A Report
on
Event Management System

(By Afan Ahmad Khan, Sujith Narayan, Jyothi Prasad and Rakesh Buchan)

Introduction

Purpose : A catalog of events , with the ability to detect the events happening in a particular area.

Objectives :

The objectives of the system can be categorized according to the people using the system, which can be grouped into Organizers, Users and Admin.

- Organizers can create new events, update the event once it's added or delete the event.
- Users can retrieve the events, register for the event, find details about the event and de-register for an event.
- Admin - will be able to modify any event, delete any event, delete the users or the organizers.

Title

Event Management System

Authors

Afan Ahmad Khan
Sujith Narayan
Jyothi Prasad
Rakesh Buchan

Abstract

This report lists out the design and implementation details of our project ‘Event Management System’. This project is built using Java as a programming language and uses Spring Framework in order to implement the MVC design pattern. The database used in the project is MySQL.

Introduction

Nowadays, traveling has become much more about experiencing local cultures than just visiting tourist places. But, finding cool local hangouts and events as an undercover tourist or for someone who is newly moved can be difficult, especially when you know no one in the area. Looking to bridge that gap, we propose an online application which helps in knowing “What’s Happening Around!”

Event Management System (EMS) is an online application which provides users with the catalog of events.

Any user registered with EMS can browse for all the events from EMS database, find all details about the event and register for an event. EMS provides a feature for the user to list all his attending events, which helps in keeping track of events, so that any updates to the event occur in future can be easily viewed.

A registered user can advertise his hosted event which can be of type Sport, Music or Technology. He also has an option to update/cancel his event at a later point. User can easily find all his organized events by navigating to “Events I am organizing” section which helps in making quick updates to his events

EMS also facilitates Admin features to few users who can view all users registered with EMS and also an option to remove a user and an event.

EMS acts as an event repository which contains all the past and future events with their details, attendees and users who were interested.

The EMS implementation can be further enhanced by adding features like option to browse for events based on area code and user interests, collecting the user's feedback for his attended event and widen the types of events being advertised.

Requirements

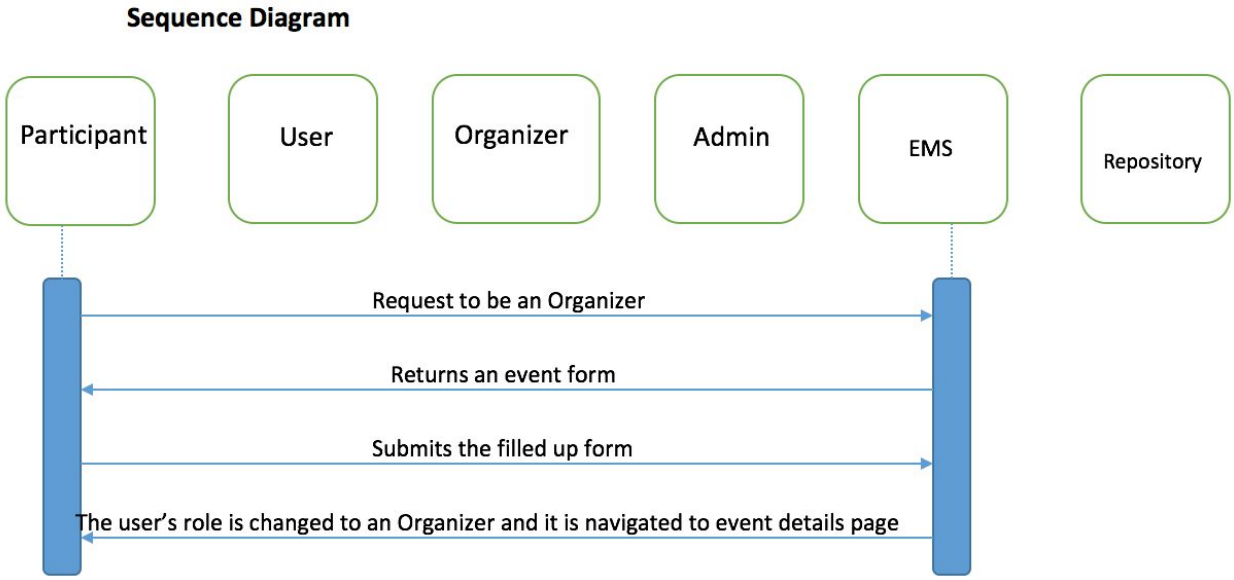
The requirements for the project being produced are broadly classified into the following categories:

1. All the users registered in the system are classified as Participants. Each participant has the ability to view all the events listed out in the portal. A participant can also register for an event or show interest in an event through the application.
2. A participant can also organize an event of it's own. A participant will also have the ability to modify or remove an event created by it.
3. A participant organizing an event attains the role of an Organizer in EMS.
4. A participant attending an event attains the role of an Attendee in EMS.
5. A participant attending an event can also bring a fixed number of guests to the event.
6. An event will have a fixed number of seats, as determined by the organizer.
7. An event can be an indoor event or an outdoor event, and is classified into three categories – Sport event, Music event and Technology event.
8. A participant can also be an Admin in EMS. An admin will have the ability to modify or remove any event. An admin can also remove any participant from EMS.

Based on the above requirements, the following use cases have been proposed for the system:

1. Organizer Registration [eventOrganizer]

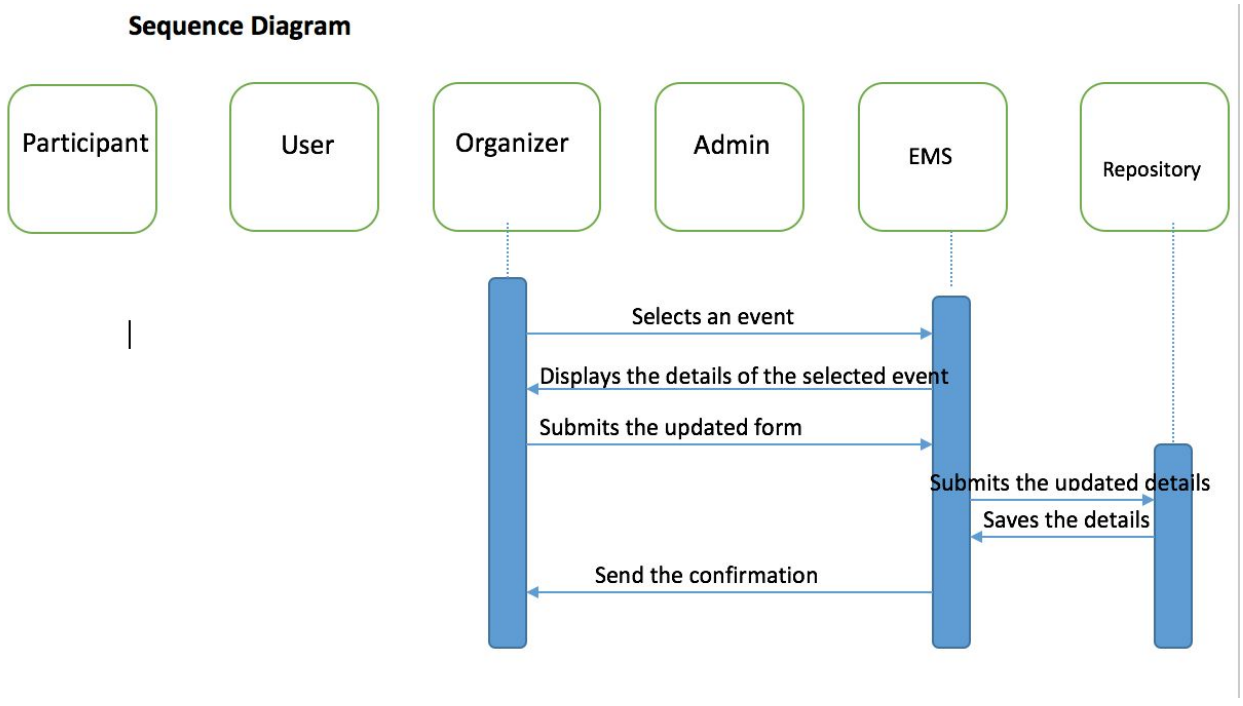
Exposition: A user wants to become an Organizer for the EMS. **Precondition**: The user has registered into the EMS. **Post condition**: The user's role is changed to an Organizer.
Step-by-step Description: 1. Includes: [#queryEvent] 2. [#Participant] - The registered user places the request to be an Organizer to the EMS. 3. [#EMS] - The system returns an event form to be filled up by the registered user. 4. [#Participant] - The registered user submits the filled up form to the EMS. 5. [#EMS] - The system changes the registered user's role to an Organizer and creates an event. 6. [#EMS] Exception - Registered User's role change failure 7. [#Participant] - The Registered User is navigated to the event details page.



2. Modify Event [modifyEvent]

Exposition: The Organizer updates/modifies an event's details **Precondition**: The event to be modified must be an event that is created by that Organizer itself, and not by by anyone else. **Post condition**: The event details are updated in the EMS system
Step-by-step description: 1. Includes: [#queryEvent] 2. [#Organizer] - The organizer selects an event 3. [#EMS] - The system displays the details of the selected event 4. [#Organizer] - The Organizer makes the changes and submits it to the system to be saved. 5. [#EMS] - The system submits the update to the repository 6. [#Repository] - Saves the event's new details and sends confirmation to the system 7. [#Repository] Exception - Event save failure 8. [#EMS] - The system sends the confirmation to the Organizer

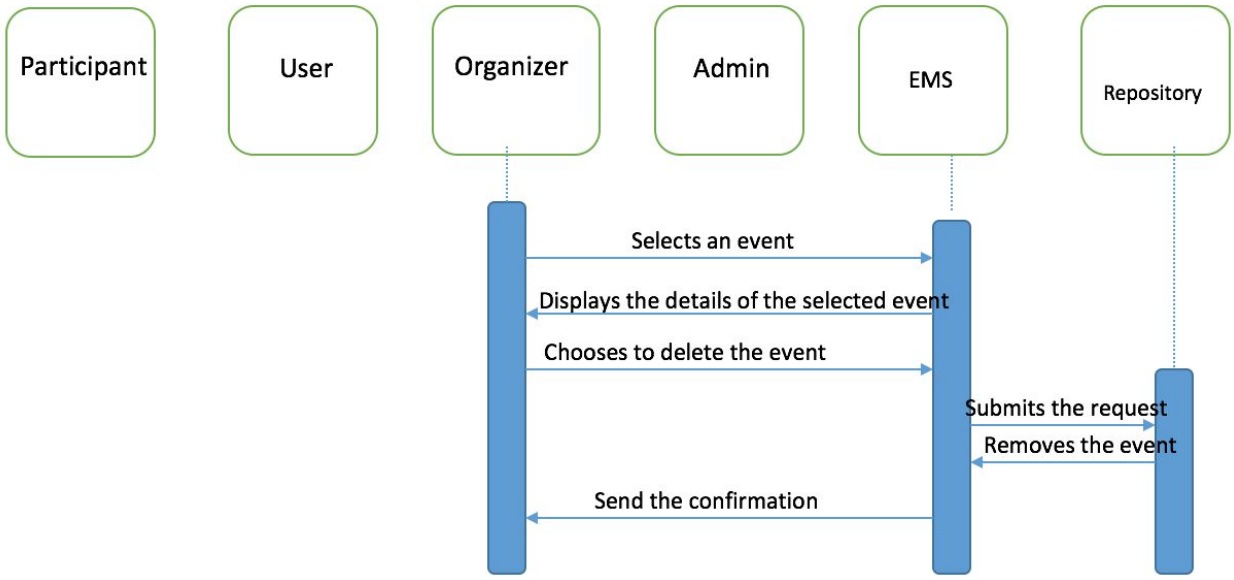
Sequence Diagram



3. Remove Event [removeEvent]

Exposition: The Organizer removes an event from the repository **Precondition**: The event to be removed must be an event created by the Organizer itself **Post condition**: The event is removed from the repository of the EMS system
Step-by-step description: 1. Includes: [#queryEvent] 2. [#Organizer] - The organizer selects an event 3. [#EMS] - The system displays the details of the selected event 4. [#Organizer] - the organizer chooses to remove the selected event 5. [#EMS] - the system submits the remove request to the repository 6. [#Repository] - removes the event and sends confirmation to the system 7. [#Repository] Exception - Event remove failure 8. [#EMS] - The system sends the confirmation to the organizer

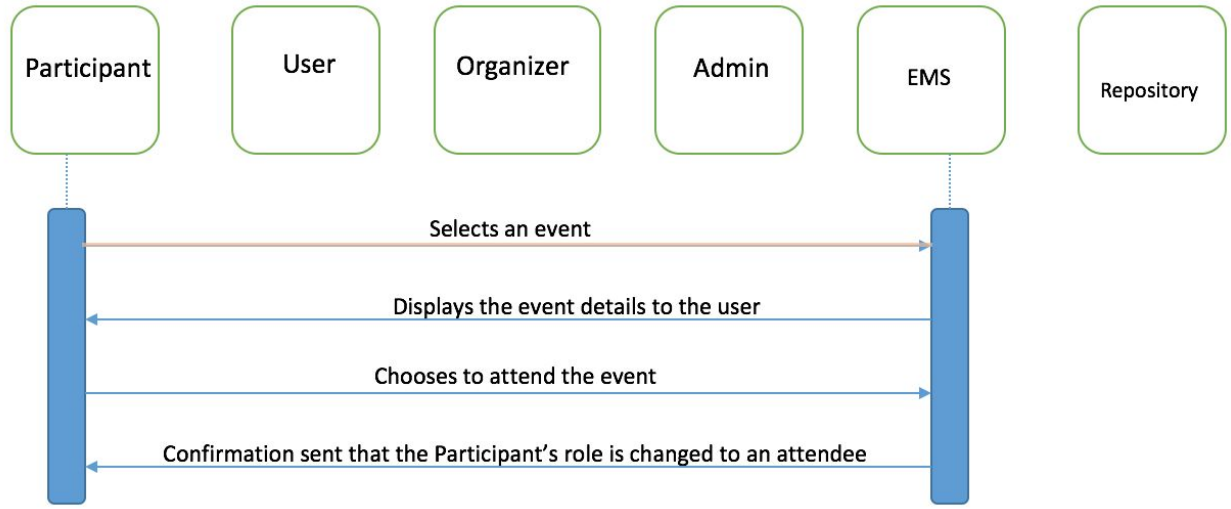
Sequence Diagram



4. Attend Event [attendEvent]

Exposition: A user chooses to attend an event. **Precondition**: The event is at a future date. **Post condition**: The user attends the event.
Step-by-step description: 1. Includes: [#userRegistration] 2. Includes: [#queryEvent] 3. [#Participant] - The registered user selects an event. 4. [#EMS] - The system displays the event details to the user. 5. [#Participant] - The registered user chooses to attend the event. 6. [#EMS] - The system changes the registered user's role to Attendee and notifies that the user is attending the event.

Sequence Diagram



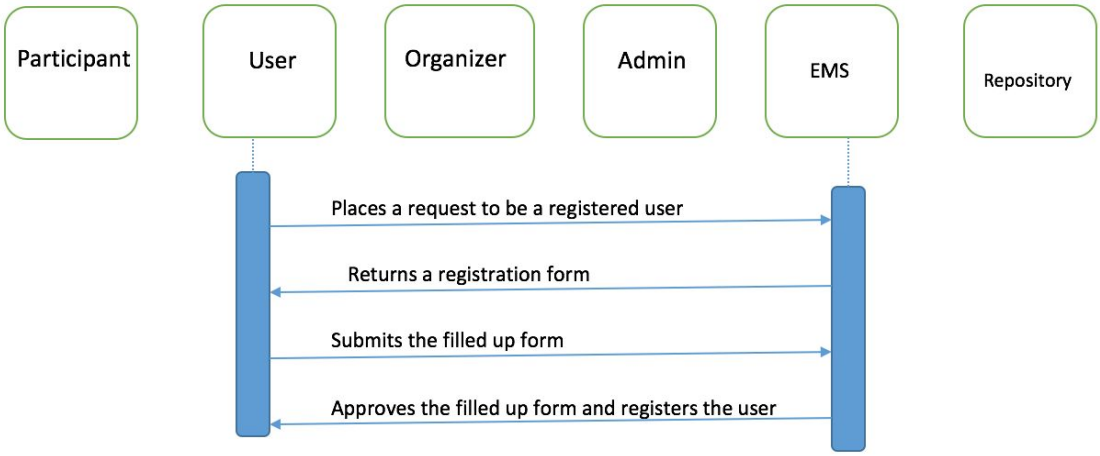
5. Rate Event [rateEvent]

Exposition: A user rates an event. **Precondition**: The user has attended the event. **Post condition**: The user's rating is used to calculate the average rating.
Step-by-step description: 1. [#Participant] - The registered user selects an attended event. 2. [#EMS] - The system displays the event details to the user. 3. [#Participant] - The registered user chooses to assign a rating to the event. 4. [#EMS] - The system assigns a rating to the event and notifies the user of the assignment.

6. User Registration [registerUser]

Exposition: A user wants to become a registered user. **Precondition**: The user is not already a registered user. **Post condition**: The user's details are stored in the database.
Step-by-step description: 1. [#User] - the user places a request to be a registered user. 2. [#EMS] - the system returns a form to be filled up by the user. 3. [#User] - the user fills the form and submits it. 4. [#EMS] - the system approves the request and notifies the user.

Sequence Diagram

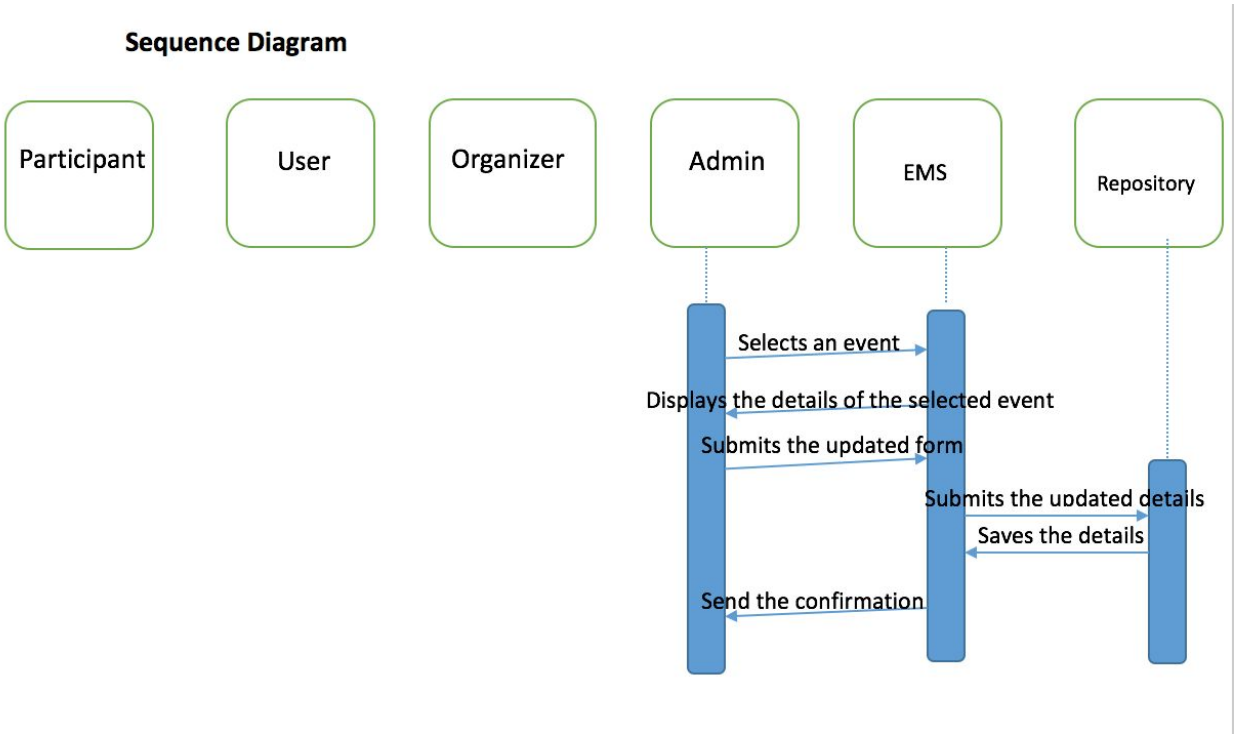


7. Query Event [queryEvent]

Exposition: A user queries the system to find an event. **Precondition**: The user knows some or all of the event values. **Post condition:** The user is returned with the event/events that match the search criteria.
Step-by-step description: 1. [#User] - The user submits a combination of query event values. These can include title, event date and place name. 2. [#EMS] - The system returns a list of events that match the user query.

8. Modify Event [modifyEvent] – by the Admin

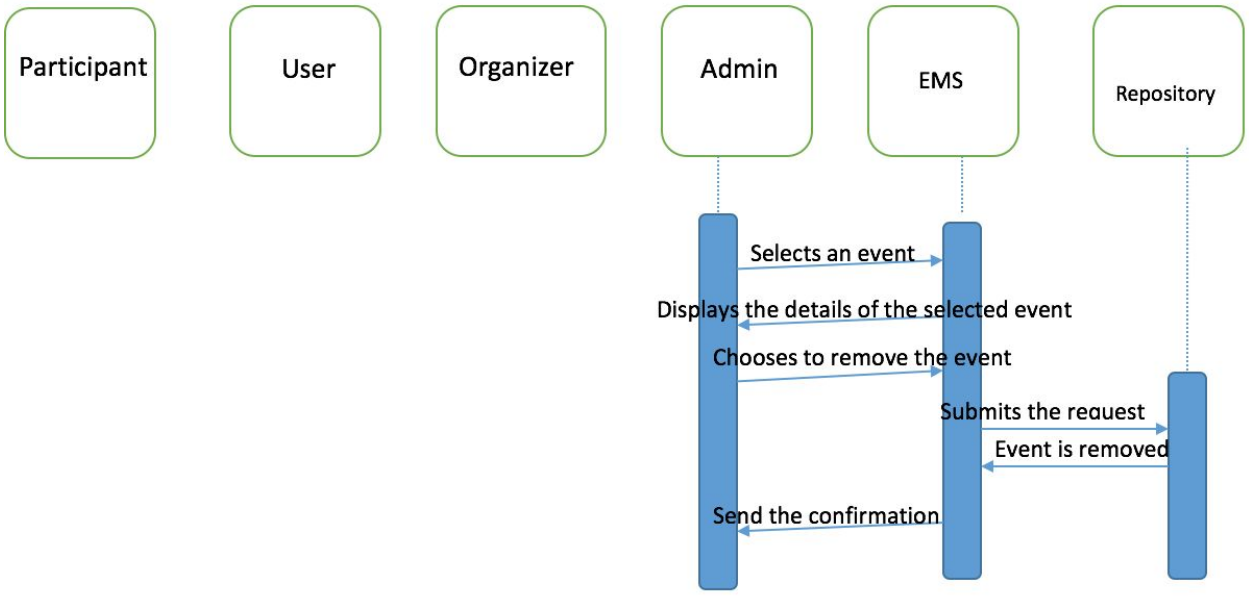
Exposition: The Administrator updates/modifies an event's details **Precondition**: The administrator has proper authentication to make the changes **Post condition**: The event details are updated in the EMS system
Step-by-step description: 1. Include: #queryEvent 2. [#Administrator] - the administrator selects an event 3. [#EMS] - the system displays the details of the selected event 4. [#EMS] - the system submits the update to the repository 5. [#Repository] - saves the event's new details and sends confirmation to the system 6. [#Repository] Exception - Event save failure 7. [#EMS] - the system sends the confirmation to the administrator



9. Remove Event [removeEvent]

Exposition: The Administrator removes an event from the repository **Precondition**: The administrator has proper authentication to remove an event **Post condition**: The event is removed from the repository of the EMS system
Step-by-step description: 1. Include: #queryEvent 2. [#Administrator] - the administrator selects an event 3. [#EMS] - the system displays the details of the selected event 4. [#Administrator] - the administrator chooses to remove the selected event 5. [#EMS] - the system submits the remove request to the repository 6. [#Repository] - removes the event and sends confirmation to the system 7. [#Repository] Exception - Event remove failure 8. [#EMS] - the system sends the confirmation to the administrator

Sequence Diagram



10. Query Participant [queryParticipant]

Exposition: A user queries the system to find a participant. **Precondition**: The user knows some or all of the participant details. **Post condition**: The user is returned with the participant(s) that match the search criteria.
Step-by-step description: 1. [#Administrator] - The administrator submits a combination of query event values. These can include name, location and age. 2. [#EMS] - The system returns a list of participants that match the search query.

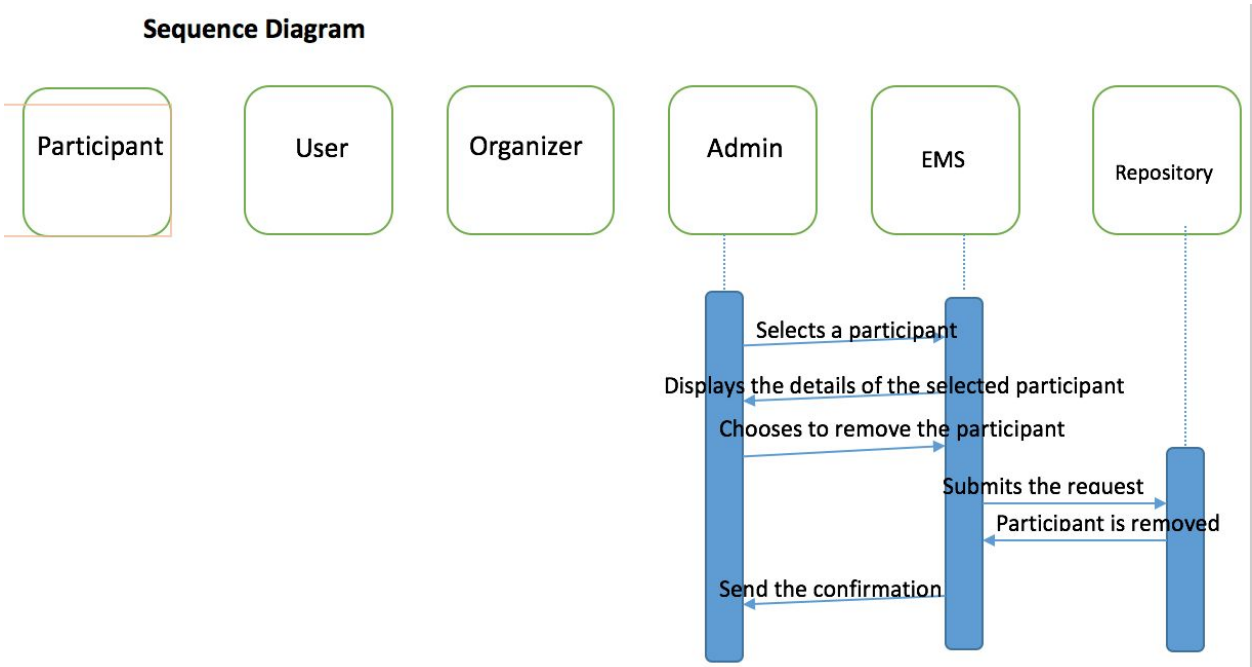
11. Modify Participant [modifyParticipant]

Exposition: The Administrator updates/modifies a participant's details **Precondition**: The administrator has proper authentication to make the changes **Post condition**: The participant details are updated in the EMS system
Step-by-step description: 1. Include: [#queryParticipant] 2. [#Administrator] - the administrator selects a participant 3. [#EMS] - the system displays the details of the selected participant 4. [#Administrator] - the administrator makes the changes and submits it to the system to be saved 5. [#EMS] - the system submits the update to the repository 6. [#Repository] - saves the participant's new details and sends confirmation to the system 7. [#Repository] Exception - Participant save failure 8. [#EMS] - the system sends the confirmation to the administrator

12. Remove Participant [removeParticipant]

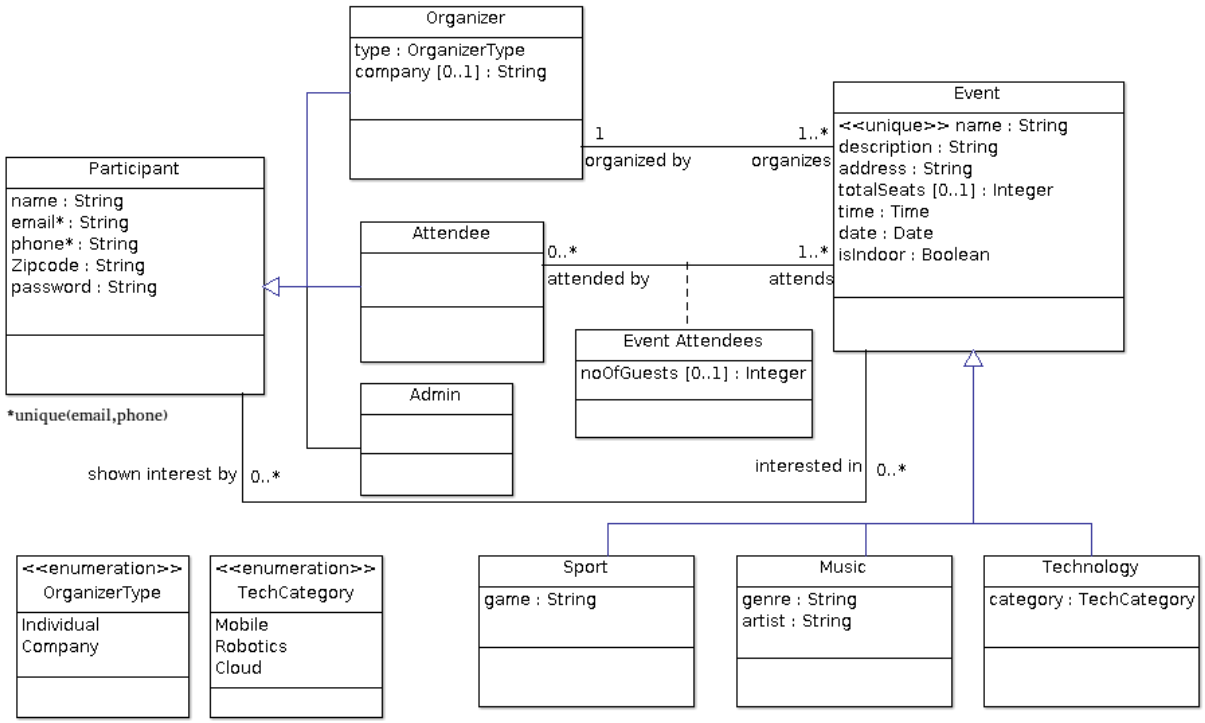
Exposition: The Administrator removes a participant from the repository **Precondition**: The administrator has proper authentication to remove a participant **Post condition**: The participant is removed from the repository of the EMS system
Step-by-step description: 1. Include: [#queryParticipant] 2. [#Administrator] - the administrator selects a participant 3. [#EMS] - the system displays the details of the selected participant 4. [#Administrator] - the administrator chooses to remove the selected participant 5. [#EMS] - the system submits the remove request to the repository 6. [#Repository] - removes the participant and sends confirmation to the system 7. [#Repository] Exception - Participant remove failure 8. [#EMS] - the system sends the confirmation to the administrator

Sequence Diagram



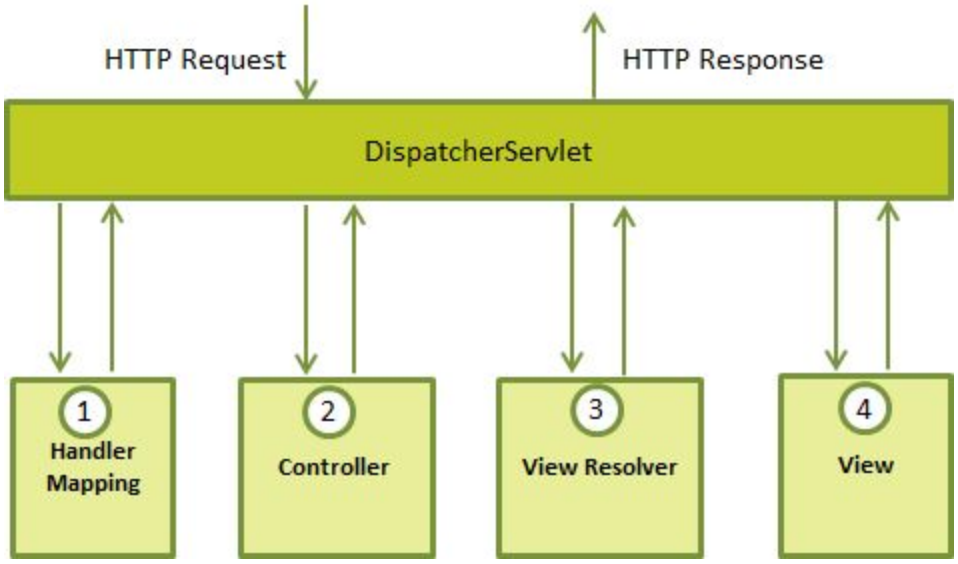
Design

UML Class Diagram



Implementation

Language used: Java
Framework: Spring MVC
Web Technology: JSP, Javascript and Bootstrap CSS



- Model

Model is the interface that defines a holder for model attributes. The model's attributes are stored in a java.util.Map structure and can be referenced elegantly from the application's server pages. In EMS, the controller methods add attributes to the model and the server pages dereference these attributes.

Example: The event details are retrieved as a list of Event objects and this list is stored in the model.
model.addAttribute("events",events);
This list of events is then referenced from the server pages on the UI by ${events}.

- View

The view is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret. In EMS, the view consists of a set of java server pages that interact with the model and controllers. A directory called WEB-INF exists and this contains the JSP files that are not available to the root of the application. A file called web.xml stores the servlet mappings.

The view also contains logic to restrict certain user operations. For instance, when the user has already selected to attend the event, the attend button is greyed out to ensure the user does not click on attend again which would result in a DuplicateKeyException.

- Controller

In the spring framework, the controller is responsible for processing user requests and building appropriate model and passes it to the view for rendering. In EMS, controllers exist for home, event, participant and login requests.

The annotation @RequestMapping is used to map requests from the UI to corresponding methods on the back-end.

The HomeController maps requests for displaying the home page.
The EventController maps requests for performing CRUD operations on the events.
The ParticipantController maps requests for performing CRUD operations on the participants.
The LoginController maps requests for performing authentication and session initialization.

- Service

Service layer interfaces between the controllers and the data access layers. The classes in the service layer are marked with the annotation @Service. The controllers declare service references and these references are used to call methods of the service. The service classes instantiate the DAO objects and call data access methods of the DAO.

- Data Access Objects (DAO)

The data access objects handle connection to the data source and the classes in the DAO layer contain methods to perform CRUD operations on the database. The DAO classes are annotated with the @Component(<dao_name>) annotation.

- Session

The HttpSession interface is used to identify the user that is currently logged in. The interface provides a method called setAttribute which can be used to set the username and ID of the currently logged in user. The details are set when the user registers or logs in. The ID attribute is retrieved using the getAttribute method whenever the user ID is needed to perform a CRUD operation on the database. The session attributes are cleared when the user logs out.

Discussion

Event Management System provides a user a platform to view all the events happening in a particular area. It presents the users with a catalog of Sport, Music and Technology events. Some of the notable features of this application are:

· Event Management System ensures that any user to the application is registered as a participant in the system. In this way, a user can attend an event or multiple events without providing it's details again and again.

· A participant is also given an option to organize an event, and list it as one of the events in the portal. In this way, a participant can be an Organizer in the system.

· The EMS ensures that all the necessary event details are filled up the Organizer before creating an event. This ensures that any 'Prospective Attendee' to an event is given a detailed information about the event before it chooses to attend the event.

· The EMS also gives the Organizers to modify or remove an event created by it. This ensures that the Organizer can create an event without worrying about the prospects of it being cancelled in the future.

· The EMS also gives an option to some of the participants to be an Admin of the system. An Admin of EMS will have total control on EMS. It can remove any participant, in addition to updating or removing any event in the system.

· The whole application is built following the MVC Design pattern – The most popular design choice while developing web applications.

· The software gives an option for one to set up the entire database automatically before deploying the application on the local server. This ensures that anyone who wants to setup the application does not face any schema level hassles.

· The application is built keeping the Spring Framework design pattern in mind. This means that all the view pages, the java classes interacting with the database, and the model classes for the databases are properly grouped into the JSP files, the DAO files and the POJO files respectively.

· The EMS system also ensures that the user does not have to make changes to the JDBC properties at multiple places. One just has to modify the context.xml file for the local server in order to set the application up and running.

Conclusion

EMS can be made more relevant to the user by adding browse feature based on locality and his interests. Also providing event suggestions to the user based on his relevance feedback would make EMS more beneficial

References

1. Spring tutorial videos by ‘Javabrains’.
   https://www.youtube.com/channel/UCYt1sfh5464XaDBH0oH_o7Q
2. Wikipedia
   https://www.wikipedia.org/
3. Tutorials point
   https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm
4. Spring docs
   http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jdbc/core/namedparam/NamedParameterJdbcTemplate.html
5. Spring JDBC query examples
   https://www.mkyong.com/spring/spring-jdbctemplate-querying-examples/
6. Guide to UML diagrams
   http://creately.com/blog/diagrams/uml-diagram-types-examples/