

Code Developer Project:

Revenue and User Management Website

Goal: The primary aim of this project is to develop a comprehensive revenue and user management bookstore website.

I – Requirements Gathering and Project Planning

Part I – Requirements Survey

Following a client meeting, the project involves the creation of a bookstore website with the following features:

For Any User:

- View books
- Search for books by various fields (Book details)
- Register user
- Log in User (access to Personal Area)

For Registered Users (call Readers):

- Submit interests in buy the book (if flag is “Available”) Appear the Button “interested in buy”
- Comment on books
- Rate books (1 to 5)
- Access a personal area
- Maintain a list of favorite books

For Administrators:

- Manage users/Readers, with the ability to block them
- Manager books:
 - Adding and editing all fields
- Managing flags (counting flag “sold”, search /counting the Button: "Interested in buy" + user email + Book details, change the Flag “available” to “sold”)

Books Details:

- List of Books (Title, Author, Editor)
- Genres (Unkown, Fiction, Non Fiction)
- Language (portugues, english, other)
- Flag (available, sold, not available)

Registered User's/Readers Personal Area:

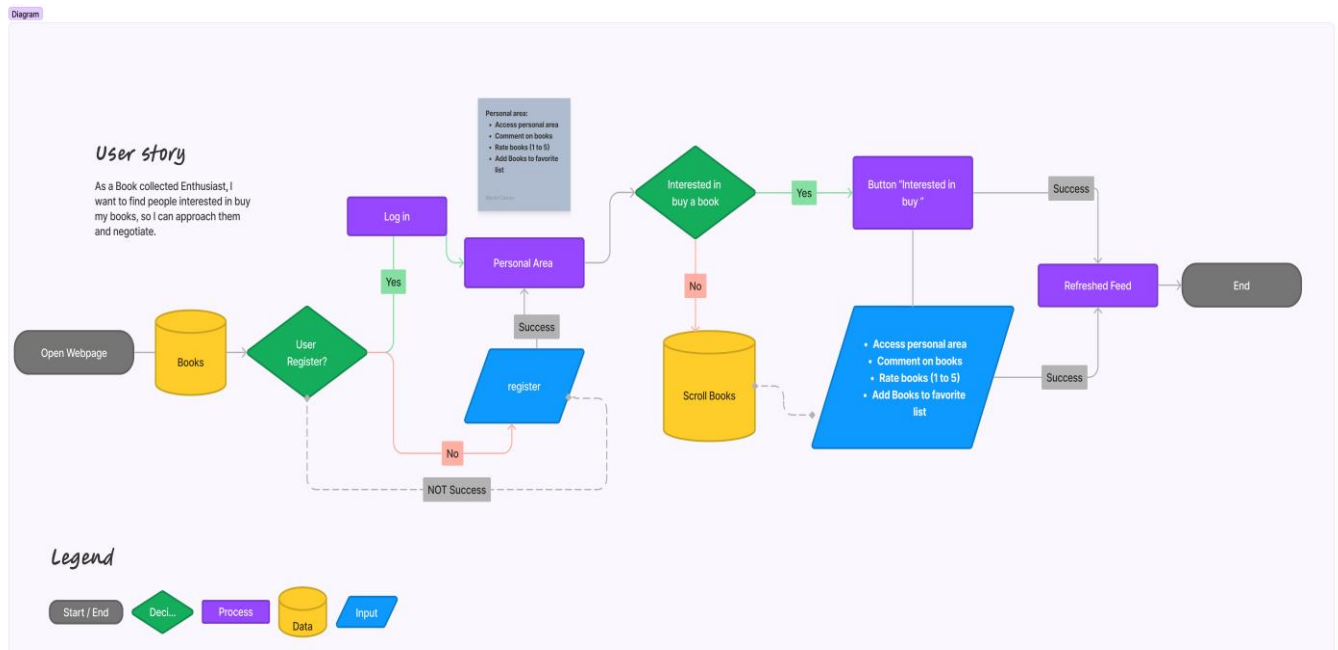
- User Name (First and Last name, only letters are acceptable)
- User Email (no spaces are acceptable, has to have @ and .com in it)
- Password (has to have at least one number, one letter, one Upper Letter, one special character and all together have to be 8. No space is acceptable)

Current Stage Objectives:

- Create a list of functionalities and sub-functionalities associated with the identified needs.
- Develop use case diagrams (flowcharts) and application functionalities, along with their pseudocode .
- Design mockups/layouts of website forms for both the front office and back office.

Note: conceptualizing the structure for all elements, as the remaining diagrams will be essential for the comprehensive development of the project.

User Story - to easily understand the purpose of the WebSite



List of functionalities and sub-functionalities:

For Any User:

1. **View Books:**
 - Sub-functionality: Browse books by title, author, or editor.
 - Sub-functionality: Display book details.
2. **Search for Books:**
 - Sub-functionality: Search books by various fields (title, author, editor).
 - Sub-functionality: Filter books based on genre, language, and availability.
3. **Register User:**
 - Sub-functionality: Provide user details (name, email, password).
 - Sub-functionality: Validate user details (name format, email format, password requirements).
4. **Log in User (access to Personal Area):**
 - Sub-functionality: Authenticate user (check email and password match).
 - Sub-functionality: Redirect to the user's personal area upon successful login.

For Registered Users:

1. **Submit Interest in Buying a Book (if available):**
 - Sub-functionality: Display "Interested in buy" button for available books.
 - Sub-functionality: Submit interest in buying a book.
2. **Comment on Books:**
 - Sub-functionality: Allow users to add comments to books.
 - Sub-functionality: Display comments for each book.
3. **Rate Books (1 to 5):**
 - Sub-functionality: Allow users to rate books on a scale of 1 to 5.
 - Sub-functionality: Display average ratings for each book.
4. **Access Personal Area:**
 - Sub-functionality: Display user name (first and last name).
 - Sub-functionality: Display user email.
 - Sub-functionality: Display user's list of interests, comments, ratings, and favorite books.
 - Sub-functionality: Edit user details (name, email, password).
5. **Add or Unadd Books to the List of Favorite Books:**
 - Sub-functionality: Display a button to add/unadd books to/from the list of favorite books.
 - Sub-functionality: Manage the list of favorite books.

For Administrators:

1. **Manage Users (with the ability to block them):**

- Sub-functionality: View list of users.
- Sub-functionality: Block/unblock users.

2. Manage Books:

- Sub-functionality: Add new books to the system.
- Sub-functionality: Edit existing book details.
- Sub-functionality: Manage flags (counting flag "sold," search/counting "Interested in buy" flags, change flag "available" to "sold").

Diagrams & Pseudocode

(step-by-step guides for someone who's not into coding)

1 - View Books:

What it Does:

Lets you see what books are there.

Steps:

2.1 You say you want to look at books.

2.2 The system shows you the list of books.

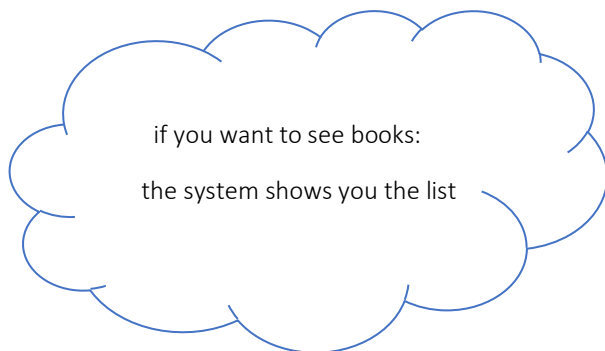
Diagram:

Pseudocode:

if user selects "View Books":

 retrieve list of books from the database

 display books on the front end



2 - Search for Books:

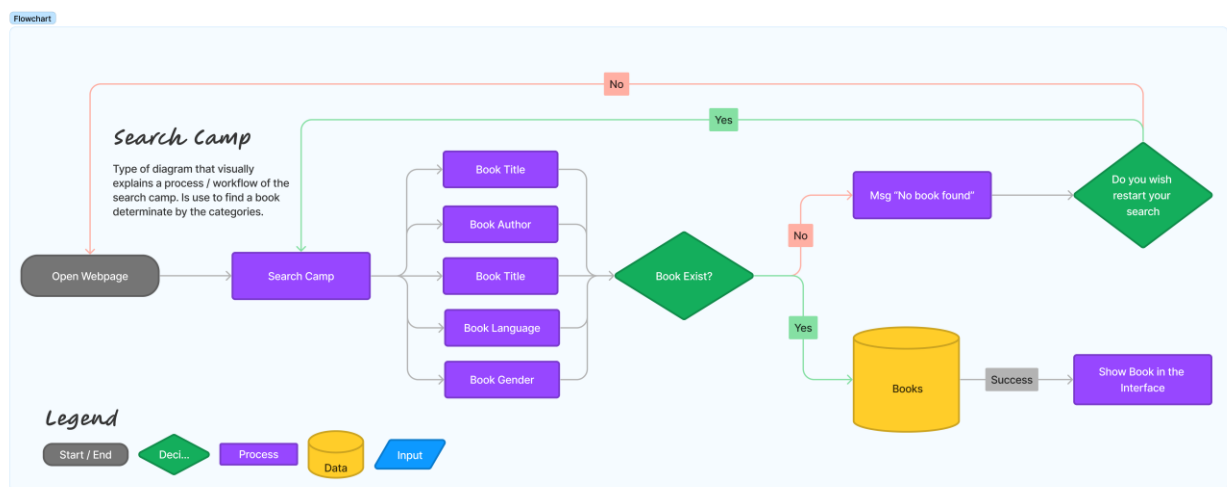
What it Does:

Helps you find specific books based on what you're looking for.

Steps:

- 2.1 You decide whether you want to search or filter books.
- 2.2 If you choose to search, you enter details like the title, author, or editor.
- 2.3 The system finds books that match your criteria.
- 2.4 The system shows you the list of books that match your search.

Diagram:



Pseudocode:

if you want to find a book:

- choose to search or filter

- if you choose to search:

- enter criteria (title, author, editor, etc.)

- find books based on criteria

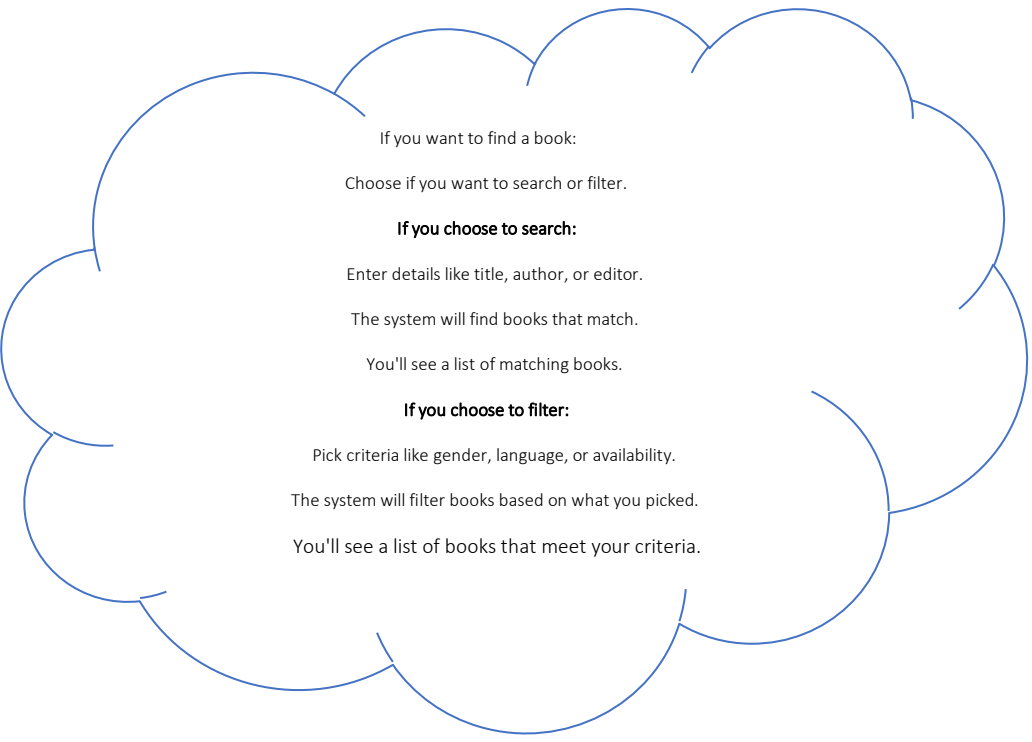
- display the results

- else if you choose to filter:

- choose filter criteria (genre, language, availability)

- filter books based on criteria

- display the results



If you want to find a book:

Choose if you want to search or filter.

If you choose to search:

Enter details like title, author, or editor.

The system will find books that match.

You'll see a list of matching books.

If you choose to filter:

Pick criteria like gender, language, or availability.

The system will filter books based on what you picked.

You'll see a list of books that meet your criteria.

3 - User Registration:

Sub-functionality: Provide user details (name, email, password).

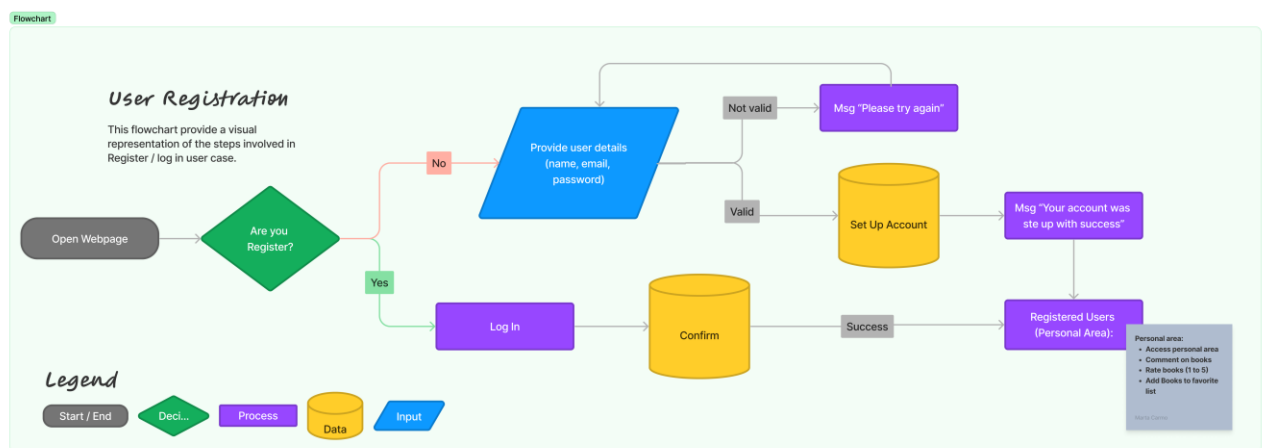
What it Does:

Helps you sign up for the website.

Steps:

- 1.1 You fill in your name, email, and password.
- 1.2 The system checks if everything looks good.
- 1.3 If it's okay, you're all set!

Diagram:



Pseudocode:

if user submits registration form:

 validate user input


 if input is valid:

 create user account

 redirect to user area

 else:

 display error messages



if you want to sign up:

fill in your details

check if everything is okay

if yes, you're done!

4 - Registered Users (personal area):

Sub-functionality: Submit interest in buying a book (if available).

What it Does:

Tells the system you want to buy a book.

Steps:

- 3.1 You express interest in a book.
- 3.2 The system checks if the book is available.
- 3.3 If yes, it tells the admin you want it.

Diagram:

Pseudocode:

if user submits interest in a book:

 check availability of the book

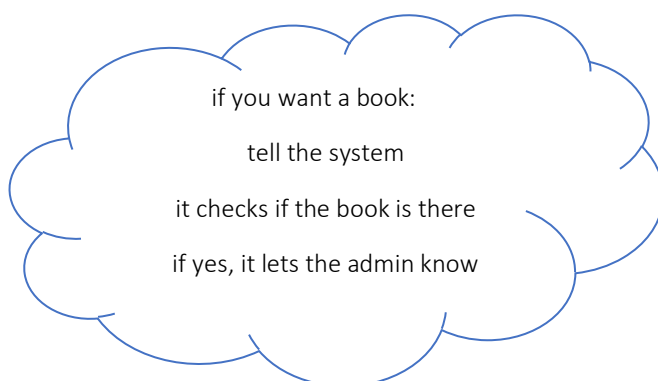
 if available:

 notify administrator

 update book status to "Interested"

 else:

 display message that the book is not available



5 - Administrators:

Sub-functionality: Manage books, including adding/editing all fields.

What it Does:

Helps the admin handle books.

Steps:

- 4.1 The admin can see a list of books.
- 4.2 They can add or edit book details.
- 4.3 They can manage flags like available/sold/not available.

Diagram:

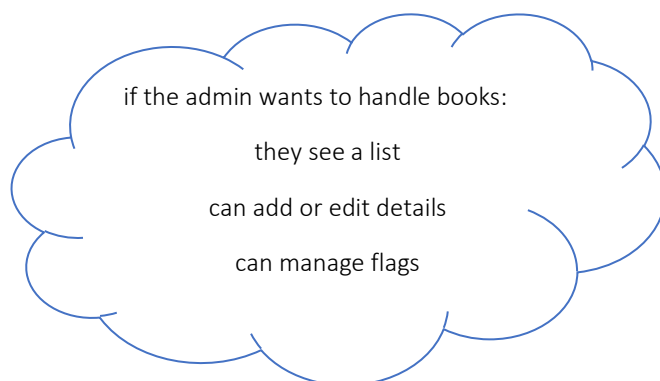
Pseudocode:

if admin wants to manage books:

 show list of books

 allow adding/editing book details

 allow managing flags (counting, searching, updating availability)



Desing Mockups/Layouts

(look at the webpage, you'll see clear sections for entering your details, exploring book information, and the admin having control over users and books.)

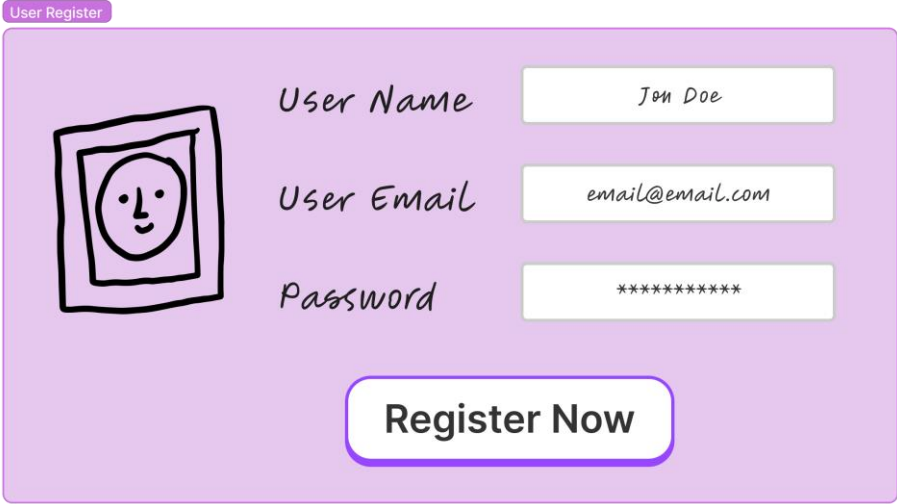
1. User Registration Form:

Fields:

- **Name:** Your full name goes here.
- **Email:** Your email address, so we can keep in touch.
- **Password:** Your secret key to access your account.

Button:

- **Register:** Once you've filled in your details, hit this button to officially join.



A mockup of a user registration form. It features a purple header with the text "User Register". The form itself is a light purple rectangle. On the left side of the form is a hand-drawn icon of a person's face inside a square frame. To the right of the icon are three input fields. The first is labeled "User Name" and contains the text "Jon Doe". The second is labeled "User Email" and contains the text "email@email.com". The third is labeled "Password" and contains a series of asterisks "*****". Below these fields is a large, rounded rectangular button with a purple border and the text "Register Now".

2. Book Details Page:

Display:

- **Title:** The name of the book.
- **Author:** The person who wrote the book.
- **Editor:** The person who prepared the book for publishing.
- **Genre:** Whether the book is Fiction, Non-Fiction, or Unknown.
- **Language:** The language the book is written in.
- **Flag:** This shows if the book is available, sold, or not available.

Button:

- **Submit "Interest to Buy":** If you really want to buy this book, hit this button.
- **Comment (Area):** Share your thoughts on the book in this space.
- **Rate (from 1-5):** Give the book a rating from 1 to 5 stars.

Book Details Page:



3. Admin Panel:

User Management Section:

- **List of Users:** See a list of everyone using the website.
- **Block/Unblock Buttons:** Control who can and can't use the website.

Book Management Section:

- **Add/Edit Books:** Put new books into the system or change details of existing ones.
- **Manage Flags:** Keep track of how many books are sold, search for books with user interest, and update availability.

“Interested in Buy” List:

- List of user that press interested in buy button in each book details.

User Management

| | | | |
|---|--------|-------------------------|-----------------|
| 1 | User X | emailX@email.com | Block / Unblock |
| 2 | User Y | <u>emailY@email.com</u> | Block / Unblock |
| 3 | User W | <u>emailW@email.com</u> | Block / Unblock |

Book Management

| | | | | | | |
|---|--------------|-----------------|----------|----------|--------|------|
| 1 | Book Title X | Author X | Editor X | Language | Gender | Flag |
| 2 | Book Title Y | <u>Author Y</u> | Editor Y | Language | Gender | Flag |
| 3 | Book Title W | <u>Author W</u> | Editor W | Language | Gender | Flag |

"Interested in Buy" List

| | | | | | | | |
|---|--------|------------------|--------------|----------|----------|----------|--------|
| 1 | User W | emailW@email.com | Book Title X | Author X | Editor X | Language | Gender |
| 2 | User X | emailX@email.com | Book Title Y | Author Y | Editor Y | Language | Gender |
| 3 | User X | emailX@email.com | Book Title W | Author W | Editor W | Language | Gender |

Part II – Backend

Structure: Below is the diagram which shows the Project structure that we are going to implement. We have divided the API into a different layer.

5. Use Postman / SoapUI to communicate and test and save all requests in one collection. Postman is a web debugging tool that lets you compose HTTP requests and view the raw HTTP responses.

4. Create the services layer for communication between Web API and DAL

Service Layer: The main role of this layer is to have the business logic as well as to convert the ViewModel object to DTO (Data Transfer Object) and vice versa. You can have a private validation method to validate the ViewModel object and take necessary actions on it.

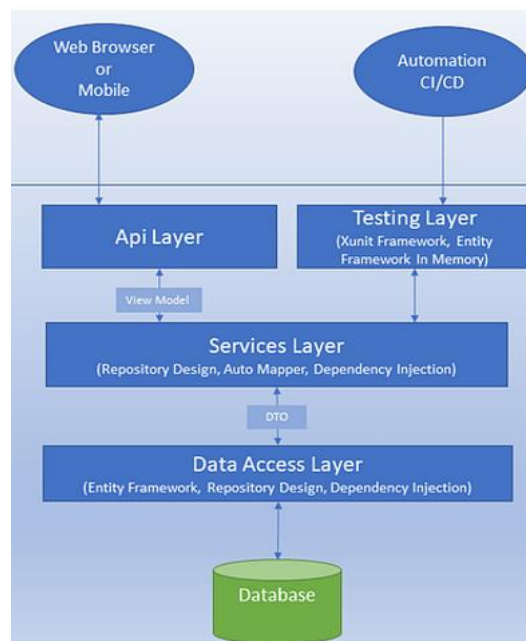
3. Create the Web API (REST) layer (protected by authentication)

API Layer: It will be used to handle the request and send back the appropriate responses. This layer doesn't know about the service layer functionality, its main function is to pass the request to the service layer and handle any Global Exception. Authentication and authorization will fall under this layer.

2. Create the project's persistence layer (DAL=Data Access Layer) using the Entity Framework.

DAL=Data Access Layer: it is used to communicate with the Databases. (SQL Server, MySQL, NoSQL, Redis Cache, etc.)

1. Create the model layer (entities) of the application, with the description of each entity and your relationships



Note 1: each layer must be in separate projects, within the same solution.

Note 2: at least one class, repository, service and controller must be developed for books. It means that CRUD operations must be functional for revenue. These requests (as well as the authentication method) must be delivered to a collection of the Postman/SoapUI.

Note 3: When running the program, it must check whether a local database exists, otherwise you must automatically carry out the migration as well as seed the database with the information they find relevant to test the API.

Model Layer (Entities):

Entities:

1. Book Entity:

- **Properties:** BookId (PK), Title, Author, Editor, Genre, Language, Flag.
- **Relationships:**
 - No direct relationships mentioned in the requirements.

2. Reader Entity:

- **Properties:** ReaderId (PK), Name, Email, Password.
- **Relationships:**
 - One-to-Many with Interest: A user can have multiple interests.
 - One-to-Many with Comment: A user can have multiple comments.
 - One-to-Many with Rating: A user can have multiple ratings.
 - Many-to-Many with Book (FavoriteBooks): A user can have many favorite books, and a book can be a favorite of many users.

3. InterestInBuy Entity:

- **Properties:** InterestId (PK), BookId (FK), ReaderId (FK), Timestamp.
- **Relationships:**
 - Many-to-One with Book: Many interests can be linked to one book.
 - Many-to-One with User: Many interests can be linked to one user.

4. Comment Entity:

- **Properties:** CommentId (PK), BookId (FK), ReaderId (FK), CommentContent , Timestamp.
- **Relationships:**
 - Many-to-One with Book: Many comments can be linked to one book.
 - Many-to-One with User: Many comments can be linked to one user.

5. Rating Entity:

- **Properties:** RatingId (PK), BookId (FK), UserId (FK), Value, Timestamp.
- **Relationships:**
 - Many-to-One with Book: Many ratings can be linked to one book.
 - Many-to-One with User: Many ratings can be linked to one user.

Relationships:

- **User-Interest:**
One-to-Many relationship between User and Interest.
- **User-Comment:**
One-to-Many relationship between User and Comment.
- **User-Rating:**
One-to-Many relationship between User and Rating.
- **User-FavoriteBooks (Many-to-Many):**
Many-to-Many relationship between User and Book (FavoriteBooks).
- **Interest-Book:**
Many-to-One relationship between Interest and Book.
- **Interest-User:**
Many-to-One relationship between Interest and User.
- **Comment-Book:**
Many-to-One relationship between Comment and Book.
- **Comment-User:**
Many-to-One relationship between Comment and User.
- **Rating-Book:**
Many-to-One relationship between Rating and Book.
- **Rating-User:**
Many-to-One relationship between Rating and User.

2. Persistence Layer (DAL) using Entity Framework:

- **BookRepository:**
 - Implements CRUD operations for the Book entity.
 - Utilizes Entity Framework for data access.

3. Web API (REST) Layer:

- **BookController:**
 - Exposes CRUD operations for the Book entity through RESTful endpoints.

4. Services Layer:

- **BookService:**
 - Acts as an intermediary between the Web API and DAL.
 - Contains business logic if needed.

5. Postman/SoapUI Testing:

- **Postman Collection:**
 - Includes requests for CRUD operations on Books.
 - Includes an authentication request.

BookStore Project structure (Visual Studio)

- Create the application's model layer (entities):

I've defined entities like Reader, Book, Rating, etc., each with their properties and relationships.

- Create the project's persistence layer (DAL) using the Entity Framework:

I've started creating the DAL with a DbContext (AppDbContext).

- Create the Web API (REST) layer:

I've set up the Web API layer with controllers like BooksController, ReadersController, etc.

- Create the services layer for communication between Web API and DAL:

I've initiated services like BookService, ReaderService, etc.

- Step 6: Database Migration and Seeding

Rest – Representational State Transfer

- Style of Architecture for Building web services
- Set of principles

Stateless API

- Style of architecture for building web services
- Set principles

.NET Core

- HTTP verbs
- Routing
- Model Binding
- Content Types
- Response Types

HTTP Verbs (HTTP verbs define the types of action that can be performed on a resource identified by a URI)

- GET
- POST
- PUT
- DELETE
- PATCH
- OPTION

DbContext Class

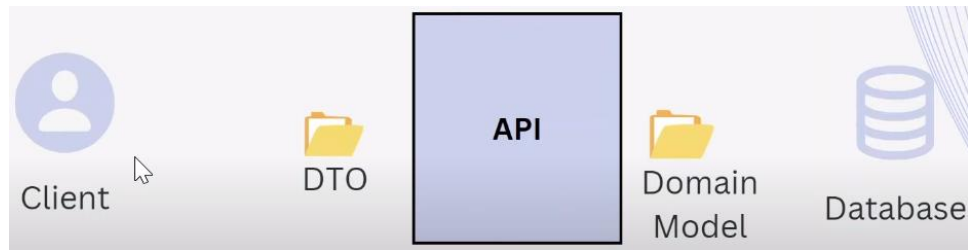
- Maintaining Connection To Db
- Track Changes
- Perform CRUD operations
- Bridge between domain models and the database

Dependency Injection

- Design pattern to increase maintainability, testability
- DI built into ASP.Net Core
- DI container is responsible for creating and managing instances

DTOs – Data Transfer Objects

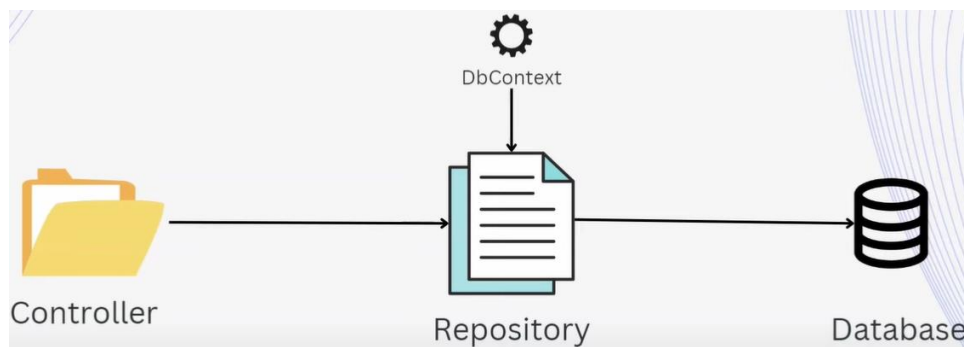
- Used to transfer data between different layers
- Typically contain a subset of the properties of the domain model
- For example, transferring data over a network



Advantages of DTOs: Separation of concerns, performance, security and versioning.

Repository Pattern

- Design pattern to separate the data access layer from the application
- Provides interface without exposing implementation
- Helps create abstraction



Benefits: Decoupling, consistency, performance and multiple data sources (switching).

Part III - Frontend

Based on the requirements survey carried out previously, as well as the layouts/mockups layouts/mockups, we want you to implement an SPA using the Angular framework:

1. Draw the navigation diagram for the website
2. Implement the components needed to implement the site
3. In the forms, validate the user inputs
4. Use Bootstrap to implement the layout and style of the site, but use and override some rules using CSS.

Note 1: one of the forms and data collection processes should be revenue.

Navigation Diagram:

1. Home Page

Welcome users, provide an overview of the bookstore.

2. Books

Display a list of books.

Allow users to view details, add new books, and edit existing ones.

3. Authors

Showcase authors, their works, and related information.

4. Readers

Manage readers, their interests, and interactions.

5. User Authentication

Allow users to register, log in, and manage their accounts.

6. Revenue Form

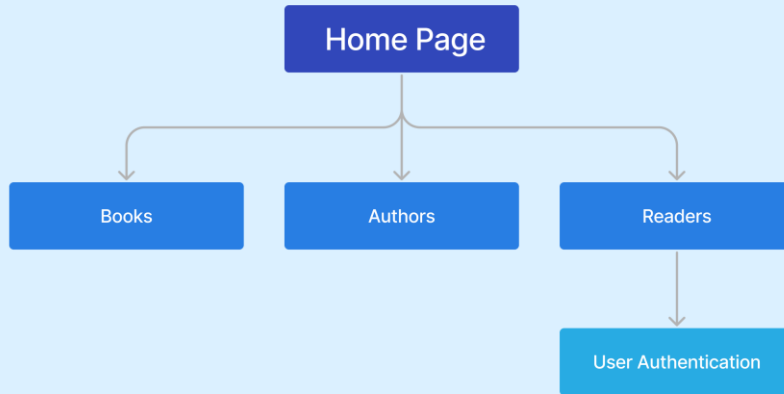
Collect information related to revenue.

Navigation Connections:

- Navigation from Home to Books.
- Navigation from Home to Authors.
- Navigation from Home to Readers.
- Navigation from Home to User Authentication.
- Navigation from Home to Revenue Form.
- Navigation from Books to Authors (to explore authors of a specific book).
- Navigation from Books to Readers (to see who's interested in a specific book).
- Navigation from Books to Revenue Form (to add revenue related to a book).
- Navigation from Authors to Books (to explore books by a specific author).
- Navigation from Readers to Books (to see books of interest).
- Navigation from Revenue Form to Books (to associate revenue with a specific book).

Diagram:

Navigation diagram for the website



Part IV - Testing and Deployment

Based on the requirements of the functionalities needed in the project developed, unit tests should be unit tests must be created in order to validate their operation and use. There should tests for:

5. Backend:

- a. Unit tests for business rules;
- b. Unit tests on repositories;

6. Frontend:

- a. Unit tests on Angular components;

Based on the DevOps methodology, it is intended to:

- 1. Creation of an Azure DevOps repository;
 - a. Use of epics, backlog items and branches
- 2. Creating a branch for project delivery;