### Additional project report details

#### Project Overview

This project is a Web-based book management system developed using the Spring Boot framework. The front-end uses HTML and CSS to build the user interface, and the back-end uses Java for business logic processing and database interaction. The system is designed to provide an intuitive, easy-to-use interface that enables users to view, search, add, and manage book information efficiently. The system's responsive design ensures compatibility and user experience on different devices.

#### system structure

- \*\*Front-end\*\*: Use HTML5 and CSS3, combined with a small amount of JavaScript to achieve dynamic interaction. The front-end interface includes form input, book list display, and editing and deletion of book information. Use AJAX technology to achieve refresh-free data interaction with the backend to improve user experience.

- \*\*Backend\*\*: The backend architecture is based on Spring Boot and adopts RESTful API design principles to ensure the modularity and scalability of the system. Backend processing logic includes request parsing, data processing, and sending responses. Use Spring Data JPA or MyBatis to handle database interactions to improve the efficiency and security of data processing.

- \*\*Database\*\*: Using MySQL database, a database schema for book management was designed, including the definition of tables and relationships. The database table structure is designed to store information such as the book's ISBN, title, author, year of publication, etc., taking into account query efficiency and data integrity.

#### The main function

- \*\*Book Query\*\*: Through the search form provided on the front end, users can query books based on multiple conditions such as ISBN, title, author, etc. The backend handles these query requests through methods of the `BookService` class, using complex SQL queries or JPA methods to retrieve the database.

- \*\*Add Books\*\*: Users can add new books to the system by filling out a detailed form. After the form is submitted, the backend verifies the validity of the data and then stores it in the database. This process includes error handling mechanisms to ensure that user-entered data meets requirements.

- \*\*Book Management\*\*: The system provides complete book management functions, including updating and deleting books. Users can select corresponding operations in the book list, edit book information or delete books directly through the dynamically generated interface.

#### Database interaction

- \*\*Database configuration\*\*: Database connection information, including database URL, username and password, is configured in the `application.properties` file. Spring Boot uses this information to establish database connections, supports connection pool management, and improves system operating efficiency.

- \*\*Service layer\*\*: The `BookService` class encapsulates all book-related business logic on the backend and interacts with the `BookRepository` interface for data access. This layered design separates business logic from data access code, making maintenance and testing easier.

#### test

- \*\*Unit Test\*\*: Use JUnit for unit testing and test the business logic layer to ensure that each functional module runs as expected.

- \*\*Integration testing\*\*: Use Spring Boot's testing framework for integration testing, including `contextLoads()` testing to ensure that the application context is fully loaded, and `testGetBooksByMultipleConditions()` to verify the correctness of the book query function.

#### in conclusion

This project successfully leveraged modern technology stacks such as Spring Boot, HTML, CSS, and JavaScript to build a fully functional and user-friendly book management system. The system is designed and implemented with ease of use, efficiency and maintainability in mind, making it a reliable solution suitable for enterprise use. Ongoing maintenance and upgrades of the project are also ensured through detailed documentation and testing.

### Supplementary draft technical report

#### System Overview

This project is a Web-based book information query system that uses the Spring Boot framework to implement back-end logic, and the front-end implements user interaction through HTML and JavaScript. The system supports the query, modification and management functions of books, and has a good user interaction interface and efficient data processing capabilities.

#### Front-end implementation

- \*\*User Interface Design\*\*: The front-end page is designed with HTML5 and CSS3, combined with JavaScript and jQuery libraries to enhance page interactivity. The page mainly includes form input, book display form and modal window for modifying book information.

- \*\*Function Implementation\*\*:

- \*\*Query function\*\*: Users can query books by entering relevant information about the books (such as ISBN, title, author, etc.). The query form is sent to the backend via AJAX requests, and the query results are dynamically displayed on the page.

- \*\*Modification function\*\*: In the book query results, each row of data provides a "Modify" button. After the user clicks, a modal window will pop up, allowing the user to modify the book information and submit the modified data to rear end.

#### Backend implementation

- \*\*Spring Boot configuration\*\*: The backend uses the Spring Boot framework. The configuration file `application.properties` contains database connection information and other application configurations to ensure that the application can correctly connect to the MySQL database.

- \*\*Data processing\*\*: The SQL mapping of MyBatis is defined through the `BookMapper.xml` file, which effectively separates database operations from business logic and improves the maintainability and performance of the code.

#### Database Design

- \*\*Database interaction\*\*: Use the MyBatis framework for database operations, which improves the organization and execution efficiency of SQL statements. The database design takes into account various attributes of books, such as ISBN, title, year of publication, etc., making the data structure both concise and comprehensive.

#### Testing and Verification

- \*\*Automated testing\*\*: `DemoApplicationTests.java` provides automated testing code to ensure that the main functions of the application, such as book query and data loading, work properly. Use Spring Boot's testing framework to conduct integration testing to verify the integration and performance of application components.

#### Security and performance optimization

- \*\*Security Measures\*\*: Appropriate error handling and data validation are used in AJAX requests to ensure that front-end to back-end data transfer is safe and reliable.

- \*\*Performance Optimization\*\*: Reduce page reloading through asynchronous loading and dynamic content updates, improving user experience and application response speed.

#### in conclusion

This project shows how to build a full-featured book management system using modern web technologies and frameworks. The entire system not only provides powerful functions, but also focuses on user experience and application performance. It is a typical enterprise-level application example.

This report provides a detailed technical description based on the documentation and project structure information you provide. If you have more details or specific aspects that you would like to delve into, please feel free to let us know.

Project Overview

This project is a Web-based book management system developed using the Spring Boot framework. The front-end uses HTML and CSS to build the user interface, and the back-end uses Java for business logic processing and database interaction. The system is designed to provide an intuitive, easy-to-use interface that enables users to view, search, add, and manage book information efficiently. The system's responsive design ensures compatibility and user experience on different devices.

system structure

- \*\*Front-end\*\*: Use HTML5 and CSS3, combined with a small amount of JavaScript to achieve dynamic interaction. The front-end interface includes form input, book list display, and editing and deletion of book information. Use AJAX technology to achieve refresh-free data interaction with the backend to improve user experience.

- \*\*Backend\*\*: The backend architecture is based on Spring Boot and adopts RESTful API design principles to ensure the modularity and scalability of the system. Backend processing logic includes request parsing, data processing, and sending responses. Use Spring Data JPA or MyBatis to handle database interactions to improve the efficiency and security of data processing.

- \*\*Database\*\*: Using MySQL database, a database schema for book management was designed, including the definition of tables and relationships. The database table structure is designed to store information such as the book's ISBN, title, author, year of publication, etc., taking into account query efficiency and data integrity.

The main function

- \*\*Book Query\*\*: Through the search form provided on the front end, users can query books based on multiple conditions such as ISBN, title, author, etc. The backend handles these query requests through methods of the `BookService` class, using complex SQL queries or JPA methods to retrieve the database.

- \*\*Add Books\*\*: Users can add new books to the system by filling out a detailed form. After the form is submitted, the backend verifies the validity of the data and then stores it in the database. This process includes error handling mechanisms to ensure that user-entered data meets requirements.

- \*\*Book Management\*\*: The system provides complete book management functions, including updating and deleting books. Users can select corresponding operations in the book list, edit book information or delete books directly through the dynamically generated interface.

Database interaction

- \*\*Database configuration\*\*: Database connection information, including database URL, username and password, is configured in the `application.properties` file. Spring Boot uses this information to establish database connections, supports connection pool management, and improves system operating efficiency.

- \*\*Service layer\*\*: The `BookService` class encapsulates all book-related business logic on the backend and interacts with the `BookRepository` interface for data access. This layered design separates business logic from data access code, making maintenance and testing easier.

test

- \*\*Unit Test\*\*: Use JUnit for unit testing and test the business logic layer to ensure that each functional module runs as expected.

- \*\*Integration testing\*\*: Use Spring Boot's testing framework for integration testing, including `contextLoads()` testing to ensure that the application context is fully loaded, and `testGetBooksByMultipleConditions()` to verify the correctness of the book query function.

- \*\*Automated testing\*\*: `DemoApplicationTests.java` provides automated testing code to ensure that the main functions of the application, such as book query and data loading, work properly. Use Spring Boot's testing framework to conduct integration testing to verify the integration and performance of application components.

Security and performance optimization

- \*\*Security Measures\*\*: Appropriate error handling and data validation are used in AJAX requests to ensure that front-end to back-end data transfer is safe and reliable.

- \*\*Performance Optimization\*\*: Reduce page reloading through asynchronous loading and dynamic content updates, improving user experience and application response speed.