Test Plan

ASSOCIATED ENGINEERING

07, April 2020

Version:
1.0

Team:
AE3: Team Turtle

Team Members:
Bob Ghosh
Siddhartha Gupta
Eddie Huang
Jungwook Jang
Shuaiqi Zhang
Wenhong Zhang

Documented By:
Bob Ghosh
Jungwook Jang

Approved By:
Siddhartha Gupta
Eddie Huang
Jungwook Jang
Shuaiqi Zhang
Wenhong Zhang

Changelog:

03-12-2020, v0.1

Documenting the first implementation of the Test plan.

04-06-2020. v1.0

Revised the test plan as the project details have been changed.

Contents

Introduction

Scope

This document gives a detailed description of the overall test plan of the project. it describes how to set up the environment for the test, testing methodology including functional and non- functional components. it also describes details of testing related to security and data. and lastly it will discuss some of the regression and limitations of the current test plan.

Roles and Responsibilities

Role	Name
Front-end Dev	Jungwook Jang
Front-end Dev	Wenhong Zhang
Front-end Dev	Shuaiqi Zhang
Back-end Dev	Bob Ghosh
Back-end Dev	Eddie Huang
Database Manager	Siddhartha Gupta

Summary

For testing the DotNetCore 3.1 Web.API and the React Client, we will be using multitude of different tests. The objective of the tests will be to:

1. Check the Functional Properties of the Full-Stack Web Application (Functions and Commands).
2. Analyse the qualitative aspects of the application (Speed, Capacity, Response).
3. Check the integration of the system and the client.

The particular tests and program will be detailed in the following sections.

Test Environment Setup

Since we have two different sides of the application, and as we have mentioned in previous documents, development and testing are divided into two sections: frontend and backend. For backend we will be using Postman (v7.20.0) to test different routes, their response and the assertion scripts. Moreover, we are using the hosted backend site, https://turtle-ae.azurewebsites.net/. For frontend, we will be using Redux Devtools (v2.17.0) to debug our application's state changes, and test scripts will be provided for manual testing of user interface. For capacity test we will use web.dev tool. We are using the deployed frontend site https://turtle-319-ui.azurewebsites.net/ (for regular clients), and https://turtle-319-admin.azurewebsites.net/ (for admins).

Test Plan

Frontend

For front-end testing, we will use Redux Devtools and Chrome Devtools for debugging purpose. And integration test will be performed on each component (personal profile, project, search, forecasting and admin). The details of testing scripts are available in the **Appendix**. End-to-end testing will be performed at the end of the project phase with the testing scripts, and with the help of the user case scenario document (provided by AE) to ensure that the application meets sponsor's specification and requirements.

Backend

We usually test the functions and implementation during development, albeit, informally. This document is about the formal tests for functional and non-functional requirements. As mentioned in the previous section, we are going to use Postman for testing. The methodologies we are using are Whitebox UAT, Sequential Whitebox SAT and Manual testing, where we put requests for different routes, with valid body (if required). We have also used integrated Node.js testing with Postman for assertions. We will be combining separate assertion tests into multiple sequential automated request-based tests which fulfil the functional requirements put forward by AE.

Functional Test

Frontend

Disclaimer: All the details of the testing scripts are available in Appendix, and additional scripts were already provided by AE.

User personal profile component will be tested manually to ensure all edit functions for skills, location, disciplines work as expected.

Project component will be tested manually to ensure adding individual to projects, creating and deleting a project, update the project status, and assigning workload to individuals.

Search component will be tested manually to ensure the module correctly return the list of users.

Forecasting component will be tested manually to ensure that the module correctly displays current resource utilization forecasting resource utilization and appropriate metrics.

Administration component will be tested manually to ensure functions for creating and deleting datasets work correctly.

Backend

Component	Technique	Test	Risk
User	• Whitebox unit testing using	• Should be able to successfully create a User with valid user data.	• Database changes might not be technically shown.

	Node.js assertion inside Postman. • Manual testing and comparison with returned response body. • Automated testing with scripted sequential requests. • Number of Anticipated Scripts: 10	• Should be able to update a User's location and type with valid user data. • Should be able to delete a User with a valid username. • Should be able to get a User information with username. • Should not be able to create records with same User id and Username. • Should not be able to get information for deleted User.	• SQL Exceptions may be thrown if data is invalid, or not of expected form.
Project	• Whitebox unit testing using Node.js assertion inside Postman. • Manual testing and comparison with returned response body. • Automated testing with scripted sequential requests. • Number of Anticipated Scripts: 10	• Should be able to create a new Project with valid project data. • Should be able to update the Project's information with valid project data. • Should be able to delete the Project with project number. • Should be able to get information about a Project using the project name. • Should not be able to add projects with same id and number. • Should not be able to get information for a deleted project.	• Database changes might not be technically shown. • SQL Exceptions may be thrown if data is invalid, or not of expected form.
Personal Profile	• Whitebox unit testing using Node.js assertion inside Postman.	• Should be able to assign skills to a user. • Should be able to assign disciplines to user. • Should be able to update years of experience, disciplines, and skills attached to discipline.	• Database changes might not be technically shown. • SQL Exceptions may be thrown if data is invalid, or

	• Manual testing and comparison with returned response body. • Automated testing with scripted sequential requests. • Number of Anticipated Scripts: 20	• Should be able to add user to projects. • Should be able to update a project for a user. • Should be able to delete all the skills attached to discipline, particular skill or a project. • Should be able to get information about the projects of a user. • Should be able to get information about the skills, discipline and years of experience of a user. • Should only be able to add Skills belonging to particular discipline. • Should not be able to return deleted information	not of expected form.
Search	• Whitebox unit testing using Node.js assertion inside Postman. • Manual testing and comparison with returned response body. • Automated testing with scripted sequential requests. • Number of Anticipated Scripts: 10	• Should be able to return Users matching the Search Profile data from the frontend client. • Should be able to return Projects matching the Search Profile data from the frontend client. • Should return 204 when search data is invalid.	. • SQL Exceptions may be thrown if data is invalid, or not of expected form.
Forecast	• Whitebox unit testing using Node.js assertion	• Should be able to return required forecasted data for user or groups of users.	• SQL Exceptions may be thrown if data is invalid, or not of expected form.

	inside Postman. • Manual testing and comparison with returned response body. • Automated testing with scripted sequential requests. • Number of Anticipated Scripts: 10		

Non-Functional Test

UI Tests

For user interface, UI testing will be performed together when the functional tests for frontend is performed. These will be analysed to ensure that the data is displayed correctly. Moreover, UI testing will be only accepted if the UI display correctly data or any updates.

Capacity Test

Capacity tests will be performed to ensure the performance of our application meets the specification at the stage of the end-to-end testing using the web.dev tool (which is available online). The capacity test will include load test, stress test, and spike test. The acceptance criteria for the capacity that we are aiming is 500 users can use the project simultaneously.

Usability Test

Once we are done implementing the functions and the completely integrating the client and the server, we will be looking for Ease of Use, and testing if the UI and the functional elements can be easily understood. The acceptance criteria for the usability test will be accepted upon the feedback from other uses, testers from CPSC 319 courses.

Test Script

Frontend

Please check the **Appendix** for test setup, and steps with the expected output.

Backend

The unit tests for the functional testing works like following:

- Creating new users, projects, skills or discipline would firstly send a POST request to the backend server. The backend server should respond with 201 (Created). We will be testing for the response code, and the response

data. We will be comparing the data in the body posted and the body of the response. Creating with invalid data should return 500 (Internal Server Error).

- Updating information for user, project, skills, or discipline would firstly send a PATCH request to the backend server. The backend server should respond with 202 (Accepted). We will be testing for the response code, and the response data. We will be comparing the data in the body posted and the body of the response. Updating with invalid data should return 500 (Internal Server Error).
- Deleting user, project, skills or discipline would firstly send a DELETE request to the backend server. The backend server should respond with 200 (Ok). We will be testing for the response code, and in some cases, response body. We will be testing them with GET commands to make sure the data was deleted. Deleting an already deleted data should either return nothing (204, No Content), or the request body.
- Getting user, project, skills or discipline for a user, or in general, would firstly send a GET request to the backend server. The backend server should respond with 200 (Ok). We will be testing for the response code and the response body. As mentioned above, if GET-ting information about a deleted record, the response should either be 204 or a body with requested data.
- The tests for the Search and Forecast Module are to be implemented. But, in simple terms, the backend server will be requested with PUT and valid search data in the request body. The valid requested body will be based on the pre-defined Search model. The response code for comparison will be 202 (Accepted), and the body will either be list of users matching the criteria or the list of projects. If the requested search data is invalid, the response will be 500, and if nothing matches the criteria, the response body should be empty. These criterions will be used for assertions and comparisons.
- The tests for the Forecast Module are to be implemented. But it will be similar to the search module, where we request a PUT with the forecast criterion, and the backend server should return 202 or 500 based on validity.

The **Appendix** will have a more detailed information about the aforementioned tests, including the format of the body and the response body. It will also include the scripts we use to test on Postman. We conducted some of testing in sequential order, to test the system. These tests will be combined for automated testing, fitting the scenarios recommended by AE.

Security Test and Test Data approach

Security Test

- Since we are using Azure AD to authenticate all user, it is not our responsibility to test for regular user (client) authentication. However, we have security tests for administrator component (details in the **Appendix**), since we are creating a table inside the SQL Server Database for comparison and authentication of admins.
- To prevent input is data being corrupted by user, we introduce auto-complete options with dynamically updated on some input fields to create or edit various datasets. Moreover, we guide user with default input value to inform valid input formats.

Test Data Approach

We will be conducting tests based on the User-Case scenarios provided to us by AE, our sponsor. For unit tests, we have, and we will be using multiple sets of data of different sizes. However, the User-Case scenarios does provide us with information regarding typical use cases.

Regression Test

Regression test will give us information about basic capacity of our application, and analytical metrics about the relationship between input requests and output response time. We will be testing with 1000+ inputs, like creating many

users, adding skills, disciplines, creating projects, and then deleting all data. We will be looking for performance degradation, endurance and response.

Limitation of Test

There are certain limitations of using the test methodologies and steps we are going to employ. Firstly, we cannot check the database in the SQL server for changes, thus, we will be relying on the functions we created to see the changes. Secondly, as with most application tests, qualitative analysis and metrics will be system dependent, this is especially a concern since we will be using cloud hosted system to do some of our testing.

Appendix

Frontend

1. Admin Page Test Scripts

Functional/Non-functional	Functional Area	Test Name	Test Steps	Expected Output
Non-functional	Sign-up	Successfully sign-in	Enter Valid Admin name and password	Admin is logged in successfully
Non-functional	Sign-up	Sign-in with invalid password	Enter invalid admin name or password	Error message that contains check username or password
Functional	changing admin information	change password	click edit button and enter current password and new password	message that contains password is successfully changed
Non-functional	Log-out	successfully log-out	click the logout buttons	Admin is successfully logged out
Functional	Create User	Successfully create user	go to user section and click create user and enter necessary information to create user	successfully displayed New user in the user table
Functional	Create User	Failed to Create duplicate User	go to user section and click create user and enter user information	message that the user is already in the database.

			that already in the user table	
Functional	Delete User	Successfully delete user	search a user to be deleted in the user table and the click delete button	User table will be re-rendered without the user that has been deleted
Functional	Create Skills	Successfully create Skill	go to skill section and click create skill and enter necessary information to create skill	successfully displayed New skill in the skill table
Functional	Create Skill	Failed to Create duplicate Skill	go to user section and click create skill and enter skill information that already in the skill table	message that the skill is already in the database.
Functional	Delete Skill	Successfully delete Skill	search a skill to be deleted in the skill table and the click delete button	User table will be re-rendered without the skill that has been deleted
Functional	Create Location	Successfully create location	go to location section and click create location and enter necessary information to create location	successfully displayed New location in the location table
Functional	Create Location	Failed to Create duplicate location	go to location section and click create location and enter location information that already in the location table	message that the location is already in the database.
Functional	Delete Location	Successfully delete location	search a location to be deleted in the location table and the click delete button	Location table will be re-rendered without the location that has been deleted

Functional	Create Discipline	Successfully create discipline	go to discipline section and click create and enter necessary information to create discipline	successfully displayed New discipline in the discipline table
Functional	Create Discipline	Failed to Create duplicate discipline	go to discipline section and click create discipline and enter location information that already in the discipline table	message that the discipline is already in the database.
Functional	Delete Discipline	Successfully delete discipline	search a discipline to be deleted in the discipline table and the click delete button	discipline table will be re-rendered without the discipline that has been deleted

2. User profile Page Test Script

Functional/Non-functional	Functional Area	Test Name	Test Steps	Expected Output
Non-functional	Azure authentication	authentication success	enter correct account and password to login	redirect to Application website
Non-functional	Azure authentication	authentication fails	enter incorrect account and password to login	Error message that you have to check username or password
Non-functional	User Type	PM login	Log in with PM account	the application will show the project page for PM
Non-functional	User Type	Normal user login	Log in with normal user account	the application will hide the project page
Non-functional	Log out	Log-out successfully	click logout button	will direct to Azure login page

Non-functional	Load User basic information	Show location and name	Click profile component	will display location and name of the user
Non-functional	Load User basic Information	Show Availability	Click profile component	will display availability of the user
Non-functional	Load User basic information	Show Discipline	Click profile component	will display discipline of the user
Non-functional	Load User basic information	Show project	Click profile component	will display all the project that the user has been assigned
Functional	Edit Location	Successfully change location	Enter new valid location and submit	the new location will be changed to new location and will be displayed in the user profile
Functional	Edit Location	Failed to change location	Enter invalid location and submit	Error message that the location is not in the server
Functional	Edit Skill	Successfully add skill	Enter new valid skill and submit	the new skill will be added to and will be displayed in the user profile
Functional	Edit Skill	Failed to add skill	Enter invalid skill and submit	Error message that the skill is not in the server
Functional	Edit Skill	Successfully delete skill	click delete button in the table	the deleted skill will be removed from the user skill table
Functional	Edit Discipline	Successfully add discipline	Enter new valid discipline and submit	the new discipline will be added to and will be displayed in the user profile
Functional	Edit Discipline	Failed to add discipline	Enter invalid discipline and submit	Error message that the discipline is not in the server

Functional	Edit Discipline	Successfully delete Discipline	click delete button in the table	the deleted discipline will be removed from the user discipline table

3. Search module Test Script

Scenario#1 (use search module to add user to the project)

Setup: Create a project and set all disciplines that are required in the project and Create 4 User that have all same disciplines and skills.

Steps: Enter the disciplines, skills all other information that is required for search

Expected Output: The list of User will be returned with the size at least 4

Scenario#2 (use search module with all empty column)

Setup: N/A

Steps: leave all empty column and click search

Expected Output: The list of users will be returned with the size 0

4. Project module Test Script

Functional/Non-functional	Functional Area	Test Name	Test Steps	Expected Output
Non-functional	table expand	project expandable	click project rows	project details will be displayed
Non-functional	table expand	project expandable	click expanded rows	project details will be collapsed
Functional	add project	successful adding	add a project with the proper attributes, refresh the current page	PM can see the newly added project in the list
Functional	add project	failed adding	add a project with the absence of some attributes, refresh the current page	the newly added project shouldn't be there.

Functional	delete project	successfully delete	delete an existing project, then refresh page	this project should be deleted from the PM project page
Functional	edit project info	edit project location	click edit button for project, change location, then refresh page	location should change for this project
Functional	edit project info	edit project id	click edit button for project, change id, then refresh page	id should change for this project
Functional	edit project info	edit project name	click edit button for project, change name, then refresh page	name should change for this project
Functional	edit project info	edit project start date	click edit button for project, change start date, then refresh page	start date should change for this project
Functional	edit project info	edit project end date	click edit button for project, change end date, then refresh page	end date should change for this project
Functional	user manipulation	successful assign user	click add user button, input a existing user id, with normal utilization info	user should appear in the user list of projects
Functional	user manipulation	failed add user	click add user button, input a non-existing user id, with normal utilization info	user should not appear in the user list after refreshing
Functional	Create project	Successfully create a project	Click create button and enter all required information and submit	Successfully created project and displayed in project table

Functional	Create project	Failed to create a project	Click create button and leave it all required information columns then submit	Error message that you should fill all required information columns

5. Forecast module Test Script

Scenario#1 (Resource Forecasting in a project)

Setup: Create 4 Projects and Create 4 Users for each project. Assign workload to each user in the projects.

Steps: set the period for forecasting and click the forecasting

Expected output: the returned table will display all resource utilization within the periods.

Scenario#2(the manpower color coding in forecasting module)

Setup: User same 4 user from the scenario#1.

Steps: set the period for forecasting and click the forecasting

Expected Output: the returned table will display manpower utilization with color code (e.g., if user is underutilization then yellow color, if the user is between 0.5 and 1.2 then green color, and lastly, if the user is above 1.2 then it will be displayed with red color).

Backend

Function	Request	Test Script	Expected Output

Creating a User	POST http://turtle-ae.azurewebsites.net/users/
	
	Body: {
	“id”: 85,
	“organization”: ”Organization1”,
	“firstName”: “Bob”,
	“lastName”: “Ghosh”,
	“username”: “bobg”,
	“location”: “Vancouver”,
	“type”: “Individual Contributor”
	}
		pm.test("Status code is 201", function () {
		pm.response.to.have.status(201);
		});
		pm.test("Your test name", function () {
		var res = "bobg";
		var jsonData = pm.response.json();
		
		pm.expect(jsonData.username).to.eql(res);
		});
			Response: 201 Created
			{
			“id”: 85,
			“firstName”: “Bob”,
			“lastName”: “Ghosh”,
			“username”: “bobg”,
			“location”: “Vancouver”,
			“type”: “Individual Contributor”
			
			“Organization”: “Organization1”
			}

Assigning Discipline and Skill to a User	POST http://turtle-ae.azurewebsites.net /personal/
	
	Body: {
	“username”: “bobg”,
	“skill”: “Quality Management”,
	“discipline”: “Services”,
	“yoe”: 1
	}
		pm.test("Status code is 201", function () {
		pm.response.to.have.status(201);
		});
		pm.test("Your test name", function () {
		var res1 = "bobg";
		var res2 = “Quality Management”;
		var res3 = “Services”;
		var jsonData = pm.response.json();
		
		pm.expect(jsonData.username).to.eql(res);
		pm.expect(jsonData.skill).to.eql(res2);
		
		pm.expect(jsonData.discipline).to.eql(res3);
		});
			Response: 201 Created
			{
			“username”: “bobg”,
			“skill”: “Quality Management”,
			“discipline”: “Services”,
			“yoe”: 1
			}

Creating a Project	POST http://turtle-ae.azurewebsite.net/projects/
	
	Body: {
	"id": 101,
	"number": "2010-FAKE-1",
	"title": "Fake Project One.",
	"locationId": 8,
	"createdAt": "2020-02-28T01:48:11.8458104",
	"updatedAt": "2020-02-28T01:48:11.8458104",
	"startDate": "2020-02-28T00:00:00",
	"endDate": "2020-02-28T00:00:00",
	"hours": 240
	}
		pm.test("Status code is 201", function () {
		pm.response.to.have.status(201);
		});
		pm.test("Your test name", function () {
		var res = "2010-FAKE-1";
		var jsonData = pm.response.json();
		
		pm.expect(jsonData.number).to.eql(res);
		});
			Response: 201 Created
			{
			"id": 101,
			"number": "2010-FAKE-1",
			"title": "Fake Project One.",
			"locationId": 8,
			"createdAt": "2020-02-28T01:48:11.8458104",
			"updatedAt": "2020-02-28T01:48:11.8458104",
			"startDate": "2020-02-28T00:00:00",
			"endDate": "2020-02-28T00:00:00",
			"hours": 240
			}

Assigning a Project to a User	POST http://turtle-ae.azurewebsite.net/userprojects/bobg
	
	Body:
	{
	"project": "Fake Project One.",
	"location": "Vancouver",
	"fromDate": "2020-02-28T00:00:00",
		pm.test("Status code is 201", function () {
		pm.response.to.have.status(201);
		});
		pm.test("Your test name", function () {
		var res = "Fake Project One.";
		var jsonData = pm.response.json();
		
		pm.expect(jsonData.project).to.eql(res);
		});
			Response: 201 Created
			{
			"project": "Fake Project One.",
			"location": "Vancouver",
			"fromDate": "2020-02-28T00:00:00",

	"toDate": "2020-02-28T00:00:00", "Hours": 20 }		"toDate": "2020-02-28T00:00:00", "hours": 20 }

Updating Project for a User	PATCH https://turtle-ae.azurewebsites.net/userprojects/bobg
	
	Body: {
	"project": "Fake Project One.",
	"location": "Vancouver",
	"fromDate": "2020-02-28T00:00:00",
	"toDate": "2020-02-28T00:00:00",
	"Hours": 35
	}
		pm.test("Status code is 202", function () {
		pm.response.to.have.status(202);
		});
		pm.test("Your test name", function () {
		var res = 35;
		var jsonData = pm.response.json();
		pm.expect(jsonData.hours).to.eql(res);
		});
			Response: 202 Accepted
			{
			"project": "Fake Project One.",
			"location": "Vancouver",
			"fromDate": "2020-02-28T00:00:00",
			"toDate": "2020-02-28T00:00:00",
			"Hours": 35
			}

Deleting the User from Projects	DELETE https://turtle-ae.azurewebsites.net/userprojects/bobg
	
	Body: {
	"project": "Fake Project One.",
	"location": "Vancouver",
	"fromDate": "2020-02-28T00:00:00",
	"toDate": "2020-02-28T00:00:00",
	"Hours": 35
	}
		pm.test("Status code is 200", function () {
		pm.response.to.have.status(200);
		});
		pm.test("Your test name", function () {
		var res = 35;
		var jsonData = pm.response.json();
		pm.expect(jsonData.hours).to.eql(res);
		});
			Response: 200 Ok
			{
			"project": "Fake Project One.",
			"location": "Vancouver",
			"fromDate": "2020-02-28T00:00:00",
			"toDate": "2020-02-28T00:00:00",
			"hours": 35
			}

Getting User Project Information Post-Deletion	GET https://turtle-ae.azurewebsites.net/userprojects/bobg	pm.test("Status code is 200", function () {
		pm.response.to.have.status(200);
		});
		pm.test("Your test name", function () {
		var res = [];
		var jsonData = pm.response.json();
		pm.expect(jsonData).to.eql(res);
		});
			Response: 200 Ok
			[]

Deleting Skills and Discipline of a User	DELETE https://turtle-ae.azurewebsites.net/personal
	
	Body: {
	“username”: “bobg”,
	“skill”: “Quality Management”,
	“discipline”: “Services”,
	“yoe”: 1
		pm.test("Status code is 200", function () {
		pm.response.to.have.status(200);
		});
		pm.test("Your test name", function () {
		var res = [];
		var jsonData = pm.response.json();
		pm.expect(jsonData).to.eql(res);
		});
			Response: 200 Ok
			[]

	}		

Getting the User's assigned Skill and Discipline Post-Deletion	GET https://turtle-ae.azurewebsites.net/personal/bobg	pm.test("Status code is 200", function () {
		  pm.response.to.have.status(200);
		});
		pm.test("Your test name", function () {
		  var res = [];
		  var jsonData = pm.response.json();
		  pm.expect(jsonData).to.eql(res);
		});
			Response: 200 Ok
			[]

Updating the User's Location and Type	PATCH https://turtle-ae.azurewebsites.net/users
	
	Body: {
	"id": 85,
	"firstName": "Bob",
	"lastName": "Ghosh",
	"username": "bobg",
	"location": "Kelowna",
	"type": "Resource Manager"
	}
		pm.test("Status code is 202", function () {
		  pm.response.to.have.status(202);
		});
		pm.test("Your test name", function () {
		  var res1 = "Kelowna";
		  var res2 = "Resource Manager"
		  var jsonData = pm.response.json();
		
		pm.expect(jsonData.location).to.eql(res1);
		  pm.expect(jsonData.type).to.eql(res2);
		});
			Response: 202 Accepted
			{
			  "id": 85,
			  "firstName": "Bob",
			  "lastName": "Ghosh",
			  "username": "bobg",
			  "location": "Kelowna",
			  "type": "Resource Manager"
			}

Deleting the User	DELETE https://turtle-ae.azurewebsites.net/users/bobg	pm.test("Status code is 200", function () {
		  pm.response.to.have.status(200);
		});
		pm.test("Your test name", function () {
		  var res1 = "Kelowna";
		  var res2 = "Resource Manager"
		  var jsonData = pm.response.json();
		
		pm.expect(jsonData.location).to.eql(res1);
		  pm.expect(jsonData.type).to.eql(res2);
		});
			Response: 200 Ok
			{
			  "id": 85,
			  "firstName": "Bob",
			  "lastName": "Ghosh",
			  "username": "bobg",
			  "location": "Kelowna",
			  "type": "Resource Manager"
			}

Updating the Project	PATCH https://turtle-ae.azurewebsites.net/projects
	
	Body: {
	"id": 101,
	"number": "2010-FAKE-1",
	"title": "Fake Project One.",
	"locationId": 21,
	"createdAt": "2020-02-28T01:48:11.8458104",
	"updatedAt": "2020-02-28T01:48:11.8458104",
		pm.test("Status code is 202", function () {
		  pm.response.to.have.status(202);
		});
		pm.test("Your test name", function () {
		  var res1 = 21;
		  var res2 = 300
		  var jsonData = pm.response.json();
		
		pm.expect(jsonData.locationId).to.eql(res1);
			Response: 202 Accepted
			{
			"id": 101,
			"number": "2010-FAKE-1",
			"title": "Fake Project One.",
			"locationId": 21,
			"createdAt": "2020-02-

	"startDate": "2020-02-28T00:00:00", "endDate": "2020-02-28T00:00:00", "hours": 300 }	pm.expect(jsonData.hours).to.eql(res2); });	28T01:48:11.8458104", "updatedAt": "2020-02-28T01:48:11.8458104", "startDate": "2020-02-28T00:00:00", "endDate": "2020-02-28T00:00:00", "hours": 300 }

Deleting the Project	DELETE https://turtle-ae.azurewebsites.net/projects/2010-FAKE-1	pm.test("Status code is 200", function () {
		pm.response.to.have.status(200);
		});
		pm.test("Your test name", function () {
		var res1 = 21;
		var res2 = 300
		var jsonData = pm.response.json();
		
		pm.expect(jsonData.locationId).to.eql(res1);
		
		pm.expect(jsonData.hours).to.eql(res2);
		});
			Response: 200 Ok
			{
			"id": 101,
			"number": "2010-FAKE-1",
			"title": "Fake Project One.",
			"locationId": 21,
			"createdAt": "2020-03-12T21:23:18.8201821",
			"updatedAt": "2020-03-12T21:23:18.8201821",
			"startDate": "2020-02-28T00:00:00",
			"endDate": "2020-02-28T00:00:00",
			"hours": 300
			}

Getting the Deleted Project	GET https://turtle-ae.azurewebsites.net/projects/2010-FAKE-1	Script:
		pm.test("Status code is 204", function () {
		pm.response.to.have.status(204);
		});
			Response: 204 No Content