**Technical Document for Eventure**

**1. Introduction**

**1.1 Project Overview**

Eventure is an event management platform designed to streamline the creation, management, and participation in events. The platform allows users to create events, assign tasks, manage participants, and track event progress. Users are classified into three roles: Admin, Organizer, and Participant, each with specific permissions and capabilities within the platform.

**1.2 Purpose**

This document provides a technical overview of the Eventure platform, detailing its architecture, components, and functionalities. It is intended for developers, system architects, and technical stakeholders interested in understanding the inner workings of the platform.

**2. System Architecture**

**2.1 Overview**

Eventure is a full-stack application with a clear separation between the frontend and backend. The system architecture is divided into three main components:

1. **Frontend**: Developed using React.
2. **Backend**:
   * **Node.js with Express**: Handles user authentication and tasks, participants management.
   * **Java with Spring Boot**: Manages event creation, task assignments, and participant management.

**3. System Components**

**3.1 Frontend**

**3.1.1 Overview**

The frontend of Eventure is built using React. It serves as the user interface for all user interactions, including event creation, task assignment, and participant management.

**3.1.2 Key Features**

* **Dashboard**: Displays an overview of events, tasks, and invites relevant to the user.
* **Event Management**: Allows users to create, view, and edit events.
* **Task Management**: Enables users to view, update, and filter tasks based on their status (Waiting, In Process, Done).
* **Participant Management**: Provides the functionality for users to accept or reject event invitations.
* **Calendar Integration**: Users can filter events by date using a calendar feature.

**3.2 Backend**

**3.2.1 Node.js with Express**

**3.2.1.1 Overview**

The Node.js backend handles the authentication flow and user role management. It is responsible for creating user accounts and authenticating users to allow them to manage or participate in events.

**3.2.1.2 Key Features**

* **Authentication**: Manages user sign-up, login, and session handling.
* **Role Management**: Determines the user's role (Admin, Organizer, Participant) within the context of an event.

**3.2.1.3 API Endpoints**

* **POST /auth/sign-up**: Create a new user account.
* **POST /auth/login**: Authenticate a user and generate a access and refresh token.
* **GET /auth/name-check**: Check user name exist or not.
* **POST /auth/refresh-token**: Refresh access token with refresh token
* **POST /auth/change-password**: Change current user password.
* **GET** **/users**: Get all users .
* **GET** **/tasks/my-tasks**: Get current user’s assigned tasks.
* **POST** **/tasks/:taskId/update-status**: Update task status.
* **GET** **/participants/my-particiapnts**: Get current user’s participants.
* **POST** **/participants/:participantId/update-status**: Update participant status.

**3.2.2 Java with Spring Boot**

**3.2.2.1 Overview**

The Java Spring Boot backend manages event-related data, including event creation, task assignments, and participant management.

**3.2.2.2 Key Features**

* **Event Management**: Handles the creation, updating, and deletion of events.
* **Task Management**: Allows admins to create and assign tasks to users, and enables organizers to update task statuses.
* **Participant Management**: Manages participant invites and their responses.

**3.2.2.3 API Endpoints**

* **POST /api/events/my-events**: Get current user’s Events.
* **GET /events/{eventId}**: Retrieve details of a specific event.
* **PUT /events/save**: Create or Update an event.

**4. User Roles and Permissions**

**4.1 Admin**

* **Create/Edit Events**: Admins can create new events and modify existing ones.
* **Task Assignment**: Admins can create tasks and assign them to any user.
* **Participant Management**: Admins can invite users to events as participants or organizers.

**4.2 Organizer**

* **Task Management**: Organizers can view and update the status of tasks assigned to them.

**4.3 Participant**

* **Event Participation**: Participants can view events they have been invited to and accept or decline invitations.

**5. User Authentication**

**5.1 Authentication Flow**

1. **User Registration**: Users must create an account through the Node.js backend by providing necessary credentials.
2. **User Login**: Users log in using their credentials, and a session token is generated.

**6. Dashboard Functionality**

**6.1 Overview**

The dashboard is the central hub for users, providing a consolidated view of all events, tasks, and invites.

**6.2 Event Section**

* **Event List**: Displays all events that the user is involved in, either as an admin, organizer, or participant.
* **Event Filters**: Users can filter events based on their start date.

**6.3 Task Section**

* **Task List**: Displays tasks assigned to the user.
* **Task Filters**: Allows filtering tasks based on their status (Waiting, In Process, Done).

**6.4 Event Invites**

* **Invite List**: Shows all event invitations that the user has received.
* **Invite Management**: Users can accept or reject invites, with the ability to filter invites based on their status (Waiting, Accepted, Rejected).

**6.5 Calendar Integration**

* **Date-Based Filtering**: Users can filter events by selecting a date from the calendar.

**7. Running the Application and Deployment Services**

**7.1 Running the Application**

1. **Navigate to the Docker Compose Directory**: Open your terminal and navigate to the directory where the docker-compose.yml file is located.

cd /path/to/docker-compose

1. **Build the Docker Images**: Before running the services, build the necessary Docker images. This step ensures that the latest changes in your codebase are reflected in the containers.

docker-compose build

1. **Run the Application**: Use Docker Compose to start all services defined in the docker-compose.yml file. This will include both the application and the deployment service.

docker-compose up

If you want to run the services in the background (detached mode), use the -d flag:

docker-compose up -d

1. **Access the Application**: Once the services are running, you can access the application through your web browser at http://localhost. The frontend service is configured to run on port 80, so no port number is required in the URL.

yaml

Copy code

services:

eventure-frontend:

build: ../eventure-frontend/.

ports:

- "80:80"

If you need to use a different port, update the ports section in the docker-compose.yml file accordingly.