ACADEMIC EVENT AND CONFERENCE MANAGEMENT

1)NAME: MADAMANCHI RAHUL SRN: PES1UG22CS316
2)NAME: MALLIKARJUN R SRN:PES1UG22CS326

2)Description:

This project is all about simplifying how academic events like conferences, seminars, and workshops are organized. It's a digital platform where organizers can plan events, manage registrations, handle paper submissions, and coordinate schedules—all in one place. By automating repetitive tasks and keeping everything organized, the system makes it easier for everyone involved to focus on what matters: sharing knowledge and ideas.

3)Purpose of the Project

The purpose of the Academic Event and Conference Management System is to streamline the complex and time-consuming process of organizing academic events. It aims to simplify tasks such as registration, paper submission, review management, and scheduling while ensuring a seamless experience for participants, organizers, and reviewers. By leveraging automation and a centralized platform, this system reduces administrative workload, eliminates errors, and fosters collaboration within the academic community, allowing organizers to focus more on the event's content and impact.

Scope of the Project

This system is designed for academic institutions, research organizations, and event organizers who conduct conferences, seminars, and workshops. It supports end-to-end event management, including pre-event activities like planning, registration, and paper reviews, real-time event management, and post-event reporting. With features like secure payment integration, automated notifications, and user-friendly dashboards, the platform is scalable to handle both small and large events. The project also includes multi-device accessibility, ensuring that users can engage with the platform from anywhere, making it versatile and highly adaptable.

Detailed Description

The Academic Event and Conference Management System is a one-stop solution for managing academic events. Here's how it works:

1. **Registration and Ticketing:** Participants and attendees can register through a user-friendly interface and make secure payments online.
2. **Paper Submission and Review:** Authors can submit research papers, which are then assigned to reviewers for evaluation. The system supports transparent, structured review workflows.
3. **Event Scheduling:** Organizers can create and share schedules, assign sessions to speakers, and provide real-time updates.
4. **Notifications and Updates:** Automated email and SMS notifications keep participants informed about deadlines, event updates, and session changes.

5. **Post-Event Analytics:** The system generates insightful reports on attendance, feedback, and performance, helping organizers improve future events.

4)Technologies, Tools, and Programming Languages Used;

Programming Languages:

- **JavaScript/TypeScript:** Used for both frontend and backend development, providing flexibility and ensuring code consistency across the system.

Frontend Development:

- **Next.js:** A React-based framework for building a fast, responsive, and dynamic user interface.

Backend Development:

- **Node.js:** The runtime environment powering the server-side logic of the application.
- **Prisma:** A database ORM (Object-Relational Mapping) tool used to simplify database operations.

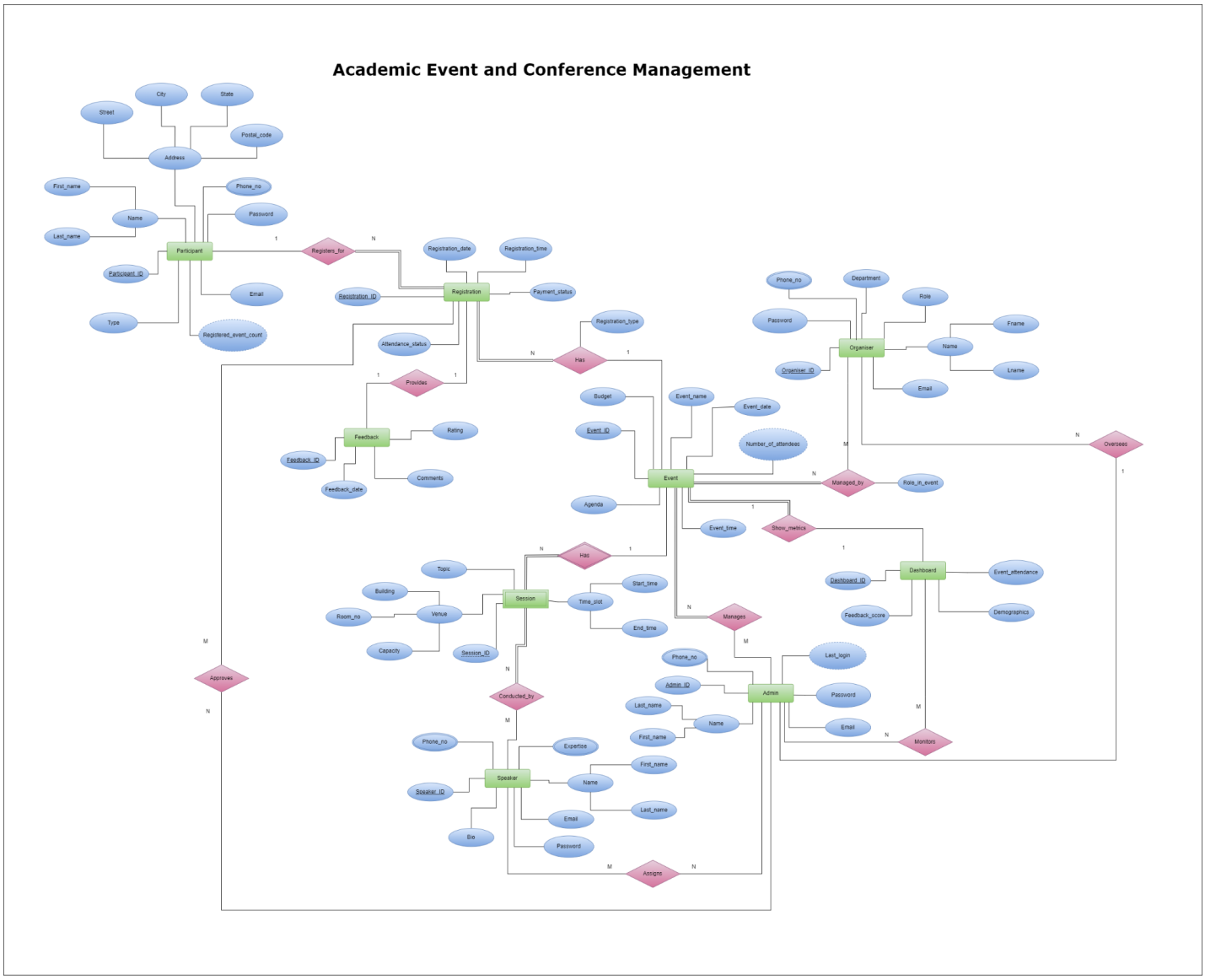
Database Management:

- **SQL:** The language used to manage and query structured data stored in the database.

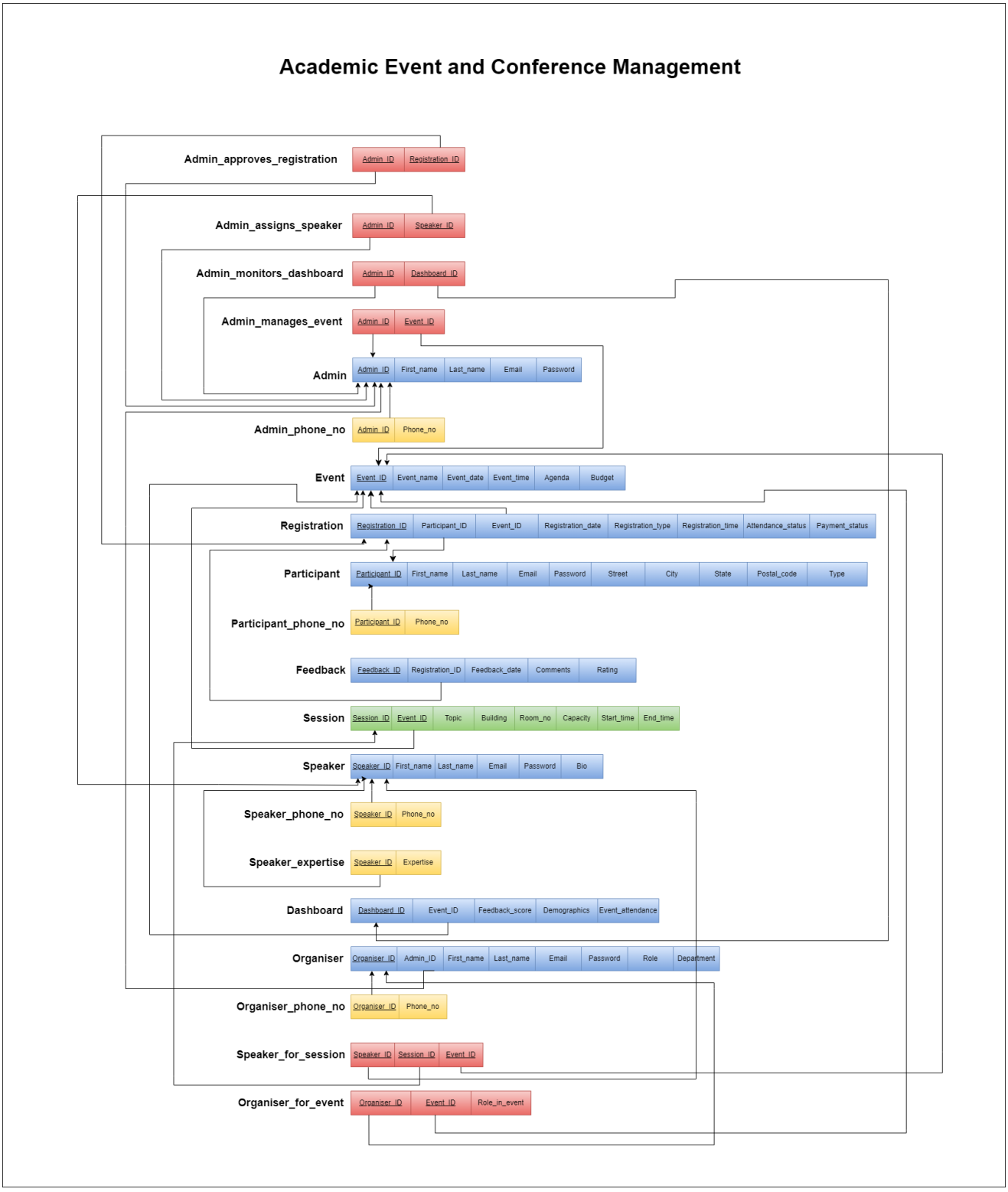
Development Tools:

- **Visual Studio Code:** A code editor for writing and debugging the application.
- **Git:** Version control to track changes and collaborate effectively.

5)
ER diagram:


Academic Event and Conference Management
City
State
Street
Postal_code
Address
First_name
Phone_no
Name
Password
Last_name
Participant
Participant_ID
Email
Type
Registered_event_count
Registers_for
Registration_date
Registration_time
Registration
Registration_ID
Payment_status
Attendance_status
Registration_type
Has
Provides
Feedback
Rating
Feedback_ID
Comments
Feedback_date
Budget
Event_name
Event_date
Event_ID
Number_of_attendees
Event
Agenda
Event_time
Show_metrics
Phone_no
Department
Role
Password
Fname
Organiser
Name
Organiser_ID
Lname
Email
Oversees
Managed_by
Role_in_event
Has
Topic
Building
Start_time
Session
Time_slot
Room_no
Venue
End_time
Capacity
Session_ID
Manages
Dashboard
Dashboard_ID
Event_attendance
Feedback_score
Demographics
Approves
Conducted_by
Phone_no
Last_login
Admin_ID
Admin
Password
Last_name
Name
Email
First_name
Monitors
Phone_no
Expertise
First_name
Speaker
Speaker_ID
Name
Email
Last_name
Bio
Password
Assigns

6)Relational Schema:



7) DDL commands:

1. `CREATE DATABASE` to create the `event_master` database.
2. `CREATE TABLE` statements to create all the required tables, including their columns, data types, primary keys, foreign keys, and other constraints.
3. `DROP TABLE` statement to drop the `dashboard` table if it already exists (in case of schema changes).
4. Junction/bridge tables like `admin_approves_registration`, `admin_assigns_speaker`, `admin_manages_event`, `admin_monitors_dashboard`, and `organiser_for_event` are also created.

```
CREATE DATABASE event_master;

USE event_master;

-- Create admin table
CREATE TABLE admin (
  admin_id VARCHAR(20) PRIMARY KEY,
  first_name VARCHAR(20) NOT NULL,
  last_name VARCHAR(20) NOT NULL,
  email VARCHAR(32) UNIQUE,
  -- Other fields and relations
);

-- Create admin_phone_no table
CREATE TABLE admin_phone_no (
  admin_id VARCHAR(20),
  phone_number VARCHAR(20),
  PRIMARY KEY (admin_id, phone_number),
  FOREIGN KEY (admin_id) REFERENCES admin(admin_id) ON DELETE CASCADE ON
UPDATE NO ACTION
);

-- Create dashboard table
DROP TABLE IF EXISTS dashboard;
CREATE TABLE dashboard (
  dashboard_id VARCHAR(20) PRIMARY KEY,
  event_id VARCHAR(20) NOT NULL,
  feedback_score DECIMAL(3,2) DEFAULT 0.00,
  attendance_count INT DEFAULT 0,
  FOREIGN KEY (event_id) REFERENCES event(id) ON DELETE CASCADE ON UPDATE
NO ACTION
);

-- Create event table
CREATE TABLE event (
  id VARCHAR(20) PRIMARY KEY,
  title VARCHAR(32) NOT NULL,
  date DATE NOT NULL,
  time TIME(0) NOT NULL,
```

```
  budget INT,
  description TEXT,
  -- Other fields and relations
);

-- Create event_agenda table
CREATE TABLE event_agenda (
  agenda_id VARCHAR(20) PRIMARY KEY,
  event_id VARCHAR(20) NOT NULL,
  item TEXT NOT NULL,
  FOREIGN KEY (event_id) REFERENCES event(id) ON DELETE CASCADE ON UPDATE
NO ACTION
);

-- Create event_session table
CREATE TABLE event_session (
  event_session_id VARCHAR(20),
  event_id VARCHAR(20),
  topic VARCHAR(100) NOT NULL,
  building VARCHAR(20) NOT NULL,
  room_no VARCHAR(10) NOT NULL,
  start_time TIME(0) NOT NULL,
  end_time TIME(0) NOT NULL,
  PRIMARY KEY (event_session_id, event_id),
  FOREIGN KEY (event_id) REFERENCES event(id) ON DELETE CASCADE ON UPDATE
NO ACTION
);

-- Create organiser table
CREATE TABLE organiser (
  organiser_id VARCHAR(20) PRIMARY KEY,
  admin_id VARCHAR(20),
  first_name VARCHAR(20) NOT NULL,
  last_name VARCHAR(20),
  email VARCHAR(32) UNIQUE,
  role VARCHAR(20) NOT NULL,
  department VARCHAR(32),
  FOREIGN KEY (admin_id) REFERENCES admin(admin_id) ON DELETE CASCADE ON
UPDATE NO ACTION
);
```

```sql
-- Create organiser_phone_no table
CREATE TABLE organiser_phone_no (
  organiser_id VARCHAR(20),
  phone_number VARCHAR(20),
  PRIMARY KEY (organiser_id, phone_number),
  FOREIGN KEY (organiser_id) REFERENCES organiser(organiser_id) ON DELETE
CASCADE ON UPDATE NO ACTION
);

-- Create participant table
DROP TABLE IF EXISTS participant;
CREATE TABLE participant (
  participant_id VARCHAR(20) PRIMARY KEY,
  first_name VARCHAR(20) NOT NULL,
  last_name VARCHAR(20),
  email VARCHAR(30) UNIQUE,
  type ENUM('Student', 'Faculty', 'External') NOT NULL,
  address_id VARCHAR(20),
  FOREIGN KEY (address_id) REFERENCES address(address_id) ON UPDATE NO
ACTION
);

-- Create participant_phone_no table
CREATE TABLE participant_phone_no (
  participant_id VARCHAR(20),
  phone_number VARCHAR(15),
  PRIMARY KEY (participant_id, phone_number),
  FOREIGN KEY (participant_id) REFERENCES participant(participant_id) ON
DELETE CASCADE ON UPDATE NO ACTION
);

-- Create registration table
CREATE TABLE registration (
  registration_id VARCHAR(20) PRIMARY KEY,
  participant_id VARCHAR(20),
  event_id VARCHAR(20) NOT NULL,
  registration_date DATE NOT NULL,
  type ENUM('Early Bird', 'Standard', 'VIP') NOT NULL,
  registration_time TIME(0) NOT NULL,
```

```
  attendance_status ENUM('Attended', 'Absent') NOT NULL,
  FOREIGN KEY (participant_id) REFERENCES participant(participant_id) ON
DELETE NO ACTION ON UPDATE NO ACTION,
  FOREIGN KEY (event_id) REFERENCES event(id) ON DELETE CASCADE ON UPDATE
NO ACTION,
  UNIQUE (event_id, participant_id)
);

-- Create session table
CREATE TABLE session (
  userId VARCHAR(20),
  created_at DATETIME(0) DEFAULT CURRENT_TIMESTAMP,
  id VARCHAR(64) PRIMARY KEY,
  expiresAt DATETIME(0) NOT NULL,
  type ENUM('Participant', 'Organiser', 'Speaker') NOT NULL,
  FOREIGN KEY (userId) REFERENCES User(id) ON DELETE CASCADE ON UPDATE NO
ACTION
);

-- Create speaker table
CREATE TABLE speaker (
  speaker_id VARCHAR(20) PRIMARY KEY,
  first_name VARCHAR(20) NOT NULL,
  last_name VARCHAR(20),
  email VARCHAR(30) UNIQUE,
  bio TEXT
);

-- Create speaker_expertise table
CREATE TABLE speaker_expertise (
  speaker_id VARCHAR(20),
  expertise VARCHAR(30),
  PRIMARY KEY (speaker_id, expertise),
  FOREIGN KEY (speaker_id) REFERENCES speaker(speaker_id) ON DELETE
CASCADE ON UPDATE NO ACTION
);

-- Create speaker_for_session table
CREATE TABLE speaker_for_session (
  speaker_id VARCHAR(20),
```

```
  session_id VARCHAR(20),
  event_id VARCHAR(20),
  PRIMARY KEY (speaker_id, session_id, event_id),
  FOREIGN KEY (speaker_id) REFERENCES speaker(speaker_id) ON DELETE
CASCADE ON UPDATE NO ACTION,
  FOREIGN KEY (session_id, event_id) REFERENCES
event_session(event_session_id, event_id) ON DELETE CASCADE ON UPDATE NO
ACTION
);

-- Create speaker_phone_no table
CREATE TABLE speaker_phone_no (
  speaker_id VARCHAR(20),
  phone_number VARCHAR(20),
  PRIMARY KEY (speaker_id, phone_number),
  FOREIGN KEY (speaker_id) REFERENCES speaker(speaker_id) ON DELETE
CASCADE ON UPDATE NO ACTION
);

-- Create feedback table
CREATE TABLE feedback (
  feedback_id VARCHAR(20) PRIMARY KEY,
  registration_id VARCHAR(20),
  feedback_date DATE NOT NULL,
  rating INT,
  comments TEXT,
  FOREIGN KEY (registration_id) REFERENCES registration(registration_id)
ON DELETE NO ACTION ON UPDATE NO ACTION
);

-- Create User table
CREATE TABLE User (
  email VARCHAR(32),
  password VARCHAR(40),
  created_at DATETIME(0) DEFAULT CURRENT_TIMESTAMP,
  id VARCHAR(20) PRIMARY KEY,
  participant_id VARCHAR(20),
  organiser_id VARCHAR(20),
  speaker_id VARCHAR(20),
```

```
  FOREIGN KEY (participant_id) REFERENCES participant(participant_id) ON
UPDATE NO ACTION,
  FOREIGN KEY (organiser_id) REFERENCES organiser(organiser_id) ON UPDATE
NO ACTION,
  FOREIGN KEY (speaker_id) REFERENCES speaker(speaker_id) ON UPDATE NO
ACTION,
  INDEX (organiser_id),
  INDEX (participant_id),
  INDEX (speaker_id)
);

-- Create address table
CREATE TABLE address (
  address_id VARCHAR(20) PRIMARY KEY,
  street TEXT NOT NULL,
  postal_code CHAR(6) NOT NULL,
  city_id VARCHAR(20),
  FOREIGN KEY (city_id) REFERENCES city(city_id) ON DELETE CASCADE ON
UPDATE NO ACTION
);

-- Create city table
CREATE TABLE city (
  city_id VARCHAR(20) PRIMARY KEY,
  city_name VARCHAR(50) NOT NULL,
  state_id VARCHAR(20),
  FOREIGN KEY (state_id) REFERENCES state(state_id) ON DELETE CASCADE ON
UPDATE NO ACTION
);

-- Create state table
CREATE TABLE state (
  state_id VARCHAR(20) PRIMARY KEY,
  state_name VARCHAR(50) NOT NULL
);

-- Create demographic_data table
CREATE TABLE demographic_data (
  demographic_id VARCHAR(20) PRIMARY KEY,
  dashboard_id VARCHAR(20),
```

```
  category VARCHAR(50) NOT NULL,
  value VARCHAR(50) NOT NULL,
  count INT NOT NULL,
  FOREIGN KEY (dashboard_id) REFERENCES dashboard(dashboard_id) ON DELETE
CASCADE ON UPDATE NO ACTION
);

-- Create admin bridge tables
CREATE TABLE admin_approves_registration (
  admin_id VARCHAR(20),
  registration_id VARCHAR(20),
  PRIMARY KEY (admin_id, registration_id),
  FOREIGN KEY (admin_id) REFERENCES admin(admin_id) ON DELETE CASCADE ON
UPDATE NO ACTION,
  FOREIGN KEY (registration_id) REFERENCES registration(registration_id)
ON DELETE CASCADE ON UPDATE NO ACTION
);

CREATE TABLE admin_assigns_speaker (
  admin_id VARCHAR(20),
  speaker_id VARCHAR(20),
  PRIMARY KEY (admin_id, speaker_id),
  FOREIGN KEY (admin_id) REFERENCES admin(admin_id) ON DELETE CASCADE ON
UPDATE NO ACTION,
  FOREIGN KEY (speaker_id) REFERENCES speaker(speaker_id) ON DELETE
CASCADE ON UPDATE NO ACTION
);

CREATE TABLE admin_manages_event (
  admin_id VARCHAR(20),
  event_id VARCHAR(20),
  PRIMARY KEY (admin_id, event_id),
  FOREIGN KEY (admin_id) REFERENCES admin(admin_id) ON DELETE CASCADE ON
UPDATE NO ACTION,
  FOREIGN KEY (event_id) REFERENCES event(id) ON DELETE CASCADE ON UPDATE
NO ACTION
);

CREATE TABLE admin_monitors_dashboard (
  admin_id VARCHAR(20),
```

```
  dashboard_id VARCHAR(20),
  PRIMARY KEY (admin_id, dashboard_id),
  FOREIGN KEY (admin_id) REFERENCES admin(admin_id) ON DELETE CASCADE ON
UPDATE NO ACTION,
  FOREIGN KEY (dashboard_id) REFERENCES dashboard(dashboard_id) ON DELETE
CASCADE ON UPDATE NO ACTION
);

-- Create organiser_for_event bridge table
CREATE TABLE organiser_for_event (
  organiser_id VARCHAR(20),
  event_id VARCHAR(20),
  PRIMARY KEY (organiser_id, event_id),
  FOREIGN KEY (organiser_id) REFERENCES organiser(organiser_id) ON DELETE
CASCADE ON UPDATE NO ACTION,
  FOREIGN KEY (event_id) REFERENCES event(id) ON DELETE CASCADE ON UPDATE
NO ACTION
);
```

8. CRUD Operation Screenshots:

```
// Update existing session
await prisma.$executeRaw`
  UPDATE event_session
  SET
    topic = ${session.title},
    start_time = ${new Date(session.start_time)},
    end_time = ${new Date(session.end_time)},
    building = ${session.location.split(' - ')[0]},
    room_no = ${session.location.split(' - ')[1]}
  WHERE event_session_id = ${session.session_id}
  AND event_id = ${eventId}
`;

// Update speaker assignment
await prisma.$executeRaw`
  DELETE FROM speaker_for_session
  WHERE session_id = ${session.session_id}
  AND event_id = ${eventId}
`;

if (session.speaker_id) {
  await prisma.$executeRaw`
    INSERT INTO speaker_for_session (speaker_id, session_id, event_id)
    VALUES (${session.speaker_id}, ${session.session_id}, ${eventId})
  `;
}
```

```
const [newSession] = await prisma.$queryRaw<[{ event_session_id: string }]>`
  INSERT INTO event_session (
    event_session_id, event_id, topic,
    start_time, end_time, building, room_no
  )
  VALUES (
    UUID(), ${eventId}, ${session.title},
    ${new Date(session.start_time)}, ${new Date(session.end_time)},
    ${session.location.split(' - ')[0]}, ${session.location.split(' - ')[1]}
  )
  RETURNING event_session_id
`;

// Insert speaker assignment if speaker is selected
if (session.speaker_id) {
  await prisma.$executeRaw`
    INSERT INTO speaker_for_session (speaker_id, session_id, event_id)
    VALUES (${session.speaker_id}, ${newSession.event_session_id}, ${eventId})
  `;
}
```

```
for (const sessionId of deletedSessionIds) {
  await prisma.$executeRaw`
    DELETE FROM event_session
    WHERE event_session_id = ${sessionId}
    AND event_id = ${eventId}
  `;
}
```

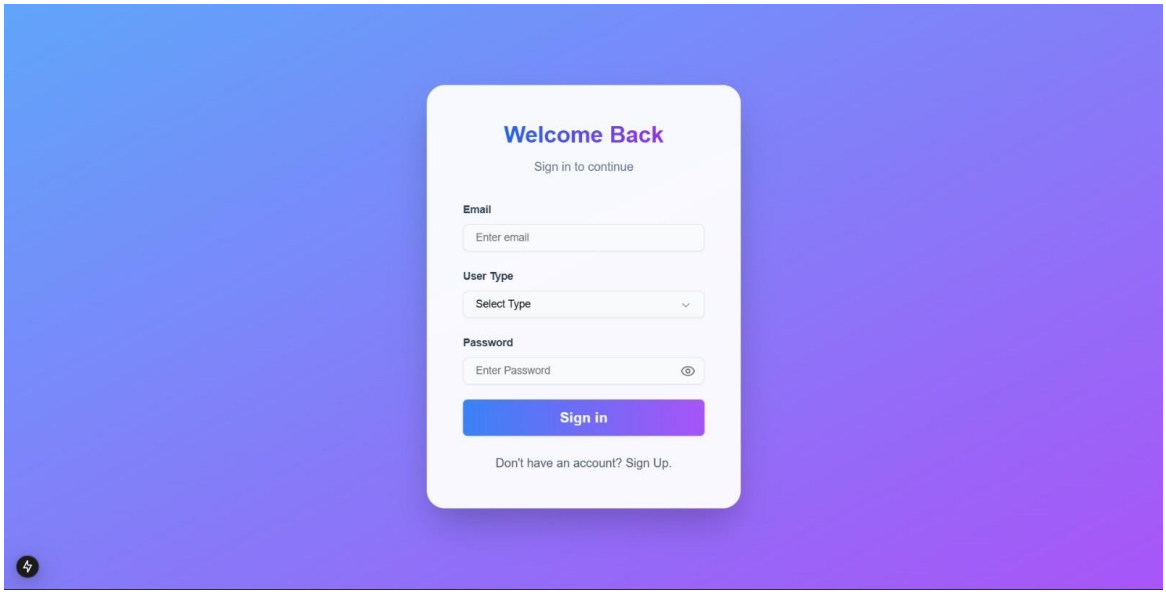
```
const existingSessions = await prisma.$queryRaw<{ event_session_id: string }[]>`
  SELECT event_session_id
  FROM event_session
  WHERE event_id = ${eventId}
`;
```

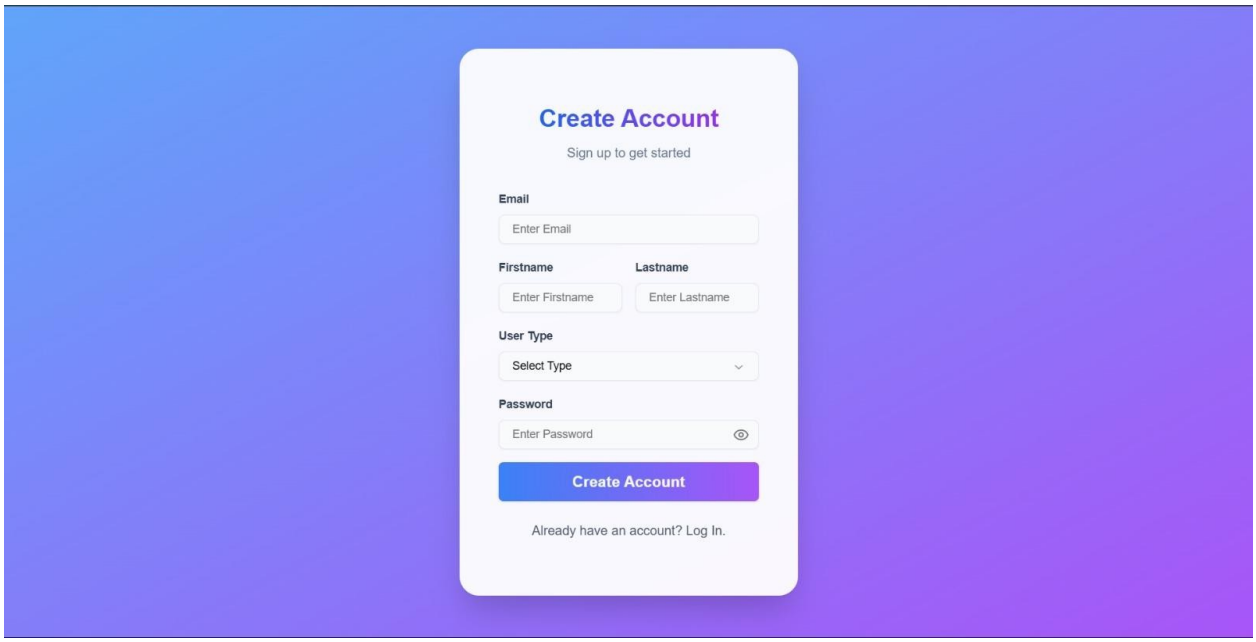
```
const speakers = await prisma.$queryRaw<Speaker[]>`
  SELECT
    speaker_id,
    CONCAT(first_name, ' ', COALESCE(last_name, '')) as name
  FROM speaker
  ORDER BY first_name, last_name
`;
return { data: speakers, error: null };
```

9. List of Functionalities / features of the application and itsassociated screenshots using frontend:

i. User Authentication

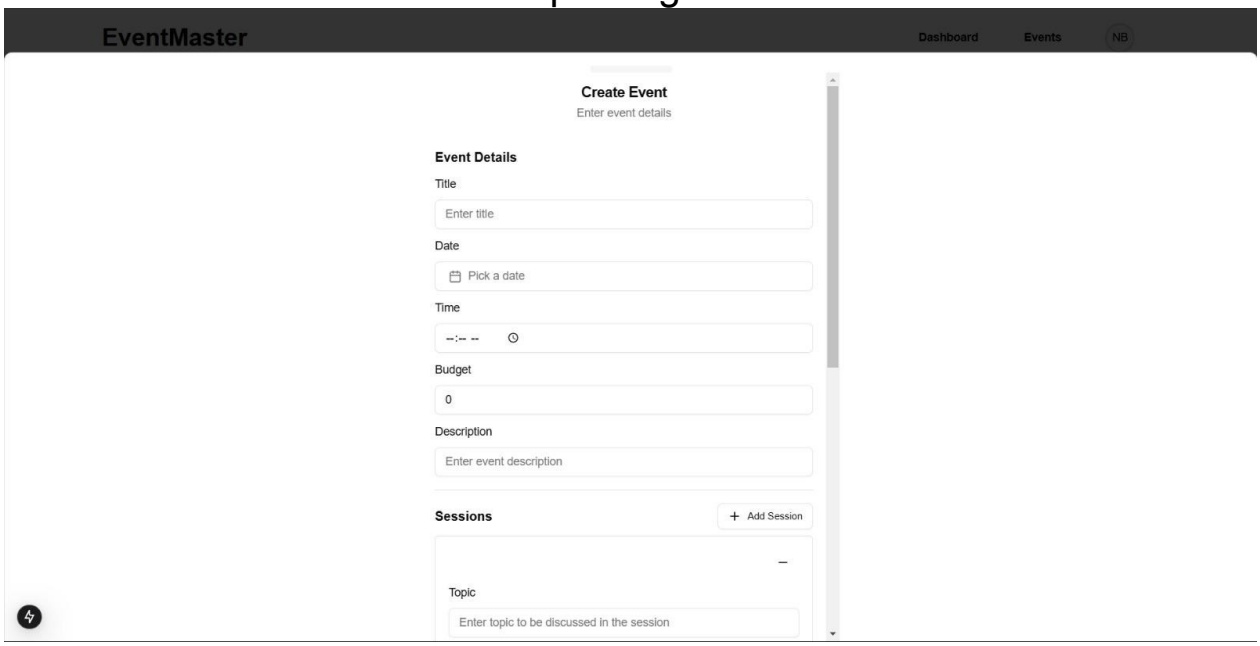
- Login and Sign-Up for Participants, Organizers, and Speakers.
- Description:
  - Secure authentication using credentials.
  - Different views based on user roles (participant, organizer,speaker).





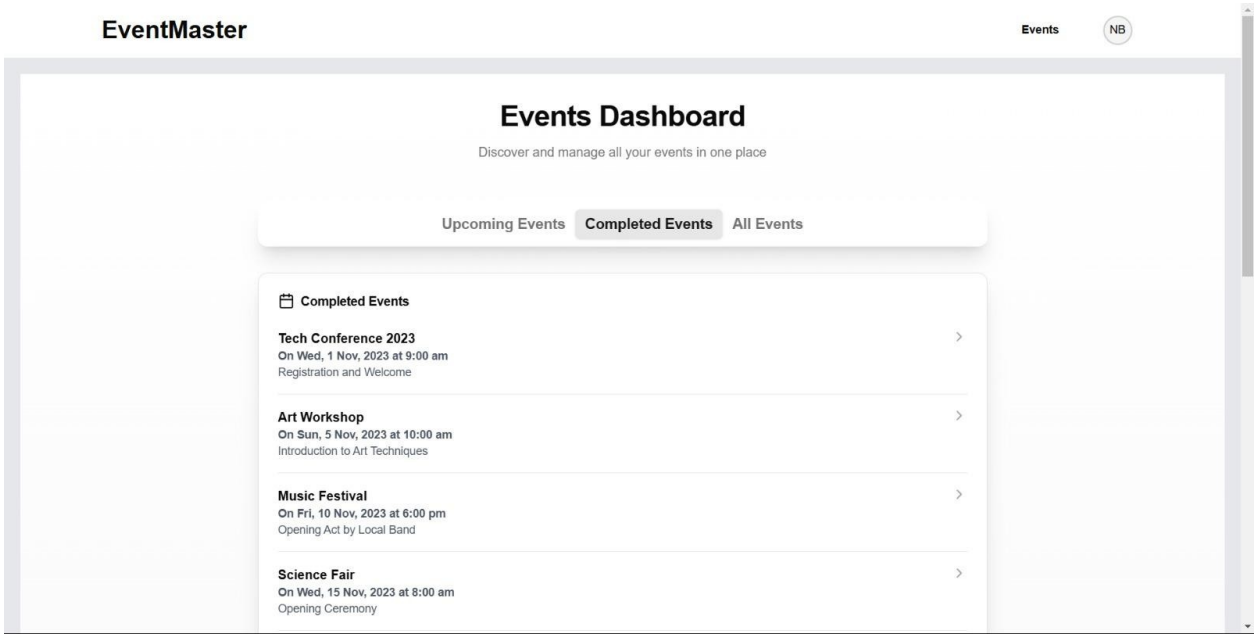
ii. Event Creation and Management

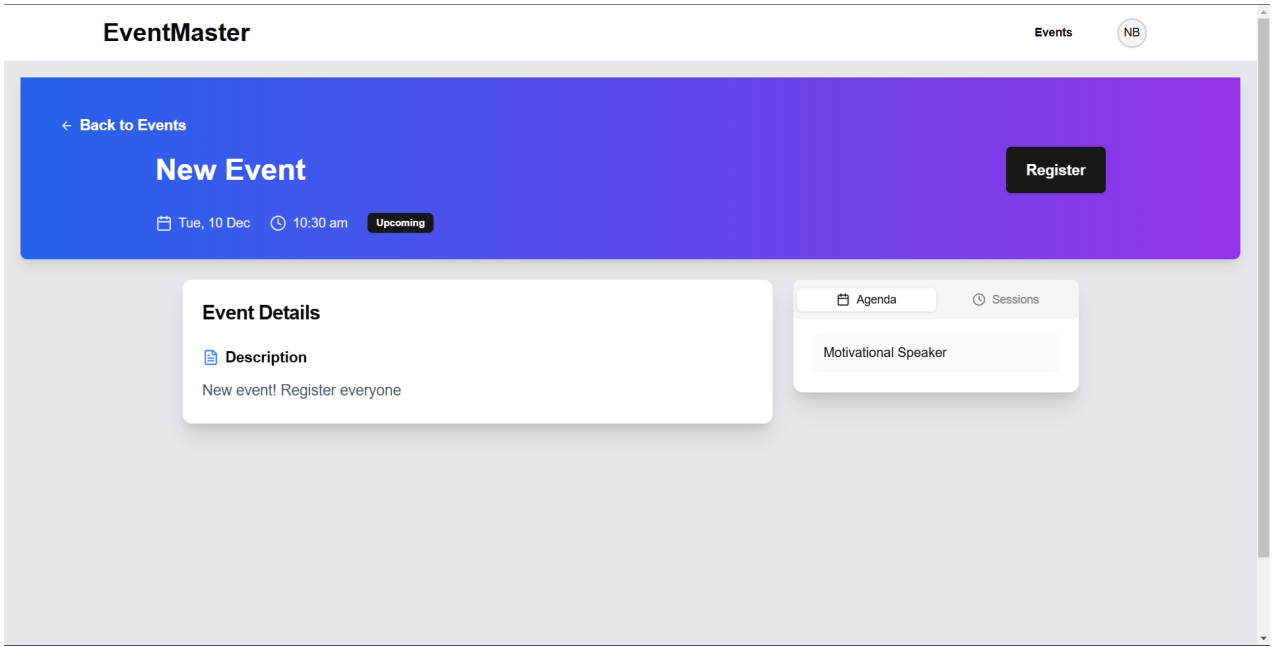
- Event creation and updates by organizers.
- Description:
    - Organizers can add event details (name, description, date,venue, etc.).
    - Editable fields for updating event information.



iii. Participant Registration

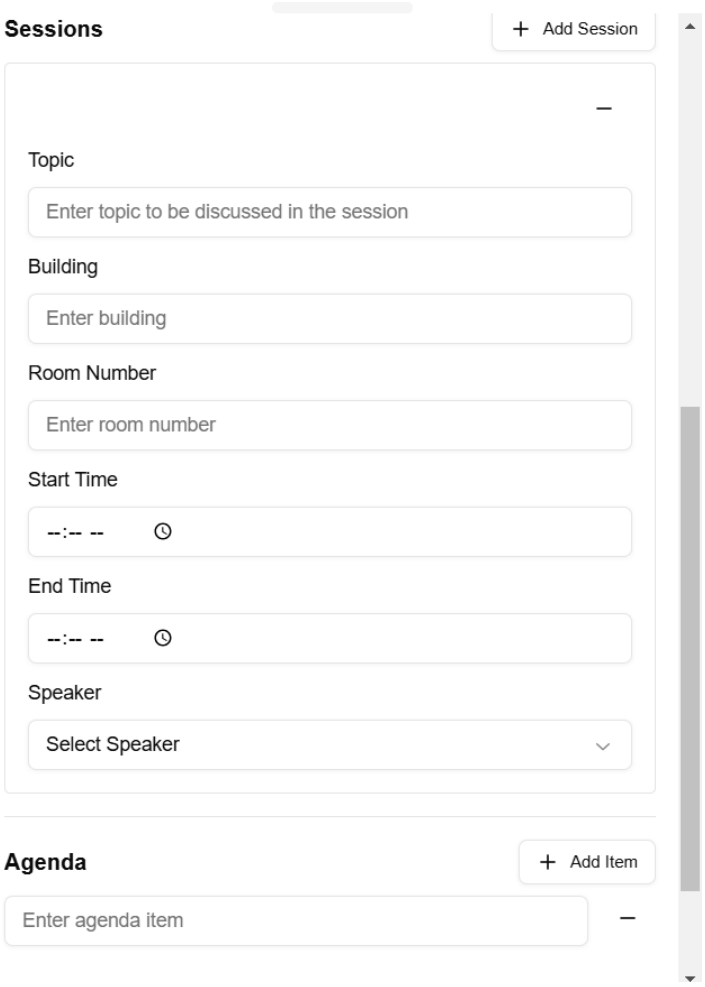
- Participants can register for events.
- Description:
    - User-friendly interface for selecting events.
    - Registration types (e.g., early bird, regular).


EventMaster
Events
NB
Events Dashboard
Discover and manage all your events in one place
Upcoming Events
Completed Events
All Events
Completed Events
Tech Conference 2023
On Wed, 1 Nov, 2023 at 9:00 am
Registration and Welcome
Art Workshop
On Sun, 5 Nov, 2023 at 10:00 am
Introduction to Art Techniques
Music Festival
On Fri, 10 Nov, 2023 at 6:00 pm
Opening Act by Local Band
Science Fair
On Wed, 15 Nov, 2023 at 8:00 am
Opening Ceremony


EventMaster
Events
NB
Back to Events
New Event
Register
Tue, 10 Dec
10:30 am
Upcoming
Event Details
Description
New event! Register everyone
Agenda
Sessions
Motivational Speaker

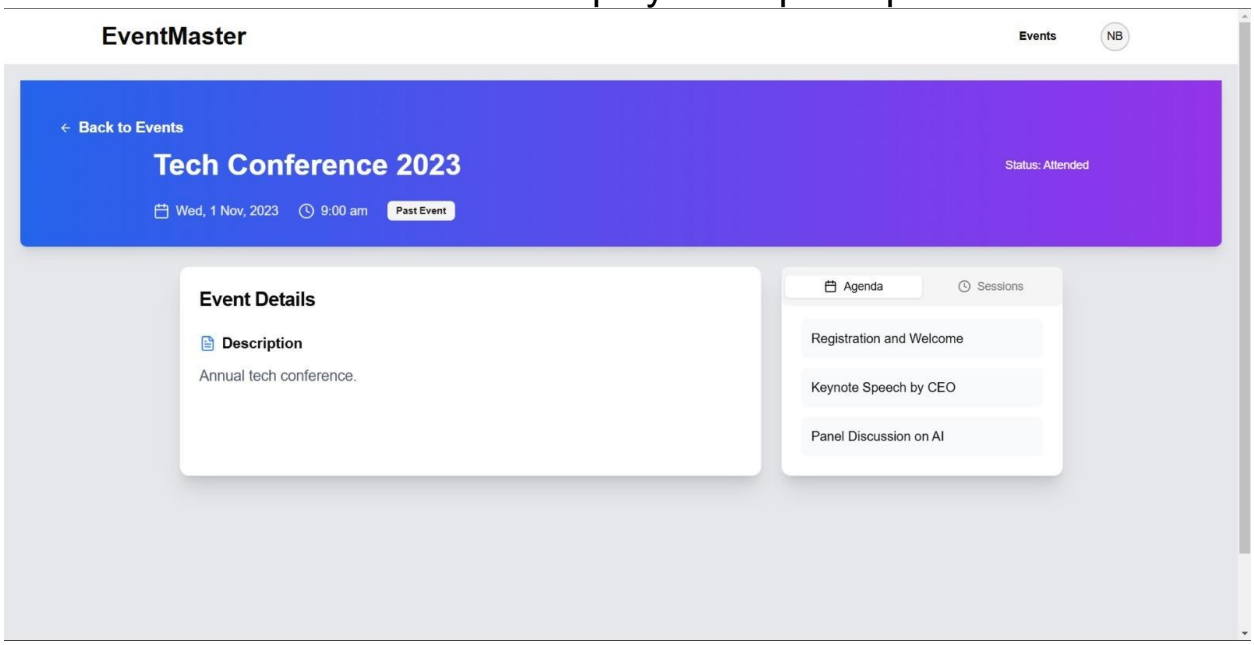
iv. Session and Speaker Management

- Manage sessions and assign speakers.
- Description:
    - Organizers can create



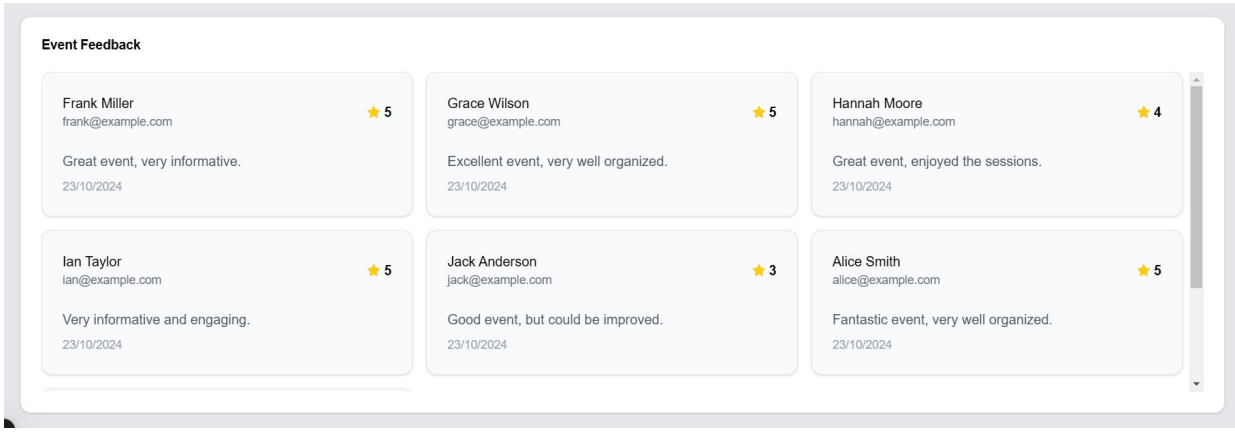
v. Attendance Management

- Mark attendance for participants.
- Description:
    - Manual attendance marking.
    - Attendance status displayed for participants.



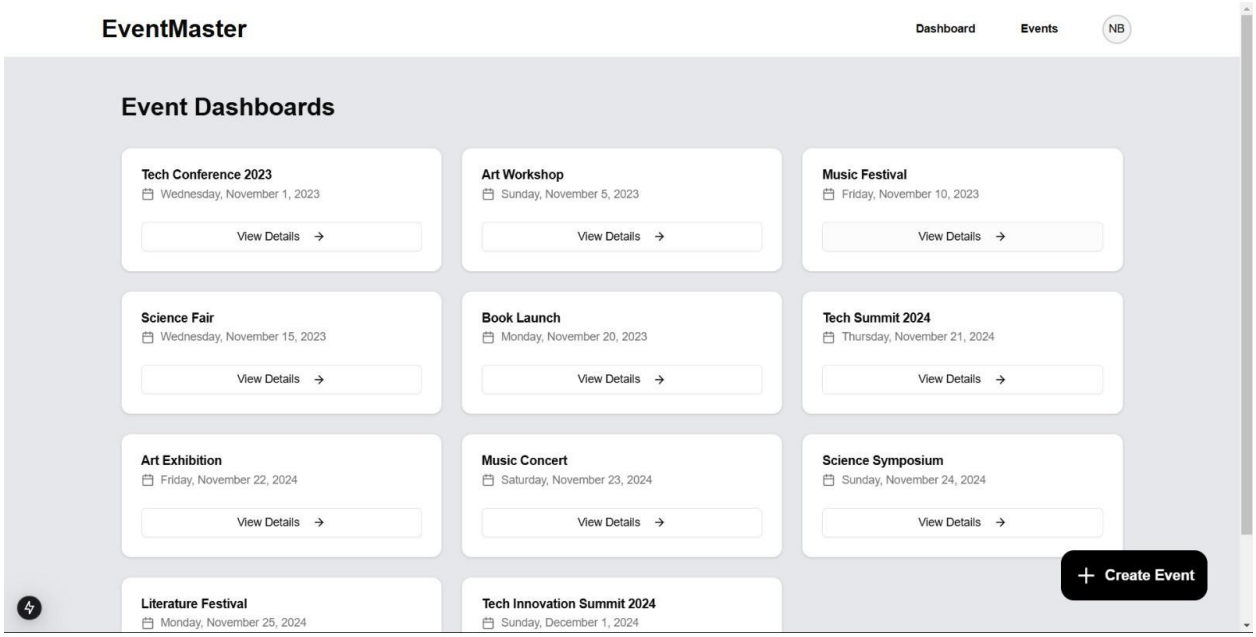
vi. Feedback Collection

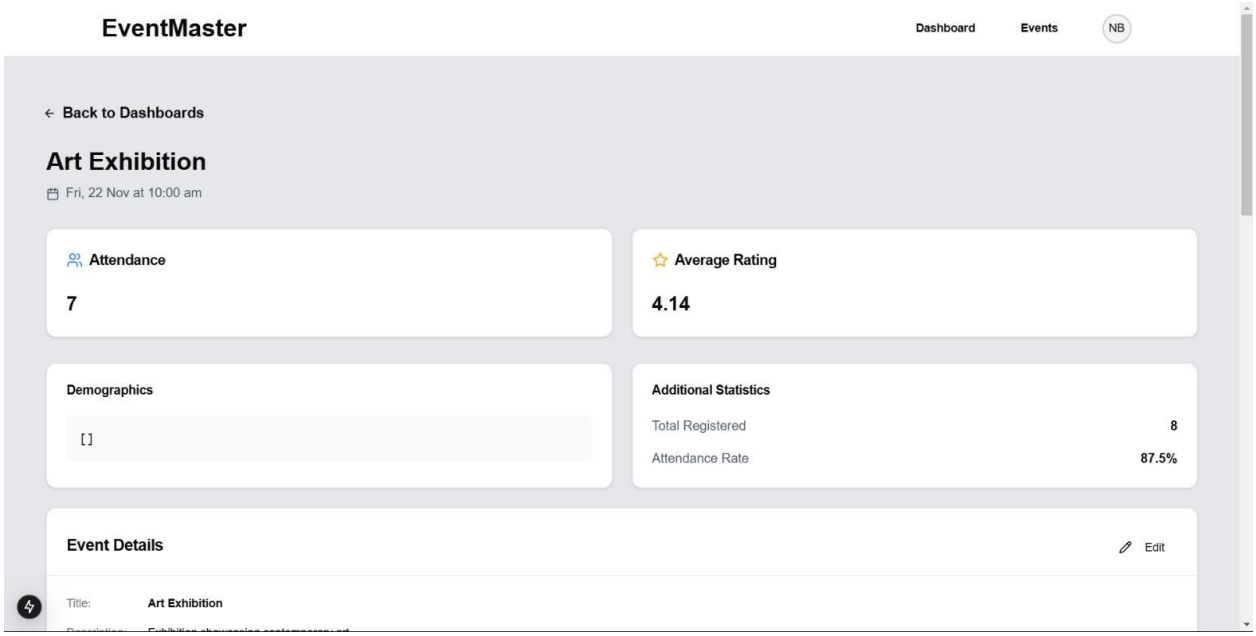
- Feedback submission after an event.
- Description:
    - Participants provide feedback on events/sessions.
    - Feedback analyzed and displayed to organizers.

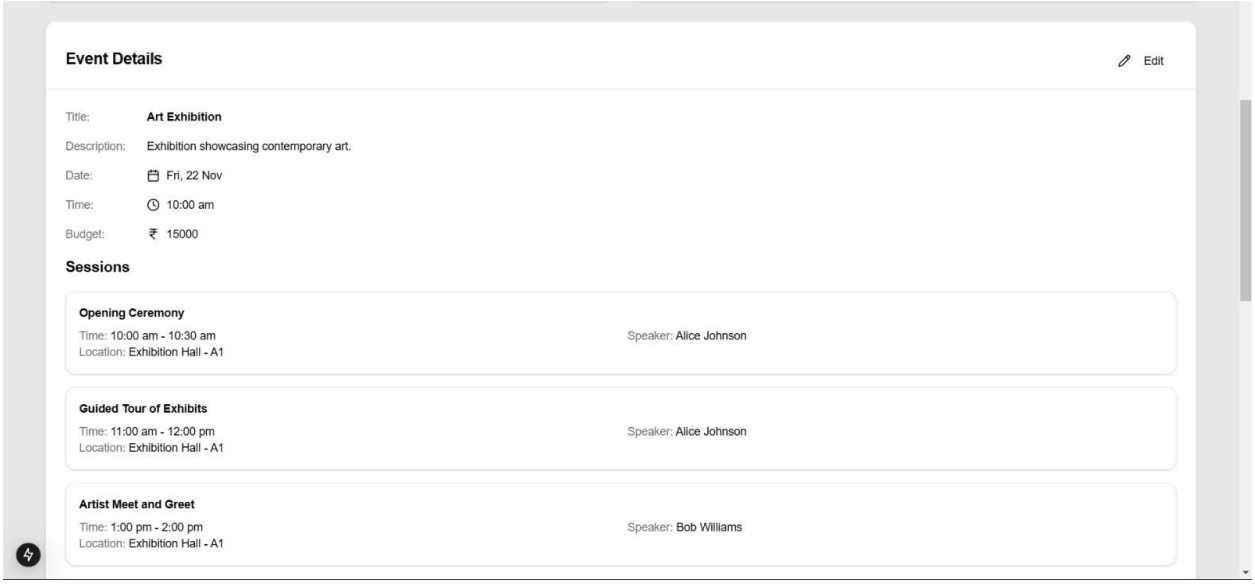


vii. Dashboard and Analytics

- Event dashboard with key metrics.
- Description:
    - Organizers can view the number of participants, sessiondetails, feedback summary, and attendance records.
    - Graphical representation of analytics.







10. Triggers, Procedures / Functions, Nested query, Join,Aggregate queries:

- Triggers:

```
ocalhost` TRIGGER `update_feedback_score` AFTER INSERT ON `feedback` FOR EACH ROW BEGIN
    DECLARE avg_rating DECIMAL(3,2);


    SELECT AVG(f.rating)
    INTO avg_rating
    FROM feedback f
    JOIN registration r ON f.registration_id = r.registration_id
    WHERE r.event_id = (
        SELECT event_id
        FROM registration
        WHERE registration_id = NEW.registration_id
    )
    AND f.rating IS NOT NULL;


    UPDATE dashboard d
    SET d.feedback_score = avg_rating
    WHERE d.event_id = (
        SELECT event_id
        FROM registration
        WHERE registration_id = NEW.registration_id
    );
```

```
`localhost` TRIGGER `update_attendance_count` AFTER UPDATE ON `registration` FOR EACH ROW BEGIN

    IF OLD.attendance_status = 'Absent' AND NEW.attendance_status = 'Attended' THEN
        UPDATE dashboard
        SET attendance_count = attendance_count + 1
        WHERE event_id = NEW.event_id;


    ELSEIF OLD.attendance_status = 'Attended' AND NEW.attendance_status = 'Absent' THEN
        UPDATE dashboard
        SET attendance_count = GREATEST(0, attendance_count - 1)
        WHERE event_id = NEW.event_id;
    END IF;
```

- Procedure:

```
root`@`localhost` PROCEDURE `GetEventDetails`(IN event_id VARCHAR(20))
BEGIN
    SELECT
        e.title, e.date, e.time, e.budget, e.description,
        JSON_ARRAYAGG(
            JSON_OBJECT(
                'topic', s.topic,
                'building', s.building,
                'room_no', s.room_no,
                'start_time', TIME_FORMAT(s.start_time, '%H:%i'),
                'end_time', TIME_FORMAT(s.end_time, '%H:%i'),
                'speaker_name', CONCAT(COALESCE(sp.first_name, ''), ' ', COALESCE(sp.last_name, ''))
            )
        ) AS sessions,
        JSON_ARRAYAGG(
            JSON_OBJECT(
                'agenda_id', ea.agenda_id,
                'item', ea.item
            )
        ) AS agendas
    FROM event e
    LEFT JOIN event_session s ON e.id = s.event_id
    LEFT JOIN speaker_for_session sfs ON s.event_session_id = sfs.session_id AND s.event_id = sfs.event_id
    LEFT JOIN speaker sp ON sfs.speaker_id = sp.speaker_id
    LEFT JOIN event_agenda ea ON e.id = ea.event_id
    WHERE e.id = event_id
    GROUP BY e.id;
```

- Functions:

```
NER=`root`@`localhost` FUNCTION `get_average_feedback`(p_event_id VARCHAR(20)) RETURNS decimal(3,2)
    DETERMINISTIC
BEGIN
    DECLARE avg_rating DECIMAL(3,2);

    SELECT AVG(f.rating)
    INTO avg_rating
    FROM feedback f
    INNER JOIN registration r ON f.registration_id = r.registration_id
    WHERE r.event_id = p_event_id
    AND f.rating IS NOT NULL;

    RETURN COALESCE(avg_rating, 0.00);
```

```
INER=`root`@`localhost` FUNCTION `get_total_registrants`(p_event_id VARCHAR(20)) RETURNS int
    DETERMINISTIC
BEGIN
    DECLARE total INT;
    SELECT COUNT(*) INTO total
    FROM registration
    WHERE event_id = p_event_id;
    RETURN total;
```

- Nested Query:

```
export async function getDashboardById(id: string) {
  try {
    const dashboard = await prisma.$queryRaw<DashboardDetailQueryResult[]>`
      SELECT
        d.dashboard_id,
        e.id as event_id,
        e.title,
        e.date,
        e.time,
        e.budget,
        e.description,
        get_total_registrants(e.id) as total_registrants,
        (
          SELECT COUNT(r.registration_id)
          FROM registration r
          WHERE r.event_id = e.id AND r.attendance_status = 'Attended'
        ) as attendance_count,
        (
          SELECT AVG(f.rating)
          FROM feedback f
          JOIN registration r ON f.registration_id = r.registration_id
          WHERE r.event_id = e.id
        ) as average_rating
      FROM dashboard d
      JOIN event e ON d.event_id = e.id
      WHERE d.dashboard_id = ${id}
    `;
```

- Join and aggregate queries:

```
export async function getDashboardData() {
  try {
    const dashboards = await prisma.$queryRaw<DashboardQueryResult[]>`
      SELECT
        e.id AS event_id,
        e.title AS event_title,
        e.date AS event_date,
        e.time AS event_time,
        e.description as event_description,
        COUNT(DISTINCT r.participant_id) AS total_registrants,
        d.attendance_count AS attended_count,
        (
          SELECT AVG(f.rating)
          FROM feedback f
          JOIN registration r ON f.registration_id = r.registration_id
          WHERE r.event_id = e.id
        ) as average_rating,
        d.dashboard_id
      FROM
        event e
      LEFT JOIN
        registration r ON e.id = r.event_id
      LEFT JOIN
        dashboard d ON e.id = d.event_id
      GROUP BY
        e.id, e.title, e.date, e.time, e.description, d.dashboard_id, d.attendance_count
      ORDER BY
        e.date ASC, e.time ASC
    `;
```

```
export async function getDashboardById(id: string)
  {try {
    const dashboard = await
prisma.
      $queryRaw<DashboardDetailQueryResult[]>`
      SELECT
        d.dashboard_id,
        e.id as
        event_id,e.title,
        e.date,
        e.time,
```

11. Code Snippets for invoking the Procedures / Functions:

- **Procedure:**
- **Function:**

```
export async function getAverageFeedback(eventId: string)
  {try {
    const [result] = await prisma.$queryRaw<[{ average:
      number }]>`SELECT get_average_feedback(${eventId}) as
      average
    `;
    return { data: result.average, error: null };
  } catch (error) {
    console.error("Error getting average feedback:", error);
    return { data: null, error: "Failed to get average feedback" };
  }
```

```
      e.description,
      get_total_registrants(e.id) as
      total_registrants,(
        SELECT
        COUNT(r.registration_id)FROM
        registration r
        WHERE r.event_id = e.id AND r.attendance_status = 'Attended'
      ) as
      attendance_count,(
        SELECT
        AVG(f.rating)FROM
        feedback f
        JOIN registration r ON f.registration_id =
        r.registration_idWHERE r.event_id = e.id
      ) as
    average_ratingFROM
    dashboard d
```