College Festival Management Web Application Development

Group name: Three Musketeers

Sreejita Saha (21CS30052)
Anwesha Das (21CS30007)
Ratan Junior (21CS30041)
and the friends we made along the way

Introduction

The University Cultural Festival Management System is a web application designed to efficiently organise and manage a university's cultural festival. The system caters to different user roles, including external participants, students, volunteers, organisers, and database administrators. The system's functional requirements cover the entire event life-cycle, from registration to real-time event updates and logistics management.

System Architecture

This web-based application is built using Flask in python for the front-end and PostgreSQL for back-end database management, supporting triggers and procedures to facilitate the workflow.

Functional Requirements

1. External Students

1.1 Account Creation and Event Browsing

- External students from other colleges can **create accounts** by signing up on the website by entering `Name`, `Email`, `Password` and their `College Name`.
- They have the ability to **browse** for events and their respective details such as name, schedules, description and winners if updated.
- Information about **event winners** is accessible after the completion of the event as will be updated by the organisers.

1.2 Event Registration and Logistics

- External participants can **register** for more than one specific events.
- Access to **logistics**, such as accommodation in the form of `hall` and food in the form of `canteen` allocated, is provided through the web application.

2. College Students

2.1 Event Browsing and Registration

- Students can **create accounts** by signing up on the website by entering `Name`, `Email`, `Password`, `Roll Number`, `Department Name` and `Hall Name`.
- Students can **register** and participate in more than one specific events.

2.2 Volunteer Registration

- Students that wish not to participate in any event may **register as volunteers** for specific events of the festival.
- One student may wish to volunteer for more than one event.

3. Organisers

3.1 Account Creation and Event Details

- Organisers of the festival can **create their accounts** by signing up on the website by entering `Name`, `Email` and `Password`.
- Similar to external participants and students, organisers have a **detailed view of the events** including their associated volunteers.
- Organisers are also responsible for the **updation of winners** of the specific events after the start of the event though the web portal.

3.2 Event Management:

- In addition to viewing volunteer details, the organiser can remove volunteer from the specific events.
- Organisers can also remove participants in a certain event.

4. Database Administrators

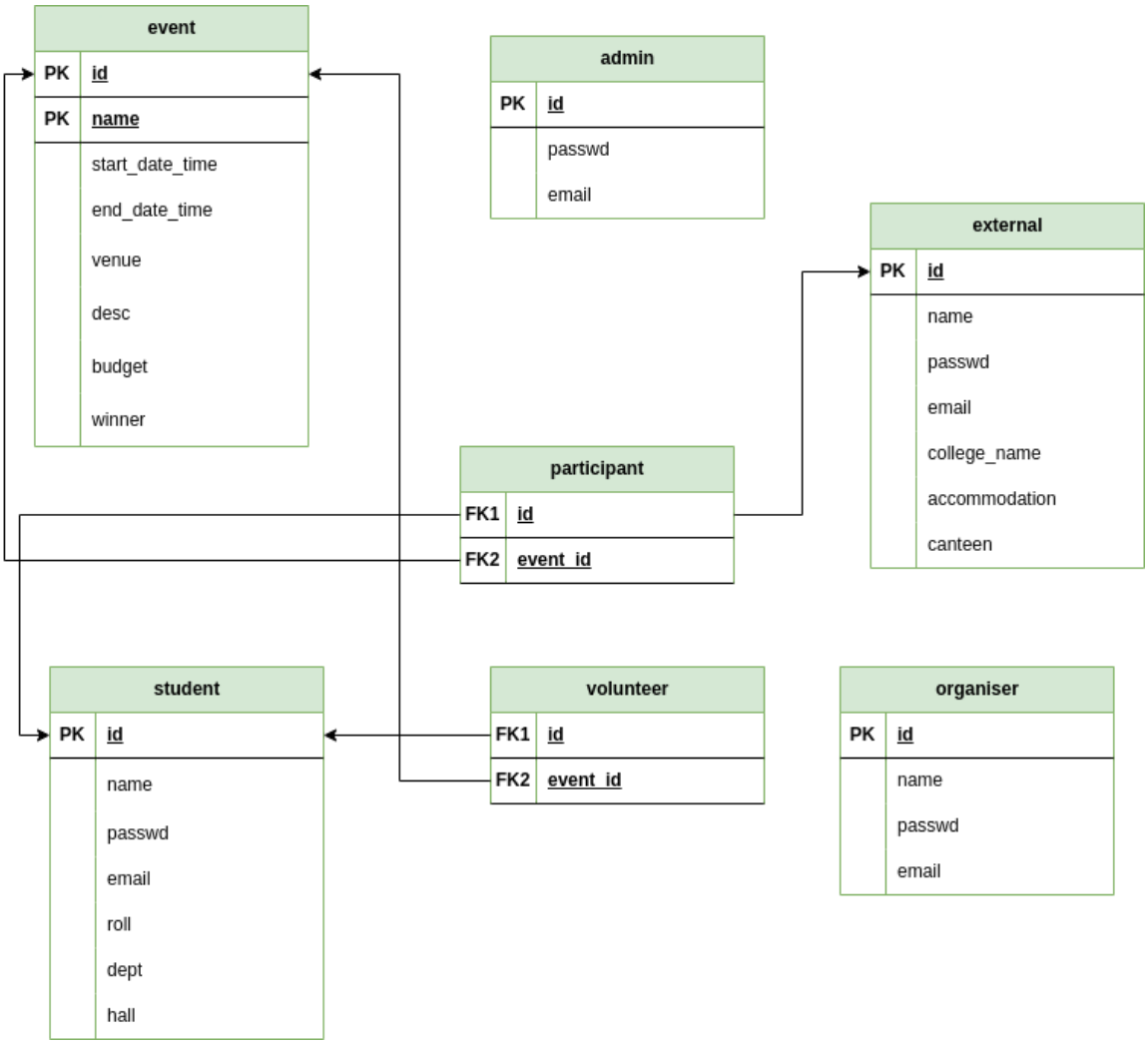
4.1 Supreme Authority

- Database administration is given to a select few individuals whose details are pre-loaded in the corresponding database `admin`.
- These administrators may create their accounts by entering their respective `Username`, `Email` and `Password`.
- Admins must know these details beforehand in order to create an account through the web portal.

4.2 User Management

- Database administrators have **access to existing database of all users** including external participants, students and organisers.
- Administrators can also **add and delete these users** including the organisers through the web portal.
- Details about volunteers, event participants, events and logistics is also available to the administrators.

Database Schema



Triggers

An SQL trigger is an object that is associated with a table and automatically executes a set of SQL statements when a specific event occurs on that table.

`create_org_trigger`:

- This trigger is executed when a new organiser is added to the database and a new user is created after the organiser signs up and hits the submit button.

```
CREATE TRIGGER create_user_trigger
AFTER INSERT ON organiser
FOR EACH ROW
EXECUTE FUNCTION create_user();
```

`create_std_trigger`:

- This trigger is executed when a new student is added to the database and a new user is created after the student signs up and hits the submit button.

```
CREATE TRIGGER create_user_trigger
AFTER INSERT ON student
FOR EACH ROW
EXECUTE FUNCTION create_user();
```

Forms

We make use of forms in `app.py` to retrieve user entries on the web portal to update the databases. Following show their use corresponding to updating various tables in the database. Note that the form aren't used to directly update other tables of volunteer or participants.

Administrator

Entries for the signup and login page for the admin:

```
username = request.form['name_admin']
email = request.form['email_admin']
password = request.form['password_admin']
```

Organiser

Entries for the signup (or adding one by the administrator) and login page for the organiser:

```
name = request.form['name_org'] # only in case of sign in
username = request.form['id_org']   # only in case of loggin in
email = request.form['email_org']
password = request.form['password_org']
```

Student

Entries for the signup (or adding one by the administrator) and login page for the student user:

```
name = request.form['name_std']  # only in case of sign in
username = request.form['id_std'] # only in case of login
roll = request.form['roll_std']  # only in case of sign in
department = request.form['dept_std'] # only in case of sign in
email = request.form['email_std']
hall = request.form['hall_std']  # only in case of sign in
password = request.form['password_std']
```

External Student

Entries for the signup (or adding one by the administrator) and login page for the external student:

```
username = request.form['id_ext']  # only in case of log in
name = request.form['name_ext'] # only in case of sign in
email = request.form['email_ext']
college_name = request.form['college_name_ext'] # only in case of sign in
password = request.form['password_ext']
```

The external may also update their profile for which following is used to take updated their attributes

```
external = External.query.get_or_404(ext_id)
external.name_ = request.form.get('name_external')
```

```
external.email = request.form.get('email')
external.college_name = request.form.get('college_name')
old_password = request.form.get('old_password') ## to authenticate the changes in the profile
external.passwd = request.form.get('password')  # new password
```

Queries

Flask-SQLAlchemy provides a `query` attribute on the `Model` class. On accessing it, we get back a new query object over all records.

We have made use of methods like `filter()` to filter the records before firing the select with `all()` or `first()`. Wherever needed, as shown below, we have also used `get()` method especially `get_or_404()` using the primary key.

Queries for logging and signing up:

- We perform the following queries when the database **administrator, organiser, student or external participant logs** into the portal with their `username` and `passwd`

```
user = Admin.query.filter_by(id=username).first()
passwd = Admin.query.filter_by(passwd=password).first()

#or the following like in the case of student login
user = User.query.filter_by(username=username,passwd = password).first()
```

- We perform the following queries when the **organiser, student , external participant, signs** into the portal with their `username` and we check if it matches any of the entered usernames in the user database.

```
User.query.filter_by(id=userid).first()
```

  in case that it does the portal reports an error to the user asking them to choose a different username.

Queries for listing entries and adding/deleting

1. Listing:

- For listing all students, external students, organisers, participants, student volunteers or events in the database:

```
students = Student.query.all()

externals = External.query.all()

organisers = Organiser.query.all()

participants = Participant.query.all()

volunteers = Volunteer.query.all()
```

```
events = Event.query.all()
```

2. Adding/Deleting by the database administrator:

- For querying before deleting participants, volunteers in the database:

```
participant = Participant.query.get_or_404(part_id)

volunteer = Volunteer.query.get_or_404(vol_id)
```

- For querying before deleting an event which involves deleting all the enrolled participants and volunteers corresponding to the event queries for which are given just above:

```
event = Event.query.get_or_404(event_id)
```

- For querying before deleting a user which involves deleting all the corresponding roles from other tables like student, participant, organiser etc.

```
User.query.filter_by(id=user_id).first()

# filtering by role attribute
User.query.filter_by(role_='student').first()

User.query.filter_by(role_='external').first()

User.query.filter_by(role_='organiser').first()
```

- For querying before adding students, organisers in the database:

```
User.query.filter_by(id=userid).first()
```

- For querying before registering external students, organisers in the database:

```
id = Participant.query.filter_by(id=ext_id , event_id = user_id).all()
```

- In case of editing external student profile too:

```
external = External.query.get_or_404(ext_name)
```