TDTP

Documentation

Academic year: 2020-2021, Spring Semester

Team Name: Morning Coffee

Team members:

Contents

1. Application Details. Investigated Features

EVOZON LightHouse is a desk booking management app used by the employees of a company for booking desks during the work week which came as a solution in the context of the pandemic.

Our team, Morning Coffee, tested a wide range of features through various test design techniques, from the basics, such as the sign in feature with function testing, to the workload the app can handle through performance and load testing.

2. AC. IOs

Application Context: You are part of the testing team that joined the project two weeks before product release. You are experienced testers.

Part I: The project manager wants you to check for critical bugs that may block the planned release.

Part II: A month after release users report application severe issues that threaten product credibility. The project manager suspects already reported bugs might be the cause and asks your team to investigate them and asks you to design a complex test that may reproduce the failure.

Part III: The project manager wants you to check and test special cases for the delivered features.

3. Testing Mission

Testing mission for Part I.

Test the product, find the bugs if they exist and classify them, in order to see if some are critical and may result in the postponing of said release, or are minor and can be solved after.

Testing mission for Part II.

Investigate the bugs reported by the user, and compare them with already known/reported bugs to see if the former are caused by the latter.

Testing mission for Part III.

Check if the delivered features have no critical bugs, in order to know if some other features can be implemented starting from these**.**

4. Testing Strategy

Testing strategy for Part I.

We decided to make function testing and test security related functionalities, both on UI and API level. Application and data security is crucial for users and service providers and any issue found may block the release.

We decided to load test the web application to make sure it will run seamlessly, when used by multiple users. The application may, depending on the limits of its resources, perform completely differently for one user (functional testing) compared to many (load testing).

Testing strategy for Part II.

We decided to use risk-based testing techniques for this part because tests are derived from ideas about how the program could fail. Since the IO states that there are bugs, we think that applying risk-based testing will help find the bug quickly and as efficiently as possible because time is of the essence when issues are severe and the product was released.

Testing strategy for Part III.

Since we are experienced testers (according to the application context), we should use our previous experience in choosing special cases for the implemented features. If time is enough, we could focus on achieving a relatively high coverage. If time is short, we should choose best representatives and boundary values in order to make sure they perform as expected in very specific scenarios.

5. Selected Test Design Techniques

Part	Test Strategy	Test Design Technique	Test Attributes	Dimension covered	Students and Features
Part I	Coverage-based	Function testing	Coverage, Credible, Valid	Coverage	[redacted] *(Get user roles, before and after successful authentication)*
	Coverage-based	Function Testing	coverage, easy to evaluate, credible, power, information value, validity	Coverage	[redacted] *(User Login)*
	Coverage-based	Boundary Testing	Valid, Easy to evaluate, Representative	Coverage	[redacted] *(Add new desk)*
	Coverage-based	Best representative testing	Representative, Power, Non-redundant	Coverage	[redacted] *(Edit building)*
	Structure-based	Condition testing	Coverage, Valid		[redacted] *(User Login API)*
	Coverage-based	Boundary Testing	Coverage, Valid	Coverage	[redacted] *(Edit Building)*
Part II	Activity-based	*Scenario-based Testing (ST)*	*credible, motivating, power*	*Activity*	[redacted] *(Create Visitor account, Delete Visitor account, Create a visitor account with the same email)*
	Risk-based	Shoe Test	information value, easy to evaluate, power, validity	Risk	[redacted] (User Login)
	Specification-based	Constraint Check	Representative, Non-redundant, value and valid	Evaluation	[redacted] *(explore map calendar values)*
	Coverage-based	Boundary testing	Coverage, easy to evaluate, motivating	Risk	[redacted] *(add building)*
	Evaluation-based	Comparison with saved results	Power, Credible, Representative	Evaluation	[redacted] (Testing resolution for responsive web design features)
	Risk-Based	Stress Testing	Coverage, Information value, power	Risk	[redacted] *(add Booking)*
Part III	Coverage-based / Risk-based	*Boundary Testing(BT)*	*Coverage, Information value, Easy to evaluate*	Coverage	[redacted] *(Retrieve checkin data in a specific date range)*
	Activity-based	*Random Testing*	Representative, Non-redundant, value and valid	Activity	[redacted] *(Edit buildings safe distance)*

	Coverage-based / Risk-based	Boundary Testing (BT)	coverage, easy to evaluate, motivating, credible, validity, power, representative, non-redundant	Coverage, Risk	[redacted](User Login)
	Risk-based	Quick tests	effective, representative, powerful, affordable	Risk	[redacted] *(Add booking)*
	Risk-based	Configuration/Compatibility Testing	easy to evaluate, coverage, valid	Risk	[redacted] *(Browser Compatibility)*
	Structure-based	Condition testing	Coverage, Valid	Coverage	[redacted] *(Manage Equipment List)*

6. [optional] Test Design Techniques Improvement

Not an improvement, but if it was available and I was a stakeholder for this project, I would organize a bug bash session. From what I understand, it seems like a very fun activity and can be very useful in finding bugs quickly before releasing a product.

7. Test Design. implementation. Test execution Test n. Test Report

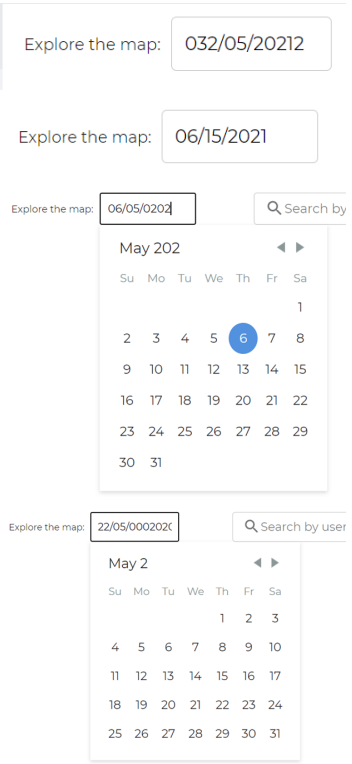
7.1.Test Design

<table>
<tr><td colspan="5">Information objective (Part I): The project manager wants you to check for critical bugs that may block the planned release</td></tr>
<tr><th>Student</th><th>Feature(s)</th><th>Test Design Technique</th><th>Details</th><th>Input, Expected output</th></tr>
<tr><td></td><td>Get user roles, before and after successful authentication</td><td>Function testing</td><td>User roles should be accessible only after successful authentication.

We generally want to make sure that any API is accessible only with a valid authentication token.</td><td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>Get user roles, before authentication</td><td>HTTP 401 Unauthorized</td></tr>
<tr><td>Get user roles, after authentication</td><td>HTTP 200, user roles JSON response.</td></tr>
</table>
</td></tr>
<tr><td></td><td>Load all users, load all checkins</td><td>Load Testing</td><td>Perform load test on most used functions.</td><td>Input : 4800 requests. Expected output : 4800 successful responses with requested data.</td></tr>
<tr><td></td><td>User Sign in</td><td>Function Testing</td><td>Check if user sign in functionality works</td><td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>valid credentials</td><td>user signed in (sees main page of the LightHouse app)</td></tr>
<tr><td>invalid credentials</td><td>user not signed in (sees sign in page of the LightHouse app)</td></tr>
</table>
* credentials = username + password
</td></tr>
<tr><td></td><td>User Login (API)</td><td>API Testing with Serenity</td><td></td><td>
<table>
<tr><th colspan="2">Input</th><th>Output</th><th>Expected</th></tr>
<tr><td>camelia.zalum@stud.ubbcluj.ro</td><td>Aa#123456</td><td>200</td><td>200</td></tr>
<tr><td>camelia.zalum@stud.ubbcluj.ro</td><td>Aa#1234.56</td><td>503</td><td>500</td></tr>
<tr><td>camelia.zalum@stud.ubbcluj.ro</td><td></td><td>503</td><td>400</td></tr>
<tr><td>cameliazalum@stud.ubbcluj.ro</td><td>Aa#123456</td><td>403</td><td>500</td></tr>
<tr><td>camelia.zalum</td><td>Aa#123456</td><td>403</td><td>400</td></tr>
<tr><td></td><td>Aa#123456</td><td>403</td><td>400</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td></td>
<td>Edit building</td>
<td>Best representative</td>
<td>Check if edit building works for a set of high-risk representative</td>
<td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>valid name and distance</td><td>building name and distance are updated</td></tr>
<tr><td>invalid name and distance</td><td>building is not updated, error message is shown or save button is disabled</td></tr>
</table>
</td>
</tr>
<tr>
<td></td>
<td>Add new desk</td>
<td>Boundary testing</td>
<td>Check if add new desk works properly</td>
<td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>valid type, name and equipment</td><td>new desk was added if the location was accessible</td></tr>
<tr><td>valid type, name and invalid equipment</td><td>error message pops up</td></tr>
<tr><td>valid type, equipment and invalid name</td><td>error message pops up</td></tr>
</table>
</td>
</tr>
<tr>
<td></td>
<td>Edit Building</td>
<td>Boundary Testing</td>
<td>Check if the functionality updates with valid information</td>
<td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>valid name, valid distance</td><td>the name and distance are updated</td></tr>
<tr><td>valid name
invalid distance</td><td>updates name</td></tr>
<tr><td>invalid name
valid distance</td><td>update distance</td></tr>
<tr><td>invalid name
invalid distance</td><td>doesn t update anything</td></tr>
</table>
</td>
</tr>
</table>

Information objective (Part II): A month after release users report application severe issues that threaten product credibility. The project manager suspects already reported bugs might be the cause and asks your team to investigate them and asks you to design a complex test that may reproduce the failure.

<table>
<tr><th>Student</th><th>Feature(s)</th><th>Test Design Technique</th><th>Details</th><th>Input, Expected output</th></tr>
<tr><td></td><td>date pick-up from the explore map feature</td><td>Constraint check (Evaluation-based Technique)</td><td>Invalid values are considered correct in the application</td><td> 
</td></tr>
<tr><td></td><td>Add Building</td><td>Boundary testing</td><td>Check if the add building feature is the one causing problems for the users</td><td>
<table>
<tr><th>Input</th><th>Rule1</th><th>Rule2</th></tr>
<tr><td>name</td><td>length >= 1</td><td></td></tr>
<tr><td>distance</td><td>is a number</td><td>not 0</td></tr>
<tr><td>result</td><td>True</td><td>True</td></tr>
</table>
</td></tr>
<tr><td></td><td>Create a visitor account.</td><td>Scenario Testing(ST)</td><td>Scenario in which a user creates a visitor account, deletes the account and wants to create the account again.</td><td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>Account Details(Email, first name, last name)</td><td>Visitor account created</td></tr>
<tr><td>Delete account</td><td>Account is deleted</td></tr>
<tr><td>Same Account Details</td><td>Visitor account created</td></tr>
<tr><td></td><td></td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>User sign in</td><td>Shoe Test</td><td>Check how the app behaves when given very long inputs for the username and password input fields from the sign in page.</td><td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>1000 chars for the username field + non-empty input for the password field</td><td>no buffer overflow</td></tr>
<tr><td>1000 chars for the password field + non-empty input for the username field</td><td>no buffer overflow</td></tr>
<tr><td>10000 chars for the username field + non-empty input for the password field</td><td>no buffer overflow</td></tr>
<tr><td>10000 chars for the password field + non-empty input for the username field</td><td>no buffer overflow</td></tr>
</table>
</td></tr>
<tr><td></td><td>Resolution for responsive web design</td><td>Comparison with saved results (Automated Testing with Selenium)</td><td>Check how the app looks in different customs resolutions</td><td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>Screenshot with smaller resolution</td><td>Screenshot with the look as in initial resolution</td></tr>
<tr><td>Screenshot with random resolution</td><td>Screenshot with the look as in initial resolution</td></tr>
<tr><td>Screenshot with bigger resolution</td><td>Screenshot with the look as in initial resolution</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>Add Booking</td><td>Stress Testing</td><td>Add a new Reservation in a vacant date, desk and floor</td><td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>Correct inputs</td><td>a new booking is created</td></tr>
<tr><td>one or more inputs is incorrect</td><td>cant create a new booking</td></tr>
<tr><td>all inputs are incorrect</td><td>cant create a new booking</td></tr>
</table>
</td></tr>
</table>

Information objective (Part III): *The project manager wants you to check and test special cases for the delivered features.*

<table>
<tr><th>Student</th><th>Feature(s)</th><th>Test Design Technique</th><th>Details</th><th>Input, Expected output</th></tr>
<tr><td></td><td>Retrieve checkin data for a specific date range</td><td>Equivalence & Boundary Testing</td><td>Variables from and to act for From Date and To Date representing the input for the endpoint :/api/v1/checkins . We test the date boundaries and the expected result corresponding to the given interval.</td><td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>LocalDate.now().minusYears(1) - LocalDate.now()</td><td>All existing checkins in the given interval.</td></tr>
<tr><td>LocalDate.now().minusDays(1) - LocalDate.now().with(TemporalAdjusters.lastDayOfMonth()).plusDays(1)</td><td>Invalid request (32 days for 31 days month OR 31 days for 30 days month)</td></tr>
<tr><td>dd-mm-yyyy, where {mm| mm <1 } = (-∞, 1) OR {mm| mm>12} = (12, +∞)</td><td>Invalid request (Invalid month ECs)</td></tr>
</table>
</td></tr>
<tr><td></td><td>Edit safe distance for buildings</td><td>Random testing</td><td></td><td>
<table>
<tr><th>Input</th><th>Output</th><th>Desired Output</th></tr>
<tr><td>-2</td><td>The value is set</td><td>Warning that negative numbers cannot be set.</td></tr>
<tr><td>600000000</td><td>The value is set</td><td>Warning or Error that the value is bigger than the maximum allowed (the actual building size).</td></tr>
<tr><td>2</td><td>The value is set</td><td>The value is set.</td></tr>
<tr><td>0</td><td>The value is not set</td><td>The value should be updated with 0.</td></tr>
<tr><td>-</td><td>The value is not set</td><td>Should update the value with “no restriction” or 0.</td></tr>
</table>
</td></tr>
<tr><td></td><td>User Sign in</td><td>Boundary Testing (BT)</td><td>Check if the app behaves accordingly when given different inputs which represent the bounds of the equivalence classes.</td><td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>valid credentials</td><td>user signed in (sees main page of the LightHouse app)</td></tr>
<tr><td>empty credentials</td><td>user not signed in (sees sign in page of the LightHouse app with two error messages, “Username is a required field.” under the</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td></td><td>
<table>
<tr><td></td><td>username input and “Please enter a password.” under the password input)</td></tr>
<tr><td>empty username + non-empty password</td><td>user not signed in (sees sign in page of the LightHouse app with one error message, “Username is a required field.” under the username input)</td></tr>
<tr><td>non-empty username + empty password</td><td>user not signed in (sees sign in page of the LightHouse app with one error message, “Please enter a password.” under the password input)</td></tr>
<tr><td>other credentials, i.e. non-empty username and password, but invalid</td><td>user not signed in (sees sign in page of the LightHouse app with no error message)</td></tr>
</table>
* credentials = username + password
</td></tr>
<tr><td></td><td>Add a booking</td><td>Quick-testing</td><td>Check quickly different types of dates in the calendar, paired with different requirements for the desk and building combinations</td><td></td></tr>
<tr><td></td><td>Browser Compatibility</td><td>Configuration/Compatibility Testing</td><td>Check if functionalities of the web app works properly in different browsers</td><td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>add new desk in chrome browser</td><td>the desk was successful added if the valid information to create a desk was provided, otherwise an error message</td></tr>
<tr><td>add new desk in firefox browser</td><td>the desk was successful added if the valid information to create a desk was</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>
<table>
<tr><td></td><td>provided, otherwise an error message</td></tr>
<tr><td>add new desk in safari browser</td><td>the desk was successful added if the valid information to create a desk was provided, otherwise an error message</td></tr>
<tr><td>add new desk in edge browser</td><td>the desk was successful added if the valid information to create a desk was provided, otherwise an error message</td></tr>
</table>
</td>
</tr>
<tr>
<td></td>
<td>Manage Equipment List</td>
<td>Configuration/Compatibility Testing</td>
<td>Create, update and remove an equipment from the list</td>
<td>
<table>
<tr><th>Input</th><th>Expected Output</th></tr>
<tr><td>Create new equipment with correct input</td><td>Add a new Equipment into list</td></tr>
<tr><td>Updating an equipment with a correct name</td><td>updating him</td></tr>
<tr><td>remove an equipment</td><td>removes it from the list</td></tr>
<tr><td>cancel updating in the middle of editing the name of an equipment</td><td>the equipment remains the same</td></tr>
<tr><td>Create new equipment with incorrect input</td><td>Error message is shown</td></tr>
<tr><td>Updating an equipment with a incorrect name</td><td>Error message is shown</td></tr>
</table>
</td>
</tr>
</table>

7.2.Test Implementation. Test Execution

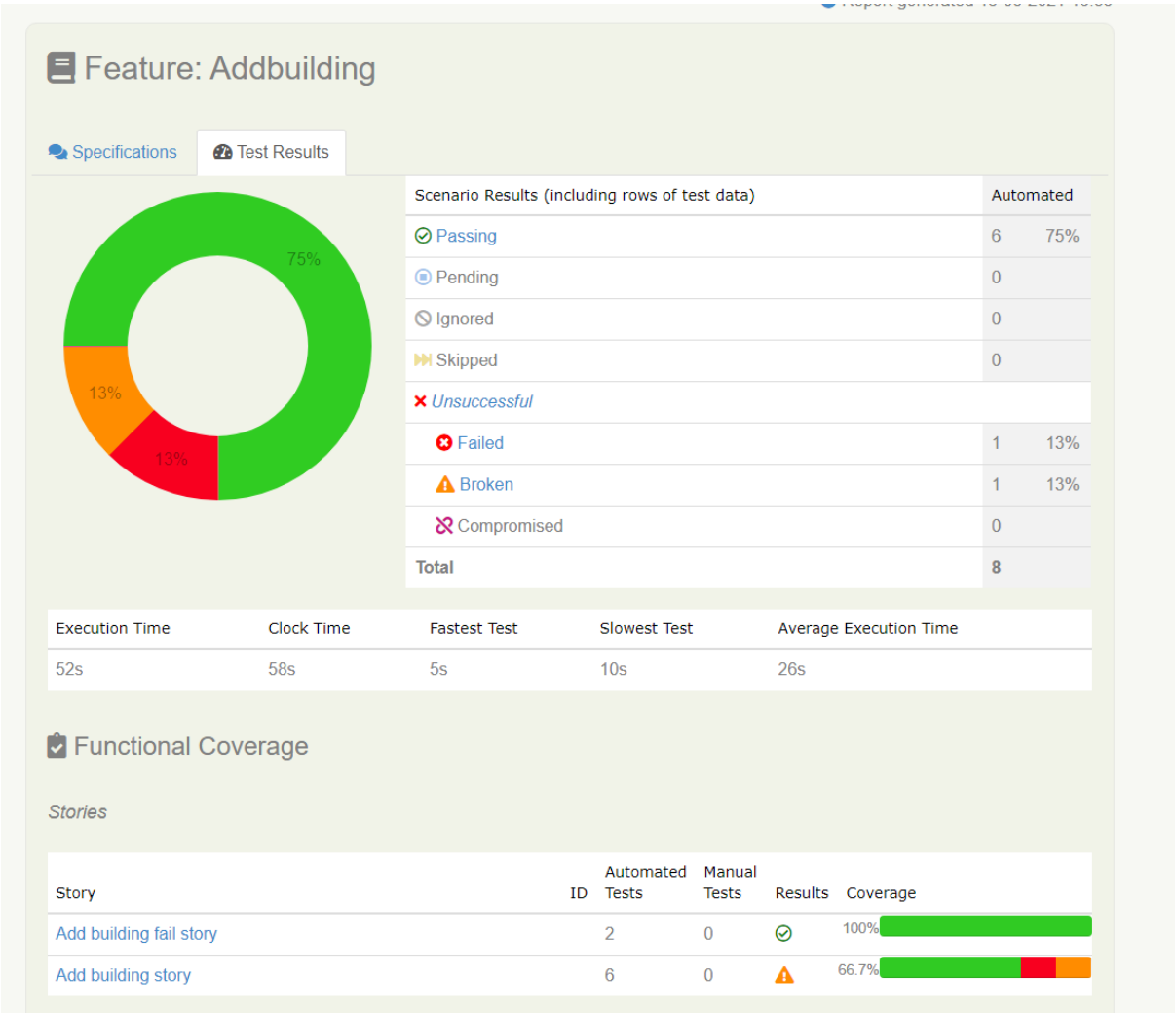
<table>
<tr><th>Part</th><th>Student</th><th>Feature(s)</th><th>Input, Expected Output, Actual Output</th></tr>
<tr><td>Part II</td><td></td><td>Add building</td><td>
<table>
<tr><th>TCs</th><th>Input</th><th>Expected Output</th><th>Actual Output</th></tr>
<tr><td>TC01</td><td>“eu”, “1”</td><td>building added</td><td>Passed</td></tr>
<tr><td>TC02</td><td>“eu”, “0”</td><td>building not added, error message</td><td>Passed</td></tr>
<tr><td>TC03</td><td>“eu”, “256”</td><td>building added</td><td>Passed</td></tr>
<tr><td>TC04</td><td>“ ”, “1”</td><td>building added</td><td>Passed</td></tr>
<tr><td>TC05</td><td>“eu”, “1.24”</td><td>building added</td><td>Passed</td></tr>
<tr><td>TC06</td><td>“eu”, “-1”</td><td>building added</td><td>Passed</td></tr>
<tr><td>TC07</td><td>“123”, “1”</td><td>building added</td><td>Passed</td></tr>
<tr><td>TC08</td><td>“eu”, “abc”</td><td>building not added, error message</td><td>Passed</td></tr>
<tr><td>~~TCX~~ ~~X~~</td><td>~~“eu”, “5 “~~</td><td></td><td>cannot be implemented</td></tr>
</table>
</td></tr>
<tr><td>Part II</td><td></td><td>Create a visitor account.</td><td>
<table>
<tr><th>TCs</th><th>Input</th><th>Expected Output</th><th>Actual Output</th></tr>
<tr><td>TC01</td><td>Account Details (Email, First name, Last name)</td><td>Visitor account created</td><td>Passed</td></tr>
<tr><td>TC02</td><td>Delete account (email)</td><td>Account is deleted</td><td>Passed</td></tr>
<tr><td>TC03</td><td>Same Account Details (Email, First name, Last name)</td><td>Visitor account created</td><td>Failed</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td>Part I</td>
<td></td>
<td>Login API</td>
<td>
<table>
<tr><th>Capability</th><th>Scenario</th><th>Steps</th><th>Time</th><th>Duration</th><th>Result</th></tr>
<tr><td>Function test</td><td>First test</td><td>2</td><td>14:38:44</td><td>00:03</td><td>✓</td></tr>
<tr><td>Function test</td><td>First test</td><td>2</td><td>16:38:15</td><td>00:04</td><td>✓</td></tr>
<tr><td>Function test</td><td>Test1</td><td>2</td><td>18:38:40</td><td>00:03</td><td>✓</td></tr>
<tr><td>Function test</td><td>Test2</td><td>2</td><td>18:38:44</td><td>00:00</td><td>✓</td></tr>
<tr><td>Function test</td><td>Test3</td><td>2</td><td>18:38:45</td><td>00:00</td><td>✓</td></tr>
<tr><td>Function test</td><td>Test4</td><td>2</td><td>18:38:45</td><td>00:00</td><td>✓</td></tr>
<tr><td>Function test</td><td>Test5</td><td>2</td><td>18:38:46</td><td>00:00</td><td>✓</td></tr>
<tr><td>Function test</td><td>Test6</td><td>2</td><td>18:38:46</td><td>00:00</td><td>✓</td></tr>
<tr><td>Search by keyword story</td><td>Searching by keyword apple should display the corresponding article</td><td>0</td><td>14:38:49</td><td>00:00</td><td>✓</td></tr>
<tr><td>Search by keyword story</td><td>Searching by keyword apple should display the corresponding article</td><td>0</td><td>16:38:21</td><td>00:00</td><td>✓</td></tr>
</table>
</td>
</tr>
<tr>
<td>Part III</td>
<td></td>
<td>User Sign In</td>
<td>
<table>
<tr><th>TCs</th><th>Input</th><th>Expected Output</th><th>Actual Output</th></tr>
<tr><td>TC01</td><td>valid username & password</td><td>user signed in</td><td>Passed</td></tr>
<tr><td>TC02</td><td>empty username & password</td><td>both error messages are shown</td><td>Passed</td></tr>
<tr><td>TC03</td><td>empty username + non-empty password</td><td>username error message is shown</td><td>Passed</td></tr>
<tr><td>TC04</td><td>non-empty username + empty password</td><td>password error message is shown</td><td>Passed</td></tr>
<tr><td>TC05</td><td>multiple non-empty username & password, but invalid</td><td>no error message is shown</td><td>Passed</td></tr>
</table>
* Please see 7. Test Design -> Part II table -> Pascan A. entry for more detailed inputs
</td>
</tr>
<tr>
<td>Part II</td>
<td></td>
<td>Resolution testing for sign in page</td>
<td>
<table>
<tr><th>TCs</th><th>Input</th><th>Expected Output</th><th>Actual Output</th></tr>
<tr><td>TC01</td><td>screen resolution (720x480)</td><td>  </td><td> 
Passed</td></tr>
<tr><td>TC02</td><td>screen resolution (1280x720)</td><td>  </td><td> 
Passed</td></tr>
</table>
</td>
</tr>
</table>

			TC03	screen resolution (1366x768)		 Passed
			TC04	screen resolution (1920x1080)		 Passed
			TC05	screen resolution (3840x2160)		 Passed
Part I		Edit Building				

TCs	Input	Expected Output	Actual Output
TC01	[asd] [123]	update	Passed
TC02	[asd] [123] (again)	no update error msg	cannot be implemented
TC03	[asd] [-123]	no update	Failed
TC04	[] [123]	no update	cannot be implemented
TC05	[asd] []	no update	cannot be implemented
TC06	[asd] [asd]	no update	Passed
TC07	[123] [123]	update	Passed
TC08	[asd] [0]	no update	Passed

7.3.Test Report



Feature: Addbuilding

Specifications | Test Results

Scenario Results (including rows of test data)	Automated	
Passing	6	75%
Pending	0	
Ignored	0	
Skipped	0	
Unsuccessful		
Failed	1	13%
Broken	1	13%
Compromised	0	
Total	**8**	

Execution Time	Clock Time	Fastest Test	Slowest Test	Average Execution Time
52s	58s	5s	10s	26s

Functional Coverage

Stories

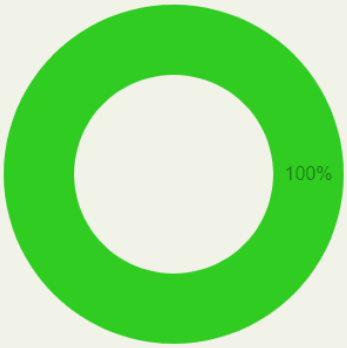
Story	ID	Automated Tests	Manual Tests	Results	Coverage
Add building fail story		2	0		100%
Add building story		6	0		66.7%

!!! TESTS RUN AND PASS INDIVIDUALLY, BUT AFTER SOME LOGINS, THE LIGHTHOUSE APPLICATION SIMPLY DOES NOT ALLOW LOGGING IN. THIS ISSUE COULD NOT BE SOLVED BY LOGGING IN IN A DIFFERENT METHOD ANNOTATED WITH @Before.

Test Results: All Tests

10 test scenarios |

Summary | Test Results



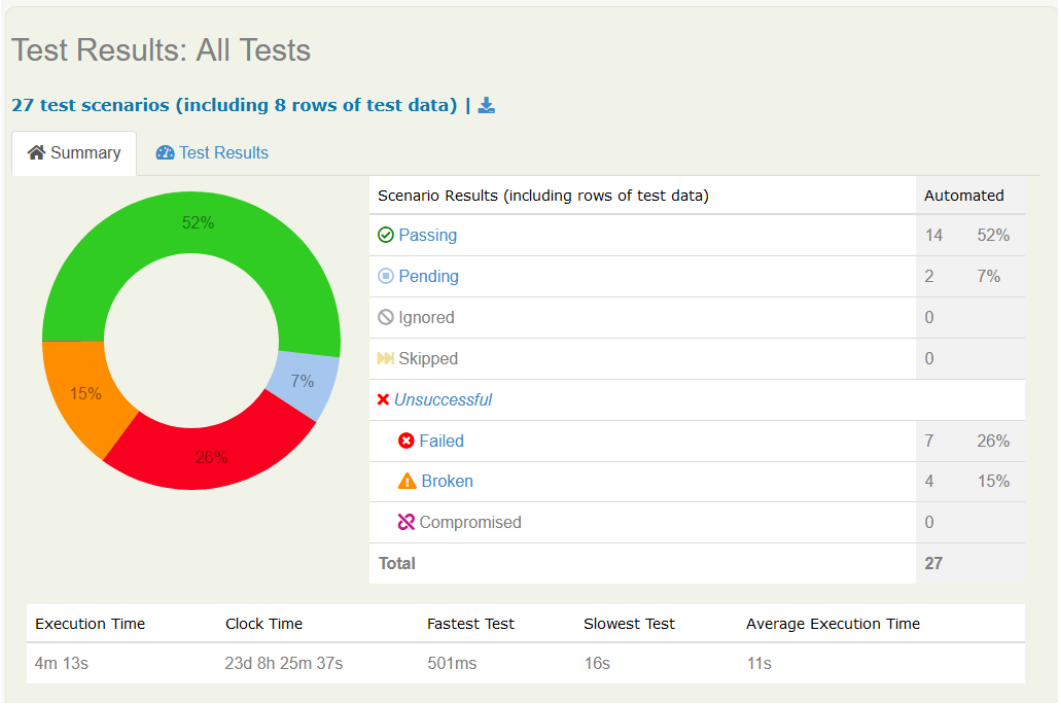
Scenario Results (including rows of test data)	Automated	
Passing	10	100%
Pending	0	
Ignored	0	
Skipped	0	
Unsuccessful		
Failed	0	
Broken	0	
Compromised	0	
Total	**10**	

Execution Time	Clock Time	Fastest Test	Slowest Test	Average Execution Time
15s	4h 2s	279ms	4s	1s

Functional Coverage Overview

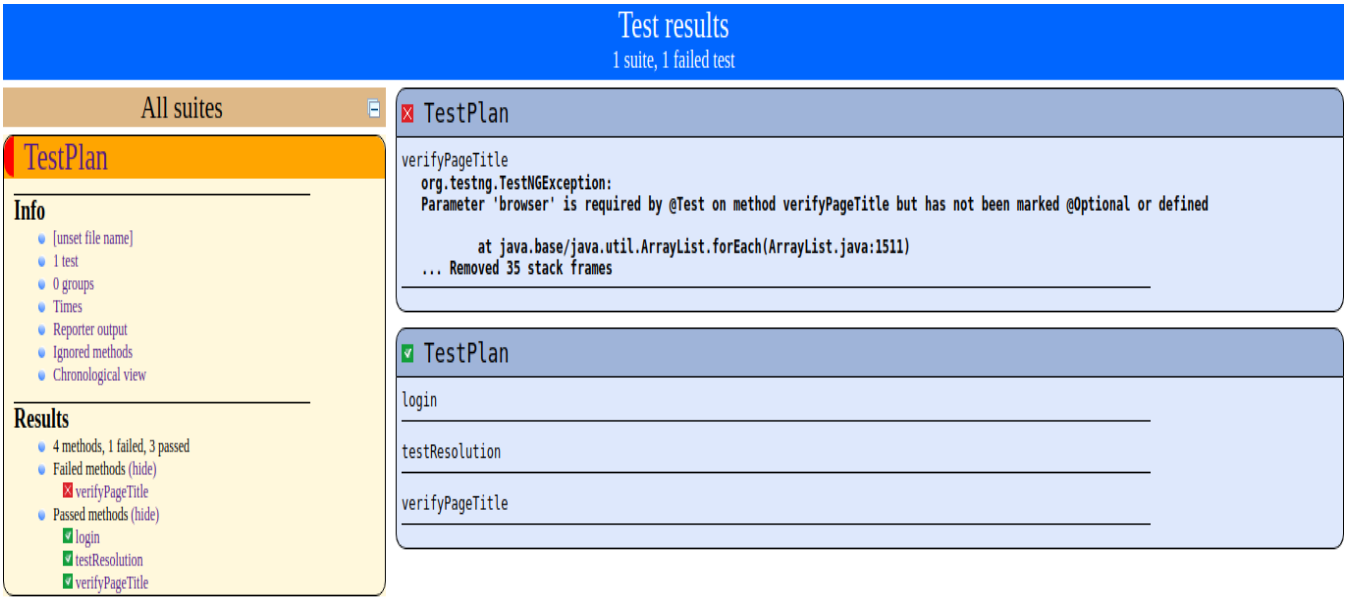
Features

Feature	Scenarios	% Pass	Result	Coverage
Search	10	100%		

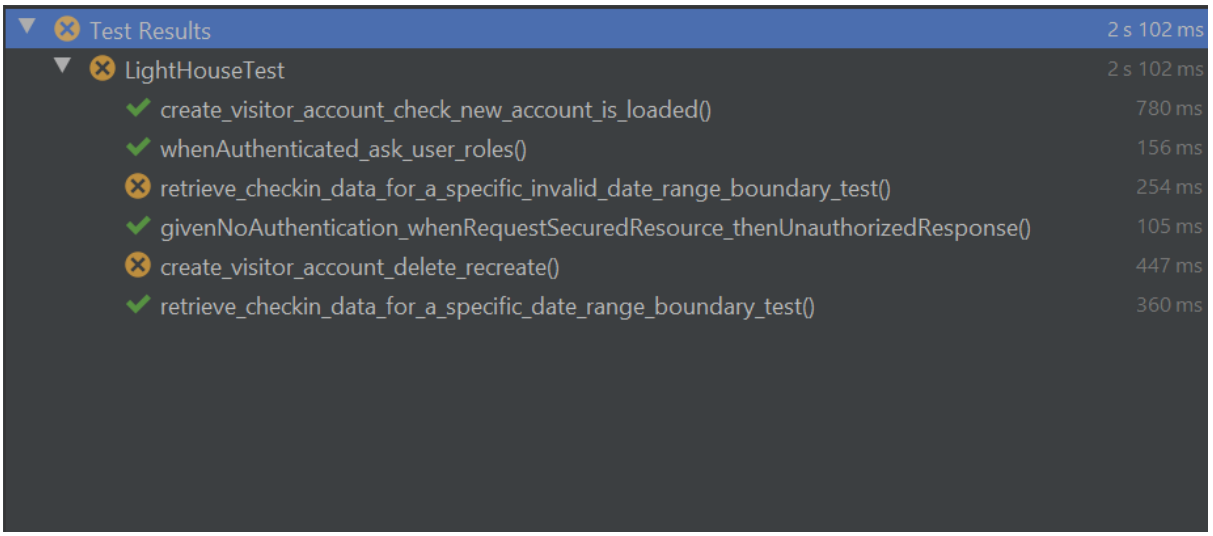


!!! For whatever reason, the prior tests from the skeleton project (Wiktionary search + others added on top of them) are kept in the Maven built, even though I deleted them from the project.

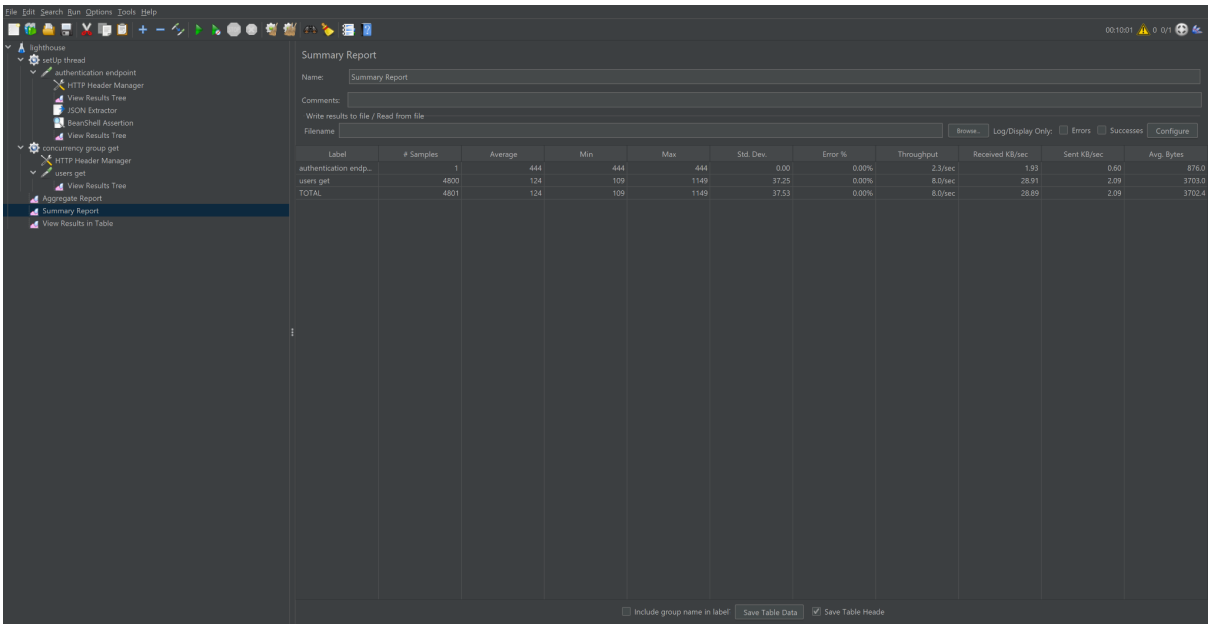
(report via surefire-reports)



RESTAssured Test Report



Load Test Report (JMeter)



8. Issue Reporting

1. *Failing scenario :*

Create a visitor account.

Delete the visitor account.

Try to create a visitor account with the same email.

Expected : A visitor account is created.
Actual : Internal Server Error.

2. *No reply attack protection. The api does not include a nonce or similar string to prevent a scenario where an attacker runs same request many times.*

See more : https://en.wikipedia.org/wiki/Replay_attack
https://www.sitepoint.com/how-to-prevent-replay-attacks-on-your-website/
https://crypto.stackexchange.com/questions/76875/why-does-a-nonce-prevent-a-replay-attack

3. *Cross-origin resource sharing (CORS) - Arbitrary Origin Trusted*

The application implements an HTML5 cross-origin resource sharing (CORS) policy that allows access from any domain. Allowing access from arbitrary domains means that those domains can perform two-way interaction with the application via the request. Rather than using a wildcard or programmatically verifying supplied origins, use a whitelist of trusted domains.

See more:
https://portswigger.net/web-security/cors/access-control-allow-origin

4. *No messages for invalid sign in :*

1. Go to https://lighthouse-demo.evozon.com/login .

2. Input an invalid non-empty username (in the username input field) and non-empty password (in the password field) combination (i.e. there is no account having these credentials).

3. Click the 'SIGN IN' button.

4. The page is reloaded.

5. No message is shown to the user.

Expected : The user is shown an error message, e.g. "Incorrect password." / "No account having this username." / etc.
Actual : No message is shown.

5. *When adding a building, save and cancel button move outside their space :*

1. Click add building button on buildings page

2.Resize the window to a width of 959 or smaller

Expected output:

The design should adapt itself, buttons should stay on the left

Actual output:

The buttons move to the right and bottom of the page

9. Conclusions. Lessons Learned

The type of application was appropriate as it had lots of features to choose from that are found in many applications making the project relevant for inexperienced students like us. The amount of required knowledge was balanced, i.e. not too low so that it wouldn't impose any problems, but also not high so that we would've had to look them up for hours.

The tools that we used, mainly Selenium WebDriver and different browsers, exposed enough setup issues for many of us that took more than expected to solve. One worth mentioning was the fact that with Firefox browser the app would get stuck in the loading state, whereas with Chrome it didn't (and we were not happy about it). The versioning is the reason behind why we didn't have a single GitHub project as we planned in the beginning.

The team collaboration was AMAZING (we used A for 'And say it with a neutral tone' from RIMGEA here, even if you may not believe us). We organized and split the work quite well. We fulfilled the tasks in a decent amount of time (we really don't know if we could've done better).

The collaboration with EVOZON was outstanding as we didn't ask a question for which we didn't get a response back. Thank you!