**Exercise 1: Sum of Array Elements**

**Task:** Calculate the sum of all elements in an array.

**Solution:**

const numbers = [1, 2, 3, 4, 5];

let sum = 0;

for (let i = 0; i < numbers.length; i++) {

sum += numbers[i];

}

console.log(sum); // Output: 15

**Exercise 2: Find the Maximum Number in an Array**

**Task:** Find the largest number in an array.

**Solution:**

const numbers = [10, 5, 100, 0, 55];

let max = numbers[0];

for (let i = 1; i < numbers.length; i++) {

if (numbers[i] > max) {

max = numbers[i];

}

}

console.log(max); // Output: 100

**Exercise 3: Remove Duplicates from an Array**

**Task:** Remove duplicate values from an array.

**Solution:**

const numbers = [1, 2, 2, 3, 4, 4, 5];

let uniqueNumbers = [];

for (let i = 0; i < numbers.length; i++) {

if (uniqueNumbers.indexOf(numbers[i]) === -1) {

uniqueNumbers.push(numbers[i]);

}

}

console.log(uniqueNumbers); // Output: [1, 2, 3, 4, 5]

**Exercise 4: Reverse an Array**

**Task:** Reverse the elements of an array.

**Solution:**

const numbers = [1, 2, 3, 4, 5];

let reversed = [];

for (let i = numbers.length - 1; i >= 0; i--) {

reversed.push(numbers[i]);

}

console.log(reversed); // Output: [5, 4, 3, 2, 1]

**Exercise 5: Find All Even Numbers in an Array**

**Task:** Find all even numbers in an array.

**Solution:**

const numbers = [1, 2, 3, 4, 5, 6];

let evenNumbers = [];

for (let i = 0; i < numbers.length; i++) {

if (numbers[i] % 2 === 0) {

evenNumbers.push(numbers[i]);

}

}

console.log(evenNumbers); // Output: [2, 4, 6]

**Exercise 6: Merge Two Arrays**

**Task:** Merge two arrays into one array without duplicates.

**Solution:**

const array1 = [1, 2, 3];

const array2 = [3, 4, 5];

let merged = [];

for (let i = 0; i < array1.length; i++) {

if (merged.indexOf(array1[i]) === -1) {

merged.push(array1[i]);

}

}

for (let i = 0; i < array2.length; i++) {

if (merged.indexOf(array2[i]) === -1) {

merged.push(array2[i]);

}

}

console.log(merged); // Output: [1, 2, 3, 4, 5]

**Exercise 7: Find the Index of an Element**

**Task:** Find the index of a specific element in an array.

**Solution:**

const fruits = ['apple', 'banana', 'cherry'];

const searchElement = 'banana';

let index = -1;

for (let i = 0; i < fruits.length; i++) {

if (fruits[i] === searchElement) {

index = i;

break;

}

}

console.log(index); // Output: 1

**Exercise 8: Sum of Array Elements**

**Task:** Write a function that takes an array of numbers and returns the sum of all the elements.

**Solution:**

function sumArray(arr) {

return arr.reduce((acc, num) => acc + num, 0);

}

// Example usage:

const numbers = [1, 2, 3, 4, 5];

console.log(sumArray(numbers)); // Output: 15

**Exercise 9: Find the Maximum Number in an Array**

**Task:** Write a function that returns the largest number from an array of numbers.

**Solution:**

function findMax(arr) {

return Math.max(...arr);

}

// Example usage:

const numbers = [10, 5, 100, 0, 55];

console.log(findMax(numbers)); // Output: 100

**Exercise 10: Remove Duplicates from an Array**

**Task:** Write a function that removes duplicate values from an array and returns a new array with unique values.

**Solution:**

function removeDuplicates(arr) {

return [...new Set(arr)];

}

// Example usage:

const numbers = [1, 2, 2, 3, 4, 4, 5];

console.log(removeDuplicates(numbers)); // Output: [1, 2, 3, 4, 5]

**Exercise 11: Reverse an Array**

**Task:** Write a function that reverses the elements of an array.

**Solution:**

function reverseArray(arr) {

return arr.slice().reverse();

}

// Example usage:

const numbers = [1, 2, 3, 4, 5];

console.log(reverseArray(numbers)); // Output: [5, 4, 3, 2, 1]

**Exercise 12: Find All Even Numbers in an Array**

**Task:** Write a function that returns an array containing only the even numbers from the input array.

**Solution:**

function filterEvenNumbers(arr) {

return arr.filter(num => num % 2 === 0);

}

// Example usage:

const numbers = [1, 2, 3, 4, 5, 6];

console.log(filterEvenNumbers(numbers)); // Output: [2, 4, 6]

**Exercise 13: Merge Two Arrays**

**Task:** Write a function that takes two arrays and merges them into one array without duplicates.

**Solution:**

function mergeArrays(arr1, arr2) {

return [...new Set([...arr1, ...arr2])];

}

// Example usage:

const array1 = [1, 2, 3];

const array2 = [3, 4, 5];

console.log(mergeArrays(array1, array2)); // Output: [1, 2, 3, 4, 5]

**Exercise 14: Find the Index of an Element**

**Task:** Write a function that returns the index of a specific element in an array. If the element is not found, return -1.

**Solution:**

function findIndex(arr, element) {

return arr.indexOf(element);

}

// Example usage:

const fruits = ['apple', 'banana', 'cherry'];

console.log(findIndex(fruits, 'banana')); // Output: 1

console.log(findIndex(fruits, 'orange')); // Output: -1

**Exercise 15: Dynamic List Creation**

**Task:** Create a form that allows users to add items to a list. Each item should be displayed in a styled list below the form.

**HTML & CSS:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Dynamic List Creation</title>

<style>

body {

font-family: Arial, sans-serif;

}

.container {

width: 50%;

margin: 0 auto;

padding: 20px;

border: 1px solid #ddd;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

input[type="text"] {

padding: 10px;

width: 80%;

margin-right: 10px;

}

button {

padding: 10px 20px;

background-color: #007bff;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

ul {

list-style-type: none;

padding: 0;

}

li {

padding: 10px;

border-bottom: 1px solid #ddd;

}

</style>

</head>

<body>

<div class="container">

<h1>Add Items to List</h1>

<form id="itemForm">

<input type="text" id="itemInput" placeholder="Enter item">

<button type="submit">Add Item</button>

</form>

<ul id="itemList"></ul>

</div>

<script>

const form = document.getElementById('itemForm');

const input = document.getElementById('itemInput');

const itemList = document.getElementById('itemList');

const items = [];

form.addEventListener('submit', function(event) {

event.preventDefault();

const newItem = input.value.trim();

if (newItem && !items.includes(newItem)) {

items.push(newItem);

const li = document.createElement('li');

li.textContent = newItem;

itemList.appendChild(li);

input.value = '';

}

});

</script>

</body>

</html>

**Exercise 16: Form Validation with Array Data**

**Task:** Create a form to input user details (name and email). Validate the form to ensure no duplicate names are entered. Display the list of unique names in a styled list.

**HTML & CSS:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Unique Names Form</title>

<style>

body {

font-family: Arial, sans-serif;

}

.container {

width: 50%;

margin: 0 auto;

padding: 20px;

border: 1px solid #ddd;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

input, button {

padding: 10px;

margin-bottom: 10px;

width: calc(100% - 22px);

}

button {

background-color: #28a745;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background-color: #218838;

}

ul {

list-style-type: none;

padding: 0;

}

li {

padding: 10px;

border-bottom: 1px solid #ddd;

}

</style>

</head>

<body>

<div class="container">

<h1>User Details Form</h1>

<form id="userForm">

<input type="text" id="nameInput" placeholder="Enter name" required>

<input type="email" id="emailInput" placeholder="Enter email" required>

<button type="submit">Add User</button>

</form>

<ul id="nameList"></ul>

</div>

<script>

const form = document.getElementById('userForm');

const nameInput = document.getElementById('nameInput');

const emailInput = document.getElementById('emailInput');

const nameList = document.getElementById('nameList');

const names = [];

form.addEventListener('submit', function(event) {

event.preventDefault();

const name = nameInput.value.trim();

const email = emailInput.value.trim();

if (name && !names.includes(name)) {

names.push(name);

const li = document.createElement('li');

li.textContent = name;

nameList.appendChild(li);

nameInput.value = '';

emailInput.value = '';

} else {

alert('Name is either empty or already exists.');

}

});

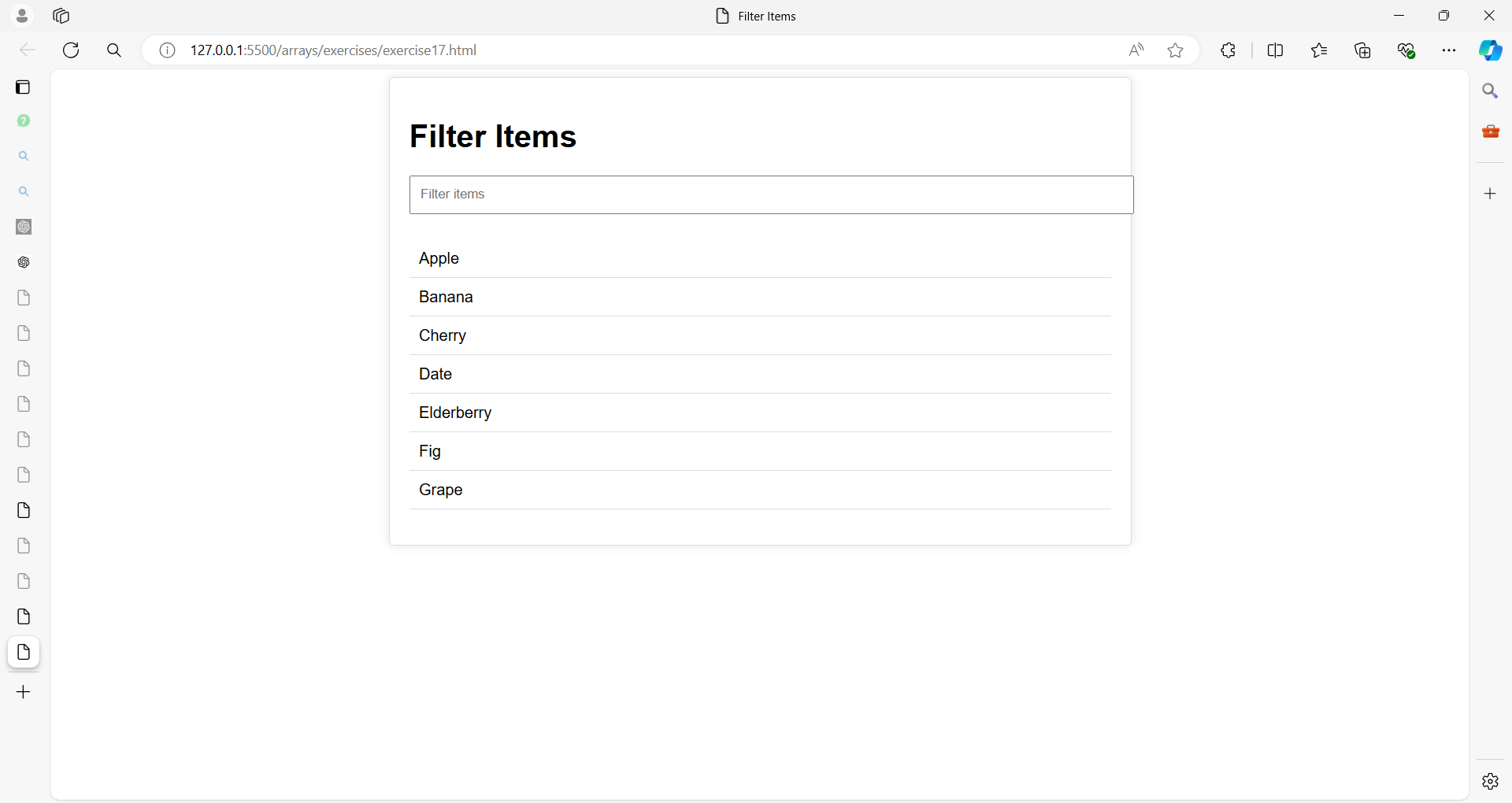
</script>

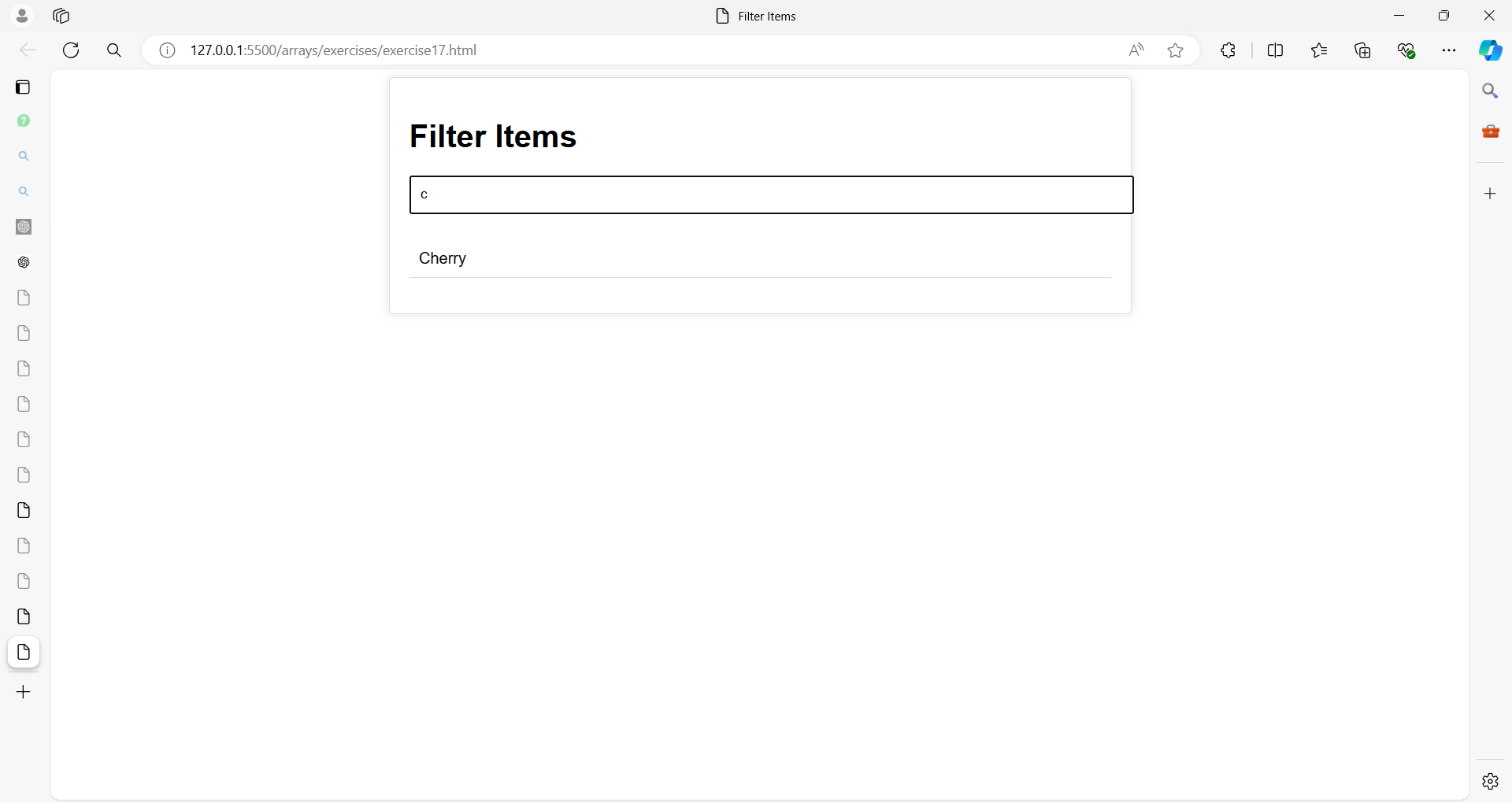
</body>

</html>

**Exercise 17: Filter and Display Array Data Based on Form Input**

**Task:** Create a form with an input field for filtering items. The list of items should be filtered based on the input and displayed in a styled list.





**HTML & CSS:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Filter Items</title>

<style>

body {

font-family: Arial, sans-serif;

}

.container {

width: 50%;

margin: 0 auto;

padding: 20px;

border: 1px solid #ddd;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

input[type="text"] {

padding: 10px;

width: 100%;

margin-bottom: 10px;

}

ul {

list-style-type: none;

padding: 0;

}

li {

padding: 10px;

border-bottom: 1px solid #ddd;

}

</style>

</head>

<body>

<div class="container">

<h1>Filter Items</h1>

<input type="text" id="filterInput" placeholder="Filter items">

<ul id="itemList"></ul>

</div>

<script>

const filterInput = document.getElementById('filterInput');

const itemList = document.getElementById('itemList');

const items = ['Apple', 'Banana', 'Cherry', 'Date', 'Elderberry', 'Fig', 'Grape'];

function displayItems(filteredItems) {

itemList.innerHTML = '';

filteredItems.forEach(item => {

const li = document.createElement('li');

li.textContent = item;

itemList.appendChild(li);

});

}

filterInput.addEventListener('input', function() {

const filterText = filterInput.value.toLowerCase();

const filteredItems = items.filter(item => item.toLowerCase().includes(filterText));

displayItems(filteredItems);

});

// Initial display

displayItems(items);

</script>

</body>

</html>

**Exercise 18: Product Management Application**

**Objective:** Build a Product Management Application where users can:

* Add products with various attributes (name, category, price, quantity).
* Edit existing products.
* Delete products.
* Filter products by name or category.
* Validate inputs before adding or updating products.

**Requirements:**

1. **Form Elements:**
   * Product Name
   * Category (Dropdown)
   * Price
   * Quantity
2. **Buttons:**
   * Add Product
   * Edit Product
   * Delete Product
3. **Product List Display:**
   * Display the list of products in a table.
4. **Filter:**
   * Filter products by name or category using an input field.
5. **Validation:**
   * Ensure that all fields are filled in correctly before adding or editing a product.
6. **CSS Styling:**
   * Apply modern styling for a clean and user-friendly interface.

Here are the values for the products:

iPhone 14, Electronics, 999, 50

Nike Air Max, Clothing, 120, 100

Samsung Galaxy S22, Electronics, 899, 75

Levi's 501 Jeans, Clothing, 89, 200

Apple MacBook Pro, Electronics, 1299, 30

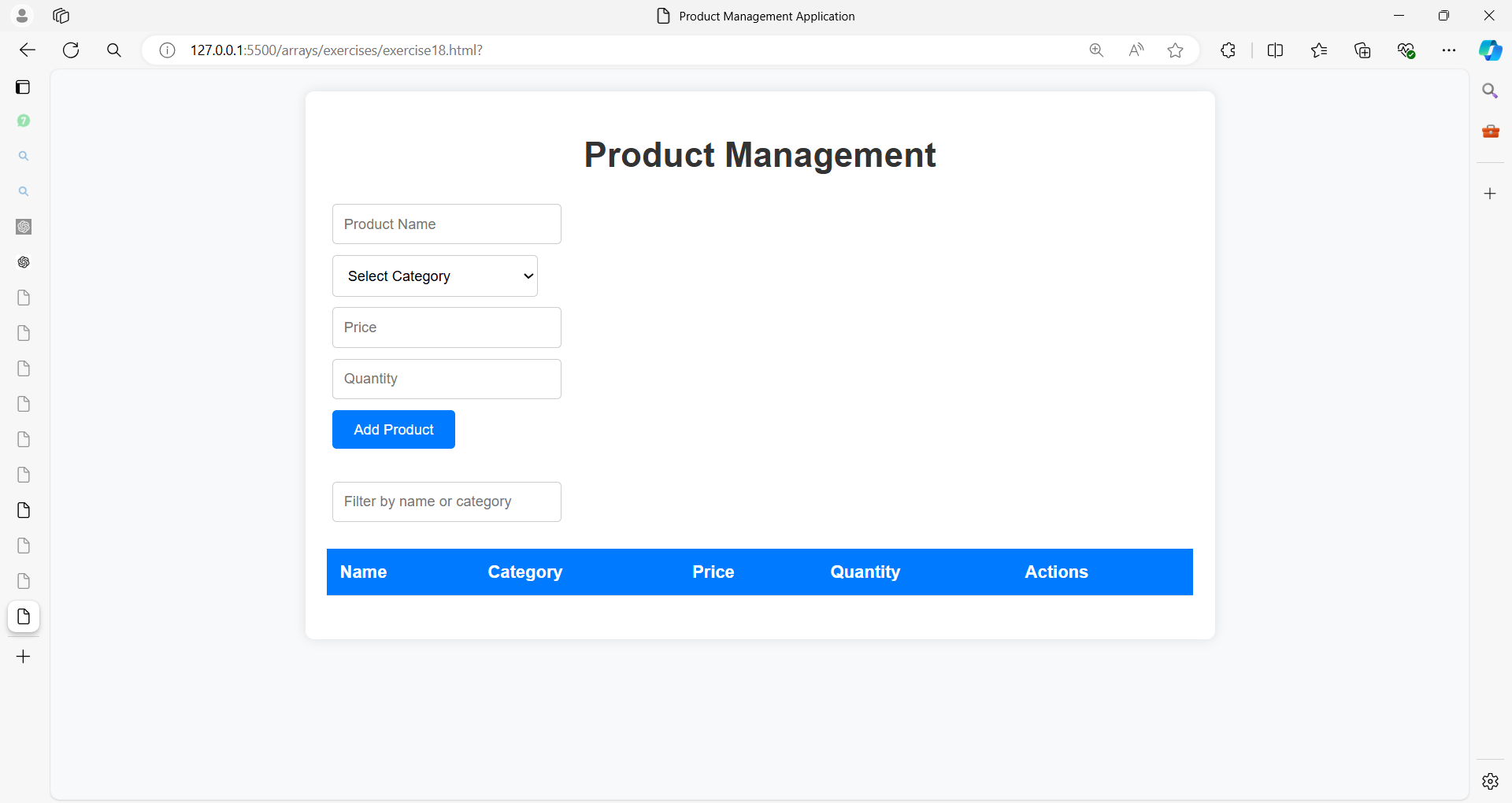
Organic Almonds (1 lb), Groceries, 15, 150

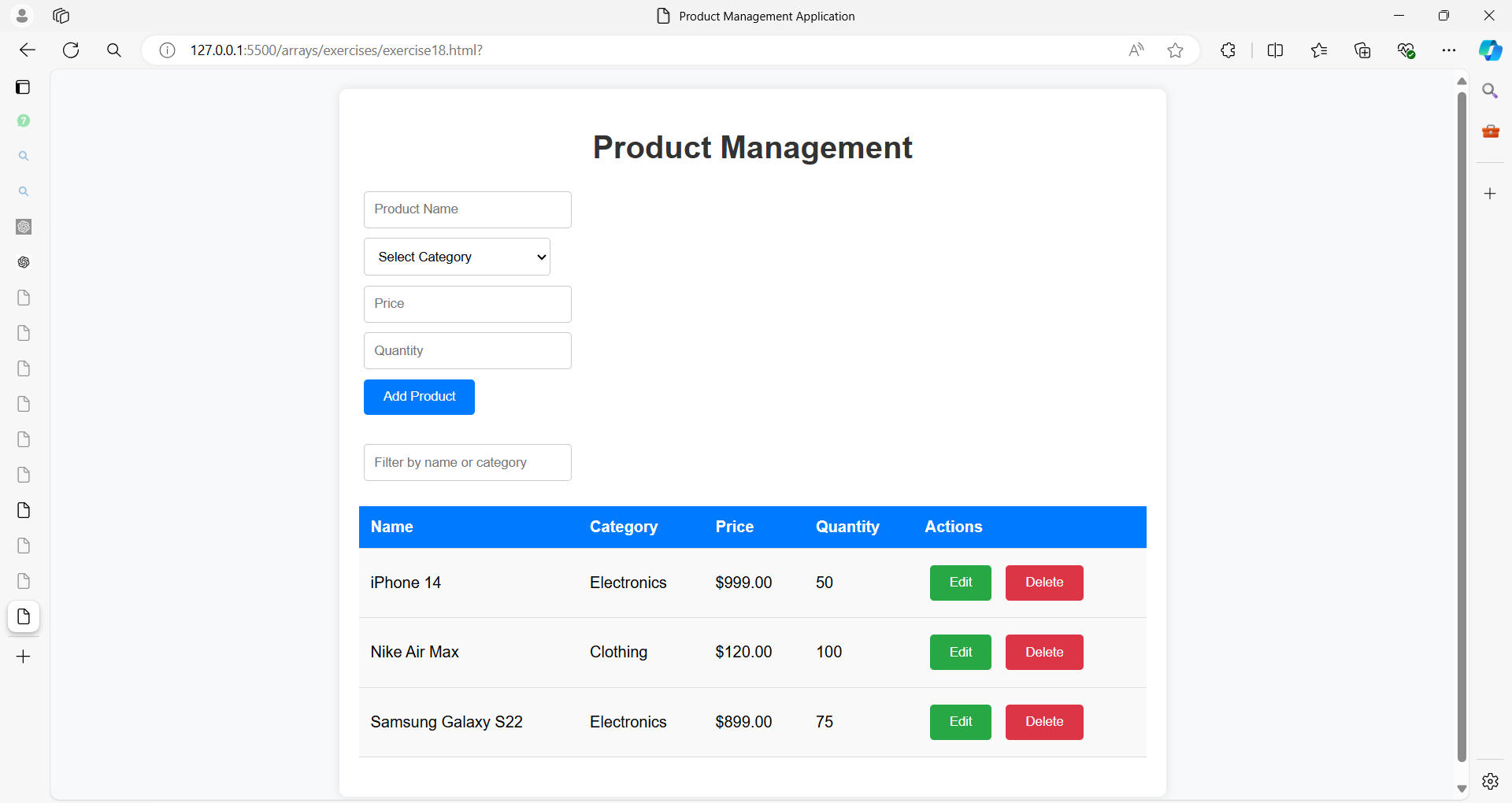
Sony WH-1000XM4 Headphones, Electronics, 349, 40

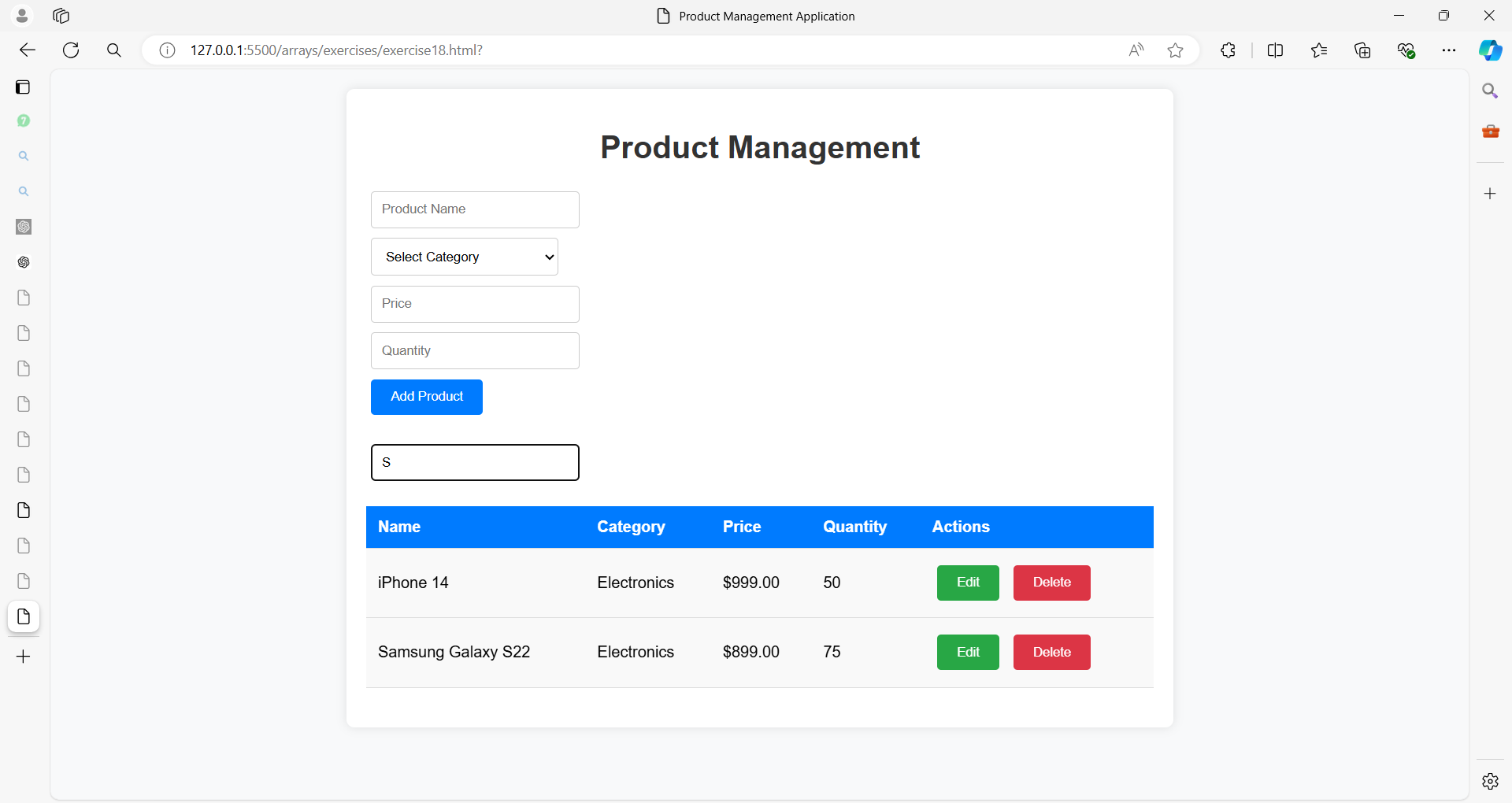
Patagonia Down Jacket, Clothing, 279, 60

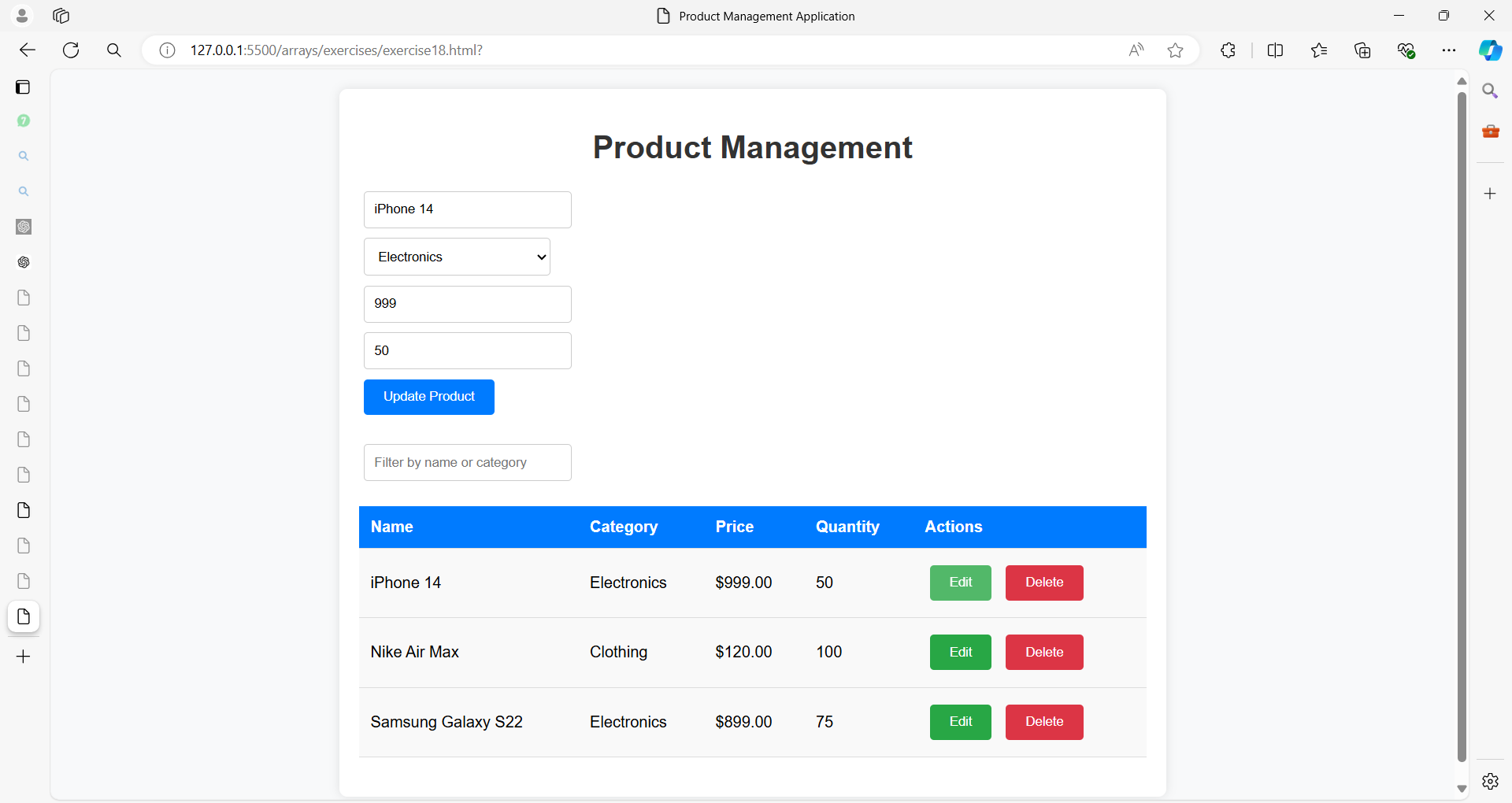
Organic Quinoa (2 lb), Groceries, 10, 250

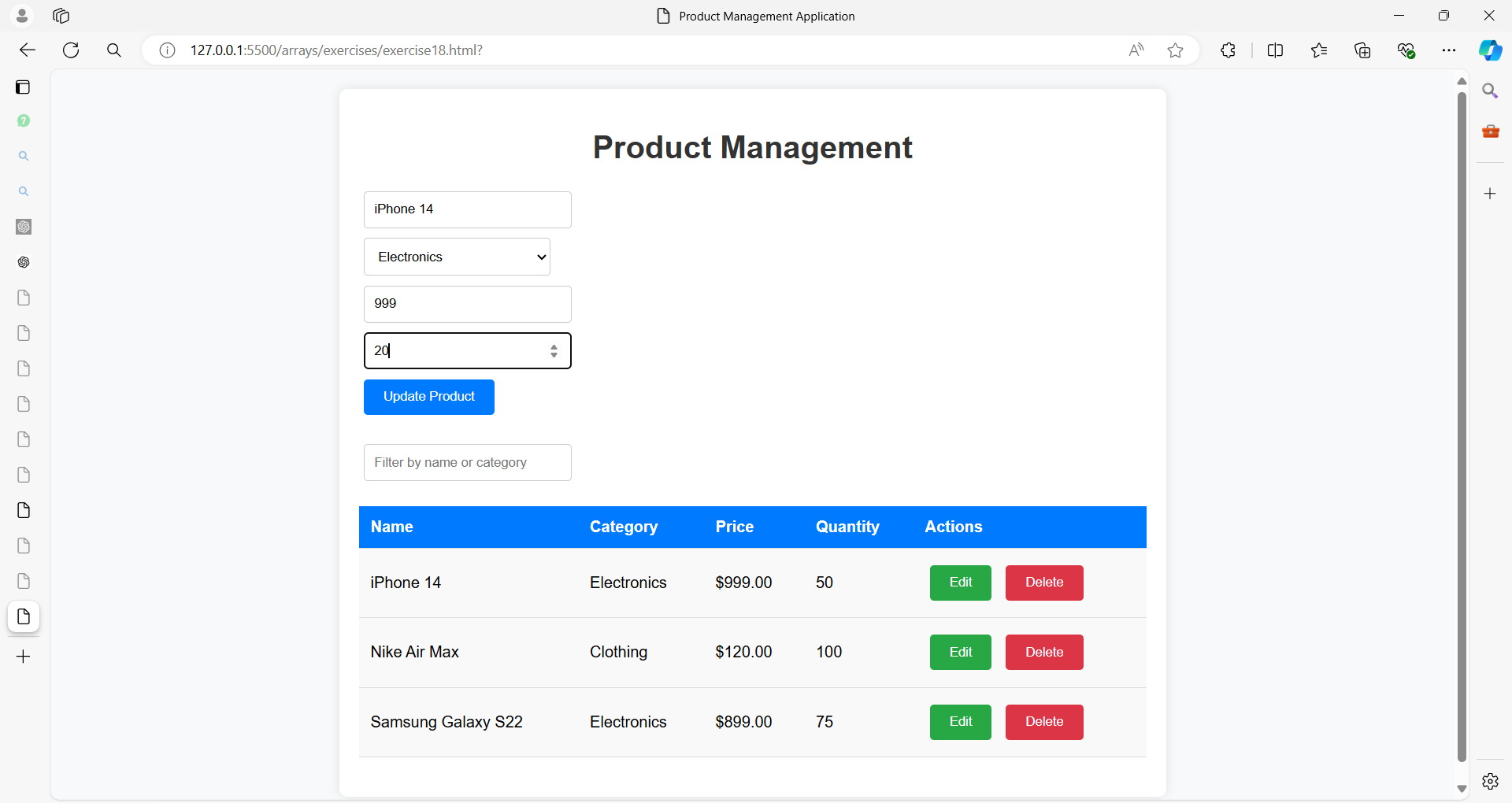
ASUS ROG Gaming Laptop, Electronics, 1499, 25

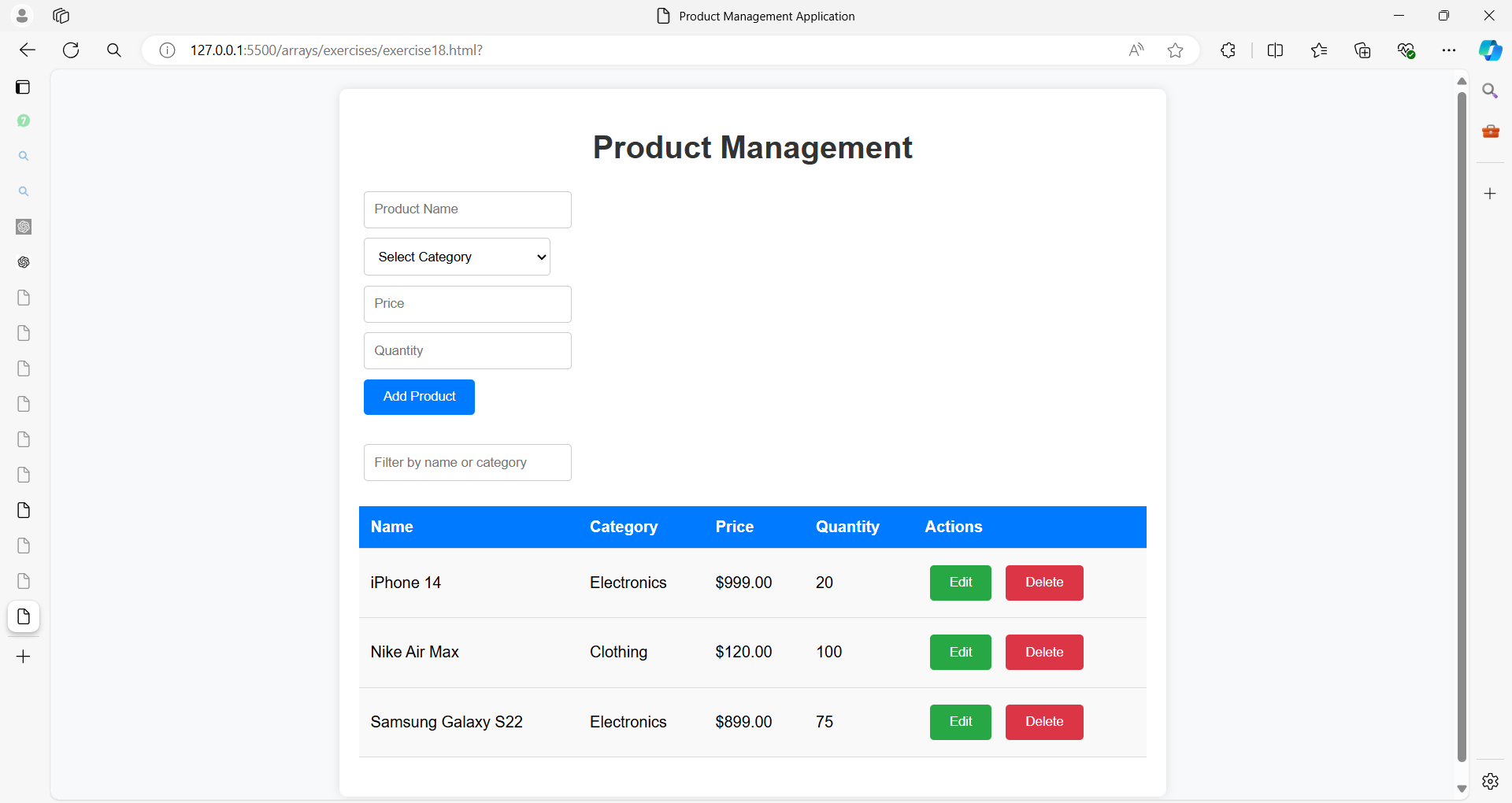


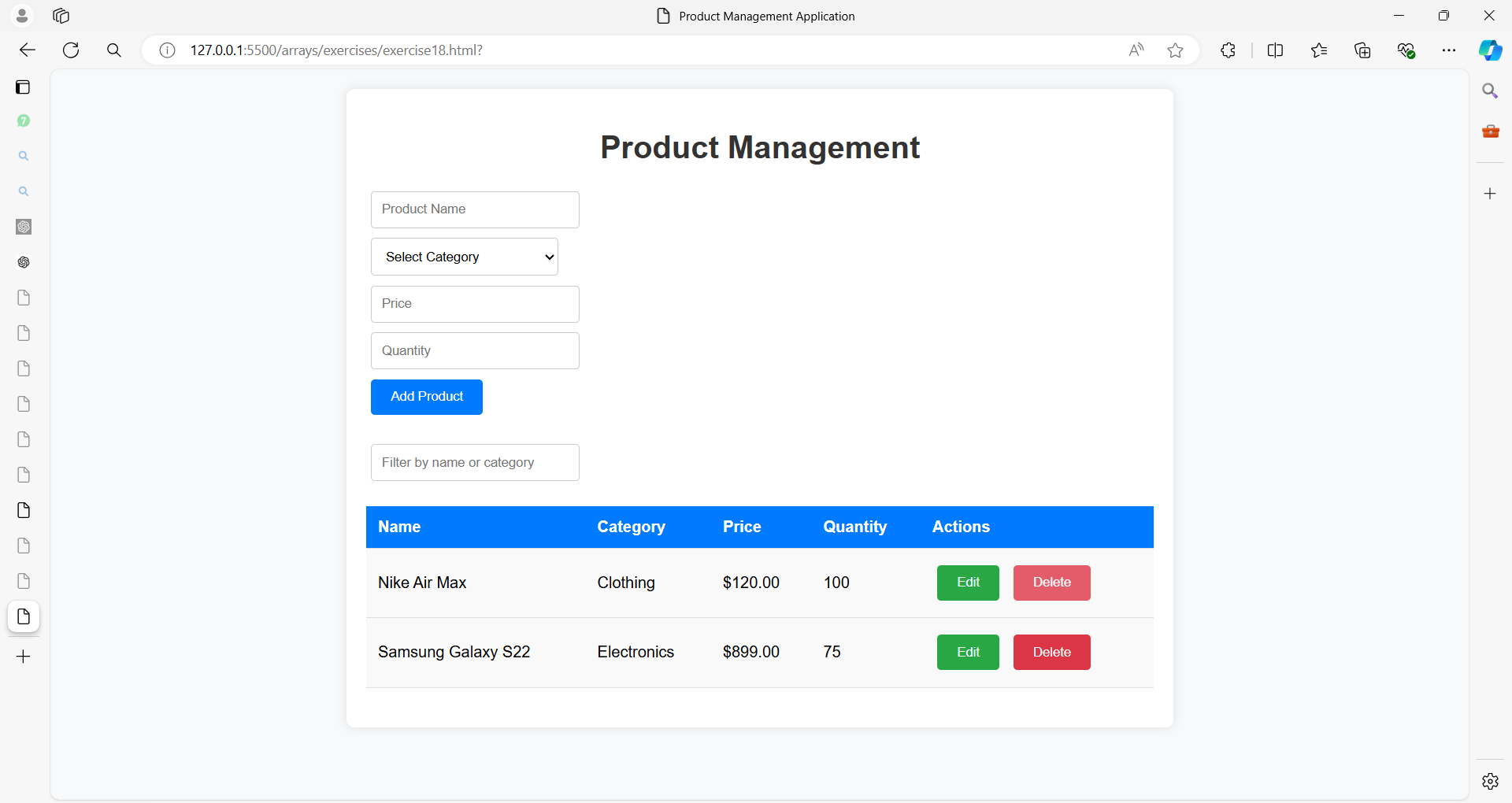












**Solution:**

**Step 1: HTML Structure**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Product Management Application</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h1>Product Management</h1>

<div class="form-container">

<form id="productForm">

<div class="form-group">

<input type="text" id="productName" placeholder="Product Name" required>

<span class="error-message" id="nameError"></span>

</div>

<div class="form-group">

<select id="productCategory" required>

<option value="">Select Category</option>

<option value="Electronics">Electronics</option>

<option value="Clothing">Clothing</option>

<option value="Groceries">Groceries</option>

</select>

<span class="error-message" id="categoryError"></span>

</div>

<div class="form-group">

<input type="number" id="productPrice" placeholder="Price" required>

<span class="error-message" id="priceError"></span>

</div>

<div class="form-group">

<input type="number" id="productQuantity" placeholder="Quantity" required>

<span class="error-message" id="quantityError"></span>

</div>

<button type="submit" id="addProductBtn">Add Product</button>

</form>

</div>

<div class="filter-container">

<input type="text" id="filterInput" placeholder="Filter by name or category">

</div>

<div class="table-container">

<table id="productTable">

<thead>

<tr>

<th>Name</th>

<th>Category</th>

<th>Price</th>

<th>Quantity</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

<!-- Products will be dynamically added here -->

</tbody>

</table>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**Step 2: CSS Styling**

/\* styles.css \*/

body {

font-family: Arial, sans-serif;

background-color: #f8f9fa;

margin: 0;

padding: 20px;

}

.container {

max-width: 800px;

margin: 0 auto;

background-color: #fff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

h1 {

text-align: center;

color: #333;

}

.form-container,

.filter-container,

.table-container {

margin-bottom: 20px;

}

input[type="text"],

input[type="number"],

select {

width: calc(25% - 10px);

padding: 10px;

margin: 5px;

border: 1px solid #ccc;

border-radius: 4px;

}

button {

padding: 10px 20px;

margin: 5px;

border: none;

background-color: #007bff;

color: white;

border-radius: 4px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

table {

width: 100%;

border-collapse: collapse;

}

th,

td {

padding: 12px;

text-align: left;

border-bottom: 1px solid #ddd;

}

th {

background-color: #007bff;

color: white;

}

td {

background-color: #f9f9f9;

}

.action-buttons button {

background-color: #28a745;

margin-right: 5px;

}

.action-buttons button.delete-btn {

background-color: #dc3545;

}

.action-buttons button:hover {

opacity: 0.8;

}

.error-message {

color: red;

font-size: 0.9em;

margin-left: 5px;

}

.form-group {

display: flex;

flex-direction: column;

}

**Step 3: JavaScript Logic**

// script.js

document.addEventListener('DOMContentLoaded', function() {

    const form = document.getElementById('productForm');

    const productName = document.getElementById('productName');

    const productCategory = document.getElementById('productCategory');

    const productPrice = document.getElementById('productPrice');

    const productQuantity = document.getElementById('productQuantity');

    const filterInput = document.getElementById('filterInput');

    const productTable = document.getElementById('productTable').querySelector('tbody');

    const nameError = document.getElementById('nameError');

    const categoryError = document.getElementById('categoryError');

    const priceError = document.getElementById('priceError');

    const quantityError = document.getElementById('quantityError');

    let products = [];

    let editIndex = -1;

    form.addEventListener('submit', function(event) {

        event.preventDefault();

        clearErrors();

        const name = productName.value.trim();

        const category = productCategory.value;

        const price = parseFloat(productPrice.value);

        const quantity = parseInt(productQuantity.value);

        if (validateForm(name, category, price, quantity)) {

            if (editIndex === -1) {

                products.push({ name, category, price, quantity });

            } else {

                products[editIndex] = { name, category, price, quantity };

                editIndex = -1;

            }

            resetForm();

            renderTable();

        }

    });

    function validateForm(name, category, price, quantity) {

        let isValid = true;

        if (!name) {

            nameError.textContent = 'Product name is required.';

            isValid = false;

        }

        if (!category) {

            categoryError.textContent = 'Please select a category.';

            isValid = false;

        }

        if (!price || price <= 0) {

            priceError.textContent = 'Price must be a positive number.';

            isValid = false;

        }

        if (!quantity || quantity <= 0) {

            quantityError.textContent = 'Quantity must be a positive number.';

            isValid = false;

        }

        return isValid;

    }

    function clearErrors() {

        nameError.textContent = '';

        categoryError.textContent = '';

        priceError.textContent = '';

        quantityError.textContent = '';

    }

    function resetForm() {

        productName.value = '';

        productCategory.value = '';

        productPrice.value = '';

        productQuantity.value = '';

        document.getElementById('addProductBtn').textContent = 'Add Product';

    }

    function renderTable() {

        productTable.innerHTML = '';

        products.forEach((product, index) => {

            const row = document.createElement('tr');

            row.innerHTML = `

                <td>${product.name}</td>

                <td>${product.category}</td>

                <td>$${product.price.toFixed(2)}</td>

                <td>${product.quantity}</td>

                <td class="action-buttons">

                    <button onclick="editProduct(${index})">Edit</button>

                    <button class="delete-btn" onclick="deleteProduct(${index})">Delete</button>

                </td>

            `;

            productTable.appendChild(row);

        });

    }

    window.editProduct = function(index) {

        const product = products[index];

        productName.value = product.name;

        productCategory.value = product.category;

        productPrice.value = product.price;

        productQuantity.value = product.quantity;

        editIndex = index;

        document.getElementById('addProductBtn').textContent = 'Update Product';

    };

    window.deleteProduct = function(index) {

        products.splice(index, 1);

        renderTable();

    };

    filterInput.addEventListener('input', function() {

        const filterText = filterInput.value.toLowerCase();

        const filteredProducts = products.filter(product =>

            product.name.toLowerCase().includes(filterText) ||

            product.category.toLowerCase().includes(filterText)

        );

        renderFilteredTable(filteredProducts);

    });

    function renderFilteredTable(filteredProducts) {

        productTable.innerHTML = '';

        filteredProducts.forEach((product) => {

            const index = products.indexOf(product);

            const row = document.createElement('tr');

            row.innerHTML = `

                <td>${product.name}</td>

                <td>${product.category}</td>

                <td>$${product.price.toFixed(2)}</td>

                <td>${product.quantity}</td>

                <td class="action-buttons">

                    <button onclick="editProduct(${index})">Edit</button>

                    <button class="delete-btn" onclick="deleteProduct(${index})">Delete</button>

                </td>

            `;

            productTable.appendChild(row);

        });

    }

});

**Explanation of the Solution:**

1. **HTML Structure:**
   * The structure includes a form for adding products, a filter input for searching products, and a table to display the products.
   * The form includes fields for the product name, category, price, and quantity.
   * The table dynamically displays the products added by the user.
2. **CSS Styling:**
   * The CSS styles provide a clean, modern look with a responsive layout.
   * Buttons, input fields, and table rows are styled for a better user experience.
3. **JavaScript Logic:**
   * The productForm handles adding and updating products.
   * Products are stored in an array, and each product is represented as an object.
   * The renderTable() function dynamically creates table rows based on the products array.
   * The editProduct() function populates the form with the selected product’s data for editing.
   * The deleteProduct() function removes a product from the array and updates the table.
   * The filterInput allows users to filter the products by name or category, updating the table with matching results.

**How to Extend This Application:**

* Add more fields to the product form, such as descriptions, images, or ratings.
* Implement sorting functionality by different columns (e.g., sort by price or name).
* Add pagination if the number of products exceeds a certain limit.
* Store products in localStorage to persist data across sessions.

This exercise provides a strong foundation for building more complex web applications that involve form handling, data validation, dynamic DOM manipulation, and user interaction with JavaScript and CSS.

**Exercise 19: Library Management System**

**Objective:**

Build a Library Management System using HTML, CSS3, and JavaScript. The system will allow users to manage a collection of books, including adding, viewing, editing, deleting, and filtering books.

**Step 1: HTML Structure**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Library Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h1>Library Management System</h1>

<div class="form-container">

<form id="bookForm">

<div class="form-group">

<input type="text" id="bookTitle" placeholder="Book Title" required>

<span class="error-message" id="titleError"></span>

</div>

<div class="form-group">

<input type="text" id="bookAuthor" placeholder="Author" required>

<span class="error-message" id="authorError"></span>

</div>

<div class="form-group">

<input type="text" id="bookGenre" placeholder="Genre" required>

<span class="error-message" id="genreError"></span>

</div>

<div class="form-group">

<input type="number" id="bookYear" placeholder="Publication Year" required>

<span class="error-message" id="yearError"></span>

</div>

<button type="submit" id="addBookBtn">Add Book</button>

</form>

</div>

<div class="filter-container">

<input type="text" id="filterInput" placeholder="Filter by title, author, or genre">

</div>

<div class="table-container">

<table id="bookTable">

<thead>

<tr>

<th>Title</th>

<th>Author</th>

<th>Genre</th>

<th>Year</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

<!-- Books will be dynamically added here -->

</tbody>

</table>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**Step 2: CSS Styling**

/\* styles.css \*/

body {

font-family: Arial, sans-serif;

background-color: #f0f4f8;

margin: 0;

padding: 20px;

}

.container {

max-width: 900px;

margin: 0 auto;

background-color: #ffffff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);

}

h1 {

text-align: center;

color: #333333;

}

.form-container,

.filter-container,

.table-container {

margin-bottom: 20px;

}

input[type="text"],

input[type="number"] {

width: calc(25% - 10px);

padding: 10px;

margin: 5px;

border: 1px solid #ccc;

border-radius: 4px;

}

button {

padding: 10px 20px;

margin: 5px;

border: none;

background-color: #007bff;

color: white;

border-radius: 4px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

table {

width: 100%;

border-collapse: collapse;

}

th,

td {

padding: 12px;

text-align: left;

border-bottom: 1px solid #ddd;

}

th {

background-color: #007bff;

color: white;

}

td {

background-color: #f9f9f9;

}

.action-buttons button {

background-color: #28a745;

margin-right: 5px;

}

.action-buttons button.delete-btn {

background-color: #dc3545;

}

.action-buttons button:hover {

opacity: 0.8;

}

.error-message {

color: red;

font-size: 0.9em;

margin-left: 5px;

}

.form-group {

display: flex;

flex-direction: column;

}

**Step 3: JavaScript Logic**

// script.js

document.addEventListener('DOMContentLoaded', function() {

const form = document.getElementById('bookForm');

const bookTitle = document.getElementById('bookTitle');

const bookAuthor = document.getElementById('bookAuthor');

const bookGenre = document.getElementById('bookGenre');

const bookYear = document.getElementById('bookYear');

const filterInput = document.getElementById('filterInput');

const bookTable = document.getElementById('bookTable').querySelector('tbody');

const titleError = document.getElementById('titleError');

const authorError = document.getElementById('authorError');

const genreError = document.getElementById('genreError');

const yearError = document.getElementById('yearError');

let books = [];

let editIndex = -1;

form.addEventListener('submit', function(event) {

event.preventDefault();

clearErrors();

const title = bookTitle.value.trim();

const author = bookAuthor.value.trim();

const genre = bookGenre.value.trim();

const year = parseInt(bookYear.value);

if (validateForm(title, author, genre, year)) {

if (editIndex === -1) {

books.push({ title, author, genre, year });

} else {

books[editIndex] = { title, author, genre, year };

editIndex = -1;

}

resetForm();

renderTable();

}

});

function validateForm(title, author, genre, year) {

let isValid = true;

if (!title) {

titleError.textContent = 'Book title is required.';

isValid = false;

}

if (!author) {

authorError.textContent = 'Author name is required.';

isValid = false;

}

if (!genre) {

genreError.textContent = 'Genre is required.';

isValid = false;

}

if (!year || year <= 0 || year > new Date().getFullYear()) {

yearError.textContent = 'Please enter a valid publication year.';

isValid = false;

}

return isValid;

}

function clearErrors() {

titleError.textContent = '';

authorError.textContent = '';

genreError.textContent = '';

yearError.textContent = '';

}

function resetForm() {

bookTitle.value = '';

bookAuthor.value = '';

bookGenre.value = '';

bookYear.value = '';

document.getElementById('addBookBtn').textContent = 'Add Book';

}

function renderTable() {

bookTable.innerHTML = '';

books.forEach((book, index) => {

const row = document.createElement('tr');

row.innerHTML = `

<td>${book.title}</td>

<td>${book.author}</td>

<td>${book.genre}</td>

<td>${book.year}</td>

<td class="action-buttons">

<button onclick="editBook(${index})">Edit</button>

<button class="delete-btn" onclick="deleteBook(${index})">Delete</button>

</td>

`;

bookTable.appendChild(row);

});

}

window.editBook = function(index) {

const book = books[index];

bookTitle.value = book.title;

bookAuthor.value = book.author;

bookGenre.value = book.genre;

bookYear.value = book.year;

editIndex = index;

document.getElementById('addBookBtn').textContent = 'Update Book';

};

window.deleteBook = function(index) {

books.splice(index, 1);

renderTable();

};

filterInput.addEventListener('input', function() {

const filterText = filterInput.value.toLowerCase();

const filteredBooks = books.filter(book =>

book.title.toLowerCase().includes(filterText) ||

book.author.toLowerCase().includes(filterText) ||

book.genre.toLowerCase().includes(filterText)

);

renderFilteredTable(filteredBooks);

});

function renderFilteredTable(filteredBooks) {

bookTable.innerHTML = '';

filteredBooks.forEach((book) => {

const index = books.indexOf(book);

const row = document.createElement('tr');

row.innerHTML = `

<td>${book.title}</td>

<td>${book.author}</td>

<td>${book.genre}</td>

<td>${book.year}</td>

<td class="action-buttons">

<button onclick="editBook(${index})">Edit</button>

<button class="delete-btn" onclick="deleteBook(${index})">Delete</button>

</td>

`;

bookTable.appendChild(row);

});

}

});

**Explanation and Functionality:**

1. **HTML Structure**:
   * The HTML contains a form for adding books, a filter input, and a table for displaying the books.
   * Form inputs include fields for the book's title, author, genre, and publication year.
2. **CSS Styling**:
   * The CSS styles the form, table, and buttons, ensuring a clean and responsive design.
   * Error messages are styled to appear below the respective input fields.
3. **JavaScript Logic**:
   * The JavaScript handles the addition, editing, deletion, and filtering of books using an array.
   * **Form Validation**: Before adding or updating a book, the form is validated to ensure all fields are filled correctly.
   * **Filtering**: Users can filter books by title, author, or genre.
   * **CRUD Operations**: Users can add new books, edit existing ones, or delete them from the array.
   * **Dynamic Table Rendering**: The table updates dynamically based on the current state of the books array and any applied filters.

This exercise integrates multiple aspects of web development, including HTML forms, CSS styling, and JavaScript array manipulation, providing a comprehensive example of how these technologies can work together in a practical application.

**Exercise 20: Student Management System**

**Objective:**  
Build a Student Management System using HTML, CSS3, and JavaScript. The system will allow users to manage a list of students, including adding, viewing, editing, deleting, filtering, sorting by various fields, and calculating statistics like the average grade, highest grade, and lowest grade.

**Step 1: HTML Structure**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Student Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h1>Student Management System</h1>

<div class="form-container">

<form id="studentForm">

<div class="form-group">

<input type="text" id="studentName" placeholder="Student Name" required>

<span class="error-message" id="nameError"></span>

</div>

<div class="form-group">

<input type="number" id="studentAge" placeholder="Age" required>

<span class="error-message" id="ageError"></span>

</div>

<div class="form-group">

<input type="text" id="studentClass" placeholder="Class" required>

<span class="error-message" id="classError"></span>

</div>

<div class="form-group">

<input type="number" id="studentGrade" placeholder="Grade (0-100)" required>

<span class="error-message" id="gradeError"></span>

</div>

<button type="submit" id="addStudentBtn">Add Student</button>

</form>

</div>

<div class="filter-container">

<input type="text" id="filterInput" placeholder="Filter by name, class, or grade">

<select id="sortBy" class="sort-dropdown">

<option value="name">Sort by Name</option>

<option value="age">Sort by Age</option>

<option value="class">Sort by Class</option>

<option value="grade">Sort by Grade</option>

</select>

</div>

<div class="table-container">

<table id="studentTable">

<thead>

<tr>

<th>Name</th>

<th>Age</th>

<th>Class</th>

<th>Grade</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

<!-- Students will be dynamically added here -->

</tbody>

</table>

</div>

<div class="statistics-container">

<h2>Statistics</h2>

<p id="averageGrade">Average Grade: N/A</p>

<p id="highestGrade">Highest Grade: N/A</p>

<p id="lowestGrade">Lowest Grade: N/A</p>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**Step 2: CSS Styling**

/\* styles.css \*/

body {

font-family: Arial, sans-serif;

background-color: #f0f4f8;

margin: 0;

padding: 20px;

}

.container {

max-width: 1000px;

margin: 0 auto;

background-color: #ffffff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);

}

h1, h2 {

text-align: center;

color: #333333;

}

.form-container,

.filter-container,

.table-container,

.statistics-container {

margin-bottom: 20px;

}

input[type="text"],

input[type="number"],

select.sort-dropdown {

width: calc(25% - 10px);

padding: 10px;

margin: 5px;

border: 1px solid #ccc;

border-radius: 4px;

}

button {

padding: 10px 20px;

margin: 5px;

border: none;

background-color: #007bff;

color: white;

border-radius: 4px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

table {

width: 100%;

border-collapse: collapse;

}

th,

td {

padding: 12px;

text-align: left;

border-bottom: 1px solid #ddd;

}

th {

background-color: #007bff;

color: white;

cursor: pointer;

}

td {

background-color: #f9f9f9;

}

.action-buttons button {

background-color: #28a745;

margin-right: 5px;

}

.action-buttons button.delete-btn {

background-color: #dc3545;

}

.action-buttons button:hover {

opacity: 0.8;

}

.error-message {

color: red;

font-size: 0.9em;

margin-left: 5px;

}

.form-group {

display: flex;

flex-direction: column;

}

.statistics-container {

text-align: center;

font-size: 1.2em;

color: #555555;

}

**Step 3: JavaScript Logic**

// script.js

document.addEventListener('DOMContentLoaded', function() {

const form = document.getElementById('studentForm');

const studentName = document.getElementById('studentName');

const studentAge = document.getElementById('studentAge');

const studentClass = document.getElementById('studentClass');

const studentGrade = document.getElementById('studentGrade');

const filterInput = document.getElementById('filterInput');

const sortBy = document.getElementById('sortBy');

const studentTable = document.getElementById('studentTable').querySelector('tbody');

const averageGradeElem = document.getElementById('averageGrade');

const highestGradeElem = document.getElementById('highestGrade');

const lowestGradeElem = document.getElementById('lowestGrade');

const nameError = document.getElementById('nameError');

const ageError = document.getElementById('ageError');

const classError = document.getElementById('classError');

const gradeError = document.getElementById('gradeError');

let students = [];

let editIndex = -1;

form.addEventListener('submit', function(event) {

event.preventDefault();

clearErrors();

const name = studentName.value.trim();

const age = parseInt(studentAge.value);

const className = studentClass.value.trim();

const grade = parseFloat(studentGrade.value);

if (validateForm(name, age, className, grade)) {

if (editIndex === -1) {

students.push({ name, age, class: className, grade });

} else {

students[editIndex] = { name, age, class: className, grade };

editIndex = -1;

}

resetForm();

renderTable();

calculateStatistics();

}

});

function validateForm(name, age, className, grade) {

let isValid = true;

if (!name) {

nameError.textContent = 'Student name is required.';

isValid = false;

}

if (!age || age <= 0) {

ageError.textContent = 'Please enter a valid age.';

isValid = false;

}

if (!className) {

classError.textContent = 'Class is required.';

isValid = false;

}

if (!grade || grade < 0 || grade > 100) {

gradeError.textContent = 'Please enter a valid grade (0-100).';

isValid = false;

}

return isValid;

}

function clearErrors() {

nameError.textContent = '';

ageError.textContent = '';

classError.textContent = '';

gradeError.textContent = '';

}

function resetForm() {

studentName.value = '';

studentAge.value = '';

studentClass.value = '';

studentGrade.value = '';

document.getElementById('addStudentBtn').textContent = 'Add Student';

}

function renderTable() {

studentTable.innerHTML = '';

const sortedStudents = getSortedStudents();

sortedStudents.forEach((student, index) => {

const row = document.createElement('tr');

row.innerHTML = `

<td>${student.name}</td>

<td>${student.age}</td>

<td>${student.class}</td>

<td>${student.grade.toFixed(2)}</td>

<td class="action-buttons">

<button onclick="editStudent(${index})">Edit</button>

<button class="delete-btn" onclick="deleteStudent(${index})">Delete</button>

</td>

`;

studentTable.appendChild(row);

});

}

function getSortedStudents() {

const sortByValue = sortBy.value;

const filteredStudents = students.filter(student =>

student.name.toLowerCase().includes(filterInput.value.toLowerCase()) ||

student.class.toLowerCase().includes(filterInput.value.toLowerCase()) ||

student.grade.toString().includes(filterInput.value)

);

return filteredStudents.sort((a, b) => {

if (sortByValue === 'name') {

return a.name.localeCompare(b.name);

} else if (sortByValue === 'age') {

return a.age - b.age;

} else if (sortByValue === 'class') {

return a.class.localeCompare(b.class);

} else if (sortByValue === 'grade') {

return a.grade - b.grade;

}

});

}

filterInput.addEventListener('input', renderTable);

sortBy.addEventListener('change', renderTable);

window.editStudent = function(index) {

const student = students[index];

studentName.value = student.name;

studentAge.value = student.age;

studentClass.value = student.class;

studentGrade.value = student.grade;

document.getElementById('addStudentBtn').textContent = 'Update Student';

editIndex = index;

};

window.deleteStudent = function(index) {

students.splice(index, 1);

renderTable();

calculateStatistics();

};

function calculateStatistics() {

if (students.length === 0) {

averageGradeElem.textContent = 'Average Grade: N/A';

highestGradeElem.textContent = 'Highest Grade: N/A';

lowestGradeElem.textContent = 'Lowest Grade: N/A';

return;

}

const totalGrade = students.reduce((total, student) => total + student.grade, 0);

const averageGrade = totalGrade / students.length;

const highestGrade = Math.max(...students.map(student => student.grade));

const lowestGrade = Math.min(...students.map(student => student.grade));

averageGradeElem.textContent = `Average Grade: ${averageGrade.toFixed(2)}`;

highestGradeElem.textContent = `Highest Grade: ${highestGrade.toFixed(2)}`;

lowestGradeElem.textContent = `Lowest Grade: ${lowestGrade.toFixed(2)}`;

}

});

**Explanation and Functionality:**

1. **HTML Structure**:
   * The HTML includes a form for adding students, a filter input, a sort dropdown, a table for displaying student data, and a statistics section.
   * Form fields include student name, age, class, and grade.
2. **CSS Styling**:
   * The CSS styles the form, table, buttons, and statistics section, ensuring a responsive design.
   * Error messages are displayed below the relevant input fields when validation fails.
3. **JavaScript Logic**:
   * **Form Validation**: The form is validated to ensure all fields are filled in correctly before adding or updating a student.
   * **Filtering and Sorting**: The list of students can be filtered by name, class, or grade, and sorted by name, age, class, or grade.
   * **CRUD Operations**: Users can add, edit, and delete student records.
   * **Dynamic Table Rendering**: The student data table updates dynamically based on the current state of the students array, with sorting and filtering applied.
   * **Statistics Calculation**: The system calculates and displays the average grade, highest grade, and lowest grade of the students.

This exercise combines array manipulation, form validation, DOM manipulation, and user interface design in a more comprehensive manner, providing a robust example of building a full-featured application using HTML, CSS, and JavaScript.

**Exercise 21: Task Management System**

**Objective:**  
Build a Task Management System using HTML, CSS3, and JavaScript. The system will allow users to manage tasks by adding, viewing, editing, deleting, filtering, sorting, and marking tasks as completed. Users will also be able to categorize tasks, prioritize them, and set due dates.

**Step 1: HTML Structure**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Task Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h1>Task Management System</h1>

<div class="form-container">

<form id="taskForm">

<div class="form-group">

<input type="text" id="taskTitle" placeholder="Task Title" required>

<span class="error-message" id="titleError"></span>

</div>

<div class="form-group">

<textarea id="taskDescription" placeholder="Task Description"></textarea>

</div>

<div class="form-group">

<input type="date" id="taskDueDate" required>

<span class="error-message" id="dueDateError"></span>

</div>

<div class="form-group">

<select id="taskPriority">

<option value="low">Low Priority</option>

<option value="medium">Medium Priority</option>

<option value="high">High Priority</option>

</select>

</div>

<div class="form-group">

<input type="text" id="taskCategory" placeholder="Category (e.g., Work, Personal)" required>

<span class="error-message" id="categoryError"></span>

</div>

<button type="submit" id="addTaskBtn">Add Task</button>

</form>

</div>

<div class="filter-container">

<input type="text" id="filterInput" placeholder="Filter by title, category, or priority">

<select id="sortBy" class="sort-dropdown">

<option value="priority">Sort by Priority</option>

<option value="dueDate">Sort by Due Date</option>

</select>

</div>

<div class="table-container">

<table id="taskTable">

<thead>

<tr>

<th>Title</th>

<th>Description</th>

<th>Due Date</th>

<th>Priority</th>

<th>Category</th>

<th>Status</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

<!-- Tasks will be dynamically added here -->

</tbody>

</table>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**Step 2: CSS Styling**

/\* styles.css \*/

body {

font-family: Arial, sans-serif;

background-color: #f0f4f8;

margin: 0;

padding: 20px;

}

.container {

max-width: 1200px;

margin: 0 auto;

background-color: #ffffff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);

}

h1 {

text-align: center;

color: #333333;

}

.form-container,

.filter-container,

.table-container {

margin-bottom: 20px;

}

input[type="text"],

input[type="date"],

textarea,

select.sort-dropdown {

width: calc(50% - 10px);

padding: 10px;

margin: 5px;

border: 1px solid #ccc;

border-radius: 4px;

}

textarea {

height: 80px;

}

button {

padding: 10px 20px;

margin: 5px;

border: none;

background-color: #007bff;

color: white;

border-radius: 4px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

table {

width: 100%;

border-collapse: collapse;

}

th,

td {

padding: 12px;

text-align: left;

border-bottom: 1px solid #ddd;

}

th {

background-color: #007bff;

color: white;

cursor: pointer;

}

td {

background-color: #f9f9f9;

}

td.completed {

text-decoration: line-through;

color: green;

}

.action-buttons button {

background-color: #28a745;

margin-right: 5px;

}

.action-buttons button.delete-btn {

background-color: #dc3545;

}

.action-buttons button:hover {

opacity: 0.8;

}

.error-message {

color: red;

font-size: 0.9em;

margin-left: 5px;

}

.form-group {

display: flex;

flex-direction: column;

}

**Step 3: JavaScript Logic**

// script.js

document.addEventListener('DOMContentLoaded', function() {

const form = document.getElementById('taskForm');

const taskTitle = document.getElementById('taskTitle');

const taskDescription = document.getElementById('taskDescription');

const taskDueDate = document.getElementById('taskDueDate');

const taskPriority = document.getElementById('taskPriority');

const taskCategory = document.getElementById('taskCategory');

const filterInput = document.getElementById('filterInput');

const sortBy = document.getElementById('sortBy');

const taskTable = document.getElementById('taskTable').querySelector('tbody');

const titleError = document.getElementById('titleError');

const dueDateError = document.getElementById('dueDateError');

const categoryError = document.getElementById('categoryError');

let tasks = [];

let editIndex = -1;

form.addEventListener('submit', function(event) {

event.preventDefault();

clearErrors();

const title = taskTitle.value.trim();

const description = taskDescription.value.trim();

const dueDate = taskDueDate.value;

const priority = taskPriority.value;

const category = taskCategory.value.trim();

if (validateForm(title, dueDate, category)) {

if (editIndex === -1) {

tasks.push({ title, description, dueDate, priority, category, completed: false });

} else {

tasks[editIndex] = { title, description, dueDate, priority, category, completed: false };

editIndex = -1;

}

resetForm();

renderTable();

}

});

function validateForm(title, dueDate, category) {

let isValid = true;

if (!title) {

titleError.textContent = 'Task title is required.';

isValid = false;

}

if (!dueDate) {

dueDateError.textContent = 'Please select a due date.';

isValid = false;

}

if (!category) {

categoryError.textContent = 'Category is required.';

isValid = false;

}

return isValid;

}

function clearErrors() {

titleError.textContent = '';

dueDateError.textContent = '';

categoryError.textContent = '';

}

function resetForm() {

taskTitle.value = '';

taskDescription.value = '';

taskDueDate.value = '';

taskPriority.value = 'low';

taskCategory.value = '';

document.getElementById('addTaskBtn').textContent = 'Add Task';

}

function renderTable() {

taskTable.innerHTML = '';

const sortedTasks = getSortedTasks();

sortedTasks.forEach((task, index) => {

const row = document.createElement('tr');

row.innerHTML = `

<td>${task.title}</td>

<td>${task.description}</td>

<td>${task.dueDate}</td>

<td>${task.priority}</td>

<td>${task.category}</td>

<td class="${task.completed ? 'completed' : ''}">

${task.completed ? 'Completed' : 'Pending'}

</td>

<td class="action-buttons">

<button onclick="markCompleted(${index})">${task.completed ? 'Undo' : 'Complete'}</button>

<button onclick="editTask(${index})">Edit</button>

<button class="delete-btn" onclick="deleteTask(${index})">Delete</button>

</td>

`;

taskTable.appendChild(row);

});

}

function getSortedTasks() {

const sortByValue = sortBy.value;

const filteredTasks = tasks.filter(task =>

task.title.toLowerCase().includes(filterInput.value.toLowerCase()) ||

task.category.toLowerCase().includes(filterInput.value.toLowerCase()) ||

task.priority.toLowerCase().includes(filterInput.value.toLowerCase())

);

return filteredTasks.sort((a, b) => {

if (sortByValue === 'priority') {

const priorityOrder = { 'low': 1, 'medium': 2, 'high': 3 };

return priorityOrder[a.priority] - priorityOrder[b.priority];

} else if (sortByValue === 'dueDate') {

return new Date(a.dueDate) - new Date(b.dueDate);

}

});

}

filterInput.addEventListener('input', renderTable);

sortBy.addEventListener('change', renderTable);

window.markCompleted = function(index) {

tasks[index].completed = !tasks[index].completed;

renderTable();

};

window.editTask = function(index) {

const task = tasks[index];

taskTitle.value = task.title;

taskDescription.value = task.description;

taskDueDate.value = task.dueDate;

taskPriority.value = task.priority;

taskCategory.value = task.category;

document.getElementById('addTaskBtn').textContent = 'Update Task';

editIndex = index;

};

window.deleteTask = function(index) {

tasks.splice(index, 1);

renderTable();

};

});

**Explanation and Functionality:**

1. **HTML Structure**:
   * The form allows the user to add tasks with fields for title, description, due date, priority, and category.
   * A filter input and sort dropdown allow users to filter and sort tasks by various criteria.
   * A table displays the tasks, with columns for title, description, due date, priority, category, status, and actions.
2. **CSS Styling**:
   * The CSS styles the form, table, buttons, and error messages.
   * Completed tasks are styled with a strikethrough and green text.
3. **JavaScript Logic**:
   * **Form Validation**: The form ensures that mandatory fields are filled out before submitting.
   * **CRUD Operations**: The user can add, edit, delete, and mark tasks as completed or undo the completion.
   * **Filtering and Sorting**: The task list can be filtered by title, category, or priority, and sorted by priority or due date.
   * **Dynamic Table Rendering**: The task table updates dynamically based on the current state of the tasks array.

This exercise offers an extensive, interactive experience with JavaScript arrays, form validation, and DOM manipulation, providing a complete task management application.

**Exercise 22: Project Management System**

**Objective:**  
Create a Project Management System using HTML, CSS3, and JavaScript. The system will allow users to manage a list of projects, including adding, editing, deleting, filtering, sorting by deadline or priority, and marking projects as completed. Each project can have a title, description, deadline, priority, category, and completion status.

**Step 1: HTML Structure**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Project Management System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h1>Project Management System</h1>

<div class="form-container">

<form id="projectForm">

<div class="form-group">

<input type="text" id="projectTitle" placeholder="Project Title" required>

<span class="error-message" id="titleError"></span>

</div>

<div class="form-group">

<textarea id="projectDescription" placeholder="Project Description"></textarea>

</div>

<div class="form-group">

<input type="date" id="projectDeadline" required>

<span class="error-message" id="deadlineError"></span>

</div>

<div class="form-group">

<select id="projectPriority">

<option value="low">Low Priority</option>

<option value="medium">Medium Priority</option>

<option value="high">High Priority</option>

</select>

</div>

<div class="form-group">

<input type="text" id="projectCategory" placeholder="Category (e.g., Work, Personal)" required>

<span class="error-message" id="categoryError"></span>

</div>

<button type="submit" id="addProjectBtn">Add Project</button>

</form>

</div>

<div class="filter-container">

<input type="text" id="filterInput" placeholder="Filter by title, category, or priority">

<select id="sortBy" class="sort-dropdown">

<option value="priority">Sort by Priority</option>

<option value="deadline">Sort by Deadline</option>

</select>

</div>

<div class="project-list" id="projectList">

<!-- Projects will be dynamically added here -->

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**Step 2: CSS Styling**

/\* styles.css \*/

body {

font-family: Arial, sans-serif;

background-color: #f0f4f8;

margin: 0;

padding: 20px;

}

.container {

max-width: 1200px;

margin: 0 auto;

background-color: #ffffff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);

}

h1 {

text-align: center;

color: #333333;

}

.form-container,

.filter-container,

.project-list {

margin-bottom: 20px;

}

input[type="text"],

input[type="date"],

textarea,

select.sort-dropdown {

width: calc(50% - 10px);

padding: 10px;

margin: 5px;

border: 1px solid #ccc;

border-radius: 4px;

}

textarea {

height: 80px;

}

button {

padding: 10px 20px;

margin: 5px;

border: none;

background-color: #007bff;

color: white;

border-radius: 4px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

.project-list {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));

gap: 20px;

}

.project-card {

background-color: #f9f9f9;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

display: flex;

flex-direction: column;

}

.project-card .project-title {

font-size: 1.5em;

color: #007bff;

margin-bottom: 10px;

}

.project-card .project-description {

font-size: 1em;

margin-bottom: 10px;

}

.project-card .project-meta {

font-size: 0.9em;

margin-bottom: 10px;

}

.project-card .project-status {

font-weight: bold;

color: green;

}

.project-card.completed .project-status {

text-decoration: line-through;

color: gray;

}

.action-buttons {

display: flex;

justify-content: space-between;

}

.action-buttons button {

background-color: #28a745;

}

.action-buttons button.delete-btn {

background-color: #dc3545;

}

.action-buttons button:hover {

opacity: 0.8;

}

.error-message {

color: red;

font-size: 0.9em;

margin-left: 5px;

}

.form-group {

display: flex;

flex-direction: column;

}

**Step 3: JavaScript Logic**

// script.js

document.addEventListener('DOMContentLoaded', function() {

const form = document.getElementById('projectForm');

const projectTitle = document.getElementById('projectTitle');

const projectDescription = document.getElementById('projectDescription');

const projectDeadline = document.getElementById('projectDeadline');

const projectPriority = document.getElementById('projectPriority');

const projectCategory = document.getElementById('projectCategory');

const filterInput = document.getElementById('filterInput');

const sortBy = document.getElementById('sortBy');

const projectList = document.getElementById('projectList');

const titleError = document.getElementById('titleError');

const deadlineError = document.getElementById('deadlineError');

const categoryError = document.getElementById('categoryError');

let projects = [];

let editIndex = -1;

form.addEventListener('submit', function(event) {

event.preventDefault();

clearErrors();

const title = projectTitle.value.trim();

const description = projectDescription.value.trim();

const deadline = projectDeadline.value;

const priority = projectPriority.value;

const category = projectCategory.value.trim();

if (validateForm(title, deadline, category)) {

if (editIndex === -1) {

projects.push({ title, description, deadline, priority, category, completed: false });

} else {

projects[editIndex] = { title, description, deadline, priority, category, completed: false };

editIndex = -1;

}

resetForm();

renderProjects();

}

});

function validateForm(title, deadline, category) {

let isValid = true;

if (!title) {

titleError.textContent = 'Project title is required.';

isValid = false;

}

if (!deadline) {

deadlineError.textContent = 'Please select a deadline.';

isValid = false;

}

if (!category) {

categoryError.textContent = 'Category is required.';

isValid = false;

}

return isValid;

}

function clearErrors() {

titleError.textContent = '';

deadlineError.textContent = '';

categoryError.textContent = '';

}

function resetForm() {

projectTitle.value = '';

projectDescription.value = '';

projectDeadline.value = '';

projectPriority.value = 'low';

projectCategory.value = '';

document.getElementById('addProjectBtn').textContent = 'Add Project';

}

function renderProjects() {

projectList.innerHTML = '';

const sortedProjects = getSortedProjects();

sortedProjects.forEach((project, index) => {

const projectCard = document.createElement('div');

projectCard.classList.add('project-card');

if (project.completed) projectCard.classList.add('completed');

projectCard.innerHTML = `

<div class="project-title">${project.title}</div>

<div class="project-description">${project.description}</div>

<div class="project-meta">

<span>Deadline: ${project.deadline}</span> |

<span>Priority: ${project.priority}</span> |

<span>Category: ${project.category}</span>

</div>

<div class="project-status">

${project.completed ? 'Completed' : 'In Progress'}

</div>

<div class="action-buttons">

<button onclick="markCompleted(${index})">${project.completed ? 'Undo' : 'Complete'}</button>

<button onclick="editProject(${index})">Edit</button>

<button class="delete-btn" onclick="deleteProject(${index})">Delete</button>

</div>

`;

projectList.appendChild(projectCard);

});

}

function getSortedProjects() {

let filteredProjects = projects.filter(project =>

project.title.toLowerCase().includes(filterInput.value.toLowerCase()) ||

project.category.toLowerCase().includes(filterInput.value.toLowerCase()) ||

project.priority.toLowerCase().includes(filterInput.value.toLowerCase())

);

return filteredProjects.sort((a, b) => {

if (sortBy.value === 'priority') {

const priorityOrder = { 'low': 1, 'medium': 2, 'high': 3 };

return priorityOrder[a.priority] - priorityOrder[b.priority];

} else if (sortBy.value === 'deadline') {

return new Date(a.deadline) - new Date(b.deadline);

}

});

}

filterInput.addEventListener('input', renderProjects);

sortBy.addEventListener('change', renderProjects);

window.markCompleted = function(index) {

projects[index].completed = !projects[index].completed;

renderProjects();

};

window.editProject = function(index) {

const project = projects[index];

projectTitle.value = project.title;

projectDescription.value = project.description;

projectDeadline.value = project.deadline;

projectPriority.value = project.priority;

projectCategory.value = project.category;

document.getElementById('addProjectBtn').textContent = 'Update Project';

editIndex = index;

};

window.deleteProject = function(index) {

projects.splice(index, 1);

renderProjects();

};

});

**Explanation and Functionality:**

1. **HTML Structure**:
   * The form allows users to add projects with fields for title, description, deadline, priority, and category.
   * A filter input and sort dropdown allow users to filter and sort projects by various criteria.
   * A project list dynamically displays projects using <div> elements, styled as cards.
2. **CSS Styling**:
   * CSS styles the form, project cards, buttons, and error messages.
   * Completed projects are styled with a strikethrough and gray text, while ongoing projects are styled normally.
3. **JavaScript Logic**:
   * **Form Validation**: The form ensures that mandatory fields are filled out before submitting.
   * **CRUD Operations**: Users can add, edit, delete, and mark projects as completed or undo the completion.
   * **Filtering and Sorting**: The project list can be filtered by title, category, or priority, and sorted by priority or deadline.
   * **Dynamic Project Rendering**: The project list updates dynamically based on the current state of the projects array.
   * **No <table> Usage**: The layout is entirely built using <div> elements and CSS Grid, providing a more modern and flexible structure.

This exercise further enhances your understanding of working with arrays in JavaScript, handling dynamic UI elements, and managing user interactions in a more complex project management application.