Phase 3: Enhancements (Extra Credit)

This document outlines the features implemented for Phase 3 of the TrojanBook project. This phase focuses on extending the functionality beyond the core requirements, incorporating an expanded data model, a recommendation engine, and a web-based user interface that interacts directly with compiled C++ code and a Python script for backend logic.

Goals

1. **Enlarge Database (C++):** Extend the C++ data structures to store more user information (like major, college) using a flexible format and update file I/O accordingly.
2. **Recommendation System (Python):** Develop a Python script to analyze the C++ database file and suggest potential friend connections based on shared attributes. This script is the *sole* source of recommendation logic.
3. **Web User Interface (Node.js):** Create a dynamic web interface using Node.js and Express. **The Node.js backend calls the compiled C++ executable for all core data operations (load, add, remove, connect, get details) and executes the Python script for recommendations.**

Implementation Details

This section provides a more detailed breakdown of how each component was implemented.

1. C++ Database Enhancement & Command-Line Interface

The core C++ codebase from Phase 1 and 2 was extended to handle more flexible user data and refactored to be controlled via command-line arguments instead of an interactive menu.

- **`Person` Class (`person.h`, `person.cpp`):**
  - Added `std::map<std::string, std::string> additional_info;` for flexible key-value data.
  - Added helper methods (`add_info`, `get_info`, `get_all_info`).
  - **Note:** Basic data members (`f_name`, `l_name`, `birthdate`, etc.) remain private. Access from outside `Person` or `Network` requires getters (which were *not* added to maintain minimal changes) or friend access.
- **`Network` Class (`network.h`, `network.cpp`):**
  - **File Format:** Extended `networkDB.txt` format to include `key:value` pairs for additional info, terminated by `---INFO_END---` before friend codes.
  - **File I/O (`saveDB`, `loadDB`):** Updated to handle the new format and improved friend linking logic using maps during load.

  - **New Methods for Command-Line Control:**
    * `searchByCodeName(string codeName)`: Finds a person by their unique codeName.
    * `removeByCodeName(string codeName)`: Removes a person by codeName and crucially, **also removes them from the friend lists of all other people** before deleting the person object.
    * `printAllSummaries()`: Prints a summary (codeName:fName:lName) of all people to standard output.
    * `printPersonDetailsParsable(string codeName)`: Prints detailed information for a specific person to standard output in a machine-parsable format (using `key:value` lines), leveraging a private helper method (`print_details_parsable_helper`) that can access private `Person` members due to `Network` being a `friend`.
  - The interactive `showMenu()` method remains but is no longer used by the primary executable.
- **Executable (`test_network.cpp`, compiled to `test_network.o`):**
  - The `main` function was rewritten to parse command-line arguments (`argc`, `argv`).
  - It accepts commands like `--get-all`, `--get-details <codename>`, `--add --fname <fname> ...`, `--remove <codename>`, `--connect <code1> <code2>`.
  - Based on the command, it loads the database (`networkDB.txt`), calls the appropriate `Network` class methods, performs the action, prints results (or errors) to standard output/error, and saves the database if modifications occurred.
  - This executable is now called directly by the Node.js server.

2. Python Recommendation System

A separate Python script (`recommendations.py`) performs offline analysis and generates friend recommendations. **This script is the only component responsible for generating recommendations.**

- **Technology Choice:** Python, operating independently of the C++ runtime.
- **Data Input:** Parses the `networkDB.txt` file, mirroring the C++ format including `---INFO_END---`.
- **Recommendation Logic (`recommend_friends` function):** Uses content-based filtering, scoring potential friends based on shared `additional_info` (key/value matches) and age proximity. Excludes self and existing friends.
- **Integration:** The script accepts a target `codeName` via command-line argument. The Node.js server executes this script (`python recommendations.py <target_codename>`), captures its standard output (the list of recommended codeNames), and passes this list to the

web UI. **The Node.js server and the frontend JavaScript do not contain any recommendation logic themselves.**

3. Node.js Web UI

A web-based frontend and backend using Node.js, now acting primarily as an interface layer to the C++ executable and the Python script.

- **Backend (`server.js`):**
  - **Framework:** Express.js.
  - **Data Handling Shift: Removed** the JavaScript `parseNetworkFile` and `saveNetworkFile` functions. All core data operations now rely on calling the compiled C++ executable (`test_network.o`) using `child_process.exec`.
  - **C++ Interaction:** Implemented a helper function (`runCppTool`) to manage executing the C++ tool with specified command-line arguments and parsing its `stdout` and `stderr`.
  - **API Endpoints:** The API endpoints (`/api/people`, `/api/people/:codename`, `/api/people` (POST), `/api/people/:codename` (DELETE), `/api/connect`) were refactored to:
    1. Construct the correct command-line arguments for `test_network.o`.
    2. Call `runCppTool`.
    3. Parse the output from the C++ tool (e.g., summary lists, detail lines, success/error messages).
    4. Format the results into JSON responses for the frontend.
    5. Handle errors reported by the C++ tool via `stderr` or exit codes.
  - **Recommendations Endpoint (`/api/recommendations/:codename`):** This endpoint remains unchanged in its *logic* – it still executes the `recommendations.py` script using `child_process.exec` and returns the script's output.
  - **Demo Data Endpoint (`/api/generate`):** This endpoint still uses JavaScript to directly write the demo `networkDB.txt` file for simplicity, avoiding the need to encode the demo data within the C++ application.
- **Frontend (`public/` directory):**
  - **Structure (`index.html`), Styling (`style.css`):** No changes required here.
  - **Interactivity (`script.js`):**
    - No changes required to the core logic. It continues to make `fetch` calls to the backend API endpoints.
    - **Crucially, it contains no logic for parsing the `networkDB.txt` file, saving data, connecting users, or generating recommendations itself.** It relies entirely on the backend API, which in turn relies on the C++ executable and the Python script.
    - Includes the fix for attaching event listeners correctly to the dy-

namically generated recommendation connect buttons.

Running Phase 3

Follow these steps to compile and run the different components of Phase 3:

1. Compile C++ Code

Make sure you have a C++ compiler (like g++) installed.

```
# Compile the C++ application with the command-line interface
make clean # Optional: Clean previous builds
make test_network
# This creates the test_network.o executable
```

2. Run C++ Tool (Standalone Test - Optional)

You can access the network Menu by directy running:

```
./test_network.o
```

Or you can test the command-line tool individually:

```
# List all people
./test_network.o --get-all

# Get details for a specific person (replace 'johndoe')
./test_network.o --get-details johndoe

# Add a person
./test_network.o --add --fname New --lname Person --bdate 01/01/2000 --email new@person.com

# Connect two people (replace code names)
./test_network.o --connect newperson johndoe

# Remove a person
./test_network.o --remove newperson
```

3. Run Python Recommendation Script (Standalone Test - Optional)

Make sure you have Python 3 installed.

```
# Test the recommendation script directly (It must be a valid codeName, e.g., davidjohnson)
python recommendations.py davidjohnson
```

This script reads `networkDB.txt` by default and prints recommended codeNames.

4. Run the Node.js Web Server

Make sure you have Node.js and npm installed.

```
# Install dependencies (only needed once)
# npm install express

# Start the server
node server.js
```

The server will start (e.g., `http://localhost:3000`) and will now use `./test_network.o` for data operations.

5. Access the Web UI

Open your web browser and navigate to `http://localhost:3000`.

- The UI functions as before, but the backend now correctly delegates operations to the C++ tool and the Python script.

(This concludes the detailed overview of the Phase 3 implementation)