UNIVERSIDAD AUTONOMA DE B. C. S

DEPARTAMENTO ACADEMICO DE SISTEMAS COMPUTACIONALES (DASC)

-Entregable Q/A

Por

Luis Daniel Ontiveros Lares

Diego Careaga Celis

Axdiael Trinidad Cardenas

David Camacho Olivas

Programación III

Ingeniería en Desarrollo de Software

4to Semestre

Jonathan Soto

La Paz, Baja California Sur a Lunes 02 Junio del 2025

Proyecto MVC - Documento QA de Controladores

Introducción

Este documento QA tiene como objetivo evaluar y documentar las funcionalidades desarrolladas en el sistema, su estructura de código, y registrar tanto los aspectos positivos como las áreas de mejora. Se basa en los principios de calidad del software, usabilidad, coherencia estructural y mantenibilidad del código. Este análisis toma como referencia el estilo y profundidad del documento "QA Tips" (2021) y sigue una línea similar para la documentación técnica detallada.

1. Análisis de arquitectura y funcionalidad

El sistema implementado sigue el patrón de diseño MVC (Modelo - Vista - Controlador). Hasta el momento, se han desarrollado múltiples controladores que vinculan las vistas con la lógica de presentación y basándonos en el diseño previamente realizado.

1.1 Paquetes principales

- **controllers**: Contiene los controladores encargados de la navegación y lógica entre las vistas.
- **views**: Módulo en desarrollo. Contiene las interfaces gráficas de usuario correspondientes.
- **buttonCells**: Contiene componentes gráficos personalizados utilizados en tablas (JTable) para mostrar botones de acción (editar, eliminar, consultar detalles).
- **models**: Contiene clases que se encargan de representar y gestionar los datos del sistema.
- **viewsWB:** Contiene versiones personalizadas de las vistas con estilo mejorado (estética y comportamiento visual más moderno, generalmente usando layouts más consistentes y botones con iconografía) todo gracias al uso de window builder.

2. Controladores evaluados

2.1 AuthController

Responsabilidad: Muestra la vista de inicio de sesión.
Observación positiva: Correcta instancia de AuthView y método login() que reedirecciona directamente en la vista.

2.2 HomeController

Responsabilidad: Carga la vista principal del sistema (pantalla de inicio o home).
Observación positiva: Bien estructurado, con asignación simple de vista y método de invocación home() claro y directo.

2.3 ClientsController

Responsabilidad: Controla el módulo de clientes.
Métodos funcionales:

- Mostrar listado de clientes (clients())
- Crear, editar, consultar, eliminar clientes.
- Mensajes de confirmación y error (successDelete, errorDelete, etc.)

Aspectos positivos:

- La delegación de responsabilidades está bien separada.

2.4 RentalsController

Responsabilidad: Gestiona el flujo de datos y vistas de alquileres del hotel.
Métodos: Crear, editar, consultar, eliminar alquileres.

Aspecto positivo: Métodos intuitivos y acordes a su nombre.

2.5 RoomsController

Responsabilidad: Administrar habitaciones.
Métodos similares a los controladores anteriores.

Observación positiva: Cumple con todos los principios

2.6 RoomTypesController

Responsabilidad: Maneja los tipos de habitaciones.
Observación: Estructura idéntica a la de RoomsController, bien organizada.

2.7 TariffsController

Responsabilidad: Gestión de tarifas.
Problema detectado:

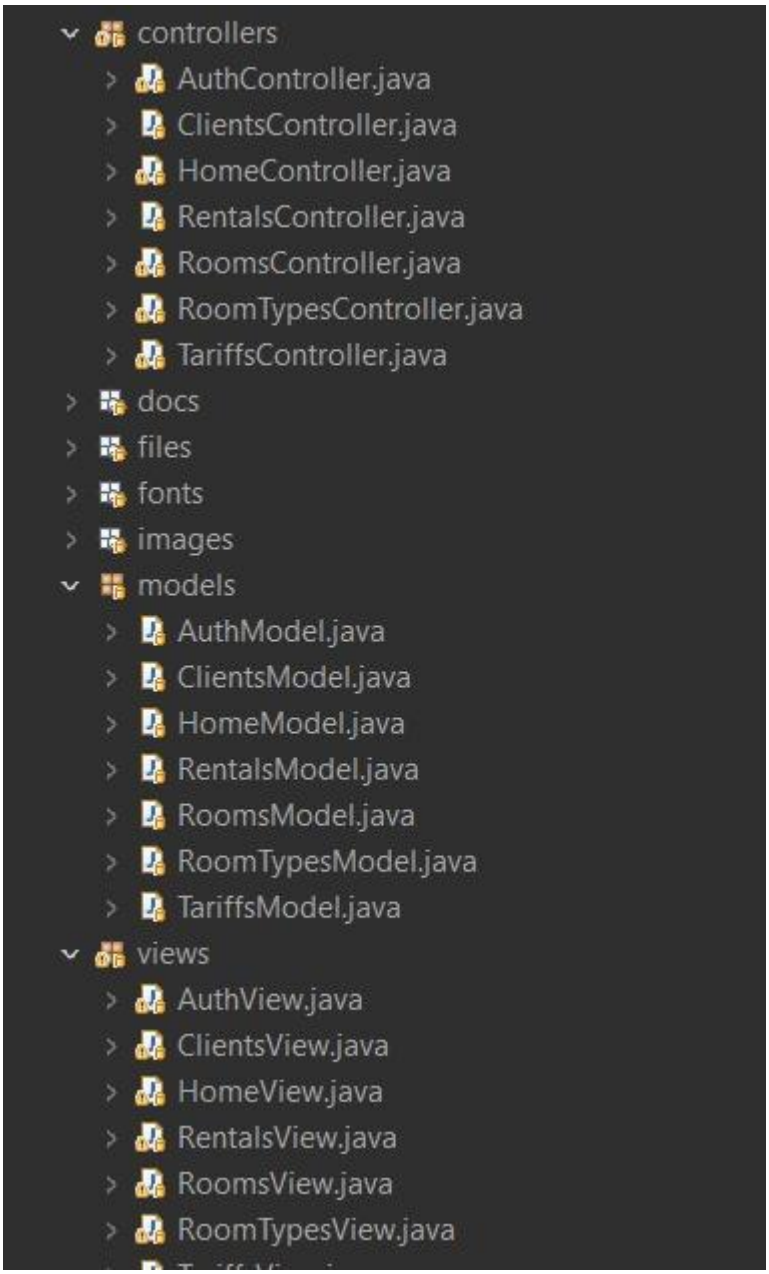
- Los métodos están incorrectamente nombrados: createClient(), editClient() deberían ser createTariff(), editTariff().

Solución propuesta: Renombrar los métodos para reflejar correctamente su propósito.

Encargado: El encargado de resolver el problema será el QA.

```
17    public void tariffs(){
18        view.tariff();
19    }
20
21    public void createClient() {
22        view.createTariff();
23    }
24
25    public void editClient() {
26        view.editTariff();
27    }
28
29    public void consultClient() {
30        view.consultTariff();
31    }
32
33    public void deleteClient() {
34        view.deleteConfirm();
35    }
```

Observación adicional: El término *tariff* no es el más adecuado para el sistema, por petición del profesor se ha solicitado cambiarloo por *rate*. Esto implica renombrar clases, vistas, métodos y variables de TariffsController, TariffsView y cualquier modelo asociado por RatesController, RatesView y Rate.


controllers
AuthController.java
ClientsController.java
HomeController.java
RentalsController.java
RoomsController.java
RoomTypesController.java
TariffsController.java
docs
files
fonts
images
models
AuthModel.java
ClientsModel.java
HomeModel.java
RentalsModel.java
RoomsModel.java
RoomTypesModel.java
TariffsModel.java
views
AuthView.java
ClientsView.java
HomeView.java
RentalsView.java
RoomsView.java
RoomTypesView.java

3. Evaluación del paquete buttonCells

Contiene componentes personalizados para insertar botones funcionales en tablas.

3.1 ActionButton.java

Función: JButton redondeado con efecto visual.
Observaciones:

- Buen uso de Graphics2D.
- Correcta implementación del efecto presionado (mousePressed).

3.2 PanelAction.java

Función: Panel con botones Eliminar, Editar, Consultar.
Observaciones:

- Reutilización clara.
- Repetición excesiva de código para crear y configurar cada botón.

3.3 TableActionCellEditor.java y TableActionCellRender.java

Función: Integración de los botones en celdas de tabla.

- Editor: para edición funcional.
- Render: para visualización en pantalla.

3.4 TableActionEvent.java

Función: Interfaz que define los eventos disponibles en las celdas.

- onEdit(int row), onDelete(int row), onView(int row)

Observación positiva: Bien definida, permite un desacoplamiento claro.

4. Evaluación del paquete models

4.1 AuthModel

Función: Validar credenciales del usuario usando una base de datos remota.
Observaciones:

- El método conectado(String u, String p) compara email y contraseña contra una tabla users.
- Si las credenciales coinciden, carga HomeView.

Aspectos positivos:

- Uso funcional de JDBC y manejo adecuado de conexiones.
- Implementación básica de seguridad con try-catch y finally.

4.2 RoomType

Función: Modelo de datos que representa un tipo de habitación. **Campos:** id, id_tariff, habitaciones incluidas, piso, tipo de habitación.

Aspectos positivos:

- Incluye getters, setters, constructor bien implementado y toString().
- Bien encapsulado.

4.3 RoomTypesModel

Función: Realiza operaciones CRUD sobre room_type en la base de datos.

Funciones:

- createRoomType(): inserta nuevo registro.
- deleteRoomType(): elimina.
- updateRoomType(): actualiza campos.
- getAvailableRoomType(): obtiene todos los tipos de habitación.
- getAvailableTariffs(): obtiene lista de tarifas — deberá cambiar a getAvailableRates().

Aspectos positivos:

- Uso adecuado de PreparedStatement y recuperación de claves generadas.

Mejoras sugeridas:

- Renombrar todos los métodos, variables y consultas de tariff por rate.

4.4 Tariff

Función: Representa a las tarifas de habitación. **Campos:** id, habitación, precio por noche, capacidad, tipo, si es reembolsable.

Aspectos positivos:

- Bien definido con toString(), equals() y hashCode().
- Suficiente para ser usado como entidad visual en comboBoxes u otros selectores.

5. Evaluación del paquete views

5.1 AuthView

Función: Vista de inicio de sesión del sistema.

Características clave:

- Interfaz gráfica con campos para email y contraseña.
- Mensajes visuales de validación si los campos están vacíos.
- Integración con AuthModel para verificación de usuario.

Aspectos positivos:

- Diseño atractivo y visualmente organizado muy apegado al diseño original.
- Uso de colores y fuentes personalizados coherentes con la identidad del sistema.
- Implementación de íconos y escalado de imágenes.

5.2 ClientsView

Función: Vista del módulo de clientes: incluye listados, creación, edición, consulta y eliminación de clientes, así como exportación de historial.

Pantallas incluidas:

- Lista de clientes con acciones (editar, eliminar, consultar).
- Formulario de creación de nuevo cliente.
- Formulario para edición de cliente existente.
- Vista detallada del cliente y su historial.
- Confirmación de eliminación.
- Ventanas de confirmación (éxito o error) y descarga de PDF.

Aspectos positivos:

- Uso de JTable con renderizadores y editores personalizados (buttonCells) para acciones directas.
- Buen manejo de diseño visual con JPanel, Color.decode() y tipografía consistente.
- Modularidad de botones (btnHome, btnCreate, btnCancel) bien implementada.

- Uso de controladores para redirigir acciones como client.createClient() o client.deleteClient().
- Existencia de múltiples vistas por caso de uso (crear, editar, consultar, confirmar eliminación, éxito o error).

Observaciones:

- La tabla actualmente utiliza datos de prueba, lo cual es adecuado para prototipos, pero deberá modificarse para tener datos reales.
- Se implementa lógica de navegación desde botones hacia controladores (ClientsController, HomeController), lo cual es correcto.

Mejoras sugeridas:

- El campo de fecha está tratado como JTextField, se recomienda uso de JDatePicker o componentes equivalentes para mejor UX.
- Los valores de prueba deben ser reemplazados por datos dinámicos.

Código comentado:

- Comentarios // TODO o código no funcional deben eliminarse o implementarse antes de la entrega final.

5.3 HomeView

Función: Vista principal del sistema tras el inicio de sesión y sirve como punto de partida hacia los distintos módulos del sistema.

Acciones disponibles:

- Ir a tipos de habitación (RoomTypesController)
- Ir a habitaciones (RoomsController)
- Ir a rentas (RentalsController)
- Ir a clientes (ClientsController)
- Ir a tarifas (TariffsController, pronto a renombrarse como RatesController)
- Cerrar sesión (AuthController)

Aspectos positivos:

- Buena organización visual con paneles diferenciados para cada opción.
- Uso adecuado de JButton con íconos escalados para mejorar la estética de la interfaz.
- Lógica de navegación correctamente hecho.

- Botón de cerrar sesión correctamente redirige a login().

Observaciones técnicas:

- Se reutiliza correctamente AuthView para cargar íconos.

Mejoras sugeridas:.

- Considerar uso de CardLayout en lugar de múltiples ventanas JFrame para optimizar navegación.

5.4 RentalsView

Función: Vista del módulo de rentas. Permite gestionar operaciones como crear, editar, consultar, eliminar y descargar información de las rentas realizadas.

Pantallas incluidas:

- Vista principal con botones de acción (crear, editar, eliminar, consultar).
- Formularios para creación y edición de rentas.
- Consulta de detalles y exportación a PDF.
- Confirmación y resultado de eliminación (éxito o error).

Aspectos positivos:

- Estructura coherente con las otras vistas del sistema.
- Navegación controlada mediante botones que invocan a métodos en RentalsController.
- Separación clara entre cada funcionalidad (crear, editar, consultar, eliminar).
- Uso correcto del patrón MVC: la vista no contiene lógica de negocio.

Observaciones técnicas:

- El botón Descargar .pdf ejecuta directamente successDownload() desde el controlador, sin verificar existencia o éxito real de la operación.

Mejoras sugeridas:

- Reemplazar JFrame individuales por un contenedor central (ej. CardLayout) si se busca mejorar el flujo general de ventanas.

5.5 RoomsView

Función: Esta vista permite la gestión de habitaciones. Presenta botones para crear, editar, consultar, eliminar y regresar al menú principal.

Pantallas incluidas:

- Vista principal de habitaciones con botones de navegación.
- Formularios para crear, editar y consultar habitaciones.
- Ventanas para confirmación de eliminación, éxito y error.

Aspectos positivos:

- Seguimiento consistente del patrón de vistas usado en el sistema.
- Todas las operaciones principales están disponibles como botones independientes.
- Colores y diseño consistentes con el resto del sistema.

Observaciones técnicas:

- La variable functions se inicializa correctamente en el constructor RoomsView().

Mejoras sugeridas:

- Agregar campos interactivos en createRoom, editRoom y consultRoom.
- Implementar validación de datos y mensajes de error claros.

Errores menores detectados:

- Se utiliza "exito" en lugar de "éxito" en algunos mensajes (falta tilde).

Solución propuesta: Reescribir la palabra de manera correcta.

Encargado: El encargado de resolver el problema será el QA.

```
381         JLabel title = new JLabel("Habitación eliminada con exito");
382         title.setBounds(50,100,600,70);
383         title.setFont(new Font("Inter_18pt Bold", Font.PLAIN, 32));
384         title.setVisible(true);
385         panel.add(title);
386
```

5.6 RoomTypesView

Función: Vista para la gestión de los tipos de habitaciones disponibles en el sistema. Permite visualizar, crear, editar, eliminar y consultar tipos de habitaciones, así como asociar imágenes, tarifas y otras propiedades relacionadas.

Pantallas incluidas:

- Vista principal con galería de tarjetas (cards) por tipo de habitación.

- Formulario para creación de nuevo tipo de habitación.
- Integración visual con imágenes cargadas desde archivos locales.

Aspectos positivos:

- Presentación gráfica mejorada con paneles para cada tipo de habitación incluyendo imagen, nombre y precio por noche.
- La funcionalidad para carga de imágenes se encuentra implementada con JFileChooser, mostrando vista previa inmediata.
- Se hace uso de JComboBox para selección de tarifa y capacidad, lo que reduce errores de entrada del usuario.
- Interfaz visual personalizada y más moderna que otras vistas (uso de GridLayout, bordes, colores consistentes).
- Botones de acción (Eliminar, Editar, Detalles) bien distribuidos dentro de cada card de habitación.

Observaciones técnicas:

- Parte del código comentado (vieja versión de interfaz) podría eliminarse para claridad.

Mejoras sugeridas:

- Mostrar mensaje de confirmación antes de eliminar un tipo de habitación.

5.7 TariffsView

Función: Vista dedicada a la gestión de tarifas del hotel. Permite visualizar, crear, editar, consultar y eliminar tarifas, incluyendo navegación y retroalimentación visual.

Pantallas incluidas:

- Vista principal (tariff) con botones de acción.
- Formulario de creación (createTariff).
- Formulario de edición (editTariff).
- Vista de detalles (consultTariff).
- Diálogos de confirmación y mensajes (deleteConfirm, successDelete, errorDelete, succesDownload).

Aspectos positivos:

- Cada función se encuentra organizada en su propio método (crear, editar, consultar, eliminar).
- Estética coherente con el resto del sistema (Inter_18pt, paleta de colores uniforme).
- Navegación clara y bien conectada con TariffsController.
- Retroalimentación visual con mensajes claros de éxito o error.

Observaciones técnicas:

- Texto "Tarifa eliminado con exito" tiene errores de género y ortografía (debe ser "eliminada" y "éxito").

  Solución propuesta: Reescribir la palabra de manera correcta.
  Encargado: El encargado de resolver el problema será el QA.

```
367         JLabel title = new JLabel("Tarifa eliminado con exito");
368         title.setBounds(50,100,600,70);
369         title.setFont(new Font("Inter_18pt Bold", Font.PLAIN, 32));
370         title.setVisible(true);
371         panel.add(title);
372
373     }
```

5.8 ClientsAddWB (paquete viewsWB)

Función: Vista moderna para registrar nuevos clientes, incluye campos de entrada y carga de imagen.

Pantallas incluidas:

- Formulario de captura de datos de un nuevo cliente: nombre, teléfono, correo electrónico, fecha de nacimiento, imagen.

Aspectos positivos:

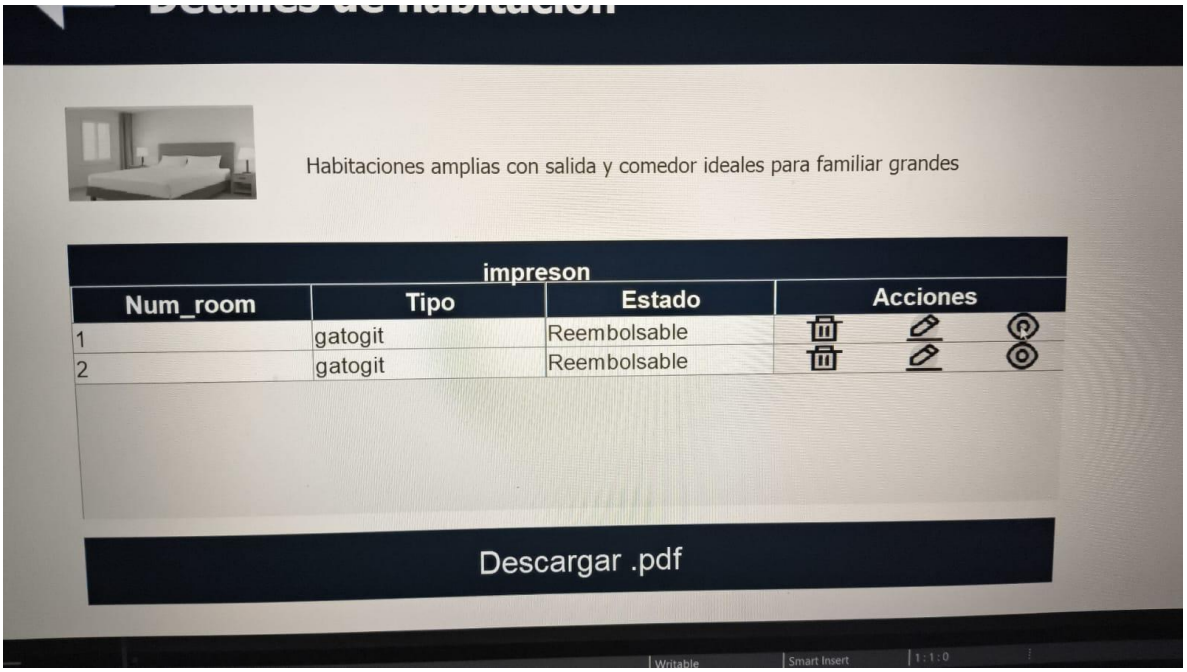
- Diseño visual moderno, uso de JPanel, JLabel y JButton con estilo unificado.
- Carga de imagen visual.

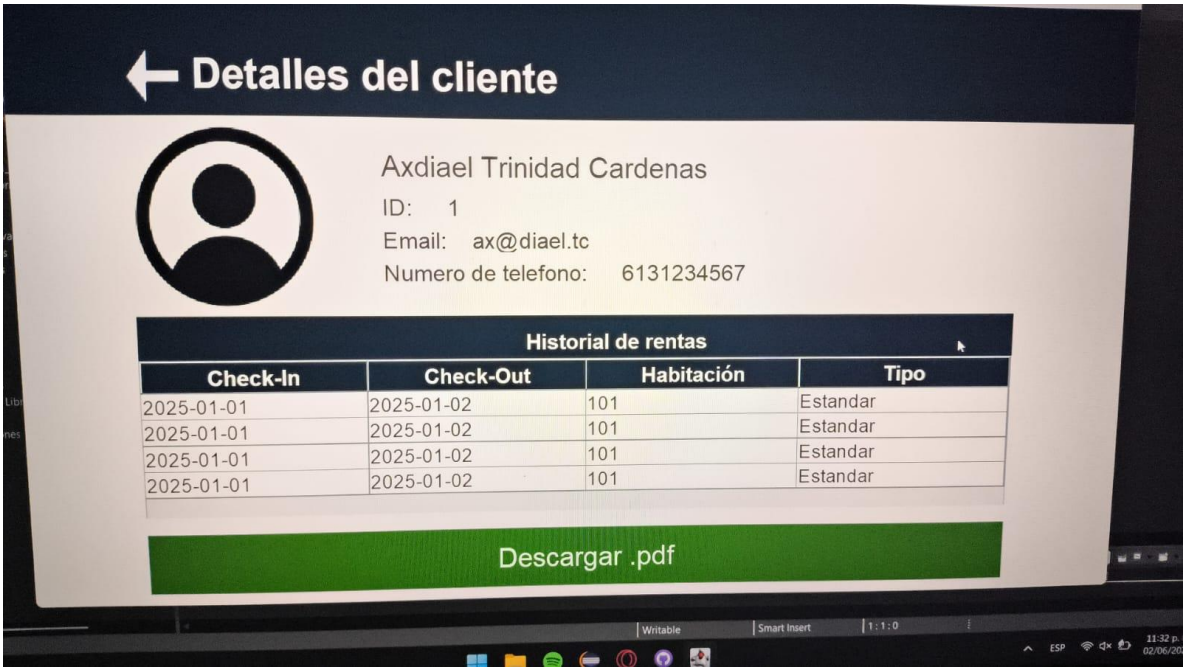
Observaciones técnicas:

- El botón “Guardar” no redirige automáticamente a la vista de clientes.
- Ambos botones (Guardar y Cancelar) son demasiado grandes y deben adaptarse al tamaño de los campos.

Mejoras sugeridas:

- Añadir acción post-guardar para regresar a ClientsWB.
- Reducir tamaño y mejorar alineación de botones de acción.





5.9 ClientsDetailsWB (paquete viewsWB)

Función: Vista que muestra los detalles de un cliente específico, con historial simulado de rentas.

Pantallas incluidas:

- Detalle del cliente (nombre, ID, email, teléfono).
- Tabla con historial de rentas usando JTable.
- Botón para descargar PDF.

Aspectos positivos:

- Tipografía clara y estética consistente.
- La tabla tiene columnas definidas con JTable y formato personalizado.
- Imagen de usuario incluida.

5.10 ClientsWB (paquete viewsWB)

Función: Vista principal de clientes y presenta una tabla de clientes con columnas (ID, nombre, email, teléfono, acciones) y botones para CRUD.

Aspectos positivos:

- Implementa correctamente una tabla interactiva(TableActionCellRender y TableActionCellEditor).
- Botones visuales (crear, editar, consultar, eliminar) bien posicionados.
- Buen uso de separación visual con encabezado fijo (JPanel) y componente principal contenido en scrollPane.

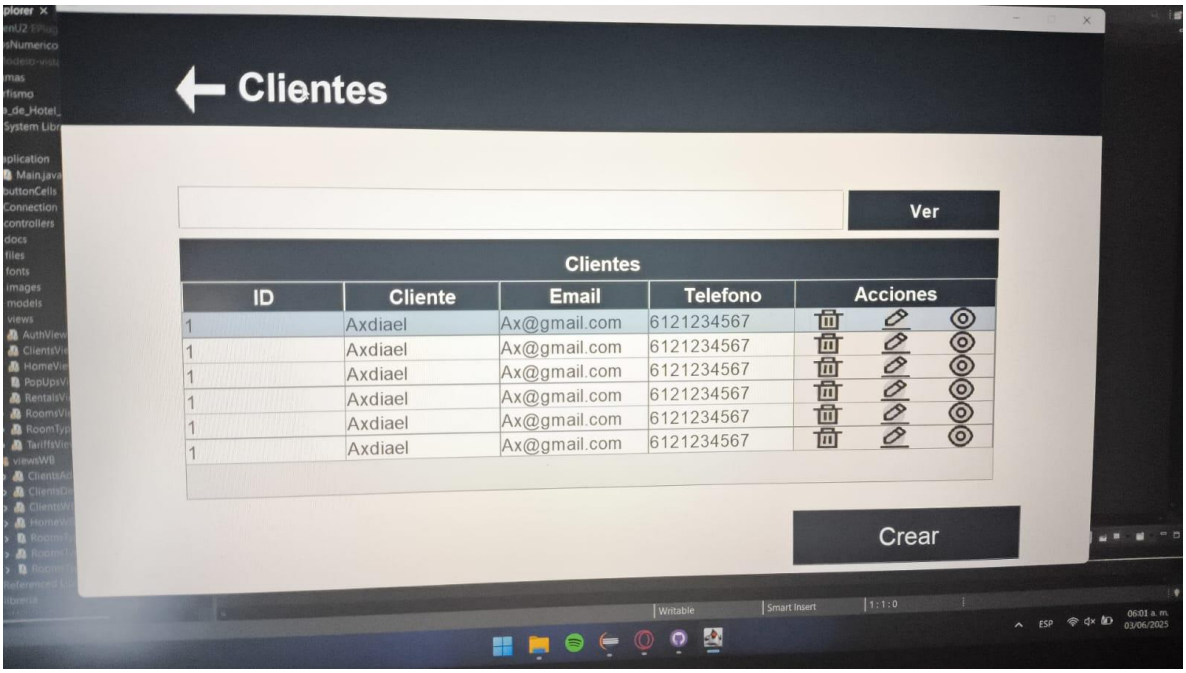
Observaciones técnicas:

- El botón “Crear” no está alineado con el resto de elementos.
- Botón “Detalles” abre información incorrecta (muestra detalles de cliente en otras secciones como habitaciones).
- El botón “Editar” también abre pantalla incorrecta.
- Eliminar muestra un pop-up con texto mal formateado y botones del mismo color del fondo.

Mejoras sugeridas:

- Corregir referencias del controlador en los botones de acción.
- Ajustar alineaciones.
- Corregir lógica de redirección para Editar y Detalles.

Encargado de solucionar: Los errores serán redirigidos al backend y al Fronted



5.11 HomeWB (paquete viewsWB)

Función: Vista de inicio para navegación entre módulos (Clientes, Rentas, Habitaciones, Tipos Hab., Tarifas).

Aspectos positivos:

- Excelente diseño visual, con botones grandes, íconos personalizados y una interfaz intuitiva muy apegada la diseño original.
- Navegación funcional a través de controladores específicos.
- Botón de cerrar sesión implementado correctamente.

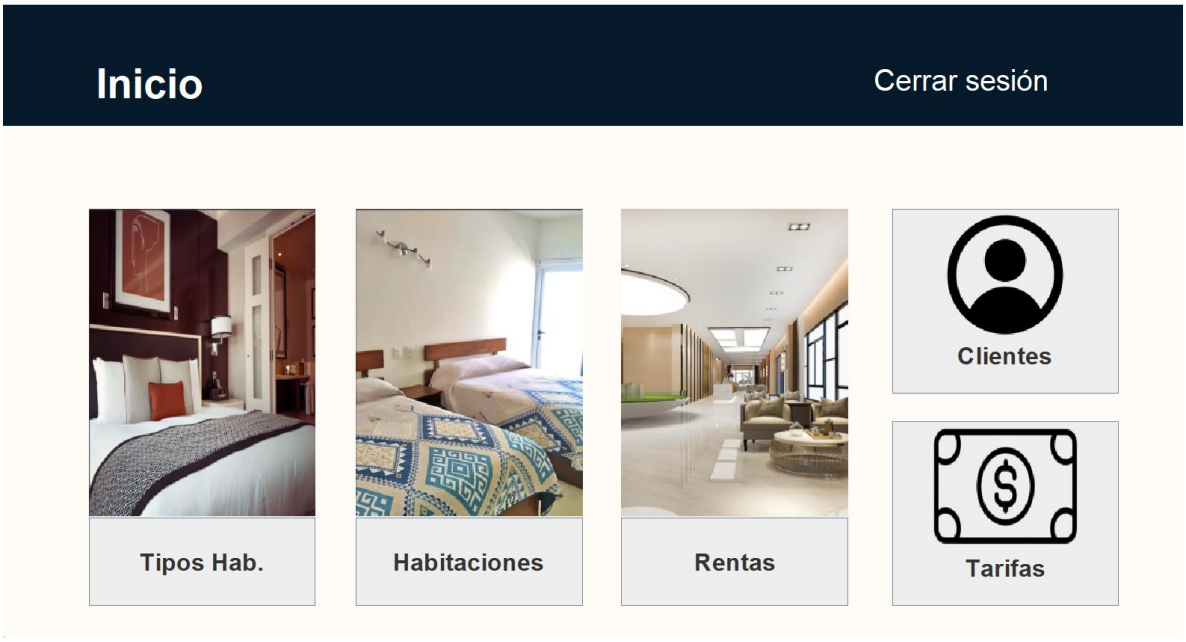
Observaciones técnicas:

- El botón “Cerrar sesión” no está alineado correctamente con el encabezado.

Mejoras sugeridas:

- Alinear correctamente el botón en el panel superior derecho.

Encargado de solucionar: Los errores serán redirigidos al Fronted



5.12 RoomsTypeAddWB (paquete viewsWB)

Función: Vista moderna para registrar un nuevo tipo de habitación.

Aspectos positivos:

- Interfaz clara y moderna, consistente visualmente con el resto del sistema.
- Uso correcto de JComboBox para piso y capacidad.
- Distribución visual adecuada, componentes bien espaciados.

Observaciones técnicas:

- No hay conexión con el controlador ni acciones implementadas en los botones.

Mejoras sugeridas:

- Implementar conexión al controlador (RoomTypesController) en el botón Aceptar.

5.13 RoomsTypeDetailsWB (paquete viewsWB)

Función: Vista para mostrar los detalles de un tipo de habitación.

Aspectos positivos:

- Encabezado visual estilizado con ícono de navegación.
- Tabla de habitaciones detalladas con formato visual profesional.
- Sección textual para descripción o mensaje explicativo.

Observaciones técnicas:

- La descripción de la habitación está fija y no editable.
- Botón inferior btnNewButton_1 no tiene acción ni etiqueta visible.
- El título de la tabla no es claro ni bien posicionado.

Mejoras sugeridas:

- Reemplazar descripción fija por una editable.
- Corregir el título de la tabla y ubicarlo correctamente.

Encargado de solucionar: Los errores serán redirigidos al Fronted

5.14 RoomsTypeWB (paquete viewsWB)

Función: Vista principal para poner en lista todos los tipos de habitación mediante tarjetas (cards).

Aspectos positivos:

- Excelente uso de GridLayout para mostrar habitaciones en formato de tarjetas.
- Componentes bien organizados: imagen, nombre, precio, y acciones por tipo de habitación.
- Botones diferenciados por color para Eliminar, Editar y Detalles.

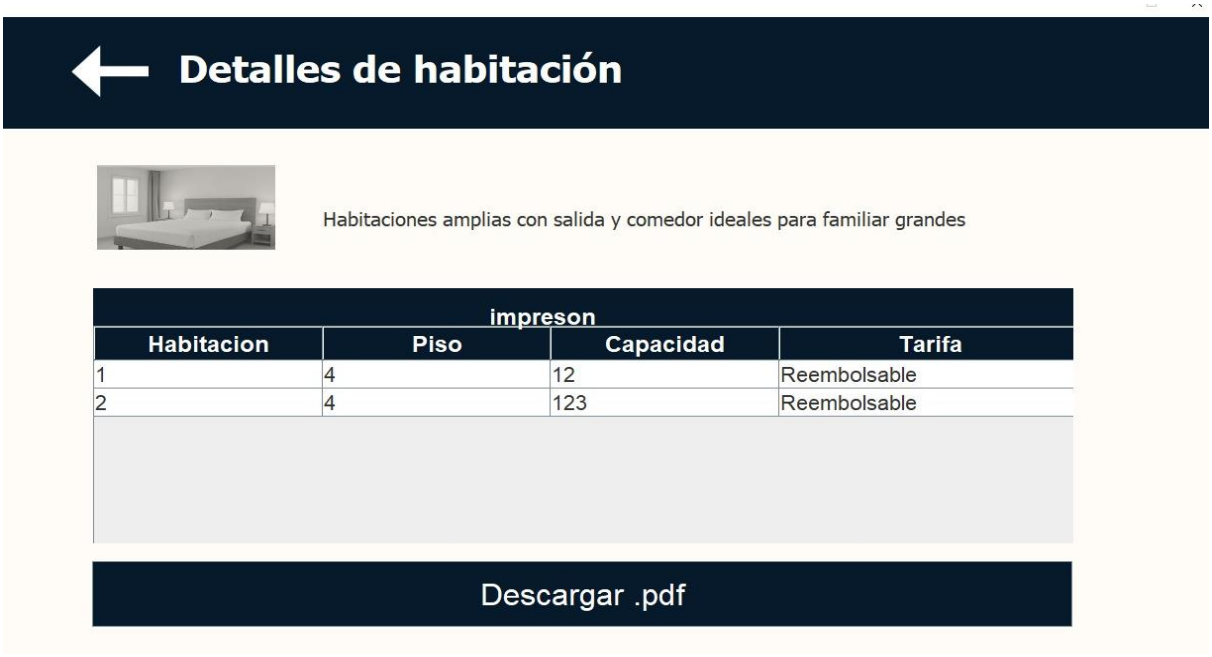
Observaciones técnicas:

- Botones no están conectados a controladores ni lógica de navegación.
- Texto del encabezado mal posicionado.
- Exceso de espacio lateral derecho comparado con el izquierdo.

Mejoras sugeridas:

- Conectar de manera correcta los botones a los controladores.
- Corregir alineación y márgenes de los componentes para evitar desequilibrio visual.

Encargado de solucionar: Los errores serán redirigidos al Fronted



6. Registro de errores de versión inicial

6.1 Conexión a base de datos en AuthModel

Descripción: La implementación inicial de la conexión a la base de datos no seguía lo aprendido en clase.

Clase afectada: AuthModel (versión inicial del proyecto).

Síntomas: Aunque la conexión funcionaba y permitía recuperar datos de autenticación, el diseño era frágil y no era del agrado del profesor

Solución aplicada:

- Se rediseñó completamente el esquema de conexión.
- Se respetó el principio de separación de responsabilidades.

Resultado actual: Conexión estable y se acopla a lo que el profesor nos pidio.

```
Jedis jedis = new Jedis("splendid-hound-18755.upstash.io", 6379, true);
jedis.auth("AUlDAAIjcDEzNTBiYmZkMTg2OTY0N2YyYmE4NDVhY2VjM2NhMjk0N3AxMA");

List<String> listaRedis = jedis.lrange("cuenta", 0, 3);

  if (!jedis.isConnected()) {
    System.err.println("ERROR: No se pudo conectar con la base de datos");
    return false;
}


try{

   for (String linea : listaRedis) {

        System.out.println(linea);
        String[] datos = linea.split(",");
        String pass = "";
        String user = "";

        if (datos.length < 3) {
        user = datos[0].trim();
        pass = datos[1].trim();
        }
        if (user.equals(u) && pass.equals(p)) {
            HomeView entrada = new HomeView();
            entrada.home();
            return true;
```