2019

YouTestMe

GetCertified API Documentation

youtestme

Table of Contents

1 Introduction 3
2 User Service 3
2.1 listUsers 3
2.2 getUser(userId) 4
2.3 insertUser 4
2.4 updateUser(userId) 4
2.5 deleteUser(userId) 5
3 Retrieving test results 5
3.1 listTestResults 5
3.2 getTestResultForUser(testId, userId) 7
3.3 listAnswers(testId, userId) 7
4 Creating a question 7
4.1 listQuestions 7
4.2 getQuestion(questionId) 8
4.3 insertQuestion 8
4.4 updateQuestion 9
4.5 deleteQuestion(questionId) 10
4.6 listPools 10
4.7 getPool(poolId) 10
4.8 insertPool 10
4.9 updatePool 10
4.10 deletePool(poolId) 10
4.11 listPoolQuestions 10
4.12 getPoolQuestion(poolId, questionId) 10
4.13 insertPoolQuestion 11
4.14 deletePoolQuestion(poolId, questionId) 11
5 Assigning a test to a user 11
5.1 listQuizzes 11
5.2 getQuiz(quizDefinitionId) 11
5.3 listUniqueQuizzes 11

5.4 getUniqueQuiz(uniqueQuizId) 11
5.5 listQuizInstances 12
5.6 insertQuizInstance 12
6 Importing answers 13
6.1 importAnswersSelected 13
6.2 importAnswersAll 14

1 Introduction

This document contains the information regarding YouTestMe GetCertified REST API. In order to access the API, you need to have an admin account. The authentication that is used in the API is a combination of Basic Auth and JWT. Username and password should be sent to **/auth/login** using Basic Auth and HTTP GET Request in order to obtain the token (that the response consists of). After that, every request (except those that are sent to /auth/login) should contain that token in the headers, the Header should be **Authorization** and the Value should be **Bearer {token}**, while {token} should be the actual token. **Every request sent to the API should be sent over HTTPS.**
Every POST or PUT request must contain the header **Content-Type** whose value is **application/json**.

2 User Service

2.1 listUsers

This functionality provides getting the list of all users and getting user by username, in GetCertified with their basic information (userId, username, firstName, lastName, email, userRole, userStatus), by using HTTP GET request. The format of the response is JSON.

Provided	Uri	Comment
List of all users	/users	
User by username	/users?username=XXXXX	Uses query parameter, an actual username needs to be appended to the Uri

2.2 getUser(userId)

This functionality provides getting the user for the given userId, and an error if there is no user with that userId, by using HTTP GET request. The format of the response is JSON.

Provided	Uri	Comment
User by userId	/users/{userId}	The type of userId is BigDecimal

2.3 insertUser

This functionality provides the creation of a new user, while giving the attributes like email, firstName, lastName, username, userRole and userStatus (all String type) in JSON format, by using HTTP POST request. If any of the attributes doesn't contain a value, there will be an error. The userRole attribute should contain value ATT, ADM or INS, if that user represents student, administrator or instructor, respectively. The userStatus attribute should contain value ACT, DEL or PEN, if the user's status is active, deleted or pending, respectively. If userStatus is not specified, the status ACT will be assigned to the user. The response is the content of the new user along with HTTP Status 201 - Created.

Provided	Uri	Comment
User creation	/users	

2.4 updateUser(userId)

This functionality provides the update of the attributes like email, username and userRole, by giving them in JSON format in HTTP PUT request. You can give only those attributes you want to update in your request. The response is the new content of the user, along with HTTP Status 200 - OK. If the user with that userId does not exist, an exception is thrown.

Provided	Uri	Comment

User update	/users/{userId}	Example (if you want to change userRole of some user to ADM (admin), request in JSON format should look like this):
		{
		"userRole": "ADM"
		}

2.5 deleteUser(userId)

This functionality provides the deletion of a particular user, by using HTTP DELETE request. The response contains HTTP Status 204 - No Content. If the user with that userId does not exist, an exception is thrown.

Provided	Uri	Comment
User deletion	/users/{userId}	

3 Retrieving test results

3.1 listTestResults

This functionality provides listing the results of all tests specified by query parameters, and if parameters are not specified, listing the results of all tests. The basic Uri for this functionality (without query parameters) is **/testresults.** HTTP GET Request should be used with this functionality.

There are following query parameters:

Query parameters	Type	Comment
username	String	
userId, testId, courseId	BigDecimal	**NOTE**: CourseId represents Id of the group user belongs to. Needs renaming in one of the next releases. The rule can be applied to the whole document.
start, size	int	
After	Timestamp	Example: How to forward Timestamp after=2018-14-01 15:00:00

In the table below is specified what certain combinations of query parameters provide:

Provided	Query parameters
List of the results of all tests	/
List of the results of all tests taken by a particular user, specified by userId	userId
List of the results of all tests taken by a particular user, specified by username	username
List of the results of the particular test, specified by testId	testId
List of the results of all tests in the particular group, specified by courseId	courseId
List of the results of all tests taken after a particular moment in time, specified by parameter after	after
List of the results of all tests taken before the particular moment in time, specified by parameter before	before
List of the results of all tests paginated, specified by start and size	start, size
List of the results of all tests taken between moments specified by after and before	after, before
List of the results of the particular test, specified by testId, taken after a particular moment in time, specified by parameter after	testId, after
List of the results of the particular test, specified by testId, taken before the particular moment in time, specified by parameter before	testId, before
List of the results of all tests in the particular group, specified by courseId, taken after a particular moment in time, specified by parameter after	courseId, after
List of the results of all tests in the particular group, specified by courseId, taken before the particular moment in time, specified by parameter before	courseId, before
List of the results of the particular test, specified by testId, taken between moments specified by parameters after and before	testId, after, before
List of the results of all tests in the particular group, specified by courseId, taken between moments specified by parameters after and before	courseId, after, before

3.2 getTestResultForUser(testId, userId)

This functionality provides getting the result of a particular user on a particular test, specified by userId and testId, respectively. If there is more than one attempt of that user on that test, the result of the most recent attempt is returned. The type of testId and userId is BigDecimal. Also, there is a link included in the response for all answers by that user on that test.
HTTP GET Request should be used with this functionality.
If the user with those testId and userId does not exist, an exception is thrown.

Provided	Uri
Result of a particular user on a particular test	/testresults/{testId}_{userId}

3.3 listAnswers(testId, userId)

This functionality provides listing all answers of a particular test taken by a particular user. If there is more than one attempt of that user on that test, answers returned are those from the most recent attempt.
HTTP GET Request should be used with this functionality.

Provided	Uri
All answers of a particular user on a particular test	/testresults/{testId}_{userId}/answers

4 Creating a question

4.1 listQuestions

This functionality provides listing all questions, or all questions of a particular type, specified by query parameter **type** (whose type is String). In this case, HTTP GET Request is used.

Provided	Query parameters	Uri
List of all questions	/	/questions
List of all questions of particular type	type	/questions?type=XXXXX

There are the following types of question:

- SNC - Single Choice Question
- MLC - Multiple Choice Question
- TFC - True False Question
- ESY - Essay Question
- ORD - Ordering Question
- FBL - Fill in the Blanks Question
- MCH - Matching Question

Example Uri for listing all questions of the Single Choice Question type: /questions?type=SNC

4.2 getQuestion(questionId)

This functionality provides getting a single question specified by questionId, whose type is BigDecimal. Uri for this functionality is /questions/{questionId} , and the request used in this case is HTTP GET Request. If the requested question does not exist, an exception is thrown.

4.3 insertQuestion

This functionality provides the creation of a new question, along with its answers, while giving the attributes like questionText (the only necessary attribute, String type), questionType(String type), points, penalty, duration, difficulty (all BigDecimal type), difficultyCode (String type), and , of course, a list of answers, in JSON format, by using HTTP POST request. In order to give one answer, you need to give its attributes: answerText (the only necessary attribute for the answer, String type), answerCorrect (String type), answerWeight and ordinalNumber (both BigDecimal type). The response is the content of the new question, along with HTTP Status 201 - Created.

Provided	Uri	Comment
Question creation	/questions	HTTP POST Request should be used

List of attributes of the question, and their types:

Attribute	Type	Comment
questionText	String	The only necessary attribute
questionType	String	Possible values mentioned in sub-paragraph *listQuestions*
points	BigDecimal	
penalty	BigDecimal	
duration	BigDecimal	In seconds
difficulty	BigDecimal	Bigger number - harder question
difficultyCode	String	DHA - hard DME - medium DEA - easy
answers		

List of attributes of the answer, and their types:

Attribute	Type	Comment
answerText	String	The only necessary attribute of the answer
answerCorrect	String	Possible values: Y if the answer is correct, N if the answer is incorrect; If the type of question is ordering question, the N value should be given for every answer
answerWeight	BigDecimal	If the survey is the type of quiz
ordinalNumber	BigDecimal	If the type of the question is either matching or ordering question

Example:

```
{
  "answers": [
    {
      "answerCorrect": "N",
      "answerText": "4"
    },
    {
      "answerCorrect": "Y",
      "answerText": "5"
    },
    {
      "answerCorrect": "N",
      "answerText": "6"
    }
  ],
  "questionText": "How many NBA Championships Kobe Bryant won?",
  "questionType": "SCH"
}
```

4.4 updateQuestion

This functionality provides the update of some question attributes: questionText, penalty, points or duration, by sending them in JSON format with HTTP PUT Request. The request should be sent to **/questions/{questionId}**. The response is the new content of the question, along with HTTP Status 200 - OK.
If the question with that questionId does not exist, an exception is thrown.

4.5 deleteQuestion(questionId)

This functionality provides the deletion of a particular question, by using HTTP DELETE request. The request should be sent to **/questions/{questionId}**. The response contains HTTP Status 204 - No Content. If the question with that questionId does not exist, an exception is thrown.

4.6 listPools

This functionality provides listing all pools with their basic information (poolId, parentPoolId, poolName, poolDescription, poolType), by using HTTP GET Request. The request should be sent to **/pools**.

4.7 getPool(poolId)

This functionality provides getting a single pool, specified by poolId, by using HTTP GET Request, which should be sent to **/pools/{poolId}**. If the pool with that poolId does not exist, an exception is thrown.

4.8 insertPool

This functionality provides the creation of a new pool, while giving the attributes like parentPoolId (BigDecimal type, necessary), poolName, poolDescription and poolType (all String type) in JSON format, by using HTTP POST request. The request should be sent to **/pools**. The response is the content of the new pool, along with HTTP Status 201 - Created.

4.9 updatePool

This functionality provides the update of some pool attributes: poolDescription or poolName, by sending them in JSON format with HTTP PUT Request. The request should be sent to **/pools/{poolId}**. The response is the new content of that pool, along with HTTP Status 200 - OK. If the pool with that poolId does not exist, an exception is thrown.

4.10 deletePool(poolId)

This functionality provides the deletion of a particular pool, by using HTTP DELETE request. The request should be sent to **/pools/{poolId}**. The response contains HTTP Status 204 - No Content. If the pool with that poolId does not exist, an exception is thrown.

4.11 listPoolQuestions

This functionality provides listing all pool questions with their information (poolId, questionId), specified by the query parameter: by either poolId or questionId, and if not specified, listing all pool questions, by using HTTP GET Request. The basic Uri (without query parameters) is **/poolquestions**.

4.12 getPoolQuestion(poolId, questionId)

This functionality provides getting a single pool question, specified by poolId and questionId, by using HTTP GET Request, which should be sent to **/poolquestions/{poolId}_{questionId}**. If the pool question with that poolId and that questionId does not exist, an exception is thrown.

4.13 insertPoolQuestion

This functionality provides the creation of a new pool question (the addition of a particular question to a particular pool) while giving poolId and questionId (both necessary, and both BigDecimal) in JSON format by using HTTP POST Request. The request should be sent to **/poolquestions**. The response is the content of the new pool question, along with HTTP Status 201 - Created.

4.14 deletePoolQuestion(poolId, questionId)

This functionality provides the deletion of a particular pool question, by using HTTP DELETE request. The request should be sent to **/poolquestions/{poolId}_{questionId}**. The response contains HTTP Status 204 - No Content. If the pool question with that poolId and that questionId does not exist, an exception is thrown.

5 Assigning a test to a user

5.1 listQuizzes

This functionality provides listing all quizzes with their basic information (quizDefinitionId, quizName), by using HTTP GET Request. The request should be sent to **/quizzes**.

5.2 getQuiz(quizDefinitionId)

This functionality provides getting a single quiz, specified by quizDefinitionId, by using HTTP GET Request, which should be sent to **/quizzes/{quizDefinitionId}**. If the quiz with that quizDefinitionId does not exist, an exception is thrown.

5.3 listUniqueQuizzes

This functionality provides listing all unique quizzes with their basic information (uniqueQuizId, quizDefinitionId, uniqueQuizName, quizDefinitionName), by using HTTP GET Request. The request should be sent to **/tests**.

5.4 getUniqueQuiz(uniqueQuizId)

This functionality provides getting a single unique quiz, specified by uniqueQuizId, by using HTTP GET Request, which should be sent to **/tests/{uniqueQuizId}**. If the quiz with that uniqueQuizId does not exist, an exception is thrown.

5.5 listQuizInstances

This functionality provides listing all quiz instances, with their basic information (quizInstanceId, uniqueQuizId, userIdTested, quizInstanceStatus, enabledFrom, enabledTo, duration) specified by query parameter **quiz** (which represents uniqueQuizId) by using HTTP GET Request. If the query parameter is not specified, all quiz instances are returned. The request should be sent to **/quizinstances**.

Provided	Uri	Query parameter	Type of the query parameter
All quiz instances	/quizinstances	/	/
All quiz instances specified by quiz parameter (quiz represents uniqueQuizId)	/quizinstances?quiz=XXXX	quiz	BigDecimal

5.6 insertQuizInstance

This functionality provides the creation of a new quiz instance, which, actually, assigns a test to a user. The attributes like uniqueQuizId, userIdTested, quizInstanceStatus, enabledFrom, enabledTo, duration, courseId and classId should be given in JSON format, by using HTTP POST request. The request should be sent to **/quizinstances**. The response is the content of the new quiz instance, along with HTTP Status 201 - Created. There are following types of quizInstanceStatus: ATR (attempted with report), ATN (attempted without report), NAT (not attempted), QSS (suspended)

Attribute	Attribute type	Necessary/ Not necessary	Comment
uniqueQuizId	BigDecimal	Necessary	
userIdTested	BigDecimal	Necessary	
quizInstanceStatus	String	Not necessary	If not specified, quizInstanceStatus will be NAT
enabledFrom	Timestamp	Not necessary	If not specified, enabledFrom will be the same as the test's
enabledTo	Timestamp	Not necessary	If not specified, enabledTo will be the same as the test's
duration	BigDecimal	Not necessary	If not specified, duration will be the same as the test's
courseId	BigDecimal	Not necessary	CourseId represents user's group.
classId	BigDecimal	Not necessary	Not used anymore

6 Importing answers

6.1 importAnswersSelected

This functionality provides importing answers for a selected user on a selected test (which contains only single choice questions), by using HTTP POST Request, while sending quizInstanceId(BigDecimal) and answers in JSON format to **/importanswersselected**.

answers represent a list of objects that contain the following attributes:
1) answerIdSelected
2) quizResult;

quizResult(necessary) contains:
1) questionId (necessary, BigDecimal, represents an Id of the question on the test that should be answered)
2) notSureFlag(optional, String, “Y” for selected);

answerIdSelected(necessary, BigDecimal, represents an Id of the answer that is selected for the question that is specified by questionId).

All questions that belong to the test specified with quizInstanceId should be answered with an answer that belongs to that particular question. If not, all questions are answered or a questionId that does not belong to that test is sent, an exception will be thrown; also, if answerId that does not belong to that question is sent, an exception will be thrown.

Example of the body of the valid request:

```
{
  "answers": [
    {
      "quizResult": {
        "questionId": 14340
      },
      "answerIdSelected": 40520
    },
    {
      "quizResult": {
        "questionId": 14341
      },
      "answerIdSelected": 40525
    }
  ],
  "quizInstanceId": 4000
}
```

The response of the previous request:


```
{
    "answers": [
        {
            "answerIdSelected": 40520,
            "answersIds": [
                40520,
                40521,
                40522
            ],
            "quizResult": {
                "pointsAssigned": 150,
                "questionId": 14340,
                "quizInstanceId": 4000,
                "quizResultId": 37602
            }
        },
        {
            "answerIdSelected": 40525,
            "answersIds": [
                40524,
                40525,
                40523
            ],
            "quizResult": {
                "pointsAssigned": -30,
                "questionId": 14341,
                "quizInstanceId": 4000,
                "quizResultId": 37603
            }
        }
    ],
    "quizInstanceId": 4000
}
```


The response contains HTTP Status 201 - Created, as well as points assigned to each question on the test, based on the answer that is selected or notSureFlag, and lists of answers that belong to each question, while each answer is represented with its answerId.

6.2 importAnswersAll

This functionality provides importing answers for a selected user on a selected test (which contains single or multiple choice questions), by using HTTP POST Request, while sending quizInstanceId(BigDecimal) and answers in JSON format to **/importanswersall**.

answers represent a list of objects that contain the following attributes:
1) quizResult,
2) quizResultAnswers;

quizResult(necessary) contains:
1) questionId (necessary, BigDecimal, represents an Id of the question on the test that should be answered),
2) notSureFlag(optional, String, “Y” for selected);

quizResultAnswers(necessary) represents a list of objects that contain the following necessary attributes:
1) answerId(BigDecimal, represents answer),
2) answerShown(String; if the answer is shown on the test, should be “Y”, otherwise, should be “N”),
3) answeredIndicator(String, if the answer represented with answerId is selected, should be “Y”, and if not selected, should be “N”).

All questions that belong to the test specified with quizInstanceId should be answered with all answers that belong to that particular question. If not all questions are answered or a questionId that does not belong to that test is sent, an exception will be thrown; also, if answerId that does not belong to that question is sent, an exception will be thrown.

Example of the body of the valid request:


```
{
  "answers": [
    {
      "quizResult": {
        "questionId": 14340
      },
      "quizResultAnswers": [
        {
          "answeredIndicator": "N",
          "answerId": 40520,
          "answerShown": "Y"
        },
        {
          "answeredIndicator": "N",
          "answerId": 40521,
          "answerShown": "Y"
        },
        {
          "answeredIndicator": "Y",
          "answerId": 40522,
          "answerShown": "Y"
        }
      ]
    },
    {
      "quizResult": {
        "notSureFlag": "Y",
        "questionId": 14341
      },
      "quizResultAnswers": [
        {
          "answeredIndicator": "N",
          "answerId": 40523,
          "answerShown": "Y"
        },
        {
          "answeredIndicator": "N",
          "answerId": 40524,
          "answerShown": "Y"
        },
        {
          "answeredIndicator": "N",
          "answerId": 40525,
          "answerShown": "Y"
        }
      ]
    }
  ],
  "quizInstanceId": 4001
}
```

The response of the previous request:


```
{
    "answers": [
        {
            "quizResult": {
                "pointsAssigned": -30,
                "questionId": 14340,
                "quizInstanceId": 4001,
                "quizResultId": 37604
            },
            "quizResultAnswers": [
                {
                    "answerId": 40520,
                    "answerShown": "Y",
                    "answeredIndicator": "N",
                    "quizResultId": 37604
                },
                {
                    "answerId": 40521,
                    "answerShown": "Y",
                    "answeredIndicator": "N",
                    "quizResultId": 37604
                },
                {
                    "answerId": 40522,
                    "answerShown": "Y",
                    "answeredIndicator": "Y",
                    "quizResultId": 37604
                }
            ]
        },
        {
            "quizResult": {
                "notSureFlag": "Y",
                "pointsAssigned": 0,
                "questionId": 14341,
                "quizInstanceId": 4001,
                "quizResultId": 37605
            },
            "quizResultAnswers": [
                {
                    "answerId": 40523,
                    "answerShown": "Y",
                    "answeredIndicator": "N",
                    "quizResultId": 37605
                },
                {
                    "answerId": 40524,
                    "answerShown": "Y",
                    "answeredIndicator": "N",
                    "quizResultId": 37605
                },
                {
                    "answerId": 40525,
                    "answerShown": "Y",
                    "answeredIndicator": "N",
                    "quizResultId": 37605
                }
            ]
        }
    ],
    "quizInstanceId": 4001
}
```


The response contains HTTP Status 201 - Created, as well as points assigned to each question on the test, based on the answeredIndicator for each answer or notSureFlag for the question.