PES University, Bengaluru

(Established under Karnataka Act 16 of 2013)

Department of Computer Science & Engineering

Session: Jan - May 2022

UE19CS353 – Object Oriented Analysis and Design with Java

Theory ISA (Mini Project)

Report on

Event Management Application

By:

Lakshmi Narayan P – PES2UG19CS200

Namith Telkar – PES2UG19CS246

Netra Shaligram – PES2UG19CS253

6th Semester

Section D

1. Project Description

Events are always entertaining and fun but they are equally challenging to organise. The job just doesn't end with organising but a lot goes into gathering people and spreading the word about the event. Our project aims at easing the problem and providing a solution where an event can be easily hosted and plays a major role in the spread of the event which is a major factor for any event to be successful.

The application has the following features:

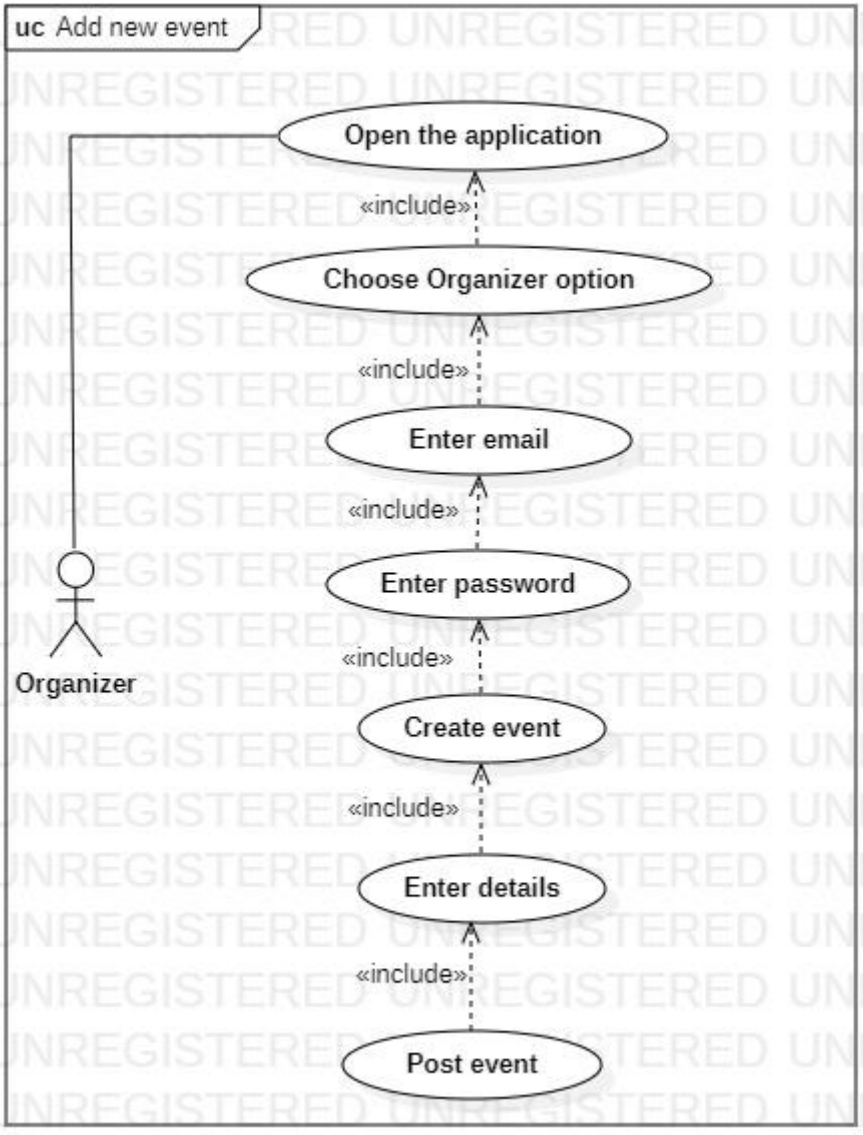
- To sign up/sign in as a User and view all the events and look into their details and register for them via the link provided by the event organiser.
- To sign in as an Admin to create an event by filling out details required and uploading an image to represent the event banner.
- The user's and admins can also filter the events based on certain criteria thus allowing for easier search of events.

Link to Github repository - https://github.com/Namith-Telkar/Aight-Events

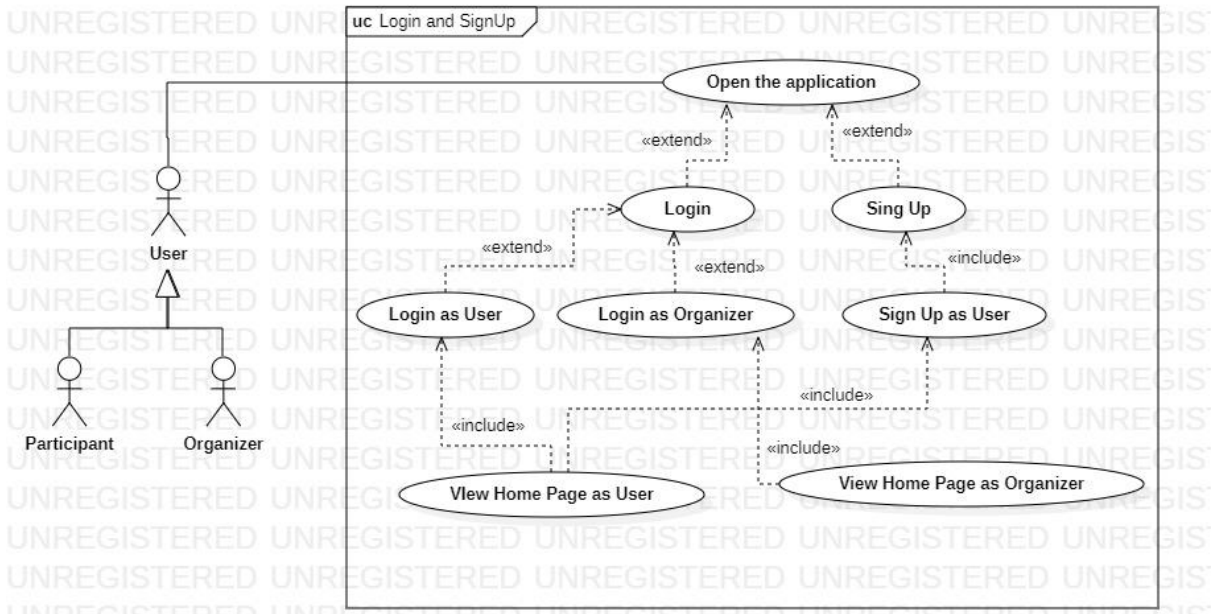
2. Analysis and Design Models

Use Case Diagrams :

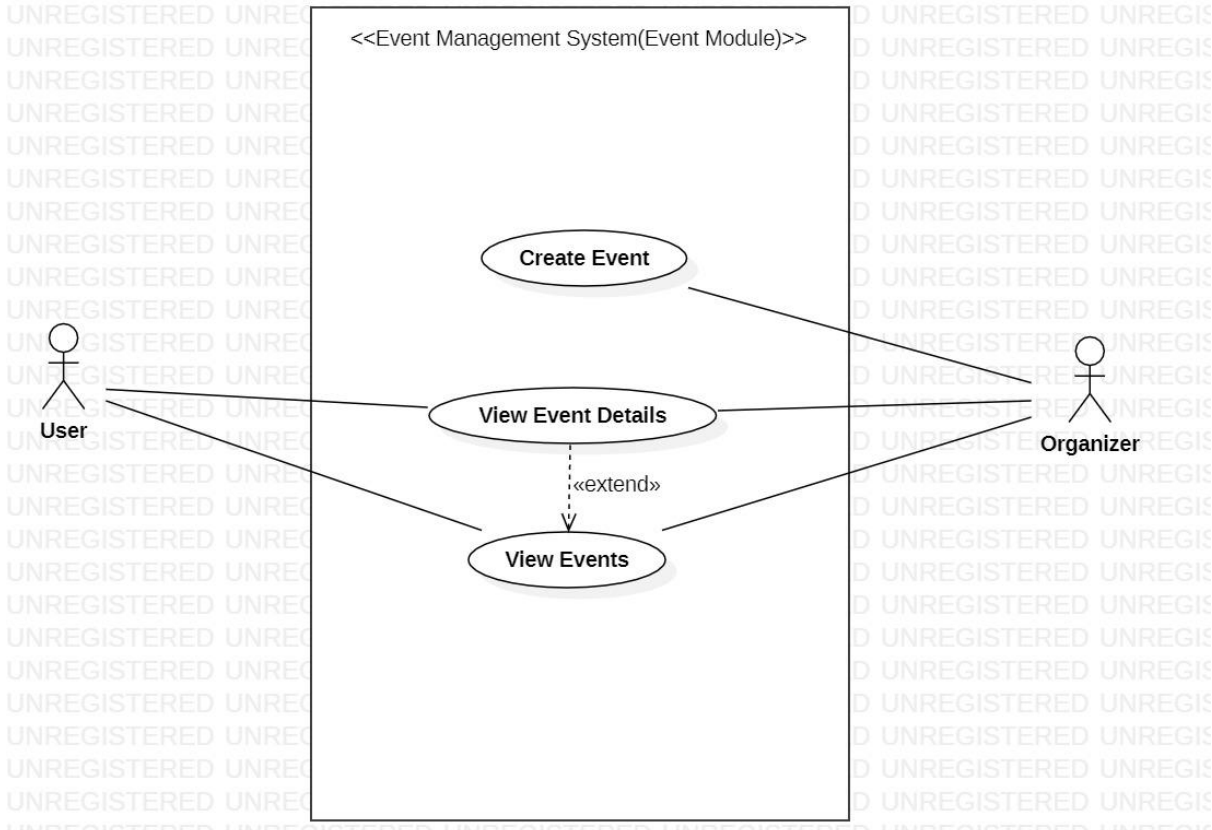
Add new event (Organiser)



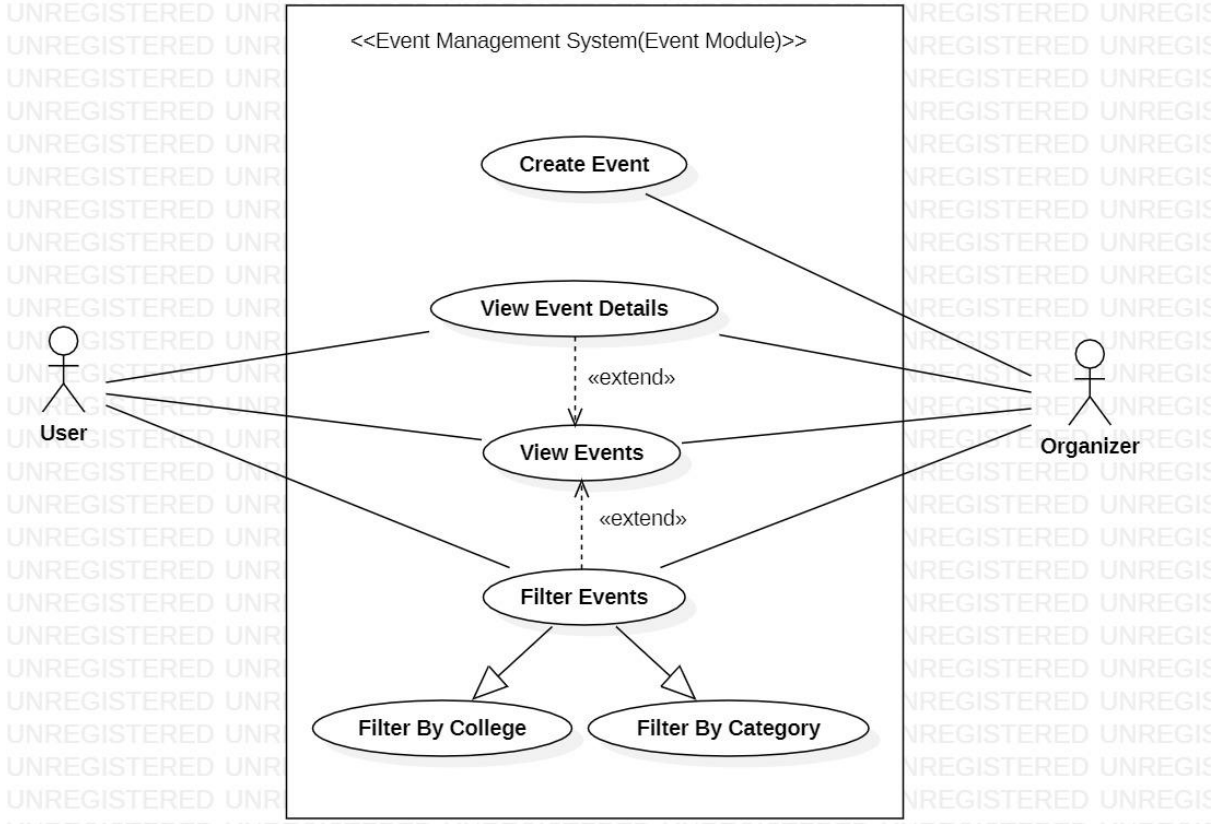
Login and SignUp (Organiser)



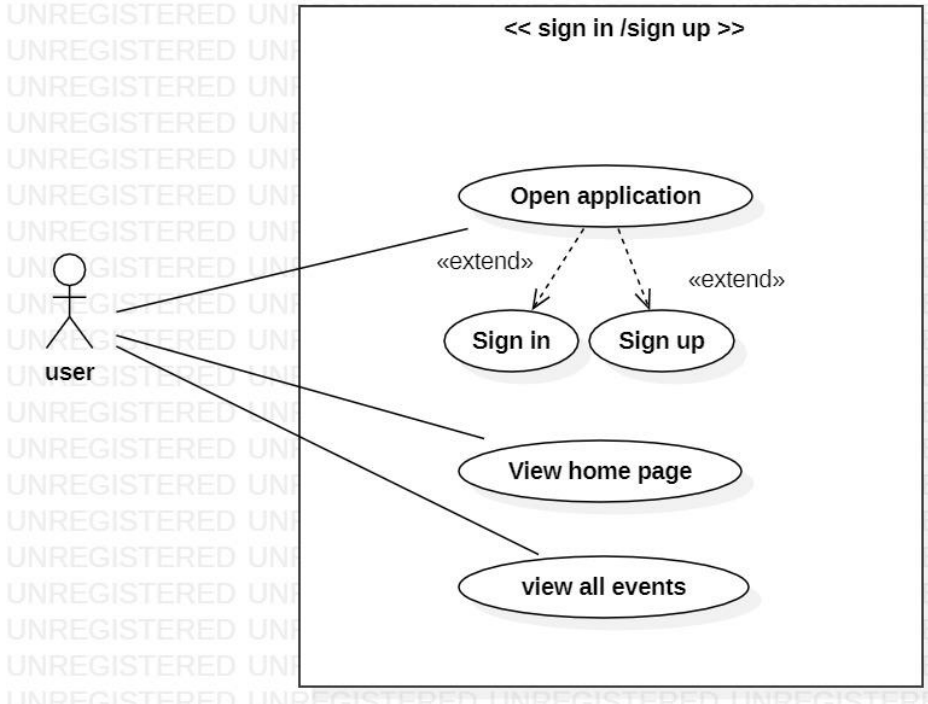
View Events (User & Organiser)



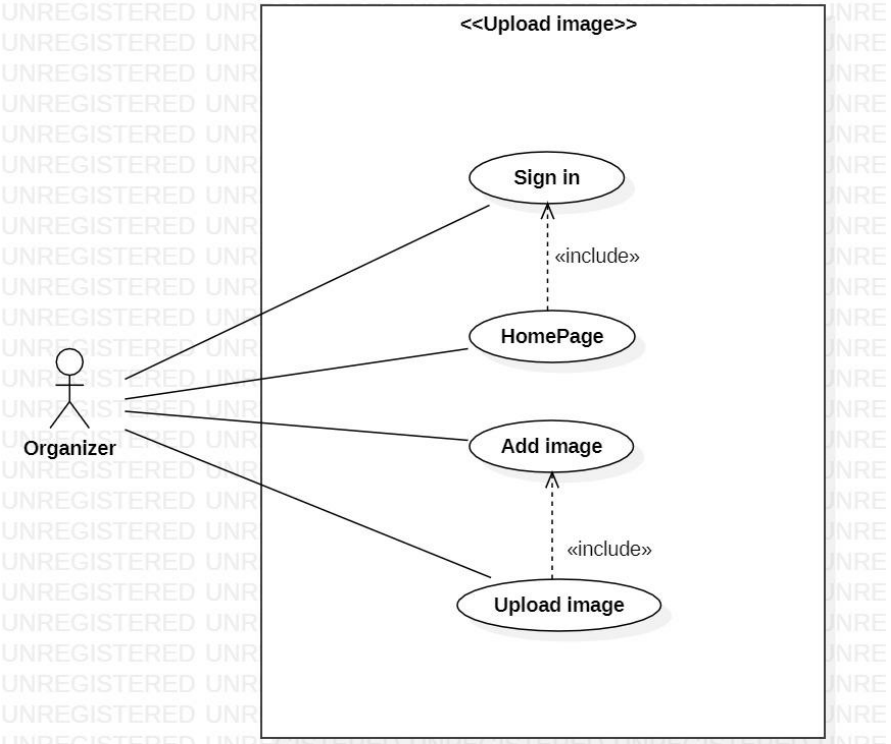
Filter Events(User & Organiser)



Sign up /Sign in and view events (user)

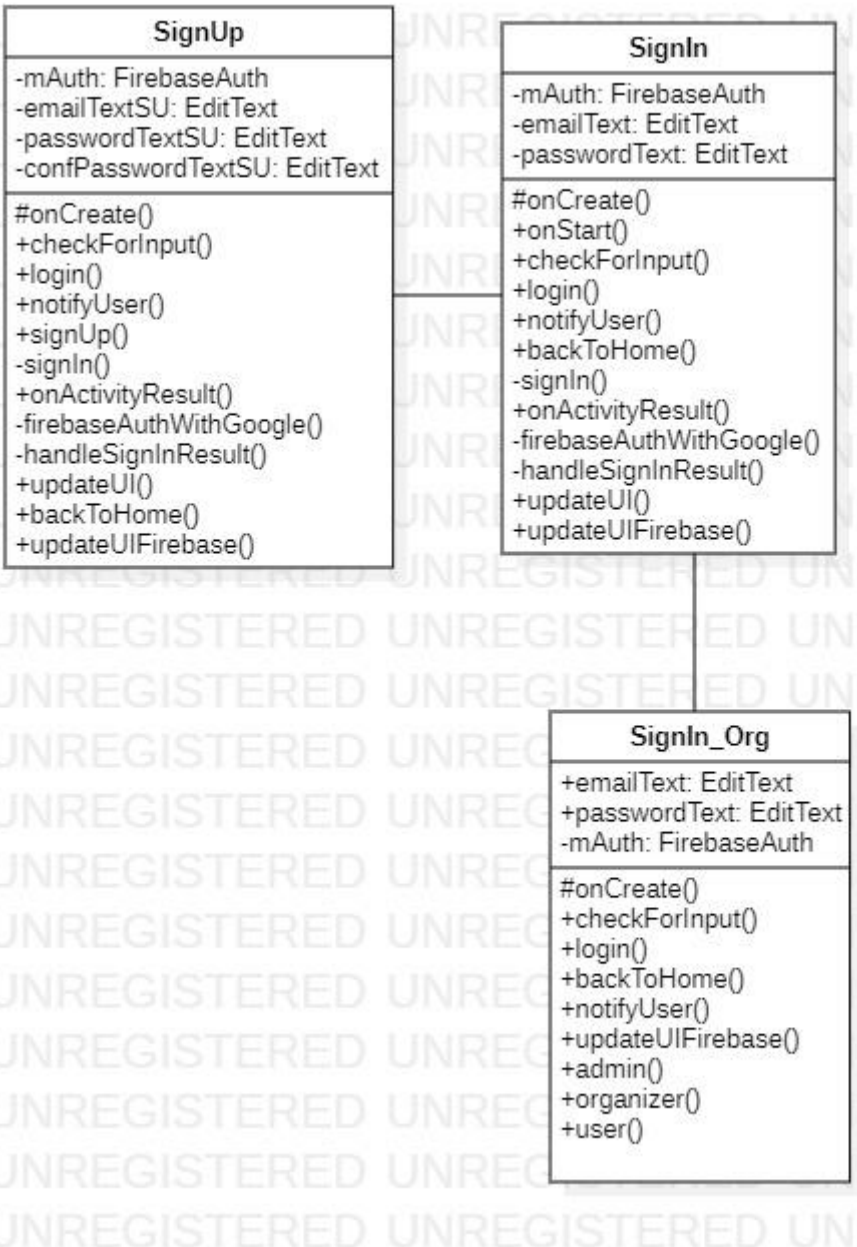


Adding and uploading image

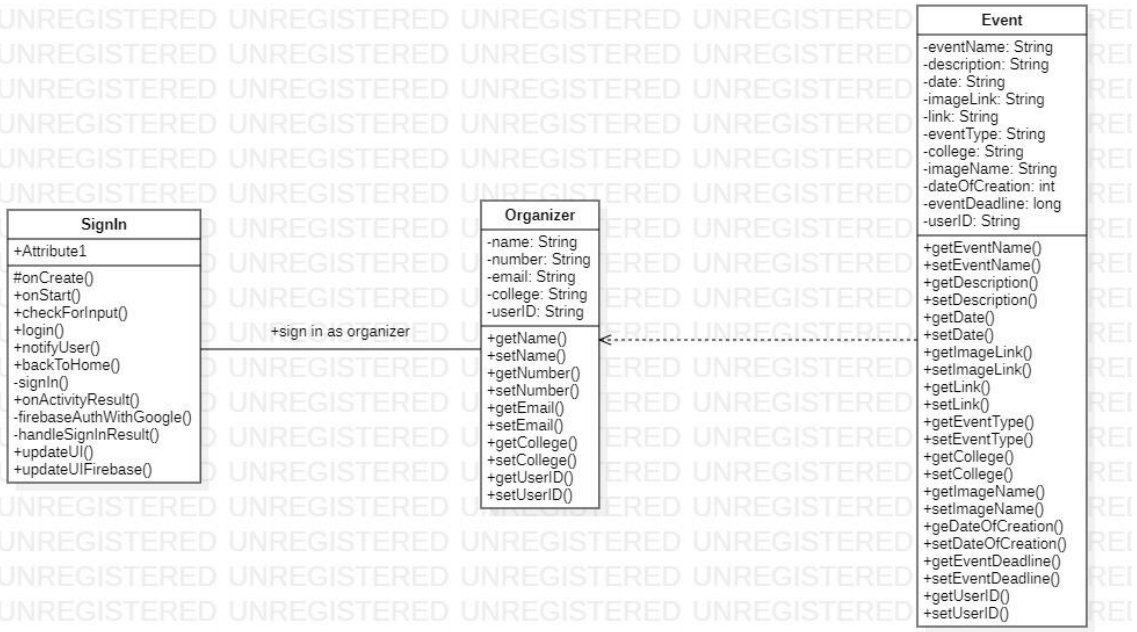


Class Diagrams

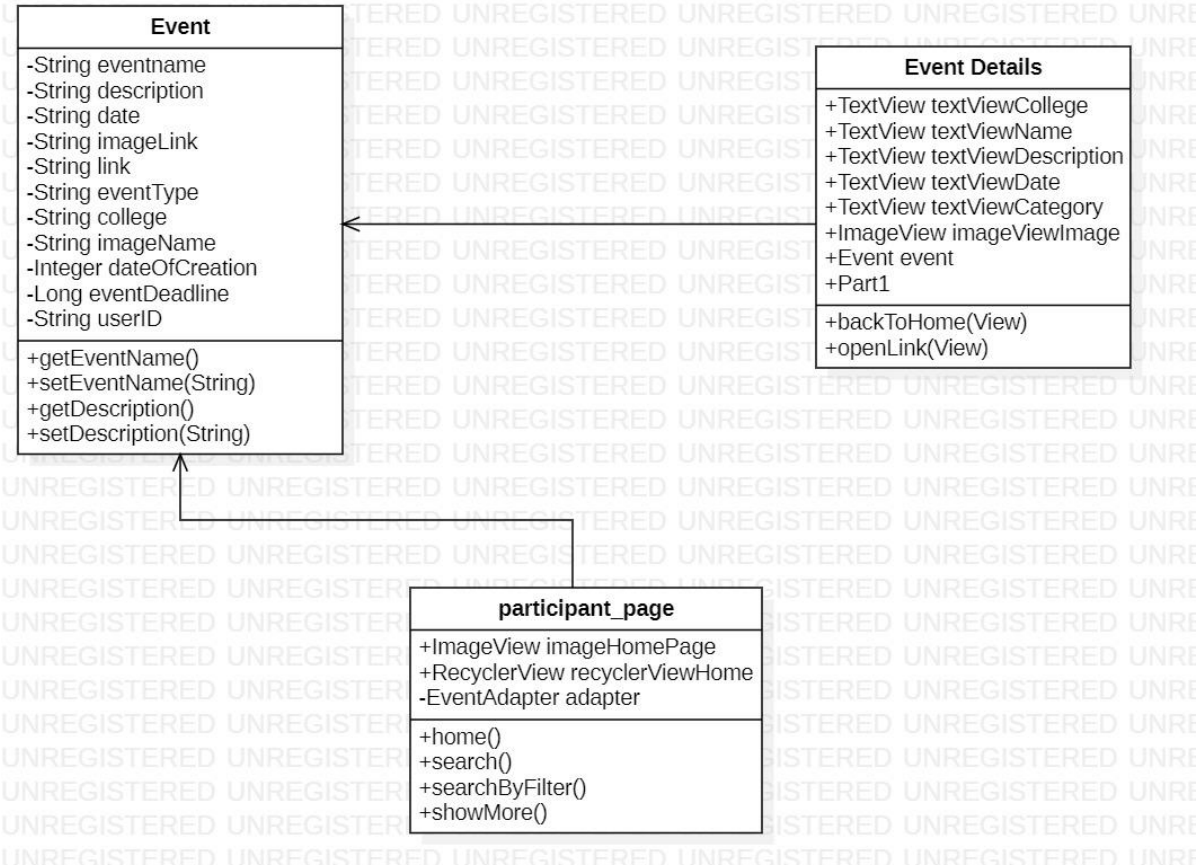
Sign in or Sign up as an organiser



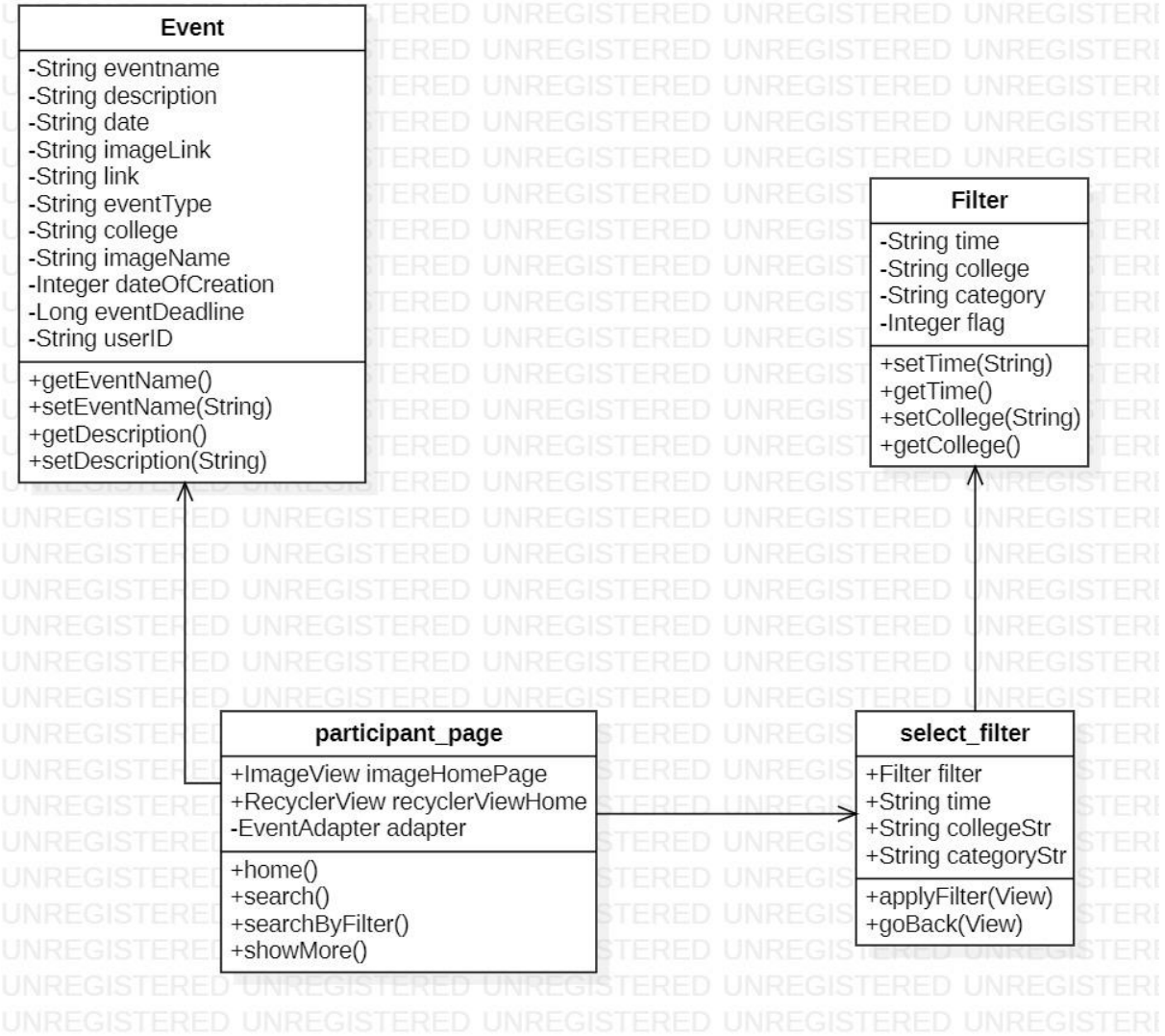
Add event as an organiser



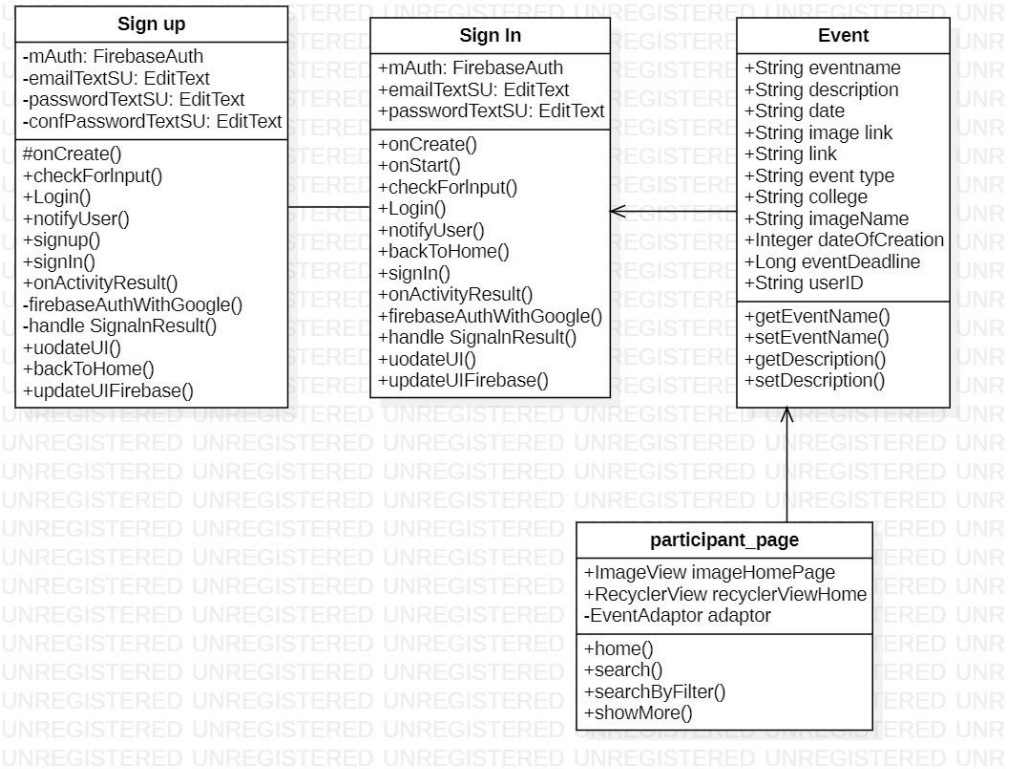
View Event Class Diagram



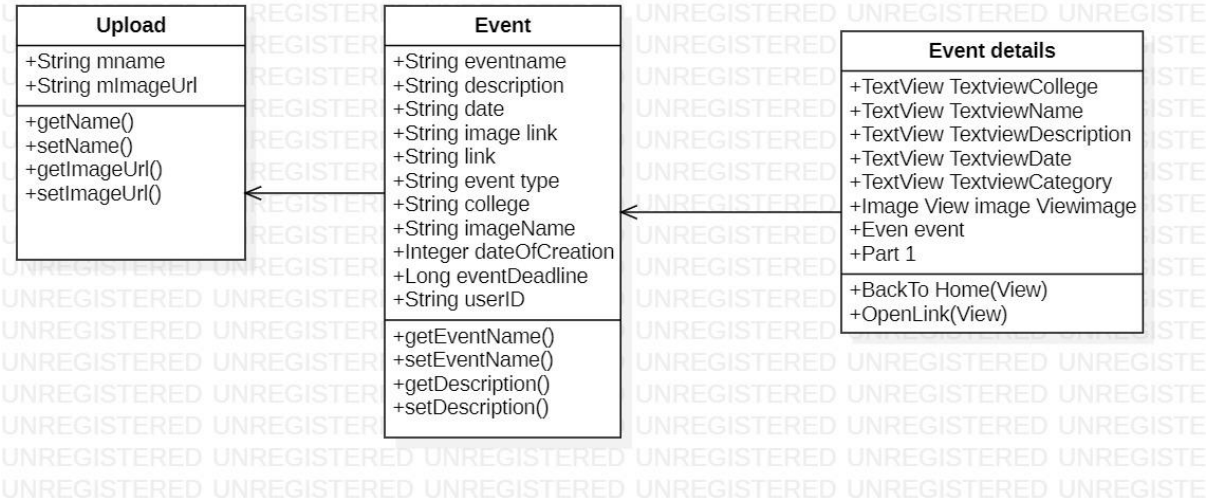
Filter Class Diagram



Sign up /Sign in class and view events (user)

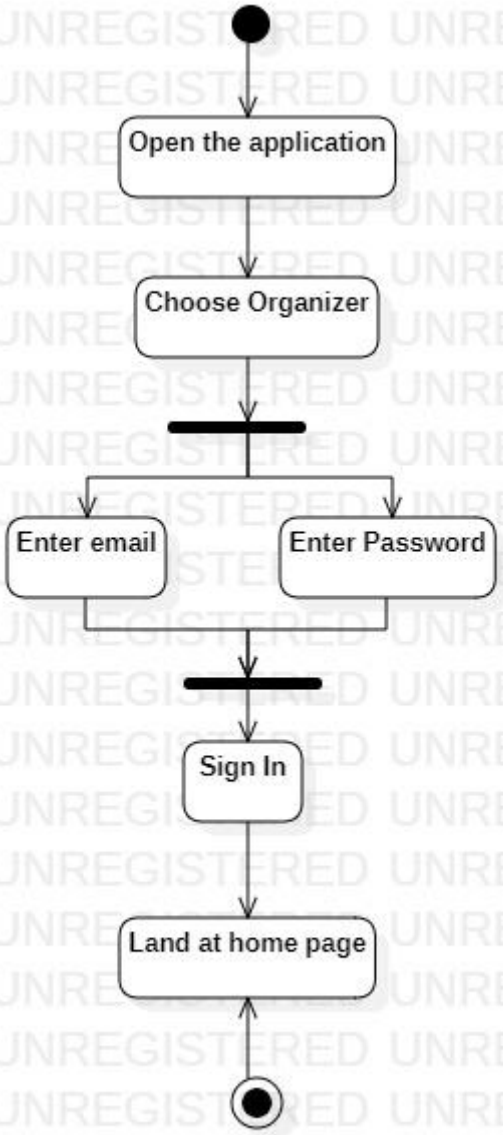


Upload class

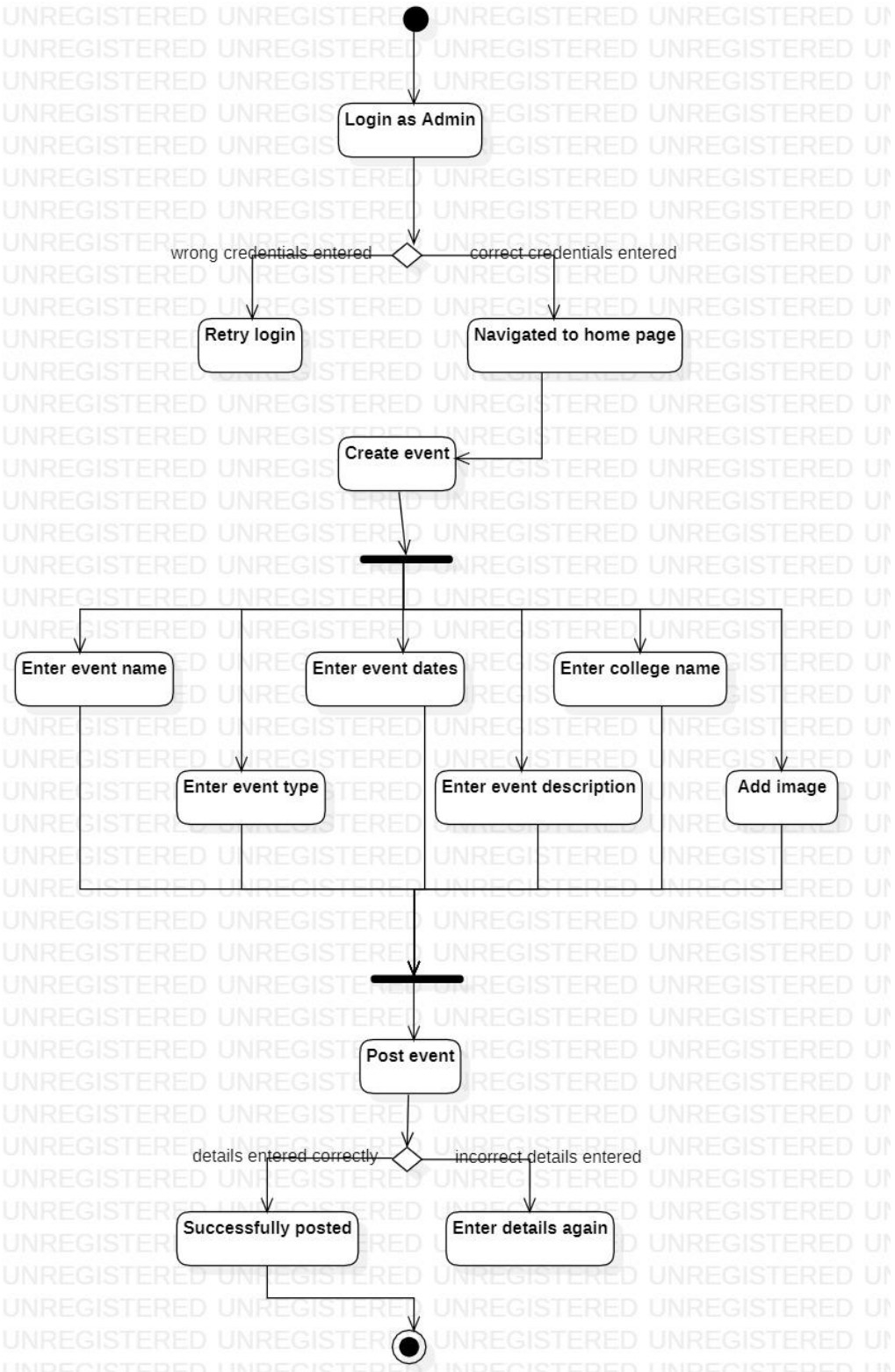


Activity Diagram

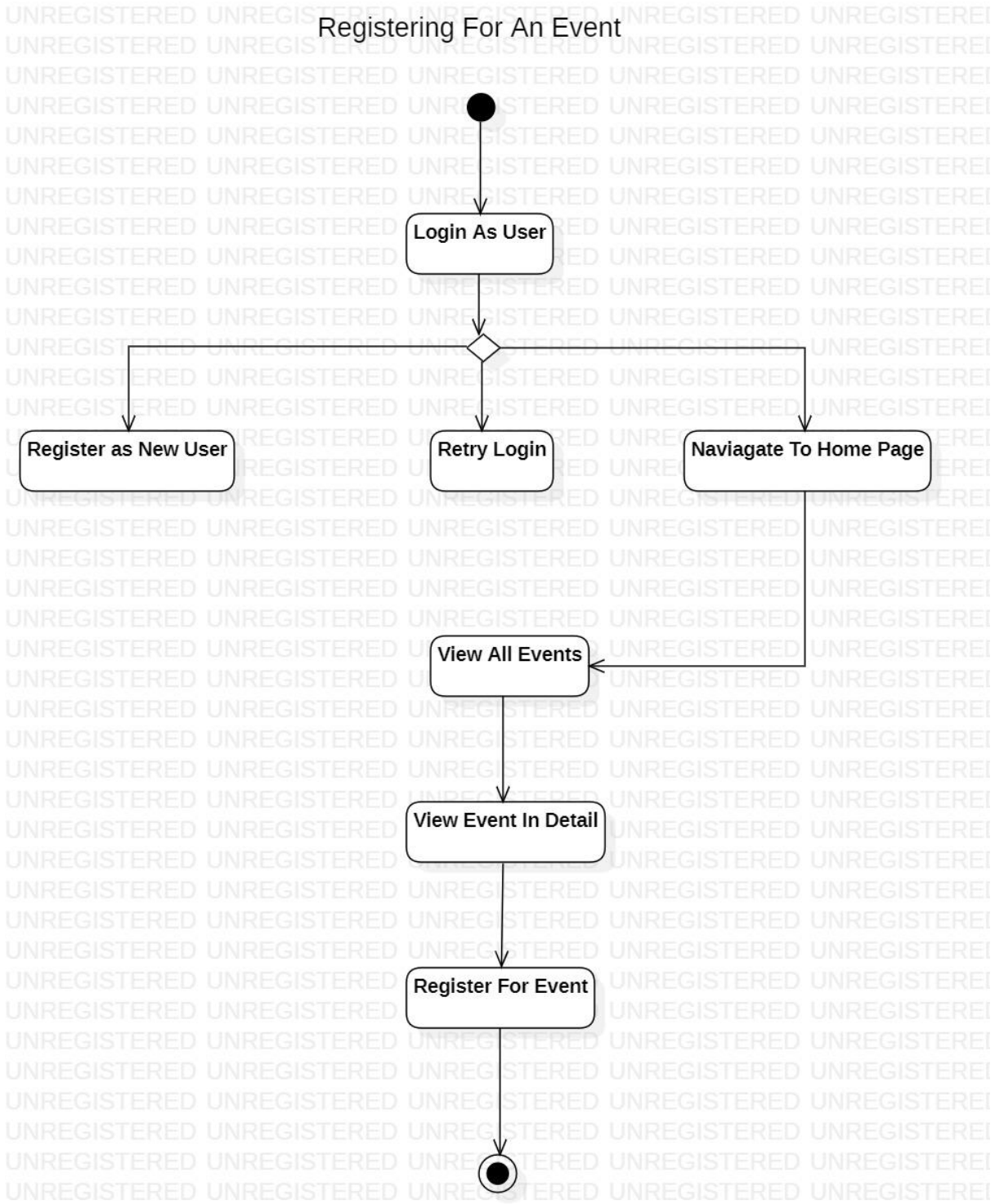
Sign in or sign up as organiser



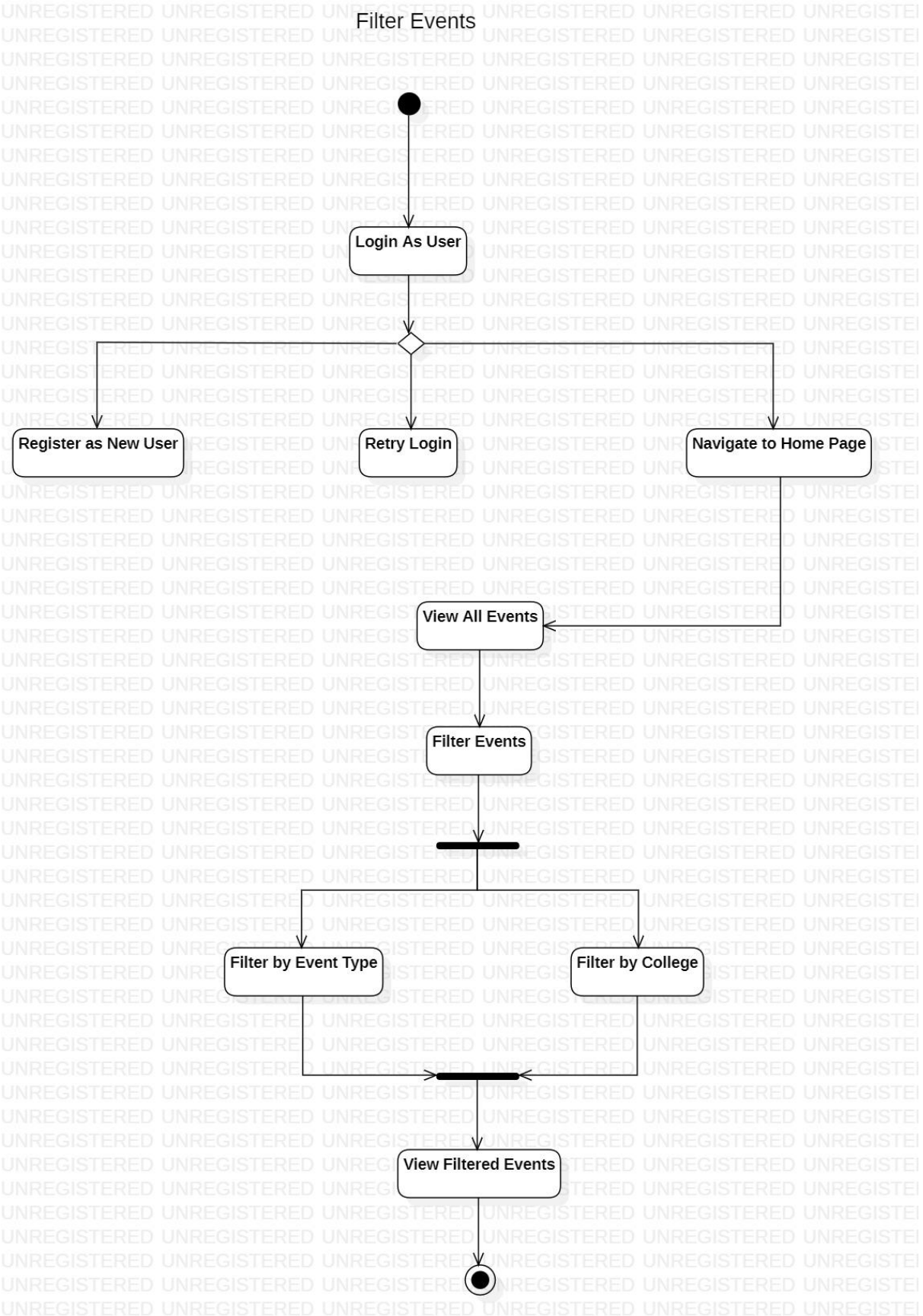
Add events as organiser



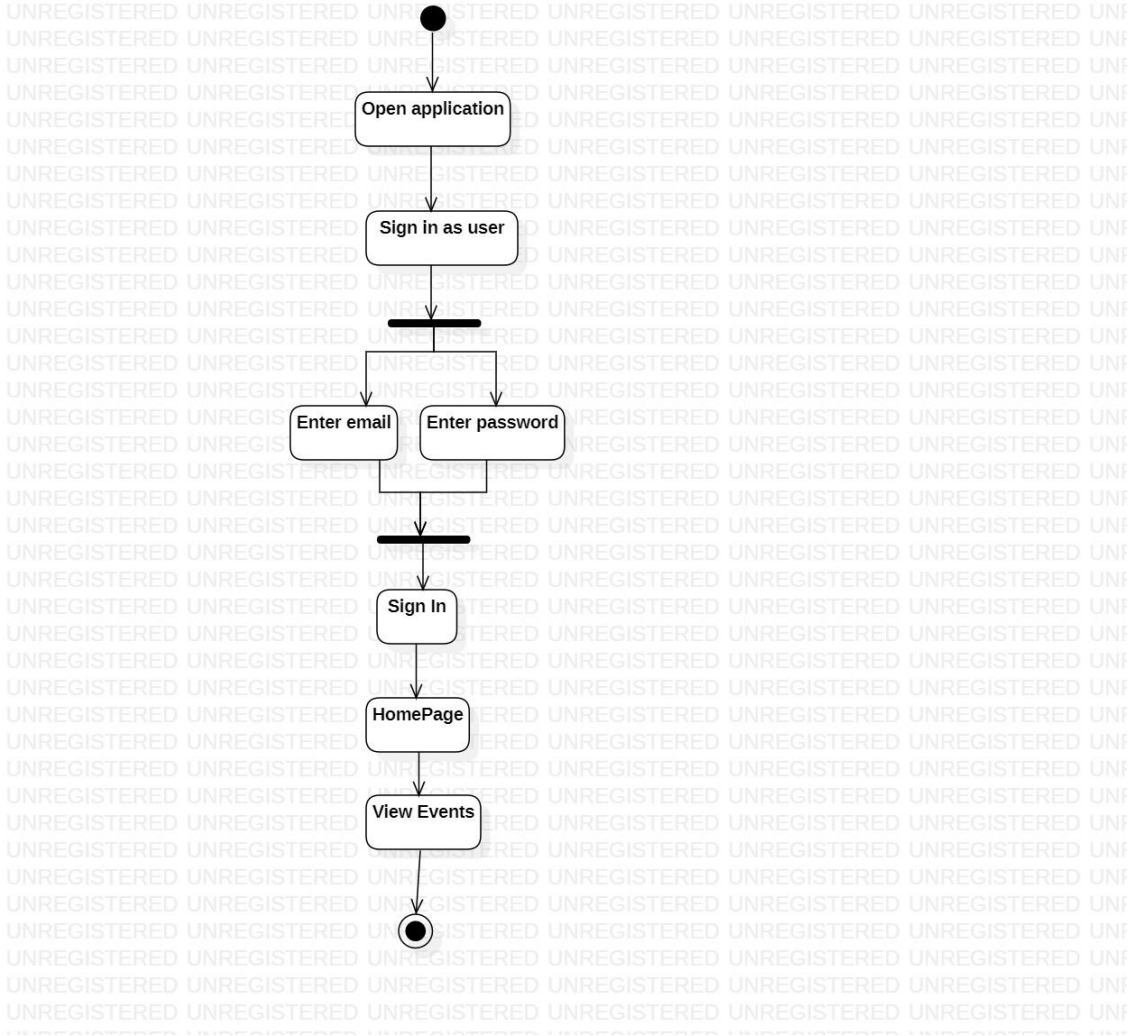
Registering for an event



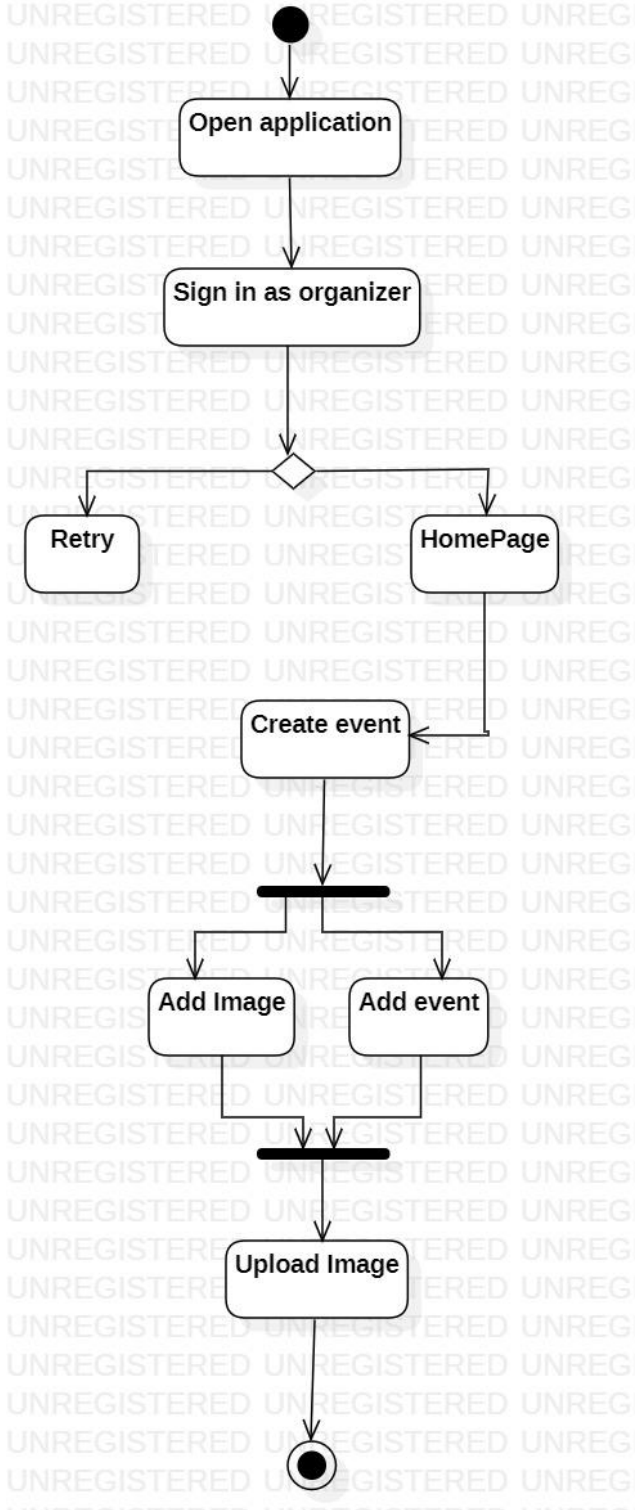
Filtering Events



Sign up /Sign in and view events (user)

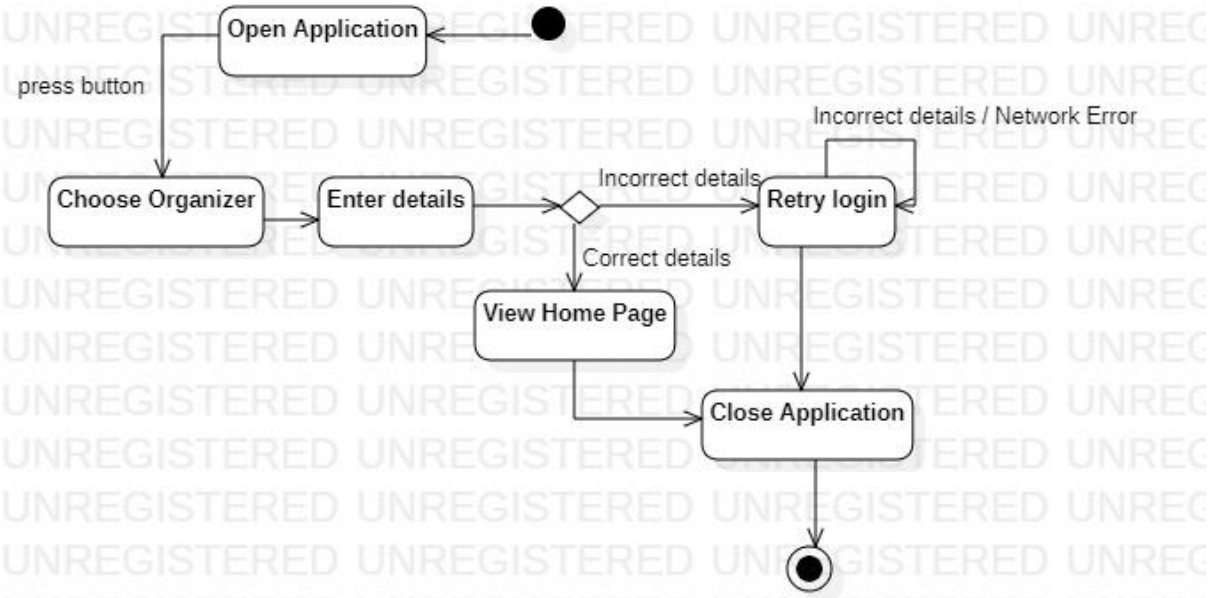


Uploading an image while creating an event

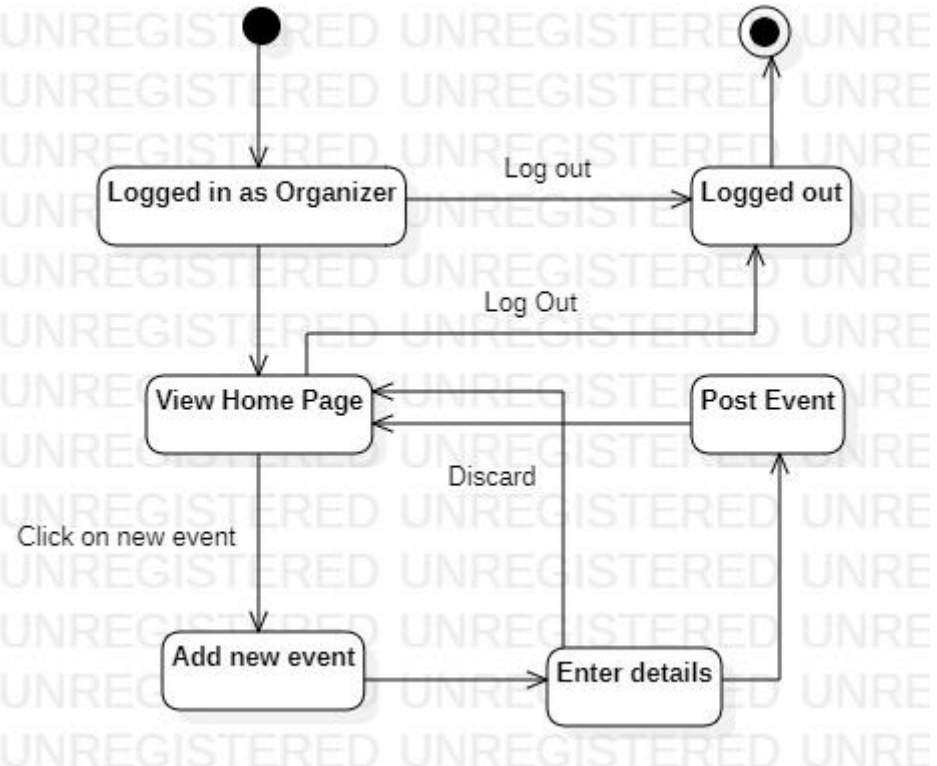


State Diagrams

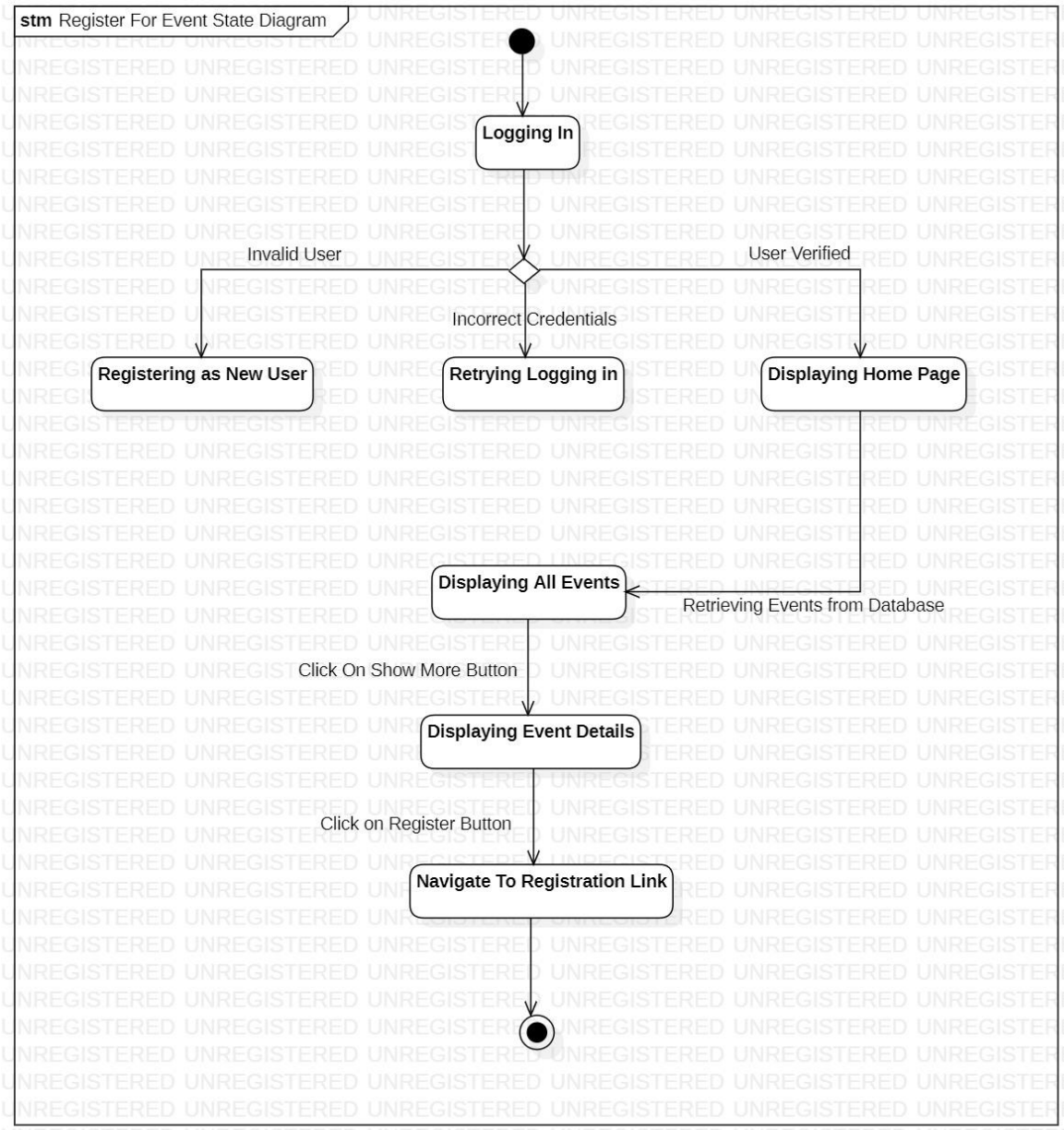
Sign in or sign up as organiser



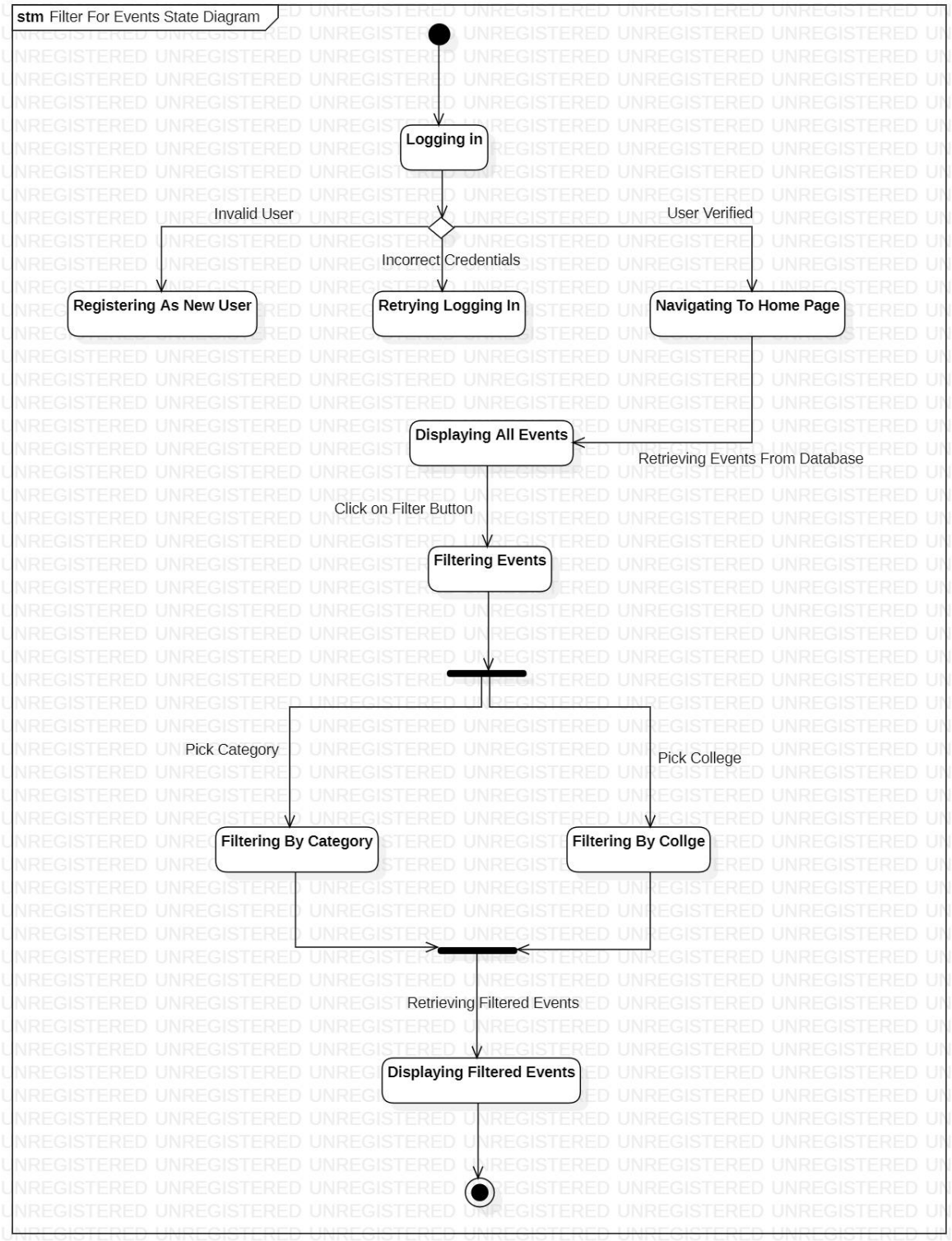
Add event as organiser



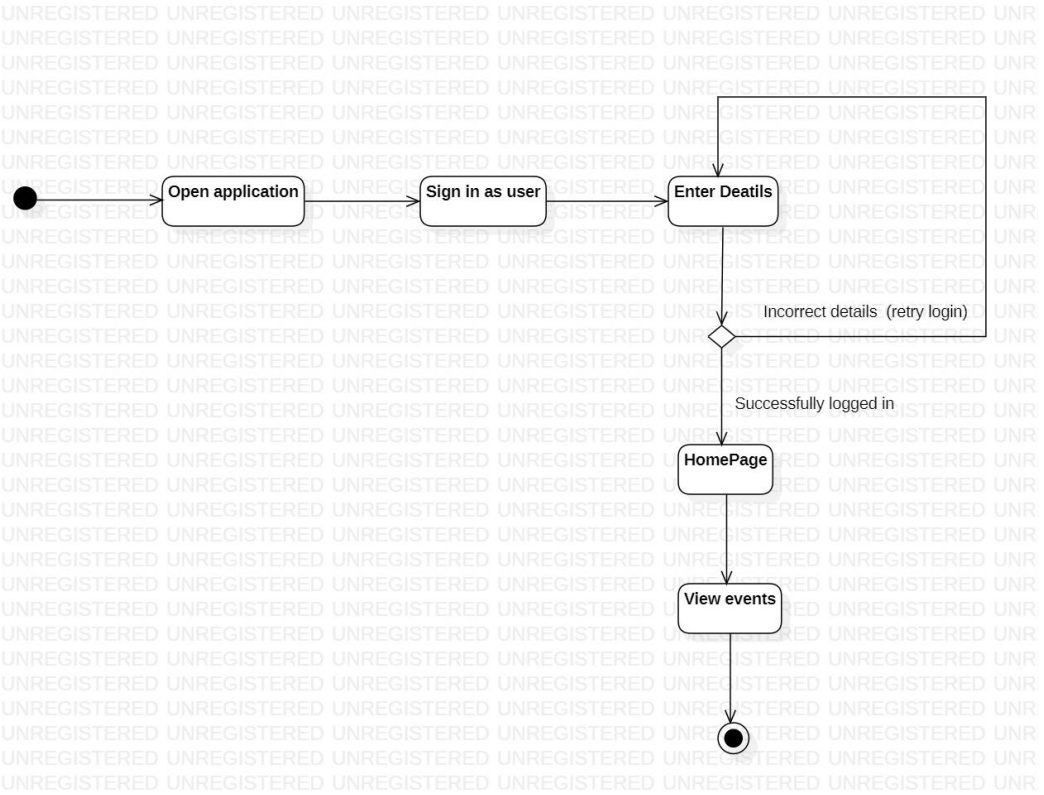
Registering for Event



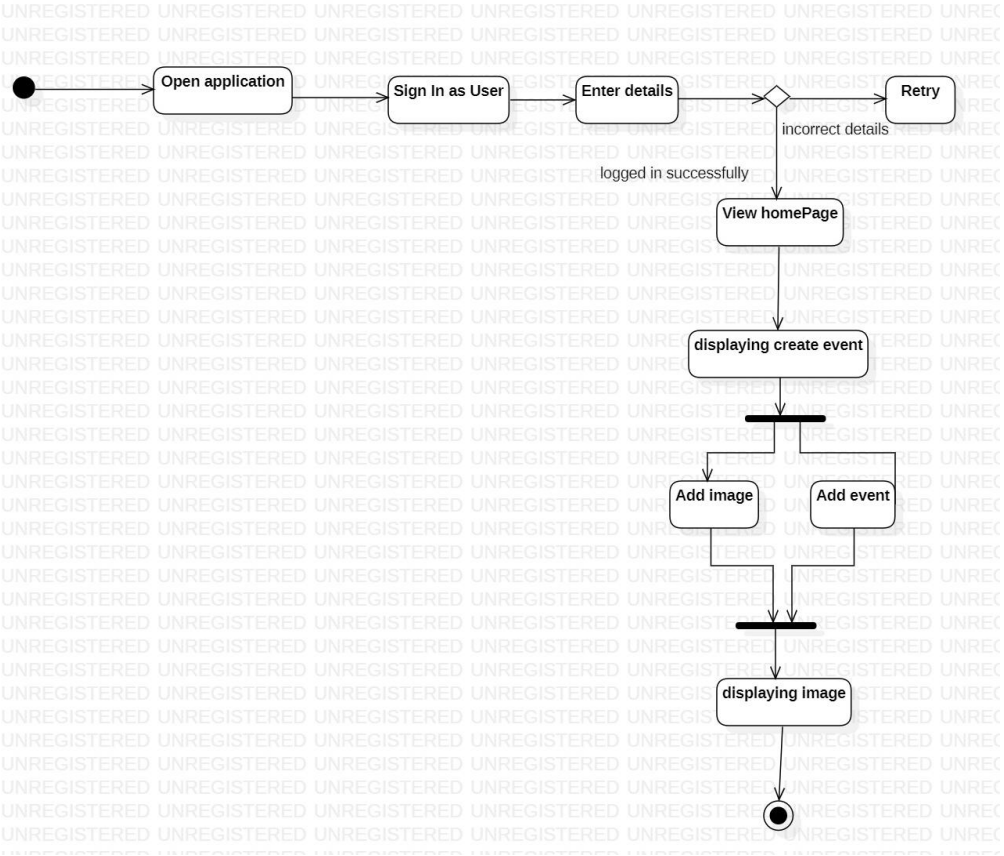
Filtering of event



Sign up /Sign in and view events (user)



Uploading an image while creating an event



3. Tools and Frameworks Used

MVC Framework

MVC (Model View Controller) is a design pattern in software design commonly used for implementing user interfaces, data and controlling logic.
The main aim of this design pattern is to lay a strong emphasis on the separation between the application's logic and the display. This proves to help in division of labour and improved maintenance.

The application built here as a part of the project implements the above design pattern. There is a clear separation between the UI of the app and the business logic and the communication is taken care of by the controller.

The **Model** component contains all the data related logic that the user works with. This can either represent the data transferred between the View and the Controller or any other business logic data. In this project, the Model primarily

deals with the data that is in flow between the View and the Controller (event details and user information) and also the business related logic, such as filtering out the events, searching for a specific event etc.

The **View** component contains all the UI logic of the application. The components include text boxes, drop downs, buttons etc. These are used by the user to interact with the application in terms of sending and receiving data. This application includes a variety of UI components such as buttons, text boxes, event cars and so on, which the user can click upon to interact.

The **Controller** acts as an interface between the Model and the View to process all the business logic and handle the incoming data requests, manipulate data and send data out to render on the output. Here the controller takes in data from the UI when the admin creates an event and passes it on to the Model and helps return event and user information back to View to be rendered on the output.

4. Design Principles and Design Patterns Applied

Adapter

Adapter pattern is a software design pattern that allows the interface of an existing class to be used as another interface. It is often used to make existing classes work with others without modifying their source code.

In our project we have used adapter design for viewing events as follows:

1. The user makes a request to the adapter by calling a method(viewing events) on it using the target interface.
2. The adapter translates that request on the adaptee(creating event) using the adaptee interface.
3. The client receives the result of the call `````(list of events) and is unaware of the adapters presence.

Single Class Responsibility

In SOLID, the first letter talks about Single Class Responsibility. This principle states that “A class should have and only one responsibility to fulfil and only one reason to change”.

Following this principle makes the software a lot easier to implement and prevents unexpected side effects in case of any changes made to the class in the future.

It is very well known that in the process of developing a software application, requirements frequently change over time. These requirements could result in changing the responsibility of at least one class. If the class ends up having multiple responsibilities to fulfil, then the classes are no longer independent of each other. As soon as one of its responsibilities changes, the class must also reflect that change. Depending on the change, the team might be required to update dependencies or re-compile the dependent classes although they are not directly affected by the change.

In the end, classes would undergo changes very frequently and each change could be complicated, expensive and produce more side-effects. This will also increase the time and the effort spent on changs to be made. This clearly emphasises on the need to always follow this principle.

In this project, single class responsibility has been very well implemented. The organiser and the user have multiple activities and functions to perform on the application. Keeping them under one class each for the user and the organiser would make it difficult to make changes in the future. To prevent the above mentioned difficulties during development and being open to reflect changes based on the new requirements would make it easy to implement.

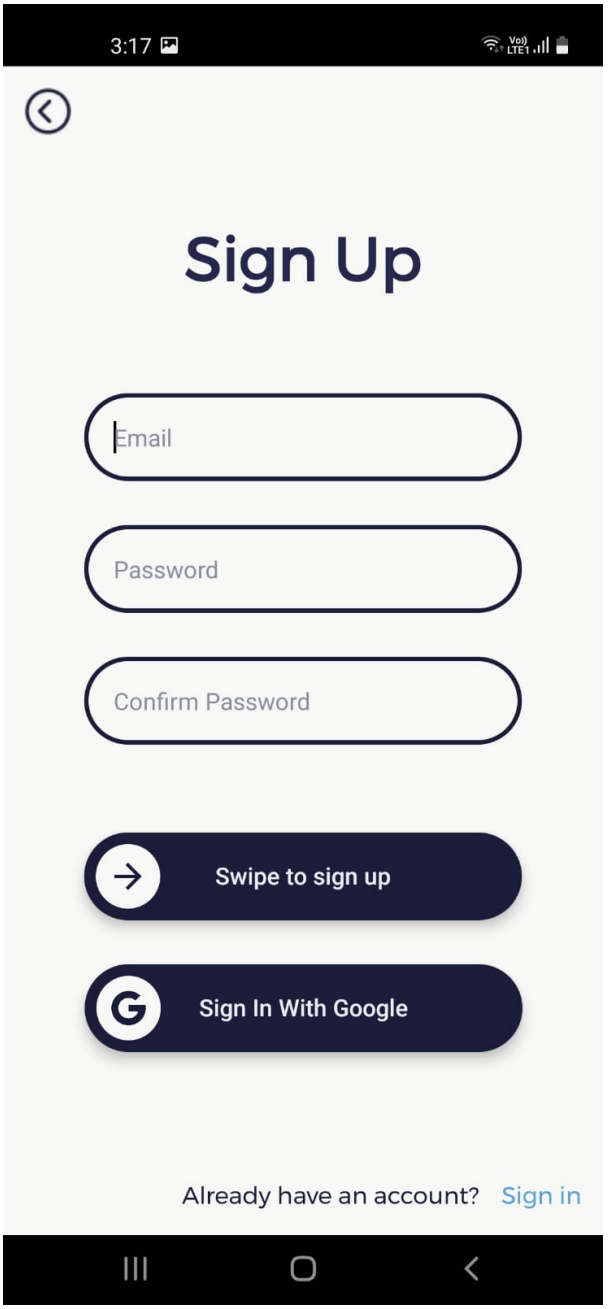
Low coupling and High Cohesion

Low coupling talks about how different modules depend on each other. Modules typically must be as independent as possible from the other modules so that changes made to a specific module would not heavily impact the other modules.

Cohesion often refers to how the elements of a module belong together. Related code should be close to each other to make it highly cohesive. Easy to maintain code usually has high cohesion. The elements within the module are directly related to the functionality that the module is meant to provide.

In this application, each user and organiser have their functionality split up into multiple classes. This makes them less dependent on each other which indicates the existence of low coupling and high cohesion.

5. Application Screenshots (3-4 important pages)



3:18

AiGHT

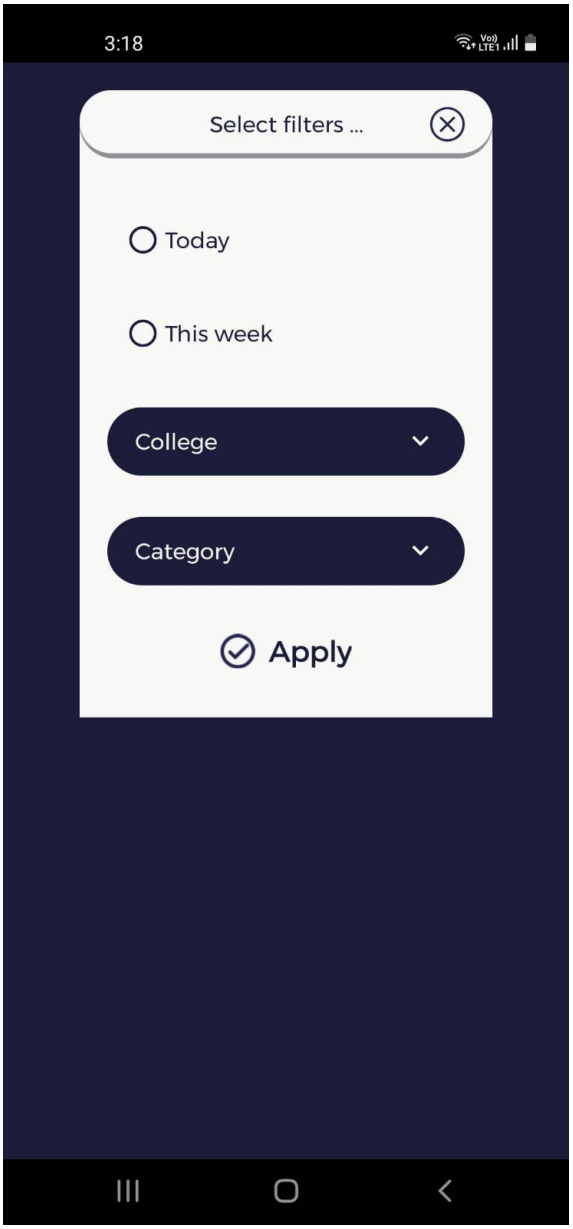
Art contest

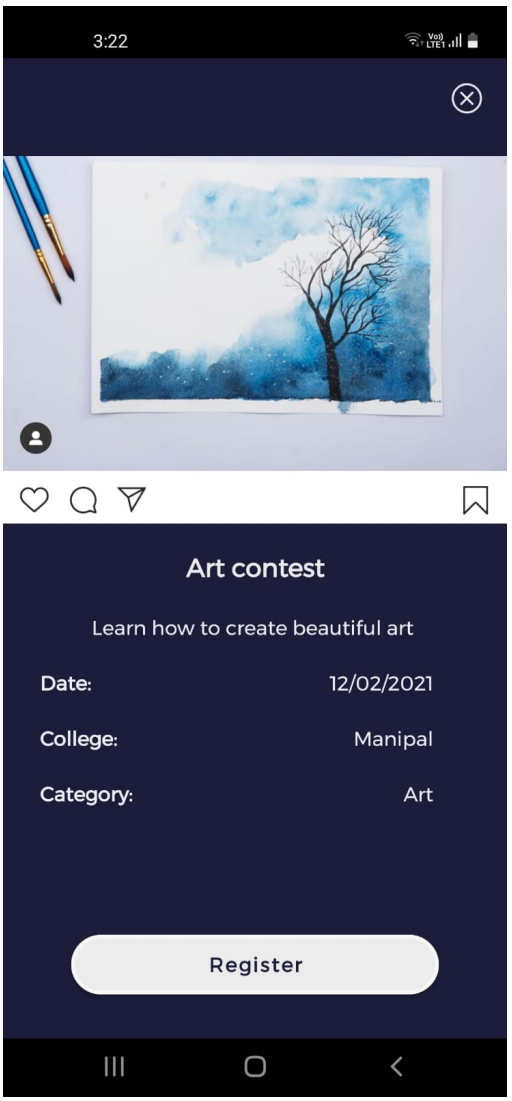
Art 12/02/2021

Manipal

ZEN TIME

Home


3:18
Select filters ...
Today
This week
College
Category
Apply



7. Team member contributions

Name	SRN	Contribution
Lakshmi Narayan P	PES2UG19CS200	33%
Namith Telkar	PES2UG19CS246	33%
Netra Shaligram	PES2UG19CS253	33%