# **Introduction**

Welcome to the documentation for the Book Management CRUD (Create, Read, Update, Delete) Application. This app is specifically designed to handle and manage a collection of books in a database. It allows users to perform essential operations on the book records, including adding new books, viewing details, updating book information, and removing books from the table.

Whether you're building a personal library, managing an inventory of books for a bookstore, or organizing books for a library system, this application provides a straightforward and efficient way to perform the four core operations of CRUD.

### **Key Features:**

* **Create:** Add new books to the database by specifying details such as title, author, publication year, genre, and ISBN.
* **Read:** View existing books with options to filter, search, and sort based on various attributes (e.g., title, author, genre).
* **Update:** Modify existing book records to correct information or reflect new details (e.g., change author name, update publication year).
* **Delete:** Safely remove books from the database when they are no longer needed or are incorrectly added.

This documentation will guide you through the app’s setup, features, and usage, providing clear instructions for both developers and users. Whether you are managing a small collection or a large database, this Book Management CRUD app will streamline your workflow and improve data organization. Let’s get started!

# **How To Use Application**

### **Prerequisites**

Before running the Book Management CRUD application, ensure you have the following installed:

1. **MySQL:**
   * A MySQL server instance (either locally or hosted) is required to store and manage book records.
   * Make sure MySQL is running and you have access to create databases and tables.
2. **.NET SDK (for server-side):**
   * Ensure that you have the **.NET SDK** installed on your machine. You can download it from the [official .NET website](https://dotnet.microsoft.com/download).
   * The backend of this application is built with .NET, and you'll need the SDK to run and develop the server-side code.
3. **Node.js and npm (for client-side):**
   * The front-end client is built using **Vite** and **npm**. Ensure that you have **Node.js** and **npm** installed. You can download them from the [official Node.js website](https://nodejs.org/).
   * **Vite** is used for fast development server setup and bundling in the front-end application.

### **Running the Application**

Once the prerequisites are installed, follow these steps to run the application:

1. **Set up MySQL Database:**
   * First, create a MySQL instance on your local machine or server.
2. **Configure the Connection String:**
   * Open the .env file in the /server directory. Update the CONNECTION\_STRING variable with your MySQL instance details (host, username, password, and database name). This will allow the application to connect to your database.
3. **Update database schema:** 
   * use EF migrations tool to update db to newest version using command dotnet ef database update
4. **Run the Backend Server:**
   * Once the database is set up and the connection string is configured, navigate to the /server directory in your terminal. Run the command dotnet run to start the backend server. This will launch the API that handles the CRUD operations for the book records.
5. **Run the Frontend Client:**
   * After the server is running, navigate to the /client directory in your terminal. Run npm install to install the required dependencies. Once the dependencies are installed, run npm run dev to start the Vite development server. This will open the front-end application in your browser, allowing you to interact with the CRUD interface.

By following these steps, you’ll have both the backend and frontend running locally, enabling you to manage your book collection seamlessly.

# **Application Structure**

This section provides an overview of the key components of the Book Management CRUD application, including the server-side controllers, models, and internal services that enable the handling of book records. The application follows a clean architecture, organizing related functionality into distinct modules for ease of development and maintenance.

#### **Server**

The Server folder contains all the necessary components to run the backend API, handle app logic, and interact with the database.

##### **Controllers**

Controllers are responsible for handling incoming HTTP requests, processing them, and returning appropriate responses. These controllers define the API endpoints for performing CRUD operations on the book records.

* **BooksController**The BooksController handles the core CRUD operations for books. It provides the following API endpoints:
  + **GET /get-books** – Retrieves a list of all books.
  + **GET /get-books-genres** – Retrieves a list of all available genres.
  + **POST /add-book** – Creates a new book in the database.
  + **POST /update-book** – Updates an existing book record.
  + **DELETE /delete-book** – Deletes a book record.
* The controller delegates the actual data manipulation to the service layer BookService, which is responsible for interacting with the database.
* **BookModel**The BookModel represents the data structure for a book. It defines the fields that can be passed in API requests to create or update books, including:
  + ID – The ID of the book, used for updating.
  + Title – The title of the book.
  + Author – The author of the book.
  + ISBN – The ISBN number of the book.
  + Genre – The genre of the book.
  + PublishedDate – The publication date of the book.
  + Publisher – The publisher of the book.
  + Language – The language of the book.

##### **Internals**

The Internals folder contains all internal logic and services that interact with the database, perform business operations, and provide data to the controllers. It is organized into modules to keep the codebase modular and easy to manage.

* **Database**The Database module contains the core database logic that interacts with the MySQL database, including models and services that help perform CRUD operations on the Book table.
  + **Book**The Book class represents the database entity that maps to the Books table in MySQL. Is identical to BookModel mentioned previously
  + **BookService**The BookService handles the business logic related to book management. It provides methods for creating, reading, updating, and deleting books in the database. The service layer abstracts the database operations from the controller to maintain clean separation of concerns. Example methods include:
    - GetBooks() – Retrieves all books from the database.
    - AddBook(Book book) – Adds a new book to the database.
    - UpdateBook(Book book) – Updates an existing book record.
    - DeleteBook(Guid id) – Deletes a book record by GUID.
  + **DatabaseContext**The DatabaseContext class is responsible for managing the connection between the application and the MySQL database using Entity Framework Core. It defines the DbSet for the Books table, allowing the application to interact with the database in an object-oriented manner. The context also handles database migrations and ensures that the application can persist changes to the database. OnModelCreating method defines layout of DB table.
* **Utils**

In the Internals/Utils folder, you will find important utility classes that are used throughout the application for error handling and constant data management. These classes ensure that common values and error codes are standardized and easily accessible across various components of the application, such as controllers and services.

#### **ErrorCodes Enum**

The ErrorCodes enum defines different error types that can occur within the application, making it easier to manage and communicate errors consistently.

* **NoError**: Represents a successful operation with no errors.
* **SQLError**: Represents a database-related error.

The enum is used throughout the application, especially in service layers like BookService, to return meaningful error codes when an operation fails.

#### **Constants Class**

The Constants class contains static, read-only fields for various constant values used throughout the application. Specifically, it contains a list of predefined book genres that can be used in API responses to fetch available genres. This list of genres is returned in the GET /get-books-genres API endpoint.

* **BookGenres**: A string array containing various book genres, ranging from "Fiction" and "Science Fiction" to "Psychology" and "Children's Literature." These genres are typically used when users are creating or categorizing books.

### **How the Components Interact**

1. **Controller Layer:**When a request is made to one of the API endpoints (e.g., creating a new book or updating an existing one), the request is handled by the appropriate controller method (such as BooksController.AddBook). The controller validates the input, prepares the data, and calls the service layer for processing.
2. **Service Layer:**The service layer (BookService) is where the core logic resides. It communicates with the database through the DatabaseContext and manages data operations. For example, when creating a book, the service layer maps the data from the request (in the form of BookModel) into a Book entity and saves it to the database.
3. **Database Interaction:**The DatabaseContext class is responsible for managing the database connection. It uses Entity Framework Core to interact with the Books table, performing operations like inserting new records, updating existing ones, and deleting records. The Book class serves as the representation of a book in the database, containing fields like Title, Author, and ISBN.

### 

### 

### 

### 

### 

### **Frontend Overview**

This section provides an overview of the frontend setup for the application, which is built using **React**. The frontend allows users to interact with the backend API, including adding and viewing books.

#### **Frontend Folder Structure**

* **/components**: Contains reusable React components used across the application, such as forms, pages, and UI elements.
  + **AddBookForm**: A form component for adding new books to the system.
  + **ContentSwitcher**: A component that dynamically switches between different views (like adding a book or listing all books).
  + **ListBooksPage**: A component that fetches and displays the list of books from the backend.
* **/utils**: Contains utility files like paths.ts, which holds the various paths (URLs) used in the app for routing, queries.ts which contains functions to fetch data from back-end and types.ts which contain common types used by application.

#### **Sample UI**

#### 

### **Database Overview**

The backend of the application relies on a MySQL database to store book-related information. This section provides an overview of the database schema used for storing books, including table creation scripts and a description of each field.  
  
**Database Schema**App relies on “Books” tables, that is generated automatically from DbSet<Books> in DbContext class using EF ORM. No manual involvement is required.