PIIQUANTE

Application API Requirements

Project Context

Piiquante focuses on creating spicy sauces whose recipes are secret. To build on its success and generate more buzz, the company wants to create a web application in which users can add their favorite sauces and like or dislike ones added by others.

API Requirements

	Endpoint	Request body (where applicable)	Expected response type	Function
POST	/api/auth/signup	{ email: string, password: string }	{ message: string }	Hashes user password, adds user to database.
POST	/api/auth/login	{ email: string, password: string }	{ userId: string, token: string }	Checks user credentials, returning the user's _id from the database and a signed JSON web token (also containing the user's _id).
GET	/api/sauces	-	Array of sauces	Returns an array of all sauces in the database.
GET	/api/sauces/:id	-	Single sauce	Returns the sauce with the provided _id.
POST	/api/sauces	{ sauce: String, image: File }	{ message: String } **Verb**	Captures and saves the image, parses stringified sauce, and saves it to the database, setting its imageUrl correctly. Initializes sauces likes and dislikes to 0, and usersLiked and usersDisliked to empty arrays. Note that the initial body request is empty; when multer is added, it returns a string for the body request based on the data submitted with the file.

PUT	/api/sauces/:id	EITHER Sauce as JSON OR { sauce: String, image: File }	{ message: String }	Updates the sauce with the provided _id. If an image is uploaded, capture it, and update the sauces imageUrl. If there are no files, the sauce details are directly within the request body (req.body.name, req.body.heat, etc.). If a file is provided, the stringified sauce is in req.body.sauce. Note that the initial body request is empty; when multer is added, it returns a string of the body request based on the data submitted with the file.
DELETE	/api/sauces/:id	-	{ message: String }	Deletes the sauce with the provided _id.
POST	/api/sauces/:id/like	{ userId: String, like: Number }	{ message: String }	Sets “like'' status for the userId provided. If like = 1, the user likes the sauce. If like = 0, the user is canceling their like or dislike. If like = -1, the user dislikes the sauce. The user's ID must be added to or removed from the appropriate array. This keeps track of their preferences and prevents them from liking or disliking the same sauce multiple times: one user can only have one value for each sauce. Total number of likes and dislikes to be updated with each like.

API Errors

Any errors must be returned as they are thrown, with no modification or additions. If needed, use new Error().

API Routes

All Sauce routes must have authorization (the token is sent from the front-end with the Authorization header: "Bearer <token>"). Before the user can make changes to the Sauce route, the code must check if the current userId matches the userId for the sauce. If the userId does not match, send “403: unauthorized request.” This ensures that only the Sauce owner can make changes.

Data ModelsSauce

- **userId**: *String* — the MongoDB unique identifier for the user who created the sauce.
- **name**: *String* — name of the sauce.
- **manufacturer**: *String* — manufacturer of the sauce.
- **description**: *String* — description of the sauce.
- **mainPepper**: *String* — the main pepper ingredient in the sauce.
- **imageUrl**: *String* — the URL for the picture of the sauce uploaded by the user.
- **heat**: *Number* — a number between 1 and 10 describing the sauce.
- **likes**: *Number* — the number of users liking the sauce.
- **dislikes**: *Number* — the number of users disliking the sauce.
- **usersLiked**: *["String <userId>"]* — an array of user IDs of those who have liked the sauce.
- **usersDisliked**: *["String <userId>"]* — an array of user IDs of those who have disliked the sauce.

User

- **email**: *String* — the user's email address **[unique]**.
- **password**: *String* — hash of the user's password.

Security Requirements

- User passwords must be hashed.
- Authentication must be reinforced on all of the required Sauce routes.
- Email addresses in the database are unique. An appropriate Mongoose plugin is used to ensure their uniqueness and report errors.
- Security of the MongoDB database (from a service such as MongoDB Atlas) must not impede the application from launching on a user's machine.
- A Mongoose plugin must ensure reporting of database errors.
- The most recent software versions are used with updated security patches.
- The content of the images folder must not be uploaded to GitHub.

GitHub Repository

Pull the front-end app code from the project repo and take the following steps:

1. Clone repo.
2. Open a terminal (Linux/Mac) or command prompt/PowerShell (Windows).
3. Run npm install from within the project directory.
4. Run npm start.
5. Run the back end on http://localhost:3000 only.

If you are using VSCode, use the LiveShare extension to run the front-end server without using npm install.