**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* README FILE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

|  |  |  |  |
| --- | --- | --- | --- |
| Document Name | **ReadMe.docx** | Course | **CSE564: Hypermedia Services Assignment** |
| Written by: | Varun Gaur (1210414176) |
| Submitted to: | Dr. Frank Calliss | Date | 23rd April 2016 |

## CONTENTS

1. DELIVERABLES
2. PROCESS FOLLOWED FOR IMPLEMENTING CRUD
3. HOW TO BUILD & EXECUTE
4. MINIMUM SYSTEM REQUIREMENTS
5. FEEDBACK & UPDATES

## DELIVERABLES

Following files are delivered for this assignment:

* *HATEOAS-Appeals-vgaur1-NetBeans.zip*
* *ReadMe.docx*

## PROCESS FOLLOWED FOR IMPLEMENTATION

Two separate programs has been delivered for this assignment. One for REST Client and another for REST Server(HATEOAS). Please find below the clarification for each:

## RESTful Server:

## This is a RESTFul Web server which implements RMM-Level3 architecture. It ensure transition of resource from one application state to another. Here the process is initiated by an ENTRY\_POINT\_URI and based on Client requests traverse through different states of a resource.

## Key points of this State transition are as follows:

## ENTRY POINT URI for server is <http://localhost:8080/HATEOAS-Appeals-vgaur1-NetBeans-Server/webapi/gradeAppealResource/>

## Initial state of Resource will be UNGRADED.

## END POINT of server are APPEAL\_ACKNOWLEDGED and APPEAL\_ABANDONED denoted by “E” in state transition diagram.

## Further details of state transition is given in below diagram:

## STATE TRANSITION DIAGRAM

## 

## RESTful Client:

## This is a test driver which generates Client Request and runs through different scenarios one after the other without asking for an end user provide input. It generates output on console with details for each Test Cases. I have covered following test cases:

## HAPPY PATH TEST CASE-1: This TEST CASE send following requests to server:

## *[PROFESSOR] CREATES a Grade:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – APPEAL\_EXPECTED

## *[STUDENT] ADD an Appeal:* Student now updates the Appeal Comment. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] UPDATES already created Appeal:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] SUBMITS the created Appeal:* Student submits the created appeal. Transition to state – PENDING\_REVIEW

## *[PROFESSOR] UPDATES a Grade:* Professor review the appeal comments and submitted assignment/exam document and found that student is eligible for marks update, hence update the grade for student. Transition to state – APPEAL\_ACKNOWLEDGED

## HAPPY PATH TEST CASE-2: This TEST CASE send following requests to server:

## *[PROFESSOR] CREATES a Grade:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – APPEAL\_EXPECTED

## *[STUDENT] ADD an Appeal:* Student now updates the Appeal Comment. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] UPDATES already created Appeal:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] SUBMITS the created Appeal:* Student submits the created appeal. Transition to state – PENDING\_REVIEW

## *[PROFESSOR] DENIAL of Appeal:* Professor review the appeal comments and submitted assignment/exam document and found that student is not eligible for marks update, hence just updates his review comment and send the request back to Student for further action. Transition to state – APPEAL\_PENDING\_SUBMISSION

## ABANDON TEST CASE: This TEST CASE send following requests to server:

## *[PROFESSOR] CREATES a Grade:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – APPEAL\_EXPECTED

## *[STUDENT] ADD an Appeal:* Student now updates the Appeal Comment. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] UPDATES already created Appeal:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] ABANDON already created Appeal:* Student post creating the appeal re-evaluate the professor initial comments and corresponding and feels that grades obtained in test is correct and Abandons the appeal. Transition to state – APPEAL\_ABANDONED

## FORGOTTEN TEST CASE-1: This TEST CASE send following requests to server:

## *[PROFESSOR] CREATES a Grade:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – APPEAL\_EXPECTED

## *[STUDENT] ADD an Appeal:* Student now updates the Appeal Comment. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] UPDATES already created Appeal:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] SUBMITS the created Appeal:* Student submits the created appeal. Transition to state – PENDING\_REVIEW

## *[STUDENT] FOLLOW-UP for his Appeal:* Student submitted an appeal and after few weeks, he have not got any response from Professor and hence he follows up on his appeal. Transition to state – APPEAL\_FOLLOWED\_UP

## *[PROFESSOR] UPDATES a Grade:* Professor review the appeal comments and submitted assignment/exam document and found that student is not eligible for marks update, hence just updates his review comment and send the request back to Student for further action. Transition to state – APPEAL\_PENDING\_SUBMISSION

## FORGOTTEN TEST CASE-2: This TEST CASE send following requests to server:

## *[PROFESSOR] CREATES a Grade:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – APPEAL\_EXPECTED

## *[STUDENT] ADD an Appeal:* Student now updates the Appeal Comment. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] UPDATES already created Appeal:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] SUBMITS the created Appeal:* Student submits the created appeal. Transition to state – PENDING\_REVIEW

## *[STUDENT] FOLLOW-UP for his Appeal:* Student submitted an appeal and after few weeks, he have not got any response from Professor and hence he follows up on his appeal. Transition to state – APPEAL\_FOLLOWED\_UP

## *[PROFESSOR] DENIAL of Appeal:* Professor review the appeal comments and submitted assignment/exam document and found that student is not eligible for marks update, hence just updates his review comment and send the request back to Student for further action. Transition to state – APPEAL\_PENDING\_SUBMISSION

## BAD START TEST CASE: This TEST CASE send following requests to server:

## *[PROFESSOR] CREATES a Grade:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – APPEAL\_EXPECTED

## *[STUDENT] ADD an Appeal with BAD URI:* Here Client send a request to server with BAD Uri as: <http://localhost:8080/HATEOAS-Appeals-vgaur1-NetBeans-Server/webapi/gradeAppealResource/createAppeal/9a53bd77-fb29-4040-a0e0-1e357d9b4a3e/bad-uri>

## Server sends a HTTP response as 404

## BAD ID TEST CASE: This TEST CASE send following requests to server:

## *[PROFESSOR] CREATES a Grade:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – APPEAL\_EXPECTED

## *[STUDENT] ADD an Appeal:* Student now updates the Appeal Comment. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] UPDATES already created Appeal:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] SUBMITS the created Appeal:* Student submits the created appeal. Transition to state – PENDING\_REVIEW

## *[STUDENT] FOLLOW-UP with Wrong ID:* Here student send a request with wrong Student ID as: <http://localhost:8080/HATEOAS-Appeals-vgaur1-NetBeans-Server/webapi/gradeAppealResource/followUp/1234>

## Server sends a HTTP response as 404

## BAD STATE TEST CASE: This TEST CASE send following requests to server:

## *[PROFESSOR] CREATES a Grade:* It creates a distinct Student Id and Updates details like Grade Item Name, Grades. Transition to state – APPEAL\_EXPECTED

## *[STUDENT] ADD an Appeal:* Student now updates the Appeal Comment. Transition to state – PENDING\_SUBMISSION

## *[STUDENT] FOLLOW-UP for his Appeal:* Since the current state of resource is not PENDING\_REVIEW (i.e. appeal is not yet submitted), hence, it is not allowed to follow up for an Appeal. Server will not allow such transition of state and sends and HTTP error response code as follows:

## Server sends a HTTP response as 409

## As mentioned above, this is a RMM-Level3 implementation (HATEOAS) and hence this implementation involves multiple URI’s and HTTP verbs. Actions, URI’s and HTTP method mapping is described in diagram below.

## ACTION – URI – HTTP METHOD MAPPING

## 

## HOW TO EXECUTE (BUILD and RUN)

## Following steps need to be followed for building the project and executing it:

## Unzip the shared zipped folder to any user location (directory).

## Import both Client and server project in NetBeans:

## FILE->Import Projects->From Zip File

## (Zip File Name: HATEOAS-Appeals-vgaur1-NetBeans-Client.zip)

## (Zip File Name: HATEOAS-Appeals-vgaur1-NetBeans-Server.zip)

## 

## \*\* At this point, both Client and Server projects are imported in NetBeans.

## START REST SERVER: Following steps should be followed:

## Right click on the Server (HATEOAS-Appeals-vgaur1-NetBeans-Server) and Select RUN.

## Select SERVER as GlassFish Server 4.1.1 & Remember in Current IDE Session.

## Click OK. Following screen should open on system’s Default browser:

## 

## START CLIENT: Similar to server following steps should be followed:

## Right click on the Client (HATEOAS-Appeals-vgaur1-NetBeans-Client) and Select RUN.

## If system prompt for selecting Main Class, Please select ClientTestCaller

## Once client is executed system will generate following output on console:

## 

## \*Note (For step-2): I am sharing unzipped folders both Server and Client for your reference. In case shared Zipped files are corrupted for some reason (& doesn’t get loaded in NetBeans). For loading folders directly instead of zip files, use following option: FILE 🡪 OPEN PROJECT 🡪 *{Select both Server/Client Folder}*

## IMPORTANT POINT:

## Web-Service is built to listen on localhost/8080 PORT. IF PORT NUMBER is changed in CLIENT/SERVER, program would not execute as expected and in that scenario, Port number need to be changed to 8080.

## MINIMUM SYSTEM REQUIREMENTS [PART III]

* NetBeans IDE (8.1)
* GlassFish Server (4.1.1)
* Window 7 (32/64 Bit)

## FEEDBACK & UPDATES

For any updates and feedback, please contact:

Varun Gaur (ASU ID: **1210414176**)

M: +1 408 480 2742