Class 7: Current Order History Page & About Us Section

SESSION OVERVIEW:
By the end of this session, students will be able to:

- Design a Current Order Status Page to display order details dynamically using JavaScript.
- Implement features to track order details, including order number, total price, and order status.
- Simulate dynamic status updates with JavaScript by leveraging mock API calls.
- Build an About Us Section that includes:
    - Restaurant images to visually represent the business.
    - A testimonials section with auto-scroll functionality or a manual click mechanism for user navigation.
    - A responsive feedback form to capture user inputs.
- Apply HTML and CSS to create a visually appealing and responsive layout for both pages, ensuring usability on different devices.

SESSION TOPICS:

Current Order Status Page:

1. **Designing the Order History Page:**
    - Learn how to create the HTML structure for the "Order History" page, displaying order details such as order number, items, total price, and status.
2. **Track Order Details:**
    - Understand how to represent order details on the page, including visual display of the order status, item quantity, and price.
3. **Dynamic Status Updates :**
    - Use JavaScript to simulate the dynamic update of the order status without backend functionality. Explore how to use basic DOM manipulation techniques to change the status text dynamically.

About Us Page:

1. **Designing the About Us Section:**
    - Learn how to create the layout and structure for the "About Us" page, showcasing restaurant details such as the mission, team, and history.

2. **Restaurant Images and Visual Appeal:**
    - Add images that represent the restaurant’s atmosphere, dishes, and interior design to enhance the page visually.
3. **Testimonials Section:**
    - Learn how to implement a testimonials section with auto-scroll or manual click functionality to showcase customer reviews. This provides a real-world interactive element to the page.
4. **Feedback Form:**
    - Design and implement a feedback form where customers can provide their input. This section will include fields for name, email, and comments. Discuss form styling and submission.

Responsive Design:

1. **CSS Layout for Both Pages:**
    - Understand how to apply CSS Flexbox/Grid for effective layout design, ensuring the pages are visually appealing and easy to navigate on various devices.
2. **Optimizing for Mobile and Desktop:**
    - Implement responsive design practices, including media queries, to ensure the pages look great on both mobile and desktop devices.

Building Current Order History Page:

First, let's create the basic HTML structure of Current Order History Page which will be displaying all the orders till now user has made.

HTML Structure:

```
 <!-- Order Section -->
<section class="order-section">
        <h1 class="order-title">Your Order</h1>
        <!-- Table -->
        <table class="order-table">
            <thead class="order-table-head">
                <tr>
                    <th class="order-table-head-cell">Order No.</th>
                    <th class="order-table-head-cell">Order details</th>
                    <th class="order-table-head-cell">Total Price</th>
                    <th class="order-table-head-cell">Status</th>
                </tr>
            </thead>
        </table>
        <!-- Order Cards -->
        <div id="order-cards" class="order-cards-container">
            <!-- Order cards will be dynamically added here -->
        </div>
    </section>
```

Explanation:

Heading (h1):
The title "Your Order" acts as the main heading for this section.

Order Table (`<table>`):

- Displays order details in a structured format with columns:
  - **Order No.**: Unique identifier for each order.
  - **Order details**: Information about items in the order.
  - **Total Price**: The total cost of the order.
  - **Status**: Current status of the order (e.g., Pending, Delivered).
- Only the table header (`<thead>`) is defined; rows (`<tbody>`) will be dynamically added.

Order Cards Container (`<div>`):

- A container (`id="order-cards"`) designed to display order details in a card-based layout.
- Cards (alternative to the table view) will be dynamically created and appended using JavaScript.

Javascript Code:

This script dynamically generates and displays **order cards** on a webpage using the `orders` array.

```
const orders = [
            {
                id: 1,
                items: [
                    { name: "Pizza", quantity: 2 },
                    { name: "Garlic Bread", quantity: 1 },
                ],
                totalPrice: 25.99,
                status: "Delivered",
            },
            {
                id: 2,
                items: [{ name: "Burger", quantity: 1 }],
                totalPrice: 10.99,
                status: "In Transit",
            },
            {
                id: 3,
                items: [
                    { name: "Salad", quantity: 3 },
                    { name: "Juice", quantity: 2 },
                ],
                totalPrice: 18.99,
                status: "Pending",
            },
        ];

        // Function to create an order card
        function createOrderCard(order) {
            const orderCard = document.createElement("div");
            orderCard.className = "order-card";

            // Card Content
            orderCard.innerHTML = `
                <div class="order-header">
                    <span class="order-number">Order #${order.id}</span>
                </div>
                <div class="order-details">
                    ${order.items
                        .map(
                            (item) => `
                        <div class="order-details-item">
                            <span>${item.name}</span>
                            <span>x${item.quantity}</span>
                        </div>
                    `
                        )
```

```
                        .join("")}
                    </div>
                    <div class="order-total">Total:
$${order.totalPrice.toFixed(2)}</div>
                    <span class="order-status status-${order.status
                        .toLowerCase()
                        .replace(" ", "-")}">
                        ${order.status}
                    </span>
                `;

            return orderCard;
        }

          // Add order cards to the container
        const orderCardsContainer = document.getElementById("order-cards");
        if (orders.length > 0) {
            orders.forEach((order) => {
                const orderCard = createOrderCard(order);
                orderCardsContainer.appendChild(orderCard);
            });
        } else {
            const noOrdersMessage = document.createElement("div");
            noOrdersMessage.className = "no-orders";
            noOrdersMessage.textContent = "You have not ordered any food yet";
            orderCardsContainer.appendChild(noOrdersMessage);
        }
```

Explanation:

1. Orders Array

- Contains a list of **order objects** with properties:
    - `id`: The unique identifier for the order.
    - `items`: An array of items in the order, each with a `name` and `quantity`.
    - `totalPrice`: The total cost of the order.
    - `status`: The current status of the order (e.g., "Delivered", "In Transit").

2. `createOrderCard` Function

- **Purpose**: Creates an HTML card for each order.
- **Process**:
    1. Creates a `div` element with the class `order-card`.
    2. Dynamically fills the card with:
        - **Order Number**: Displayed in the header.
        - **Order Items**: Loops through the `items` array, formatting the name and quantity for each item.
        - **Total Price**: Displays the total cost of the order.
        - **Order Status**: Shows the order's status, dynamically adding a class (`status-delivered`, `status-in-transit`, etc.) for styling.

---

3. Adding Cards to the Page

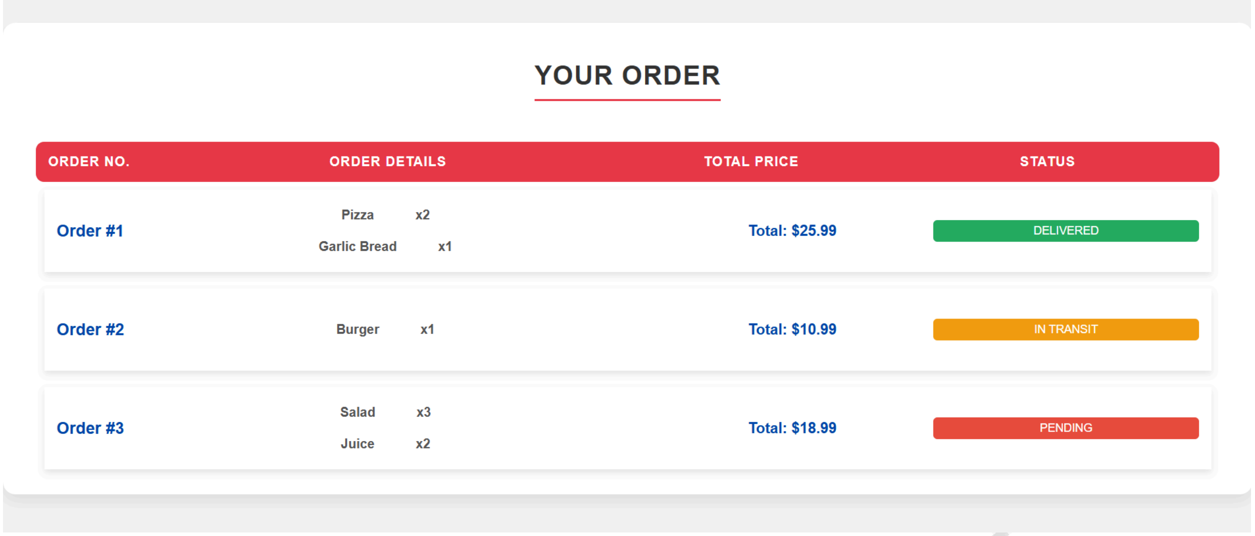
- **Container**: The order cards are appended to the `<div>` with `id="order-cards"`.
- **Logic**:
    - If `orders` contains data:
        - Loops through each order, creates a card using `createOrderCard`, and appends it to the container.
    - If `orders` is empty:
        - Displays a fallback message: "You have not ordered any food yet."

CSS Style:

```css
/* Order Section Styling */
.order-section {
    margin: 50px auto;
    /* max-width: 800px; */
    text-align: center;
    background: #fff;
    padding: 20px 40px;
    border-radius: 15px;
    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);
}

.order-title {
    font-size: 32px;
    margin-bottom: 30px;
    color: #333;
    font-weight: bold;
    text-transform: uppercase;
    letter-spacing: 1px;
    text-align: center;
    border-bottom: 3px solid #e63946;
    display: inline-block;
    padding-bottom: 10px;
}

/* Table Styling */
.order-table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
    background: #f9f9f9;
    border-radius: 10px;
    overflow: hidden;
}

thead {
    background:#e63946;
    color: #fff;
    text-transform: uppercase;
}

.order-table-head-cell {
    padding: 15px;
    font-size: 16px;
    font-weight: bold;
    text-align: left;
    letter-spacing: 1px;
}


/* Order Cards Container */
.order-cards-container {
    display: flex;
    flex-direction: column;
    gap: 20px;
    padding: 10px;
}

/* Order Card (Landscape Mode) */
.order-card {
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    align-items: center;
    background: #fff;
    /* width: 90%; */
    min-height: 70px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    padding: 15px;
    transition: transform 0.3s;
    gap: 15px;
}

.order-card:hover {
    transform: scale(1.02);
}

/* Order Number */
.order-number {
    font-size: 20px;
    font-weight: bold;
    color: #004aad;
    flex: 1;
}

/* Order Details Section */
.order-details {
    display: flex;
    flex-direction: column;
    gap: 20px;
    flex: 2;
    color: #555;
```

```
}

.order-details-item {
    display: flex;
    justify-content: center;
    gap: 50px;
    align-items: center;
    font-size: 16px;
    color: #555;
   font-weight: 700;
    margin-right:20px;

}

/* Total Price Section */
.order-total {
    font-size: 18px;
    font-weight: bold;
    color: #004aad;
    justify-self: flex-start;
    /* text-align: center; */
    flex: 1;
}

/* Order Status */
.order-status {
    padding: 5px 5px;
    border-radius: 5px;
    font-size: 14px;
    text-transform: uppercase;
    text-align: center;
    align-self: center;
    justify-self: center;
    /* width: 50%; */
    color: #fff;
    flex: 1;
}

.status-delivered {
    background-color: #27ae60;
}

.status-in-transit {
    background-color: #f39c12;
}

.status-pending {
    background-color: #e74c3c;
}

.no-orders{
    margin: 20px;
    text-transform: uppercase;
    font-weight: bolder;
}
```

Output:



YOUR ORDER

ORDER NO.	ORDER DETAILS	TOTAL PRICE	STATUS
Order #1	Pizza x2 Garlic Bread x1	Total: $25.99	DELIVERED
Order #2	Burger x1	Total: $10.99	IN TRANSIT
Order #3	Salad x3 Juice x2	Total: $18.99	PENDING

Designing the About Us Page:

Let's create the basic HTML structure of About Us page of the Restaurant Menu App so that users can learn more about the restaurant, its values, and its offerings while enjoying an engaging and visually appealing experience.

HTML Structure:

```html
   <!-- About us page -->
    <section class="restaurant-overview">
        <div class="restaurant-info">
            <h3 class="restaurant-name">Restaurant Name</h3>
            <p class="restaurant-established">Established in 1999</p>
        </div>
    </section>
    <section class="about-section">
        <div class="about-restaurant">
            <div class="about-image-container">
                <img src="https://placehold.co/300x400" alt="About Restaurant"
class="about-image">
            </div>
            <div class="about-text">
                <h3>About Our Restaurant</h3>
                <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
Beatae tempora maiores laboriosam culpa ullam consequatur sit? Minus autem
quibusdam, eveniet incidunt, natus repudiandae cumque inventore veritatis
deleniti suscipit tenetur eligendi harum fugiat odio est totam, eum et officiis
aliquid quam!</p>
            </div>
        </div>

        <div class="about-restaurant">
            <div class="about-text">
                <h3>About Our Founder</h3>
                <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
Beatae tempora maiores laboriosam culpa ullam consequatur sit? Minus autem
quibusdam, eveniet incidunt, natus repudiandae cumque inventore veritatis
deleniti suscipit tenetur eligendi harum fugiat odio est totam, eum et officiis
aliquid quam!</p>
            </div>
            <div class="about-image-container">
                <img src="https://placehold.co/300x400" alt="Founder"
class="about-image">
            </div>
        </div>
        <div class="about-restaurant">
            <div class="about-image-container">
                <img src="https://placehold.co/300x400" alt="Speciality"
class="about-image">
            </div>
            <div class="about-text">
                <h3>Our Speciality</h3>
                <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
Beatae tempora maiores laboriosam culpa ullam consequatur sit? Minus autem
quibusdam, eveniet incidunt, natus repudiandae cumque inventore veritatis
deleniti suscipit tenetur eligendi harum fugiat odio est totam, eum et officiis
aliquid quam!</p>
          <button onclick="visitMenuHtml" class="explore-menu-btn">Explore our
menu</button>
            </div>
        </div>
    </section>
    <!-- Testimonial Section -->
    <section class="testimonial-section">
        <h2 class="testimonial-title">What Our Customers Say</h2>
        <div class="testimonial-carousel">
            <div class="carousel-track">
                <div class="testimonial-item">
                    <div class="testimonial-photo">
                        <img src="https://placehold.co/100x100?text=JD" alt="John
Doe">
                    </div>
                    <p class="testimonial-text">"The food here is absolutely
amazing! Highly recommend their specialty dishes. A delightful experience!"</p>
                    <h4 class="customer-name">John Doe</h4>
                    <div class="customer-rating">⭐⭐⭐⭐⭐</div>
                </div>
                <div class="testimonial-item">
                    <div class="testimonial-photo">
                        <img src="https://placehold.co/100x100?text=JS" alt="Jane
Smith">
                    </div>
                    <p class="testimonial-text">"Excellent service and
mouthwatering dishes. This restaurant never disappoints!"</p>
                    <h4 class="customer-name">Jane Smith</h4>
                    <div class="customer-rating">⭐⭐⭐⭐⭐</div>
                </div>
                <div class="testimonial-item">
                    <div class="testimonial-photo">
                        <img src="https://placehold.co/100x100?text=ML" alt="Mark
Lee">
                    </div>
                    <p class="testimonial-text">"The ambiance is perfect, and the
food quality is top-notch. Definitely a must-visit!"</p>
                    <h4 class="customer-name">Mark Lee</h4>
                    <div class="customer-rating">⭐⭐⭐⭐⭐</div>
                </div>
                <div class="testimonial-item">
                    <div class="testimonial-photo">
                        <img src="https://placehold.co/100x100?text=EM"
alt="Emily Martin">
                    </div>
                    <p class="testimonial-text">"I've never had a bad experience
here. The staff is friendly, and the food is always delicious!"</p>
                    <h4 class="customer-name">Emily Martin</h4>
                    <div class="customer-rating">⭐⭐⭐⭐⭐</div>
                </div>
                <div class="testimonial-item">
                    <div class="testimonial-photo">
                        <img src="https://placehold.co/100x100?text=SB"
alt="Sarah Brown">
                    </div>
                    <p class="testimonial-text">"This restaurant is a gem! The
atmosphere is cozy, and the dishes are creative and tasty."</p>
                    <h4 class="customer-name">Sarah Brown</h4>
                    <div class="customer-rating">⭐⭐⭐⭐⭐</div>
                </div>
            </div>
        </div>
    </section>
   <!-- Feedback Section -->
```

```
    <section class="feedback-section">
            <div class="feedback-heading">
                <h2 class="feedback-title">We Value Your Feedback</h2>
                <p class="feedback-description">
                    Your thoughts help us improve. Please take a moment to share
your feedback with us.
                </p>
            </div>
                <form class="feedback-form">
                    <div class="form-group">
                        <label for="name">Your Name</label>
                        <input type="text" id="name" placeholder="Enter your
name" required>
                    </div>
                    <div class="form-group">
                        <label for="email">Your Email</label>
                        <input type="email" id="email" placeholder="Enter your
email" required>
                    </div>
                    <div class="form-group">
                        <label for="feedback">Your Feedback</label>
                    <textarea id="feedback" placeholder="Share your experience"
rows="5" required></textarea>
                </div>
                <button type="submit" class="submit-button">Submit
Feedback</button>
            </form>
    </section>
```

Explanation:

This HTML structure defines an **About Us page** for a restaurant website, including:

1. **Restaurant Overview**: A brief introduction featuring the restaurant's name and establishment year.
2. **About Sections**:
    - **About the Restaurant**: Includes an image and a description of the restaurant's story.
    - **About the Founder**: Details about the founder with an image and description.

- **Specialties**: Highlights the restaurant's specialty dishes with a call-to-action button to explore the menu.

3. **Testimonials Section**: A carousel showcasing customer reviews, their photos, names, and ratings, creating a trustworthy impression.
4. **Feedback Section**: A form where users can provide their name, email, and feedback about their experience, helping improve services.

Javascript Code:

```
    // script.js
    let currentIndex = 0;
// Get the carousel track and items
const carouselTrack = document.querySelector('.carousel-track');
const testimonialItems = document.querySelectorAll('.testimonial-item');
// Auto-scroll every 4 seconds
function moveCarousel() {
    currentIndex++;
    if (currentIndex >= testimonialItems.length) {
        currentIndex = 0;
    }
    const offset = -currentIndex * 100; // Each item is 100% wide
    carouselTrack.style.transform = `translateX(${offset}%)`;
}
function visitMenuHtml() {
    window.location.href = "ourmenu.html";
}
// Start the carousel
setInterval(moveCarousel, 2000);
```

Explanation

This script manages the **testimonial carousel** and a navigation function:

1. **Testimonial Carousel**:
   - Automatically scrolls through testimonials every 2 seconds using `setInterval()`.
   - Adjusts the carousel's position by applying a `translateX` transform based on the current testimonial index.
2. **Navigation to Menu**:
   - The `visitMenuHtml()` function redirects users to the menu page (`ourmenu.html`) when triggered.

CSS Style:

```css
/* Restaurant Overview */
.restaurant-overview {
    text-align: center;
    background-image:
url("https://placehold.co/1500x800?text=Restaurant%20Image");
    background-position: center;
    background-size: cover;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 90vh;
}
.restaurant-info{
    background-color: #ffffff;
    height: 150px;
    width: 500px;
    margin-top: 40px;
    display: flex;
    align-items: center;
    flex-direction: column;
    opacity: 0.8;
    justify-content: center;
}
.restaurant-name {
    font-size: 28px;
    font-weight: bold;
    color: #555;
}
.restaurant-established {
    font-size: 16px;
    color: #666;
}
/* About Section */
.about-section {
    padding: 20px;
}
.about-restaurant {
    display: flex;
    justify-content: space-around;
    align-items: center;
    margin-bottom: 30px;
    background-color: rgb(255, 255, 255);
    padding: 50px;
    gap: 30px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
.about-image{
    width: 400px;
    height: 300px;
    object-fit: cover;
    border-radius: 8px;
}
.about-text {
    display: flex;
justify-self: flex-start;
align-self: flex-start;
flex-direction: column;
    /* flex: 1; */
    margin-left: 40px;
    /* font-family: ; */
    /* width: 0%; */
    /* text: wrap 2px;  */
}
.about-text h3{
    text-align: center;
    font-size: 35px;
}
.about-text p {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    color:goldenrod
}
.explore-menu-btn{background-image: linear-gradient(to right, #ffb347 0%, #ffcc33
51%, #ffb347  100%)}
.explore-menu-btn {
   margin: 10px;
   padding: 15px 45px;
   align-self: center;
  border: none;
   text-align: center;
   text-transform: uppercase;
   transition: 0.5s;
   background-size: 200% auto;
   color: rgb(12, 11, 11);
   box-shadow: 0 0 20px #eee;
   border-radius: 10px;
   display: block;
 }

 .explore-menu-btn:hover {
   background-position: right center; /* change the direction of the change here
*/
   color: #000000;
   text-decoration: none;
   cursor: pointer;
 }
/* Testimonial Section */
.testimonial-section {
    padding: 50px 20px;
    background-color: #ffffff;
    text-align: center;
}

.testimonial-title {
    font-size: 28px;
    font-weight: bold;
    margin-bottom: 30px;
    color: #333;
    text-transform: uppercase;
    letter-spacing: 1.5px;
}

.testimonial-carousel {
    overflow: hidden;
    position: relative;
    /* width: 90%; */
```

```css
    max-width: 800px;
    margin: 0 auto;
}

.carousel-track {
    display: flex;
    width: 90%;
    transition: transform 0.5s ease-in-out;
}

.testimonial-item {
    min-width: 95%;
    padding: 20px;
    background: #f5f5f5;  /* fallback for old browsers */
    border-radius: 15px;
    box-shadow: 0 5px 15px rgba(222, 222, 222, 0.2);
    margin: 10px;
    text-align: center;
}
.testimonial-photo {
    margin: 0 auto 15px auto;
    width: 100px;
    height: 100px;
    border-radius: 50%;
    overflow: hidden;
    border: 3px solid #e63946;
}
.testimonial-photo img {
    width: 100%;
    height: 100%;
    object-fit: cover;
}
.testimonial-text {
    font-size: 18px;
    font-style: italic;
    margin-bottom: 20px;
    color: #555;
}
.customer-name {
    font-size: 20px;
    font-weight: bold;
    color: #004aad;
}
.customer-rating {
    font-size: 18px;
    color: #ff9900;
    margin-top: 10px;
}
/* Smoother transition */
.carousel-track {
    transition: transform 0.5s ease-in-out;
}
/* Feedback Section */
.feedback-section {
    padding: 25px 20px;
    display: flex;
    justify-content: space-around;
    /* background-color: rgb(246, 237, 237); */
    /* min-height: 100vh; */
}
.feedback-form-container {
    /* max-width: 500px; */
    padding: 30px;
    display: flex;
    border-radius: 15px;
    justify-content: space-between;
    text-align: center;
}
.feedback-title {
    font-size: 28px;
    color: #e63946;
    margin-bottom: 10px;
    text-transform: uppercase;
    font-weight: bold;
}
.feedback-description {
    font-size: 16px;
    color: #666;
    margin-bottom: 20px;
}
.feedback-heading{
    justify-content: center;
    /* margin-top: 20px; */
    text-align: center;
    align-self: center;
}
/* Form Styles */
.feedback-form {
    display: flex;
    background: #fff;
    box-shadow: 0 5px 15px rgba(222, 222, 222, 0.2);
/* width: 60%; */
width: 600px;
    padding: 20px;
    border-radius: 20px;
    justify-content: center;
    align-items: center;

    flex-direction: column;
    gap: 20px;
}
.form-group {
    align-self: flex-start;
    width: 100%;
    text-align: left;
}
label {
    font-size: 14px;
    font-weight: bold;
    margin-bottom: 5px;
    display: block;
    color: #555;
}
input, textarea {
    width: 90%;
    padding: 10px;
    border: 1px solid #ddd;
```

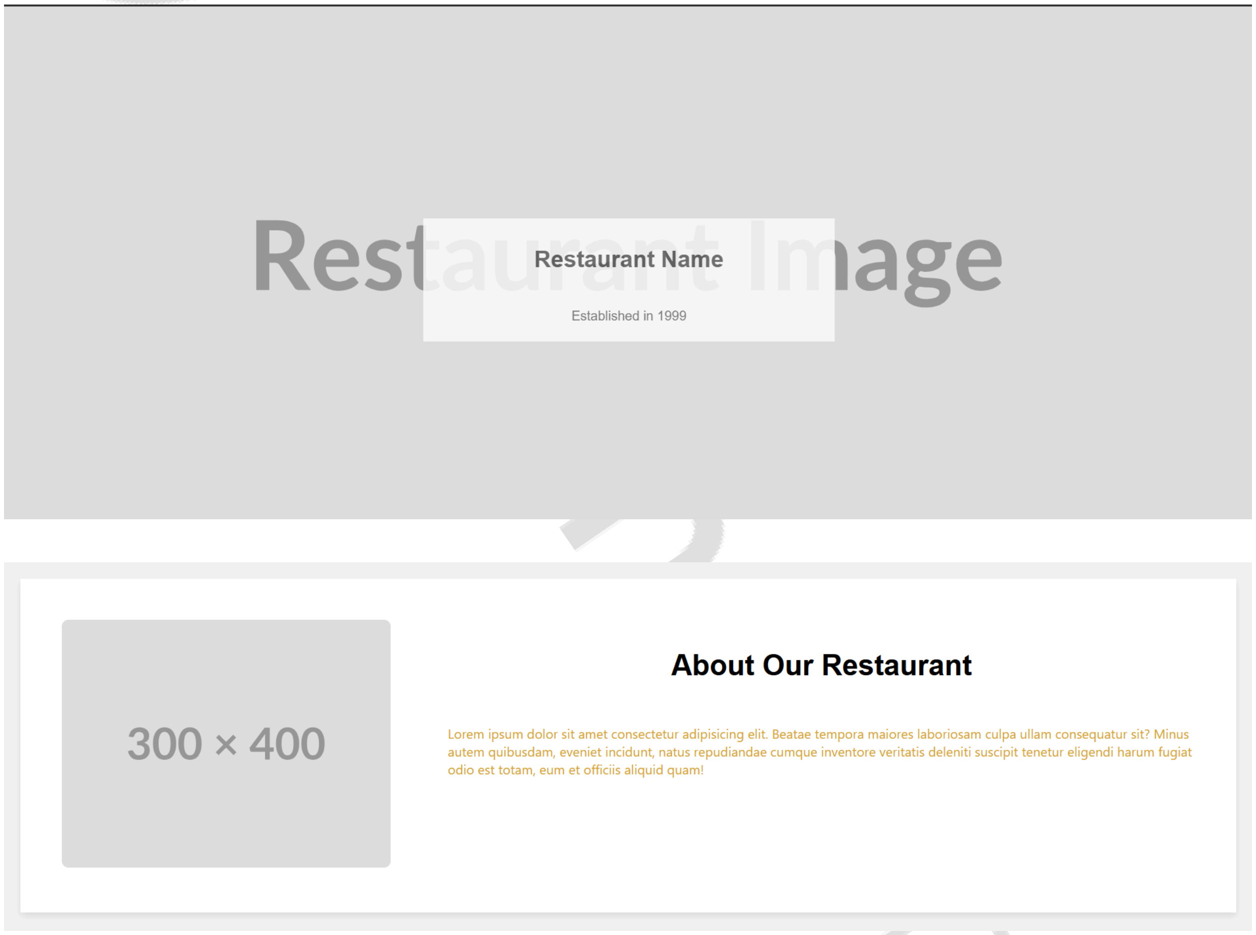
```css
    border-radius: 8px;
    font-size: 16px;
    background: #f9f9f9;
    transition: all 0.3s ease;
}
input:focus, textarea:focus {
    border-color: #e63946;
    box-shadow: 0 0 5px rgba(230, 57, 70, 0.4);
}
textarea{
    max-width:  90%;
    min-width: 90%;
    min-height: 100px;
}
/* Button Styles */
.submit-button {
    background: #e63946;
    color: #fff;
    border: none;
    padding: 12px;
    font-size: 16px;
    font-weight: bold;
    border-radius: 50px;
    margin-bottom: 20px;
    cursor: pointer;
    transition: background 0.3s ease;
    text-transform: uppercase;
}
.submit-button:hover {
    background: #c5313e;
}
```

Note:

1. You can also use this image link as the placeholder for the restaurant image to enhance the visual appeal of your "About Us" page. This image offers a warm and inviting aesthetic, perfect for showcasing a restaurant's ambiance.

2. To make the "About Us" page a complete webpage, consider adding a header and footer with relevant navigation links and contact details. This will improve the structure and usability of your page.

Output:

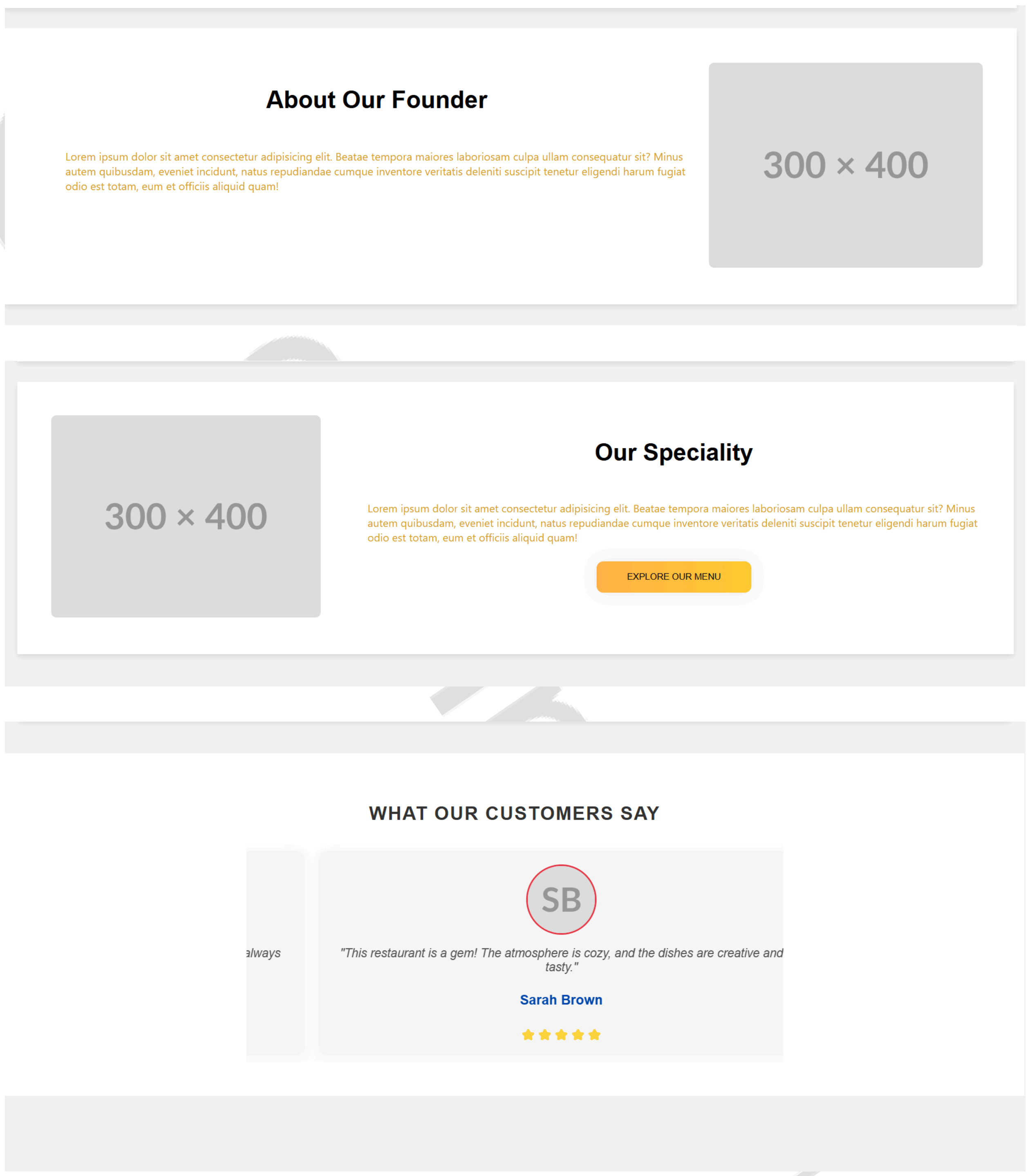


Restaurant Name

Established in 1999

About Our Restaurant

Lorem ipsum dolor sit amet consectetur adipisicing elit. Beatae tempora maiores laboriosam culpa ullam consequatur sit? Minus autem quibusdam, eveniet incidunt, natus repudiandae cumque inventore veritatis deleniti suscipit tenetur eligendi harum fugiat odio est totam, eum et officiis aliquid quam!



About Our Founder

Lorem ipsum dolor sit amet consectetur adipisicing elit. Beatae tempora maiores laboriosam culpa ullam consequatur sit? Minus autem quibusdam, eveniet incidunt, natus repudiandae cumque inventore veritatis deleniti suscipit tenetur eligendi harum fugiat odio est totam, eum et officiis aliquid quam!

300 × 400

300 × 400

Our Speciality

Lorem ipsum dolor sit amet consectetur adipisicing elit. Beatae tempora maiores laboriosam culpa ullam consequatur sit? Minus autem quibusdam, eveniet incidunt, natus repudiandae cumque inventore veritatis deleniti suscipit tenetur eligendi harum fugiat odio est totam, eum et officiis aliquid quam!

WHAT OUR CUSTOMERS SAY

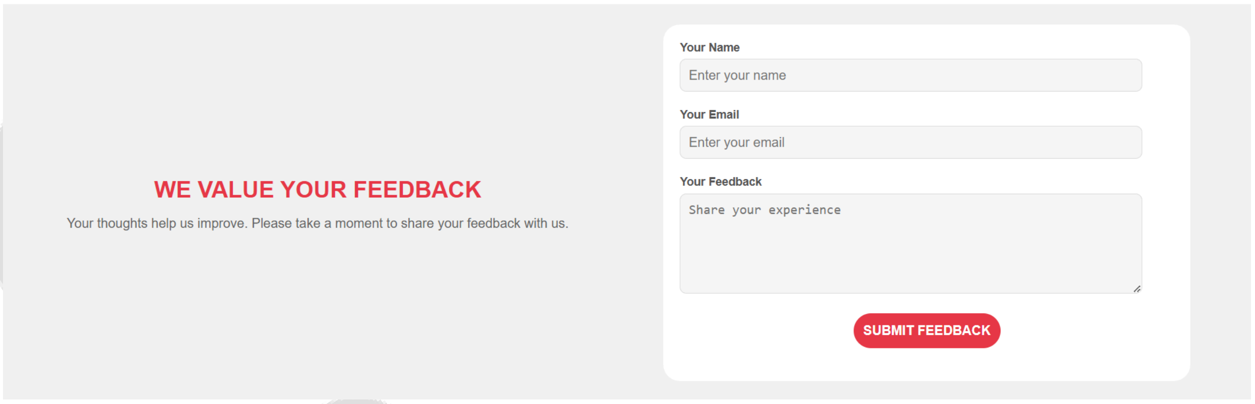
always

SB

"This restaurant is a gem! The atmosphere is cozy, and the dishes are creative and tasty."

Sarah Brown

★★★★★


WE VALUE YOUR FEEDBACK

Your thoughts help us improve. Please take a moment to share your feedback with us.

Your Name

Enter your name

Your Email

Enter your email

Your Feedback

Share your experience

Final Output:

[Aboutus.html]

[Aboutstyle.css]

[Order.html]

[Orderstyle.css]

Concept Indepths:

High-Order Functions (HOFs)

Introduction to High-Order Functions

In JavaScript, **functions are first-class citizens**, which means they can be treated like any other variable. You can pass functions as arguments, return them from other functions, and even assign them to variables. This capability enables **High-Order Functions (HOFs)**, which are functions that **either take one or more functions as arguments** or **return a function** as a result.

A **High-Order Function (HOF)** is a function that satisfies either of the following two conditions:

1. **Takes one or more functions as arguments.**
2. **Returns a function as a result.**

This concept is deeply rooted in **functional programming**, where functions are treated as data that can be manipulated in powerful ways.

Why High-Order Functions Are Important

High-order functions are crucial because they allow for the abstraction of behavior, making the code more modular, flexible, and reusable. Instead of repeating code or defining functions for specific tasks, you can define **generic functions** and pass different logic (functions) to them.

Benefits of High-Order Functions:

1. **Reusability**: A high-order function can work with any function, making it reusable in multiple places.
2. **Modularity**: You can abstract specific behaviors into smaller, reusable functions.
3. **Conciseness**: High-order functions like `map()`, `filter()`, and `reduce()` simplify code, avoiding the need for writing repetitive loops.
4. **Flexibility**: You can dynamically apply functions, creating more expressive and flexible code.
5. **Declarative Programming**: HOFs enable a declarative approach, where you describe what to do (the action) rather than how to do it (step-by-step logic).

Examples of High-Order Functions

Let's dive into some concrete examples to better understand how high-order functions work.

1. A Function that Takes Another Function as an Argument

In this case, the high-order function accepts a **callback function** as an argument and uses it to perform a specific action on data.

```
Example: forEach()

const numbers = [1, 2, 3, 4];
// Applying the forEach method with a function as an argument
numbers.forEach((num) => {
    console.log(num * 2); // Output: 2, 4, 6, 8
});
```

Here:

- **`forEach()`** is a high-order function that takes a function as an argument.

- The function `(num) => { console.log(num * 2); }` is applied to each element of the `numbers` array, and the result is printed.
- **`forEach`** is commonly used for executing a given function on each element of an array.

---

2. A Function that Returns Another Function

Here, the high-order function returns a new function that can be invoked later with different arguments.

Example: Function Returning Another Function

```
function multiplyBy(multiplier) {
    return function (number) {
        return number * multiplier;
    };
}

const double = multiplyBy(2); // Returns a function that doubles a number
console.log(double(5)); // Output: 10
```

Here:

- **`multiplyBy`** is a high-order function that returns another function.
- The returned function takes a number and multiplies it by the `multiplier`.
- The value of `multiplier` is preserved when the new function is created (`double`), allowing us to reuse the logic to **double** any number.

Built-in High-Order Functions in JavaScript

JavaScript provides several built-in high-order functions that make it easy to work with arrays and other data structures. These functions are frequently used in day-to-day JavaScript development.

1. `map()`

The **`map()`** function allows you to **transform** each element of an array based on the provided function, returning a new array with the transformed elements.

Example:

```
const numbers = [1, 2, 3, 4];
const squares = numbers.map((num) => num * num);
console.log(squares); // Output: [1, 4, 9, 16]
```

Here:

- **`map()`** takes a function `(num) => num * num` and applies it to each element of the `numbers` array.
- A new array `squares` is returned with the squared values of the original numbers.

2. `filter()`

The **`filter()`** function allows you to **filter** out elements that do not meet a certain condition, returning a new array with only the elements that satisfy the condition.

Example:

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter((num) => num % 2 === 0);
console.log(evenNumbers); // Output: [2, 4]
```

Here:

- **`filter()`** takes a function `(num) => num % 2 === 0` and applies it to each element of the `numbers` array.
- Only even numbers (2, 4) are included in the resulting array `evenNumbers`.

3. `reduce()`

The **`reduce()`** function is used to **reduce** an array to a single value by applying a **callback function** that takes an accumulator and the current value of the array.

Example:

```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce((total, num) => total + num, 0);
```

```
console.log(sum); // Output: 10
```

Here:

- `reduce()` takes a callback function `(total, num) => total + num` and a **starting value** of `0`.
- The callback function is executed for each element, accumulating the total sum of the numbers.

4. `sort()`

The `sort()` function is used to sort the elements of an array based on a **comparison function**.

Example:

```
const numbers = [5, 3, 8, 1];
numbers.sort((a, b) => a - b); // Ascending order
console.log(numbers); // Output: [1, 3, 5, 8]
```

Here:

- `sort()` sorts the `numbers` array in ascending order using the comparison function `(a, b) => a - b`.

Creating Custom High-Order Functions

You can create your own high-order functions to add additional functionality to your applications. Let's create a custom high-order function that adds logging to any function:

```
function withLogging(action) {
    return function (...args) {
        console.log(`Action started`);
        const result = action(...args);
        console.log(`Action finished`);
        return result;
    };
}

const add = (a, b) => a + b;
const loggedAdd = withLogging(add);

console.log(loggedAdd(5, 3)); // Output: Action started, Action finished, 8
```

Here:

- **`withLogging`** is a high-order function that takes another function (`action`) as an argument and returns a new function.
- The new function logs the action before and after calling the original function, providing additional behavior dynamically.

Chaining High-Order Functions

One of the powerful features of high-order functions is **chaining**, where you can use multiple functions one after another. Functions like `map()`, `filter()`, and `reduce()` are designed to be chained together to perform complex operations on data.

Example: Chaining `map()`, `filter()`, and `reduce()`

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9];

// Find the sum of squares of even numbers
const result = numbers
    .filter((num) => num % 2 === 0) // Filter even numbers
    .map((num) => num * num) // Square the numbers
    .reduce((total, num) => total + num, 0); // Sum the squares

console.log(result); // Output: 120
```

Here:

- **`filter()`** filters out odd numbers.
- **`map()`** squares the even numbers.

- `reduce()` adds the squared values to get the total.

This allows you to perform multiple operations in a concise and readable manner.

Important Concepts and Key Points

1. **Functions as First-Class Citizens**: In JavaScript, functions are treated like any other data type, which makes high-order functions possible.
2. **Pure Functions**: It's common to use **pure functions** (functions without side effects) as arguments to high-order functions. This makes the code predictable and easier to test.
3. **Functional Programming**: High-order functions are one of the core features of **functional programming**, where the focus is on composing functions rather than relying on mutable states.

Industry Trends in High-Order Functions

1. **Functional Programming (FP) Popularity** Functional programming principles, including the use of high-order functions, are gaining traction due to their ability to handle **complex state management** and **side-effect-free operations**. Popular frameworks like **React**, **Redux**, and **Vue.js** embrace these principles, making HOFs central to modern web development.
    - **React** utilizes HOFs such as `map()`, `filter()`, and `reduce()` for state updates and data transformations. This is especially prominent when managing arrays or lists of components.
    - **Redux** encourages the use of **pure functions** and **reducers**, which are often high-order functions that return new state objects based on the previous state and actions.
2. **Declarative Programming** High-order functions promote a **declarative programming style** over an imperative one. In declarative programming, you express what should happen (the action) without specifying how it should happen step by step. With HOFs like `map()`, `filter()`, and `reduce()`, developers can express complex transformations in a concise, readable way.
    - Example: Using `.map()` in JavaScript provides a **declarative way** to transform an array, rather than manually looping through it and applying changes.
3. **Immutability and Pure Functions** The trend of **immutability** (ensuring that data is not modified directly) fits perfectly with high-order functions. HOFs encourage the use of **pure functions** (functions that don't modify external states) by design. This enhances predictability, maintainability, and **debugging**.
    - For example, using `map()` to return a new array without modifying the original one is a best practice when handling **state transformations** in libraries like **React**.

4. **Asynchronous JavaScript and HOFs** With the rise of **asynchronous programming** in JavaScript (especially with **Promises** and **async/await**), HOFs are increasingly used to manage **async behavior** in a more declarative manner. High-order functions such as `map()` and `filter()` are commonly used in **Promise chaining** to manage lists of async operations.
    - For instance, you can use `map()` to create an array of **Promises** and then use `Promise.all()` to wait for all promises to resolve.
5. **Serverless and Functional Programming** Serverless computing and **cloud-native architectures** are rapidly adopting functional programming paradigms. As these services often require stateless and **function-based** workflows, high-order functions provide a simple, composable way to implement these stateless behaviors.
    - Serverless functions (like AWS Lambda or Google Cloud Functions) often rely on high-order functions to transform data or compose various API responses.

Follow tech blogs like [Smashing Magazine](), and [Dev.to]() to stay current with industry trends