QuestMetrics API Documentation

Overview and Startup

Developed by the QUESTMetrics CMSC435 team in Fall 2020.

Note: We've done our best to test the API, and at one point had functional unit tests done with pytest, but circumstances during the semester have prevented us from keeping those up to date. The API is fairly robust, but particularly recently developed resources and last minute changes (especially the survey) are subject to bugs and/or unnecessary lines. More on that in the Notes on Testing section. Also, despite our best efforts, conventions are a bit loose in return values for POST and PUT requests.

Points of contact:
William Schulmeister (general): wschulme@gmail.com
Ellen Pearson (weights and metrics): nellietp@gmail.com
Jessica Gomez (weights and metrics) jegomez1@terpmail.umd.edu

General Use:
While the VM for CMSC435 is up and functional, the api is reachable with the root http://valerian.cs.umd.edu:5000/

In the case that the VM is down, developers will be able to host the API locally by running the *wsgi.py* file or using the *startServer.sh* script provided. In either case, the root will be http://localhost:5000/

General Notes:

- For sake of brevity, all Query Params that are required are underlined in their respective section. If a call fails to supply these required parameters, it will fail with a 400 status code.
- If the endpoint's url contains any portion of the format: <param>, that is to be replaced by a valid parameter.
- The design of people in the database implies the people database persists and is only deleted in a complete wipe of all data.

Security

Most endpoints in the API require authorization in the form of a JWT token in appended to the header in the form:

Authorization: Bearer <token>

For now, there are 3 roles a given user can have:

1. **Admin:** a user has their directory ID listed explicitly in the admin table of the database and has privileges to all API endpoints, including the ones designated as student.
2. **QUEST student (student):** a user is not listed in the admin table, but is a student stored in the database. Their privileges are limited.
3. **Other:** Users that do not satisfy either of the above conditions will not have access to the API's endpoints.

Users cannot request auth without access to the JWT secret key:

katcBDCk0UhDDUjvfQxoaZNxFAwNZLZl

Additionally, the API uses flask's CORS library and protects against invalid SQL injections.

API

Authorization: *auth.py*

AuthResource: `/auth`

GET
Request a JWT using the JWT secret key and a directory ID. Upon success, endpoint returns a token with an identity object containing the associated directory ID and privilege.

AUTH	QUERY PARAMS	BODY	RETURN
n/a	• key: <string> • directoryId: <string>	n/a	200: success 401: - queried key does not match canonical secret - directoryId not associated with a student or admin

Returns:
{
“access_token”: <string>
}

Class Related Endpoints: *classes.py*

ClassResource: `/classes`

GET
Gets the names of all classes in the database.

AUTH	QUERY PARAMS	BODY	RETURN
student	n/a	n/a	200: success

Returns:
[{ “className”: <string> }, ...]

POST
Posts a new class to the database, which also creates the associated survey slot for surveys administered to that class.

AUTH	QUERY PARAMS	BODY	RETURN
admin	• className: <string>	n/a	200: success 400: - If the class with name className already

			exists - If there is a body to the request

Returns:
{ “classId”: <integer>, “className”: <string> }

DELETE
Deletes the queried class from the database and removes its associated survey slot, survey history, groups and all student-group associations.

AUTH	QUERY PARAMS	BODY	RETURN
admin	• className: <string>	n/a	204: success 400: - If there is a body to the request 404: - If no class could be found

Returns: Empty response

ClassesByDirectoryIdResource: `/classes/student/directoryId/<directoryId>`

GET
Gets the names of all classes in the database that the queried

AUTH	QUERY PARAMS	BODY	RETURN
student	• className: <string>	n/a	200: success 404: - If no person with the given directoryId was found - If the person with the given directoryId is not a student

Returns:
[{ “className”: <string> }, ...]

Student Related Endpoints: *students.py*

StudentResource: `/students`

GET
Gets all students from the database.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success

Returns:
[{ “studentId”: <integer>, “firstName”: <string>, “lastName”: <string>, “directoryId”: <string>, “uid”: <integer> }, ...]

POST
Posts a new student to the database, which also creates their respective entry in the people table.

AUTH	QUERY PARAMS	BODY	RETURN
admin	• firstName: <string> • lastName: <string> • directoryId: <string> • uid: <integer>	n/a	200: success 400: - If there is a body to the request - If directoryId is not unique - If UID is given and not unique - If UID is not numeric

Returns:
{ “studentId”: <integer>, “firstName”: <string>, “lastName”: <string>, “directoryId”: <string>, “uid”: <integer> }

StudentByUIDResource: `/student/uid/<uid>`

GET
Gets the student in the database associated with the given UID.

AUTH	QUERY PARAMS	BODY	RETURN

admin	n/a	n/a	200: success 400: - If the UID is not numeric 404: - If the UID is not found

Returns:
{ “studentId”: <integer>, “firstName”: <string>, “lastName”: <string>, “directoryId”: <string>, “uid”: <integer> }

DELETE
Deletes the student with the associated UID from the database, including the entries in the elmsData, survey answers, and all student-group associations.
Note: This does not remove this person and their directoryId from the people table.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	204: success 404: - If no class could be found

Returns: Empty response

StudentByDirectoryIdResource: `/student/directoryId/<directoryId>`

GET
Gets the student in the database associated with the given directoryId.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 404: - If the directoryId is not found - If the directoryId is not associated with a student

Returns:
{ “studentId”: <integer>, “firstName”: <string>, “lastName”: <string>, “directoryId”: <string>, “uid”: <integer> }

POST
Creates a new student in the database based on an existing person.

AUTH	QUERY PARAMS	BODY	RETURN
admin	• uid: <integer>	n/a	200: success 400: - If the UID is not numeric - If the UID is given and not unique - If the associated directoryId is already a student 404: - If the directoryId is not found - If the directoryId is not associated with a student

Returns:
{ “studentId”: <integer>, “firstName”: <string>, “lastName”: <string>, “directoryId”: <string>, “uid”: <integer> }

DELETE
Deletes the student with the associated directory ID from the database, including the entries in the elmsData, survey answers, and all student-group associations.
Note: This does not remove this person and their directoryId from the people table.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	204: success 404: - If the directoryId is not found - If the directoryId is not associated with a student

Returns: Empty response

StudentGroupResource: `/students/group/<name>`

GET
Gets the students in the database associated with the given group name.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 404: - If the group with the given name does not exist

Returns:
[{ “studentId”: <integer>, “firstName”: <string>, “lastName”: <string>, “directoryId”: <string>, “uid”: <integer> }, …]

StudentByUIDGroupResource: `/student/uid/<uid>/group/<name>`

POST
Adds the student with the given UID to the group with the given name.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 400: - If the UID is not numeric - If the student is already in the given group 404: - If the UID is not found - If the group is not found

Returns:
{ “studentId”: <integer>, “groupId”: <integer> }

DELETE
Deletes the student identified by the given UID from the group identified by the given group name.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	204: success

			400: - If the UID is not numeric 404: - If the UID is not found - If the group is not found - If there is no association currently between the given student and group

Returns: Empty response

StudentByDirectoryIdGroupResource:

`/student/directoryId/<directoryId>/group/<name>`

POST

Adds the student with the given directory ID to the group with the given name.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 400: - If the student is already in the given group 404: - If the group is not found - If the directoryId is not found - If the directoryId is not associated with a student - If there is no association currently between the given student and group

Returns:
{ “studentId”: <integer>, “groupId”: <integer> }

DELETE

Deletes the student identified by the given UID from the group identified by the given group name.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 400: - If the student is already in the given group 404: - If the group is not found - If the directoryId is not found - If the directoryId is not associated with a student - If there is no association currently between the given student and group

Returns: Empty response

Groups Related Endpoints: *groups.py*

Groups: `/groups`

GET
Gets all the groups in the database with their corresponding group data.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success

Returns:
[{"groupId": <integer>, "name": <string>, "className": <string>, "watch": <integer>, "groupHealth": <integer>, "groupScore": <float>}, ...]

POST
Posts a new group to the database. Group based data is initialized with group health initized to 3 (Green) and the watch status is 0 (False).

AUTH	QUERY PARAMS	BODY	RETURN

admin	• name: <string> • className: <string> • channel: <string>	n/a	200: success 404: given class does not exist

Returns:
[{"groupId": <integer>, "name": <string>, "className": <string>, "watch": <integer>, "groupHealth": <integer>]

DELETE
Deletes a group from the database along with its group based data and any student - team relationships that included it.

AUTH	QUERY PARAMS	BODY	RETURN
admin	• name: <string>	n/a	204: success 404: given group does not exist

Returns:
Empty response

WatchResource: `/groups/watch/<name>`

GET
Get the binary value for the given groups status of being on the watchlist with the following convention: 0 is not being watched, and 1 is being watched.

AUTH	QUERY PARAMS	BODY	RETURN
admin	N/A	n/a	200: success 404: given group does not exist

Returns:
{'result': 0 or 1}

POST
Toggles value that flags the given group as being on the watchlist.

AUTH	QUERY PARAMS	BODY	RETURN
admin	N/A	n/a	204: success 404: given group does not

			exist

Returns:
Empty response

GroupsWatchedResource: `/groups/watching`

GET
Gets all the groups in the database that are currently being watched.

AUTH	QUERY PARAMS	BODY	RETURN
admin	N/A	n/a	200: success

Returns:
[{ "groupId": <integer>, "name": <string>, "className": <string>, "watch": <integer>, "groupHealth": <integer>, "groupScore": <double>}, …]

GroupsByClassResource: `/groups/class/<className>`

GET
Gets all the information of the groups within a given class.

AUTH	QUERY PARAMS	BODY	RETURN
admin	• className: <string>	n/a	200: success 404: - given class does not exist

Returns:
[{ "groupId": <integer>, "name": <string>, "className": <string>, "watch": <integer>, "groupHealth": <integer>, "groupScore": <double>}, …]

GroupHealthResource: `/group/health/<name>`

GET
Gets the integer value of the queried group's health. It must be either 1 (red), 2 (yellow) or 3 (green).

AUTH	QUERY PARAMS	BODY	RETURN
admin	• name: <string>	n/a	200: success 404: - given group does not exist

Returns:

```
{
   "result": <integer>
}
```

PUT

Puts the given integer value in the corresponding group's groupHealth value. It must be an integer value in between 1 and 3 or it will not be accepted.

AUTH	QUERY PARAMS	BODY	RETURN
admin	• name: <string> • groupHealth: <integer>	n/a	204: success 400: - the value inputted into the group health is smaller than 1 or greater than 3 and therefore invalid 404: - given group does not exist

Returns: Empty response

GroupRedResource: `/groups/red`

GET

Gets all the information of the groups that are determined to be red, which means that their groupHealth value is 1.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success

Returns:
[{ "groupId": <integer>, "name": <string>, "className": <string>, "watch": <integer>, "groupHealth": <integer>, "groupScore": <double>}, ...]

GroupsByStudentUIDResource: `/groups/student/uid/<uid>`

GET
Gets all the groups that a student is in based off of their university ID (a student can be in more than one group at a time).

AUTH	QUERY PARAMS	BODY	RETURN
admin	• uid: <integer>	n/a	200: success 400: - the uid is not a numerical value 404: - the student does not exist

Returns:
[{ "groupId": <integer>, "name": <string>, "className": <string>, "watch": <integer>, "groupHealth": <integer>, "groupScore": <double>}, …]

GroupsByStudentDirectoryIdResource:
`/groups/student/directoryId/<directoryId>`

GET
Gets all the groups that a student is in based off of their directoryID (a student can be in more than one group at a time).

AUTH	QUERY PARAMS	BODY	RETURN
admin	• directoryId: <string>	n/a	200: success 404: - If the directoryId is not found - If the directoryId is not associated with a student

Returns:
[{ "groupId": <integer>, "name": <string>, "className": <string>, "watch": <integer>, "groupHealth": <integer>, "groupScore": <double>}, …]

GroupHealthScoreResource: `/group/score/<name>`

GET
Gets the double floating point value of the queried group's health score. It will be between the values 0.0 and 1.0.

AUTH	QUERY PARAMS	BODY	RETURN
admin	• name: <string>	n/a	200: success 404: - given group does not exist

Returns:
{
"result": <double>
}

PUT
Puts the given double floating point value in the corresponding group's groupHealth value. The value has to be between 0.0 and 1.0.

AUTH	QUERY PARAMS	BODY	RETURN
admin	• name: <string> • groupScore: <double>	n/a	204: success 400: - The value is not between the range of 0.0 and 1.0. 404: - given group does not exist

Returns: Empty response

Metrics Related Endpoints: *metrics.py*

Metrics: `/metrics`

PUT
Updates the groupHealth and groupScore values for each group in all the classes according to the calculated metrics.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success

Returns:
[{ "groupId": <integer>, "name": <string>, "className": <string>, "watch": <integer>, "groupHealth": <integer>, "groupScore": <double>}, ...]

Weights Related Endpoints: *weights.py*

WeightsResource: `/weights`

GET
Gets the current weights of each metric.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success

Returns:
{ "slack": <double>, "grades": <double>, "lateness": <double>, "survey": <double>, "lastView": <double> }

PUT

Updates the weights of each metric according to the different values inputted in the query.

AUTH	QUERY PARAMS	BODY	RETURN
admin	• slack: <double> • grades: <double> • lateness: <double> • survey: <double> • lastView: <double>	n/a	200: success 400: - If the value inputted is not numeric

Returns:
{ "slack": <double>, "grades": <double>, "lateness": <double>, "survey": <double>, "lastView": <double> }

DELETE
Deletes the current weights and assigns the default values: slack = 20, grades = 15, lateness = 17.5, survey = 30 and lastView = 17.5.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	204: success

Returns: Empty response

Survey Related Endpoints: *survey.py*

SurveyResource: `/survey/class/<className>`

GET
Gets all the current survey data from the database.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 404: - If the given class does not exist

Returns:
{ “questions”: [<string>, <string>, ...] }, “results”: [{
“answers”: [<string>, <string>, …],
“firstName”: <string>,
“lastName”: <string>,
“uid”: <integer>,
className”: <string>}
, …] }

POST
Posts a new set of questions to the survey of the given class.

AUTH	**QUERY PARAMS**	**BODY**	**RETURN**
admin	n/a	[{“question” : <string>}, {“question”: <string>}, ...]	204: success 400: - If the body is not a list - If the elements of the list do not include a question key 404: - If the given class does not exist

Returns: Empty response

DELETE
Clears the questions and answers from the survey associated with the given class, adding the results to the survey history.

AUTH	**QUERY PARAMS**	**BODY**	**RETURN**
admin	n/a	n/a	200: success 404: - If the given class does not exist

Returns: Empty response

StudentReponseResource:

`/survey/class/<className>/student/directoryId/<directoryId>`

GET

Gets the most recent responses for the given student in the given class.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 404: - If the directoryId is not found - If the directoryId is not associated with a student - If the class doesn't exist

Returns:
{ "questions": [<string>, <string>, ...] }, "results": {
"answers": [<string>, <string>, …],
"firstName": <string>,
"lastName": <string>,
"uid": <integer>,
className": <string>}
}

PUT

Puts new responses for the given student for the survey associated with the given class.

AUTH	QUERY PARAMS	BODY	RETURN
student	n/a	[{"answer" : <string>}, {"answer": <string>}, ...]	204: success 400: - If the body is not a list - If the elements of the list do not include a question key - If the number of answer objects does not equal the number of questions in the survey 404: - If the directoryId is not found - If the directoryId is not associated with a student

			- If the class doesn't exist

Returns: Empty Response

QuestionResource: `/survey/questions/class/<className>`

GET
Gets the current list of questions of the survey associated by the given className.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 404: - If the class doesn't exist

Returns:
{ ["surveyId": <integer>, "className": <string>, "question": <string>], … }

Survey History: *history.py*

HistoryAllResource: `/survey/history`

GET
Gets all survey history stored in the database.
Note: the timestamp returned is a string cast of a datetime object of the time the survey was harvested and the json in data is an exact mirror of the output format for the get on the SurveyResource.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success

Returns:
[{ "className": <string>, "harvestedOn": <string>, "data": <JSON> }, ...]

DELETE

Clears survey history.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	204: success

Returns: Empty response

HistoryByClassResource: `/survey/history/class/<className>`

GET
Gets all survey history stored in the database associated with the given class.
Note: the timestamp returned is a string cast of a datetime object of the time the survey was harvested and the json in data is an exact mirror of the output format for the get on the SurveyResource.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 404: - If the class doesn't exist

Returns:
[{ "className": <string>, "harvestedOn": <string>, "data": <JSON> }, ...]

DELETE
Clears survey history associated with the given class.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	204: success 404: - If the class doesn't exist

Returns: Empty response

Data Based Resources: *data.py*

ClearResource: `/clear`

DELETE
Deletes all data from all tables that are listed in the tables list found in app.py.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	204: success

Returns: Empty response

GroupBasedDataResource: `/<table>/group/<name>`

GET
Gets the group based data associated with the given group in the associated table.
Note: assumes that the table which is included in groupTablesIgnorePopulate is indexed by groupId, which is a foreign key to the groups table.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 404: - If the table is not included in groupTablesIgnorePopulate - If the group does not exist

Returns: { “result”: “” } if the resulting query returns NONE or a list of objects where each key is the column of the respective table.

StudentBasedDataByUIDResource: `/<table>/student/uid/<uid>/class/<className>`

GET
Gets the student based data associated with the given student in the associated table.
Note: assumes that the table which is included in studentTablesIgnorePopulate is indexed by classId AND studentId, which are foreign keys to their respective table.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 400: - If the given UID is not numeric 404: - If the table is not included in studentTablesIgnorePopulate - If the student does not exist - If the class does not exist - If the student is not in the queried class

Returns: A list of objects where each key is the column of the respective table.

SlackResource: `/slack/group/<groupName>`

GET
Gets all messages from the slackData table.
Note: Although the groupName is specified in the request, the API returns all slack records across all teams and groups. It's then grouped by class and groupName in Apache Superset. A future update would use the groupName feature.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 404: - If the group does not exist

Returns: A list of objects where each key is the column of the respective table.
[{messageId: int, groupId, int, slackMsgId: hash, channel: hash, user: hash, timestamp: timestamp}]

POST
Post new Slack messages or events to the table from a .csv. Can generate this .csv using the Slack.py in the /backend directory module.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	204: success 404: - If the group does not exist

Returns: Empty Response

SlackResource: `/elms`

GET
Gets all data from the elmsData table.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success 404: - If the group does not exist

Returns: A list of objects where each key is the column of the respective table.
[{studentId: int, classId: int, timestamp: timestamp, lastView: string, currentGrade: float, percentLate: float}]

POST
Post new elms metrics to the table from a .csv. Can generate this .csv using the elms.py file in the /backend directory module.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	204: success 404: - If the group does not exist

Returns: Empty Response

StudentBasedDataByDirectoryIdResource:
`/<table>/student/directoryId/<directoryId>/class/<className>`

GET
Gets the student based data associated with the given student in the associated table.
Note: assumes that the table which is included in studentTablesIgnorePopulate is indexed by classId AND studentId, which are foreign keys to their respective table.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	200: success

			404: - If the table is not included in studentTablesIgnorePopulate - If the person does not exist - If a student does not exist with the given directoryId - If the class does not exist - If the student is not in the queried class

Returns: A list of objects where each key is the column of the respective table.

Mock Current Data: *mock.py*

MockData: `/mock`

POST
Modifies existing database to randomly assign all firstNames, lastNames and UIDs with exception to our current team members Collin Draper and Amy Odenthal.

AUTH	QUERY PARAMS	BODY	RETURN
admin	n/a	n/a	204: success

Returns:
[{ “studentId”: <integer>, “firstName”: <string>, “lastName”: <string>, “directoryId”: <string>, “uid”: <integer> }, ...]

Notes on Testing:

The testing right now is pretty severely out of date, but about half of them should work the same, barring minor fixes. The major issues stem from the following changes:

1. We initially had ELMS and Slack data generically defined as blob data in an attempt to easily add a new data source when necessary. However, it soon became apparent that a new approach would be necessary to accommodate the visualization on the front end which needed to poll the database. To accommodate, the endpoints for data were

expanded to take data as a csv and convert into SQL columns (See Data Endpoints). Tests that relate to these endpoints will fail. Reach out to Sam Schoberg for information on this or Will for his best guess.

2. Once it became apparent we had to have surveys individually for each class, we changed the entire survey infrastructure to something much more sensible. This meant doing away with the surveyData table. While these changes were reflected in the code, some tests (particularly those that test the propagating delete procedure) will fail naively because the table no longer exists. Individual survey functionality has not yet been tested meaningfully as it has been only a handful of days since its creation. Reach out to Will for information on this front.
3. Metrics have not been tested. Reach out to Ellie Pearson and Jessica Gomez for notes on that front or Will for his best guess.
4. In general, test files are named based on the resource they are testing, so it should be easy to find the endpoints that haven't been tested.

The good news is that I can guarantee that the API is mostly functional, particularly in GET, POST and PUT requests as those are tested very frequently during regular use of the frontend. We've also standardized error handling in errors.py, so it should be easy to update, throw, and track.

These tests can be run with the command py.test.