ASP.NET Core MVC Application Report

Murat Ali Alkan B221202003

December 15, 2024

Introduction

This report describes the development of an ASP.NET Core MVC application for a veterinary clinic. The clinic is named *HealthyPaws*, a fictional name created for this project. The application uses Entity Framework (EF) Core for managing PostgreSQL database queries (CRUD operations) and ASP.NET Core Identity for authorization. It includes a total of 31 models, with 20 corresponding to database tables, 3 custom validation attributes, 2 Dbcontext classes and 6 ViewModels for handling form submissions and specific views.

Database Design

The database consists of 20 tables, each serving a specific purpose within the application. Below is a summary of the tables and their relationships:

Tables

1. **AnimalType**
2. **City**
3. **District**
4. **Village**
5. **CompanyProduct**
6. **Food**
7. **Medicine**
8. **MedicineCategory**
9. **Product**
10. **Purchase**
11. **Sale**
12. **Stock**

13. **Specialization**
14. **SpecializationType**
15. **VetCustomer**
16. **Veterinarian**
17. **VetSpecialization**
18. **AnimalProduct**
19. **Company**
20. **Customer**

Relationships

- **SpecializationType - Specialization**: One-to-Many
- **Specialization - Veterinarian**: Many-to-Many (via *VetSpecialization*)
- **Customer - Veterinarian**: Many-to-Many (via *VetCustomer*)
- **Customer - Village**: One-to-Many
- **Village - District**: One-to-Many
- **District - City**: One-to-Many
- **Product - Company**: Many-to-Many (via *Purchase*)
- **Product - Company**: Many-to-Many (via *CompanyProduct*)
- **Product - Stock**: One-to-Many
- **Product - Animal**: Many-to-Many (via *AnimalProduct*)
- **Product - Sales**: One-to-Many
- **Medicine - MedicineCategory**: One-to-Many

Application Design

The application follows the MVC (Model-View-Controller) architecture and includes the following controllers:

Controllers

1. **AccountController**
    - Manages user authentication using ASP.NET Core Identity.
    - Endpoints:
        - Login (HTTP GET, HTTP POST)
        - Logout
        - Register (HTTP GET, HTTP POST)
    - Has two views:
        - **Login.cshtml**: Renders the login form for user authentication.
        - **Register.cshtml**: Renders the registration form for new user accounts.
2. **CompanyController**
    - Handles company-related operations.
    - Endpoints:
        - List
        - Add (HTTP GET, HTTP POST)
        - ListProducts (Uses ProductController's List View)
    - Includes two views:
        - **AddCompany.cshtml**: Provides a form for adding a new company.
        - **ListCompanies.cshtml**: Displays a list of existing companies.
3. **CustomerController**
    - Manages customer-related operations.
    - Endpoints:
        - List
        - Add (HTTP GET, HTTP POST)
        - ListVeterinarians (Uses VeterinarianController's List View)
    - Has two views:
        - **AddCustomer.cshtml**: Provides a form for adding a new customer.
        - **ListCustomers.cshtml**: Displays a list of customers.
4. **HomeController**
    - Provides a welcome page.
    - Endpoints:
        - Index
    - Has one view:
        - **Index.cshtml**: Welcomes the user to the application.
5. **ProductController**

- Manages product-related operations.
- Endpoints:
  - List
  - Add (HTTP GET, HTTP POST)
  - Update (HTTP GET, HTTP POST)
- Has three views:
  - **AddProduct.cshtml**: Provides a form for adding new products.
  - **ListProducts.cshtml**: Displays a list of all products.
  - **UpdateProduct.cshtml**: Renders a form for updating product prices.

6. **PurchaseController**
   - Handles purchase-related operations.
   - Endpoints:
     - List
     - Add (HTTP GET, HTTP POST)
   - Has two views:
     - **ListPurchases.cshtml**: Displays a list of purchase records.
     - **AddPurchase.cshtml**: Provides a form for recording a new purchase.
7. **SaleController**
   - Manages sale-related operations.
   - Endpoints:
     - List
     - Add (HTTP GET, HTTP POST)
   - Has two views:
     - **ListSales.cshtml**: Displays a list of sales records.
     - **AddSale.cshtml**: Provides a form for recording a new sale.
8. **StockController**
   - Handles stock-related operations.
   - Endpoints:
     - List
     - Delete
   - Has one view:
     - **ListStocks.cshtml**: Displays the current stock levels.
9. **VeterinarianController**
   - Manages veterinarian-related operations.
   - Endpoints:

  - List
  - Add (HTTP GET, HTTP POST)
  - ListCustomers (Uses CustomerController's List View)
- Has two views:
  - **AddVeterinarian.cshtml**: Provides a form for adding new veterinarians.
  - **ListVeterinarians.cshtml**: Displays a list of veterinarians.

Custom Validations

The application includes custom validation attributes to ensure data integrity:

- **ValidateAnimalTypesAttribute**: Validates a list of animal types in forms.
- **ValidateSpecializationsAttribute**: Validates a list of specializations in forms.
- **ValidateVeterinarianAttribute**: Validates a list of veterinarians in forms.

ViewModels

Several ViewModels were created to manage data submission and display for specific views:

- **CompanyViewModel**: Used for adding a new company.
- **CustomerViewModel**: Used for adding a new customer.
- **LoginViewModel**: Used for login functionality.
- **ProductViewModel**: Used for adding a new product.
- **RegisterViewModel**: Used for registration functionality.
- **VeterinarianViewModel**: Used for adding a new veterinarian.

DbContext Classes

Two DbContext classes were created for managing database interactions:

- **AppIdentityDbContext**: Handles operations related to the Identity database.
- **HealthyPawsContext**: Handles operations related to the *HealthyPaws* database.

Shared Views

The **Shared** folder contains views that are accessible across the application:

- **_Layout.cshtml**: Defines the common layout structure for all views, including the header, navigation, and footer.
- **Success.cshtml**: Provides user feedback, such as successful completion of actions like adding a new product or customer.

The **Shared** folder ensures consistency in design and reduces code duplication.

Special View Files

_ViewImports.cshtml

This file manages namespaces,Tag Helpers and injections used across views.

_ViewStart.cshtml

This file defines the layout file for all views by default, ensuring a uniform structure across the application.

Conclusion

This ASP.NET Core MVC application integrates EF Core for database management and ASP.NET Core Identity for user authentication. The application design, including its structured database schema, custom validation attributes, and ViewModels, ensures a modular and scalable solution for managing a veterinary clinic efficiently.