personal encargado del servicio de asistencia de un ayuntamiento, en concreto de en torno a los 3500 habitantes, aunque podría extrapolarse a más ayuntamientos con un volumen de población similar. En el ayuntamiento contactado es el gerente del servicio el que se encarga de interpretar la información proveniente de distintas fuentes. Algunas de las fuentes son: los listados de concesiones de dependencia de usuarios ofrecidos por la *Xunta de Galicia*, los listados de trabajadores contratados por el ayuntamiento o las anotaciones de comunicaciones con trabajadores y usuarios

del servicio. Toda la información es necesaria para componer los horarios semanalmente, ayudándose de hojas de cálculo para realizar el trabajo. Una vez organizados, se reúne con cada uno de los trabajadores para entregar el horario semanal, impreso en papel.

La comunicación, tanto de los usuarios del servicio de ayuda como de los trabajadores con el gerente del ayuntamiento se hace vía telefónica o presencial, únicamente en horario de mañana y de lunes a viernes, cuando el consistorio permanece abierto. Si un usuario quiere saber quién acudirá a realizar el servicio, no puede solicitar esa información cuando desee.

La aplicación de la tecnología digital en este ámbito añadiría numerosas ventajas como la agrupación de los datos requeridos por el gerente del servicio para la gestión de trabajadores, pacientes y tareas, ofreciendo mayor facilidad para la composición de horarios o la consulta de información por parte de trabajadores y pacientes del servicio, permitiendo el acceso desde cualquier dispositivo y en cualquier momento, solventando la limitación que supone el horario del ayuntamiento.

1.2 Alcance y objetivos

El objetivo de este trabajo de fin de grado es el diseño e implementación un sistema que facilite a los trabajadores las tareas de gestión del servicio de asistencia social a domicilio disponible algunos ayuntamientos.

La aplicación debe de ser capaz de almacenar la información de varios ayuntamientos y gestionarlos de forma independiente, además de permitir registrar usuarios que desempeñan una función dentro de la misma. Dependiendo del perfil de usuario, podrá utilizar distintas funcionalidades: gestión de ayuntamientos, gestión usuarios (empleados, gerentes del servicio y usuarios asociados a pacientes), gestión de pacientes y organización de horarios.

Los pacientes, o sus familiares en caso de incapacidad, deben de poder consultar sus propios horarios y anotaciones. De esta manera se solventa el problema de los horarios para obtener información del servicio.

La aplicación debe tener soporte para varios idiomas y mantener una interfaz sencilla para facilitar el uso a los diferentes perfiles de usuarios.

Adicionalmente, el sistema desarrollado debe ser fácil de mantener y permitir la inclusión de nuevas funcionalidades en caso de que fuesen requeridas en un futuro. Esto se consigue con la aplicación de buenas prácticas de diseño y desarrollo.

Capítulo 2

Base Tecnológica

Este capítulo abordará las tecnologías utilizadas para la realización del proyecto. Se comentarán los lenguajes empleados, así como los *frameworks* y librerías. También se describirán protocolos y herramientas utilizadas.

2.1 Lenguajes

2.1.1 Java

Java [3] es un lenguaje de propósito general orientado a objetos. El código escrito en Java se ejecuta en su propia máquina virtual (JVM), lo que permite que Java pueda ejecutar sus aplicaciones en cualquier dispositivo para el que exista una versión de la JVM, ya sea un ordenador, dispositivo móvil o sistema embebido

Existe una gran cantidad de librerías y *frameworks* disponibles que hacen, por ejemplo, más sencilla la interacción del código con una base de datos o la creación de servicios web.

2.1.2 HTML

HTML [4] es un lenguaje de marcado (*Hypertext Markup Language*) utilizado para definir la estructura de una página web y sus contenidos. Estos contenidos pueden ser, entre otros, texto, imágenes, tablas o enlaces.

Este lenguaje está estandarizado por parte del W3C y actualmente su última versión es HTML5.

2.1.3 CSS

CSS [5] es un lenguaje de estilos (Cascading Style Sheets) que se usa para cambiar la apariencia de los documentos HTML o XML. En CSS se describe como será renderizado cada elemento en la pantalla.

La utilización de CSS permite separar el contenido del documento de la forma de presentación de este, lo que hace posible que una misma página con el mismo contenido se muestre diferente para cada tipo de dispositivo, facilitando su uso.

2.1.4 JavaScript

JavaScript [6] es un lenguaje de programación interpretado, compilado en tiempo de ejecución (*just-in-time*), basado en prototipos, multiparadigma, de un solo hilo, dinámico y con soporte para orientación a objetos, programación imperativa y declarativa.

Es utilizado en muchos entornos distintos y en él se basan entornos de ejecución tales como *Node.js*, *Apache CouchDB*, lenguajes como *TypeScript* y frameworks como *Angular*, *Vue* o *React*. JavaScript es un dialecto del estándar *ECMAScript* cuya última versión estable es *ECMAScript 2016*.

2.1.5 JSON

JSON (*JavaScript Object Notation*) [7] es un lenguaje de formato ligero ideado para la transmisión de datos en la web. Este lenguaje es sencillo de leer por parte de las personas y sencillo de interpretar para las máquinas, lo que permite que sea muy utilizado y extendido.

Los documentos JSON constan de dos tipos de contenedores: una colección de estructuras campo-valor, como por ejemplo un diccionario, una tabla hash o equivalentes, y una lista ordenada de valores, por ejemplo un array o una lista.

2.2 Frameworks y librerías

2.2.1 Core

Spring

El ecosistema Spring [8] es un conjunto de frameworks creado por Apache para la plataforma Java que facilita la creación del backend y de las aplicaciones web del lado del servidor en general (*server-side-applications*). Spring está formado por módulos, cada uno destinado a una característica en concreto: inyección de dependencias, recursos, validación, pruebas, acceso a datos, etc.

Para facilitar aún más su uso, Spring tiene un framework llamado Spring-Boot que hace más sencilla la interacción del código con los distintos módulos del propio ecosistema. Alguno de los módulos utilizados para la realización del proyecto son *spring-data-jpa* y *spring-web* para facilitar el mapeo de objetos con la base datos y la creación de una API REST, respectivamente.

Hibernate

Hibernate [9] es un herramienta de software libre de mapeo objeto-relacional para Java, desarrollado por la compañía RedHat. Este framework ayuda en el mapeo entre objetos y atributos de una base de datos relacional. Se sitúa una capa por encima de JDBC lo que permite una mayor abstracción al encargarse el framework de mapear cada objeto y entidad además de gestionar la persistencia.

Actualmente hibernate se encuentra en la versión 5.4 *stable* y 6.0 *development*

2.2.2 Web

React

React [10] es un framework para el desarollo de interfaces web de aplicaciones diseñado para JavaScript y desarrollado por Facebook. Es un framework basado en componentes, que permite que estos sean encapsulados y puedan reutilizarse lo máximo posible.

React también facilita la gestión y el envío de datos a través de la aplicación al utlizar JavaScript en lugar de plantillas HTML.

Redux

Redux [11] permite que una página web maneje el estado de los componentes permitiendo que algunos datos sean cacheados por el navegador y no sea necesario repetir peticiones, o se puedan almacenar o modificar los datos antes de ser enviados al servidor. Este framework permite que los datos tengan una única fuente de verdad.

Redux suele utilizarse con React pero puede ser utilizado con cualquier otro framework que maneje vistas.

Bootstrap

Bootstrap [12] es una librería para estilos que permite aplicarlos a páginas web para hacerlas adaptativas utilizando HTML, CSS y JavaScript. Esto permite que la utilización de estilos para la creación de una página web sea mucho más sencilla y rápida, con clases ya predefinidas. Actualmente Bootstrap tiene soporte para HTML5 y CSS3 y puede ser utilizado en la mayoría de navegadores existentes en la actualidad.

FullCalendar

FullCalendar [13] es una librería de código abierto que hace más sencilla la utilización de calendarios y la gestión de eventos en una página web. La librería contiene manejadores y métodos que permiten modificar el estado de la aplicación y trabajar con los datos que los

elementos de la librería proporcionan. Además FullCalendar se integra con los frameworks más utilizados actualmente para el desarollo web (React, Vue y Angular)

FullCalendar tiene un tipo de licencia estándar que es gratuita, incluso en el ámbito empresarial y actualmente se encuentra en la versión 5.3.2.

2.2.3 Pruebas

JUnit

JUnit [14] es un framework que permite la automatización de pruebas unitarias para el lenguaje Java. Este framework proporciona anotaciones que hacen más sencilla la creación de los tests, permitiendo analizar la salida de cada método y proporcionar integración con muchos de los IDEs más utilizados. JUnit también tiene integración con sistemas de gestión de proyectos como, por ejemplo, Apache Maven.

2.3 Protocolos

HTTP / HTTPS

HTTP (*HypertText Transfer Protocol*) [15] es un protocolo de comunicación que permite la transferencia de datos en Internet, desarollado por el W3C y la IETF. Es un protocolo que sigue el patrón petición-respuesta entre una parte cliente y otra servidor, respectivamente.

En HTTP existen distintos métodos de petición y códigos de respuesta que nos indican si la petición ha sido resuelta correctamente, permitiendo matizar el tipo de respuesta, o si por el contrario la petición ha causado algún tipo de error respondiéndose con el código pertinente (que tiene su significado asociado).

HTTPS es una versión segura basada en el protocolo HTTP, para securizar el protocolo se utiliza SSL/TLS (*Secure Sockets Layer*/*Transport Layer Security*) para crear una conexión cifrada entre cliente y servidor.

2.4 Herramientas de desarrollo

Git

Git [16] es una herramienta de control de versiones gratuita y de código abierto, que nos facilita el manejo de proyectos en un equipo de desarollo. Git permite la utilización de ramificaciones en la línea de desarollo de un mismo repositorio para crear, por ejemplo, distintas funcionalidades de una misma aplicación en paralelo, que pueden posteriormente fusionarse para publicar una versión final.

Apache Maven

Apache Maven [17] es una herramienta de gestión de proyectos software muy popular. Se basa en un fichero (*POM*), donde se indican las librerías, frameworks y distintos elementos que se necesitan para instalar dependencias, desplegar o probar un proyecto, entre otras acciones. Permite administrar la construcción, la creación de informes y documentación de un proyecto y centralizarlo en una única herramienta.

Los objetivos principales de Apache Maven son hacer más sencillo el proceso de construcción de los proyectos, proveer un sistema de creación uniforme y fomentar el uso de buenas prácticas de desarrollo.

Eclipse

Eclipse [18] es un IDE desarrollado en Java inicialmente por IBM, destinado en un comienzo para programar en el mismo lenguaje, y ahora funcional para muchos otros lenguajes como Python, C/C++, entre otros.

Eclipse emplea *plug-ins* para proporcionar más funcionalidades, permitiéndole incluso trabajar con lenguajes de procesamiento de texto como Latex o herramientas de pruebas automatizadas como JUnit y tener además su propio compilador y depurador para probar en tiempo de ejecución los proyectos desarrollados en su entorno.

VisualStudio Code

Visual Studio Code [19] es un editor de código de Microsoft en el que se puede desarrollar con la mayoría de lenguajes de programación que se usan en la actualidad. VS Code es un editor bastante ligero que permite la utilización de extensiones creadas por terceros que le añaden funcionalidades adicionales.

Postman

Postman [20] es una herramienta de desarrollo de software que permite crear peticiones para testar una API. Ofrece una gran cantidad de herramientas para poder crear peticiones y visualizar las respuestas de un servidor de forma cómoda, agilizando el proceso de pruebas de una API.

2.5 Sistemas de gestión de base de datos

MySQL

MySQL es un gestor de base de datos relacional, en el TOP 5 de los más utilizados entre los sistemas de gestión de bases de datos (*SGBD*) relacionales en los últimos años. [21] Además se considera un componente de la pila de desarrollo *LAMP* (Linux, Apache, MySQL, Python/Pearl) y *WAMP* (Windows, Apache, MySQL, Python/Pearl)

MySQL es un SGDB en el que destacan algunas características como: disponibilidad para una gran cantidad de plataformas y sistemas, el uso de conexiones seguras, transaccionalidad en los accesos y el uso de claves foráneas, entre otras. [22]

MySQL Workbench

MySQL Workbench es una herramienta gráfica que permite visualizar y gestionar una base de datos MySQL. Además de gestionar la configuración de la propia base de datos, los usuarios pueden acceder a ella, realizar peticiones SQL utilizando su propio editor entre otras muchas acciones.

MySQL Workbench ha sido desarollado por Oracle y cuenta con una versión gratuita de código abierto (*MySQL Workbench Community Edition*) y otras dos versiones dedicadas a ámbitos empresariales (*MySQL Workbench Standard Edition* y *MySQL Workbench Enterprise Edition*). [23]

Capítulo 3

Estado del arte

En la actualidad, existen algunas soluciones software para la gestión de ayuda social a domicilios. Algunas de las aplicaciones existentes ofrecen facilidades en la gestión de pacientes y otras brindan un sistema completo de gestión de pacientes, trabajadores y horarios. No se han encontrado soluciones que ofrezcan las funcionalidades requeridas *open source*, por lo que el uso de los sistemas existentes implican, normalmente, el pago de un servicio mensual o anual.

Aunque sí se han encontrado algunas aplicaciones gratuitas que presentan algunas funcionalidades como la formación de gestión de empleados, diseño de cuadrantes y generación de informes, como por ejemplo *LaborOfficeFree* [24] que es el plan gratuito de la aplicación *LaborOffice*.

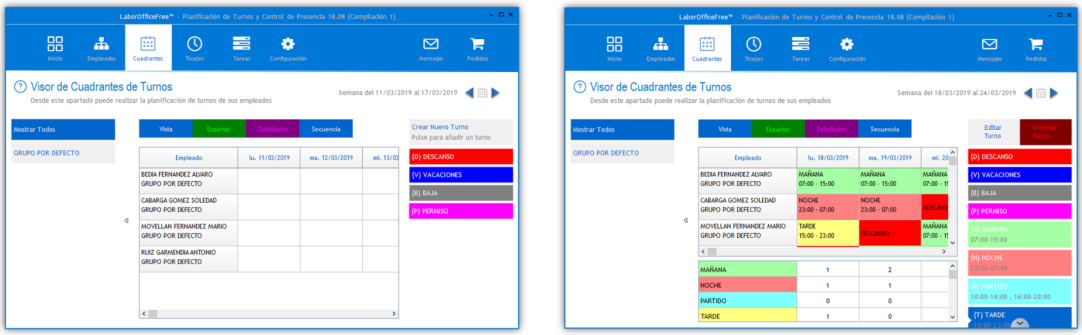


Figura 3.1: Ejemplo de interfaz de *LaborOfficeFree*

El inconveniente de una aplicación como *LaborOfficeFree* es que no se ajusta a las necesidades requeridas por el servicio de ayuda a domicilio de un ayuntamiento. El tipo de trabajo necesita la gestión de trabajadores y pacientes. Además no podría habilitarse un perfil para los pacientes o familiares.

Por otra parte, aplicaciones como *gesad* [25], del Grupo Trevenque, ofrecen un sistema

completo con una gran cantidad de funcionalidades como la planificación y control de incidencias, control de presencia, facturación y cobros, copagos, gestión de un posible servicio de cátering o generación de informes.

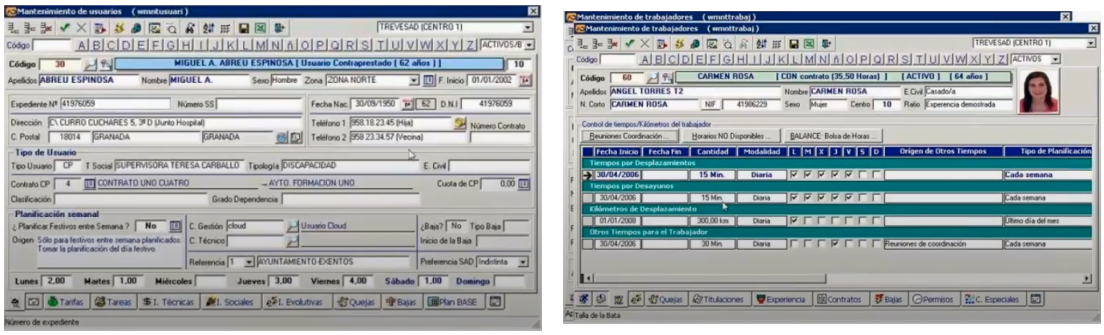


Figura 3.2: Ejemplo de interfaz de *gesad*

Otras aplicaciones, como *ecosad* [26] están más orientadas a ser un programa de facturación de servicios de ayuda a domicilio, aunque también ofrecen gestión de usuarios y facilidades para la creación de horarios.

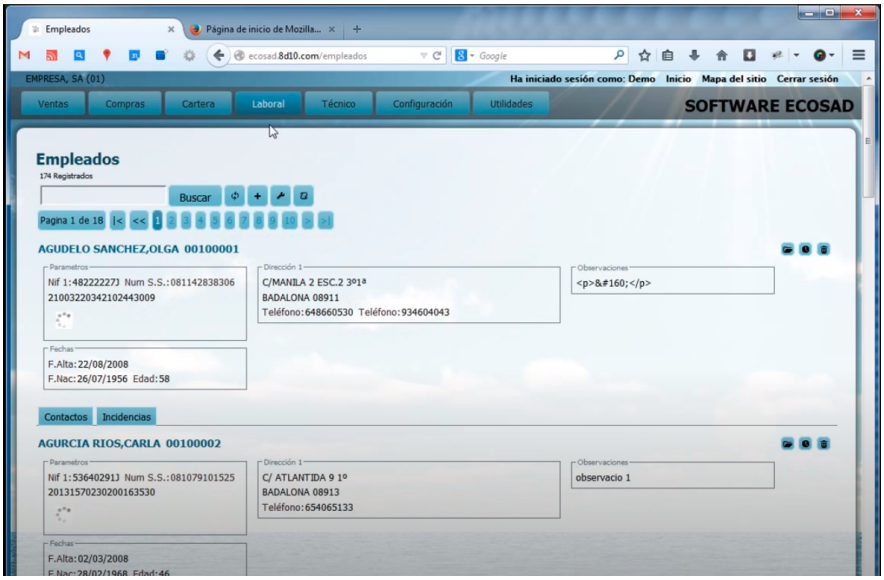


Figura 3.3: Ejemplo de interfaz de *ecosad*

Las interfaces de algunos de los sistemas pueden ser poco amigables y suponer una curva de aprendizaje grande para los nuevos usuarios de la aplicación, ya que es bastante probable que los usuarios no estén familiarizados con las plataformas de gestión informatizadas.

Las aplicaciones que ofrecen un macrosistema, con un gran abanico de funcionalidades, podrían ser sobredimensionadas para ayuntamientos con un volumen reducido de usuarios

del servicio de asistencia domiciliaria, ya que en ayuntamientos pequeños los recursos destinados a este servicio suelen ser limitados.

Por todo lo anterior, parece justificado el desarrollo de una aplicación web adecuada a las funcionalidades de un dominio tan específico como el de la gestión de asistencia social a domicilio de ayuntamientos de pequeño y mediano tamaño. De esta forma la usabilidad de la aplicación se adapta a los perfiles concretos de usuarios.

Capítulo 4

Análisis de viabilidad

Para afrontar la realización del proyecto, previamente se realiza un análisis de viabilidad, en el que se tienen en cuenta la dimensión temporal, tecnológica y económica. Este análisis determina si es posible afrontar la realización del proyecto.

Comenzando por la dimensión temporal, el tiempo no supone un gran inconveniente ya que al tratarse de un trabajo de fin de grado, el alcance del anteproyecto fue definido teniendo en cuenta la duración de un proyecto de estas características, que debería ser de al menos de 25 horas de trabajo por crédito ects.

Centrándose en la dimensión tecnológica, para realizar la aplicación web se ha decidido la utilización de *Spring + JPA/Hibernate*, ya que con este *framework* se cubren las necesidades de comunicación con la base de datos, gestión de lógica de negocio y creación de API que sirva al cliente. Para la base de datos se ha optado por el uso de *MySQL* debido a que es previamente conocido y una opción que se adecúa al uso que se le va a dar. Por último, para el desarrollo del subsistema *frontend* se ha optado por el uso de *React* debido a su gran cantidad de librerías disponibles, el uso de herramientas como *Redux*, abundante documentación y formación, tanto de conceptos avanzados como iniciación. Por lo tanto no se encuentran problemas en la utilización de las tecnologías debido a que todas ellas están disponibles, son de código abierto y son ampliamente utilizadas de forma efectiva para el desarrollo de aplicaciones web.

El mayor riesgo que afectaría tanto a la dimensión tecnológica como temporal sería el aprendizaje en las tecnologías empleadas, aunque no es necesaria la formación en todas, ya que algunas de ellas son previamente conocidas.

Finalmente, poniendo el foco sobre la dimensión económica tampoco se encuentran problemas, debido a que solo es necesario un ordenador y acceso a Internet ya que todas las tecnologías utilizadas son del tipo *open source*. Pese a que los recursos disponibles son limitados, no se encuentra un problema en el aspecto económico ya que se posee un equipo en el que realizar el desarrollo antes de la comenzar el proyecto y el coste del acceso a Internet es fácilmente asumible.

Capítulo 5

Introducción al desarrollo realizado

5.1 Introducción

En este proyecto se propone el desarrollo de una aplicación web que permita facilitar la gestión del servicio de ayuda social a domicilio de un ayuntamiento, tanto por parte del gerente y su tarea de composición de horarios, como los trabajadores y pacientes, permitiendo acceder a la información de las tareas, trabajadores y pacientes en cualquier momento.

Durante la realización de este trabajo de fin de grado se han utilizado las tecnologías y herramientas descritas en el capítulo 2. En este capítulo se contextualiza el uso de cada una de ellas en la arquitectura del sistema desarrollado. Además se presenta la metodología utilizada y las iteraciones en las que se divide el desarrollo.

5.2 Tecnologías

El sistema desarrollado se basa en una arquitectura cliente-servidor. Esto implica que la aplicación se divida en dos subsistemas: *backend* y *frontend*. A continuación se describen cada uno de ellos contextualizando las tecnologías utilizadas, apoyándose en la figura 5.1:

La aplicación servidor (*backend*), construida en lenguaje *Java* [3] y basada en un proyecto de *SpingBoot* [8], es la encargada de comunicarse con la base de datos, reaccionando a las peticiones que le llegan a través de la capa de servicio. La base de datos utilizada es *MySQL* [22], debido a que es una base de datos ampliamente utilizada, con bajos requerimientos y gran estabilidad. La comunicación entre los objetos *Java* con la BD se realiza utilizando *Hibernate* [9], una implementación del estándar *JPA*. La capa modelo, donde se almacena la lógica de negocio se comunica con la capa servicios del *backend*. En esta se desarrolla una API dividida en controladores que ponen accesibles las funcionalidades del modelo de la aplicación a clientes (en este caso el frontal web o *frontend*).

El subsistema backend es un proyecto de *SpringBoot* y esto presenta ventajas como la

integración con *Hibernate* o la facilidad para desarrollar la API *REST* a través de los controladores. El módulo de *Springboot* llamado *Spring Boot Starter Web* proporciona un servidor *Apache Tomcat* en el que estará accesible la API y algunos otros módulos como *spring-web*. Utilizando este último se proporcionan métodos que permiten la transformación de objetos java utilizados en los controladores a objetos *JSON* [7]

La comunicación entre los subsistemas *frontend* y *backend* se realiza mediante el protocolo HTTP [15]. Las peticiones al servidor son enviadas utilizando los métodos *POST, GET, PUT* y *DELETE*. Las respuestas son enviadas utilizando *JSON* por ser extensamente utilizado en las comunicaciones web.

El otro subsistema principal, *frontend* está alojado en un servidor al que el navegador accede para obtener los recursos necesarios para ejecutar el frontend. Aunque el servidor utilizado podría ser cualquiera, incluso el mismo *Apache Tomcat* utilizado para el *backend*, en la figura 5.1 se ilustra con un servidor *webpack* por ser el utilizado durante el desarrollo del proyecto. El *frontend* es construido utilizando el lenguaje *Javascript* [6], utilizando *React* [10] y *Redux* [11]. *React* propone una construcción de componentes que son reutilizables e independientes, manteniendo su propio estado. Cuando el estado tiene que ser compartido por dos o más componentes se utiliza *Redux*.

React renderiza la interfaz a partir de lenguaje *HTML* y se le aplican los estilos (*CSS*) a través de la librería de *Bootstrap*.

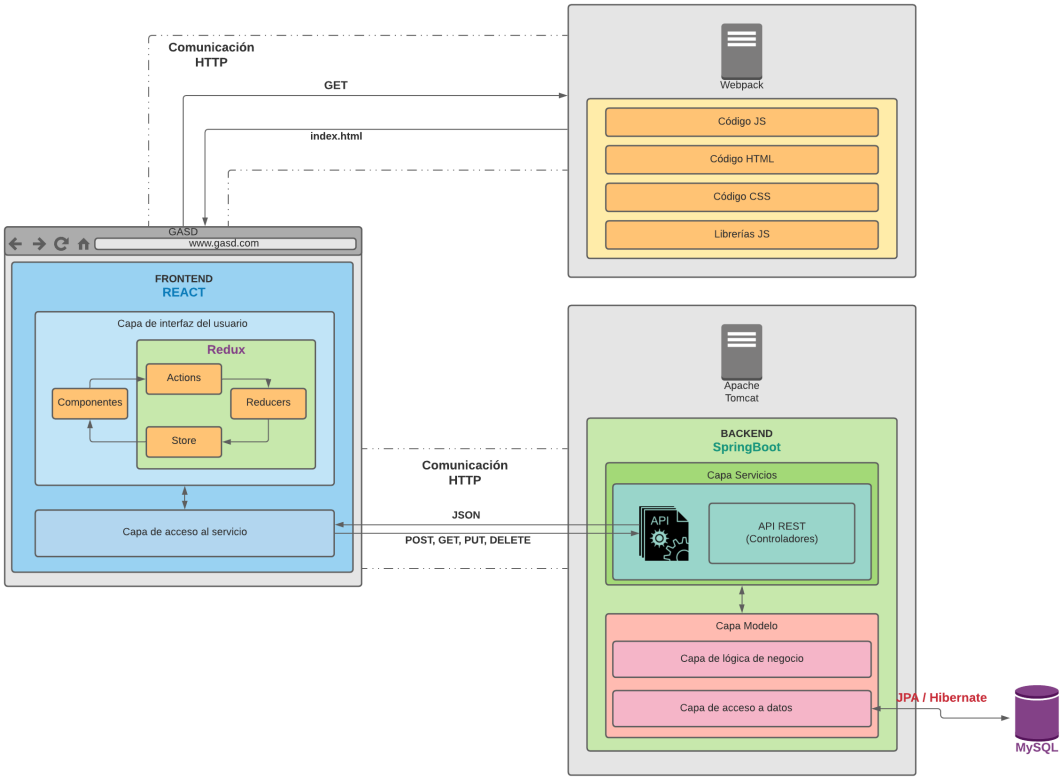


Figura 5.1: Diagrama de tecnologías utilizadas

5.3 Metodología e iteraciones

Se ha utilizado una metodología de es desarrollo iterativa e incremental basada en casos de uso. Se han definido las iteraciones a partir de los casos de uso recogidos en la especificación de requisitos. Al final de cada iteración se obtiene un sistema que irá incorporando funcionalidades a medida que las iteraciones van avanzando.

Para la realización de los diagramas, se ha hecho uso del lenguaje de modelado UML [27].

En particular, se han definido 5 iteraciones, donde la primera es utilizada únicamente para formación y la especificación de requisitos, mientras que las otras 4 se corresponden con los cuatro grupos de funcionalidades lógicamente relacionadas. A continuación se describen:

- **Formación**: Formación tanto en tecnologías como en el dominio, además de la especificación de requisitos.
- **Iteración 1**: Funcionalidades de gestión de usuarios.
- **Iteración 2**: Funcionalidades de gestión de ayuntamientos.
- **Iteración 3**: Funcionalidades de gestión de pacientes.
- **Iteración 4**: Funcionalidades de gestión de tareas.

En el capítulo 6, página 19, se analizan en detalle cada una de las iteraciones.

Capítulo 6

Planificación y análisis de costes

6.1 Planificación y seguimiento

Para la realización del proyecto se ha utilizado desarrollo iterativo e incremental. La utilización de este tipo de desarrollo permite que al final de cada iteración, exista una versión de la aplicación a la que se van añadiendo funcionalidades a medida que se van realizando dichas iteraciones.

En este proyecto se diferencian dos fases en la planificación. La primera fase es de formación en las tecnologías y herramientas utilizadas a lo largo del mismo. La segunda, se compone de cuatro iteraciones en las que se van construyendo las distintas funcionalidades que componen el sistema. Las iteraciones siguen la división en la que se agrupan las funcionalidades que están lógicamente relacionadas. Al final de cada una de las iteraciones se lleva a cabo una reunión con el director del proyecto para revisar todo lo realizado durante la misma. A continuación se detallan todas las fases de la planificación:

- **Fase de formación**: Durante esta primera fase, representada en la figura 6.1 se realiza la formación en las tecnologías que van a ser utilizadas en el proyecto y se llevan a cabo las reuniones con el trabajador del servicio y el gerente del ayuntamiento, utilizadas para entender mejor el dominio de la aplicación y las carencias del sistema actual. Por último se realiza la especificación de requisitos con la información recopilada.

- **Iteración 1: Gestión de usuarios**: Durante esta iteración, mostrada en la figura 6.2, se implementan todos los casos de uso relacionados con los usuarios y su gestión. Se comienza con un análisis de los requisitos y se va diseñando, implementando y probando desde las entidades hasta la interfaz en el *frontend*. Al finalizar esta iteración se consigue una aplicación en la que se pueden realizar operaciones como la creación usuarios y la gestión de las parejas de trabajadores, entre otras.

- **Iteración 2: Gestión de ayuntamientos**: En esta iteración se desarrollan todas las

funcionalidades de los ayuntamientos y su gestión (fig. 6.3). La primera tarea realizada es el análisis de los requisitos y, como en la iteración anterior, se diseña, implementa y prueba en sentido ascendente, desde las entidades hasta el *frontend*. Al final de esta iteración se podrán realizar las funcionalidades de gestión de ayuntamientos además de las implementadas en al iteración anterior.

- **Iteración 3: Gestión de pacientes**: En esta iteración, la más corta, representada en la figura 6.4, se desarrollan las funcionalidades de la gestión de pacientes siguiendo la misma metodología utilizada en las iteraciones anteriores. Al finalizar esta iteración se podrá, por ejemplo, añadir pacientes al sistema y crear notas asignadas a ellos, entre otras funcionalidades.

- **Iteración 4: Gestión de tareas**: Siguiendo los pasos descritos en las anteriores iteraciones, al finalizar esta, se obtendrá la versión final del sistema, que contará con todas las funcionalidades descritas durante la especificación de requisitos. La planificación para esta iteración se puede observar en la figura 6.5.

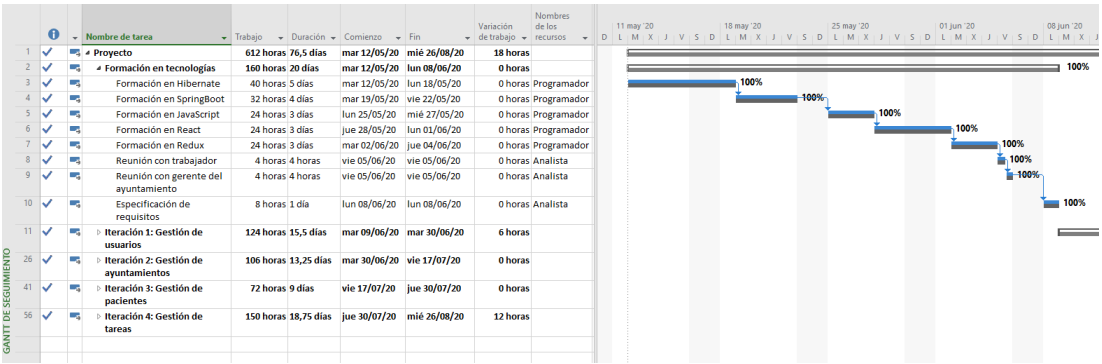
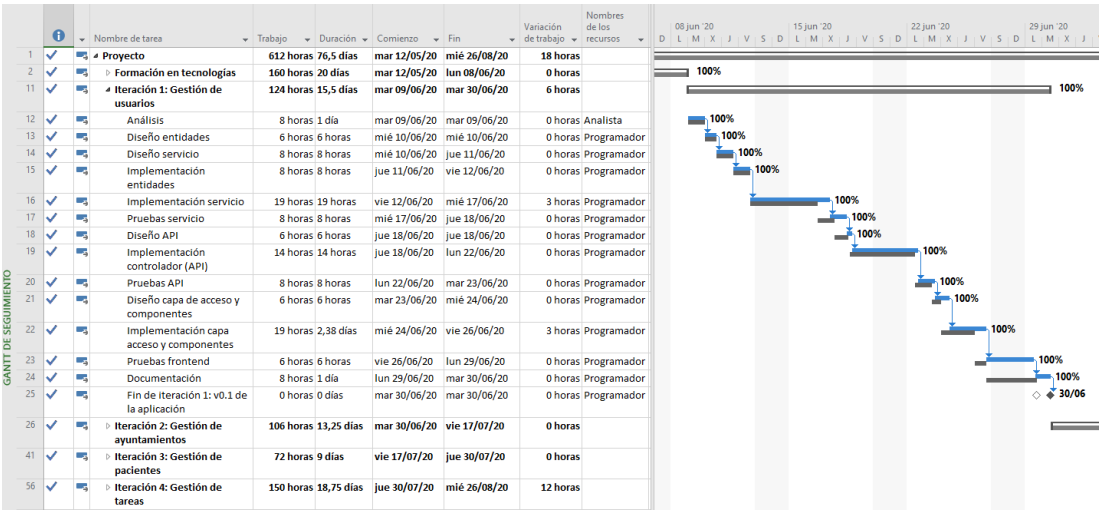


Figura 6.1: Formación en tecnologías

La planificación establecida ha sido cumplida, excepto en dos de las iteraciones, gestión de usuarios y gestión de tareas.

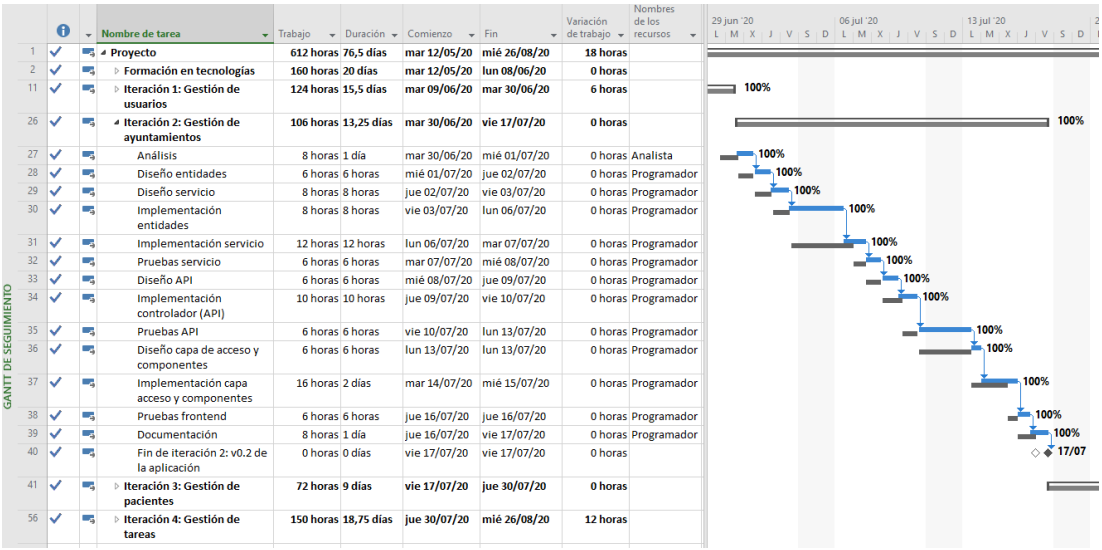
La iteración 1 ha tenido una variación de 6 horas respecto a la planificación establecida debido a que la familiarización con las tecnologías era baja y a pesar de que la planificación había sido ligeramente sobreestimada, no ha sido suficiente. En la figura 6.2 se puede observar como la variación se produce en las tareas de implementación del servicio y en la implementación de la capa de acceso y los componentes del *frontend*.

Por otro lado, la iteración 4 también ha tenido variaciones debido a problemas en la implementación de las funcionalidades de creación y modificación de tareas, tanto en el servicio como en el controlador. La variación de la implementación del *frontend* se debe a problemas con el uso de la librería *full-calendar*.



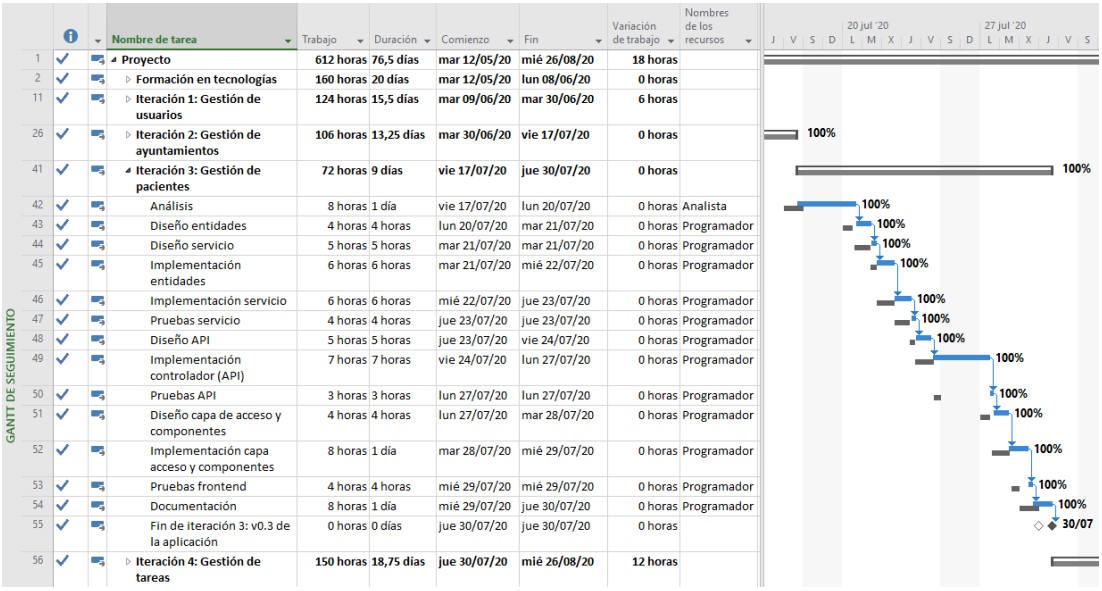
	Nombre de tarea	Trabajo	Duración	Comienzo	Fin	Variación de trabajo	Nombres de los recursos
1	**Proyecto**	**612 horas**	**76,5 días**	**mar 12/05/20**	**mié 26/08/20**	**18 horas**	
2	**Formación en tecnologías**	**160 horas**	**20 días**	**mar 12/05/20**	**lun 08/06/20**	**0 horas**	
11	**Iteración 1: Gestión de usuarios**	**124 horas**	**15,5 días**	**mar 09/06/20**	**mar 30/06/20**	**6 horas**	
12	Análisis	8 horas	1 día	mar 09/06/20	mar 09/06/20	0 horas	Analista
13	Diseño entidades	6 horas	6 horas	mié 10/06/20	mié 10/06/20	0 horas	Programador
14	Diseño servicio	8 horas	8 horas	mié 10/06/20	jue 11/06/20	0 horas	Programador
15	Implementación entidades	8 horas	8 horas	jue 11/06/20	vie 12/06/20	0 horas	Programador
16	Implementación servicio	19 horas	19 horas	vie 12/06/20	mié 17/06/20	3 horas	Programador
17	Pruebas servicio	8 horas	8 horas	mié 17/06/20	jue 18/06/20	0 horas	Programador
18	Diseño API	6 horas	6 horas	jue 18/06/20	jue 18/06/20	0 horas	Programador
19	Implementación controlador (API)	14 horas	14 horas	jue 18/06/20	lun 22/06/20	0 horas	Programador
20	Pruebas API	8 horas	8 horas	lun 22/06/20	mar 23/06/20	0 horas	Programador
21	Diseño capa de acceso y componentes	6 horas	6 horas	mar 23/06/20	mié 24/06/20	0 horas	Programador
22	Implementación capa acceso y componentes	19 horas	2,38 días	mié 24/06/20	vie 26/06/20	3 horas	Programador
23	Pruebas frontend	6 horas	6 horas	vie 26/06/20	lun 29/06/20	0 horas	Programador
24	Documentación	8 horas	1 día	lun 29/06/20	mar 30/06/20	0 horas	Programador
25	Fin de iteración 1: v0.1 de la aplicación	0 horas	0 días	mar 30/06/20	mar 30/06/20	0 horas	Programador
26	**Iteración 2: Gestión de ayuntamientos**	**106 horas**	**13,25 días**	**mar 30/06/20**	**vie 17/07/20**	**0 horas**	
41	**Iteración 3: Gestión de pacientes**	**72 horas**	**9 días**	**vie 17/07/20**	**jue 30/07/20**	**0 horas**	
56	**Iteración 4: Gestión de tareas**	**150 horas**	**18,75 días**	**jue 30/07/20**	**mié 26/08/20**	**12 horas**	

Figura 6.2: Iteración 1: Gestión de usuarios



	Nombre de tarea	Trabajo	Duración	Comienzo	Fin	Variación de trabajo	Nombres de los recursos
1	**Proyecto**	**612 horas**	**76,5 días**	**mar 12/05/20**	**mié 26/08/20**	**18 horas**	
2	**Formación en tecnologías**	**160 horas**	**20 días**	**mar 12/05/20**	**lun 08/06/20**	**0 horas**	
11	**Iteración 1: Gestión de usuarios**	**124 horas**	**15,5 días**	**mar 09/06/20**	**mar 30/06/20**	**6 horas**	
26	**Iteración 2: Gestión de ayuntamientos**	**106 horas**	**13,25 días**	**mar 30/06/20**	**vie 17/07/20**	**0 horas**	
27	Análisis	8 horas	1 día	mar 30/06/20	mié 01/07/20	0 horas	Analista
28	Diseño entidades	6 horas	6 horas	mié 01/07/20	jue 02/07/20	0 horas	Programador
29	Diseño servicio	8 horas	8 horas	jue 02/07/20	vie 03/07/20	0 horas	Programador
30	Implementación entidades	8 horas	8 horas	vie 03/07/20	lun 06/07/20	0 horas	Programador
31	Implementación servicio	12 horas	12 horas	lun 06/07/20	mar 07/07/20	0 horas	Programador
32	Pruebas servicio	6 horas	6 horas	mar 07/07/20	mié 08/07/20	0 horas	Programador
33	Diseño API	6 horas	6 horas	mié 08/07/20	jue 09/07/20	0 horas	Programador
34	Implementación controlador (API)	10 horas	10 horas	jue 09/07/20	vie 10/07/20	0 horas	Programador
35	Pruebas API	6 horas	6 horas	vie 10/07/20	lun 13/07/20	0 horas	Programador
36	Diseño capa de acceso y componentes	6 horas	6 horas	lun 13/07/20	lun 13/07/20	0 horas	Programador
37	Implementación capa acceso y componentes	16 horas	2 días	mar 14/07/20	mié 15/07/20	0 horas	Programador
38	Pruebas frontend	6 horas	6 horas	jue 16/07/20	jue 16/07/20	0 horas	Programador
39	Documentación	8 horas	1 día	jue 16/07/20	vie 17/07/20	0 horas	Programador
40	Fin de iteración 2: v0.2 de la aplicación	0 horas	0 días	vie 17/07/20	vie 17/07/20	0 horas	
41	**Iteración 3: Gestión de pacientes**	**72 horas**	**9 días**	**vie 17/07/20**	**jue 30/07/20**	**0 horas**	
56	**Iteración 4: Gestión de tareas**	**150 horas**	**18,75 días**	**jue 30/07/20**	**mié 26/08/20**	**12 horas**	

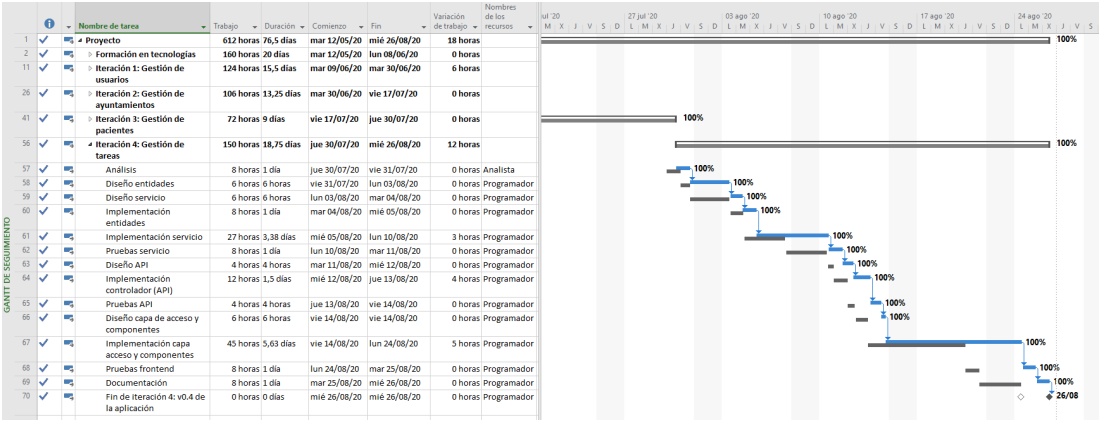
Figura 6.3: Iteración 2: Gestión de ayuntamientos



	Nombre de tarea	Trabajo	Duración	Comienzo	Fin	Variación de trabajo	Nombres de los recursos
1	**Proyecto**	**612 horas**	**76,5 días**	**mar 12/05/20**	**mié 26/08/20**	**18 horas**	
2	**Formación en tecnologías**	**160 horas**	**20 días**	**mar 12/05/20**	**lun 08/06/20**	**0 horas**	
11	**Iteración 1: Gestión de usuarios**	**124 horas**	**15,5 días**	**mar 09/06/20**	**mar 30/06/20**	**6 horas**	
26	**Iteración 2: Gestión de ayuntamientos**	**106 horas**	**13,25 días**	**mar 30/06/20**	**vie 17/07/20**	**0 horas**	
41	**Iteración 3: Gestión de pacientes**	**72 horas**	**9 días**	**vie 17/07/20**	**jue 30/07/20**	**0 horas**	
42	Análisis	8 horas	1 día	vie 17/07/20	lun 20/07/20	0 horas	Analista
43	Diseño entidades	4 horas	4 horas	lun 20/07/20	mar 21/07/20	0 horas	Programador
44	Diseño servicio	5 horas	5 horas	mar 21/07/20	mar 21/07/20	0 horas	Programador
45	Implementación entidades	6 horas	6 horas	mar 21/07/20	mié 22/07/20	0 horas	Programador
46	Implementación servicio	6 horas	6 horas	mié 22/07/20	jue 23/07/20	0 horas	Programador
47	Pruebas servicio	4 horas	4 horas	jue 23/07/20	jue 23/07/20	0 horas	Programador
48	Diseño API	5 horas	5 horas	jue 23/07/20	vie 24/07/20	0 horas	Programador
49	Implementación controlador (API)	7 horas	7 horas	vie 24/07/20	lun 27/07/20	0 horas	Programador
50	Pruebas API	3 horas	3 horas	lun 27/07/20	lun 27/07/20	0 horas	Programador
51	Diseño capa de acceso y componentes	4 horas	4 horas	lun 27/07/20	mar 28/07/20	0 horas	Programador
52	Implementación capa acceso y componentes	8 horas	1 día	mar 28/07/20	mié 29/07/20	0 horas	Programador
53	Pruebas frontend	4 horas	4 horas	mié 29/07/20	mié 29/07/20	0 horas	Programador
54	Documentación	8 horas	1 día	mié 29/07/20	jue 30/07/20	0 horas	Programador
55	Fin de iteración 3: v0.3 de la aplicación	0 horas	0 días	jue 30/07/20	jue 30/07/20	0 horas	
56	**Iteración 4: Gestión de tareas**	**150 horas**	**18,75 días**	**jue 30/07/20**	**mié 26/08/20**	**12 horas**	

GANTT DE SEGUIMIENTO

Figura 6.4: Iteración 3: Gestión de pacientes



	Nombre de tarea	Trabajo	Duración	Comienzo	Fin	Variación de trabajo	Nombres de los recursos
1	**Proyecto**	**612 horas**	**76,5 días**	**mar 12/05/20**	**mié 26/08/20**	**18 horas**	
2	**Formación en tecnologías**	**160 horas**	**20 días**	**mar 12/05/20**	**lun 08/06/20**	**0 horas**	
11	**Iteración 1: Gestión de usuarios**	**124 horas**	**15,5 días**	**mar 09/06/20**	**mar 30/06/20**	**6 horas**	
26	**Iteración 2: Gestión de ayuntamientos**	**106 horas**	**13,25 días**	**mar 30/06/20**	**vie 17/07/20**	**0 horas**	
41	**Iteración 3: Gestión de pacientes**	**72 horas**	**9 días**	**vie 17/07/20**	**jue 30/07/20**	**0 horas**	
56	**Iteración 4: Gestión de tareas**	**150 horas**	**18,75 días**	**jue 30/07/20**	**mié 26/08/20**	**12 horas**	
57	Análisis	8 horas	1 día	jue 30/07/20	vie 31/07/20	0 horas	Analista
58	Diseño entidades	6 horas	6 horas	vie 31/07/20	lun 03/08/20	0 horas	Programador
59	Diseño servicio	6 horas	6 horas	lun 03/08/20	mar 04/08/20	0 horas	Programador
60	Implementación entidades	8 horas	1 día	mar 04/08/20	mié 05/08/20	0 horas	Programador
61	Implementación servicio	27 horas	3,38 días	mié 05/08/20	lun 10/08/20	3 horas	Programador
62	Pruebas servicio	8 horas	1 día	lun 10/08/20	mar 11/08/20	0 horas	Programador
63	Diseño API	4 horas	4 horas	mar 11/08/20	mié 12/08/20	0 horas	Programador
64	Implementación controlador (API)	12 horas	1,5 días	mié 12/08/20	jue 13/08/20	4 horas	Programador
65	Pruebas API	4 horas	4 horas	jue 13/08/20	vie 14/08/20	0 horas	Programador
66	Diseño capa de acceso y componentes	6 horas	6 horas	vie 14/08/20	vie 14/08/20	0 horas	Programador
67	Implementación capa acceso y componentes	45 horas	5,63 días	vie 14/08/20	lun 24/08/20	5 horas	Programador
68	Pruebas frontend	8 horas	1 día	lun 24/08/20	mar 25/08/20	0 horas	Programador
69	Documentación	8 horas	1 día	mar 25/08/20	mié 26/08/20	0 horas	Programador
70	Fin de iteración 4: v0.4 de la aplicación	0 horas	0 días	mié 26/08/20	mié 26/08/20	0 horas	Programador

GANTT DE SEGUIMIENTO

Figura 6.5: Iteración 4: Gestión de tareas

Se ha optado por asumir las variaciones producidas en el seguimiento debido a que el retraso producido no es significativo.

6.2 Análisis de costes

Los costes económicos del proyecto se han obtenido a partir de las horas realizadas y los costo que esto implica sumado al coste del hardware utilizado para el desarrollo. Los costes indirectos como la electricidad y el acceso a Internet, no se han tenido en cuenta.

Hardware utilizado para el desarrollo:

Modelo	ASUS ROG GL753VD
CPU	Intel®Core i7-7700HQ
RAM	16 GB
Capacidad	1TB HDD + 500GB SSD
OS	Windows 10 + Ubuntu 20.04
Tarjeta gráfica	NVIDIA GeForce GTX 1050

El ordenador portátil tiene un precio de 1369€. Estimando que la esperanza de vida media es de 4 años, el costo por año son 342€. Teniendo en cuenta que el proyecto tenía una duración prevista de 74,25 días y ha durado un total de 76,5 días, el costo prorrateado del equipo son 69,57€ en planificación y 71,68€ al final del seguimiento.

Las horas realizadas por cada uno de los recursos son las siguientes:

	Planificado	**Seguimiento**
Analista	48	48
Programador	546	564

Tabla 6.1: Horas trabajadas por recurso. Planificado vs seguimiento

Teniendo en cuenta los datos de horas de trabajo anteriores y tomando como coste por hora de un analista de 19,22€/h y de un programador de 18,35€/h, obtenidos del salario medio anual publicado en la *Guía Spring Professional 2020* [28], se obtienen los siguientes costos totales:

	Planificado	**Seguimiento**
Analista	922,56€	922,56€
Programador	10.019,10€	10.349,40€
Hardware	69,57€	71,68€
TOTAL	11.011,23€	11.343,64€

Tabla 6.2: Costos totales. Planificado vs seguimiento

Capítulo 7

Requisitos del sistema

7.1 Introducción

Para conseguir una mayor concreción en lo que el sistema necesitaría realizar, se llevaron a cabo varias reuniones con el actual gerente de un ayuntamiento, encargado de la gestión de un equipo de asistencia a domicilio. Tras analizar la información recopilada se identificaron los requisitos funcionales y no funcionales, así como los actores y casos de uso requeridos.

7.2 Requisitos funcionales

Cuando el gerente de un ayuntamiento tiene que organizar los horarios semanales de los servicios de ayuda a domicilio, obtiene la información de varias fuentes. En algunas ocasiones estas no son consistentes ya que, pueden no estar actualizadas o simplemente no estar en el mismo formato. Con la aplicación propuesta se busca tener una fuente común y consistente de información, que el gerente puede consultar para componer los horarios, revisar un contrato de un trabajador, o añadir una nota a un paciente que le ha sido comunicada, entre otras tareas.

En esta sección se enumerarán y describirán los requisitos funcionales que el sistema debe cumplir:

RF 1 **Gerencia de ayuntamientos.** Se necesita que el sistema sea capaz de gestionar varios ayuntamientos a la vez y que sus datos sean tratados de manera independiente.

RF 2 **Gestión de usuarios.** Se requiere que la aplicación sea capaz de administrar los usuarios que van a utilizarla. Dependiendo del rol que vaya a ejecutar cada tipo de usuario, tendrá a su disposición distintas funcionalidades. Los usuarios pertenecerán a un único ayuntamiento (a excepción del rol del administrador que podrá no estar ligado a un ayuntamiento). Además cada usuario podrá modificar ciertos datos propios, mientras otros solo serán modificables por los usuarios con mayores privilegios.

RF 3 **Gestión de contratos.** Es necesario que la aplicación sea capaz de almacenar los datos de un contrato de un trabajador. Esta acción solo puede ser realizada por un gerente.

RF 4 **Creación y edición de parejas de trabajadores.** Es conveniente que la aplicación sea capaz de crear asociaciones de trabajadores en forma de parejas y modificar las mismas.

RF 5 **Administración de pacientes.** La aplicación debe permitir la gestión de pacientes pero la podrán realizar los gerentes del ayuntamiento. Los pacientes pertenecen a un único ayuntamiento, permaneciendo invisibles por otros ayuntamientos de la aplicación.

RF 6 **Creación y edición de notas asociadas a pacientes.** La aplicación debe tener la capacidad de crear notas asociadas a cada paciente. Cada nota debe tener una visibilidad y dependiendo de esta, la verán todos los tipos de usuarios o solamente los trabajadores del ayuntamiento al que pertenece el paciente en cuestión.

RF 7 **Creación de tareas.** El sistema tiene que poder crear tareas asignando un trabajador a un paciente durante un tiempo concreto en una fecha señalada. Esta acción solo debe estar disponible para los gerentes del ayuntamiento.

RF 8 **Modificación del estado de una tarea de pendiente a completado.** Se precisa que la aplicación disponga de una acción que permita modificar el estado de una tarea a estado completado. Esta acción estará disponible tanto para el trabajador que está asignado a esa tarea como para el gerente del ayuntamiento.

RF 9 **Copia de tareas de una semana a semanas posteriores.** El sistema debe facilitar la creación de tareas de forma semanal. Para ello debe permitir copiar las tareas de un paciente de una semana a otra.

RF 10 **Gestión de días no trabajados.** Se requiere que la aplicación permita almacenar los días que un trabajador no acude al trabajo por el motivo que sea (día de asuntos propios, baja, vacaciones,etc.)

RF 11 **Internacionalización.** El sistema estará disponible en gallego, inglés y castellano.

7.3 Requisitos no funcionales

Los requisitos no funcionales son cruciales para que la aplicación sea cómoda y fácil de utilizar para el usuario. En el sistema consideramos las siguientes:

RNF 1 **Control de errores.** Es necesario que la aplicación controle los errores que puede introducir un usuario a la hora de utilizarla, evitando introducir datos incorrectos donde no se debería y mostrándole al usuario la causa.

RNF 2 **Usabilidad.** La aplicación debe ser usable, esto es, de utilización intuitiva porque facilite la lectura de los textos, descargue rápidamente la información y presente funciones y menús sencillos, para que al usuario le resulte cómodo su uso.

7.4 Actores

Como resultado del análisis de los requisitos, se han identificado los siguientes actores:

- **Usuario no autenticado** Es un usuario que va a utilizar la aplicación pero todavía no ha entrado al sistema y no se sabe que rol ocupa.

- **Administrador** Es el usuario que se encarga de crear los usuarios gerentes de un ayuntamiento y los propios ayuntamientos. Puede no pertenecer a ningún ayuntamiento en concreto y no tiene información concreta de los pacientes y tareas de ninguno de los ayuntamientos.

- **Gerente del ayuntamiento** Es el encargado de la creación y administración de los usuarios, la forma de organizarse de los mismos y de la gestión de los contratos de los trabajadores que están a su cargo. También se encarga de los detalles de su ayuntamiento y de la creación y gestión de los pacientes. Es el gerente el usuario que crea y administra las tareas y horarios de los pacientes, las bajas y los períodos no trabajados de los otros empleados.

- **Trabajador** Es el personal de un ayuntamiento en concreto, encargado de realizar el trabajo asistencial en el domicilio de un paciente.

- **Usuario de paciente** Usuario asociado a un paciente. Este tipo de usuario será el que utilizarán los pacientes o sus familiares y/o tutores.

En el la figura 7.1 se puede observar la relación entre los distintos actores que intervienen en el sistema.

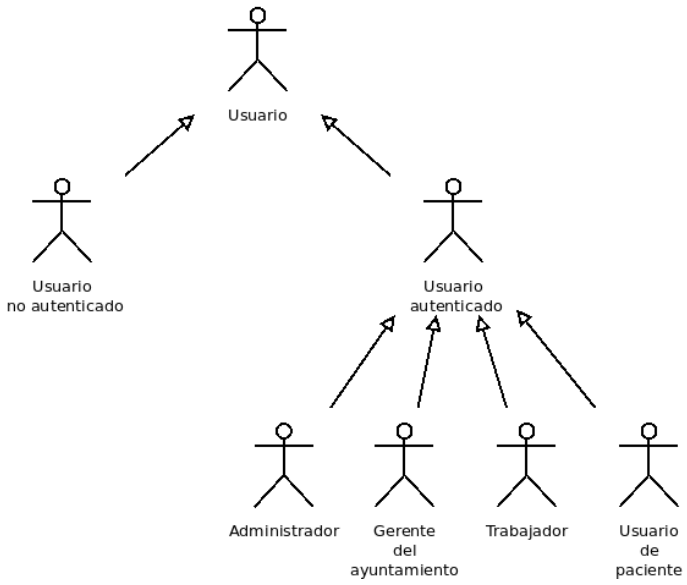


Figura 7.1: Diagrama de actores

7.5 Casos de uso

En la siguiente sección se describen los casos de uso que han sido identificados en la aplicación. Se dividen en cuatro grandes grupos: los casos de uso relativos a la gestión de usuarios y trabajadores, los relacionados con los ayuntamientos y la información relativa a ellos, los casos de uso que tratan sobre los pacientes y los casos de uso relacionados las tareas.

Cada caso de uso o grupo de casos se describe en forma de tabla. En cada una se indica, en este mismo orden: el nombre, el grupo de actores que ejecuta el caso (o casos) de uso, las precondiciones que deben darse para que se lance, la descripción de lo que realiza, el estado en el que se queda el sistema una vez se ha realizado y las excepciones que puede lanzar.

Para evitar repeticiones en la mayoría de tablas de casos de uso se han agrupado las acciones CRUD (Create, Read, Update, Delete) y se han obviado las excepciones típicas como la modificación o borrado de elementos ya eliminados o que todavía no han sido creados.

7.5.1 Gestión de usuarios

La primera sección de casos de uso que se describen son los relacionados con las acciones de los usuarios. Los casos del 2 al 6 son acciones CRUD, por ello estos casos se presentarán agrupados, ya que su ejecución no presenta peculiaridad alguna. En esta sección también se trata la gestión de parejas de trabajadores y los contratos de los mismos.

Es importante aclarar que la gestión de los usuarios y sus direcciones se realiza de forma independiente, en casos de uso distintos.

CU 1	Autenticación
Actores que lo ejecutan	Usuario no autenticado
Precondiciones	El usuario no está autenticado.
Descripción	El actor introduce el usuario y su contraseña y presiona el botón autenticarse para acceder a la aplicación.
Postcondiciones	El usuario permanecerá autenticado.
Excepciones	Las credenciales no coinciden.

Tabla 7.1: CU 1

CU 2 - 6	CRUD Usuarios
Actores que lo ejecutan	Administrador, Gerente del ayuntamiento, Trabajador (Solo lectura)
Precondiciones	El usuario debe estar autenticado. Para eliminar o modificar un usuario debe haber añadido alguno a la base de datos. El usuario debe tener los permisos necesarios para ejecutar la acción, por ejemplo, solo un administrador podrá realizar las acciones de este grupo sobre otros usuarios administradores.
Descripción	**Añadir usuarios:** Se presenta un formulario con campos a rellenar para añadir el usuario al sistema. Si esta acción la ejecuta un gerente, el usuario creado solo podrá pertenecer al mismo ayuntamiento que él. Si este caso de uso es lanzado por un administrador, este podrá elegir a que ayuntamiento es asignado el nuevo usuario. **Visualizar usuarios:** Muestra una lista de usuarios. **Visualizar datos usuario:** Muestra los datos de un usuario en concreto. Si lo ejecuta un trabajador, solo podrá ver los suyos. **Modificar usuario:** Modificar los datos de un usuario. **Borrar usuarios:** Eliminar un usuario de la base de datos.
Postcondiciones	Se almacena, muestra, modifica o borra la información del usuario en el sistema.
Excepciones	No se rellenan campos obligatorios o se rellenan con el formato incorrecto. Nombre de usuario duplicado en la creación o modificación de un usuario

Tabla 7.2: CU 2 - 6

CU 7 - 9	Añadir, visualizar y eliminar direcciones de usuario
Actores que lo ejecutan	Administrador, Gerente del ayuntamiento, Trabajador (Solo lectura)
Precondiciones	El usuario debe estar autenticado. Debe haber creada al menos una dirección de usuario para poder eliminarla. El usuario debe tener los permisos necesarios para ejecutar la acción, por ejemplo, solo un administrador podrá realizar las acciones de este grupo sobre otros usuarios administradores.
Descripción	**Añadir dirección:** Se presenta un formulario con campos a rellenar para introducir el usuario en el sistema. **Visualizar direcciones:** Muestra una lista con las direcciones del usuario. Si lo ejecuta un trabajador, solo podrá ver sus propias direcciones. **Borrar direcciones:** Eliminar la dirección de un usuario de la base de datos.
Postcondiciones	Se almacena, muestra, o borra la dirección de un usuario del sistema.
Excepciones	No se rellenan campos obligatorios o se rellenan con el formato incorrecto.

Tabla 7.3: CU 7 - 9

CU 10	Crear Pareja
Actores que lo ejecutan	Gerente del ayuntamiento
Precondiciones	El usuario debe estar autenticado. Debe haber trabajadores introducidos en la base de datos asignados al ayuntamiento al que pertenece el gerente.
Descripción	Se deben seleccionar dos de los usuarios con el rol de trabajador que no tienen formada pareja actualmente. Además se deben introducir las fechas de inicio y opcionalmente la fecha de fin de validez de la pareja. Se puede indicar fecha de inicio y de fin o solamente la de inicio, siendo la pareja válida hasta que se modifique este campo.
Postcondiciones	Se almacena en la base de datos la información de la pareja creada
Excepciones	No se rellenan campos obligatorios Se introducen datos no válidos

Tabla 7.4: CU 10

CU 11	Visualizar pareja
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador
Precondiciones	El usuario debe estar autenticado.
Descripción	La descripción de este CU es distinto dependiendo de quien lo ejecute: • **Gerente de ayuntamiento:** Se muestran todas las parejas del ayuntamiento al que pertenece el actor que lo ejecuta. • **Trabajador:** Se muestran las parejas de las que el trabajador forma parte.
Postcondiciones	-
Excepciones	-

Tabla 7.5: CU 11

CU 12	Modificar Pareja
Actores que lo ejecutan	Gerente del ayuntamiento
Precondiciones	El usuario debe estar autenticado. Deber haber parejas creadas para poder realizar su modificación.
Descripción	Se presenta un formulario en el que ser permite la modificación de los usuarios que forman la pareja, mostrando como usuarios seleccionables los usuarios que no tienen una pareja válida en el momento de la modificación. Se pueden cambiar también las fechas de inicio y de fin de validez de la pareja.
Postcondiciones	La pareja actualiza los campos modificados.
Excepciones	No se rellenan campos obligatorios Se introducen datos no válidos

Tabla 7.6: CU 12

CU 13	Eliminar Pareja
Actores que lo ejecutan	Gerente del ayuntamiento
Precondiciones	El usuario debe estar autenticado. Deber haber al menos una pareja creada para poder eliminarla.
Descripción	Se muestra un diálogo que se debe aceptar para proceder con la eliminación de la pareja.
Postcondiciones	La pareja es borrada del sistema.
Excepciones	-

Tabla 7.7: CU 13

CU 14 - 17	Añadir, visualizar y modificar contratos
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador (Solo lectura)
Precondiciones	Debe haber tipos de trabajo almacenados en la base de datos para poder crear un contrato. Para modificar un contrato debe haber alguno añadido a la base de datos.
Descripción	Antes de crear o modificar un contrato el actor debe seleccionar un usuario con el rol gerente de ayuntamiento o trabajador para gestionar sus contratos. Una vez seleccionado el usuario el actor puede: **Añadir contrato:** Se debe establecer una fecha de inicio y otra de fin. Se selecciona un valor en el desplegable del campo tipo de contrato, eligiendo entre jornada a tiempo completo o a tiempo parcial. Además se selecciona también en el desplegable del campo tipo de trabajo un valor, eligiendo entre los almacenados en la base de datos. Por último se introducen las horas diarias de trabajo del usuario. **Visualizar contrato:** Se visualiza el contrato en vigor de un usuario seleccionado. Si lo ejecuta un trabajador, verá únicamente su contrato vigente. **Visualizar contratos:** Se visualiza todos los contratos almacenados en la base de datos de un usuario seleccionado. Si lo ejecuta un trabajador, solo podrá ver sus contratos. **Modificar contrato:** Se puede modificar únicamente la fecha de fin. Se puede actualizar el campo tipo de contrato, así como el tipo de trabajo. Además se pueden cambiar las horas diarias de trabajo.
Postcondiciones	Se crea o modifica el contrato en la base de datos.
Excepciones	El usuario no rellena todos los campos.

Tabla 7.8: CU 14 - 17

7.5.2 Gestión de ayuntamientos

En esta sección se tratan los casos de uso relacionados con la gestión de ayuntamientos y de los detalles de ayuntamiento, que son grupos de información de algunos campos de la entidad, que son válidos durante un tiempo estipulado. En esta sección también se incluyen las acciones de gestión de periodos festivos y la definición de los tipos de trabajo. Algunos casos de uso se describen agrupados, debido a que su descripción se puede resumir de forma sencilla.

De forma análoga a la sección anterior, la gestión de los ayuntamientos, sus direcciones y conjuntos de detalles se realizan de forma independiente, con casos de uso completamente diferenciados.

CU 18 - 22	CRUD Ayuntamientos
Actores que lo ejecutan	Administrador
Precondiciones	El usuario debe estar autenticado. Para modificar y eliminar debe haber creado un ayuntamiento.
Descripción	**Añadir ayuntamiento:** Se presenta un formulario con campos a rellenar para añadir el ayuntamiento al sistema. **Visualizar ayuntamientos:** Muestra una lista de ayuntamientos. **Visualizar datos ayuntamiento:** Muestra los datos de un ayuntamiento en concreto. **Modificar ayuntamiento:** Modificar los datos de un ayuntamiento. **Borrar ayuntamiento:** Eliminar la información de un ayuntamiento de la base de datos.
Postcondiciones	Se almacena, muestra, modifica o borra los datos de un ayuntamiento del sistema. La eliminación produce el borrado en cascada de los elementos asociados a él (usuarios, pacientes y todos sus elementos relacionados), acción que es irreversible.
Excepciones	El usuario no rellena todos los campos.

Tabla 7.9: CU 18 - 22

CU 23 - 25	Añadir, visualizar y borrar dirección ayuntamiento
Actores que lo ejecutan	Administrador
Precondiciones	El usuario debe estar autenticado. Para borrar debe haber creada una dirección de ayuntamiento
Descripción	**Añadir dirección de ayuntamiento:** Se presenta un formulario con campos a rellenar para añadir la dirección de un ayuntamiento al sistema. **Visualizar dirección de ayuntamiento:** Muestra una lista de las direcciones de un ayuntamiento. **Borrar dirección de ayuntamiento:** Eliminar la información de la dirección de un ayuntamiento de la base de datos.
Postcondiciones	Se almacena, muestra o borra los datos de la dirección de un ayuntamiento del sistema.
Excepciones	El usuario no rellena todos los campos.

Tabla 7.10: CU 23 - 25

CU 26 - 28	Añadir, mostrar y actualizar detalles de ayuntamientos
Actores que lo ejecutan	Gerente de ayuntamiento
Precondiciones	El usuario debe estar autenticado. Para actualizar debe haber creado al menos un grupo de detalles en el ayuntamiento seleccionado.
Descripción	**Crear detalles ayuntamiento:** Se presenta un formulario con campos a rellenar para añadir el grupo de información del ayuntamiento al sistema, que será valido durante el tiempo indicado. **Visualizar detalles de ayuntamiento:** Muestra una lista con todos los conjuntos de datos de los detalles de un ayuntamiento. **Modificar detalles ayuntamiento:** Modificar los datos de los detalles de un ayuntamiento.
Postcondiciones	Se almacena, muestra o modifica un conjunto de detalles de un ayuntamiento.
Excepciones	El usuario no rellena los campos obligatorios.

Tabla 7.11: CU 26 - 28

CU 29 - 31	Añadir, visualizar y eliminar períodos festivos
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador (Solo visualización)
Precondiciones	El usuario debe estar autenticado. Debe haber creados un grupo de detalles del ayuntamiento al que pertenece el gerente.
Descripción	**Añadir período festivo:** Se muestra un formulario para introducir una fecha de inicio y una fecha de fin del período festivo. Es necesario que la fecha se encuentre dentro del período de validez del grupo de detalles del ayuntamiento. Se muestra además de un campo de descripción para asignarle una etiqueta al período. **Visualizar períodos festivos:** Se cargan todos los períodos festivos asociados al grupo de detalles seleccionado para mostrarlos en pantalla. **Eliminar período festivo:** Se presenta un diálogo que pide confirmación. Si se acepta el período festivo es eliminado de la base de datos.
Postcondiciones	Se añade, lee o elimina un período festivo de la base de datos.
Excepciones	En la creación, el usuario establece una fecha del período festivo fuera del período de validez del grupo de detalles del ayuntamiento. El usuario no rellena los campos obligatorios

Tabla 7.12: CU 29 - 31

CU 32 - 34	Añadir, visualizar y modificar tipos de trabajo.
Actores que lo ejecutan	Gerente del ayuntamiento
Precondiciones	El usuario debe estar autenticado. Para modificar el tipo de trabajo debe haber creado alguno en la base de datos.
Descripción	**Añadir tipo de trabajo:** Se presenta un formulario un campo a rellenar para añadir un tipo de trabajo al sistema. **Visualizar tipos de trabajo:** Muestra una lista con los tipos de trabajo almacenados en la base de datos. **Actualizar tipo de trabajo:** Se presenta un formulario para modificar la información del tipo de trabajo almacenado en la base de datos.
Postcondiciones	Se añade, visualiza y modifica un tipo de trabajo en la base de datos.
Excepciones	No se rellenan todos los campos obligatorios en el formulario de creación o actualización.

Tabla 7.13: CU 32 - 34

7.5.3 Gestión de pacientes

En esta sección se describen los casos de uso relativos a la gestión de pacientes. Además se pueden añadir y visualizar notas de un paciente, para añadir información puntual, consejos o recordatorios sobre un paciente. Los casos de uso de gestión de pacientes se presentan agrupados como CRUD. Las operaciones de gestión de las notas de paciente sí serán detalladas para explicar en profundidad la visibilidad de las mismas y los permisos de gestión.

Al igual que en las dos secciones anteriores, la gestión de los propios pacientes y sus direcciones se realizan de forma independiente, con casos de uso completamente diferenciados.

CU 35 - 39	CRUD pacientes.
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador (Solo visualización)
Precondiciones	El usuario debe estar autenticado. Para modificar o eliminar el paciente debe haber creado alguno en la base de datos.
Descripción	**Añadir paciente:** Se presenta un formulario para rellenar con los datos de un paciente para añadir al sistema. **Visualizar pacientes:** Muestra una lista con los pacientes del ayuntamiento del gerente almacenados en la base de datos. **Visualizar datos paciente:** Muestra los datos de un paciente en concreto almacenado en la base de datos. **Actualizar paciente:** Se presenta un formulario para modificar la información del paciente almacenado en la base de datos. **Eliminar paciente:** Se muestra un diálogo preguntando si el usuario está seguro de borrar el paciente de la base de datos.
Postcondiciones	Se añade, visualiza, modifica y borra un paciente de la base de datos. El borrado de un paciente provoca el borrado en cascada de las notas y tareas asociadas.
Excepciones	No se rellenan todos los campos obligatorios en el formulario de creación o actualización.

Tabla 7.14: CU 35 - 39

CU 40 - 42	Añadir, visualizar y eliminar direcciones de pacientes.
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador (Solo visualización)
Precondiciones	El usuario debe estar autenticado. Para eliminar una dirección de un paciente debe haber creada alguna en la base de datos.
Descripción	**Añadir dirección paciente:** Se presenta un formulario para rellenar con los datos de un paciente para añadirlo al sistema. **Visualizar direcciones de paciente:** Muestra una lista de las direcciones de un paciente, almacenadas en la base de datos. **Eliminar dirección paciente:** Se muestra un diálogo preguntando si el usuario está seguro de borrar la dirección del paciente de la base de datos.
Postcondiciones	Se añade, visualiza y borra la dirección de un paciente de la base de datos.
Excepciones	No se rellenan todos los campos obligatorios en el formulario de creación.

Tabla 7.15: CU 40 - 42

CU 43	Añadir nota
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador, Usuario de paciente
Precondiciones	El usuario debe estar autenticado. Debe haber creados pacientes en el ayuntamiento al que pertenece el actor que ejecuta el CU.
Descripción	Una vez elegido el paciente al que se le va a asignar una nota, se muestra un formulario con dos campos, uno con el cuerpo de la nota y otro con un selector para indicar su visibilidad: • Privado: Visible únicamente para los gerentes de ayuntamiento y los trabajadores. Solo se puede seleccionar por estos dos tipos de usuarios. • Público: Visible para los gerentes de ayuntamiento, los trabajadores y los usuarios de paciente, seleccionable por todos los actores que pueden ejecutar este caso de uso.
Postcondiciones	Debe haber creados pacientes en el ayuntamiento al que pertenece el actor que ejecuta el caso de uso
Excepciones	El usuario no rellena los campos obligatorios

Tabla 7.16: CU 43

CU 44	Visualizar notas
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador, Usuario de paciente
Precondiciones	El usuario debe estar autenticado. Debe haber creados pacientes en el ayuntamiento al que pertenece el actor que ejecuta el CU.
Descripción	El usuario que ejecuta el caso de uso debe elegir el paciente del que quiere visualizar las notas. Si el actor que ejecuta el caso de uso es un gerente del ayuntamiento o trabajador, se muestran todas las notas que tiene el paciente creadas. Si por el contrario, el actor que ejecuta el CU es un usuario de paciente, solo verá las notas que tienen visibilidad pública.
Postcondiciones	-
Excepciones	-

Tabla 7.17: CU 44

CU 45	Cambiar visibilidad de nota
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador
Precondiciones	El usuario debe estar autenticado Debe haber creadas notas asignadas al paciente que hemos seleccionado para visualizarlas.
Descripción	Una vez lanzado el caso de uso, aparece un diálogo que pide confirmación, en caso de aceptar se cambiará la visibilidad de la nota, pasando de privado a público o viceversa.
Postcondiciones	-
Excepciones	-

Tabla 7.18: CU 45

CU 46	Eliminar nota
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador, Usuario de paciente
Precondiciones	El usuario debe estar autenticado. Debe haber notas creadas y asignadas al paciente que hemos seleccionado para visualizarlas.
Descripción	Una vez elegido el paciente para ver las notas que tiene creadas, si el actor es un gerente o un trabajador puede ejecutar el CU con cualquier nota, sin embargo, si el actor a realizar el caso de uso es un usuario de paciente solo podrá borrar la nota si esta es de su autoría. Una vez ejecutado el caso de uso, se mostrará un diálogo pidiendo confirmación. Si esta es aceptada, la nota será borrada.
Postcondiciones	-
Excepciones	Si un usuario no autorizado para eliminar una nota lanza el caso de uso, aparece un aviso que indica que su operación no está permitida.

Tabla 7.19: CU 46

7.5.4 Gestión de tareas

En esta última sección se describen los casos de uso asociados a la gestión de tareas y horarios, tanto de pacientes como de trabajadores. Además se explican las tareas de gestión de los días o períodos no trabajados, estos son, los días de bajas, los asuntos propios y demás. En esta sección no se agrupa ningún caso de uso, debido a que cada uno tiene alguna particularidad que debe ser explicada.

CU 47	Visualizar horario por trabajador
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador
Precondiciones	El usuario debe estar autenticado Debe haber creados trabajadores en el ayuntamiento al que pertenece el actor ejecuta el CU.
Descripción	Si es un gerente el que lanza el caso de uso, elegirá el empleado de su ayuntamiento para visualizar el horario. Mientras que si es un trabajador el que lo ejecuta, solo podrá visualizar el suyo propio. Una vez indicado de quién se va a visualizar el horario se presenta un calendario en formato semanal. En él se muestran las tareas que están programadas para el tiempo que se está visualizando. Se puede navegar entre los períodos de las distintas vistas: semanas, meses o días.
Postcondiciones	-
Excepciones	-

Tabla 7.20: CU 47

CU 48	Visualizar horario por paciente
Actores que lo ejecutan	Gerente del ayuntamiento, Usuario de paciente
Precondiciones	El usuario debe estar autenticado. Debe haber creados pacientes en el ayuntamiento al que pertenece el actor que va a ejecutar el CU.
Descripción	Si es un gerente el que lanza el caso de uso, elegirá un paciente entre todos los almacenados en su ayuntamiento para visualizar el horario. Si por el contrario es un usuario de paciente, elegirá entre los pacientes que tiene asociados. Una vez indicado de quién se va a visualizar el horario se presenta un horario en formato semanal. En él se muestran las tareas programadas para el tiempo que se está visualizando. Se puede navegar entre los períodos de las distintas vistas: semanas, meses o días.
Precondiciones	-
Excepciones	-

Tabla 7.21: CU 48

CU 49	Crear tarea
Actores que lo ejecutan	Gerente del ayuntamiento
Precondiciones	El usuario debe estar autenticado Debe haber creados pacientes en el ayuntamiento al que pertenece el actor que va a ejecutar el CU.
Descripción	Se acciona seleccionando en el calendario de un paciente, el intervalo de tiempo que se va a dedicar a la tarea. Una vez seleccionado, se muestra un diálogo en la hay un formulario con varios campos. En el campo trabajador se habilitará un desplegable que, además de mostrar los trabajadores que hay libres para ese intervalo de tiempo, indica la carga de trabajo que tiene para el período que se está visualizando. Si lo deseamos, podemos añadir comentarios a la tarea.
Postcondiciones	Una vez creada la tarea el trabajador no estará disponible en tiempo que dure esta, por lo que no aparecerá en los formularios de creación de tareas nuevas.
Excepciones	No se rellenan todos los campos del formulario.

Tabla 7.22: CU 49

CU 50	Modificar tarea
Actores que lo ejecutan	Gerente del ayuntamiento
Precondiciones	El usuario debe estar autenticado. Debe haber creada una tarea para poder modificarla.
Descripción	Una vez elegida la tarea que se va a modificar, se habilita un formulario, en el que se permite modificar las horas de inicio y fin de la tarea, la fecha, el trabajador asignado a la tarea, el estado de la misma y los comentarios.
Postcondiciones	Los campos modificados actualizarán la tarea.
Excepciones	No se rellenan todos los campos del formulario.

Tabla 7.23: CU 50

CU 51	Marcar tarea como completada
Actores que lo ejecutan	Gerente del ayuntamiento, Trabajador
Precondiciones	El usuario debe estar autenticado. Debe haber creadas tareas en la semana que estemos visualizando del paciente seleccionado.
Descripción	Este caso de uso modifica el estado de las tareas que se encuentran en estado pendiente y las pasa a estado completado. Al accionarlo se muestra un diálogo pidiendo confirmación. Si la confirmación es aceptada la tarea cambia su estado a completada.
Postcondiciones	La tarea pasa a estado completado.
Excepciones	-

Tabla 7.24: CU 51

CU 52	Copiar horario semanal
Actores que lo ejecutan	Gerente del ayuntamiento
Precondiciones	El usuario debe estar autenticado. Debe haber creadas tareas en la semana que estemos visualizando del paciente seleccionado.
Descripción	Mientras se visualiza una semana en concreto del horario de un paciente, se puede ejecutar este caso de uso que muestra un diálogo. En él, se habilita un campo en el que se introduce el número de semanas, partiendo desde la visualizada; que se desea avanzar para copiar las tareas. Una vez confirmado, se realiza la copia.
Postcondiciones	Se crean nuevas tareas con la misma información pero trasladadas el número de semanas indicado en el díalogo.
Excepciones	No se rellena el campo o se introduce un valor menor que uno

Tabla 7.25: CU 52

CU 53	Añadir período no trabajado
Actores que lo ejecutan	Gerente del ayuntamiento
Precondiciones	El usuario debe estar autenticado. Debe haber creados trabajadores en el ayuntamiento al que pertenece el actor que ejecuta el CU.
Descripción	Se elige en un calendario el intervalo de tiempo que va a durar el período no trabajado, una vez elegido se mostrará un formulario que permite modificar las horas de inicio y fin elegidas, etiquetar el período con la causa que lo justifica e indicar una descripción si se precisa.
Postcondiciones	El período no trabajado se guarda en la base de datos.
Excepciones	No se rellenan los campos obligatorios.

Tabla 7.26: CU 53

CU 54	Visualizar períodos no trabajados
Actores que lo ejecutan	Gerente del ayuntamiento
Precondiciones	El usuario debe estar autenticado. Debe haber creados trabajadores en el ayuntamiento al que pertenece el actor que va a ejecutar el CU.
Descripción	El gerente elige un trabajador entre todos los del ayuntamiento. Una vez escogido se muestra un calendario en el que se exponen los períodos no trabajados del empleado.
Postcondiciones	-
Excepciones	-

Tabla 7.27: CU 54

CU 55	Modificar período no trabajado
Actores que lo ejecutan	Gerente del ayuntamiento
Precondiciones	El usuario debe estar autenticado. Debe haber creado un período no trabajado asociado a un trabajador del ayuntamiento al que pertenece el actor que ejecuta el CU.
Descripción	Se muestra un formulario en el que se puede modificar las fechas de inicio y fin del período, el tipo de período no trabajado y el comentario asociado.
Postcondiciones	Los cambios realizados son almacenados en la base de datos.
Excepciones	Se dejan en blanco campos obligatorios.

Tabla 7.28: CU 55

7.6 Modelo de casos de uso

En la siguiente sección se muestran y comentan los diagramas de casos de uso. Se han dividido los casos de uso en cuatro diagramas, que se corresponden con la agregación de casos de uso que tienen cierta relación lógica entre ellos.

En el primer diagrama (Fig. 7.2), se muestran 14 casos de uso, todos ellos relacionados con la gestión de usuarios. Todos los casos de uso de este diagrama, exceptuando el de autenticación (CU 1), son realizados únicamente por el administrador y el gerente del ayuntamiento.

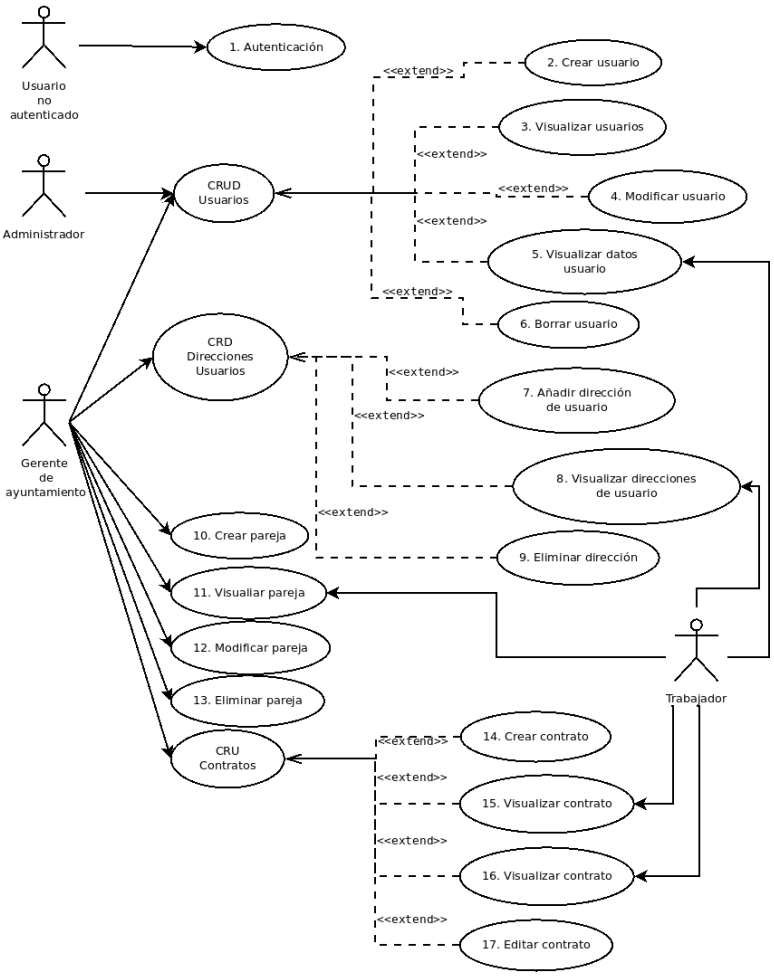


Figura 7.2: Casos de uso: gestión de usuarios

En la figura 7.3 se muestra el diagrama de los casos de uso relativos a la gestión de ayuntamientos. Este grupo es el más grande con 17 casos de uso (CU 15 - CU 31) y todas son ejecutados únicamente por el administrador y el gerente del ayuntamiento.

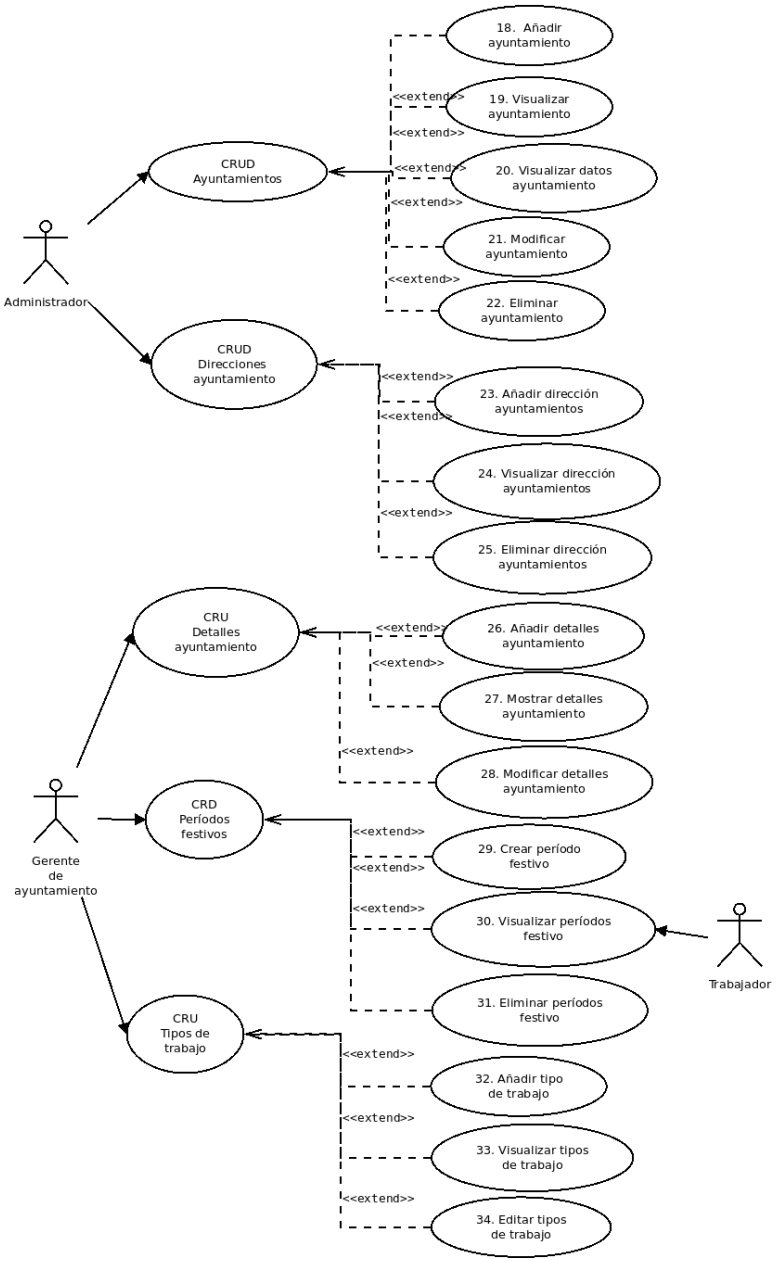


Figura 7.3: Casos de uso: gestión de ayuntamientos

La figura 7.4 es el tercer diagrama y contiene 12 casos de uso (CU 32 - CU 43). En él que se exponen los casos de uso de gestión de pacientes y sus notas asignadas. En este diagrama se puede ver como la gestión de los pacientes es realizada por el gerente del ayuntamiento, mientras que los trabajadores y usuarios de paciente solo pueden ejecutar casos de uso de gestión de notas.

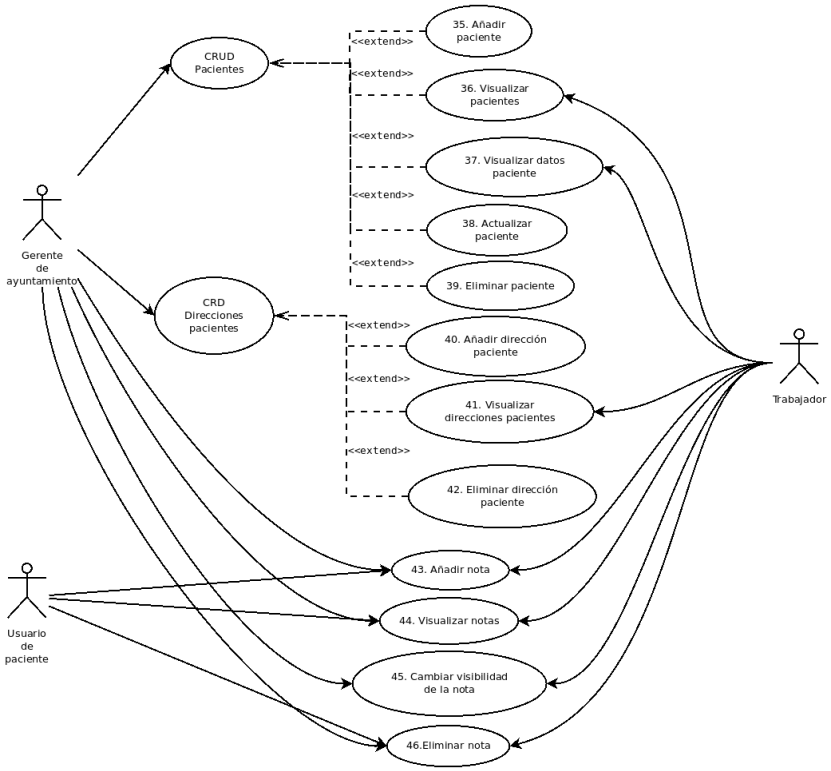


Figura 7.4: Casos de uso: gestión de pacientes

En el último diagrama (Fig. 7.5), se muestran todos los casos de uso relacionados con la gestión de tareas y períodos no trabajados. Este diagrama contiene 9 casos de uso (CU 43 - CU 52).

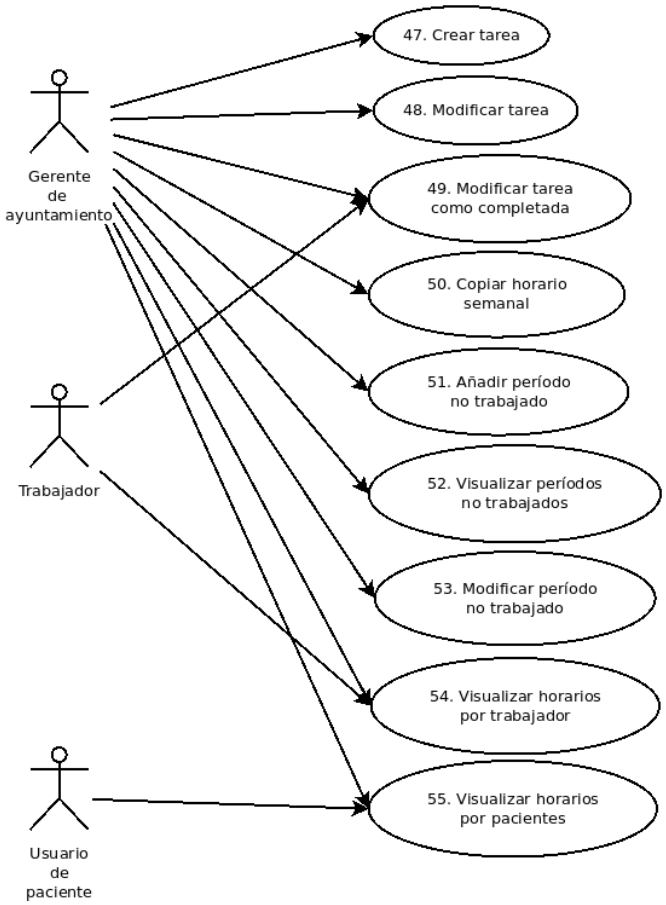


Figura 7.5: Casos de uso: gestión de tareas

7.7 Prototipado de interfaz

Como última parte del análisis de requisitos, se crean unos prototipos de la interfaz de usuario. Con ellos se pretende tener una idea básica de la interfaz y de como se desarrollarán los casos de uso.

En las dos primeras figuras (Fig. 7.6 y Fig. 7.7) se ideó como serían los casos de uso número 19 (Visualizar Ayuntamientos, Tabla 7.9) y número 3 (Visualizar usuarios, Tabla 7.2), respectivamente. En los prototipos se presenta una barra lateral para intercambiar entre la gestión de usuarios y ayuntamientos. En la tabla hay presentes unos botones que permiten acceder otros casos de uso como por ejemplo la edición.

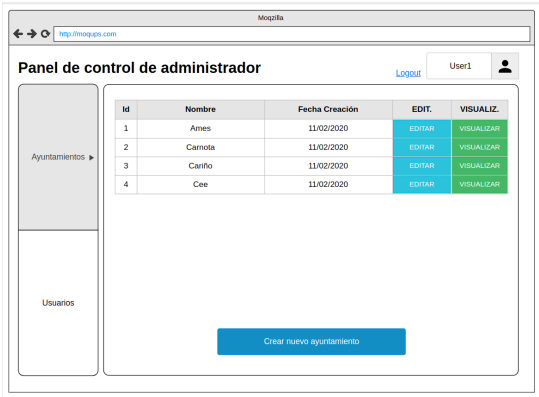


Figura 7.6: Prototipo de la interfaz: Gestión ayuntamientos

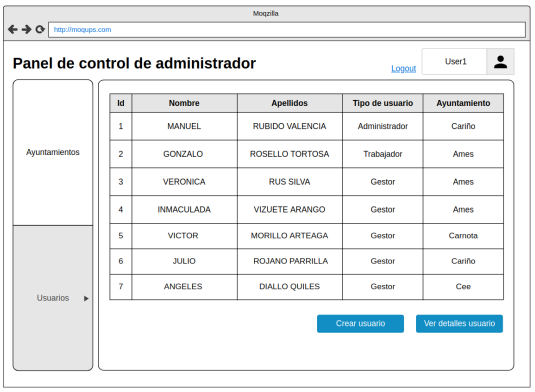


Figura 7.7: Prototipo de la interfaz: Gestión de usuarios

El prototipo del caso de uso 48 (Visualizar horario por trabajador, fig. 7.26), se compone de dos pantallas. La primera es una página de selección del paciente, en la que se mostrarán los pacientes que están añadidos para visualizar su horario. La segunda pantalla muestra un horario semanal en el que aparecerán las tareas creadas para un paciente, en las que se indicaría quién la realizaría, los comentarios añadidos y una marca en caso de que esté completada.

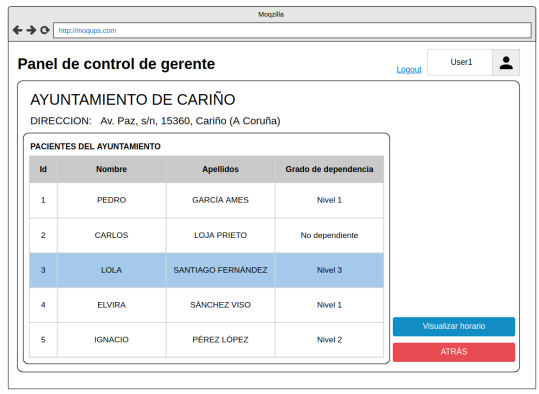


Figura 7.8: Prototipo de la interfaz: Gestión de horarios (Pantalla 1)

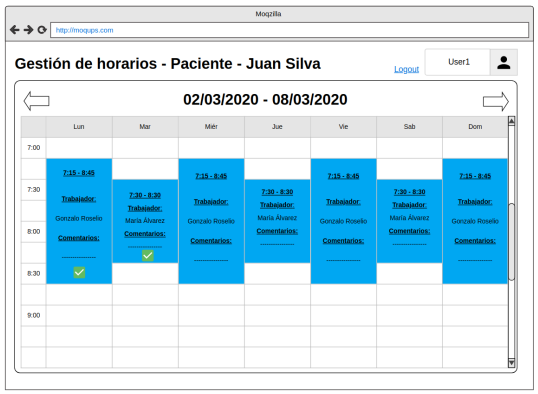


Figura 7.9: Prototipo de la interfaz: Gestión de horarios (Pantalla 2)

Aunque a medida que se fue realizando el proyecto se fueron haciendo algunas modificaciones, intentando mejorar la usabilidad, la creación de estos prototipos sirvió como base para la interfaz.

Capítulo 8

Diseño

Este capítulo incluye la documentación de diseño del sistema desarrollado en este trabajo fin de grado. En primer lugar se realizada una pequeña enumeración de los principales patrones de diseño utilizados, para a continuación presentar la arquitectura general y la descripción de cada uno de los subsistemas en los que se descompone.

8.1 Resumen de principales patrones usados

- **Patrones de arquitectura**
  - **Arquitectura en capas** [29]: Utilizado para favorecer el desacoplamiento entre las distintas partes que componen un sistema software, en este caso, siguiendo una estructura cliente-servidor. Este diseño en capas evita dependencias en el sistema y favorece el mantenimiento y las actualizaciones del mismo.
- **Patrones de diseño**
  - **Inversión de control e Inyección de dependencias** [30]: Tanto la inversión de control (principio de ingeniería) como la inyección de dependencias (patrón de desarrollo) son utilizados por *Spring*. El principio, indica que el control de objetos o partes de un programa se transfiere a un contenedor o marco, permitiendo desacoplar la ejecución de su implementación y favorecer la modularidad de un sistema, entre otras virtudes. El patrón de inyección de dependencias, que implementa el principio descrito, invierte la configuración de las dependencias de un objeto.
  - **Patrón fachada** [31]: El patrón fachada consigue reducir la complejidad cuando se divide una aplicación en subsistemas. El patrón sugiere proporcionar una interfaz simple para un subsistema más complejo en su interior, siendo el punto de

acceso de cada subsistema la fachada. Este patrón favorece el desacoplamiento e independencia de los subsistemas que conforman una aplicación.

- **Patrón DAO** [32]: El patrón DAO (*Data Access Object*) indica el uso de interfaces comunes entre la aplicación y el dispositivo de almacenamiento que se esté utilizando, independientemente de cual sea. Este patrón permite desacoplar la capa de acceso a datos del dispositivo de almacenamiento que se esté utilizando.
- **Patrón DTO**: El patrón DTO (*Data Transfer Object*) utiliza objetos planos (*POJO*), de solo lectura y fácilmente serializables, cuya finalidad es obtener un objeto simple para que sea sencillo de enviar por la red.
- **Patrón Front Controller**: Este patrón es implementado por *Spring* en la capa de servicio, con la intención de lograr la flexibilidad y reusabilidad del código. Se utiliza para recibir las peticiones http y delegarlas sobre los diferentes controladores.

8.2 Arquitectura general

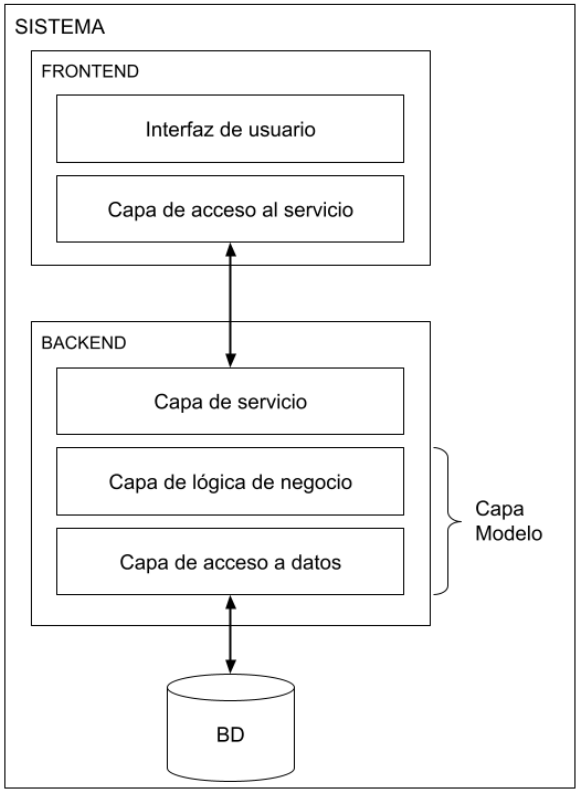


Figura 8.1: Diagrama de arquitectura general

La aplicación se ha dividido en dos partes claramente diferenciadas, como se puede ver en

la figura 8.1, denominadas *backend* y *frontend*.

La parte *backend* o parte servidor, se divide a su vez en tres capas. Comenzando en modo ascendente desde la base de datos, nos encontramos con la primera capa de acceso a datos. Esta interactúa con la base de datos, consultando la información almacenada en ella cuando es requerida. La siguiente capa es la de lógica de negocio, en la que se implementan los casos de uso de la aplicación, que usan la capa de acceso a datos para la búsqueda y persistencia de los datos asociados. La última capa del subsistema *backend* es la capa de servicio, en la que se proporciona una API REST que proporciona un acceso remoto a la funcionalidad de la capa de lógica de negocio.

El subsistema *frontend* es el que el usuario utiliza para comunicarse con el sistema. Siguiendo una dirección ascendente, primero se encuentra la capa de acceso al servicio, que es la encargada de comunicarse con la API REST que expone el subsistema *backend*, y de tratar los datos que se requieren o solicitan del sistema. Por último, la capa interfaz de usuario es la que muestra la información al usuario y con la que este interactúa. En los siguientes apartados se documenta el diseño de cada uno de estos subsistemas.

8.3 Subsistema *Backend*

8.3.1 Objetivos

El subsistema *backend* es el lado servidor de la aplicación y es donde se almacena la mayor parte de la lógica de todo el sistema. Se ha utilizado *SpringBoot* [8] e *Hibernate* [9] para facilitar el desarrollo de la aplicación y hacer más sencillo el mantenimiento de la misma. En esta parte del sistema se hace uso de los patrones DAO, fachada e inversión de dependencias, entre otros; para favorecer la independencia entre capas, fomentando el desacoplamiento de las mismas.

8.3.2 Arquitectura

Como se puede observar en la figura 8.2, el paquete *es.udc.gasd.backend* (*gasd* son las siglas de Gestión de Asistencia Social a Domicilio) se divide a su vez en dos: *model* y *rest*, que se relacionan entre sí.

En el primero de ellos se encuentra el paquete llamado *entities* donde se localiza la capa de acceso a datos. En él se definen las clases que aplican el patrón DAO ya que en este paquete se utilizan los *Data Access Objects*, objetos que *hibernate* usa para mapear objetos java a tuplas de datos en la base de datos. Por otro lado, en *exceptions* se incluyen las excepciones que se utilizan en la capa de lógica de negocio, incluída en el paquete *services*. Este último se apoya en las utilidades creadas en *entities* y las excepciones de *exceptions* para trabajar y transformar los datos que se leen o se van a introducir en la base de datos.

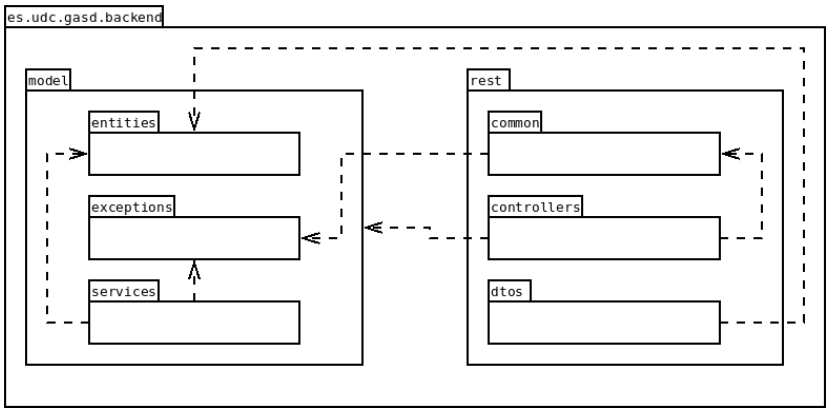


Figura 8.2: *Backend*: Diagrama de paquetes

En el paquete *rest* se incluyen los paquetes que servirán como acceso desde el exterior a las funcionalidades que expone el servicio. El primero de los paquetes que contiene es *common*, donde se reúnen las clases comunes que utilizan todos los controladores (situados en *controllers*). El paquete *dtos*, donde se aplica el patrón homónimo, contiene los objetos utilizados para enviar los datos por la red y sus clases auxiliares para trasformarlos.

En el paquete *controllers* se aplica el patrón *FrontController* que *Spring* implementa para dirigir las peticiones HTTP a las diferentes clases controladro, en función de las anotaciones indicadas en las clases almacenadas en él. Mientras que en el paquete *services* se utiliza el patrón fachada, exponiendo un único punto de acceso a toda una parte del sistema.

8.3.3 Modelo del dominio

Diagrama de entidades

En la figura 8.3 se puede ver como se relacionan las clases persistentes del sistema. A continuación se detallan cada una de ellas:

- **Council (Ayuntamiento)**: Entidad que indica el ayuntamiento, identificando a los gerentes, trabajadores y pacientes que pertenecen al mismo.
- **CouncilDetails (Detalles del ayuntamiento)**: Conjunto de información del ayuntamiento que es válida durante el tiempo especificado. Un mismo ayuntamiento puede tener varios grupos de detalles pero cada grupo de detalles puede hacer referencia a un único ayuntamiento.
- **HolidayPeriod (Período festivo)**: Agrupaciones de días catalogados como no laborables, pertenecientes al conjunto de información del ayuntamiento. Los períodos festivos deben estar entre los límites especificados del conjunto de detalles al que pertenecen.

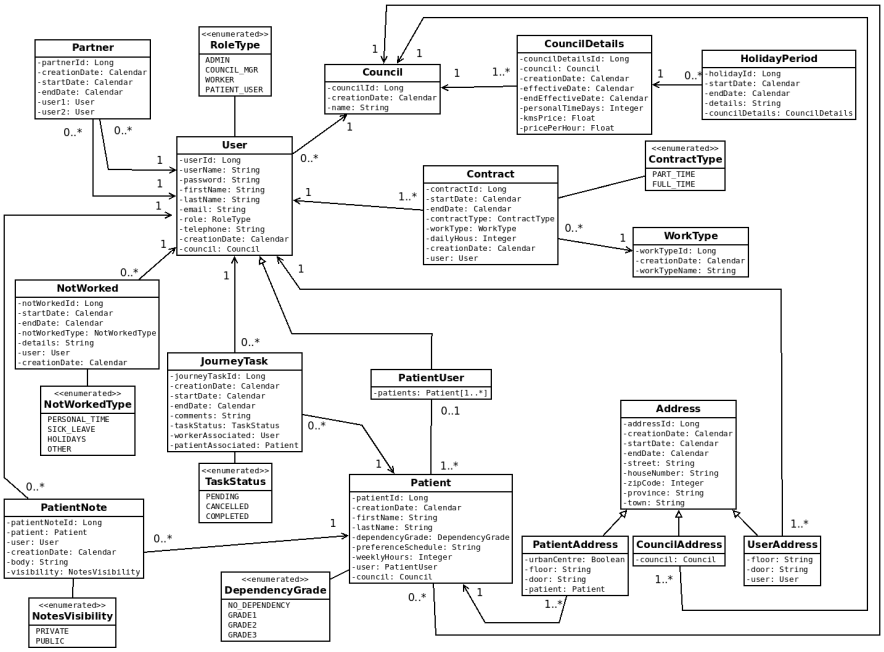


Figura 8.3: *Backend*: Diagrama de clases persistentes

- **User (Usuario)**: Información de un usuario del sistema.
- **RoleType (Rol)**: Enumerado utilizado en los usuarios que indica el papel que desempeñan en el sistema.
- **PatientUser (Usuario de paciente)**: Clase que extiende la información de un usuario, utilizada por un paciente o su tutor.
- **Partner (Pareja)**: Agrupación entre dos trabajadores que es válida durante el tiempo especificado.
- **Contract (Contrato)**: Contrato de trabajo de los usuarios del sistema, a excepción de los usuarios de paciente. Un mismo usuario puede tener varios contratos, pero cada contrato puede hacer referencia a un único usuario. En él se indican las fechas de inicio y fin, la de creación, el tipo de trabajo que va a desempeñar, el total de horas de trabajo diarias, entre otros atributos.
- **ContractType (Tipo de contrato)**: Enumerado utilizado para los contratos que indica si es de media jornada o jornada completa.
- **WorkType (Tipo de trabajo)**: Clase utilizada en los contratos que indica el tipo de trabajo que va a realizar un trabajador o gerente.

- **Patient (Paciente)**: Información de un consumidor del servicio ofrecido por el ayuntamiento.
- **DependencyGrade (Grado de dependencia)**: Enumerado que indica el grado de dependencia que tiene un paciente, utilizado en la entidad Patient.
- **PatientNote (Nota de un paciente)**: Información de una nota que un usuario registra en relación a un paciente.
- **NotesVisibility (Visibilidad de nota)**: Enumerado que indica la visibilidad de una nota de paciente.
- **JourneyTask (Tarea)**: Datos de un servicio a realizar por un trabajador a un paciente.
- **TaskStatus (Estado de tarea)**: Enumerado utilizado en las tareas, que indica el estado en el que se encuentran.
- **NotWorked (No trabajado)**: Entidad que mantiene la información de los días que un trabajador no ha realizado sus servicios.
- **Address (Dirección)**: Atributos comunes que todas las direcciones deben tener. Tiene fechas de inicio y fin que indican su período de validez. Esta clase es extendida cuando se necesitan atributos más específicos.
- **CouncilAddress (Dirección del ayuntamiento)**: Atributos específicos de la dirección de un ayuntamiento. Esta clase extiende la clase Address.
- **UserAddress (Dirección de usuario)**: Extensión de la clase Address que incluye la información de la dirección de un usuario del sistema.
- **PatientAddress (Dirección de un paciente)**: Atributos específicos del domicilio de un paciente. Esta clase extiende la clase Address.

Realaciones entre entidades y aspectos de implementación

Para mapear las tablas de la base de datos y sus relaciones al modelo de objetos se ha utilizado *Hibernate*, una implementación del estándar *Java Persistence API* (JPA), facilitando la definición de claves primarias y foráneas, las tablas y sus relaciones además del uso de enumerados.

Las claves primarias son autogeneradas por MySQL y conjuntamente se utilizan las anotaciones *@Id* y *@GeneratedValue* de *Hibernate*

Para la relación entre tablas y el uso de claves foráneas se utiliza las notaciones *@OneToMany* y *@ManyToOne* en función de la cardinalidad de la relación, además de la notación

@JoinColumn para indicar qué columna hace referencia a la clave foránea utilizada en las relaciones muchos a uno.

Cuando es necesaria la utilización de un enumerado se usa la notación *@Enumerated*.

Si se utiliza la notación *@ManyToOne*, la política que *Hibernate* utiliza por defecto es *EAGER*, esto quiere decir que si se solicita una entidad, se traerían las clases relacionadas a memoria en caso de que no estuviesen presentes. Esto podría provocar reacciones en cadena resultasen en una carga masiva de datos al consultar cierta tabla. En este caso, la política EAGER sería contraproducente, ya que en muchas ocasiones se traerían a memoria datos que no necesitamos utilizar, provocando un mayor carga totalmente innecesaria. Para evitarlo, se ha indicado expresamente que la política a utilizar sea *LAZY*, de esta manera, solo los datos que se necesiten serán traídos a la memoria.

Las tablas, representadas en la figura 8.4, se han creado de forma manual utilizando un fichero SQL debido a que esta aproximación es la más habitual en la industria. En la figura se puede observar como se relacionan las tablas entre sí.

8.3.4 Capa de acceso a datos

Para la capa de acceso a datos se ha utilizado la interfaz facilitada por *Spring-Boot* llamada *PatgingAndSortingRepository*. Esta interfaz nos permite realizar consultas y algunos métodos básicos que extienden cada uno de los DAOs (*Data Access Object*). Cada entidad persistente tiene una interfaz DAO que es utilizada para la comunicación con la base de datos.

La arquitectura que sigue la capa de acceso a datos es la mostrada en la figura 8.5.

Debido a que la interfaz facilitada por *Spring-Boot* solo proporciona métodos básicos, se han tenido que crear algunas interfaces personalizadas y sus correspondientes implementaciones para realizar las consultas que se necesitaban. De esta manera algunas interfaces DAO extienden tanto de *PatgingAndSortingRepository* como de su interfaz personalizada.

8.3.5 Capa de lógica de negocio

Para implementar la capa de negocio, se han creado los servicios agrupando casos de uso lógicamente relacionados. Se ha optado por seguir la división realizada en los casos de uso (explicado en el capítulo 7) y desarrollar 4 servicios: *CouncilService*, *PatientService*, *TaskService*, *UserService*.

Para implementar esta capa se ha utilizado la notación *@Service* y *@Transactional*. Esta capa es la encargada de utilizar la capa modelo y sus DAOs explicados anteriormente. Utilizando las etiquetas *@Autowired*, *Spring* realiza la inyección de dependencias, creando los objetos e inyectando las referencias de los objetos de los que dependen.

Así se pueden trabajar con los datos facilitados por la capa modelo y posteriormente proporcionárselos a la capa de servicios REST del *backend*.

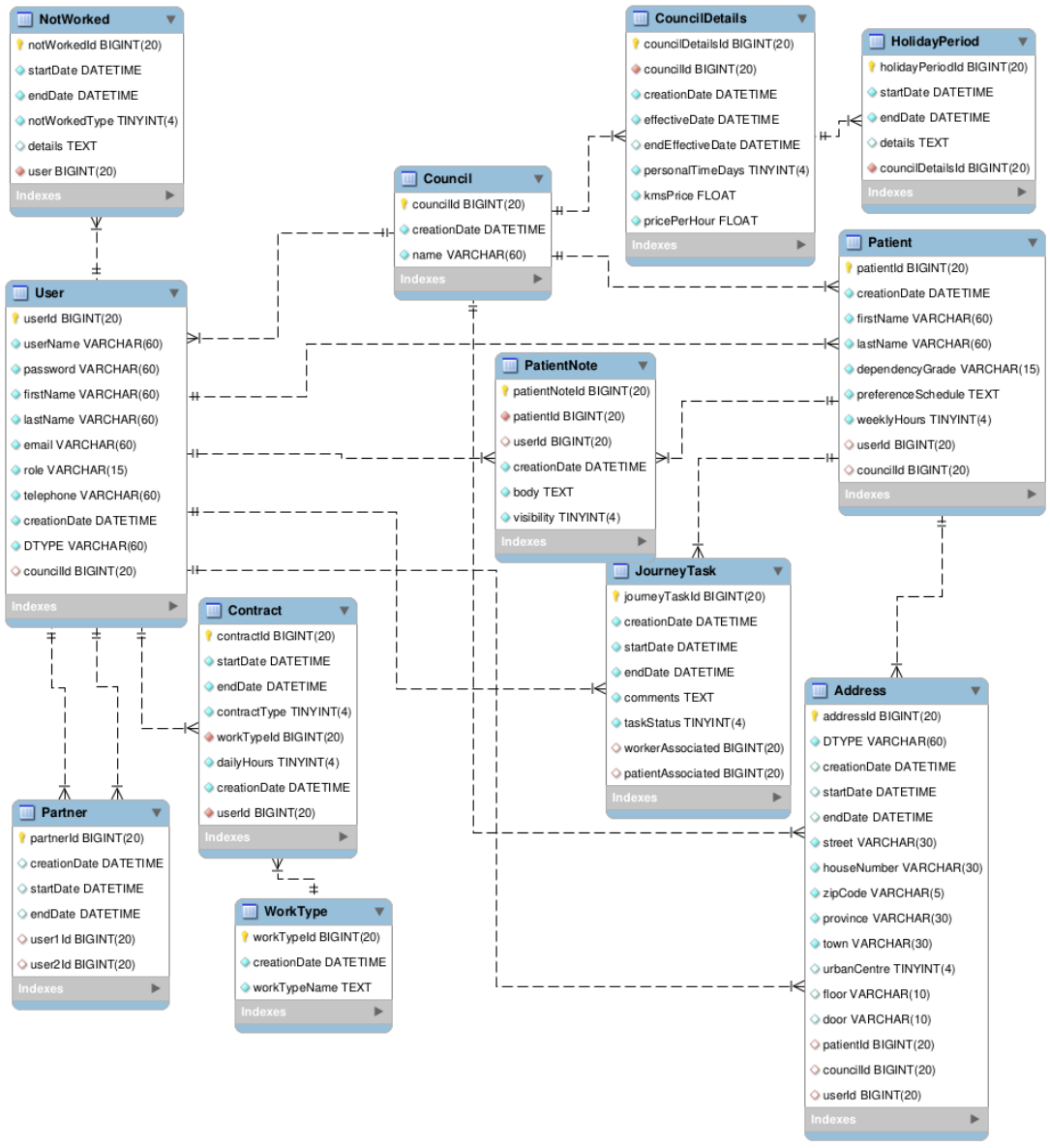


Figura 8.4: *Backend*: Diagrama relacional de tablas

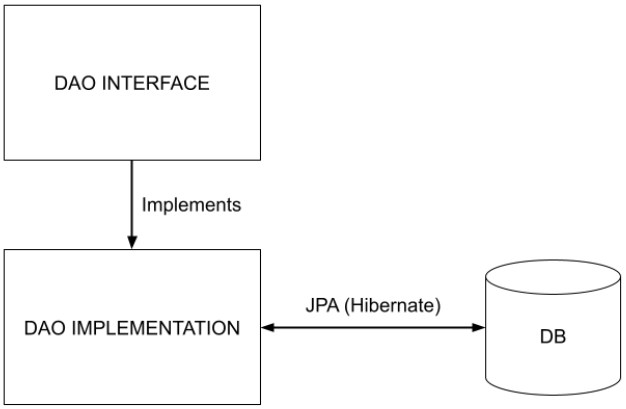


Figura 8.5: *Backend*: Arquitectura DAO

A continuación se detallan cada uno de los servicios:

CouncilService

Operaciones relacionadas con la gestión de ayuntamientos y sus datos asociados.

- **Operaciones *CRUD* de la entidad Council**: Operaciones de creación y búsqueda, tanto de todas las entidades, como de una en concreto dependiendo del identificador. Además de tareas de actualización y borrado de la entidad que representa a los ayuntamientos.

- **Operaciones *CRD* de la entidad CouncilAddress**: Métodos de creación, búsqueda y borrado de las direcciones de un ayuntamiento. Existen distintos tipos de búsqueda: todas las direcciones de un mismo ayuntamiento, la dirección que es válida en el instante que se realiza la búsqueda o búsqueda de una dirección en concreto por identificador.

- **Operaciones *CRUD* de la entidad CouncilDetails**: Métodos de creación, lectura, actualización y borrado de la entidad que representa el conjunto de detalles de un ayuntamiento.

- **Operaciones *CRD* de la entidad HolidayPeriod**: Métodos de creación, búsqueda y borrado de períodos festivos.

- **Operaciones *CRD* de la entidad WorkType**: Operaciones de creación, búsqueda y borrado de los tipos de trabajo.

PatientService

- **Operaciones *CRU* de la entidad Patient**: Métodos de creación, búsqueda y actualización de la entidad de pacientes. Existen tres tipos de búsqueda: búsqueda de todos los pacientes de un ayuntamiento, todos los pacientes asignados a un usuario con el rol de Usuario de paciente y una búsqueda de pacientes por el identificador.

- **Operaciones *CRD* de la entidad PatientAddress**: Operaciones de creación, búsqueda y borrado de direcciones de un paciente. Hay implementados tres métodos de búsqueda, por identificador, todas las direcciones de un paciente y la dirección actual de un paciente.

- **Operaciones *CRD* de la entidad PatientNote**: Operaciones de creación, lectura y borrado de las notas de un paciente. Hay implementados tres tipos de búsqueda de notas: todas las notas de un paciente, solo notas públicas y búsqueda por identificador.

- **PatientNote changeVisibility(Long patientNoteId, NotesVisibility newVisibility)**: Cambio del parámetro visibilidad de una nota de paciente.

TaskService

- **Long createTask(JourneyTask JourneyTask)**: Creación de una tarea, devuelve el identificador de la misma.

- **JourneyTask updateTask(Long journeyTaskId, Calendar startDate, Calendar endDate, String comments, TaskStatus taskStatus, User workerAssociated, Patient patientAssociated)**: Actualización de una tarea, devuelve la tarea actualizada.

- **JourneyTask updateTaskSatus(Long journeyTaskId, TaskStatus newTaskStatus)**: Actualización del estado de una tarea, la devuelve con el estado actualizado.

- **List<JourneyTask> findTasksByUserBetweenDates(Long userId, Calendar startDate, Calendar endDate)**: Búsqueda de tareas de un usuario comprendidas entre las fechas elegidas, devuelve las tareas encontradas.

- **List<JourneyTask> findTasksByPatientBetweenDates(Long patientId, Calendar startDate, Calendar endDate)**: Búsqueda de tareas de un paciente comprendidas entre las fechas elegidas, devuelve las tareas encontradas.

- **JourneyTask findTask(Long journeyTaskId)**: Búsqueda de tarea por identificador, devuelve la tarea encontrada.

- **Map<User, Long> findFreeWorkers(Long councilId, Calendar startDate, Calendar endDate, Calendar startScope, Calendar endScope)**: Busca trabajadores libres de tareas en las fechas indicadas. Devuelve un mapa con los trabajadores libres con el número de horas asignadas durante el rango indicado con startScope y endScope.

- **Operaciones *CRU* de la entidad NotWorked**: Operaciones de creación, búsqueda y actualización de los períodos no trabajados. Se implementan dos tipos de búsqueda, por usuario en un rango de fechas y búsqueda por identificador.

UserService

- **Operaciones *CRUD* de la entidad User**: Métodos de creación, búsqueda, actualización y borrado de usuarios. Hay tres tipos de búsqueda: por ayuntamiento, por tipo de rol y ayuntamiento y por identificador.

- **Operaciones *CRD* de la entidad UserAddress**: Métodos de creación, búsqueda y borrado de las direcciones de los usuarios. Hay tres tipos de consulta: búsqueda de todas las direcciones de un usuario, búsqueda de la dirección vigente y búsqueda por identificador.

- **Operaciones *CRUD* de la entidad Partner**: Operaciones de creación, búsqueda, actualización y borrado de parejas de trabajadores. Se implementan cuatro tipos de búsqueda: por identificador, todas las parejas, todas las parejas por ayuntamiento y todas las parejas de un trabajador.

- **List<User> findWorkersWithNoPartner(Long councilId)**: Busqueda de trabajadores sin pareja, se devuelve la lista de usuarios.

- **Operaciones *CRUD* de la entidad Contract**: Operaciones de creación, búsqueda y actualización de los contratos. Se pueden realizar búsquedas por usuario, por identificador y el contrato vigente de un usuario.

- **Operaciones de *CRUD* de la entidad PatientUser**: Operaciones de creación y búsqueda de usuarios de paciente.

8.3.6 Capa servicios

La capa servicios es la encargada de gestionar las peticiones que le llegan a la API REST que expone. En concreto, esta aplicación publica cuatro *endpoints*, agrupando las operaciones como se ha realizado en la capa inmediatamente inferior.

Para facilitar la creación de cada uno de los *endpoints*, *Spring* permite utilizar las anotaciones *@RestController*. Para gestionar las peticiones *POST*, *PUT*, *DELETE* y *GET*, el framework también proporciona las anotaciones *@PostMapping*, *@PutMapping*, *@DeleteMapping* y *@GetMapping*, respectivamente.

La API, descrita en el apéndice A.1, ha sido diseñada siguiendo una aproximación *RESTful*, evitando mantener estado entre peticiones, orientando las peticiones a recursos, haciendo *overloading* del método POST cuando se necesita realizar una acciónc uya semántica no tiene correspondencia con los verbos HTTP para recursos *RESTful* y utilizando los códigos HTTP más afines para cada excepción.

En cada uno de los controladores creados se inyectan los servicios expuestos utilizando la anotación *@Autowired* que, como en capas anteriores, se encarga de inyectar las dependencias.

Para gestionar las excepciones en los controladores, *Spring* proporciona la anotación *@ExceptionHandler* con la que crear manejadores que se encargan de capturar una excepción proveniente del servicio y crear una respuesta HTTP adecuada. Por cada excepción se ha configurado un manejador que envía una respuesta con un código de error HTTP y una descripción. Esta descripción está disponible en tres idiomas distintos: castellano, gallego e inglés y dependerá del idioma introducido en las cabeceras de la petición. Esto es posible gracias al método *getMessage* de la clase *MessageSource* proporcionada por *Spring*, que haría sencilla la incorporación de más idiomas en caso de que fuese requerido.

Excepción	**Cód. HTTP**	**Descripción**
InstanceNotFoundException	404	No se encuentra el recurso al que se está accediendo.
DuplicateInstanceException	400	Se intenta crear o modificar un recurso con un identificador duplicado.
PermissionException	403	No está permitida la acción para el usuario que la realiza
NoCurrentAvaliableException	404	No hay un recurso que incluya entre sus límites la fecha del momento de la petición.
DifferentCouncilException	400	Recursos no compatibles entre sí por pertenecer a distintos ayuntamientos.
UserNotAWorkerException	400	El usuario que se está proporcionando no es un trabajador.
HolidayOutOfDetailsBoundsException	400	El período festivo está fuera del rango de fechas del grupo de detalles del ayuntamiento.
IncorrectLoginException	404	Usuario o contraseña erróneos.
IncorrectPasswordException	404	Contraseña errónea.

Tabla 8.1: Excepciones

Cuando se recibe una petición, antes de llegar al controlador, es tratada por un filtro proporcionado por *Spring* (*SecurityConfig*) que permite o rechaza la petición dependiendo del

rol que tiene el usuario que realiza la petición. Para realizar estas comprobaciones, el usuario debe pasar por el proceso de autenticación utilizando sus credenciales con el fin de obtener el *token JWT (JSON Web Token)*. Como se puede ver en la figura 8.6, una vez autenticado el usuario, el *token JWT* se intercambia cada vez que se realiza una petición para mantener la sesión.

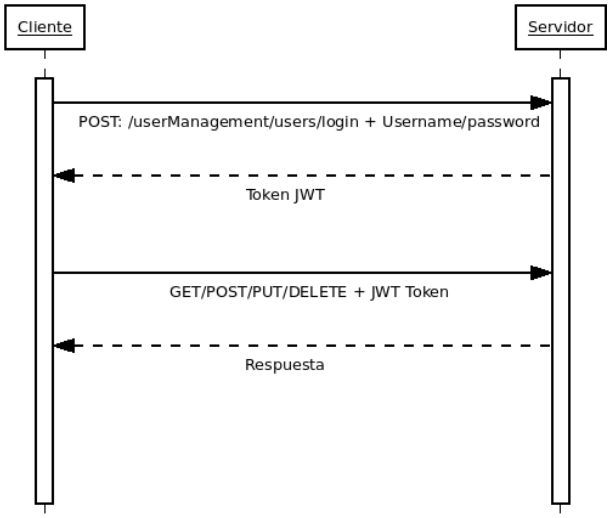


Figura 8.6: *Backend*: Proceso de autenticación JWT

Este *token* es utilizado para comprobar la lista de control de acceso, que rechaza las peticiones por defecto y aceptan las detalladas en la tabla 8.2.

Tabla 8.2: Control de acceso a endpoints

Método	**Endpoint**	**Autorizados**
POST	/userManagement/users/login	Todos
POST	/userManagement/users/loginFromService-Token	Todos
POST	/userManagement/users/id/selfUpdate	Autenticados
POST	/userManagement/users/*/changePassword	Autenticados
POST	/userManagement/users	Administrador, Gerente de ayuntamiento
GET	/userManagement/users	Administrador, Gerente de ayuntamiento
GET	/userManagement/users/*/addresses	Administrador, Gerente de ayuntamiento, Trabajador, Usuario de paciente
GET	/userManagement/users/*/addresses/*	Administrador, Gerente de ayuntamiento, Trabajador, Usuario de paciente
POST	/userManagement/users/*/addresses	Administrador, Gerente de ayuntamiento, Trabajador, Usuario de paciente
DELETE	/userManagement/users/*/addresses/*	Administrador, Gerente de ayuntamiento, Trabajador, Usuario de paciente
PUT	/userManagement/users/*	Administrador, Gerente de ayuntamiento
GET	/userManagement/users/*	Administrador, Gerente de ayuntamiento, Trabajador, Usuario de paciente

Continúa en la página siguiente...

Tabla 8.2 – *Continuación de la página anterior*

Método	Endpoint	Autorizados
DELETE	/userManagement/users/*	Administrador, Gerente de ayuntamiento
POST	/userManagement/partners	Gerente de ayuntamiento
GET	/userManagement/partners	Gerente de ayuntamiento, Trabajador
GET	/userManagement/partners/*	Gerente de ayuntamiento
PUT	/userManagement/partners/*	Gerente de ayuntamiento
DELETE	/userManagement/partners/*	Gerente de ayuntamiento
POST	/userManagement/users/*/contracts	Gerente de ayuntamiento
GET	/userManagement/users/*/contracts	Gerente de ayuntamiento, Trabajador
GET	/userManagement/users/*/contracts/*	Gerente de ayuntamiento, Trabajador
PUT	/userManagement/users/*/contracts/*	Gerente de ayuntamiento
POST	/councilManagement/councils	Administrador
GET	/councilManagement/councils	Administrador
GET	/councilManagement/councils/*	Administrador
DELETE	/councilManagement/councils/*	Administrador
PUT	/councilManagement/councils/*	Administrador
GET	/councilManagement/councils/*/addresses	Administrador
GET	/councilManagement/councils/*/addresses/*	Administrador
POST	/councilManagement/councils/*/addresses	Administrador
DELETE	/councilManagement/councils/*/addresses/*	Administrador
PUT	/councilManagement/councils/*/addresses/*	Administrador
POST	/councilManagement/councils/*/details	Gerente de ayuntamiento
GET	/councilManagement/councils/*/details	Gerente de ayuntamiento
DELETE	/councilManagement/councils/*/details/*	Gerente de ayuntamiento
PUT	/councilManagement/councils/*/details/*	Gerente de ayuntamiento
POST	/councilManagement/councils/*/details/*/holidayPeriods	Gerente de ayuntamiento
GET	/councilManagement/councils/*/details/*/holidayPeriods	Gerente de ayuntamiento
DELETE	/councilManagement/councils/*/details/*/holidayPeriods/*	Gerente de ayuntamiento
POST	/councilManagement/workTypes	Gerente de ayuntamiento
GET	/councilManagement/workTypes	Gerente de ayuntamiento, Trabajador
PUT	/councilManagement/workTypes/*	Gerente de ayuntamiento
POST	/patientManagement/patients	Gerente de ayuntamiento
GET	/patientManagement/patients	Gerente de ayuntamiento, Trabajador, Usuario de paciente
PUT	/patientManagement/patients/*	Gerente de ayuntamiento
GET	/patientManagement/patients/*	Gerente de ayuntamiento, Usuario de paciente, Trabajador
GET	/patientManagement/patients/*/addresses	Gerente de ayuntamiento, Trabajador
POST	/patientManagement/patients/*/addresses	Gerente de ayuntamiento
DELETE	/patientManagement/patients/*/addresses/*	Gerente de ayuntamiento
POST	/patientManagement/patients/*/patientNotes	Gerente de ayuntamiento, Trabajador, Usuario de paciente
GET	/patientManagement/patients/*/patientNotes	Gerente de ayuntamiento, Trabajador, Usuario de paciente
DELETE	/patientManagement/patients/*/patientNotes/*	Gerente de ayuntamiento, Trabajador, Usuario de paciente

Continúa en la página siguiente...

Tabla 8.2 – *Continuación de la página anterior*

Método	Endpoint	Autorizados
POST	/patientManagement/patients/*/patientNotes/*/changeVisibility	Gerente de ayuntamiento, Trabajador
GET	/tasksManagement/tasks	Gerente de ayuntamiento, Usuario de paciente, Trabajador
GET	/tasksManagement/tasks/*	Gerente de ayuntamiento, Usuario de paciente, Trabajador
POST	/tasksManagement/tasks	Gerente de ayuntamiento
POST	/tasksManagement/tasks/*/taskStatus	Gerente de ayuntamiento, Trabajador
PUT	/tasksManagement/tasks/*	Gerente de ayuntamiento
GET	/tasksManagement/notWorkedPeriods	Gerente de ayuntamiento
GET	/tasksManagement/notWorkedPeriods/*	Gerente de ayuntamiento
POST	/tasksManagement/notWorkedPeriods	Gerente de ayuntamiento
PUT	/tasksManagement/notWorkedPeriods/*	Gerente de ayuntamiento

8.4 Subsistema *Frontend*

8.4.1 Objetivos

Para desarrollar el *frontend* de la aplicación, se ha utilizado *React* [10] ayudándose de *Redux* [11]. El objetivo de este subsistema es implementar una aplicación del tipo SPA (*Single Page Application*) que realice la función de la interfaz de usuario y se comunique con el *backend* para obtener la información.

Una aplicación del tipo SPA propone el uso de varias vistas en lugar del uso de páginas, lo que permite que la navegación sea más cómoda, ya que el navegador no tiene que recargarse. Un tipo de aplicación SPA proporciona un único punto de entrada, normalmente *index.html*, y es en él donde se van intercambiando las vistas que ofrece.

Es importante que a la hora de implementar el subsistema se apliquen buenas prácticas de diseño para facilitar su desarrollo y mantenimiento.

8.4.2 Arquitectura

La arquitectura general del subsistema es la representada en la figura 8.7. En ella se puede observar como el *frontend* se divide en dos capas. La capa de acceso al servicio, que es la encargada de realizar las peticiones al sistema y recibir los datos que estas devuelven; y la capa de componente de la interfaz de usuario, que es la que muestra la información en el navegador.

La forma de comunicarse del frontend es la descrita en la figura 8.8. El usuario es que interactúa con la interfaz y a través de ella se envían los datos que el usuario ha introducido (en el ejemplo de la figura, los datos del nuevo usuario a crear) a la capa de acceso al servicio, que a su vez se comunica con la API que expone el backend. Cuando esta responde, dependiendo de la respuesta que se obtenga, se realizarán las operaciones oportunas que modifican el componente que el usuario está visualizando.

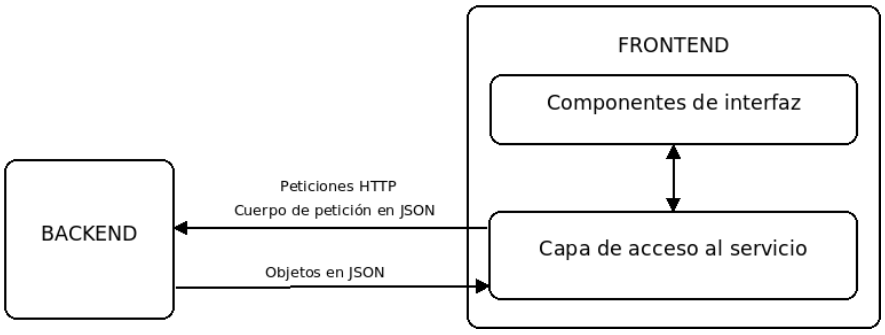


Figura 8.7: *Frontend*: Arquitectura

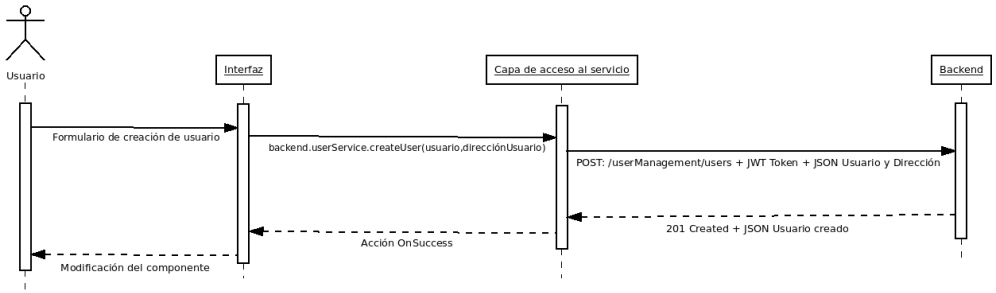


Figura 8.8: *Frontend*: Diagrama de secuencia de comunicación con el servidor

8.4.3 Capa de acceso al servicio

Esta capa es la encargada de componer los mensajes HTTP que se le envían al *backend*. Para realizar esta tarea, se definen los métodos en un fichero común que todos los servicios utilizan para componer las peticiones, esto es, definir las cabeceras, introducir el token de autenticación, pasar los objetos de javascript a *JSON* e introducirlos como cuerpo de la petición en caso de requerirlo, entre otras acciones.

Hay definidos cuatro servicios, siguiendo la división vista en los controladores del *backend*. En cada uno de ellos se utilizan los métodos antes descritos para realizar la petición al endpoint requerido, además se exportan métodos que permitan variar ciertos parámetros de las peticiones, para ser usados en cada componente cuando sea necesario.

8.4.4 Capa de componentes de interfaz

La capa de interfaz de usuario se divide a su vez en módulos en los que se almacenan los componentes que tienen una temática en común. Los módulos que conforman la capa de interfaz de usuario son los mostrados en el diagrama 8.9.

Los módulos *councils*, *patients*, *tasks* y *users* contienen la interfaz de cada una de las funcionalidades que exponen los controladores de la capa servicio del *backend*. Por otra parte, el

módulo *common* contiene los elementos que son utilizados por dos o más componentes, con el objeto de que todos los módulos tengan acceso a un elemento de uso común.

Por último, el módulo *app* es el principal. El componente *app* es el encargado de servir el componente *Body*, que mediante elementos *Router*, permiten asociar *URLs* a componentes. De esta manera, la página que se carga es siempre la misma, modificando su contenido en función de la *URL*, mostrándose un elemento u otro.

En la figura 8.10, se puede observar como se distribuyen los componentes en una página, en este caso la página principal de la aplicación; y como siguen el comportamiento antes descrito.

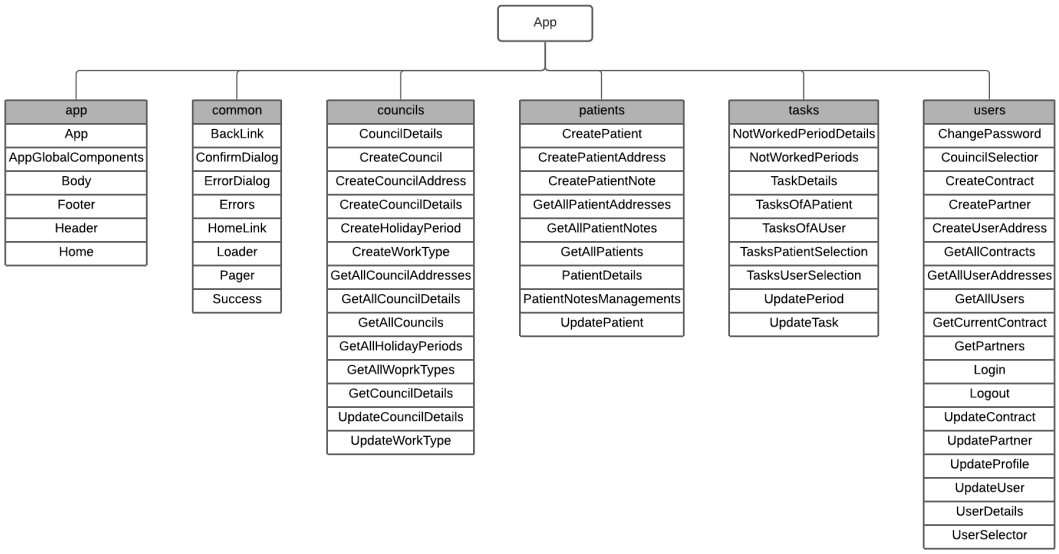


Figura 8.9: *Frontend*: Componentes

A nivel de interfaz, se realizan comprobaciones para evitar que los datos que el usuario introduzca puedan generar errores en la comunicación con el servidor. Para evitarlo, en los formularios se restringe el tipo de dato que se puede introducir, en otros casos, se le obliga a elegir entre una lista de opciones. Antes de comenzar la comunicación, siempre se validan los datos introducidos, intentando minimizar el número de acciones erradas que un usuario puede cometer.

Otra forma de evitar que un usuario realice operaciones no permitidas es mostrando u ocultando opciones, menús o ciertos campos a los usuarios que no tienen permisos para realizar algunas acciones o seleccionar ciertas opciones. La utilización de componentes y del elemento *Router* hace más sencillo modificar la interfaz, por ejemplo, dependiendo del rol de un usuario.

En la creación de componentes, se han utilizado *React hooks*. Una característica de *React*,

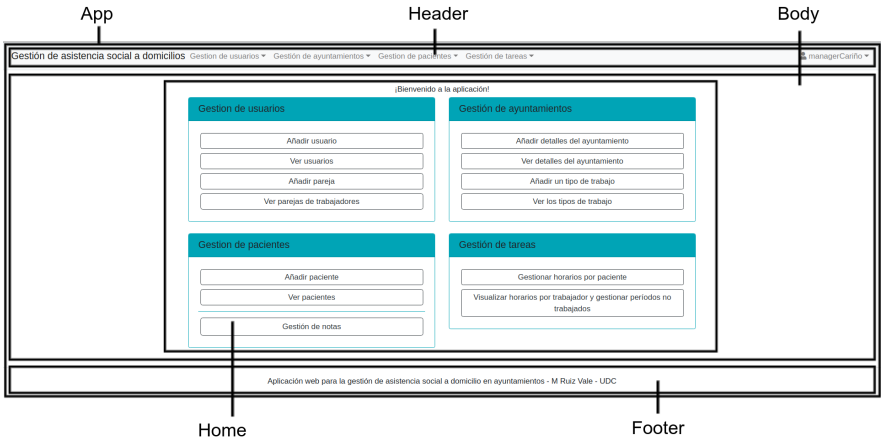


Figura 8.10: *Frontend*: Ejemplo Componentes

introducida en la versión 16.8. Los *hooks* son funciones predefinidas de *React* que permiten acceder a ciertas características como, por ejemplo, el estado o el ciclo de vida de *React*. El uso del *hook useEffect* permite realizar acciones cuando un componente es renderizado, actualizado o desmontado. Otro *hook* utilizado en el sistema es *useState*, lo que crea un parámetro, asignándole el valor indicado en el *hook* y un método para modificarlo (el valor puede ser un valor numérico, un objeto o una función).

En ocasiones es necesario compartir el estado de algún componente con otros, esto es, la información que utiliza este. Para ello se utiliza *Redux*, que permite mantener un estado global y una fuente de información única, accesible en todo momento sin la necesidad de realizar una petición al *backend*.

La forma en que *Redux* consigue mantener el estado utilizando una única fuente de datos, en este caso esta fuente es un objeto donde se almacena todo el árbol del estado de una aplicación, llamado *store*. Además, el estado del *Redux* no se puede modificar directamente, sino que tiene que realizarse a través de acciones, por lo que solamente está permitido acceder a él para mostrarlo en la vista. Para realizar los cambios en el estado se utilizan los *reducers*, funciones que con los mismos parámetros de entrada producen la misma salida siempre. De esta manera, el estado no se modifica nunca sino que se crean copias a partir de los estados anteriores. Lo que impide que existan inconsistencias entre los datos almacenados en el *store*. En la figura 8.11 se puede observar el funcionamiento y la comunicación entre los componentes que forman *Redux*.

Por ejemplo, el usuario que ha accedido a la aplicación está guardado en el estado de *Redux* en todo momento, lo que permite consultar su rol para comprobar si tiene los permisos necesarios para realizar cierta acción.

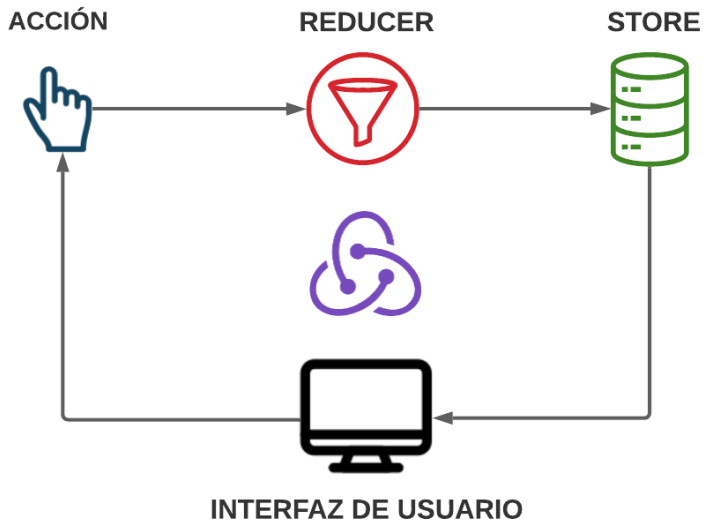


Figura 8.11: *Frontend*: Funcionamiento y comunicación *Redux*

8.4.5 *Bootstrap*

Para la creación de la interfaz se ha utilizado la librería *Bootstrap*, que contiene plantillas con menús, formularios, botones y herramientas de navegación, entre otros muchos elementos. El uso de las clases de *Bootstrap* facilita la consistencia entre componentes y la maquetación de cada uno de ellos.

8.4.6 *react-intl*

Para lograr una internacionalización de la interfaz, se ha utilizado la librería *react-intl*. Esta proporciona un componente llamado *<FormattedMessage>*, que transforma un identificador en el texto que se corresponda, dependiendo del idioma del navegador.

Para proporcionar la internacionalización en gallego, inglés y castellano se han creado tres ficheros en formato campo:valor en el que se relaciona cada identificador con el texto en el idioma correspondiente.

8.4.7 *FullCalendar*

Para facilitar el uso de calendarios y sus elementos, necesarios para la gestión de tareas de pacientes, trabajadores y la gestión de sus períodos no trabajados; se ha utilizado la librería *FullCalendar* [13], que proporciona el componente también llamado *<FullCalendar>*.

De esta manera, muchas de las interacciones con los calendarios tales como el movimiento

entre fechas, la representación de tareas en una cuadrícula semanal, las acciones de click sobre las tareas, entre otras; son facilitadas por este componente, agilizando el desarollo de la aplicación.

Capítulo 9

Implementación

9.1 Software requerido

Para poder compilar el código fuente de la aplicación, se necesitan los siguientes prerrequisitos:

- El entorno de ejecución **Node** en la versión **12.14.0 o posterior**.
- El gestor de paquetes JavaScript **Yarn** en su versión **1.21.1 o posterior**.
- El paquete de desarrollo **Java SE 8+**.
- La herramienta de gestión y construcción de proyectos **Maven 3+**.
- La base de datos **MySQL** en su versión **8.0 o posterior**.

9.2 Estructura

Las carpetas del código fuente se dividen en los dos subsistemas de la aplicación: *backend* y *frontend*. A continuación se detallan los directorios más reseñables de cada uno de ellos.

En la estructura del backend (fig. 9.1) destaca, en la raíz de la carpeta del subsistema, el fichero *pom.xml* en donde se describen las dependencias y otras configuraciones del proyecto *Maven* [17].

La carpeta *src* es la principal, donde está la mayoría del código implementado del *backend*. En *main* se almacenan todos las ficheros fuente de java y en ella se puede ver, además de la división en carpetas según la capa que se implementa en ellas, el archivo *Application.java*. En él se describe la aplicación de *Spring-boot*[8].

En la carpeta *resources* se encuentran los ficheros de configuración de la aplicación y la internacionalización de los errores de la capa servicios.

La carpeta *sql* es la que almacena los archivos utilizados para inicializar la base de datos, crear las tablas e introducir algunos datos iniciales.

Por otra parte, en la estructura del *frontend* (fig. 9.2), destacan los ficheros *package.json* y *package-lock.json*. El primero de ellos utilizado para definir parámetros del proyecto de npm como el autor, el nombre del proyecto y, sobre todo, las dependencias. El segundo archivo, *package-lock.json*, indica la versión específica de cada dependencia que se va a utilizar.

En la carpeta *node_modules* se almacenan los ficheros que conforman las dependencias del proyecto, descargados en el momento de hacer el *install*.

Es en la carpeta *src* en la que se almacena el código de la aplicación. A su vez, esta se divide en subcarpetas que albergan, el código de la capa de acceso al servicio, la internacionalización, los componentes de la aplicación y el *store*, que forma parte de *Redux*.

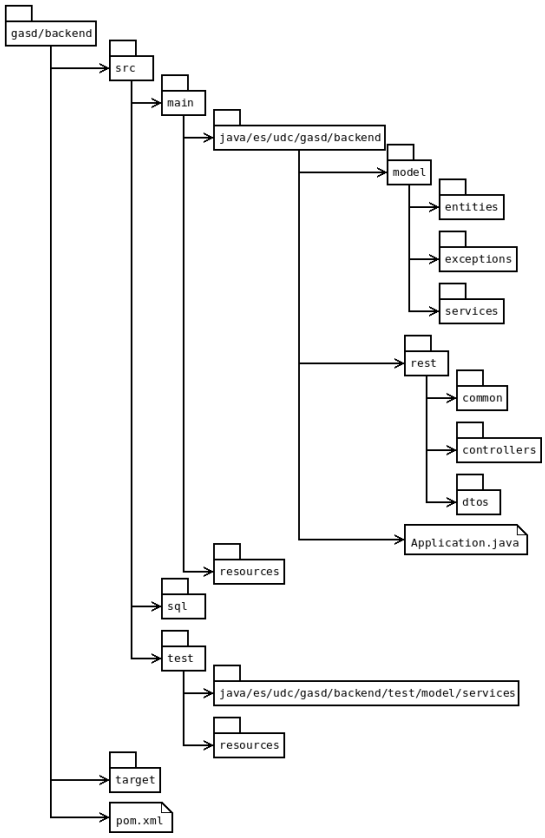


Figura 9.1: Estructura de carpetas *backend*

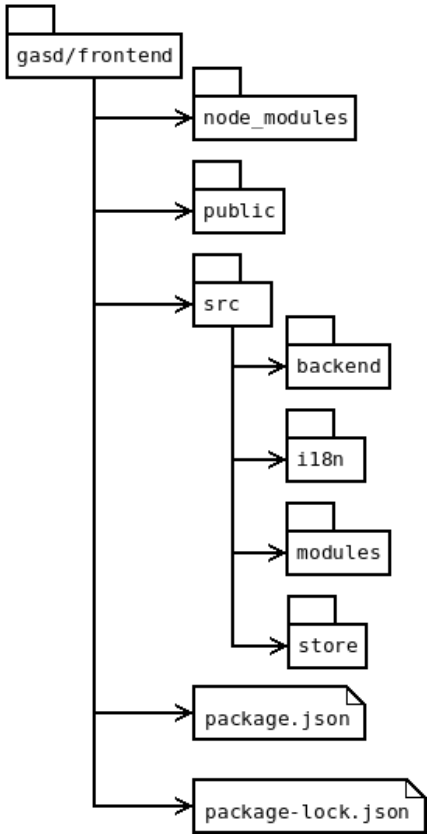


Figura 9.2: Estructura de carpetas *frontend*

9.3 Instrucciones de compilación y ejecución

Para realizar la compilación y ejecución del sistema, se deben seguir los siguientes pasos:

Creación de la base de datos

Inicio del servidor Mysql server si no está en ejecución:

```
$ mysqld&
```

Creación de las bases de datos gasd y gasdtest (Bases de datos utilizadas en producción y testeo, respectivamente). Creación y dotación de permisos sobre dichas tablas al usuario gasd:

```
$ mysqladmin -u root create gasd
$ mysqladmin -u root create gasdtest

$ mysql -u root
    CREATE USER 'gasd'@'localhost' IDENTIFIED BY 'gasd';
    GRANT ALL PRIVILEGES ON gasd.* to 'gasd'@'localhost' WITH GRANT
    OPTION;
    GRANT ALL PRIVILEGES ON gasdtest.* to 'gasd'@'localhost' WITH
    GRANT OPTION;
    exit
```

Compilación del proyecto

Posteriormente se compila y construye el proyecto:

```
$ mvn package
```

Este comando además de compilar y realiza el *build* del proyecto, que resulta en un archivo *jar* almacenado en el directorio *target*, realiza previamente la validación compilación y tests del código.

Ejecución

Desde la carpeta *backend* del proyecto:

```
$ mvn sql:execute (Solo la primera vez, para crear las tablas en la
    base de datos)
$ mvn spring-boot:run
```

Se quedará el servidor iniciado y aparecerán las peticiones que llegan a el a través del terminal, a modo de log.

Desde la carpeta *frontend* del proyecto:

```
$ yarn install (Solo la primera vez, con este comando se descargan
    las librerías necesarias)
$ yarn start
```

El terminal abrirá automáticamente en el navegador predeterminado la página web que proporciona la interfaz del sistema y se quedará sirviendo el subsistema frontend.

Capítulo 10

Pruebas

En los siguientes apartados se describen los diferentes tipos de pruebas realizados en las distintas capas del software que compone la aplicación desarrollada en este proyecto. Se distinguen las pruebas de la capa modelo y servicios del backend y las pruebas finales sobre la capa web.

10.1 Pruebas capa modelo

Durante la implementación de la capa servicios del modelo, en el *backend*, se han ido desarrollando también las pruebas automatizadas de la misma, para comprobar que cada una de las funciones que se estaban creando realizaba el proceso esperado y los resultados eran los deseados.

Realizar las pruebas sobre la capa servicios del modelo asegura que la capa inferior opera como se espera y sobre todo, evita que un error en una capa inferior se extienda sobre el resto de capas dificultando su detección. No se ha considerado la creación de pruebas de unidad de los DAOs y entidades de la capa modelo, debido a que la lógica de negocio está centralizada en la capa de servicios del modelo y tanto entidades como DAOs se están probando indirectamente.

Para realizar las pruebas se ha utilizado la herramienta *JUnit* citeJUnitDocs y su integración con el *framework Spring* [8]. *JUnit* permite la automatización de pruebas, comprobando el correcto funcionamiento de los diferentes métodos y generando estadísticas al respecto. La anotación *@Test* indica qué métodos van a ser comprobados por *JUnit*. Para verificar los resultados de cada uno de los métodos probados, se utiliza la clase *Assertions* de *JUnit*, en la que se proporcionan distintos métodos para comparar el estado esperado y el retorno de un método, ya sea un valor o una excepción. Cada prueba se implementa de forma independiente a las anteriores, de modo que cada una se encarga de crear los datos necesarios para ejecutar y validar un caso de uso. Una vez finalizada cada prueba, se eliminan los datos que se han

creado tanto durante la inicialización de la prueba como durante la ejecución y validación del caso de uso concreto, de modo que no afecte a la ejecución de prueba posteriores.

Para esta aplicación, se ha creado una clase de prueba diferente por cada servicio del modelo desarrollado, y varios métodos de prueba por cada caso de uso para validar los diferentes casos de éxito así como las situaciones en las que se devuelven excepciones de lógica de negocio.

Una vez creado un servicio y sus correspondientes pruebas se puede obtener la cobertura de los mismos, esto es el porcentaje de líneas del código que los tests comprueban. Se estima que una cobertura de entre el 70 u 80 por ciento es necesaria para la detección de la mayoría de los *bugs* y errores en el código. En los tests realizados en la capa servicios se han obtenido (fig. 10.1) coberturas con un porcentaje mayor al 99% en las clases que implementan los servicios y superiores al 94% en el paquete de entidades. Las instrucciones no probadas en el paquete de entidades son debidas a que los métodos *set* no son utilizados pero son requeridos por *Spring-boot* para el mapeo de entidades, al igual que sucede en el paquete de excepciones. Obviamente como no se realizan pruebas automatizadas de la capa servicios (ver apartado 10.2), la cobertura en los paquetes que la componen es nula.

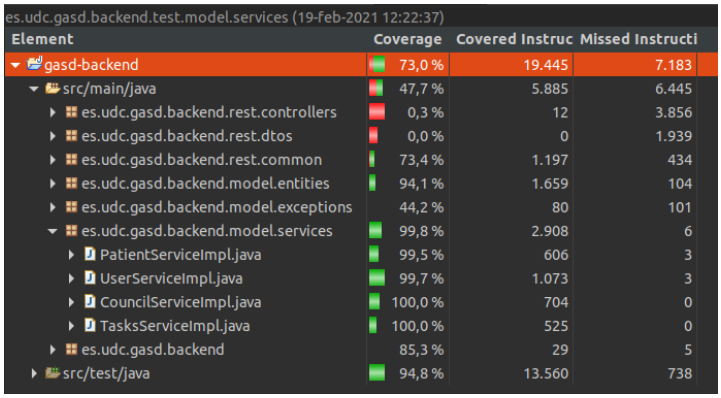


Figura 10.1: Cobertura de las pruebas. Capa modelo

10.2 Pruebas capa servicio

Para comprobar la capa servicio se han realizado pruebas manuales, ayudándonos de la herramienta *Postman* [20]. Esta aplicación proporciona una interfaz sencilla para realizar las peticiones a la API que expone la capa servicios y analizar las respuestas de la misma.

Por cada uno de los *endpoints* documentados en la sección del apéndice A.1, se comprueba su buen funcionamiento. Se debe validar que, por cada petición, el código de la respuesta (respuesta es satisfactoria o código de error) y el cuerpo de la respuesta coincida con lo esperado.

10.3 Pruebas de la capa interfaz de usuario

Al final de cada iteración, se comprueba que lo desarrollado en cada una de ellas funciona correctamente en la capa de interfaz de usuario. Si su funcionamiento es el deseado, se puede dar por concluido el fin de cada una de ellas ya que implicaría que el funcionamiento en las capas inferiores también es el esperado.

Las pruebas realizadas en esta capa son manuales y se han realizado para cada una de las funcionalidades de la aplicación. Algunos ejemplos de pruebas realizadas en la capa de usuario son las siguientes:

- Creación de un paciente:
  - Se accede a la aplicación con el usuario administrador.
  - Se procede a crear un ayuntamiento, pulsando el botón *Añadir Ayuntamiento*.
  - Se introducen unos datos válidos y se presiona el botón *Añadir Ayuntamiento* para añadir el ayuntamiento al sistema.
  - Se vuelve a la pantalla principal y se pulsa en *Añadir usuario* para crear un mánager para el ayuntamiento recién creado.
  - Una vez en el formulario se elige el ayuntamiento que hemos creado y el rol gerente, se añaden los datos de la dirección y se presiona el botón *Añadir usuario*.
  - Se cierra la sesión con el administrador y se vuelve a iniciar con las credenciales del mánager que se acaba de crear.
  - En la pantalla principal se accede a la creación de un paciente presionando el botón *Añadir paciente*
  - En el formulario de creación, se comprueba que los campos obligatorios no se pueden dejar vacíos.
  - Una vez introducidos datos válidos se pulsa el botón *Añadir paciente* y se añade el paciente al sistema.
  - En la pantalla principal se accede a la función *Ver pacientes* para comprobar que los datos introducidos del paciente se han guardado correctamente.
- Añadir una tarea a un paciente:
  - Se accede a la aplicación con el usuario administrador.
  - Se procede a crear un ayuntamiento.
  - Se crea un usuario con el rol de mánager para el ayuntamiento recién creado.

- Se cierra la sesión con el administrador y se inicia con el usuario mánager.
- Se crea un usuario trabajador en el ayuntamiento al que pertenece el mánager.
- Se crea un paciente.
- En el menú principal se accede a la función de gestión presionando el botón *Gestionar horarios por paciente*
- Se elige el paciente anteriormente creado y se presiona el icono de la columna ver horarios
- En la vista de horarios se hace clic y se arrastra para elegir el tiempo de duración de la tarea.
- Una vez desplegado el menú de *Añadir tarea* se comprueba que no se puede añadir una tarea sin elegir un trabajador.
- En el menú de *Añadir tarea*, se elige el trabajador acabado de crear en el campo homónimo, se añaden unos comentarios a la tarea y se pulsa el botón *Añadir tarea*.
- Una vez creado se comprueba como las horas semanales en la semana que se está visualizando se corresponden con el tiempo asignado en la tarea.
- Se vuelve a realizar el procedimiento de creación de tarea para comprobar que las horas asignadas semanales de el trabajador escogido en la tarea anterior se han incrementado correctamente.

Capítulo 11

Conclusiones y futuras líneas de trabajo

11.1 Conclusiones

Para concluir este trabajo se analizan los objetivos marcados que se pretendían alcanzar al comienzo del mismo. Con los objetivos planteados se ha conseguido un sistema que:

- Es capaz de gestionar varios ayuntamientos simultáneamente.
- Consigue manejar distintos perfiles de usuario y gestionarlos de manera independiente.
- Facilita la gestión de los empleados del ayuntamiento.
- Proporciona un sistema sencillo de creación de tareas y composición de horarios.
- Habilita un perfil de usuario destinado al paciente o familiar.
- Tiene soporte en varios idiomas e interfaz sencilla.

El proyecto ha sido diseñado y desarrollado siguiendo las metas en el inicio. Debido a que estas han sido alcanzadas y las operaciones a realizar, identificadas en la fase de requisitos, han sido desarrolladas satisfactoriamente, se puede concluir que en cuanto a funcionalidades, el sistema cumple con lo deseado.

En cuanto a la facilidad de mantenimiento y de extensión, durante todo el desarrollo de la aplicación se han tenido en cuenta estos dos requisitos y se han aplicado buenas prácticas de diseño y desarrollo, mediante el empleo de patrones y principios de arquitectura y desarrollo, además de estándares de programación.

Aunque la realización del proyecto ha propuesto un reto en cuanto el desarrollo de la parte *frontend* y la formación en ciertas tecnologías utilizadas como *React* o *Redux*, ha sido una

forma de completar la formación adquirida durante el grado, especialmente al haber realizado la mención en TI, menos orientada al desarrollo de aplicaciones.

11.2 Futuras líneas de trabajo

Una vez finalizado el desarrollo del sistema, alcanzando los objetivos fijados y partiendo de la aplicación creada, se abren varias líneas por donde podrían avanzar el trabajo futuro:

- **Funcionalidades que completan y extienden el sistema actual**:
  - **Integración con aplicaciones de localización sobre mapa**. Integración con aplicaciones como *Google Maps* para el posicionamiento de los domicilios de los pacientes.
  - **Sistemas de comunicación entre usuarios de la aplicación.** Sistema de mensajería privada entre el gerente o coordinador y los usuarios de paciente para gestión de quejas e incidencias.
- **Aplicación móvil.** Aunque la aplicación ha sido desarrollada siguiendo un modelo *responsive*, se podría crear una aplicación móvil, a través de *react-native*, que fuese compatible con los sistemas *Android* e *iOS* y facilitase el uso de la aplicación en dispositivos móviles.
- **Integración con etiquetas *RFID* (Radio Frequency Identification)**. Integración del sistema con el uso de etiquetas de identificación por radiofrecuencia en los domicilios para gestionar el momento de entrada y salida de los trabajadores en cada , permitiendo un seguimiento más preciso de los trabajadores.

Apéndices

Apéndice A

Material adicional

A.1 Controladores capa servicios

CouncilController

Tabla A.1: CouncilController

Recurso	Método	Excepciones	Descripción
/councilManagement/councils	POST	DuplicateInstanceException	Crea un ayuntamiento, devuelve el objeto ayuntamiento creado.
/councilManagement/councils	GET	-	Devuelve una lista con todos los ayuntamientos almacenados.
/councilManagement/councils/{id}	PUT	InstanceNotFoundException, DuplicateInstanceException	Actualiza un ayuntamiento.
/councilManagement/councils/{id}	GET	InstanceNotFoundException	Devuelve el ayuntamiento que coincide con el identificador indicado.
/councilManagement/councils/{id}	DELETE	InstanceNotFoundException	Borra el ayuntamiento que coincide con el identificador.
/councilManagement/councils/{id}/addresses	POST	InstanceNotFoundException, NoCurrentAvaliableException, PermissionException	Crea una dirección para el ayuntamiento que tiene el identificador indicado.
/councilManagement/councils/{id}/addresses	GET	InstanceNotFoundException, NoCurrentAvaliableException	Devuelve una lista con todas las direcciones del ayuntamiento que tiene el identificador indicado.
/councilManagement/councils/{id}/addresses?current=true	GET	InstanceNotFoundException, NoCurrentAvaliableException	Devuelve una lista con la dirección actual del ayuntamiento que tiene el identificador indicado.
/councilManagement/councils/{id}/addresses/{addressId}	GET	InstanceNotFoundException	Devuelve la dirección que coincida con el addressId, perteneciente al ayuntamiento con el identificador indicado.
/councilManagement/councils/{id}/addresses/{addressId}	DELETE	InstanceNotFoundException	Borra la dirección con la identificador indicada con addressId del ayuntamiento que coincida con el identificador.

Continúa en la página siguiente...

Tabla A.1 – *Continuación de la página anterior*

Recurso	Método	Excepciones	Descripción
/councilManagement/councils/{id}/details	POST	InstanceNotFoundException, PermissionException	Se crea un conjunto de detalles del ayuntamiento, devuelve el objeto creado.
/councilManagement/councils/{id}/details	GET	InstanceNotFoundException, NoCurrentAvaliableException	Devuelve una lista con todos los objetos de detalles del ayuntamiento que coincide con el identificador indicado.
/councilManagement/councils/{id}/details?current=true	GET	InstanceNotFoundException, NoCurrentAvaliableException	Devuelve una lista con el objeto de detalles en vigor del ayuntamiento que coincide con el identificador indicado.
/councilManagement/councils/{id}/details/{detailsId}	PUT	InstanceNotFoundException, PermissionException	Actualiza el conjunto de detalles con el identificador detailsId del ayuntamiento que coincide con el identificador indicado. Devuelve el objeto actualizado
/councilManagement/councils/{id}/details/{detailsId}	DELETE	InstanceNotFoundException, PermissionException	Elimina el conjunto de detalles que coincida con detailsId, perteneciente al ayuntamiento que concuerda con el identificador indicado.
/councilManagement/councils/{id}/details/{detailsId}/holidayPeriods	POST	InstanceNotFoundException, PermissionException, HolidayOutOfDetailsBoundsException	Crea un período festivo, perteneciente al conjunto de detalles que coincida con detailsId y relativo al ayuntamiento que coincida con el identificador indicado. Devuelve el objeto creado
/councilManagement/councils/{id}/details/{detailsId}/holidayPeriods	GET	InstanceNotFoundException	Devuelve una lista con los períodos de vacaciones de el conjunto de detalles que coincida con detailsId.
/councilManagement/councils/{id}/details/{detailsId}/holidayPeriods/{holidayPeriodId}	DELETE	InstanceNotFoundException, PermissionException	Elimina el período festivo identificado con holidayPeriodId.
/councilManagement/workTypes	POST	DuplicateInstanceException	Crea un tipo de trabajo. Devuelve el objeto creado.
/councilManagement/workTypes	GET	-	Devuelve una lista con todos los tipos de trabajo almacenados.
/councilManagement/workTypes/{id}	PUT	DuplicateInstanceException	Actualiza el tipo de trabajo. Devuelve el objeto actualizado.

PatientController

Tabla A.2: PatientController

Recurso	Método	Excepciones	Descripción
/patientManagement/patients	POST	InstanceNotFoundException, PatientUserRoleRequiredException	Crea un paciente, devuelve el objeto ayuntamiento creado.
/patientManagement/patients	GET	InstanceNotFoundException, PermissionException	Devuelve una lista con todos los pacientes almacenados si la ejecuta un gerente o trabajador.
/patientManagement/patients?patientUserId={userId}	GET	InstanceNotFoundException, PermissionException	Devuelve una lista con todos los pacientes almacenados a cargo del usuario de paciente con el identificador indicado.
/patientManagement/patients/{id}	GET	InstanceNotFoundException, PermissionException	Devuelve el paciente que coincide con el identificador indicado.
/patientManagement/patients/{id}	PUT	InstanceNotFoundException, PermissionException, PatientUserRoleRequiredException	Actualiza los datos de un paciente.
/patientManagement/patients/{id}/addresses	POST	InstanceNotFoundException, PermissionException	Crea una dirección para el paciente que tiene el identificador indicado.
/patientManagement/patients/{id}/addresses	GET	InstanceNotFoundException, NoCurrentAvaliableException, PermissionException	Devuelve una lista con todas las direcciones del paciente que tiene el identificador indicado.
/patientManagement/patients/{id}/addresses?current=true	GET	InstanceNotFoundException, NoCurrentAvaliableException, PermissionException	Devuelve una lista con la dirección actual del paciente que tiene el identificador indicado.
/patientManagement/patients/{id}/addresses/{addressId}	GET	InstanceNotFoundException, PermissionException	Devuelve la dirección que coincida con el addressId, perteneciente al paciente con el identificador indicado.
/patientManagement/patients/{id}/addresses/{addressId}	DELETE	InstanceNotFoundException, PermissionException	Borra la dirección con la identificador indicada con addressId del paciente que coincida con el identificador.

Continúa en la página siguiente...

Tabla A.2 – *Continuación de la página anterior*

Recurso	**Método**	**Excepciones**	**Descripción**
/patientManagement/patients/{id}/patientNotes	POST	InstanceNotFoundException, PermissionException	Crea una nota asignada al paciente que tiene el identificador indicado. Devuelve el objeto creado.
/patientManagement/patients/{id}/patientNotes	GET	InstanceNotFoundException, PermissionException	Devuelve una lista con las notas con visibilidad pública asignadas al paciente del identificador indicado y la lista de usuarios que las crearon.
/patientManagement/patients/{id}/patientNotes?all=true	GET	InstanceNotFoundException, PermissionException	Devuelve una lista todas las notas asignadas al paciente del identificador indicado y la lista de usuarios que las crearon.
/patientManagement/patients/{id}/patientNotes/{patientNoteId}/changeVisibility	POST	InstanceNotFoundException, PermissionException	Modifica la visibilidad de la nota con identificador patientNoteId, relativa al paciente con el identificador indicado. Devuelve la nota modificada
/patientManagement/patients/{id}/patientNotes/{patientNoteId}/	DELETE	InstanceNotFoundException, PermissionException	Elimina la nota identificada con patientNoteId, perteneciente al paciente con el identificador indicado.

TaskController

Tabla A.3: TaskController

Recurso	Método	Excepciones	Descripción
/tasksManagement/tasks	POST	InstanceNotFoundException, DifferentCouncilException, UserNotAWorkerException, PermissionException	Crea una tarea. Devuelve el objeto creado.
/tasksManagement/tasks?entityFilter={entityFilter}&entityId={entityId}&startDate={startDate}&endDate={endDate}	GET	InstanceNotFoundException, PermissionException	Devuelve una lista con las tareas que estén comprendidas entre las fechas indicadas (en millisegundos) de un usuario (si el valor entityFilter es ”user”) o de un paciente (si el valor de entityFilter es ”patient”). Además devuelve todas las entidades que participen en las tareas, tanto pacientes como trabajadores.
/tasksManagement/tasks/{id}	GET	InstanceNotFoundException, PermissionException	Devuelve la tarea que tiene el identificador indicado.
/tasksManagement/tasks/{id}	PUT	InstanceNotFoundException, DifferentCouncilException, UserNotAWorkerException, PermissionException	Modifica los atributos de una tarea. Devuelve el objeto modificado.
/tasksManagement/tasks/{id}/taskStatus	POST	InstanceNotFoundException, PermissionException	Modifica el estado de una tarea. Devuelve el objeto modificado.
/tasksManagement/notWorkedPeriods	POST	InstanceNotFoundException, PermissionException	Crea un período no laborable, en el que un trabajador no realizará o ha realizado su trabajo. Devuelve el objeto creado.
/tasksManagement/notWorkedPeriods?userId={userId}&startDate={startDate}&endDate={endDate}	GET	InstanceNotFoundException, PermissionException	Devuelve la lista de períodos no trabajados de un usuario entre las fechas indicadas.
/tasksManagement/notWorkedPeriods?userId={userId}&startDate={startDate}&endDate={endDate}&onlyTime=true	GET	InstanceNotFoundException, PermissionException	Devuelve una lista con el tiempo total de cada uno de los períodos no trabajados de un usuario entre las fechas indicadas.

Continúa en la página siguiente...

Tabla A.3 – *Continuación de la página anterior*

Recurso	**Método**	**Excepciones**	**Descripción**
/tasksManagement/notWorkedPeriods/{id}	GET	InstanceNotFoundException, PermissionException	Devuelve el período no trabajado con el identificador indicado.
/tasksManagement/notWorkedPeriods/{id}	PUT	InstanceNotFoundException, PermissionException	Modifica un período no trabajado. Devuelve el objeto modificado.

UserController

Tabla A.4: UserController

Recurso	Método	Excepciones	Descripción
/userManagement/users	POST	DuplicateInstanceException, InstanceNotFoundException, PatientUserRoleRequiredException	Crea un usuario, devuelve el objeto creado.
/userManagement/users/login	POST	IncorrectLoginException	Identifica a un usuario en el sistema. Devuelve el objeto usuario con los parámetros de autenticación.
/userManagement/users/loginFromServiceToken	POST	InstanceNotFoundException	Identifica a un usuario en el sistema utilizando el token. Devuelve el objeto usuario con los parámetros de autenticación.
/userManagement/users	GET	InstanceNotFoundException, PermissionException	Si hace la petición un administardor, devuelve una lista con todos los usuarios almacenados. Si lo hace otro tipo de usuario, recibe una lista con los usuarios de su ayuntamiento asociado.
/userManagement/users?workersWithPartner=no	GET	InstanceNotFoundException, PermissionException	Devuelve una lista con todos los usuarios del tipo trabajador sin parejas asociadas.
/userManagement/users?freeWorkers=true&startDate={startDate}&endDate={endDate}&startTimeVisualized={startTimeVisualized}&endTimeVisualized={endTimeVisualized}	GET	InstanceNotFoundException, PermissionException	Devuelve un objeto usuario que esté libre entre las horas *startDate* y *endDate* y las horas trabajadas en el tiempo de visualización.
/userManagement/users?role={rol}	GET	InstanceNotFoundException, PermissionException	Devuelve una lista con todos los usuarios del rol especificado y que se correspondan al ayuntamiento del usuario que realiza la petición.
/userManagement/users/{id}/changePassword	POST	PermissionException, InstanceNotFoundException, IncorrectPasswordException	Modifica la contraseña del usuario con el identificador indicado, que debe ser el que realiza la petición.

Continúa en la página siguiente...

Tabla A.4 – *Continuación de la página anterior*

Recurso	Método	Excepciones	Descripción
/userManagement/users/{id}/selfUpdate	POST	PermissionException, InstanceNotFoundException, DuplicateInstanceException	Modifica algunos parámetros del propio usuario que realiza la petición.
/userManagement/users/{id}	PUT	InstanceNotFoundException, DuplicateInstanceException, PermissionException	Actualiza los datos de un usuario.
/userManagement/users/{id}	GET	InstanceNotFoundException, PermissionException	Devuelve el usuario que coincide con el identificador indicado.
/userManagement/users/{id}	DELETE	InstanceNotFoundException, PermissionException	Elimina el usuario indicado.
/userManagement/users/{id}/addresses	POST	InstanceNotFoundException, PermissionException	Crea una dirección para el usuario que tiene el identificador indicado.
/userManagement/users/{id}/addresses	GET	InstanceNotFoundException, NoCurrentAvaliableException, PermissionException	Devuelve una lista con todas las direcciones del usuario que tiene el identificador indicado.
/userManagement/users/{id}/addresses?current=true	GET	InstanceNotFoundException, NoCurrentAvaliableException, PermissionException	Devuelve una lista con la dirección actual del usuario que tiene el identificador indicado.
/userManagement/users/{id}/addresses/{addressId}	GET	InstanceNotFoundException, PermissionException	Devuelve la dirección que coincida con el addressId, perteneciente al paciente con el identificador indicado.
/userManagement/users/{id}/addresses/{addressId}	DELETE	InstanceNotFoundException, PermissionException	Borra la dirección con la identificador indicada con addressId del paciente que coincida con el identificador.
/userManagement/users/{id}/contracts	POST	PermissionException, InstanceNotFoundException	Crea un objeto contrato asociado al usuario identificado con el identificador indicado. Devuelve el objeto creado.
/userManagement/users/{id}/contracts	GET	NoCurrentAvaliableException, InstanceNotFoundException	Devuelve una lista con todos los contratos del usuario identificado con el identificador indicado.

Continúa en la página siguiente...

Tabla A.4 – *Continuación de la página anterior*

Recurso	Método	Excepciones	Descripción
/userManagement/users/{id}/contracts?current=true	GET	NoCurrentAvaliableException, InstanceNotFoundException	Devuelve una lista con el contrato vigente del usuario identificado con el identificador indicado.
/userManagement/users/{id}/contracts/{contractId}	GET	NoCurrentAvaliableException, InstanceNotFoundException	Devuelve una lista con el contrato identificado con contractId, perteneciente al usuario con el identificador indicado.
/userManagement/users/{id}/contracts/{contractId}	PUT	InstanceNotFoundException	Actualiza el contrato identificado con contractId, perteneciente al usuario con el identificador indicado. Devuelve el contrato actualizado.
/userManagement/partners	POST	InstanceNotFoundException, UserNotAWorkerException, DifferentCouncilException	Crea una pareja de trabajadores. Devuelve el objeto creado.
/userManagement/partners	GET	InstanceNotFoundException, PermissionException	Devuelve una lista con las parejas almacenadas y una lista con los trabajadores que forman parte de esas parejas.
/userManagement/partners?workerId={workerId}	GET	InstanceNotFoundException, PermissionException	Devuelve una lista con las parejas en las que el trabajador que tiene el identificador indicado forma parte y una lista con todos los trabajadores que forman parte de esas parejas.
/userManagement/partners/{id}	GET	InstanceNotFoundException, PermissionException	Devuelve la pareja de trabajadores identificada con el identificador indicado.
/userManagement/partners/{id}	PUT	InstanceNotFoundException, UserNotAWorkerException, PermissionException	Modifica la pareja de trabajadores identificada con el identificador indicado. Devuelve el objeto modificado.
/userManagement/partners/{id}	DELETE	InstanceNotFoundException, PermissionException	Elimina la pareja de trabajadores identificada con el identificador indicado.

A.2 Instalación del Software

Para realizar la instalación de la aplicación, se deben realizar los pasos descritos en el capítulo 9, sección 9.3, página 70 para compilar el proyecto.

En el *backend*, una vez realizada la compilación, se obtiene un archivo *jar*, alojado en la capeta *target*, que incluye el ejecutable de la aplicación.

En cuanto al *frontend*, la compilación del mismo crea un archivo *zip* que se pondrá accesible desde un servidor (*webpack* o *Apache Tomcat*)

A.3 Manual de usuario

En esta sección se describen las distintas funcionalidades del sistema y el funcionamiento de las mismas. Para realizar una descripción organizada, se han dividido las funcionalidades por los tipos de usuario que las ejecutan:

- Funcionalidades comunes a todos los usuarios.
- Funcionalidades del administrador.
- Funcionalidades del gerente del ayuntamiento.
- Funcionalidades de los trabajadores.
- Funcionalidades de los usuarios de paciente.

A.3.1 Funcionalidades comunes a todos los usuarios

En la figura A.1 se puede ver la pantalla de login, que muestra los campos que se deben rellenar para ingresar en la aplicación. Si las credenciales no son correctas se mostrará un aviso donde se indicará que el usuario o la contraseña son incorrectos.

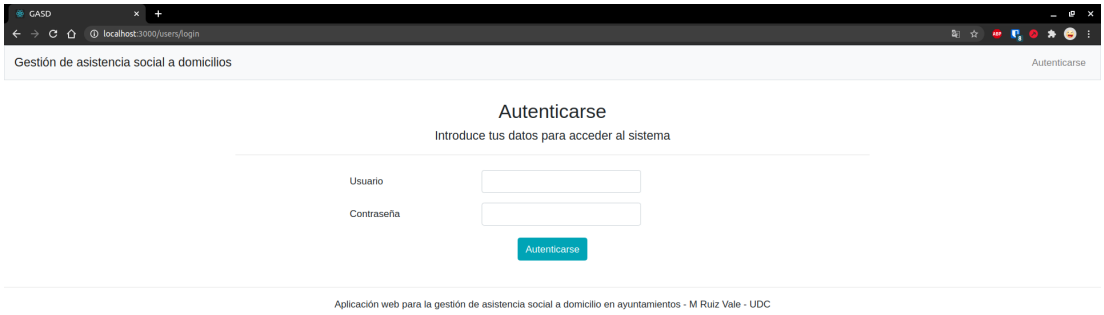


Figura A.1: Login

Una vez un usuario ha ingresado en la aplicación, independientemente del usuario que sea, se dispondrá del menú que se puede ver en la derecha de la figura A.2.

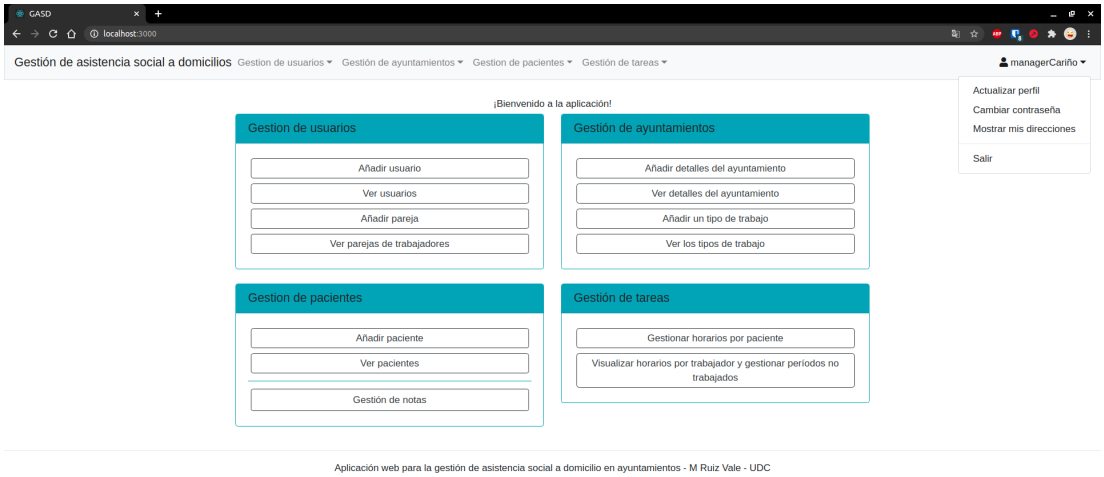


Figura A.2: Menú de usuario

Cualquier usuario puede actualizar ciertos datos del perfil, como se puede observar en la figura A.3. Una vez presionado el botón *guardar*, los datos serán modificados.

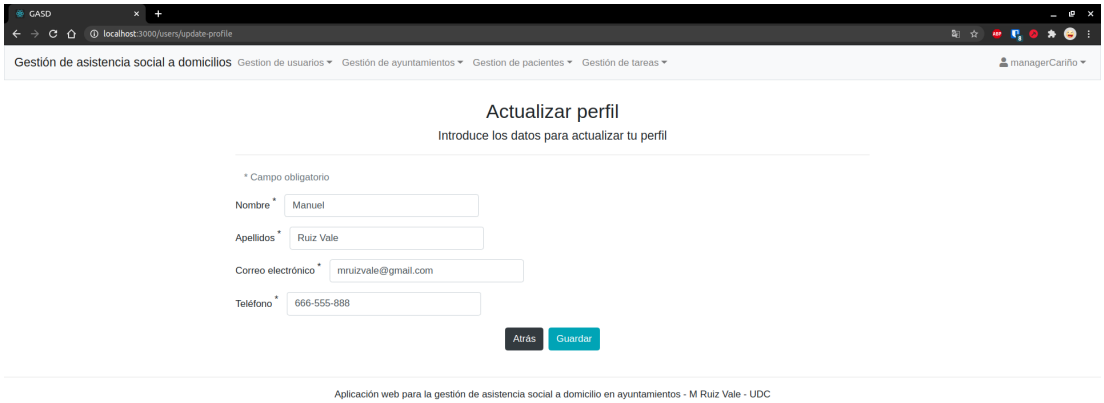


Figura A.3: Actualizar perfil

Es posible modificar la contraseña la contraseña del usuario ingresado en la aplicación, como se puede ver en la figura A.4, se mostrará un formulario en el que se tendrá que introducir la contraseña antigua y la nueva contraseña que se quiere utilizar. Una vez presionado el botón *Guardar* los datos serán actualizados.

Figura A.4: Modificar contraseña

Independientemente del usuario con el que se haya accedido a la aplicación, este podrá

consultar sus direcciones almacenadas (fig. A.5).

Si lo desea, el usuario podrá crear una nueva entrada en las sus direcciones, pulsando el botón *Añadir nueva dirección*, que le mostrará un formulario a rellenar con los campos necesarios.

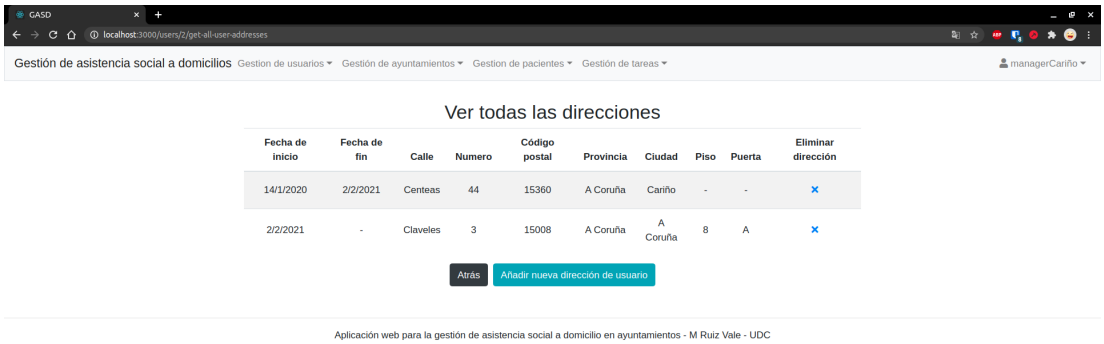


Figura A.5: Direcciones almacenadas de un usuario

A.3.2 Funcionalidades del administrador

Si se accede al sistema con un usuario administrador, el menú principal de la aplicación que se podrá observar es el mostrado en la figura A.6

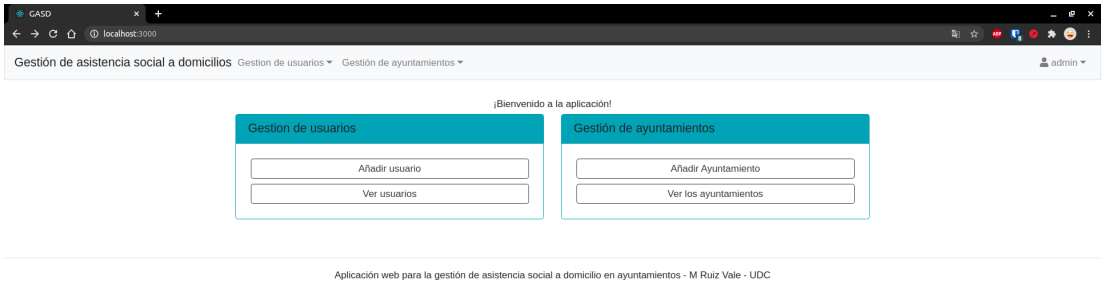


Figura A.6: Menú principal administrador

El formulario de creación de ayuntamientos es el mostrado en las figuras A.7 y A.8, accesible desde el menú principal presionando el botón *Añadir ayuntamiento*. En ellos se introducirán los datos del ayuntamiento a crear y presionando el botón *Añadir ayuntamiento* se efectuará la creación.

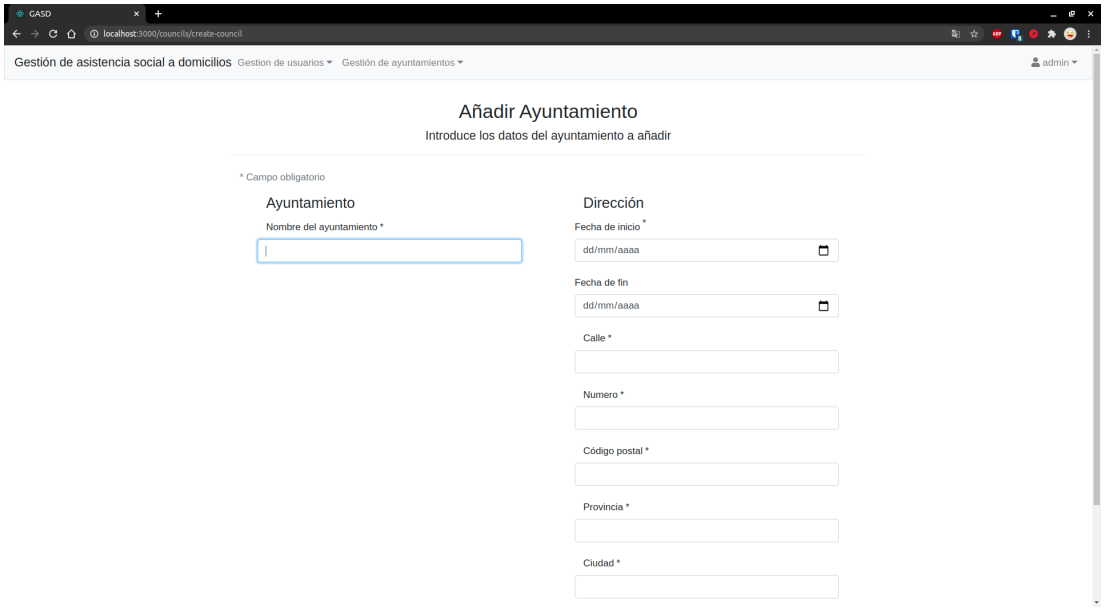


Figura A.7: Creación de ayuntamiento (1)

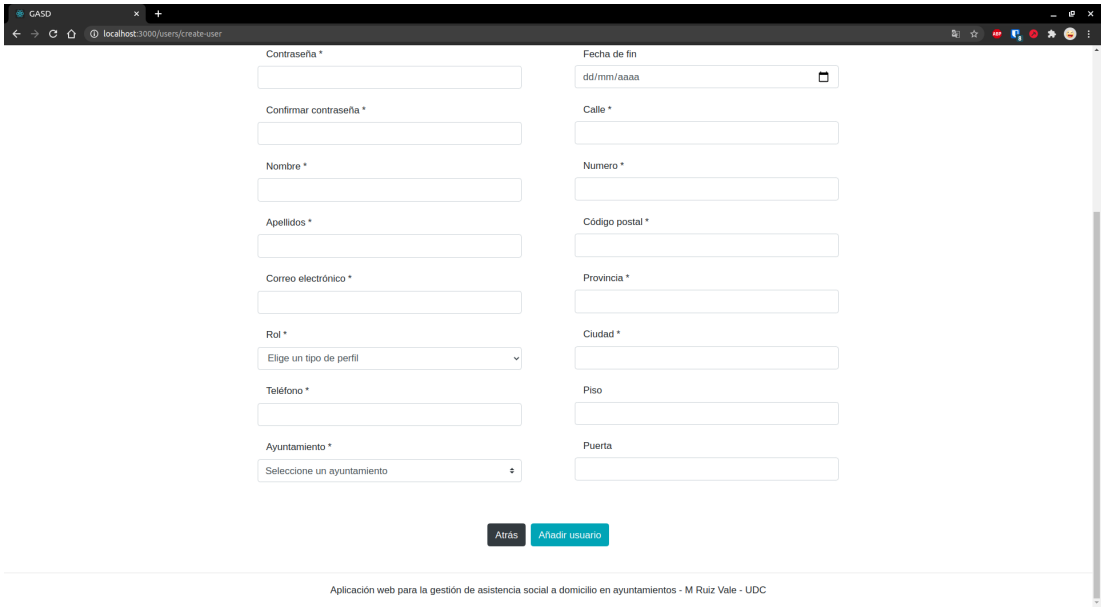


Figura A.8: Creación de ayuntamiento (2)

Si hay ayuntamientos creados se podrán visualizar presionando el botón del menú principal *Ver los ayuntamientos*. Este botón llevará a la pantalla que se puede observar en la figura A.9.

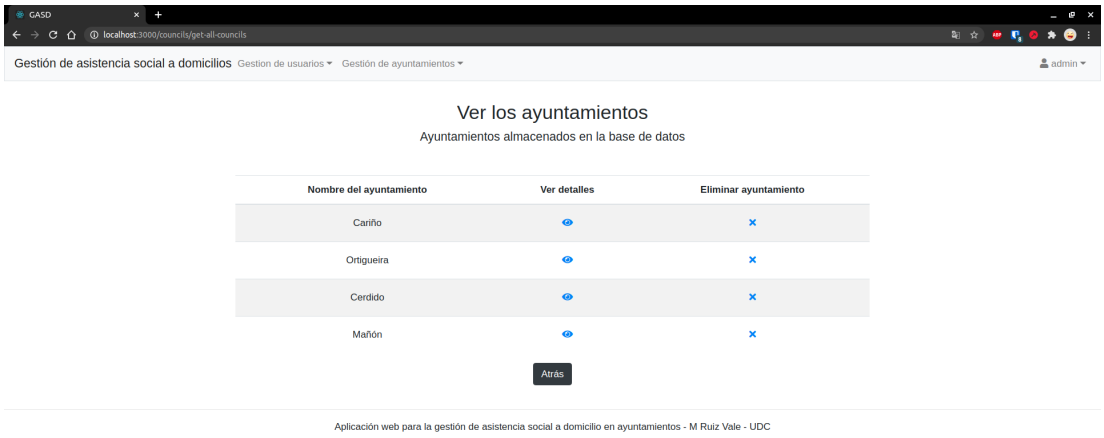


Figura A.9: Visualización de ayuntamientos

Si se desea eliminar un ayuntamiento, presionando la cruz de la derecha de cada uno de los ayuntamientos, en la pantalla de la figura A.9, se desplegará un aviso que pedirá confirmación

(fig. A.10), ya que esto elimina todo el ayuntamiento y sus acciones son **IRREVERSIBLES**.

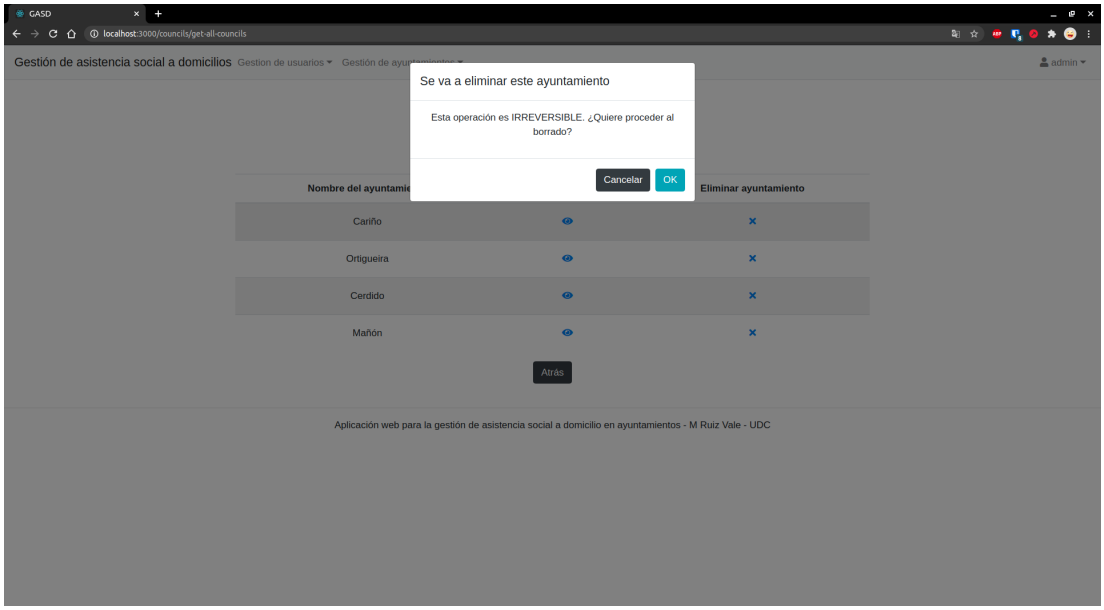


Figura A.10: Borrado de ayuntamiento

Si se hace clic en el ojo de la columna *Ver detalles* en la pantalla de la figura A.9, se podrán observar los datos de cada ayuntamiento, como se puede observar en la figura A.11.

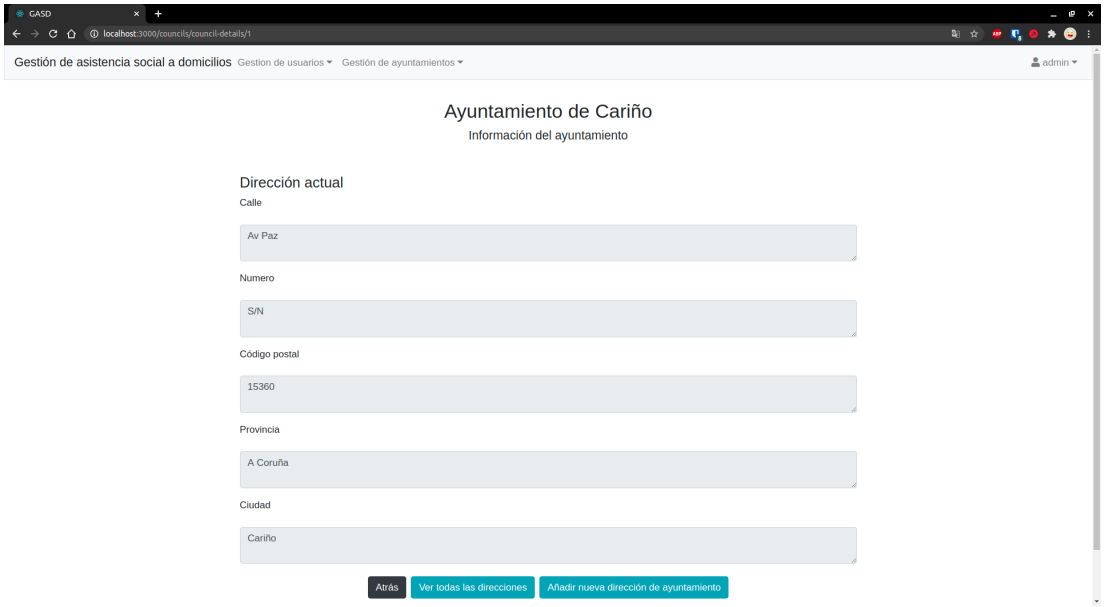


Figura A.11: Datos de ayuntamiento

El formulario de creación de usuario, accesible desde el menú principal en el botón *Añadir*

usuario, es el observado en las figuras A.12 y A.13 en los que se introducirán los campos del nuevo usuario a crear y presionando el botón *Añadir usuario* se efectuará la creación. Un usuario administrador podrá crear cualquier otro usuario (administrador, gerente, trabajador y usuario de paciente) y asignarlo a cualquiera de los ayuntamientos creados.

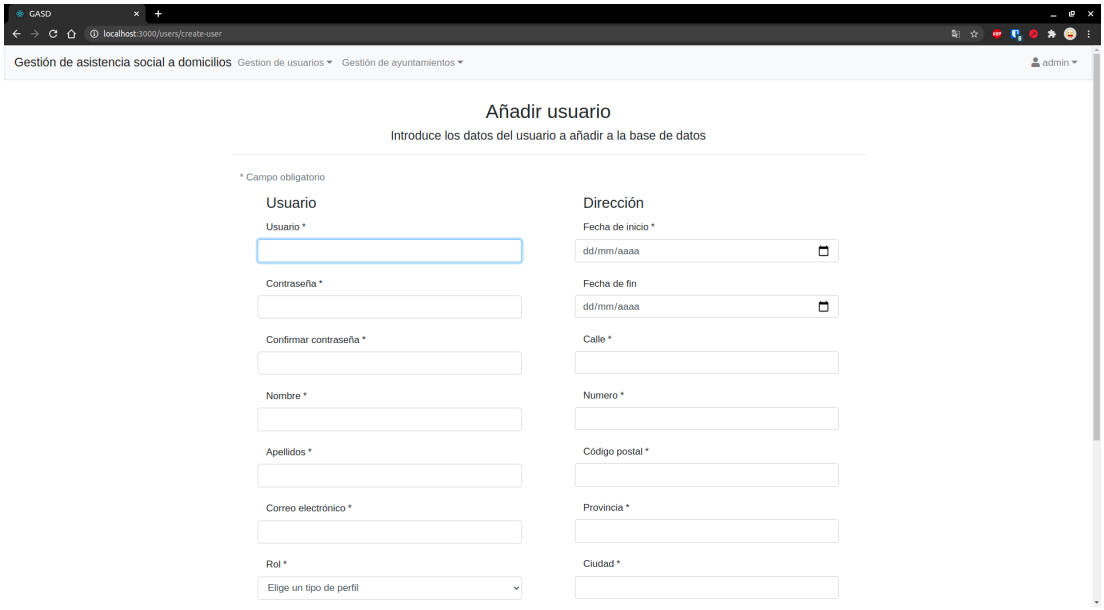


Figura A.12: Creación de usuario (1)

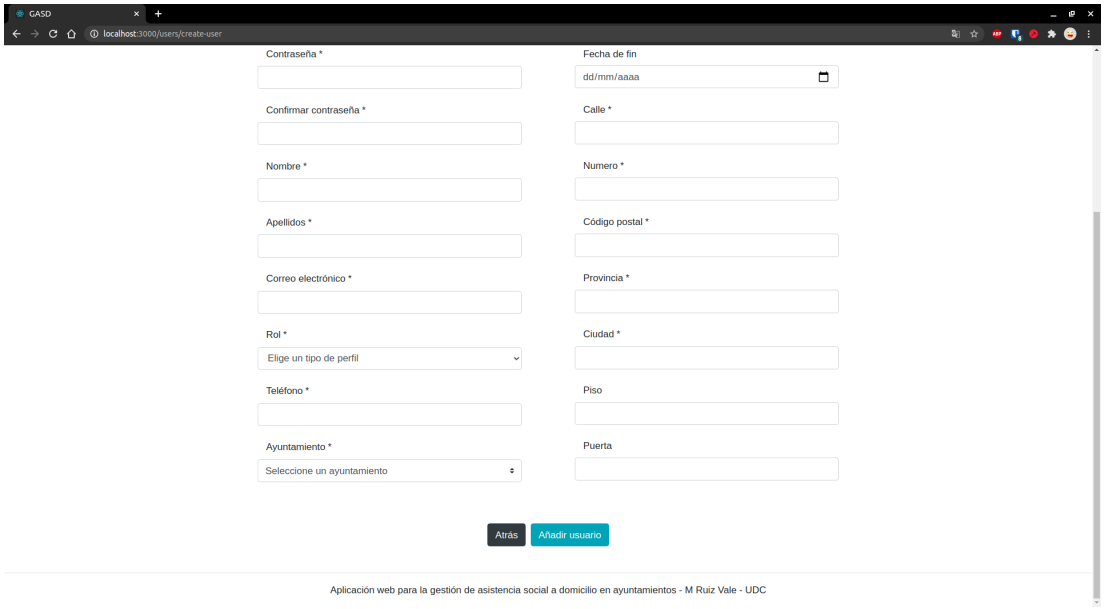


Figura A.13: Creación de usuario (2)

Se pueden visualizar todos los usuarios de la aplicación accediendo desde el menú principal presionando el botón *Ver usuarios*. La pantalla mostrada será la que se puede observar en la figura A.14. La cruz a la derecha de cada uno de los usuarios visibles en la tabla, eliminará el usuario pidiendo antes una confirmación y el botón ver detalles mostrará los datos de cada usuario (información y direcciones almacenadas), que podrán ser modificados.

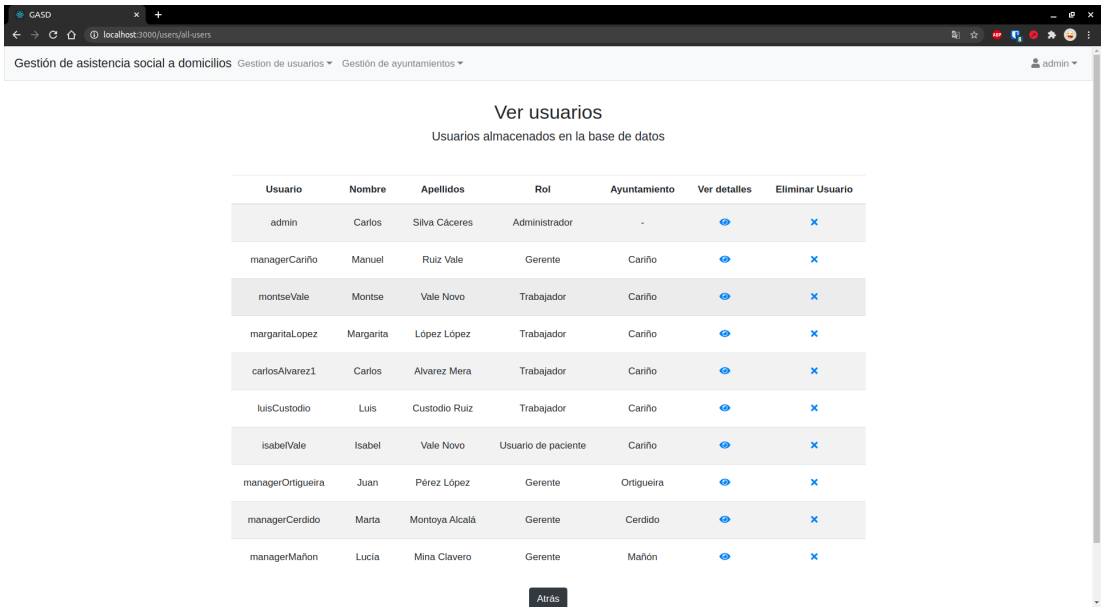


Figura A.14: Visualización de usuarios

A.3.3 Funcionalidades del gerente

Cuando se accede al sistema con un usuario del tipo gerente, el menú principal que se obtiene es el que se puede observar en la figura A.15

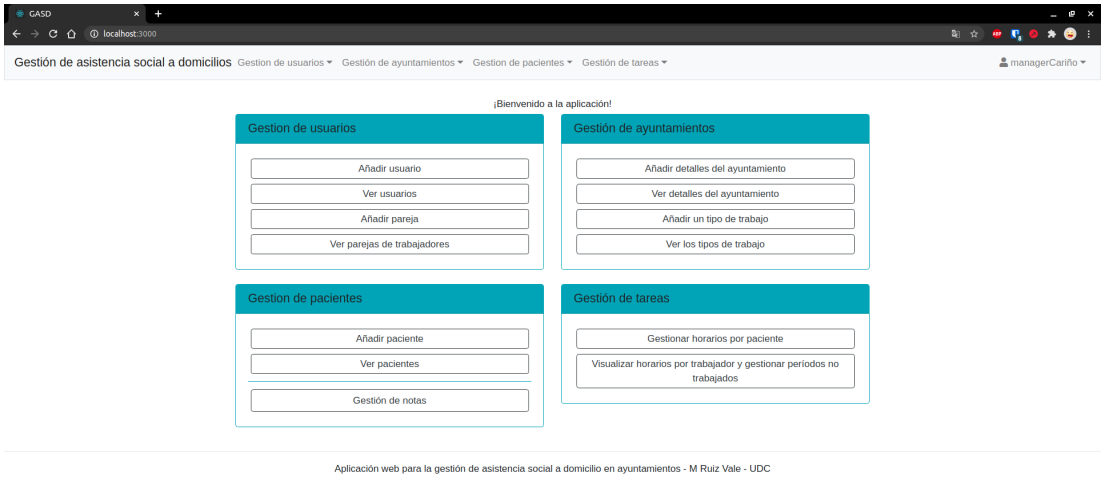


Figura A.15: Menú principal gerente

Gestión de ayuntamientos y sus datos

El mánager es capaz de crear grupos de detalles, que representan información que es válida durante un tiempo estipulado, en el ayuntamiento al que pertenece. Se accede a la funcionalidad desde el menú principal haciendo clic en el botón *Añadir detalles de ayuntamiento*. Esto le llevará a la pantalla de la figura A.16 en la que se puede observar un formulario que creará el grupo de información, válida durante el tiempo estipulado.

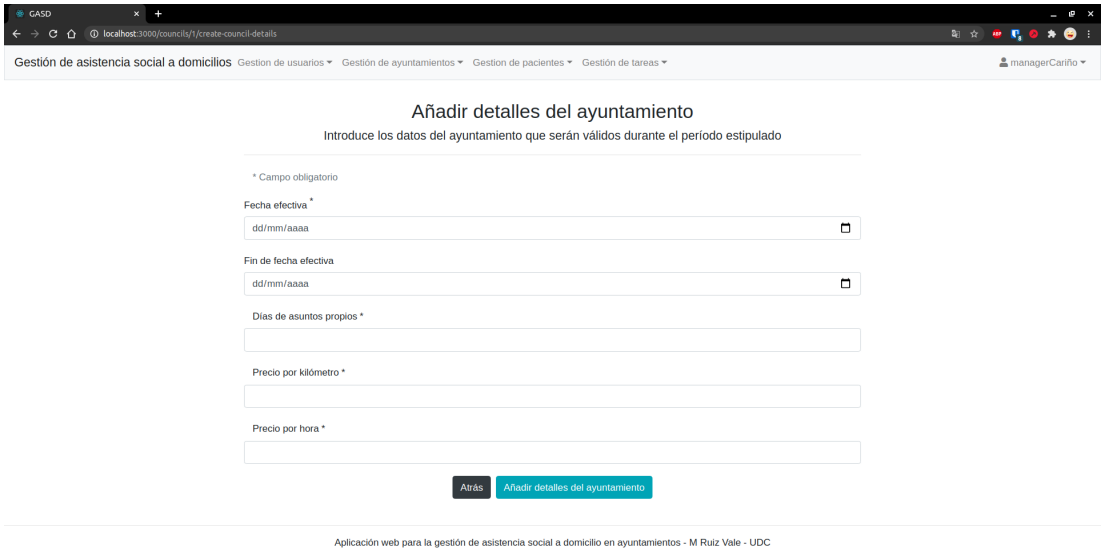


Figura A.16: Creación grupo detalles

Si se accede a visualizar el grupo de detalles del ayuntamiento, presionando el botón *Ver detalles del ayuntamiento*, se accede a pantalla observable en la figura A.17, que mostrará la información en vigor del ayuntamiento.

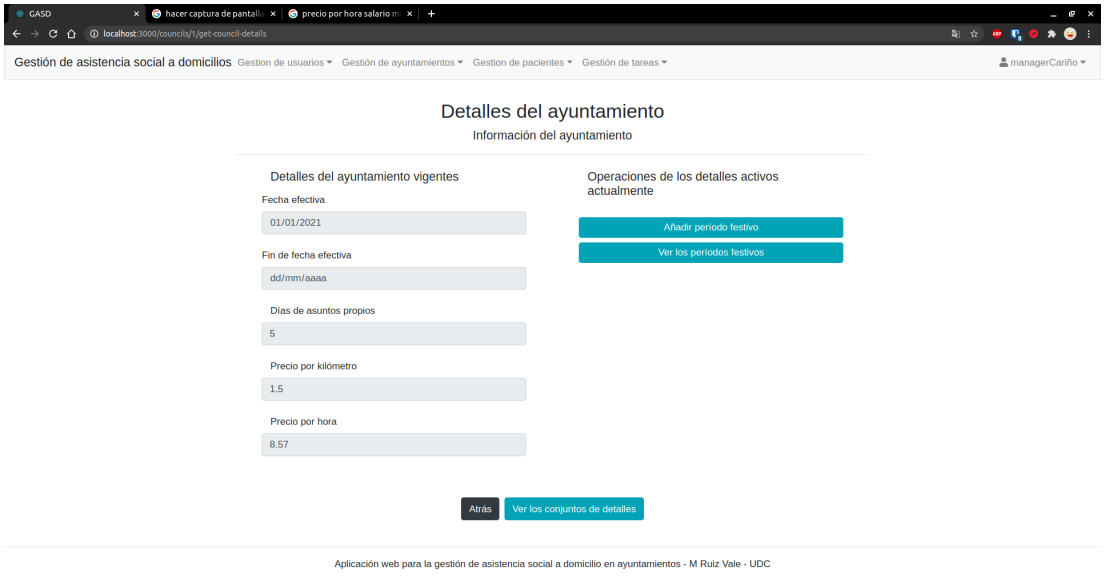


Figura A.17: Visualización de detalles en vigor

Se podrán añadir los períodos festivos que afectarán al conjunto de detalles en vigor, haciendo clic en el botón *Añadir período festivo* se mostrará un formulario donde se indicarán los

períodos que son festivos (fig. A.18). Si el período festivo está fuera de los límites temporales del conjunto de detalles al que pertenece, se mostrará un mensaje de error indicando que esta acción no es posible. Por otro lado, también se podrán visualizar todos los períodos almacenados en el sistema presionando el botón *Ver los períodos festivos*, que mostrará la lista que se puede observar en la figura A.19. Los períodos festivos podrán ser eliminados presionando el botón de la columna *Eliminar período*, que mostrará un aviso pidiendo confirmación.

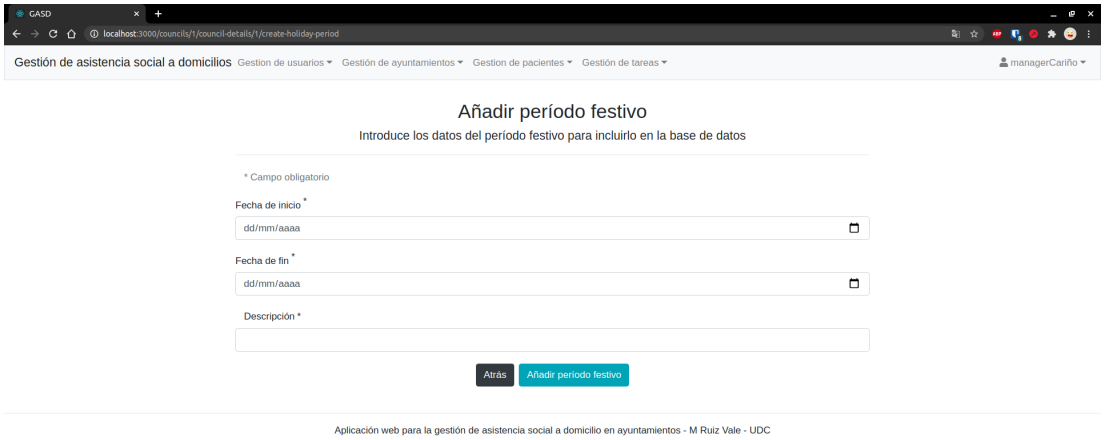


Figura A.18: Creación periodo festivo

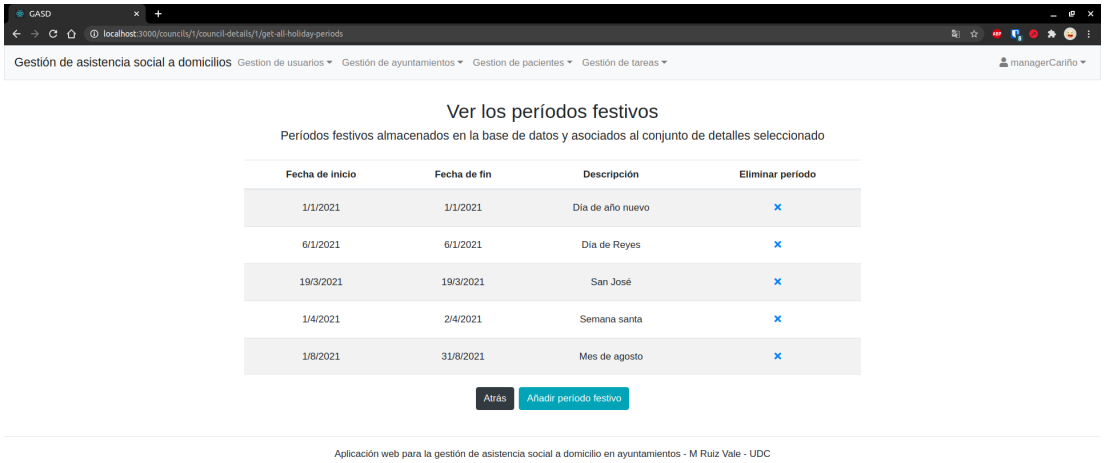


Figura A.19: Visualización de períodos festivos

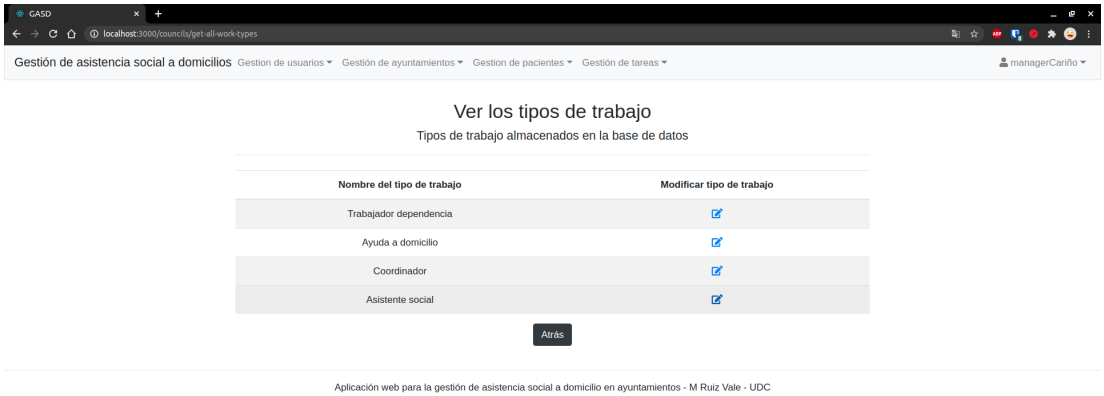
Desde la pantalla de la figura A.17, se puede presionar el botón *Ver los conjuntos de detalles*, que accede a la lista de todos los grupos de información del ayuntamiento. Presionando el botón de la columna *Actualizar detalles del ayuntamiento*, se accederá a un formulario que permite la modificación de los datos de ese conjunto.

Figura A.20: Visualización de conjuntos de detalles

Por otra parte, desde el menú principal, se puede acceder al formulario de creación de *tipos*

de trabajo, mostrado en la figura A.21. Los tipos de trabajo son necesarios para la creación de contratos, que se explicará más adelante. Si se presiona el botón *Ver los tipos de trabajo* se pueden observar los que están disponibles (fig. A.22) y modificarlos. Para realizar la modificación se muestra un formulario que permite modificar el valor.

Figura A.21: Creación de un tipo de trabajo



localhost:3000/councils/update-work-type/4

Figura A.22: Visualización de tipos de trabajo

Gestión de usuarios

El formulario de creación de usuario, accesible desde el menú principal en el botón *Añadir usuario*, es el observado en las figuras A.23 y A.24 en los que se introducirán los campos del nuevo usuario a crear y presionando el botón *Añadir usuario* se efectuará la creación. Un usuario gerente podrá crear los tipos de usuario: gerente, trabajador y usuario de paciente.

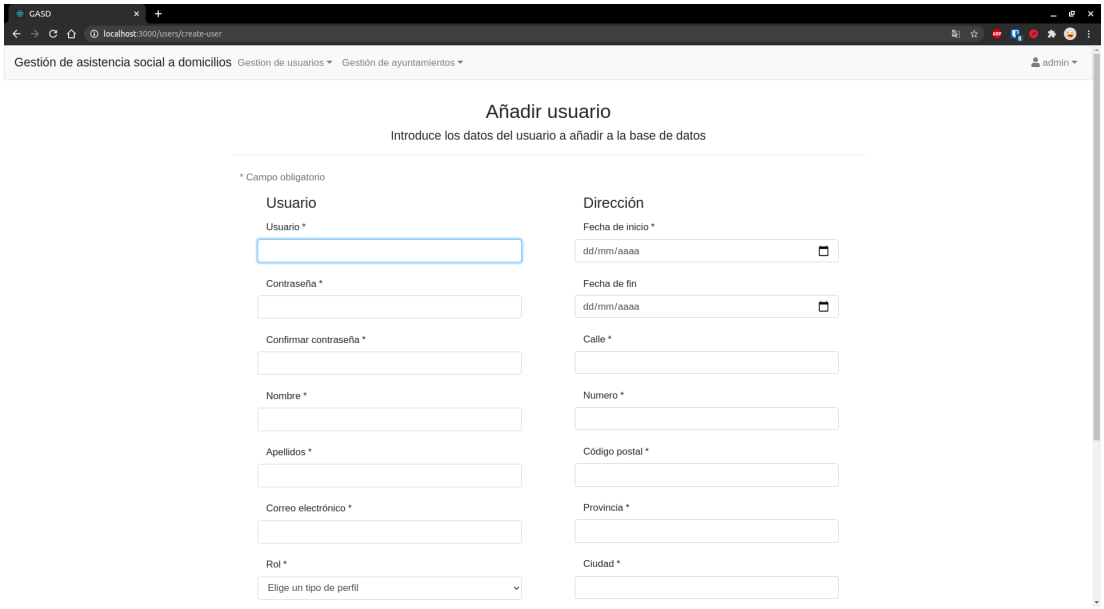


Figura A.23: Creación de usuario (1)

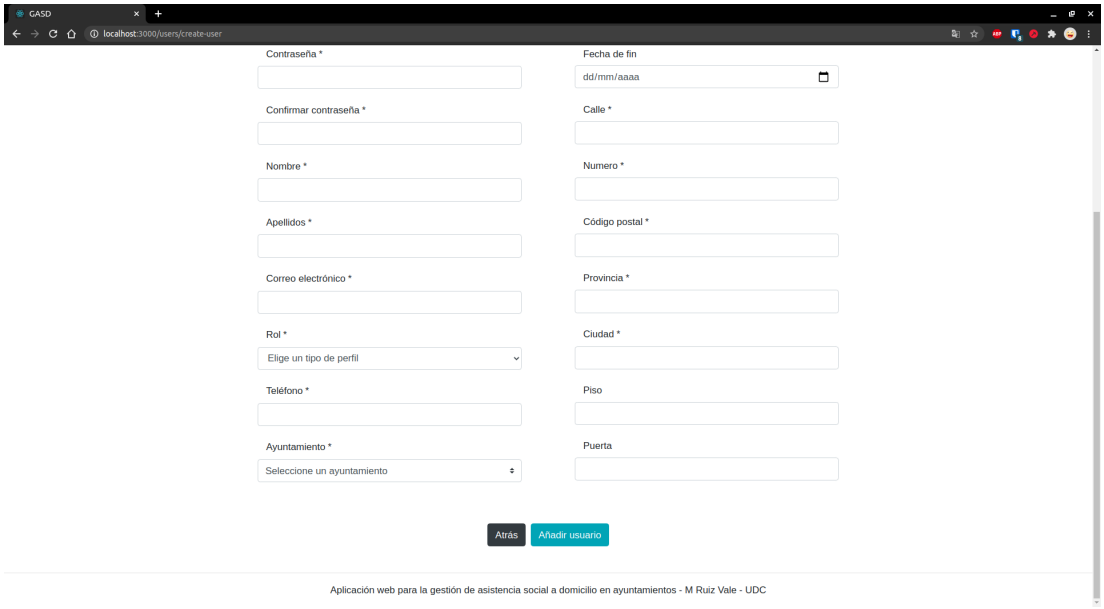


Figura A.24: Creación de usuario (2)

Se pueden visualizar todos los usuarios de la aplicación accediendo desde el menú principal presionando el botón *Ver usuarios*. La pantalla mostrada será la que se puede observar en la figura A.25. La cruz a la derecha de cada uno de los usuarios visibles en la tabla, eliminará el usuario pidiendo antes una confirmación y el botón ver detalles mostrará los datos de cada usuario, visibles en la figura A.26: información, direcciones añadidas y gestión de contratos (en caso de que el usuario sea un gerente o trabajador).

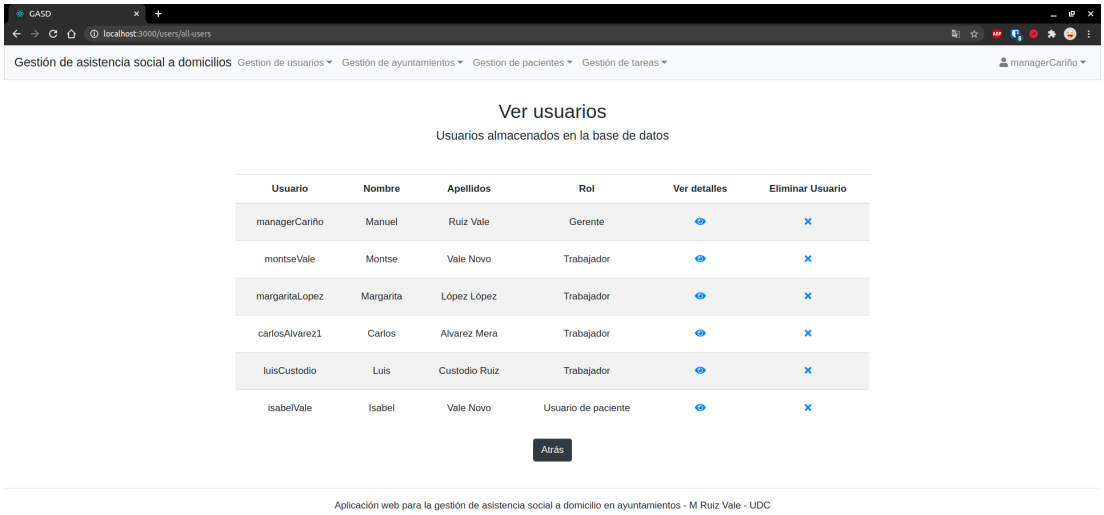


Figura A.25: Visualización de usuarios

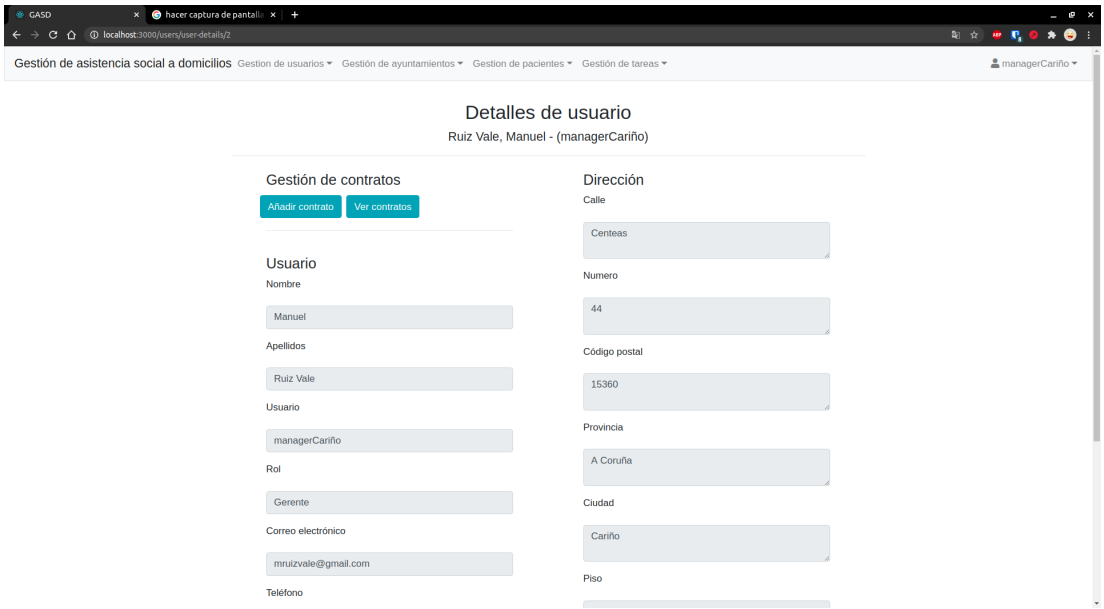


Figura A.26: Visualización de datos de usuario

La creación de contratos se realizará accediendo desde el botón *Crear contrato*, que muestra el formulario de creación, observable en la figura A.27. En él se deberá estipular una duración, un tipo de contrato (jornada completa o partida), un tipo de trabajo (debe estar creado anteriormente, si no se aviará con un mensaje de error) y las horas de trabajo diarias.

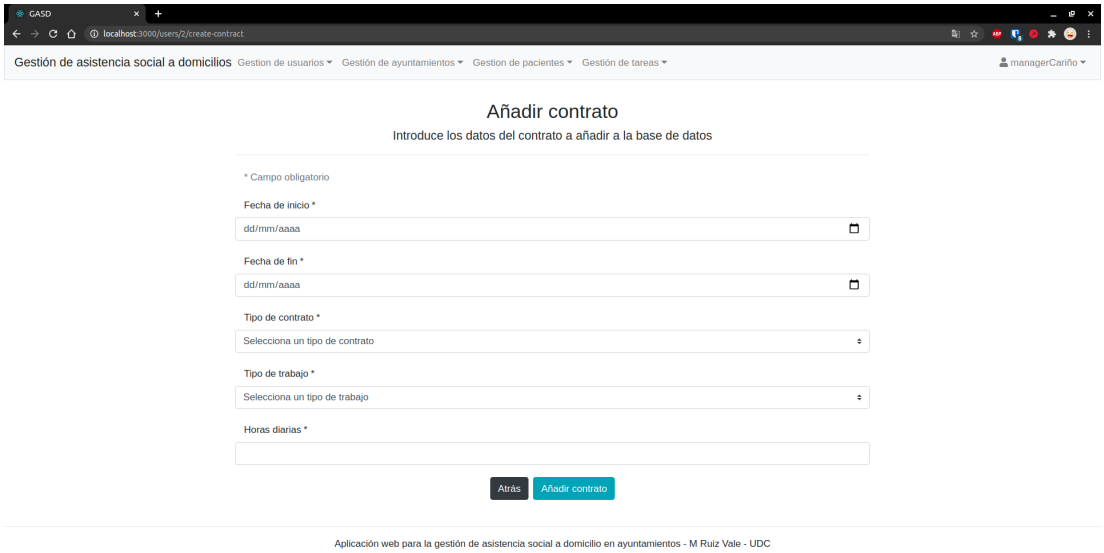


Figura A.27: Creación de contrato

Se podrá visualizar el contrato en vigor de un trabajador, desde la pantalla A.26 se debe presionar el botón *Ver contrato*, que mostrará el contrato en vigor del usuario (fig. A.28). Desde esta pantalla se podrá modificar el contrato, haciendo posible su extensión en el tiempo o modificando alguno de los campos (fig. A.29). Desde la pantalla A.28, se podrá acceder al registro de todos los contratos de un usuario, como se puede observar en la figura A.30

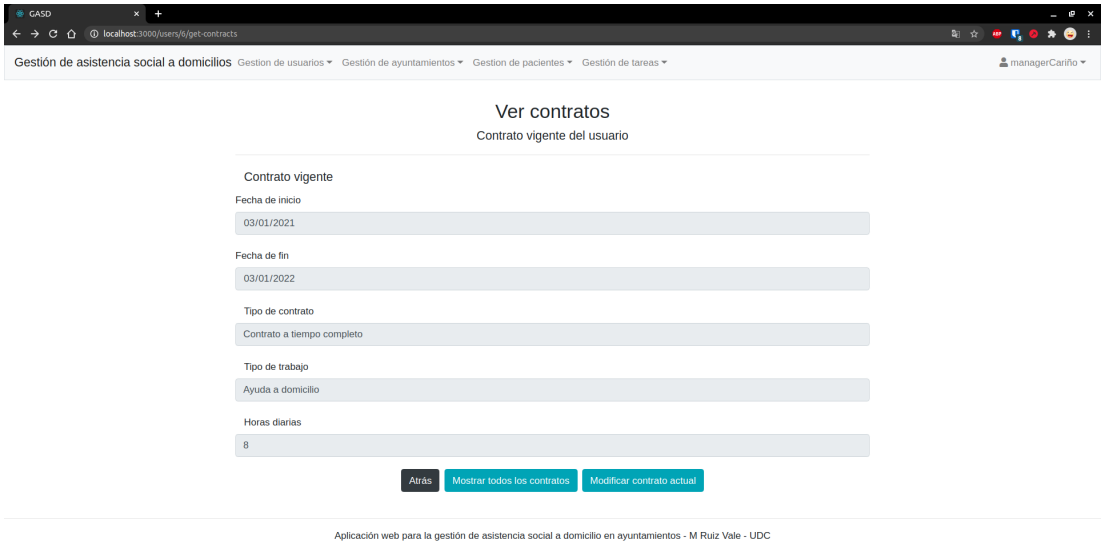


Figura A.28: Contrato en vigor

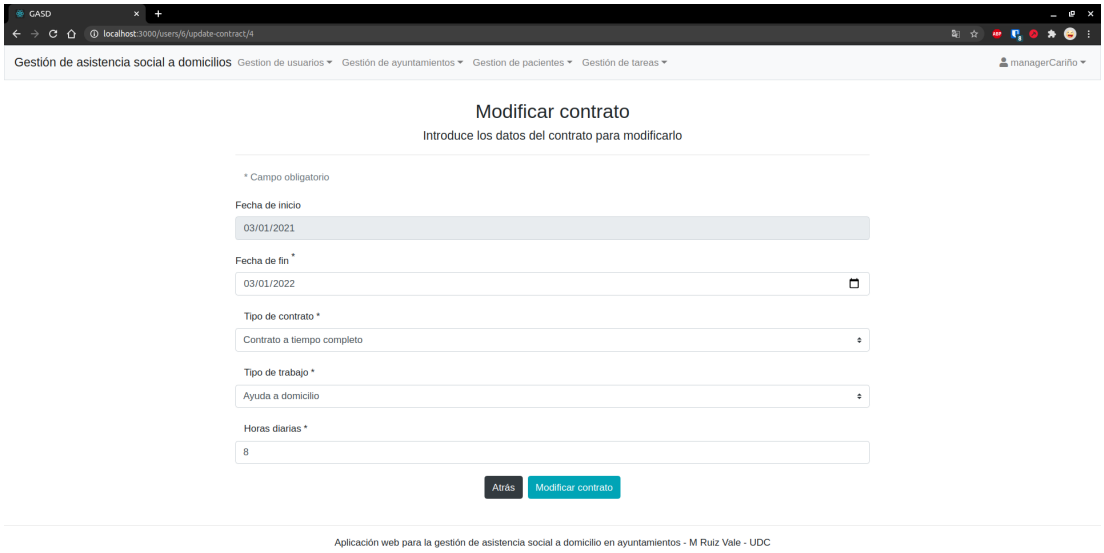


Figura A.29: Modificación de contrato

Figura A.30: Visualización registro contratos

Desde el menú principal se puede acceder a la creación de parejas de trabajadores, presionando el botón *Añadir pareja* se mostrará un formulario (fig. A.31) en el que se deberán indicar los dos trabajadores que forman la pareja y a partir de qué momento lo van a hacer. Se podrá dejar el campo fecha de fin vacío, para indicar que hasta el momento, la pareja tiene duración indefinida.

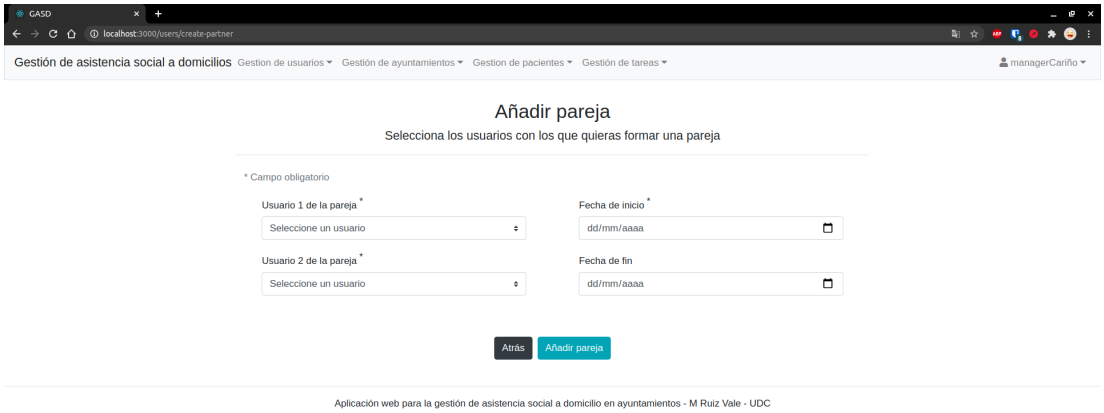


Figura A.31: Formulario creación de pareja de trabajadores

Si se presiona el botón *Ver parejas* del menú principal, se podrá acceder a la visualización de todas las parejas de trabajadores del ayuntamiento al que pertenece el gerente (fig. A.32). Con la posibilidad de modificar alguno de sus datos, presionando el botón de la columna *Actualizar Pareja* (fig. A.33), o eliminarla, presionando el botón con la cruz de la columna *Eliminar Pareja*, que mostrará una alerta de confirmación para el borrado.

Figura A.32: Visualización de las parejas de un ayuntamiento

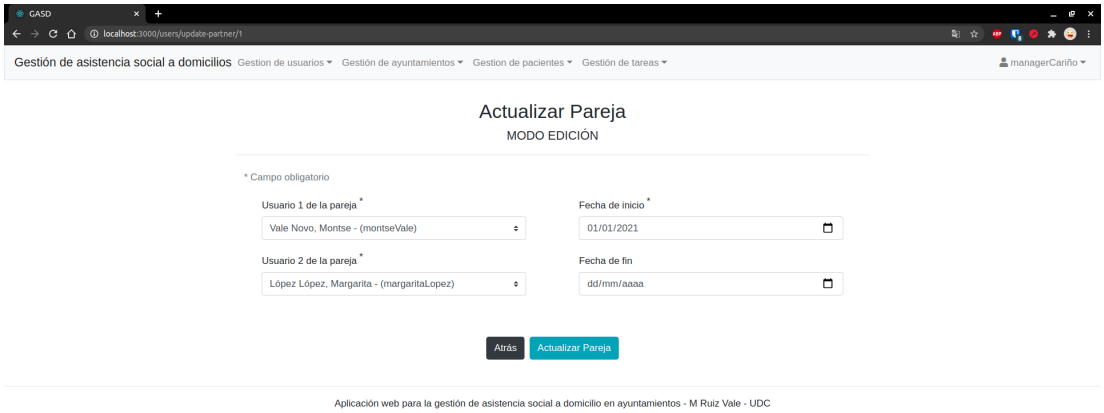


Figura A.33: Formulario modificación de pareja de trabajadores

Gestión de pacientes y notas

Para la creación de pacientes se debe presionar el botón *Añadir paciente* del menú principal. Esto habilitará el formulario de creación visible en las figuras A.34 y A.35. El formulario se debe rellenar con los datos del paciente, el grado de dependencia que tiene otorgado según la ley 39/2006 [1] y su última reforma en el año 2012[2], las horas semanales asignadas y, si se desea o se tiene disponible, un usuario asociado del tipo usuario de paciente. La creación de la dirección del paciente se realiza de forma análoga a los usuarios, con la única diferencia de que se debe indicar si su residencia se encuentra dentro del casco urbano o no.

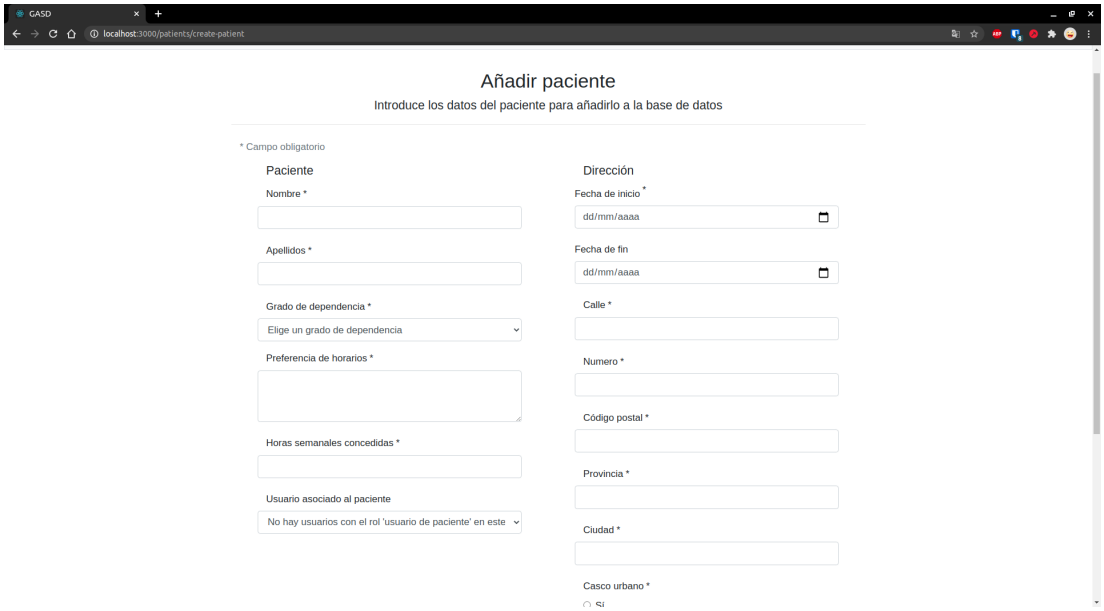


Figura A.34: Formulario creación de paciente (1)

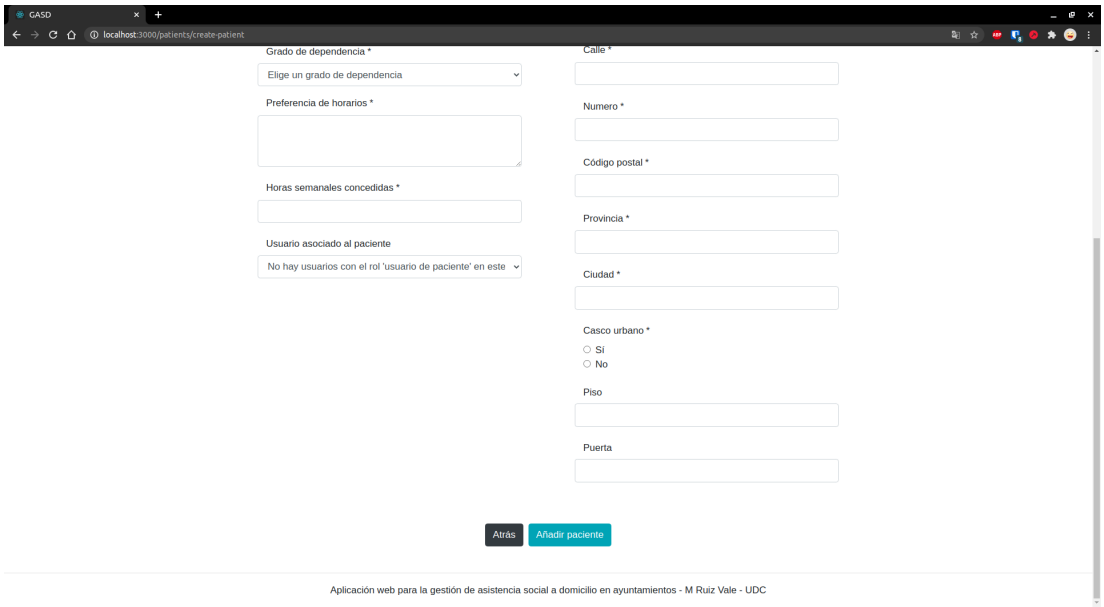


Figura A.35: Formulario creación de paciente (2)

La visualización de los pacientes se realiza presionando el botón *Ver pacientes* del menú principal. Esto muestra una lista de todos los pacientes almacenados en el ayuntamiento del gerente (fig. A.36). Es posible el acceso a toda la información del paciente presionando el botón de la columna *Ver detalles* que mostrará la pantalla representada en las figuras A.37 y A.38. Haciendo posible la modificación de cualquiera de los datos mediante el botón *Actualizar*

Paciente y también la gestión de las direcciones del paciente mediante los botones *Añadir nueva dirección de paciente* y *Ver todas las direcciones.*

Figura A.36: Visualización de los pacientes de un ayuntamiento

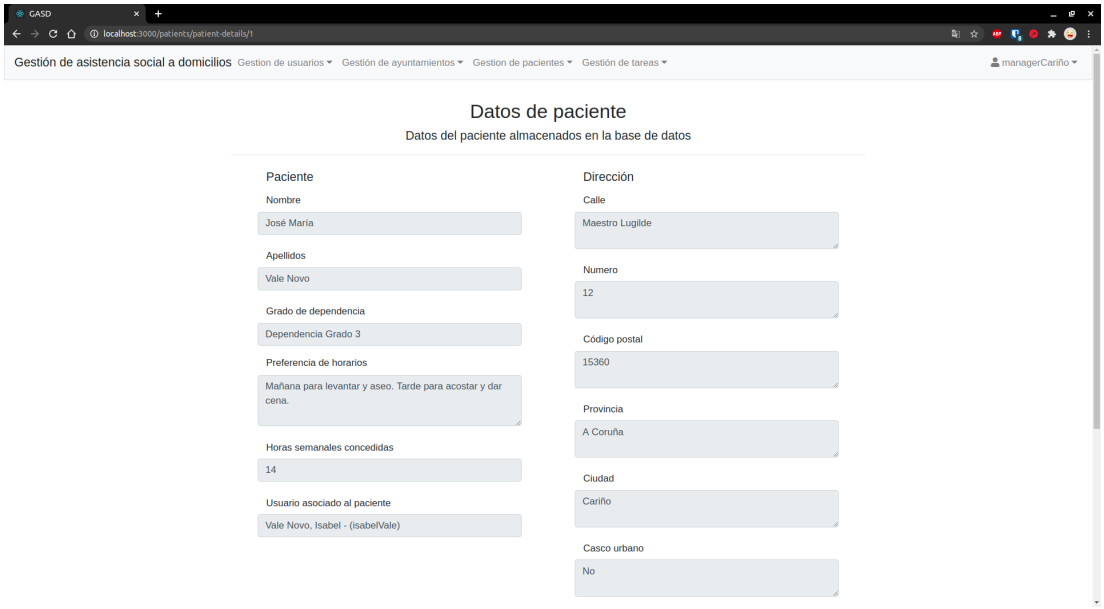


Figura A.37: Visualización datos de paciente (1)

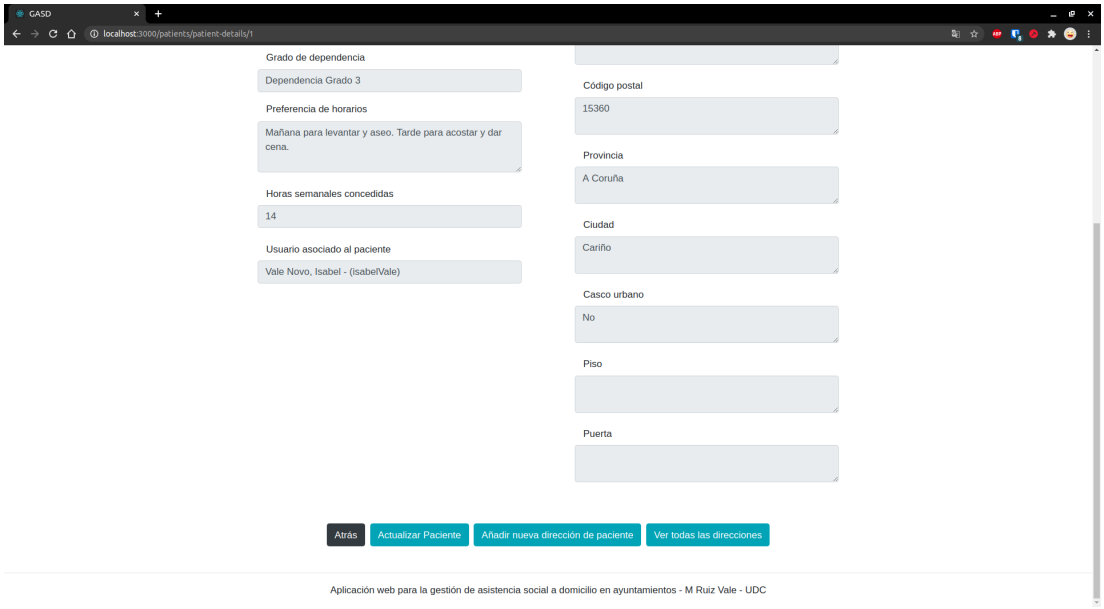


Figura A.38: Visualización datos de paciente (2)

La gestión de notas de los pacientes se realiza a través del menú principal, presionando el botón *Gestionar notas*. Esto muestra una lista con todos los pacientes del ayuntamiento y dos botones por cada fila de paciente, *Añadir nota* y *Ver todas las notas de un paciente*. Presionando el primero se accede al formulario de creación de nota, dejando seleccionar visibilidad pública o privada para la misma (fig. A.39). El segundo botón lleva a la funcionalidad de mostrar todas las notas (públicas y privadas) de un paciente (fig. A.40).

Figura A.39: Formulario de creación de notas

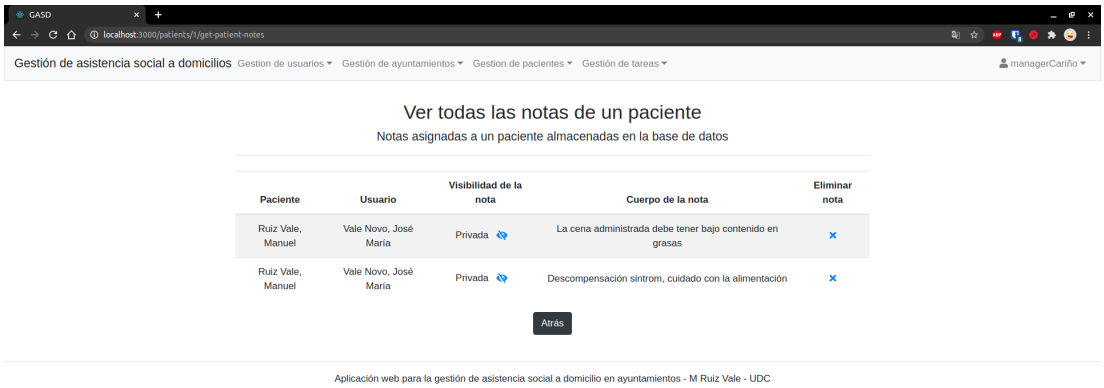


Figura A.40: Visualización de notas

Una vez en la pantalla de la figura A.40, se puede modificar la visibilidad de la misma presionando el botón de la columna *Visibilidad de la nota*. Esto mostrará una alerta que se debe confirmar si se desea llevar a cabo la modificación (fig. A.41). Como la acción la está ejecutando un gerente, también se podrá eliminar cualquier nota presionando el botón de la columna *Eliminar nota*, que solicitará confirmación.

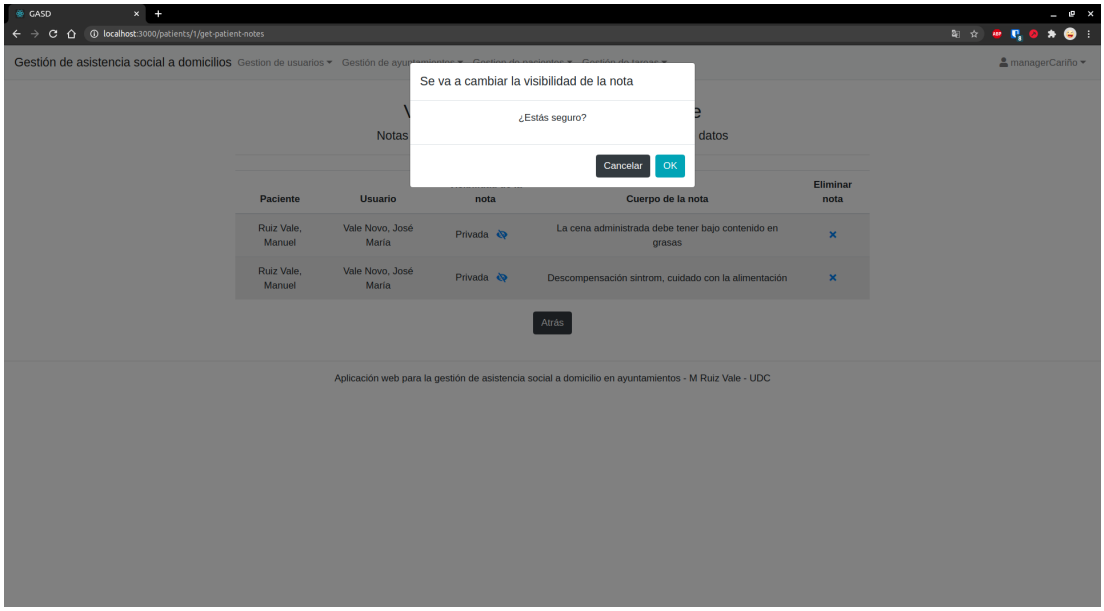


Figura A.41: Cambiar visibilidad de nota

Gestión de horarios de pacientes

Para acceder a la gestión de horarios se hará desde el menú principal, presionando el botón *Gestionar horarios por paciente* (fig. A.42). Esto llevará a un menú de selección del paciente a gestionar. Presionando el botón de la columna *Ver horarios* se accede a la vista de calendario semanal, como se puede observar en la figura A.43.

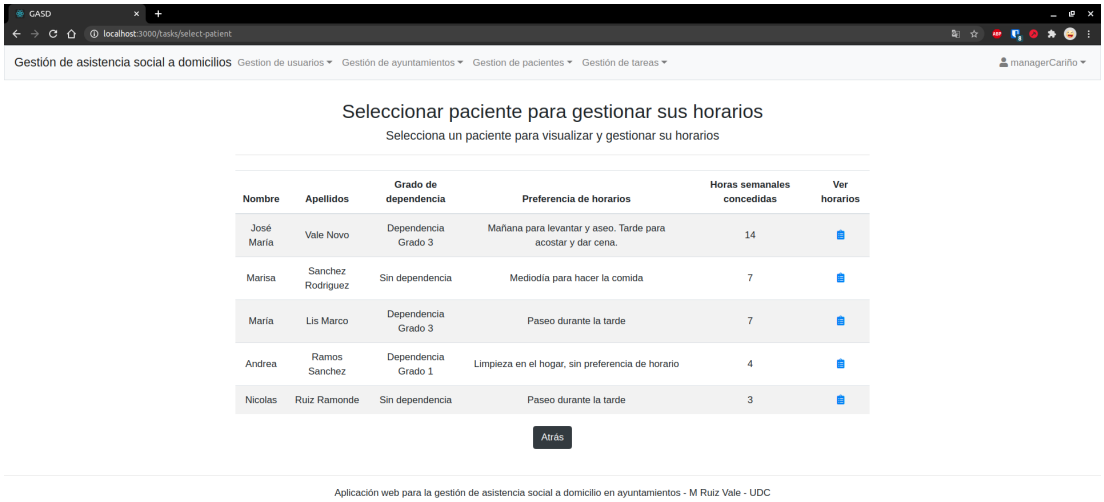


Figura A.42: Elección de paciente para gestionar horarios

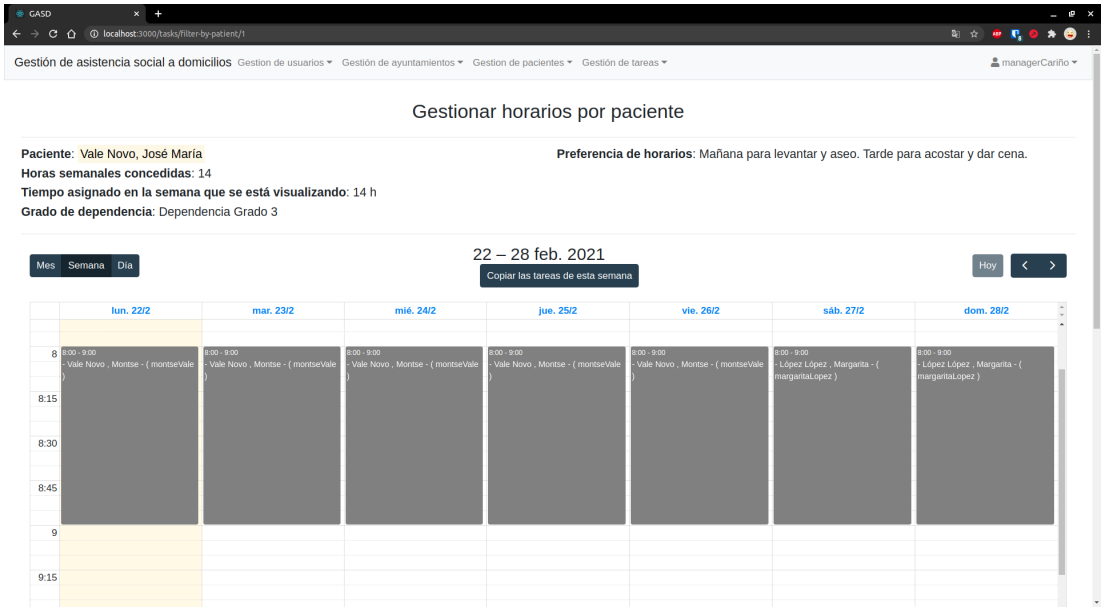


Figura A.43: Horario de paciente

En la pantalla de gestión de horario de un paciente, se pueden observar algunos datos relevantes del paciente, incluidas las preferencias horarias, las horas concedidas y el tiempo que tiene asignado en la semana de visualización.

Para crear una tarea (fig. A.44), basta con pinchar y arrastrar para establecer la duración de la tarea. Una vez se suelte el botón del clic, se mostrará un formulario en el que se debe indicar el trabajador. En campo de trabajador se muestran **únicamente** los trabajadores libres durante la duración de la tarea a crear. Además junto al nombre de cada uno de los trabajadores, se muestran las horas que ya tienen asignadas durante la semana de visualización. Una vez presionado el botón *Crear Tarea*, aparecerá la tarea con un fondo de color gris en el horario, indicando que está en estado pendiente.

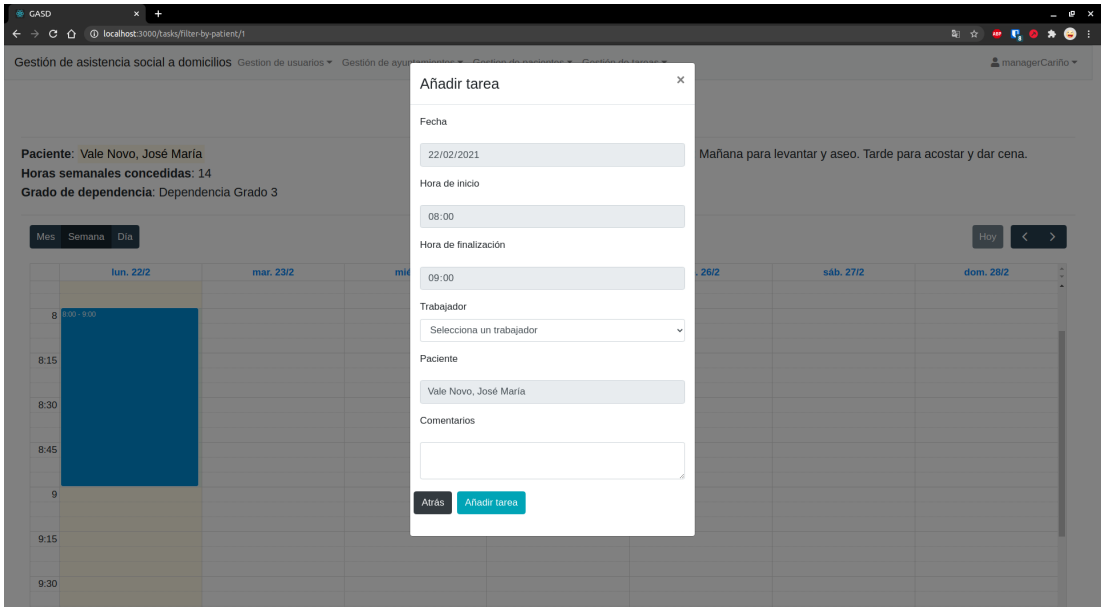


Figura A.44: Crear nueva tarea

Si se presiona sobre alguna de las tareas creadas, se accede a los datos de la misma, como se puede observar en las figuras A.45 y A.46, que indican la duración, el estado de la tarea, el trabajador el paciente y el campo de comentarios de la tarea. Además se muestra un botón que, si es presionado, modifica el estado de la tarea a completado, solicitando confirmación para realizar la actualización (fig. A.47). Una vez confirmada la acción, la tarea pasaría a estado completado, como se puede visualizar en la figura A.48

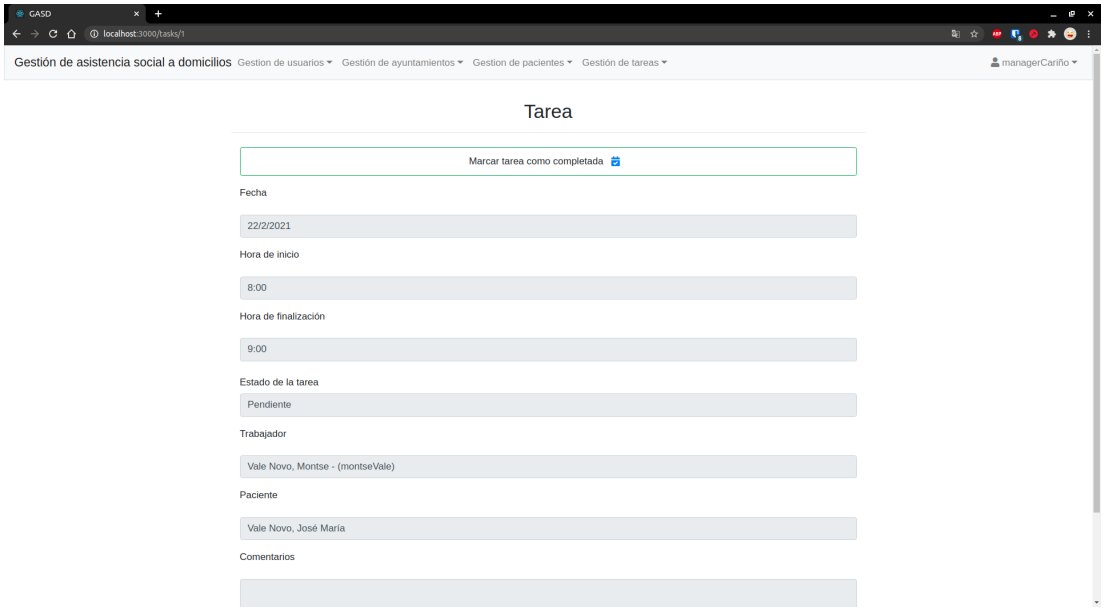


Figura A.45: Datos tarea (1)

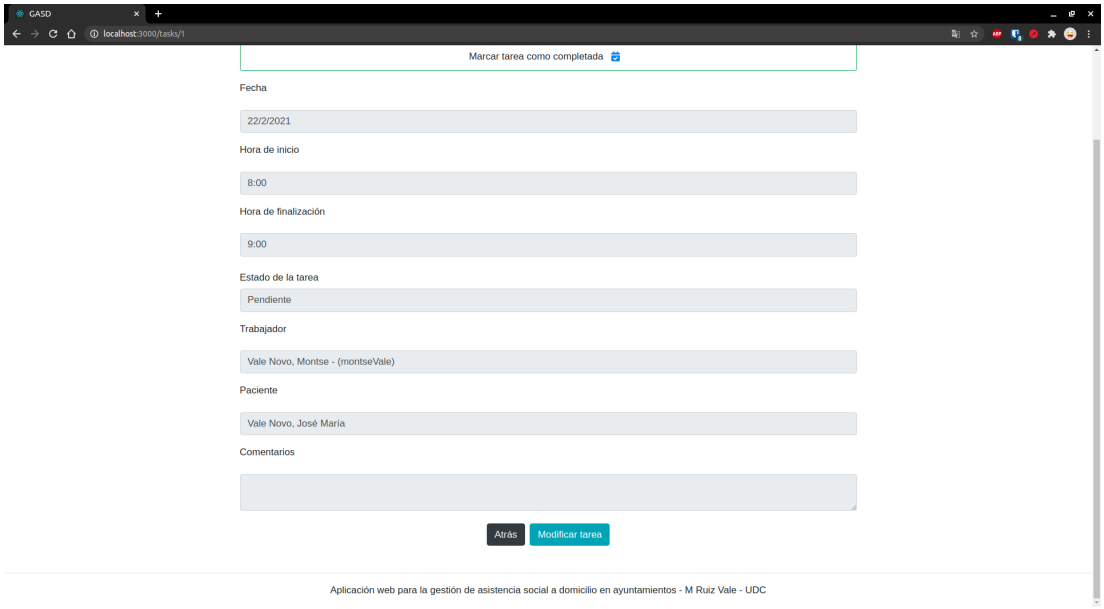


Figura A.46: Datos tarea (2)

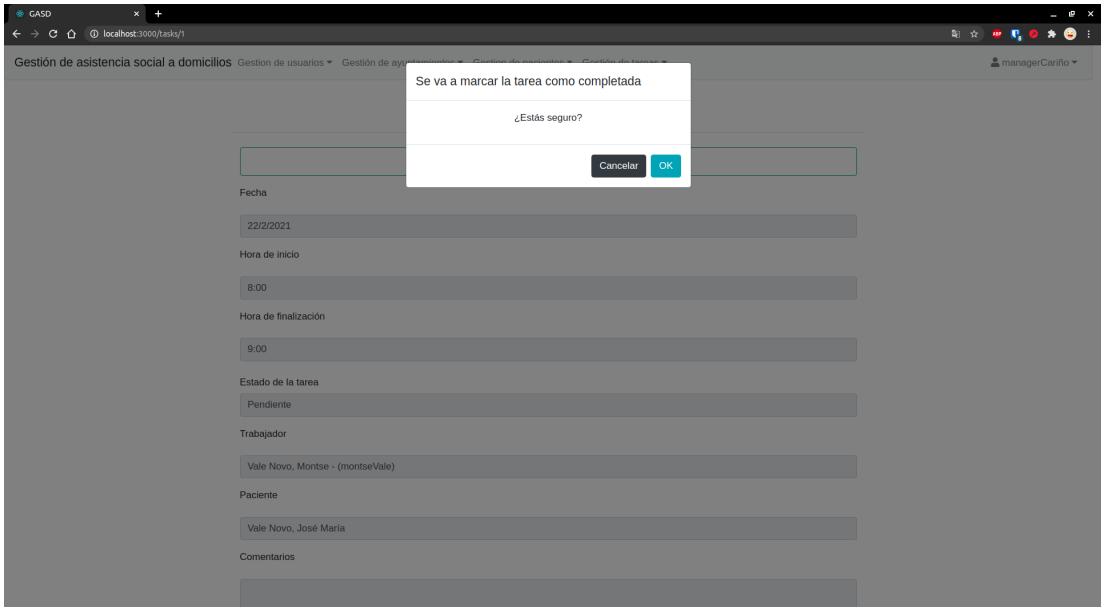


Figura A.47: Confirmación marcar tarea como completada

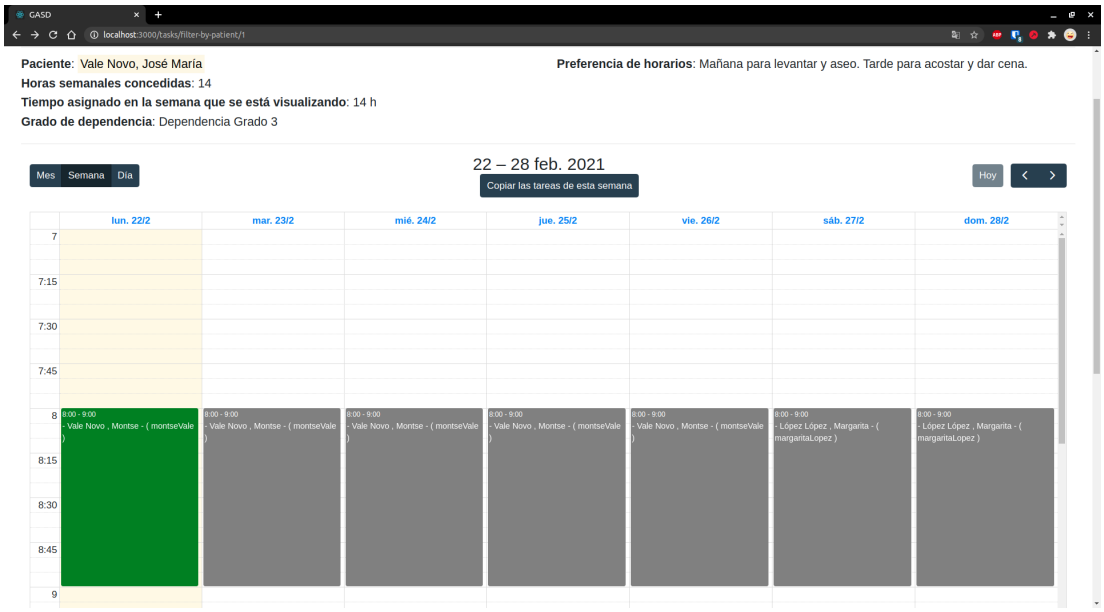


Figura A.48: Tarea completada

Si se accede al formulario de modificación de la tarea, mediante el botón *Actualizar tarea* de la pantalla representada en la figura A.46, se podrán modificar cualquiera de los campos de la tarea. Si se marca una tarea como *cancelada*, el resultado en la pantalla de gestión se verá reflejado con la tarea coloreada de rojo, como se puede observar en la figura A.49

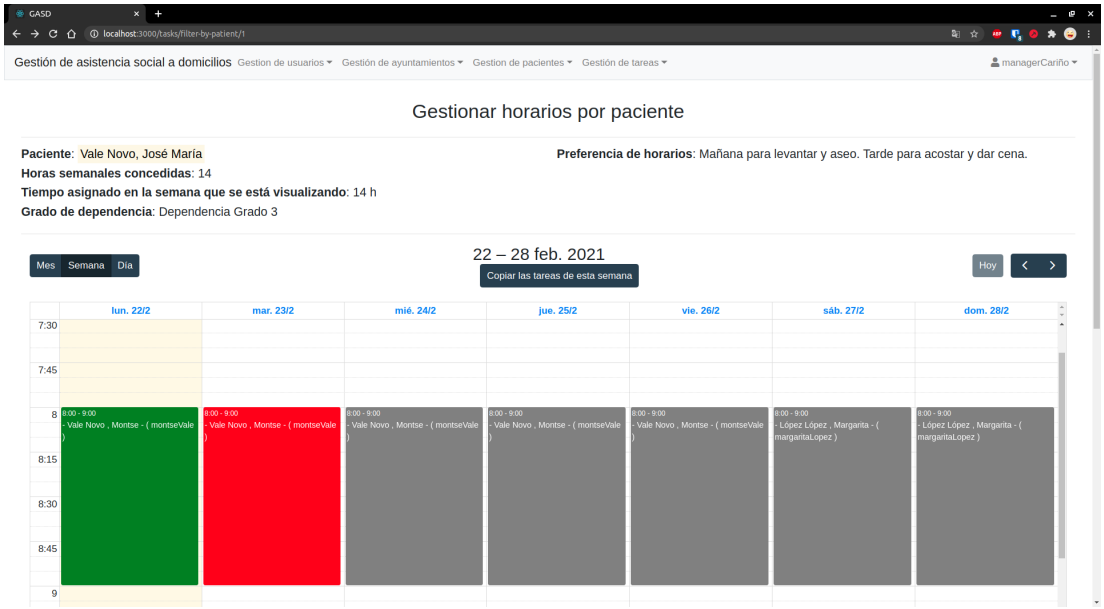


Figura A.49: Tarea cancelada

Para facilitar el trabajo de creación de tareas y como estas suelen repetirse semanalmente, el botón *Copiar las tareas de esta semana*, realiza una copia de las tareas de la semana que se está visualizando en la semana que se le indique. Las nuevas tareas, naturalmente, estarán en estado *pendiente*. Se puede observar en la figura A.50, como se muestra un formulario en el que se indicará el numero de semanas que se quiere desplazar la copia, respecto a la que se está visualizando. Una vez confirmado, se indicará que la copia de tareas se ha realizado (fig. A.51) y se puede comprobar como ya están disponibles (fig. A.52).

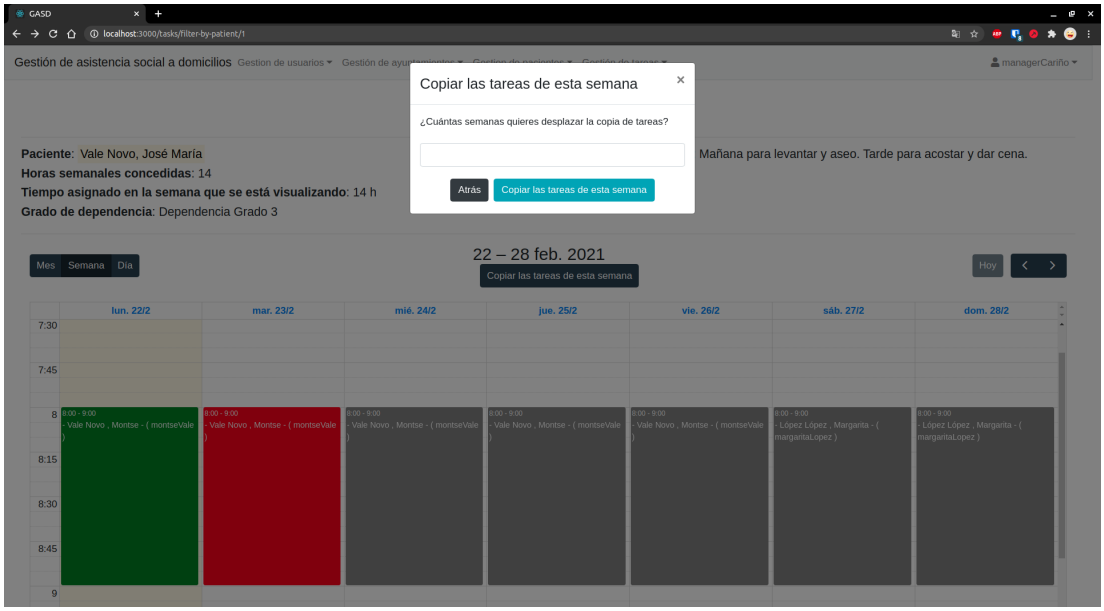


Figura A.50: Copia de tareas (1)

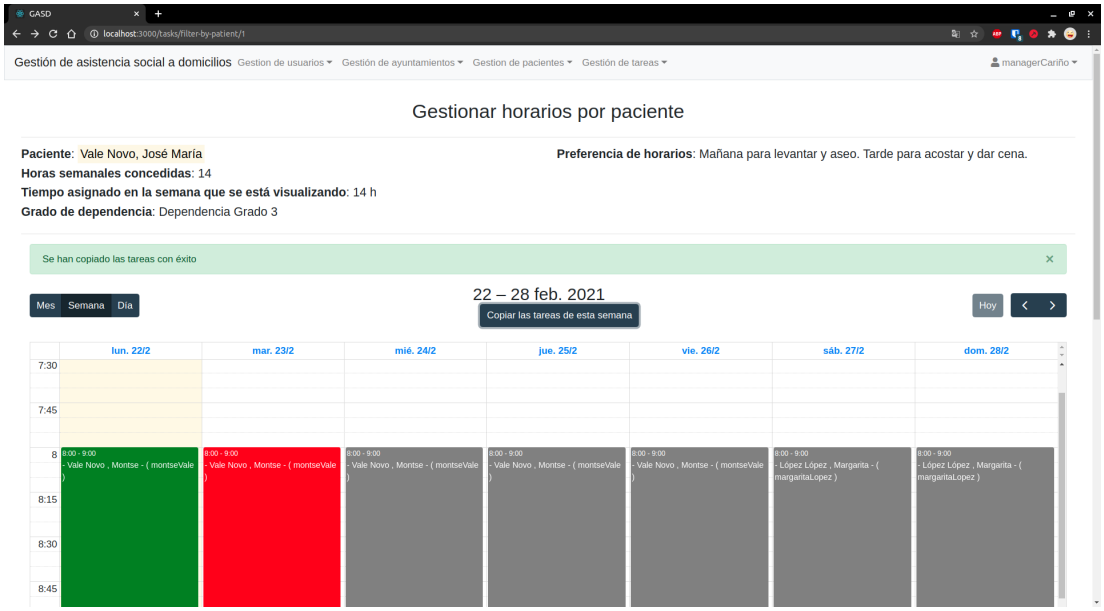


Figura A.51: Copia de tareas (2)

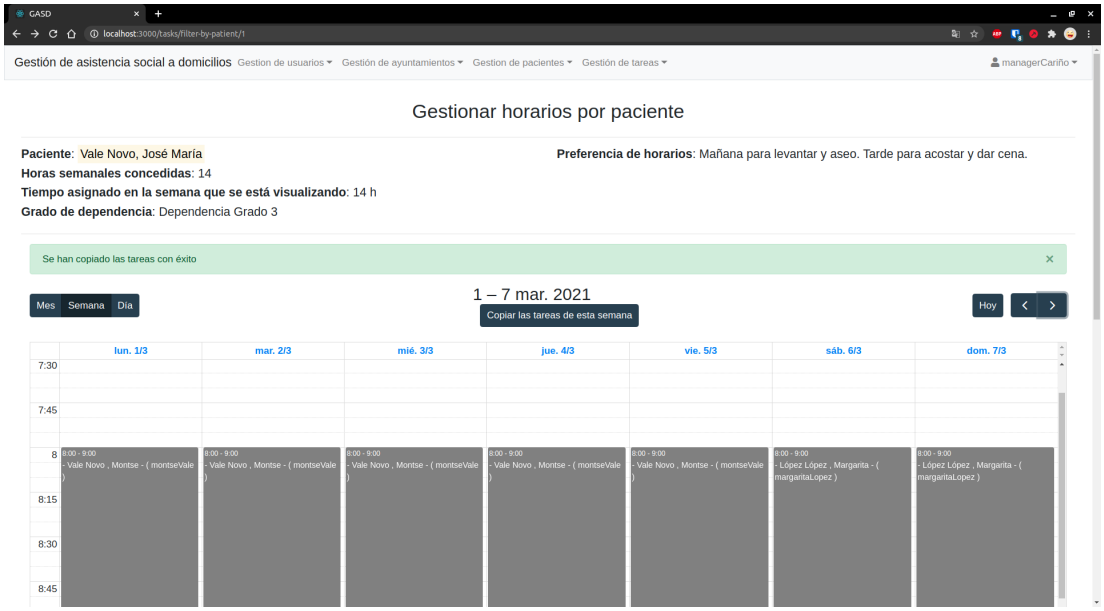


Figura A.52: Copia de tareas (3)

Gestión de horarios y días no trabajados de trabajadores

Desde el menú principal se puede acceder a la visualización de horarios de trabajadores y sus períodos no trabajados, mediante el botón *Visualizar horarios por trabajador y gestionar períodos no trabajados* se accede a la lista de selección de trabajador, mostrada en la figura A.53. En ella se puede acceder a la visualización de horarios, mediante el botón de la columna *Ver horarios* o a la gestión de períodos no trabajados mediante la columna homónima.

Figura A.53: Selección de trabajador

Comenzando con la visualización de horarios, esta se realiza en un calendario similar al de la gestión de horas de un paciente (fig. A.54), pero que no permite la creación de tareas, ya que estas se realizan siempre desde los pacientes. En el horario se pueden ver las tareas asignadas y acceder a ellas para marcarlas como completadas o realizar alguna modificación.

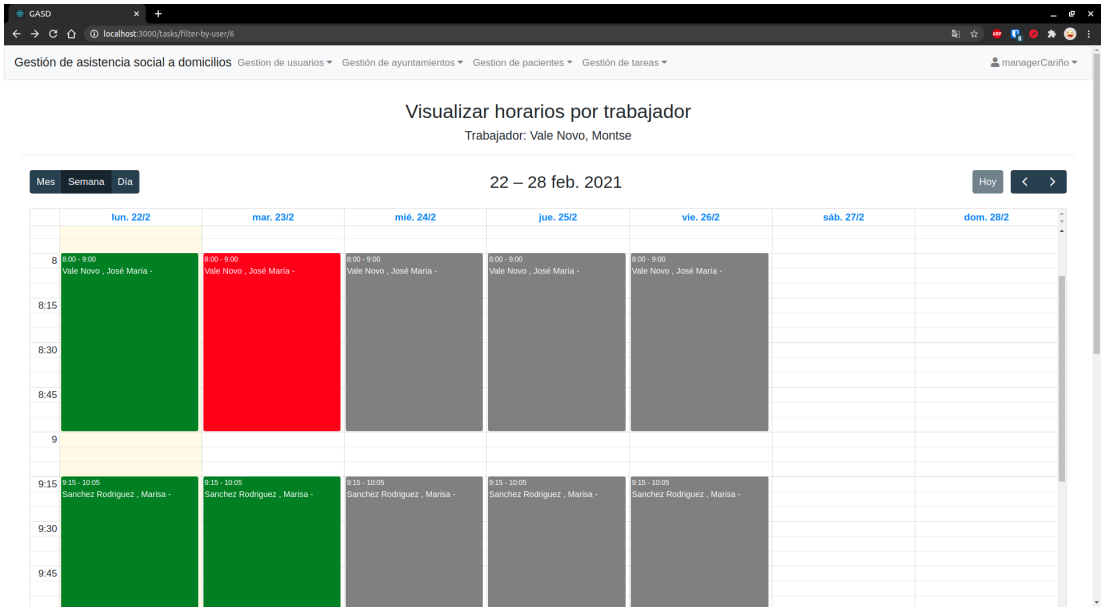


Figura A.54: Horario de trabajador

La gestión de períodos no trabajados se realiza desde un calendario en formato mensual

(fig. A.55), aunque todos los calendarios pueden modificar la vista y navegar entre las distintas fechas. En la vista indica el total anual de días no trabajados, para llevar una cuenta de la cantidad de días que un trabajador no ha podido trabajar. Si se quiere añadir un período no trabajado como por ejemplo una baja de 15 días, se deberá hacer clic y arrastrar para indicar la duración. Cuando se suelta el botón aparece un formulario (fig. A.56) que permite precisar más la duración, por si por ejemplo fuese únicamente un día asuntos propios o una visita al médico durante la mañana. Se debe rellenar el campo del tipo de período (pudiendo elegir entre baja, vacaciones, asuntos propios y otros), y si se desea se puede añadir una descripción que complete la información del período.

Se puede acceder a la información de un período haciendo clic en él. Esto muestra los datos del período en cuestión (fig. A.57) y permite modificarlo utilizando el botón *Modificar período no trabajado* que habilita un formulario en el que se pueden actualizar los datos.

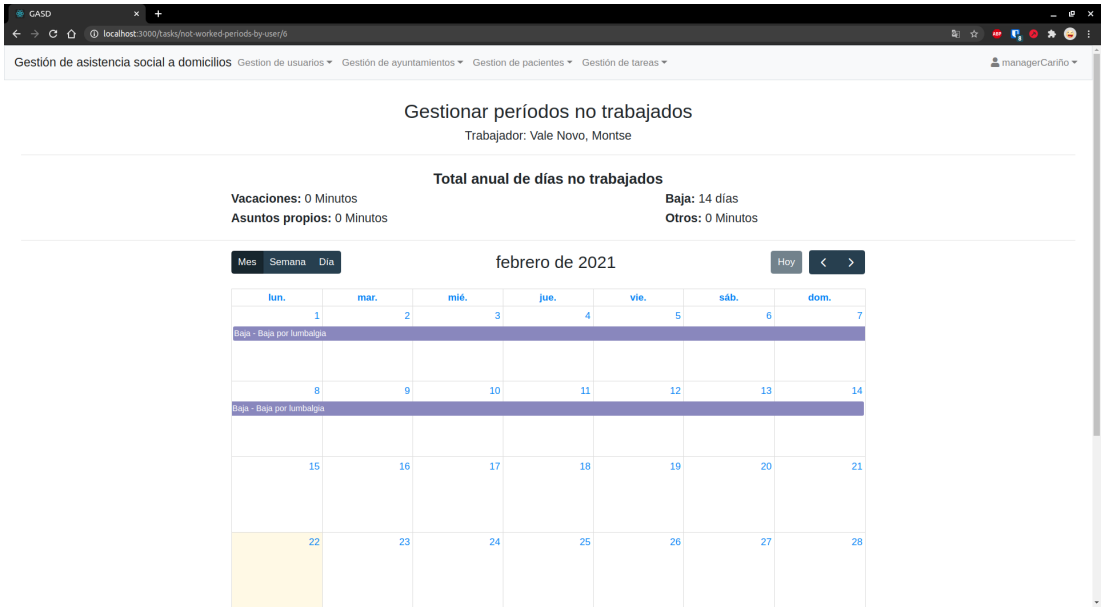


Figura A.55: Períodos no trabajados

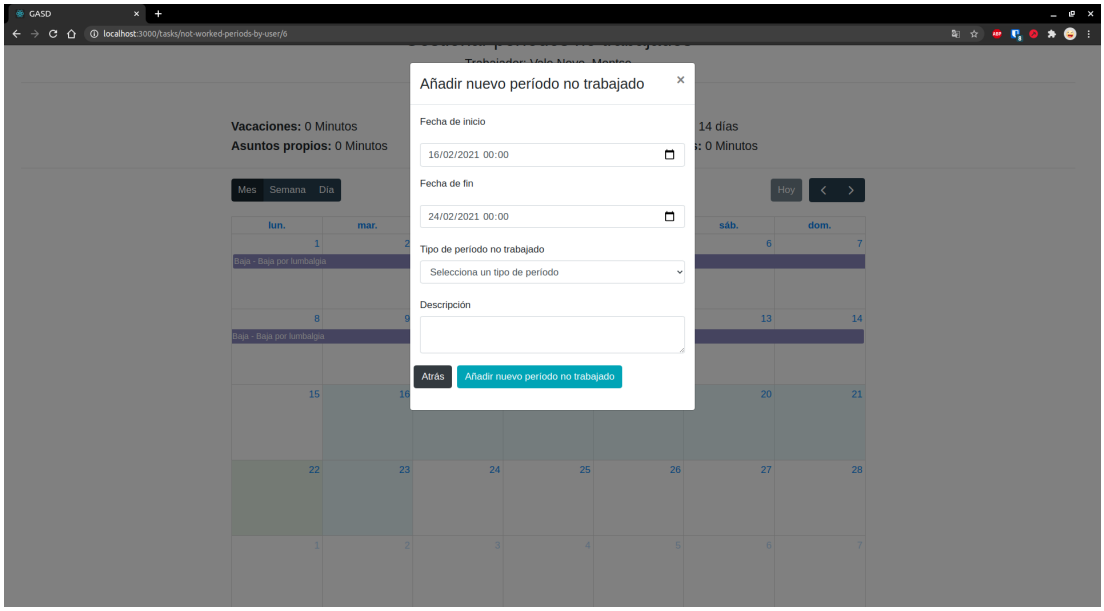


Figura A.56: Creación de período no trabajado

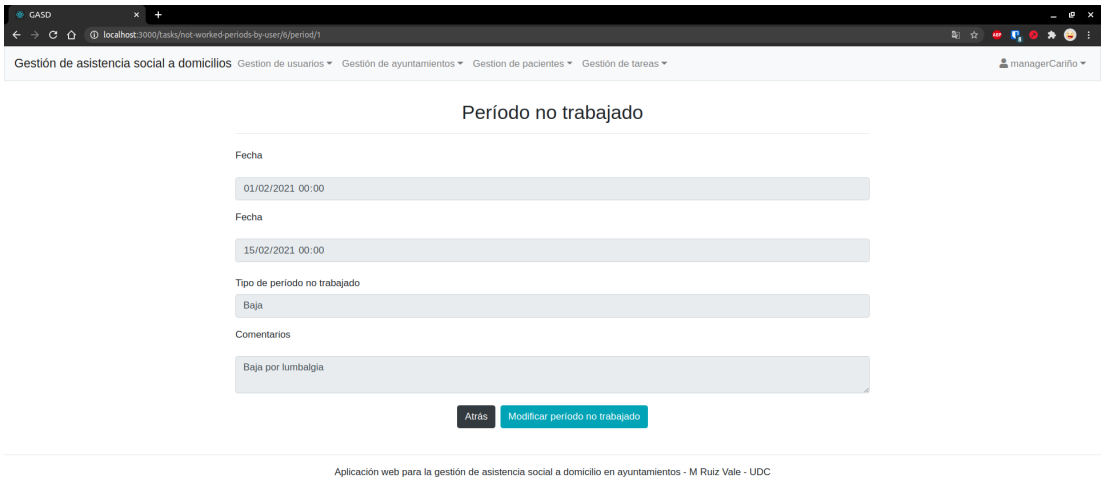


Figura A.57: Información de un período no trabajado

A.3.4 Funcionalidades del trabajador

Si se accede como trabajador a la aplicación, el menú principal mostrado en ella es el visible en la figura A.58.

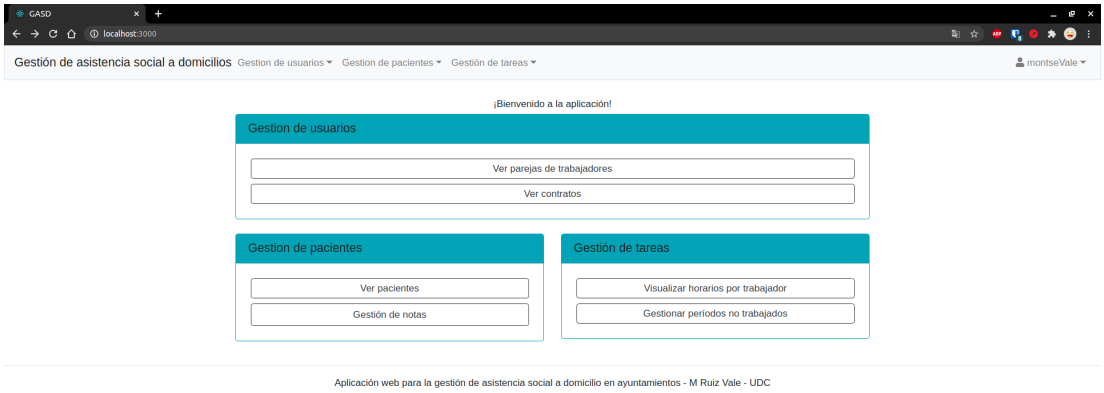


Figura A.58: Menú principal trabajador

Desde el menú principal, presionando el botón *Ver parejas de trabajadores*, el trabajador tiene acceso a la visualización de las parejas en las que forma parte (fig. A.59).

Figura A.59: Visualización parejas de trabajador

El trabajador puede acceder al contrato que está vigente en el momento de realizar la consulta, como se puede observar en la figura A.60 (si no hubiese ninguno vigente se indicaría con una alerta). Además puede acceder al registro de sus contratos mediante el botón *Mostrar*

todos los contratos, que le mostraría una pantalla como la representada en la figura A.61.

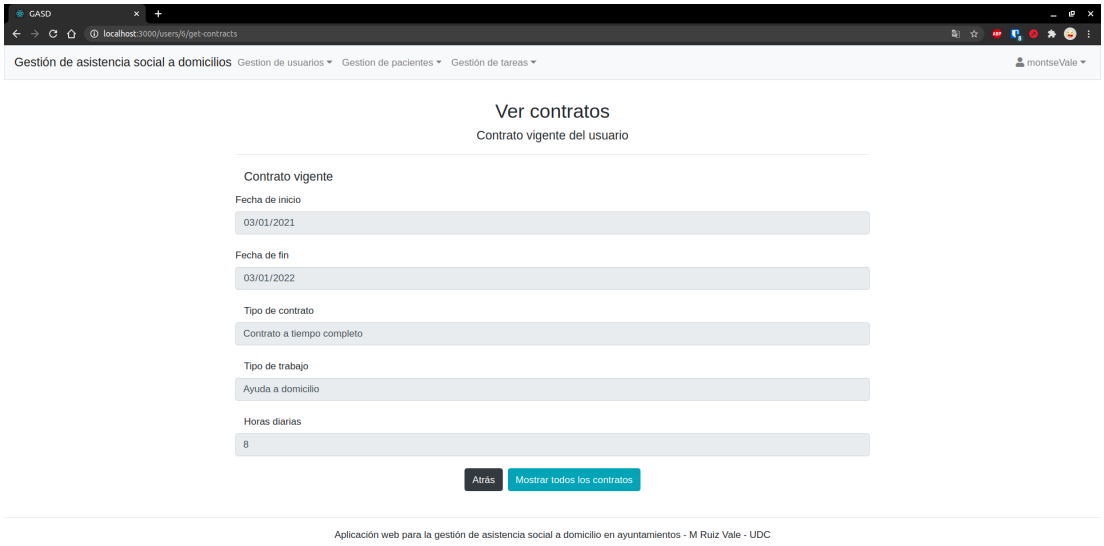


Figura A.60: Contrato actual de un trabajador

Figura A.61: Registro de contratos de un trabajador

Desde el menú principal, presionando el botón *Ver pacientes*, el trabajador puede consultar los datos de los pacientes de su ayuntamiento. A.62. Accediendo a través del botón de la columna *Ver detalles* puede acceder a la información completa del paciente y ver todas sus

direcciones de domicilio.

Figura A.62: Pacientes del ayuntamiento del trabajador

El trabajador puede gestionar las notas de un paciente, accediendo desde el menú con el botón *Gestión de notas* (fig. A.63), que le permite añadir una nota (presionando en el botón de la columna *Añadir nota*) y ver todas las notas, públicas y privadas, de un paciente (presionando el botón de la columna *Ver todas las notas de un paciente*) (fig.A.64).

Figura A.63: Gestión de notas de un paciente por parte de un trabajador

Figura A.64: Visualización de notas de un paciente por parte de un trabajador

El trabajador puede consultar el horario que le ha sido asignado desde el botón *Visualizar horarios por trabajador*. Además puede marcar las tareas como completadas accediendo los datos de cada una haciendo clic en una tarea y presionando el botón de *Marcar como completada*.

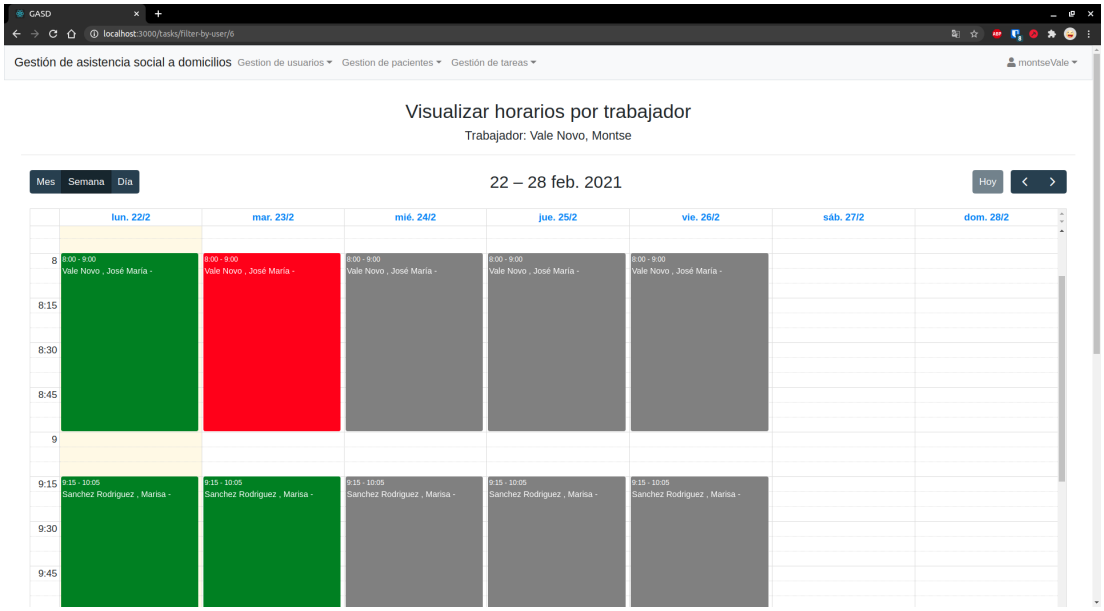


Figura A.65: Ver horario de trabajador

Por último, el trabajador puede visualizar los períodos que no ha trabajado, además del

total del año. Como se puede observar en la figura A.66

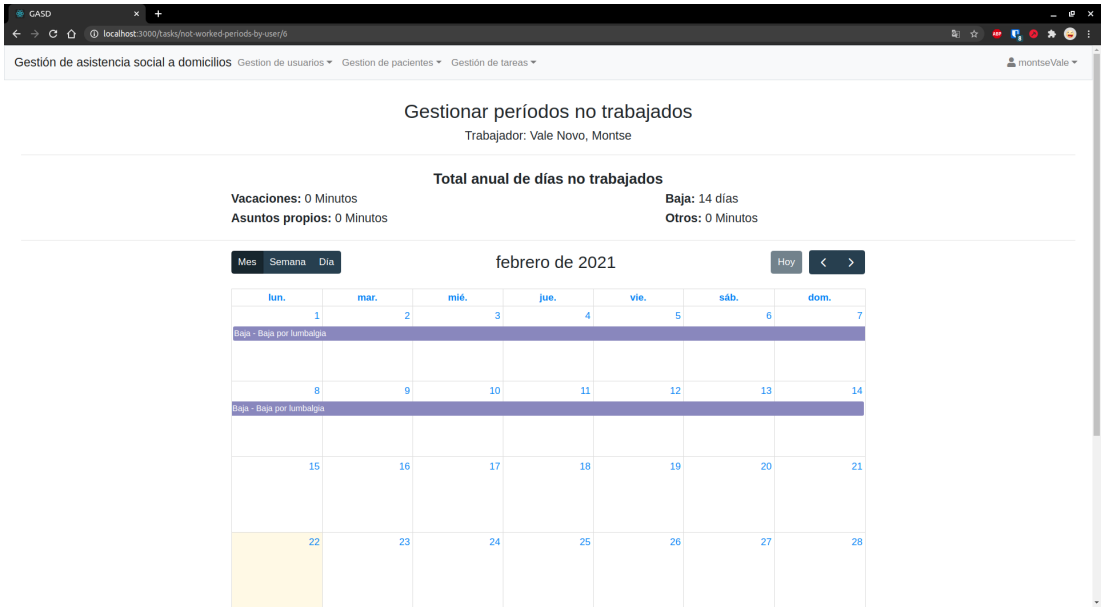


Figura A.66: Ver períodos no trabajados de trabajador

A.3.5 Usuario de paciente

Si se ingresa en la aplicación con un usuario de paciente, se muestra un menú principal como el de la figura A.67

Figura A.67: Menú principal del usuario de paciente

En el botón *Ver pacientes*, el usuario tiene acceso a los pacientes que están a su cargo, acceder a sus datos y direcciones (fig. A.62)

Figura A.68: Pacientes a asignados a un usuario de paciente

El usuario de paciente puede crear y visualizar notas en los pacientes que tiene asignados a su perfil. Como se puede ver en la figura A.69, podría gestionar las notas únicamente de los pacientes que están asignados a su perfil. Si accede al formulario mediante el botón de la columna *Añadir nota* puede crear notas con visibilidad pública (fig. A.70). Si el usuario de paciente quiere ver las notas de uno de sus pacientes asignados, solamente podrá hacerlo de sus notas con visibilidad pública (fig. A.71).

Figura A.69: Gestión de notas de los pacientes asignados a su perfil

Figura A.70: Creación de nota a un paciente asignado a su perfil de usuario

Figura A.71: Visualización de notas públicas de pacientes asignados a su perfil de usuario

Por último, el usuario de paciente tiene acceso a los horarios asignados a sus pacientes, mediante el botón *Gestionar horarios por paciente* del menú principal se accede a una lista que le permitirá consultar los horarios de cada uno de los pacientes (fig. A.72). Haciendo clic en el botón de la columna *Ver horarios*, se accede a la visualización del horario del paciente (fig. A.73).

Una vez en la visualización del horario del paciente se puede acceder a la información completa de la tarea haciendo clic en ella.

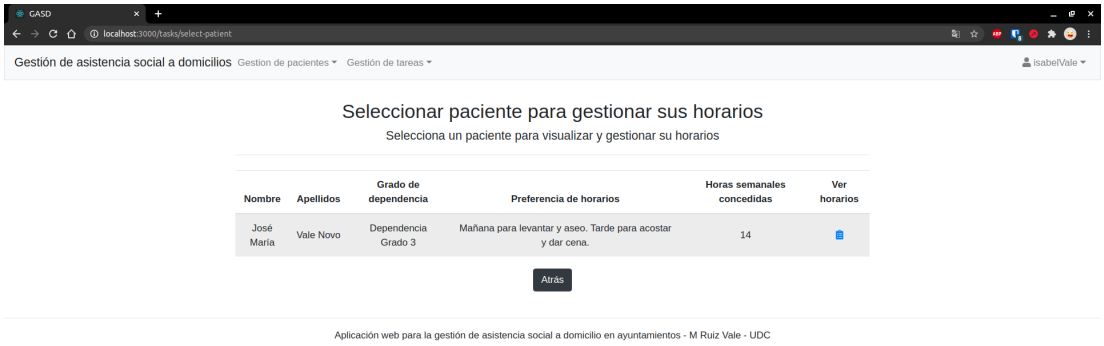


Figura A.72: Seleccionar paciente asignado al perfil para ver horario

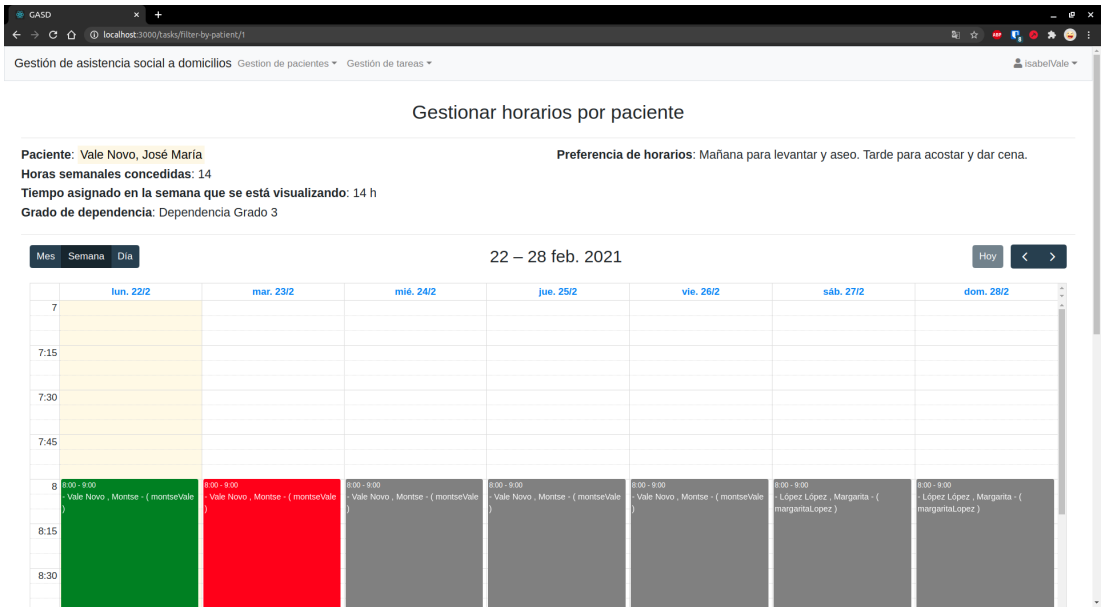


Figura A.73: Ver horarios de paciente asignado al perfil

Lista de acrónimos

SPA *Single Page Application.*

IETF *Internet Engineering Task Force.*

JVM *Java Virtual Machine.*

HTML *HyperText Markup Language.*

CSS *Cascading Style Sheets.*

XML *eXtensible Markup Languag.*

JSON *JavaScript Object Notation.*

JDCB *Java Database Connectivity.*

API *Application Programming Interface,.*

REST *REpresentational State Transfer.*

IDE *Integrated Development Environment.*

HTTP *HyperText Transfer Protocol.*

SSL/TLS *Secure Sockets Layer/Transport Layer Security.*

POM *Project Object Model .*

LAMP *Linux, Apache, MySQL and PHP.*

BD *Base de Datos.*

JPA *Java Persistence API.*

CRUD *Create Read Update Delete.*

CU *Caso de Uso.*

UML *Unified Modeling Language.*

DAO *Data Access Object.*

DTO *Data Transfer Object .*

POJO *Plain Old Java Objectl.*

JWT *JSON Web Token.*

URL *Uniform Resource Locator.*

RFID *Radio Frequency Identification.*

Glosario

Frameworks Un *framework* es un marco o esquema de trabajo generalmente utilizado por programadores para realizar el desarrollo de software.

Open Source El software *open source* es un código diseñado de manera que sea accesible al público: todos pueden ver, modificar y distribuir el código de la forma que consideren conveniente.

Backend Aplicación del lado servidor de un sistema.

Frontend Subsistema del lado cliente de una aplicación web.

Token Objeto o dato no sensible que sustituye un dato sensible como una contraseña.

Endpoints Cada una de las interfaces expuestas de una API.

Bibliografía

[1] Boletin Oficial del Estado, "Ley 39/2006, de 14 de diciembre, de promoción de la autonomía personal y atención a las personas en situación de dependencia," *«BOE» número 299*, 2006.

[2] ——, "Real decreto-ley 20/2012, de 13 de julio, de medidas para garantizar la estabilidad presupuestaria y de fomento de la competitividad," *«BOE» número 168*, pp. 50 428 – 50 518, 2012.

[3] Oracle, "Java," Accedido en Octubre 2020. [En línea]. Disponible en: https://www.java.com

[4] MozillaWebDocs, "Html básico," Accedido en Octubre 2020. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Learn/HTML

[5] ——, "Css," Accedido en Octubre 2020. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/CSS

[6] ——, "Javascript," Accedido en Octubre 2020. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript

[7] "Json docs," Accedido en Noviembre 2020. [En línea]. Disponible en: https://www.json.org/json-es.html

[8] Apache, "Spring framework overview," Accedido en Octubre 2020. [En línea]. Disponible en: https://spring.io/projects/spring-framework

[9] RedHat, "Hibernate framework overview," Accedido en Noviembre 2020. [En línea]. Disponible en: https://hibernate.org

[10] Facebook, "Documentación de react," Accedido en Noviembre 2020. [En línea]. Disponible en: https://es.reactjs.org/

[11] D. Abramov, "Documentación de redux," Accedido en Noviembre 2020. [En línea]. Disponible en: https://es.redux.js.org/

[12] M. Otto and J. Thornton, "Documentación de bootstrap," Accedido en Noviembre 2020. [En línea]. Disponible en: https://getbootstrap.com/

[13] D. Ferris, A. Shaw, and Rustam, "Documentación de fullcalendar," Accedido en Noviembre 2020. [En línea]. Disponible en: https://fullcalendar.io/

[14] E. Gamma, K. Beck, and D. Saffe, "Junit docs," Accedido en Noviembre 2020. [En línea]. Disponible en: https://junit.org/

[15] IETF, "Hypertext transfer protocol," Accedido en Noviembre 2020. [En línea]. Disponible en: https://www.ietf.org/rfc/rfc2616.txt

[16] L. Torvalds, "Git documentation," Accedido en Noviembre 2020. [En línea]. Disponible en: https://git-scm.com/

[17] The Apache Software Foundation, "About maven / what is maven?" Accedido en Noviembre 2020. [En línea]. Disponible en: https://maven.apache.org/what-is-maven.html

[18] The Eclipse Foundation, "Eclipse ide," Accedido en Febrero 2021. [En línea]. Disponible en: https://help.eclipse.org/2020-12/index.jsp

[19] Microsoft, "Visual studio code," Accedido en Noviembre 2020. [En línea]. Disponible en: https://code.visualstudio.com/docs

[20] "Documentación postman," Accedido en Febrero 2021. [En línea]. Disponible en: https://learning.postman.com/

[21] Solid IT, "Db-engines ranking," Accedido en Noviembre 2020. [En línea]. Disponible en: https://db-engines.com/en/ranking/

[22] Oracle, "Mysql," Accedido en Noviembre 2020. [En línea]. Disponible en: https://dev.mysql.com/doc/

[23] ——, "Mysql workbench," Accedido en Noviembre 2020. [En línea]. Disponible en: https://www.mysql.com/products/workbench/

[24] "Laborofficefree," Accedido en Febrero 2021. [En línea]. Disponible en: https://www.laborofficefree.com/

[25] Grupo Trevenque, "gesad," Accedido en Febrero 2021. [En línea]. Disponible en: https://gesad.trevenque.es/funcionalidades/

[26] Servisoft.cat, "ecosad," Accedido en Febrero 2021. [En línea]. Disponible en: http://www.8d10.com/ecosad

[27] J. Rumbaugh, I. Jabobson, and G. Booch, *El lenguaje unificado de modelado. Manual de referencia.* Pearson Educación, 2000.

[28] Grupo Adecco, "Guía spring del mercado laboral 2020," Accedido en Febrero 2021. [En línea]. Disponible en: https://www.springspain.com/-/media/adeccogroup/brands/spring-professional/spain/media/shared/Guia-Spring%20Professional-del%20Mercado-Laboral%202020-72ppp.pdf

[29] OReilly, "Software architecture patterns," Accedido en Febrero 2021. [En línea]. Disponible en: https://www.oreilly.com/content/software-architecture-patterns/

[30] L. Crusoveanu, "Intro to inversion of control and dependency injection with spring," Accedido en Febrero 2021. [En línea]. Disponible en: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

[31] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Patrones de diseño: elementos de software orientado a objetos reutilizables.* Pearson Educación, 2003.

[32] Baeldung, "Persistence with spring," Accedido en Febrero 2021, pp. 24 –28. [En línea]. Disponible en: https://s3.amazonaws.com/baeldung.com/Persistence+with+Spring.pdf?__s=pe3hxfs176vyqk4ib2sf