

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №2**

з дисципліни

**«Бази даних і засоби управління»**

Виконав: студент 3 курсу

ФПМ групи КВ-83

Проценко Владислав

Перевірив: Павловський В.І.

Київ – 2020

**Варіант 20**

*Загальне завдання* роботи полягає у наступному:

1. Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати **вилучення** рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.
2. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не мовою програмування, а відповідним SQL-запитом**!

Нормалізована база даних з першої лабораторної роботи

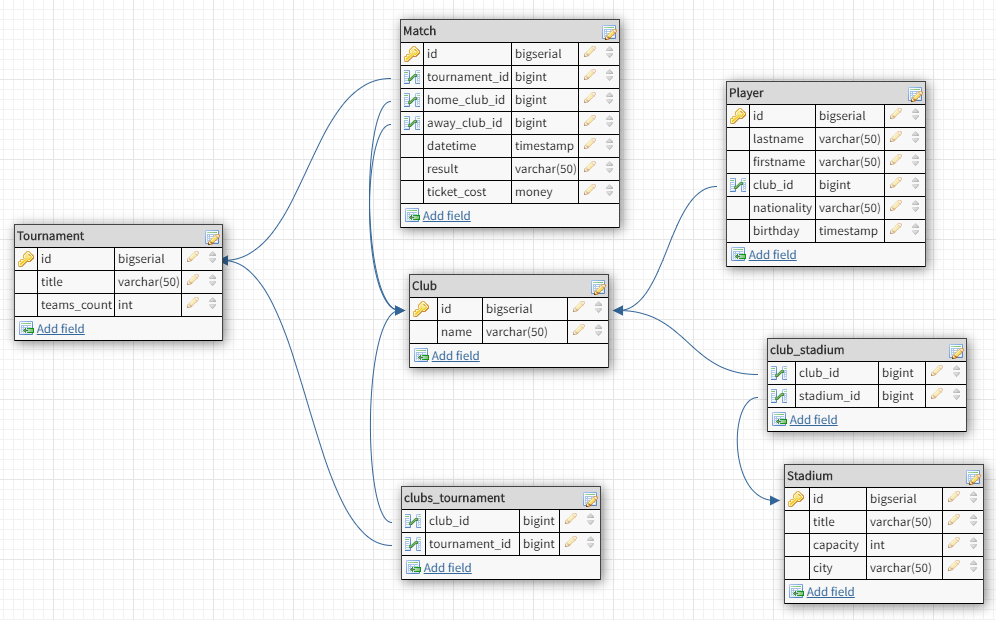


Рисунок 1 - Нормалізована база даних.

**Опис програми**

**Структура программи**

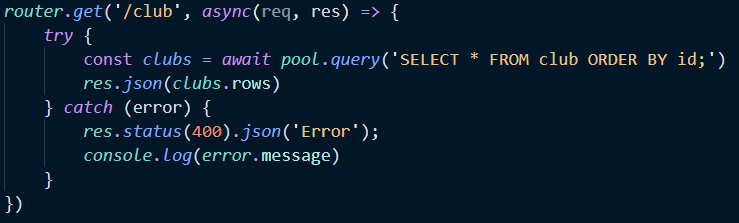
Програма складається з 4 основних модулів.

1. Файл index.js, який запускає сервер для прийому запитів клієнта;
2. Файли роутингу на серверній частині(для кожної таблиці свій окремий), вони обробляють запити, які надійшли на сервер;
3. Файл Table.js, модуль, для роботи в браузері з таблицею, а саме – пошук, створення, редагування, видалення даних з таблиці бази даних;
4. Файл routes.js для навігації користувача по програмі;
5. Модуль підключення до БД.

Навігація по модулях

1. Сервер
   1. [Запуск сервера](#index).
2. Файли роутингу
   1. [Для таблиці club](#clubRoutes);
   2. [Для таблиці stadium;](#stadiumRoutes)
   3. [Для таблиці player;](#playerRoutes)
   4. [Для таблиці tournament;](#tournamentRoutes)
   5. [Для таблиці match;](#matchRoutes)
   6. [Для таблиці clubs\_stadiums;](#clubs_tournaments_routes)
   7. [Для таблиці clubs\_tournametns.](#clubs_tournaments_routes)
3. Модулі для роботи з таблицями.
   1. [Задання сутностей;](#entity)
   2. [Модуль, який відображає їх у браузері.](#table)
4. Навігація користовуча
   1. [Модуль навігації;](#routes)
   2. [Відбраження навігації(Меню).](#menu)
5. [Модуль для підключення до БД.](#db)

**Обробка помилок**

Для цього завдання кожен SQL запит було виконанно в контсрукції try\_catch, приклад:  


Коли при запиті виникає якась помилка, то сервер поперне користувача відповідь зі статусом помилки(400) і текстом “error”. І коли користувач на свій запит до сервера отримує статус 400 йому на єкран буде виведено повідомлення про помилку. Приклад:   


Коли, користувач отримує відповідь, яка має статус 400(тобто помилки), функція window.alert виведе на єкран, що сталася помилка.

**Робота delete запитів**

В своїй роботі, при видаленні рядків, що мають посилання на зовнішні ключі, я встановлював значення NULL в рядку(якщо він дозволенний в таблиці), на який посилався данних зовнішній ключ, якшо NULL не дозволенно – видалява рядок.

Приклад з моеї програми, при видалення рядку з таблиці “club”:

*router.delete*('/club/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const*qu*=`*DELETE FROM club\_stadium WHERE club\_id =*${id}*;*

*DELETE FROM clubs\_tournaments WHERE club\_id =*${id}*;*

*DELETE FROM match WHERE home\_club\_id =*${id}*OR away\_club\_id =*${id}*;*

*UPDATE player SET club\_id = NULL WHERE club\_id =*${id}*;*

*DELETE FROM club WHERE id =*${id}*;*`

        const*response*=*await pool.query*(qu)

*res.json*('Successfuly deleted')

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

Тут можна побачити, що в поле club\_id таблиці player встановлюється значення NULL, а в інших випадках видаляються. Тому, що наприклад, футболіст може існувати без клубу, а футбольний матч, без цього клубу – вже не може.

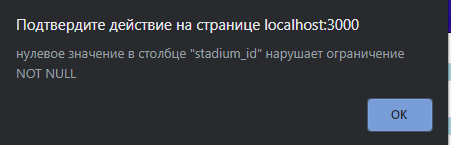
**Дослідження режимів обмеження ON DELETE**

*1. Режим CASCADE*

При видаленні запису з таблиці Stadium, запис з таблиці Club\_Stadium видаляється.

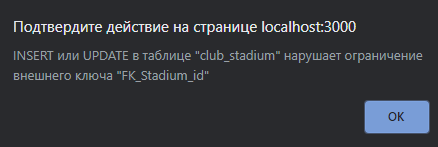
*2. Режим SET NULL*

При видаленні запису з таблиці Stadium, stadium\_id запис з таблиці Club\_Stadium встановлюється в null. Якщо в налаштуваннях таблиці вказати, що stadium\_id не може бути null, то перехоплюємо повідомлення про помилку.



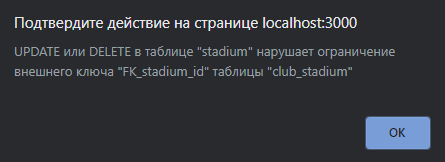
*3. Режим SET DEFAULT (значення за замовчуванням = 0)*

При видаленні запису з таблиці Stadium, перехоплюємо повідомлення про помилку, так як стадіону з id = 0 не існує.



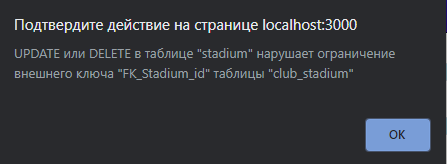
*4. Режим NO ACTION*

При видаленні запису з таблиці Stadium, виникає помилка; це поведінка за замовчуванням.



*5.Режим RESTRICT*

При видаленні запису з таблиці Stadium, виникає помилка; це пояснюється тим, що режим RESTRICT не дає можливості видалити батьківський рядок, якщо в нього є дочірні.



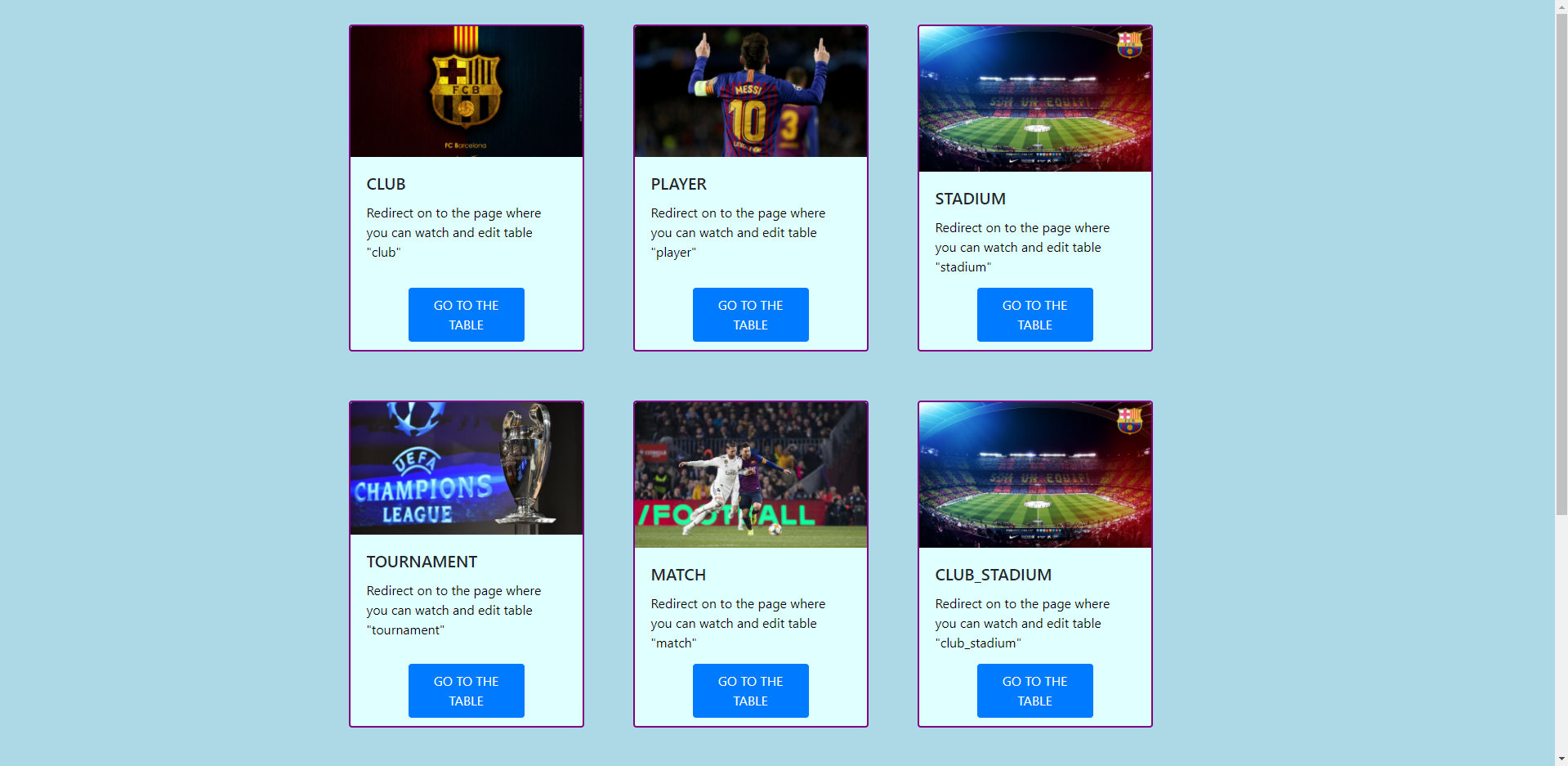
Головне меню програми  


Рисунок 2 - Меню

**Інтерфейс програми**

При виконані лабораторної роботи для всіх таблиць бази даних було розробленно однаковий інтерфейс. Розглянемо його на прикладі найбільшої з них.

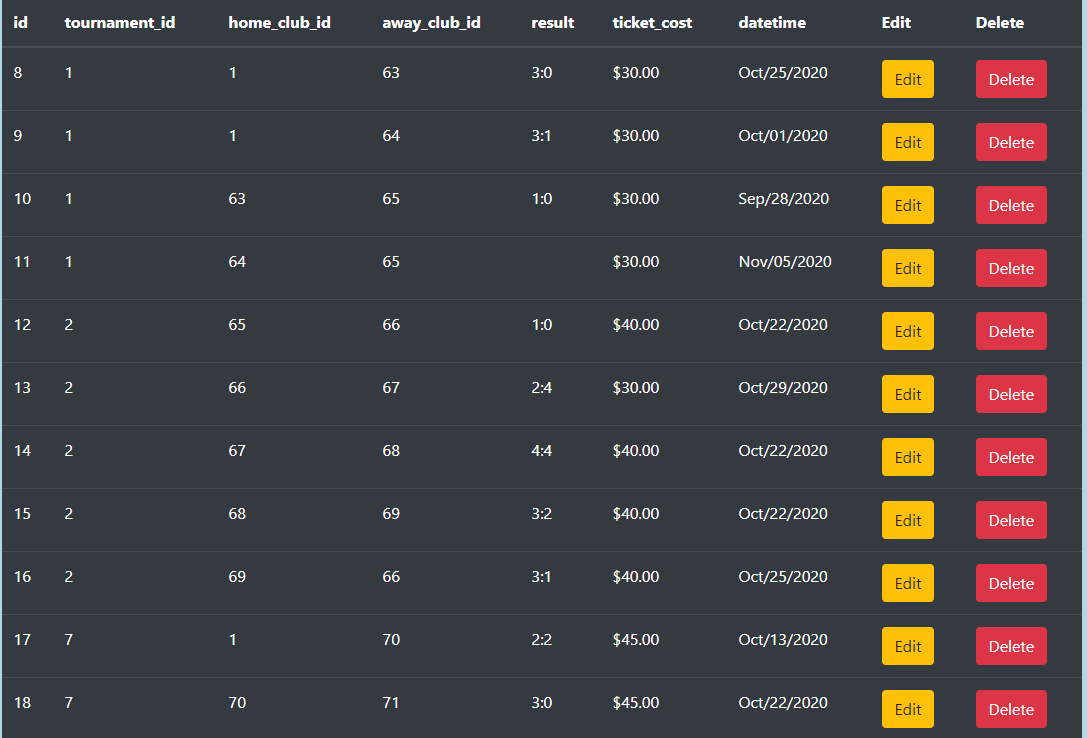
При переході на сторінку таблиці відображаються усі її записи.

Рисунок 3 - Приклад Таблиці.

Далі кожен запис можна видалити за допомогою кнопки “Delete”.  


Рисунок 4 - Кнопка видалення.  
Або можна редагувати натиснувши кнопку “Edit”.  

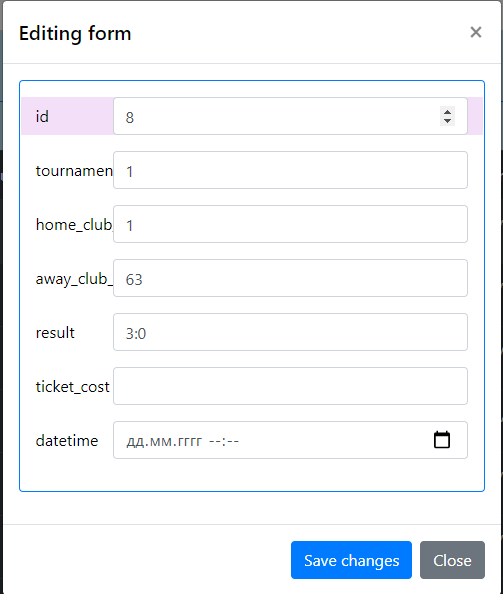

Рисунок 5 - Кнопка редагування.  
Після чого відкриється невелике віконце де можна ввести нові дані для цього рядка.  


Рисунок 6 - Форма редагування даних.  
Також вверху сторінки є форма для пошуку або створення нових записів.  


Рисунок 7 - Пошук рядків.  
Для того, щоб почати пошук – необхідно ввести інформацію хоча б в одне поле і натиснути кнопку “Find”.  

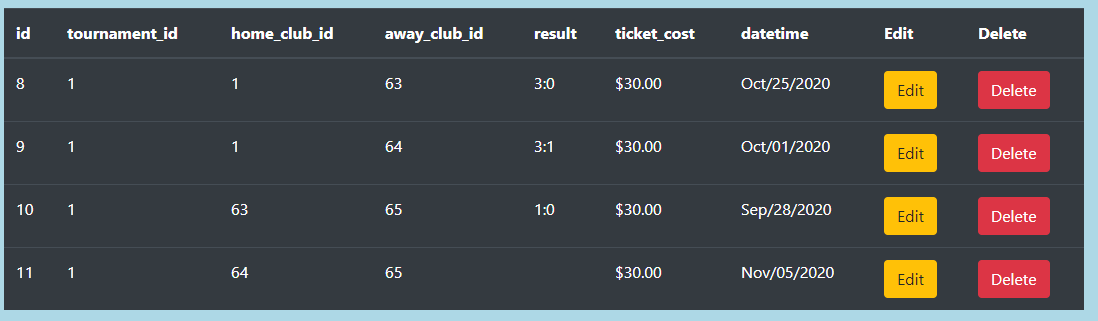

Рисунок 8 - Кнопка пошуку.  
Після чого, в таблиці будуть значення, які відповідають значенням в цій формі. Для прикладу, знайдемо всі рядки, в яких tournament\_id = 1. Для цього в поле tournament\_id введемо 1 і натиснемо кнопку “Find”. І отримаємо…  


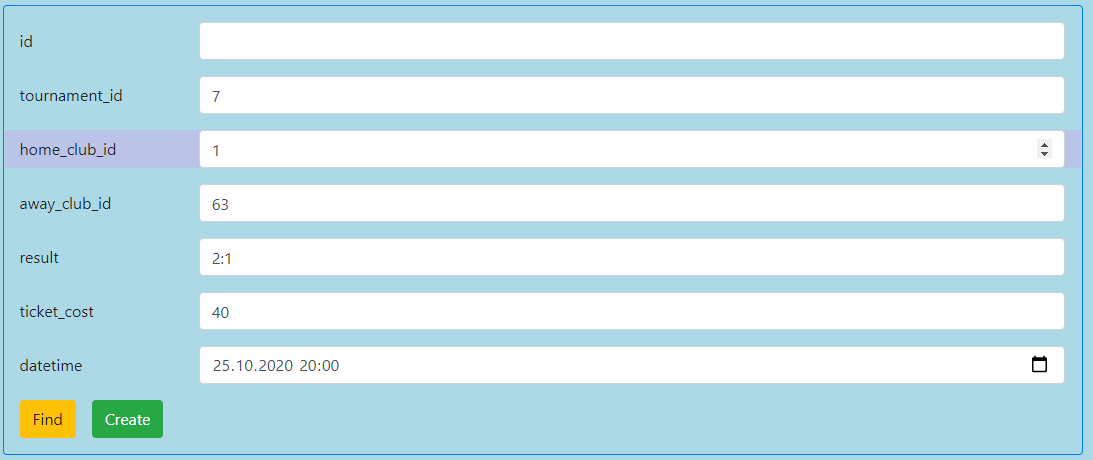
Рисунок 9 - Приклад пошуку.  
  
Далі для прикладу створимо, нехай такий рядок.  


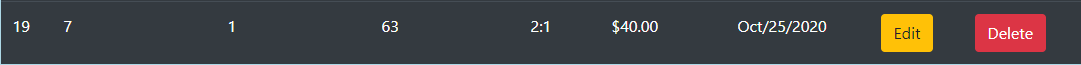
Рисунок 10 - Приклад створення.  
Після чого натиснемо кнопку “Create” і він допишеться в кінець таблиці.  


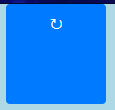
Рисунок 11 - Створений рядок.  
Також є кнопка, щоб видалити параметри пошуку, що виведе на екран всі рядки цієї таблиці.  


Рисунок 12 - Кнопка видалення пошуку.

Для виконання цієї лабораторної роботи, я використовував середовище розробки VS Code. Для підключення бази даних використовувалась платформа Node.js та додатково бібліотека “PG”.

Файл для підключення до БД.  
db.js

const*Pool*=*require*('pg')*.Pool*

//*connecting to the local database*

const*pool*=newPool({

*user*:"postgres",

*password*:"1928sfsf",

*host*:"localhost",

*port*:5432,

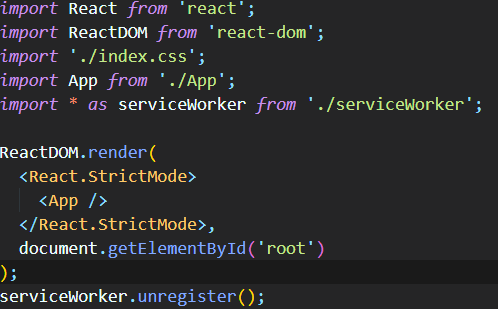
*database*:"footballtournaments",

});

module*.*exports = pool;

Код програми написан мовою програмування JavaScript.

**Клієнтська частина**

index.js  


App.js

*import* React *from* 'react';

*import* {BrowserRouter *as* Router} *from* 'react-router-dom'

*import* { Routes } *from* './Routes';

*function* App() {

*const* routes = Routes()

*return* (

    <div className="container">

        <Router>

            {routes}

        </Router>

    </div>

  );

}

*export* *default* App;

Routes.js

*import* React *from* 'react'

*import* {Switch, Route, Redirect} *from* 'react-router-dom'

*import* Table *from* './components/Table'

*import* { MainMenu } *from* './components/MainMenu'

*import* {types} *from* './components/Table'

*export* *const* Routes = () *=>* {

*//creating structure of my tables*

*//it is are arrays of objects, those pbjects represents columns*

*//first value in object is name of the column, second - it is type*

*const* clubTable = [

        {field: 'id', type: types.number},

        {field: 'name', type: types.string},

    ]

*const* clubsTournamentsTable = [

        {field: 'id', type: types.number},

        {field: 'club\_id', type: types.number},

        {field: 'tournament\_id', type: types.number},

    ]

*const* clubStadiumTable = [

        {field: 'club\_id', type: types.number},

        {field: 'stadium\_id', type: types.number},

    ]

*const* tournamentTable = [

        {field: 'id', type: types.number},

        {field: 'title', type: types.string},

        {field: 'teams\_count', type: types.number},

    ]

*const* playerTable = [

        {field: 'id', type:types.number},

        {field: 'firstname', type:types.string},

        {field: 'lastname', type:types.string},

        {field: 'club\_id', type: types.number},

        {field: 'birthday', type:types.date},

    ]

*const* stadiumTable = [

        {field: 'id', type:types.number},

        {field: 'title', type:types.string},

        {field: 'capacity', type:types.number},

        {field: 'city', type:types.string},

    ]

*const* matchTable = [

        {field: 'id', type:types.number},

        {field: 'tournament\_id', type:types.number},

        {field: 'home\_club\_id', type:types.number},

        {field: 'away\_club\_id', type:types.number},

        {field: 'result', type:types.string},

        {field: 'ticket\_cost', type:types.money},

        {field: 'datetime', type:types.datetime},

    ]

*return*(

*//creating routes, it is set up component(s) for each possible api of the application*

<Switch>

            <Route path="/" exact>

                <MainMenu />

            </Route>

            <Route path="/club" exact>

                <Table table={clubTable} tableName="club" randomize={true}/>

            </Route>

            <Route path="/player" exact>

                <Table table={playerTable} tableName="player" randomize={false}/>

            </Route>

            <Route path="/stadium" exact>

                <Table table={stadiumTable} tableName="stadium" randomize={true}/>

            </Route>

            <Route path="/match" exact>

                <Table table={matchTable} tableName="match" randomize={false}/>

            </Route>

            <Route path="/tournament" exact>

                <Table table={tournamentTable} tableName="tournament" randomize={true}/>

            </Route>

            <Route path="/clubs\_tournaments" exact>

                <Table table={clubsTournamentsTable} tableName="clubs\_tournaments" randomize={false}/>

            </Route>

            <Route path="/club\_stadium" exact>

                <Table table={clubStadiumTable} tableName="club\_stadium" randomize={false}/>

            </Route>

            <Redirect to="/" />

        </Switch>

    )}

Table.js

*import* React, {Fragment, useState, useEffect} *from* 'react'

*import* { EditRow } *from* './EditRow'

*export*const*serverUrl*='http://192.168.0.20:5000'

*export*const*types*={

*date*:'date',

*datetime*:'timestamp',

*money*:'money',

*number*:'integer',

*string*:'varchar',

}

const*Table*=(*props*)=>{

    //*array of all records, which need to be shown for viewer*

const[*rows*,*setRows*]=*useState*([])

    //*hold data, for filtering information*

const[*searchRow*,*setSearchRow*]=*useState*(*null*)

    //*contain count of records which should be randomly created*

const[*randomRows*,*setRandomRows*]=*useState*(0)

    //*variable, used for updating edit window(presented like modal)*

const[*toChange*,*setToChange*]=*useState*(*false*)

    //*makes GET requset to server and getting all rows of a table*

const*getRows*=*async*(*tableName*)=>{

*try*{

const*response*=*await fetch*(

`${serverUrl}*/*${tableName}`,{

*method*:'GET',

}

)

const*jsonData*=*await response.json*()

if(*response.status*===400){

*window.alert*(jsonData)

*return*

}

*setRows*(jsonData)

}*catch*(error){

*console.log*(*error.message*);

}

}

    //*loading all rows of the table*

*useEffect*(()=>{

*getRows*(*props.tableName*)

},[*props.tableName*])

    //*looking is it need to updat modal window*

*useEffect*(()=>{

*setToChange*(*true*)

},[rows])

    //*called when typing in search form*

const*searchInputHandler*=*event*=>{

        //*set value of field with name == to it is input name*

*setSearchRow*({

...searchRow,

[*event.target.name*]:*event.target.value*,

})

}

    //*called when trying to send data on server for filltering records*

const*toSearch*=*async*()=>{

*try*{

const*body*=searchRow

const*response*=*await fetch*(

`${serverUrl}*/*${*props.tableName*}*/search*`,{

*method*:'POST',

*headers*:{'Content-Type':'application/json'},

*body*:*JSON.stringify*(body)

})

const*data*=*await response.json*()

if(*response.status*===400){

*window.alert*(data)

*return*

}

*setRows*(data)

*setSearchRow*(*null*)

}*catch*(error){

*console.log*(*error.message*);

}

}

    //*Sends POST request on server to create new row in the table*

const*createRow*=*async*()=>{

*try*{

const*body*={}

*props.table.forEach*((*row*,*index*)=>{

body[*row.field*]=*document.getElementById*(`*search-create-*${*row.field*}`)*.value*

})

const*response*=*await fetch*(

`${serverUrl}*/*${*props.tableName*}*/new*`,{

*method*:'POST',

*headers*:{'Content-Type':'application/json'},

*body*:*JSON.stringify*(body)

}

)

const*data*=*await response.json*()

if(*response.status*===400){

*window.alert*(data)

*return*

}

*data.forEach*(*element*=>{

if(*Object.keys*(element)*.includes*(*props.table*[0]*.field*))*rows.push*(element)

})

*props.table.forEach*((*row*)=>{

*document.getElementById*(`*search-create-*${*row.field*}`)*.value*=''

})

*setSearchRow*(*null*)

*setRows*(rows)

}*catch*(error){

*console.log*(*error.message*)

}

}

    //*deleting row from rows array by it index*

const*deleteRow*=*async row\_id*=>{

*try*{

const*response*=*await fetch*(

`${serverUrl}*/*${*props.tableName*}*/*${rows[row\_id][*props.table*[0]*.field*]}`,{

*method*:'DELETE',

}

)

if(*response.status*===400){

*window.alert*(*await response.json*())

*return*

}else{

*setRows*(*rows.filter*((*row*,*index*)=>index!==row\_id))

}

            //*editingField(false)*

}*catch*(error){

*console.log*(*error.message*)

}

}

    //*load all records from current table*

const*reload*=(*tableName*)=>{

*setSearchRow*({})

*getRows*(tableName)

}

    //*update how many random records should be created*

const*countInputHandler*=(*e*)=>{

*setRandomRows*(*e.target.value*)

}

    //*send requset on the server to create randoms records*

const*createRandomRows*=*async*()=>{

*try*{

const*body*={*count*:randomRows}

const*response*=*await fetch*(

`${serverUrl}*/*${*props.tableName*}*/rand*`,{

*method*:'POST',

*headers*:{'Content-Type':'application/json'},

*body*:*JSON.stringify*(body)

}

)

const*data*=*await response.json*()

if(*response.status*===400){

*window.alert*(data)

*return*;

}

let\_=[...rows]

*data.forEach*(*element*=>{

if(*Object.keys*(element)*.includes*(*props.table*[0]*.field*))*\_.push*(element)

})

*setRows*(\_)

}*catch*(error){

*console.log*(*error.message*)

}

}

*return*(

<div *className*="container">

            <div >

                <span *onClick*={() => *reload*(*props.*tableName)} *className*="position-absolute btn btn-primary " *style*={{right: '20px', width: '100px', height: '100px'}} >&#8635;</span>

            </div>

            {/\**block for filltering rows or creating records*\*/}

            <div *className*="search-create-form">

                <h1 *className*="text-center">Searching/Creating form</h1>

                {/\**div play role of form*\*/}

                <div *className*="p-3 rounded border border-primary">

                {

*props.table.map*( (row, index) => {

                        lettype

                        switch (*row.*type) {

                            case *types.*date:

                                type='date'

*break*

                            case *types.*datetime:

                                type='datetime-local'

*break*

                            case *types.*money:

                                type='number'

*break*

                            case *types.*string:

                                type='text'

*break*

                            case *types.*number:

                                type='number'

*break*

                            default:

                                type='text'

*break*;

                        }

*return* (

                            <div *className*="form-group row" *key*={index}>

                            <label *className*="col-sm-2 col-form-label" *name*={*row.*field} *htmlFor*={`search-create-${*row.*field}`}>{*row.*field}</label>

                            {/\**block for typing data*\*/}

                            <div *className*="col-sm-10"><input *className*="form-control" *id*={`search-create-${*row.*field}`} *name*={*row.*field} *type*={type} *value*={searchRow ? searchRow[*row.*field] : ''} *onChange*={ e => *searchInputHandler*(e)} /></div>

                            </div>

                        )

                    })

                }

                    {/\**start searching for rows*\*/}

                    <button *className*="btn btn-warning mr-3" *onClick*={toSearch} >Find</button>

                    {/\**creates record*\*/}

                    <button *className*="btn btn-success" *onClick*={createRow} >Create</button>

                </div>

            </div>

            <table *className*='table mt-5 table-hover table-dark'>

                {/\**headers of the table*\*/}

                <thead>

                    <tr>

                        {

*props.table.map*( (row, index) => {

*return* (

                                    <Fragment *key*={index}>

                                        <th >{*row.*field}</th>

                                    </Fragment>

                                )

                            })

                        }

                        <th>Edit</th>

                        <th>Delete</th>

                    </tr>

                </thead>

                <tbody>

                    {

                        //*converting rows arrat into html*

*rows.map*( (row, index) => {

*return* (

                                <tr *key*={index}>

                                {

*props.table.map*( (entityRow, ind) => {

*return* (

                                        <td *key*={ind} >{row[*entityRow.*field]}</td>

                                        )

                                    })

                                }

                                    <td><EditRow *table*={*props.*table} *update*={getRows} *tableName*={*props.*tableName} *entity*={{...rows[index]}} *changeRow*={toChange}/></td>

                                    <td><button *className*="btn btn-danger" *onClick*={() => *deleteRow*(index)}>Delete</button></td>

                                </tr>

                            )

                        })

                    }

                </tbody>

            </table>

            {/\**block for creating random data if it possible for this table*\*/}

            {

*props.*randomize ?

                    (<div *className*="form-inline">

                        <div *className*="form-group mt-2" >

                            <input *className*="form-control mr-2" *type*="number" *value*={randomRows} *onChange*={e => *countInputHandler*(e)} />

                            <button *onClick*={createRandomRows} *className*="btn btn-success">Create random rows</button>

                        </div>

                    </div>)

                : null

            }

        </div>

)

}

*export* *default* Table

MainMenu.js

*import* React, {Fragment} *from* 'react'

*import* './styles.css'

*import* { Card } *from* './Card'

*export* *const* MainMenu = () *=>* {

*return* (

        <Fragment>

            <div className="d-flex flex-wrap">

                <Card link="/club" tableName="club" img="Barca" />

                <Card link="/player" tableName="player" img="Messi2" />

                <Card link="/stadium" tableName="stadium" img="Stadium" />

                <Card link="/tournament" tableName="tournament" img="UCL" />

                <Card link="/match" tableName="match" img="match" />

                <Card link="/club\_stadium" tableName="club\_stadium" img="Stadium" />

                <Card link="/clubs\_tournaments" tableName="clubs\_tournaments" img="UCL" />

            </div>

        </Fragment>

    )

}

Card.js

*import* React, {Fragment} *from* 'react'

*import* { NavLink } *from* 'react-router-dom'

*import* '../components/styles.css'

*export* *const* Card = (props) *=>* {

*return* (

        <Fragment>

            <div  className="card div-card" style={{width: '18rem'}}>

                <img className="card-img-top" src={process.env.PUBLIC\_URL + `/images/${props.img}.jpg`} alt={props.img} />

                <div className="card-body">

                    <h5 className="card-title text-uppercase">{props.tableName}</h5>

                    <p className="card-text" style={{color: 'black'}} >Redirect on to the page where you can watch and edit table "{props.tableName}"</p>

                    <NavLink className="nav-link btn btn-primary card-btn"  to={props.link}>

                        GO TO THE TABLE

                    </NavLink>

                </div>

            </div>

        </Fragment>

    )

}

EditRow.js

*import* React, {Fragment, useState, useEffect} *from* 'react'

*import* {types} *from* './Table'

*import* {serverUrl} *from* './Table'

*export* *const* EditRow = (props) *=>* {

*const* [entity, setEntity] = useState({...props.entity})

    useEffect(() *=>* {

        setEntity({...props.entity})

    }, [props.toChange, props.entity])

*const* editRowHandler = async () *=>* {

*try* {

            console.log('aaaaaaaa');

*const* body = entity

*const* response = *await* fetch(

                `${serverUrl}/${props.tableName}/${entity[props.table[0].field]}`, {

                    method: 'PUT',

                    headers: {'Content-Type': 'application/json'},

                    body: JSON.stringify(body)

                }

            )

*await* response.json()

            props.update(props.tableName)

            setEntity({})

        } *catch* (error) {

            console.log(error.message)

        }

    }

*const* editInputChangeHandler = event *=>* {

        console.log('aaaaaa, event', event.target.value);

        setEntity({...entity, [event.target.name]: event.target.value})

        console.log(entity)

    }

*return* (

        <Fragment>

            <button

                type="button"

                className="btn btn-warning"

                data-toggle="modal"

                data-target={`#edit-modal-${entity[props.table[0].field]}`}>Edit</button>

            <div style={{color: 'black'}} className="modal" id={`edit-modal-${entity[props.table[0].field]}`} tabIndex="-1" role="dialog">

                <div className="modal-dialog" role="document">

                    <div className="modal-content">

                    <div className="modal-header">

                        <h5 className="modal-title">Editing form</h5>

                        <button type="button" className="close" data-dismiss="modal" aria-label="Close">

                        <span aria-hidden="true">&times;</span>

                        </button>

                    </div>

                    <div className="modal-body">

                        {*/\* start of editing form \*/*}

                        <div id="form-editing" >

                            <div  className="p-3 border border-primary rounded mb-3">

                            {

                                props.table.map( (row, index) *=>* {

*let* type

*switch* (row.type) {

*case* types.date:

                                            type='date'

*break*

*case* types.datetime:

                                            type='datetime-local'

*break*

*case* types.money:

                                            type='number'

*break*

*case* types.string:

                                            type='text'

*break*

*case* types.number:

                                            type='number'

*break*

*default*:

                                            type='text'

*break*;

                                    }

*return* (

                                        <div className="form-group row" key={index}>

                                            <label className="col-sm-2 col-form-label" name={row.field} htmlFor={`#edit-${row.field}-${entity[props.table[0].field]}`}>{row.field}</label>

                                            <div className="col-sm-10"><input className="form-control" id={`edit-${row.field}-${entity[props.table[0].field]}`} name={row.field} type={type} onChange={e *=>* editInputChangeHandler(e)} value={entity[row.field]} /></div>

                                        </div>

                                    )

                                })

                            }

                            </div>

                            {*/\* ending of editing form \*/*}

                        </div>

                        {*/\* enging of editing form inside of modal\*/*}

                    </div>

                    <div className="modal-footer">

                        <button type="button" onClick={editRowHandler} data-dismiss="modal" className="btn btn-primary">Save changes</button>

                        <button type="button" onClick={()*=>* setEntity(props.entity)} className="btn btn-secondary" data-dismiss="modal">Close</button>

                    </div>

                    </div>

                </div>

            </div>

        </Fragment>

    )

}

**Серверна частина програми**index.js

*const* express = require('express')

*const* cors = require('cors')

*const* app = express()

app.use(cors())

app.use(express.json())

app.use('/', require('./routes/clubRoutes'))

app.use('/', require('./routes/playerRoutes'))

app.use('/', require('./routes/stadiumRouter'))

app.use('/', require('./routes/matchRoutes'))

app.use('/', require('./routes/tournamentRoutes'))

app.use('/', require('./routes/clubs\_tournaments'))

app.use('/', require('./routes/club\_stadium'))

app.listen(5000, () *=>* {

    console.log('Server has started on port 5000');

})

clubRoutes.js

const{*Router*}=*require*('express')

const*pool*=*require*('../db')

const*router*=*Router*()

//*returns all records in the table*

*router.get*('/club', *async*(req, res) => {

*try* {

        const*clubs*=*await pool.query*('SELECT \* FROM club ORDER BY id;')

*res.json*(*clubs.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

//*returns all rows which match to given paramatres*

*router.post*('/club/search', *async*(req, res) => {

*try* {

        const{*name*}=*req.body*

        const*qu*=`*SELECT \* FROM club WHERE LOWER(name) LIKE \'%*${name}*%\'*`

        const*response*=*await pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*create new record into the table by givin information*

*router.post*('/club/new', *async*(req, res) => {

*try* {

        const{*name*}=*req.body*

        letqu=`*INSERT INTO club(name) values (\'*${name}*\');*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM club ORDER BY id DESC FETCH FIRST 1 ROW ONLY;`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*create random record*

*router.post*('/club/rand', *async*(req, res) => {

*try* {

        const{*count*}=*req.body*

        letqu=`*INSERT INTO club(name) select getrandomstring(10) as name from generate\_series(1,*${count}*);*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM club ORDER BY id DESC FETCH FIRST ${count} ROW ONLY;`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*changes record with given id*

*router.put*('/club/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const{*name*}=*req.body*

        letqu=`*UPDATE club SET name = \'*${name}*\' WHERE id =*${id}*;*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM club WHERE id = ${id};`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows[0])

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*delete record by given id*

*router.delete*('/club/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const*qu*=`*DELETE FROM club\_stadium WHERE club\_id =*${id}*;*

*DELETE FROM clubs\_tournaments WHERE club\_id =*${id}*;*

*DELETE FROM match WHERE home\_club\_id =*${id}*OR away\_club\_id =*${id}*;*

*UPDATE player SET club\_id = NULL WHERE club\_id =*${id}*;*

*DELETE FROM club WHERE id =*${id}*;*`

        const*response*=*await pool.query*(qu)

*res.json*('Successfuly deleted')

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

module*.*exports = router

playerRoutes.js

const{*Router*}=*require*('express')

const*pool*=*require*('../db')

const*router*=*Router*()

const*selectAll*='SELECT id, firstname, lastname, club\_id, FORMAT(birthday::varchar, \'d\', \'en-US\') as birthday FROM'

//*returns all records in the table*

*router.get*('/player', *async*(req, res) => {

*try* {

        const*players*=*await pool.query*(`${selectAll}*player;*`)

*res.json*(*players.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

//*create random record*

*router.post*('/player/rand', *async*(req, res) => {

*try* {

        const{*count*}=*req.body*

        letqu=`*INSERT INTO player(name) select getrandomstring(10) as firstname, getrandomstring(10) as lastname, getrandomdate() as birthday, from generate\_series(1,*${count}*);*`

        letresponse=*await pool.query*(qu)

        qu = `${selectAll} player ORDER BY id DESC FETCH FIRST ${count} ROW ONLY;`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*returns all rows which match to given paramatres*

*router.post*('/player/search', *async*(req, res) => {

*try* {

        let{lastname,firstname,club\_id,birthday}=*req.body*

        letqu=`${selectAll}*player WHERE TRUE*`

        if(lastname !== undefined && *lastname.*length !== 0) {

            qu += `AND LOWER(lastname) LIKE \'%${lastname}%\' `

        }

        if(firstname !== undefined && *firstname.*length !== 0) {

            qu += `AND LOWER(firstname) LIKE \'%${firstname}%\' `

        }

        if(club\_id !== undefined && *club\_id.*length !== 0) {

            qu += `AND club\_id = ${club\_id} `

        }

        if(birthday !== undefined && *birthday.*length !== 0) {

            qu += `AND birthday = '${birthday}'::date`

        }

        qu += ';'

        const*response*=*await pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

*router.post*('/player/new', *async*(req, res) => {

*try* {

        let{lastname,firstname,club\_id,birthday}=*req.body*

        letqu=`*INSERT INTO player(firstname, lastname, club\_id, birthday) values (\'*${firstname}*\', \'*${lastname}*\',*${club\_id}*, \'*${birthday}*\'::date);*`

        letresponse=*await pool.query*(qu)

        qu = `${selectAll} player ORDER BY id DESC FETCH FIRST 1 ROW ONLY;`

        response = *await* *pool.query*(qu)

        //*response = await pool.query(qu)*

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*changes record with given id*

*router.put*('/player/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const{*firstname*,*lastname*,*club\_id*,*birthday*}=*req.body*

        letqu=``

        if(lastname !== undefined && *lastname.*length !== 0) {

            qu += `lastname = '${lastname}\', `

        }

        if(firstname !== undefined && *firstname.*length !== 0) {

            qu += `firstname = '${firstname}\', `

        }

        if(club\_id !== undefined && *club\_id.*length !== 0) {

            qu += `club\_id = ${club\_id}, `

        }

        if(birthday !== undefined && *birthday.*length !== 0) {

            qu += `birthday = '${birthday}'::date`

        }

        if(qu[*qu.*length - 1] === ',') qu[*qu.*length - 1] = ' '

        letquery2=`*UPDATE player SET*${qu}*WHERE id =*${id}*;*`

        letresponse=*await pool.query*(query2)

        qu = `${selectAll} player WHERE id = ${id};`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows[0])

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*delete record by given id*

*router.delete*('/player/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const*qu*=`*DELETE FROM player WHERE id =*${id}*;*`

        const*response*=*await pool.query*(qu)

*res.json*('Successfuly deleted')

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

module*.*exports = router

stadiumRoutes.js

const{*Router*}=*require*('express')

const*pool*=*require*('../db')

const*router*=*Router*()

//*returns all records in the table*

*router.get*('/stadium', *async*(req, res) => {

*try* {

        const*stadiums*=*await pool.query*(`*SELECT \* FROM stadium;*`)

*res.json*(*stadiums.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

//*create random record*

*router.post*('/stadium/rand', *async*(req, res) => {

*try* {

        const{*count*}=*req.body*

        letqu=`*INSERT INTO stadium(title, city, capacity) select getrandomstring(10) as title, getrandomstring(10) as city, randomnum(100000) as capacity from generate\_series(1,*${count}*);*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM stadium ORDER BY id DESC FETCH FIRST ${count} ROW ONLY;`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*returns all rows which match to given paramatres*

*router.post*('/stadium/search', *async*(req, res) => {

*try* {

        let{title,city,capacity}=*req.body*

        letqu=`*SELECT \* FROM stadium WHERE TRUE*`

        if(title !== undefined && *title.*length !== 0) {

            qu += `AND LOWER(title) LIKE \'%${title}%\' `

        }

        if(city !== undefined && *city.*length !== 0) {

            qu += `AND LOWER(city) LIKE \'%${city}%\' `

            //*qu += `AND LOCATE(\'${city}\', LOWER(city)) > 0`*

        }

        if(capacity !== undefined && *capacity.*length !== 0) {

            qu += `AND capacity = ${capacity} `

        }

        qu += ';'

        const*response*=*await pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*creates new record by give data*

*router.post*('/stadium/new', *async*(req, res) => {

*try* {

        let{title,city,capacity}=*req.body*

        letqu=`*INSERT INTO stadium(city, title, capacity) values (\'*${city}*\', \'*${title}*\',*${capacity}*);*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM stadium ORDER BY id DESC FETCH FIRST 1 ROW ONLY;`

        response = *await* *pool.query*(qu)

        //*response = await pool.query(qu)*

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*edit record with given id by replacing it with new data*

*router.put*('/stadium/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const{*city*,*title*,*capacity*}=*req.body*

        letqu=``

        if(title !== undefined && *title.*length !== 0) {

            qu += `title = '${title}\', `

        }

        if(city !== undefined && *city.*length !== 0) {

            qu += `city = '${city}\', `

        }

        if(capacity !== undefined && *capacity.*length !== 0) {

            qu += `capacity = ${capacity} `

        }

        if(qu[*qu.*length - 1] === ',') qu[*qu.*length - 1] = ' '

        letquery2=`*UPDATE stadium SET*${qu}*WHERE id =*${id}*;*`

        letresponse=*await pool.query*(query2)

        qu = `SELECT \* FROM stadium WHERE id = ${id};`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows[0])

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*delete record by given id*

*router.delete*('/stadium/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const*qu*=`*DELETE FROM club\_stadium WHERE stadium\_id =*${id}*;*

*DELETE FROM stadium WHERE id =*${id}*;*`

        const*response*=*await pool.query*(qu)

*res.json*('Successfuly deleted')

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

module*.*exports = router

matchRoutes.js

const{*Router*}=*require*('express')

const*pool*=*require*('../db')

const*router*=*Router*()

const*selectAll*='SELECT id, tournament\_id, home\_club\_id, away\_club\_id, to\_char(datetime, \'Mon/DD/YYYY\') as datetime, result, concat(\'$\', ticket\_cost::numeric) as ticket\_cost FROM'

//*returns all records in the table*

*router.get*('/match', *async*(req, res) => {

*try* {

        const*matchs*=*await pool.query*(`${selectAll}*match;*`)

*res.json*(*matchs.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

//*returns all rows which match to given paramatres*

*router.post*('/match/search', *async*(req, res) => {

*try* {

        let{result,tournament\_id,home\_club\_id,away\_club\_id,ticket\_cost,datetime}=*req.body*

        letqu=`${selectAll}*match WHERE TRUE*`

        if(result !== undefined && *result.*length !== 0) {

            qu += `AND LOWER(result) LIKE LOWER(\'%${result}%\') `

        }

        if(home\_club\_id !== undefined && *home\_club\_id.*length !== 0) {

            qu += `AND home\_club\_id = ${home\_club\_id} `

        }

        if(away\_club\_id !== undefined && *away\_club\_id.*length !== 0) {

            qu += `AND away\_club\_id = ${away\_club\_id} `

        }

        if(tournament\_id !== undefined && *tournament\_id.*length !== 0) {

            qu += `AND tournament\_id = ${tournament\_id} `

        }

        if(ticket\_cost !== undefined && *ticket\_cost.*length !== 0) {

            qu += `AND ticket\_cost::integer = ${ticket\_cost} `

        }

        if(datetime !== undefined && *datetime.*length !== 0) {

            qu += `AND datetime::date = '${datetime}'::date`

        }

        qu += ';'

        const*response*=*await pool.query*(qu)

*res.json*(*response.*rows)

        //*res.json('Server has got a data')*

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*create new record into the table by givin information*

*router.post*('/match/new', *async*(req, res) => {

*try* {

        let{result,tournament\_id,home\_club\_id,away\_club\_id,ticket\_cost,datetime}=*req.body*

        letqu=`*INSERT INTO match(tournament\_id, result, away\_club\_id, home\_club\_id, datetime, ticket\_cost) values (*${tournament\_id}*, \'*${result}*\',*${away\_club\_id}*,*${home\_club\_id}*, \'*${datetime}*\'::timestamp, \'*${ticket\_cost}*\'::money);*`

        letresponse=*await pool.query*(qu)

        qu = `${selectAll} match ORDER BY id DESC FETCH FIRST 1 ROW ONLY;`

        response = *await* *pool.query*(qu)

        //*response = await pool.query(qu)*

*res.json*(*response.*rows)

    } *catch* (error) {

*console.log*(*error.*message);

    }

})

//*changes record with given id*

*router.put*('/match/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const{*result*,*tournament\_id*,*home\_club\_id*,*away\_club\_id*,*ticket\_cost*,*datetime*}=*req.body*

        letqu=``

        if(result !== undefined && *result.*length !== 0) {

            qu += `result = \'${result}\' `

        }

        if(home\_club\_id !== undefined && *home\_club\_id.*length !== 0) {

            qu += `home\_club\_id = ${home\_club\_id} `

        }

        if(away\_club\_id !== undefined && *away\_club\_id.*length !== 0) {

            qu += `away\_club\_id = ${away\_club\_id} `

        }

        if(tournament\_id !== undefined && *tournament\_id.*length !== 0) {

            qu += `tournament\_id = ${tournament\_id} `

        }

        if(ticket\_cost !== undefined && *ticket\_cost.*length !== 0) {

            qu += `ticket\_cost = \'${ticket\_cost}\'::money `

        }

        if(datetime !== undefined && *datetime.*length !== 0) {

            qu += `datetime = '${datetime}'::timestamp`

        }

        if(qu[*qu.*length - 1] === ',') qu[*qu.*length - 1] = ' '

        letquery2=`*UPDATE match SET*${qu}*WHERE id =*${id}*;*`

        letresponse=*await pool.query*(query2)

        qu = `${selectAll} match WHERE id = ${id};`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows[0])

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*delete record by given id*

*router.delete*('/match/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const*qu*=`*DELETE FROM match WHERE id =*${id}*;*`

        const*response*=*await pool.query*(qu)

*res.json*('Successfuly deleted')

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

module*.*exports = router

tournamentRoutes.js

const{*Router*}=*require*('express')

const*pool*=*require*('../db')

const*router*=*Router*()

//*returns all records in the table*

*router.get*('/tournament', *async*(req, res) => {

*try* {

        const*tournaments*=*await pool.query*('SELECT \* FROM tournament;')

*res.json*(*tournaments.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

//*returns all rows which match to given paramatres*

*router.post*('/tournament/search', *async*(req, res) => {

*try* {

        const{*title*,*teams\_count*}=*req.body*

        letqu=`*SELECT \* FROM tournament WHERE TRUE*`

        if(teams\_count !== undefined && *teams\_count.*length !== 0) {

            qu += `AND teams\_count = ${teams\_count} `

        }

        if(title !== undefined && *title.*length !== 0) {

            qu += ` AND LOWER(title) LIKE LOWER(\'%${title}%\')`

        }

        qu += ';'

         ;

        const*response*=*await pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*create new record into the table by givin information*

*router.post*('/tournament/new', *async*(req, res) => {

*try* {

        const{*title*,*teams\_count*}=*req.body*

        letqu=`*INSERT INTO tournament(title, teams\_count) values (\'*${title}*\',*${teams\_count}*);*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM tournament ORDER BY id DESC FETCH FIRST 1 ROW ONLY;`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*create random record*

*router.post*('/tournament/rand', *async*(req, res) => {

*try* {

        const{*count*}=*req.body*

        letqu=`*INSERT INTO tournament(title, teams\_count) select UPPER(getrandomstring(4)) as title, (10 + (randomnum(10))) as teams\_count from generate\_series(1,*${count}*);*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM tournament ORDER BY id DESC FETCH FIRST ${count} ROW ONLY;`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*changes record with given id*

*router.put*('/tournament/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const{*title*,*teams\_count*}=*req.body*

        letqu=`*UPDATE tournament SET title = \'*${title}*\', teams\_count =*${teams\_count}*WHERE id =*${id}*;*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM tournament WHERE id = ${id};`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows[0])

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*delete record by given id*

*router.delete*('/tournament/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const*qu*=`*DELETE FROM tournament where id =*${id}*;*`

        const*response*=*await pool.query*(qu)

*res.json*('Successfuly deleted')

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

module*.*exports = router

club\_stadium.js

const{*Router*}=*require*('express')

const*pool*=*require*('../db')

const*router*=*Router*()

//*returns all records in the table*

*router.get*('/club\_stadium', *async*(req, res) => {

*try* {

        const*club\_stadiums*=*await pool.query*('SELECT \* FROM club\_stadium;')

*res.json*(*club\_stadiums.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

//*returns all rows which match to given paramatres*

*router.post*('/club\_stadium/search', *async*(req, res) => {

*try* {

        const{*club\_id*,*stadium\_id*}=*req.body*

        letqu=`*SELECT \* FROM club\_stadium WHERE TRUE*`

        if(club\_id !== undefined && *club\_id.*length !== 0) {

            qu += `AND club\_id = ${club\_id} `

        }

        if(stadium\_id !== undefined && *stadium\_id.*length !== 0) {

            qu += ` AND stadium\_id = ${stadium\_id} `

        }

        qu += ';'

        const*response*=*await pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*create new record into the table by givin information*

*router.post*('/club\_stadium/new', *async*(req, res) => {

*try* {

        const{*club\_id*,*stadium\_id*}=*req.body*

        letqu=`*INSERT INTO club\_stadium(club\_id, stadium\_id) values (*${club\_id}*,*${stadium\_id}*);*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM club\_stadium ORDER BY club\_id DESC FETCH FIRST 1 ROW ONLY;`

        response = *await* *pool.query*(qu)

        //*response = await pool.query(qu)*

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*changes record with given id*

*router.put*('/club\_stadium/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const{*club\_id*,*stadium\_id*}=*req.body*

        letqu=`*UPDATE club\_stadium SET club\_id =*${club\_id}*, teams\_count =*${stadium\_id}*WHERE id =*${id}*;*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM club\_stadium WHERE id = ${id};`

        response = *await* *pool.query*(qu)

*console.log*(*response.*rows);

*res.json*(*response.*rows[0])

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*delete record by given id*

*router.delete*('/club\_stadium/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const*qu*=`*DELETE FROM club\_stadium where id =*${id}*;*`

        const*response*=*await pool.query*(qu)

*res.json*('Successfuly deleted')

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

module*.*exports = router

clubs\_tournaments.js

const{*Router*}=*require*('express')

const*pool*=*require*('../db')

const*router*=*Router*()

//*returns all records in the table*

*router.get*('/clubs\_tournaments', *async*(req, res) => {

*try* {

        const*clubs\_tournamentss*=*await pool.query*('SELECT \* FROM clubs\_tournaments;')

*res.json*(*clubs\_tournamentss.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

//*returns all rows which match to given paramatres*

*router.post*('/clubs\_tournaments/search', *async*(req, res) => {

*try* {

        const{*club\_id*,*tournament\_id*}=*req.body*

        letqu=`*SELECT \* FROM clubs\_tournaments WHERE TRUE*`

        if(club\_id !== undefined && *club\_id.*length !== 0) {

            qu += `AND club\_id = ${club\_id} `

        }

        if(tournament\_id !== undefined && *tournament\_id.*length !== 0) {

            qu += ` AND tournament\_id = ${tournament\_id} `

        }

        qu += ';'

        const*response*=*await pool.query*(qu)

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*create new record into the table by givin information*

*router.post*('/clubs\_tournaments/new', *async*(req, res) => {

*try* {

        const{*club\_id*,*tournament\_id*}=*req.body*

        letqu=`*INSERT INTO clubs\_tournaments(club\_id, tournament\_id) values (*${club\_id}*,*${tournament\_id}*);*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM clubs\_tournaments ORDER BY id DESC FETCH FIRST 1 ROW ONLY;`

        response = *await* *pool.query*(qu)

        //*response = await pool.query(qu)*

*res.json*(*response.*rows)

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*changes record with given id*

*router.put*('/clubs\_tournaments/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const{*club\_id*,*tournament\_id*}=*req.body*

        letqu=`*UPDATE clubs\_tournaments SET club\_id =*${club\_id}*, teams\_count =*${tournament\_id}*WHERE id =*${id}*;*`

        letresponse=*await pool.query*(qu)

        qu = `SELECT \* FROM clubs\_tournaments WHERE id = ${id};`

        response = *await* *pool.query*(qu)

*res.json*(*response.*rows[0])

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message);

    }

})

//*delete record by given id*

*router.delete*('/clubs\_tournaments/:id', *async*(req, res) => {

*try* {

        const{*id*}=*req.params*

        const*qu*=`*DELETE FROM clubs\_tournaments where id =*${id}*;*`

        const*response*=*await pool.query*(qu)

*res.json*('Successfuly deleted')

    } *catch* (error) {

*res.status*(400)*.json*('Error');

*console.log*(*error.*message)

    }

})

module*.*exports = router