Imagen que contiene interior, mamífero, gato, viendo

Descripción generada automáticamente

**Tema**

CONVERSIÓN DE UNIDADES RESTFULL JAVA

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS.MDU.CCNA.CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

27/11/2024

**CONVERSIÓN DE UNIDADES REST JAVA**

[INTRODUCCIÓN 7](#_Toc183586508)

[PARTE PRÁCTICA 7](#_Toc183586509)

[OBJETIVO GENERAL. 7](#_Toc183586510)

[OBJETIVOS ESPECÍFICOS. 7](#_Toc183586511)

[PARTE PRÁCTICA 7](#_Toc183586512)

[INSTALACIÓN DE HERRAMIENTAS 7](#_Toc183586513)

[INSTALACIÓN DEL JDK 7](#_Toc183586514)

[APLICACIÓN SERVIDOR 8](#_Toc183586515)

[CREACIÓN DEL PROYECTO SERVIDOR 8](#_Toc183586516)

[CREACIÓN DE LA ARQUITECTURA DEL PROYECTO SERVIDOR 10](#_Toc183586517)

[PRUEBA SERVIDOR 29](#_Toc183586518)

[EJECUCIÓN DEL SERVICIO 32](#_Toc183586519)

[APLICACIÓN CLIENTE CONSOLA 35](#_Toc183586520)

[CREACIÓN DEL PROYECTO CLIENTE 35](#_Toc183586521)

[CREACIÓN DEL MODELO MVC 36](#_Toc183586522)

[IMPLEMENTACIÓN DEL SERVICIO 38](#_Toc183586523)

[PAQUETE MODELO 45](#_Toc183586524)

[PAQUETE CONTROLADOR 52](#_Toc183586525)

[PAQUETE VISTA 60](#_Toc183586526)

[EJECUCIÓN 66](#_Toc183586527)

[APLICACIÓN CLIENTE ESCRITORIO 66](#_Toc183586528)

[CREACIÓN DEL PROYECTO CLIENTE 66](#_Toc183586529)

[CREACIÓN DEL MODELO MVC 68](#_Toc183586530)

[IMPLEMENTACIÓN DEL SERVICIO 69](#_Toc183586531)

[PAQUETE MODELO 70](#_Toc183586532)

[PAQUETE CONTROLADOR 72](#_Toc183586533)

[PAQUETE VISTA 74](#_Toc183586534)

[EJECUCIÓN 76](#_Toc183586535)

[APLICACIÓN CLIENTE WEB 76](#_Toc183586536)

[CREACIÓN DEL PROYECTO 77](#_Toc183586537)

[CREACIÓN DEL MODELO MVC 77](#_Toc183586538)

[PAQUETE MODELO 78](#_Toc183586539)

[PAQUETE CONTROLADOR 81](#_Toc183586540)

[PAQUETE TEMPLATES 85](#_Toc183586541)

[ARCHIVO App.py 86](#_Toc183586542)

[EJECUCIÓN 89](#_Toc183586543)

[APLICACIÓN CLIENTE MÓVIL 90](#_Toc183586544)

[CREACIÓN DEL PROYECTO CLIENTE 90](#_Toc183586545)

[CREACIÓN DEL MODELO MVC 92](#_Toc183586546)

[PAQUETE SERVICIO 93](#_Toc183586547)

[PAQUETE MODELO 102](#_Toc183586548)

[PAQUETE CONTROLADOR 109](#_Toc183586549)

[PAQUETE VISTA 116](#_Toc183586550)

[EJECUCIÓN 127](#_Toc183586551)

[Conclusiones 128](#_Toc183586552)

[Recomendaciones 128](#_Toc183586553)

**ÍNDICE DE IMÁGENES**

[Figura 1. Descarga del JDK 8](#_Toc183586554)

[Figura 2: Creación del proyecto base 8](#_Toc183586555)

[Figura 3:Nombre y ubicación del proyecto base 9](#_Toc183586556)

[Figura 4: Selección y configuración de servidor 9](#_Toc183586557)

[Figura 5: No se selecciona framework de java 10](#_Toc183586558)

[Figura 6: Modelo MVC para el servicio 10](#_Toc183586559)

[Figura 7: Implementación del Web Service RESTful 11](#_Toc183586560)

[Figura 8: Selección del tipo de patrón del Web Service 11](#_Toc183586561)

[Figura 9: Especificación de los recursos de la clase 12](#_Toc183586562)

[Figura 10: Estructura de los servicios dentro del paquete servicios 12](#_Toc183586563)

[Figura 11: Archivo de configuración del Web Service 13](#_Toc183586564)

[Figura 12: Modelo del objeto de conversión 14](#_Toc183586565)

[Figura 13: Modelo del objeto de login 17](#_Toc183586566)

[Figura 14: Controlador de conversión. 20](#_Toc183586567)

[Figura 15: Controlador del Login 23](#_Toc183586568)

[Figura 16: Implementación del servicio de conversión 25](#_Toc183586569)

[Figura 17: Implementación del servicios de login 27](#_Toc183586570)

[Figura 18: Estructura final del servicio 29](#_Toc183586571)

[Figura 19: Implementación de una prueba de funcionamiento del servicio. 30](#_Toc183586572)

[Figura 20: Opción de prueba del servicio 32](#_Toc183586573)

[Figura 21: Configuración de prueba del servicio. 33](#_Toc183586574)

[Figura 22: Página principal de pruebas del servicio. 33](#_Toc183586575)

[Figura 23: Sección de prueba de un servicio en específico. 34](#_Toc183586576)

[Figura 24: Prueba del servicio con el objeto. 34](#_Toc183586577)

[Figura 25: Prueba fallida del servicio. 35](#_Toc183586578)

[Figura 26: Botón de creación de nuevo proyecto 35](#_Toc183586579)

[Figura 27: Selección del tipo de proyecto. 36](#_Toc183586580)

[Figura 28: Proyecto inicial para el cliente de consola. 36](#_Toc183586581)

[Figura 29: Creación de un nuevo paquete 37](#_Toc183586582)

[Figura 30: Creación de un paquete. 37](#_Toc183586583)

[Figura 31: Estructura MVC. 38](#_Toc183586584)

[Figura 32: Implementación de un cliente RESTful 38](#_Toc183586585)

[Figura 33: Nombre y localización del cliente RESTful 39](#_Toc183586586)

[Figura 34: Servicios disponibles en netbeans. 39](#_Toc183586587)

[Figura 35: Selección de servicios RESTful disponibles. 40](#_Toc183586588)

[Figura 36: Archivos del cliente RESTful de los servicios implementados. 40](#_Toc183586589)

[Figura 37: Archivo de cliente RESTful para el Login. 41](#_Toc183586590)

[Figura 38: Archivo de cliente RESTful para la conversión. 43](#_Toc183586591)

[Figura 39: Modelo de los objetos de conversión. 45](#_Toc183586592)

[Figura 40: Modelo de objetos del Login 49](#_Toc183586593)

[Figura 41: Estructura final del paquete modelos. 52](#_Toc183586594)

[Figura 42: Controlador de conversiones. 53](#_Toc183586595)

[Figura 43: Controlador de Login 56](#_Toc183586596)

[Figura 44: Vista de cliente de consola 60](#_Toc183586597)

[Figura 45: Ejecución del cliente de consola. 66](#_Toc183586598)

[Figura 46: Selección de proyecto 67](#_Toc183586599)

[Figura 47: Configuración del proyecto 67](#_Toc183586600)

[Figura 48: Creación de nuevo paquete 68](#_Toc183586601)

[Figura 49: Nombrar paquetes 68](#_Toc183586602)

[Figura 50: Estructura final del proyecto 69](#_Toc183586603)

[Figura 51: Crear nuevo archivo en paquete 69](#_Toc183586604)

[Figura 52: Selección de RESTful Java Client 70](#_Toc183586605)

[Figura 53: LoginModel 71](#_Toc183586606)

[Figura 54: ConversionModel 72](#_Toc183586607)

[Figura 55: LoginController 73](#_Toc183586608)

[Figura 56: ConversionController 74](#_Toc183586609)

[Figura 57: Crear Jframe Form 74](#_Toc183586610)

[Figura 58: LoginView 75](#_Toc183586611)

[Figura 59: Clean and Build 76](#_Toc183586612)

[Figura 60: Ejecutar Proyecto 76](#_Toc183586613)

[Figura 61: Carpeta Entorno Virtual 77](#_Toc183586614)

[Figura 62: Estructura final del proyecto 78](#_Toc183586615)

[Figura 63: LoginModel 78](#_Toc183586616)

[Figura 64: ConversionModel 79](#_Toc183586617)

[Figura 65: LoginController 81](#_Toc183586618)

[Figura 66: ConversionController 82](#_Toc183586619)

[Figura 67: LoginTemplate 85](#_Toc183586620)

[Figura 68: ConversionTemplate 85](#_Toc183586621)

[Figura 69: Ruta Login 86](#_Toc183586622)

[Figura 70: Ruta convert 86](#_Toc183586623)

[Figura 71: Pantalla Login 89](#_Toc183586624)

[Figura 72: Pantalla Conversión 89](#_Toc183586625)

[Figura 73: Ventana principal de Android Studio. 90](#_Toc183586626)

[Figura 74: Tipo de proyecto de Android Studio. 91](#_Toc183586627)

[Figura 75: Nombre y ubicación del proyecto. 91](#_Toc183586628)

[Figura 76: Proyecto vacío de Android Studio. 92](#_Toc183586629)

[Figura 77: Creación de un paquete 92](#_Toc183586630)

[Figura 78: Asignación de nombre al paquete 93](#_Toc183586631)

[Figura 79: Estructura final de paquetes con MVC. 93](#_Toc183586632)

[Figura 80: Propiedades del proyecto Android Studio. 93](#_Toc183586633)

[Figura 81: Configuración de las dependencias de Android Studio. 94](#_Toc183586634)

[Figura 82: Sincronización de las configuraciones. 97](#_Toc183586635)

[Figura 83: Creación del cliente retrofit. 97](#_Toc183586636)

[Figura 84: Asignación del nombre de la nueva clase. 97](#_Toc183586637)

[Figura 85: Implementación de la dependencia Retrofit. 98](#_Toc183586638)

[Figura 86: Interfaz de consumo del servicio. 100](#_Toc183586639)

[Figura 87: Modelo del objeto Login 102](#_Toc183586640)

[Figura 88: Modelo del objeto de conversión. 105](#_Toc183586641)

[Figura 89: Controlador de conversión. 109](#_Toc183586642)

[Figura 90: Controlador del Login 111](#_Toc183586643)

[Figura 91: Clase para validar el hash de la contraseña. 114](#_Toc183586644)

[Figura 92: Vista principal del proyecto. 116](#_Toc183586645)

[Figura 93:Vista de la sección de conversión del proyecto. 118](#_Toc183586646)

[Figura 94: Diseño de la pantalla de Login. 121](#_Toc183586647)

[Figura 95: Diseño de la página de conversión. 124](#_Toc183586648)

[Figura 96: Botón para ejecutar el cliente de móvil. 127](#_Toc183586649)

[Figura 97: Ejecución del cliente de móvil. 128](#_Toc183586650)

**ÍNDICE DE TABLAS**

[Tabla 1: Modelo de Conversión 16](#_Toc183586468)

[Tabla 2: Modelo de Login 19](#_Toc183586469)

[Tabla 3: Código del controlador de conversión 22](#_Toc183586470)

[Tabla 4: Código del controlador de Login 25](#_Toc183586471)

[Tabla 5: Código del servicio conversión 27](#_Toc183586472)

[Tabla 6: Código del servcio Login 29](#_Toc183586473)

[Tabla 7: Código de prueba 32](#_Toc183586474)

[Tabla 8: Código generado por el cliente RESTful 43](#_Toc183586475)

[Tabla 9: Código del cliente RESTful de conversión. 45](#_Toc183586476)

[Tabla 10: Código del modelo de objetos de la conversión. 47](#_Toc183586477)

[Tabla 11: Código del modelo de objetos del Login 51](#_Toc183586478)

[Tabla 12: Código del controlador de conversiones. 55](#_Toc183586479)

[Tabla 13: Código de controlador del Login. 58](#_Toc183586480)

[Tabla 14: Código de la vista del cliente. 62](#_Toc183586481)

[Tabla 15: Código LoginView 76](#_Toc183586482)

[Tabla 16: LoginModel 81](#_Toc183586483)

[Tabla 17: ConversionModel 81](#_Toc183586484)

[Tabla 18: LoginController 84](#_Toc183586485)

[Tabla 19: ConversionController 85](#_Toc183586486)

[Tabla 20: Archivo App.py 88](#_Toc183586487)

[Tabla 21: Código de configuración de Android Studio. 96](#_Toc183586488)

[Tabla 22: Código de implementación de Retrofit 100](#_Toc183586489)

[Tabla 23:Código de la interfaz del servicio. 102](#_Toc183586490)

[Tabla 24: Código del modelo de objeto Login. 104](#_Toc183586491)

[Tabla 25: Código del modelo de objeto conversión. 107](#_Toc183586492)

[Tabla 26: Condigo del controlador de conversión. 111](#_Toc183586493)

[Tabla 27: Código del controlador de Login. 113](#_Toc183586494)

[Tabla 28: Código de la clase de validación de contraseña. 116](#_Toc183586495)

[Tabla 29: Código de la vista principal del cliente. 118](#_Toc183586496)

[Tabla 30: Código de la vista de conversión del cliente. 120](#_Toc183586497)

[Tabla 31: Código xml del diseño de la pantalla Login. 123](#_Toc183586498)

[Tabla 32: Código xml del diseño de la pantalla conversión. 126](#_Toc183586499)

# INTRODUCCIÓN

Este proyecto de investigación y desarrollo se centra en la creación de servicios RESTful con Java, una técnica ampliamente empleada en la construcción de aplicaciones web modernas y escalables. Su objetivo principal es estudiar los principios básicos de los servicios RESTful y su implementación en Java, con el propósito de aprovechar sus ventajas en la construcción de aplicaciones web eficientes y fácilmente integrables.

# PARTE PRÁCTICA

## OBJETIVO GENERAL.

Analizar y comprender los conceptos fundamentales de los servicios RESTful y su uso en el desarrollo de aplicaciones web utilizando Java.

## OBJETIVOS ESPECÍFICOS.

* Diseñar y establecer las rutas y recursos necesarios para los servicios RESTful, seleccionando los métodos HTTP más apropiados para cada operación.
* Desarrollar las funcionalidades de los servicios RESTful en Java, garantizando un manejo eficiente de solicitudes y respuestas.

# PARTE PRÁCTICA

Para la realización de la parte práctica de este proyecto, es necesario realizar los siguientes pasos detallados a continuación y adaptarlo a las necesidades de cada individuo.

## INSTALACIÓN DE HERRAMIENTAS

### INSTALACIÓN DEL JDK

En primer lugar, se ingresa al enlace <https://www.oracle.com/java/technologies/downloads/> para descargar el JDK 8u371.



Figura 1. Descarga del JDK

## APLICACIÓN SERVIDOR

### CREACIÓN DEL PROYECTO SERVIDOR

Para iniciar con el proyecto servidor, es necesario hacer clic en File/New Project y crear un proyecto web de java.

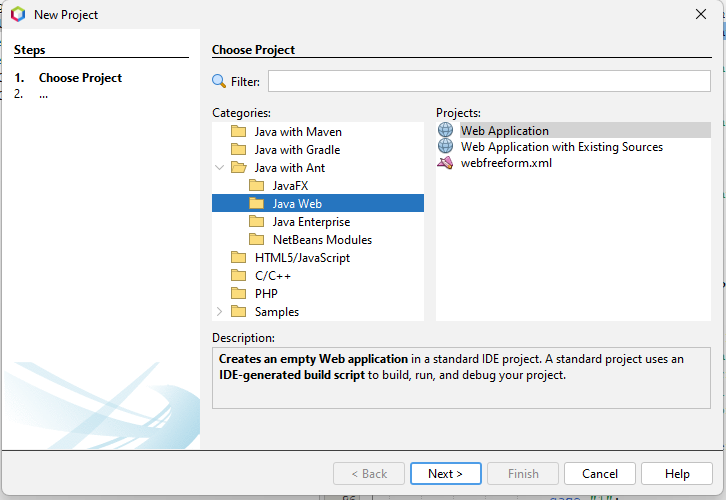


Figura 2: Creación del proyecto base

Se le da click a next y luego se establece el nombre y la ubicación del proyecto.

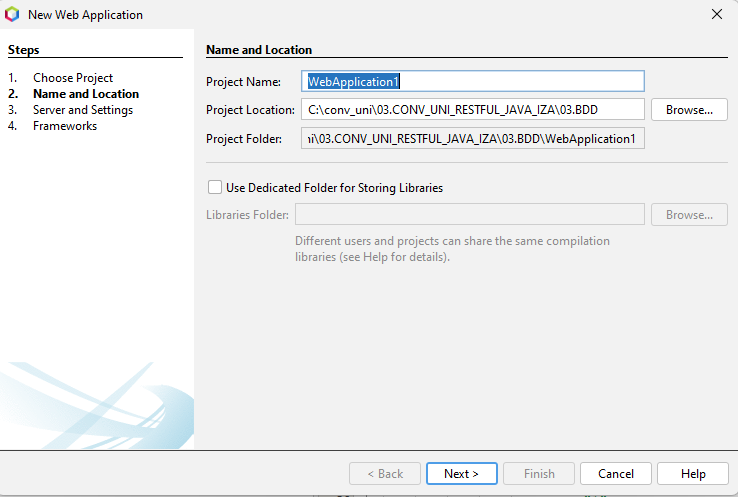


Figura 3:Nombre y ubicación del proyecto base

Luego de esto se debe seleccionar el servidor que se va a utilizar para el servicio web, en este caso se utilizará payara server con Jakarata EE 10 Web.

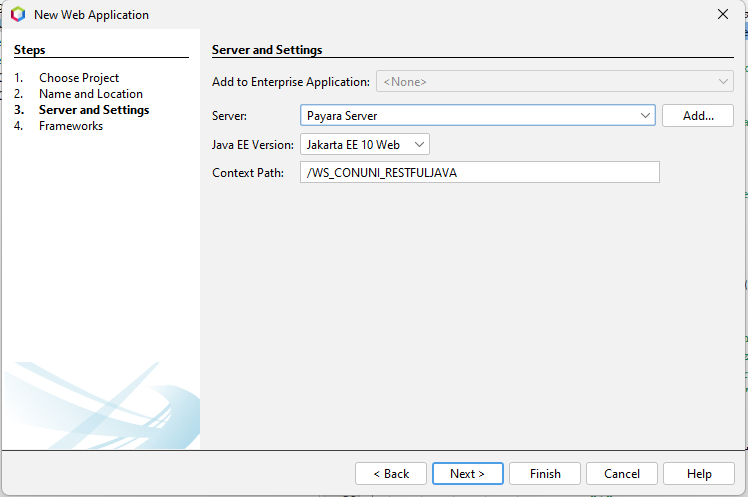


Figura 4: Selección y configuración de servidor

Luego de esto aparecerá la ventana de frameworks en la cual no se debe seleccionar nada.

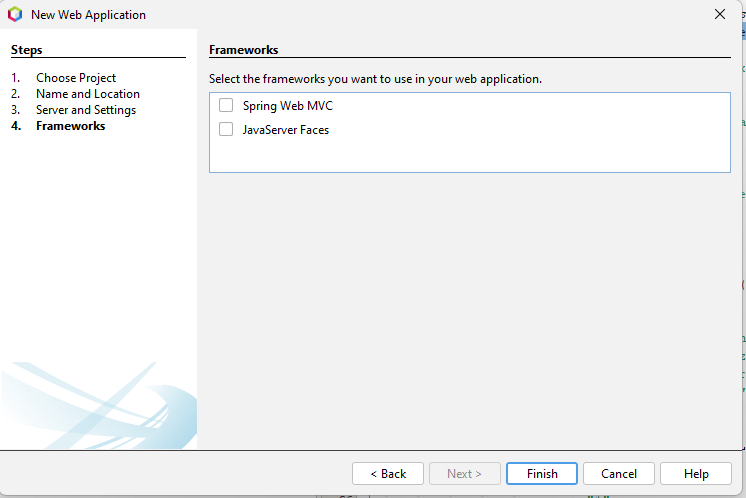


Figura 5: No se selecciona framework de java

### CREACIÓN DE LA ARQUITECTURA DEL PROYECTO SERVIDOR

Dentro de la carpeta “Source Packages” se crea la estructura de carpetas correspondiente al patrón MVC. Los 4 paquetes son: controlador, modelos, pruebas y por último el servicio.

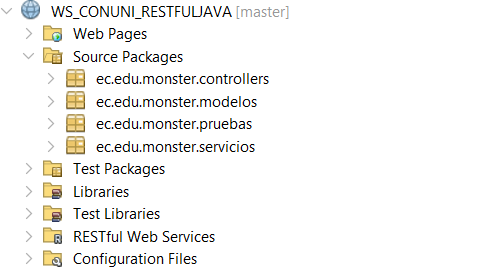


Figura 6: Modelo MVC para el servicio

Luego de esto se creó el servicio RESTful dentro del paquete servicios, esto se lo realiza al implementar un servicio web RESTful.

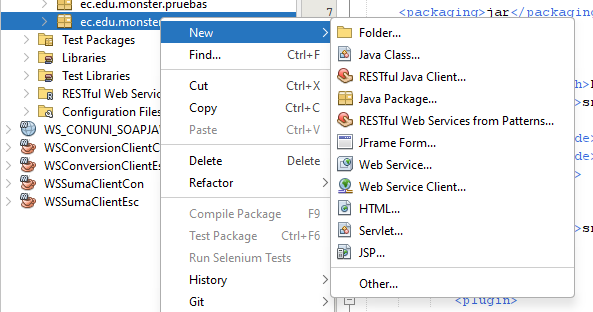


Figura 7: Implementación del Web Service RESTful

Luego de esto aparecen los siguientes pasos, en la primera venta se escoge la primera opción que viene por default.

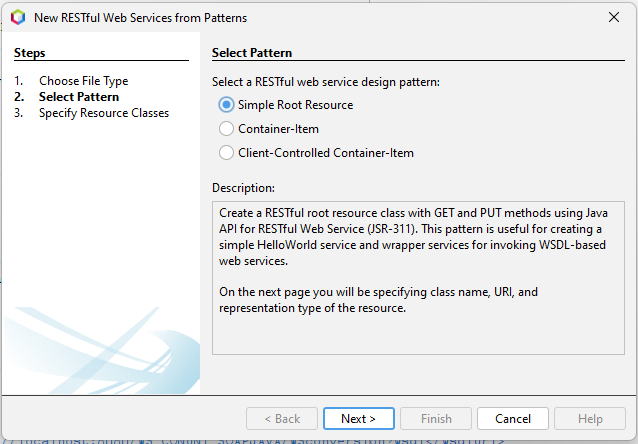


Figura 8: Selección del tipo de patrón del Web Service

Luego se debe verificar el nombre del paquete donde se va a implementar el servicios, así como el path de la dirección que tendrá el servicio. Finalmente se debe especificar el tipo de resultado que se espera de la peticiones y la clase que tendrá el mismo.

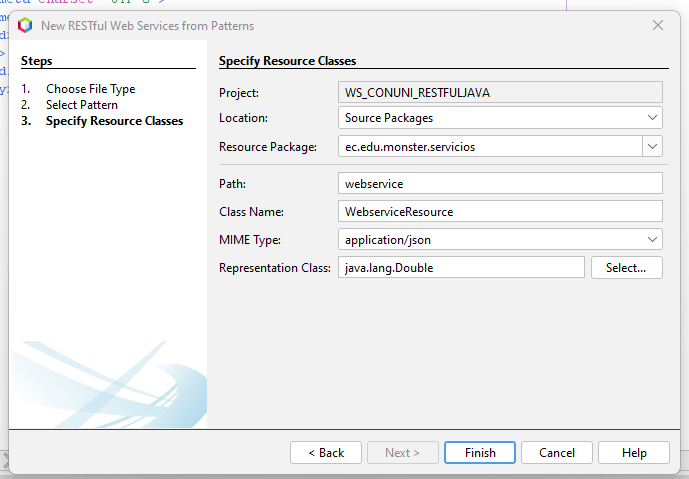


Figura 9: Especificación de los recursos de la clase

Debido a la forma en la que se crearon los servicios en este proyecto se implementaron dos web services, el primero para el Login llamado LoginResource y el segundo para la conversión, ConversionResource.

A computer screen shot of a computer

Description automatically generated

Figura 10: Estructura de los servicios dentro del paquete servicios

El tercer archivo se creó de manera automática y este guarda la configuración del servicio y cada vez que se le añada un servicio nuevo este se actualizará para que se pueda acceder al servicio.

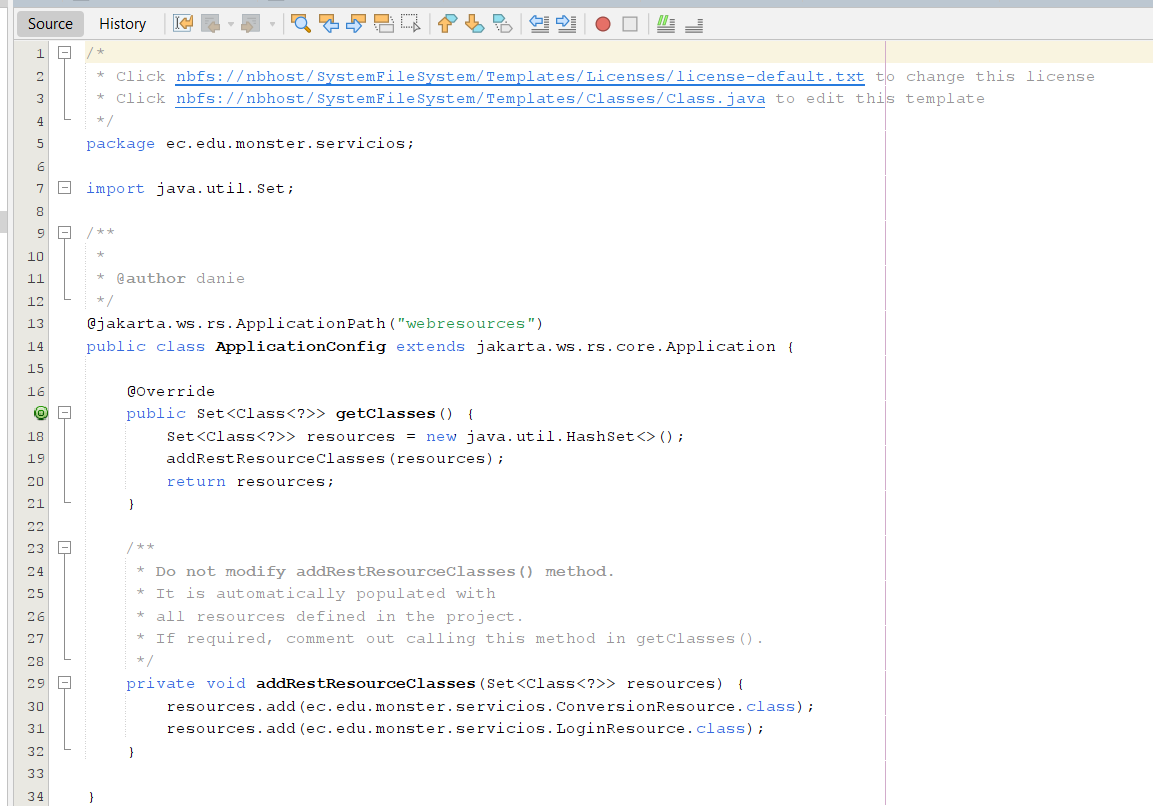


Figura 11: Archivo de configuración del Web Service

Para la implementación de la lógica del servicio se hizo uso de la estructura mvc en donde la sección de modelos se colocaron las definiciones de los objetos que se utilizarían en el web service.

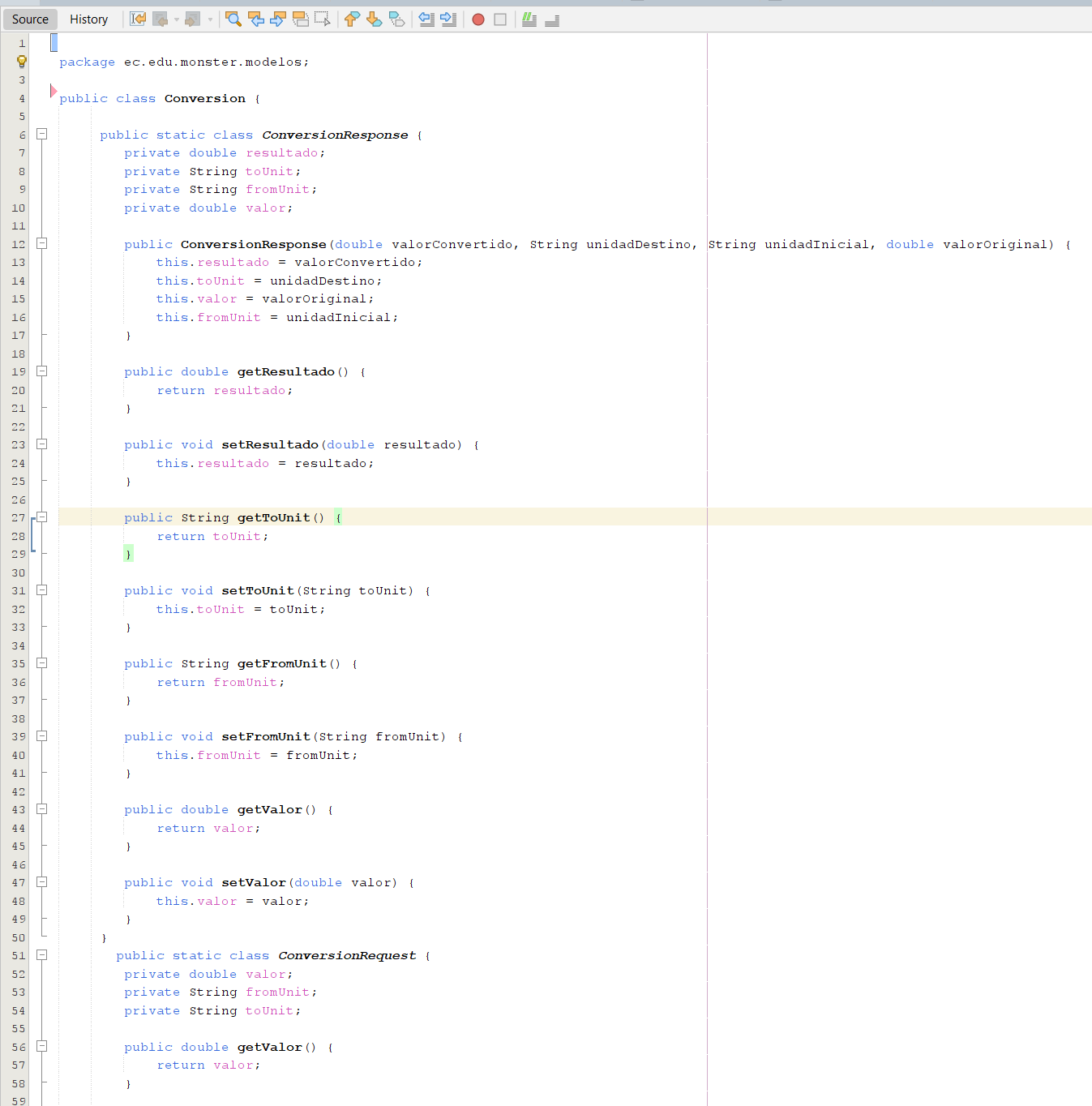


Figura 12: Modelo del objeto de conversión

Tabla 1: Modelo de Conversión

|  |
| --- |
| 1. package ec.edu.monster.modelos;  2.  3. public class Conversion {  4.  5. public static class ConversionResponse {  6. private double resultado;  7. private String toUnit;  8. private String fromUnit;  9. private double valor;  10.  11. public ConversionResponse(double valorConvertido, String unidadDestino, String unidadInicial, double valorOriginal) {  12. this.resultado = valorConvertido;  13. this.toUnit = unidadDestino;  14. this.valor = valorOriginal;  15. this.fromUnit = unidadInicial;  16. }  17.  18. public double getResultado() {  19. return resultado;  20. }  21.  22. public void setResultado(double resultado) {  23. this.resultado = resultado;  24. }  25.  26. public String getToUnit() {  27. return toUnit;  28. }  29.  30. public void setToUnit(String toUnit) {  31. this.toUnit = toUnit;  32. }  33.  34. public String getFromUnit() {  35. return fromUnit;  36. }  37.  38. public void setFromUnit(String fromUnit) {  39. this.fromUnit = fromUnit;  40. }  41.  42. public double getValor() {  43. return valor;  44. }  45.  46. public void setValor(double valor) {  47. this.valor = valor;  48. }  49. }  50. public static class ConversionRequest {  51. private double valor;  52. private String fromUnit;  53. private String toUnit;  54.  55. public double getValor() {  56. return valor;  57. }  58.  59. public void setValor(double valor) {  60. this.valor = valor;  61. }  62.  63. public String getFromUnit() {  64. return fromUnit;  65. }  66.  67. public void setFromUnit(String fromUnit) {  68. this.fromUnit = fromUnit;  69. }  70.  71. public String getToUnit() {  72. return toUnit;  73. }  74.  75. public void setToUnit(String toUnit) {  76. this.toUnit = toUnit;  77. }  78. }  79. }  80. |

Adicionalmente se creó también un modelo para los objetos de autenticación.

A screenshot of a computer

Description automatically generated

Figura 13: Modelo del objeto de login

Tabla 2: Modelo de Login

|  |
| --- |
| 1. package ec.edu.monster.modelos;  2.  3. public class Login {  4. public static class ConversionRequestL {  5.  6. private String username;  7. private String password;  8.  9. public String getUsername() {  10. return username;  11. }  12.  13. public void setUsername(String username) {  14. this.username = username;  15. }  16.  17. public String getPassword() {  18. return password;  19. }  20.  21. public void setPassword(String password) {  22. this.password = password;  23. }  24.  25. }  26.  27. public static class ConversionResponseL{  28. private String username;  29. private String password;  30. public ConversionResponseL(String username, String password){  31. this.username = username;  32. this.password = password;  33. }  34.  35. public String getUsername() {  36. return username;  37. }  38.  39. public void setUsername(String username) {  40. this.username = username;  41. }  42.  43. public String getPassword() {  44. return password;  45. }  46.  47. public void setPassword(String password) {  48. this.password = password;  49. }  50. }}  51. |

Luego de la creación de los modelos de objetos que se van a enviar y recibir a través del servicio se puede realizar la lógica implementando estos modelos dentro de los controladores. Se creó 2 controladores, uno para cada servicio.

El primer controlador se encarga de la selección de unidades y conversión de las mismas, contiene los valores entre unidades y la lógica de la conversión.

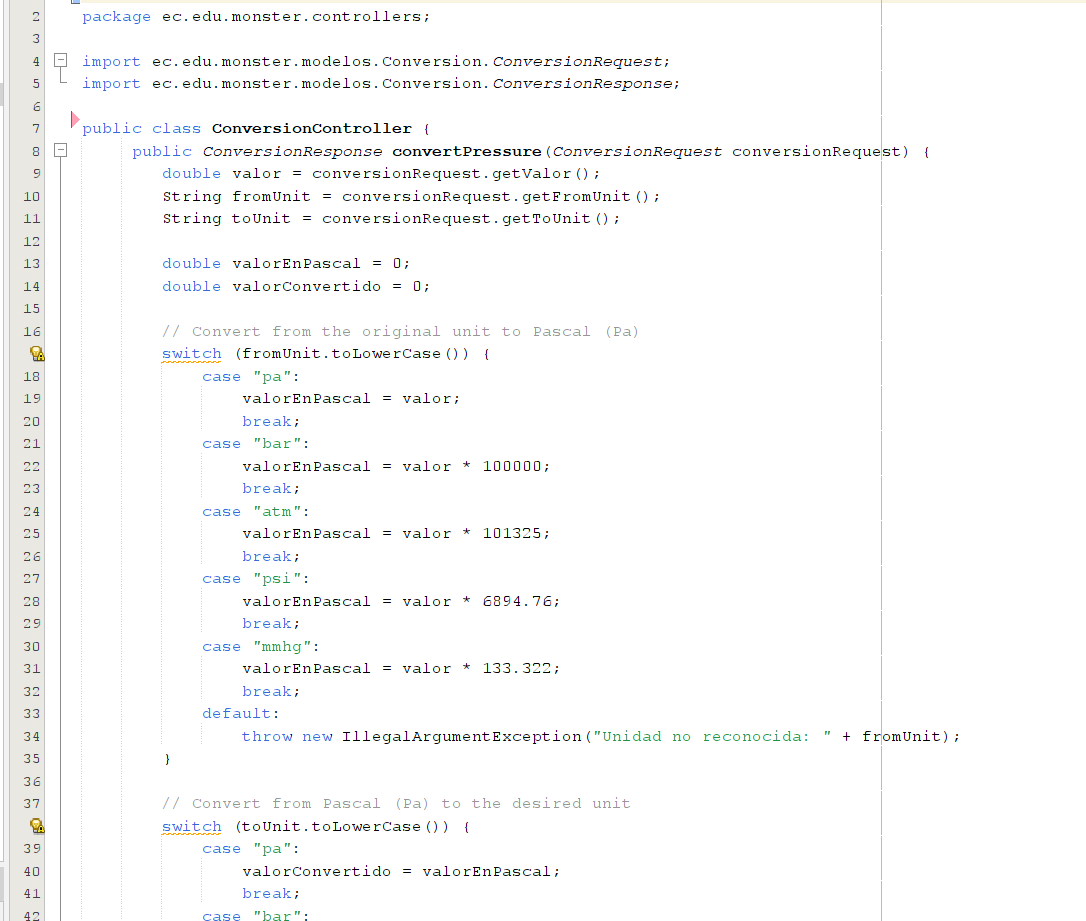


Figura 14: Controlador de conversión.

Este controlador hace uso de los modelos de objetos creados previamente en la sección de modelos.

Tabla 3: Código del controlador de conversión

|  |
| --- |
| 1. package ec.edu.monster.controllers;  2.  3. import ec.edu.monster.modelos.Conversion.ConversionRequest;  4. import ec.edu.monster.modelos.Conversion.ConversionResponse;  5.  6. public class ConversionController {  7. public ConversionResponse convertPressure(ConversionRequest conversionRequest) {  8. double valor = conversionRequest.getValor();  9. String fromUnit = conversionRequest.getFromUnit();  10. String toUnit = conversionRequest.getToUnit();  11.  12. double valorEnPascal = 0;  13. double valorConvertido = 0;  14.  15. // Convert from the original unit to Pascal (Pa)  16. switch (fromUnit.toLowerCase()) {  17. case "pa":  18. valorEnPascal = valor;  19. break;  20. case "bar":  21. valorEnPascal = valor \* 100000;  22. break;  23. case "atm":  24. valorEnPascal = valor \* 101325;  25. break;  26. case "psi":  27. valorEnPascal = valor \* 6894.76;  28. break;  29. case "mmhg":  30. valorEnPascal = valor \* 133.322;  31. break;  32. default:  33. throw new IllegalArgumentException("Unidad no reconocida: " + fromUnit);  34. }  35.  36. // Convert from Pascal (Pa) to the desired unit  37. switch (toUnit.toLowerCase()) {  38. case "pa":  39. valorConvertido = valorEnPascal;  40. break;  41. case "bar":  42. valorConvertido = valorEnPascal / 100000;  43. break;  44. case "atm":  45. valorConvertido = valorEnPascal / 101325;  46. break;  47. case "psi":  48. valorConvertido = valorEnPascal / 6894.76;  49. break;  50. case "mmhg":  51. valorConvertido = valorEnPascal / 133.322;  52. break;  53. default:  54. throw new IllegalArgumentException("Unidad no reconocida: " + toUnit);  55. }  56.  57. return new ConversionResponse(valorConvertido, toUnit, fromUnit, valor);  58. }  59. }  60. |

Luego se puede observar el controlador de la autenticación, el cual debe comprobar que el usuario y la contraseña sea la adecuada. En este caso se utiliza una contraseña que ha sido convertida a hash utilizando SHA-256.



Figura 15: Controlador del Login

Tabla 4: Código del controlador de Login

|  |
| --- |
| 1. package ec.edu.monster.controllers;  2.  3. import ec.edu.monster.modelos.Login.ConversionRequestL;  4. import ec.edu.monster.modelos.Login.ConversionResponseL;  5.  6. public class LoginController {  7.  8.  9. public ConversionResponseL login(ConversionRequestL conversionRequest) {  10. String username = conversionRequest.getUsername();  11. String password = conversionRequest.getPassword();  12. if(username.equals("monster") &&( password.equals( "774e993500f4027acfd72b7a7ee564b76ae43cf7c4c943ed0e0f364cca16b6ec") )){  13. ConversionResponseL resultado = new ConversionResponseL(username,password);  14. return resultado;  15. }else{  16. ConversionResponseL resultado = new ConversionResponseL("null","null");  17. return resultado;  18. }  19. }  20.  21. }  22. |

Una vez se han implementado tanto los modelos de objetos de los servicios y los controladores que manejan la lógica del servicio, se pueden retornar a la configuración de los servicios para poder ser utilizados.

Dentro del archivo ConversionResource se establece el método POST para enviar información por el servicio.

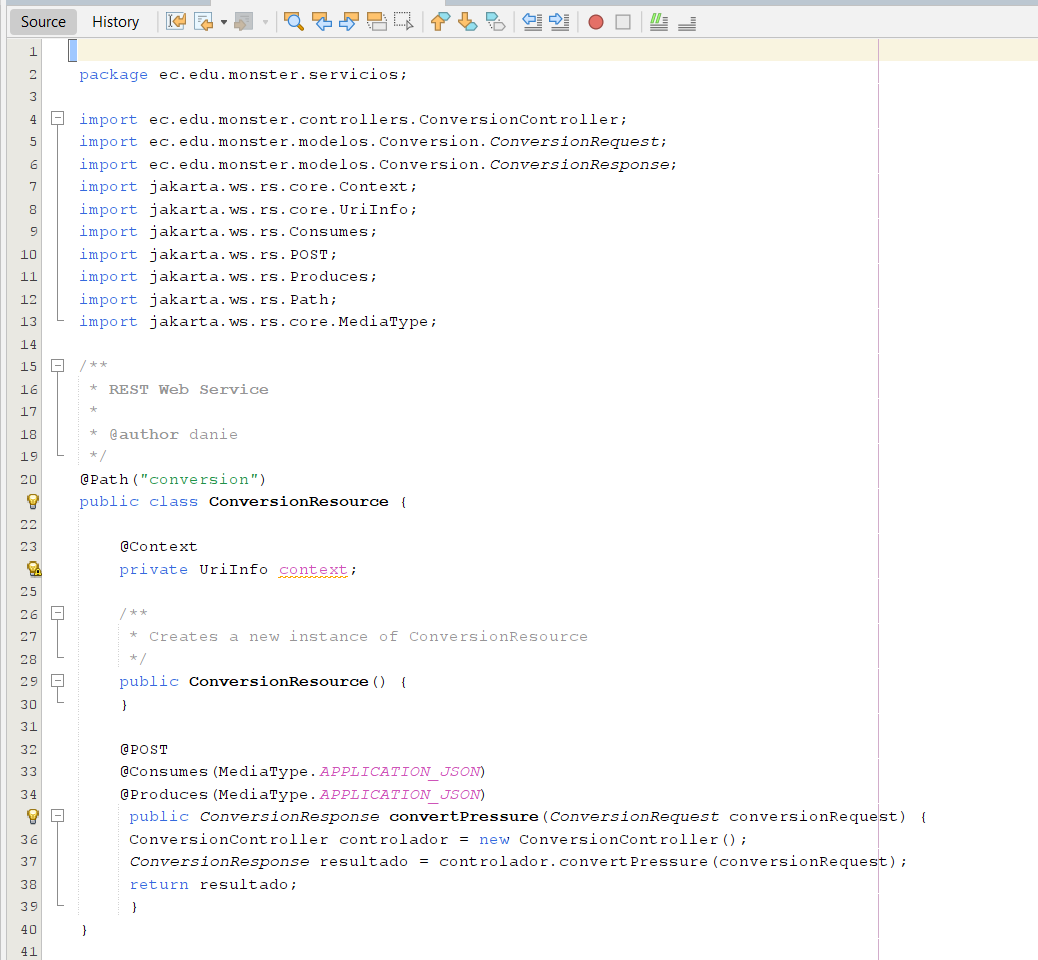


Figura 16: Implementación del servicio de conversión

Tabla 5: Código del servicio conversión

|  |
| --- |
| 1. package ec.edu.monster.servicios;  2.  3. import ec.edu.monster.controllers.ConversionController;  4. import ec.edu.monster.modelos.Conversion.ConversionRequest;  5. import ec.edu.monster.modelos.Conversion.ConversionResponse;  6. import jakarta.ws.rs.core.Context;  7. import jakarta.ws.rs.core.UriInfo;  8. import jakarta.ws.rs.Consumes;  9. import jakarta.ws.rs.POST;  10. import jakarta.ws.rs.Produces;  11. import jakarta.ws.rs.Path;  12. import jakarta.ws.rs.core.MediaType;  13.  14. /\*\*  15. \* REST Web Service  16. \*  17. \* @author danie  18. \*/  19. @Path("conversion")  20. public class ConversionResource {  21.  22. @Context  23. private UriInfo context;  24.  25. /\*\*  26. \* Creates a new instance of ConversionResource  27. \*/  28. public ConversionResource() {  29. }  30.  31. @POST  32. @Consumes(MediaType.APPLICATION\_JSON)  33. @Produces(MediaType.APPLICATION\_JSON)  34. public ConversionResponse convertPressure(ConversionRequest conversionRequest) {  35. ConversionController controlador = new ConversionController();  36. ConversionResponse resultado = controlador.convertPressure(conversionRequest);  37. return resultado;  38. }  39. } |

De la misma manera se creó el la lógica del servicio para la validación de credenciales implementando el método POST.

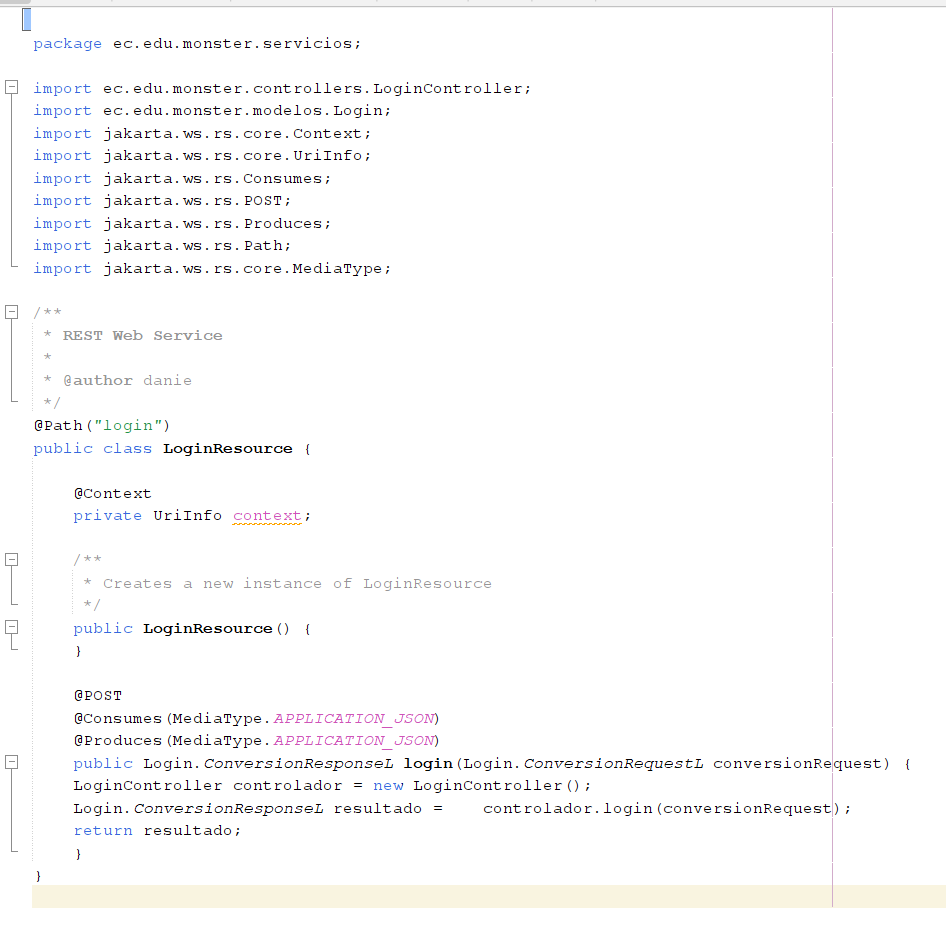


Figura 17: Implementación del servicios de login

Tabla 6: Código del servcio Login

|  |
| --- |
| 1. package ec.edu.monster.servicios;  2.  3. import ec.edu.monster.controllers.LoginController;  4. import ec.edu.monster.modelos.Login;  5. import jakarta.ws.rs.core.Context;  6. import jakarta.ws.rs.core.UriInfo;  7. import jakarta.ws.rs.Consumes;  8. import jakarta.ws.rs.POST;  9. import jakarta.ws.rs.Produces;  10. import jakarta.ws.rs.Path;  11. import jakarta.ws.rs.core.MediaType;  12.  13. /\*\*  14. \* REST Web Service  15. \*  16. \* @author danie  17. \*/  18. @Path("login")  19. public class LoginResource {  20.  21. @Context  22. private UriInfo context;  23.  24. /\*\*  25. \* Creates a new instance of LoginResource  26. \*/  27. public LoginResource() {  28. }  29.  30. @POST  31. @Consumes(MediaType.APPLICATION\_JSON)  32. @Produces(MediaType.APPLICATION\_JSON)  33. public Login.ConversionResponseL login(Login.ConversionRequestL conversionRequest) {  34. LoginController controlador = new LoginController();  35. Login.ConversionResponseL resultado = controlador.login(conversionRequest);  36. return resultado;  37. }  38. }  39. |

Finalmente, la estructura del servicio quedará de la siguiente forma

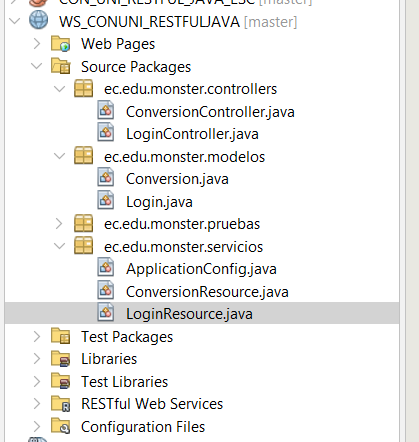


Figura 18: Estructura final del servicio

En donde se implementa el modelo, el controlador y como en este caso se no se tiene una vista se creó el paquete de servicios y pruebas.

Luego de esto probará el correcto funcionamiento del servicio.

## PRUEBA SERVIDOR

Dentro del servicio se creó una prueba para comprobar el correcto funcionamiento del servicio antes de levantarlo, para esto dentro del paquete de pruebas se creó una clase de pruebas.

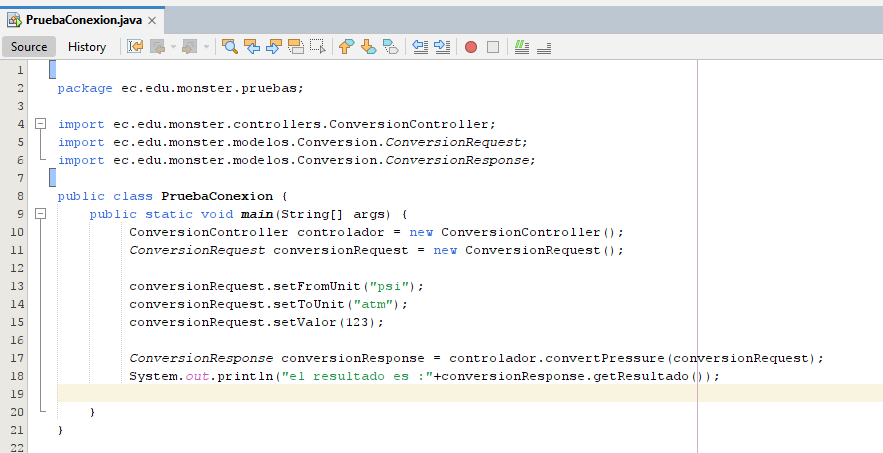


Figura 19: Implementación de una prueba de funcionamiento del servicio.

Tabla 7: Código de prueba

|  |
| --- |
| 1. package ec.edu.monster.pruebas;  2.  3. import ec.edu.monster.controllers.ConversionController;  4. import ec.edu.monster.modelos.Conversion.ConversionRequest;  5. import ec.edu.monster.modelos.Conversion.ConversionResponse;  6.  7. public class PruebaConexion {  8. public static void main(String[] args) {  9. ConversionController controlador = new ConversionController();  10. ConversionRequest conversionRequest = new ConversionRequest();  11.  12. conversionRequest.setFromUnit("psi");  13. conversionRequest.setToUnit("atm");  14. conversionRequest.setValor(123);  15.  16. ConversionResponse conversionResponse = controlador.convertPressure(conversionRequest);  17. System.out.println("el resultado es :"+conversionResponse.getResultado());  18.  19. }  20. }  21. |

### EJECUCIÓN DEL SERVICIO

Para realizar la prueba del servicio levantado dentro del servidor de payara se debe dar click derecho sobre el proyecto del servicio y se debe seleccionar la opción de Test RESTful Web Service y seguir los siguientes pasos.

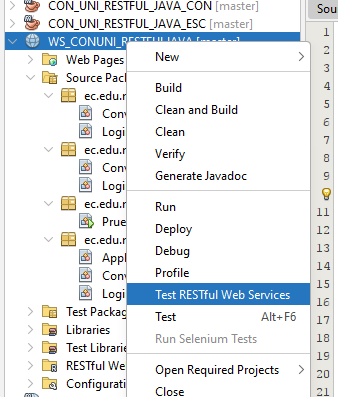


Figura 20: Opción de prueba del servicio

A continuación, se abrirá la siguiente ventana en donde se debe seleccionar la opción que viene marcada por default.



Figura 21: Configuración de prueba del servicio.

Luego de darle al botón de OK se desplegará una página web donde se mostrarán los servicios creados. Dentro de esta página se tienen las opciones de probar ambos servicios.

A screenshot of a computer

Description automatically generated

Figura 22: Página principal de pruebas del servicio.

Al darle click a cualquiera de los dos servicios mostrados en la barra lateral izquierda se mostrará una sección donde se puede probar cada servicio con los parámetros necesarios para el mismo.

A screenshot of a computer

Description automatically generated

Figura 23: Sección de prueba de un servicio en específico.

En este caso se prueba el funcionamiento del servicio login, enviando como contenido el objeto que está esperando el servicio y obteniendo más abajo el resultado.

A screenshot of a computer

Description automatically generated

Figura 24: Prueba del servicio con el objeto.

Adicionalmente se probó una petición utilizando credenciales incorrectas.



Figura 25: Prueba fallida del servicio.

## APLICACIÓN CLIENTE CONSOLA

### CREACIÓN DEL PROYECTO CLIENTE

Para la creación de este proyecto se da click al botón de nuevo proyecto.

A screenshot of a computer

Description automatically generated

Figura 26: Botón de creación de nuevo proyecto

Ahí se escoge la categoría de Java con Maven y dentro de los proyectos se selecciona un proyecto de Aplicación de Java.

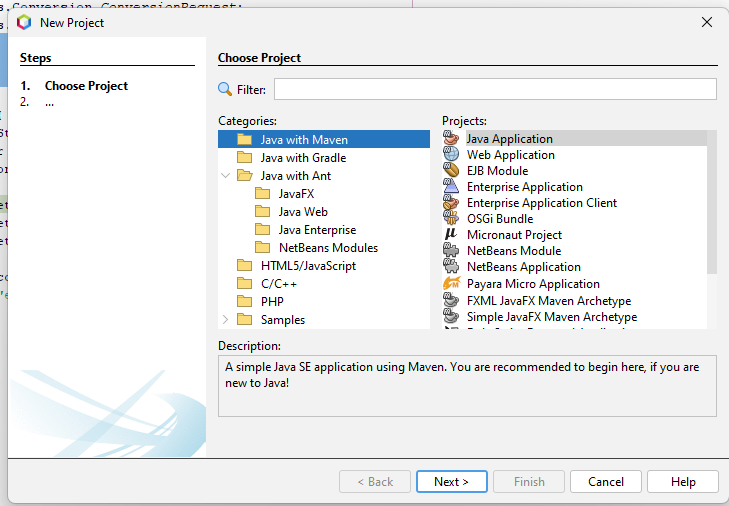


Figura 27: Selección del tipo de proyecto.

De esta manera, se crea un proyecto que contiene el cliente de consola. En este cliente, se consume el [servicio creado](#_APLICACIÓN_SERVIDOR).

A screenshot of a computer

Description automatically generated

Figura 28: Proyecto inicial para el cliente de consola.

### CREACIÓN DEL MODELO MVC

Una vez se tiene el proyecto creado se puede implementar la arquitectura MVC para el cliente de consola. Se debe dar click derecho sobre la carpeta source y añadir un nuevo paquete.

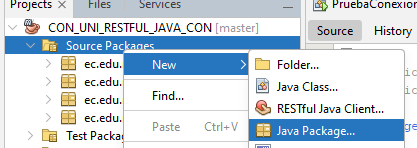


Figura 29: Creación de un nuevo paquete

Luego en la ventana emergente se debe añadir el nombre del paquete, en este caso se deben crear por lo menos los 3 paquetes de modelo, vista y controlador.



Figura 30: Creación de un paquete.

Finalmente, cuando se hayan creado los paquetes se deberá tener la siguiente estructura.

A screenshot of a computer program

Description automatically generated

Figura 31: Estructura MVC.

### IMPLEMENTACIÓN DEL SERVICIO

Dentro de los paquetes se creó un paquete específico para consumir el servicios RESTful, en este se debe dar click derecho, en nuevo y seleccionar un cliente RESTful de java.

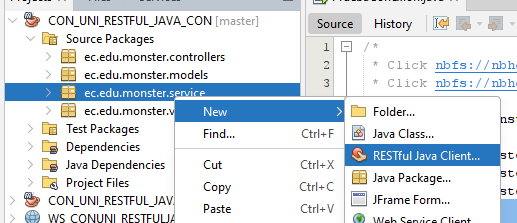


Figura 32: Implementación de un cliente RESTful

Al hacer click se desplegará una ventana emergente para la configuración y selección del servicio. Ahí se deberá asignar el nombre y realizar la búsqueda del servicio, para esto se le da click sobre el botón browse y se desplegará una nueva ventana.

A screenshot of a computer

Description automatically generated

Figura 33: Nombre y localización del cliente RESTful

Al darle click a browse se desplegará una ventana para poder seleccionar un servicio de los que estén disponibles dentro de los proyectos de netbeans, en este caso se muestran dos proyectos, sin embargo, se selecionará el que tiene el nombre WS\_CONUNI\_RESTFULJAVA.

A screenshot of a computer

Description automatically generated

Figura 34: Servicios disponibles en netbeans.

Al seleccionar el proyecto con el servicio se desplegarán los dos servicios creados previamente, se debe seleccionar uno por uno y darle a ok.

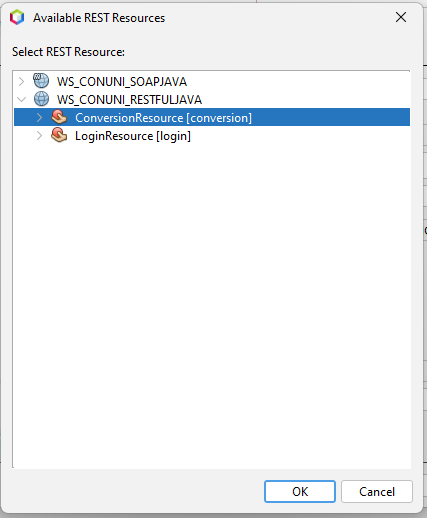


Figura 35: Selección de servicios RESTful disponibles.

Una vez seleccionado y se le da click al botón ok, esto se debe realizar para ambos servicios y se crearán 2 archivos con código generado.

A screenshot of a computer

Description automatically generated

Figura 36: Archivos del cliente RESTful de los servicios implementados.

Con los archivos creados no es necesario realizar ningún cambio ya que se pueden utilizar tal y como se genera.

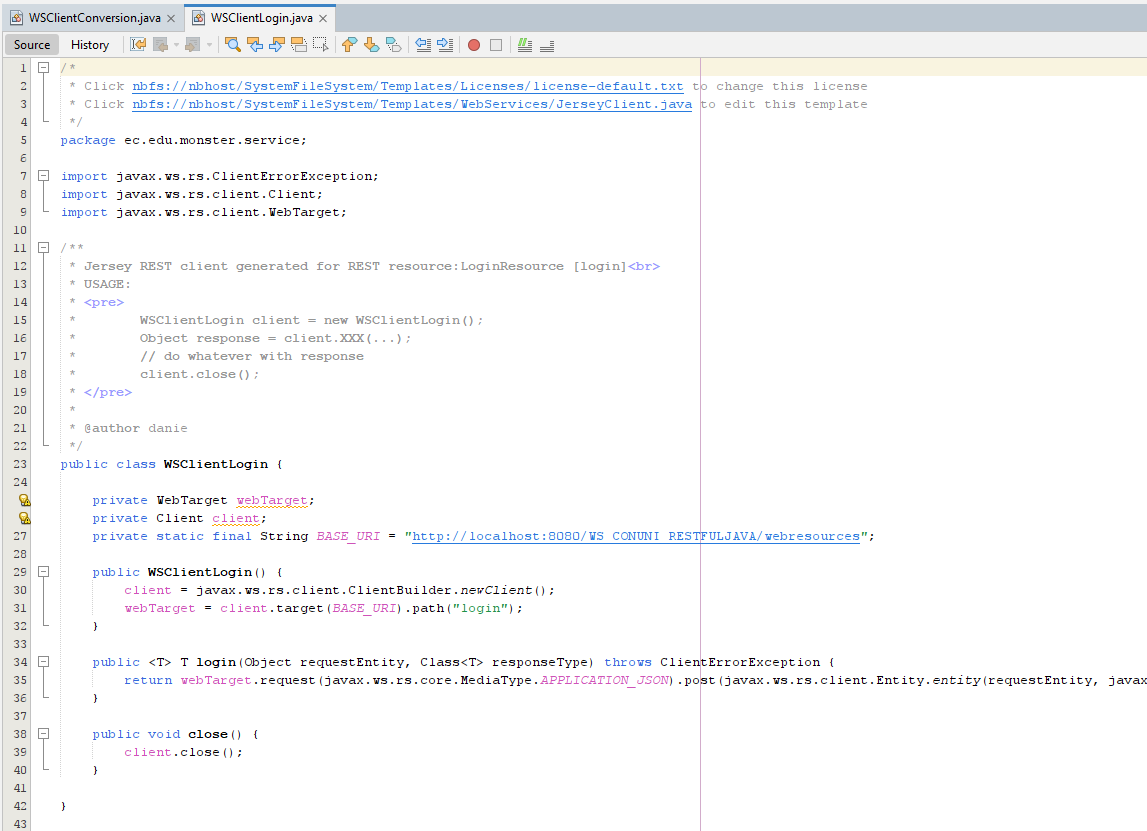


Figura 37: Archivo de cliente RESTful para el Login.

Tabla 8: Código generado por el cliente RESTful

|  |
| --- |
| 1. package ec.edu.monster.service;  2.  3. import javax.ws.rs.ClientErrorException;  4. import javax.ws.rs.client.Client;  5. import javax.ws.rs.client.WebTarget;  6.  7. /\*\*  8. \* Jersey REST client generated for REST resource:LoginResource [login]<br>  9. \* USAGE:  10. \* <pre>  11. \* WSClientLogin client = new WSClientLogin();  12. \* Object response = client.XXX(...);  13. \* // do whatever with response  14. \* client.close();  15. \* </pre>  16. \*  17. \* @author danie  18. \*/  19. public class WSClientLogin {  20.  21. private WebTarget webTarget;  22. private Client client;  23. private static final String BASE\_URI = "http://localhost:8080/WS\_CONUNI\_RESTFULJAVA/webresources";  24.  25. public WSClientLogin() {  26. client = javax.ws.rs.client.ClientBuilder.newClient();  27. webTarget = client.target(BASE\_URI).path("login");  28. }  29.  30. public <T> T login(Object requestEntity, Class<T> responseType) throws ClientErrorException {  31. return webTarget.request(javax.ws.rs.core.MediaType.APPLICATION\_JSON).post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION\_JSON), responseType);  32. }  33.  34. public void close() {  35. client.close();  36. }  37. } |

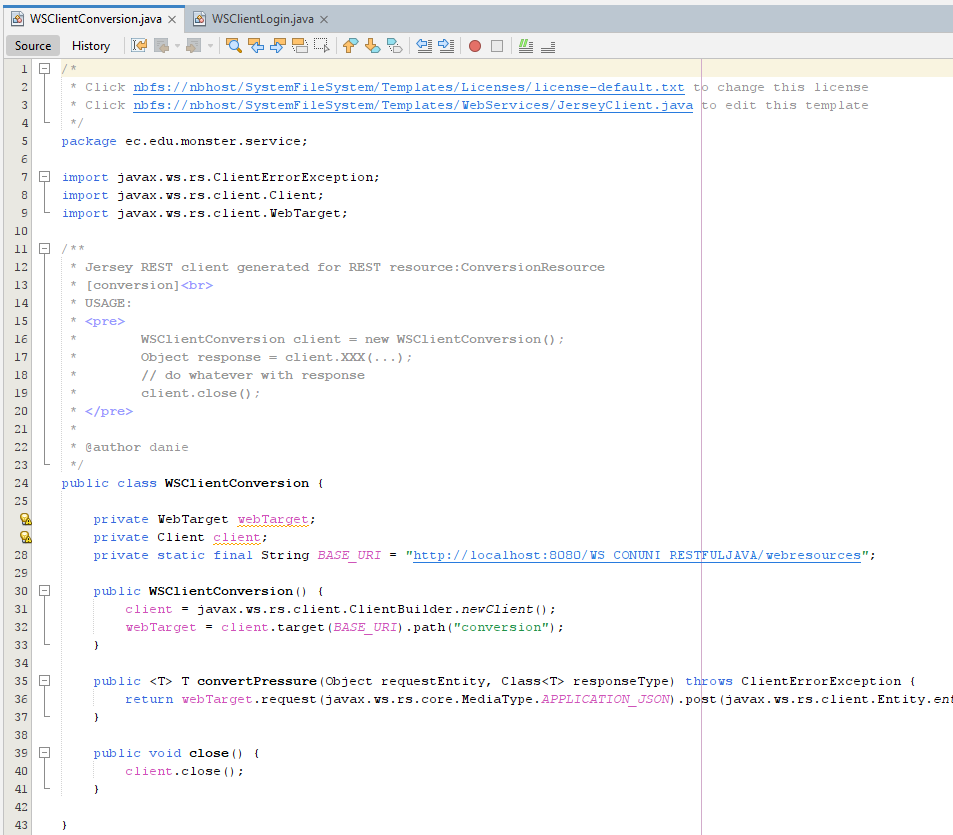


Figura 38: Archivo de cliente RESTful para la conversión.

Tabla 9: Código del cliente RESTful de conversión.

|  |
| --- |
| 1. package ec.edu.monster.service;  2.  3. import javax.ws.rs.ClientErrorException;  4. import javax.ws.rs.client.Client;  5. import javax.ws.rs.client.WebTarget;  6.  7. /\*\*  8. \* Jersey REST client generated for REST resource:ConversionResource  9. \* [conversion]<br>  10. \* USAGE:  11. \* <pre>  12. \* WSClientConversion client = new WSClientConversion();  13. \* Object response = client.XXX(...);  14. \* // do whatever with response  15. \* client.close();  16. \* </pre>  17. \*  18. \* @author danie  19. \*/  20. public class WSClientConversion {  21.  22. private WebTarget webTarget;  23. private Client client;  24. private static final String BASE\_URI = "http://localhost:8080/WS\_CONUNI\_RESTFULJAVA/webresources";  25.  26. public WSClientConversion() {  27. client = javax.ws.rs.client.ClientBuilder.newClient();  28. webTarget = client.target(BASE\_URI).path("conversion");  29. }  30.  31. public <T> T convertPressure(Object requestEntity, Class<T> responseType) throws ClientErrorException {  32. return webTarget.request(javax.ws.rs.core.MediaType.APPLICATION\_JSON).post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION\_JSON), responseType);  33. }  34.  35. public void close() {  36. client.close();  37. }  38. } |

### PAQUETE MODELO

Dentro del paquete modelo se crearon dos modelos en específico, el modelo del login y el de conversión esto debido a que el servicio consume y devuelve objetos, por tanto la información que se envia y recibe debe tener una misma estructura.

Para el primer modelo llamado ConversionModel se implementó la misma estructura del modelo creado en el servicio en el cual se crearon un pedido y una respuesta. En el pedido se envían tres atributos y en la respuesta se reciben 4.

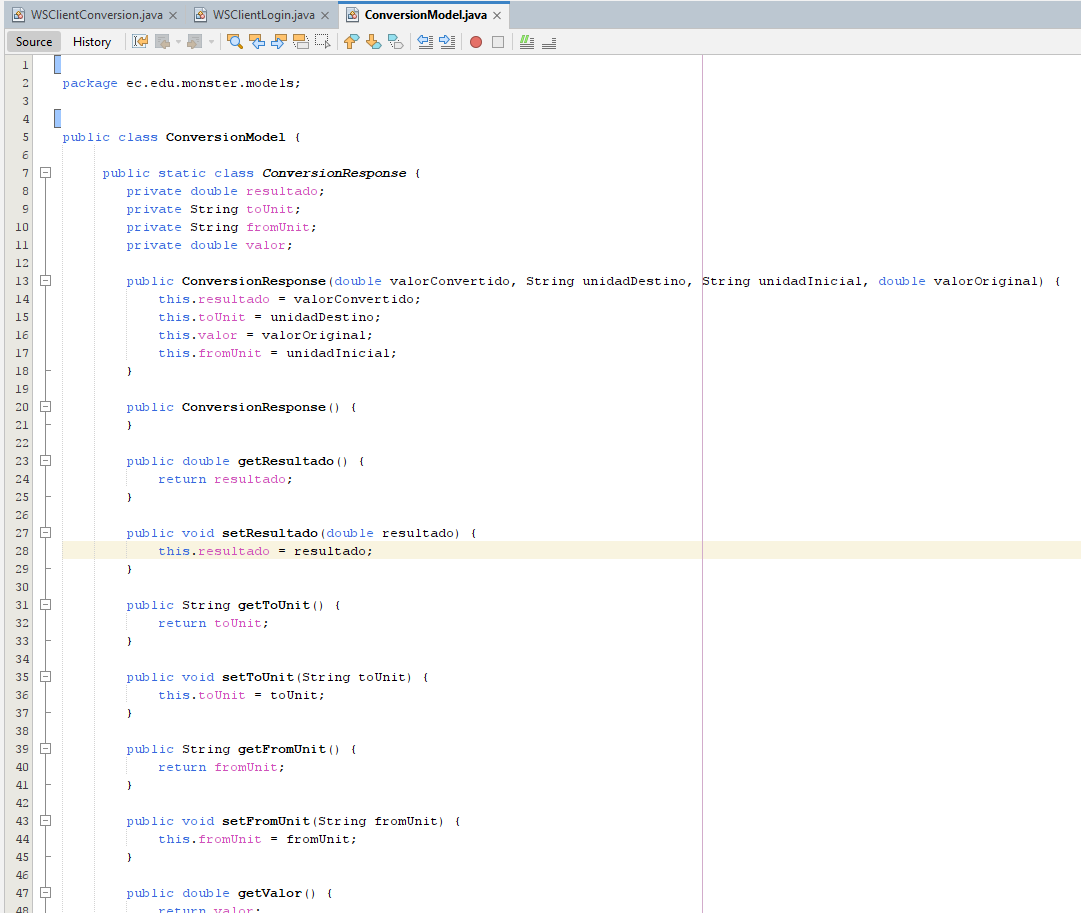


Figura 39: Modelo de los objetos de conversión.

Tabla 10: Código del modelo de objetos de la conversión.

|  |
| --- |
| 1. package ec.edu.monster.models;  2.  3. /\*\*  4. \*  5. \* @author danie  6. \*/  7. public class ConversionModel {  8.  9. public static class ConversionResponse {  10. private double resultado;  11. private String toUnit;  12. private String fromUnit;  13. private double valor;  14.  15. public ConversionResponse(double valorConvertido, String unidadDestino, String unidadInicial, double valorOriginal) {  16. this.resultado = valorConvertido;  17. this.toUnit = unidadDestino;  18. this.valor = valorOriginal;  19. this.fromUnit = unidadInicial;  20. }  21.  22. public ConversionResponse() {  23. }  24.  25. public double getResultado() {  26. return resultado;  27. }  28.  29. public void setResultado(double resultado) {  30. this.resultado = resultado;  31. }  32.  33. public String getToUnit() {  34. return toUnit;  35. }  36.  37. public void setToUnit(String toUnit) {  38. this.toUnit = toUnit;  39. }  40.  41. public String getFromUnit() {  42. return fromUnit;  43. }  44.  45. public void setFromUnit(String fromUnit) {  46. this.fromUnit = fromUnit;  47. }  48.  49. public double getValor() {  50. return valor;  51. }  52.  53. public void setValor(double valor) {  54. this.valor = valor;  55. }  56. }  57. public static class ConversionRequest {  58. private double valor;  59. private String fromUnit;  60. private String toUnit;  61.  62. public ConversionRequest() {  63. }  64.  65.  66. public ConversionRequest(double valor, String fromUnit, String toUnit) {  67. this.valor = valor;  68. this.fromUnit = fromUnit;  69. this.toUnit = toUnit;  70. }  71.  72.  73. public double getValor() {  74. return valor;  75. }  76.  77. public void setValor(double valor) {  78. this.valor = valor;  79. }  80.  81. public String getFromUnit() {  82. return fromUnit;  83. }  84.  85. public void setFromUnit(String fromUnit) {  86. this.fromUnit = fromUnit;  87. }  88.  89. public String getToUnit() {  90. return toUnit;  91. }  92.  93. public void setToUnit(String toUnit) {  94. this.toUnit = toUnit;  95. }  96. }  97. }  98. |

Debido a que se tienen dos servicios que tienen la misma naturaleza de enviar y recibir objetos se creó igual de igual manera un modelo para el Login, en donde se especificaron los pedidos y las respuestas.

A screenshot of a computer

Description automatically generated

Figura 40: Modelo de objetos del Login

Tabla 11: Código del modelo de objetos del Login

|  |
| --- |
| 1. package ec.edu.monster.models;  2.  3. /\*\*  4. \*  5. \* @author danie  6. \*/  7. public class LoginModel {  8. public static class ConversionRequestL {  9.  10. private String username;  11. private String password;  12.  13. public ConversionRequestL() {  14. }  15.  16. public ConversionRequestL(String username, String password) {  17. this.username = username;  18. this.password = password;  19. }  20.  21.  22. public String getUsername() {  23. return username;  24. }  25.  26. public void setUsername(String username) {  27. this.username = username;  28. }  29.  30. public String getPassword() {  31. return password;  32. }  33.  34. public void setPassword(String password) {  35. this.password = password;  36. }  37.  38. }  39.  40. public static class ConversionResponseL{  41. private String username;  42. private String password;  43.  44. public ConversionResponseL() {  45. }  46.  47. public ConversionResponseL(String username, String password){  48. this.username = username;  49. this.password = password;  50. }  51.  52. public String getUsername() {  53. return username;  54. }  55.  56. public void setUsername(String username) {  57. this.username = username;  58. }  59.  60. public String getPassword() {  61. return password;  62. }  63.  64. public void setPassword(String password) {  65. this.password = password;  66. }  67. }  68. }  69. |

Finalmente, la estructura del paquete de modelo debe quedar de la siguiente forma.

A screenshot of a computer

Description automatically generated

Figura 41: Estructura final del paquete modelos.

### PAQUETE CONTROLADOR

Para realizar la implementación tanto de los servicios y modelos, así como la lógica del programa se crearon dos controladores, el primero para la conversión y el segundo para la autenticación de los usuarios.

Dentro del controlador de conversiones se implementaron tanto el modelo como el servicio de conversión. Aquí se implementó la lógica de la conversión, selección de unidades, ingreso de valores y validaciones necesarias para el correcto funcionamiento del cliente.

A screenshot of a computer

Description automatically generated

Figura 42: Controlador de conversiones.

Tabla 12: Código del controlador de conversiones.

|  |
| --- |
| 1. package ec.edu.monster.controllers;  2.  3. import ec.edu.monster.models.ConversionModel.ConversionRequest;  4. import ec.edu.monster.models.ConversionModel.ConversionResponse;  5. import ec.edu.monster.service.WSClientConversion;  6.  7. /\*\*  8. \*  9. \* @author danie  10. \*/  11. public class ConversionController {  12.  13. public ConversionResponse convertPressure(double value, String fromUnit, String toUnit) throws Exception {  14. WSClientConversion client = new WSClientConversion();  15. try {  16. // Prepare the request  17. ConversionRequest request = new ConversionRequest();  18. request.setValor(value);  19. request.setFromUnit(fromUnit);  20. request.setToUnit(toUnit);  21.  22.  23. // Make the request to the web service  24. ConversionResponse response = client.convertPressure(request, ConversionResponse.class);  25.  26. // Validate response  27. if (response == null) {  28. throw new Exception("No se recibió respuesta del servicio de conversión");  29. }  30.  31. return response;  32.  33. } catch (javax.ws.rs.ClientErrorException e) {  34. System.err.println("Error en la conversión: " + e.getMessage());  35. throw new Exception("Error al realizar la conversión: " + e.getMessage());  36. } finally {  37. client.close();  38. }  39. }  40.  41. // Método de utilidad para validar unidades de presión  42. public boolean isValidPressureUnit(String unit) {  43. String[] validUnits = {"pa", "bar", "psi", "atm", "mmhg"};  44. for (String validUnit : validUnits) {  45. if (validUnit.equalsIgnoreCase(unit)) {  46. return true;  47. }  48. }  49. return false;  50. }  51.  52. // Método para facilitar el uso con manejo de errores incluido  53. public double convertPressureValue(double value, String fromUnit, String toUnit) throws Exception {  54. // Validar unidades  55. if (!isValidPressureUnit(fromUnit) || !isValidPressureUnit(toUnit)) {  56. throw new Exception("Unidades de presión no válidas");  57. }  58.  59. // Realizar conversión  60. ConversionResponse response = convertPressure(value, fromUnit, toUnit);  61. return response.getResultado();  62. }  63. }  64. |

De igual manera se creó un controlador específico para la sección de autenticación, aquí se implementó el uso de los modelos, el servicio de Login, validaciones y el hash de la contraseña.

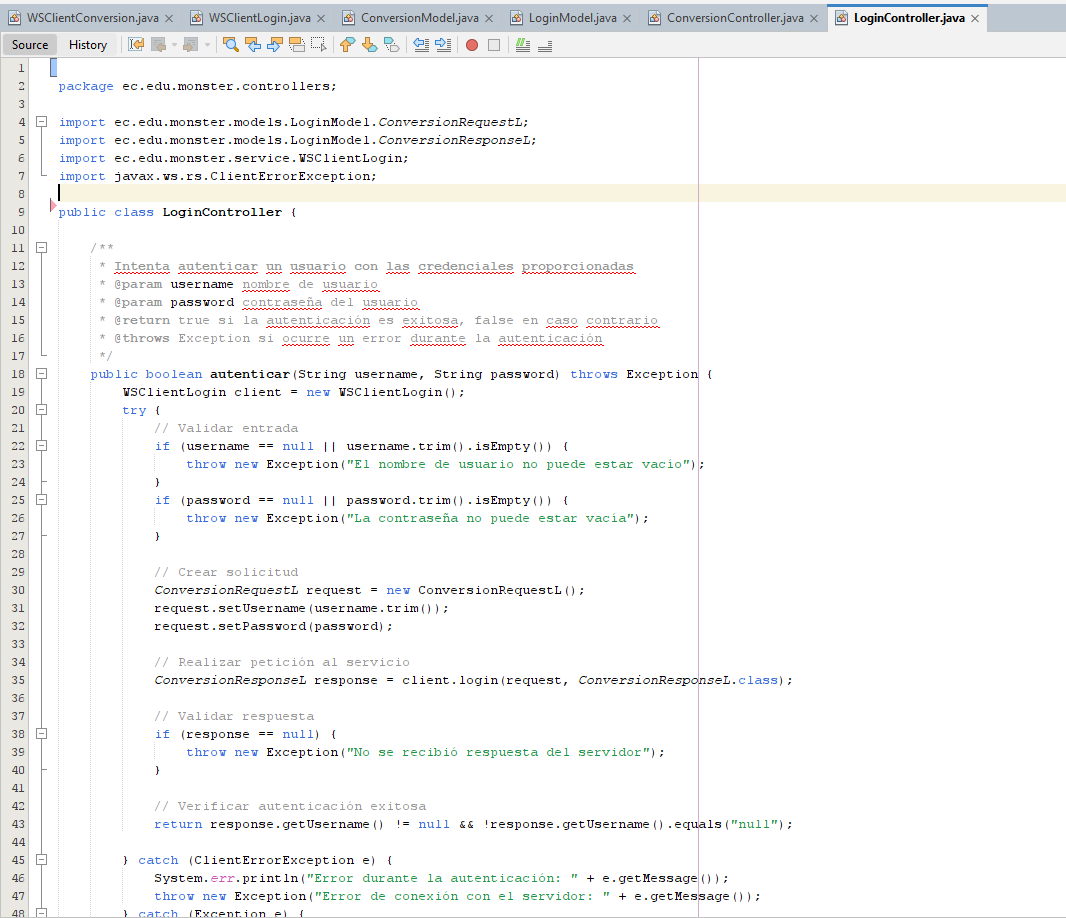


Figura 43: Controlador de Login

Tabla 13: Código de controlador del Login.

|  |
| --- |
| 1. package ec.edu.monster.controllers;  2.  3. import ec.edu.monster.models.LoginModel.ConversionRequestL;  4. import ec.edu.monster.models.LoginModel.ConversionResponseL;  5. import ec.edu.monster.service.WSClientLogin;  6. import javax.ws.rs.ClientErrorException;  7.  8.  9. /\*\*  10. \*  11. \* @author danie  12. \*/  13. public class LoginController {  14.  15. /\*\*  16. \* Intenta autenticar un usuario con las credenciales proporcionadas  17. \* @param username nombre de usuario  18. \* @param password contraseña del usuario  19. \* @return true si la autenticación es exitosa, false en caso contrario  20. \* @throws Exception si ocurre un error durante la autenticación  21. \*/  22. public boolean autenticar(String username, String password) throws Exception {  23. WSClientLogin client = new WSClientLogin();  24. try {  25. // Validar entrada  26. if (username == null || username.trim().isEmpty()) {  27. throw new Exception("El nombre de usuario no puede estar vacío");  28. }  29. if (password == null || password.trim().isEmpty()) {  30. throw new Exception("La contraseña no puede estar vacía");  31. }  32.  33. // Crear solicitud  34. ConversionRequestL request = new ConversionRequestL();  35. request.setUsername(username.trim());  36. request.setPassword(password);  37.  38. // Realizar petición al servicio  39. ConversionResponseL response = client.login(request, ConversionResponseL.class);  40.  41. // Validar respuesta  42. if (response == null) {  43. throw new Exception("No se recibió respuesta del servidor");  44. }  45.  46. // Verificar autenticación exitosa  47. return response.getUsername() != null && !response.getUsername().equals("null");  48.  49. } catch (ClientErrorException e) {  50. System.err.println("Error durante la autenticación: " + e.getMessage());  51. throw new Exception("Error de conexión con el servidor: " + e.getMessage());  52. } catch (Exception e) {  53. System.err.println("Error inesperado: " + e.getMessage());  54. throw e;  55. } finally {  56. client.close();  57. }  58. }  59.  60. /\*\*  61. \* Método simplificado para verificar credenciales con manejo de errores incluido  62. \* @param username nombre de usuario  63. \* @param password contraseña  64. \* @return mensaje de estado de la autenticación  65. \*/  66. public String verificarCredenciales(String username, String password) {  67. try {  68. boolean autenticado = autenticar(username, password);  69. if (autenticado) {  70. return "Autenticación exitosa";  71. } else {  72. return "Credenciales inválidas";  73. }  74. } catch (Exception e) {  75. return "Error: " + e.getMessage();  76. }  77. }  78.  79. /\*\*  80. \* Verifica si un nombre de usuario tiene un formato válido  81. \* @param username nombre de usuario a validar  82. \* @return true si el formato es válido, false en caso contrario  83. \*/  84. public boolean validarFormatoUsuario(String username) {  85. if (username == null || username.trim().isEmpty()) {  86. return false;  87. }  88. // Puedes agregar más reglas de validación según tus necesidades  89. return username.matches("^[a-zA-Z0-9\_]{3,20}$");  90. }  91. }  92. |

### PAQUETE VISTA

Para finalizar se implementó una clase dentro del paquete la cual mostrará toda la información por consola, aquí se establecieron los menús y el uso de las funciones de los controladores.

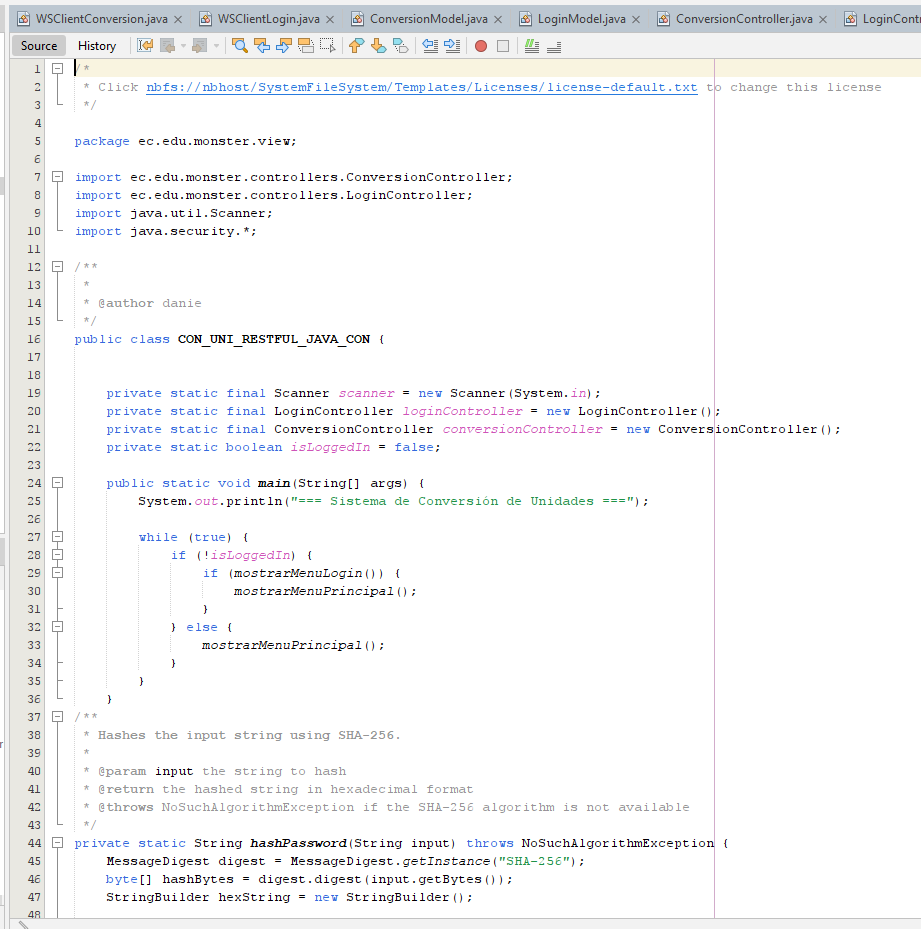


Figura 44: Vista de cliente de consola

Tabla 14: Código de la vista del cliente.

|  |
| --- |
| 1. package ec.edu.monster.view;  2.  3. import ec.edu.monster.controllers.ConversionController;  4. import ec.edu.monster.controllers.LoginController;  5. import java.util.Scanner;  6. import java.security.\*;  7.  8. /\*\*  9. \*  10. \* @author danie  11. \*/  12. public class CON\_UNI\_RESTFUL\_JAVA\_CON {  13.  14.  15. private static final Scanner scanner = new Scanner(System.in);  16. private static final LoginController loginController = new LoginController();  17. private static final ConversionController conversionController = new ConversionController();  18. private static boolean isLoggedIn = false;  19.  20. public static void main(String[] args) {  21. System.out.println("=== Sistema de Conversión de Unidades ===");  22.  23. while (true) {  24. if (!isLoggedIn) {  25. if (mostrarMenuLogin()) {  26. mostrarMenuPrincipal();  27. }  28. } else {  29. mostrarMenuPrincipal();  30. }  31. }  32. }  33. /\*\*  34. \* Hashes the input string using SHA-256.  35. \*  36. \* @param input the string to hash  37. \* @return the hashed string in hexadecimal format  38. \* @throws NoSuchAlgorithmException if the SHA-256 algorithm is not available  39. \*/  40. private static String hashPassword(String input) throws NoSuchAlgorithmException {  41. MessageDigest digest = MessageDigest.getInstance("SHA-256");  42. byte[] hashBytes = digest.digest(input.getBytes());  43. StringBuilder hexString = new StringBuilder();  44.  45. for (byte b : hashBytes) {  46. String hex = Integer.toHexString(0xff & b);  47. if (hex.length() == 1) {  48. hexString.append('0');  49. }  50. hexString.append(hex);  51. }  52.  53. return hexString.toString();  54. }  55.  56.  57. private static boolean mostrarMenuLogin() {  58. System.out.println("\n=== Login ===");  59. System.out.print("Usuario: ");  60. String username = scanner.nextLine();  61. System.out.print("Contraseña: ");  62. String password = scanner.nextLine();  63.  64. try {  65. String hashedPassword = hashPassword(password);  66. isLoggedIn = loginController.autenticar(username, hashedPassword);  67. if (isLoggedIn) {  68. System.out.println("¡Login exitoso!");  69. return true;  70. } else {  71. System.out.println("Credenciales incorrectas. Por favor, intente nuevamente.");  72. return false;  73. }  74. } catch (Exception e) {  75. System.out.println("Error durante el login: " + e.getMessage());  76. return false;  77. }  78. }  79.  80. private static void mostrarMenuPrincipal() {  81. while (isLoggedIn) {  82. try {  83. System.out.println("\n=== Menú Principal ===");  84. System.out.println("1. Realizar conversión de presión");  85. System.out.println("2. Cerrar sesión");  86. System.out.println("3. Salir");  87. System.out.print("Seleccione una opción: ");  88.  89. String opcion = scanner.nextLine();  90.  91. switch (opcion) {  92. case "1":  93. realizarConversion();  94. break;  95. case "2":  96. isLoggedIn = false;  97. System.out.println("Sesión cerrada exitosamente.");  98. return;  99. case "3":  100. System.out.println("¡Gracias por usar el sistema!");  101. System.exit(0);  102. break;  103. default:  104. System.out.println("Opción no válida. Por favor, intente nuevamente.");  105. }  106. } catch (Exception e) {  107. System.out.println("Error: " + e.getMessage());  108. }  109. }  110. }  111.  112. private static void realizarConversion() {  113. try {  114. System.out.println("\n=== Conversión de Presión ===");  115. System.out.println("Seleccione las unidades disponibles:");  116. System.out.println("1. Pascal (pa)");  117. System.out.println("2. Bar (bar)");  118. System.out.println("3. Psi (psi)");  119. System.out.println("4. Atmósfera (atm)");  120. System.out.println("5. Milimetros de mercurio (mmhg)");  121.  122. // Ingresar el valor a convertir  123. System.out.print("Ingrese el valor a convertir: ");  124. double valor = Double.parseDouble(scanner.nextLine());  125.  126. // Seleccionar la unidad de origen  127. System.out.print("Seleccione la unidad de origen (1-5): ");  128. int opcionOrigen = Integer.parseInt(scanner.nextLine());  129. String unidadOrigen = obtenerUnidad(opcionOrigen);  130. if (unidadOrigen == null) {  131. System.out.println("Opción no válida para la unidad de origen.");  132. return;  133. }  134.  135. // Seleccionar la unidad de destino  136. System.out.print("Seleccione la unidad de destino (1-5): ");  137. int opcionDestino = Integer.parseInt(scanner.nextLine());  138. String unidadDestino = obtenerUnidad(opcionDestino);  139. if (unidadDestino == null) {  140. System.out.println("Opción no válida para la unidad de destino.");  141. return;  142. }  143.  144. // Validar y realizar la conversión  145. if (!conversionController.isValidPressureUnit(unidadOrigen) ||  146. !conversionController.isValidPressureUnit(unidadDestino)) {  147. System.out.println("Error: Unidades no válidas");  148. return;  149. }  150.  151. double resultado = conversionController.convertPressureValue(valor, unidadOrigen, unidadDestino);  152. System.out.printf("Resultado: %.4f %s = %.4f %s%n",  153. valor, unidadOrigen, resultado, unidadDestino);  154.  155. } catch (NumberFormatException e) {  156. System.out.println("Error: Por favor ingrese un valor numérico válido.");  157. } catch (Exception e) {  158. System.out.println("Error durante la conversión: " + e.getMessage());  159. }  160. }  161.  162. /\*\*  163. \* Obtiene la unidad correspondiente a la opción seleccionada.  164. \* @param opcion número de la opción seleccionada  165. \* @return el nombre de la unidad o null si la opción no es válida  166. \*/  167. private static String obtenerUnidad(int opcion) {  168. switch (opcion) {  169. case 1: return "pa";  170. case 2: return "bar";  171. case 3: return "psi";  172. case 4: return "atm";  173. case 5: return "mmhg";  174. default: return null;  175. }  176. }  177. }  178. |

### EJECUCIÓN

Para finalizar, con la tecla F5 dentro del código de la vista se ejecutó el programa y se probaron los menús, así como la funcionalidad del servicio dentro del cliente de consola.

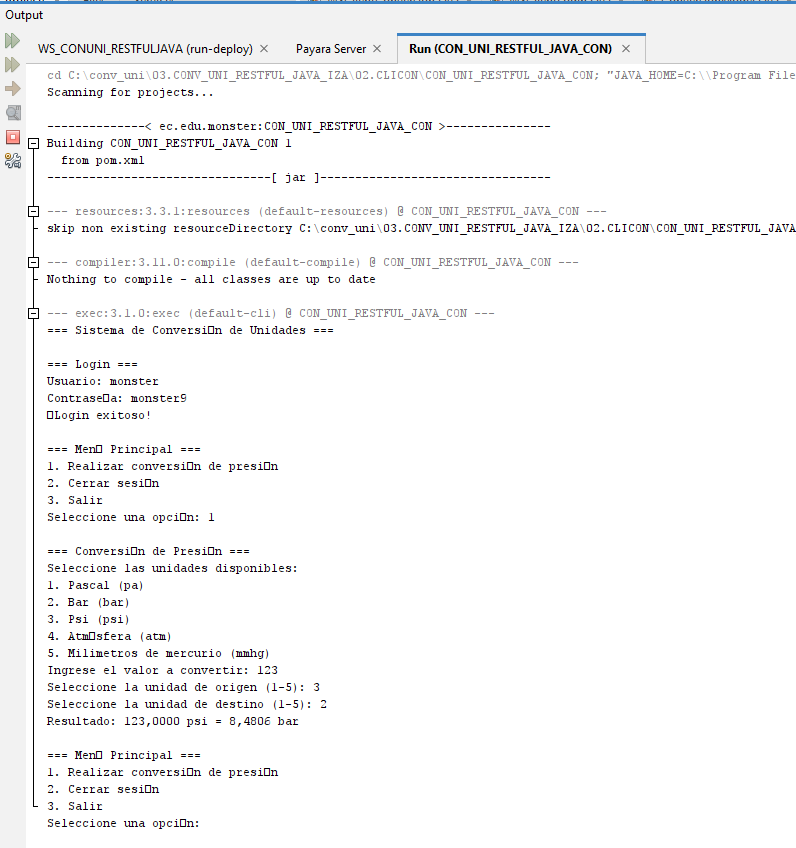


Figura 45: Ejecución del cliente de consola.

## APLICACIÓN CLIENTE ESCRITORIO

### CREACIÓN DEL PROYECTO CLIENTE

Para desarrollar el cliente de escritorio, es necesario iniciar un proyecto del tipo **Java Application** utilizando **Maven** como herramienta de gestión de dependencias. Este entorno facilita la creación de vistas gráficas para aplicaciones de escritorio

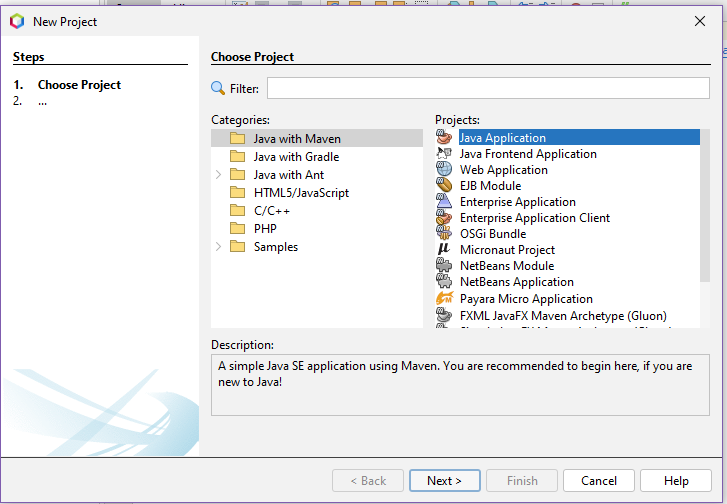


Figura 46: Selección de proyecto

Como siguiente paso, NetBeans solicita la configuración inicial del proyecto. En esta etapa, se debe ingresar el nombre del proyecto y definir los paquetes donde se organizarán los archivos. Para este caso específico, se utiliza el paquete **ec.edu.monster**

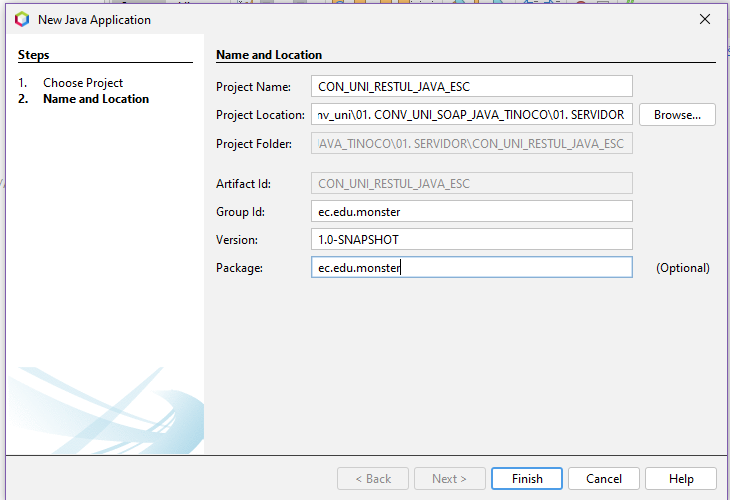


Figura 47: Configuración del proyecto

### CREACIÓN DEL MODELO MVC

Para organizar el proyecto siguiendo la arquitectura MVC, se deben crear nuevos paquetes. Este proceso se realiza haciendo clic derecho sobre el paquete raíz, seleccionando Nuevo y luego Java Package

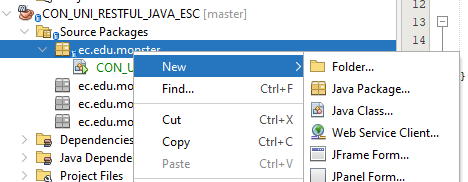


Figura 48: Creación de nuevo paquete

Al crear un nuevo paquete, se debe ingresar la ruta correspondiente al paquete raíz y, al final de esta, agregar el nombre del nuevo paquete. Este paso garantiza que los nuevos paquetes queden organizados dentro de la estructura principal del proyecto.

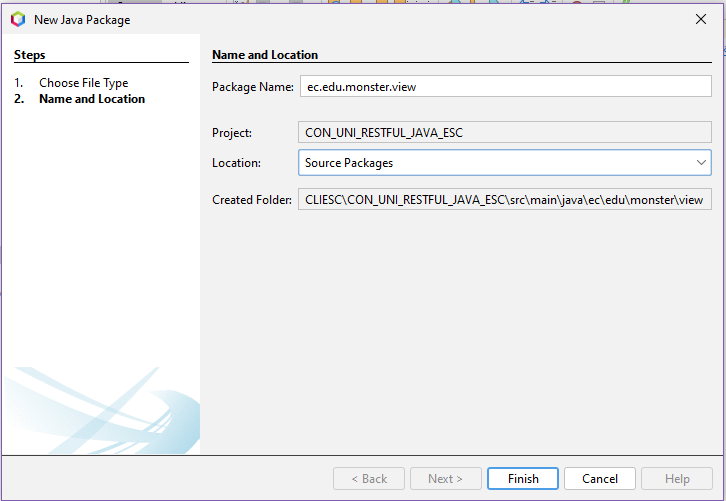


Figura 49: Nombrar paquetes

Al finalizar la creación de paquetes se tiene un proyecto estructurado de la siguiente manera

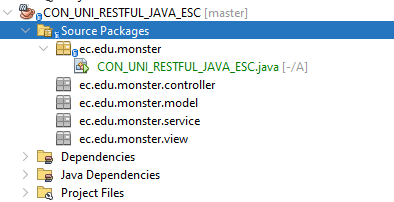


Figura 50: Estructura final del proyecto

### IMPLEMENTACIÓN DEL SERVICIO

Al igual que en el [cliente de consola](#_IMPLEMENTACIÓN_DEL_SERVICIO), en el cliente de escritorio es necesario crear clientes RESTful para consumir el servicio. Para lograr esto, dentro del paquete services, se debe hacer clic derecho, seleccionar la opción Nuevo y luego elegir RESTful Java Client.

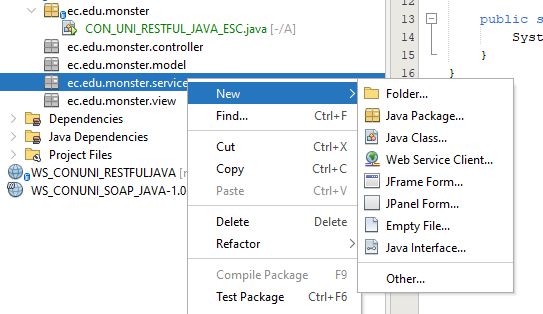


Figura 51: Crear nuevo archivo en paquete

En caso de que esta opción no esté visible en el menú principal, se debe hacer clic en Otro para desplegar más opciones de tipos de archivo. Dentro de la categoría Web Services, se encuentra la opción para crear un RESTful Java Client.

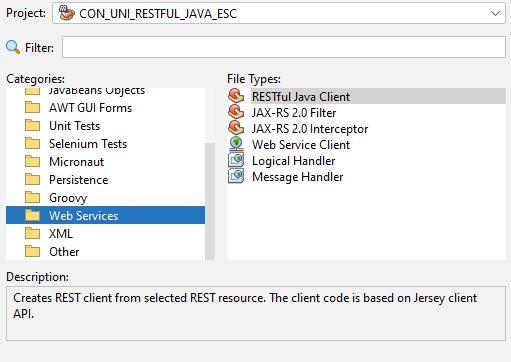


Figura 52: Selección de RESTful Java Client

A continuación, se desplegará una ventana donde se debe seleccionar el servicio RESTful.

### PAQUETE MODELO

Dentro del paquete modelo, al igual que en el [cliente de consola](#_PAQUETE_MODELO), se crean dos modelos en específico, el modelo del login y el de conversión esto debido a que el servicio consume y devuelve objetos, por tanto, la información que se envía y recibe debe tener una misma estructura.

El modelo Login contiene dos clases internas:

* ConversionRequestL: Representa los datos necesarios para realizar una solicitud de autenticación.
* ConversionResponseL: Estructura la respuesta del sistema después de procesar la autenticación.



Figura 53: LoginModel

El modelo conversión se encarga de manejar los datos relacionados con la conversión de unidades. También incluye dos clases internas:

* ConversionRequest: Define los atributos necesarios para realizar una solicitud de conversión.
* ConversionResponse: Almacena los resultados de una conversión.

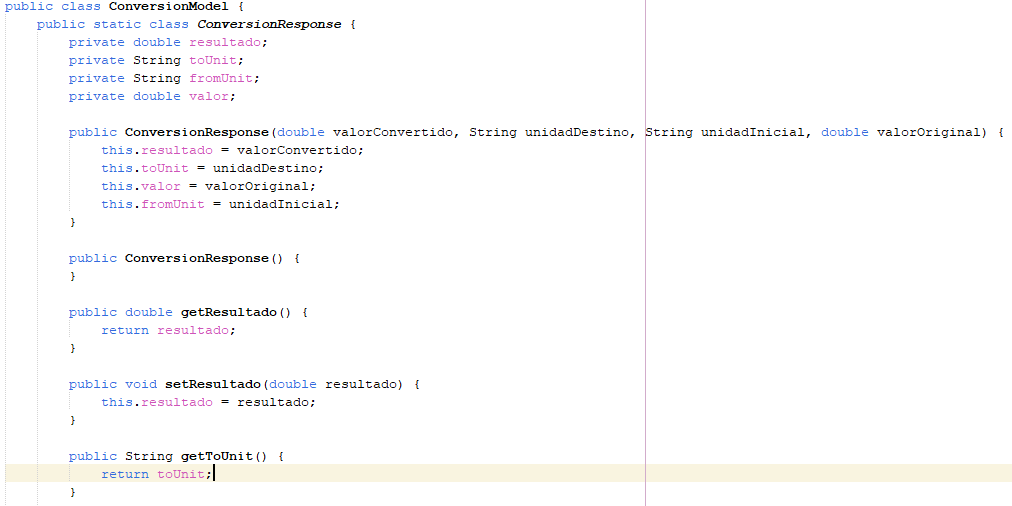


Figura 54: ConversionModel

El código de los modelos es idéntico al utilizado en el cliente de consola, que se describe detalladamente en las [10 y 11](#_PAQUETE_MODELO) de este mismo documento.

### PAQUETE CONTROLADOR

Los controladores gestionan la lógica de la interacción entre la vista y el modelo. En este caso, se han creado dos controladores fundamentales: uno para manejar la autenticación del usuario (LoginController) y otro para procesar las conversiones de unidades (ConversionController).

#### LoginController

Este controlador se encarga de gestionar la autenticación del usuario en la vista de login, para esta lógica se usan los siguientes métodos:

* Autenticar(): Realiza la autenticación del usuario. Valida las credenciales ingresadas, envía una solicitud al servicio web y evalúa la respuesta
* verificarCredenciales(): Este método obtiene las credenciales de la vista de login, las pasa al método autenticar() y, si la autenticación es correcta, cierra la vista de login y abre la vista de conversión. En caso de error, muestra un mensaje de invalidación de las credenciales.

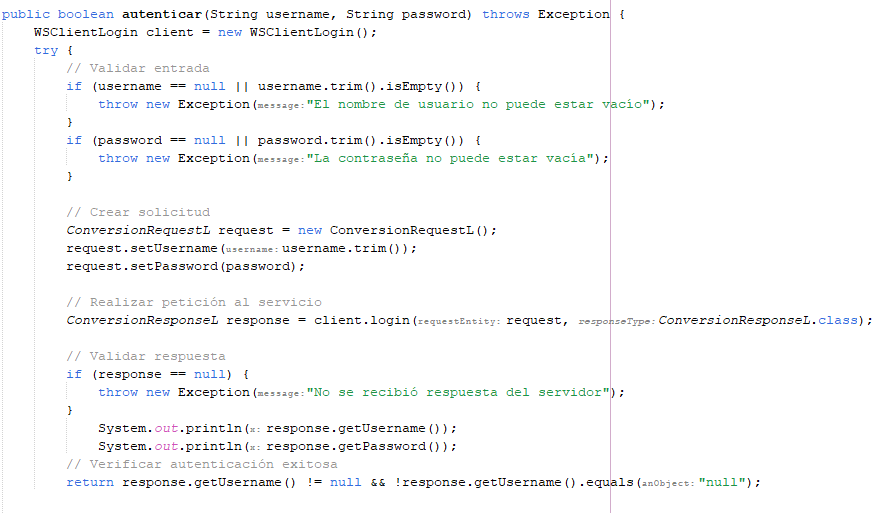


Figura 55: LoginController

#### ConversionController

Este controlador se ocupa de gestionar las conversiones de unidades de presión. Se encarga de preparar la solicitud, realizar la conversión a través del servicio web y devolver el resultado.

* convertPressure(): Este método recibe los parámetros de la conversión (valor, unidad de origen y unidad de destino), prepara la solicitud, realiza la llamada al servicio de conversión y devuelve el resultado.
* isValidPressureUnit(): Valida que las unidades de presión ingresadas sean correctas, comprobando si pertenecen a un conjunto de unidades predefinidas.
* convertPressureValue(): Este es un método de utilidad que valida las unidades antes de realizar la conversión y devuelve el valor convertido.

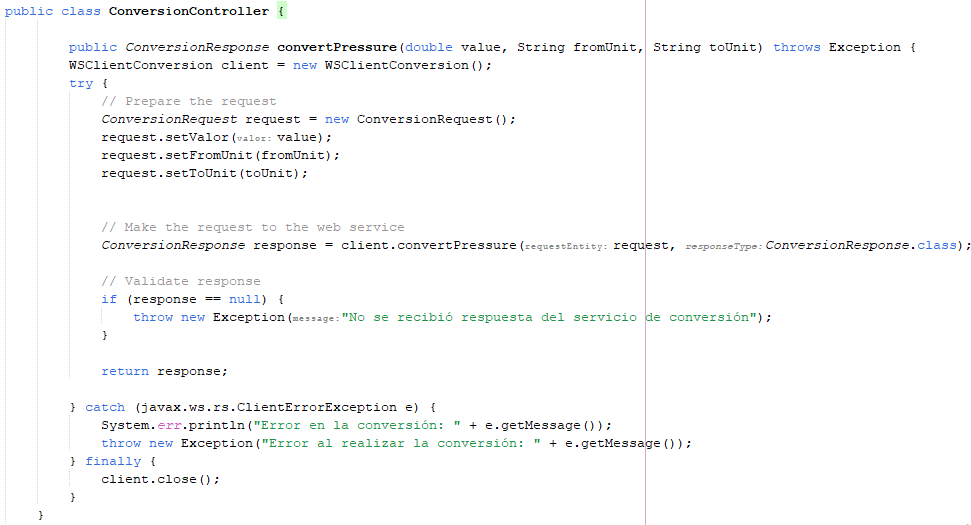


Figura 56: ConversionController

El código de los controladores es idéntico al utilizado en el cliente de consola, que se describe detalladamente en las [13 y 14](#_PAQUETE_CONTROLADOR) de este mismo documento.

### PAQUETE VISTA

Para crear las vistas utilizar JFrame Form en NetBeans, para crearlos se debe acceder al paquete view del proyecto, hacer clic derecho y seleccionar Nuevo > JFrame Form. Luego, en la ventana emergente, se debe ingresar el nombre del archivo para la vista

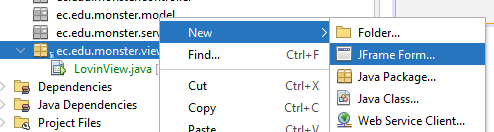


Figura 57: Crear Jframe Form

Una vez creado el archivo, se abrirá el diseñador visual de JFrame donde se pueden agregar componentes gráficos como botones, etiquetas y campos de texto. Finalmente se obtiene una vista de login como la siguiente



Figura 58: LoginView

En la vista de Login, al hacer clic en el botón de Iniciar Sesión, se captura el nombre de usuario y se procesa la contraseña mediante el algoritmo SHA-256 para generar su hash. Este hash se utiliza para autenticar las credenciales a través del controlador. Si la autenticación es exitosa, el usuario es redirigido; de lo contrario, se muestra un mensaje de error.

Tabla 15: Código LoginView

|  |
| --- |
| 1. private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {  2. String username = lblUsername.getText();  3. LoginController controlador = new LoginController();  4. try {  5. boolean autenticar = controlador.autenticar(username, this.hashPassword());  6. } catch (Exception ex) {  7.  8. public String hashPassword() throws NoSuchAlgorithmException {  9. char[] passwordChars = lblPassword.getPassword();  10. String input = new String(passwordChars);  11. MessageDigest digest = MessageDigest.getInstance("SHA-256");  12. byte[] hashBytes = digest.digest(input.getBytes());  13. StringBuilder hexString = new StringBuilder();  14. for (byte b : hashBytes) {  15. String hex = Integer.toHexString(0xff & b);  16. if (hex.length() == 1) {  17. hexString.append('0');  18. }  19. hexString.append(hex);  20. }  21.  22. return hexString.toString();  23. }  24. Logger.getLogger(LoginView.class.getName()).log(Level.SEVERE, null, ex);  25. }  26. }  27. |

### EJECUCIÓN

Para ejecutar el cliente de escritorio, es importante primero limpiar y construir el proyecto. Esto se logra haciendo clic derecho sobre el proyecto en el panel de Proyectos de NetBeans y seleccionando la opción Clean and Build. Este proceso elimina los archivos generados previamente y compila nuevamente el proyecto desde cero, asegurando que todos los cambios recientes en el código se apliquen correctamente antes de la ejecución.

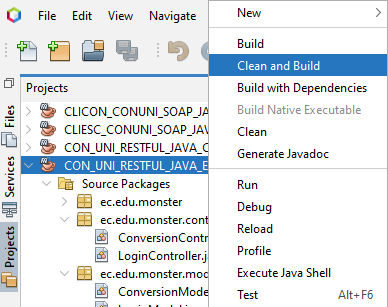


Figura 59: Clean and Build

Finalmente, dar clic derecho en el proyecto y seleccionar la opción Run.

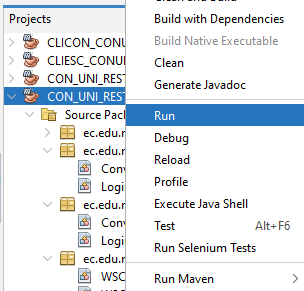


Figura 60: Ejecutar Proyecto

## APLICACIÓN CLIENTE WEB

### CREACIÓN DEL PROYECTO

Para el proyecto web se usará un entorno virtual en Python, el cual se crea ejecutando el siguiente comando:

python -m venv venv

Una vez creado, se debe activar el entorno virtual con el comando:

.\venv\Scripts\Activate

Luego, es necesario instalar las librerías requeridas, como Flask para la creación de la aplicación web y Requests para gestionar las solicitudes HTTP.

pip install flask requests

Al finalizar, se generará la carpeta correspondiente dentro del proyecto.

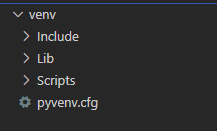


Figura 61: Carpeta Entorno Virtual

### CREACIÓN DEL MODELO MVC

Una vez creado el entorno virtual, es necesario estructurar el proyecto creando las carpetas pertinentes. En el caso de un proyecto en Python con Flask, la carpeta destinada para las vistas debe llamarse templates para que sea reconocida correctamente por el framework. Además, es común organizar los archivos estáticos (CSS, JavaScript, imágenes) en una carpeta llamada static. Con esta estructura, el proyecto quedará organizado de la siguiente manera:

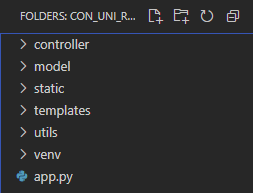


Figura 62: Estructura final del proyecto

### PAQUETE MODELO

La carpeta de modelos contiene las clases encargadas de manejar y procesar los datos obtenidos del web service. Estas clases actúan como una capa intermedia entre las respuestas JSON del servicio y las vistas o controladores, permitiendo estructurar y validar la información recibida.

Para el modelo del Login se crea la clase LoginModel, la cual se encarga de procesar las respuestas del servicio relacionadas con la autenticación. A través del método estático parse\_response, se extraen los valores de username y password del JSON devuelto por el web service.

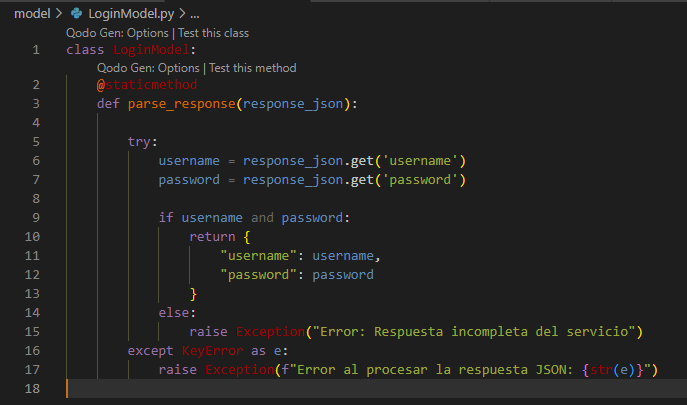


Figura 63: LoginModel

La clase ConversionModel procesa las respuestas del servicio para operaciones de conversión de unidades. El método parse\_response extrae los valores de resultado, valor, from\_unit y to\_unit del JSON proporcionado por el web service.

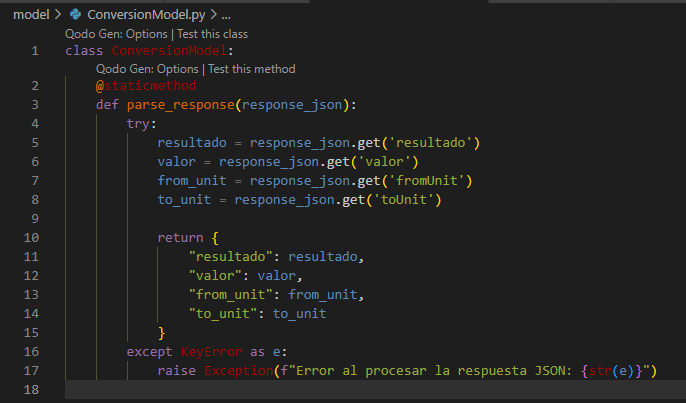


Figura 64: ConversionModel

Tabla 16: LoginModel

|  |
| --- |
| 1. class LoginModel:  2.     @staticmethod  3.     def parse\_response(response\_json):  4.  5.         try:  6.             username = response\_json.get('username')  7.             password = response\_json.get('password')  8.  9.             if username and password:  10.                 return {  11.                     "username": username,  12.                     "password": password  13.                 }  14.             else:  15.                 raise Exception("Error: Respuesta incompleta del servicio")  16.         except KeyError as e:  17.             raise Exception(f"Error al procesar la respuesta JSON: {str(e)}")  18. |

Tabla 17: ConversionModel

|  |
| --- |
| 1. class ConversionModel:  2.     @staticmethod  3.     def parse\_response(response\_json):  4.         try:  5.             resultado = response\_json.get('resultado')  6.             valor = response\_json.get('valor')  7.             from\_unit = response\_json.get('fromUnit')  8.             to\_unit = response\_json.get('toUnit')  9.  10.             return {  11.                 "resultado": resultado,  12.                 "valor": valor,  13.                 "from\_unit": from\_unit,  14.                 "to\_unit": to\_unit  15.             }  16.         except KeyError as e:  17.             raise Exception(f"Error al procesar la respuesta JSON: {str(e)}") |

### PAQUETE CONTROLADOR

La carpeta de controladores agrupa las clases responsables de la interacción directa con el web service. Estas clases implementan la lógica necesaria para realizar solicitudes HTTP, enviar datos al servicio y procesar las respuestas. De este modo, los controladores actúan como intermediarios entre la lógica del negocio y el servicio remoto.

El controlador del login implementa la funcionalidad de autenticación al realizar solicitudes POST al servicio web. A través del método estático login, se envían las credenciales (username y password) en formato JSON, verificando que la respuesta sea exitosa y que los datos del usuario sean válidos.

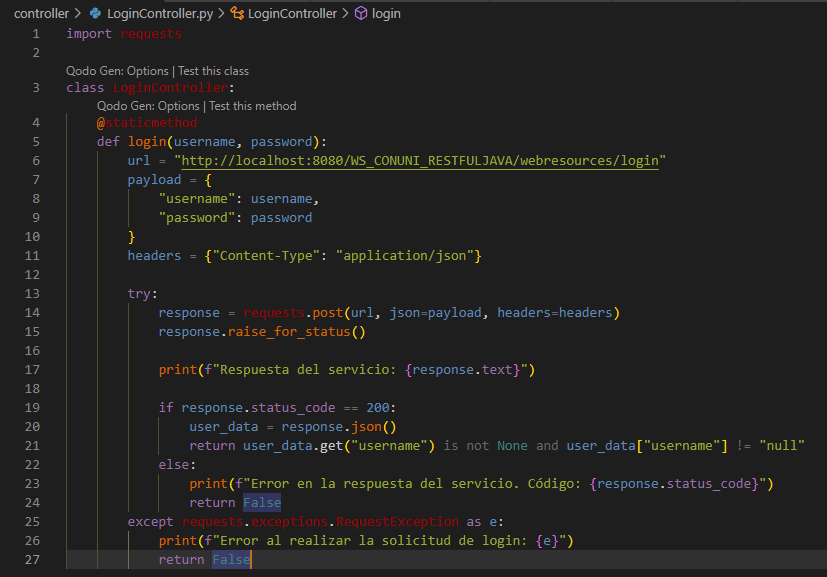


Figura 65: LoginController

Este controlador de conversión gestiona las operaciones de conversión de unidades mediante el envío de datos al web service. Utilizando el método convert, se envía un payload que contiene el valor a convertir, la unidad de origen y la unidad de destino. El controlador valida la respuesta del servicio y procesa los datos utilizando el modelo de conversión.

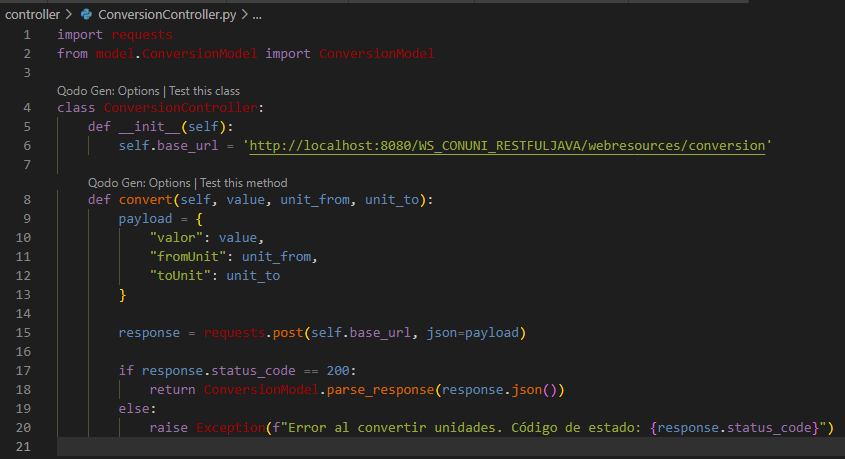


Figura 66: ConversionController

Tabla 18: LoginController

|  |
| --- |
| 1. import requests  2.  3. class LoginController:  4.     @staticmethod  5.     def login(username, password):  6.         url = "http://localhost:8080/WS\_CONUNI\_RESTFULJAVA/webresources/login"  7.         payload = {  8.             "username": username,  9.             "password": password  10.         }  11.         headers = {"Content-Type": "application/json"}  12.  13.         try:  14.             response = requests.post(url, json=payload, headers=headers)  15.             response.raise\_for\_status()  16.  17.             print(f"Respuesta del servicio: {response.text}")  18.  19.             if response.status\_code == 200:  20.                 user\_data = response.json()  21.                 return user\_data.get("username") is not None and user\_data["username"] != "null"  22.             else:  23.                 print(f"Error en la respuesta del servicio. Código: {response.status\_code}")  24.                 return False  25.         except requests.exceptions.RequestException as e:  26.             print(f"Error al realizar la solicitud de login: {e}")  27.             return False  28. |

Tabla 19: ConversionController

|  |
| --- |
| 1. import requests  2. from model.ConversionModel import ConversionModel  3.  4. class ConversionController:  5.     def \_\_init\_\_(self):  6.         self.base\_url = 'http://localhost:8080/WS\_CONUNI\_RESTFULJAVA/webresources/conversion'  7.  8.     def convert(self, value, unit\_from, unit\_to):  9.         payload = {  10.             "valor": value,  11.             "fromUnit": unit\_from,  12.             "toUnit": unit\_to  13.         }  14.  15.         response = requests.post(self.base\_url, json=payload)  16.  17.         if response.status\_code == 200:  18.             return ConversionModel.parse\_response(response.json())  19.         else:  20.             raise Exception(f"Error al convertir unidades. Código de estado: {response.status\_code}")  21. |

### PAQUETE TEMPLATES

La carpeta templates contiene las vistas dinámicas del proyecto web, implementadas con HTML.

El formulario captura el nombre de usuario y la contraseña, enviándolos al endpoint (/login). Estos datos son enviados para autenticarse contra el web service, que valida las credenciales ingresadas y devuelve una respuesta indicando si el usuario está autorizado.

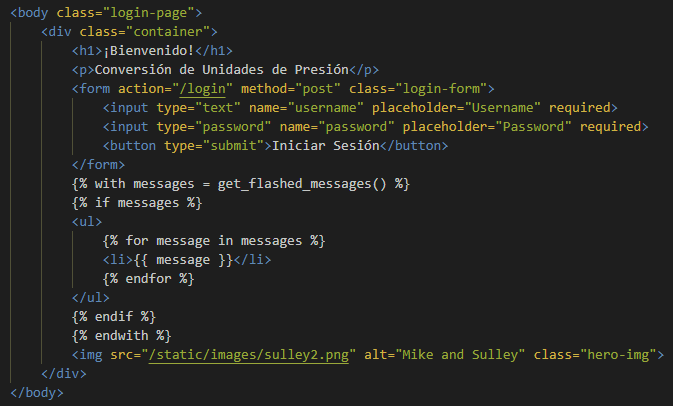


Figura 67: LoginTemplate

El formulario recoge el valor a convertir, la unidad de origen y la unidad de destino, enviando esta información al endpoint (/convert).



Figura 68: ConversionTemplate

### ARCHIVO App.py

El archivo principal del proyecto en Flask actúa como intermediario entre las vistas y los controladores para consumir el web service y manejar las interacciones del usuario.

Cuando el usuario envía sus credenciales a la ruta (/login) desde la vista, los datos se capturan mediante request.form. La contraseña se encripta usando una utilidad de hash antes de ser enviada al LoginController, que a su vez consume el web service para validar las credenciales.

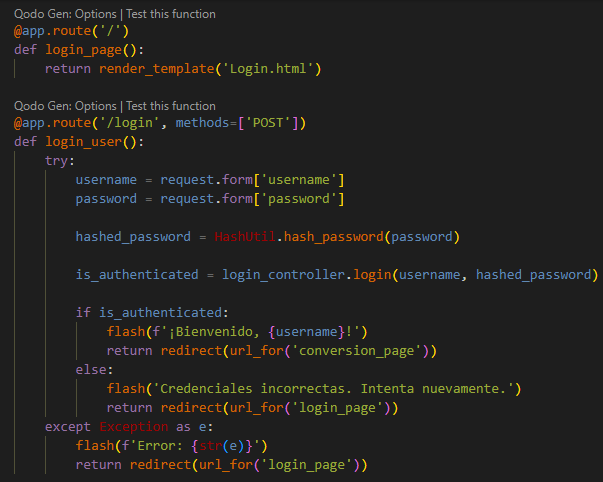


Figura 69: Ruta Login

El formulario recoge el valor a convertir, la unidad de origen y la unidad de destino, enviando esta información al endpoint /convert.

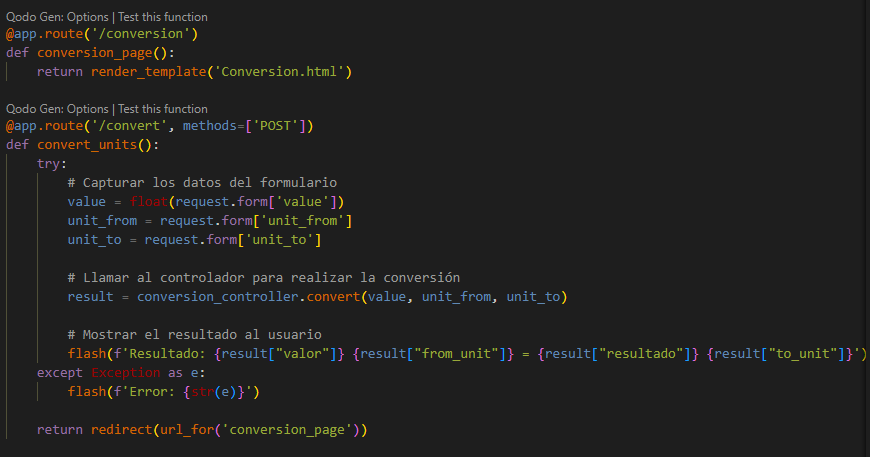


Figura 70: Ruta convert

Tabla 20: Archivo App.py

|  |
| --- |
| 1. from flask import Flask, render\_template, request, redirect, url\_for, flash  2. from utils.hash\_util import HashUtil  3. from controller.LoginController import LoginController  4. from controller.ConversionController import ConversionController  5.  6. app = Flask(\_\_name\_\_)  7. app.secret\_key = 'supersecretkey'  8.  9. # Controladores  10. login\_controller = LoginController()  11. conversion\_controller = ConversionController()  12.  13. @app.route('/')  14. def login\_page():  15.     return render\_template('Login.html')  16.  17. @app.route('/login', methods=['POST'])  18. def login\_user():  19.     try:  20.         username = request.form['username']  21.         password = request.form['password']  22.  23.         hashed\_password = HashUtil.hash\_password(password)  24.  25.         is\_authenticated = login\_controller.login(username, hashed\_password)  26.  27.         if is\_authenticated:  28.             flash(f'¡Bienvenido, {username}!')  29.             return redirect(url\_for('conversion\_page'))  30.         else:  31.             flash('Credenciales incorrectas. Intenta nuevamente.')  32.             return redirect(url\_for('login\_page'))  33.     except Exception as e:  34.         flash(f'Error: {str(e)}')  35.         return redirect(url\_for('login\_page'))  36.  37. @app.route('/conversion')  38. def conversion\_page():  39.     return render\_template('Conversion.html')  40.  41. @app.route('/convert', methods=['POST'])  42. def convert\_units():  43.     try:  44.         # Capturar los datos del formulario  45.         value = float(request.form['value'])  46.         unit\_from = request.form['unit\_from']  47.         unit\_to = request.form['unit\_to']  48.  49.         # Llamar al controlador para realizar la conversión  50.         result = conversion\_controller.convert(value, unit\_from, unit\_to)  51.  52.         # Mostrar el resultado al usuario  53.         flash(f'Resultado: {result["valor"]} {result["from\_unit"]} = {result["resultado"]} {result["to\_unit"]}')  54.     except Exception as e:  55.         flash(f'Error: {str(e)}')  56.  57.     return redirect(url\_for('conversion\_page'))  58.  59. if \_\_name\_\_ == '\_\_main\_\_':  60.     app.run(debug=True) |

### EJECUCIÓN

Para ejecutar el proyecto, primero es necesario activar el entorno virtual utilizando el comando:

.\venv\Scripts\Activate

Una vez activado el entorno, se debe iniciar la aplicación con el siguiente comando:

python App.py

La aplicación se mostrará de la siguiente manera en la dirección http://127.0.0.1:5000/

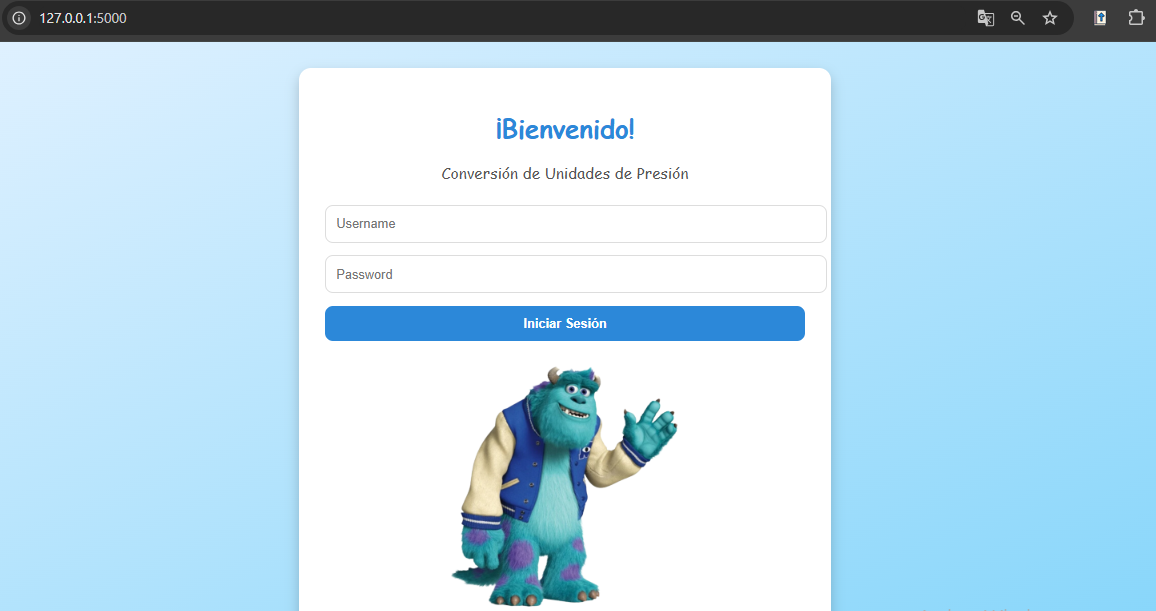


Figura 71: Pantalla Login

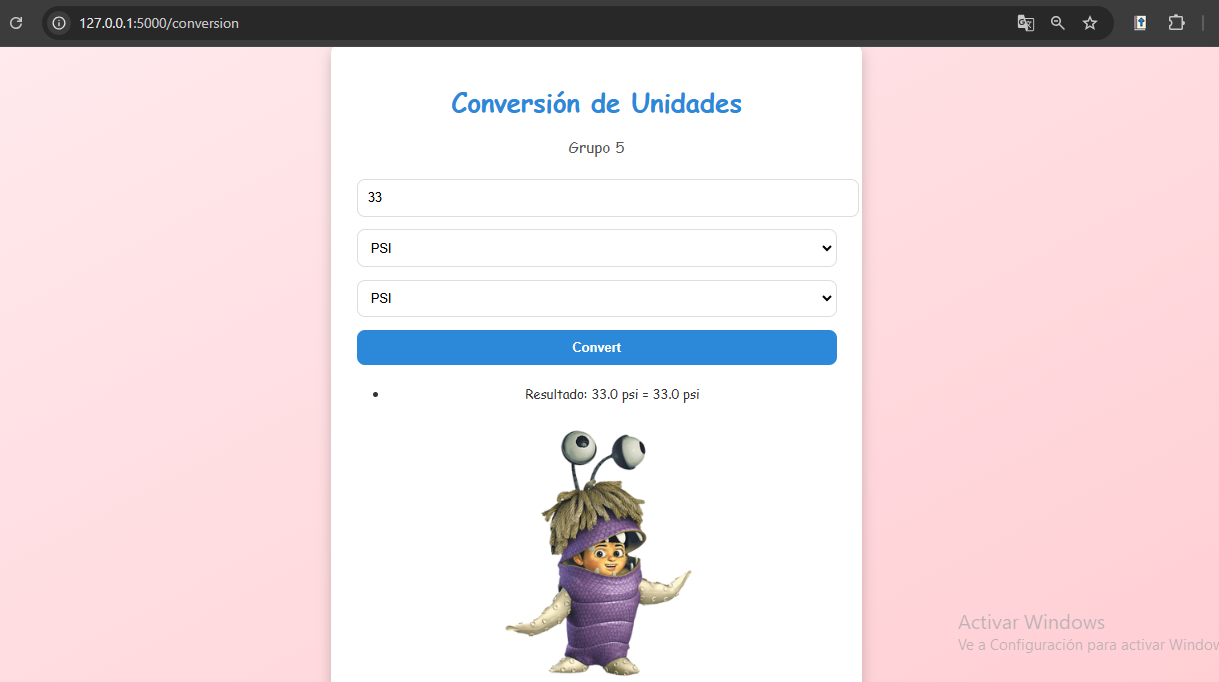


Figura 72: Pantalla Conversión

## APLICACIÓN CLIENTE MÓVIL

### CREACIÓN DEL PROYECTO CLIENTE

Para la creación del aplicativo móvil se utilizará Android Studio ya que este hace uso de Java y permite desarrollar aplicaciones para dispositivos Android. Dentro de la primera venta que muestra el aplicativo se debe seleccionar nuevo proyecto, esto permite escoger el tipo de proyecto.

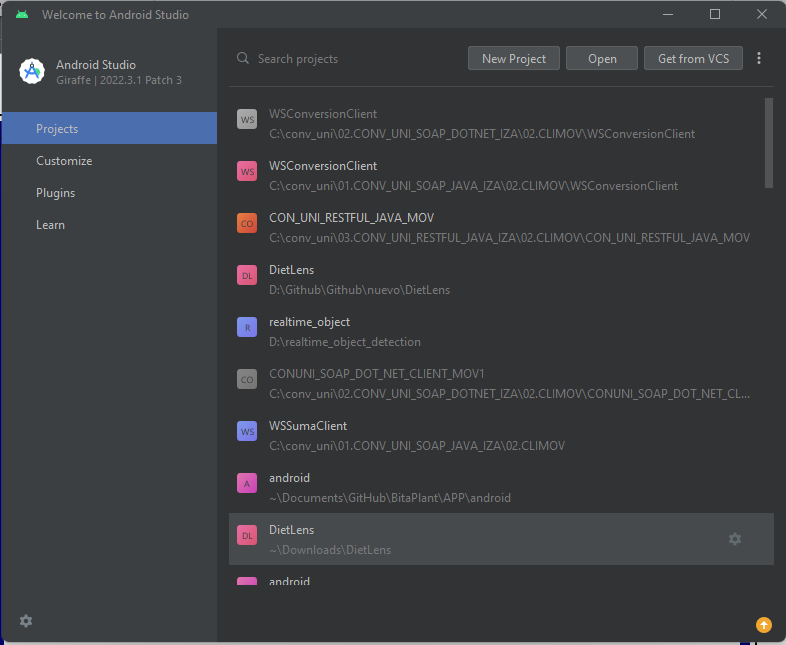


Figura 73: Ventana principal de Android Studio.

Luego se selecciona el tipo Empty Views Activity y darle a siguiente.

A screenshot of a computer

Description automatically generated

Figura 74: Tipo de proyecto de Android Studio.

Para finalizar se debe asignar el nombre y seleccionar la ubicación del proyecto, las configuraciones que vienen predefinidas no es necesario modificarlas.

A screenshot of a computer

Description automatically generated

Figura 75: Nombre y ubicación del proyecto.

Finalmente, se le da a finalizar, creando un proyecto vació para poder iniciar con el cliente móvil. El proyecto suele tomar un par de minutos en cargar y en construir su solución.

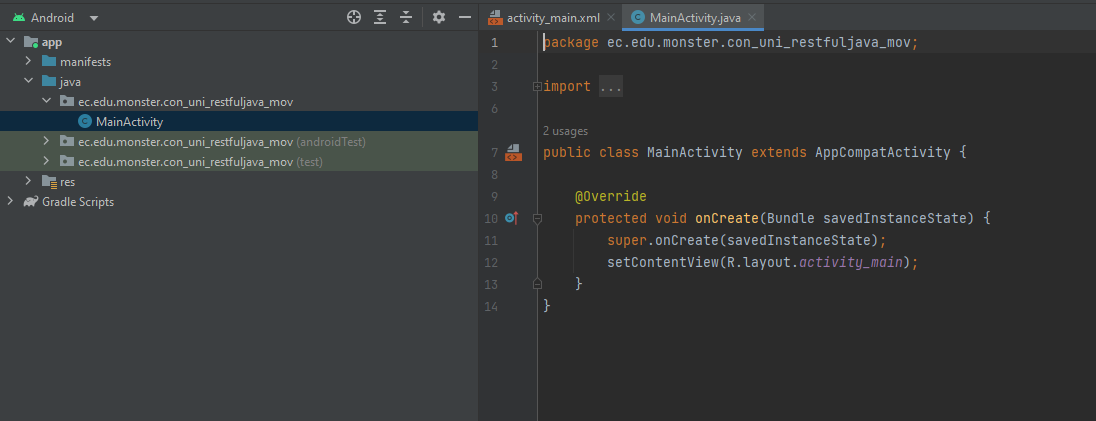


Figura 76: Proyecto vacío de Android Studio.

### CREACIÓN DEL MODELO MVC

Para la creación del proyecto, se debe implementar la arquitectura MVC. A fin de lograr esto, se da click derecho al paquete principal/Nuevo/Paquete.

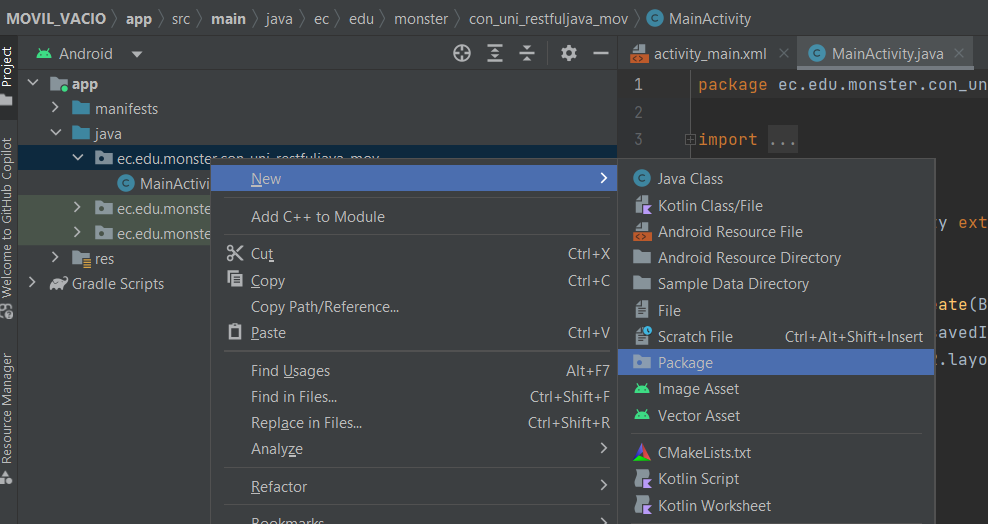


Figura 77: Creación de un paquete

Se crean 4 paquetes:

* Modelo
* Vista
* Controlador
* Servicio

A screen shot of a computer

Description automatically generated

Figura 78: Asignación de nombre al paquete

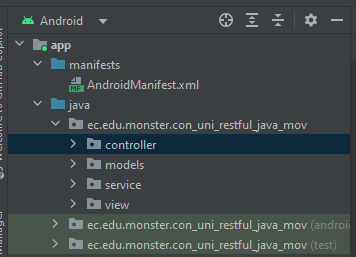


Figura 79: Estructura final de paquetes con MVC.

### PAQUETE SERVICIO

Para consumir el [servicio RESTful creado en NetBeans](#_CREACIÓN_DEL_PROYECTO) desde Android Studio es necesario implementar 2 paquetes dentro de la configuración de Android Studio, esta dependencia se llama “retrofit” y permite consumir servicios RESTful.

A screenshot of a computer program

Description automatically generated

Figura 80: Propiedades del proyecto Android Studio.

Dentro del archivo llamado “build.gradle.kts (Module:app)” se aumentan las dependencias pertenecientes a “retrofit” en la sección de implementación.



Figura 81: Configuración de las dependencias de Android Studio.

Tabla 21: Código de configuración de Android Studio.

|  |
| --- |
| 1. plugins {  2. id("com.android.application")  3. }  4.  5. android {  6. namespace = "ec.edu.monster.con\_uni\_restful\_java\_mov"  7. compileSdk = 34  8.  9. defaultConfig {  10. applicationId = "ec.edu.monster.con\_uni\_restful\_java\_mov"  11. minSdk = 34  12. targetSdk = 34  13. versionCode = 1  14. versionName = "1.0"  15.  16. testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"  17. }  18.  19. buildTypes {  20. release {  21. isMinifyEnabled = false  22. proguardFiles(  23. getDefaultProguardFile("proguard-android-optimize.txt"),  24. "proguard-rules.pro"  25. )  26. }  27. }  28. compileOptions {  29. sourceCompatibility = JavaVersion.VERSION\_1\_8  30. targetCompatibility = JavaVersion.VERSION\_1\_8  31. }  32. }  33.  34. dependencies {  35.  36. implementation("androidx.appcompat:appcompat:1.7.0")  37. implementation("com.google.android.material:material:1.12.0")  38. implementation ("com.squareup.retrofit2:retrofit:2.9.0")  39. implementation ("com.squareup.retrofit2:converter-gson:2.9.0")  40. implementation("androidx.constraintlayout:constraintlayout:2.2.0")  41. testImplementation("junit:junit:4.13.2")  42. androidTestImplementation("androidx.test.ext:junit:1.2.1")  43. androidTestImplementation("androidx.test.espresso:espresso-core:3.6.1")  44. }  45. |

Una vez implementado, se debe dar click al botón sincronizar que se encuentra en la parte superior derecha.

A screen shot of a computer

Description automatically generated

Figura 82: Sincronización de las configuraciones.

Una vez implementadas las dependencias se crea un archivo que permita realizar la conexión con el servicio. Dentro del paquete servicio, se da click derecho, luego a Nuevo/Java Class.

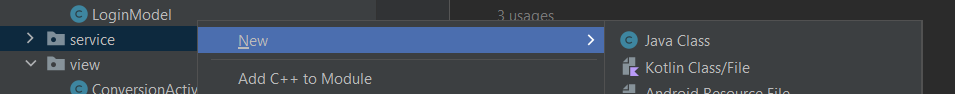


Figura 83: Creación del cliente retrofit.

Se le asigna el nombre RetrofitClient.

A screenshot of a computer

Description automatically generated

Figura 84: Asignación del nombre de la nueva clase.

Dentro de esta nueva clase se realiza la configuración del origen del servicio utilizando la funcionalidad de la dependencia. Esta permite consumir servicios a través de su URL.

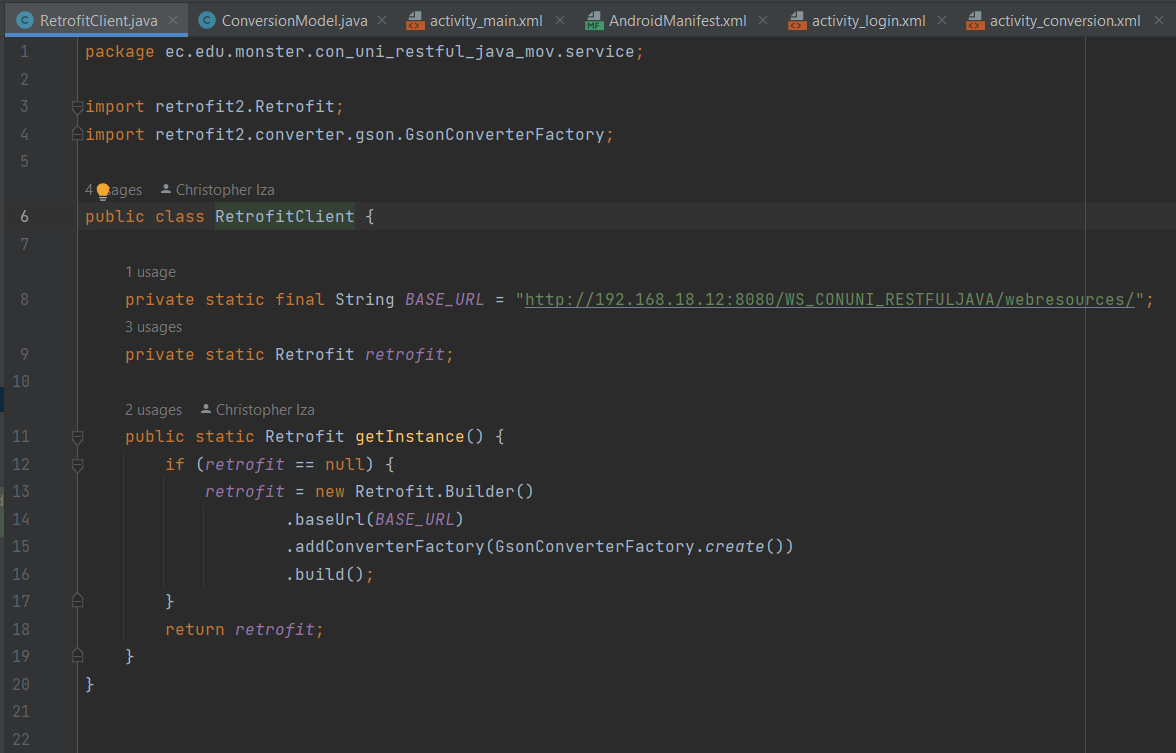


Figura 85: Implementación de la dependencia Retrofit.

Tabla 22: Código de implementación de Retrofit

|  |
| --- |
| 1. package ec.edu.monster.con\_uni\_restful\_java\_mov.service;  2.  3. import retrofit2.Retrofit;  4. import retrofit2.converter.gson.GsonConverterFactory;  5.  6. public class RetrofitClient {  7.  8. private static final String BASE\_URL = "http://192.168.18.12:8080/WS\_CONUNI\_RESTFULJAVA/webresources/";  9. private static Retrofit retrofit;  10.  11. public static Retrofit getInstance() {  12. if (retrofit == null) {  13. retrofit = new Retrofit.Builder()  14. .baseUrl(BASE\_URL)  15. .addConverterFactory(GsonConverterFactory.create())  16. .build();  17. }  18. return retrofit;  19. }  20. }  21. |

Adicionalmente, para el consumo del servicio es necesario crear una interfaz en donde se especifique el tipo de método que se va a utilizar así como el nombre respectivo del servicio.

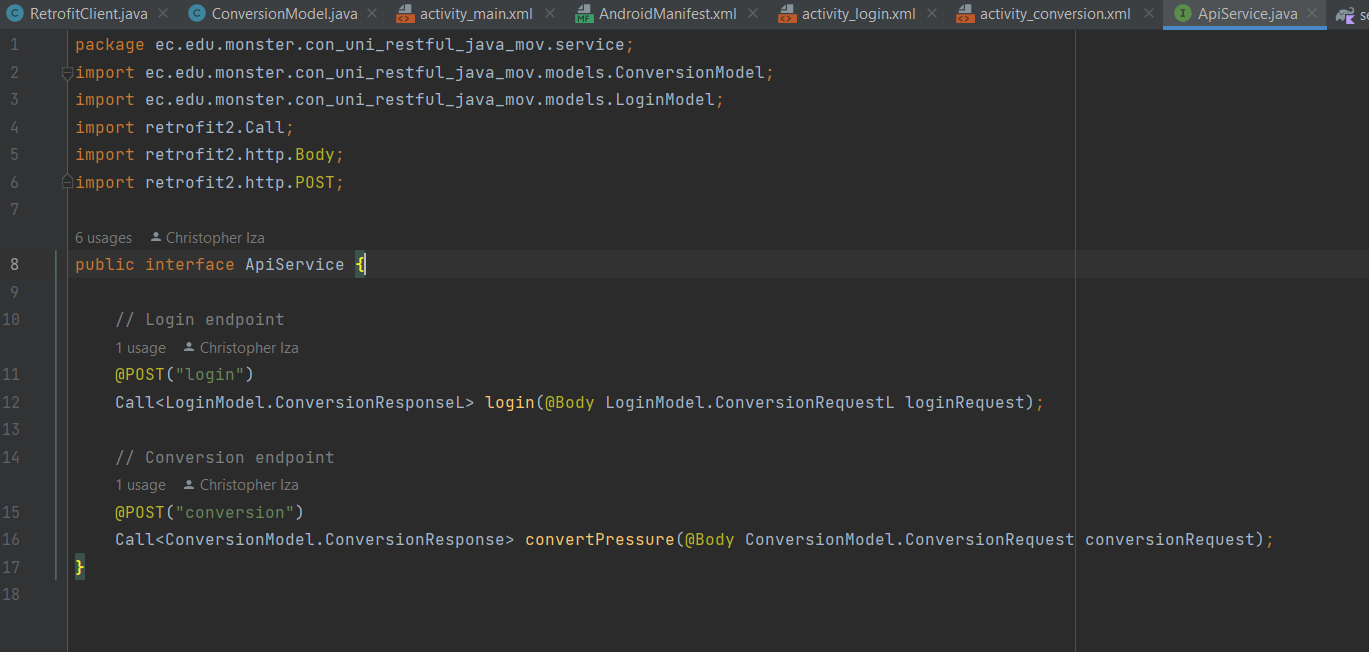


Figura 86: Interfaz de consumo del servicio.

Tabla 23:Código de la interfaz del servicio.

|  |
| --- |
| 1. package ec.edu.monster.con\_uni\_restful\_java\_mov.service;  2. import ec.edu.monster.con\_uni\_restful\_java\_mov.models.ConversionModel;  3. import ec.edu.monster.con\_uni\_restful\_java\_mov.models.LoginModel;  4. import retrofit2.Call;  5. import retrofit2.http.Body;  6. import retrofit2.http.POST;  7.  8. public interface ApiService {  9.  10. // Login endpoint  11. @POST("login")  12. Call<LoginModel.ConversionResponseL> login(@Body LoginModel.ConversionRequestL loginRequest);  13.  14. // Conversion endpoint  15. @POST("conversion")  16. Call<ConversionModel.ConversionResponse> convertPressure(@Body ConversionModel.ConversionRequest conversionRequest);  17. }  18. |

### PAQUETE MODELO

Al igual que en los clientes previos se debe trabajar con los modelos de los objetos que maneja el servicio tanto para la conversión como para la autenticación. Los modelos mantienen la misma estructura de pedido y respuesta del objeto.



Figura 87: Modelo del objeto Login

Tabla 24: Código del modelo de objeto Login.

|  |
| --- |
| 1. package ec.edu.monster.con\_uni\_restful\_java\_mov.models;  2.  3. public class LoginModel {  4. public static class ConversionRequestL {  5.  6. private String username;  7. private String password;  8.  9. public ConversionRequestL() {  10. }  11.  12. public ConversionRequestL(String username, String password) {  13. this.username = username;  14. this.password = password;  15. }  16.  17.  18. public String getUsername() {  19. return username;  20. }  21.  22. public void setUsername(String username) {  23. this.username = username;  24. }  25.  26. public String getPassword() {  27. return password;  28. }  29.  30. public void setPassword(String password) {  31. this.password = password;  32. }  33.  34. }  35.  36. public static class ConversionResponseL{  37. private String username;  38. private String password;  39.  40. public ConversionResponseL() {  41. }  42.  43. public ConversionResponseL(String username, String password){  44. this.username = username;  45. this.password = password;  46. }  47.  48. public String getUsername() {  49. return username;  50. }  51.  52. public void setUsername(String username) {  53. this.username = username;  54. }  55.  56. public String getPassword() {  57. return password;  58. }  59.  60. public void setPassword(String password) {  61. this.password = password;  62. }  63. }  64.  65.  66. }  67. |

De igual manera es necesario crear un modelo para describir los objetos de las peticiones y los resultados de la conversión.

A screenshot of a computer program

Description automatically generated

Figura 88: Modelo del objeto de conversión.

Tabla 25: Código del modelo de objeto conversión.

|  |
| --- |
| 1. package ec.edu.monster.con\_uni\_restful\_java\_mov.models;  2.  3. public class ConversionModel {  4. public static class ConversionResponse {  5. private double resultado;  6. private String toUnit;  7. private String fromUnit;  8. private double valor;  9.  10. public ConversionResponse(double valorConvertido, String unidadDestino, String unidadInicial, double valorOriginal) {  11. this.resultado = valorConvertido;  12. this.toUnit = unidadDestino;  13. this.valor = valorOriginal;  14. this.fromUnit = unidadInicial;  15. }  16.  17. public ConversionResponse() {  18. }  19.  20. public double getResultado() {  21. return resultado;  22. }  23.  24. public void setResultado(double resultado) {  25. this.resultado = resultado;  26. }  27.  28. public String getToUnit() {  29. return toUnit;  30. }  31.  32. public void setToUnit(String toUnit) {  33. this.toUnit = toUnit;  34. }  35.  36. public String getFromUnit() {  37. return fromUnit;  38. }  39.  40. public void setFromUnit(String fromUnit) {  41. this.fromUnit = fromUnit;  42. }  43.  44. public double getValor() {  45. return valor;  46. }  47.  48. public void setValor(double valor) {  49. this.valor = valor;  50. }  51. }  52. public static class ConversionRequest {  53. private double valor;  54. private String fromUnit;  55. private String toUnit;  56.  57. public ConversionRequest() {  58. }  59.  60.  61. public ConversionRequest(double valor, String fromUnit, String toUnit) {  62. this.valor = valor;  63. this.fromUnit = fromUnit;  64. this.toUnit = toUnit;  65. }  66.  67.  68. public double getValor() {  69. return valor;  70. }  71.  72. public void setValor(double valor) {  73. this.valor = valor;  74. }  75.  76. public String getFromUnit() {  77. return fromUnit;  78. }  79.  80. public void setFromUnit(String fromUnit) {  81. this.fromUnit = fromUnit;  82. }  83.  84. public String getToUnit() {  85. return toUnit;  86. }  87.  88. public void setToUnit(String toUnit) {  89. this.toUnit = toUnit;  90. }  91. }  92. }  93. |

### PAQUETE CONTROLADOR

Dentro de la arquitectura MVC el controlador es el que se encarga de gestionar la lógica del sistema, aquí se hace uso de los modelos y el servicio para implementar su uso.

Dentro del controlador de conversión se establece dos posibles estados, el exitoso y el fallido y en base al resultado obtenido se imprimirá el mensaje correspondiente.

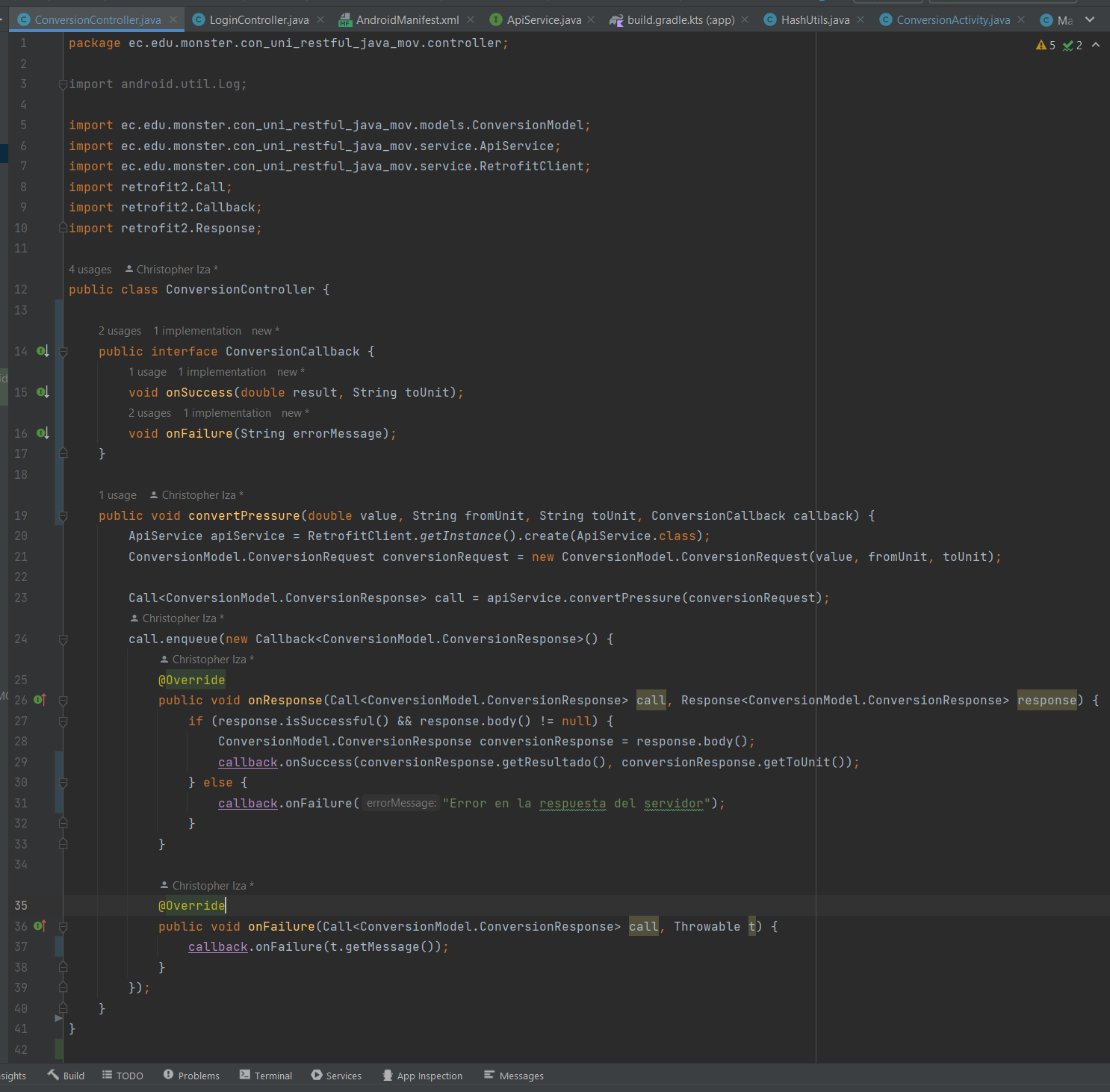


Figura 89: Controlador de conversión.

Tabla 26: Condigo del controlador de conversión.

|  |
| --- |
| 1. import retrofit2.Callback;  2. import retrofit2.Response;  3.  4. public class ConversionController {  5.  6. public interface ConversionCallback {  7. void onSuccess(double result, String toUnit);  8. void onFailure(String errorMessage);  9. }  10.  11. public void convertPressure(double value, String fromUnit, String toUnit, ConversionCallback callback) {  12. ApiService apiService = RetrofitClient.getInstance().create(ApiService.class);  13. ConversionModel.ConversionRequest conversionRequest = new ConversionModel.ConversionRequest(value, fromUnit, toUnit);  14.  15. Call<ConversionModel.ConversionResponse> call = apiService.convertPressure(conversionRequest);  16. call.enqueue(new Callback<ConversionModel.ConversionResponse>() {  17. @Override  18. public void onResponse(Call<ConversionModel.ConversionResponse> call, Response<ConversionModel.ConversionResponse> response) {  19. if (response.isSuccessful() && response.body() != null) {  20. ConversionModel.ConversionResponse conversionResponse = response.body();  21. callback.onSuccess(conversionResponse.getResultado(), conversionResponse.getToUnit());  22. } else {  23. callback.onFailure("Error en la respuesta del servidor");  24. }  25. }  26.  27. @Override  28. public void onFailure(Call<ConversionModel.ConversionResponse> call, Throwable t) {  29. callback.onFailure(t.getMessage());  30. }  31. });  32. }  33. }  34. |

Aplicando la misma lógica que en el controlador anterior se establece dos estados dentro del controlador de Login para establecer si las credenciales enviadas son las correctas o son invalidas.

A screenshot of a computer program

Description automatically generated

Figura 90: Controlador del Login

Tabla 27: Código del controlador de Login.

|  |
| --- |
| 1. package ec.edu.monster.con\_uni\_restful\_java\_mov.controller;  2. import android.content.Context;  3. import android.content.Intent;  4. import android.util.Log;  5.  6. import ec.edu.monster.con\_uni\_restful\_java\_mov.models.LoginModel;  7. import ec.edu.monster.con\_uni\_restful\_java\_mov.service.ApiService;  8. import ec.edu.monster.con\_uni\_restful\_java\_mov.view.ConversionActivity;  9. import ec.edu.monster.con\_uni\_restful\_java\_mov.service.RetrofitClient;  10. import retrofit2.Call;  11. import retrofit2.Callback;  12. import retrofit2.Response;  13. public class LoginController {  14.  15.  16. public void autenticar(String username, String password, Context context) {  17. ApiService apiService = RetrofitClient.getInstance().create(ApiService.class);  18. LoginModel.ConversionRequestL loginRequest = new LoginModel.ConversionRequestL(username, password);  19.  20. Call<LoginModel.ConversionResponseL> call = apiService.login(loginRequest);  21. call.enqueue(new Callback<LoginModel.ConversionResponseL>() {  22. @Override  23. public void onResponse(Call<LoginModel.ConversionResponseL> call, Response<LoginModel.ConversionResponseL> response) {  24. if (response.isSuccessful() && response.body() != null) {  25. LoginModel.ConversionResponseL loginResponse = response.body();  26. if (!"null".equals(loginResponse.getUsername())) {  27. // Login successful  28. Log.d("Login", "Welcome, " + loginResponse.getUsername());  29.  30. // Navigate to ConversionActivity  31. Intent intent = new Intent(context, ConversionActivity.class);  32. context.startActivity(intent);  33. } else {  34. // Invalid credentials  35. Log.d("Login", "Invalid credentials");  36. }  37. } else {  38. Log.d("Login", "Response unsuccessful or empty");  39. }  40. }  41.  42. @Override  43. public void onFailure(Call<LoginModel.ConversionResponseL> call, Throwable t) {  44. Log.e("Login", "Error: " + t.getMessage());  45. }  46. });  47. }  48.  49.  50. }  51. |

Dentro de la lógica de este sistema se tomó en cuenta que la contraseña debe transportarse de manera segura desde los clientes hacia el servidor RESTful, por lo tanto, se hizo el hash de la contraseña. Por lo mismo, fue necesario implementar una clase que se encargue de gestionar el hash de la contraseña provista por el cliente para su debida transformación.

A screen shot of a computer program

Description automatically generated

Figura 91: Clase para validar el hash de la contraseña.

Tabla 28: Código de la clase de validación de contraseña.

|  |
| --- |
| 1. package ec.edu.monster.con\_uni\_restful\_java\_mov.controller;  2.  3. import java.nio.charset.StandardCharsets;  4. import java.security.MessageDigest;  5. import java.security.NoSuchAlgorithmException;  6.  7. public class HashUtils {  8.  9. /\*\*  10. \* Hashes the input string using SHA-256.  11. \*  12. \* @param input the string to hash  13. \* @return the hashed string in hexadecimal format  14. \*/  15. public static String hashPassword(String input) {  16. try {  17. MessageDigest digest = MessageDigest.getInstance("SHA-256");  18. byte[] hashBytes = digest.digest(input.getBytes(StandardCharsets.UTF\_8));  19. StringBuilder hexString = new StringBuilder();  20.  21. for (byte b : hashBytes) {  22. String hex = Integer.toHexString(0xff & b);  23. if (hex.length() == 1) {  24. hexString.append('0');  25. }  26. hexString.append(hex);  27. }  28.  29. return hexString.toString();  30. } catch (NoSuchAlgorithmException e) {  31. throw new RuntimeException("SHA-256 algorithm not found", e);  32. }  33. }  34. }  35. |

### PAQUETE VISTA

Dentro de Android Studio se trabaja la parte de las pantallas o vistas con actividades, en donde se gestionan los recursos de la pantalla. En este caso la pantalla principal donde se encuentran los llamados a los controladores del login se llama MainActivity. Desde aquí se recoge la información y se la envía al controlador para que procese esa información.

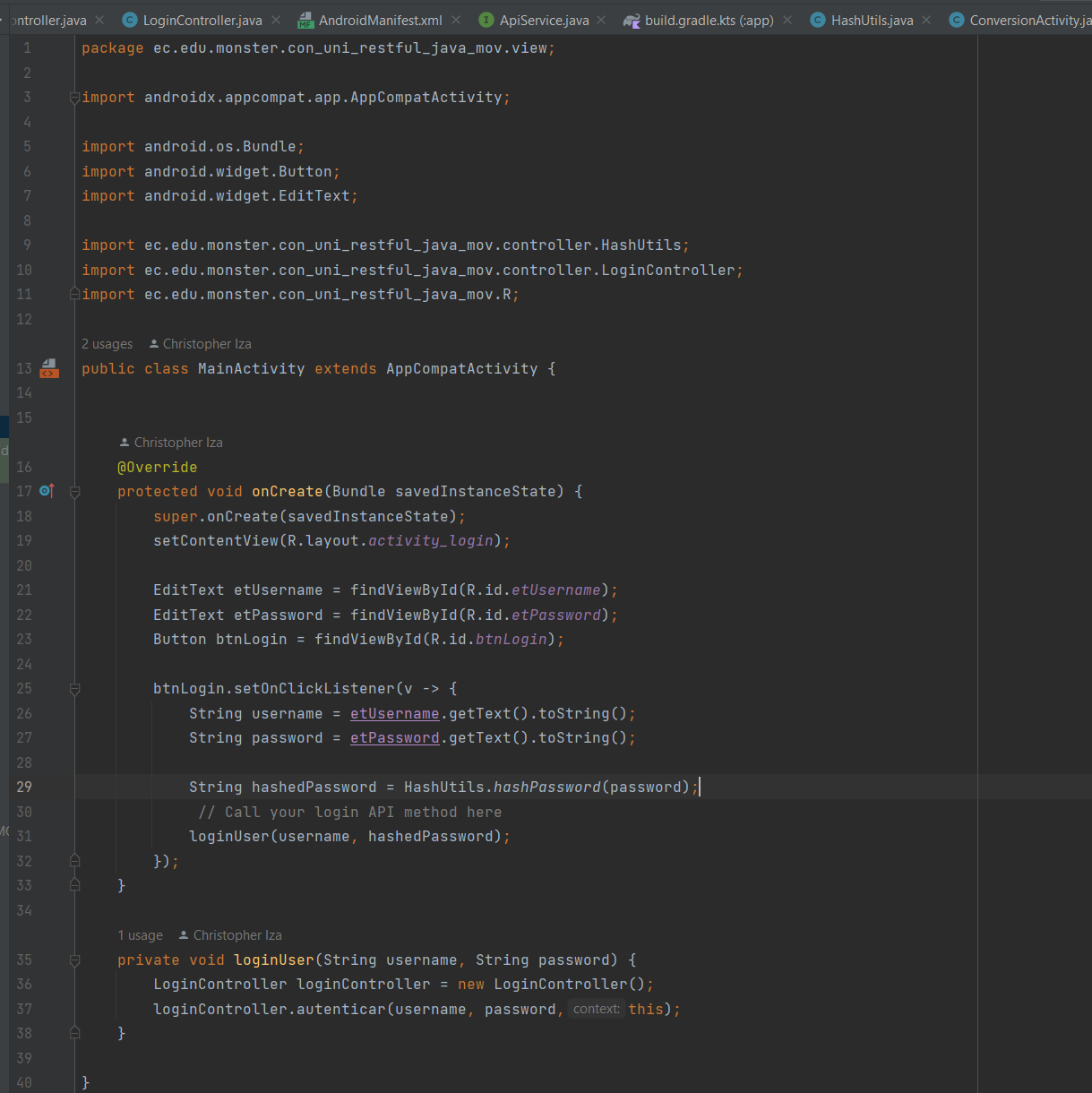


Figura 92: Vista principal del proyecto.

Tabla 29: Código de la vista principal del cliente.

|  |
| --- |
| 1. package ec.edu.monster.con\_uni\_restful\_java\_mov.view;  2.  3. import androidx.appcompat.app.AppCompatActivity;  4.  5. import android.os.Bundle;  6. import android.widget.Button;  7. import android.widget.EditText;  8.  9. import ec.edu.monster.con\_uni\_restful\_java\_mov.controller.HashUtils;  10. import ec.edu.monster.con\_uni\_restful\_java\_mov.controller.LoginController;  11. import ec.edu.monster.con\_uni\_restful\_java\_mov.R;  12.  13. public class MainActivity extends AppCompatActivity {  14.  15.  16. @Override  17. protected void onCreate(Bundle savedInstanceState) {  18. super.onCreate(savedInstanceState);  19. setContentView(R.layout.activity\_login);  20.  21. EditText etUsername = findViewById(R.id.etUsername);  22. EditText etPassword = findViewById(R.id.etPassword);  23. Button btnLogin = findViewById(R.id.btnLogin);  24.  25. btnLogin.setOnClickListener(v -> {  26. String username = etUsername.getText().toString();  27. String password = etPassword.getText().toString();  28.  29. String hashedPassword = HashUtils.hashPassword(password);  30. // Call your login API method here  31. loginUser(username, hashedPassword);  32. });  33. }  34. private void loginUser(String username, String password) {  35. LoginController loginController = new LoginController();  36. loginController.autenticar(username, password,this);  37. }  38.  39. } |

Una vez que se valida adecuadamente las credenciales se puede pasar a la siguiente pantalla, la pantalla de conversión, en donde se puede ingresar la información de del valor a convertir y escoger el tipo de conversión, así como ver el resultado.

A screenshot of a computer program

Description automatically generated

Figura 93:Vista de la sección de conversión del proyecto.

Tabla 30: Código de la vista de conversión del cliente.

|  |
| --- |
| 1. package ec.edu.monster.con\_uni\_restful\_java\_mov.view;  2.  3. import android.os.Bundle;  4. import android.widget.ArrayAdapter;  5. import android.widget.Button;  6. import android.widget.EditText;  7. import android.widget.Spinner;  8. import android.widget.TextView;  9. import android.widget.Toast;  10. import java.util.HashMap;  11. import androidx.appcompat.app.AppCompatActivity;  12.  13. import ec.edu.monster.con\_uni\_restful\_java\_mov.R;  14. import ec.edu.monster.con\_uni\_restful\_java\_mov.controller.ConversionController;  15. import ec.edu.monster.con\_uni\_restful\_java\_mov.models.ConversionModel;  16.  17. public class ConversionActivity extends AppCompatActivity {  18.  19. private EditText etValue;  20. private Spinner spnFromUnit, spnToUnit;  21. private TextView tvResult;  22. private Button btnConvert;  23.  24. @Override  25. protected void onCreate(Bundle savedInstanceState) {  26. super.onCreate(savedInstanceState);  27. setContentView(R.layout.activity\_conversion);  28.  29. // Vincular elementos de la vista  30. etValue = findViewById(R.id.etValue);  31. spnFromUnit = findViewById(R.id.spnFromUnit);  32. spnToUnit = findViewById(R.id.spnToUnit);  33. tvResult = findViewById(R.id.tvResult);  34. btnConvert = findViewById(R.id.btnConvert);  35.  36. // Opciones para las unidades de conversión  37. String[] units = {"Pa", "Bar", "Psi", "Atm", "mmHg"};  38.  39. // Configurar los spinners con las unidades  40. ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple\_spinner\_item, units);  41. adapter.setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item);  42. spnFromUnit.setAdapter(adapter);  43. spnToUnit.setAdapter(adapter);  44.  45. // Acción al presionar el botón de conversión  46. btnConvert.setOnClickListener(v -> {  47. try {  48. // Obtener valores de entrada  49. String fromUnit = spnFromUnit.getSelectedItem().toString();  50. String toUnit = spnToUnit.getSelectedItem().toString();  51. double value = Double.parseDouble(etValue.getText().toString());  52.  53. // Llamar al servicio de conversión  54. performConversion(value, fromUnit, toUnit);  55. } catch (NumberFormatException e) {  56. Toast.makeText(this, "Ingrese un valor válido", Toast.LENGTH\_SHORT).show();  57. }  58. });  59. }  60.  61. private void performConversion(double value, String fromUnit, String toUnit) {  62. // Llama al servicio usando ConversionController  63. ConversionController conversionController = new ConversionController();  64. conversionController.convertPressure(value, fromUnit, toUnit, new ConversionController.ConversionCallback() {  65. @Override  66. public void onSuccess(double result, String toUnit) {  67. runOnUiThread(() -> {  68. tvResult.setText(String.format("Resultado: %.4f %s", result, toUnit));  69. });  70. }  71.  72. @Override  73. public void onFailure(String errorMessage) {  74. runOnUiThread(() -> {  75. Toast.makeText(ConversionActivity.this, "Error: " + errorMessage, Toast.LENGTH\_SHORT).show();  76. });  77. }  78. });  79. }  80.  81. } |

Finalmente, estas vistas trabajan con archivos donde se diseñan los elementos, estos son los layout de las pantallas. A continuación, se muestran los layouts de las pantallas en el formato xml.

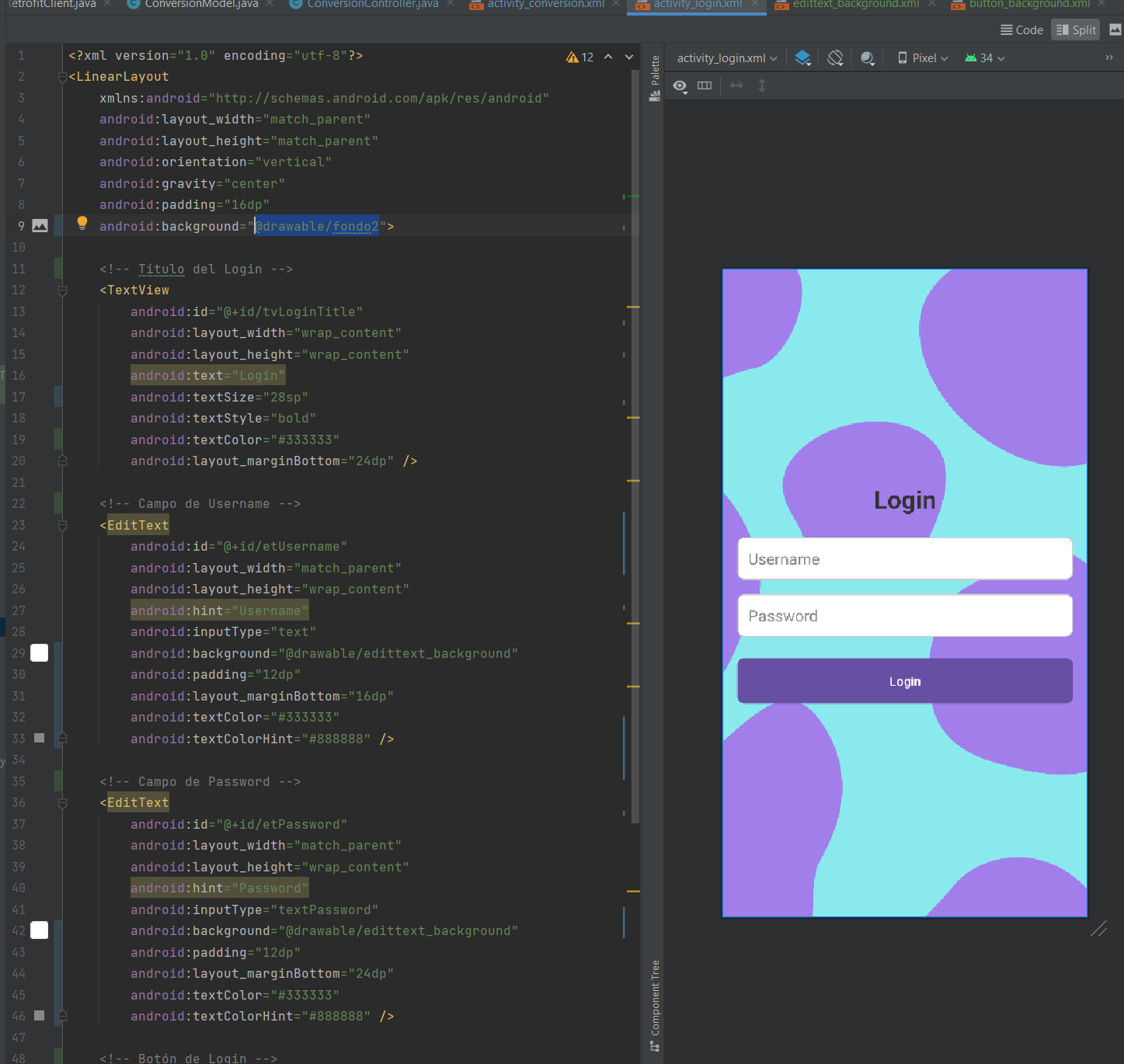


Figura 94: Diseño de la pantalla de Login.

Tabla 31: Código xml del diseño de la pantalla Login.

|  |
| --- |
| 1. <?xml version="1.0" encoding="utf-8"?>  2. <LinearLayout  3. xmlns:android="http://schemas.android.com/apk/res/android"  4. android:layout\_width="match\_parent"  5. android:layout\_height="match\_parent"  6. android:orientation="vertical"  7. android:gravity="center"  8. android:padding="16dp"  9. android:background="@drawable/fondo2">  10.  11. <!-- Título del Login -->  12. <TextView  13. android:id="@+id/tvLoginTitle"  14. android:layout\_width="wrap\_content"  15. android:layout\_height="wrap\_content"  16. android:text="Login"  17. android:textSize="28sp"  18. android:textStyle="bold"  19. android:textColor="#333333"  20. android:layout\_marginBottom="24dp" />  21.  22. <!-- Campo de Username -->  23. <EditText  24. android:id="@+id/etUsername"  25. android:layout\_width="match\_parent"  26. android:layout\_height="wrap\_content"  27. android:hint="Username"  28. android:inputType="text"  29. android:background="@drawable/edittext\_background"  30. android:padding="12dp"  31. android:layout\_marginBottom="16dp"  32. android:textColor="#333333"  33. android:textColorHint="#888888" />  34.  35. <!-- Campo de Password -->  36. <EditText  37. android:id="@+id/etPassword"  38. android:layout\_width="match\_parent"  39. android:layout\_height="wrap\_content"  40. android:hint="Password"  41. android:inputType="textPassword"  42. android:background="@drawable/edittext\_background"  43. android:padding="12dp"  44. android:layout\_marginBottom="24dp"  45. android:textColor="#333333"  46. android:textColorHint="#888888" />  47.  48. <!-- Botón de Login -->  49. <Button  50. android:id="@+id/btnLogin"  51. android:layout\_width="match\_parent"  52. android:layout\_height="wrap\_content"  53. android:text="Login"  54. android:background="@drawable/button\_background"  55. android:textColor="#ffffff"  56. android:padding="12dp" />  57. </LinearLayout>  58. |

De la misma manera se tiene el layout de la pantalla de conversión.

A screenshot of a computer

Description automatically generated

Figura 95: Diseño de la página de conversión.

Tabla 32: Código xml del diseño de la pantalla conversión.

|  |
| --- |
| 1. <LinearLayout  2. xmlns:android="http://schemas.android.com/apk/res/android"  3. android:layout\_width="match\_parent"  4. android:layout\_height="match\_parent"  5. android:orientation="vertical"  6. android:gravity="center"  7. android:background="@drawable/fondo2"  8. android:padding="16dp">  9.  10. <TextView  11. android:id="@+id/tvTitle"  12. android:layout\_width="wrap\_content"  13. android:layout\_height="wrap\_content"  14. android:text="Conversión de Presión"  15. android:textStyle="bold"  16. android:textSize="28sp"  17.  18.  19. android:layout\_gravity="center\_horizontal"  20. android:paddingBottom="16dp"  21. android:textColor="#0D5B8F" />  22.  23. <EditText  24. android:id="@+id/etValue"  25. android:layout\_width="match\_parent"  26. android:layout\_height="wrap\_content"  27. android:hint="Ingrese el valor"  28. android:inputType="numberDecimal"  29. android:padding="8dp"  30. android:background="@android:drawable/editbox\_background"  31. android:textColor="#00796b"  32. android:layout\_marginTop="8dp" />  33.  34. <Spinner  35. android:id="@+id/spnFromUnit"  36. android:layout\_width="match\_parent"  37. android:layout\_height="wrap\_content"  38. android:padding="8dp"  39. android:layout\_marginTop="16dp"  40. android:background="#ffffff" />  41.  42. <Spinner  43. android:id="@+id/spnToUnit"  44. android:layout\_width="match\_parent"  45. android:layout\_height="wrap\_content"  46. android:padding="8dp"  47. android:layout\_marginTop="8dp"  48. android:background="#ffffff" />  49.  50. <Button  51. android:id="@+id/btnConvert"  52. android:layout\_width="match\_parent"  53. android:layout\_height="wrap\_content"  54. android:text="Convertir"  55. android:textColor="#ffffff"  56. android:background="#00796b"  57. android:layout\_marginTop="16dp" />  58.  59. <TextView  60. android:id="@+id/tvResult"  61. android:layout\_width="match\_parent"  62. android:layout\_height="wrap\_content"  63. android:text="Resultado: "  64. android:textSize="16sp"  65. android:layout\_marginTop="16dp"  66. android:padding="8dp"  67. android:background="#ffffff"  68. android:textColor="#00796b"  69. android:gravity="center" />  70. </LinearLayout>  71. |

### EJECUCIÓN

Finalmente, se puede probar la ejecución del programa utilizando un simulador o conectando por USB un teléfono Android. En este caso se probó utilizando un teléfono, dándole click al botón de play de Android Studio.

A screenshot of a computer

Description automatically generated

Figura 96: Botón para ejecutar el cliente de móvil.

Screens screenshot of a mobile phone

Description automatically generated

Figura 97: Ejecución del cliente de móvil.

# Conclusiones

La implementación del servicio RESTful en Java demostró ser una solución eficiente y flexible para la comunicación entre clientes y servidores. Este enfoque facilita la integración con múltiples clientes, independientemente de su plataforma o lenguaje de desarrollo, destacando la interoperabilidad del protocolo HTTP y los formatos estándar como JSON o XML.

La arquitectura RESTful permitió estandarizar las operaciones mediante métodos HTTP (GET, POST, PUT, DELETE), simplificando la estructura del sistema y facilitando su comprensión y mantenimiento. Esto asegura que los desarrolladores puedan interactuar con el servicio de manera intuitiva y predecible.

La implementación de los cuatro clientes (posiblemente aplicaciones web, móviles, o de escritorio) y un servicio RESTful único cumplió con los objetivos planteados. Cada cliente pudo interactuar exitosamente con el servicio, validando tanto el diseño de las rutas y recursos como la correcta manipulación de solicitudes y respuestas.

# Recomendaciones

Se recomienda el uso de pruebas unitarias y de integración para asegurar la calidad del servicio RESTful, utilizando frameworks como JUnit para el servicio y herramientas como Postman o Newman para pruebas de los clientes.

Para escenarios de alta demanda, sería beneficioso implementar técnicas de optimización como la caché para solicitudes frecuentes, el uso de conexiones persistentes o la compresión de datos. Estas prácticas pueden mejorar la velocidad y reducir el uso de recursos del servidor.

Dado que los servicios RESTful suelen manejar datos sensibles, se recomienda implementar medidas de seguridad como autenticación mediante tokens (JWT), encriptación HTTPS y validación estricta de entradas para evitar vulnerabilidades como inyecciones o accesos no autorizados.