Schrodinger's Book Club

Jerry Klos 19200313

Simon Farrelly 19739425

Noemi Banal 19413292

Synopsis:

The system created for this project is a distributed book recommendation system. The application domain is books and literature, and the primary goal of the application is to help users discover new books that match their preferences. In order to achieve this, the system will provide users with book recommendations through two methods, which are implemented as two separate services: an interactive quiz and a feature that allows the user to read short book extracts without knowing which book they belong to. In addition to these, the system includes a database server containing the book information and a front-end service that will display the recommendations to the user. The user will be able to create an account to keep track of their book recommendations.

Technology Stack

The technology stack used to implement Schrodinger's Book Club is presented below:

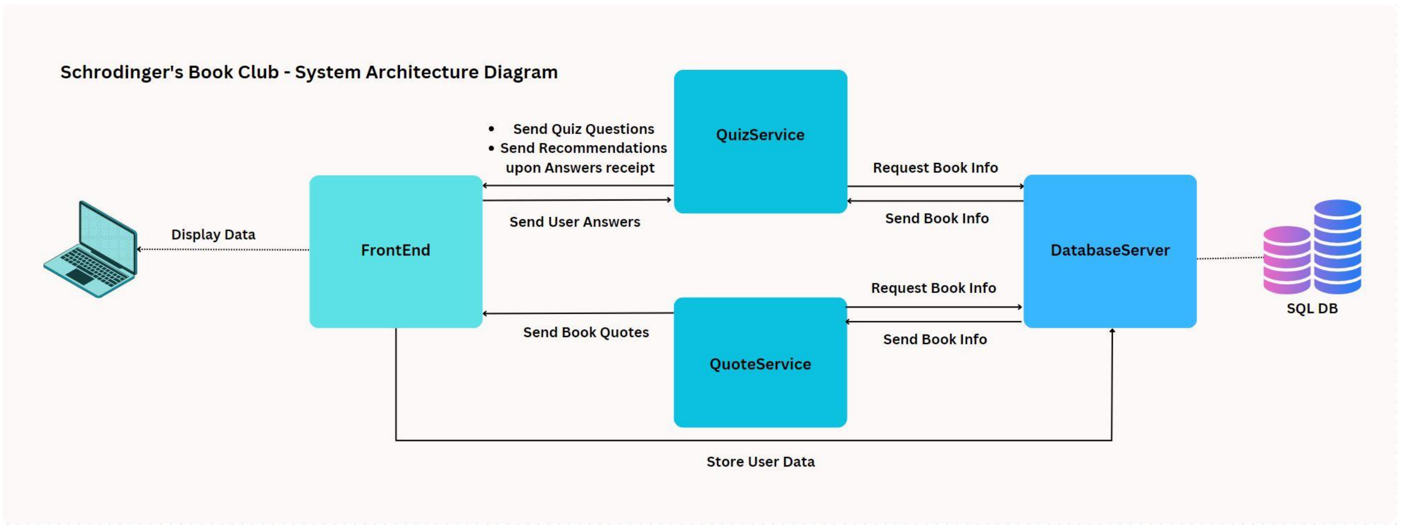
- Vue.js was used for the Front-end service. This is due to the fact that it offers a fast and efficient way to develop RESTful front-ends, especially when also using the Bootstrap-Vue library. Additionally, it can be used with the axios library to manage asynchronous GET requests, making it a reliable and scalable choice.
- Node.js was used for both the Quotes Service and the Database Service. This technology was used due to its great support in terms of libraries for building RESTful APIs. For instance, the NodeCache module can be used to cache data fetched from the database, thus improving the application performance. In addition, Node.js is also well-suited for handling asynchronous requests.
- Java was used to develop the Quiz Service. This was chosen because it can be used in conjunction with Maven and Springboot, which can provide an efficient way to manage dependencies when building RESTful web applications. Another advantage of this choice was given by our familiarity with the language syntax (since it was used for the labs), translating in smoother development.
- MySQL was chosen as a database to store both user and book information for various reasons. Firstly, this was due to its simple and familiar syntax. Additionally, because it is easy to set up as a database server when using Node.js, since the mysql library can be imported and a MySQL pool can be created in just a few lines of code. It is however worth noting that the database is its own container, and as such it could be replaced by another type of database if necessary.
- Docker was employed to containerize the application, providing portability and simplifying deployment of the different services as a single system. By including a docker-compose file, it becomes possible to launch the entire stack with a single command.
- Eureka was chosen following a recommendation from our lecturer. This technology was used as a service registry and discovery server, and enabled us to offload responsibilities such as health monitoring, scaling, and fault tolerance.
- Kubernetes was selected as the container orchestration platform due to its open-source nature, extensive documentation and compatibility with Eureka. This technology enhances fault tolerance by automatically replicating and managing application containers.

System Overview

The book recommendation system is a distributed REST system composed of four main services that work together to provide users with personalized book recommendations.

- The first component is the QuizService, which offers an interactive quiz with four multiple-choice questions designed to determine the user's reading preferences. Based on the user's responses, the service will recommend a number of books that are likely to appeal to their tastes.
- The second component is the QuotesService, which provides an alternative way of recommending books to the user. This service allows users to read short quote excerpts from books without knowing which books the quotes are from.
- The third component is the DatabaseServer, which stores information about books, users, and their past recommendations. The server connects to the database, which starts separately, and retrieves the data to be accessed by the other services.
- Finally, the FrontEnd service is responsible for displaying the recommendations to the users: it will take the data from the quiz and quotes services, and present them in an easy-to-browse format for the user.

The system architecture diagram below illustrates the interactions among these system components, which will be further detailed in the following paragraphs.



When the web page or each component is first mounted, the FrontEnd initiates a GET request to the QuizService, requesting the quiz questions and possible answers to be displayed. Additionally, the FrontEnd sends a GET request to the QuoteService, asking for book quotes to be displayed, along with their associated book information.

Once the user successfully completes the quiz and submits their answers, these are sent to the quiz service using a POST request, which then generates an array of quiz recommendations specifically tailored for the user. In order to achieve this, the QuizService first requests the book information from the DatabaseServer. After it receives a response, this information is then forwarded to the QuizService, which then forwards it to the FrontEnd for display to the user.

Similarly, before responding to the GET request from the FrontEnd, the QuoteService needs to obtain the book information from the DatabaseServer. It achieves this by iterating through a list of ISBNs of the books it intends to display. It sends GET requests to the database, searching for the required information. Once the QuoteService receives the information, it can pass it on to the FrontEnd service for display.

Furthermore, the system allows users to create accounts. This involves sending a POST request directly to the DatabaseServer, which creates a new user. Additionally, when a user is logged in, their recommendations (obtained either by answering the quiz questions or by clicking on a quote to see the corresponding book) are stored in the DatabaseServer. These recommendations are also sent using POST requests, and can be displayed to the user through the use of a GET request from the FrontEnd to the DatabaseServer.

Scalability and Fault Tolerance

The Schrodinger's Book Club application is designed to be scalable through its distributed architecture: the system has been broken down into micoservices, thus allowing for the scaling of specific services based on demand. For instance, if increased traffic was noticed for the QuizService but not for the QuotesService, the first could be independently scaled to handle this increase in traffic. Moreover, the use of a distributed architecture makes it possible to easily add new microservices to the current system.

Additionally, the system uses Eureka to handle health monitoring, scaling and fault tolerance. Eureka provides a discovery service that allows the services to find and communicate with each other. It also handles load balancing, meaning that it evenly distributes incoming traffic amongst the available instances of each service. Eureka also ensures that the system is robust by monitoring the health of each service in continuous heartbeats, and by removing any instances of the services that are found to be not responding.

Finally, Kubernetes was also employed to further improve the fault tolerance capabilities of the application. Kubernetes achieves this by ensuring that multiple replicas of each microservice are running simultaneously. If any instance of a microservice fails or otherwise becomes unresponsive, Kubernetes will automatically launch a new replica to take its place. This ensures no interruption of service for the user, and contributes to the overall robustness of the application.

Contributions

Please note that the following bullet points only indicate which group member was primarily responsible for a particular component of the system, but other group members may have also contributed to said component. Additionally, some of the GitLab commits attributed to a particular account were made during in-person group programming exercises. With that said, the main responsibilities for each group member were:

- Jerry Klos: implemented the QuizService and integrated Kubernetes to orchestrate the containers.
- Simon Farelly: developed the Database Service and containerized it using Docker for easier deployment.
- Noemi Banal: Created the FrontEnd and implemented the QuotesService. Also contributed to the final project report.

Eureka was integrated and the video presentation was recorded by the entire team during one of the group meetings.

Reflections

One of the key challenges we encountered during the project was the issue of Cross-Origin Resource Sharing (CORS) during the implementation of the application. CORS restrictions caused several of our requests to fail during development. In order to overcome this challenge, we allowed any origin from within the services. However, this solution would not be viable nor secure in a real-world environment. As such, for the Schrodinger Book Club application to be deployed on the web, we would first need to implement proper CORS handling mechanisms to ensure the security and integrity of the system.

Another significant challenge we encountered was the limited amount of time due to the high pressure that comes with the end of a semester. We dealt with this by prioritizing the most essential aspects of the project. While we were unable to implement all the features we had initially planned, we were still happy with the result as we were able to deliver a cohesive and well thought-out system.

The time constraints also influenced some aspects of our development process that we would have approached differently given more time. For instance, we would have liked to build the application using test-driven development (TDD), which would have improved our confidence in the robustness of the code. Another thing we would change if we had the opportunity to start again is that we would have allocated more time to upfront planning and design, to ensure that all team members had a clear understanding of the project goals, as well as of how the different components of the system would interact with each other. A great part of the system design emerged in an Agile manner as we went along, however this sometimes meant having to go back and change parts of the code to fit some of the new changes made.

Working on this project, we had the opportunity to use a number of different technologies, which allowed us to discover both their limitations and their benefits. Using Vue.js proved to be a significant advantage in this project, as it permitted us to develop the application rapidly and efficiently. Some of the key benefits we found to this technology included the ability to create reusable components, implement routing, and extend state management. Additionally, using pre-built bootstrap vue components allowed us to focus on the functionality of the application without having to worry too much about the aesthetics of it.

Another technology that provided substantial benefits was Eureka, a service registry and discovery server. Eureka simplified several aspects for us which would have been challenging to handle otherwise, such as health monitoring, scaling and fault tolerance. Using Eureka in our project allowed us to offload these responsibilities, and once again to simply focus on the functionality of the system. However, an issue we found when using this technology was that we were unable to get the newest version working due to some compatibility issues, and as such we had to switch to an earlier version to get it functioning correctly. Despite this, the advantages of using this service are significant, and as such we will definitely come back to it in the future.