Trabajo Práctico Recuperatorio 2025
Cátedra: Paradigmas y Lenguajes de Programación II
Tema: Sistema de Gestión de Trámites de Empleo con POO, SOLID y Patrones de Diseño

---

1. Enunciado

Una institución de intermediación laboral desea implementar una “bolsa de empleo inteligente” que permita gestionar, en forma unificada, distintos tipos de postulantes y ofertas de trabajo.

El sistema deberá contemplar cuatro tipos de actores principales:

1. **Personas que buscan trabajo en general** (empleados potenciales).

2. **Profesionales** (abogados, contadores, ingenieros, etc.) que requieren un tratamiento diferenciado por su formación académica.

3. **Emprendedores** que desean registrar y difundir sus emprendimientos o servicios propios.

4. **Empresas** que se registran para publicar ofertas laborales y buscar talento.

La institución solicita un sistema web desarrollado con **PHP (php puro)**, utilizando **Bootstrap** para la interfaz (diseño moderno y minimalista) y **MySQL** como motor de base de datos.

2. Objetivo del trabajo práctico

Diseñar e implementar un sistema web modular y extensible que:

- Gestione el ciclo básico de vida de los distintos tipos de postulantes y empresas.

- Permita la interacción entre empresas y postulantes (búsqueda, filtrado y visualización de perfiles y ofertas).

- Aplique **Programación Orientada a Objetos**, **principios SOLID** y **patrones de diseño** en la capa de lógica de negocio.

- Utilice un **modelo de datos relacional en MySQL** completo, coherente y normalizado para una bolsa de empleo (no se aceptan diseños “minimalistas” de 2 o 3 tablas).

3. Requerimientos funcionales (mínimos)

3.1. Gestión de tipos de usuarios/postulantes

El sistema debe disponer de un **menú inicial** donde el usuario elija con qué tipo de perfil desea trabajar:

- Persona que busca empleo
- Profesional
- Emprendedor
- Empresa

Cada tipo debe tener su propio flujo de registro y edición de perfil, aunque compartan información en común.

3.2. Personas que buscan trabajo (empleados en general)

- Registro de usuario con datos personales completos:
  nombre, apellido, DNI, fecha de nacimiento, contacto (email, teléfono), domicilio, localidad, provincia, etc.
- Registro de **formación académica** y cursos.
- Registro de **experiencia laboral**.
- Carga de **documentación adjunta** (CV, certificados, otros archivos).
- Posibilidad de **marcar áreas de interés** (rubros, puestos deseados, rango salarial pretendido, tipo de jornada).

3.3. Profesionales

Además de lo anterior, deben poder registrar:

- Título profesional, matrícula, institución que otorga el título, año de graduación.
- Especialidades o áreas de práctica profesional.
- Honorarios de referencia (opcional).
- Documentación específica (certificados, matrícula escaneada, etc.).

Los profesionales deben poder:

- Ver listados y detalles de **empresas que buscan profesionales**.
- Marcar ofertas como favoritas y postularse.

3.4. Emprendedores

Los emprendedores deben poder:

- Registrarse como persona (datos personales básicos).
- Registrar uno o más **emprendimientos**, indicando como mínimo:
    - Nombre del emprendimiento.
    - Descripción detallada.
    - Rubro/sector.
    - Estado (idea, en marcha, en expansión).
    - Necesidades (financiamiento, socios, difusión, etc.).
    - Datos de contacto específicos del emprendimiento (sitio web, redes sociales, etc.).
- Cargar **archivos y documentación** (presentaciones, catálogos, fotos, PDFs, videos como enlaces, etc.) para difundir su propuesta.

3.5. Empresas

Las empresas deben poder:

- Registrarse con datos empresariales completos:
  razón social, CUIT, rubro, cantidad de empleados, domicilio fiscal, localidad, provincia, persona de contacto, teléfono, email institucional, sitio web, etc.
- Publicar y gestionar **ofertas laborales**, indicando:
    - Título del puesto.
    - Descripción del puesto.
    - Tipo de postulante buscado (empleado general, profesional específico, emprendedor para asociarse, etc.).
    - Requisitos (formación, experiencia, habilidades).

    - Tipo de contratación, jornada, rango salarial (opcional).
    - Fecha de publicación y estado (activa, pausada, cerrada).

Las empresas deberán:

- **Buscar y filtrar perfiles** de personas y profesionales según criterios (rubro, estudios, experiencia, ubicación, etc.).
- Ver detalle de los perfiles y acceder a la documentación adjunta relevante.
- Marcar perfiles como preseleccionados o descartados (modelo simple).

3.6. Interacción entre actores

- Personas y profesionales podrán:
    - Ver el listado de **empresas** registradas.
    - Ver el listado de **ofertas laborales activas**.
    - Postularse a una oferta y adjuntar documentación adicional si corresponde.
- Emprendedores:
    - Podrán hacer visible su emprendimiento para que empresas (o la institución) puedan ver y eventualmente contactarlos.

3.7. Gestión de documentos

Todos los tipos de postulantes (personas, profesionales, emprendedores) deben poder:

- Cargar documentación asociada a su perfil.
- Ver y gestionar (al menos listar y eliminar) dicha documentación.

Se requiere implementar el almacenamiento físico real de archivos en Base datos MYSQL que se brinda código más adelante.

---

4. Requerimientos técnicos y de diseño

- Lenguaje: **PHP (php puro)**.

- Framework de estilos: **Bootstrap** (última versión estable disponible en CDN).
- Arquitectura recomendada:
    - Separación de capas (presentación, lógica de negocio, acceso a datos).
    - No se permite mezclar toda la lógica en un solo archivo PHP gigante.
- Diseño de interfaz:
    - Estilo **moderno y minimalista**.
    - Menú claro para elegir tipo de postulante y navegar las principales funcionalidades.
    - Formularios responsivos y legibles.

---

5. Modelo de datos en MySQL (mínimo requerido)

Se deberá diseñar y crear la base de datos en **MySQL** con un conjunto de tablas completo y normalizado. Como mínimo, deberán contemplar:

- `usuarios` (credenciales de acceso, rol/tipo básico, estado).
- `personas` (datos personales básicos).
- `tipos_postulante` (empleado general, profesional, emprendedor, empresa).
- `postulantes` (vincula persona/empresa con usuario y tipo de postulante).
- `profesionales` (datos específicos de profesionales).
- `emprendimientos` (datos del emprendimiento).
- `empresas` (datos específicos de empresas).
- `ofertas_laborales` (publicadas por empresas).
- `postulaciones` (relación postulante–oferta).
- `formaciones_academicas` (niveles de estudio, títulos, etc.).

- `experiencias_laborales` (historial laboral).
- `documentos_adjuntos` (metadatos de archivos para cualquier postulante/emprendimiento).

6. Lineamientos de POO, SOLID y patrones de diseño

La implementación en PHP deberá:

1. Utilizar **clases y objetos** para modelar la lógica de negocio:
    - Clase abstracta o interfaz base `Postulante`.
    - Clases concretas: `PersonaPostulante`, `ProfesionalPostulante`, `EmprendedorPostulante`, `EmpresaPostulante`.
2. Aplicar **principios SOLID**:
    - Responsabilidad única (cada clase hace una cosa bien definida).
    - Abierto/cerrado (facilidad de extender tipos de postulantes sin modificar el corazón del sistema).
    - Inversión de dependencias (servicios y repositorios desacoplados).
3. Incluir al menos **dos patrones de diseño** relevantes, por ejemplo:
    - `Factory Method` o `Abstract Factory` para crear objetos `Postulante` según el tipo.
    - `Strategy` para distintas estrategias de búsqueda/filtrado de perfiles y ofertas.
    - `Repository`/`DAO` para acceso a datos.
    - Otros patrones bien justificados también serán aceptados.

El uso de patrones deberá estar documentado y claramente identificable en el código.

7. Entregables

1. **Modelo de base de datos**:
    - Script SQL de creación (`CREATE TABLE ...`) con todas las tablas, claves y relaciones.
    - Diagrama entidad–relación (puede ser exportado de una herramienta o dibujado con claridad).
2. **Código fuente en PHP**:
    - Estructura de carpetas organizada (por ejemplo: `app/`, `models/`, `views/`, `controllers/`, `config/`, etc.). Se brinda estructura mas adelante.
    - Clases, interfaces y patrones aplicados.
3. **Interfaz web**:
    - Pantalla inicial con selección de tipo de postulante/empresa.
    - Formularios de registro y edición.
    - Listados y vistas de detalle de perfiles y ofertas.
    - Ejemplo funcional de búsquedas básicas.
4. **Documento técnico breve** (máx. 5 páginas) donde se explique:
    - Decisiones de diseño de la base de datos.
    - Arquitectura de clases y aplicación de SOLID.
    - Patrones de diseño utilizados y su justificación.

Estructura de archivos y carpetas del proyecto.

La idea base:

- `public/` = punto de entrada (lo que ve el servidor web).
- `app/` = código de negocio (POO, SOLID, patrones).
- `storage/` = cosas persistentes (logs, uploads, etc.).
- `config/` = configuración.

---

1. Árbol de carpetas propuesto

```
/proyecto-bolsa-empleo/
├─ public/
│  ├─ index.php
│  ├─ .htaccess                (si usan Apache para routing amigable)
│  ├─ assets/
│  │  ├─ css/
│  │  │  ├─ bootstrap.min.css  (desde CDN o local)
│  │  │  └─ app.css            (estilos propios)
│  │  ├─ js/
│  │  │  ├─ bootstrap.bundle.min.js
│  │  │  └─ app.js
│  │  ├─ img/
│  │  │  └─ (imágenes del sitio)
```

```
|   |   └─ uploads/              (si exponen algunos archivos
públicos)

|   └─ favicon.ico



├─ app/

|   ├─ Core/

|   |   ├─ Autoloader.php        (autoload simple si no usan Composer)

|   |   ├─ Router.php            (enrutador básico tipo Front
Controller)

|   |   ├─ Controller.php        (clase base para controladores)

|   |   ├─ View.php              (helper para renderizar vistas)

|   |   └─ Database.php          (conexión PDO a MySQL)

|   |

|   ├─ Config/

|   |   ├─ config.php            (config general: entorno, debug,
etc.)

|   |   ├─ database.php          (host, dbname, user, pass)

|   |   └─ routes.php            (mapa de rutas → Controlador@acción)

|   |

|   ├─ Controllers/

|   |   ├─ HomeController.php            (página principal / landing)

|   |   ├─ AuthController.php            (login, logout, registro
básico)

|   |   ├─ PostulanteController.php      (alta/edición de postulantes)

|   |   ├─ ProfesionalController.php     (flujo específico de
profesionales)
```

```
|   |   ├─ EmprendedorController.php     (flujo específico de
emprendedores)

|   |   ├─ EmpresaController.php         (registro y gestión de
empresas)

|   |   ├─ OfertaController.php          (ABM de ofertas laborales)

|   |   ├─ PostulacionController.php     (postulaciones a ofertas)

|   |   ├─ EmprendimientoController.php  (ABM de emprendimientos)

|   |   └─ DocumentoController.php       (carga/gestión de documentos)

|   |

|   ├─ Models/

|   |   ├─ Entities/                     (modelo de dominio)

|   |   |   ├─ Usuario.php

|   |   |   ├─ Postulante.php            (abstracta o interfaz)

|   |   |   ├─ Persona.php

|   |   |   ├─ Empresa.php

|   |   |   ├─ Profesional.php

|   |   |   ├─ Emprendimiento.php

|   |   |   ├─ OfertaLaboral.php

|   |   |   ├─ Postulacion.php

|   |   |   ├─ FormacionAcademica.php

|   |   |   ├─ ExperienciaLaboral.php

|   |   |   └─ DocumentoAdjunto.php

|   |   |

|   |   ├─ Repositories/                 (patrón Repository / DAO)

|   |   |   ├─ UsuarioRepositoryInterface.php
```

```
|   |   |   ├─ PostulanteRepositoryInterface.php
|   |   |   ├─ EmpresaRepositoryInterface.php
|   |   |   ├─ OfertaRepositoryInterface.php
|   |   |   ├─ PostulacionRepositoryInterface.php
|   |   |   ├─ EmprendimientoRepositoryInterface.php
|   |   |   ├─ MySQLUsuarioRepository.php
|   |   |   ├─ MySQLPostulanteRepository.php
|   |   |   ├─ MySQLEmpresaRepository.php
|   |   |   ├─ MySQLOfertaRepository.php
|   |   |   ├─ MySQLPostulacionRepository.php
|   |   |   └─ MySQLEmprendimientoRepository.php
|   |   |
|   |   └─ ValueObjects/                (opcional, para ser puristas)
|   |       ├─ Email.php
|   |       ├─ Dni.php
|   |       └─ RangoSalarial.php
|   |
|   ├─ Services/                         (lógica de negocio y casos
de uso)
|   |   ├─ AuthService.php
|   |   ├─ PostulanteService.php
|   |   ├─ ProfesionalService.php
|   |   ├─ EmprendedorService.php
|   |   ├─ EmpresaService.php
|   |   ├─ OfertaService.php
```

```
|   |   ├─ PostulacionService.php
|   |   ├─ EmprendimientoService.php
|   |   └─ DocumentoService.php
|   |
|   ├─ Strategies/                        (patrón Strategy para
búsquedas)
|   |   ├─ BusquedaStrategyInterface.php
|   |   ├─ BusquedaPersonasStrategy.php
|   |   ├─ BusquedaProfesionalesStrategy.php
|   |   ├─ BusquedaEmpresasStrategy.php
|   |   └─ BusquedaEmprendimientosStrategy.php
|   |
|   ├─ Factories/                         (Factory Method / Abstract
Factory)
|   |   ├─ PostulanteFactoryInterface.php
|   |   ├─ PostulanteFactory.php         (crea
Persona/Profesional/Emprendedor/Empresa)
|   |   └─ OfertaFactory.php             (si quieren factoría de
ofertas)
|   |
|   ├─ Helpers/
|   |   ├─ HtmlHelper.php
|   |   ├─ SessionHelper.php
|   |   └─ ValidationHelper.php
|   |
|   └─ Views/
```

```
|     ├─ layouts/
|     |  ├─ main.php                  (layout principal con <head>, menú, etc.)
|     |  └─ auth.php                  (layout simple para login/registro)
|     |
|     ├─ partials/
|     |  ├─ navbar.php
|     |  ├─ footer.php
|     |  └─ alerts.php                (alertas Bootstrap)
|     |
|     ├─ home/
|     |  └─ index.php                 (landing con menú de tipos de postulante)
|     |
|     ├─ auth/
|     |  ├─ login.php
|     |  └─ register.php
|     |
|     ├─ postulantes/
|     |  ├─ seleccionar_tipo.php      (pantalla para elegir tipo de postulante)
|     |  ├─ form_persona.php
|     |  ├─ form_profesional.php
|     |  ├─ form_emprendedor.php
|     |  └─ perfil.php
```

```
|     |
|     ├─ empresas/
|     |  ├─ form_empresa.php
|     |  └─ perfil_empresa.php
|     |
|     ├─ ofertas/
|     |  ├─ listado.php
|     |  ├─ detalle.php
|     |  └─ form_oferta.php
|     |
|     ├─ postulaciones/
|     |  ├─ listado_postulaciones.php
|     |  └─ detalle_postulacion.php
|     |
|     ├─ emprendimientos/
|     |  ├─ listado.php
|     |  ├─ detalle.php
|     |  └─ form_emprendimiento.php
|     |
|     └─ documentos/
|        ├─ listado.php
|        └─ form_upload.php

├─ storage/
```

```
|   ├─ logs/
|   |   └─ app.log
|   └─ uploads/                (almacenamiento interno de archivos)
|
├─ vendor/                      (si usan Composer para autoload)
|   └─ ...                      (opcional)
|
├─ composer.json                (opcional, para PSR-4/autoload)
└─ README.md
```

2. Rol de los archivos clave

2.1. Entrada y routing

`public/index.php`
Punto de entrada único. Carga el autoloader, config, router y despacha la request:

```
<?php

// public/index.php

require __DIR__ . '/../app/Core/Autoloader.php';

require __DIR__ . '/../app/Config/config.php';

require __DIR__ . '/../app/Config/routes.php';


use App\Core\Router;


$router = new Router();
```

```
require __DIR__ . '/../app/Config/routes.php';


$router->dispatch($_SERVER['REQUEST_URI'],
$_SERVER['REQUEST_METHOD']);
```

-
- `app/Core/Router.php`
  Implementa un router simple tipo: `GET /ofertas -> OfertaController@index`.

`app/Config/routes.php`
Registra las rutas, por ejemplo:

```
<?php

// app/Config/routes.php


$router->get('/', 'HomeController@index');

$router->get('/login', 'AuthController@showLoginForm');

$router->post('/login', 'AuthController@login');


$router->get('/postulantes/seleccionar-tipo',
'PostulanteController@seleccionarTipo');

$router->get('/postulantes/crear', 'PostulanteController@create');

$router->post('/postulantes/crear', 'PostulanteController@store');


$router->get('/ofertas', 'OfertaController@index');

$router->get('/ofertas/{id}', 'OfertaController@show');

$router->post('/ofertas/{id}/postular',
'PostulacionController@store');
```

-

2.2. Capa de dominio y repositorios

- `app/Models/Entities/*.php`
  Clases que representan las entidades del modelo de datos (un objeto por tabla "importante").

- `app/Models/Repositories/*.php`
  Interfaces y clases concretas para acceso a datos (PDO/MySQL).
  Ejemplo: `MySQLOfertaRepository` implementa `OfertaRepositoryInterface`.

Esto les permite aplicar SOLID (DIP) inyectando interfaces en los servicios.

2.3. Capa de servicios

- `app/Services/*Service.php`
  Encapsulan casos de uso:

  - `PostulanteService`: alta y edición de postulantes.
  - `OfertaService`: gestión de ofertas.
  - `Busqueda*Strategy` + `BusquedaService`: búsquedas diferenciadas por tipo de postulante/empresa usando Strategy.

2.4. Vistas con Bootstrap

- `app/Views/layouts/main.php`
  Contiene `<html>`, `<head>`, incluye Bootstrap (CDN) y un `<?php echo $content; ?>` para el contenido de cada vista.

- Cada vista concreta (`ofertas/listado.php`, `postulantes/form_profesional.php`, etc.) solo se centra en el contenido específico del caso.

Diseño de Base de Datos

Script SQL para MySQL

-- ==========

```
-- CREACIÓN DE BASE DE DATOS
-- ============================================================
CREATE DATABASE IF NOT EXISTS bolsa_empleo
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_unicode_ci;

USE bolsa_empleo;

-- ============================================================
-- TABLAS DE CATÁLOGOS / MAESTROS
-- ============================================================

CREATE TABLE tipos_postulante (
    id_tipo_postulante TINYINT UNSIGNED PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

INSERT INTO tipos_postulante (id_tipo_postulante, nombre, descripcion) VALUES
(1, 'Persona', 'Persona que busca empleo en general'),
(2, 'Profesional', 'Profesional con título y matrícula'),
(3, 'Emprendedor', 'Persona que desarrolla un emprendimiento propio'),
(4, 'Empresa', 'Empresa que publica ofertas laborales');

CREATE TABLE estados_postulante (
    id_estado_postulante TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

INSERT INTO estados_postulante (nombre, descripcion) VALUES
('Activo', 'Postulante activo'),
('Inactivo', 'Postulante inactivo'),
('Suspendido', 'Postulante suspendido');

CREATE TABLE rubros (
    id_rubro INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

CREATE TABLE tamano_empresas (
    id_tamano_empresa TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

CREATE TABLE niveles_estudio (
```

```
    id_nivel_estudio TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

CREATE TABLE estados_emprendimiento (
    id_estado_emprendimiento TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

CREATE TABLE tipos_contratacion (
    id_tipo_contratacion TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

CREATE TABLE tipos_jornada (
    id_tipo_jornada TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

CREATE TABLE estados_oferta (
    id_estado_oferta TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

CREATE TABLE estados_postulacion (
    id_estado_postulacion TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

CREATE TABLE tipos_documento (
    id_tipo_documento TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(255) NULL
) ENGINE=InnoDB;

-- ========================================================
-- TABLAS DE USUARIOS, PERSONAS, EMPRESAS Y POSTULANTES
-- ========================================================

CREATE TABLE usuarios (
    id_usuario INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL,
```

```
    email VARCHAR(100) NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    fecha_alta DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    activo TINYINT(1) NOT NULL DEFAULT 1,
    UNIQUE KEY uq_usuarios_username (username),
    UNIQUE KEY uq_usuarios_email (email)
) ENGINE=InnoDB;

CREATE TABLE personas (
    id_persona INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    dni VARCHAR(20) NOT NULL,
    fecha_nacimiento DATE NULL,
    telefono VARCHAR(30) NULL,
    email_contacto VARCHAR(100) NULL,
    direccion VARCHAR(255) NULL,
    localidad VARCHAR(100) NULL,
    provincia VARCHAR(100) NULL,
    pais VARCHAR(100) NULL,
    UNIQUE KEY uq_personas_dni (dni)
) ENGINE=InnoDB;

CREATE TABLE empresas (
    id_empresa INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    razon_social VARCHAR(150) NOT NULL,
    cuit VARCHAR(20) NOT NULL,
    id_rubro INT UNSIGNED NULL,
    id_tamano_empresa TINYINT UNSIGNED NULL,
    direccion VARCHAR(255) NULL,
    localidad VARCHAR(100) NULL,
    provincia VARCHAR(100) NULL,
    pais VARCHAR(100) NULL,
    telefono VARCHAR(30) NULL,
    email_contacto VARCHAR(100) NULL,
    sitio_web VARCHAR(200) NULL,
    UNIQUE KEY uq_empresas_cuit (cuit),
    CONSTRAINT fk_empresas_rubros
        FOREIGN KEY (id_rubro) REFERENCES rubros (id_rubro)
        ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT fk_empresas_tamano
        FOREIGN KEY (id_tamano_empresa) REFERENCES tamano_empresas
(id_tamano_empresa)
        ON UPDATE CASCADE ON DELETE SET NULL
) ENGINE=InnoDB;

CREATE TABLE postulantes (
    id_postulante INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
```

```
    id_usuario INT UNSIGNED NOT NULL,
    id_tipo_postulante TINYINT UNSIGNED NOT NULL,
    id_persona INT UNSIGNED NULL,
    id_empresa INT UNSIGNED NULL,
    id_estado_postulante TINYINT UNSIGNED NOT NULL,
    fecha_alta DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    UNIQUE KEY uq_postulantes_usuario (id_usuario),
    KEY idx_postulantes_tipo (id_tipo_postulante),
    KEY idx_postulantes_persona (id_persona),
    KEY idx_postulantes_empresa (id_empresa),
    CONSTRAINT fk_postulantes_usuarios
        FOREIGN KEY (id_usuario) REFERENCES usuarios (id_usuario)
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT fk_postulantes_tipo
        FOREIGN KEY (id_tipo_postulante) REFERENCES tipos_postulante
(id_tipo_postulante)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    CONSTRAINT fk_postulantes_personas
        FOREIGN KEY (id_persona) REFERENCES personas (id_persona)
        ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT fk_postulantes_empresas
        FOREIGN KEY (id_empresa) REFERENCES empresas (id_empresa)
        ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT fk_postulantes_estado
        FOREIGN KEY (id_estado_postulante) REFERENCES estados_postulante
(id_estado_postulante)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    CONSTRAINT chk_postulantes_persona_o_empresa
        CHECK (
            (id_persona IS NOT NULL AND id_empresa IS NULL)
            OR (id_persona IS NULL AND id_empresa IS NOT NULL)
        )
) ENGINE=InnoDB;

-- ========================================================
-- TABLA DE PROFESIONALES
-- ========================================================

CREATE TABLE profesionales (
    id_profesional INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    id_postulante INT UNSIGNED NOT NULL,
    titulo VARCHAR(150) NOT NULL,
    institucion VARCHAR(150) NOT NULL,
    matricula VARCHAR(50) NULL,
    anio_graduacion YEAR NULL,
    descripcion_perfil TEXT NULL,
    honorarios_referencia DECIMAL(10,2) NULL,
    UNIQUE KEY uq_profesionales_postulante (id_postulante),
```

```
    CONSTRAINT fk_profesionales_postulantes
        FOREIGN KEY (id_postulante) REFERENCES postulantes (id_postulante)
        ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=InnoDB;

-- ========================================================
-- TABLA DE EMPRENDIMIENTOS
-- ========================================================

CREATE TABLE emprendimientos (
    id_emprendimiento INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    id_postulante_emprendedor INT UNSIGNED NOT NULL,
    nombre VARCHAR(150) NOT NULL,
    descripcion TEXT NOT NULL,
    id_rubro INT UNSIGNED NULL,
    id_estado_emprendimiento TINYINT UNSIGNED NULL,
    sitio_web VARCHAR(200) NULL,
    redes_sociales VARCHAR(255) NULL,
    necesidades TEXT NULL,
    KEY idx_emprendimientos_postulante (id_postulante_emprendedor),
    CONSTRAINT fk_emprendimientos_postulante
        FOREIGN KEY (id_postulante_emprendedor) REFERENCES postulantes
(id_postulante)
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT fk_emprendimientos_rubro
        FOREIGN KEY (id_rubro) REFERENCES rubros (id_rubro)
        ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT fk_emprendimientos_estado
        FOREIGN KEY (id_estado_emprendimiento) REFERENCES estados_emprendimiento
(id_estado_emprendimiento)
        ON UPDATE CASCADE ON DELETE SET NULL
) ENGINE=InnoDB;

-- ========================================================
-- FORMACIONES ACADÉMICAS Y EXPERIENCIAS
-- ========================================================

CREATE TABLE formaciones_academicas (
    id_formacion INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    id_postulante INT UNSIGNED NOT NULL,
    id_nivel_estudio TINYINT UNSIGNED NOT NULL,
    titulo VARCHAR(150) NOT NULL,
    institucion VARCHAR(150) NOT NULL,
    anio_inicio YEAR NULL,
    anio_fin YEAR NULL,
    en_curso TINYINT(1) NOT NULL DEFAULT 0,
    KEY idx_formaciones_postulante (id_postulante),
    CONSTRAINT fk_formaciones_postulante
```

```
    FOREIGN KEY (id_postulante) REFERENCES postulantes (id_postulante)
    ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT fk_formaciones_nivel
    FOREIGN KEY (id_nivel_estudio) REFERENCES niveles_estudio (id_nivel_estudio)
    ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB;

CREATE TABLE experiencias_laborales (
  id_experiencia INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  id_postulante INT UNSIGNED NOT NULL,
  empresa_nombre VARCHAR(150) NOT NULL,
  puesto VARCHAR(150) NOT NULL,
  fecha_desde DATE NULL,
  fecha_hasta DATE NULL,
  actualmente TINYINT(1) NOT NULL DEFAULT 0,
  descripcion TEXT NULL,
  KEY idx_experiencias_postulante (id_postulante),
  CONSTRAINT fk_experiencias_postulante
    FOREIGN KEY (id_postulante) REFERENCES postulantes (id_postulante)
    ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=InnoDB;

-- ============================================================
-- OFERTAS LABORALES Y POSTULACIONES
-- ============================================================

CREATE TABLE ofertas_laborales (
  id_oferta INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  id_empresa INT UNSIGNED NOT NULL,
  titulo VARCHAR(150) NOT NULL,
  descripcion TEXT NOT NULL,
  id_rubro INT UNSIGNED NULL,
  id_tipo_postulante_objetivo TINYINT UNSIGNED NOT NULL,
  requisitos TEXT NULL,
  id_tipo_contratacion TINYINT UNSIGNED NULL,
  id_tipo_jornada TINYINT UNSIGNED NULL,
  rango_salarial_desde DECIMAL(10,2) NULL,
  rango_salarial_hasta DECIMAL(10,2) NULL,
  fecha_publicacion DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  id_estado_oferta TINYINT UNSIGNED NOT NULL,
  KEY idx_ofertas_empresa (id_empresa),
  KEY idx_ofertas_rubro (id_rubro),
  CONSTRAINT fk_ofertas_empresas
    FOREIGN KEY (id_empresa) REFERENCES empresas (id_empresa)
    ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT fk_ofertas_rubros
    FOREIGN KEY (id_rubro) REFERENCES rubros (id_rubro)
    ON UPDATE CASCADE ON DELETE SET NULL,
```

```
    CONSTRAINT fk_ofertas_tipo_postulante_obj
        FOREIGN KEY (id_tipo_postulante_objetivo) REFERENCES tipos_postulante
(id_tipo_postulante)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    CONSTRAINT fk_ofertas_tipo_contratacion
        FOREIGN KEY (id_tipo_contratacion) REFERENCES tipos_contratacion
(id_tipo_contratacion)
        ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT fk_ofertas_tipo_jornada
        FOREIGN KEY (id_tipo_jornada) REFERENCES tipos_jornada (id_tipo_jornada)
        ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT fk_ofertas_estado
        FOREIGN KEY (id_estado_oferta) REFERENCES estados_oferta (id_estado_oferta)
        ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB;

CREATE TABLE postulaciones (
    id_postulacion INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    id_oferta INT UNSIGNED NOT NULL,
    id_postulante INT UNSIGNED NOT NULL,
    fecha_postulacion DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    id_estado_postulacion TINYINT UNSIGNED NOT NULL,
    mensaje_postulante TEXT NULL,
    calificacion_empresa TINYINT NULL,
    UNIQUE KEY uq_postulaciones_oferta_postulante (id_oferta, id_postulante),
    KEY idx_postulaciones_oferta (id_oferta),
    KEY idx_postulaciones_postulante (id_postulante),
    CONSTRAINT fk_postulaciones_ofertas
        FOREIGN KEY (id_oferta) REFERENCES ofertas_laborales (id_oferta)
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT fk_postulaciones_postulantes
        FOREIGN KEY (id_postulante) REFERENCES postulantes (id_postulante)
        ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT fk_postulaciones_estado
        FOREIGN KEY (id_estado_postulacion) REFERENCES estados_postulacion
(id_estado_postulacion)
        ON UPDATE CASCADE ON DELETE RESTRICT
) ENGINE=InnoDB;

-- ============================================================
-- DOCUMENTOS ADJUNTOS
-- ============================================================

CREATE TABLE documentos_adjuntos (
    id_documento INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    id_postulante INT UNSIGNED NULL,
    id_emprendimiento INT UNSIGNED NULL,
    id_tipo_documento TINYINT UNSIGNED NOT NULL,
```

```
    nombre_archivo_original VARCHAR(255) NOT NULL,
    nombre_archivo_sistema VARCHAR(255) NOT NULL,
    ruta_archivo VARCHAR(255) NOT NULL,
    mime_type VARCHAR(100) NULL,
    tamano_bytes BIGINT NULL,
    fecha_carga DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    KEY idx_documentos_postulante (id_postulante),
    KEY idx_documentos_emprendimiento (id_emprendimiento),
    CONSTRAINT fk_documentos_postulante
        FOREIGN KEY (id_postulante) REFERENCES postulantes (id_postulante)
        ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT fk_documentos_emprendimiento
        FOREIGN KEY (id_emprendimiento) REFERENCES emprendimientos
(id_emprendimiento)
        ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT fk_documentos_tipo
        FOREIGN KEY (id_tipo_documento) REFERENCES tipos_documento
(id_tipo_documento)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    CONSTRAINT chk_documentos_destino
        CHECK (
            id_postulante IS NOT NULL
            OR id_emprendimiento IS NOT NULL
        )
) ENGINE=InnoDB;
```

PASO A PASO

1. Aterrizar el problema y los actores

1. Identificar los 4 tipos de actores:
    - Persona que busca empleo.
    - Profesional.
    - Emprendedor.
    - Empresa.
2. Definir para cada actor:
    - Qué datos necesita registrar.
    - Qué acciones puede realizar en el sistema.
    - Con quién interactúa (ej.: Empresa ↔ Persona/Profesional; Emprendedor ↔ Empresas/Institución).
3. Listar los **casos de uso** principales:
    - Registro y login.
    - Alta/edición de perfil para cada tipo.
    - Publicación y gestión de ofertas (empresas).
    - Búsqueda de perfiles (empresas).
    - Búsqueda de ofertas (personas/profesionales).
    - Registro y difusión de emprendimientos (emprendedores).
    - Carga y gestión de documentación (todos).

Resultado: un documento breve con los casos de uso y actores, para no improvisar sobre la marcha.

---

2. Validar y ajustar el modelo de datos

1. Tomar el diseño de base de datos que ya definimos (tablas y relaciones) y:

- Verificar que cada caso de uso tenga soporte en el modelo.
- Agregar/ajustar columnas solo si hacen falta para un caso de uso concreto (por ejemplo, flags, timestamps extra, etc.).

2. Chequear:
   - Claves primarias y foráneas.
   - Índices básicos (búsquedas por rubro, tipo de postulante, empresa, etc.).
   - Consistencia de tipos de datos (fechas, textos, flags booleanos, etc.).
3. Dibujar o actualizar un **diagrama ER**:
   - Entidades principales: usuarios, postulantes, personas, empresas, ofertas_laborales, postulaciones, emprendimientos, documentos_adjuntos, etc.
   - Relaciones cardinalidad 1–N, N–N resueltas con tablas intermedias si fuese necesario.

Resultado: modelo de datos “cerrado” y validado frente a los casos de uso.

---

3. Definir el modelo de dominio (POO)

1. Identificar las **entidades de dominio** principales:
   - Usuario, Postulante, Persona, Empresa, Profesional, Emprendimiento, OfertaLaboral, Postulacion, FormacionAcademica, ExperienciaLaboral, DocumentoAdjunto, etc.
2. Decidir la jerarquía de clases:
   - Clase base/abstracta o interfaz `Postulante`.
   - Subtipos: `PersonaPostulante`, `ProfesionalPostulante`, `EmprendedorPostulante`, `EmpresaPostulante`.
3. Para cada entidad:
   - Definir sus atributos conceptuales (sin pensar todavía en columnas exactas).

    - Definir las responsabilidades: qué sabe hacer (comportamiento), no solo qué datos tiene.

4. Alinear con **SOLID**:
    - Responsabilidad Única: una clase = un concepto claro.
    - Abierto/Cerrado: posibilidad de agregar nuevos tipos de postulante sin romper lo existente.
    - Sustitución de Liskov: los subtipos de `Postulante` deben comportarse como un `Postulante`.
    - Inversión de Dependencias: controladores/servicios dependen de interfaces, no de implementaciones concretas.

Resultado: un diagrama de clases simple con entidades, relaciones y jerarquías.

---

4. Elegir y ubicar los patrones de diseño

1. **Factory (o Abstract Factory)**
    - Objetivo: crear instancias del tipo de postulante correcto según `id_tipo_postulante`.
    - Decisión: diseñar una `PostulanteFactory` que reciba datos (o un registro DB) y devuelva el objeto específico.
2. **Strategy (búsquedas)**
    - Objetivo: aplicar distintas estrategias de búsqueda/filtrado según el tipo de actor o contexto:
        - Búsqueda de personas/profesionales.
        - Búsqueda de empresas.
        - Búsqueda de emprendimientos.
    - Definir una interfaz de estrategia de búsqueda y varias implementaciones concretas.
3. **Repository / DAO**

- Objetivo: desacoplar la lógica de negocio del acceso a datos.
- Diseñar interfaces tipo `UsuarioRepositoryInterface`, `PostulanteRepositoryInterface`, `OfertaRepositoryInterface`, etc.
- Decidir qué métodos expone cada repositorio: guardar, buscar por id, buscar por criterios, etc.

4. (Opcional) Otros patrones:
   - DTOs para transporte de datos entre capas.
   - Builder para objetos complejos (si se complica la construcción de ofertas o perfiles).

Resultado: documento corto que diga:
"Usamos Factory para esto, Strategy para esto, Repository para esto. Interfaces involucradas: X, Y, Z."

---

5. Diseñar la arquitectura de capas y carpetas

1. Definir las capas lógicas:
   - Presentación (vistas con PHP + Bootstrap).
   - Controladores (reciben petición HTTP, validan, llaman a servicios).
   - Servicios (lógica de negocio y casos de uso).
   - Repositorios (acceso a base de datos).
   - Modelo de dominio (entidades).
2. Mapear estas capas a la estructura de carpetas:
   - `app/Core` → Router, Controller base, View, Database.
   - `app/Controllers` → controladores principales.
   - `app/Models/Entities` → clases de dominio.

- `app/Models/Repositories` → interfaces + implementación MySQL.
- `app/Services` → casos de uso concretos (gestión de postulantes, ofertas, etc.).
- `app/Views` → vistas para cada módulo (auth, postulantes, empresas, ofertas, etc.).

3. Asegurar que:
   - La capa superior **no dependa directamente** de MySQL, sino de interfaces (DIP).
   - La lógica de negocio **no dependa** de detalles del framework o del HTML.

Resultado: un diagrama o listado claro de capas y dependencias.

---

6. Definir las rutas y flujos de navegación

1. Listar las rutas principales:
   - Inicio / landing.
   - Login / logout / registro.
   - Selección de tipo de postulante.
   - CRUD de perfil (persona/profesional/emprendedor/empresa).
   - Listado de ofertas.
   - Detalle de oferta.
   - Postulación a oferta.
   - Listado de emprendimientos / detalle.
   - Carga y gestión de documentos.
2. Para cada ruta:
   - Asignar controlador y acción (por ejemplo: `OfertaController@index`, `PostulanteController@create`, etc.).

    - Definir qué vista se renderiza y qué datos espera.

3. Diseñar el **menú principal**:
    - Link a selección de tipo de postulante.
    - Acceso a ofertas.
    - Acceso a emprendimientos.
    - Acceso a panel de empresa (si logueado como empresa).
4. Definir flujos típicos:
    - “Nuevo usuario” → registro → seleccionar tipo → completar datos del perfil.
    - “Empresa logueada” → crear oferta → ver postulaciones → ver CV y profesionales.

Resultado: un mapa de navegación y una tabla de rutas.

---

7. Diseñar las vistas (sin maquetar todavía)

1. Para cada caso de uso, pensar:
    - Qué formularios hacen falta (campos, validaciones mínimas).
    - Qué listados hacen falta (columnas clave, filtros).
    - Qué vistas de detalle hacen falta (datos críticos y documentos).
2. Definir un layout base:
    - Encabezado con menú, logo, enlaces de sesión.
    - Zona de contenido.
    - Footer simple.
3. Diseñar la distribución de componentes con mentalidad Bootstrap:
    - Grid para formularios.

- Cards para ofertas y perfiles.
- Alerts para mensajes de error/éxito.

Resultado: bocetos (aunque sea en papel) de las pantallas clave.

---

8. Planificar el orden de implementación

Recomendación pragmática:

1. Autenticación básica (usuarios, login, registro).
2. Módulo de postulantes:
    - Selección de tipo.
    - Creación y edición de perfil para persona/profesional/emprendedor.
    - Asociación con `postulantes` y `personas`.
3. Módulo de empresas:
    - Registro de empresas.
    - Panel básico de empresa.
4. Módulo de ofertas:
    - ABM de ofertas por empresa.
    - Listado público de ofertas.
    - Detalle de oferta.
5. Módulo de postulaciones:
    - Postulación a oferta (como persona/profesional).
    - Listado de postulaciones para empresa.
6. Módulo de emprendimientos:
    - Alta/edición de emprendimientos por emprendedor.

    - Listado y detalle para visualización.

7. Módulo de documentos:
    - Alta/listado/eliminación de documentos por tipo de postulante/emprendimiento.

Resultado: un cronograma de implementación en iteraciones, ideal para trabajar en equipo o por entregables.

---

9. Definir criterios de prueba

1. Pruebas funcionales básicas:
    - Alta de usuario + login + selección de tipo.
    - Alta de perfil persona/profesional/emprendedor/empresa.
    - Publicación de oferta y visualización desde otro tipo de usuario.
    - Postulación a oferta y visualización por la empresa.
    - Alta de emprendimiento y visualización.
2. Pruebas de integridad de datos:
    - Que no se creen postulantes sin usuario.
    - Que no se puedan duplicar DNI o CUIT donde no corresponda.
    - Que las relaciones (FK) funcionen correctamente.
3. Pruebas de usabilidad mínima:
    - Flujo entendible desde el menú.
    - Formularios claros y consistentes.

Resultado: checklist de pruebas que los alumnos deben ejecutar y documentar.

PROMPT

You are an elite-level software engineer and code generator.

No matter the input language, you must always answer in **Spanish**.
Your main job is to **design and write high-quality code**.

1. Role and scope

You operate as a senior developer specialized in:

- Backend: PHP 8+, Python, Node.js
- Frontend: HTML5, CSS3, JavaScript, Bootstrap
- Databases: MySQL, PostgreSQL, SQL in general
- Arquitectura limpia, SOLID, patrones de diseño
- REST APIs, autenticación, seguridad básica (OWASP)
- Refactorización, pruebas y buenas prácticas

You prioritize **working, clean, maintainable code** over teoría.

2. Language & style

- Always respond in **Spanish**, even if the user writes in English.
- Use a **professional, directo, técnico** tone.
- Avoid explicaciones para principiantes a menos que el usuario las pida.
- Estructura las respuestas con secciones claras cuando sean extensas.

3. Code generation rules

When the user asks for code, you must:

1. Entregar **código funcional y coherente**, no pseudo-código.
2. Usar **buenas prácticas estándar** del lenguaje (por ejemplo, PSR-12 en PHP).

3. Incluir lo mínimo necesario para que el ejemplo sea **ejecutable o fácilmente integrable**.
4. Indicar claramente:
    - Supuestos (por ejemplo: nombre de la base de datos, tabla, entorno)
    - Dependencias necesarias (extensiones, librerías, etc.)
5. Evitar dependencias innecesarias o frameworks si el usuario pide “código puro”.

Para SQL:

- Utiliza sintaxis correcta para MySQL (o la base indicada por el usuario).
- Declara claves primarias, foráneas e índices cuando corresponda.
- Cuida tipos de datos coherentes.

Para PHP:

- Usa PHP 8+ (tipado, nullsafe, etc., cuando tenga sentido).
- Usa PDO para acceso a base de datos, con consultas preparadas.
- Evita mezclar lógica de negocio y HTML de forma caótica.

4. Manejo de especificaciones y ambigüedades

- Si el requerimiento es **poco claro o incompleto**, primero:
    - Pide las aclaraciones mínimas necesarias, con preguntas concretas.
- Si aun así debes asumir algo, deja explícitos tus supuestos antes del código.
- No inventes librerías, APIs o características que no existan.

5. Seguridad y calidad

Siempre que escribas código:

- Evita SQL injection → usa consultas preparadas.
- No escribas ejemplos que manejen contraseñas sin hashing.
- Señala cuando algo que el usuario pide es inseguro o mala práctica.
- Sugiere una versión segura o mejorada.

6. Formato de la respuesta

Cuando entregues código:

1. Explica muy brevemente el objetivo del snippet (1–3 líneas máximo).
2. Muestra el código en bloques bien formateados.
3. Si el proyecto es grande, propone una estructura de carpetas y aclara dónde va cada archivo.
4. Si el usuario lo pide, agrega comentarios en el código (en español, concisos y útiles).

Ejemplo de estructura deseada para respuestas largas:

- Contexto / objetivo
- Supuestos (si los hay)
- Estructura propuesta (carpetas, capas, módulos) – opcional
- Código(s) fuente
- Notas finales (seguridad, extensibilidad, mejoras posibles)

7. Comportamiento ante errores del usuario

Si el usuario propone algo:

- Técnicamente incorrecto
- Inseguro
- Muy ineficiente

Debes:

1. Decir claramente por qué es un problema.

2. Proponer una solución correcta o una alternativa mejor.

8. Regla final

Tu prioridad absoluta es ayudar al usuario a **diseñar y escribir código profesional**, ejecutable y mantenible, en español, con criterios de desarrollador senior.

Guía para crear el sistema usando repositorio GitHub

---

1. Objetivo de la guía

Dar un paso a paso para que el sistema se desarrolle:

- En equipo.
- Versionado correctamente con **Git + GitHub**.
- Con un flujo de trabajo profesional (branches, PR, issues, releases).

La idea es que el repositorio cuente la historia del proyecto: decisiones, commits, issues y versiones.

---

2. Preparación inicial

2.1. Requisitos técnicos

Cada integrante del equipo debe tener:

- Cuenta en **GitHub**.
- **Git** instalado en su máquina.
- Entorno de desarrollo:
    - PHP 8+.
    - Servidor local (XAMPP, Laragon, etc.).
    - MySQL.
- Editor/IDE (VS Code, PhpStorm, etc.).

2.2. Organización del equipo

Definir:

- 1 responsable de repositorio (owner/maintainer).
- 1 responsable de integración (merge de PR).
- Resto: desarrolladores con tareas específicas (backend, vistas, DB, etc.).

---

3. Crear el repositorio en GitHub

1. En GitHub, crear un nuevo repositorio:
    - Nombre sugerido: `bolsa-empleo-poo-php` o similar.
    - Visibilidad: `Private` (para uso académico) o `Public` si se desea mostrar.
2. Opciones iniciales:
    - Agregar archivo `README.md` básico.
    - Agregar `.gitignore` de PHP (GitHub tiene plantilla).
    - No hace falta añadir licencia si es solo académico (opcional).
3. Configurar rama principal:
    - Nombre: `main`.
    - En **Settings** → **Branches**:
        - Proteger `main`:
            - Requerir pull request para merge.
            - Requerir al menos 1 code review antes de merge (si el equipo lo permite).

---

4. Clonar el repositorio para trabajar localmente

Cada integrante:

1. Clona el repo:
    - Usando HTTPS o SSH (según config de cada uno).
2. Verifica que todo está bien:
    - Que exista `README.md`.

- Que pueda crear branches y hacer `push`.

---

5. Estructura base del proyecto (primer commit serio)

Sobre el repositorio clonado, crear la estructura mínima:

- `public/`
- `app/` (Core, Controllers, Models, Services, Views, etc.)
- `storage/`
- `config/` (o `app/Config/` según convención que uses)
- `sql/` (para guardar el script de creación de base de datos)
- `README.md` (extendido)
- `.gitignore` (ajustado para excluir `/storage/uploads`, `vendor`, etc.)

Flujo recomendado:

1. Crear una branch inicial:
   - Algo como: `feature/project-skeleton`.
2. Dentro de esa branch:
   - Crear carpetas vacías.
   - Añadir un `README.md` actualizado explicando el proyecto.
   - Crear archivo `sql/schema.sql` con el script de creación de tablas.
3. Hacer commits pequeños y descriptivos:
   - Ej: `chore: add base project structure`
   - `feat: add initial database schema`

4. Subir la branch al remoto y crear un **Pull Request (PR)** hacia `main`.
5. Revisar el PR (otro integrante, no el mismo).
6. Hacer el merge a `main`.

Resultado: rama `main` con estructura base + modelo de datos.

---

6. Definir el flujo de trabajo con branches (GitHub Flow)

6.1. Naming de branches

Usar nombres consistentes, por ejemplo:

- `feature/autenticacion-usuarios`
- `feature/registro-postulantes`
- `feature/empresas-ofertas`
- `feature/emprendimientos`
- `fix/ajuste-validaciones-form`
- `chore/mejora-readme`

6.2. Flujo estándar para cada tarea

Para cada funcionalidad:

1. Crear una **issue** en GitHub:
    - Describir:
        - Objetivo (ej: “Implementar registro de empresas”).
        - Tablas involucradas.
        - Controladores/vistas a modificar.

2. Crear una **branch** desde `main` vinculada a esa issue:
    - Usar el número de issue en el nombre, ej: `feature/12-registro-empresa`.
3. Desarrollar la funcionalidad localmente:
    - Cambios en modelos, servicios, controladores, vistas.
    - Ajustes en SQL si es necesario (migraciones simples o nota en `schema.sql`).
4. Hacer commits incrementales:
    - Un commit por gran paso (modelo, controlador, vistas, etc.).
5. Hacer `push` de la branch al remoto.
6. Crear un **Pull Request**:
    - Referenciar la issue: `Closes #12`.
    - Describir brevemente qué hace el cambio.
7. Otro miembro del equipo revisa el código (code review):
    - Verifica consistencia con el diseño general.
    - Revisa nombres, responsabilidades, seguridad básica.
8. Una vez aprobado, hacer **merge** a `main` vía PR (no directo).

---

7. Uso de GitHub Projects e Issues para organizar el trabajo

7.1. Crear un Project (Kanban simple)

1. En el repo, crear un **Project** tipo “Board” (To do / In progress / Done).
2. Crear columnas:

- `Backlog`
- `To Do`
- `In Progress`
- `Review`
- `Done`

7.2. Gestionar issues como tareas

1. Para cada módulo o iteración, crear issues:
   - “Implementar login y registro de usuarios.”
   - “Registrar postulantes persona/profesional/emprendedor.”
   - “Módulo ABM de empresas.”
   - “Ofertas laborales: ABM + listado + detalle.”
   - “Postulaciones de postulantes a ofertas.”
   - “Emprendimientos: alta y visualización.”
   - “Carga de documentos adjuntos.”
2. Asignar cada issue a miembros del equipo y moverla en el tablero:
   - `Backlog` → `To Do` → `In Progress` → `Review` → `Done`.

Resultado: visualización clara del avance del desarrollo.

---

8. Iteraciones recomendadas (sprints/modulos)

Iteración 1: Base del sistema

- Estructura de carpetas.
- Configuración de conexión a base de datos (clase Database, etc.).

- Script SQL en `/sql/schema.sql`.
- Página de inicio simple.

Branches típicas:

- `feature/base-config-db`
- `feature/home-landing`

Iteración 2: Autenticación y usuarios

- Registro de usuario.
- Login/logout.
- Asociación usuario ↔ postulante.
- Control de acceso básico (menú según tipo de usuario).

Branches:

- `feature/auth-register-login`
- `feature/nav-dinamica-por-tipo-usuario`

Iteración 3: Perfiles de postulantes (persona, profesional, emprendedor)

- Formularios para alta/edición de personas.
- Extensión a profesionales (datos específicos).
- Emprendedores (perfil + enlace a futuros emprendimientos).

Branches:

- `feature/perfil-persona`

- `feature/perfil-profesional`
- `feature/perfil-emprendedor`

Iteración 4: Empresas y ofertas laborales

- Registro y perfil de empresas.
- ABM de ofertas laborales.
- Vistas de listados y detalles de ofertas.

Branches:

- `feature/empresa-registro-perfil`
- `feature/ofertas-abm`
- `feature/ofertas-listado-detalle`

Iteración 5: Postulaciones y búsquedas

- Postulación a ofertas.
- Listado de postulaciones por empresa.
- Estrategias de búsqueda (Strategy) para distintos tipos de filtros.

Branches:

- `feature/postulaciones`
- `feature/busqueda-perfiles`
- `feature/busqueda-ofertas`

Iteración 6: Emprendimientos y documentos

- ABM de emprendimientos (emprendedores).
- Carga y administración de documentos adjuntos.
- Asociar documentos a postulantes/emprendimientos.

Branches:

- `feature/emprendimientos-abm`
- `feature/documentos-upload-gestion`

---

9. Buenas prácticas de commits y PR

9.1. Mensajes de commit

Usar mensajes claros, por ejemplo:

- `feat: add company registration form`
- `fix: correct validation on postulant profile`
- `refactor: extract oferta service`
- `chore: update readme with setup instructions`

9.2. Pull Requests

Cada PR debe incluir:

- Breve resumen de lo que hace.
- Referencia a la issue: `Closes #X`.
- Notas sobre cambios de schema (si aplican).
- Capturas de pantalla (opcional) si afecta a la UI.

---

10. Documentación en el README del repositorio

El `README.md` debe incluir:

1. Descripción del proyecto (breve).
2. Requisitos (PHP, MySQL, etc.).
3. Instrucciones de instalación:
    - Clonar repo.
    - Crear base de datos.
    - Importar `/sql/schema.sql`.
    - Configurar archivo de conexión (ruta y formato).
4. Estructura de carpetas.
5. Flujo de contribución (cómo crear branches, PR, etc.).
6. Autores/equipo.

11. Cierre de versión y “release” en GitHub

Cuando el sistema está en un estado estable (mínimo funcional):

1. Crear un **tag** en Git:
    - Algo como `v1.0.0-tp-integrador`.
2. Crear un **Release** en GitHub:
    - Título: “Versión 1.0 – TP Bolsa de Empleo”.
    - Describir qué funcionalidades están completas.
3. Opcional: adjuntar un PDF con documentación técnica del proyecto.

RESUMEN PRÁCTICO DEL TRABAJO QUE DEBEN REALIZAR LOS ALUMNOS

El objetivo del equipo es **entregar un sistema funcional mínimo (MVP)** de una bolsa de empleo con 4 tipos de postulantes, siguiendo POO, SOLID, patrones de diseño, una arquitectura ordenada y la base de datos previamente diseñada.

Los alumnos deben:

1) Preparar el entorno y proyecto

- Configurar repositorio GitHub.
- Subir la estructura de carpetas y archivos ya definida.
- Subir el script SQL de creación de tablas.
- Configurar entorno local: PHP 8+, MySQL, servidor local, Bootstrap.

---

2) Implementar los módulos esenciales (mínimo obligatorio)

A. Autenticación y Usuarios

- Registro y login de usuario.
- Relación usuario → tipo de postulante.
- Menú dinámico según el tipo.

B. Perfiles de Postulantes

Implementar formularios y guardado de:

- Persona (datos básicos).
- Profesional (datos + estudios + experiencia).
- Emprendedor (datos + emprendimiento base).

C. Empresas

- Registro y edición del perfil de empresa.

D. Ofertas laborales

- Creación, edición, eliminación de ofertas.
- Listado público.
- Detalle de oferta.

E. Postulaciones

- Postularse a una oferta.
- Empresas viendo postulaciones recibidas.

F. Emprendimientos

- Alta/edición de emprendimientos de emprendedores.
- Listado + detalle para visualización general.

G. Documentos adjuntos

- Carga, listado y borrado de documentos para cada tipo de postulante.

---

3) Aplicar patrones de diseño y buenas prácticas

- Factory para instanciar diferentes tipos de postulantes.
- Strategy para filtros de búsqueda según tipo.
- Repository pattern para acceso a la base de datos via PDO.
- Arquitectura limpia: separando Controladores, Servicios, Modelos, Vistas y Repositorios.

---

4) Crear vistas Bootstrap simples pero limpias

- Formularios coherentes y ordenados.
- Listados con tablas o cards.
- Navegación funcional.

---

5) Versionar todo correctamente en GitHub

- Issues por tarea.
- Branches por funcionalidad.
- Pull Requests.
- Readme con instrucciones de instalación.

DIVISIÓN DE TAREAS (GRUPO DE 2 PERSONAS – 15 DÍAS DE TRABAJO)

Dado que son dos personas, se divide el trabajo **por capas y módulos**, evitando bloqueos.

Rol A: Backend / Lógica / Base de Datos

Responsable del “corazón” del sistema.

Tareas:

- Conectar PHP ↔ MySQL utilizando PDO.
- Crear todos los Repositorios y Servicios.
- Implementar:
    - Registro/login
    - CRUD postulantes (persona/profesional/emprendedor)
    - CRUD empresas
    - CRUD ofertas
    - CRUD postulaciones
    - CRUD documentos
- Construir y mantener la lógica común en controladores.
- Implementar Factory + Strategy correctamente.
- Revisar integridad y relaciones del modelo.

Objetivo del rol:

Que **toda la lógica funcione** sin depender de las vistas.

Rol B: Frontend / Vistas / Navegación / Pruebas

Responsable de interacción y visualización.

Tareas:

- Crear las vistas en Bootstrap (formularios, listados, detalles).
- Armar layout general (header, footer, navegación).
- Diseñar y montar el flujo visual:
    - Registro → Selección del tipo → Perfil.
    - Empresa → Alta de oferta → Visualización de postulaciones.

    - Emprendedor → Emprendimiento.
- Crear mensajes de error/éxito.
- Probar los flujos completos con datos reales.
- Reportar problemas al backend via Issues de GitHub.

Objetivo del rol:

Que **todo sea usable**, visible, navegable y consistente.

PLAN DE TRABAJO DIARIO (15 DÍAS)

Días 1–2

- Configurar GitHub.
- Subir estructura inicial.
- Crear DB en local.
- Validar modelo SQL.

Días 3–5

- Autenticación completa.
- Menú dinámico.
- Layout base (Bootstrap).

Días 6–8

Backend:

- CRUD de postulantes (persona, profesional, emprendedor).

Frontend:

- Formularios y vistas para cada tipo de postulante.

Días 9–11

Backend:

- Empresas + Ofertas laborales.

Frontend:

- Vistas de empresas y ofertas.

Días 12–13

Backend:

- Postulaciones.
- Emprendimientos.
- Documentos.

Frontend:

- Listados, detalles y paneles finales.

Día 14

- Pruebas completas.
- Correcciones.
- Ajustes de navegación.

Día 15

- Preparar entrega:
    - README final + instrucciones.
    - Taxonomía de carpetas.

    - Capturas de pantalla.
    - Export DB.
    - Últimos PR y merge final a `main`.

RESULTADO ESPERADO

- Un sistema funcional con base en PHP-Puro, POO, SOLID, patrones y MySQL.
- Navegación completa y formularios funcionales.
- Gestión integral de postulantes, empresas, ofertas, postulaciones y emprendimientos.
- Repositorio bien organizado con commits, issues y PRs.
- Documentación mínima en README.