DOCUMENTO SITENOW

ISMAEL MOHAMED NAYIB EL ANDALOUSSI

2ªDAM

ÍNDICE

CONTEXTUALIZACIÓN:

SiteNow es una **aplicación para reservar habitaciones de hoteles** en todo el mundo con infinidad de funcionalidades que permiten al usuario reservar una habitación, añadir una crítica, visualizar sus reservas...

SiteNow también cuenta con **funcionalidades para administradores** que permiten ver las críticas de todas las personas, ver el personal activo, registrar las actualizaciones realizadas en las diferentes actualizaciones...

SiteNow es muy similar a las plataformas de reservas de hoteles como **trivago, Booking, Hoteles.com** ...

TECNOLOGÍAS:

¿QUÉ NECESITAMOS?

La aplicación necesitará una **base de datos con múltiples tablas** para las funcionalidades que se quieren implementar.
La aplicación necesitará un **servidor** que se comunique con la base de datos.
La aplicación necesitará una **interfaz gráfica** que ofrezca las funcionalidades y que recopile o muestre la información.

¿QUÉ TECNOLOGÍAS SE USARÁN?

Para la creación de la interfaz se usará la biblioteca gráfica **swing** en java.

Para la **base de datos** se usarán tres tecnologías:

- ➔ **MariaDB**.
  - ◆ Base de datos relacional.
- ➔ **MongoDB**
  - ◆ Base de datos no relacional.
- ➔ Ficheros **XML**
  - ◆ Contenedor con información.

El **servidor**, en función de la acción que se quiera realizar, se comunicará con una base de datos u otra.
En las comunicaciones se usarán:

- ➔ Para mariaDB se usará **hibernate**.
- ➔ Para MongoDB se usará un **driver para la conexión.**
- ➔ Para ficheros XML se usará **SAX**.

Para la comunicación entre la interfaz y el servidor se usará socket (socket cliente y socket servidor) que **mandaran y recibirán objetos Json** serializados (JsonObject) o arrays de objetos (JsonArray) que contendrán la información con la que se va a trabajar.

FASES DE DESARROLLO:

ANÁLISIS:

Se busca programar una aplicación de reservas para hoteles que cuente con

las funcionalidades de:

- **Para Usuarios:**
  - Buscar un hotel.
  - Insertar, modificar, borrar o insertar una reserva.
  - Insertar, modificar, borrar o insertar una reclamación.

- **Para administradores:**
  - Las funcionalidades del usuario.
  - Visualizar todas las reclamaciones.
  - Visualizar el personal
  - Insertar mantenimientos

La aplicación contará con una pantalla para el **inicio de sesión** de los usuarios. Si el usuario no existe tendrá la **opción de registrarse**.

El usuario podrá **acceder como usuario normal o como administrador** en función de su **DNI**.

DISEÑO:

En primer lugar se creará la base de datos junto a sus tablas que deberán estar normalizadas hasta tercera forma normal **(3FN)** en el caso de la base de datos relacional.

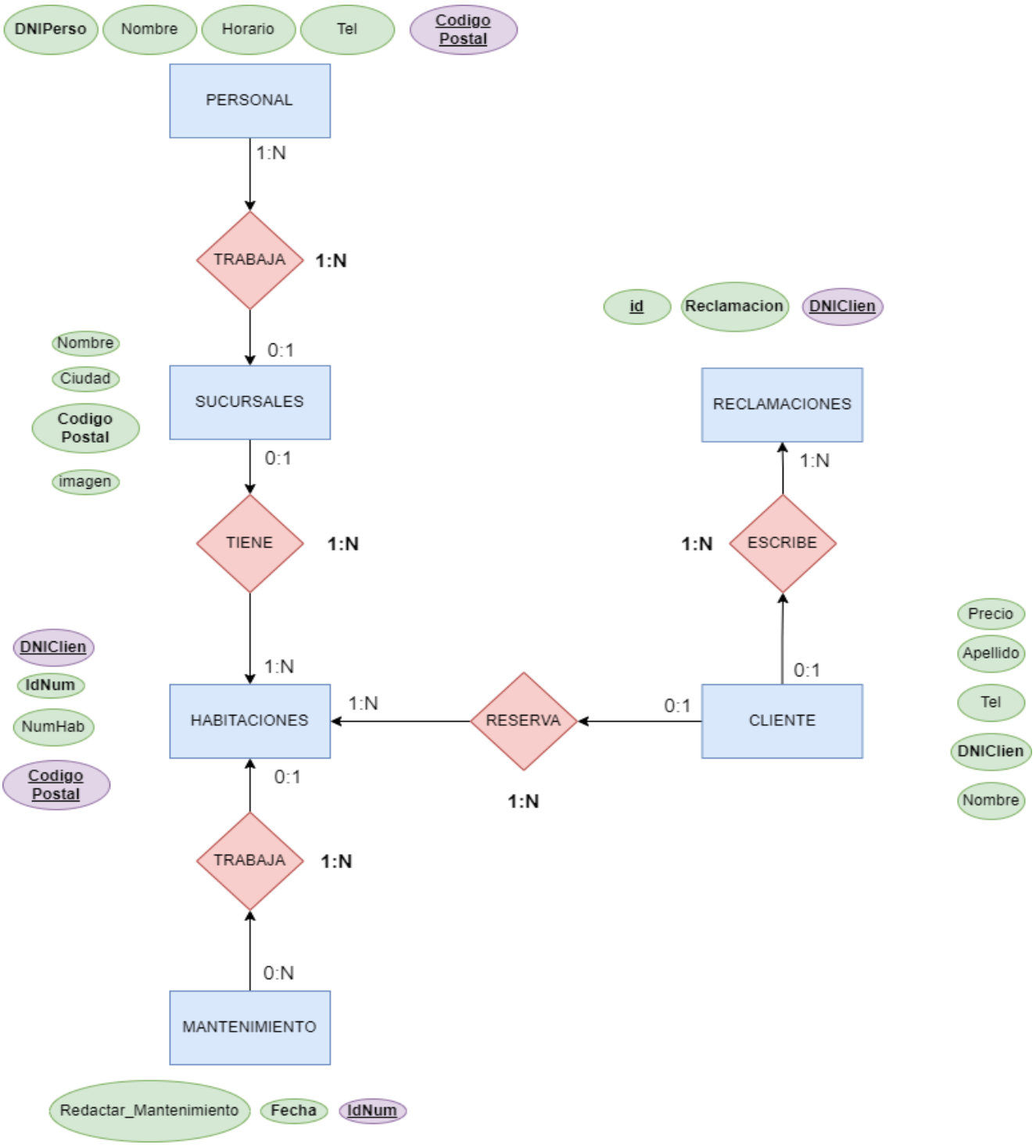
La base de datos relacional cuenta con las siguientes **tablas**:

- **Cliente:**
  - ➢ Son los clientes registrados en el hotel. También contiene a los usuarios administradores.
- **Habitaciones:**
  - ➢ Son las habitaciones reservadas por los clientes.
- **Mantenimiento:**
  - ➢ Almacena los mantenimientos realizados por el personal del hotel a las habitaciones.
- **Personal:**
  - ➢ Es el personal que trabaja en el hotel.
- **Reclamaciones:**
  - ➢ Son las reclamaciones hechas por los clientes
- **Sucursal:**
  - ➢ El la dirección en la que se encuentra el hotel.

Para la **base de datos relacional** se usará el sistema de gestión de bases de datos **MariaDB**.

Las tablas están en **1FN, 2FN y 3FN** y todas las claves primarias y foráneas están en las tablas pertinentes.

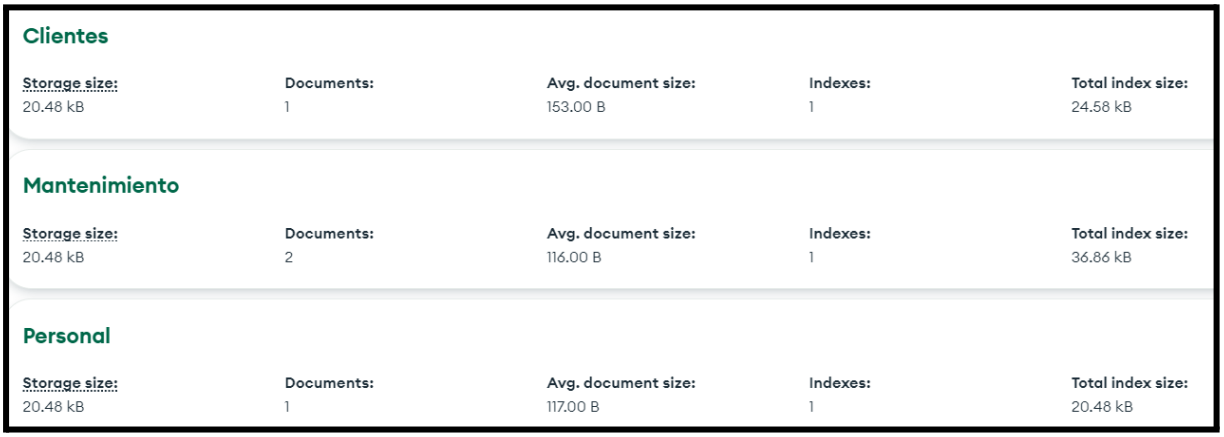
Diagrama E/R con claves primarias y foráneas:
Óvalo verde con letra **negrita** clave primaria.
Óvalo morado: Clave foránea.



La base de datos no relacional contará con tres tablas:

- ❖ **Clientes:**
  - ➢ Contendrá los datos de los clientes, el lugar donde se hospedan y las reclamaciones que hayan hecho.

- **Mantenimiento:**
  - ➢ Contiene los datos sobre el mantenimiento y el id de la habitación.
- **Personal:**
  - ➢ Contiene datos sobre el personal y el código postal de la sucursal en la que trabajan.



Ficheros XML con datos:

También se usaran ficheros **XML** que contendrá al personal del hotel que se usarán para mostrar la información en la interfaz.

Comunicación:

Las **bases de datos se comunicarán solo con el servidor y el servidor se comunicará con la interfaz usando sockets**.

El **servidor** se compone de un **bucle while** infinito que acepta las peticiones

de los clientes que se conecten al servidor usando el **puerto asignado y una ip**.

```
ServerSocket server = new ServerSocket( port: 5555);
Socket socket;
Persistencia p = new Persistencia();

while (true) {
    socket = server.accept();
    DataInputStream dis = new DataInputStream( in: socket.getInputStream());
    DataOutputStream dos = new DataOutputStream( out: socket.getOutputStream());
    ObjectOutputStream oos = new ObjectOutputStream( out: socket.getOutputStream());
    ObjectInputStream ois; //= new ObjectInputStream(socket.getInputStream());
```

El **bucle** también **contiene un switch** con las distintas acciones que se pueden realizar. Dependiendo del frame se llevará a cabo una acción (case) u otra.

Las **comunicaciones** exceptuando la elección de la acción (case) a realizar (readUTF) se realizarán con **objetos Json** que contendrán información.

```
String consulta = dis.readUTF();
System.out.println( x: consulta);
switch (consulta) {

    case ("buscar sucursal"):
```

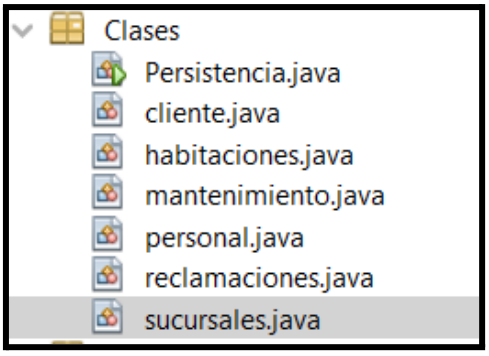
```
oos.writeObject( obj: jso);
```

```
jso = (JsonObject) ois.readObject();
```

¿CÓMO SE REALIZARÁN LAS CONSULTAS A LA BASE DE DATOS?

Para las consultas a la **base de datos relacional mariaDB** se realizarán con la ayuda de **Hibernate**:

Se crea una clase por tabla en la base de datos.



Las clases cuentan variables y características:

- ❖ **Cliente:**
  - ➢ La clase **cliente** cuenta con:
    - "**dni**" como clave primaria.
    - **nombre**
    - **teléfono**
    - **apellido**
    - **precio**

```
@Entity
@Table(name="cliente")
public class cliente {

        @Id
        private String dni;
        private String nombre;
        private String telefono;
        private String apellido;
        private String precio;
```

- ❖ **Habitaciones:**
  - ➢ La clase **habitaciones** cuenta con:
    - "**id**" como clave primaria. Es autoincremental.
    - **dni_cliente** como clave foránea que proviene de la tabla "cliente". Solo necesitamos el dni.

- **“sucursal”** como clave foránea que proviene de la tabla sucursales. Del objeto sucursal solo necesitamos el **código postal**
- **número_habitación**.

```
@Entity
@Table(name="habitaciones")
public class habitaciones {

        @Id
        @GeneratedValue(strategy=GenerationType.IDENTITY)
        private int id;
        private String numero_habitacion;

        @ManyToOne(cascade = CascadeType.ALL)
        @JoinColumn(name="dni_cliente")
        private cliente dni_cliente;

        @ManyToOne(cascade = CascadeType.ALL)
        @JoinColumn(name="codigo_postal")
        private sucursales sucursal;
```

❖ **Mantenimiento:**

➢ La clase **mantenimiento** cuenta con:

- “**fecha**” como clave primaria.
- **“redactar”** donde se almacena lo que se ha hecho.
- “**habitaciones**” como clave foránea que proviene de la tabla habitaciones. Solo se necesita el **id de la habitación**.

```
@Entity
@Table(name="mantenimiento")
public class mantenimiento {

        @Id
        private String fecha;
        private String redactar;


        @ManyToOne(cascade = CascadeType.ALL)
        @JoinColumn(name="id_num")
        private habitaciones habitacion;
```

❖ **Personal:**

➢ La clase **personal** cuenta con:

- “**dni_personal**” que es la clave primaria.
- **nombre**
- **horario**

- **teléfono**
- **sucursal** como clave foránea que proviene de la tabla sucursales. Solo se necesita el **código postal**.

```
@Entity
@Table(name="personal")
public class personal {

        @Id
        private String dni_personal;

        private String nombre;
        private String horario;
        private int telefono;

        @ManyToOne(cascade = CascadeType.ALL)
        @JoinColumn(name="codigo_postal")
        private sucursales sucursal;
```

❖ **Reclamaciones:**

➢ La clase **reclamaciones** cuenta con:

- "**id**" como clave primaria. Es autoincremental.
- **reclamación**
- **cliente** como clave foránea que proviene de la tabla cliente. Solo se necesita el **dni del cliente**.

```
@Entity
@Table(name="reclamaciones")
public class reclamaciones {

        @Id
        @GeneratedValue(strategy=GenerationType.IDENTITY)
        private int id;

        private String reclamacion;


        @ManyToOne(cascade = CascadeType.ALL)
        @JoinColumn(name="dni_cliente")
        private cliente cliente;
```

❖ **Sucursal:**

➢ La clase **sucursal** cuenta con:

- **"codigo_postal"** es la clave primaria.
- **nombre**
- **ciudad**

- "**imagen**" que es una foto del hotel.

```
@Entity
@Table(name="sucursal")
public class sucursales {

        private String nombre;
        private String ciudad;
        String imagen;

        @Id
        private String codigo_postal;
```

IMPORTANTE:

@Entity
@Table (name="table")

- Sirve para referenciar la tabla de la base de datos en la clase.

@Id

- Sirve para indicar cual es el id. Se declara justo encima de la variable.

@GeneratedValue (strategy=GenerationType.IDENTIFY)

- Sirve para indicar que el id es autoincremental.

@ManyToOne(cascade = CascadeType.ALL)

- Sirve para indicar el tipo de relación entre dos o más tablas.
- En este caso es de "muchos a uno" y se tendrá que crear un objeto de la tabla con que se tiene relación.

@JoinColumn(name="dni_cliente")

- Se declara justo encima del objeto de la clase con la que se tiene conexión y referencia a la columna de la tabla en la que se ha declarado. Ej: declaración en tabla cliente → hará referencia a la columna en la tabla cliente.

Las **clases cuentan con getters y setters** para cada variable.
Se le realiza el **CRUD a todas las clases**.

Para la explicación tomaré como ejemplo el **CRUD de la clase personal**:
Antes de empezar cabe destacar que se debe realizar la **conexión con la base de datos**:

```
EntityManagerFactory factory= Persistence.createEntityManagerFactory("hotel");
EntityManager entityManager = factory.createEntityManager();
entityManager.getTransaction().begin();
```

Para indicar el **tipo de operación** se usa:

```
entityManager
```

Que cuenta con los **métodos**:

`persist(Objeto);` → **Para escribir en la base de datos.**
`.merge(p);` → **Para modificar una entrada en la base de datos.**
`remove(p);` → **Para borrar una entrada en la base de datos.**

Para **realizar la acción** en la base de datos se usa:

```
entityManager.getTransaction().commit();
```

Para **buscar** contenido en la base de datos y devolverlo se usa:

`String consulta="SELECT p FROM personal p";` → **es la consulta.**
`Query query = entityManager.createQuery(consulta);` → **para realizar consultas.**
`lista = query.getResultList();` → **para que el resultado sea un "List"**

La conexión se **cierra** con:

```
entityManager.close();
factory.close();
```

Este procedimiento aplica para todos los métodos.

1. **Insertar personal:**
   - ➢ Se crea un objeto "persona" y se llama a sus setters.
   - ➢ A los setters se le pasa por parámetros el contenido pertinente.
   - ➢ Para la clave foránea se usa el método ".find" del objeto "entityManager" y se le pasa por parámetros la clase a la que

pertenece el objeto y la clave primaria para su posterior búsqueda en la tabla a la que pertenece.

```
public static void insertarPersonal(String nombre, String tel, String dni, String horario, String cd) {
        EntityManagerFactory factory= Persistence.createEntityManagerFactory( persistenceUnitName: "hotel");
        EntityManager entityManager = factory.createEntityManager();
        entityManager.getTransaction().begin();


        personal p = new personal();
        p.setDni_personal( dni_personal: dni);
        p.setNombre(nombre);
        p.setHorario(horario);
        p.setTelefono( telefono: Integer.parseInt( s: tel));

        p.setSucursal( sucursal: entityManager.find( entityClass: sucursales.class,  primaryKey: cd));


        entityManager.persist( entity: p);
        entityManager.getTransaction().commit();

        entityManager.close();
        factory.close();
}
```

2. Modificar personal:

➢ Muy similar a la inserción pero con la peculiaridad de que se usa el método “merge” para modificar la entrada con el mismo id introducido.

```
public static void modificarPersonal(String nombre, String tel, String dni, String horario, String cd) {
        EntityManagerFactory factory= Persistence.createEntityManagerFactory( persistenceUnitName: "hotel");
        EntityManager entityManager = factory.createEntityManager();
        entityManager.getTransaction().begin();


        personal p = new personal();
        p.setDni_personal( dni_personal: dni);
        p.setNombre(nombre);
        p.setHorario(horario);
        p.setTelefono( telefono: Integer.parseInt( s: tel));

        p.setSucursal( sucursal: entityManager.find( entityClass: sucursales.class,  primaryKey: cd));


        entityManager.merge( entity: p);
        entityManager.getTransaction().commit();

        entityManager.close();
        factory.close();
}
```

3. Eliminar personal:

➢ Se declara un objeto del mismo tipo que la clase y se usa el método “find” de entitymanager para inicializarlo buscando en la base de datos un objeto con la misma clave.

➢ Al método find se le facilita por parámetros la clase del dato y la

clave primaria.

```
public void eliminarPersonal(String dni) {
        EntityManagerFactory factory= Persistence.createEntityManagerFactory( persistenceUnitName: "hotel");
        EntityManager entityManager = factory.createEntityManager();
        entityManager.getTransaction().begin();


        personal p = entityManager.find( entityClass: personal.class,  primaryKey: dni);
        entityManager.remove( entity: p);
        entityManager.getTransaction().commit();

}
```

4. **Buscar personal:**
   - ➢ En este caso se hace una consulta estándar.
   - ➢ El resultado de la consulta devolverá un listado gracias al método "getResultList" del objeto "query".

```
public ArrayList<personal> mostrarPersonal() {
        EntityManagerFactory factory= Persistence.createEntityManagerFactory( persistenceUnitName: "hotel");
        EntityManager entityManager = factory.createEntityManager();
        entityManager.getTransaction().begin();

        String consulta="SELECT p FROM personal p";
        Query query = entityManager.createQuery( qlString: consulta);
        List<personal> lista = new ArrayList<personal>();
        personal p =null;
        lista = query.getResultList();

        entityManager.getTransaction().commit();
        return (ArrayList<personal>) lista;
}
```

Para las consultas a la **base de datos no relacional mongoDB** se realizarán de la siguiente forma:

La conexión es muy sencilla:

```java
String uri = "mongodb://localhost:27017";  //conexión

// crear un cliente de conexión con MongoDB
MongoClient mongoClient = MongoClients.create( connectionString: uri);

// Crear un objeto que se conecta a una base de datos en concreta
MongoDatabase database = mongoClient.getDatabase( string: "Hotel");
```

En el caso de mongoDB se escribe y lee directamente usando unos métodos sin la necesidad de crear **clases**.

Con mongoDB solo **escribiremos el mantenimiento** de una habitación.
Todo esto se llevará a cabo dentro de un método que se llamará desde la base de datos.

```java
InsertOneResult result = database.getCollection( string: "Mantenimiento").insertOne( td: new Document()
.append( key: "id_habitacion", value: id_hab)
.append( key: "redactar",  value: redactar)
.append( key: "fecha", value: fecha)
.append( key: "cp",  value: cp));

mongoClient.close();
```

Para las **consultas al fichero XML** se usará SAX.
Se creará una clase que heredará de DefaultHandler y se configurará para que extraiga la información del fichero XML.

```xml
<?xml version="1.0" encoding="windows-1252"?>

<Lista>
    <personal dni = "45432312T" >
        <nombre>Manuel</nombre>
        <horario>tarde</horario>
        <telefono>654231234</telefono>
        <cp>51002</cp>
    </personal>
    <personal dni = "46124352R" >
        <nombre>Samir</nombre>
        <horario>diurno</horario>
        <telefono>643123423</telefono>
        <cp>51003</cp>
    </personal>
    <personal dni = "44124356Y" >
        <nombre>Aaron</nombre>
        <horario>tarde</horario>
        <telefono>633124323</telefono>
        <cp>51002</cp>
    </personal>
</Lista>
```

Con la información extraída se creará un objeto "**personal**" haciendo uso de la clase usada anteriormente en hibernate.

```java
public static List<personal> metodoDeSax() throws ParserConfigurationException, IOException {
    try {
        SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();

        SAXParser saxParser = saxParserFactory.newSAXParser();
        MyHandler h = new MyHandler();
        saxParser.parse(new File( pathname: "C://Users//Admin//eclipse-workspace//Proyecto_Hotel//src//mai

        return h.personalLista;
    } catch (SAXException ex) {
        Logger.getLogger( name: ManejadorSAX.class.getName()).log( level: Level.SEVERE, msg: null, thrown: ex);
    }

    return null;
```

XML sólo se usará para mostrar al personal de un hotel.

¿CÓMO FUNCIONA EL SERVIDOR?

Las **bases de datos se comunicarán solo con el servidor y el servidor se**

comunicará con la interfaz usando sockets.

El **servidor** se compone de un **bucle while** infinito que acepta las peticiones de los clientes que se conecten al servidor usando el **puerto asignado y una ip**.

```
ServerSocket server = new ServerSocket( port: 5555);
Socket socket;
Persistencia p = new Persistencia();

while (true) {
    socket = server.accept();
    DataInputStream dis = new DataInputStream( in: socket.getInputStream());
    DataOutputStream dos = new DataOutputStream( out: socket.getOutputStream());
    ObjectOutputStream oos = new ObjectOutputStream( out: socket.getOutputStream());
    ObjectInputStream ois; //= new ObjectInputStream(socket.getInputStream());
```

El bucle también **contiene un switch** con las distintas acciones que se pueden realizar. Dependiendo del frame se llevará a cabo una acción (case) u otra.

```
switch (consulta) {

    case ("buscar_sucursal"):
        dis = new DataInputStream( in: socket.getInputStream());
        System.out.println( x: "Prueba");
        String hotel = dis.readUTF();
        System.out.println( x: hotel);
        sucursales su = p.mostrarSucursal( nombre: hotel);

        jso.put( key: "nombre",  value: su.getNombre());
        jso.put( key: "ciudad",  value: su.getCiudad());
        jso.put( key: "codigo_postal",  value: su.getCodigo_postal());
        jso.put( key: "imagen",  value: su.getImagen());

        oos.writeObject( obj: jso);
        break;

    case ("reservar_habitacion"):
        ois = new ObjectInputStream( in: socket.getInputStream());
        jso = (JsonObject) ois.readObject();
```

Las **comunicaciones** exceptuando la elección de la acción (case) a realizar (readUTF) se realizarán con **objetos Json** que contendrán información.

```
String consulta = dis.readUTF();
System.out.println( x: consulta);
switch (consulta) {

    case ("buscar sucursal");
```

```
oos.writeObject( obj: jso);
```

```
jso = (JsonObject) ois.readObject();
```

Dependiendo del "case" se llamará o a los métodos de hibernate, mongoDB o SAX.

IMPLEMENTACIÓN:

INICIO SESIÓN:

- ➢ Al pulsar el botón de inicio de sesión se crea el socket cliente con el puerto del servidor y ocurre lo siguiente:
  1. Primero se manda el "case" del switch que recibirá el servidor.
  2. Luego se construye el Json con los datos de la persona y se manda al servidor.
  3. El servidor buscará si el cliente se encuentra en la base de datos. Si se encuentra iniciará sesión pero si no se encuentra se creará.

- ➢ **Cliente**:

```
cliente = new Socket( host: "localhost", port: 5555);
//DataInputStream dis = new DataInputStream(cliente.getInputStream());
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());
ObjectOutputStream oos = new ObjectOutputStream( out: cliente.getOutputStream());
//ObjectInputStream ois = new ObjectInputStream(cliente.getInputStream());

    dos.writeUTF( str: "insertar_cliente");
    JsonObject jsoLL = new JsonObject();

    Jsoner.serialize( jsonSerializable: jsoLL);
    jsoLL.put( key: "nombre",  value: nombreT.getText());
    jsoLL.put( key: "apellido",  value: apellidoT.getText());
    jsoLL.put( key: "dni",  value: dniT.getText());
    jsoLL.put( key: "tel",  value: telefonoT.getText());
    oos.writeObject( obj: jsoLL);
```

➢ **Servidor**:

```
case ("insertar_cliente"):
    ois = new ObjectInputStream( in: socket.getInputStream());
    JsonObject jso3 = (JsonObject) ois.readObject();

    String nom = (String) jso3.get( key: "nombre");
    String ape = (String) jso3.get( key: "apellido");
    String dni = (String) jso3.get( key: "dni");
    String tel = (String) jso3.get( key: "tel");

    cliente c = p.mostrarCliente(dni);

    if (c.getDni().length() < 6) {
        p.insertarCliente( nombre: nom,  apellido: ape, dni, tel,  precio: "10");
    } else {
        System.out.println( x: "La persona existe.");
    }
    ois.close();
    break;
```

BUSCAR SUCURSAL:

➢ Al pulsar el botón de buscar se crea el socket cliente con el puerto del servidor y ocurre lo siguiente:

1. Primero se manda el "case" del switch para que se ejecute el código adecuado en el servidor.
2. Luego se manda el nombre del hotel que se quiere buscar.
3. El servidor recibirá el nombre del hotel y lo buscará haciendo uso del método de hibernate "mostrarSucursal(nombre)" que devolverá un objeto de tipo sucursal
4. Con el objeto sucursal se construirá el Json con los datos de la sucursal.
5. El Json se manda desde el servidor y se recibe en el cliente.

6. Una vez recibido el objeto Json se procede darle valor a los TextFields pertinentes.

➢ **Cliente**:

```
cliente = new Socket( host: "localhost", port: 5555);
DataInputStream dis = new DataInputStream( in: cliente.getInputStream());
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());
ObjectInputStream ois = new ObjectInputStream( in: cliente.getInputStream());

dos.writeUTF( str: "buscar_sucursal");
dos.writeUTF( str: busqueda.busqueda.getText());

JsonObject jso = jso=(JsonObject) ois.readObject();
nombreHotel.setText((String) jso.get( key: "nombre"));
ubicacionReal.setText((String) jso.get( key: "ciudad"));
cp.setText((String) jso.get( key: "codigo_postal"));

String ruta=(String)jso.get( key: "imagen");
System.out.println("Contenido ruta: "+ruta);
URL url = new URL( spec: ruta);
    fotoHotel = ImageIO.read( input: url);
    imagen.setIcon(new ImageIcon( image: fotoHotel));
imagen.setVisible( aFlag: true);
```

➢ **Servidor**:

```
case ("buscar_sucursal"):
    dis = new DataInputStream( in: socket.getInputStream());
    System.out.println( x: "Prueba");
    String hotel = dis.readUTF();
    System.out.println( x: hotel);
    sucursales su = p.mostrarSucursal( nombre: hotel);

    jso.put( key: "nombre",  value: su.getNombre());
    jso.put( key: "ciudad",  value: su.getCiudad());
    jso.put( key: "codigo_postal",  value: su.getCodigo_postal());
    jso.put( key: "imagen",  value: su.getImagen());

    oos.writeObject( obj: jso);
    break;
```

RESERVAR HABITACIÓN:

➢ Una vez buscado el hotel aparecerá la información de este junto a un botón para reservar una habitación. Al pulsar este botón se creará el socket y ocurrirá lo siguiente:

1. Primero se manda el "case" al servidor para que se ejecute el código adecuado.
2. Se construye el Json con los datos de la reserva.
3. El código postal enviado en el Json proviene de la búsqueda del hotel.
4. El servidor recibe el Json enviado por el cliente y ejecuta el método de Hibernate "insertarHabitacion()" con los datos enviados.

➢ **Cliente**:

```
Socket cliente = new Socket( host: "localhost",  port: 5555);
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());
ObjectOutputStream oos = new ObjectOutputStream( out: cliente.getOutputStream());

//if(dni_comp_Usu != dniF.getText()){
//    JOptionPane.showMessageDialog(this, "El dni introducido no coincide con el
//}else{
dos.writeUTF( str: "reservar_habitacion");
JsonObject jso = new JsonObject();
//Jsoner.serialize(jso);
jso.put( key: "num_Hab",  value: numHabF.getText());
jso.put( key: "cp",  value: cp);
jso.put( key: "dni",  value: dniF.getText());
oos.writeObject( obj: jso);
```

➢ **Servidor**:

```
case ("reservar_habitacion"):
    ois = new ObjectInputStream( in: socket.getInputStream());
    jso = (JsonObject) ois.readObject();

    p.insertarHabitacion((String) jso.get( key: "num_Hab"), (String) jso.get( key: "dni"), (String) jso.get( key: "cp"));
    ois.close();
    break;
```

MODIFICAR RESERVA:

➢ Al pulsar el botón situado en la barra de tareas "modificar reserva" el programa te llevará a un frame distinto que contará con un formulario para modificar la reserva. Si se pulsa el botón "modificar" se creará un socket cliente y ocurrirá lo siguiente:

1. Primero se manda el "case" al servidor para que se ejecute el código adecuado.
2. Se construye el Json con los datos de la reserva y en función del id de reserva se modificará un cliente u otro.
3. El servidor recibe el Json enviado por el cliente y busca al cliente en el servidor a través de su id de reserva con el método de Hibernate "mostrarHabitacionesCliente(id)"
4. Una vez encontrado el cliente se procede a modificar los datos con el método de Hibernate "modifiicarReserva(datos)".

➢ **Cliente**:

```
Socket cliente = new Socket( host: "localhost",  port: 5555);
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());
ObjectOutputStream oos = new ObjectOutputStream( out: cliente.getOutputStream());

dos.writeUTF( str: "modificar_reserva");

JsonObject jso = new JsonObject();
Jsoner.serialize( jsonSerializable: jso);
jso.put( key: "id",  value: idBuscar.getText());
jso.put( key: "numHab",  value: numHab.getText());
jso.put( key: "dni",  value: dniF.getText());

oos.writeObject( obj: jso);
cliente.close();
```

➢ **Servidor**:

```
case("modificar_reserva"):
    ois = new ObjectInputStream( in: socket.getInputStream());

    JsonObject jsoMR =(JsonObject) ois.readObject();
    int idMR = Integer.parseInt( s: jsoMR.get( key: "id").toString());
    habitaciones h = p.mostrarHabitacionesClienteHab( id: idMR).get( index: 0);

    p.modificarHabitacion( id: idMR, num_hab: jsoMR.get( key: "numHab").toString()
            , dni_cli: jsoMR.get( key: "dni").toString(), cp: h.getSucursal().getCodigo_postal());
    break;
```

ELIMINAR RESERVAS:

➢ Al pulsar el botón situado en la barra de tareas "eliminar reserva" el programa te llevará a un frame distinto que contará con un formulario para eliminar la reserva. Si se pulsa el botón "eliminar" se creará un socket cliente y ocurrirá

lo siguiente:

1. Primero se manda el "case" al servidor para que se ejecute el código adecuado.
2. Se construye el Json con el id de la reserva que se quiere eliminar y se manda al servidor.
3. El servidor recibe el objeto Json y ejecuta el método de Hibernate "eliminarHabitación(id)".

➢ **Cliente**:

```
Socket cliente = new Socket( host: "localhost",  port: 5555);
DataInputStream dis = new DataInputStream( in: cliente.getInputStream());
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());
ObjectOutputStream oos = new ObjectOutputStream( out: cliente.getOutputStream());

dos.writeUTF( str: "borrar_reserva");
JsonObject jso = new JsonObject();
jso.put( key: "id",  value: idReserva.getText());
oos.writeObject( obj: jso);
cliente.close();
```

➢ **Servidor**:

```
case ("borrar_reserva"):
    ois = new ObjectInputStream( in: socket.getInputStream());
    jso = (JsonObject) ois.readObject();
    int id= Integer.parseInt(  s: jso.get( key: "id").toString());
    p.eliminarHabitacion(id);
    break;
```

MOSTRAR MIS RESERVAS:

➢ Al pulsar el botón situado en la barra de tareas "mostrar reserva" el programa te llevará a un frame distinto que contará con un formulario para buscar tus reservas. Si se pulsa el botón "mostrar" se creará un socket cliente y ocurrirá lo siguiente:

1. Primero se manda el "case" al servidor para que se ejecute el código adecuado.
2. Se manda el dni de la persona de la que se quiere ver las reservas.
3. El servidor recibe el dni del cliente y busca en la base de datos las reservas que tiene activas. Se usa el método "mostrarHabitacionesCliente(dni)" que devuelve un listado.
4. Con el listado devuelto se crea un array de Json que se mandará al cliente.

5. El cliente recibe el array de Json y crea las columnas de la tabla.

➢ **Cliente**:

```
Socket cliente = new Socket( host: "localhost",  port: 5555);

DataInputStream dis = new DataInputStream( in: cliente.getInputStream());
ObjectInputStream ois = new ObjectInputStream( in: cliente.getInputStream());
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());

dos.writeUTF( str: "ver_reservas");
dos.writeUTF( str: dniF.getText().toString());
JsonArray jsos = (JsonArray) ois.readObject();
for (int i = 0; i < jsos.size(); i++) {

    JsonObject jso = (JsonObject) jsos.get( index: i);
    System.out.println("Mira aqui: " +jso.get( key: "dni").toString()+"\n"+jso.get( key: "num"));

    String [] info = new String[4];
    info[0]=""+jso.get( key: "id".toString());
    info[1]=jso.get( key: "num").toString();
    info[2]=jso.get( key: "dni").toString();
    info[3]=jso.get( key: "cp").toString();

    modelo.addRow( rowData: info);

}
```

➢ **Servidor**:

```
case ("ver_reservas"):
    String dniR = dis.readUTF();
    ArrayList<habitaciones> reservas = p.mostrarHabitacionesCliente( dni: dniR);
    JsonArray jsos = new JsonArray();
    JsonObject json;
    for (int i = 0; i < reservas.size(); i++) {
        json = new JsonObject();
        json.put( key: "id",  value: reservas.get( index: i).getId());
        json.put( key: "num",  value: reservas.get( index: i).getNumero_habitacion());
        json.put( key: "dni",  value: reservas.get( index: i).getDni_Cliente().getDni());
        json.put( key: "cp",  value: reservas.get( index: i).getSucursal().getCodigo_postal());

        System.out.println("Habitacion: " + reservas.get( index: i).getNumero_habitacion());
        jsos.add( e: json);
    }
    oos.writeObject( obj: jsos);

    break;
```

INSERTAR RECLAMACIÓN:

➢ Al pulsar el botón situado en la barra de tareas "insertar reclamación" el programa te llevará a un frame distinto que contará con un formulario para insertar reclamaciones. Si se pulsa el botón "enviar" se creará un socket cliente y ocurrirá lo siguiente:

1. Primero se manda el "case" al servidor para que se ejecute el código adecuado.
2. Se construye un Json con la reclamación y el dni del cliente.
3. Se manda el Json al servidor.
4. El servidor recibe el json y hace uso del método de Hibernate "insertarReclamacion(datos)" para insertar la reclamación en la base de datos.

➢ **Cliente**:

```
Socket cliente = new Socket( host: "localhost", port: 5555);
DataInputStream dis = new DataInputStream( in: cliente.getInputStream());
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());
ObjectInputStream ois = new ObjectInputStream( in: cliente.getInputStream());
ObjectOutputStream oos = new ObjectOutputStream( out: cliente.getOutputStream());

dos.writeUTF( str: "insertar_reclamacion");
JsonObject jso = new JsonObject();
Jsoner.serialize( jsonSerializable: jso);
jso.put( key: "dni",  value: dni.getText());
jso.put( key: "reclamacion",  value: textReclamacion.getText());

oos.writeObject( obj: jso);
cliente.close();
```

➢ **Servidor**:

```
case ("insertar_reclamacion"):
    ois = new ObjectInputStream( in: socket.getInputStream());
    jso = (JsonObject) ois.readObject();
    p.insertarReclamacion( reclamacion: jso.get( key: "reclamacion").toString(),  dni_cli: jso.get( key: "dni").toString());

    ois.close();
    break;
```

BORRAR RECLAMACIÓN:

➢ Al pulsar el botón situado en la barra de tareas "borrar reclamación" el programa te llevará a un frame distinto que contará con un formulario para borrar reclamaciones. Si se pulsa el botón "borrar" se creará un socket cliente y ocurrirá lo siguiente:

1. Primero se manda el "case" al servidor para que se ejecute el código adecuado.
2. Se crea un Json y se rellena con la información de la reclamación que se quiere eliminar (dni e id).
3. El servidor recibe el Json y busca si la reclamación existe haciendo uso del método de hibernate "buscarRaclamaciónPersona(dni)".
4. En caso de que exista la borrará haciendo uso del método de hibernate "borrarReclamación(id)" y mandará al cliente un mensaje de éxito.
5. Si la reclamación no existe se le mandará al cliente un mensaje de error.

➢ **Cliente**:

```
cliente = new Socket( host: "localhost", port: 5555);
DataInputStream dis = new DataInputStream( in: cliente.getInputStream());
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());
ObjectInputStream ois = new ObjectInputStream( in: cliente.getInputStream());
ObjectOutputStream oos = new ObjectOutputStream( out: cliente.getOutputStream());

dos.writeUTF( str: "borrar_reclamacion");
JsonObject jso = new JsonObject();
Jsoner.serialize( jsonSerializable: jso);
jso.put( key: "dni",  value: dni.getText());
jso.put( key: "id",  value: idReclamacion.getText());

oos.writeObject( obj: jso);

mensaje=dis.readUTF();

cliente.close();
```

➢ **Servidor**:

```
case ("borrar_reclamacion"):
    ois = new ObjectInputStream( in: socket.getInputStream());
    jso = (JsonObject) ois.readObject();
    int eliminar = Integer.parseInt( s: jso.get( key: "id").toString());
    boolean isEliminado = false;
    ArrayList<reclamaciones> misRecla = p.mostrarReclamacionPersona( dni: jso.get( key: "dni").toString());

    for (int i = 0; i < misRecla.size(); i++) {
        if (misRecla.get( index: i).getId() == eliminar) {
            p.eliminarReclamacion( id: eliminar);
            isEliminado = true;
        }
    }

    if (isEliminado == true) {
        dos.writeUTF( str: "Reclamacion eliminada.");
    } else {
        dos.writeUTF( str: "La reclamacion no se encuentra en la base de datos.");
    }
    ois.close();
    break;
```

MOSTRAR RECLAMACIONES:

➢ Al pulsar el botón situado en la barra de tareas "mostrar reclamaciones" el programa te llevará a un frame distinto que mostrará todas las reclamaciones. Si se pulsa el botón "mostrar" se creará un socket cliente y ocurrirá lo siguiente:

1. Primero se manda el "case" al servidor para que se ejecute el código adecuado.
2. El servidor extrae de la base de datos todas las reclamaciones y las manda en un array de Json. Las reclamaciones se extraen haciendo uso del método de Hibernate "mostrarReclamaciones()" que devuelve una lista.
3. El cliente recibe el array de Json y construye las columnas de la tabla con su información.

➢ **Cliente**:

```
Socket cliente = new Socket( host: "localhost",  port: 5555);

DataInputStream dis = new DataInputStream( in: cliente.getInputStream());
ObjectInputStream ois = new ObjectInputStream( in: cliente.getInputStream());
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());

dos.writeUTF( str: "ver_reclamaciones");
JsonArray jsos = (JsonArray) ois.readObject();
for (int i = 0; i < jsos.size(); i++) {
    JsonObject jso = (JsonObject) jsos.get( index: i);

    System.out.println( x: jso.get( key: "dni").toString());
    String [] info = new String[3];
    info[0]=""+jso.get( key: "id".toString());
    info[1]=jso.get( key: "reclamacion").toString();
    info[2]=jso.get( key: "dni").toString();

    modelo.addRow( rowData: info);

}

cliente.close();
```

➢ **Servidor**:

```
case ("ver_reclamaciones"):
    ArrayList<reclamaciones> reclamaciones = p.mostrarReclamacion();
    JsonArray jsos2 = new JsonArray();
    JsonObject json2;
    for (int i = 0; i < reclamaciones.size(); i++) {
        json2 = new JsonObject();
        json2.put( key: "id",  value: reclamaciones.get( index: i).getId());
        json2.put( key: "reclamacion",  value: reclamaciones.get( index: i).getReclamacion());
        json2.put( key: "dni",  value: reclamaciones.get( index: i).getCliente().getDni());

        jsos2.add( e: json2);
    }
    oos.writeObject( obj: jsos2);

    break;
```

INSERTAR MANTENIMIENTO:

➢ Al pulsar el botón situado en la barra de tareas "insertar mantenimiento" el programa te llevará a un frame distinto que mostrará un formulario para insertar un mantenimiento. Si se pulsa el botón "insertar" se creará un socket cliente y ocurrirá lo siguiente:

1. Primero se manda el "case" al servidor para que se ejecute el código adecuado.
2. Se construye el json con la información sobre el mantenimiento y se manda al servidor.
3. En servidor recibe el Json con la información y hace uso del método de MongoDB "insertarMantenimiento(datos)" para insertar la información en la base de datos.

➢ **Cliente**:

```
Socket cliente = new Socket( host: "localhost", port: 5555);
DataInputStream dis = new DataInputStream( in: cliente.getInputStream());
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());
ObjectInputStream ois = new ObjectInputStream( in: cliente.getInputStream());
ObjectOutputStream oos = new ObjectOutputStream( out: cliente.getOutputStream());

dos.writeUTF( str: "insertar_mantenimiento");
JsonObject jso = new JsonObject();
Jsoner.serialize( jsonSerializable: jso);
jso.put( key: "id_hab",  value: id_hab.getText());
jso.put( key: "redactar",  value: txtMantenimiento.getText());
jso.put( key: "fecha",  value: fecha.getText());
jso.put( key: "cp",  value: cp.getText());

oos.writeObject( obj: jso);
cliente.close();

JOptionPane.showMessageDialog( parentComponent: this,  message: "Mantenimiento almacenado.");
id_hab.setText( t: "");
txtMantenimiento.setText( t: "");
fecha.setText( t: "");
cp.setText( t: "");
```

➢ **Servidor**:

```
case ("insertar_mantenimiento"):
    ois = new ObjectInputStream( in: socket.getInputStream());
    JsonObject jsoMan = (JsonObject) ois.readObject();
    Consultas_Mongo mongo = new Consultas_Mongo();
    String idHab = jsoMan.get( key: "id_hab").toString();
    String redactar = jsoMan.get( key: "redactar").toString();
    String fecha = jsoMan.get( key: "fecha").toString();
    String cp = jsoMan.get( key: "cp").toString();
    mongo.insertarMantenimiento( id_hab: idHab,redactar,fecha,cp);

    break;
```

MOSTRAR PERSONAL:

➢ Al pulsar el botón situado en la barra de tareas "insertar mantenimiento" el programa te llevará a un frame distinto que mostrará un formulario para insertar un mantenimiento. Si se pulsa el botón "insertar" se creará un socket cliente y ocurrirá lo siguiente:

1. Primero se manda el "case" al servidor para que se ejecute el código adecuado.
2. El servidor hace uso del método de SAX "metodoDeSax()" para buscar la información en el fichero XML. Este método devuelve una lista.
3. Se construye un array de Json con la lista devuelta.
4. Se manda el array de Json al cliente.
5. El cliente recibe el array de Json y construye las columnas de la tabla con la información.

➢ **Cliente**:

```
Socket cliente = new Socket( host: "localhost",  port: 5555);

DataInputStream dis = new DataInputStream( in: cliente.getInputStream());
ObjectInputStream ois = new ObjectInputStream( in: cliente.getInputStream());
DataOutputStream dos = new DataOutputStream( out: cliente.getOutputStream());

dos.writeUTF( str: "ver_personal_sax");
JsonArray jsos = (JsonArray) ois.readObject();
for (int i = 0; i < jsos.size(); i++) {

    JsonObject jso = (JsonObject) jsos.get( index: i);
    String[] info = new String[5];
    info[0] = "" + jso.get( key: "nombre").toString();
    info[1] = jso.get( key: "dni").toString();
    info[2] = jso.get( key: "horario").toString();
    info[3] = jso.get( key: "tel").toString();
    info[4] = jso.get( key: "cp").toString();

    modelo.addRow( rowData: info);

}

cliente.close();
```

➢ **Servidor**:

```
case ("ver_personal_sax"):
    ManejadorSAX s = new ManejadorSAX();
    List<personal> personal = (ArrayList<personal>) s.metodoDeSax();

    JsonArray jsos3 = new JsonArray();
    JsonObject json3;
    for (int i = 0; i < personal.size(); i++) {
        json3= new JsonObject();
        json3.put( key: "nombre",  value: personal.get( index: i).getNombre());
        json3.put( key: "dni",  value: personal.get( index: i).getDni_personal());
        json3.put( key: "horario",  value: personal.get( index: i).getHorario());
        json3.put( key: "tel",  value: personal.get( index: i).getTelefono());
        json3.put( key: "cp",  value: personal.get( index: i).getSucursal().getCodigo_postal());

        jsos3.add( e: json3);
    }
    oos.writeObject( obj: jsos3);
    break;
```

PRUEBAS:

Iniciar sesión:

Video 3

Buscar Hotel, reservar, mostrar reservas,mostrar mantenimientos y mostrar personal:

Video 1

Modificar reserva, insertar mantenimiento, insertar reclamación, eliminar reserva y borrar reclamación:

Video 2

Los videos estarán disponibles en la entrega del proyecto ya que no se pueden adjuntar al documento por culpa de las restricciones de la cuenta del centro.

CONCLUSIÓN:

El proyecto desarrollado cumple con **todos los requisitos solicitados** por el cliente pero se han implementado tecnologías que realmente no son del todo apropiadas para este tipo de programa.

Los **sockets no están pensados para el tipo de comunicación** que realiza el programa ya que el servidor está constantemente escuchando y tiende a generar **errores** ya que las comunicaciones se sobreponen y no llegan cuando deben.

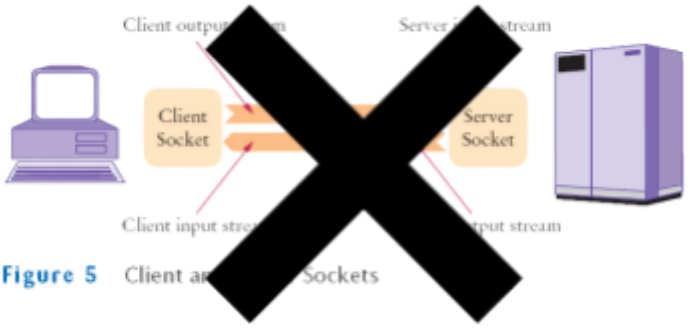


Figure 5 Client ar Sockets

BIBLIOGRAFÍA:

Stack Overflow

Documentación MongoDB

Documentación Hibernate