PlanItRight - Event Management System

Business Requirements Document

Prepared By:
Vivekanand. R
Kiran Kumar Reddy. R

September 3, 2024

Contents

1 Project Overview

1.1 Project Name

PlanItRight Event Management System

1.2 Objective

The objective of the PlanItRight Event Management System is to develop a comprehensive platform that allows users to manage events, guests, tasks, vendors, and budgets efficiently. The system aims to streamline event planning processes and improve overall user experience by providing an integrated solution for event management.

2 Scope

2.1 In-Scope

- Event management functionalities including creation, modification, and deletion of events.
- Guest management with features for invitation, RSVP tracking, and guest list management.
- Task management, allowing users to create, assign, and track tasks associated with events.
- Vendor and budget management, including vendor selection, contract management, payment processing, and budget tracking.
- Integration with external systems such as payment gateways and communication platforms.

2.2 Out-of-Scope

- Marketing and promotional activities outside the core event management functionality.
- Non-essential features such as social media integrations and AI-based event planning recommendations.

3 Functional Requirements

1. **Event Management:**
    - Create, update, and delete events.
    - Define event details such as date, location, and description.
2. **Guest Management:**
    - Import guest lists.

- Send invitations and track RSVPs.

3. **Task Management:**
   - Create tasks, assign them to team members, and set deadlines.
   - Monitor task progress through a dashboard.
4. **Vendor and Budget Management:**
   - Add and manage vendors, track contracts, and process payments.
   - Track budgets and monitor spending against allocated funds.

4 Non-Functional Requirements

1. **Performance:** The system should handle up to 10,000 concurrent users without significant performance degradation.
2. **Scalability:** The system must be scalable to accommodate additional users and features as needed.
3. **Reliability:** The system should maintain a 99.9% uptime, with robust backup and disaster recovery plans.
4. **Security:** The system should implement JWT (JSON Web Token) authentication for secure and stateless communication, with data encryption to protect sensitive information both at rest and in transit.

5 Technologies Used

- **Frontend:** Angular, HTML5, CSS3
- **Backend:** Java Spring Boot, RESTful APIs
- **Database:** PostgreSQL
- **Hosting:** Heroku
- **Version Control:** Git, GitHub

6 High-Level Design

6.1 Architectural Overview

This section provides a high-level view of the system architecture, including the main components and how they interact with each other.

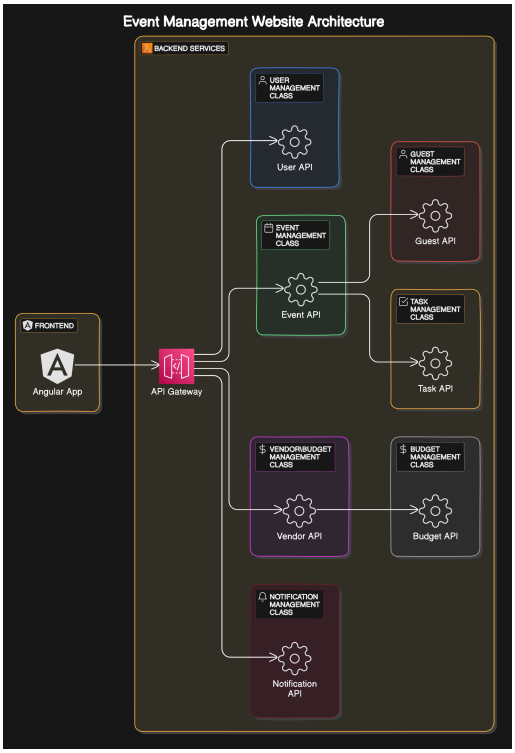

Figure 1: High-Level Architecture Diagram for PlanItRight

7 Functional Requirements

1. **Event Management:**
   - The system should allow users to create new events by specifying details such as event name, description, date, time, and location.
   - Users should be able to update the details of an existing event, including rescheduling or canceling the event.
   - The system should enable users to delete events. Deleted events should be permanently removed from the system.
   - Event organizers can view a dashboard showing all events they are organizing, including past, current, and upcoming events.
   - The system should provide notifications to users when an event is updated or canceled.
2. **Guest Management:**
   - The system should allow users to import guest lists from CSV or Excel files.
   - Users should be able to manually add, update, and delete guests from an event guest list.
   - The system should send invitations to guests via email, and the guest should be able to RSVP (Yes, No, Maybe).
   - Users should be able to track RSVP statuses for all invited guests in real-time.

- The system should send reminders to guests who haven't responded by a certain deadline.

3. **Task Management:**
    - The system should allow event organizers to create tasks associated with an event.
    - Organizers can assign tasks to team members and set due dates for each task.
    - The system should provide a task management dashboard that displays the status of all tasks (e.g., To Do, In Progress, Completed).
    - Task assignees should receive notifications when they are assigned a task or when the task is due soon.
    - Users should be able to update the progress of tasks and mark them as completed.
4. **Vendor and Budget Management:**
    - The system should allow users to add vendors associated with an event, including vendor name, contact information, and service type (e.g., catering, photography).
    - Users should be able to track contracts and payment statuses with vendors.
    - The system should allow event organizers to allocate a budget for an event and track expenses.
    - Users should receive notifications when expenses are approaching or exceeding the allocated budget.
    - The system should provide a detailed view of budget breakdown, including total allocated, spent, and remaining amounts.

7.1 System Interactions

Each component will interact with others via APIs, enabling a modular and flexible system. Below is a wireframe diagram illustrating how the components interact with each other and how the user interfaces are connected.

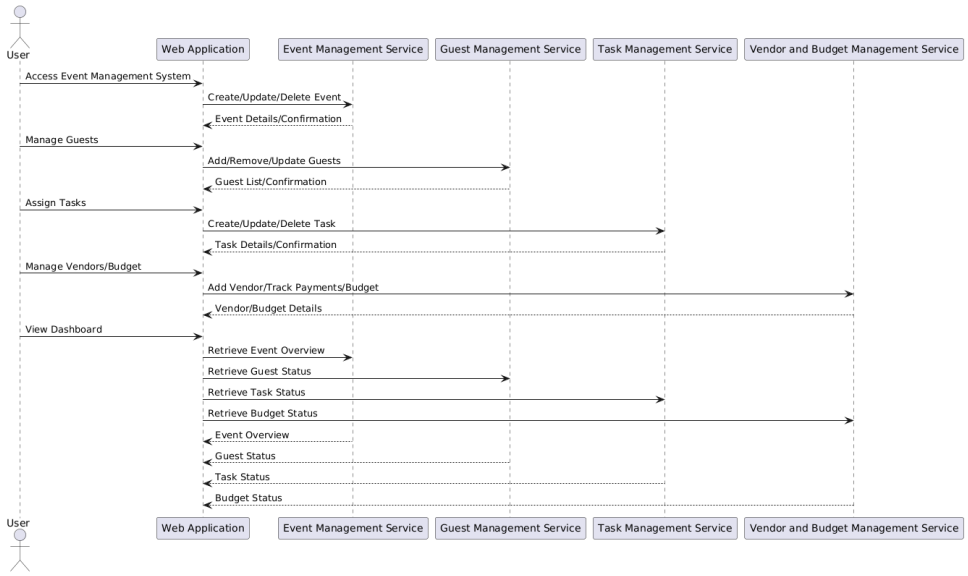

Figure 2: System Interaction Diagram

7.2 Performance Considerations

The system is optimized to handle high concurrency, with caching mechanisms in place for frequently accessed data. Database queries are optimized for speed, and the system includes load balancing to distribute traffic efficiently.

7.3 Technology Stack

The application leverages a technology stack consisting of React.js for the frontend, Spring Boot for the backend, and MySQL for the database. The system is hosted on AWS and uses Docker for containerization.

8 Low-Level Design

8.1 Detailed Component Specification

8.1.1 Event Management Service

- **Endpoints:**
  - `POST /events` - Create a new event
  - `GET /events/{id}` - Retrieve event details by ID
  - `PUT /events/{id}` - Update an existing event
  - `DELETE /events/{id}` - Delete an event

8.1.2 Guest Management Service

- **Endpoints:**
  - `POST /guests` - Add a new guest to an event

- `GET /guests/{eventId}` - Retrieve guests for a specific event
- `PUT /guests/{id}` - Update guest details
- `DELETE /guests/{id}` - Remove a guest

8.1.3 Task Management Service

- **Endpoints:**
  - `POST /tasks` - Create a new task
  - `GET /tasks/{eventId}` - Retrieve tasks associated with an event
  - `PUT /tasks/{id}` - Update task details
  - `DELETE /tasks/{id}` - Delete a task

8.1.4 Vendor and Budget Management Service

- **Endpoints:**
  - `POST /vendors` - Add a new vendor
  - `GET /vendors/{id}` - Retrieve vendor details
  - `POST /payments` - Process a payment
  - `GET /budget/{eventId}` - Retrieve budget details for an event

8.2 Interface Definitions

Each service exposes a RESTful API, with clear endpoint definitions for CRUD operations (Create, Read, Update, Delete). The APIs are designed to be stateless and secure, using JWT (JSON Web Tokens) for authentication and authorization.

8.3 Resource Allocation

The application is designed to efficiently use resources, with each microservice being allocated appropriate compute and memory resources based on expected load. Resource allocation is managed through Kubernetes, enabling dynamic scaling as needed.

8.4 Error Handling and Recovery

The system includes comprehensive error handling mechanisms. Each service returns appropriate HTTP status codes for different types of errors (e.g., 404 for not found, 500 for server errors).

The system is designed to be resilient, with automatic recovery mechanisms in place for critical failures.

8.5 Security Considerations

Security is a top priority, with multiple layers of protection including:

- JWT (JSON Web Token) authentication for secure and stateless communication
- Data encryption (both at rest and in transit)
- Secure API communication using HTTPS
- Regular security audits and penetration testing

8.6 Code Structure

The codebase follows the MVC (Model-View-Controller) architecture, with a clear separation of concerns. Each microservice is organized into layers: Controller, Service, Repository, and Model. This structure facilitates maintainability and scalability.

9 ER Diagrams

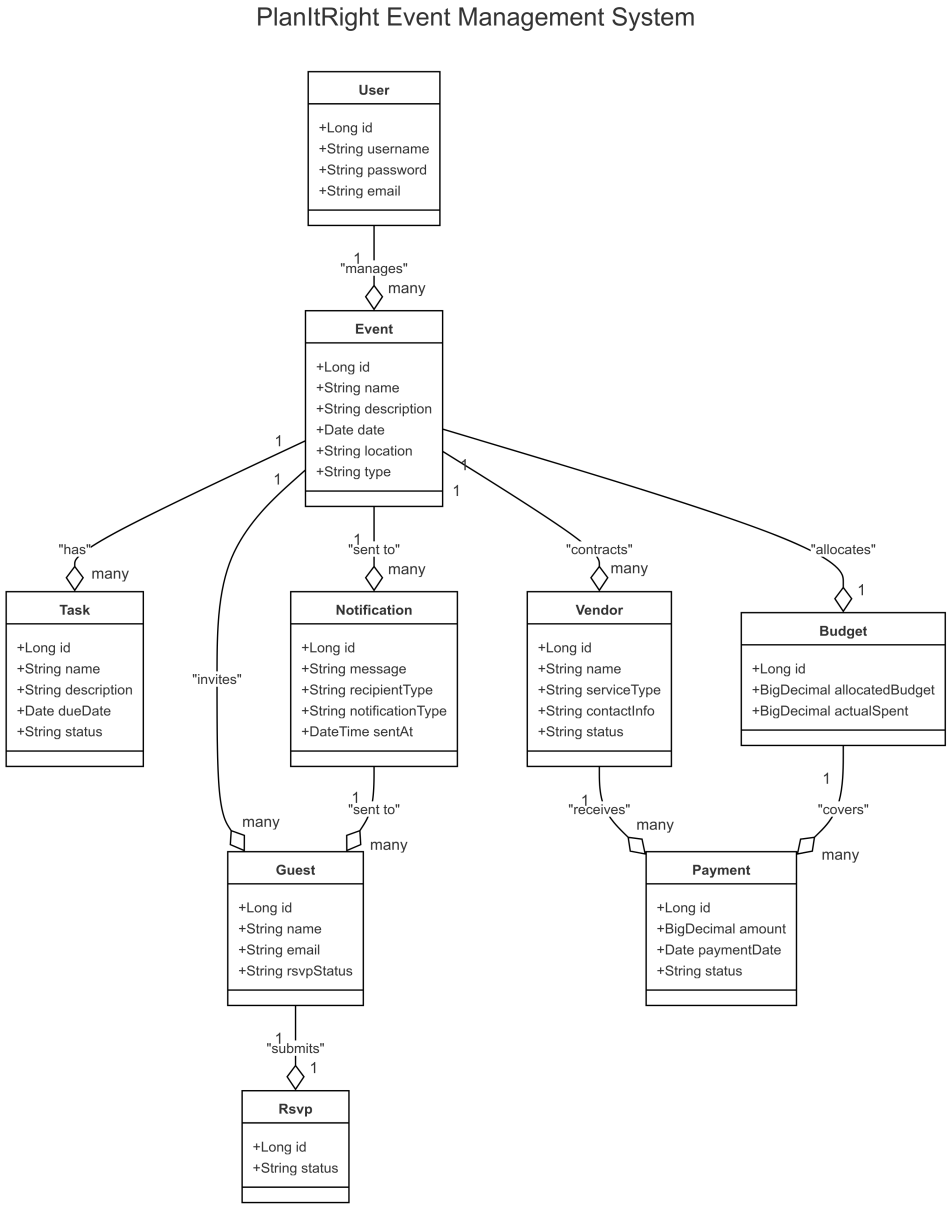

Figure 3: Entity-Relationship Diagram for PlanItRight

10 Class Diagrams

Figure 4: Class Diagram for PlanItRight

11 Sequence Diagram

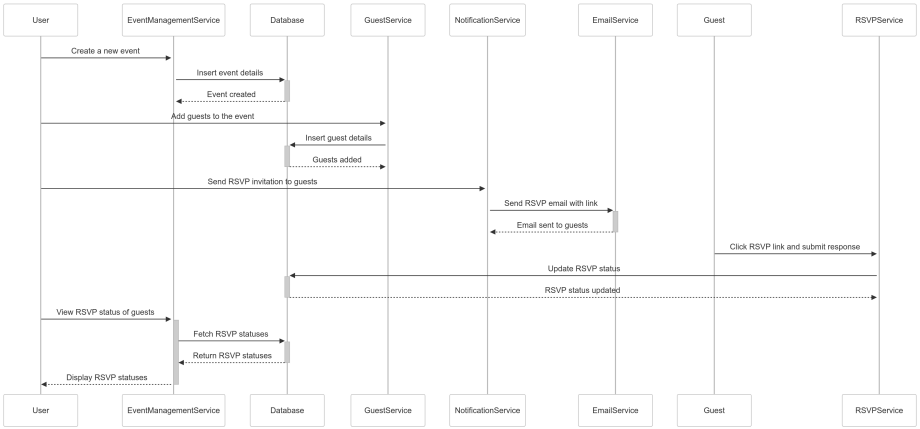

Figure 5: Sequence Diagram for PlanItRight

12 Conclusion

The PlanItRight Event Management System is designed to be a robust, scalable, and user-friendly platform that meets the needs of modern event planners. With a microservices architecture, secure APIs, and a focus on performance, the system is well-equipped to handle the demands of large-scale events. This document outlines the high-level and low-level designs, ensuring that all aspects of the system are covered, from initial requirements to detailed implementation.