FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# AI-Powered Career Opportunity Distribution System

## Architecture and Database - Tech Stack

## Realization

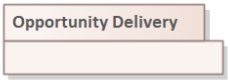
**Project documentation for the purpose of BIE-SWI course**

# 1. Realization Model:

*The realization model explains how the most critical Use Cases of the AI-Powered Career Opportunity Distribution System are implemented through the system's logic and structure. It integrates two perspectives to give a complete picture of each implementation. The dynamic view uses sequence diagrams to illustrate the interactions between system components over time, tracking the flow across the Presentation layer (Discord bot), the Business layer (AI Agent and Matching Engine), and the Data layer (PostgreSQL). These diagrams show how objects and actors communicate through method calls to fulfill the Use Case. Complementing this, the static view presents class diagrams that define the system's structural foundation, laying out the classes, interfaces, and relationships that make the dynamic interactions possible. Together, these views reveal how the system operates both in motion and at rest, connecting behavior with architecture to show exactly how each Use Case is realized within the application.*

In this chapter we document three core use cases of the system, illustrating how the classes of the Discord-based application collaborate:





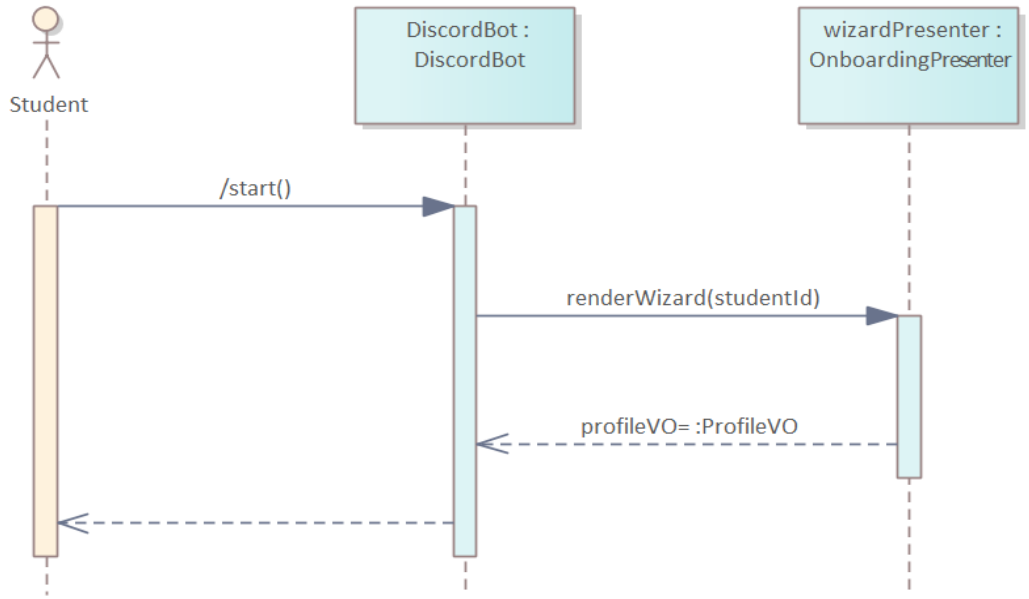

## 1.1 Profile Creation:

This section describes the realization of the UC to create a profile via a private Discord DM wizard.

### 1.1.1 Communication Model:

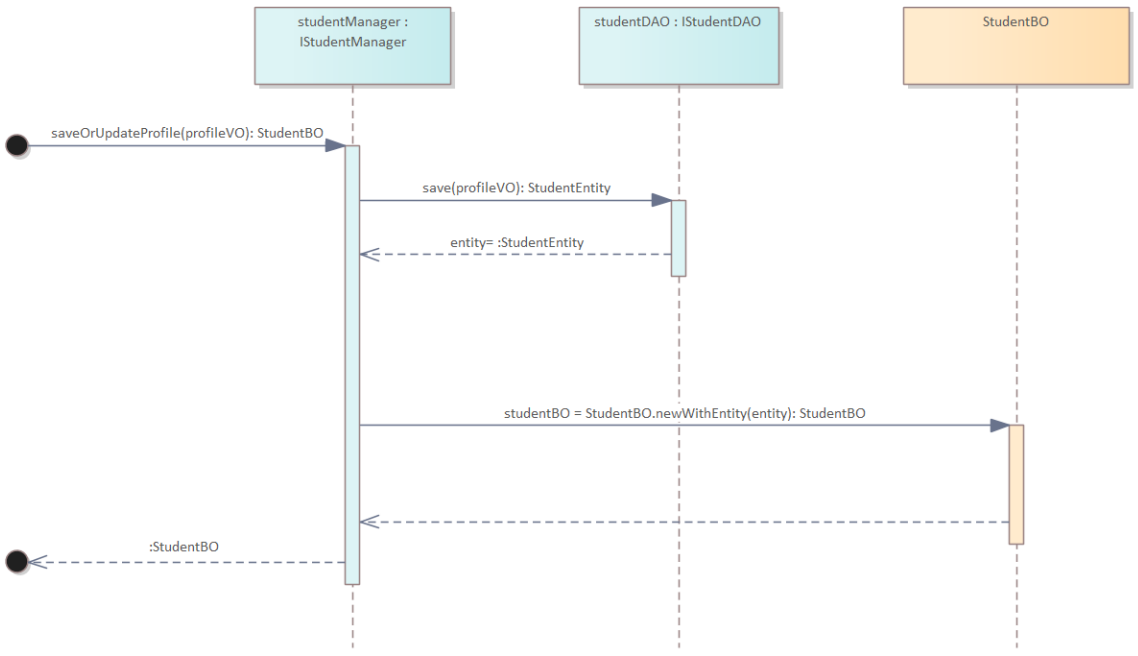
In this section, the interaction of individual classes and their instances realizing the behaviour is shown.

**Presentation Layer:**



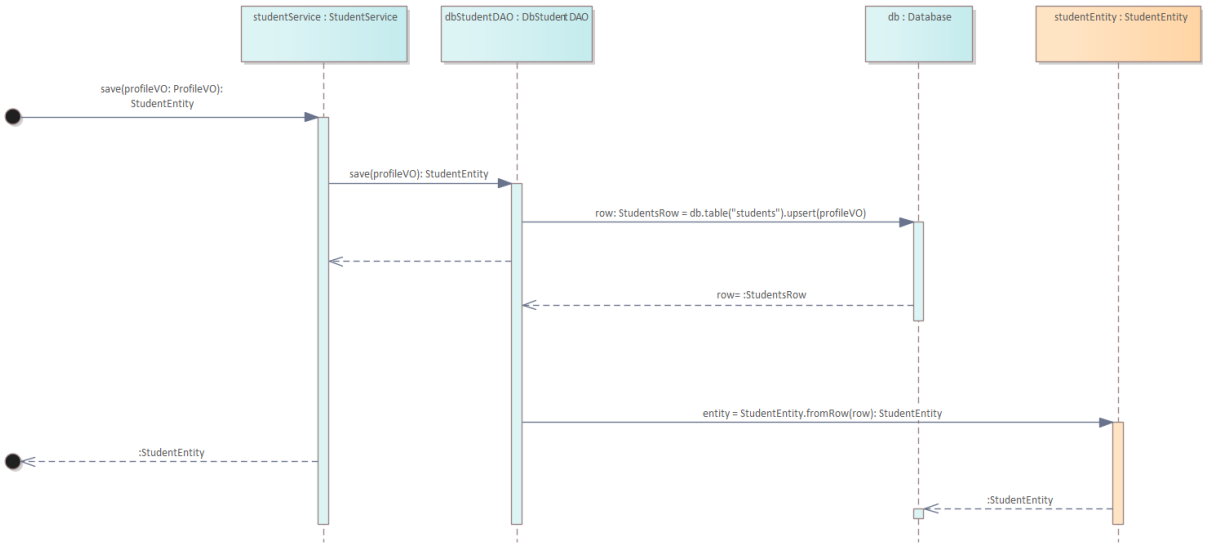

When a student types /start in any Discord channel, the DiscordBot handles the slash-command event and calls renderWizard(studentId) on the OnboardingPresenter, which opens a private DM and sequentially prompts the user for their full name, key skills, career interests and CV upload; once the CV file is attached, the presenter packages all responses into a ProfileVO and returns it to the bot, which then delivers a richly formatted embed summarizing the collected profile if student wants to choose to view the profile.

**Business Layer:**



When saveOrUpdateProfile(profileVO) arrives from the PL, the StudentManager calls StudentDAO.save(profileVO) to insert or update the student record in the database. It then takes the returned persistence entity and hands it to StudentBO.newWithEntity(entity), which builds a fully initialized business object, and finally returns that StudentBO back up to the Presentation Layer.

**Data Layer:**



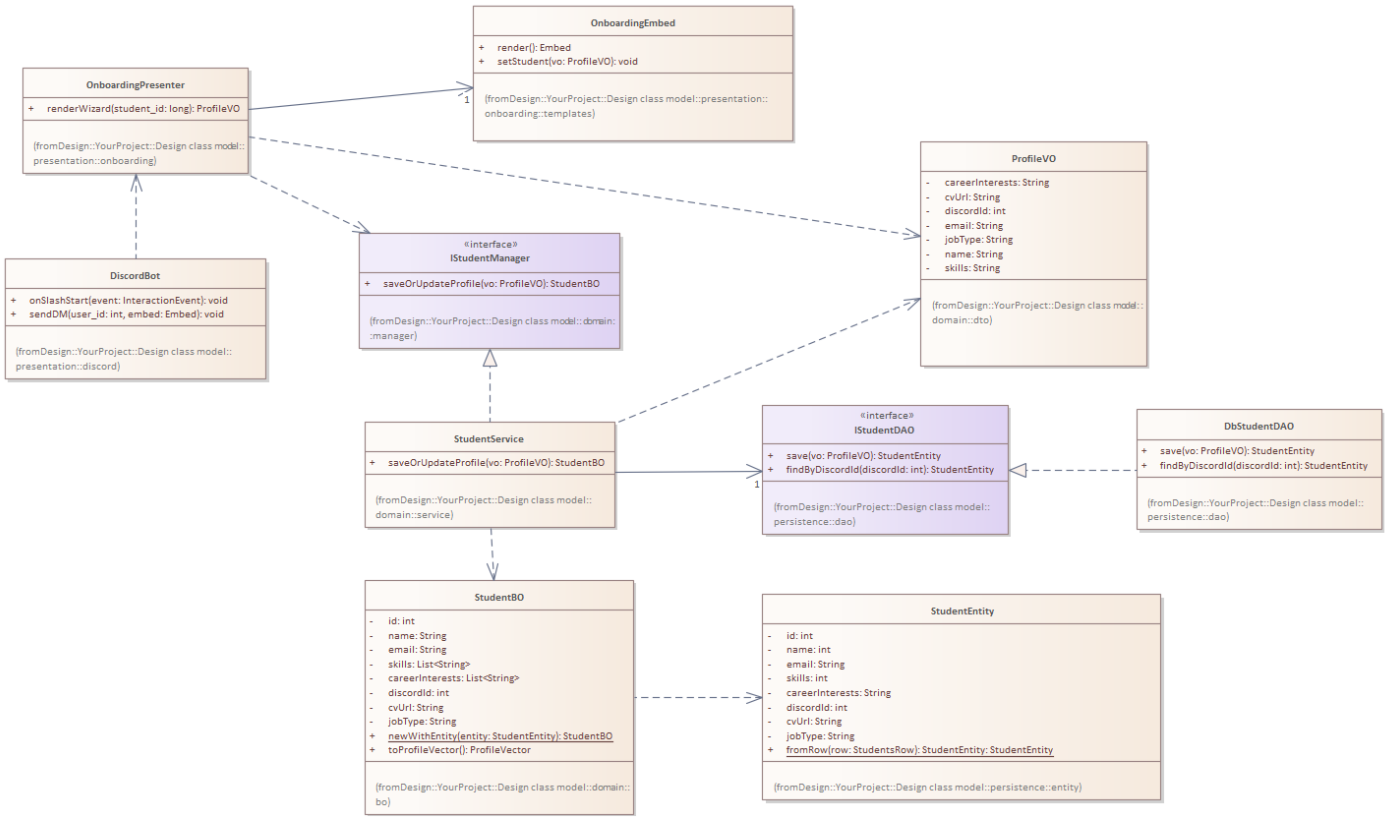

When save(profileVO) is invoked, DbStudentDAO opens a database transaction and runs an upsert on the students table using the data in ProfileVO (name, skills, career_interests, cv_url, discord_id). The database driver returns a StudentsRow object containing every column of the newly inserted or updated record (including its generated primary key and any default timestamps). The DAO then calls StudentEntity.fromRow(row), which reads each field out of that raw row (parsing text blobs into typed collections, converting timestamps, enforcing non-null constraints, etc.) and builds a fully populated StudentEntity. Finally, this entity, now carrying its database-assigned ID and cleaned data—is returned to the Business Layer.

#### 1.1.1.1 OnboardingPresenter:

An instance of the OnboardingPresenter, responsible for coordinating the Discord DM wizard: it sends prompts to the student, collects replies, assembles a ProfileVO, and hands it off to the Business Layer.

### 1.1.2 Class Model:

The classes realizing the Onboarding use case are shown below with their key attributes, operations and relations:



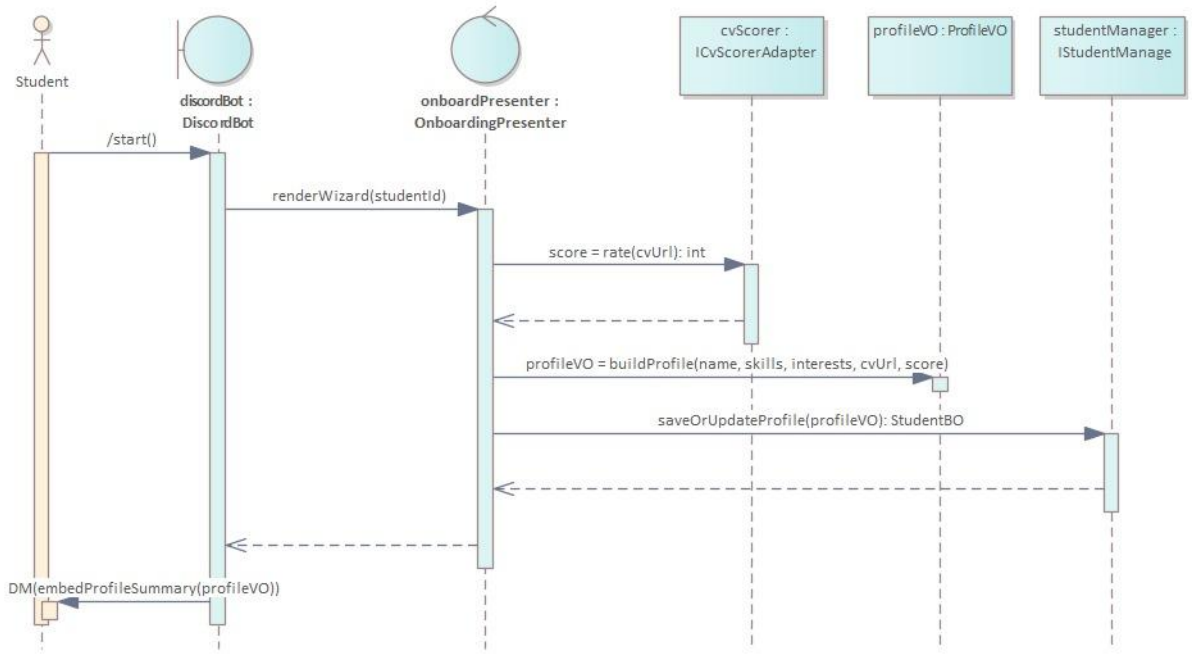

## 1.2 CV Rating:

This section describes the process of assigning a score to the student's CV using the external AI scorer.

### 1.2.1 Communication Model:

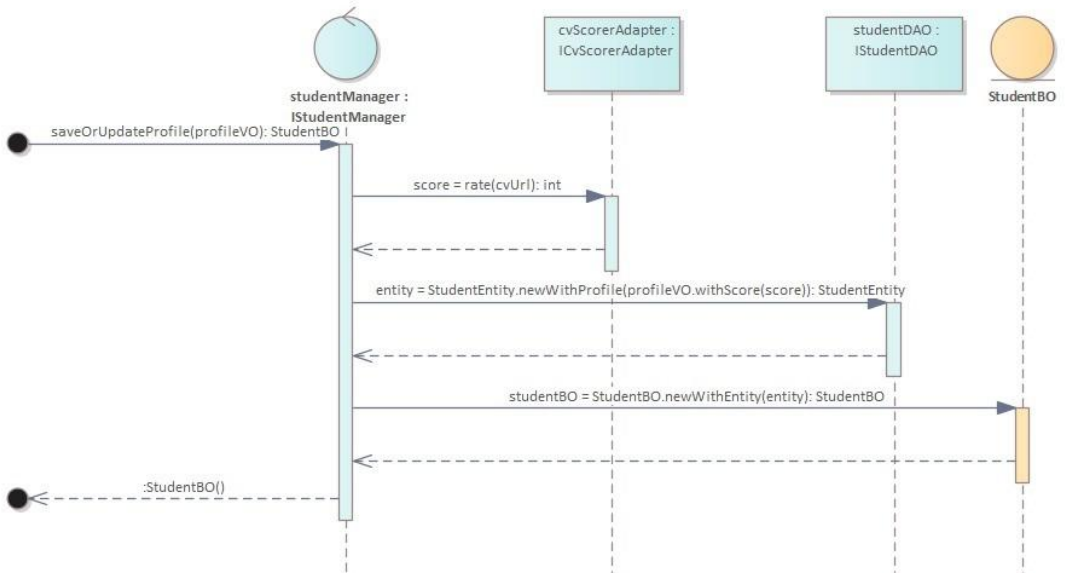
In this section, the interaction of individual classes and their instances realizing the behaviour is shown.

**Presentation Layer:**



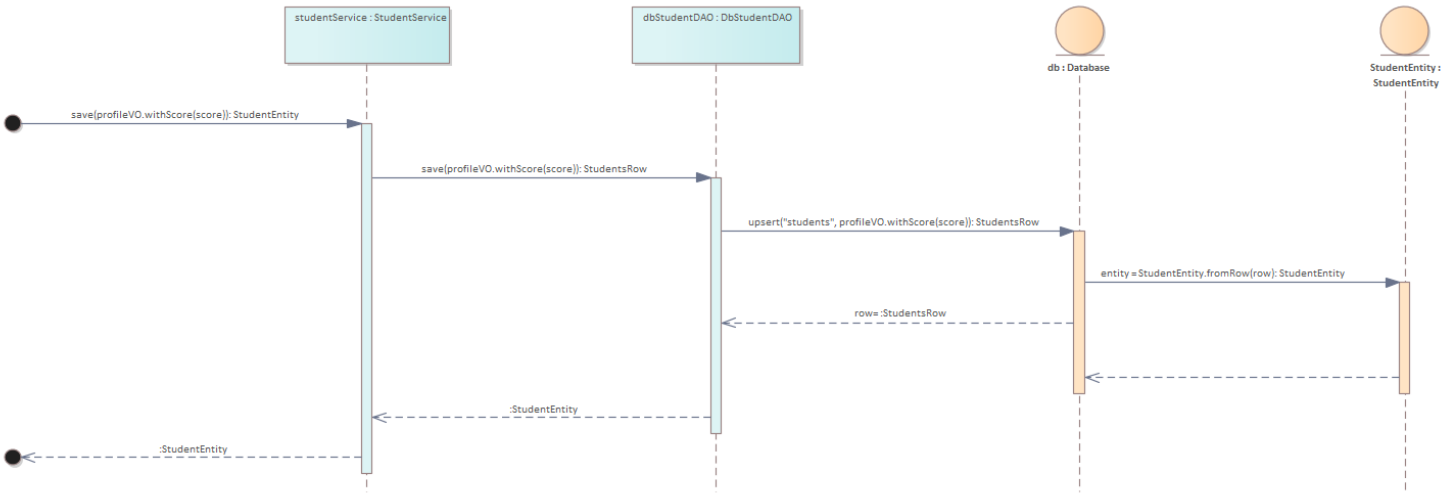

This collaboration diagram shows how the student's CV is scored during On-Boarding. When the student types /start(), the DiscordBot boundary forwards the slash-command to the OnboardingPresenter control via renderWizard(studentId). After collecting the CV URL, the presenter calls the ICvScorerAdapter interface (cvScorer) with score = rate(cvUrl): int. The adapter returns the numeric score back to the presenter (score = :int), which then builds a ProfileVO value object containing the name, skills, interests, CV URL, and the newly obtained CV score (profileVO = buildProfile(...)). Next, the presenter delegates persistence to IStudentManager (saveOrUpdateProfile(profileVO): StudentBO), which returns the saved StudentBO (studentBO = :StudentBO). Finally, the presenter returns the fully populated ProfileVO to the DiscordBot, which DMs the student a summary embed (DM(embedProfileSummary(profileVO))). This sequence makes explicit both the AI-driven scoring step and the hand-off of return values at each stage.

**Business Layer:**




The StudentManager receives saveOrUpdateProfile(profileVO) in BL. It first delegates to the ICvScorerAdapter to compute an integer CV score. Once the score returns, it creates a StudentEntity (merging the score into the profile) via the DAO, then converts that entity into a StudentBO using a static factory. Finally, it returns the populated StudentBO back to the Presentation Layer.

**Data Layer:**

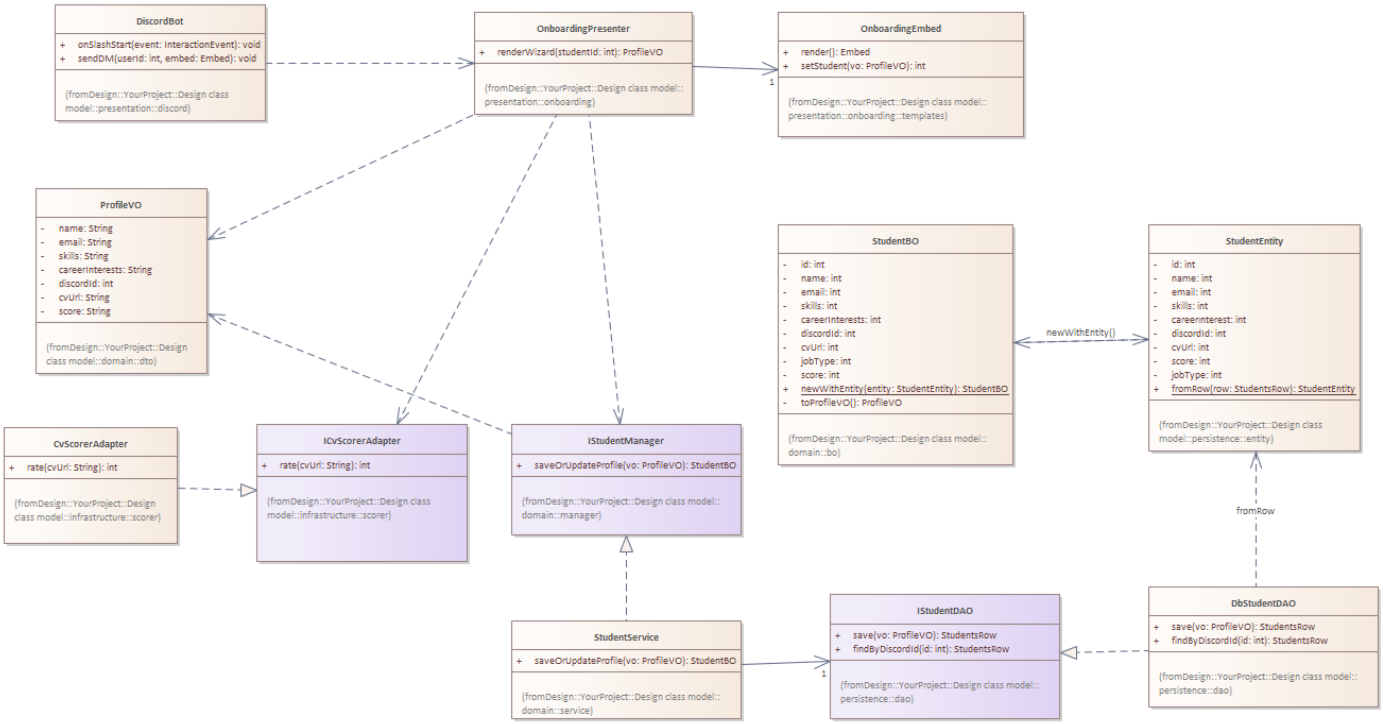


When save(profileVO.withScore(score)) is invoked, the DbStudentDAO begins a transaction and executes an UPSERT against the students table using the values from the ProfileVO (including name, skills, career interests, CV URL, Discord ID and computed score). The database driver returns a raw StudentsRow containing every column of the inserted or updated record (including any generated primary key and default timestamps). The DAO then calls StudentEntity.fromRow(row), which reads each field from the StudentsRow (converting text blobs into typed collections, parsing timestamps, etc.), applies any necessary validation rules, and constructs a fully populated StudentEntity. Finally, that StudentEntity—complete with its database-assigned primary key and cleaned data—is returned to the Business Layer.

### 1.2.2 Persistence Adapter:

The CV-scoring integration uses an external RESTful AI service. We encapsulate that behind an adapter interface.

### 1.2.3 Class Model:



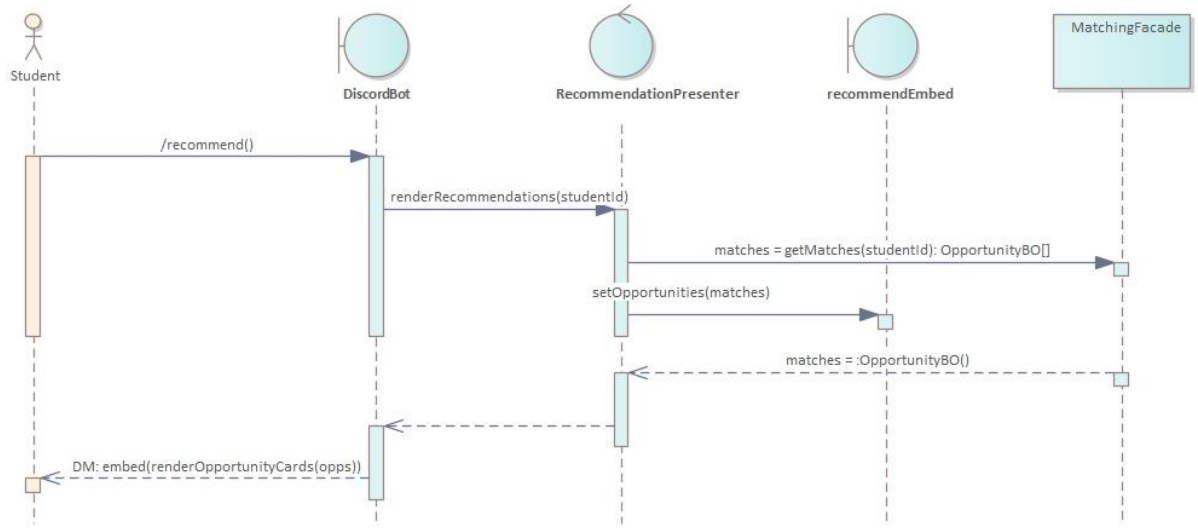
## 1.3 Opportunity Delivery:

This section describes the realization of the UC to deliver a personalized list of opportunity recommendations to the student via Discord. It shows the interaction between the Student actor, the DiscordBot boundary, the RecommendationPresenter control, the recommendEmbed boundary, and the MatchingFacade interface.

### 1.3.1 Communication Model:

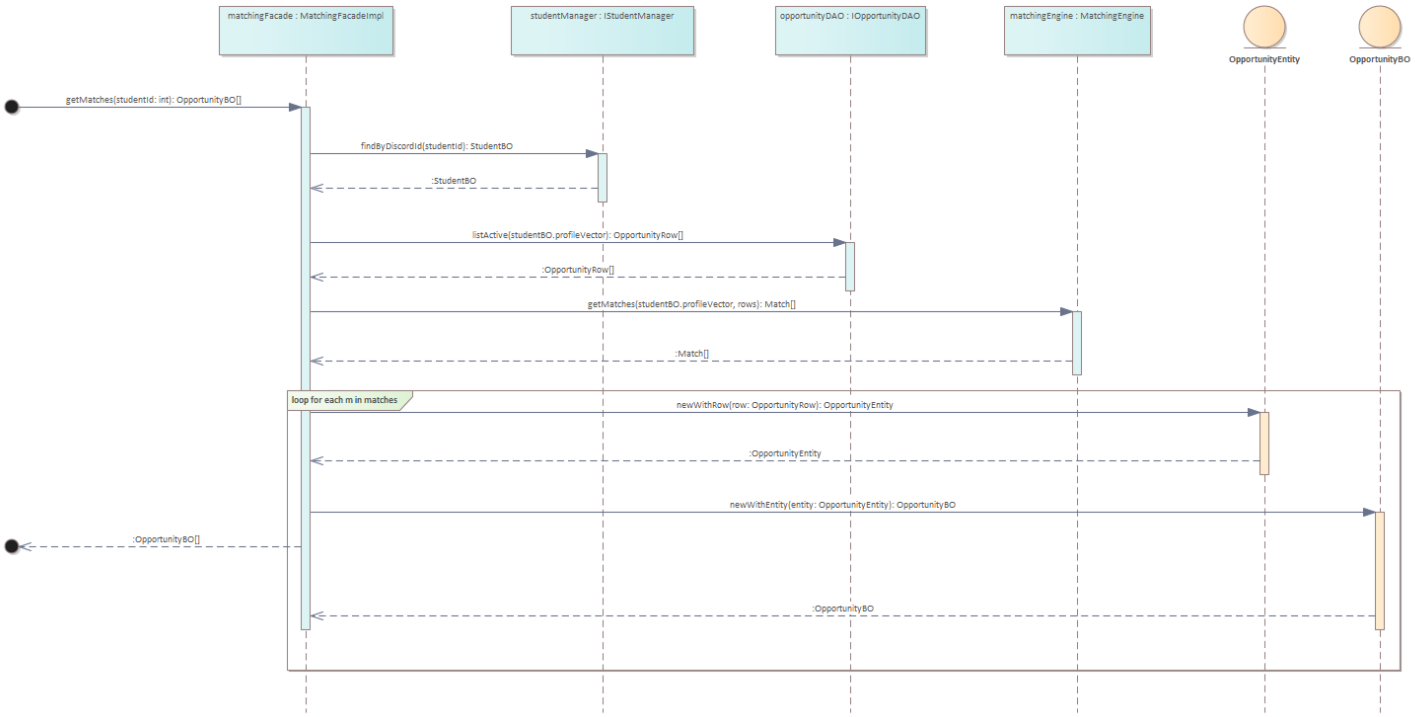
In this section, the interaction of individual classes and their instances realizing the Opportunity Delivery use case is shown.

**Presentation Later:**



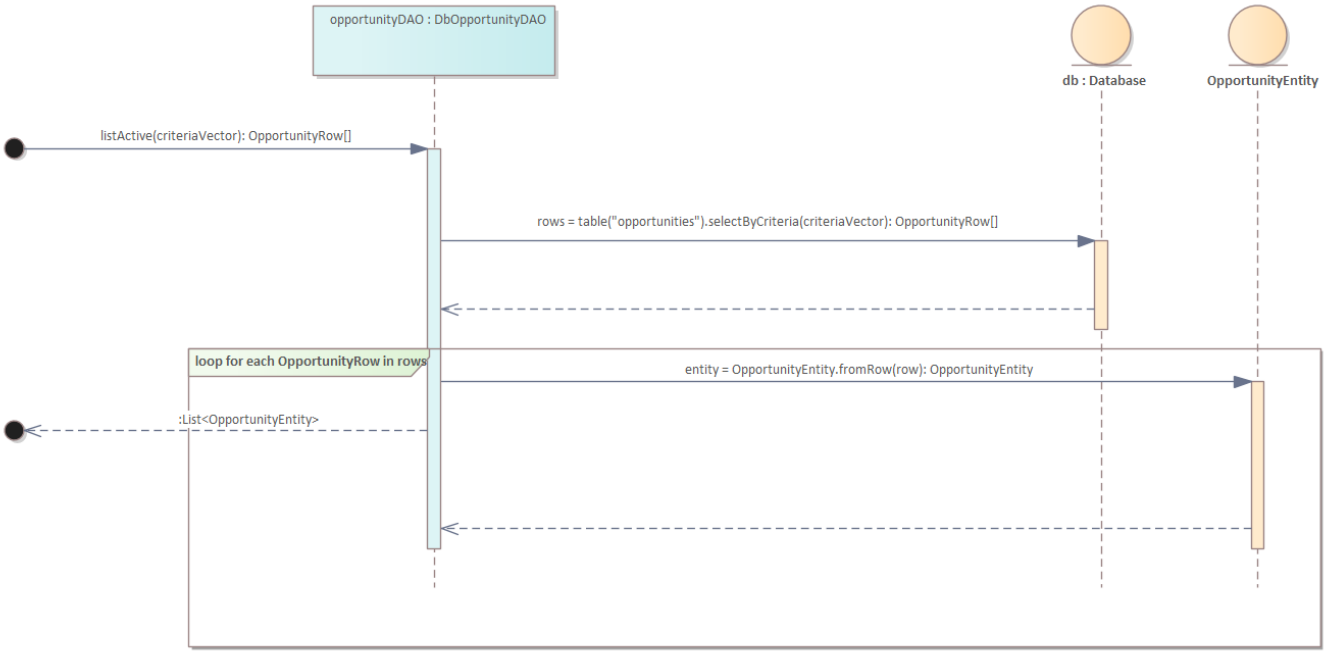

The realization at the PL level is shown in this diagram. The DiscordBot boundary calls renderWizard(studentId) on the OnboardingPresenter control after receiving the slash-command event when a student issues /start in any Discord channel. After launching a four-step wizard that asks for the presenter's full name, abilities, job interests, and CV upload, the presenter packages all of the responses into a single ProfileVO value object and sends it back to the bot. The profile card is then sent back to the student in a direct message after DiscordBot uses that ProfileVO to produce a summary embed using the Discord API.

**Business Layer:**




At the business-logic layer, the MatchingFacade begins by handling the getMatches(studentId) request. It first looks up the student's profile by calling IStudentManager.findByDiscordId(studentId), which returns a fully populated StudentBO. Next, it retrieves all currently active opportunities by invoking IOpportunityDAO.listActive(studentBO.getProfileVector()), yielding an array of raw OpportunityRow records. The facade then delegates to MatchingEngine.getMatches(profileVector, rows) to score and rank these raw rows according to the student's profile. Finally, it transforms each OpportunityRow into a domain entity with OpportunityEntity.fromRow(row) and then into a business object via OpportunityBO.newWithEntity(entity). Collecting all OpportunityBO instances into a list, the facade returns that array of recommendations back to the presentation layer.

**Data Layer:**




When listByCriteria(criteriaVector) is invoked, DbOpportunityDAO issues a query against the opportunities table using the given criteria and retrieves a raw array of OpportunityRow records. It then iterates over each OpportunityRow, converting it into a fully populated OpportunityEntity via the static factory method OpportunityEntity.fromRow(row). After transforming every record, the DAO returns the resulting List<OpportunityEntity> to the Business Layer.

### 1.3.2 Class Model: