**Deliverable**: in a single word document “CSS360\_Testing.docs”

**Description**: The next project task is to plan your tests based on the requirements list your team has developed and the design/implementation plans your group created.

Your testing document will include the following parts:

1. Based on the requirements list, either use one of the scenarios you created earlier or create a better one (based on your latest thinking and customer input) for your acceptance test plan. Specify which user requirements you are testing with this scenario. Make sure that you follow the format of the scenario (see below).
2. Based on the design/implementation plans, choose two attacks discussed in “How to Break Software” (see deck in Readings folder) to test your user interface. Why did you choose these attacks?
3. Based on the design/implementation plans, create two test cases, based on the lectures or textbook examples. Each test case will contain input test data and expected output (see Lecture slides and chapter 8.1 on input domain - equivalence partitioning, boundary value analysis etc.).

{beginning of submission}

############################################################################

**Scenario**

Example Scenario: Requesting Parking Information

Actors: UW Bothell student

Pre-conditions: User is about to leave her house to head to UW Bothell, doesn’t have a lot of time to spare before class and needs to figure out where she should park her car so she’s not late.

Scenario:

|  |  |
| --- | --- |
| 1) The user requests UWB parking information.  2) The system asks whether the user would rather view the parking information on their smartphone or hear the parking information on their smart speaker. | |
| 3) The user requests to view the information.  4) The system opens the map display on the user’s smartphone that shows all the general parking spots.  5) The user interacts with the map display by clicking on each parking lot to be able to see detailed information about the availability, the daily traffic in and out of the lot, and the system’s estimate on how full it will be at any given time.  6) The user has the option to save the parking information to their smartphone in case they forget anything.  7) The user closes the smartphone application and heads to school.  8) If there are any significant changes in Google Student’s estimates or in UWB’s parking data, the user gets a notification on their smartphone with the new updated information. | 3) The user requests to hear the information.  4) The system starts listing off the amount of general parking spots open in each lot and gives a recommendation on the best lot to pick for the day based off of the amount of traffic in and out of there.  5) The user thanks the voice assistant application and heads to school.  6) If there are any significant changes in Google Student’s estimates or in UWB’s parking data, the user gets a notification on their smartphone with the new updated information. |

Conversation:

User: “Ok Google, I need to get the UWB parking information.”

System: “Would you rather view or hear the information?”

User: “View!”

System: displays parking map on smartphone

User: “Ok Google, I need to get the UWB parking information.”

System: “Would you rather view or hear the information?”

User: “Hear!”

System: “The South parking lot has 140 spots open and the North parking lot has 67 spots open, however I estimate that in the next hour, the North parking lot will have more open spots than the South parking lot. I recommend parking in the North parking lot today.”

User: “Awesome, thanks Google!”

System: “My pleasure!”

Post-conditions: User is able to arrive to class on time because Google Student has given them information about the best parking lot to use to be efficient.

**Attacks Chosen**

**UI Attack #16: Find features that share data or interact poorly.**

We chose this attack because our product interacts with a lot of servers and APIs that are necessary for the functionality it provides. Having errors with integration could mean that most of our services would not work, so testing to make sure that our CANVAS integration and the data shared by UWB for parking information is essential.

**UI Attack #10: Force the screen to refresh.**

This attack forces the screen to refresh to check for errors when the user will specifically request to view the parking information or travel information. This specific request opens the Google Student map display on their smartphone. This map should be updating every 5 minutes when the application is opened, meaning that it should be refreshing by itself every 5 minutes as well. Making sure that once the application refreshes it has all of the right information and the display is correct is extremely important to the user experience.

**Test Cases**

After completing unit testing within each individual component, we will design multiple component and system test cases to verify the functionality of each component, and of the components interacting with each other as a system.

Component Test Case - should focus on showing that the component interface(s) behave according to its specification

System Test Case - should focus on testing the interactions between the components and objects that make up a system

< I FOUND THIS LINK ARNIE POSTED <https://reqtest.com/testing-blog/learn-how-to-write-effective-test-cases/>>

## Component Test Case

|  |  |
| --- | --- |
| **ID** | 01 |
| **Title** | Parking Availability Component Through Android Mobile App |
| **Priority** | High |
| **Execution Result** | Pass |
| **Purpose** | Verifies that the parking availability component operates as intended when accessed through our mobile application for Android phones |
| **Created By** | Taylor Eyler, 2019-06-06 |

|  |  |  |
| --- | --- | --- |
| **Step** | **Description** | **Expected Result** |
| 1 | Launch the Google Student mobile app |  |
| 2 | Navigate to the parking screen within the app | Heat map is visible; “Last updated” within last 5 min |
| 3 | Click ‘refresh’ on the parking screen | Updated heat map is shown; “Last updated” is current time |

## System Test Case

|  |  |
| --- | --- |
| **ID** | 02 |
| **Title** | ‘Student Digest’ Screen on Android Mobile App |
| **Priority** | High |
| **Execution Result** | Pass |
| **Purpose** | Verifies the functionality of the ‘student digest’ screen on the front page of the mobile app, which incorporates information from the CANVAS, travel time, parking availability, and food truck components |
| **Created By** | Taylor Eyler, 2019-06-06 |

|  |  |  |
| --- | --- | --- |
| **Step** | **Description** | **Expected Result** |
| 1 | Launch the Google Student (GS) mobile app |  |
| 2 | Ensure you are logged in to CANVAS dummy profile |  |
| 3 | Click the ‘refresh’ button on the GS home page | Data is updated in >10s |
| 4 | Ensure the list of upcoming CANVAS assignments appears |  |
| 5 | Ensure correct travel time is displayed |  |
| 6 | Ensure the parking heat map is displayed | “Last updated” time is within the last five minutes |
| 7 | Ensure the correct food truck for that day is displayed |  |

############################################################################

{end of submission}