Project Phase II - Relational Data Modelling

Member: Brianna Pinson

Problem Statement:

My project idea is a trading card database, specifically for Riftbound, the upcoming trading card game from Riot Games. With the card game releasing in October, there will be growing interest from collectors and players who want to keep track of their cards. Currently, there are very few applications available for Riftbound collectors and I would like my application to be a structured and reliable tool.

The primary purpose of this database is to aid collectors by providing a system to log and manage cards, including multiple copies. Users will be able to view their collections in an organized format and query or filter cards based on specific attributes. A secondary goal is to provide detailed descriptions for each card in the database. Attributes will include card name, artist, release date, rarity, card type, energy, might, power, and additional features such as alternative arts or finishes (eg. foil, overnumbered). This trading card database will be designed as a normalized relational database using ER modeling, with core functions for managing, searching and organizing cards and user collections. The long-term intention for this project is to deliver a fully operational card database with potential recognition as a web application. By the end of the semester, users will be able to browse for cards, maintain their collections, and, if time allows, build and manage their own decks.

Conceptual Database Design

Entities and Attributes

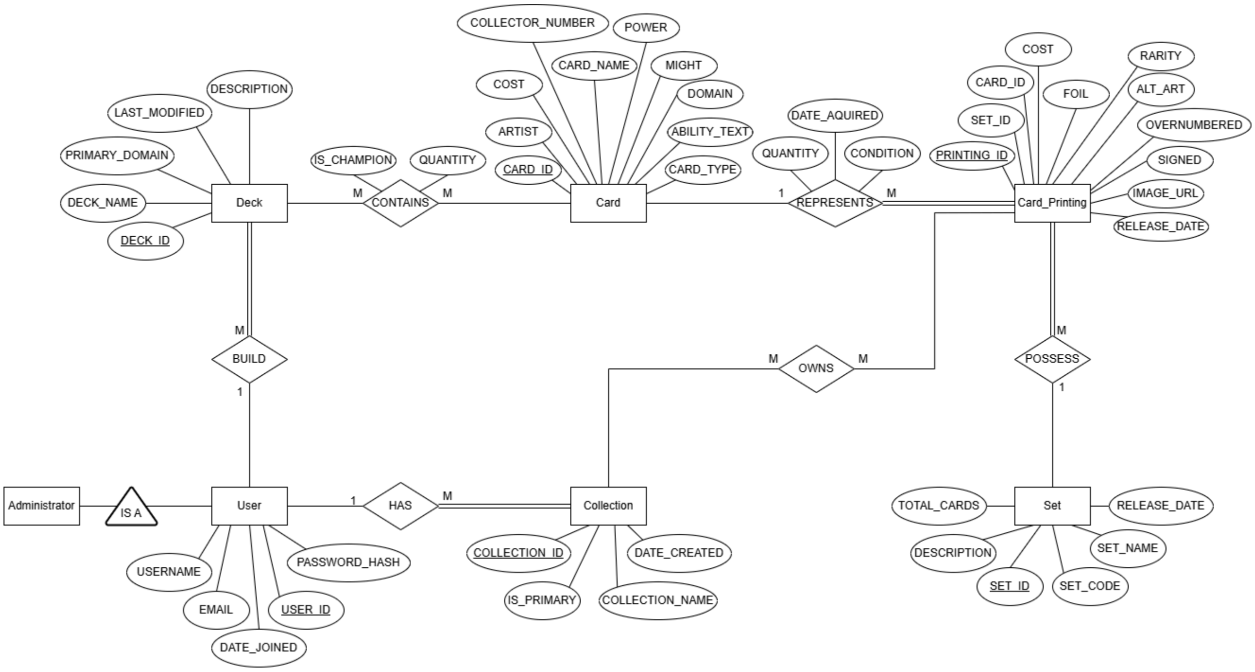
User: Represents a user of the application and is a superclass. Attributes include user ID, username, password hash, email, date joined.

Collection: Represents a user's card collection. Attributes include collection ID, collection name, date created, and is_primary.

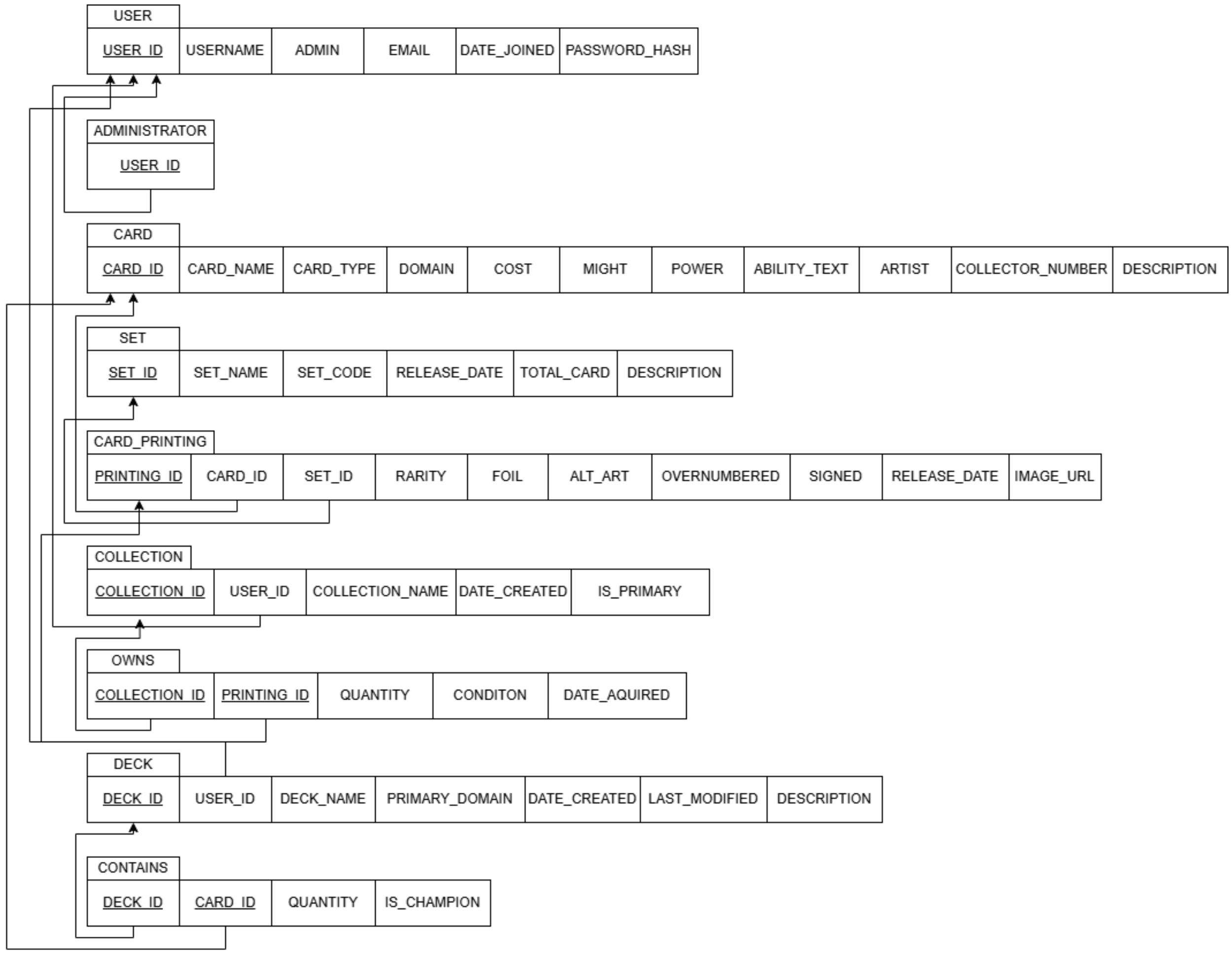
Card: Represents an individual unique card design. Attributes include card ID, card name, artist domain, might power, collector number, cost, ability text, and card type.

Card_Printing: Represents a specific printing or version of a card. Attributes include card ID, set ID, cost, rarity, foil, alt art, overnumbered flag, signed flag, image URL, release date, quantity, date acquired, and condition.

Set: Represents a set in which cards are published or part of. Attributes include set ID, set name, set code, release date, description, and total cards.



Logical Database Design



USER

USER_ID	USERNAME	ADMIN	EMAIL	DATE_JOINED	PASSWORD_HASH

ADMINISTRATOR

USER_ID

CARD

CARD_ID	CARD_NAME	CARD_TYPE	DOMAIN	COST	MIGHT	POWER	ABILITY_TEXT	ARTIST	COLLECTOR_NUMBER	DESCRIPTION

SET

SET_ID	SET_NAME	SET_CODE	RELEASE_DATE	TOTAL_CARD	DESCRIPTION

CARD_PRINTING

PRINTING_ID	CARD_ID	SET_ID	RARITY	FOIL	ALT_ART	OVERNUMBERED	SIGNED	RELEASE_DATE	IMAGE_URL

COLLECTION

COLLECTION_ID	USER_ID	COLLECTION_NAME	DATE_CREATED	IS_PRIMARY

OWNS

COLLECTION_ID	PRINTING_ID	QUANTITY	CONDITON	DATE_AQUIRED

DECK

DECK_ID	USER_ID	DECK_NAME	PRIMARY_DOMAIN	DATE_CREATED	LAST_MODIFIED	DESCRIPTION

CONTAINS

DECK_ID	CARD_ID	QUANTITY	IS_CHAMPION

Summary Table of Data Types

TABLE	ATTRIBUTES	TYPE	CONSTRAINT
USERS	USER_ID	SERIAL	PRIMARY KEY
USERS	USERNAME	VARCHAR(50)	NOT NULL, UNIQUE
USERS	EMAIL	VARCHAR(100)	NOT NULL, UNIQUE
USERS	PASSWORD	VARCHAR(255)	NOT NULL
USERS	DATE_JOINED	DATE	NOT NULL, DEFAULT CURRENT_DATE
CARD	CARD_ID	SERIAL	PRIMARY KEY
CARD	CARD_NAME	VARCHAR(100)	NOT NULL
CARD	CARD_TYPE	VARCHAR(20)	NOT NULL
CARD	DOMAIN	VARCHAR(5)	NULL
CARD	ENERGY	INTEGER	CHECK (ENERGY >= 0)
CARD	MIGHT	INTEGER	CHECK (MIGHT >= 0)
CARD	POWER	INTEGER	CHECK (POWER >= 0)
CARD	ABILITY_TEXT	TEXT	NULL
CARD	ARTIST	VARCHAR(100)	NOTNULL
CARD	COLLECTOR_NUMBER	VARCHAR(10)	NOT NULL
CARD	FLAVOR_TEXT	SERIAL	NULL
CARD_SET	SET_ID	INTEGER	PRIMARY KEY
CARD_SET	SET_NAME	VARCHAR(100)	NOT NULL
CARD_SET	SET_CODE	VARCHAR(10)	NOT NULL, UNIQUE
CARD_SET	RELEASE_DATE	DATE	NOT NULL
CARD_SET	TOTAL_CARDS	INTEGER	CHECK (TOTAL_CARDS > 0)
CARD_SET	DESCRIPTION	TEXT	NULL

Summary Table of Data Types (CONT. 1)

TABLE	ATTRIBUTES	TYPE	CONSTRAINT
CARD_PRINTING	PRINTING_ID	VARCHAR(20)	PRIMARY KEY
CARD_PRINTING	CARD_ID	INTEGER	NOT NULL, FOREIGN KEY
CARD_PRINTING	SET_ID	INTEGER	NOT NULL, FOREIGN KEY
CARD_PRINTING	RARITY	VARCHAR(20)	NOT NULL
CARD_PRINTING	FOIL	BOOLEAN	DEFAULT FALSE
CARD_PRINTING	ALT_ART	BOOLEAN	DEFAULT FALSE
CARD_PRINTING	OVERNUMBERED	BOOLEAN	DEFAULT FALSE
CARD_PRINTING	SIGNED	BOOLEAN	DEFAULT FALSE
CARD_PRINTING	RELEASE_DATE	DATE	NOT NULL
CARD_PRINTING	IMAGE_URL	VARCHAR(255)	NULL
COLLECTION	COLLECTION_ID	SERIAL	PRIMARY KEY
COLLECTION	USER_ID	INTEGER	NOT NULL, FOREIGN KEY
COLLECTION	COLLECTION_NAME	VARCHAR(100)	DEFAULT ‘My Collection’
COLLECTION	DATE_CREATED	DATE	NOT NULL
COLLECTION	IS_PRIMARY	BOOLEAN	DEFAULT TRUE
OWNS	COLLECTION_ID	INTEGER	PRIMARY KEY, FOREIGN KEY
OWNS	PRINTING_ID	INTEGER	PRIMARY KEY, FOREIGN KEY
OWNS	QUANTITY	INTEGER	NOT NULL, CHECK (QUANTITY > 0)
OWNS	CONDITION	VARCHAR(20)	NOT NULL
OWNS	DATE_ACQUIRED	DATE	
CONTAINS	CARD_ID	INTEGER	PRIMARY KEY, FOREIGN KEY
CONTAINS	QUANTITY	INTEGER	NOT NULL, CHECK (QUANTITY > 0)
CONTAINS	IS_CHAMPION	BOOLEAN	DEFAULT FALSE

Application program design

1. User Authentication

```
Login()
        Input: username, password
        Output: Login success/failure message
        Process:
          username = prompt for username
          password = prompt for password

          user = query USERS table WHERE USERNAME = username

          if (user exists)
            if (VerifyPassword(password, user.PASSWORD))
              logged_in = true
              current_user_id = user.USER_ID
              display "Login successful"
              return user.USER_ID
            else
              display "Incorrect password"
              return null
            end if
          else
            display "User not found"
            return null
          end if

Register_User()
        Input: username, email, password
        Output: Registration success/failure message
        Process:
          username = prompt for username
          email = prompt for email
          password = prompt for password

          // Check for duplicate username
          existing_user = query USERS table WHERE USERNAME = username
          if (existing_user exists)
            display "Username already exists"
            return false
          end if

          // Check for duplicate email
          existing_email = query USERS table WHERE EMAIL = email
          if (existing_email exists)
```

```
            display "Email already exists"
            return false
          end if

          // Hash password for security
          password_hash = hash(password)

          // Create new user
          INSERT INTO USERS (USERNAME, EMAIL, PASSWORD, DATE_JOINED)
          VALUES (username, email, password_hash, CURRENT_DATE)

          new_user_id = get last inserted USER_ID

          // Create default collection for new user
          INSERT INTO COLLECTION (USER_ID, COLLECTION_NAME, DATE_CREATED, IS_PRIMARY)
          VALUES (new_user_id, 'My Collection', CURRENT_DATE, TRUE)

          display "Registration successful"
          return true

Logout()
        Process:
          logged_in = false
          current_user_id = null
          display "Logged out successfully"
```

2. Collection Management

```
Add_Card_To_Collection()
        Input: card_name, set_name, foil (boolean), quantity, condition
        Output: Success/failure message
        Process:
          card_name = prompt for card name
          set_name = prompt for set name
          foil = prompt for foil (yes/no)
          quantity = prompt for quantity (default 1)
          condition = prompt for condition

          // Find the specific printing
          printing = query CARD_PRINTING
                JOIN CARD ON CARD_PRINTING.CARD_ID = CARD.CARD_ID
                JOIN CARD_SET ON CARD_PRINTING.SET_ID = CARD_SET.SET_ID
                WHERE CARD.CARD_NAME = card_name
                AND CARD_SET.SET_NAME = set_name
                AND CARD_PRINTING.FOIL = foil

          if (printing not found)
            display "Card printing not found in database"
            return false
          end if

          // Get user's primary collection
          collection = query COLLECTION
                  WHERE USER_ID = current_user_id
                  AND IS_PRIMARY = TRUE

          // Check if card already in collection
          existing_card = query COLLECTION_CARD
                    WHERE COLLECTION_ID = collection.COLLECTION_ID
                    AND PRINTING_ID = printing.PRINTING_ID

          if (existing_card exists)
            // Update quantity
            UPDATE COLLECTION_CARD
            SET QUANTITY = QUANTITY + quantity
            WHERE COLLECTION_ID = collection.COLLECTION_ID
            AND PRINTING_ID = printing.PRINTING_ID

            display "Card quantity updated"
          else
            // Add new card to collection
            INSERT INTO COLLECTION_CARD (COLLECTION_ID, PRINTING_ID, QUANTITY,
        CONDITION, DATE_ADDED)
            VALUES (collection.COLLECTION_ID, printing.PRINTING_ID, quantity, condition,
        CURRENT_DATE)

            display "Card added to collection"
          end if
          return true

Edit_Card_Quantity()
        Input: printing_id, quantity_change (+ or -)
        Output: Updated quantity\
        Process:
          printing_id = selected card printing
          quantity_change = increment (+1) or decrement (-1)
```

```
      // Get user's primary collection
      collection = query COLLECTION
            WHERE USER_ID = current_user_id
            AND IS_PRIMARY = TRUE

      // Get current card entry
      card_entry = query COLLECTION_CARD
            WHERE COLLECTION_ID = collection.COLLECTION_ID
            AND PRINTING_ID = printing_id

      if (card_entry not found)
        display "Card not found in collection"
        return false
      end if

      new_quantity = card_entry.QUANTITY + quantity_change

      if (new_quantity <= 0)
        // Remove card if quantity reaches 0
        DELETE FROM COLLECTION_CARD
        WHERE COLLECTION_ID = collection.COLLECTION_ID
        AND PRINTING_ID = printing_id

        display "Card removed from collection"
      else
        // Update quantity
        UPDATE COLLECTION_CARD
        SET QUANTITY = new_quantity
        WHERE COLLECTION_ID = collection.COLLECTION_ID
        AND PRINTING_ID = printing_id

        display "Quantity updated to " + new_quantity
      end if
      return true

Remove_Card_From_Collection()
    Input: printing_id
    Output: Success/failure message
    Process:
      printing_id = selected card to remove

      collection = query COLLECTION
            WHERE USER_ID = current_user_id
            AND IS_PRIMARY = TRUE
```

```
          card_entry = query COLLECTION_CARD
                   WHERE COLLECTION_ID = collection.COLLECTION_ID
                   AND PRINTING_ID = printing_id

          if (card_entry not found)
             display "Card not found in collection"
             return false
          end if

          DELETE FROM COLLECTION_CARD
          WHERE COLLECTION_ID = collection.COLLECTION_ID
          AND PRINTING_ID = printing_id

          display "Card removed from collection"
          return true

View_Collection()
        Input: Optional filters (rarity, domain, type), sort preference
        Output: List of cards in collection with details
        Process:
          filter_rarity = optional rarity filter
          filter_domain = optional domain filter
          filter_type = optional card type filter
          sort_by = sort preference (name/rarity/date)

          collection = query COLLECTION
                   WHERE USER_ID = current_user_id
                   AND IS_PRIMARY = TRUE

          // Base query with all necessary joins
          results = query CARD.CARD_NAME, CARD.CARD_TYPE, CARD.DOMAIN,
                     CARD.ENERGY, CARD.POWER, CARD.MIGHT,
                     CARD_SET.SET_NAME, CARD_PRINTING.RARITY,
                     CARD_PRINTING.FOIL, CARD_PRINTING.IMAGE_URL,
                     COLLECTION_CARD.QUANTITY, COLLECTION_CARD.CONDITION,
                     COLLECTION_CARD.DATE_ADDED
                 FROM COLLECTION_CARD
                 JOIN CARD_PRINTING ON COLLECTION_CARD.PRINTING_ID =
        CARD_PRINTING.PRINTING_ID
                 JOIN CARD ON CARD_PRINTING.CARD_ID = CARD.CARD_ID
                 JOIN CARD_SET ON CARD_PRINTING.SET_ID = CARD_SET.SET_ID
                 WHERE COLLECTION_CARD.COLLECTION_ID = collection.COLLECTION_ID
```

```
      // Apply filters
      if (filter_rarity not null)
        results = filter WHERE CARD_PRINTING.RARITY = filter_rarity
      end if
      if (filter_domain not null)
        results = filter WHERE CARD.DOMAIN = filter_domain
      end if
      if (filter_type not null)
        results = filter WHERE CARD.CARD_TYPE = filter_type
      end if

      // Apply sorting
      if (sort_by == "name")
        results = ORDER BY CARD.CARD_NAME ASC
      else if (sort_by == "rarity")
        results = ORDER BY CARD_PRINTING.RARITY DESC
      else if (sort_by == "date")
        results = ORDER BY COLLECTION_CARD.DATE_ADDED DESC
      end if

      display results
      return results

Get_Collection_Statistics()
    Output: Total cards, unique cards, completion percentage
    Process:
      collection = query COLLECTION
            WHERE USER_ID = current_user_id
            AND IS_PRIMARY = TRUE
      // Calculate total cards (sum of all quantities)
      total_cards = query SUM(QUANTITY)
             FROM COLLECTION_CARD
             WHERE COLLECTION_ID = collection.COLLECTION_ID
      // Calculate unique cards (distinct printings)
      unique_cards = query COUNT(DISTINCT PRINTING_ID)
              FROM COLLECTION_CARD
              WHERE COLLECTION_ID = collection.COLLECTION_ID
      // Calculate total available cards in database
      total_available = query COUNT(*)
               FROM CARD_PRINTING
      // Calculate completion percentage
      completion_percentage = (unique_cards / total_available) * 100

      display "Total Cards: " + total_cards
      display "Unique Cards: " + unique_cards
      display "Collection Completion: " + completion_percentage + "%"
      return {total_cards, unique_cards, completion_percentage}
```

3. Search

```
Search_Cards()
    Input: Optional search term, filters (type, domain, rarity, energy, power, might)
    Output: List of matching cards
    Process:
      search_term = optional search term
      filter_type = optional card type filter
      filter_domain = optional domain filter
      filter_rarity = optional rarity filter
      filter_energy_min = optional minimum energy
      filter_energy_max = optional maximum energy
      filter_power_min = optional minimum power
      filter_power_max = optional maximum power
      filter_might_min = optional minimum might
      filter_might_max = optional maximum might
      // Base query with joins
      results = query CARD.CARD_ID, CARD.CARD_NAME, CARD.CARD_TYPE,
                  CARD.DOMAIN, CARD.ENERGY, CARD.POWER, CARD.MIGHT,
                  CARD_PRINTING.RARITY, CARD_SET.SET_NAME,
                  CARD_PRINTING.IMAGE_URL, CARD_PRINTING.PRINTING_ID
             FROM CARD
             JOIN CARD_PRINTING ON CARD.CARD_ID = CARD_PRINTING.CARD_ID
             JOIN CARD_SET ON CARD_PRINTING.SET_ID = CARD_SET.SET_ID
             WHERE 1=1

      // Apply search term (case-insensitive)
      if (search_term not null)
        results = filter WHERE LOWER(CARD.CARD_NAME) LIKE '%' + LOWER(search_term)
    + '%'
      end if

      // Apply filters
      if (filter_type not null)
        results = filter WHERE CARD.CARD_TYPE = filter_type
      end if
      if (filter_domain not null)
        results = filter WHERE CARD.DOMAIN = filter_domain
      end if
      if (filter_rarity not null)
        results = filter WHERE CARD_PRINTING.RARITY = filter_rarity
      end if
      if (filter_energy_min not null)
        results = filter WHERE CARD.ENERGY >= filter_energy_min
      end if
      if (filter_energy_max not null)
        results = filter WHERE CARD.ENERGY <= filter_energy_max
      end i
      if (filter_power_min not null)
        results = filter WHERE CARD.POWER >= filter_power_min
      end if
      if (filter_power_max not null)
        results = filter WHERE CARD.POWER <= filter_power_max
      end if
      if (filter_might_min not null)
        results = filter WHERE CARD.MIGHT >= filter_might_min
      end if
      if (filter_might_max not null)
        results = filter WHERE CARD.MIGHT <= filter_might_max
      end if
      // Sort results alphabetically
      results = ORDER BY CARD.CARD_NAME ASC

      display results
      return results

Get_Card_Details()
    Input: card_name or card_id
    Output: Complete card information and all printings
    Process:
      card_identifier = card_name or card_id
      // Get base card information
      if (card_identifier is card_id)
        card_info = query CARD WHERE CARD_ID = card_identifier
      else
        card_info = query CARD WHERE CARD_NAME = card_identifier
      end if
      if (card_info not found)
        display "Card not found"
        return null
      end if

      // Get all printings of this card
      printings = query CARD_SET.SET_NAME, CARD_SET.SET_CODE,
                  CARD_PRINTING.PRINTING_ID, CARD_PRINTING.RARITY,
                  CARD_PRINTING.FOIL, CARD_PRINTING.ALT_ART,
                  CARD_PRINTING.OVERNUMBERED, CARD_PRINTING.SIGNED,
                  CARD_PRINTING.RELEASE_DATE, CARD_PRINTING.IMAGE_URL
              FROM CARD_PRINTING
              JOIN CARD_SET ON CARD_PRINTING.SET_ID = CARD_SET.SET_ID
              WHERE CARD_PRINTING.CARD_ID = card_info.CARD_ID
              ORDER BY CARD_PRINTING.RELEASE_DATE DESC

      // Display card details
      display "Card Name: " + card_info.CARD_NAME
      display "Type: " + card_info.CARD_TYPE
      display "Domain: " + card_info.DOMAIN
      display "Energy: " + card_info.ENERGY
      display "Power: " + card_info.POWER
      display "Might: " + card_info.MIGHT
      display "Ability: " + card_info.ABILITY_TEXT
      display "Artist: " + card_info.ARTIST
      display "Collector Number: " + card_info.COLLECTOR_NUMBER
      display "Flavor Text: " + card_info.FLAVOR_TEXT

      // Display all printings
      display "Available Printings:"
      for each printing in printings
        display "  - " + printing.SET_NAME + " (" + printing.RARITY + ")"
        if (printing.FOIL) display "    Foil"
        if (printing.ALT_ART) display "    Alternate Art"
        if (printing.OVERNUMBERED) display "    Overnumbered"
        if (printing.SIGNED) display "    Signed"
      end for
      return {card_info, printings}
```

Installation Instructions

Prerequisites

The intended operating system for this application is Windows, as this is the only one I tested.

Install:

- Node.js (v16 or higher) https://nodejs.org/en
- PostgreSQL (v15 or higher) https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

Database Setup

1. Access the SQL Shell (psql) and create a new PostgreSQL database:
   ```
   CREATE DATABASE riftbounddb;
   ```
2. Connect to database:
   ```
   \c riftbounddb
   ```
3. Import the database:
   ```
   \i 'C:/Users/YourName/Downloads/riftbound_backup.sql';   // Example path
   ```
   Tips:
   - *Ensure file path is accurate and wrapped in single quotes.*
   - *All backslashes are changed to forward slashes in path.*
   - *Path must include the riftbound_backup.sql file.*
4. If import is successful, you are free to close the shell with
   ```
   \q
   ```

Backend Setup

1. Navigate to the backend folder using terminal in VS Code or Command Prompt:
   ```
   cd backend
   ```
2. Install dependencies:
   ```
   npm install
   npm install dotenv
   ```

Tips: If you encounter an issue that 'npm' is not recognized, visit this link for a comprehensive guide and fix: *https://www.codewithharry.com/blogpost/solving-npm-not-recognized-error-windows*

3. Create a .env file and update database credentials:

```
DB_USER=postgres
DB_HOST=localhost
DB_NAME=riftbounddb
DB_PASSWORD=Enter-your-DB-password-here
DB_PORT=5432
JWT_SECRET=Your-Secret-Key
PORT=3001
```

Tips: To generate a random key, use this command in the terminal and copy into the .env file for JWT_SECRET:

```
node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"
```

4. Start the backend server:

```
node server.js
```

Frontend Setup

Note: Keep the backend server running. Open a new terminal window and navigate back to the project root before proceeding.

1. Navigate to the frontend folder:

```
cd frontend
```

2. Install dependencies:

```
npm install
```

3. Start the development server:

```
npm start
```

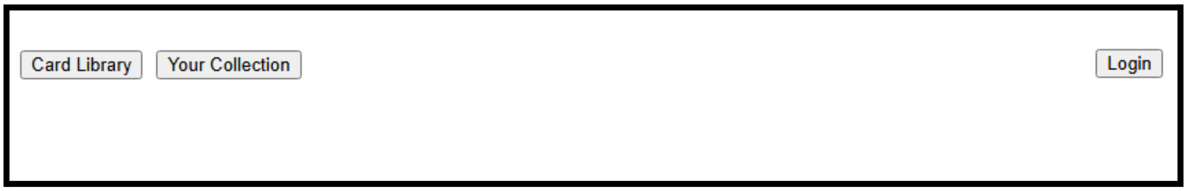
4. Open your browser to http://localhost:3000

User Manual

When you first access the application, you'll arrive at the landing page. The navigation bar at the top provides access to:

Card Library - Browse all available cards
Your Collection - View and manage your personal card collection
Login/Sign Up - Access your account

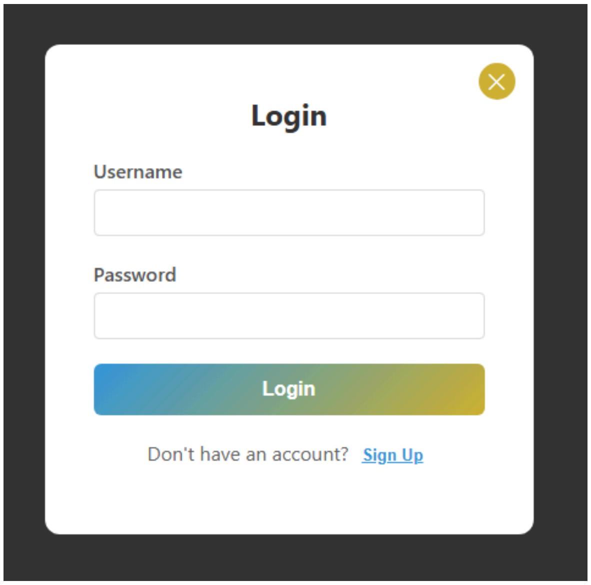
The bottom of the landing page features Riftbound content created by one of my friends.

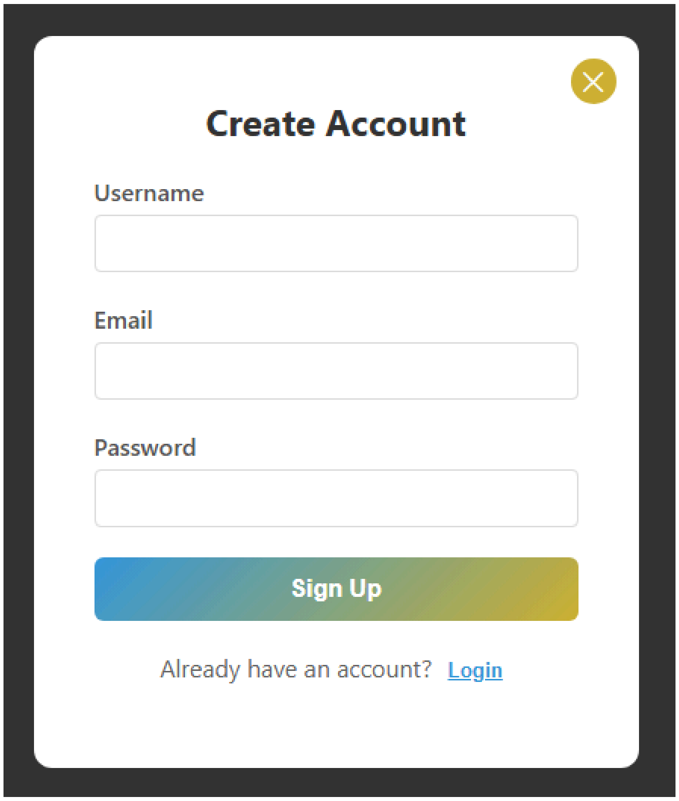


Account Management

First-time visitors can browse the Card Library as a guest without creating an account. However, to add cards to a collection or build decks, you must sign in.

1. Click the Login button in the navigation bar
2. A modal window will appear with login fields
3. New users: Click "Create an account" and follow the prompts
4. Returning users: Enter your username and password, then click Login
5. Once logged in, your username will appear in the navigation bar next to the Logout button





Search and Filter Options

The Card Library features a comprehensive filter container with the following tools:

- **Search Bar-** Search for cards by name (not case-sensitive)
    - Example: Typing "fury" will display cards like Fury Rune, Fury Rune (overnumbered), and Blind Fury

Filter Options

Rarity Filter - Multi-select dropdown to filter by card rarity

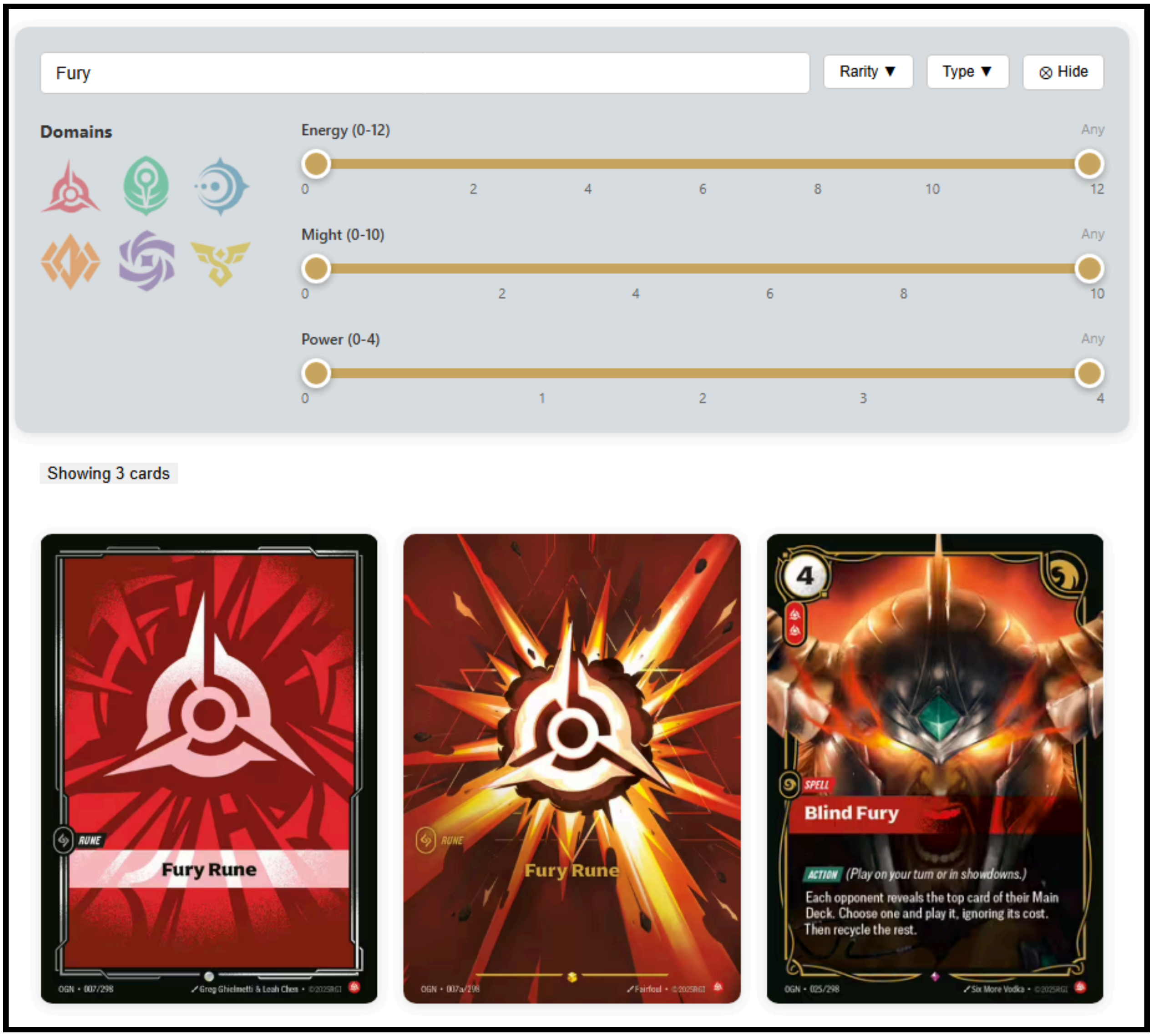
Type Filter - Multi-select dropdown to filter by card type

Domain Icons - Clickable icons to show cards from specific domains

Attribute Sliders - Set ranges for:

- Energy
- Might
- Power

Collapse Filter - Minimizes the filter panel to show only the search bar, dropdowns, and toggle



Fury

Energy (0-12) Any

Might (0-10) Any

Power (0-4) Any

Showing 3 cards

ACTION (Play on your turn or in showdowns.)

Each opponent reveals the top card of their Main Deck. Choose one and play it, ignoring its cost. Then recycle the rest.

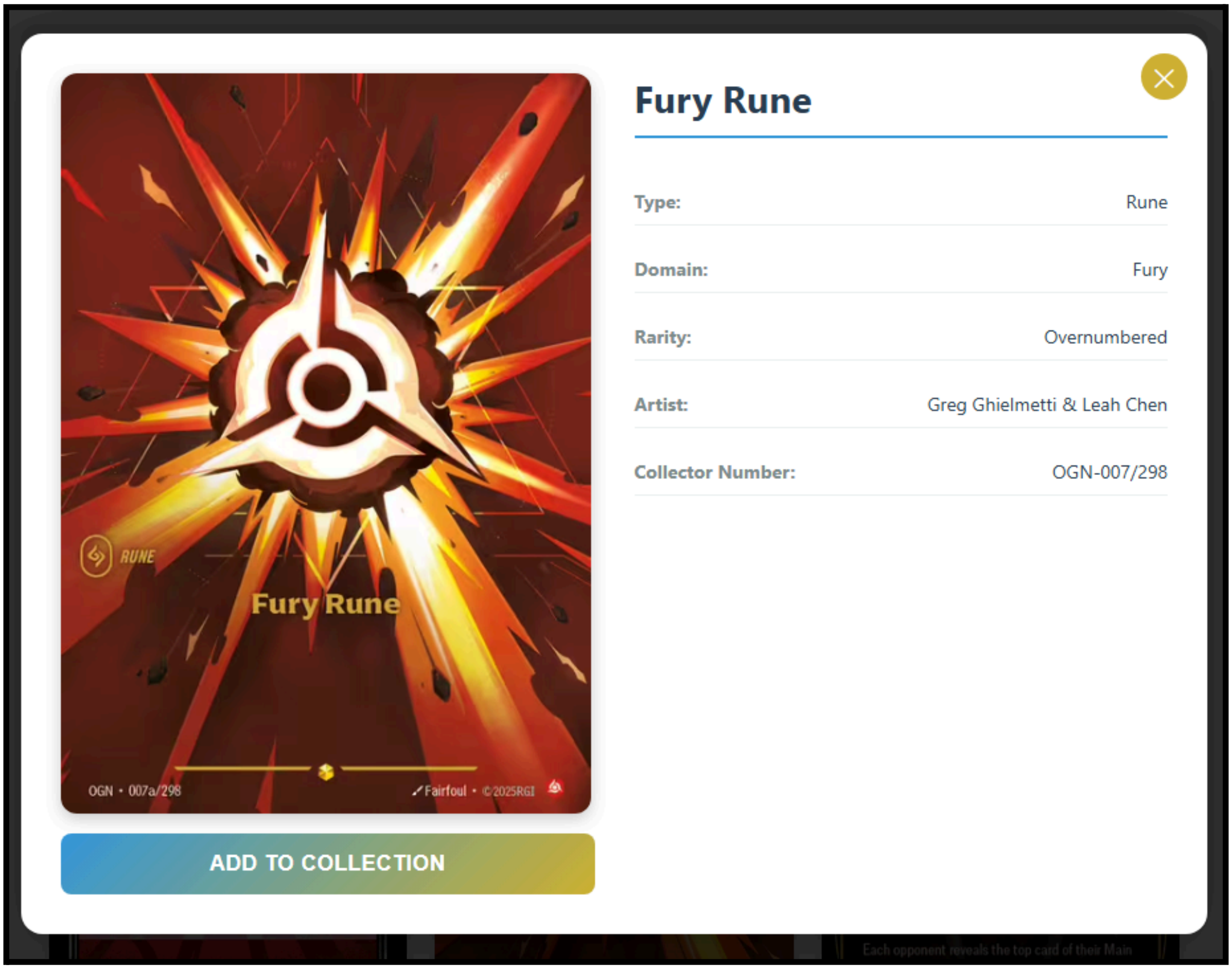
Viewing Card Details

Click on any card to open a detailed view showing:

- Card image
- Card traits and statistics
- Add to Collection button

Note: If you're not logged in, clicking "Add to Collection" will prompt you to log in or create an account. Once logged in, the card will be added to your collection with confirmation from the server.

To close the card detail view, click the X button or click outside the modal.



Fury Rune

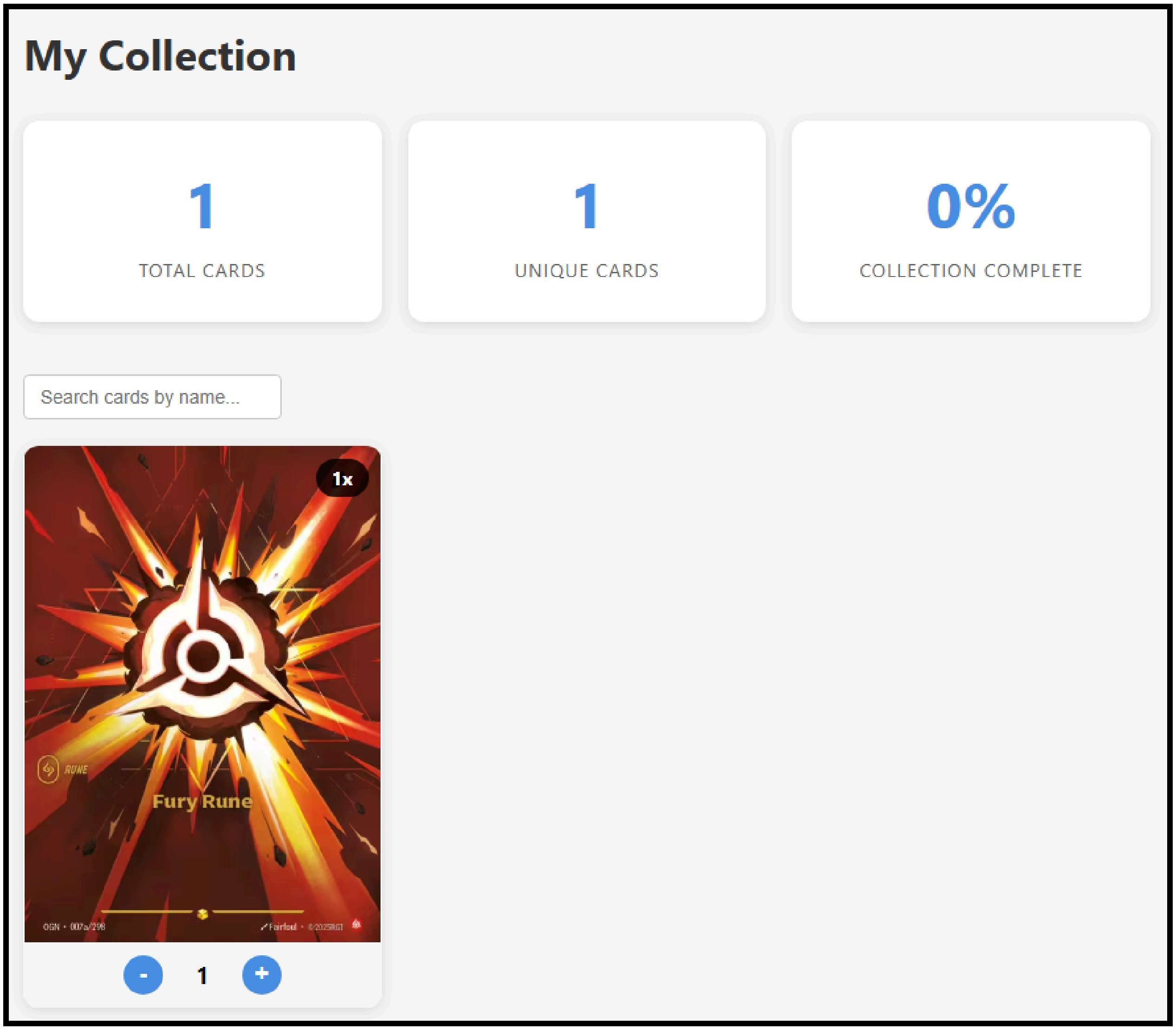
Type: Rune

Domain: Fury

Rarity: Overnumbered

Artist: Greg Ghielmetti & Leah Chen

Collector Number: OGN-007/298



My Collection

1 TOTAL CARDS

1 UNIQUE CARDS

0% COLLECTION COMPLETE

Accessing Your Collection
Click Your Collection in the navigation bar to view your default collection page.

The collection page displays:

- Total Cards - The total number of cards you own (including duplicates)
- Unique Cards - The number of different cards in your collection
- Completion Percentage - Progress toward collecting all available cards

Managing Card Quantities
Each card in your collection shows:

- Card image and details
- Plus (+) icon - Increment the quantity of this card
- Minus (−) icon - Decrement the quantity of this card

Use these buttons to accurately track how many copies of each card you own.