Casa Domus

CMSC 447

Software Design Description (SDD)

Date: May 14th, 2018

Version: 1.3

Authors: Ryan Coleman, Elia Deppe, James Williams

1. About the Web Service

1.1 Identification

This Software Design Document (SDD) document applies to Casa Domus 1.0, a web-based home finding software for the lower 48 States of the US.

1.2 System Overview

The purpose of this web-based software is to enable users to easily find counties to live in based off their living preferences. The web-based software will perform these tasks by obtaining user preferences from a questionnaire and comparing those preferences to information gathered from various APIs (Application Programming Interface), .csv (Comma Separated Value) files, and databases. A map will display county shapes that gradient based off the comparability of the housing options in various areas to the user preferences. The project sponsor is Professor Charles Nicholas. The development team is Casa Domus. This software was developed over the spring of 2018.

1.3 Document Overview

This document will serve as overall guidance for the development of the Casa Domus web service. This includes the behavior of the service from the user's perspective, a description of the framework running the web service, the API's and Resources used in the service along with their usage within the system, and all UI and design choices. This document will also provide detailed design of each software component. This includes all relations between all software components and any algorithms used in the process.

The project is sponsored by Russ Cain. The customer is Charles Nicholas. The developers working on this project are: Elia Deppe, Cynthia Chou, Ryan Coleman, William Gao, Haoran Ren, and James Williams.

2. High Level Design Choices

This section shall describe the exact execution of the web service from the perspective of the user. The purpose of each page will be given, and the exact interaction given to the user will be provided, along with the provided outcome.

2.1 Page 1: Initial Statement

Upon opening the service, the user will be given an initial statement explaining the nature of the website. The Casa Domus name and logo will be presented, along with a "Begin" button that will move to the next page of the service. No other buttons or inputs will be presented on this first page.

2.2 Page 2: Questionnaire

After the first page is given, page 2 will present the user with a questionnaire. The nature of the questions given shall be answered by use of sliders. Inputs to these questions will be graded on

a numerical scale by the use of these sliders. Sliders shall be used in order to reduce user input as much as possible while still obtaining enough information to generate accurate results.

The questions given shall obtain the user's current yearly income, prefered property value, prefered cost of living index, prefered price of rent per month for one bedroom unit, and climate (including both summer and winter temperatures). The sliders will be presented with each question, and will show the associated unit of measure.

Under each of the questions relating to yearly income, prefered property value, and prefered price of rent per month will be sliders showing a monetary amount, with both ends of the sliders representing minimum and maximum values associated with each question.

Under the question for cost of living will be a slider with the middle of the slider representing the median county or counties and with both ends representing the county with the highest and lowest index for cost of living. This slider is more involved, as it must explain to the user to choose their maximum index for cost of living that they are willing to pay. The higher the cost of living index, the more their money is worth in that county.

Under the two questions for climate in the summer and the winter will be sliders showing a temperature amount, with both ends of the of the slider representing minimum and maximum values associated with temperature. The slider for winter represents average temperature in January in the county. The slider for summer represents average temperature in July for that county.

Once the questions are completed by the user, the user shall be able to submit their answers with a "Submit" button that will move to the next page of the service. Once submitted by the user, the program will proceed to generate the next and final page.

2.3 Page 3: Map/Results

The final page shall display the results on a map of the counties of the lower 48 US states. Counties contained in the results of the program will be colored with a gradient indicating likeness to the user's preference. Gradient coloring on the map is used because it shows location and level of likeness more intuitively than just a list of results. The user will be able to look at their generated map and examine their results.

Along with the map on this final page, the user shall be able to further interact with their map by re adjusting their preference via the same sliders from the questionnaire page and resubmitting.

3. CSCI architectural design

This section shall describe the planned framework of the web service along with all components and interfacing API's that will be used in the framework of the design.

3.1 CSCI Components

The Casa Domus web service shall be made up of the following CSCI components, including but not limited to:

a. The AngularJS framework upholding the webservice
b. Papa Parse is used to translate CSV resources into iterable data for the website.
c. searchCounty: A module that can obtain all information about a county given its county name and the state the county lies in.
d. sortCounty: A module that can compare the results of the user's answers from the questionnaire to every county available in the master list of counties.
e. mapInterface: A module that serves as an interface to the Google Maps Javascript API. It allows the programmer to easily feed data into the map and to dynamically apply styles (e.g. color) on each county displayed on the map.

3.2 Concept of Execution

Upon opening the web service, the first component that will activate is the view of the AngularJS framework displaying page 1 of 3. This includes rendering the Casa Domus title and logo, along with a "begin' button. Once the the "begin" button has been clicked, page 2 will be rendered bringing the user to the questionnaire.

Upon loading the page, the framework will call the Parser to gather all the necessary data for all counties from the CSV's. Data is is parsed from CSV into JSON (Javascript Object Notation) using the Parser. Once parsed into JSON, they will be stored locally on the user's browser. Once the user completes the survey and submits, the framework will call the sortCounty module to compare the user's results with every other county. In order for the sortCounty module to get the individual data for each county, the sortCounty module will call the searchCounty module for each county. The searchCounty module can supply the information for any single county, so the sortCounty module will loop through every county in the US, using the searchCounty module to find each county, and comparing the results with the user's results.

The comparison done by the sortCounty module will be executed by using a Manhattan Distance function. Manhattan Distance is used because of its speed compared to other distance functions. Each county is given a 'distance' rating. The lower the distance, the closer of a match it is to the user. Once each county in the US is given a distance, the entire list of counties is sorted by distance and then returned back to the framework.

The sorted list of counties is passed to the mapInterface module. The mapInterface module loads a map from the Google Maps API and applies a data layer to the map. The data layer accepts a geoJSON (Geographical Javascript Object Notation) file that defines the polygon information for each county as well as the GeoID and a simple county name. Each individual county will be shown in a color gradient from red (0°) to blue (260°) in the HSV color module. The module Counties shown in colors close to blue are counties that closely match the user's preferences, counties shown in colors close to green somewhat match the users preferences, and counties shown in colors close to red do not match the users preferences. Counties shown

in the color black do not have all the necessary data associated with them to be put through the distance calculation.

Users will then be able to edit their responses to the questionnaire and receive different results.

4. CSCI detailed design

This Section shall describe in finer detail the design of each component involved in the web service.

4.1 CSV to JSON Parser

The free and open source Papa Parse will be used to convert CSV strings into JSON. As Papa parse is completely free use with an MIT License and is open source to the public.

4.2 The searchCounty Module

The searchCounty module will be written as a Javascript module that will be called by the sortCounty module when needed. Before it can be used, it needs to have the JSON arrays that contain each aspect of the counties data. The searchCounty module will also need to be given the name of a county and the state that the county lies in. For each JSON array of data (Median Income, Median Property Value, etc), the searchCounty module will loop through each county until the desired county is found by its name or GeoID. Once each piece of data is found from it's respective JSON array, the county is bundled into a single JSON object, and is returned. The structure of the JSON object returned is as follows:

```
county = {
        state,              (The county's state)
        countyName,         (Name of the county)
        areaName,           (Name of the county and the state)
        medianHHIncome,     (The county's median household income)
        medianProperty,     (The county's median property values)
        rent1bed,           (The county's median cost of rent for a one bedroom unit)
        costOfLiving,       (The state's cost of living index)
        geoID,              (The county's geo ID)
        jan,                (The county's average temperature in January)
        july,               (The county's average temperature in July)
        normProp,           (The county's normalized median property value)
        normLiving,         (The state's normalized median cost of living index)
        normIncome,         (The county's normalized median household income)
        normRent,           (The county's normalized median cost of rent for one bed)
        normJan,            (The county's normalized average temperature in January)
        normJuly            (The county's normalized average temperature in July)
}
```

4.3 The sortCounty Module

The sortCounty module will be written as a Javascript module that will be called when the framework needs to compare the user's answers to the questionnaire against each and every county in the US. To do this, the module will loop through a list of all the counties in the US, look up each one's information using the searchCounty module, and calculate each county's 'distance'. Distance will be appended to to each county's JSON object, and the JSON object will be appended to a master JSON list called userResults. Once each county's 'distance' is calculated, userResults is sorted by distance to the user. This is sorted in a way that index 0 will be the closest county, index 1 is the second closest county, and so on. The JSON array will userResults is then given to the mapInterface module.

4.3.1 Algorithm Deciding the Best Counties

The web service will obtain at least 4 values submitted by the user in the questionnaire.

1. Median Property Value
2. Median Cost of Rent
3. Cost of Living
4. Median Income
5. Winter Temperature
6. Summer Temperature

Updated:

In addition the web service will also receive the same 6 values from each county. Selecting those counties which are closest to the user's preference will simply be done through a Manhattan Distance calculation. The formula is as follows:

$$Distance(C,\ U)\ =$$

$$|MedPropVal_C - MedPropVal_U| + |MedRent_C - MedRent_U| + |Living_C - Living_U| + |Income_C - Income_U| +$$

$$|SummerTemp_C - SummerTemp_U| + |WinterTemp_C - WinterTemp_U|$$

This formula will have to be applied for every county retrieved, againsts the user's values. Once calculated, counties that are closest to the user's preference will be those with the smallest euclidean distance calculated by the formula above.

Outdated:

> In addition the web service will also receive the same 4 values from each county. Selecting those counties which are closest to the user's preference will simply be done through a Euclidean Distance calculation. The formula is as follows:

$$Distance(C,\ U)\ =$$

$$\sqrt{(Housing_C - Housing_U)^2 + (Living_C - Living_U)^2 + (Income_C - Income_U)^2 + (Climate_C - Climate_U)^2}$$

where C represents a county value, and U represents the user's value

This formula will have to be applied for every county retrieved, againsts the user's values. Once calculated, counties that are closest to the user's preference will be those with the smallest euclidean distance calculated by the formula above.

4.4 The mapInterface Module

The mapInterface module shall be written as a Javascript module that uses the Google Maps Javascript API to visualize the data received from the sortCounty module. The map is first initialized by calling to the Google Maps constructor function for maps, the map is centered at coordinates 37°N, 95°W (around the center of the US). The map then is then fed the GeoJson file which contains identifying attributes such as its county shape, its GeoID, land area, etc. Each county is given a "gradient" property which represent their position in the userResult list. The gradient value is the county's position in the list normalized between 0 and 1, 1 being closer to the front of the list, and 0 being towards the end of the list. If a county is not in the list it is not given a gradient value. The map then has a style applied. Those counties whose gradient value is close to 0 are colored red, while those counties whose gradient value is close 1 are colored blue. Those counties that did not receive a gradient value are colored black. The map also has a legend pushed onto it to display the color relationships. A click event listener is added to the map that will display all available info on a county when it is clicked on the map. The mapInteface module also provides the ability to focus on any provided coordinates on the map if a county from the searchCounty module contains proper latitude and longitude for the county center.

4.4.1 GeoJson

GeoJson comes in the form of a Feature Collection, which is an array of features. In this case the features are counties that have their shapes defined as arrays of coordinates. Then they have an object called properties which holds identify attributes such as the county's GeoID. Example:

```
{"type": "FeatureCollection", "features": [
 {"type" : "Feature", "geometry": { "type": "Polygon", "coordinates" : [[[latitude1,longitude1,],...]]] },
                        "properties": {"AFFGEOID" : "09000US06007", etc.}
 {"type" : "Feature", "geometry": { "type": "Polygon", "coordinates" : [[[latitude1,longitude1,],...]]] },
                        "properties": {"AFFGEOID" : "07000US0321", etc.}
]}
```

4.4.2 County Color Interpolation

Color Interpolation will be done using linear interpolation with HSV color data. The higher the value Distance(C, U) is, the farther the color will be from its original value (Blue). The formula is as follows:

$$Color_{County} = Color_{Blue} * t + Color_{Red} * (1 - t) \text{ where } 0 <= t <= 1$$

where t represents a modified weight value calculated from Distance(C, U)

5. Requirements traceability

5.1 Requirements from the Client

1. **PLATFORM**
    a. 1, 1.1, 1.1.1, 1.1.2
2. **FUNCTIONAL**
    a. 1.1, 1.2, 1.3,1.4,~~1.5~~,1.6, 1.7, ~~1.8, 1.9, 1.10, 1.11, 1.13~~
    b. 2, 2.1, 2.2
    c. 3, 3.1.1, 3.1.2, 3.2, 3.3, ~~3.4, 3.5, 3.6~~, 3.8
    d. ~~4~~
    e. 5, 5.1, ~~5.2~~
3. **GUI**
    a. 1
    b. 2, 2.2, 2.3
    c. 3

5.2 Derived Requirements

1. **PERFORMANCE**
    a. 1, 1.1
2. **FUNCTIONAL**
    a. ~~1.12, 1.14~~, 1.15
    b. 3.7
    c. 4.5, 4.6, 4.7
    d. 5
    e. 6
3. **GUI**
    a. 2.1,2.3.1

6. Notes

This section shall contain any general information that aids in understanding this document (e.g., background information, glossary, rationale). This section shall include an alphabetical

listing of all acronyms, abbreviations, and their meanings as used in this document and a list of any terms and definitions needed to understand this document.

API	Application Programming Interface
AngularJS	Angular Javascript web framework
CSCI	Computer Software Configuration Item
HSV	Hue, Saturation, Value RGB representation model
RGB	Red, Green, Blue additive color model
SDD	Software Design Description

~~6.1 Coding Standards~~

1. ~~Classes~~
    a. ~~Classes will be in CamelCase with the first letter capitalized. Example: ExampleClass~~
    b. ~~Class declarations should be contained to one file, with the file named after the class it contains.~~
    c. ~~Proper data control/hiding should be done with the classes. Any variable should be private/protected (unless it is a const global) and should only be accessed via a getter method. All methods that are only called internally should be private/protected; otherwise they are public.~~
2. ~~Functions~~
    a. ~~Functions will be in CamelCase with the first letter capitalized. Example: ExampleFunction();~~
    b. ~~All functions should be declared as prototype, body implemented later (if supported.)~~
    c. ~~Functions can be sorted in any way you prefer but if possible try to do so alphabetically or by call order.~~
3. ~~Variables~~
    a. ~~All variables will be in camelCase with the first letter lower case. Example: exampleVariable~~
    b. ~~A variable that is a const global should be done in underscore caps. Example: GLOBAL_VARIABLE~~
    c. ~~Any variable that is affiliated with the user interface should start with ui and followed by what type of object it is (Slider, Button, etc.), followed by a specifier. Example: uiSliderWeather~~

d. Variables can be listed in any order but if possible try to instantiate them in alphabetical order according to type or usage order.
e. Please declare all variables at the top of a class/function. This does not apply to variables instantiated for loops (for(int i = 0…)).
f. If possible, declaring variables on the same line is okay (same type only.)

4. Comments
   a. There should be at least one comment per five or so lines of code.
   b. Comments should be set above the code, not to the side.
   c. Functions
      i. Functions should be commented with block comments, stating its name, a description of its parameters, what it returns, and a description of what it does. Example:

```
/*
 Name: ExampleFunction
 Author(s): <Authors>
 Date: <Date>
 Parameters:
  - <type> variableName1
  - <type> variableName2
 Returns: <type> <name> - <Description of what it should be>
 Description: <Sentence to paragraph depending on length and complexity, if more is needed, consider breaking your function into more than one function>
 Sources: <Optional field, include any links to code that you've used as reference. Please inline comment where you used the source code>
 Changes:
  - <Author> <Date>: <Sentence/paragraph detailing change>
*/
```

5. Algorithms
   a. Algorithms should be appropriately commented with a block comment stating its functionality and how it works. If a function encompasses an algorithm and just that, the function comment will suffice.

```
/*
 Algorithm: <Name>
 Author: <Name>
 Date: <Date>
 Description: <Sentence to paragraph depending on length and complexity>
 Source: <Optional field, include any links to code that you've used as reference. Please inline comment where you used the source code>
*/
```

6. Error Handling

a. ~~Proper handling should be taken for any possible errors using try catch blocks, printing the error to stderr if possible.~~
b. ~~The exception caught should be specified by the appropriate error type instead of the parent Exception, unless there are multiple possible exceptions, which should then be specified via switch statement, the cases testing the exception types.~~
c. ~~If an error occurs, do our best to recover from the error so that the program does not need to be reloaded.~~

(Removed: This section was removed because we did not ultimately adhere to the coding standards set by ourselves. Not following these standards did not hold us back from implementing most of the required features.)

7. Referenced Data Sources and APIs

Name	**Link**	**Use**
Google Maps API	https://developers.google.com/maps/	Displaying Map
Google Fonts	https://fonts.google.com/	Font Styling
Papa Parse	https://www.papaparse.com/	Parsing CSVs
FlatIcon	https://www.flaticon.com/	Provided Favicon
Weather: NCDC Climate Data Online	https://www.ncdc.noaa.gov/cdo-web/webservices/v2	Provided Weather Data
DataUSA	https://datausa.io/	Provided Median Property Value
HudUser	https://www.huduser.gov/portal/datasets/50per.html	Provided 50th Percentil Rent Estimates
Census Bureau	https://factfinder.census.gov/faces/tableservices/jsf/pages/productview.xhtml?pid=ACS_16_5YR_S1903&prodType=table	Provided Median Income by County
Missouri Economy	https://www.missourieconomy.org/indicators/cost_of_living/	Provided cost of living (COLI) dataset for entire USA*

*Next page

*Data is done by state (Data found at a county level is of too high a cost)

1. Cost of living is determined by the Cost of Living Index. The index works as such
    a. $\frac{Amount\ 1}{State\ Index\ 1} = \frac{Amount\ 2}{State\ Index\ 2}$
    b. That is to say, the monetary value in one state is comparable to another monetary value in another state. Using the dataset from above, $200 of groceries in Texas is equivalent to $300.89 of groceries in Alaska.

Appendix A: Licenses

Name	**License**
Google Maps API	Google APIs License
Google Fonts	Google APIs License
Papa Parse	MIT
FlatIcon	Free Use, with Credit
Weather: NCDC Climate Data Online	Free Use
DataUSA	GLPv3
HudUser	Free Use
Census Bureau	Free Use
Missouri Economy	Free Use