Deliverable#3:

**Ibtisam Shahzad (22i-1201)**

**Haider Zia (22i-1196)**

**Ibrahim Asim (22i-1330)**

**Company : Smartsphere**

[**A- Software Project Plan 2**](#_w6etnnewgrpv)

[**Work Breakdown Structure (WBS) 2**](#_n9s671crdj6q)

[1. Project Initiation 2](#_1n07t11l7buo)

[2. System Design 2](#_q2oxq62gfoo)

[3. Frontend Development 2](#_cvu3n3uwtnmv)

[4. Backend Development 3](#_jbg3isigmbgv)

[5. Testing & QA 3](#_l20hzammcgrz)

[6. Deployment 3](#_1tn9wzwtn3ga)

[WBS Chart: 4](#_gjpeqi2e5s0a)

[**Gantt Chart: 5**](#_r8sdy643li0v)

[**B-System Architecture: 5**](#_m9lxpjs6x1il)

[**1-Identifying Subsystems: 5**](#_v0qh1yg9adyn)

[1. Authentication Subsystem 5](#_1de33b6odp6z)

[2. User Management Subsystem 5](#_ok5i4uv4j9a6)

[3. Event Management Subsystem 5](#_myzyw2bzlms4)

[4. Notification & Feedback Subsystem 6](#_xamnojaavi99)

[5. Reporting Subsystem 6](#_y28rmkj62l30)

[6. Security Subsystem 6](#_nl7ckqz9exzt)

[**2-Architecture Styles: 7**](#_m09ft01ss9gk)

[1. Layered Architecture (N-Tier Architecture) 7](#_wan4elc38wgn)

[2. Client-Server Architecture 7](#_paqwpw5cer8s)

[3. Model-View-Controller (MVC) 8](#_ua1385ssq9tr)

[4. RESTful Architecture 8](#_5eoumghnpt3t)

[**3-Deployment diagram for client deployments: 9**](#_eeq3rksyp6af)

[Key Nodes in SmartSphere Deployment: 9](#_ogimcp5tg9af)

[Deployment Flow: 10](#_3vzwrr4mb4ya)

[**4-Component Diagram: 10**](#_s83f6zk588c5)

[Key Components: 10](#_dvf7v1ax16jw)

# **A- Software Project Plan**

# Work Breakdown Structure (WBS)

## 1. Project Initiation

* 1.1 Requirement Analysis
* 1.2 Feasibility Study
* 1.3 Project Planning (Trello, GitHub Setup)

## 2. System Design

* 2.1 High-Level Architecture Design
* 2.2 UML Diagrams
  + 2.2.1 Use Case Diagram
  + 2.2.2 Class Diagram
  + 2.2.3 Sequence Diagram
  + 2.2.4 Package Diagram
* 2.3 Database Schema Design

## 3. Frontend Development

* 3.1 Authentication Pages
  + 3.1.1 Login
  + 3.1.2 Signup
* 3.2 User Dashboards
  + 3.2.1 Admin Dashboard
  + 3.2.2 Organizer Dashboard
  + 3.2.3 Participant Dashboard
* 3.3 Event Management Pages
  + 3.3.1 Create/Edit/Delete Events
  + 3.3.2 Announcements
* 3.4 Feedback & Notifications UI

## 4. Backend Development

* 4.1 API Development
  + 4.1.1 User Controller
  + 4.1.2 Event Controller
  + 4.1.3 Ticket & Feedback Controller
* 4.2 Services Implementation
* 4.3 Security Configuration (BCrypt, JWT)
* 4.4 Database Integration

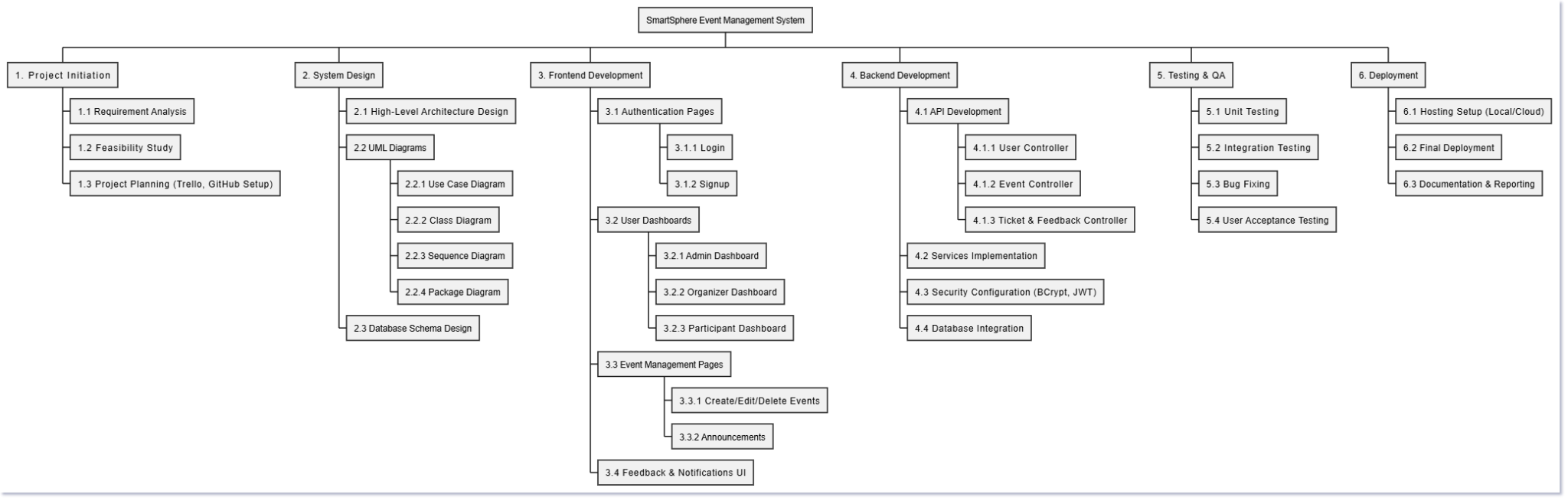
## 5. Testing & QA

* 5.1 Unit Testing
* 5.2 Integration Testing
* 5.3 Bug Fixing
* 5.4 User Acceptance Testing

## 6. Deployment

* 6.1 Hosting Setup (Local/Cloud)
* 6.2 Final Deployment
* 6.3 Documentation & Reporting

### **WBS Chart:**



## Gantt Chart:

# **B-System Architecture:**

# **1-Identifying Subsystems:**

## Authentication Subsystem

* 1. Handles login, signup, password encryption (BCrypt).
  2. Related Classes: AuthController, SecurityConfig, UserService.

## User Management Subsystem

* 1. Manages user profiles, roles (Admin, Organizer, Participant).
  2. Related Classes: UserController, UserService, UserRepository, User.

## Event Management Subsystem

* 1. Creation, modification, deletion of events by organizers.
  2. Related Classes: EventController, EventService, EventRepository, Event.

## Notification & Feedback Subsystem

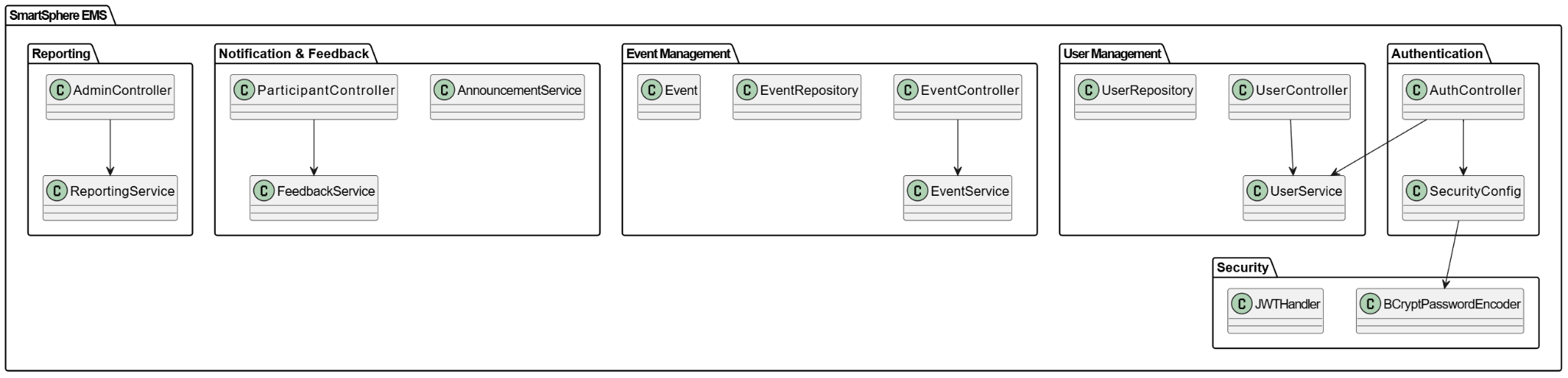
* 1. Sends announcements, collects feedback from participants.
  2. Related Classes: Announcement, Feedback, ParticipantController.

## Reporting Subsystem

* 1. Admin views reports, participant lists, feedback.
  2. Related Classes: AdminController, Reporting.

## Security Subsystem

* 1. Authentication, authorization, session management.
  2. Related Classes: SecurityConfig, JWT (if used), Spring Security.



# 

# **2-Architecture Styles:**

## 1. Layered Architecture (N-Tier Architecture)

**Description**:  
 The system is organized into **separate layers**, each with distinct responsibilities.

**Layers in SmartSphere**:

* **Presentation Layer**: React.js Frontend UI.
* **Business Logic Layer**: Spring Boot Services handling core operations.
* **Data Access Layer**: Repositories interacting with the database.
* **Database Layer**: MySQL/PostgreSQL for persistent storage.

**Why Used?**

* Clear separation of concerns.
* Easier to maintain and scale.
* Independent development of frontend and backend.

## 2. Client-Server Architecture

**Description**:  
 The system follows a **Client-Server model**, where:

* The **client (React.js)** requests resources.
* The **server (Spring Boot)** processes and responds.

**Communication**:

* Uses **HTTP/HTTPS** protocols.
* **RESTful APIs** serve as the interface between client and server.

**Why Used?**

* **Scalability**: Clients and servers can scale independently.
* **Modularity**: Frontend and backend can evolve separately.

## 3. Model-View-Controller (MVC)

**Description**:  
 The Spring Boot backend follows the **MVC pattern**:

* **Model**: Data layer (Entities like User, Event, Ticket).
* **View**: Not directly applicable (handled by React), but can include API responses.
* **Controller**: Handles HTTP requests (e.g., UserController, EventController).

**Why Used?**

* Makes the backend more organized.
* Separates data, logic, and request handling.

## 4. RESTful Architecture

**Description**:  
 Backend services expose **RESTful APIs**:

* CRUD operations via **HTTP verbs** (GET, POST, PUT, DELETE).
* Stateless interactions.

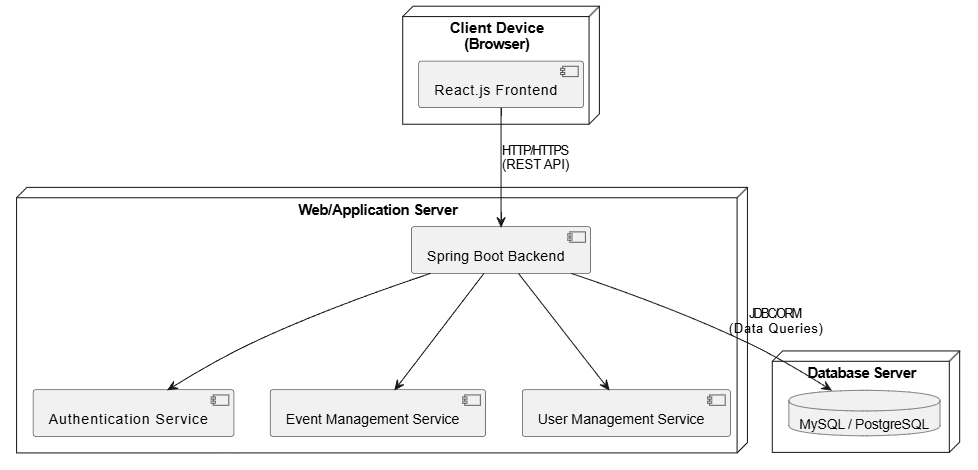
**Why Used?**

* **Interoperability**: Easily integrates with any frontend.
* **Scalability**: Lightweight, ideal for distributed systems.

# **3-Deployment diagram for client deployments:**

## Key Nodes in SmartSphere Deployment:

1. **Client Node** (User’s Browser)
   * Runs the **React.js Frontend**.
   * Interacts with the backend via **RESTful APIs**.
2. **Web Server Node**
   * Hosts **static React build** (Optional if hosted separately).
3. **Application Server Node**
   * Runs **Spring Boot Backend** (APIs).
   * Handles business logic, authentication, event management, etc.
4. **Database Server Node**
   * Stores data in **MySQL/PostgreSQL**.
   * Accessed by the Application Server.



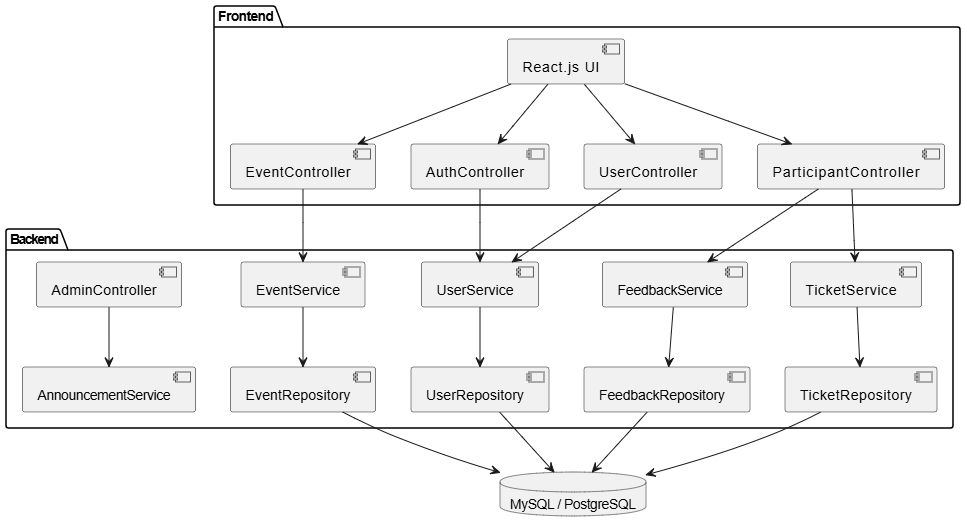
## **Deployment Flow:**

1. User accesses **frontend UI** from the browser.
2. UI sends **HTTP requests** to **Spring Boot REST API**.
3. Backend processes requests and queries **database**.
4. Responses are sent back to the frontend.

# **4-Component Diagram:**

## **Key Components:**

1. **Frontend (React.js)**
   * Interfaces with backend via REST APIs.
2. **Spring Boot Backend Components**:  
   * **Controllers**:  
     + AuthController
     + UserController
     + EventController
     + ParticipantController
     + AdminController
   * **Services**:  
     + UserService
     + EventService
     + TicketService
     + FeedbackService
     + AnnouncementService
   * **Repositories**:  
     + UserRepository
     + EventRepository
     + TicketRepository
     + FeedbackRepository
3. **Database**:  
   * MySQL/PostgreSQL.



# **C-TestCases BlackBox:**

# **D-TestCases WhiteBox:**