Card Collector Application

Design, Validation, and Verification Document

Version 0
Group 9
COMPSCI 4ZP6B
Dr. Mehdi Moradi
January 23rd, 2026

Tania Da Silva	dasilt2@mcmaster.ca	400314513
Norman Liang	liangl20@mcmaster.ca	400377856
Elite Lu	lue13@mcmaster.ca	400364692
Ishpreet Nagi	nagii@mcmaster.ca	400394382
James Nickoli	nickolij@mcmaster.ca	400399883
Kenneth Ong	salimk4@mcmaster.ca	400395100
Geon Youn	young2@mcmaster.ca	400362097

Table of Contents

0. Document Information

0.1 Contribution History

Authors	Sections
Tânia Da Silva	UI Design/Figma
Norman Liang	Database/API stuff
Elite Lu	UI Design/Figma
Ishpreet Nagi	UI Design/Figma
James Nickoli	Card Search & Identifier
Kenneth Ong	Authentication Server
Geon Youn	Card Search & Identifier

0.2 Revision History

Version	Authors	Description	Date
0	Everyone	Initial document	Jan. 23, 2026

0.3 Glossary

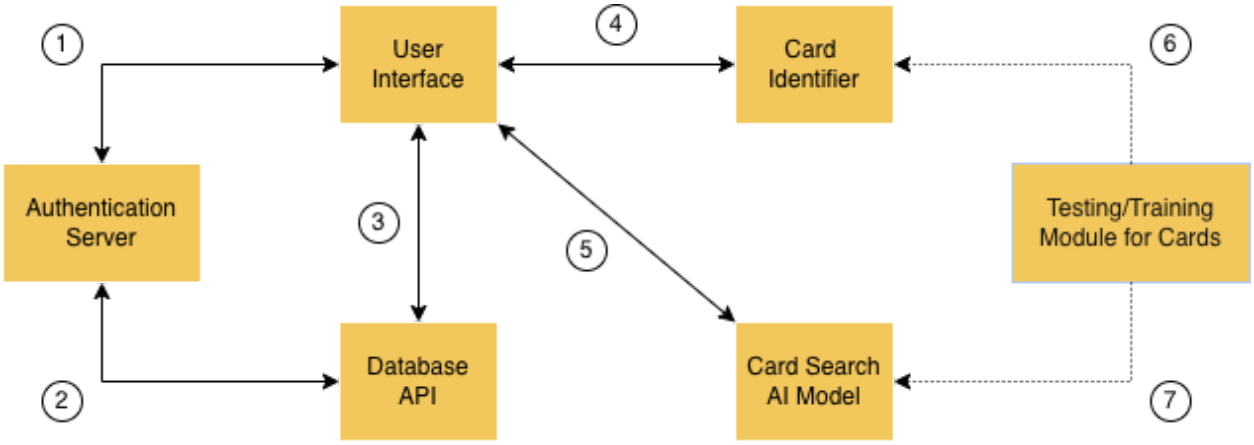
TCG	TCG is an abbreviation for “Trading Card Game”, which is a set of cards that are typically used in a turn-based strategy game where players build custom decks of cards to compete against each other. These cards are also often traded amongst players.
(Card) Collection	A group/collection of a set of TCG cards.
Pokémon Set	A collection of unique Pokémon cards released together, forming a specific set list, theme, and release date.
Raw/Graded	Cards can get sent to grading companies that will typically rank a card’s quality from 0 to 10. A raw card is a card that hasn’t been graded yet.

1. Purpose Statement

The purpose of our card collection and management system is to create a seamless way for Pokémon TCG users to manage their collection and to engage with other users by trading their cards. In this document, we will describe all the components of our system in detail and provide a full plan for testing to ensure our system meets the requirements and metrics from our Software Requirements Specification document.

2. Component Diagrams

Component diagram: Components and their relationships. If any component is complex, consider a secondary chart for more details for that component. Example for an AI based model with a front end:



1. The UI sends user information to the authentication server, then the authentication server responds with the appropriate data or message. If the user is authenticated, the response includes a session.
2. The authentication server queries the database API to do credential checks and user information changes. The database API will reply with an error message if an error has occurred or a success message if it was successful.
3. The UI calls the database API to access various details from the database such as cards (card details, various Pokémon sets, Pokémon cards organized by Pokémon, etc.) and other users. The database API processes the requests from the UI and responds with the appropriate data or message.
4. The UI sends the card identifier a stream of images. The card identifier parses the image details and uses the hash to properly identify the card and returns the information to the user.
5. The UI sends a natural language description of a given card to the Card Search AI Model. Using the text information, the Card Search AI Model will determine card(s) that fit the description and return it to the UI.

6. The Testing/Training Module will use an external API to obtain the card images. Then, the module will compute the hashes of the cards and publish the algorithm to the Card Identifier Component periodically.
7. The Testing/Training Module will use an external API to obtain the card information. Then, the module will use it to train a text-based AI model for natural language search and publish the model periodically.

3. Component-Requirement Relationships

Relationship between components and requirements: Describe what requirement each component meets. This might not be a 1 to 1 relationship.

System Component	Requirements Covered
UI	**P0:** ● Basic web and mobile GUI: ○ A way for users to add cards to their collection. ■ Automatically through their camera or an uploaded image. ■ Manually by searching through the Pokémon TCG database. ○ A way for users to view their added cards. ○ A way for users to search through their collection. ○ A way for users to delete cards from their collection. **P1:** ● A way to select the card's condition (raw condition or graded) and type (ex. holo/reverse holo) and enter additional notes. ● Users should be able to create a public profile that can be viewed by other users who have an account. ● Users should be able to add their contact information to their profile. ● Users should be able to mark certain items in their collection to be available for trade. ● Users should be able to rate each other after completing a trade.
Card Identifier	**P0:** ● Identify and classify a Pokémon card from an image.

	○ An object detection model to identify Pokémon card locations in an image. ■ Users should be able to provide the image either (1) manually or (2) in real-time through their camera (option 2 is preferred). ○ A way to perform this task locally on the user's device. **P2:** ● The system should allow scanning for multiple cards at a time when the user is trying to add a card to their collection.
Testing/Training Module for Cards	**P0:** ● Identify and classify a Pokémon card from an image **P3:** ● Expand the computer vision model to other collectible items such as K-pop albums, photocards, comics, coins, stamps, etc., and other language Pokémon cards.
Card Search AI	**P0:** ● Identify and classify a Pokémon card from an image. ○ An object detection model to identify Pokémon card locations in an image. ■ Users should be able to provide the image either (1) manually or (2) in real-time through their camera (option 2 is preferred). ○ A perceptual hashing algorithm to classify English Pokémon cards from the identified Pokémon cards by the object detection model. ○ A way to perform this task locally on the user's device. ● A way to manually identify a card if the system can't classify it.
Database API	**P0:**

	• A way for users to search through the Pokémon TCG database either directly, by describing their card with natural language, or by set.

Authentication Server	**P0:**
	• Support user-specific actions such as:
	  ○ A way for users to add cards to their collection.
	  ○ A way for users to view their added cards.
	  ○ A way for users to search through their collection.
	P1:
	• Users should be able to create a public profile that can be viewed by other users who have an account.
	• Users should be able to add their contact information to their profile.
	• The system should allow users to register for an account
	• The system should have a mechanism that can deal with logging duplicate entries.
	• The system should allow users to share their collections between each other.

4. Component Details

For each component, describe:
a. Normal behavior: What is the expected behavior from each component.
b. API (input/outputs with emphasis on data structures being exchanged between components).
c. Implementation (Include details: classes, methods, and their relationships).
d. Potential undesired behaviours (this would be the place to think what can go wrong).

4.1 UI

4.1.1 Normal Behaviour

The normal behaviour of the application UI will consist of all the users' cards being displayed correctly and according to their specific collection, their entered user information will be correctly displayed, and all edits to the information will be correctly updated on the application promptly. Furthermore, all interactable elements, such as buttons and menus, will complete their intended function. For an

in-depth breakdown of each page and its expected behaviour, please refer to Table 4A in the appendix.

4.1.2 API

The UI does not inherently take in any inputs. Many of the given inputs and outputs by users are considered in the Card Identifier. All information directly relating to the user, including their name, location, and social media links, while being accessed and displayed by the UI, is classified and considered in the database.

4.1.3 Implementation

The implementation is completed using Next.js and ChakraUI. The code is written with TypeScript, with certain icons being imported from React Icons. All the components and pages will be listed below. Types will be omitted.

Pages

For an in-depth breakdown of each page and the components it uses, please refer to Table 4B in the appendix.

Components

For an in-depth breakdown of each component, its input variables, and its details, please refer to Table 4C in the appendix.

4.1.4 Potential Undesired Behaviours

Page Name	**Potential Undesired Behaviour**
pokemon-grid	Since all the images are currently being loaded in right now, this could result in images not showing up if the device can not handle 1025 images simultaneously loading in.
filter-cards	Some filters may override each other, and as a result, this could mean cards that should be shown are shown, and cards that are not supposed to be shown are not.
user-profile | trade | wish	The user can have too many cards that are being marked for trade or wish, and if the device cannot handle this many images loading in simultaneously, it can result in images not appearing.

4.2 Card Identifier

4.2.1 Normal Behaviour

The card identifier is responsible for finding potential matching Pokémon cards in a given image. This component is split into two parts: first, identifying a card-like figure in an image and then classifying the card-like figure as a Pokémon card.

4.2.2 API

The API for the card identifier includes functions for taking in a camera feed and classifying an input image as a Pokémon card.

Specifically, the former function would take in a camera feed (series of images) and output a cropped section of a frame containing the detected card-like figure. The latter function would take in the former function's output and return the card ID of the closest matching card.

4.2.3 Implementation

Our current implementation is to detect bounding boxes for card-like figures in the camera feed and find matching cards through perceptual hashing. We compare the hash against hashes that were precomputed for all cards. We find the hamming distance, sort them in ascending order, and return the card ID for the k smallest distances.

4.2.4 Potential Undesired Behaviours

We might not be able to correctly identify a card in the image or we're unable to classify the correct card. There's many variations of the same card, where the only difference may be the copyright year, which may seem small but can have a huge impact on its value (e.g. a 1999 Charizard is worth much more than a 2016 Charizard).

4.3 Testing/Training Module for Cards

4.3.1 Normal Behaviour

Quantitative testing for card identification should take in a series of images that are labelled either with the set ID and the number of the card present, or labelled as "NoCard" if there is no card in the image. The testing should run each image through the current identification stack and produce metrics based on the correctness of the identification. Training the card identification should involve specifying a set of images and associated metadata, and producing a separate identification model for each category of images (Pokémon, One Piece, K-pop, ect).

4.3.2 API

The API for testing involves inputting a series of labelled images, and the output is numbers representing metrics like accuracy and F1 score. The API for training involves inputting a folder of images and a table in our database to pull image metadata from, and the output is a model that can be used by our identification code to identify cards within the input folder.

4.3.3 Implementation

To test the identification, we use a component in our front-end that uses the browser's file dialogue to take in images, displays the source images and found images along with IDs, and displays the metrics as text. The training should be able to run as a process on our Vercel server, or as a Python script locally.

4.3.4 Potential Undesired Behaviours

We don't want other computations to interfere with the speed benchmarking of our testing. We don't want training to produce worse models as the number of images within a category increases.

4.4 Card Search AI

4.4.1 Normal Behaviour

The card search AI allows users to find cards through natural language by describing its visual details.

4.4.2 API

The API for the card search AI involves taking in a string containing the description for the card. The output will be the card IDs that correspond to that description.

4.4.3 Implementation

We generate captions for each card and create embeddings. Then, when the user sends a query, we create an embedding for that query and return the top k matching cards based on cosine similarity.

4.4.4 Potential Undesired Behaviours

The generated caption may not include all the visual details of the card, making it difficult for that card to be found depending on the user's query.

4.5 Database API

4.5.1 Normal Behaviour

The database API is responsible for maintaining a record of registered users on the platform, including their information (such as biography, associated social media links, and cards collected). The application will interact with the database to

retrieve information, but will also be able to modify its contents, such as when a new user registers for the service, they will be added to the database. Or if a user adds a new card to their collection, the database will be updated to reflect the added card. Furthermore, the database API acts as a bridge for the database and frontend, taking instructions from the frontend in the form of API calls to perform operations such as database queries, data manipulation, and other backend/server-side logic.

4.5.2 API

The database API provides functions for creating, reading, updating and deleting data in the database. For example, when a card is to be added to a user's collection on the application, the database API's create function is called. Prisma sends this request as an SQL command to Supabase, which will allow the card to be properly added to the respective user's collection.

4.5.3 Implementation

The database API is implemented using a combination of Next.js serverless functions, Prisma, and Supabase. Prisma serves as an ORM and defines the schema of the database (mainly for storing user and card information), while Supabase hosts the PostgreSQL database. Supabase will also manage the user logins; Supabase Auth provides an autogenerated user table for user authentication, which is linked to our own User table using an SQL trigger on Supabase. The database schema definition is written in Prisma's own Prisma Schema Language.

4.5.4 Potential Undesired Behaviours

Since the user profile information is stored in our database, and each profile is created upon account creation via Supabase Auth, should the SQL triggers fail, a user could register without having an assigned profile created on the database.

4.6 Authentication Server

4.6.1 Normal Behaviour

The authentication server is the foundation of the user management workflows in the application. It is primarily responsible for registering and authenticating new or existing users, which will also provide them with a valid session. Additionally, it will also perform read and write operations to the database to pull, add, or modify user-related information such as username, password, and email. This component also ensures that users are logged into the correct account and that the application displays information only relevant to the authenticated user.

4.6.2 API

Values from the sign-in or sign-up forms are checked against the database or stored into the database. Only when the credentials match, the server will return a session that can be used by the user to access protected routes and pages.

4.6.3 Implementation

The implementation of the authentication server is encapsulated in the React Context Provider called AuthContextProvider. In this context provider, there are methods that handle operations such as getting the session, signing up, signing in, and signing out. The Supabase Auth library is used to handle the session management and some of the interactions with the database, which is also hosted in Supabase.

4.6.4 Potential Undesired Behaviours

Since our implementation relies on the Supabase Auth library, any bugs, vulnerabilities, or breaking changes that the Supabase team might put out in the future may also affect our application.

5. User Interface Details

The frontend design is a multi-page application that allows the user to add cards to their collection, interact with other users, scan and store their cards, and various other activities. There will be a login page for the user. Upon logging in, the user is able to interact with their own collection and the listed various activities.

The font that is used is Lexend Deca for its ease of reading for the user. The primary colour scheme uses #F2C75C (marigold colour) as a main accent colour and #003B49 (turquoise “turtoise”) for the text on the accent colour. Normal text will be black (#000000) and the background of the application is a grey (#F2F2F2). The main accent colour and the accent text colour has a contrast ratio of 7.61, which allows the text to be easily legible.

The Figma will be linked here, which has the notes for all interactions and UI design.

6. Component Test Plan (Validation & Verification)

6.1 UI

6.1.1 Unit Test

We are using Jest and React Testing Library to perform unit tests for all of our pages and UI components in our application. For the purposes of testing the UI, we will be writing assertions on user interactions (clicks, swipes) and what the user sees on the screen. The data will be mocked so that unit tests are resource efficient and very quick to run.

6.1.2 Performance Tests and Metrics

It is widely known that page load times should not exceed 2-3 seconds for a desirable user experience and a better performance in Search Engine Optimization (SEO). Thus, using the built in tools provided by Vercel and the analytics hook useReportWebVitals provided by Next.js, we plan to continuously monitor and ensure that this page load time target is met. Furthermore, we also plan to have around 70% unit test coverage for all of the pages and UI components.

6.2 Card Identifier & Card Search AI

6.2.1 Unit Tests

Specific unit tests do not provide valuable insight for the identification model, so we aim to be able to test large numbers of images to cover as many edge cases as possible. Same for the search model, except we will create and use a test set of card descriptions for a number of cards.

6.2.2 Performance Tests and Metrics

We run the identification/search code on each image/description and produce the following metrics: average speed, accuracy, precision, recall, and F1 score. If we produce a model that uses a confidence value, we will also include an ROC curve as a metric.

6.3 Testing/Training Module for Cards

6.3.1 Unit Tests

Since the testing is a front-end component, we can use the same Jest and React libraries to test the behaviours of the testing component.

6.3.2 Performance Tests and Metrics

We aim to have the functionality of the testing outside of running the identification code have minimal impact on the identification speed.

6.4 Database API

6.4.1 Unit Tests

Similar to the unit tests for the UI components, Jest and React Testing Library will be used to write the unit tests for the Database API. Each API route/endpoint should have its own test suite, as it represents a single serverless function once it is deployed in Vercel. For each of those test suites, assertions on the correctness of the backend/server-side logic are performed. Since the database queries are done using Prisma, we will also perform assertions on the correctness of the queries, and Prisma has made it easy to mock the database so that we are able to easily isolate the unit test from our actual database.

6.4.2 Performance Tests and Metrics

We will be using Supabase's dashboard and monitoring tools to identify long-running queries that might be present throughout the application. As we aim for under 2-3 seconds of page load time, this would also mean that there should be no database query that runs for longer than 1-2 seconds, unless they are cached or some batching or background loading mechanisms have been implemented.

6.5 Authentication Server

6.5.1 Unit Tests

Once again, Jest and React Testing Library will be used here to test the functionality of the context provider and hook that supports the authentication server. These unit tests will perform assertions on whether the implementation calls the correct API endpoints with the correct arguments. The unit tests will mock the Supabase Auth library and any database API calls since they should have already been tested in a different test suite.

Appendix

Table 4A: Normal Behaviour of each page

Page Name	Expected Behaviour
pokemon-grid	All 1025 Pokémon are able to be displayed in a grid on the page. The user is able to click on any of them to display the cards of that corresponding Pokémon.
filter-cards	The user is able to properly filter by set or Pokémon (more filters to be added), along with various other filters. After filtering, the user should be able to click/tap on the filtered items to see the information.
user-profile	This is the version of the user profile page you would see when you view someone else's user profile on the app that is not your own. Being able to view all of the user's information they have entered, including their name, rating on the app, location, and social links, as well as the cards they would like to trade and have wishlisted. The profile will only show 3 cards on the main user profile page for the trade and wish lists; however, there will be buttons that can be pressed to go to the trade and wish list pages to view all of them. It will also showcase the maximum three cards the user will have marked for 'Showcase'. From this page, the user can also click on a menu to choose options to block or report users.
user-profile | trade	This is the trade page for the version of the user profile page you would see when you view someone else's user profile. This page displays all the cards the user has marked for trading in an organized 3-card-per-row grid.
user-profile | wish	This is the wish list page for the version of the user profile page you would see when you view someone else's user profile. This page displays all the cards the user has marked that they wish for in an organized 3 cards per row grid.
personal-profile	This is the version of the user profile page you would see when you view your own user profile on the app. It shows a good view of what others viewing your profile will see. The profile will only display 3 cards on the main user profile

	page for both the trade and wish lists; however, buttons will be available to navigate to the trade and wish list pages to view all of them. It will also showcase the maximum three cards the user will have marked for 'Showcase'. Replacing the menu, hiding the block and report button, there will be an 'Edit Profile' button that will take the user to the page where they can edit the contents of their user profile.
personal-profile | edit-profile	This is a page that allows the user with the option to edit their profile and change the contents that are displayed. The user can edit their name, bio, location, social media handles, 'Showcase' cards, 'Wish List' cards, profile visibility (selecting between Public and Private), and have the options to sign out of their account or delete it.
personal-profile | edit-profile | wishlist	This page displays all the cards the user has marked as they wish for on their profile in a 3-cards-a-row grid. The first card slot in the first row is not a card but a plus button that the user can select to go to the card search page.
personal-profile | edit-profile | wishlist | search	The page where users can search our database to find the item they want to add to their wishlist. They can do so by either manually typing its name to find it or by going through our categories.
editCard	On this page, the user should be able to edit the features of any card they are adding to their collection or any card already in their collection. The user can edit the card name, set, grade, condition, holofoil, and tags. Additionally, the user can add the card to their trade list or highlighted list from this screen.
logIn	On this screen the user should be able to enter their username or email and password to log in to their collection database. If the user does not have an account they should be able to navigate to the sign up page from here. If the user does not remember their password they should be able to navigate to the forgot password page from here.
signUp	On this screen a user should be able to enter their email, chosen username, and chosen password to create an

	account. If the user already has an account they should be able to navigate to the log in page from here.
forgotPassword	On this page the user should be able to enter their email to receive a link to change their password. They will then be able to enter their new password after they will be navigated to the sign in page.
TradePost	On this page, users should be able to view other people's cards that they are willing to trade and the cards the user has that the other users want. In addition to this, users should be able to view the star rating and the other user's (star rating is the reliability).

Table 4B: Page and Components Used

Page Name	**Components Used**
pokemon-grid	PokemonPolaroid, PokemonSet
filter-cards	PokemonCardMini
user-profile	| user-profile | SocialLinks | Showcase | TradeList | WishList | AccountOptions
user-profile | trade	PokemonCardMini
user-profile | wish	PokemonCardMini
personal-profile	| user-profile | SocialLinks | Showcase | TradeList | WishList
personal-profile | edit-profile	| edit-profile | Showcase | DeleteAccount

personal-profile | edit-profile | wishlist	PokemonCardMini
personal-profile | edit-profile | wishlist | search	PokemonCardMini
editCard	PokemonCardMini
logIn	AuthForm
signUp	RegisterForm
forgotPassword	ResetPasswordForm
TradePost	PokemonCardMini, StarRating, TradePostUser

Note. All pages will have their main code in page.tsx in the specified folder, with a further breakdown of pages not named page.tsx in an organized file folder format.

Table 4C: Component Breakdown

Component Name	**Input Variables**	**Information**
PokemonCardMini	cardName (String) image (String) cardId (String)	Displays an individual Pokémon card.
PokemonPolaroid	id (number) masterSet (number) grandmasterSet (number)	Displays an individual Pokémon along with the completion status of the master set and grandmaster set.
PokemonSet	label (string) image (string) setId (string) masterSet (number) grandmasterSet (number)	Displays an individual Pokémon set along with the completion status of the master set and grandmaster set.
SocialLinks	socials • icon	Displays the social media handles the user has entered,

	(React.ElementType) • handle (string)	along with their icon.
Showcase (user-profile)	PokemonCardImage • image (string) • name (string)	Displays the maximum of three cards the user has selected in an organized manner for the user profile.
TradeList	PokemonCardImage • image (string) • name (string)	Displays the cards the user has marked for trading in a 3-cards-a-row grid.
WishList	PokemonCardImage • image (string) • name (string)	Displays the cards the user has wish-listed in a 3-cards-a-row grid.
AccountOptions	None	Displays the menu that holds the block and report account options.
Showcase (edit-profile)	PokemonCardImage • image (string) • name (string)	Displays the maximum of three cards the user has selected in an organized manner for the edit profile page, giving the option to remove cards as well as add them.
DeleteAccount	password (string)	Displays the button to delete the user's account, prompting them to enter their account password for security purposes before the account can actually be deleted.
AuthForm	Username or email (string) Password (string)	Displays 2 input fields, one for username or email and one for password along with a log in button. The user is also given an option to sign in with Google or Apple or sign up.
RegistrationForm	Email (string) Username (string)	Displays 4 input fields, one for each input variable. Only if

	Password (string) Retype password (string)	the email has never been registered before, the password meets the criteria, and the two passwords match, then the user can click the sign up button.
ResetPassword Form	Password (string) Retype password (string)	Users are only directed to this page through an email link with a unique uuid that expires within a set amount of time. Displays 2 input fields one for each input variable. The same constraint for passwords in RegistrationForm also applies in this component.