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
- Role: Handling API requests, routing, business logic, ML model interaction, and communication with the database.

Database/Service

- Database: Supabase (PostgreSQL)
- 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