SEZG503

FULL STACK APPLICATION DEVELOPMENT

BOOK EXCHANGE PLATFORM DESIGN DOCUMENTATION

From,

SUPRIYA P

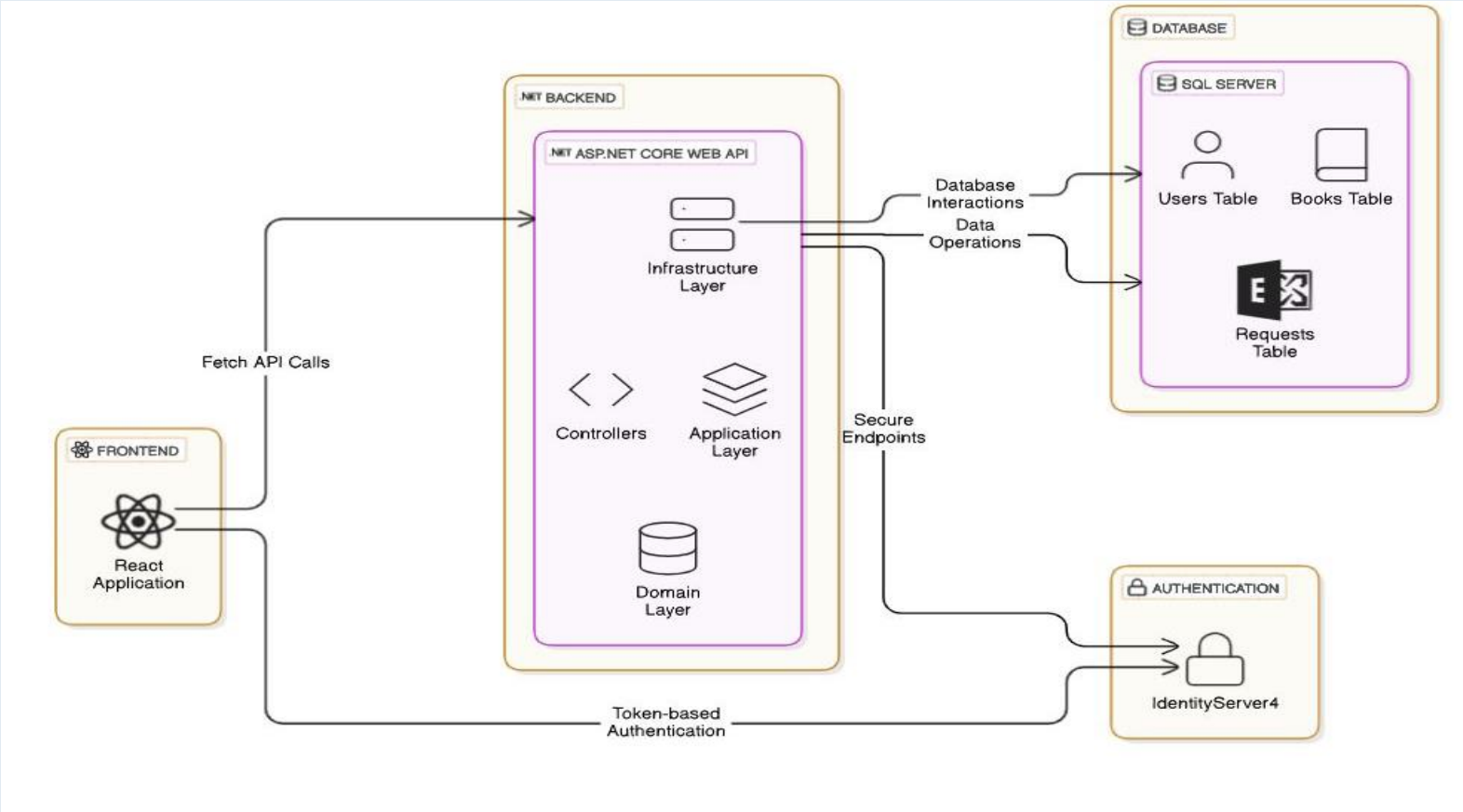
2023TM93755

INDEX

BOOK EXCHANGE APPLICATION

FULL ARCHITECTURE DIAGRAM

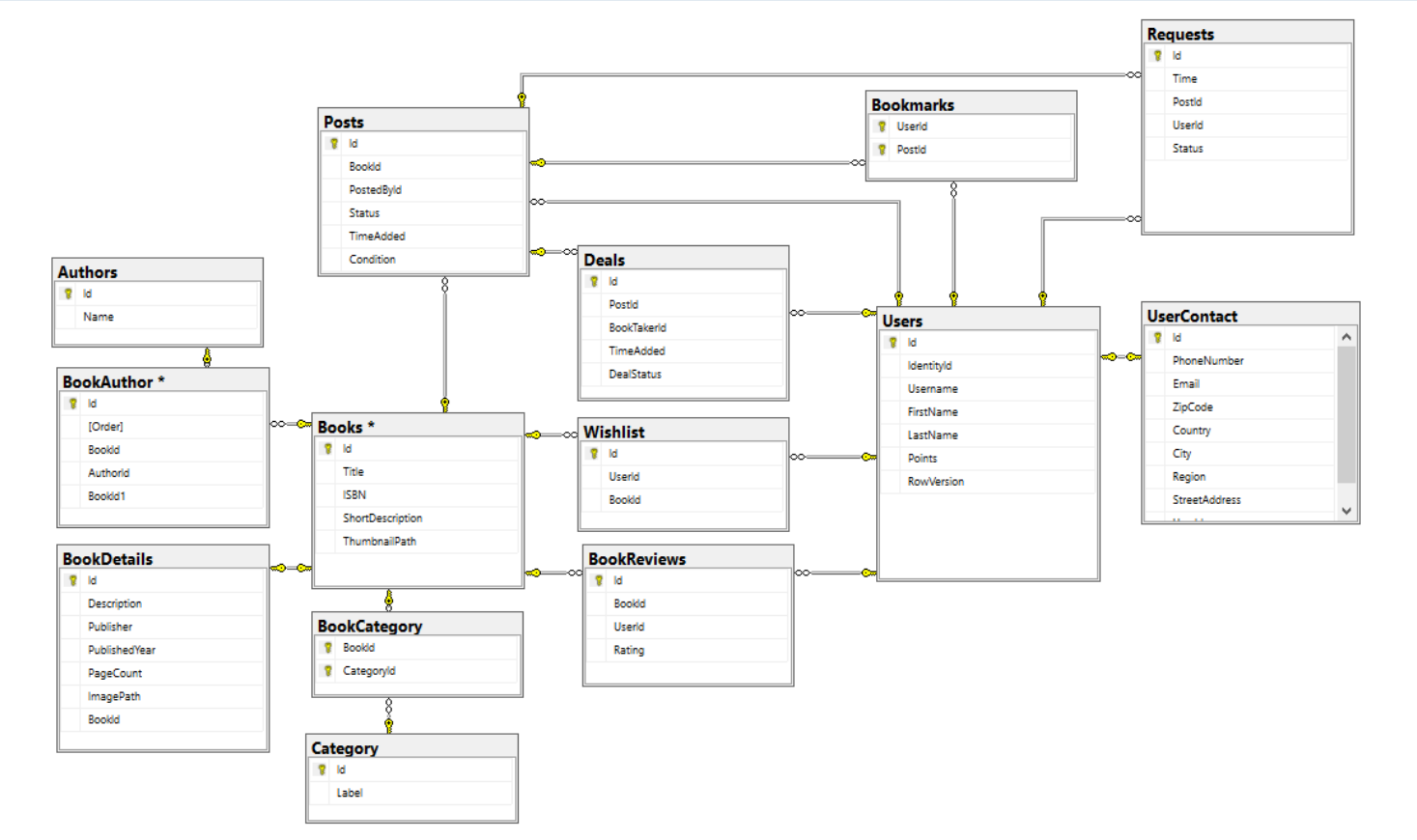
1.1 Frontend – Backend Interaction



BOOK EXCHANGE APPLICATION

FULL ARCHITECTURE DIAGRAM

1.2 Database diagram



BOOK EXCHANGE APPLICATION

FULL ARCHITECTURE DIAGRAM

1.3 API Endpoints

Resource	HTTP Method	Endpoint	Description
Author	GET	/api/Author	Retrieve all authors
	POST	/api/Author	Add a new author
	DELETE	/api/Author/{id}	Delete an author by ID
Book	GET	/api/Book/{id}	Retrieve a book by ID
	PATCH	/api/Book/{id}	Update a book partially by ID
	PUT	/api/Book/{id}	Update a book completely by ID
	DELETE	/api/Book/{id}	Delete a book by ID
	GET	/api/Book	Retrieve all books
	POST	/api/Book	Add a new book
Category	GET	/api/Category	Retrieve all categories
	POST	/api/Category	Add a new category
	DELETE	/api/Category/{id}	Delete a category by ID
Deal	PATCH	/api/Deal/{id}	Update a deal partially by ID
Post	GET	/api/Post/{id}	Retrieve a post by ID
	GET	/api/Post	Retrieve all posts
	POST	/api/Post	Add a new post
	PATCH	/api/Post/{id}	Update a post partially by ID
	PUT	/api/Post/{id}	Update a post completely by ID
	DELETE	/api/Post/{id}	Delete a post by ID
	GET	/api/Post/conditions	Retrieve post conditions
Request	PUT	/api/Request/{id}	Update a request by ID

BOOK EXCHANGE APPLICATION

FULL ARCHITECTURE DIAGRAM

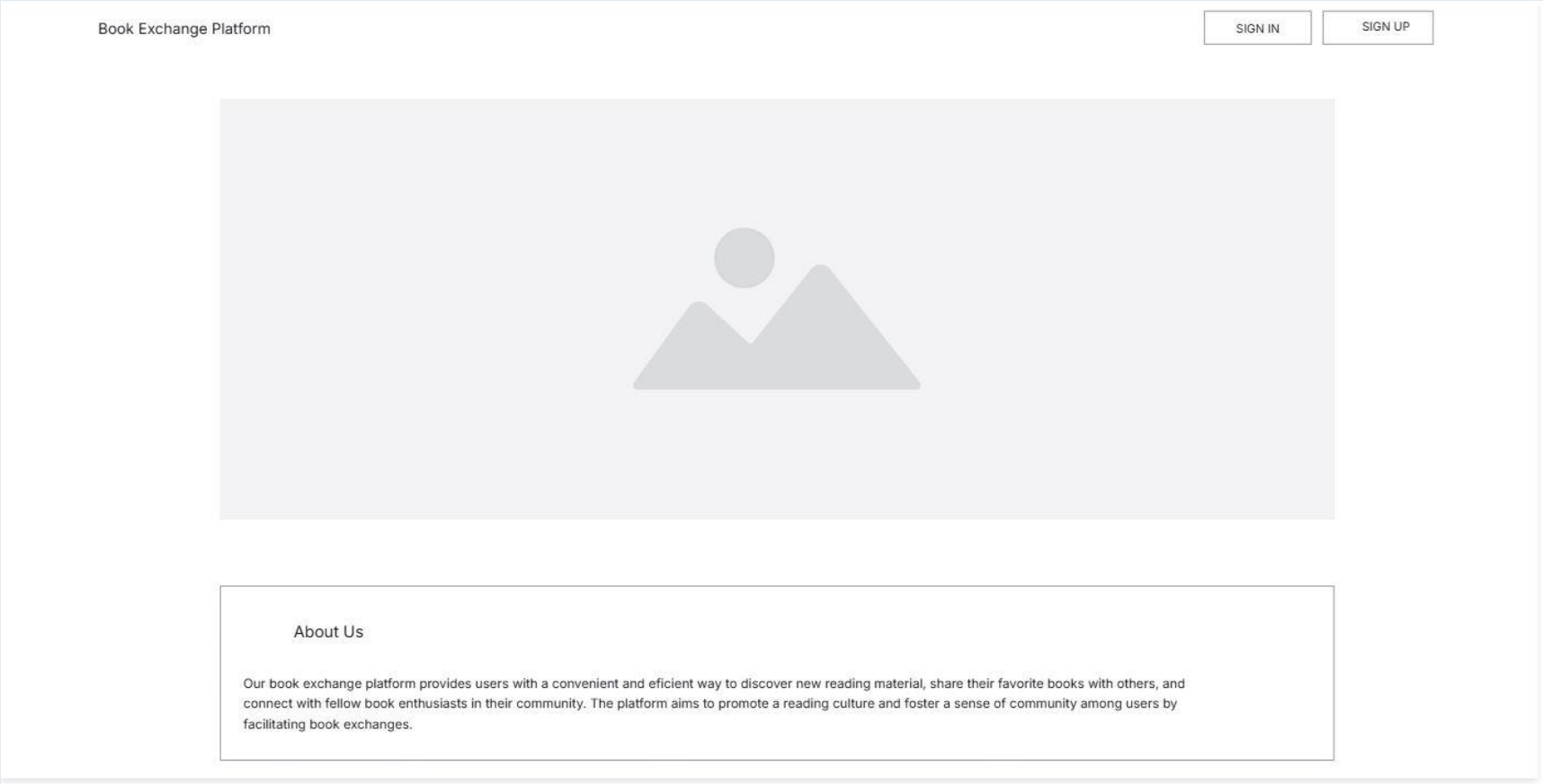
1.3 API Endpoints

Resource	HTTP Method	Endpoint	Description
User	GET	/api/User/current-user	Retrieve the current logged-in user
	POST	/api/User	Register a new user
	GET	/api/User	Retrieve all users
	DELETE	/api/User	Delete a user
	GET	/api/User/{id}	Retrieve a user by ID
	GET	/api/User/{id}/stats	Retrieve stats of a user by ID
	GET	/api/User/{id}/books/recommended	Retrieve recommended books for a user by ID
	GET	/api/User/{id}/books/wished	Retrieve wished books for a user by ID
	POST	/api/User/{id}/books/wished	Add a book to a user's wishlist
	GET	/api/User/{id}/posts/owned	Retrieve posts owned by a user by ID
	POST	/api/User/{id}/requests	Create a request for a user by ID
	GET	/api/User/{id}/requests/to	Retrieve incoming requests for a user by ID
	GET	/api/User/{id}/requests/from	Retrieve outgoing requests from a user by ID
	GET	/api/User/{id}/deals/from	Retrieve deals from a user by ID
	GET	/api/User/{id}/deals/to	Retrieve deals to a user by ID
Wishlist	GET	/api/Wishlist	Retrieve the wishlist

BOOK EXCHANGE APPLICATION

UI/UX WIREFRAME

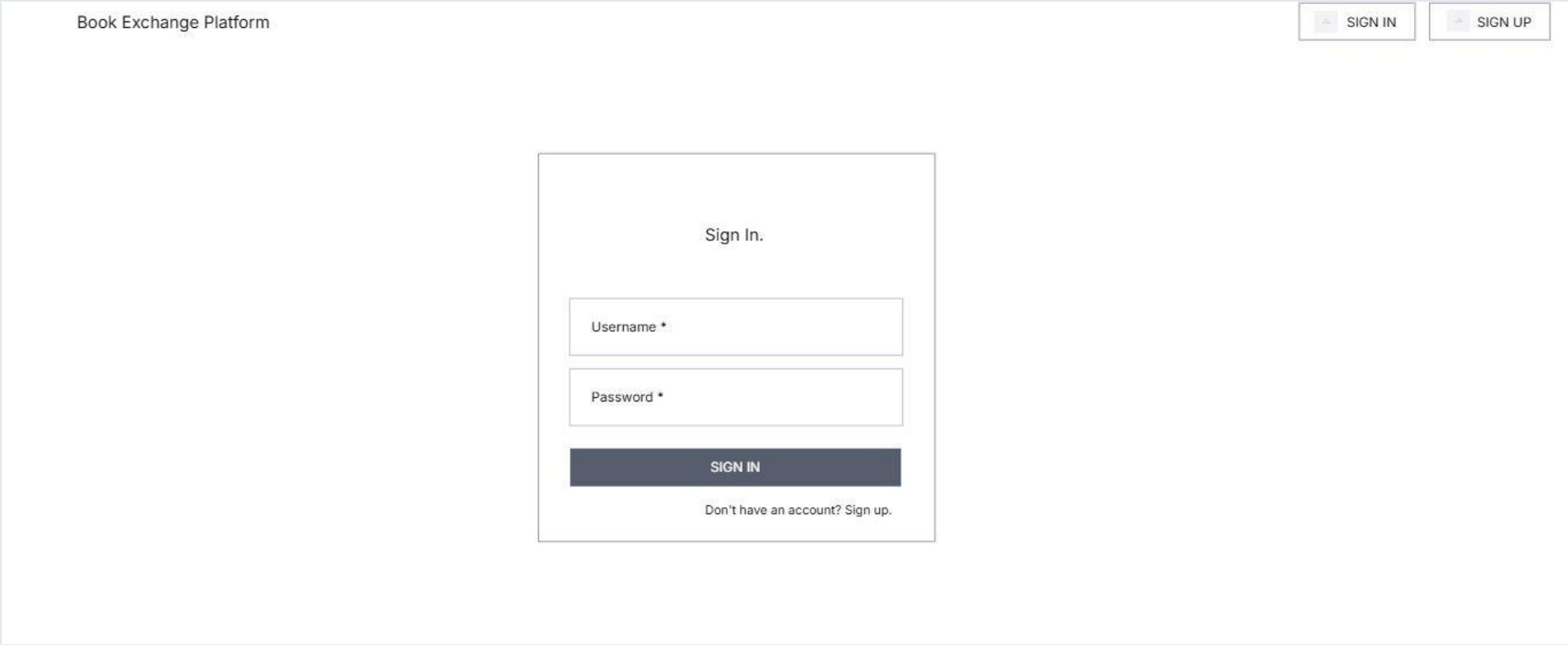
2.1 Home page



BOOK EXCHANGE APPLICATION

UI/UX WIREFRAME

2.2 Sign in page



BOOK EXCHANGE APPLICATION

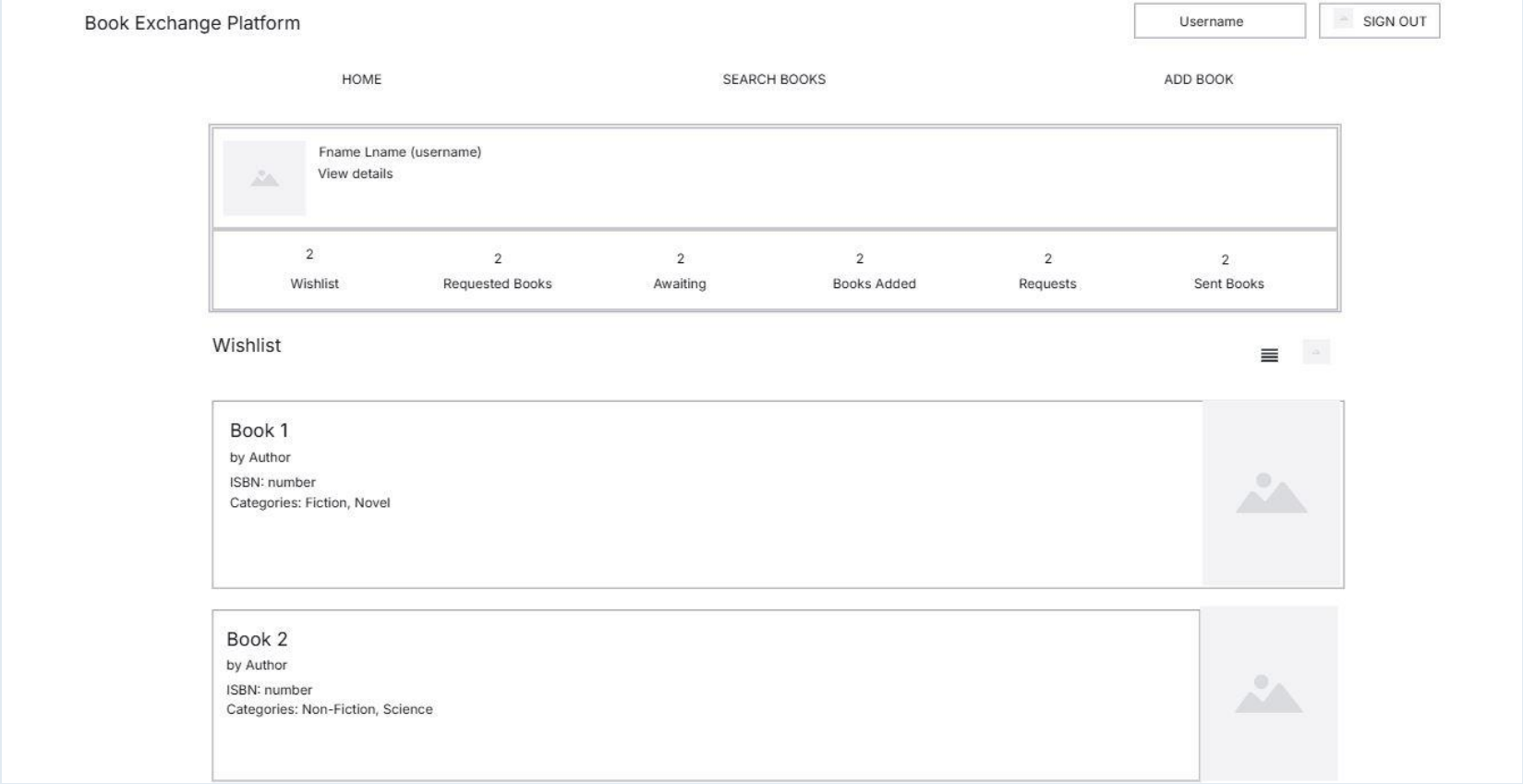
UI/UX WIREFRAME

2.3 Sign up page

BOOK EXCHANGE APPLICATION

UI/UX WIREFRAME

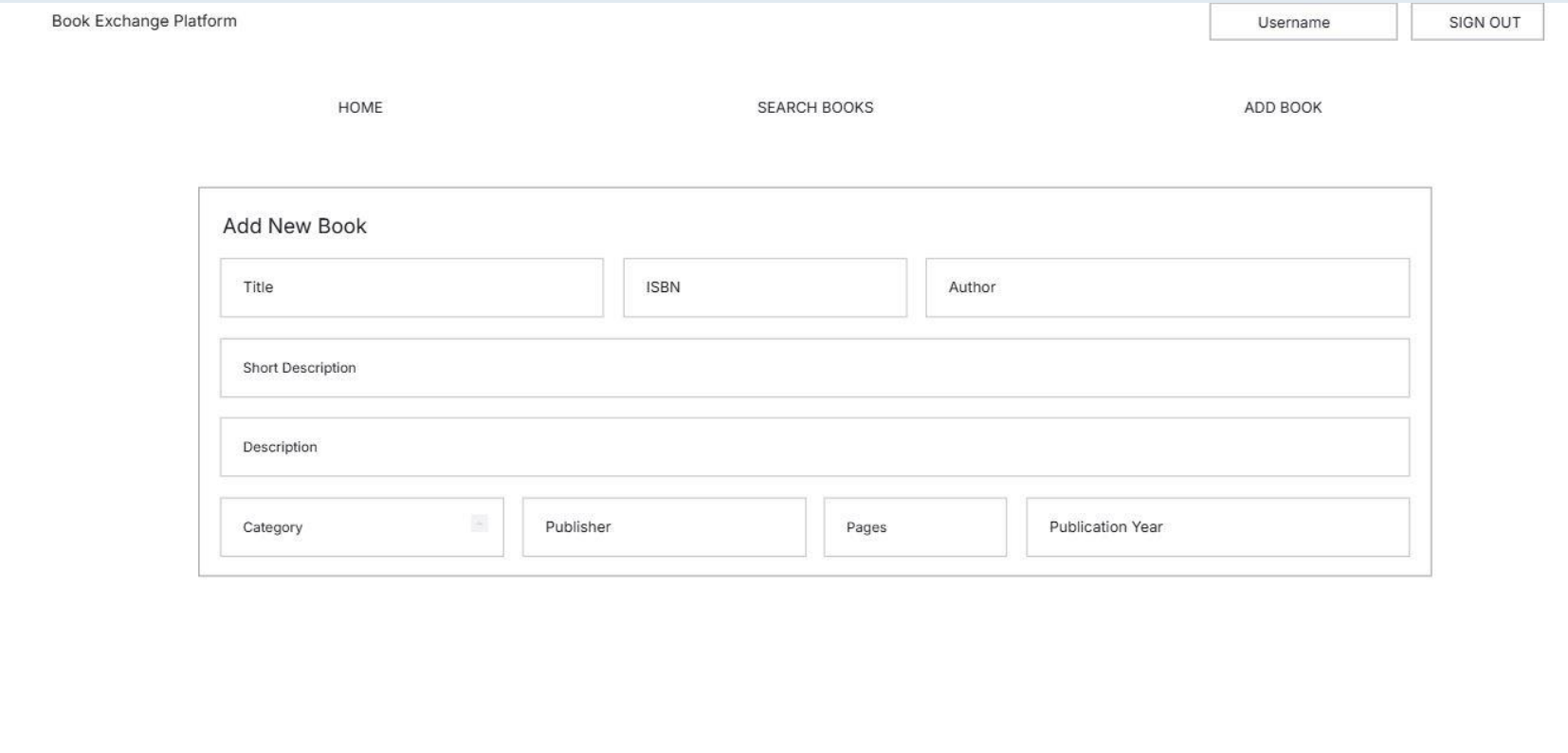
2.4 Profile Page



BOOK EXCHANGE APPLICATION

UI/UX WIREFRAME

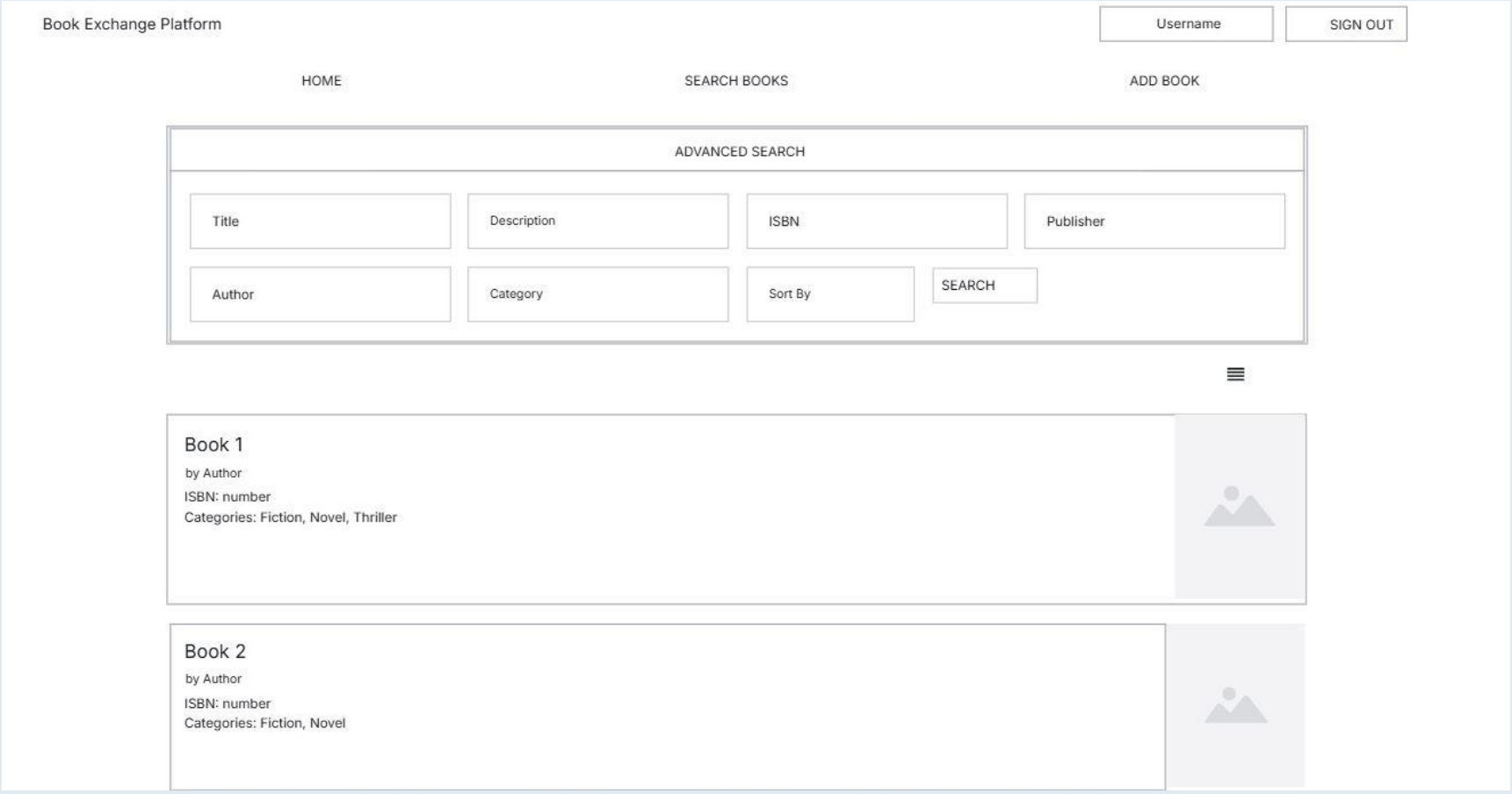
2.5 Add Page



BOOK EXCHANGE APPLICATION

UI/UX WIREFRAME

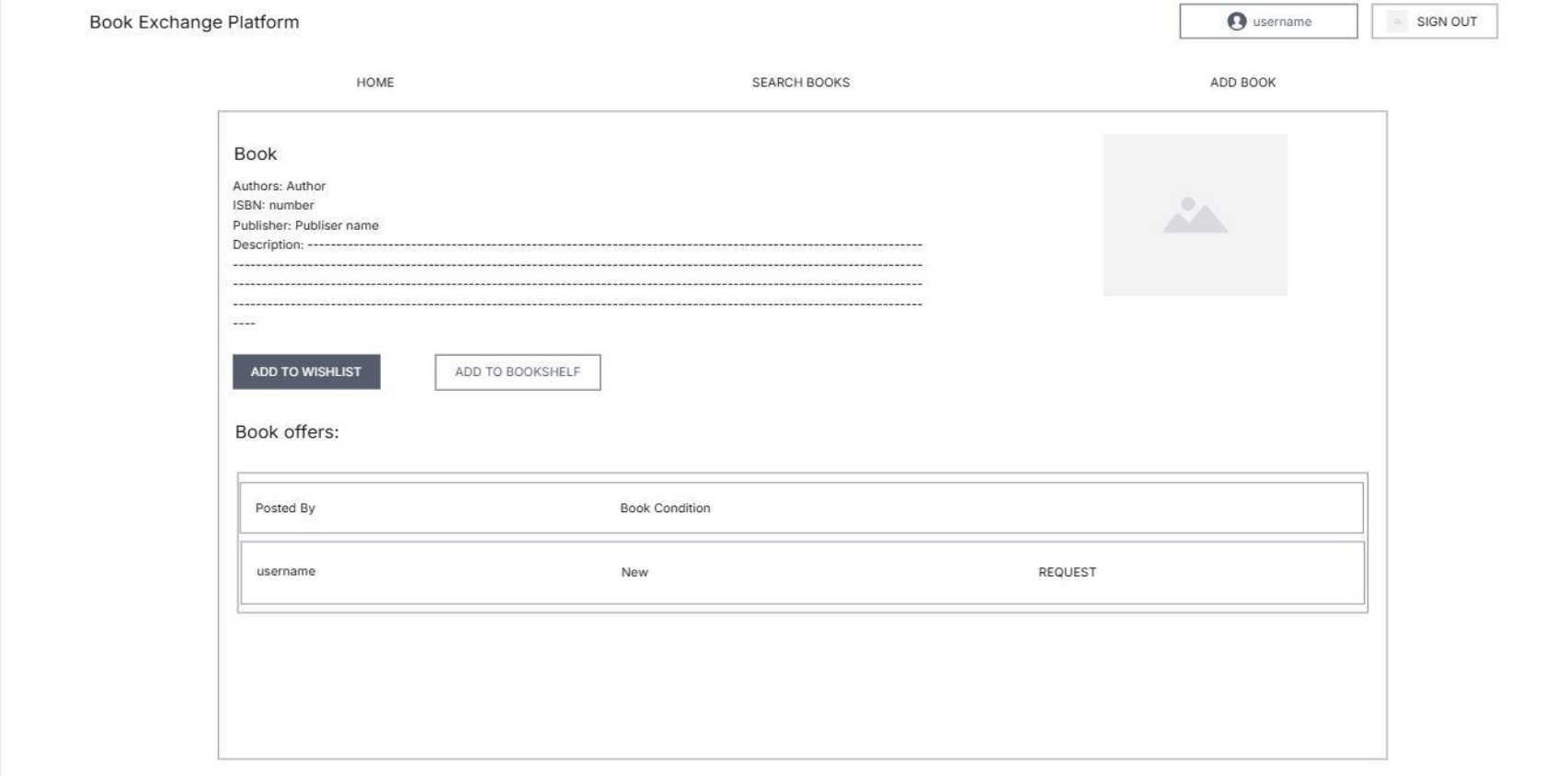
2.6 Search Page



BOOK EXCHANGE APPLICATION

UI/UX WIREFRAME

2.7 Request Page

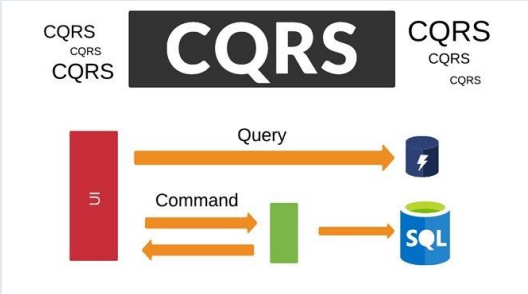


BOOK EXCHANGE APPLICATION

APPROACH TO IMPLEMENTING SCALABILITY AND MODULARITY.

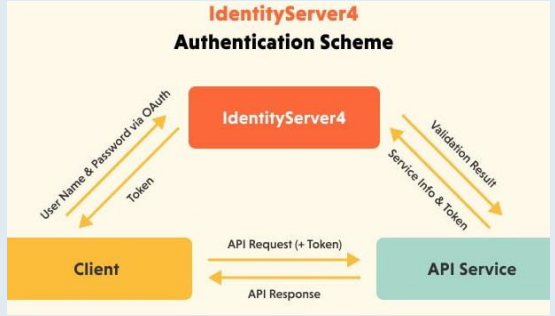
3.1 Scalability

- Adopted the CQRS pattern in API, which divides read and write operations, allows to manage read-heavy tasks more effectively. This architecture supports scalable query management with the capacity to optimize reading processes independently from writing.



CQRS (Command Query Responsibility Segregation)

- By implementing IdentityServer4 as an external authentication service, can independently scale our authentication functions away from the main application. This approach enables the application to efficiently accommodate a growing number of login requests without affecting primary operations.



- We utilized Docker to containerize API and React, facilitating horizontal scaling through Kubernetes or other orchestration platforms. This configuration enables us to adjust the number of instances dynamically according to user demand, enhancing the system's ability to manage both API and frontend requests.

kubernetes

docker

BOOK EXCHANGE APPLICATION

APPROACH TO IMPLEMENTING SCALABILITY AND MODULARITY.

3.1 Scalability (Continued..)

- SQL Server database is designed for scalability through table partitioning to manage large datasets and indexed columns for quick lookups, essential for high-traffic queries such as book searches and recommendations.
- The recommendation feature in Application is structured as a background service, enabling recommendations to be calculated in advance and stored. This method alleviates the burden on the primary application server and enhances response times for users.

APPROACH TO IMPLEMENTING SCALABILITY AND MODULARITY

3.2 Modularity

- **Multi-Tiered Structure:**
  Every layer in our solution (API, Application, Domain, Infrastructure) is separate and adheres to a clearly defined role, enables to change or substitute components without impacting other layers. For instance, Infrastructure manages database operations independently, ensuring that changes to the database do not affect business logic.

- **Autonomous Identity Server:**
  IdentityServer functions as a dedicated service for authentication. This division allows us to control and enhance the authentication system separately, rendering it more modular and simpler to maintain.

- **Microservice-Optimized Framework for Recommendation Service:**
  The recommendation service in BEP.Application is designed in a way that it can eventually be split into a microservice. At present, it functions as an internal service; however, the design allows for future implementation as a standalone API for modular expansion or autonomous updates.

APPROACH TO IMPLEMENTING SCALABILITY AND MODULARITY

3.2 Modularity

- **Reusable React Components :**

  The front-end code employs a modular structure featuring reusable React components. Every component is standalone, allowing for independent updates of UI elements. To manage shared state, I've utilized React Context, making sure that global state management is centralized while remaining independent from individual components.

- **Dependency Injection (DI) to Achieve Loose Coupling:**

  I have employed ASP.NET Core’s dependency injection to maintain a loosely coupled structure for services, repositories, and other dependencies in API and Application. This design facilitates the replacement of components (such as repositories) without changing the fundamental logic.

- **Common Code Shared Library:**

  To ensure modularity and prevent duplication, common functions are housed in a specific Common library. This library contains DTOs, utilities, and constants that can be accessed across layers without code duplication, thereby improving maintainability.

THANK YOU

SUPRIYA P

2023TM93755