Absolutely! Here's the provided code examples reformatted with line numbers and additional explanations for clarity:

From Gemini 3-8-2024

**GET Call**

Python

1. fetch('http://127.0.0.1:8000/api/menu-items')

2. .then(response => response.json())

3. .then(data => {

4. console.log(data)

5. })

**Explanation:**

* **Line 1:** Initiates a GET request using fetch to retrieve data from the menu items endpoint.
* **Line 2:** Handles the response, converting it to JSON format.
* **Line 3-4:** Processes the parsed JSON data (e.g., display on the console).

**POST Call**

Python

1. const payload = {

2. "title": "Ambrosia Ice cream",

3. "price": 5.00,

4. "inventory": 100

5. }

6. const endpoint = 'http://127.0.0.1:8000/api/menu-items'

7. fetch(endpoint,

8. {

9. method: 'POST',

10. headers: {

11. 'Accept': 'application/json',

12. 'Content-Type': 'application/json'

13. },

14. body: JSON.stringify(payload)

15. })

**Explanation:**

* **Lines 1-5:** Defines the payload object containing data for the new menu item.
* **Line 6:** Stores the API endpoint URL.
* **Lines 7-15:** Performs a POST request, including:
  + **method:** 'POST'
  + **headers:** Indicates JSON data exchange.
  + **body:** Sends the stringified JSON payload.

**DELETE Call**

Python

1. const endpoint = 'http://127.0.0.1:8000/api/menu-items/17'

2. fetch(endpoint,

3. {

4. method: 'DELETE',

5. headers: {

6. 'Accept': 'application/json',

7. 'Content-Type': 'application/json'

8. }

9. })

10. .then(response => response.json())

**Explanation:**

* **Line 1:** The URL includes ID '17', specifying the resource to delete.
* **Lines 2-9:** Makes a DELETE request, including headers.
* **Line 10:** Optionally handles the response (which might be empty or contain confirmation).

**Authenticated POST Call**

Python

1. const endpoint = 'http://127.0.0.1:8000/api/menu-items/17'

2. const token = "Some token"

3. fetch(endpoint,

4. {

5. method: 'POST',

6. headers: {

7. 'Accept': 'application/json',

8. 'Content-Type': 'application/json',

9. 'Authorization': 'Bearer ' + token

10. }

11. })

**Explanation:**

* **Line 2:** Stores your authentication token.
* **Lines 3-11:** Makes a POST request.
  + **Line 9:** Includes the 'Authorization' header with the bearer token for authentication.

**Let me know if you would like me to elaborate on any aspect or provide examples for PUT/PATCH calls!**

It looks like you're asking to format JavaScript code snippets with line numbers, but the mention of Python code might have been a mistake. I'll proceed with formatting the JavaScript code snippets you've provided for making API calls with line numbers for clarity.

### Making a GET call

```javascript

1 | fetch('http://127.0.0.1:8000/api/menu-items')

2 | .then(response => response.json())

3 | .then(data => {

4 | console.log(data)

5 | })

```

### Making a POST call

```javascript

1 | const payload = {

2 | "title": "Ambrosia Ice cream",

3 | "price": 5.00,

4 | "inventory": 100

5 | }

6 | const endpoint = 'http://127.0.0.1:8000/api/menu-items'

7 | fetch(endpoint,

8 | {

9 | method: 'POST',

10 | headers: {

11 | 'Accept': 'application/json',

12 | 'Content-Type': 'application/json'

13 | },

14 | body: JSON.stringify(payload)

15 | })

16 | .then(response => response.json())

17 | .then(data => console.log(data))

```

### Making DELETE call

```javascript

1 | const endpoint = 'http://127.0.0.1:8000/api/menu-items/17'

2 | fetch(endpoint,

3 | {

4 | method: 'DELETE',

5 | headers: {

6 | 'Accept': 'application/json',

7 | 'Content-Type': 'application/json'

8 | }

9 | })

10 | .then(response => response.json())

11 | .then(data => console.log(data))

```

### Making an authenticated POST call

```javascript

1 | const endpoint = 'http://127.0.0.1:8000/api/menu-items/17'

2 | const token = "Some token"

3 | fetch(endpoint,

4 | {

5 | method: 'POST',

6 | headers: {

7 | 'Accept': 'application/json',

8 | 'Content-Type': 'application/json',

9 | 'Authentication': 'Bearer ' + token

10 | },

11 | body: JSON.stringify({/\* your payload here \*/})

12 | })

13 | .then(response => response.json())

14 | .then(data => console.log(data))

```

These snippets demonstrate the basic structure for making GET, POST, PUT, PATCH, DELETE, and authenticated calls using the `fetch` API in JavaScript, formatted with line numbers for easier reading and understanding.  
  
This lesson plan was awkward for me because it wanted us to use a previous project to try the code snippets on. However I had to learn how to copy my previous project, not to mention find the correct project. I finally copied the little lemon project to the directory holding this lesson plane.  
  
I read the instructions and possibly did not know enough to fill in the blanks. I looked at the code below:

fetch('http://127.0.0.1:8000/api/menu-items')

.then(response => response.json())

.then(data => {

console.log(data)

})

And was confused how to use it in a program or template. Since the LittleLemon project had no templates I create one called “js\_example.html and placed it in a templates folder. Below is my file:

<!DOCTYPE html>

<html>

<head>

    <title>Little Lemon Menu (JS Example)</title>

</head>

<body>

    <script>

        fetch('http://127.0.0.1:8000/api/menu-items')

            .then(response => response.json())

            .then(data => {

                console.log(data)

            })

            .catch(error => console.error('Error:', error));

    </script>

</body>

</html>

I created a view function:

def js\_example(request):

menu\_items = MenuItem.objects.all()  # Fetch menu items using Django ORM

context = {'menu\_items': menu\_items}

    return render(request, 'js\_example.html', context)

and added this url to the app level urls.py:

    path('js-example/', views.js\_example, name='js\_example'),

After completing the code I was confused since there was no output to the terminal like I expected, however with the help of Gemini I was able to learn that the output is not in the terminal but the browser console located in developer tools…yes I know we went over this. The data was there and I was ecstatic. I played around a little and printed the data through the use of stringify as suggested in the lesson. My code is below. I commented out the original template code and added the following:

<!DOCTYPE html>

<html>

<head>

    <title>Little Lemon Menu (JS Example)</title>

</head>

<body>

    <!-- Placeholder for the fetched data -->

    <div id="menu-items"></div>

    <script>

        fetch('http://127.0.0.1:8000/api/menu-items')

            .then(response => response.json())

            .then(data => {

                // Convert the data to a JSON string

                const jsonString = JSON.stringify(data, null, 2); // The 'null' and '2' arguments format the JSON string for readability

                // Display the JSON string

                document.getElementById('menu-items').textContent = jsonString;

            })

            .catch(error => console.error('Error:', error));

    </script>

</body>

</html>

The following was printed to the browser:

{ "count": 2, "next": null, "previous": null, "results": [ { "id": 1, "title": "Beef Pasta", "price": "10.00", "featured": false, "category": 2 }, { "id": 2, "title": "Vanilla Ice Cream", "price": "5.00", "featured": false, "category": 1 } ] }

I tried the rest of the examples but was totally lost. I will return to this in the future. 3-8-2024 1329