

**Tema**

Web Service

SOAP\_CONUI\_JAVA\_GR10

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU. CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

04/05/2025

**Tabla de contenido**

[INTRODUCCION A LA ARQUITECTURA DE SOFTWARE 4](#_Toc197257749)

[INTRODUCCIÓN 4](#_Toc197257750)

[OBJETIVO 4](#_Toc197257751)

[ARQUITECTURA DE SOFTWARE PARTE I 4](#_Toc197257752)

[ARQUITECTURA DE SOFTWARE PARTE II 9](#_Toc197257753)

[ARQUITECTURA DE SOFTWARE PARTE III 15](#_Toc197257754)

[CONCLUSIONES 20](#_Toc197257755)

[RECOMENDACIONES 21](#_Toc197257756)

[BIBLIOGRAFÍA 21](#_Toc197257757)

**ÍNDICE DE IMÁGENES**

[Figura 1. Analogía entre tipos de construcciones físicas y niveles de complejidad en la arquitectura de software 6](#_Toc197257553)

[Figura 2. La Mansión Winchester: una analogía con sistemas de software sin planeación arquitectónica clara 7](#_Toc197257554)

[Figura 3. La metáfora de las piedras: priorización de elementos clave en la arquitectura de software 9](#_Toc197257555)

[Figura 4. Modelo 4+1 vistas de la arquitectura de software según Kruchten 17](#_Toc197257556)

**ÍNDICE DE TABLAS**

[Tabla 1. Tipos de Requerimientos No Funcionales 13](#_Toc197257232)

# Servicio web SOAP para la conversión de unidades implementado en Netbeans

## INTRODUCCIÓN

La arquitectura de software constituye un componente esencial en el desarrollo de sistemas, ya que establece la estructura y el diseño general que guiará su construcción. Para comprender mejor su propósito y complejidad, puede compararse con la arquitectura tradicional: ambas disciplinas requieren planificación rigurosa, toma de decisiones estructurales y una orientación constante hacia la satisfacción de necesidades reales. Así como un arquitecto de edificaciones debe dominar aspectos técnicos y contextuales, el arquitecto de software debe comprender los requisitos del sistema, anticipar cambios y diseñar soluciones funcionales y sostenibles.

Dentro de esta disciplina, se destacan elementos clave como los componentes, las conexiones, los patrones arquitectónicos y la documentación, los cuales permiten definir el comportamiento y la adaptabilidad del sistema. Además, el modelado y diseño arquitectónico, mediante enfoques como el modelo 4+1 vistas y el uso de UML, resulta fundamental para representar, analizar y comunicar la estructura del software desde distintas perspectivas. Estos recursos no solo facilitan la colaboración entre equipos, sino que aseguran una base sólida para el desarrollo de sistemas escalables y coherentes con su entorno operativo.

## OBJETIVO

Explorar los principios fundamentales de la arquitectura de software mediante su analogía con la arquitectura tradicional, con el fin de comprender su propósito estructural, sus elementos esenciales y la relevancia del modelado y diseño arquitectónico como herramientas clave para construir sistemas robustos, escalables y alineados con las necesidades del entorno y los usuarios.

## ARQUITECTURA DE SOFTWARE PARTE I

## ARQUITECTURA DE SOFTWARE PARTE II

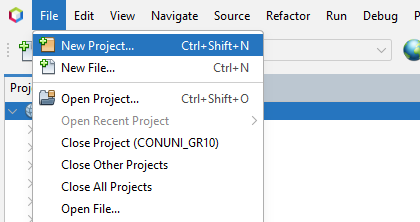
## ARQUITECTURA DE SOFTWARE PARTE III

## DESARROLLO

### APLICACIÓN SERVIDOR

#### Creación del proyecto servidor

Iniciar el IDE Apache NetBeans y, en la pantalla principal, seleccionar la opción “Archivo” en la barra superior, seguida de “Nuevo proyecto”.



1. Creación nuevo proyecto para el servidor en Netbeans

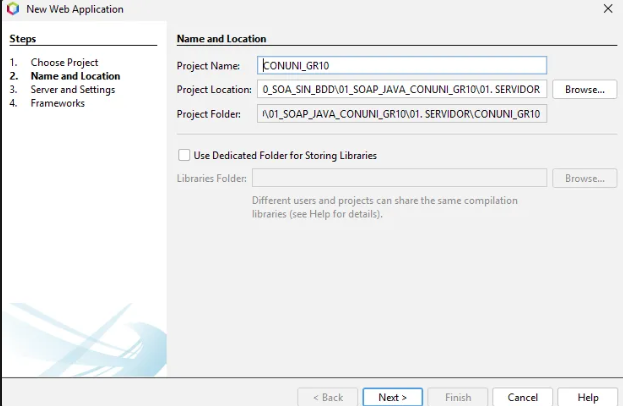
En la ventana emergente, ubicar en la categoría “Java with Ant” el tipo de proyecto “Java Web” y seleccionar “Aplicación Web”.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

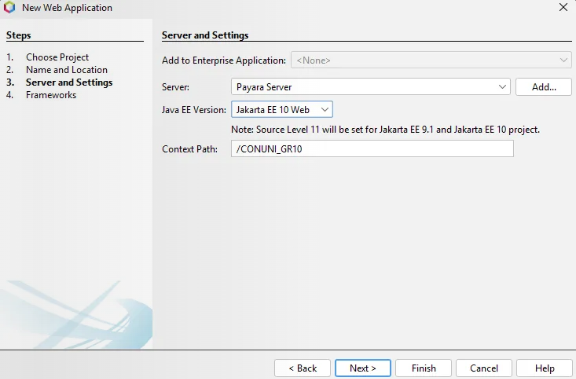
1. Selección de tipo de proyecto en Apache NetBeans

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



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.



1. Configuración del servidor y versión de Java

Seguidamente, elegir el Framework de desarrollo a utilizar. En este proyecto, no se requerirá ninguno, por lo que no se debe marcar ninguna casilla.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Una vez creado el proyecto, hacer clic derecho sobre el proyecto, seleccionar “Propiedades” en el menú, dirigirse a la pestaña “Librerías” y agregar la dependencia jakarta.jws-api.rar mediante la opción “Add JAR/Folder”, ubicada en la carpeta donde se instaló Payara Server en la máquina local.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

1. Agregar dependencia jakarta.jws-api.rar en las propiedades del proyecto

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

Asignar el nombre “CONUNI” al Web Service y verificar que se cree dentro del paquete “ec.edu.monster.controlador”.

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 CONUNI

Después de crear el Web Service, modificar las importaciones de javax.jws a jakarta.jws. Este cambio es necesario porque las especificaciones de Java EE han evolucionado hacia Jakarta EE, y jakarta.jws es el paquete actualizado para las anotaciones de servicios web en entornos modernos como Payara Server.

1. Actualización de importaciones a jakarta.jws en el Web Service CONUNI.java

|  |
| --- |
| package ec.edu.monster.controlador;  import jakarta.jws.WebService; // Cambia javax.jws por jakarta.jws  import jakarta.jws.WebMethod;  import jakarta.jws.WebParam; |

Con el servicio web creado, proceder a desarrollar el servicio para la conversión de unidades en el paquete “ec.edu.monster.servicio”. Hacer clic derecho sobre el paquete, seleccionar “New”, luego “Java Class” y nombrar la clase como “ConversionService”.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Creación de la clase ConversionService en el paquete servicio

En el archivo “ConversionService”, implementar la lógica para la conversión de unidades, incluyendo los siguientes métodos.

1. Implementación de los métodos de conversión en ConversionService

|  |
| --- |
| Aw  package ec.edu.monster.servicio;  /\*\*  \* Service class for unit conversions.  \* @author MATIAS  \*/  public class ConversionService {  public double pulgadasACentimetros(double pulgadas) {  return pulgadas \* 2.54;  }  public double centimetrosAPulgadas(double centimetros) {  return centimetros / 2.54;  }  public double metrosAPies(double metros) {  return metros \* 3.28084;  }  public double piesAMetros(double pies) {  return pies / 3.28084;  }  public double metrosAYardas(double metros) {  return metros \* 1.09361;  }  public double yardasAMetros(double yardas) {  return yardas / 1.09361;  }  } |

Con los métodos desarrollados, integrarlos al Web Service “CONUNI” añadiendo lo siguiente:

1. Integración de los métodos de conversión en el Web Service CONUNI

|  |
| --- |
| package ec.edu.monster.controlador;  import jakarta.jws.WebService; // Cambia javax.jws por jakarta.jws  import jakarta.jws.WebMethod;  import jakarta.jws.WebParam;  import ec.edu.monster.servicio.ConversionService;  import ec.edu.monster.servicio.LoginService;  import ec.edu.monster.servicio.SumaService;  /\*\*  \*  \* @author MATIAS  \*/  @WebService(serviceName = "CONUNI")  public class CONUNI {  @WebMethod(operationName = "pulgadasACentimetros")  public double pulgadasACentimetros(@WebParam(name = "pulgadas") double pulgadas) {  ConversionService service = new ConversionService();  return service.pulgadasACentimetros(pulgadas);  }  @WebMethod(operationName = "centimetrosAPulgadas")  public double centimetrosAPulgadas(@WebParam(name = "centimetros") double centimetros) {  ConversionService service = new ConversionService();  return service.centimetrosAPulgadas(centimetros);  }  } |

Una vez implementado correctamente, ejecutar el proyecto para generar el archivo WSDL. Al hacerlo, se abrirá el navegador con la URL http://localhost:8080/CONUNI\_GR10/. A esta URL, añadir el nombre del servicio web “CONUNI”, resultando en http://localhost:8080/CONUNI\_GR10/CONUNI. Esto mostrará la página del servicio, desde donde se podrá copiar la URL del WSDL: http://localhost:8080/CONUNI\_GR10/CONUNI?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

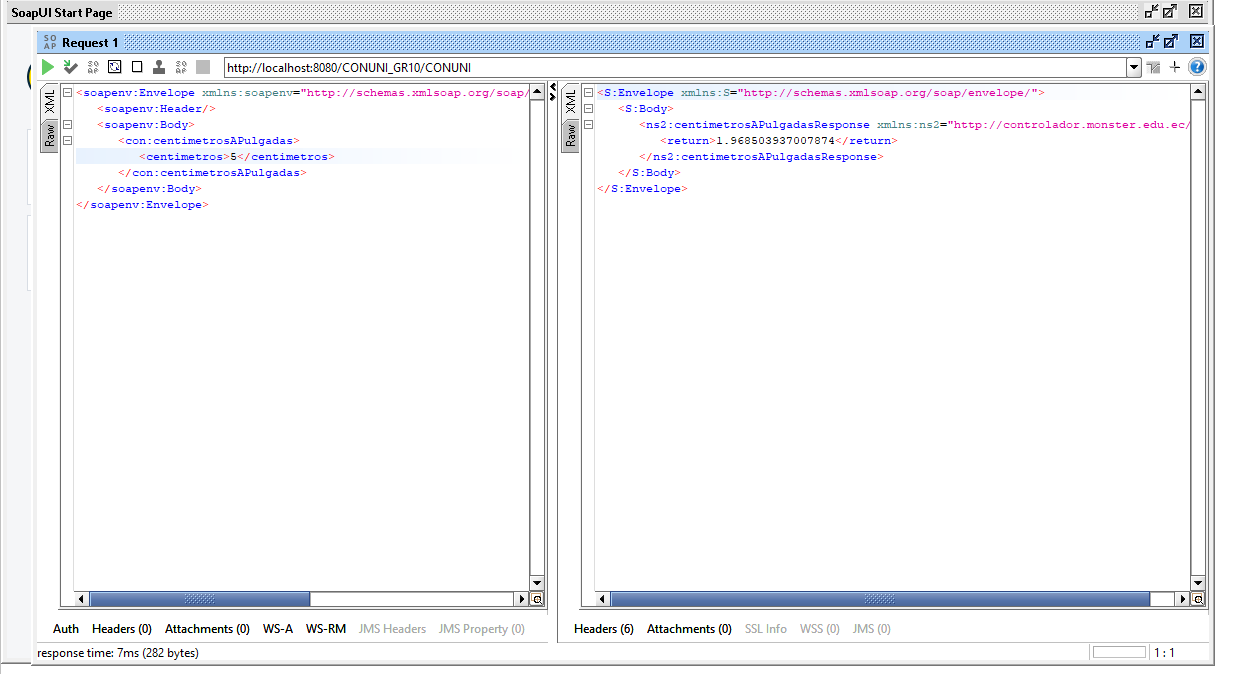
Para verificar el correcto funcionamiento del servicio web de conversiones, utilizar SOAP UI para probar la URL http://localhost:8080/CONUNI\_GR10/CONUNI?wsdl, evaluando las operaciones de conversión disponibles.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

1. Confoguración del servicio web en SOAP UI

Probar la operación “centímetros a pulgadas” en SOAP UI, verificando el correcto funcionamiento del servicio web.



1. Prueba de la operación centímetros a pulgadas en SOAP UI

Para agregar el servicio de Login, hacer clic derecho en el paquete “ec.edu.monster.servicio”, seleccionar “New” y luego “Java Class” para crear la clase “LoginService”. Implementar en esta clase el método de autenticación con datos predefinidos:

1. Creación e implementación del método autenticar en LoginService

|  |
| --- |
| package ec.edu.monster.servicio;  /\*\*  \*  \* @author MATIAS  \*/  public class LoginService {  public boolean autenticar(String user, String password) {  return user.equals("monster") && password.equals("monster9");  }  } |

Incorporar el método de autenticación al servicio web “CONUNI” añadiendo lo siguiente:

1. Integración del método autenticar en el Web Service CONUNI

|  |
| --- |
| 1. @WebMethod(operationName = "login") 2. public boolean login(@WebParam(name = "usuario") String usuario, @WebParam(name = "contraseña") String contraseña) { 3. LoginService service = new LoginService(); 4. return service.autenticar(usuario,contraseña);   } |

Ejecutar nuevamente el servicio web y utilizar SOAP UI para verificar el correcto funcionamiento de la operación de autenticación.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Ejecución y prueba de la operación autenticar en SOAP UI

## Cliente escritorio

### 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

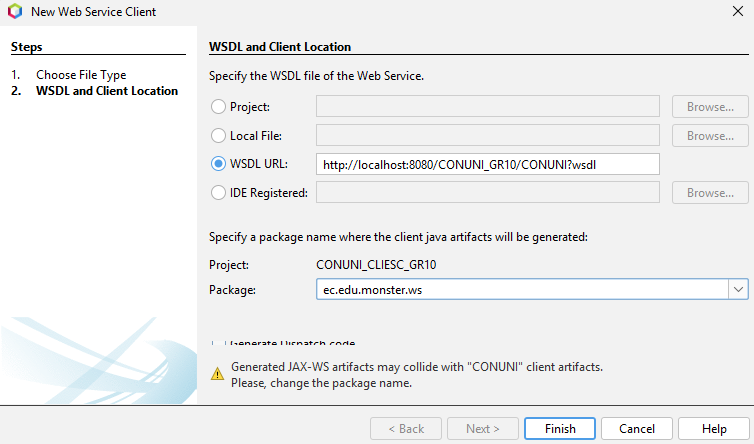
En la ventana emergente, asignar el nombre “CONUNI\_CLIESC\_GR10” al proyecto en el campo “Project Name”. Una vez creado, establecer una estructura adecuada implementando el patrón arquitectónico MVC. Para ello, crear en “Source Packages” los siguientes paquetes: ec.edu.monster.controlador ec.edu.monster.vista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

Texto

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/CONUNI\_GR10/CONUNI?wsdl. En el campo de paquete, especificar ec.edu.monster.ws



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

En el paquete “ec.edu.monster.servicio”, crear una nueva clase llamada “CONUNIService”. 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 CONUNIService y adición de operaciones del Web Service

|  |
| --- |
| package ec.edu.monster.servicio;  /\*\*  \*  \* @author MATIAS  \*/  public class CONUNIService {  private static double centimetrosAPulgadas(double centimetros) {  ec.edu.monster.ws.CONUNI\_Service service = new ec.edu.monster.ws.CONUNI\_Service();  ec.edu.monster.ws.CONUNI port = service.getCONUNIPort();  return port.centimetrosAPulgadas(centimetros);  }  private static double pulgadasACentimetros(double pulgadas) {  ec.edu.monster.ws.CONUNI\_Service service = new ec.edu.monster.ws.CONUNI\_Service();  ec.edu.monster.ws.CONUNI port = service.getCONUNIPort();  return port.pulgadasACentimetros(pulgadas);  }  private static double metrosAPies(double metros) {  ec.edu.monster.ws.CONUNI\_Service service = new ec.edu.monster.ws.CONUNI\_Service();  ec.edu.monster.ws.CONUNI port = service.getCONUNIPort();  return port.metrosAPies(metros);  }  private static double piesAMetros(double pies) {  ec.edu.monster.ws.CONUNI\_Service service = new ec.edu.monster.ws.CONUNI\_Service();  ec.edu.monster.ws.CONUNI port = service.getCONUNIPort();  return port.piesAMetros(pies);  }  private static double metrosAYardas(double metros) {  ec.edu.monster.ws.CONUNI\_Service service = new ec.edu.monster.ws.CONUNI\_Service();  ec.edu.monster.ws.CONUNI port = service.getCONUNIPort();  return port.metrosAYardas(metros);  }  private static double yardasAMetros(double yardas) {  ec.edu.monster.ws.CONUNI\_Service service = new ec.edu.monster.ws.CONUNI\_Service();  ec.edu.monster.ws.CONUNI port = service.getCONUNIPort();  return port.yardasAMetros(yardas);  }      } |

En el paquete “ec.edu.monster.servicio”, crear una nueva clase llamada “LoginService”. Hacer clic derecho sobre la clase, seleccionar “Insert Code” y luego “Call Web Service Operation”. A continuación, agregar la operación de autenticación del servicio web.

1. Creación de LoginService y adición de la operación de autenticación

|  |
| --- |
| package ec.edu.monster.servicio;  /\*\*  \*  \* @author MATIAS  \*/  public class LoginService {  private static boolean login(java.lang.String usuario, java.lang.String contraseña) {  ec.edu.monster.ws.CONUNI\_Service service = new ec.edu.monster.ws.CONUNI\_Service();  ec.edu.monster.ws.CONUNI port = service.getCONUNIPort();  return port.login(usuario, contraseña);  }    } |

Con los servicios creados, en el paquete “ec.edu.monster.controlador” crear una nueva clase llamada “AppControlador”. Implementar en esta clase los métodos correspondientes a los servicios “CONUNIService” y “LoginService”.

1. Creación e implementación de métodos en AppControlador

|  |
| --- |
| package ec.edu.monster.controlador;  import ec.edu.monster.servicio.CONUNIService;  import ec.edu.monster.servicio.LoginService;  /\*\*  \*  \* @author MATIAS  \*/  public class AppControlador {  public double pulgadasACentimetros(double pulgadas) {  CONUNIService service = new CONUNIService();  return service.pulgadasACentimetros(pulgadas);  }  public double centimetrosAPulgadas(double centimetros) {  CONUNIService service = new CONUNIService();  return service.centimetrosAPulgadas(centimetros);  }  public double metrosAPies(double metros) {  CONUNIService service = new CONUNIService();  return service.metrosAPies(metros);  }  public double piesAMetros(double pies) {  CONUNIService service = new CONUNIService();  return service.piesAMetros(pies);  }  public double metrosAYardas(double metros) {  CONUNIService service = new CONUNIService();  return service.metrosAYardas(metros);  }  public double yardasAMetros(double yardas) {  CONUNIService service = new CONUNIService();  return service.yardasAMetros(yardas);  }  public boolean login(String user, String password) {  LoginService service = new LoginService();  return service.login(user, password);  }  } |

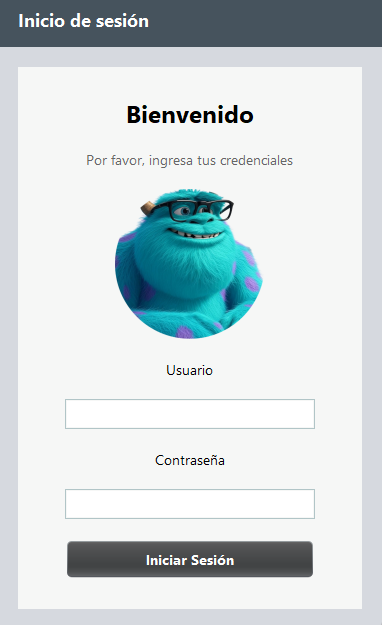
Para crear la interfaz de Login, hacer clic derecho en el paquete “ec.edu.monster.vista”, seleccionar “New” y luego “JFrame Form”. Nombrar el formulario “LoginVista”. Una vez creado, diseñar la interfaz gráfica correspondiente.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

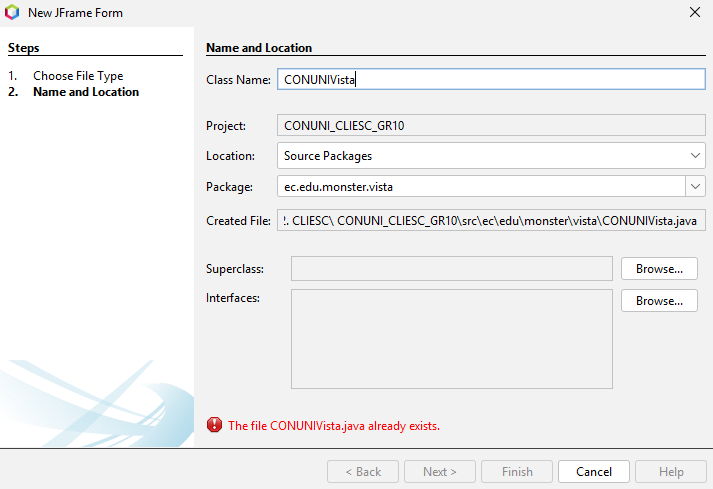
1. Creación y diseño de la interfaz LoginVista

La interfaz de “LoginVista” se diseña con un estilo minimalista y conciso, optimizado para el proceso de inicio de sesión.



1. Diseño minimalista de la interfaz LoginVista

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



1. Creación y diseño de la interfaz CONUNIVista

La interfaz de “CONUNIVista” se diseña con un estilo minimalista y conciso, optimizado para facilitar el proceso de conversión de unidades. Se realiza la prueba correcta del conversor de unidades.



1. Diseño minimalista de la interfaz CONUNIVista y prueba con éxito en el cliente de escritorio

## Cliente web

#### 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 “CONUNI\_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

|  |
| --- |
| <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>CONUNI\_CLI\_WEB\_GR10</artifactId>  <version>1.0-SNAPSHOT</version>  <packaging>war</packaging>  <name>CONUNI\_CLI\_WEB\_GR10-1.0-SNAPSHOT</name>    <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <jakartaee>10.0.0</jakartaee>  <cxf.version>4.0.3</cxf.version>  </properties>    <dependencies>  <dependency>  <groupId>jakarta.platform</groupId>  <artifactId>jakarta.jakartaee-api</artifactId>  <version>${jakartaee}</version>  <scope>provided</scope>  </dependency>  <!-- JAX-WS API para Jakarta EE -->  <dependency>  <groupId>jakarta.xml.ws</groupId>  <artifactId>jakarta.xml.ws-api</artifactId>  <version>4.0.0</version>  </dependency>  <!-- Implementación de JAX-WS (Metro) -->  <dependency>  <groupId>com.sun.xml.ws</groupId>  <artifactId>jaxws-rt</artifactId>  <version>4.0.0</version>  </dependency>  <!-- Dependencias de Apache CXF -->  <dependency>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-rt-frontend-jaxws</artifactId>  <version>${cxf.version}</version>  </dependency>  <dependency>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-rt-transports-http</artifactId>  <version>${cxf.version}</version>  </dependency>  </dependencies>    <build>  <finalName>CONUNI\_CLI\_WEB\_GR10</finalName>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.8.1</version>  <configuration>  <source>17</source> <!-- Actualizado a Java 17 para consistencia con el ejemplo -->  <target>17</target>  </configuration>  </plugin>  <!-- Plugin para empaquetar el WAR -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-war-plugin</artifactId>  <version>3.3.2</version>  </plugin>  <!-- Plugin para generar clases a partir del WSDL -->  <plugin>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-codegen-plugin</artifactId>  <version>${cxf.version}</version>  <executions>  <execution>  <id>generate-sources</id>  <phase>generate-sources</phase>  <configuration>  <sourceRoot>${project.basedir}/src/main/java</sourceRoot>  <wsdlOptions>  <!-- WSDL para CONUNI -->  <wsdlOption>  <wsdl>http://localhost:8080/CONUNI\_GR10/CONUNI?wsdl</wsdl>  <extraargs>  <extraarg>-verbose</extraarg>  <extraarg>-p</extraarg>  <extraarg>ec.edu.monster.ws</extraarg>  <extraarg>-client</extraarg> <!-- Genera solo el cliente -->  </extraargs>  </wsdlOption>  </wsdlOptions>  </configuration>  <goals>  <goal>wsdl2java</goal>  </goals>  </execution>  </executions>  <dependencies>  <!-- Agregar transporte HTTP como dependencia del plugin -->  <dependency>  <groupId>org.apache.cxf</groupId>  <artifactId>cxf-rt-transports-http</artifactId>  <version>${cxf.version}</version>  </dependency>  </dependencies>  </plugin>    </plugins>  </build>  </project> |

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

1. Creación e implementación de métodos en CONUNIService

|  |
| --- |
| package ec.edu.monster.servicio;  import ec.edu.monster.ws.CONUNI;  import ec.edu.monster.ws.CONUNI\_Service;  public class CONUNIService {  private final CONUNI port;  public CONUNIService() {  CONUNI\_Service service = new CONUNI\_Service();  this.port = service.getCONUNIPort();  }  public double pulgadasACentimetros(double pulgadas) {  return port.pulgadasACentimetros(pulgadas);  }  public double centimetrosAPulgadas(double centimetros) {  return port.centimetrosAPulgadas(centimetros);  }  public double metrosAPies(double metros) {  return port.metrosAPies(metros);  }  public double piesAMetros(double pies) {  return port.piesAMetros(pies);  }  public double metrosAYardas(double metros) {  return port.metrosAYardas(metros);  }  public double yardasAMetros(double yardas) {  return port.yardasAMetros(yardas);  }  public boolean login(String usuario, String contraseña) {  return port.login(usuario, contraseña);  }  } |

En el paquete “ec.edu.monster.controlador”, crear una nueva clase llamada “AppControlador” para gestionar la lógica de control del cliente web.

1. Creación de AppControlador en el paquete controlador

|  |
| --- |
| package ec.edu.monster.controlador;  import ec.edu.monster.servicio.CONUNIService;  public class AppControlador {  private final CONUNIService service;  public AppControlador() {  this.service = new CONUNIService();  }  public double pulgadasACentimetros(double pulgadas) {  return service.pulgadasACentimetros(pulgadas);  }  public double centimetrosAPulgadas(double centimetros) {  return service.centimetrosAPulgadas(centimetros);  }  public boolean login(String user, String password) {  return service.login(user, password);  }  public double metrosAPies(double metros) {  return service.metrosAPies(metros);  }  public double piesAMetros(double pies) {  return service.piesAMetros(pies);  }  public double metrosAYardas(double metros) {  return service.metrosAYardas(metros);  }  public double yardasAMetros(double yardas) {  return service.yardasAMetros(yardas);  }  } |

En el paquete “ec.edu.monster.controlador”, crear un nuevo servlet llamado “ConversorServlet” y configurarlo con los métodos correspondientes para gestionar las operaciones del servicio web.

1. Creación y configuración de ConversorServlet en el paquete controlador

|  |
| --- |
| package ec.edu.monster.controlador;  import java.io.IOException;  import jakarta.servlet.ServletException;  import jakarta.servlet.annotation.WebServlet;  import jakarta.servlet.http.HttpServlet;  import jakarta.servlet.http.HttpServletRequest;  import jakarta.servlet.http.HttpServletResponse;  @WebServlet(name = "ConversorServlet", urlPatterns = {"/convertir"})  public class ConversorServlet extends HttpServlet {  private final AppControlador controlador = new AppControlador();  @Override  protected void doPost(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  String accion = request.getParameter("accion");  try {  switch (accion) {  case "pulgadasACentimetros":  double pulgadas = Double.parseDouble(request.getParameter("pulgadas"));  if (pulgadas < 0) {  request.setAttribute("error", "Por favor, ingrese un valor no negativo.");  } else {  double centimetros = controlador.pulgadasACentimetros(pulgadas);  request.setAttribute("resultado", String.format("%.2f centímetros", centimetros));  }  break;  case "centimetrosAPulgadas":  double centimetros = Double.parseDouble(request.getParameter("centimetros"));  if (centimetros < 0) {  request.setAttribute("error", "Por favor, ingrese un valor no negativo.");  } else {  pulgadas = controlador.centimetrosAPulgadas(centimetros);  request.setAttribute("resultado", String.format("%.2f pulgadas", pulgadas));  }  break;  case "metrosAPies":  double metros = Double.parseDouble(request.getParameter("metros"));  if (metros < 0) {  request.setAttribute("error", "Por favor, ingrese un valor no negativo.");  } else {  double pies = controlador.metrosAPies(metros);  request.setAttribute("resultado", String.format("%.2f pies", pies));  }  break;  case "piesAMetros":  double pies = Double.parseDouble(request.getParameter("pies"));  if (pies < 0) {  request.setAttribute("error", "Por favor, ingrese un valor no negativo.");  } else {  metros = controlador.piesAMetros(pies);  request.setAttribute("resultado", String.format("%.2f metros", metros));  }  break;  case "metrosAYardas":  metros = Double.parseDouble(request.getParameter("metros"));  if (metros < 0) {  request.setAttribute("error", "Por favor, ingrese un valor no negativo.");  } else {  double yardas = controlador.metrosAYardas(metros);  request.setAttribute("resultado", String.format("%.2f yardas", yardas));  }  break;  case "yardasAMetros":  double yardas = Double.parseDouble(request.getParameter("yardas"));  if (yardas < 0) {  request.setAttribute("error", "Por favor, ingrese un valor no negativo.");  } else {  metros = controlador.yardasAMetros(yardas);  request.setAttribute("resultado", String.format("%.2f metros", metros));  }  break;  default:  request.setAttribute("error", "Acción no reconocida.");  }  } catch (NumberFormatException e) {  request.setAttribute("error", "Por favor, ingrese un número válido.");  } catch (Exception e) {  request.setAttribute("error", "Error al realizar la conversión: " + e.getMessage());  }  request.getRequestDispatcher("/conversor.jsp").forward(request, response);  }  } |

En el paquete “ec.edu.monster.controlador”, crear un nuevo servlet llamado “LoginServlet” para gestionar las funcionalidades de autenticación del cliente web.

1. Creación de LoginServlet en el paquete controlador

|  |
| --- |
| package ec.edu.monster.controlador;  import java.io.IOException;  import jakarta.servlet.ServletException;  import jakarta.servlet.annotation.WebServlet;  import jakarta.servlet.http.HttpServlet;  import jakarta.servlet.http.HttpServletRequest;  import jakarta.servlet.http.HttpServletResponse;  import jakarta.servlet.http.HttpSession;  @WebServlet(name = "LoginServlet", urlPatterns = {"/login"})  public class LoginServlet extends HttpServlet {  private final AppControlador controlador = new AppControlador();  @Override  protected void doPost(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  String user = request.getParameter("usuario");  String password = request.getParameter("contrasena");  if (user == null || user.isEmpty() || password == null || password.isEmpty()) {  request.setAttribute("error", "Por favor, complete todos los campos.");  request.getRequestDispatcher("/login.jsp").forward(request, response);  return;  }  try {  boolean success = controlador.login(user, password);  if (success) {  HttpSession session = request.getSession();  session.setAttribute("usuario", user);  response.sendRedirect("conversor.jsp");  } else {  request.setAttribute("error", "Usuario o contraseña incorrectos.");  request.getRequestDispatcher("/login.jsp").forward(request, response);  }  } catch (Exception e) {  request.setAttribute("error", "Error al iniciar sesión: " + e.getMessage());  request.getRequestDispatcher("/login.jsp").forward(request, response);  }  }  } |

En la carpeta “Web Pages”, crear un nuevo archivo JSP llamado “login.jsp” para diseñar la vista del formulario de inicio de sesión.

1. Creación del archivo login.jsp en Web Pages

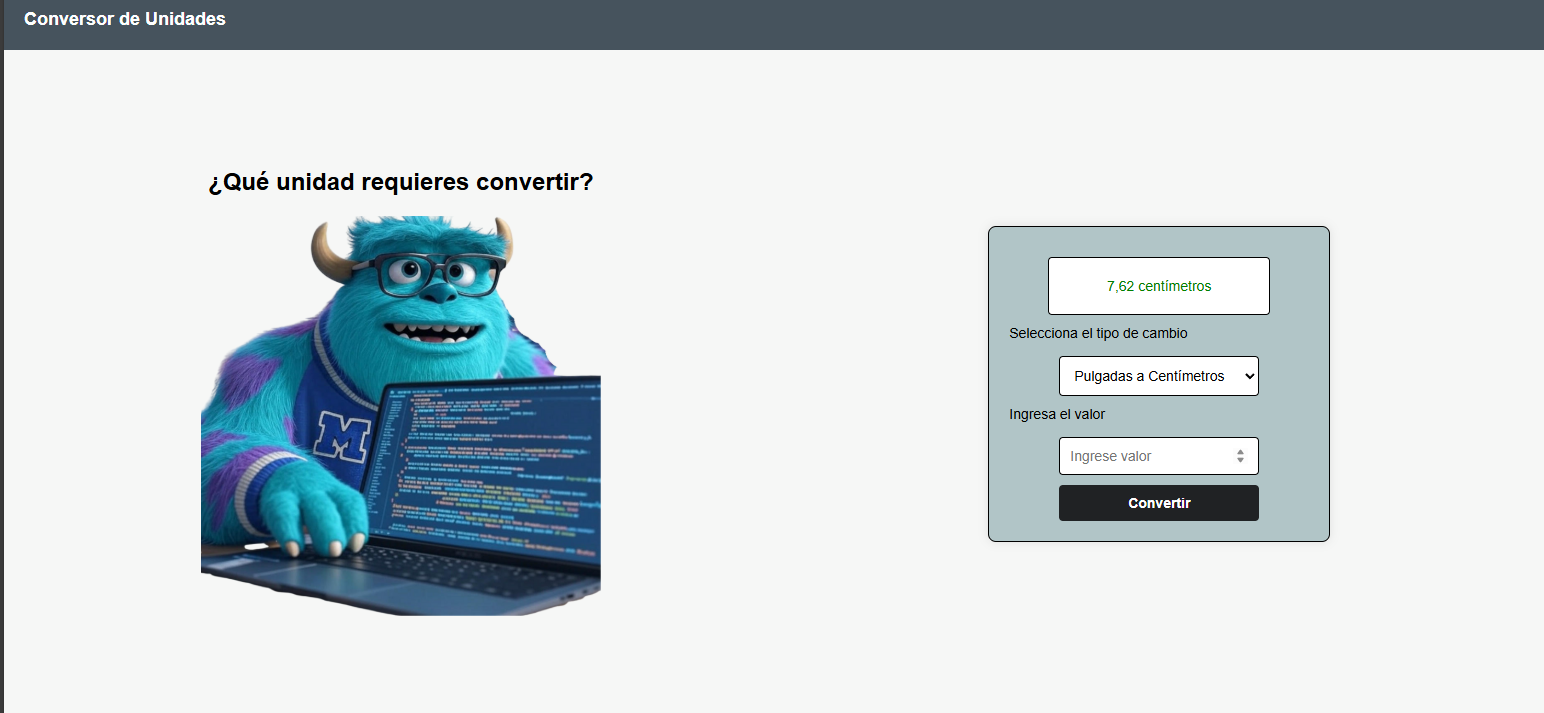
|  |
| --- |
| a <%@page contentType="text/html" pageEncoding="UTF-8"%>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Iniciar Sesión - CONUNI</title>  <style>  body {  font-family: Arial, sans-serif;  background-color: #f6f7f6;  display: flex;  justify-content: center;  align-items: center;  height: 100vh;  margin: 0;  }  .header {  background-color: #46535d;  color: white;  padding: 10px 20px;  font-family: Arial, sans-serif;  font-weight: bold;  font-size: 18px;  position: fixed;  top: 0;  width: 100%;  z-index: 1000;  }  .login-container {  background-color: #b1c5c7;  padding: 20px;  border: 1px solid black;  border-radius: 8px;  box-shadow: 0 0 10px rgba(0,0,0,0.1);  width: 300px;  text-align: center;  margin-top: 60px; /\* Espacio para el encabezado fijo \*/  }  .login-container h2 {  margin-bottom: 20px;  font-family: Arial, sans-serif;  font-weight: bold;  font-size: 24px;  }  .login-container img {  width: 150px;  height: 150px;  margin-bottom: 20px;  }  .login-container label {  font-family: Arial, sans-serif;  font-size: 14px;  display: block;  text-align: left;  margin-bottom: 5px;  }  .login-container input {  width: 100%;  padding: 10px;  margin: 10px 0;  border: 1px solid #b1c5c7;  border-radius: 4px;  box-sizing: border-box;  }  .login-container button {  width: 100%;  padding: 10px;  background-color: #202224;  border: none;  color: white;  border-radius: 4px;  cursor: pointer;  font-family: Arial, sans-serif;  font-size: 14px;  font-weight: bold;  }  .login-container button:hover {  background-color: #333;  }  .error {  color: red;  margin-bottom: 10px;  font-family: Arial, sans-serif;  font-size: 14px;  }  </style>  </head>  <body>  <div class="header">Inicio de Sesión</div>  <div class="login-container">  <h2>Bienvenido</h2>  <img src="images/sulley.png" alt="Sulley Avatar">  <p style="font-family: Arial, sans-serif; font-size: 14px; color: #666666;">Por favor, ingresa tus credenciales</p>  <% if (request.getAttribute("error") != null) { %>  <p class="error"><%= request.getAttribute("error") %></p>  <% } %>  <form action="login" method="post">  <label for="usuario">Usuario</label>  <input type="text" id="usuario" name="usuario" placeholder="Usuario" required>  <label for="contrasena">Contraseña</label>  <input type="password" id="contrasena" name="contrasena" placeholder="Contraseña" required>  <button type="submit">Iniciar Sesión</button>  </form>  </div>  </body>  </html> |

En la carpeta “Web Pages”, crear un nuevo archivo JSP llamado “conversor.jsp” para diseñar la vista del formulario de conversión de unidades.

1. Creación del archivo conversor.jsp en Web Pages

|  |
| --- |
| a <%@page contentType="text/html" pageEncoding="UTF-8"%>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Conversor de Unidades - CONUNI</title>  <style>  body {  font-family: Arial, sans-serif;  background-color: #f6f7f6;  margin: 0;  padding: 0;  height: 100vh;  display: flex;  flex-direction: column;  }  .header {  background-color: #46535d;  color: white;  padding: 20px 20px;  font-family: Arial, sans-serif;  font-weight: bold;  font-size: 18px;  position: fixed;  top: 0;  width: 100%;  z-index: 1000;  }  .container {  display: flex;  justify-content: center;  align-items: center;  gap: 5px; /\* Espacio reducido entre columnas, igual que en CONUNIVista \*/  flex-grow: 1;  margin-top: 50px; /\* Espacio para el encabezado fijo \*/  padding: 20px;  }  .left-column, .right-column {  display: flex;  justify-content: center;  align-items: center;  flex: 1;  }  .left-column-content {  text-align: center;  }  .left-column-content h2 {  font-family: Arial, sans-serif;  font-weight: bold;  font-size: 24px;  margin-bottom: 20px;  }  .left-column-content img {  width: 400px;  height: 400px;  }  .right-column-content {  background-color: #b1c5c7;  padding: 20px;  border: 1px solid black;  border-radius: 8px;  box-shadow: 0 0 10px rgba(0,0,0,0.1);  width: 300px;  text-align: center;  }  .right-column-content label {  font-family: Arial, sans-serif;  font-size: 14px;  display: block;  text-align: left;  margin-bottom: 5px;  }  .right-column-content select, .right-column-content input {  width: 200px;  padding: 10px;  margin: 10px 0;  border: 1px solid black;  border-radius: 4px;  box-sizing: border-box;  font-family: Arial, sans-serif;  font-size: 14px;  }  .right-column-content button {  width: 200px;  padding: 10px;  background-color: #202224;  border: none;  color: white;  border-radius: 4px;  cursor: pointer;  font-family: Arial, sans-serif;  font-size: 14px;  font-weight: bold;  }  .right-column-content button:hover {  background-color: #333;  }  .error, .resultado {  margin: 10px 0;  font-family: Arial, sans-serif;  font-size: 14px;  }  .error {  color: red;  }  .resultado {  color: green;  }  .result-container {  width: 200px;  padding: 10px;  border: 1px solid black;  border-radius: 4px;  background-color: white;  margin: 10px auto;  font-family: Arial, sans-serif;  font-size: 14px;  }  </style>  </head>  <body>  <div class="header">Conversor de Unidades</div>  <div class="container">  <div class="left-column">  <div class="left-column-content">  <h2>¿Qué unidad requieres convertir?</h2>  <img src="images/sulleyconuni.png" alt="Sulley">  </div>  </div>  <div class="right-column">  <div class="right-column-content">  <% if (request.getAttribute("error") != null) { %>  <p class="error"><%= request.getAttribute("error") %></p>  <% } %>  <% if (request.getAttribute("resultado") != null) { %>  <div class="result-container">  <p class="resultado"><%= request.getAttribute("resultado") %></p>  </div>  <% } %>  <form action="convertir" method="post">  <label for="tipoConversion">Selecciona el tipo de cambio</label>  <select id="tipoConversion" name="accion">  <option value="pulgadasACentimetros">Pulgadas a Centímetros</option>  <option value="centimetrosAPulgadas">Centímetros a Pulgadas</option>  <option value="metrosAPies">Metros a Pies</option>  <option value="piesAMetros">Pies a Metros</option>  <option value="metrosAYardas">Metros a Yardas</option>  <option value="yardasAMetros">Yardas a Metros</option>  </select>  <label for="valor">Ingresa el valor</label>  <input type="number" step="any" id="valor" name="valor" placeholder="Ingrese valor" required>  <button type="submit">Convertir</button>  </form>  </div>  </div>  </div>  </body>  </html> |

Para verificar el correcto funcionamiento del Cliente Web al probar la conversión de unidades, ejecutar el proyecto y acceder a la URL correspondiente (por ejemplo, http://localhost:8080/CONUNI\_CLIWEB\_GR10/). Navegar al formulario conversor.jsp, ingresar un valor para la conversión (como centímetros a pulgadas o viceversa) y enviar la solicitud. Validar que la respuesta generada por el servlet ConversorServlet sea correcta, comparando el resultado con el valor esperado según la lógica implementada en CONUNIService. Además, confirmar que no se presenten errores en la interfaz ni en el procesamiento de la solicitud.

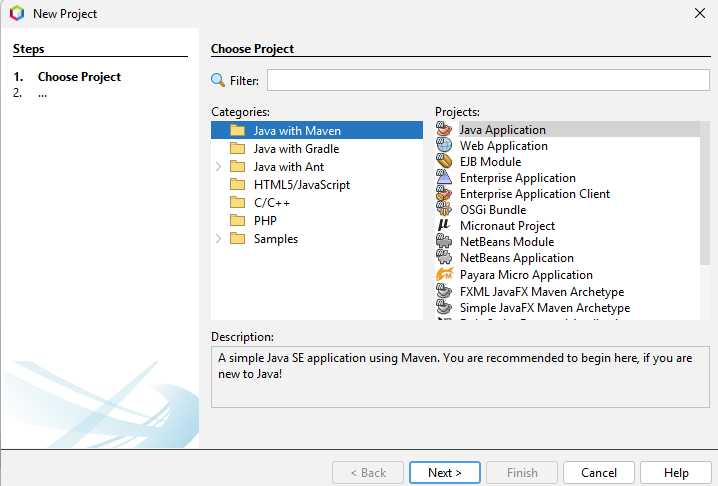


1. Verificación de la funcionalidad de conversión en el Cliente Web

## Cliente consola

#### 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 “CONUNI\_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 CONUNI\_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

Texto

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/CONUNI\_GR10/CONUNI?wsdl. Especificar el paquete como ec.edu.monster.

Interfaz de usuario gráfica, Texto, Aplicación

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 “CONUNIService” e implementar los métodos necesarios para interactuar con las operaciones del servicio web, como la conversión de unidades y autenticación.

1. Creación e implementación de métodos en CONUNIService

|  |
| --- |
| package ec.edu.monster.servicio;  /\*\*  \* Service class to interact with the CONUNI web service.  \* @author MATIAS  \*/  public class CONUNIService {  public double pulgadasACentimetros(double pulgadas) {  ec.edu.monster.CONUNI\_Service service = new ec.edu.monster.CONUNI\_Service();  ec.edu.monster.CONUNI port = service.getCONUNIPort();  return port.pulgadasACentimetros(pulgadas);  }  public double centimetrosAPulgadas(double centimetros) {  ec.edu.monster.CONUNI\_Service service = new ec.edu.monster.CONUNI\_Service();  ec.edu.monster.CONUNI port = service.getCONUNIPort();  return port.centimetrosAPulgadas(centimetros);  }  public boolean login(String usuario, String contraseña) {  ec.edu.monster.CONUNI\_Service service = new ec.edu.monster.CONUNI\_Service();  ec.edu.monster.CONUNI port = service.getCONUNIPort();  return port.login(usuario, contraseña);  }  public double metrosAPies(double metros) {  ec.edu.monster.CONUNI\_Service service = new ec.edu.monster.CONUNI\_Service();  ec.edu.monster.CONUNI port = service.getCONUNIPort();  return port.metrosAPies(metros);  }  public double piesAMetros(double pies) {  ec.edu.monster.CONUNI\_Service service = new ec.edu.monster.CONUNI\_Service();  ec.edu.monster.CONUNI port = service.getCONUNIPort();  return port.piesAMetros(pies);  }  public double metrosAYardas(double metros) {  ec.edu.monster.CONUNI\_Service service = new ec.edu.monster.CONUNI\_Service();  ec.edu.monster.CONUNI port = service.getCONUNIPort();  return port.metrosAYardas(metros);  }  public double yardasAMetros(double yardas) {  ec.edu.monster.CONUNI\_Service service = new ec.edu.monster.CONUNI\_Service();  ec.edu.monster.CONUNI port = service.getCONUNIPort();  return port.yardasAMetros(yardas);  }  } |

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

1. Creación e implementación de métodos en AppControlador

|  |
| --- |
| package ec.edu.monster.controlador;  import ec.edu.monster.servicio.CONUNIService;  /\*\*  \* Controller class to handle operations for the CONUNI client.  \* @author MATIAS  \*/  public class AppControlador {  private final CONUNIService service;  public AppControlador() {  this.service = new CONUNIService();  }  public double pulgadasACentimetros(double pulgadas) {  return service.pulgadasACentimetros(pulgadas);  }  public double centimetrosAPulgadas(double centimetros) {  return service.centimetrosAPulgadas(centimetros);  }  public double metrosAPies(double metros) {  return service.metrosAPies(metros);  }  public double piesAMetros(double pies) {  return service.piesAMetros(pies);  }  public double metrosAYardas(double metros) {  return service.metrosAYardas(metros);  }  public double yardasAMetros(double yardas) {  return service.yardasAMetros(yardas);  }  public boolean login(String user, String password) {  return service.login(user, password);  }  } |

## CONCLUSIONES

La arquitectura de software comparte principios fundamentales con la arquitectura tradicional, como la planificación meticulosa, la toma de decisiones estructurales y la orientación a satisfacer necesidades reales.

A su vez, así como un arquitecto de edificaciones requiere conocimientos técnicos y visión estructural, un arquitecto de software debe dominar la programación y comprender profundamente los requerimientos del sistema para construir soluciones funcionales, sostenibles y adaptables, incluso con recursos limitados.

Por otro lado, la arquitectura de software se compone de elementos clave tales como: componentes, conexiones, patrones, requisitos y documentación, que, en conjunto, definen su estructura, comportamiento y capacidad de adaptación.

Es importante contemplar que un arquitecto de software no solo debe dominar estos elementos, sino también anticipar cambios tecnológicos y del negocio, brindando soluciones flexibles

Por último, el modelado y diseño de la arquitectura de software es fundamental para visualizar la estructura, el comportamiento y la interacción de los componentes dentro del entorno donde se desplegará el sistema.

Modelos clave como el 4+1 vistas de Philippe Kruchten, permiten analizar el sistema desde diferentes ángulos, facilitando la comunicación y comprensión entre los distintos grupos involucrados. Al integrar el modelo 4+1 con UML, se logra una visión integral que asegura el diseño y la implementación efectiva de soluciones flexibles y escalables.

## RECOMENDACIONES

Es fundamental considerar que un arquitecto de software no debe centrarse solo en las necesidades actuales del sistema, sino que anticiparse a posibles cambios futuros tanto tecnológicos como comerciales. Esto garantiza que el sistema se mantenga relevante y eficiente a lo largo del tiempo, adaptándose a nuevas demandas sin necesidad de rediseños costosos.

Por otro lado, la arquitectura de software debe facilitar la comunicación clara entre todos los grupos involucrados en el proyecto, por lo que, es recomendable optar por modelos como el 4+1 vistas y UML, lo que asegura que todos los actores, desde desarrolladores hasta clientes, comprendan cómo se estructura y opera el sistema, lo que optimiza la colaboración y la toma de decisiones.

## BIBLIOGRAFÍA

Bass, L., Clements, P., & Kazman, R. (2016). *Software Architecture in Practice* (3.ª edición ed.). Boston, MA: Addison-Wesley.

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language User Guide* (Segunda edición (2nd ed.) ed.). Boston: Addison-Wesley.

Elizondo, P. V., Cervantes Maceda, H., & Castro Careaga, L. (2015). *Arquitectura de Software: Conceptos y Ciclo de Desarrollo.* México, D.F.: Cengage Learning Editores, S.A. de C.V.

Ken, A. (26 de 11 de 2023). *Arquitectura de software: ¿Qué es y qué tipos hay?* Obtenido de Gluo: https://www.gluo.mx/blog/arquitectura-de-software-que-es-y-que-tipos-hay

Kruchten, P. (1995). Architectural Blueprints—The “4+1” View Model of Software Architecture. *IEEE Software, 12*, 42–50. doi:https://doi.org/10.1109/52.469759

Ojeda Montoya, J. F. (13 de 9 de 2023). *Arquitectura de Software: Fundamentos, Características, Usos y Ejemplos.* Obtenido de Linkedin: https://es.linkedin.com/pulse/arquitectura-de-software-fundamentos-caracter%C3%ADsticas-y-ojeda-montoya