3. Detailed design for the system components:

In this section we will dive into the detailed design for each component of the system, also mentions the design principles and patterns used to build a strong software system that fulfills its requirements.

Design Class Diagram:

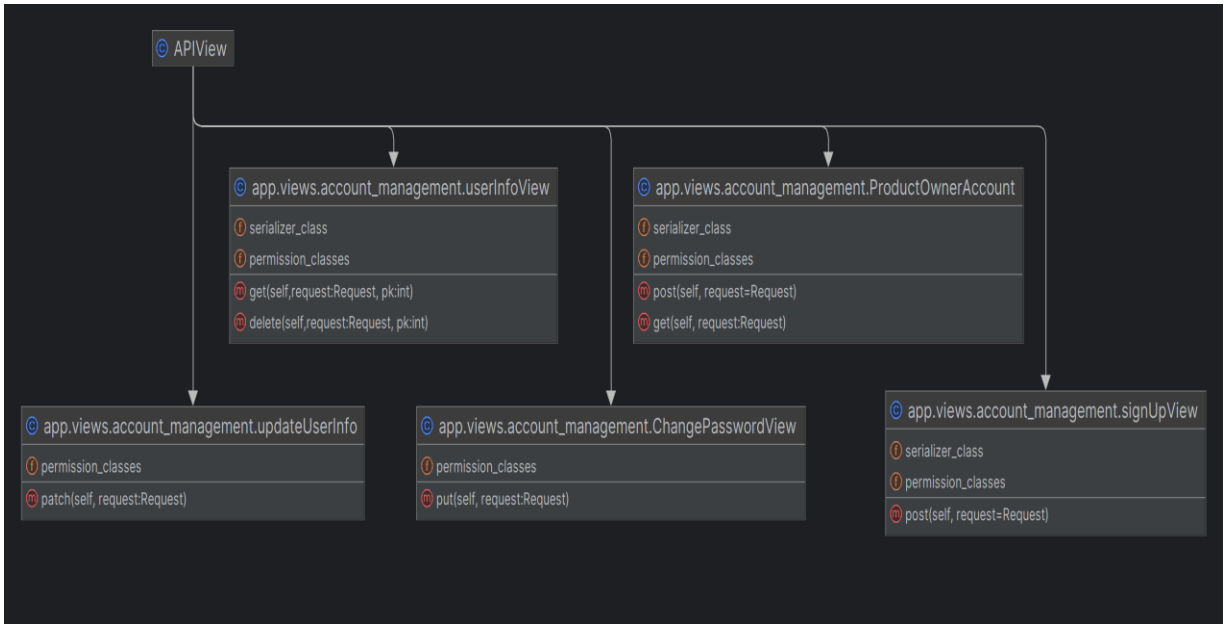


Figure 1 account management - controllers class diagram

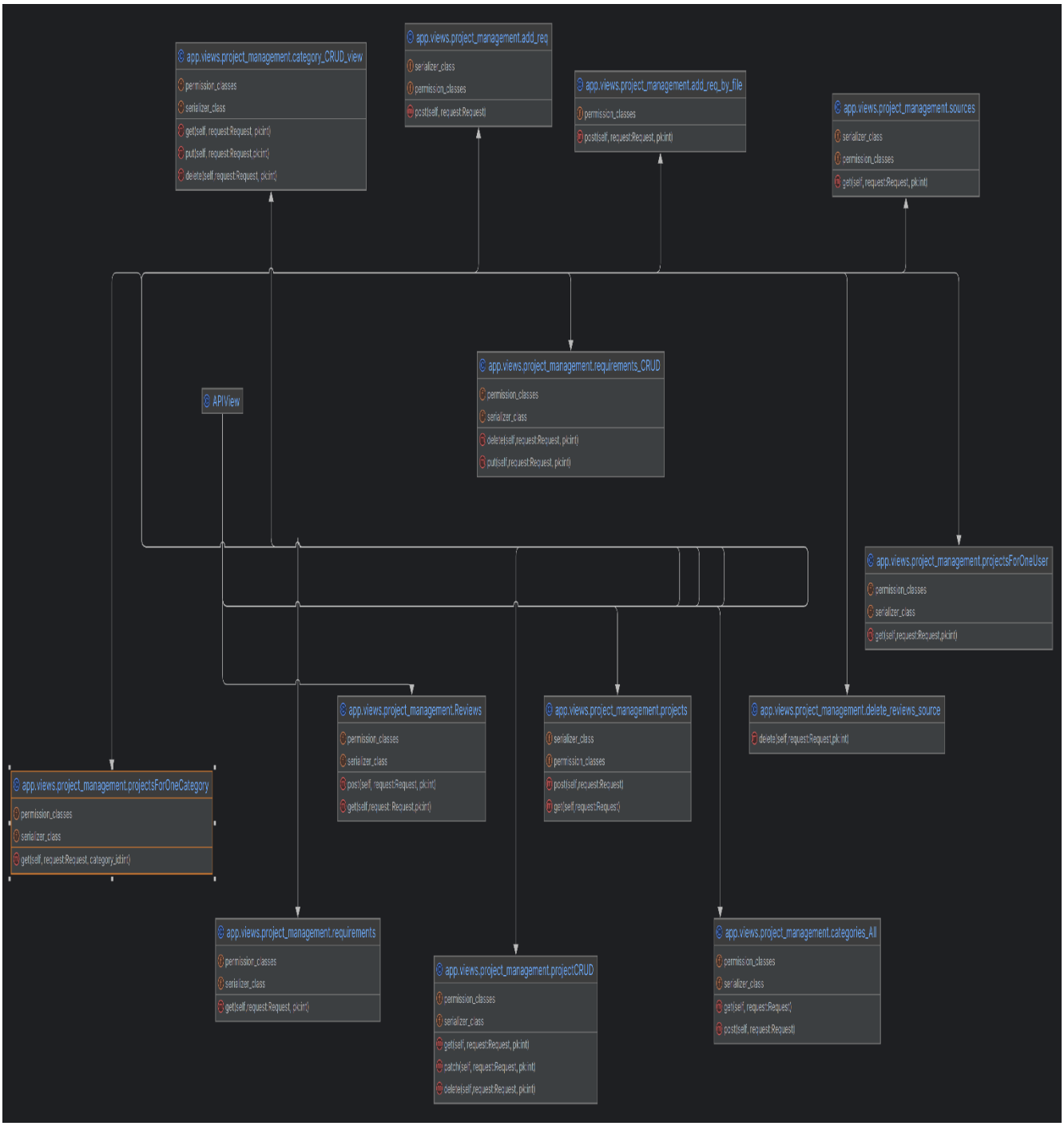


Figure 2 project management - controllers class

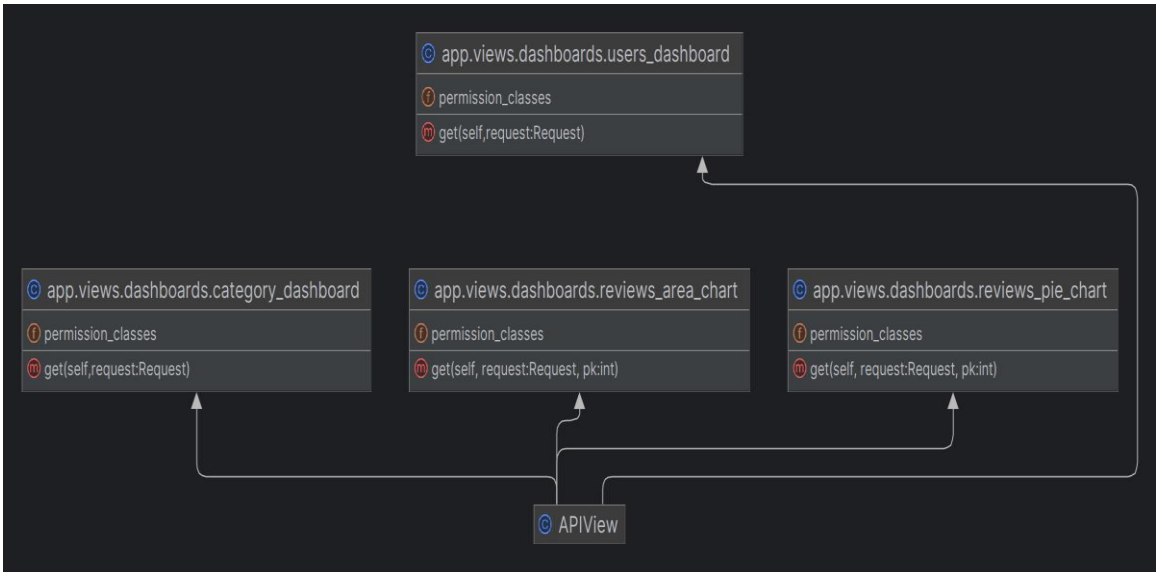


Figure 4 dashboards controllers - class diagram

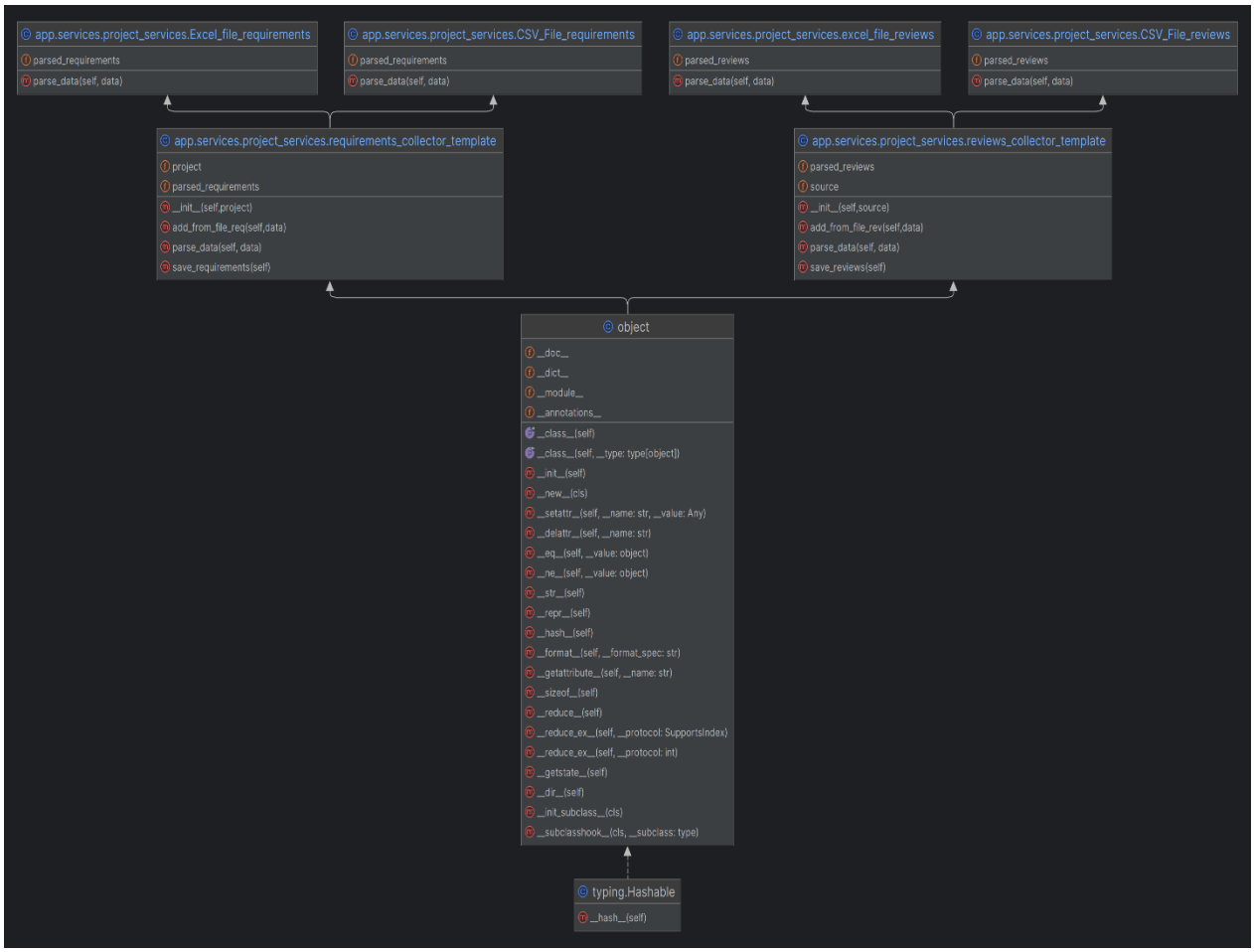


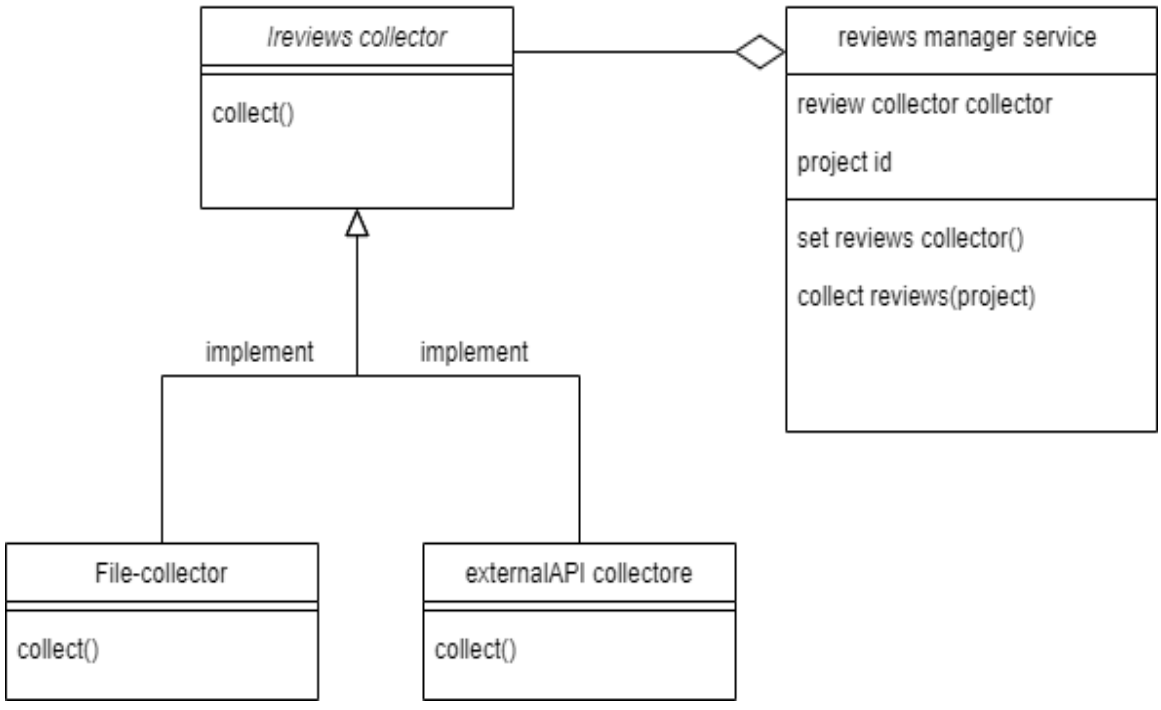
Figure 3 services project management - class

Used design patterns:

1. **Reviews collector component:**

   In this component, especially in the process of managing the reviews for a project we have several available ways to add or collect the project reviews, and also in the future, adding more ways will be possible, so we need a design pattern that enable us to add new algorithms, we will use the strategy design patter where the strategy is the collecting or entering process for the reviews.

   now we can choose the way we want to collect the reviews with for a project, also we can add more ways or logic with no code changes.



In the same component when the user can add the project requirements or reviews by uploading files either Excel or CSV files (different file formats), so the file parsing process will be similar between these files except for some specific steps. So, in this case it is suitable to use the Template method design pattern, to define the algorithm skeleton "files parsing", and allow concrete class to change specific parts of this algorithm.

Figure 5 reviews collecting - strategy

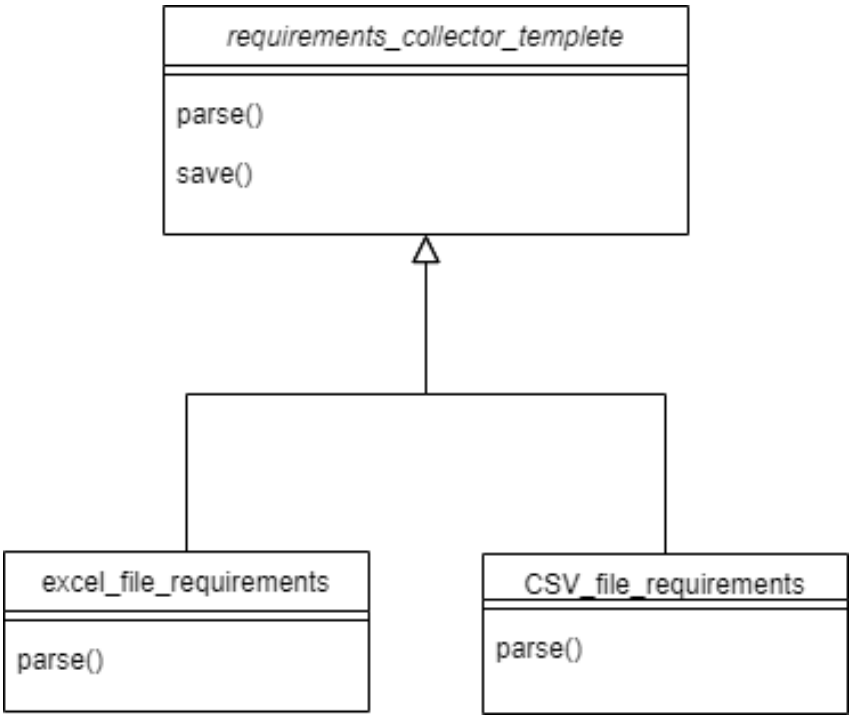


Figure 6 Template class

2. **AI Processing component:**

In this component, our goal is to implement design solutions to address the nonfunctional requirements defined in our system. One of the critical nonfunctional requirements for this component is extensibility. We want to build the component in a way that allows for future updates and upgrades.

To achieve this, we employ the design pattern called the **strategy pattern**. The strategy pattern allows us to encapsulate the AI services of sentiment analysis and semantic similarity within the NLP processor. By doing so, the client code, which requires these services, depends on an abstraction rather than the specific execution details.

By using the strategy pattern, we create a flexible and modular structure. New AI services can be added, or existing ones can be modified without impacting the client code. This approach enables us to seamlessly incorporate future updates and enhancements to the component without disrupting the overall system.

Also, by this solution, we implement the design principle **dependency injection** by introducing interfaces to break the dependencies between higher and lower-level classes.

Dependency injection is a programming technique that makes a class independent of its dependencies. It achieves that by decoupling the usage of an object from its creation.

This will be implemented for both the sentiment analysis functionality and the Semantic Similarity.

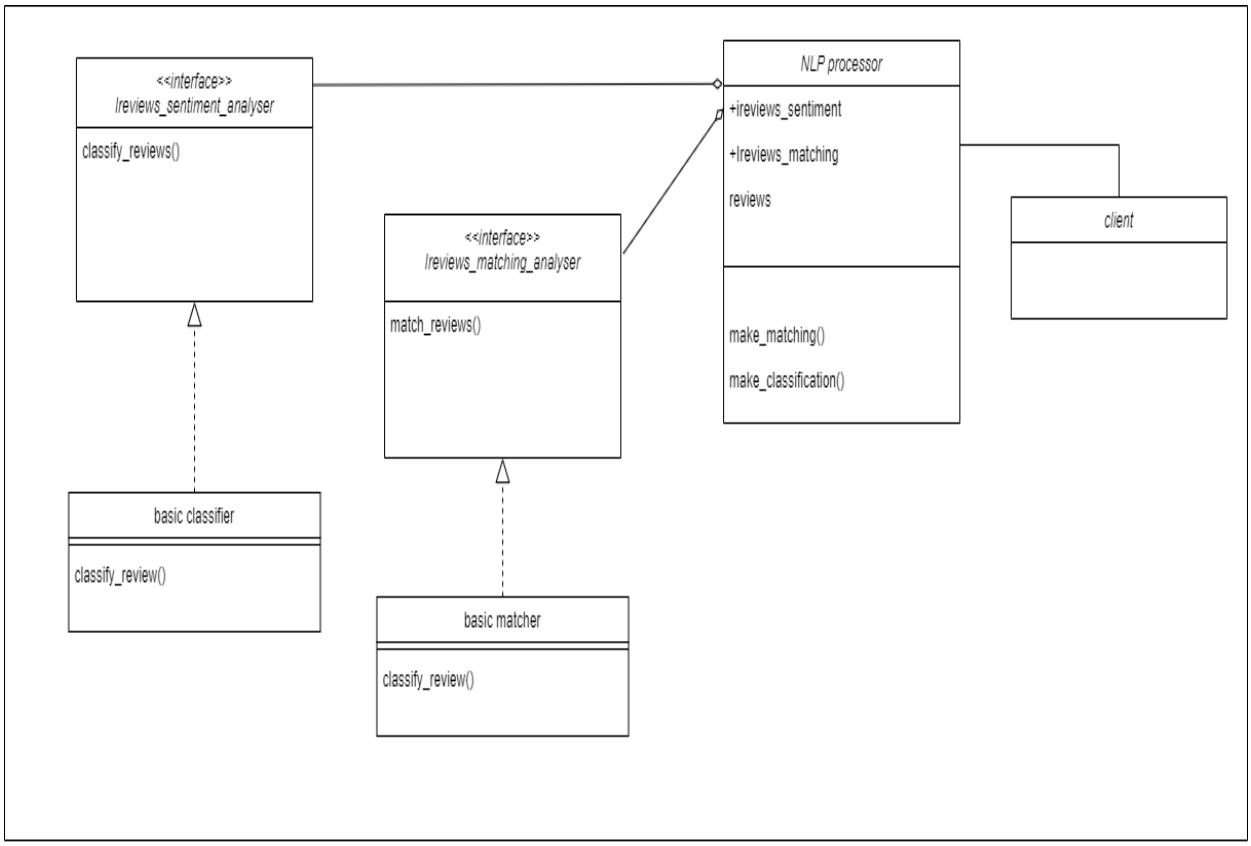


Figure 7 AI processor component design

4. Database Design:

NoSQL Database – MongoDB: For data related to user reviews and AI algorithm results, we employ MongoDB. This NoSQL database is well-suited for handling large volumes of unstructured data, providing the flexibility and scalability needed to manage diverse and dynamic datasets efficiently.

Review document structure:

Figure 8 reviews document structure

SQL Database – MySQL: In our system, MySQL is utilized for storing operational data related to accounts, projects, and requirements. This data is highly structured, making MySQL an ideal choice due to its efficiency in handling defined schemas and transactional integrity.

MySQL- database modeling:

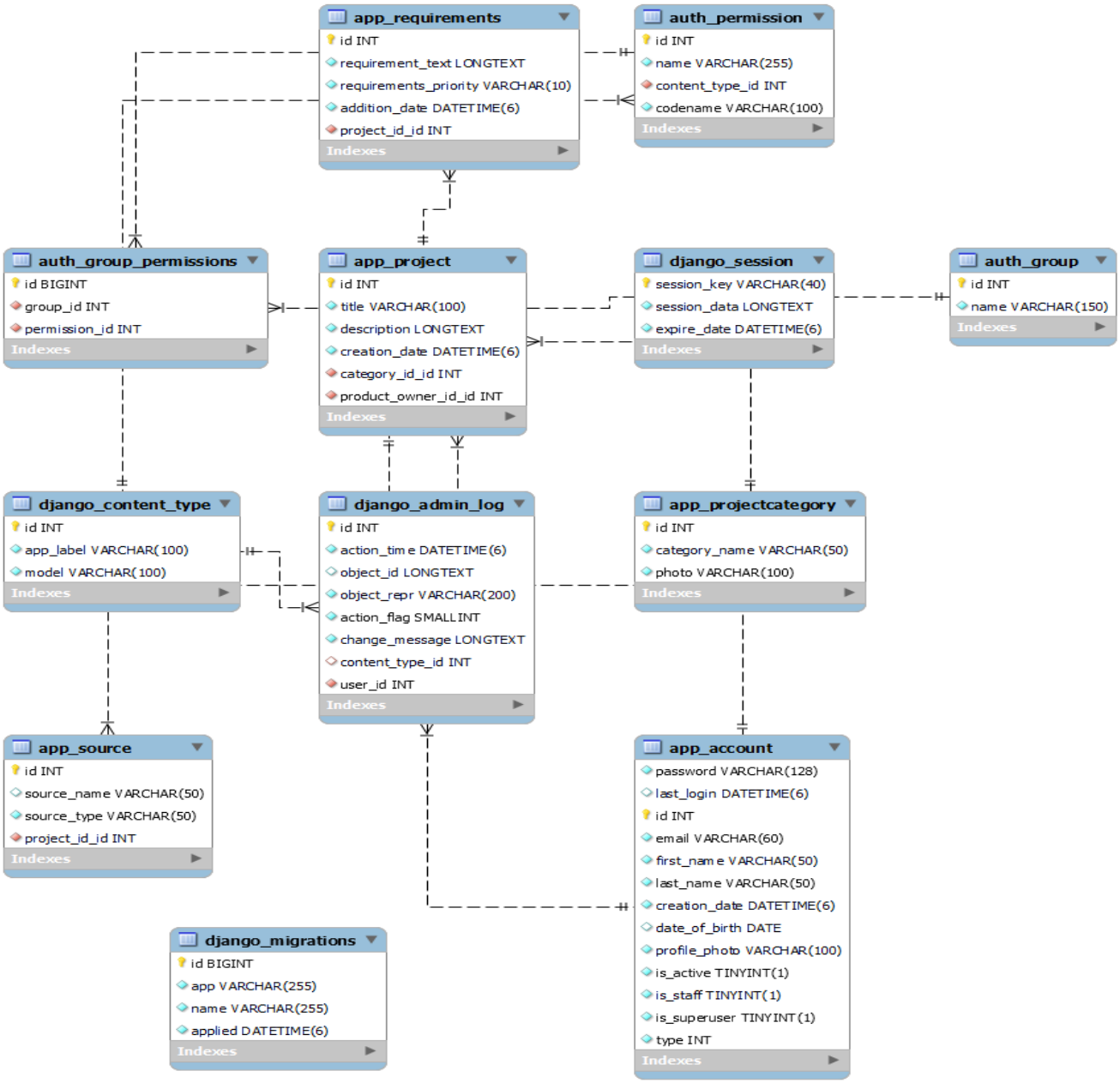


Figure 9 MySQL database modeling