EVENT BOOKING WEB SERVICE API DOCUMENTATION

1. Project Overview (technology stacks)

This project is backend web service that will facilitate the online booking of wellness events (health talks, onsite screenings, etc) and the vendor approval or rejection of said events. Company Human Resource accounts will create the events with 3 proposed dates and 1 proposed location while the Vendor Accounts (vendors are tagged to different event items) will reject (specify a reason) or approve (select a date) the event.

Technology Stacks:

- **Framework**: Spring Boot, Spring Data JPA, Spring Security, Spring Web (REST)
- **Database**: PostgreSQL, Spring Data JPA (for managing interaction with Database), Hibernate
- **Security**: Spring Security, JWT (Json Web Token), Spring Security JWT (managing authentication and authorization)
- **Error Handling & Validation**: Spring Validation and Custom Exception Handling with Controller Advice
- **Logging**: SLF4J

2. Database Schema

This section outlines the structure of the database used in this backend application, including details about tables, relationships, and relevant configuration. The application uses PostgreSQL relational database to store data and manage persistent entities. The database is designed with the following entities:

- **Account** (id, username, password, role)

  Relationships:

  - One-to-One with **Company** and **Vendor** (each account is either a company or a vendor)

- **Company** (id, name, user_id (FK), company_code)

  Relationships:

  - One-to-One with **Account**
  - One-to-Many with **Event** (a company can create multiple events)

- **Vendor** (id, name, user_id (FK))

  Relationships:

  - One-to-One with **Account**
  - One-to-Many with **Event** (a vendor can approve or reject multiple events created by companies)

- **Event** (id, event_name, description, date_created, company_id (FK), vendor_id (FK), location_id (FK), event_status_id (FK), remarks)

  Relationships:

  - Many-to-One with **Company** (multiple events created by a company)
  - Many-to-One with **Vendor** (multiple events managed by a vendor)
  - Many-to-One with **Location** (each event is assigned a location)
  - One-to-Many with **Event Status** (an event can have multiple status updates)
  - One-to-Many with **Event Date** (an event has multiple proposed dates)

- **Location** (id, address, name, postal_code)

  Relationship:

  - One-to-Many with **Event** (a location can be assigned to multiple events)

- **Event Status** (id, status_name)

  Relationship:

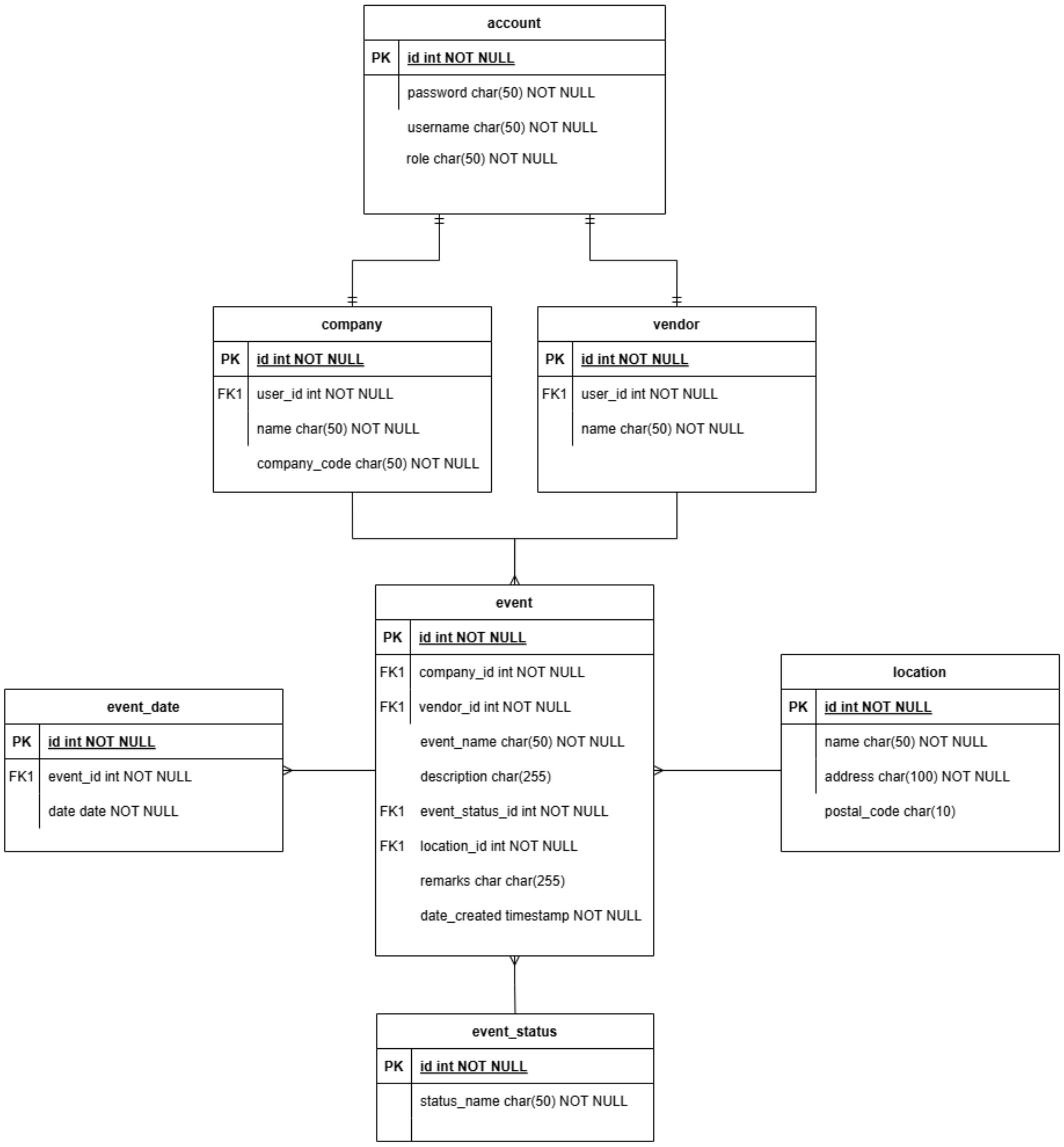
  - One-to-Many with **Event** (a status can be referenced by multiple events)

- **Event Date** (id, date, event_id(FK))

  Relationships:

  - Many-to-One with **Event** (each even date is tied to an event and an event can have multiple proposed dates)

ERD DIAGRAM


account
PK id int NOT NULL
password char(50) NOT NULL
username char(50) NOT NULL
role char(50) NOT NULL
company
PK id int NOT NULL
FK1 user_id int NOT NULL
name char(50) NOT NULL
company_code char(50) NOT NULL
vendor
PK id int NOT NULL
FK1 user_id int NOT NULL
name char(50) NOT NULL
event
PK id int NOT NULL
FK1 company_id int NOT NULL
FK1 vendor_id int NOT NULL
event_name char(50) NOT NULL
description char(255)
FK1 event_status_id int NOT NULL
FK1 location_id int NOT NULL
remarks char char(255)
date_created timestamp NOT NULL
event_date
PK id int NOT NULL
FK1 event_id int NOT NULL
date date NOT NULL
location
PK id int NOT NULL
name char(50) NOT NULL
address char(100) NOT NULL
postal_code char(10)
event_status
PK id int NOT NULL
status_name char(50) NOT NULL

3. Project Structure

Following are details on the project structure:

Packages and Files	Description
src	Contains source file and packages
src/config	Contains configuration classes for the application (SecurityConfig, WebConfig)
src/constants	Contains constant values
src/controllers	Contains rest controllers for handling HTTP request
src/dto	Contains Data Transfer Objects (DTO) for representation of data to application layer (request DTOs and response DTOs)
src/exceptiom	Contains custom exception classes to handle specific errors and manage global exception handling.
src/factory	Contains factory classes and methods to create complex objects
src/filter	Contains classes for filtering requests or responses (e.g. jwt authentication filters)
src/mappers	Contains mapper classes for converting entities to DTOs and vice versa
src/models	Contains entity classes that map to database tables
src/repositories	Contains repository interfaces for database access
src/services	Contains service classes that implement business logic
src/utils	Contains utility/helper classes with commonly used methods
src/validations	Contains validation classes or annotations
resources/data.sql	SQL file for initializing the database with default data upon application startup
resources/application.properties	The main configuration file for setting environment-specific properties such as database URL and other application settings
build.gradle.kts	Gradle Kotlin build script file to manage dependencies and project configuration

4. Installation Guide

4.1. Clone Repository

```
git clone https://github.com/MelisaApriliani/OnlineEventsBooking.git
    cd OnlineEventsBooking\backend
```

4.2. Install Java & Gradle

Ensure Java (JDK 17 or above) and Gradle are installed on your machine.

```
java -version
gradle -version
```

4.3. Install PostgreSQL

If PostgreSQL is not installed, download and install it from https://www.postgresql.org/download/

Setup PostgreSQL:

- Open PostgreSQL CLI or PGAdmin to create a new database:
  ```
  psql -U {postgre_username}
  CREATE DATABASE eventsbookingdb;
  ```

- Create a PostgreSQL User (if not using the default postgres user)
  ```
  CREATE USER event_user WITH PASSWORD 'event_password';
  ```

4.4. Configure Application Properties

Open file resources/application.properties and configure the following details:

```
spring.application.name=eventsbooking
spring.datasource.url=jdbc:postgresql://localhost:5432/eventsbookingdb
spring.datasource.username=postgres <= change to your postgre account username
spring.datasource.password= password <= set you postgre account password
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update

# Enable execution of SQL scripts for testing only(schema.sql and data.sql)
spring.sql.init.mode=always

# Ensure data.sql is executed after hibernate has crated all the tables
spring.jpa.defer-datasource-initialization=true
spring.sql.init.data-locations=classpath:data.sql

spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.web.servlet.error.ErrorMvcAut
oConfiguration
```

4.5. Build & Run the project

You may build and run the project either via command line or IDE (recommended: IntelliJ)

```
./gradlew build

./gradlew bootRun
```

5. API Doc

For testing purpose of this web service API, a set of predefined records are populated on the DB tables upon Application startup. Here are some credentials you may use for testing:

ADMIN (Vendor):

admin1@test.com (password: password123)
admin2@test.com (password: password123)
admin3@test.com (password: password123)

User (Company HR):

user1@test.com (password: password123)
user2@test.com (password: password123)
user3@test.com (password: password123)

POST Authentication
{base_url}/ auth/login

Sample request:
http://localhost:8080/auth/login
Request body:
```
{
   "username": "user2@test.com",
   "password": "password123"
}
```
Reponse:
```
{
   "jwt":
"eyJhbGciOiJIUzI1NiJ9.eyJyb2xlcyI6WyJST0xFX1VTRVIiXSwic3ViIjoidXNlcjJAdGVzdC5jb20iLCJpYXQiOjE3M
zEyODM4NTgsImV4cCI6MTczMTMxOTg1OH0.npFonRg42OyAE4IZ4edNAy4CBVsjvw_9TuwrRT4vclg"
}
```

GET User Details
{base_url/api/user/details

Sample request:
http://localhost:8080/api/user/details
Header:
```
{
   Authorization: Bearer {token}
   ContentType: application/json
}
```

Reponse:
```
{
   "status": 200,
   "message": "User details retrieved successfully.",
   "data": {
      "businessId": 2,
      "businessEntityName": "RANS Corporation",
      "role": "USER"
   }
}
```

GET All Vendors
{base_url/api/vendor

Sample request:
http://localhost:8080/api/vendor
Header:
```
{
   Authorization: Bearer {token}
   ContentType: application/json
}
```
Response:
```
{
   "status": 200,
   "message": "Vendors retrieved successfully.",
   "data": [
      {
         "businessId": 1,
         "businessEntityName": "Zenith Wellness Co.",
      },
      {
         "businessId": 2,
         "businessEntityName": "Tranquil Moments",
      },
      {
         "businessId": 3,
         "businessEntityName": "VitalEssence Wellness",
      }
   ]
}
```

GET All Locations
{base_url/api/location

Sample request:
http://localhost:8080/api/location
Header:
```
{
   Authorization: Bearer {token}
   ContentType: application/json
}
```
Response:
{"status":200,"message":"Locations retrieved successfully.","data":[{"id":1,"name":"CGC Hall","address":"5th Avenue, New York City","postalCode":"10001"},{"id":2,"name":"XYZ Convention Centre","address":"Main Street, Los Angeles","postalCode":"90001"},{"id":3,"name":"One Avenue","address":"Michigan Avenue, Chicago","postalCode":"60601"},{"id":4,"name":"City Hospital","address":"Texas Street, Houston","postalCode":"77001"},{"id":5,"name":"BYD Hall","address":"Central Avenue, Phoenix","postalCode":"85001"},{"id":6,"name":"UNITEN","address":"Market Street, Philadelphia","postalCode":"19101"},{"id":7,"name":"San Antonio","address":"Alamo Plaza, San Antonio","postalCode":"78201"},{"id":8,"name":"Saujana Resort","address":"Petaling Jaya, Ara Damansara","postalCode":"123124"}]}

POST Add New Location
{base_url/api/location/create

Sample request:
http://localhost:8080/api/location/create
Header:
```
{
   Authorization: Bearer {token}
   ContentType: application/json
}
```
Request Body:
```
{
   "name":"Berjaya Park",
   "address":"1232 Street, Genting Highland",
   "postalCode":"10001"
}
```

Reponse:
```
{
   "status": 201,
   "message": "Location added successfully.",
   "data": {
      "id": 11,
      "name": "Berjaya Park",
      "address": "1232 Street, Genting Highland",
      "postalCode": "10001"
   }
}
```

POST Create Event
{base_url/api/ event/create

Sample request:
http://localhost:8080/api/event/create
Header:
```
{
   Authorization: Bearer {token}
   ContentType: application/json
```
Request Body:
```
{
   "eventName":"Mega Yoga",
   "description":"This yoga event for all women employee at company 2",
   "companyId":2,
   "vendorId":1,
   "dateCreated":"2024-11-10",
   "location":{
      "name":"Saujana Resort",
      "postalCode": 123124,
      "address":"Petaling Jaya, Ara Damansara"
   },
   "eventDates":[{
      "date":"2024-11-20"
   },
   {
      "date":"2024-11-22"
   },
   {
      "date":"2024-11-28"
   }]
}
```

Reponse:

```
{
  "status": 201,
  "message": "Event created successfully.",
  "data": {
    "eventId": 3,
    "companyId": 2,
    "eventName": "Mega Yoga",
    "description": "This yoga event for all women employee at company 2",
    "location": {
      "id": 10,
      "name": "Saujana Resort",
      "address": "Petaling Jaya, Ara Damansara",
      "postalCode": "123124"
    },
    "dateCreated": "2024-11-10",
    "eventDates": [
      {
        "id": 3,
        "date": "2024-11-20"
      },
      {
        "id": 4,
        "date": "2024-11-22"
      },
      {
        "id": 5,
        "date": "2024-11-28"
      }
    ],
    "status": {},
    "remarks": null,
    "vendor": {
      "businessId": 1,
      "businessEntityName": null,
      "role": null
    }
  }
}
```

POST Update Event
{base_url/api/event/update

Sample request:
http://localhost:8080/api/event/update
Header:
```
{
   Authorization: Bearer {token}
   ContentType: application/json
}
```
Request Body:
```
{
   "eventId":3,
   "eventName":"Mega Yoga",
   "companyId":2,
   "vendorId":1,
   "statusId":1,
   "remarks":"updating event",
   "eventDates":[{
      "id":3,
      "date":"2024-11-20"
   }]
}
```

Reponse:
```
{
   "status": 200,
   "message": "Event updated successfully.",
   "data": {
      "eventId": 3,
      "companyId": 2,
      "eventName": "Mega Yoga",
      "description": "This yoga event for all women employee at company 2",
      "location": {
         "id": 10,
         "name": "Saujana Resort",
         "address": "Petaling Jaya, Ara Damansara",
         "postalCode": "123124"
      },
      "dateCreated": "2024-11-10",
      "eventDates": [
         {
            "id": 3,
            "date": "2024-11-20"
         }
      ],
      "status": {},
      "remarks": null,
        "vendor": {
          "businessId": 1,
          "businessEntityName": "Zenith Wellness Co.",
          "role": null
        }
    }
}
```

GET Events by Company
{base_url/api/event/company/{company_id}

Sample request:
http://localhost:8080/api/event/company/2
Header:

```
{
   Authorization: Bearer {token}
   ContentType: application/json
}
```

Reponse:

```
{
   "status": 200,
   "message": "Event details list retrieved successfully.",
   "data": [
      {
         "eventId": 3,
         "companyId": 2,
         "eventName": "Mega Yoga",
         "description": "This yoga event for all women employee at company 2",
         "location": {
            "id": 10,
            "name": "Saujana Resort",
            "address": "Petaling Jaya, Ara Damansara",
            "postalCode": "123124"
         },
         "dateCreated": "2024-11-10",
         "eventDates": [
            {
               "id": 3,
               "date": "2024-11-20"
            }
         ],
        "eventStatus": {
          "id": 1,
          "statusName": "Approved"
        },
        "remarks": null,
        "vendor": {
          "businessId": 1,
          "businessEntityName": "Zenith Wellness Co.",
          "role": null
        }
      }
    ]
}
```

GET Events by Vendor
{base_url/api/event/vendor/{vendor_id}

Sample request:
http://localhost:8080/api/event/company/2

Header:

```
{
  Authorization: Bearer {token}
  ContentType: application/json
}
```

Reponse:

```
{"status":200,"message":"Event details list retrieved successfully.","data":[{"eventId":7,"companyId":1,"eventName":"Running","description":"this is running event","location":{"id":3,"name":"One Avenue","address":"Michigan Avenue, Chicago","postalCode":"60601"},"dateCreated":"2024-11-10","eventDates":[{"id":9,"date":"2024-11-29"}],"eventStatus":{"id":1,"statusName":"Approved"},"remarks":null,"vendor":{"businessId":2,"businessEntityName":"Tranquil Moments","role":null}}]}
```

6. Troubleshooting and Configuration Guide

This section provides solutions for common issues and additional configurations that may be required while developing or running this application:

6.1. CORS Errors

If you encounter a CORS (Cross-Origin Resource Sharing) error while accessing the API from the frontend, you may need to adjust the CORS configuration.

Solution:

1. Locate the WebConfig or configuration class in src/config.
2. Update the CORS mapping to allow you frontend origin:

```
public void addCorsMappings(CorsRegistry registry) {
  registry.addMapping("/**") // Allow CORS on all endpoints
        .allowedOrigins("http://localhost:5174")
        .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
        .allowedHeaders("*")
        .allowCredentials(true);
}
```

6.2. Database Connection Issues

If you encounter issues connecting to PostgreSQL, check on the following:

- Ensure Postgre is running
- Verify credentials on application.properties

6.3. Unauthorized error

If you encounter unauthorized access to certain API endpoints, ensure that you are calling the API using the correct token. A token is tagged to a username and role that specified authorized access to APIs. At the moment, **POST Update Event** and **GET Event Details by Vendora** are only permitted for user with "ADMIN" role.