DBMS - Assignment - 4 Report

Team Details:

K K Tarun Kumar– PES1UG19CS222
Koushik Varma Mandapati – PES1UG19CS230
Kosaraju Bhargava Krishna – PES1UG19CS225

Project Title: Event Registration Management System

Dependencies Installed for Database Connectivity:

Axios: Library used in the front end to POST data to backend or GET data from backend Simplifies the process of sending User Queries as JSON objects and receiving server results as JSON objects asynchronously.

Node-postgres pg : Non-blocking POSTGRESQL client for JavaScript. Helps us execute POSTGRESQL commands from the backend node js server and fetch the results to be sent to the frontend. This is the heart of the server business logic.

Statements Executed by backend:

Home Page:

Participant Details:

- Home
- Participant Details
- Login
- Event Display
- Marketer Details
- Participants in Team

Logout?

display participant details with participant's first name and last name

Nidhi | Burman | Retrieve

participant ID: PRT003,name:Nidhi,participant age:33,participant mail_id:nidhiburman@ersfg.com

Query in Backend:

```
app.get("/getParticipantDetails/:first_name/:lastname", async (req, res) => {
  console.log(req.params.first_name)
  first_name = `'${req.params.first_name.split("=")[1]}'`;
  last_name = `'${req.params.lastname.split("=")[1]}'`;

  const q = `select *
  from Participant_
  where Participant_.L_Name=${last_name} and Participant_.F_Name=${first_name};`;
  console.log(q)

  try {
    let order = await db.query(q);
    order = order["rows"];
    console.log(order);

    if (order) {
      return res
        .status(200)
        .json({ success: "Found orders", orders: order });
    } else {
      throw "Issues with searching in database";
    }
  } catch (err) {
    console.log(err);
    return res.status(404).json({ failure: "Internal server err" });
  }
});
```

Display Marketer Details:

- Home
- Participant Details
- Login
- Event Display
- Marketer Details
- Participants in Team

Logout?

display all marketers

Search

1. MAR001, username: John, Referal_Code: REF001
2. MAR002, username: Joe, Referal_Code: REF002
3. MAR003, username: Michael, Referal_Code: REF003

Query in Backend:

```
app.get("/displayMarketer", async (req, res) => {
  console.log("Called display all marketers....");

  try {
    const query = `select *
    from Marketer`;

    console.log(query);
```

Participants Details who are in a Team:

- Home
- Participant Details
- Login
- Event Display
- Marketer Details
- Participants in Team

Logout?

display participant details who are in a team

Search

1. participant team: TEM002,participant ID: PRT003,name:Nidhi,participant age:33,participant mail_id:nidhiburman@ersfg.com
2. participant team: TEM001,participant ID: PRT004,name:Kaushal,participant age:22,participant mail_id:kaushalreddy@ersfg.com
3. participant team: TEM001,participant ID: PRT005,name:Janardhan,participant age:21,participant mail_id:janardhancheth@ekffg.com
4. participant team: TEM003,participant ID: PRT006,name:Pooja,participant age:29,participant mail_id:poojapattekar@erchg.com

Query in Backend:

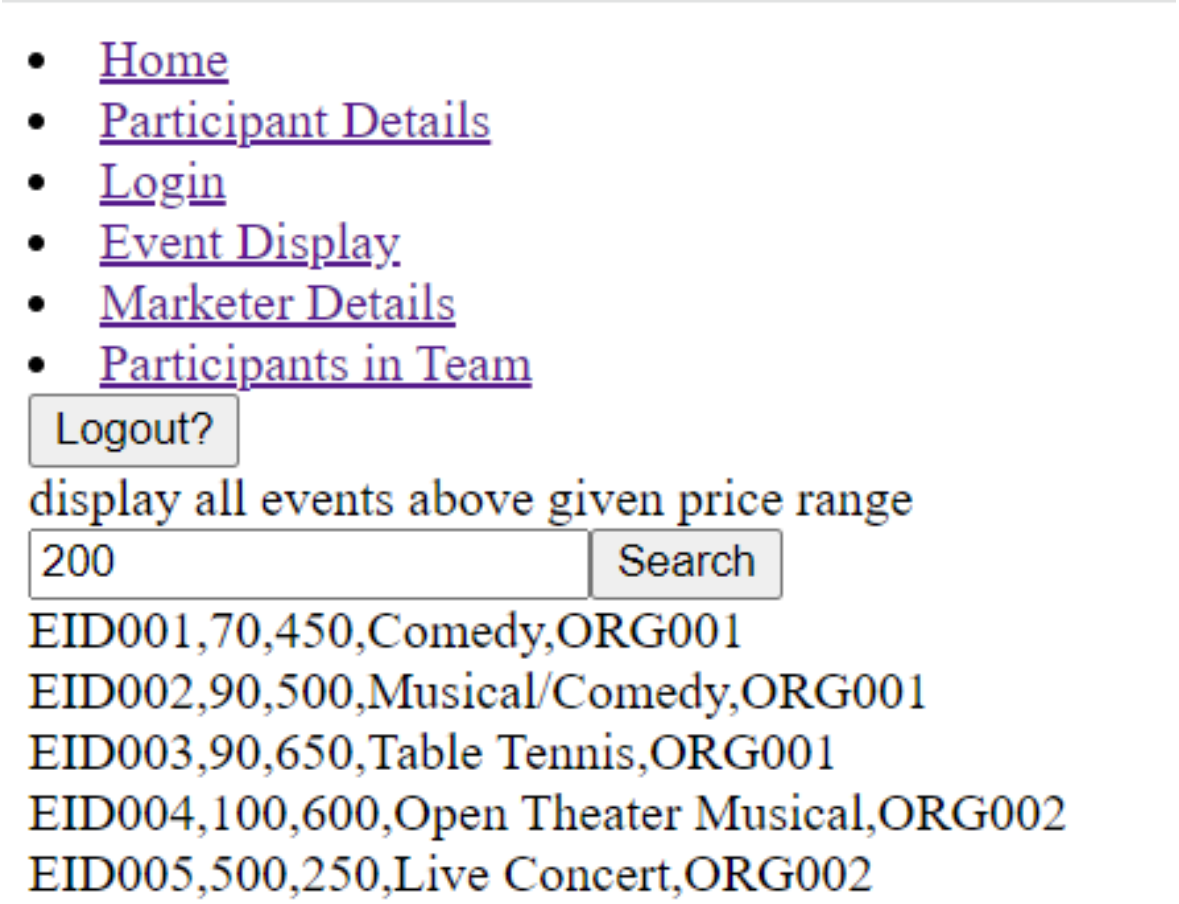
```
app.get("/teamParticipants", async (req, res) => {
  console.log("Called sort order display teams....");

  try {
    const query_rest = `select *
    from participant_
    where team_id in (
      select team_id
      from Participant_
      intersect
      select team.team_id
      from team
    );`;

    console.log(query_rest);

    let all_rest = await db.query(query_rest);
    all_rest = all_rest["rows"];
    console.log(all_rest);
```

Event Details Above a specified price range:



Query in Backend:

```
app.get("/displayEvents/:price", async (req, res) => {
  console.log("Called /displayEvents/:price....");
  const q_price = req.params.price.split("=")[1];
  console.log(q_price);

  try {
    const query_dishes = `select
        *
        from Event
        where Event_Price >= ${q_price};`;
    console.log(query_dishes);
```

Changes/Expansion in Business Need:

1) Schema Changes
    a) In order to track events, we need to make an even more robust schema to store the details of the all the organizers, marketers and participants.
    b) We also need to track the status of the event organized by the organizer and store it in the database. Hence the schema for the status should be changed to accommodate it.
2) Constraint
    a) We have currently allowed changes to be made to the payment details of the payment made to the organizer. This should not be allowed since an organizer should not handle the payment details of a customer/participant. Hence, we want to add this constraint additionally.
3) DBMS Migration:
    a) As the event management system needs to scale, we need to allow storage of more and more unstructured user data like Organizer, marketer and participants usernames and passwords, Events Descriptions, Team registered, etc for creating recommender systems. Hence, we would have to migrate to NoSQL like MongoDB.

Migrating to No-SQL:

Out of the 4 varieties, we would like to migrate to key-value stores because our data is semi structured and not completely unstructured. Hence key-value retrieval of data would be fast and easier to store.

No-SQL also supports horizontal scaling in case customer data traffic increases.

MongoDB is the No-SQL we would like to migrate to because it scales out of the box which would reduce the efforts required to scale the Database.

But however, a hybrid model is still better suited since transactions for bank related tasks is still preferable because of the atomicity that it offers.

Relations can be modelled easily using graph-based No-SQL varieties like Neo4js.This will allow us to easily do data analytics.

Steps for migration:

1)Heterogeneous migration tools, handling NULL values

2)MongoExport and ETL

3)No-SQL conversion tool

Contribution:

K K Tarun Kumar (PES1UG19CS222)- Backend, Data Migration

Koushik Varma Mandapati (PES1UG19CS230) - Front end, Report

Kosaraju Bhargava Krishna (PES1UG19CS225) – Constraints, Schema Changes