SocialHive API Documentation

Elisea Mizzi

Part 2 - Using a 3rd Party API

As someone new to backend development, I wanted to include something fun in my project that would also let me practice using a 3rd party API. I wasn't sure what would work at first, but after doing some research, I discovered the **Official Joke API**, which sends random jokes in a simple format. I thought it would be a great way to add a light-hearted feature to the homepage of SocialHive and also meet the criteria for using an external API.

Why I Chose This API

I picked the Official Joke API because it was beginner-friendly. It didn't require any authentication, tokens, or signup—just a simple URL. This made it a perfect fit for someone still learning the basics. The documentation on their GitHub was clear, with examples of what the response looked like and how to access it. I didn't have to worry about any complicated parameters or rate limits.

What the API Returns

The Official Joke API returns a random joke in JSON format with two main keys: setup and punchline. Here's an example of what the response looks like when you make a request:

```
{
  "type": "general",
  "setup": "Why did the burglar hang his mugshot on the wall?",
  "punchline": "To prove that he was framed!"
}
```

This was easy to use in PHP because all I had to do was fetch the API response and combine the setup and punchline into one string.

How I Used It in My Project

On the homepage (index.php), I used PHP's curl function to make a GET request to the API endpoint (https://official-joke-api.appspot.com/random_joke). This happens every time the homepage loads. Here's a simplified version of the code I added:

```php
$curl = curl_init("https://official-joke-api.appspot.com/random_joke");
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($curl);
curl_close($curl);

if ($response) {
    $joke_data = json_decode($response, true);
    if (isset($joke_data['setup'], $joke_data['punchline'])) {
        $joke = $joke_data['setup'] . " - " . $joke_data['punchline'];
    } else {
        $joke = "No joke available right now.";
    }
}
```

This little block checks for a successful response, decodes it, and then displays it in a friendly way on the homepage. It's a small feature, but it adds personality to the project.

Handling Issues

One of the challenges I ran into was making sure that if the API failed (for example, if there was no internet connection or the API was down), it didn't break the homepage. I added a fallback message like "No joke available right now." to handle that gracefully. That way, the page still works even if the joke doesn't load.

How I Understood the API

Since it was my first time using a 3rd party API, I spent a bit of time reading through their GitHub documentation. Luckily, they had clear examples and explained that no headers or keys were required. That made testing much easier. I also looked at some tutorials on using cURL with PHP to make sure I was doing it right.

Part 3 – Testing My API

After building all the functionality, I needed to make sure it actually worked. That's where **Postman** came in. I used it to send requests to my API and check if the responses were correct. It was a huge help in catching bugs and improving how I handled errors.

Creating My Workspace

I made a new workspace in Postman called **API-SocialMediaApplication**. Inside this workspace, I created separate **collections** for each main area of the API:

- Users
- Posts
- Comments
- Likes
- Messages

- Groups
- Notifications

Each collection contained all the relevant requests like

- POST for creating new things like users or posts
- GET for fetching data like comments or followers
- DELETE to remove posts, users, or messages
- POST/PUT for updates

How I Tested

For example, when testing the create_user.php file, I:

- Set the request type to POST
- Used this URL: http://localhost/SocialMediaApplication1/SocialMediaApplication1/api/User/create_user.php
- Went to the "Body" tab, selected raw (JSON) and added:

```
{
  "username": "elise",
  "email": "elise@example.com",
  "password": "mypassword"
}
```

- Then I clicked send and checked if I got a 201 Created response.

If everything worked correctly, I received a 201 Created status and a success message.

Testing All the Endpoints

I repeated this process for every single API route. Some of the other tests I ran included:

- **Getting Posts**: Checking if /get_posts.php returned all posts
- **Liking a Post**: Sending a POST to /like_post.php and confirming the status updated
- **Commenting**: Posting a comment and verifying it was stored correctly
- **Following/Unfollowing Users**: Testing both follow and unfollow actions
- **Fetching Notifications**: Making sure notifications were accurate per user

For each test, I looked for:

- The correct **HTTP response code** (like 200, 201, 400, or 404)
- JSON that made sense and was structured properly
- Useful **error messages** when something went wrong
- How it handled **missing data**, invalid IDs, or duplicated values

What I Learned from Testing

Postman taught me a lot about how APIs behave in real-world use. I learned how important it is to:

- Handle errors clearly
- Return the right status codes
- Make sure the API is easy to understand from the outside
- Create consistent and predictable output

I also learned how useful it is to **save and export** collections. I exported all my collections and added them to the project's repository. That way, anyone else can test the API easily by importing them into Postman.

Summary

This was my first time working with a 3rd party API. I wanted to keep things simple but still learn something valuable. By using the Official Joke API, I added a light and fun feature that shows how an external API can enhance a project. And by testing thoroughly with Postman, I made sure everything worked as it should.