Lab 3 – ParkODU Prototype Product Specification

Team Gold

CS 411

Professor Thomas J. Kennedy

2 April 2018

Version 1

Table of Contents

1. Objectives

The purpose of this Test Plan and Procedure is to test the operation and performance of the ParkODU Prototype. It includes tests for four major categories - User Access, User Interface (UI), JUnit, and User Experience (UX).

User access tests will cover user login and access control defined by user roles. User interface tests will cover the presentation format of information and the responses to user actions. JUnit tests will be used to test the functionality of the back-end controllers and services to ensure that the controllers and services properly manipulate data models and return correct information. User experience tests will cover all applicable use cases and the intuitiveness of the user interfaces in real use scenarios.

2. References

Lab 1 - ParkODU Description. Version 2. (2018, February). Team Gold. CS411W

Lab 2 - ParkODU Prototype Product Specification. Version 2. (2018, March). CS411W

3. Test Plan

The test plan of the prototype will cover the testing approach, the identification of tests, the test schedule, fault reporting and data recording, resource requirements, test environment, and test responsibilities.

[This space intentionally left blank]

3.1 Testing Approach

Performance of the prototype will be evaluated by the following categories of unit and feedback testing:

1. **User Access** Tests verify that authentication mechanisms have been properly implemented to allow users to log into the web application and are assigned permissions appropriate to their role.

2. **UI** tests verify that HTML elements display properly and function appropriately.

3. **JUnit** Tests verify code quality and functionality of Controller and Model methods.

4. **User Experience (UX)** Tests verify the usability of accessibility of all web pages to strive for compliance with Section 508 of the Rehabilitation Act of 1973.

[This space intentionally left blank]

3.2 Identification of Tests

Each test will be identified by category and test case number.

Procedure ID	Category	Test Case	Name	Objective
1	User Access	1.1	User Login Prompt	Verify that users are prompted for login when they first visit the website.
		1.2	User Logout Option	Verify that users have the capability to logout from the website.
		1.3	Admin Access	Verify that users logged in as Admin have access to Settings where they can modify configurations.
		1.4	User Access	Verify that regular users can access ParkODU and they can view and modify their profile.
		1.5	Anonymous Access	Verify that anonymous users can access ParkODU without any credentials but they have no access to their profile
2	UI	2.1	Create Space	Verify that the Create Parking Space page displays all elements and responds to user actions correctly.
		2.2	Edit Space	Verify that the Edit Parking Space page displays all elements and responds to user actions correctly.
		2.3	Delete Space	Verify that the Delete button and Modal are displayed and responds to user actions correctly.
		2.4	Create Floor	Verify that Create Floor page displays all elements and responds to user actions correctly.
		2.5	Edit Floor	Verify that the Edit Floor page displays all elements and responds to user actions correctly.
		2.6	Delete Floor	Verify that the Delete button and Modal are displayed and responds to user actions correctly.
		2.7	Create Garage	Verify that the Create Garage page displays all elements and responds to user actions correctly.
		2.8	Edit Garage	Verify that the Edit Garage page displays all elements and responds to user actions correctly.
		2.9	Delete Garage	Verify that the Delete button and Modal are displayed and responds to user actions correctly.
		2.10	Search	Verify that search page displays all elements and responds to user actions correctly.
		2.11	Directions Display	Verify that Navigate page displays all elements and responds to user actions correctly.

Procedure ID	Category	Test Case	Name	Objective
		2.12	User Preferences	Verify that the User Settings page displays all elements and responds to user actions correctly.
		2.13	Events Notification	Verify that event notifications are displayed and respond to user actions correctly.
		2.14	Create Event	Verify that the Create Events page in Admin Settings displays all elements and responds to user actions correctly.
		2.15	Edit Event	Verify that the Edit Events page in Admin Settings displays all elements and responds to user actions correctly.
		2.16	Delete Event	Verify that the Delete button and Modal are displayed and responds to user actions correctly.
		2.17	Charts	Verify that the Charts page displays all elements and responds to user actions correctly.
3	JUnit	3.1	Index Garage	Verify that the controller logic to process an Index Garage request is executed properly.
		3.2	Create Garage	Verify that the controller logic to process a Create Garage request is executed properly.
		3.3	Edit Garage	Verify that the controller logic to process an Edit Garage request is executed properly.
		3.4	Delete Garage	Verify that the controller logic to process a Delete Garage request is executed properly.
		3.5	Index Floor	Verify that the controller logic to process an Index Floor request is executed properly.
		3.6	Create Floor	Verify that the controller logic to process a Create Floor request is executed properly.
		3.7	Edit Floor	Verify that the controller logic to process an Edit Floor request is executed properly.
		3.8	Delete Floor	Verify that the controller logic to process a Delete Floor request is executed properly.
		3.9	Index Parking Space	Verify that the controller logic to process an Index Parking Space request is executed properly
		3.10	Create Parking Space	Verify that the controller logic to process a Create Parking Space request is executed properly.
		3.11	Edit Parking Space	Verify that the controller logic to process an Edit Parking Space request is executed properly.

Procedure ID	Category	Test Case	Name	Objective
		3.12	Delete Parking Space	Verify that the controller logic to process a Delete Parking Space request is executed properly.
		3.13	Building Index	Verify that the controller logic to process an Index Building request is executed properly.
		3.14	Create Building	Verify that the controller logic to process a Create Building request is executed properly.
		3.15	Edit Building	Verify that the controller logic to process an Edit Building request is executed properly.
		3.16	Delete Building	Verify that the controller logic to process a Delete Building request is executed properly.
		3.17	Index User	Verify that the controller logic to process an Index User request is executed properly.
		3.18	Create User	Verify that the controller logic to process a Create User request is executed properly.
		3.19	Edit User	Verify that the controller logic to process an Edit User request is executed properly.
		3.20	Delete User	Verify that the controller logic to process a Delete User request is executed properly.
		3.21	Register User	Verify that the controller logic to process a Register User request is executed properly.
		3.22	Confirm User	Verify that the controller logic to process a Confirm User request is executed properly.
		3.23	Directions	Verify that the controller logic to process an Index Directions request is executed properly.
		3.24	User Model	Verify the completeness of the User model interface, that all User.java get methods properly return requested attributes, and that all set methods properly update the attributes
4	UX		Accessibility	To make sure the application conforms to the font size readability, color contrast, images, and ease of navigation according to section 508.

[This space intentionally left blank]

3.3 Test Schedule

A full test will take an approximate time of one hour to complete.

Start Time (hour:min)	Duration (minutes)	Test Case	Description	Comments
0:00	1	1.1	User Login Prompt	
0:01	1	1.2	User Logout Option	
0:02	5	1.3	Admin Access	
0:07	5	1.4	User Access	
0:11	3	1.5	Anonymous Access	
0:14	1	2.1	Create Space	
0:15	2	2.2	Edit Space	
0:17	1	2.3	Delete Space	
0:18	3	2.4	Create Floor	
0:21	1	2.5	Edit Floor	
0:22	1	2.6	Delete Floor	
0:23	3	2.7	Create Garage	
0:26	3	2.8	Edit Garage	
0:29	1	2.9	Delete Garage	
0:30	2	2.10	Search	
0:32	3	2.11	Direction display	
0:35	5	2.12	User Preferences	
0:40	5	2.13	Events Notification	
0:45	2	2.14	Create Event	
0:47	1	2.15	Edit Event	
0:48	1	2.16	Delete Event	
0:49	5	2.17	Charts	
0:54	2	2.18	Register User	
0:56	2	2.19	Confirm User	

3.4 Fault Reporting and Data Recording

The results of unit tests will verify the functionality of the system components using Cobertura Test Coverage Report to generate the JUnit test coverage results in HTML format. JUnit also generates a Test Summary report that displays the success rate and how many tests passed, failed, and ignored. All the failures that occur when the program is executed will be viewable from the interface being used. Tests requiring user feedback responses will be recorded by a member of Team Gold during the testing demonstration.

3.5 Resource Requirements

For the testing procedure, sample historical data generated by the simulator for chart demonstration must be made available. This data is based on real-world parking trends. Database objects must exist that represent ODU: buildings, garages, floors, parking spaces, parking space types, parking permit types, events, and users. An active connection to the Internet with a modern browser is also necessary.

3.6 Test Environment

The test will be conducted remotely on the ParkODU website using virtual Windows 7 machine. The testing will be done by Team Gold. Tests may or may not require user feedback which will be conducted in a recitation classroom in the Gornto Telecommunications Building at Old Dominion University.

3.7 Test Responsibilities

Each member of Team Gold will perform a role necessary for the completion of the tests.

Team Member	Role	Responsibilities
Asante, Isaac	Consumer	Admin UI navigation
Coughenour, Cody	Test Manager	Lead test demonstration team
Mason, Imani	Technical Writer	Generate test reports
Mokha, Sangeet	Consumer	User UI navigation
Park, Michael	Technical Advisor	Provide technical assistance
Sheikh, Ahsif	Presenter	Explain test results
Silverio, Gerard	Technical Advisor	Provide technical assistance
Stevenson, Matthew	Presenter	Explain test results

[This space intentionally left blank]

4. Test Procedures

The following sections contain all test cases for this test procedure. Each test case includes the category, number, fulfilled requirement, a description, name, version, author, purpose, setup conditions, steps, and expected results of the test case. There are also spaces for comments and indicating if the test passed or failed.

[This space intentionally left blank]

4.1 User Access Tests

User access tests are involved with the views users see based on their access level.

<table>
<tr><td>Test Category:
User Access</td><td colspan="3">Description: Login prompted for user when visiting ParkODU</td></tr>
<tr><td>Test Case:
1.1</td><td>Case Name:
User Login Prompt</td><td>Version:
1.0</td><td>Written by:
Imani Mason</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.8.1</td><td colspan="3">Purpose: Verify that users are prompted for login when they first visit the website.</td></tr>
<tr><td colspan="4">Setup Conditions:
• Open ParkODU Web Application
• Navigate to the User Login Page</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1.Navigate to ParkODU</td><td></td><td></td><td>ParkODU Welcome Screen displayed</td></tr>
<tr><td>2.Click Login</td><td></td><td>User is prompted to enter username and password</td><td>Login screen displayed</td></tr>
</table>

<table>
<tr><td>Test Category:
User Access</td><td colspan="3">Description: On the User Login page users can logout</td></tr>
<tr><td>Test Case:
1.2</td><td>Case Name:
User Login Option</td><td>Version:
1.0</td><td>Written by:
Imani Mason</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.8.2</td><td colspan="3">Purpose: Verify that users have the capability to logout from the website.</td></tr>
<tr><td colspan="4">Setup Conditions:
• Open ParkODU Web Application
• Navigate to the User Login Page</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1.Click Login</td><td></td><td></td><td>Login screen displayed</td></tr>
<tr><td>2. Login in with username and password</td><td></td><td></td><td>"Welcome User!" displayed at top of screen</td></tr>
<tr><td>3.Click settings</td><td></td><td></td><td>Log Out option is displayed</td></tr>
<tr><td>4. Click Log Out</td><td></td><td></td><td>User is logged out of account and Login page is displayed</td></tr>
</table>

<table>
<tr><td>Test Category:
User Access</td><td colspan="3">Description: As an admin user on the Configuration page admin can modify configurations</td></tr>
<tr><td>Test Case:
1.3</td><td>Case Name:
Admin Access</td><td>Version:
1.0</td><td>Written by:
Imani Mason</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.7</td><td colspan="3">Purpose: Verify that users logged in as Admin have access to Settings where they can modify configurations.</td></tr>
<tr><td colspan="4">Setup Conditions:
• Open ParkODU Web Application
• Navigate to the Admin Page
• Navigate to the Configuration Page</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1.Click Login</td><td></td><td></td><td>Login screen displayed</td></tr>
<tr><td>2. Login in with username and password</td><td></td><td>Login as admin</td><td>“Welcome Admin!” displayed at top of screen</td></tr>
<tr><td>3. Navigate to the Configurations tab</td><td></td><td></td><td>Configuration options are displayed</td></tr>
<tr><td>4. Click a configuration option</td><td></td><td></td><td>Create, edit, and delete settings are displayed</td></tr>
</table>

[This space intentionally left blank]

<table>
<tr><td>Test Category:
User Access</td><td colspan="3">Description: Users of ParkODU can view and edit profiles</td></tr>
<tr><td>Test Case:
1.4</td><td>Case Name:
User Access</td><td>Version:
1.0</td><td>Written by:
Imani Mason</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.2</td><td colspan="3">Purpose: Verify that regular users can access ParkODU and can view and modify their profile.</td></tr>
<tr><td colspan="4">Setup Conditions:
• Open ParkODU Web Application
• Navigate to the User Login Page
• Navigate to User Profile Page</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1.Click Login</td><td></td><td></td><td>Login screen displayed</td></tr>
<tr><td>2. Login in with username and password</td><td></td><td></td><td>“Welcome User!” displayed at top of screen</td></tr>
<tr><td>3.Navigate to Settings Page</td><td></td><td></td><td>Profile option displayed</td></tr>
<tr><td>4. Click Profile</td><td></td><td></td><td>View and Modify options are displayed</td></tr>
</table>

[This space intentionally left blank]

<table>
<tr><td>Test Category:
User Access</td><td colspan="3">Description: Users can access ParkODU without credentials but cannot access profile</td></tr>
<tr><td>Test Case:
1.5</td><td>Case Name:
Anonymous Access</td><td>Version:
1.0</td><td>Written by:
Imani Mason</td></tr>
<tr><td>Requirements Fulfilled:</td><td colspan="3">Purpose: Verify that anonymous users can access ParkODU without any credentials but they have no access to their profile</td></tr>
<tr><td colspan="4">Setup Conditions:
• Open ParkODU Web Application</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1.Navigate around ParkODU</td><td></td><td></td><td>Can access: Garage (Details, Floor Details), Search (Results, Navigate), Chart, Login, and Event Notifications</td></tr>
</table>

[This space intentionally left blank]

4.2 UI Tests

UI tests involve testing the functionality of each page within the application including proper navigation.

Test Category: User Interface	Description: On the Create Parking Spaces page in Admin Settings spaces can be created.		
Test Case: 2.1	Case Name: Create Space	Version: 1.0	Written by: Cody Coughenour
Requirements Fulfilled: 3.1.1.7.2.7	Purpose: Verify that the Create Parking Space page displays all elements and responds to user actions correctly.		
Setup Conditions: · Open ParkODU Web Application · Navigate to the Admin Page · Log in as an Admin · Navigate to the Settings page · Navigate to the Configure Spaces Page			
Test Case Activity	**Pass/Fail**	**Comments**	**Expected Result**
1. Click Create button			Links to create page
2. Enter a value for space Number		Accepts text	Field should be editable
3. Select a value for Permit Type			All Permit Types displayed
4. Select a value for Space Type			All Space Types displayed
5. Click the Submit button			Returns to list of spaces for that garage and floor
6. Click the Reset button			Resets all values to defaults
7. Click the Back button			Returns to list of spaces for that garage and floor

[This space intentionally left blank]

<table>
<tr><td>Test Category:
User Interface</td><td colspan="3">Description: On the Configure Spaces page in Admin Settings spaces can be edited.</td></tr>
<tr><td>Test Case:
2.2</td><td>Case Name:
Edit Space</td><td>Version:
1.0</td><td>Written by:
Cody Coughenour</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.7.2.8</td><td colspan="3">Purpose: Verify that the Edit Parking Space page displays all elements and responds to user actions correctly.</td></tr>
<tr><td colspan="4">Setup Conditions:
· Open ParkODU Web Application
· Navigate to the Admin Page
· Log in as an Admin
· Navigate to the Settings page
· Navigate to the Configure Spaces Page
· Select a Floor of spaces to edit</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1. Change the value of Number</td><td></td><td>A confirmation message should appear. Last Updated field should change. This comment applies to all other activities.</td><td>Field should be editable</td></tr>
<tr><td>2. Change the Permit Type with drop down menu</td><td></td><td></td><td>All Permit Types displayed</td></tr>
<tr><td>3. Change the Space Type</td><td></td><td></td><td>All Space Types displayed</td></tr>
<tr><td>4. Click the Available switch</td><td></td><td></td><td>Switch changes from blue to grey on click</td></tr>
<tr><td>5. Click Reload button</td><td></td><td></td><td>Refreshes page</td></tr>
</table>

[This space intentionally left blank]

Test Category: User Interface	Description: On the Configure Spaces page in Admin Settings spaces can be deleted.		
Test Case: 2.3	Case Name: Delete Space	Version: 1.0	Written by: Cody Coughenour
Requirements Fulfilled: 3.1.1.7.2.9	Purpose: Verify that the Delete button and Modal displays all elements and responds to user actions correctly.		
Setup Conditions: · Open ParkODU Web Application · Navigate to the Admin Page · Log in as an Admin · Navigate to the Settings page · Navigate to the Configure Spaces Page · Select a Floor of spaces to edit			
Test Case Activity	Pass/Fail	Comments	Expected Result
1. Click the delete button		Verify warning shows correct space number	Displays warning, Delete, and Close
2. Click Close			Closes warning
3. Click the X			Closes warning
4. Click Delete			Closes dialog and refreshes page

[This space intentionally left blank]

<table>
<tr><td>Test Category:
User Interface</td><td colspan="3">Description: On the Create Floor page in Admin Settings a floor can be created.</td></tr>
<tr><td>Test Case:
2.4</td><td>Case Name: Create Floor</td><td>Version:
1.0</td><td>Written by:
Cody Coughenour</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.7.2.4</td><td colspan="3">Purpose: Verify that the Create Floor page in Admin Settings displays all elements and responds to user actions correctly.</td></tr>
<tr><td colspan="4">Setup Conditions:
Open ParkODU Web Application
Navigate to the Admin Page
Log in as an Admin
Navigate to the Settings page
Navigate to the Configure Floors Page
Select a garage</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1. Click Create button</td><td></td><td></td><td>Links to create page</td></tr>
<tr><td>2. Type in a Number</td><td></td><td>Shows placeholder text prior</td><td>Editable</td></tr>
<tr><td>3.Type in a Description</td><td></td><td>Shows placeholder text prior</td><td>Editable, Resizable</td></tr>
<tr><td>4. Type in Total Spaces</td><td></td><td>Shows placeholder text prior</td><td>Numbers only, Editable, Can increment with arrow</td></tr>
<tr><td>5. Click Reset</td><td></td><td></td><td>All values cleared</td></tr>
<tr><td>6. Click Back</td><td></td><td></td><td>Returns to list of floors for that garage</td></tr>
<tr><td>7. Click Submit</td><td></td><td></td><td>Returns to list of floors for that garage</td></tr>
</table>

[This space intentionally left blank]

Test Category: User Interface	Description: On the Edit Floor page in Admin Settings a floor can be edited.		
Test Case: 2.5	Case Name: Edit Floor	Version: 1.0	Written by: Cody Coughenour
Requirements Fulfilled: 3.1.1.7.2.5	Purpose: Verify that the Edit Floor page in Admin Settings displays all elements and responds to user actions correctly.		
Setup Conditions: • Open ParkODU Web Application • Navigate to the Admin Page • Log in as an Admin • Navigate to the Settings page • Navigate to the Configure Floors Page • Select a garage			
Test Case Activity	Pass/Fail	Comments	Expected Result
1. Click Edit button for a floor			Links to Edit Page
2. Type in a Number		Does not allow duplicate. Try adding duplicate.	Editable, Error message if necessary
3.Type in a Description			Editable, Displays description
4. Type in Total Spaces			Read Only, Displays Total Spaces
5. Click Reset			All values returned to original values
6. Click Back			Returns to list of floors for that garage
7. Click Submit			Returns to list of floors for that garage

[This space intentionally left blank]

Test Category: User Interface	Description: On the Configure Floor page in Admin Settings a floor can be deleted.		
Test Case: 2.6	Case Name: Delete Floor	Version: 1.0	Written by: Cody Coughenour
Requirements Fulfilled: 3.1.1.7.2.6	Purpose: Verify that the Delete button and Modal displays all elements and responds to user actions correctly.		
Setup Conditions: • Open ParkODU Web Application • Navigate to the Admin Page • Log in as an Admin • Navigate to the Settings page • Navigate to the Configure Floors Page • Select a garage			
Test Case Activity	Pass/Fail	Comments	Expected Result
1. Click Delete		Verify Floor Number	Displays warning, delete, close
2. Click Close			Closes warning
3. Click the X			Closes warning
4. Click Delete			Closes dialog and refreshes page

[This space intentionally left blank]

Test Category: User Interface	Description: On the Create Garage page in Admin Settings a garage can be created and responds to user actions correctly.		
Test Case: 2.7	Case Name: Create Garage	Version: 1.0	Written by: Cody Coughenour
Requirements Fulfilled: 3.1.1.7.2.1	Purpose: Verify that the Create Garage page in Admin Settings displays all elements.		

Setup Conditions:

- Open ParkODU Web Application
- Navigate to the Admin Page
- Log in as an Admin
- Navigate to the Settings page
- Navigate to the Configure Garages Page

Test Case Activity	Pass/Fail	Comments	Expected Result
1. Click Create button			Links to create page
2. Type in a Name		Shows placeholder text prior. Try duplicate.	Editable, Text, No duplicate Name
3.Type in a Description		Shows placeholder text prior	Editable, Resizable
4. Type in Height Description		Show placeholder text prior	Editable, Text
5. Enter an address		Shows placeholder text prior. Lat/Long are read only.	Editable, Latitude and Longitude should change based on address
6. Click Back			Returns to list of garages
7. Click Submit			Returns to list of garages
8. Click Reset			All values returned to original value.

[This space intentionally left blank]

Test Category: User Interface	Description: On the Edit Garage page in Admin Settings a garage can be edited.		
Test Case: 2.8	Case Name: Edit Garage	Version: 1.0	Written by: Cody Coughenour
Requirements Fulfilled: 3.1.1.7.2.2	Purpose: Verify that the Edit Garage page in Admin Settings displays all elements and responds to user actions correctly.		
Setup Conditions: • Open ParkODU Web Application • Navigate to the Admin Page • Log in as an Admin • Navigate to the Settings page • Navigate to the Configure Garages Page			
Test Case Activity	**Pass/Fail**	**Comments**	**Expected Result**
1. Click Edit button			Links to create page
2. Type in a Name		No duplicate. Try duplicate.	Editable, Text, Displays Name
3.Type in a Description		Can be resized.	Editable, Shows Description
4. Type in Height Description			Editable, Text, Shows Height Description
5. Enter an address		Lat/Long are read only.	Editable, Autocompletes, Latitude and Longitude should change based on address
6. Click Back			Returns to garages
7. Click Submit			Returns to garages
8. Click Reset			All values returned to original value.

[This space intentionally left blank]

Test Category: User Interface	Description: On the Configure Garage page in Admin Settings a garage can be deleted.		
Test Case: 2.9	Case Name: Delete Garage	Version: 1.0	Written by: Cody Coughenour
Requirements Fulfilled: 3.1.1.7.2.3	Purpose: Verify that the Delete button and Modal are displayed correctly.		

Setup Conditions:

- Open ParkODU Web Application
- Navigate to the Admin Page
- Log in as an Admin
- Navigate to the Settings page
- Navigate to the Configure Garages Page

Test Case Activity	Pass/Fail	Comments	Expected Result
1. Click Delete		Verify Garage Name	Displays warning, delete, close
2. Click Close			Closes warning
3. Click the X			Closes warning
4. Click Delete			Closes dialog and refreshes page

[This space intentionally left blank]

Test Category: **User Interface**	**Description: The search page should function and be accessible.**		
Test Case: **2.10**	**Case Name:** **Search**	**Version:** **1.0**	**Written by:** **Sangeet Mokha, Cody Coughenour**
Requirements Fulfilled: **3.1.1.4.2, 3.1.1.4.3, 3.1.1.4.4, 3.1.1.4.5, 3.1.1.4.6**	**Purpose: Verify that search page displays all elements and responds to user actions correctly.**		
Setup Conditions: • **Open ParkODU Web Application** • **Click on Search**			
Test Case Activity	**Pass/Fail**	**Comments**	**Expected Result**
1. Type your starting location		**Has autocomplete**	**Editable**
2. Select all the desired Permit Types			**Multi-selectable, all permit types displayed**
3.Select all the desired Space Type			**Multi-selectable, all space types displayed**
4. Add the minimum available spaces			**Editable, numbers only**
5. Choose the desired destination building			**Displays all Buildings**
5. Click search			**Links to Search Results page**

[This space intentionally left blank]

<table>
<tr><td>Test Category:
User Interface</td><td colspan="3">Description: On the Navigate page, directions are provided to the destination garage</td></tr>
<tr><td>Test Case:
2.11</td><td>Case Name:
Directions Display</td><td>Version:
1.0</td><td>Written by:
Sangeet Mokha, Cody Coughenour</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.5.1</td><td colspan="3">Purpose: Verify that Navigate page displays all elements and responds to user actions correctly.</td></tr>
<tr><td colspan="4">Setup Conditions:
• Open ParkODU Web Application
• Enter valid search criteria
• Click on Search
• Click on Navigate</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1. View Page</td><td></td><td></td><td>Map should be displayed indicating point A and B, directions should be displayed indicating all turns and merges</td></tr>
<tr><td>2. Alter route (click and drag)</td><td></td><td></td><td>Displays new route and directions</td></tr>
<tr><td>3. Click Go</td><td></td><td>Mobile Only</td><td>Starts navigation (voice)</td></tr>
</table>

[This space intentionally left blank]

Test Category: User Interface	**Description: On the User Settings page user's schedule can be imported and preferences set.**		
Test Case: 2.12	**Case Name: User Preferences**	**Version:** 1.0	**Written by:** Imani Mason, Sangeet Mokha, Cody Coughenour
Requirements Fulfilled: 3.1.1.4.1, 3.1.1.2.1, 3.1.1.2.2, 3.1.1.2.3	**Purpose: Verify that the User Settings page displays all elements and responds to user actions correctly.**		
Setup Conditions: • Open ParkODU Web Application • Login as a User • Navigate to the Users Preference Page			
Test Case Activity	**Pass/Fail**	**Comments**	**Expected Result**
1. Import your schedule		These fields should also be manually editable	Displays Schedule, Displays appropriate message
2. Set Address			Autocompletes
3. Set Permit Type(s)			Displays all permit types, multi-selectable
4. Set Space Type(s)			Displays all space types, multi-selectable
5. Click Save Button		Set fields to cause an error and set fields to cause success message	Displays appropriate message

[This space intentionally left blank]

<table>
<tr><td>Test Category:
User Interface</td><td colspan="3">Description: Event Notifications are viewable from the Navigation Bar</td></tr>
<tr><td>Test Case:
2.13</td><td>Case Name: Event Notification</td><td>Version:
1.0</td><td>Written by:
Sangeet Mokha</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.6.1</td><td colspan="3">Purpose: Verify that event notifications are displayed and respond to user actions correctly.</td></tr>
<tr><td colspan="4">Setup Conditions:
• Open ParkODU Web Application</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1.Click on the notification button (bell)</td><td></td><td>Number of unread notifications displayed on bell icon</td><td>Displays events and an optional view all</td></tr>
<tr><td>2. Click Close button</td><td></td><td></td><td>Closes event notifications</td></tr>
<tr><td>3. Click X</td><td></td><td></td><td>Closes event notifications</td></tr>
</table>

[This space intentionally left blank]

Test Category: User Interface	Description: Events can be created on the Create Event Notifications page.		
Test Case: 2.14	Case Name: Create Event Notification	Version: 1.0	Written by: Ahsif Sheikh, Isaac Asante, Cody Coughenour
Requirements Fulfilled: 3.1.1.7.1.1	Purpose: Verify that the Create Events page in Admin Settings displays all elements and responds to user actions correctly.		
Setup Conditions: • Open ParkODU Web Application • Go to the Administrative User Page • Log in as an Administrator • Go to "Settings" • Go to "Configure Event Notifications Page"			
Test Case Activity	Pass/Fail	Comments	Expected Result
1. Click Create			Links to Create page
2. Type in an Event Name			Editable, Text, No duplicates
3. Type in an Event Message			Editable, Text
4. Add Affected Locations			Autocompletes, Dropdown, shows all Garages and Buildings
5. Set Date Time Effective			Date select with calendar or keyboard, time hh:mm:ss
6. Set Tags			Text, add multiple
7. Click Submit			Returns to Index
8. Click Back			Returns to Index

[This space intentionally left blank]

Test Category: **User Interface**	**Description: Events can be edited on the Edit Event Notifications page.**		
Test Case: **2.15**	**Case Name: Edit Event Notification**	**Version:** **1.0**	**Written by:** **Ahsif Sheikh, Isaac Asante, Cody Coughenour**
Requirements Fulfilled: **3.1.1.7.1.2**	**Purpose: Verify that the Edit Events page in Admin Settings displays all elements and responds to user actions correctly.**		
Setup Conditions: • **Open ParkODU Web Application** • **Go to the Administrative User Page** • **Log in as an Administrator** • **Go to "Settings"** • **Go to "Configure Event Notifications Page"**			
Test Case Activity	**Pass/Fail**	**Comments**	**Expected Result**
1. Click Edit			**Links to Edit page**
2. Type in an Event Name			**Editable, Text, No duplicates**
3. Type in an Event Message			**Editable, Text**
4. Add Affected Locations			**Autocompletes, Dropdown, shows all Garages and Buildings**
5. Set Date Time Effective			**Date select with calendar or keyboard, time hh:mm:ss**
6. Set Tags			**Text, add multiple**
7. Click Submit			**Returns to Index**
8. Click Back			**Returns to Index**
9. Click Reset			**Returns fields to original values**

[This space intentionally left blank]

<table>
<tr><td>Test Category:
User Interface</td><td colspan="3">Description: Events can be deleted on the Delete Event Notifications page.</td></tr>
<tr><td>Test Case:
2.16</td><td>Case Name: Delete Event Notification</td><td>Version:
1.0</td><td>Written by:
Ahsif Sheikh, Isaac Asante, Cody Coughenour</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.7.1.3</td><td colspan="3">Purpose: Verify that the Edit Events page in Admin Settings displays all elements and responds to user actions correctly.</td></tr>
<tr><td colspan="4">Setup Conditions:
Open ParkODU Web Application
Go to the Administrative User Page
Log in as an Administrator
Go to “Settings”
Go to “Configure Event Notifications Page”</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1. Click Delete</td><td></td><td></td><td>Opens warning, delete, and close</td></tr>
<tr><td>2. Click Delete again</td><td></td><td></td><td>Closes warning</td></tr>
<tr><td>3. Click X</td><td></td><td></td><td>Closes warning</td></tr>
<tr><td>4. Click Close</td><td></td><td></td><td>Closes warning and refreshes page</td></tr>
</table>

[This space intentionally left blank]

Test Category: User Interface	Description: As an admin user on the Configuration page admin can modify configurations		
Test Case: 2.17	Case Name: Charts	Version: 1.0	Written by: Cody Coughenour
Requirements Fulfilled: 3.1.1.3.2	Purpose: Verify that the Charts page displays all elements and responds to user actions correctly.		
Setup Conditions: • Open ParkODU Web Application • Navigate to the Charts page			
Test Case Activity	Pass/Fail	Comments	Expected Result
1.Set Garage			Shows list of all garages
2. Set Floor Number			Text
3. Set Date			Calendar or keyboard input
4. Click Create			Creates a new chart element along with a delete button
5. Click Delete			Removes chart element

[This space intentionally left blank]

4.3 JUnit Tests

The JUnit Test Framework is used to ensure code quality and the accuracy of ParkODU's algorithms. The test cases below are designed to test the functionality of ParkODU's Controller endpoint and Model methods to confirm the validity of the data returned to the user, that the user is redirected to the proper web page, and if applicable, an appropriate alert message is displayed back to the user. Due to time constraints, a full JUnit test case coverage could not be fully documented.

[This space intentionally left blank]

<table>
<tr><td>Test Category:
JUnit Tests</td><td colspan="3">Description: Provide the capability for an administrative user to add, edit, and delete garages</td></tr>
<tr><td>Test Cases:
3.1, 3.2, 3.3, 3.4</td><td>Case Name:
GarageSettingsController Tests</td><td>Version:
1.0</td><td>Written by:
Gerard Silverio</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.7.2.1
3.1.1.7.2.2
3.1.1.7.2.3</td><td colspan="3">Purpose: Verify that all GarageSettingsController endpoints properly return the correct Thymeleaf template page, the Model contains the necessary objects, and the RedirectAttributes contains the necessary alerts.</td></tr>
<tr><td colspan="4">Setup Conditions:
Open IntelliJ
Open GarageSettingsControllerTests.java
Run ‘GarageSettingsControllerTests’
JUnit Setup Conditions
Create two Garage objects
Mock the GarageRepository class
Mock the GarageRepository:findByKey method to return a Garage
Mock the GarageRepository:findAll method to return a collections of two Garages
Mock the GarageRepository:save method
Mock the GarageRepository:delete method</td></tr>
<tr><th>Test Case Activity</th><th>Pass/Fail</th><th>Comments</th><th>Expected Result</th></tr>
<tr><td>1. TestIndex</td><td></td><td>Tests the index get method</td><td>Returns the "settings/garage/index" Thymeleaf template and a Model that contains a collection of two Garages</td></tr>
<tr><td>2. TestCreate_Get</td><td></td><td>Tests the create get method</td><td>Returns the “settings/garage/create” Thymeleaf template and a Model that contains a new Garage object</td></tr>
<tr><td>3. TestCreate_Post_Success</td><td></td><td>Tests the create post method for a non-duplicate Garage object</td><td>Returns a redirect to the “settings/garage/index” Thymeleaf template, an empty Model, and the success message within the RedirectAttributes</td></tr>
<tr><td>4. TestCreate_Post_Duplicate</td><td></td><td>Tests the create post method for a duplicate Garage object</td><td>Returns the “settings/garage/create” Thymeleaf template, and a Model that contains the submitted Garage object and the error message.</td></tr>
<tr><td>5. TestEdit_Get</td><td></td><td>Tests the edit get method</td><td>Returns the “settings/garage/edit” Thymeleaf template, and a Model that contains an existing Garage object</td></tr>
</table>

6. TestEdit_Post_Success		Tests the edit post method for a non-duplicate Garage object	Returns a redirect to the "settings/garage/index" Thymeleaf template, an empty Model, and the success message within the RedirectAttributes
7. TestEdit_Post_Duplicate		Tests the edit post method for a duplicate Garage object	Returns the "settings/garage/edit" Thymeleaf template, and a Model that contains the submitted Garage object and the error message.
8. TestDelete_Post_Success		Tests the delete post method for a successful attempt to delete a Garage object	Returns a redirect to the "settings/garage/index" Thymeleaf template, an empty Model, and the success message within the RedirectAttributes
9. TestDelete_Post_Fail		Tests the delete post method for a failed attempt to delete a Garage object	Returns a redirect to the "settings/garage/index" Thymeleaf template, an empty Model, and the error message within the RedirectAttributes

[This space intentionally left blank]

<table>
<tr><td>Test Category:
JUnit Tests</td><td colspan="3">Description: Provide the capability for an administrative user to add, edit, and delete floors</td></tr>
<tr><td>Test Cases:
3.5, 3.6, 3.7, 3.8</td><td>Case Name:
FloorSettingsController Tests</td><td>Version:
1.0</td><td>Written by:
Michael Park</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.7.2.4
3.1.1.7.2.5
3.1.1.7.2.6</td><td colspan="3">Purpose: Verify that all FloorSettingsController endpoints properly return the correct Thymeleaf template page, the Model contains the necessary objects, and the RedirectAttributes contains the necessary alerts.</td></tr>
<tr><td colspan="4">Setup Conditions:
• Open IntelliJ
• Open FloorSettingsControllerTests.java
• Run ‘FloorSettingsControllerTests’

JUnit Setup Conditions
• Create two Floor objects
• Mock the FloorRepository class
• Mock the FloorRepository:findByKey method to return a Floor
• Mock the FloorRepository:findAll method to return a collections of two Floors
• Mock the FloorRepository:save method
• Mock the FloorRepository:delete method</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1. TestIndex</td><td></td><td>Tests the index get method with no alert</td><td>Returns the "settings/floor/index" Thymeleaf template and a Model that contains a collection of two Garages</td></tr>
<tr><td>2. TestIndexSuccessMessage</td><td></td><td>Tests the index get method with a success alert</td><td>Returns the "settings/floor/index" Thymeleaf template and a Model that contains a collection of two Garages with a success alert</td></tr>
<tr><td>3. TestIndexInfoMessage</td><td></td><td>Tests the index get method with a information alert</td><td>Returns the "settings/floor/index" Thymeleaf template and a Model that contains a collection of two Garages with a information alert</td></tr>
<tr><td>4. TestIndexWarningMessage</td><td></td><td>Tests the index get method with a warning alert</td><td>Returns the "settings/floor/index" Thymeleaf template and a Model that contains a collection of two Garages with a warning alert</td></tr>
<tr><td>5. TestIndexDangerMessage</td><td></td><td>Tests the index get method with a danger alert</td><td>Returns the "settings/floor/index" Thymeleaf template and a Model that contains a</td></tr>
</table>

			collection of two Garages with a danger alert
6. TestGarageNoMessage		Tests the garage get method with no alert	Returns the "settings/floor/garage" Thymeleaf template and a Model that contains a garage object with no alert
7. TestGarageSuccessMessage		Tests the garage get method with a success alert	Returns the "settings/floor/garage" Thymeleaf template and a Model that contains a garage object with a success alert
8. TestGarageInfoMessage		Tests the garage get method with a information alert	Returns the "settings/floor/garage" Thymeleaf template and a Model that contains a garage object with a information alert
9. TestGarageWarningMessage		Tests the garage get method with a warning alert	Returns the "settings/floor/garage" Thymeleaf template and a Model that contains a garage object with a warning alert
10. TestGarageDangerMessage		Tests the garage get method with a danger alert	Returns the "settings/floor/garage" Thymeleaf template and a Model that contains a garage object with a danger alert
11. TestCreate_Get_NoMessage		Tests the create get method with no alert	Returns the "settings/floor/create" Thymeleaf template and a Model that contains a floor object and a garage object with no alert
12. TestCreate_Get_SuccessMessage		Tests the create get method with a success alert	Returns the "settings/floor/create" Thymeleaf template and a Model that contains a floor object and a garage object with a success alert
13. TestCreate_Get_InfoMessage		Tests the create get method with a information alert	Returns the "settings/floor/create" Thymeleaf template and a Model that contains a floor object and a garage object with an information alert
14. TestCreate_Get_WarningMessage		Tests the create get method with a warning alert	Returns the "settings/floor/create" Thymeleaf template and a Model that contains a floor object and a garage object with a warning alert

15. TestCreate_Get_DangerMessage		Tests the create get method with a danger alert	Returns the "settings/floor/create" Thymeleaf template and a Model that contains a floor object and a garage object with a danger alert
16. TestCreate_Post_NullGarageKey		Tests the create post method when a null garage key is passed.	Returns a redirect to "settings/floor/index" with a danger alert that a garage key cannot be null
17. TestCreate_Post_EmptyGarageKey		Tests the create post method when an empty garage key is passed.	Returns a redirect to "settings/floor/index" with a danger alert that a garage key cannot be an empty string
18. TestCreate_Post_Duplicate		Tests the create post method for a duplicate floor object	Returns a redirect to "settings/floor/create" with a danger alert that a duplicate floor exists
19. TestCreate_Post_Success		Tests the create post method for a successful floor object creation	Returns a redirect to "settings/floor/garage" with a success alert that a new floor has been successfully created
20. TestEdit_Get_NoMessage		Tests the edit get method with no alert	Returns "settings/floor/edit"Thymeleaf template and a Model that contains a floor object with no alert
21. TestEdit_Get_SuccessMessage		Tests the edit get method with a success alert	Returns "settings/floor/edit"Thymeleaf template and a Model that contains a floor object with a success alert
22. TestEdit_Get_InfoMessage		Tests the edit get method with a information alert	Returns "settings/floor/edit"Thymeleaf template and a Model that contains a floor object with a information alert
23. TestEdit_Get_WarningMessage		Tests the edit get method with a warning alert	Returns "settings/floor/edit"Thymeleaf template and a Model that contains a floor object with a warning alert
24. TestEdit_Get_DangerMessage		Tests the edit get method with a danger alert	Returns "settings/floor/edit"Thymeleaf template and a Model that contains a floor object with a danger alert

25. TestEdit_Post_SameFloorNumber		Tests the edit post method for the same floor number	Returns a redirect to "settings/floor/garage" with a success alert that the floor has been updated successfully
26. TestEdit_Post_Duplicate		Tests the edit post method for the same floor number and different floor key	Returns a redirect to "settings/floor/edit" with a danger alert that another floor already has the same floor number
27. TestEdit_Post_DifferentFloorNumber_Success		Tests the edit post method for different floor number	Returns a redirect to "settings/floor/garage" with a success alert that the floor has been updated successfully
28. TestDelete_NullFloorKey		Tests the delete post method for a null floor key	Returns a redirect to "settings/floor/index" with a danger alert that a floor key cannot be null
29.TestDelete_EmptyFloorKey		Tests the delete post method for an empty floor key	Returns a redirect to "settings/floor/index" with a danger alert that a floor key cannot be an empty string
30. TestDelete_NonExistentFloorKey		Tests the delete post method for a non-existing floor key	Returns a redirect to "settings/floor/index" with a danger alert that a floor with the specified floor key does not exist
31. TestDelete_Success		Tests the delete post method for a successful floor deletion	Returns a redirect to "settings/floor/garage" with a success alert that the floor has been successfully deleted

[This space intentionally left blank]

<table>
<tr><td>Test Category:
JUnit Tests</td><td colspan="3">Description: Provide the capability for an administrative user to add, edit, and delete parking spaces</td></tr>
<tr><td>Test Cases:
3.9, 3.10, 3.11, 3.12</td><td>Case Name:
ParkingSpaceSettingsController Tests</td><td>Version:
1.0</td><td>Written by:
Cody Coughenour</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.7.2.7
3.1.1.7.2.8
3.1.1.7.2.9</td><td colspan="3">Purpose: Verify that all ParkingSpaceSettingsController endpoints properly return the correct Thymeleaf template page, the Model contains the necessary objects, and the RedirectAttributes contains the necessary alerts.</td></tr>
<tr><td colspan="4">Setup Conditions:
Open IntelliJ
Open ParkingSpaceSettingsControllerTests.java
Run ‘ParkingSpaceSettingsControllerTests’
JUnit Setup Conditions
Create four Parking Space objects
Mock the ParkingSpaceRepository class
Mock the ParkingSpaceRepository:findByKey method to return a ParkingSpace
Mock the ParkingSpaceRepository:findAll method to return a collections of four ParkingSpace
Mock the ParkingSpaceRepository:save method
Mock the ParkingSpaceRepository:delete method
Repeat above for two Garages, two Floors, four PermitTypes, and four SpaceTypes
Create a GarageService object
Mock the GarageService: refresh method to do nothing
Mock the GarageService: save method to do nothing</td></tr>
<tr><th>Test Case Activity</th><th>Pass/Fail</th><th>Comments</th><th>Expected Result</th></tr>
<tr><td>1. TestIndex</td><td></td><td>Tests the index get method</td><td>Returns the "settings/parking_space/index" Thymeleaf template and a Model that contains a collection of two Garages</td></tr>
<tr><td>2. TestFloor</td><td></td><td>Tests the floor get method

The Floor is floorOne

The Garage is garageOne</td><td>Returns the “settings/parking_space/floor” Thymeleaf template and a Model that contains a collection of two Parking Spaces for that floor and four Permit Types and Space Types</td></tr>
<tr><td>3. TestCreate_Get</td><td></td><td>Tests the create get method

The Floor is floorOne</td><td>Returns the “settings/parkingSpace/create” Thymeleaf template and a Model that contains a new ParkingSpace object, a Floor, Permit Types, and Space Types</td></tr>
</table>

4. TestCreate_Post_Success		Tests the create post method for a non-duplicate Parking Space object	Returns a redirect to the "settings/parking_space/floor" Thymeleaf template, an empty Model, and the success message within the RedirectAttributes
5. TestCreate_Post_Duplicate		Tests the create post method for a duplicate Parking Space object	Returns the "settings/parking_space/create" Thymeleaf template, and a Model that contains the submitted Garage object and the error message.
6. TestSetSpaceNumber_success		Tests the setSpaceNumber post method for a non-duplicate Parking Space number (5)	Returns the "The space number was set to 5" String
7. TestSetSpaceNumber_duplicate		Tests the setSpaceNumber post method for a duplicate Parking Space number (2)	Returns the "The space number, 2, already exists." String
8. TestSetSpaceType		Tests the SetSpaceType post method (spaceTypeTwo)	Returns the "The space type of the space number 1 was set to spaceTypeTwo" String
9. TestSetPermitType		Tests the SetPermitType post method (permitTypeTwo)	Returns the "The permit type of the space number 1 was set to permitTypetwo" String
10. TestSetAvailability_available		Tests the SetAvailability post method (true)	Returns the "The space number 1 was set to available" String
11.TestSetAvailability_unavailable		Tests the SetAvailability post method (false)	Returns the "The space number 1 was set to unavailable" String
8. TestDelete_Post_Success		Tests the delete	Returns a redirect to the "settings/parking_spacee/floor/F

		post method for a successful attempt to delete a Parking Space object	LOOR_ONE_KEY” Thymeleaf template, an empty Model, and the success message within the RedirectAttributes

[This space intentionally left blank]

<table>
<tr><td>Test Category:
JUnit Tests</td><td colspan="3">Description: Provide the capability for an administrative user to add, edit, and delete university buildings</td></tr>
<tr><td>Test Cases:
3.13, 3.14, 3.15, 3.16</td><td>Case Name:
BuildingSettingsController Tests</td><td>Version:
1.0</td><td>Written by:
Michael Park</td></tr>
<tr><td>Requirements Fulfilled:
3.4.1.1.1
3.4.1.1.2
3.4.1.1.3</td><td colspan="3">Purpose: Verify that all BuildingSettingsController endpoints properly return the correct Thymeleaf template page, the Model contains the necessary objects, and the RedirectAttributes contains the necessary alerts.</td></tr>
<tr><td colspan="4">Setup Conditions:
• Open IntelliJ
• Open BuildingSettingsControllerTests.java
• Run ‘BuildingSettingsControllerTests’

JUnit Setup Conditions
• Create two Building objects
• Mock the BuildingRepository class
• Mock the BuildingRepository:findByKey method to return a Building
• Mock the BuildingRepository:findAll method to return a collections of two Buildings
• Mock the BuildingRepository:save method
• Mock the BuildingRepository:delete method</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1. TestIndex_NoMessage</td><td></td><td>Tests the index get method with no alert</td><td>Returns the "settings/building/index" Thymeleaf template and a Model that contains a collection of two building objects</td></tr>
<tr><td>2. TestIndex_SuccessMessage</td><td></td><td>Tests the index get method with a success alert</td><td>Returns the "settings/building/index" Thymeleaf template and a model that contains a collection of two building objects with a success alert</td></tr>
<tr><td>3. TestIndex_InfoMessage</td><td></td><td>Tests the index get method with an information alert</td><td>Returns the "settings/building/index" Thymeleaf template and a model that contains a collection of two building objects with an information alert</td></tr>
<tr><td>4. TestIndex_WarningMessage</td><td></td><td>Tests the index get method with a warning alert</td><td>Returns the "settings/building/index" Thymeleaf template and a model that contains a collection of two building objects with a warning alert</td></tr>
<tr><td>5. TestIndex_DangerMessage</td><td></td><td>Tests the index get method with a danger alert</td><td>Returns the "settings/building/index" Thymeleaf template and a model that contains a collection of two building</td></tr>
</table>

			objects with a danger alert
6. TestCreate_Get		Tests the create get method	Returns the "settings/building/create" Thymeleaf template and a model that contains a building object
7. TestCreate_Post_Successful		Tests the create post method for successful Building creation	Returns a redirection to the "settings/building/index" with a success alert
8. TestCreate_Post_Duplicate		Tests the create post method for unsuccessful Building creation	Returns a redirection to the "settings/building/create", a model that contains a building object and a danger alert
9. TestEdit_Get		Tests the edit get method	Returns the "settings/building/edit" Thymeleaf template and a model that contains a building object
10. TestEdit_Post_Successful		Tests the edit post method when editing is successful	Returns a redirection to "settings/building/index" and a redirect attribute that contains a success alert
11.TestEdit_Post_Duplicate		Tests the edit post method when editing is unsuccessful	Returns the "settings/building/edit" Thymeleaf template and a model that contains a building object and a danger alert
12. TestDelete_Successful		Tests the delete post method for a successful attempt to delete a building object	Returns a redirection to the "settings/building/index" Thymeleaf template and a redirect attribute that contains a success alert
13. TestDelete_Unsuccessful		Tests the delete post method for an unsuccessful attempt to delete a building object	Returns a redirection to the "settings/building/index" Thymeleaf page with a redirect attribute that contains a danger alert

[This space intentionally left blank]

<table>
<tr><td>Test Category:
JUnit Tests</td><td colspan="3">Description: Provide the capability for an administrative user to add, edit, and delete users</td></tr>
<tr><td>Test Cases:
3.17, 3.18, 3.19, 3.20</td><td>Case Name:
AccountController Tests</td><td>Version:
1.0</td><td>Written by:
Matthew Stevenson</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.7.4.1
3.1.1.7.4.2
3.1.1.7.4.3</td><td colspan="3">Purpose: Verify that all AccountController endpoints properly return the correct Thymeleaf template page, the Model contains the necessary objects, and the RedirectAttributes contains the necessary alerts.</td></tr>
<tr><td colspan="4">Setup Conditions:
Open IntelliJ
Open AccountsControllerTests.java
Run ‘AccountsControllerTests’
JUnit Setup Conditions
Create two User objects
Mock the UserRepository class
Mock the UserRepository:findByKey method to return User based on specified key
Mock the UserRepository:findAll method to return a collection of Users
Mock the UserRepository:findByEmail method to return a User with specified email
Mock the UserRepositoyu:userExists method to return a boolean value
Mock the UserRepository:save method
Mock the UserRepository:delete method</td></tr>
<tr><th>Test Case Activity</th><th>Pass/Fail</th><th>Comments</th><th>Expected Result</th></tr>
<tr><td>1. TestIndex</td><td></td><td>Tests the index get method with no alert</td><td>Returns the "settings/accounts/index" Thymeleaf template and a Model that contains a collection of two Garages</td></tr>
<tr><td>2. TestIndexSuccessMessage</td><td></td><td>Tests the index get method with a success alert</td><td>Returns the "settings/accounts/index" Thymeleaf template and a Model that contains a collection of two Garages with a success alert</td></tr>
<tr><td>3. TestIndexInfoMessage</td><td></td><td>Tests the index get method with a information alert</td><td>Returns the "settings/accounts/index" Thymeleaf template and a Model that contains a collection user with a information alert</td></tr>
<tr><td>4. TestIndexWarningMessage</td><td></td><td>Tests the index get method with a warning alert</td><td>Returns the "settings/accounts/index" Thymeleaf template and a Model that contains a collection of two users with a warning alert</td></tr>
</table>

5. TestIndexDangerMessage		Tests the index get method with a danger alert	Returns the "settings/accounts/index" Thymeleaf template and a Model that contains a collection of two users with a danger alert
6. TestCreate_Get_NoMessage		Tests the create get method with no alert	Returns the "settings/accounts/create" Thymeleaf template and a Model that contains an user object with no alert
7. TestCreate_Get_SuccessMessage		Tests the create get method with a success alert	Returns the "settings/accounts/create" Thymeleaf template and a Model that contains an user object with a success alert
8. TestCreate_Get_InfoMessage		Tests the create get method with a information alert	Returns the "settings/accounts/create" Thymeleaf template and a Model that contains an user object with an information alert
9. TestCreate_Get_WarningMessage		Tests the create get method with a warning alert	Returns the "settings/accounts/create" Thymeleaf template and a Model that contains an user object with a warning alert
10. TestCreate_Get_DangerMessage		Tests the create get method with a danger alert	Returns the "settings/accounts/create" Thymeleaf template and a Model that contains an user object with a danger alert
11. TestCreate_Post_NullUserKey		Tests the create post method when a null user key is passed.	Returns a redirect to "settings/accounts/index" with a danger alert that a user key cannot be null
12. TestCreate_Post_EmptyUserKey		Tests the create post method when an empty user key is passed.	Returns a redirect to "settings/accounts/index" with a danger alert that a user key cannot be an empty string
13. TestCreate_Post_Duplicate		Tests the create post method for a duplicate	Returns a redirect to "settings/accounts/create" with a danger alert that a duplicate user exists

		user object	
14. TestCreate_Post_Success		Tests the create post method for a successful user object creation	Returns a redirect to "settings/accounts/index " with a success alert that a new user has been successfully created
15. TestEdit_Get_NoMessage		Tests the edit get method with no alert	Returns a redirect to "settings/accounts/edit" Thymeleaf template and a Model that contains a user object with no alert
16. TestEdit_Get_SuccessMessage		Tests the edit get method with a success alert	Returns "settings/accounts/edit "Thymeleaf template and a Model that contains a user object with a success alert
17. TestEdit_Get_InfoMessage		Tests the edit get method with an informatio n alert	Returns "settings/accounts/edit "Thymeleaf template and a Model that contains a user object with an information alert
18. TestEdit_Get_WarningMessage		Tests the edit get method with a warning alert	Returns "settings/accounts/edit "Thymeleaf template and a Model that contains a user object with a warning alert
19. TestEdit_Get_DangerMessage		Tests the edit get method with a danger alert	Returns "settings/floor/edit "Thymeleaf template and a Model that contains a floor object with a danger alert
20. TestEdit_Post_DuplicateEmail		Tests the edit post method for a duplicate email	Returns a redirect to "settings/floor/garage" with a success alert that the floor has been updated successfully
21. TestEdit_Post_DuplicateUserName		Tests the edit post method for a duplicate username	Returns a redirect to "settings/accounts/edit" with a danger alert that another user with that username exits
22. TestEdit_Post_DifferentEmail_Success		Tests the edit post method for different email	Returns a redirect to "settings/accounts/index " with a success alert that a user has been updated successfully

23. TestEdit_Post_DifferentUsername_Success		Tests the edit post method for different username	Returns a redirect to "settings/accounts/index" with a success alert that a user has been updated successfully
24. TestDelete_NullUserKey		Tests the delete post method for a null user key	Returns a redirect to "settings/accounts/index" with a danger alert that am user key cannot be null
25. TestDelete_EmptyUserKey		Tests the delete post method for an empty user key	Returns a redirect to "settings/accounts/index" with a danger alert that a user key cannot be an empty string
26. TestDelete_NonExistentUserKey		Tests the delete post method for a non-existing user key	Returns a redirect to "settings/accounts/index" with a danger alert that a user with the specified user key does not exist
27. TestDelete_Success		Tests the delete post method for a successful user deletion	Returns a redirect to "settings/accounts/index" with a success alert that a user has been successfully deleted

[This space intentionally left blank]

<table>
<tr><td>Test Category:
JUnit Tests</td><td colspan="3">Description: Provide the capability for a user to create an account to save user as a self-service</td></tr>
<tr><td>Test Cases:
3.21, 3.22</td><td>Case Name:
RegisterController Tests</td><td>Version:
1.0</td><td>Written by:
Matthew Stevenson</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.9.1
3.1.1.9.2</td><td colspan="3">Purpose: Verify that all RegisterController endpoints properly return the correct Thymeleaf template page, the Model contains the necessary objects, and the RedirectAttributes contains the necessary alerts.</td></tr>
<tr><td colspan="4">Setup Conditions:
Open IntelliJ
Open RegisterControllerTests.java
Run ‘RegisterControllerTests’
JUnit Setup Conditions
Create two User objects
Create simpleMailMessage message object to mock email messages
Mock the UserRepository class
Mock the UserRepository:findByKey method to return User based on specified key
Mock the UserRepository:findAll method to return a collection of Users
Mock the UserRepository:findByConfirmationLink method to return a User based on specified confirmation link
Mock the UserRepository:findByEmail method to return a User with specified email
Mock the UserRepositoyu:userExists method to return a boolean value
Mock the UserRepository:save method</td></tr>
<tr><th>Test Case Activity</th><th>Pass/Fail</th><th>Comments</th><th>Expected Result</th></tr>
<tr><td>1. TestRegister</td><td></td><td>Tests the register get method with no alert</td><td>Returns the "user/register" Thymeleaf template and a Model that contains an user object for potential new users</td></tr>
<tr><td>2. TestRegisterSuccessMessage_Post</td><td></td><td>Tests the register post method with a success alert</td><td>Returns the "user/register" Thymeleaf template and a Model that a confirmation link has been sent to the user’s email</td></tr>
<tr><td>3. TestRegisterDangerMesasgeDuplicateEmail_Post</td><td></td><td>Tests the register post method with a danger alert</td><td>Returns the "user/register" Thymeleaf template and a Model that contains a danger alert based on submission of duplicate email</td></tr>
</table>

3. TestRegisterDangerMesasgeDuplicateUsername_Post		Tests the register post method with a danger alert	Returns the "user/register" Thymeleaf template and a Model that contains a danger alert based on submission of duplicate username
7. TestConfirm_Get_SuccessMessage		Tests the confirm get method with a success	Returns the "user/confirm" Thymeleaf template and a Model that contains a success message notifying the user the confirmation link is valid and redirects to "home/login". User account is set to enabled and can login.
8. TestConfirm_Get_DangerMessage		Tests the confirm get method with a danger message	Returns the "user/confirm" Thymeleaf template and a Model that contains danger message notifying the user that their confirmation link is invalid.

[This space intentionally left blank]

<table>
<tr><td>Test Category:
JUnit Tests</td><td colspan="3">Description: Provide the capability for a user get directions to a garage based on specified starting location using the Google Maps API</td></tr>
<tr><td>Test Cases:
3.23</td><td>Case Name:
MapsController Tests</td><td>Version:
1.0</td><td>Written by:
Matthew Stevenson</td></tr>
<tr><td>Requirements Fulfilled:
3.1.1.5.1</td><td colspan="3">Purpose: Verify that all MapController endpoints properly return the correct Thymeleaf template page, the Model contains the necessary objects.</td></tr>
<tr><td colspan="4">Setup Conditions:
• Open IntelliJ
• Open MapsControllerTests.java
• Run ‘MapsControllerTests’

JUnit Setup Conditions
• Create two Location objects
• Create two Garage objects
• Mock the GarageRepository class
• Mock the GarageRepository:findByKey method to return a Garage based on specified key</td></tr>
<tr><td>Test Case Activity</td><td>Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1. TestNavigate</td><td></td><td>Tests the navigate get method</td><td>Returns the "maps/navigate" Thymeleaf template and a Model that contains a location object and a garage object</td></tr>
<tr><td>2. TestStartNavigation_Post</td><td></td><td>Tests the navigation post method</td><td>Redirects to “Google maps” Thymeleaf template and a Model that push starting location and garage location to navigate functionality</td></tr>
</table>

[This space intentionally left blank]

<table>
<tr><td colspan="2">Test Category:
JUnit Tests</td><td colspan="3">Description: The user data model should encapsulate all necessary attributes of a typical ParkODU end user and a ParkODU administrator. The data model should also provide methods to access and manipulate the attributes.</td></tr>
<tr><td colspan="2">Test Cases:
3.24</td><td>Case Name:
UserTests.
java</td><td>Version:
1.0</td><td>Written by:
Michael Park</td></tr>
<tr><td colspan="2">Requirements Fulfilled:
3.1.1.2.1
3.1.1.7.4.1
3.1.1.7.4.2
3.1.1.7.4.3
3.1.1.8.1
3.1.1.8.2</td><td colspan="3">Purpose: Verify the completeness of the User model interface, that all User.java get methods properly return requested attributes, and that all set methods properly update the attributes</td></tr>
<tr><td colspan="5">Setup Conditions:
• Open IntelliJ
• Open UserTests.java
• Run ‘UserTests’
JUnit Setup Conditions
• Create a user object
• Create mock attributes</td></tr>
<tr><td>Test Case Activity</td><td colspan="2">Pass/Fail</td><td>Comments</td><td>Expected Result</td></tr>
<tr><td>1. TestUser</td><td colspan="2"></td><td>Tests the default constructor of the User model</td><td>Creates an instance of the User class with all attributes set to null or empty if the attributes are a collection</td></tr>
<tr><td>2. TestUserStringStringStringStringStringStringStringBoolean</td><td colspan="2"></td><td>Tests the constructor that accepts user’s email, username, password, first name, last name, role type, role type key, enabled flag, and confirmation token</td><td>Creates an instance of the User class with all of the following attributes initialized: Email, username, password, first name, last name, role type, role type key, enabled flag, and confirmation token</td></tr>
<tr><td>3. TestGenerateUserKey</td><td colspan="2"></td><td>The generateUserKey method must generate a unique ID for this user</td><td>This instance of the User class has a non-null and non-empty unique ID string</td></tr>
<tr><td>4. TestGenerateConfirmationToken</td><td colspan="2"></td><td>The generateConfirmationToken method must generate a</td><td>This instance of the User class has a non-null and non-empty</td></tr>
</table>

		unique confirmation token for this user	unique confirmation token.
5. TestSetConfirmationToken		The setConfirmationToken method accepts a confirmation token as a parameter and updates this user's confirmation token	This instance of the User class has a new confirmation token.
6. TestGetConfirmationToken		The getConfirmationToken method returns the confirmation token.	The method returns the confirmation token of this instance of the User class. The data type of the returned variable is String.
7. TestSetUserKey		The setUserKey method accepts a user ID as a parameter and updates this user's unique ID.	This instance of the User class has a new unique ID.
8. TestGetUserKey		The getUserKey method returns the unique user ID.	The method returns the unique ID of this instance of the User class. The data type of the returned variable is String.
9. TestGetPassword		The getPassword method returns the password of this user instance.	The method returns the password of this instance of the User class. The data type of the returned variable is String.
10. TestSetPassword		The setPassword method accepts a password as a parameter and updates this user's password.	This instance of the User class has a new password.
11. TestGetFirstName		The getFirstName method returns the first name of this user instance.	The method returns the first name of this instance of the User class. The data type of the returned variable is String.

12. TestSetFirstName		The setFirstName method accepts a first name as a parameter and updates this user's first name.	This instance of the User class has a new first name.
13. TestGetLastName		The getLastName method returns the last name of this user instance.	The method returns the last name of this instance of the User class. The data type of the returned variable is String.
14. TestSetLastName		The setLastName method accepts a last name as a parameter and updates this user's last name.	This instance of the User class has a new last name.
15. TestGetEmail		The getEmail method returns the e-mail address of this user instance.	The method returns the e-mail address of this instance of the User class. The data type of the returned variable is String.
16. TestSetEmail		The setEmail method accepts an e-mail address as a parameter and updates this user's e-mail address.	This instance of the User class has a new e-mail address.
17. TestGetUsername		The getUsername method returns the username of this user instance.	The method returns the username of this instance of the User class. The data type of the returned variable is String.
18. TestSetUsername		The setUsername method accepts a username as a parameter and updates this user's username.	This instance of the User class has a new username.
19. TestGetEnabled		The getEnabled method returns the enabled flag of this user instance.	The method returns the enabled flag of this instance of the User class. The data type of the returned variable is Boolean.

20. TestSetEnabled		The setEnabled method accepts a boolean flag as a parameter and updates this user's status.	This instance of the User class has a new status flag.
21. TestGetRoleType		The getRoleType method returns the role type name (User or Admin) of this user instance.	The method returns the role type name of this instance of the User class. The data type of the returned variable is String.
22. TestSetRoleType		The setRoleType method accepts a role type name as a parameter and updates this user's role.	This instance of the User class has a new role type.
23. TestGetRoleTypeKey		The getRoleTypeKey method returns the unique ID of the role type that this user instance has.	The method returns the unique ID of the role type of this instance of the User class. The data type of the returned variable is String.
24. TestSetRoleTypeKey		The setRoleTypeKey method accepts a role type unique ID as a parameter and updates this user's role type ID.	This instance of the User class has a new role type ID.
25. TestGetPermissions		The getGetPermissions method returns a collection of permissions that this user instance should have.	The method returns a set of permissions for this instance of the User class. The data type of the returned variable is HashSet of String.
26. TestSetPermissions		The setRoleTypeKey method accepts a set of permissions as a parameter and updates this user's permissions.	This instance of the User class has a new set of permissions.
27. TestGetPreferredPermitTypes		The getPreferredPermitTypes method returns a collection of permit types that	The method returns a set of preferred permit types of this instance of the User class. The

		this user instance has set preference for.	data type of the returned variable is HashSet of String.
28.TestSetPreferredPermitTypes		The setPreferredPermitTypes method accepts a set of permit types as a parameter and updates this user’s preferred permit types.	This instance of the User class has a new set of preferred permit types.
29. TestGetPreferredSpaceTypes		The getPreferredSpaceTypes method accepts a set of space types as a parameter and updates this user’s preferred space types.	The method returns a set of preferred space types of this instance of the User class. The data type of the returned variable is HashSet of String.
30. TestSetPreferredSpaceTypes		The setPreferredSpaceTypes method accepts a set of space types as a parameter and updates this user’s preferred space types.	This instance of the User class has a new set of preferred space types.
31. TestGetAuthorities		The getAuthorities method returns a collection of SimpleGrantedAuthority object for Spring Security based on the permissions that this user instance has.	The methods returns a set of SimpleGrantedAuthority objects which are initialized with this user instance’s existing permissions. The data type of the returned variable is a HashSet of SimpleGrantedAuthority class
32. TestToString		The toString method returns the String representation of this instance of the User class.	The method returns a string literal that represents this user instance and closely resembles the json format that includes all attribute names and values.

4.4 UX Tests

User Experience (UX) Tests are conducted to ensure the usability and accessibility of ParkODU. To provide usability and accessibility to all users including people with disabilities, the ParkODU development team strives to develop web pages in compliance with Section 508 of the Rehabilitation Act of 1973.

[This space intentionally left blank]

Test Category: User Experience	Description: To verify that the application is accessible as per some section 508 Accessibility program guidelines.		
Test Case: 4.1	Case Name: Accessibility management	Version: 1.0	Written By: Sangeet Mokha
Requirements Fulfilled: 3.4.4.1 – 3.4.4.5	Purpose: To make sure the application conforms to the font size readability, color contrast, images, and ease of navigation according to section 508.		

Setup Conditions:
- Open ParkODU web application
- Navigate all the link/pages on it
- Go back on the main page

	Test Case Activity	Pass/Fail	Comments	Expected Result
1	Check accessibility for font size			Characters on all the pages shall be in a sans serif font.
2	Check accessibility for color contrast			There must be contrasting colors/shades at a minimum ratio of 4.5:1 on all pages.
3	Check accessibility for images			All images must have associated text describing the purpose and/or function of the image. Decorative images do not require a description.
4	Check for ease of navigation with a keyboard.			Use the standard keyboard commands (Tab, Space bar, arrow keys, Enter, etc.) to navigate through each interactive interface component (including form drop-down lists and form fields), reveal hidden content, and activate all interface components.
5	Check for ease of navigation with a mouse			To Find all visible and hidden interactive interface components (links, form fields, drop down menus, show/hide content, tree views, pop ups/light boxes, frames, iframes, etc.) use a mouse (hover and/or click).

[This space intentionally left blank]

5. Traceability of Requirements

This matrix maps each test case to any requirements it fulfills.

Requirement ID	Test Case ID																																																
	1.1	1.2	1.3	1.4	1.5	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	2.10.	2.11	2.12	2.13	2.14	2.15	2.16	2.17	2.18	2.19	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	3.1	3.11	3.12	3.13	3.14	3.15	3.16	3.17	3.18	3.19	3.2.0	3.21	3.22	3.23	3.24	4.1
3.1.1.2.1				x													x																															x	
3.1.1.2.2				x													x																															x	
3.1.1.2.3				x													x																															x	
3.1.1.3.1				x																																													
3.1.1.3.2																						x																											
3.1.1.4.1																	x																																
3.1.1.4.2															x																																		
3.1.1.4.3															x																																		
3.1.1.4.4															x																																		
3.1.1.4.5															x																																		
3.1.1.4.6															x																																		
3.1.1.5.1																x																															x		
3.1.1.6.1																		x																															
3.1.1.7.1			x																																														
3.1.1.7.1.1			x																x																														
3.1.1.7.1.2			x																	x																													
3.1.1.7.1.3			x																		x																												
3.1.1.7.2.1			x									x													x	x	x	x																					
3.1.1.7.2.2			x										x												x	x	x	x																					
3.1.1.7.2.3			x											x											x	x	x	x																					
3.1.1.7.2.4			x						x																				x	x	x	x																	
3.1.1.7.2.5			x							x																			x	x	x	x																	
3.1.1.7.2.6			x								x																		x	x	x	x																	
3.1.1.7.2.7			x			x																											x	x	x	x													
3.1.1.7.2.8			x				x																										x	x	x	x													
3.1.1.7.2.9			x					x																									x	x	x	x													
3.1.1.7.3.1			x																				x																										
3.1.1.7.3.2			x																					x																									
3.1.1.7.4.1			x																																						x	x	x	x				x	
3.1.1.7.4.2			x																																						x	x	x	x				x	
3.1.1.7.4.3			x																																						x	x	x	x				x	
3.1.1.8.1	x																																												x	x			
3.1.1.8.2		x																																											x	x			
3.1.1.9.1																																																	
3.1.1.9.2																																																	
3.1.2.1.1																																																	
3.1.2.1.2																																																	
3.1.2.1.3																																																	
3.2.1.1																																																	
3.2.2.1																																																	
3.2.2.2																																																	
3.2.3.1																																																	
3.4.1.1.1																																					x	x	x	x									
3.4.1.1.2																																					x	x	x	x									
3.4.1.1.3																																					x	x	x	x									
3.4.4.1																																																	x
3.4.4.2																																																	x
3.4.4.3																																																	x
3.4.4.4																																																	x
3.4.4.5																																																	x

[This space intentionally left blank]