

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

PRACTICAS INICIALES

CATEDRÁTICO: INGA. FLORITZA AVILA

TUTOR ACADÉMICO: STEVEN ZUÑIEGA/ EDUARDOMENDOZA



MANUAL DE **USUARIO**

Daniel Alejandro Castellanos Rodriguez

CARLOS FERNANDO MATTA PEÑA

CARNÉ: 202200176

202302190

SECCIÓN: C

GUATEMALA, 19 de septiembre de 2024

Manual Técnico

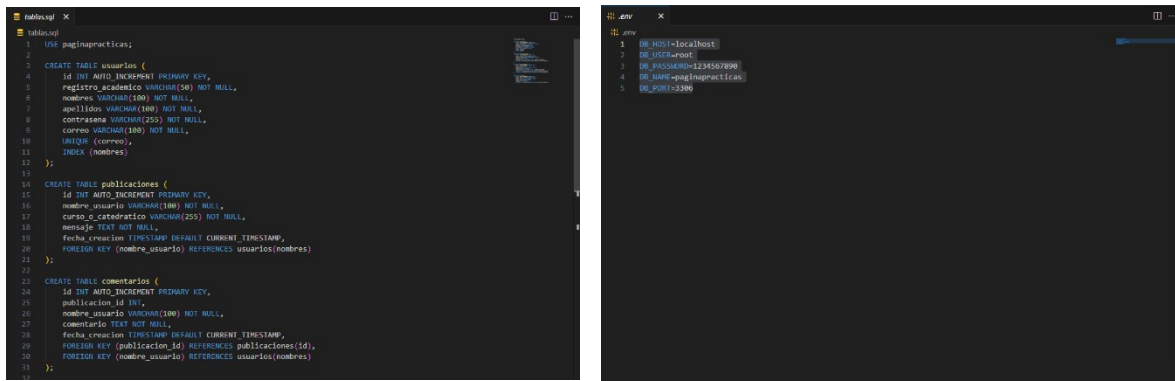
Requisitos del sistema:

- Instalación de Mysql para manejo de base de datos.
- Instalación de Node js para manejo de backend y todas las solicitudes.
- Instalación de React para la realización de la parte del cliente.

Estructura de código:

Para la realización del código se opto por utilizar las diferentes herramientas anteriormente mencionadas, por lado de la base de datos para su estructura únicamente se opto por crear un servidor el cual tendría las diferentes tablas para el almacenamiento de los datos dentro de la aplicación.

Se crearon tablas para los usuarios dentro del cual registra y almacena su información para que pueda iniciar sesión luego y manejar sus datos. Se crearon tablas para almacenar el contenido de publicaciones y comentarios y los cuales estaban conectados al nombre del usuario de quien lo realizaba. Y por último una tabla la cual maneja los diferentes cursos los cuales cada usuario puede guardar.



```
1 use pugnapracticar;
2
3 CREATE TABLE usuarios (
4   id INT AUTO INCREMENT PRIMARY KEY,
5   registro_academico VARCHAR(50) NOT NULL,
6   nombres VARCHAR(100) NOT NULL,
7   apellidos VARCHAR(100) NOT NULL,
8   contrasena VARCHAR(255) NOT NULL,
9   correo VARCHAR(100) NOT NULL,
10  INDEX (nombres)
11 );
12
13 CREATE TABLE publicaciones (
14   id INT AUTO INCREMENT PRIMARY KEY,
15   nombre_usuario VARCHAR(100) NOT NULL,
16   curso_catadratico VARCHAR(255) NOT NULL,
17   mensaje TEXT NOT NULL,
18   fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
19   FOREIGN KEY (nombre_usuario) REFERENCES usuarios(nombres)
20 );
21
22 CREATE TABLE comentarios (
23   id INT AUTO INCREMENT PRIMARY KEY,
24   publicacion_id INT,
25   nombre_usuario VARCHAR(100) NOT NULL,
26   comentario TEXT NOT NULL,
27   fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
28   FOREIGN KEY (publicacion_id) REFERENCES publicaciones(id),
29   FOREIGN KEY (nombre_usuario) REFERENCES usuarios(nombres)
30 );
31
32 CREATE TABLE cursos (
33   id INT AUTO INCREMENT PRIMARY KEY,
34   nombre VARCHAR(100) NOT NULL,
35   descripcion VARCHAR(255) NOT NULL,
36   fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
37   FOREIGN KEY (nombre_usuario) REFERENCES usuarios(nombres)
38 );
```

Ahora para la parte del backend es el lugar donde nosotros realizamos todas nuestras peticiones las cuales van directamente vinculadas a la base de datos para obtener los datos y hacer uso de estos datos para mandarlos a la parte del cliente ose el fronted.

Se utilizaron posts and gets para cada una de las peticiones, tanto el post para realizar todo el almacenamiento de los datos como los gets para la salida de estos datos, para acceder a estos datos se tuvo que hacer en el mismo backend los datos para acceder a nuestra base de datos y asi poder jalar los diferentes datos.

```
backendjs X
1 const express = require('express');
2 const cors = require('cors');
3 const fs = require('fs');
4 const mysql = require('mysql2');
5 require('dotenv').config();
6
7 const app = express();
8 const port = process.env.PORT || 5000;
9
10 const connection = mysql.createConnection({
11   host: process.env.DB_HOST,
12   user: process.env.DB_USER,
13   password: process.env.DB_PASSWORD,
14   database: process.env.DB_NAME,
15   port: process.env.DB_PORT
16 });
17
18 connection.connect((err) => {
19   if (err) {
20     console.error('Error connecting to the database:', err);
21     return;
22   }
23   console.log('Connected to the MySQL server.');
```

```
backendjs X
33 // Ruta de registro de usuarios
34 app.post('/register', (req, res) => {
35   const { registro_academico, nombre, apellidos, contrasena, correo } = req.body;
36   const sql = 'INSERT INTO usuarios (registro_academico, nombres, apellidos, contrasena, correo) VALUES';
37   connection.query(sql, [registro_academico, nombres, apellidos, contrasena, correo], (err, result) => {
38     if (err) {
39       console.error('Error inserting data:', err);
40       return res.status(500).send({ message: 'Error en el servidor al registrar el usuario', error: err });
41     }
42     res.status(201).send({ message: 'Usuario registrado correctamente' });
43   });
44 });
45
46 // Ruta de login de usuarios
47 app.post('/login', (req, res) => {
48   const { registro_academico, correo, contrasena } = req.body;
49   const sql = 'SELECT * FROM usuarios WHERE registro_academico = ? AND contrasena = ?';
50   connection.query(sql, [registro_academico, contrasena], (err, result) => {
51     if (err) return res.status(500).send(err);
52     if (result.length > 0) {
53       res.send({ message: 'Inicio de sesión exitoso', user: result[0] });
54     } else {
55       res.status(401).send({ message: 'Credenciales incorrectas' });
56     }
57   });
58 });
59
60 // Ruta para restablecer contraseña
61 app.post('/reset-password', (req, res) => {
62   const { registro_academico, correo, nueva_contrasena } = req.body;
63   const checkSql = 'SELECT * FROM usuarios WHERE registro_academico = ? AND correo = ?';
64   connection.query(checkSql, [registro_academico, correo], (err, result) => {
65     if (err) return res.status(500).send(err);
66     if (result.length > 0) {
67       const sql = 'UPDATE usuarios SET contrasena = ? WHERE registro_academico = ? AND correo = ?';
68       connection.query(sql, [nueva_contrasena, registro_academico, correo], (err, result) => {
69         if (err) return res.status(500).send(err);
70         res.status(200).send({ message: 'Contraseña restablecida exitosamente' });
71       });
72     } else {
73       res.status(404).send({ message: 'Usuario no encontrado' });
74     }
75   });
76 });
```

```
backendjs X
88 });
89
90 app.get('/publicaciones', (req, res) => {
91   const sql = 'SELECT p.id, p.nombre_usuario, p.cursos_o_catedratico, p.mensaje, p.fecha_creacion, c.id AS comentario_id, c.nombre_usuario AS comentario_usuario, c.comentario, c.fecha_creacion FROM publicaciones p LEFT JOIN comentarios c ON p.id = c.publicacion_id ORDER BY p.fecha_creacion DESC, c.fecha_creacion ASC';
92   connection.query(sql, (err, results) => {
93     if (err) {
94       console.error('Error fetching publicaciones:', err);
95       return res.status(500).send({ message: 'Error en el servidor al obtener las publicaciones', error: err });
96     }
97     const publicaciones = results.reduce((acc, row) => {
98       const pub = acc.find(p => p.id === row.id);
99       if (pub) {
100         if (row.comentario_id) {
101           pub.comentarios.push({
102             id: row.comentario_id,
103             nombre_usuario: row.comentario_usuario,
104             comentario: row.comentario,
105             fecha_creacion: row.comentario_fecha
106           });
107         } else {
108           acc.push({
109             id: row.id,
110             nombre_usuario: row.nombre_usuario,
111             cursos_o_catedratico: row.cursos_o_catedratico,
```

```
backendjs X
225 app.get('/perfil/nombre_usuario', (req, res) => {
226   connection.query(sqlUser, [nombreUsuario], (err, userResult) => {
227     if (userResult.length === 0) {
228       return res.status(404).send({ message: 'Usuario no encontrado' });
229     }
230     const user = userResult[0];
231
232     connection.query(sqlCursos, [nombreUsuario], (err, cursosResult) => {
233       if (err) {
234         console.error('Error fetching cursos aprobados:', err);
235         return res.status(500).send({ message: 'Error en el servidor al obtener los cursos aprobados' });
236       }
237       connection.query(sqlSum, [nombreUsuario], (err, sumResult) => {
238         if (err) {
239           console.error('Error fetching total credits:', err);
240           return res.status(500).send({ message: 'Error en el servidor al obtener el total de créditos' });
241         }
242         const totalCredits = sumResult[0].totalCredits || 0;
243         res.send({ user, cursos: cursosResult, totalCredits });
244       });
245     });
246   });
247 });
248
249 // Ruta principal
250 app.get('/', (req, res) => {
251   res.send('Hola desde el servidor a Practicas Iniciales');
252 });
```

Ahora para la parte donde el cliente hará los diferentes movimientos y obtención de los datos se uso el frotned donde con ayuda de React pudimos levantar nuestras diferentes paginas y de esta forma poder coenctar las diferentes paginas y hacer que el cliente interactue con las diferentes peticiones al servidor. Se utilizaron paginas para registrar, iniciar sesión, publicaciones, perfil, cursos cada uno en lenguaje de javascript y haciendo uso de Taiwind para mejorar la interfaz grafica.

```
iniciojs X
1 import React, { useState, useEffect, useReducer } from 'react';
2 import { useNavigate } from 'react-router-dom';
3 import axios from 'axios';
4 import { PublicacionContext } from './PublicacionContext';
5
6 const Inicio = () => {
7   const [publicaciones, setPublicaciones] = useReducer(PublicacionContext);
8   const [cursoFilter, setCursoFilter] = useState('');
9   const [catedraticoFilter, setCatedraticoFilter] = useState('');
10   const [nombreCursoFilter, setNombreCursoFilter] = useState('');
11   const [nombreCatedraticoFilter, setNombreCatedraticoFilter] = useState('');
12   const [user, setUser] = useState(null);
13   const [comentarios, setComentarios] = useState([]);
14
15   const navigate = useNavigate();
16
17   useEffect(() => {
18     const storedUser = localStorage.getItem('user');
19     if (storedUser) {
20       setUser(JSON.parse(storedUser));
21     } else {
22       navigate('/');
23     }
24   });
25
26   axios.get('http://localhost:5000/publicaciones')
27     .then(response => {
28       setPublicaciones(response.data);
29     })
30     .catch(error => {
31       console.error('Error fetching publicaciones:', error);
32     });
33 }, [navigate, setPublicaciones]);
```

```
loginjs X
1 import React, { useState } from 'react';
2 import axios from 'axios';
3 import { useNavigate } from 'react-router-dom';
4 import './App.css';
5
6 const login = () => {
7   const [registroAcademico, setRegistroAcademico] = useState('');
8   const [password, setPassword] = useState('');
9   const [error, setError] = useState('');
10   const navigate = useNavigate();
11
12   const handleLogin = async (event) => {
13     event.preventDefault();
14     try {
15       const response = await axios.post('http://localhost:5000/login', {
16         registro_academico: registroAcademico,
17         contrasena: password,
18       });
19       console.log(response.data);
20       localStorage.setItem('user', JSON.stringify(response.data.user));
21       navigate('/inicio');
22     } catch (error) {
23       console.error('Error durante el inicio de sesión:', error);
24       setError('Error durante el inicio de sesión. Por favor, verifica tus credenciales.');
```

En conclusion pudimos realizar la función del servidor a través de peticiones API/ rest conectando backend con fronted y todo esto almacenando en base de datos.