Of course. For **Practical 5**, the core task is to demonstrate how your website dynamically displays data by fetching it, usually in JSON format, and then rendering it using JavaScript. Your project does this excellently with products and categories.

Here is a guide on what to write in your practical file for Practical 5.

Practical 5: Display Data using JSON

Aim: To implement dynamic data display on the ByteBazaar website by fetching product and category information asynchronously (using JSON) and rendering it into the HTML structure via JavaScript.

1. Key Questions & Analysis

1.1. How is data fetched?

Data is fetched asynchronously from the server using the fetch API, which returns data in JSON format. This approach ensures that the web page remains responsive while data is being loaded.

- **Endpoint:** Data is requested from dedicated PHP API endpoints (e.g., api/get_products.php, api/get_categories.php, api/get_product.php). These scripts query the MySQL database and encode the results as JSON.
- **Asynchronous Operation:** The fetch API is non-blocking, meaning the browser can continue to render the page while waiting for the server response. This is crucial for a smooth user experience.
- **Error Handling:** The fetch calls include .catch() blocks to handle potential network errors or issues with the server response, providing a robust user experience (e.g., displaying "Error fetching products").

Example of Data Fetching from js/products.js:

JavaScript

```javascript
// From js/products.js
```

```javascript
async function fetchProducts() {
    try {
        const response = await fetch('api/get_products.php'); // API endpoint
        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }
        const data = await response.json(); // Parse JSON response
        if (data.success) {
            allProducts = data.products; // Store products
            displayProducts(allProducts); // Render products
        } else {
            console.error('Error fetching products:', data.message);
            // Display an error message to the user
        }
    } catch (error) {
        console.error('Error fetching products:', error);
        // Display a user-friendly error message
    }
}
```

1.2. How is data parsed and rendered?

Once the JSON data is received, it is parsed and then used to dynamically generate HTML elements, replacing static content.

- **Parsing:** The response.json() method is used to parse the incoming JSON string into a JavaScript array of objects.
- **Dynamic HTML Generation:** A loop iterates through the parsed product or category objects. For each object, an HTML string is constructed using template literals (backticks `) which allows for easy embedding of variable data.
- **DOM Injection:** The generated HTML strings are then injected into the appropriate container elements in the DOM (e.g., .products-grid for product cards, .category-grid for category cards) using innerHTML.

Example of Data Rendering from js/products.js:

JavaScript

```javascript
// From js/products.js
function displayProducts(productsToDisplay) {
    const productsGrid = document.getElementById('productsGrid');
    if (!productsGrid) return;

    productsGrid.innerHTML = ''; // Clear existing products
    productsToDisplay.forEach(product => {
        const productCard = `
            <div class="product-card">
                <div class="product-image">
                    <img src="${product.image}" alt="${product.title}">
                    <div class="product-overlay">
                        <button class="btn-icon wishlist-btn" aria-label="Add to wishlist"
data-product-id="${product.id}">
                            <i class="far fa-heart"></i>
                        </button>
                        <button class="btn-icon quick-view-btn" aria-label="Quick view">
                            <i class="fas fa-eye"></i>
                        </button>
                    </div>
                    ${product.badge ? `<div class="product-badge">${product.badge}</div>` : ''}
                </div>
                <div class="product-info">
                    <h3>
                        <a href="product-details.html?id=${product.id}"
class="product-title">${product.title}</a>
                        <div class="product-rating">
                            <i class="fas fa-star"></i>
                            <span>${product.rating}</span>
                        </div>
                    </h3>
                    <p class="product-price">${product.price}</p>
                    <button class="btn btn-primary add-to-cart-btn" data-product-id="${product.id}">Add to
Cart</button>
                </div>
            </div>
        `;
        productsGrid.innerHTML += productCard; // Append to the grid
    });
}
```

2. Filtering, Searching, and Sorting (Supplementary Problem)

The project effectively implements filtering, searching, and sorting functionalities as part of the dynamic data display.

- **Filtering:** Users can filter products by category. The filterProducts() function dynamically updates the displayed products based on selected categories.
- **Searching:** A search input allows users to search products by title.
- **Sorting:** Options are provided to sort products by price (low to high, high to low) and by rating.
- **Live Updates:** All these operations re-render the product grid in real-time without requiring a page reload, providing a highly interactive experience.

Example of Sorting Logic from js/products.js:

JavaScript

```javascript
// From js/products.js
function sortProducts(products, sortBy) {
  if (sortBy === 'price-asc') {
    products.sort((a, b) => parseFloat(a.price.substring(1)) - parseFloat(b.price.substring(1)));
  } else if (sortBy === 'price-desc') {
    products.sort((a, b) => parseFloat(b.price.substring(1)) - parseFloat(a.price.substring(1)));
  } else if (sortBy === 'rating-desc') {
    products.sort((a, b) => b.rating - a.rating);
  }
  // Default or other sorting (e.g., 'name-asc')
  return products;
}
```

3. Testing of Dynamic Content (Post-Laboratory Work)

You should include screenshots demonstrating the dynamic loading and manipulation of data.

Test Case 1: Products Page Loaded
[Insert a screenshot of the products.html page fully loaded with products fetched from the database.]
Test Case 2: Products Filtered by Category
[Insert a screenshot of the products.html page after filtering by a specific category (e.g., "Electronics"), showing only relevant products.]
Test Case 3: Products Sorted by Price
[Insert a screenshot of the products.html page after sorting by "Price: Low to High," showing

products in the correct order.]
This comprehensive documentation, complete with code examples and visual proof, effectively demonstrates your successful implementation of Practical 5, showcasing dynamic data display using JSON and advanced filtering/sorting.