Flow of Work Report for Distributed Online Book Store

1. Introduction

This project implements a distributed online book store using the .NET Framework with C# and follows the Model-View-Controller (MVC) architectural pattern. The system is designed as a microservices-based application consisting of three distinct services: the **Catalog Service**, the **Order Service**, and the **Front End Service**. Each service is deployed in its own Docker container, which promotes isolation, scalability, and ease of deployment in distributed environments. Communication among the services is achieved using RESTful HTTP endpoints. The Front End Service provides a terminal-based user interface that lets users interact by viewing and purchasing books.

2. System Architecture and Design

2.1 Microservices Overview

- **Catalog Service:**
  This service manages book data—including titles, quantities, prices, and topics. It exposes RESTful endpoints for:
  - **Searching** for books by topic.
  - **Retrieving detailed information** for a book by its identifier.
    The Catalog Service handles queries by reading from a persistent data store (a CSV file or an SQLite database) and responds with data formatted as JSON.
- **Order Service:**
  The Order Service processes purchase requests. On a purchase request, it:
  - Verifies the availability of the book by querying the Catalog Service.
  - Decrements the stock count if the book is available.
  - Logs the transaction in a persistent order log.
    This service ensures data integrity and error handling when a

purchase request fails (for example, when a requested book is out-of-stock).

- **Front End Service:**
  The Front End Service acts as a terminal-based client interface, displaying a menu that enables the following operations:
  - **Show All Books:** Displays a complete list of available books.
  - **Show Book by ID:** Retrieves detailed information about a specified book.
  - **Search for Books:** Filters books based on user-provided topics.
  - **Purchase a Book:** Initiates the process of buying a selected book. When the user selects an option, the Front End Service forwards the request to the corresponding service using HTTP REST calls and displays formatted JSON responses in the terminal.

2.2 Technologies and Methodologies

- **Framework:**
  The project is developed using the .NET Framework with C#. The MVC pattern decouples the presentation layer from the business logic. This separation simplifies both testing and maintenance and supports a scalable and organized structure.
- **RESTful Communication:**
  Each microservice exposes its operations using RESTful APIs. For instance:
  - **GET /search/{topic}** and **GET /info/{id}** in the Catalog Service.
  - **POST/PUT /purchase/{id}** in the Order Service.
    These REST endpoints ensure loose coupling and easy integration of the components.
- **Docker Containers:**
  To simulate a real-world distributed environment, each service is hosted in its own Docker container. This containerization:

  - Simplifies deployment across different machines.
  - Provides isolation between the services.
  - Allows independent scaling and updates without affecting other components.

- **Persistent Storage:**
  Minimal persistent storage is provided by using CSV files or an SQLite database for both the Catalog and Order Services, ensuring the durability of data across service restarts.

3. Flow of Work

3.1 User Interaction Flow

1. **Terminal-Based Menu:**
   The user interacts with the Front End Service through a terminal-based menu. This interface provides clear options:
   - **Show All Books:** Lists available books.
   - **Show Book by ID:** Displays detailed information of a book.
   - **Search for Books:** Retrieves a filtered list based on a topic.
   - **Purchase a Book:** Initiates the purchase process.
   - **Exit:** Terminates the session.
2. **Request Processing:**
   Once an option is selected:
   - The Front End Service issues an HTTP GET request to the Catalog Service for search or detailed information operations.
   - For purchase operations, it sends an HTTP POST (or PUT) request to the Order Service.
   - The results are then received as JSON objects and presented in a user-friendly format within the terminal.

3.2 Backend Flow and Service Coordination

- **Search and Info Operations:**
  The Catalog Service handles GET requests by reading from its persistent storage. For example, a search by topic returns an array of JSON objects that include each book's ID and title, while detailed information includes fields like the title, quantity, and price.
- **Purchase Operation:**
  The Order Service validates the purchase request by:
  - Querying the Catalog Service for current stock information.
  - Decrementing the stock count if available.
  - Logging the purchase in its order log.
  - Sending a confirmation message such as "bought book [book_name]" back to the Front End Service.
  - If the book is out-of-stock, the service responds with an appropriate error message.
- **Concurrency and Persistence:**
  The .NET Framework's built-in features for multi-threading or asynchronous handling enable the services to manage multiple concurrent requests efficiently. Persistence is ensured by using lightweight storage mechanisms, providing reliability without the overhead of more complex database systems.

4. Discussion and Future Enhancements

4.1 Tradeoffs and Design Considerations

- **Simplicity vs. Scalability:**
  By using lightweight microservices, the system remains easy to maintain and deploy. However, for larger-scale deployments, additional considerations such as more advanced load balancing and fault tolerance would be required.

- **Containerization:**
  Docker provides a practical way to emulate a distributed environment. This approach simplifies deployment and scaling, though further automation through orchestration tools like Kubernetes could be explored in future iterations.
- **Framework Choice:**
  The use of the .NET Framework with an MVC pattern ensures clear separation of concerns. Future projects might consider ASP.NET Core, which offers built-in cross-platform support and better integration with container orchestration systems.

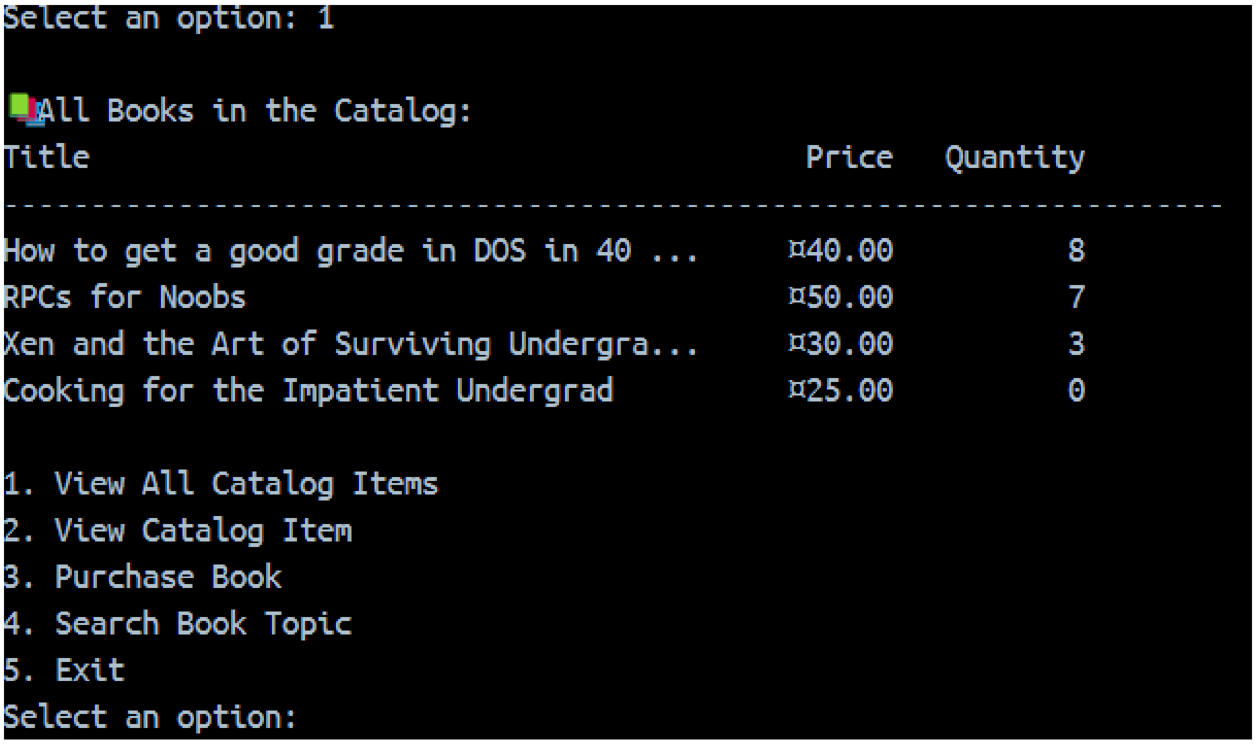
4.2 Potential Enhancements

- **Enhanced User Interface:**
  Consider evolving the terminal-based client into a web-based or GUI application for improved user experience.
- **Robust Error Handling and Logging:**
  Implementing more advanced error logging and monitoring will help troubleshoot issues in a distributed environment and handle edge cases gracefully.
- **Security Improvements:**
  Adding security measures such as HTTPS, API authentication, and secure input validation will protect against common vulnerabilities.
- **Scalability Upgrades:**
  Incorporating advanced orchestration and scaling solutions, including Docker Compose or Kubernetes, can help manage increased load and streamline deployment.

5. Front-end Description

The main list is as follows:
1. View All Catalog Items
2. View Catalog Item
3. Purchase Book
4. Search Book Topic
5. Exit
Select an option:

If we select the first option as bellow, we can list all Books we have in catalog.

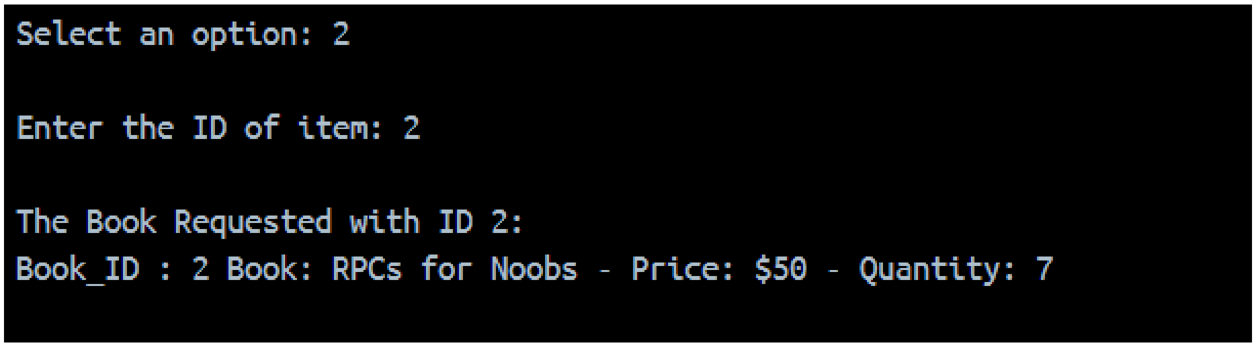


```
Select an option: 1

📚All Books in the Catalog:
Title                                       Price   Quantity
----------------------------------------------------------------------
How to get a good grade in DOS in 40 ...   ¤40.00         8
RPCs for Noobs                             ¤50.00         7
Xen and the Art of Surviving Undergra...   ¤30.00         3
Cooking for the Impatient Undergrad        ¤25.00         0

1. View All Catalog Items
2. View Catalog Item
3. Purchase Book
4. Search Book Topic
5. Exit
Select an option:
```

If we select the second option we can enter any id of the books, the result is the all information about this book:

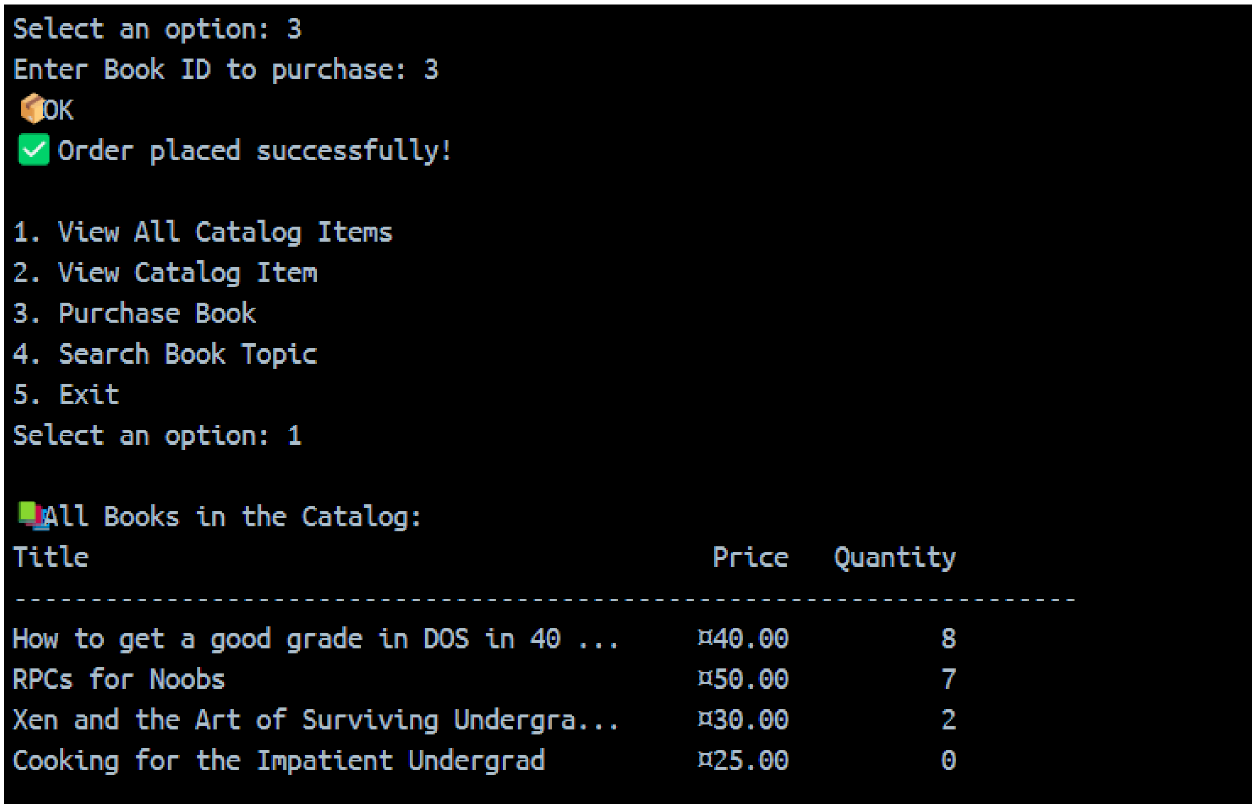


```
Select an option: 2

Enter the ID of item: 2

The Book Requested with ID 2:
Book_ID : 2 Book: RPCs for Noobs - Price: $50 - Quantity: 7
```

If we select the third option the we can purchase and book with the id we specified and we see that the book quantity of that we purchase decreased by one:

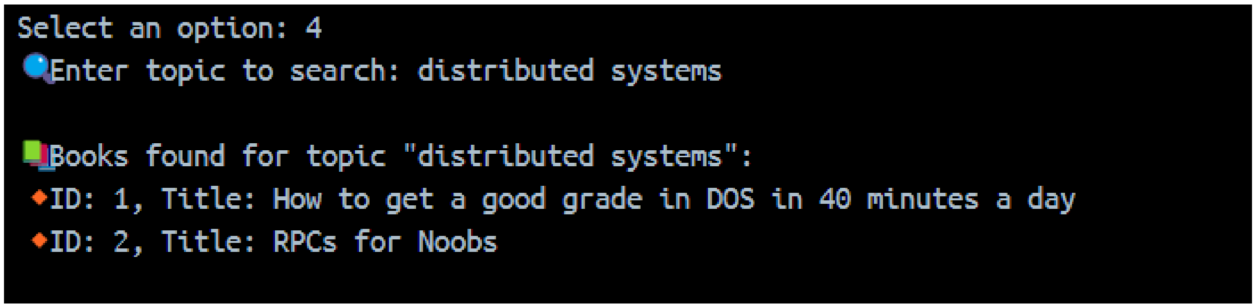


```
Select an option: 3
Enter Book ID to purchase: 3
📦OK
✅ Order placed successfully!

1. View All Catalog Items
2. View Catalog Item
3. Purchase Book
4. Search Book Topic
5. Exit
Select an option: 1

📚All Books in the Catalog:
Title                                         Price   Quantity
----------------------------------------------------------------
How to get a good grade in DOS in 40 ...     ¤40.00          8
RPCs for Noobs                               ¤50.00          7
Xen and the Art of Surviving Undergra...     ¤30.00          2
Cooking for the Impatient Undergrad          ¤25.00          0
```

Finally, if we select the fourth option, we can search on any topic we write in the search bar, the result is all book have the same topic:



```
Select an option: 4
🔍Enter topic to search: distributed systems

📚Books found for topic "distributed systems":
 🔸ID: 1, Title: How to get a good grade in DOS in 40 minutes a day
 🔸ID: 2, Title: RPCs for Noobs
```

6. Conclusion

This report has presented the design, architecture, and workflow of our distributed online book store project. By leveraging the .NET Framework, MVC, RESTful APIs, and Docker containerization, the system successfully separates concerns into distinct microservices—Catalog, Order, and Front End. This modular approach not only simplifies development and testing but also sets the stage for future enhancements and scalability. The design decisions made balance simplicity with the practical needs of distributed application deployment, and future improvements could further enhance user experience, security, and robustness.