Population Generator Project Series 1

System Requirements for distributed population generator microservice project

Anthony Corton

College of Engineering

Oregon State University

CS-361-400-W2021

Table of Contents

1 Introduction

This section of the Software Requirements Specification (SRS) will cover a brief introduction into the purpose of the purpose of the population generator microservice and the scope of the population generator microservice. The project and its relevant documentation including instructions, rubrics, and other material belong to Oregon State University and are subject to its rules.

1.1 Purpose

The purpose of the Software Requirements Specifications (SRS) includes various reasons one of which is to introduce the name and specifications of the population generator microservice through creating the document itself. The other reasons include defining the scope of the population generator microservice and setting forth the requirements for it as well in the form of non-functional requirements. There are also user stories set forth in the SRS to help define the purpose of the microservice.

1.2 Scope

The scope of the population generator microservice include generating the number of people within a particular location and year. The inputs include a location (state) and a respective year (2005-2019) and with these inputs the population generator microservice will be able to output actual population data for people within that state for that particular year. The population generator uses real data gathered from the ACS annual data via the census.gov website. The microservice can retrieve data via an API call and is developed in the Python programming language. Finally, the population generator microservice can communicate with other microservices such as the person generator microservice and the content generator microservice output from the content-generator microservice can be read via the terminal while output can be exported to the person generator microservice for feeding.

2.1 User Stories

This section of the Software Requirements Specification (SRS) will cover five user stories which reflect the requirements of INVEST in the “As a” format.

- As a non-profit organization, I would like to put in a city and get a list of the breakdown of how many people in that city are reasonably part of a minority group so that I can better scale the organization to help disentranced groups within that city.

- As a Population scientist, I would like to put in a state and year to get the population amount for that respective year and state.

- As a data analyst, I would like to enter a csv file with states and years and have a csv file outputted with the population data for all the states and years on my input csv file.

- As a practicing and licensed geriatrician, I would like to put in a zip-code and get a detailed breakdown of how many people in that area are reasonably 50 or older for a population size of 500 so that I can become more informed about possible potential clients when looking for areas to open up a new office.

- As a DevOps-engineer, I would like to have a population generator microservice be able to communicate with other microservices so that I can read output from them and communicate with other services.

- As a data analyst, I would like the option to gather population data for a particular state and year and have the option to export that data as a csv only when I require it.

- As an educator, I would like to be able to gather the population data for a list of states and years and have that information exported to a csv without having to interact with a user interface.

- As a startup looking for areas to expand into, I would like to be able to put in a city and get a detailed breakdown of how many people in a population size of 500 would

reasonably be women in that city, so that the board and I can decide on which areas would be better for growth as we look for more female hires

2.2 Non-functional Requirements

This section of the Software Requirements Specification (SRS) will cover three non-functional requirements.

- Quality Attribute: Interoperability
  - Non-functional requirement: The population generator microservice must be able to communicate and use output from at least one other classmates microservice that they create.

  - Justification: Per the project requirements outlined in the course the project needs to be able to communicate with another students project to get full credit which is why interoperability is essential as the microservice needs to be able to communicate by using output from another students microservice.

- Quality Attribute: Portability
  - Non-functional requirement: The population generator microservice must be created in Python programming language and must run on Windows and Linux operating Systems.

  - Justification: Python is what the class was notified the program would be made in and it is also the new language for the program. Also Windows is a popular OS operating system and the universities online engineering servers provide a testing environment for running program on in Linux which is why it is important to get Windows and Linux right since it provides an equal playing field for the users to run the microservice.

- Quality Attribute: Performance
  - Non-functional requirement: The population generator microservice must have a run time that is no longer than 5 minutes.

  - Justification: If the performance is poor and takes some significant time that will make debugging, running, and grading the program much more difficult for the developer, user, and grader. Which is why the performance must be relatively fast and no longer than 5 minutes.

2.3 Current – Planned Functional Requirement

For the service currently there are only a few functional requirements implemented. The functional requirements currently implemented are allowing for users to pass an input csv file with states and years at run time and get an output csv file with those states and years with their respective population. The data for the population is real data from census.gov. Users can also choose if they want to just display their output or export their output to a csv file. Another thing user can do is communicate via other microservices although the implementation is limited to known services the person generator microservice and content generator microservice.

For planned functional requirements those include allowing users to use a city as input and get back the population data for that city. Allow users to get a detail breakdown how the percentage of the population that are minorities. Another planned feature is allowing users to see a detailed breakdown of the percentage of females or males in a particular city or state.

2.4 Current – Planned Non-Functional Requirement

For the service currently communication is allowed and working. The method of communication is via reading and exporting csv files to classmate's microservices which are different from the population generator. As of this moment two microservices can be communicated with via a FEED and CONSUME relationship which is the person-generator microservice and the content-generator microservice respectively. The population-generator microservice also currently runs on Windows, Linux, and Mac systems. In conclusion the population-generator microservice has a run time of less than 2 minutes as well.

There are a few planned non-functional requirements one of which is for reducing response time to a few seconds or less in all environments. Another planned non-functional requirement is to increasing interoperability so that the microservice can be integrated with all the microservices generators from the course and not just the two that are already integrated.

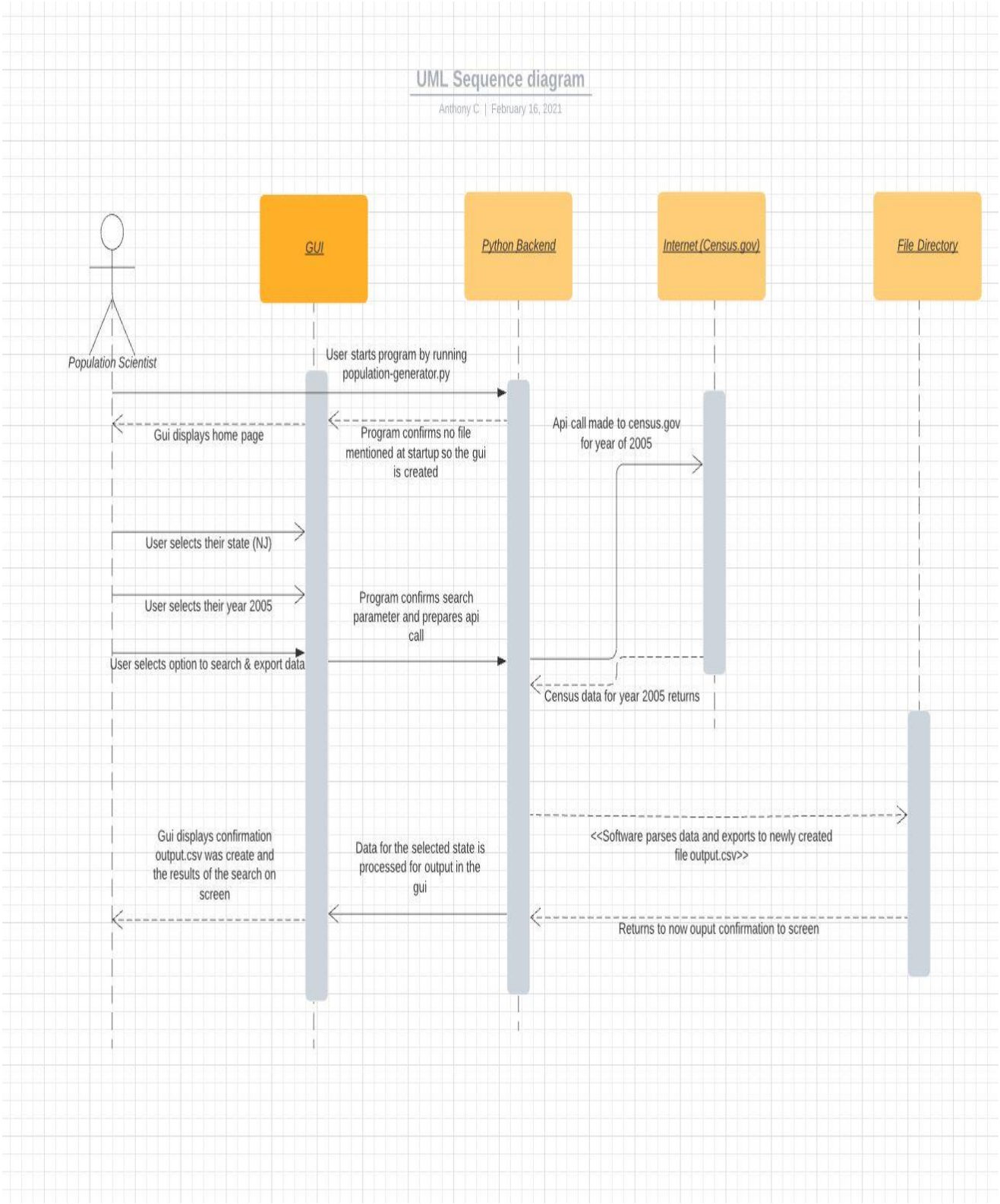
2.5 High-Level Architecture

The high-level architecture of the population-generator microservices involves using distributed microservices developed in Python 3. The microservice architecture allows for fault isolation and flexibility when using varies services such as the person-generator microservice and the content-generator microservice for example. For the GUI interface the library Tkinter was used with other libraries like urlib and request for making API calls.

2.6 Communication

The communication between microservices is handled by a system of csv files that will be outputted when a request is made or used as input to check for a response from a request. For example users can request data from a microservice they want to consume from such as in the case of the content-generator microservice which can be done by outputting a request csv to the content-generator microservice folder than checking for a response in the form of a csv file. The data received from the response can be consumed by reading and printing the data to console. For the feed portion of communication there is a button for checking for request from the person-generator microservice and if such a request if found currently selected data will be outputted to the person-generator microservice folder as a csv file to be read.

2.7 Code design (Sequence Diagram)



2.8 GUI Design

Heuristic #1 (of 9): Explain what new features do, and why they are useful

- I would say my code does not necessarily reflect the first heuristic and the reason is because I did not create any announcements that new features have been added to the microservice so a user won't be aware if they haven't noticed already that there are new features like communication between other microservices.

Heuristic #2 (of 9): Explain what existing features do, and why they are useful

- My design reflect the second heuristic in the aspect of explaining what a feature does and the benefit of using the feature by having the button blocks be explanatory in their purpose and action so a user knows for instance if they click export data as a csv it exports data as a csv when they click it.

Heuristic #3 (of 9): Let people gather as much information as they want, and no more than they want

- My design reflects the third heuristic in the aspect of allowing people to gather as much information as they want and no more than what they want. One example is that users can quickly see the contents of the microservice functions they are interested in and jump to them since all relevant functions are displayed on the left-hand side of the screen in an orderly fashion.

Heuristic #4 (of 9): Keep familiar features available

- My design abides by the fourth heuristic in the aspect that I keep familiar features like a drop-down menu available for users. The importance of the drop down menu feature is that it allows Abi, Pat, and Tim to keep using the microservice in a way that they expect it to work through selection.

Heuristic #5 (of 9): Make undo/redo and backtracking available

- My design does not really reflect the make / undo features defined by the fifth heuristic since there are no options to cancel outgoing request for communications for example. A couple of ways I can better implement this would be to allow for a button to cancel outgoing request or a button to undo the previously exported output.csv file.

Heuristic #6 (of 9): Provide ways to try out different approaches

- I feel my design does not accurately reflect the sixth heuristic and one way it could better reflect the sixth heuristic is to include a choose a question drop down menu so users can figure out how to proceed down certain paths in the microservice.

Heuristic #7 (of 9): Communicate the amount of effort that will be required to use a feature

- My design does reflect the seventh heuristic and the reason is would be because the way I formatted the buttons on the left hand side of the screen are reflected in an orderly fashion with the easiest options being listed up to while the more advanced options like microservice communication are listed down towards the bottom reflecting they will require more effort to use their features.

Heuristic #8 (of 9): Provide a path through the task

- I would say my microservice provides a path through the task since users can visually see which part of the process they are in through the order of buttons and pop-ups that occur during the function of let's say getting population data or exporting population data.

Heuristic #9 (of 9): Encourage mindful tinkering

- I would say my current design does not accurately reflect the ninth heuristic of encouraging mindful tinkering, but it could with a few changes. One such change would be adding a Are you sure notification box when a user goes to make a request or export data as a csv that way they are better aware of their actions particularly if they have clicked something while tinkering around.

3.1 How to run code

The code can be run in two different ways one of which is to specify a csv input file at runtime and the other is to just run the python file without specifying an input file at runtime. When the program is executing at runtime by specifying an input such as in the case of "population-generator.py input.csv" the GUI will not run and instead the population data for all the states with their respective years will be outputted to a file called output.csv without requiring any user interface or actions from the user themselves. The location of ouput.csv will be in the same folder directory as the program and the input.csv file must also be located within the same folder directory.

When the program is run without specifying an input.csv file such as in the case of "population-generator.py" then the GUI will automatically startup. When the GUI is up users can select a state and year through the two drop down menus on the left side of the screen. After a selection of state and year is made the user can choose either the button to submit the query or the button to submit the query and output results to a csv file either way the results will be displayed on the gui screen itself so the user can see.

The communication will be handled using the three buttons, request input from content generator, check for content output.csv, and check for request from person generator microservice. To facilitate data from the content generator the user must first click the request input button then wait for the user using the content-generator microservice to check for a request in their own folder and then output their data. Once the content-generator microservice has outputted its data the check for content ouput.csv button can be pressed to check for that data in the directory of the population-generator microservice and then display it to the console if found. The final button check for request from person generator microservice can be clicked to check for request and if a request is found in the directory of the population-generator microservice the program will automatically handle outputting the current selection of data to the person-generator microservice folder.

3.2 Known Issues

A known issue includes attempting communication with other microservices via the buttons when no such service exists in their required subfolders in the current directory. This will be fixed in the future to properly display a message to the GUI instead notifying the user to check the readme for the required file structure and programs for communication. Another know issue is attempting to gather data when the microservice is offline which can cause the program to be stuck running. To fix the problem with no connection in the future data can be downloaded and stored from the census site so that the service can be run offline and online without issue.