CASA DOMUS

Software Design Description (SDD)

Date: May 12th, 2018

1. About the Web Service

1.1 Identification

This Software Design Document (SDD) document applies to CasaDomus 1.0, a web-based home finding software that utilizes user inputs to generate a map with a color gradient for each county in the lower-forty eight states of the United States of America, corresponding to the closeness that the conditions in each county has with the user's preferences.

1.2 System Overview

The purpose of this piece of software, which should run on any system that is capable of any of the two following web browsers, Chrome version 60.0+, and Firefox version 58.0+, is to enable users to easily find housing locations based off their preferences. The web-based software should perform these tasks by using user preferences obtained from a questionnaire to search through information obtained through various APIs, .csv files, and a database, which display to the user a map that is gradiented 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 CasaDomus. 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 CasaDomus 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. UI and 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 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.

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. This slider will have to be marked with familiar reference cities that show the user example indices to base their answer on.

In the case that the 'cost of living' slider fails to convey the nature of the question, then either the method of input must be changed from a slider to something that better represents the question, or the question itself must be abstracted in some way to elicit an answer with a different unit of measure but with the ability to derive a cost of living index value.

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 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. Any and all API's or resources used strictly in the retrieval of county information:
    i. The NCDC's Climate Online Web Service
    ii. The Data USA API
    iii. 50th Percentile Rent Estimates from the Office of Policy Development and Research (PD&R)
    iv. 2016 Median Income data from the US Census
    v. Cost of Living Data Series 2017 Annual Average From the Missouri Economic Research and Information Center
    vi. Google Maps Javascript API
    vii. Google Maps Fusion Tables API
c. A Parser used to translate CSV resources into iterable data for the website.
d. searchCounty: A module that can obtain all information about any county given its county name and the state the county lies in.
e. 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.

~~Components will fit together starting with the framework written in AngularJS. The framework will be layered as a Model-Controller-View as per the AngularJS standard model for single page websites. All API's and data from resources will connect to the controller of the framework. Data being handled by the website will be done in background out of site from the user.~~

~~After the Questionnaire is submitted by the user, the controller will make all needed API requests, and all retrieval of information needed to calculate the user's estimated results. Data collected by the webservice is to be kept locally on memory allocated by the user's browser, until the service is terminated.~~

~~Upon initializing page 3, the view will be updated using the Google Maps API to present an augmented map with resulting counties displayed with the use of gradient. Colors on the map will be interpolated using an HSV linear interpolation algorithm to adjust hue on a scale from blue to green to yellow to red.~~

~~Lastly, any further adjustments made by the user on page 3 that result in a resubmit will repeat the same process over again, starting with the controller retrieving all necessary data and the Google Maps API re-updating the map view.~~

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 completing the questionnaire and submitting it, the framework controller will proceed to send all API requests out and retrieve all data needed. Once retrieved, this data is to be cached locally on memory provided by the model and allocated by the user’s browser.~~

Upon loading the questionnaire, 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 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 and then returned back to the framework.

~~Shortly after all county information is received and processed, a list of counties shall be generated on the basis of a euclidean distance algorithm.~~

Given every counties ‘distance’ to the user, the list of counties sorted by distance is then sent to the Google Maps API. [continue here]

Each individual county will be shown in a color gradient from red to blue, based on HSV color interpolation. 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, therefore are not shown on the map.

Users will also be able to go back 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 loaded into the framework to 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)
            rent4bed,             (The county's median cost of rent for a four bedroom unit)
            costOfLiving,         (The county's cost of living index)
            costOfGroceries,      (The county's cost of groceries index)
            population,           (The county's population)
            hu2010,               (?)
            geoID,                (The county's geo ID)
            fips2010,             (The county's fips 2010 number)
            fipstxt,              (The county's fips number)
            jan,                  (The county's average temperature in January)
            july,                 (The county's average temperature in July)
}
```

4.3 The searchCounty Module

The sortCounty module will be written as a Javascript module that will be loaded into the framework to 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 userResults is returned back the framework to be used to display the results. More simply, userResults are the results of the user's questionnaire answers.

4.4 Algorithm Deciding the Best Counties

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

1. Cost of Housing
2. Cost of Living
3. Income
4. Climate

Updated:

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 Manhattan Distance calculation. The formula is as follows:

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

$$a * |Housing_C - Housing_U|\ \ + b * |Living_C - Living_U|\ + c * |Income_C - Income_U|\ + d * |Climate_C - Climate_U|$$

where a, b, c, and d are the user selected weights for each choice.

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.

4.2 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:

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

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

4.3 The mapInterface Module

asdfasdf

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, 1.6, 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~~, 4.1, 4.2, 4.3, 4.4
    e. 5, 5.1, ~~5.2~~
3. GUI
    a. 1
    b. 2, 2.2, 2.3

5.2 Derived Requirements

1. PERFORMANCE
    a. 1, 1.1
2. FUNCTIONAL
    a. ~~1.12, 1.14~~, 1.15, 1.16
    ~~b. 3.7~~
    c. 4.5, 4.6, 4.7
    d. 5
    e. 6
3. GUI
    a. 2.1

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.

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.

7. Referenced API's

1. Google Maps API - https://developers.google.com/maps/
2. Google Fonts - https://fonts.google.com/
3. Papa Parse - https://www.papaparse.com/
    a. Free Use, Open Source, MIT License
4. FlatIcon - https://www.flaticon.com/
    a. Free Use (Must credit the creators)
5. Weather: NCDC Climate Data Online - https://www.ncdc.noaa.gov/cdo-web/webservices/v2
    a. Free Use
6. Median Property Value by County - https://datausa.io/
    a. License: GNU Affero General Public License v3.0 (GPLv3)
        i. https://www.gnu.org/licenses/agpl-3.0.txt
7. 50th Percentile Rent Estimates - https://www.huduser.gov/portal/datasets/50per.html
8. Median Income by County - https://factfinder.census.gov/faces/tableservices/jsf/pages/productview.xhtml?pid=ACS_16_5YR_S1903&prodType=table
9. Cost of Living - Dataset available from https://www.missourieconomy.org/indicators/cost_of_living/
    a. Data is done by state (Data found at a county level is of too high a cost)
    b. Cost of living is determined by the Cost of Living Index. The index works as such
        i. $\frac{Amount\ 1}{State\ Index\ 1} = \frac{Amount\ 2}{State\ Index\ 2}$
        ii. 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.

A. Appendixes

DESCRIPTION/PURPOSE

The Software Design Description (SDD) describes the design of a Computer Software Configuration Item (CSCI). It describes the CSCI-wide design decisions, the CSCI architectural design, and the detailed design needed to implement the software. The SDD may be supplemented by Interface Design Descriptions (IDDs) and Database Design Descriptions (DBDDs).

APPLICATION/INTERRELATIONSHIP

Portions of this plan may be bound separately if this approach enhances their usability. Examples include plans for software configuration management and software quality assurance.

The Contract Data Requirements List (CDRL) should specify whether deliverable data are to be delivered on paper or electronic media; are to be in a given electronic form (such as ASCII, CALS, or compatible with a specified word processor or other support software); may be delivered in developer format rather than in the format specified herein; and may reside in a computer-aided software engineering (CASE) or other automated tool rather than in the form of a traditional document.

PREPARATION INSTRUCTIONS

General instructions.

a. Automated techniques. Use of automated techniques is encouraged. The term "document" in this means a collection of data regardless of its medium.

b. Alternate presentation styles. Diagrams, tables, matrices, and other presentation styles are acceptable substitutes for text when data required can be made more readable using these styles.

c. Title page or identifier. The document shall include a title page containing, as applicable: document number; volume number; version/revision indicator; security markings or other restrictions on the handling of the document; date; document title; name, abbreviation, and any other identifier for the system, subsystem, or item to which the document applies; contract number; CDRL item number; organization for which the document has been prepared; name and address of the preparing organization; and distribution statement. For data in a database or other alternative form, this information shall be included on external and internal labels or by equivalent identification methods.

d. Table of contents. The document shall contain a table of contents providing the number, title, and page number of each titled paragraph, figure, table, and appendix. For data in a database or other alternative form, this information shall consist of an internal or external table of contents containing pointers to, or instructions for accessing, each paragraph, figure, table, and appendix or their equivalents.

e. Page numbering/labeling. Each page shall contain a unique page number and display the document number, including version, volume, and date, as applicable. For data in a database or other alternative form, files, screens, or other entities shall be assigned names or numbers in such a way that desired data can be indexed and accessed.

f. Response to tailoring instructions. If a paragraph is tailored out of this document, the resulting document shall contain the corresponding paragraph number and title, followed by "This paragraph has been tailored out." For data in a database or other alternative form, this representation need occur only in the table of contents or equivalent.

g. Multiple paragraphs and subparagraphs. Any section, paragraph, or subparagraph in this DID may be written as multiple paragraphs or subparagraphs to enhance readability.

h. Standard data descriptions. If a data description required by this document has been published in a standard data element dictionary specified in the contract, reference to an entry in that dictionary is preferred over including the description itself.

i. Substitution of existing documents. Commercial or other existing documents, including other project plans, may be substituted for all or part of the document if they contain the required data.