Software Design Description (SDD)

CMSC 447 Group 4

Project Vesta

1 Scope

1.1 Identification

This document refers to the software package known as Vesta.

1.2 System overview

Vesta is a web based application that will be utilized by an average internet user. This web application is intended to allow users to acquire potential living locations based on abstract specifications regarding the area. To garner said information, the user will specify quality of life attributes desired within a defined radius. These characters may include meteorological data (ie average temperature, average weather conditions), geographical information, etc. The application will compare the inputted information to specified open source databases to determine the locations that fit the criteria. Once the locations have been determined by the software on the back-end, they will be displayed to the user.

1.3 Document overview

This document will specify the process and specifications in which Vesta will be designed.

2 Referenced documents

Section 6 references the requirements outlined in the Software Requirements Specification.

3 CSCI-wide design decisions

The software package will accept a dictionary of parameters related to cost of living, including but not limited to the following:

- Property Values
- Minimum and maximum temperatures
- Political information
- Population information
- Climate information

The frontend system of the software package will pass these parameters from the user by means of a web interface written in JavaScript, HTML, and CSS, to through a backend daemon running on an Amazon Web Services (AWS) Lightsail instance written in Python to a MySQL database.

- The frontend system will use an AJAX request called from the JavaScript code to pass a dictionary of parameters in the JSON format to the backend.
- The backend Python daemon will use the Flask package in order to host a local server which only purpose is to communicate with the site frontend.
- The backend Python daemon will then make queries to a MySQL database running on the same AWS instance.

The software package will use the Apache2 HTTP Web server project to host the website and the related frontend code.

- The Apache2 web server will use a modified proxy configuration file in order to properly forward AJAX requests from the frontend to the backend.
    - Specifically, HTTP GET requests on ports 8080-8090 will be forwarded directly

to the Python Flask daemon.

The software package will output one or many markers to the screen by means of the Google Maps API, and will display information for each city to the user

- The backend Python daemon will query a MySQL database to find locations which will match the parameters entered by the user.
- When it finds a list of suitable locations, it will return these as a simple JSON array to the frontend

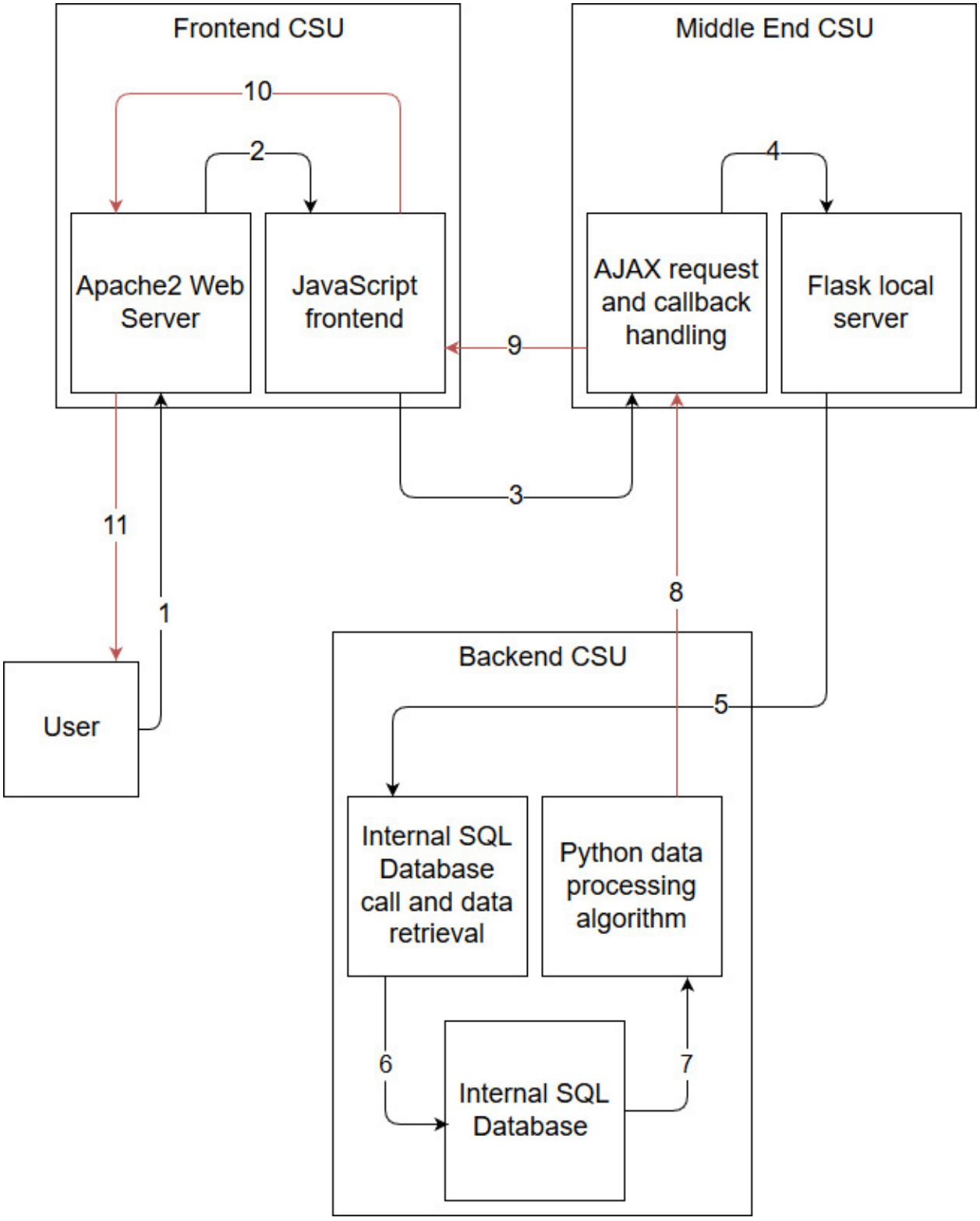
4 CSCI architectural design

4.1 CSCI components

CSU and ID#	Description	Development Status	Planned Resources Utilization (Scale of 1-10, with 10 being the most resource intensive)
1.0 Frontend CSU	Handle all frontend communications with the user and web server functionality	New Development / Existing Software	4 (Low)
1.1 Apache2 Web server	Display website to user and allow for site navigation and functionality	Existing Software	2 (Low)
1.2 JavaScript frontend	Provide advanced GUI elements and middle-end integration	New Development	2 (Low)
2.0 Middle end CSU	Handle communications between frontend and backend CSUs	Existing Software / New Development	1 (Minimal)
2.1 AJAX request and callback handling	Package user queries into JSON format for	New Development	1 (Minimal)

	transmission to backend		
2.2 Flask local server	Receive data from frontend CSU and parse into format that the backend CSU can use	Existing Software / New Development	1 (Minimal)
3.0 Backend CSU	Handle querying of internal SQL database and location intersection	New Development	8 (High)
3.1 Internal SQL data retrieval	Responsible for querying the internal SQL database to get usable and relevant data	New Development	4 (Medium)
3.2 Python data processing algorithms	Correlate retrieved data into a list of locations to return to the user	New Development	4 (Medium)

4.2 Concept of execution

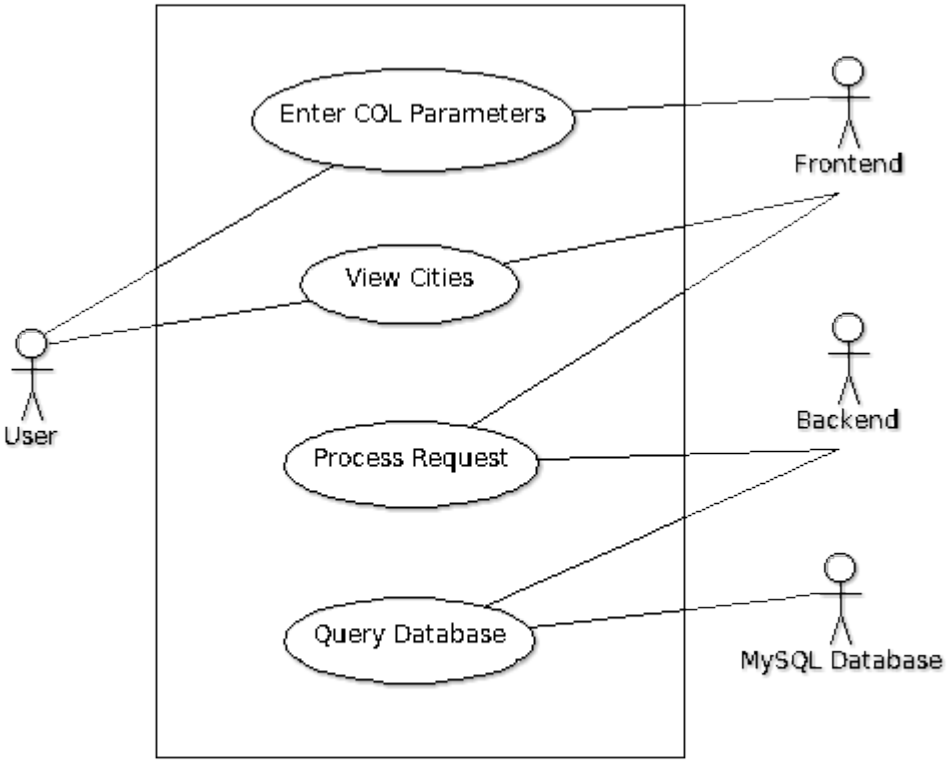


Black arrows describe processes and data flowing away from the user, while red arrows describe processes and data flowing back to the user.

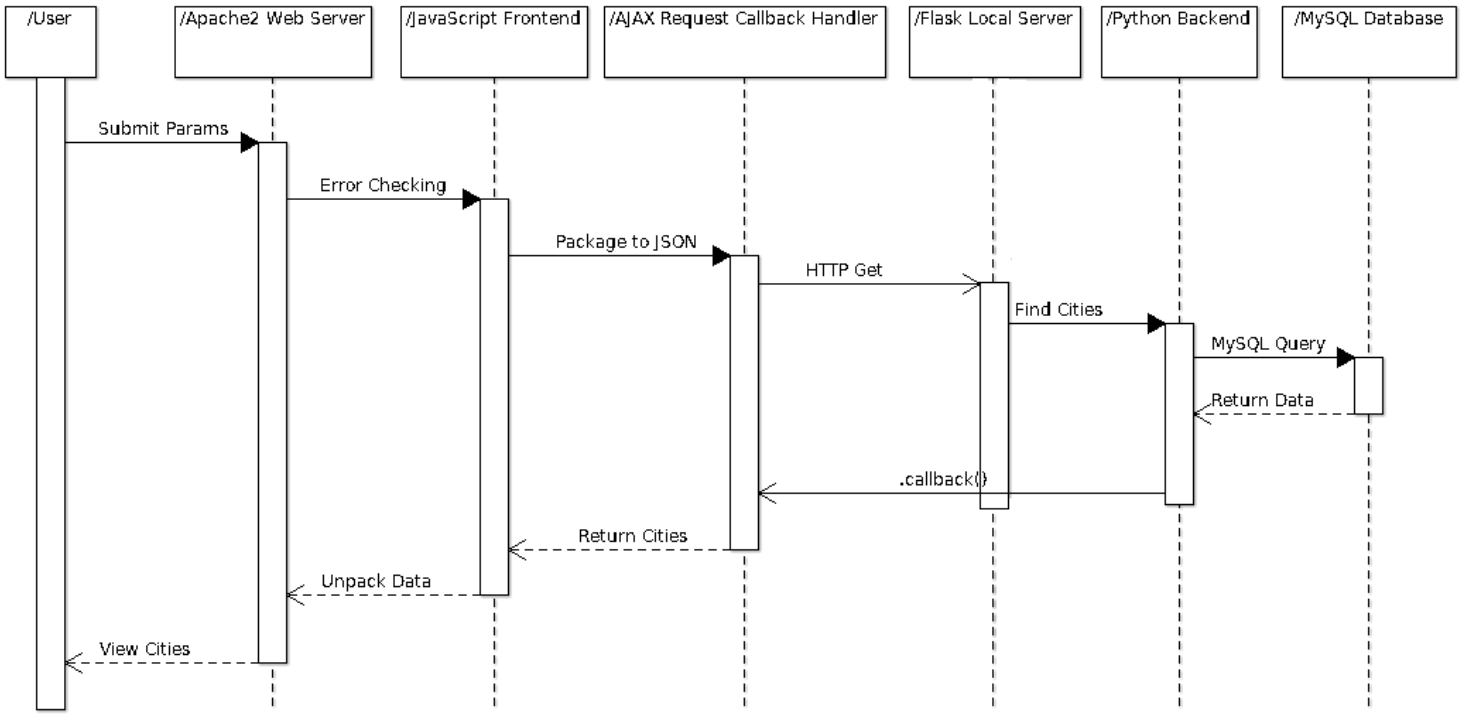
1. The user enters desired parameters on the website interface and submits them.
2. The Apache2 web server passes the parameters to the JavaScript frontend code, which checks for any glaring input errors.
3. The parameters are passed to the Asynchronous JavaScript and XML (AJAX) request creation code, which packages them into a JavaScript Object Notation (JSON) dictionary.
4. The AJAX request then sends the parameters to the waiting Flask server (a basic communications server framework for python) by means of a HTTP GET request.
5. The parameters are parsed by the local Flask server into something the rest of the Python backend can understand.
6. The Python backend software reaches out to the internal SQL database in order to get a list of locations that match the user's parameters.
7. The relevant data is returned to the waiting Python backend, which translates the information into a JSON dictionary.
8. The data is passed back to the AJAX request by means of the .callback() function.
9. The data returns to the rest of the JavaScript code and is unpacked.
10. The unpacked data returns to the rest of the frontend system, which formats it properly and prepares to display it to the user.
11. The data is displayed on the web interface to the user in an easy-to-read format and waits for further instruction.

4.3 Interface design

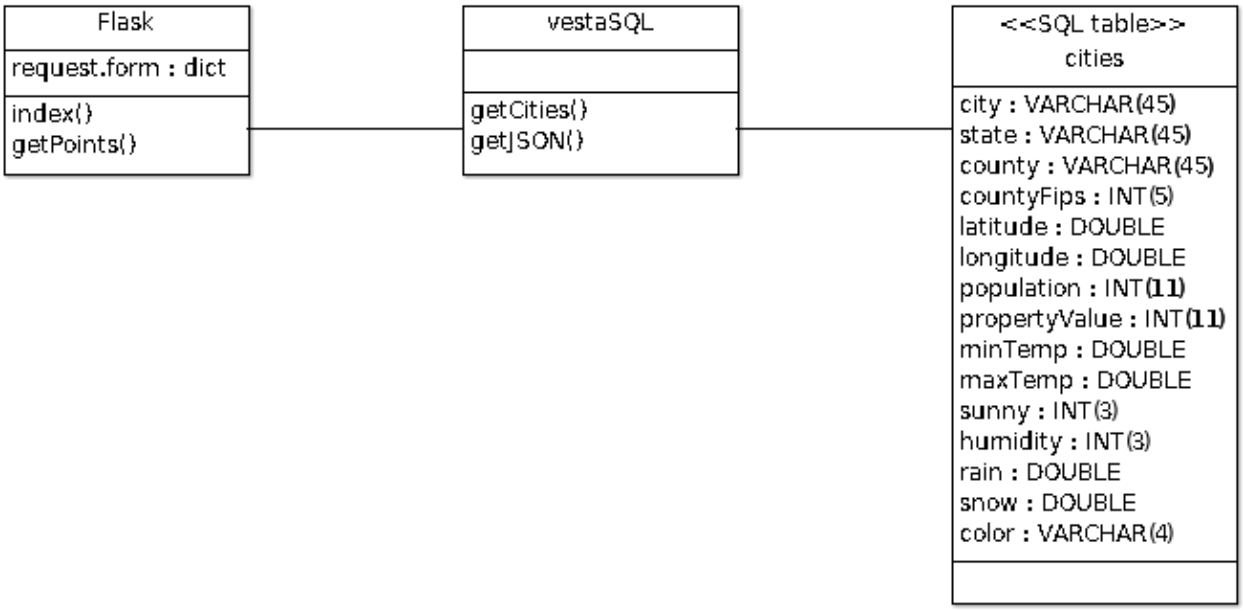
Use Case Diagram



Sequence Diagram



Class Diagram



5 CSCI detailed design

1.0. Frontend CSU

This CSU encompasses the web interface for the software and the advanced functionality for the website. The web server used for the software package is the Apache2 Web Server Project. The software package will use the Bootstrap GUI framework in order to provide better visuals and modular GUI constructability. The map system of the GUI will be backed by the public Google Maps API, in which a simple map is requested that can scroll and zoom around the world. This map will be centered on the United States and will have the capability to zoom to areas of interest returned by the system. This CSU will have multiple inputs related to quality of life that the user can input to get a set of results back. A detailed list of these input parameters can be found in section 3.2.1 of the SRS.

The frontend code is a mixture of JavaScript, HTML, and CSS. These are all very common website creation languages and as such they will be easy to build the system with.

2.0. Middle End CSU

This CSU describes all the code and networking functionality related to the communication between the frontend and backend CSUs. As described in the diagram above, an AJAX request generator embedded in the JavaScript frontend will generate a local HTTP GET request to the server itself, which will be handled by a flask Python daemon running in the background. Once the data processing is complete by the backend, the flask daemon will

repackage the returned data into an HTTP response and send it back to the JavaScript frontend.

Flask will be used for the Python daemon because it is versatile, simple to use, and there is a lot of quality documentation on the internet for what our software package is exactly trying to do. An AJAX request generator will be used to communicate with the Python daemon because it is the simplest way to make an HTTP GET request in JavaScript code.

3.0. Backend CSU

This CSU contains all of the Python code responsible for parsing the input from the flask daemon, reaching out to the internal SQL database for relevant information, and processing the responses of those requests. This CSU will take as an input a Python dictionary of quality of life parameters, and will return a list of cities, in a JSON string format, that match those parameters. The backend will query an internal SQL database that has been populated from public database information. Once it has gathered all of the required information, it will convert the list of cities to a JSON string. It then returns this string back to the front end for the user to view.

Python will be used for the backend as it is an easy-to-write scripting language that can process large amounts of data efficiently and quickly. It is also easy to write HTTP requests in Python using the requests library, which is necessary in order to communicate with external databases to get the required locations. MySQL will be used as the internal database because is is relatively simple to use and set up and it is free to use.

6 Requirements traceability

The following chart shows which software unit will satisfy each requirement from the Software Requirements Specification.

Requirements	**Software Unit**
3.2.1.I	1.2
3.2.1.II	3
3.2.1.III	1.2, 3
3.2.1.IV	1.2, 3
3.3.1	3.1
3.9	1, 2, 3
3.10.1	1, 2, 3
3.10.2	1, 2