API and Backend Documentation

Database Design:

- We used AWS DynamoDB for our database, a NoSQL database
- DynamoDB tables have Primary Keys made up of a Partition Key and an optional Sort Key
- Both tables use username as the partition key and neither table has a sort key

user_table:

- Purpose: stores data related to user-login
- Partition Key: username
- Sort Key: None

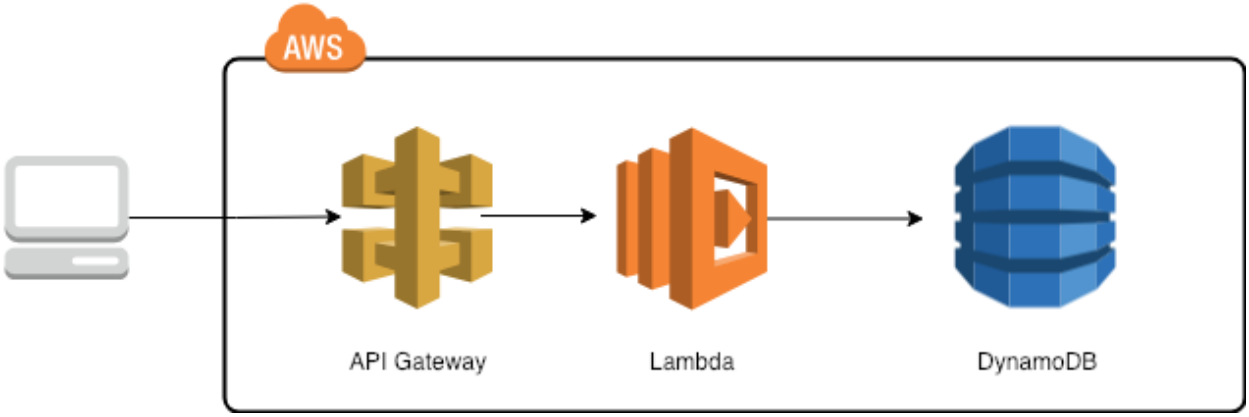
Item	Datatype	Notes
Username	String	Partition Key
Password	Integer	Hashed value using SHA-512 hash function
Email	String	

score_table:

- Purpose: stores data related to user gameplay
- Partition Key: username
- Sort Key: None

Item	Datatype	Notes
username	String	Partition Key
top_score	Integer	Calculated on front end
total_words_solved	Integer	Derived from words_solved
words_solved	Array<String>	
longest_word_solved	Integer	Derived from words_solved
games_played	Integer	

API Overview:

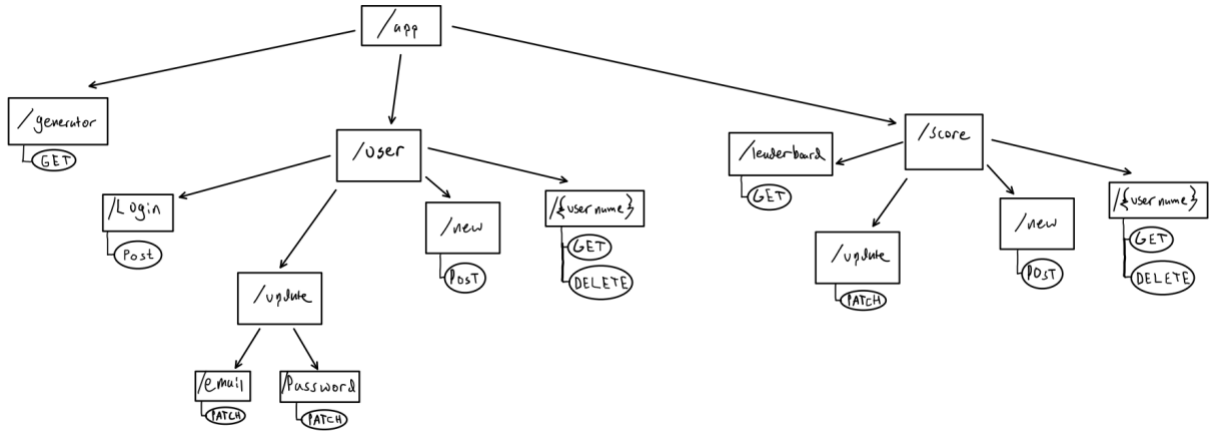


- The API is created using AWS API Gateway and AWS Lambda
- Each API call is set to trigger an associated Lambda function (written in python) upon invocation
- The API has three main branches: /generator, /user, and /score
- The methods under /user are used to perform CRUD operations on the user_table DynamoDB table. These methods are focused on logging in, creating an account, retrieving user data, and updating user data
- The methods under /score are used to perform CRUD operations on the score_table DynamoDB table. These methods are focused on storing user performance in the hangman game and retrieving leaderboard data
- All GET methods will either have a path parameter or no parameters, all DELETE methods will have a path parameter, and all other methods will have only body parameters
- Each Lambda function is wrapped in a try-except block
    - If the function raises an exception at any point a 400 error and status message will be returned detailing the source of the error
    - Otherwise, if the function runs successfully, a 200 status message will be returned along with some additional data
- All GET methods will return the RAW JSON directly from the DynamoDB tables
- All other methods will return a block of JSON formatted as follows (some methods will return additional attributes such as username, etc.)

```
{
        “statusCode”: 200,
        “body”: {
                “success”: True,
                “message”: “Successfully created user”
            }
}
```

- Some methods will return {“success”: False} to indicate to the front-end that the user has entered an invalid username, password, etc.

API Diagram



- /
  - /app
    - /generator
      - GET
    - /score
      - /leaderboard
        - GET
      - /new
        - POST
      - /update
        - PATCH
      - /{username}
        - DELETE
        - GET
    - /user
      - /login
        - POST
      - /new
        - POST
      - /update
        - /email
          - PATCH
        - /password
          - PATCH
      - /{username}
        - DELETE
        - GET

Root Endpoint

/app
- No methods

Generator Endpoint

/app/generator
- GET Method
    - Returns a single randomly generated word from http://random-word-api.herokuapp.com/home
    - Body parameters:
        - None

User Endpoints

/app/user/new
- POST method
- Creates a new user in user_table
- Body parameters:
    - Username (str)
    - Password (str)
        - Stores a hashed value of the password using SHA512 hash function
    - Email
- Returns:
    - True if username is not already taken and user is created
    - False if username is already taken and user is not created

/app/user/update/email
- PATCH method
- Updates user email in user_table
- Assumes correct username since the user will already have logged in on front end in order to invoke this method
- Body parameters
    - Username (str)
    - Email (str)
- Returns:
    - True if email is successfully updated

/app/user/update/password
- PATCH method
- Updates user password in user_table only if the old password matches
- Assumes correct username since the user will already have logged in on front end in order to invoke this method
- Body parameters:

        - Username (str)
        - Password (str)
        - New_password (str)
            - Passwords are hashed using SHA512 hash function
- Returns
    - True if the passwords match and the password is successfully updated
    - False if the passwords do not match and the password is not updated

/app/user/{username}

- GET method
    - Body parameters:
        - None
    - Returns user data from user_table based on username path parameter

- DELETE method
    - Body parameters:
        - None
    - Deletes a user from user_table based on the username path parameter
    - Assumes correct username since the user will already have logged in on front end in order to invoke this method
    - Returns:
        - True if user successfully deleted

/app/user/login

- POST method
- Body parameters
    - Username
    - Password
- Check if username and hash value of password matches that stored in database
- Returns:
    - True if username exists in database and passwords match
        - Also returns the username to be stored as a global variable in the frontend
    - False if username does not exist in database / passwords do not match

Score Endpoints

/app/score/{username}

- GET Method
    - Body parameters:
        - None
    - Returns data from leaderboard_table based on username path parameter

- DELETE method
    - Body parameters:
        - None
    - Delete user from leaderboard_table based on username path parameter

    - Assumes correct username since the user will already have logged in on front end in order to invoke this method
    - Returns:
        - True if user successfully deleted

/app/score/new

- POST method
- Creates a new entry in leaderboard_table
- Body parameters
    - Username
- Initializes all other values in table as follows:
    - top_score: 0
    - words_solved: []
    - total_words_solved: 0
    - longest_word_solved: 0
    - games_played: 0
- Returns:
    - True if username is not already taken and user is created
    - False if username is already taken and user is not created

/app/score/update

- PATCH method
- Update the top score, words solved, number of words solved, longest word, games played after the user finishes a game
- Stores the word in score_table if the user solves the hangman game
- Assumes correct username since the user will already have logged in on front end in order to invoke this method
- Body parameters:
    - Username
    - Score
    - Word
        - If word = “” (empty string)
            - --> then the user failed to solve the puzzle
- Returns:
    - True if score_table is successfully updated

/app/score/leaderboard

- GET method
- Body paramters:
    - None
- Return entire score_table for all users sorted by top_score