DeckFix

Angelina Morris

N00220782

Project supervisor: Michael McAndrew

Second reader: Cyril Connolly

Major Project

Year 4 2025-2026

DL836 BSc (Hons) in Creative Computing

Abstract

This document describes the development of DeckFix, a web application that allows users to build and generate Pokémon Trading Card Game (TCG) decks. The aim of this project was to answer the following research question: **How can constrained optimisation be used to evaluate and generate Pokémon Trading Card Game decks?** Decks were optimised using deck data from international Pokémon TCG tournaments, while still following game rules and addressing inconsistencies in other deck building resources. The application was built using Node.js, Express.js, MongoDB and SolidJS. The app was structured using the Layered Architecture and Model-View-Controller software architectural design patterns. The SCRUM methodology was used to develop the project to ensure consistent progress. The application was deployed using Render. Manual, automated and user testing was conducted. The project successfully answered the research question and achieved its aim by producing a tournament-quality deck evaluation and generation system.

Acknowledgments

Thank you to Michael McAndrew, the project supervisor for guiding me throughout this project.

Thank you to Becca Walsh, for designing the user testing poster.

Finally, thank you to all participants who took part in user testing.

Declaration of Ownership

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

Warning: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not leave copies of your own files on a hard disk where they can be accessed by others. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties (IADT Student Handbook 2024-2025).

The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below.

Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

Declaration: I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student: Angelina Morris

Signed:

Failure to complete and submit this form may lead to an investigation into your work.

Contents

1 Introduction

This document describes the development of DeckFix, a web application that allows users to build and generate Pokémon Trading Card Game (TCG) decks using constrained optimisation.

1.1 Project Aim

The aim of this project was to answer the following research question: **How can constrained optimisation be used to evaluate and generate Pokémon Trading Card Game decks?** Constrained optimisation is the process of optimising an objective function with respect to a set of decision variables while imposing constraints on those variables (Luu, 2024). In the context of this project, the function is evaluating and generating Pokémon TCG decks, the decision variables are the chosen cards, and the constraints are the game rules (see Appendix B) and user preferences. The function is optimised by integrating effective deck principles found in tournament decks from Limitless TCG, a database containing decks from international Pokémon TCG tournaments (Tournament Deck Lists, 2025). This project will use Standard Format 2026-2027 (see Appendix B) to keep the project's data relevant.

1.2 Project Objectives

The following objectives outline the tasks that were completed to realise the project's aim:

1. Built a database that contains user, session, card, and deck data.
2. Built an Application Programming Interface (API) that defines functionality for user authentication, card retrieval, deck building and generating with certain functionality being restricted to only authenticated users.
3. Built a user interface (UI) that allows users to access the API's protocols.
4. Built a deck building and generating tool that incorporates effective deck principles, game rules and user constraints.

1.3 Success Criteria and Scope

The project will be considered successful if it:

- Handles user authentication and data securely.
- Allows users to view, search and filter cards.
- Allows authenticated users to build and generate decks that incorporate effective deck principles, game rules and user constraints.
- Allows authenticated users to save cards to their collection so that they can be used when building/generating a deck.

1.4 Project Context

This project falls under the web development area of computing. It is a full-stack project that is deployed to the web, intended for desktop users. The UI is responsive so that users can access it on mobile web browsers, but it is not published to any mobile application stores.

2 Research and Background

This section describes the research conducted before and during the project's implementation.

2.1 Literature Review

(Tieber & Felfernig, 2021) describe a configuration system that builds decks under certain constraints for Magic: The Gathering. The authors' intention was to make deck-building more accessible for players, by creating balanced decks under the rules imposed by the player (for example, the deck price must not exceed the user's budget). The system would also assign a quality score to each card, based on how often those cards would appear in successful public decks, allowing decks to receive a quality score consequently. This study is extremely relevant to this project, as the system is performing deck generation based on given constraints, which is part of this project's objective alongside evaluating given decks. This study is a valuable source that illustrates a very similar architecture and goal to this project.

(Kowalski & Miernik, 2020) describe an active-gene evolutionary algorithm used to build decks for collectible card games where the cards must be picked from a random selection instead of a fixed set. The authors evolved the algorithm by selecting relevant cards from a random pool per given situation, teaching it strategies for a given selection of cards to limit interest in irrelevant cards. Decks were simulated using the strategies and tested in simulated games. The algorithm performed better than non-algorithm decks using random strategies. Although this project does not require decks to have such adaptive strategies, this study relates to this project as it performs deck building under given constraints.

2.1.1 Deck Optimisation Research

The aim of this project was to build a deck evaluation/generating system that incorporates the quality standard of tournament decks. (Tournament Deck Lists, 2025) provides detailed information on what cards are used in each deck in international tournaments, and how common those cards are in decks overall, indicating how useful they may be. This resource was used to record reoccurring patterns in successful

decks, such as the number of certain cards used in a deck. (Peterson, O'Connor, & Hay, 2025) give 11 criteria for an effective deck which can be used regardless of strategy or chosen Pokémon. This article supports much of the research from the tournament decks resource. It also argues that a player should choose powerful enough Pokémon cards that can sustain a game while only taking up a quarter of the deck, so that most of the deck can include trainer cards that will benefit the Pokémon as much as possible (and potentially sabotage the other player). (JustInBasil, A Basic Guide to Deck Building in the Standard Format, 2024) suggests the concept of a Main Attacker: a primary Pokémon card with high health points (HP) that the rest of the deck works around.

Game terminology and details of game and deck rules can be found in Appendix B. The first iteration of deck optimisation research can be found in Sprint Seven of the Implementation section, and the second iteration in Sprint Eight.

2.1.1.1 Final Iteration of Deck Optimisation Research

The following criteria were found to produce effective, tournament-quality decks when implemented.

Main Attacker: Four copies of a Main Attacker Pokémon card should exist in the deck. This Pokémon should have minimum 280 HP and an ability or two attacks. Four copies of the card ensure it is more likely to be drawn and can be used sooner. This Pokémon should sustain the game alone, with the help of utility Pokémon and trainer cards. Any pre-evolutions of the Pokémon should also be added.

Rare candy: If the Main Attacker is a Stage 2 Pokémon, the deck should have four copies of the “Rare Candy” trainer card (see Figure 1). This boosts the evolution process by skipping the Stage 1 step.

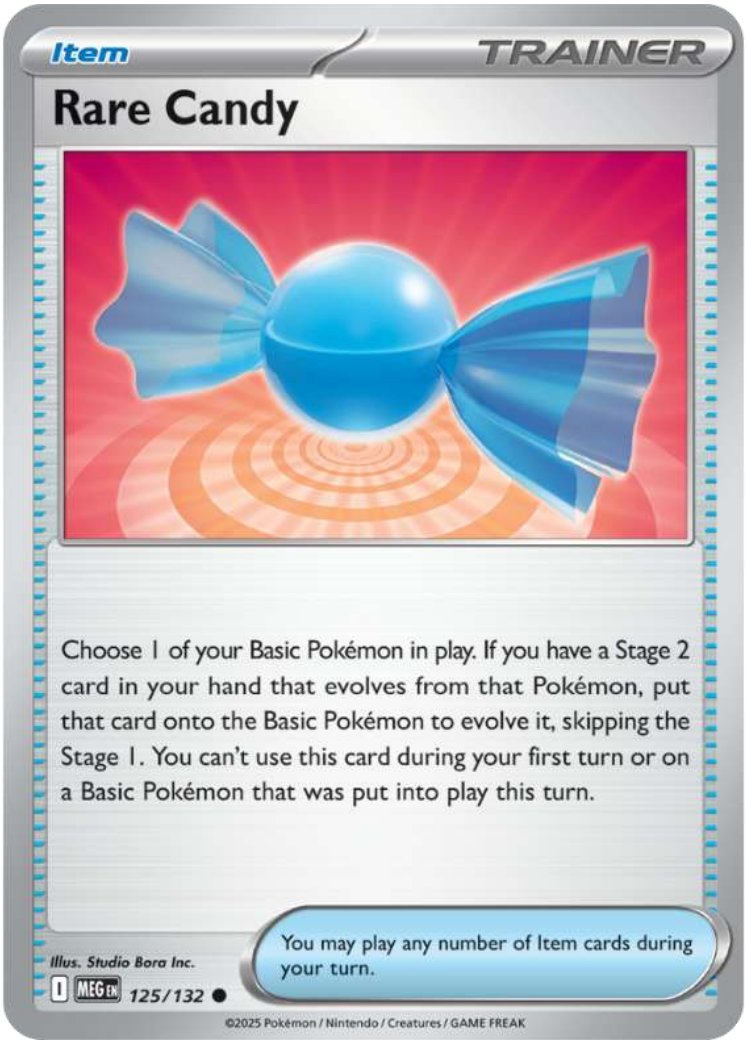

Figure 1: Rare Candy card

Utility Pokémon: The following utility Pokémon are found in almost every tournament deck and should be added to each deck:

- If the Main Attacker is N's Zoroark EX, add N's Reshiram and N's Zekrom (N's Zoroark EX copies the attacks of benched N's Pokémon).
- If the Main Attacker is of Darkness energy type, add Munkidori, which has the following ability: "Once during your turn, if this Pokémon has any Darkness Energy attached, you may move up to 3 damage counters from 1 of your Pokémon to 1 of your opponent's Pokémon."
- If the Main Attacker is of Fighting energy type, add Lunatone and Solrock from the Ascended Heroes expansion. Lunatone has the ability: "Once during your turn, if you have Solrock in play, you may discard a Basic Fighting Energy card from your hand to use this Ability. Draw 3 cards."
- For all other Main Attacker energy types, add:
    - Latias EX, with the ability: "Your Basic Pokémon in play have no Retreat Cost."

- Fezandipiti EX, with the ability: “Once during your turn, if any of your Pokémon were Knocked Out during your opponent's last turn, you may draw 3 cards.”

Main Attacker-specific trainer cards: Trainer cards that pertain to the Main Attacker should be included in a deck. This includes energy type-specific cards, prefix-specific cards and subtype-specific cards.

Generic trainer cards: Alongside Main-Attacker specific trainer cards, trainer cards that allow for drawing cards, switching your or your opponent’s Pokémon out of the active spot and healing Pokémon should be added.

Supertype proportions: A deck should comprise of approximately 8-16 Pokémon cards, 32-40 trainer cards and 12 energy cards. Trainer cards should supplement a lower Pokémon card count, but Pokémon cards should not supplement a lower trainer card count.

2.2 Technical Research

2.2.1 Constrained Optimisation

Constrained optimisation is the process of optimising an objective function with respect to a set of decision variables while imposing constraints on those variables (Luu, 2024). In the context of this project, the function is evaluating and generating Pokémon TCG decks, the decision variables are the chosen cards, and the constraints are the game rules. Using the data found in the deck optimisation research, the evaluating/generating functions will be optimised to provide tournament-quality decks. A deck “score” will be used as the optimisation metric. The more criteria fulfilled by the deck, the higher its score.

2.2.2 Feasibility Study

To find whether this project’s objectives were possible, a feasibility study was conducted. With guidance from the project supervisor, the author created a rudimentary deck evaluating system to expand on the project’s ideas and identify potential faults. The feasibility study only used the first iteration of deck optimisation research. The feasibility study can be found in Appendix A. Deck generation was not researched due to time constraints.

2.2.2.1 First Iteration of Deck Evaluation System

In the early stages of research, the Pokémon TCG API (Backes, Pokémon TCG API Documentation, 2022) was chosen as the preferred dataset for the project as it had detailed documentation and a large volume of data. It provided developers with software development kits (SDKs) for a variety of programming languages. The first iteration of the deck evaluating system used the JavaScript SDK (Backes, pokemon-tcg-

sdk-javascript, 2021). However, upon testing, it was found that the API was extremely unreliable. It would frequently return internal server errors, or the request would time out, even with filters applied. This project requires a stable data resource, so the next iteration would implement an alternative.

2.2.2.2 Second Iteration of Deck Evaluation System

The author of the Pokémon TCG API, Andrew Backes, released a resource containing all the data from the API in JSON format (Backes, pokemon-tcg-data, 2025) which is regularly updated. This proved to be a far more reliable resource, as it provides the detailed card information without the risk of poor server stability.

This resource was implemented by copying two card expansion (set of cards) files into a new Node.js project (Node.js v25.9.0 documentation, 2026). The data was stored using MongoDB (Welcome to the MongoDB Docs, 2026), as seen in Figure 2.

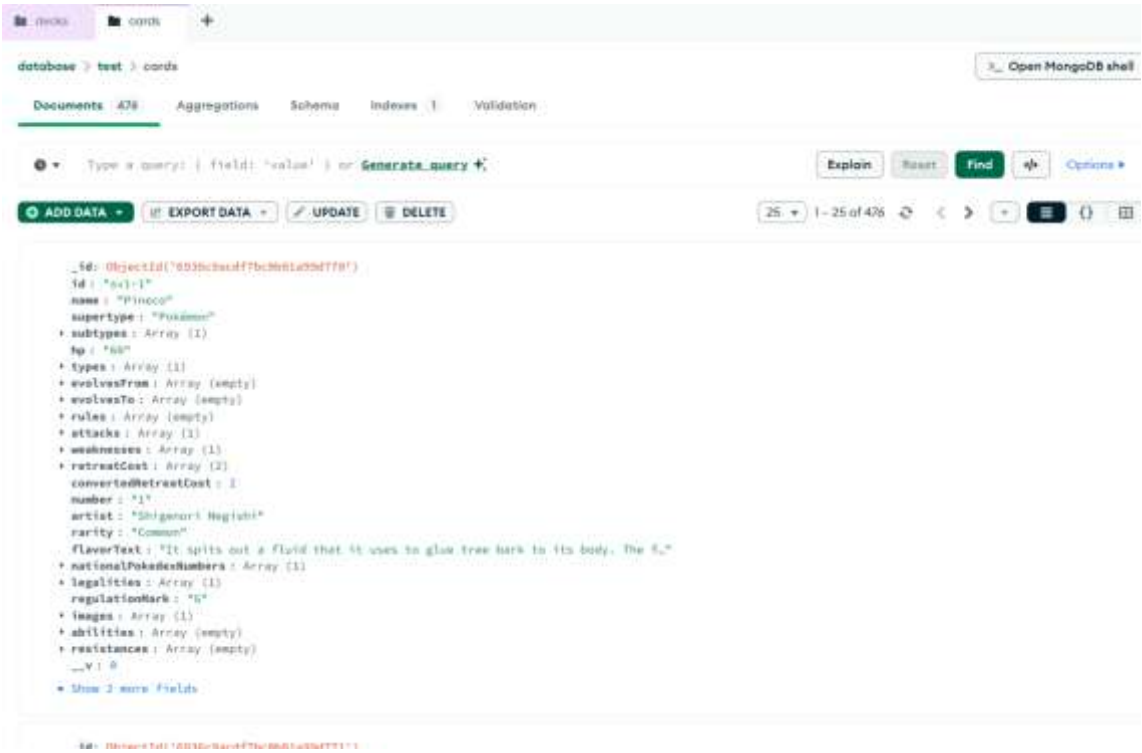

Figure 2: Data stored in MongoDB

Five decks consisting of 30-55 random cards from the two expansions were created as seen in Figure 3.

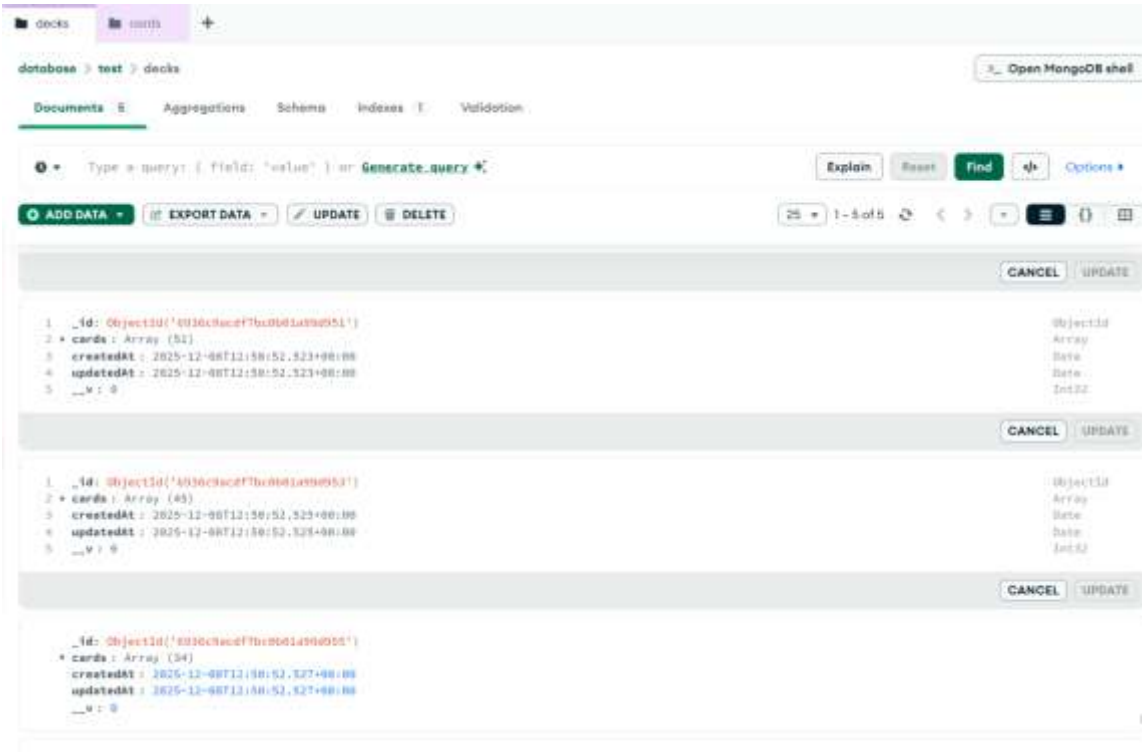

Figure 3: Deck objects in MongoDB

A simple application was created using Express.js (5.x API, n.d.), containing three endpoints: get a card by ID (GET), get a deck by ID (GET), and grade (evaluate) a deck by ID (GET). This was tested using Postman, an API testing tool (Postman Docs, 2026). In the grade endpoint, the deck's score would be reflected on whether the deck:

- Had an ACE-SPEC card.
- Had between 10 and 20 energy cards.
- Had any Pokémon card with an attack that only requires Colourless energy.
- Had any Pokémon card with an ability.

The deck was assigned a score based on how many of these criteria it fulfilled. The score would be raised by 25 for each criterion fulfilled to estimate a score out of 100.

As expected, the decks performed extremely poorly as the cards were randomly generated (see Figure 4). Using MongoDB was considered redundant, as the expansions were already stored as JSON files within the project's directory, and the evaluating could be performed locally by using Node.js scripts instead of endpoints. This also meant a JSON file with cards based on a tournament deck could be created to see how it would perform against the optimisation criteria. This was reflected in the third iteration of the deck evaluating system feasibility study.

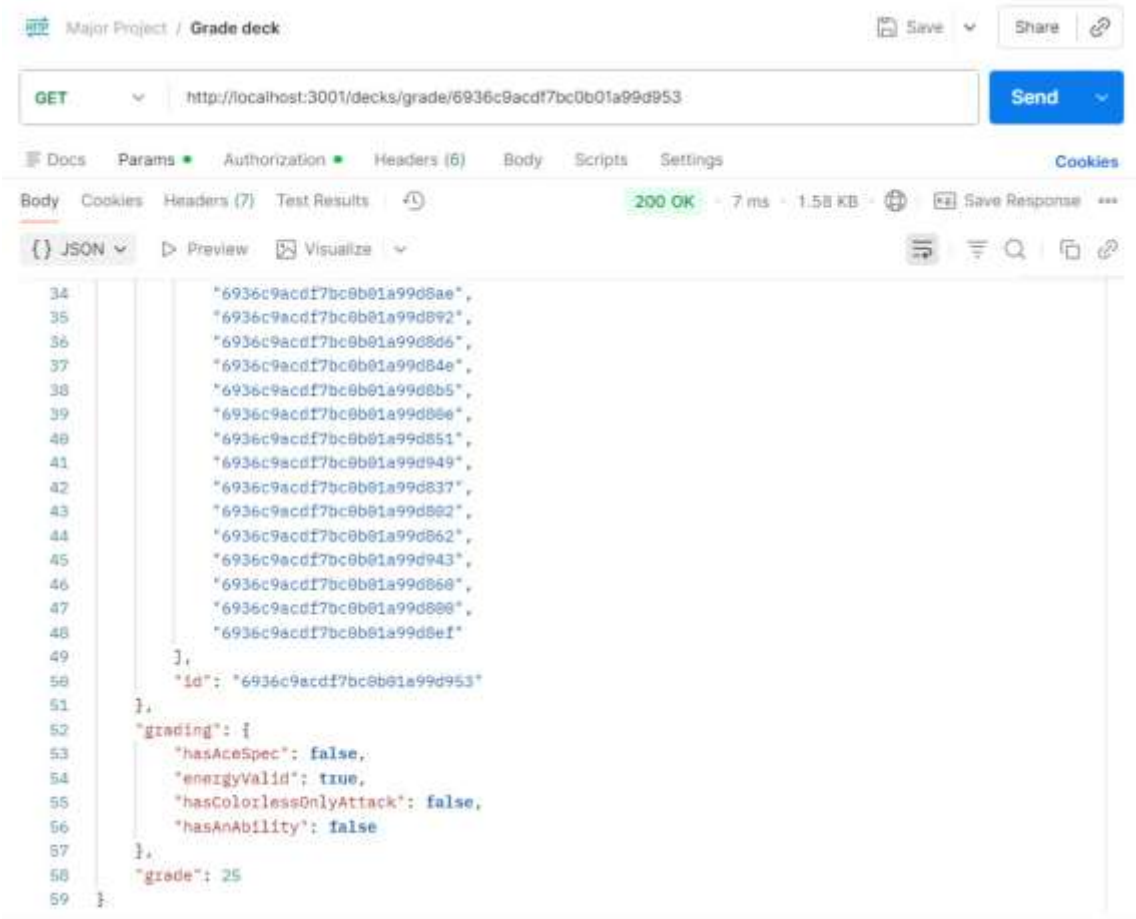

Figure 4: Grade deck endpoint response

2.2.2.3 Third Iteration of Deck Evaluation System

This iteration aimed to simplify the technology used, evaluate tournament decks using JSON files, and implement more deck optimisation criteria.

MongoDB was disintegrated to simplify data handling. Instead of an Express.js application containing endpoints, the evaluating system was reduced to a file called evaluate.js that was ran using Node.js. The JSON data was copied from (Backes, pokemon-tcg-data, 2025) and pasted into new JSON files to mimic tournament decks. Additional criteria were added to the evaluating system. The score assigned to each criterion was lowered to 12.5, as there were now 8 criteria. In this iteration, the deck's score was reflected on whether it:

- Had an ACE-SPEC card.
- Had between 10 and 15 energy cards.
- Had any Pokémon card with an attack that only requires Colourless energy.
- Had any Pokémon card with an ability.
- Had 30 trainer cards.
- Had any Pokémon card with two attacks.
- Had five to nine Pokémon EX cards.
- Had any ability, attack or trainer card that allows the player to draw cards.

Two tournament decks were evaluated (see Figures 5 and 6).

Figure 5: First deck evaluation result

Figure 6: Second deck evaluation result

As expected, the decks performed far better, as they were retrieved from tournament data and the cards were not randomised.

2.2.2.4 In the Official Iteration

The third iteration accomplished the most project objectives, however in the final iteration MongoDB was implemented as cloud storage was essential, as well as API endpoints. Due to time constraints, the following research was not conducted:

- Adding more criteria that further optimises the deck evaluating.
- Implementing criteria weighing that prioritises certain aspects of the deck over others and assigns score accordingly.
- Adding more tournament decks in JSON format and evaluating them, to find the correlation between deck score and position in tournament.
- Deck generation.

This was explored during the official project implementation.

2.2.3 Project Technologies

2.2.3.1 Back End

During the technical research feasibility study, Node.js, Express.js and MongoDB were used to explore the objectives of the back end of the project. They were familiar and proved to be suitable, and were therefore used in the official implementation.

JavaScript was used in the feasibility study; however, TypeScript was used in the final project due to its improved code reliability and error detection (Pattanayak, 2024).

Amazon (AWS) S3 and CloudFront were used to host card images (Welcome to AWS Documentation, 2026).

2.2.3.2 Front End

For the UI, React.js (Quick Start, n.d.) was considered due to its familiarity and large library of resources. However, deviating from technology found in the academic course material would show initiative in finding a technology suited to the project's needs and expand the author's skill range as a software developer. Svelte (Overview, 2026), a front end framework was researched first. (Fireship B. , 2023) was initially consulted to identify similarities between React.js and Svelte, as React.js was featured largely in course material and can be easily referenced. This source mentioned SolidJS (Quick start, 2026), another JavaScript library, prompting further investigation. (MINUTES, 2025) compares Svelte.js and SolidJS. It praises SolidJS' speed, small size and its similarities to React.js, which is desirable due to the author's experience with it. The source built two applications to navigate each technology. In the second application, which had more similar objectives to this project than the first application, SolidJS was deemed more efficient. SolidJS' was chosen for the final application due to its likeness to React.js, speed, scalability and suitability. SolidJS' unfamiliarity helped the author expand her software development skills.

As SolidJS was an unfamiliar technology, research on its implementation was conducted. (snnsnn, 2025) is a book that provides SolidJS code examples, which is a great resource to learn about SolidJS. Example 01 in Chapter 35 was used as the foundation for the UI of the final project. This example used Vinxi (Getting Started, n.d.), an application bundler, which was also integrated into the final project.

(Soriano, 2022) describes a tutorial to build a shopping application using SolidJS. The shopping cart aspect was used as a foundation for the deck building tool UI in this project as it shared similar objectives (adding an item is the equivalent to adding a card, "checking out" is like saving a deck, etc.) Integrating practical tutorials into the project helped minimise the intimidation of learning a new technology.

2.2.3.3 Project Management

Miro was used to record project research and to create paper prototypes of the UI (see Appendix A).

Trello was used to record project progress (see Appendix A).

GitHub is the preferred source control technology that has been used throughout the academic course due to its reliability and ease of use; therefore, it was used for this project.

Render is a full-stack deployment technology that was used previously in the academic course. It is reliable and easy to use; the only drawback being the loading screen on the free plan. However, most deployment technologies will have similar drawbacks when using free versions. Therefore, Render was implemented into this project.

3 Requirements Analysis

This section describes the investigation conducted into the project's requirements to ensure that it is feasible and achievable within the given time.

3.1 Similar Applications

Pokémon TCG deck building resources already exist; however, they provide no evaluation context, and the only existing deck generation tool provides only partial deck generation.

3.1.1 Limitless TCG Deck Builder

(Builder, n.d.) originates from (Tournament Deck Lists, 2025), the resource documenting tournament decks. It provides a simple deck building tool with search and filter functionality (see Figure 7), and detailed card information (see Figure 8). However, it does not allow filtering by expansion, meaning users cannot browse cards unless they know the name of the card. The deck is also lost upon refresh if not saved to the user's profile. It provides no deck evaluation or deck generation functionality.

Figure 7: Limitless TCG deck builder

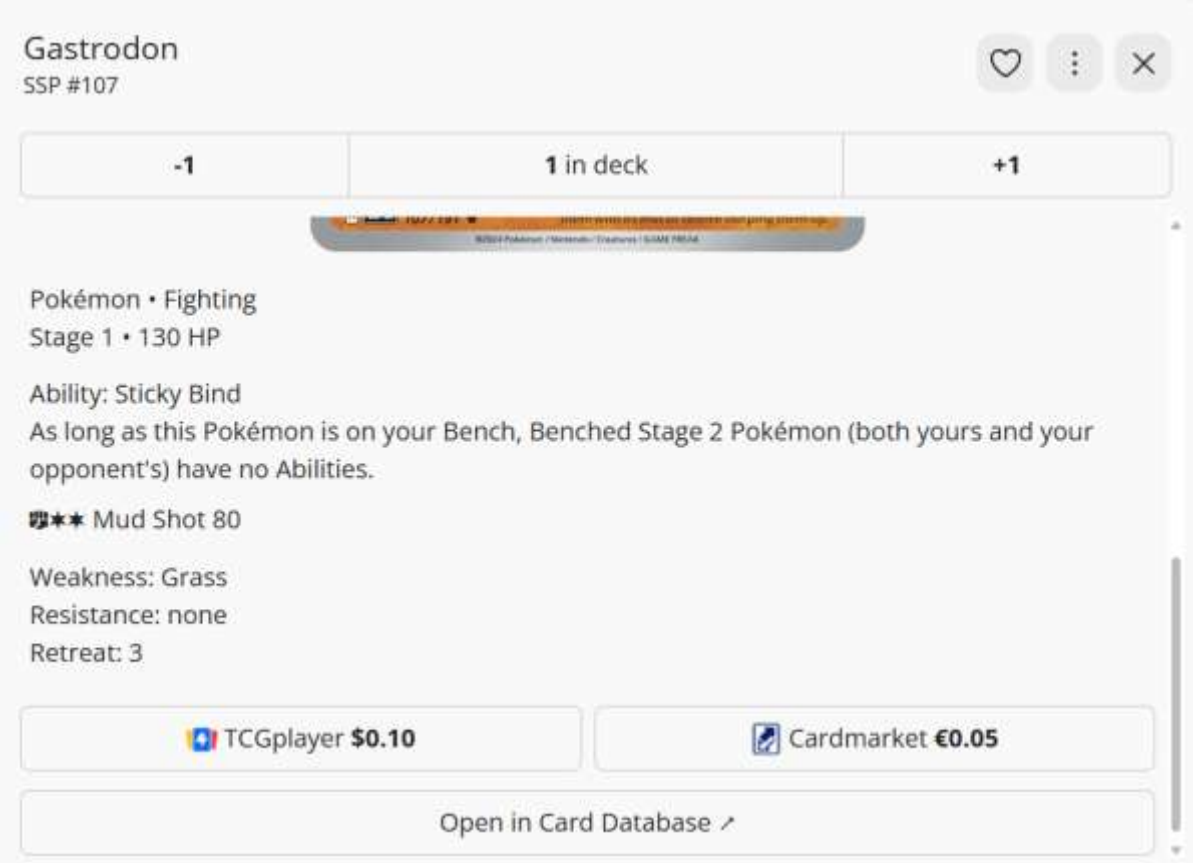

Figure 8: Limitless TCG card information within deck builder

3.1.2 PokemonCard Deck Builder

(Deck builder, n.d.) provides a deck building tool with partial deck generation functionality (see Figure 9). Like the previous application, it includes search and filter functionality, and detailed card information (see Figure 10), but does not allow filtering by expansion, and the deck is lost upon refresh if not saved to the user's profile. It does not provide deck evaluation.

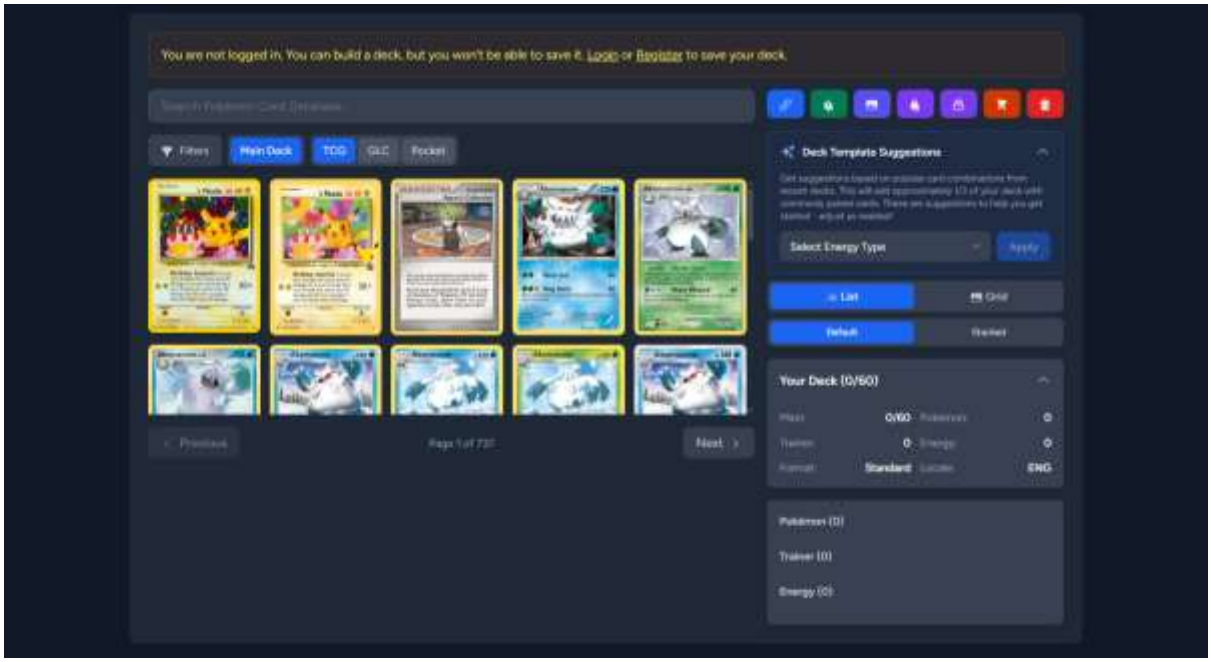

Figure 9: PokemonCard deck builder

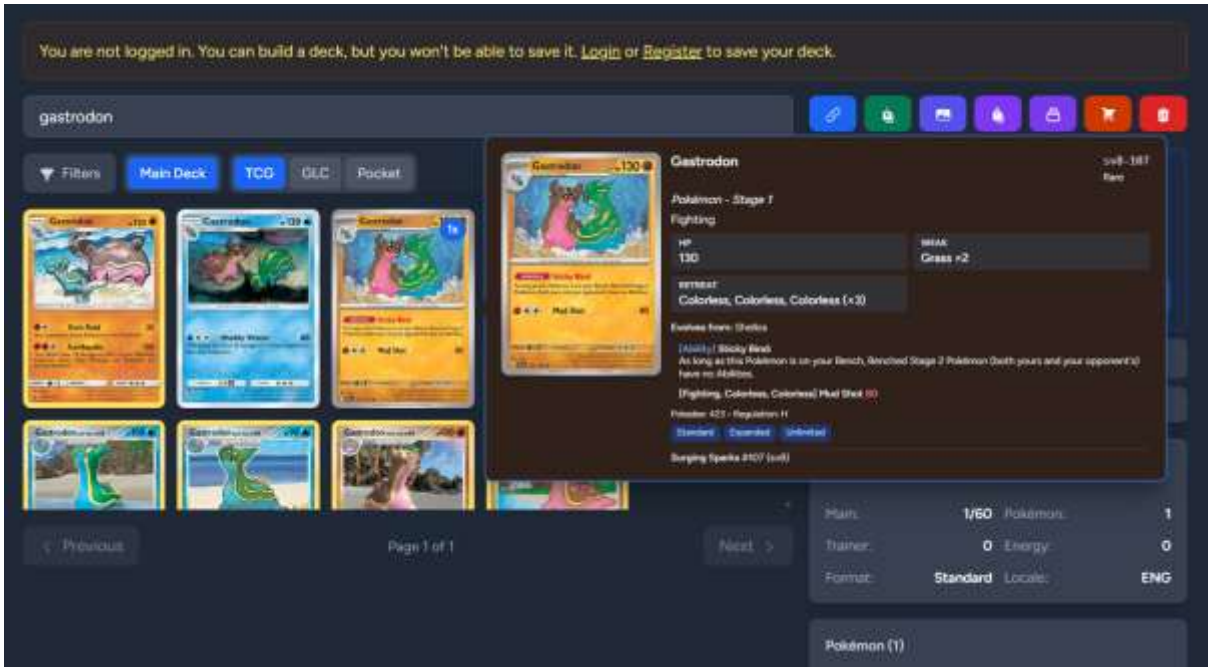

Figure 10: PokemonCard card information within deck builder

The deck generation functionality is referred to as a deck template suggestion and is based on the energy type chosen by the user (see Figure 11). The tool chooses 20 popular (high volume of appearances in other decks) cards to help the user start a deck. However, it does not generate a full deck, does not generate trainer cards, and can break game rules. In Figure 11, when Grass energy type was chosen, the tool generated six different Budew cards, which breaks the rule of only four copies of a card with the same name existing within a deck (see Appendix B).

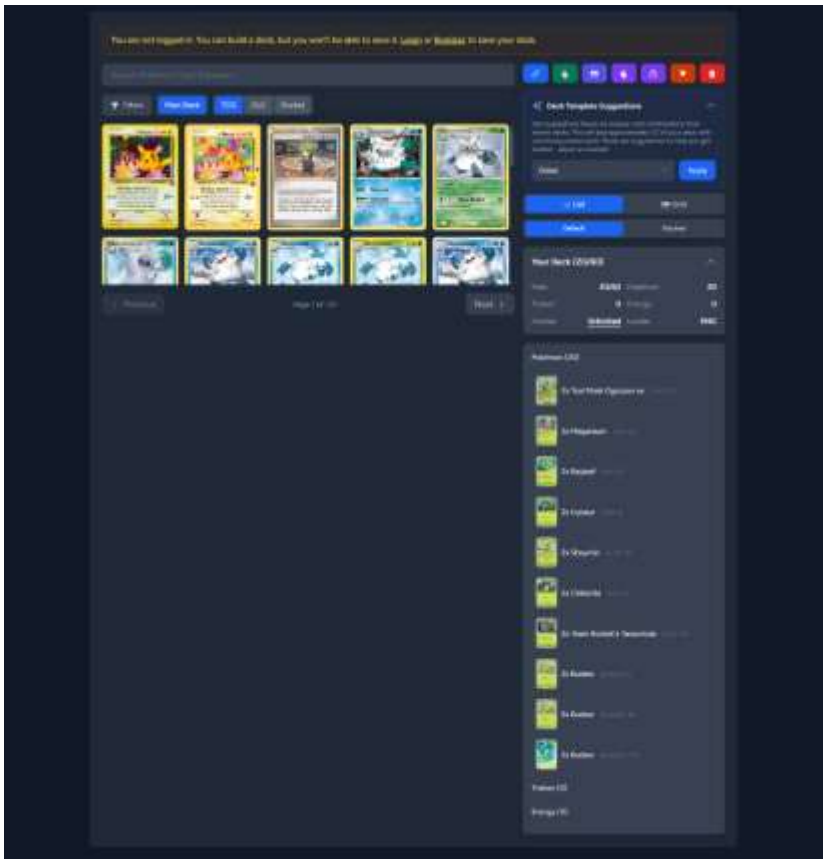

Figure 11: PokemonCard deck template suggestion

3.1.3 Analysis

This project should incorporate the previous applications' search and filter functionality and simple UIs. Clicking to add a card feels more intuitive in PokemonCard's tool than dragging the card to the deck window in Limitless TCG's tool. The detailed card information feels redundant as all information can be found on the card image. Both tools can load the entire card dataset (these applications are not limited to Standard Format, meaning they load 18,000+ cards) which can overburden the server's resources (it is paginated in PokemonCard but not in Limitless TCG). This project should improve on the deck's state persistence (the deck remains on page refresh) and should allow users to browse cards by expansion.

3.2 Use Case Diagram

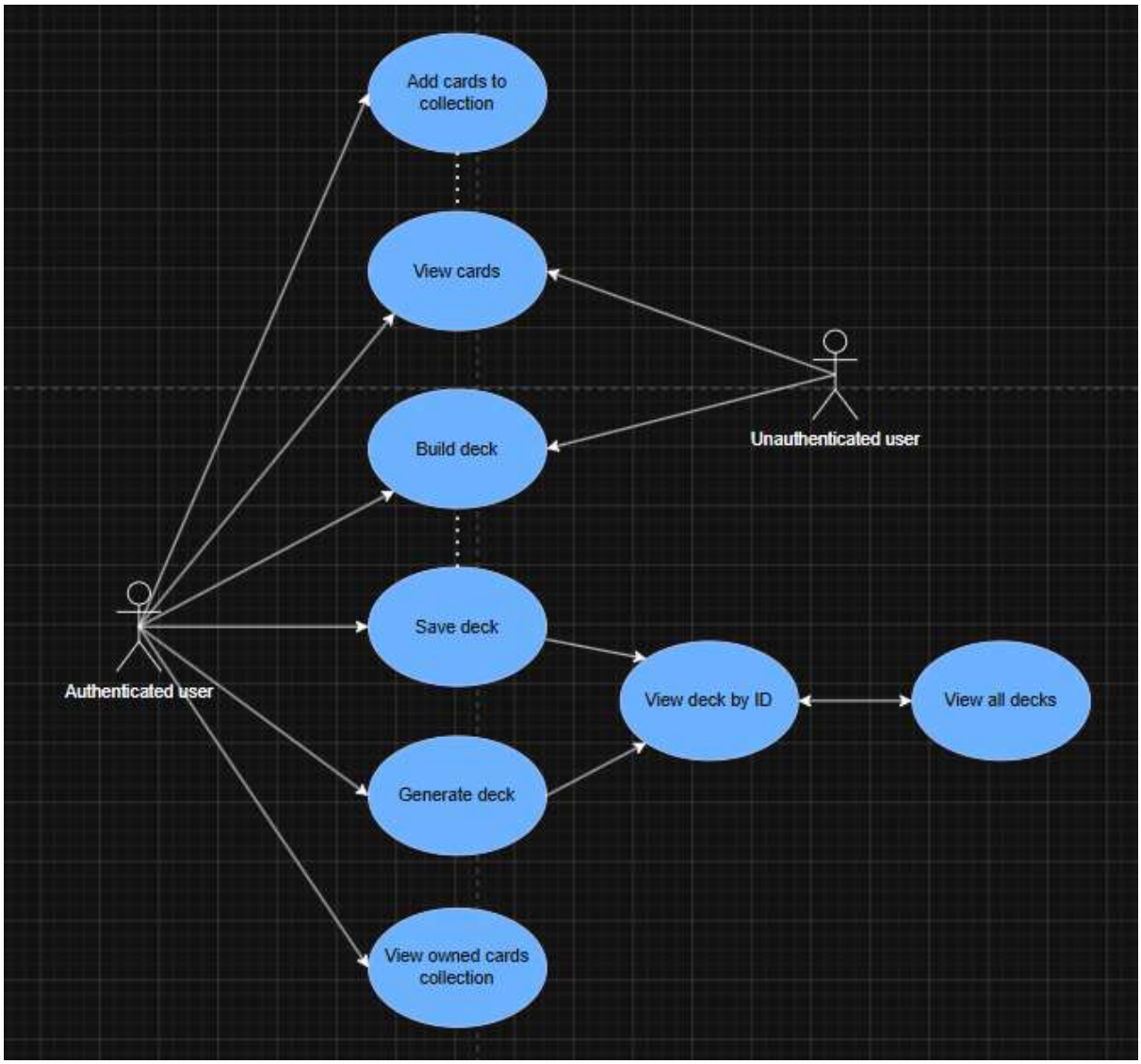

Figure 12: Use case diagram

Figure 12 shows a use case diagram for the application.

An unauthenticated user can view cards but not add them to their collection and build decks but not save them.

An authenticated user can view cards and add them to their collection, build and save a deck, generate a deck, view all or one of their decks and view their saved card collection.

3.3 Functional Requirements

Functional requirements define the features of an application that are necessary for it to perform (Functional and Non Functional Requirements, 2026). The list of functional requirements this project should incorporate is defined using the MoSCoW method: a project management technique that divides elements of the project into the following

categories: Must-have, Should-have, Could-have and Won't-have (MoSCoW Prioritization, n.d.).

Must-have:

- Basic deck building tool with evaluation.
- Basic deck generation tool with no user preference constraints, only game rule constraints.
- User authentication that allows users to save decks to their profile.
- Page that displays saved user decks.
- Page that displays all cards.

Should-have:

- User preference constraints within the deck generation tool.
- Functionality that allows authenticated users to save cards to their collection which can be found in the "mycards" section when building a deck and on the "mycards" page.
- Basic search and filter functionality (such as energy type) within the cards page, the mycards page and the deck building tool.
- Admin interface.

Could-have:

- Advanced search filters (such as minimum HP).
- Share link for individual saved decks.
- "Explore" page that shows different user decks (with user consent).

Won't-have:

- Chat function between users.
- Test feature allowing users to test their decks against an automated user or another user.
- Advanced card collecting - users cannot save special card editions (such as the Holiday Calendar editions).
- Deck/card cost data.

3.4 Non-functional Requirements

Non-functional requirements define how the application should perform its features (Functional and Non Functional Requirements, 2026). The non-functional requirements of this project are also defined using the Moscow method:

Must-have:

- Fast request processing and response handling.
- Reliable system stability and availability.

- Secure error-handling.
- Secure user authentication, user data handling and protection against unauthorised access.
- Navigable UI.

Should-have:

- Visually pleasing UI.
- Responsive UI (scale elements to screen size).
- Scalable architecture.

3.5 Feasibility Study

3.5.1 Technical

During the research phase for this project, a technical feasibility study was conducted for the deck evaluation tool. This research proved that the deck evaluation tool was feasible. Due to time constraints, the deck generation tool was not put under a feasibility study.

3.5.2 Economic

The primary economic aspect of this project concerned application hosting.

There was a small fee to host the project's database on MongoDB Cloud, which has a "Flex" plan that charges $0.011 per hour at a maximum of $30 per month. This plan is aimed towards development that may encounter bursts of unprecedented traffic, which suits the needs of this project.

The other fee stems from application hosting on Render. Render has a Free plan that serves the needs of the project, however the server "sleeps" after 15 minutes of inactivity. This meant that when a request was sent during the server's sleep, a lengthy loading screen was displayed before the content of the website is shown. This would not accommodate the needs of user testing; therefore a $7 (€5.96) monthly instance fee was paid.

3.5.3 Operational

3.5.3.1 Pokémon TCG Card Dataset

A reliable Pokémon TCG card dataset was paramount to the success of this project. In the early stages of research, the Pokémon TCG API (Backes, Pokémon TCG API Documentation, 2022) was chosen as the preferred dataset. However, during the technical feasibility study, it was found that the API was extremely unreliable. (Backes, pokemon-tcg-data, 2025) was used instead as it contained all the data from the API in JSON format. This would prove to be a far more reliable resource, as it removed the risk of poor server stability.

Each card in the JSON dataset contained links to images stored in the Pokémon TCG API. When inspecting these links, the request headers include “strict-origin-when-cross-origin”. This raised concern as to whether the images would be accessible when integrating the API into this project. A feasibility study was conducted to find whether the images would have to be published to an AWS S3 bucket. This was done using the code from the technical feasibility study and SolidJS Example 1, Chapter 35 (snnsnn, 2025). The images were accessible (see Figure 13), meaning using S3 was unnecessary. However, this test was redundant as in the official implementation, the images were published to S3 to ensure the images would always be accessible and controlled by the project owner.

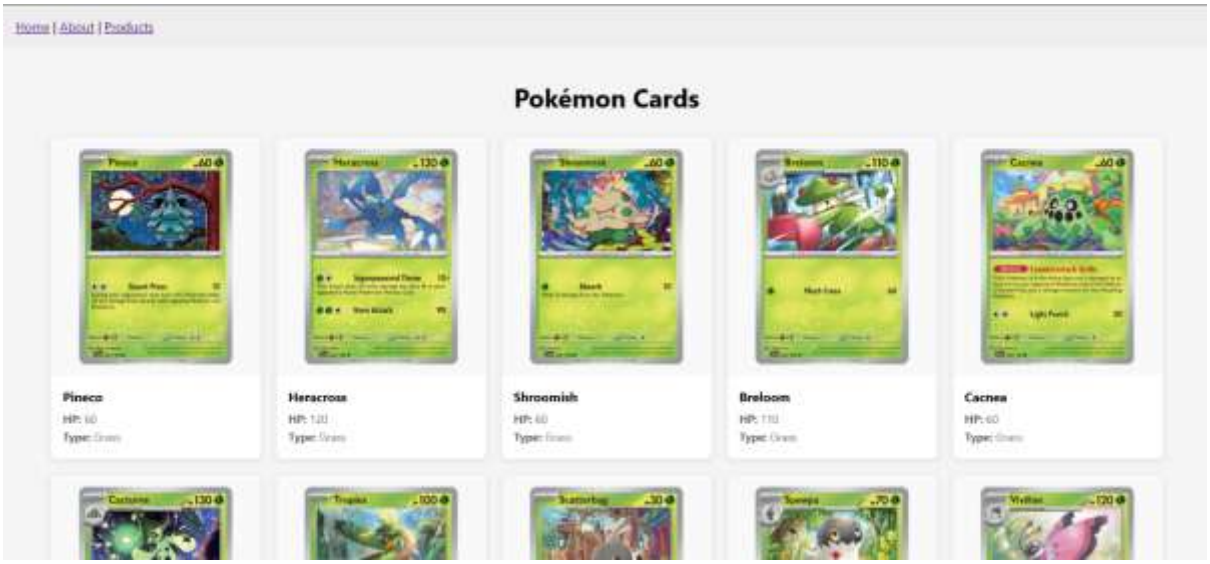

Figure 13: Card image feasibility study

3.5.3.2 User Testers

User testers who are familiar with Pokémon TCG were preferred for project user testing, as they could offer deeper insight into the project’s strengths and weaknesses such as the accuracy of the deck evaluation. To ensure reliable user feedback, a poster advertising the project was displayed in local card shops.

4 Design

This section describes what the project’s architecture, interface and behaviour should look like when completed.

4.1 System Architecture

4.1.1 Architectural Patterns

This project implements a combination of two architectural patterns: the Layered Architecture pattern and Model-View-Controller (MVC) pattern.

Layered Architecture is a pattern that structures an application into multiple distinct layers, each responsible for specific tasks or concerns (Software Architectural Patterns in System Design, 2025). In this application, the layers are:

- **UI:** Displays information to the user and handles user input.
- **API:** Contains the application's logic, such as processing user requests.
- **Database:** Where data is stored and retrieved from.

MVC is a pattern that separates an application into three interconnected components: Model, View, and Controller (Software Architectural Patterns in System Design, 2025).

- **Model:** Represents the data and business logic of the application. It retrieves, stores, and processes data.
- **View:** The UI component that displays the data to the user and responds to user interactions, which is updated whenever the Model changes.
- **Controller:** Acts as an intermediary between the Model and the View. It handles user input, updates the Model accordingly, and updates the View to reflect any changes in the Model.

4.1.2 API Design

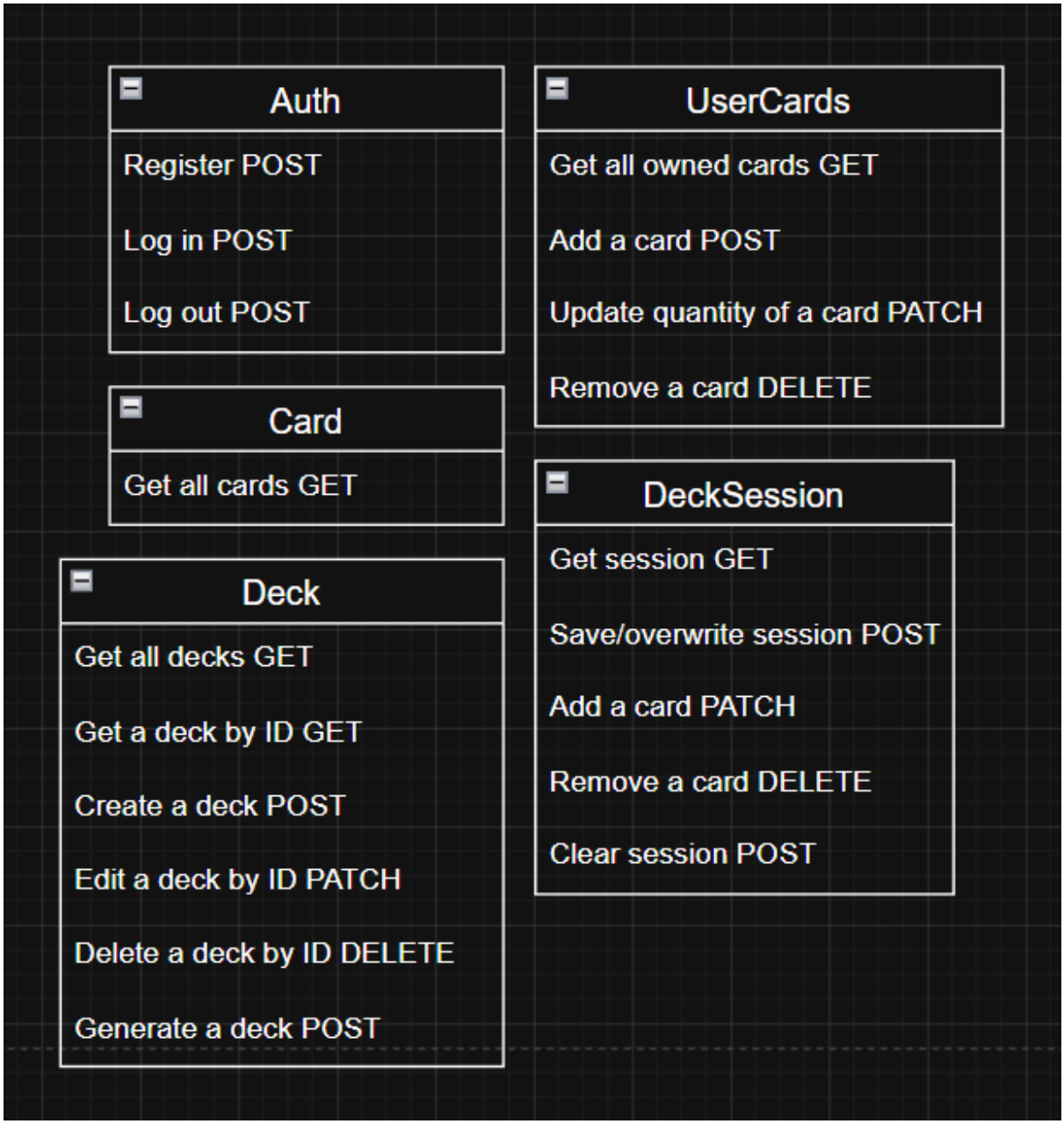

Figure 14: API design diagram

Figure 14 describes the application's API structure. The front end of the application will make requests to the API.

When developing, the requirements of the API included:

- **Authentication:** register, login, logout.
- **Card:** view all cards.
- **Deck:** save a deck, generate a deck, view all user decks, view a user's deck by ID.
- **UserCards:** save a card to a user's collection, remove a card from a user's collection, view all saved cards in a user's collection.

The final API has the following functionality:

Authentication:

- Register (POST): Allows users to submit their data to create an account in the application.
- Login (POST): Allows users to submit their credentials to access their account in the application.
- Logout (POST): Allows users to clear their session.

Card:

- Get all cards (GET): Allows users to access all card information. This was later changed to only return card data given a card expansion parameter. This would reduce the request load by only returning data from the chosen card expansion (50-300 cards) as opposed to the entire card collection (3000+ cards).

Deck:

- Get all user decks (GET): Allows an authenticated user to get all of their decks.
- Get a user's deck by ID (GET): Allows an authenticated user to get one of their decks by ID.
- Create/save a deck (POST): Allows an authenticated user to save the deck they created in the deck building tool.
- Edit a deck (PATCH): Allows an authenticated user to update a deck they created by ID.
- Delete a deck (DELETE): Allows an authenticated user to delete a deck they created by ID.
- Generate a deck (POST): Allows an authenticated user to generate a deck given their preferences (user constraints).

UserCards:

- Get all saved cards (GET): Allows an authenticated user to view all the cards they saved to their collection.
- Add a card to collection (POST): Allows an authenticated user to save a card to their collection.
- Update card copy quantity of a saved card (PATCH): Allows an authenticated user to add/remove copies of the cards in their collection.
- Remove a card from collection (DELETE): Allows an authenticated user to remove a card from their collection.

DeckSession was added to allow users to securely maintain the state of an unsaved deck in the deck building tool and prevents data loss if, for example, the page refreshes:

- Get a session (GET): Retrieves the state of a user's deck building tool session.

- Save/overwrite a session (POST): Saves the state of a user's deck building tool session. The front end makes a request to this endpoint every few seconds, which overwrites the previous session. This ensures that if a user refreshes or re-enters the deck building tool page, the most recent state is returned using the GET endpoint.
- Add a card (PATCH): Add a card to the user's deck building tool session.
- Remove a card (DELETE): Remove a card from the user's deck building tool session.
- Clear a session (POST): Clear the user's deck building tool session.

4.2 Interface Design

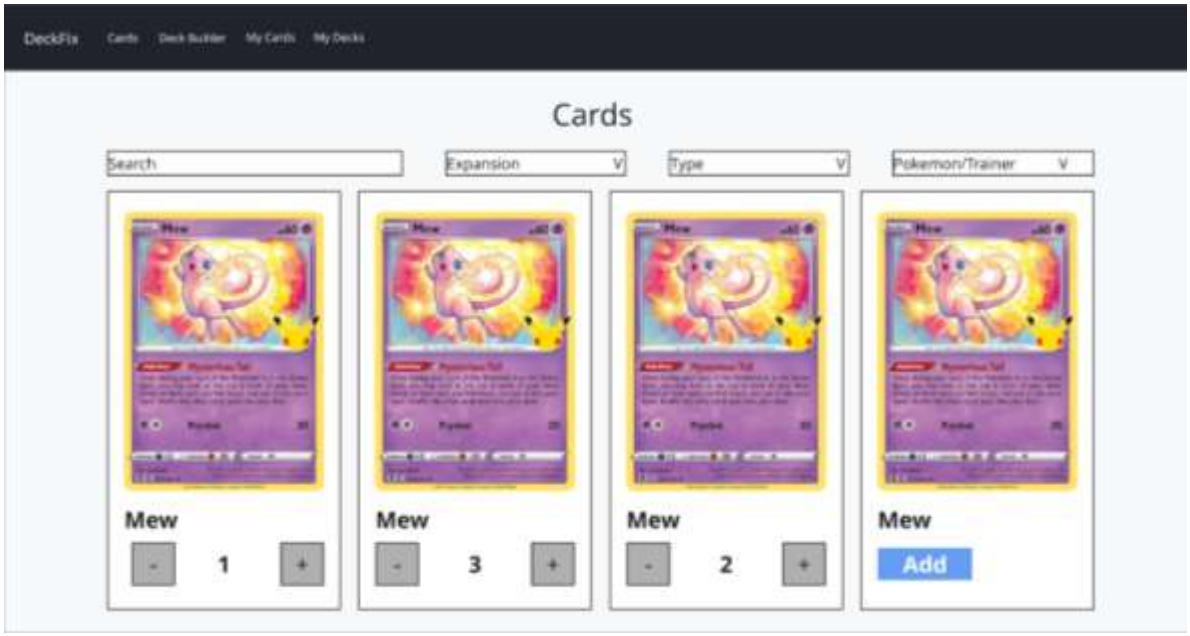

Figure 15: Cards page wireframe

Figure 15 shows a wireframe for the cards page. The top of the page includes a navigation bar that is reused across all of the application's pages. The page itself shows a search bar and filters with a grid of card components below. The cards are automatically filtered by expansion and can be further filtered by energy type and by Pokémon/trainer. The card grid features card components with the card image, name and different buttons that display based on the quantity of the card in the user's collection.

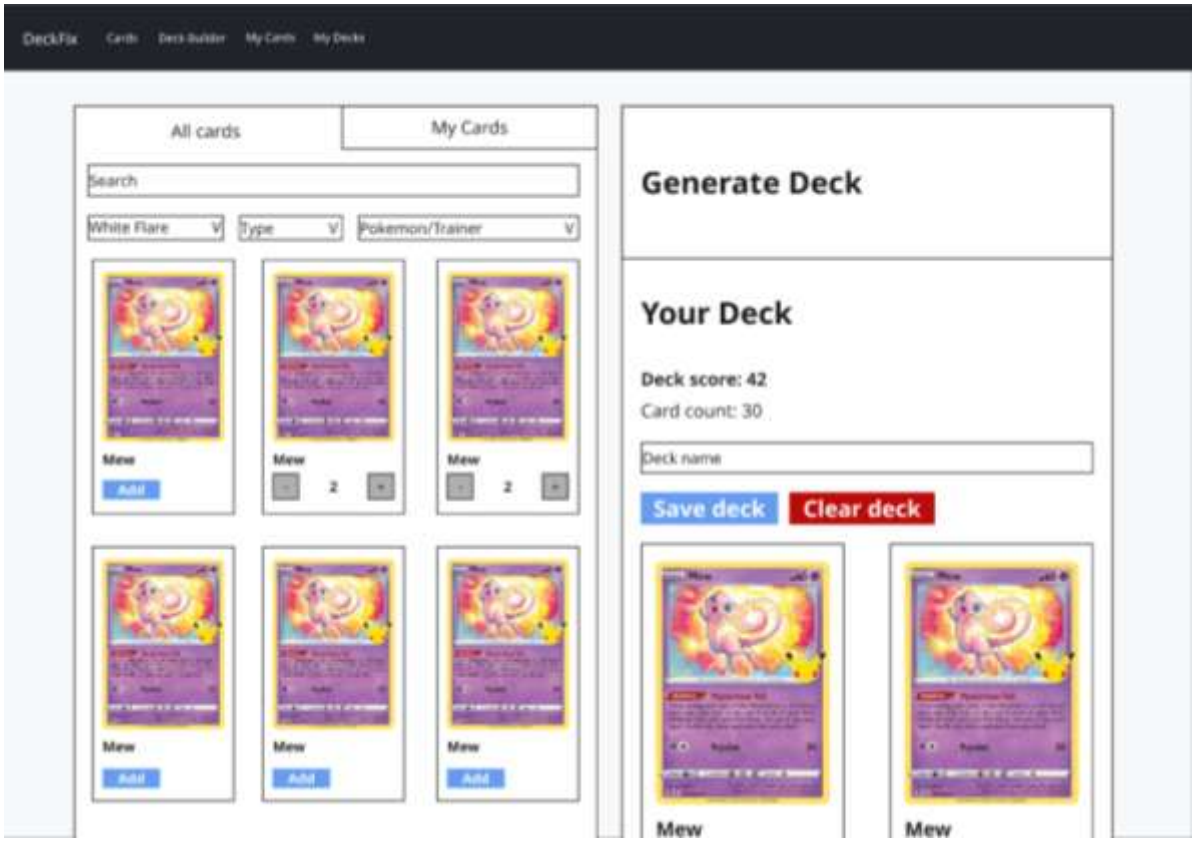

Figure 16: Deck building tool page wireframe

Figure 16 shows a wireframe for the deck building tool page. It includes the same navigation bar as the cards page. On the left, a panel with “All Cards” and “My Cards” tabs is seen. These allow the user to build a deck using all cards or using cards they have saved to their collection. The same search bar and filters can be seen here as in the cards page. On the right side of the page, an accordion can be seen that displays either the deck generation menu (“Generate Deck”) or the save deck menu (“Your Deck”) depending on what is toggled. The Your Deck menu displays the deck’s card count and score. It includes a name input field, a save deck button and a clear deck button. The card components in both panels are the same as in the cards page with the card’s image, name and quantity buttons. However, in the deck building tool page the quantity buttons refer to the card quantity in the deck, as opposed to the quantity of cards saved in the user’s collection.

Both wireframes use a simple style guide to aid user comprehension and navigation. Solid-bootstrap (Overview, n.d.) provides a basic style with a minimalistic colour palette that suits the needs of the application by providing ready-to-use, visually pleasing, reusable components without the need of customisation or alteration.

4.3 Process Design

4.3.1 Frameworks and Libraries

This project uses the following frameworks:

- Express.js: used to build API (includes Express-session and Express-validator).
- Playwright: used to perform end-to-end testing (Installation, 2026).
- Jest: used to test API (Getting Started, n.d.).

This project uses the following libraries:

- SolidJS: used to build user interface (includes solid-meta, solid-bootstrap, solid-router, solid-start and solid-toast).
- Axios: used to fetch Pokemon TCG API image links to publish to the AWS S3 bucket (First steps, 2026).
- Bcrypt: used to encrypt user passwords (node.bcrypt.js, 2026).
- Dotenv: used to load sensitive environment variables (env) into the application to avoid exposing them (dotenv, 2026).
- Helmet: used to set HTTP response headers in the API (helmet, 2026).
- Mongoose: used as an interface between Node.js and MongoDB to simplify database interactions (Getting Started, n.d.).
- Supertest: used to test API (supertest, 2026).

4.3.2 Event-handling

Asynchronous functions were used when performing intense functionality (such as data fetching) to allow for other tasks and events to proceed to avoid long wait times and improve user experience.

Toast notifications were implemented to inform users of event occurrence and loading using solid-toast (solid-toast, 2024). In Figure 17, a toast notification can be seen in the top right corner to alert the user of deck generation.

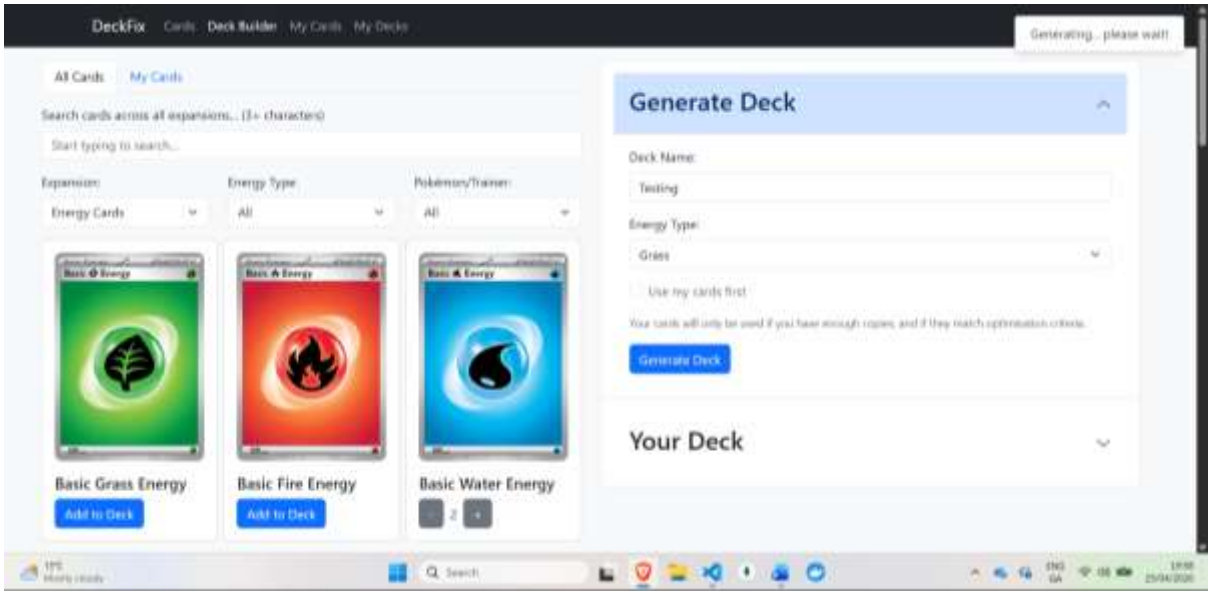

Figure 17: Example of Toast notification

4.3.3 Error-handling

Try–catch statements were implemented when performing different tasks to stop the process when an error occurred, to prevent the rest of the process from attempting to run. This can be seen in Figure 18.

```
try {
  const res = await fetch("/api/auth/login", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    credentials: "include",
    body: JSON.stringify({
      email: email(),
      password: password(),
    }),
  });

  if (!res.ok) {
    const data = await res.json();
    toast.error(data.error);
    return;
  }

  toast.success("Logged in successfully!");
  navigate("/decktool/svenergy", { replace: true });
} catch (err) {
  toast.error("Something went wrong.");
  return;
} finally {
  setLoading(false);
}
};
```

Figure 18: Example of try-catch statement

Toast notifications were implemented to inform users of errors. In Figure 19, a toast notification can be seen in the top right corner to alert the user of a missing deck name.

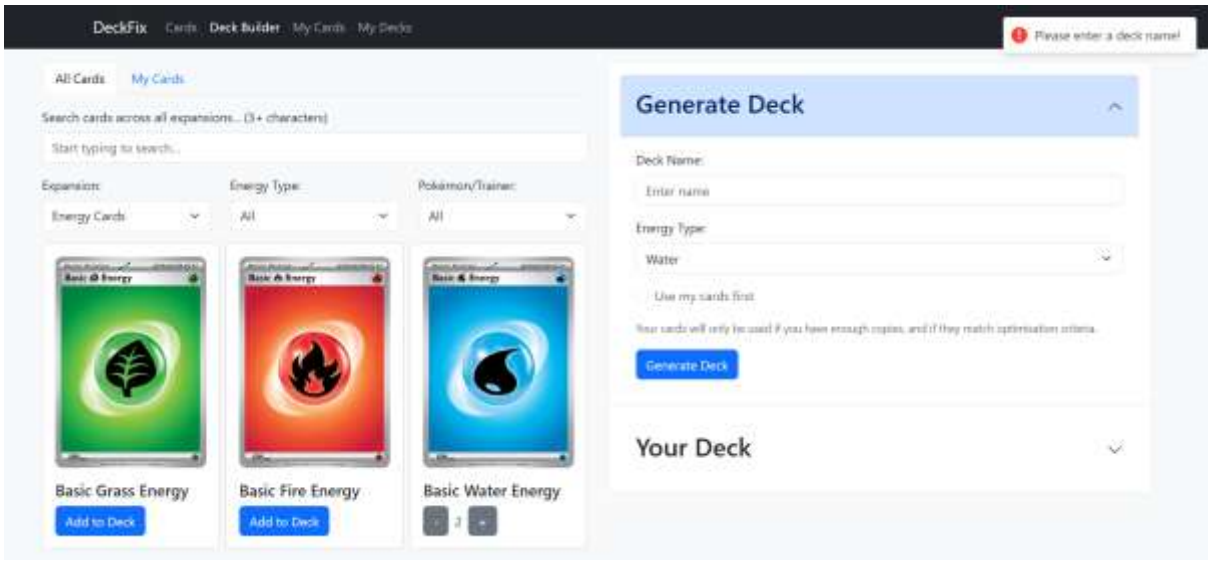

Figure 19: Example of Toast notification

4.4 Database Design

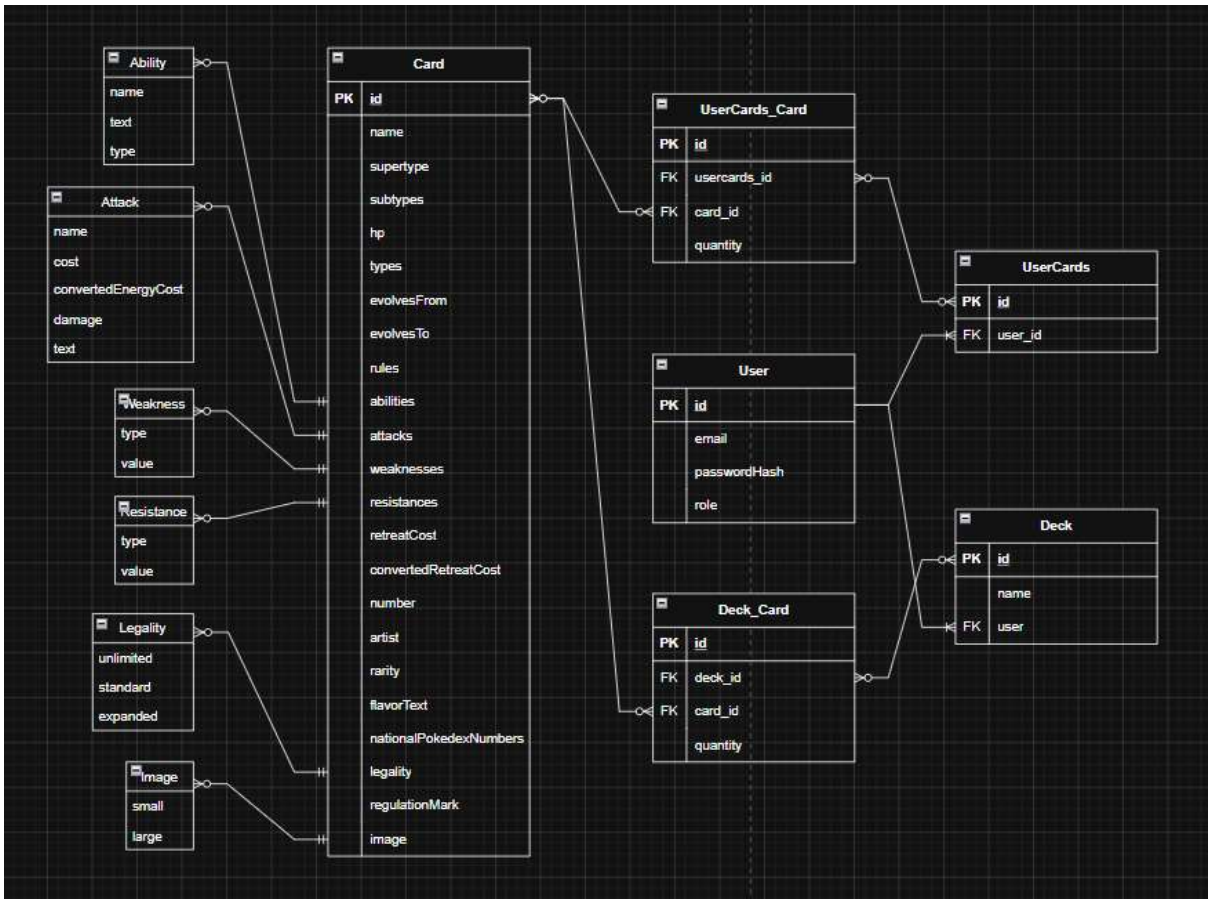

Figure 20: Entity Relationship diagram

Figure 20 depicts an Entity Relationship diagram for the application.

Ability, Attack, Image, Legality, Resistance and Weakness have identifying relationships to the Card class as they cannot exist independently of the Card class.

UserCards_Card and Deck_Card are junction tables between Card and UserCards, and Card and Deck. Card has many-to-many relationships with both UserCards and Deck. In both junction tables, quantity represents the number of copies existing within the usercards collection and deck. It can be seen represented in both usercards and deck in Figure 21.

```
cards: [
  {
    card: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Card",
      required: true,
    },
    quantity: {
      type: Number,
      required: true,
      min: 1,
      default: 1,
    },
  },
],
```

Figure 21: Representation of junction table functionality

User has one-to-many relationships with Deck and UserCards as one user can own many “usercards” and decks, but usercards and decks may only belong to one user.

MongoDB, a NoSQL database, was used for the application. It is a familiar database with helpful associated technologies:

- Mongoose: A library for Node.js that provides an interface to MongoDB, allowing for easier database interacting from within the application.
- MongoDB Compass: Software that provides a graphical user interface that allows for easy data visualisation and manipulation without writing code.
- MongoDB Atlas: A deployment service for MongoDB databases.

5 Implementation

This section describes the project's development methodology, tools and progression.

5.1 Development Methodology

The Scrum software development methodology will be followed to complete this project. Scrum is a part of the Agile approach which both feature short-term development cycles and encourage implementation of feedback. Scrum specifically highlights one to four week sprints and a backlog that tracks what still needs to be done (Coursera Staff, 2025). This methodology was helpful in distinguishing the different project elements and ensuring they are completed in time. One week sprints were implemented in this project's development.

5.2 Sprints

There were 13 sprints across the project dedicated to the application's development.

5.2.1 Sprint One (24th – 30th January)

This sprint was spent on setting up the development environment, tools and beginning the application development. This included setting up a Trello board, a GitHub repository, and continuing the application using the technical feasibility study as a foundation (converted from JavaScript to TypeScript). By the end of the sprint, the application's API had:

- User authentication (register (POST), login (POST) and logout (POST) endpoints with validation).
- Middleware (authentication and error handling).
- An endpoint to get all cards (GET).

The application's user interface included:

- A page that displays cards (including the card image, name, HP and energy types, see Figure 22)

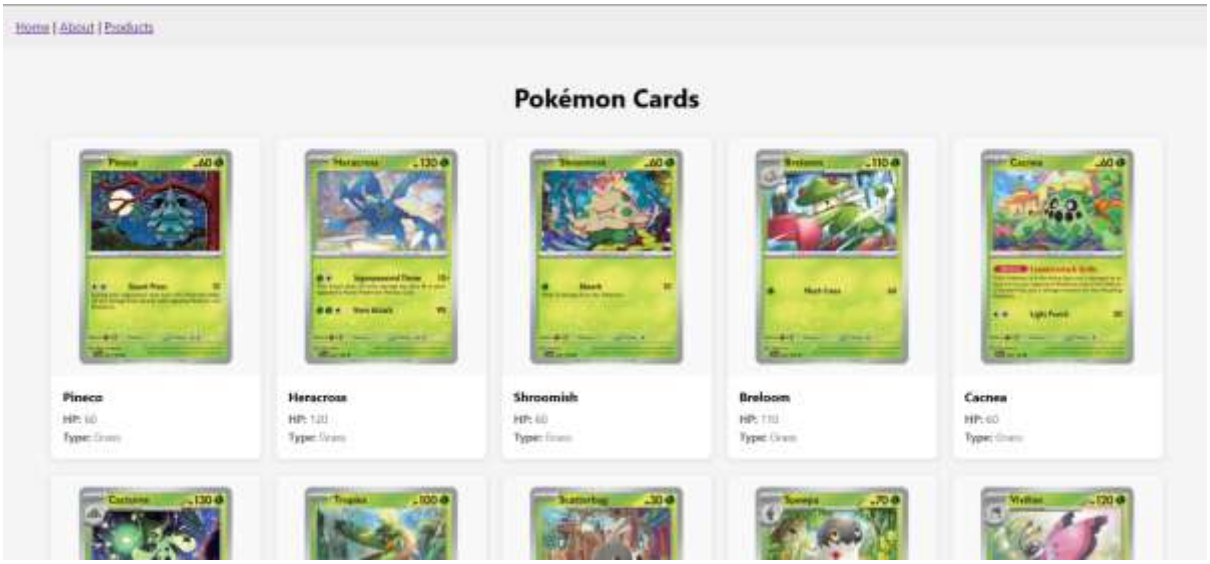

Figure 22: Cards page

5.2.2 Sprint Two (31st January – 4th February)

This sprint continued to develop the application by:

- Seeding the database with all relevant card data using JSON data (Backes, pokemon-tcg-data, 2025).
- Optimising the get all cards endpoint to only respond with data if the request includes an expansion parameter (will only return data from that expansion). This reduces the request load, especially as at this point the application was constantly making hundreds of requests to the Pokémon TCG API for images (Backes, Pokémon TCG API Documentation, 2022). This was reflected in the front end by changing the cards route to cards/expansion (route must contain card expansion parameter to display page).
- Add register and login components to the UI.
- Create usercards endpoints that allow users to get all their saved cards (GET), add a card to their collection (POST) and remove a card from their collection (DELETE).

When meeting with the project supervisor, the following was discussed:

- When should the application be deployed – as soon as possible.
- Is the usercards functionality being handled correctly – yes, the structure of the model and endpoints serve the needs of the application.

The following was not completed this sprint:

- Fixing a bug that, on login/register, returned status 200 instead of the expected 201.
- Setting up application deployment.
- Researching how to design the deck building tool UI.

5.2.3 Sprint Three (5th – 11th February)

This sprint continued to develop the application by:

- Fixing the login/register bug that was not completed in the last sprint.
- Creating a mycards page that shows the user's card collection when authenticated.
- Allow users to add/remove cards from their collection in the cards page.
- Researched how to design the deck building tool UI which was not completed in the last sprint. (Soriano, 2022) was the chosen tutorial, which described a shopping card tutorial in SolidJS. This was to be refactored into the deck building tool.

Pagination was attempted in the cards page but was unsuccessful. Due to the card expansion parameters interfering with the pagination, the cards page was unable to load because of hydration errors. Pagination was removed and the application was restored to a previous state. Application deployment was also attempted but unsuccessful.

A meeting with the project's second reader was attended where he was informed on the project's aims and current progress. Due to unforeseen circumstances the meeting with the project supervisor was cancelled and rescheduled to 16th February in the following sprint.

Due to the struggles with pagination and deployment, the following was not completed:

- Developing a deck building tool page using the research conducted in this sprint.
- Researching how to integrate deck evaluation and generation as the deck building tool was not created.
- Due to the postponed supervisor meeting, application deployment and pagination were not fixed.

5.2.4 Sprint Four (12th – 18th February)

This sprint continued to develop the application by:

- Creating a deck saving (POST) endpoint with basic validation that contained the user's id and cards.
- Converting the shopping cart tutorial into a basic deck building tool page.
- Adding a "catch-all" route that shows the homepage if a user tries to access a route that cannot be found.
- Adding the Kobalte (Introduction, n.d.) component library to improve the user interface.

By the end of this sprint, the deck building tool was extremely basic and not fully functional. A user could only add one of each card to a deck (no copies of the same card possible) and save it.

The following was discussed in the first supervisor meeting (to compensate for the cancelled meeting in the last sprint):

- Hydration errors when implementing pagination – add pagination when higher priority requirements are complete.
- Deployment – discussed fixes. Deployment was fixed.

There were no problems to discuss in the second supervisor meeting (this sprint's meeting), the supervisor was informed of progress and plans for the next sprint.

The following was not completed:

- Fixing a bug that disallowed users from saving a deck after removing cards from it in the deck building tool.

5.2.5 Sprint Five (19th – 25th February)

This sprint continued to develop the application by:

- Fixing the bug that disallowed users from saving a deck after removing cards from it. A PATCH endpoint was necessary to fix this.
- Allowing users to add/remove cards from their collection in the mycards page (page that displays only the cards added to the user's collection).
- Adding a name property to the deck model to allow users to name each deck in the deck building tool.
- Adding deck session functionality to persist the deck building tool's state on refresh. This included three endpoints: get deck session (GET), overwrite the session (POST), add a card to the session (PATCH), remove a card from the session (DELETE) and clear the session (POST).
- Adding a "mydecks" page that shows the user's saved decks.
- Adding a delete deck (DELETE) endpoint and integrating it into the mydecks page.
- Creating an edit deck (PATCH) endpoint.
- Adding a mydecks/id page that displays the details of a chosen deck by its id.
- Adding initial styling to the user interface using the Kobalte component library.

The following was discussed in the supervisor meeting:

- A new error had appeared in the deployed version of the application.
- On refresh, the mycards and mydecks pages did not load data.

The following was not completed:

- Adding basic deck generation with no optimisation, deck rules or user constraints.
- Allowing users to upload a saved deck to the deck building tool to edit it.
- Fixing the new error in the deployed version of the application.
- Fixing the refresh bug in mycards and mydecks.

5.2.6 Sprint Six (26th February – 4th March)

This sprint continued to develop the application by:

- Fixing the refresh bug in mycards and mydecks: the problem lied in using SolidJs' createResource (asynchronous data fetch) without using createSignal (event listener). It was amended by adding a createSignal called shouldFetch and setShouldFetch that triggers on refresh.
- Adding basic deck generation with no optimisation, deck rules or user constraints.
- Integrated card styling from solid-bootstrap as the Kobalte library had none.
- Adding toast notifications to the user interface using solid-toast.
- Fixing the deployment error.

The following was discussed in the supervisor meeting:

- Should the Kobalte library be removed in favour of solid-bootstrap – yes. The Kobalte library was removed and replaced with solid-bootstrap.

The following was not completed:

- Allowing users to upload a saved deck to the deck building tool to edit it.

5.2.6.1 First iteration of deck generation

Deck generation by the end of this sprint was written to choose 60 random cards from the entire card data pool. It was extremely basic and produced unplayable decks due to being completely random. It incorporated no optimisation, deck rules or user constraints.

5.2.7 Sprint Seven (5th – 11th March)

This sprint continued to develop the application by:

- Allowing users to add more than one copy of the same card in the deck building tool and to their collection. This should have been implemented much sooner.
- Adding deck rule validation to deck generation, save and edit (see Appendix B). This is intended to prevent extreme database uploads: a user cannot save a deck with, for example, 1000 cards or 100 copies of the same card. However, a user can save a deck with less than 60 cards, in case they want to save the deck as a draft and return to editing it later.

- Adding error toast notifications that show when a user tries to save an invalid deck.
- Allowing users to use usercards in the deck building tool.
- Adding optimisation to deck generation.

Due to unforeseen circumstances the meeting with the project supervisor was cancelled and rescheduled to 12th March, in the following sprint.

The following was not completed:

- Allowing users to use usercards in deck generation (first user constraint).
- Deciding on a layout for the deck building tool page and making a prototype.

5.2.7.1 First Iteration of Deck Optimisation Criteria

The following research was conducted before the official start of the project and is improved upon in later sprints. It is integrated into the second iteration of deck generation.

Upon further investigation, the concept of a "Main Attacker" grew prevalent in resources that documented effective decks (JustInBasil, A Basic Guide to Deck Building in the Standard Format, 2024). This suggested that a deck should contain a primary Pokémon with high health points (HP) that the rest of the deck works around. The following changes were implemented into the optimisation criteria:

3-4 Basic Pokémon that evolve: If a player chooses to include Stage 2 Pokémon in their deck, they should include 3-4 cards of that Pokémon's Basic form. As the Stage 2 form cannot be played until the evolution process has occurred, there must be 3-4 cards of the Basic form in the deck, so that the chance of drawing the cards is larger, allowing the evolution process to occur faster.

Cards with two attacks: As an energy card can only be attached to a Pokémon once per turn, and stronger attacks require more energy, it is advised to include cards with two attacks. As the Pokémon is collecting energy cards, it can use the weaker attack on

the opponent's Pokémon, while it is charging the more effective attack (see Figure 23).

Figure 23: Pokémon card with two attacks

5-9 EX cards: An effective deck should include 5-9 EX cards.

EX cards with Colourless energy attacks: A deck is usually built around one or two energy types, meaning the Pokémon and energy will be of that energy type. This enables the player to perform attacks that require a certain type of energy more efficiently: if the deck were to consist of too many energy types, the player would be unable to draw the energy cards they need for an attack. But because every energy type has a type they are weak to, efficient decks will include cards of different types with attacks that only require Colourless energy, meaning they can perform an attack using the deck's primary energy type cards. This will prevent a deck being entirely weak to one type.

Cards that include abilities: Abilities are essential to offer as much support to a deck as possible.

Inclusion of abilities, attacks or trainer cards that allow the player to draw cards: Drawing cards is extremely important as it allows players to find the cards they need as quickly as possible. A player can draw a card once per turn, but more if they have an ability, trainer card or attack that allows it. Efficient decks will include abilities, attacks or trainer cards that allow a player to draw more cards per turn to allow the player to find the cards they need in their deck.

Over 30 trainer cards and 10-15 energy cards: The Pokémon chosen for a deck should sustain a game by being powerful enough to not require a large team. This will allow for a low overall energy card cost (hence the low energy card amount). Together, the low amount of Pokémon and energy cards will allow for a higher trainer amount, which will benefit the player's Pokémon and can sabotage the other player.

Inclusion of an ACE-SPEC card: The ACE-SPEC is a type of powerful trainer card that is essential to an effective deck.

5.2.7.2 Second Iteration of Deck Generation

Deck generation by the end of this sprint correlated with the first iteration of deck optimisation research and was written to:

- Choose a random energy type.
- Pick two basic Pokémon of that/Colourless energy type with abilities. Add two copies of each.
- Add two copies of the appropriate Stage 1 Pokémon that evolve from the chosen basic Pokémon. Prioritise EX cards and abilities.
- Add basic EX Pokémon cards with Colourless energy attacks until the total Pokémon count is 15.
- Add an ACE-SPEC card.
- Add 30 trainer cards that allow for drawing cards.
- Fill the deck with energy cards of the chosen energy type until the deck contains 60 cards.

This iteration was still extremely basic and produced valid, yet unplayable decks due to insufficient energy cards (limited to four copies like all other cards), broad energy type variance and no Stage 2 Pokémon. It incorporated no user constraints. Figures 24 and 25 depict the decks that were created by this iteration of the deck generation.

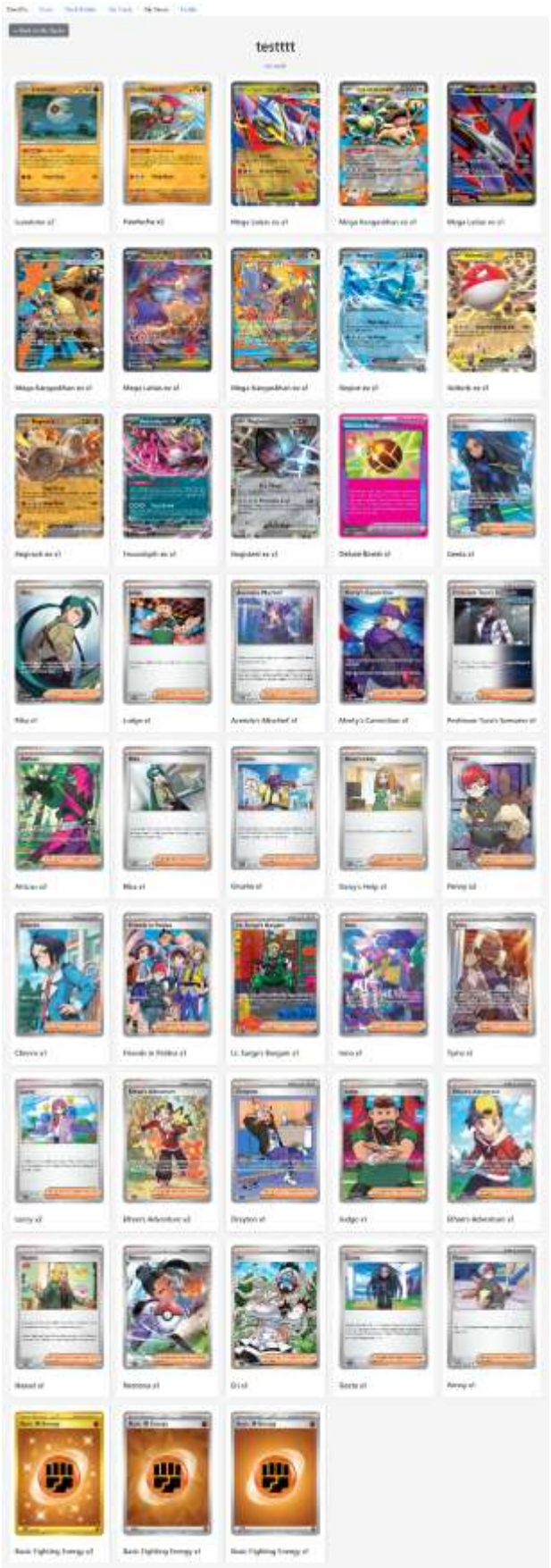

Figure 24: First example of generated deck

Figure 25: Second example of generated deck

5.2.8 Sprint Eight (12th – 18th March)

This sprint continued to develop the application by:

- Creating the first iteration of the deck evaluation system.
- Improving deck generation.

- Add user constraints to deck generation that allow the user to use usercards and to choose an energy type for the deck.

There were no problems to discuss in both (this and last sprint's) supervisor meetings. The supervisor was informed of progress and plans for the next sprint.

5.2.8.1 Second Iteration of Deck Optimisation Criteria

The following research improved on the findings in the first iteration of deck optimisation criteria and was integrated into the third iteration of deck generation.

Main Attacker: Four copies of a Pokémon with minimum 300 HP and an ability or two attacks should exist within the deck. Four copies of the card ensure it is more likely to be drawn and can be used sooner.

Rare candy: If the Main Attacker is a Stage 2 Pokémon, the deck should have four copies of the "Rare Candy" trainer card (see Figure 26). This boosts the evolution process by skipping the Stage 1 step.

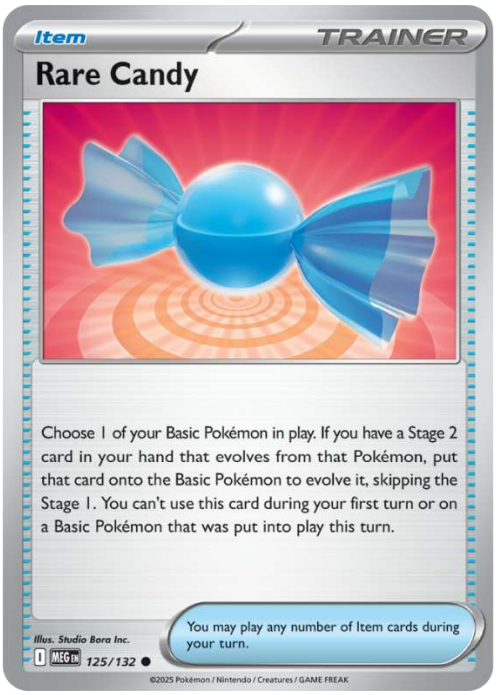

Figure 26: Rare Candy card

Secondary attackers: A deck should contain secondary attackers, which are Pokémon that can be used while the Main Attacker is being prepared. They should also have an ability or two attacks.

Utility Pokémon: A deck should contain Pokémon that are not intended for attacking but for their abilities.

Trainer cards: Trainer cards that allow for drawing cards, searching the deck, switching your or your opponent's Pokémon out of the active spot and healing Pokémon should be prioritized.

5.2.8.2 Third Iteration of Deck Generation

Deck generation by the end of this sprint correlated with the second iteration of deck optimisation research and was written to:

- Add four Main Attacker Pokémon cards with minimum 300 HP. Prioritise cards with two attacks or an ability that allows for drawing cards.
    - If Main Attacker is Stage 2: add two Stage 1 pre-evolution cards, four Basic pre-evolution cards, and four Rare Candy cards.
    - If Main Attacker is Stage 1: add four Basic pre-evolution cards.
    - If Main Attacker is Basic: continue.
- Add five non-EX Pokémon cards with abilities that match the energy type of the Main Attacker. If the Main Attacker is of Dragon energy type, add Colourless energy type Pokémon.
- Add 4-11 EX Pokémon cards that match the energy type of the Main Attacker or have only Colourless attacks. They should have two attacks or an ability. If the Main Attacker is of Dragon energy type, add Colourless energy type Pokémon.
- Add an ACE-SPEC card.
- Add trainer cards until the deck has 50 cards. The cards should prioritise card drawing, deck searching, switching your or your opponent's active Pokémon and healing.
- Add 10 energy cards that pertain to the Main Attacker's strongest attack. Basic energy cards are no longer restricted to four copies per deck like other cards.
    - If it has one energy type: add 10 energy cards of that type.
    - If it has more (common in Dragon energy type/Tera Pokémon): proportionally add 10 energy cards divided by energy type. For example, if the attack requires two Fire energy and one Water energy, add seven Fire energy cards and three Water energy cards.
    - If the attack requires only Colourless energy: add 10 Psychic energy cards.

This iteration improved on the previous iteration by creating valid *and* playable decks. It improved on the flaws of the previous iteration by changing the script to prioritise the Main Attacker Pokémon. It could now incorporate user constraints. Figures 27 and 28 below depict the decks that were created by this iteration of the deck generation.

Figure 27: First example of generated deck

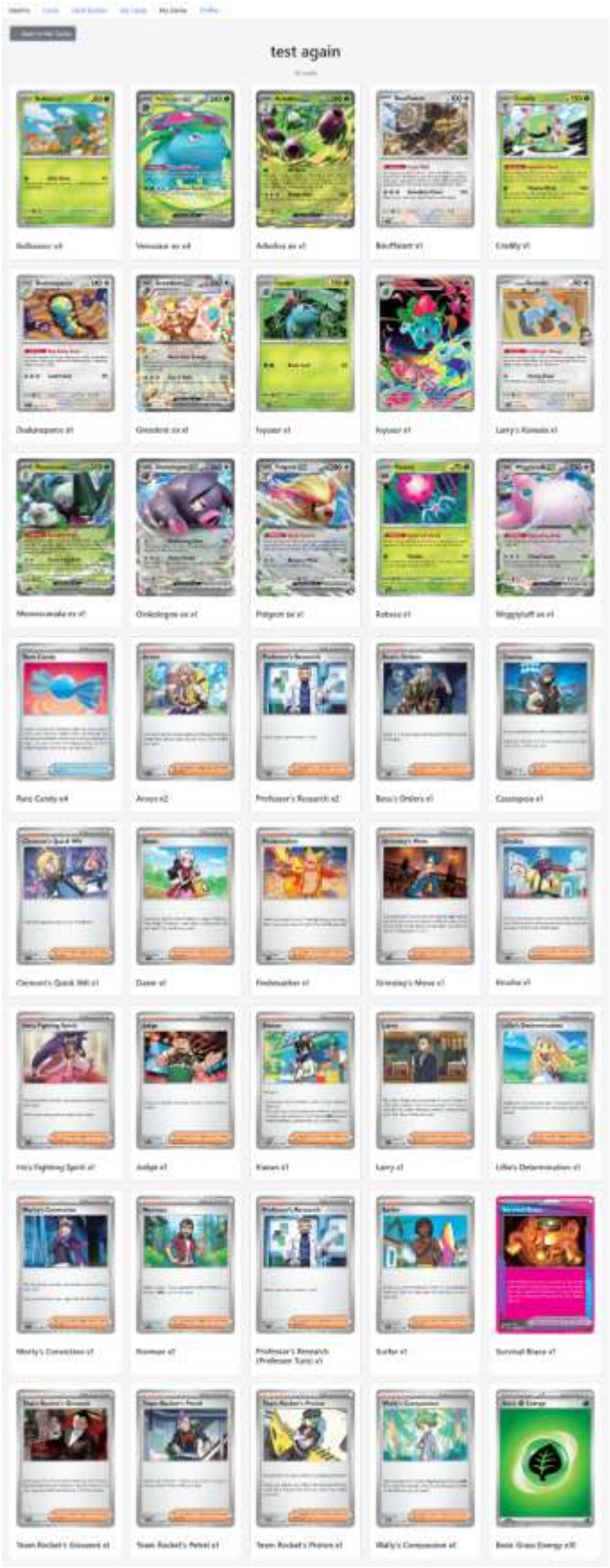

Figure 28: Second example of generated deck

5.2.8.3 First Iteration of Deck Evaluation

Deck evaluation was designed to iterate over each card in the deck and assign points where appropriate to find a score out of 100. By the end of this sprint, deck evaluation correlated with the second iteration of deck optimisation research and was written to assign:

- +1 point for each Pokémon card over 260 HP (maximum +10 points).
- +2 points for each Pokémon card with an ability (maximum +20 points).
- +2 points for each Pokémon card with two attacks (maximum +20 points).
- +10 points for including an ACE-SPEC card.
- +2 points for each card that allows for drawing cards (maximum +10 points).
- +2 points for each card that allows for searching your deck (maximum +10 points).
- +2 points for each card that allows for switching your/your opponent's active Pokémon (maximum +10 points).
- +2 points for each card that allows for healing (maximum +10 points).
- -5 points for every extra energy type card requirement (maximum -15 points) (not including Pokémon of Colourless or Dragon energy type).
- -5 points for each energy card over 20 energy cards (for example, -15 points if 23 energy cards).
- -5 points for each card under 20 trainer cards (for example, -15 points if 17 trainer cards).

5.2.9 Sprint Nine (19th – 25th March)

This sprint continued to develop the application by:

- Sorting the cards in the cards page by expansion id.
- Sorting the cards in mydecks/id by supertype: Pokémon, Trainer then Energy. Each supertype is then sorted by card copy quantity in descending order.
- Sorting the cards in mycards by supertype.
- Integrating the evaluation system into the deck building tool by adding a deck score (see Figure 29).
- Finalising card rarity filters in generation: if using usercards, full-art (see Appendix B) cards can be used, but otherwise cards of normal rarity should be used.
- Adding filters to the cards page, mycards page and the deck building tool to allow filtering my Pokémon energy type and Pokémon/trainer.
- Improving the deck evaluation system.
- Improving deck generation by implementing the deck evaluation system, generating 100 decks and returning the deck with the best score.

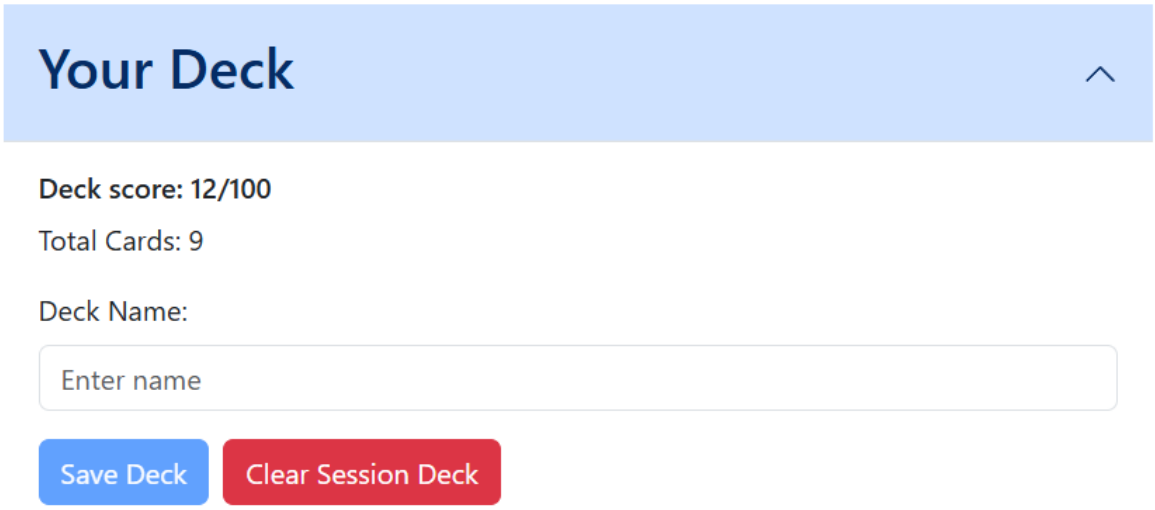

Figure 29: Deck score in deck building tool

Due to unforeseen circumstances the meeting with the project supervisor was cancelled and rescheduled to 26th March, in the following sprint.

The following was not completed:

- Adding search functionality to search by card name in the cards page, mycards page and the deck building tool.

5.2.9.1 Second Iteration of Deck Evaluation

By the end of this sprint, deck evaluation still correlated with the second iteration of deck optimisation research found in Sprint Eight and was written to assign:

- +3 points for each Pokémon card over 260 HP (maximum +15 points).
- +3 points for each Pokémon card with an ability (maximum +15 points).
- +3 points for each Pokémon card with two attacks (maximum +15 points).
- +10 points for including an ACE-SPEC card.
- +1 point for each card that allows for drawing cards (maximum +20 points).
- +1 point for each card that allows for searching your deck (maximum +5 points).
- +1 point for each card that allows for switching your/your opponent's active Pokémon (maximum +15 points).
- +1 point for each card that allows for healing (maximum +5 points).
- -5 points for each unique energy requirement over three (maximum -15 points).
- -5 points for each energy card over 20 energy cards.
- -5 points for each trainer card under 20 trainer cards.

5.2.10 Sprint Ten (26th March – 1st April)

This sprint continued to develop the application by:

- Allowing users to upload a saved deck to the deck building tool to edit it. This required the deck building tool route to accept a deck ID parameter in addition to

the card expansion parameter. The deck building tool was refactored from a route into a component that can be loaded into either a regular deck building tool page (route containing card expansion parameter) or a deck editing tool page (route containing deck ID parameter).

- Adding search functionality to search by card name in the cards page, mycards page and the deck building tool.
- Styling the user interface.
- Adding a deck legality check in mydecks/id (see Figure 30). A deck is considered legal if:
    - It contains exactly 60 cards.
    - It contains a Basic Pokémon card.
    - It contains all necessary evolution stages for each Pokémon card.
    - It contains an energy card.
- Adding a pop-up with information for user testers (see Figure 31). It shows when the homepage is accessed.
- Improving deck evaluation and generation.

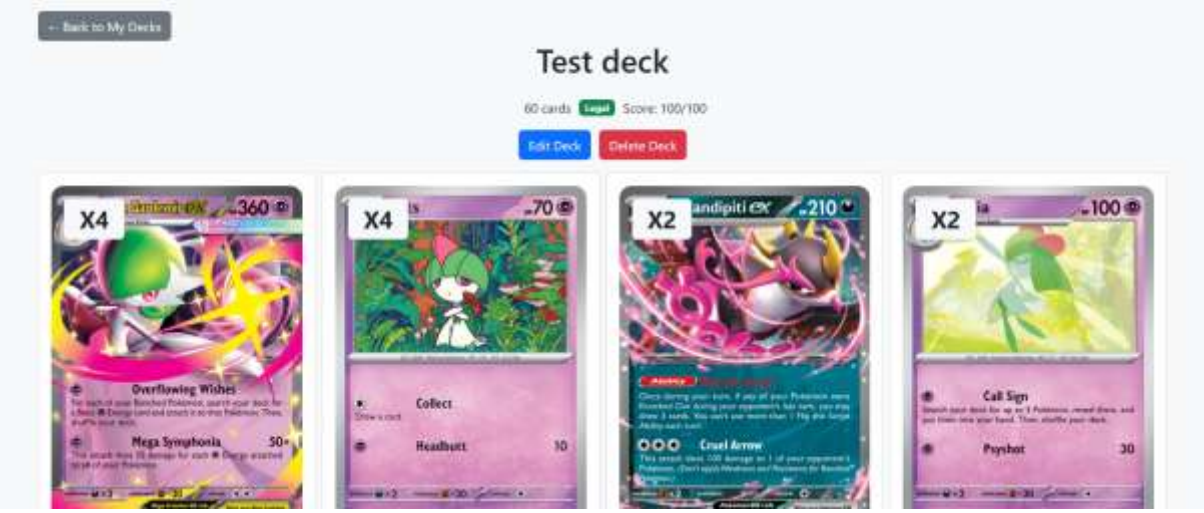

Figure 30: Deck legality check

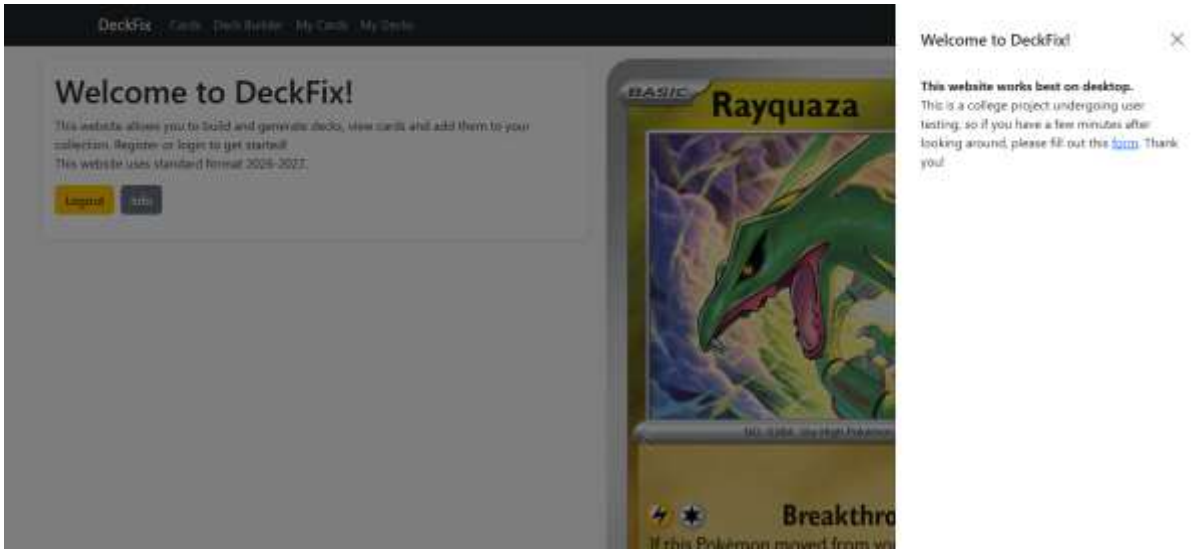

Figure 31: Homepage pop-up for user testing

There were no problems to discuss in both (this and last sprint's) supervisor meetings. The supervisor was informed of progress and plans for the next sprint.

5.2.10.1 Final Iteration of Deck Generation

The final iteration of deck generation correlated with the final iteration of deck optimisation research which can be found in the Research section of this document and was written to:

- Add four Main Attacker Pokémon cards with minimum 280 HP (300 HP is too strict). Prioritise cards with two attacks or an ability that allows for drawing cards.
    - If Main Attacker is Stage 2: add two Stage 1 pre-evolution cards, four Basic pre-evolution cards, and four Rare Candy cards.
    - If Main Attacker is Stage 1: add four Basic pre-evolution cards.
    - If Main Attacker is Basic: continue.
- Add specific (not randomised) utility Pokémon cards:
    - If the Main Attacker is N's Zoroark EX, add N's Reshiram and N's Zekrom.
    - If the Main Attacker is of Darkness energy type, add Munkidori.
    - If the Main Attacker is of Fighting energy type, add Lunatone and Solrock from the Ascended Heroes expansion.
    - For all other Main Attacker energy types, add Latias EX and Fezandipiti EX.
- Add an ACE-SPEC card.
- Add trainer cards that pertain to the Main Attacker (their energy type, prefix or subtype) until the deck has 48 cards. Then add cards that allow for card drawing, switching your or your opponent's active Pokémon and healing. (Deck searching trainer cards were removed as they were too specific to be used efficiently.)
- Add 12 energy cards that pertain to the Main Attacker's strongest attack.
    - If it has one energy type: add 12 energy cards of that type.

- If it has more: proportionally add 12 energy cards divided by energy type.
- If the attack requires only Colourless energy: add 12 Psychic energy cards.

This iteration improved on the last iteration by wholly focusing on the Main Attacker. It removed irrelevant cards such as secondary attackers and geared the trainer cards and utility Pokémon to pertain to the Main Attacker. Figures 32 and 33 depict the decks that were created by this iteration of the deck generation.

Figure 32: First example of deck generation

Figure 33: Second example of deck generation

5.2.10.2 Final Iteration of Deck Evaluation

By the end of this sprint, deck evaluation correlated with the final iteration of deck optimisation research which can be found in the Research section of this document and was written to assign:

- +5 points for each Pokémon card over 280 hp (maximum +20 points).
- +5 points for each Pokémon card with an ability (maximum +20 points).
- +5 points for each Pokémon card with two attacks (maximum +20 points).
- +5 points for the inclusion of an ACE-SPEC card.
- +3 points for each card that allows for drawing cards (maximum +15 points).
- +1 points for each card that allows for switching your/your opponent's active Pokémon (maximum +5 points).
- +1 points for each card that allows for healing (maximum +5 points).
- +2 points for each card that matches energy type (maximum +10 points).
- -5 points for each energy card over 16 energy cards.
- -5 points for each energy card under 25 supporter cards.

5.2.11 Sprint Eleven (2nd – 8th April)

This sprint continued to develop the application by:

- Styling the user interface.
- Securing the card images by uploading them to an AWS S3 bucket. This was done by writing a script that extracted the URL from each card's image field and uploading it to the bucket. This ensured that the images would always be accessible and controlled by the project owner and be independent from a third party API. Fetching the S3 bucket images was intense for the server memory, so they were integrated as a backup to the third party API image links, meaning if the links failed an image would still be available.
- Conducting user testing.

There were no problems to discuss in the supervisor meeting. The supervisor was informed of progress and plans for the next sprint.

The following was not completed:

- Conducting automated and manual testing.

5.2.12 Sprint Twelve (9th – 15th April)

This sprint continued to develop the application by:

- Optimising image load by first fetching images from the S3 bucket and using the third party API images as backup. AWS CloudFront and lazy loading was implemented to reduce server memory strain.
- Conducting automated and manual testing.

- Integrating user feedback.

There were no problems to discuss in the supervisor meeting. The supervisor was informed of progress and plans for the report.

5.2.13 Sprint Thirteen (16th – 22nd April)

This sprint continued to develop the application by:

- Improving the application's homepage by adding a carousel that displays different cards.
- Integrating user feedback.

There were no problems to discuss in the supervisor meeting. The supervisor was informed of progress and plans for the report.

5.3 Challenges

During the application's development, the following challenges were faced:

5.3.1 Learning SolidJS

SolidJS was chosen as the preferred front end technology due to its unfamiliarity, similarity to React.js and suitability for the application's needs. However, as it had not been used by the author previously it posed challenges that were overcame during the research and implementation stages of the project. Resources such as (Quick start, 2026), (snnsnn, 2025) and (Soriano, 2022) were useful in familiarising the author with the new technology by providing complete application development processes to analyse.

The first challenge involved learning SolidJS' terminology. Functions such as createEffect and createSignal were essential to the success of the application but were difficult to accustom to.

Hydration and uncaught client exception errors were common in the first few sprints of the project's implementation. They were intimidating as they would prevent a page from displaying and the associated messages were vague.

Due to SolidJS being a relatively new technology, there are fewer resources available such as question threads on StackOverflow. This affected application debugging as encountered bugs were harder to contextualise.

5.3.2 Application Deployment

The application required stable deployment to conduct user testing. Render was the preferred deployment service as it had been used previously in the academic course. However, due to the new technologies (SolidJS and Vinxi), deployment was a major

challenge throughout the project's implementation. It took several sprints to stabilise the application's deployment and ensure it could accommodate future project editions.

5.4 Final Application Structure

The front end build of the application is inserted into the back end in a folder named dist. The application connects to the database, then the server is ran from the back end and is launched at "index.mjs", which is the application's front end homepage route in the back end dist folder. The API routes all start with "/api" to prevent conflict with the front end routes (see Figure 34). The front end uses an environment variable called VITE_API_URL which is set to the API's index to make all requests to the API (see Figures 35 and 36).

```
//deck session for refresh persistance
app.use("/api/decksession", deckSessionRouter);

//register and login
app.use("/api/auth", authRouter);

//no auth required route for card browsing
app.use("/api/cards", cardRouter);

//auth required route for user-owned card browsing
app.use("/api/usercards", userCardsRouter);

//(for now) no auth required route for deck tool
app.use("/api/decks", requireRole("user"), deckRouter);

//unknown API endpoints return error messages
app.use("/api", unknownEndpoint);
```

Figure 34: API routes

```
export default defineConfig({
  base: "/_build/assets",
  resolve: {
    alias: {
      "~": fileURLToPath(new URL("./src", import.meta.url)),
    },
  },
  server: {
    proxy: {
      "/api": {
        target: process.env.VITE_API_URL || "http://localhost:3001",
        changeOrigin: true,
        secure: false,
      },
    },
  },
});
```

Figure 35: Configuration of API in the front end

```
export const fetchUserDecks = async (): Promise<Deck[] | null> => {
  try {
    const res = await fetch("/api/decks", {
      credentials: "include",
    });

    if (res.status === 401) {
      return null;
    }

    if (!res.ok) {
      throw new Error("Failed to fetch user cards");
    }

    return await res.json();
  } catch (err) {
    return [];
  }
};
```

Figure 36: Example of API request in the front end

The application is deployed as a single web service on Render. It is structured the same way as the local version, by inserting the front end build into the back end and serving the application from the front end homepage found in the dist folder.

5.5 Development Environment and Tools

The application was developed in Visual Studio Code, a familiar code editor. The application was run locally on a Node.js development server to view changes and test functionality. Postman was used to perform API requests. MongoDB Compass was used to view the application's database.

Resources such as Stack Overflow and AI tools such as GitHub Copilot and ChatGPT were used when debugging.

Render was used to deploy the deployed application.

6 Testing and Evaluation

This section describes the testing conducted to ensure the successful functioning of the application, and an evaluation of the finished application.

Resources such as Stack Overflow and AI tools such as GitHub Copilot and ChatGPT were used for debugging.

Sprint	Testing conducted	Types of tests
Sprints One – Thirteen (24th January – 22nd April)	- Manual testing	- Application testing through Node.js local development server - Postman API testing
Sprint Eleven (2nd – 8th April)	- User testing	- User testing
Sprint Twelve (9th – 15th April)	- Automated testing - Manual testing - Continued user testing	- API unit testing - API integration testing - End-to-end testing - Deck evaluation testing - User testing

6.1 Automated Testing

Supertest, Jest and Playwright were used to conduct automated testing. Supertest is a Node.js library that was used to simulate requests to the API. Jest is a JavaScript testing

framework that was used with Supertest to conduct API unit and integration testing by validating that the API's responses matched expected results. Playwright is a testing framework that was used to conduct end-to-end testing by simulating user workflows to ensure the application behaved as expected.

6.1.1 Unit Testing

Unit testing is a testing method in which individual components of an application are tested to verify they work as expected (Unit Testing, 2026). Supertest and Jest were used to conduct unit testing.

The following API middleware unit tests were conducted:

- Should call next() if user is authenticated.
- Should throw UNAUTHORISED if there is no session.

The following API user model unit tests were conducted:

- Should hash a password.
- Should generate different hashes for the same password.
- Should hash different passwords differently.
- Should verify correct password.
- Should reject incorrect password.
- Should reject empty password.

6.1.2 Integration Testing

Integration testing is a testing method in which the interactions and data exchanges between different components of an application are tested to verify they behave as expected. (Integration Testing, 2026). Integration testing was conducted after unit testing to ensure the individual components worked before testing their interactions with other components. Supertest and Jest were used to conduct integration testing.

The following API authentication integration tests were conducted:

- Should register a new user with valid credentials.
- Should not register a user with a duplicate email.
- Should reject registration with invalid email.
- Should reject registration with weak password.
- Should reject registration without uppercase letter in password.
- Should reject registration without number in password.
- Should reject registration with missing fields.
- Should set session cookie upon successful registration.
- Should login with correct credentials.
- Should reject login with incorrect password.
- Should reject login with non-existent email.

- Should reject login with missing fields.
- Should reject login with invalid email format.
- Should logout and clear session cookie.
- Should allow logout even without session.
- Should return user information when authenticated.
- Should return unauthenticated when no session.
- Should return unauthenticated after logout.
- Should maintain session after multiple requests.

The following API deck integration tests were conducted:

- Should create a new deck with authentication.
- Should retrieve only the authenticated user's decks.
- Should delete a user's own deck.
- Should not allow deleting another user's deck.
- Should return 400 for invalid deck ID format.
- Should return 404 for non-existent deck ID.

6.1.3 End-to-end Testing

End-to-end testing is a testing method in which the interactions between different layers of an application (database, API and user interface) are tested together by simulating a user workflow (What is End to End Testing?, 2025). End-to-end testing was conducted after unit and integration testing to ensure the back end was behaving as expected before testing its interactions with the front end. Playwright was used to conduct end-to-end testing.

The following end-to-end tests were conducted:

- The front page (homepage) can be opened.
- A user can register with a new account.
- A user can log in.
- A deck can be created with one card.
- A new deck can be generated.
- The user can log out.

6.1.4 Results

The unit and integration tests can be seen passing successfully in Figure 37.

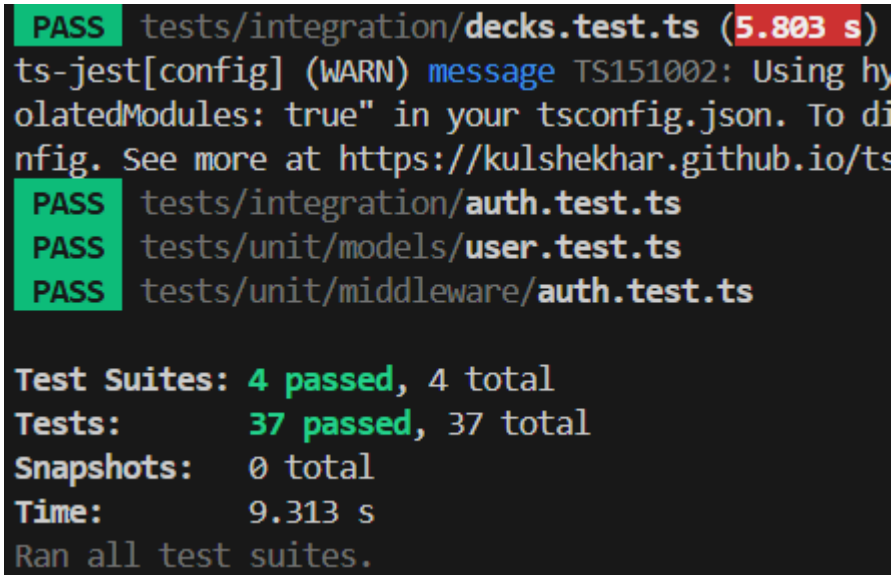

Figure 37: Results of unit and integration testing

The end-to-end tests can be seen passing successfully in Figure 38.

Figure 38: Results of end-to-end testing

6.2 Manual Testing

Postman was used to test API endpoints. The application was tested using a Node.js local development server when developing.

6.2.1 Deck Evaluation Testing

Three tournament decks (see Figures 39-42) were used to test the accuracy of the project's deck evaluation. The decks came in 1st and 2nd place respectively. The following changes were made to accommodate the recent changes to Standard Format:

- Iono was replaced by Judge.
- Hawlucha from the Scarlet&Violet expansion was replaced with Prismatic Evolution's Hawlucha.

- Counter Catcher was replaced by Boss's Orders.
- Arven was replaced by Team Rocket’s Petrel.
- Technical Machine: Evolution was replaced by Rare Candy.
- Technical Machine: Devolution was replaced by Repel.
- Artazon was replaced by Lumiose City.
- Super Rod was replaced by Night Stretcher.
- Nest Ball was replaced by Dawn.
- Professor Turo’s Scenario was replaced by Night Stretcher.
- Pal Pad was replaced by Judge.
- In the tournament deck resource, the 3rd place deck details were rounded to 57 cards. For testing, more copies of N’s Pokémon were added to test a 60 card deck.

Efforts were made to keep card replacements as true to the original cards as possible.

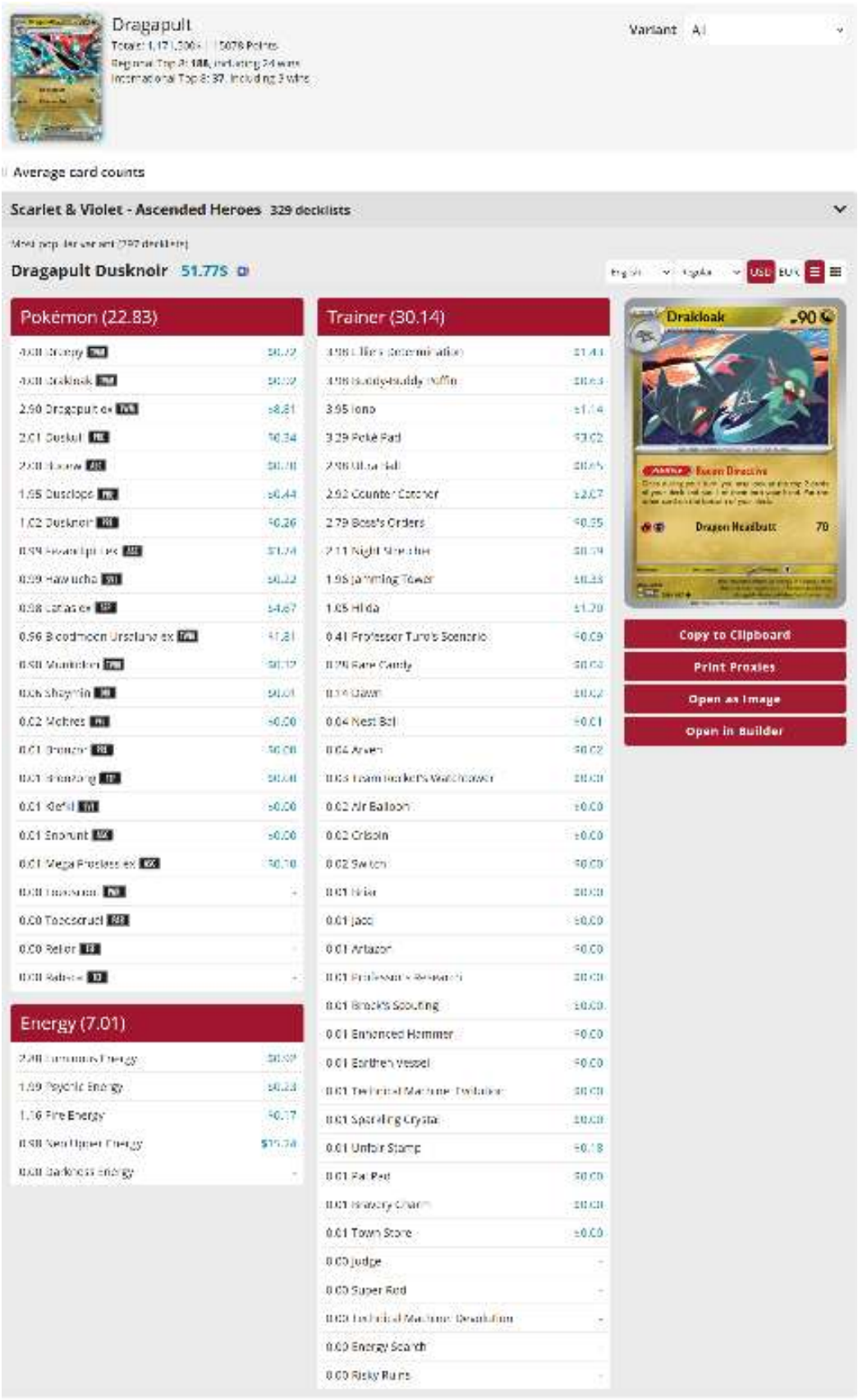

Figure 39: First tournament deck (1st place)

Figure 40: Evaluation of first tournament deck

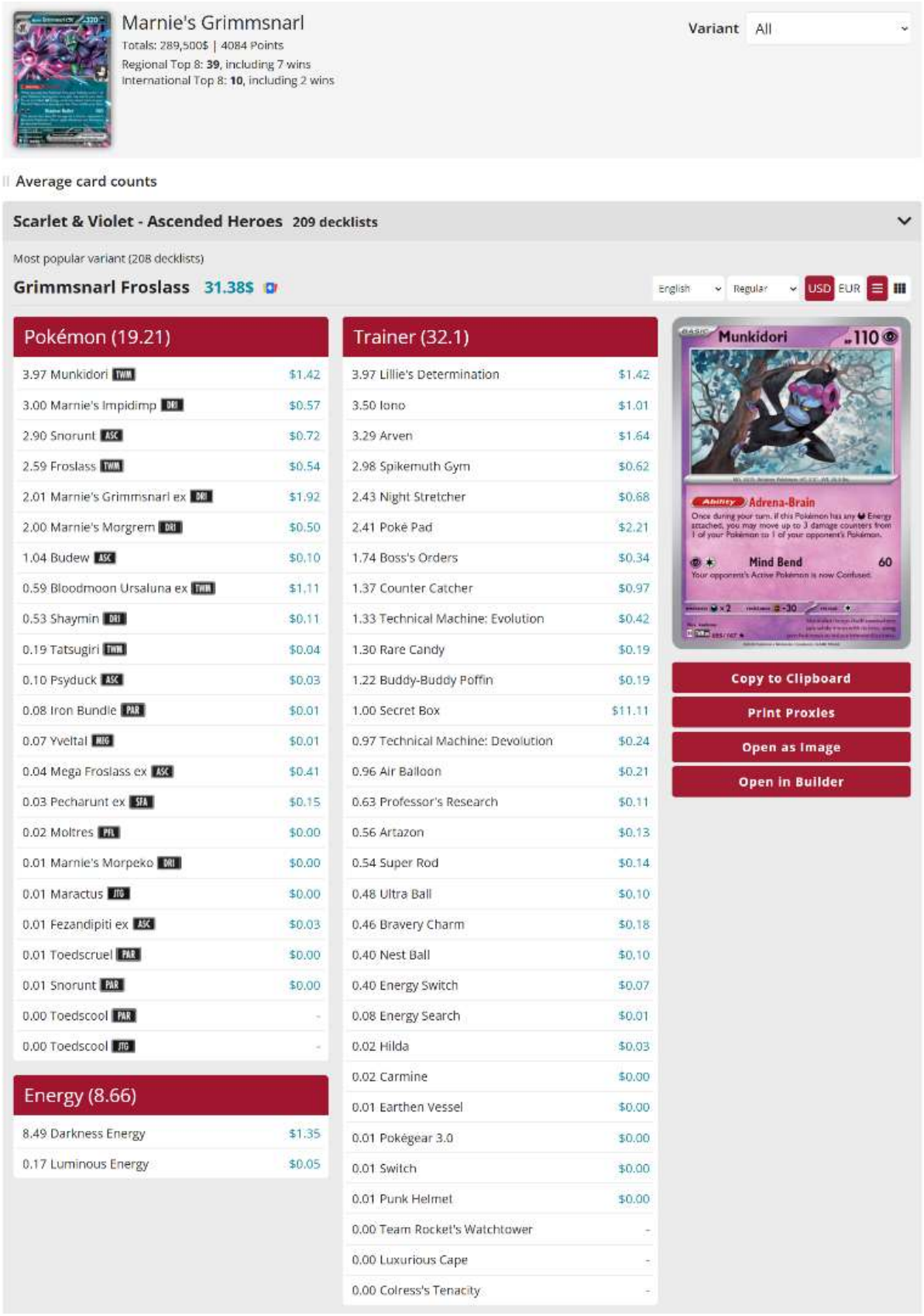

Figure 41: Second tournament deck (2nd place)

Figure 42: Evaluation of second tournament deck

The results of testing can be seen in the table below. The results may have a margin of error due to the card adjustments.

Tournament ranking	Score
1st	97/100
2nd	95/100

6.3 User Testing

User testing was conducted by advertising the project using a poster in local card shops. This will ensure anonymous, unbiased feedback from users who should be familiar with the Pokémon TCG.

Becca Walsh was commissioned to design the poster (see Figure 43). It included a QR code (see Figure 44) to the deployed application. This poster was displayed in Board & Brewed, Dun Laoghaire and Forbidden Planet, Dublin (see Figures 45 and 46).

Figure 43: DeckFix user testing poster by Becca Walsh

Figure 44: QR code to deployed application

Figure 45: Poster displayed in Board&Brewed

Figure 46: Poster displayed in Forbidden Planet

The application's homepage was updated to include a pop-up with user testing information and a link to the user testing form (see Appendix A) when the homepage was accessed (see Figure 47).

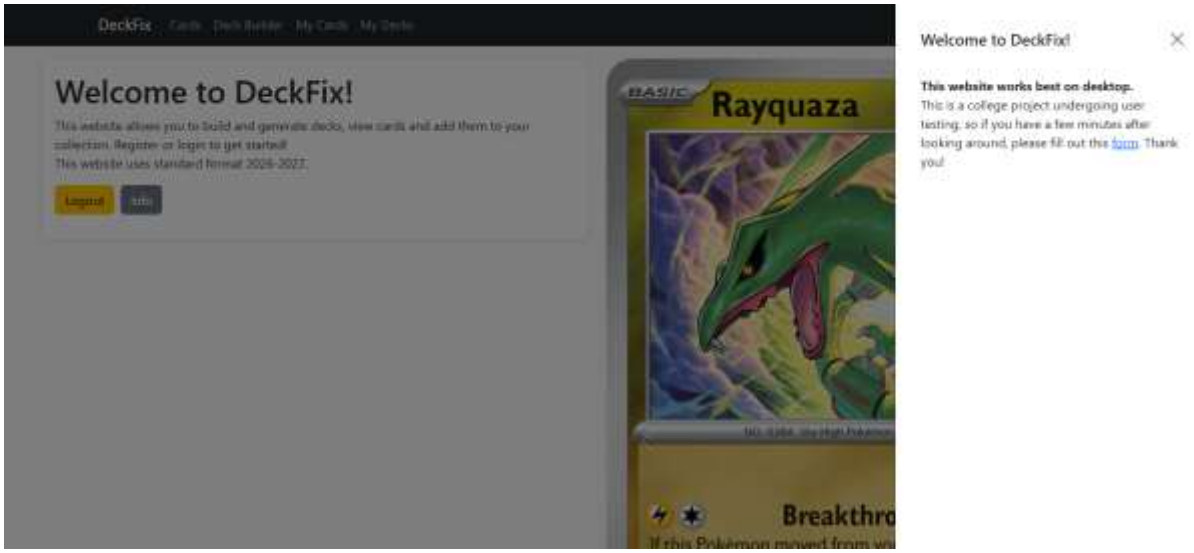

Figure 47: Homepage pop-up for user testing

6.3.1 Results

The form was divided into two sections: application quality and deck generation/evaluation quality. The results can be seen in Appendix A.

Summary of results as of 26th April:

- Ease of navigation received a score of 4.57/5, ease of understanding received 4.43 and the design of the website received 4.43/5.
- Some bugs were encountered.
- Some users found the usercards functionality confusing.
- Users praised the intuitive, user-friendly design.
- Accuracy of deck evaluation received a score of 3.8/5. The generated decks received a score of 3.67/5 in terms of playability.

6.3.2 Implementation of Results

The following feedback was implemented:

- Improvements to search bar user experience.
- Improvements to error handling: added more detailed toast notifications.
- Moved the homepage pop-up to the right side of the page.
- Fixed the “Total Cards” counter in the deck building tool to accurately update.
- Changed the deck building tool to open on the energy cards expansion to reduce confusion when adding energy cards.
- Improved generated item cards in generated decks as seen in the final iteration of deck generation in the Implementation section.

The following feedback was not implemented due to time constraints:

- Adding the option to change deck display (list view, 4 x 4, etc).

- Adding the ability to upload or share decks.
- Changing the button styling.
- Pinning the Your Deck/Generate Deck accordion to the screen in the deck building tool.
- Adding a guide for new players.

6.4 Evaluation

The following bugs remain in the final iteration of the application:

- When adding a card to a user's collection in the cards page, the page refreshes. The card is added successfully, but the page refresh affects user experience.
- When editing a deck in the deck building tool, if the card expansion dropdown is changed, the editing state is lost, and the deck can no longer be edited. This is due to the route handling for the deck building tool: the expansion parameter overwrites the deck ID parameter. A deck can still be edited successfully if the card expansion is not changed (all other filters and search can be used).

Overall, the application successfully achieved the aims and objectives of the project and fulfils the success criteria. Users can build, evaluate, generate and view decks, view, search and filter cards and save cards to their collection. The deck building and generating tools incorporate effective deck principles, game rules and user constraints. User authentication and data is handled effectively and securely.

7 Project Management

As mentioned in the Implementation section, the Scrum software development methodology was followed to complete this project. There was a total of 13 sprints that encompassed the total development and completion of the project, the details of which can be seen in the Implementation section.

GitHub was used to store code and track its development through commits. It was also used with Render to deploy the application, by retrieving the most recent GitHub application edition and publishing it.

Trello was used to track project development. Sprints were recorded as "lists", with each list containing the tasks for that sprint (see Figure 48).

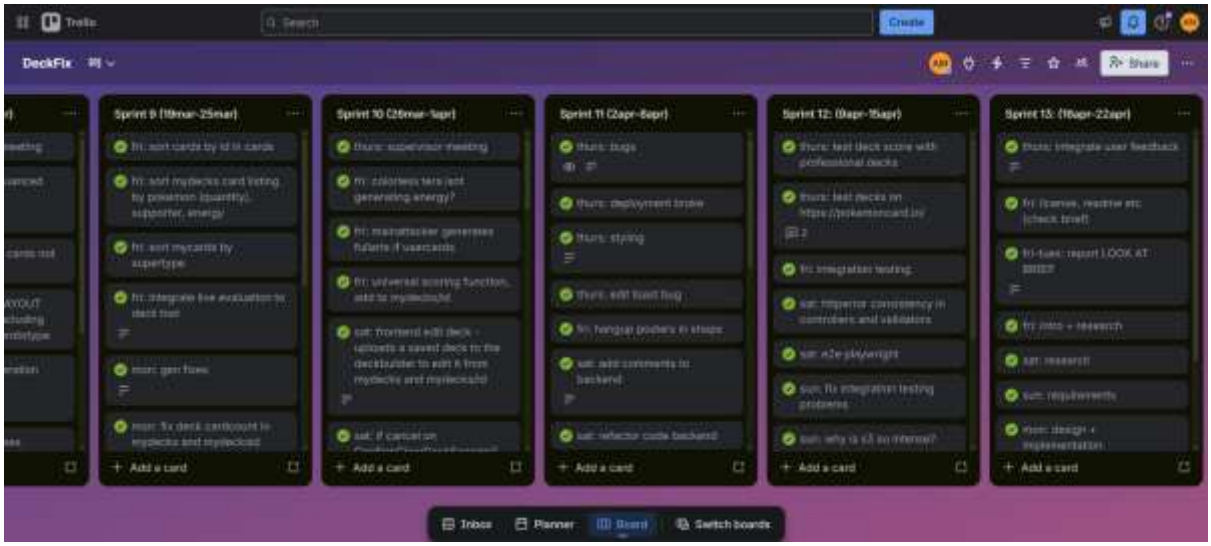

Figure 48: Project Trello board

Meetings with the project supervisor were conducted weekly on Wednesdays. This was an opportunity to discuss application bugs, questions and upcoming tasks. The details of these meetings can be seen in each sprint in the Implementation section.

8 Conclusion and Future Work

This project successfully answered its research question and achieved its aim by proving that constrained optimisation can be used to improve the evaluation and generation of Pokémon TCG decks to produce tournament-quality decks. It successfully achieved its aims and objectives and fulfils the success criteria. Users can build, evaluate, generate and view decks, view, search and filter cards and save cards to their collection. The deck building and generating tools incorporate effective deck principles, game rules and user constraints. User authentication and data is handled effectively and securely.

Limitations of the current application include:

- No tooltip with advice for deck evaluation in the deck building tool.
- No admin interface.
- Could not implement all user feedback.
- Two bugs remain.
- UserCards/mycards/saved cards should have been referred to as “favourites” in the application to reduce confusion.

The use of Trello, SCRUM methodology and regular meetings with the project supervisor ensured steady, consistent progress. Supervisor meetings were essential to overcoming problems and planning for upcoming sprints.

This project was a great opportunity for me to holistically develop my research, technical and project management skills. It is a great addition to my professional portfolio, and I am very proud of it. I feel better prepared for similar projects in the future, where I will apply the experience I gained from this project.

References

2026-27 Standard format (TCG). (2026, March 27). Retrieved from Bulbapedia-Bulbagarden: https://bulbapedia.bulbagarden.net/wiki/2026-27_Standard_format_(TCG)

5.x API. (n.d.). Retrieved from Express.js: https://expressjs.com/en/5x/api.html

About Play! Pokémon. (n.d.). Retrieved from Pokémon: https://www.pokemon.com/us/play-pokemon/about

Backes, A. (2021, February 22). *pokemon-tcg-sdk-javascript*. Retrieved from GitHub: https://github.com/PokemonTCG/pokemon-tcg-sdk-javascript

Backes, A. (2022). *Pokémon TCG API Documentation*. Retrieved from Pokémon TCG API: https://docs.pokemontcg.io

Backes, A. (2025, November 15). *pokemon-tcg-data*. Retrieved from GitHub: https://github.com/PokemonTCG/pokemon-tcg-data

Builder. (n.d.). Retrieved from Limitless TCG: https://my.limitlesstcg.com/builder

Coursera Staff. (2025, August 20). *Agile vs. Scrum: Which Should You Use, and Why?* Retrieved from coursera: https://www.coursera.org/in/articles/agile-vs-scrum

Deck builder. (n.d.). Retrieved from PokemonCard: https://pokemoncard.io/deckbuilder/

dotenv. (2026, April 17). Retrieved from GitHub: https://github.com/motdotla/dotenv

Fireship. (2022, May 16). *Solid in 100 Seconds*. Retrieved from YouTube: https://www.youtube.com/watch?v=hw3Bx5vxKl0

Fireship, B. (2023, June 30). *React VS Svelte... 10 Examples*. Retrieved from Youtube: https://www.youtube.com/watch?v=MnpuK0MK4yo

First steps. (2026). Retrieved from axios: https://axios.rest/pages/getting-started/first-steps

Functional and Non Functional Requirements. (2026, April 11). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/software-engineering/functional-vs-non-functional-requirements/

Getting Started. (n.d.). Retrieved from Vinxi: https://vinxi.vercel.app/guide/getting-started.html

Getting Started. (n.d.). Retrieved from Jest: https://jestjs.io/docs/getting-started

Getting Started. (n.d.). Retrieved from mongoose: https://mongoosejs.com/docs/

helmet. (2026, April 1). Retrieved from GitHub: https://github.com/helmetjs/helmet

IADT Student Handbook 2024-2025. (n.d.). Retrieved from IADT: https://iadt.ie/wp-content/uploads/2024/09/IADT-Student-Handbook-2024-2025.pdf

Installation. (2026). Retrieved from Playwright: https://playwright.dev/docs/intro

Integration Testing. (2026, April 14). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/software-testing/software-engineering-integration-testing/

Introduction. (n.d.). Retrieved from Kobalte: https://kobalte.dev/docs/core/overview/introduction/

JustInBasil. (2024). *A Basic Guide to Deck Building in the Standard Format*. Retrieved from justinbasil: https://www.justinbasil.com/guide

JustInBasil. (n.d.). *Limits - What You Can and Can't Put into Your Deck*. Retrieved from justinbasil: https://www.justinbasil.com/guide/limits

Kowalski, J., & Miernik, R. (2020). *Evolutionary Approach to Collectible Card Game Arena Deckbuilding using Active Genes*. Wrocław: University of Wrocław.

Luu, Q. T. (2024, March 18). *What Is Constrained Optimization?* Retrieved from Baeldung: https://www.baeldung.com/cs/constrained-optimization

McAndrew, M. (2025, December 8). *ajs-examples/notes-full-stack*. Retrieved from GitHub: https://github.com/mcandru/ajs-examples/tree/edec50e4d177d63ca0dd8416d4f8326a0bb5e67a/notes-full-stack

McAndrew, M. (2025, November 4). *mcandru/ajs-examples/notes-api-typescript*. Retrieved from GitHub: https://github.com/mcandru/ajs-examples/tree/master/notes-api-typescript

MINUTES, J. |. (2025, May 21). *We built 2 apps: Svelte 5 vs Solid.js - here's the shocking winner!* Retrieved from YouTube: https://www.youtube.com/watch?v=2tjU3HMDx70

MoSCoW Prioritization. (n.d.). Retrieved from ProductPlan: https://www.productplan.com/glossary/moscow-prioritization/

node.bcrypt.js. (2026, April 14). Retrieved from GitHub: https://github.com/kelektiv/node.bcrypt.js#readme

Node.js v25.9.0 documentation. (2026, April 1). Retrieved from Node.js: https://nodejs.org/docs/latest/api/

Overview. (n.d.). Retrieved from Solid-Bootstrap: https://solid-libs.github.io/solid-bootstrap/getting-started

Overview. (n.d.). Retrieved from Solid Router: https://docs.solidjs.com/solid-router

Overview. (2026). Retrieved from Svelte: https://svelte.dev/docs/svelte/overview

Pattanayak, L. N. (2024, April 11). *Reasons to Choose TypeScript Over JavaScript*. Retrieved from DEV: https://dev.to/laxminarayana31/reasons-to-choose-typescript-over-javascript-16nd

Peterson, D., O'Connor, Q., & Hay, R. (2025, February 12). *Pokemon TCG: 11 Best Tips For Making An Awesome Deck*. Retrieved from TheGamer: https://www.thegamer.com/pokemon-tcg-best-tips-making-awesome-deck/

Postman Docs. (2026). Retrieved from Postman: https://learning.postman.com/

Quick Start. (n.d.). Retrieved from React: https://react.dev/learn

Quick start. (2026, January 26). Retrieved from SolidJS: https://docs.solidjs.com/quick-start

snnsnn. (2025, October 9). *solid-courses/solidjs-the-complete-guide*. Retrieved from GutHub: https://github.com/solid-courses/solidjs-the-complete-guide

Software Architectural Patterns in System Design. (2025, July 23). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/system-design/design-patterns-architecture/#layered-architecture-ntier-architecture

solid-toast. (2024, February 12). Retrieved from GitHub: https://github.com/ardeora/solid-toast

Soriano, M. (2022, July 4). *Let's build a Shopping Cart website with SolidJS and Bootstrap – Part 2*. Retrieved from Michael Soriano: https://michaelsoriano.com/lets-build-a-shopping-cart-website-with-solidjs-and-bootstrap-part-2/

supertest. (2026, January 6). Retrieved from GitHub: https://github.com/forwardemail/supertest

Tieber, R., & Felfernig, A. (2021). A Knowledge-based Configurator for Building Magic: The Gathering Card Decks. *Proceedings of the 42nd International Conference on Conceptual Modeling Workshops.* CEUR-WS.org.

Tournament Deck Lists. (2025, November 29). Retrieved from Limitless TCG: https://limitlesstcg.com/decks/lists?time=3months&type=all&format=standard&region=all&division=all

TPCi Editing Staff & Sucevich, Kyle. (2025, November). *Pokémon Trading Card Game Rulebook*. Retrieved from Pokémon: https://www.pokemon.com/static-assets/content-assets/cms2/pdf/trading-card-game/rulebook/pfl_rulebook_en.pdf

UML Class Diagram. (2026, January 21). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/system-design/unified-modeling-language-uml-class-diagrams/

Unit Testing. (2026, April 13). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/software-testing/unit-testing-software-testing/

Welcome to AWS Documentation. (2026). Retrieved from AWS: https://docs.aws.amazon.com/

Welcome to the MongoDB Docs. (2026). Retrieved from MongoDB: https://www.mongodb.com/docs/

What Does TCG Mean? Everything You Need to Know. (2025, March 3). Retrieved from QP Market Network: https://www.qpmarketnetwork.com/trading-card-game/what-does-tcg-mean-everything-you-need-to-know/

What is End to End Testing? (2025, June 26). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/software-testing/what-is-end-to-end-testing/

Appendices

A External Links

The technical feasibility study can be found in the "research" folder of the final submission, and here: https://github.com/angelinamorris2003/CCY4-MajorProject-DeckAlgorithm

The project's Miro board can be found here: https://miro.com/app/board/uXjVJAIRAwA=/?share_link_id=650117691975

The project's Trello board can be found here: https://trello.com/invite/b/697661cda554855f37bfbbec/ATTIe17090538347cf4e30fe26e2160ab9e85CDCD6B0/deckfix

The user testing form can be found here: https://docs.google.com/forms/d/e/1FAIpQLScR1lLeS9rdCQfUJiA8ImhnlHnmce7FDLr2yf7XH8fdIT4UnQ/viewform?usp=sharing&ouid=100292538368355879467

The results of the user testing can be found here: https://docs.google.com/spreadsheets/d/1Nakk6XWJIgpsh-sBFNEidpSG7sWLI9KZ_o4_JMcEuAo/edit?usp=sharing

B The Pokémon Trading Card Game

Pokémon are fictional creatures that are used by players for battle against other players' Pokémon, to win prizes. They first debuted in Pokémon Red and Blue; videogames made for the Nintendo Game Boy console in 1996. A Trading Card Game (TCG) was made based on this the same year. A TCG is a game in which you collect cards to use in decks against other players: these cards are often traded between players, hence the name (What Does TCG Mean? Everything You Need to Know, 2025). The cards have different attacks, rules and weaknesses that influence different strategies.

B.1 The Cards

There are three different types of cards that exist in the TCG.

- **Pokémon** (the Pokémon creature itself): The cards that performs attacks and takes damage.
- **Trainer:** The cards that give the player a benefit without attacking the other player, such as allowing that player to search their deck for a card.
- **Energy:** The fuel for a Pokémon's attack. Each attack has an energy cost that must be fulfilled to perform the attack.

B.1.2 The Pokémon Card

Figure 49 shows a Pokémon card. Its health points (how much damage it can take before it is knocked out and discarded) is in the top right corner. Its attacks are listed underneath the illustration. The attack has an energy cost (the symbol on the left; the amount of symbols correlates to the amount of energy cards required), damage points (the number on the right; how much is taken from the opponent's Pokémon's health) and instructions (additional text; must be followed when performing that attack). Figure 50 shows another Pokémon card with an ability label, which is a further set of instructions for that card that offer the player passive benefits (such as drawing a card).

Figure 49: Pokémon card

Figure 50: Pokémon card with an Ability

B.1.2.1 Card Energy Type

In Figures 49 and 50, there is a symbol next to the Pokémon's health points, which can also be found in the attack's energy cost. This symbol is that card's energy type, in this case, Water. There are ten different types a Pokémon can be: Grass, Water, Fire, Lightning, Psychic, Fighting, Darkness, Steel, Dragon and Colourless. A card's type dictates what type of energy card it requires, and what type that card is weak to (found at the bottom of the card, labelled "weakness"). Some cards have attacks that require different types of energy cards, as seen in Figure 50. The only exception to this is Colourless energy (seen in the second attack in Figure 49) which means any type of energy can be used to fulfil the energy cost. This means, that if a player wishes to perform the Icicle Missile attack, they can attach two Water energy cards, or one Water energy card and an energy card of another type. But there must be two energy cards attached, and one of them must be of Water type.

B.1.2.2 Evolution and Basic Pokémon

Certain Pokémon can evolve. Most Pokémon belong to an evolution family, meaning they are either the first, second or third form of a species. Pokémon cannot evolve into another evolution family's form. The first Pokémon form of an evolution line is referred to as a Basic Pokémon. When a Pokémon evolves, it becomes stronger. In Figure 51,

Pokémon evolution in the TCG is shown. The first Pokémon has a “Basic” label in the top left corner, and the second Pokémon has the previous Pokémon shown in the same place, with “Stage 1” replacing the “Basic” label. This is an important aspect of the TCG as the different forms of a Pokémon have different health and attacks which makes the gameplay more complex.

Figure 51: Pokémon evolution

B.1.2.3 Card Rarity

Cards come in different rarities, which correlate to how powerful the cards are in the game. For example, a Pokémon EX card is rarer than a common card, boasts higher health points and stronger attacks in exchange for offering two prize cards instead of one. The difference in strength between an EX and standard Pokémon card can be observed in Figures 52 and 53: it is the same Pokémon, but the EX suffix means it is

stronger.

Figure 52: Pokémon card with EX suffix

Figure 53: Pokémon card without EX suffix

Cards can also be of “full-art” rarity: meaning they are functionally the same as their non full-art counterparts, but visually different. Figure 54 shows a normal Pokémon card, and Figure 55 shows the full-art variant of that card. The full-art variant has the same HP and attacks, but a different image.

Figure 54: Regular Pokémon card

Figure 55: Full-art variant

B.1.2.4 Card Subtype

A Pokémon card can be of specific subtype, such as Mega or Tera. This can change the mechanics of the card by changing the prize card award upon knockout (see Game rules), or the attack's energy costs. A Mega card is shown in Figure 56, and a Tera card is shown in Figure 57.

Figure 56: Mega Pokémon card

Figure 57: Tera Pokémon card

B.2 Rules

The official rules of the TCG are outlined by the Pokémon company in the rulebook (TPCi Editing Staff & Sucevich, Kyle, 2025). This covers the game rules and deck rules in detail.

B.2.1 Game Rules

A game starts with each player shuffling their deck and removing six random cards, which will be used as prize cards. A coin is flipped to determine who goes first. Players draw seven cards and place Basic Pokémon on their bench. If no Basic Pokémon are drawn, a player can redraw, allowing the opponent to draw an extra card. Players take turns drawing cards, attaching energy to Pokémon, evolving their Pokémon, using trainer cards, and attacking. A Pokémon can only evolve once per turn and cannot evolve on the first turn. A game is won when a player has taken six of their opponent's prize cards, when the opponent has no cards left to draw, or when the opponent has no Pokémon left to play.

B.2.2 Deck Rules

(JustInBasil, Limits - What You Can and Can't Put into Your Deck, n.d.) retrieves information from the official rulebook and explains deck rules in greater detail. A deck must have at least one Basic Pokémon. A deck must always contain 60 cards. No more than four copies of a card with the same name can exist within a deck, except for basic energy cards and ACE-SPEC cards. For example:

- Four Mew, two Umbreon and fifteen Basic Psychic Energy = Legal
- Four Morpeko, four Marnie's Morpeko = Legal (the "Marnie's" prefix changes the card's name)
- Five Appletun = Illegal

B.3 Standard Format

Play! Pokémon (About Play! Pokémon, n.d.) is the official Pokémon TCG organised play program that hosts local and international tournaments. The standard format is the selection of current cards that are legal for play in Play! Pokémon, which is updated yearly (2026-27 Standard format (TCG), 2026). There are also the unlimited format and expanded format, but these are far less popular. The Pokémon company has been releasing playable cards since 1996, so cards are rotated in and out of the standard format, to get rid of old expansions and keep the cards relevant in tournaments. This project will implement the standard format, as it is the most popular and relevant format.