# System Architecture, Database Models and Testing Overview

## Architecture Overview

My application follows a clean architecture approach with API, Application, Domain and Infrastructure layers. The Controllers delegate business logic to the services, which rely on repository interfaces. DTOs and Automapper handle transformation and mapping.

Api Layer:

- Controllers (Authors, Books, Members, Loans, Dashboard)
- Routing, validation, global exception handling, logging.

Application Layer:

- Services containing business logic and validation.
- Repository interfaces
- Dtos and AutoMapper profile.

Domain Layer:

- Entities: Author, Book, Member, Loan.
- BaseEntity for shared things such as Id.

Infrastructure Layer

- EF Core, Sql Server.
- Repository implementations.
- DbContext and Migrations.

## Database Models.

I used a SQL Server database through Azure, which is why I haven't added SQL create database instructions, since you can access it right now.

- Author (1) → Books (many)
- Member (1) → Loans (many)
- Book (1) → Loans (many)

I added a special rule such as you can't delete an Author that has existing books.

## Testing Overview

## Unit Tests

**AuthorServiceTests**

- CRUD
- Exceptions

**BookServiceTests**

- Validation
- NotFound handling

**MemberServiceTests**

- Deletion
- NotFound handling

**LoanServiceTests**

- Book availability
- Member existence
- Return logic

**DashboardServiceTests**

- Count

- Aggregation

## Integration Tests

**AuthorsControllerTests**

- GetAll
- Create + GetById

**BooksControllerTests**

- GetAll
- Create

**MembersControllerTests**

- GetAllMembers
- CreateMember + GetById

**LoansControllerTests**

- CreateLoan
- ReturnBook

**DashboardControllerTests**

- GetDashboard

These tests verify routing, validation, controller behavior, persistence, and full end-to-end functionality.