Rottenreact

POST Register

http://localhost:3001/users/signup

POST /users/signup

This endpoint allows a new user to register by providing their email and password inside a user object.

Request:

- **Headers:**
  - Content-type: application/json
- **Body (JSON):**
  - user (object): Contains the users credentials.
    - email (string): The users email address.
    - password (string): The users password.

Response:

- **201 OK:**
  - Returns a JSON object containing the new users ID (UUID) and email.
- **400 BAD REQUEST:**
  - Email and password is required.
- **409 CONFLICT:**
  - Email already in use.

AUTHORIZATION

Algorithm	<algorithm>
Is Secret Base64Encoded	<is-secret-base64encoded>
Payload	<payload>
Add Token To	<add-token-to>
Header Prefix	<header-prefix>
Query Param Key	<query-param-key>

Header	<header>

HEADERS

Content-Type	application/json

PARAMS

Body raw (json)

```
json

{
  "user": {
    "email": "user@gmail.com",
    "password": "Password123"
  }
}
```

POST Login

http://localhost:3001/users/signin

POST /users/signin

This endpoint is to authenticate a user by verifying their email and password. By signing in, you get additional features in your use.

Request:

- **Headers:**
  - Content-type: applications/json
- **Body (JSON):**
  - user (object): Contains the users credentials.
    - email (string): The users email adress.
    - password (string): The users password.

Response:

- **200 OK:**

    - Returns a JSON object containg users ID, email and token.
- **400 BAD REQUEST:**
    - Email and password are required.
- **401 UNAUTHORIZED:**
    - Invalid email or password.

AUTHORIZATION

Algorithm	<algorithm>
Is Secret Base64Encoded	<is-secret-base64encoded>
Payload	<payload>
Add Token To	<add-token-to>
Header Prefix	<header-prefix>
Query Param Key	<query-param-key>
Header	<header>

HEADERS

Content-Type	application/json

Body raw (json)

```json
{
  "user": {
    "email": "user@gmail.com",
    "password": "Password123"
  }
}
```

POST Logout

This endpoint does not exist in backend, logout is handled on frontend.

When the user logs out, the frontend removes the authetication token from the session storage, which ends the session.

Frontend logic:

const signout = () => {

sessionsStorage.removeItem('user');

setUser(null);

};

DELETE Delete user

http://localhost:3001/users/delete/b47a6237-0b89-4e5e-ba0a-e016d48cb143

Delete /users/delete/:id

This endpoint is to delete an existing account.

Request:

- **Headers:**
  - Content-type: application/json.
  - Authorization: Bearer token.
- **Body:**
  - None.
- **Params:**
  - Leave this empty.

Response:

- **200 OK:**
  - Returns "user deleted succesfully" and shows the deleted user in JSON object containing user ID and email.
- **400 BAD REQUEST:**
  - User ID is required.
- **404 NOT FOUND:**
  - User not found in the database.

HEADERS

Content-Type	application/json
Authorization	bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJiNDdhNjIzNy0wYjg5LTRlNWUtYmEwYS1lMDE2ZDQ4Y2IxNDMiLCJlbWFpbCI6InVzZXJAZ21haWwuY29tIiwiaWF0IjoxNzU5OTkwODA0LCJleHAiOjE3NjA1OTU2MDR9.bx8iDLrNKzyD4mCaCF_PyvhMj8Traj-5jTmFkqTtTF0

GET Request user

GET Browse reviews

GET /api/reviews

This endpoint is to browse movie reviews.

Request:

- **Headers:**
  - Content-type: application/json.
  - Authorization: Bearer token.
- **Body:**
  - None.
- **Params:**
  - Leave this empty.

Response:

- **200 OK:**
  - Returns all current reviewed movies by users, containing movie_id, user_id, id, raiting, created_at and user.