Project *Pokédex* – Architecture Design Document

Team: Ghosti Matas, Uma Desai, Blake Mosley, Nura Mouktar, Carlos Diaz

1. Introduction

This document defines the technical architecture of *Project Pokédex*. It outlines the system components, key features, data flow, and architecture layers involved in building and deploying the application. The architecture focuses on an AI-powered card collection system that leverages machine learning models for card recognition and integrates backend and frontend components for a smooth user experience.

2. System Overview

Project Pokédex is an intelligent web-based application that allows users to scan Pokémon cards, automatically identify them using machine learning, and view detailed metadata. The system also provides user profile management, card collection features, and interactive AI capabilities.

Key Features

- **ML Card Identification** – Real-time image recognition using YOLOv8 and CLIP.
- **Profile Creation** – Secure user authentication and personalized collection storage.
- **Card Collection Management** – Users can browse, organize, and manage their collections.
- **Card Metadata Display** – Rich Pokémon card data retrieved from external APIs.
- **AI Voice Relay** – Voice-assisted interaction for accessibility and engagement.

3. Architecture Overview

The application follows a modular architecture with distinct layers for frontend, backend, database, and AI API services.

Frontend

- Languages: HTML, CSS, JavaScript
- Frameworks: Vue JS, Tailwind CSS
- API's: HTML5 media devices API
- Role: UI/UX, image capture/upload, user authentication interfaces, and displaying card metadata.

Backend

- Language: Python
- Framework: Flask, Pytorch
- Software tools: Pokémon TCG SDK
- Role: Handling API requests, routing, business logic, ML model interaction, and communication with the database.

Database/Service

- Database: Supabase (PostgreSQL)
- API: Pokémon TCG SDK Card data API
- Storage: Supabase Storage Solution
- Role: User profiles, authentication data, and saved card collection data.

ML APIs

- YOLO v8: Object detection for card identification.
- CLIP: Image-text matching to enhance card recognition accuracy.

  (tensorflow/Open AI: local ML usage if necessary vs. online ML dependency)

- OpenAI API (LLM): Conversational AI and voice relay integration.

4. Deployment and Environment Setup

Infrastructure

- Container: Docker
- Role: compartmentalize project for easy deployment. Packages dependencies.

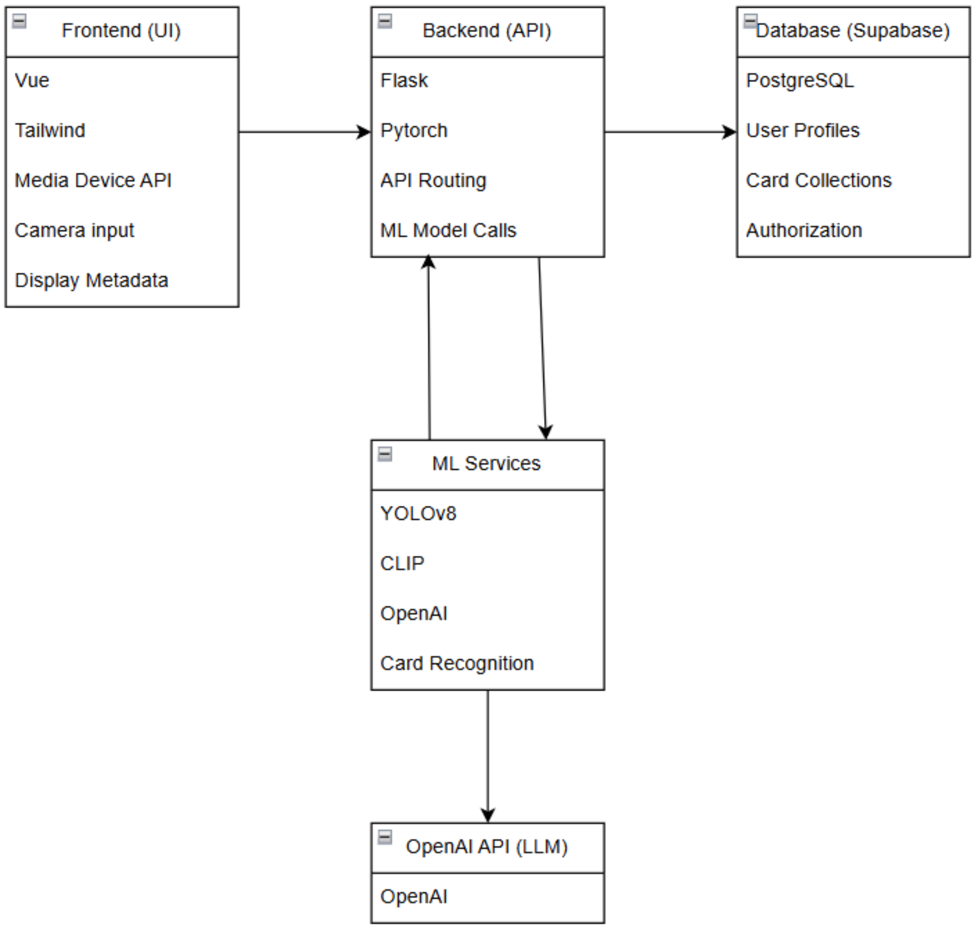
Platform

- Frontend: Local docker image
- Database Hosting: Supabase
- ML generation: Local Host Machine
- Role: deploys the application in an isolated environment. Accounts for any dependencies for ease of use by any device using a docker image.

5. Security Architecture

Authentication

- Supabase Auth API: Json Web tokenizer and Row Level Security infrastructure
- Role: Authenticates user data and verifies user credentials



6. Database Schema

```
Create Table users (
        id UUID primary key gen random UUID();
        email - Text (unique and not null);
        username - Text (unique and not null);
        display name - Text;
        avatar_url Text;
        created_at Timestamptz default now();
        updated_at Timestamptz default now();
);
Create Table cards (
        id UUID primary key gen random UUID();
        // all the following are from external Pokémon TCG API
        external card id Text;
        name Text;
        set name Text;
        image url Text;
        synopsis jsonb; //ML generated synopsis of the card.
        created at Timestamp default now();
);
Create Table collections (
        id UUID primary key gen random UUID();
        owner_id UUID references users(id);
        title Text (from external Pokémon data API);
        visible bool default false;
        card visible image url Text;
        card invisible image png;
        added at Timestamp default now();
);
Create Table scans (
        id UUID primary key gen random UUID();
        user id UUID reference;
        image url png(not null);
        status Text default ‘undefined’;
        result jsonb // ML confidence of match;
        created at Timestamp default now();
);


Create Table ml result of scan (
        id UUID primary key gen random UUID();
        scan id UUID Reference scan id;
        card id UUID Reference card id;
        score Float;
        details jsonb; //identified card and match, and burst time for debug purposes.
        create at Timestamp default now();
);
Create Table voice interaction (
        id UUID primary key gen random UUID();
        user id UUID reference;
        transcript Text;
        voice jsonb; //generated/captured voice module for speaking
        created at Timestamp default now();
);
```

7. Model

App level

- User - attributes: id, email, username, display name, avatar
- Card - id, UUID, name, set, card image, metadata
- Collection - id, owner id, title, visibility, visibility images,
- Scans - id, user id, scan image, status, result
- MLResult - id, scan id, card id, confidence score, details
- VoiceInteraction - id, user id, transcript, voice

8. View

- ❖ Log in page
    - ➢ Purpose:
        - ■ create account
        - ■ log in to existing
    - ➢ Components:
        - ■ Login request
        - ■ email text box
        - ■ pass text box
        - ■ sign up button
        - ■ log in button
- ❖ Pokédex page
    - ➢ Purpose:
        - ■ Profile settings, library, and scanning access.
        - ■ ML chat capability
        - ■ ML voice chat on scan

- Components
  - Tool bar: profile icon, library slider, 3 setting switchers.
  - Display: camera interaction, card display mode/profile display mode (setting switchers change pages for each mode)
    - Card display mode pages
      - card image
      - card meta data
      - card trade data
    - Profile display mode pages
      - profile settings
        - edit name, description, avatar, privacy
      - Accessibility settings
        - TBD
      - App settings
        - download/delete app data
        - configure data privacy
        - Customize app
        - TBD
  - Rotom chat window: activate rotom button, Rotom mode/text mode
    - Text mode: displays additional text during Card display mode. Delivers transcript of TTS.
    - Rotom mode: interactive chatbot

❖ Library page
- Purpose:
  - display library of collected and undiscovered cards.
  - sort cards by filter (collected/undiscovered).
  - search for cards by name/image.
- Components:
  - Card list
  - card icon buttons
  - card filter (collected/undiscovered)
  - library page button
  - scan page button
  - Rotom button
    - Opens chat window for extended search options

❖ Scans page
- Purpose:
  - display scans of confirmed and unverified camera captures.
  - sort cards by filter

        - search with AI
- Components:
    - Scan list
    - Scan icon buttons
    - scan filter (captured/unconfirmed)
    - library page button
    - scan page button
    - Rotom button
        - Opens chat window for more abstract search

9. Functions and Controllers

(*subject to change*)

```
# app/api/cards.py (blueprint 'cards')
@bp.route('/api/cards/<card_id>', methods=['GET'])
def get_card(card_id): -> returns card JSON

@bp.route('/api/cards/search', methods=['GET'])
def search_cards(): -> params: q, page, per_page

# app/api/collections.py (blueprint 'collections')
@bp.route('/api/collections', methods=['GET'])
def list_collections(): -> returns user's collections

@bp.route('/api/collections/<id>/items', methods=['POST'])
def add_collection_item(id): -> body: {card_id}

# app/api/scan.py (blueprint 'scan' or 'ml')
@bp.route('/html/media-devices/scan', methods=['POST'])
def submit_scan(): -> body: {image_base64 or image_url}, returns scan_id

@bp.route('/html/media-devices/scan/<scan_id>', methods=['GET'])
def get_scan_result(scan_id): -> returns ml results

# app/api/auth.py (blueprint 'auth') - primarily passthrough to Supabase - token
verification middleware
```

10. URL’s

(*subject to change, possible representation of main endpoints*)

Auth

- `POST /api/auth/signup` - body: `{email, password, username}` -> create user (or use Supabase).
- `POST /api/auth/login` - body: `{email, password}` -> returns JWT (handled by Supabase recommended).

Cards

- `GET /api/cards/:cardId` - fetch card metadata (response: card JSON).
- `GET /api/cards?name=...` - search cards by name / set.
- `GET /api/cards/:cardId/image` - serve card image (or proxied URL).

Collections

- `GET /api/collections` - list collections for current user.
- `GET /api/collections/:id/items` - view collection item `{card_id}.`
- `POST /api/collections/:id/items` - add item `{card_id}.`
- `PATCH /api/collections/:id/items/:itemId` - edit item (, notes).
- `DELETE /api/collections/:id/items/:itemId` - remove item.

Scan / ML

- `POST /api/scan` - upload image or URL; starts processing. Body: `{image_base64|image_url, scan_options}` → returns `{scan_id}`.

- `GET /api/scan/:scan_id` - returns status (`pending|processed|failed`) and `result` JSON.
- `GET /api/scan/:scan_id/results` - returns MLResult entries.

Voice / AI

- `POST /api/voice` - upload audio or transcript; returns response from LLM/voice relay.
- `GET /api/voice/:id` - fetch interaction.

Admin

- `GET /api/admin/ml/status` - ML model status.
- `POST /api/admin/ml/retrain` - trigger retrain (protected).

11. Use Cases

Card Scan Case:

1. User clicks Scan → front-end captures image (HTML5).
2. Frontend `POST/api/scan` with image.
    a. Controller: `scan.submit_scan()` → validates, stores image (Supabase Storage), creates `scans` entry.
    b. Service: `MLQueueService.enqueue(scan_id, image_url)`
3. ML worker picks job → runs YOLOv8 detection + CLIP matching.
    a. Worker writes `ml_results` and updates `scan` `(status:processed)`
4. Frontend polls `GET/api/scan/:id` → reads results, shows candidate matches.
5. User selects match → frontend `POST/api/collections/:id/items` to add card.
    a. Controller: `collections.add_collection_item()` writes `collection_items.`

Search library case:

1. Frontend `GET/api/cards  || GET/scans`
2. Click item → `GET/api/cards/:id` → `CardController.get_card || GET/api/scan/:id`