User Story 1 – Event Management Homepage (HTML, CSS, Bootstrap)

Objective

To design a responsive homepage for an event management platform called Eventsphere which showcases the events and allows the users to register using semantic HTML5 and Bootstrap 5 for layout and styling.

Implementation Overview

- I have Used Bootstrap Grid System (container, row, col) for responsive design.
- And Implemented semantic HTML structure — header, section, footer.
- After that I have added a hero section with call-to-action ("Start Planning").
- And Created cards for upcoming events with titles, locations, and dates.
- Built a registration form with client-side validation using JavaScript.

Key Code Snippets

HTML – Semantic Structure

```
<header>
  <nav class="navbar navbar-expand-lg navbar-dark shadow-sm">
    <div class="container">
      <a class="navbar-brand fw-bold" href="#">EventSphere</a>
      ...
    </div>
  </nav>
</header>
```

CSS – Theme Styling

```
:root {
  --primary: #6366f1;
  --secondary: #8b5cf6;
  --dark: #1e293b;
  --light: #f8fafc;
}
.hero-section {
  background: linear-gradient(135deg, var(--primary),
var(--secondary));
  color: var(--white);
}
```

JavaScript – Form Validation

```
form.addEventListener('submit', e => {
  e.preventDefault();
  if (!form.checkValidity()) {
    form.classList.add('was-validated');
  } else {
    alert("Thank you! Your registration has been received.");
    form.reset();
  }
});
```

Bootstrap Components Used

- Navbar
- Cards
- Buttons
- Grid layout (container/row/columns)

Summary

I have built the homepage using Bootstrap 5 for responsive design and custom CSS variables for styling consistency. This meets the semantic and aesthetic standards side by side providing user interactivity by a validated registration form.

User Story 2 – Event Dashboard (JavaScript + ES6)

Objective

To develop an interactive Event Dashboard that dynamically displays event details and allows users to filter by category, date, or search term.
The dashboard demonstrates modern JavaScript ES6 features, Fetch API for data loading, and Bootstrap for responsive design.

Implementation Overview

- The dashboard loads the event data from a local JSON file events.json using the Fetch API with async/await.

- Implemented ES6 syntax:

    - `const` and `let` for state management
    - Arrow functions for cleaner syntax
    - Destructuring to extract event properties
    - Template literals for dynamic HTML creation

- Users can filter events by:

    - Category (via dropdown)
    - Date (via date picker)
    - Search (by title or keyword)

- Filter logic runs in real-time and updates event cards dynamically using DOM manipulation.

Key Code Snippets

🔹 Fetching Event Data

```
const fetchEvents = async () => {
  try {
    const res = await fetch('events.json');
    if (!res.ok) throw new Error('Failed to load events');
    const { events } = await res.json();
    render(events);
    setupFilters(events);
  } catch (err) {
    message.textContent = err.message;
  }
};
```

🔹 **Rendering Event Cards Dynamically**

```
const render = (data) => {
  eventList.innerHTML = '';
  if (!data.length) return message.textContent = 'No events found.';
  message.textContent = '';
  data.forEach(({ title, category, date, location }) => {
    eventList.innerHTML += `
      <div class="col-md-4">
        <div class="card shadow-sm h-100">
          <div class="card-body">
            <h5>${title}</h5>
            <p class="text-muted">${category}</p>
            <p>${new Date(date).toDateString()}</p>
            <p class="small">${location}</p>
          </div>
        </div>
      </div>`;
  });
};
```

🔹 **Filtering Events Using ES6 Arrow Functions**

```
const setupFilters = (events) => {
  const apply = () => {
    const filtered = events.filter(e => {
      const c = filters.category === 'all' || e.category ===
filters.category;
      const d = !filters.date || e.date === filters.date;
      const s = !filters.search ||
e.title.toLowerCase().includes(filters.search.toLowerCase());
      return c && d && s;
    });
    render(filtered);
  };
  document.getElementById('category').onchange = e =>
(filters.category = e.target.value, apply());
  document.getElementById('date').onchange = e => (filters.date =
e.target.value, apply());
  document.getElementById('search').oninput = e => (filters.search =
e.target.value, apply());
};
```

User Interface and Styling

- Used Bootstrap Grid (`.container`, `.row`, `.col-md-*`) for responsive layout.
- Applied custom minimal CSS for clean visual hierarchy and hover effects.
- Soft gradient text header, rounded cards, and focus glow on input elements for accessibility.

Summary

I have implemented the clean and modular JavaScript ES6 design Fetch API integration with async/await and Real-time filtering with minimal code with a responsive and visually appealing interface built using Bootstrap + custom CSS

User Story 3 – TypeScript Customer Registration Module

In this user story, a TypeScript-based module was developed to handle customer registration and management for the event platform. The implementation demonstrates core TypeScript features such as **interfaces, classes, inheritance, enums, tuples, decorators, and iterators**.

An `ICustomer` interface defines the contract for customer data. A base `Customer` class implements this interface, while a `VIPCustomer` class extends it to add special behavior. An `enum` named `TicketType` enforces strict typing for ticket categories (`Standard`, `VIP`, `Student`).

A **custom decorator** `@logMethod` was implemented using the modern TypeScript 5 decorator syntax to log method calls and outputs, showcasing metaprogramming capabilities. The `CustomerManager` class maintains a collection of customers, supports iteration through the

`Symbol.iterator` protocol, and uses a **tuple** to record a customer's name and registration date during registration.

The module compiles cleanly using `tsc`, adheres to modern ES2022 standards, and successfully outputs customer details and decorator logs in the console.

Key Features Implemented:

- Interface & Class Design with inheritance
- Enum and Tuple for type safety
- Custom method Decorator for logging
  Iterator implementation in the Manager class
- Successful compilation and runtime execution

Outcome:
The module fulfills all TypeScript-related requirements of User Story 3 by demonstrating a robust, type-safe, and extensible registration system integrated into the event platform.