

**Tema**

EXAMENU1

AEROLÍNEA\_SOAP\_JAVA

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU. CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

02/06/2025

**Tabla de contenido**

[EXAMEN UNIDAD 1 AEROLÍNEA\_SOAP\_JAVA 6](#_Toc199755576)

[INTRODUCCIÓN 6](#_Toc199755577)

[OBJETIVO 6](#_Toc199755578)

[DESARROLLO 7](#_Toc199755579)

[CREACIÓN MODELADO BASE DE DATOS 7](#_Toc199755580)

[INTEGRACIÓN CON MYSQL 8](#_Toc199755581)

[APLICACIÓN SERVIDOR JAVA SOAP 9](#_Toc199755582)

[CLIENTE ESCRITORIO JAVA SOAP 47](#_Toc199755583)

[Creación del cliente para escritorio 47](#_Toc199755584)

[CLIENTE WEB JAVA SOAP 64](#_Toc199755585)

[CLIENTE CONSOLA JAVA SOAP 76](#_Toc199755586)

[CLIENTE MÓVIL JAVA SOAP 80](#_Toc199755587)

[CONCLUSIONES 86](#_Toc199755588)

[RECOMENDACIONES 86](#_Toc199755589)

[BIBLIOGRAFÍA 87](#_Toc199755590)

**ÍNDICE DE IMÁGENES**

[Figura 1. Modelo Conceptual del servicio web 7](#_Toc199756302)

[Figura 2. Modelo Lógico con tablas, claves primarias y relaciones entre entidades 8](#_Toc199756303)

[Figura 3. Modelo Físico implementado en MySQL para la base de datos vuelos\_db 8](#_Toc199756304)

[Figura 4. Creación de la base de datos vuelos\_db en phpMyAdmin del gestor XAMPP a partir del modelo físico 9](#_Toc199756305)

[Figura 5. Creación nuevo proyecto para el servidor en Netbeans 9](#_Toc199756306)

[Figura 6. Selección de tipo de proyecto en Apache NetBeans 9](#_Toc199756307)

[Figura 7. Configuración del nombre y ubicación del proyecto 10](#_Toc199756308)

[Figura 8. Configuración del servidor y versión de Java 10](#_Toc199756309)

[Figura 9. Creación de paquetes para la estructura MVC 11](#_Toc199756310)

[Figura 10. Creación del Web Service en el paquete controlador 11](#_Toc199756311)

[Figura 11. Configuración del nombre y paquete del Web Service WSAerolineasCondor 12](#_Toc199756312)

[Figura 12. Creación de la clase ConversionService en el paquete servicio 15](#_Toc199756313)

[Figura 13. Creación del Paquete ec.edu.monster.modelo 35](#_Toc199756314)

[Figura 14. Creación del Paquete ec.edu.monster.pruebas y Pruebas para ViajecitosService 43](#_Toc199756315)

[Figura 15. Ejecución del proyecto y obtención de la URL del WSDL 44](#_Toc199756316)

[Figura 16. Confoguración del servicio web en SOAP UI 44](#_Toc199756317)

[Figura 17. Ejecución de la operación buscarVuelos en SOAP UI y validación de la respuesta del servicio 45](#_Toc199756318)

[Figura 18. Prueba de la operación iniciarSesion en SOAP UI con respuesta exitosa del servicio 45](#_Toc199756319)

[Figura 19. Ejecución de la operación obtenerComprasCliente en SOAP UI para consultar el historial de compras del cliente 45](#_Toc199756320)

[Figura 20. Prueba de la operación obtenerVueloMasCaro en SOAP UI y visualización de la respuesta del vuelo con mayor costo 46](#_Toc199756321)

[Figura 21. Ejecución de la operación registrarCliente en SOAP UI 46](#_Toc199756322)

[Figura 22. Prueba de la operación registrarCompra en SOAP UI con confirmación de la transacción realizada 47](#_Toc199756323)

[Figura 23. Ejecución de la operación registrarUsuario en SOAP UI con envío de credenciales para el registro 47](#_Toc199756324)

[Figura 24. Creación de un nuevo proyecto para el cliente desktop como Java Application en NetBeans 48](#_Toc199756325)

[Figura 25. Configuración del nombre y creación de paquetes para la estructura MVC del Cliente Escritorio 48](#_Toc199756326)

[Figura 26. Creación del Web Service Client con la URL del WSDL 49](#_Toc199756327)

[Figura 27. Creación y diseño de la interfaz LoginVista 61](#_Toc199756328)

[Figura 28. Diseño de la interfaz “MainFrame” con estilo pastel celeste y opciones principales para el Cliente Escritorio 62](#_Toc199756329)

[Figura 29. Creación del formulario “LoginFrame” en NetBeans para la interfaz de inicio de sesión 62](#_Toc199756330)

[La interfaz de “LoginFrame” se diseña con un estilo limpio y sencillo, adecuado para facilitar el proceso de inicio de sesión del usuario. 62](#_Toc199756331)

[Figura 30. Diseño de la interfaz “LoginFrame” para el inicio de sesión en el Cliente Escritorio 62](#_Toc199756332)

[Figura 31. Diseño de la interfaz “SearchFlightsFrame” para la búsqueda del vuelo más caro 63](#_Toc199756333)

[Figura 32. Diseño de la interfaz “RegisterFrame” para el registro de nuevos usuarios 63](#_Toc199756334)

[Figura 33. Diseño de la interfaz “FlightResultFrame” para visualizar resultados y comprar vuelos 64](#_Toc199756335)

[Figura 34. Diseño de la interfaz “PurchasesFrame” para la visualización del historial de compras 64](#_Toc199756336)

[Figura 35. Creación de un nuevo proyecto Web Application con Maven en NetBeans 65](#_Toc199756337)

[Figura 36. Configuración del nombre y Group ID del proyecto Cliente Web 65](#_Toc199756338)

[Figura 37. Creación de paquetes para la estructura MVC y el paquete ws en el Cliente Web 66](#_Toc199756339)

[Figura 38. Creación e implementación de métodos en ViajecitosService 72](#_Toc199756340)

[Figura 39. Creación de AppControlador en el paquete controlador 72](#_Toc199756341)

[Figura 40. Creación del servlet ComprasClienteServlet para consultar compras del cliente mediante el servicio web 73](#_Toc199756342)

[Figura 41. Creación del servlet LoginServlet para el manejo del inicio de sesión en el Cliente Web 73](#_Toc199756343)

[Figura 42. Creación del servlet RegisterServlet para el registro de nuevos usuarios desde el Cliente Web 73](#_Toc199756344)

[Figura 43. Creación del servlet VueloServlet para gestionar la búsqueda de vuelos en el Cliente Web 74](#_Toc199756345)

[Figura 44. Diseño de la vista “index.jsp” con sección hero para la búsqueda de vuelos en el Cliente Web 74](#_Toc199756346)

[Figura 45. Diseño del formulario de inicio de sesión en la vista “login.jsp” del Cliente Web 75](#_Toc199756347)

[Figura 46. Formulario de registro de usuario en la vista “register.jsp” del Cliente Web 75](#_Toc199756348)

[Figura 47. Visualización del historial de compras del cliente en la vista “compras.jsp” 76](#_Toc199756349)

[Figura 48. Resultados de búsqueda de vuelos mostrados en la vista “result.jsp” 76](#_Toc199756350)

[Figura 49. Creación de un nuevo proyecto Java Application para el Cliente de Consola 77](#_Toc199756351)

[Figura 50. Configuración del nombre del proyecto VIAJECITOS\_SOAP\_CLICON\_GR10 77](#_Toc199756352)

[Figura 51. Creación de paquetes controlador y servicio para el Cliente de Consola 78](#_Toc199756353)

[Figura 52. Creación del Web Service Client con la URL del WSDL 78](#_Toc199756354)

[Figura 53. Implementación de la clase “ViajecitosService” para el consumo del servicio web en el Cliente Consola 79](#_Toc199756355)

[Figura 54. Creación e implementación de métodos en ViajecitosController 79](#_Toc199756356)

[Figura 55. Diseño de la interfaz de consola en la clase MainConsole para el Cliente Consola 80](#_Toc199756357)

[Figura 56. Creación del proyecto VIAJECITOS\_SOAP\_GR10 en Android Studio 81](#_Toc199756358)

[Figura 57. Creación de la clase SoapConstants para las constantes del servicio SOAP 83](#_Toc199756359)

[Figura 58. Integración del servicio ViajecitosService 83](#_Toc199756360)

[Figura 59. Creación e implementación de AppControlado 84](#_Toc199756361)

[Figura 60. Interfaz de LoginActivity para el inicio de sesión en la aplicación móvil 84](#_Toc199756362)

[Figura 61. Pantalla principal MainActivity con opciones principales para el usuario 85](#_Toc199756363)

[Figura 62. Visualización del historial de compras en PurchasesActivity 85](#_Toc199756364)

[Figura 63. Pantalla de resultados de vuelos en ResultActivity para la selección y compra 86](#_Toc199756365)

**ÍNDICE DE TABLAS**

[*Tabla 1.* Configuración de Dependencias y Plugins en el archivo pom.xml para WSAerolineasCondor 12](#_Toc199756366)

[*Tabla 2.* Implementación de Métodos para la Compra de Boletos en ViajecitosService 15](#_Toc199756367)

[*Tabla 3.* Integración de Métodos al Web Service WSAerolineasCondor 26](#_Toc199756368)

[*Tabla 4.* Creación del Paquete ec.edu.monster.db y la Clase AccesoDB 33](#_Toc199756369)

[*Tabla 5.* Creación de la Clase Cliente en ec.edu.monster.modelo 35](#_Toc199756370)

[*Tabla 6.* Creación de la Clase Compra en ec.edu.monster.modelo 37](#_Toc199756371)

[*Tabla 7.* Creación de las Clases Usuario y Vuelo en ec.edu.monster.modelo 40](#_Toc199756372)

[*Tabla 8.* Creación de ViajecitosService y adición de operaciones del Web Service 49](#_Toc199756373)

[*Tabla 9.* Creación e implementación de métodos en ViajecitosController 53](#_Toc199756374)

[*Tabla 10.* Configuración del archivo pom.xml para el consumo del servicio web 66](#_Toc199756375)

[*Tabla 11.* Configuración de dependencias y permisos en el proyecto 81](#_Toc199756376)

# EXAMEN UNIDAD 1 AEROLÍNEA\_SOAP\_JAVA

## INTRODUCCIÓN

El presente documento describe la implementación de dos proyectos web desarrollados como parte del Examen Práctico del Primer Parcial de la asignatura PAO202550. Ambos proyectos están enfocados en simular un sistema de compra de boletos de avión para la agencia de viajes "Viajecitos SA", la cual se conecta con un servicio web expuesto por la aerolínea "Aerolíneas Cóndor".

Cada uno de los proyectos cumple la función de servidor que expone los servicios web requeridos por *Viajecitos SA*. Uno de estos servidores fue implementado en Java mediante NetBeans 21, utilizando el protocolo SOAP y una base de datos MySQL gestionada con XAMPP. El segundo servidor fue desarrollado en C# utilizando Visual Studio Community 2022, empleando el protocolo REST y SQL Server como sistema gestor de base de datos.

A su vez, cada servidor cuenta con cuatro clientes web que simulan el sistema de la agencia *Viajecitos SA*, cumpliendo funciones como búsqueda de vuelos, visualización del vuelo de mayor valor, y compra de boletos. Esta estructura permite demostrar cómo un mismo requerimiento funcional puede ser abordado mediante dos tecnologías distintas y protocolos de comunicación diferentes, destacando la flexibilidad y adaptabilidad de los servicios web en entornos heterogéneos.

## OBJETIVO

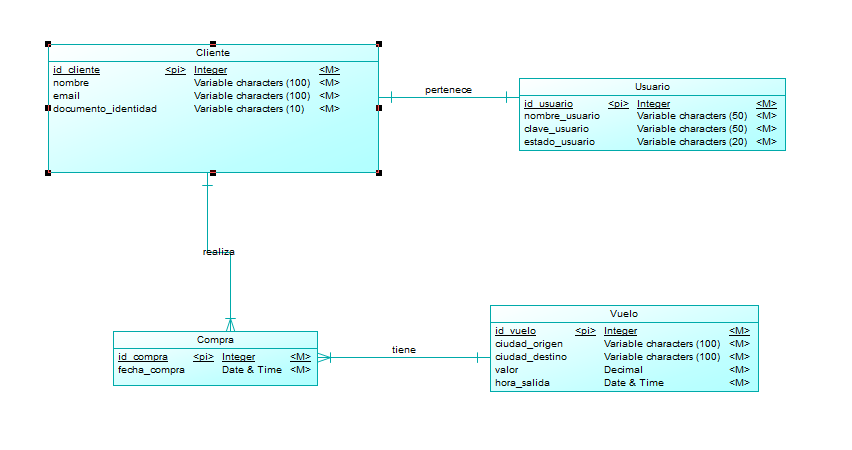
Desarrollar dos aplicaciones servidoras que expongan servicios web para la simulación de un sistema de venta de boletos de avión, uno utilizando el protocolo SOAP con Java y MySQL, y otro utilizando el protocolo REST con .NET y SQL Server. Cada aplicación debe integrar cuatro clientes web que representen el sistema de la agencia “Viajecitos SA”, permitiendo realizar búsquedas de vuelos, mostrar el vuelo de mayor valor y registrar la compra del boleto. El objetivo principal es evidenciar la implementación de servicios web funcionales y su consumo por parte de interfaces web, utilizando distintas tecnologías y protocolos de comunicación para resolver un mismo problema en contextos diferentes.

## DESARROLLO

### CREACIÓN MODELADO BASE DE DATOS

#### Modelo conceptual

* **Entidad Vuelo**: representa los vuelos ofrecidos, con atributos como ciudad de origen, destino, valor y hora de salida.
* **Entidad Cliente**: representa a los clientes que pueden realizar compras, identificados por su nombre, email y documento de identidad.
* **Entidad Usuario**: representa las credenciales de acceso asociadas a un cliente específico, incluyendo nombre de usuario, clave y estado.
* **Entidad Compra**: representa la relación entre un cliente y un vuelo, indicando que un cliente ha comprado un boleto.



1. Modelo Conceptual del servicio web

#### Modelo lógico

Este modelo traduce las entidades del modelo conceptual a tablas relacionales, con atributos bien definidos, claves primarias y relaciones por medio de claves foráneas. Establece restricciones como unicidad en usuarios y emails de cliente, y relaciones entre vuelos, clientes y compras.

Diagrama

El contenido generado por IA puede ser incorrecto.

1. Modelo Lógico con tablas, claves primarias y relaciones entre entidades

#### Modelo físico

El modelo físico implementa la estructura real en el motor de base de datos. Utiliza tipos de datos específicos, como VARCHAR, NUMERIC, y DATETIME, e incluye la creación de índices para mejorar el rendimiento. Además, garantiza la integridad referencial mediante claves foráneas y restricciones únicas.

Diagrama

El contenido generado por IA puede ser incorrecto.

1. Modelo Físico implementado en MySQL para la base de datos vuelos\_db

### INTEGRACIÓN CON MYSQL

Para integrar el sistema con el gestor de bases de datos local, se utilizó XAMPP como entorno de desarrollo. Desde phpMyAdmin, se procedió a la creación de la base de datos vuelos\_db, empleando la estructura generada previamente en Power Designer bajo el modelo físico compatible con MySQL. Esta integración permitió importar directamente el script SQL que contiene la definición de tablas, relaciones y restricciones.

Una captura de pantalla de una red social

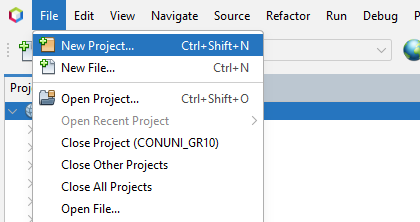
El contenido generado por IA puede ser incorrecto.

1. Creación de la base de datos vuelos\_db en phpMyAdmin del gestor XAMPP a partir del modelo físico

### APLICACIÓN SERVIDOR JAVA SOAP

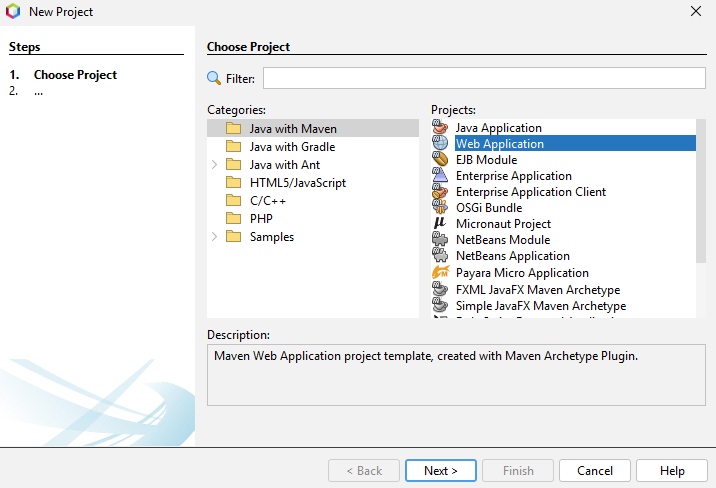
#### Creación del proyecto servidor

Para comenzar, abra el IDE Apache NetBeans y, desde la pantalla principal, diríjase a la barra de menús superior. Haga clic en Archivo y seleccione Nuevo Proyecto para iniciar la creación de la aplicación web SOAP destinada a la agencia de viajes



1. Creación nuevo proyecto para el servidor en Netbeans

En la ventana emergente, ubicar en la categoría “Java with Maven” el tipo de proyecto “Java Web” y seleccionar “Aplicación Web”.



1. Selección de tipo de proyecto en Apache NetBeans

A continuación, asignar el nombre del proyecto, en este caso “VIAJECITOS\_SOAP\_GR10”, y especificar su ubicación en el directorio correspondiente al servidor de SOAP en Java.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

1. Configuración del nombre y ubicación del proyecto

Posteriormente, configurar las opciones del proyecto y del servidor donde se ejecutará una vez desplegado. Seleccionar “Payara Server” como servidor y, para la versión de Java, elegir preferentemente la más reciente.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Configuración del servidor y versión de Java

Para garantizar una estructura adecuada del proyecto, se implementará el patrón arquitectónico MVC. Para ello, crear en “Source Packages” los siguientes paquetes: ec.edu.monster.controlador, ec.edu.monster.vista, ec.edu.monster.modelo, ec.edu.monster.prueba, ec.edu.monster.servicio

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Creación de paquetes para la estructura MVC

#### Creación del servicio web

Crear el servicio web en el paquete “ec.edu.monster.controlador” haciendo clic en “Other…”, seleccionando la carpeta “Web Services” y luego “Web Service”. El Web Service, al facilitar la comunicación entre dos aplicaciones, funciona de manera similar a un controlador, ya que recibe solicitudes de una aplicación externa y las transmite a otro componente para su ejecución. Por esta razón, se crea dentro del paquete “controlador”.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Creación del Web Service en el paquete controlador

En el proceso de configuración, asigne el nombre **WSAerolineasCondor** al Web Service en Apache NetBeans. Asegúrese de que se cree dentro del paquete **ec.edu.monster.controlador** para mantener la estructura organizada del proyecto SOAP de la agencia de viajes.

.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

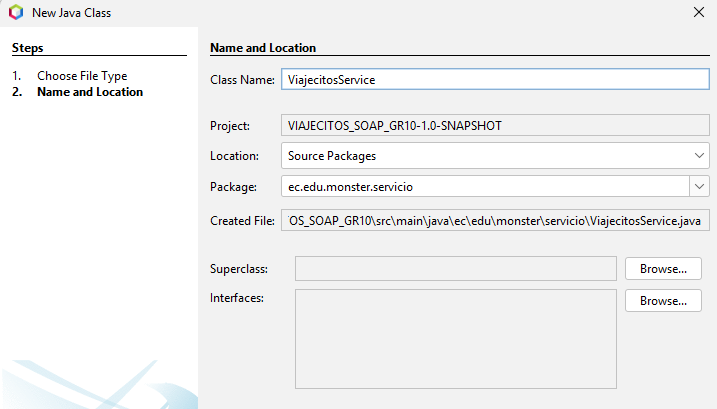
1. Configuración del nombre y paquete del Web Service WSAerolineasCondor

Una vez creado el Web Service, abra la carpeta Project Files en Apache NetBeans y edite el archivo pom.xml. Actualice las dependencias para incluir jakarta.jakartaee-api y el conector mysql-connector-j, y configure los plugins como maven-compiler-plugin y maven-war-plugin para soportar Jakarta y la conexión a MySQL.

1. Configuración de Dependencias y Plugins en el archivo pom.xml para WSAerolineasCondor

|  |
| --- |
| <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>ec.edu.monster</groupId>  <artifactId>WSAerolineasCondor</artifactId>  <version>1.0-SNAPSHOT</version>  <packaging>war</packaging>  <name>WSAerolineasCondor-1.0-SNAPSHOT</name>    <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <jakartaee>10.0.0</jakartaee>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </properties>    <dependencies>  <dependency>  <groupId>jakarta.platform</groupId>  <artifactId>jakarta.jakartaee-api</artifactId>  <version>${jakartaee}</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-j</artifactId>  <version>8.2.0</version>  </dependency>  <dependency>  <groupId>org.glassfish.metro</groupId>  <artifactId>webservices-rt</artifactId>  <version>4.0.1</version>  <scope>provided</scope>  </dependency>  </dependencies>    <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.13.0</version>  <configuration>  <source>${maven.compiler.source}</source>  <target>${maven.compiler.target}</target>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-war-plugin</artifactId>  <version>3.4.0</version>  </plugin>  </plugins>  </build>  </project> |

Con el servicio web ya creado, desarrolle las operaciones necesarias para la compra de boletos dentro del paquete ec.edu.monster.servicio. Haga clic derecho sobre el paquete, seleccione New, luego Java Class y asigne el nombre ViajecitosService a la clase para implementar la lógica del servicio.



1. Creación de la clase ConversionService en el paquete servicio

En el archivo ViajecitosService.java, ubicado en el paquete ec.edu.monster.servicio, implemente la lógica para la compra de boletos de vuelos, incluyendo métodos como buscarVuelos, obtenerVueloMasCaro, registrarCompra, registrarCliente, registrarUsuario, iniciarSesion y obtenerComprasCliente. Estas funciones gestionan la búsqueda de vuelos, registro de clientes y usuarios, autenticación, y manejo de compras para la agencia de viajes.

1. Implementación de Métodos para la Compra de Boletos en ViajecitosService

|  |
| --- |
| package ec.edu.monster.servicio;  import ec.edu.monster.db.AccesoDB;  import ec.edu.monster.modelo.Cliente;  import ec.edu.monster.modelo.Compra;  import ec.edu.monster.modelo.Usuario;  import ec.edu.monster.modelo.Vuelo;  import java.sql.Connection;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.Date;  import java.util.List;  /\*\*  \* Servicio para operaciones de Viajecitos SA.  \* @author MATIAS  \*/  public class ViajecitosService {  private Connection getConnection() throws SQLException {  return AccesoDB.getConnection();  }  public List<Vuelo> buscarVuelos(String ciudadOrigen, String ciudadDestino, Date fecha) throws SQLException {  List<Vuelo> lista = new ArrayList<>();  String sql = "SELECT id\_vuelo, ciudad\_origen, ciudad\_destino, valor, hora\_salida " +  "FROM Vuelo WHERE ciudad\_origen = ? AND ciudad\_destino = ? AND DATE(hora\_salida) = ?";  try (Connection conn = getConnection(); PreparedStatement ps = conn.prepareStatement(sql)) {  ps.setString(1, ciudadOrigen);  ps.setString(2, ciudadDestino);  ps.setDate(3, new java.sql.Date(fecha.getTime()));  try (ResultSet rs = ps.executeQuery()) {  while (rs.next()) {  Vuelo vuelo = new Vuelo();  vuelo.setIdVuelo(rs.getInt("id\_vuelo"));  vuelo.setCiudadOrigen(rs.getString("ciudad\_origen"));  vuelo.setCiudadDestino(rs.getString("ciudad\_destino"));  vuelo.setValor(rs.getDouble("valor"));  vuelo.setHoraSalida(rs.getTimestamp("hora\_salida"));  lista.add(vuelo);  }  }  }  return lista;  }  public Vuelo obtenerVueloMasCaro(String ciudadOrigen, String ciudadDestino, Date fecha) throws SQLException {  String sql = "SELECT id\_vuelo, ciudad\_origen, ciudad\_destino, valor, hora\_salida " +  "FROM Vuelo WHERE ciudad\_origen = ? AND ciudad\_destino = ? AND DATE(hora\_salida) = ? " +  "ORDER BY valor DESC LIMIT 1";  try (Connection conn = getConnection(); PreparedStatement ps = conn.prepareStatement(sql)) {  ps.setString(1, ciudadOrigen);  ps.setString(2, ciudadDestino);  ps.setDate(3, new java.sql.Date(fecha.getTime()));  try (ResultSet rs = ps.executeQuery()) {  if (rs.next()) {  Vuelo vuelo = new Vuelo();  vuelo.setIdVuelo(rs.getInt("id\_vuelo"));  vuelo.setCiudadOrigen(rs.getString("ciudad\_origen"));  vuelo.setCiudadDestino(rs.getString("ciudad\_destino"));  vuelo.setValor(rs.getDouble("valor"));  vuelo.setHoraSalida(rs.getTimestamp("hora\_salida"));  return vuelo;  }  }  }  return null;  }  public void registrarCompra(int idVuelo, int idCliente) throws Exception {  Connection conn = null;  try {  conn = getConnection();  conn.setAutoCommit(false);  // Validar vuelo  PreparedStatement ps = conn.prepareStatement("SELECT id\_vuelo FROM Vuelo WHERE id\_vuelo = ?");  ps.setInt(1, idVuelo);  ResultSet rs = ps.executeQuery();  if (!rs.next()) {  throw new Exception("Vuelo no válido.");  }  // Validar cliente  ps = conn.prepareStatement("SELECT id\_cliente FROM Cliente WHERE id\_cliente = ?");  ps.setInt(1, idCliente);  rs = ps.executeQuery();  if (!rs.next()) {  throw new Exception("Cliente no válido.");  }  // Registrar compra  ps = conn.prepareStatement(  "INSERT INTO Compra (id\_vuelo, id\_cliente, fecha\_compra) VALUES (?, ?, NOW())");  ps.setInt(1, idVuelo);  ps.setInt(2, idCliente);  ps.executeUpdate();  conn.commit();  } catch (Exception e) {  if (conn != null) {  conn.rollback();  }  throw e;  } finally {  if (conn != null) {  conn.close();  }  }  }  public Cliente registrarCliente(String nombre, String email, String documentoIdentidad) throws Exception {  Connection conn = null;  try {  conn = getConnection();  conn.setAutoCommit(false);  // Verificar si el email o documento ya existen  PreparedStatement ps = conn.prepareStatement(  "SELECT id\_cliente FROM Cliente WHERE email = ? OR documento\_identidad = ?");  ps.setString(1, email);  ps.setString(2, documentoIdentidad);  ResultSet rs = ps.executeQuery();  if (rs.next()) {  throw new Exception("Email o documento de identidad ya registrado.");  }  // Registrar cliente  ps = conn.prepareStatement(  "INSERT INTO Cliente (nombre, email, documento\_identidad) VALUES (?, ?, ?)",  PreparedStatement.RETURN\_GENERATED\_KEYS);  ps.setString(1, nombre);  ps.setString(2, email);  ps.setString(3, documentoIdentidad);  ps.executeUpdate();  rs = ps.getGeneratedKeys();  int idCliente = rs.next() ? rs.getInt(1) : 0;  Cliente cliente = new Cliente();  cliente.setIdCliente(idCliente);  cliente.setNombre(nombre);  cliente.setEmail(email);  cliente.setDocumentoIdentidad(documentoIdentidad);  conn.commit();  return cliente;  } catch (Exception e) {  if (conn != null) {  conn.rollback();  }  throw e;  } finally {  if (conn != null) {  conn.close();  }  }  }  public Usuario registrarUsuario(int idCliente, String nombreUsuario, String claveUsuario) throws Exception {  Connection conn = null;  try {  conn = getConnection();  conn.setAutoCommit(false);  // Verificar cliente  PreparedStatement ps = conn.prepareStatement("SELECT id\_cliente FROM Cliente WHERE id\_cliente = ?");  ps.setInt(1, idCliente);  ResultSet rs = ps.executeQuery();  if (!rs.next()) {  throw new Exception("Cliente no válido.");  }  // Verificar nombre de usuario  ps = conn.prepareStatement("SELECT id\_usuario FROM Usuario WHERE nombre\_usuario = ?");  ps.setString(1, nombreUsuario);  rs = ps.executeQuery();  if (rs.next()) {  throw new Exception("Nombre de usuario ya registrado.");  }  // Registrar usuario  ps = conn.prepareStatement(  "INSERT INTO Usuario (id\_cliente, nombre\_usuario, clave\_usuario, estado\_usuario) VALUES (?, ?, ?, 'ACTIVO')",  PreparedStatement.RETURN\_GENERATED\_KEYS);  ps.setInt(1, idCliente);  ps.setString(2, nombreUsuario);  ps.setString(3, claveUsuario); // En una aplicación real, hashear la contraseña  ps.executeUpdate();  rs = ps.getGeneratedKeys();  int idUsuario = rs.next() ? rs.getInt(1) : 0;  Usuario usuario = new Usuario();  usuario.setIdUsuario(idUsuario);  usuario.setIdCliente(idCliente);  usuario.setNombreUsuario(nombreUsuario);  usuario.setClaveUsuario(claveUsuario);  usuario.setEstadoUsuario("ACTIVO");  conn.commit();  return usuario;  } catch (Exception e) {  if (conn != null) {  conn.rollback();  }  throw e;  } finally {  if (conn != null) {  conn.close();  }  }  }  public Usuario iniciarSesion(String nombreUsuario, String claveUsuario) throws Exception {  Connection conn = null;  try {  conn = getConnection();  // Validar credenciales  PreparedStatement ps = conn.prepareStatement(  "SELECT id\_usuario, id\_cliente, nombre\_usuario, clave\_usuario, estado\_usuario " +  "FROM Usuario WHERE nombre\_usuario = ? AND clave\_usuario = ? AND estado\_usuario = 'ACTIVO'");  ps.setString(1, nombreUsuario);  ps.setString(2, claveUsuario); // En una aplicación real, comparar contraseña hasheada  ResultSet rs = ps.executeQuery();  if (!rs.next()) {  throw new Exception("Credenciales inválidas o usuario inactivo.");  }  // Crear objeto Usuario  Usuario usuario = new Usuario();  usuario.setIdUsuario(rs.getInt("id\_usuario"));  usuario.setIdCliente(rs.getInt("id\_cliente"));  usuario.setNombreUsuario(rs.getString("nombre\_usuario"));  usuario.setClaveUsuario(rs.getString("clave\_usuario"));  usuario.setEstadoUsuario(rs.getString("estado\_usuario"));  return usuario;  } catch (Exception e) {  throw e;  } finally {  if (conn != null) {  try {  conn.close();  } catch (SQLException e) {  throw new SQLException("Error al cerrar la conexión: " + e.getMessage());  }  }  }  }    public List<Compra> obtenerComprasCliente(int idCliente) throws SQLException {  List<Compra> lista = new ArrayList<>();  String sql = "SELECT c.id\_compra, c.id\_vuelo, c.id\_cliente, c.fecha\_compra, " +  "v.ciudad\_origen, v.ciudad\_destino, v.valor, v.hora\_salida " +  "FROM Compra c JOIN Vuelo v ON c.id\_vuelo = v.id\_vuelo " +  "WHERE c.id\_cliente = ?";  try (Connection conn = getConnection(); PreparedStatement ps = conn.prepareStatement(sql)) {  ps.setInt(1, idCliente);  try (ResultSet rs = ps.executeQuery()) {  while (rs.next()) {  Compra compra = new Compra();  compra.setIdCompra(rs.getInt("id\_compra"));  compra.setIdVuelo(rs.getInt("id\_vuelo"));  compra.setIdCliente(rs.getInt("id\_cliente"));  compra.setFechaCompra(rs.getTimestamp("fecha\_compra"));  Vuelo vuelo = new Vuelo();  vuelo.setIdVuelo(rs.getInt("id\_vuelo"));  vuelo.setCiudadOrigen(rs.getString("ciudad\_origen"));  vuelo.setCiudadDestino(rs.getString("ciudad\_destino"));  vuelo.setValor(rs.getDouble("valor"));  vuelo.setHoraSalida(rs.getTimestamp("hora\_salida"));  compra.setVuelo(vuelo);  lista.add(compra);  }  }  }  return lista;  }  } |

Una vez desarrollados los métodos en ViajecitosService, incorpórelos al Web Service WSAerolineasCondor para habilitar las operaciones de compra de boletos. Asegúrese de anotar los métodos con @WebMethod y configurar el archivo adecuadamente para exponer las funcionalidades a través del servicio SOAP.

1. Integración de Métodos al Web Service WSAerolineasCondor

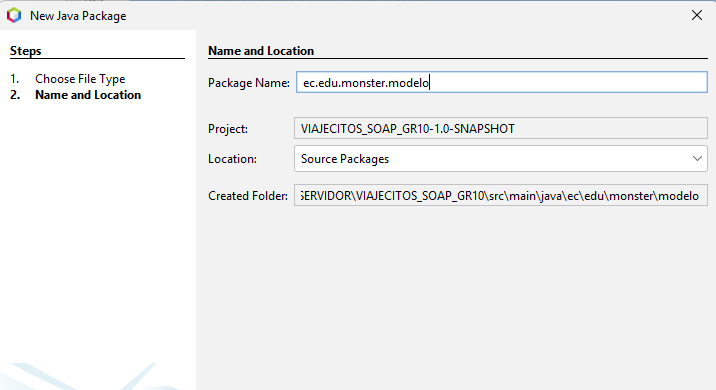
|  |
| --- |
| package ec.edu.monster.controlador;  import ec.edu.monster.modelo.Cliente;  import ec.edu.monster.modelo.Usuario;  import ec.edu.monster.modelo.Vuelo;  import ec.edu.monster.modelo.Compra;  import ec.edu.monster.servicio.ViajecitosService;  import jakarta.jws.WebMethod;  import jakarta.jws.WebParam;  import jakarta.jws.WebResult;  import jakarta.jws.WebService;  import java.util.ArrayList;  import java.util.Date;  import java.util.List;  /\*\*  \* SOAP Web Service for Aerolíneas Cóndor operations.  \*  \* @author MATIAS  \*/  @WebService(serviceName = "WSAerolineasCondor")  public class WSAerolineasCondor {  private final ViajecitosService viajecitosService;  public WSAerolineasCondor() {  this.viajecitosService = new ViajecitosService();  }  /\*\*  \* Retrieves a list of flights based on origin, destination, and date.  \*  \* @param ciudadOrigen Origin city  \* @param ciudadDestino Destination city  \* @param fecha Travel date  \* @return List of available flights  \*/  @WebMethod(operationName = "buscarVuelos")  @WebResult(name = "Vuelo")  public List<Vuelo> buscarVuelos(  @WebParam(name = "ciudadOrigen") String ciudadOrigen,  @WebParam(name = "ciudadDestino") String ciudadDestino,  @WebParam(name = "fecha") Date fecha) {  List<Vuelo> lista;  try {  lista = viajecitosService.buscarVuelos(ciudadOrigen, ciudadDestino, fecha);  } catch (Exception e) {  lista = new ArrayList<>();  }  return lista;  }  /\*\*  \* Retrieves the most expensive flight based on origin, destination, and  \* date.  \*  \* @param ciudadOrigen Origin city  \* @param ciudadDestino Destination city  \* @param fecha Travel date  \* @return Most expensive flight or null if none available  \*/  @WebMethod(operationName = "obtenerVueloMasCaro")  @WebResult(name = "Vuelo")  public Vuelo obtenerVueloMasCaro(  @WebParam(name = "ciudadOrigen") String ciudadOrigen,  @WebParam(name = "ciudadDestino") String ciudadDestino,  @WebParam(name = "fecha") Date fecha) {  try {  return viajecitosService.obtenerVueloMasCaro(ciudadOrigen, ciudadDestino, fecha);  } catch (Exception e) {  return null;  }  }  /\*\*  \* Registers a ticket purchase.  \*  \* @param idVuelo Flight ID  \* @param idCliente Client ID  \* @return Status (1 for success, -1 for failure)  \*/  @WebMethod(operationName = "registrarCompra")  @WebResult(name = "estado")  public int registrarCompra(  @WebParam(name = "idVuelo") int idVuelo,  @WebParam(name = "idCliente") int idCliente) {  try {  viajecitosService.registrarCompra(idVuelo, idCliente);  return 1;  } catch (Exception e) {  return -1;  }  }  /\*\*  \* Registers a new client.  \*  \* @param nombre Client name  \* @param email Client email  \* @param documentoIdentidad Client ID document  \* @return Registered client or null if failure  \*/  @WebMethod(operationName = "registrarCliente")  @WebResult(name = "Cliente")  public Cliente registrarCliente(  @WebParam(name = "nombre") String nombre,  @WebParam(name = "email") String email,  @WebParam(name = "documentoIdentidad") String documentoIdentidad) {  try {  return viajecitosService.registrarCliente(nombre, email, documentoIdentidad);  } catch (Exception e) {  return null;  }  }  /\*\*  \* Registers a new user.  \*  \* @param idCliente Client ID  \* @param nombreUsuario Username  \* @param claveUsuario Password  \* @return Registered user or null if failure  \*/  @WebMethod(operationName = "registrarUsuario")  @WebResult(name = "Usuario")  public Usuario registrarUsuario(  @WebParam(name = "idCliente") int idCliente,  @WebParam(name = "nombreUsuario") String nombreUsuario,  @WebParam(name = "claveUsuario") String claveUsuario) {  try {  return viajecitosService.registrarUsuario(idCliente, nombreUsuario, claveUsuario);  } catch (Exception e) {  return null;  }  }  /\*\*  \* Logs in a user.  \*  \* @param nombreUsuario Username  \* @param claveUsuario Password  \* @return Authenticated user or null if failure  \*/  @WebMethod(operationName = "iniciarSesion")  @WebResult(name = "Usuario")  public Usuario iniciarSesion(  @WebParam(name = "nombreUsuario") String nombreUsuario,  @WebParam(name = "claveUsuario") String claveUsuario) {  try {  return viajecitosService.iniciarSesion(nombreUsuario, claveUsuario);  } catch (Exception e) {  return null;  }  }  /\*\*  \* Retrieves all purchases made by a client.  \*  \* @param idCliente Client ID  \* @return List of purchases or empty list if failure  \*/  @WebMethod(operationName = "obtenerComprasCliente")  @WebResult(name = "Compra")  public List<Compra> obtenerComprasCliente(  @WebParam(name = "idCliente") int idCliente) {  try {  return viajecitosService.obtenerComprasCliente(idCliente);  } catch (Exception e) {  return new ArrayList<>();  }  }  } |

Crear el paquete **ec.edu.monster.db** haciendo clic derecho sobre el proyecto, seleccionando **New** y luego **Java Package**. A continuación, dentro de este paquete, haga clic derecho, seleccione **New**, luego **Java Class**, y nombre la clase **AccesoDB** para gestionar la conexión a la base de datos del sistema de la agencia de viajes.

1. Creación del Paquete ec.edu.monster.db y la Clase AccesoDB

|  |
| --- |
| package ec.edu.monster.db;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.SQLException;  /\*\*  \* Provides access to the vuelos\_db database.  \* @author MATIAS  \*/  public class AccesoDB {  // Database connection parameters  private static final String URL = "jdbc:mysql://localhost:3306/vuelos\_db?useSSL=false&serverTimezone=UTC";  private static final String USER = "root"; // Default XAMPP MySQL user  private static final String PASSWORD = ""; // Default XAMPP MySQL password (empty)  // Private constructor to prevent instantiation  private AccesoDB() {  }  public static Connection getConnection() throws SQLException {  Connection conn = null;  try {  // Load the MySQL JDBC driver  Class.forName("com.mysql.cj.jdbc.Driver");  // Establish the connection  conn = DriverManager.getConnection(URL, USER, PASSWORD);  } catch (ClassNotFoundException e) {  throw new SQLException("MySQL JDBC Driver not found: " + e.getMessage());  } catch (SQLException e) {  throw new SQLException("ERROR, no se tiene acceso al Servidor: " + e.getMessage());  }  return conn;  }  } |

En Apache NetBeans, diríjase a **Source Packages** y haga clic derecho para crear un nuevo paquete. Seleccione **New** y luego **Java Package**, nombrándolo **ec.edu.monster.modelo** para organizar las entidades de la base de datos de la agencia de viajes.



1. Creación del Paquete ec.edu.monster.modelo

Dentro del paquete **ec.edu.monster.modelo**, cree la clase para la entidad **Cliente**. Haga clic derecho sobre el paquete, seleccione **New**, luego **Java Class**, y asigne el nombre **Cliente** para modelar los datos de los clientes, como nombre, email y documento de identidad.

1. Creación de la Clase Cliente en ec.edu.monster.modelo

|  |
| --- |
| a package ec.edu.monster.modelo;  import jakarta.xml.bind.annotation.XmlRootElement;  /\*\*  \* Model for the Cliente table.  \* @author MATIAS  \*/  @XmlRootElement(name = "cliente")  public class Cliente {    private int idCliente;  private String nombre;  private String email;  private String documentoIdentidad;  public Cliente() {  }  public Cliente(int idCliente, String nombre, String email, String documentoIdentidad) {  this.idCliente = idCliente;  this.nombre = nombre;  this.email = email;  this.documentoIdentidad = documentoIdentidad;  }  public int getIdCliente() {  return idCliente;  }  public void setIdCliente(int idCliente) {  this.idCliente = idCliente;  }  public String getNombre() {  return nombre;  }  public void setNombre(String nombre) {  this.nombre = nombre;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public String getDocumentoIdentidad() {  return documentoIdentidad;  }  public void setDocumentoIdentidad(String documentoIdentidad) {  this.documentoIdentidad = documentoIdentidad;  }  } |

A continuación, en el mismo paquete, cree la clase para la entidad Compra. Haga clic derecho, seleccione New, luego Java Class, y nombre la clase Compra para representar las transacciones de compra de boletos, incluyendo atributos como ID de vuelo, ID de cliente y fecha de compra.

1. Creación de la Clase Compra en ec.edu.monster.modelo

|  |
| --- |
| a package ec.edu.monster.modelo;  import jakarta.xml.bind.annotation.XmlRootElement;  import java.util.Date;  /\*\*  \* Model for the Compra table.  \*  \* @author MATIAS  \*/  @XmlRootElement(name = "compra")  public class Compra {  private int idCompra;  private int idVuelo;  private int idCliente;  private Date fechaCompra;  private Vuelo vuelo;  public Compra() {  }  public Compra(int idCompra, int idVuelo, int idCliente, Date fechaCompra) {  this.idCompra = idCompra;  this.idVuelo = idVuelo;  this.idCliente = idCliente;  this.fechaCompra = fechaCompra;  }  public int getIdCompra() {  return idCompra;  }  public void setIdCompra(int idCompra) {  this.idCompra = idCompra;  }  public int getIdVuelo() {  return idVuelo;  }  public void setIdVuelo(int idVuelo) {  this.idVuelo = idVuelo;  }  public int getIdCliente() {  return idCliente;  }  public void setIdCliente(int idCliente) {  this.idCliente = idCliente;  }  public Date getFechaCompra() {  return fechaCompra;  }  public void setFechaCompra(Date fechaCompra) {  this.fechaCompra = fechaCompra;  }  public Vuelo getVuelo() {  return vuelo;  }  public void setVuelo(Vuelo vuelo) {  this.vuelo = vuelo;  }  } |

Finalmente, dentro de ec.edu.monster.modelo, cree las clases Usuario y Vuelo. Haga clic derecho, seleccione New, luego Java Class, y asigne los nombres Usuario y Vuelo para modelar los datos de usuarios (credenciales y estado) y vuelos (origen, destino, valor y hora de salida), respectivamente.

1. Creación de las Clases Usuario y Vuelo en ec.edu.monster.modelo

|  |
| --- |
| a package ec.edu.monster.modelo;  import jakarta.xml.bind.annotation.XmlRootElement;  /\*\*  \* Model for the Usuario table.  \* @author MATIAS  \*/  @XmlRootElement(name = "usuario")  public class Usuario {    private int idUsuario;  private int idCliente;  private String nombreUsuario;  private String claveUsuario;  private String estadoUsuario;  public Usuario() {  }  public Usuario(int idUsuario, int idCliente, String nombreUsuario, String claveUsuario, String estadoUsuario) {  this.idUsuario = idUsuario;  this.idCliente = idCliente;  this.nombreUsuario = nombreUsuario;  this.claveUsuario = claveUsuario;  this.estadoUsuario = estadoUsuario;  }  public int getIdUsuario() {  return idUsuario;  }  public void setIdUsuario(int idUsuario) {  this.idUsuario = idUsuario;  }  public int getIdCliente() {  return idCliente;  }  public void setIdCliente(int idCliente) {  this.idCliente = idCliente;  }  public String getNombreUsuario() {  return nombreUsuario;  }  public void setNombreUsuario(String nombreUsuario) {  this.nombreUsuario = nombreUsuario;  }  public String getClaveUsuario() {  return claveUsuario;  }  public void setClaveUsuario(String claveUsuario) {  this.claveUsuario = claveUsuario;  }  public String getEstadoUsuario() {  return estadoUsuario;  }  public void setEstadoUsuario(String estadoUsuario) {  this.estadoUsuario = estadoUsuario;  }  } |

En Apache NetBeans, diríjase a Source Packages, haga clic derecho y seleccione New, luego Java Package, nombrándolo ec.edu.monster.pruebas para los métodos de ViajecitosService, como buscarVuelos, obtenerVueloMasCaro, registrarCompra, registrarCliente, registrarUsuario, iniciarSesion y obtenerComprasCliente, validando su funcionalidad y manejo de excepciones.

Texto

El contenido generado por IA puede ser incorrecto.

1. Creación del Paquete ec.edu.monster.pruebas y Pruebas para ViajecitosService

Una vez implementado correctamente, ejecute el proyecto en Apache NetBeans para generar el archivo WSDL; al hacerlo, el navegador se abrirá en la URL http://localhost:8080/CONUNI\_GR10/, a la cual debe añadir el nombre del servicio web WSAerolineasCondor, resultando en http://localhost:8080/CONUNI\_GR10/WSAerolineasCondor, donde podrá visualizar la página del servicio y copiar la URL del WSDL: http://localhost:8080/CONUNI\_GR10/WSAerolineasCondor?wsdl.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Ejecución del proyecto y obtención de la URL del WSDL

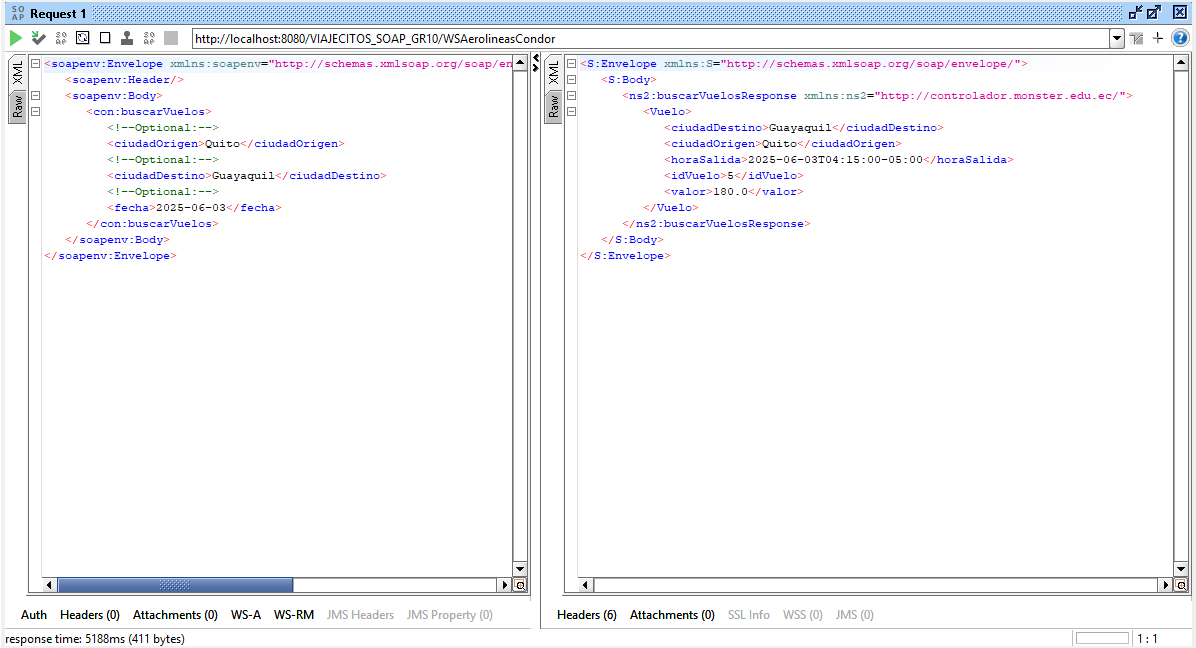
Para validar el funcionamiento del servicio web de la agencia de viajes, inicie SOAP UI y configure un nuevo proyecto, ingresando la URL del WSDL generada previamente: http://localhost:8080/CONUNI\_GR10/WSAerolineasCondor?wsdl. Utilice esta herramienta para enviar solicitudes de prueba a los métodos expuestos, verificando las respuestas y asegurando que el servicio opere correctamente para la agencia de viajes.

Texto

El contenido generado por IA puede ser incorrecto.

1. Confoguración del servicio web en SOAP UI

Una vez creado el proyecto en SOAP UI, se procede a probar la operación buscarVuelos. Esta operación permite consultar los vuelos disponibles según los parámetros definidos en el servicio. Al ejecutar la solicitud desde SOAP UI, se puede verificar que el servicio web responde correctamente con la información esperada.



1. Ejecución de la operación buscarVuelos en SOAP UI y validación de la respuesta del servicio

Posteriormente, se realiza la prueba de la operación iniciarSesion desde SOAP UI. Esta operación permite autenticar a un usuario dentro del sistema de la agencia de viajes. Al enviar la solicitud con las credenciales correspondientes, se debe recibir una respuesta satisfactoria que confirme el inicio de sesión, validando así el correcto funcionamiento del servicio web.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Prueba de la operación iniciarSesion en SOAP UI con respuesta exitosa del servicio

A continuación, se prueba la operación obtenerComprasCliente desde SOAP UI. Esta funcionalidad permite recuperar el historial de compras de boletos realizadas por un cliente específico. Al ejecutar la solicitud con el identificador del cliente, el servicio debe retornar la lista de compras asociadas, confirmando que la operación funciona correctamente.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

1. Ejecución de la operación obtenerComprasCliente en SOAP UI para consultar el historial de compras del cliente

Se lleva a cabo la prueba de la operación obtenerVueloMasCaro mediante SOAP UI. Esta operación permite consultar cuál es el vuelo con el precio más elevado registrado en el sistema. Al ejecutar la solicitud, el servicio debe responder con los datos correspondientes al vuelo más costoso, evidenciando así que la lógica de negocio implementada funciona correctamente.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Prueba de la operación obtenerVueloMasCaro en SOAP UI y visualización de la respuesta del vuelo con mayor costo

Seguidamente, se realiza la prueba de la operación registrarCliente utilizando SOAP UI. Esta operación permite registrar un nuevo cliente en el sistema proporcionando los datos requeridos como nombre, documento de identidad, correo electrónico.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Ejecución de la operación registrarCliente en SOAP UI

Se prueba la operación registrarCompra en SOAP UI, la cual permite registrar una nueva compra de boleto por parte de un cliente. Al enviar los datos requeridos, el servicio debe confirmar la transacción correctamente.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Prueba de la operación registrarCompra en SOAP UI con confirmación de la transacción realizada

Se realiza la prueba de la operación registrarUsuario en SOAP UI, mediante el envío de un mensaje SOAP que incluye el ID del cliente, nombre de usuario y clave. La respuesta debe confirmar el registro exitoso del nuevo usuario en el sistema.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Ejecución de la operación registrarUsuario en SOAP UI con envío de credenciales para el registro

## CLIENTE ESCRITORIO JAVA SOAP

### Creación del cliente para escritorio

Para iniciar el desarrollo del Cliente Escritorio, en la barra superior de NetBeans seleccionar “File” y luego “New Project”. Elegir la categoría “Java with Ant” y el tipo de proyecto “Java Application”.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Creación de un nuevo proyecto para el cliente desktop como Java Application en NetBeans

En la ventana emergente, asignar el nombre “VIAJECITOS\_SOAP\_CLIESC\_GR10” al proyecto en el campo “Project Name”. Una vez creado, establecer una estructura adecuada implementando el patrón arquitectónico MVC. Para ello, crear en “Source Packages” los siguientes paquetes: ec.edu.monster.controlador ec.edu.monster.vista ec.edu.monster.modelo ec.edu.monster.prueba ec.edu.monster.servicio

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Configuración del nombre y creación de paquetes para la estructura MVC del Cliente Escritorio

Para integrar el servicio web, hacer clic derecho en “Source Packages” y seleccionar “New” seguido de “Web Service Client”. En la ventana emergente, elegir la opción “WSDL URL” e ingresar la URL http://localhost:8080/VIAJECITOS\_SOAP\_GR10/WSAerolineasCondor?wsdl. En el campo de paquete, especificar ec.edu.monster.ws

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

1. Creación del Web Service Client con la URL del WSDL

En el paquete “ec.edu.monster.servicio”, crear una nueva clase llamada “ViajecitosService”. Hacer clic derecho sobre la clase, seleccionar “Insert Code” y luego “Call Web Service Operation”. A continuación, agregar todas las operaciones correspondientes del servicio web.

1. Creación de ViajecitosService y adición de operaciones del Web Service

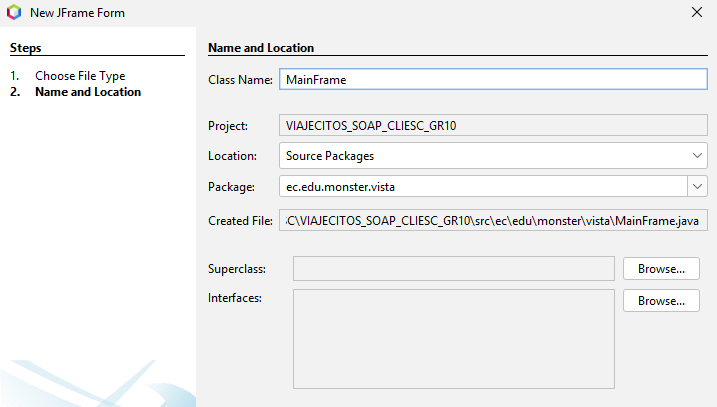
|  |
| --- |
| a package ec.edu.monster.servicio;  import ec.edu.monster.ws.Cliente;  import ec.edu.monster.ws.Compra;  import ec.edu.monster.ws.Usuario;  import ec.edu.monster.ws.Vuelo;  import ec.edu.monster.ws.WSAerolineasCondor;  import ec.edu.monster.ws.WSAerolineasCondor\_Service;  import java.util.List;  import javax.xml.datatype.XMLGregorianCalendar;  /\*\*  \* Service class to interact with the WSAerolineasCondor web service for Viajecitos SA.  \*  \* @author MATIAS  \*/  public class ViajecitosService {  private final WSAerolineasCondor port;  public ViajecitosService() {  WSAerolineasCondor\_Service service = new WSAerolineasCondor\_Service();  this.port = service.getWSAerolineasCondorPort();  }  /\*\*  \* Obtiene el vuelo más caro para un origen, destino y fecha específicos.  \*  \* @param ciudadOrigen Ciudad de origen.  \* @param ciudadDestino Ciudad de destino.  \* @param fecha Fecha del viaje.  \* @return Vuelo más caro.  \*/  public Vuelo obtenerVueloMasCaro(String ciudadOrigen, String ciudadDestino, XMLGregorianCalendar fecha) {  return port.obtenerVueloMasCaro(ciudadOrigen, ciudadDestino, fecha);  }  /\*\*  \* Inicia sesión para un usuario.  \*  \* @param nombreUsuario Nombre de usuario.  \* @param claveUsuario Contraseña del usuario.  \* @return Objeto Usuario si la autenticación es exitosa, null si falla.  \*/  public Usuario iniciarSesion(String nombreUsuario, String claveUsuario) {  return port.iniciarSesion(nombreUsuario, claveUsuario);  }  /\*\*  \* Obtiene las compras de un cliente.  \*  \* @param idCliente ID del cliente.  \* @return Lista de compras.  \*/  public List<Compra> obtenerComprasCliente(int idCliente) {  return port.obtenerComprasCliente(idCliente);  }  /\*\*  \* Busca vuelos disponibles.  \*  \* @param ciudadOrigen Ciudad de origen.  \* @param ciudadDestino Ciudad de destino.  \* @param fecha Fecha del viaje.  \* @return Lista de vuelos disponibles.  \*/  public List<Vuelo> buscarVuelos(String ciudadOrigen, String ciudadDestino, XMLGregorianCalendar fecha) {  return port.buscarVuelos(ciudadOrigen, ciudadDestino, fecha);  }  /\*\*  \* Registra un nuevo usuario.  \*  \* @param idCliente ID del cliente asociado.  \* @param nombreUsuario Nombre de usuario.  \* @param claveUsuario Contraseña del usuario.  \* @return Objeto Usuario registrado.  \*/  public Usuario registrarUsuario(int idCliente, String nombreUsuario, String claveUsuario) {  return port.registrarUsuario(idCliente, nombreUsuario, claveUsuario);  }  /\*\*  \* Registra una compra de boleto.  \*  \* @param idVuelo ID del vuelo.  \* @param idCliente ID del cliente.  \* @return ID de la compra registrada.  \*/  public int registrarCompra(int idVuelo, int idCliente) {  return port.registrarCompra(idVuelo, idCliente);  }  /\*\*  \* Registra un nuevo cliente.  \*  \* @param nombre Nombre del cliente.  \* @param email Email del cliente.  \* @param documentoIdentidad Documento de identidad.  \* @return Objeto Cliente registrado.  \*/  public Cliente registrarCliente(String nombre, String email, String documentoIdentidad) {  return port.registrarCliente(nombre, email, documentoIdentidad);  }  } |

Con los servicios creados, en el paquete “ec.edu.monster.controlador” crear una nueva clase llamada “ViajecitosController”. Implementar en esta clase los métodos correspondiente servicio de “ViajecitosService”

1. Creación e implementación de métodos en ViajecitosController

|  |
| --- |
| a package ec.edu.monster.controlador;  import ec.edu.monster.servicio.ViajecitosService;  import ec.edu.monster.ws.Cliente;  import ec.edu.monster.ws.Compra;  import ec.edu.monster.ws.Usuario;  import ec.edu.monster.ws.Vuelo;  import java.text.SimpleDateFormat;  import java.util.Date;  import java.util.GregorianCalendar;  import java.util.List;  import javax.xml.datatype.DatatypeFactory;  import javax.xml.datatype.XMLGregorianCalendar;  /\*\*  \* Controller to coordinate between console input and service layer for Viajecitos SA.  \*  \* @author MATIAS  \*/  public class ViajecitosController {  private final ViajecitosService service;  private Usuario usuarioAutenticado;  public ViajecitosController() {  this.service = new ViajecitosService();  }  /\*\*  \* Inicia sesión para un usuario.  \*  \* @param nombreUsuario Nombre de usuario.  \* @param claveUsuario Contraseña del usuario.  \* @return true si la autenticación es exitosa, false si falla.  \* @throws Exception si ocurre un error en el proceso.  \*/  public boolean iniciarSesion(String nombreUsuario, String claveUsuario) throws Exception {  if (nombreUsuario == null || nombreUsuario.trim().isEmpty() || claveUsuario == null || claveUsuario.trim().isEmpty()) {  throw new Exception("Usuario y contraseña son requeridos.");  }  usuarioAutenticado = service.iniciarSesion(nombreUsuario, claveUsuario);  return usuarioAutenticado != null;  }  /\*\*  \* Obtiene el ID del cliente autenticado.  \*  \* @return ID del cliente.  \* @throws Exception si no hay usuario autenticado.  \*/  public int getIdClienteAutenticado() throws Exception {  if (usuarioAutenticado == null) {  throw new Exception("No hay usuario autenticado.");  }  return usuarioAutenticado.getIdCliente();  }  /\*\*  \* Busca vuelos disponibles.  \*  \* @param ciudadOrigen Ciudad de origen.  \* @param ciudadDestino Ciudad de destino.  \* @param fechaStr Fecha del viaje (formato yyyy-MM-dd).  \* @return Lista de vuelos disponibles.  \* @throws Exception si ocurre un error o no se encuentran vuelos.  \*/  public List<Vuelo> buscarVuelos(String ciudadOrigen, String ciudadDestino, String fechaStr) throws Exception {  if (ciudadOrigen == null || ciudadOrigen.trim().isEmpty() || ciudadDestino == null || ciudadDestino.trim().isEmpty()) {  throw new Exception("Ciudad de origen y destino son requeridas.");  }  if (fechaStr == null || fechaStr.trim().isEmpty()) {  throw new Exception("Fecha de viaje es requerida.");  }  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");  Date fecha = sdf.parse(fechaStr);  GregorianCalendar gcal = new GregorianCalendar();  gcal.setTime(fecha);  XMLGregorianCalendar xmlFecha = DatatypeFactory.newInstance().newXMLGregorianCalendar(gcal);  List<Vuelo> vuelos = service.buscarVuelos(ciudadOrigen, ciudadDestino, xmlFecha);  if (vuelos.isEmpty()) {  throw new Exception("No se encontraron vuelos disponibles.");  }  return vuelos;  }  /\*\*  \* Obtiene el vuelo más caro.  \*  \* @param ciudadOrigen Ciudad de origen.  \* @param ciudadDestino Ciudad de destino.  \* @param fechaStr Fecha del viaje (formato yyyy-MM-dd).  \* @return Vuelo más caro.  \* @throws Exception si ocurre un error o no se encuentra el vuelo.  \*/  public Vuelo obtenerVueloMasCaro(String ciudadOrigen, String ciudadDestino, String fechaStr) throws Exception {  if (ciudadOrigen == null || ciudadOrigen.trim().isEmpty() || ciudadDestino == null || ciudadDestino.trim().isEmpty()) {  throw new Exception("Ciudad de origen y destino son requeridas.");  }  if (fechaStr == null || fechaStr.trim().isEmpty()) {  throw new Exception("Fecha de viaje es requerida.");  }  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");  Date fecha = sdf.parse(fechaStr);  GregorianCalendar gcal = new GregorianCalendar();  gcal.setTime(fecha);  XMLGregorianCalendar xmlFecha = DatatypeFactory.newInstance().newXMLGregorianCalendar(gcal);  Vuelo vuelo = service.obtenerVueloMasCaro(ciudadOrigen, ciudadDestino, xmlFecha);  if (vuelo == null) {  throw new Exception("No se encontró un vuelo para los criterios especificados.");  }  return vuelo;  }  /\*\*  \* Realiza la compra de un boleto.  \*  \* @param idVuelo ID del vuelo.  \* @param idCliente ID del cliente.  \* @return true si la compra es exitosa, false si falla.  \* @throws Exception si ocurre un error.  \*/  public boolean registrarCompra(int idVuelo, int idCliente) throws Exception {  if (idVuelo <= 0) {  throw new Exception("ID de vuelo inválido.");  }  if (idCliente <= 0) {  throw new Exception("ID de cliente inválido.");  }  int idCompra = service.registrarCompra(idVuelo, idCliente);  return idCompra > 0;  }  /\*\*  \* Consulta las compras de un cliente.  \*  \* @param idCliente ID del cliente.  \* @return Lista de compras.  \* @throws Exception si ocurre un error o no se encuentran compras.  \*/  public List<Compra> obtenerComprasCliente(int idCliente) throws Exception {  if (idCliente <= 0) {  throw new Exception("ID de cliente inválido.");  }  List<Compra> compras = service.obtenerComprasCliente(idCliente);  if (compras.isEmpty()) {  throw new Exception("No se encontraron compras para el cliente.");  }  return compras;  }  /\*\*  \* Registra un nuevo cliente.  \*  \* @param nombre Nombre del cliente.  \* @param email Email del cliente.  \* @param documentoIdentidad Documento de identidad.  \* @return ID del cliente registrado.  \* @throws Exception si ocurre un error.  \*/  public int registrarCliente(String nombre, String email, String documentoIdentidad) throws Exception {  if (nombre == null || nombre.trim().isEmpty() || email == null || email.trim().isEmpty() || documentoIdentidad == null || documentoIdentidad.trim().isEmpty()) {  throw new Exception("Todos los campos son requeridos.");  }  Cliente cliente = service.registrarCliente(nombre, email, documentoIdentidad);  if (cliente == null) {  throw new Exception("Error al registrar el cliente.");  }  return cliente.getIdCliente();  }  /\*\*  \* Registra un nuevo usuario.  \*  \* @param idCliente ID del cliente asociado.  \* @param nombreUsuario Nombre de usuario.  \* @param claveUsuario Contraseña del usuario.  \* @return true si el registro es exitoso, false si falla.  \* @throws Exception si ocurre un error.  \*/  public boolean registrarUsuario(int idCliente, String nombreUsuario, String claveUsuario) throws Exception {  if (idCliente <= 0) {  throw new Exception("ID de cliente inválido.");  }  if (nombreUsuario == null || nombreUsuario.trim().isEmpty() || claveUsuario == null || claveUsuario.trim().isEmpty()) {  throw new Exception("Nombre de usuario y contraseña son requeridos.");  }  Usuario usuario = service.registrarUsuario(idCliente, nombreUsuario, claveUsuario);  return usuario != null;  }  } |

Para crear la interfaz del inicio, hacer clic derecho en el paquete “ec.edu.monster.vista”, seleccionar “New” y luego “JFrame Form”. Nombrar el formulario “MainFrame”. Una vez creado, diseñar la interfaz gráfica correspondiente.



1. Creación y diseño de la interfaz LoginVista

La interfaz principal “MainFrame” se diseñó con colores pastel celestes para reflejar la imagen de Viajecitos SA. Incluye opciones para buscar vuelos, ver compras, iniciar sesión y registrarse, facilitando la navegación del usuario.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Diseño de la interfaz “MainFrame” con estilo pastel celeste y opciones principales para el Cliente Escritorio

Para crear la interfaz de inicio de sesión, hacer clic derecho en el paquete **“ec.edu.monster.vista”**, seleccionar **“New”** y luego **“JFrame Form”**. Nombrar el formulario como **“LoginFrame”** y diseñar su interfaz gráfica según los requerimientos.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

1. Creación del formulario “LoginFrame” en NetBeans para la interfaz de inicio de sesión

La interfaz de “LoginFrame” se diseña con un estilo limpio y sencillo, adecuado para facilitar el proceso de inicio de sesión del usuario.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Diseño de la interfaz “LoginFrame” para el inicio de sesión en el Cliente Escritorio

La interfaz de **“SearchFlightsFrame”** se diseñó con un estilo limpio y sencillo, facilitando la búsqueda del vuelo con mayor costo para el usuario.

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

1. Diseño de la interfaz “SearchFlightsFrame” para la búsqueda del vuelo más caro

La interfaz de **“RegisterFrame”** se diseña con un estilo limpio y sencillo, adecuado para facilitar el registro de nuevos usuarios en el sistema.

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

1. Diseño de la interfaz “RegisterFrame” para el registro de nuevos usuarios

La interfaz de “FlightResultFrame” permite mostrar los resultados de la búsqueda de vuelos y ofrece opciones para seleccionar y comprar el vuelo deseado de manera sencilla y clara.

Interfaz de usuario gráfica, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

1. Diseño de la interfaz “FlightResultFrame” para visualizar resultados y comprar vuelos

La interfaz de “PurchasesFrame” se diseña con un estilo claro y sencillo, permitiendo al usuario visualizar fácilmente el historial de compras realizadas.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Diseño de la interfaz “PurchasesFrame” para la visualización del historial de compras

## CLIENTE WEB JAVA SOAP

#### Creación del cliente web

Para comenzar el desarrollo del Cliente Web, en la barra superior de NetBeans seleccionar “File” y luego “New Project”. Elegir la categoría “Java with Maven” y el tipo de proyecto “Web Application”.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Creación de un nuevo proyecto Web Application con Maven en NetBeans

En la ventana emergente, nombrar el proyecto “VIAJECITOS\_SOAP\_CLIWEB\_GR10” y establecer el Group ID como “ec.edu.monster”.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

1. Configuración del nombre y Group ID del proyecto Cliente Web

En “Source Packages”, crear la estructura de paquetes según el patrón MVC, incluyendo los siguientes paquetes: ec.edu.monster.controlador ec.edu.monster.vista ec.edu.monster.modelo ec.edu.monster.prueba ec.edu.monster.servicio ec.edu.monster.ws

Texto

El contenido generado por IA puede ser incorrecto.

1. Creación de paquetes para la estructura MVC y el paquete ws en el Cliente Web

Actualizar el archivo pom.xml del proyecto para incluir el plugin y las dependencias necesarias que permitan consumir el servicio web y sus operaciones. A continuación, se presenta el contenido configurado:

1. Configuración del archivo pom.xml para el consumo del servicio web

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>ec.edu.monster</groupId>  <artifactId>VIAJECITOS\_SOAP\_CLIWEB\_GR10</artifactId>  <version>1.0-SNAPSHOT</version>  <packaging>war</packaging>  <name>VIAJECITOS\_SOAP\_CLIWEB\_GR10</name>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <jakartaee>10.0.0</jakartaee>  <cxf.version>4.0.3</cxf.version>  <maven.compiler.source>17</maven.compiler.source>  <maven.compiler.target>17</maven.compiler.target>  </properties>  <dependencies>  <!-- Jakarta EE Web API -->  <dependency>  <groupId>jakarta.platform</groupId>  <artifactId>jakarta.jakartaee-web-api</artifactId>  <version>${jakartaee}</version>  <scope>provided</scope>  </dependency>  <!-- JAX-WS API -->  <dependency>  <groupId>jakarta.xml.ws</groupId>  <artifactId>jakarta.xml.ws-api</artifactId>  <version>4.0.0</version>  </dependency>  <!-- JAX-WS Implementation (Metro) -->  <dependency>  <groupId>com.sun.xml.ws</groupId>  <artifactId>jaxws-rt</artifactId>  <version>4.0.0</version>  </dependency>  <!-- Apache CXF -->  <dependency>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-rt-frontend-jaxws</artifactId>  <version>${cxf.version}</version>  </dependency>  <dependency>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-rt-transports-http</artifactId>  <version>${cxf.version}</version>  </dependency>  <!-- Servlet API -->  <dependency>  <groupId>jakarta.servlet</groupId>  <artifactId>jakarta.servlet-api</artifactId>  <version>6.0.0</version>  <scope>provided</scope>  </dependency>  <!-- JSP API -->  <dependency>  <groupId>jakarta.servlet.jsp</groupId>  <artifactId>jakarta.servlet.jsp-api</artifactId>  <version>3.1.0</version>  <scope>provided</scope>  </dependency>  <!-- JSTL for JSPs -->  <dependency>  <groupId>jakarta.servlet.jsp.jstl</groupId>  <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>  <version>3.0.0</version>  </dependency>  <dependency>  <groupId>org.glassfish.web</groupId>  <artifactId>jakarta.servlet.jsp.jstl</artifactId>  <version>3.0.0</version>  </dependency>  </dependencies>  <build>  <finalName>VIAJECITOS\_SOAP\_CLIWEB\_GR10</finalName>  <plugins>  <!-- Maven Compiler Plugin -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.8.1</version>  <configuration>  <source>17</source>  <target>17</target>  </configuration>  </plugin>  <!-- Maven WAR Plugin -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-war-plugin</artifactId>  <version>3.3.2</version>  </plugin>  <!-- CXF Codegen Plugin for WSDL -->  <plugin>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-codegen-plugin</artifactId>  <version>${cxf.version}</version>  <executions>  <execution>  <id>generate-sources</id>  <phase>generate-sources</phase>  <configuration>  <sourceRoot>${project.basedir}/src/main/java</sourceRoot>  <wsdlOptions>  <wsdlOption>  <wsdl>http://localhost:8080/VIAJECITOS\_SOAP\_GR10/WSAerolineasCondor?wsdl</wsdl>  <extraargs>  <extraarg>-verbose</extraarg>  <extraarg>-p</extraarg>  <extraarg>ec.edu.monster.ws</extraarg>  <extraarg>-client</extraarg>  </extraargs>  </wsdlOption>  </wsdlOptions>  </configuration>  <goals>  <goal>wsdl2java</goal>  </goals>  </execution>  </executions>  <dependencies>  <dependency>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-rt-transports-http</artifactId>  <version>${cxf.version}</version>  </dependency>  </dependencies>  </plugin>  </plugins>  </build>  </project> |

En el paquete “ec.edu.monster.servicio”, crear una nueva clase llamada “ViajecitosService” e implementar los métodos necesarios para interactuar con las operaciones del servicio web.

Texto

El contenido generado por IA puede ser incorrecto.

1. Creación e implementación de métodos en ViajecitosService

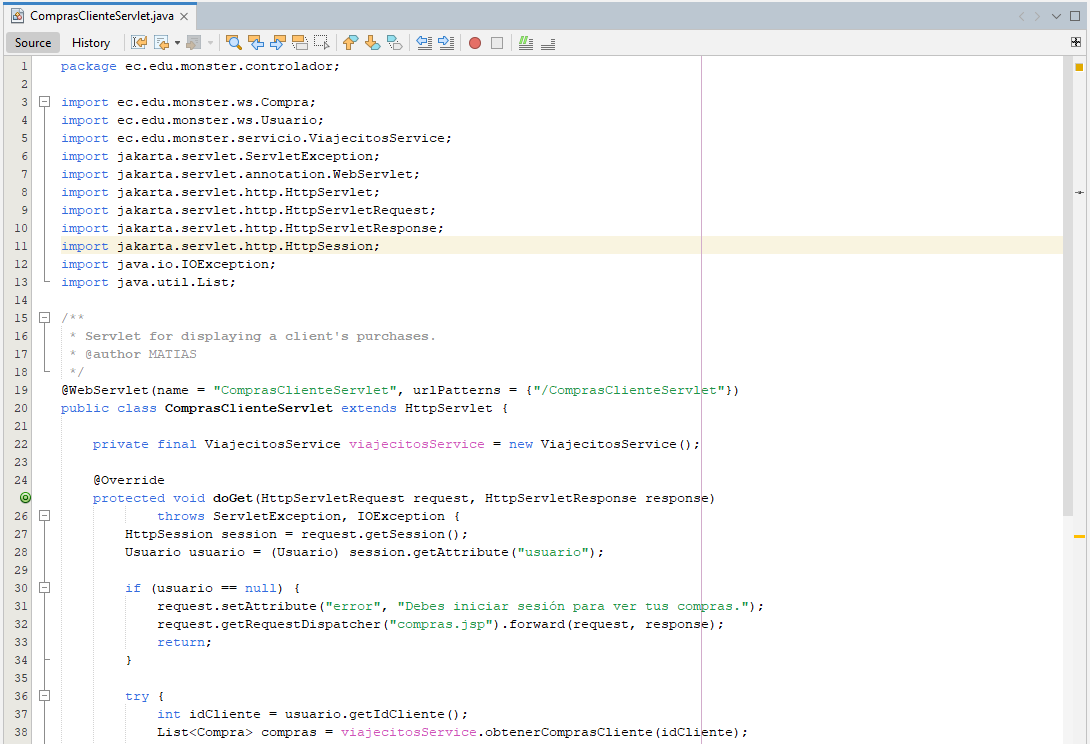
En el paquete **“ec.edu.monster.controlador”**, se crea un servlet llamado **CompraServlet** para manejar las funciones relacionadas con el consumo del servicio web.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

1. Creación de AppControlador en el paquete controlador

Dentro del paquete **“ec.edu.monster.controlador”**, se crea el servlet **ComprasClienteServlet**, encargado de gestionar la consulta de compras realizadas por un cliente a través del servicio web.



1. Creación del servlet ComprasClienteServlet para consultar compras del cliente mediante el servicio web

En el mismo paquete **“ec.edu.monster.controlador”**, se implementa el servlet **LoginServlet**, encargado de procesar las credenciales de acceso del usuario y validar su autenticación mediante el servicio web.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

1. Creación del servlet LoginServlet para el manejo del inicio de sesión en el Cliente Web

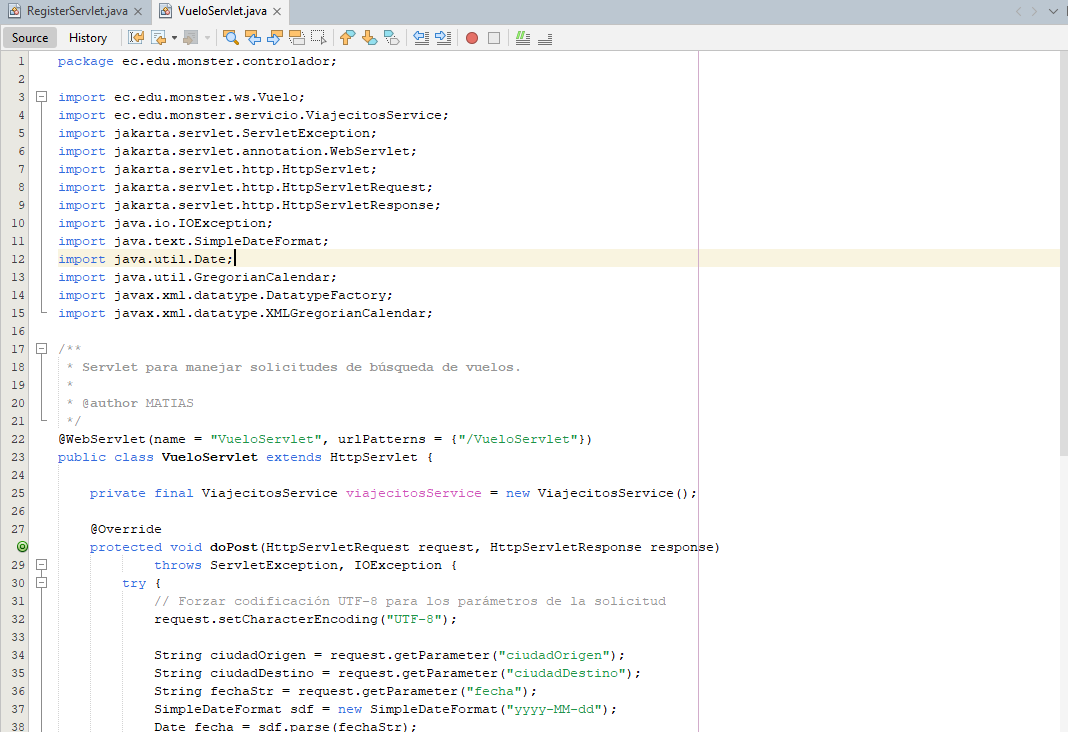
También en el paquete “ec.edu.monster.controlador”, se desarrolla el servlet RegisterServlet, responsable de gestionar el registro de nuevos usuarios enviando sus datos al servicio web.

Imagen que contiene Escala de tiempo

El contenido generado por IA puede ser incorrecto.

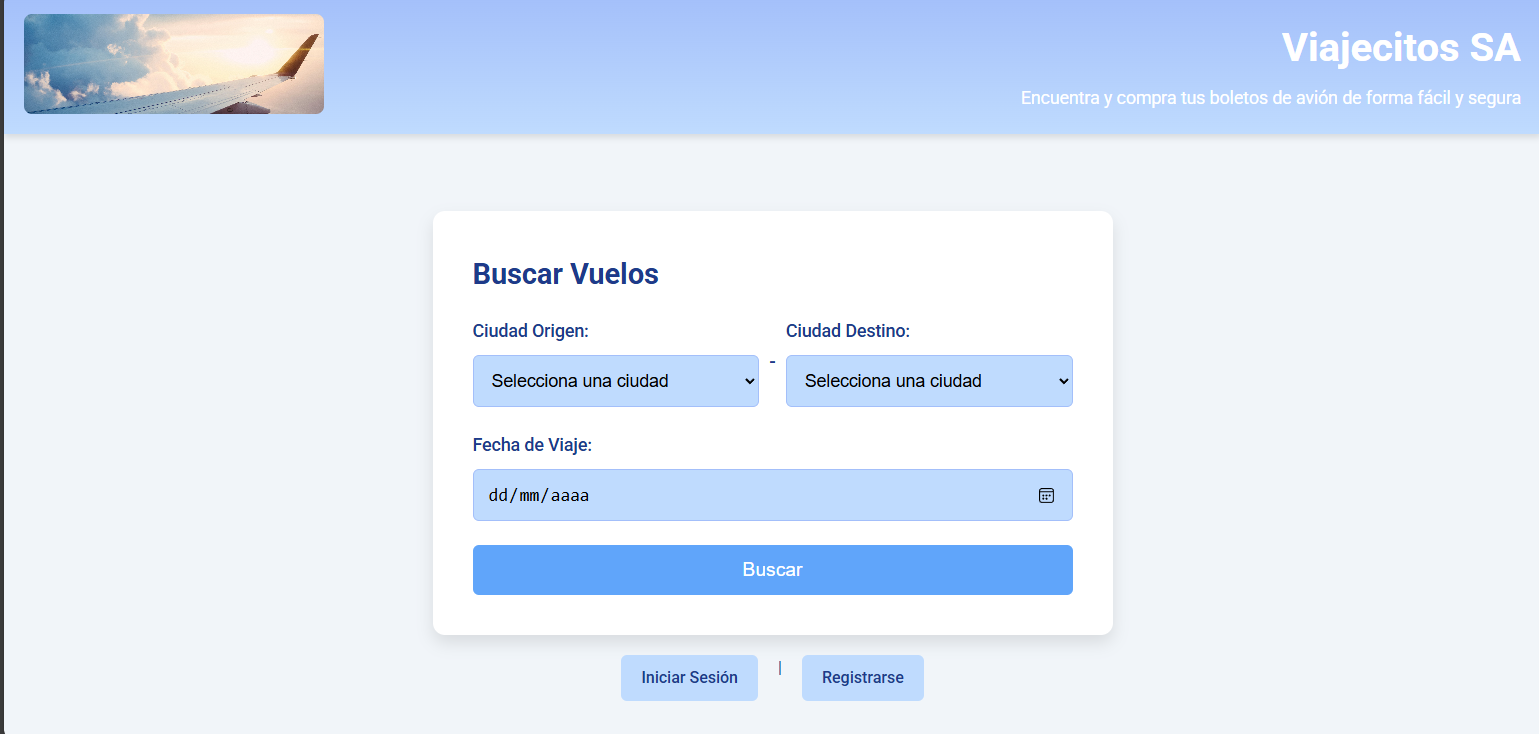
1. Creación del servlet RegisterServlet para el registro de nuevos usuarios desde el Cliente Web

Se implementa el servlet VueloServlet en el paquete “ec.edu.monster.controlador”, encargado de manejar las solicitudes de búsqueda de vuelos enviadas desde el Cliente Web, utilizando el servicio web correspondiente.



1. Creación del servlet VueloServlet para gestionar la búsqueda de vuelos en el Cliente Web

En la carpeta **“Web Pages”**, se crea un archivo JSP llamado **“index.jsp”** para diseñar la vista principal. Esta incluye una sección hero con estilos en tonos celestes y azules, permitiendo al usuario realizar la búsqueda de vuelos de forma intuitiva.



1. Diseño de la vista “index.jsp” con sección hero para la búsqueda de vuelos en el Cliente Web

Se crea el archivo “login.jsp” en la carpeta “Web Pages”, destinado a mostrar el formulario de inicio de sesión. La interfaz utiliza colores suaves y campos claros para que el usuario pueda ingresar sus credenciales de forma sencilla.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Diseño del formulario de inicio de sesión en la vista “login.jsp” del Cliente Web

En la carpeta “Web Pages”, se crea el archivo “register.jsp” que contiene el formulario de registro de nuevos usuarios. La vista mantiene un diseño limpio y amigable, con campos necesarios para completar el proceso de registro.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Formulario de registro de usuario en la vista “register.jsp” del Cliente Web

Se implementa la vista “compras.jsp” dentro de la carpeta “Web Pages”, destinada a mostrar al usuario el historial de compras realizadas. La información se presenta en una tabla organizada y con un diseño visual claro y accesible.

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

1. Visualización del historial de compras del cliente en la vista “compras.jsp”

Se crea la vista “result.jsp” para mostrar los resultados de la búsqueda de vuelos realizada por el usuario. La información se presenta en una tabla estilizada, permitiendo seleccionar y proceder a la compra del vuelo deseado.

Interfaz de usuario gráfica, Aplicación, Word

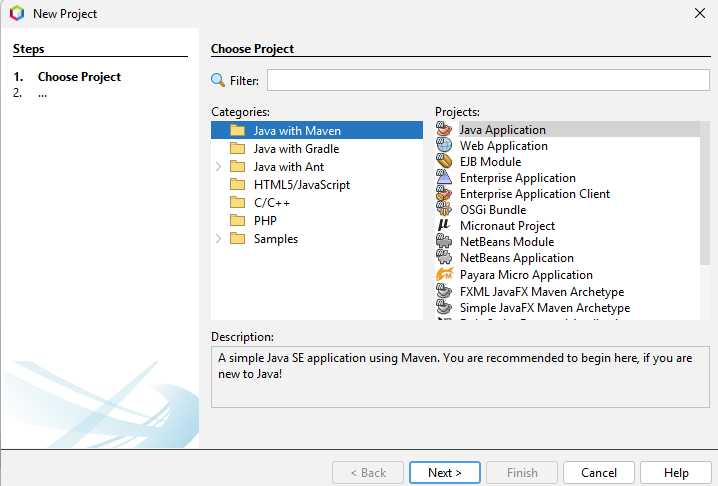
El contenido generado por IA puede ser incorrecto.

1. Resultados de búsqueda de vuelos mostrados en la vista “result.jsp”

## CLIENTE CONSOLA JAVA SOAP

#### Creación del cliente para consola

Para iniciar el desarrollo del Cliente de Consola, en la barra superior de NetBeans seleccionar “File” y luego “New Project”. Elegir la categoría “Java with Ant” y el tipo de proyecto “Java Application”.



1. Creación de un nuevo proyecto Java Application para el Cliente de Consola

En la ventana emergente, asignar el nombre “VIAJECITOS\_SOAP\_CLICON\_GR10” al proyecto en el campo “Project Name”.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

1. Configuración del nombre del proyecto VIAJECITOS\_SOAP\_CLICON\_GR10

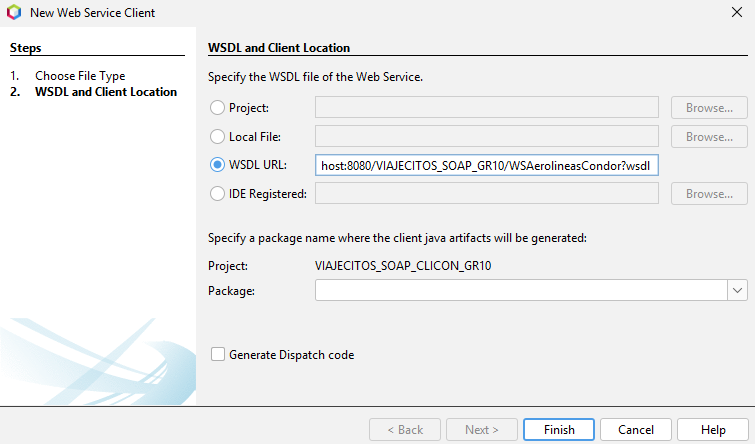
Crear la estructura de paquetes para el Cliente de Consola, incluyendo los siguientes paquetes en “Source Packages” según el patrón MVC adaptado, considerando que la vista será la clase Main: ec.edu.monster.controlador ec.edu.monster.servicio

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Creación de paquetes controlador y servicio para el Cliente de Consola

Para consumir el servicio web, hacer clic derecho en “Source Packages” y seleccionar “New” seguido de “Web Service Client”. En la ventana emergente, elegir la opción “WSDL URL” e ingresar la URL http://localhost:8080/VIAJECITOS\_SOAP\_GR10/WSAerolineasCondor?wsdl. Especificar el paquete como ec.edu.monster.



1. Creación del Web Service Client con la URL del WSDL

En el paquete **“ec.edu.monster.servicio”**, se crea la clase **“ViajecitosService”**, en la cual se implementan los métodos necesarios para interactuar con las distintas operaciones expuestas por el servicio web de la agencia.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

1. Implementación de la clase “ViajecitosService” para el consumo del servicio web en el Cliente Consola

En el paquete “ec.edu.monster.controlador”, crear una nueva clase llamada “ViajecitosController”. Implementar en esta clase los métodos correspondientes a los servicios definidos en “ViajecitosService” para gestionar la lógica de negocio del Cliente de Consola.

Texto

El contenido generado por IA puede ser incorrecto.

1. Creación e implementación de métodos en ViajecitosController

En el paquete **“vista”**, se crea la clase **MainConsole**, encargada de diseñar la interfaz de consola que permite al usuario interactuar con el servicio, mostrando opciones para buscar vuelos, consultar compras y realizar otras operaciones.

Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

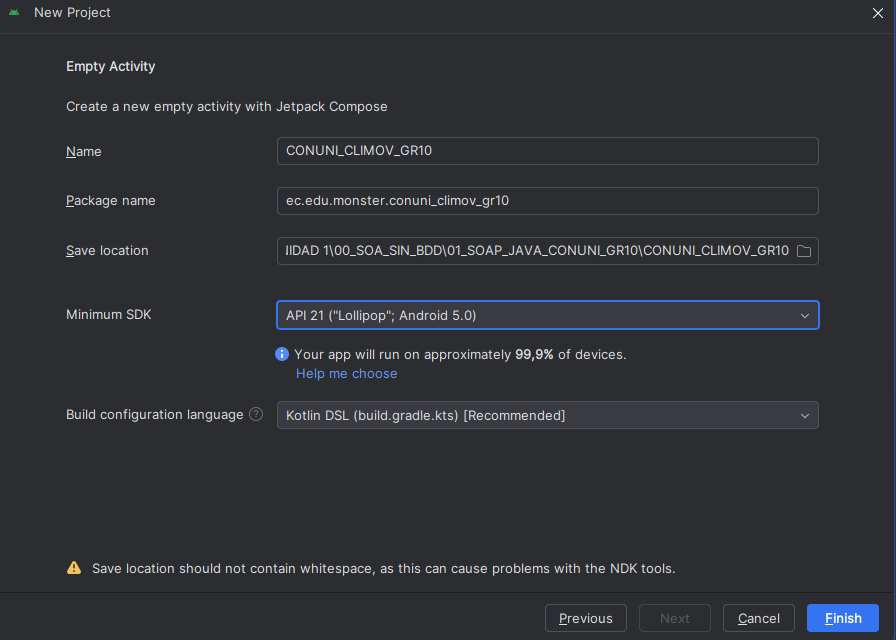
1. Diseño de la interfaz de consola en la clase MainConsole para el Cliente Consola

## CLIENTE MÓVIL JAVA SOAP

#### Creación del cliente móvil

**Crear un nuevo proyecto en Android Studio**  
Abrir Android Studio y seleccionar “File” > “New” > “New Project”. Escoger la plantilla “Empty Activity” y configurar el proyecto con los siguientes detalles:

* **Name**: VAIJECITOS\_SOAP\_GR10
* **Package name**: ec.edu.monster
* **Save location**: Seleccionar una carpeta adecuada.
* **Language**: Java
* **Minimum API**: API 21 (Lollipop) o superior para garantizar compatibilidad.  
  Hacer clic en “Finish” para crear el proyecto.



1. Creación del proyecto VIAJECITOS\_SOAP\_GR10 en Android Studio

Agregar dependencias para consumir el servicio SOAP, incorporar la biblioteca ksoap2-android.

1. Abrir el archivo build.gradle (nivel de módulo, ubicado en app/build.gradle) y añadir la siguiente dependencia en la sección dependencies:
2. Configuración de dependencias y permisos en el proyecto

|  |
| --- |
| *dependencies* **{** *implementation*(*libs*.*androidx*.*core*.*ktx*)  *implementation*(*libs*.*androidx*.*lifecycle*.*runtime*.*ktx*)  *implementation*(*libs*.*androidx*.*activity*.*compose*)  *implementation*(platform(*libs*.*androidx*.*compose*.*bom*))  *implementation*(*libs*.*androidx*.*ui*)  *implementation*(*libs*.*androidx*.*ui*.*graphics*)  *implementation*(*libs*.*androidx*.*ui*.*tooling*.*preview*)  *implementation*(*libs*.*androidx*.*material3*)  *implementation*("com.google.code.ksoap2-android:ksoap2-android:3.6.4")  *testImplementation*(*libs*.*junit*)  *androidTestImplementation*(*libs*.*androidx*.*junit*)  *androidTestImplementation*(*libs*.*androidx*.*espresso*.*core*)  *androidTestImplementation*(platform(*libs*.*androidx*.*compose*.*bom*))  *androidTestImplementation*(*libs*.*androidx*.*ui*.*test*.*junit4*)  *debugImplementation*(*libs*.*androidx*.*ui*.*tooling*)  *debugImplementation*(*libs*.*androidx*.*ui*.*test*.*manifest*)  **}** |

Agregar el permiso de acceso a internet en el archivo AndroidManifest.xml (ubicado en app/src/main/), colocando la siguiente línea antes de la etiqueta :

|  |
| --- |
| <uses-permission android:name="android.permission.INTERNET" /> |

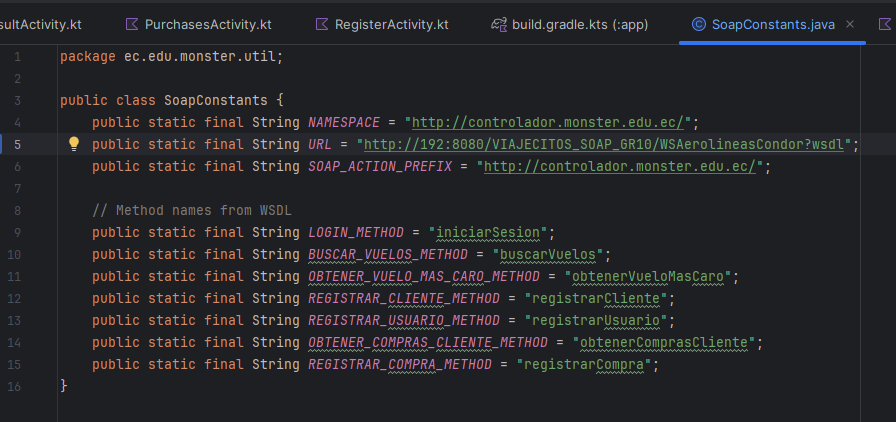
Actualizar el archivo settings.gradle.kts para incluir los repositorios necesarios

|  |
| --- |
| pluginManagement {  repositories {  google {  content {  includeGroupByRegex("com\\.android.\*")  includeGroupByRegex("com\\.google.\*")  includeGroupByRegex("androidx.\*")  }  }  mavenCentral()  gradlePluginPortal()  maven { url = uri("https://oss.sonatype.org/content/repositories/ksoap2-android-releases/") }  }  }  dependencyResolutionManagement {  repositoriesMode.set(RepositoriesMode.FAIL\_ON\_PROJECT\_REPOS)  repositories {  google()  mavenCentral()  maven { url = uri("https://oss.sonatype.org/content/repositories/ksoap2-android-releases/") }  }  }  rootProject.name = "VIAJECITOS\_SOAP\_GR10"  include(":app") |

Dado que Android no soporta JAX-WS directamente, se utilizará ksoap2-android para enviar solicitudes SOAP de forma dinámica, sin necesidad de generar clases a partir del WSDL.

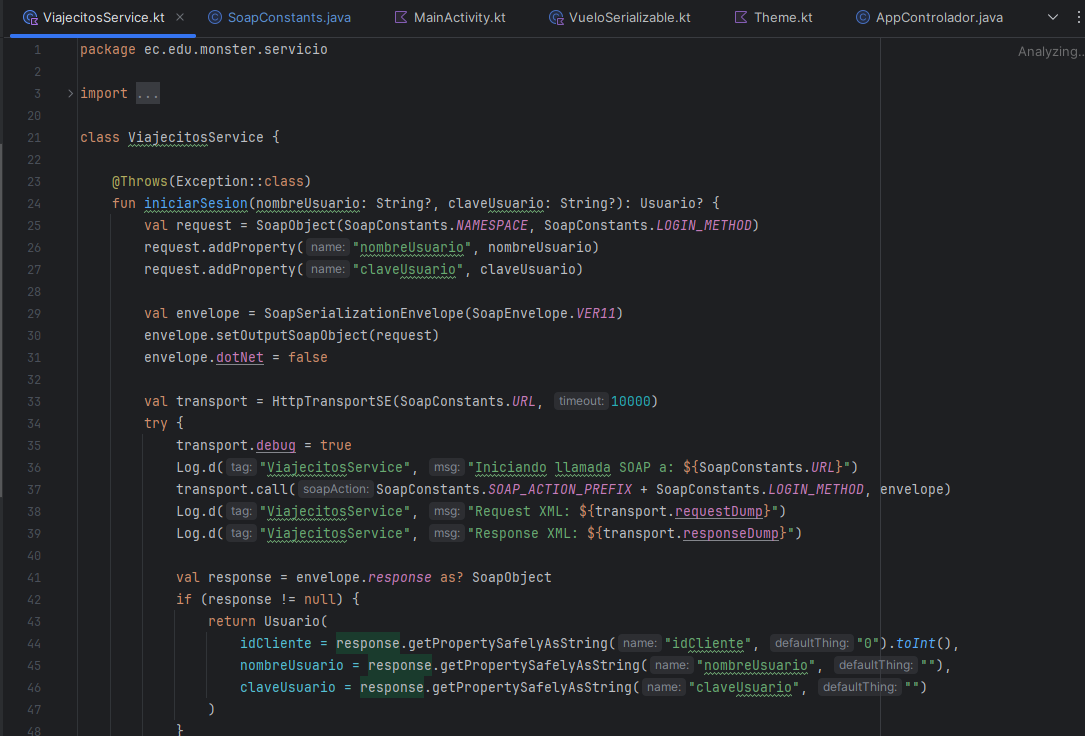
Definir las constantes del servicio SOAP:

1. Crear un nuevo paquete llamado ec.edu.monster.util en app/src/main/java.
2. Crear la clase SoapConstants.java con las constantes necesarias, como la URL del WSDL y los nombres de las operaciones.  
   **Nota**: Usar 10.0.2.2 para pruebas en emuladores; para dispositivos físicos, reemplazar por la IP de la máquina o una URL pública si el servidor está en producción.



1. Creación de la clase SoapConstants para las constantes del servicio SOAP

Implementar el servicio ViajecitosService



1. Integración del servicio ViajecitosService

El controlador actúa como intermediario entre la vista y el servicio SOAP.

1. Crear un paquete ec.edu.monster.controlador.
2. Crear la clase AppControlador.java e implementar los métodos necesarios para gestionar las operaciones del servicio

Texto

El contenido generado por IA puede ser incorrecto.

1. Creación e implementación de AppControlado

Se diseña el LoginActivity para permitir a los usuarios ingresar sus credenciales y autenticarse. La interfaz es sencilla, con campos claros y botones accesibles para facilitar el inicio de sesión.

Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.

1. Interfaz de LoginActivity para el inicio de sesión en la aplicación móvil

El MainActivity actúa como pantalla principal después del login, mostrando las opciones principales de la app como búsqueda de vuelos y acceso al historial de compras, con un diseño intuitivo y amigable.



1. Pantalla principal MainActivity con opciones principales para el usuario

El PurchasesActivity muestra un listado claro y ordenado de las compras realizadas por el usuario, permitiendo la fácil consulta y seguimiento de sus reservas.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

1. Visualización del historial de compras en PurchasesActivity

El ResultActivity presenta los resultados de búsqueda de vuelos con opciones para seleccionar y proceder a la compra, manteniendo un diseño claro y accesible.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

1. Pantalla de resultados de vuelos en ResultActivity para la selección y compra

## CONCLUSIONES

* La implementación de ambos proyectos permitió demostrar que es posible construir sistemas funcionales y paralelos utilizando diferentes tecnologías y protocolos (SOAP y REST), cumpliendo con los mismos requerimientos funcionales desde distintas plataformas.
* La separación entre el servidor (aerolínea) y los clientes (agencia) garantiza una arquitectura distribuida clara, donde la lógica de negocio se centraliza en los servicios web y las interfaces web se enfocan en la interacción con el usuario.
* El uso de servicios web permitió estandarizar la comunicación entre los módulos, facilitando la interoperabilidad entre lenguajes de programación, entornos de desarrollo y motores de base de datos heterogéneos.
* Las funcionalidades requeridas fueron cubiertas completamente, evidenciando el dominio técnico en el desarrollo, consumo y prueba de servicios web, así como en el manejo de datos mediante bases relacionales independientes.

## RECOMENDACIONES

Es importante asegurar que los servidores cuenten con una adecuada gestión de los servicios web, de modo que el controlador pueda procesar correctamente las solicitudes y generar las respuestas esperadas sin errores de lógica o conexión.

La creación y ejecución de cada cliente debe demostrar de forma clara y sincrónica cómo se consumen los servicios, evidenciando que, aunque un cliente realice una acción y otro consulte su resultado, el sistema mantiene la coherencia y consistencia de los datos.

Se recomienda reforzar el manejo de estados y respuestas entre los clientes y el servidor, para asegurar que cualquier operación realizada (como una compra) sea inmediatamente visible desde los demás módulos que dependan de esa información.

## BIBLIOGRAFÍA

* Fowler, M. (2019). Patterns of Enterprise Application Architecture. Addison-Wesley.
* McGovern, J., Tyagi, S., Stevens, M., & Mathew, S. (2003). Java Web Services Architecture. Morgan Kaufmann.
* Troelsen, A., & Japikse, P. (2021). Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming (10th ed.). Apress.
* Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures [Doctoral dissertation, University of California, Irvine]. ProQuest Dissertations Publishing.
* W3C. (2004). SOAP Version 1.2 Specification. World Wide Web Consortium (W3C). https://www.w3.org/TR/soap12/