# Masterclass on Asynchronous JavaScript: Async/Await, Fetch API, and Building a Product Page with Cart Functionality

**Instructor: Pramod Kumar Jena**

---

## 1. Introduction to Asynchronous JavaScript

**What is Asynchronous Programming?**

- **Definition**: Asynchronous programming allows code to start tasks and continue running without waiting for the tasks to finish.
- **Real-Life Example**: Think of ordering food in a restaurant (you can place an order and do other tasks until the food is ready).

**Key Concepts:**

- **Blocking vs. Non-Blocking**:
  - Blocking: Code execution waits for a task to finish before moving on.
  - Non-Blocking: Code execution continues without waiting, allowing for multitasking.
- **Callback Hell**: When nested callbacks make code hard to read and maintain. Async/Await was introduced to simplify asynchronous code.

**Examples:**

```
// Synchronous code
console.log("Start");
console.log("Middle");
console.log("End");

// Asynchronous code with setTimeout
console.log("Start");
setTimeout(() => console.log("Middle"), 2000); // Executes after 2
seconds
console.log("End");
```

---

## 2. Introducing Async/Await and Promises

**Promises:**

- A **Promise** is an object that represents the eventual completion or failure of an asynchronous operation.
- **States** of a Promise:
    - `Pending`: Initial state, neither fulfilled nor rejected.
    - `Fulfilled`: The operation completed successfully.
    - `Rejected`: The operation failed.

**Async/Await:**

- **Async** functions return a promise and allow for handling asynchronous code with a cleaner syntax.
- **Await** pauses the execution of the async function until the promise settles.

**Example:**

```
// Function that returns a promise
function fetchData() {
    return new Promise((resolve) => {
        setTimeout(() => resolve("Data received"), 2000);
    });
}

// Using async/await
async function getData() {
    console.log("Fetching data...");
    const data = await fetchData();
    console.log(data); // "Data received"
}

getData();
```

---

## 3. The Fetch API

**What is the Fetch API?**

- The Fetch API allows you to make network requests to retrieve resources, commonly used for fetching data from APIs.
- **Fetch** returns a promise and allows handling asynchronous HTTP requests.

**Basic Syntax:**

```
fetch(url)
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error("Error:", error));
```

**Using Async/Await with Fetch:**

```
async function fetchData() {
    try {
        const response = await
fetch("https://fakestoreapi.com/products");
        const data = await response.json();
        console.log(data);
    } catch (error) {
        console.error("Error:", error);
    }
}

fetchData();
```

---

## 1. Project Structure

- `index.html` - The Home page with a link to the products page.
- `products.html` - Fetches and displays products with "Add to Cart" buttons.
- `cart.html` - Displays items added to the cart with options to remove items.
- `style.css` - CSS file for styling.
- `script.js` - JavaScript file to handle data fetching, cart management, and local storage.

---

## index.html (Home Page)

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Home - Mini E-commerce</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Welcome to Our Mini E-commerce Site!</h1>
    <p>Explore our products and add them to your cart.</p>
    <a href="products.html" class="btn">View Products</a>
</body>
</html>
```

---

**products.html (Products Page)**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Products</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Our Products</h1>
    <div id="product-list" class="product-list"></div>
    <a href="cart.html" class="btn">Go to Cart</a>
    <script src="script.js"></script>
</body>
</html>
```

---

**cart.html (Cart Page)**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Shopping Cart</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Your Cart</h1>
    <div id="cart-items" class="cart-items"></div>
    <a href="products.html" class="btn">Back to Products</a>
    <script src="script.js"></script>
</body>
</html>
```

---

**style.css (Basic Styling)**

```
body {
    font-family: Arial, sans-serif;
    display: flex;
    flex-direction: column;
    align-items: center;
    margin: 0;
    padding: 0;
}

h1 {
    color: #333;
}

.product-list, .cart-items {
    display: flex;
    flex-wrap: wrap;
    gap: 20px;
    justify-content: center;
```

```
}

.product, .cart-item {
    border: 1px solid #ddd;
    padding: 10px;
    width: 200px;
    text-align: center;
}

button, .btn {
    background-color: #007bff;
    color: white;
    padding: 10px;
    border: none;
    cursor: pointer;
    text-decoration: none;
}
```

---

**script.js (JavaScript File)**

```
// Fetch products and display them on the Products page
async function fetchProducts() {
    const response = await fetch('https://fakestoreapi.com/products');
    const products = await response.json();
    const productList = document.getElementById('product-list');

    products.forEach(product => {
        const productDiv = document.createElement('div');
        productDiv.classList.add('product');

        productDiv.innerHTML = `
            <h3>${product.title}</h3>
            <img src="${product.image}" alt="${product.title}"
width="100">
            <p>$${product.price}</p>
```

```
            <button onclick="addToCart(${product.id},
'${product.title}', ${product.price}, '${product.image}')">Add to
Cart</button>
        `;

        productList.appendChild(productDiv);
    });
}

// Add product to the cart and save to local storage
function addToCart(id, title, price, image) {
    let cart = JSON.parse(localStorage.getItem('cart')) || [];
    cart.push({ id, title, price, image });
    localStorage.setItem('cart', JSON.stringify(cart));
    alert(`${title} has been added to your cart!`);
}

// Display cart items on the Cart page
function displayCartItems() {
    const cartItems = document.getElementById('cart-items');
    let cart = JSON.parse(localStorage.getItem('cart')) || [];

    if (cart.length === 0) {
        cartItems.innerHTML = "<p>Your cart is empty.</p>";
        return;
    }

    cartItems.innerHTML = '';
    cart.forEach((item, index) => {
        const cartItemDiv = document.createElement('div');
        cartItemDiv.classList.add('cart-item');

        cartItemDiv.innerHTML = `
            <h3>${item.title}</h3>
            <img src="${item.image}" alt="${item.title}" width="100">
            <p>Price: $${item.price}</p>
            <button onclick="removeFromCart(${index})">Remove from
Cart</button>
```

```
        `;

        cartItems.appendChild(cartItemDiv);
    });
}

// Remove item from cart
function removeFromCart(index) {
    let cart = JSON.parse(localStorage.getItem('cart')) || [];
    cart.splice(index, 1);
    localStorage.setItem('cart', JSON.stringify(cart));
    displayCartItems(); // Refresh the cart display
}

// Call functions based on the page
if (document.getElementById('product-list')) {
    fetchProducts();
}

if (document.getElementById('cart-items')) {
    displayCartItems();
}
```

---

## Explanation

1. **index.html**: A simple landing page with a link to `products.html`.
2. **products.html**:
    - Calls `fetchProducts()` to fetch product data from the Fake Store API and displays them.
    - Each product has an "Add to Cart" button, which calls `addToCart()` to store the product in local storage.
3. **cart.html**:
    - Calls `displayCartItems()` to load and display items from the cart stored in local storage.
    - Each item has a "Remove from Cart" button, which calls `removeFromCart()` to remove it from the cart and update local storage.

This setup allows users to add products to their cart and manage their cart across multiple pages, using local storage to persist data.