Database Management Systems Laboratory

Assignment 4

College Festival Management Web Application Development

Aritra Chakraborty
21CS10009

Bratin Mondal
21CS10016

Sarika Bishnoi
21CS10058

Anish Datta
21CS30006

Somya Kumar
21CS30050

Department of Computer Science and Engineering,
Indian Institute of Technology Kharagpur

Contents

1 Introduction

Our web-based system for managing university cultural festivals offers a seamless platform catering to participants, volunteers, organizers, and administrators. Through intuitive interfaces and robust backend infrastructure, it streamlines event registration, scheduling, volunteer management, and database administration, revolutionizing the festival experience.

2 Technologies Used

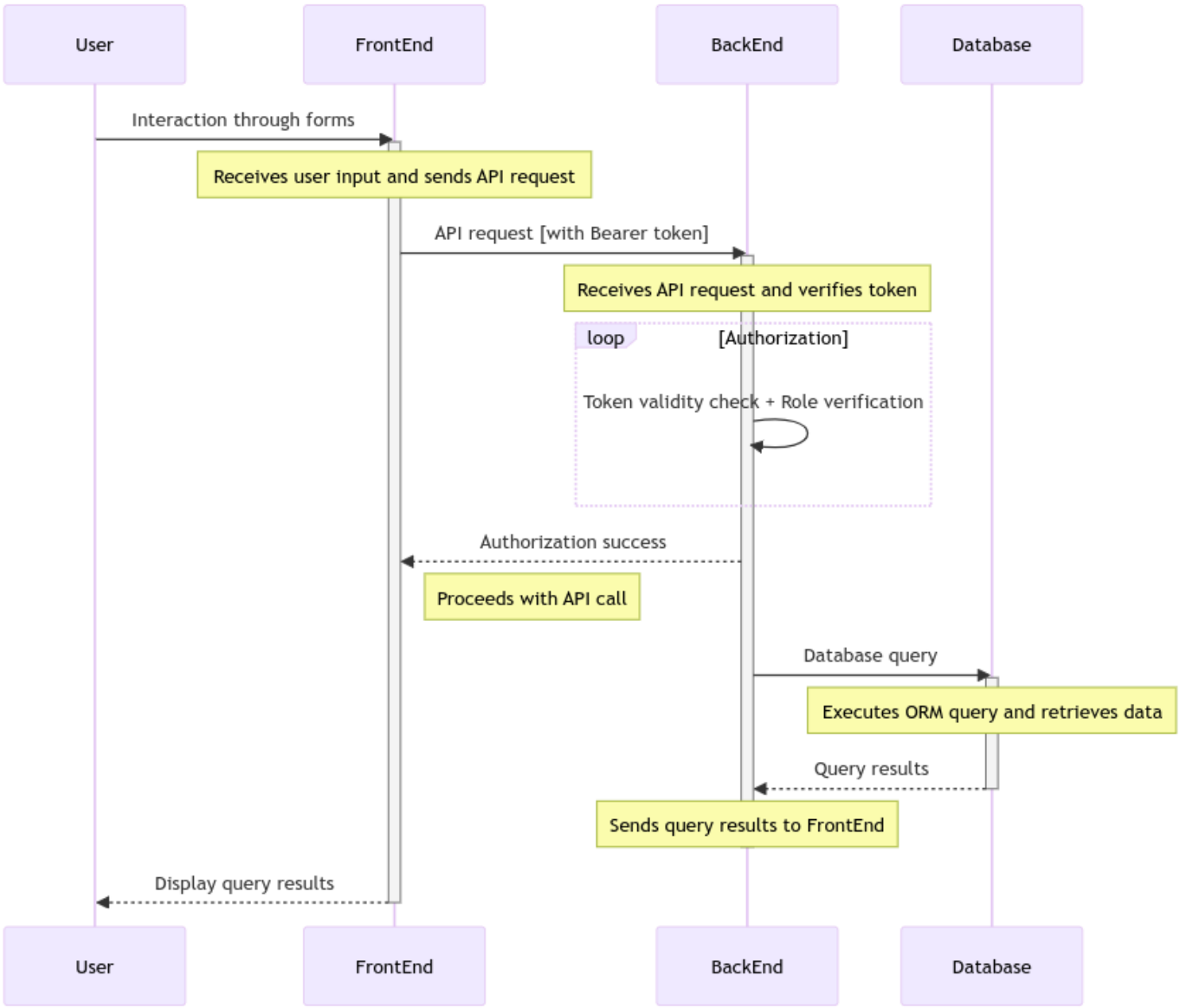

Figure 1: Illustration of the technologies used in the project

This figure is generated using [1].

2.1 Backend Frameworks

2.1.1 Flask for Framework

Flask is a lightweight web framework for Python used to build web applications. It is employed in our project as the backend framework to handle routing, request handling, and database operations efficiently. Additionally, Flask facilitates role-based backend implementation, enabling different types of users to have personalized access control. This role-based approach is significant as it enhances security by restricting access to sensitive information or functionalities based on user roles. Moreover, it improves user experience

by providing tailored content and features, ensuring that each user interacts with the application in a way that is relevant to their role or privileges.

2.1.2 JWT for Route Protection

JSON Web Tokens (JWT) are used in our project for route protection. JWT is a compact, URL-safe means of representing claims to be transferred between two parties. In our backend implementation with Flask, JWTs are issued to authenticated users upon successful login and are included in subsequent requests. These tokens are then verified on protected routes to ensure that only authenticated users with valid tokens can access specific endpoints. By employing JWT for route protection, we enhance the security of our application by preventing unauthorized access to sensitive resources, thus safeguarding the integrity of our data and ensuring a secure user experience.

2.1.3 Password Storing and Validation using One-way Hashing with bcrypt

In our project, passwords are securely stored and validated using one-way hashing with bcrypt. When a user creates an account or updates their password, bcrypt is employed to hash the password before storing it in the database. This hashing process converts the password into an irreversible string of characters, making it computationally infeasible to reverse engineer the original password. During the login process, bcrypt is used to hash the provided password, and the resulting hash is compared with the stored hash in the database. This ensures that passwords are never stored in plaintext and provides robust protection against potential data breaches. By utilizing bcrypt for password hashing and validation, we prioritize the security of user credentials and uphold best practices for safeguarding sensitive information.

2.1.4 Flask-Mail for Sending Emails

Flask-Mail is utilized in our project for sending emails. It is an extension for Flask that simplifies the process of sending emails from your application. With Flask-Mail, we can easily configure our email settings, such as SMTP server details, sender address, and authentication credentials. This allows us to seamlessly integrate email functionality into our web application, enabling features like email verification, password reset, and notifications. By leveraging Flask-Mail, we ensure reliable email delivery and enhance user engagement by keeping users informed through timely and relevant email communication.

2.2 Frontend Frameworks

2.2.1 React as Frontend Framework

React is a JavaScript library for building user interfaces, primarily for single-page applications. In our project, React serves as the frontend framework responsible for managing the user interface and handling dynamic content rendering. With React's component-based architecture, we can create modular and reusable UI components, enhancing code maintainability and scalability. React also provides a virtual DOM, which efficiently updates and re-renders components, resulting in improved performance and a smoother user experience. By utilizing React, we can develop interactive and responsive web applications that deliver a seamless user interface and rich user interactions.

2.2.2 Form Validation and Responsive Design

We implement form validation for user inputs, ensuring data integrity and improving user experience. Additionally, our frontend design is responsive, adapting to different screen sizes and devices, providing a consistent experience across platforms.

2.2.3 Tailwind CSS for Styling

Tailwind CSS is used for styling in our project, offering utility-first CSS-in-JS approach. It allows us to rapidly build custom designs with minimal CSS code, facilitating quick iteration and consistent styling throughout the application.

2.2.4 Notifications and Loading States

We utilize toast notifications to provide feedback to users, such as success messages or error alerts. Additionally, loading states are incorporated into dashboards and buttons to indicate when fetch requests are in progress, enhancing user understanding and interaction.

2.2.5 Persistence and Rerouting

Persistent storage is ensured by saving tab data in local storage, enabling users to retrieve their data even after reloading the page. Furthermore, rerouting is implemented based on the availability of the access token of logged-in users, enhancing security and usability.

3 Database Design Schema

We now present the database schema for our system. The database consists of the following tables:

3.1 Event

3.1.1 Schema

Table 1: Event Table

Attribute Name	Attribute Data Type	Constraints
id	varchar(100)	PRIMARY KEY, NOT NULL
name	varchar(100)	NOT NULL, UNIQUE
type	varchar(100)	NOT NULL, CHECK (type IN ('cultural', 'competition', 'workshop', 'talk', 'other'))
info	varchar(1000)	NOT NULL
start_date_time	timestamp	NOT NULL
end_date_time	timestamp	NOT NULL
location	varchar(100)	NOT NULL
first_prize	INTEGER	CHECK (first_prize $>$ 0)
second_prize	INTEGER	CHECK (second_prize $>$ 0 AND second_prize $<$ first_prize)
third_prize	INTEGER	CHECK (third_prize $>$ 0 AND third_prize $<$ second_prize)
created_at	timestamp	NOT NULL, DEFAULT CURRENT_TIMESTAMP

3.1.2 Additional Constraints

- The `prizes_null_constraint` ensures that for events of type 'competition', all three prize fields (`first_prize`, `second_prize`, and `third_prize`) must have non-null values. Alternatively, if the event is not of type 'competition', all three prize fields must be null.

3.2 Student

3.2.1 Schema

Table 2: Student Table

Attribute Name	Attribute Data Type	Constraints
sid	varchar(100)	PRIMARY KEY, NOT NULL
email	varchar(100)	NOT NULL, UNIQUE
name	varchar(100)	NOT NULL
roll_number	varchar(20)	UNIQUE
phone	varchar(20)	NOT NULL
college	varchar(100)	NOT NULL
department	varchar(100)	NOT NULL
year	INTEGER	NOT NULL, CHECK (year $>$ 0 AND year $\leq$ 5)
type	varchar(100)	NOT NULL, CHECK (type IN ('internal', 'external'))
password	varchar(100)	NOT NULL

3.2.2 Additional Constraints

- `valid_email`: Email format must adhere to standard email conventions.
- `phone`: Phone number must be exactly 10 digits long.

3.3 Organisers

3.3.1 Schema

Table 3: Organisers Table

Attribute Name	Attribute Data Type	Constraints
oid	varchar(100)	PRIMARY KEY, NOT NULL
email	varchar(100)	NOT NULL, UNIQUE
name	varchar(100)	NOT NULL
phone	varchar(20)	NOT NULL, UNIQUE
password	varchar(100)	NOT NULL

3.3.2 Additional Constraints

- `valid_email`: Email format must adhere to standard email conventions.
- `phone`: Phone number must be exactly 10 digits long.

3.4 Admin

3.4.1 Schema

Table 4: Admin Table

Attribute Name	Attribute Data Type	Constraints
id	varchar(100)	PRIMARY KEY, NOT NULL
name	varchar(100)	NOT NULL
email	varchar(100)	NOT NULL, UNIQUE
password	varchar(100)	NOT NULL

3.4.2 Additional Constraints

- `valid_email`: Email format must adhere to standard email conventions.

3.5 Accomodation

3.5.1 Schema

Table 5: Accommodation Table

Attribute Name	Attribute Data Type	Constraints
id	varchar(100)	PRIMARY KEY, NOT NULL
location	varchar(100)	NOT NULL
check_in	date	NOT NULL
check_out	date	NOT NULL
food_type	varchar(100)	NOT NULL, CHECK (food_type IN ('veg', 'non-veg'))
cost	INTEGER	NOT NULL, CHECK (cost $>$ 0)

3.5.2 Additional Constraints

- `check_in_out`: The check-in date must be before the check-out date.

3.5.3 Primary Keys

- (`event_id`, `logistics_id`)

3.5.4 Foreign Keys

- `event_id` references `event(id)` on **DELETE CASCADE** and **UPDATE CASCADE**.
- `logistics_id` references `event_logistics_item(logistics_id)` on **DELETE CASCADE** and **UPDATE CASCADE**.

3.6 Accomodated_at

3.6.1 Schema

Table 6: Accommodated At Table

Attribute Name	Attribute Data Type	Constraints
participant_id	varchar(100)	NOT NULL
logistics_id	varchar(100)	NOT NULL
payment_status	varchar(100)	DEFAULT 'paid' NOT NULL, CHECK (payment_status IN ('paid', 'pending'))

3.6.2 Primary Keys

- (`participant_id`, `logistics_id`)

3.6.3 Foreign Keys

- `participant_id` references `student(sid)` on **DELETE CASCADE** and **UPDATE CASCADE**.
- `logistics_id` references `accommodation(id)` on **DELETE CASCADE** and **UPDATE CASCADE**.

3.7 Participation

3.7.1 Schema

Table 7: Participation Table

Attribute Name	Attribute Data Type	Constraints
event_id	varchar(100)	NOT NULL
student_id	varchar(100)	NOT NULL

3.7.2 Primary Keys

- (`event_id`, `student_id`)

3.7.3 Foreign Keys

- `event_id` references `event(id)` on **DELETE CASCADE** and **UPDATE CASCADE**.
- `student_id` references `student(sid)` on **DELETE CASCADE** and **UPDATE CASCADE**.

3.8 Winners

3.8.1 Schema

Table 8: Winners Table

Attribute Name	Attribute Data Type	Constraints
event_id	varchar(100)	NOT NULL
first_p	varchar(100)	NOT NULL
second_p	varchar(100)	NOT NULL
third_p	varchar(100)	NOT NULL

3.8.2 Primary Keys

- `event_id`

3.8.3 Foreign Keys

- `event_id` references `event(id)` on **DELETE CASCADE** and **UPDATE CASCADE**.
- `first_p`, `second_p`, `third_p` references `student(sid)` on **DELETE CASCADE** and **UPDATE CASCADE**.

3.8.4 Additional Constraints

- `first_second_third`: First, second, and third positions must be different.

3.9 Manages

3.9.1 Schema

Table 9: Manages Table

Attribute Name	Attribute Data Type	Constraints
event_id	varchar(100)	NOT NULL
organiser_id	varchar(100)	NOT NULL
sponsorship_amount	INTEGER	NOT NULL, CHECK (sponsorship_amount $>$ 0)
payment_status	varchar(100)	DEFAULT 'pending' NOT NULL, CHECK (payment_status IN ('paid', 'pending'))
request_status	varchar(100)	DEFAULT 'pending' NOT NULL, CHECK (request_status IN ('approved', 'pending', 'rejected'))

3.9.2 Primary Keys

- (`event_id`, `organiser_id`)

3.9.3 Foreign Keys

- `event_id` references `event(id)` on **DELETE CASCADE** and **UPDATE CASCADE**.
- `organiser_id` references `organisers(oid)` on **DELETE CASCADE** and **UPDATE CASCADE**.

3.10 Volunteers

3.10.1 Schema

Table 10: Volunteers Table

Attribute Name	Attribute Data Type	Constraints
event_id	varchar(100)	NOT NULL
student_id	varchar(100)	NOT NULL
info	varchar(1000)	NOT NULL
role	varchar(100)	NOT NULL

3.10.2 Primary Keys

- (`event_id`, `student_id`)

3.10.3 Foreign Keys

- `event_id` references `event(id)` on **DELETE CASCADE** and **UPDATE CASCADE**.
- `student_id` references `student(sid)` on **DELETE CASCADE** and **UPDATE CASCADE**.

ER Diagram

Generated ER diagram for the above schema is shown in Figure 2 using [2].

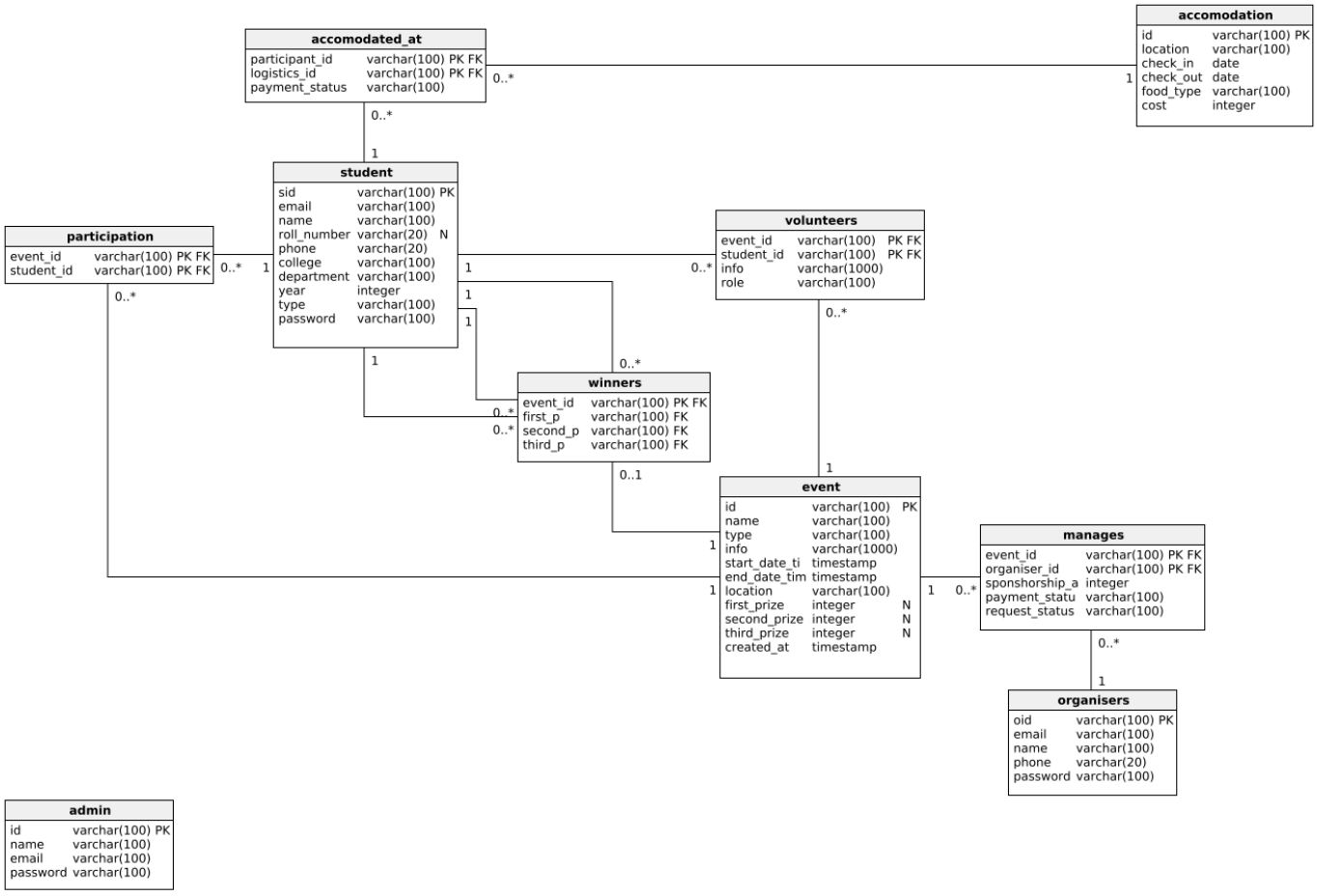

Figure 2: ER Diagram of the Relational Database Schema

4 Database Triggers

Triggers in the database enforce crucial constraints and automate processes, enhancing data integrity and consistency. They ensure that emails, roll numbers, and phone numbers adhere to specified formats, preventing data entry errors. Triggers also validate event timing, guaranteeing events start before they end, avoiding scheduling conflicts. Additionally, they synchronize participation records with winner announcements, maintaining accurate event outcomes. These triggers safeguard against invalid data and streamline database operations, contributing to robust and reliable data management.

4.1 validate_email_format

4.1.1 Importance

This trigger ensures that the email format is valid before inserting data into the ”student”, ”organisers”, and ”admin” tables. It helps maintain data integrity by enforcing a standard email format across these entities.

4.1.2 Procedure

The trigger is implemented as a PL/pgSQL function named ”validate_email_format”. It checks if the new email value (NEW.email) matches a regular expression pattern representing a valid email format. If the email format is valid, the trigger allows the insertion to proceed. Otherwise, it raises an exception indicating an invalid email format.

4.1.3 Invocation

The trigger ”validate_email_format” is invoked before inserting a new row into the ”student”, ”organisers”, and ”admin” tables. It executes the ”validate_email_format” function for each row, ensuring that the email

format is validated prior to insertion.

4.2 check_unique_roll_number

4.2.1 Importance

This trigger ensures that the roll number is unique for internal students before inserting data into the ”student” table. It prevents duplicate roll numbers among internal student entries, maintaining data consistency and integrity.

4.2.2 Procedure

The trigger is implemented as a PL/pgSQL function named ”check_unique_roll_number”. It checks if the new entry is of type ’internal’ and if a student with the same roll number already exists in the ”student” table. If a duplicate entry is found, the trigger raises an exception indicating that an internal student with the given roll number already exists.

4.2.3 Invocation

The trigger ”ensure_unique_roll_number” is invoked before inserting a new row into the ”student” table. It executes the ”check_unique_roll_number” function for each row, ensuring that the roll number uniqueness constraint is enforced for internal students.

4.3 validate_phone_format

4.3.1 Importance

This trigger ensures that the phone number format is correct before inserting data into the ”student” and ”organisers” tables. It validates phone numbers to adhere to a specific format, preventing invalid phone number entries.

4.3.2 Procedure

The trigger is implemented as a PL/pgSQL function named ”validate_phone_format”. It checks if the length of the new phone number (NEW.phone) is equal to 10, indicating a valid phone number format. If the format is incorrect, the trigger raises an exception indicating an invalid phone number format.

4.3.3 Invocation

The triggers ”validate_student_phone_format” and ”validate_organiser_phone_format” are invoked before inserting a new row into the ”student” and ”organisers” tables, respectively. They execute the ”validate_phone_format” function for each row, ensuring that the phone number format is validated prior to insertion.

4.4 check_event_time_constraints

4.4.1 Importance

This trigger ensures that the start date and time are before the end date and time before inserting data into the ”event” table. It enforces the temporal integrity constraint, preventing events from having an end date and time before the start date and time.

4.4.2 Procedure

The trigger is implemented as a PL/pgSQL function named ”check_event_time_constraints”. It compares the start date and time (NEW.start_date_time) with the end date and time (NEW.end_date_time) of the event. If the start date and time are not before the end date and time, the trigger raises an exception indicating that the start date and time must precede the end date and time.

4.4.3 Invocation

The trigger "enforce_event_time_constraints" is invoked before inserting a new row into the "event" table. It executes the "check_event_time_constraints" function for each row, ensuring that the event time constraints are enforced prior to insertion.

4.5 update_participation_on_winner_insert

4.5.1 Importance

This trigger updates the participation table when a new row is inserted into the winners table, ensuring that the position secured by participants reflects the winners' positions. It maintains accurate participation records and event outcomes.

4.5.2 Procedure

The trigger is implemented as a PL/pgSQL function named "update_participation_on_winner_insert". It updates the position secured column in the participation table based on the winners' positions in the winners table. If a participant wins a position, their participation record is updated accordingly.

4.5.3 Invocation

The trigger "update_participation_on_winner_insert_trigger" is invoked after inserting a new row into the winners table. It executes the "update_participation_on_winner_insert" function, ensuring that participation records are synchronized with winner announcements.

4.6 check_participation_before_winner_insert

4.6.1 Importance

This trigger ensures that there is a participation entry for the winner before inserting data into the "winners" table. It prevents inserting winner data without corresponding participation records, maintaining data consistency and integrity.

4.6.2 Procedure

The trigger is implemented as a PL/pgSQL function named "check_participation_before_winner_insert". It checks if there is a participation entry for the winner by querying the participation table based on the event ID and student IDs of the winners. If no participation entry is found for any winner, the trigger raises an exception indicating that no participation entry was found for the winner.

4.6.3 Invocation

The trigger "check_participation_before_winner_insert_trigger" is invoked before inserting a new row into the "winners" table. It executes the "check_participation_before_winner_insert" function for each row, ensuring that participation entries exist for all winners.

4.7 validate_sponsorship_amount

4.7.1 Importance

This trigger ensures that the sponsorship amount is greater than 0 before inserting data into the "manages" table. It validates sponsorship amounts to prevent inserting non-positive sponsorship values, ensuring accurate financial records.

4.7.2 Procedure

The trigger is implemented as a PL/pgSQL function named "validate_sponsorship_amount". It checks if the new sponsorship amount (NEW.sponshorship_amount) is greater than 0. If the amount is not greater than 0, the trigger raises an exception indicating that the sponsorship amount must be greater than 0.

4.7.3 Invocation

The trigger "validate_sponsorship_amount_trigger" is invoked before inserting a new row into the "manages" table. It executes the "validate_sponsorship_amount" function for each row, ensuring that sponsorship amounts are validated prior to insertion.

4.8 ensure_unique_event_name

4.8.1 Importance

This trigger ensures that the event name is unique before inserting data into the "event" table. It prevents inserting events with duplicate names, maintaining data consistency and integrity.

4.8.2 Procedure

The trigger is implemented as a PL/pgSQL function named "ensure_unique_event_name". It checks if there is any existing event with the same name as the new event being inserted. If a duplicate event name is found, the trigger raises an exception indicating that the event name must be unique.

4.8.3 Invocation

The trigger "ensure_unique_event_name_trigger" is invoked before inserting a new row into the "event" table. It executes the "ensure_unique_event_name" function for each row, ensuring that event names are unique prior to insertion.

4.9 check_winner_exists

4.9.1 Importance

This trigger prevents insertion into the "volunteers" and "participation" tables if there is already a winner entry for the corresponding event in the "winners" table. It ensures that volunteers and participants cannot be added to an event after winners have been declared, maintaining the integrity of event outcomes.

4.9.2 Procedure

The trigger is implemented as a PL/pgSQL function named "check_winner_exists". It checks if there are any entries in the "winners" table for the event specified in the new row being inserted into the "volunteers" or "participation" table. If a winner entry exists, the trigger raises an exception indicating that insertions into the "volunteers" or "participation" tables are not allowed for events with declared winners.

4.9.3 Invocation

The triggers "prevent_winner_insert" and "prevent_winner_insert_participation" are invoked before inserting a new row into the "volunteers" and "participation" tables, respectively. They execute the "check_winner_exists" function for each row, preventing insertions if winners have already been declared for the corresponding event.

4.10 update_manage_status

4.10.1 Importance

This trigger updates the request status of manages entries when a request is approved. It ensures that only one request can be approved for a given event, automatically rejecting other pending requests for the same event. This prevents conflicts and maintains consistency in event management.

4.10.2 Procedure

The trigger is implemented as a PL/pgSQL function named "update_manage_status". It checks if the request status has changed from pending to approved for a manages entry. If such a change occurs, the trigger updates the request status of all other manages entries for the same event to "rejected", ensuring that only one request is approved.

4.10.3 Invocation

The trigger "trigger_update_manage_status" is attached to the "manages" table and is invoked after an update operation on the table. It executes the "update_manage_status" function for each row, handling the update of request statuses based on the conditions specified in the trigger function.

4.11 update_rejected_status

4.11.1 Importance

This trigger updates the request status of manages entries when a previously approved request is deleted. It ensures that if an approved request is deleted, other rejected requests for the same event are updated to pending, allowing them to be reconsidered. This maintains consistency in request handling and event management.

4.11.2 Procedure

The trigger is implemented as a PL/pgSQL function named "update_rejected_status". It checks if the request status of the deleted entry was previously approved. If so, the trigger updates the request status of other manages entries for the same event that were previously rejected to pending, allowing them to be considered again.

4.11.3 Invocation

The trigger "trigger_update_rejected_status" is attached to the "manages" table and is invoked after a delete operation on the table. It executes the "update_rejected_status" function for each deleted row, handling the update of request statuses based on the conditions specified in the trigger function.

4.12 check_duplicate_request

4.12.1 Importance

This trigger prevents duplicate requests for managing the same event by automatically rejecting subsequent requests from different organisers if there is already an approved request for the event. It ensures that only one request is approved for managing an event at a time, maintaining clarity and avoiding conflicts in event management.

4.12.2 Procedure

The trigger is implemented as a PL/pgSQL function named "check_duplicate_request". It checks if there is already an approved request from a different organiser for the same event. If such a request exists, the trigger automatically updates the request status of the new request to rejected, preventing duplicate requests from being approved.

4.12.3 Invocation

The trigger "trigger_check_duplicate_request" is attached to the "manages" table and is invoked before an insert operation on the table. It executes the "check_duplicate_request" function for each new row being inserted, handling the validation of requests based on the conditions specified in the trigger function.

5 Queries: Database Operations

In this section, we provide a diverse collection of database queries commonly utilized in the backend of our system. While our presentation does not encompass all possible queries, it offers a representative sample of insert, delete, update, and select operations, showcasing the versatility of our database interactions.

The showcased queries encompass various aspects of our system's functionality. Insert queries are employed to add new data entries into the database, while delete queries ensure the removal of outdated or redundant records. Update queries play a crucial role in modifying existing data to reflect changes or corrections. Additionally, select queries enable the retrieval of targeted information to fulfill application requirements and user requests.

By presenting a selection of queries across different operations, we aim to illustrate the comprehensive capabilities of our database system. These queries represent integral components of our backend infrastructure, enabling seamless data management and efficient interaction with the application layer.

5.1 SQL Functions

In this subsection, we describe several SQL functions created to enhance the functionality and manageability of our database system. These functions are designed to perform specific tasks, manipulate data, or enforce constraints within the database environment. By encapsulating logic into reusable functions, we promote code modularity, simplify complex operations, and ensure consistency across database interactions.

The following SQL functions are integral components of our database infrastructure:

5.1.1 get_sponsored_events

Input

- `organiser_id_input` (char(20)): The ID of the organiser for whom sponsored events are retrieved.

Output

- `event_id` (varchar(100)): The ID of the event.
- `event_name` (varchar(100)): The name of the event.
- `event_type` (varchar(100)): The type of the event.
- `event_start` (timestamp): The start date and time of the event.
- `event_end` (timestamp): The end date and time of the event.
- `event_location` (varchar(100)): The location of the event.
- `sponsorship_amount` (integer): The sponsorship amount for the event.
- `payment_status` (varchar(100)): The payment status for the sponsorship.
- `request_status` (varchar(100)): The request status for the sponsorship.

Procedure This function retrieves details of events sponsored by a specific organiser based on the input organiser ID.

5.1.2 get_participated_events

Input

- `student_id_input` (char(20)): The ID of the student for whom participated events are retrieved.

Output

- `event_id` (varchar(100)): The ID of the event.
- `event_name` (varchar(100)): The name of the event.
- `event_type` (varchar(100)): The type of the event.
- `event_start` (timestamp): The start date and time of the event.
- `event_end` (timestamp): The end date and time of the event.
- `event_location` (varchar(100)): The location of the event.
- `position_secured` (varchar(100)): The position secured by the student in the event.

Procedure This function retrieves details of events participated by a specific student based on the input student ID.

5.1.3 get_winning_events

Input

- `student_name_input` (varchar(100)): The name of the student for whom winning events are retrieved.

Output

- `event_id` (varchar(100)): The ID of the event.
- `event_name` (varchar(100)): The name of the event.
- `position_secured` (varchar(100)): The position secured by the student in the event.

Procedure This function retrieves details of events won by a specific student based on the input student name.

5.1.4 get_event_volunteers

Input

- `event_id_input` (varchar(100)): The ID of the event for which volunteers are retrieved.

Output

- `student_id` (char(20)): The ID of the student volunteering for the event.
- `student_name` (varchar(100)): The name of the student volunteering for the event.
- `info` (varchar(1000)): Additional information provided by the volunteer.
- `role` (varchar(100)): The role of the volunteer in the event.

Procedure This function retrieves details of volunteers participating in a specific event based on the input event ID.

5.2 SQL Queries

We have provided a diverse collection of SQL queries commonly utilized in the backend of our system. These queries are representative of the operations performed on the database, including insert, delete, update, and select operations. The queries are designed to showcase the versatility of our database interactions and the comprehensive capabilities of our database system.

First we describe funtcions used for authentication and user management.

5.2.1 signup_student

5.2.1.1 Queries Used

```
SELECT * FROM STUDENT WHERE email='{email}';
SELECT * FROM ORGANISERS WHERE email='{email}';
INSERT INTO STUDENT VALUES ('{sid}', '{email}', '{data['name']}',
   '{data['roll_number']}', '{data['phone']}', '{data['college']}',
   '{data['department']}', {int(data['year'])}, '{data['type']}',
   '{hashed_password}');
```

Listing 1: SQL Queries

5.2.1.2 Usage

This function is used to sign up a student by inserting their details into the `STUDENT` table in the database after hashing their password for security.

5.2.2 signup_organiser

```
SELECT * FROM ORGANISERS WHERE email='{email}';
SELECT * FROM STUDENT WHERE email='{email}';
INSERT INTO ORGANISERS VALUES ('{oid}', '{email}', '{data['name']}',
   '{data['phone']}', '{hashed_password}');
```

Listing 2: SQL Queries

5.2.2.1 Usage

This function is used to sign up an organiser by inserting their details into the `ORGANISERS` table in the database after hashing their password for security.

5.2.3 login

5.2.3.1 Queries Used

```
SELECT * FROM STUDENT WHERE email='{email}';
SELECT * FROM ORGANISERS WHERE email='{email}';
SELECT * FROM ADMIN WHERE email='{email}';
```

Listing 3: SQL Queries

5.2.3.2 Usage

This function is used to log in a user (student, organiser, or admin) by checking their email and password against the records stored in the database. If the credentials are valid, an access token is created for authentication.

5.2.4 profile

5.2.4.1 Queries Used

```
SELECT * FROM STUDENT WHERE sid='{profile_info.get('sid', 0)}';
SELECT * FROM ORGANISERS WHERE oid='{profile_info.get('oid', 0)}';
SELECT * FROM ADMIN WHERE id='{profile_info.get('id', 0)}';
```

Listing 4: SQL Queries

5.2.4.2 Usage

This function retrieves the profile information of the user (student, organiser, or admin) based on the identity provided in the JWT token. The profile information includes details such as name, email, phone number, and role.

5.2.5 create_admin

5.2.5.1 Queries Used

```
SELECT * FROM ADMIN WHERE email='{admin_email}';
INSERT INTO ADMIN VALUES ('{admin_id}', '{admin_name}', '{admin_email}',
   '{hashed_password}');
```

Listing 5: SQL Queries

5.2.5.2 Usage

This function is used to create a new admin account by inserting the admin's details into the `ADMIN` table in the database after hashing their password for security.

5.2.6 forgot_password

5.2.6.1 Queries Used

```
SELECT * FROM STUDENT WHERE email='{email}';
UPDATE STUDENT SET password='{hashed_password}' WHERE email='{email}';
SELECT * FROM ORGANISERS WHERE email='{email}';
UPDATE ORGANISERS SET password='{hashed_password}' WHERE email='{email}';
```

Listing 6: SQL Queries

5.2.6.2 Usage

This function is used to reset the password of a user (student) who has forgotten their password. A new password is generated, hashed, and updated in the database. An email is sent to the user's email address with the new password.

5.2.7 get_accomodation

5.2.7.1 Queries Used

```
SELECT location, check_in, check_out, food_type, cost
FROM accomodated_at, accomodation
WHERE participant_id='{user_id}' AND accomodated_at.logistics_id =
   accomodation.id;
```

Listing 7: SQL Queries

5.2.7.2 Usage

This function retrieves the accommodation details for a user based on their JWT token. It fetches the location, check-in and check-out dates, food type, and payment amount from the database.

5.2.8 book_accomodation

5.2.8.1 Queries Used

```
SELECT * FROM accomodated_at WHERE participant_id='{user_id}';
SELECT * FROM accomodation WHERE location='{data['location']}' AND check_in
   = '{data['from']}' AND check_out = '{data['to']}' AND food_type =
   '{data['food_type']}';
INSERT INTO accomodation VALUES ('{logistics_id}', '{data['location']}',
   '{data['from']}', '{data['to']}', '{data['food_type']}',
   {int(data['payment'])});
INSERT INTO accomodated_at VALUES ('{user_id}', '{logistics_id}',
   'pending');
```

Listing 8: SQL Queries

5.2.8.2 Usage

This function allows a user to book accommodation by inserting the booking details into the database. It checks if the user already has accommodation booked and inserts the booking details into the `accomodated_at` and `accomodation` tables if the accommodation is available.

5.2.9 all_students

5.2.9.1 Queries Used

```
SELECT * FROM STUDENT;
```

Listing 9: SQL Queries

5.2.9.2 Usage

This function retrieves all student details from the database. It is accessible only to users with admin privileges.

5.2.10 remove_student

5.2.10.1 Queries Used

```
DELETE FROM STUDENT WHERE sid='{id}';
```

Listing 10: SQL Queries

5.2.10.2 Usage

This function removes a student from the database based on the provided student ID. It is accessible only to users with admin privileges.

5.2.11 add_student

5.2.11.1 Queries Used

```
SELECT * FROM STUDENT WHERE email='{email}';
SELECT * FROM ORGANISERS WHERE email='{email}';
INSERT INTO STUDENT VALUES ('{sid}', '{email}', '{data['name']}',
   '{data['roll_number']}', '{data['phone']}', '{data['college']}',
   '{data['department']}', {int(data['year'])}, '{data['type']}',
   '{hashed_password}');
```

Listing 11: SQL Queries

5.2.11.2 Usage

This function adds a new student to the database. It checks if the student or an organiser with the same email already exists in the database before insertion.

5.2.12 update_student

5.2.12.1 Queries Used

```
SELECT * FROM STUDENT WHERE sid='{data['sid']}';
UPDATE STUDENT SET <field1>='<value1>', <field2>='<value2>', ... WHERE
   sid='{data['sid']}';
```

Listing 12: SQL Queries

5.2.12.2 Usage

This function updates the details of a student in the database based on the provided student ID. It constructs the SQL query dynamically based on the available fields provided in the request JSON.

5.2.13 all_events

5.2.13.1 Queries Used

```
SELECT * FROM EVENT;
```

Listing 13: SQL Queries

5.2.13.2 Usage

This function retrieves all events from the database. It is accessible only to users with admin privileges.

5.2.14 add_event

5.2.14.1 Queries Used

```
SELECT * FROM EVENT WHERE name='{data['name']}';
INSERT INTO EVENT VALUES ('{event_id}', '{data['name']}', '{data['type']}',
   '{data['info']}', '{data['start_date_time']}',
   '{data['end_date_time']}', '{data['location']}',
   '{data['first_prize']}', '{data['second_prize']}',
   '{data['third_prize']}');
```

Listing 14: SQL Queries

5.2.14.2 Usage

This function adds a new event to the database. It checks if the event already exists before insertion.

5.2.15 delete_event

5.2.15.1 Queries Used

```
DELETE FROM EVENT WHERE id='{id}';
```

Listing 15: SQL Queries

5.2.15.2 Usage

This function deletes an event from the database based on the provided event ID. It is accessible only to users with admin privileges.

5.2.16 update_event

5.2.16.1 Queries Used

```
UPDATE EVENT SET <field1>='<value1>', <field2>='<value2>', ... WHERE
   id='{id}';
```

Listing 16: SQL Queries

5.2.16.2 Usage

This function updates the details of an event in the database based on the provided event ID. It constructs the SQL query dynamically based on the available fields provided in the request JSON. For non-competition events, it allows updating all fields including null values. For competition events, it does not allow null values.

5.2.17 all_organisers

5.2.17.1 Queries Used

```
SELECT * FROM ORGANISERS;
SELECT * FROM MANAGES WHERE organiser_id='{oid}';
SELECT * FROM EVENT WHERE id='{event_id}';
```

Listing 17: SQL Queries

5.2.17.2 Usage

This function retrieves all organisers from the database along with the events they sponsor. It is accessible only to users with admin privileges.

5.2.18 remove_organiser

5.2.18.1 Queries Used

```
SELECT * FROM ORGANISERS WHERE oid='{id}';
DELETE FROM ORGANISERS WHERE oid='{id}';
```

Listing 18: SQL Queries

5.2.18.2 Usage

This function deletes an organiser from the database based on the provided organiser ID. It is accessible only to users with admin privileges.

5.2.19 add_organiser

5.2.19.1 Queries Used

```
SELECT * FROM ORGANISERS WHERE email='{email}';
SELECT * FROM STUDENT WHERE email='{email}';
INSERT INTO ORGANISERS VALUES ('{oid}', '{email}', '{data['name']}',
   '{data['phone']}', '{hashed_password}');
```

Listing 19: SQL Queries

5.2.19.2 Usage

This function adds a new organiser to the database. It checks if the organiser or user already exists before insertion.

5.2.20 update_organiser

5.2.20.1 Queries Used

```
SELECT * FROM ORGANISERS WHERE oid='{data['oid']}';
```

Listing 20: SQL Queries

5.2.20.2 Usage

This function updates the details of an existing organiser in the database based on the provided organiser ID. It is accessible only to users with admin privileges.

5.2.21 all_notifs

```
SELECT event_id, organiser_id, ORGANISERS.name, ORGANISERS.email,
   sponsorship_amount, EVENT.name
FROM EVENT, MANAGES, ORGANISERS
WHERE organiser_id=oid AND event_id=id AND request_status='pending';
```

Listing 21: SQL Queries

5.2.21.1 Usage

This function retrieves all pending notifications for event sponsorships from the database. It is accessible only to users with admin privileges.

5.2.22 approve_organiser

```
SELECT * FROM MANAGES WHERE organiser_id='{oid}' AND event_id='{event_id}';
UPDATE MANAGES SET request_status='approved' WHERE organiser_id='{oid}' AND
   event_id='{event_id}';
```

Listing 22: SQL Queries

5.2.22.1 Usage

This function approves an organiser for sponsoring an event. It updates the request status to 'approved' and sends an email notification to the organiser.

5.2.23 reject_organiser

```
SELECT * FROM MANAGES WHERE organiser_id='{oid}' AND event_id='{event_id}';
UPDATE MANAGES SET request_status='rejected' WHERE organiser_id='{oid}' AND
   event_id='{event_id}';
```

Listing 23: SQL Queries

5.2.23.1 Usage

This function rejects an organiser for sponsoring an event. It updates the request status to 'rejected'.

Next we describe functions used for event management.

5.2.24 get_all_events

5.2.24.1 Queries Used

```
SELECT id, name, type, start_date_time, end_date_time FROM EVENT;
SELECT * FROM PARTICIPATION WHERE student_id='{sid}' AND event_id='{eid}';
SELECT * FROM VOLUNTEERS WHERE student_id='{sid}' AND event_id='{eid}';
SELECT request_status FROM MANAGES WHERE organiser_id='{oid}' AND
   event_id='{eid}';
```

Listing 24: SQL Queries

5.2.24.2 Usage

This function retrieves all events from the database along with additional information such as whether a student is registered or volunteered for an event, or whether an organiser sponsors an event. It is accessible only to authenticated users.

5.2.25 get_an_event

5.2.25.1 Queries Used

```
SELECT * FROM EVENT LEFT OUTER JOIN WINNERS ON id=event_id WHERE
   id='{event_id}';
SELECT * FROM PARTICIPATION WHERE student_id='{sid}' AND event_id='{eid}';
SELECT * FROM VOLUNTEERS WHERE student_id='{sid}' AND event_id='{eid}';
SELECT name, email FROM STUDENT WHERE sid='{f\_prize}';
```

Listing 25: SQL Queries

5.2.25.2 Usage

This function retrieves details of a specific event from the database, including winners if applicable. It checks whether the requesting user is registered or volunteered for the event and provides additional information accordingly.

5.2.26 register_student

5.2.26.1 Queries Used

```
SELECT * FROM VOLUNTEERS WHERE student_id='{sid}' AND event_id='{event_id}';
SELECT * FROM PARTICIPATION WHERE student_id='{sid}' AND
   event_id='{event_id}';
INSERT INTO PARTICIPATION VALUES ('{event_id}', '{sid}');
```

Listing 26: SQL Queries

5.2.26.2 Usage

This function allows a student to register for an event. It checks if the student is already registered or volunteered for the event and handles registration accordingly.

5.2.27 volunteer_student

5.2.27.1 Queries Used

```
SELECT * FROM PARTICIPATION WHERE student_id='{sid}' AND
   event_id='{event_id}';
SELECT * FROM VOLUNTEERS WHERE student_id='{sid}' AND event_id='{event_id}';
INSERT INTO VOLUNTEERS VALUES ('{event_id}', '{sid}', '{info}', '{role}');
```

Listing 27: SQL Queries

5.2.27.2 Usage

This function allows a student to volunteer for an event. It checks if the student is already registered or volunteered for the event and handles volunteering accordingly.

5.2.28 sponsor

5.2.28.1 Queries Used

```
SELECT * FROM MANAGES WHERE organiser_id='{oid}' AND event_id='{event_id}';
INSERT INTO MANAGES VALUES ('{event_id}', '{oid}', '{sponsorship_amount}');
```

Listing 28: SQL Queries

5.2.28.2 Usage

This function allows an organiser to apply for sponsoring an event. It checks if the organiser has already applied for sponsoring the event and handles sponsorship accordingly.

Next we describe functions used for student management.

5.2.29 edit_student

5.2.29.1 Queries Used

```
SELECT * FROM STUDENT WHERE sid='{sid}';
UPDATE STUDENT SET name='{name}', roll_number='{roll_number}',
   phone='{phone}', college='{college}', department='{department}',
   year='{year}', password='{hashed_password}' WHERE sid='{sid}';
```

Listing 29: SQL Queries

5.2.29.2 Usage

This function allows a student to edit their profile information. It checks if the student exists and updates the provided fields accordingly.

5.2.30 delete_student

5.2.30.1 Queries Used

```
SELECT password FROM STUDENT WHERE sid='{sid}';
DELETE FROM STUDENT WHERE sid='{sid}';
```

Listing 30: SQL Queries

5.2.30.2 Usage

This function allows a student to delete their account. It checks if the student exists and verifies the provided password before deleting the account.

Next we describe functions used for organiser

5.2.31 edit_organiser

5.2.31.1 Queries Used

```
SELECT * FROM ORGANISERS WHERE oid='{oid}';
UPDATE ORGANISERS SET name='{name}', phone='{phone}',
   password='{hashed_password}' WHERE oid='{oid}';
```

Listing 31: SQL Queries

5.2.31.2 Usage

This function allows an organiser to edit their profile information. It checks if the organiser exists and updates the provided fields accordingly.

5.2.32 delete_organiser

5.2.32.1 Queries Used

```
SELECT password FROM ORGANISERS WHERE oid='{oid}';
DELETE FROM ORGANISERS WHERE oid='{oid}';
```

Listing 32: SQL Queries

5.2.32.2 Usage

This function allows an organiser to delete their account. It checks if the organiser exists and verifies the provided password before deleting the account.

5.2.33 get_an_event

5.2.33.1 Queries Used

```
SELECT * FROM EVENT LEFT OUTER JOIN WINNERS ON id=event_id WHERE
   id='{event_id}';
SELECT request_status FROM MANAGES WHERE organiser_id='{oid}' AND
   event_id='{eid}';
SELECT DISTINCT student_id FROM PARTICIPATION WHERE event_id='{eid}';
SELECT logistics_id, quantity FROM EVENT_LOGISTICS WHERE event_id='{eid}';
SELECT item_name, item_price FROM EVENT_LOGISTICS_ITEM WHERE
   logistics_id='{lid}';
SELECT student_id, info, role FROM VOLUNTEERS WHERE event_id='{eid}';
```

Listing 33: SQL Queries

5.2.33.2 Usage

This function retrieves detailed information about a specific event organized by an organiser. It returns details such as event participants, logistics, volunteers, and winners.

5.2.34 get_student_profile

5.2.34.1 Queries Used

```
SELECT * FROM STUDENT WHERE sid='{student_id}';
```

Listing 34: SQL Queries

5.2.34.2 Usage

This function retrieves the profile information of a specific student by their ID. It returns details such as email, name, roll number, phone, college, department, year, and type.

5.2.35 set_winners

5.2.35.1 Queries Used

```
SELECT * FROM WINNERS WHERE event_id='{event_id}';
INSERT INTO WINNERS VALUES ('{event_id}', '{sid1}', '{sid2}', '{sid3}');
SELECT name FROM EVENT WHERE id='{event_id}';
SELECT name, email FROM STUDENT WHERE sid='{request.get_json()['sid']}';
```

Listing 35: SQL Queries

5.2.35.2 Usage

This function sets the winners for an event. It checks if the winners have already been set. If not, it inserts the winners into the database. It also sends congratulatory emails to the winners.

6 Functional Dependencies

6.1 Event Table

id → name, type, info, start_date_time, end_date_time,
location, first_prize, second_prize, third_prize, created_at
name → id, type, info, start_date_time, end_date_time,
location, first_prize, second_prize, third_prize, created_at

6.2 Student Table

sid → email, name, roll_number, phone, college, department, year, type, password
email → sid, name, roll_number, phone, college, department, year, type, password
roll_number → sid, email, name, phone, college, department, year, type, password
phone → sid, email, name, roll_number, college, department, year, type, password

6.3 Organisers Table

oid → email, name, phone, password
email → oid, name, phone, password
phone → oid, email, name, password

6.4 Admin Table

$$\text{id} \rightarrow \text{name, email, password}$$
$$\text{email} \rightarrow \text{id, name, password}$$

6.5 Accomodation Table

$$\text{id} \rightarrow \text{location, check\_in, check\_out, food\_type, cost}$$

6.6 Accomodated at Table

$$\text{participant\_id, logistics\_id} \rightarrow \text{payment\_status}$$

6.7 Winners Table

$$\text{event\_id} \rightarrow \text{first\_p, second\_p, third\_p}$$

6.8 Manages Table

$$\text{event\_id, organiser\_id} \rightarrow \text{sponsorship\_amount, payment\_status, request\_status}$$

6.9 Volunteers Table

$$\text{event\_id, student\_id} \rightarrow \text{info, role}$$

7 Table Form

7.1 Database Schema Overview

The database schema consists of multiple tables designed to store information about events, participants, organisers, winners, volunteers, accommodations, and administrators. Each table serves a specific purpose and contributes to the overall functionality of the event management system.

7.2 BCNF and 3NF Tables

BCNF (Boyce-Codd Normal Form) and 3NF (Third Normal Form) are important principles in database design to ensure data integrity, minimize redundancy, and reduce anomalies during database operations.

7.2.1 Importance of BCNF and 3NF

- **Data Integrity**: BCNF and 3NF help maintain data integrity by reducing redundancy and ensuring that each piece of information is stored in only one place, preventing inconsistencies and anomalies.
- **Minimized Redundancy**: By eliminating redundant data, BCNF and 3NF reduce storage space and improve query performance by minimizing the need for repetitive data storage and updates.
- **Normalization**: These normalization forms help organize data into logical and efficient structures, making it easier to manage and query the database.

7.2.2 Principles in the Database Schema

The database schema follows the principles of BCNF and 3NF to a certain extent, ensuring data integrity and minimizing redundancy. However, some redundancy may exist in certain tables to simplify queries and reduce the number of joins required for data retrieval.

7.2.3 Redundancy for Query Optimization

In some cases, redundancy is intentionally introduced in the schema to improve query performance and simplify data retrieval. For example, denormalization techniques may be applied to store redundant data in tables to avoid complex joins and optimize query execution time.

Overall, while BCNF and 3NF are essential principles in database design, practical considerations such as query optimization and performance may sometimes justify deviating from strict normalization rules.

8 Frontend Forms

8.1 App

8.1.1 Forms Used

- Login form (Route: /login) - Used for user authentication.
- Signup form for guest students (Route: /signup/guest-student) - Used for registering guest students.
- Signup form for native students (Route: /signup/native-student) - Used for registering native students.
- Signup form for organizers (Route: /signup/organizer) - Used for registering organizers.

8.1.2 Features

- Routing using React Router (BrowserRouter used from react-router-dom) - Facilitates navigation between different pages of the application.
- Styling with Mantine CSS (Importing MantineProvider and associated styles) - Provides consistent and visually appealing styling for the application components.
- Toast notifications using Sonner (Toaster component imported from ”sonner”) - Provides feedback to the user about the outcome of their actions, such as successful login or signup.

8.2 Dashboard

8.2.1 Forms Used

- Login form (Route: /login) - Used for user authentication.

8.2.2 Features

- AppShell component from ”@mantine/core” - Provides a layout structure for the dashboard page.
- Burger component from ”@mantine/core” - Allows toggling the navigation menu on small screens.
- Logo component - Displays the logo of the application.
- Various icons imported from ”lucide-react” - Used for visual representation of different features.
- useDisclosure hook from ”@mantine/hooks” - Manages the state of a disclosure component.
- useEffect and useState hooks from React - Used for managing component lifecycle and state.
- Schedule, Accommodation, Events, Profile, Event, AdminAccommodations, AdminStudents, AdminEvents, AdminOrganisers, AdminNotif components - Represent different sections of the dashboard page.
- Navigate component from ”react-router-dom” - Allows navigating to different routes programmatically.
- toast function from ”sonner” - Displays toast notifications for user feedback.
- TailSpin component from ”react-loader-spinner” - Displays a loading spinner while fetching data.

8.3 Landing

8.3.1 Forms Used

- Signup form (Link to: /signup) - Used for user registration.
- Login form (Link to: /login) - Used for user authentication.

8.3.2 Features

- Drama icon from ”lucide-react” - Represents a visual element on the landing page.
- Logo component - Displays the logo of the application.
- Link component from ”react-router-dom” - Allows navigation to different routes.
- TypeAnimation component from ”react-type-animation” - Animates text to display a sequence of strings.
- useEffect and useState hooks from React - Used for managing component lifecycle and state.

8.4 Login

8.4.1 Forms Used

- Login form - Used for user authentication.
- Forgot Password form - Used for resetting the password.

8.4.2 Features

- useForm hook from ”@mantine/form” - Manages form state and validation.
- TextInput and Modal components from ”@mantine/core” - Used for input fields and displaying a modal for the ”Forgot Password” feature.
- Button component - Represents a button element with loading state support.
- Link component from ”react-router-dom” - Allows navigation to different routes.
- Logo component - Displays the logo of the application.
- toast function from ”sonner” - Displays toast notifications for user feedback.
- useState hook from React - Used for managing component state.
- useEffect hook from React - Used for performing side effects such as fetching data.
- TailSpin component from ”react-loader-spinner” - Displays a loading spinner while fetching data.
- useDisclosure hook from ”@mantine/hooks” - Manages the state of a disclosure component.

8.5 Signup

8.5.1 Forms Used

- Signup form for different user types:
  - Guest Student - Sign up for participants not from IIT KGP.
  - Native Student - Sign up for students of IIT KGP.
  - Organizer - Sign up for event organizers.

8.5.2 Features

- Link component from ”react-router-dom” - Allows navigation to different routes.
- Conditional rendering based on whether the user is logged in or not.

8.6 Signup Organizer

8.6.1 Forms Used

- Organizer signup form - Allows organizers to create an account.

8.6.2 Features

- useForm hook from "@mantine/form" - Manages form state and validation.
- TextInput component from "@mantine/core" - Used for input fields.
- Link component from "react-router-dom" - Allows navigation to different routes.
- Button component - Represents a button element with loading state support.
- useState hook from React - Used for managing component state.
- useEffect hook from React - Used for performing side effects such as fetching data.

8.7 Signup Student

8.7.1 Forms Used

- Student signup form - Allows students to create an account with different steps for entering details.

8.7.2 Features

- useForm hook from "@mantine/form" - Manages form state and validation.
- TextInput, Stepper, and Select components from "@mantine/core" - Used for input fields, stepper for multi-step form, and select dropdowns, respectively.
- Link component from "react-router-dom" - Allows navigation to different routes.
- Button component - Represents a button element with loading state support.
- useState and useEffect hooks from React - Used for managing component state and performing side effects such as fetching data, respectively.

8.8 Admin Events

8.8.1 Components Used

- Input, Modal, NumberInput, Select, Table, TextInput, Textarea, and Tooltip components from "@mantine/core" - Various UI components for user input, modal dialogs, tables, and tooltips.
- DateInput, DateTimePicker, and DatesProvider components from "@mantine/dates" - Components for date and date-time input with date selection functionality.
- useForm hook from "@mantine/form" - Manages form state and validation.
- useDisclosure hook from "@mantine/hooks" - Manages the state of modal dialogs.
- Info, Pen, Plus, and Trash2 icons from "lucide-react" - Icon components representing information, edit, add, and delete actions, respectively.
- useState and useEffect hooks from React - Used for managing component state and performing side effects such as fetching data, respectively.
- format and set functions from "date-fns" - Used for formatting date objects and setting specific parts of the date, respectively.
- toast function from "sonner" - Used for displaying toast notifications.

8.8.2 Features

- Displaying a list of admin events with options to view, edit, and delete each event.
- Modal dialogs for adding new events, updating existing events, and confirming event deletion.
- Form for adding and updating events with validation and date-time selection.
- Fetching existing events from the server and updating the list accordingly.
- Deleting events with confirmation dialog and updating the list after deletion.

8.9 Admin Notifications

8.9.1 Components Used

- IndianRupee icon from "lucide-react" - Represents the currency symbol for Indian Rupee.
- useEffect and useState hooks from React - Used for performing side effects and managing component state, respectively.
- toast function from "sonner" - Utilized for displaying toast notifications.
- Button component from "./button" - A custom button component for triggering actions.

8.9.2 Features

- Fetches notifications from the server upon component mounting and updates the state accordingly.
- Displays a list of notifications, each containing details such as the organizer's name, email, event name, and sponsorship amount.
- Provides options to approve or reject each notification.
- Sends requests to the server to approve or reject notifications, updating the UI and displaying toast messages upon success.

8.10 Admin Organisers

8.10.1 Components Used

- Input, Modal, Select, Table, TextInput, Tooltip from "@mantine/core" - Various UI components for input, modal dialogs, tables, and text input.
- DateInput from "@mantine/dates" - Component for selecting dates.
- useForm hook from "@mantine/form" - Hook for managing form state and validation.
- useDisclosure hook from "@mantine/hooks" - Hook for managing modal state.
- Info, Pen, Plus, Trash2 from "lucide-react" - Icons for displaying information, editing, adding, and deleting.
- useEffect, useState hooks from React - Used for performing side effects and managing component state, respectively.
- toast function from "sonner" - Utilized for displaying toast notifications.
- Button component from "./button" - A custom button component for triggering actions.

8.10.2 Features

- Fetches organiser data from the server and updates the state when the component mounts or when the search query changes.
- Displays a list of organisers in a table, each with options to view sponsored events, update, or delete.
- Provides modals for adding new organisers, confirming deletion, and viewing sponsored events for a particular organiser.

- Implements form validation for email, name, phone number, and password fields when adding or updating organisers.
- Sends requests to the server to add, update, or delete organisers, updating the UI and displaying toast messages upon success.

8.11 Admin Students

8.11.1 Components Used

- Input, Modal, NumberInput, Select, Table, TextInput, Tooltip from "@mantine/core" - Various UI components for input, modal dialogs, tables, and text input.
- DateInput from "@mantine/dates" - Component for selecting dates.
- useForm hook from "@mantine/form" - Hook for managing form state and validation.
- useDisclosure hook from "@mantine/hooks" - Hook for managing modal state.
- Info, Pen, Plus, Trash2 from "lucide-react" - Icons for displaying information, editing, adding, and deleting.
- useEffect, useState hooks from React - Used for performing side effects and managing component state, respectively.
- toast function from "sonner" - Utilized for displaying toast notifications.
- Button component from "./button" - A custom button component for triggering actions.

8.11.2 Features

- Fetches student data from the server and updates the state when the component mounts or when the search query changes.
- Displays a list of students in a table, each with options to update or delete.
- Provides modals for adding new students and confirming deletion.
- Implements form validation for email, name, password, roll number, phone number, college, department, year, and student type fields when adding or updating students.
- Sends requests to the server to add or update students, updating the UI and displaying toast messages upon success.

8.12 Event

8.12.1 Forms Used

- `Input`: General input field.
- `Modal`: Modal dialog for confirmation and input.
- `NumberInput`: Input field for numeric values.
- `Pill`: Stylish pill-shaped component.
- `Radio`: Radio button for selecting options.
- `Select`: Dropdown select component.
- `Table`: Table component for displaying data.
- `Textarea`: Textarea input for multiline text.

8.12.2 Features

- Event Registration: Users can register for events.
- Volunteering: Users can volunteer for events.
- Sponsorship: Users can sponsor events.

- Winner Declaration: Organizers can declare winners of events.
- Real-time Clock: The component updates the current date and time every second.
- Form Validation: Forms have validation for required fields.
- API Integration: The component interacts with backend APIs for various functionalities.
- User Feedback: Toast notifications are displayed to provide feedback to the user.

8.13 Events

8.13.1 Forms Used

- `Card`: Component for displaying event details.
- `Button`: Button component for user interaction.
- `TextInput`: Input field for searching events.

8.13.2 Features

- Event Filtering: Users can filter events by name or category.
- Date Formatting: Dates and times are formatted for display.
- Dynamic Rendering: Event details are dynamically rendered based on search criteria.
- Link Navigation: Users can navigate to event details page for more information.
- Local Storage Interaction: The component interacts with local storage to store tab and event ID.

8.14 Profile

8.14.1 Forms Used

- `Input`: General input component for text input.
- `Modal`: Component for displaying modals.
- `NumberInput`: Input component for numerical input.
- `Select`: Dropdown select component.
- `Table`: Component for displaying tabular data.
- `TextInput`: Input component for text input.
- `Tooltip`: Component for displaying tooltips.
- `DateInput`: Input component for date selection.

8.14.2 Features

- Data Fetching: Retrieves user profile data for display.
- Data Editing: Allows users to edit their profile information.
- Data Validation: Validates user input data before submission.
- Modal Interaction: Utilizes modals for displaying and editing profile information.
- Form Submission: Handles form submission for updating profile information.
- Account Deletion: Allows users to delete their account with password confirmation.
- Error Handling: Displays error messages for failed operations.

8.15 Schedule

8.15.1 Forms Used

- `Input`: General input component for text input.
- `Select`: Dropdown select component.
- `Table`: Component for displaying tabular data.
- `TextInput`: Input component for text input.

8.15.2 Features

- Data Fetching: Retrieves event data for display.
- Sorting: Allows sorting events based on start date-time, end date-time, or duration.
- Filtering: Filters events based on event name.
- Date Formatting: Formats date-time data for display.
- Conditional Rendering: Renders different columns based on user type (student or organiser).

9 Roles and Access Controls

Roles and access controls are essential for managing user privileges in a database system. Below are the commands used to create users, roles, grant and revoke privileges, view user roles and privileges, create policies at the row level, and connect via psql.

9.1 Create a New User

The `CREATE USER` command is used to create a new user in the database system. Users are individual accounts that can connect to the database and perform operations based on their assigned privileges. The command takes the username and password as parameters, allowing the user to authenticate when connecting to the database.

9.2 Create a Role

The `CREATE ROLE` command is used to create a new role in the database system. Roles are a collection of privileges that can be assigned to users or other roles. Roles allow for easier management of privileges by grouping them together under a single name. In our system, we have four types of roles:

- **Internal Student**: Has access to events and the right to participate and volunteer.
- **External Student**: Has access to events, accommodations, and the right to participate.
- **Organizer**: Has access to participants, volunteers, and winners of sponsored events.
- **Admin**: Has all access privileges.

9.3 Grant Privileges

The `GRANT` command is used to grant specific privileges on database objects to a user or role. Privileges include operations such as SELECT, INSERT, UPDATE, and DELETE on tables, as well as the ability to create or modify objects in the database. The command specifies the type of privilege (e.g., SELECT), the object on which the privilege is granted (e.g., TABLE tablename), and the user or role to which the privilege is granted (e.g., TO username).

9.4 Revoke Privileges

The `REVOKE` command is used to revoke previously granted privileges from a user or role. It works similarly to the `GRANT` command but removes privileges instead of granting them. The command specifies the type of privilege to revoke, the object from which to revoke the privilege, and the user or role from which to revoke the privilege.

9.5 View User Roles and Privileges

The `SELECT` command is used to query the system catalog to view information about user roles and their privileges. The query retrieves details such as the name of the role, whether it has superuser privileges, the ability to create roles or databases, and other relevant information. This allows administrators to manage user roles effectively.

9.6 Create Policies at the Row Level

The `CREATE POLICY` command is used to create row-level security policies on tables. Row-level security allows administrators to control which rows of a table users can access based on specified conditions. Policies are defined using SQL expressions that evaluate to true or false, and they are applied to all operations (e.g., SELECT, INSERT, UPDATE, DELETE) on the table.

9.7 Remove User

The `DROP USER` command is used to remove a user from the database system. This command permanently deletes the user account and all associated privileges. It is important to use caution when using this command, as it cannot be undone.

9.8 Remove Role

The `DROP ROLE` command is used to remove a role from the database system. Similar to `DROP USER`, this command permanently deletes the role and all associated privileges. It is important to use caution when using this command, as it cannot be undone.

9.9 psql Connect Statement

The psql connect statement is used to connect to a PostgreSQL database using the psql command-line client. It specifies parameters such as the username (-U), database name (-d), and hostname (-h) to establish a connection to the database. This allows users to interact with the database directly from the command line, executing queries and performing administrative tasks.

10 User Functionalitites

10.1 Login

The admin, student, and organiser can log in to the system using their email and password. Upon successful authentication, a JWT token is generated and returned to the user, which is used for subsequent requests to the server. On expiry of the access token, the user is given a new access token using the refresh token. On entering the wrong credentials, the user is prompted to enter the correct credentials.

10.2 Forgot Password

The user can reset their password by entering their email address. A new password is generated, hashed, and updated in the database. An email is sent to the user's email address with the new password. The user cam them log in using the new password amd reset it to a new one.

Subject: Forgot Password

Dear Bratin,
Your account has been successfully restored. Your new password is: `3fc9c365`**. For security reasons, we recommend changing it after logging in.**
Thank you for registering in our fest. We look forward to your presence.
The link to login is `https://dbms-frontend-flask.vercel.app/login`
Best regards,
Tech Team

10.3 Admin Functionalities

10.3.1 Event Management

10.3.1.1 Add Event

The admin can add a new event to the system by providing details such as event name, type, information, start date and time, end date and time, location, and prize amounts. The event is then added to the database and becomes available for participants and organisers to interact with.

10.3.1.2 Delete Event

The admin can delete an existing event from the system from the event list. The event is then removed from the database and is no longer accessible to participants and organisers.

10.3.1.3 Update Event

The admin can update the details of an existing event in the system by providing the event ID and the fields to be updated. The event details are then modified in the database, reflecting the changes made by the admin.

10.3.2 Student Management

10.3.2.1 Add Student

The admin can add a new student to the system by providing details such as email, name, roll number, phone, college, department, year, type, and password. The student is then added to the database and becomes a registered user of the system. If a student with the same email already exists, the admin is prompted to enter a different email.

10.3.2.2 Delete Student

The admin can delete an existing student from the system by providing the student ID. The student's account is then removed from the database and the student is no longer able to access the system.

10.3.2.3 Update Student

The admin can update the details of an existing student in the system by providing the student ID and the fields to be updated. The student's details are then modified in the database, reflecting the changes made by the admin.

10.3.3 Organiser Management

10.3.3.1 Add Organiser

The admin can add a new organiser to the system by providing details such as email, name, phone, and password. The organiser is then added to the database and becomes a registered user of the system. If an organiser with the same email already exists, the admin is prompted to enter a different email.

10.3.3.2 Delete Organiser

The admin can delete an existing organiser from the system by providing the organiser ID. The organiser's account is then removed from the database and the organiser is no longer able to access the system.

10.3.3.3 Update Organiser

The admin can update the details of an existing organiser in the system by providing the organiser ID and the fields to be updated. The organiser's details are then modified in the database, reflecting the changes made by the admin.

10.3.4 Manage sponsorship

10.3.4.1 Approve Sponsorship

The admin can approve sponsorship requests from organisers. The admin can view all pending notifications for event sponsorships and approve. If a particular organiser is accepted, the request status is updated to 'approved' and an email is sent to the organiser. Also all the other organiser request for the same event are rejected.

Subject: Congratulations!!!

Dear Somya,
We are pleased to inform you that your sponsorship request for the event Codenite has been approved. We greatly appreciate your support and commitment to our cause. Your contribution will play a significant role in the success of the event.
Thank you for your generosity and partnership.
The link to login is `https://dbms-frontend-flask.vercel.app/login`
Warm regards,
The Event Management Team

10.3.4.2 Reject Sponsorship

The admin can reject sponsorship requests from organisers. The admin can view all pending notifications for event sponsorships and reject.

10.4 Naitve Student Functionalities

10.4.1 Signup

The student can sign up for the system by providing details such as email, name, roll number, phone, college, department, year, type, and password. The student is then added to the database and becomes a registered user of the system. If a student with the same email already exists, the student is prompted to enter a different email.

10.4.2 Update Profile

The student can update their profile information such as name, roll number, phone, college, department, year, and password. The student's details are then modified in the database, reflecting the changes made by the student.

10.4.3 Delete Account

The student can delete their account from the system by providing their password. The student's account is then removed from the database and the student is no longer able to access the system.

10.4.4 View Events

The student can view all events available in the system along with details such as event name, type, start date and time, and end date and time.

10.4.5 Register for Event

The student can register for an event by providing the event ID. The student is then added to the list of participants for the event in the database.

10.4.6 Volunteer for Event

The student can volunteer for an event by providing the event ID, additional information, and role. The student is then added to the list of volunteers for the event in the database.

10.4.7 View Schedule

The student can view their schedule of participated events, winning events, and volunteered events. The student can view the details of the events such as event name, type, start date and time, end date and time.

10.5 Guest Student Functionalities

10.5.1 Signup

The student can sign up for the system by providing details such as email, name, roll number, phone, college, department, year, type, and password. The student is then added to the database and becomes a registered user of the system. If a student with the same email already exists, the student is prompted to enter a different email.

10.5.2 Update Profile

The student can update their profile information such as name, roll number, phone, college, department, year, and password. The student's details are then modified in the database, reflecting the changes made by the student.

10.5.3 Delete Account

The student can delete their account from the system by providing their password. The student's account is then removed from the database and the student is no longer able to access the system.

10.5.4 View Events

The student can view all events available in the system along with details such as event name, type, start date and time, and end date and time.

10.5.5 Register for Event

The student can register for an event by providing the event ID. The student is then added to the list of participants for the event in the database.

10.5.6 Book Accommodation

The student can book accommodation for an event by providing the location, check-in and check-out dates, food type. The booking details are then added to the database and the student is provided with accommodation for the event.

10.6 Organiser Functionalities

10.6.1 Signup

The organiser can sign up for the system by providing details such as email, name, phone, and password. The organiser is then added to the database and becomes a registered user of the system. If an organiser with the same email already exists, the organiser is prompted to enter a different email.

10.6.2 Update Profile

The organiser can update their profile information such as name, phone, and password. The organiser's details are then modified in the database, reflecting the changes made by the organiser.

10.6.3 Delete Account

The organiser can delete their account from the system by providing their password. The organiser's account is then removed from the database and the organiser is no longer able to access the system.

10.6.4 View Events

The organiser can view all events available in the system along with details such as event name, type, start date and time, and end date and time.

10.6.5 Request Sponsorship

The organiser can request to sponsor an event by providing the event ID and sponsorship amount. The request is then added to the database and the organiser becomes a sponsor for the event on approval by the admin.

10.6.6 Declare Winners

The organisers can declare the winners for an event by providing the event ID and the winner's student IDs for which it is sponshoring. The winners are then added to the database and the event is updated with the winners. The winners are notified by an email.

Subject: Congratulations! You have won a prize in Codenite

Dear Sarika,
We are writing to inform you that you have won the third prize in the event Codenite. Congratulations on this well-deserved recognition! Your dedication and efforts have paid off, and we are honored to award you this prize. Please contact the organising team for the prizes.
Thank you for your participation and enthusiasm.
The link to login is `https://dbms-frontend-flask.vercel.app/login`
Kind regards,
The Organizing Committee

11 Systems Design

11.1 Error Handling and Robustness

Error handling and robustness are critical aspects of our system design to ensure smooth operation and reliability. We implement comprehensive error handling mechanisms throughout the application to detect and handle unexpected errors gracefully. This includes thorough validation of user inputs, defensive programming practices, and proper exception handling. Additionally, we employ logging to capture errors and debug information, facilitating quick diagnosis and resolution of issues. By prioritizing error handling and robustness, we enhance the stability and resilience of our system, minimizing downtime and providing a seamless user experience.

11.2 Data Security Policy and Suitable Access Control

Our system adheres to a stringent data security policy and employs suitable access control measures to safeguard sensitive information. We implement encryption techniques to protect data both in transit and at rest, ensuring confidentiality and integrity. Access to data is controlled based on user roles and permissions, following the principle of least privilege. Additionally, we utilize authentication mechanisms such as JWT (JSON Web Tokens) to authenticate users and enforce access control policies. Regular security audits and vulnerability assessments are conducted to identify and mitigate potential security risks proactively. By adopting a robust data security policy and suitable access control measures, we prioritize the protection of user data and maintain compliance with regulatory standards.

11.3 Scalability and Performance Optimization

Scalability and performance optimization are fundamental considerations in our system design to accommodate growth and ensure optimal performance under varying loads. We employ scalable architectural patterns such as microservices and containerization to facilitate horizontal scaling and efficient resource utilization. Additionally, we implement caching mechanisms, database indexing, and asynchronous processing to optimize performance and reduce latency. Continuous performance monitoring and tuning are performed to identify bottlenecks and fine-tune system parameters for optimal efficiency. By prioritizing scalability and performance optimization, we ensure that our system can handle increased workload and deliver responsive user experiences without compromising on performance.

11.4 API Design

API design is a crucial aspect of our system to ensure seamless integration with external systems and promote developer productivity. We adhere to RESTful API design principles, ensuring that our APIs are intuitive, consistent, and easy to consume. Clear and comprehensive API design guidelines are established, outlining

best practices for resource naming, HTTP methods usage, error handling, and authentication mechanisms. Additionally, we implement versioning strategies to manage changes and deprecations effectively, enabling backward compatibility and smooth migration paths for consumers. By prioritizing API design, we foster interoperability, empower third-party developers, and accelerate the adoption of our platform.

11.5 Testing

Testing ensures system reliability and functionality through methodologies such as unit testing, integration testing, and performance testing, ultimately guaranteeing a high-quality product.

12 Acknowledgments

We gratefully acknowledge the guidance and support of Professors Pabitra Mitra[3] and K. Sreenivasa Rao[4] for their invaluable contributions to our group project in the Database Management Systems Laboratory.

References

[1] Mermaid. `https://mermaid-js.github.io/mermaid/`, 2024. Accessed: March 1, 2024.

[2] Vertabelo. `https://my.vertabelo.com`, 2024. Accessed: March 1, 2024.

[3] Pabitra Mitra. Personal website. `https://cse.iitkgp.ac.in/~pabitra/`. Accessed: March 1, 2024.

[4] K S Rao. Personal website. `https://cse.iitkgp.ac.in/~ksrao/`. Accessed: March 1, 2024.