Endpoints Document:

Max Wilson

Endpoints:

/user/auth

/user/find/all

/user/find/email

/user/find/email/basic

/user/find/name

/user/find/following

/user/find/following/status

/user/day/total

/user/add

/user/add/following

/user/day/add/food

/user/update

/user/update/password

/user/delete/email

/user/delete/all

/status/find/all

/status/add

/status/delete/all

/user/auth

What it does:

Used for login. Takes in email and password and returns success if email and password match.

HTTP Method:

GET

Input: URL Params

email: String denoting the email of the user that's logging in

password: String denoting the password for the user

Output: JSON

response: Int, will be 1 for success and 0 for failure

message: String, describes the issue if the response is 0

/user/find/all

What it does:

Query all users in the User collection.

HTTP Method:

GET

Input:

Nothing!

Output: JSON

users: User[], array of JSON objects for all users. Each user will have structure similar to the output in the /user/find/email endpoint. Try it in postman to see structure.

/user/find/email

What it does:

Query the data for one user specified in the url params.

HTTP Method:

GET

Input: URL Params

email: String, the email of the user you are searching for

Output: JSON

email: String, the user's email

name: String, user's name

age: Int, user's age

height: Int, user's stored height value

weight: Int, user's stored weight value

lifestyle: String, user's lifestyle. This will likely change soon

gender: String, user's gender

calorieLimit: Int, user's saved calorie limit

userType: String, user's type. The default user type is "default"

allergens: String[], String array of allergen names

following: String[], String array of following user's emails

foods: Food[], Array of Food JSON Objects associated with user

calendar: Map<String, Day>, Map or dictionary of Day JSON objects. The key is a date string with the format "MM/dd/yyyy"

/user/find/email/basic

What it does:

Query user data but not including following, foods, and calendar

HTTP Method:

GET

Input: URL Params

email: String, the email of the user you are searching for

Output: JSON

email: String, the user's email

name: String, user's name

age: Int, user's age

height: Int, user's stored height value

weight: Int, user's stored weight value

lifestyle: String, user's lifestyle. This will likely change soon

gender: String, user's gender

calorieLimit: Int, user's saved calorie limit

userType: String, user's type. The default user type is "default"

allergens: String[], String array of allergen names

/user/find/name

What it does:

Query all users with the same name.

HTTP Method:

GET

Input: URL Params

name: String, users' name

Output: JSON

Users: User[], array of JSON objects for all matching users.

/user/find/following

What it does:

Query for list of user's following users

HTTP Method:

GET

Input: URL Params

email: String, user's email

Output: JSON

Content: String[], array of user emails

/user/find/following/status

What it does:

Query all statuses from both the given user and all of the users they're following.

HTTP Method:

GET

Input: URL Params

email: String, the email of the user you are searching for

Output: JSON

content: Status[], array of statuses, each status has the following:

email: String, user's email

timestamp: String, date string in the format "MM/dd/yyyy HH:mm:ss.SS"

flagged: Boolean, true/false on if the status is flagged

message: String, the message for this status

/user/day/total

What it does:

Query a user's nutritional totals for either the current day or a given day

HTTP Method:

GET

Input: URL Params

email: String, user's email

date: String, optional param to specify the date you're querying. Format: "MM/dd/yyyy"

Output: JSON

calories: Int, total calories

sodium: Double

carbs: Double

protein: Double

fat: Double

cholesterol: Double

/user/add

What it does:

Add a user to the User collection

HTTP Method:

POST

Input: JSON

email: String, user's email

password: String, user's password

Output: JSON

response: Int, will be 1 for success and 0 for failure

message: String, describes the issue if the response is 0

/user/add/following

What it does:

Add a user to a user's list of following

HTTP Method:

POST

Input: JSON

email: String, user you want to modify

following: String, user you want to add to the list

Output: JSON

response: Int, will be 1 for success and 0 for failure

message: String, describes the issue if the response is 0

/user/day/add/food

What it does:

Add a food to user's day. If that food is not already stored in the user's account it will be added. Currently this only adds to the current day. Might modify in the future to add to any day for testing purposes

HTTP Method:

POST

Input: JSON

email: String, user's email

name: String, food name, must be unique for the user's list of food

calories: Int

sodium: Double

carbs: Double

protein: Double

fat: Double

cholesterol: Double

amount: Double, the number of servings eaten of this food

Output: JSON

response: Int, will be 1 for success and 0 for failure

message: String, describes the issue if the response is 0

/user/update

What it does:

Update a user's biometric data and other data. Name, height, weight, lifestyle, gender, calorieLimit, allergens

HTTP Method:

POST

Input: JSON

email: String, the user's email

name: String, user's name

age: Int, user's age

height: Int, user's stored height value

weight: Int, user's stored weight value

lifestyle: String, user's lifestyle. This will likely change soon

gender: String, user's gender

calorieLimit: Int, user's saved calorie limit

allergens: String[], String array of allergen names

Output: JSON

response: Int, will be 1 for success and 0 for failure

message: String, describes the issue if the response is 0

/user/update/password

What it does:

Change a user's password

HTTP Method:

POST

Input: JSON

email: String, user's email

password: String, new password to be set

Output: JSON

response: Int, will be 1 for success and 0 for failure

message: String, describes the issue if the response is 0

/user/delete/email

What it does:

Delete a user based on a given email

HTTP Method:

DELETE

Input: URL Params

email: String, the user's email

Output:

response: Int, will be 1 for success and 0 for failure

message: String, describes the issue if the response is 0

/user/delete/all

What it does:

Delete all users from User collection

HTTP Method:

DELETE

Input:

Nothing!

Output:

response: Int, will be 1 for success and 0 for failure

message: String, describes the issue if the response is 0

/status/find/all

What it does:

Query all statuses

HTTP Method:

GET

Input:

Nothing!

Output: JSON

content: Status[], array of statuses, each status has the following:

email: String, user's email

timestamp: String, date string in the format "MM/dd/yyyy HH:mm:ss.SS"

flagged: Boolean, true/false on if the status is flagged

message: String, the message for this status

/status/add

What it does:

Add a status using a user's email

HTTP Method:

POST

Input: JSON

email: String, user's email

message: String, the message for this status

Output: JSON

response: Int, will be 1 for success and 0 for failure

message: String, describes the issue if the response is 0

/status/delete/all

What it does:

Delete all statuses

HTTP Method:

DELETE

Input:

Nothing!

Output:

response: Int, will be 1 for success and 0 for failure

message: String, describes the issue if the response is 0