Foodflash Api Endpoints

updated 2/15/2023

Basic Information

The base url for all endpoints is `https://concierge.cooperstandard.org:8443/api`. The endpoints are listed by what needs to be appended to this base url, i.e. the endpoint listed as `/recipe/all` has a complete url `https://concierge/cooperstandard.org:8443/api/recipe/all`. Endpoints are divided by http method and which model they operate on, either recipes or users. See the notes under each endpoint for what is required and expected in the request body and headers. All request bodies should be raw json, this means request headers should include: `'Content-Type' : 'application/json'`. All objects also have a unique `_id` field in addition to what is outlined below, these ids are automatically assigned **do not include them when posting content**. Authentication is handled through JWT bearer tokens. These are stored in the header and expire an hour after they are issued. If a token is expired send a refresh request.

this is the current Recipe data model:

```
recipeSchema = {
    title: {
        required: true,
        type: String
    },
    description: {
        required: true,
        type: String
    },
    ingredients: {
        required: false,
        type: [String]
    },
    allergens: {
        type: [String],
        required: false
    },
    photos: {
        required: false,
        type: [String]
    },
    instructions: {
        required: false,
        type: String
    },
    prepTime: {
        required: false,
        type: String
```

```
    }
}
```

desciption is required to avoid namespace collisions, **this may change soon**. ingredients (and allergens) should be included but are not strictly required (this may change). Photos are currently stored as url strings and are not required, **this method of storage and optionality will change soon**. instructions are not required but should be included if relevant.

this is the current user data model

```
const userSchema = {
    name : {
        type : String,
        required : true
    },
    email : {
        type : String,
        required : true
    },
    password : {
        type : String,
        required : true
    },

    restrictions : {
        type : [String],
        required : false
    },
    oldToken : {
        type : String,
        required : false
    },

    saved : {
        type : [String],
        required: false
    }

}
```

This will change soon, hopefully not in ways that break existing functionality. Unlike recipe only name, restrictions, and saved are accesible by public endpoints. name stores the string of the users name, given at account registration and updatable with patch requests(see endpoint list). restrictions stores an array of dietary restrictions (these should match the set of allergens in recipes). saved is an array of recipe `_id`. The `oldToken` field stores the last issued token. **before we add real users I will need to encrypt the passwords in storage**

Get Endpoints

Recipes

- `/recipe/all`:
  - requires an access token in the header: `authorization : Bearer <token>`
  - no body is required
  - returns an array of all currently available recipes.
- `/recipe/search`:
  - requires access token in the header: `authorization : Bearer <token>`
  - no body is required
  - requires a query term, ie `/api/recipe/search?term=spam`
  - returns all recipes whose title's match the term
- `/user/liked`:
  - requires access token in the header: `authorization : Bearer <token>`
  - no body is required
  - returns an array containing the id's of liked recipes

Users

- `/authenticate`:
  - requires an access token in the header: `authorization : Bearer <token>`
  - no body is required
  - if the token is correct it returns the email associated with the account, if incorrect returns a status 500
- `/user/all`:
  - requires an admin access token `authorization : Bearer <admin token>`
  - no body is required
  - returns an array of all user accounts currently in the database

Post Endpoints

Recipes

- `/recipe`
  - requires `authorization : Bearer <admin token>`
  - body requires all the fields for the recipe to be posted for example:

```
{
    "title": "Fried Rice",
    "description": "Rice fried with onion and egg",
    "ingredients": [
        "rice",
        "vegetable oil",
        "egg",
        "onion",
        "garlic",
        "soy sauce"
    ],
```

```
    "allergens": [
        "egg",
        "onion",
        "soy"
    ],
    "photos": [
        "https://cdn.britannica.com/06/234806-050-49A67E27/SPAM-
can.jpg"
    ],
    "instructions": "Cook the rice. Thinly slice the onions and
garlic. heat oil in a pan on med-high heat. Add Garlic and onion to
the pan and fry until slightly browned. crack egg into the pan and
scramble, once almost scrambled add rice and soy sauce. Turn heat to
high and mix ingredients, fry for another 3 minutes",
    "prepTime": "15 minutes"
}
```


    - returns a json representation of the posted recipe

TODO:

- [ ]: Dislike Recipe

Users

- `/user/signup`
    - requires an email, password, and name in the body. The email must be unique.
    - returns a jwt and the user information if successful, returns a 401 status if an email is already associated with that account
- `/user/login`
    - requires an email and password in the body
    - returns a jwt if successful and a 401 status if unsuccessful
- `/user/refresh`:
    - no auth header required
    - requires user id, email, and current token in the body
    - will only work if current session has the most recent token generated
    - returns a new token
- `/user/like`:
    - requires an access token in the header: `authorization : Bearer <token>`
    - requires the id of the recipe to like in the body stored like `recipe : <id here>`
    - returns status 200 if successful

Patch Endpoints

Recipes

- `/recipe/id/:id`
    - requires an access token in the header: `authorization : Bearer <token>`
    - replace `:id` at the end of the url with the id of the recipe to update
    - the body should contain the json of the state of the fields to update

    - returns the new version of the object
- `/recipe/title`
    - requires an access token in the header: `authorization : Bearer <token>`
    - the title of the recipe should be stored in a query parameter
    - the body should contain the json of the state of the fields to update
    - returns the new version of the object

Users

- `/user/:id`
    - requires an access token in the header: `authorization : Bearer <token>`
    - replace `:id` at the end of the url with the id of the user
    - the body should contain the json of the state of the fields to update
    - returns the new version of the object

TODO:

- [ ]: Patch by email

Delete Endpoints

Recipes

These are still a work in progress

Users

TODO:

- [ ]: Delete recipe by id
- [ ]: Delete Recipe by title
- [ ]: Delete User by id
- [ ]: Delete User by email