Milestone 1: Project Proposal

Link to GitHub Repo

R1 - Application high level description

PocketTrader is a small inventory/collection manager for TCG (trading card game) collectors. The app maintains a catalog of cards (cardID, name, pack/set, rarity, type, imageURL, etc.) and lets authenticated users browse and filter the catalog, add one or more copies of a card to their personal collection, add cards to their public wishlist, and have the opportunity to find trades with other uses (i.e. identify what cards are on different users' collections and wishlist, then arrange a suitable trade, such that one or more players get cards on their wishlist). Typical users are players and collectors who want to track what they own and quickly find cards by pack, rarity, or type. Administrators are the app maintainers (us), who seed and manage the database.

As of milestone 1, we have implemented the ability to view/filter cards, and to add cards to the user's personal collection.

R2 - System support description

The frontend is built with Next.js (React) and Tailwind CSS, served on port 3000. The backend is a Flask (Python) app that exposes REST endpoints (ports: backend 5000) and executes SQL queries stored in app/backend/sql/ (one .sql file per query). The database is MySQL (5.7 in the provided Docker Compose), which holds tables like Card, User, and Collection. Docker Compose is used to orchestrate the three services (db, backend, frontend) so it runs consistently across OSes (containers run on Linux via Docker Desktop on Windows/macOS).

R3 - Database with sample dataset

We will use a Python script to fetch card metadata from the public TCGDEX Pocket API (https://tcgdex.dev/tcg-pocket). The script should normalize fields (cardID, name, packName, rarity, type, imageURL). We produce an artifact, which is a SQL seed file (e.g., app/database/migrations/init.sql) with INSERTs for Card, User, and sample Collection rows. For our sample data, we'll only include a small set of ~25 cards. When we create our full production database, we can use or extend this same script to gather ALL existing cards.

Here is our sample Card instance:

```
+--------+--------------+----------+--------+-----------+-------------------------------------------------+
| cardID | name         | packName | rarity | type      | imageURL                                        |
+--------+--------------+----------+--------+-----------+-------------------------------------------------+
| A1-001 | Bulbasaur    | Mewtwo   | 1D     | Grass     | https://assets.tcgdex.net/en/tcgp/A1/001/high.webp |
| A1-002 | Ivysaur      | Mewtwo   | 2D     | Grass     | https://assets.tcgdex.net/en/tcgp/A1/002/high.webp |
| A1-003 | Venusaur     | Mewtwo   | 3D     | Grass     | https://assets.tcgdex.net/en/tcgp/A1/003/high.webp |
| A1-004 | Venusaur ex  | Mewtwo   | 4D     | Grass     | https://assets.tcgdex.net/en/tcgp/A1/004/high.webp |
| A1-033 | Charmander   | Charizard | 1D    | Fire      | https://assets.tcgdex.net/en/tcgp/A1/033/high.webp |
| A1-034 | Charmeleon   | Charizard | 2D    | Fire      | https://assets.tcgdex.net/en/tcgp/A1/034/high.webp |
| A1-035 | Charizard    | Charizard | 3D    | Fire      | https://assets.tcgdex.net/en/tcgp/A1/035/high.webp |
| A1-036 | Charizard ex | Charizard | 4D    | Fire      | https://assets.tcgdex.net/en/tcgp/A1/036/high.webp |
| A1-053 | Squirtle     | Pikachu  | 1D     | Water     | https://assets.tcgdex.net/en/tcgp/A1/053/high.webp |
| A1-054 | Wartortle    | Pikachu  | 2D     | Water     | https://assets.tcgdex.net/en/tcgp/A1/054/high.webp |
| A1-055 | Blastoise    | Pikachu  | 3D     | Water     | https://assets.tcgdex.net/en/tcgp/A1/055/high.webp |
| A1-056 | Blastoise ex | Pikachu  | 4D     | Water     | https://assets.tcgdex.net/en/tcgp/A1/056/high.webp |
| A1-094 | Pikachu      | Pikachu  | 1D     | Lightning | https://assets.tcgdex.net/en/tcgp/A1/094/high.webp |
| A1-096 | Pikachu ex   | Pikachu  | 4D     | Lightning | https://assets.tcgdex.net/en/tcgp/A1/096/high.webp |
| A1-129 | Mewtwo ex    | Mewtwo   | 4D     | Psychic   | https://assets.tcgdex.net/en/tcgp/A1/129/high.webp |
| A1-227 | Bulbasaur    | Mewtwo   | 1S     | Grass     | https://assets.tcgdex.net/en/tcgp/A1/227/high.webp |
| A1-230 | Charmander   | Charizard | 1S    | Fire      | https://assets.tcgdex.net/en/tcgp/A1/230/high.webp |
| A1-232 | Squirtle     | Pikachu  | 1S     | Water     | https://assets.tcgdex.net/en/tcgp/A1/232/high.webp |
| A1-266 | Erika        | Charizard | 2S    | Trainer   | https://assets.tcgdex.net/en/tcgp/A1/266/high.webp |
| A1-267 | Misty        | Pikachu  | 2S     | Trainer   | https://assets.tcgdex.net/en/tcgp/A1/267/high.webp |
| A1-268 | Blaine       | Charizard | 2S    | Trainer   | https://assets.tcgdex.net/en/tcgp/A1/268/high.webp |
| A1-280 | Charizard ex | Charizard | 3S    | Fire      | https://assets.tcgdex.net/en/tcgp/A1/280/high.webp |
| A1-281 | Pikachu ex   | Pikachu  | 3S     | Lightning | https://assets.tcgdex.net/en/tcgp/A1/281/high.webp |
| A1-282 | Mewtwo ex    | Mewtwo   | 3S     | Psychic   | https://assets.tcgdex.net/en/tcgp/A1/282/high.webp |
| A1-285 | Pikachu ex   | Shared   | C      | Lightning | https://assets.tcgdex.net/en/tcgp/A1/285/high.webp |
| A1-286 | Mewtwo ex    | Shared   | C      | Psychic   | https://assets.tcgdex.net/en/tcgp/A1/286/high.webp |
+--------+--------------+----------+--------+-----------+-------------------------------------------------+
```

Here is our sample User instance:

```
+--------+----------+--------------+---------------------+
| userID | username | passwordHash | dateJoined          |
+--------+----------+--------------+---------------------+
|      1 | trainer  | password123  | 2025-10-20 13:30:00 |
+--------+----------+--------------+---------------------+
```

Here is our sample Collection instance:

```
+--------+--------+----------+---------------------+
| userID | cardID | quantity | dateAcquired        |
+--------+--------+----------+---------------------+
|      1 | A1-001 |        2 | 2025-10-22 02:22:36 |
|      1 | A1-003 |        1 | 2025-10-22 02:22:36 |
|      1 | A1-033 |        1 | 2025-10-22 02:22:36 |
|      1 | A1-035 |        1 | 2025-10-22 02:22:36 |
|      1 | A1-053 |        1 | 2025-10-22 02:22:36 |
|      1 | A1-055 |        1 | 2025-10-22 02:22:36 |
|      1 | A1-094 |        3 | 2025-10-22 02:22:36 |
|      1 | A1-096 |        1 | 2025-10-22 02:22:36 |
+--------+--------+----------+---------------------+
```

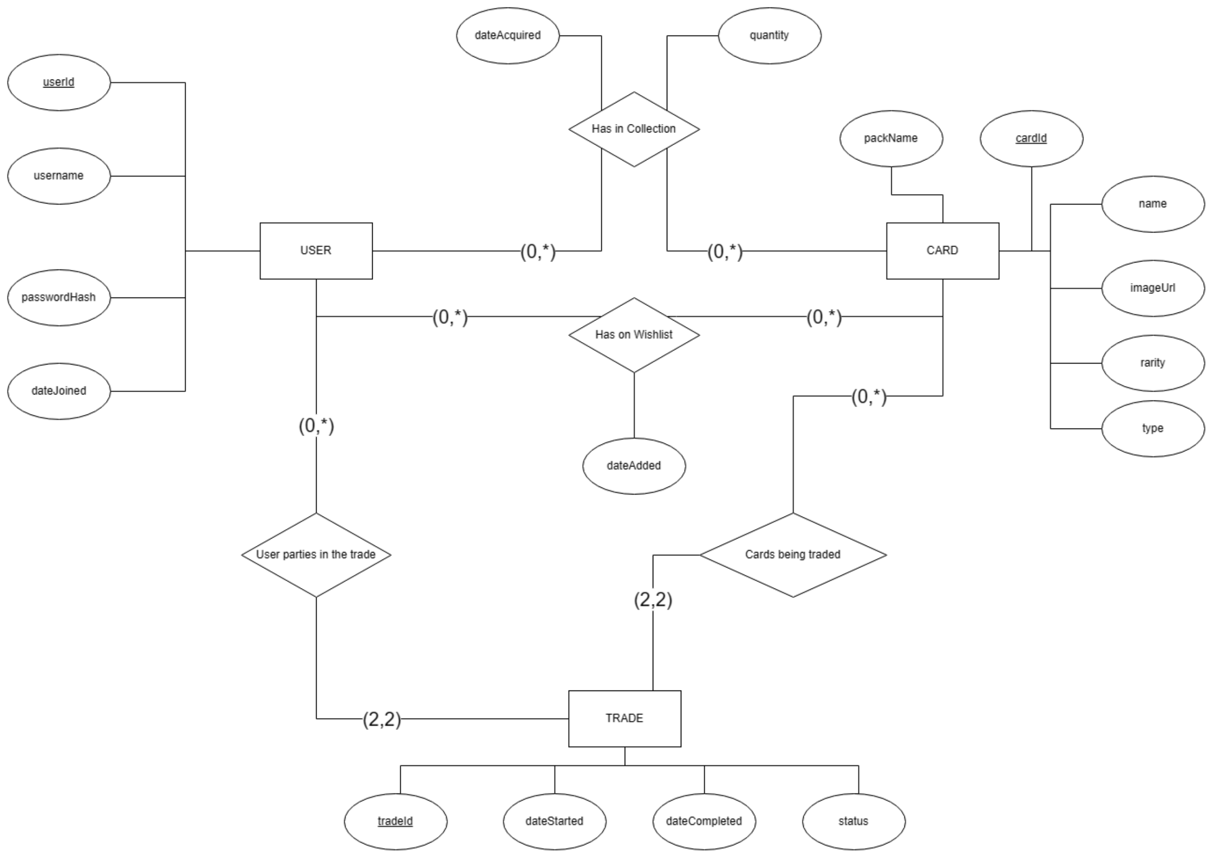
R5 - Design database schema

R5a - Assumptions

1. Each user has a unique user ID and username
2. Each Pokemon card has a unique card ID
3. Card rarity levels follows Pokemon TCG Pocket rarity levels (1-4 diamond, 1-2 star)
4. All cards have defined types (fire, water, grass, etc.)
5. A user can own multiple copies of the same card, tracked by a quantity field. Users can only trade cards they currently own
6. User can add multiple cards to their wishlist. The same card can appear on multiple users' wishlists.
7. A trade involves exactly 2 users and 2 cards
8. Both cards in trade should have the same rarity level
9. Trade statuses are: "pending", "accepted", "rejected", "cancelled", "completed"
    a. Only completed trades affect user collections
10. When trade is completed, card quantities are automatically updated in both users' collections
11. Users should not be able to delete their account if they have pending trades
12. Date fields should use format YYYY-MM-DD HH:MI:SS

R5b - E/R diagram

(See attached PDF for a more clearer picture)



R5c - Relational Data Model

User (<u>userId</u>, username, passwordHash, dateJoined)
Card (<u>cardId</u>, name, imageUrl, rarity, type, packName)
Trade (<u>tradeId</u>, dateStarted, dateCompleted, status)
Collection (<u>userId</u>, <u>cardId</u>, dateAcquired, quantity)

- PK: (<u>userId</u>, <u>cardId</u>)
- FK: userId references User
- FK: cardId references Card

Wishlist (<u>userId</u>, <u>cardId</u>, dateAdded)

- PK: (<u>userId</u>, <u>cardId</u>)
- FK: userId references User
- FK: cardId references Card

TradeParticipant (<u>tradeId</u>, <u>userId</u>)

- PK: (<u>tradeId</u>, <u>userId</u>)
- FK: tradeId references Trade
- FK: userId references User
- Check: Each tradeId must have exactly 2 rows

TradedCard (<u>tradeId</u>, <u>cardId</u>)

- PK: (<u>tradeId</u>, <u>cardId</u>)

- FK: tradeId references Trade
- FK: cardId references Card
- Check: Each tradeId must have exactly 2 rows

R6 - Basic Feature/Functionality 1: Browse & Filter my collection

R6-a:

This feature allows users to browse and filter their cards. The user is any logged-in player.

Flow: User selects optional filters (rarity/type/text search) → Clicking search renders a table of matching cards with quantities → Clicking a row opens a card detail panel.

R6-b:

Query template:

```
-- :userId, :rarityOpt, :typeOpt, :packOpt, :nameSearchOpt are parameters
SELECT c.cardID, c.name, c.rarity, c.type, col.quantity
FROM COLLECTION col
JOIN CARD c ON c.cardID = col.cardID
WHERE col.userID = :userId
  AND (:rarityOpt IS NULL OR c.rarity = :rarityOpt)
  AND (:typeOpt   IS NULL OR c.type   = :typeOpt)
  AND (:packOpt   IS NULL OR c.packName = :packOpt)
  AND (:nameSearchOpt IS NULL OR c.name LIKE CONCAT('%%', :nameSearchOpt,
'%%'))
ORDER BY c.rarity, c.name;
```

Sample run:

```
SELECT c.cardID, c.name, c.rarity, col.quantity
FROM COLLECTION col
JOIN CARD c ON c.cardID = col.cardID
WHERE col.userID = 1 AND c.rarity = '1D'
ORDER BY c.name;
```

Expected output:

```
cardID | name       | rarity | quantity
-------+------------+--------+---------
P-001  | Charmander | 1D     | 2
```

R7 - Basic Feature/Functionality 2: Add a card to my collection

R7-a:

This feature allows a user to add a card to their collection. The user is any logged-in user.

Flow: The user navigates to the “Add to Collection” page. They search for a Pokémon card by name, enter a quantity, and click “Add Card.” If the card already exists in their collection, we increase the quantity. Otherwise, we add a new record. The app then displays a confirmation message and updates the visible collection table.

R7-b:

Query template:

```
INSERT INTO COLLECTION(userID, cardID, quantity, dateAcquired)
VALUES (:userID, :cardID, :quantity, NOW())
ON DUPLICATE KEY UPDATE quantity = quantity + VALUES(quantity);
```

Sampled query:

```
INSERT INTO COLLECTION(userID, cardID, quantity, dateAcquired)
VALUES (1, 'A1-001', 1, '2025-10-20 13:30:00')
ON DUPLICATE KEY UPDATE quantity = quantity + VALUES(quantity);

-- Verify
SELECT userID, cardID, quantity FROM COLLECTION WHERE userID=1 AND
cardID='A1-001';
```

Expected output:

```
userID | cardID | quantity
-------+--------+---------
1      | P-001  | 1
```

R8 - Basic Feature/Functionality 3: Wishlist availability

R8-a:

This feature allows a user to see other users who have cards they want. The user is any logged-in player on their Wishlist page.

Flow: The page lists each wishlist card with Owners (count of distinct users other than the logged-in user who own ≥1 copy) and Total Copies (sum of those copies). Clicking a card reveals a modal with the list of owners’ usernames.

R8-b:

Query template:

```
-- :userId is the viewer
SELECT w.cardID, c.name,
       COUNT(DISTINCT col.userID)           AS owners,
       COALESCE(SUM(col.quantity), 0)       AS total_copies
FROM WISHLIST w
JOIN CARD c ON c.cardID = w.cardID
LEFT JOIN COLLECTION col
       ON col.cardID = w.cardID AND col.userID <> :userId
WHERE w.userID = :userId
GROUP BY w.cardID, c.name
ORDER BY owners DESC, c.name;
```

Sampled Query:

```
SELECT w.cardID, c.name,
       COUNT(DISTINCT col.userID)     AS owners,
       COALESCE(SUM(col.quantity), 0) AS total_copies
FROM WISHLIST w
JOIN CARD c ON c.cardID = w.cardID
LEFT JOIN COLLECTION col
  ON col.cardID = w.cardID AND col.userID <> 1
WHERE w.userID = 1
GROUP BY w.cardID, c.name
ORDER BY owners DESC, c.name;
```

Expected output:

```
cardID | name      | owners | total_copies
-------+-----------+--------+-------------
P-002  | Squirtle  | 1      | 2
P-003  | Bulbasaur | 1      | 1
```

R9 - Basic Feature/Functionality 4: Find mutual trade matches

R9-a:

This feature allows a user A to find users B satisfying A wants X, B has X, B wants Y, and A has Y, with X and Y having the same rarity. The user is any logged-in user on the Find Matches page.

Flow: The user clicks Find Mutual Matches. The result cards show: Partner, They give → I want, I give → They want. Only pairs where both cards share the same rarity are shown.

R9-b:

Query template:

```sql
-- :me is the logged-in userId
SELECT
  other.userID           AS partnerID,
  other.username         AS partnerName,
  wantMine.cardID        AS iWant_cardID,
  c1.name                AS iWant_name,
  c1.rarity              AS rarity_required,
  wantTheirs.cardID      AS theyWant_cardID,
  c2.name                AS theyWant_name
FROM USER me
JOIN USER other ON other.userID <> me.userID
-- I want a card the other owns
JOIN WISHLIST w_me        AS wantMine   ON wantMine.userID = me.userID
JOIN COLLECTION col_other               ON col_other.userID = other.userID AND col_other.cardID = wantMine.cardID AND col_other.quantity > 0
JOIN CARD c1                            ON c1.cardID = wantMine.cardID
-- They want a card I own
JOIN WISHLIST w_other     AS wantTheirs ON wantTheirs.userID = other.userID
JOIN COLLECTION col_me                  ON col_me.userID = me.userID AND col_me.cardID = wantTheirs.cardID AND col_me.quantity > 0
JOIN CARD c2                            ON c2.cardID = wantTheirs.cardID
-- Rarity must match
WHERE me.userID = :me
  AND c1.rarity = c2.rarity
ORDER BY other.username, c1.name, c2.name;
```

Sampled query:

```sql
SELECT
  other.userID           AS partnerID,
  other.username         AS partnerName,
  wantMine.cardID        AS iWant_cardID,
  c1.name                AS iWant_name,
  c1.rarity              AS rarity_required,
  wantTheirs.cardID      AS theyWant_cardID,
  c2.name                AS theyWant_name
FROM USER me
JOIN USER other ON other.userID <> me.userID
JOIN WISHLIST wantMine  ON wantMine.userID = me.userID
JOIN COLLECTION col_other
  ON col_other.userID = other.userID
 AND col_other.cardID = wantMine.cardID
 AND col_other.quantity > 0
JOIN CARD c1 ON c1.cardID = wantMine.cardID
JOIN WISHLIST wantTheirs ON wantTheirs.userID = other.userID
JOIN COLLECTION col_me
  ON col_me.userID = me.userID
 AND col_me.cardID = wantTheirs.cardID
 AND col_me.quantity > 0
JOIN CARD c2 ON c2.cardID = wantTheirs.cardID
WHERE me.userID = 1
  AND c1.rarity = c2.rarity
ORDER BY other.username, c1.name, c2.name;
```

Expected output:

```
partnerID | partnerName | iWant_cardID | iWant_name | rarity_required | theyWant_cardID | theyWant_name
----------+-------------+--------------+------------+-----------------+-----------------+--------------
2         | Bob         | P-002        | Squirtle   | 1D              | P-001           | Charmander
3         | Chloe       | P-003        | Bulbasaur  | 1D              | P-001           | Charmander
```

R10 - Sign up and Login

R10-a:

This feature lets users sign up and login to the platform.

Flow: Sign up and log in buttons appear on the platform if the user is not already signed in. When each one of the buttons is pressed, a form opens up. For signup, the passwords are checked if they match, the username is checked to see if it doesn't exist already, and the columns of the user table are filled out. For login, the username and password hash are compared with the stored one.

R10-b

R16 - Members

- **Jonathan Polina -** 4 features' description (R6ab - R9ab)
- **Steven Wu -** Application code
- **Brasen Xu -** Designed database scheme (R5)
- **Samuel Zheng -** R1-R3 project proposal, application code, gathering the sample dataset, sample SQL queries, putting together README instructions