

## Documentacion Tecnica

**Gestión de Reclutamiento de Talentos**

**Integrantes:**

Abal, Fernando Gabriel

Abalos, Lucas

Bianciotto, Matías Felipe

Gutiérrez, Diego Germán

Roldan Sambrana, Guadalupe Milagros

Sotelo, Evelyn Romina

Tabla de contenido

[Documentacion Tecnica 0](#_Toc197964207)

[Arquitectura del sistema 2](#_Toc197964208)

[Diagrama de Entidad-Relación (DER) 2](#_Toc197964209)

[Diagrama de Caso de Uso 3](#_Toc197964210)

[Estructura de almacenamiento de directorios y archivos. 4](#_Toc197964211)

[Modelo de prediccion con *Machine Learning* 5](#_Toc197964212)

[FlaskLocal.py 6](#_Toc197964213)

[Utilizalas siguientes dependecias: 6](#_Toc197964214)

[Implementa: 7](#_Toc197964215)

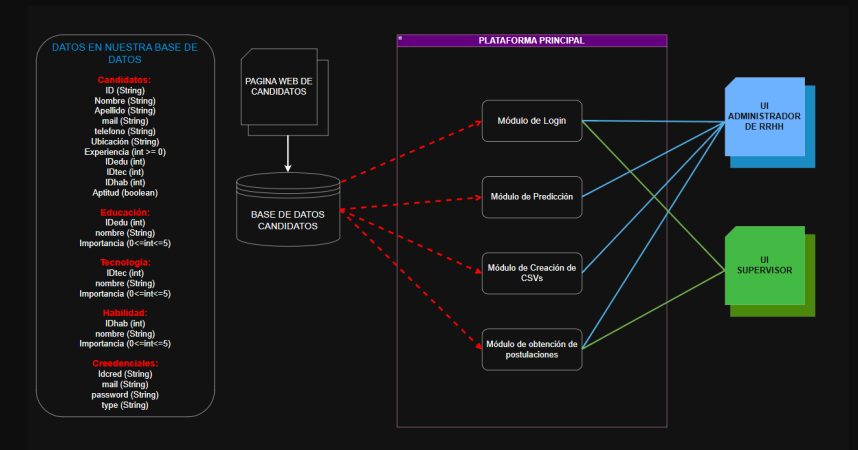
[FLaskCandidatos.py 12](#_Toc197964216)

[Utilizalas siguientes dependecias: 12](#_Toc197964217)

[Implementa: 13](#_Toc197964218)

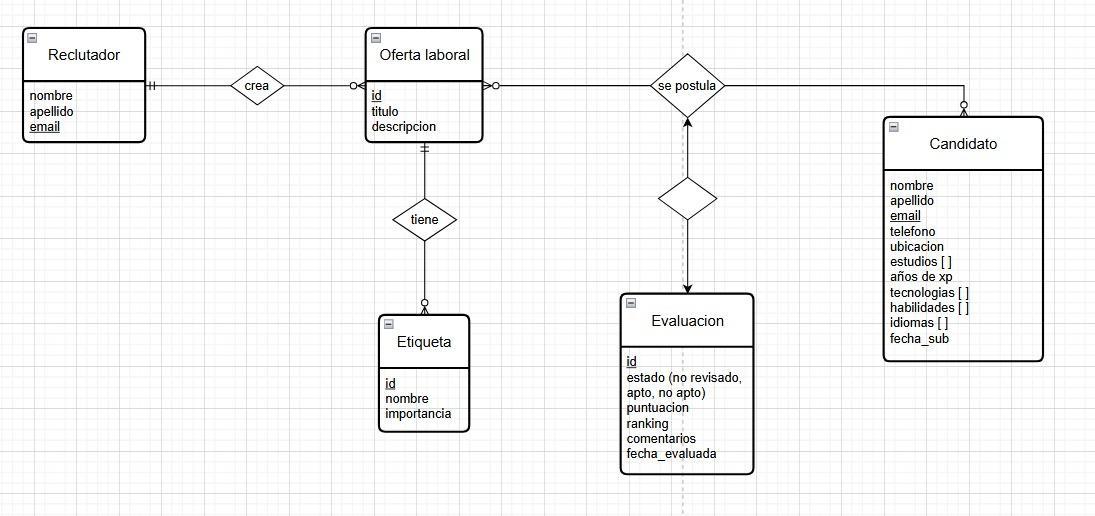
[Plan de Testing 14](#_Toc197964219)

## Arquitectura del sistema

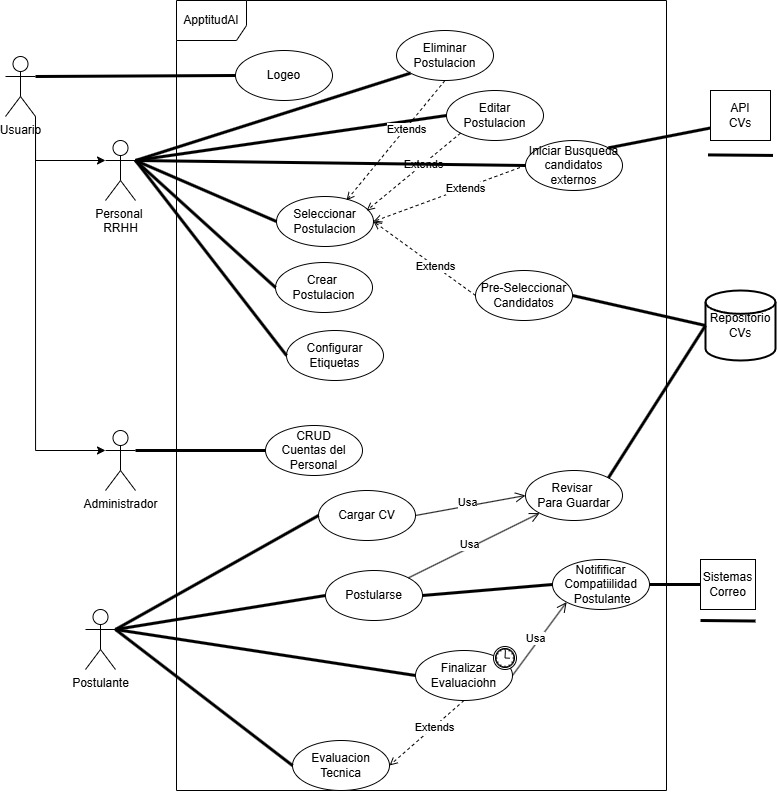


## Diagrama de Entidad-Relación (DER)

El siguiente [diagrama](https://drive.google.com/file/d/15Iron3d7x_bdPK7_NU7asYXO-oqRa7g1/view?usp=sharing) de entidad de relación (DER) consiste en un diseño de  base de datos para un sistema de gestión de reclutamiento de talento humano. Al inicio, un reclutador crea una oferta laboral la cual indica los requisitos necesarios para el puesto y contiene una etiqueta que indica la importancia. Luego, el candidato se postula a la oferta y a partir de ello se realiza la evaluación, teniendo en cuenta la puntuación, ranking, etc.



## Diagrama de Caso de Uso



## Estructura de almacenamiento de directorios y archivos.

📂 Apptitudai/

│

├── 📂 app/                     # Código principal de la aplicación

│   ├── \_\_init\_\_.py             # Inicialización de la aplicación Flask

│   ├── FlaskLocal.py

│   ├── FlaskCandidatos.py

│

├── 📂 static/                  # Archivos estáticos (CSS, JS, imágenes)

│   ├── 📂 styles/

│   │   ├── stylesCrear.css

│   │   ├── stylesIndex.css

│   │   ├── login.css

│   ├── 📂 js/

│   │   ├── crear.js

│   │   ├──drab.js

│   │   ├── nombreArchivo.js

│   ├── 📂 imágenes/

│   │   ├── logo.png

│

├── 📂 templates/               # Plantillas HTML para la interfaz

│   ├── base.html               # Plantilla base con estructura común

│   ├── login.html              # Página de autenticación

│   ├── crear.html

│   ├── estadisticas.html

│   ├── etiquetas.html

│   ├── index.html

│   ├── postulacion.html        # Formulario de postulación

│   ├── postulantes.html          # Panel de

│   ├── predecir.html        # Formulario de postulación

│   ├── resultados.html          # Panel de

│

├── 📂 database/

│   ├── erp\_rrhh.db

│

├── 📂 models/

│   ├── candidato.py

│   ├── educacion.py

│   ├── tecnologia.py

│   ├── habilidad.py

│   ├── usuario.py

│

├── 📂 encoders/

│   ├── encoder\_educacion.pkl

│   ├── encoder\_tecnologias.pkl

│   ├── encoder\_habilidades.pkl

│   ├── modelo\_candidatos.pkl

│

├── 📂 tests/                   # Pruebas automatizadas

│   ├── test\_auth.py            # Pruebas de autenticación

│   ├── test\_ranking.py         # Pruebas del algoritmo de clasificación

│

│

├── .gitignore

├── run.py

└── config.yaml

## Modelo de prediccion con *Machine Learning*

**Model.py**

**Preparación de datos:**

Se cargan en la variable “dataSet” el conjunto de datos desde “candidatos.csv”.

Se convierten los datos pertenecientes a las variables independientes “X” (“Experiencia”, “Educación”, “Tecnologías”, “Habilidades”) a valores numéricos gracias a los encoders respectivos.

Los datos pertenecientes a la categoría de apto(1) y no apto(0) están respetando lo mencionado anteriormente, quedando los valores de forma binaria.

En la variable dependiente “Y” se almacenan los datos que queremos predecir, en este caso “apto”.

**Creación del Modelo, Entrenamiento**

Con decisionTreeClasifier() se establece la técnica de machine learning y con accuracy\_score() se evalúa la precisión del modelo.

Con “joblib” se almacena el modelo entrenado en un archivo “.pkl”.

## FlaskLocal.py

### Utilizalas siguientes dependecias:

Bibliotecas estándar de Python:

* os (Manejo de archivos y directorios).
* sys (Control del sistema y ejecución de scripts).
* threading (Manejo de procesos en paralelo).
* re (Manejo de expresiones regulares).
* io (Manipulación de flujos de entrada y salida).
* base64 (Codificación y decodificación de datos en Base64).
* webbrowser (Apertura de URLs en navegadores).

Flask y manejo de aplicaciones web:

* Módulos importados de flask: Flask, redirect, render\_template, request, send\_file, session, url\_for, flash (Creación y administración de aplicaciones web).
* Modulo importado de flask\_sqlalchemy: SQLAlchemy (Uso de SQLAlchemy como Objet Ralational Mapping(ORM) para manejar bases de datos).

Procesamiento de datos:

* pandas (Manejo y análisis de datos estructurados).
* numpy (Cálculos numéricos y estructuras de datos avanzadas).

Machine Learning y procesamiento de datos:

* Módulo importado de sklearn.preprocessing: LabelEncoder (Codificación de variables categóricas).
* Módulos importados de sklearn.tree: DecisionTreeClassifier, export\_text, plot\_tree (Modelado de árboles de decisión).
* joblib (Guardado y carga de modelos entrenados).

Visualización de datos:

* matplotlib (Creación de gráficos y visualización de resultados).

Seguridad y autenticación:

* Módulos importados de werkzeug.security: generate\_password\_hash, check\_password\_hash (Manejo seguro de contraseñas).

### Implementa:

Configuracion de la aplicación Flask:

* Se crea una instancia de “Flask”, que es el marco de trabajo utilizado para construir aplicaciones web.
* Configuración de la clave secreta.

Configuracion de la conexion a la base de datos con SQLAlchemy como Objet Relational Mapping:

* Se define la URI de conexión con SQLite, especificando que la base de datos se almacenará en el archivo “erp\_rrhh.db”.
* Se desactiva “SQLALCHEMY\_TRACK\_MODIFICATIONS” para evitar el sobrecosto de rendimiento, ya que no se necesita rastrear cada modificación en objetos de SQLAlchemy.

Inicialización del ORM SQLAlchemy:

* Se crea una instancia de “SQLAlchemy”, que se vincula con la aplicación Flask (“app”).
* Esto permite manejar la base de datos de manera eficiente, sin necesidad de escribir consultas SQL manualmente.

Verificación de la conexión con la base de datos:

* print("Base de datos utilizada:", app.config['SQLALCHEMY\_DATABASE\_URI']), se imprime en consola la URL de la base de datos, asegurando que la configuración es correcta.

Modelos de datos:

* Candidato:
  + Representa a los postulantes dentro del sistema, almacenando información clave como nombre, apellido, correo y ubicación.
  + Se vincula con otros modelos (“Educacion”, “Tecnologia”, “Habilidad”) mediante claves foráneas para evaluar sus capacidades.
  + Incluye los atributos aptitud y puntaje, determinantes en la clasificación de candidatos.
* Educacion
  + Contiene los niveles educativos y su respectiva importancia, un parámetro utilizado en la evaluación de aptitudes.
  + Cada registro es único y permite establecer criterios diferenciados según la formación académica.
* Tecnologia
  + Define el catálogo de tecnologías que pueden poseer los postulantes.
  + Incluye un nivel de importancia, lo que facilita la ponderación en el algoritmo de clasificación.
* Habilidad
  + Similar al modelo “Tecnologia”, pero enfocado en habilidades generales.
  + Se vincula con los candidatos para registrar competencias clave en el proceso de evaluación.
* Usuario
  + Gestiona la autenticación dentro del sistema.
  + Almacena credenciales cifradas, garantizando seguridad en el acceso.
  + Incluye un atributo type, que diferencia roles y niveles de acceso en la plataforma.

Inicialización de la Base de Datos:

* Creación de la Base de Datos:
  + Se verifica si el archivo “erp\_rrhh.db” existe.
  + Si no existe, se crea utilizando “db.create\_all()” dentro del contexto de la aplicación Flask.
* Carga de Encoders:
  + Se cargan los modelos de codificación (“encoder\_educacion.pkl”, “encoder\_tecnologias.pkl”, “encoder\_habilidades.pkl”) para transformar los valores categóricos en datos estructurados.
* Inserción de Datos en Tablas de Clasificación:
  + Se iteran las clases de los encoders para poblar las tablas “Educacion”, “Tecnologia” y “Habilidad”.
  + Se usa “db.session.merge()” para evitar duplicados en la base de datos.
* Creación de Usuarios Iniciales:
* Se generan usuarios ficticios con roles definidos.
* Se emplea “generate\_password\_hash()” para almacenar contraseñas de manera segura.
* Los usuarios se guardan en la base de datos con “db.session.add()” y “db.session.commit()”.

Carga del Modelo y Encoders:

* Carga el modelo entrenado para evaluar postulantes.
* Se cargan los archivos de codificación (Label Encoding) para transformar los datos en valores numéricos.
* Se utiliza manejo de excepciones para evitar errores si los archivos no están disponibles.

get\_path(relative\_path():

* Usado para obtener la ruta del archivo, asegurando compatibilidad con entornos ejecutables.

abrir\_navegador():

* Abre automáticamente la aplicación web en el navegador, iniciando la interfaz de usuario.
* Utiliza la URL local (“127.0.0.1:5000”), lo que indica que la aplicación se ejecuta en modo desarrollo.

Definición de Rutas en Flask:

* Página Principal - Redirección a Login:
  + La ruta raíz (“/”) redirige automáticamente a la página de login.
  + Garantiza que los usuarios deben autenticarse antes de acceder a otras secciones del sistema.
* Autenticación de Usuarios:
  + Permite a los usuarios autenticarse mediante un formulario (“GET/POST”).
  + Valida las credenciales consultando la tabla “Usuario”.
  + Usa “check\_password\_hash()” para verificar contraseñas de manera segura.
* Redirección según el Rol:
* Después del login, los usuarios son redirigidos según su tipo de perfil.
* La lógica garantiza que cada usuario acceda a su panel correspondiente.

estadísticas():

* + “POST”: Redirige a la página de predicción (“predecir.html”).
  + “GET”:
    - 1. Carga el modelo de candidatos (“modelo\_candidatos.pkl”) y los encoders de datos (“encoder\_educacion.pkl”, “encoder\_habilidades.pkl”, “encoder\_tecnologias.pkl”).
    - 2. Procesa el dataset de entrenamiento (“entrenamientoActualizado.csv”), codificando las variables categóricas y ajustando la variable “Apto” a valores enteros (“1” para "Apto", “0” para "No Apto").
    - 3. Evalúa el modelo con los datos de entrenamiento y calcula su precisión.
    - 4. Extrae las clases de los encoders (“Educacion”, “Habilidades”, “Tecnologías”).
    - 5. Genera una visualización del árbol de decisión, la convierte en formato base64 y la envía a la plantilla “estadisticas.html”.

predecir():

* “POST”:
  + Verifica si se ha subido un archivo CSV y su validez.
  + Carga los encoders de datos y el modelo de candidatos.
  + Valida que el archivo contenga las columnas necesarias (“Experiencia”, “Educacion”, “Tecnologías”, “Habilidades”).
  + Transforma las columnas categóricas utilizando los encoders.
  + Realiza las predicciones con el modelo.
  + Agrega las predicciones al DataFrame y lo convierte en HTML.
  + Renderiza la plantilla “resultado.html”, mostrando la tabla con las predicciones.

postuantes():

* “GET”:
  + Obtiene todos los candidatos de la base de datos ordenados por puntaje.
  + Si no hay candidatos disponibles, muestra un mensaje informativo.
  + Reemplaza los valores de educación, habilidades y tecnologías con sus nombres correspondientes.
  + Si se proporciona el parámetro “filtro=apto”, filtra la lista para mostrar solo los candidatos aptos.

limpiar\_postulantes():

* “POST”:
* Borra todos los registros de la tabla “Candidato”.
* Vacía el archivo CSV de candidatos (“candidatosLocales.csv”).
* Redirige a la página de postulantes.

predecir\_postulantes():

* “POST”:
  + Obtiene los candidatos de la base de datos.
  + Carga el modelo entrenado (“modelo\_candidatos.pkl”).
  + Verifica que el dataset contenga las columnas requeridas.
  + Realiza predicciones sobre los candidatos.
  + Guarda los resultados en la base de datos, marcando a cada candidato como "Apto" o "No Apto".
  + Reemplaza los valores de educación, habilidades y tecnologías con sus nombres descriptivos.
  + Genera una tabla HTML con los resultados.

asignar\_puntajes():

* “GET”, “POST”:
  + Obtiene los candidatos aptos (“aptitud=True”).
  + Calcula su puntaje basado en:
    - Experiencia (`puntaje = experiencia \* 2`).
    - Educación (`importancia \* 3`).
    - Tecnologías (`importancia \* 5`).
    - Habilidades (`importancia \* 2`).
  + Guarda los puntajes en la base de datos y muestra un mensaje de éxito.

crear\_csv():

* “GET”, “POST”:
  + Carga opciones de educación, habilidades y tecnologías desde los encoders.
  + Permite al usuario ingresar datos y validar información como correo electrónico, teléfono y ubicación.
  + Agrega candidatos a la sesión y muestra los existentes.

guardar\_csv():

* “POST”:
  + Obtiene candidatos desde la sesión.
  + Convierte la lista en un DataFrame y lo guarda como “candidatosPagina.csv”.
  + Limpia la sesión tras la generación del archivo.

mostrar\_etiquetas():

* “GET”:
  + Obtiene datos de las tablas “Educacion”, “Tecnologia” y “Habilidad”.
  + Genera tablas HTML para visualizar los valores de importancia.

asignar\_valores():

* “POST”:
  + Modifica los valores de importancia de los elementos seleccionados en la base de datos.
  + Guarda los cambios y redirige a “/etiquetas”.

Descripción adicional de Ranking de Postulantes.

Cuando el reclutador se loguea en la aplicación tiene una pestaña extra para configurar las etiquetas.

En esta nueva pestaña extra, se puede ver todas las etiquetas que hay separadas en 3 tablas:

* Educación
* Tecnología
* Habilidades

Por ejemplo, en educación hay 3 etiquetas: secundario, universitario y posgrado.

Abajo de la vista de estas 3 tablas (educación, tecnología y habilidades), hay una sección para cambiarle la importancia a cada una. Por default, 0 es el nivel de importancia predeterminado. El nivel de importancia es establecido por el reclutador, es decir, el asigna el nivel de importancia a esa etiqueta. Se puede modificar el nivel de importancia las veces que requiera. Esto lo puede guardar al hacer clic en  botón guardar, de esta forma, se almacena la importancia de esa etiqueta en la base de datos.

El nivel de importancia es de 0 a 3( donde 0 es el nivel de menor importancia y 3 es el nivel de mayor importancia). Por ejemplo, si está buscando un desarrollador JAVA, busca la etiqueta “JAVA” en la tabla de tecnología  y le asigna nivel de importancia 3.

La importancia en las etiquetas se utiliza luego para el ranking de candidatos aptos. Por ejemplo, si un candidato tiene en su CV las etiquetas de mayor importancia va a estar más arriba del ranking.

Para crear el ranking se asigna un puntaje a cada candidato apto.

## FLaskCandidatos.py

### Utilizalas siguientes dependecias:

Bibliotecas estándar de Python:

* os (Manejo de archivos y directorios).
* sys (Control del sistema y ejecución de scripts).
* threading (Manejo de procesos en paralelo).
* re (Manejo de expresiones regulares).
* webbrowser (Apertura de URLs en navegadores).

Flask y manejo de aplicaciones web:

* Módulos importados de flask: Flask, redirect, render\_template, request, session, url\_for, flash (Creación y administración de aplicaciones web).

Módulo Local:

* Módulos importados de FlaskLocal: db, Candidato, Educacion, Tecnologia, Habilidad.

### Implementa:

Configuracion de la aplicación Flask:

* Se crea una instancia de “Flask”, que es el marco de trabajo utilizado para construir aplicaciones web.
* Configuración de la clave secreta.

Configuracion de la conexion a la base de datos con SQLAlchemy como Objet Relational Mapping:

* Se define la URI de conexión con SQLite, especificando que la base de datos se almacenará en el archivo “erp\_rrhh.db”.
* Se desactiva “SQLALCHEMY\_TRACK\_MODIFICATIONS” para evitar el sobrecosto de rendimiento, ya que no se necesita rastrear cada modificación en objetos de SQLAlchemy.
* Vincula la base de datos con “db.init\_app(app)”.

get\_path(relative\_path():

* Usado para obtener la ruta del archivo, asegurando compatibilidad con entornos ejecutables.

abrir\_navegador():

* Abre automáticamente la aplicación web en el navegador, iniciando la interfaz de usuario.
* Utiliza la URL local (“127.0.0.1:5001”), lo que indica que la aplicación se ejecuta en modo desarrollo.

home():

* “GET”:
  + Obtiene opciones de educación, tecnologías y habilidades desde la base de datos.
  + Almacena las opciones en la sesión.
  + Renderiza la plantilla “postulacion.html” con las opciones dinámicas.

postulacion():

* “GET”, “POST”:
  + Obtiene los datos del formulario de postulación (nombre, apellido, email, teléfono, ubicación, experiencia, educación, tecnologías y habilidades).
  + Valida el correo electrónico y el formato del teléfono.
  + Verifica que la ubicación esté dentro de las provincias de Argentina.
  + Busca los IDs correspondientes en las tablas “Educacion”, “Tecnologia” y “Habilidad”.
  + Crea un nuevo candidato y lo guarda en la base de datos.
  + Redirige a la página principal tras el registro exitoso.

Ejecución de la aplicación

* Inicia un temporizador para abrir el navegador automáticamente (“threading.Timer”) “threading.Timer(1.5, abrir\_navegador).start()”.
* Ejecuta la aplicación en `http://127.0.0.1:5001/` con “debug=False”, “app.run(debug=False, host="127.0.0.1", port=5001)”.

## Plan de Testing

Accede al siguiente enlace: [PlandeTesting](https://drive.google.com/drive/folders/1eBB6eBT-ZSVB-PO07kprNxT0D3qKv3N6?usp=drive_link)