**🎬 Video Movie Rental System – ASP.NET Core MVC**

This project is a web-based movie rental system built with ASP.NET Core MVC and Entity Framework Core. It allows users to browse, rent, and return movies, and managers to manage movie inventory.

**Overview**

This is an MVC application for a video movie rental system that allows users to:

* Browse available movies
* Login using a card number
* Rent available movies
* View and return their rented movies

**🧱 Application Layers**

**1. Domain Layer**

Contains core models and enums:

* User, Movie, Rental, Cast
* Shared base entity: BaseEntity (adds Id)
* Enumerations: Genre, Language, Part, SubscriptionType

**2. Database Layer**

* Uses **Entity Framework Core**
* VideoMovieRentDbContext: Holds DbSet<T> for all models.
* Seed data: Populates users, movies, casts, and rentals.

**3. Repository Layer**

Defines and implements data access contracts:

* IRepository<T>: Generic CRUD
* IMovieRepository, IUserRepository, IRentalRepository

**4. Service Layer**

* Business logic abstraction
* Interfaces: IMovieService, IUserService, IRentalService
* DTOs: MovieDto, MovieDetailsDto, RentalDto

**5. Controller Layer**

* MovieController: Handles login, browse, rent, return, view logic.

**6. Views**

* Razor Views for:
  + Login, Index (Movie list), Details, Return

🔁 Dependency Injection (in Program.cs)

// Configure maximum file size for form uploads (image upload readiness)

builder.Services.Configure<FormOptions>(options => {

options.MultipartBodyLengthLimit = 104857600; // 100 MB});

// Inject EF Core with SQL Server

builder.Services.AddDbContext<VideoMovieRentDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnString")));

// Repositories

builder.Services.AddScoped<IRepository<Movie>, MovieRepository>();

builder.Services.AddScoped<IUserRepository, UserRepository>();

builder.Services.AddScoped<IRentalRepository, RentalRepository>();

// Services

builder.Services.AddScoped<IMovieService, MovieService>();

builder.Services.AddScoped<IUserService, UserService>();

builder.Services.AddScoped<IRentalService, RentalService>();

// Enable Session for login simulation

builder.Services.AddSession();

**📁 Domain Models**

**BaseEntity.cs**

public abstract class BaseEntity

{

public int Id { get; set; } // Shared Id for all domain objects

}

**User.cs**

public class User : BaseEntity

{

public string FullName { get; set; } = null!;

public int Age { get; set; }

[Required] // Ensures CardNumber is provided

public string CardNumber { get; set; } = null!;

public DateTime CreatedOn { get; set; }

public bool IsSubscriptionExpired { get; set; }

public SubscriptionType SubscriptionType { get; set; }

}

**Movie.cs**

public class Movie : BaseEntity

{

public string Title { get; set; } = null!;

public Genre Genre { get; set; } // Enum-based genre classification

public Language Language { get; set; } // Enum-based language

public bool IsAvailable { get; set; } // Availability status

public DateTime ReleaseDate { get; set; }

public TimeSpan Length { get; set; }

public int AgeRestriction { get; set; }

public int Quantity { get; set; }

public string? ImagePath { get; set; } // Relative image path for UI

}

**📂 Repositories**

**IRepository<T>**

Generic CRUD contract:

public interface IRepository<T> where T : BaseEntity

{

IEnumerable<T> GetAll();

T GetById(int id);

void Create(T entity);

void Update(T entity);

void Delete(int id);

}

**UserRepository.cs**

public class UserRepository : IUserRepository

{

public User GetUserByCardNumber(string cardNumber)

{

return \_db.Users.FirstOrDefault(x => x.CardNumber == cardNumber);

}

}

**RentalRepository.cs**

public IEnumerable<Rental> GetRentalsByUserId(int userId)

{

// Only return not yet returned rentals

return \_db.Rentals.Where(r => r.UserId == userId && r.ReturnedOn == DateTime.MinValue).ToList();

}

**🔧 Services**

**MovieService.cs**

* Uses repository to return movie list and details
* Converts domain model to DTOs for views

**RentalService.cs**

* RentMovie(): Creates a rental, decrements movie quantity
* MarkAsReturned(): Marks rental returned, updates availability

**Controller**

// Main controller for movie operations

public class MovieController : Controller

{

// Actions for:

// - Listing movies (Index)

// - User login (Login)

// - Movie details (Details)

// - Renting movies (Rent)

// - Returning movies (Return, ReturnMovie)

// - Logout (Logout)

}

**🖼️ Views (Razor)**

**Login.cshtml**

* Accepts card number
* Displays error if not found

**Index.cshtml**

* Displays all movies using MovieDto
* Enables renting only when logged in

**Details.cshtml**

* Displays full movie info
* Allows renting based on login status

**Return.cshtml**

* Shows user’s currently rented movies
* Allows returning

**🎯 Functionality Flow**

* **🔐 Login Flow**

Login View → MovieController.Login() → UserService.GetUserByCardNumber()

→ If success → Redirect to Index (with userId)

* **🎬 Browse & Rent**

MovieController.Index() → MovieService.GetAllMovies() → index.cshtml

MovieController.Details() → MovieService.GetMovieDetails() → details.cshtml

MovieController.Rent() → RentalService.RentMovie()

* 📦 **Return**

MovieController.Return() → RentalService.GetRentalsByUserId() → return.cshtml

MovieController.ReturnMovie() → RentalService.MarkAsReturned()

**🧪 Sample Seed Data (DbContext)**

* **10 Users**
* **10 Movies**
* **10 Rentals**
* **10 Cast entries**

**🔍 Notes**

* **Login is card number only** (mock authentication).
* **Session-based login optional** — you can extend with HttpContext.Session.
* **Image paths** reference wwwroot/images/movie.