LOADED_SOULS DBMS MINI PROJECT REPORT

Group Members:

Vishwajeet Singh Solanki (21CS10079)

Priyanshu Kumar (21CS10053)

Paramananda Bhaskar (21CS30035)

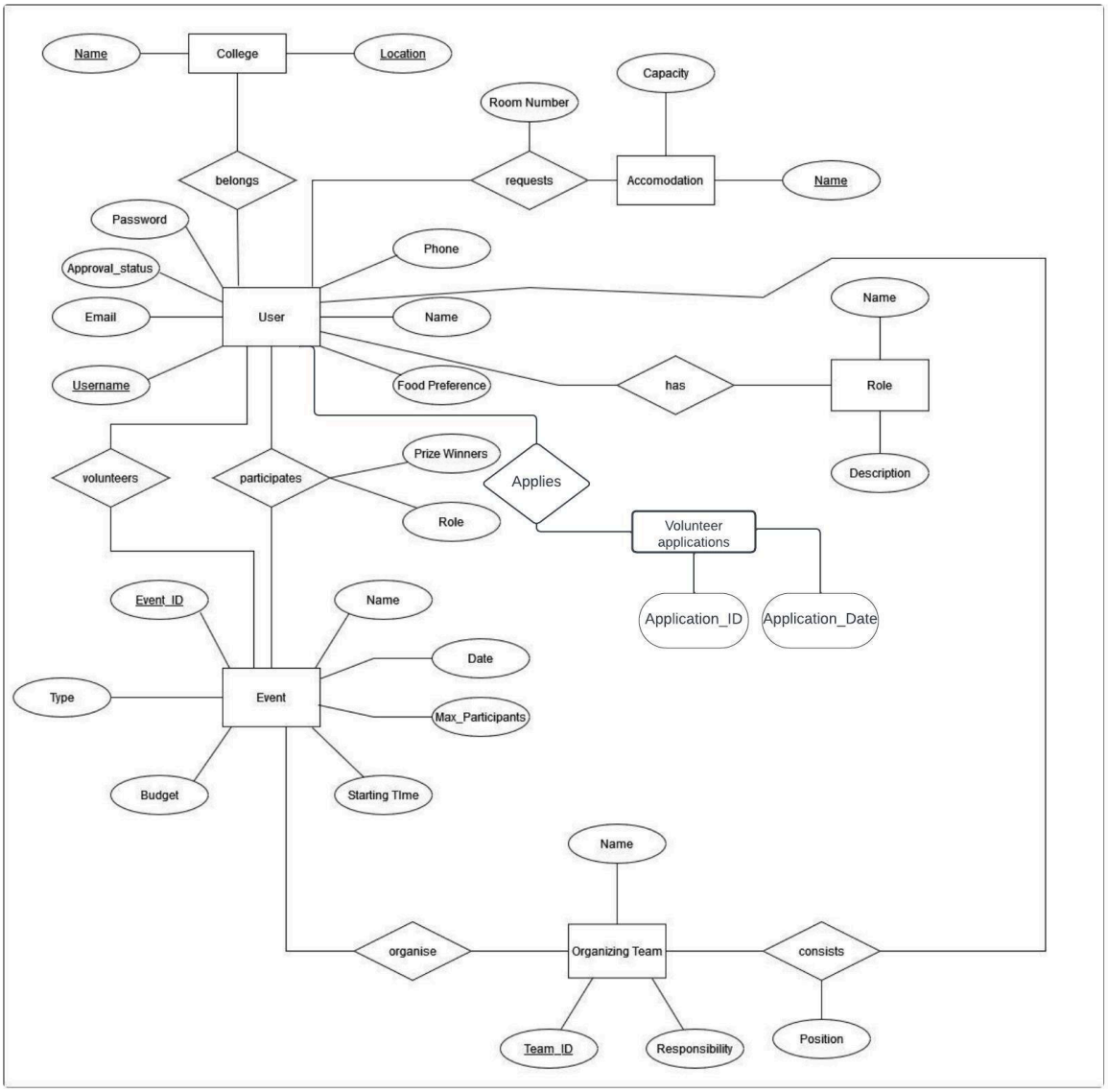
Akshat Ayush Modi (21CS30003)

Aditya Patankar (21CS10051)

This report covers:

- ER diagram
- Actual schema used
- Views, Triggers, Procedures, Roles used
- Forms used

1. ER DIAGRAM



Above is the ER diagram which we used to design the schema.
Explanation of some not-so-obvious attributes:

Approval_status in user entity: In our design, a student can apply to become a volunteer. This application has to be approved by the admin. The attribute Approval_status represents the status of this application and takes values in ('Applied' ,'Approved' ,'NA').

Role in user event relation: This attribute carries the information about the participant's role in the event. For example, a talk can have the roles as speaker and attendee, a hackathon can have roles as evaluator and participant, etc.

Position in organizing_team user relation: This attribute specifies what position the user has in the team like secretary, manager, etc.

Prize_Winners in user event relation: This attribute is used to store the information of the prize winners. The attribute takes integer values in the range -1 to 3. Different values have following meanings:

-1 : Event didn't have prizes
0 : Event had prizes but user didn't win any
1 : Event had prizes and user won first prize
2 : Event had prizes and user won second prize
3 : Event had prizes and user won third prize

Food_Preference in user event relation: This is a Boolean attribute specifying food choice - 0 for veg, 1 for non-veg

External participants and students are covered under the same entity **user**. The user roles used are:

1. Ext_participant
2. Student
3. Organiser
4. Admin

2. ACTUAL SCHEMA USED

Following are the SQL table creation commands used:

```
CREATE SCHEMA IF NOT EXISTS UnivFest;

   CREATE TABLE IF NOT EXISTS UnivFest.role (
       Name varchar(50) PRIMARY KEY,
       Description varchar(100)
   );

   CREATE TABLE IF NOT EXISTS UnivFest.college (
   Name varchar(50),
   Location varchar(100),
   PRIMARY KEY (Name, Location)
   );

   CREATE TABLE IF NOT EXISTS UnivFest."user" (
   Username varchar(50) PRIMARY KEY,
   Passcode varchar(50) NOT NULL,
   Appl_status varchar(20) NOT NULL DEFAULT 'NA' CHECK (Appl_status in
('Approved','NA','Applied')),
   Name varchar(50),
   Email varchar(50) NOT NULL UNIQUE,
   Phone varchar(15) NOT NULL UNIQUE,
   Food_preference boolean,
   CName varchar(50),
   CLocation varchar(50),
   RName varchar(50),
   FOREIGN KEY (CName, CLocation) REFERENCES UnivFest.college ON DELETE
CASCADE,
   FOREIGN KEY (RName) REFERENCES UnivFest.role ON DELETE CASCADE
   );

   CREATE TABLE IF NOT EXISTS UnivFest."event" (
       Event_ID SERIAL PRIMARY KEY,
       Name VARCHAR(20) NOT NULL,
       Date DATE,
       Starting_time TIME,
       Max_participants INT NOT NULL,
       Budget INT,
       Type VARCHAR(20),
```

```
        CHECK (Max_participants > 0 AND Budget > 0)
    );

    CREATE TABLE IF NOT EXISTS UnivFest.organizing_team (
        Team_ID SERIAL PRIMARY KEY,
        Name VARCHAR(20) NOT NULL,
        Responsibility VARCHAR(100)
    );

    create table IF NOT EXISTS UnivFest.organizing_team_consists_users (
        Username    varchar(50),
        Team_ID    INT,
        Position  varchar(20) not null,
        foreign key (Username) references UnivFest.user on delete
cascade,
        foreign key (Team_ID) references UnivFest.organizing_team on
delete cascade,
        UNIQUE (Username, Team_ID)
    );


    create table IF NOT EXISTS UnivFest.organizing_team_organizes_event
(
        Team_ID    INT,
        Event_ID   INT,
        foreign key (Event_ID) references UnivFest.event on delete
cascade,
        foreign key (Team_ID) references UnivFest.organizing_team on
delete cascade
    );

    CREATE TABLE IF NOT EXISTS UnivFest.user_participatesin_event (
    Username varchar(50),
    Event_ID INT,
    Role varchar (20) not null,
    Prize_winners int,
    foreign key (Username) references UnivFest.user on delete cascade,
    foreign key (Event_ID) references UnivFest.event on delete cascade,
    check(Prize_winners <= 3 and Prize_winners >= -1)
    );

    CREATE TABLE IF NOT EXISTS UnivFest.volunteer_applications (
        Application_ID serial PRIMARY KEY,
```

```
       Username varchar(50) NOT NULL,
       Application_Date date DEFAULT CURRENT_DATE,
       FOREIGN KEY (Username) REFERENCES UnivFest.user(Username) ON
DELETE CASCADE
   );

   create table IF NOT EXISTS UnivFest.user_volunteersfor_event (
   Username    varchar(50),
   Team_ID    INT,
   foreign key (Username) references UnivFest.user on delete cascade,
   foreign key (Team_ID) references UnivFest.organizing_team on delete
cascade
   );

   create table IF NOT EXISTS UnivFest.accomodation (
   Name   varchar(50),
   Capacity    int,
   primary key (Name),
   check(capacity > 0)
   );
   create table IF NOT EXISTS UnivFest.user_requests_accomodation (
   Username    varchar(50),
   Name   varchar(50),
   Room_Number int,
   foreign key (Username) references UnivFest.user on delete cascade,
   foreign key (Name) references UnivFest.accomodation on delete
cascade
   );
```

UnivFest is the name of the schema used. It is the schema in the database that has all the tables.

The user to role and user to college relation are many to one. So, rather than create a relation table, we have introduced the role name and college name and location attributes in the user table itself to avoid duplication of information. This is the only change from the ER diagram. Same thing is done for user to volunteer applications relation.

The check and unique constraints are in place to ensure certain integrity constraints.

The on delete cascade option is used to specify that rows referencing a certain foreign key value in the referencing table will be deleted if the tuple with that value is deleted from the referenced table.

3. VIEWS, TRIGGERS, PROCEDURES, ROLES USED

Roles:

We have implemented database access control using roles and grant access commands. A user is created in the database for each of the four roles namely ext_participant, student, organizer and admin.
The users are then granted selected permissions only on some tables.

```
create user ext_participant with password 'ext_part123';
create user student with password 'student123';
create user organizer with password 'organizer123';
create user admin with password 'admin123';

grant usage on schema UnivFest to organiser;
```

Triggers:

Triggers are used in three places:

1. To check whether an event has vacant seats : When a user registers for it. If there are no vacant seats, an exception 'No vacant seats ' is thrown and the user isn't registered.
2. To check whether a certain accommodation has free rooms : If a user tries to book a room in an already full accommodation instance, this trigger is called which blocks any update to the table and throws an exception 'No vacant rooms'.
3. When the admin approves a student's application to be an organizer, a trigger automatically changes the role in the user table and sets the Appl_status attribute to 'NA'.

```
create trigger EventAfterRegistration before insert or update

on UnivFest.user_participatesin_event
for each row
execute procedure checkVacantSeats();
```

Functions:

Functions are used in the triggers as mentioned above. The function is the one that actually checks constraints and blocks or makes the update.

```
create or replace function checkVacantSeats()
returns trigger as
$$
begin
if((select UnivFest.Event.Max_participants from UnivFest.Event where
UnivFest.Event.Event_ID = New.Event_ID) = (select count(Username) from
UnivFest.user_participatesin_event where
UnivFest.user_participatesin_event.EVent_ID = New.Event_ID))
then
  raise exception 'No vacant seats';
end if;
return NEW;
end;
$$
language PLPGSQL;
```

Views:

We have created a view for the **event** table that excludes the budget attribute. The ext_participant and student roles are granted access only on this view and not on the actual **event** table.

```
create view UnivFest.Event_view as
select Event_ID, Name, Date, Starting_time, Type,Max_participants
from UnivFest.Event;
```

4. QUERIES USED

--Insert the user into the database

```
INSERT INTO UnivFest.user (Username, Passcode, Name, Email, Phone,
CName, CLocation, RName) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
```

--Retrieve all columns of user table for the particular user

```
SELECT *
FROM UnivFest.user
WHERE Username = %s AND Passcode = %s
```

--Check application status (application to be a volunteer) for a particular user

```
SELECT Appl_status FROM UnivFest.user WHERE Username = %s
```

-- Check if the role exists, if not, insert it

```
SELECT 1 FROM UnivFest.role WHERE Name = %s
```

-- Insert the role if it doesn't exist

```
INSERT INTO UnivFest.role (Name, Description) VALUES (%s, %s)
```

-- Check if the college exists, if not, insert it

```
SELECT 1 FROM UnivFest.college WHERE Name = %s AND Location = %s
```

-- Insert the college if it doesn't exist

```
INSERT INTO UnivFest.college (Name, Location) VALUES (%s, %s)
```

-- Check if the admin user already exists

```
SELECT 1 FROM UnivFest.user WHERE Username = %s
```

-- Insert the admin user if it doesn't exist

```
INSERT INTO UnivFest.user (Username, Passcode, Name, Email, Phone, CName, CLocation, RName)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
```

-- Check if passwords match

```
SELECT 1 FROM UnivFest.user WHERE Username = %s
```

```
SELECT 1 FROM UnivFest.user WHERE Email = %s
```

```
SELECT 1 FROM UnivFest.user WHERE Phone = %s
```

-- Check if the role exists, if not, insert it

```
SELECT 1 FROM UnivFest.role WHERE Name = %s
```

-- Insert the role if it doesn't exist

```
INSERT INTO UnivFest.role (Name, Description) VALUES (%s, %s)
```

-- Check if the college exists, if not, insert it

```
SELECT 1 FROM UnivFest.college WHERE Name = %s AND Location = %s
```

-- Insert the college if it doesn't exist

```
INSERT INTO UnivFest.college (Name, Location) VALUES (%s, %s)
```

-- If all checks pass, proceed with registration

```
INSERT INTO UnivFest.user (Username, Passcode, Name, Email, Phone, CName, CLocation, RName)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
```

-- Query the user

```
SELECT * FROM UnivFest.user WHERE Username = %s AND Passcode = %s
```

-- Update the application status

```
UPDATE UnivFest.user SET Appl_status = 'Applied' WHERE Username = %s
```

-- Insert a new row into the volunteer applications table

```
INSERT INTO UnivFest.volunteer_applications (Username) VALUES (%s)
```

-- Update the application status

```
UPDATE UnivFest.user SET Appl_status = 'Approved' WHERE Username = %s
```

-- Fetch team data

```
SELECT ot.Team_id, ot.Name, ot.Responsibility, array_agg(concat(e.Event_ID, ' - ', e.Name)) AS
Event_Details

FROM UnivFest.organizing_team ot

LEFT JOIN UnivFest.organizing_team_organizes_event otoe ON ot.Team_ID = otoe.Team_ID

LEFT JOIN UnivFest.event e ON otoe.Event_ID = e.Event_ID

GROUP BY ot.Team_id, ot.Name, ot.Responsibility
```

-- Fetch team member data

```
SELECT ot.Team_ID, ot.Username, ot.Position
```

```
FROM UnivFest.organizing_team_consists_users AS ot


-- Insert new team member
INSERT INTO UnivFest.organizing_team_consists_users (Username, Team_ID, Position) VALUES (%s, %s, %s)
-- Fetch organizers not in the team
SELECT u.Username, u.Email
FROM UnivFest.user AS u
JOIN UnivFest.role AS r ON u.RName = r.Name
LEFT JOIN UnivFest.organizing_team_consists_users AS ot ON u.Username = ot.Username AND ot.Team_ID = %s
WHERE r.Name = 'organizer' AND ot.Username IS NULL
-- Delete team member
DELETE FROM UnivFest.organizing_team_consists_users WHERE Username = %s AND Team_ID = %s
-- Fetch pre-existing information for the organizing team
SELECT Name, Responsibility FROM UnivFest.organizing_team WHERE Team_id = %s
-- Update the organizing team
UPDATE UnivFest.organizing_team SET Name = %s, Responsibility = %s WHERE Team_id = %s
-- Delete the organizing team
DELETE FROM UnivFest.organizing_team WHERE team_id = %s
-- Get participants and their events for a given team_id
SELECT u.Name, u.Email, u.Phone, e.Name AS Event_Name
FROM UnivFest.user u
JOIN UnivFest.user_participatesin_event upe ON u.Username = upe.Username
JOIN UnivFest.event e ON upe.Event_ID = e.Event_ID
JOIN UnivFest.organizing_team_organizes_event otoe ON e.Event_ID = otoe.Event_ID
WHERE otoe.Team_ID = %s
-- Retrieve event details for admin and organizer view
SELECT
    e.Event_ID, e.Name, e.Date, e.Starting_time, e.Max_participants,
    e.Budget,
```

```
    e.Type,
    ot.Team_ID,
    ot.Name AS Organizer_Team_Name,
    ot.Responsibility AS Organizer_Team_Responsibility,
    (e.Max_participants - COUNT(upe.Event_ID)) AS Remaining_spots
FROM
    UnivFest.event e
JOIN
    UnivFest.organizing_team_organizes_event otoe ON e.Event_ID = otoe.Event_ID
JOIN
    UnivFest.organizing_team ot ON otoe.Team_ID = ot.Team_ID
LEFT JOIN
    UnivFest.user_participatesin_event upe ON e.Event_ID = upe.Event_ID
GROUP BY
    e.Event_ID, ot.Team_ID;
-- Insert an event by admin
INSERT INTO UnivFest.event (Name, Date, Starting_time, Max_participants, Budget, Type)
VALUES (%s, %s, %s, %s, %s, %s) RETURNING Event_ID;
INSERT INTO UnivFest.organizing_team_organizes_event (team_id, Event_ID)
VALUES (%s, %s);
-- Update event details by admin or organizer
UPDATE UnivFest.event
SET
    Name = %s,
    Date = %s,
    Starting_time = %s,
    Max_participants = %s,
    Budget = %s,
    Type = %s
WHERE
    Event_ID = %s;
```

```
UPDATE UnivFest.organizing_team_organizes_event
SET
   Team_ID = %s
WHERE
   Event_ID = %s;
-- Delete an event by organizer or admin
DELETE FROM UnivFest.event WHERE Event_ID = %s;
-- Retrieve winner list
SELECT
   e.Name AS event_name,
   u.Username,
   upe.Prize_winners
FROM
   UnivFest.user_participatesin_event upe
JOIN
 UnivFest.user u ON upe.Username = u.Username
JOIN
   UnivFest.event e ON upe.Event_ID = e.Event_ID
WHERE
   upe.Prize_winners >= 0
ORDER BY
   e.Name, upe.Prize_winners ASC;


-- Add a prize winner
UPDATE UnivFest.user_participatesin_event
SET Prize_winners = %s
WHERE
   UnivFest.user_participatesin_event.Username = %s
   AND UnivFest.user_participatesin_event.Event_ID = %s;
```

```
-- Select volunteer applications
SELECT
    va.Application_ID,
    va.Username,
    va.Application_Date,
    u.Name,
    u.Email
FROM
    UnivFest.volunteer_applications AS va
JOIN
    UnivFest.user AS u ON va.Username = u.Username
WHERE
    u.Appl_status = 'Applied'
ORDER BY
    va.Application_Date DESC;

-- Add an accommodation
INSERT INTO UnivFest.accomodation (Name, Capacity)
VALUES (%s, %s);

-- View accommodations
SELECT * FROM UnivFest.accomodation;

-- View volunteer list by team
SELECT
    u.Name,
    u.Email,
    u.Phone
FROM
    UnivFest.user u
```

```
JOIN
   UnivFest.user_volunteersfor_event v ON u.Username = v.Username
WHERE
   v.Team_ID = %s;

-- Request accommodation
INSERT INTO UnivFest.user_requests_accomodation (username, name)
VALUES (%s, %s);

-- Request accommodation
INSERT INTO UnivFest.user_requests_accomodation (username, name)
VALUES (%s, %s);

-- Fetch available accommodations with capacity
SELECT a.name
FROM UnivFest.accomodation a
WHERE a.capacity > (SELECT COUNT(*)
                    FROM UnivFest.user_requests_accomodation ura
                    WHERE ura.name = a.name);

-- Select a random accommodation from available ones
SELECT a.name
FROM UnivFest.accomodation a
WHERE a.capacity > (SELECT COUNT(*)
                    FROM UnivFest.user_requests_accomodation ura
                    WHERE ura.name = a.name)
ORDER BY RANDOM()
LIMIT 1;

-- Assign selected accommodation to the user
```

```
INSERT INTO UnivFest.user_requests_accomodation (username, name)

VALUES (%s, %s);
```

-- Update food preference query

```
UPDATE UnivFest.user SET Food_preference = %s WHERE Username = %s
```

-- Fetch all events query

```
SELECT ev.Event_ID, ev.Name, ev.Date, ev.Starting_time, ev.Max_participants, ev.Type,
(ev.Max_participants - COALESCE(reg.registered_count, 0)) as remaining_spots FROM
UnivFest.Event_view as ev LEFT JOIN ( SELECT Event_ID, COUNT(*) as registered_count FROM
UnivFest.user_participatesin_event GROUP BY Event_ID ) as reg ON ev.Event_ID = reg.Event_ID
```

-- Fetch registered events query

```
SELECT Event_ID FROM UnivFest.user_participatesin_event WHERE Username = %s
```

-- Fetch application status query

```
SELECT Appl_status from UnivFest.user where Username = %s
```

-- Register for event query

```
INSERT INTO UnivFest.user_participatesin_event (Username, Event_ID, Prize_winners) VALUES (%s,
%s, '-1')
```

-- Cancel event registration query

```
DELETE FROM UnivFest.user_participatesin_event WHERE Username = %s AND Event_ID = %s
```

-- Fetch all users query

```
SELECT Username, Name, Email, Phone, CName, CLocation, RName FROM UnivFest.user WHERE
Username != %s
```

-- Fetch organizing teams query

```
SELECT Name FROM UnivFest.organizing_team WHERE Team_ID IN (SELECT Team_ID FROM UnivFest.organizing_team_consists_users WHERE Username = %s)
```

-- Fetch registered events for user query

```
SELECT Name FROM UnivFest.event WHERE Event_ID IN (SELECT Event_ID FROM UnivFest.user_participatesin_event WHERE Username = %s)
```

-- Fetch user profile query

```
SELECT Username, Name, Email, Phone FROM UnivFest.user WHERE Username = %s
```

-- Fetch volunteered events query

```
SELECT e.Name FROM UnivFest.user_volunteersfor_event uvfe JOIN UnivFest.organizing_team_organizes_event otoe ON uvfe.Team_ID = otoe.Team_ID JOIN UnivFest.event e ON otoe.Event_ID = e.Event_ID WHERE uvfe.Username = %s
```

-- Fetch food preference query

```
SELECT Food_preference FROM UnivFest.user WHERE Username = %s
```

-- Fetch accommodation request query

```
SELECT Name FROM UnivFest.user_requests_accomodation WHERE Username = %s
```

-- Fetch user profile for organizer query

```
SELECT Username, Name, Email, Phone FROM UnivFest.user WHERE Username = %s
```

-- Fetch events organized by user query

```
SELECT e.Name FROM UnivFest.organizing_team_organizes_event otoe JOIN UnivFest.event e ON otoe.Event_ID = e.Event_ID WHERE otoe.Team_ID IN (SELECT Team_ID FROM UnivFest.organizing_team_consists_users WHERE Username = %s)
```

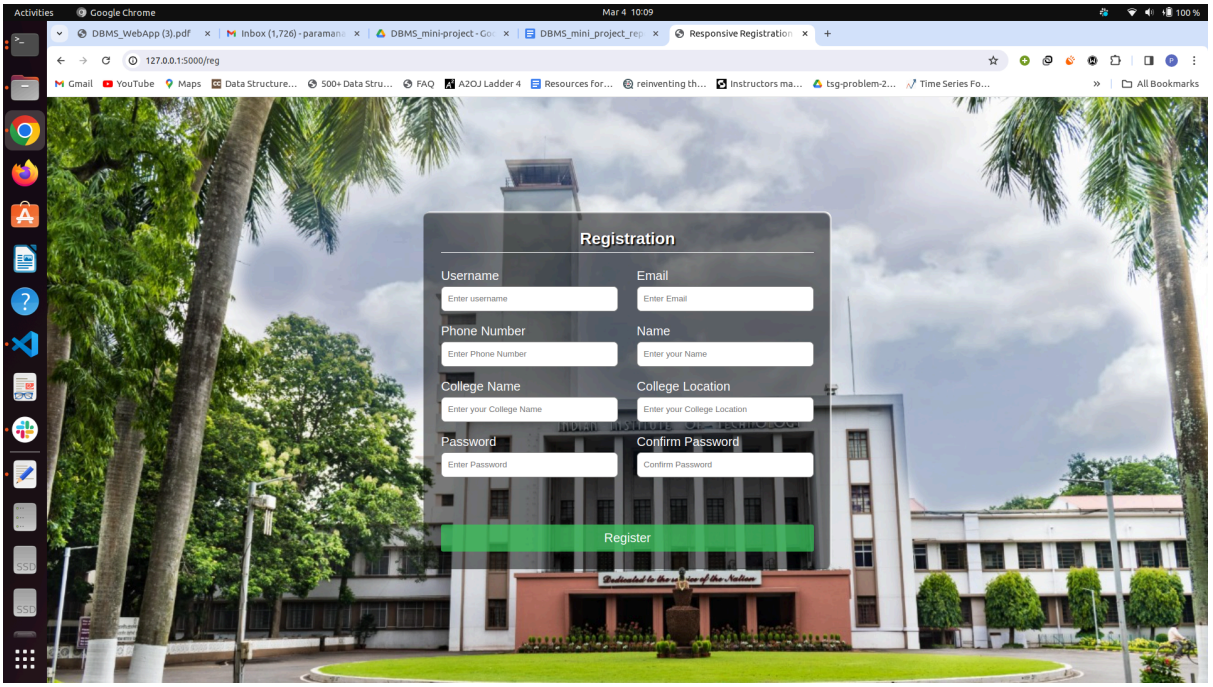
-- Update user profile query

```
UPDATE UnivFest.user SET Name = %s, Email = %s, Phone = %s WHERE Username = %s
```

5.FORMS

1. Registration Form (`/reg` route):

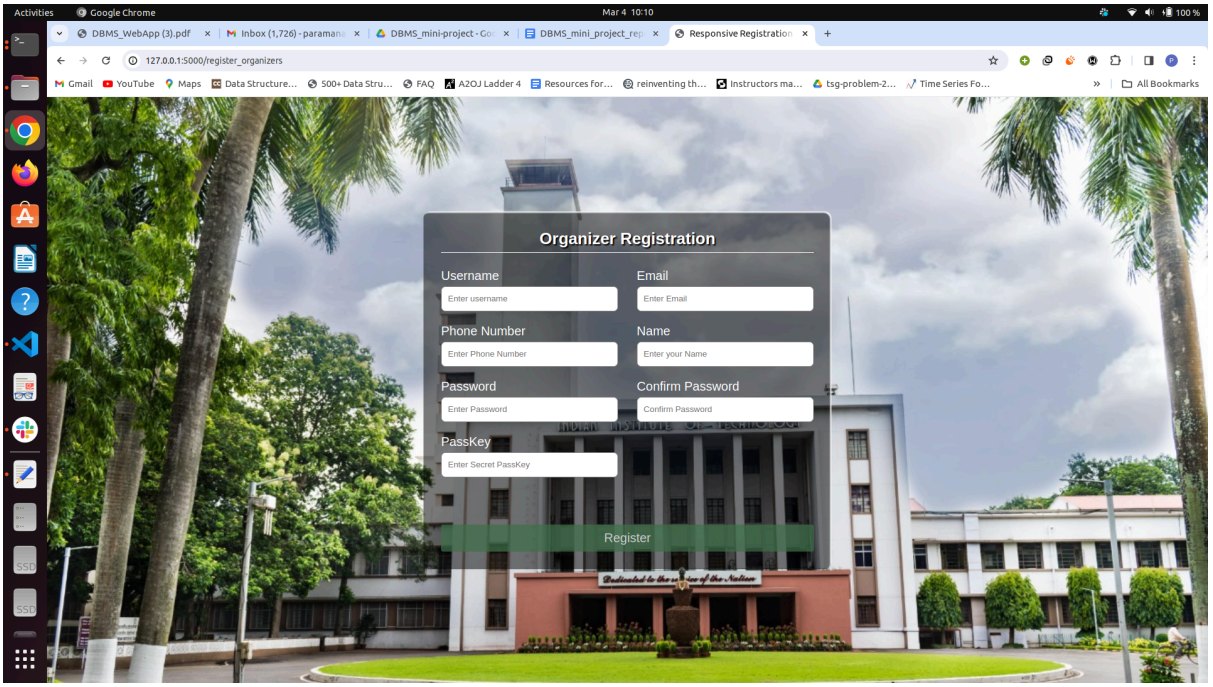
- Description:
  - The registration form allows users to create a new account by providing necessary details such as username, password, email, phone number, name, college name, and college location.
- Validation Rules:
  - Password and confirm password fields must match.
  - Username, email, and phone number must be unique and not already exist in the database.
  - The role (`rname`) is determined based on the email domain. If the email domain contains "kgpian" or "iitkgp", the role is set to "student"; otherwise, it is set to "ext_participant".
- Actions on Submission:
  - Upon successful validation, a new user entry is inserted into the database with the provided details.
  - If any validation checks fail, appropriate error messages are flashed to the user, and they are redirected back to the registration page.
- Error Handling:
  - Errors related to password mismatch, duplicate username/email/phone, or database errors are handled gracefully by flashing error messages to the user.



---

2. Organizer Registration Form (`/register_organizers` route):

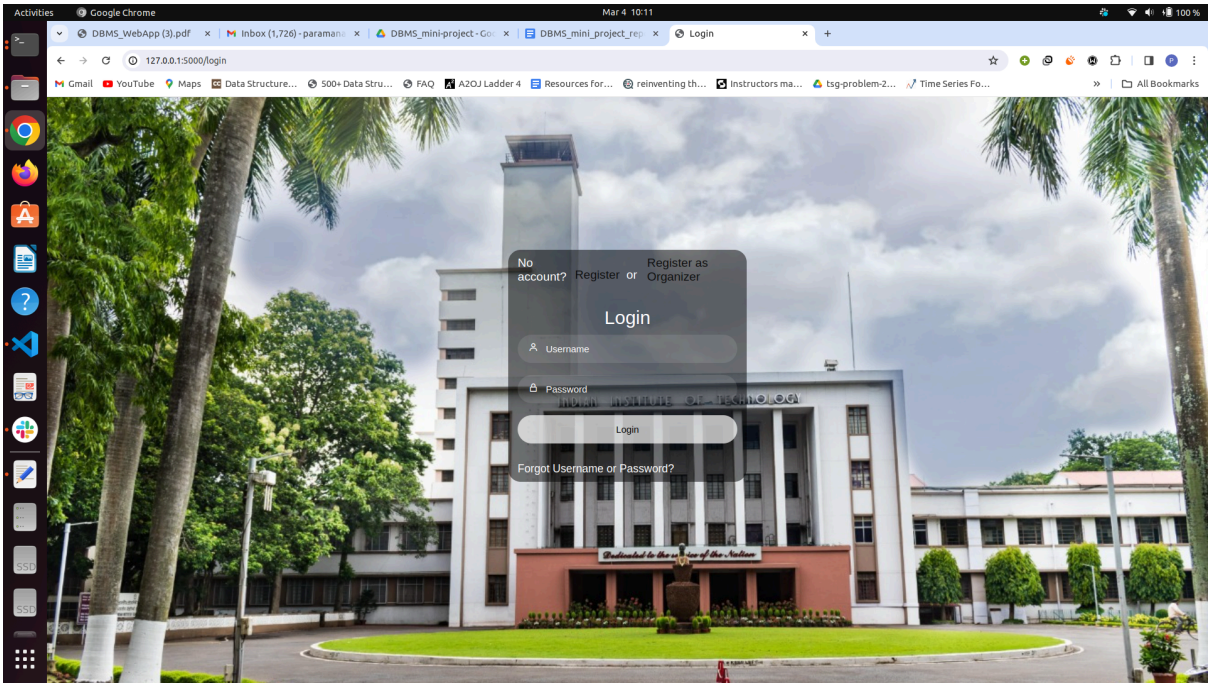
- Description:
  - The organizer registration form is similar to the general registration form but is specifically designed for organizers.
  - It collects details such as username, password, email, phone number, name, and a passkey for organizer registration.
- Validation Rules:
  - Password and confirm password fields must match.
  - Username, email, and phone number must be unique and not already exist in the database.
  - A passkey is required for organizer registration, and it must match a predefined value ("Loaded_Souls").
- Actions on Submission:
  - Upon successful validation and passkey verification, a new organizer entry is inserted into the database with the provided details.
  - If any validation checks fail, appropriate error messages are flashed to the user, and they are redirected back to the organizer registration page.
- Error Handling:
  - Errors related to password mismatch, duplicate username/email/phone, incorrect passkey, or database errors are handled gracefully by flashing error messages to the user.



---

3. Login Form (`/login` route):

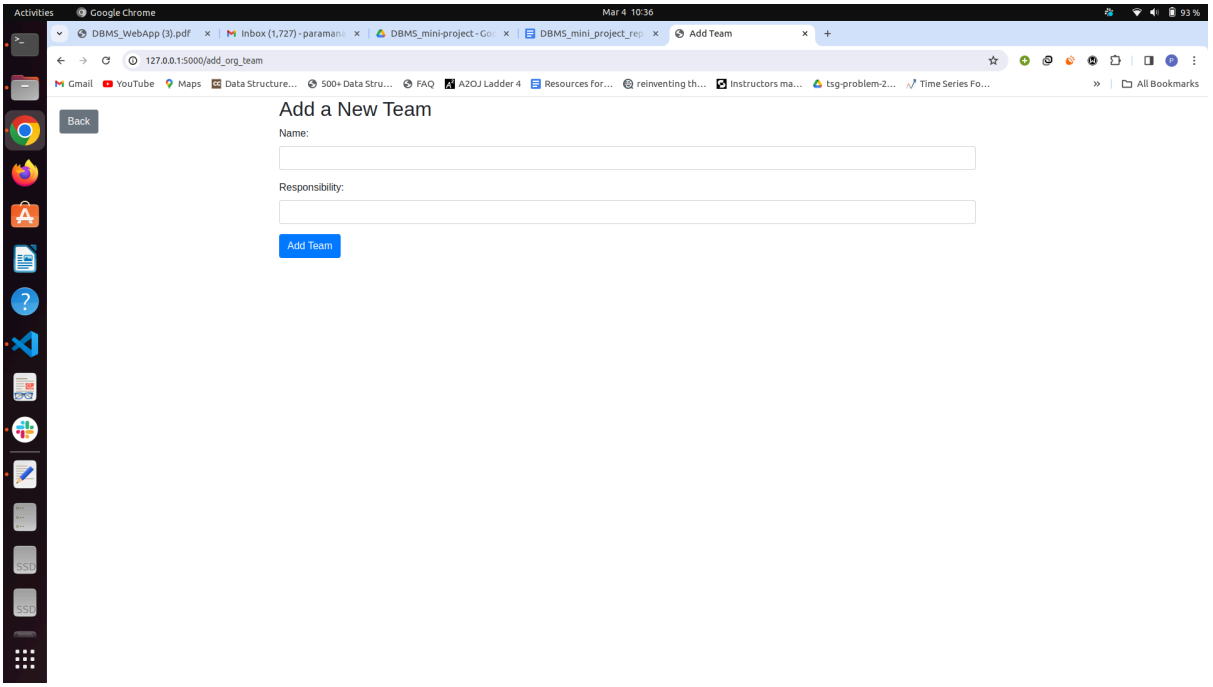
- Description:
  - The login form allows existing users to authenticate themselves by providing their username and password.
- Validation Rules:
  - Username and password must match an existing user entry in the database.
- Actions on Submission:
  - Upon successful authentication, the user is redirected to the appropriate home page based on their role (student, external participant, organizer, admin).
  - If authentication fails, an error message is flashed, and the user remains on the login page.
- Error Handling:
  - Errors related to invalid username/password or database errors are handled by flashing error messages to the user.



---

4. Add Organizing Team Form (`/add_org_team` route):

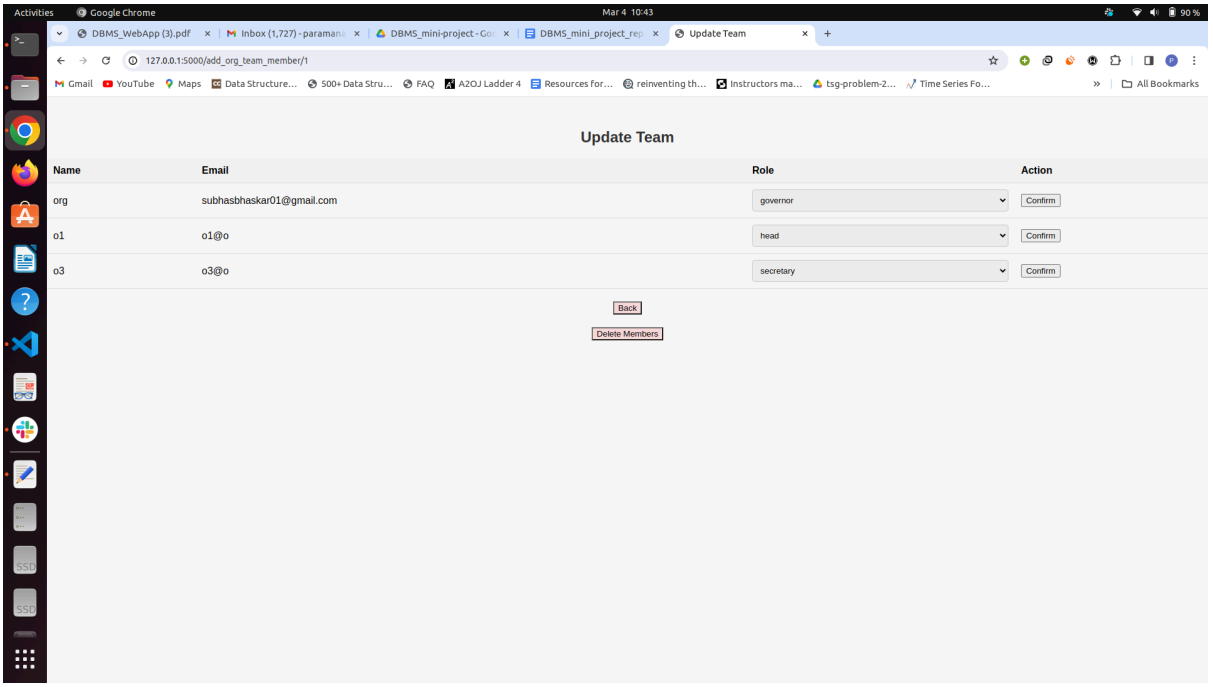
- Description:
  - This form allows an admin user to add a new organizing team by providing the team name and responsibility.
- Validation Rules:
  - Only admin users are allowed to access this form.
  - The form must be submitted via POST method to add a new team to the database.
  - The team name and responsibility fields are required.
- Actions on Submission:
  - Upon successful submission, the provided team details are inserted into the database, and the user(admin here) is redirected back to the add organizing team page.
  - If the form submission method is GET or if the user is not an admin, they are redirected to the login page with an appropriate flash message.

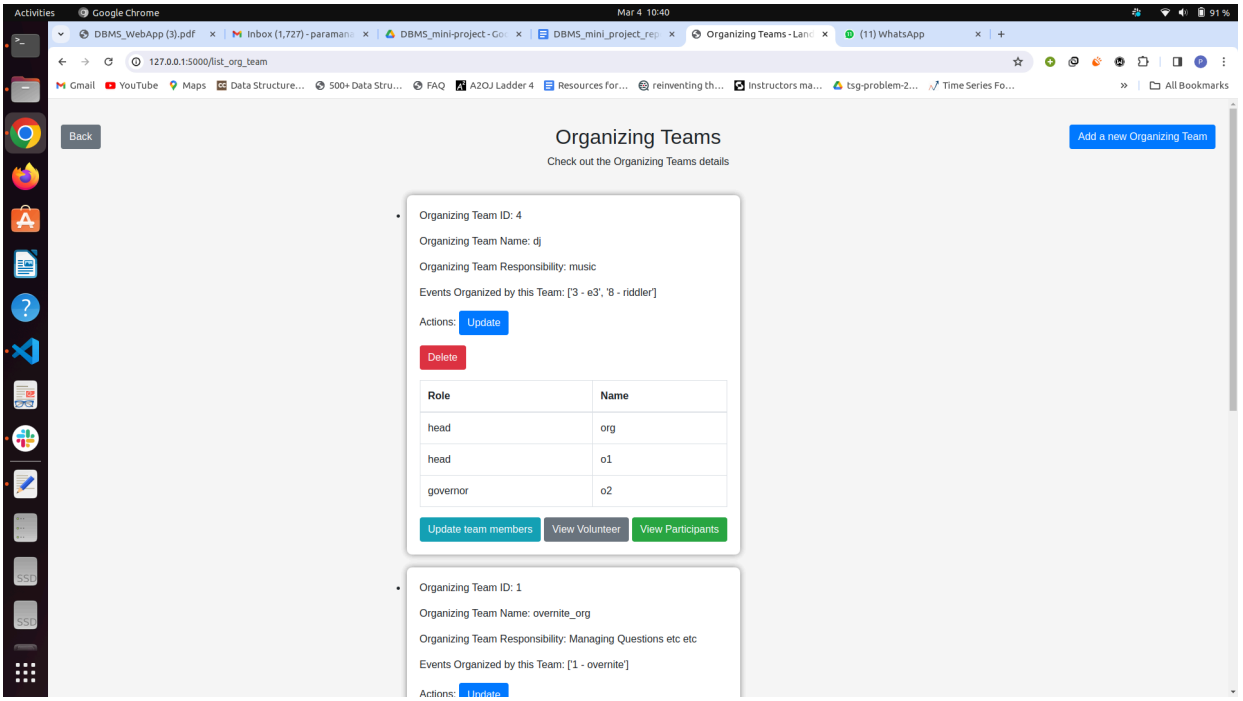


---

5. Add Organizing Team Member Form (`/add_org_team_member/<team_id>` route):

- Description:
  - This form allows an admin user to add members to an existing organizing team identified by the `team_id` parameter.
- Validation Rules:
  - Only admin users are allowed to access this form.
  - The form must be submitted via POST method to add a new member to the team.
  - The username and position fields are required.
- Actions on Submission:
  - Upon successful submission, the provided member details are inserted into the database for the specified team, and the user is redirected back to the same page.
  - If the form submission method is GET or if the user is not an admin, they are redirected to the login page with an appropriate flash message.



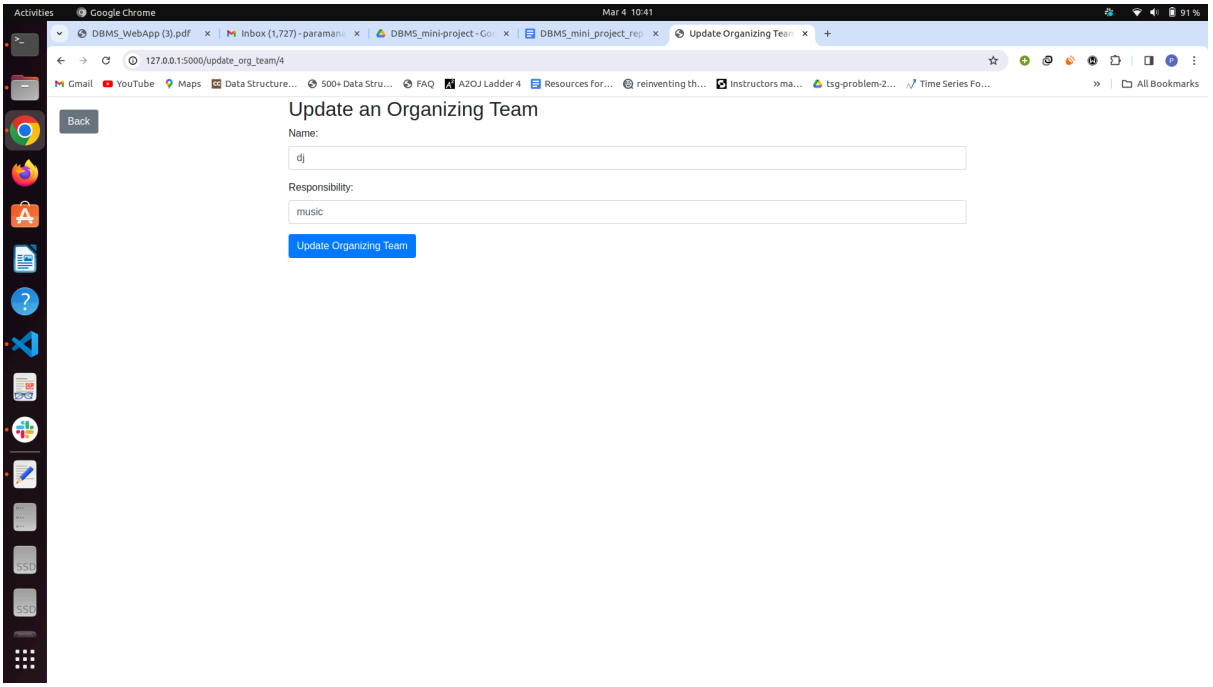


---

6. Update Organizing Team Form (`/update_org_team/<team_id>` route):

- Description:
  - This form allows an admin user to update the details of an existing organizing team identified by the `team_id` parameter.
- Validation Rules:
  - Only admin users are allowed to access this form.

    - The form must be submitted via POST method to update the team details.
    - The team name and responsibility fields are required.
- Actions on Submission:
    - Upon successful submission, the details of the specified team are updated in the database, and the user is redirected to the list of organizing teams.
    - If the form submission method is GET or if the user is not an admin, they are redirected to the login page with an appropriate flash message.

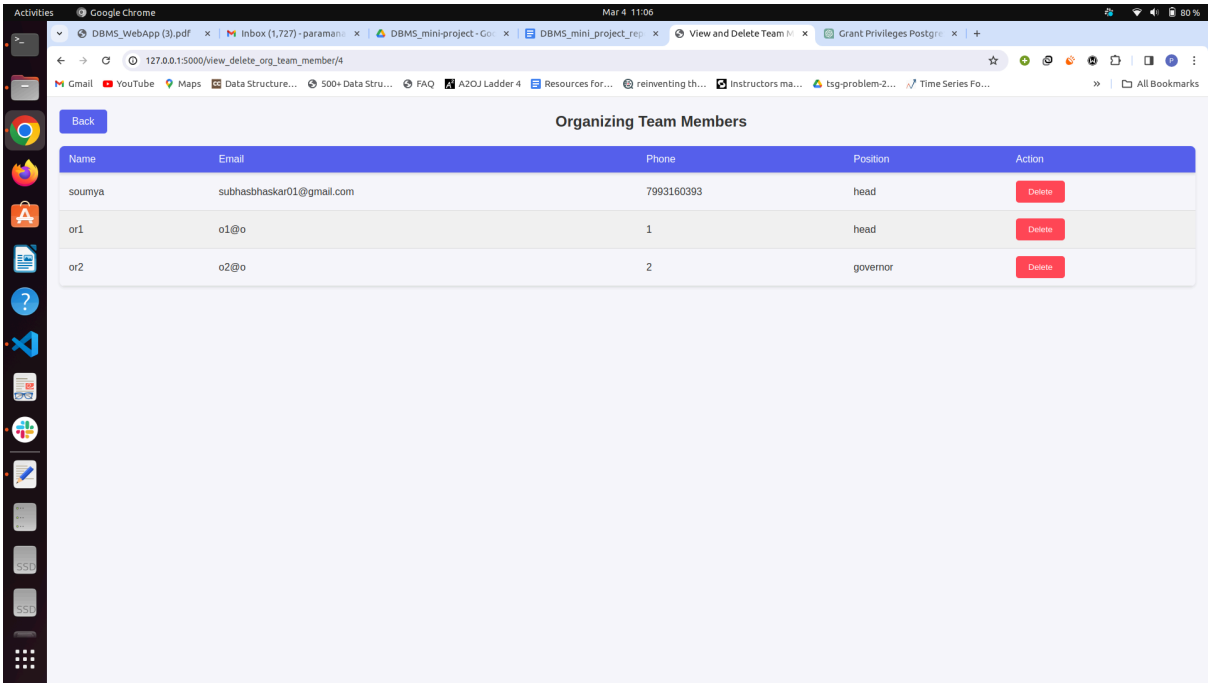


---

7. Delete Team Members Function:

- Description:
    - This function allows an admin to delete members from an organizing team.
- Validation Rules:
    - Only admins can access this function.
    - Users must be logged in to perform deletion.
    - Proper error handling is implemented to handle exceptions during deletion operations.
- Actions:

- View Delete Team Members Route (`/view_delete_org_team_member/<team_id>`):
  - Fetches the members of the specified organizing team from the database.
  - Renders a page to view and delete team members.
- Delete Organizing Team Member Route (`/delete_org_team_member/<int:team_id>/<username>`):
  - Deletes the specified member from the organizing team in the database.
  - Redirects back to the view page for the organizing team members after deletion.

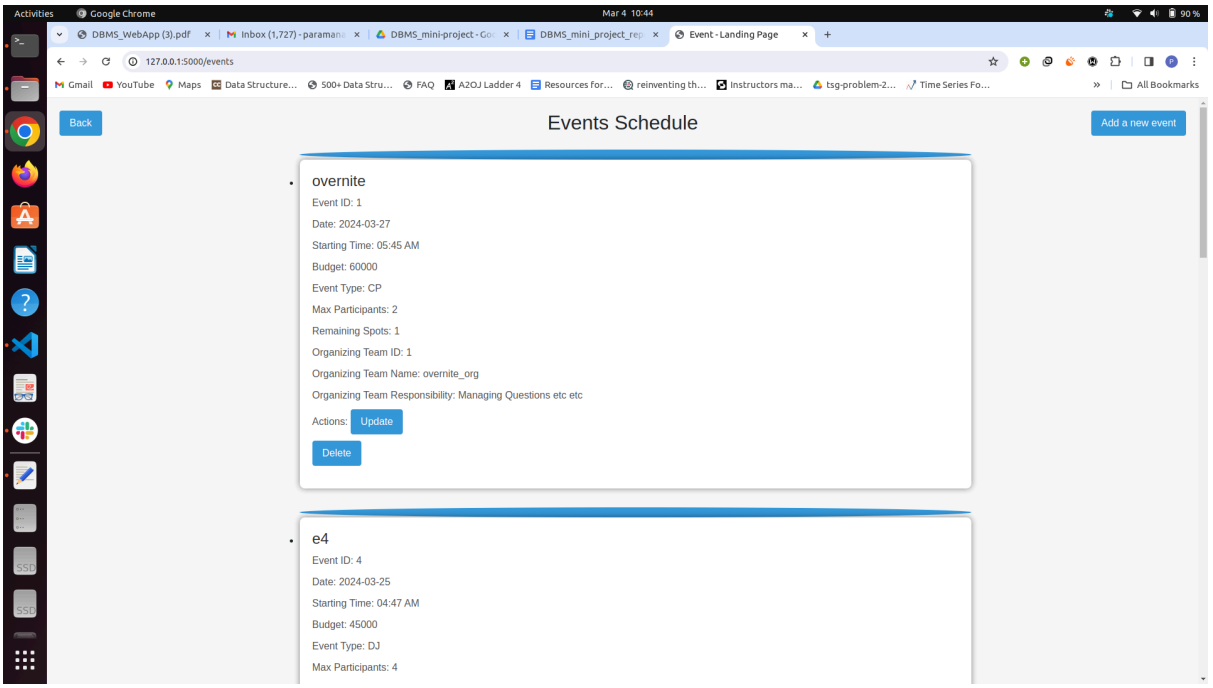


---

8. List Events (`/events` route):

- Description:
  - This route displays a list of events along with their details, including the organizing team's information.
  - Both organizers and admins can access this route to view events.
- Validation Rules:
  - Users must be logged in as an organizer or admin to view the event list.
- Actions:
  - Fetches event data from the database, including event name, date, starting time, maximum participants, budget, event type, organizing team details, and remaining spots.

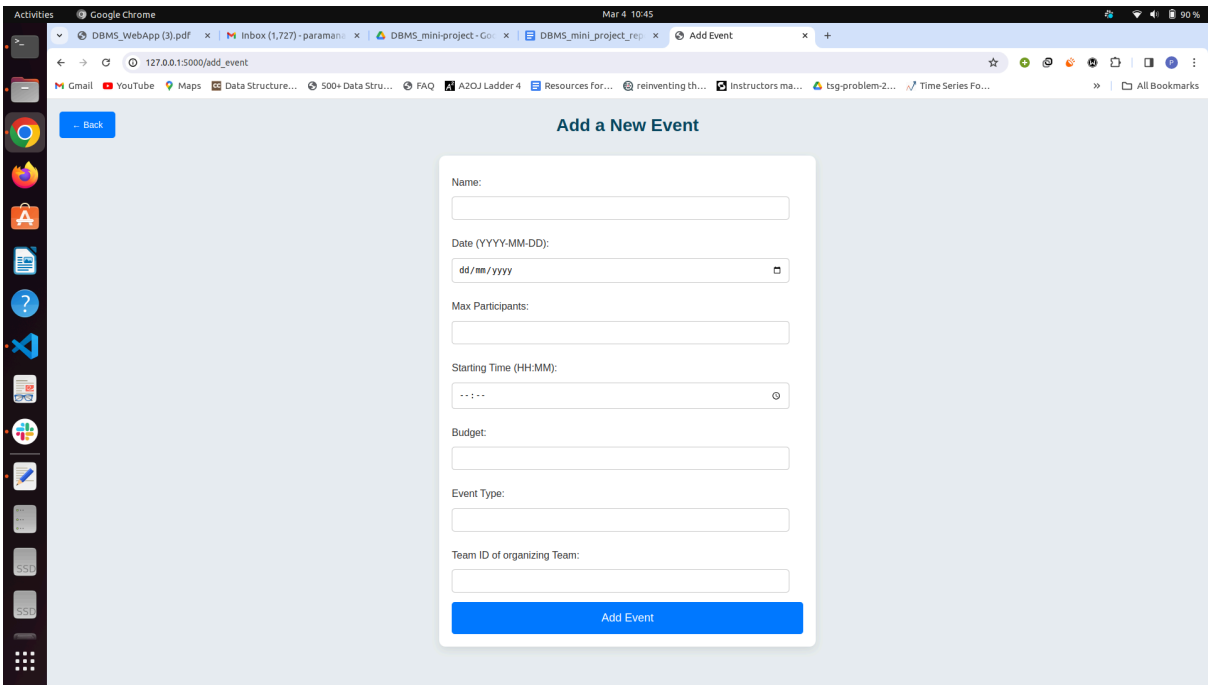
- Renders the `events_admin_organizer.html` template, passing the event data and the current user's role.
- If the user is not logged in as an organizer or admin, they are redirected to the login page with an error flash message.



---

9. Add Event (`/add_event` route):

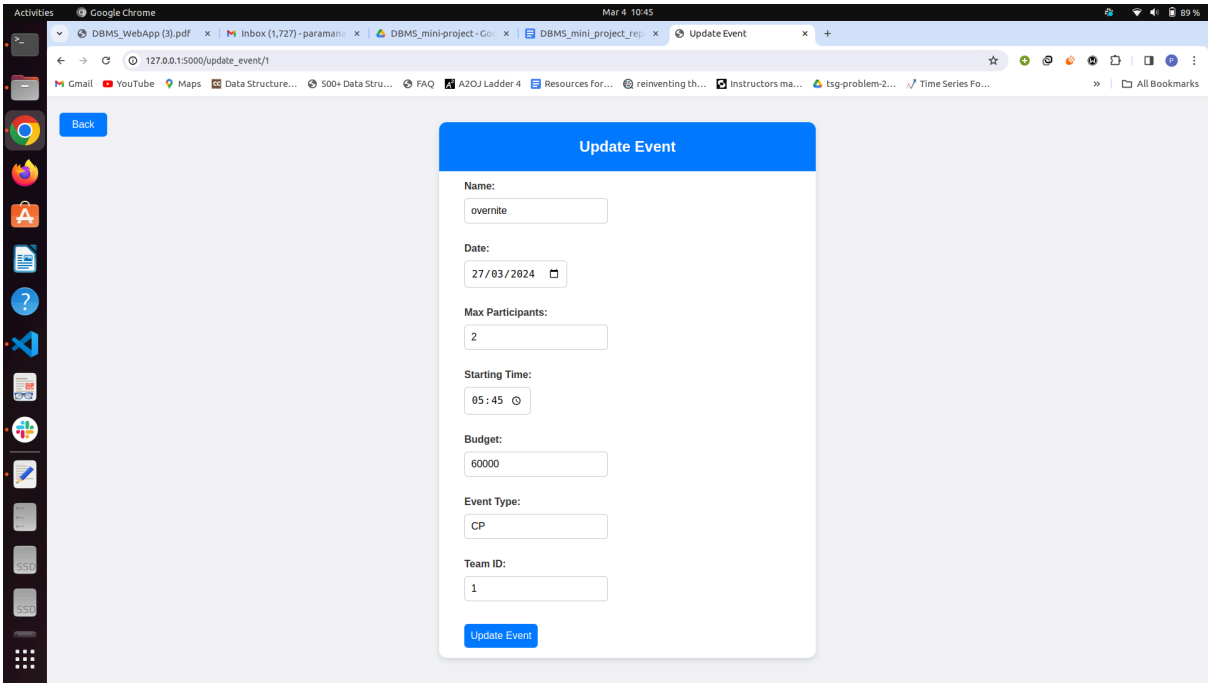
- Description:
  - This route allows admin users to add new events by providing event details such as name, date, starting time, maximum participants, budget, event type, and organizing team ID.
- Validation Rules:
  - Only admin users are allowed to access this route.
  - The form must be submitted via POST method to add a new event.
  - Event name must be unique.
  - The organizing team ID provided must exist in the database.
- Actions:
  - Inserts the event details into the database upon successful form submission.
  - If the form submission method is GET or if the user is not an admin, they are redirected to the login page with an error flash message.



---

10. Update Event (`/update_event/<event_id>` route):

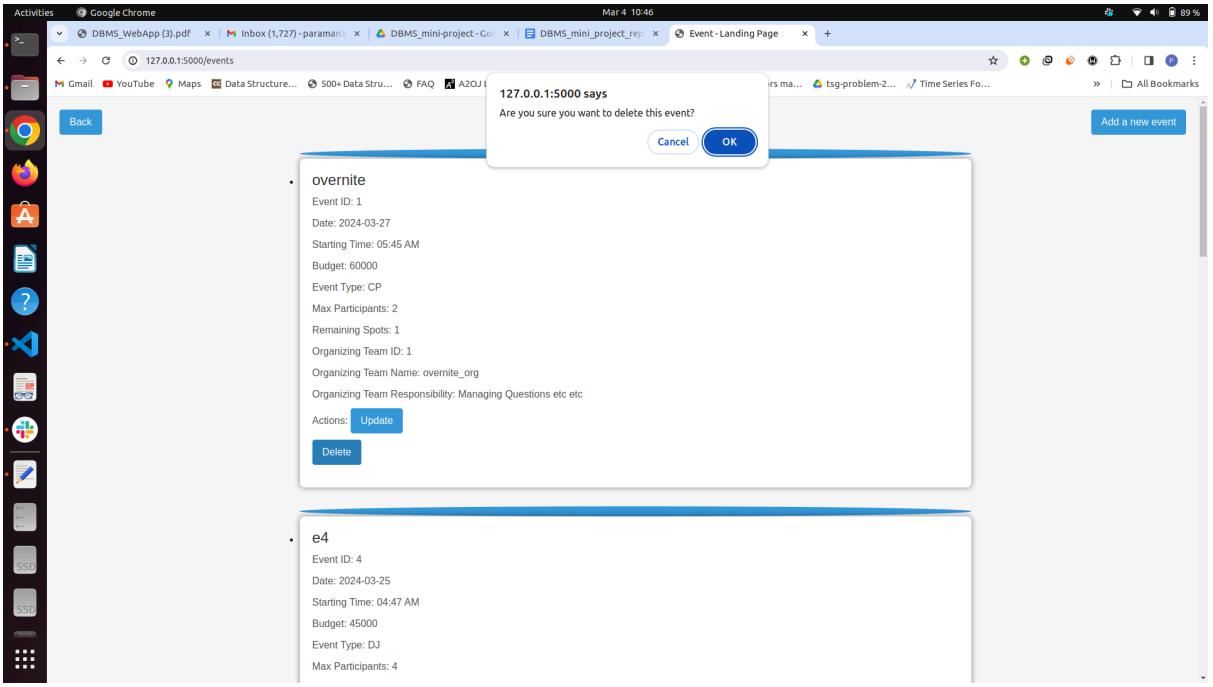
- Description:
  - This route allows organizers and admins to update the details of an existing event identified by the `event_id` parameter.
- Validation Rules:
  - Only organizers and admins are allowed to access this route.
  - The form must be submitted via POST method to update the event details.
  - The provided event ID must exist in the database, and the user must have permission to update the event.
- Actions:
  - Fetches existing event details to pre-fill the update form.
  - Updates the event details in the database upon successful form submission.
  - If the user does not have permission to update the event or if the event is not found, appropriate flash messages are displayed.



---

11. Delete Event (`/delete_event/<event_id>` route):

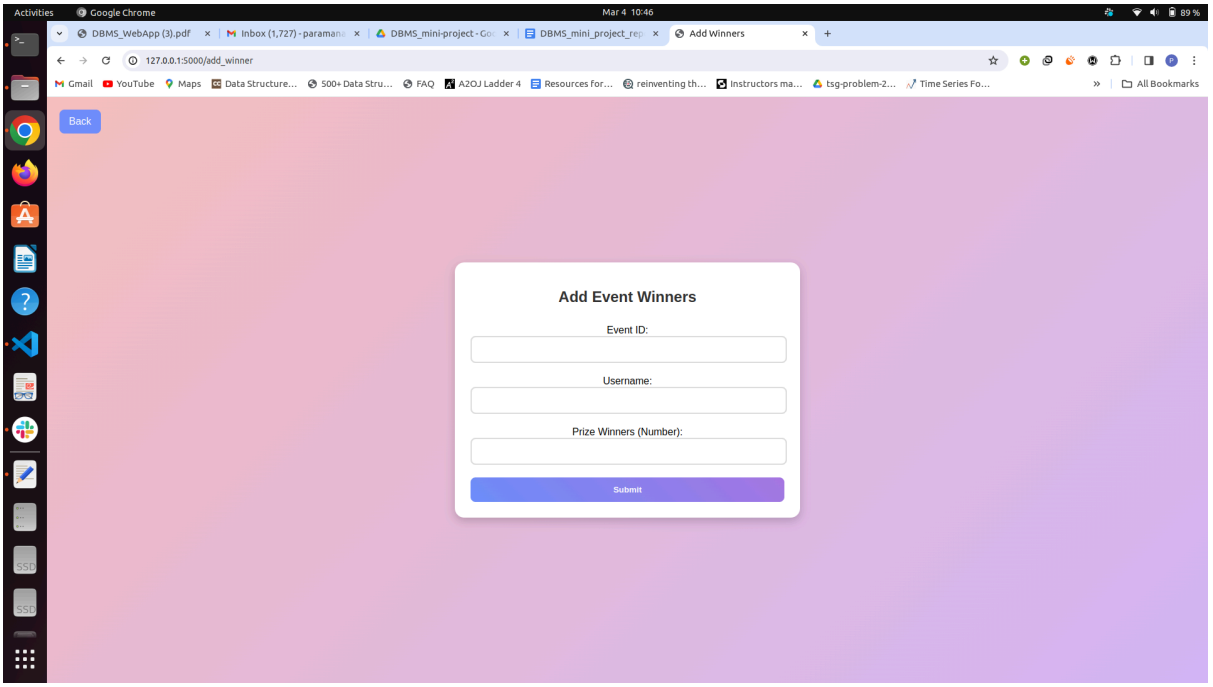
- Description:
  - This route allows organizers and admins to delete an existing event identified by the `event_id` parameter.
- Validation Rules:
  - Only organizers and admins are allowed to access this route.
  - The form must be submitted via POST method to delete the event.
  - The provided event ID must exist in the database, and the user must have permission to delete the event.
- Actions:
  - Deletes the specified event from the database upon successful form submission.
  - If the user does not have permission to delete the event or if an error occurs during deletion, appropriate flash messages are displayed.

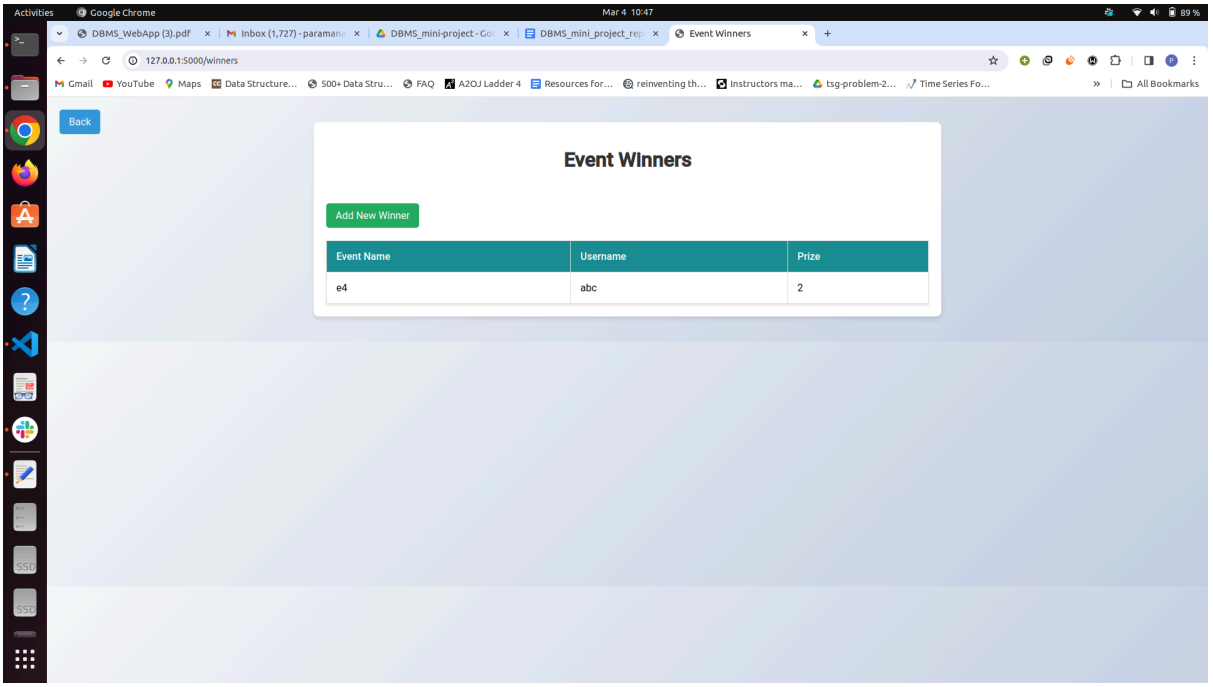


---

12. Add Winner (`/add_winner` route):

- Description:
  - This route allows organizers and admins to add prize winners for events.
  - Users (here organizers and admins) can specify the event ID, username of the winner, and the number of prizes won.
- Validation Rules:
  - Only organizers and admins can access this route.
  - Users (here organizers and admins) must provide valid event ID and username.
  - The event organizer's team must be associated with the event.
  - The user must have participated in the specified event.
- Actions:
  - Validates the user's permission to update the event.
  - Checks if the event ID and username are valid and if the user has participated in the event.
  - Adds the prize winner information to the database.
  - Displays appropriate flash messages for success or failure.



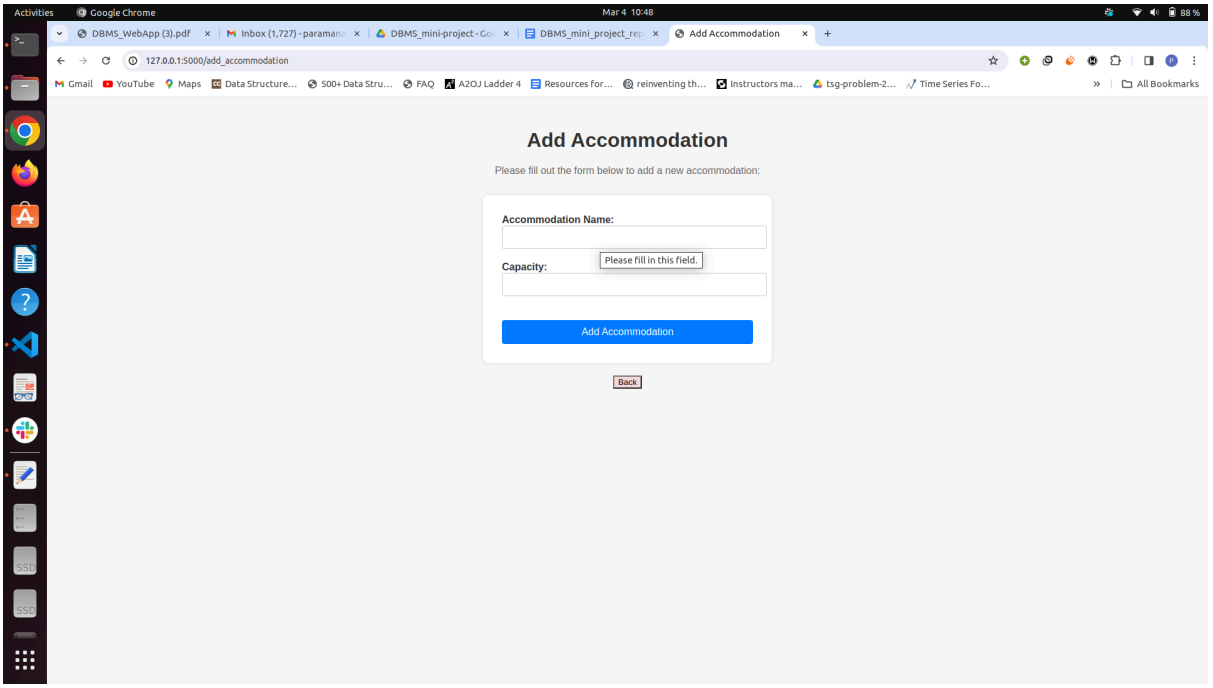


---

13. Add Accommodation (`/add_accommodation` route):

- Description:
  - This route allows admins to add new accommodations along with their capacities.
- Validation Rules:
  - Only admin users can access this route.
  - Accommodation capacity must be a positive integer.

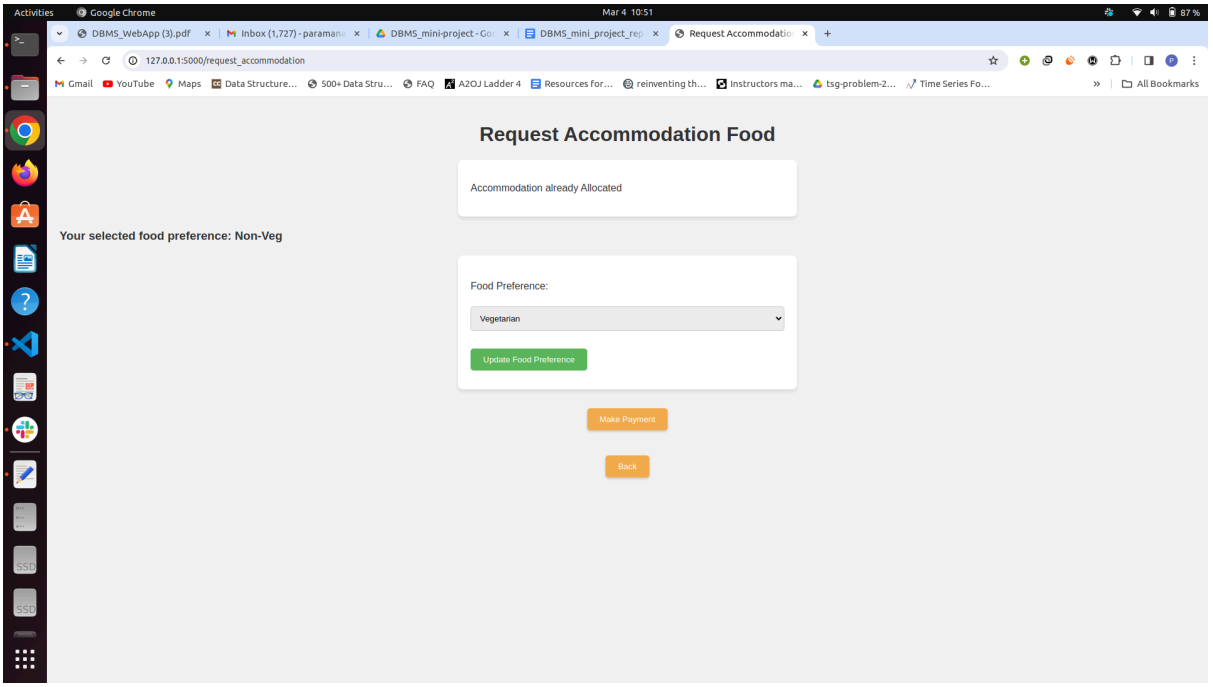
- Actions:
    - Inserts the accommodation details into the database upon successful form submission.
    - Displays appropriate flash messages for success or failure.



---

14. Request Accommodation (`/request_accommodation` route):

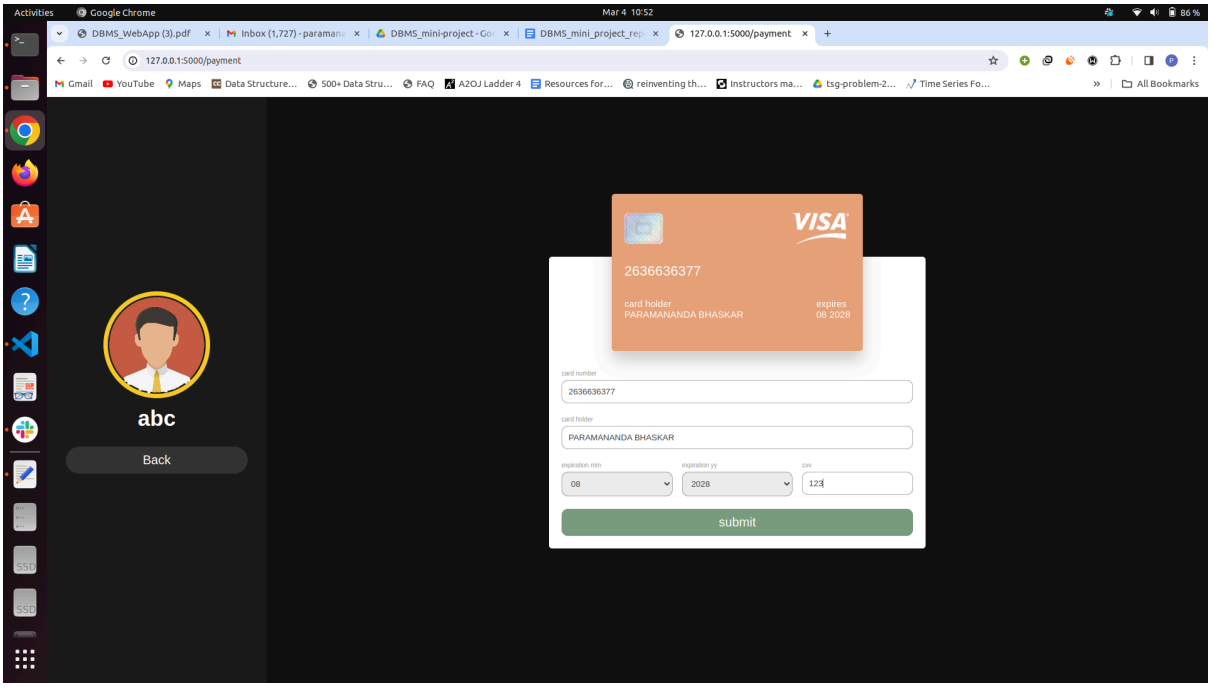
- Description:
    - External participants can use this route to request accommodation.
    - Upon submission, the system assigns an available accommodation to the user.
- Validation Rules:
    - Only external participants can access this route.
    - Users can request accommodation only once.
    - Food preference must be specified before requesting accommodation.
- Actions:
    - Checks if the user already has an assigned accommodation.
    - Finds available accommodations with free capacity.
    - Randomly selects an available accommodation and assigns it to the user.
    - Displays appropriate flash messages for success or failure.



---

15. Update Food Preference (`/update_food_preference` route):

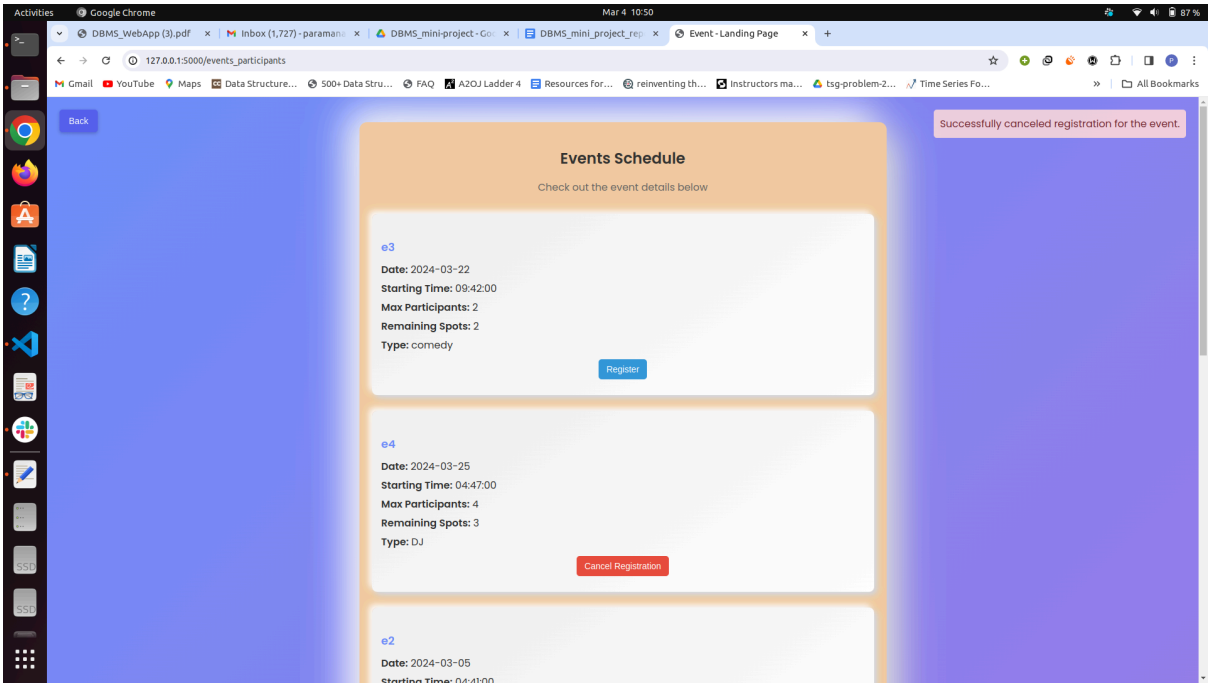
- Description:
    - External participants can update their food preferences (Vegetarian or Non-Vegetarian) using this route.
- Validation Rules:
    - Only external participants can access this route.
    - Food preference must be specified as either Vegetarian or Non-Vegetarian.
- Actions:
    - Updates the user's food preference in the database.
    - Redirects the user to the accommodation page after updating the food preference.
    - Displays appropriate flash messages for success or failure.



---

16. Event Page for Participants (`/events_participants` route):

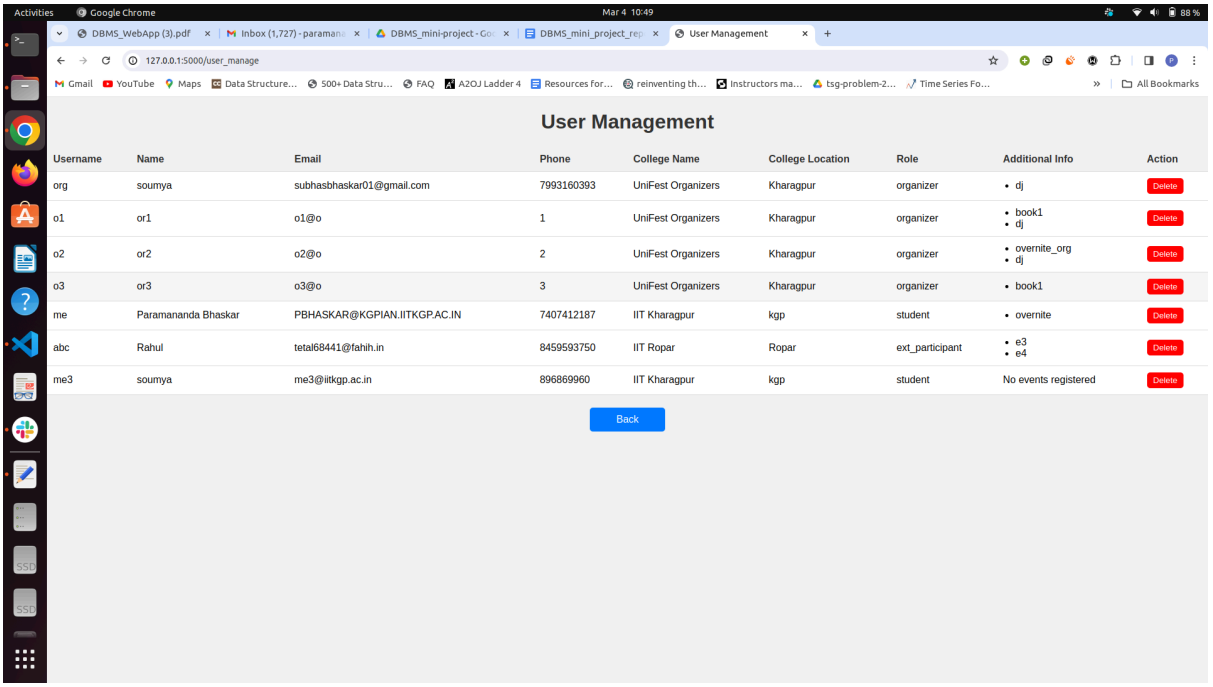
- Description:
    - This route allows students and external participants to view and register for events.
    - Participants can see event details such as name, date, starting time, maximum participants, event type, and remaining spots.
    - Registered participants can unregister from events.
- Validation Rules:
    - Only students and external participants can access this route.
    - Volunteers cannot participate in events.
    - Users can register for events only if they have not reached the maximum participation limit.
- Actions:
    - Fetches event details from the database along with the remaining spots.
    - Retrieves the user's registered events.
    - Handles event registration and cancellation based on user actions.
    - Displays appropriate flash messages for success or failure.
    - Renders the events page for participants with event details and registration options.



---

17. User Management Page (`/user_manage` route):

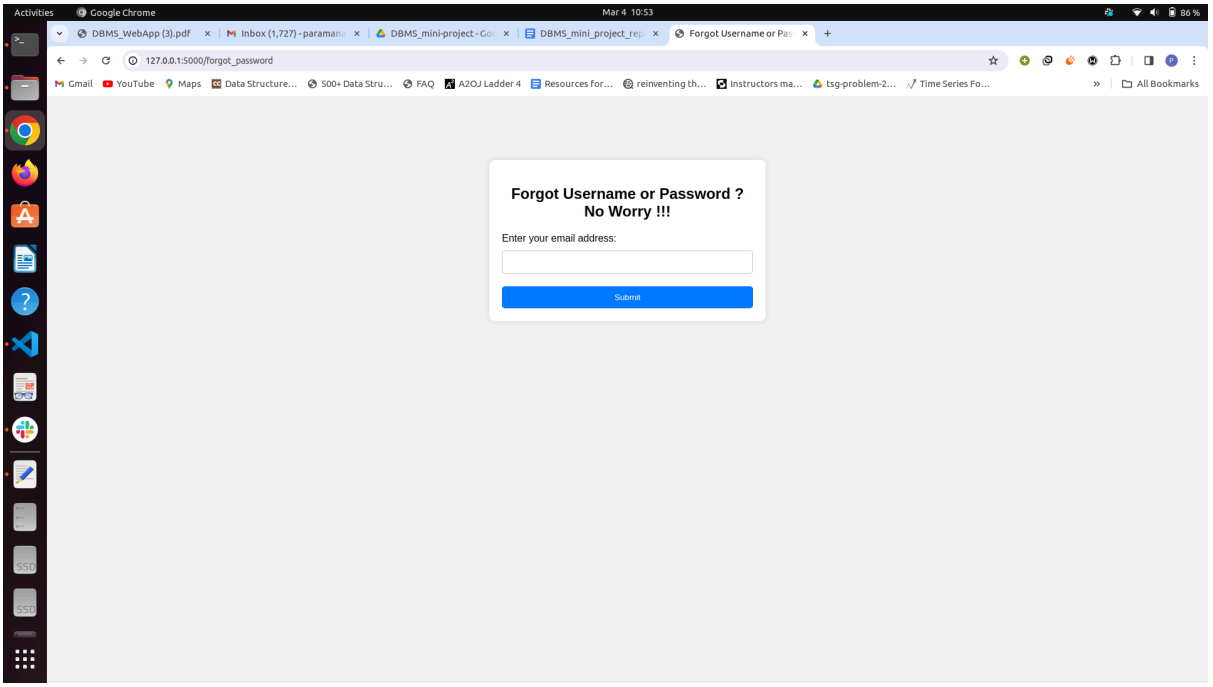
- Description:
  - This route allows admins to manage users, including deleting users and viewing their details.
  - Admins can delete users from the system if needed.
- Validation Rules:
  - Only admins can access this route.
- Actions:
  - Retrieves user details from the database excluding the admin user.
  - Fetches additional information based on the user's role (organizer or participant).
  - Displays a list of users with their details and associated organizing teams or registered events.
  - Provides an option for admins to delete users from the system.
  - Renders the user management page for admins.



---

18. Forgot Password Page (`/forgot_password` route):

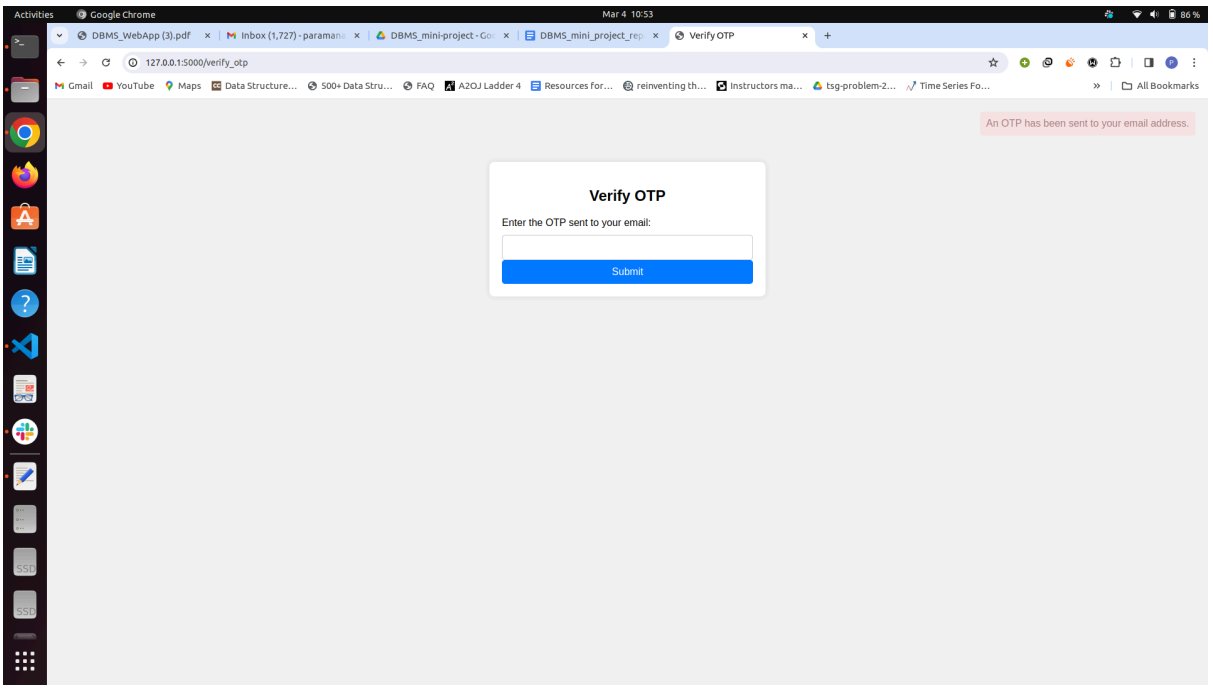
- Description:
  - This route allows users to reset their password if they forget it.
  - Users provide their email address to receive a one-time password (OTP) for verification.
- Validation Rules:
  - Users must enter a valid email address registered in the system.
- Actions:
  - Checks if the email exists in the database.
  - Generates and sends an OTP to the user's email address.
  - Stores the OTP in the session for verification.
  - Displays appropriate flash messages for success or failure.



---

19. OTP Verification Page (`/verify_otp` route):

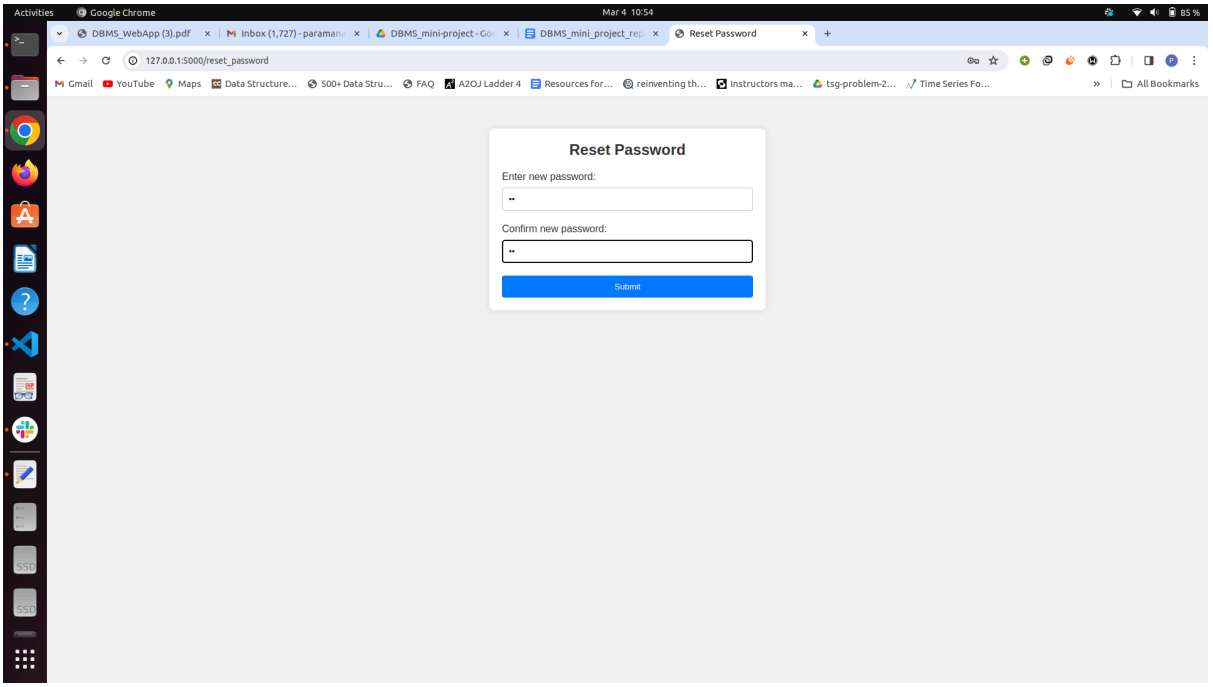
- Description:
  - This route allows users to verify the OTP sent to their email address during the password reset process.
- Validation Rules:
  - Users must enter the correct OTP received in their email.
- Actions:
  - Verifies the entered OTP against the one stored in the session.
  - Redirects the user to the password reset page upon successful verification.
  - Displays appropriate flash messages for success or failure.



---

20. Password Reset Page (`/reset_password` route):

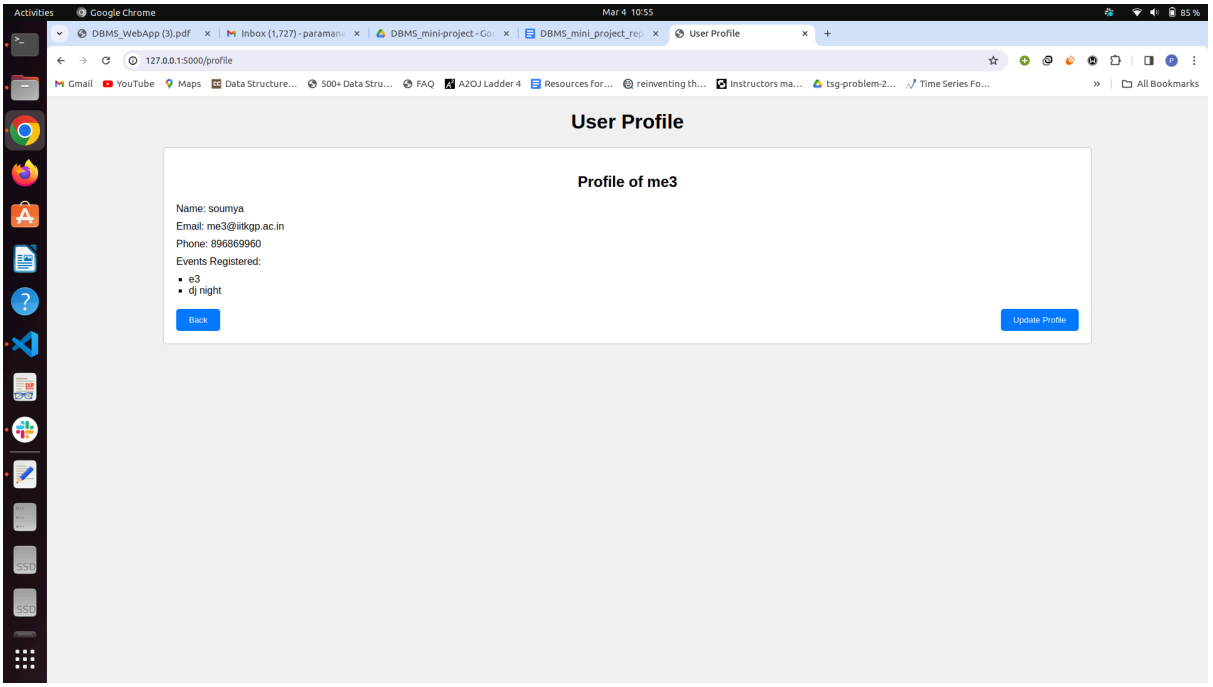
- Description:
  - This route allows users to reset their password after OTP verification.
- Validation Rules:
  - Users must enter a new password and confirm it correctly.
- Actions:
  - Updates the user's password in the database.
  - Clears the OTP and email from the session upon successful password reset.
  - Displays appropriate flash messages for success or failure.

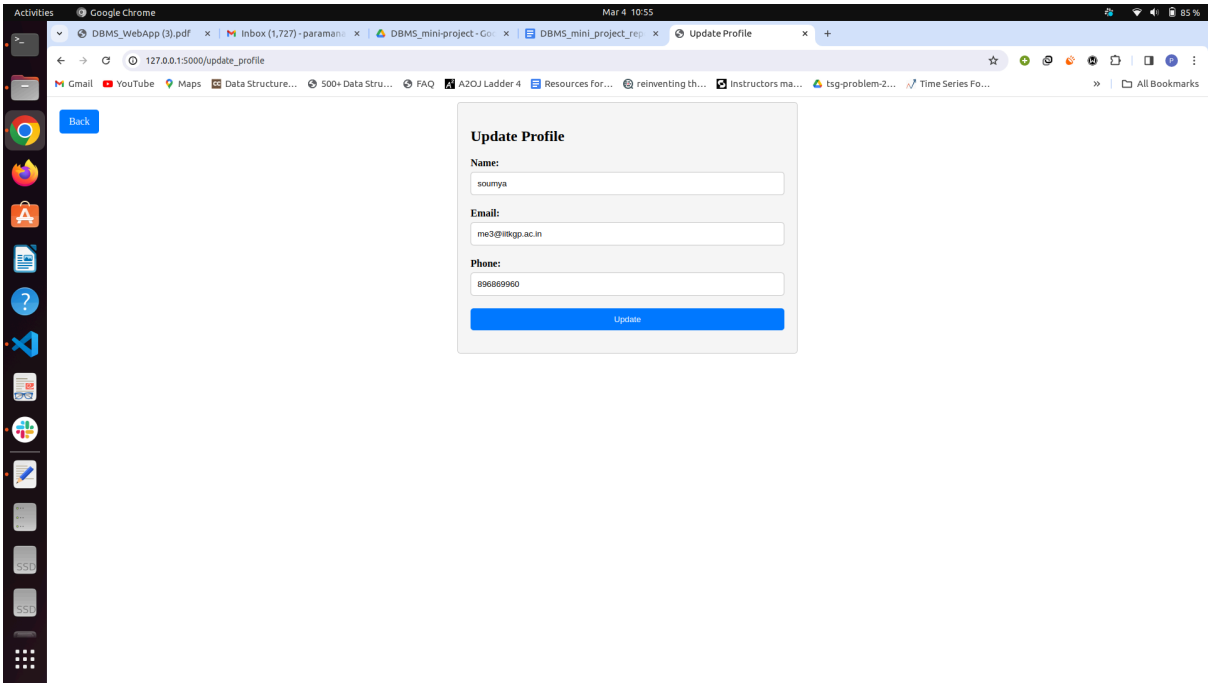


---

21. Update Profile Page (`/update_profile` route):

- Description:
  - This route allows users to update their profile information such as name, email, and phone number.
- Validation Rules:
  - Users must be logged in to access this page.
- Actions:
  - If accessed via GET request:
    - Fetches the user's current profile information from the database.
    - Renders the update profile page with pre-filled form fields.
  - If accessed via POST request:
    - Retrieves the updated profile information from the form.
    - Updates the user's profile information in the database.
    - Displays appropriate flash messages for success or failure.
    - Redirects the user to the profile page after updating.


Activities
Google Chrome
Mar 4 10:55
85 %
DBMS_WebApp (3).pdf
Inbox (1,727) - paramana
DBMS_mini-project - Goo
DBMS_mini_project_rep
User Profile
127.0.0.1:5000/profile
Gmail
YouTube
Maps
Data Structure...
500+ Data Stru...
FAQ
A2OJ Ladder 4
Resources for...
reinventing th...
Instructors ma...
tsg-problem-2...
Time Series Fo...
All Bookmarks
User Profile
Profile of me3
Name: soumya
Email: me3@iitkgp.ac.in
Phone: 896869960
Events Registered:
e3
dj night
Back
Update Profile


Activities
Google Chrome
Mar 4 10:55
85 %
DBMS_WebApp (3).pdf
Inbox (1,727) - paramana
DBMS_mini-project - Goo
DBMS_mini_project_rep
Update Profile
127.0.0.1:5000/update_profile
Gmail
YouTube
Maps
Data Structure...
500+ Data Stru...
FAQ
A2OJ Ladder 4
Resources for...
reinventing th...
Instructors ma...
tsg-problem-2...
Time Series Fo...
All Bookmarks
Back
Update Profile
Name:
soumya
Email:
me3@iitkgp.ac.in
Phone:
896869960
Update