CS673 Software Engineering 

Team 1 - GetActive

Software Design Document

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Arshdeep Dhillon | Security Lead  Design and Implementations  Team Lead | *AD* | 5/17/2025 |
| Jin Hao Li | Design and Implementation Lead, Team Lead | *JHL* | 5/26/2025 |
| Shaohua Yue |  | *Shaohua Yue* | 5/26/2025 |
| Jie Shi | Requirements Lead | *JS* | 5/26/2025 |
| Xiaoyao Yin | Design and Implementation | *Xiaoyao Yin* | 5/26/2025 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| Version | Author | Date | Change |
| --- | --- | --- | --- |
| 0.1 | Arshdeep Dhillon | 5/17/2025 | Filled initial draft of the document. |
| 0.2 | Jin Hao Li | 5/26/2025 | Updated Design Pattern, Status Code, Rest APIs |
| 1.0 | Arshdeep Dhillon | 5/26/2025 | Finalized document for Iteration 1. |
| 2.0 | Arshdeep Dhillon | 6/8/2025 | Update REST APIs.  Address feedback comments. |
| 2.1 | Jin Hao Li | 6/8/25 | Update REST APIs |
| 2.2 | Shaohua Yue | 6/9/25 | Update frontend structure introduction, Design patterns, Security Design and Business logic. |
| 2.3 | Xiaoyao Yin | 6/9/25 | Update activity frontend and backend files. |
| 3.1 | Jin Hao Li | 6/15 | Update Database Diagram, REST APIs |
| 3.2 | Xiaoyao Yin | 6/15 | Update activity discovery page |
| 3.3 | Shaohua Yue | 6/16 | Upload image for frontend security |

[● Introduction 3](#_heading=h.lgmuc0fy3s4z)

[● Software Architecture 3](#_heading=h.kxvygcz24w52)

[Component Decomposition 4](#_heading=h.gwvg6qhh7ih9)

[Development and Deployment Workflow 5](#_heading=h.uhxf0c7smkn)

[Technology Stack and Frameworks 5](#_heading=h.7s82xdfzysze)

[● Class Diagram 6](#_heading=h.2twv4ypjg33c)

[● UI Design (if applicable) 9](#_heading=h.d0xykk1qiafs)

[● Database Design (if applicable) 10](#_heading=h.qvkk81q7fyi5)

[● Security Design 10](#_heading=h.ep8n3je57cjz)

[● Business Logic and/or Key Algorithms 11](#_heading=h.wybk2bkiywag)

[● Design Patterns 12](#_heading=h.6imjz13jd656)

[HTTP response status codes used by REST API 12](#_heading=h.ld1u2v2oeg0j)

[● Rest APIs 13](#_heading=h.7ayt1720vo4r)

[Endpoint “/v1/register” 13](#_heading=h.lh33fbue3u3g)

[Endpoint “/v1/register/confirmation” 14](#_heading=h.6fwsanqoc3tp)

[Endpoint “/v1/register/confirmation/resend” 14](#_heading=h.qw9dy01uualp)

[Endpoint “/v1/login” 15](#_heading=h.hoer4iicysx8)

[Endpoint “/v1/reset” 16](#_heading=h.44u0tro5jp89)

[Endpoint “/v1/reset/verify” 16](#_heading=h.lm81ze88h5yn)

[Endpoint “/v1/activity” 17](#_heading=h.1qb7jp389jic)

[Endpoint “/v1/activity/{id}” 17](#_heading=h.8a9v3y58j4rx)

[Endpoint “/v1/activity/{id}” 18](#_heading=h.l65u8lgh58x7)

[Endpoint “/v1/activity/{name}” 19](#_heading=h.2nbfes465xmy)

[Endpoint “/v1/activities” 20](#_heading=h.30opej563a7x)

[Endpoint “/v1/activity/participants” 20](#_heading=h.gdr9h9ypjcbu)

[Endpoint “/v1/activity/participants” 21](#_heading=h.65w2hho9q9jb)

[Endpoint “/v1/activity/participants” 21](#_heading=h.aed8xpdeu2hj)

[Endpoint “/v1/activity/{id}/participants” 22](#_heading=h.bqkaq1up9don)

[Endpoint “/v1/activity/{id}/comment” 22](#_heading=h.o6bxgotna2vf)

[Endpoint “/v1/activity/{id}/comments” 22](#_heading=h.j1x3mtam71h1)

[Endpoint “/v1/avatar” 23](#_heading=h.4cj6cdr5ooxv)

[● Any Additional Topics you would like to include. 23](#_heading=h.on0gjpy35iem)

[● AI usage Log 23](#_heading=h.161vmjht87fy)

[● References 24](#_heading=h.ii3hoac97f36)

[● Glossary 24](#_heading=h.smujc3z5pbzn)

# Introduction

In this section, give an overview of this document, and also address the design goals of your software system.

This Software Design Document presents a comprehensive and structured overview of the *GetActive* system, a student activity engagement platform built to help university students connect through shared interests and group activities. By providing an easy-to-use interface for discovering, creating, and joining on-campus events, *GetActive* aims to reduce social isolation and encourage community building among students.

The purpose of this document is to detail the internal design of the software system, from high-level architecture to low-level implementation choices, ensuring a clear understanding for both developers and stakeholders. It begins with the **Software Architecture** section, which describes the decomposition of the system into frontend, backend, and database components, along with their interactions and dependencies. This includes a visual architecture diagram and explanations of frameworks like React, Spring Boot, and MySQL, all deployed using Docker containers. The **Class Diagram** section explores the structure of each component, presenting UML class diagrams to show the relationships between entities. The **UI Design** section explains the layout and navigation of key user-facing pages, including registration, login, activity listing, and creation screens, with a focus on usability and flow. In the **Database Design** section, the relational schema of the system is outlined, showing how data such as users, activities, and tokens are stored and managed in a MySQL database. The **Security Design** section describes the mechanisms used to ensure secure access, including JWT-based authentication, email verification, and password encryption.

# Software Architecture

In this section, you will describe the decomposition of your software system, which includes each component (which may be in terms of package or folder) and the relationship between components. You shall have at least one diagram to show the whole architecture of . The interface of each component and dependency between components should also be described. If any framework is used, it shall be defined here too.



The *GetActive* application is composed of modular components distributed across a client-server architecture. The system is decomposed into three main components: the **frontend**, the **backend**, and the **database**, each of which is containerized using Docker and deployed to the AWS platform. The architecture follows a layered design pattern within the backend and uses industry-standard frameworks and tools for development, deployment, and scalability.

## Component Decomposition

1. **Frontend (React Application)**
   * Implemented using the **React** JavaScript library.
   * Runs within a **Docker container** during development and is served as a static site after being built for production.
   * Communicates with the backend via **HTTP** calls using **RESTful APIs** (GET, POST, PUT, DELETE).
   * Provides the user interface for registration, login, browsing activities, joining/leaving groups, and creating/editing events.
2. **Backend (Spring Boot Application)**
   * Built using the **Spring Boot** framework and runs inside a **Docker container**.
   * Implements a layered architecture:
     + **REST API Controllers** handle incoming HTTP requests from the frontend and send appropriate responses.
     + **Service Layer** contains the core business logic and mediates between the controllers and data access layer.
     + **Repository Layer** uses **Spring Data JPA** to interact with the database.
   * Exposes secured RESTful endpoints using **JWT authentication** and email verification mechanisms.
3. **Database (MySQL)**
   * A **MySQL** relational database managed within its own Docker container.
   * Stores persistent data, including users, activities, verification tokens, and membership relations.
   * Communicated via the JPA Repository layer in the backend.

## Development and Deployment Workflow

Development follows a Git branching strategy where each feature, fix, or change is developed on a separate branch categorized as feat, fix, docs, style, refactor, test, or other. These branches are merged into the main branch after review. Once committed to the main branch:

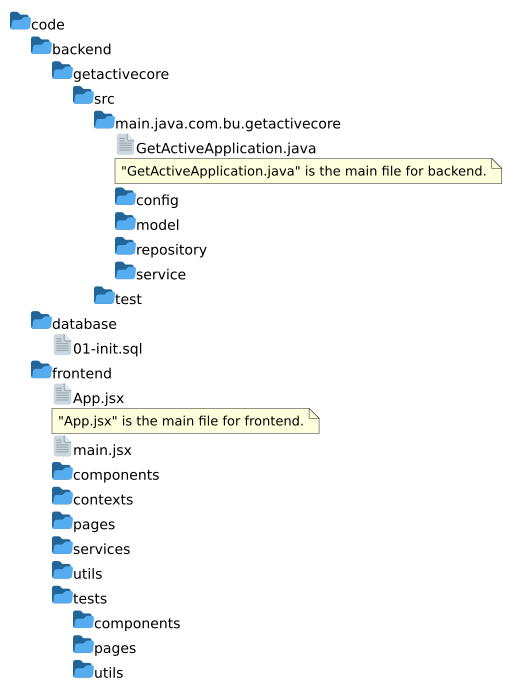
1. Frontend, Backend, and Database are packed into Docker images.
2. The docker images are then deployed to the AWS platform.
3. From the AWS platform, our application is exposed to the public where clients can access the frontend using their web browser.

## Technology Stack and Frameworks

* **Frontend**: React, HTML5, CSS3, Axios/Fetch API (for REST communication)
* **Backend**: Java, Spring Boot, Spring Security, Spring Data JPA
* **Database**: MySQL
* **Authentication**: JWT (JSON Web Tokens)
* **Deployment**: Docker, GitHub, AWS
* **Version Control**: Git, GitHub

# Class Diagram

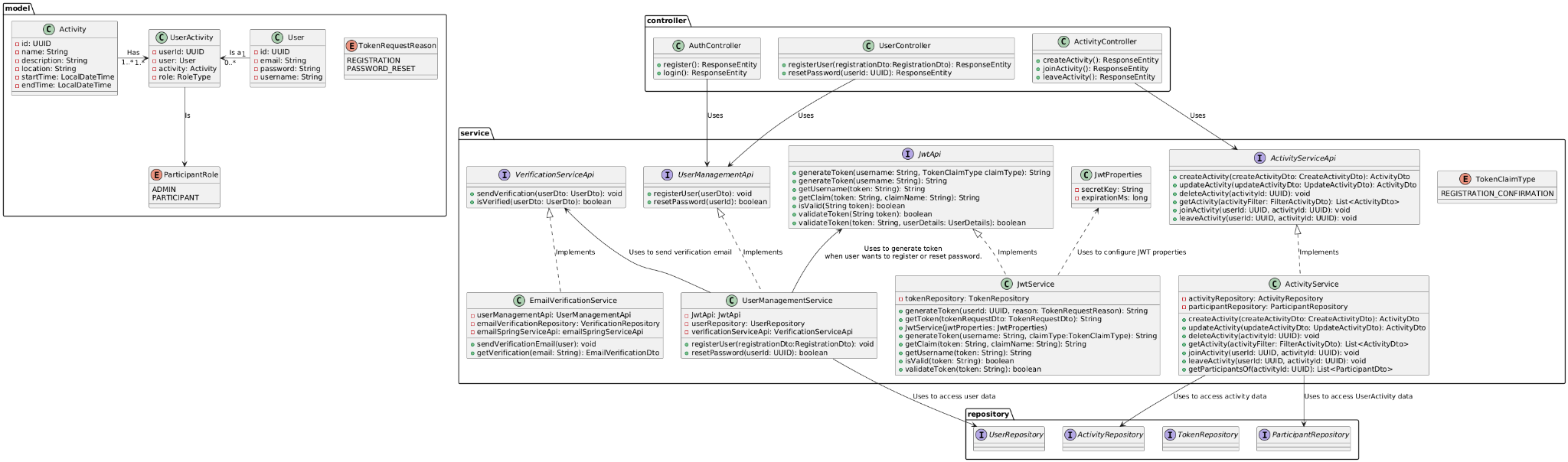
In this section, you will provide a detailed description of each component (or package) and use one or multiple class diagrams to show the main classes and their relationships in each component.



The application is organized in a layered architecture which allows us to easily expand the capabilities of each component and allows us to deploy the components using the containerization technology without having a hard dependency on each other. The primary packages of the backend are:

1. “config”
   1. This package will contain the security and JWT configuration.
   2. Example classes: SecurityConfig such as HTTP security, filters, and endpoint authorization configuration.
   3. JwtAuthenticationFilter, JwtTokenProvider: Handle token creation and validation.
2. “model”
   1. This package will contain the definition of the domain entities and DTOs.
   2. Example classes:
      1. User: Represents a registered student.
      2. Activity: Represents a campus activity.
      3. JoinRequest, VerificationToken: Support user verification and activity participation.
3. “repository”
   1. This package will contain interfaces extending JpaRepository for database access.
   2. Example interfaces:
      1. UserRepository
      2. ActivityRepository
      3. VerificationTokenRepository
      4. UserActivityRepository
4. “service”
   1. This package will contain business logic and act as the core of the backend.
   2. Example classes:
      1. UserService: Handle registration, login, and user lookup.
      2. ActivityService: Manage activity creation, update, and participation.
      3. EmailService: Send email to user’s email to verify their registration.
5. “controller“
   1. This package will contain REST controllers which will expose endpoints to the frontend.
   2. Example classes:
      1. AuthController
      2. ActivityController
      3. UserController
6. “test”
   1. This package will contain unit and integration tests for backend logic.

Following contains the class diagram of the application:



For frontend:

### components

This package contains reusable React components that can be shared across different pages of the application.

Example components:ProtectedRoute

### pages

This package contains the main page components, each representing a distinct route or view in the application.

Example pages:HomeRegistrationRegisterConfirmationCreateActivityLogin

### services

This package contains service modules responsible for handling API requests and business logic, such as authentication and activity management.

Example services:

authService

activityService

api

### utils

This package contains utility functions and helpers that provide common functionality used throughout the application.

Example utilities:jwt

### contexts

This package contains React context providers for managing global state, such as authentication status.

Example contexts:

AuthContext

### test

This package contains test files for components, pages, and utilities to ensure code quality and correctness.

Example tests:

Component tests,

Page tests,

Service tests,

Utility tests

### App.jsx

This is the main application component that sets up routing and global layout.

### Main.jsx

This is the entry point for rendering the React application into the DOM.

In modern React projects, function components are widely used instead of class components. Function components are simply JavaScript functions and do not use object-oriented concepts such as classes, inheritance, or interfaces. As a result, traditional class diagrams—which are designed to show class hierarchies and relationships—are not applicable.

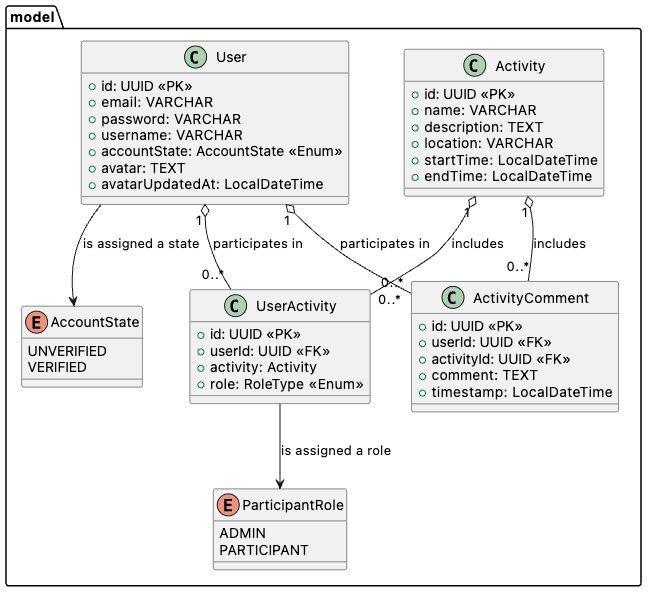
# UI Design (if applicable)

In this section, you can describe your UI design

We choose the Material UI library in our frontend project.

# Database Design (if applicable)

In this section, you shall describe any database schema if used in your software system.



# Security Design

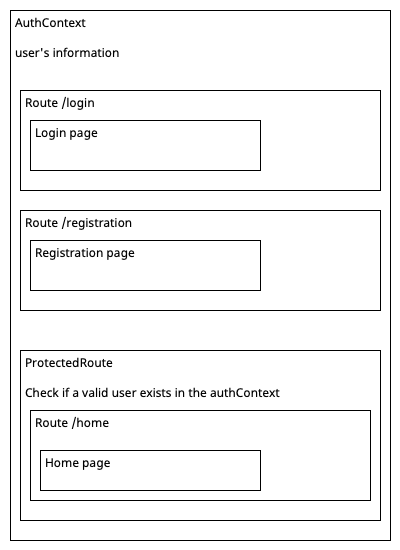
In this section, you shall describe any security design in your software system.

The system implements stateless authentication using JSON Web Tokens (JWT) to secure access to protected resources. Users can register through the */v1/register* endpoint, which is publicly accessible and does not require authentication. Upon successful registration, users can log in via the */v1/login* endpoint, which validates their credentials and returns a JWT if authentication is successful. This token must then be included in the *Authorization* header of subsequent requests to access secured endpoints such as */v1/activity*. The server validates the JWT on each request to ensure the authenticity and integrity of the token, providing secure access control without maintaining session state on the server. This design ensures that only authenticated users can access sensitive functionality while keeping the registration process open and secure.

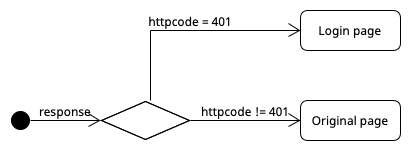
For frontend:

The system exposes Registration, Registration Confirmation and Login page for users. Using ProtectedRoute to protect the home page which is only visible for registered users.

The authContext(React context) will manage the user’s information including the token. The ProtectedRoute will access this information and decide if the protected page should be shown for users. If the user’s token is expired, the system will redirect the user to the Login page.



We use an interceptor pattern while sending or receiving http request and http response. The application will automatically add a user's token to http request. And parse the http status code from response. If the user is invalid, the system will redirect the user to the Login page.



# Business Logic and/or Key Algorithms

In this section, you shall describe any key algorithms used in your software system, either in terms of pseudocode or flowchart, or sequence diagrams.

The core business logic of the application revolves around user authentication, token management and access control for protected resources. Upon receiving a login request at /v1/login, the system verifies the provided credentials against stored user records. If valid, a JWT is generated containing the user's username and signed using a secure secret key. This token serves as a stateless mechanism for authenticating subsequent requests. For each request to secured endpoints like /v1/activity, the system extracts the token from the Authorization header, verifies its signature and expiration time, and grants access only if the token is valid. This logic ensures that only authenticated users can perform authorized operations.

For frontend:

The project includes multiple pages managed by route. The route will protect pages or components from unregistered users. The authcontext will store and provide functions for user’s register,login and logout. When a user login successfully, the authservice will parse the token to get the user's id, role and other information and store them into local storage which is provided by the browser. This user information could be used in other pages or components. Users could browse all current activities and participated activities. Join or leave any activity. As an administrator, the user could create or delete activities.

# Design Patterns

In this section, you shall describe any design patterns used in your software system.

**Repository Pattern** - This design pattern is used in Spring Boot to provide access to the data in the database while providing a layer between the business logic and database. In our system, we have the UserRespository and ActivityRepository to handle data access to the User model and Activity mode respectively.

**Builder Pattern -** This design pattern in our software system allows us to pass a variable number of arguments to create an instance of a class through Lombok.

**Singleton Pattern** - SpringBoot is using Singleton Pattern to ensure only instances of a class through annotation such as @Component, @Service, @Repository and @Controller. Only that instance is reused every time it is needed.

**Interceptor Pattern** - We use this pattern to automatically add a token to http request and parse the http status code from response.

## HTTP response status codes used by REST API

### Successful responses

#### 200 OK

* The request succeeded. The result and meaning of "success" depends on the HTTP method:
* GET: A retrieve request to the backend
* PUT: A update request to the backend
* DELETE: A delete request to the backend
* POST: A create request to the backend

#### 400 Bad Request

401 Unauthorized

403 Forbidden

404 Not Found

500 Internal Service Error

# Rest APIs

## Endpoint “/v1/register”

### Purpose

To register into the application, so the user can log in.

### HTTP Method

POST

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| email | Yes | String |
| username | Yes | String |
| password | Yes | String |

### Response

200 - Successful response will have HTTP response code 200.

### Error

400 - Bad request sent. If a user with the provided username is already registered.

400 - Bad request sent. If a user with the provided email is already registered.

## 

## Endpoint “/v1/register/confirmation”

### Purpose

In order to access secured resources, the user must confirm their registration. This REST API takes username and email which was used during registration and sends a registration confirmation email with a token. The user must enter that token in the UI to be granted access.

### HTTP Method

POST

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| username | Yes | String |
| password | Yes | String |

### Response

200 - Confirmation email is successfully sent.

200 - If the username or email does not exist.

200 - If the user has already confirmed their registration.

### Error

400 - Missing required fields.

500 - Unable to send the confirmation email.

## Endpoint “/v1/register/confirmation/resend”

### Purpose

Allows the user to resend the registration confirmation email.

### HTTP Method

POST

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| username | Yes | String |
| password | Yes | String |

### Response

200 - Confirmation email is successfully sent.

200 - If the username or email does not exist.

200 - If the user has already confirmed their registration.

### Error

400 - Missing required fields.

500 - Unable to send the confirmation email.

## Endpoint “/v1/login”

### Purpose

To log into the application and access authenticated resources.

### HTTP Method

POST

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| email | Yes | String |
| password | Yes | String |

### Response

200 - Successful response will have HTTP response code 200. Also, User’s metadata in JSON format will be returned.

### Error

400 - Bad request sent.

404 - Email is not registered or the client didn’t approve the registration request to their email.

429 - When the user has attempted too many times, they must retry later.

## Endpoint “/v1/reset”

### Purpose

To reset a password when a user wants to change their password..

### HTTP Method

POST

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| email | Yes | String |

### Response

200 - Response will have HTTP response code 200 whether or not the given email exists. If an email matches, a reset verification email will be sent to that email.

### Error

400 - Bad request sent.

429 - When the user has attempted too many times, they must retry later.

## Endpoint “/v1/reset/verify”

### Purpose

User has clicked on the reset verification email. Verify the reset token.

### HTTP Method

POST

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| token | Yes | String |
| password | Yes | String |

### Response

200 - Response will have HTTP response code 200 if the rest was successful.

### Error

400 - Bad request sent.

404 - Token is no longer valid or is not found.

## Endpoint “/v1/activity”

### Purpose

To create an activity.

### HTTP Method

POST

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| name | Yes | String |
| location | Yes | String |
| description | No | String |
| startDateTime | Yes | DateTime |
| endDateTime | Yes | DateTime |

### Response

201 - Response will have HTTP response code 201 if activity was created successfully.

### Error

400 -Bad request sent.

* If the start date time or end date time of activity is in the past
* empty activity name, start datetime, end datetime or location is provided

403 - If a non-admin user made this request.

404 - If invalid user id is provided.

## Endpoint “/v1/activity/{id}”

### Purpose

To update an activity.

### HTTP Method

PUT

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| name | Yes | String |
| location | Yes | String |
| description | No | String |
| startDateTime | Yes | DateTime |
| endDateTime | Yes | DateTime |

### Response

200 - Response will have HTTP response code 200 if activity was updated successfully.

### Error

400 -Bad request sent.

* If the start date time or end date time of activity is in the past
* empty activity name, start datetime, end datetime or location is provided

403 - If a non-admin user made this request.

404 - If invalid activity id is provided.

## Endpoint “/v1/activity/{id}”

### Purpose

To delete an activity.

### HTTP Method

DELETE

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| force | No | Boolean (Default=false) |

### Response

200 - When activity is deleted successfully, response will have HTTP response code 200. If force is set, then all participants of this activity will be removed.

### Error

400 - Bad request sent.

403 - If a non-admin user made this request or there were participants in the activity and “force” was set to false.

404 - If invalid user id or activity id is provided.

## Endpoint “/v1/activity/{name}”

### Purpose

To get a list of similar activities by name

### HTTP Method

GET

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| page | No | Integer (Default=0) |
| size | No | Integer(Default=10) |
| sortBy | No | String(Default=id) |
| ascending | No | Boolean(Default=true) |

### Response

200 - When a list of activities are returned successfully. The list can be empty.

## Endpoint “/v1/activities”

### Purpose

To get a list of activities for activity discovery purposes

### HTTP Method

GET

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| page | No | Integer (Default=0) |
| size | No | Integer(Default=10) |
| sortBy | No | String(Default=id) |
| ascending | No | Boolean(Default=true) |

### Response

200 - When a list of activities are returned successfully. The list can be empty.

## Endpoint “/v1/activity/participants”

### Purpose

Join an activity

### HTTP Method

POST

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| activityId | Yes | String |

### Response

200 - When a list of activities are returned successfully. The list can be empty.

400 - already joined an activity

## Endpoint “/v1/activity/participants”

### Purpose

leave an activity

### HTTP Method

DELETE

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| activityId | Yes | String |

### Response

200 - When a list of activities are returned successfully. The list can be empty.

## Endpoint “/v1/activity/participants”

### Purpose

Get a list of activities by user id(token)

### HTTP Method

GET

### Parameters

N/A

### Response

200 - return a list of activities for a given user

## Endpoint “/v1/activity/{id}/participants”

### Purpose

Get the roster of participants for a specific activity

### HTTP Method

GET

### Parameters

N/A

### Response

200 - When a list of roster of participants is returned successfully. The list can be empty.

## Endpoint “/v1/activity/{id}/comment”

### Purpose

Create an activity comment

### HTTP Method

POST

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| comment | Yes | String |

## Endpoint “/v1/activity/{id}/comments”

### Purpose

Get a list of comments by activity id

### HTTP Method

GET

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| page | No | Integer (Default=0) |
| size | No | Integer(Default=10) |
| sortBy | No | String(Default=id) |
| ascending | No | Boolean(Default=true) |

## Endpoint “/v1/avatar”

### Purpose

Update an avatar

### HTTP Method

PUT

### Parameters

| Parameter | Required (Yes/No) | Type |
| --- | --- | --- |
| avatar | Yes | String |

### Response

200 - Response will have HTTP response code 200 if the avatar was successful.

### Error

400 - Avatar data cannot be blank, invalid image format. .

### Response

200 - When a list of comments is returned successfully. The list can be empty.

# Any Additional Topics you would like to include.

# AI usage Log

You are allowed and even encouraged to use AI tools to help you generate the project idea, plan it and build it, but you need to clearly describe 1) What tools were used? 2) for what specific tasks and 3) Is it helpful? 4) how did you evaluate or modify AI-generated content? Additionally, you should submit the exported AI chat history as an appendix or share that with the instructor and facilitators.

| Tools | Who | Tasks | helpful | Evaluation/modification | links |
| --- | --- | --- | --- | --- | --- |
| ChatGPT | Arshdeep | Securing the REST APIs | Yes | We did use “@PreAuthorize” and JWT. | https://chatgpt.com/share/68350ab1-e458-8013-8a0f-704727c178c0 |
| ChatGPT | Shaohua Yue | Learning CI/CD workflow setup and Vitest framework | Yes | I learned basic grammar of Vitest framework and core concepts of CI/CD. Apply these theories into the project. |  |

# 

# References

* + <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status#client_error_responses>

# Glossary