

**Tema**

CONVERSIÓN DE UNIDADES SOAP JAVA SIN BDD

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU. CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**INTEGRANTES**

Bazurto Christopher & Drouet Stephen

**Fecha**

04/05/2025

Tabla de contenido

[INTRODUCCIÓN 6](#_Toc197272943)

[OBJETIVOS 6](#_Toc197272944)

[Objetivo General 6](#_Toc197272945)

[Objetivos Específicos 6](#_Toc197272946)

[MARCO TEÓRICO 6](#_Toc197272947)

[Arquitectura de Software 6](#_Toc197272948)

[Arquitectura Orientada a Servicios 7](#_Toc197272949)

[Protocolo SOAP 7](#_Toc197272950)

[HERRAMIENTAS 7](#_Toc197272951)

[Aplicación de Consola con Java 7](#_Toc197272952)

[Aplicación de Escritorio con Java 8](#_Toc197272953)

[Node.js con React para la Aplicación Web 8](#_Toc197272954)

[React Native para la Aplicación Móvil 8](#_Toc197272955)

[PARTE PRÁCTICA 8](#_Toc197272956)

[ESTRUCTURA DEL PROYECTO 8](#_Toc197272957)

[SERVIDOR 9](#_Toc197272958)

[CREACION DE LA APLICACIÓN 9](#_Toc197272959)

[MODELO MVC 10](#_Toc197272960)

[IMPLEMENTACIÓN 11](#_Toc197272961)

[HABILITAR CORS 12](#_Toc197272962)

[PRUEBAS UNITARIAS 13](#_Toc197272963)

[EJECUCIÓN DE LA APLICACIÓN 16](#_Toc197272964)

[CLIENTE DE CONSOLA 17](#_Toc197272965)

[CREACION DE LA APLICACIÓN 17](#_Toc197272966)

[MODELO MVC 17](#_Toc197272967)

[IMPLEMENTACIÓN 18](#_Toc197272968)

[EJECUCIÓN DE LA APLICACIÓN 21](#_Toc197272969)

[CLIENTE DE ESCRITORIO 22](#_Toc197272970)

[CREACION DE LA APLICACIÓN 22](#_Toc197272971)

[MODELO MVC 23](#_Toc197272972)

[IMPLEMENTACIÓN 24](#_Toc197272973)

[EJECUCIÓN DE LA APLICACIÓN 26](#_Toc197272974)

[CLIENTE WEB 27](#_Toc197272975)

[CREACION DE LA APLICACIÓN 27](#_Toc197272976)

[MODELO MVC 27](#_Toc197272977)

[IMPLEMENTACIÓN 28](#_Toc197272978)

[EJECUCIÓN DE LA APLICACIÓN 34](#_Toc197272979)

[CLIENTE MÓVIL 35](#_Toc197272980)

[CREACION DE LA APLICACIÓN 35](#_Toc197272981)

[MODELO MVC 36](#_Toc197272982)

[IMPLEMENTACIÓN 36](#_Toc197272983)

[EJECUCIÓN DE LA APLICACIÓN 41](#_Toc197272984)

[CONCLUSIONES 43](#_Toc197272985)

[RECOMENDACIONES 43](#_Toc197272986)

[BIBLIOGRAFÍA 43](#_Toc197272987)

**ÍNDICE DE IMÁGENES**

[Figura 1. Arquitectura de Software 6](#_Toc197272988)

[Figura 2. Arquitectura de Orientada a Servicios 7](#_Toc197272989)

[Figura 3. SOAP 7](#_Toc197272990)

[Figura 5. Estructura del proyecto carpeta raíz 8](#_Toc197272991)

[Figura 6. Estructura del proyecto carpetas internas 9](#_Toc197272992)

[Figura 7. Crear Proyecto en NetBeans 9](#_Toc197272993)

[Figura 8. Configuración del Proyecto en NetBeans 10](#_Toc197272994)

[Figura 9. Creación de un nuevo paquete 10](#_Toc197272995)

[Figura 10. Estructura MVC del proyecto 11](#_Toc197272996)

[Figura 11. Creación de pruebas unitarias 14](#_Toc197272997)

[Figura 12. Ejecución de pruebas unitarias 16](#_Toc197272998)

[Figura 13. Ejecución del servidor payara 17](#_Toc197272999)

[Figura 14. Configuración de la aplicación de consola 17](#_Toc197273000)

[Figura 15. Creación de un nuevo paquete en cliente de consola 18](#_Toc197273001)

[Figura 16. Estructura MVC del proyecto de consola 18](#_Toc197273002)

[Figura 17. Creación del cliente del Web Service 19](#_Toc197273003)

[Figura 18. Creación del cliente del Web Service archivos necesarios 19](#_Toc197273004)

[Figura 19. Creación de funciones del Web Service en el lado del cliente 20](#_Toc197273005)

[Figura 20. Resolver error de archivos faltantes 21](#_Toc197273006)

[Figura 21. Login en aplicación de consola. 22](#_Toc197273007)

[Figura 22. Funciones en aplicación de consola. 22](#_Toc197273008)

[Figura 23. Configuración de la aplicación de escritorio. 23](#_Toc197273009)

[Figura 24. Creación de un nuevo paquete en cliente de escritorio 23](#_Toc197273010)

[Figura 25. Estructura MVC del proyecto de escritorio 24](#_Toc197273011)

[Figura 26. Crear formularios en aplicación de escritorio 24](#_Toc197273012)

[Figura 27. Crear formularios de inicio de sesión en aplicación de escritorio 25](#_Toc197273013)

[Figura 28. Crear formularios de menú en aplicación de escritorio 25](#_Toc197273014)

[Figura 29. Login en aplicación de escritorio. 26](#_Toc197273015)

[Figura 30. Funciones en aplicación de escritorio 27](#_Toc197273016)

[Figura 31. Crear proyecto base de la aplicación web 27](#_Toc197273017)

[Figura 32. Crear modelo MVC en el proyecto de la aplicación web 28](#_Toc197273018)

[Figura 33. Crear modelo User en el proyecto de la aplicación web 28](#_Toc197273019)

[Figura 34. Login en aplicación web. 34](#_Toc197273020)

[Figura 35. Funciones en aplicación de web 35](#_Toc197273021)

[Figura 36. Creación y configuración del proyecto móvil 35](#_Toc197273022)

[Figura 37. Crear modelo MVC en el proyecto de la aplicación móvil 36](#_Toc197273023)

[Figura 38. Login en aplicación móvil. 42](#_Toc197273024)

[Figura 39. Funciones en aplicación de móvil 42](#_Toc197273025)

**ÍNDICE DE TABLAS**

[Tabla 1**. Código ConversionUnidades** 11](#_Toc197273047)

[Tabla 2. **Código del modelo User.** 11](#_Toc197273048)

[Tabla 3. **Código del controlador Login.** 12](#_Toc197273049)

[Tabla 4. **Habilitar CORS en el servidor.** 12](#_Toc197273050)

[Tabla 5. **Código de prueba unitaria LoginTest.** 14](#_Toc197273051)

[Tabla 6. **Código de prueba unitaria ConversionUnidadesTest.** 15](#_Toc197273052)

[Tabla 7. **Código de vista de consola ConsoleView.** 20](#_Toc197273053)

[Tabla 8. **Código del modelo User en cliente consola.** 21](#_Toc197273054)

[Tabla 9. **Código de eventos para la aplicación de escritorio.** 25](#_Toc197273055)

[Tabla 10. **Modelo User en aplicación web.** 28](#_Toc197273056)

[Tabla 11. **Código ClientController.jsx** 29](#_Toc197273057)

[Tabla 12. **Código ConversionUnidadesController.jsx** 29](#_Toc197273058)

[Tabla 13. **Código de la vista Login.jsx** 30](#_Toc197273059)

[Tabla 14. **Código de la vista Menu.jsx** 31](#_Toc197273060)

[Tabla 15. **Código App.jsx** 33](#_Toc197273061)

[Tabla 16. **Estilo de la aplicación web styles.css** 33](#_Toc197273062)

[Tabla 17. **Código LoginController.js para aplicación móvil** 36](#_Toc197273063)

[Tabla 18. **Código ConversionUnidadesController.js para aplicación móvil** 37](#_Toc197273064)

[Tabla 19. **Código de la vista LoginScreen.jsx de la aplicación móvil** 38](#_Toc197273065)

[Tabla 20. **Código de la vista MenuScreen.jsx de la aplicación móvil** 40](#_Toc197273066)

[Tabla 21. **Código index.tsx de la aplicación móvil** 41](#_Toc197273067)

**INTRODUCCIÓN A LA ARQUITECTURA DE SOFTWARE**

# INTRODUCCIÓN

En el ámbito del desarrollo de software, la implementación de servicios web permite la interoperabilidad entre diferentes aplicaciones y plataformas. El protocolo SOAP (Simple Object Access Protocol) es una de las tecnologías que facilita esta comunicación estructurada y estandarizada. Este informe detalla la propuesta para desarrollar un sistema de conversión de unidades utilizando servicios web SOAP en Java, sin la necesidad de una base de datos, lo que simplifica la arquitectura y mejora el rendimiento en aplicaciones específicas, así como el consumo de este atreves de clientes de consola, escritorio, web y móvil.

# OBJETIVOS

## Objetivo General

* Diseñar e implementar un sistema de conversión de unidades basado en servicios web SOAP utilizando Java y varios clientes, sin la dependencia de una base de datos.

## Objetivos Específicos

* Desarrollar e implementar un servidor en Java que exponga servicios web SOAP para la conversión de unidades.
* Diseñar e implementar clientes en plataformas de consola, escritorio, web y móvil que consuman los servicios ofrecidos por el servidor.
* Ejecutar el sistema completo y realizar pruebas funcionales para verificar la correcta interacción entre el servidor y los distintos tipos de clientes.

# MARCO TEÓRICO

## Arquitectura de Software

La arquitectura de software se refiere a la estructura organizativa y conceptual que subyace en el diseño y desarrollo de sistemas de software. Es el proceso de definir una solución estructural que cumple con todos los requisitos funcionales y no funcionales, como la escalabilidad, el rendimiento y la seguridad. Su importancia radica en su capacidad para proporcionar un marco de trabajo que guía las decisiones de diseño y desarrollo.

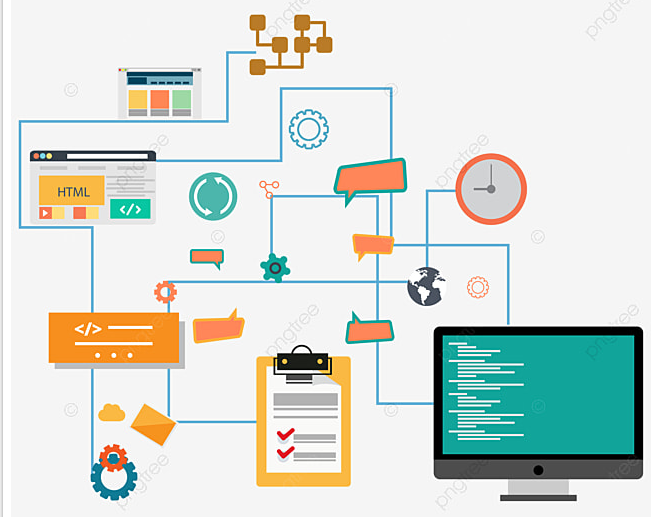


Figura 1. Arquitectura de Software

## Arquitectura Orientada a Servicios

La Arquitectura Orientada a Servicios es un enfoque moderno y flexible para el diseño de aplicaciones. En lugar de construir una sola aplicación monolítica, se descompone en pequeños servicios independientes. Cada servicio se centra en una tarea específica y se comunica con otros servicios a través de interfaces bien definidas.



Figura 2. Arquitectura de Orientada a Servicios

## Protocolo SOAP

SOAP (Simple Object Access Protocol) es un protocolo ligero para el intercambio de información en entornos descentralizados y distribuidos. Los mensajes SOAP son las transmisiones de información de remitentes a destinatarios y se pueden combinar para crear patrones de petición/respuesta. SOAP se transporta principalmente a través de HTTP y sistemas de mensajería de middleware, pero también puede utilizar otros protocolos como SMTP y FTP.

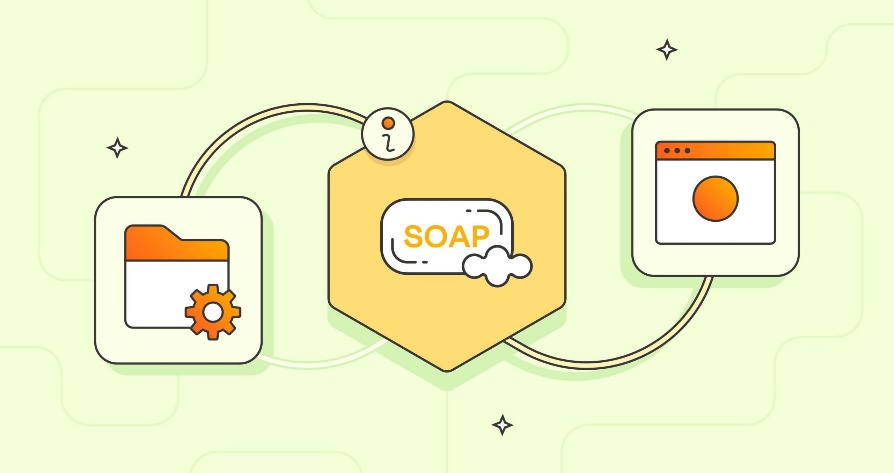


Figura 3. SOAP

# HERRAMIENTAS

## Aplicación de Consola con Java

Las aplicaciones de consola en Java son programas que se ejecutan en la línea de comandos, ideales para tareas automatizadas o de administración. Java proporciona una sintaxis clara y una amplia biblioteca estándar que facilita el desarrollo de este tipo de aplicaciones.

## Aplicación de Escritorio con Java

Para el desarrollo de aplicaciones de escritorio, Java ofrece bibliotecas como Swing y JavaFX. Swing es una biblioteca más antigua que proporciona componentes gráficos básicos, mientras que JavaFX es una plataforma más moderna que permite la creación de interfaces de usuario ricas y estilizadas, con soporte para multimedia y gráficos avanzados. JavaFX es especialmente adecuado para aplicaciones que requieren una interfaz de usuario atractiva y dinámica.

## Node.js con React para la Aplicación Web

Node.js es un entorno de ejecución para JavaScript que permite el desarrollo del lado del servidor. Su arquitectura orientada a eventos y su modelo de entrada/salida no bloqueante lo hacen ideal para aplicaciones web escalables y en tiempo real. React es una biblioteca de JavaScript desarrollada por Meta (anteriormente Facebook) para la construcción de interfaces de usuario basadas en componentes. La combinación de Node.js y React permite el desarrollo de aplicaciones web modernas, con una separación clara entre el backend y el frontend, y facilita la creación de interfaces de usuario reactivas y eficientes.

## React Native para la Aplicación Móvil

React Native es un framework de código abierto desarrollado por Meta que permite el desarrollo de aplicaciones móviles nativas para iOS y Android utilizando JavaScript y React. Ofrece la ventaja de compartir una base de código entre plataformas, reduciendo el tiempo y los recursos necesarios para el desarrollo. React Native proporciona componentes nativos y acceso a las funcionalidades del dispositivo, lo que resulta en aplicaciones con rendimiento cercano al nativo y una experiencia de usuario consistente.

# PARTE PRÁCTICA

## ESTRUCTURA DEL PROYECTO

Se procede a la creación de la estructura del proyecto, la cual es organizada mediante la generación de las carpetas correspondientes. Estas carpetas son definidas de acuerdo con los módulos, funcionalidades o capas que compondrán la aplicación, permitiendo así una disposición clara y mantenible del código fuente desde las etapas iniciales del desarrollo.

Se crea la carpeta raíz la cual contiene todo el proyecto.

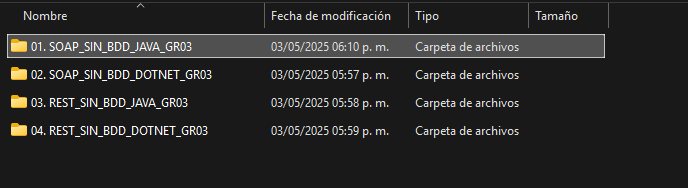


Figura 5. Estructura del proyecto carpeta raíz

Se crean las carpetas internas las cuales dividen el proyecto en el lado del servidor, los clientes a usar y la documentación.

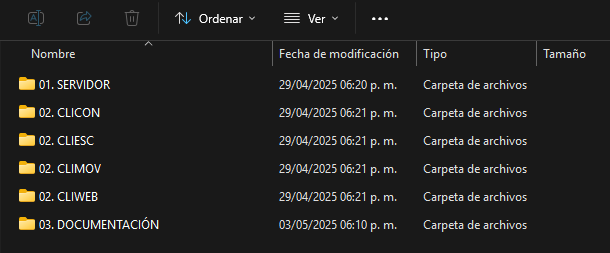


Figura 6. Estructura del proyecto carpetas internas

## SERVIDOR

### CREACION DE LA APLICACIÓN

Para el desarrollo de una aplicación SOAP de conversión de unidades en Java, se utilizó el entorno de desarrollo NetBeans y se configuró el servidor Payara como contenedor de aplicaciones. Como primer paso, se verificó que estuvieran instaladas todas las herramientas necesarias, entre ellas NetBeans con soporte para Jakarta EE, el servidor Payara Server (preferentemente en su versión Full Profile) y el JDK correspondiente a la versión del servidor.

Una vez verificados los requisitos, se procedió a la creación de un nuevo proyecto web. Para ello, se seleccionó la opción "Java with Maven Web Application" desde el asistente de nuevo proyecto.

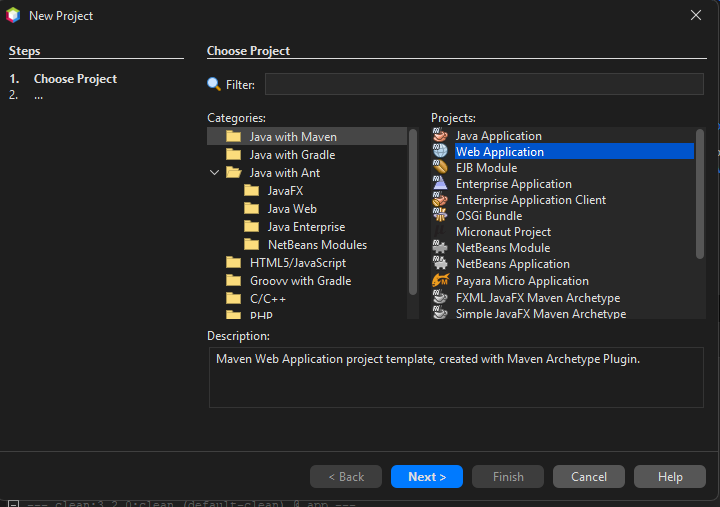


Figura 7. Crear Proyecto en NetBeans

Se debe asignar un nombre a la aplicación servidor y especificar su ubicación en el sistema de archivos. Para la configuración del servidor, se recomienda seleccionar preferentemente la versión 6 de Payara, y se debe evitar la inclusión de cualquier framework adicional durante esta etapa.

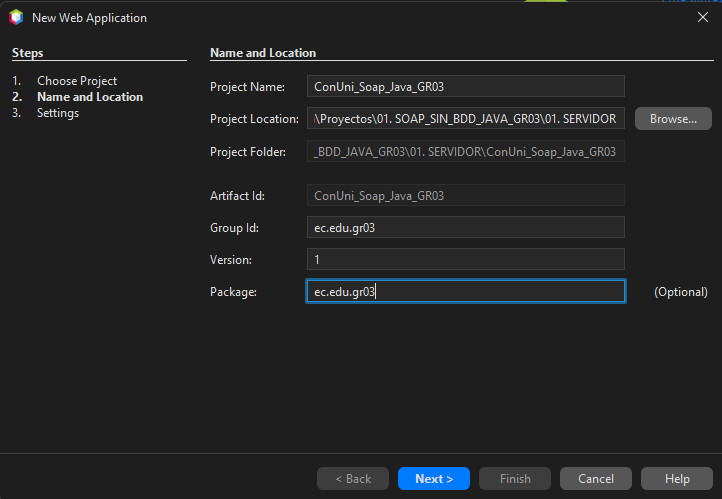


Figura 8. Configuración del Proyecto en NetBeans

### MODELO MVC

A continuación, será necesario definir la estructura del proyecto mediante la creación de tres paquetes correspondientes a los componentes Controlador, Modelo y Vista. Esta acción debe ser realizada haciendo clic derecho sobre la carpeta "Paquetes de recursos", seleccionando la opción "Nuevo" y posteriormente eligiendo "Paquete Java".

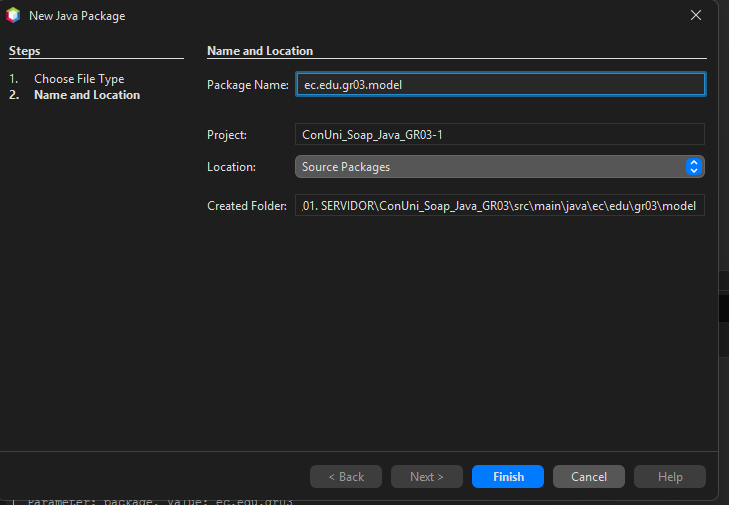


Figura 9. Creación de un nuevo paquete

Cada uno de estos paquetes estará destinado a contener las clases y archivos que representarán las responsabilidades específicas del patrón de arquitectura MVC. En este contexto, el paquete del modelo se encargará de gestionar las clases relacionadas con la lógica de datos y su acceso; el paquete del controlador se encargará de recibir las acciones de la vista y coordinar su ejecución sobre el modelo; mientras que el paquete de vista incluirá los elementos responsables de la interacción con el usuario. Por esta razón, se recomienda asignar a cada paquete un nombre que refleje su función, utilizando un formato como el siguiente: ec.edu.gr03.modelo, ec.edu. gr03.vista, y ec.edu. gr03.controlador.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Figura 10. Estructura MVC del proyecto

### IMPLEMENTACIÓN

La lógica correspondiente a la conversión de unidades es implementada en la clase ConversionUnidades en el paquete controlador, en la cual se desarrollan las funciones necesarias para realizar la conversión de centímetros a pulgadas y viceversa.

Tabla 1**. Código ConversionUnidades**

|  |
| --- |
| package ec.edu.gr03.controlador;  import jakarta.jws.WebService;  import jakarta.jws.WebMethod;  import jakarta.jws.WebParam;  @WebService(serviceName = "ConversionUnidades")  public class ConversionUnidades {  @WebMethod(operationName = "pulgadasACentimetros")  public double pulgadasACentimetros(@WebParam(name = "pulgadas") double pulgadas) {  return pulgadas\*2.54;  }  @WebMethod(operationName = "centimetrosAPulgadas")  public double centimetrosAPulgadas(@WebParam(name = "centimetros") double centimetros) {  return centimetros/2.54;  }  } |

Se procede a implementar el modelo User para realizar el login en la aplicación

Tabla 2. **Código del modelo User.**

|  |
| --- |
| package ec.edu.gr03.modelo;  public class User {  private final String username = "MONSTER";  private final String password = "MONSTER9";  public User() {  }  public String getUsername() {  return username;  }  public String getPassword() {  return password;  }    public boolean verify(String username, String password) {  return this.username.equals(username) && this.password.equals(password);  }  } |

Se procede a implementar el controlador para permitir el inicio de sesión llamado Login.

Tabla 3. **Código del controlador Login.**

|  |
| --- |
| package ec.edu.gr03.controlador;  import ec.edu.gr03.modelo.User;  import jakarta.jws.WebService;  import jakarta.jws.WebMethod;  import jakarta.jws.WebParam;  @WebService(serviceName = "Login")  public class Login {  @WebMethod(operationName = "login")  public boolean login(@WebParam(name = "username") String username, @WebParam(name = "password") String password) {  User user = new User();    return user.verify(username, password);  }  } |

### HABILITAR CORS

Para consumir el Web Service desde aplicación web o móviles se requiere configurar CORS en el servidor de modo que permita las peticiones desde estos tipos de clientes para ello se procede a crear un paquete llamado ec.edu.gr03.filtro y la clase CORSFilter.java, aquí se define las reglas para permitir la conexión de los clientes mencionados.

Tabla 4. **Habilitar CORS en el servidor.**

|  |
| --- |
| package ec.edu.gr03.filtro;  import jakarta.servlet.Filter;  import jakarta.servlet.FilterChain;  import jakarta.servlet.FilterConfig;  import jakarta.servlet.ServletException;  import jakarta.servlet.ServletRequest;  import jakarta.servlet.ServletResponse;  import jakarta.servlet.annotation.WebFilter;  import jakarta.servlet.http.HttpServletRequest;  import jakarta.servlet.http.HttpServletResponse;  import java.io.IOException;  /\*  \* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license  \* Click nbfs://nbhost/SystemFileSystem/Templates/JSP\_Servlet/Filter.java to edit this template  \*/  @WebFilter(urlPatterns = "/\*")  public class CORSFilter implements Filter {  @Override  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  throws IOException, ServletException {    HttpServletResponse httpResponse = (HttpServletResponse) response;    // Configurar cabeceras CORS  httpResponse.setHeader("Access-Control-Allow-Origin", "\*");  httpResponse.setHeader("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE, OPTIONS");  httpResponse.setHeader("Access-Control-Allow-Headers", "Content-Type, Authorization, X-Requested-With, SOAPAction");  httpResponse.setHeader("Access-Control-Max-Age", "3600");    // Para solicitudes OPTIONS (preflight)  HttpServletRequest httpRequest = (HttpServletRequest) request;  if ("OPTIONS".equalsIgnoreCase(httpRequest.getMethod())) {  httpResponse.setStatus(HttpServletResponse.SC\_OK);  return;  }    chain.doFilter(request, response);  }  @Override  public void init(FilterConfig filterConfig) throws ServletException {  // No se requiere inicialización  }  @Override  public void destroy() {  // No se requiere limpieza  }  } |

### PRUEBAS UNITARIAS

Ahora se procede a crear las pruebas unitarias de los Web Service como se indica en la figura.

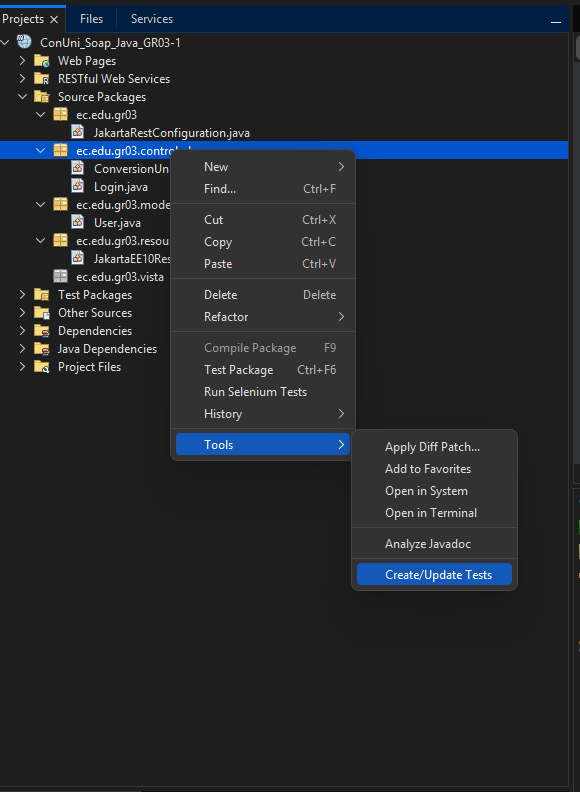


Figura 11. Creación de pruebas unitarias

A continuación, se muestra el código de las pruebas

Tabla 5. **Código de prueba unitaria LoginTest.**

|  |
| --- |
| package ec.edu.gr03.controlador;  import org.junit.jupiter.api.AfterEach;  import org.junit.jupiter.api.AfterAll;  import org.junit.jupiter.api.BeforeEach;  import org.junit.jupiter.api.BeforeAll;  import org.junit.jupiter.api.Test;  import static org.junit.jupiter.api.Assertions.\*;  public class LoginTest {    public LoginTest() {  }    @BeforeAll  public static void setUpClass() {  }    @AfterAll  public static void tearDownClass() {  }    @BeforeEach  public void setUp() {  }    @AfterEach  public void tearDown() {  }  /\*\*  \* Test of login method, of class Login.  \*/  @Test  public void testLoginIncorrectCredentials() {  System.out.println("login");  String username = "123";  String password = "123";  Login instance = new Login();  boolean expResult = false;  boolean result = instance.login(username, password);  assertEquals(expResult, result);  }    @Test  public void testLoginCorrectCredentials() {  System.out.println("login");  String username = "MONSTER";  String password = "MONSTER9";  Login instance = new Login();  boolean expResult = true;  boolean result = instance.login(username, password);  assertEquals(expResult, result);  }  } |

Tabla 6. **Código de prueba unitaria ConversionUnidadesTest.**

|  |
| --- |
| /\*  \* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license  \* Click nbfs://nbhost/SystemFileSystem/Templates/UnitTests/JUnit5TestClass.java to edit this template  \*/  package ec.edu.gr03.controlador;  import org.junit.jupiter.api.AfterEach;  import org.junit.jupiter.api.AfterAll;  import org.junit.jupiter.api.BeforeEach;  import org.junit.jupiter.api.BeforeAll;  import org.junit.jupiter.api.Test;  import static org.junit.jupiter.api.Assertions.\*;  /\*\*  \*  \* @author Drouet  \*/  public class ConversionUnidadesTest {    public ConversionUnidadesTest() {  }    @BeforeAll  public static void setUpClass() {  }    @AfterAll  public static void tearDownClass() {  }    @BeforeEach  public void setUp() {  }    @AfterEach  public void tearDown() {  }  /\*\*  \* Test of pulgadasACentimetros method, of class ConversionUnidades.  \*/  @Test  public void testPulgadasACentimetros() {  System.out.println("pulgadasACentimetros");  double pulgadas = 3;  ConversionUnidades instance = new ConversionUnidades();  double expResult = 7.62;  double result = instance.pulgadasACentimetros(pulgadas);  assertEquals(expResult, result, 0);  }  /\*\*  \* Test of centimetrosAPulgadas method, of class ConversionUnidades.  \*/  @Test  public void testCentimetrosAPulgadas() {  System.out.println("centimetrosAPulgadas");  double centimetros = 7.62;  ConversionUnidades instance = new ConversionUnidades();  double expResult = 3;  double result = instance.centimetrosAPulgadas(centimetros);  assertEquals(expResult, result, 0);  }    } |

Texto

El contenido generado por IA puede ser incorrecto.

Texto, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Figura 12. Ejecución de pruebas unitarias

### EJECUCIÓN DE LA APLICACIÓN

A continuación, se procede a con la ejecución del servidor Payara para habilitar los servicios web.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 13. Ejecución del servidor payara

## CLIENTE DE CONSOLA

### CREACION DE LA APLICACIÓN

Para la creación de la aplicación de consola se procede a crear un nuevo proyecto de Java with Ant, luego se selecciona java application para la creación de una aplicación de consola, este proyecto debe estar en la carpeta 02. CLICON.

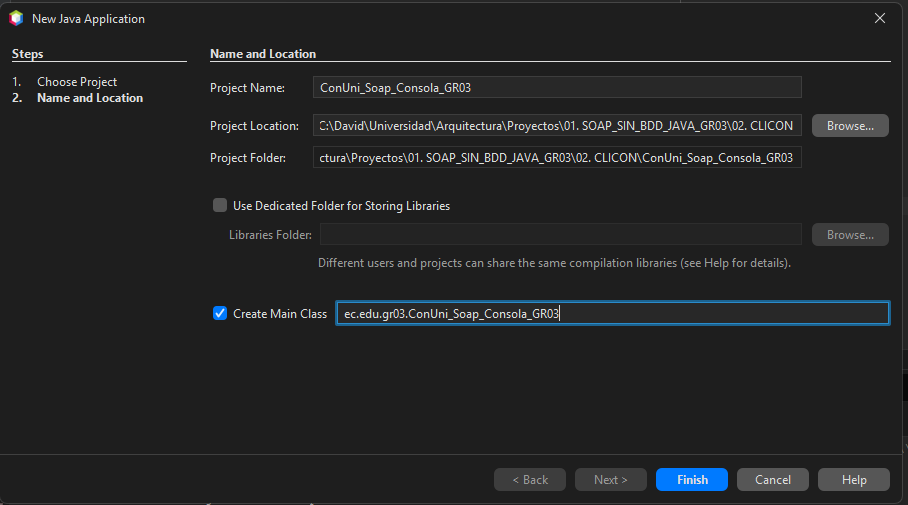


Figura 14. Configuración de la aplicación de consola

### MODELO MVC

A continuación, será necesario definir la estructura del proyecto mediante la creación de tres paquetes correspondientes a los componentes Controlador, Modelo y Vista. Esta acción debe ser realizada haciendo clic derecho sobre la carpeta "Paquetes de recursos", seleccionando la opción "Nuevo" y posteriormente eligiendo "Paquete Java".

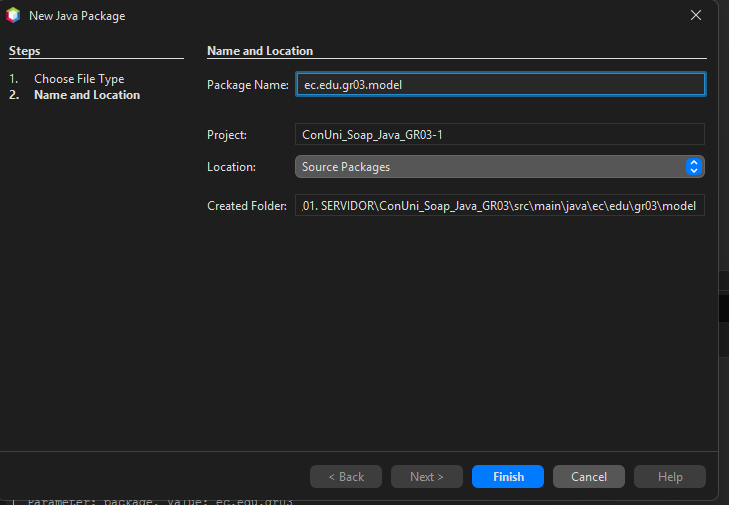


Figura 15. Creación de un nuevo paquete en cliente de consola

Cada uno de estos paquetes estará destinado a contener las clases y archivos que representarán las responsabilidades específicas del patrón de arquitectura MVC. En este contexto, el paquete del modelo se encargará de gestionar las clases relacionadas con la lógica de datos y su acceso; el paquete del controlador se encargará de recibir las acciones de la vista y coordinar su ejecución sobre el modelo; mientras que el paquete de vista incluirá los elementos responsables de la interacción con el usuario. Por esta razón, se recomienda asignar a cada paquete un nombre que refleje su función, utilizando un formato como el siguiente: ec.edu.gr03.modelo, ec.edu. gr03.vista, y ec.edu. gr03.controlador.

Texto

El contenido generado por IA puede ser incorrecto.

Figura 16. Estructura MVC del proyecto de consola

### IMPLEMENTACIÓN

Se crea la referencia al Web Service del servidor como se muestra en la figura, para ello se hace clic derecho sobre el proyecto y luego Web Service Client. Se procede a colocar los servicios web en el campo WSDL URL, los cuales son:

|  |
| --- |
| * http://localhost:8080/ConUni\_Soap\_Java\_GR03/Login?wsdl |
| * http://localhost:8080/ConUni\_Soap\_Java\_GR03/ConversionUnidades?wsdl |

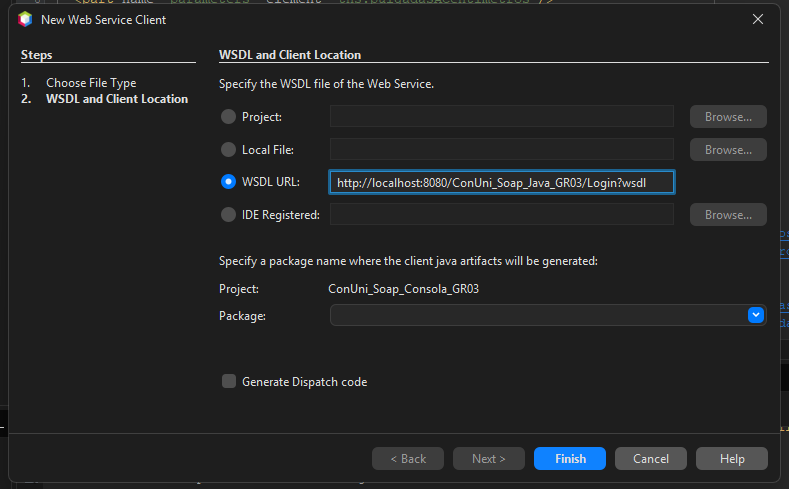


Figura 17. Creación del cliente del Web Service

Si todo sale bien se tiene todas las referencias con las funciones listas para usar.

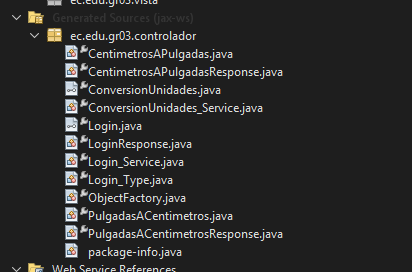


Figura 18. Creación del cliente del Web Service archivos necesarios

Se procede a crear el ViewController el cual contiene los métodos necesarios de los Web Service y exponiendo el método para ser usado en otro módulo.

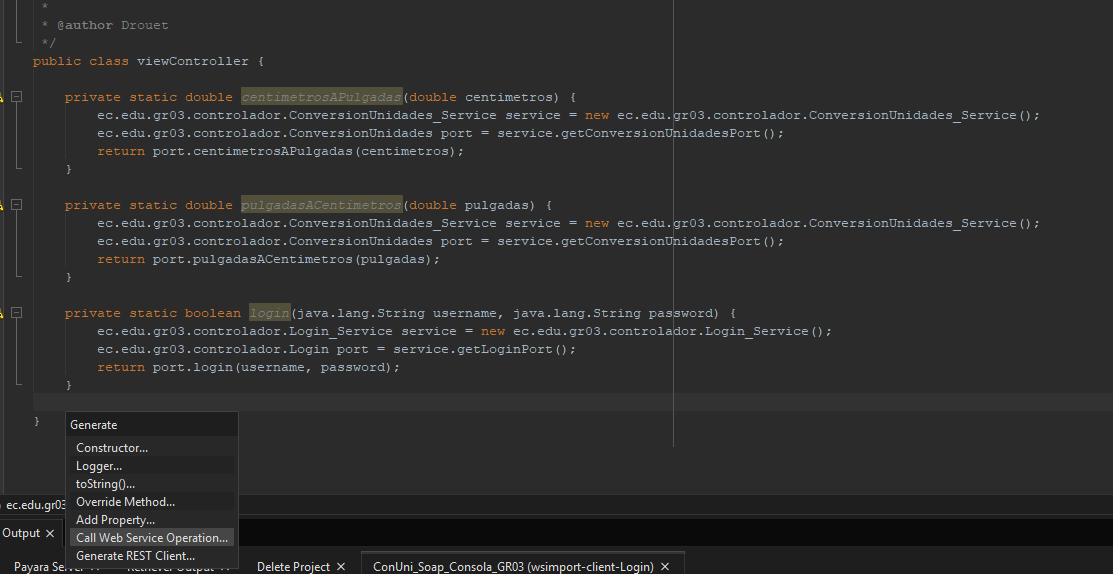


Figura 19. Creación de funciones del Web Service en el lado del cliente

Se procede a crear el ConsoleView el cual contendrá los métodos necesarios para mostrar al usuario.

Tabla 7. **Código de vista de consola ConsoleView.**

|  |
| --- |
| package ec.edu.gr03.vista;  import java.util.Scanner;  public class ConsoleView {  private final Scanner scanner = new Scanner(System.in);  public String askUsername() {  System.out.print("Ingrese usuario: ");  return scanner.nextLine();  }  public String askPassword() {  System.out.print("Ingrese contraseña: ");  return scanner.nextLine();  }  public int showMenu() {  System.out.println("\n--- Menú ---");  System.out.println("1. Convertir pulgadas a centímetros");  System.out.println("2. Convertir centímetros a pulgadas");  System.out.println("3. Cerrar sesión");  System.out.print("Seleccione una opción: ");  return scanner.nextInt();  }  public double askDouble(String message) {  System.out.print(message + " ");  return scanner.nextDouble();  }  public void showResult(String message) {  System.out.println(message);  }  public void showMessage(String message) {  System.out.println(message);  }  public void showError(String error) {  System.out.println("Error: " + error);  }  } |

Tabla 8. **Código del modelo User en cliente consola.**

|  |
| --- |
| package ec.edu.gr03.modelo;  public class User {  private String username;  public String getUsername() {  return username;  }  public void setUsername(String username) {  this.username = username;  }  } |

En caso de que al ejecutar de un error el cual indica que no se encuentra un archivo wsdl, se debe copiar el wsdl en la ruta indicada como se muestra a continuación.

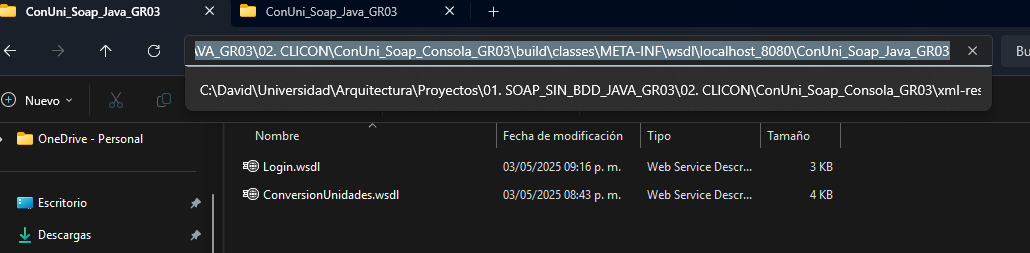


Figura 20. Resolver error de archivos faltantes

### EJECUCIÓN DE LA APLICACIÓN

Finalmente se ejecuta la aplicación de consola y se prueba las funcionalidades del Web Service, primero se procede a probar el login de la aplicación de consola.

Texto

El contenido generado por IA puede ser incorrecto.

Figura 21. Login en aplicación de consola.

A continuación, se procede a probar las funciones de convertir de pulgadas a centímetros y viceversa.

Texto

El contenido generado por IA puede ser incorrecto.

Figura 22. Funciones en aplicación de consola.

Con ello se finalizó la aplicación de consola consumiendo un Web Service del servidor y creando las interfaces de usuario de login y menú con las funcionalidades operativas.

## CLIENTE DE ESCRITORIO

### CREACION DE LA APLICACIÓN

Para la creación de la aplicación de consola se procede a crear un nuevo proyecto de Java with Ant, luego se selecciona java application para la creación de una aplicación de consola, este proyecto debe estar en la carpeta 02. CLIESC.

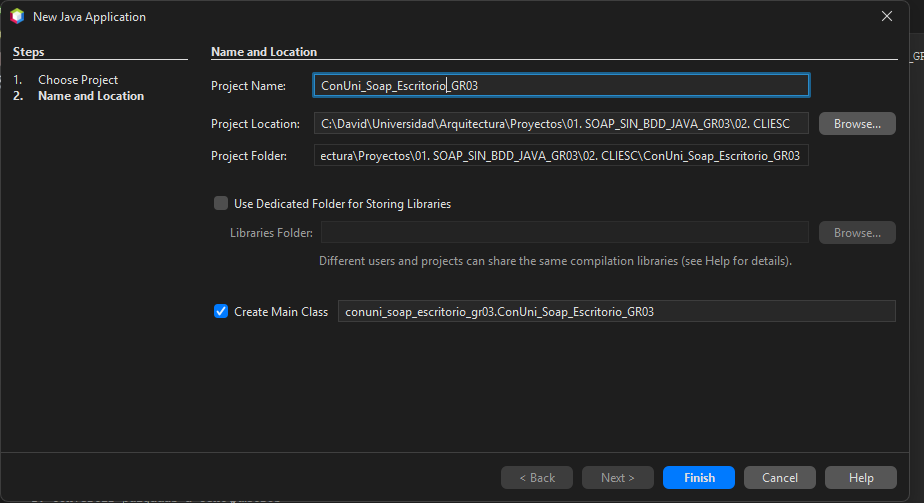


Figura 23. Configuración de la aplicación de escritorio.

### MODELO MVC

A continuación, será necesario definir la estructura del proyecto mediante la creación de tres paquetes correspondientes a los componentes Controlador, Modelo y Vista. Esta acción debe ser realizada haciendo clic derecho sobre la carpeta "Paquetes de recursos", seleccionando la opción "Nuevo" y posteriormente eligiendo "Paquete Java".

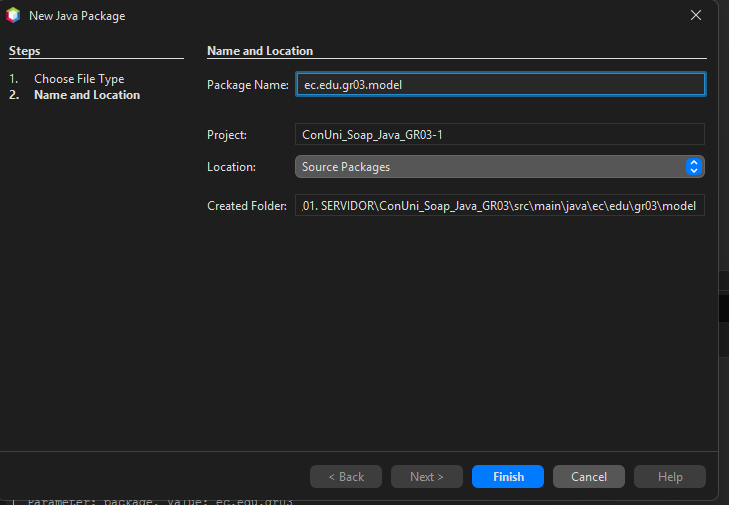


Figura 24. Creación de un nuevo paquete en cliente de escritorio

Cada uno de estos paquetes estará destinado a contener las clases y archivos que representarán las responsabilidades específicas del patrón de arquitectura MVC. En este contexto, el paquete del modelo se encargará de gestionar las clases relacionadas con la lógica de datos y su acceso; el paquete del controlador se encargará de recibir las acciones de la vista y coordinar su ejecución sobre el modelo; mientras que el paquete de vista incluirá los elementos responsables de la interacción con el usuario. Por esta razón, se recomienda asignar a cada paquete un nombre que refleje su función, utilizando un formato como el siguiente: ec.edu.gr03.modelo, ec.edu. gr03.vista, y ec.edu. gr03.controlador.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 25. Estructura MVC del proyecto de escritorio

### IMPLEMENTACIÓN

La implementación inicial es igual que la implementación de escritorio la cual se puede observar en la creación del cliente como se observa en la figura 17, la implementación del código que se observa en la figura 19 para las funciones y tabla 7 para el modelo de un usuario. Con ello se procede a realizar la implantación de la vista con los formularios LoginFrm y MenuFrm.

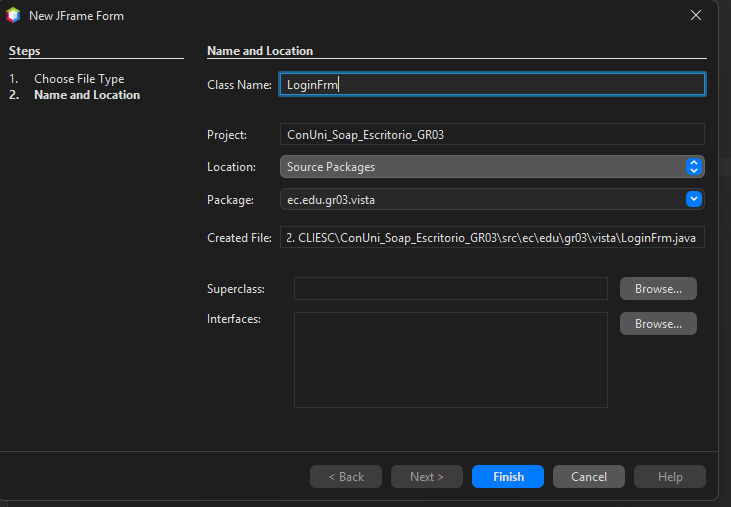


Figura 26. Crear formularios en aplicación de escritorio

Se diseña la vista de inicio de sesión de la aplicación de escritorio.

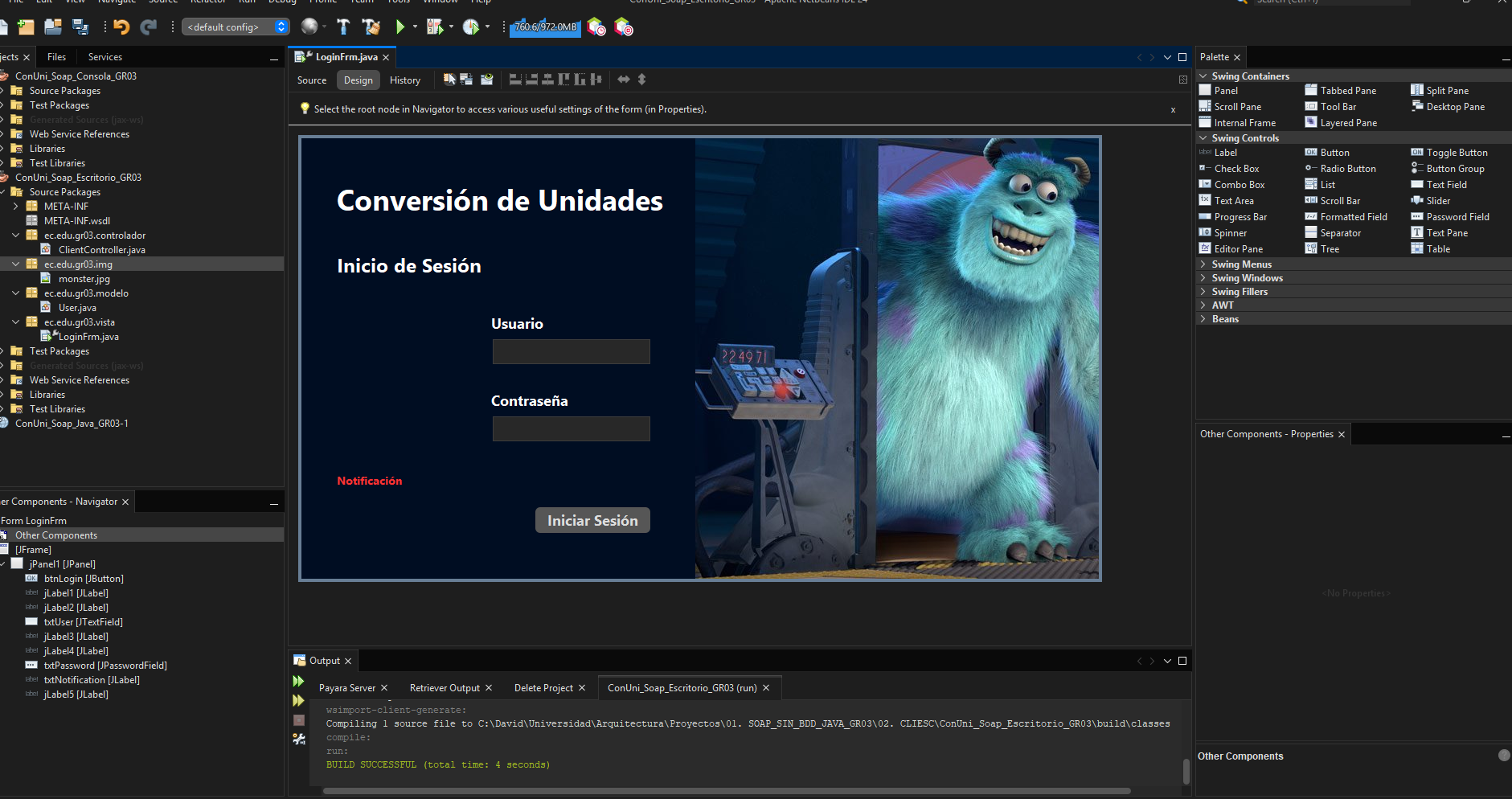


Figura 27. Crear formularios de inicio de sesión en aplicación de escritorio

Se diseña la vista de menú con las funcionalidades de conversión de unidades para la aplicación de escritorio.

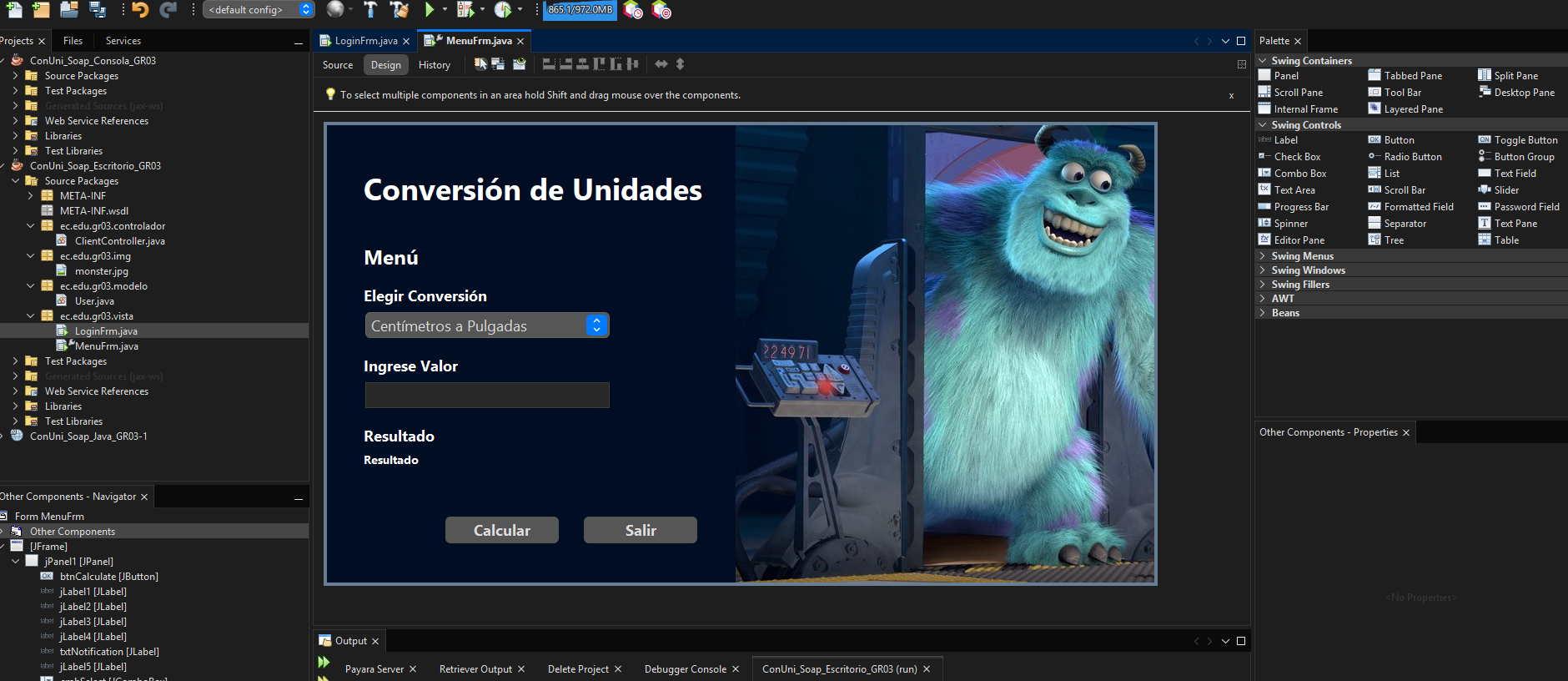


Figura 28. Crear formularios de menú en aplicación de escritorio

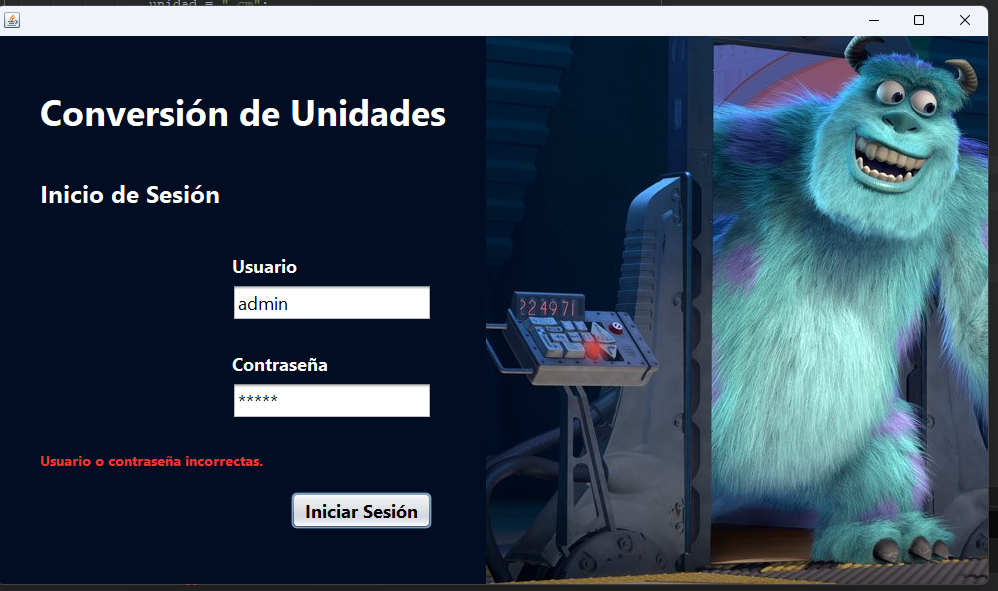
Finalmente se implementa la llamada a las funciones y la navegación entre ventanas como se muestra en el código a continuación en los eventos generados por los botones.

Tabla 9. **Código de eventos para la aplicación de escritorio.**

|  |
| --- |
| // LoginFrm  private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {  ClientController clientController = new ClientController();    String username = txtUser.getText();  String password = new String(txtPassword.getPassword());    if (clientController.login(username, password)) {  MenuFrm menuFrm = new MenuFrm();  menuFrm.setVisible(true);  this.dispose();  } else {  txtNotification.setText("Usuario o contraseña incorrectas.");  }    }  // MenuFrm  private void btnCalculateActionPerformed(java.awt.event.ActionEvent evt) {  ClientController clientController = new ClientController();  int selectFunction = cmbSelect.getSelectedIndex(); // 0: cm a pulgadas, 1: pulgadas a cm  try {  double number = Double.parseDouble(txtValue.getText());  double result;  String unidad;  if (selectFunction == 0) {  result = clientController.centimetrosAPulgadas(number);  unidad = " pulgadas";  } else {  result = clientController.pulgadasACentimetros(number);  unidad = " cm";  }  txtResult.setText(String.format("%.2f%s", result, unidad)); // Ej: 25.40 cm  } catch (NumberFormatException e) {  txtResult.setText("Error: número inválido");  }    }  private void btnExit1ActionPerformed(java.awt.event.ActionEvent evt) {  LoginFrm loginFrm = new LoginFrm();  loginFrm.setVisible(true);  this.dispose();  } |

### EJECUCIÓN DE LA APLICACIÓN

Finalmente se ejecuta la aplicación de escritorio y se prueba las funcionalidades del Web Service, primero se procede a probar el login de la aplicación.

Una captura de pantalla de un celular con texto e imagen

El contenido generado por IA puede ser incorrecto.

Figura 29. Login en aplicación de escritorio.

A continuación, se procede a probar las funciones de convertir de pulgadas a centímetros y viceversa.

Una captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto. Una captura de pantalla de un celular con texto e imagen

El contenido generado por IA puede ser incorrecto.

Figura 30. Funciones en aplicación de escritorio

Con ello se finalizó la aplicación de escritorio consumiendo un Web Service del servidor y creando las interfaces de usuario de login y menú con las funcionalidades operativas.

## CLIENTE WEB

### CREACION DE LA APLICACIÓN

Para la creación de la aplicación web se usa React + Vite con axios como cliente, el cual ayuda con la comunicación del Web Service, a continuación, se muestra el proceso de creación del proyecto base.

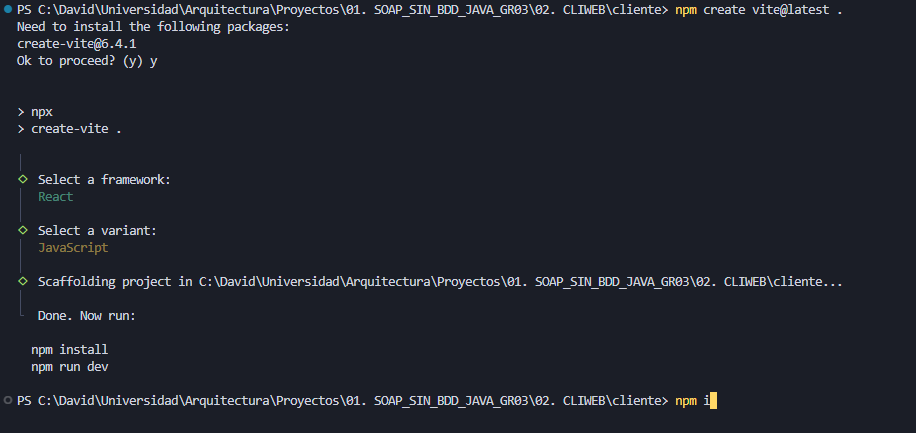


Figura 31. Crear proyecto base de la aplicación web

Luego se procede con la instalación de la biblioteca axios con el comando “npm install axios”

### MODELO MVC

Para aplicar el modelo MVC se procede a crear las carpetas model, view y controller dentro de la carpeta src del proyecto creado.

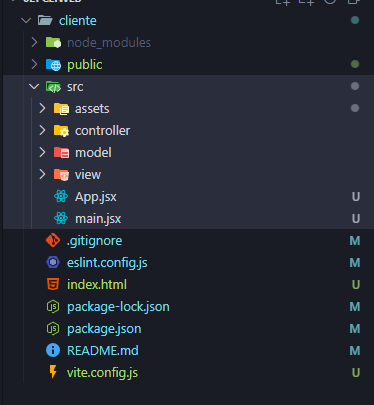


Figura 32. Crear modelo MVC en el proyecto de la aplicación web

### IMPLEMENTACIÓN

Primero se procede a crear el modelo de User el cual contiene el nombre del usuario, este se lo crea como un archivo User.js en la carpeta model.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 33. Crear modelo User en el proyecto de la aplicación web

Luego se procede a escribir el código de la clase User el cual se observa a continuación.

Tabla 10. **Modelo User en aplicación web.**

|  |
| --- |
| class User {      constructor(name) {          this.name = name;      }        getName() {          return this.name;      }        setName(name) {          this.name = name;      }  }  export default User; |

Ahora se procede a crear la conexión con el Web Service en el controller el cual se llamará ClientController.jsx para la función del Login y ConversionUnidadesController.jsx para las funciones, en los cuales se debe definir la estructura de la consulta SOAP para ello, se puede obtener la estructura correcta con SoapUI.

Tabla 11. **Código ClientController.jsx**

|  |
| --- |
| import axios from 'axios';  const SOAP\_URL = 'http://localhost:8080/ConUni\_Soap\_Java\_GR03/Login?wsdl';  *// Función auxiliar para envolver el cuerpo XML en un envelope SOAP*  const createSoapEnvelope = (body) => `    <soapenv:Envelope      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"      xmlns:con="http://controlador.gr03.edu.ec/">      <soapenv:Header/>      <soapenv:Body>        ${body}      </soapenv:Body>    </soapenv:Envelope>  `;  *// Función auxiliar para extraer la respuesta del XML*  const extractResponse = (xmlString, action) => {      const parser = new DOMParser();      const xmlDoc = parser.parseFromString(xmlString, 'text/xml');      return xmlDoc.querySelector(`${action}Response`);  };  *// Función para realizar la llamada SOAP*  const callSoapService = async (action, body) => {      const envelope = createSoapEnvelope(body);      const response = await axios.post(          SOAP\_URL,          envelope,          {              headers: {                  'Content-Type': 'text/xml; charset=utf-8',                  SOAPAction: `"${action}"`,              },          }      );      return response.data;  };  *// Función para iniciar sesión*  export const login = async (username, password) => {      const body =          `<con:login>              <username>${username}</username>              <password>${password}</password>          </con:login>`;      const response = await callSoapService('', body);      const loginResponse = extractResponse(response, 'login');      return loginResponse ? loginResponse.textContent : null;  }; |

Tabla 12. **Código ConversionUnidadesController.jsx**

|  |
| --- |
| *// ConversionUnidadesController.jsx*  import axios from 'axios';  const SOAP\_URL = 'http://localhost:8080/ConUni\_Soap\_Java\_GR03/ConversionUnidades?wsdl';  *// Función auxiliar para crear el envelope SOAP*  const createSoapEnvelope = (body) => `    <soapenv:Envelope      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"      xmlns:con="http://controlador.gr03.edu.ec/">      <soapenv:Header/>      <soapenv:Body>        ${body}      </soapenv:Body>    </soapenv:Envelope>  `;  *// Función auxiliar para extraer la respuesta del XML*  const extractResponseValue = (xmlString, tagName) => {    const parser = new DOMParser();    const xmlDoc = parser.parseFromString(xmlString, 'text/xml');    const valueNode = xmlDoc.querySelector(tagName);    return valueNode ? valueNode.textContent : null;  };  *// Función para hacer la llamada SOAP*  const callSoapService = async (action, body) => {    const envelope = createSoapEnvelope(body);    const response = await axios.post(SOAP\_URL, envelope, {      headers: {        'Content-Type': 'text/xml; charset=utf-8',        SOAPAction: `"${action}"`, *// En tu WSDL estaba*      },    });    return response.data;  };  *// Convertir pulgadas a centímetros*  export const pulgadasACentimetros = async (pulgadas) => {    const body = `      <con:pulgadasACentimetros>        <pulgadas>${pulgadas}</pulgadas>      </con:pulgadasACentimetros>    `;    const response = await callSoapService('', body);    return extractResponseValue(response, 'return');  };  *// Convertir centímetros a pulgadas*  export const centimetrosAPulgadas = async (centimetros) => {    const body = `      <con:centimetrosAPulgadas>        <centimetros>${centimetros}</centimetros>      </con:centimetrosAPulgadas>    `;    const response = await callSoapService('', body);    return extractResponseValue(response, 'return');  }; |

Ahora se procede a crear las pantallas de Login y Menú los cuales deben ir en la carpeta views como se muestra a continuación, aquí se llamará a los métodos del Web Service y se le mostrará la información en forma agradable.

Tabla 13. **Código de la vista Login.jsx**

|  |
| --- |
| import { useState } from 'react';  import { login } from '../controller/ClientController';  import { useNavigate } from 'react-router-dom';  function Login() {    const [username, setUsername] = useState('');    const [password, setPassword] = useState('');    const [loading, setLoading] = useState(false);    const [error, setError] = useState('');    const navigate = useNavigate();    const handleLogin = async (e) => {      e.preventDefault();      setLoading(true);      try {        const result = await login(username, password);        if (result === 'true') {          navigate('/menu');        } else {          setError('Usuario o contraseña incorrectos');        }      } catch (error) {        setError('Error al iniciar sesión');        console.error(error);      }      setLoading(false);    };    return (      <div className="login-wrapper">        {*/\* Contenedor del formulario \*/*}        <div className="container">          <h2>Conversión de Unidades</h2>          <h2>Inicio de Sesión</h2>          <form onSubmit={handleLogin}>            <input              type="text"              placeholder="Usuario"              value={username}              onChange={(e) => setUsername(e.target.value)}              required            />            <input              type="password"              placeholder="Contraseña"              value={password}              onChange={(e) => setPassword(e.target.value)}              required            />            <button type="submit" disabled={loading}>              {loading ? 'Cargando...' : 'Iniciar sesión'}            </button>          </form>          {error && <p style={{ color: 'red' }}>{error}</p>}        </div>        {*/\* Imagen decorativa \*/*}        <img src="/monster.jpg" alt="monster" className="login-image" />      </div>    );  }  export default Login; |

Tabla 14. **Código de la vista Menu.jsx**

|  |
| --- |
| *// src/views/Menu.jsx*  import { useState } from 'react';  import { centimetrosAPulgadas, pulgadasACentimetros } from '../controller/ConversionUnidadesController';  import { useNavigate } from 'react-router-dom';  function Menu() {    const [tipo, setTipo] = useState('cm-pulg');    const [valor, setValor] = useState('');    const [resultado, setResultado] = useState('');    const navigate = useNavigate();    const calcular = async () => {      try {        const num = parseFloat(valor);        if (isNaN(num)) return setResultado('Debe ingresar un número válido.');        if (tipo === 'cm-pulg') {          const res = await centimetrosAPulgadas(num);          setResultado(`${num} cm = ${res} pulgadas`);        } else {          const res = await pulgadasACentimetros(num);          setResultado(`${num} pulgadas = ${res} cm`);        }      } catch (error) {        setResultado('Error al convertir');        console.error(error);      }    };    const cerrarSesion = () => {      navigate('/');    };    return (      <div className="login-wrapper">        <div className="container">        <h2>Conversión de Unidades</h2>        <select value={tipo} onChange={(e) => setTipo(e.target.value)} style={{ marginBottom: '10px', width: '100%' }}>          <option value="cm-pulg">Centímetros a Pulgadas</option>          <option value="pulg-cm">Pulgadas a Centímetros</option>        </select>        <input          type="text"          placeholder="Ingrese valor"          value={valor}          onChange={(e) => setValor(e.target.value)}          style={{ display: 'block', marginBottom: '10px', width: '100%' }}        />        <button onClick={calcular} style={{ marginRight: '10px', marginBottom: '10px', width: '100%' }}>          Calcular        </button>        <button onClick={cerrarSesion}>Cerrar sesión</button>        {resultado && <p style={{ marginTop: '20px' }}>{resultado}</p>}        </div>          <img src="/monster.jpg" alt="monster" className="login-image" />      </div>      );  }  export default Menu; |

Se procede a realizar el enrutamiento de la aplicación en el App.jsx como se muestra en el código.

Tabla 15. **Código App.jsx**

|  |
| --- |
| *// src/App.jsx*  import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';  import Login from './views/Login';  import Menu from './views/Menu';  function App() {    return (      <Router>        <Routes>          <Route path="/" element={<Login />} />          <Route path="/menu" element={<Menu />} />        </Routes>      </Router>    );  }  export default App; |

Finalmente se añade estilos css para dar un estilo agradable a la aplicación web, este se encuentra dentro de la carpeta src/styles.

Tabla 16. **Estilo de la aplicación web styles.css**

|  |
| --- |
| *body {*  *display: flex;*  *justify-content: center;*  *align-items: center;*  *min-height: 100vh;*  *margin: 0;*  *background-color: #f0f0f0;*  *}*    *.login-wrapper {*  *display: flex;*  *align-items: center;*  *justify-content: center;*  *}*    *.container {*  *width: 400px;*  *padding: 30px;*  *background-color: white;*  *box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);*  *border-radius: 16px;*  *margin-right: 50px;*  *}*    *h2 {*  *text-align: center;*  *margin-bottom: 24px;*  *color: #1976d2;*  *}*    *input, select {*  *width: 100%;*  *padding: 12px;*  *margin-bottom: 16px;*  *border: 1px solid #ccc;*  *border-radius: 8px;*  *box-sizing: border-box;*  *font-size: 16px;*  *}*    *button {*  *background-color: #1976d2;*  *color: white;*  *border: none;*  *padding: 12px 16px;*  *border-radius: 8px;*  *font-size: 16px;*  *cursor: pointer;*  *width: 100%;*  *transition: background-color 0.3s;*  *display: flex;*  *justify-content: center;*  *align-items: center;*  *text-align: center;*  *}*    *button:hover {*  *background-color: #1565c0;*  *}*    *p {*  *margin-top: 16px;*  *font-weight: bold;*  *text-align: center;*  *color: #2e7d32;*  *}*    *.login-image {*  *max-width: 600px;*  *height: auto;*  *border-radius: 4px;*  *}* |

### EJECUCIÓN DE LA APLICACIÓN

Finalmente se ejecuta la aplicación web y se prueba las funcionalidades del Web Service, primero se procede a probar el login de la aplicación.

Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto. Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.

Figura 34. Login en aplicación web.

A continuación, se procede a probar las funciones de convertir de pulgadas a centímetros y viceversa.

Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.

Figura 35. Funciones en aplicación de web

Con ello se finalizó la aplicación web, consumiendo un Web Service del servidor y creando las interfaces de usuario de login y menú con las funcionalidades operativas.

## CLIENTE MÓVIL

### CREACION DE LA APLICACIÓN

Para la configuración del proyecto móvil se procede a utilizar React native, el comando para empezar la configuración del proyecto es “npx create-expo-app@latest .”, con ello se iniciará la creación del proyecto en la carpeta “02. CLIMOV/cliente”

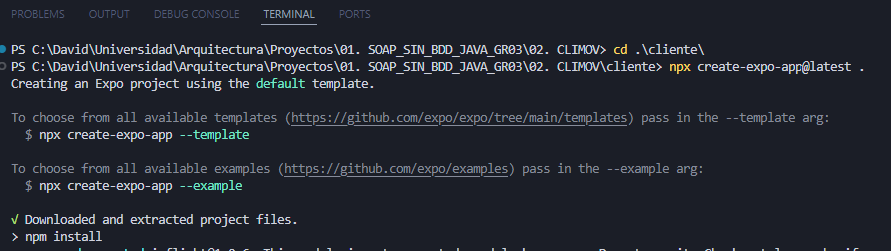


Figura 36. Creación y configuración del proyecto móvil

Para la realización de pruebas y ejecución del proyecto se usará la aplicación Expo Go la cual permite ejecutar los proyectos en desarrollo en el celular, luego se procede con la instalación de la biblioteca axios y fast-xml-parser con el comando “npm install axios fast-xml-parser”.

.

### MODELO MVC

Para aplicar el modelo MVC se procede a crear las carpetas model, views y controllers dentro de la carpeta app del proyecto creado.

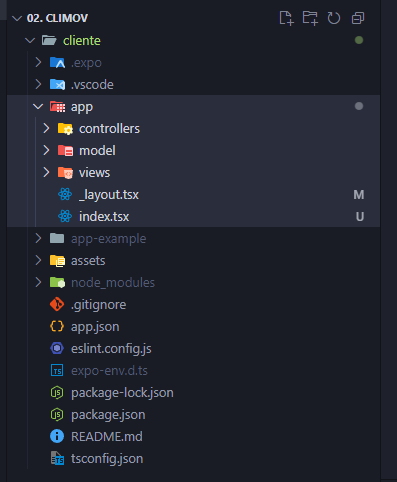


Figura 37. Crear modelo MVC en el proyecto de la aplicación móvil

### IMPLEMENTACIÓN

Se procede a crear la conexión con el Web Service en el controller el cual se llamará LoginController.js para la función del Login y ConversionUnidadesController.js para las funciones, en los cuales se debe definir la estructura de la consulta SOAP para ello, se puede obtener la estructura correcta con SoapUI.

Tabla 17. **Código LoginController.js para aplicación móvil**

|  |
| --- |
| import axios from 'axios';  import { XMLParser } from 'fast-xml-parser';  const SOAP\_URL = 'http://192.168.1.5:8080/ConUni\_Soap\_Java\_GR03/Login?wsdl';  *// Función auxiliar para envolver el cuerpo XML en un envelope SOAP*  const createSoapEnvelope = (body) => `    <soapenv:Envelope      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"      xmlns:con="http://controlador.gr03.edu.ec/">      <soapenv:Header/>      <soapenv:Body>        ${body}      </soapenv:Body>    </soapenv:Envelope>  `;  const extractResponse = (xmlString, action) => {      const parser = new XMLParser({        ignoreAttributes: false,        ignoreDeclaration: true,        removeNSPrefix: true *// ⚠️ Esto elimina el prefijo como soapenv:, con:, etc.*      });      const jsonObj = parser.parse(xmlString);    *// Después de eliminar los prefijos*      const body = jsonObj.Envelope?.Body;      if (!body) return null;    *// Buscar la respuesta (ej. loginResponse)*      const responseKey = Object.keys(body).find(key => key.includes(`${action}Response`));      const returnValue = body[responseKey]?.return;        return returnValue ?? null;    };    *// Función para realizar la llamada SOAP*  const callSoapService = async (action, body) => {    const envelope = createSoapEnvelope(body);    const response = await axios.post(      SOAP\_URL,      envelope,      {        headers: {          'Content-Type': 'text/xml; charset=utf-8',          SOAPAction: `"${action}"`,        },      }    );    return response.data;  };  *// Función para iniciar sesión*  export const login = async (username, password) => {    const body = `      <con:login>        <username>${username}</username>        <password>${password}</password>      </con:login>    `;    const response = await callSoapService('', body);    const loginResponse = extractResponse(response, 'login');    return loginResponse;  }; |

Tabla 18. **Código ConversionUnidadesController.js para aplicación móvil**

|  |
| --- |
| *import axios from 'axios';*  *import { XMLParser } from 'fast-xml-parser';*  *const SOAP\_URL = 'http://192.168.1.5:8080/ConUni\_Soap\_Java\_GR03/ConversionUnidades?wsdl';*  *// Función auxiliar para crear el envelope SOAP*  *const createSoapEnvelope = (body) => `*  *<soapenv:Envelope*  *xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"*  *xmlns:con="http://controlador.gr03.edu.ec/">*  *<soapenv:Header/>*  *<soapenv:Body>*  *${body}*  *</soapenv:Body>*  *</soapenv:Envelope>*  *`;*  *const extractResponseValue = (xmlString, action) => {*  *const parser = new XMLParser({*  *ignoreAttributes: false,*  *ignoreDeclaration: true,*  *removeNSPrefix: true*  *});*  *const jsonObj = parser.parse(xmlString);*    *// Después de eliminar los prefijos*  *const body = jsonObj.Envelope?.Body;*  *if (!body) return null;*    *// Buscar la respuesta (ej. loginResponse)*  *const responseKey = Object.keys(body).find(key => key.includes(`${action}Response`));*  *const returnValue = body[responseKey]?.return;*    *return returnValue ?? null;*  *};*  *// Función para hacer la llamada SOAP*  *const callSoapService = async (action, body) => {*  *const envelope = createSoapEnvelope(body);*  *const response = await axios.post(SOAP\_URL, envelope, {*  *headers: {*  *'Content-Type': 'text/xml; charset=utf-8',*  *SOAPAction: `"${action}"`, // Puede quedar vacío*  *},*  *});*  *return response.data;*  *};*  *// Convertir pulgadas a centímetros*  *export const pulgadasACentimetros = async (pulgadas) => {*  *const body = `*  *<con:pulgadasACentimetros>*  *<pulgadas>${pulgadas}</pulgadas>*  *</con:pulgadasACentimetros>*  *`;*  *const response = await callSoapService('', body);*  *return extractResponseValue(response, 'pulgadasACentimetros');*  *};*  *// Convertir centímetros a pulgadas*  *export const centimetrosAPulgadas = async (centimetros) => {*  *const body = `*  *<con:centimetrosAPulgadas>*  *<centimetros>${centimetros}</centimetros>*  *</con:centimetrosAPulgadas>*  *`;*  *const response = await callSoapService('', body);*  *return extractResponseValue(response, 'centimetrosAPulgadas');*  *};* |

Ahora se procede a crear las pantallas de Login y Menú los cuales deben ir en la carpeta views como se muestra a continuación, aquí se llamará a los métodos del Web Service y se le mostrará la información en forma agradable.

Tabla 19. **Código de la vista LoginScreen.jsx de la aplicación móvil**

|  |
| --- |
| import React, { useState } from "react";  import { View, TextInput, Button, Text, StyleSheet, Image } from "react-native";  import { login } from "../controllers/LoginController";  import { useRouter } from "expo-router";  export default function LoginScreen() {    const [username, setUsername] = useState("");    const [password, setPassword] = useState("");    const [error, setError] = useState("");    const router = useRouter();    const handleLogin = async () => {      try {        const result = await login(username, password);        if (true) {          router.push("/views/MenuScreen");        } else {          setError("Credenciales incorrectas.");        }      } catch (err) {        setError("Error de conexión: " + err.message);      }    };    return (      <View style={styles.container}>        <Image source={require("@/assets/images/monster.jpg")} style={styles.image} />        <TextInput          placeholder="Usuario"          value={username}          onChangeText={setUsername}          style={styles.input}        />        <TextInput          placeholder="Contraseña"          secureTextEntry          value={password}          onChangeText={setPassword}          style={styles.input}        />        <View style={styles.button}>          <Button title="Iniciar sesión" onPress={handleLogin} color="#007AFF" />        </View>        {error ? <Text style={styles.error}>{error}</Text> : null}      </View>    );  }  const styles = StyleSheet.create({    container: {      flex: 1, justifyContent: "center", alignItems: "center", padding: 20, backgroundColor: "#f2f2f2",    },    image: {      width: "100%", height: 220, marginBottom: 20, borderRadius: 8,    },    input: {      width: "100%",      height: 50,      backgroundColor: "#fff",      borderRadius: 10,      paddingHorizontal: 15,      marginBottom: 12,      borderWidth: 1,      borderColor: "#ccc",    },    button: {      width: "100%",      marginTop: 10,      borderRadius: 10,      overflow: "hidden",      backgroundColor: "#007AFF",    },    error: {      color: "red",      marginTop: 10,      textAlign: "center",    },  }); |

Tabla 20. **Código de la vista MenuScreen.jsx de la aplicación móvil**

|  |
| --- |
| *import React, { useState } from "react";*  *import { View, TextInput, Button, Text, StyleSheet, Image } from "react-native";*  *import { pulgadasACentimetros, centimetrosAPulgadas } from "../controllers/ConversionUnidadesController";*  *import { useRouter } from "expo-router";*  *export default function MenuScreen() {*  *const [input, setInput] = useState("");*  *const [resultado, setResultado] = useState("");*  *const router = useRouter();*  *const convertirPC = async () => {*  *const res = await pulgadasACentimetros(input);*  *setResultado(`${input} pulgadas = ${res} cm`);*  *};*  *const convertirCP = async () => {*  *const res = await centimetrosAPulgadas(input);*  *setResultado(`${input} cm = ${res} pulgadas`);*  *};*  *const logout = () => {*  *router.replace("/views/LoginScreen");*  *};*  *return (*  *<View style={styles.container}>*  *<Image source={require("@/assets/images/monster.jpg")} style={styles.image} />*  *<TextInput*  *placeholder="Ingrese valor"*  *keyboardType="numeric"*  *value={input}*  *onChangeText={setInput}*  *style={styles.input}*  */>*  *<View style={styles.button}>*  *<Button title="Pulgadas a Centímetros" onPress={convertirPC} color="#34C759" />*  *</View>*  *<View style={styles.button}>*  *<Button title="Centímetros a Pulgadas" onPress={convertirCP} color="#FF9500" />*  *</View>*  *{resultado ? <Text style={styles.result}>{resultado}</Text> : null}*  *<View style={[styles.button, { marginTop: 20 }]}>*  *<Button title="Cerrar sesión" color="#FF3B30" onPress={logout} />*  *</View>*  *</View>*  *);*  *}*  *const styles = StyleSheet.create({*  *container: {*  *flex: 1, justifyContent: "center", alignItems: "center", padding: 20, backgroundColor: "#f2f2f2",*  *},*  *image: {*  *width: "100%", height: 220, marginBottom: 20, borderRadius: 8,*  *},*  *input: {*  *width: "100%",*  *height: 50,*  *backgroundColor: "#fff",*  *borderRadius: 10,*  *paddingHorizontal: 15,*  *marginBottom: 12,*  *borderWidth: 1,*  *borderColor: "#ccc",*  *},*  *button: {*  *width: "100%",*  *marginVertical: 5,*  *},*  *result: {*  *marginTop: 20,*  *fontSize: 16,*  *fontWeight: "bold",*  *color: "#333",*  *},*  *});* |

Se procede a realizar el enrutamiento de la aplicación en el App.jsx como se muestra en el código.

Tabla 21. **Código index.tsx de la aplicación móvil**

|  |
| --- |
| *// src/App.jsx*  import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';  import Login from './views/Login';  import Menu from './views/Menu';  function App() {    return (      <Router>        <Routes>          <Route path="/" element={<Login />} />          <Route path="/menu" element={<Menu />} />        </Routes>      </Router>    );  }  export default App; |

Con ello la aplicación móvil esta lista para ser ejecutada, para este fin se usará expo Go el cual permite ejecutar la aplicación desde el celular.

### EJECUCIÓN DE LA APLICACIÓN

Finalmente se ejecuta la aplicación móvil y se prueba las funcionalidades del Web Service, primero se procede a probar el login de la aplicación.

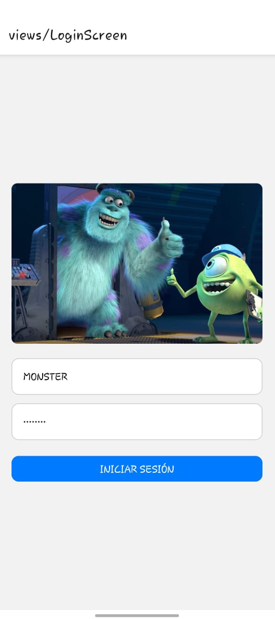


Figura 38. Login en aplicación móvil.

A continuación, se procede a probar las funciones de convertir de pulgadas a centímetros y viceversa.

Captura de pantalla de un celular con la imagen de un gato

El contenido generado por IA puede ser incorrecto. Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 39. Funciones en aplicación de móvil

Con ello se finalizó la aplicación móvil, consumiendo un Web Service del servidor y creando las interfaces de usuario de login y menú con las funcionalidades operativas.

# CONCLUSIONES

* La implementación de un sistema de conversión de unidades utilizando servicios web SOAP en Java sin base de datos es viable y eficiente para aplicaciones que requieren alta interoperabilidad y estructuras de datos estáticas.
* La elección de Java para el desarrollo del servidor y las aplicaciones cliente garantiza la portabilidad y la robustez del sistema.
* La combinación de Node.js con React y React Native permite el desarrollo de interfaces de usuario modernas y reactivas para aplicaciones web y móviles, respectivamente.

# RECOMENDACIONES

* Para futuras implementación considerar usar el framework de expo para crear la aplicación web y móvil al mismo tiempo.
* Mantener una documentación detallada y actualizada del sistema, incluyendo la arquitectura, las tecnologías utilizadas y las decisiones de diseño.
* Evaluar la posibilidad de migrar a una arquitectura basada en microservicios en caso de que el sistema requiera mayor escalabilidad y flexibilidad en el futuro.

# BIBLIOGRAFÍA

* IBM. (s.f.). SOAP. Recuperado de <https://www.ibm.com/docs/es/rsas/7.5.0?topic=standards-soap>
* F5. (s.f.). ¿Qué es el Protocolo Simple de Acceso a Objetos (SOAP)?. Recuperado de <https://www.f5.com/es_es/glossary/simple-object-access-protocol-soap>
* Atlassian. (s.f.). Arquitectura de microservicios. Recuperado de <https://www.atlassian.com/es/microservices/microservices-architecture>
* Paradigma Digital. (s.f.). Patrones de arquitectura de microservicios, ¿qué son y qué ventajas ofrecen?. Recuperado de <https://www.paradigmadigital.com/dev/patrones-arquitectura-microservicios-que-son-ventajas/>
* Profile. (2024). ¿Qué es la arquitectura de software? Fundamentos clave. Recuperado de <https://profile.es/blog/que-es-la-arquitectura-de-software/>