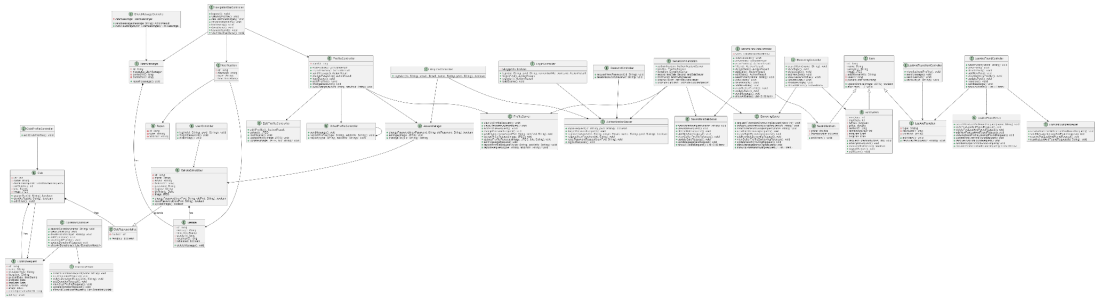
1. Class Diagram

A higher resolution of this diagram can be found under the name D5_classdiagram.png on the CampusConnect repository.



2. Design Patterns

2.1. Strategy Pattern

The infusion of the strategy pattern into the CampusConnect system signifies a pivotal design enhancement crafted to augment its flexibility and extensibility. This strategic integration empowers CampusConnect to dynamically modulate its behavior during runtime by encapsulating distinct algorithms or strategies within interchangeable components. Such a strategic approach facilitates the seamless adaptation of the system to diverse user authentication needs and evolving security requirements, all achieved without necessitating modifications to the core codebase.

To illustrate this concept further, consider the authentication process within CampusConnect. The strategy pattern becomes instrumental in managing various authentication methods, such as username/password and two-factor authentication. Each authentication method becomes encapsulated within a dedicated strategy class, offering the system the capability to dynamically switch between these strategies based on user preferences or evolving security standards. This nuanced approach ensures that CampusConnect remains adept at responding to changing authentication landscapes while adhering to the highest security standards.

The adoption of the strategy pattern for authentication yields a plethora of advantages, including heightened code reusability, maintainability, and scalability. Novel authentication methods seamlessly integrate into the system, and adjustments to existing methods can be executed without imposing disruptive alterations on the overall system architecture. In essence, the strategy pattern serves as a linchpin for future-proofing the CampusConnect system, ensuring it remains agile and adaptive in the face of evolving technological and security requirements.

2.2. Façade Pattern

In the intricate fabric of a full-stack web application like CampusConnect, the Façade pattern emerges as a key orchestrator, seamlessly harmonizing interactions between the front end and the sophisticated backend subsystems, including Java Spring Boot. The strategic integration of the Façade pattern introduces a refined simplicity, presenting a unified interface (facade) to the front end while expertly abstracting away the intricacies of underlying subsystems. This architectural finesse not only expedites development but also elevates the overall user experience. CampusConnect employs the Façade pattern in two strategic realms — firstly, at the web-server layer, where distinct structures known as services encapsulate the implementation of requests at each endpoint. This approach ensures that controllers experience minimal disruption even when there are changes in the implementation. Furthermore, the Façade pattern extends its influence to the database abstraction layer, utilizing repositories to abstract database communication. Repositories adeptly manage interactions with the database, isolating the logic from services. Consequently, changes in the database layer, encapsulated within repositories, exert no influence on the services layer. This bifocal implementation of the Façade pattern enhances not only code reusability, maintainability, and scalability but also fortifies the robust architecture of CampusConnect across its web-server and database realms.

2.3. Observer Pattern

The Observer pattern can be effectively utilized in the CampusConnect system to facilitate communication and synchronization between different components. Specifically, it can be employed to notify the frontend, developed using React, about any changes or updates occurring in the backend, implemented with Java Spring Boot. By implementing the Observer pattern, the system can ensure that relevant frontend components are promptly informed about new data or updates in the backend, enabling them to update their views accordingly. This pattern enhances the overall user experience by ensuring that the front end remains synchronized with the backend, providing real-time information and seamless interactions for users.