FINAL REPORT

Nolan, Ahbab, Richard, Ahmad, SJ

Final Video: https://youtu.be/NL2qo5UKaHg
Gitlab Link: https://gitlab.oit.duke.edu/ndp25/cs-316-proj
GitHub Link: https://github.com/2Ahmad5/MonPoke-Champions

Problem: We created a card based game with a supplemental website. The game and the website are separate but both access the same user accounts and MonPoke details.

Motivation: We were all video game enthusiasts with Richard actually having game experience. We thought it would be cool to merge the game aspect with the database aspects we learned in class.

Related works: We used other supplementary game websites as inspiration, though we have many more features on the website that would usually be reserved for in-game.

- League: https://lolprofile.net/
- Clash Royale: https://royaleapi.com/player/2JPPYVQJQ

Tables

- Users (id, firstname, lastname, email, username, #wins, #losses, #monPoke, balance, profile_pic)
- Owns(uid, monpoke_id)

- Monpoke (id, name, type1, type2, mana, health, damage, ability_id, rarity)
- Attacks(id, name, type, damage)
- Abilities(id, name, description)

- ShoppingCarts(uid, pid, name, price, quantity)
- Products(id, name, price, available, type)
- Purchases(id, uid, pid, time_purchased)
- Trades(requesterId, requesterName, cardWant, cardOffer)

- Forum_Posts(id, content, created_at, user_id, likes)
- Reply_likes(user_id, reply_id)
- Forum_replies(id, content, created_at, user_id, post_id)

User Related Features - Ahbab

We had three tables of relevance that the related to the profile:

- User - User data
- Owns - Stores UID and MonPoke_ID
- Monpoke - MonPoke Data.

Features

- **Public Access:** Non-logged-in users can browse the *MonDex* and the *Shop*, as these do not require user data.
- **Login Page:** Users input a username and password. The password is verified against the hashed version in the `Users` table. Includes a button to register new users.
- **Registration Page:** Users provide a unique email and username, matching passwords, and are added as a new row in the `Users` table.
- **Profile Page:** After logging in, users view their profile, showing their username, wins, games, MonPoke count, and a blank profile picture.
    - Displays collected MonPoke using a `JOIN` of the `Owns` and `MonPoke` tables.
    - Includes a button to edit the profile.
- **Edit Page:** Allows users to update their information.
    - Fields are pre-filled but editable. Users must re-enter their old password to save changes.
    - Email and username must remain unique.
    - Passwords can also be updated. Changes are applied to the existing `Users` row.
- **Players Page:**
    - Displays a list of all players with a search bar. Ranked by # wins.
    - Done through filtering the `Users` Table.
    - Clicking a name shows that player's profile.

Status: All Completed

Bonus Features added after MS4:

- Profile Picture,
- Editing profile information,
- Editing passwords.

Scrapped Bonus Idea:

- Match history.

Notable Design Choices / Approaches: I opted to store the number of MonPoke collected directly in the `Users` table, updating it manually whenever a user gains a new MonPoke.

An alternative approach was to compute the number of MonPoke owned by counting rows in the `Owns` table whenever needed, rather than storing it in the `Users` table. However, it would require scanning the potentially large `Owns` table for every profile view. The tradeoffs for manually updating Users is manageable as |Users| $\ll$ |Owns| and because a user can only acquire a limited number of new MonPoke.

Shop Related Features - Nolan

We had 6 tables of relevance to the shop:

- Owns
- Users
- Monpoke
- ShoppingCart
- Purchases
- Products

Features:

- **Shop**: The store displays all the products that are listed in the Products table. This involves getting tuples from the Products table to display.
- **Balance**: Users have a balance, which is taken from the Users table. This balance is updated when they purchase cards.
- **Adding Currency**: Users can add money through the credit card mockup, which adds currency to their balance in the Users table.
- **Purchase History**: Anytime a purchase is made, data is added to the Purchases table to track purchases
- **Cart**: Every user can add products to a cart that can be mass bought, in which products are added to the ShoppingCart table with corresponding uid.
- **Purchasing Cards**: When a card pack is purchased, it is drawn from the pool of cards from the database, with different chances for rarities. Purchased cards are added to the Owns table with the corresponding user.

Status: All Above Completed
No Bonus features completed

Design Choices: No notable design choices for the shop, the relations are fairly straightforward. We have a table with all the products including pid, name, cost, and type, ShoppingCart with all the pids for a given uid, Purchases for tracking history with date and time, uids, and product, and update Owns, Users, and access MonPokes as needed when purchasing and drawing cards.

Trade Related Features - Nolan

Bonus Feature Of Website
We had tables of relevance to the Trades:

- Trades
- Owns
- Monpoke

Features:

- Requesting Trades: Users can put up trades through a form, which checks with the MonPoke database for valid cards and the Owns table to make sure the user owns the card
- Accepting Trades: Users can accept trades, the Owns table is checked to make sure the users own the requested card. Corresponding cards are then added and removed from the traders in Owns.

Status: Trades are Bonus Features and Fully Implemented

Design Choices: When displaying the trades, we opted to share the username of the requester, but we also need the user_id to check and update the Owns table, so rather than query for a given user everytime we wish to display the name, we opted to just have both stored in the table.

MonDex Related Features - Ahmad

The MonDex has 1 associated table:

- MonPoke

Features

- **MonDex**: A dex for viewing all the MonPoke's available in the game. A search bar allows the user to search for partial names or full names to make it easier to view.
- **Types**: All the MonPokes will be listed with their corresponding types
- **MonPoke Profiles**: In the MonDex, users can click on the names of the MonPoke that will show a page of that MonPoke's info. The MonPoke's health, type, weakness, strengths, rarity, and ability will be displayed.
- Bonus: **Advantage Page**: When viewing any MonPoke's page, users can select another MonPoke from a dropdown to compare. We compare the health, attack, and typing affinities and use a custom function to calculate a relative advantage of a given card.

Status: All Realized

Design Choices: Fairly straightforward design of Table. The MonPoke table stores all of pid, name, rarity, ability, attack, health, mana cost for all the MonPokes. The MonDex queries this table for all the different information as needed.

Unity game(Richard Schulz)

In our main menu, users can input their username and password information and then click login. We used a flask api and Unity's UnityWebRequest feature to send a post request to our api, which returns information about what cards the user owns to the Unity environment. The UnityWebRequest calls the api link and sends a post request with the username and password of a user. The API checks to see if there is a matching user with that information, then returns that user's cards in a json file. The game then reads this json file and updates theefgv deck list for that player. Once in the game users can spawn their cards, start and end their turn, and place and attack with cards. Once a user wins, the game resets the scene to the menu and an api request is sent to update the database with the winner and loser of that match.

We never got around to the additional features like animations and sounds. We also changed how the matchmaking worked, rather than a user sending an invite, the user just creates a lobby and the other user joins. Additionally, there is no longer any inventory aspect of the game. You simply just have the cards from the database. One difficulty I faced when first approaching this project was figuring out how to access our postgres database through Unity. There is a plugin used for accessing postgres that I could not get to work for the life of me, so our group had to create a flask API to access the information through Unity.

feature	Completion status
Menu	Completed
Database interactions	Completed
Matchmaking functions	Changed from original, but completed
Gameplay functions	partial(minor bugs with spawning cards/ attack, mana cost, health values dont change)
Inventory	scrapped

Forum - SJ

Relavent Tables:

- Forum Posts - Details about each post
- Reply Likes - stores who liked what
- Forum Replies - Details about replies to posts

Forum Page Features

- **View Posts and Replies:** Users can view all posts and replies, along with their authors and timestamps, using the `Forum Posts` and `Forum Replies` tables.
- **Create Posts:** Users can add new posts, which updates the `Forum Posts` table.
- **Reply to Posts:** Users can reply to existing posts, which updates the `Forum Replies` table.
- **Like Posts:** Users can like posts, updating the `Reply Likes` table. A check ensures each user can only like a post once.

Status: All Above Completed

Bonus Features added after MS4:

- Upvote system
- Reply functionality

Srapped:

- FAQ page (could easily just make it a post on the forum)
- Rating Monpokes (could easily make it a forum post and upvote system)
- Submit Bug report - Bonus

Design Choice: Replies and posts are stored separately, with each reply containing a `post ID` to link it to the relevant post.

An alternative was to use a separate (posts, replies) relationship table, but this approach avoids the need for an extra table because each reply only has ONE post it replies to.

Reflection and Future Direction:

Evaluation: One issue with our system was integrating Unity with our Postgres database, which we had to find a workaround. Our method of connecting the database to the game was through Unity web requests that use api calls to our database to both get Users decks and to update their wins and losses. The game also has some minor bugs in each player's actions, which need to be fixed in the future. As for the rest of the database and the website, we successfully integrated the different aspects of the forum, dex, userprofile, shop, and trades into one system that is usable by players to enjoy the game with.

Future Direction:

- Make the game and website hostable online would make our project a reality. It would also be necessary to host the database somewhere public where it can serve clients worldwide. We would add back some of the scrapped ideas, such as match history.
- We would work on game balancing (something we mostly ignored) to try and make the game actually fun and fair, as well as introducing a better way of ranking players (currently we just use # wins).
- We would love to add in a ton more new cards, as 50 is not much.