REPORT

Basic Outline

- Organiser:

  - Organiser registers himself by giving a username and password and creates an account. If the same username already exists for any other participant it pops an error otherwise a tuple for organiser is created and an id is given to him. He then logins himself.

  - He can now create events by giving the details of the event. A unique event id is given. He can search all the events present but can edit those only created by him. He can see all the details of the event like volunteers, venue, date, description etc. He can see the winners if the event is over.

- External Participant

- External participants registers himself by giving a username and password and creates an account. If the same username already exists it pops an error otherwise a tuple for external participant is created and an id is given to him.

- He then logins himself. He can now search for events and register for them and know the winners of it.

- Student

- Student directly logins himself by giving roll as username and password assigned to him.He can now search for events and see their details and register for them.
- He has the option to register himself as a volunteer by giving event id of the event he wants to volunteer.

Administrator

- Administrator has the right to add or delete any kind of user directly. Appropriate triggers are there to delete an event if all its organisers are deleted/delete volunteers if the event is deleted.

Schema

1)

Participant=(id,participant_name,college_name,college_location,username,password);

2)
Student=(roll,student_name,dept_name,username,password)

3)
Organiser=(id,username,password,first_name,last_name)

4)

Event=(id,event_name,event_type,date,time,venue,description,organiser_id)

5)

Volunteer=(id,roll,event_id)

6)
Organiser_Event=(id,organiser_id,event_id)

7)

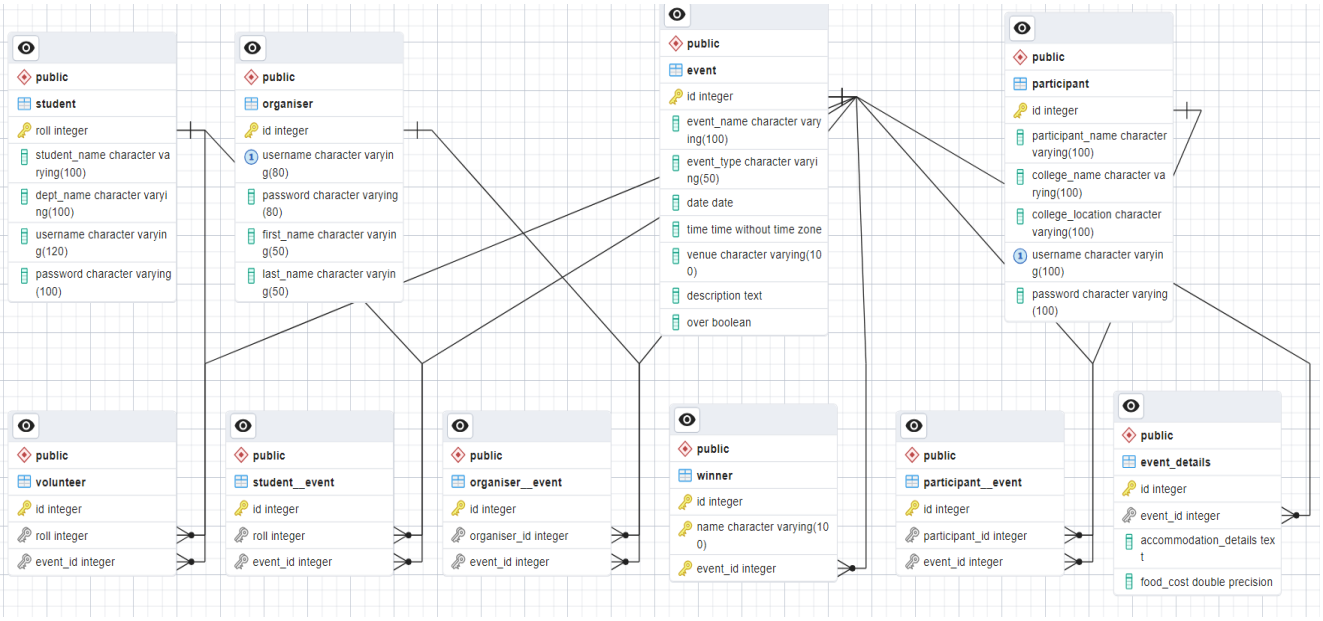
Participant_Event=(id,participant_id,event_id)

8)

Student_Event=(id,roll,event_id)

9)Admin=(id,username,password)

Entity Relationship Diagram



Triggers:

1)Deleting an event if all organizers of the event are removed by Admin:

```
CREATE OR REPLACE FUNCTION delete_event_if_last_row()
RETURNS TRIGGER AS
$$
```

```
BEGIN
    -- Check if the deleted row is the only row for the event_id
    IF NOT EXISTS (
        SELECT 1 FROM organiser__event WHERE event_id = OLD.event_id AND id !=
OLD.id
    ) THEN
        -- If it is the only row, delete the corresponding event from the event table
        DELETE FROM event WHERE id = OLD.event_id;
    END IF;

    RETURN OLD;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER check_last_row_delete_event_trigger
AFTER DELETE ON organiser__event
FOR EACH ROW
EXECUTE FUNCTION delete_event_if_last_row()
```

Function delete_event_if_last_row: This function is created with the CREATE OR REPLACE FUNCTION statement. It returns a trigger, indicating that it can be used as a trigger function. The function is written in PL/pgSQL, which is a procedural language for PostgreSQL.

Function Logic: Inside the function, there's an IF statement that checks if there are any other rows in the organiser__event table with the same event_id as the one being deleted (OLD.event_id). If there are no other rows (NOT EXISTS), it means the row being deleted is the last row associated with that event_id. In that case, it proceeds to the DELETE statement, which deletes the corresponding event from the event table using the event_id.

Trigger Creation: The CREATE TRIGGER statement defines a trigger named check_last_row_delete_event_trigger. It specifies that the trigger should fire AFTER DELETE on the organiser__event table for each row. The trigger is associated with the trigger function delete_event_if_last_row.

Trigger Execution: The trigger function delete_event_if_last_row will be executed automatically after each DELETE operation on the organiser__event table. It checks if the deleted row is the last one associated with a specific event_id and deletes the corresponding event if necessary.

2) delete_event_cascade Function:

Description: This PostgreSQL function is designed to be triggered when a row in the event table is deleted. It performs a cascading delete by removing corresponding records from the event_details table where the event_id matches the deleted event's id.

```
t_function_sql = text("""

CREATE OR REPLACE FUNCTION delete_event_cascade()
```

```
RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM event_details WHERE event_id = OLD.id;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
""")
```

3) event_delete_trigger Trigger:

Description: This trigger is set to fire before a row is deleted from the event table. It is associated with the delete_event_cascade function, executing it for each deleted row.

```
t_sql = text("""
CREATE TRIGGER event_delete_trigger
BEFORE DELETE ON event
FOR EACH ROW
EXECUTE FUNCTION delete_event_cascade();
""")
```

4) delete_participant_cascade :

Description: This PostgreSQL function serves as a trigger function for cascading deletes associated with the participant table. It removes records from the participant__event table where the participant_id matches the deleted participant's id. Purpose: To maintain data consistency by deleting participant-event associations when a participant is removed.

```
trigger_function = text("""
CREATE OR REPLACE FUNCTION delete_participant_cascade()
RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM participant__event WHERE participant_id = OLD.id;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
""")
```

5) participant_delete_trigger :

Description: This trigger is set to execute before a row is deleted from the participant table. It is linked to the delete_participant_cascade function, ensuring that associated participant-event records are deleted before the participant is removed.

```
trigger = text("""
CREATE TRIGGER participant_delete_trigger
BEFORE DELETE ON participant
FOR EACH ROW
EXECUTE FUNCTION delete_participant_cascade();
""")
```

6)student_delete_trigger :

Description: This trigger is set to execute before a row is deleted from the student table. It is linked to the student_before_delete function, ensuring that associated volunteer records are deleted before the student is removed.

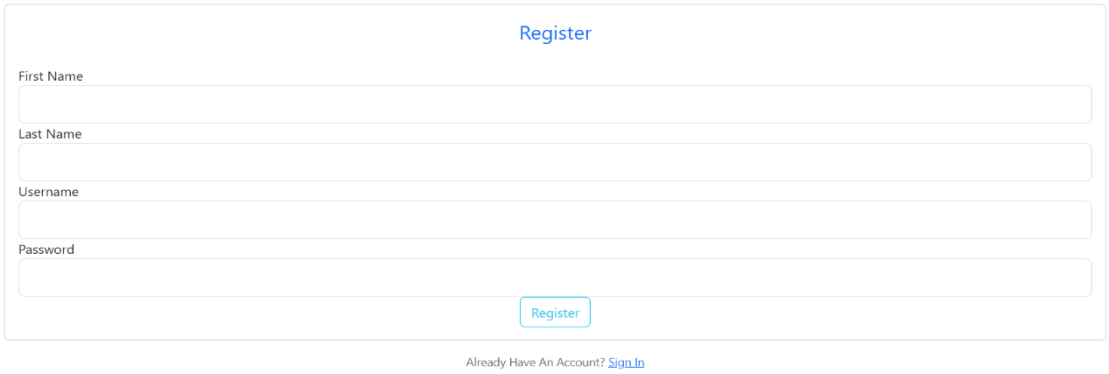
```
trigger_function_sql = text( """
CREATE OR REPLACE FUNCTION student_before_delete()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT COUNT(*) FROM volunteer WHERE roll = OLD.roll) > 0 THEN
        DELETE FROM volunteer WHERE roll = OLD.roll;
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
""")
```

Forms:

1) Organizer Registration

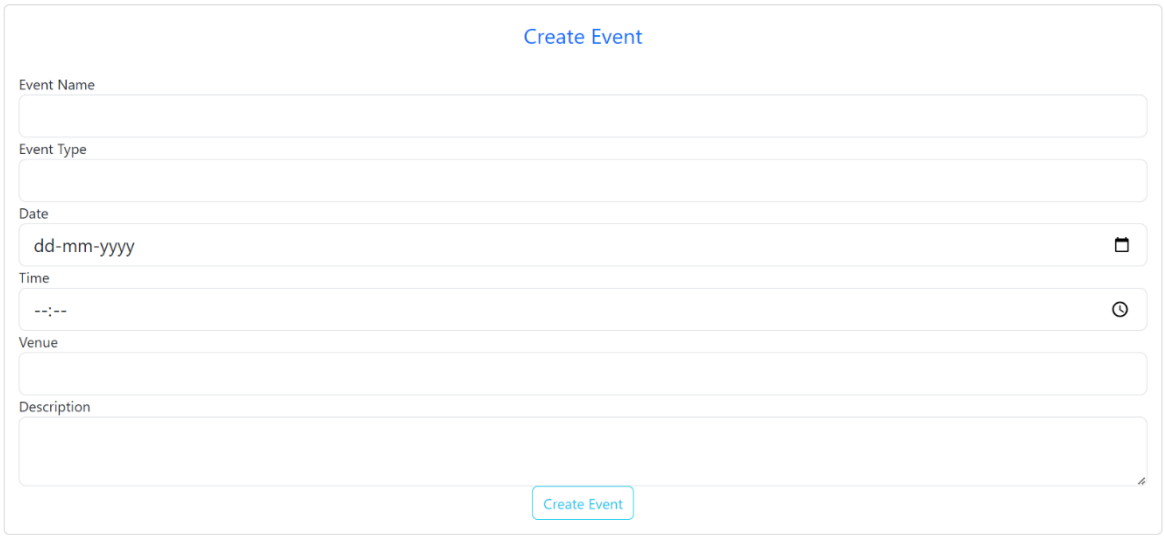
```
2) class OrganizerRegistrationForm(FlaskForm):
3)     username = StringField('Username', validators=[DataRequired(),
   Length(min=2, max=20)])
4)     password = PasswordField('Password', validators=[DataRequired()])
5)     first_name = StringField('First Name', validators=[DataRequired(),
   Length(min=2, max=50)])
```

```
6)      last_name = StringField('Last Name', validators=[DataRequired(),
    Length(min=2, max=50)])
7)      submit = SubmitField('Register')
```



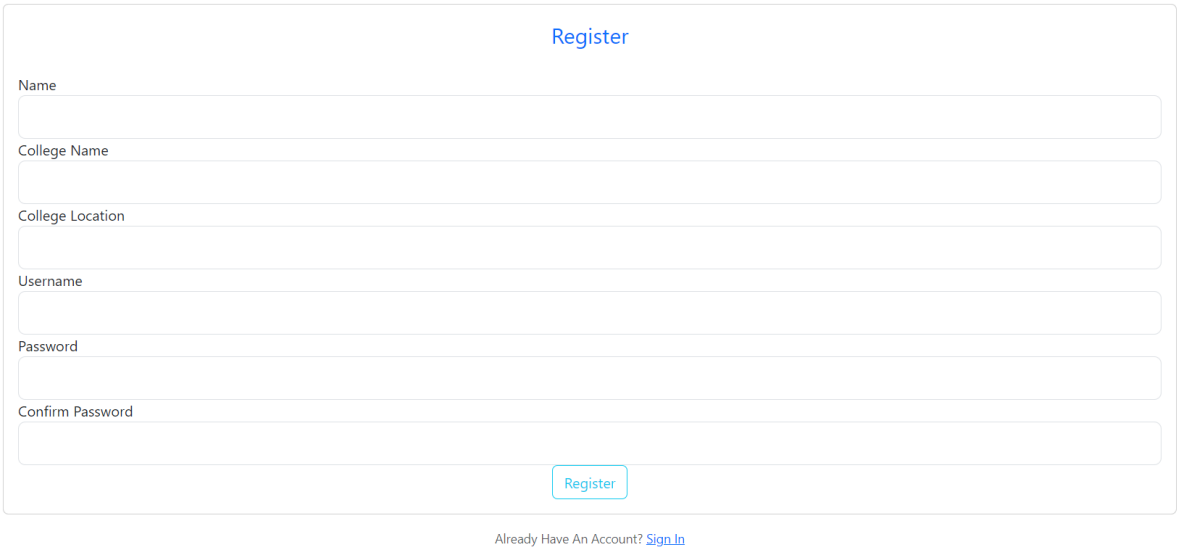
2)Create Event

```
class EventForm(FlaskForm):
    event_name = StringField('Event Name', validators=[DataRequired()])
    event_type = StringField('Event Type', validators=[DataRequired()])
    date = DateField('Date', validators=[DataRequired()])
    time = TimeField('Time', validators=[DataRequired()])
    venue = StringField('Venue', validators=[DataRequired()])
    description = TextAreaField('Description')
    submit = SubmitField('Create Event')
```



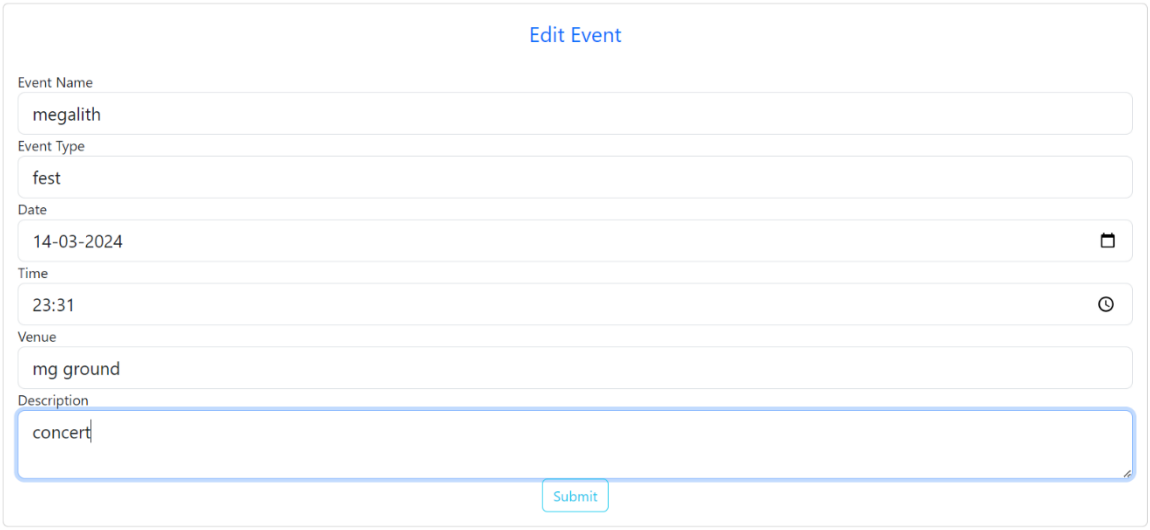
3)Participant Registration

```
class ParticipantRegistrationForm(FlaskForm):
    participant_name = StringField('Name',validators=[DataRequired()])
    college_name = StringField('College Name',validators=[DataRequired()])
    username = StringField('Username',validators=[DataRequired()])
    college_location = StringField('College
Location',validators=[DataRequired()])
    password = PasswordField('Password',validators=[DataRequired()])
    confirm_password = PasswordField('Confirm
Password',validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Register')
```



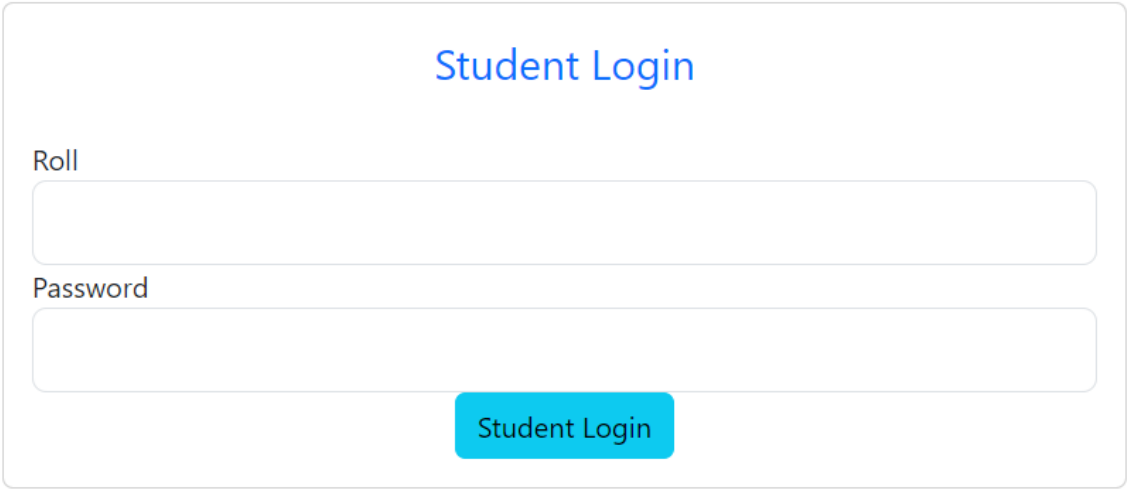
4)Edit Event

```
class EditEventForm(FlaskForm):
    event_name = StringField('Event Name', validators=[DataRequired()])
    event_type = StringField('Event Type', validators=[DataRequired()])
    date = DateField('Date', validators=[DataRequired()])
    time = TimeField('Time', validators=[DataRequired()])
    venue = StringField('Venue', validators=[DataRequired()])
    description = TextAreaField('Description')
    submit = SubmitField('Submit')
```



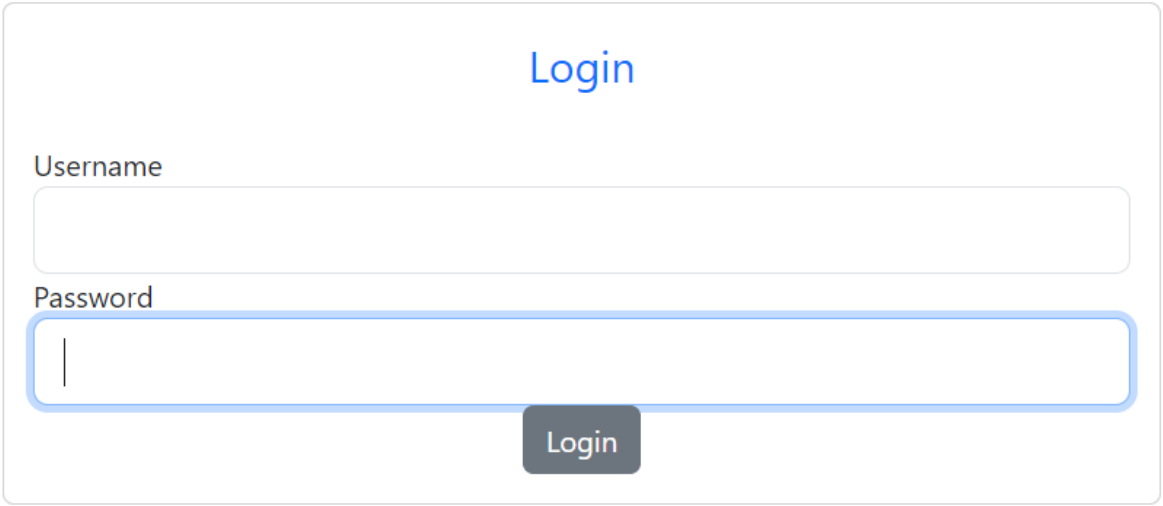
5)Student Login

```
class StudentLoginForm(FlaskForm):
    roll = StringField('Roll', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Student Login')
```



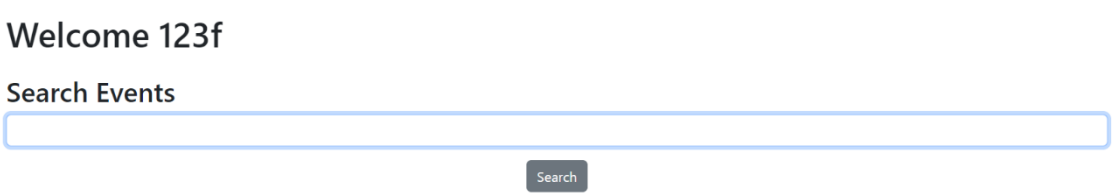
6) Login

```
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')
```



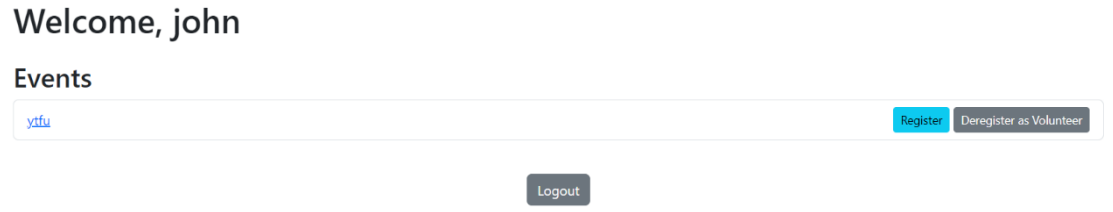
7)Search

```
class SearchForm(FlaskForm):
    search = StringField('Search Events', validators=[DataRequired()])
    submit = SubmitField('Search')
```



8)Student Registration

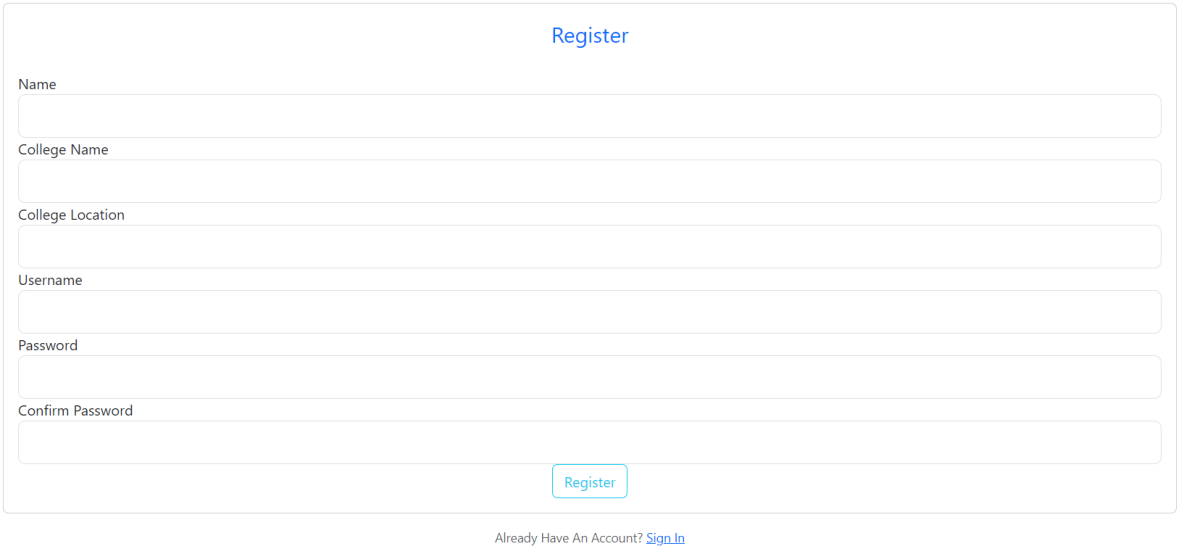
```
class StudentRegistrationForm(FlaskForm):
    roll = StringField('Roll', validators=[DataRequired()])
    name = StringField('Name', validators=[DataRequired()])
    department = StringField('department', validators=[DataRequired()])
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    # confirm_password = PasswordField('Confirm Password',
validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('add_student')
```

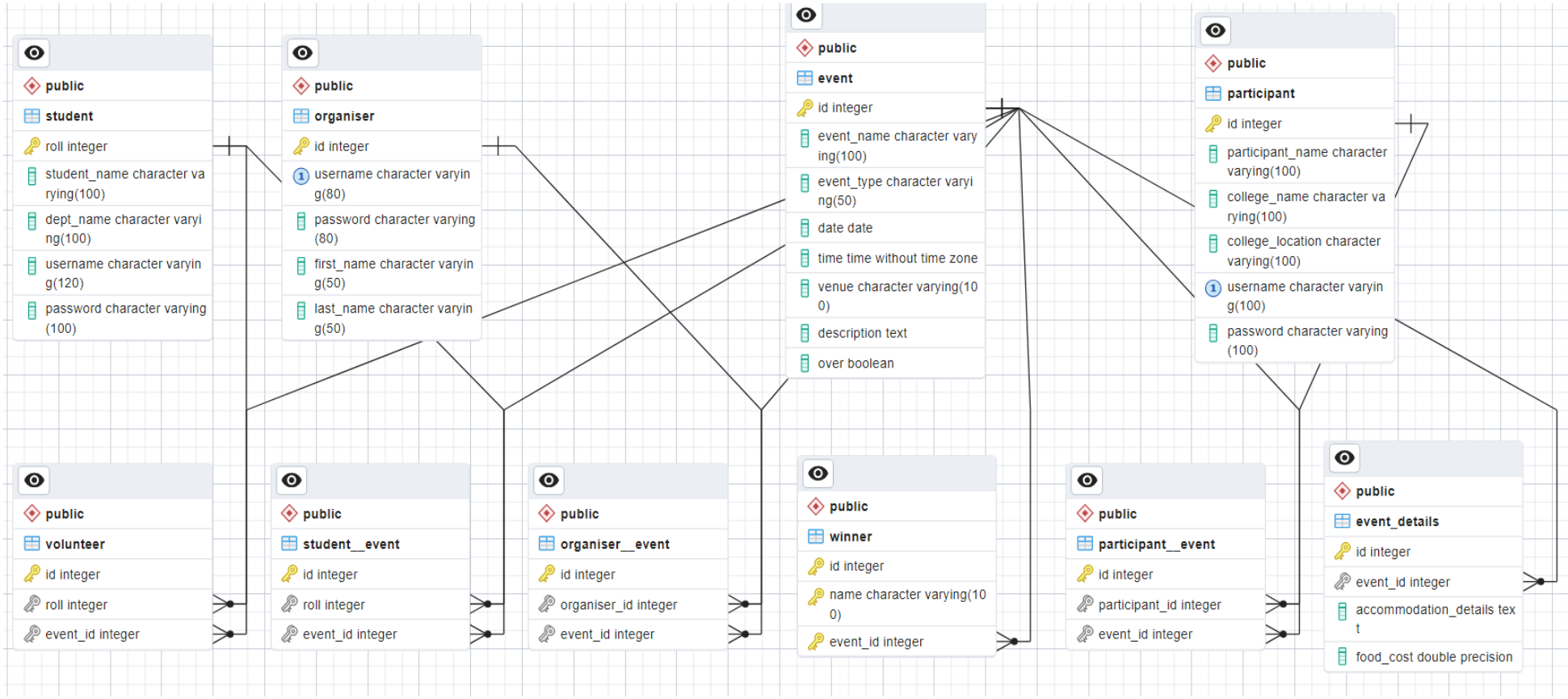


9)Participant Registration

```
class ParticipantRegistrationForm1(FlaskForm):
    participant_name = StringField('Name',validators=[DataRequired()])
    college_name = StringField('College Name',validators=[DataRequired()])
```

```
    username = StringField('Username',validators=[DataRequired()])
    college_location = StringField('College
Location',validators=[DataRequired()])
    password = PasswordField('Password',validators=[DataRequired()])
    # confirm_password = PasswordField('Confirm
Password',validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Add Participant')
```




public
student
roll integer
student_name character varying(100)
dept_name character varying(100)
username character varying(120)
password character varying (100)
public
organiser
id integer
username character varying(80)
password character varying (80)
first_name character varying(50)
last_name character varying(50)
public
event
id integer
event_name character varying(100)
event_type character varying(50)
date date
time time without time zone
venue character varying(100)
description text
over boolean
public
participant
id integer
participant_name character varying(100)
college_name character varying(100)
college_location character varying(100)
username character varying(100)
password character varying (100)
public
volunteer
id integer
roll integer
event_id integer
public
student__event
id integer
roll integer
event_id integer
public
organiser__event
id integer
organiser_id integer
event_id integer
public
winner
id integer
name character varying(100)
event_id integer
public
participant__event
id integer
participant_id integer
event_id integer
public
event_details
id integer
event_id integer
accommodation_details text
food_cost double precision