



**Tecnológico
de Monterrey**

Desarrollo e implantación de sistemas de software (Gpo 103)

Manual de Inteligencia Artificial para PathExplorer

Docente:

Juan Carlos Lavariega Jarquín

Equipo “The Bytles”

Ana Karina Aramoni Ruíz

A07192068

Repositorio: https://github.com/AnaK-AR/TheBytles_Backend.git

Fecha: 12/Junio/2025

ÍNDICE

1. Introducción al Sistema y Glosario de Conceptos Clave	3
2. Flujo General de Inteligencia Artificial en PathExplorer	3
3. Generación de Resúmenes de Usuario	4
4. Extracción de Roles desde RFPs	4
5. Generación de Microtarefas de Crecimiento (Keypoints)	5
6. Generación y Uso de Embeddings	5
7. Matching Semántico con Funciones RPC en Supabase	6
8. Endpoints del Backend (FastAPI)	6
9. Archivos Python	6
10. Herramientas y Servicios Externos	7
11. Buenas Prácticas Aplicadas	7
12. Ejemplos Visuales y Resultados	8

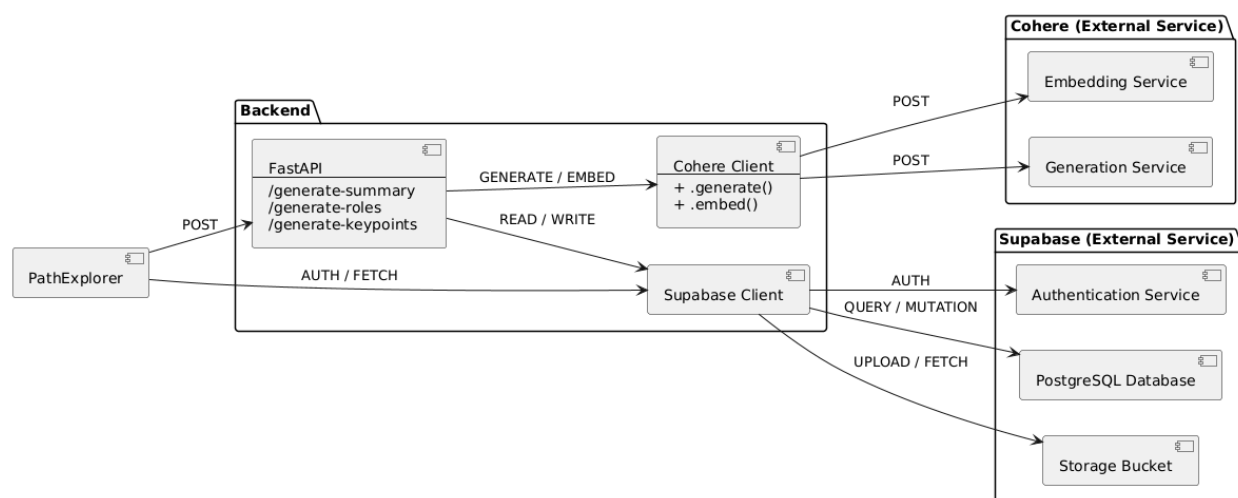
1. Introducción al Sistema y Glosario de Conceptos Clave

PathExplorer es una plataforma inteligente diseñada para optimizar la asignación de personas a roles o proyectos dentro de Accenture. Para facilitar y agilizar este proceso, se ha incorporado el uso de Inteligencia Artificial, que permite generar resúmenes de usuario, identificar necesidades de proyectos (RFPs), proponer tareas de desarrollo profesional, y recomendar certificaciones o cursos relevantes. Para aprovechar al máximo esta herramienta, es fundamental comprender los siguientes conceptos y definiciones:

- **Embedding:** Representación numérica de un texto que permite compararlo con otros textos en función de su similitud semántica.
- **Prompt:** Texto de entrada que se proporciona a un modelo de lenguaje (como los de Cohere) para guiar y estructurar su respuesta.
- **RFP (Request for Proposal):** Documento que describe los requerimientos que una organización necesita cubrir para un producto o servicio a desarrollar.
- **Keypoints:** Microtareas personalizadas diseñadas para impulsar el desarrollo profesional de un usuario.
- **RPC (Remote Procedure Call):** función que se ejecuta directamente en la base de datos (Supabase) para buscar resultados similares mediante embeddings.

2. Flujo General de Inteligencia Artificial en PathExplorer

El siguiente diagrama presenta la arquitectura del sistema junto con su flujo de funcionamiento, incluyendo la integración de componentes de inteligencia artificial. En él se distribuyen las tareas de la siguiente manera:



Arquitectura del sistema PathExplorer

- **FastAPI:** Hosteado en Railway, orquesta las solicitudes entre el frontend y los módulos de inteligencia artificial, exponiendo endpoints limpios, asíncronos y con documentación automática.
- **Cohere:** Se utiliza para dos tareas principales: la generación de texto mediante el modelo command-r-plus (resúmenes, roles, keypoints) y la creación de vectores semánticos (embeddings) con embed-english-v3.0, lo que habilita comparaciones inteligentes entre textos.
- **Supabase:** Proporciona una base de datos relacional (PostgreSQL), autenticación de usuarios segura, almacenamiento de archivos (como CVs y RFPs), y ejecución de funciones remotas (RPC) que permiten realizar búsquedas semánticas directamente desde la base de datos.

3. Generación de Resúmenes de Usuario

Para generar el resumen de usuario, se combinan múltiples fuentes de información: el contenido parseado del CV en PDF, la biografía del usuario, su capability y la lista de habilidades registradas en su perfil. Estos datos se estructuran en un prompt que se envía al modelo command-r-plus de Cohere, especializado en generación de texto. El modelo devuelve un resumen en una sola línea con el siguiente formato:

Responsibilities: [responsabilidades clave] · Skills: [lista de habilidades separadas por comas]

El resultado se guarda en la variable ai_summary y se convierte en un embedding para habilitar comparaciones semánticas en procesos posteriores.

4. Extracción de Roles desde RFPs

La identificación de roles a partir de documentos RFP se realiza extrayendo el texto parseado de archivos RFP en formato PDF. A partir de este contenido, se construye un prompt diseñado específicamente para solicitar a Cohere la generación de una línea por rol, con el siguiente formato:

Role: [nombre del rol] · Responsibilities: [funciones clave] · Skills: [habilidades requeridas]

Este proceso permite obtener roles estructurados y comparables, sin mención de empresas y limitados a las áreas de negocio relevantes para Accenture.

5. Generación de Microtarefas de Crecimiento (Keypoints)

A partir del resumen generado por el sistema y las metas definidas por el usuario, se construye un prompt que solicita a Cohere la generación exacta de cuatro tareas, divididas entre habilidades técnicas y blandas. Cada tarea debe formularse como una frase breve, concreta, accionable y fácil de comenzar de inmediato. Ejemplos de tareas generadas:

`Regularly review and analyze test coverage reports to identify areas of improvement and ensure comprehensive testing.`

`Practice active listening during team meetings to better understand colleagues' perspectives and foster a collaborative environment.`

`Develop a habit of breaking down complex testing scenarios into manageable steps, enhancing your problem-solving skills and logical thinking.`

`Create a GitHub repository to showcase your automated testing projects, leveraging its version control and collaboration features.`

6. Generación y Uso de Embeddings

Los embeddings son vectores numéricos generados a partir de texto mediante el modelo `embed-english-v3.0` de Cohere. En este sistema, se crean a partir de los siguientes campos:

- Resúmenes de usuario (`ai_summary`)
- Descripciones de roles generadas desde RFPs
- Certificaciones (`Cert_Des`)
- Cursos (`Course_Des`)

Una vez generados, estos embeddings se almacenan en campos tipo vector dentro de Supabase. Su principal utilidad es permitir comparaciones de similitud semántica entre perfiles, certificaciones, cursos, roles y necesidades de proyecto. De igual modo, los scripts `cohere_embed.py`, `embedding_course.py` y `embedding_certs.py` automatizan este proceso, asegurando que la base de datos se mantenga actualizada de forma continua.

7. Matching Semántico con Funciones RPC en Supabase

Una vez generados los resúmenes y embeddings, el sistema realiza recomendaciones utilizando funciones RPC en Supabase. Estas funciones devuelven un número determinado de elementos más cercanos semánticamente, y por tanto, más compatibles con el usuario solicitante:

- ***get_top8(vector)***: Encuentra los 8 usuarios más similares a un rol generado.
- ***get_top5_certificates(vector)***: Sugiere los 5 certificados más adecuados para el usuario.
- ***get_top5_courses(vector)***: Devuelve los 5 cursos más relevantes para un perfil.

Estas funciones comparan vectores utilizando distancia semántica (\leq), lo que permite establecer similitudes de forma precisa y eficiente.

8. Endpoints del Backend (FastAPI)

- ***GET /***: Verifica la disponibilidad del servidor backend.
- ***POST /generate-summary***: Genera un resumen de perfil a partir de la información del usuario.
- ***POST /generate-roles***: Extrae roles estructurados desde documentos RFP.
- ***POST /generate-keypoints***: Crea tareas de desarrollo profesional personalizadas.

9. Archivos Python

- ***main.py***: Define la API utilizando FastAPI, configura CORS y expone los distintos endpoints del servicio.
- ***generate_profile_summaries.py***: Descarga el CV en PDF, extrae el texto, recupera la biografía, capability y habilidades del usuario desde Supabase. Luego genera un resumen con *cohere_summarizer.py*, calcula su embedding y guarda ambos en la base de datos.
- ***cohere_summarizer.py***: Genera un resumen de una sola línea estructurada con responsabilidades y habilidades clave, utilizando el modelo *command-r-plus* de Cohere.
- ***generate_roles_from_rfp.py***: Extrae el texto de un archivo RFP, llama a *cohere_roles.py* para generar los roles, calcula sus embeddings y guarda cada rol junto a su vector correspondiente en Supabase.
- ***cohere_roles.py***: Genera múltiples roles en una línea cada uno, estructurando los campos de título, responsabilidades y habilidades, a partir del texto de un RFP.
- ***generate_keypoints.py***: Recupera el resumen (*ai_summary*) y las metas del usuario desde Supabase, llama a *cohere_keypoints.py*, y guarda las cuatro tareas de crecimiento en la tabla *Grow*.

- ***cohere_keypoints.py***: Genera cuatro microtarefas accionables (dos técnicas y dos blandas) basadas en las habilidades y objetivos del usuario. Las tareas se devuelven en texto directo, sin bullets ni frases de relleno.
- ***cohere_embed.py***: Convierte cualquier texto (resumen, rol, curso o certificación) en un vector numérico utilizando el modelo embed-english-v3.0 de Cohere.
- ***embedding_course.py***: Busca cursos sin vector en la tabla Course_Recomendation, genera sus embeddings y los guarda en Supabase.
- ***embeddings_certs.py***: Realiza el mismo proceso que embedding_course.py, pero aplicado a certificaciones en la tabla Cert_Recomendation.

10. Herramientas y Servicios Externos

- ***Cohere API***: Utilizada para la generación de texto mediante el modelo command-r-plus y la creación de embeddings con embed-english-v3.0.
- ***Supabase***: Proporciona servicios de autenticación, almacenamiento de archivos, base de datos relacional (PostgreSQL) y ejecución de funciones RPC.
- ***PyMuPDF***: Biblioteca empleada para la extracción precisa de texto desde archivos PDF.
- ***Transformers (Hugging Face)***: Utilizada para la tokenización segura de texto basada en modelos preentrenados.
- ***FastAPI***: Framework de backend en Python utilizado para construir la API de forma rápida, asincrónica y con documentación automática.

11. Buenas Prácticas Aplicadas

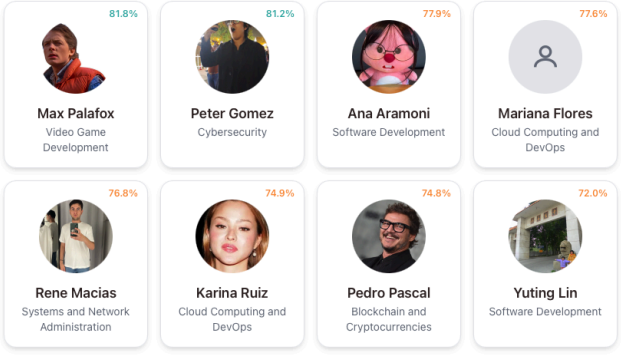
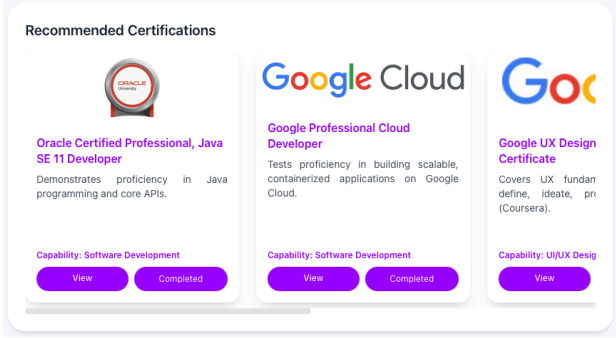
- ***Tokenización previa***: Se utiliza el tokenizador de Hugging Face para segmentar y recortar los textos antes de enviarlos a Cohere, evitando errores por exceder el límite de tokens y asegurando prompts más eficientes.
- ***Separación modular del código***: Cada funcionalidad (resúmenes, roles, keypoints, embeddings) está implementada en archivos independientes, lo que facilita la organización, el mantenimiento y la escalabilidad del sistema.
- ***Uso de .env***: Las claves API y credenciales sensibles se almacenan en archivos .env y se excluyen del repositorio mediante .gitignore, siguiendo buenas prácticas de seguridad.
- ***Manejo de errores con try-except***: Todas las operaciones críticas están encapsuladas en bloques try-except, lo que permite capturar errores, evitar caídas inesperadas y registrar mensajes útiles para la depuración.
- ***Nombres descriptivos y comentarios***: Las funciones y variables emplean nombres significativos, y se incluyen comentarios explicativos en las partes clave del código para mejorar la legibilidad y mantenibilidad.

- **Despliegue controlado y versionado:** La aplicación está preparada para ejecutarse en contenedores Docker y entornos virtuales, con todas sus dependencias definidas en requirements.txt, lo que facilita su distribución y actualización controlada.

12. Ejemplos Visuales y Resultados

A continuación se presentan ejemplos de salida generados por el sistema, organizados por tipo de campo. Cada sección ilustra cómo se procesan los datos y cuál es el resultado final tras aplicar los distintos módulos de inteligencia artificial.

	Entrada esperada	Salida generada	Ejemplo visual
POST /generate-summary	{"user_id": "uuid"}	Una línea con las principales responsabilidades y habilidades del usuario.	Responsibilities: QA Engineer and Tester, designing, executing, and documenting functional test cases that validate each feature against its acceptance criteria · Skills: Hardware/Software Debugging, Agile Team Collaboration, Multidisciplinary Teamwork Capability, Data Analysis, Storage and Backups, GitHub/GitLab, Team Leadership, Software Development
POST /generate-roles	{"project_id": "uuid"}	Varias líneas, cada una con: nombre del rol, responsabilidades y habilidades clave.	<p>Available Roles:</p> <p>Role: Full-Stack Developer · Responsibilities: Build the front-end and back-end layers of the platform, ensuring seamless integration with the cloud infrastructure. · Skills: JavaScript, React, Node.js, HTML/CSS, AWS/Azure/GCP, API Integration</p> <p>Role: QA Engineer · Responsibilities: Plan and execute comprehensive test strategies to ensure the platform's functionality, performance, and reliability meet expectations. · Skills: Software Testing, Automation, Performance Testing, Test Case Design, Defect Management</p>
POST /generate-keypoints	{"user_id": "uuid"}	Cuatro frases: dos enfocadas en soft skills y dos en habilidades técnicas.	<p>Recommendations for your Professional Growth</p> <p>Employee: Ana Aramoni</p> <ul style="list-style-type: none"> ✓ Develop a small Android app using Kotlin to familiarize yourself with the language and its capabilities. ✓ Engage in daily Spanish conversations to maintain and improve your language proficiency. ✓ Create a series of short coding tutorials on YouTube to teach others and reinforce your own JavaScript skills. ✓ Practice active listening during team meetings to foster collaboration and empathy.

RPC: get_top8()	Vector del rol (embedding)	Lista de usuarios con: firstName, lastName, capability, y nivel de similitud en forma de porcentaje.	
RPC: get_top5_certificate s()	Vector del usuario (embedding)	Lista de certificaciones con: Cert_Name, Cert_Des, y Capability.	
RPC: get_top5_ courses()	Vector del usuario (embedding)	Lista de cursos con: Course_Name, Course_Des, y Capability.	