CS3300: Introduction to Software Engineering

Test Plan for RESTful API

Version 0.0: 6/9/15
Team: RESTful API

Contents

1 Introduction

1.1 Overview (Executive Summary)

We can ensure the quality of the testing process by testing every API call thoroughly. Our API has a finite number of test conditions. In addition, each condition has an expected JSON response. If the response from the API is not as expected, an error has been detected. If every case is tested, the API is working as expected.

1.2 Assumptions

The system was designed with the assumption that clients will be able to use HTTP requests with the JSON mime-type. This format will be used in the manipulation of resources. In order for clients to interact with our service, they will be provided with the information to do so. Client's will have access to class and sequence diagrams. This will define the classes, and inform the clients of class attributes. The class attributes will define the resource representation to be used by its JSON format. If a team cannot make requests with the JSON format, a different data format will need to be agreed upon.

1.3 Definitions and Acronyms

Term	**Definition**
Test #	Test Case Number / Identifier
Requirement	Requirement that the test cases are validating (number / identifier)
Action	Action to perform or input to produce
Expected Result	Result expected when action is complete
Actual Result	What was actually seen
P / F	Pass / Fail indicator. Checkmark = Pass. "F" = Fail
Notes	Additional notes, error messages, or other information about the test.

1.4 References

NA

1.5 Items Not Covered By These Test Cases

NA

1. Quality Control

1.6 Test Plan Quality

We can ensure the quality of the testing process by testing every API call thoroughly. Our API has a finite number of test conditions. In addition, each condition has an expected JSON response. If the response from the API is not as expected, an error has been detected. If every case is tested, the API is working as expected.

1.7 Adequacy criterion

Adequacy of testing will be determined by its structural coverage and functionality. Adequate coverage of structure includes accounting for possible expected errors from user input. Adequacy of code functionality will be determined by the correct mapping of JSON responses to client requests.

1.8 Bug Tracking

We will use GitHub's bug tracking software to track bugs in our software and implement desired features. In addition, intermediate unit testing to specify the error. The software allows the entire team to post information about bugs, including the source of the bug, its status, who is currently trying to fix it, and some information on how it was finally fixed.

2. Test Strategy

1.9 Testing Process

Unit Testing:
Similar to a top down approach. Start with basic location then add more complex parameters, assert that the responses are expected based on the requests.

Integration Testing: Bottom Up [Request Route → Route Controller → Model → DB → Response Controller]
Our integration testing will be using a bottom up strategy. Starting from a User request, the test will assert that the request route reaches the corresponding Controller. The Controller will then pass on the request information onto the corresponding Model. Our testing will assert that the Model queries the database (DB) properly. The DB will pass response information to our Response Controller

and finally send a response to our user. We will be assert that each of these modules communicate correctly with one another.

System Testing:

Our system testing will assert that our API can communicate with each of the client. Our general strategy for testing functional requirements is using both an external test, POSTMan, and an internal tool, JUnit. POSTMan will simulate actual clients interacting with our API, while JUnit tests will be more thorough internal tests. Unit and Integration testing will be divided amongst the team as modules and route handling development is assigned. System testing will be a collaboration amongst all teams with our team.

1.10 Technology

Our application is written in Java, so we intend to use JUnit testing to test our API calls. We will use POSTman to test our requests and responses, as it can simulate an external client interacting with our system

3. Test Cases

Date test performed: _____
Tester: _____

Test #	Requirement or Purpose		Expected Result	Actual Result	P/F	Notes
0.a	User log in	Username through URI	User ID returned to external client			
1.a	Create user	Username through URI	Database updated with new user. New user JSON returned to external client			
1.b	Read user	Username through URI	JSON user returned to external client			

1.c	Update user	JSON user as parameter	Updated JSON user returned to external client			
1.d	Delete user	Username through URI	Updated database without the requested user. Return deleted user JSON to external client			
2.a	Create group	Group name through URI	Database updated with new group. New group JSON returned to external client			
2.b	Read group	Group name through URI	JSON group returned to external client			
2.c	Update group	JSON group as parameter	Updated JSON group returned to external client			
2.d	Delete group	Group name through URI	Updated database without the requested group. Return deleted group JSON to external client			
3.a	Create message	Message data, sender ID, and reciepient ID	Database updated with new message. New message JSON returned to external client			
3.b	Read message	Message ID through URI	JSON message returned to external client			
3.c	Update message	JSON message as parameter	Updated JSON message returned to external client			

3.d	Delete message	Message ID through URI	Updated database without the requested message. Return deleted message JSON to external client			
4.a	Create document	Message ID, Content .mime, and content passed as JSON parameter	Database updated with new document. New document ID returned to external client			
4.b	Read document	Document ID through URI	JSON document returned to external client			
4.c	Update document	JSON document as parameter	Updated JSON document returned to external client			
4.d	Delete document	Document ID through URI	Updated database without the requested document. Return deleted document JSON to external client			