GatorSecurity Backend API Documentation

Jacob Boney

Kerry Hannigan

Brian Hoblin

Dylan Tosh

Connor Wojtak

Introduction

This documentation explains the backend API for the GatorSecurity application. It is designed to accompany example Postman requests that can be used to contact the backend API endpoints.

User API Endpoints

The backend stores user data using the UserInfo schema. UserInfo has a string fname, string lname, unique string email, string password, boolean isAdmin (set by default to false), and two arrays — learnscore and gamescore — that consist of a list of TraditionalQuestion and GameQuestion IDs in the database that the user has completed.

POST /users/register

This endpoint registers a new user. Requests should be sent as application/json and should contain a string fname, a string lname, a string email, and a string password. The fname and lname represent a user's first and last name.

If the operation is completed successfully, an "ok" status will be returned. If there is a user that already exists with the given email, an "A GatorSecurity account already exists with this email address." error will be returned. If any other errors occur, an "error" status will be returned.

POST /users/login

This endpoint logs in an existing user. Requests should be sent as application/json and should contain a string email and string password.

If the operation is completed successfully, a status "ok" will be returned along with a data attribute containing a string representing the user's JSON web token (JWT). If the user's

password is incorrect, a status “error” will be returned along with an error of “Invalid password.” If any other errors occur, an error “error” will be returned.

POST /users/userInfo

This endpoint gets a user’s info. Requests should be sent as application/json and should contain a string token representing the JWT of the user whose information should be retrieved.

If the operation is completed successfully, a status “ok” will be returned along with a data attribute containing the user’s data as stored in the UserInfo schema. If the operation fails, either a status “error” with a data field containing an error message will be returned, or a status of 500 will be returned.

PUT /users/update/:id

This endpoint updates a user’s info. Requests should be sent as application/json and should contain at least a string email, a string fname, a string lname, a string email, and a string password. The fname and lname should represent the first and last name of the user. The :id in the request URL should be replaced with the ID of the user that should be updated.

If this operation is completed successfully, a 202 status will be returned. If no user is found from the given ID, a 404 status will be returned. If another user exists in the database with the provided email, a 403 status will be returned. If any errors occur during the operation, a 500 status will be returned.

POST /users/allUsers

This endpoint gets all of the info of all of the users in the database. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true in the database.

If this operation is completed successfully, a data field will be sent containing an array of UserInfo as stored in the database. If a non-admin attempts to access this endpoint, a 403 status will be returned. If any errors occur during the operation, either a status “error” along with a data field containing the error will be returned, or a 500 status will be returned.

PUT /users/updatelearnscore

This endpoint updates a user’s Learn score. Requests should be sent as application/json and should contain a string token that represents a JWT of the user whose score should be updated, a string qid representing an ID for the question that is being answered, and a string answer to the question.

If the operation completes successfully, a 200 status will be sent back along with a data.correct field set to true and a data.qid field set to the question ID. If the user answered the question wrong, the score will not be updated, and a 200 status will be returned along with a data.correct field set to false. If any errors occur during the operation, a 500 status will be returned.

POST /users/checkPrivileges

This endpoint checks a user’s privileges. Requests should be sent as application/json and should contain a string token that represents a JWT of the user whose privileges should be checked.

If the user has admin privileges, a 200 status will be returned. If the user does not have admin privileges, a 403 status will be returned. If an error occurs, a 500 status will be returned.

POST /users/updateScore

This endpoint updates a user's Game score. Requests should be sent as application/json and should contain a string token that represents a JWT of the user whose score should be updated and a string qid representing an ID for the question that is being answered.

If the operation completes successfully, a 204 status will be returned. If any errors occur during the operation, a 500 status will be returned.

Traditional Learn Question API Endpoints

The backend stores traditional learn question data using the TraditionalQuestion schema. TraditionalQuestions have a string question, numeric type, numeric topic, string answer, enum displayType (which must either be 'learn' or 'game'), and an array of string answer options. See Appendix A for representations of the numeric constants.

POST /questions/getcount

This endpoint gets the count of all the TraditionalQuestions in the database. The request should be sent as application/json.

If the operation completes successfully, a 200 status will be returned along with a data field containing the count of the questions. If an error occurs, a 500 status will be returned.

POST /questions/get/:topic

This endpoint gets all TraditionalQuestions of a certain topic. The request should be sent as application/json.The :topic in the request URL should be a number or string representing the topic filter. See Appendix A for representations of type constants.

If the operation completes successfully, a 200 status will be returned along with a data field containing an array of TraditionalQuestions as they appear in the database. If the operation is not successful, a 500 status will be returned.

DELETE /questions/delete/:id

This endpoint deletes a TraditionalQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true in the database. The :id in the request URL should be replaced with the ID of the TraditionalQuestion in the database to delete.

If the operation completes successfully, a 202 status will be returned. If the question was not found from the given ID, a 404 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

PUT /questions/update/:id

This endpoint updates a TraditionalQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true the the database. The following optional fields can be included to update the TraditionalQuestion: a string question representing the question being asked, a numeric type, a numeric topic, an array of string options, a string answer, and an enum displayType. See Appendix A for a representation of type and topic constants. The :id in the request URL should be replaced with the ID of the TraditionalQuestion in the database to update.

If the operation completes successfully, a 202 status will be returned. If the question was not found from the given ID, a 404 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

POST /questions/create

This endpoint creates a TraditionalQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true the the database. The following fields should be included to create the TraditionalQuestion: a string question representing the question being asked, a numeric type, a numeric topic, an array of string options, a string answer, and an enum displayType. See Appendix A for a representation of type and topic constants.

If the operation completes successfully, a 201 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

Game Question API Endpoints – GameQuestion Endpoints

The backend stores game question data using the GameQuestion schema. GameQuestions have a string name, a numeric topic, a numeric type, and then an array of IDs to subquestions that compose the GameQuestion. See Appendix A for representations of the numeric constants.

POST /games/getcount

This endpoint gets the count of all the GameQuestions in the database. The request should be sent as application/json.

If the operation completes successfully, a 200 status will be returned along with a data field containing the count of the questions. If an error occurs, a 500 status will be returned.

POST /games/getByTopic/:topic

This endpoint gets all GameQuestions of a certain topic. The request should be sent as application/json.The :topic in the request URL should be a number or string representing the topic filter. See Appendix A for representations of type constants.

If the operation completes successfully, a 200 status will be returned along with a data field containing an array of GameQuestions as they appear in the database. If the operation is not successful, a 500 status will be returned.

POST /games/getByTypes/:type

This endpoint gets all GameQuestions of a certain type. The request should be sent as application/json.The :type in the request URL should be a number or string representing the type filter. See Appendix A for representations of type constants.

If the operation is completed successfully, a 200 status will be returned along with a data field containing an array of GameQuestions as they appear in the database. If the operation is not successful, a 500 status will be returned.

POST /games/getById/:id

This endpoint gets all GameQuestions given an ID. The request should be sent as application/json.The :id in the request URL should be a number or string representing the ID filter. See Appendix A for representations of type constants.

If the operation is completed successfully, a 200 status will be returned along with a data field containing a GameQuestion as it appears in the database. If the operation is not successful, a 500 status will be returned.

DELETE /games/delete/:id

This endpoint deletes a GameQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true in the database. The :id in the request URL should be replaced with the ID of the GameQuestion in the database to delete.

If the operation completes successfully, a 202 status will be returned. If the question was not found from the given ID, a 404 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

PUT /games/update/:id

This endpoint updates a GameQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true the the database. The following optional fields can be included to update the GameQuestion: a numeric topic and a string name. See Appendix A for a representation of topic constants. The :id in the request URL should be replaced with the ID of the GameQuestion in the database to update.

If the operation is completed successfully, a 202 status will be returned. If the question was not found from the given ID, a 404 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

POST /games/create

This endpoint creates a GameQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true the the database. The following fields should be included to create the GameQuestion: a numeric type, a numeric topic, and a string name for the game. See Appendix A for a representation of type and topic constants.

If the operation is completed successfully, a 201 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

Game Question API Endpoints – CYOAQuestion Endpoints

The backend stores choose your own adventure question data using the CYOAQuestion schema. CYOAQuestions have a mongoose.Schema.Types.ObjectId parentQuestionId, a numeric questionNumber, a string question, a numeric type, an array of string answer options, a string answer, a string stimulus, and a string explanation. See Appendix A for representations of the numeric constants.

POST /games/cyoa/getById/:id

This endpoint gets a CYOAQuestion given an ID. The request should be sent as application/json.The :id in the request URL should be a number or string representing the ID filter. See Appendix A for representations of type constants.

If the operation is completed successfully, a 200 status will be returned along with a data field containing a GameQuestion as it appears in the database. If the operation is not successful, a 500 status will be returned.

DELETE /games/cyoa/delete/:id

This endpoint deletes a CYOAQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true in the database. The :id in the request URL should be replaced with the ID of the CYOAQuestion in the database to delete.

If the operation completes successfully, a 202 status will be returned. If the question was not found from the given ID, a 404 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

PUT /games/cyoa/update/:id

This endpoint updates a CYOAQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as multipart/form-data and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true the database. The following optional fields can be included to update the CYOAQuestion: a numeric question number, a string question, a numeric type, an array of strings options, an answer string, an explanation string, and a stimulus image file. See Appendix A for a representation of topic constants. The :id in the request URL should be replaced with the ID of the CYOAQuestion in the database to update.

If the operation is completed successfully, a 202 status will be returned. If the question was not found from the given ID, a 404 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

POST /games/cyoa/create

This endpoint creates a CYOAQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as multipart/form-data and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true the database. The following fields should be included to create the CYOAQuestion: a numeric question number, a string question, a numeric type, an array of strings options, an answer string, an explanation string, and a stimulus image file. See Appendix A for a representation of topic constants.

If the operation is completed successfully, a 202 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

POST /games/checkAnswer/:id

This endpoint checks the answer to a CYOAQuestion without updating the user's score. This is used for when users are progressing through the subparts of a CYOA game. Requests should be sent as application/json and contain a string answer. The :id in the request URL should be replaced with the ID of the CYOAQuestion that is being answered.

If the operation is completed successfully, an "ok" status will be returned with a data field either containing true or false depending on whether the user answered correctly. If any errors occur, a 401 status will be returned.

Game Question API Endpoints – DNDQuestion Endpoints

The backend stores drag and drop question data using the DNDQuestion schema. DNDQuestions have a mongoose.Schema.Types.ObjectId parentQuestionId, a string question, an array of strings answer representing the draggable elements in the correct order, a string stimulus, and a string explanation.

POST /games/dnd/getById/:id

This endpoint gets a DNDQuestion given an ID. The request should be sent as application/json.The :id in the request URL should be a number or string representing the ID filter. See Appendix A for representations of type constants.

If the operation is completed successfully, a 200 status will be returned along with a data field containing a DNDQuestion as it appears in the database. If the operation is not successful, a 500 status will be returned.

DELETE /games/dnd/delete/:id

This endpoint deletes a DNDQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true in the database. The :id in the request URL should be replaced with the ID of the DNDQuestion in the database to delete.

If the operation completes successfully, a 202 status will be returned. If the question was not found from the given ID, a 404 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

PUT /games/dnd/update/:id

This endpoint updates a DNDQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as multipart/form-data and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true the database. The following optional fields can be included to update the DNDQuestion: a string question, a string explanation, a string answer, and a stimulus image file. The :id in the request URL should be replaced with the ID of the DNDQuestion in the database to update.

If the operation is completed successfully, a 202 status will be returned. If the question was not found from the given ID, a 404 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

POST /games/dnd/create

This endpoint creates a DNDQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as multipart/form-data and should contain a string token

that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true the database. The following fields should be included to create the DNDQuestion: a string question, a string answer, a string explanation, and a stimulus image file.

If the operation is completed successfully, a 202 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

Game Question API Endpoints – MatchingQuestion Endpoints

The backend stores matching question data using the MatchingQuestion schema. MatchingQuestions have a mongoose.Schema.Types.ObjectId parentQuestionId and a content map of strings to strings containing words as the keys and their definitions as the values.

POST /games/matching/getById/:id

This endpoint creates a MatchingQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true the database. The following fields should be included to update the DNDQuestion: a string question, a string answer, a string explanation, and a stimulus image file.

If the operation is completed successfully, a 202 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

POST /games/matching/create

This endpoint creates a MatchingQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their

isAdmin attribute set to true the database. The following fields should be included to create the MatchingQuestion: content, which is a map of strings to strings representing word-definition pairs.

If the operation is completed successfully, a 202 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

PUT /games/matching/update/:id

This endpoint updates a MatchingQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true the database. The following optional fields can be included to update the MatchingQuestion: content, which is a map of strings to strings representing word-definition pairs.

If the operation is completed successfully, a 202 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

DELETE /games/matching/delete/:id

This endpoint deletes a MatchingQuestion. Administrative privileges are required to access this endpoint. Requests should be sent as application/json and should contain a string token that represents a JWT of the user performing the operation. This user must have their isAdmin attribute set to true in the database. The :id in the request URL should be replaced with the ID of the MatchingQuestion in the database to delete.

If the operation completes successfully, a 202 status will be returned. If the question was not found from the given ID, a 404 status will be returned. If the user is not an admin, a 403 status will be returned. Finally, if there were any errors, a 500 status will be returned.

Appendix A – Representations of Application Constants

Traditional Question Types

1 = Typed Answer

2 = T/F

3 = MC

Gamified Question Types

0 = CYOA

1 = DND

2 = MM

Topics

1 = input_validation = Input Validation

2 = encoding_escaping = Encoding & Escaping

3 = xss = Cross-Site Scripting

4 = sql_injection = SQL Injection

5 = crypto = Cryptography

6 = auth = User Authentication

7 = URL SQL Injection

8 = Login SQL Injection

9 = Buffer Overflow

10 = Cybercrime Laws

11 = Malware Disassembly

12 = Encryption

13 = Networking

14 = Password Security

15 = Standard Cybersecurity Terms

16 = Principles of Secure Design

17 = Attack Trees

18 = Adversarial Thinking