TENNIS TOURNAMENT MANAGEMENT

SUMMARY REPORT

GROUP (5)

GROUP MEMBERS

Student Name	Student ID	Roll No
Saurav Kumar	02049606	36
Shubham Singh	02131135	76
Shruti Balaji	02109411	03

PART 1 Database-driven Application

DESCRIPTION

Tennis competitions are complicated activities that need careful preparation and administration. The procedure can be streamlined with a well-designed database, which will facilitate the management of players, matches, and results. To aid effective tournament management, the goal of this project is to design an extensive database for a tennis tournament management system.

Tennis tournament information management database schema is a detail schema containing the details of a tennis tournaments, matches, player, venue, court, result, fixtures. It is a comprehensive database containing about 7 major entities. These entities are connected to one another through various relationships by connecting various attributes of all tables.

DATABASE-DRIVEN APPLICATION USE-CASES:

Entity descriptions:

1. **Match -** Acts like a fact table containing IDs connecting four other tables like tournament. ID, result ID, player ID, fixture ID. Also gives the type of pitch the match is to be played in.
2. **Player -** Contains the player details like their name, age, nationality, date of birth, gender, world ranking, id.

3. **Tournament** – this table is used to store information about the number of tournaments, held where, starting, and ending date and the type of surface played on
4. **Result** – the result of every match is stored here with details of player id, winner or loser, final score, sets played, match duration.
5. **Fixture** – fixture name and id are stored in this table.
6. **Venue** – this table is used to store the venue of each tournament and matches including details like country, continent, location of the venue and its ID.
7. **Court** – the court represents the details of the specific court in a particular venue where the match is to be played which includes the court ID, court name and venue ID.

TECHNOLOGIES:

Programming Language	JavaScript & NoSQL
Database Server	MongoDB
Web Framework	MERN
IDE	VsCode

IMPLEMENTATIONS:

1) **Establish Database Connection**: The first step is to establish a connection to the MongoDB database. This can be done using the mongoose module in Node.js. MongoDB is a NoSQL database that allows for flexible schema design.

2) **Define Schema and Establish Relationships**: Using MongoDB, you define the database schema, detailing the different collections and their relationships. These collections correspond to entities like tournaments, players, matches, results, fixtures, venues, and courts. Establishing these relationships is crucial for smooth data management.

3) **CRUD Operations:** MongoDB provides built-in support for CRUD operations: create, read, update, and delete. By using Express.js, you can set up RESTful endpoints that allow you to perform these operations on your database. This framework serves as the backend of your tennis tournament management application.

4) **Data Population:** Once the schema is defined, you can populate the database with initial data. This is crucial for testing the application and ensuring all relationships between entities work as expected.

5) **Frontend Implementation:** The frontend is developed using React.js, part of the MERN stack. This component interacts with the backend to display data and handle user interactions. CRUD operations from the frontend are performed by calling the appropriate RESTful endpoints defined in the Express.js server.

6) **Develop User Interface:** Using React.js, you create a user-friendly interface that allows users to interact with the database. This involves creating components for adding, updating, reading, and deleting tournament-related data. React's component-based structure makes it easy to build and maintain the frontend.

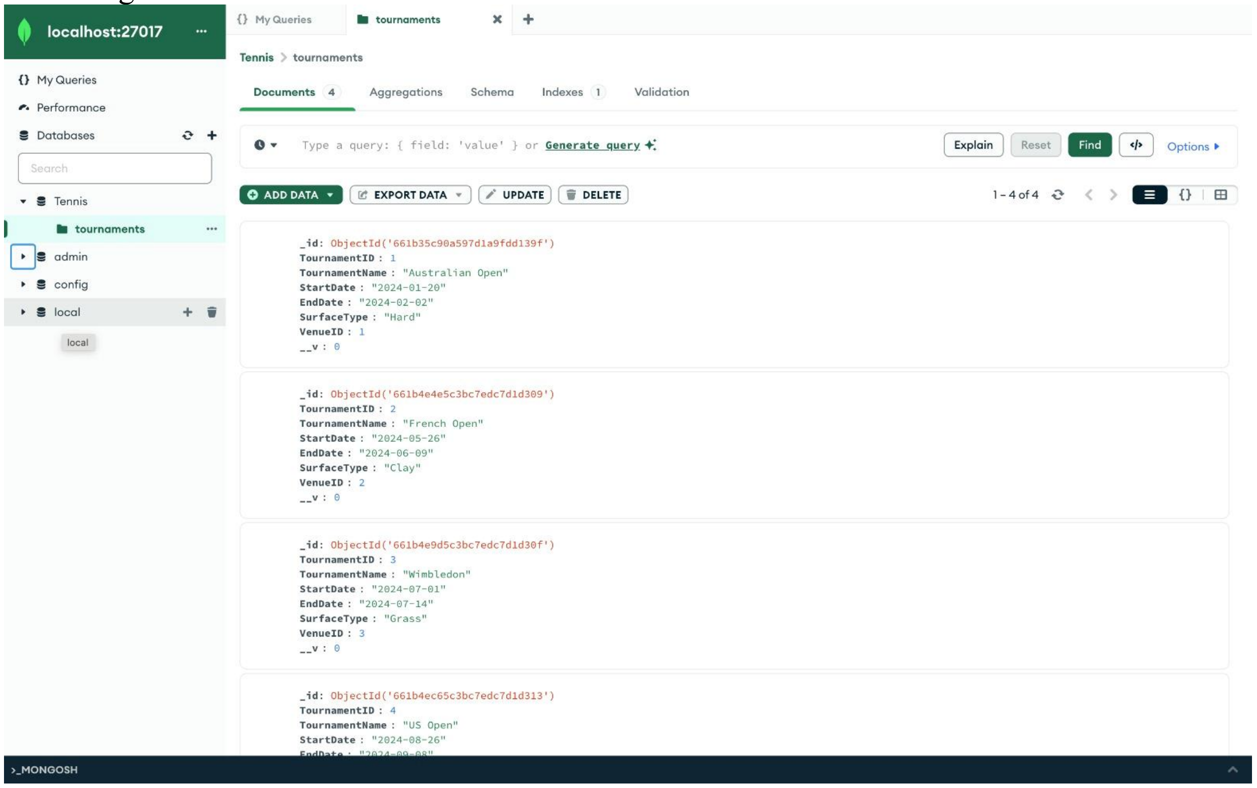
7) **End-to-End Integration:** With the backend and frontend components in place, the next step is to integrate them for seamless operation. The React frontend sends requests to the Express.js backend, which in turn communicates with MongoDB. This full integration allows for real-time data updates and CRUD operations.

8) **Deployment and Testing:** After the application is developed, it's crucial to deploy and test it to ensure it meets all requirements. Testing involves checking CRUD operations, user interface responsiveness, and data integrity. The application can be deployed locally for initial testing and later moved to a cloud environment for wider accessibility.

PART 2 Working Incorporation of Data Storage

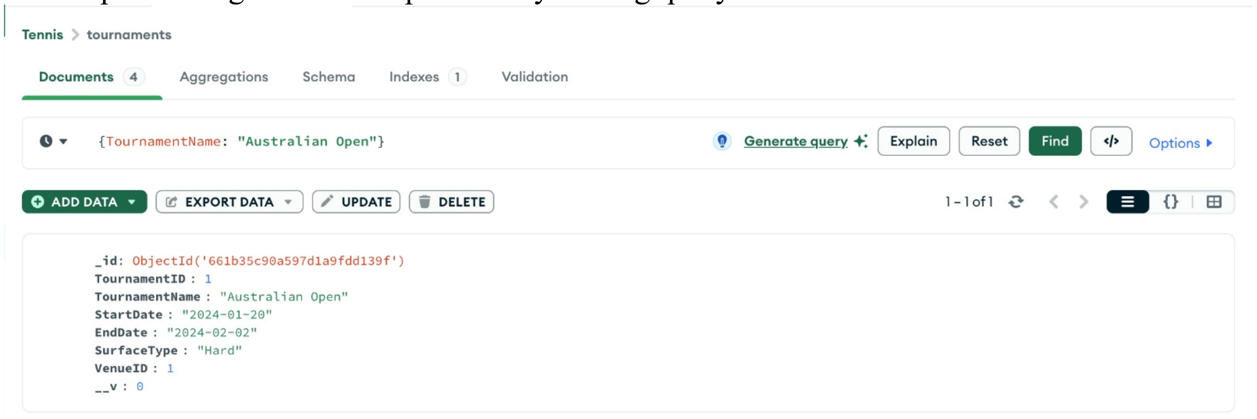
We have used Mongo database as our backend to store the data received on our tennis tournament webpage. Our database is capable of performing the CRUD operation with the basic configuration of validating the data with the proper data type.

1. Our mongo database is hosted on localhost 27017.



```
_id: ObjectId('661b35c90a597d1a9fdd139f')
TournamentID : 1
TournamentName : "Australian Open"
StartDate : "2024-01-20"
EndDate : "2024-02-02"
SurfaceType : "Hard"
VenueID : 1
__v : 0

_id: ObjectId('661b4e4e5c3bc7edc7d1d309')
TournamentID : 2
TournamentName : "French Open"
StartDate : "2024-05-26"
EndDate : "2024-06-09"
SurfaceType : "Clay"
VenueID : 2
__v : 0

_id: ObjectId('661b4e9d5c3bc7edc7d1d30f')
TournamentID : 3
TournamentName : "Wimbledon"
StartDate : "2024-07-01"
EndDate : "2024-07-14"
SurfaceType : "Grass"
VenueID : 3
__v : 0

_id: ObjectId('661b4ec65c3bc7edc7d1d313')
TournamentID : 4
TournamentName : "US Open"
StartDate : "2024-08-26"
EndDate : "2024-09-08"
```

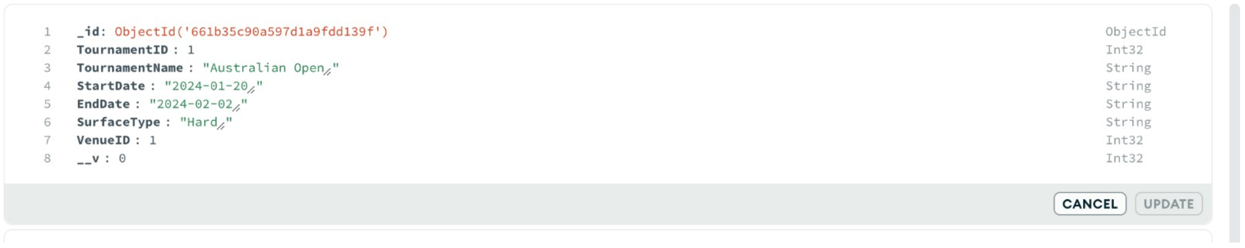
2. We are performing the search operation by writing query.



```
{TournamentName: "Australian Open"}
```

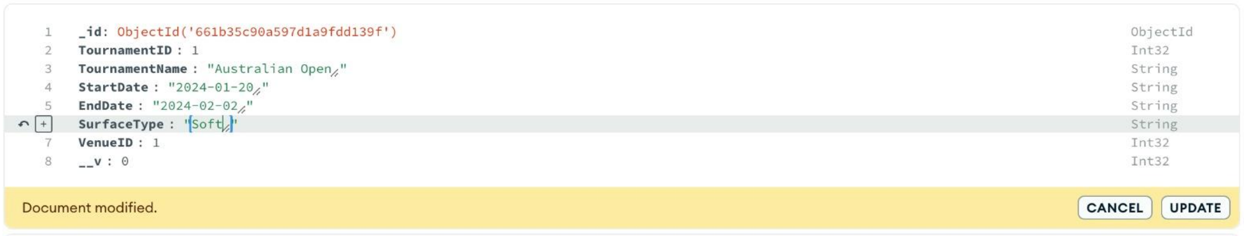
```
_id: ObjectId('661b35c90a597d1a9fdd139f')
TournamentID : 1
TournamentName : "Australian Open"
StartDate : "2024-01-20"
EndDate : "2024-02-02"
SurfaceType : "Hard"
VenueID : 1
__v : 0
```

3. Now we will perform the update operation in which we will change the surface type from "HARD" to "SOFT".



```
_id: ObjectId('661b35c90a597d1a9fdd139f')              ObjectId
TournamentID : 1                                       Int32
TournamentName : "Australian Open"                     String
StartDate : "2024-01-20"                               String
EndDate : "2024-02-02"                                 String
SurfaceType : "Hard"                                   String
VenueID : 1                                            Int32
__v : 0                                                Int32
```

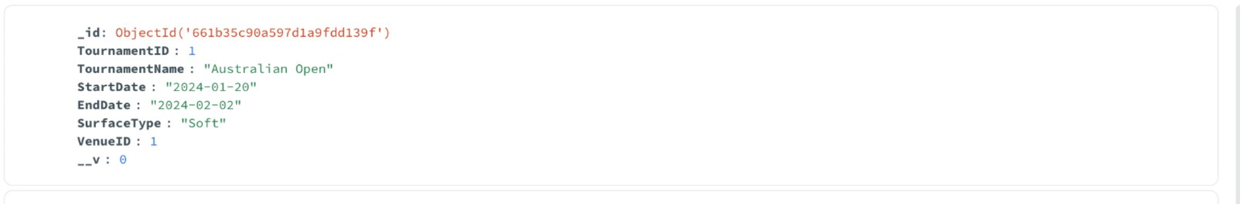
After changing it to soft we can click on update button



```
_id: ObjectId('661b35c90a597d1a9fdd139f')              ObjectId
TournamentID : 1                                       Int32
TournamentName : "Australian Open"                     String
StartDate : "2024-01-20"                               String
EndDate : "2024-02-02"                                 String
SurfaceType : "Soft"                                   String
VenueID : 1                                            Int32
__v : 0                                                Int32
```

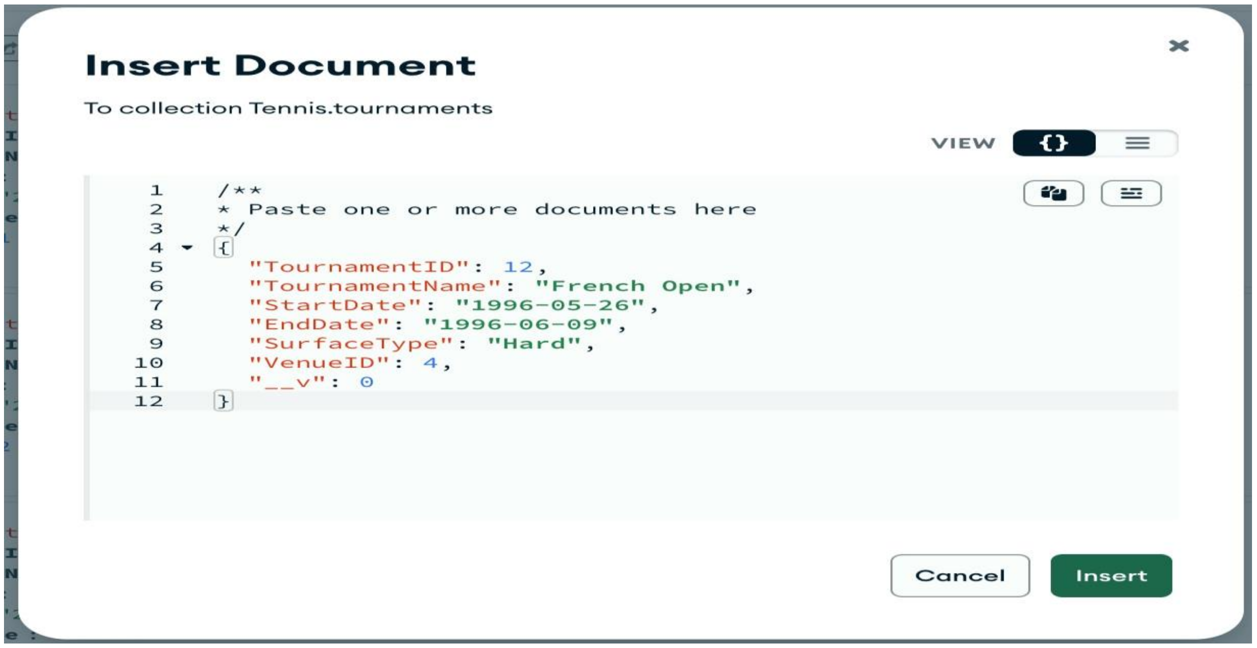
Document modified.

Finally, we can see that the data has been updated.



```
_id: ObjectId('661b35c90a597d1a9fdd139f')
TournamentID : 1
TournamentName : "Australian Open"
StartDate : "2024-01-20"
EndDate : "2024-02-02"
SurfaceType : "Soft"
VenueID : 1
__v : 0
```

4. We can insert the data using the insert button and then we can select the appropriate option to insert the data.

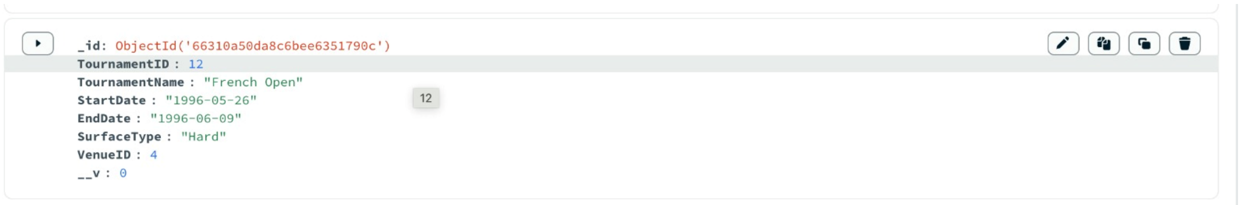


Insert Document

To collection Tennis.tournaments

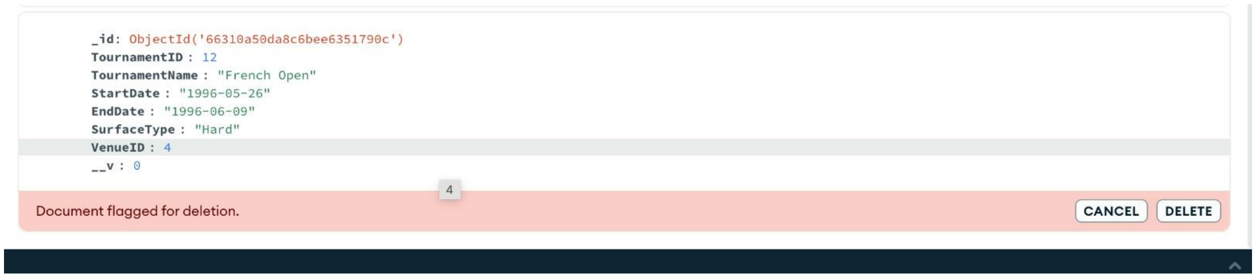
```
/**
* Paste one or more documents here
*/
{
  "TournamentID": 12,
  "TournamentName": "French Open",
  "StartDate": "1996-05-26",
  "EndDate": "1996-06-09",
  "SurfaceType": "Hard",
  "VenueID": 4,
  "__v": 0
}
```

After clicking on the insert button we can see that data has been inserted in our database.



```
_id: ObjectId('66310a50da8c6bee6351790c')
TournamentID : 12
TournamentName : "French Open"
StartDate : "1996-05-26"
EndDate : "1996-06-09"
SurfaceType : "Hard"
VenueID : 4
__v : 0
```

5. To perform the delete operation we can click on delete button next to the record and then we can confirm our deletion.

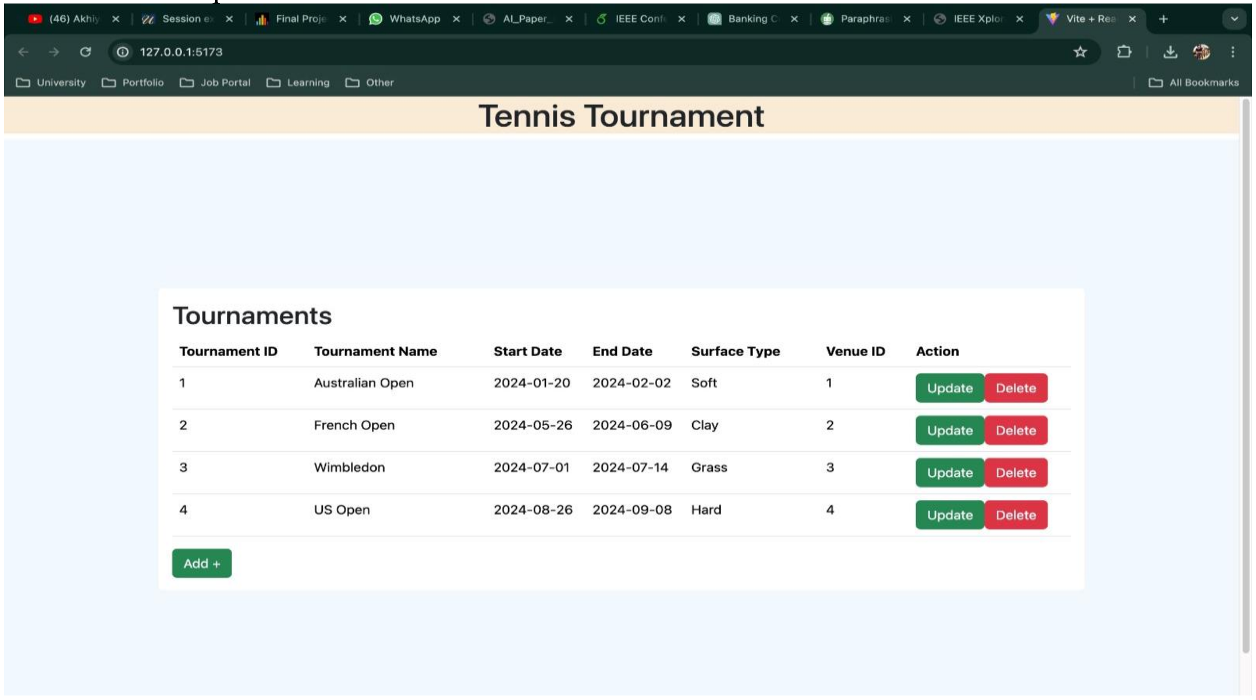


```
_id: ObjectId('66310a50da8c6bee6351790c')
TournamentID : 12
TournamentName : "French Open"
StartDate : "1996-05-26"
EndDate : "1996-06-09"
SurfaceType : "Hard"
VenueID : 4
__v : 0
```

Document flagged for deletion.

PART 3 FRONT END INTERFACE

We developed our project using the MERN stack framework where M stands for MongoDB, E stands for Express JS, R stands for React JS, and N stands for Node JS. In our front end we take care of CRUD operation. Below is our front end looks like.



127.0.0.1:5173

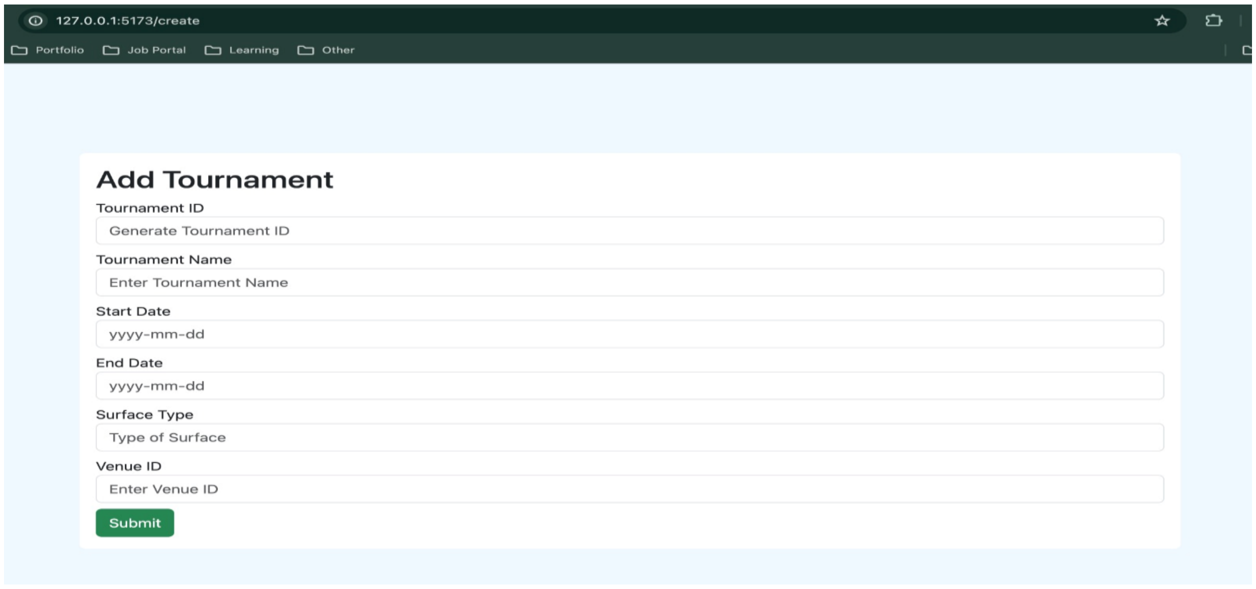
Tennis Tournament

Tournaments

Tournament ID	Tournament Name	Start Date	End Date	Surface Type	Venue ID	Action
1	Australian Open	2024-01-20	2024-02-02	Soft	1	Update Delete
2	French Open	2024-05-26	2024-06-09	Clay	2	Update Delete
3	Wimbledon	2024-07-01	2024-07-14	Grass	3	Update Delete
4	US Open	2024-08-26	2024-09-08	Hard	4	Update Delete

Add +

1. To add the data, we need to click on add button. After clicking on add we will be prompted by a screen to add the details and that will add the data in our database and reflect on our screen.



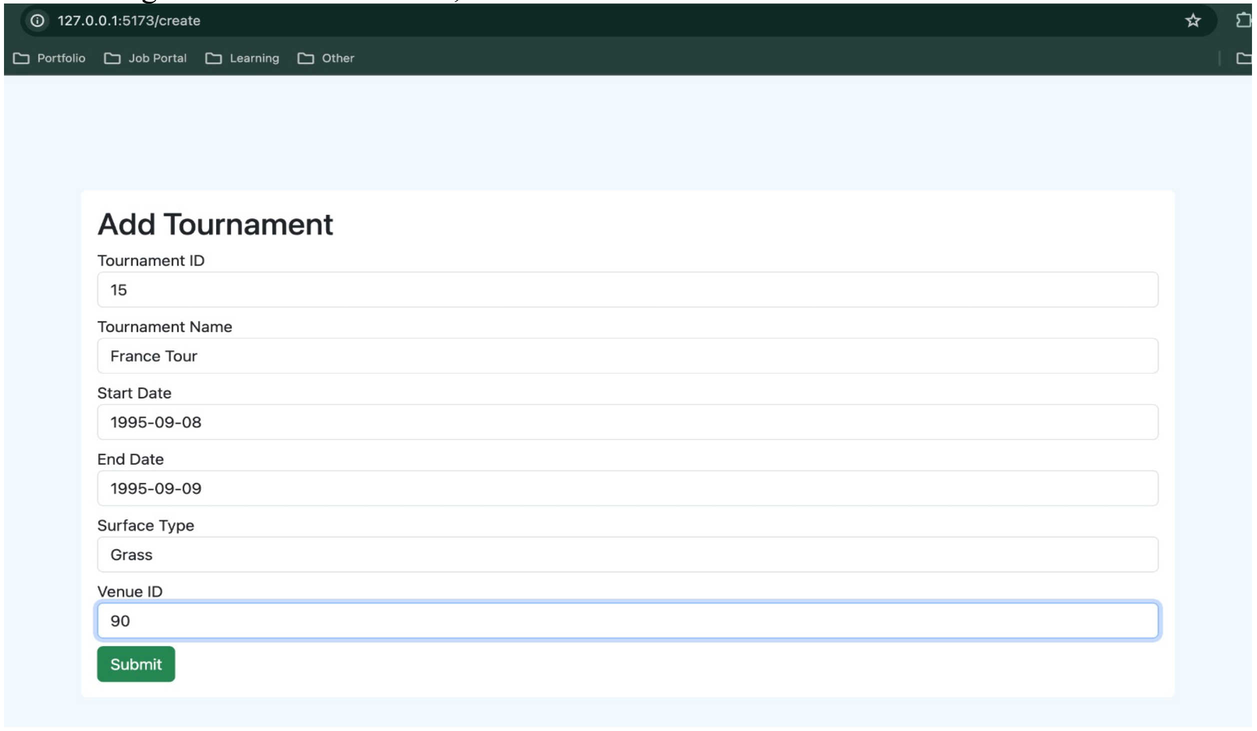
127.0.0.1:5173/create

Add Tournament

Tournament ID	Generate Tournament ID
Tournament Name	Enter Tournament Name
Start Date	yyyy-mm-dd
End Date	yyyy-mm-dd
Surface Type	Type of Surface
Venue ID	Enter Venue ID

Submit

After filling the data in above form, we can click on submit.



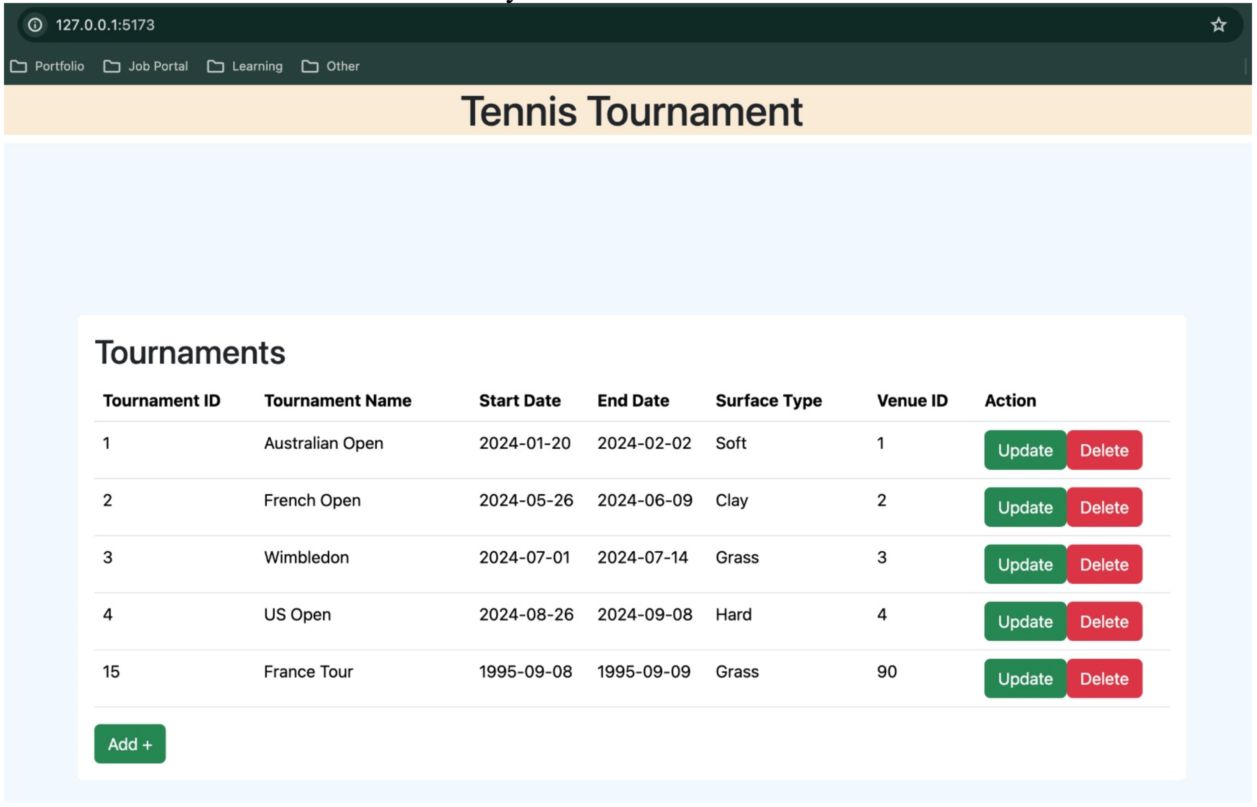
127.0.0.1:5173/create

Portfolio | Job Portal | Learning | Other

Add Tournament

Tournament ID	15
Tournament Name	France Tour
Start Date	1995-09-08
End Date	1995-09-09
Surface Type	Grass
Venue ID	90

This will trigger the submit form in backend and the data will be stored in our backend. After that read file will reflect the newly entered data in our frontend.



127.0.0.1:5173

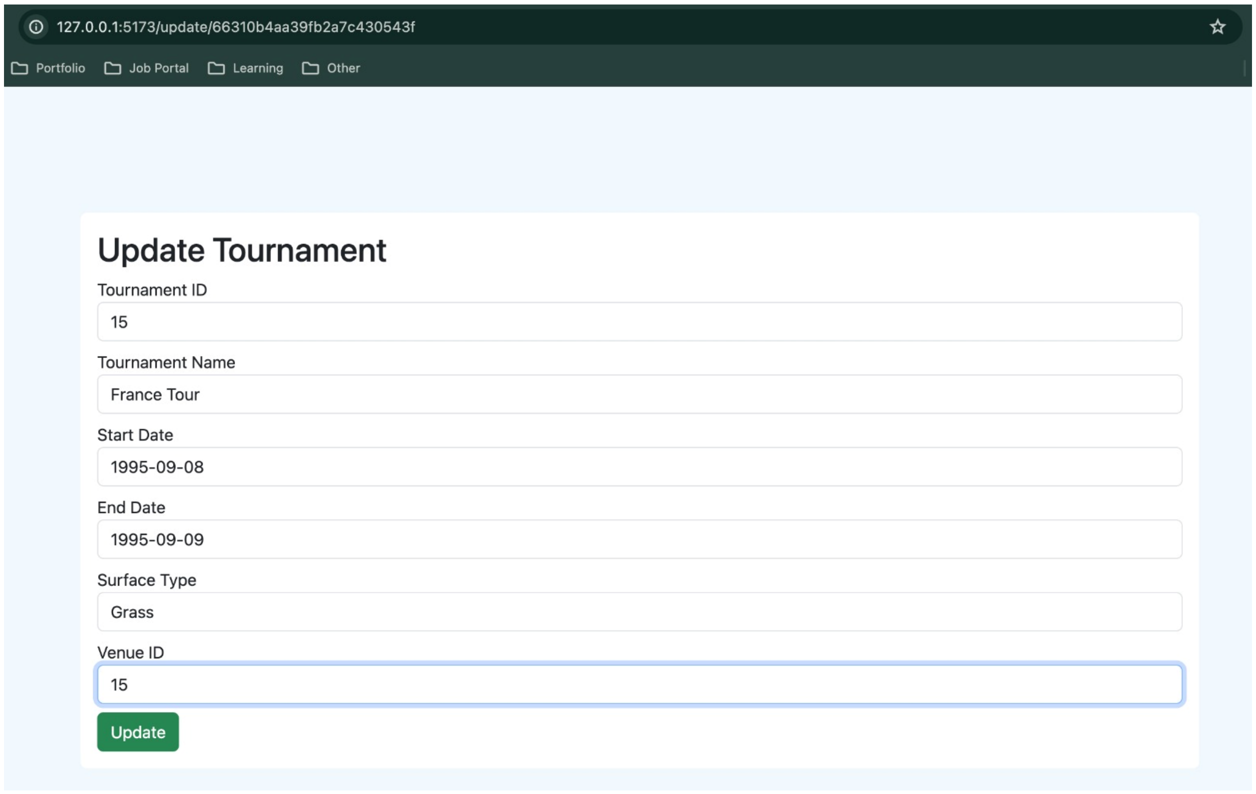
Portfolio | Job Portal | Learning | Other

Tennis Tournament

Tournaments

Tournament ID	Tournament Name	Start Date	End Date	Surface Type	Venue ID	Action
1	Australian Open	2024-01-20	2024-02-02	Soft	1	Update Delete
2	French Open	2024-05-26	2024-06-09	Clay	2	Update Delete
3	Wimbledon	2024-07-01	2024-07-14	Grass	3	Update Delete
4	US Open	2024-08-26	2024-09-08	Hard	4	Update Delete
15	France Tour	1995-09-08	1995-09-09	Grass	90	Update Delete

2. To update the data we can directly click on the update button on the right side of the data. This will open the data and then we can modify the data as per our requirement.

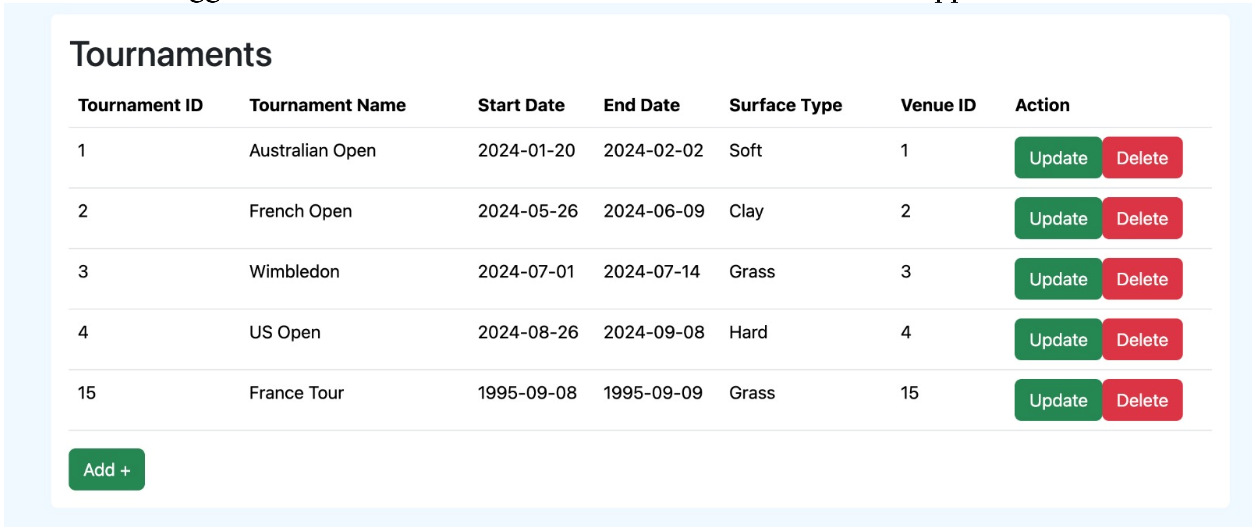


127.0.0.1:5173/update/66310b4aa39fb2a7c430543f

Update Tournament

Tournament ID	15
Tournament Name	France Tour
Start Date	1995-09-08
End Date	1995-09-09
Surface Type	Grass
Venue ID	15

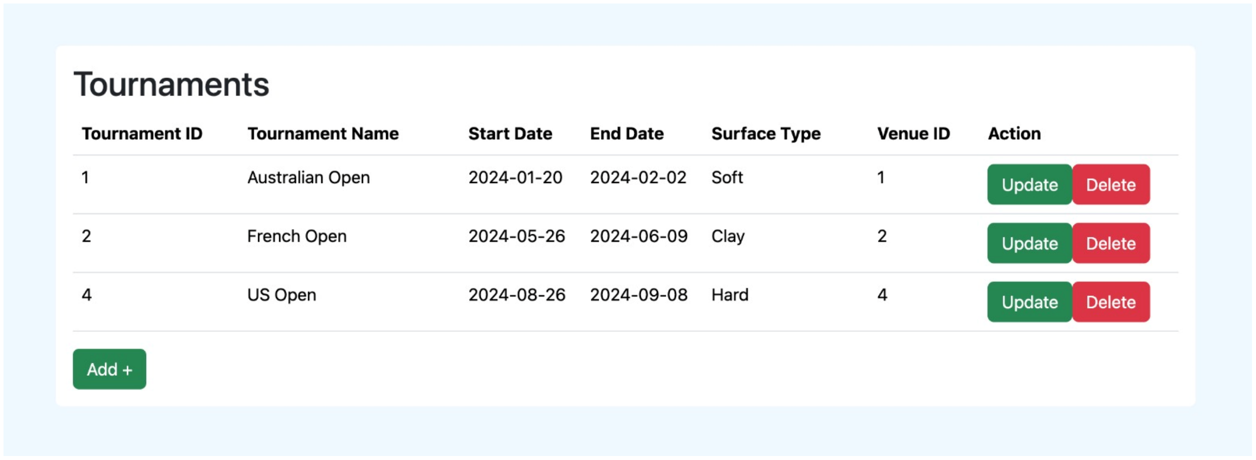
Here we have changed the venue ID from "90" to "15". And after clicking the Update button, update form will get triggered and in backend data will be updated. Further the read data will be triggered and the data will be shown in the front end of the application.



Tournaments

Tournament ID	Tournament Name	Start Date	End Date	Surface Type	Venue ID	Action
1	Australian Open	2024-01-20	2024-02-02	Soft	1	Update Delete
2	French Open	2024-05-26	2024-06-09	Clay	2	Update Delete
3	Wimbledon	2024-07-01	2024-07-14	Grass	3	Update Delete
4	US Open	2024-08-26	2024-09-08	Hard	4	Update Delete
15	France Tour	1995-09-08	1995-09-09	Grass	15	Update Delete

3. To delete the record, we can simply delete it by clicking on the delete button on the right side of the data. This delete will trigger the delete record form and data will be removed from the database in backend.
Here we have deleted two records, one is the created data and old existing data.

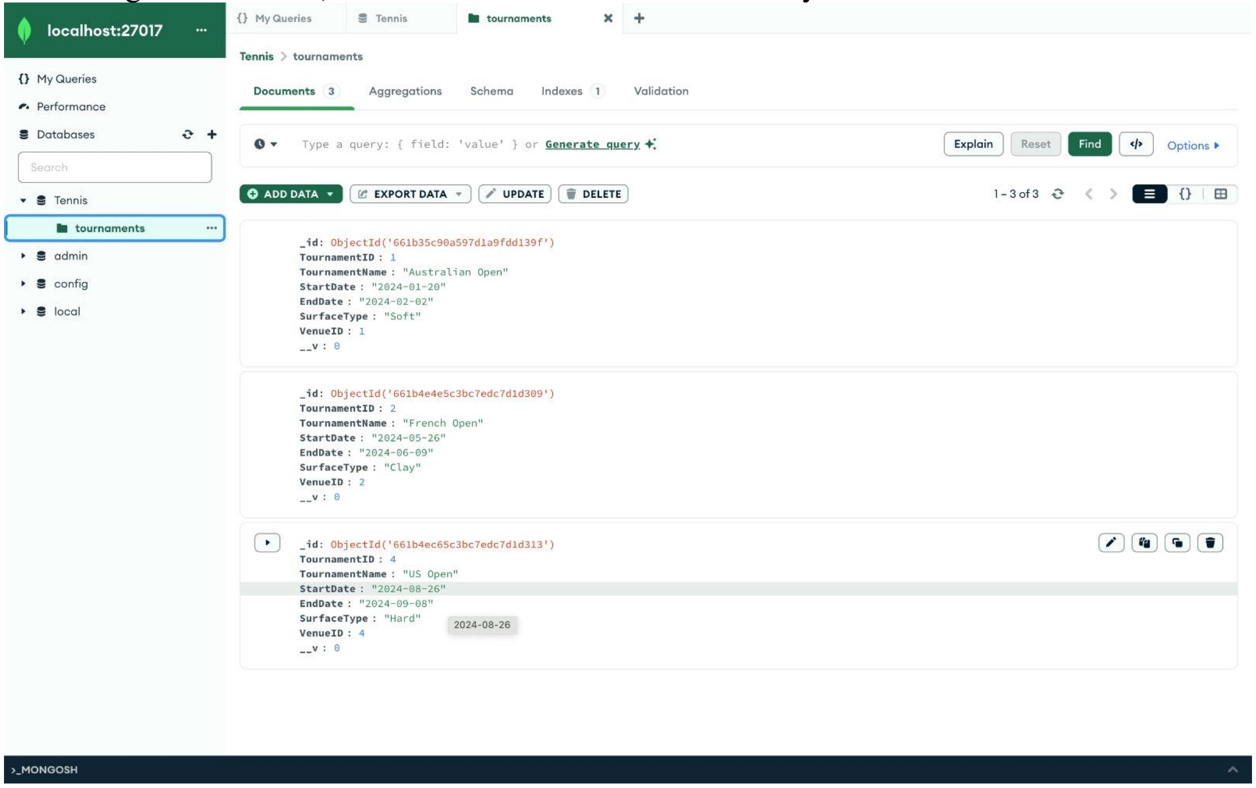


Tournaments

Tournament ID	Tournament Name	Start Date	End Date	Surface Type	Venue ID	Action
1	Australian Open	2024-01-20	2024-02-02	Soft	1	Update Delete
2	French Open	2024-05-26	2024-06-09	Clay	2	Update Delete
4	US Open	2024-08-26	2024-09-08	Hard	4	Update Delete

Add +

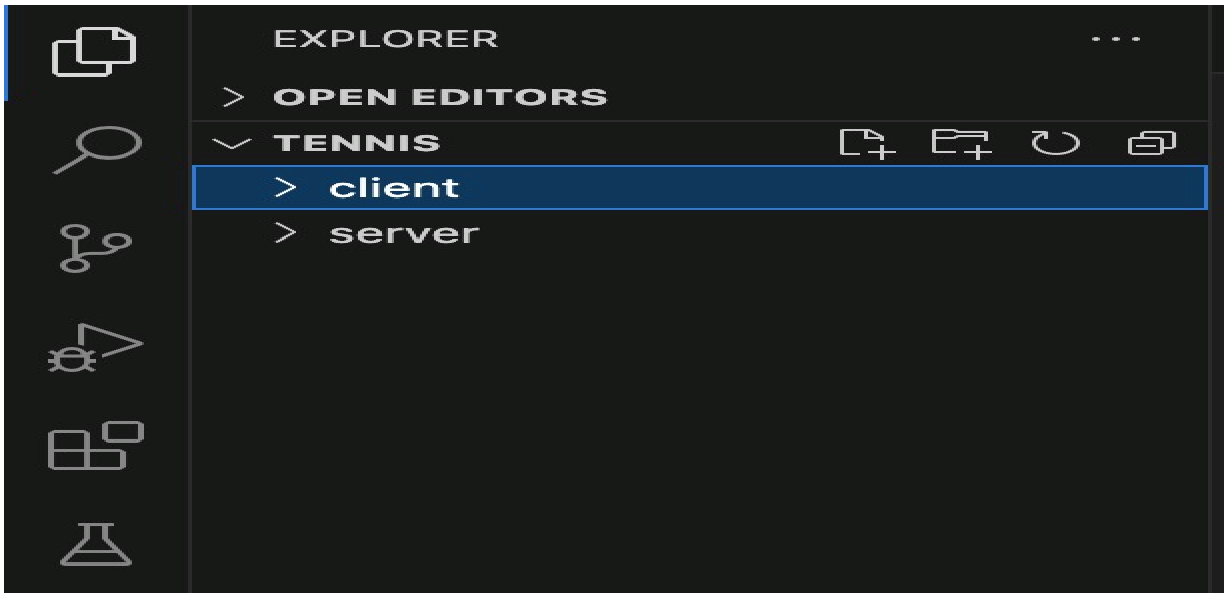
4. We can verify the final status by cross verifying the database and frontend. As our frontend is showing the three data, in same we can see that there are only three data in our backend.



```
_id: ObjectId('661b35c90a597d1a9fdd139f')
TournamentID : 1
TournamentName : "Australian Open"
StartDate : "2024-01-20"
EndDate : "2024-02-02"
SurfaceType : "Soft"
VenueID : 1
__v : 0

_id: ObjectId('661b4e4e5c3bc7edc7d1d309')
TournamentID : 2
TournamentName : "French Open"
StartDate : "2024-05-26"
EndDate : "2024-06-09"
SurfaceType : "Clay"
VenueID : 2
__v : 0

_id: ObjectId('661b4ec65c3bc7edc7d1d313')
TournamentID : 4
TournamentName : "US Open"
StartDate : "2024-08-26"
EndDate : "2024-09-08"
SurfaceType : "Hard"
VenueID : 4
__v : 0
```

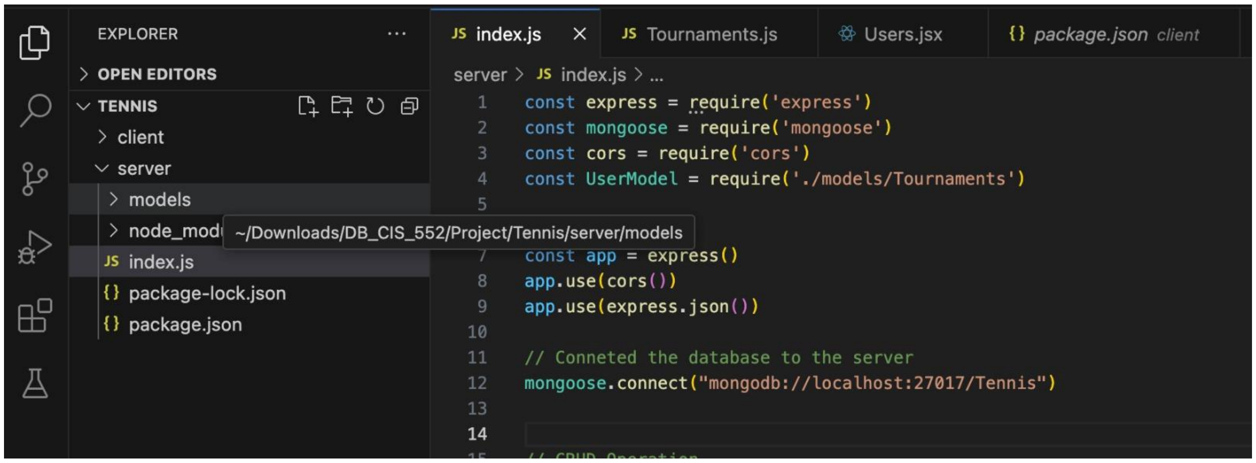
PART 4 Source Code

In our backend we have two files "client" and "server". Server takes care of the database and the client take care of the frontend. We have used VS Code as our IDE.



Exploring the SERVER

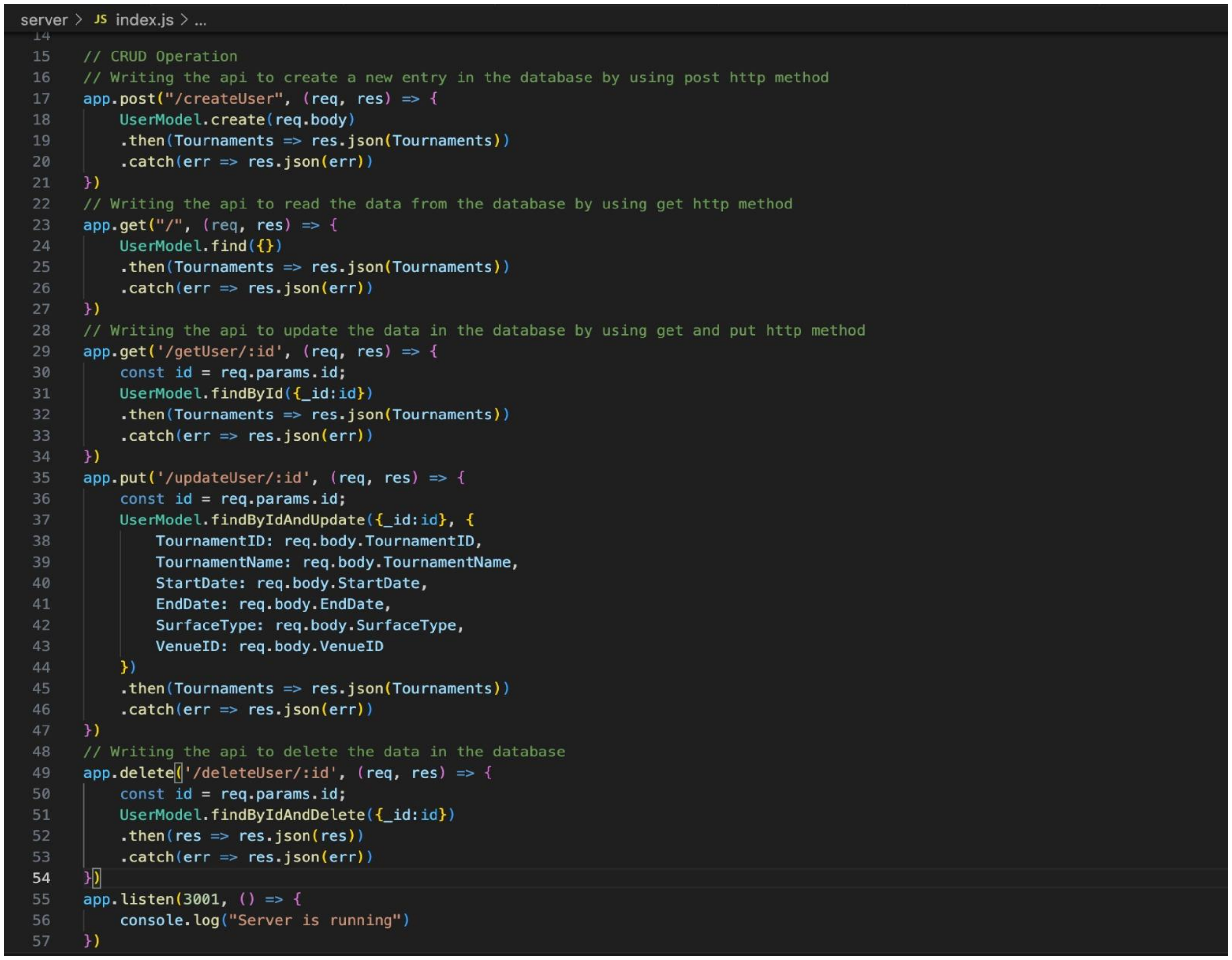
The server file has index.js file which contains the code of our backend connection. It contain express, mongoose and UserModel which are creating the connection at local host 27017 port.



```
const express = require('express')
const mongoose = require('mongoose')
const cors = require('cors')
const UserModel = require('./models/Tournaments')

const app = express()
app.use(cors())
app.use(express.json())

// Conneted the database to the server
mongoose.connect("mongodb://localhost:27017/Tennis")

```

Further in the same file we have written the four API's to perform the CRUD operation. These will be triggered when the respective end point is called. We have utilized 4 HTTP protocols which includes.

POST: Sends data to a server to create a new resource.
PUT: Updates or replaces an existing resource on a server.
GET: Retrieves data or resources from a server.
DELETE: Removes or deletes a resource from a server.

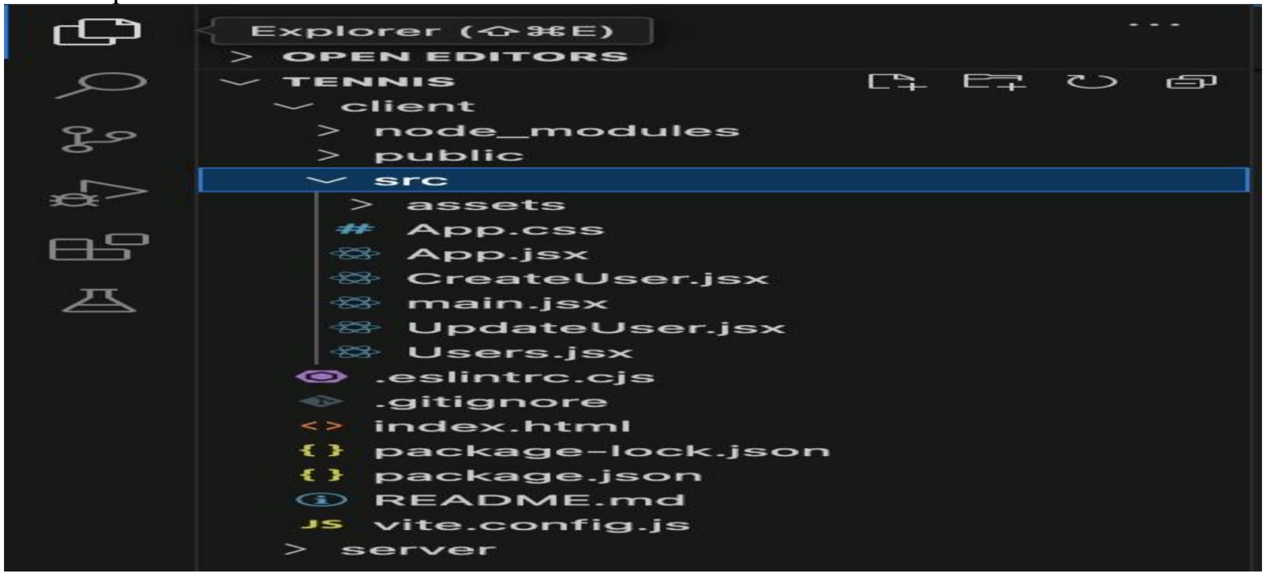


server > JS index.js > ...

```
// CRUD Operation
// Writing the api to create a new entry in the database by using post http method
app.post("/createUser", (req, res) => {
    UserModel.create(req.body)
    .then(Tournaments => res.json(Tournaments))
    .catch(err => res.json(err))
})
// Writing the api to read the data from the database by using get http method
app.get("/", (req, res) => {
    UserModel.find({})
    .then(Tournaments => res.json(Tournaments))
    .catch(err => res.json(err))
})
// Writing the api to update the data in the database by using get and put http method
app.get('/getUser/:id', (req, res) => {
    const id = req.params.id;
    UserModel.findById({_id:id})
    .then(Tournaments => res.json(Tournaments))
    .catch(err => res.json(err))
})
app.put('/updateUser/:id', (req, res) => {
    const id = req.params.id;
    UserModel.findByIdAndUpdate({_id:id}, {
        TournamentID: req.body.TournamentID,
        TournamentName: req.body.TournamentName,
        StartDate: req.body.StartDate,
        EndDate: req.body.EndDate,
        SurfaceType: req.body.SurfaceType,
        VenueID: req.body.VenueID
    })
    .then(Tournaments => res.json(Tournaments))
    .catch(err => res.json(err))
})
// Writing the api to delete the data in the database
app.delete('/deleteUser/:id', (req, res) => {
    const id = req.params.id;
    UserModel.findByIdAndDelete({_id:id})
    .then(res => res.json(res))
    .catch(err => res.json(err))
})
app.listen(3001, () => {
    console.log("Server is running")
})
```

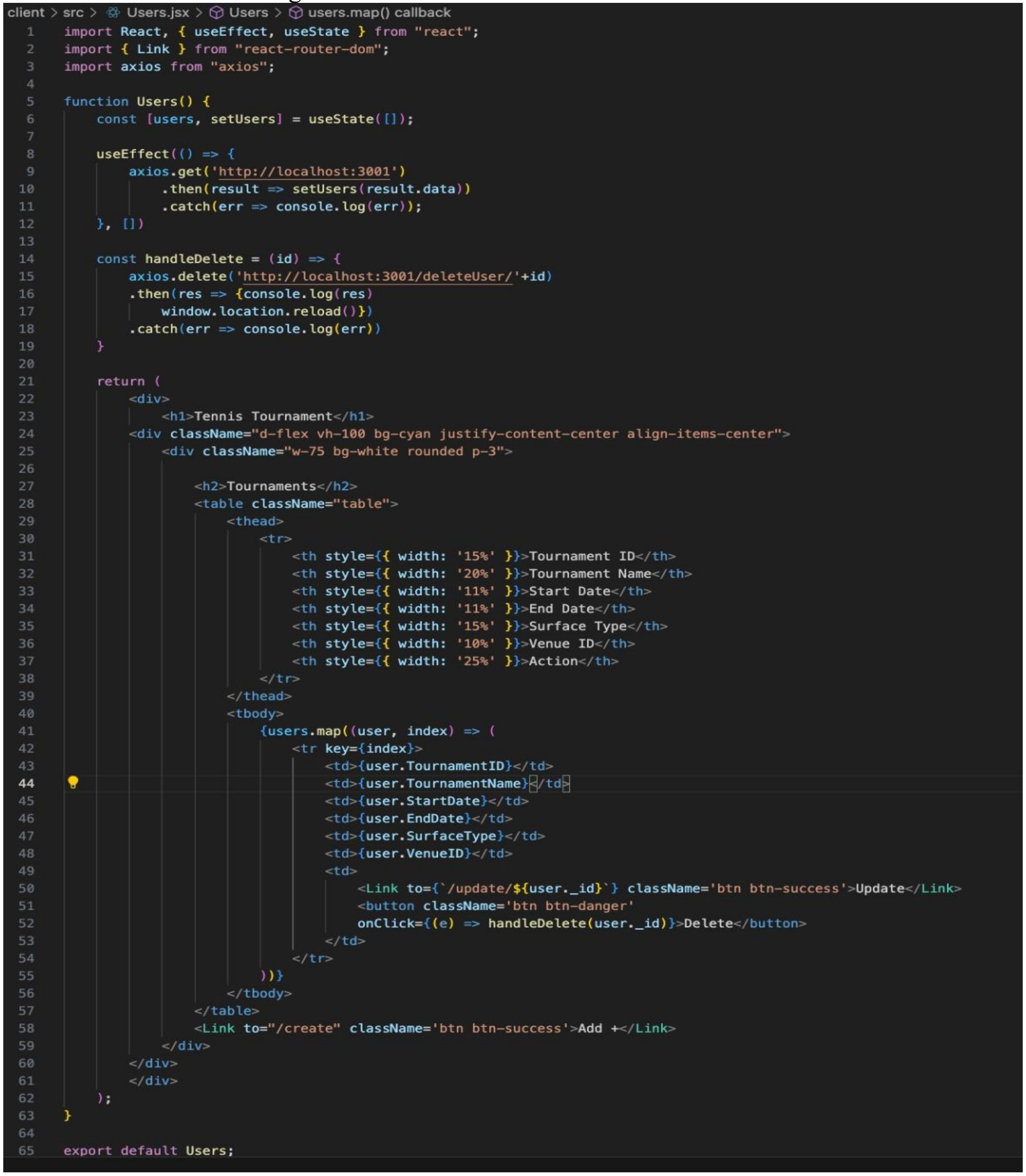
Explore the Client

The client contain the code for the front end where we will write different components to perform different activities. We are explicitly using jsx file which is used for React development to combine JavaScript with XML.



```
Explorer (⇧⌘E)
> OPEN EDITORS
∨ TENNIS
  ∨ client
    > node_modules
    > public
    ∨ src
      > assets
      App.css
      App.jsx
      CreateUser.jsx
      main.jsx
      UpdateUser.jsx
      Users.jsx
    .eslintrc.cjs
    .gitignore
    index.html
    package-lock.json
    package.json
    README.md
    vite.config.js
  > server
```

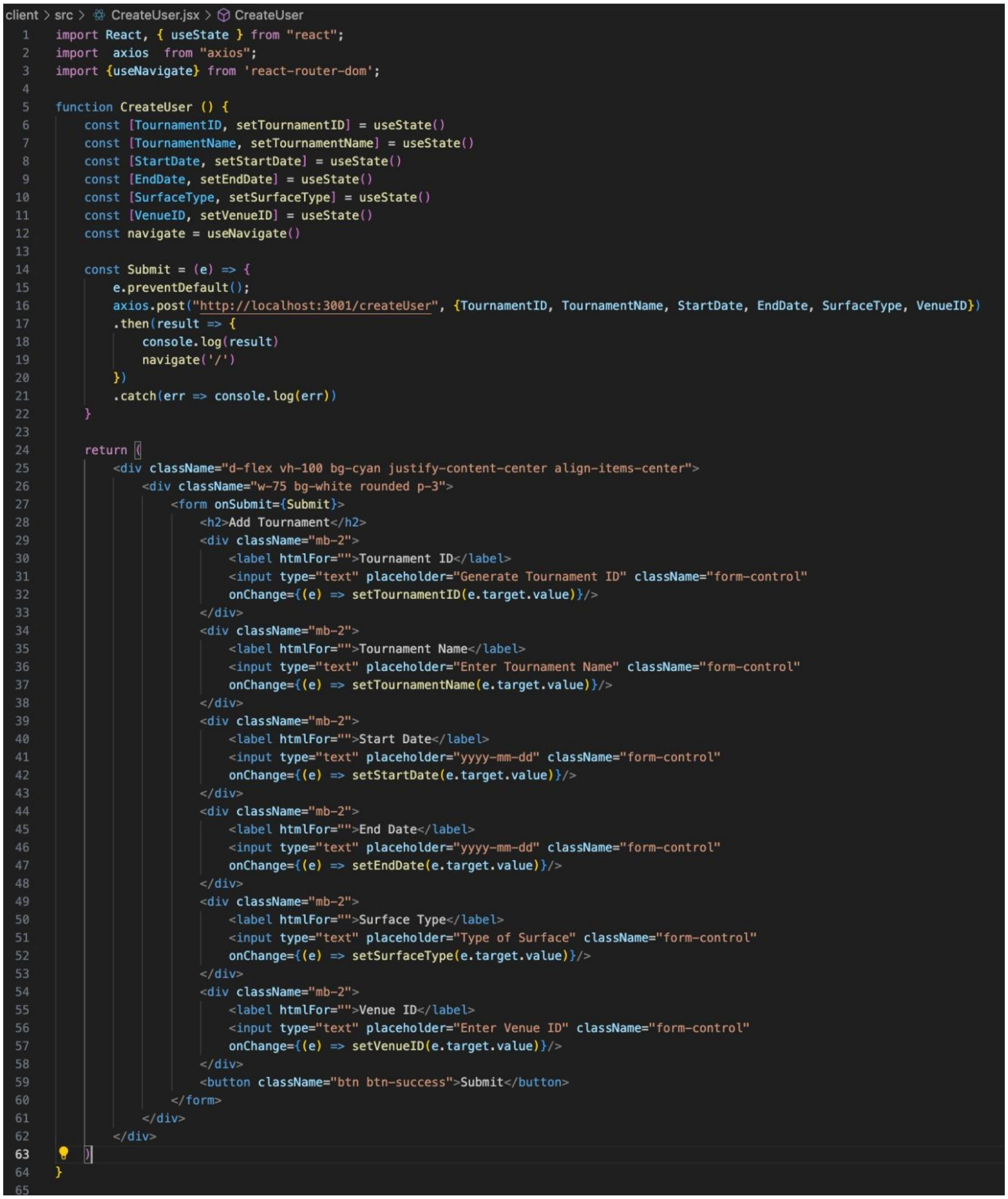
Users.jsx file contain the population of data code where it is populating the data taken from the frontend and it is also taking care of the deletion of the data.



client > src > Users.jsx > Users > users.map() callback

```
import React, { useEffect, useState } from "react";
import { Link } from "react-router-dom";
import axios from "axios";

function Users() {
    const [users, setUsers] = useState([]);

    useEffect(() => {
        axios.get('http://localhost:3001')
            .then(result => setUsers(result.data))
            .catch(err => console.log(err));
    }, [])

    const handleDelete = (id) => {
        axios.delete('http://localhost:3001/deleteUser/'+id)
        .then(res => {console.log(res)
            window.location.reload()})
        .catch(err => console.log(err))
    }

    return (
        <div>
            <h1>Tennis Tournament</h1>
        <div className="d-flex vh-100 bg-cyan justify-content-center align-items-center">
            <div className="w-75 bg-white rounded p-3">

                <h2>Tournaments</h2>
                <table className="table">
                    <thead>
                        <tr>
                            <th style={{ width: '15%' }}>Tournament ID</th>
                            <th style={{ width: '20%' }}>Tournament Name</th>
                            <th style={{ width: '11%' }}>Start Date</th>
                            <th style={{ width: '11%' }}>End Date</th>
                            <th style={{ width: '15%' }}>Surface Type</th>
                            <th style={{ width: '10%' }}>Venue ID</th>
                            <th style={{ width: '25%' }}>Action</th>
                        </tr>
                    </thead>
                    <tbody>
                        {users.map((user, index) => (
                            <tr key={index}>
                                <td>{user.TournamentID}</td>
                                <td>{user.TournamentName}</td>
                                <td>{user.StartDate}</td>
                                <td>{user.EndDate}</td>
                                <td>{user.SurfaceType}</td>
                                <td>{user.VenueID}</td>
                                <td>
                                    <Link to={`/update/${user._id}`} className='btn btn-success'>Update</Link>
                                    <button className='btn btn-danger'
                                    onClick={(e) => handleDelete(user._id)}>Delete</button>
                                </td>
                            </tr>
                        ))}
                    </tbody>
                </table>
                <Link to="/create" className='btn btn-success'>Add +</Link>
            </div>
        </div>
        </div>
    );
}

export default Users;
```

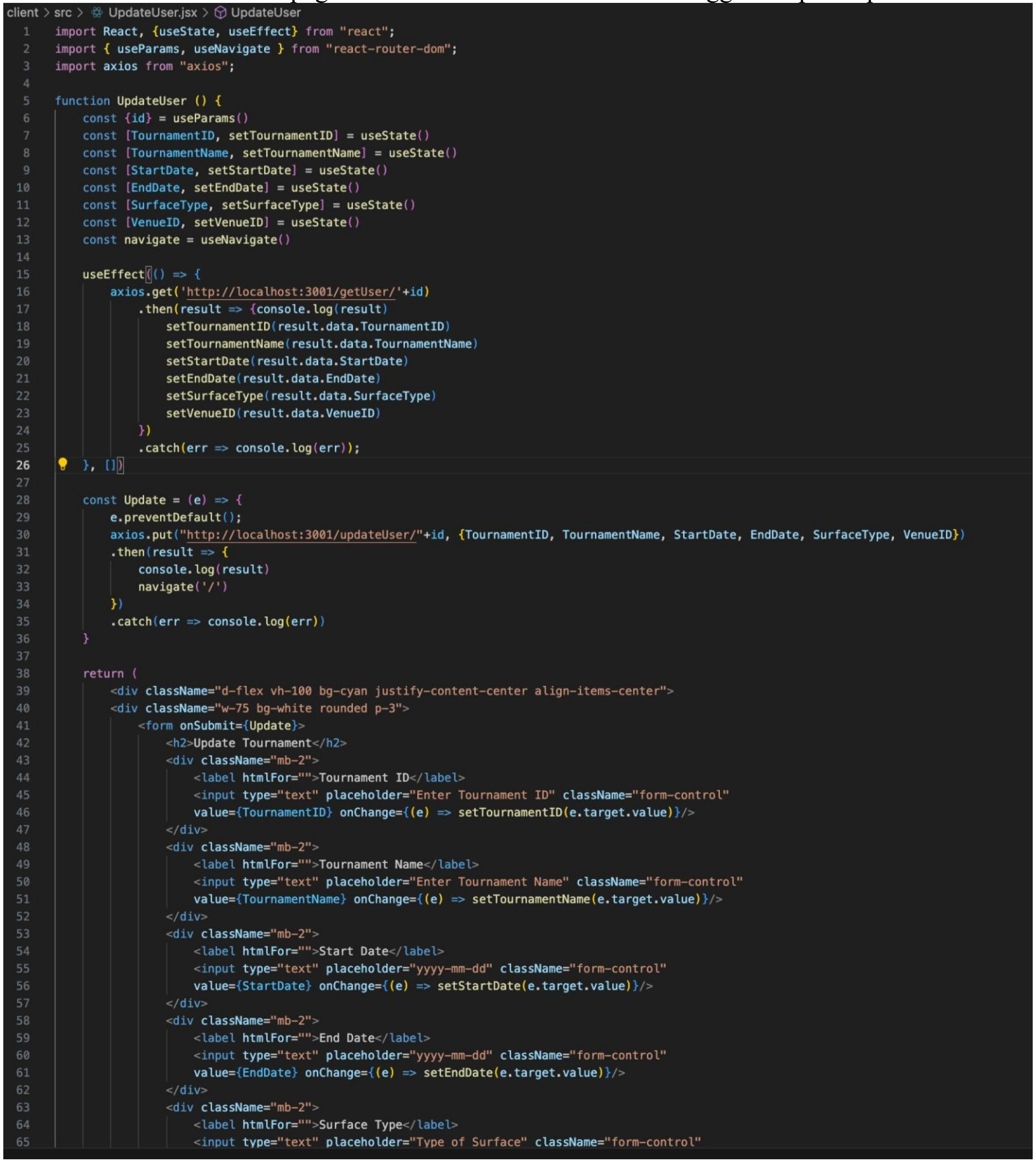
Createuser.jsx file contain the code for creating the new user. When the user will try to add new data then this file will called and the screen with empty values will be pop up in which we will send the data in post form.



client > src > CreateUser.jsx > CreateUser

```jsx
import React, { useState } from "react";
import  axios  from "axios";
import {useNavigate} from 'react-router-dom';

function CreateUser () {
    const [TournamentID, setTournamentID] = useState()
    const [TournamentName, setTournamentName] = useState()
    const [StartDate, setStartDate] = useState()
    const [EndDate, setEndDate] = useState()
    const [SurfaceType, setSurfaceType] = useState()
    const [VenueID, setVenueID] = useState()
    const navigate = useNavigate()

    const Submit = (e) => {
        e.preventDefault();
        axios.post("http://localhost:3001/createUser", {TournamentID, TournamentName, StartDate, EndDate, SurfaceType, VenueID})
        .then(result => {
            console.log(result)
            navigate('/')
        })
        .catch(err => console.log(err))
    }

    return (
        <div className="d-flex vh-100 bg-cyan justify-content-center align-items-center">
            <div className="w-75 bg-white rounded p-3">
                <form onSubmit={Submit}>
                    <h2>Add Tournament</h2>
                    <div className="mb-2">
                        <label htmlFor="">Tournament ID</label>
                        <input type="text" placeholder="Generate Tournament ID" className="form-control"
                        onChange={(e) => setTournamentID(e.target.value)}/>
                    </div>
                    <div className="mb-2">
                        <label htmlFor="">Tournament Name</label>
                        <input type="text" placeholder="Enter Tournament Name" className="form-control"
                        onChange={(e) => setTournamentName(e.target.value)}/>
                    </div>
                    <div className="mb-2">
                        <label htmlFor="">Start Date</label>
                        <input type="text" placeholder="yyyy-mm-dd" className="form-control"
                        onChange={(e) => setStartDate(e.target.value)}/>
                    </div>
                    <div className="mb-2">
                        <label htmlFor="">End Date</label>
                        <input type="text" placeholder="yyyy-mm-dd" className="form-control"
                        onChange={(e) => setEndDate(e.target.value)}/>
                    </div>
                    <div className="mb-2">
                        <label htmlFor="">Surface Type</label>
                        <input type="text" placeholder="Type of Surface" className="form-control"
                        onChange={(e) => setSurfaceType(e.target.value)}/>
                    </div>
                    <div className="mb-2">
                        <label htmlFor="">Venue ID</label>
                        <input type="text" placeholder="Enter Venue ID" className="form-control"
                        onChange={(e) => setVenueID(e.target.value)}/>
                    </div>
                    <button className="btn btn-success">Submit</button>
                </form>
            </div>
        </div>
    )
}
```

Updateuser.jsx file contains the code to update the record. When user will try to update the record then this file will show the page where user can edit the file and trigger the put request.


client > src > UpdateUser.jsx > UpdateUser

```jsx
import React, {useState, useEffect} from "react";
import { useParams, useNavigate } from "react-router-dom";
import axios from "axios";

function UpdateUser () {
    const {id} = useParams()
    const [TournamentID, setTournamentID] = useState()
    const [TournamentName, setTournamentName] = useState()
    const [StartDate, setStartDate] = useState()
    const [EndDate, setEndDate] = useState()
    const [SurfaceType, setSurfaceType] = useState()
    const [VenueID, setVenueID] = useState()
    const navigate = useNavigate()

    useEffect(() => {
        axios.get('http://localhost:3001/getUser/'+id)
            .then(result => {console.log(result)
                setTournamentID(result.data.TournamentID)
                setTournamentName(result.data.TournamentName)
                setStartDate(result.data.StartDate)
                setEndDate(result.data.EndDate)
                setSurfaceType(result.data.SurfaceType)
                setVenueID(result.data.VenueID)
            })
            .catch(err => console.log(err));
    }, [])

    const Update = (e) => {
        e.preventDefault();
        axios.put("http://localhost:3001/updateUser/"+id, {TournamentID, TournamentName, StartDate, EndDate, SurfaceType, VenueID})
        .then(result => {
            console.log(result)
            navigate('/')
        })
        .catch(err => console.log(err))
    }

    return (
        <div className="d-flex vh-100 bg-cyan justify-content-center align-items-center">
        <div className="w-75 bg-white rounded p-3">
            <form onSubmit={Update}>
                <h2>Update Tournament</h2>
                <div className="mb-2">
                    <label htmlFor="">Tournament ID</label>
                    <input type="text" placeholder="Enter Tournament ID" className="form-control"
                    value={TournamentID} onChange={(e) => setTournamentID(e.target.value)}/>
                </div>
                <div className="mb-2">
                    <label htmlFor="">Tournament Name</label>
                    <input type="text" placeholder="Enter Tournament Name" className="form-control"
                    value={TournamentName} onChange={(e) => setTournamentName(e.target.value)}/>
                </div>
                <div className="mb-2">
                    <label htmlFor="">Start Date</label>
                    <input type="text" placeholder="yyyy-mm-dd" className="form-control"
                    value={StartDate} onChange={(e) => setStartDate(e.target.value)}/>
                </div>
                <div className="mb-2">
                    <label htmlFor="">End Date</label>
                    <input type="text" placeholder="yyyy-mm-dd" className="form-control"
                    value={EndDate} onChange={(e) => setEndDate(e.target.value)}/>
                </div>
                <div className="mb-2">
                    <label htmlFor="">Surface Type</label>
                    <input type="text" placeholder="Type of Surface" className="form-control"
```

Conclusion Remarks

The Tennis Tournament Management system presented by Group 5 has demonstrated a comprehensive approach to handling various aspects of a tennis tournament through a database-driven application. The system's core functionality revolves around efficient storage and retrieval of crucial data like matches, players, tournaments, results, fixtures, venues, and courts. The project's use of advanced technologies like MongoDB for the backend and the MERN stack framework for the frontend allows for smooth CRUD operations, ensuring a seamless user experience.

One of the key strengths of this system is its flexible architecture, allowing users to perform a range of operations such as creating, reading, updating, and deleting tournament-related data. This flexibility is crucial for managing large-scale tennis tournaments where schedules and results can change frequently. The backend's connection to a database server on localhost 27017 ensures rapid data retrieval, while the frontend, built using React JS, provides an intuitive interface for users to interact with the application.

Additionally, the project's integration of robust database schema design with detailed entity relationships fosters a well-structured system that can handle complex tournament logistics. This design allows tournament organizers to maintain accurate records and simplifies the process of updating tournament information. The use of Python and Django for database connections and server-side scripting further enhances the project's scalability and performance.

The provided source code in the backend and frontend sections outlines how different components work together to create a coherent system. The thorough documentation of operations like data insertion, updating, and deletion gives users clear guidance on utilizing the system effectively.

REFERENCE

[1] Facebook. (2024). *“React Framework for UI Building”. Reactjs*
https://legacy.reactjs.org/

[2] Chris B. (January 4, 2024). *“MERN Stack Roadmap – How to Learn MERN and Become a Full-Stack Developer”. Freecodecamp*
https://www.freecodecamp.org/news/mern-stack-roadmap-what-you-need-to-know-to-build-full-stack-apps/

[3] Dr. AshokKumar P. (2024). *“Database Design”. myCourses*
https://webapps2.umassd.edu/ue/syllabi/3240/ce6/3126.pdf