CSC 337 Final Project

Team Members:

1. Zachary Hansen
2. Sameeka Maroli
3. Jordan Demler

Sadly, a group member dropped the class, Alex Roy (Dec 10).

Overview:

Pocket Pond (Formerly known as Fish Kingdom) is a web game that allows players to:

1. Make an account (supported by the code in index.html)
2. Login in to your aquarium with one starter fish
3. Interact with your fish through cuddling and feeding to earn coins
4. Go to the shop (a modal and html powered by shopScript.js) and purchase new fish
5. Keep buying more fish to level up and become higher on the leaderboard
6. View a leaderboard of the person with the most coins and fish
7. View other users' aquariums!

Players begin with a single little fish and can interact with it through feeding and petting, all while collecting coins along the way. Coins can be used in the games shop to purchase new fish, encouraging the user to grow their tanks and level up.

A leaderboard can be opened that displays the top 3 users and their points and number of fish. The program has cute sprite work and dynamic animations, coin counter to monitor how you interact with fish, and a shop with different sea animals available for purchase.

Frontend:

The frontend is implemented using HTML, CSS, and JavaScript. It includes the following screens and features:

Screens

1. Opening Screen/ Login Screen
    - A welcoming splash screen with animated bubbles and a mer-russ.
    - Options to log in, sign up, or access the instructions screen.

        - Instruction option for the login screen explains the game mechanics, including feeding, earning coins, using the shop, and understanding the leaderboard.
2. Fish Tank Screen
    - A cute little aquarium graphic
    - The main gameplay area where users:
        - Feed and pet their fish (to earn coins)
        - View the coin counter
        - Go to shop and view leaderboard
    - Shop Screen (an html page powered by shopScript.js)
    - Allows users to buy new fish
3. Leaderboard
    - Shows the top players ranked by number of fish and coins
    - Allows users to visit the aquariums of the top 3 players.

Key Features

- Dynamic DOM Updates: Actions like feeding and petting update the coin counter without reloading the page.
- Animations: Include basic animations for fish movements and user interactions.

(A single sprite for an unlockable fish in the game store)

Backend

The backend is implemented using NodeJS, Express, MongoDB, and Mongoose. It will handle API requests, user data, and gameplay logic.

NPM Modules

- express: For server and routing.
- mongoose: For database interactions with MongoDB.
- body-parser: For parsing incoming requests.

Routes : The server will handle the following API routes:

1. **User Routes**:
    - POST /api/users/login: Log in an existing user.
    - POST /api/users/signup: Register a new user.
2. **Gameplay Routes**:
    - POST /api/fish/feed: Feed a fish and update its health.
    - POST /api/fish/cuddle: Cuddle a fish and earn coins.
3. **Leaderboard Routes**:
    - GET /api/leaderboard: Fetch top players and their scores.
4. **Trade Routes**:
    - POST /api/trade: Trade fish between two users.

Database Structure

The MongoDB database:

1. **Users**:
    - ➢ username: String
    - ➢ last accessed: String
    - ➢ coins: Number
    - ➢ level: Number
    - ➢ fish: Array of fish models
    - ➢ Decorations: Array of string names of Decorations
- ❖ **Fish**:
    - ➢ id: Unique identifier
    - ➢ name: String
    - ➢ type: String (e.g., small, medium, large)
    - ➢ health: Number(0-2)
    - ➢ beenFed: Number(0-2) I think fish should be fed 3 times before the are full
    - ➢ beenPet: Boolean
    - ➢ isHungry: Boolean
    - ➢ value: Number
    - ➢ accessories: Array of Strings (e.g., hats, sunglasses)
- ❖ **Leaderboard**:
    - ➢ username: String
    - ➢ fishValue: Number (sum of all fish values)

Reflections:

Due to the unexpected departure of a key team member who was responsible for a significant portion of the front-end development—specifically, the petting, feeding, and coin drop functionalities—we encountered considerable setbacks. On December 10th, we learned that this member would be dropping the class, leaving us with over two weeks of unfinished work and less than 48 hours to compensate. Despite our best efforts to divide the workload and produce a functional web game, it proved impossible to fully realize the original design.

As a result, we were forced to remove several planned features, including the trading system and the ability to customize fish with hats and accessories. Additionally, setting up the server presented multiple challenges, including CORS errors, favicon issues, and difficulties related to a teammate's malfunctioning laptop, which prevented them from installing essential tools like npm.

Ultimately, the project's final outcome differed considerably from our initial vision. Nevertheless, the experience broadened our understanding of HTML, JavaScript, and CSS, and we gained valuable insight into managing complex projects under pressure.