University of Perpetual Help System Laguna
City of Biñan, Laguna, Philippines, 4024
(02)779-5310 • (049)554-5150 • (049)554-5161
www. uphsl.edu.ph

UPHSL College of Computer Studies

Binan, Laguna

PokeDex

Programming Languages Laboratory

Bachelor of Science in Computer Science

J4A

Submitted to

Prof. Grachel Eliza Greta

Submitted by

Kelvin Adam Aninang

Angela Cabanes

Michaela Jornaes

November 2025

Executive Summary

The Pokedex software is an encyclopedia for cataloging information about the Pokémon. Its purpose is to educate users about the diverse creatures living in the Pokémon world, including detailed stats, physical attributes (height and weight), and species data. The system addresses the challenge of navigating the vast list of regional Pokémon canon by centralizing and organizing this data for quick and easy access.

The target audience for the Pokedex are Pokémon lovers, competitive players, and casual fans who require quick, reliable access to creature data. The system operates as an information retrieval tool, focusing on the presentation of pre-existing Pokémon data. It includes information for every Pokémon organized by their respective regions.

The Pokedex has a few key features to enhance the user experience. First is the Regional Tab. It filters and views a list of Pokémon within the specific region. Second is an information widget that displays details such as base stats, species characteristics, height, and weight when a Pokémon is selected. Lastly is the search function. You can search up the Pokémon either of its ID number or through its name

System Overview

1. Background of the Project

This started was an idea for our project feature within the python programming of our Pokémon game called “Who’s that Pokémon?” Their idea at first was to make a simple pokedex to see their list of pokemon for the user to review. With that in mind, I decided to make the Pokedex with its full information as a same mindset for C++.

2. Problem addressed by the system

With the 2nd project should be inline with the 1st project, aka the Pokémon guessing game, I decided to make a fully fleshed out pokedex with information for the user to see and review. For pokemon enthusiast beginner who are wondering what pokemons there are per region, this system provide the exact details and what they are.

3. Objective of the Developed Solution

a. API Fetching - Gather the Data of every pokemon from the PokeAPI
b. Region Tab - Cluster the Pokémon per region
c. Display Tab- Gather Each Pokémon data from name to description
d. Search Function - Search the name of the pokemon either ID or Name

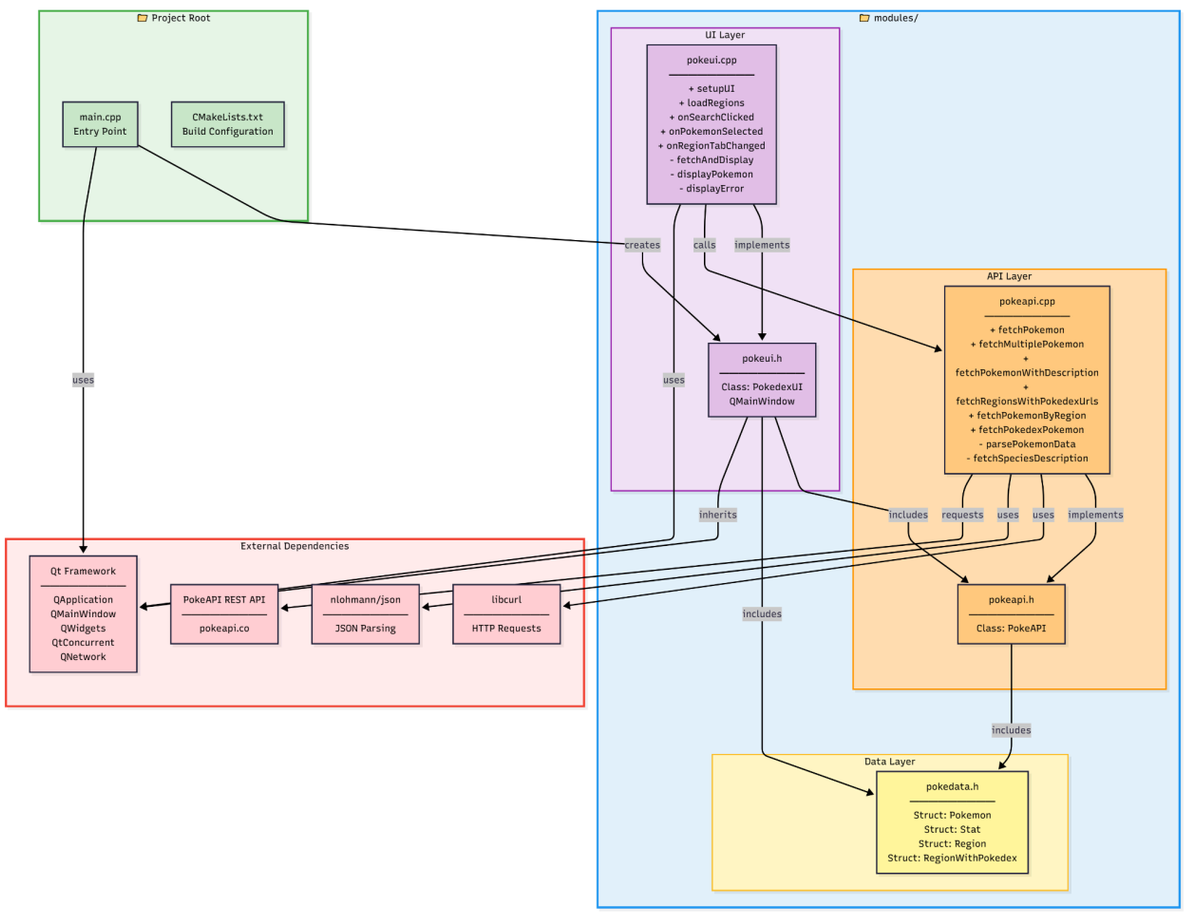
4. Development Environment and Tool Used

Category	Tool/Version	Purpose
Programming Language	C++(23)	Main Implementation language
IDE/Editor	Visual Studio Code	Code editing and debugging
Version Control	GIthub	Team collaboration and history tracking

Libraries	QT, Libcurl, nwollah-json	For frontend development/ HTTP response, and JSON data manipulation

Software Design & Architecture

Diagrams of Program Structure



Data Layer

This holds the foundation of the pokedex software, wherein pokedata.h structures data for the API and the UI to hold the data. This makes it the blueprint when fetching data from the pokeapi.

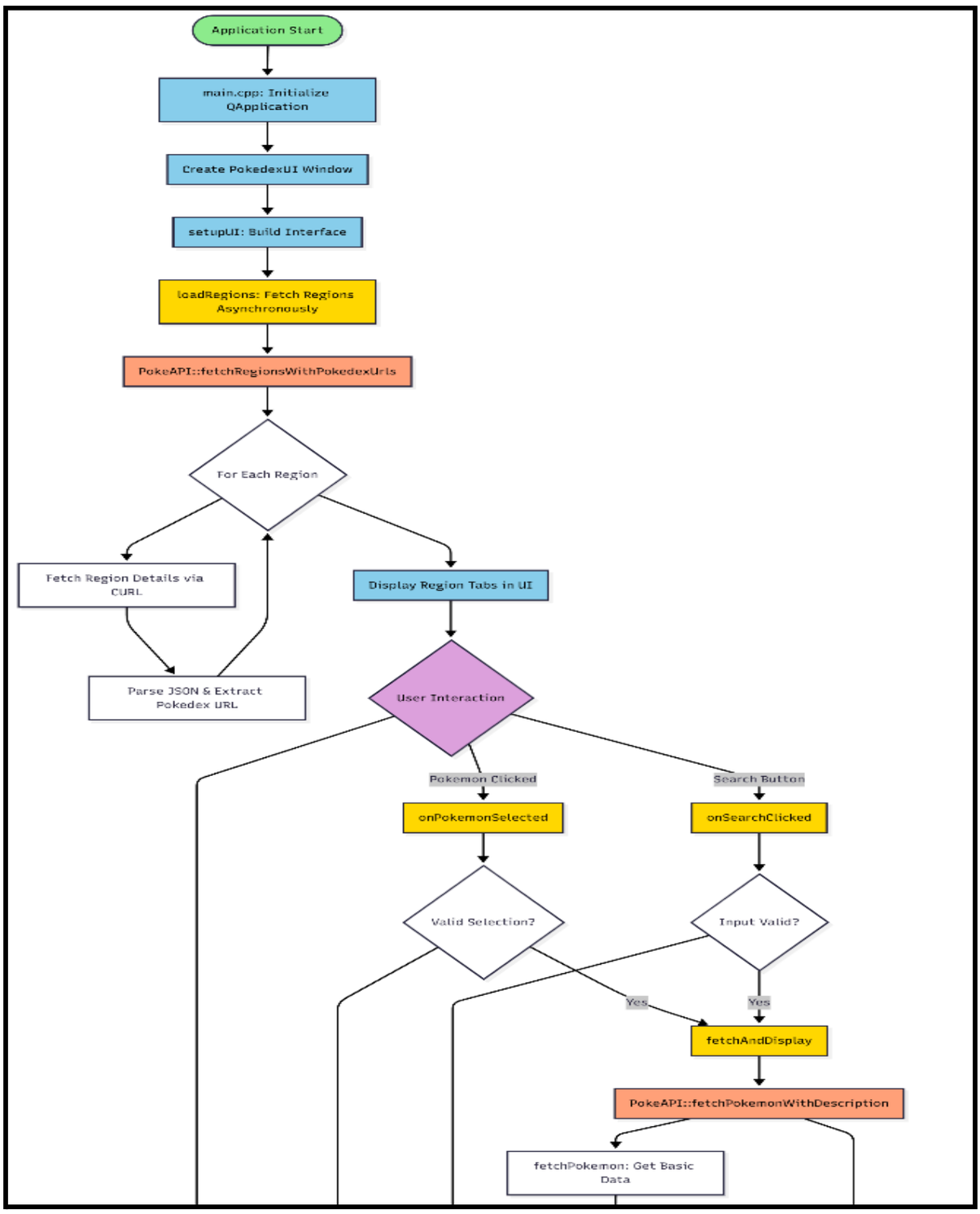
API layer

The api layer handles the communication between the pokeapi to the pokedex. In away, it fetches the necessary data needed for the pokedex to display. This holds the data about the name, stats, description, url sprites, and region of the pokemon through the libcurl library. Once it gets the necessary data, it converts the data through nlohmann-json to its intended structure and asynchronously returns the data

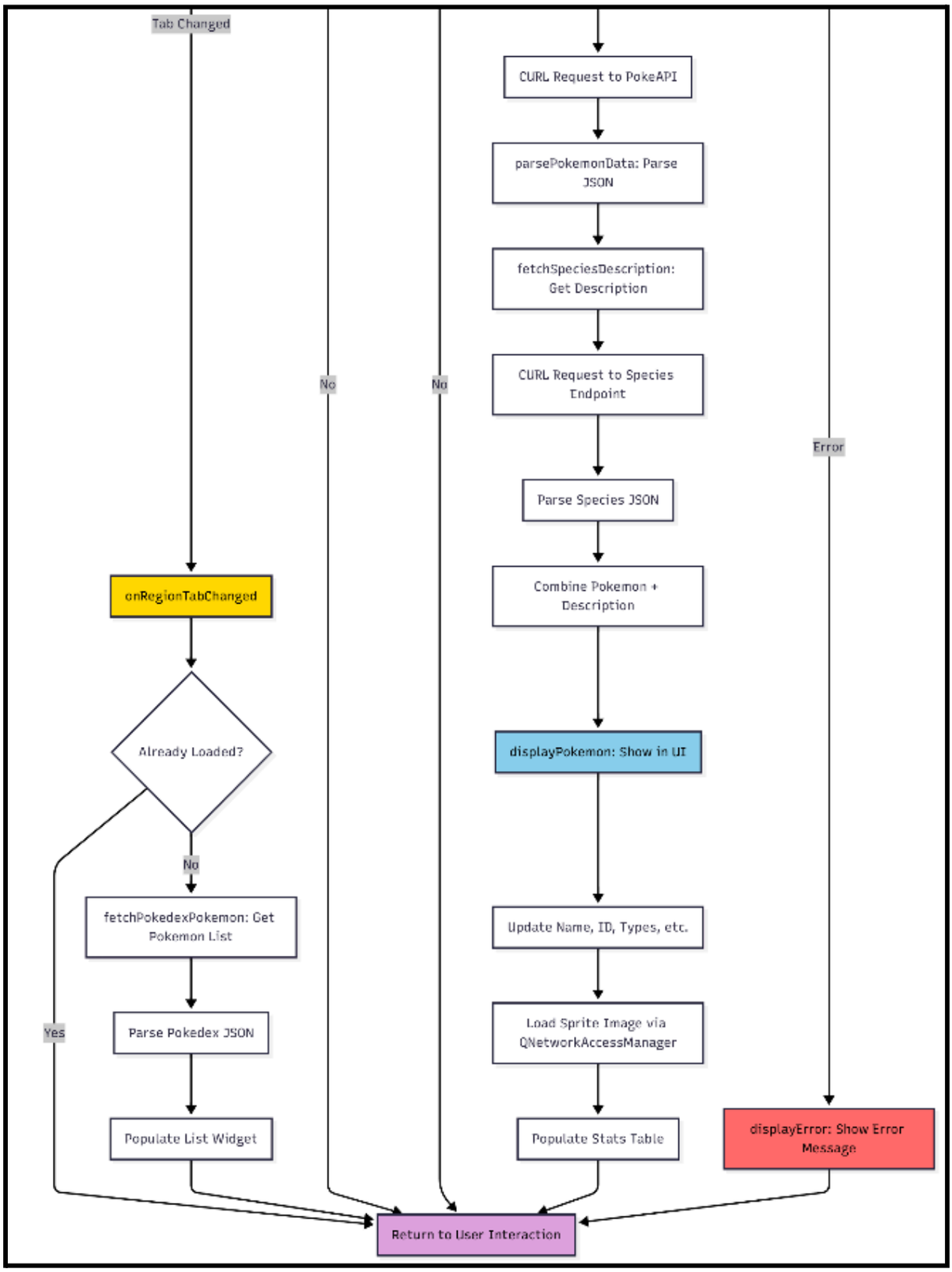
UI Layer

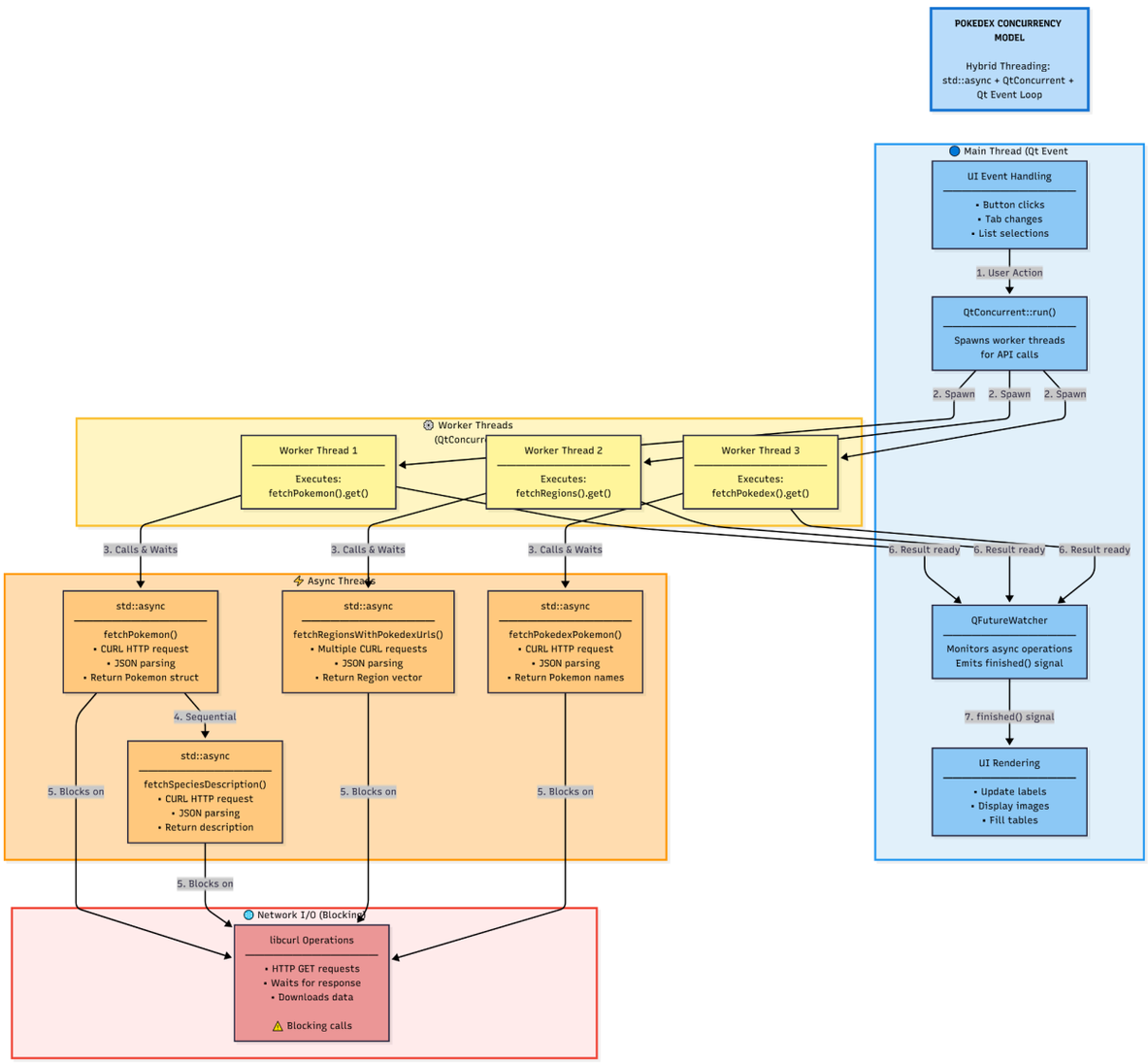
This is where we visualize the data into the pokedex we made. We first call the widgets for displaying the tabs, list, and buttons. Calls the API to fetch the pokemon data while using QTConcurrent to avoid freezing the UI and displays the necessary data to show.

Flowchart



Concurrency Model


Tab Changed
onRegionTabChanged
Already Loaded?
No
Yes
fetchPokedexPokemon: Get Pokemon List
Parse Pokedex JSON
Populate List Widget
No
No
CURL Request to PokeAPI
parsePokemonData: Parse JSON
fetchSpeciesDescription: Get Description
CURL Request to Species Endpoint
Parse Species JSON
Combine Pokemon + Description
displayPokemon: Show in UI
Update Name, ID, Types, etc.
Load Sprite Image via QNetworkAccessManager
Populate Stats Table
Error
displayError: Show Error Message
Return to User Interaction

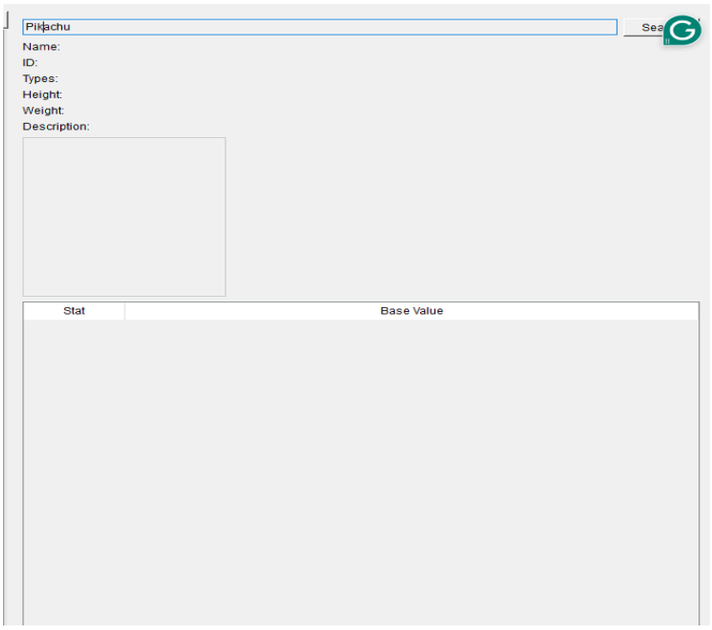


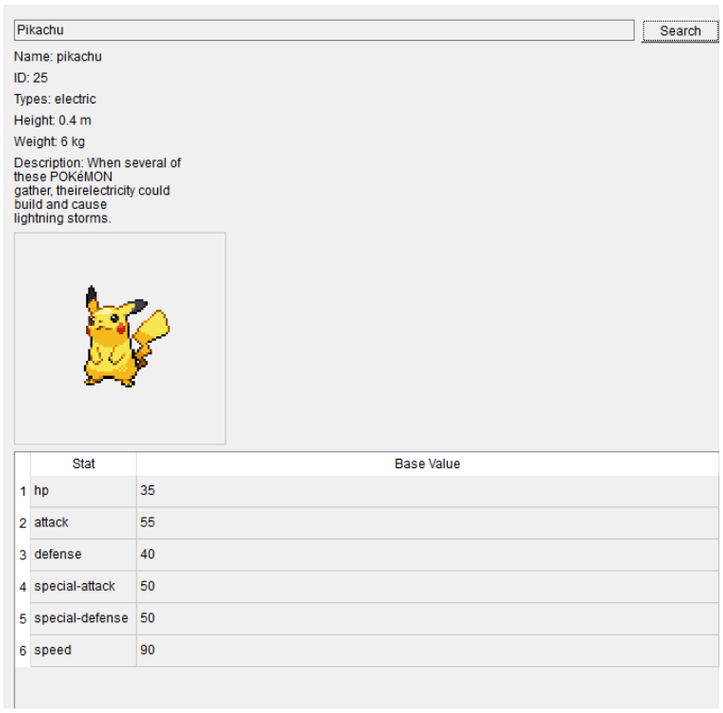
The Pokedex has implemented a hybrid concurrency with QT's threading and C++ async operations to maintain a responsive UI while working with potential slow network. The main thread handles all the UI rendering and user interactions, QtConcurrent worker threads work to manage API calls and the C++ async library perform HTTP request and JSON parsing.

The main thread never blocks on network operations, which is needed for the UI responsiveness. Whenever a user picks a pokemon or region from the API, the UI uses QtConcurrent::run() to make a worker thread that executes the API call by .get() on the std::future objects returned by the PokeAPI class methods, effectively waiting for the async operation to complete without freezing the UI. Then there's the asynchronous function (std::async) that guarantees each API function runs its own thread, creating a second level of concurrency. Once the data or API fetch is complete, it gives off a Qt single slot mechanism called QFutureWatcher. It emits a signal (finished())that automatically queues back up to the main thread. This ensures the UI is always updated safely and avoid race condition and threading bugs.

Implementation

1. **Sample Input/Output**






```
[main] Building folder: c:/Users/kelvi/OneDrive/Documents/GitHub/j4a-pl-cs4/pokedex-project/build all
[build] Starting build
[proc] Executing command: C:\Qt\Tools\CMake_64\bin\cmake.exe --build c:/Users/kelvi/OneDrive/Documents/GitHub/j4a-pl-cs4/pokedex-project/
build --config Release --target all --
[build] [1/4  25% :: 1.444] Automatic MOC and UIC for target pokedex-project
[build] [2/3  66% :: 14.644] Building CXX object CMakeFiles/pokedex-project.dir/main.cpp.obj
[build] [3/3 100% :: 18.612] Linking CXX executable pokedex-project.exe
[driver] Build completed: 00:00:19.050
[build] Build finished with exit code 0
```

2. Key Highlight

a. Control Flow Logic

Entry point of the software

main.cpp > main(int, char * [])

```cpp
#include <QApplication>
#include "modules/pokeui.h"

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    PokedexUI window;
    window.show();
    return app.exec();
}
```

UI Initilization

```cpp
// Constructor
PokedexUI::PokedexUI(QWidget *parent) : QMainWindow(parent) {
    setupUI();
    setWindowTitle("Pokedex v1.0");
    setFixedSize(1000, 800);
    setStyleSheet("background-color: #f0f0f0; font-family: Arial;");
    loadRegions();
}
```

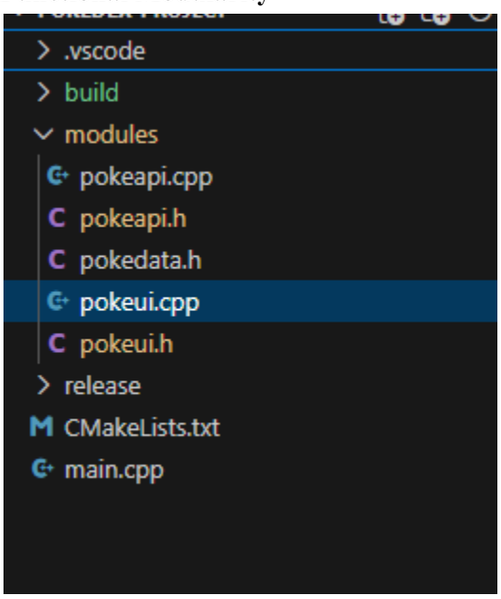
User control flow

```cpp
void PokedexUI::onPokemonSelected(QListWidgetItem *item) {
    // Prevent fetching if clicking the placeholder text
    QString text = item->text();
    if(text != "Select a region to load a pokemon..." &&
       text != "Loading..." &&
       text != "No Pokémon found for this region") {
        fetchAndDisplay(text);
    }
}

void PokedexUI::onSearchClicked() {
    QString query = searchInput->text().trimmed();
    if (!query.isEmpty()) {
        fetchAndDisplay(query);
    }
}
```

Display pokemon

```cpp
void PokedexUI::fetchAndDisplay(const QString& query) {
    std::string qStd = query.toStdString();
    QFuture<Pokemon> future = QtConcurrent::run([qStd]() {
        return PokeAPI::fetchPokemonWithDescription(qStd).get();
    });

    QFutureWatcher<Pokemon> *watcher = new QFutureWatcher<Pokemon>(this);
    connect(watcher, &QFutureWatcher<Pokemon>::finished, this, [this, watcher]() {
        try {
            Pokemon p = watcher->result();
            displayPokemon(p);
        } catch (const std::exception& e) {
            displayError(QString::fromStdString(e.what()));
        }
        watcher->deleteLater();
    });
    watcher->setFuture(future);
}
```

b. **Functional Modularity**



c. **Concurrency**
QtConcurrent Worker Threads

```
void PokedexUI::loadRegions() {
    // Spawn worker thread
    QFuture<std::vector<RegionWithPokedex>> future = QtConcurrent::run([]() {
        return PokeAPI::fetchRegionsWithPokedexUrls().get();
    });
    //Monitors the worker thread from main thread
    QFutureWatcher<std::vector<RegionWithPokedex>> *watcher =
        new QFutureWatcher<std::vector<RegionWithPokedex>>(this);
    //Callback executes on main thread when worker thread finishes
    connect(watcher, &QFutureWatcher<std::vector<RegionWithPokedex>>::finished,
            this, [this, watcher]() {
        std::vector<RegionWithPokedex> regionsWithPokedex = watcher->result();

        for (const auto& rwp : regionsWithPokedex) {
            regions.push_back(rwp.region);
            pokedexUrls[regions.size() - 1] = rwp.pokedex_url;

            QListWidget *list = new QListWidget;
            list->addItem("Select this region to load Pokémon...");
            connect(list, &QListWidget::itemClicked, this, &PokedexUI::onPokemonSelected);
            regionTabs->addTab(list, QString::fromStdString(rwp.region.name));
        }

        connect(regionTabs, &QTabWidget::currentChanged, this, &PokedexUI::onRegionTabChanged);

        // Auto-load first region
        if (!regions.empty()) {
            onRegionTabChanged(0);
        }

        watcher->deleteLater();
    });
    watcher->setFuture(future);
}
```

Async Function per API fetch

```
// Async fetch a single Pokémon
std::future<Pokemon> PokeAPI::fetchPokemon(const std::string& query) {
    // std::launch::async forces a new thread
    return std::async(std::launch::async, [query]() {
        CURL* curl = curl_easy_init();
        std::string url = "https://pokeapi.co/api/v2/pokemon/" + query;
        std::string response;

        if (curl) {
            curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
            curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
            curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response);

            // IMPORTANT: Set User-Agent, some APIs reject requests without it
            curl_easy_setopt(curl, CURLOPT_USERAGENT, "libcurl-agent/1.0");

            CURLcode res = curl_easy_perform(curl);
            curl_easy_cleanup(curl);

            if (res == CURLE_OK) {
                return parsePokemonData(response);
            } else {
                // Return a dummy pokemon on error or throw
                Pokemon errorPoke;
                errorPoke.name = "Error: " + std::string(curl_easy_strerror(res));
                return errorPoke;
            }
        }
        throw std::runtime_error("CURL initialization failed");
    });
}
```

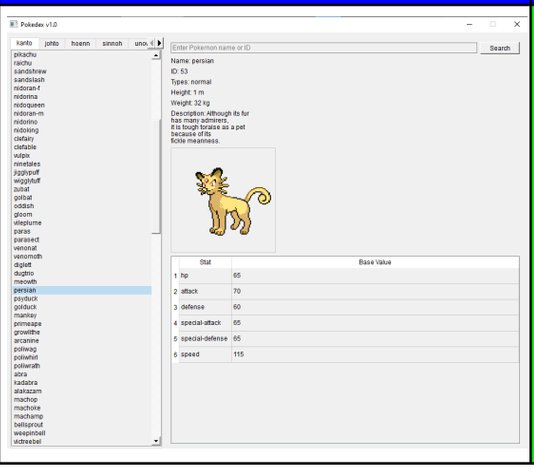
3. Explanation of Git Commit

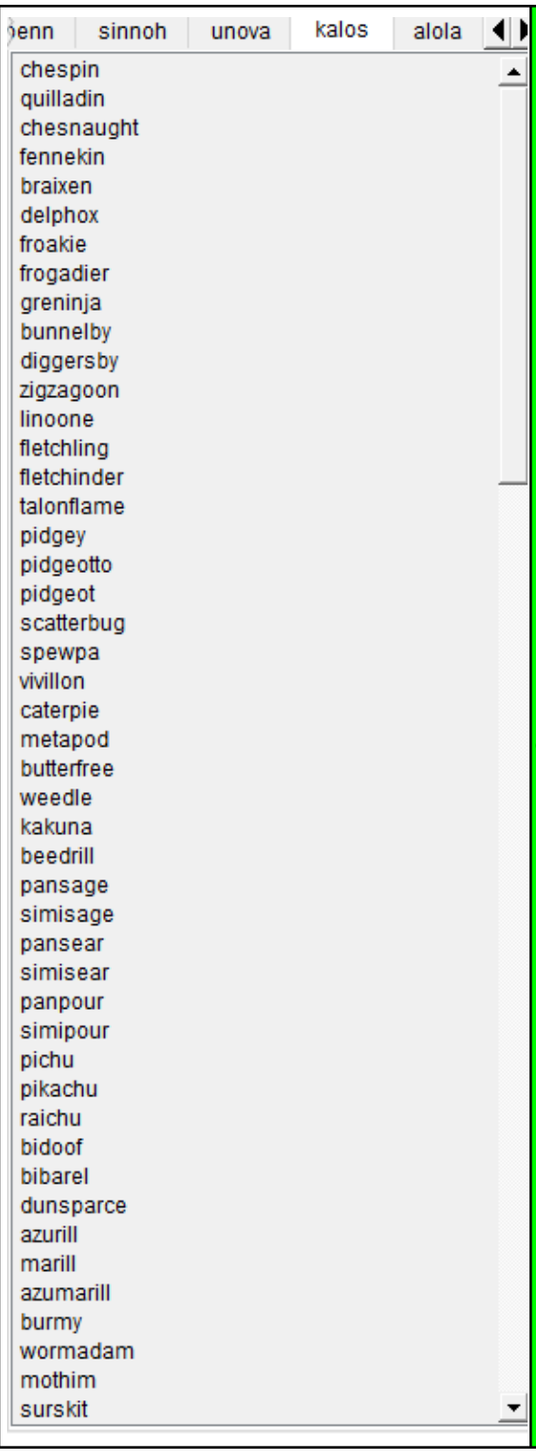
Git commit logs show how you and your group contribute to each other when developing a system.

Git commit helps us understand what our team is makingJ or contributing to the system, whether it be a feature or bug fixes. This also shows accountability if ever something goes wrong within the system. This is why we work on branches to minimize those mistakes and contribute to the system more safely.

Testing and Evaluation

1. Test Cases

Feature	User Interaction	Expected Result (Pass Criteria)	Proof of Technology	Status
Fetch Pokemon	1. Click Pokemon or Search to Display	The software would display the specific pokemon the user is searching for		Passed

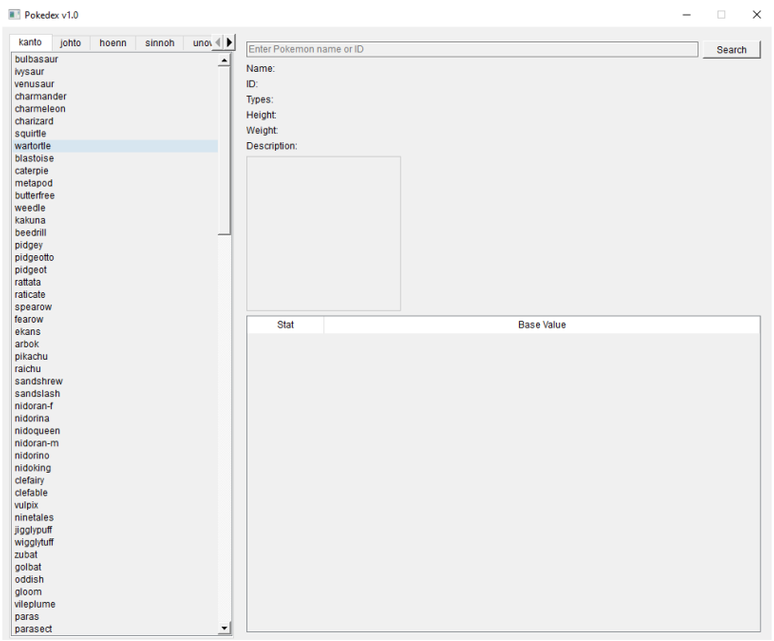
Region load	1. Click the region to load the pokemon list	The tab on the left should show the set of pokemon that live in that region		Passed

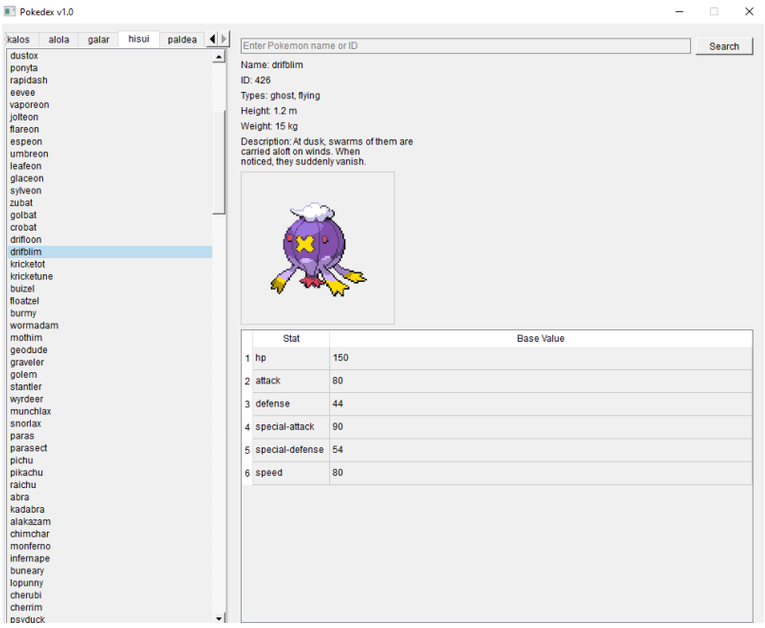
```
Test 1: Fetching Pikachu...
  SUCCESS!
  ID: 25
  Name: pikachu
  Type: electric
  Height: 0.4 m
  Weight: 6 kg

Test 2: Fetching Charizard with description...
  SUCCESS!
  ID: 6
  Name: charizard
  Description: Spits fire that
is hot enough to
melt boulders.♀Known to cau...

Test 3: Fetching first 3 Pokemon...
  - 1: bulbasaur
  - 2: ivysaur
  - 3: venusaur
  SUCCESS! All 3 Pokemon fetched.

=== API Tests Complete! ===
```

2. Discussions of results, issues, and limitations

   After testing, the results came clear and the API fetching works. Developing the UI under Qt is another work to go around, especially with its concurrency. There were a few drawbacks since the pokedex fetches 1025 pokemon. The solution to it was to implement a regional tab to partition the amount of pokemon to be loaded simultaneously. Next was its speeds for UI. Sure that everything was loaded correctly but it was identified that the UI was slower when displaying the pokemons based on regions. So what we did was to cache the pokemon region if its loaded first. Once it was fetched, it would only load the pre-existing data within the regional pokemon when you go back to the previous tab

3. Screenshots of working output





Ethical and Professional Reflection

1. **How did your team ensure ethical collaboration (no plagiarism, fair contribution)?**

   We utilized Git and GitHub to maintain a transparent history of contributions. Each member's work is verifiable through commit logs, ensuring accountability. To avoid plagiarism, we strictly adhered to Open Source licensing by properly importing and attributing external libraries rather than copying source code directly. We also divided the project using a Modular Architecture UI, Logic, API, which naturally enforced fair division of labor without overlapping code. We also contributed to each other's documentation and user interface development, balancing the workload across the different languages (Python and C++) used by the group.

2. **How does your system ensure data privacy (if applicable) and responsible programming?**

   While our system does not store personal information, we practiced Data Minimization by keeping game states in ephemeral memory rather than writing to persistent storage. Regarding responsible programming, we implemented API stewardship. Instead of spamming the PokéAPI with requests on every frame update, we fetch data only when necessary and cache results where possible to respect the provider's server resources and rate limits

3. **What lessons can you apply from professional practice and version control ethics?**

We applied the industry standard of making small, descriptive changes rather than bulk, vague uploads and making each draft branches (e.g. draft-angela, draft-aninang, and draft-mickz). This ensures that if a bug is introduced, we can easily rollback without breaking the entire build.

Independent Learning Courses

1. What new concept or tool they independently learned (e.g., Git branching, threading, or modular architecture).
2. How it improved their contribution.

Aninang, Kelvin Adam

Learning API through this project brings me so much insight on how to handle and call data from a different source. Handling API is a tricky but as long as the document exists, I could understand how it functions and where to use them for our system. I also learned how concurrency not only exists within the data but for UI. Worker threads do help a lot when managing 151 pokemons for CPU processing.

Learning these concept and tools, I have utilized and applied this knowledge to our 2nd project called Pokedex. These knowledge proves working with different languages can sometimes parallel because I applied the same of concurrency with worker threads to not only the 151 pokemons but to the quiz to avoid UI freezing. With these, the system would run smoothly just like the pokedex

Cabanes, Angela

This project really is a breakdown of every lesson taught in programming. I learned about the difference between syntax and systematic errors and how to avoid them, which is achieved by simply understanding the logic of your program better and not forgetting semicolons and such. I also learned about installing extensions like linters, the use of tokens, variable scopes, and subprograms (modularity and abstraction). The subprograms are represented by the multiple program files inside our repo, and the abstraction is the use of the API itself. I also used concurrency, as this program fetches data from the API and runs the time at the same time without crashing the whole app. Adding the feature “view catalog” took a lot of time even though the said feature may be simple, as the app kept crashing. With that, I also learned to use threading again so you can wait for everything to wait on the catalog and still have the app running. Doing this activity really is tricky, but I am also learning as I go along.
Other than that, I also learned how to do Git branching, as I am new to Git. Everything I have learned, I have applied to the internship and project I am currently working on. I learned how to use them in a way that is useful and would expose the contributions you made inside the project. Overall, as stressful as it may be, I really enjoyed doing it. I had fun learning.

Jornales, Michaela

I learned how to implement Python Threading in the Tkinter GUI application. And then using threading modules to run background tasks parallel to the main application loop.

The Pokémon API helped the system by fetching data from the internet but running this on the main thread caused the entire application window to freeze until the download was complete.

References

libcurl - everything curl. (n.d.). https://everything.curl.dev/libcurl/index.html

PokéAPI. (n.d.). https://pokeapi.co/

QT Examples and Tutorials | QT 6.10. (n.d.). https://doc.qt.io/qt-6/qtexamplesandtutorials.html

W3Schools.com. (n.d.). https://www.w3schools.com/cpp/