1.Main Components:

1. **Presentation Layer (Frontend):**
   - Technology: Thymeleaf (templating engine), HTML, CSS
   - Purpose: Displays dynamic content to the users (books, login, admin views, etc.). Thymeleaf allows server-side rendering of templates with ease of integration in Spring Boot applications.
   - Key Pages:
     - index.html: Displays the list of books (paginated view, loaded with async calls for infinite scrolling).
     - login.html: Admin login page.
     - add-book.html: Admin form for adding a new book.
2. **Application Layer (Backend/Business Logic):**
   - Technology: Spring Boot (with Spring MVC, Spring Data JPA, and Spring Security), Ehcache.
   - Purpose: Handles HTTP requests, implements business logic, communicates with the database, and manages security.
   - Components:
     - Controllers: Responsible for handling requests and responses (e.g., BookstoreController for book management).
     - Service Layer: Contains business logic (e.g., BookService for CRUD operations on books)
     - Ehcache - caching implementation that allows to have cached data and avoid calling the database every time when there is need to retrieve the whole list of the books.
     - Security: Admin authentication using Spring Security and custom AdminDetailsServiceImpl.
3. **Persistence Layer (Database):**
   - Technology: MySQL (from db4free.net)
   - Purpose: Manages the persistence of books and admin data.
   - Entities:
     - Book: Stores book details such as name, price, and date added.
     - Admin: Stores admin credentials and roles for access control.

Additional Components:

- **E-commerce Synchronization Module:**
  - Purpose: Synchronizes book data with an external e-commerce system.
  - Technologies: REST API integration, Spring Scheduling (for scheduled synchronization at midnight).

Architecture Reasoning:

- **Spring Boot:** It's a highly productive framework that simplifies the development of robust and scalable web applications. Spring Boot has an autoconfiguration, embedded server that is really helpful in building applications.
- **Thymeleaf:** Chosen for its tight integration with Spring Boot. Which allows us to use dynamic web pages that are easy to implement and use in development.
- **Spring Security**: Provides a simple, customizable solution for securing the application, handling admin authentication, and restricting access to admin features.
- **MySQL Database:** MySQL is a well-established and widely-used relational database management system (RDBMS). It's known for stability, consistency, and reliable transaction support.
- **Modularity**: The separation of concerns between the frontend, business logic, and persistence layers ensures that each component can be developed, maintained, and scaled independently.

2. Risks in the Chosen Solution

1. **Scalability**:
   - **Risk**: Although Spring Boot scales well for small to medium-sized applications, handling high traffic might become a challenge if the bookstore grows significantly.
   - **Solution**: Implement load balancing and consider microservices architecture for specific features (e.g., book search, recommendations).
2. **Security**:
   - **Risk**: The security implementation is primarily for admin login. If not properly configured, potential vulnerabilities such as injection attacks or insecure session management could arise.
   - **Solution**: To implement more security it would be good to have HTTPS instead of HTTP, enabling advanced authentication mechanisms (OAuth, JWT), and restricting sensitive data access, for example for e-commerce REST API should be enabled security with some role and authentication.
3. **Database Performance:**
   - **Risk**:
     i. As the database is implemented from free source db4free.net there is possibility that the database will work slower than mySQL that can be runned locally and db4free.net have also strict size to use and time limit in which database might be used(2 weeks). And as it is a free database there might be security issues.
     ii. If the dataset grows (e.g., thousands of books), performance issues could occur with frequent CRUD operations, especially during synchronization with the e-commerce platform.
   - **Solution**:
     i. Use databases that are provided by MySQL locally or to use some cloud databases that are more secure and do not have time limits that are not controlled by the owner.

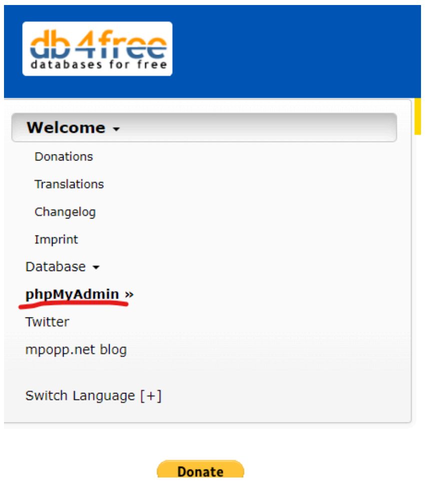
ii. Optimize queries, introduce indexing in the database, and use database replication/sharding if needed.

4. **E-commerce Integration:**
   - **Risk**: The external e-commerce system might change its API or experience downtime, affecting synchronization. For now the application does not implement a security layer to get REST api by e-commerce.
   - **Solution**: Implement a security layer that will help to control accessing REST API and bookstore data. Use circuit breakers (if API fails the system stops calling it) and fallback mechanisms(system can have cached prices from database or return some default message) to handle API failures gracefully. Log all synchronization attempts and retries.

3. Set up guide

- As Java is platform independent the project can be runned by any OS (tested on Windows).
- Download JDK 17 and install (check version java -version)
- Download Apache maven and install (check version maven - version)
- Clone and unzip the Project, go to the project directory
- Build the application with maven clean install
- Database:

  i. For some period (approximately two weeks), a database that is provided in the project by https://www.db4free.net will be available. To login go to phpMyAdmin, and use **username**: *bookstoreapp* and **password**: *bookstore*.



  ii. After this period you can create a new mySQL database and change application.properties files to provide new database datasource url, username and password. It is important to create table Book and table admin, using this sql statements:
  CREATE TABLE book (id SERIAL PRIMARY KEY, name VARCHAR(255) UNIQUE NOT NULL, price DECIMAL(10, 2), date_added TIMESTAMP);
  CREATE TABLE admin (id SERIAL PRIMARY KEY, username VARCHAR(255) UNIQUE NOT NULL, password VARCHAR(255) NOT

NULL, role VARCHAR(255) NOT NULL);

To use the admin/add page, it is required to create an admin with role admin and BCrypted password.

- Run the application on local environment java -jar target/bookstore.jar
- Access in Browser
    i. The application will be accessible at http://localhost:8080/
    ii. Endpoints:
        1. http://localhost:8080/v1/books (view books with pagination)
        2. http://localhost:8080/v1/admin/add (admin-only page for adding books)
        3. http://localhost:8080/v2/books (view books with price in json format to pass it to e-commerce)
        4. http://localhost:8080/v2/books/{name} (PUT method to update the price of the book, body of the book must be provided and name provided in the api should be the same as in the provided body in json format.
           Example: `{ "name": "Name", "price": 19.99}`

Docker image can be accessed after login in docker hub, by next commands:

```
docker pull alijajeniceka/bookstore:latest
docker run -d -p 8080:8080 alijajeniceka/bookstore:latest
```