Assignment 2 Report .NET

Dicky Evaldo

Amy Yan

Tutorial 3

Table of Contents

Introduction and Summary

The Pet Shop Management system is a digital platform crafted to streamline pet shop services and enhance customer experiences. The web itself provides a range of services, including vet and check-up, pet grooming, and pet sitting. Additionally, the application also offers an adoption center for users and provides a comprehensive shop for pet food, accessories, and other necessities. Users can navigate between a Home Page, a Pet Store (with adoption options and a general store), an Order Page, a Checkout Page, a Pet Service Registration Form for scheduling services, and other aspects of the system.

This project aims to simplify the process and streamline for pet owners to access a range of pet services and pet's products such as food, accessories, and more, making it easier to manage their pet needs in one integrated platform. Customers and pet shop employees alike gain from it as it improves customer convenience, makes services more accessible, and supports effective pet care management. This web will be called Care Buddies.

Development Approach

Our Pet Shop Management System was developed using a combination of modern, robust frameworks and libraries, allowing us to build an efficient, maintainable, and scalable application. The stack that powers this system includes Blazor, ASP.NET, Tailwind CSS, and Entity Framework Core (EF Core), each playing a crucial role in different layers of the application.

- Blazor (Front-End)

For the front-end, we leveraged **Blazor**, a framework from Microsoft that allows developers to create rich, interactive web applications using C#. Blazor provides an alternative to traditional JavaScript-based frameworks like React or Angular by enabling C# to be used in both the client-side and server-side.

Implementation:

1. **Interactive Components**: In our project, we developed the user interface with Blazor's **Razor components**. Each page of the Pet Shop Management System, such as the pet services booking

page, product listings, and appointment scheduling, is created as a Blazor component (e.g., FormService.razor, FormServiceDoctor.razor, Shop.razor, Shop.Products._.razor, and more).
    - These components are reusable UI elements that encapsulate both HTML markup and C# logic, improving maintainability and code reuse.
2. **Client-Side Interactivity**: We used Blazor's **two-way data binding** feature to create dynamic and responsive pages. For example, the drop-down selections for pet type or services (like vet check-up or grooming) are powered by Blazor's binding, allowing us to capture user input in real-time without relying on JavaScript. Blazor handles the UI updates efficiently as users interact with forms and services.
3. **Routing**: Blazor's built-in routing system was used to navigate between different pages in the application. For instance, when users want to switch, they can easily switch between booking services, viewing product details, or checking their appointment confirmation without requiring a page refresh, enhancing the overall user experience.
4. **State Management**: Blazor's support for **state management** ensures that data like user selections or shopping cart details are preserved across pages during the user's session, even as they navigate between different components or refresh pages.

- ASP.NET (Back-End)

On the server-side, we employed **ASP.NET Core**, a powerful web framework by Microsoft, to handle the back-end logic, service administration, and API endpoints. ASP.NET Core provides a strong, scalable, and secure foundation for building modern web applications.

Implementation:

1. **Service Handling**: ASP.NET manages all back-end services, including business logic for handling pet appointments, managing inventory, and processing customer orders. For example, when a user books a grooming service, the ASP.NET server processes the request, checks available time slots, and updates the database accordingly.
2. **API Development**: We created several APIs that allowed communication between the front-end Blazor components and the back-end services. These APIs are responsible for handling tasks like submitting appointment bookings, retrieving available pet services, or fetching customer order

histories. The requests from the Blazor front-end are sent to the ASP.NET API controllers, which then process the requests and return the relevant data.

3. **Security and Authentication**: ASP.NET Core provides built-in support for **authentication and authorization** mechanisms. We implemented **ASP.NET Identity** to handle user registration, login, and role management, ensuring that sensitive operations (such as accessing admin pages or customer order data) are restricted to authenticated users only.
4. **Middleware Configuration**: ASP.NET allows the configuration of middleware components that handle HTTP requests and responses. In our project, we configured middleware for **exception handling**, **HTTPS redirection**, and **CORS policies**, ensuring secure and smooth communication between the client and the server.

- Tailwind CSS (Styling)

To style the user interface, we opted for **Tailwind CSS**, a utility-first CSS framework that allows for rapid and responsive design with minimal effort. Tailwind CSS provides low-level utility classes that we used to customize the appearance of components directly in the markup.

Implementation:

1. **Responsive Design**: Tailwind's utility classes (w-64, p-4, text-center, etc.) allowed us to create a responsive design that adjusts seamlessly across different devices and screen sizes. This ensures that our pet shop application is accessible and user-friendly whether accessed on a desktop, tablet, or mobile device.
2. **Custom UI Components**: We created custom-styled components such as buttons, dropdown menus, and cards using Tailwind's predefined classes. For example, the service booking form features a clean and modern look by using classes like border-gray-300 (for form borders) and rounded-md (for rounded corners), which enhances the user experience and visual consistency.
3. **Flexible Layouts**: Tailwind's flexible grid and flexbox utilities helped us design complex layouts without writing custom CSS from scratch. For example, the booking form layout is structured using flex and gap utilities to align form fields and buttons neatly and intuitively.

4. **Theme Customization**: We took advantage of Tailwind's **customization options** by configuring custom colors and spacing in the tailwind.config.js file. This allowed us to align the application's styling with the brand identity, creating a cohesive and professional design.

- Entity Framework Core (EF Core) - ORM

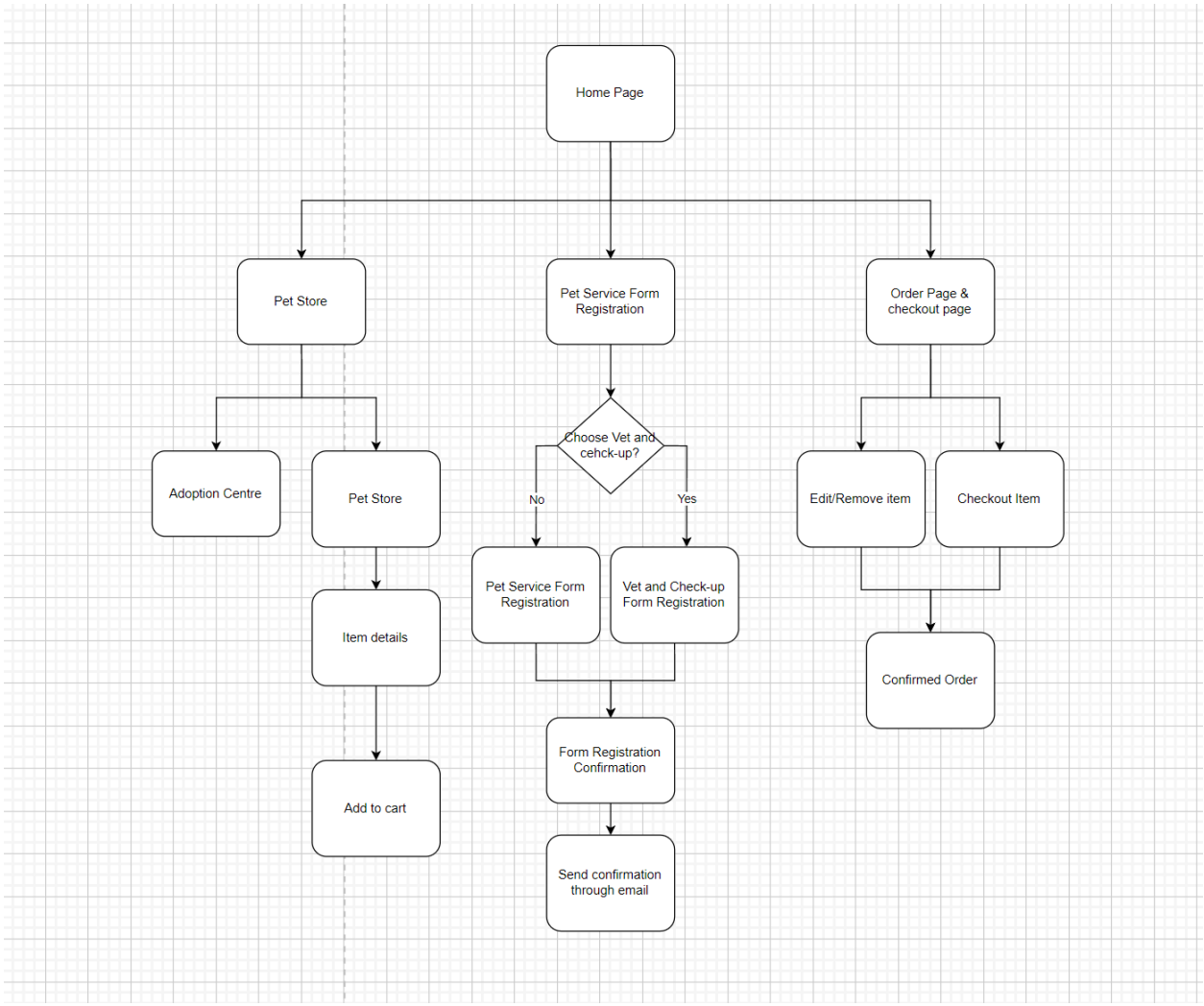
For the data layer, we used **Entity Framework Core (EF Core)**, a modern Object-Relational Mapper (ORM) for .NET. EF Core abstracts database access, allowing developers to work with database records as C# objects, significantly simplifying data management tasks.

Implementation:

1. **Database Models**: We defined C# **model classes** to represent entities like `Pet`, `Service`, and `Appointment`. These classes are mapped directly to the database tables, allowing us to perform CRUD (Create, Read, Update, Delete) operations on the database with ease.
2. **Database Migrations**: Using EF Core's **migration feature**, we managed database schema changes over time. Every time we added new features or updated the database schema (e.g., adding new fields to the `Service` entity), we used migrations to update the database without losing existing data. This made it easy to evolve the database structure as the project progressed.
3. **Queries and Data Manipulation**: EF Core allows us to write LINQ queries that interact with the database. For example, to fetch available time slots for a service, we use EF Core to query the `Appointment` table, ensuring only free slots are shown to the user. Similarly, booking an appointment is done by adding a new record to the `Appointment` table, all via EF Core, making the data access code much simpler and safer compared to raw SQL.
4. **Performance Optimization**: EF Core's built-in support for **lazy loading** and **query optimization** ensures that only the necessary data is loaded from the database, improving performance. For example, when retrieving a list of available services, EF Core loads only the necessary data fields, ensuring fast response times.

-

Flowchart



Flow chart details:

Home Page

- Introduction to services and available categories.
- Navigation to other sections.

Pet Store

- **Adoption Center**
    - Browse adoptable pets.
    - Add pets to the adoption cart.

- **Pet Product**
    - Browse pet food, accessories, and essentials.
    - **Item Details**
        - Add items to the shopping cart.
        - Add item to cart

Pet Service Form Registration

- Select desired service (Grooming, Pet Sitting, Vet Check-up).
- **Vet and Check-up registration Form**
    - Choose doctor
    - Fill out the registration form and submit.
    - **Form Registration Confirmation:** Display confirmation message on-screen.
    - **Send Confirmation via Email:** Automatically send registration confirmation details to the user's email.
- **Pet Service Registration Form**
    - Fill out the registration form and submit.
    - **Form Registration Confirmation:** Display confirmation message on-screen.
    - **Send Confirmation via Email:** Automatically send registration confirmation details to the user's email.

Order Page

- Review items and services added to the cart.
- Edit or remove selections.

Checkout Page

- Finalize purchase.
- Enter payment details and confirm order.

Roles

Team Member	Role	Responsibilities
Amy Yen	Front-End and Back-End Programmer	Focuses on developing the front-end interfaces, implementing the back-end functionalities, and ensuring a seamless integration between both. Amy focuses heavily on the Home page, shop page, and the back-end of all of the pages, making sure a smooth and streamlined flow for all pages.
Dicky Evaldo	UI designer and Front-End Programmer	Responsible for designing the user interface and assisting with programming tasks, particularly in the front-end development and user interactions. Dicky focuses on pages such as Service form and form confirmation, also refining design of all the pages making the front-end process easier.