Casa Domus

Software Test Description (STD)

1. About the Test Description

The purpose of this document is to describe the testing procedures that will ensure the operation of the web service, and that it has met the requirements described in the Casa Domus Software Requirements Specification (SRS). Included in this document are the methods that will be used to judge it's function and performance.

1.1 Identification

This Software Test Description (STD) is applicable to the Casa Domus web service and the software components used by the webservice. This includes but is not limited to the AngularJS framework, the modules used to parse, search, and sort all county data, and the Google Maps Javascript API. Tests described in this document shall apply to these components and their operation as a whole.

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 main function of the Casa Domus webservice is to find housing locations on the county level based off their preferences. When the webservice is loaded the system presents an initial questionnaire to obtain the user's basic living preferences. The user shall answer the questionnaire by the use of sliders. After submitting, the web service generates a colored map using the user's preferences obtained in the previous questionnaire. The system then searches through county information obtained through various .csv files and compares every county in the US with the user's preferences. After every county has been ranked, these results are sent to the Google Maps API, which displays a map that is gradiented based off similarity to the user's living preferences.

1.3 Referenced documents

1. Referenced Casa Domus Documents:
    a. Casa Domus Software Requirements Specification (SRS)
        i. All requirements in this document are referenced from the Casa Domus SRS
    b. Casa Domus Software Design Description (SDD)
        i. All CSCI's and System Components are referenced from the Casa Domus SDD

2. Test preparations

Due to the nature of the Casa Domus webservice, potential risk from a website failure is low. Sensitive account information is not handled by the webservice and all county data acquired is freely available from their respective US government agency. Risk for data loss or leakage is not a concern. Therefore no precautions or preparations are necessary in the testing procedure.

2.1 Hardware preparation

The web service runs on Chrome version 60.0+ and Firefox 58.0+. It designed to be run on a desktop client using either of the browsers mentioned. Testing of the web service will be done using both browsers on a desktop client.

2.2 Software preparation

During the test, memory used by the website is stored on RAM allocated by the user's browser. Memory tests are not required as the browser will manage the user's RAM. The software can be loaded simply by accessing its files with an appropriate browser. index.html is always the first page to be opened upon opening the web service. It is the greeting page, and it should have a button leading to survey.html, the questionnaire page. On the questionnaire page, a set of questions and sliders should be rendered, along with a submit button that takes you to the final page, map.html, the results page where the map with the user's results is rendered. On the results page should be the same sliders present from the questionnaire, along with a button to resubmit answers, and reload the results map.

3. Test descriptions

This section will describe the individual tests used to ensure that each requirement is fulfilled.

3.1 Project-unique identifier of a test

Each of the tests have identifiers similar to the requirements that they are testing. Tests are named beginning with the requirement identifier that it tests with the string "-TEST" appended to the end. For example, a test that tests the requirement with identifier "PLATFORM-49" would have the test identifier "PLATFORM-49-TEST."

Each test serves to test a requirement that was specified in the Software Requirements Specification (SRS), their relationships are described below:

Identifier	Description of relationship to Requirements
PLAT-1-TEST	Tests whether or not the product meets requirement PLAT-1.
PLAT-1.1-TEST	Tests whether or not the product meets requirement PLAT-1.1.

PER-1-TEST	Tests whether or not the product meets requirement PER-1.
PER-1.1-TEST	Tests whether or not the product meets requirement PER-1.1.
FUNC-1-TEST	Tests whether or not the product meets requirement FUNC-1, FUNC-1.1, FUNC-1.2, FUNC-1.3, FUNC-1.4, FUNC-1.5, FUNC-1.6, FUNC-1.7, FUNC-1.8, FUNC-1.9, FUNC-1.10, FUNC-1.11, FUNC-1.12, FUNC-1.13, FUNC-1.14, FUNC-1.15.
FUNC-2-TEST	Tests whether or not the product meets requirement FUNC-2.
FUNC-2.1-TEST	Tests whether or not the product meets requirement FUNC-2.1.
FUNC-2.2-TEST	Tests whether or not the product meets requirement FUNC-2.2.
FUNC-3-TEST	Tests whether or not the product meets requirement FUNC-3.
FUNC-3.X-TEST	Tests whether or not the product meets requirement FUNC-3.1, FUNC-3.2, FUNC-3.3, FUNC-3.4, FUNC-3.5, FUNC-3.6, FUNC-3.7.
FUNC-4-TEST	Tests whether or not the product meets requirement FUNC-4.
FUNC-5-TEST	Tests whether or not the product meets requirement FUNC-5, FUNC-5.1, FUNC-5.2.
GUI-1-TEST	Tests whether or not the product meets requirement GUI-1.
GUI-2-2.1-TEST	Tests whether or not the product meets requirement GUI-2, GUI-2.1.
GUI-2.2-2.3-TEST	Tests whether or not the product meets requirement GUI-2.2, GUI-2.3.
GUI-3-TEST	Tests whether or not the product meets requirement GUI-3.

3.2 Requirements addressed

As referenced in the Casa Domus, these are the CSCI components that make up the system:

a. The AngularJS framework upholding the webservice
b. Any and all API's or resources used strictly in the retrieval of county information
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.
f. 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, specifically a color on each county, to the map.

Given the CSCI's used by the system, these are the respective tests that correspond with each CSCI:

Test	**CSCI**
PLAT-1-TEST, PLAT-1.1-TEST, PER-1-TEST, PER-1.1-TEST	AngularJS Framework
FUNC-1-TEST	AngularJS Framework, searchCounty, sortCounty, Parser
FUNC-2-TEST, FUNC-2.1-TEST, FUNC-2.2-TEST	searchCounty, sortCounty, mapInterface
FUNC-3-TEST, FUNC-3.X-TEST	AngularJS Framework
FUNC-5-TEST	searchCounty, sortCounty, mapInterface, AngularJS Interface
GUI-1-TEST, GUI-2-2.1-TEST, GUI-2.2-2.3, GUI-3-TEST	AngularJS, mapInterface

3.3 Prerequisite conditions

For all the tests listed, none of them have prerequisites except for the following:

1. PLAT-1.1-TEST: The version of Chrome used to test this requirement shall be at least version 66.X.XX (Release 2018-04-17) and the version of Firefox used to test this requirement should be at least version 59.X.X (Release 2018-03-13)
2. GUI-2-2.2-TEST: Assumes the questionnaire has been completed and submitted to the system by the tester.
3. GUI-3-TEST: Assumes the questionnaire has been completed and submitted to the system by the tester, a map has been rendered, and is ready to receive a new set of

input from the sliders on page 3.

a. These prerequisites must be met before their respective test can be carried out. If they are not met before the start of the test, then the results of the test are inconclusive and cannot be used to verify the fulfillment of the requirement they test.

3.4 Criteria for evaluating results

Results from each test are to be compared the requirement that they test. Testers will need to analyze the requirement entirely, and if each part of the requirement is met then the test is considered passed. Only the performance tests will be evaluated for accuracy and timing. All other tests are to be evaluated subjectively by the testers. If a test is interrupted, or is halted indefinitely, then the test is considered failed. If the results of the test do not meet the expectations of the respective requirement, then the test is considered failed.

3.5 Test procedures

Included in this section is the pseudocode for the exact tests that will be used during the testing procedure. Tests are separated by row, and are shown with their project unique identifier.

Identifier	**Test Procedures (pseudo-code)**	**Assumptions and Constraints**
PLAT-1-TEST	For any supported browser item: Open the browser Attempt to open product using browser If (product_opened == True): Exit("Test successful: Product is web-based"); Exit("Test unsuccessful: Product not functional via web-based means");	• any software that can be opened and loaded successfully via a web browser is web-based

PLAT-1.1-TEST	specified_desktop_browser _clients = {Firefox_version_#, Chrome_version #};
	For all specified desktop browser clients:
	Open the browser
	Attempt to open product using browser
	If (product_opened == False):
	Exit("Test unsuccessful: cannot open in browser");
	Attempt to utilize questionnaire
	If (product_crashed == True ||
	product_not_respond == True):
	Exit("Test unsuccessful: product crashed, or has
		• Assumes that to be successful in opening it in the specified browser, the questionnaire fill-in, as well as any functions that are included in product_functions are successful before determining that the test passes.

	not responded); If (product_functions_work != True): Exit("Test unsuccessful: at least one of the product functions could not be executed); Exit("Test successful");	
PER-1-TEST	Prepare a timer For each desired browser type: Open the browser Open the product using the browser Enter test values into the questionnaire Record timer start time Start timer Wait until product done loading/responds Record timer end time timer_diff = timer_end_time - timer_start_time If (timer_diff > 10 seconds): Exit("Test unsuccessful: took > 10 seconds"); Exit("Test Successful for all browsers!");	• Assumes that there is a way to obtain a timer that is capable of timing the progress of each item being tested

PER-1.1-TEST	For each desired browser type:
	Open the browser
	Open the product using the browser
	Enter test values into the questionnaire
	If (user_submit):
	If (API_data_not_done_caching):
	Exit("Test unsuccessful: caching not completed by time user submit preferences into questionnaire");
	Exit("Test Successful for all browsers!");
		• Assumes that there is a way to check whether user has submitted data
		• Assumes that there is a way to see whether the API data has finished caching

FUNC-1-TEST	Create array_of_pieces_of_info_product_should_provide For any specified desktop browser: Open the browser Attempt to open product using browser If (product_opened == False): Exit("Test unsuccessful: cannot open on browser");	• Assumes that there is a way to create an array that contains all the pieces of information that the product requires • Assumes for the

	If (sliders_present == False):
	Exit("Test unsuccessful: no sliders present");
	// Otherwise: collect user preferences
	If (user_inputs == NULL):
	Exit("Test unsuccessful: unable to process user inputs");
	// Otherwise: process & load map
	If (map_present == False):
	Exit("Test unsuccessful: No map present");
	If (no_info_on_mouse_click == True):
	Exit("Test unsuccessful: No information provided to user on mouse click");
	//Otherwise:time to check if it has all information, as
	//specified in the
	//array_of_pieces_of_info_product_should_provide
	
	For item in
	array_of_pieces_of_info_product_should_provide:
	If (item != present_to_user_on_mouse_click):
	Exit("Test unsuccessful: " +
	item.toString() + " not presented to user on mouse click");
	Exit("Test successful: all items are presented to user on mouse click");
		product to provide information it must also be capable of opening, have sliders present, be able to collect user inputs, have the ability to display a map, and be able to display the information on a mouse click
		● Assumes that each piece of information, if available, must be presented to the user on a mouse-click event

FUNC-2-TEST	For each desired browser type: Open the browser Open the product using the browser Enter test values into the questionnaire For items in boundary: For each functionality in functionality_requirement: If (functionality_error_for_item): Exit("Test unsuccessful: functionality error for an item, either country, state or county, inside the boundary"); Exit("Test successful: all items in boundary meet all the functionalities in the functionality requirement");	● Assumes that the is a way to obtain a list of functionality requirements ● Assumes that there is a way to test for whether there is a functionality error for any item, county or state or country passed into the boundary

FUNC-2.1-TEST	For each functionality in functionality_requirement:
	For each state in lower_fourty_eight_states:
	works = test_functionality(functionality, state);
	If (!works):
	Exit("Test unsuccessful: functionality not working on state level");
	Exit("Test successful: all functionalities work in all lower forty eight states");
		• Assumes that tester understands what qualifies as and how to obtain basic information about the lower forty eight states

FUNC-2.2-TEST	For each functionality in functionality_requirement:
	For each state in lower_fourty_eight_states:
	works = test_functionality(functionality, state);
	If (!works):
	Exit("Test unsuccessful: functionality not working on state level");
	works_on_county_level = test_functionality(functionality, state, county);
	If (!works_on_county_level):
	Exit("Test unsuccessful: functionality not working on county level");
	Exit("Test successful: all functionalities work in all lower forty eight states down to the county level");
		• Assumes that tester understands what qualifies as and how to obtain basic information about the lower forty eight states
		• Assumes that it is possible to test on the county level for each state being tested.

FUNC-3-TEST	For all specified desktop browser clients:
	Open the browser
	Attempt to open product using browser
	If (product_opened == False):
	Exit("Test unsuccessful: cannot open on browser");
	// start and enter questionnaire page
	// count questionnaire questions
	If (num_questionnaire_questions >= 6):
	Exit("Test unsuccessful: questionnaire does not meet minimum of 6 questions");
	//topics_to_collect_user_pref is an array with
	//array_of_pieces_of_info_product_should_provide
	If (questions not in topics_to_collect_user_pref):
	Exit("Test unsuccessful: questionnaire topics do not involve obtaining information about user's
		• Assumes that the product has a questionnaire page
		• Assumes there is a way to check whether the questions are relevant to user preferences

	preferences."); Exit("Test successful: Questionnaire meets minimum of 6 questions");	

FUNC-3.X-TEST	For all specified desktop browser clients:
	Open the browser
	Attempt to open product using browser
	If (product_opened == False):
	Exit("Test unsuccessful: cannot open on browser");
	// start and enter questionnaire page
	//topics_to_collect_user_pref is an array with
	//array_of_pieces_of_info_product_should_provide
	//or general stuff for questions -- it's basically all the
	//stuff that the questionnaire should inquire on
	
	If (question_topics not in topics_to_collect_user_pref):
	Exit("Test unsuccessful: questionnaire topics do not involve obtaining information about user's preferences.");
	
	~~For topics in topics_to_collect_user_pref:~~
	~~If (topics not in question_topics):~~
	~~questions.append(~~
	~~generatequestion(topics));~~
	~~question_topics.append(topics);~~
	// count questionnaire questions
	If (num_questionnaire_questions < 6):
	Exit("Test unsuccessful: questionnaire does not meet minimum of 6 questions");
	
	Exit("Test successful: Questionnaire meets minimum of 6 questions and the questions of required topics are all met");
		• Assumes that the browsers that are used with the product are the ones supported by the product
		• Assumes there is a way to check whether the questions are relevant to user preferences

~~FUNC-4-TEST~~		
FUNC-5-TEST	For each desired browser type: Open the browser Open the product using the browser	• Assumes that obtaining items from the

	Enter test values into the questionnaire if (county_ suggestions_based_on_user_prefer_absent): Exit("Test unsuccessful: no county suggestions based on user preferences present"); Exit("Test successful: county suggestions based on user preferences are present");	questionnaire has no problems, and the only thing that needs to be tested is whether county suggestions can be provided based on the user preferences
~~FUNC-5.2-TEST~~		

GUI-1-TEST	For any specified desktop browser:
	Open the browser
	Attempt to open product using browser
	If (product_opened == False):
	Exit("Test unsuccessful: cannot open on browser");
	If (sliders_present == False):
	Exit("Test unsuccessful: no sliders present");
	// Otherwise: collect user preferences
	Exit("Test successful: sliders present");
		• Assumes that the browsers that are used with the product are the ones supported by the product

GUI-2-2.1-TEST	For any specified desktop browser:
	Open the browser
	Attempt to open product using browser
	If (product_opened == False):
	Exit("Test unsuccessful: cannot open on browser");
	If (sliders_present == False):
	Exit("Test unsuccessful: no sliders present");
	// Otherwise: collect user preferences
	If (user_inputs == NULL):
	Exit("Test unsuccessful: unable to process user inputs");
	// Otherwise: process & load map
	If (map_present == False):
	Exit("Test unsuccessful: No map present");
		• Assumes that the browsers that are used with the product are the ones supported by the product
		• Assumes that for GUI-2 and GUI-2.1 requirements to be fulfilled:
		○ sliders must be present
		○ user inputs must not be null

	If (no_info_on_mouse_click == True): Exit("Test unsuccessful: No information provided to user on mouse click"); Exit("Test successful: map is interactive and responds to mouse click!");	○ mouse click must be able to present the information

GUI-2.2-2.3-TEST	Set working_with_check_gradients;
	For any specified desktop browser:
	Open the browser
	Attempt to open product using browser
	If (product_opened == False):
	Exit("Test unsuccessful: cannot open on browser");
	If (sliders_present == False):
	Exit("Test unsuccessful: no sliders present");
	// Otherwise: collect user preferences
	If (user_inputs == NULL):
	Exit("Test unsuccessful: unable to process user inputs");
	// Otherwise: process & load map
	If (map_present == False):
	Exit("Test unsuccessful: No map present");
	//Otherwise: highlight map based on user preferences
	//if using gradients can color gradient here
	If (map_highlighted == False):
	Exit("Test unsuccessful: map not highlighted);
	If (working_with_gradients):
	//adjust map colors according to gradients (or
	//just color them with gradients to start with
	//Otherwise: check if highlighting correct.
	//perhaps pass in working_with_gradients into call
	//for check: then if working_with_gradients we do
	//extra check to see if it is correct according to
	//gradients as well
	If (map_highlighted_match_user_prefereces == False):
	Exit("Test unsuccessful: highlighted regions do not correspond to user preferences");
	
	Exit("Test successful: map is highlighted according to user
		● Assumes that the browsers that are used with the product are the ones supported by the product
		● Assumes that if error occurs in getting user inputs , collected user inputs are null
		● Assumes that there is a way to check whether the map is highlighted, and whether there is a way to check whether the gradients have colored correctly, in accordance to the user inputs

	preferences”);	

GUI-3-TEST	Set working_with_check_gradients;
	For any specified desktop browser:
	Open the browser
	Attempt to open product using browser
	If (product_opened == False):
	Exit(“Test unsuccessful: cannot open on browser”);
	If (sliders_present == False):
	Exit(“Test unsuccessful: no sliders present”);
	// Otherwise: collect user preferences
	If (user_inputs == NULL):
	Exit(“Test unsuccessful: unable to process user inputs”);
	// Otherwise: process & load map
	If (map_present == False):
	Exit(“Test unsuccessful: No map present”);
	If (sliders_not_present_on_map_screen):
	Exit(“Test unsuccessful: No sliders present on the map screen for users to modify their preferences after the initial questionnaire ”);
	If(cannot_modify_user_inputs_with_the_sliders()):
	Exit(“Test unsuccessful: Unable to allow users to modify their preferences after the initial questionnaire ”);
	Exit(“Test successful: users are able to modify their preferences after the initial questionnaire via sliders”);
		● Assumes that the browsers that are used with the product are the ones supported by the product
		● Assumes that screens after the questionnaire have a map
		● Assumes that the map must be present for users to modify their preferences after the initial questionnaire
		● Assumes that there is a method for users to modify their initial inputs into the questionnaire that can be used to obtain a boolean

Common Assumptions and Constraints Among the Tests:
Device used, to perform the test, is one that is capable of supporting one of the browsers supported by the product

4. Requirements traceability

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

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. 2.1

5. Notes

Included in this section are all terms and acronyms used in this document:

AngularJS	Angular Javascript web framework
CSCI	Computer Software Configuration Item
SRS	Software Requirement Specification
SDD	Software Design Description

A. Appendixes

Appendixes may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided. Appendixes may be bound as separate documents for ease in handling. Appendixes shall be lettered alphabetically (A, B, etc.).

DESCRIPTION/PURPOSE

The Software Test Description (STD) describes the test preparations, test cases, and test procedures to be used to perform qualification testing of a Computer Software Configuration Item (CSCI) or a software system or subsystem.

The STD enables the acquirer to assess the adequacy of the qualification testing to be performed.

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.