Andrew Suezaki

Title: Population Generator

High-Level Architecture: Microservice Architecture

I. Purpose

The purpose of this SRS document is to describe the user scope, user stories, and non-functional requirement specifications for a population generator microservice. The purpose of this project is to create a software that provides an easy way to access population information of a state from any year between 2005-2019. The intended audience of this software is the client: Pannapat.

II. Scope

Population information is crucial to several fields of work. Having to search through lengthy documents to get this information can be tedious and time-consuming. Population Generator is a microservice that generates population demographics data based on real US census data. Its features come together to provide users the opportunity to produce realistic datasets about Americans. Population Generator provides an intuitive graphical user interface (GUI) that opens in a window and prompts the user for input. Population Generator also integrates with the Person Generator microservice by Robert Collins and can make requests to generate addresses for select states from within Population Generator. This microservice enables anyone from population scientists to the public to easily access population information. The requirements will be divided into the current and planned functional requirements/user stories and the non-functional requirements.

III. Functional Requirements

Current User stories

- As a user prone to typing errors, I want to see dropdown selection in the UI so that I can ensure I am generating the right information.
- As a forgetful user, I want to see instructions on how to use the software somewhere in the UI so that I always know how to use it.
- As a population scientist, I only want to see information generated that is accurate to real US Census data and relevant to the last couple of years so that I am generating relevant up-to-date information for my research.
    - Input: state and year via dropdown menu
    - Output: Population size for the inputted state and year
- As a Data scientist, I want to export the results in CSV format so I can import it into my database as a test dataset for training my machine learning models.
    - Input: population data
    - Output: CSV file with header and one row per line

Planned User stories

- As a traveler in the US, I want to learn about the people in each city before I visit so I can know what to expect ahead of time
    - Input: city name, 2 letter state abbrev.
    - Output: Demographics breakdown
- As a COVID-19 vaccination distribution strategist, I want to be able to target cities that have older people as a majority population
    - Input: single demographic value (eg female) OR range (eg 65+)
    - Output: List of cities, sortable by percent of people with given demographics, should also show numbers of people
- As an international student coming to the US, I want to see demographics data by zipcode to help me decide which schools to apply to.
    - Input: 5 digit ZIP code
    - Output: total population in a given ZIP code area with a 'demographics breakdown' (including age range, racial/ethnicity code, and any other US Census Bureau demographic measures)
- As a data scientist, I want to generate a plausible population sample so I can train my machine learning models
    - Input: Population sample size, zip code (optional), city/state (optional)
    - Output: Demographics breakdown, breakdown by state of residence (optional)

Current Use Case

- **Name:** Population Generator
- **Actor:** Client: Pannapat from Tapannap, Inc.
- **Flow:**
    - **Alternative Flow 1**
        - **Precondition 1**: user passes arguments/input file in cmd line
        - **Precondition 2**: input csv file follows the required format (with header)
            - `input_year,input_state`
        - **Alternative Flow 1A**
            - 1. User runs program in command line with input csv file as argument.
            - `python population-generator.py input.csv`
        - **Alternative Flow 1B**
            - 1. Program starts normally and user selects a csv file to input in the GUI
        - 2. Program parses csv file
        - 3. Program makes GET request to US Census ACS1 API to get the population size for the requested year
        - 4. Formats generated data to required format for output file
            - `input_year,input_state,output_population_size`
        - 5. Prints requested data to GUI if started with Alt flow 1B
        - 6. Creates and writes data to a csv file within the current directory titled 'output.csv'

- **Alternative Flow 2**
  - **Precondition:** no arguments passed
  - 1. User selects a US Census year
  - 2. User selects a US state
  - 3. User presses 'Generate' button to generate population info
  - 4. Program makes GET request to US Census ACS1 API to get the requested population info
  - 5. Program receives info and prints results to GUI
  - 6. User presses 'Export to CSV' button
  - 7. Creates and writes info into output csv file in the same directory titled 'output.csv'

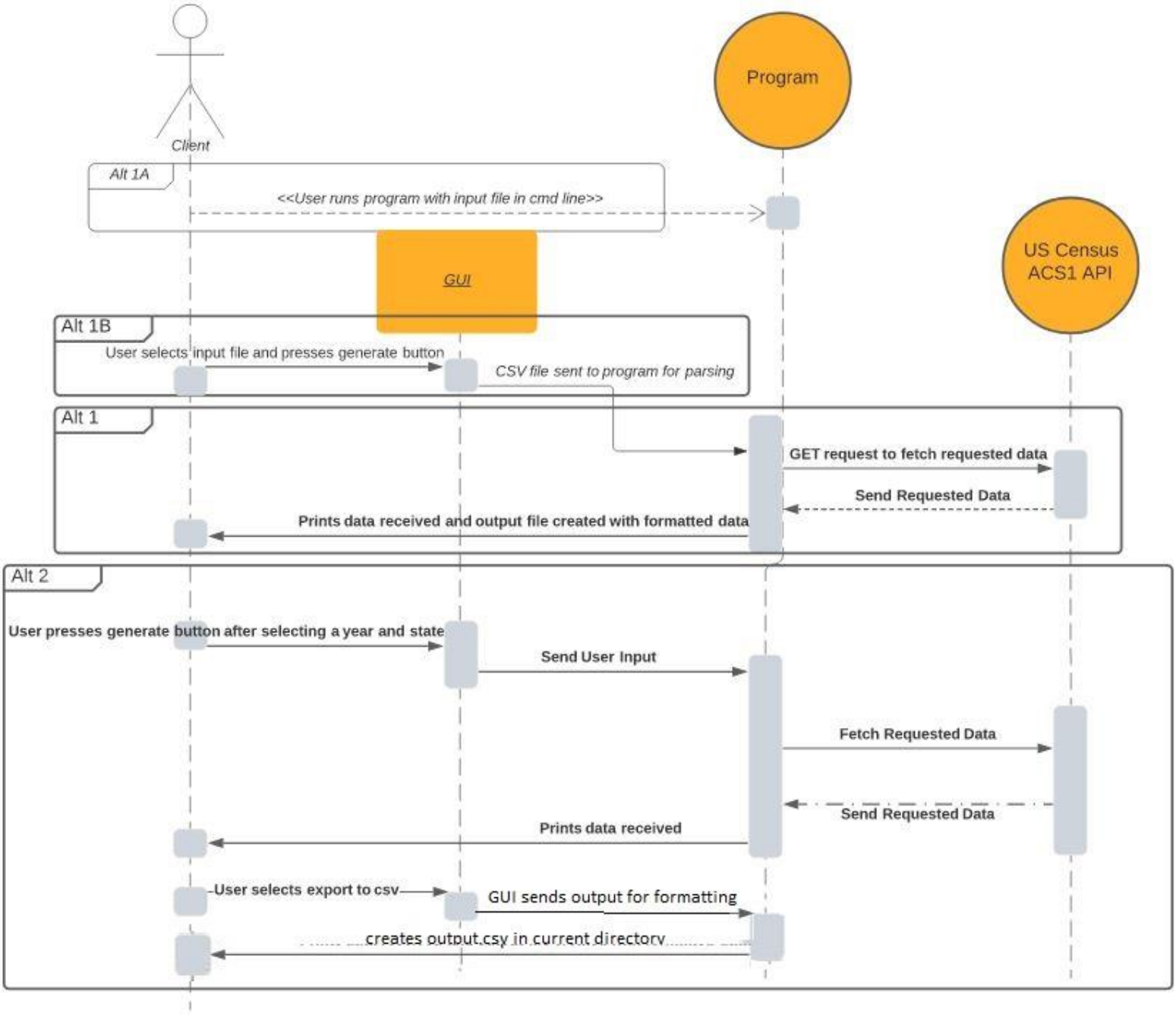
IV. Non-functional Requirements

- **Portability**
  - Requirement
    - The software must be able to run on the following platforms: Windows 10, MacOS 10 or newer, Ubuntu 20 LTS or newer
  - Current
    - Currently, Population Generator has been thoroughly tested to work on Windows 10
  - Planned
    - Population Generator still needs to be tested on MacOS 10+ and Ubuntu 20 LTS+ to meet the requirement.
- **Performance**
  - Requirement
    - The software must display the results within 15 seconds after submitting input
  - Current
    - Currently, population generator always returns results in under 15 seconds.
  - Planned
    - Population Generator displays results faster (in under 10 seconds)
- **Usability**
  - Requirement
    - An independent evaluator must determine the GUI reflects at least 2/3rds of the CSH at the 'satisfactory' level or higher
  - Current
    - Currently the reflection of CSH in my GUI have been evaluated by me and are all satisfactory or higher.
  - Planned
    - Have someone who is not me evaluate how it reflects the CSH

V. Communication Pipe – Person Generator and Content Generator

Population Generator can communicate with the Person Generator microservice created by Robert Collins to make requests given all required csv dataset files for Person Generator are within the same directory as Population Generator and there is connection to the internet. Population Generator prompts the user for input and sends the request with that input to Person Generator through a local CSV file. It waits for Person Generator to complete the request until Person Generator sends a response back in a separate local CSV file. Population Generator receives the response CSV file and prints the results to the console. Max Grier's Content Generator microservice is also able to make requests to Population Generator for population data via the same request/response CSV method.

VI. Code Design UML Sequence Diagram



VII. GUI Design

After careful analysis and revision of the current GUI, the GUI reflects at least 2/3 of the CSH to a satisfactory or higher level. For details on how each CSH is or is not reflected in the current GUI, refer to the bullet list below.

- CSH 1: Explain what new features do and why they are useful

    - The GUI reflects this heuristic in that there is a text prompt explaining how to use the new optional 'Request Person Generator' feature.
- CSH 2: Explain what existing features do, and why they are useful
    - The GUI reflects this heuristic. There is a prompt at the top explaining how to use the program and its purpose. There is also an explanation of the required format for input CSVs which improves reflection of this heuristic.
- CSH 3: Let people gather as much information as they want, and no more than they want
    - The GUI reflects this heuristic in that only the currently requests results will display to the text box and users are able to refresh this box with new results as many times as they would like. The two requesting services are also clearly divided on the GUI so only those looking to use Person Generator would read the text underneath the Generate button. The GUI is also intuitive and some users may not even need to read the instructions unless they want the extra information.
- CSH 4: Keep familiar features available
    - The Sprint 4 update only added features to below the Generate button in the GUI which kept most of the original/familiar features the same. The program can be used as it normally would before the Sprint 4 additions.
- CSH 5: Make undo/redo and backtracking available
    - The GUI does not reflect this heuristic, because there are no undo options built into the GUI. However, the flow of the program using dropdown menus for easily changeable input and constant refreshing of the text box with current results somewhat removes the need for an undo/backtracking option.
- CSH 6: Provide ways to try out different approaches
    - This heuristic is reflected in the GUI, because the program offers the option to select a CSV file for input as an alternative path to selecting input from the dropdown menus. The program can also be run via the command line and there are clear instructions within the GUI on how to do that.
- CSH 7: Communicate the amount of effort that will be required to use a feature
    - This heuristic is reflected, because the requirements/effort for each requesting service are clearly explained in a text prompt above their respective input options. The 'Select a CSV file' button and the instructions above both indicate that a CSV file is necessary to use that feature.
- CSH 8: Provide a path through the task
    - The text prompt clearly describes the path through the task with the optional entry points of using the dropdown menus, a CSV file, or a person generator request. It also includes an explanation of where to look for the results so that the entire path through the task is described to the user.
- CSH 9: Encourage mindful tinkering
    - This heuristic is not reflected in the design as there is not much room for tinkering and the flow of the program is linear. However, the additional options to request from Person Generator and select or export a CSV do allow for some tinkering. Although the buttons are self-explanatory and there are instructions, there are no popup windows to confirm any button clicks.

VIII. Running the Code and Known Issues

The code can be run normally and starting it this way will load the GUI. The program can also be run via the command line using 'python population-generator.py input.csv' where input.csv is the desired input CSV file to process using the format input_year, input_state. This method will bypass the GUI and a CSV file titled output.csv will be created within the same directory as Population Generator containing the generated information.

As for known issues, there are none aside from a memory warning when using Person Generator that is related to the Person Generator code.