RetroGames

Manual Técnico

Proyecto 2 - Prácticas Iniciales

DBMS

Lo primero por hacer en la realización del proyecto es preparar nuestro entorno de trabajo, en esta ocasión se solicita trabajar con el DBMS MySql, y ya que se solicitó en el enunciado.

- Instalación:

  La instalación del DBMS se realizará en el sistema operativo Ubuntu, para lo que se utilizará la terminal para la instalación.

  primero ingresamos a la terminal del sistema, luego ingresamos los siguientes comandos:

  $ sudo apt update

  esto para actualizar los paquetes en el servidor.

  Luego con el siguiente comando instalamos MySQL:

  $ sudo apt install mysql-server

  con estos comando ya tendremos instalado el dbms y lo podremos utilizar con el comando ***mysql -u root -p***

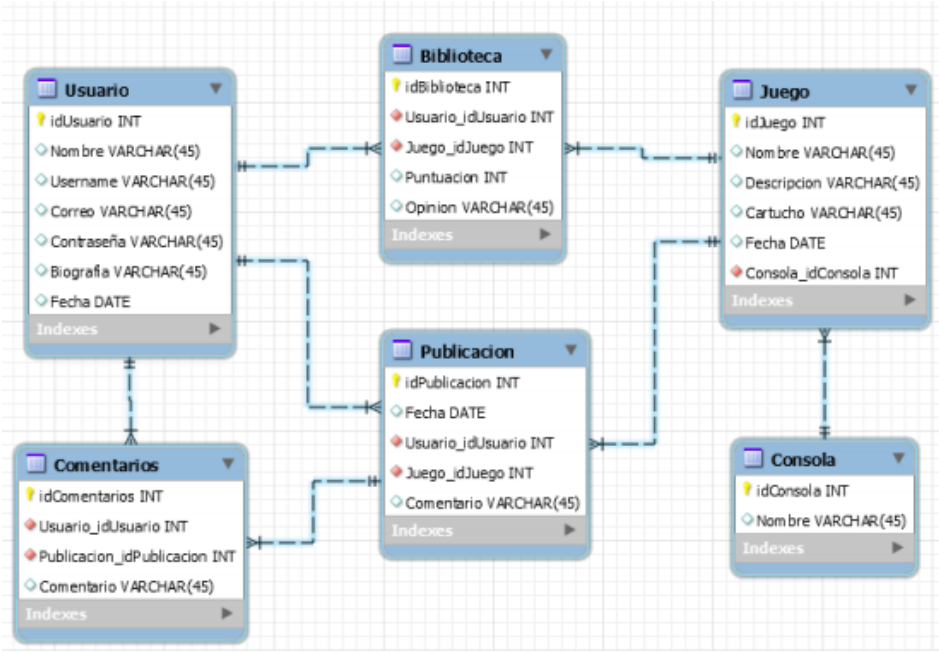
- Creación de las base de datos:

  para la creación de la base de datos que se utilizó en el proyecto se utilizaron algunos de los siguientes comandos:

```sql
#Creando base de datos
create database if not exists practicas_iniciales;
use practicas_iniciales;

#Creando tablas
create table usuario (
    id integer auto_increment not null,
    nombre varchar(45) not null,
    apellido varchar(45) not null,
    username varchar(45) not null,
    correo varchar(45) not null,
    password varchar(45) not null,
    biografia text not null,
    fecha date not null,
    primary key (id)
);
create table consola (
    id integer auto_increment not null,
    nombre varchar(45) not null,
    primary key (id)
);
```

en la primera línea se crea la base de datos “practicas_iniciales” y luego se usa esta misma.



- Creación del modelo:

  después se crean las tablas del modelo a utilizar con el comando “create table”

```
create table biblioteca (
    id integer auto_increment not null,
    id_usuario integer not null,
    id_juego integer not null,
    puntuacion integer not null,
    opinion text not null,
    primary key (id),
    foreign key (id_usuario) references usuario(id),
    foreign key (id_juego) references juego(id)
);
create table publicacion (
    id integer auto_increment not null,
    id_usuario integer not null,
    id_juego integer not null,
    fecha date not null,
    comentario text not null,
    primary key (id),
    foreign key (id_usuario) references usuario(id),
    foreign key (id_juego) references juego(id)
);
```

- Ingreso de datos:

  Una vez creado el modelo de la base de datos requiere de datos de prueba para comprobar el funcionamiento de la misma, por lo que ingresamos datos con el comando “insert into”:

```sql
#Cargando usuarios de prueba
insert into usuario (nombre,apellido,username,correo,password,biografia,fecha)
values('admin','admin','admin','admin','admin',
        'Administrador de la plataforma de juegos','2021-04-20'),
        ('Jose Andres','Rodas Arrecis','Alu','control.chi10@gmail.com','1234',
        'Estudiante de insgenieria en sistemas','2021-04-20'),
        ('Pedro Juan','Perez Batres','Piter','piter666@gmail.com','abcd',
        'Apasionado por los videojuegos retro','2021-04-20'),
        ('Jhon Stern','Smith Wick','Jhon','jhonwick@yahoo.com','pass',
        'Jugador profesional y streamer de videojuegos','2021-04-20');

#Cargando consolas de prueba
insert into consola (nombre)
values('Atari 2600'),('NES'),('SNES'),('Sega Genesis'),('Game Boy');

#Cargando juegos de prueba
insert into juego (id_consola,nombre,descripcion,cartucho,fecha)
values (1,'Pac-Man','Pac Man es un videojuego arcade creado por
el diseñador de videojuegos Toru Iwatani de la empresa Namco, y
distribuido por Midway Games al mercado estadounidense a principios
de los años 1980.','img/pacman.jpg','1980-04-21'),
        (1,'Space Invaders','Space Invaders es un videojuego
de arcade diseñado por Toshihiro Nishikado y lanzado
al mercado en 1978.','img/space.jpg','1978-04-21'),
        (1,'Asteroids','Asteroids es un popular videojuego arcade
basado en vectores lanzado en 1979 por Atari.  El objetivo del juego
es disparar y destruir asteroides evitando chocar contra los
```

- Consultas:

En el enunciado del proyecto se solicitan algunos datos para el usuario administrador, los cuales se realizan con las siguientes consultas.

```sql
#Consultas
#consulta 1
select u.nombre usuario, count(bi.id_juego) juegos from biblioteca bi
    inner join usuario u on bi.id_usuario = u.id
    group by usuario
    order by juegos desc limit 5;
#consulta 2
select u.nombre usuario, count(co.id_usuario) comentarios from comentario co
    inner join usuario u on co.id_usuario = u.id
    group by usuario order by comentarios desc limit 5;
#consulta 3
select j.nombre juego, c.nombre consola, avg(bi.puntuacion) promedio from biblioteca bi
    inner join juego j on bi.id_juego = j.id
    inner join consola c on j.id_consola = c.id
    group by juego, consola order by promedio desc limit 5;
#consulta 4
select nombre juego, fecha from juego
    order by fecha;
```

BACKEND

Para la realización del Backend de la aplicación se utilizó NodeJs el cual es un entorno multiplataforma de javascript. En este proyecto se utilizaron 3 archivos principales:

- Index.js:

  Este es el archivo principal de la aplicación del servidor, en el se instancian todas las dependencias necesarias para funcionar y se pone el puerto a disposición:

  En él se observa la instancia de el archivo rutas, en este archivo se describen las rutas que sirve nuestro backend.

- routes.js:

  En este archivo se especifican las rutas que nuestro servidor pondrá a disposición para que sean consumidas por nuestro frontend.

  hay 4 métodos principales que utilizamos en nuestro archivo de rutas:

  1. get: Con este método se puede devolver el resultado de una consulta sin editar nada de la tabla:

```
49  //get publicaciones filtradas por juego
50  router.get('/getPubliGames:nombre', async (req, res) => {
51      const { nombre } = req.params;
52      const sql = `
53      select pu.id, us.username user, ju.nombre game, pu.fecha date, pu
54      inner join usuario us on pu.id_usuario = us.id
55      inner join juego ju on pu.id_juego = ju.id
56      where ju.nombre = ?
57      order by pu.fecha desc
58      `
59      BD.query(sql, [nombre], (err, rows, fields) => {
60          if (!err) {
61              res.json(rows);
62          } else {
63              console.log('Error al hacer consulta: ' + err)
64          }
65      });
66  });
```

  2. post: Con el método post podemos insertar datos en una tabla de nuestra base de datos:

```
226  //post usuarios
227  router.post('/setUsers', async (req, res) => {
228      const { name, last_name, user, email, pass, bio, fecha } = req.body;
229      console.log(name, last_name, user, email, pass, bio, fecha)
230      const query = `
231      insert into usuario (nombre,apellido,username,correo,password,biografia,fecha)
232      values(?,?,?,?,?,?,?)
233      `;
234      BD.query(query, [name, last_name, user, email, pass, bio, fecha], (err, rows, fields) => {
235          if (!err) {
236              res.json({ Status: 'Persona ' + name + ' agregada!' });
237          } else {
238              console.log('Error al hacer consulta: ' + err)
239          }
240      });
241
242  })
```

3. put: Este método es utilizado como un Update, o sea que nos sirve para actualizar valores en nuestra base de datos.

```
308  //UPDATE MYSQL
309  router.put("/:id", async (req, res) => {
310      const { nombre, apellido, genero } = req.body;
311      const { id } = req.params;
312      const query = `
313          UPDATE persona
314          SET nombre = ?, apellido = ?, genero = ? WHERE id = ?
315      `;
316      BD.query(query, [nombre, apellido, genero, id], (err, rows, fields) => {
317          if (!err) {
318              res.json({ Status: 'Persona ' + nombre + ' editada!' });
319          } else {
320              console.log('Error al hacer consulta: ' + err)
321          }
322      })
323  })
```

4. delete: El método delete nos sirve para eliminar un registro en nuestra base de datos.

```
384  //DELETE MYSQL
385  router.delete("/:id", async (req, res) => {
386      const { id } = req.params;
387      const query = 'delete from persona where id = ?'
388      BD.query(query, [id], (err, rows, fields) => {
389          if (!err) {
390              res.json({ "msg": "Persona eliminada" })
391          } else {
392              console.log('Error al hacer consulta: ' + err)
393          }
394      });
395  })
```

- configdb.js

_____En este archivo realizamos la conexión con la base de datos que acabamos de crear, esto a través de un método al cual le pasamos los valores de nuestra base de datos.

```
Backend > src > config > JS configbd.js > ...
1   const mysql = require('mysql');
2   const { promisify } = require('util');
3
4   database = {
5       host: 'localhost',
6       user: 'root',
7       password: 'MAYQ',
8       database: 'practicas_iniciales'
9   }
10  const pool = mysql.createPool(database)
11  pool.getConnection((err,connection)=>{
12      if (err){
13          if (err.code === 'PROTOCOL_CONNECTION_LOST'){
14              console.log('La coneccion con la base de datos fue cerrada')
15          }
16          if (err.code === 'ER_CON_COUNT_ERROR'){
17              console.log('La base de datos tiene muchas conecciones')
18          }
19          if (err.code === 'ECONNREFUSED'){
20              console.log('La coneccion con la base de datos fue rechazada')
21          }
22      }
23      if(connection) connection.release();
24      console.log('Base de datos conectada!')
25      return;
```

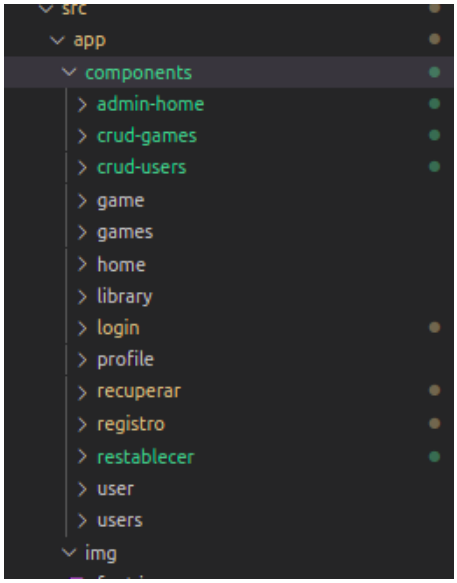
FRONTEND

La aplicación del lado del cliente se realizó con el Framework Angular, para lo que se utilizaron varios componentes para formar la aplicación.

- Componentes:

Se utilizaron varios componentes para la realización del proyecto como por ejemplo un login, una barra de navegación, etc.. estos componentes son creados a través de de Angular Cli el cual es la herramienta de consola de Angular. Los comandos para crear un componente son los siguientes:

$ ng generate component NombreComponente



```
18  export class LoginComponent implements OnInit {
19
20    //del NgModel
21    user_input: string = ''
22    pass_input: string = ''
23    //de la interfaz
24    Users: Usuario[] = []
25
26    constructor(public LoginService: GamesService, private router: Router) { }
27
28    //funcion para ingresar usuario
29    Ingresar(): void {
30
31      if (this.user_input == 'admin' && this.pass_input == 'admin') {
32        alert('¡Bienvenido usuario administrador!')
33        this.router.navigate(['/admin'])
34      } else {
35
36        this.LoginService.GetUsers().subscribe((res: Usuario[]) => {
37          let name_temp = ''
38          let user_valid: boolean
39          this.Users = res
40          //se recorren los resultados de la consulta a los usuarios
41          for (let user of this.Users) {
42            console.log(user.nombre)
43            if (user.username == this.user_input) {
44              if (user.password == this.pass_input) {
45                //Se valida el usuario y la contraseña
46                user_valid = true
47                name_temp = user.username
```

- Modelos:

Los modelos son interfaces que nos servirán para definir diferentes tipos de datos en el proyecto, como por ejemplo cuando mandamos una consulta a través de nuestro backend, necesitamos guardar los datos que nos devuelve en algún tipo específico de variable, alli entran nuestros modelos

Frontend > juegos > src > app > models > TS library.interface.ts > Library

```ts
export interface Library{
    id: number,
    id_usuario: number,
    id_juego: number,
    puntuacion: number,
    opinion:string
}
```

Frontend > juegos > src > app > models > TS game.interface.ts > Gar

```ts
export interface Game{
    id: number,
    id_consola: number,
    nombre: string,
    descripcion: string,
    cartucho:string,
    fecha: string
}
```

- Servicios:

  Los servicios nos servirán para realizar la comunicación entre el frontend y el servidor, estos se crean con el comando:

  $ ng generate service NombreServicio

  Y los archivos tienen la siguiente estructura:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from "@angular/common/http";
import { combineAll, map } from "rxjs/operators";
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class GamesService {

  constructor( private http: HttpClient ) { }

  headers: HttpHeaders = new HttpHeaders({
    "Content-Type": "application/json"
  })

  //Obtener usuarios
  GetUsers(){
    const url = 'http://localhost:3600/getUsers';
    return this.http.get(url);
  }
  //obtener publicaciones
  GetPosts(){
    const url = 'http://localhost:3600/getPosts';
    return this.http.get(url);
  }
  //obtener publicaciones por filtro usuarios
  GetPostsUsers(username:string){
    const url = 'http://localhost:3600/getPubliUsers'+username;
    return this.http.get(url);
  }
```

Para que un componente pueda utilizar un servicio debe de realizar la respectiva importación del mismo, esto se hace con la siguiente instrucción:

Frontend > juegos > src > app > components > game > TS game.component.ts > ...

```ts
import { Component, OnInit } from '@angular/core';
//importacion de servicios
import { GamesService } from '../../services/games.service'

//importacion de modelos
import { Game } from '../../models/game.interface'
import { Consola } from '../../models/consola.interface'
import { Usuario } from '../../models/user.interface';
import { Library } from 'src/app/models/library.interface';

//router
import { ActivatedRoute, RouterLink, Router } from '@angular/router'
import { isNullOrUndefined } from 'util';



@Component({
  selector: 'app-game',
  templateUrl: './game.component.html',
  styleUrls: ['./game.component.css']
})
```