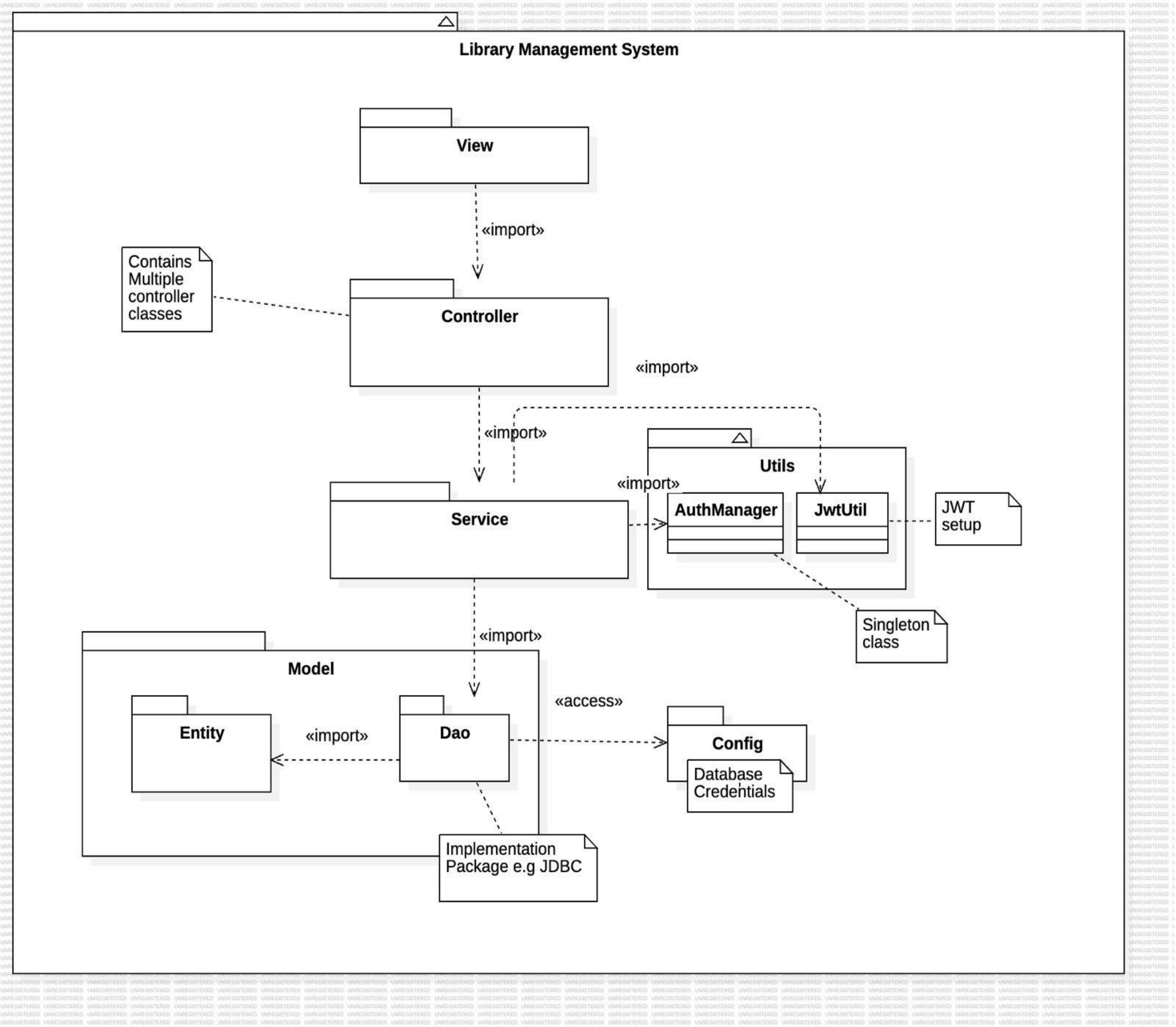
Exercise 11: Packages and Interfaces

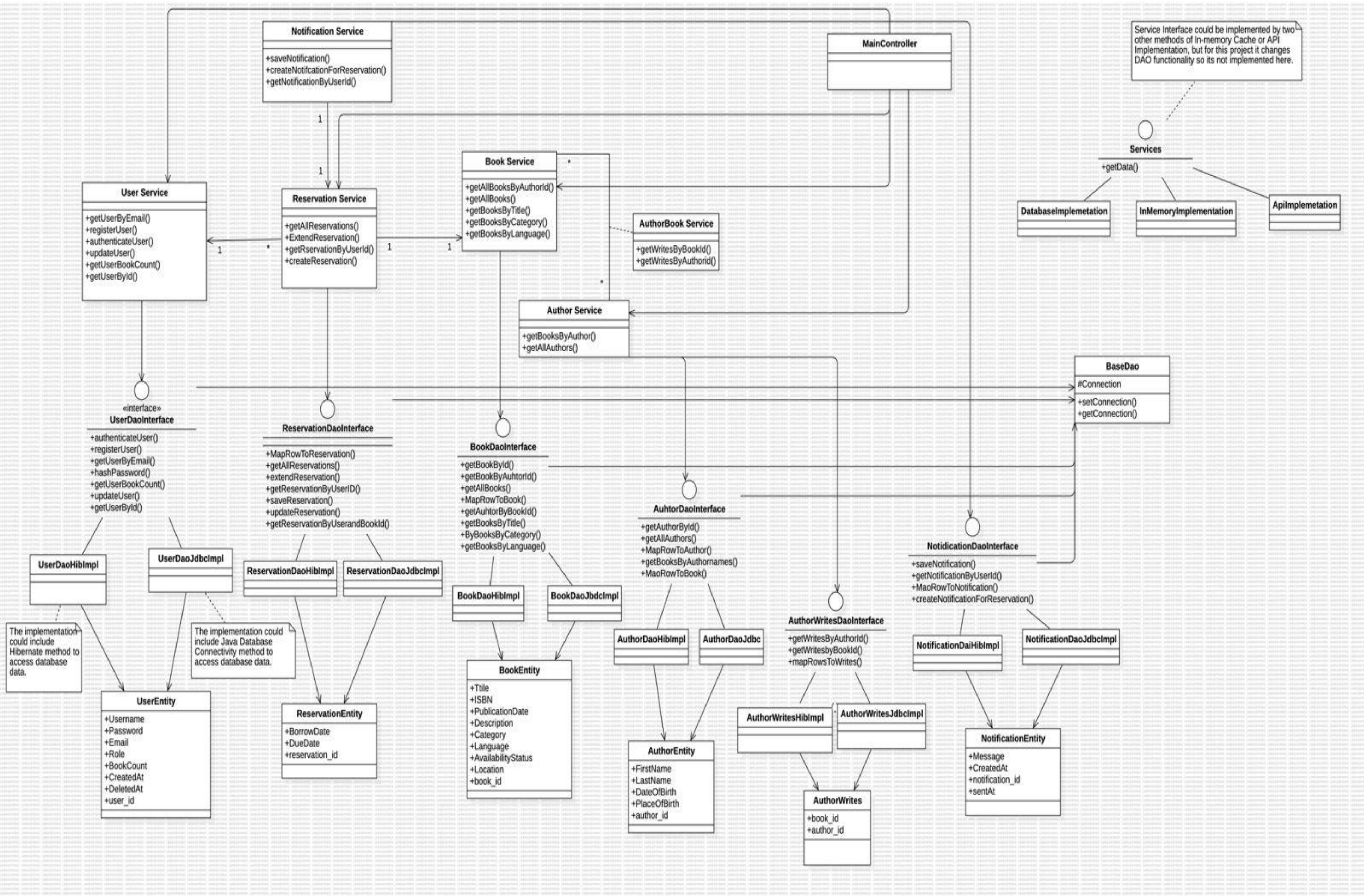
Group 5:

- **Mahnoor Fatima**
- **Sergei Vilka**
- **Trung Vu**

The package diagram:



The class diagram:



Service Interface could be implemented by two other methods of In-memory Cache or API Implementation, but for this project it changes DAO functionality so its not implemented here.

The implementation could include Hibernate method to access database data.

The implementation could include Java Database Connectivity method to access database data.

Overview

The updated class diagram and package organization reflect the latest vision for the Library Management System, ensuring modularity, flexibility, and scalability. Below are the structured updates with necessary interfaces and modularization details.

---

1. Package Organization

The system is divided into the following packages:

1.1 View Package

- Contains the user interface components.
- Imports and interacts with the Controller package.

1.2 Controller Package

- Contains multiple sub-controller classes responsible for handling user requests.
- Calls the Service Layer for processing logic.
- Ensures a clean collaboration between the UI and data calling logic.

1.3 Service Package

- Acts as an intermediary between Controllers and DAO.
- Implements easier data fetching and calls DAO functions.
- Allows for future changes without affecting other layers.

1.4 Model Package

- **Entity Package:** Contains all entity classes representing database tables.
- **DAO Package:** Implements database access functionality.

1.5 Config Package:

- **Database Config:** Manages database credentials and connection settings.

1.6 Utils Package

- Contains utility classes such as AuthManager (Singleton) and JwtUtil for authentication and authorization.
- Handles JWT setup for security.

---

2. Class Diagram Updates

The class diagram has been structured to ensure proper separation of concerns and flexibility for future enhancements.

2.1 Entity Layer

Contains the following entity classes, representing the database tables:

- UserEntity (username, password, role, bookCount, timestamps, etc.)

- ReservationEntity (borrowDate, dueDate, reservation_id)
- BookEntity (title, ISBN, description, publication date, status, availability, etc.)
- AuthorEntity (first name, last name, date of birth, etc.)
- AuthorWrites (relationship between books and authors)
- NotificationEntity (message, createdAt, sentAt)

These entities interact with the DAO layer for persistence.

2.2 DAO Layer

This layer handles all database interactions and includes:

- **BaseDao**: Manages database connections.
- **DAO Interfaces**: Define common methods for database operations.
- **DAO Implementations**:
  - UserDaoJdbcImpl, UserDaoHibImpl
  - ReservationDaoJdbcImpl, ReservationDaoHibImpl
  - BookDaoJdbcImpl, BookDaoHibImpl
  - AuthorDaoJdbcImpl, AuthorDaoHibImpl
  - NotificationDaoJdbcImpl, NotificationDaoHibImpl
  - AuthorWritesJdbcImpl, AuthorWritesHibImpl

Each DAO class includes SQL-based methods for CRUD operations, such as:

- authenticateUser(), registerUser(), getUserByEmail(), updateUser()
- getAllReservations(), extendReservation(), saveReservation()
- getBooksByTitle(), getBooksByCategory(), getBooksByLanguage()
- saveNotification(), getNotificationByUserId()

DAO implementations can later use **JDBC** and **Hibernate** for database interactions.

2.3 Service Layer

This layer provides business logic and interacts with the DAO layer. It includes:

- UserService
- ReservationService
- BookService
- AuthorService
- NotificationService

Each service class exposes methods used by controllers. For example:

- UserService provides authentication, registration, user retrieval, etc.

- BookService retrieves books by various parameters.
- NotificationService manages notifications for users.

Interfaces allow for future alternative implementations such as **in-memory cache** or **API-based services**.

2.4 Controller Layer

Controllers communicate with the View layer and process user requests via the Service layer. The main controller (MainController) delegates responsibilities to sub-controllers based on the page and also functionality:

- LoginController
- SignUpController
- CategoryPageController
- UserPageController
- AuthorPageController

Each controller:

- Handles specific requests from the UI.
- Calls relevant service methods.
- Returns responses in the expected format and changes in UI.

2.5 Authentication and Utility Classes

- AuthManager: Ensures user authentication.
- JwtUtil: Manages JWT-based authentication.

---

3. Modularization of Controller and DAO Classes

3.1 Controller Modularization

Controllers are modularized based on specific functionalities:

- LoginController handles user authentication, registration, and profile updates.
- SignUpController handles user registration.
- CategoryPageController handles all the pages that fetch books based on desired quality.
- UserPageController handles profile updates, user’s bookings, and logging out
- AuthorPageController handles author information, searches and books from that author.

This structure ensures:

- **Separation of concerns** (each controller focuses on a single responsibility).
- **Scalability** (new controllers can be added easily).

- **Maintainability** (code is easier to understand and modify).

3.2 DAO Modularization

DAO classes are divided into:

- **BaseDao**: Handles database connections.
- **SubDaos**: Handles specific queries for each type of entity.

Possible future implementations:

- **JDBC Implementations**: Use direct SQL queries.
- **Hibernate Implementations**: Use ORM for database operations.

This allows:

- **Reusability**: Common database operations are centralized in BaseDao.
- **Extensibility**: New DAO implementations can be added easily.
- **Flexibility**: DAO implementations can be swapped in the future.

Conclusion

The updated class diagram and package structure ensure:

- **Clear separation of concerns** between UI, controllers, services, and data access.
- **Extensibility and modularity** for future enhancements.
- **Scalability** through well-defined interfaces and flexible implementation choices.
- **Maintainability** by keeping the system organized and easy to update.

This design lays a strong foundation for the continued development of the Library Management System.