TheALLDictionary:

API Design Document

Spring 2021 CSE 416 Final project

Team Members:

Young-jae Moon → Product owner	
Sub-team A	Sub-team B
Young-jae Moon → Project manager youngjae.moon@stonybrook.edu Yoo-ra Kim → Lead programmer yoora.kim@stonybrook.edu	Seung-jae Kang → Designer seungjae.kang@stonybrook.edu Seo-young Ko → Designer seoyoung.ko@stonybrook.edu

REST API tables for creating accounts

Method	POST
Name	addUser
URL	users/add
Input	fullName : String nickName : String userID : String password : String email : String phoneNumber : String

Query	INSERT INTO Account (userID, isAdmin) VALUES (userID, 0);
	
	INSERT INTO UserAccount (fullName, nickName, userID, password, email, phoneNumber, isPremium, subscriptionEndDate) VALUES (fullname, nickName, userID, password, email, phoneNumber, 0, NULL);

Output	{
	“error”: false,
	“message”: “Your user account has been created successfully.”
	}

Notes	Security questions and their corresponding answers have been intentionally omitted. This is because we do not need to build this project as super secure. The front-end part should validate the password that the user has inputted before creating a user account.

Method	POST
Name	addAdmin
URL	admins/add
Input	fullName : String nickName : String userID : String password : String email : String phoneNumber : String

	jobTitle : String

Query	INSERT INTO Account (userID, isAdmin) VALUES (userID, 1);
	
	INSERT INTO AdminAccount (fullName, nickName, userID, password, email, phoneNumber, jobTitle) VALUES (fullname, nickName, userID, password, email, phoneNumber, jobTitle);

Output	{
	    “error": false,
	    “message”: “Your admin account has been created successfully.”
	}

Notes	Security questions and their corresponding answers have been intentionally omitted. This is because we do not need to build this project as super secure. The front-end part should validate the password that the admin has inputted.

REST API tables for deleting accounts

Method	DELETE
Name	deleteUser
URL	users/<string:userID>/delete
Input	userID : String password : String
Query	DELETE FROM UserAccount WHERE userID = userID;

Output	{
	"error": false,
	"message": "Your user account has been deleted successfully"
	}

Notes	The user shall confirm its password before deleting the account. Since the userID is a PK of UserAccount, we just need to find the attribute in which the userID is the same as the userID of the logged in user. The userID attribute in the UserAccount table is a FK. Thus, its delete setting is 'CASCADE.' Hence the corresponding record in the Account table will be deleted simultaneously when the record is deleted from the UserAccount table.

Method	DELETE
Name	deleteAdmin
URL	admins/<string:userID>/delete
Input	userID : String password : String
Query	DELETE FROM AdminAccount WHERE userID = userID;

Output	{
	"error": false,
	"message": "Your admin account has been deleted successfully."
	}

Notes	The admin shall confirm its password before deleting the account.

	Since the userID is a PK of AdminAccount, we just need to find the attribute in which the userID is the same as the userID of the logged in admin. The userID attribute in the AdminAccount table is a FK. Thus, its delete setting is ‘CASCADE.’ Hence the corresponding record in the Account table will be deleted simultaneously when the record is deleted from the AdminAccount table.

<u>REST API tables for getting personal data</u>

Method	GET
Name	getUserData
URL	users/<string:userID>/get
Input	None
Query	SELECT * FROM UserAccount WHERE userID = userID;
Output	{ “fullName”: String, “nickName”: String, “userID”: String, “password”: String, “email”: String, “phoneNumber”: String, “isPremium”: Boolean, “subscriptionEndDate”: Date }
Notes	

Method	GET
Name	getAdminData
URL	admins/<string:userID>/get
Input	None
Query	SELECT * FROM AdminAccount WHERE userID = userID;
Output	{ “fullName”: String, “nickName”: String, “userID”: String, “password”: String, “email”: String, “phoneNumber”: String, “jobTitle”: String }
Notes	

REST API tables for updating personal data

Method	PATCH
Name	updateUserNickName
URL	users/<string:userID>/patch/nicknames
Input	newNickName: String
Query	UPDATE UserAccount SET nickName = newNickName WHERE userID = userID;
Output	{ “error” : false “message”: “Your nickname has been updated successfully.” }
Notes	The nickname can be deleted by setting the attribute as NULL in the table.

Method	PATCH
Name	updateAdminNickName
URL	admins/<string:userID>/patch/nicknames
Input	newNickName: String
Query	UPDATE AdminAccount SET nickName = newNickName WHERE userID = userID;
Output	{ “error” : false “message”: “Your nickname has been updated successfully.” }
Notes	The nickname can be deleted by setting the attribute as NULL in the table.

Method	PATCH
Name	updateUserPassword
URL	users/<string:userID>/patch/passwords
Input	newPassword: String

Query	UPDATE UserAccount SET password = newPassword WHERE userID = userID;

Output	{
	“error” : false
	“message”: “Your password has been updated successfully.”
	}

Notes	The new password shall contain at least nine letters, including at least one capital letter, one small letter, one number, and one special character. This requirement will be checked in the front-end part. The new password shall be confirmed before updating in the database. The confirmation will be done in the front-end part. The new password will be encrypted and then saved in the database using WerkZeug.

Method	PATCH
Name	updateAdminPassword
URL	admins/<string:userID>/patch/passwords
Input	newPassword: String
Query	UPDATE AdminAccount SET password = newPassword WHERE userID = userID;

Output	{
	“error” : false
	“message”: “Your password has been updated successfully.”
	}

Reason	The new password shall contain at least nine letters, including at least one capital letter, one small letter, one number, and one special character. This requirement will be checked in the front-end part. The new password shall be confirmed before updating in the database. The confirmation will be done in the front-end part. The new password will be encrypted and then saved in the database using WerkZeug.

Method	PATCH
Name	updateUserEmail
URL	users/<string:userID>/patch/emails
Input	newEmail: String
Query	UPDATE UserAccount SET email = newEmail WHERE userID = userID;

Output	{
	 “error” : false
	 “message”: “Your email has been updated successfully.”
	 }

Notes	The new email entered will be validated in the front-end part.

Method	PATCH
Name	updateAdminEmail
URL	admins/<string:userID>/patch/emails
Input	newEmail: String
Query	UPDATE AdminAccount SET email = newEmail WHERE userID = userID;

Output	{
	 “error” : false
	 “message”: “Your email has been updated successfully.”
	 }

Notes	The new email entered will be validated in the front-end part.

Method	PATCH
Name	updateUserPhoneNumber
URL	users/<string:userID>/patch/phone_numbers
Input	newPhoneNumber: String
Query	UPDATE UserAccount SET phoneNumber = newPhoneNumber WHERE userID = userID;

Output	{
	 “error” : false
	 “message”: “Your phone number has been updated successfully.”
	 }

Notes	The new phone number entered will be validated in the front-end part.

Method	PATCH
Name	updateAdminPhoneNumber
URL	admins/<string:userID>/patch/phone_numbers
Input	newPhoneNumber: String
Query	UPDATE AdminAccount SET phoneNumber = newPhoneNumber WHERE userID = userID;

Output	{
	 “error” : false
	 “message”: “Your phone number has been updated successfully.”
	 }

Notes	The new phone number entered will be validated in the front-end part.

Method	PATCH
Name	updateUserSubscriptionStatus
URL	users/<string:userID>/patch/premium_states
Input	premiumStatus: String
Query	UPDATE UserAccount SET isPremium = premiumStatus WHERE userID = userID;

Output	{
	 “error” : false
	 “message”: “Your subscription status has been updated successfully.”
	 }

Notes	

Method	PATCH
Name	updateUserSubscriptionEndDate
URL	users/<string:userID>/patch/subscription_end_dates
Input	newEndDate: String
Query	UPDATE UserAccount SET subscriptionEndDate = newEndDate WHERE userID = userID;
Output	{ “error” : false “message”: “Your subscription end date has been updated successfully.” }
Notes	

Method	PATCH
Name	updateAdminJobTitle
URL	admins/<string:userID>/patch/job_titles
Input	newJobTitle: String
Query	UPDATE AdminAccount SET jobTitle = newJobTitle WHERE userID = userID;
Output	{ “error” : false “message”: “Your job title has been updated successfully.” }
Notes	

REST API tables for changing source order settings

Method	PUT
Name	updateSourceOrder
URL	users/<string:userID>/put/source_orders
Input	source1 : String source2 : String source3 : String source4 : String source5 : String source6 : String source7 : String source8 : String

Query	UPDATE HasSourceOrder SET source = source1 WHERE userID = userID AND order = 1;
	UPDATE HasSourceOrder SET source = source2 WHERE userID = userID AND order = 2;
	UPDATE HasSourceOrder SET source = source3 WHERE userID = userID AND order = 3;
	UPDATE HasSourceOrder SET source = source4 WHERE userID = userID AND order = 4;
	UPDATE HasSourceOrder SET source = source5 WHERE userID = userID AND order = 5;
	UPDATE HasSourceOrder SET source = source6 WHERE userID = userID AND order = 6;
	UPDATE HasSourceOrder SET source = source7 WHERE userID = userID AND order = 7;
	UPDATE HasSourceOrder SET source = source8 WHERE userID = userID AND order = 8;

Output	{
	“error” : false
	“message”: “The order of each source has been updated successfully.”
	}

Notes	Order of source inputted from left to right.

Method	PUT
Name	resetSourceOrder
URL	users/<string:userID>/reset/source_orders
Input	None

Query	UPDATE HasSourceOrder SET source = “Merriam-Webster Learner’s Dictionary” WHERE userID = userID AND order = 1;
	
	UPDATE HasSourceOrder SET source = “Merriam-Webster Dictionary” WHERE userID = userID AND order = 2;
	
	UPDATE HasSourceOrder SET source = “Oxford English Dictionary” WHERE userID = userID AND order = 3;
	
	UPDATE HasSourceOrder SET source = “Urban Dictionary” WHERE userID = userID AND order = 4;
	
	UPDATE HasSourceOrder SET source = “Wikipedia” WHERE userID = userID AND order = 5;
	
	UPDATE HasSourceOrder SET source = “Google News” WHERE userID = userID AND order = 6;
	
	UPDATE HasSourceOrder SET source = “Google Images” WHERE userID = userID AND order = 7;
	
	UPDATE HasSourceOrder SET source = “YouTube” WHERE userID = userID AND order = 8;

Output	{
	“error” : false
	“message”: “The order of each source has been changed to the default setting successfully.”
	}

Notes	

<u>REST API tables for search history</u>

Method	PUT
Name	addSearchHistory
URL	users/<string:userID>/put/search_histories
Input	searchedDateTime : Timestamp word : string
Query	INSERT INTO HasSearchHistory (userID, searchedDateTime, word) VALUES (userID, searchedDataTime, word);

Output	{
	  "error" : false
	}

Notes	The search history is just displayed when the user clicks the text box for typing the word to be searched.

Method	GET
Name	getSearchHistory
URL	users/<string:userID>/get/search_histories
Input	None
Query	SELECT searchedDateTime, word FROM HasSearchHistory WHERE userID = userID;

Output	{
	  "error" : false
	}

Notes	The search history is just displayed when the user clicks the text box for typing the word to be searched.

Method	DELETE
Name	deleteAllSearchHistory

URL	users/delete/<string:userID>/all_search_histories
Input	None
Query	DELETE FROM HasSearchHistory WHERE userID = userID;

Output	{
	“error” : false
	“message”: “Your search histories have been deleted successfully.”
	}

Notes	

Method	DELETE
Name	deleteSearchHistory
URL	users/delete/<string:userID>/search_histories
Input	None
Query	DELETE FROM HasSearchHistory WHERE userID = userID AND word = word;

Output	{
	“error” : false
	“message”: “The search history has been deleted successfully.”
	}

Notes	

REST API tables for generating and deleting challenge

Method	POST
Name	addChallenge
URL	challenges/add/<string:challengeID>
Input	challengeID: String creatorID: String dateCreated: Date totalScore: Float maximumTime: Float

Query	INSERT INTO Challenge (challengeID, totalScore, maximumTime) VALUES (challengeID, totalScore, maximumTime);
	
	INSERT INTO GenerateChallenge (creatorID, challengeID, dateCreated) VALUES (creatorID, challengeID, dateCreated);

Output	{
	“error”: false,
	“message”: “The challenge has been generated successfully.”
	}

Notes	A new challenge is generated with challengeID. Each challenge would be set for total score and maximum time.

Method	DELETE
Name	deleteChallenge
URL	challenges/delete/<string:challengeID>
Input	None
Query	DELETE FROM GenerateChallenge WHERE creatorID = creatorID AND challengeID = challengeID;

Output	{
	“error”: false,
	“message”: “The challenge has been deleted successfully.”
	}

Notes	Since the delete option for challengeID attribute for GenerateChallenge table is ‘CASCADE,’ the query above will automatically delete the corresponding record in the Challenge table as well.

REST API tables for creating and deleting challenge problem

Method	POST
Name	addChallengeProblem
URL	challenges/<string:challengeID>/add/problems
Input	challengeID: String problemID: String problem: String answer: Integer firstChoice: String secondChoice: String thirdChoice: String fourthChoice: String selected: String

Query	INSERT INTO HasChallengeProblem (challengeID, problemID) VALUES (challengeID, problemID);
	
	INSERT INTO Problem (problemID, problem, anser, firstChoice, secondChoice, thirdChoice, fourthChoice, selected) VALUES (problemID, problem, answer, firstChoice, secondChoice, thirdChoice, fourthChoice, selected);

Output	{
	    "error": false,
	    "message": "The challenge problem has been created successfully."
	}

Notes	Before making challenge problems, it must be verified existence of challenge with challengeID. Then create a problem with problemID.

Method	DELETE
Name	deleteChallengeProblem
URL	challenges/<string:challengeID>/problems/<string:problemID>/delete
Input	problemID: String
Query	DELETE FROM HasChallengeProblem WHERE problemID = problemID;

Output	{
	    "error": false,

	“message”: “The challenge problem has been deleted successfully.” }
Notes	Since the delete option for problemID attribute in HasChallengeProblem table is ‘CASCADE,’the query above will automatically delete the corresponding record in the Problem table as well.

REST API tables for taking a challenge

Method	PUT
Name	addChallengeResult
URL	take_challenges/<string:challengeID>/add
Input	userID: String challengeID: String score: Float timeTaken: Float

Query	INSERT INTO TakeChallenge (userID, challengeID, score, timeTaken) VALUES (userID, challengeID, score, timeTaken);
	
	INSERT INTO HasChallengeRanking (challengeID

Output	{
	    “error”: false
	}

Notes	

Method	PUT
Name	addChallengeRanking
URL	challenge_rankings/<string:challengeID>/add
Input	challengeID: String userID: String rank: Integer score: Float timeTaken: Float
Query	INSERT INTO HasChallengeRanking (challengeID, userID, rank, score, timeTaken) VALUES (challengeID, userId, rank, score, timeTaken);

Output	{
	    “error”: false
	}

Notes	The rank should be automatically determined by the system.

	If the ranking of previous users shall be updated due to new results, then the system shall automatically update those results as well (as shown below). It is doubtful whether a HasChallengeRanking table is necessary, as there is a similar table called TakeChallenge. The difference is that one has a rank attribute, but the other doesn't.

Method	PATCH
Name	updateRanking
URL	challenge_rankings/<string:challengeID>/patch
Input	newUserID: String newScore: Float newTimeTaken: Float
Query	UPDATE HasChallengeRanking SET userID = newUserID, score = newScore, timeTaken = newTimeTaken WHERE challengeID = challengeID;

Output	{
	    "error": false
	}

Notes	If the ranking of previous users shall be updated due to new results, then the system shall automatically update those results as well.

Method	GET
Name	showRanking
URL	challenge_rankings/<string:challengeID>/get
Input	None
Query	SELECT userID, rank, score, timeTaken FROM HasChallengeRanking WHERE challengeID = challengeID;

Output	{
	   "userID": String,
	   "Rank": Integer,
	   "score": Float,
	   "timeTaken": Float
	}

Notes	Display the challenge rank and userId of the corresponding challenge.

<u>REST API tables for creating and deleting note cards</u>

Method	POST
Name	addCardPackage
URL	users/<string:userID>/add/card_packages
Input	userID: String packageID: String packageTitle: String

Query	INSERT INTO HasCardPackage (userID, packageID) VALUES (userID, packageID);
	
	INSERT INTO CardPackage (packageID, packageTitle) VALUES (packageID, packageTitle);

Output	{
	    “error”: false,
	    “message”: “New card package has been created.”
	}

Notes	

Method	POST
Name	addNoteCard
URL	card_packages/<string:packageID>/add/note_cards
Input	noteCardID: String title: String definition: String partOfSpeech: String photoURL: String packageID: String

Query	INSERT INTO HasNoteCard (packageID, noteCardID) VALUES (packageID, noteCardID);
	
	INSERT INTO NoteCard (noteCardID, title, definition, partOfSpeech, photoURL) VALUES (noteCardID, title, definition, partOfSpeech, photoURL);

Output	{

	“error”: false, “message”: “New note card has been created.” }
Notes	

Method	GET
Name	getNoteCard
URL	card_packages/<string:packageID>/note_cards/<string:cardID>/get
Input	None

Query	SELECT noteCardID FROM HasNoteCard WHERE packageID = packageID;
	
	SELECT title, definition, partOfSpeech, photoURL, videoURL FROM NoteCard WHERE noteCardID = noteCardID;

Output	{
	“title”: String,
	“definition”: String,
	“partOfSpeech”: String,
	“photoURL”: String,
	“videoURL”: String
	}

Notes	Get all note cards’ id that belongs to the card package from the first query. Then, get the information stored in the notecard for each noteCardID.

Method	DELETE
Name	deleteCardPackage
URL	card_packages/<packageID>/delete
Input	packageID: String
Query	DELETE FROM HasCardPackage WHERE packageID = packageID;
Output	{ “error”: false, “message”: “The card package has been deleted successfully.”

	}
Notes	Since the delete option for problemID attribute in HasCardPackage table is ‘CASCADE,’the query above will automatically delete the corresponding record in the CardPackage table as well.

Method	DELETE
Name	deleteNoteCard
URL	card_packages/<string:packageID>/note_cards/<string:noteCardID>/delete
Input	packageID: String noteCardID: String
Query	DELETE FROM HasNoteCard WHERE packageID = packageID AND noteCardID = noteCardID;

Output	{
	    “error”: false,
	    “message”: “The notecard is deleted successfully.”
	}

Notes	

REST API tables for sharing card packages

Method	POST
Name	shareCardPackage
URL	card_packages/<string:packageID>/share_options/add
Input	shareList: List accessType: String
Query	For each userID in shareList, INSERT INTO HasShareList (shareOptionID, userID, accessType) VALUES (shareOptionID, userID, accessType);

Output	{
	    “error”: false,
	    “message”: “Shared successfully.”
	}

Notes	

Method	GET
Name	getCardPackageShareList
URL	card_packages/<string:packageID>/share_options/get
Input	None

Query	SELECT shareOptionID FROM ShareOption WHERE ownerUserID = userID AND packageID = packageID;
	
	SELECT * FROM HasShareList WHERE shareOptionID = shareOptionID;

Output	{
	    “error”: false
	}

Notes	Find the shareOptionID that corresponds to the packageID that the user has created. Then, find the list of users and their access options from HasShareList table.

REST API tables for quiz and quiz histories

Method	PUT
Name	addQuizHistory
URL	quiz_histories/<string:userID>/add
Input	userID: String quizID: String completedDateTime: Timestamp score: Float
Query	INSERT HasQuizHistory (userID, quizID, completedDateTime, score) VALUES (userID, quizID, completedDateTime, score);

Output	{
	“error”: false
	}

Notes	

Method	GET
Name	getQuizHistory
URL	quiz_histories/<string:userID>/get
Input	None
Query	SELECT completedDateTime, score WHERE userID = userID;

Output	{
	“completedDateTime”: Timestamp,
	“score”: Float
	}

Notes	

REST API tables for generating quizzes & problems

Method	POST
Name	generateQuiz
URL	quizzes/add
Input	userID: String packageID: String quizID: String
Query	INSERT INTO GenerateQuiz (userID, packageID, quizID) VALUES (userID, packageID, quizID);

Output	{
	"error": false,
	"Message": "The new quiz is generated successfully."
	}

Notes	

Method	PUT
Name	addQuizProblem
URL	quizzes/<string:quizID>/problems/add
Input	quizID: String problemID: String
Query	INSERT INTO HasQuizProblem (quizID, problemID) VALUES (userID, packageID, quizID);

Output	{
	"error": false,
	"Message": "Quiz problem is added successfully."
	}

Notes	

REST API tables for taking quizzes

Method	PUT
Name	addQuizHistory
URL	users/<string:userID>/quiz_histories/<string:quizID>/add
Input	myCompletedDateTime: Timestamp myScore: Float

Query	INSERT INTO HasQuizHistory (userID, quizID, completedDateTime, score)
	VALUES (userID, quizID, myCompletedDateTime, myScore);

Output	{
	    “error”: false
	}

Notes	