**PetQuestV1 Project Documentation**

**Mattias Gijsels  
C# Developer junior course - Year 2**

**1. Project Overview**

PetQuestV1 is a web application where users can create virtual pets, take care of them and play games with their pets.

The main idea is simple: users sign up, create pets, play games with them, and see how their pets rank against others. Different users have different permissions - regular users can manage their pets, SuperUsers can view all the data and admins can control the whole system.

This project represents my first attempt at building a complete web application, and while it works well, I learned a lot about project planning and technical decisions along the way.

**2. What the Application Does**

**Main Features:**

* **User accounts**: People can register and log in safely.
* **Pet management**: Users can create and take care of virtual pets.
* **Games**: There is a virtual pet game and a Pong game to play.
* **Rankings**: Users can see how their pets compare to others.
* **Admin tools**: Administrators can manage users and pets.
* **Export reports**: SuperUsers can export views of the data to a .json or an excel file.

**User Types:**

* **Regular Users**: Can create pets and play games.
* **SuperUsers**: Same as regular users plus some extra reporting features
* **Administrators**: Can manage everything in the system including users, pets, species, and breeds. And they may also have pets.

**3. How It's Built**

**Project Structure**

I organized the code into three parts:

1. **Main Application (PetQuestV1)**: Contains all the web pages and main logic.
2. **Client Application (PetQuestV1.Client)**: My intended setup for the project was for it to be a blazor WASM project, but due time constraints I opted to change it for a server side only project.
3. **Database Setup Tool**: A separate program that sets up the database automatically.

**Technologies Used**

* **C# and ASP.NET Core Blazor**: For building the web application.
* **SQL Server**: To store all the data (users, pets, etc.).
* **Entity Framework**: Makes it easier to work with the database using C# code.
* **ASP.NET Identity**: Handles user login and security.
* **Bootstrap**: Makes the website look good on phones and computers.
* **ClosedXML**: For Excel export functionality.

**4. Database Design**

The database is pretty straightforward. The main parts are:

* **Users table**: Stores account information
* **Pets table**: Each pet belongs to a user and has properties like name, age, and species etc.
* **Species table**: Different types like Dog, Cat, Fish, Bird, Penguin
* **Breeds table**: Specific breeds for each species (like Golden Retriever for dogs)

Each user can have multiple pets, but each pet belongs to only one user. This is called a "one-to-many relationship" which we learned about in class.

I also implemented soft deletion, which means when something is "deleted," it is marked as deleted but stays in the database. This is useful for data recovery and audit purposes.  
The database is also set up with a **parent-child relationship** where deleting a parent automatically handles its associated children to prevent errors. This is achieved via the delete behaviour configuration.

**5. Main Components**

**Key Pages:**

* **Home.razor**: The first page users see, handles login.
* **MyPets.razor**: Where users can manage their pets.
* **VirtualPet.razor**: Interactive pet care game than can increase the advantage of a pet.
* **Pong.razor**: Classic Pong game where the advantage is used to create the paddle size of the player.
* **AdminControlPanel.razor**: Admin-only page for system management.
* **UsersAdminPanel.razor**: For managing user accounts and adding roles.
* **PetsAdminPanel.razor**: For managing all pets in the system
* **SpeciesAdminPanel.razor**: For managing pet species
* **BreedsAdminPanel.razor**: For managing pet breeds

Each page is a Blazor component with the code behind. This code behind pattern made the structure cleaner and easier for me, adhering to the separation of concern design principle.

**6. Security and User Management**

I used ASP.NET Identity for handling user accounts. This takes care of:

* Secure password storage (passwords are encrypted)
* User login and logout
* Different user roles (regular user, admin, SuperUser.)
* Protecting pages that only certain users should see

The system automatically checks if a user is logged in and what their role is before showing them certain pages and buttons on the navigation menu.

**7. Project Evolution and Technical Decisions**

**Architecture Changes**

Originally, I planned to build this as a fully Blazor WASM project, which would run mostly in the browser as it seemed the best choice for scalability and performance. However, as the deadline approached and I encountered some technical challenges, I made the decision to switch to a mainly server-side Blazor application. This decision was driven by time constraints and the complexity of getting authentication working properly with WASM.

This change means that most of the application logic runs on the server rather than in the user's browser. While this works well, it is not what I originally intended, and reverting back to the WASM setup is one of my planned improvements.

**Code Structure Learning**

During development, I realized I was repeating a lot of code between different components. This taught me the importance of creating reusable, modular components from the start. In future projects, I will focus more on creating shared services and components to avoid duplication in the future.

**8. Challenges I Faced**

**Technical Issues:**

* **Getting login to work properly**: It took some time to understand how ASP.NET Identity works, especially when trying to implement it with Blazor WASM.
* **Database relationships**: Making sure pets are properly connected to users and species.
* **JavaScript integration**: The Pong game needed some JavaScript, which was tricky to connect with Blazor.
* **Remake of the pong game**: The pong game needed a rework after testing it on my laptop which is significantly less powerful compared to my desktop pc. This meant that certain settings (mainly pushing for a 60 fps output) gave performance issues.  
  After a rework of the code and optimization, it worked on both systems.
* **Real-time updates**: Admin panels require manual page refresh to see changes, which is not ideal for user experience. I would like to add a DataChangeNotificationService class that would make event subscriptions possible for my admin panels.
* **Architecture decisions**: Switching from WASM to Server-side mid-project created some inconsistencies I would like to address in the future.

**Project Management Issues:**

* **Scope creep**: I kept adding features without properly planning them first.
* **Time estimation**: I underestimated how long certain features would take to implement.
* **Planning**: I did not think conceptually enough about how different parts would connect together from the beginning.

**What I Learned:**

* How to build a complete web application from start to finish.
* Working with databases using Entity Framework.
* User authentication and security basics.
* Component-based programming with Blazor.
* How different parts of a web application work together.
* The importance of proper project planning and scope management.
* Being realistic about my skill level and time constraints.

**9. Future Improvements and Planned Features**

**Immediate Technical Improvements:**

* **Event-based service**: Add a singleton service that all admin panels can subscribe to for real-time updates. I would use the singleton pattern for this to make the service persist across the entire application session. This would eliminate the need for admins to refresh pages manually. So when one admin panel modifies data, all other panels automatically refresh their data.
* **Deleted items management**: Add a button to admin panels where administrators can view all soft-deleted pets, breeds, species, and users for recovery purposes.
* **Hard delete functionality**: Implement permanent deletion for unused user accounts and pets to help with database management.
* **Export functionality**: Allow SuperUsers to export search results to JSON or Excel files for reporting purposes.
* **Architecture revert**: Convert the project back to its intended Blazor WASM setup for better performance and user experience.
* **Games**: In the future I would also like to add more games that use more of the pet’s stats.

**Code Quality Improvements:**

* **Better modularity**: Refactor repeated code into shared components and services
* **Improved component interconnectivity**: Better design of how components communicate with each other
* **Service layer**: Create a proper service layer to handle business logic separately from UI components

**Process Improvements:**

* **Better user stories**: Create more detailed user stories from the start to guide development
* **Scope management**: Define project scope more clearly and stick to it
* **Time estimation**: Be more realistic about implementation time and complexity

**10. Key Learnings**

**Technical Learnings:**

This project taught me how to work with modern web development technologies and gave me hands-on experience with database design, user authentication, and component-based architecture. I now understand how Entity Framework makes database operations much better and how ASP.NET Identity handles security concerns.

**Project Management Learnings:**

The most valuable lesson was learning about project planning and scope management. I learned that it is better to fully implement fewer features than to partially implement many features. I also learned the importance of being realistic about my skill level and the time required for complex features.

**Problem-Solving Skills:**

When I encountered technical challenges, I learned to make pragmatic decisions rather than sticking to the original plan regardless of consequences. The switch from WASM to Server-side Blazor was a good example of adapting to circumstances, even though it was not ideal.

**12. Conclusion**

This project helped me understand how to build a real web application using C#. While it has some limitations and areas for improvement, it successfully demonstrates the core concepts of web development including database design, user authentication, role-based authorization, and interactive user interfaces.

One of the most valuable parts was going deeper into how everything connects together - the database, the user interface, the security system, the data flow between layers and the business logic. It is quite different from the smaller console applications we started with, and more complex than I initially expected.

The challenges I faced, particularly around project planning and technical decision-making, were just as educational as the successful parts. These experiences will help me approach future projects more strategically and with better planning.