Campus Connect 360

Database Systems
COP 4710 - Spring 2024
Group 88 - David Umanzor

Table of Contents

Project Description

Campus Connect 360 is a web application to connect students within their own campus and outside to other universities. It lets them stay connected to events and search for things happening nearby or at their own university.

GUI

Platform:

- Web Application accessible through web browsers such as Chrome, Firefox and Safari

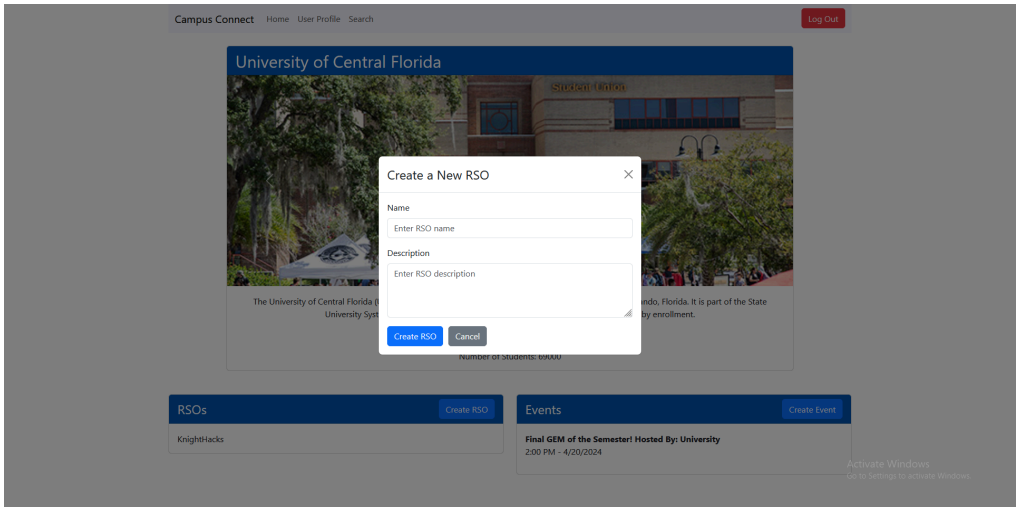
Languages:

- Frontend: JavaScript (ES6+), utilizing the React framework for dynamic component-based architecture.
- CSS3 and Bootstrap for styling and responsive design to ensure a consistent look and feel across various devices and screen sizes.
- Backend: Node.js with Express.js framework to manage server-side logic and HTTP requests

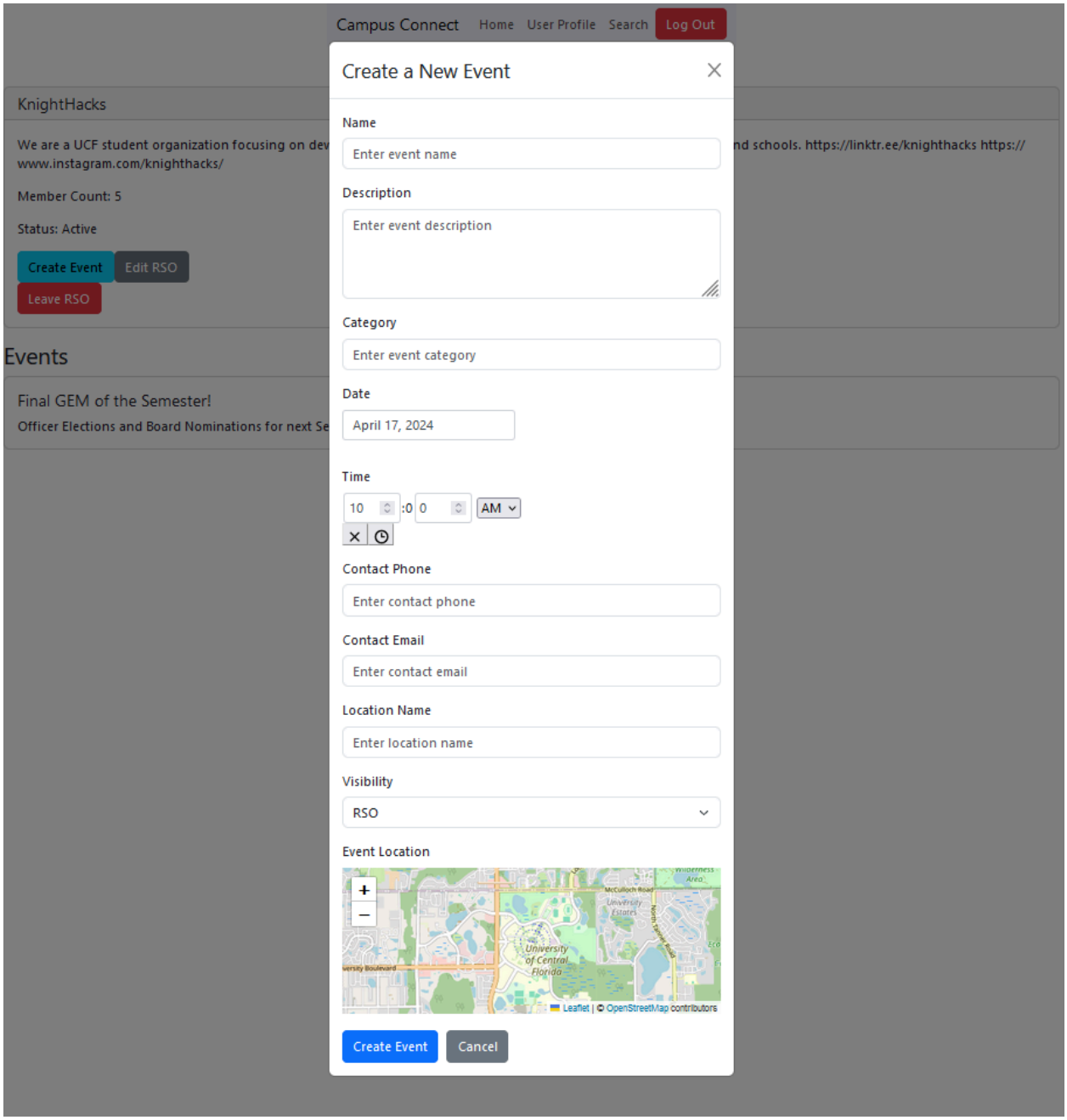
DBMS (Database Management System):

- PostgreSQL, chosen for its robustness, reliability, and support for advanced data types and functionality.
    - Hosted by Heroku.

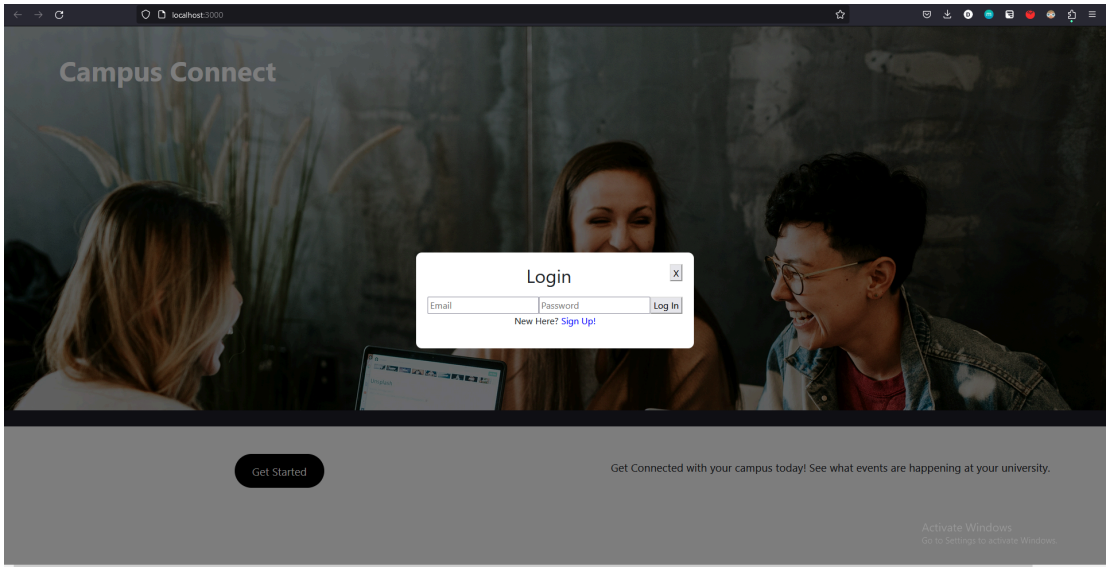
Screen Shots:
Create RSO:



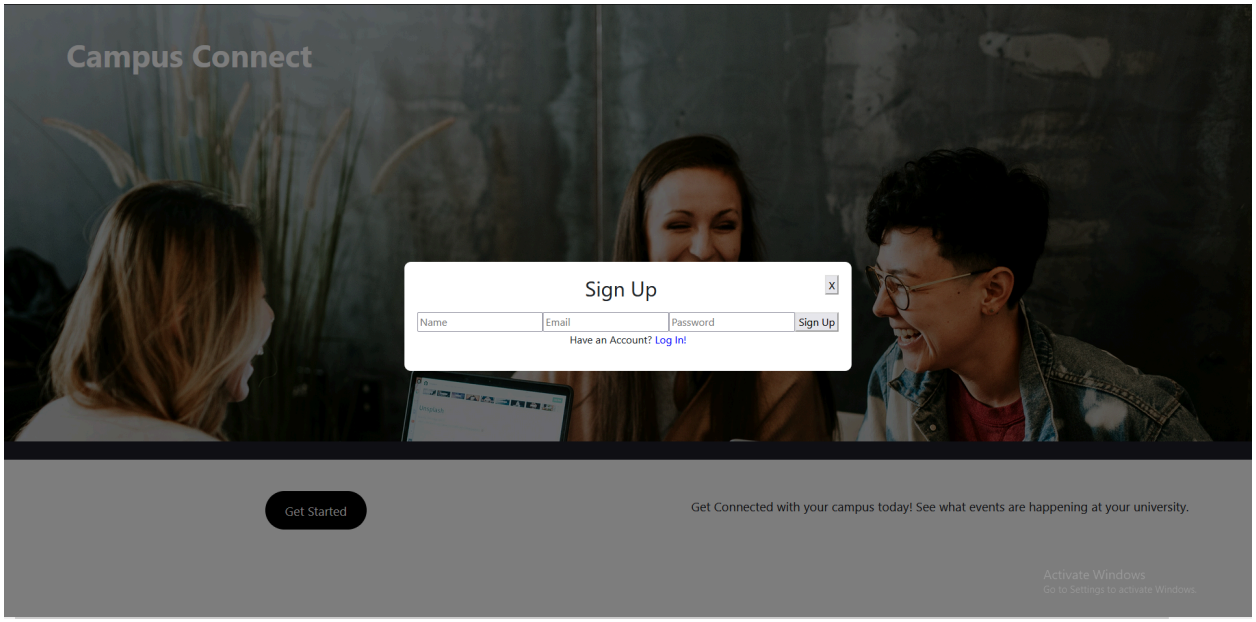
Create Event:



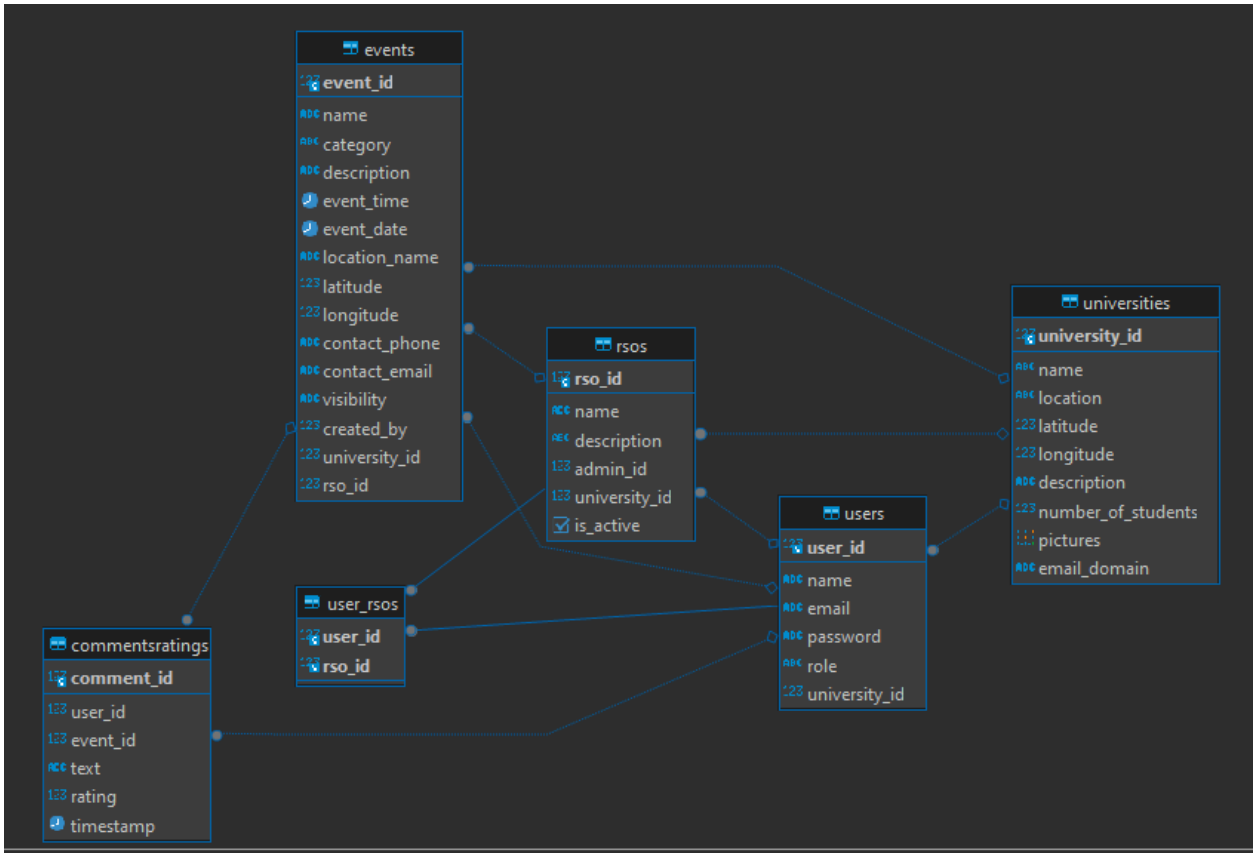
Login Form:



Sign Up Form:



ER-Model



Schema

```
CREATE DATABASE campusconnect;

CREATE TABLE Universities (
  university_id SERIAL PRIMARY KEY,
  name VARCHAR(255) UNIQUE NOT NULL,
  location TEXT,
  latitude DECIMAL(9,6),
  longitude DECIMAL(9,6),
  description TEXT,
  number_of_students INT,
  pictures TEXT[], -- Array of image URLs
  email_domain VARCHAR(255) UNIQUE
);
```

```
CREATE TABLE Users (
  user_id SERIAL PRIMARY KEY,
  name VARCHAR(255),
  email VARCHAR(255) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,
  role VARCHAR(50) CHECK (role IN ('student', 'admin',
'superAdmin')),
  university_id INT,
  FOREIGN KEY (university_id) REFERENCES
Universities(university_id)
);

CREATE TABLE RSOs (
  rso_id SERIAL PRIMARY KEY,
  name VARCHAR(255),
  description TEXT,
  admin_id INT,
  university_id INT,
  is_active BOOLEAN DEFAULT FALSE,
  FOREIGN KEY (admin_id) REFERENCES Users(user_id),
  FOREIGN KEY (university_id) REFERENCES
Universities(university_id)
);

CREATE TABLE Events (
  event_id SERIAL PRIMARY KEY,
  name VARCHAR(255),
  category VARCHAR(50),
  description TEXT,
  event_time TIME,
  event_date DATE,
  location_name VARCHAR(255),
  latitude DECIMAL(9,6),
```

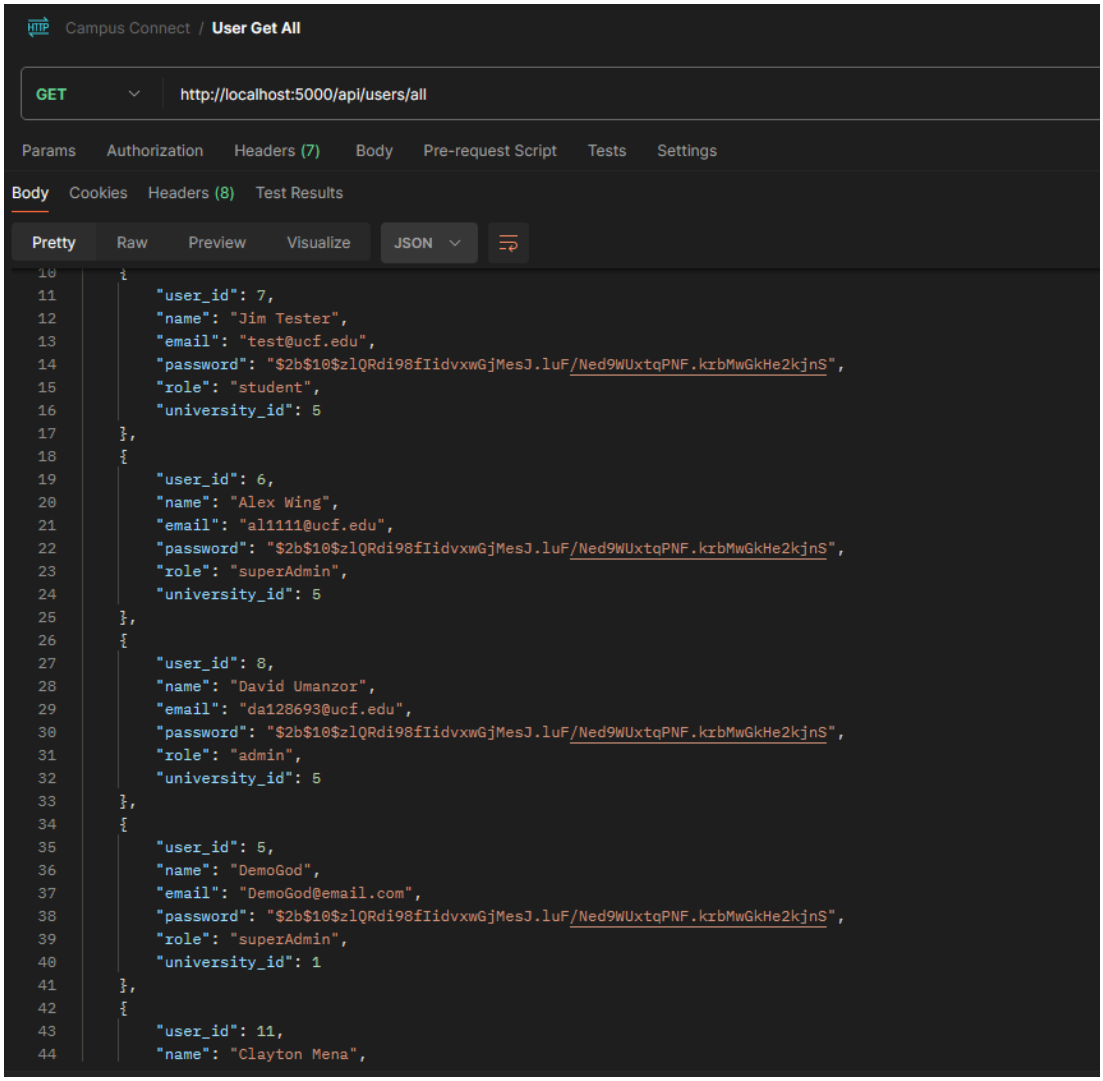
```
  longitude DECIMAL(9,6),
  contact_phone VARCHAR(20),
  contact_email VARCHAR(255),
  visibility VARCHAR(50) CHECK (visibility IN ('public',
'private', 'rso')),
  created_by INT,
  university_id INT,
  rso_id INT,
  FOREIGN KEY (created_by) REFERENCES Users(user_id),
  FOREIGN KEY (university_id) REFERENCES
Universities(university_id),
  FOREIGN KEY (rso_id) REFERENCES RSOs(rso_id)
);

CREATE TABLE CommentsRatings (
  comment_id SERIAL PRIMARY KEY,
  user_id INT,
  event_id INT,
  text TEXT,
  rating INT CHECK (rating >= 1 AND rating <= 5),
  timestamp TIMESTAMP WITHOUT TIME ZONE DEFAULT
CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES Users(user_id),
  FOREIGN KEY (event_id) REFERENCES Events(event_id)
);

CREATE TABLE User_RSOs (
  user_id INT,
  rso_id INT,
  PRIMARY KEY (user_id, rso_id),
  FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE
CASCADE,
  FOREIGN KEY (rso_id) REFERENCES RSOs(rso_id) ON DELETE CASCADE
);
```

```
CREATE OR REPLACE FUNCTION update_rso_status()
RETURNS TRIGGER AS $$
BEGIN

END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_user_rso_insert
AFTER INSERT ON User_RSOs
FOR EACH ROW
EXECUTE FUNCTION update_rso_status();

CREATE TRIGGER after_user_rso_delete
AFTER DELETE ON User_RSOs
FOR EACH ROW
EXECUTE FUNCTION update_rso_status();
```

The Tables

are prepopulated with multiple users, universities, events, and rsos

Example of Users:

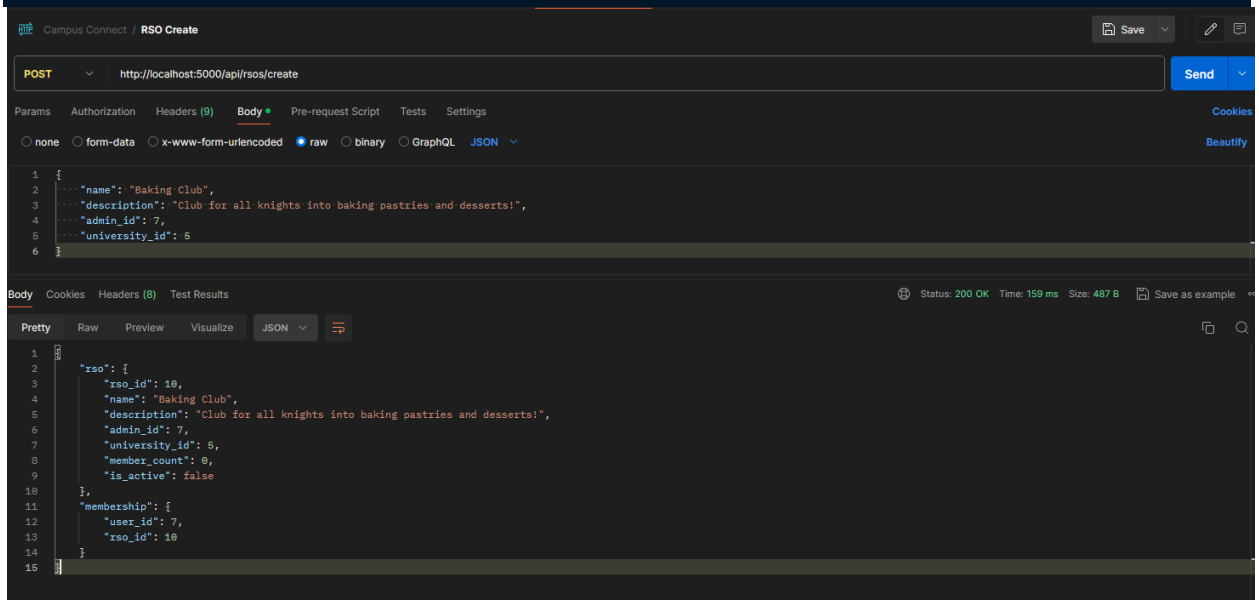


SQL examples and results

Insert a new rso (user must be a admin of rso making them an member)

```
// Create RSO and add maker as member
router.post('/create', async (req, res) => {
    const { name, description, admin_id, university_id } =
req.body;

    try {
        // Start a transaction
        await pool.query('BEGIN');

        // Insert the new RSO
        const newRso = await pool.query(
            'INSERT INTO rsos (name, description, admin_id,
university_id) VALUES ($1, $2, $3, $4) RETURNING *',
            [name, description, admin_id, university_id]
        );

        if (newRso.rows.length > 0) {
            const rso_id = newRso.rows[0].rso_id;

            // Insert the creator as a member of the RSO
            const userRso = await pool.query(
                'INSERT INTO User_RSOs (user_id, rso_id) VALUES
($1, $2) RETURNING *',
                [admin_id, rso_id]
            );

            // Commit the transaction
            await pool.query('COMMIT');

            res.json({ rso: newRso.rows[0], membership:
userRso.rows[0] });
        } else {
            // Rollback in case of any failure
```

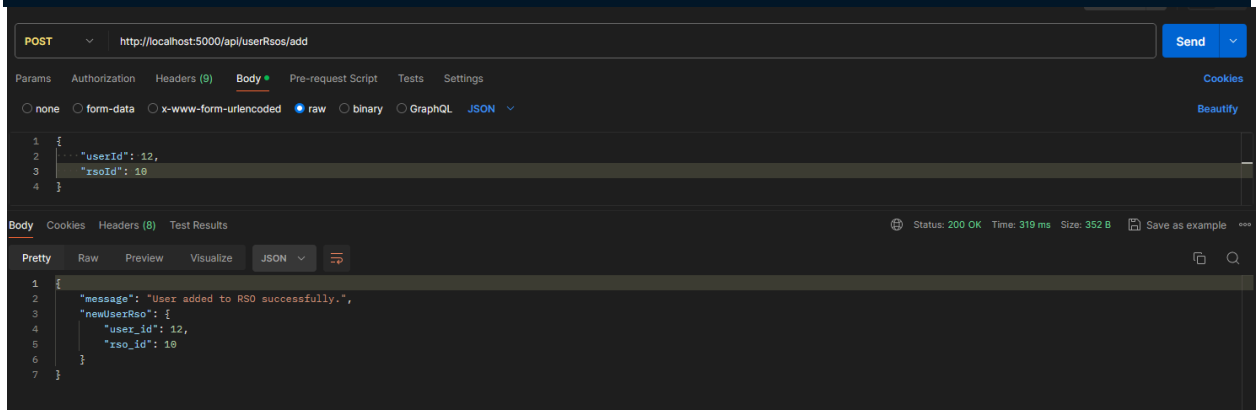
```
            await pool.query('ROLLBACK');
            res.status(400).json({ message: "Failed to create
RSO" });
        }
    } catch (err) {
        // Rollback the transaction on error
        await pool.query('ROLLBACK');
        console.error(err.message);
        res.status(500).json({ message: "Server error" });
    }
});
```



Add user to a rso

```
router.post('/add', async (req, res) => {
    const { userId, rsoId } = req.body;
    try {
        const newUserRso = await pool.query(
            'INSERT INTO User_RSOs (user_id, rso_id) VALUES ($1,
$2) ON CONFLICT DO NOTHING RETURNING *',
            [userId, rsoId]
        );
        if (newUserRso.rows.length > 0) {
```

```
            res.json({ message: "User added to RSO
successfully.", newUserRso: newUserRso.rows[0] });
        } else {
            res.status(400).json({ message: "User already a
member or invalid IDs." });
        }
    } catch (err) {
        console.error(err.message);
        res.status(500).json({ message: "Server error" });
    }
});
```

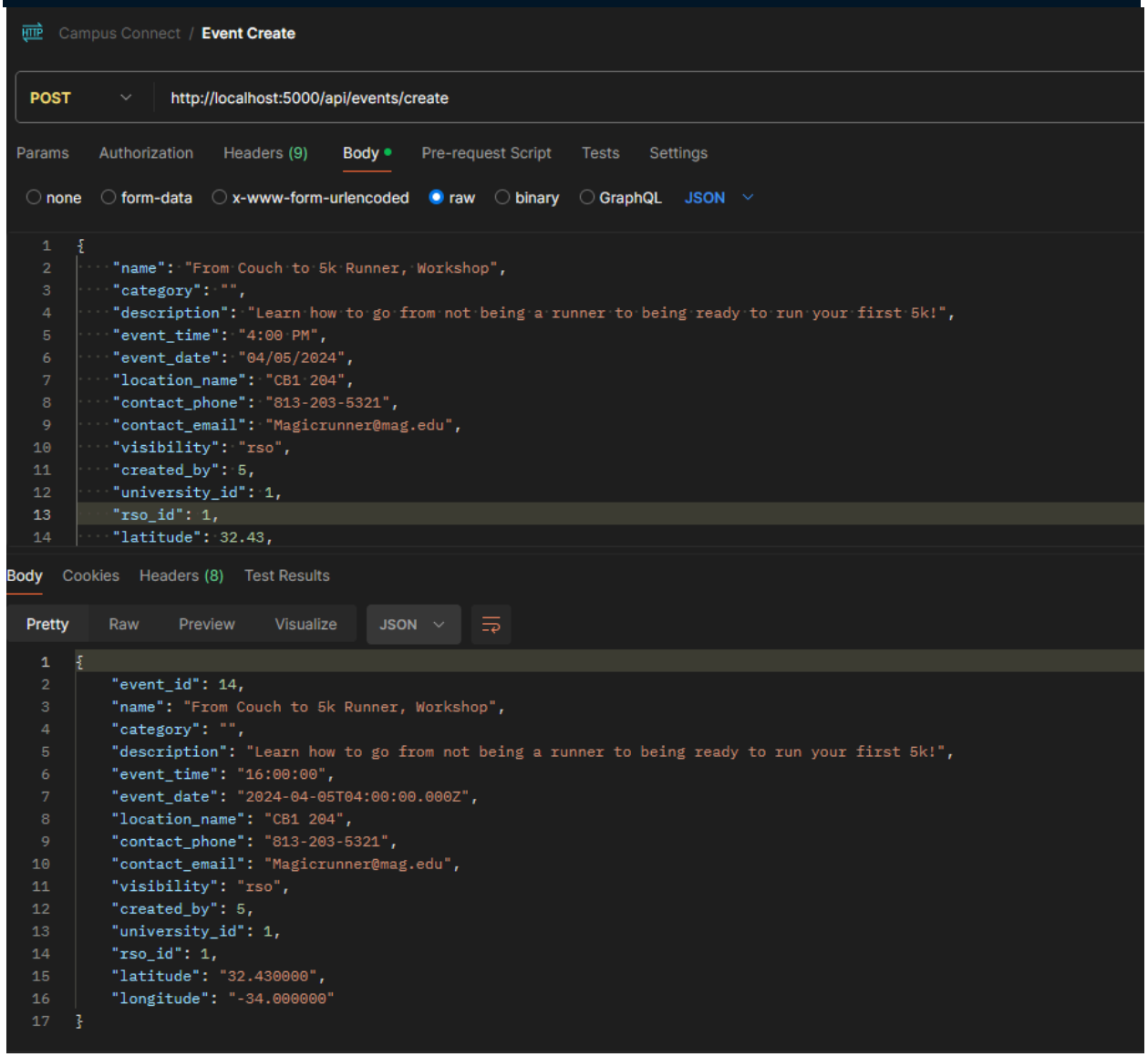


Create Event

```
router.post('/create', async (req, res) => {
    const { name, category, description, event_time, event_date,
location_name, latitude, longitude, contact_phone,
contact_email, visibility, created_by, university_id, rso_id } =
req.body;

    try {
        // Check for any events at the same location and time
        const conflictCheckQuery = `
            SELECT r.name AS rso_name, e.location_name,
e.event_time
            FROM events e
            JOIN RSOs r ON e.rso_id = r.rso_id
```

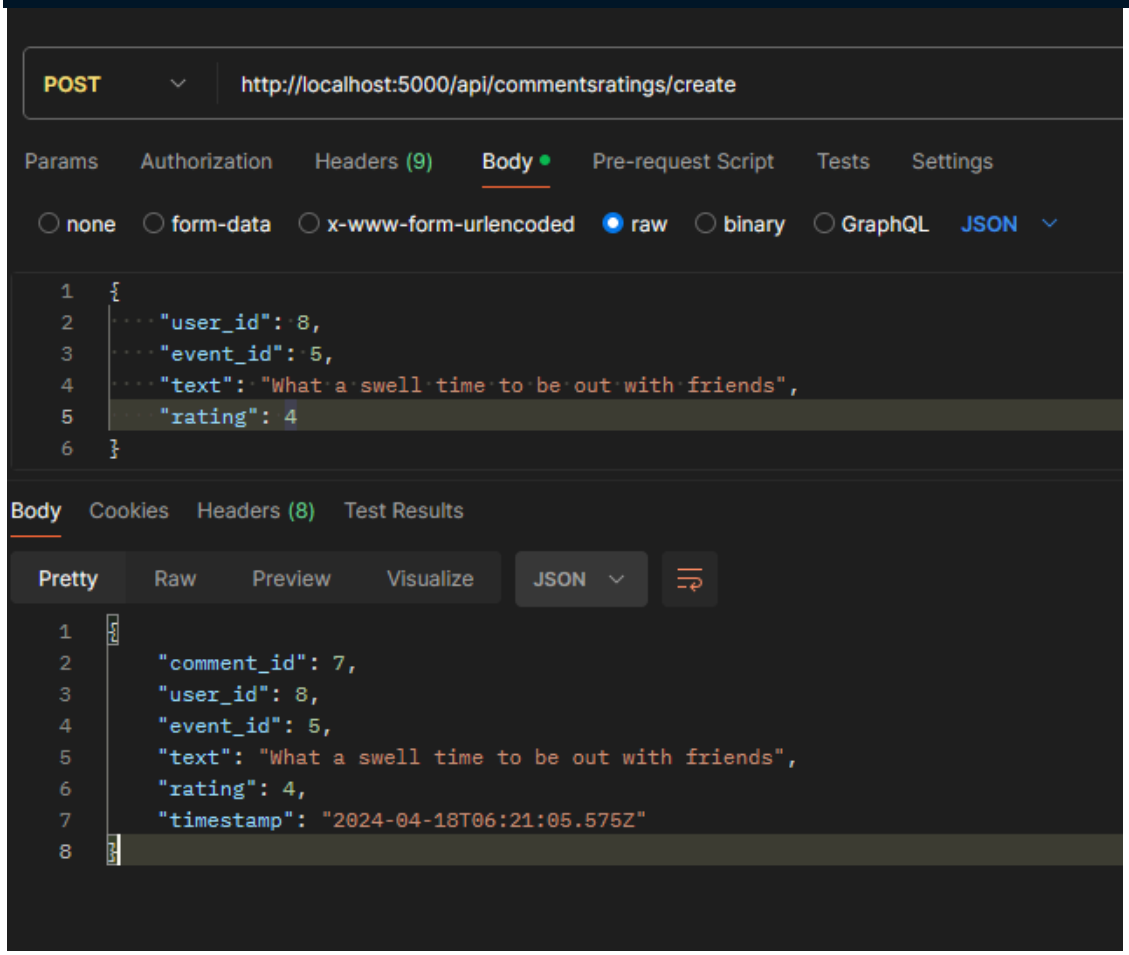
```
            WHERE e.location_name = $1 AND e.event_date = $2 AND
e.event_time = $3 AND r.is_active = true
            LIMIT 1;
        `;
        const conflictCheck = await
pool.query(conflictCheckQuery, [location_name, event_date,
event_time]);

        if (conflictCheck.rows.length > 0) {
            const conflictEvent = conflictCheck.rows[0];
            return res.status(409).json({ message: `Conflict
with event from "${conflictEvent.rso_name}" -
${conflictEvent.location_name}, at ${conflictEvent.event_time}.`
});
        }

        // No conflict found
        const createEventQuery = `
            INSERT INTO events
            (name, category, description, event_time,
event_date, location_name, latitude, longitude, contact_phone,
contact_email, visibility, created_by, university_id, rso_id)
            VALUES
            ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12,
$13, $14)
            RETURNING *`;

        // Execute the query to create a new event
        const newEvent = await pool.query(createEventQuery,
[name, category, description, event_time, event_date,
location_name, latitude, longitude, contact_phone,
contact_email, visibility, created_by, university_id, rso_id ||
null]);
        res.json(newEvent.rows[0]);
```

```
    } catch (err) {
        console.error(err.message);
        res.status(500).json({ message: "Server error", details:
err.message });
    }
});
```



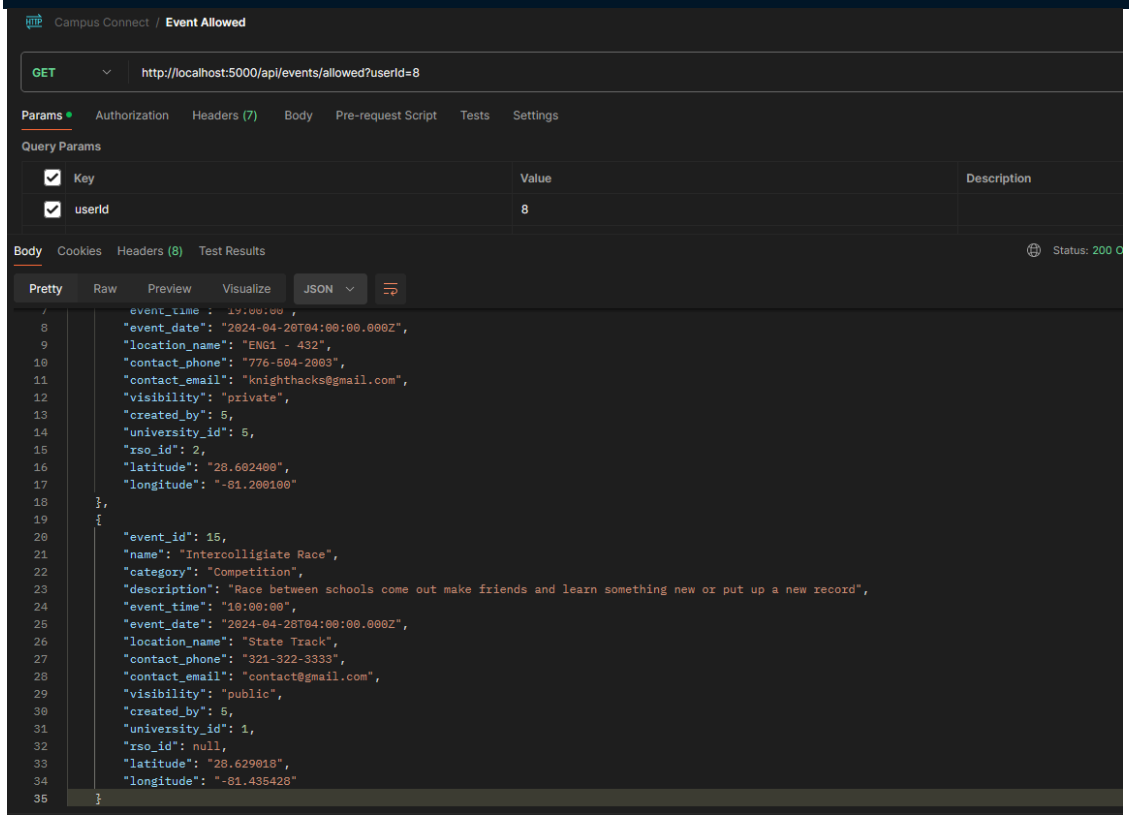
Add comment

```
router.post('/create', async (req, res) => {
    const { user_id, event_id, text, rating } = req.body;
    try {
        const newCommentRating = await pool.query(
            'INSERT INTO commentsratings (user_id, event_id,
text, rating) VALUES ($1, $2, $3, $4) RETURNING *',
            [user_id, event_id, text, rating]
        );
        res.json(newCommentRating.rows[0]);
    } catch (err) {
        console.error(err.message);
    }
});
```



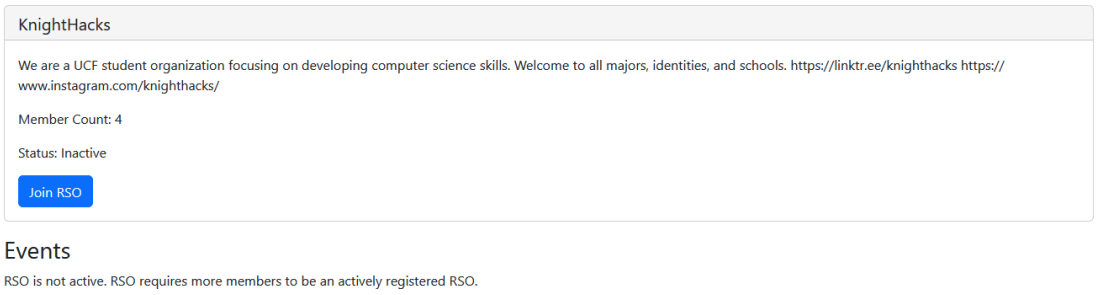
View events (I called it allowed for what user is allowed to see)

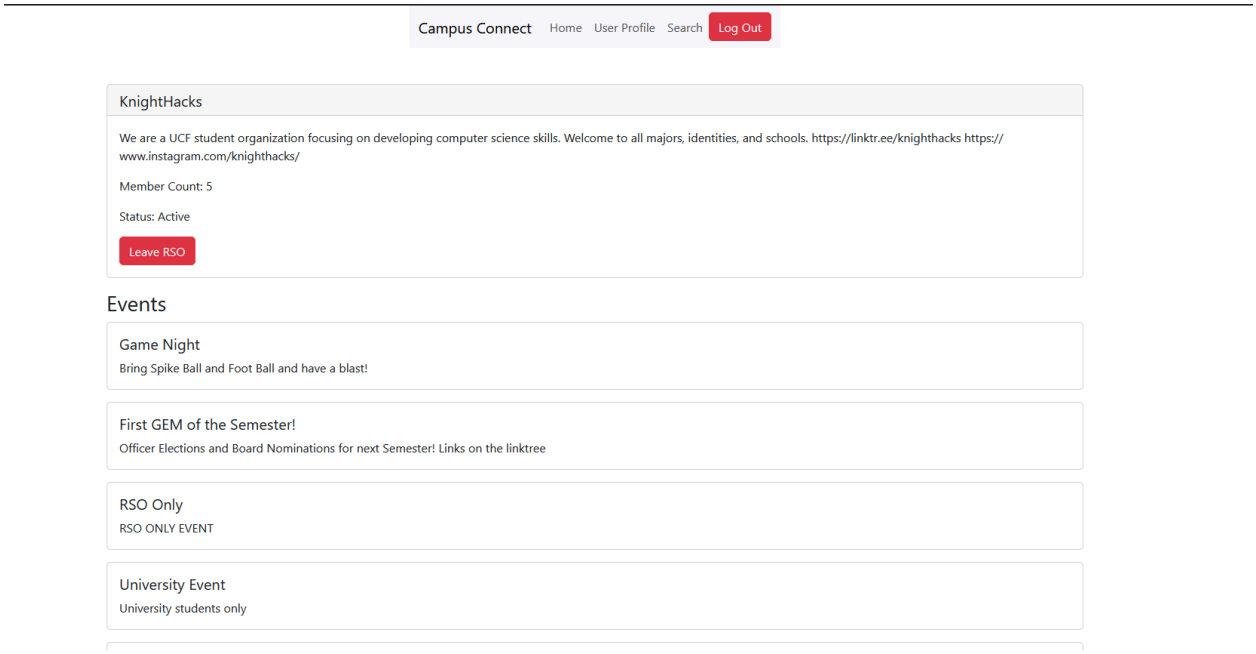
```
router.get('/allowed/', async (req, res) => {
    const { userId } = req.query;
    try {
        const events = await pool.query(`
            SELECT e.* FROM Events e
            LEFT JOIN RSOs r ON e.rso_id = r.rso_id
            WHERE e.visibility = 'public'
            OR (e.visibility = 'private' AND e.university_id =
(SELECT university_id FROM Users WHERE user_id = $1))
            OR (e.visibility = 'rso' AND e.rso_id IN (SELECT
rso_id FROM User_RSOs WHERE user_id = $1) AND r.is_active)
        `, [userId]);
        res.json(events.rows);
    } catch (err) {
        console.error(err.message);
        res.status(500).json({ message: "Server error" });
    }
});
```



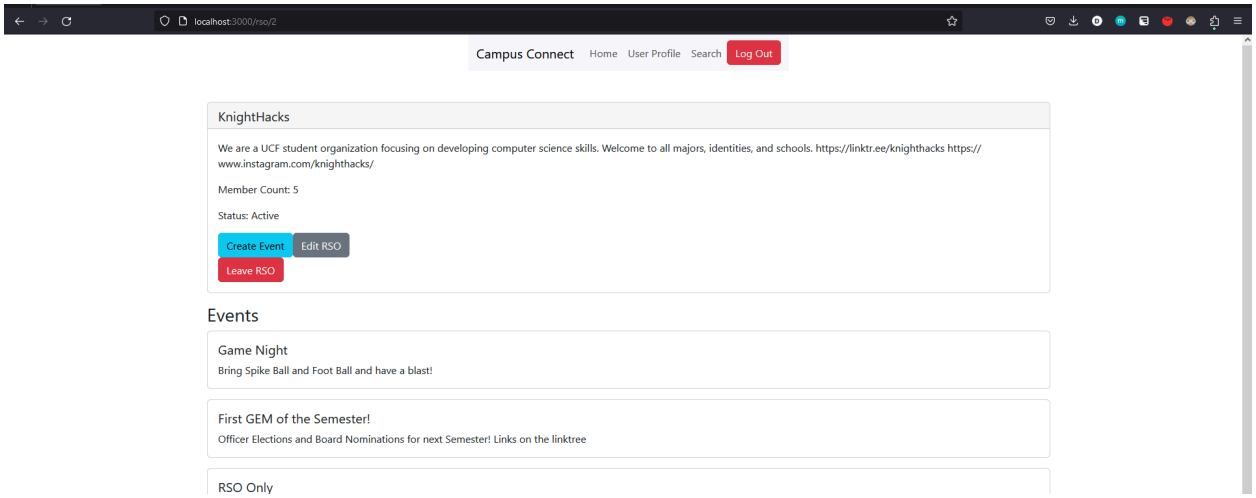
Constraint Enforcement

Active vs Inactive RSO:

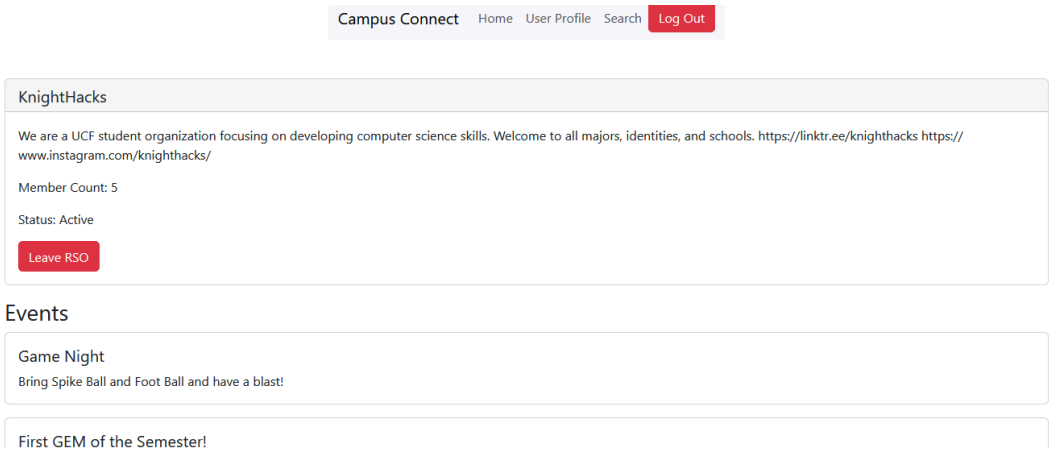




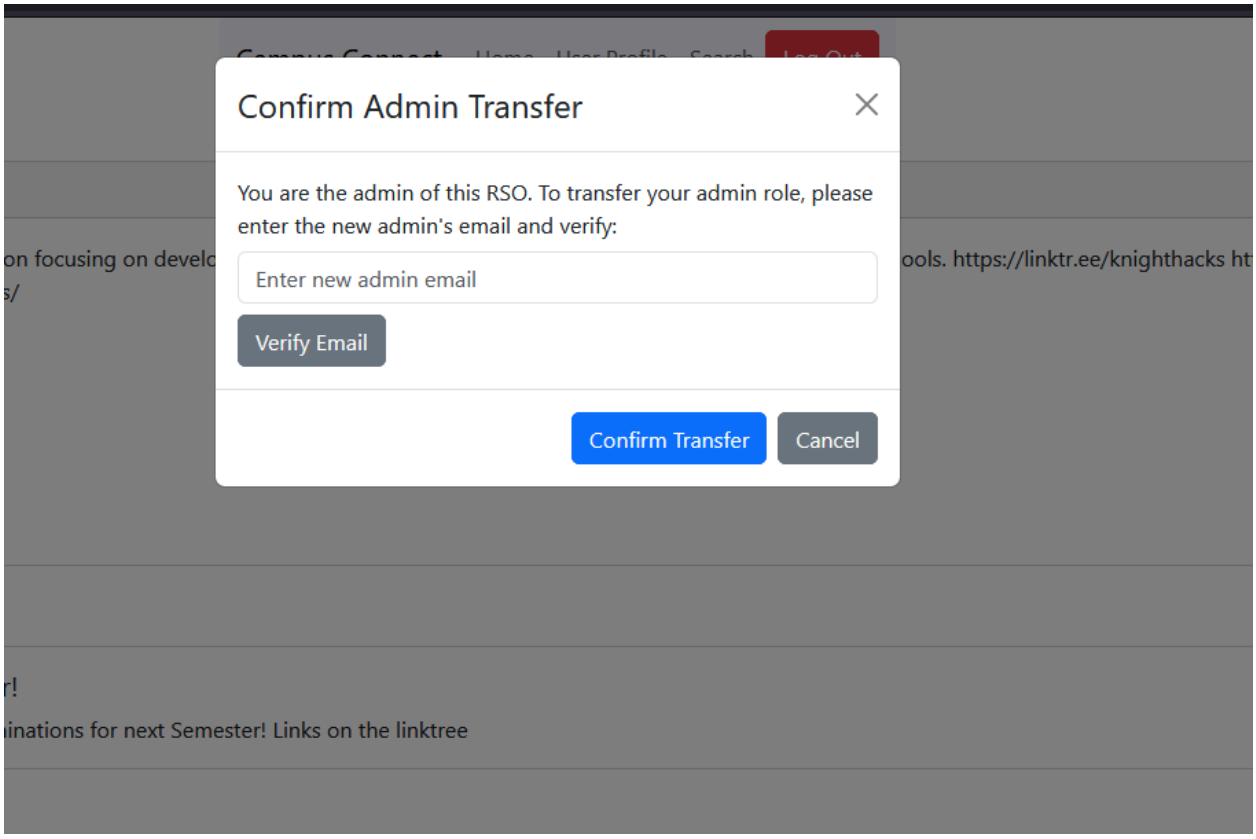
Only Admin can make events:



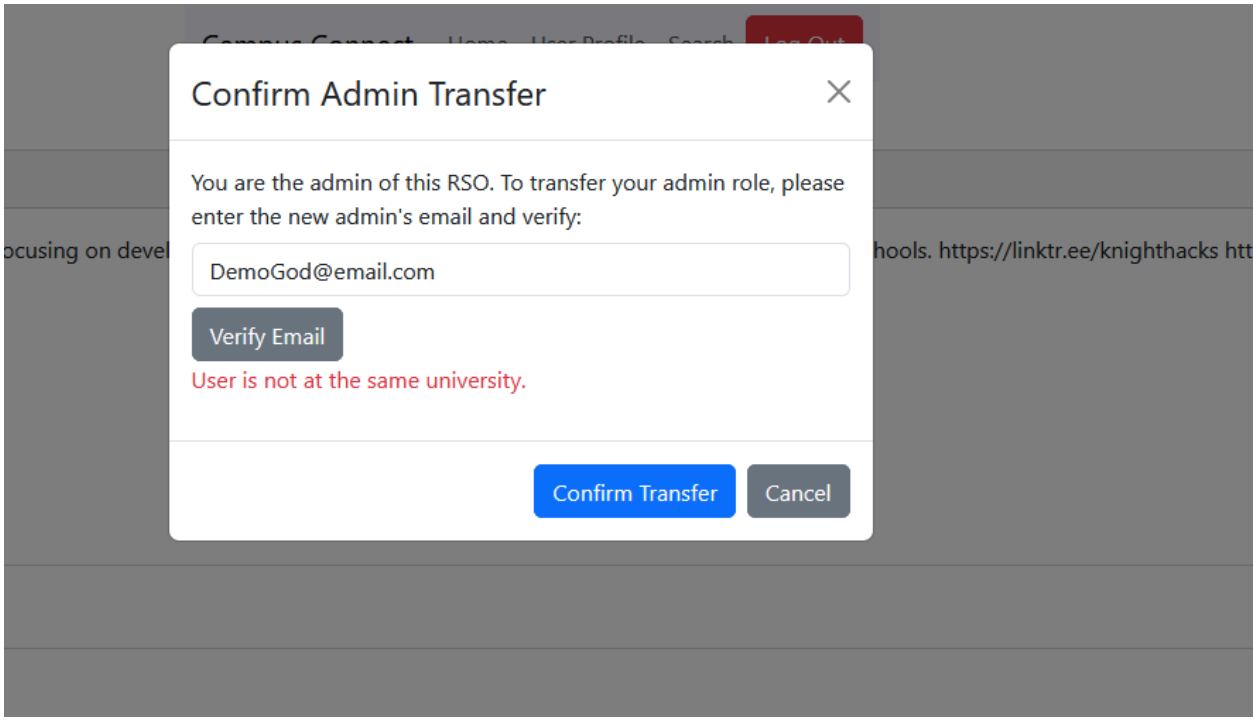
In my site the buttons for creating event and editing the rso do not show up for anyone outside the admin.



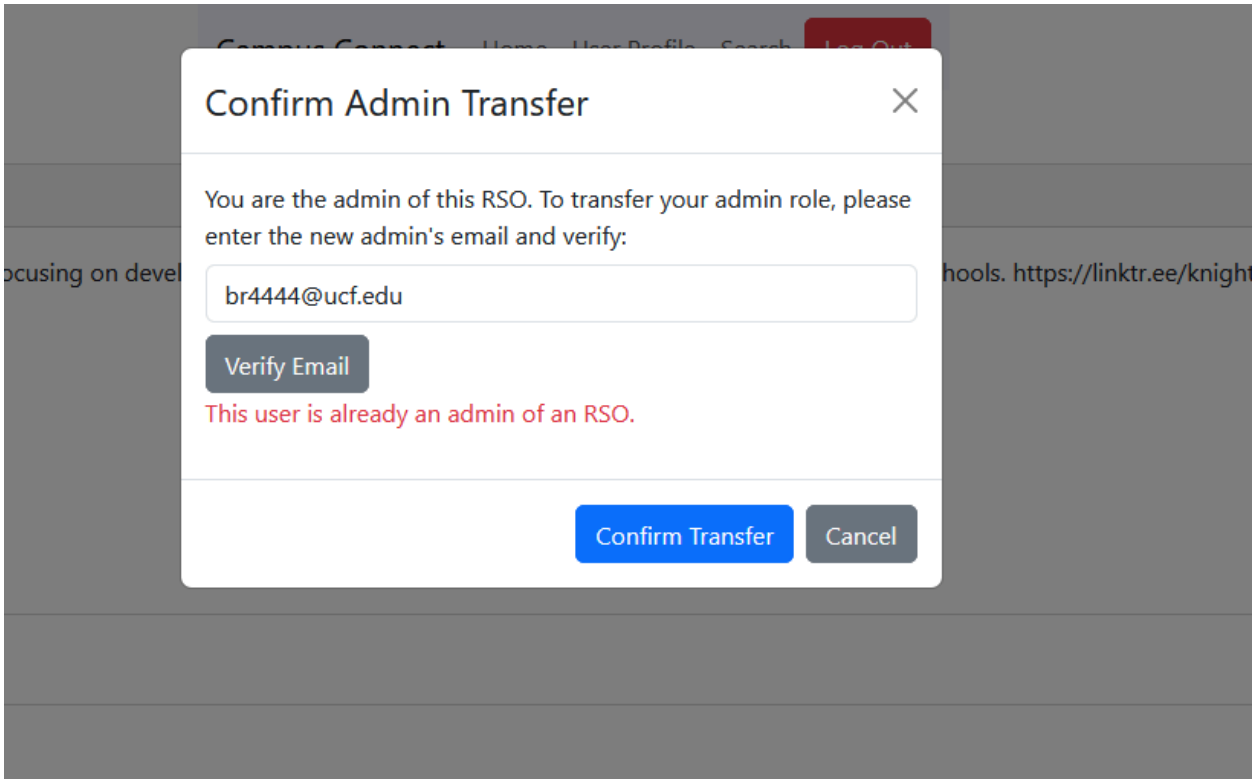
Admin in RSO:
This makes sure Admins can not leave without transferring ownership to another user.



User must be eligible such that, at same university

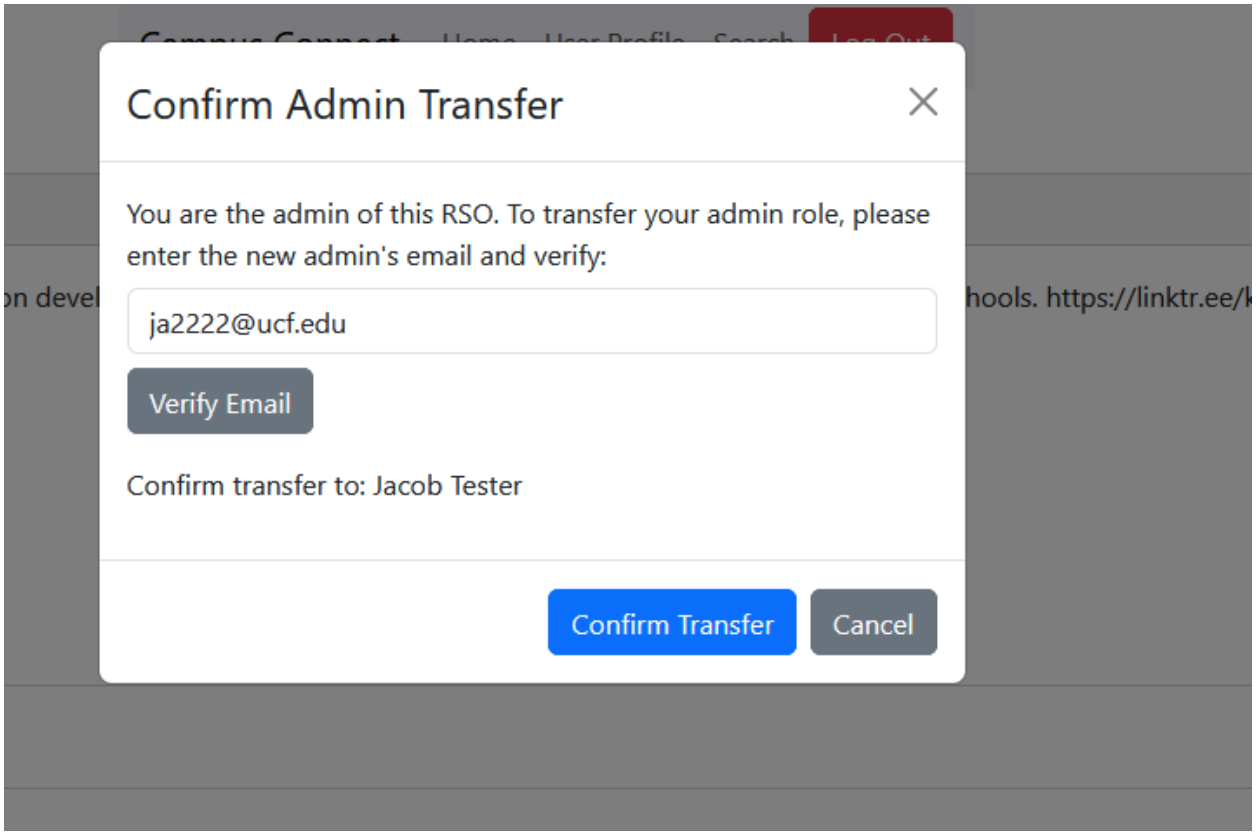


And must not be admin of another RSO

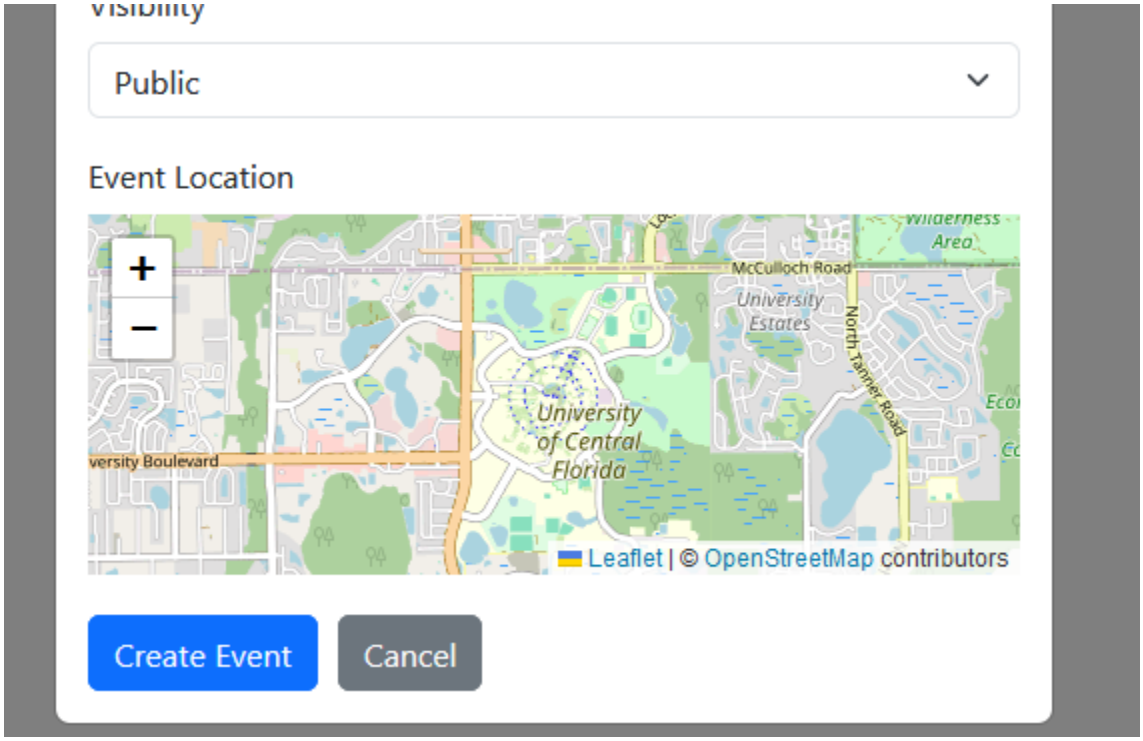


Otherwise it displays the users name for assurance your giving administration role to

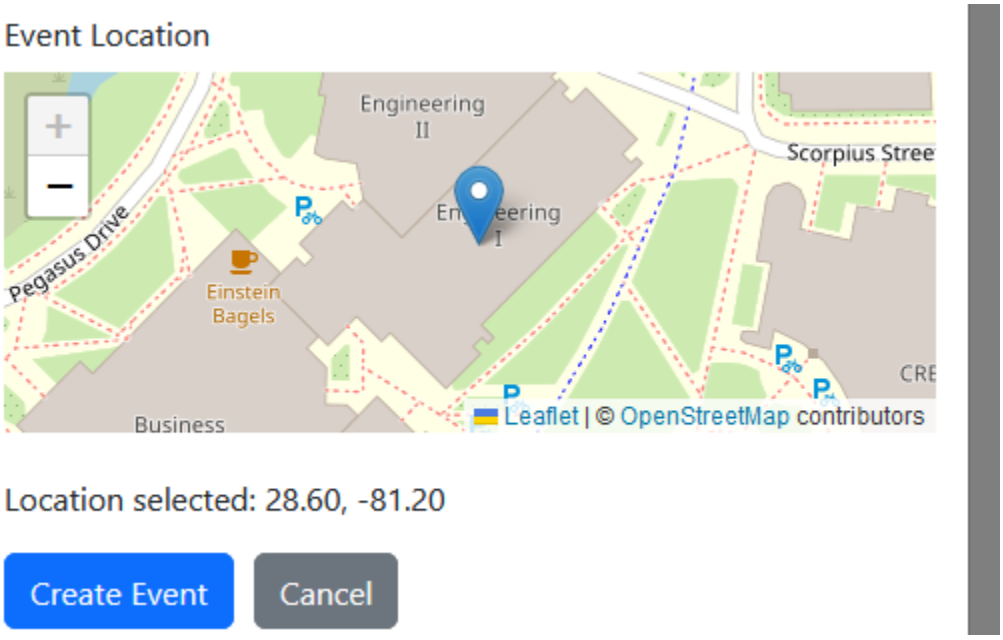
the right person



Advanced Features



Added ability to put pin on the map with lat and lon coordinates. Along with a location section for more detail such as room number, hallway directions, ect



Conclusions

Database performance:

I found query response time reasonable fast and had no issues having multiple queries happen for different checks as pages load.

Desired features:

I liked being able to add the map as it gives a better visual especially for longer range compared to just a building name and number it helps verify for locations possibly outside the standard buildings and like in memory mall.

Also being able to test using hashed passwords so that it can't just be copied from the database.

Problems encountered:

In retrospect I would have tried to separate things more clearly I tried to make forms and functions that could be used multiple times such as (create form is also the edit form) but it didn't work all the time and needed more reworks and taken more time than I would have liked.