CS 432 - DATABASES

ASSIGNMENT -4

TEAM: 8 BITS

3.1 Responsibility of G1:

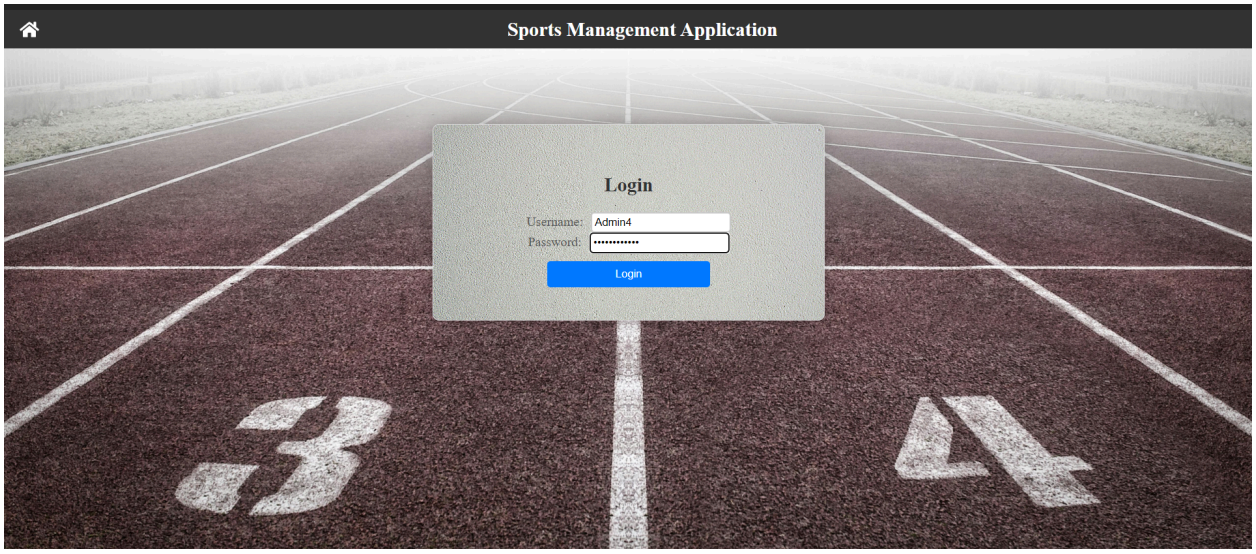
3.1.1:

The feedback form has been submitted, and screenshots of changes as per the feedback have been added below.

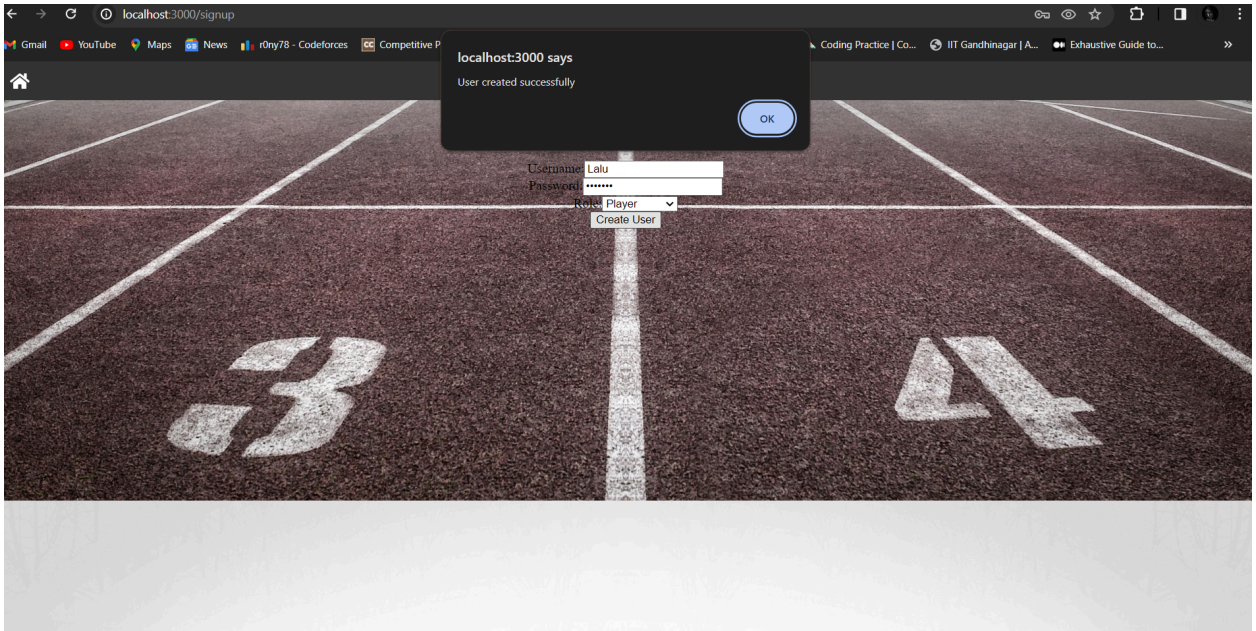
1. **Before the feedback**

   No validation checks in email and password

   Login is used to be via username earlier



There is no Validation for Creating a User.



No Search Bar

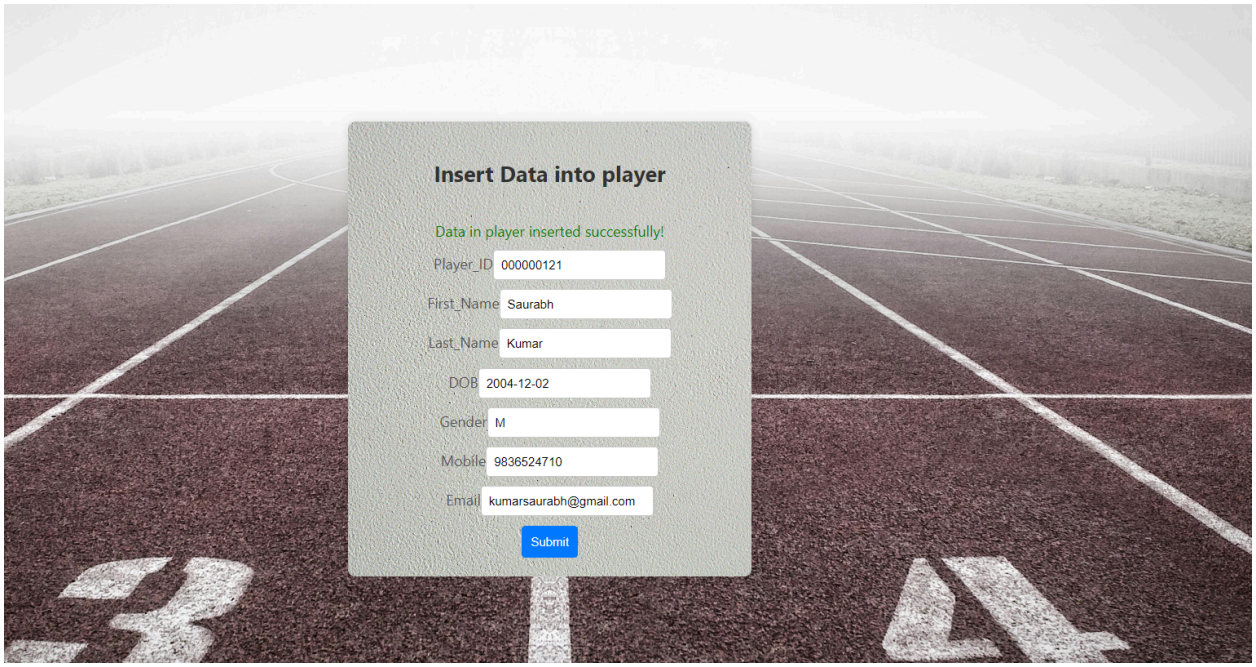
Sports Management Application

team

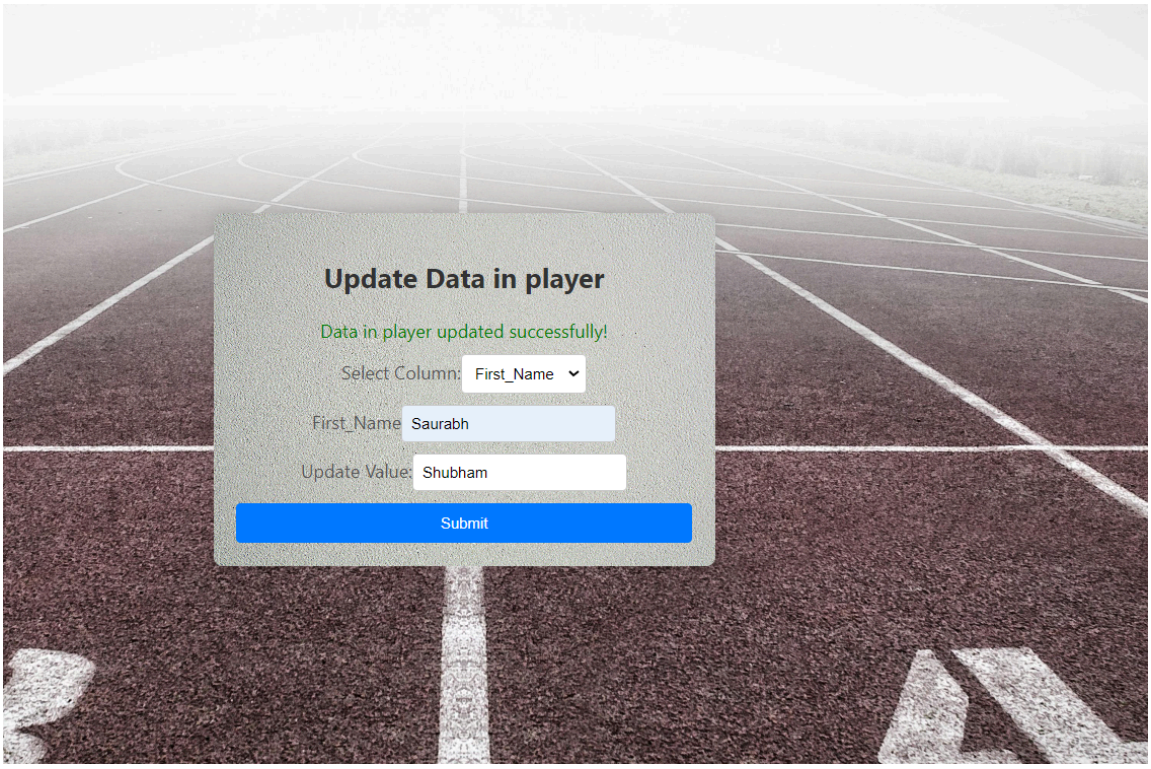
Insert Data | Delete Data | Update Data | Rename Data | Where Clause

Team_ID	Team_Name	Captain_ID	Sports_Name
200001	The Avengers		Indian_Cricket
200002	The Ninjas		Badminton
200003	The Panthers		Hockey
200004	The Knights	100653	Cycling
200005	The Stars	100854	Boxing
200006	The Pirates	101055	Volleyball
200007	The Magicians	101256	Basketball
200008	The Spartans	101457	Football
200009	The Kings	101545	Tennis
200010	The Rockets	101644	Athletics
200011	The Warriors	102141	Indian_Cricket
200012	The Dragons	102537	Badminton
200013	The Tigers	102839	Hockey

Data insertion without any checks and validation

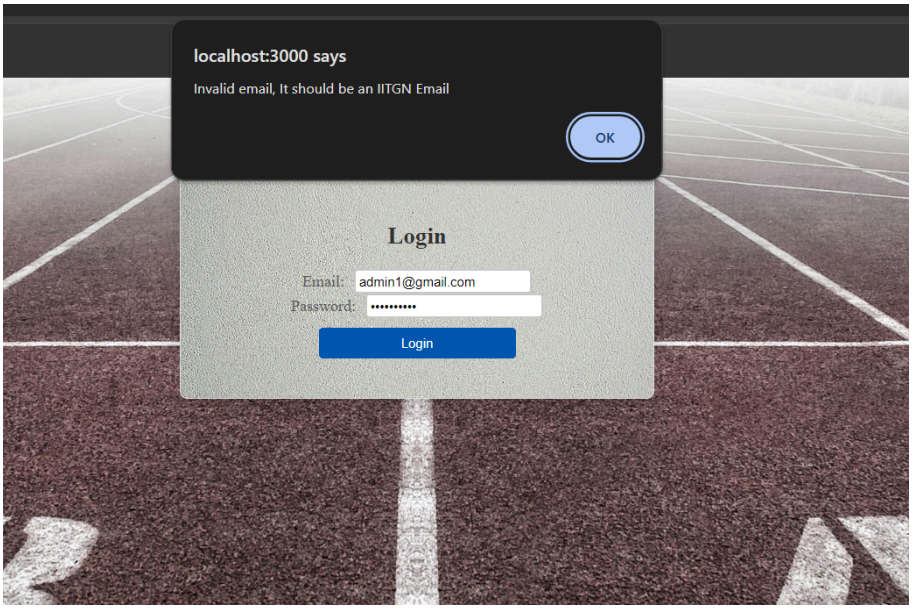


No validation in the Data update

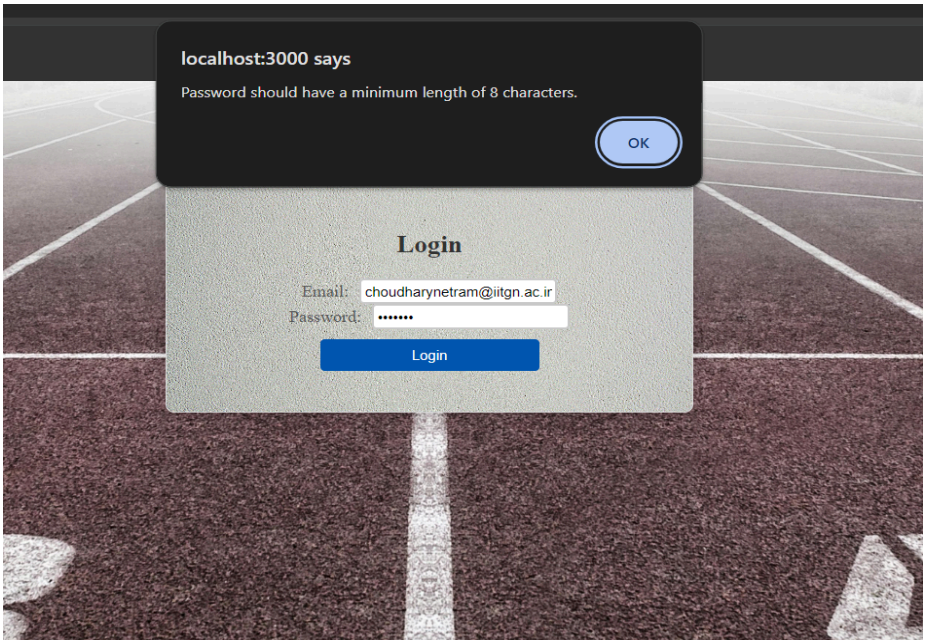


2. **After the feedback:**

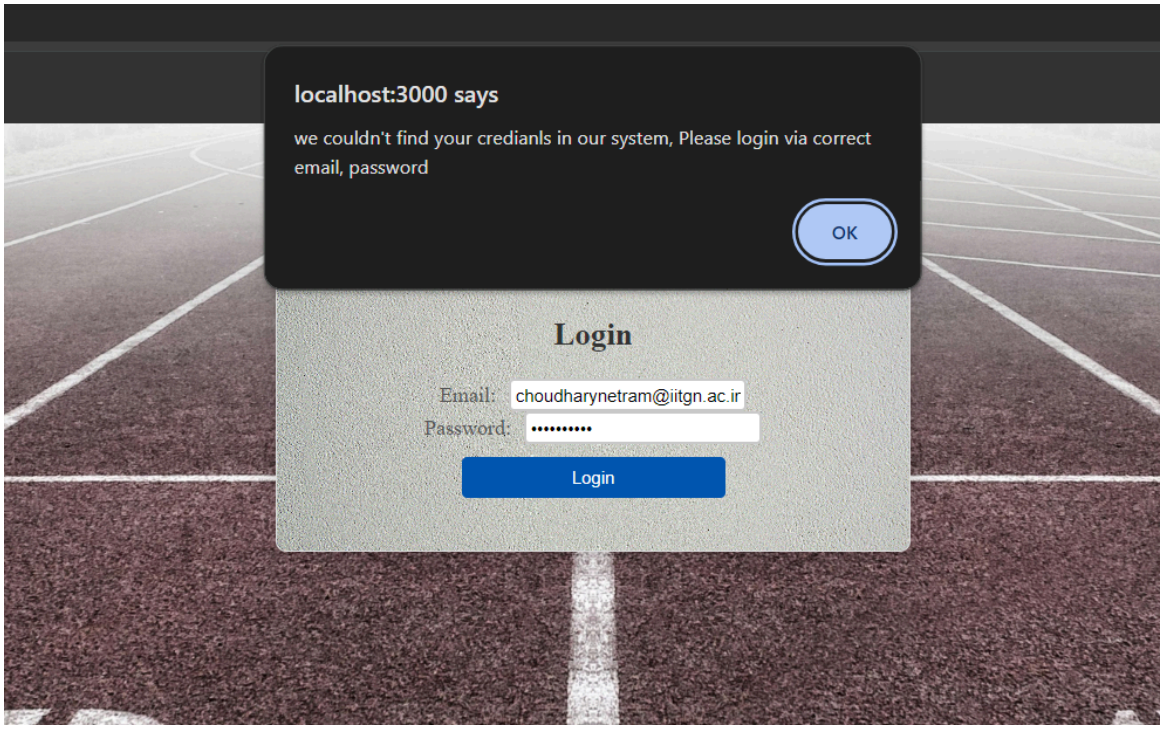
Login can only be done with the IITGN Email ID.



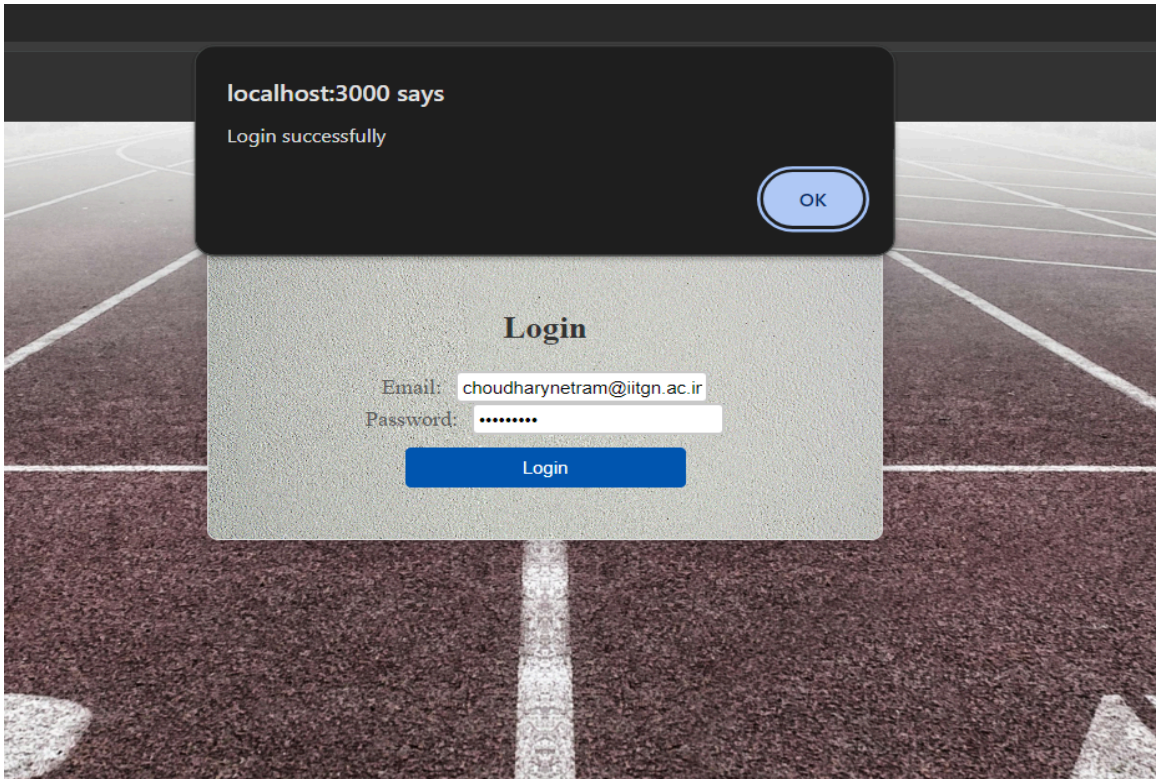
Validation on password, for making password strong:



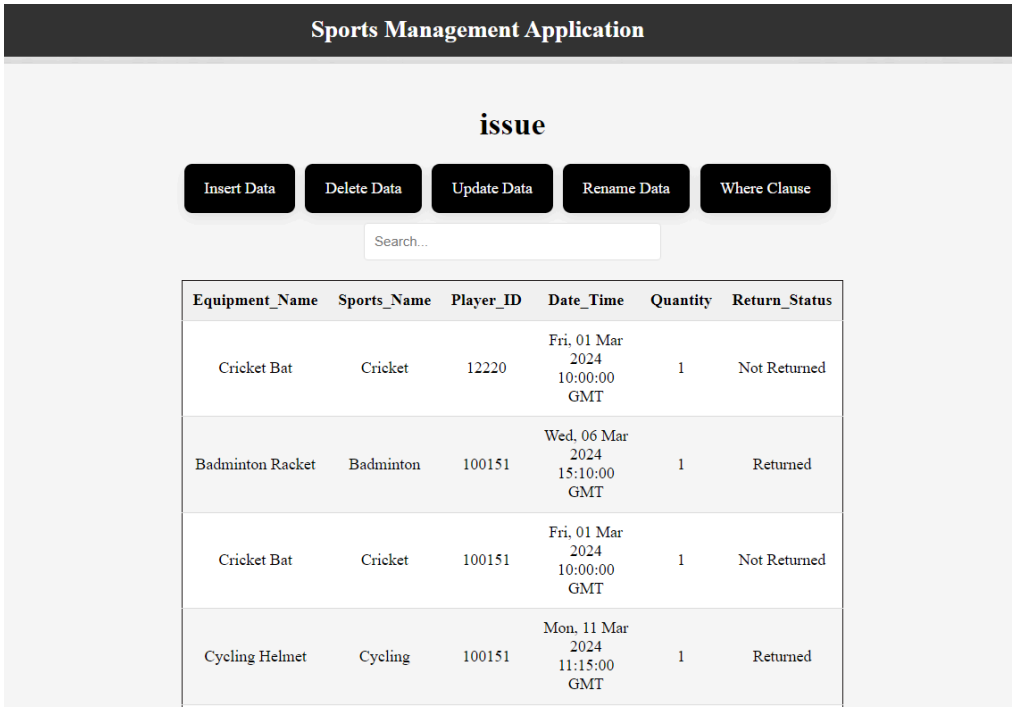
If your email and password didn't signup earlier, then the error:



If the Login credentials are correct:

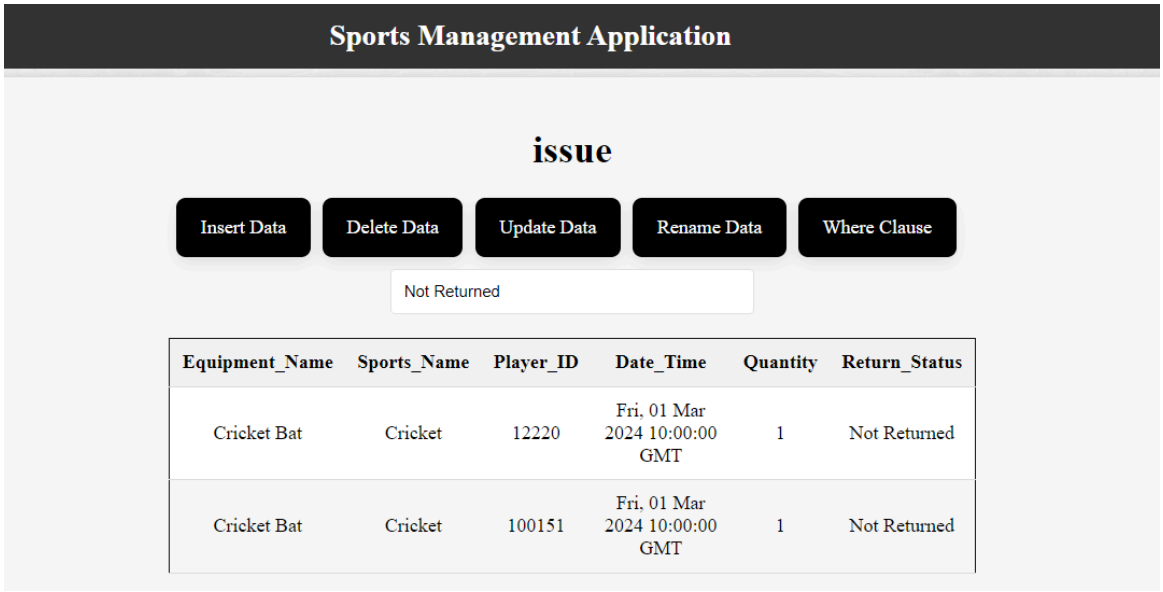


Search bar implemented:

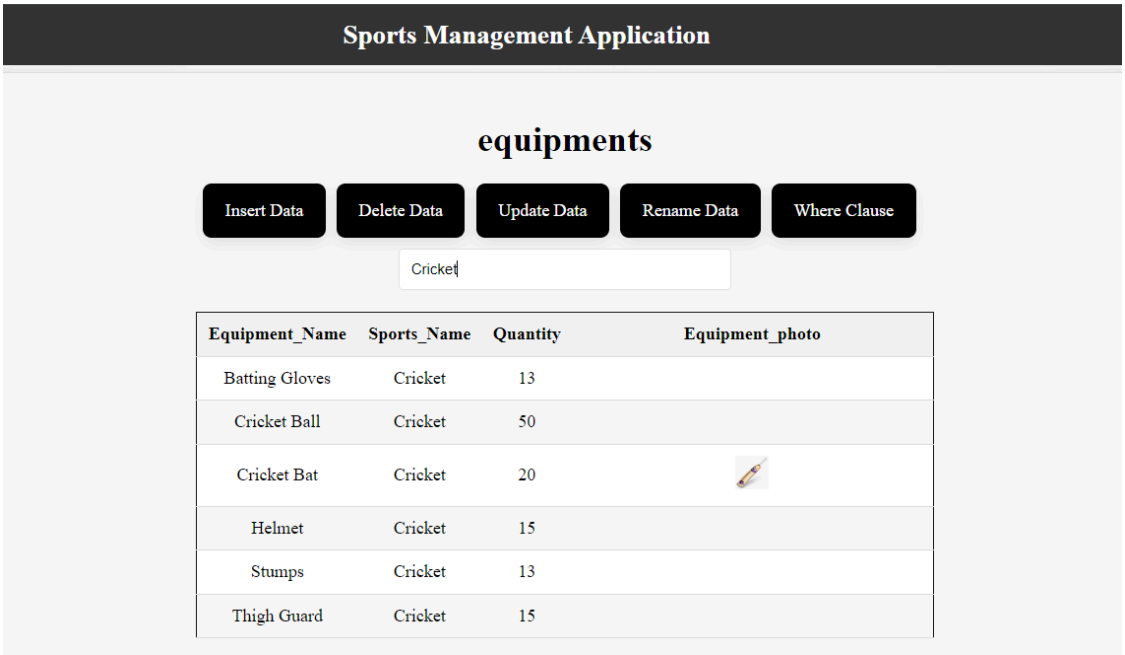


Equipment_Name	Sports_Name	Player_ID	Date_Time	Quantity	Return_Status
Cricket Bat	Cricket	12220	Fri, 01 Mar 2024 10:00:00 GMT	1	Not Returned
Badminton Racket	Badminton	100151	Wed, 06 Mar 2024 15:10:00 GMT	1	Returned
Cricket Bat	Cricket	100151	Fri, 01 Mar 2024 10:00:00 GMT	1	Not Returned
Cycling Helmet	Cycling	100151	Mon, 11 Mar 2024 11:15:00 GMT	1	Returned

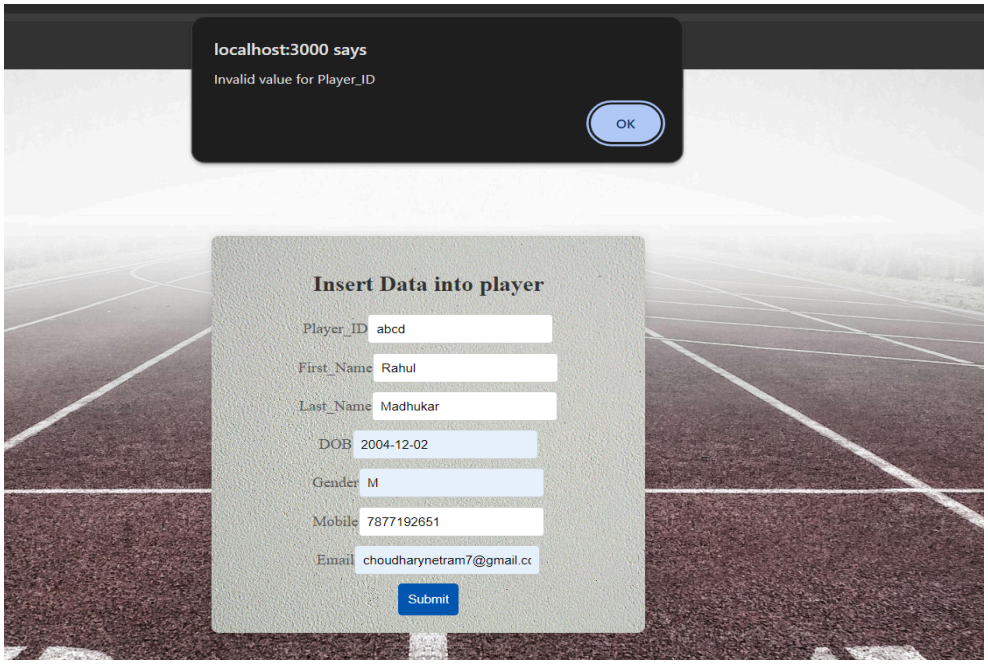
Searching those Issued equipment that hasn't been returned yet:



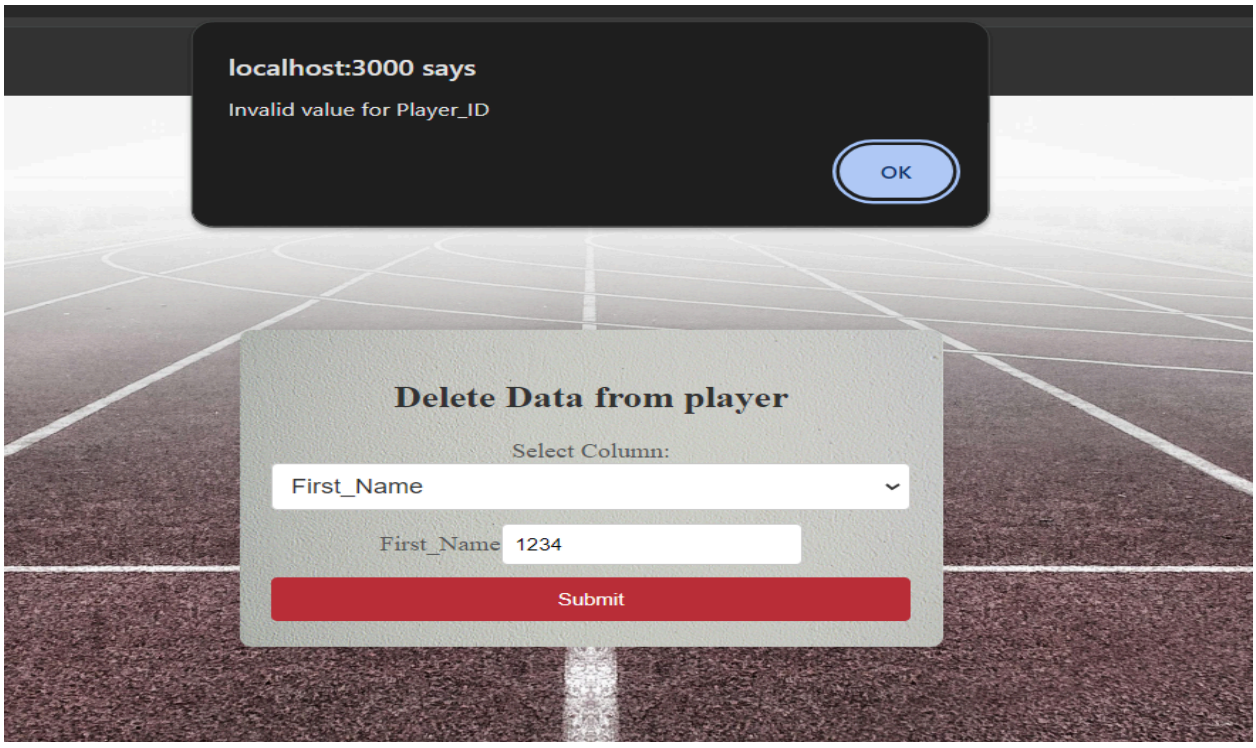
Searching all equipment of Cricket Sport:



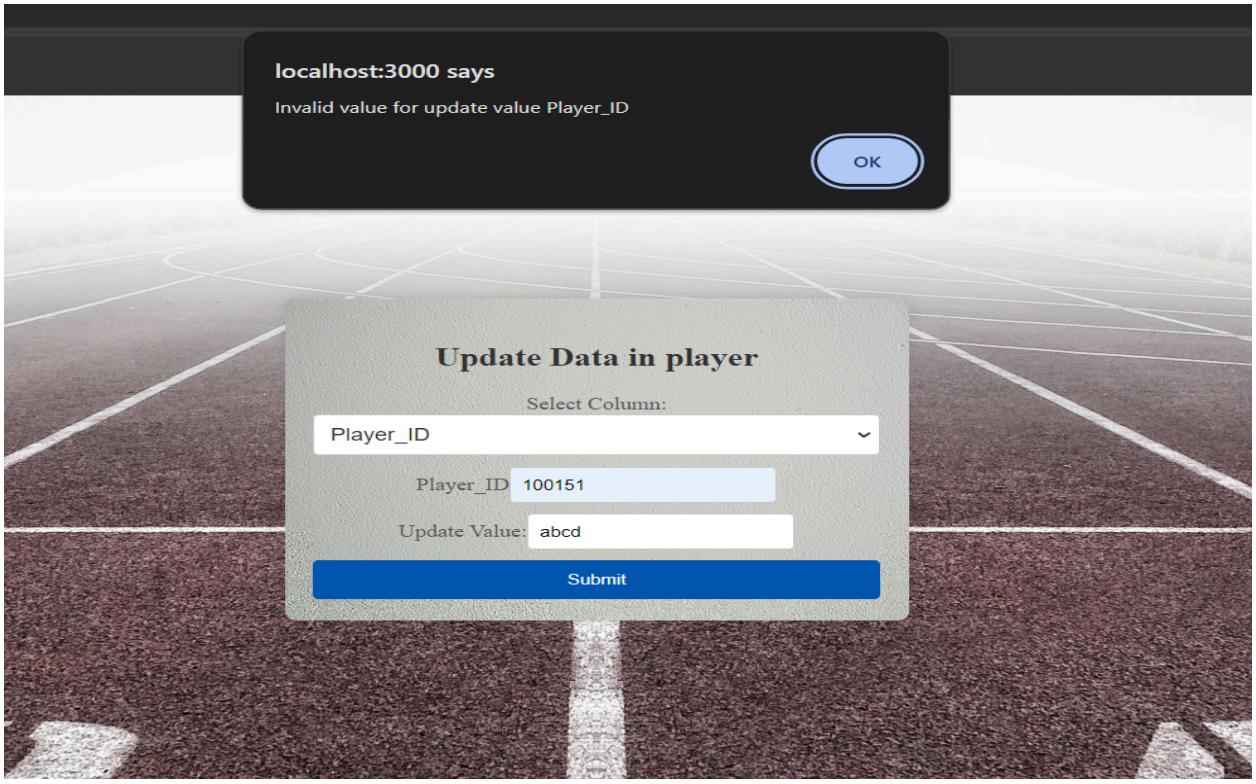
Implemented type checks for all columns of all tables.
Checking the type of input that the user is trying to parse:
Below Player_ID should be in int, but if the user parses a not int value, then an error is coming



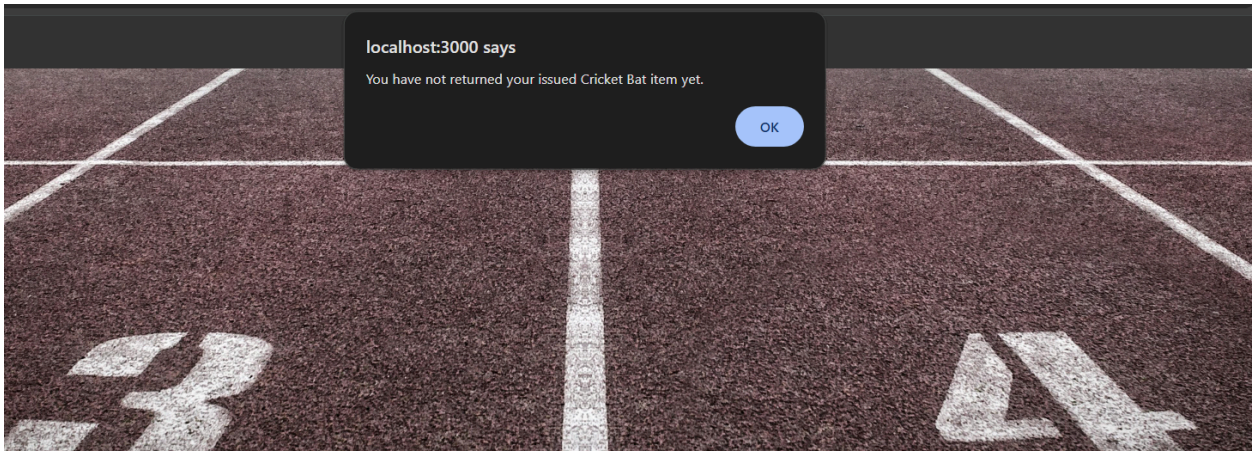
Checking the type of First_Name while doing the Delete operation:



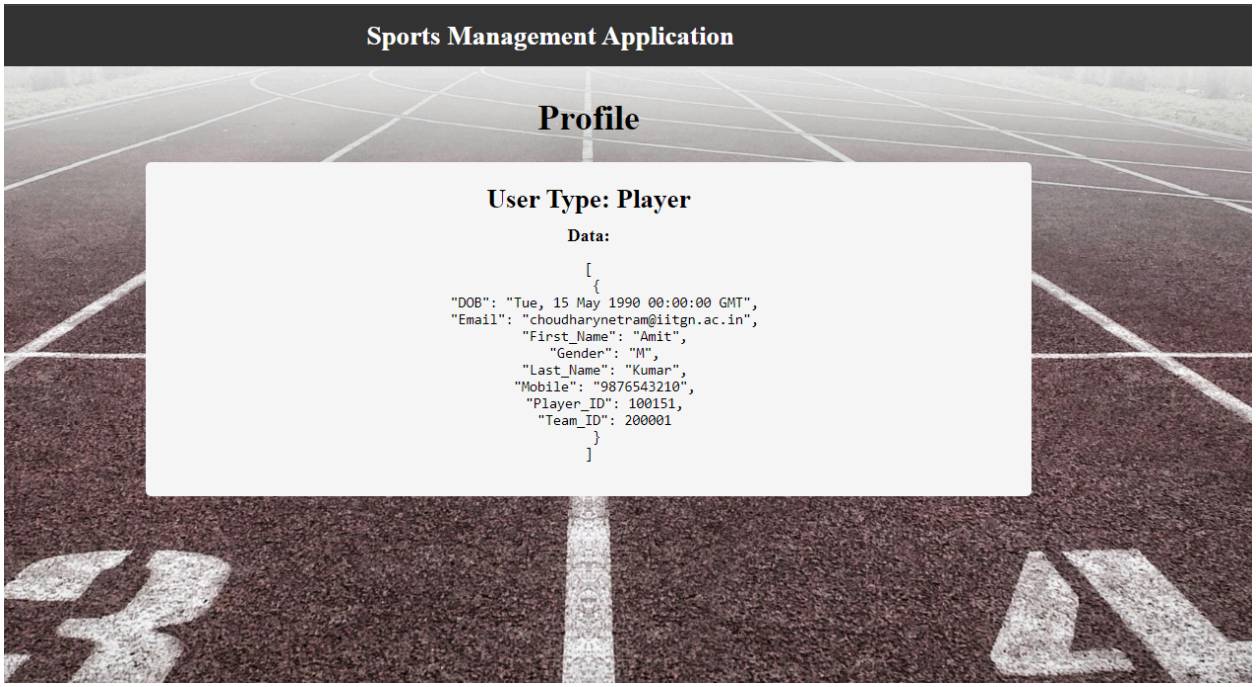
Checking the type of updated values in the Update operation:



Notification message via application to the user that hasn't returned their issued equipment:



User Profile Page:



3.1.2:

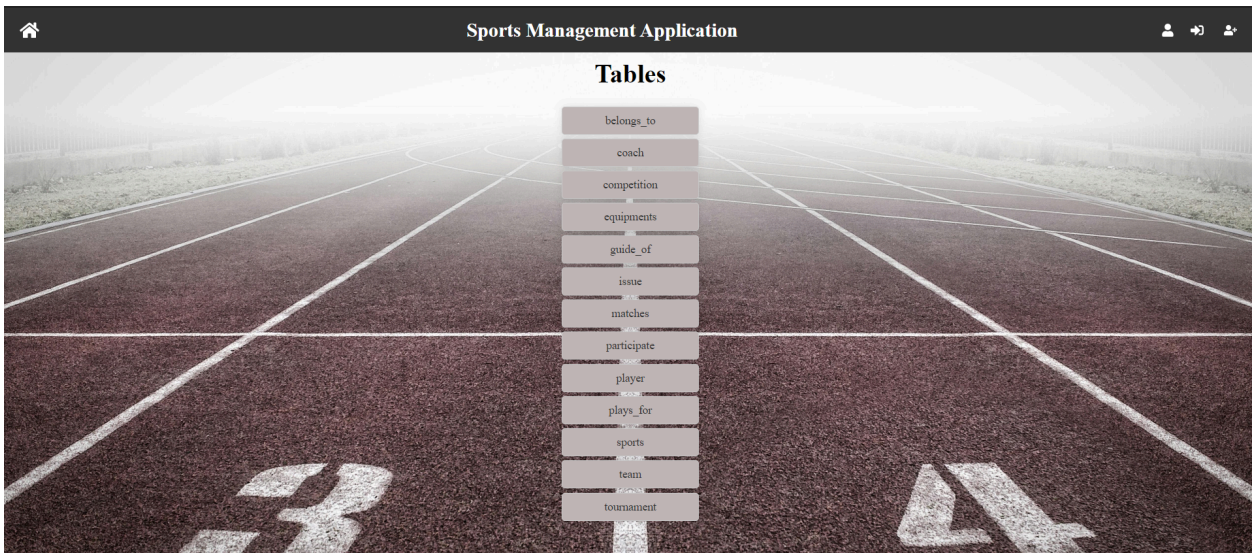
For our webpage, we have three users:

1. Admin
2. Player
3. Coach

Admin:

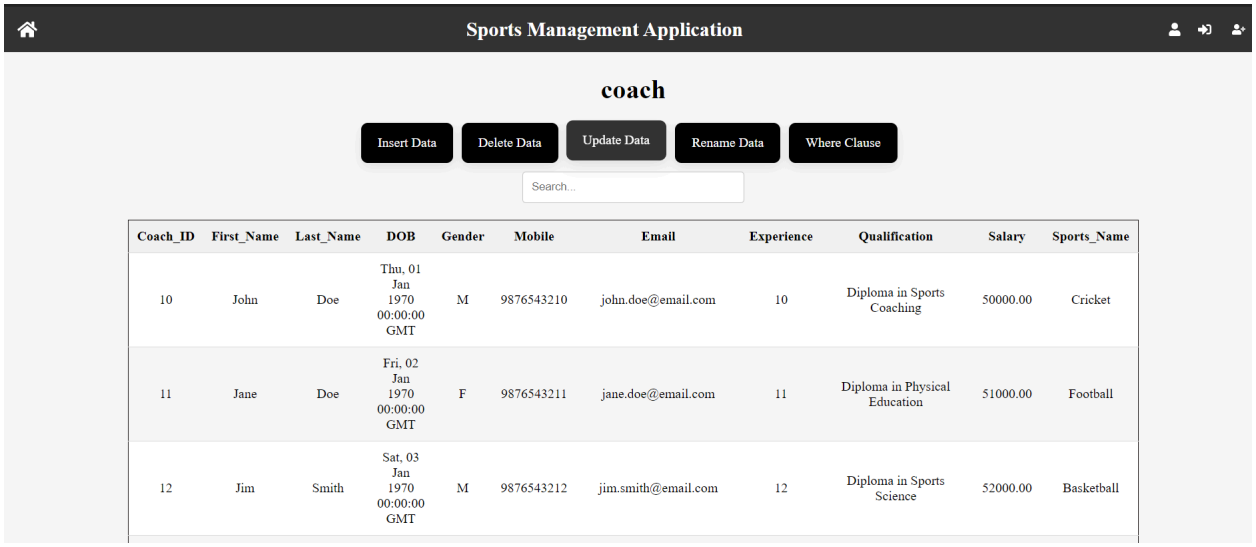
For the admin, we have created an Admin panel. In which he can see the data of all tables and edit it. Our admin has the INSERT, UPDATE, DELETE, and SELECT privileges.

The screenshots below are views for the admin.

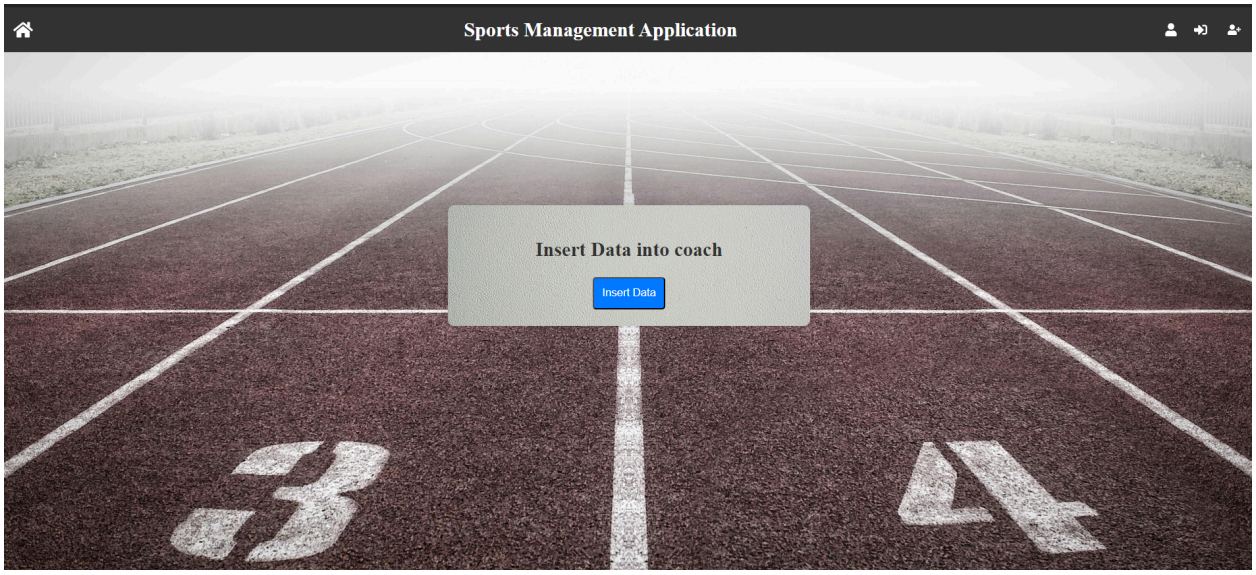


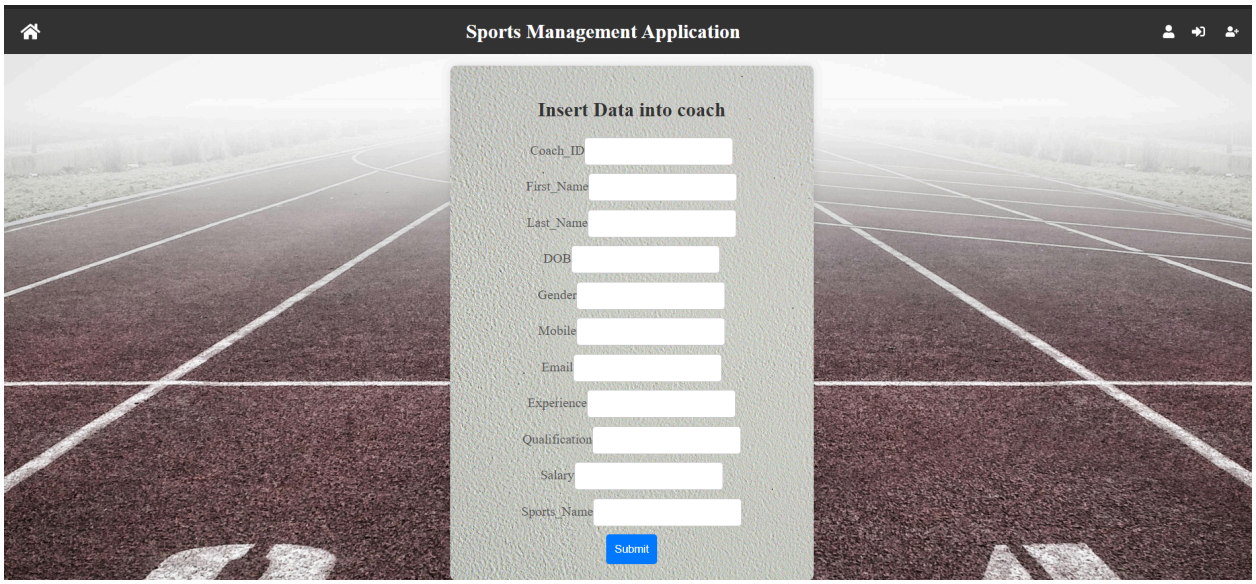
Just after login, the admin can see all the tables that we have in our database. Let's say Admin clicks on

the Coach table, and then it will show all the data for the table Coach.



Here, the Admin can see the option to insert, update, and make other changes. If I open the insert, it will show this:

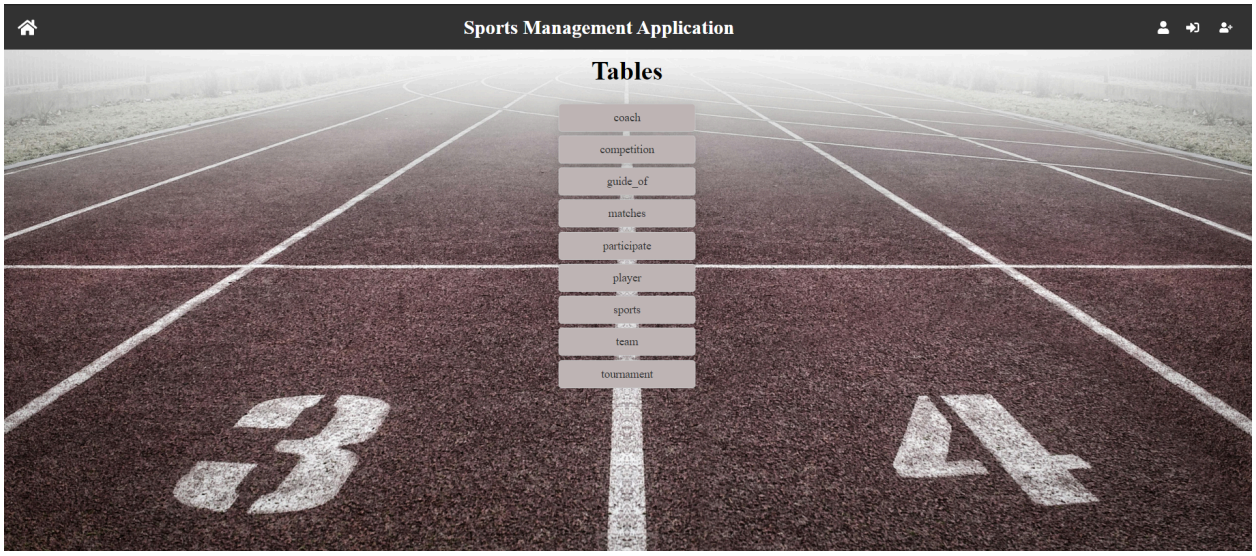




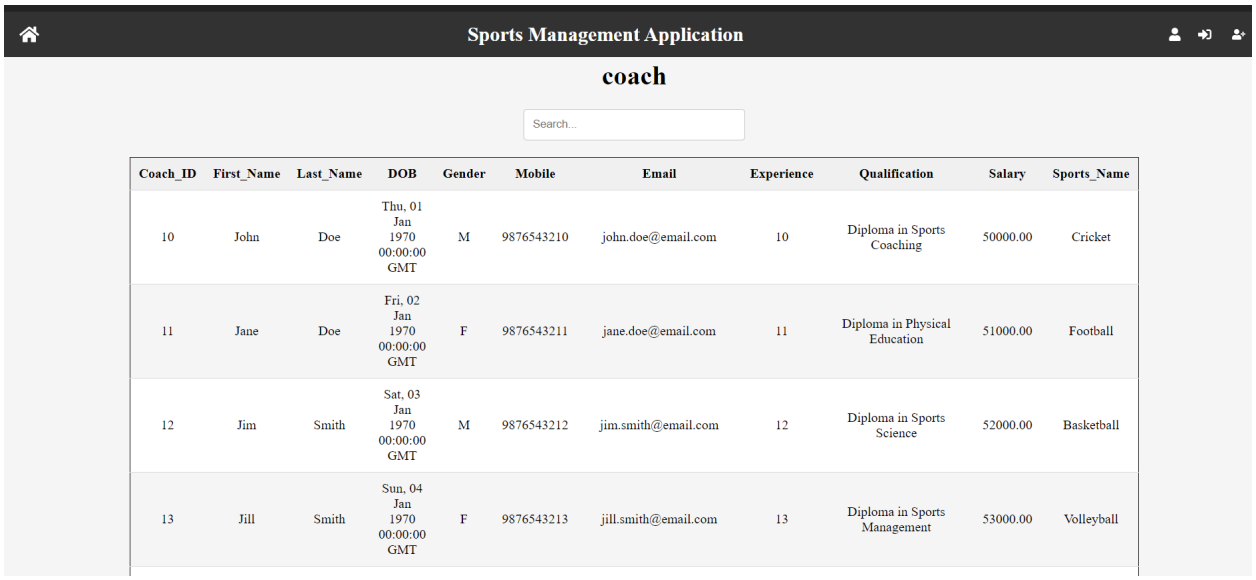
Player:

For the Player, he can see the data of permitted tables. Our player has the SELECT privilege on some limited tables.

The screenshots below are views for the Player.



After login, the Player can see the permitted tables. Let's say the Player clicks on the Coach table, and then it will show all the data for the table Coach.



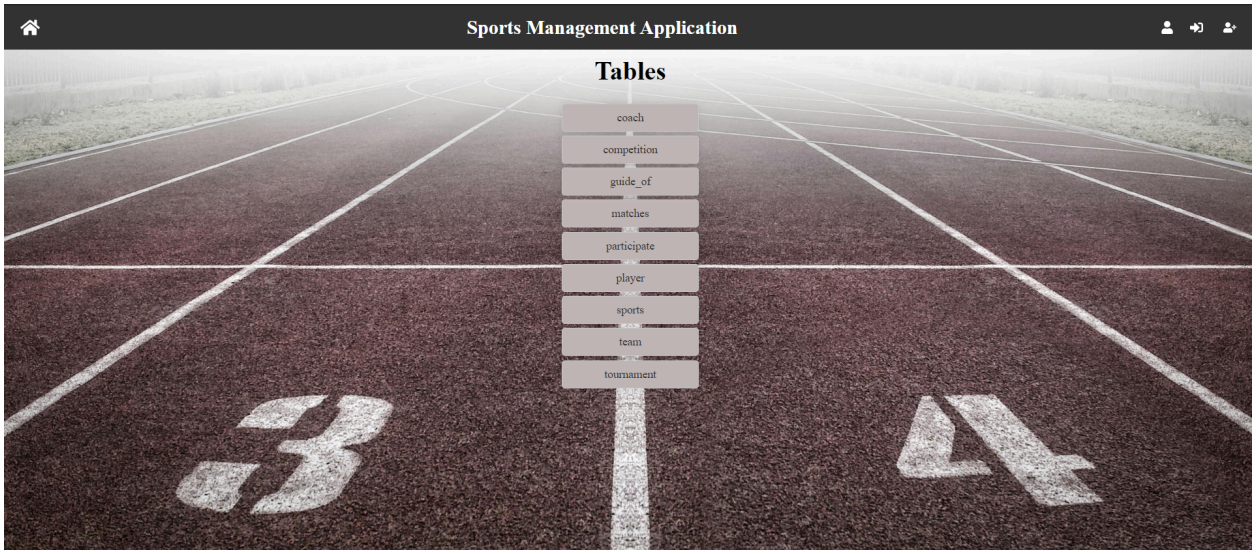
Coach_ID	First_Name	Last_Name	DOB	Gender	Mobile	Email	Experience	Qualification	Salary	Sports_Name
10	John	Doe	Thu, 01 Jan 1970 00:00:00 GMT	M	9876543210	john.doe@email.com	10	Diploma in Sports Coaching	50000.00	Cricket
11	Jane	Doe	Fri, 02 Jan 1970 00:00:00 GMT	F	9876543211	jane.doe@email.com	11	Diploma in Physical Education	51000.00	Football
12	Jim	Smith	Sat, 03 Jan 1970 00:00:00 GMT	M	9876543212	jim.smith@email.com	12	Diploma in Sports Science	52000.00	Basketball
13	Jill	Smith	Sun, 04 Jan 1970 00:00:00 GMT	F	9876543213	jill.smith@email.com	13	Diploma in Sports Management	53000.00	Volleyball

The player can see the table's data but can not perform insert and update-like queries also, he cannot see the option for it.

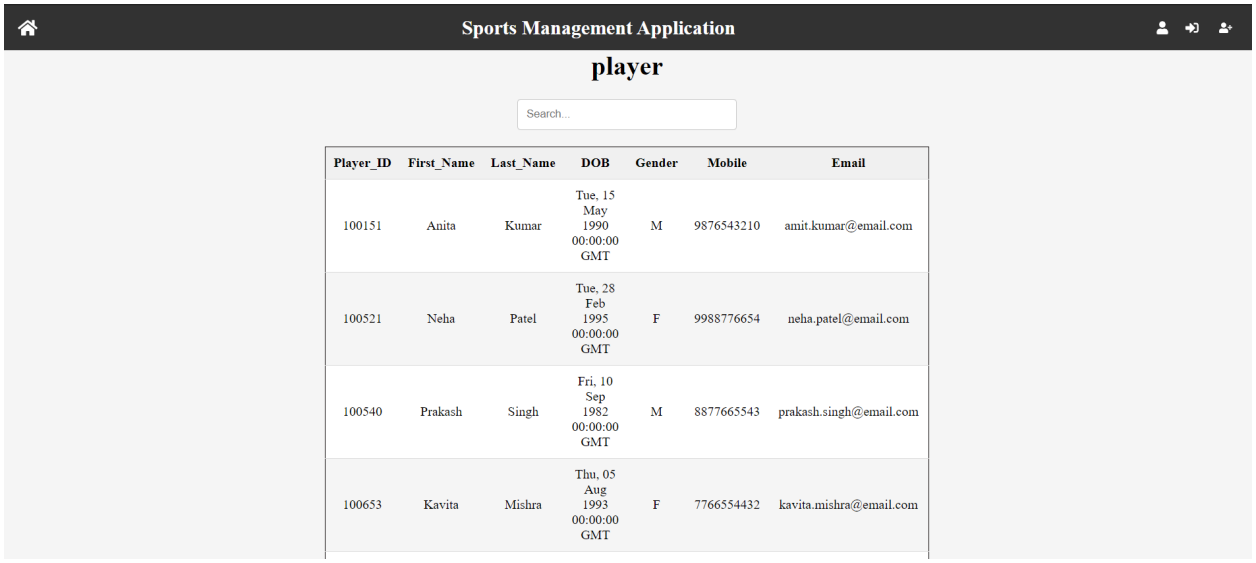
Coach:

The coach can see the data for permitted tables. Our Coach has the SELECT privilege on some limited tables.

The screenshots below are views for the Coach.



After logging in, the Coach can see the permitted tables. Let's say the Coach clicks on the player table, and then it will show all the data for the table player.



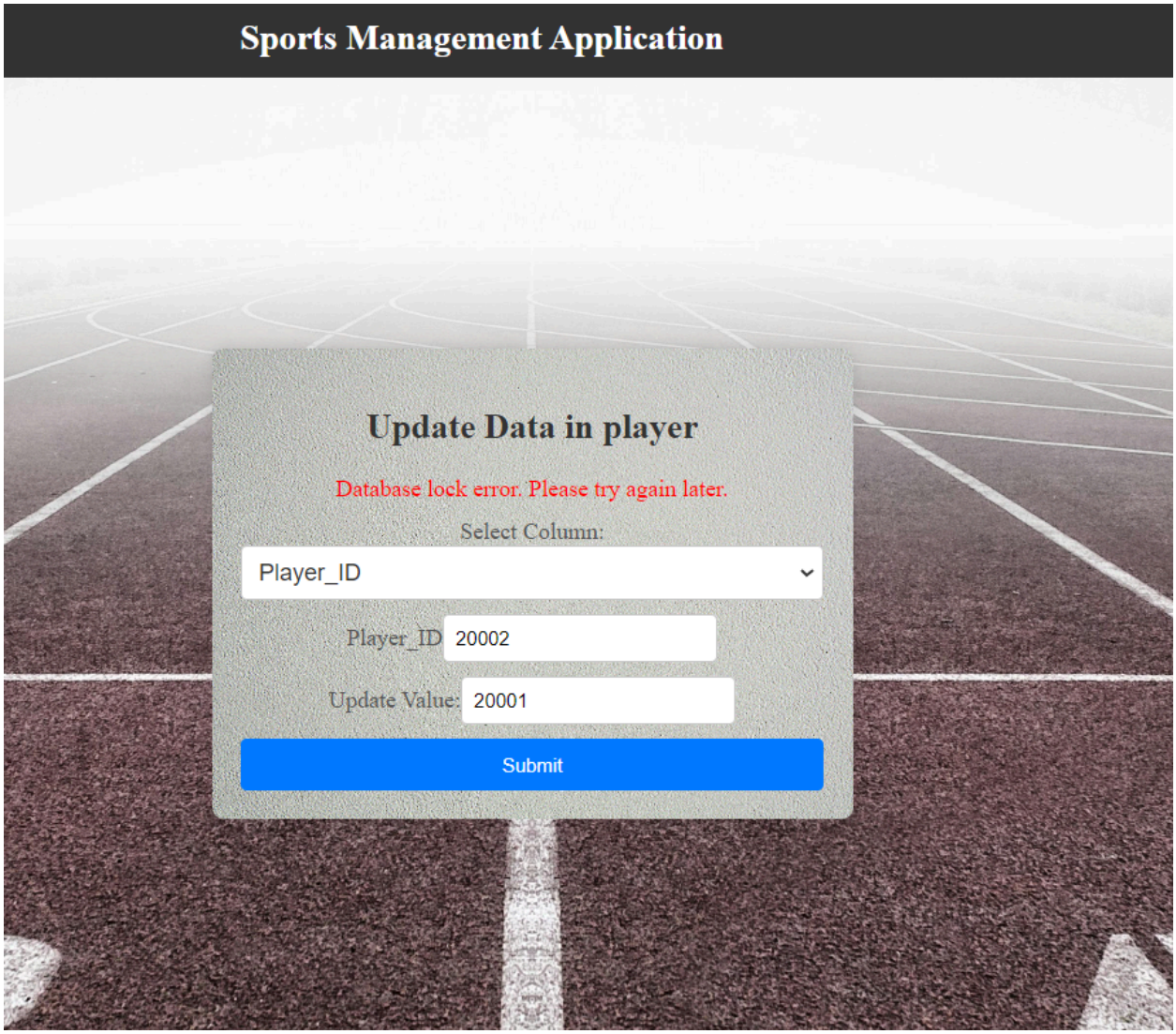
The Coach can see the table's data but can not perform insert and update-like queries also, he cannot see the option for it.

3.2 Responsibility of G2:

3.2.1:

Concurrent multi-user access:

For testing concurrent multi-user access. We locked the Player table, tried to perform an update operation, and got the below error.



Functions for locking table and unlocking table:

```python
def lock_table(table_name, lock_type):
    cursor = mysql.connection.cursor()
    cursor.execute(f"LOCK TABLES {table_name} {lock_type}")
    cursor.close()
    logging.debug(f"Table {table_name} locked successfully")

def unlock_tables():
    cursor = mysql.connection.cursor()
    cursor.execute("UNLOCK TABLES")
    cursor.close()
    logging.debug("Tables unlocked successfully")
```

3.2.2:

No changes were made to the relations, but there is a change with the constraint of the user table that only people with IITGN email ID can sign up and Log in. These changes are made after feedback is received.

```sql
CREATE TABLE Users (
    useremail VARCHAR(50) PRIMARY KEY UNIQUE,
    password VARCHAR(50),
    role VARCHAR(50),
    CONSTRAINT chk_email_format CHECK (useremail REGEXP '^[^\s@]+@iitgn\.ac\.in$')
);
```

```sql
12
13 •   select * from Users;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

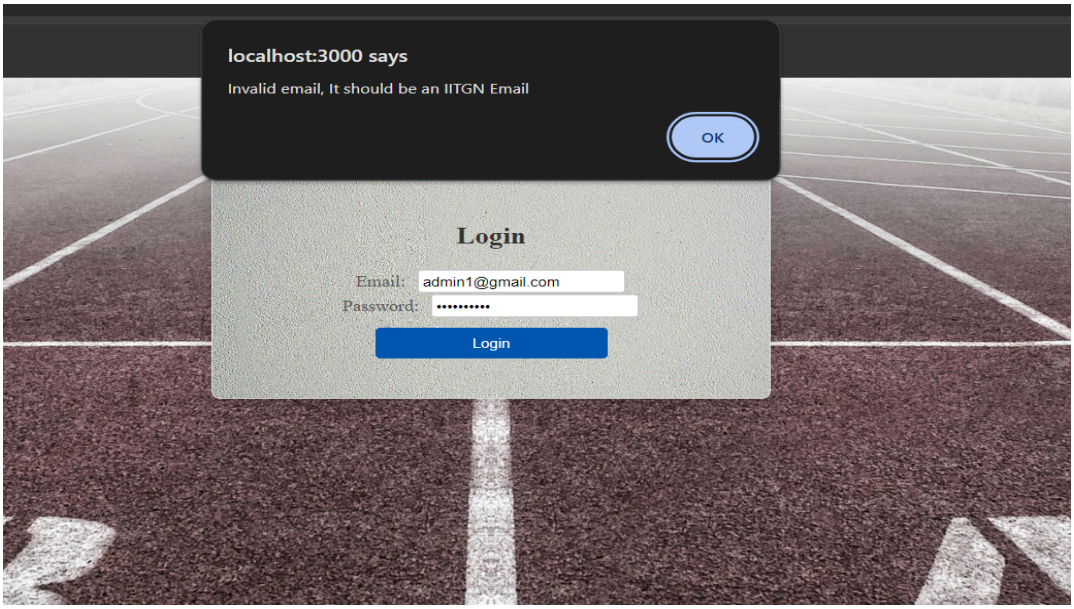
useremail	password	role
choudharynetram@iitgn.ac.in	Netram123	admin
choudharynetram7@iitgn.ac.in	Saurabh123	Coach
madhukarrahul@iitgn.ac.in	Rahul@123	Player
NULL	NULL	NULL

No significant changes were made to the database. While taking feedback from stakeholders, there were no suggestions about changing the database, only changes in the application.

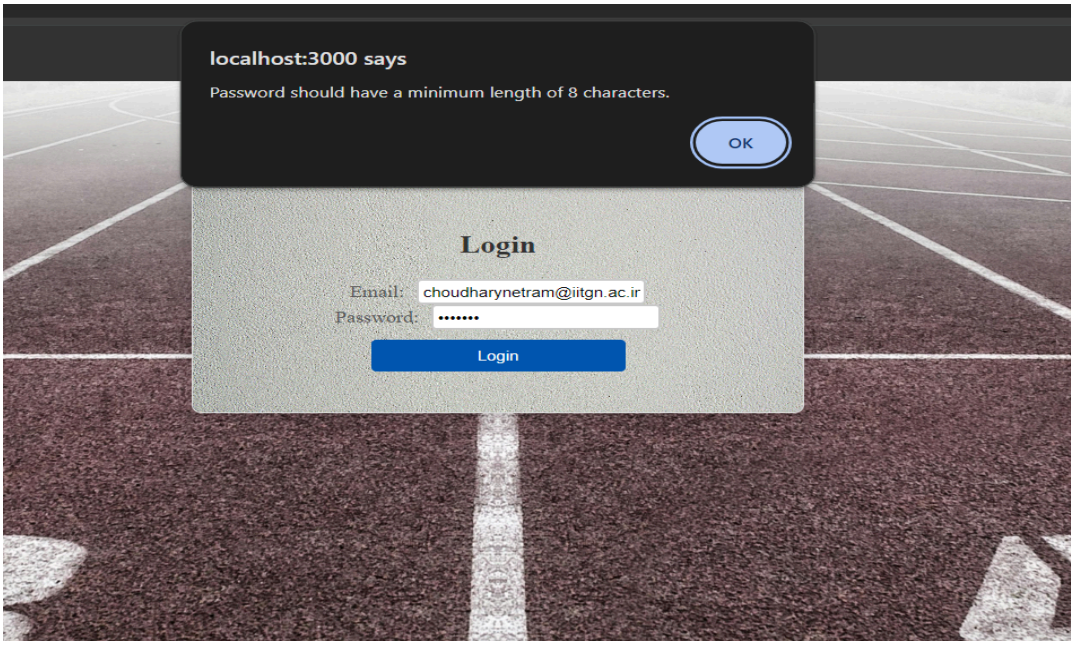
3.2.3:

Adding authentication for login and registration (only IITGN users can log in and register):

Login can only be done with the IITGN Email ID



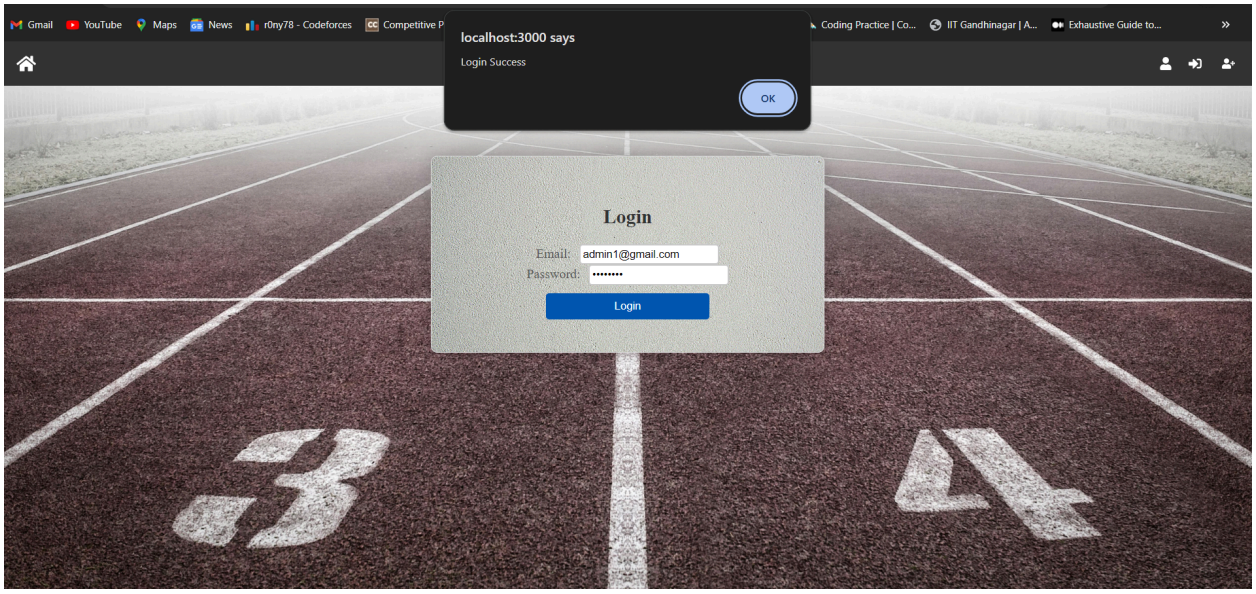
Validation on password, for making password strong:



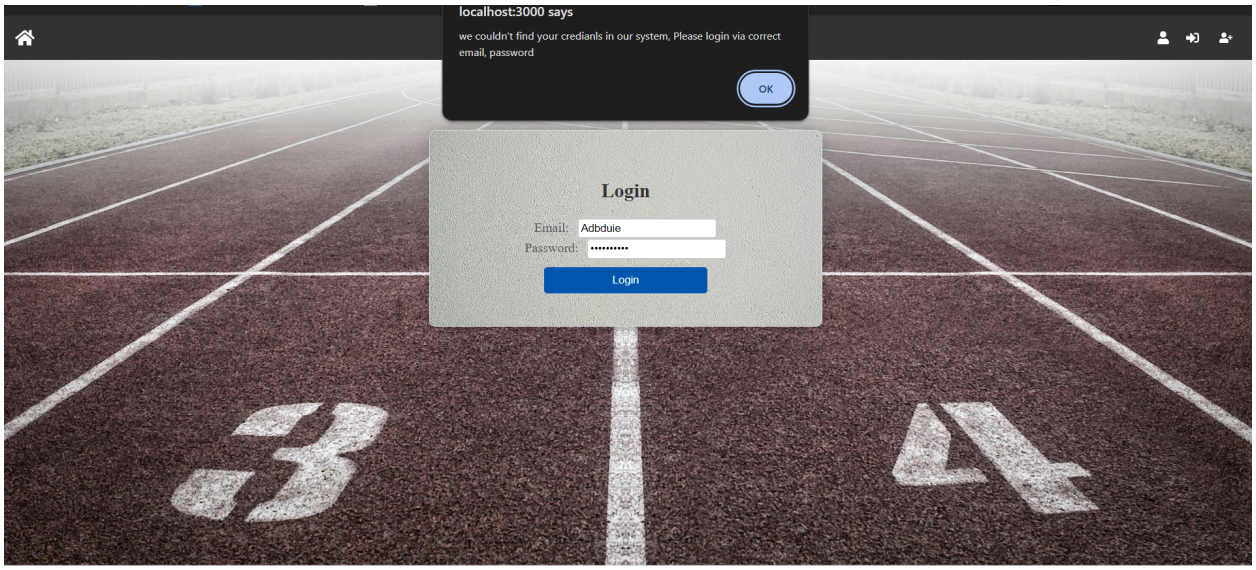
3.3 Responsibility of G1 & G2:

3.3.1:

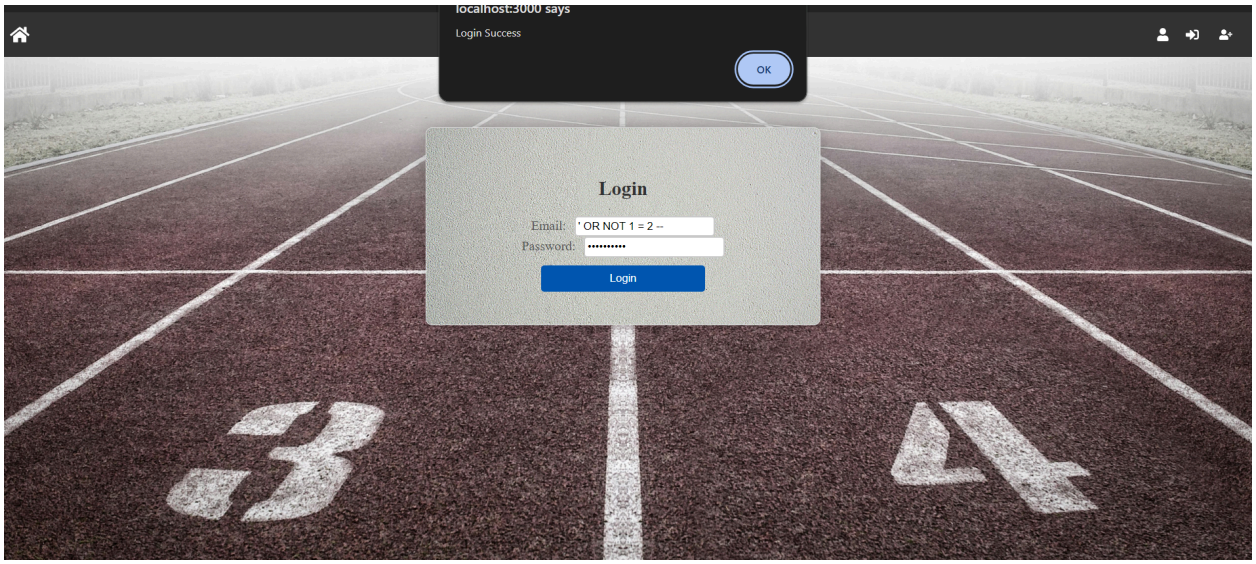
1. **SQL Injection:**



I logged in with the correct credentials and saw that alert about a successful login.



I tried logging in with the incorrect credentials and saw an alert that credentials could not be found.



I used SQL injection to log in where I used Email : “‘ OR NOT 1 = 2 –” and some random password, and we can see that this worked.

Preventing SQL injection:

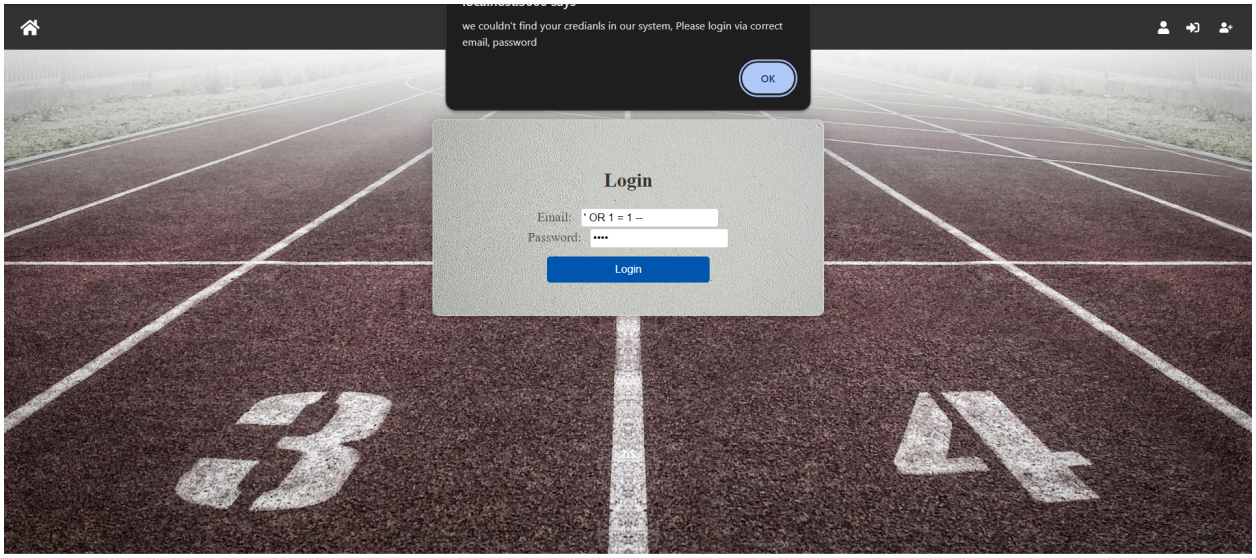
First Way: SQL works because we execute it in a simple string form or less secure form. In our code, we are executing queries like this, which is a terrible practice.

```
cur = mysql.connection.cursor()
print("1")
# Check if the user exists in the users table
cur.execute("SELECT * FROM Users")
print("5")
mysql.connection.commit()
cur.execute("SELECT * FROM Users WHERE useremail = '{}' AND password = '{}'".format(useremail, password))
print("2")
mysql.connection.commit()
```

To prevent SQL injection, we should use the parameterized query:

```
cur = mysql.connection.cursor()
print("1")
# Check if the user exists in the users table
cur.execute("SELECT * FROM Users")
print("5")
mysql.connection.commit()
cur.execute("SELECT * FROM Users WHERE useremail = %s AND password = %S",(useremail, password))
print("2")
mysql.connection.commit()
user_data = cur.fetchone()
```

After this change, let's try SQL injection again:

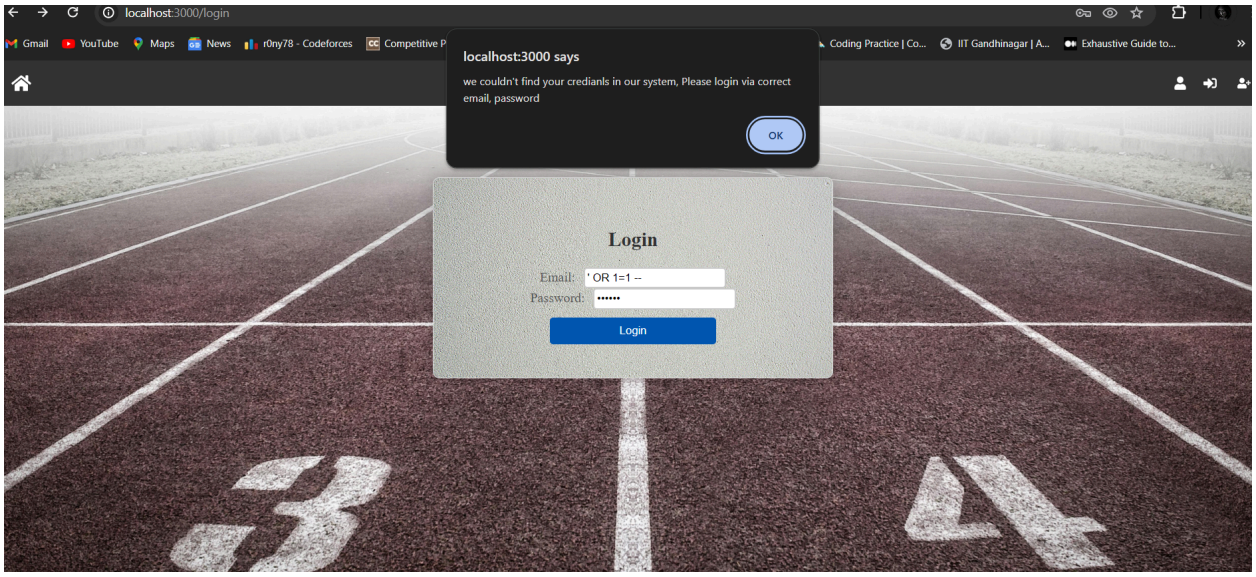


As we can see, we are getting an alert that credentials cannot be found.

Second Way: The second way to prevent SQL injection is for the login never to connect to the database as the "root" user.

```
set_mysql_credentials(useremail, password)
cur = mysql.connection.cursor()
print("1")
# Check if the user exists in the users table
cur.execute("SELECT * FROM Users")
print("5")
mysql.connection.commit()
cur.execute("SELECT * FROM Users WHERE useremail = '{}' AND password = '{}'".format(useremail, password))
```

As you can see, I am using an insecure format of query execution, but I am connecting the database using the credentials we are getting from the user. It will perform all queries according to this user, but there is no user by this name. So, SQL injection should not work.



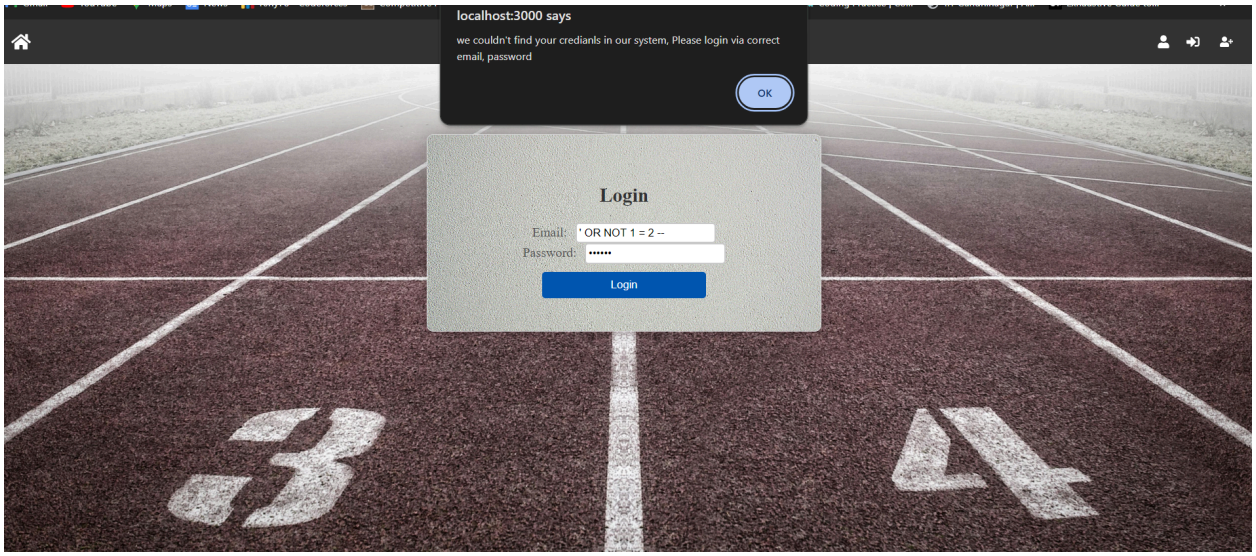
As we can see, even though I was using an insured way of writing queries, we got the error that the

credentials could not be found.

After changing both ways, the Code looks like this:

```
useremail = request.json.get('useremail')
password = request.json.get('password')
# to prevent the sql injection
set_mysql_credentials(useremail, password)
cur = mysql.connection.cursor()
print("1")
# Check if the user exists in the users table
cur.execute("SELECT * FROM Users")
print("5")
mysql.connection.commit()
cur.execute("SELECT * FROM Users WHERE useremail = %s AND password = %s",(useremail, password))
print("2")
mysql.connection.commit()
```

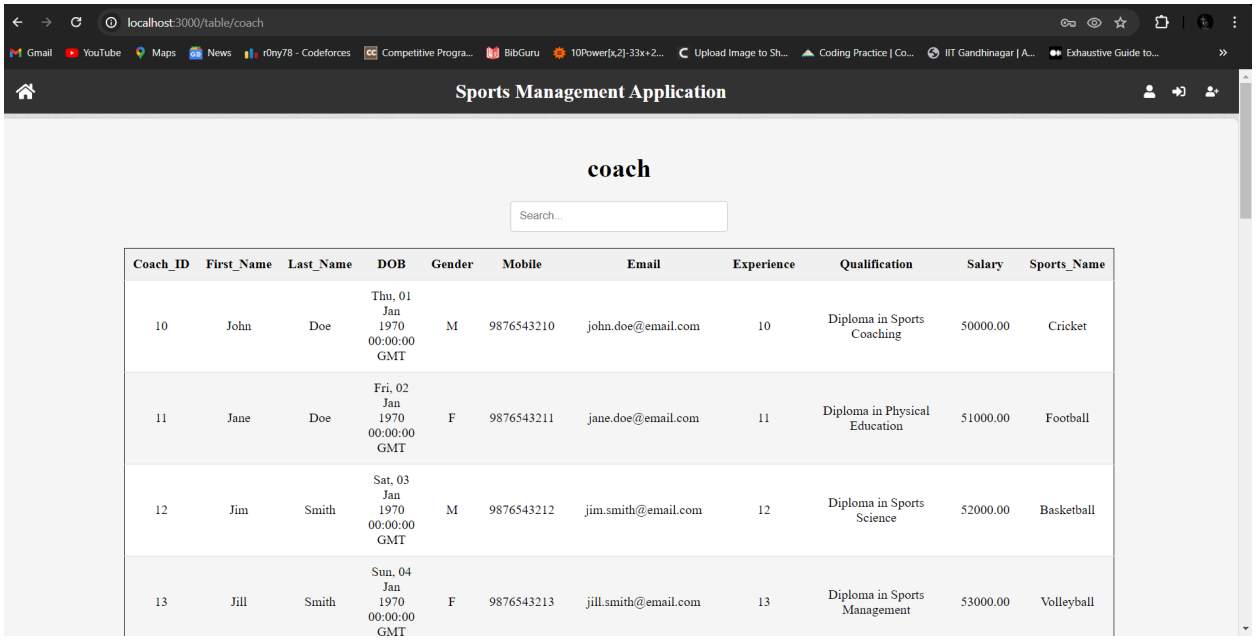
Let's perform aging SQL injection:



SQL injection is not working. So, we successfully prevented the SQL injection.

2. URL attack:

This error has been handled from the start of development. The meaning of this attack is that when I put the direct URL of my table without login, It will show the table's data.



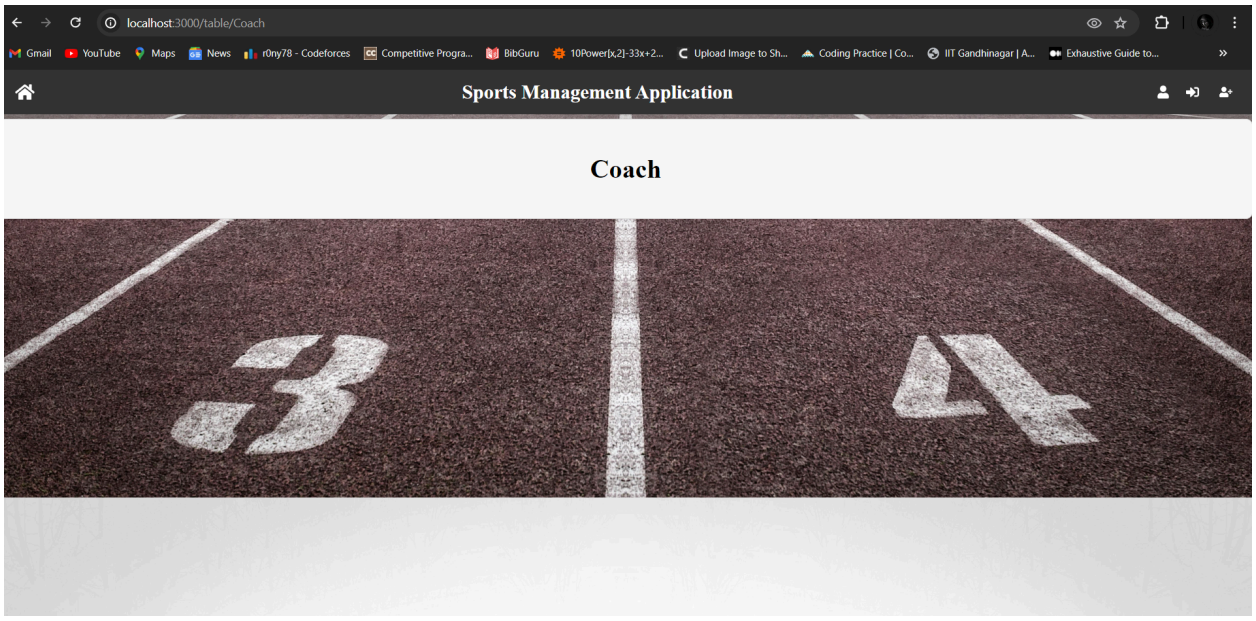
Preventing URL attack:

To prevent this, we generate the token while logging in. So whenever we log in, it will generate a token. So when we want to see the data or perform any operation, it will check whether the token is generated and compare it. If the token is there, then it will show us the data.

```
mysql.connection.commit()
cur.close()
access_token = create_acc
return jsonify({'token': access_token,'success': True}), 200
```

(variable) access_token: str

After this, the data should not be shown directly.



Here, we can see that it does not show any data if we try to access the data using a direct URL.

3. CRSF(Cross-site request forgery) Attack:

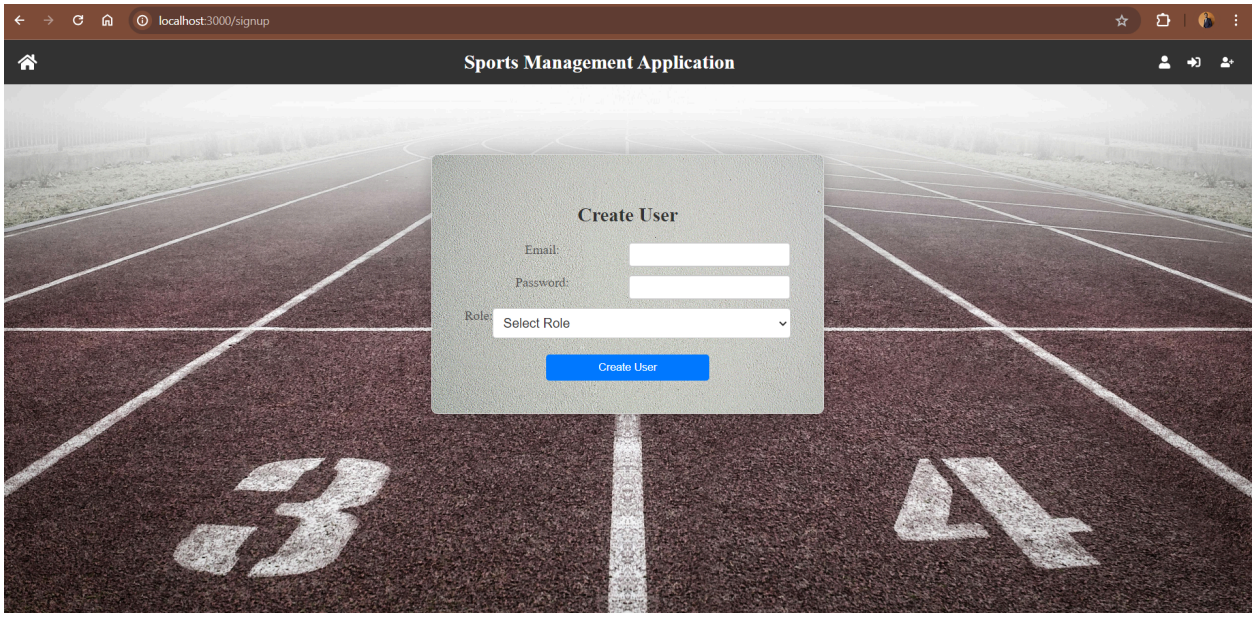
CSRF is a web security vulnerability that allows attackers to induce users to perform actions they do not intend to perform. Attackers create a malicious webpage or inject the code into this page. This code contains the request to target the website. After that, the attackers trick the user into visiting this webpage. Attackers can use many techniques, like phishing emails or malicious links. The malicious code in the webpage sends a request to the target website. This request includes the user's authentication token or session cookie.

Preventing CRSF:

To prevent this attack, we use CSRF tokens. This is generated for use in HTTP requests. This token expires after some time. This is similar to our token, which we generate while logging in. Our token also expires after some time.

```
#cur.close()
if user_data:
    cur = mysql.connection.cursor()
    cur.execute("SELECT table_name FROM information_schema.tables WHERE table_schema = 'sports_management'
    mysql.connection.commit()
    cur.close()
    access_token = create_access_token(identity=useremail)
    return jsonify({'token': access_token,'success': True}), 200
else:
    return jsonify({'success': False, 'error': 'Invalid credentials'})
```

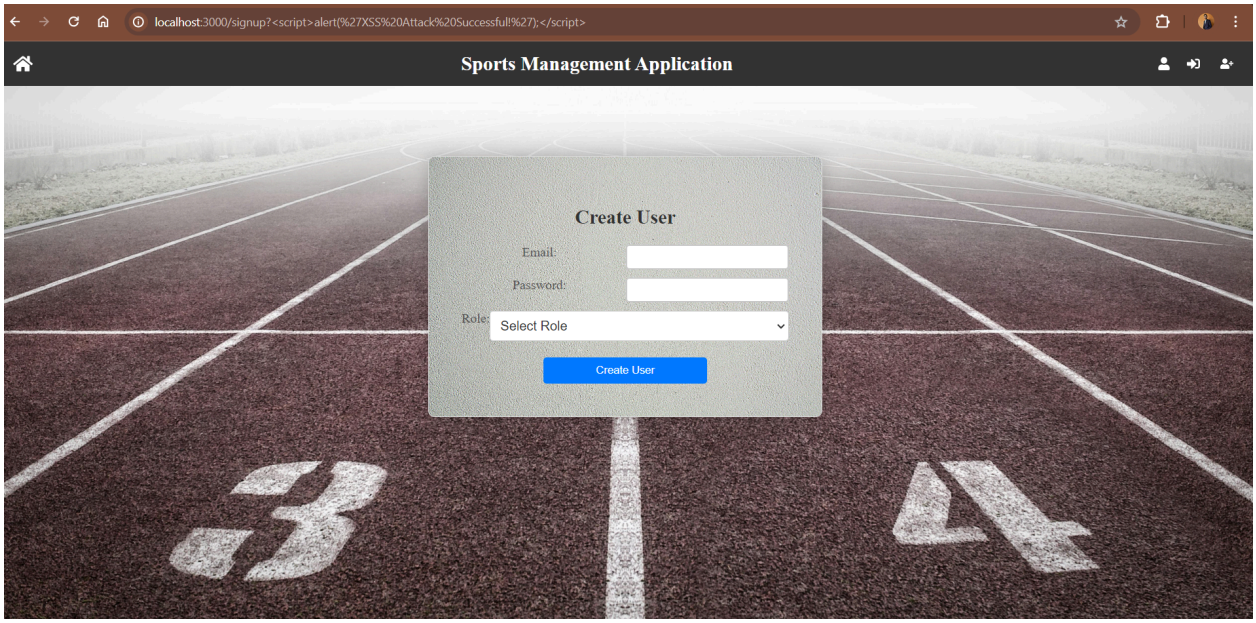
4. XSS Attack:



This is the normal signup page of our website.

Now, when I put the script in the URL (?<script>alert('XSS Attack Successful!');</script>), it should not

load the signup page.



But this is loading the signup page. This script in the URL could have dangerous files that could affect our database or cookies. We have to do something so that if we put this kind of script in the URL, this should not affect it, or the database should show in the URL that this is invalid.

Preventing XSS:

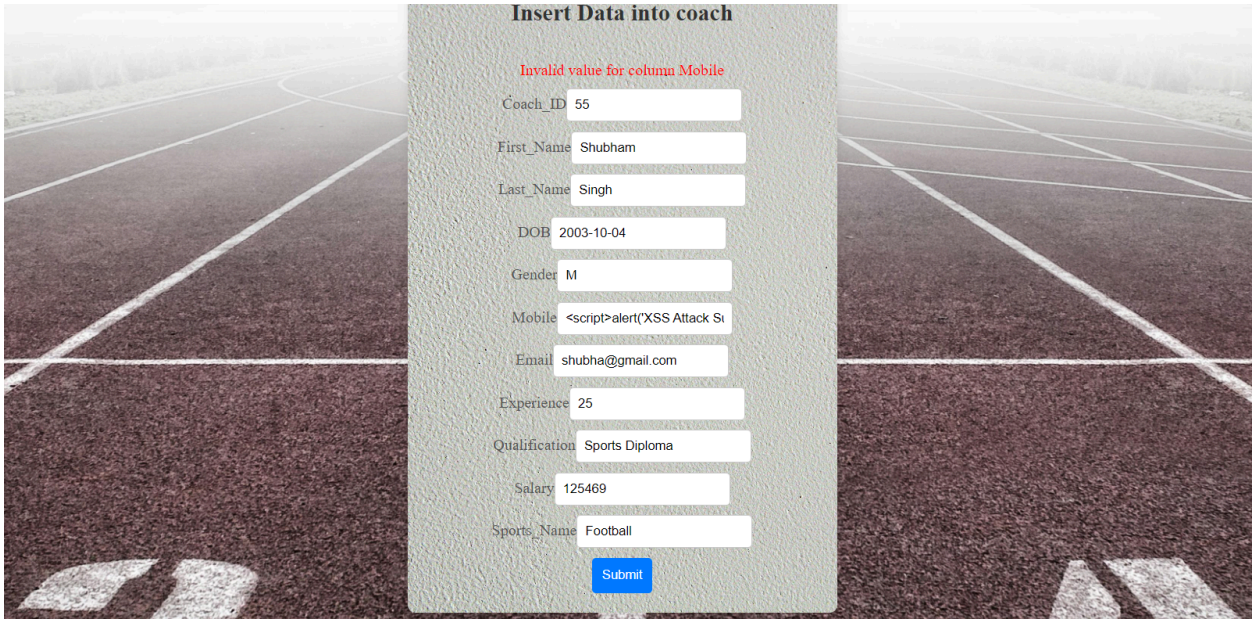
```
useEffect(() => {
  const isValidURL = validateURL(window.location.href);
  if (!isValidURL) {
    window.location.href = '/invalid-url';
  }
}, []);

const validateURL = (url) => {
  return url === 'http://localhost:3000/signup';
};
```

In this code, we check the URL. If the URL is invalid, it should navigate this kind of page/invalid-URL, which has no connection with our database.

As we can see, if the user puts some script file in the URL, the signup page will not load.

To Secure the web application, we implemented the regular expression for each table column for each input entry. While taking the input from the form, we compared the input field format with the regular expression. Because of this, no one can push the script through our input fields.

```
const columnRegex = {
  Coach_ID: /^\d+$/, // int
  First_Name: /^[a-zA-Z\s]+$/,
  Last_Name: /^[a-zA-Z\s]+$/,
  DOB: /^\d{4}-\d{2}-\d{2}$/, // date
  Gender: /^[MF]$/,
  Mobile: /^\d{10}$/,
  Email: /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/,
  Experience: /^\d+$/, // int
  Qualification: /.*/,
  Salary: /^\d+(\.\d{1,2})?$/, // decimal
  Sports_Name: /^[a-zA-Z\s]+$/,
  Player_ID: /^\d+$/, // int
  Equipment_Name: /.*/,
  Date_Time: /^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}$/, // datetime format
  Quantity: /^\d+$/,
  Return_Status: /.*/,
  Equipment_photo: /.*/,
  Team_ID: /^\d+$/, // int
  Team_Name: /.*/,
  Captain_ID: /^\d+$/, // int
  Tournament_ID: /^\d+$/, // int
  Tournament_Name: /.*/,
  Start_Date: /^\d{4}-\d{2}-\d{2}$/, // date
  End_Date: /^\d{4}-\d{2}-\d{2}$/, // date
  Venue: /.*/,
  Winner_Team_ID: /^\d+$/, // int
  Match_ID: /^\d+$/, // int
```

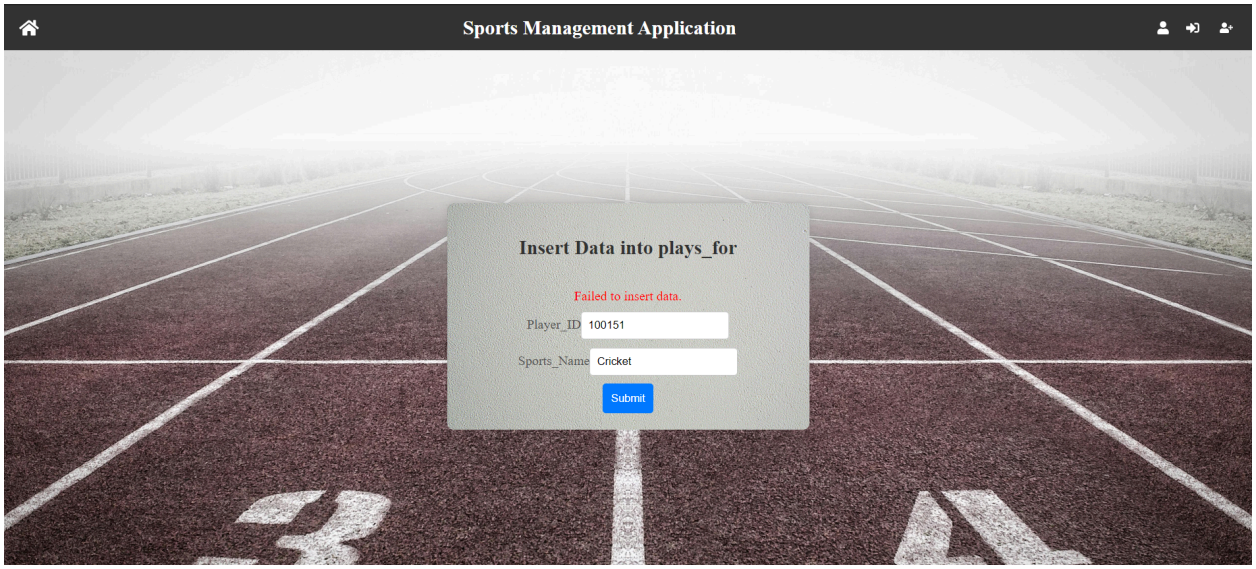


As we are getting the error “Invalid value for the column mobile”.

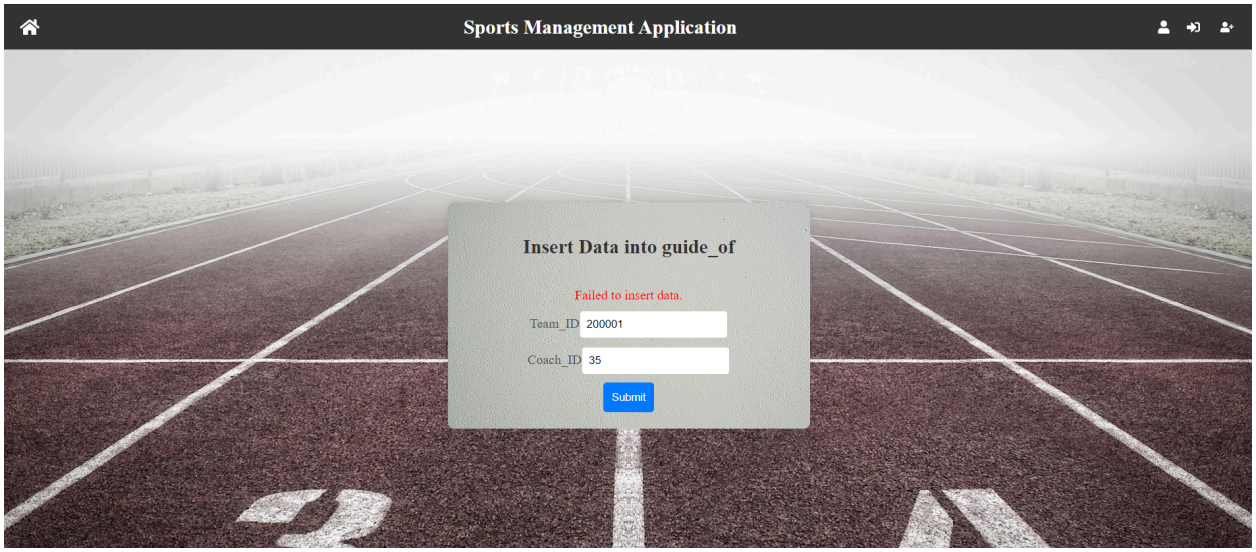
3.3.2:

No changes have been made to the relation or its constraints. Below are screenshots of all the constraints:

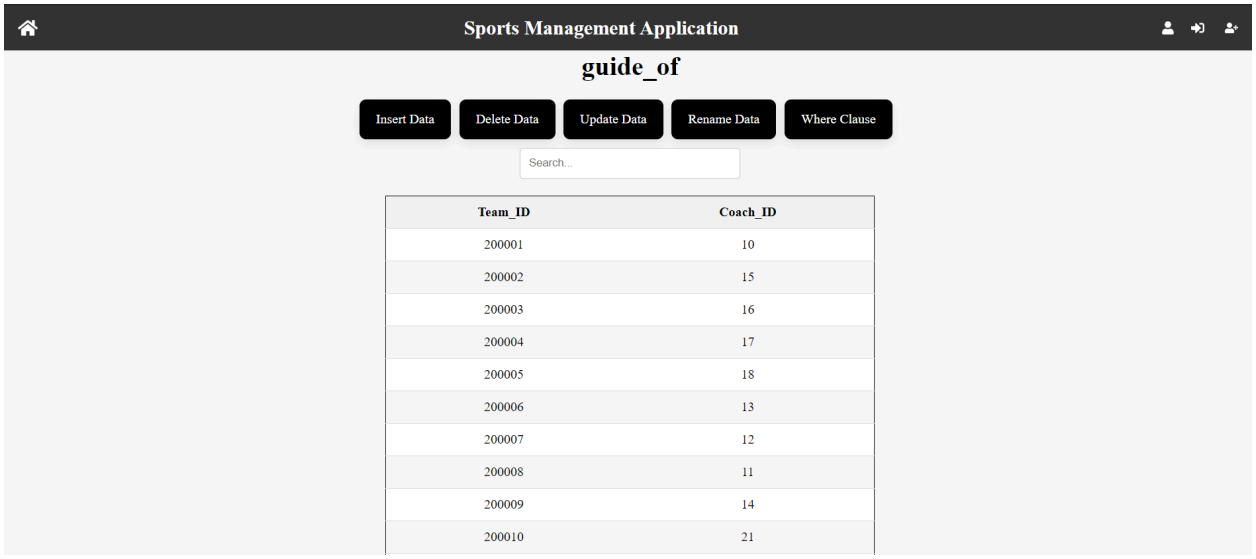
1) In the Plays_For table, the Player ID is the primary key, so if I insert another entry with an existing Player ID, it should give an error

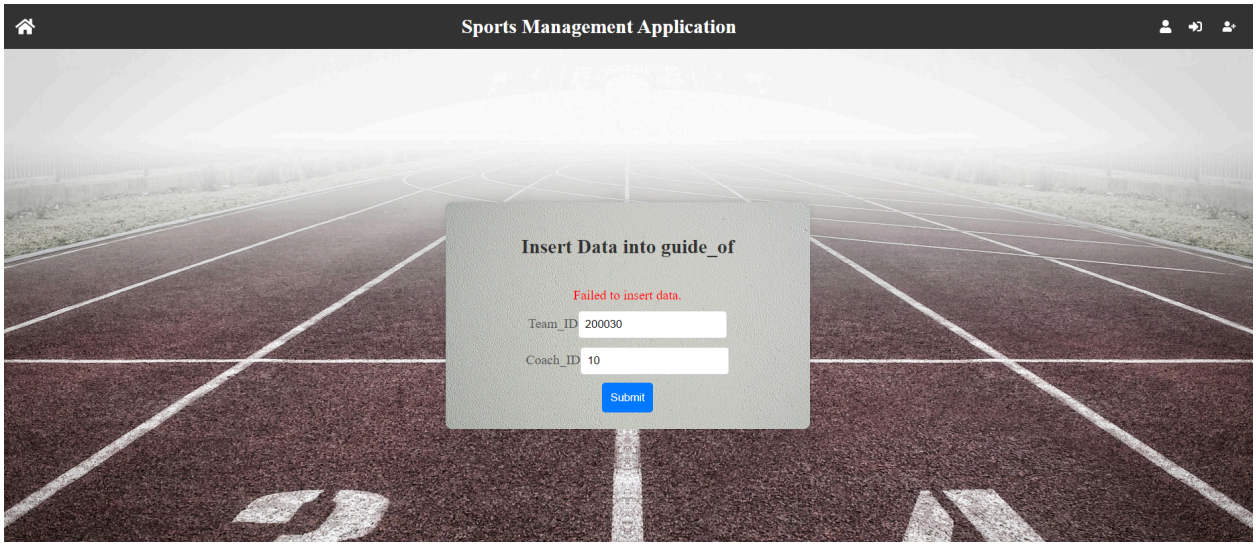


2) In the guide_of table, the Team_ID and Coach_ID are the primary keys, so if I insert another entry with an existing Team_ID, it should give an error

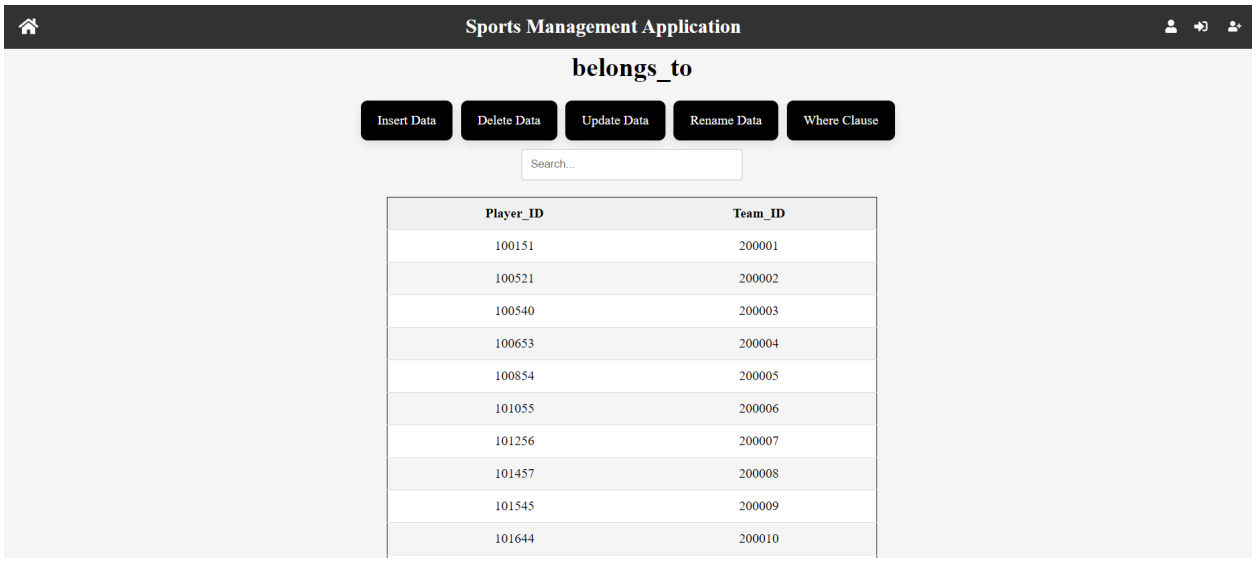


3) the relation guide_of is a one-to-one relation, so we can see that there is only one coach for one table. I try to insert with the same Coach_ID and new Team_ID, and it should give the error



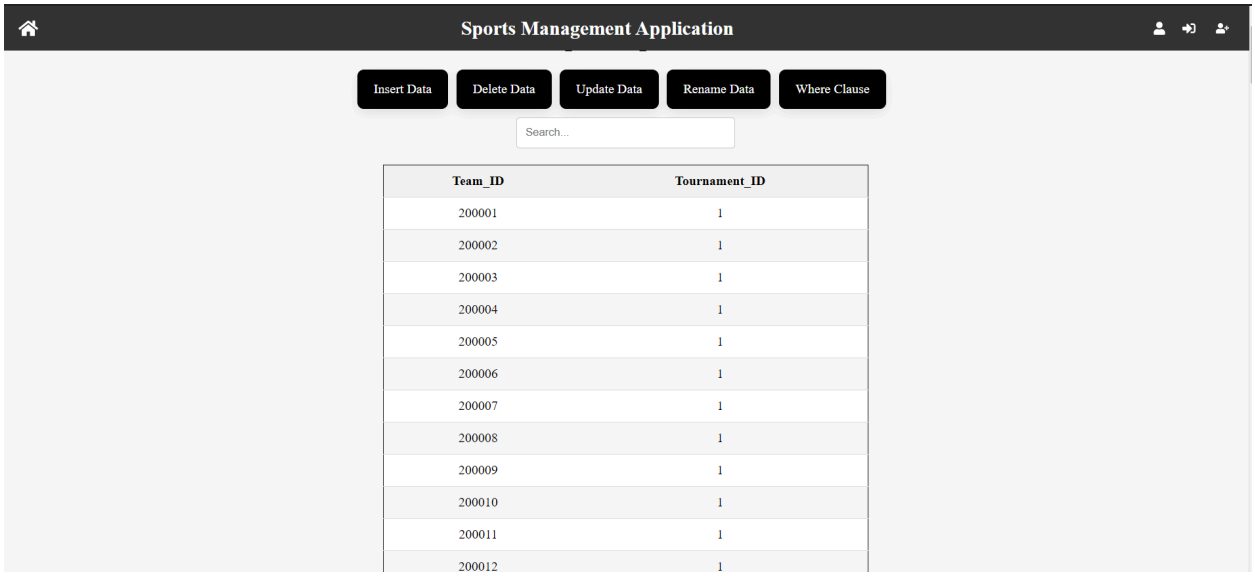


4) the relation belongs_to is a one-to-many relation, so we can see that one player is in multiple teams.

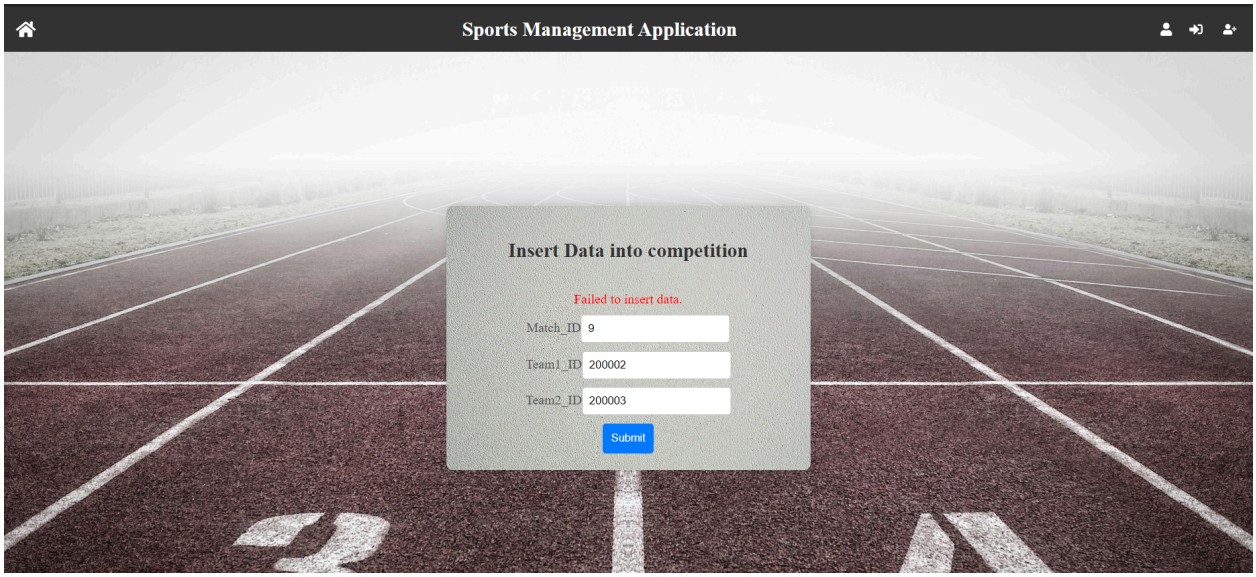


Player_ID	Team_ID
100151	200001
100521	200002
100540	200003
100653	200004
100854	200005
101055	200006
101256	200007
101457	200008
101545	200009
101644	200010

5) Participate is a one-to-many relationship, so we can see that one team can participate in multiple tournaments.



6) In table Matches, the primary key is the Match_ID, referenced in relation to Competition as a foreign key. So, I should get an error if I add an entry with a Match_ID that is not present in the Matches table.



Contributions of group

G1

Name	Roll NO.	Contribution
Saurabh Kumar	21110187	I have contributed in taking feedback from stake-holder and writing report.
Prashant Meena	21110163	I have contributed to the attacks [SQL Injection and XSS].
Rahul Madhukar	21110175	I have contributed to the implementation the Multi-user concurrency.

G2

Name	**Roll no.**	**Contribution**
Shubham Singh	21110207	I have contributed to the attacks [SQL Injection and XSS]. and creating different views.
Karna Pardeev Sai	21110097	I have contributed to taking feedback from stakeholders and writing reports.
Netram Choudhary	21110138	I have been contributing to the update on the login signup. Incorporating changes in stake-holder feedback, such as validation and checks. Implemented Notification system.