State Finder

State Finder Team: Alyiah Proctor, Emily Crabtree, Anne Nguyen, Darlyn Mendez

1. **Project Definition -**

   Career transitions, job searching, and location searching are stressful activities that one usually encounters at some stage in their lives. Our State Finder Web Application is meant to aid people in these processes by creating a tool that will provide users with information that can help their decisions. The tool will mainly be targeted to those who are looking to move to another state, college students who want to see the locations that their future career could lead them to, people who are looking for a career change, or anyone that is curious about where an occupation could allow them to live in the United States. Our State Finder Web Application attempts to increase the confidence in the career choices and location choices one may make.

   The goal of the project is to create a web application that users can rely on to find the best locations for them and their families to reside in based on their potential career income. Users should be able to input a number of career choices and family info, and the site will populate with the metro areas they can afford, as well as info on each state and further infographics regarding state costs. The team hopes to create a Web Application that is reliable, accurate, easy to use, and a great resource for our users.

   The general structure of our application will be a Python - Flask framework and Heroku to aid in the development process. Possible Python libraries that we will use include GeoPandas, Matplotlib, and Flask login. The frontend will be made with HTML, CSS, JavaScript, and Bootstrap. Additionally, the data will be stored in a SQL server database that is hosted in Azure under a student subscription.

   Our goals for the State Finder Web Application will be achieved through continuous team member collaboration on a Discord server, bi-weekly Scrum meetings, web app implementation, continuous research, troubleshooting, and great teamwork.

2. Project Requirements

The functional requirements for our project is the production of a heat map to indicate livable areas based on user input. The map should change colors indicating the livability of a specific area, according to a provided color key. Furthermore, users will be able to access state details, such as state population and average salary for the user's indicated occupation and job level. The way in which this will function is still undecided, but will either be available when the user hovers their cursor over a specific area or will automatically generate as text beneath the produced heat map.

When the web application loads, it will load to the homepage. On the top of the page, there will be the website title, a button that redirects the user to an "About Us" page, a button that redirects the user to a "Contact Us" page, and a button that redirects the user to a "Resources" page. In the center of the homepage will be a brief description of the web application and a button that says "Find A State", which will redirect the user to a webpage where they enter their information.

On this page, the user will be able to indicate their occupation, job level (entry level, intermediate, or senior), whether they are married, whether their spouse is employed, and how many kids they have (up to 3), all in various drop down menus. All of the dropdown menus must have a selected option in order for the heat map to generate. Then, the user will click a button to submit their options and will be automatically redirected to the heat map and area details. By utilizing dropdown menus and requiring that all menus have a selection, the possibility of user input error is greatly reduced and the usability of the web application is increased.

System hardware that is required in order for the user to use our web application is a CPU, a monitor or screen to display the web application's interface, a keyboard, and a mouse. System software that is required for use of the web application is an operating system on their device, a graphics card, and a web browser. The database requirements for the web application is a connection to our SQL database. The SQL database is where all the livable wage data per area is stored, as well as the available occupations and their respective salaries. The user will not require connection to this database and nothing will be stored on their local device, but they will require an internet connection so a connection to the database can be made on the web application's backend.

Security requirements for our web application is a secure login portal that will require the user to provide a unique username and password. Each user will be given a unique ID and their login credentials, as well as their personal information required for the State Finder, will be stored in our database. The database will be secure and users will not have access to information on other users, nor will they be able to directly access the salary or livable wage data.

3. Project Specification –

The statefinder has a major focus area in data analysis. The project involves manipulating and making sense of data as well as justifying those choices and drawing insightful conclusions. Our goal with the app is to consolidate data from metro areas and create a helpful one stop shop for living decisions. There is also a specification in database management and front end user design. All data used in the app will be housed in a backend server, and our UI will need to be flexible and usable to communicate with this server.

For the development of the web app the flask framework will be used to create the web aspect of the app and Azure SQL server will be the database framework for the backend. We will make changes to the Azure SQL server database in Azure Data Studio. We have all been added as collaborators to the database, so that the configuration file will work for the team for the database connection. As Python is our primary development language, the pandas, plotly, and matplotlib libraries will be used to simplify the tasks. HTML and CSS will be the primary languages used for the UI development and works hand in hand with flask. The Bootstrap front-end framework will be used to make the UI visually appealing. Any IDE is acceptable to use as long as it supports the python language. However, Visual Studio Code is the IDE used by

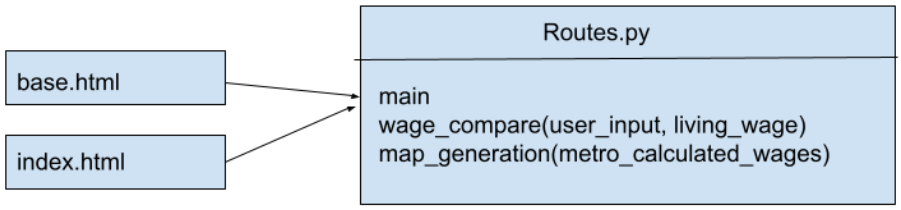
most of the development team. We will have a Google Cloud Console Setup in order to get an API key, so that we can access the Google Maps API. We will also have a Firebase Console Setup in order to have a Firebase Realtime database to store our users login and personal information.

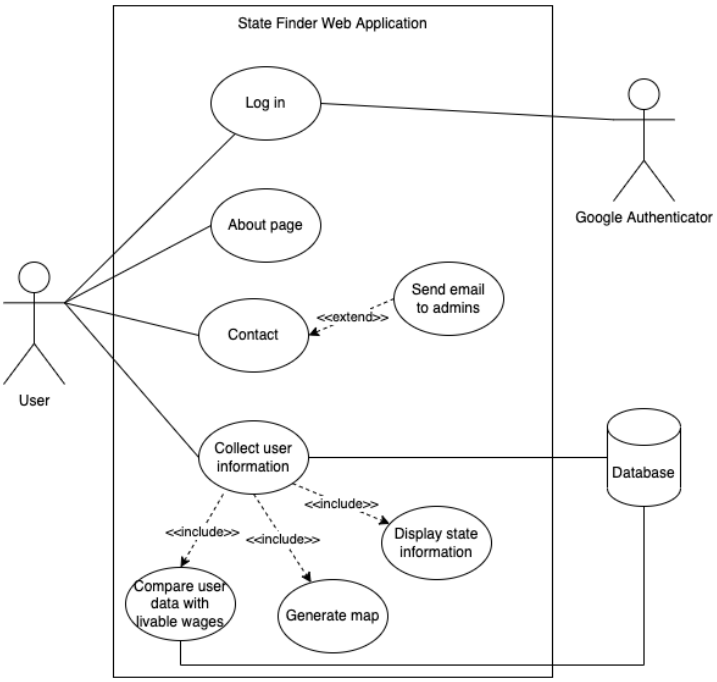
The project does not have a specific device specification. It will be built to run on most devices that can access the web through a web browser. Desktops,laptops,tablets,phones,etc.

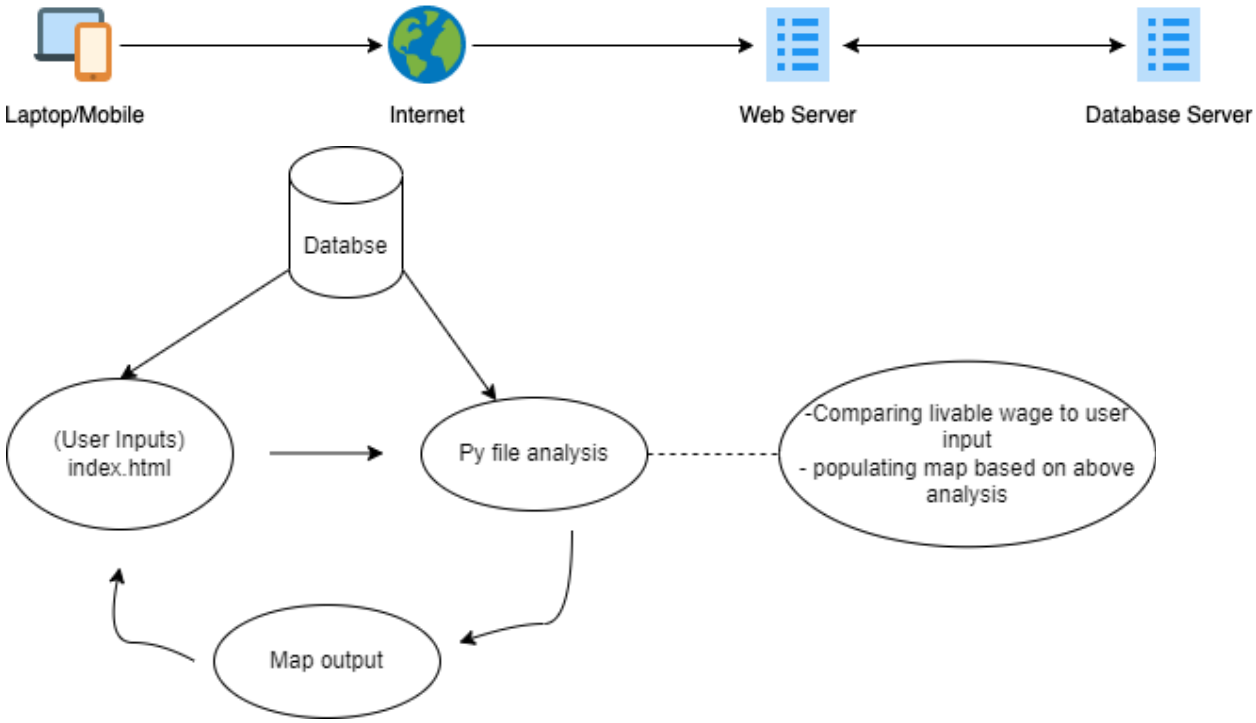
Overall the statefinder belongs in a genre of everyday web applications or sites that require very little demands from the user.

4. System – Design Perspective –

From a design perspective, our project can be broken down into three main subsystems. These subsystem are the Database subsystem, which is designed to separate livable wages by Metropolitan area, state, number of children, marital status, and whether the spouse works, and the average salaries for a large variety of occupations for ease of comparison in the code; the User Interface subsystem, which consists of the HTML files that create an aesthetically pleasing and user friendly interface; and the Python subsystem, which consists of the various Python files that are used to compare the user's information to that in the database.







Some design choices that we decided to implement in our project are the use of drop down menus for the user's information collection, which creates a more user friendly experience and lessens the chance of error. We also opted to produce a heat/choropleth map to display the results of the State Finder as it provides an easy to understand and general representation of where the user would be able to live in the United States based on their provided information. Lastly, we decided to use a Google authenticator for sign in to our web application, which will give the user a simpler way to sign in and negate the need for a database dedicated to storing login credentials.

As for sub-system communication, it will begin by the Python subsystem communicating with the Database subsystem to pull the data on livable wages and occupation salaries from the database into the necessary Python file. Next, the UI subsystem will communicate with the Python subsystem so the collected user information can be sent to the necessary Python file. Then, the Python subsystem will compare the user data with the Metropolitan livable wage data and the occupations salary data. Lastly, the Python subsystem will communicate with the UI subsystem to send the user's compared data back to the necessary HTML file, which will then be used to populate the choropleth map.

Overall, our system will communicate in the following way. Our main HTML form will collect the user's information and communicate that data to the main Python file. The Python file will communicate with the Azure SQL database to collect the information on livable wages in the Metropolitan areas and the provided occupation's salaries. The user's data will be compared with the data pulled from the database and then sent to the HTML file responsible for populating the choropleth map.

5. System – Analysis Perspective

From an analysis perspective the state finder has three main subsystems. One that calls to the database to retrieve all the data needed, another that compares the user input to the livable wage data, and the last subsystem that populates the map before it is passed to the html files. All of these subsystems will be their own python functions with the exception of the database call as it only needs to occur once. This adds flexibility as it will allow the site to effectively and efficiently run if the user decides to continuously change their selected inputs.

Our database consists of four tables: Livable_Wages, Occupations, State_info_population, and State_info_housing. The data dictionaries for all are shown below.

Data dictionary for tables 1 Adult, 2 adult 1 working, and 2 adults both working

Variable	Definition	Datatype
Metro Area	Name of Metropolitan Area	varchar
State	Abbreviation of State	char
0_kids_year	Annual wage for 1 Adult, 2 adults 1 working, or 2 adults both working respectively with 0 kids	float
1_kids_year	Annual wage for 1 Adult, 2 adults 1 working, or 2 adults both working respectively with 1 kid	float
2_kids_year	Annual wage for 1 Adult, 2 adults 1 working, or 2 adults both working respectively with 2 kids	float
2_kids_year	Annual wage for 1 Adult, 2 adults 1 working, or 2 adults both working respectively with 3 kids	float

Data dictionary for occupations

Variable	Definition	Datatype
OCC_TITLE	Title of the occupation	varchar
ANNUAL_MEAN	Average annual mean for the occupation across all states in the US	float
A_PCT10	10th percentile of the average annual mean for the occupation across all states in the US	float
A_PCT25	25th percentile of the average annual mean for the occupation across all states in the US	float
A_PCT75	75th percentile of the average annual mean for the occupation across all states in the US	float

State_info_population

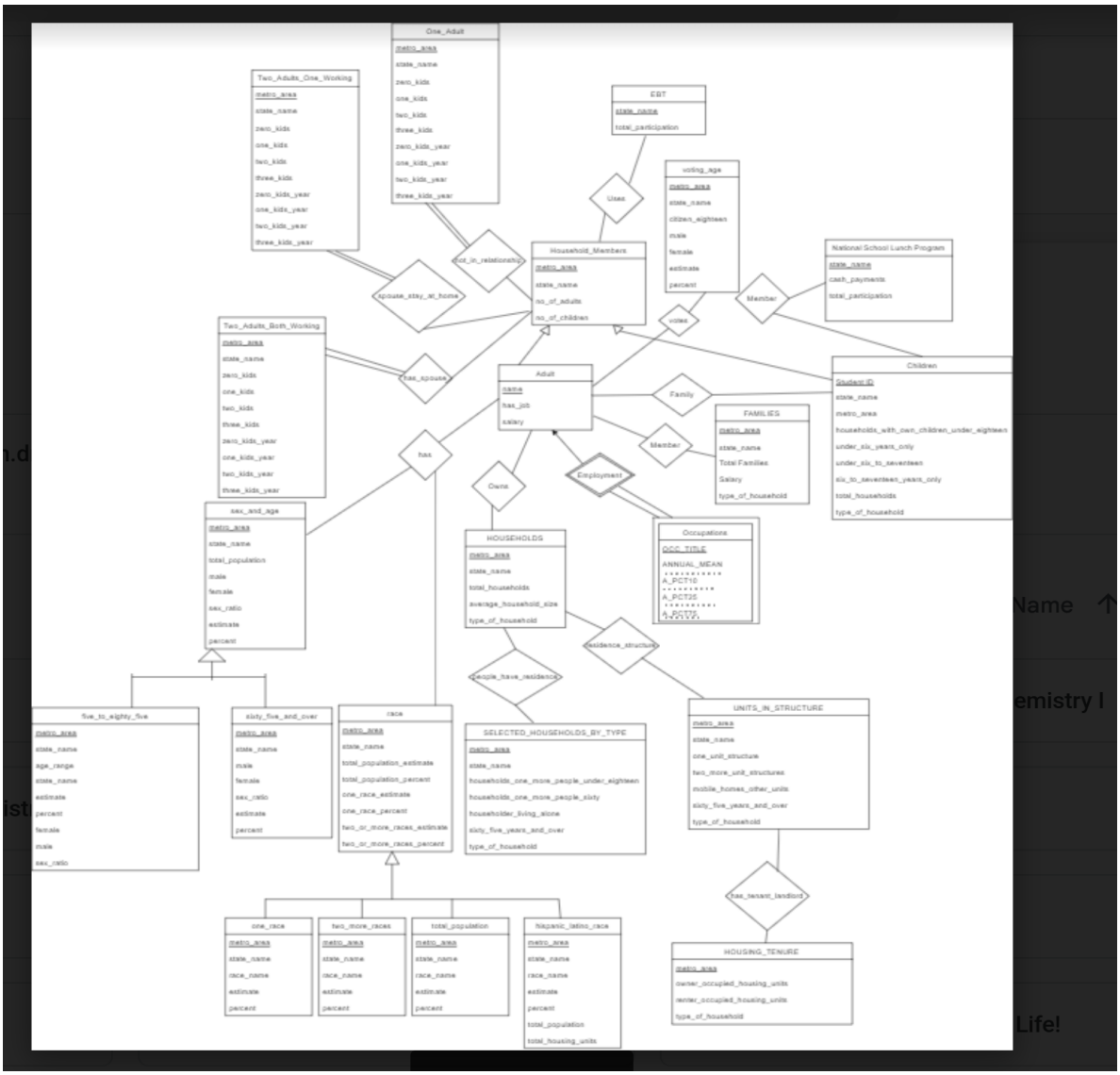
- Due to the number of columns and rows exceeding 90, a data dictionary can not be made. A simplified explanation of the table is as follows. The column headings are the names of the metro areas while the rows are demographic data such as percentage of age groups, sex ratios, and race percentages of the total population. Datatypes are varchar and floats.

State_info_housing

- Due to the number of columns and rows exceeding 20, a data dictionary can not be made. A simplified explanation of the table is as follows. The column headings are the names of the metro areas while the rows are housing demographic data such as the age demographic in households, the number of housing units occupied, number of units total in the area. Datatypes are varchar and floats.

As the layout of our project is very simple, our overall system has a runtime of Big O of n. The program should only have to run through all methods once to receive the required results. If the user calls n times it will run n times.

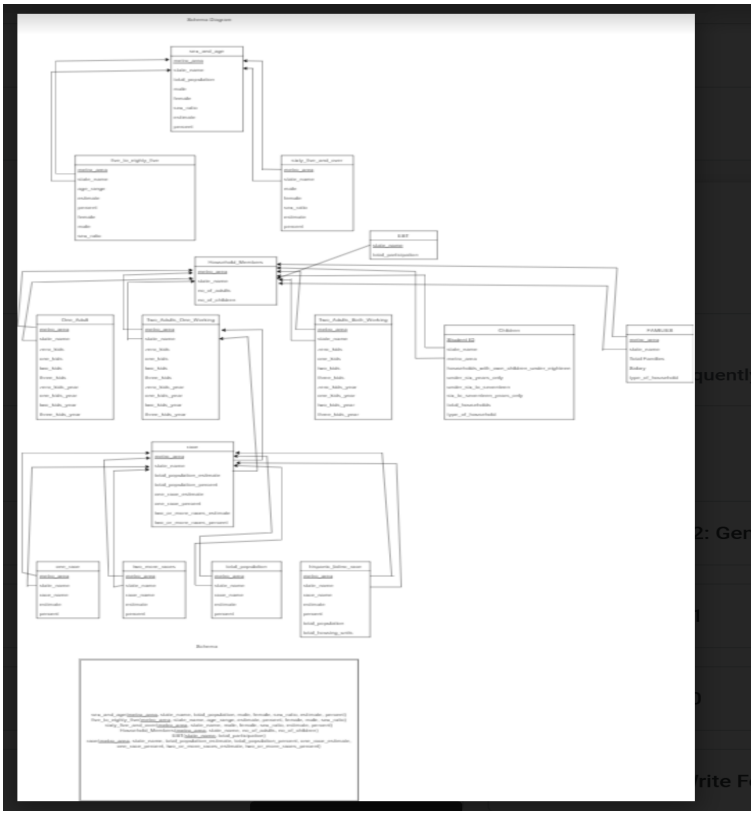
ER Diagram



In this entity relationship diagram we tried to find any relations between the different sources of data. We identified similar columns of data through foreign key constraints in order to find connections between the datasets. One of the datasets which is the state info population dataset had to be separated into tables because the layout of the data needs to be cleaned up. Some new

variables were added in order to complement the data and to reduce any confusion when writing queries for the data. Most of the tables have a relation to the Adults table which can be seen as the center of the ER diagram. Some of the tables require total participation because people have to have an age. There is disjoint generalization for race because you are part of one race or two races, but not be a part of both groups at the same time.

Schema and Schema Diagram



Most of the tables have a foreign key constraint on the attributes state_name and metro_area which shows that these datasets have these columns in common. Some of the attributes are the same in multiple tables, but they are reflecting different data so foreign key constraints do not apply. The Schema and the Schema Diagram does not show us much compared to the Entity Relationship Diagram, but it helps give us a high level overview of where our foreign key constraints are. We also did some quick exploratory data analysis of some of the datasets by using Pandas Profiling to generate an HTML EDA report which can help gain an idea of where to focus our cleanup efforts on. We noticed that there are some missing values in one of the datasets and that there is a high cardinality. This means that we have to search where those missing values are coming from.

Pandas Profiling Report

Overview

Overview | Alerts 2 | Reproduction

Dataset statistics

Number of variables	5
Number of observations	1130
Missing cells	32
Missing cells (%)	0.6%
Total size in memory	44.3 KiB
Average record size in memory	40.1 B

Variable types

Categorical	1
Numeric	4

Pandas Profiling Report

Overview

Overview | Alerts 2 | Reproduction

Dataset statistics

Number of variables	10
Number of observations	383
Missing cells	0
Missing cells (%)	0.0%
Total size in memory	30.0 KiB
Average record size in memory	80.3 B

Variable types

Categorical	2
Numeric	8