Implementation Manual

LOWES FOR GEEKS

The Events Hosting Platform

Lowes For Geeks: The Events Hosting Platform

Implementation Manual

TABLE OF CONTENTS

Serial Number	Section Name	Page Number
1.	Technology Stack	2
2.	Modularization	3
3.	Functionalities of Request Headers	8
4.	Database Setup	12
5.	Implementation Summary	15

Technology stack

"Lowes for geeks", is an event hosting platform which organizes various events which are primarily classified as Organization, Team and Private Events. This platform is also used to manage the various teams and members in the organization. The platform is implemented as a CRUD (Create, Read, Update and Delete) application which meets the standards of RESTful (Representational state transfer) API. All the requests and responses are represented in JSON (JavaScript Object Notation) format. All the request headers contain the identifier of the member performing the operation except the request that corresponds to API end point which starts the application. In order to exercise the CRUD operations of the application, "postman" tool is used.

This back-end application is written in Java language using Spring boot, which is an open source Java-based framework used to create a micro Service. Spring boot is developed by Pivotal Team and is used to build stand-alone and production ready spring applications. The MongoDB database is used to store the data corresponding to the application. MongoDB is a cross-platform document-oriented database program, classified as a NoSQL database program, MongoDB uses JSON-like documents with schema.

Technologies used: *Java 8+, Spring boot, MongoDB*

Modularization

1. Entities (package: “com.lfg.entities”)

a) Event

Event entity is used to store the information regarding an event to be hosted. This entity is stored in a collection in database. While creating an instance of this entity, the user is supposed to give the information about Identifier of the event (which must be unique), Event name, Event Description, Event Location, Event Type (Whether the event is organizational(0) , team(1) or private(2) event) and whether the event is recurring.

b) Member

Member entity is used to store the details of a member who is part of organization. This entity is stored in a collection in database. While creating an instance of this entity, the user is supposed to give information about Identifier of member (which must be unique), member first name, member last name, mail Id of the member and whether the member should be made as organization admin.

c) Member Activity

Member Activity entity stores the activity of a member in the organization, such as whether member is part of a team, if yes, the respective team Id, whether the member is an administrator of the respective team, the events watched by the member, liked by the member, participated by the member are also stored in the form of corresponding event identifiers. This entity need not be created by the user, it is updated automatically as the user performs an activity. These member activities are stored in database.

d) Team

Team entity stores the information regarding a team in the organization. It is stored in a collection in database. While creating an instance of team, user must input the team identifier and team name. A member can be part of at most one team. So, the members who are not a part of any existing team in the organization at the time of creation are allowed to create a team and be the team member. The creator of the team is automatically added as a team administrator to the list that stores corresponding team members.

e) Organization

The organization entity is the core entity of the application. It stores the list of teams, list of members and corresponding member activities and list of organization admins. This entity is neither stored in database nor given as input by user but it contains all the functions which are to be performed for the management of teams, events and members. During the application startup, all

the teams, events, members and member activities are loaded to this entity from the database.

2. Repositories (package: “com.lfg.repositories”)

A repository is a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects.

a) Event Repository

Event Repository is used to encapsulate the “events” collection in the database. It extends Mongo Repository.

b) Member Repository

Member Repository is used to encapsulate the “members” collection in the database. It extends Mongo Repository.

c) Member Activity Repository

Member Activity Repository is used to encapsulate the “member-activity” collection in the database. It extends Mongo Repository.

d) Team Repository

Team repository is used to encapsulate the “teams” collection in the database. It extends Mongo Repository.

3. Service Interfaces (package: “com.lfg.serviceinterfaces”)

a) Event Service Interface

Event Service Interface declares various database operations performed on “events” collection in the database. It is an abstract type that specify the behavior that Event Service class must implement.

b) Member Service Interface

Member Service Interface declares various database operations performed on “members” collection in the database. It is an abstract type that specify the behavior that Member Service class must implement.

c) Member Activity Service Interface

Member Activity Service Interface declares various database operations performed on “member-activity” collection in the database. It is an abstract type that specify the behavior that Member Activity Service class must implement.

d) Organization Interface

Organization Interface does not declare any database related operations but it declares several computations to be made in the form of functions in the organization class, the user can realize the organization interface as a reference to the functions that are implemented for the management of entire "Lowes for geeks" platform. It is an abstract type that specify the behavior that Organization class must implement.

e) Team Service Interface

Team Service Interface declares various database operations performed on "teams" collection in the database. It is an abstract type that specify the behavior that Team Service class must implement.

4. Services (package: "com.lfg.services")

a) Event Service

Event Service implements the various database related functions specified in the event service interface which are basically insertion, deletion, modification and retrieval of entities in "events" collection in database. During the implementation all the functions uses event repository handler to access the database.

b) Member Service

Member Service implements the various database related functions specified in the member service interface which are basically insertion, deletion, modification and retrieval of entities in "members" collection in database. During the implementation all the functions uses member repository handler to access the database.

c) Member Activity Service

Member Activity Service implements the various database related functions specified in the member activity service interface which are basically insertion, deletion, modification and retrieval of entities in "member-activity" collection in database. During the implementation all the functions uses member activity repository handler to access the database.

d) Team Service

Team Service implements the various database related functions specified in the team service interface which are basically insertion, deletion, modification and retrieval of entities in "teams" collection in database. During the implementation all the functions uses team repository handler to access the database.

5. Controllers (package: "com.lfg.controllers")

A rest controller handles various http requests sent by the user by implementing the corresponding GET, POST, PUT and DELETE methods.

a) Base Controller

Base Controller handles the GET request sent by the user during the application initialization. It implements the corresponding handler function as follows:

- Load the data from the database to corresponding lists in the organization entity.
- If the database is empty, which is generally the case when the user runs the application instance for the first time, then this function creates an Organization administrator named "Sriharsha Namilakonda" with '1' as member identifier.
- It checks the non-recurring events in the database if the respective event has already ended it is marked as expired as per the requirements.
- It prints the welcome message to the user for the application. Every time the user runs the application, this is the first management operation that is going to be executed.

b) Member Controller

Member Controller is a rest controller which implements all the CRUD operations for the member API as the response to the requests sent by the user.

c) Event Controller

Event Controller is a rest controller which implements all the CRUD operations for the event API as the response to the requests sent by the user.

d) Team Controller

Team Controller is a rest controller which implements all the CRUD operations for the team API as the response to the requests sent by the user.

6. Comparators (package: "com.lfg.comparators")

a) Popular Events Comparator

Popular Events Comparator is used to sort the list of popular events in the organization based on number of likes and then number of views inclusively by implementing the Comparator Interface.

b) Trending Events Comparator

Trending Events Comparator is used to sort the list of trending events in the organization based on number of likes, number of watchers and then number of participants inclusively by implementing the Comparator Interface.

c) Upcoming Events Comparator

Upcoming Events Comparator is used to sort the list of upcoming events in the organization based on number of views inclusively by implementing the Comparator Interface.

Functionalities of Request Headers

Let us assume Initial_Path => *http://<host>:<port>*

1. Application Initiation API

   a) GET Method(s):

   ✓ Initial_Path/lowesforgeeks

   This should be the first request sent from the user as this is the "api" endpoint for the platform. It initiates the application by fetching the data from database and modifying the status of events based on the present date.

2. Member API

   a) GET Method(s):

   ✓ Initial_Path/lowesforgeeks/member/{memberId}/all

   This request retrieves details of all the members in the organization

   ✓ Initial_Path/lowesforgeeks/member/{memberId}/{targetmemberId}

   This request is used to retrieve the details of target member with given identifier

   b) POST Method(s):

✓ Initial_Path/lowesforgeeks/member/{memberId}/create

This request is used to create a member in the organization. The details of the member must be present in the request body

c) PUT Method(s):

✓ Initial_Path/lowesforgeeks/member/{memberId}/update

This request is used to update the details of a member in the organization. The updated details of the member must be present in the request body

d) DELETE Method(s):

✓ Initial_Path/lowesforgeeks/member/{memberId}/delete/{targetmemberId}

This request is used to remove a member from the organization, it also erases all the activities associated with that member in the organization

3. Team API

a) GET Method(s):

- ✓ Initial_Path/lowesforgeeks/team/{memberId}/all
  This request displays the details of all the teams in the organization

- ✓ Initial_Path/lowesforgeeks/team/{memberId}/{teamId}
  This request displays the details of the team with the given identifier

b) POST Method(s):

- ✓ Initial_Path/lowesforgeeks/team/{memberId}/create
  This request is used to create a team in the organization. The details of the team must be present in the request body

c) PUT Method(s):

- ✓ Initial_Path/lowesforgeeks/team/{memberId}/changename/{teamId}/{newname}
  This request is used to change the name of the team with the specified identifier

- ✓ Initial_Path/lowesforgeeks/team/{memberId}/addmember/{teamId}/{targetmemberId}
  This request is used to add the target member to the specified team

- ✓ Initial_Path/lowesforgeeks/team/{memberId}/deletemember/{teamId}/{targetmemberId}
  This request is used to delete the target member from the specified team

- ✓ Initial_Path/lowesforgeeks/team/{memberId}/makeadmin/{teamId}/{targetmemberId}
  This request is used to make the target member (who is already a member of the specified team) the team admin

- ✓ Initial_Path/lowesforgeeks/team/{memberId}/removeadmin/{teamId}/{targetmemberId}
  This request is used to remove the target member as the administrator of the specified team. However, the target member continues to be part of specified team

d) DELETE Method(s):

- ✓ Initial_Path/lowesforgeeks/team/{memberId}/delete/{teamId}
  This request is used to delete the team from the organization

4. Event API

a) GET Method(s):

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/view/{eventId}
  This request is used to view the event with specified identifier. It increments the event views by 1.

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/trendingevents
  This request is used to view all the trending events which are created earlier.

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/popularevents
  This request is used to view all the popular events which are created earlier.

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/upcomingevents
  This request is used to view all the upcoming events which are created earlier.

b) POST Method(s):

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/create
  This request is used to create (host) an event. The details of the event must be present in the request body

c) PUT Method(s):

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/update
  This request is used to update the details of the created event. The updated event details must be present in the request body

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/like/{eventId}
  This request is used by the member to like the specified event

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/unlike/{eventId}
  This request is used by the member to dislike the specified event

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/watch/{eventId}
  This request is used by the member to watch the specified event

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/unwatch/{eventId}
  This request is used by the member to stop watching the specified event

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/participate/{eventId}
  This request is used by the member to participate in the specified event

- ✓ Initial_Path/lowesforgeeks/event/{memberId}/unparticipate/{eventId}
  This request is used by the member to withdraw from the specified event

d) DELETE Method(s):

- ❖ Initial_Path/lowesforgeeks/event/{memberId}/cancel/{eventId}
  This request is used to cancel (or delete) the specified event. All the activities associated with the event also get erased from the corresponding member activity history

Database Setup

Database name: “Lowesforgeeks”

1. Members (collection name: “members”)

Snippet of JSON Input:

```
{
    "memberId" : 1,
    "firstname" : "Sriharsha",
    "lastname" : "Namilakonda",
    "mailId" : "sriharsha.namilakonda@gmail.com",
    "organizationAdmin" : true
}
```

Snippet of Document in Database:

```
_id: 1
firstname: "Sriharsha"
lastname: "Namilakonda"
mailId: "sriharsha.namilakonda@gmail.com"
organizationAdmin: true
_class: "com.lfg.entities.Member"
```

2. Events (collection name: “events”)

Snippet of JSON Input:

```
{
    "eventId":100,
    "eventType":"Organization",
    "name":"Campus Drive",
    "description":"Our company visit various institutions for campus placement",
    "location":"Banglore",
    "recurring":true,
    "startDate":"2020-07-11T12:00:00",
    "endDate":"2020-07-27T12:00:00"
}
```

Snippet of Document in Database:

```
_id: 100
eventType: "Organization"
name: "Campus Drive"
description: "Our company visit various institutions for campus placement"
location: "Banglore"
recurring: true
startDate: "2020-07-11T12:00:00"
endDate: "2020-07-27T12:00:00"
› createdBy: Object
numberOfLikes: 10
numberOfWatchers: 5
numberOfViews: 1
numberOfParticipants: 4
recurringFrequency: "Month"
TeamIdForTeamEvent: 0
createdDate: "2020-05-10T10:57:58.803045900"
expired: false
_class: "com.lfg.entities.Event"
```


3. Teams (collection name: “teams”)

Snippet of JSON Input:

```
{
    "teamId" : 12,
    "teamName" : "Managers"
}
```

Snippet of Document in Database:

```
_id: 12
teamName: "Managers"
teamMembers: Array
  0: Object
      _id: 1
      firstname: "Sriharsha"
      lastname: "Namilakonda"
      mailId: "sriharsha.namilakonda@gmail.com"
      organizationAdmin: true
  1: Object
      _id: 2
      firstname: "Anish"
      lastname: "Raj"
      mailId: "rajanish@gmail.com"
      organizationAdmin: true
  2: Object
      _id: 3
      firstname: "Tanvi"
      lastname: "Tripura"
      mailId: "tanvitripura@gmail.com"
      organizationAdmin: false
  3: Object
  4: Object
_class: "com.lfg.entities.Team"
```


4. Member Activities (collection name: “member-activity”)

Snippet of Document in Database:


```
_id: 1
partOfATeam: true
teamId: 12
participatingEvents: Array
    0: 100
    1: 101
    2: 102
    3: 201
LikedEvents: Array
    0: 100
    1: 101
    2: 102
watchedEvents: Array
    0: 100
    1: 101
    2: 102
teamAdmin: true
_class: "com.lfg.entities.MemberActivity"
```

Implementation Summary

The implementation of Lowes For Geeks, event management platform, took off as much redundancy as possible from the user point of view, as the user needs to input only the primary details of an instance of entity, the secondary details that are associated with the input will be calculated and updated automatically by the application. The application implementation contains fine grained modularity which makes the extension of functionalities much easier in the future and if at all the authentication details or entity details are to be changed, the applications needs just a spoc (single point of change) and the application runs satisfying the requirement criteria without any hindrance. The user who runs the application must be aware of the fact that application session must be initiated from the specified api end point. This implementation accesses the database only for insertion, deletion, modification of data and retrieval of entire database during application initiation, which reduces the number of database accesses as much as possible, thereby reducing the operation execution time. All the function names and field names used in the implementation are descriptive which makes it easier to understand for the developer who inspect the code. The use of interfaces in order to specify the functions that are implemented in the entire application eases the process of understanding the application even further. In order to exercise the CRUD functionalities in the application, tools like “postman” can be used.

The user in order to run the application, just needs to create a database with the name “Lowesforgeeks” and collections with the names “events”, “member-activity”, ”members”, ”teams”. The implementation provides a smooth application user experience and application understandability by displaying the messages in the console as a response to authentication failure, or in the scenarios where the request does not meet the required standards of the application. This implementation uses only the default ports and host name as “localhost”, if any of these details needs to be changed the user can do so in “application.properties” file.
