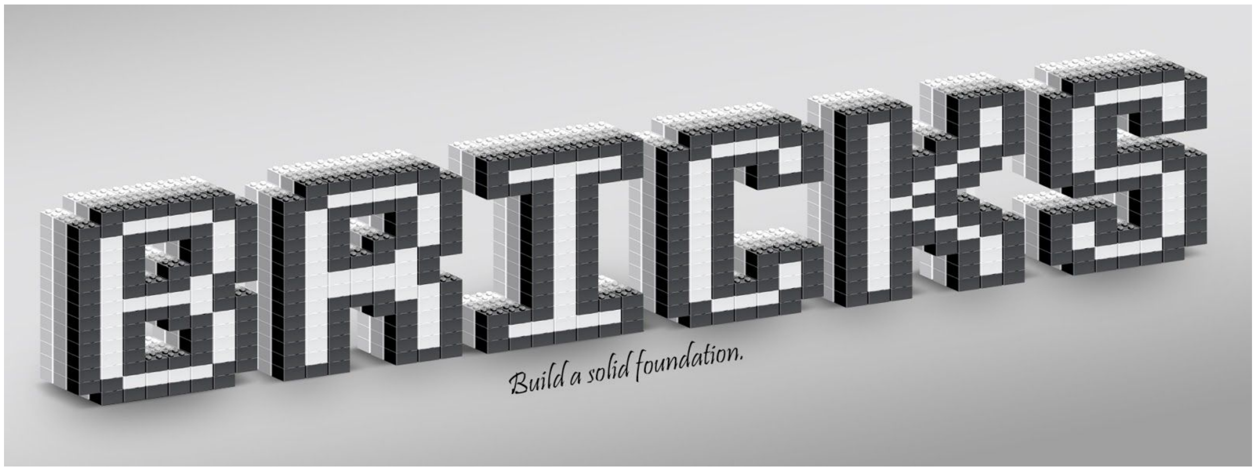
CMSC 447: Software Engineering I October 23, 2017

System Design Document

The Redistrictinator

Client: Geoffrey Weiss



BRICKS

Benjamin Jeremenko

Jacob Philip

Sumanth Neerumalla

Zachary Elliott

Francis Kato

Nathaniel Fuller

Table of Contents

	Page
1. Introduction	
1.1 Purpose of This Document	2
1.2 References	2
2. System Architecture	
2.1 Architectural Design	3
2.2 Decomposition Description	4
3. Persistent Data Design	
3.1 Database Descriptions	5
3.2 File Descriptions	5
4. Requirements Matrix	6
Appendix A – Agreement Between Customer and Contractor	7
Appendix B – Peer Review Sign-off	8
Appendix C – Document Contributions	9

1. Introduction

1.1 Purpose of This Document

The purpose of this document is to provide a description of the design of *The Redistrictinator* in order to allow for software development to proceed with an understanding of what is to be built and how it is expected to built. It also provides information necessary to provide description of the details for the application being built. The intended audience of this document is primarily for all parties directly involved in the development of *The Redistrictinator*.

1.2 References

I. (n.d.). Retrieved October 19, 2017, from https://bl.ocks.org/mbostock

II. Olson, B. (n.d.). B-Districting. Retrieved October 19, 2017, from http://blog.bdistricting.com/2016/

III. Data.gov. (n.d.). Retrieved October 19, 2017, from https://www.data.gov/

IV. Amazon Web Services (AWS) - Cloud Computing Services. (n.d.). Retrieved October 19, 2017, from https://aws.amazon.com/

V. Cloud Condo - Web Application Architecture. (n.d.). Retrieved October 23, 2017, from http://codecondo.com/web-application-architecture/

VI. NCZ Online. (n.d.). Retrieved October 23, 2017, from https://www.nczonline.net/

VII. Gliffy Online. (n.d.). Retrieved October 23, 2017, from https://www.gliffy.com/

VIII. American FactFinder - United States Census Bureau. (n.d.). Retrieved October 23, 2017, from https://factfinder.census.gov

IX. Flask - Web Development. (n.d.). Retrieved October 24, 2017, from http://flask.pocoo.org/

2. System Architecture

2.1 Architectural Design

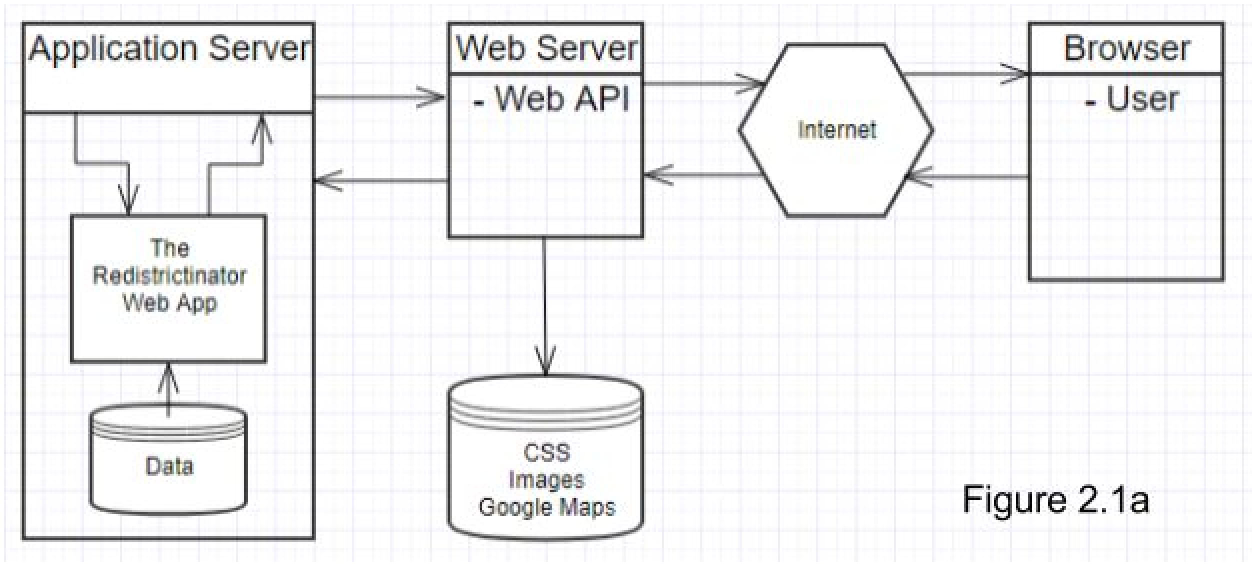
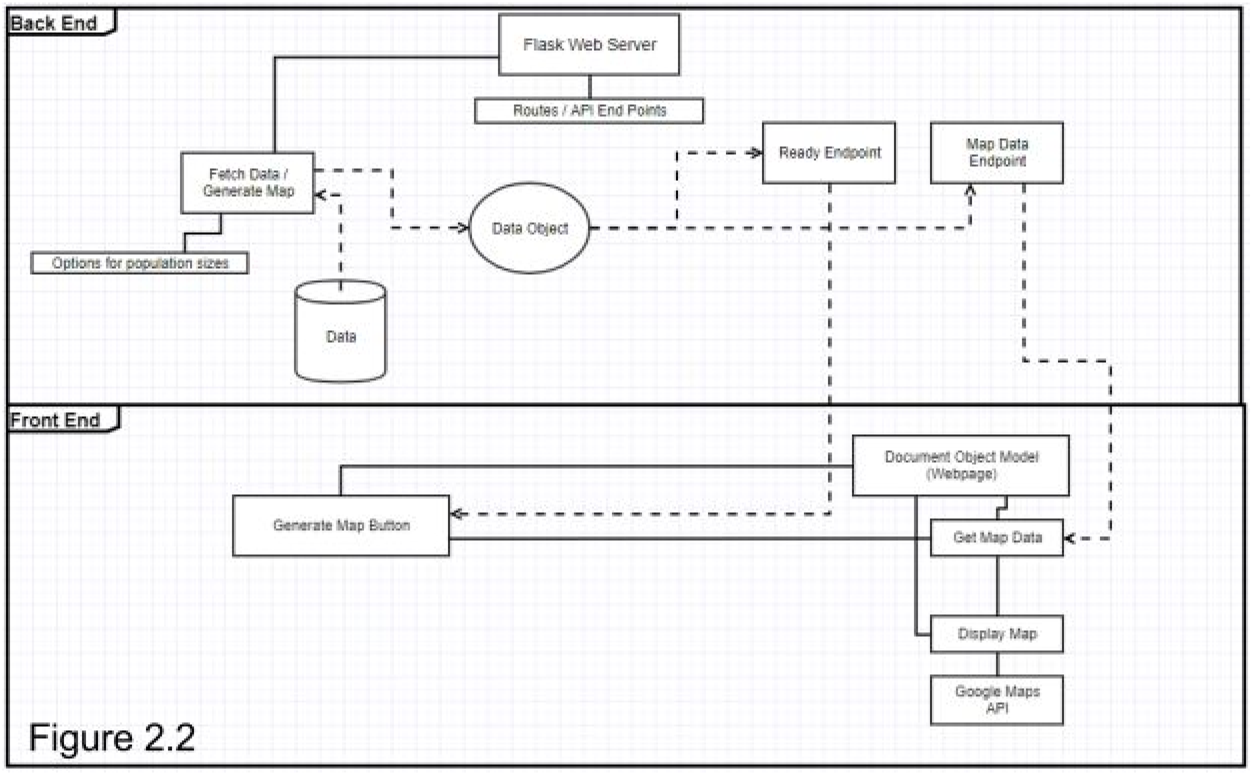


Figure 2.1a shows the logical architecture of our system. We will have an AWS database server that is in communication with the application server and then data itself. The application will receive information through the application server that is pulling information from the database server. The web server will then communicate back and forth with the application server and the internet to display the web application to the user on the other end of the browser.

Since this application is a web based application, it will be able to run on any operating system and platform that can access the web page. We will be using Amazon Web Services to provide a web server and easy access to the data and the back end application will be running in Python as long as it meets our speed requirements. The front end of the application, the UI layer, is going to be designed using HTML, Javascript, and CSS to create an overlay on Google Maps. The design is based on the individual design of various components in which the application pulls data from census sites. The software architecture is designed to incorporate any of the chosen data entries and modifications into an algorithm that manages to redistrict any given state.

2.2 Decomposition Description



Our front end will allow the user to request a new map by clicking the “Generate Map” button. The javascript on the webpage will then send this request to an API endpoint on our backend. The page will show a loading icon as long as the server says that the data is still being calculated. Once the data is ready, the front end will be made aware of the fact through the API that was just called.

The response to the front end will be a list of cities that act as seeds for the districts that have been generated. The front end will take this list of seeds and start requesting more detailed information from the back end, through a separate API endpoint, for each city. This information is stored in the front end, and used to start generating the map for the user. The data is passed onto a map generation function on the front end that will interface with the Google Maps API on our front end to depict the results of the map to the user.

Functionality that we will add after finishing core requirements is the option for the user to generate alternate maps with non-default map generation parameters.

3. Persistent Data Design

3.1 Database Descriptions (if you use a database)

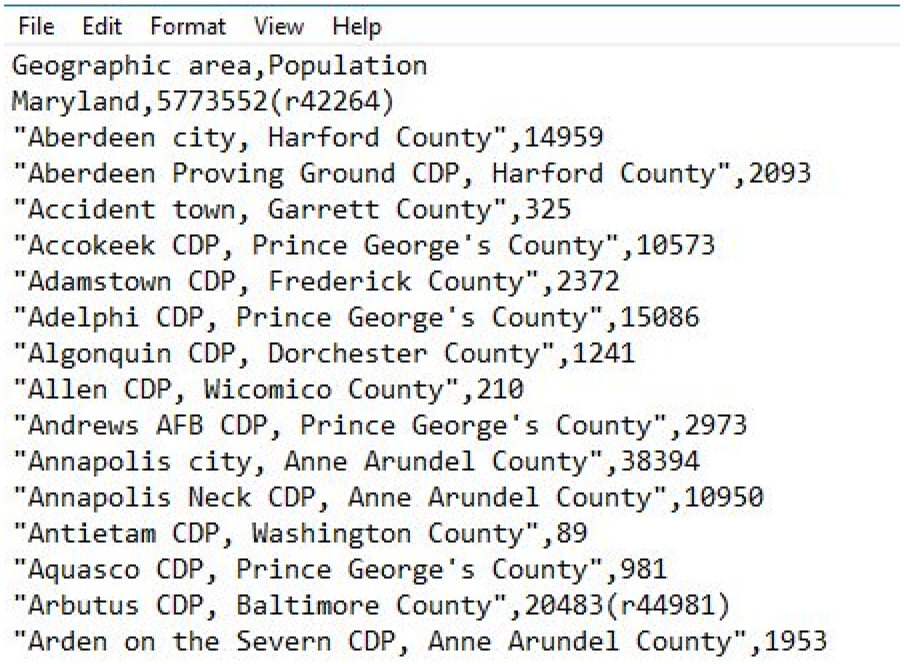
As of now, we have determined that we will not be using a database for our project. With the data sources that we have, we believe that parsing our data sets and culling extraneous data so that we only have map data that is in the scope of this project, will be necessary.

3.2 File Descriptions

We will be using census data in the form of CSV files to gather the required information. The CSV file contains strings and integers that correspond to Maryland cities and their respective populations. We will manually preprocess this data to only have information that is in the scope of this project. After doing this and converting the data into either JSON or XML, it will be ready for immediate processing whenever our front end requests it.

The backend will process the data and calculate the districts when the front end requests it, and then start passing the relevant information back in the form of JSON so the javascript-based front end can easily represent and manipulate the results.

The following is a screenshot of a small sample of our data:



```
Geographic area,Population
Maryland,5773552(r42264)
"Aberdeen city, Harford County",14959
"Aberdeen Proving Ground CDP, Harford County",2093
"Accident town, Garrett County",325
"Accokeek CDP, Prince George's County",10573
"Adamstown CDP, Frederick County",2372
"Adelphi CDP, Prince George's County",15086
"Algonquin CDP, Dorchester County",1241
"Allen CDP, Wicomico County",210
"Andrews AFB CDP, Prince George's County",2973
"Annapolis city, Anne Arundel County",38394
"Annapolis Neck CDP, Anne Arundel County",10950
"Antietam CDP, Washington County",89
"Aquasco CDP, Prince George's County",981
"Arbutus CDP, Baltimore County",20483(r44981)
"Arden on the Severn CDP, Anne Arundel County",1953
```

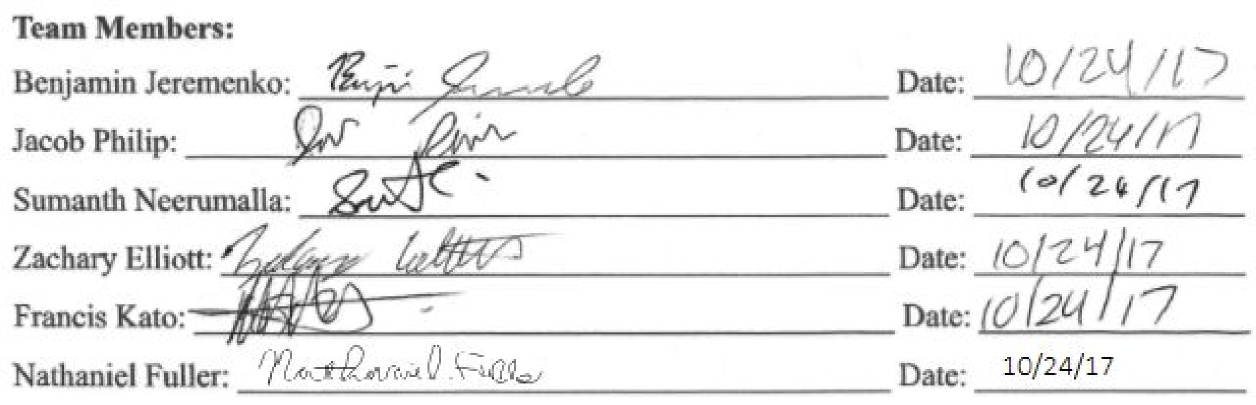
4. Requirements Matrix

Functional Requirement # / Name	System Component
001 / Mouse-Over District	The Display Map component of the front end will handle the mouse-over district requirement. The JavaScript code used in the front end will detect when a user's mouse is hovered over a district. When this happens, another function in the Display Map component will be called to expand that section of the map and display statistics.
002 / Loads Webpage	The Flask Web Server component of the back end will serve up the page and allow the user to load the webpage in their browser. During this phase the JavaScript files will be loaded into the browser.
003 / Loses Internet Access	The Get Map Data and Generate Map Button components will ensure this functionality. They will continue to try to connect to the server and resume operation as soon as the connection is re-established.
004 / Zoom In On Map	The Display Map component of the front end will handle the zoom in on map requirement. There will be zoom in and zoom out buttons on the page. The JavaScript code will detect when the button is pressed and will call the appropriate function to zoom in or out on the Google Map and our overlay.
005 / Hosting Multiple Users	The Flask Web Server component of the back end will handle hosting multiple users on the web page. Since we are building the API's to give data on the fly, only as requested, the system can support multiple clients requesting disparate sets of data.

Appendix A – Agreement Between Customer and Contractor

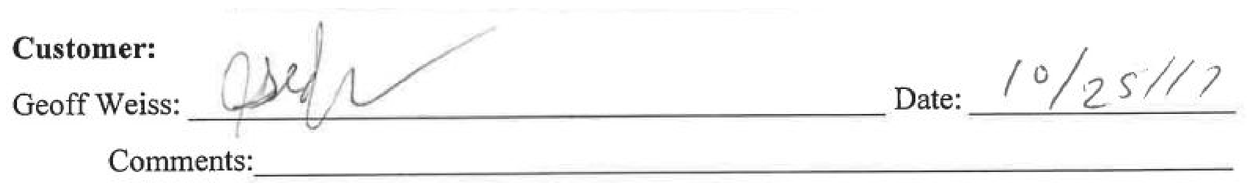
The system to be implemented will be a web app that displays 8 districts in Maryland that have been redistricted to create evenly populated districts. The architectural design, decomposition, and requirements are accurate and determine the guidelines of this system. Bricks and the customer, Geoff Weiss, all agree to the terms provided in this document.

If a change to this document is necessary, the following procedure will be used. The edit to the document will first be discussed in the biweekly meetings. If agreed upon by both parties, the new document will be drafted by the development team, Team 17. Once completed, the finished document will be sent to the customer to be resigned and then resubmitted to the github account.



Team Members:

Benjamin Jeremenko:	[illegible]	Date:	10/24/17
Jacob Philip:	[illegible]	Date:	10/24/17
Sumanth Neerumalla:	[illegible]	Date:	10/24/17
Zachary Elliott:	[illegible]	Date:	10/24/17
Francis Kato:	[illegible]	Date:	10/24/17
Nathaniel Fuller:	[illegible]	Date:	10/24/17

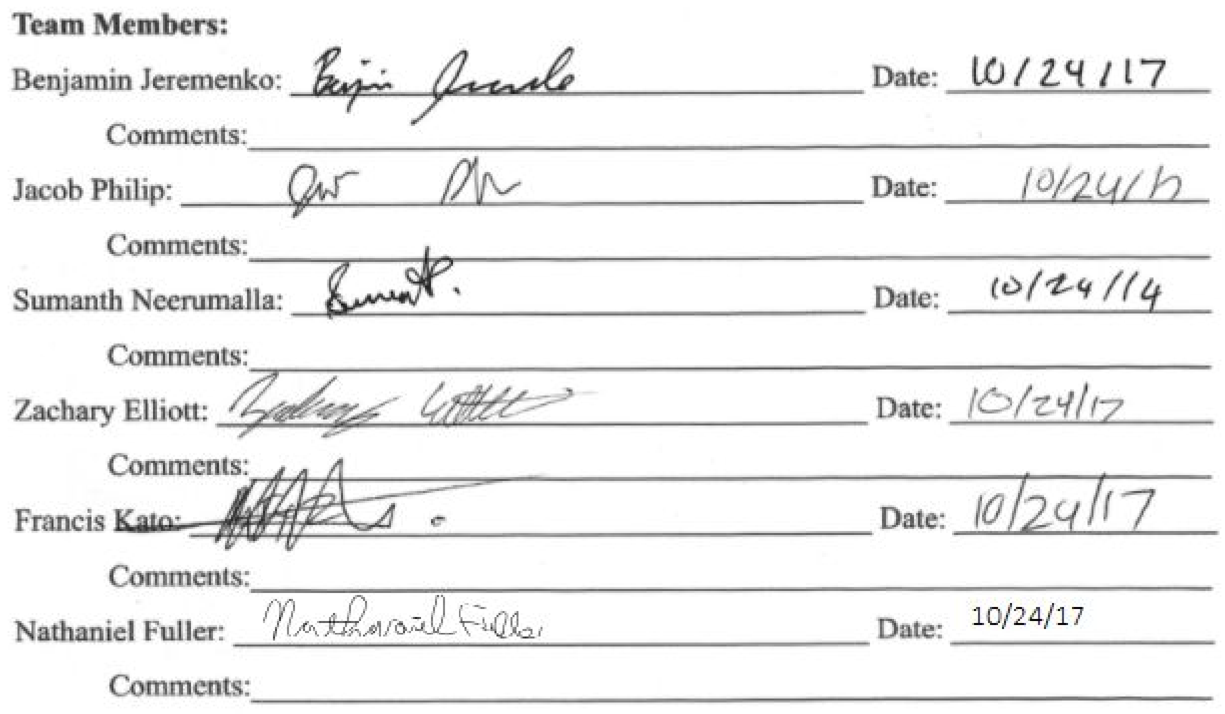


Customer:

Geoff Weiss:	[illegible]	Date:	10/25/17
Comments:			

Appendix B – Team Review Sign-off

All of the following team members have reviewed this document and agree to the content and format.



Team Members:

Benjamin Jeremenko: [illegible] Date: 10/24/17

Comments:

Jacob Philip: [illegible] Date: 10/24/17

Comments:

Sumanth Neerumalla: [illegible] Date: 10/24/14

Comments:

Zachary Elliott: [illegible] Date: 10/24/17

Comments:

Francis Kato: [illegible] Date: 10/24/17

Comments:

Nathaniel Fuller: [illegible] Date: 10/24/17

Comments:

Appendix C – Document Contributions

Benjamin Jeremenko:	Formatting and Reviewing Document	5%
Jacob Philip:	Requirements Matrix	15%
Sumanth Neerumalla:	Decomposition Description & Persistent Data Design	30%
Zachary Elliott:	Architectural Design & Decomposition Diagram	30%
Francis Kato:	Introduction, References, & File Descriptions	15%
Nathaniel Fuller:	Appendices	5%