**Documentation Tecnica API RESTful Con Spring Boot - Spring Batch Angular.**

El proyecto tiene como objetivo proporcionar una solución integral para la gestión de alojamientos en residencias. Este sistema está diseñado para facilitar la administración de habitaciones, reservas, personas, establecimientos y usuarios, ofreciendo una plataforma eficiente y fácil de usar para los administradores de residencias.

**Funcionalidades Principales**

* **Gestión de Habitaciones**
* Registro de Habitaciones: Permite a los administradores registrar nuevas habitaciones, especificando detalles como el número de habitación, tipo, estado y el establecimiento al que pertenecen.
* Actualización de Estado: Automatiza la actualización del estado de las habitaciones (disponible, ocupada) en función de las reservas activas.
* **Gestión de Reservas**
* Creación de Reservas: Facilita la creación de nuevas reservas, incluyendo información sobre la persona que realiza la reserva, la habitación asignada, las fechas de entrada y salida, y otros detalles relevantes.
* Actualización y Cancelación: Permite la actualización y cancelación de reservas existentes, manteniendo la información siempre actualizada.
* Eliminación Automática: Implementa una funcionalidad para eliminar automáticamente las reservas que tengan más de 5 años desde su fecha de salida.
* **Gestión de Personas:**
* Registro de Personas: Permite registrar información detallada sobre las personas que utilizan los servicios de la residencia, incluyendo nombre, apellidos, DNI, fecha de nacimiento, teléfono y correo electrónico.
* Búsqueda y Filtrado: Ofrece funcionalidades de búsqueda y filtrado para encontrar personas específicas basadas en diferentes criterios.
* Eliminación Automática: Implementa una funcionalidad para eliminar automáticamente las personas que tengan más de 5 años desde su última fecha de salida.
* **Gestión de Establecimientos:**
* Registro de Establecimientos: Permite registrar nuevos establecimientos, especificando detalles como nombre, dirección, teléfono y capacidad.
* Visualización de Habitaciones y Reservas: Facilita la visualización de todas las habitaciones y reservas asociadas a un establecimiento específico.
* Estadísticas de Edad: Calcula la media de edad de las personas alojadas en cada establecimiento y envía un correo electrónico con estas estadísticas a los administradores
* **Gestión de Usuarios:**
* Registro de Usuarios: Permite registrar nuevos usuarios del sistema, especificando detalles como nombre de usuario, contraseña, rol y correo electrónico.
* Encriptación de Contraseñas: Implementa la encriptación de contraseñas para asegurar que las contraseñas de los usuarios se almacenen de manera segura.
* Control de Acceso: Implementa control de acceso basado en roles para asegurar que solo los usuarios autorizados puedan realizar ciertas acciones.

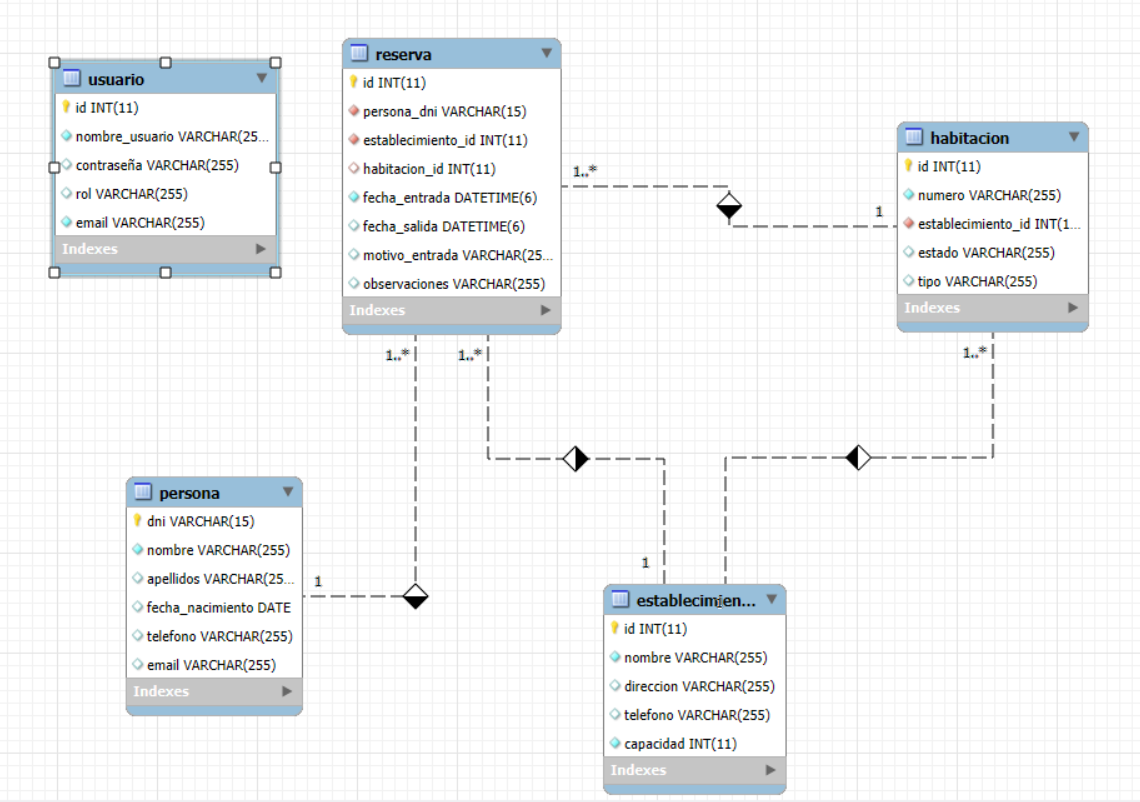
**Beneficios del Sistema**

* Eficiencia Operativa: Automatiza y simplifica la gestión de alojamientos, reduciendo el tiempo y esfuerzo necesarios para realizar tareas administrativas.
* Acceso Centralizado: Proporciona un acceso centralizado a toda la información relevante, facilitando la toma de decisiones informadas.
* Mejora en la Experiencia del Usuario: Ofrece una interfaz intuitiva y fácil de usar, mejorando la experiencia tanto para los administradores como para los usuarios finales.
* Seguridad de Datos: Implementa medidas de seguridad para proteger la información sensible y garantizar la integridad de los datos, incluyendo la encriptación de contraseñas.
* Automatización de Tareas: Automatiza la eliminación de datos antiguos y el envío de estadísticas, mejorando la eficiencia y la precisión de la gestión.

**Modelo de datos.**

El modelo de datos del proyecto está diseñado para representar las entidades principales y sus relaciones. Las entidades clave incluyen Habitación, Reserva, Persona, Establecimiento, y Usuario.

A continuación se muestra el diagrama ERD y la cardinalidad de las relaciones entre las distintas entidades.



**Arquitectura del Sistema**

El sistema está dividido en tres capas principales: Backend, Frontend y Base de Datos. Además, incluye funcionalidades adicionales como el envío de correos electrónicos y la encriptación de contraseñas.

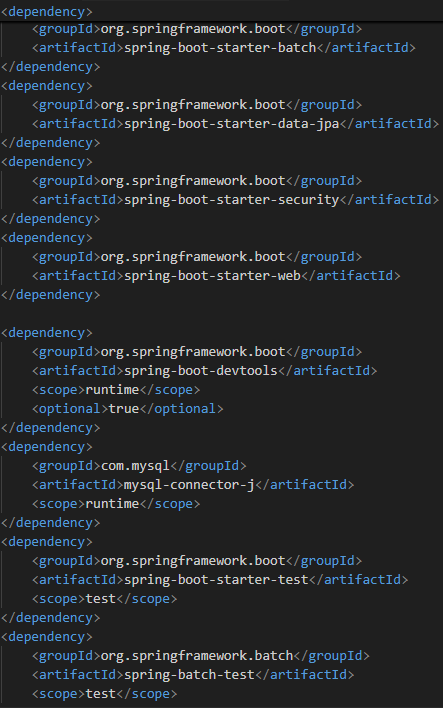
**Tecnologías utilizadas**

**JAVA - SPRING BOOT - SPRING BATCH - ANGULAR**

**Uso de Maven**

Es una herramienta de gestión y construcción de proyectos basada en Java. Proporciona un modelo de proyecto coherente, un sistema de gestión de dependencias y un ciclo de vida de construcción. Maven facilita la gestión de proyectos al automatizar tareas como la compilación, el empaquetado, las pruebas y la implementación. Utiliza un archivo de configuración llamado pom.xml para definir las dependencias, plugins y otras configuraciones del proyecto.

*Ejemplo del pom.xml*

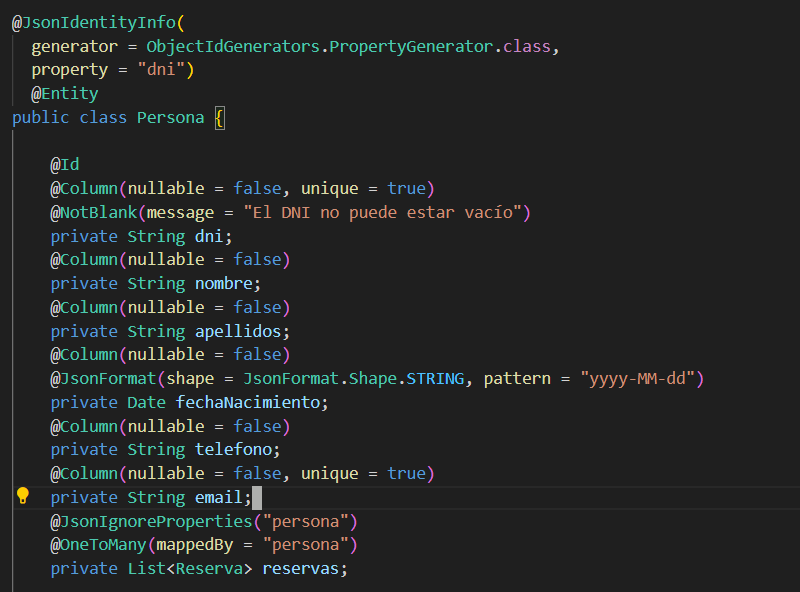


**Backend**

El backend se encarga de la lógica de negocio y la comunicación con la base de datos. Utiliza Spring Boot . Se ha utilizado el lenguaje de programación **Java** juntos a los framework **Spring Boot** para facilitar la creación de servicios RESTful y **Spring Batch** para el borrado automático y envió de estadísticas por correo electrónico.

**Modelos**

Los modelos representan las entidades del sistema y se mapean a las tablas de la base de datos.



**Anotaciones utilizadas**

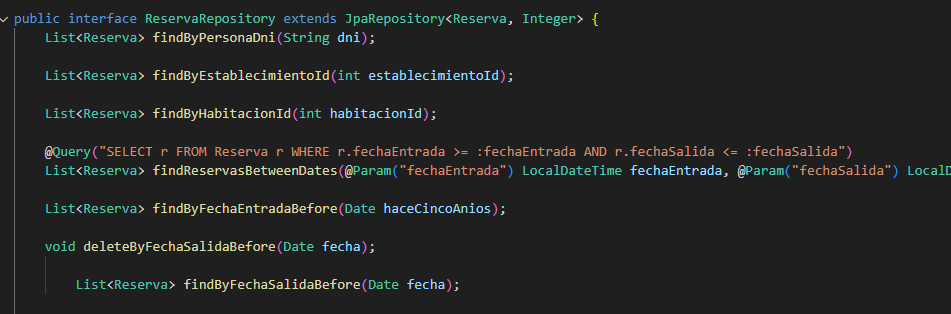
* **@Entity:** Indica que la clase es una entidad JPA y que debe ser mapeada a una tabla en la base de datos. Se utiliza para definir una clase como una entidad persistente.
* **@Id:** Indica el campo que será utilizado como la clave primaria de la entidad. Se utiliza para identificar el campo dni como la clave primaria de la entidad Persona.
* **@Table:** Especifica el nombre de la tabla en la base de datos que se mapeará a esta entidad.
* **@Column:** Se utiliza para especificar los detalles de la columna en la base de datos.
* **@JoinColumn:** Especifica el nombre de la columna que contiene la clave foránea y si puede ser nula.
* **@GeneratedValue:** Especifica que el valor de la clave primaria será generado automáticamente. Se utiliza con la estrategia GenerationType.IDENTITY para que el valor del campo id sea generado automáticamente por la base de datos.
* **@Column (unique = true):** asegura que no puede ser nulo y debe ser único en la base de datos.
* **@Column(nullable = false)** en los campos asegura que estos campos no pueden ser nulos.
* **@OneToMany** Define una relación uno a muchos. Se utiliza para indicar que una clase modelo puede tener múltiples objetos de otra entidad. El atributo mappedBy especifica el campo en la entidad Reserva que posee la clave foránea.
* **@ManyToOne:** Define una relación muchos a uno entre la entidad Habitacion y la entidad Establecimiento.
* **@NotBlank:** Asegura que el campo no sea nulo ni vacío. Se utiliza en el campo dni para asegurar que el DNI no puede estar vacío y debe contener al menos un carácter.
* **@JsonIdentityInfo:** Maneja las referencias circulares en la serialización JSON, evitando problemas de recursión infinita. Se utiliza para definir cómo se deben manejar las referencias de identidad en la serialización JSON. En este caso, se utiliza **PropertyGenerator** para generar la identidad basada en la propiedad dni.
* **@JsonFormat:** Define el formato de la fecha cuando se serializa/deserializa a JSON. Se utiliza en el campo fechaNacimiento para especificar que la fecha debe ser serializada/deserializada en el formato yyyy-MM-dd.
* **@JsonIgnoreProperties:** Ignora las propiedades especificadas durante la serialización/deserialización JSON.
* **@JsonIdentityReference:** Configura cómo se deben manejar las referencias de identidad en la serialización JSON.
* **@Enumerated:** Especifica que el campo debe ser tratado como un enumerado.

**Repositorios JPA**

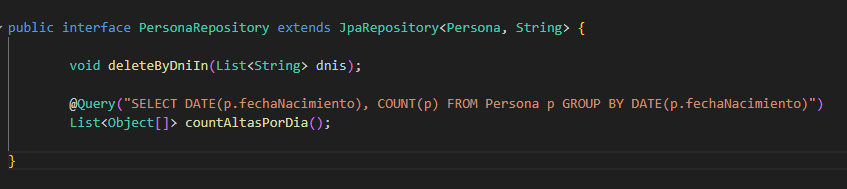
Los repositorios en JPA son interfaces que proporcionan métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en las entidades. **Spring Data JPA** proporciona una serie de interfaces que se pueden extender para crear repositorios específicos para cada entidad. La interfaz principal es **JpaRepository**, que proporciona métodos genéricos para las operaciones CRUD. **Al extender JpaRepository, se heredan estos métodos** y se pueden añadir métodos personalizados según sea necesario.

Ejemplos.

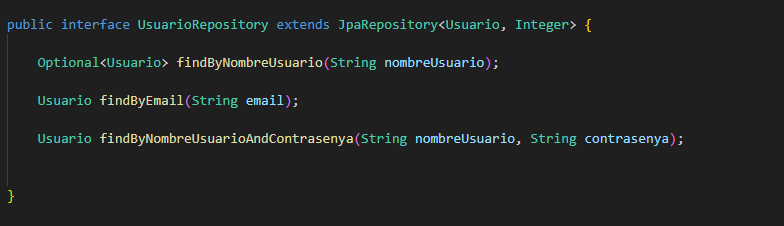
El método **findReservasBetweenDates** en el repositorio de Reserva busca todas las reservas cuya fecha de entrada es mayor o igual a fechaEntrada y cuya fecha de salida es menor o igual a fechaSalida. Se utiliza para encontrar reservas dentro de un rango de fechas especificado.



El método **countAltasPorDia** en el repositorio Persona cuenta el número de personas registradas (altas) por día, agrupando los resultados por la fecha de nacimiento. Datos que se utilizan más adelante para mandar las estadísticas por correo.

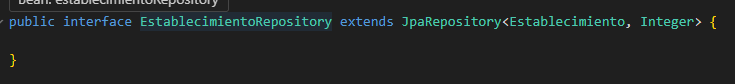


Los siguientes métodos sirven para realizar una búsqueda de usuarios a través de su nombre de usuario, email o nombre de usuario y contraseña.



Los repositorios de habitaciones y establecimientos extienden a JpaRepository y usan los métodos por defecto.





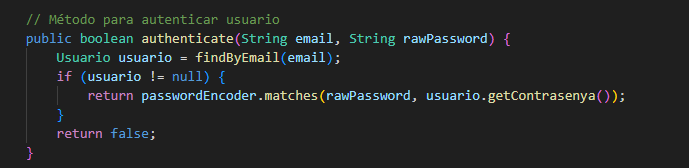
**Servicios**

Los servicios en una aplicación Spring se encargan de la lógica de negocio. Utilizan los repositorios para interactuar con la base de datos y proporcionan métodos que encapsulan las operaciones necesarias para cumplir con los requisitos de la aplicación.

Los servicios en Spring se definen como clases anotadas con **@Service.** Esta anotación indica que la clase es un componente de servicio y debe ser gestionada por el contenedor de Spring.

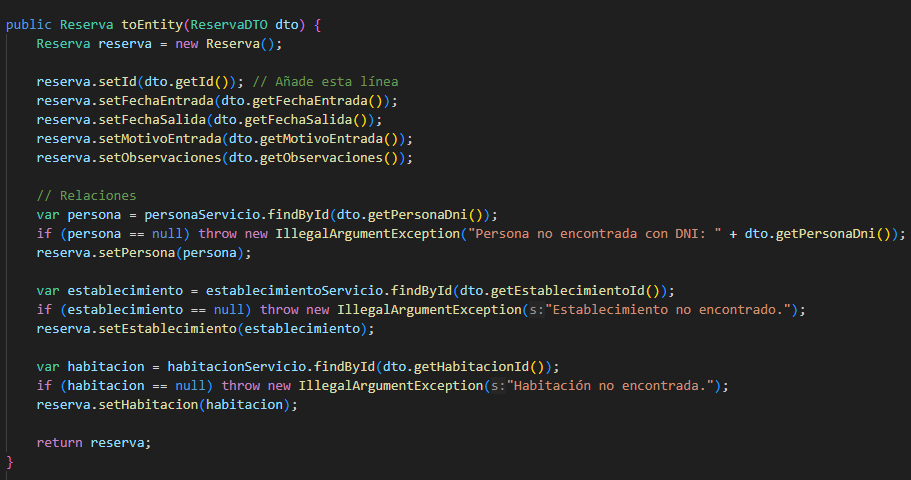
Se ha utilizado **BCryptPasswordEncorder** en el servicio de usuario para encriptar las contraseñas y proporcionar un método de autentificación.



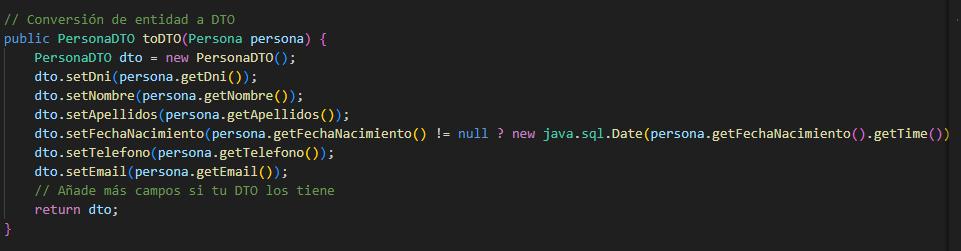


Cada servicio incluye métodos para convertir entre entidades y DTOs, facilitando la transferencia de datos entre el cliente y el servidor.

*Ejemplo método para para convertir de la entidad Reserva a DTO*.

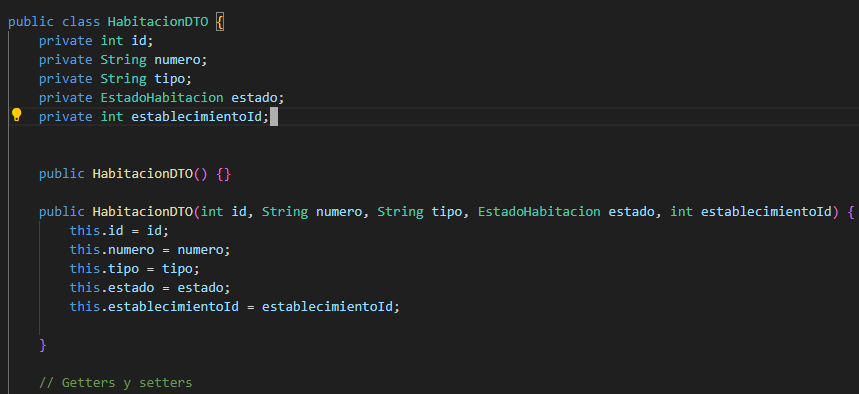


*Ejemplo método para convertir de DTO a entidad Persona.*



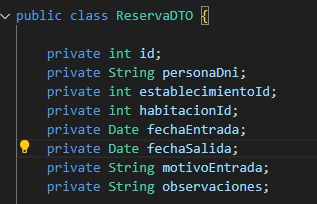
**Clases DTO (Data Transfer Objects)**

Los DTOs se utilizan para transferir datos entre el cliente y el servidor. Los DTOs pueden incluir validaciones y transformaciones de datos, y suelen omitir campos sensibles como contraseñas. Utilizar DTOs en lugar de entidades directamente ayuda a mantener la separación de preocupaciones y mejora la seguridad y la eficiencia de la aplicación. En ellos se indican las propiedades que se desean recibir/mostrar.



En este caso **establecimientoId** indica el id del establecimiento, **sin incluir el objeto entero anidado** ya que solo se necesita su Id, no el resto de la información de la entidad.

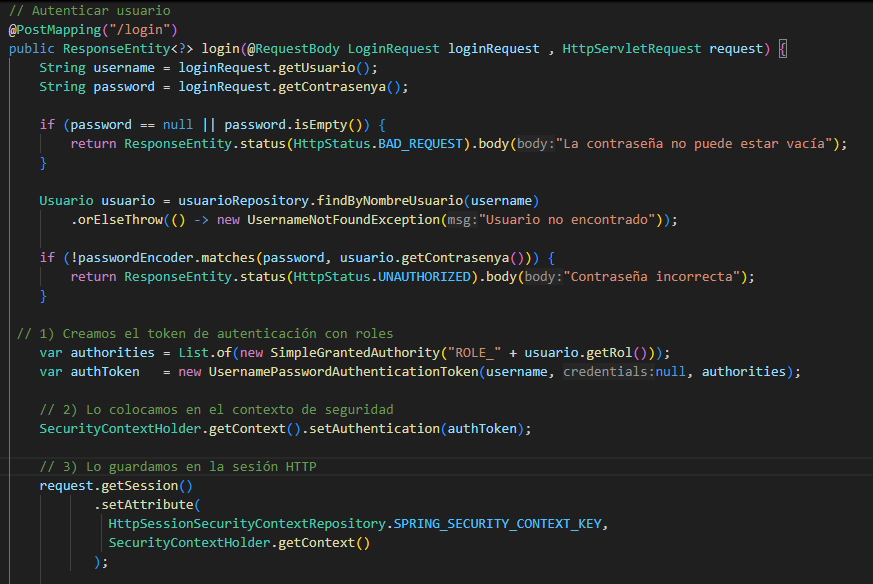
Para el resto de DTO se ha seguido el mismo patrón.



**Controladores**

Los controladores manejan las solicitudes HTTP y utilizan los servicios para realizar las operaciones necesarias. En lugar de recibir y enviar entidades directamente, **los controladores utilizan DTOs** para transferir datos entre el cliente y el servidor. También validan que los datos que se están enviando y recibiendo son los que espera el backend.

En el controller Usuario se valida y autentifica que el usuario introducido es válido y se guarde la sesión HTTP correctamente.

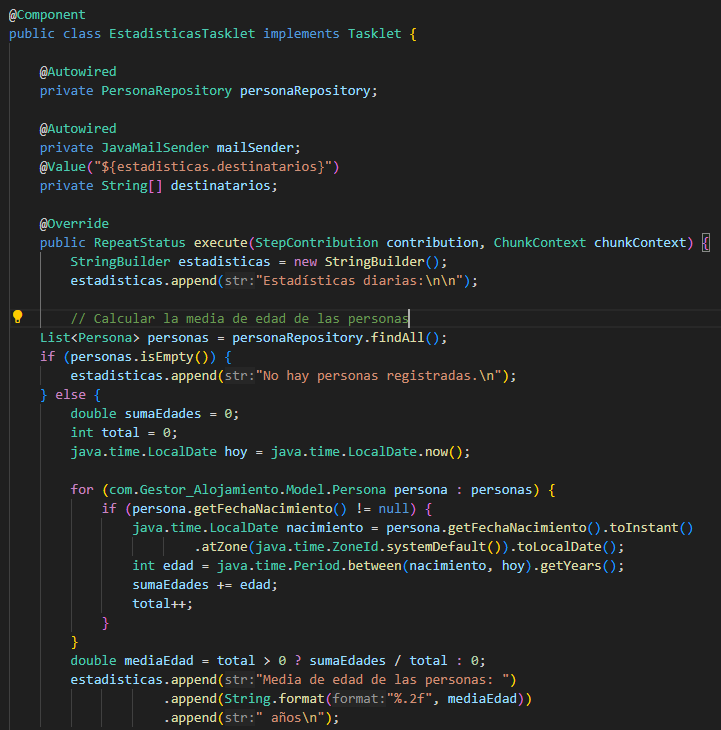


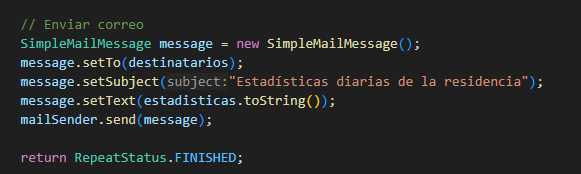
**Clases Spring batch**

A continuación, se muestra la **Tasklet** utilizada para generar estadísticas y los **ItemReader y ItemWriter** utilizados para borrar reservas y personas antiguas.

Una Tasklet es una interfaz de Spring Batch que se utiliza para definir una unidad de trabajo que se ejecuta en un paso de un trabajo. En este caso, la Tasklet se utiliza para generar estadísticas y enviarlas por correo electrónico.

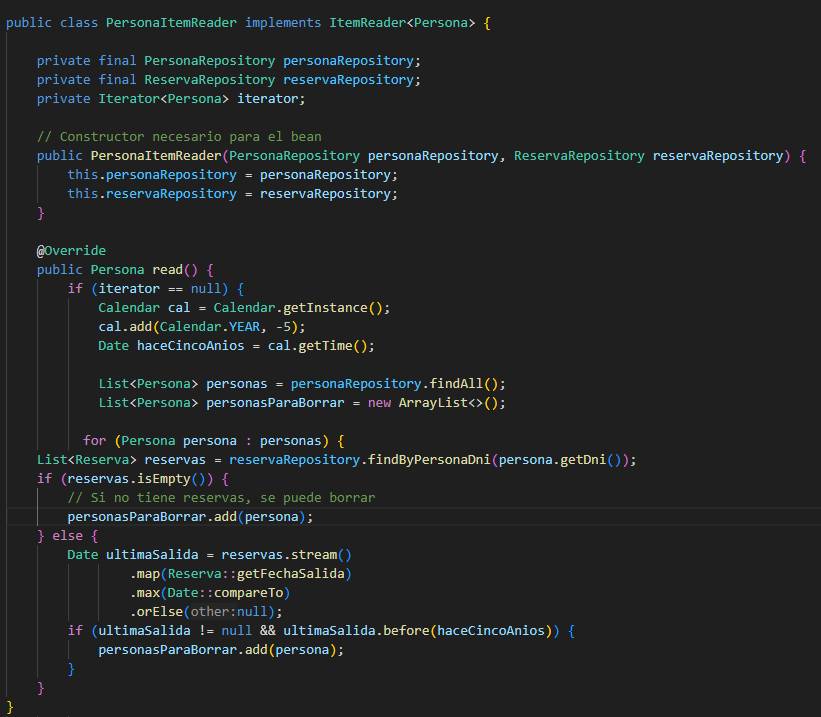
*Ejemplo:*

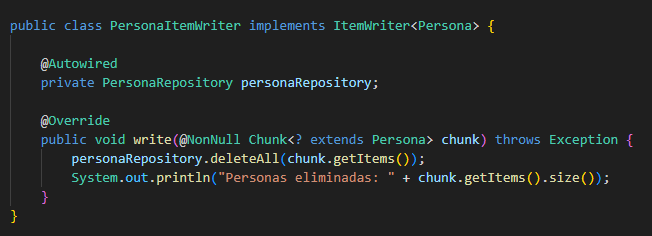




Los ItemReader y ItemWriter se utilizan para leer y escribir datos durante el procesamiento por lotes. En este caso, se utilizan para leer y borrar reservas y personas con fecha de salida de la última reserva de hace 5 años.

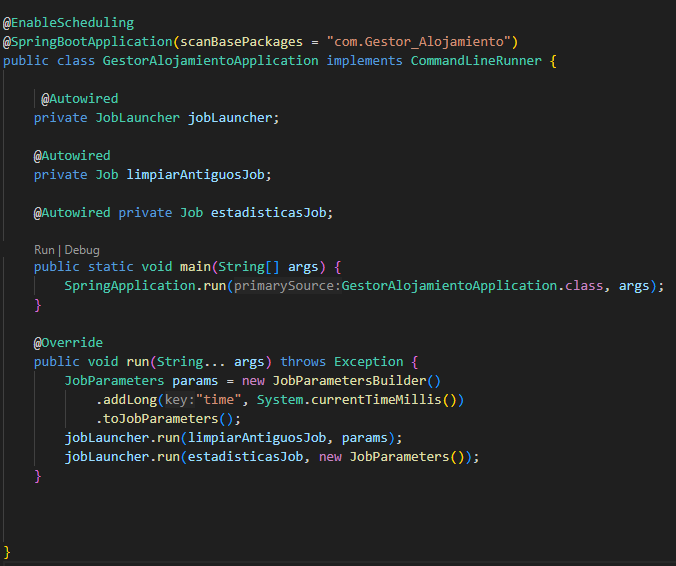
*Ejemplo:*





La clase principal de la aplicación se encarga de iniciar la aplicación Spring Boot. En este caso, también se encargará de ejecutar el trabajo de Spring Batch al inicio de la aplicación.

Utiliza el **JobLauncher** para iniciar los métodos con los parámetros especificados. Esto permite que los trabajos de procesamiento por lotes se ejecuten automáticamente cuando la aplicación se inicia.



**Configuraciones**

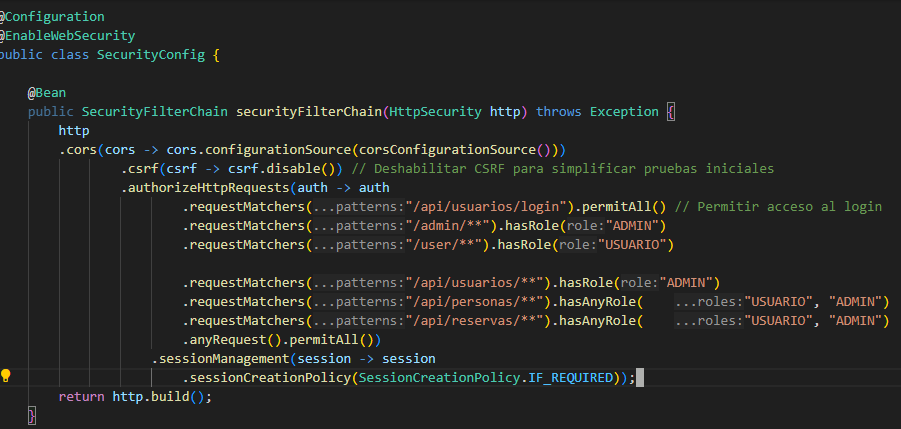
**Configuración de MVC**

Se utiliza para configurar aspectos del framework Spring MVC, como la resolución de vistas, la configuración de recursos estáticos y la configuración de **CORS (Cross-Origin Resource Sharing)**, la cual permite la conexión entre backend y frontend.

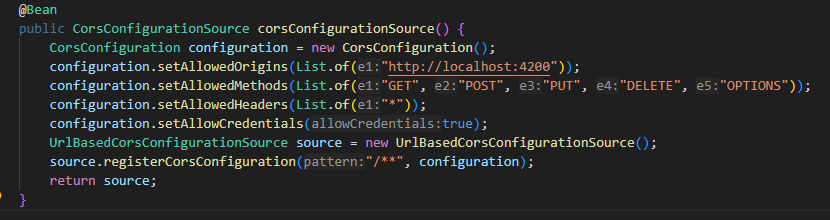


Configuración de Seguridad (SecurityConfig)

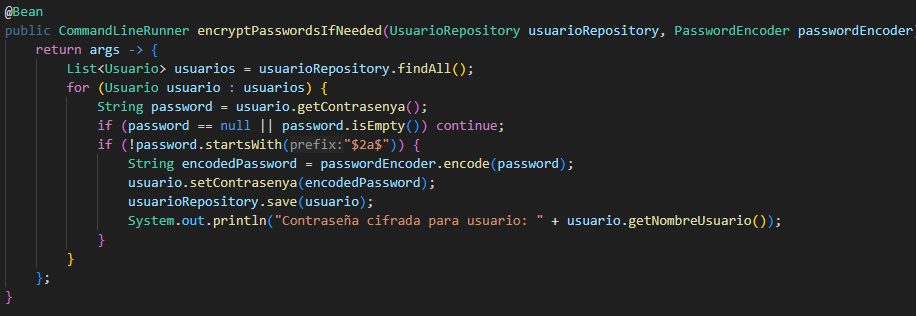
La configuración de seguridad se utiliza para definir las políticas de seguridad de la aplicación, como la autenticación y la autorización. Utiliza **Spring Security** para proteger las rutas y gestionar el acceso a los recursos de la aplicación



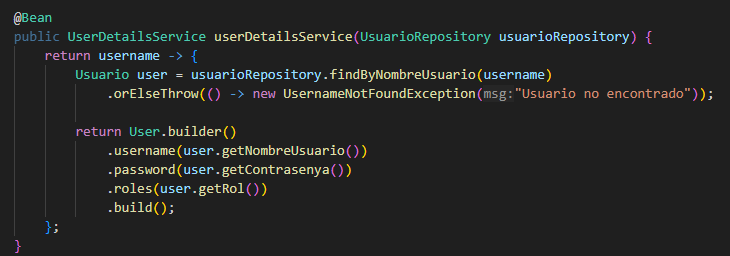
Este método define la configuración de CORS para permitir solicitudes desde orígenes específicos.



El siguiente método se ejecuta al inicio de la aplicación y verifica cada usuario de la base de datos, encriptando su contraseña si no lo está.



Este método define el **bean UserDetailsService** que se utiliza para cargar los detalles del usuario durante la autenticación.



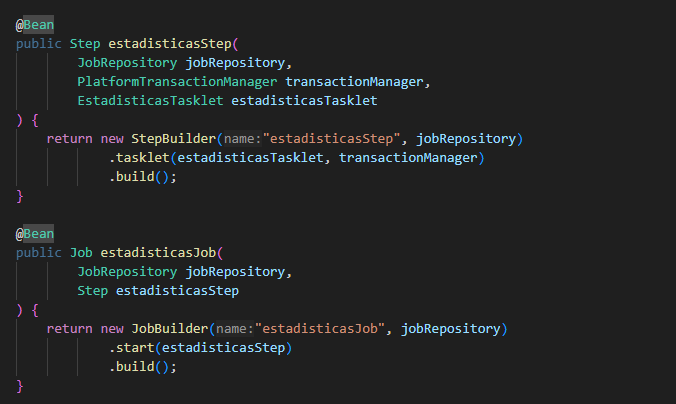
**Configuración de Batch (BatchConfig)**

se utiliza para definir y configurar trabajos de procesamiento por lotes en una aplicación Spring. Utiliza Spring Batch para gestionar la ejecución de tareas repetitivas y de gran volumen, como la limpieza de datos antiguos y la generación de estadísticas.

Jobs para borrar personas y reservas antiguas



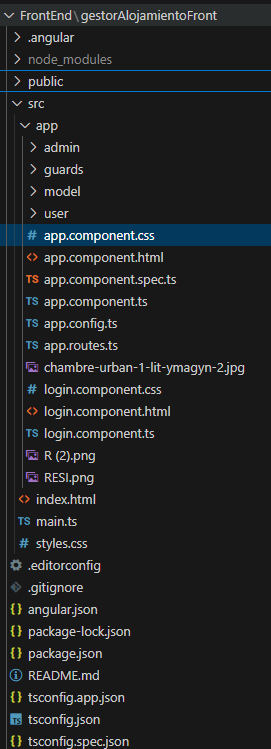
Step y job para calcular estadísticas



**FRONTEND**

El front-end es la parte que interactúa directamente con los usuarios. Se encarga de la presentación de datos, la captura de entradas del usuario y la comunicación con el back-end para realizar operaciones como la recuperación, creación, actualización y eliminación de datos. En este caso, se utiliza Angular.

**Estructura del proyecto**

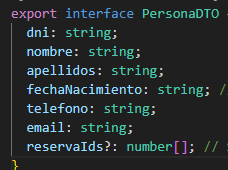


El enrutamiento en Angular permite definir rutas para navegar entre diferentes vistas o componentes de la aplicación. Cada ruta está asociada con un componente específico que se carga cuando la ruta es activada.



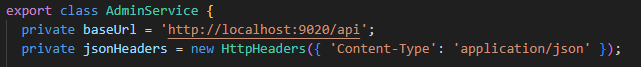
**Modelos**

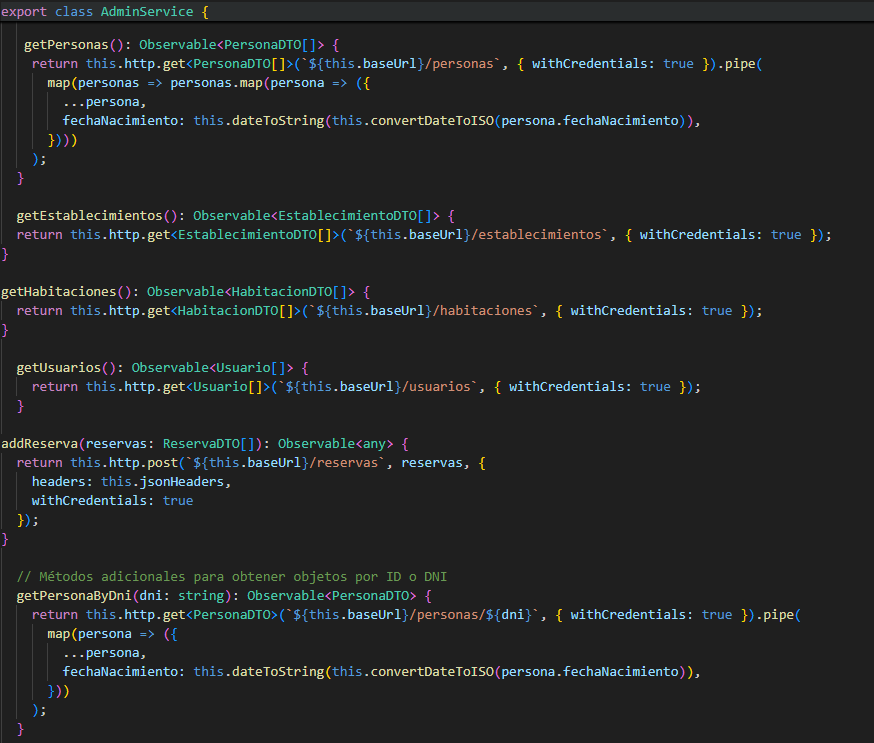
Los modelos representan la estructura de los datos que se manejan en la aplicación. Definen las propiedades de los objetos que se intercambian entre el front-end y el backend. Deben tener las mismas propiedades en ambos lados.



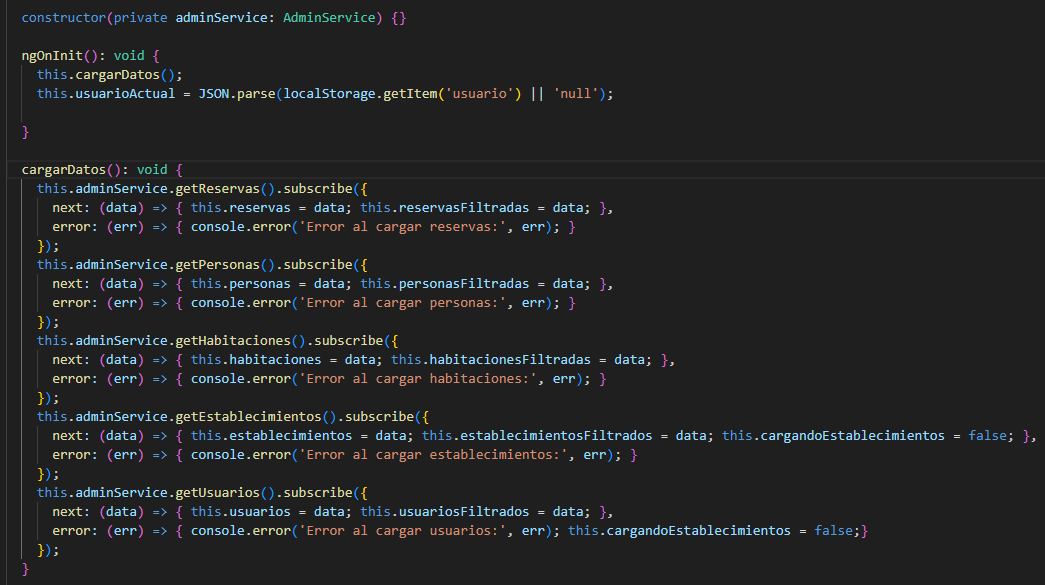
**Servicios y componentes**

Los servicios en Angular se encargan de realizar las operaciones **CRUD** y otras tareas relacionadas con la comunicación con el backend. El **AdminService** en este caso actúa como una capa intermedia entre los componentes y el backend, facilitando la comunicación y el intercambio de datos Y proporciona métodos para realizar operaciones CRUD en todas las entidades.

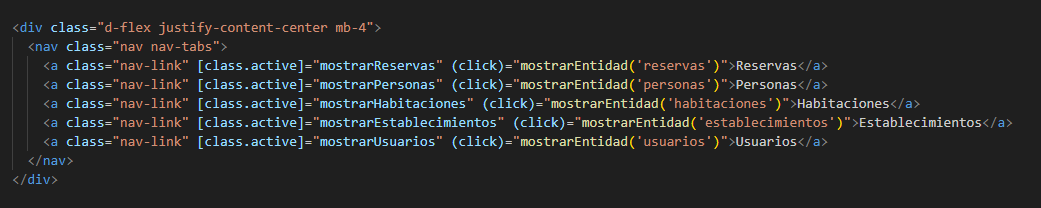




El **AdminComponent** utiliza este servicio para gestionar las entidades proporcionando una interfaz de administración. El **UserComponent** también utiliza el **AdminService** para cargar y mostrar la lista de usuarios.



Su plantilla HTML incluye listas de todas las entidades, así como botones para eliminar estos elementos.



También se incluyen métodos para filtrar los datos y mostrarlos en el html ya filtrados, como por ejemplo ver la disponibilidad de habitaciones segun la fecha elegida y el establecimiento.

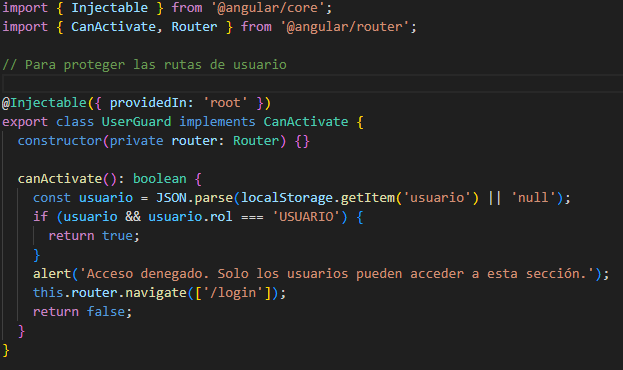


Para proteger las rutas /admin y /user desde el lado front y que solo el rol usuario o admin pueda acceder respectivamente se ha especificado en los **guards.**

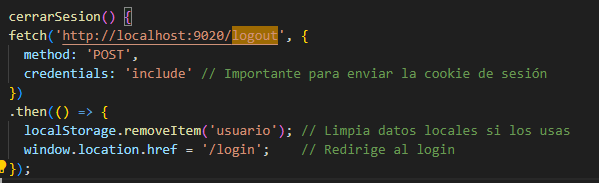
*Ejemplo para admin*



*Ejemplo para user*



Y se ha implementado un método en los dos componentes en admin y user utilizando el sistema de **logout** que proporciona Spring para cerrar sesion y redirigir a la pantalla de inicio de sesión una vez terminada la sesión actual.



**Imágenes del prototipo**

A continuación, se presentan las vistas principales del prototipo de la aplicación, incluyendo la vista de inicio de sesión, la vista de administración y la vista de usuario.

**Vista de Inicio de Sesión**

La vista de inicio de sesión permite a los usuarios autenticarse en la aplicación. Los usuarios deben ingresar sus credenciales (nombre de usuario y contraseña) para acceder a las funcionalidades de la aplicación.



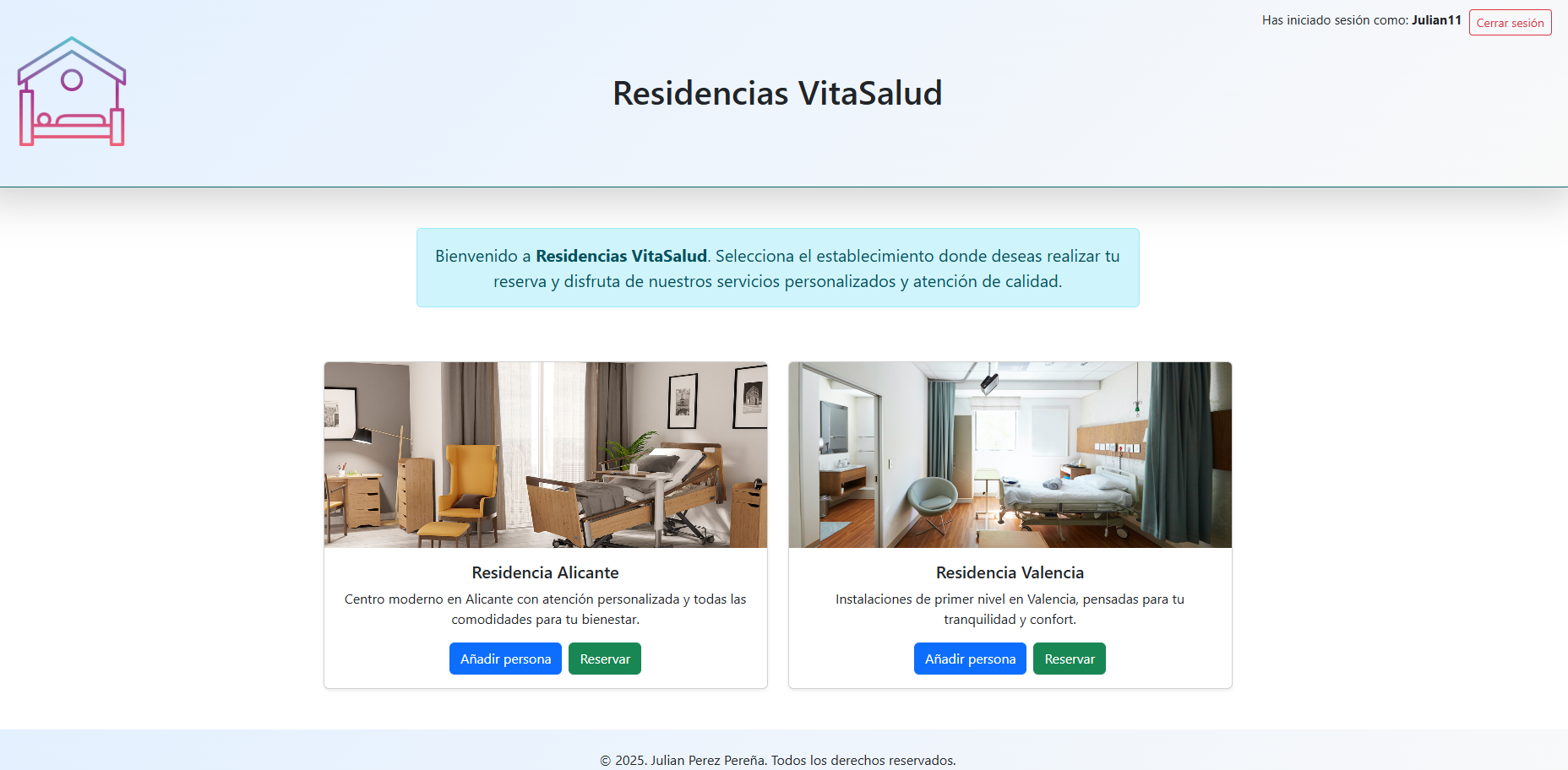
**Vista de Administración**

La vista de administración proporciona una interfaz para que los administradores gestionen usuarios, reservas, establecimientos, habitaciones y personas. Los administradores pueden ver todas las listas, filtrar y realizar operaciones como la creación, actualización y eliminación de estos elementos



**Vista de Usuario**

La vista de usuario permite añadir una persona/paciente a la base de datos y realizar una nueva reserva según el establecimiento escogido.



**Uso de Git para el Control de Versiones**

Se ha utilizado Git para el control de versiones del proyecto.

Git es un sistema de control de versiones distribuido que permite a los desarrolladores rastrear los cambios en el código fuente, colaborar con otros desarrolladores y gestionar diferentes versiones de un proyecto. Git facilita la gestión de proyectos de software, proporcionando herramientas para la creación de ramas, la fusión de cambios y la resolución de conflictos.

**En el siguiente enlace se encuentra el repositorio en git:** [**Julian-1994/Gestor-Alojamiento-Residencia**](https://github.com/Julian-1994/Gestor-Alojamiento-Residencia)

**Futuras Mejoras**

Aunque el sistema ya ofrece una amplia gama de funcionalidades, hay áreas que pueden mejorarse para aumentar su eficacia y usabilidad. En particular, es necesario implementar la validación de datos y mostrar alertas/notificaciones al usuario cuando un dato introducido no es correcto. Estas mejoras garantizarán que los datos ingresados sean precisos y completos, mejorando la experiencia del usuario y la integridad de la información.

Además, el sistema puede seguir evolucionando con futuras mejoras, como la integración de nuevas funcionalidades, la optimización del rendimiento y la ampliación de las capacidades de análisis de datos.

**Agradecimientos**

Quiero agradecer en especial a Lomax James-Robert, Abraham Oscar y Le-Rhun Maugan por su ayuda y colaboración durante el desarrollo, me fueron de gran ayuda para superar obstáculos y seguir avanzando adecuadamente en el desarrollo del proyecto