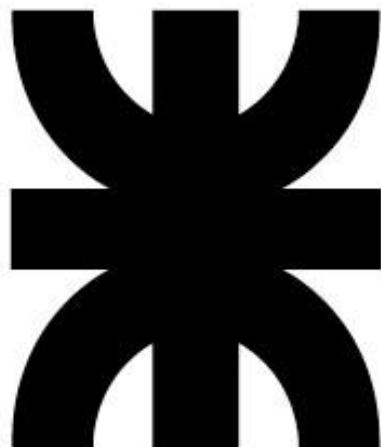


Desarrollo de Aplicaciones Cliente Servidor

TRABAJO PRÁCTICO INTEGRADOR



*Universidad Tecnológica Nacional
Facultad Regional Resistencia*

Grupo N°: 4

Año: 2022

Alumnos:

- Alaez, Magalí Sol - magui.alaez@gmail.com
- Campestrini, Luciana - lucianacampestrini1@gmail.com
- Insaurralde, Franco - insaurraldef11@gmail.com
- Maidana, Fatima - fatimamaidana3@gmail.com
- Perez, Estefanía - estefanianabell@gmail.com



Índice

Informe	2
ESCENARIO	2
LENGUAJE	2
BASE DE DATOS	3
FRAMEWORKS	3
PATRÓN DE DISEÑO	4
ESTRUCTURA DEL PROYECTO	5
BUENAS PRÁCTICAS	7
LIBRERÍAS / DEPENDENCIAS	7
DIAGRAMA DE CLASES	8
MÓDULO A DESARROLLAR	9
CAPTURAS DE PANTALLA DE PARTES DEL CÓDIGO	11
Bibliografía	37



Informe

ESCENARIO

En vísperas del mundial de Qatar 2022 la empresa Fantastic Tour (FANTUR S.A.) ha solicitado a los alumnos de la materia Desarrollo de Aplicaciones Cliente-Servidor el desarrollo de un sistema para la venta y administración de paquetes turísticos.

El sistema debe permitir a los clientes registrar en forma electrónica, realizar consultas de paquetes, reservar paquetes y abonar los mismos por diferentes medios (tarjetas de crédito u otros sistemas de pago on-line) y enviar publicidad (vía e-mail) a sus clientes.

La mayoría de los paquetes turísticos que la empresa comercializa están compuestos por pasajes aéreos o en micro, estadía en hoteles, seguros médicos COVID-19 y entradas a espectáculos. La aplicación debe contemplar el armado y cotización de estos paquetes turísticos por parte de los administradores de la empresa, pudiendo establecerse cuáles paquetes están disponibles o hasta qué fecha pueden adquirirse.

Debido a las imposiciones impuestas por los organismos de control que rigen la actividad del sector, las agencias de turismos (incluida FANTUR) deben solicitar permisos al organismo de control antes de confirmar las operaciones a sus clientes. Este tipo de solicitudes deber ser realizar en forma on-line y por medio de un web-service que el organismo provee.

Contar con una aplicación para teléfonos móviles donde los usuarios puedan consultar u operar sería una muy buena ventaja competitiva para esta empresa.

LENGUAJE

Java: Es un tipo de lenguaje de programación y una plataforma informática, creada y comercializada por Sun Microsystems en el año 1995.



Características:

- **Simple**
- **Orientado a Objetos:** Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo. Las plantillas de objetos son llamadas, como en C++, clases y sus copias, instancias. Estas instancias, como en C++, necesitan ser construidas y destruidas en espacios de memoria. Java incorpora funcionalidades inexistentes en C++ como por ejemplo, la resolución dinámica de métodos. Esta característica deriva del lenguaje Objective C, propietario del sistema operativo Next. En C++ se suele trabajar con librerías dinámicas (DLLs) que obligan a recomilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada RTTI (RunTime Type Identification) que define la interacción entre objetos excluyendo variables de instancias o implementación de métodos. Las clases en Java tienen una representación en el runtime que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda.



- **Distribuido:** Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales. La verdad es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.
- **Independiente de la plataforma**
- **Recolector de basura:** Cuando no hay referencias localizadas a un objeto, el recolector de basura de Java borra dicho objeto, liberando así la memoria que ocupaba. Esto previene posibles fugas de memoria.
- **Seguro y sólido:** administra automáticamente la memoria, provee canales de comunicación segura protegiendo la privacidad de los datos y, al tener una sintaxis rigurosa evita que se quiebre el código, es decir, no permite la corrupción del mismo.
- **Multihilo:** Java logra llevar a cabo varias tareas simultáneamente dentro del mismo programa. Esto permite mejorar el rendimiento y la velocidad de ejecución.

Ventajas:

- Se adapta a la perfección a todo tipo de dispositivos (tablets, smartphones, computadoras, laptops) permitiendo ver cualquier contenido del sitio web.
- Es posible diseñar casi cualquier elemento o aplicación.
- Es posible crear, mediante XML, páginas web dinámicas y atractivas.
- Permite incluir sonido y objetos multimedia, así como bases de datos y otras funcionalidades.

BASE DE DATOS

Microsoft SQL Server: es un sistema de gestión de bases de datos relacionales (RDBMS) que admite una amplia variedad de aplicaciones de procesamiento de transacciones, inteligencia empresarial y análisis en entornos informáticos corporativos. Microsoft SQL Server es una de las tres tecnologías de bases de datos líderes del mercado, junto con Oracle Database y DB2 de IBM. Al igual que otros programas RDBMS, Microsoft SQL Server se basa en SQL, un lenguaje de programación estandarizado que los administradores de bases de datos (DBA) y otros profesionales de TI utilizan para gestionar las bases de datos y consultar los datos que contienen. SQL Server está vinculado a Transact-SQL (T-SQL), una implementación de SQL de Microsoft que añade un conjunto de extensiones de programación propias al lenguaje estándar.



FRAMEWORKS

Hibernate: es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.



Hibernate es software libre, distribuido bajo los términos de la licencia GNU LGPL.



busca solucionar el problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación: el usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen.

Hibernate convertirá los datos entre los tipos utilizados por Java y los definidos por SQL.



Java Spring Framework (Spring Framework): Es un marco popular, de código abierto y de nivel empresarial para crear aplicaciones independientes de nivel de producción que se ejecutan en la máquina virtual de Java (JVM).



Java Spring Boot (Spring Boot): es una herramienta que hace que el desarrollo de aplicaciones web y microservicios con Spring Framework sea más rápido y fácil a través de tres capacidades principales:

1. Autoconfiguración
2. Un enfoque obstinado de la configuración
3. La capacidad de crear aplicaciones independientes

PATRÓN DE DISEÑO

Patrón Repositorio: consiste en separar la lógica que recupera los datos y los asigna a un modelo de entidad de la lógica de negocios que actúa sobre el modelo, esto permite que la lógica de negocios sea independiente del tipo de dato que comprende la capa de origen de datos, en pocas palabras un repositorio media entre el dominio y las capas de mapeo de datos, actuando como una colección de objetos de dominio en memoria.

Ventajas:

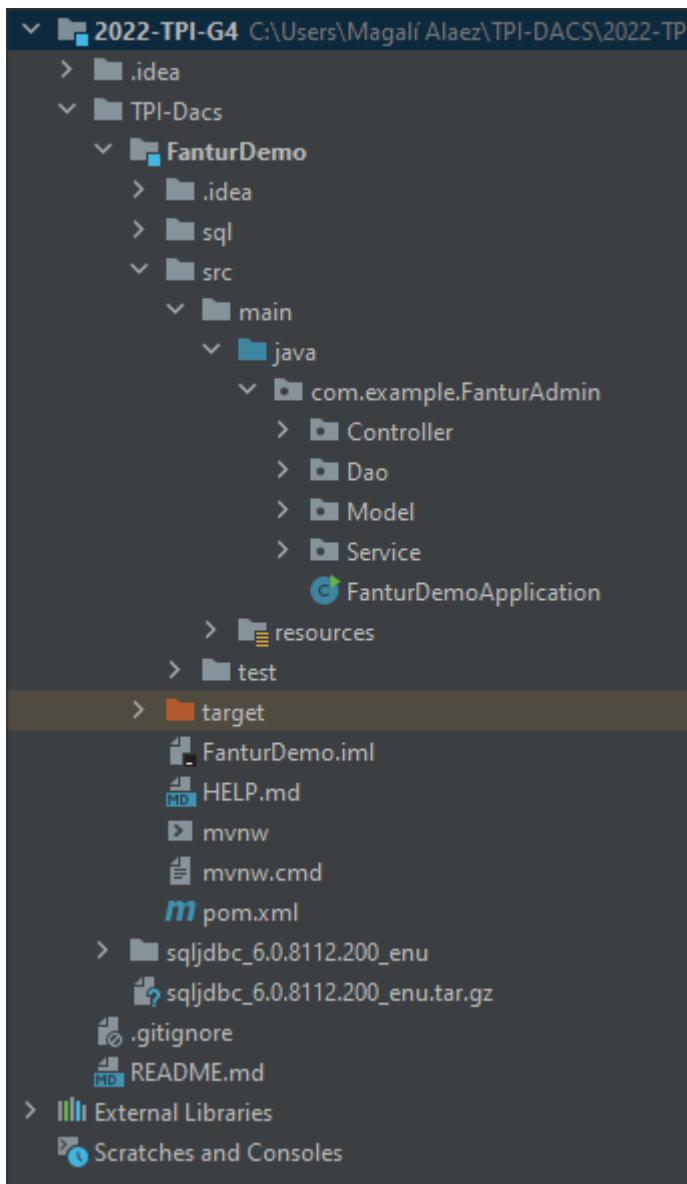
- El código de acceso a los datos puede ser reutilizado.
- Es fácil de implementar la lógica del dominio.
- Nos ayuda a desacoplar la lógica.
- La lógica de negocio puede ser probada fácilmente sin acceso a los datos.
- También es una buena manera de implementar la inyección de dependencia que hace que el código sea más testable.

DAO

El patrón DAO propone separar por completo la lógica de negocio de la lógica para acceder a los datos, de esta forma, el DAO proporcionará los métodos necesarios para insertar, actualizar, borrar y consultar la información; por otra parte, la capa de negocio solo se preocupa por la lógica de negocio y utiliza el DAO para interactuar con la fuente de datos.



ESTRUCTURA DEL PROYECTO



Construimos un Spring Boot CRUD Rest Apis usando Spring Data JPA con la base de datos de SQL Server (MSSQL) en la que:

- Cada clase tiene id, title, description, published status.
- Apis para crear (create), recuperar (retrieve), actualizar (update), eliminar (delete).
- Apis que también admitan métodos de búsqueda personalizados (por id).

Por ejemplo, estas son las API que proporcionamos para cliente:

Método	Urls	Acción
POST	/api/clientes	crea un nuevo cliente
GET	/api/clientes	recuperar todas las clases
GET	/api/clientes/:id	recuperar un cliente con ese id



PUT	/api/clientes/:id	actualizar un cliente con ese id
DELETE	/api/clientes/:id	eliminar un cliente con ese id
DELETE	/api/clientes	eliminar todos los clientes
GET	/api/clientes/published	encontrar todos los clientes publicados

CAPA DE DOMINIO

Carpeta “Model”: es donde se definen todas las clases y se establece la relación de cada una con su tabla correspondiente en la base de datos. Para ello utilizamos:

- **@Entity:** La anotación indica que la clase es una clase Java persistente.
- **@Table:** proporciona la tabla que mapea esta entidad.
- **@Id:** es para la clave principal.
- **@GeneratedValue:** se utiliza para definir la estrategia de generación de la clave principal. GenerationType.AUTO significa campo de incremento automático.
- **@Column:** se utiliza para definir la columna en la base de datos que asigna el campo anotado.

CAPA DE ACCESO A DATOS

Carpeta “Dao”: por cada clase definimos una interfaz que amplía JpaRepository para métodos CRUD y métodos de búsqueda personalizados. Se conectará automáticamente con el controlador correspondiente a esa clase.

LÓGICA DE NEGOCIO

Carpeta “Controller”: es un RestController por cada clase que tiene métodos de mapeo de solicitudes para solicitudes RESTful como: getAll (obtener todos), create(crear), update (actualizar), delete (eliminar), findByPublished (encontrar por id), entre otros. Para ello utilizamos:

- **@CrossOrigin** es para configurar los orígenes permitidos.
- **@RestController** la anotación se utiliza para definir un controlador y para indicar que el valor de retorno de los métodos debe vincularse al cuerpo de la respuesta web.
- **@RequestMapping("/api")** declara que todas las URL de Apis en el controlador comenzarán con /api.
- **@Autowired** para injectar el bean TutorialRepository a la variable local.

CAPA DE SERVICIO

Carpeta “Service”: por cada clase se define una servicio y una interfaz de ese servicio. En el servicio de cada una se establecen los métodos de listar todos los objetos de la clase, guardar eliminar y encontrar por id, además de otros métodos específicos de la clase.

Application properties: Configuración para Spring Datasource, JPA e Hibernate.

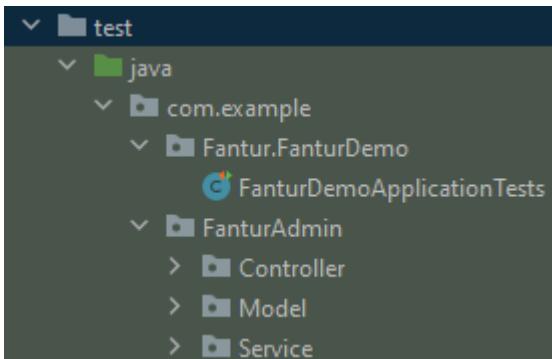
pom.xml: contiene dependencias para Spring Boot y SQL Server.

Ejecutar la aplicación: lo haremos con el comando mvn spring-boot:run. La tabla de tutoriales se generará automáticamente en la base de datos de Microsoft SQL Server.



BUENAS PRÁCTICAS

Uso de TESTS:

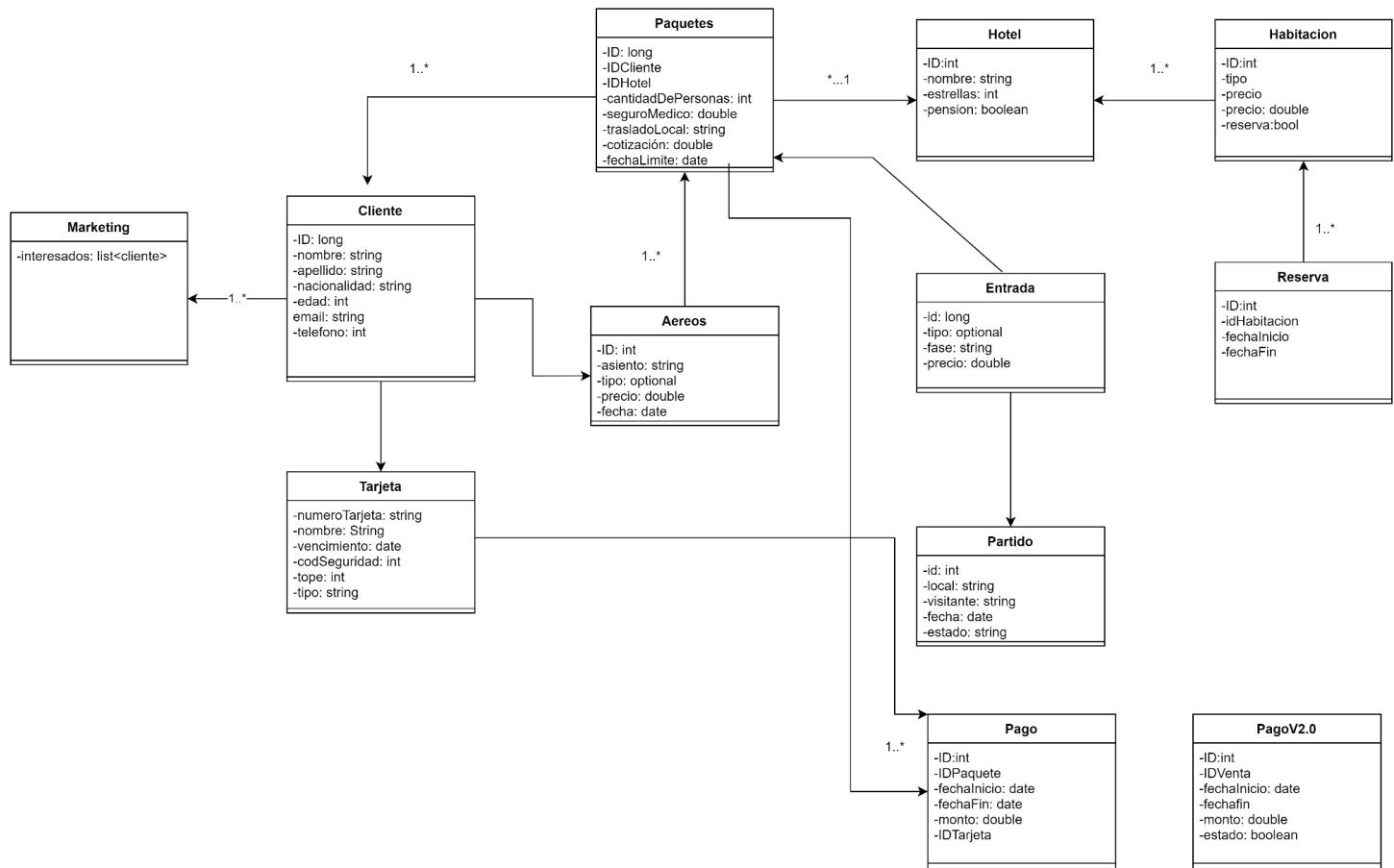


LIBRERÍAS / DEPENDENCIAS

- **Spring Boot JPA** Spring Data JPA , parte de la gran familia de Spring Data , facilita la implementación de repositorios basados en JPA. JPA es la propuesta estándar que ofrece Java para implementar un Framework Object Relational Mapping (ORM), que permite interactuar con la base de datos por medio de objetos, de esta forma, JPA es el encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB).
 - spring-boot-starter-data-jpa
- **Spring Boot Starter POMs** es una utilidad existente dentro de Spring Boot que facilita la creación y configuración de una aplicación Java.
 - spring-boot-starter-web
 - spring-boot-starter-thymeleaf
 - spring-boot-starter-test
- **Spring Boot DevTools** es la herramienta de Spring Boot que nos permite reiniciar de forma automática nuestras aplicaciones cada vez que se produce un cambio en nuestro código.
 - spring-boot-devtools
- **Spring Boot Maven Plugin** proporciona compatibilidad con Spring Boot en Apache Maven . Le permite empaquetar archivos jar o war ejecutables, ejecutar aplicaciones Spring Boot, generar información de compilación e iniciar su aplicación Spring Boot antes de ejecutar las pruebas de integración.
 - spring-boot-maven-plugin
- **Microsoft JDBC para SQL Server** este controlador es un controlador JDBC de tipo 4 que proporciona conectividad a la base de datos a través de las interfaces de programa de aplicación (API) estándar de JDBC.
 - mssql-jdbc
- **JUnit** es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. **JUnit Jupiter** nos permite utilizar el nuevo modelo de programación para la escritura de los nuevos tests de JUnit 5.
 - junit
 - junit-jupiter-engine

- **Mockito** es un marco de simulación de Java que tiene como objetivo proporcionar la capacidad de escribir con claridad una prueba de unidad legible utilizando su API simple. Se diferencia de otros marcos de simulacros al dejar el patrón de esperar-ejecutar-verificar que la mayoría de los otros marcos utilizan.
 - Mockito-core
- **Hibernate** es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java.
 - hibernate-core
- **Java Architecture for XML Binding (JAXB)** permite a los desarrolladores Java asignar clases de Java a representaciones XML. JAXB proporciona dos características principales: la capacidad de serializar las referencias de objetos Java a XML y la inversa, es decir, deserializar XML en objetos Java. En otras palabras, JAXB permite almacenar y recuperar datos en memoria en cualquier formato XML, sin la necesidad de implementar un conjunto específico de rutinas de carga y guardado de XML para la estructura de clases del programa.
 - jaxb-api
- **Apache maven compiler plugin**, el compiler plugin de apache es utilizado para compilar el código fuente del proyecto. ¿Por qué es importante? La configuración default del código fuente es 1.5, de tal modo que si tu aplicación utiliza cualquier novedad de java 1.6 o superior el código fuente no compilará.
 - maven-compiler-plugin

DIAGRAMA DE CLASES





MÓDULO A DESARROLLAR

Armado del paquete:

El cliente va a poder armar un paquete personalizado, con los partidos a los que desea asistir, cantidad de días que desea viajar y demás. A partir de los datos tomados de lo que desea el cliente se va a establecer una cotización. Para establecer esta cotización se va a determinar con cuanto tiempo de anticipación se está haciendo la reserva, a menor anticipación mayor interés.

```
@Override  
@Transactional  
public void guardar(Cliente cliente, Hotel hotel, int cantPersonas, float seguro,  
                     String traslado, List<Entrada> entradaList, List<Habitacion> habitaciones,  
                     Date fechaInicio, Date fechaFin, List<Aereos> pasajes, Tarjeta tarjeta) {  
    float cotizacion=0;  
    float precioEntradas=0;  
    float precioHabitaciones=0;  
    float precioAereos=0;  
    Paquete paquete=new Paquete();  
    for(Entrada entrada : entradaList)  
    {  
        entrada.setPaquete(paquete);  
        entradaService.guardar(entrada);  
        precioEntradas+=entrada.getPrecio();  
    }  
  
    for(Habitacion habitacion : habitaciones)  
    {  
        reservaService.guardar(new Reserva(habitacion,fechaInicio,fechaFin));  
        precioHabitaciones+=habitacion.getPrecio();  
    }  
}
```



```
for(Aereos pasaje : pasajes)
{
    pasaje.setClienteTitular(cliente);
    pasaje.setPaquete(paquete);
    aereosService.guardar(pasaje);
    precioAereos+=pasaje.getPrecio();
}

float interes=0;
LocalDate mundial = LocalDate.ofEpochDay(2022-11-01);
int diasrestantes = (int) ChronoUnit.DAYS.between(LocalDate.now(),mundial);

if (diasrestantes > 50)
{
    interes = 1.1F;
} else if (diasrestantes > 30) {
    interes = 1.3F;
} else {
    interes = 1.5F;
}
cotizacion=(precioEntradas+precioAereos+precioHabitaciones)*interes;
```

Pago del paquete:

Para efectuar el pago primero se va a verificar que el saldo de la tarjeta sea suficiente para cubrir el mismo. Luego se va a evaluar si este pago es aprobado o no. Si el saldo es suficiente y está aprobado se realiza el pago.

```
@Override
@Transactional
public void guardar(Paquete paquete, Date fechaInicio, Date fechaFin, Cliente cliente, Tarjeta tarjeta ) {
    if(tarjeta.getTope()>=paquete.getCotizacion()){
        Pagov2 pagov2= new Pagov2();
        pagov2.setAprobada(true);
        // pagov2= servicioProfe(cliente.getCuit(),fechaInicio,fechaFin,paquete.getCotizacion());
        if(pagov2.isAprobada()){
            pagoDao.save(new Pago(paquete, LocalDate.now(), paquete.getCotizacion(),cliente,tarjeta));
        }
    }
    else {
        System.out.println("El monto de la tarjeta es insuficiente para efectuar el pago");
    }
}
```



CAPTURAS DE PANTALLA DE PARTES DEL CÓDIGO

Model/Aereos

The screenshot shows a code editor with the file 'Aereos.java' open. The code defines a Java class 'Aereos' annotated with @Entity and @Table(name = "Aereos"). The class has four fields: 'ID' (Long, @Id, @GeneratedValue(strategy = GenerationType.AUTO)), 'asiento' (String, @Column(name = "asiento", nullable = false)), 'clase' (String, @Column(name = "clase", nullable = false)), and 'precio' (float, @Column(name = "precio", nullable = false)). The code is color-coded, and line numbers are visible on the left.

```
1 package com.example.FanturAdmin.Model;
2 import com.fasterxml.jackson.annotation.JsonBackReference;
3
4 import javax.persistence.*;
5
6 import java.util.Date;
7
8 * EstefaniaPerez +2
9
10 @Entity
11 @Table(name = "Aereos")
12 public class Aereos {
13
14     3 usages
15     @Id
16     @GeneratedValue(strategy = GenerationType.AUTO)
17     private Long ID;
18     3 usages
19     @Column(name = "asiento", nullable = false)
20     private String asiento;
21     3 usages
22     @Column(name = "clase", nullable = false)
23     private String clase; //TURISTA,PRIMERA_CLASE
24     3 usages
25     @Column(name = "precio", nullable = false)
26     private float precio;
27     3 usages
28     @Column(name = "fecha", nullable = false)
```



```
READMD.md Aereos.java
1  @Column(name = "fecha", nullable = false)
2  private Date fecha;
3  3 usages
4  @ManyToOne()
5  @JoinColumn(name = "idPaquete")
6  @JsonBackReference
7  private Paquete paquete;
8  3 usages
9  @ManyToOne()
10 @JoinColumn(name = "idCliente")
11 @JsonBackReference
12
13
14  private Cliente clienteTitular;
15
16
17  ▲ EstefaniaPerez
18  public Aereos(Long ID, String asiento, String clase, float precio, Date fecha) {
19      this.ID = ID;
20      this.asiento = asiento;
21      this.clase = clase;
22      this.precio = precio;
23      this.fecha = fecha;
24      this.paquete = null;
25      this.clienteTitular = null;
26  }
27
```

```
READMD.md Aereos.java
43
44  public Aereos(){}
45
46  ▲ EstefaniaPerez
47  public Long getID() { return ID; }
48
49  ▲ EstefaniaPerez
50  public void setID(Long ID) { this.ID = ID; }
51
52  ▲ EstefaniaPerez
53  public String getAsiento() { return asiento; }
54
55  ▲ EstefaniaPerez
56  public void setAsiento(String asiento) { this.asiento = asiento; }
57
58  ▲ EstefaniaPerez
59  public String getClase() { return clase; }
60
61  ▲ EstefaniaPerez
62  public void setClase(String clase) { this.clase = clase; }
63
64
65  1 usage ▲ EstefaniaPerez
66  public float getPrecio() { return precio; }
67
68  ▲ EstefaniaPerez
69  public void setPrecio(float precio) { this.precio = precio; }
70
71
```



```
▲ EstefaniaPerez
public void setPrecio(float precio) { this.precio = precio; }

▲ EstefaniaPerez
public Date getFecha() { return fecha; }

▲ EstefaniaPerez
public void setFecha(Date fecha) { this.fecha = fecha; }

▲ EstefaniaPerez
public Paquete getPaquete() { return paquete; }

▲ EstefaniaPerez
public void setPaquete(Paquete paquete) { this.paquete = paquete; }

▲ EstefaniaPerez
public Cliente getClienteTitular() { return clienteTitular; }

1 usage ▲ EstefaniaPerez
public void setClienteTitular(Cliente clienteTitular) { this.clienteTitular = clienteTitular; }
```



Model/Cliente

```
1 README.md   Cliente.java
2
3 package com.example.FanturAdmin.Model;
4
5
6 import javax.persistence.*;
7
8
9 /**
10  * Estefania Perez +2
11  */
12 @Entity
13 @Table(name = "Cliente")
14 public class Cliente {
15
16
17     /**
18      * 3 usages
19      */
20     @Id
21     @GeneratedValue(strategy = GenerationType.AUTO)
22     private Long ID;
23
24
25     /**
26      * 3 usages
27      */
28     @Column(name = "cuit", nullable = false)
29     private String cuit;
30
31     /**
32      * 3 usages
33      */
34     @Column(name = "nombre", nullable = false)
35     private String nombre;
36
37     /**
38      * 3 usages
39      */
40     @Column(name = "apellido", nullable = false)
41     private String apellido;
42
43     /**
44      * 3 usages
45      */
46     @Column(name = "nacionalidad", nullable = false)
47     private String nacionalidad;
```



```
README.md  Cliente.java
    @Column(name = "edad", nullable = false)
    private int edad;
    3 usages
    @Column(name = "email", nullable = false)
    private String email;
    3 usages
    @Column(name = "telefono", nullable = false)
    private String telefono;

    ▲ EstefaniaPerez +1"
    public Cliente(Long ID, String cuit, String nombre, String apellido, String nacionalidad, int edad,
                  String email, String telefono) {
        this.ID = ID;
        this.cuit = cuit;
        this.nombre = nombre;
        this.apellido = apellido;
        this.nacionalidad = nacionalidad;
        this.edad = edad;
        this.email = email;
        this.telefono = telefono;
    }

    ▲ EstefaniaPerez
    public Cliente() {
    }
```

```
README.md  Cliente.java
    public Long getID() { return ID; }

    ▲ EstefaniaPerez
    public void setID(Long ID) { this.ID = ID; }

    ▲ lucianacampestrini
    public String getCuit() { return cuit; }

    ▲ lucianacampestrini
    public void setCuit(String cuit) { this.cuit = cuit; }

    ▲ EstefaniaPerez
    public String getNombre() { return nombre; }

    ▲ EstefaniaPerez
    public void setNombre(String nombre) { this.nombre = nombre; }

    ▲ EstefaniaPerez
    public String getApellido() { return apellido; }

    ▲ EstefaniaPerez
    public void setApellido(String apellido) { this.apellido = apellido; }

    ▲ EstefaniaPerez
    public String getNacionalidad() { return nacionalidad; }
```



```
EADME.md - ClienteJava
public void setApellido(String apellido) { this.apellido = apellido; }

▲ EstefaniaPerez
public String getNacionalidad() { return nacionalidad; }

▲ EstefaniaPerez
public void setNacionalidad(String nacionalidad) { this.nacionalidad = nacionalidad; }

▲ EstefaniaPerez
public int getEdad() { return edad; }

▲ EstefaniaPerez
public void setEdad(int edad) { this.edad = edad; }

▲ EstefaniaPerez
public String getEmail() { return email; }

▲ EstefaniaPerez
public void setEmail(String email) { this.email = email; }

▲ EstefaniaPerez
public String getTelefono() { return telefono; }

▲ EstefaniaPerez
public void setTelefono(String telefono) { this.telefono = telefono; }
```



Model/Habitacion

The screenshot shows a code editor with a dark theme. The file being viewed is `Habitacion.java`, located in the package `com.example.FanturAdmin.Model`. The code defines a Java class `Habitacion` annotated with `@Entity` and `@Table(name = "Habitacion")`. It contains three fields: `ID` (a long type with `GeneratedValue(strategy = GenerationType.AUTO)`), `tipo` (a string type), and `precio` (a float type). There is also a many-to-one relationship to the `Hotel` entity, indicated by the `@ManyToOne` and `@JoinColumn(name = "iDHotel")` annotations.

```
package com.example.FanturAdmin.Model;

import javax.persistence.*;

30 usages + MagaAlaez +2
@Entity
@Table(name = "Habitacion")
public class Habitacion {
    3 usages
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long ID;
    3 usages
    @Column(name = "tipo", nullable = false)
    private String tipo;
    3 usages
    @Column(name = "precio", nullable = false)
    private float precio;
    3 usages
    @ManyToOne()
    @JoinColumn(name = "iDHotel")
    private Hotel hotel;
```



The screenshot shows a code editor with the file `Habitacion.java` open. The code is annotated with commit history from a version control system, likely GitHub. The commits are as follows:

- `MagaaAlaez +1`:
 `public Habitacion(Long ID, String tipo, float precio, Hotel hotel) {`
 `this.ID = ID;`
 `this.tipo = tipo;`
 `this.precio = precio;`
 `this.hotel = hotel;`
 `}`
- `MagaaAlaez`:
 `public Habitacion() { }`
- `MagaaAlaez +1`:
 `public Long getID() { return ID; }`
- `MagaaAlaez +1`:
 `public void setID(Long ID) { this.ID = ID; }`
- `MagaaAlaez`:
 `public String getTipo() { return tipo; }`
- `MagaaAlaez`:
 `public void setTipo(String tipo) { this.tipo = tipo; }`
- `1 usage MagaaAlaez`:
 `public float getPrecio() { return precio; }`



The screenshot shows a code editor with the file `Habitacion.java` open. The code defines a class `Habitacion` with various methods and annotations. The code is color-coded, and some parts are highlighted in yellow.

```
README.md  Habitacion.java
+
+ MagaAlaez
public Habitacion() { }

+
+ MagaAlaez +1
public LonggetID() { return ID; }

+
+ MagaAlaez +1
public void setID(Long ID) { this.ID = ID; }

+
+ MagaAlaez
public String getTipo() { return tipo; }

+
+ MagaAlaez
public void setTipo(String tipo) { this.tipo = tipo; }

1 usage  + MagaAlaez
public float getPrecio() { return precio; }

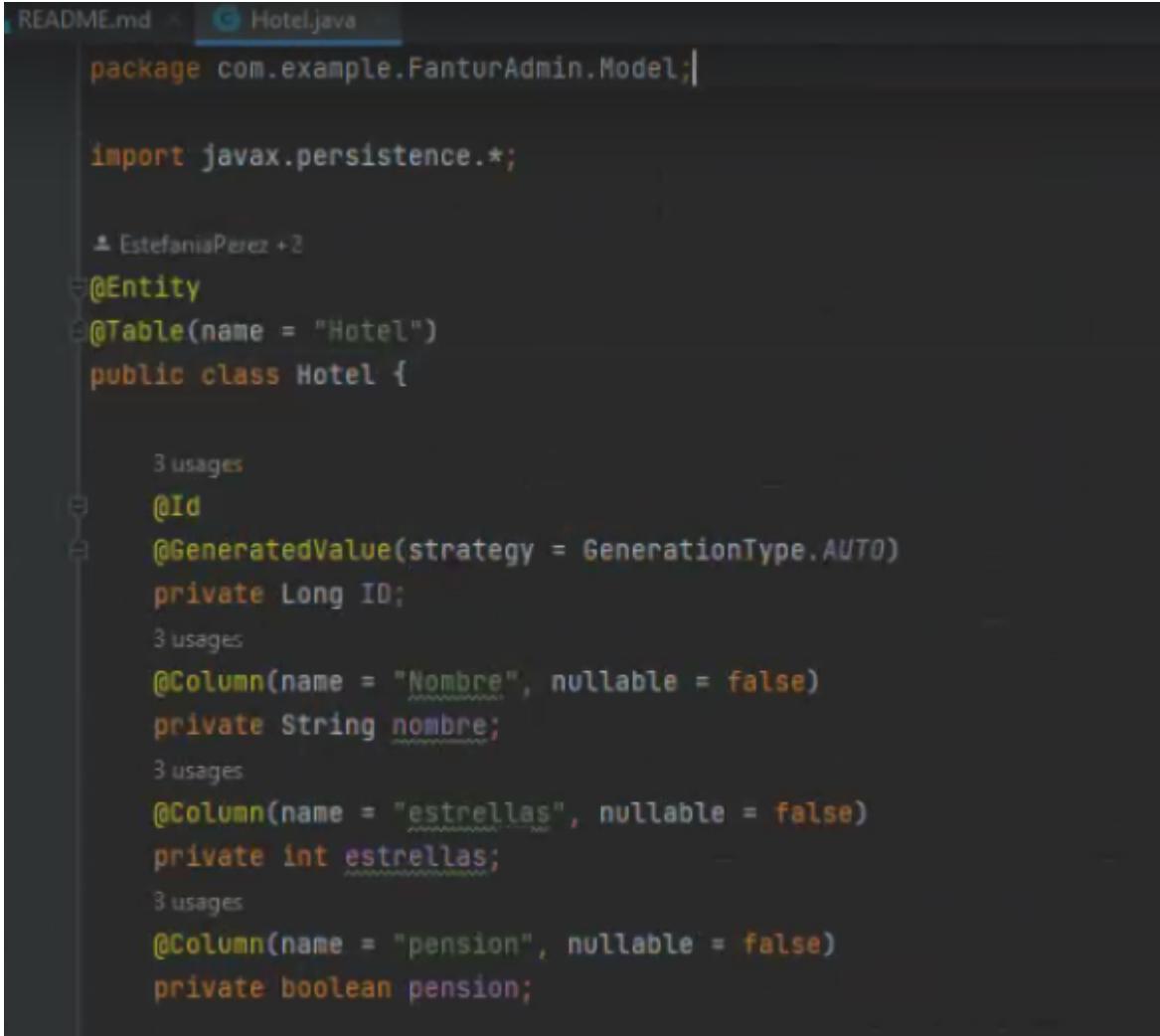
+
+ MagaAlaez
public void setPrecio(float precio) { this.precio = precio; }

+
+ EstefaniaPerez
public Hotel getHotel() { return hotel; }

+
+ EstefaniaPerez
public void setHotel(Hotel hotel) { this.hotel = hotel; }
```



Model/Hotel



The screenshot shows a Java code editor with a file named Hotel.java selected. The code defines a class Hotel with three columns: ID, Nombre, and estrellas.

```
package com.example.FanturAdmin.Model;

import javax.persistence.*;

@Entity
@Table(name = "Hotel")
public class Hotel {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long ID;
    @Column(name = "Nombre", nullable = false)
    private String nombre;
    @Column(name = "estrellas", nullable = false)
    private int estrellas;
    @Column(name = "pension", nullable = false)
    private boolean pension;
}
```



```
README.md  Hotel.java
public Hotel(Long ID, String nombre, int estrellas, boolean pension) {
    this.ID = ID;
    this.nombre = nombre;
    this.estrellas = estrellas;
    this.pension = pension;
}

▲ EstefaniaPerez
public Hotel(){ }

▲ EstefaniaPerez
public Long getID() { return ID; }

▲ EstefaniaPerez
public void setID(Long ID) { this.ID = ID; }

▲ EstefaniaPerez
public String getNombre() { return nombre; }

▲ EstefaniaPerez
public void setNombre(String nombre) { this.nombre = nombre; }

▲ EstefaniaPerez
public int getEstrellas() { return estrellas; }
```



```
 README.md  Hotel.java
1   public Hotel(){}
2
3     ▲ EstefaniaPerez
4     public Long getId() { return ID; }
5
6     ▲ EstefaniaPerez
7     public void setId(Long ID) { this.ID = ID; }
8
9     ▲ EstefaniaPerez
10    public String getNombre() { return nombre; }
11
12    ▲ EstefaniaPerez
13    public void setNombre(String nombre) { this.nombre = nombre; }
14
15    ▲ EstefaniaPerez
16    public int getEstrellas() { return estrellas; }
17
18    ▲ EstefaniaPerez
19    public void setEstrellas(int estrellas) { this.estrellas = estrellas; }
20
21    ▲ EstefaniaPerez
22    public boolean isPension() { return pension; }
23
24    ▲ EstefaniaPerez
25    public void setPension(boolean pension) { this.pension = pension; }
```



Model/Paquete

```
1 package com.example.FanturAdmin.Model;
2
3 import javax.persistence.*;
4 import java.util.Date;
5
6 // EstefaniaPerez +1
7 @Entity
8 @Table(name = "Paquete")
9 public class Paquete {
10
11     3 usages
12     @Id
13     @GeneratedValue(strategy = GenerationType.AUTO)
14     private Long ID;
15
16     3 usages
17     @Column(name = "cantidadDePersonas")
18     private int cantidadDePersonas;
19     3 usages
20     @Column(name = "seguroMedico")
21     private float seguroMedico;
22     3 usages
23     @Column(name = "trasladoLocal")
24     private String trasladoLocal;
25     3 usages
```



The screenshot shows two code snippets from a Java file named 'Paquete.java' in an IDE.

Top Snippet:

```
1 README.md  Paquete.java
2
3     @ManyToOne()
4     @JoinColumn(name = "iDHotel")
5     private Hotel estadiaHotel;
6     3 usages
7
8     @ManyToOne()
9     @JoinColumn(name = "iDCliente")
10    private Cliente clienteTitular;
11
12
13    ▲ EstefaniaPerez *
14    public Paquete(Long ID, int cantidadDePersonas, float seguroMedico, String trasladoLocal,
15                  float cotizacion, Hotel estadiaHotel, Cliente clienteTitular) {
16        this.ID = ID;
17        this.cantidadDePersonas = cantidadDePersonas;
18        this.seguroMedico = seguroMedico;
19        this.trasladoLocal = trasladoLocal;
20        this.cotizacion = cotizacion;
21        this.estadiaHotel = estadiaHotel;
22        this.clienteTitular = clienteTitular;
23    }
24
25
26    ▲ EstefaniaPerez
27    public Paquete(){ }
28
29
30    ▲ EstefaniaPerez
31    public Long getID() { return ID; }
```

Bottom Snippet:

```
1 README.md  Paquete.java
2
3     public Paquete(){ }
4
5     ▲ EstefaniaPerez
6     public LonggetID() { return ID; }
7
8     ▲ EstefaniaPerez
9     public void setID(Long ID) { this.ID = ID; }
10
11
12     ▲ EstefaniaPerez
13     public int getCantidadDePersonas() { return cantidadDePersonas; }
14
15     2 usages ▲ EstefaniaPerez
16     public void setCantidadDePersonas(int cantidadDePersonas) { this.cantidadDePersonas = cantidadDePersonas; }
17
18     ▲ EstefaniaPerez
19     public float getSeguroMedico() { return seguroMedico; }
20
21     2 usages ▲ EstefaniaPerez
22     public void setSeguroMedico(float seguroMedico) { this.seguroMedico = seguroMedico; }
23
24     ▲ EstefaniaPerez
25     public String getTrasladoLocal() { return trasladoLocal; }
26
27     2 usages ▲ EstefaniaPerez
28     public void setTrasladoLocal(String trasladoLocal) { this.trasladoLocal = trasladoLocal; }
```



```
README.md Paquetejava

  ↳ EstefaniaPerez
    public String getTrasladoLocal() { return trasladoLocal; }

    2 usages ↳ EstefaniaPerez
    public void setTrasladoLocal(String trasladoLocal) { this.trasladoLocal = trasladoLocal; }

    ↳ EstefaniaPerez
    public float getCotizacion() { return cotizacion; }

    2 usages ↳ EstefaniaPerez
    public void setCotizacion(float cotizacion) { this.cotizacion = cotizacion; }

    ↳ EstefaniaPerez
    public Hotel getEstadiaHotel() { return estadiaHotel; }

    2 usages ↳ EstefaniaPerez
    public void setEstadiaHotel(Hotel estadiaHotel) { this.estadiaHotel = estadiaHotel; }

    ↳ EstefaniaPerez
    public Cliente getClienteTitular() { return clienteTitular; }

    2 usages ↳ EstefaniaPerez
    public void setClienteTitular(Cliente clienteTitular) { this.clienteTitular = clienteTitular; }
```



Model/Reserva

The screenshot shows a code editor with the file `Reserva.java` open. The code defines a Java class `Reserva` annotated with `@Entity` and `@Table(name = "Reserva")`. It contains fields for an ID (`private Long ID;`), a many-to-one relationship to `Habitacion` (`private Habitacion habitacion;`), and two dates: `fechaInicio` and `fechaFin`.

```
package com.example.FanturAdmin.Model;

import javax.persistence.*;
import java.util.Date;

@entity
@Table(name = "Reserva")
public class Reserva {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long ID;

    @ManyToOne()
    @JoinColumn(name = "iDhabitacion")
    private Habitacion habitacion;

    @Column(name = "fechaInicio")
    private Date fechaInicio;

    @Column(name = "fechaFin")
    private Date fechaFin;
}
```



```
README.md  Reserva.java
public Reserva(Habitacion habitacion, Date fechaInicio, Date fechaFin){
    this.habitacion = habitacion;
    this.fechaInicio = fechaInicio;
    this.fechaFin = fechaFin;
}
▲ EstefaniaPerez
public Reserva(){};

▲ EstefaniaPerez
public Long getID() { return ID; }

▲ EstefaniaPerez
public void setID(Long ID) { this.ID = ID; }

▲ EstefaniaPerez
public Habitacion getHabitacion() { return habitacion; }

▲ EstefaniaPerez
public void setHabitacion(Habitacion habitacion) { this.habitacion = habitacion; }

▲ EstefaniaPerez
public Date getFechaInicio() { return fechaInicio; }

▲ EstefaniaPerez
public void setFechaInicio(Date fechaInicio) { this.fechaInicio = fechaInicio; }

▲ EstefaniaPerez
public Date getFechaFin() { return fechaFin; }

▲ EstefaniaPerez
public void setFechaFin(Date fechaFin) { this.fechaFin = fechaFin; }
```



ClienteController

The screenshot shows a Java code editor with the file `ClienteController.java` open. The code defines a REST controller for managing clients. It includes imports for `List`, `ClienteService`, `Autowired`, `RequestMapping`, and `Cliente`. The controller has two methods: `clienteList()` which returns a list of clients, and `cliente(Cliente cliente)` which finds a client by ID. Both methods are annotated with `@GetMapping`.

```
package com.example.FanturAdmin.Controller;

import java.util.List;
import com.example.FanturAdmin.Service.ClienteService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import com.example.FanturAdmin.Model.Cliente;

4 usages ▲ EstefaniaPerez
@RestController
@RequestMapping("/api/v1")
public class ClienteController {

    6 usages
    @Autowired
    private ClienteService clienteService;

    ▲ EstefaniaPerez
    @GetMapping("/listarClientes")
    public List<Cliente> clienteList() { return clienteService.listarClientes(); }

    ▲ EstefaniaPerez
    @GetMapping("/buscarCliente/{ID}")
    public Cliente cliente(Cliente cliente) { return clienteService.encontrarCliente(cliente); }

}
```


The second screenshot shows the same `ClienteController.java` file with additional annotations. It includes `@PostMapping` for saving a new client, `@DeleteMapping` for deleting a client by ID, and `@DeleteMapping` for deleting a client by its ID. The code is identical to the first screenshot but includes these additional annotations.

```
public class ClienteController {

    6 usages
    @Autowired
    private ClienteService clienteService;

    ▲ EstefaniaPerez
    @GetMapping("/listarClientes")
    public List<Cliente> clienteList() { return clienteService.listarClientes(); }

    ▲ EstefaniaPerez
    @GetMapping("/buscarCliente/{ID}")
    public Cliente cliente(Cliente cliente) { return clienteService.encontrarCliente(cliente); }

    ▲ EstefaniaPerez
    @PostMapping("/guardarCliente")
    public void guardar(@RequestBody Cliente cliente) { clienteService.guardar(cliente); }

    ▲ EstefaniaPerez
    @DeleteMapping("/borrarCliente")
    public void borrar(@RequestBody Cliente cliente) { clienteService.eliminar(cliente); }

    ▲ EstefaniaPerez
    @DeleteMapping("/borrarClientePorID/{ID}")
    public void borrar2(Cliente cliente) { clienteService.eliminar(clienteService.encontrarCliente(cliente)); }

}
```



ClienteDao

```
README.md  ClienteDao.java
package com.example.FanturAdmin.Dao;

import com.example.FanturAdmin.Model.Cliente;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

4 usages  ▲ EstefaniaPerez
@Repository
public interface ClienteDao extends JpaRepository<Cliente, Long>{}
```

PaqueteDao

```
README.md  PaqueteDao.java
1 package com.example.FanturAdmin.Dao;

2
3 import com.example.FanturAdmin.Model.Paquete;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;

6
7 4 usages  ▲ EstefaniaPerez
8 @Repository
9 public interface PaqueteDao extends JpaRepository<Paquete, Long>{
```



ClienteService

```
package com.example.FanturAdmin.Service;

import com.example.FanturAdmin.Model.Cliente;
import com.example.FanturAdmin.Dao.ClienteDao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

11 usages  ↗ EstefaniaPerez
@Service
public class ClienteService implements ClienteServiceI {

    4 usages
    @Autowired
    private ClienteDao clienteDao;

    3 usages  ↗ EstefaniaPerez
    @Override
    @Transactional(readOnly = true)
    public List<Cliente> listarClientes() { return (List<Cliente>) clienteDao.findAll(); }

    * Fisical-Phone
}
```



The screenshot shows the code for the `ClienteService.java` file. The code is annotated with usage counts and author names:

- 4 usages
@Autowired
private ClienteDao clienteDao;
- 3 usages ▲ EstefaniaPerez
@Override
@Transactional(readOnly = true)
public List<Cliente> listarClientes() { return (List<Cliente>) clienteDao.findAll(); }
- ▲ EstefaniaPerez
@Override
@Transactional
public void guardar(Cliente cliente) { clienteDao.save(cliente); }
- ▲ EstefaniaPerez
@Override
@Transactional
public void eliminar(Cliente cliente) { clienteDao.delete(cliente); }
- 4 usages ▲ EstefaniaPerez
@Override
@Transactional(readOnly = true)
public Cliente encontrarCliente(Cliente cliente) {
 return clienteDao.findById(cliente.getID()).orElse(null);
}



ClienteService Interfaz

The screenshot shows a code editor with the file `ClienteServiceI.java` open. The code defines an interface for client services. It includes imports for the package and a `Cliente` model, and a `List` utility class. The interface contains four methods: `listarClientes()`, `guardar(Cliente cliente)`, `eliminar(Cliente cliente)`, and `encontrarCliente(Cliente cliente)`. Each method is annotated with its usage count and implementation author.

```
package com.example.FanturAdmin.Service;

import com.example.FanturAdmin.Model.Cliente;
import java.util.List;

1 usage 1 implementation ▲ EstefaniaPerez
public interface ClienteServiceI {

    3 usages 1 implementation ▲ EstefaniaPerez
    List<Cliente> listarClientes();

    1 implementation ▲ EstefaniaPerez
    void guardar(Cliente cliente);

    1 implementation ▲ EstefaniaPerez
    void eliminar(Cliente cliente);

    4 usages 1 implementation ▲ EstefaniaPerez
    Cliente encontrarCliente(Cliente cliente);

}
```



PaqueteService

```
README.md  PaqueteService.java

1 package com.example.FanturAdmin.Service;
2 import com.example.FanturAdmin.Dao.PaqueteDao;
3 import com.example.FanturAdmin.Model.*;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Service;
6 import org.springframework.transaction.annotation.Transactional;
7
8 import java.time.LocalDate;
9 import java.util.Date;
10 import java.util.List;
11 import java.time.temporal.ChronoUnit;
12 import java.util.Date;
13 import java.time.*;
14
15 import static java.time.temporal.ChronoUnit.DAYS;
16
17 2 usages  ▲ EstefaniaPerez +2
18 @Service
19 public class PaqueteService implements PaqueteServiceI {
20
21     4 usages
22     @Autowired
23     private PaqueteDao paqueteDao;
```



The screenshot shows a Java code editor with the file `PaqueteService.java` open. The code defines several private fields using `@Autowired` annotations:

```
private ClienteService clienteService;
private HotelService hotelService;
private ReservaService reservaService;
private AereosService aereosService;
private EntradaService entradaService;
```

It also contains an `@Override` annotation and a `@Transactional` annotation on a method:

```
@Override
@Transactional(readOnly = true)
public List<Paquete> listarPaquetes() { return (List<Paquete>) paqueteDao.findAll(); }
```

A commit message at the bottom indicates a change by `EstefaniaPerez`:

```
EstefaniaPerez +1
```

The screenshot shows the same `PaqueteService.java` file with additional code. It includes an `@Override` annotation and a `@Transactional` annotation on a method, and a detailed implementation of another method:

```
@Override
@Transactional(readOnly = true)
public List<Paquete> listarPaquetes() { return (List<Paquete>) paqueteDao.findAll(); }

@Override
@Transactional
public void guardar(Cliente cliente, Hotel hotel, int cantPersonas, float seguro,
                    String traslado, List<Entrada> entradaList, List<Habitacion> habitaciones,
                    Date fechaInicio, Date fechaFin, List<Aereos> pasajes) {
    float cotizacion=0;
    float precioEntradas=0;
    float precioHabitaciones=0;
    float precioAereos=0;
    Paquete paquete=new Paquete();
    for(Entrada entrada : entradaList) {
        entrada.setPaquete(paquete);
        entradaService.guardar(entrada);
        precioEntradas+=entrada.getPrecio();
    }

    for(Habitacion habitacion : habitaciones) {
        reservaService.guardar(new Reserva(habitacion,fechaInicio,fechaFin));
        precioHabitaciones+=habitacion.getPrecio();
    }
}
```



```
PaqueteService.java
for(Habitacion habitacion : habitaciones)
{
    reservaService.guardar(new Reserva(habitacion, fechaInicio, fechaFin));
    precioHabitaciones+=habitacion.getPrecio();
}

for(Aeros pasaje : pasajes)
{
    pasaje.setClienteTitular(cliente);
    pasaje.setPaquete(paquete);
    aeroService.guardar(pasaje);
    precioAeros+=pasaje.getPrecio();
}

float interes=0;
LocalDate mundial = LocalDate.ofEpochDay(2022-11-01);
int diasrestantes = (int) ChronoUnit.DAYS.between(LocalDate.now(),mundial);

if (diasrestantes > 50)
{
    interes = 1.1F;
} else if (diasrestantes > 30) {
    interes = 1.3F;
} else {
    interes = 1.5F;
}
```



```
cotizacion=(precioEntradas+precioAereos+precioHabitaciones)*interes;

paquete.setCantidadDePersonas(cantPersonas);
paquete.setClienteTitular(clienteService.encontrarCliente(cliente));
paquete.setEstadiaHotel(hotelService.encontrarHotel(hotel));
paquete.setSeguroMedico(seguro);
paquete.setTrasladoLocal(traslado);
paquete.setCotizacion(cotizacion);
paqueteDao.save(paquete);
}

* EstefaniaPerez
@Override
@Transactional
public void eliminar(Paquete paquete) { paqueteDao.delete(paquete); }

2 usages * EstefaniaPerez
@Override
@Transactional(readOnly = true)
public Paquete encontrarPaquete(Paquete paquete) {
    return paqueteDao.findById(paquete.getID()).orElse( other: null);
}
```

PaqueteService Interfaz

```
 README.md  PaqueteServiceI.java
1 package com.example.FanturAdmin.Service;
2 import com.example.FanturAdmin.Model.*;
3
4 import java.util.Date;
5 import java.util.List;
6
7 1 usage 1 implementation * EstefaniaPerez*
8 ol public interface PaqueteServiceI {
9
10     1 usage 1 implementation * EstefaniaPerez
11     ol List<Paquete> listarPaquetes();
12
13     1 implementation new *
14     ol void guardar(Cliente cliente, Hotel hotel, int cantPersonas, float seguro, String traslado,
15         List<Entrada> entradaList, List<Habitacion> habitaciones, Date fechaInicio, Date fechaFin,
16         List<Aereos> pasajes);
17
18     1 implementation * EstefaniaPerez
19     ol void eliminar (Paquete paquete);
20
21     2 usages 1 implementation * EstefaniaPerez
22     Paquete encontrarPaquete(Paquete paquete);
23 }
```



Bibliografía

<https://rockcontent.com/es/blog/que-es-java/>

<https://www.computerweekly.com/es/definicion/Microsoft-SQL-Server>

<https://www.ibm.com/cloud/learn/java-spring-boot>

<https://www.bezkoder.com/spring-boot-sql-server/>

<https://www.javaguides.net/2019/01/spring-boot-microsoft-sql-server-jpa-hibernate-crud-restfull-api-tutorial.html>

<https://www.bezkoder.com/spring-boot-sql-server/>