FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# AI-Powered Career Opportunity Distribution System

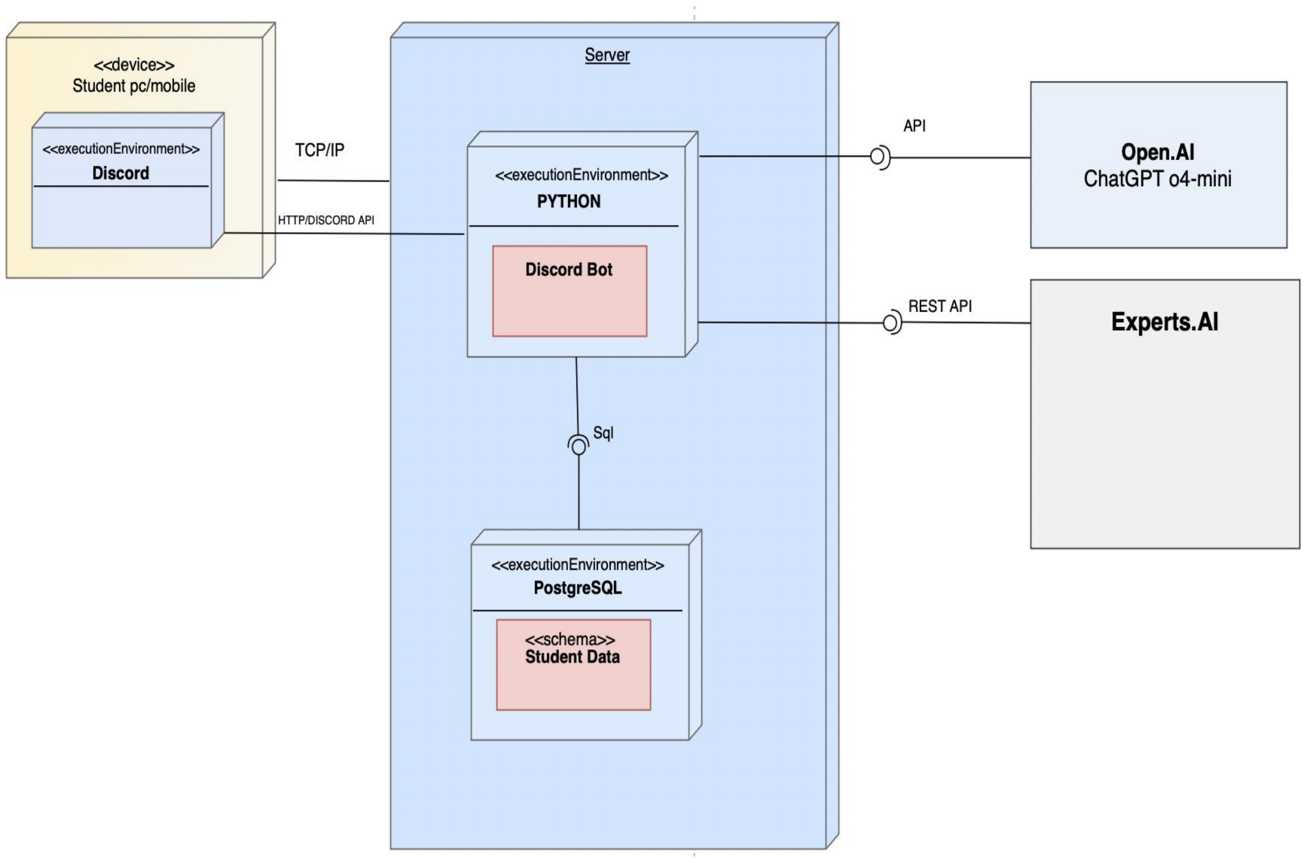
## Architecture and Database - Tech Stack

**Project documentation for the purpose of BIE-SWI course**

## 1. System architecture

The Job Match System follows a three-layered architecture: the presentation layer includes a Discord bot interface for students. Students interact with the system through the Discord application on their personal devices. The application layer handles business logic,it consists of the Discord Bot, which manages student input, handles messaging, and communicates with external services. The Discord Bot communicates directly with external services via APIs. The data layer stores student profiles in a PostgreSQL database hosted on the server. The system integrates with two external services: OpenAI's GPT-4-mini, which is accessed via API for natural language processing, and Experts.AI, which serves as a job listing provider and is accessed via a REST API. The Matching Engine acts as a backend service that pulls data from both the local student database and the remote Experts.AI system to generate relevant job matches based on student attributes and company requirements.

Figure 1 - System architecture



## 2. Architecture

The architecture is based on the following technologies:

- Python
- Experts.AI
- Discord Bot
- Open.AI ChatGpt o4-mini

The architecture is divided into three independent layers:

- Data layer, responsible for data persistence
- Business layer, responsible for business logic
- Presentation layer, responsible for presentation of application data

### 2.1. Presentation Layer(client layer)

- **Student Client (Discord Bot)**
  - Students interact with the system using the Discord application on their personal devices.
  - Through Discord messages and commands, students can submit their qualifications and preferences.
  - The Discord bot sends back personalized job matches and notifications.

### 2.2. Application Layer

- **Discord Bot service** (Python)
  - Receives and processes input from students.
  - Communicates with the matching engine via internal API calls.
  - Persists student data into the PostgreSQL database.
- **Integration with AI-driven Matching Engine** (open.ai chatgpt)
  - Consumes student profiles and fetches job data from Experts.AI via REST API.
  - Applies job-matching logic based on student qualifications and job posting requirements.
  - Sends match results to the Discord bot.
- **Integration with Experts.AI (External)**
  - Experts.AI is an external job platform accessed through a REST API.
  - It provides job listing data for use in the matching process.
  - It is not hosted within the system and operates independently from internal components.

### 2.3. Data Layer

- **Relational Database (e.g., PostgreSQL)**
  - Stores structured student profiles - data.

### 3. External Integrations

- **Discord API**
  - Enables interaction between students and the system via bots.
  - Discord API – Enables real-time interaction between the Discord bot and the Discord platform via events and commands.
  - Experts.AI REST API – Provides external job listings to be consumed by the matching engine.

### 4. Deployment Architecture

- **Backend components** (Discord Bot and Matching Engine) are deployed as **scalable Python services**, possibly containerized (e.g., via Docker) and orchestrated on a container platform.
- The **Discord bot** connects to the Discord Gateway and listens to user activity in real-time.
- The **Experts.AI platform** is external and not deployed within this system.

## 3. Authorization

### 1. Student Authentication via Discord

- **Step 1:** Student sends a message in the Discord server (e.g., !start)
- **Step 2:** The **Discord Bot** reads the message and extracts the **user's Discord ID**
- **Step 3:** Bot sends the ID to the backends' Student Service
- **Step 4:** Backend checks if the Discord ID exists in the students table in the database:

**If exists:** Student is authenticated

**If not:** Student is prompted to register (onboarding flow begins)

**Discord ID** acts as a unique, trusted identifier — we rely on **Discord's OAuth2 identity guarantee**.

**2. Admin Authentication via Web Dashboard (JWT)**

- **Step 1:** Admin opens Angular web dashboard
- **Step 2:** Admin logs in using a username and password (stored securely)
- **Step 3:** The backend validates credentials and issues a **JWT (JSON Web Token)**
- **Step 4:** This token is stored client-side (e.g., local Storage)
- **Step 5:** All future API requests from the admin panel include the JWT in the header

The backend checks:

- Token validity
- Expiration
- Role (admin, viewer, etc.)

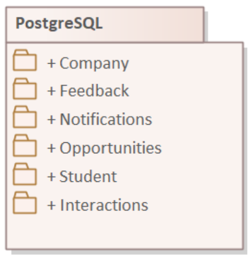
The Authorization Logic:

| **User Type** | **User Type** | **Authorization** |
|---|---|---|
| Student (via Bot) | Verified by Discord ID | Can view/respond to opportunities |
| Admin (via Dashboard) | JWT login | Can view logs, post new opportunities, and manage users |

**How We Verify the User?**

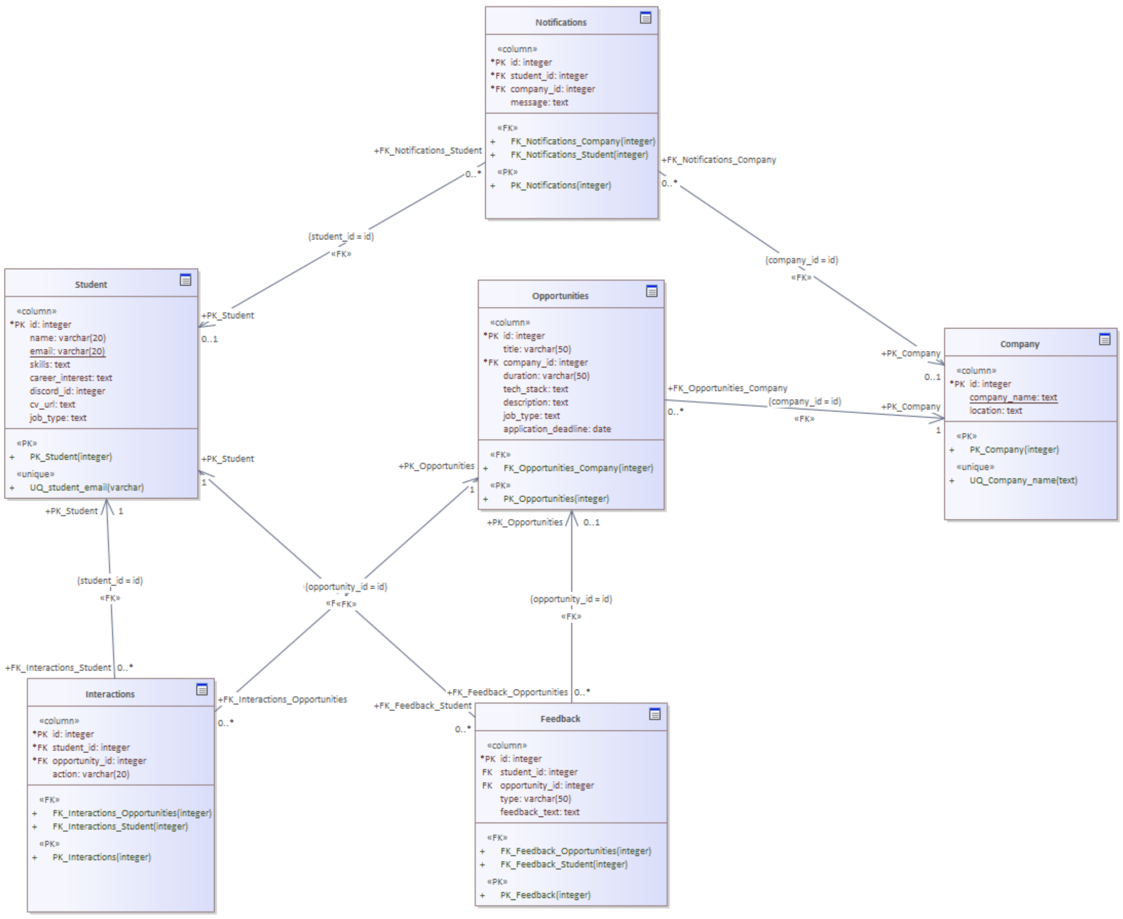
- **Discord User:** We trust Discord’s identity and store their ID in our system

Database Model:
This chapter presents the structure of the PostgreSQL database that underpins the AI-Powered Career-Opportunity Distribution System.



PostgreSQL:
This package describes the structure of the PostgreSQL database used by the AI-powered Career-Opportunity Distribution system**.**

It focuses on the core part of the scheme currently implemented — students, companies, opportunities and the supporting tables that record notifications, interactions and feedback


Notifications
«column»
*PK id: integer
*FK student_id: integer
*FK company_id: integer
message: text
«FK»
+ FK_Notifications_Company(integer)
+ FK_Notifications_Student(integer)
«PK»
+ PK_Notifications(integer)
+FK_Notifications_Student
0..*
(student_id = id)
«FK»
+FK_Notifications_Company
0..*
(company_id = id)
«FK»
Student
«column»
*PK id: integer
name: varchar(20)
email: varchar(20)
skills: text
career_interest: text
discord_id: integer
cv_url: text
job_type: text
«PK»
+ PK_Student(integer)
«unique»
+ UQ_student_email(varchar)
+PK_Student
0..1
+PK_Student
1
Opportunities
«column»
*PK id: integer
title: varchar(50)
*FK company_id: integer
duration: varchar(50)
tech_stack: text
description: text
job_type: text
application_deadline: date
«FK»
+ FK_Opportunities_Company(integer)
«PK»
+ PK_Opportunities(integer)
+FK_Opportunities_Company
0..*
(company_id = id)
«FK»
+PK_Company
0..1
+PK_Company
1
Company
«column»
*PK id: integer
company_name: text
location: text
«PK»
+ PK_Company(integer)
«unique»
+ UQ_Company_name(text)
+PK_Opportunities
1
+PK_Student
1
+PK_Opportunities
0..1
(student_id = id)
«FK»
(opportunity_id = id)
«F«FK»
(opportunity_id = id)
«FK»
+FK_Interactions_Student
0..*
+FK_Interactions_Opportunities
0..*
+FK_Feedback_Student
0..*
+FK_Feedback_Opportunities
0..*
Interactions
«column»
*PK id: integer
*FK student_id: integer
*FK opportunity_id: integer
action: varchar(20)
«FK»
+ FK_Interactions_Opportunities(integer)
+ FK_Interactions_Student(integer)
«PK»
+ PK_Interactions(integer)
Feedback
«column»
*PK id: integer
FK student_id: integer
FK opportunity_id: integer
type: varchar(50)
feedback_text: text
«FK»
+ FK_Feedback_Opportunities(integer)
+ FK_Feedback_Student(integer)
«PK»
+ PK_Feedback(integer)

Student:
A student represents an end-user of the Discord bot who receives opportunity recommendations.
A single student can receive many notifications, interact with many opportunities and leave multiple feedback items.

| Column name | Data type | Not null | Description |
|---|---|---|---|
| id | integer | true | Primary key of the Student table. |
| name | varchar | | Display name of the student. |
| email | varchar | | Unique e-mail address; used for login and notifications. |
| skills | text | | Free-text list of technical or soft skills. |
| career_interest | text | | Fields or roles the student is interested in. |
| discord_id | integer | | Discord user-ID used by the bot. |
| cv_url | text | | URL of the résumé/CV stored in an external object store. |
| job_type | text | | Preferred employment type |

Company:
A company is an organization that posts opportunities and receives student applications.
A company can have many opportunities and send many notifications to students via discord.

| Column name | Data type | Not null | Description |
|---|---|---|---|
| id | integer | true | Primary key of the Company table. |
| company_name | text | true | Legal or brand name of the company. |
| location | text | | Headquarters or primary office location. |

Opportunities:
An opportunity describes an internship, part-time or full-time role offered by a company. One opportunity may generate many notifications and interactions, and receive multiple feedback entries from students.

| Column name | Data type | Not null | Description |
|---|---|---|---|
| id | integer | true | Primary key of the Opportunities table. |
| title | varchar | true | Role or topic title. |
| company_id | integer | true | Reference to the offering company. |
| duration | varchar | | Expected duration |
| tech_stack | text | | Technologies or tools required. |
| description | text | | Detailed opportunity description. |
| job_type | text | | Type of engagement(internship, full-time…) |
| application_deadline | date | | Last date the bot should forward student applications. |

Notifications:
A notification is a message the bot sends to a student about an opportunity or company update.
Each notification references exactly one student and, optionally, one company.

| Column name | Data type | Not null | Description |
|---|---|---|---|
| id | integer | true | Primary key of the Company table. |
| student_id | integer | true | Recipient of the notification. |
| company_id | integer | | (Optional) company the notification refers to. |
| message | text | true | Text body sent via Discord DM. |

Interactions:
An interaction records a student's explicit action on an opportunity (e.g. bookmark, view, decline).
A single interaction links one student to one opportunity; over time a student may create many interactions.

| Column name | Data type | Not null | Description |
|---|---|---|---|
| id | integer | true | Primary key of the Interactions table. |
| student_id | integer | true | Student who performed the action. |
| opportunity_id | integer | true | Opportunity the action relates to. |
| action | varchar | | Enumerated string such as viewed, bookmarked, applied. |

Feedback:
Feedback captures qualitative or quantitative comments a student leaves about an opportunity, or discord bot itself.
Feedback helps the recommendation engine learn which roles resonate with students, thus improving future matches, and any error, typo or confusing wording in the bot's DM can be detected, analyzed and corrected in the next deployment.

| Column name | Data type | Not null | Description |
|---|---|---|---|
| id | integer | true | Primary key of the Feedback table. |
| student_id | integer | true | Author of the feedback. |
| opportunity_id | integer | true | Opportunity being evaluated. |
| type | varchar | | Tag such as positive, negative or neutral. |
| feedback_text | text | | Full free-text feedback. |

## 3. Tech Stack

### 1) What is Tech Stack?

The tech stack refers to the collection of technologies used in developing an application. This includes programming languages, frameworks, databases, front-end and back-end tools, and APIs. The selection of a tech stack plays a crucial role in shaping the application's architecture, influencing integration capabilities, performance, scalability, and the expertise required for development and maintenance.

### 2) Specified Programming Languages and Frameworks:

The application is built using a combination of modern programming languages and robust frameworks to ensure scalability, maintainability, and performance. Below is a list of the core technologies used:

- **Backend**:
  - **Language**: Java
- **Database**:
  - **System**: PostgreSQL

These technologies were selected based on their strong community support, proven reliability, and compatibility with the project's architecture and goals.

## 3) Summary of Major Technologies Used:

This section outlines all primary technologies employed in the development and operation of the application, covering front-end, back-end, database, DevOps, and other essential components. Each category plays a critical role in the system's performance, scalability, and maintainability.

- Backend

Server-side technologies responsible for business logic, API handling, data processing, and user authentication.

  - **Language**: The primary language is used to develop backend services.

Example: Python, Java

  - **API Protocol**: Defines how the frontend communicates with the backend.

Example: REST

- Database

Technology is used to store, retrieve, and manage application data.

  - **Primary Database:** The main system used for persistent data storage.

Example: PostgreSQL

- DevOps & Infrastructure

Tools and services that support deployment, scaling, monitoring, and automation of the application.

  - **CI/CD**: Tools that automate building, testing, and deploying code.

Example: GitHub, GitLab CI

  - **Containerization**: Technologies for packaging applications with their dependencies for consistent deployment.

Example: Docker

- Version Control & Collaboration

Tools used by development teams to manage code changes, track issues, and collaborate effectively.

  - **Version Control:** Systems for tracking changes to source code and enabling team collaboration.

Example: Git, GitLab

  - **Project Management**: Platforms for organizing tasks, planning sprints, and tracking progress.

Example: WIKI

  - **Communication:** Tools for team communication and coordination.

Example: GitHub, Microsoft Teams, Discord

## 4) Build, Testing, and Dependency Management Tools

To ensure a robust and maintainable development lifecycle, the following tools are used across the project:

- **Build Tool**:
  - **Gradle** – Used to automate the build process, manage tasks, and handle project configurations.
- **Testing Frameworks**:
  - **JUnit** – Employed for unit testing and validating backend logic.
  - **Cypress / Jest** – Used for frontend unit and end-to-end testing.
- **Dependency Management**:
  - **Gradle** – Manages project dependencies and library versions for the Java backend.
- **Containerization and Environment**:
  - **Docker** – Used to package and run the application in isolated, reproducible environments.

These tools streamline development, testing, and deployment processes, ensuring consistency and scalability throughout the application lifecycle.

## 5) Development Environment Setup:

This section outlines the tools and configurations required to set up a local development environment for the project.

- **Integrated Development Environment (IDE)**:
  - Developers are recommended to use **IntelliJ IDEA** for backend (Java + Spring Boot) and **Visual Studio Code** for frontend development (React). Both support essential plugins for code quality, syntax highlighting, and integration with version control systems.
- **Local Server Setup**:
  - The backend server runs locally using **Spring Boot's embedded Tomcat** server. Developers can start it via the. /gradlew bootRun command.
- **Containerization with Docker**:

    - **Docker** is used to run the application in isolated containers. A docker-compose.yml file is provided to start services like the backend, frontend, and PostgreSQL database with a single command (docker-compose up).
- **Environment Configuration**:
    - Environment variables are defined in .env files for local development and injected into the Docker environment as needed.

## 6) Deployment Tools and Environments:

This section outlines the tools and environments used to build, test, and deploy the application across development, staging, and production environments.

### 1. Containerization

- **Docker**:
  Docker is used to package the application and its dependencies into containers, ensuring consistency across local and production environments. A Docker file and docker-compose.yml are provided for local development and multi-service orchestration.
-

### 2. Continuous Integration / Continuous Deployment (CI/CD)

- **GitLab CI/CD**:
  Automated pipelines are configured using GitLab CI/CD to handle:
    - Code build and test execution
    - Static code analysis and linting
    - Docker image creation and pushing to a container registry
    - Automatic deployment to staging and production environments
    -

### 3. Hosting and Production Environment

- **Environment Management**:
  Environment variables and secrets are managed via GitLab CI/CD or a secrets manager

## 7) Integrated Libraries and APIs

The application incorporates several external libraries and APIs to extend functionality, streamline development, and enable third-party integrations.

### 1. Discord API

- **Purpose**: Enables real-time communication and automation within Discord servers.
- **Usage**: Used to interact with users, send messages, manage channels, and handle bot events for features such as notifications or opportunity alerts.

- **Integration**: Implemented via libraries like discord.js or REST API calls with OAuth2 for authentication.

**2. EXPERTS.AI API**

- **Purpose**: Provides natural language processing and AI-driven semantic analysis.
- **Usage**: Used to extract concepts, entities, and relationships from user inputs, improving data categorization and opportunity matching.
- **Integration**: Connected via RESTful API with authentication using API keys.

## 8) Full Technology Stack Overview:

| **Layer** | **Technology / Tool** | **Purpose** |
|---|---|---|
| **Frontend** | Discord | Building the user interface |
| **Backend** | Java | Server-side logic and API development |
| **Database** | PostgreSQL | Storing and managing application data |
| **API Integrations** | Discord API, EXPERTS.AI, GPT API | External communication and NLP processing |
| **Containerization** | Docker, Docker Compose | Isolated environments for development and deployment |
| **CI/CD** | GitLab CI/CD | Automating build, test, and deployment pipelines |
| **Version Control** | Git + GitLab | Source code management and collaboration |
| **Communication** | Discord | Team coordination and feature feedback |