DISEÑO DE PATRONES

TEMA:

Plataforma para el Empleo Juvenil (ODS 8: Trabajo Decente y Crecimiento Económico)

TÍTULO:

JOVENES360

DOCENTE:

Guevara Jiménez Jorge Alfredo

INTEGRANTES:

- Gutiérrez Cuéllar Josué Ángel (U23219921)
- Fernandez Cabrera Boris (U23214383)
- León Moreno Marco Antonio (U23258586)
- Helian (Deepseek)

SECCIÓN:

21646

2025

INDICE

CAPITULO 1:

Introducción

El Objetivo de Desarrollo Sostenible (ODS) del número 8, establecido por la agenda 2030 de las Naciones Unidas, tiene como propósito promover el crecimiento económico sostenido, inclusivo y sostenible, el empleo pleno y productivo y el trabajo decente para todos. Por lo que, este objetivo surge como respuesta a los desafíos estructurales que enfrentan las economías actuales, entre ellos el desempleo juvenil, la informalidad laboral, la desigualdad salarial, y la necesidad de garantizar condiciones dignas de trabajo.

El ODS 8 reconoce que el trabajo no solo es una fuente de ingresos, sino también una base fundamental para la dignidad humana, la estabilidad social y el desarrollo económico. Sin embargo, trabajar no garantiza por sí solo una vida digna: actualmente, millones de personas en todo el mundo siguen atrapadas en condiciones de trabajo forzoso, empleo informal y bajos ingresos, especialmente en países en vías de desarrollo. Ya que, se estima que 192 millones de personas estaban desempleadas en 2022 y que el desempleo juvenil triplicaba al de los adultos, siendo una de las poblaciones más afectadas por la falta de oportunidades. Uno de los mayores desafíos globales es ofrecer trabajo decente para los jóvenes, quienes enfrentan obstáculos como la falta de experiencia, la escasa formación profesional, y la desconexión entre el sistema educativo y las demandas del mercado laboral. En 2023, aproximadamente 22% de los jóvenes del mundo no estudiaban, ni trabajaban, ni recibían formación, lo que representa un riesgo no solo para su desarrollo personal, sino también para el crecimiento económico de los países.

En este contexto, resulta fundamental crear mecanismos y plataformas digitales que faciliten la inserción laboral de los jóvenes, fomenten el aprendizaje continuo y mejoren el acceso a empleos de calidad. La inversión en educación, capacitación técnica, inclusión financiera y derechos laborales es esencial para alcanzar un crecimiento verdaderamente inclusivo y equitativo.

El proyecto que aquí se presenta tiene como finalidad el diseño e implementación de una plataforma digital orientada a la empleabilidad juvenil, que integre herramientas para la creación de perfiles profesionales, búsqueda activa de empleo, y acceso a formación especializada. Esta iniciativa busca responder directamente a las metas del ODS 8, ofreciendo a los jóvenes una vía efectiva para mejorar sus oportunidades laborales, reducir las desigualdades y avanzar hacia un desarrollo económico más justo, sostenible y humano.

1. ODS 8: Trabajo Decente y Crecimiento Económico

El **Objetivo de Desarrollo Sostenible (ODS) 8**, establecido por la ONU en la Agenda 2030, busca:

- Promover **empleo pleno y productivo** para todos.
- Garantizar **trabajo decente** (digno, seguro, con derechos laborales).
- Reducir el **desempleo juvenil** y la **informalidad laboral**.

Datos clave (problema global):

- 22% de jóvenes no estudian ni trabajan (*2023, OIT*).
- El desempleo juvenil triplica al de adultos.
- Brecha de habilidades: 40% de empleadores no encuentra candidatos calificados (*Banco Mundial*).

---

2. Problema Específico

En el contexto peruano/latinoamericano:

- **Falta de acceso** a empleos formales para jóvenes sin experiencia.
- **Desconexión** entre educación y demanda laboral (ej: habilidades técnicas vs. lo que piden las empresas).
- **Plataformas existentes** (como LinkedIn) no están diseñadas para jóvenes con perfiles junior o sin red profesional.

---

3. Solución Propuesta: Plataforma JOVENES360

Una **plataforma digital integrada** que combina:

- **Perfiles profesionales adaptados**: Para jóvenes con poca experiencia (ej: destacar proyectos académicos, voluntariados).
- **Recomendaciones inteligentes**: Usando IA para sugerir empleos/cursos basados en habilidades.

- **Formación accesible**: Cursos cortos certificados (ej: programación, marketing digital).
- **Enfoque comunitario**: Red social estilo TikTok para que empresas muestren su cultura laboral con videos.

Tecnologías clave:

- Backend: Java + Spring Boot (APIs REST).
- Base de datos: MySQL.
- Principios SOLID y patrones de diseño (Factory, Facade).

---

4. Objetivos del Proyecto

El proyecto **"JOVENES360"** tiene como objetivo principal crear una plataforma digital inclusiva, interactiva y juvenil que impulse la **empleabilidad, el desarrollo profesional y la capacitación continua de jóvenes** a través de un entorno dinámico que fusiona la experiencia visual de redes como TikTok con la funcionalidad profesional de plataformas como LinkedIn.

- **Módulo de Creación de Perfiles Profesionales:** Los jóvenes pueden crear y gestionar sus perfiles profesionales con CV y habilidades.
- **Módulo de Búsqueda de Empleo:** Los usuarios pueden buscar empleos que se ajusten a su perfil y habilidades.
- **Módulo de Capacitación y Formación:** Ofrece cursos y formación para mejorar las habilidades de los jóvenes.
- **Módulo de Recomendación Inteligentes:** Recomendaciones personalizadas de empleos y cursos según el perfil, historial y objetivos del usuario.
- **Módulo de subida y gestión de certificados:** Los usuarios pueden subir certificados de cursos externos, talleres, voluntarios u otras experiencias formativas.
-

---

5. Requerimientos Funcionales:

1.Los usuarios pueden **crear y actualizar su perfil** profesional en línea.

2.El sistema permite **buscar y aplicar a empleos** según el perfil del usuario.

3.Los usuarios **pueden acceder a recursos educativos** para mejorar sus habilidades.

4.El sistema proporciona **recomendaciones personalizadas de empleos o cursos**.

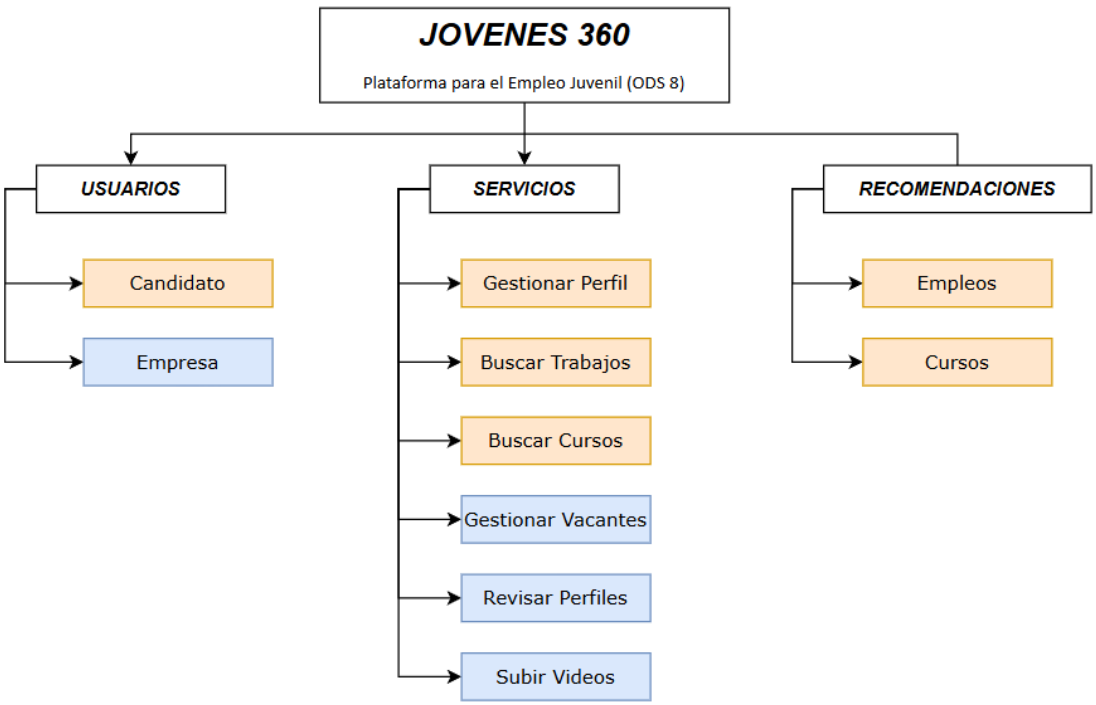
5.El sistema debe permitir a los empleadores **publicar vacantes y revisar perfiles** de candidatos.

6.Las empresas pueden subir videos sobre empleos.

A través de esta plataforma, se busca **reducir las barreras de acceso al empleo decente**, sobre todo en contextos donde la juventud enfrenta desempleo, informalidad o falta de oportunidades, integrando además un componente de comunidad y participación entre usuarios.

---

6. Diagrama de Descomposición Modular:



CAPITULO 2:

UNIDAD I – Diseño la arquitectura con la aplicación de patrones de diseño a detalle:

Desarrollo de principios ISP (Capturas de código y ejecución):

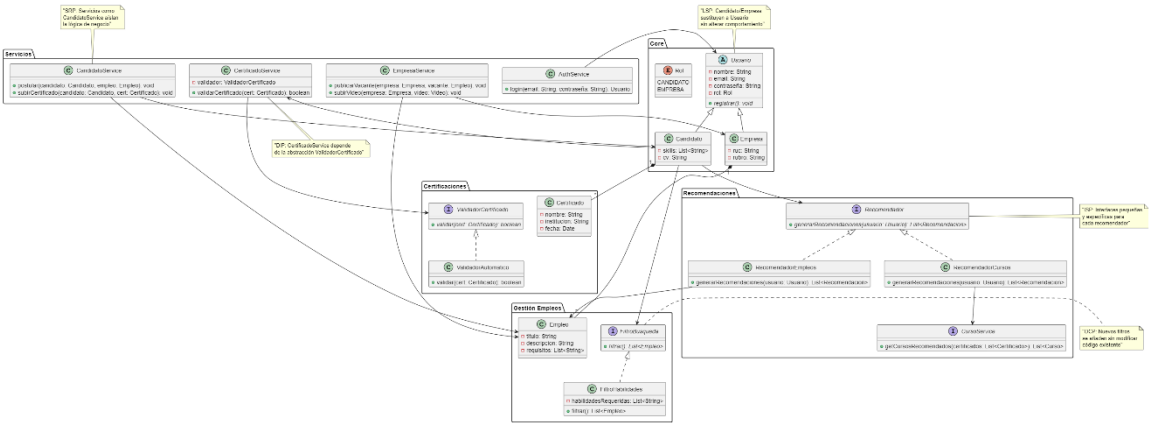


Imagen 1: Diagrama UML de Clases Completo

2.1) Principios SOLID y su Aplicación

Los principios SOLID son un conjunto de buenas prácticas de diseño orientado a objetos que buscan hacer el código más mantenible, flexible y fácil de entender. El proyecto aplica estos principios para lograr un diseño limpio y modular. A continuación, se explican cada principio y su integración en el proyecto:

1. SRP - Principio de Responsabilidad Única (Single Responsibility Principle)

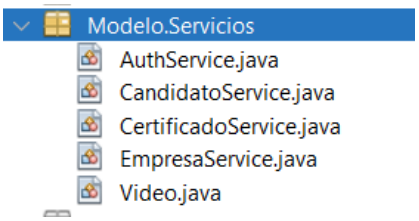
Este principio indica que una clase o módulo debe tener una única razón para cambiar, es decir, una sola responsabilidad.

Aplicación:

- En el proyecto, la lógica de negocio se encapsula en servicios especializados, como CandidatoService, EmpresaService y CertificadoService.
- Cada servicio maneja exclusivamente tareas relacionadas con su área, evitando mezclar responsabilidades. Por ejemplo, CandidatoService se encarga de la gestión de candidatos, mientras que CertificadoService valida certificados mediante un validador.

Captura de carpetas, código y ejecución:

CARPETAS:



CODIGO:

CandidatoService.java | CertificadoService.java | EmpresaService.java

Source | History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this l
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo.Servicios;

/**
 *
 * @author josue
 */
import Modelo.Core.Candidato;
import Modelo.GestionEmpleos.Empleo;
import Modelo.Certificaciones.Certificado;

public class CandidatoService {

    public void postular(Candidato candidato, Empleo empleo) {
        // Lógica para postular a un empleo
        System.out.println(candidato.getNombre() + " postuló a " + empleo.getTitulo());
    }

    public void subirCertificado(Candidato candidato, Certificado cert) {
        // Lógica para subir certificado
        System.out.println("Certificado subido para " + candidato.getNombre());
    }
}
```

CandidatoService.java | CertificadoService.java | EmpresaService.java

Source | History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo.Servicios;

/**
 *
 * @author josue
 */
import Modelo.Certificaciones.Certificado;
import Modelo.Certificaciones.ValidadorCertificado;

public class CertificadoService {
    private ValidadorCertificado validador;

    public CertificadoService(ValidadorCertificado validador) {
        this.validador = validador;
    }

    public boolean validarCertificado(Certificado cert) {
        return validador.validar(cert);
    }
}
```

CandidatoService.java × CertificadoService.java × EmpresaService.java ×

Source History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo.Servicios;

/**
 *
 * @author josue
 */
import Modelo.Core.Empresa;
import Modelo.GestionEmpleos.Empleo;

public class EmpresaService {

    public void publicarVacante(Empresa empresa, Empleo vacante) {
        // Lógica para publicar vacante
        System.out.println("Vacante publicada por " + empresa.getNombre());
    }

    public void subirVideo(Empresa empresa, Video video) {
        // Lógica para subir video
        System.out.println("Video subido por " + empresa.getNombre());
    }
}
```

EJECUCIÓN:

Main.java × Certificado.java × CandidatoService.java × CertificadoService.java × AuthService.java ×

Source History

```java
public class Main {
    public static void main(String[] args) {
        try {
            // Crear candidato con datos de prueba
            Candidato candidato = new Candidato(
                "Josue",
                "josue@gmail.com",
                "1234",
                Arrays.asList("Java", "SQL", "Spring"),
                "link_a_cv.pdf"
            );

            // Crear empleo de prueba
            Empleo empleo = new Empleo("Desarrollador Java", "Backend con Spring", Arrays.asList("Java", "SQL"));

            // Servicio de postulación
            CandidatoService candidatoService = new CandidatoService();
            candidatoService.postular(candidato, empleo);

            // Servicio de certificados
            ValidadorCertificado validador = new ValidadorAutomatico();
            CertificadoService certService = new CertificadoService(validador);

            // Crear certificado con fecha en formato Date
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            Date fechaCertificado = sdf.parse("2023-12-01");

            Certificado cert = new Certificado("CS50", "Harvard", fechaCertificado);
            candidatoService.subirCertificado(candidato, cert);

            boolean valido = certService.validarCertificado(cert);
            System.out.println("Certificado vlido?: " + valido);
        } catch (ParseException e) {
            System.err.println("Error al parsear la fecha: " + e.getMessage());
        }
    }
}
```

Output - JOVENES360 (run)

```
run:
Josue postulo a Desarrollador Java
Certificado subido para Josue
Certificado vlido?: true
BUILD SUCCESSFUL (total time: 0 seconds)
```

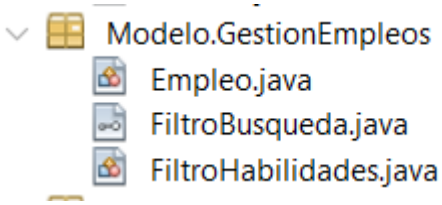
2. OCP - Principio de Abierto/Cerrado (Open/Closed Principle)

Este principio establece que las entidades de software deben estar abiertas para extensión, pero cerradas para modificación.

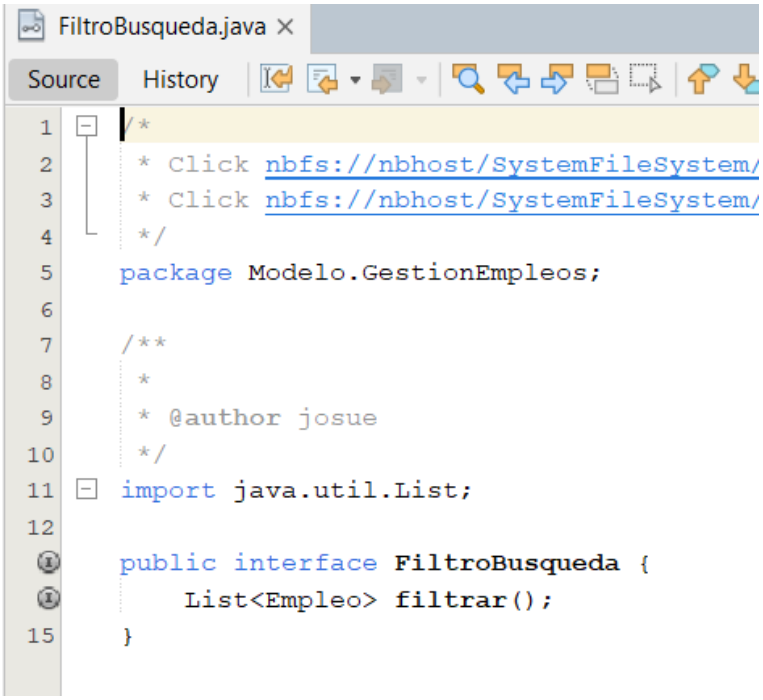
Aplicación:

- La interfaz FiltroBusqueda permite añadir nuevos tipos de filtros de búsqueda sin modificar los filtros existentes. Por ejemplo, FiltroHabilidades implementa la interfaz y se puede extender con más filtros sin alterar el código base.
- Esto promueve la extensibilidad sin comprometer la estabilidad del sistema.

CARPETAS:



CODIGO:



```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo.GestionEmpleos;

/**
 *
 * @author josue
 */
import java.util.List;
import java.util.stream.Collectors;

public class FiltroHabilidades implements FiltroBusqueda {
    private List<String> habilidadesRequeridas;
    private List<Empleo> empleos;

    public FiltroHabilidades(List<String> habilidadesRequeridas, List<Empleo> empleos) {
        this.habilidadesRequeridas = habilidadesRequeridas;
        this.empleos = empleos;
    }

    @Override
    public List<Empleo> filtrar() {
        // Ejemplo simple: filtra empleos que contengan al menos una habilidad requerida
        return empleos.stream()
            .filter(empleo -> empleo.getRequisitos().stream()
                .anyMatch(req -> habilidadesRequeridas.contains(req)))
            .collect(Collectors.toList());
    }

    // Getters y setters
    public List<String> getHabilidadesRequeridas() { return habilidadesRequeridas; }
    public void setHabilidadesRequeridas(List<String> habilidadesRequeridas) { this.habilidadesRequeridas = habilidadesRequeridas; }
}
```

EJECUCIÓN:

```java
import Modelo.Core.*;
import Modelo.GestionEmpleos.*;

import java.util.Arrays;
import java.util.List;

public class PruebaOCP {
    public static void main(String[] args) {
        // Crear candidato con habilidades
        Candidato candidato = new Candidato(
            "Josué",
            "josue@gmail.com",
            "1234",
            Arrays.asList("Java", "Spring Boot"),
            "cv.pdf"
        );

        // Crear lista de empleos
        Empleo emp1 = new Empleo("Backend Developer", "Spring Boot y APIs", Arrays.asList("Java", "Spring Boot"));
        Empleo emp2 = new Empleo("Diseñador UI", "Diseño con Figma", Arrays.asList("Figma", "UX"));
        Empleo emp3 = new Empleo("Fullstack", "React y Java", Arrays.asList("React", "Java"));

        List<Empleo> empleos = Arrays.asList(emp1, emp2, emp3);

        // Crear el filtro con los datos requeridos
        FiltroBusqueda filtro = new FiltroHabilidades(candidato.getSkills(), empleos);

        // Filtrar los empleos
        List<Empleo> empleosFiltrados = filtro.filtrar();

        // Mostrar resultados
        System.out.println("Empleos compatibles con habilidades del candidato:");
        for (Empleo empleo : empleosFiltrados) {
            System.out.println("- " + empleo.getTitulo());
        }
    }
}
```

```
Output - JOVENES360 (run)
run:
Empleos compatibles con habilidades del candidato:
- Backend Developer
- Fullstack
BUILD SUCCESSFUL (total time: 0 seconds)
```

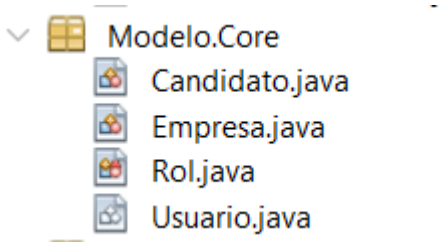
3. LSP - Principio de Sustitución de Liskov (Liskov Substitution Principle)

Indica que los objetos de una clase base deben poder ser reemplazados por objetos de sus clases derivadas sin alterar el correcto funcionamiento del programa.

Aplicación:

- La clase abstracta Usuario es la base común para Candidato y Empresa. Ambas clases derivadas pueden usarse donde se espera un Usuario sin que el comportamiento del sistema se vea afectado.
- Esto permite tratar uniformemente a candidatos y empresas en procesos que involucren usuarios.

CARPETAS:



CODIGO:

PruebaLCP.java | Usuario.java | Empresa.java | Candidato.java

Source | History

```
*/
package Modelo.Core;
/**
 *
 * @author josue
 */
public abstract class Usuario {
    protected String nombre;
    protected String email;
    protected String contraseña;
    protected Rol rol;

    public Usuario(String nombre, String email, String contraseña, Rol rol) {
        this.nombre = nombre;
        this.email = email;
        this.contraseña = contraseña;
        this.rol = rol;
    }

    public abstract void registrar();

    // Getters y setters
    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getContraseña() { return contraseña; }
    public void setContraseña(String contraseña) { this.contraseña = contraseña; }

    public Rol getRol() { return rol; }
    public void setRol(Rol rol) { this.rol = rol; }
}
```

PruebaLCP.java | Usuario.java | Empresa.java | Candidato.java

Source | History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo.Core;

/**
 *
 * @author josue
 */
public class Empresa extends Usuario {
    private String ruc;
    private String rubro;

    public Empresa(String nombre, String email, String contraseña, String ruc, String rubro) {
        super(nombre, email, contraseña, Rol.EMPRESA);
        this.ruc = ruc;
        this.rubro = rubro;
    }

    @Override
    public void registrar() {
        // Implementación específica para empresa
        System.out.println("Registrando empresa: " + nombre);
    }

    // Getters y setters
    public String getRuc() { return ruc; }
    public void setRuc(String ruc) { this.ruc = ruc; }

    public String getRubro() { return rubro; }
    public void setRubro(String rubro) { this.rubro = rubro; }
}
```

PruebaLCP.java | Usuario.java | Empresa.java | Candidato.java

Source | History

```java
 */
package Modelo.Core;

/**
 *
 * @author josue
 */
import java.util.List;

public class Candidato extends Usuario {
    private List<String> skills;
    private String cv;

    public Candidato(String nombre, String email, String contraseña, List<String> skills, String cv) {
        super(nombre, email, contraseña, Rol.CANDIDATO);
        this.skills = skills;
        this.cv = cv;
    }

    @Override
    public void registrar() {
        // Implementación específica para candidato
        System.out.println("Registrando candidato: " + nombre);
    }

    // Getters y setters
    public List<String> getSkills() { return skills; }
    public void setSkills(List<String> skills) { this.skills = skills; }

    public String getCv() { return cv; }
    public void setCv(String cv) { this.cv = cv; }
}
```

EJECUCIÓN:

PruebaLCP.java | Usuario.java | Empresa.java | Candidato.java

Source | History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package Controlador;

import Modelo.Core.Candidato;
import Modelo.Core.Empresa;
import Modelo.Core.Usuario;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author josue
 */
// Clase que demuestra el principio LSP
public class PruebaLCP {
    public static void main(String[] args) {
        // Crear lista de usuarios (puede contener tanto Candidatos como Empresas)
        List<Usuario> usuarios = new ArrayList<>();

        // Crear un candidato
        Usuario candidato = new Candidato("Juan Pérez", "juan@example.com", "pass123",
                                          List.of("Java", "Spring"), "Experto en desarrollo backend");

        // Crear una empresa
        Usuario empresa = new Empresa("Tech Solutions", "contacto@tech.com", "secure456",
                                      "123456789", "Desarrollo de software");

        // Añadir a la lista (sustitución de Liskov en acción)
        usuarios.add(candidato);
        usuarios.add(empresa);

        // Procesar usuarios (el código no necesita saber el tipo concreto)
        System.out.println("=== Procesando registro de usuarios ===");
        for (Usuario usuario : usuarios) {
```

PruebaLCP.java | Usuario.java | Empresa.java | Candidato.java

Source | History

```
34
35          // Procesar usuarios (el código no necesita saber el tipo concreto)
36          System.out.println("=== Procesando registro de usuarios ===");
37          for (Usuario usuario : usuarios) {
38              // LSP: Podemos llamar a métodos de Usuario sin saber el tipo concreto
39              usuario.registrar();
40              usuario.getNombre();
41
43              if (usuario instanceof Candidato) {
44                  System.out.println("CV del candidato: " + ((Candidato) usuario).getCv());
                    System.out.println("Skills: " + ((Candidato) usuario).getSkills());
46              } else if (usuario instanceof Empresa) {
47                  System.out.println("RUC de la empresa: " + ((Empresa) usuario).getRuc());
48                  System.out.println("Rubro: " + ((Empresa) usuario).getRubro());
49              }
50              System.out.println("--------------------");
51          }
52          // Demostración de que funciona con métodos que esperan Usuario
53          System.out.println("\n=== Enviando notificaciones ===");
54          enviarNotificacionBienvenida(candidato);
55          enviarNotificacionBienvenida(empresa);
56      }
57
58      // Método que acepta cualquier tipo de Usuario (base o derivado)
59      public static void enviarNotificacionBienvenida(Usuario usuario) {
60          System.out.println("Enviando notificacion de bienvenida a: " + usuario.getEmail());
61          // LSP: El comportamiento es correcto independientemente del tipo concreto
62      }
63  }
```

Output - JOVENES360 (run)

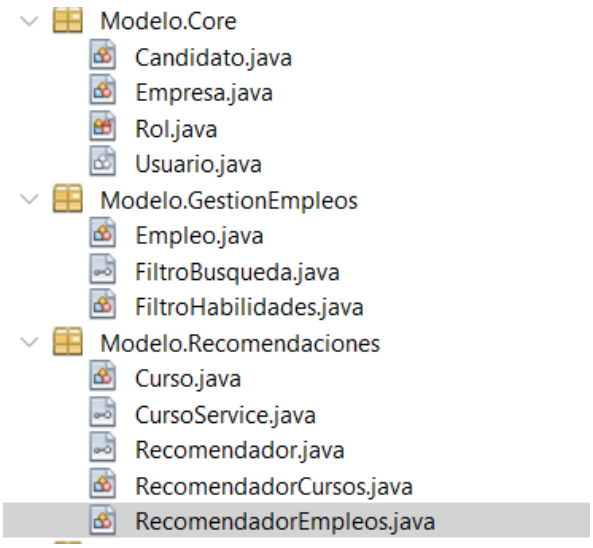
```
run:
=== Procesando registro de usuarios ===
Registrando candidato: Juan P▯rez
CV del candidato: Experto en desarrollo backend
Skills: [Java, Spring]
--------------------
Registrando empresa: Tech Solutions
RUC de la empresa: 123456789
Rubro: Desarrollo de software
--------------------

=== Enviando notificaciones ===
Enviando notificacion de bienvenida a: juan@example.com
Enviando notificacion de bienvenida a: contacto@tech.com
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. ISP - Principio de Segregación de Interfaces (Interface Segregation Principle)

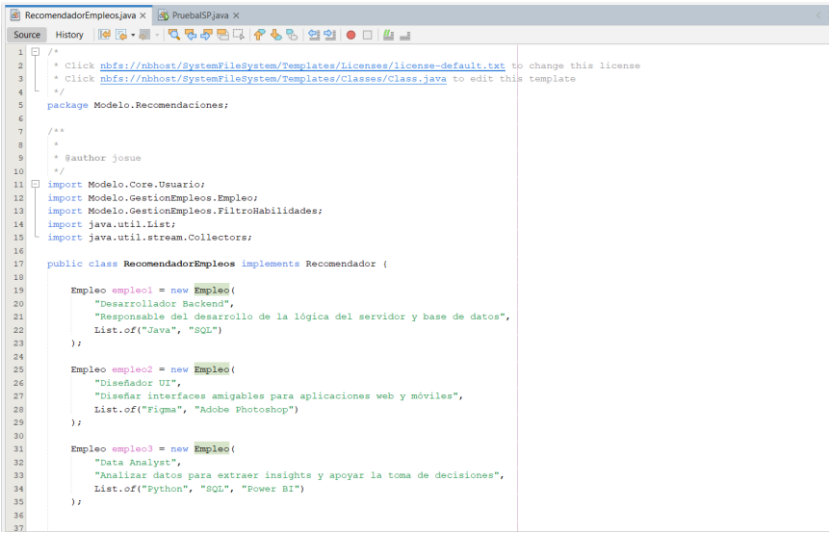
Establece que los clientes no deberían estar forzados a depender de interfaces que no usan; es preferible tener interfaces específicas y pequeñas. **Aplicación:**

- Se diseñan interfaces específicas para distintos roles, como Recomendador, con subinterfaces especializadas (RecomendadorEmpleos, RecomendadorCursos).
- Cada recomendador tiene sólo los métodos que necesita, evitando interfaces monolíticas y promoviendo un diseño desacoplado.

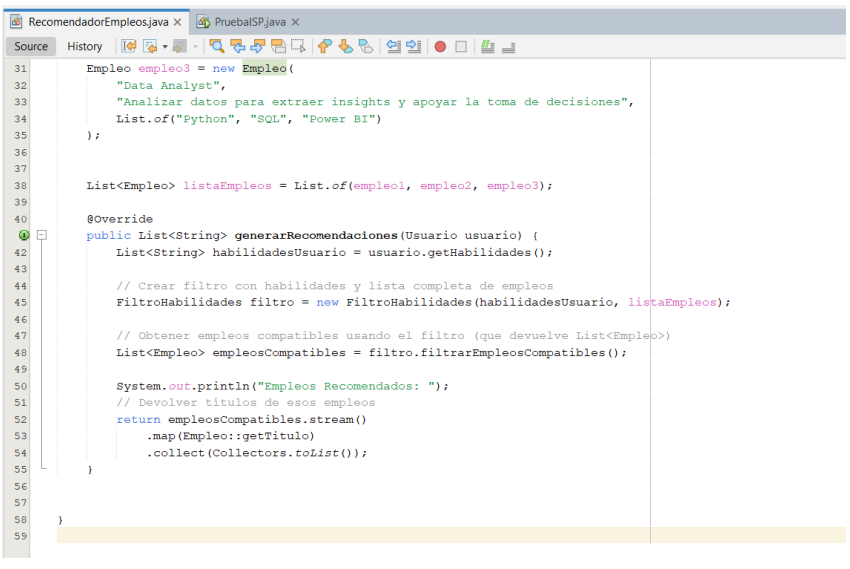
CARPETAS:



CODIGO:



```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo.Recomendaciones;

/**
 *
 * @author josue
 */
import Modelo.Core.Usuario;
import Modelo.GestionEmpleos.Empleo;
import Modelo.GestionEmpleos.FiltroHabilidades;
import java.util.List;
import java.util.stream.Collectors;

public class RecomendadorEmpleos implements Recomendador {

    Empleo empleo1 = new Empleo(
        "Desarrollador Backend",
        "Responsable del desarrollo de la lógica del servidor y base de datos",
        List.of("Java", "SQL")
    );

    Empleo empleo2 = new Empleo(
        "Diseñador UI",
        "Diseñar interfaces amigables para aplicaciones web y móviles",
        List.of("Figma", "Adobe Photoshop")
    );

    Empleo empleo3 = new Empleo(
        "Data Analyst",
        "Analizar datos para extraer insights y apoyar la toma de decisiones",
        List.of("Python", "SQL", "Power BI")
    );
```



```java
    Empleo empleo3 = new Empleo(
        "Data Analyst",
        "Analizar datos para extraer insights y apoyar la toma de decisiones",
        List.of("Python", "SQL", "Power BI")
    );


    List<Empleo> listaEmpleos = List.of(empleo1, empleo2, empleo3);

    @Override
    public List<String> generarRecomendaciones(Usuario usuario) {
        List<String> habilidadesUsuario = usuario.getHabilidades();

        // Crear filtro con habilidades y lista completa de empleos
        FiltroHabilidades filtro = new FiltroHabilidades(habilidadesUsuario, listaEmpleos);

        // Obtener empleos compatibles usando el filtro (que devuelve List<Empleo>)
        List<Empleo> empleosCompatibles = filtro.filtrarEmpleosCompatibles();

        System.out.println("Empleos Recomendados: ");
        // Devolver títulos de esos empleos
        return empleosCompatibles.stream()
            .map(Empleo::getTitulo)
            .collect(Collectors.toList());
    }


}
```

RecomendadorEmpleos.java | PruebaISP.java | RecomendadorCursos.java | **FiltroHabilidades.java**

Source | History

```java
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

public class FiltroHabilidades implements FiltroBusqueda {
    private final List<String> habilidadesRequeridas;
    private final List<Empleo> empleos;

    public FiltroHabilidades(List<String> habilidadesRequeridas, List<Empleo> empleos) {
        this.habilidadesRequeridas = habilidadesRequeridas;
        this.empleos = empleos;
    }

    // Devuelve empleos compatibles (tienen al menos una habilidad del usuario)
    public List<Empleo> filtrarEmpleosCompatibles() {
        return empleos.stream()
            .filter(empleo -> empleo.getRequisitos().stream()
                .anyMatch(habilidadesRequeridas::contains))
            .collect(Collectors.toList());
    }

    // Devuelve habilidades faltantes para empleos compatibles
    @Override
    public List<String> filtrar() {
        List<Empleo> empleosCompatibles = filtrarEmpleosCompatibles();

        Set<String> habilidadesFaltantes = new HashSet<>();
        for (Empleo empleo : empleosCompatibles) {
            for (String req : empleo.getRequisitos()) {
                if (!habilidadesRequeridas.contains(req)) {
                    habilidadesFaltantes.add(req);
                }
            }
        }
        return new ArrayList<>(habilidadesFaltantes);
    }

```

RecomendadorEmpleos.java | PruebaISP.java | RecomendadorCursos.java | **FiltroHabilidades.java** | Recomendador.java

Source | History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo.Recomendaciones;

/**
 *
 * @author josue
 */
import Modelo.Core.Usuario;
import java.util.List;

public interface Recomendador {
    List<String> generarRecomendaciones(Usuario usuario);
}

```

RecomendadorCursos.java | Curso.java | Empleo.java | PruebaISP.java

Source | History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo.Recomendaciones;

/**
 *
 * @author josue
 */
import Modelo.Core.Usuario;
import Modelo.GestionEmpleos.Empleo;
import Modelo.GestionEmpleos.FiltroHabilidades;
import java.util.List;
import java.util.stream.Collectors;

public class RecomendadorCursos implements Recomendador {

    Empleo empleo1 = new Empleo(
        "Desarrollador Backend",
        "Responsable del desarrollo de la lógica del servidor y base de datos",
        List.of("Java", "SQL")
    );

    Empleo empleo2 = new Empleo(
        "Diseñador UI",
        "Diseñar interfaces amigables para aplicaciones web y móviles",
        List.of("Figma", "Adobe Photoshop")
    );

    Empleo empleo3 = new Empleo(
        "Data Analyst",
        "Analizar datos para extraer insights y apoyar la toma de decisiones",
        List.of("Python", "SQL", "Power BI")
    );

```

RecomendadorCursos.java × Curso.java × Empleo.java × PruebaISP.java ×

Source History

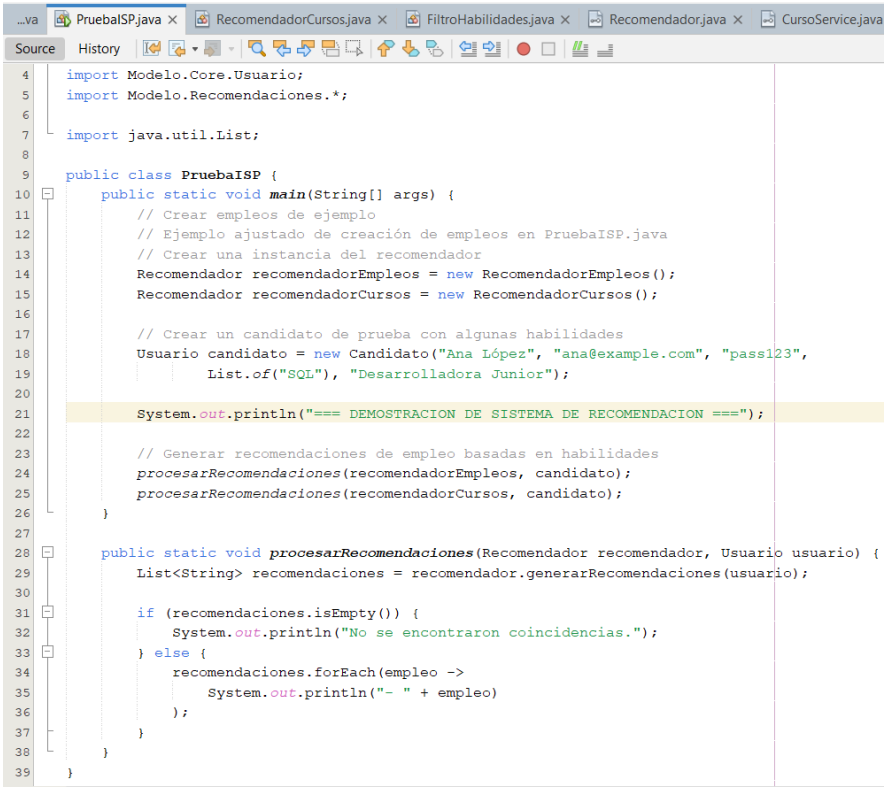
```
37
38          List<Empleo> listaEmpleos = List.of(empleo1, empleo2, empleo3);
39
40          Curso curso1 = new Curso(
41              "Fundamentos de Excel",
42              "Aprende a usar Excel para análisis de datos, funciones y tablas dinámicas.",
43              "LinkedIn Learning",
44              15,
45              "Principiante",
46              "https://www.linkedin.com/learning/excel-essential-training",
47              "Excel"
48          );
49
50          Curso curso2 = new Curso(
51              "Python para Todos",
52              "Curso completo para aprender a programar en Python desde cero.",
53              "Coursera",
54              25,
55              "Intermedio",
56              "https://www.coursera.org/specializations/python",
57              "Python"
58          );
59
60          Curso curso3 = new Curso(
61              "Programación en C++",
62              "Domina C++ desde lo básico hasta estructuras de datos avanzadas.",
63              "Udemy",
64              35,
65              "Avanzado",
66              "https://www.udemy.com/course/beginning-c-plus-plus-programming/",
67              "C++"
68          );
69
70
71          List<Curso> listaCursos = List.of(curso1, curso2, curso3);
72
73
```

RecomendadorCursos.java × Curso.java × Empleo.java × PruebaISP.java ×

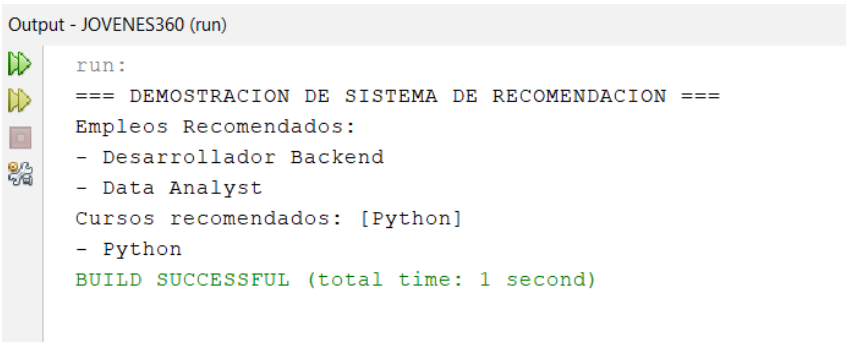
Source History

```
67              "C++"
68          );
69
70
71          List<Curso> listaCursos = List.of(curso1, curso2, curso3);
72
73
74          @Override
75          public List<String> generarRecomendaciones(Usuario usuario) {
76              List<String> habilidadesUsuario = usuario.getHabilidades();
77
78              // Obtener habilidades faltantes mediante el filtro
79              FiltroHabilidades filtro = new FiltroHabilidades(habilidadesUsuario, listaEmpleos);
80              List<String> habilidadesFaltantes = filtro.filtrar();
81
82              // Lista de cursos disponibles
83
84              // Filtrar habilidades faltantes que tengan un curso asociado
85              List<String> habilidadesConCurso = habilidadesFaltantes.stream()
86                  .filter(habilidad -> listaCursos.stream()
87                      .anyMatch(curso -> curso.getHabilidad().equalsIgnoreCase(habilidad)))
88                  .collect(Collectors.toList());
89
90              System.out.println("Cursos recomendados: " + habilidadesConCurso);
91
92              return habilidadesConCurso;
93          }
94
95
96
97
98      }
99
```

EJECUCIÓN:



```java
import Modelo.Core.Usuario;
import Modelo.Recomendaciones.*;

import java.util.List;

public class PruebaISP {
    public static void main(String[] args) {
        // Crear empleos de ejemplo
        // Ejemplo ajustado de creación de empleos en PruebaISP.java
        // Crear una instancia del recomendador
        Recomendador recomendadorEmpleos = new RecomendadorEmpleos();
        Recomendador recomendadorCursos = new RecomendadorCursos();

        // Crear un candidato de prueba con algunas habilidades
        Usuario candidato = new Candidato("Ana López", "ana@example.com", "pass123",
                List.of("SQL"), "Desarrolladora Junior");

        System.out.println("=== DEMOSTRACION DE SISTEMA DE RECOMENDACION ===");

        // Generar recomendaciones de empleo basadas en habilidades
        procesarRecomendaciones(recomendadorEmpleos, candidato);
        procesarRecomendaciones(recomendadorCursos, candidato);
    }

    public static void procesarRecomendaciones(Recomendador recomendador, Usuario usuario) {
        List<String> recomendaciones = recomendador.generarRecomendaciones(usuario);

        if (recomendaciones.isEmpty()) {
            System.out.println("No se encontraron coincidencias.");
        } else {
            recomendaciones.forEach(empleo ->
                System.out.println("- " + empleo)
            );
        }
    }
}
```



```
Output - JOVENES360 (run)
run:
=== DEMOSTRACION DE SISTEMA DE RECOMENDACION ===
Empleos Recomendados:
- Desarrollador Backend
- Data Analyst
Cursos recomendados: [Python]
- Python
BUILD SUCCESSFUL (total time: 1 second)
```

5. DIP - Principio de Inversión de Dependencias (Dependency Inversion Principle)

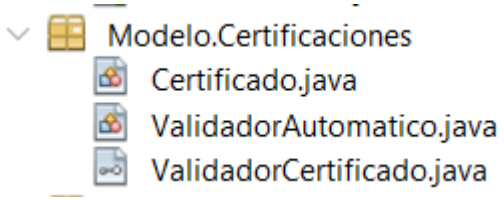
Propone que los módulos de alto nivel no deben depender de módulos de bajo nivel, sino de abstracciones. Además, las abstracciones no deben depender de detalles, sino al revés.

Aplicación:

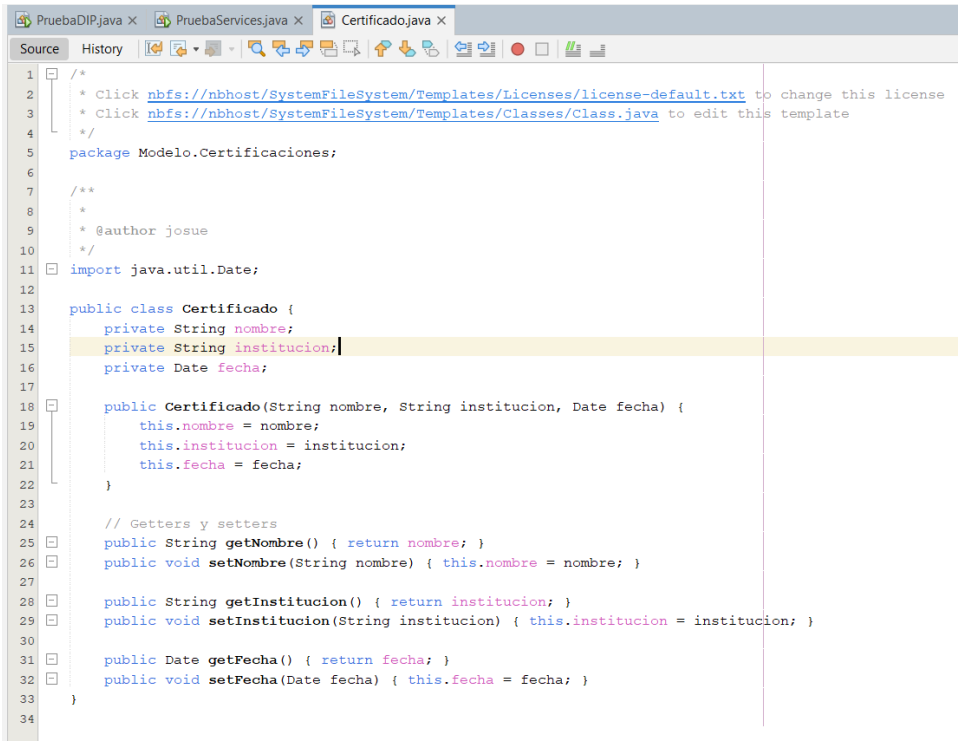
- Por ejemplo, CertificadoService depende de la interfaz ValidadorCertificado y no de una implementación concreta.

- Esto permite cambiar el validador automático por otra implementación sin afectar al servicio que lo usa, facilitando la mantenibilidad y prueba del código.

CARPETAS:



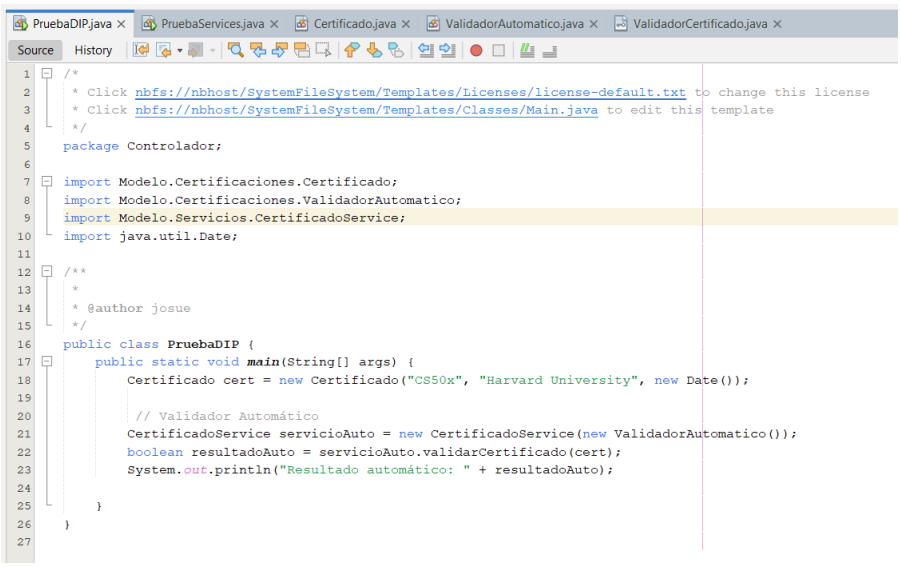
CODIGO:

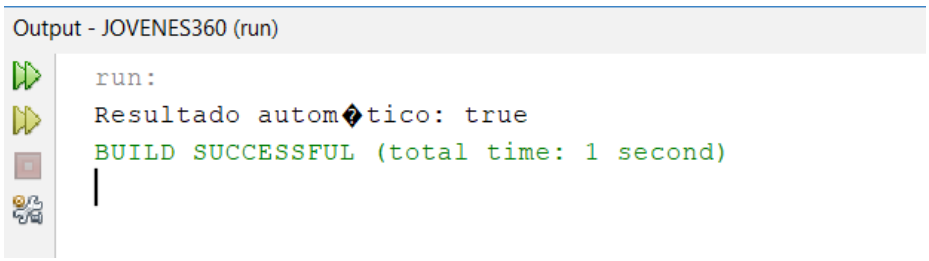


```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo.Certificaciones;

/**
 *
 * @author josue
 */
import java.util.Date;

public class Certificado {
    private String nombre;
    private String institucion;
    private Date fecha;

    public Certificado(String nombre, String institucion, Date fecha) {
        this.nombre = nombre;
        this.institucion = institucion;
        this.fecha = fecha;
    }

    // Getters y setters
    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public String getInstitucion() { return institucion; }
    public void setInstitucion(String institucion) { this.institucion = institucion; }

    public Date getFecha() { return fecha; }
    public void setFecha(Date fecha) { this.fecha = fecha; }
}
```

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this lic
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo.Certificaciones;

/**
 *
 * @author josue
 */
public class ValidadorAutomatico implements ValidadorCertificado {

    @Override
    public boolean validar(Certificado cert) {
        // Validación automática, ejemplo trivial
        return cert != null && cert.getNombre() != null && !cert.getNombre().isEmpty();
    }
}
```

EJECUCIÓN:





DESCRIPCIÓN DE ARQUITECTURA DE 4 CAPAS Y USO DE PATRONES DE DISEÑO

El proyecto JOVENES360 sigue una arquitectura en 4 capas (Modelo-Vista-Controlador-Base de Datos) para garantizar modularidad, escalabilidad y mantenibilidad. Cada capa se integra con patrones de diseño creacionales y estructurales, aprovechando los scripts SQL proporcionados para la persistencia de datos.

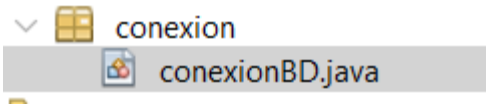
Relación entre Capas y Patrones

a) Modelo (Dominio)

- **Responsabilidad:** Representa las entidades del negocio (Usuario, Candidato, Empresa, Empleo, etc.) y su lógica.
- **Patrones aplicados:**
    - Singleton: Para acceder a servicios compartidos (ej: conexionBD.java).
    - Prototype: Para clonar objetos como plantillas de perfiles o CVs.
    - Factory Method: Para crear instancias de Usuario (Candidato/Empresa) sin acoplar el código.

1. Patrón Singleton en la clase conexionBD

- Una única instancia global: Todas las operaciones de BD usan la misma conexión.
- Reutilización: Evita crear conexiones redundantes.
- Control centralizado: La apertura/cierre se gestiona en un solo punto.



Vista1.java × Empleo.java × conexionBD.java ×

Source History

```java
package conexion;

import java.sql.Connection;
import java.sql.DriverManager;

public class conexionBD {
    private static conexionBD instancia;
    private Connection conexion;
    private String driver = "com.mysql.cj.jdbc.Driver";
    private final String url = "jdbc:mysql://localhost:3306/skill_Bridge";
    private final String usuario = "sa";
    private final String contrasena = "1234";


    private conexionBD(){
        try{
            Class.forName(driver);
            System.out.print("Conexion exitosa a mysql");
            conexion = DriverManager.getConnection(url, usuario, contrasena);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    public static conexionBD getInstancia() {
        if (instancia == null){
            instancia = new conexionBD();
        }
        return instancia;
    }

    public Connection getConnection(){
        return conexion;
    }
}
```

```sql
DELIMITER //
CREATE PROCEDURE sp_RegistrarUsuario(
    IN p_nombre VARCHAR(70),
    IN p_email VARCHAR(200),
    IN p_contrasena VARCHAR(300)
BEGIN
    INSERT INTO usuario (nombre, email, contrasena)
    VALUES (p_nombre, p_email, p_contrasena);
END //
DELIMITER ;
```

2. Patrón Prototype

Objetivo: Clonar objetos existentes para evitar la creación costosa desde cero.

Aplicación en JOVENES360:

- **Escenario:**
    - Los usuarios (candidatos) pueden crear múltiples versiones de su CV basadas en una plantilla inicial.
    - Las empresas pueden duplicar ofertas de empleo con ajustes mínimos (ej: mismo puesto pero diferente modalidad).

Beneficios:

- Eficiencia: Evita reprocesar datos estáticos (ej: información básica del usuario).

- Consistencia: Mantiene la estructura original mientras permite personalización.
- Cumple con ODS 8: Agiliza la inserción laboral al reducir tiempo en preparar documentos.

```java
public abstract class PerfilPrototype implements Cloneable {
    @Override
    public PerfilPrototype clone() throws CloneNotSupportedException {
        return (PerfilPrototype) super.clone();
    }
}

// Implementación para CV
public class CV extends PerfilPrototype {
    private String habilidades;
    private String experiencia;

    // Getters y setters
}

// Uso en el sistema
CV cvOriginal = new CV();
cvOriginal.setHabilidades("Java, Spring");
cvOriginal.setExperiencia("2 años en desarrollo");

CV cvClonado = cvOriginal.clone();
cvClonado.setExperiencia("3 años"); // Personalización rápida
```

3. Patrón Factory Method

Objetivo: Delegar la creación de objetos a subclases, evitando acoplamiento.

Aplicación en JOVENES360:

- **Escenario:**
    - Registrar diferentes tipos de usuarios (Candidato o Empresa) con lógicas distintas.
    - Crear empleos o cursos con variantes (ej: modalidad presencial/remota).

Beneficios:

- Flexibilidad: Permite añadir nuevos tipos de usuarios sin modificar código existente (ej: futuro rol Admin).
- Responsabilidad Única (SOLID): La lógica de creación está encapsulada en fábricas especializadas.
- Alineación con ODS 8: Facilita la inclusión de diversos perfiles laborales.

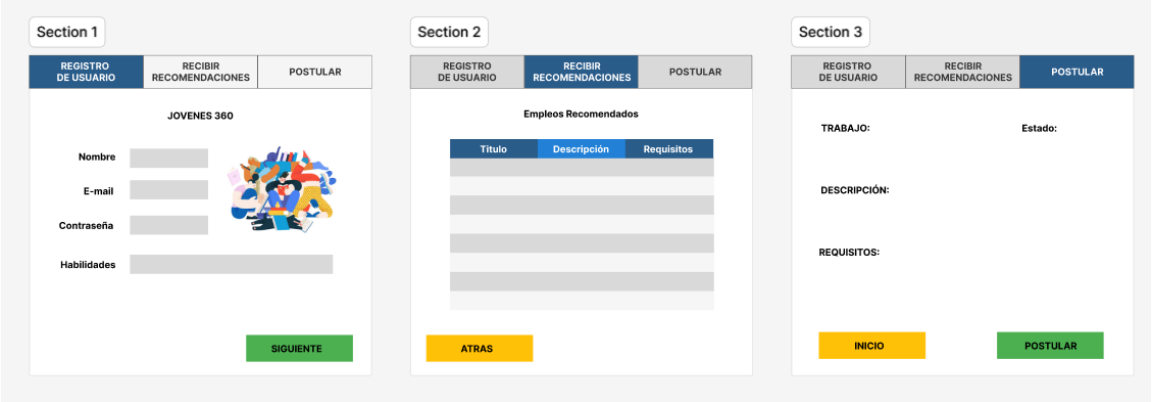
```java
// Interfaz Factory
public interface UsuarioFactory {
    Usuario crearUsuario(String tipo);
}

// Implementación concreta
public class UsuarioFactoryImpl implements UsuarioFactory {
    @Override
    public Usuario crearUsuario(String tipo) {
        return switch (tipo.toUpperCase()) {
            case "CANDIDATO" -> new Candidato();
            case "EMPRESA" -> new Empresa();
            default -> throw new IllegalArgumentException("Tipo no válido");
        };
    }
}

// Uso en el controlador
UsuarioFactory factory = new UsuarioFactoryImpl();
Usuario nuevoUsuario = factory.crearUsuario("CANDIDATO");
```

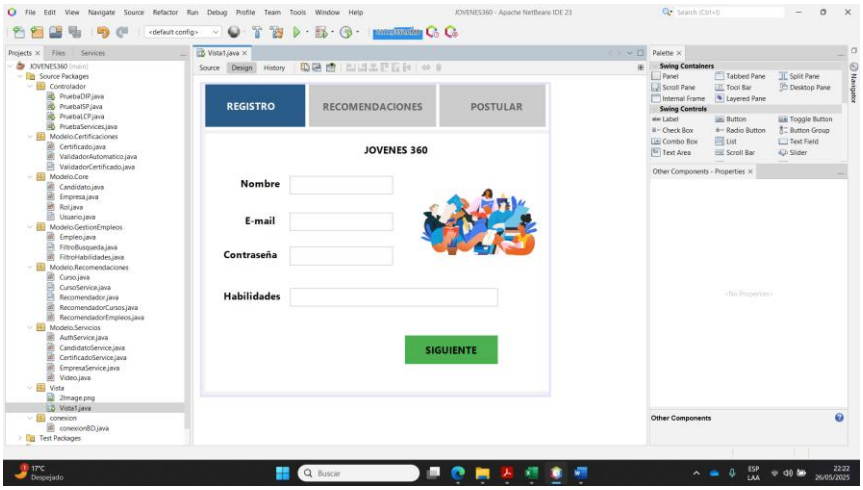
b) Controlador (Aplicación)

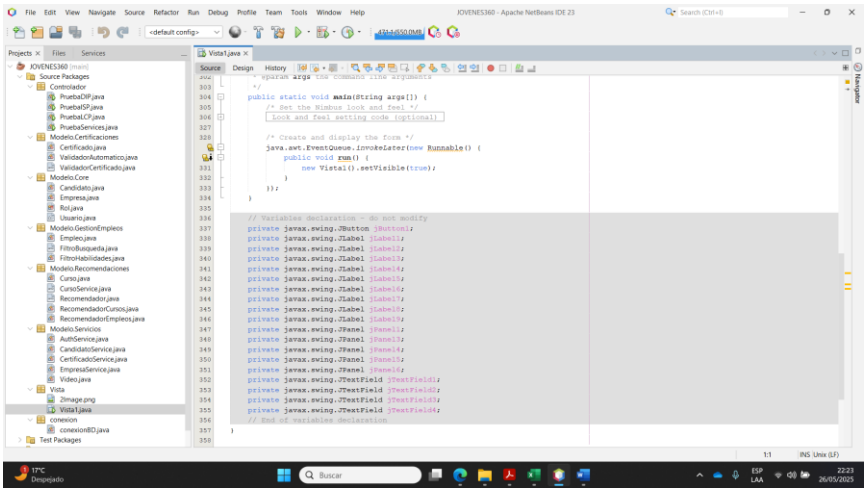
- **Responsabilidad:** Gestiona el flujo de datos entre la Vista y el Modelo.
- **Patrones aplicados:**
    - Facade: Simplifica operaciones complejas (ej: búsqueda de empleos con múltiples filtros).
    - Dependency Injection: Para desacoplar componentes (ej: inyectar UsuarioFactory).

c) Vista (Presentación)

- **Responsabilidad:** Interfaz de usuario en JFrame con Java.
- **Integración:**
    - El Controlador recibe solicitudes de la Vista y las procesa usando el Modelo.



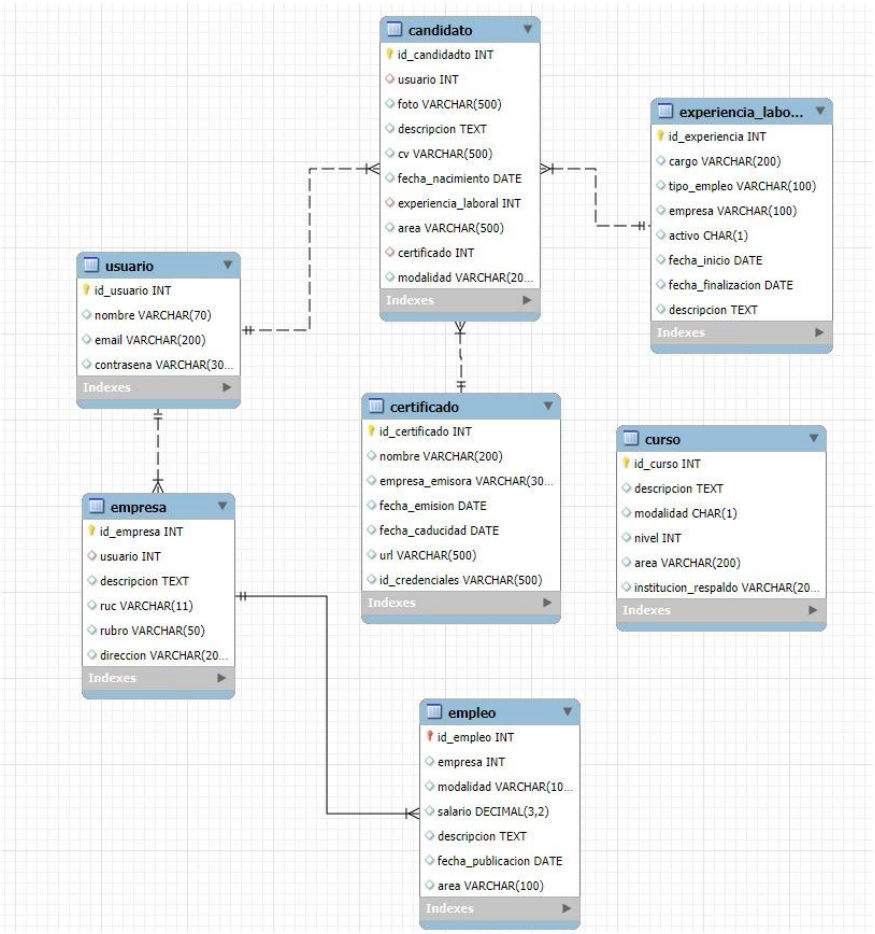


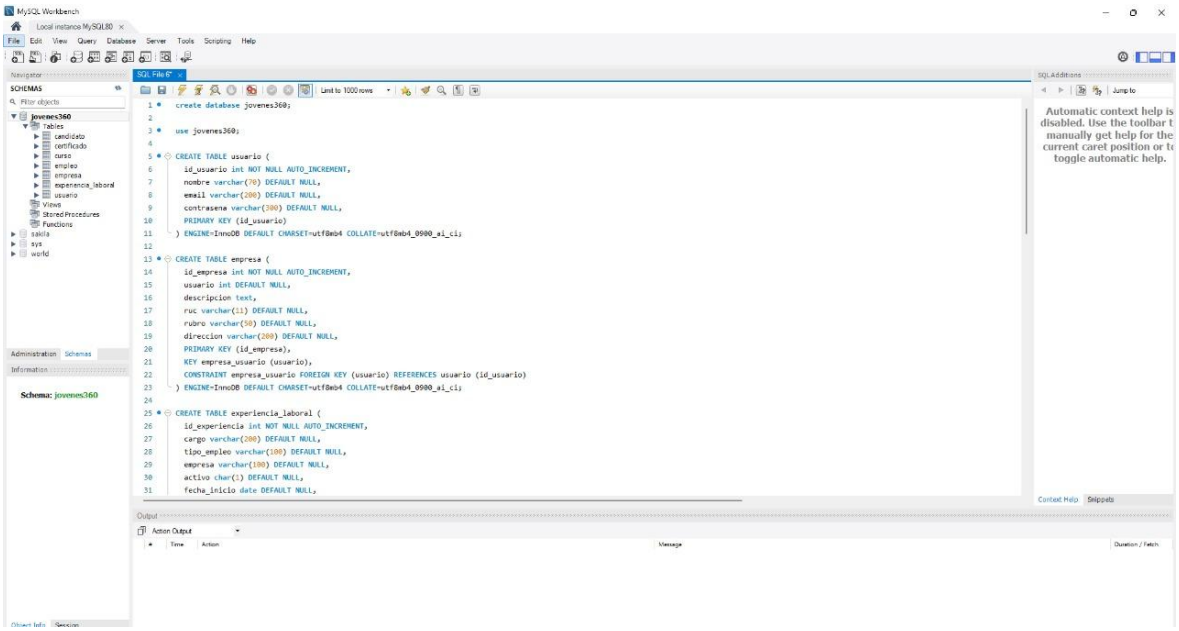


d) Base de Datos (Persistencia)

- **Responsabilidad:** Almacenar y recuperar datos usando MySQL.
- **Scripts SQL:**
    - Las tablas (usuario, candidato, empresa, etc.) reflejan las entidades del Modelo.

- Stored Procedures: Para operaciones complejas (ej: filtrar empleos).




MySQL Workbench
create database jovenes360;
use jovenes360;
CREATE TABLE usuario (
id_usuario int NOT NULL AUTO_INCREMENT,
nombre varchar(70) DEFAULT NULL,
email varchar(200) DEFAULT NULL,
contrasena varchar(200) DEFAULT NULL,
PRIMARY KEY (id_usuario)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
CREATE TABLE empresa (
id_empresa int NOT NULL AUTO_INCREMENT,
usuario int DEFAULT NULL,
descripcion text,
ruc varchar(11) DEFAULT NULL,
rubro varchar(50) DEFAULT NULL,
direccion varchar(200) DEFAULT NULL,
PRIMARY KEY (id_empresa),
KEY empresa_usuario (usuario),
CONSTRAINT empresa_usuario FOREIGN KEY (usuario) REFERENCES usuario (id_usuario)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
CREATE TABLE experiencia_laboral (
id_experiencia int NOT NULL AUTO_INCREMENT,
cargo varchar(200) DEFAULT NULL,
tipo_empleo varchar(100) DEFAULT NULL,
empresa varchar(100) DEFAULT NULL,
activo char(1) DEFAULT NULL,
fecha_inicio date DEFAULT NULL,

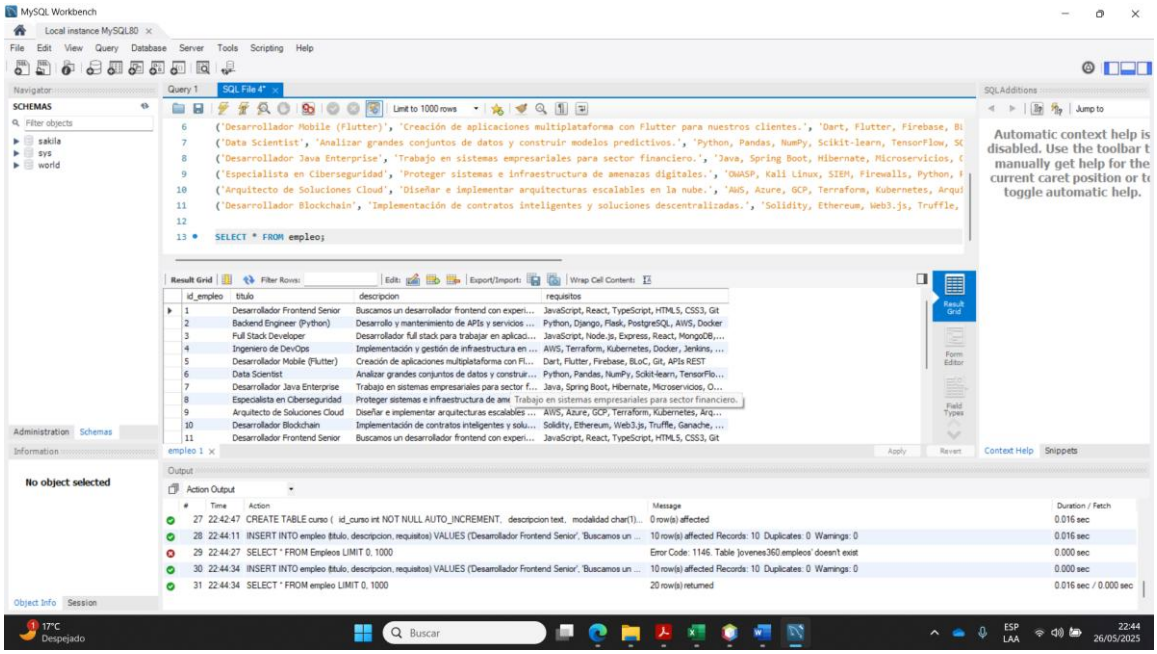
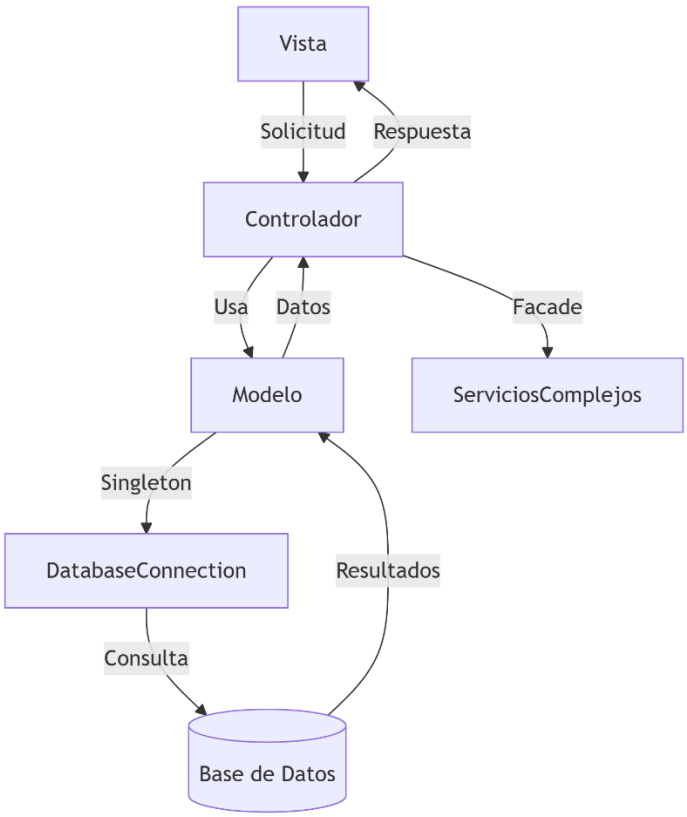

MySQL Workbench
('Desarrollador Mobile (Flutter)', 'Creación de aplicaciones multiplataforma con Flutter para nuestros clientes.', 'Dart, Flutter, Firebase,
('Data Scientist', 'Analizar grandes conjuntos de datos y construir modelos predictivos.', 'Python, Pandas, NumPy, Scikit-learn, TensorFlow,
('Desarrollador Java Enterprise', 'Trabajo en sistemas empresariales para sector financiero.', 'Java, Spring Boot, Hibernate, Microservicios,
('Especialista en Ciberseguridad', 'Proteger sistemas e infraestructura de amenazas digitales.', 'OWASP, Kali Linux, SIEM, Firewalls, Python,
('Arquitecto de Soluciones Cloud', 'Diseñar e implementar arquitecturas escalables en la nube.', 'AWS, Azure, GCP, Terraform, Kubernetes, Arqui
('Desarrollador Blockchain', 'Implementación de contratos inteligentes y soluciones descentralizadas.', 'Solidity, Ethereum, Web3.js, Truffle,
SELECT * FROM empleo;
No object selected

Diagrama de Flujo entre capas de arquitectura:



Justificación de la Paleta de Colores para JOVENES 360

Objetivo: Crear una interfaz juvenil, profesional y alineada con el **ODS 8 (Trabajo Decente y Crecimiento Económico)**, combinando accesibilidad, psicología del color y enfoque en usuarios jóvenes.

---

1. Azul Profesional (#2A5C8A)

- **Significado:**
  - **Confianza y estabilidad:** El azul es asociado a empresas y entornos laborales serios (ej: LinkedIn).
  - **Enfoque juvenil:** Tonos medios como el #2A5C8A evitan la rigidez del azul corporativo tradicional.
- **Uso en la plataforma:**
  - Barra superior, botones de acción principal (*Registro*, *Postular*).
  - Simboliza la **formalidad del empleo decente** (ODS 8).

2. Verde Crecimiento (#4CAF50)

- **Significado:**
  - **Crecimiento y sostenibilidad:** Refleja el progreso económico y la formación profesional (metas del ODS 8).

    - **Energía positiva:** Ideal para acciones de éxito (*Guardar cambios*, *Postulación exitosa*).
- **Uso en la plataforma:**
    - Botones de confirmación (*Siguiente*, *Aplicar a empleo*).
    - Secciones de capacitación y logros.

3. Amarillo Energía (#FFC107)

- **Significado:**
    - **Optimismo y creatividad:** Atrae a jóvenes y resalta oportunidades destacadas.
    - **Atención visual:** Guía al usuario a elementos clave (*Recomendaciones*, *Alertas*).
- **Uso en la plataforma:**
    - Badges de empleos "Urgentes" o "Nuevos".
    - Iconos interactivos (ej: flechas, estrellas de valoración).

4. Gris Neutro (#F5F5F5 para fondos, #757575 para texto)

- **Significado:**
    - **Neutralidad y equilibrio:** Permite destacar los colores primarios sin saturar la vista.
    - **Legibilidad:** Cumple con estándares WCAG para contraste accesible.
- **Uso en la plataforma:**
    - Fondos de formularios (*Crear perfil*).
    - Texto secundario y bordes.

5. Rojo Acción (#E53935)

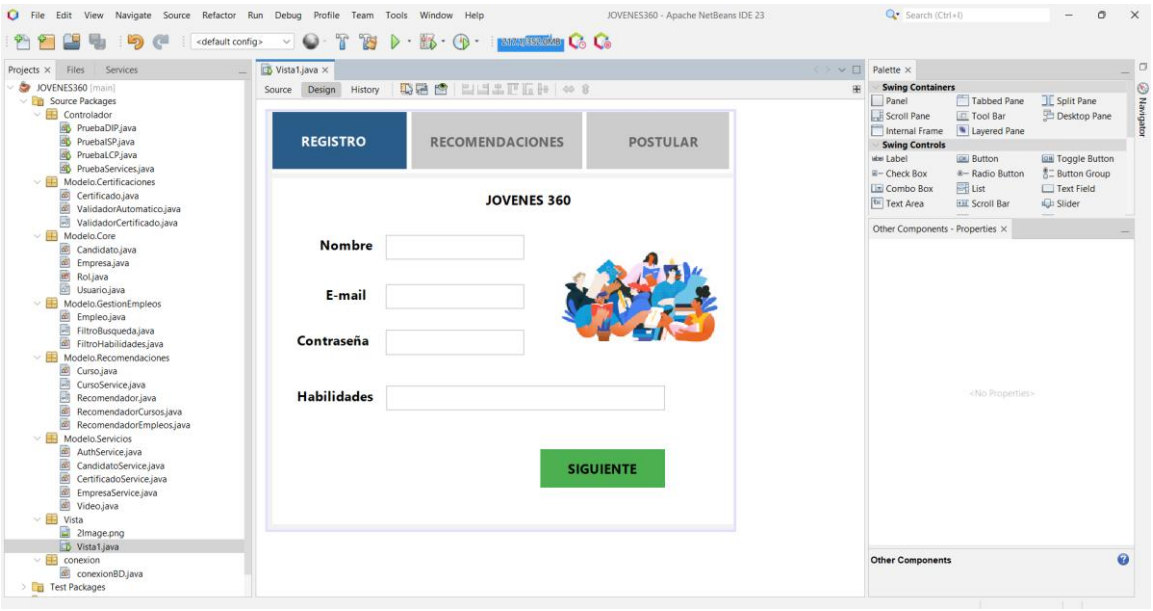
- **Significado:**
    - **Urgencia y atención:** Indica errores o acciones críticas (*Campos obligatorios*, *Advertencias*).
- **Uso en la plataforma:**
    - Mensajes de validación en formularios.
    - Botones de eliminación o cancelación.

Relación con el Público Juvenil y el ODS 8

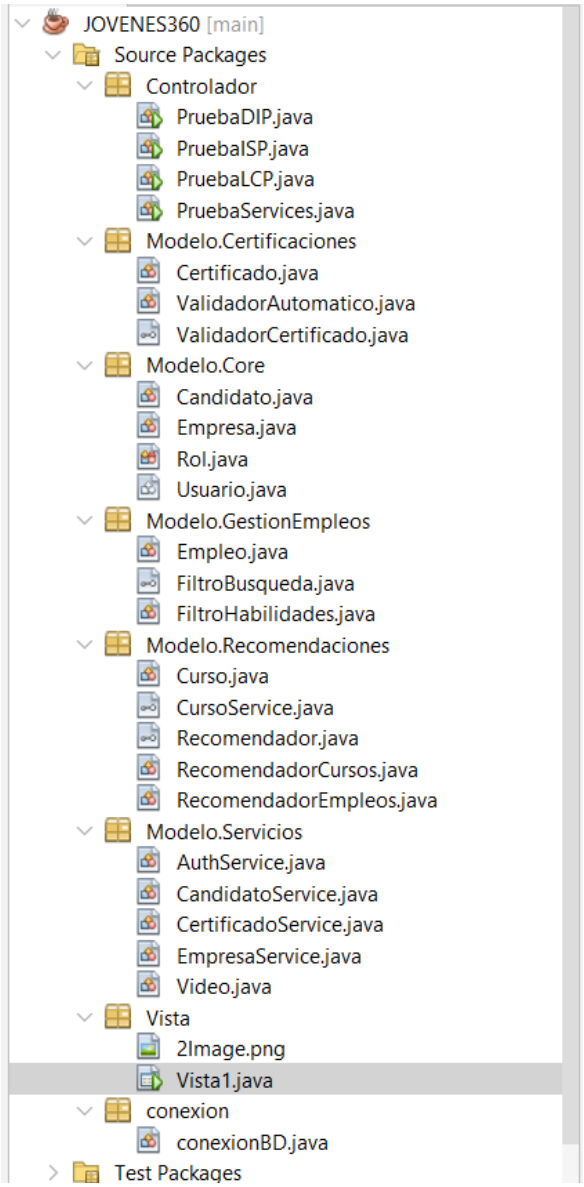
- **Juventud:** La combinación de **azul moderado + amarillo** equilibra seriedad y dinamismo, atrayendo a jóvenes sin perder profesionalismo.
- **Trabajo Decente (ODS 8):**
    - El **verde** simboliza crecimiento económico sostenible.
    - El **azul** refuerza la confianza en las oportunidades laborales.

Ejemplo de Aplicación

Sección	Color Principal	Justificación
Registro de Usuario	#2A5C8A (azul)	Transmite seguridad al ingresar datos personales.
Empleos Recomendados	#FFC107 (amarillo)	Destaca oportunidades prioritarias.
Botón "Postular"	#4CAF50 (verde)	Refuerza la acción positiva hacia el empleo.
Mensajes de Error	#E53935 (rojo)	Alerta sobre problemas en el proceso.



Estructura de Carpetas en el IDE:



CAPITULO 3: Conclusiones

- **Lecciones aprendidas por cada capa de la arquitectura**

Capa de Presentación (Vista)

1. Importancia del diseño UX: Aprendimos que una interfaz intuitiva requiere múltiples iteraciones
2. Manejo de eventos: Comprendimos la complejidad de coordinar acciones entre componentes Swing
3. Validación de datos: Validar inputs directamente en la vista mejora la experiencia de usuario

Capa de Lógica (Modelo)

1. Patrones de diseño: Implementar el patrón Recomendador nos enseñó a crear interfaces flexibles
2. Algoritmos de matching: Desarrollar el sistema de coincidencias de habilidades nos mostró la importancia de los algoritmos eficientes
3. Abstracción: Aprendimos a separar claramente la lógica de negocio de otras capas

Capa de Datos

1. Estructura de datos: Seleccionar las colecciones adecuadas (List vs Set) impacta en el rendimiento
2. Persistencia básica: Implementar el patrón Repository para manejar los datos de empleos
3. Normalización: La importancia de estructurar bien los datos desde el inicio

- **Lecciones aprendidas del equipo de trabajo**

Estudiante 1: Marco León Moreno

1. Coordinación: Aprendí a dividir tareas usando Git.
2. Comunicación: La importancia de documentar decisiones técnicas

Estudiante 2: Josué

1. Liderazgo técnico: Guié el diseño de la arquitectura MVC
2. Resolución de conflictos: Medié cuando hubo desacuerdos en implementaciones
3. Calidad de código: Insistí en implementar pruebas unitarias básicas

Estudiante 3: Boris

1. Documentación: Me enfoqué en mantener documentación actualizada
2. Integración: Aseguré que los módulos de cada uno funcionaran juntos
3. UI/UX: Propuse mejoras basadas en feedback de usuarios de prueba

Conclusiones generales del equipo

1. **Trabajo en equipo:**
    - La planificación semanal fue clave
    - Usamos Discord para comunicación constante
    - Dividimos tareas según fortalezas pero también aprendimos nuevas habilidades

2. **Gestión del proyecto:**
    - Aprendimos a usar Git de forma colaborativa

3. **Desarrollo técnico:**
    - Comprendimos la importancia de la separación de responsabilidades
    - Aprendimos a hacer debugging colaborativo
    - Implementamos código siguiendo principios SOLID

4. **Áreas de mejora:**
    - Necesitamos mejorar en debugging.
    - Gestionar mejor los tiempos para evitar trabajo de última hora

Este proyecto nos enseñó que el desarrollo de software va más allá de escribir código - incluye planificación, comunicación constante y capacidad de adaptación cuando surgen problemas inesperados. La experiencia nos preparó mejor para proyectos futuros tanto académicos como profesionales.

Demo del software con pruebas detallado (video) Simulación de caso práctico de utilidad: https://youtu.be/PBMHn2Ufv6g
Github: Josue049/ProyectoJovenes360

Referencias Bibliográficas (Formato APA 8ª edición)

- Organización de las Naciones Unidas. (s.f.). *Crecimiento económico*. https://www.un.org/sustainabledevelopment/es/economic-growth/
- Pacto Mundial. (s.f.). *ODS 8: Trabajo decente y crecimiento económico*. https://www.pactomundial.org/ods/8-trabajo-decente-y-crecimiento-economico/
- Martin, R. C. (2009). *Código limpio: Manual de estilo para el desarrollo ágil de software* (1.ª ed.). Pearson Educación. https://archive.org/details/codigo-limpiorobert-cecil-martin
- Color Assignment. (2023). *Estudio psicológico del color azul en interfaces digitales* [Informe no publicado].
- Organización de las Naciones Unidas. (2023). *Guía de branding para los Objetivos de Desarrollo Sostenible* [Documento interno].
- WebAIM. (2023). *WebAIM Contrast Checker* [Herramienta en línea]. https://webaim.org/resources/contrastchecker/