Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Practicas Iniciales
Sección: “ F- ”

Manual Técnico

Grupo No.3

Integrantes:

201800979 - Mauricio Alejandro Sagastume Barrios
202001497 - Antony Stive Fuentes Marroquín
202001523 - Luis Gabriel López Polanco
202100215 - Ángel Guillermo de Jesús Pérez Jiménez
202200129 - Luis Pablo Manuel García López
202210009 - José Luis Antonio González García

Manual Técnico del Servidor

Componentes de la Aplicación:

index.js

Este archivo es el punto de entrada principal de la aplicación y configura el servidor web utilizando *Express.js*. Aquí se encuentran las siguientes configuraciones y rutas:

- **Configuraciones del Servidor:**
  - Uso del middleware *cors* para habilitar el control de acceso a recursos compartidos entre diferentes dominios (*CORS*).
  - Uso del middleware *morgan* en modo *dev* para registrar las peticiones
    HTTP en la consola.

- **Configuraciones de Archivos del Servidor:**
  - Configuración de límites de carga para datos *JSON*, datos codificados y datos de texto para limitar el tamaño de los archivos que se pueden cargar.

- **Rutas:**
  - Ruta raíz ("/"): Devuelve un mensaje *"Hello from Backend"* como respuesta.
  - Uso del archivo de rutas *"./Routes/routes.js"* para definir rutas adicionales relacionadas con *usuarios*, *publicaciones*, *cursos* y *comentarios*.

- **Inicio del Servidor:**
  - El servidor se inicia en el puerto 4000 y se conecta a la base de datos utilizando el módulo de conexión definido en *"./DataBase/Database.js"*.

```
const express = require("express");
const cors = require("cors");
const morgan = require("morgan");
const app = express();
const ConexionDB = require("./DataBase/Database")

/*Configuraciones del servidor*/
app.use(cors());/* Politicas del servidor */
app.use(morgan("dev")); /* Controlador de peticiones en consola*/

/* Configuraciones de archivos del servidor*/
app.use(
    express.json({
        limit: "200mb", /* El servidor aguanta archivos no mas de 200 mb */
    })
);
app.use(
    express.urlencoded({
        limit: "200mb", /* El servidor aguanta archivos no mas de 200 mb */
        extended: true,
    })
);
app.use(
    express.text({
        limit: "200mb", /* El servidor aguanta archivos no mas de 200 mb */
    })
);

/* Creacion de las rutas */
app.get("/", (req, res) => {
    res.status(200).send({ message: "Hello from Backend" });
});

app.use("/Grupo3", require("./Routes/routes.js"));

app.listen(4000, () => {
    ConexionDB.Conectar()
    console.log("Servidor levantado con exito en el puerto " + 4000);
});
```

Carpeta Routes

Este archivo contiene las rutas y controladores de la aplicación. Aquí se definen las rutas para diversas operaciones relacionadas con *usuarios*, *publicaciones*, *cursos* y *comentarios*. Algunas de las rutas incluyen:

- Operaciones relacionadas con *usuarios* (*registro*, *inicio de sesión*, *actualización*, *lista de usuarios*, *datos del usuario*, *restablecimiento de contraseña*).

- Operaciones relacionadas con *comentarios* (*agregar comentario*, *listar comentarios*).
- Operaciones relacionadas con *publicaciones* (*agregar publicación*, *listar publicaciones*, *listar mis publicaciones*).
- Operaciones relacionadas con *cursos* y *catedráticos* (*listar cursos*, *listar catedráticos*).

```
const express = require("express");
const router = express.Router();
const {addUser,Login,updateUser,ListUser,MydataUser,ForgotPassword,TotalCreditos} = require("../Controllers/userscontrollers")
const {Listcourses,Listcate,addCurso,MyCourses} = require("../Controllers/coursecontrollers")
const {addpublication,listpublications,mylistpublications} = require("../Controllers/publicationcontrollers")
const {addcomment,listcomment} = require("../Controllers/commentscontrollers")

/* Usuarios */
router.post('/addUser', addUser);
router.post('/Login', Login);
router.post('/updateUser', updateUser);
router.post('/ListUser', ListUser);
router.post('/MydataUser', MydataUser);
router.post('/ForgotPassword', ForgotPassword);
router.post('/TotalCreditos', TotalCreditos);

/* Comentarios */
router.post('/addcomment', addcomment);
router.post('/listcomment', listcomment);

/* Publicaciones */
router.post('/addpublication', addpublication);
router.get('/listpublications', listpublications);
router.post('/mylistpublications', mylistpublications);

/* Cursos */
router.get('/Listcourses', Listcourses);
router.get('/Listcate', Listcate);
router.post('/addCurso', addCurso);
router.post('/MyCourses', MyCourses);

module.exports = router;
```

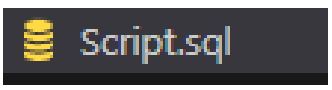
DataBase

Database.js

Este archivo contiene la configuración y la lógica de conexión a la base de datos *MySQL*. Aquí se definen las credenciales de la base de datos y se utiliza el módulo *mysql* para establecer la conexión. También se exporta la conexión para que esté disponible en otros archivos de la aplicación. La función *Conectar* se utiliza para establecer la conexión a la base de datos.

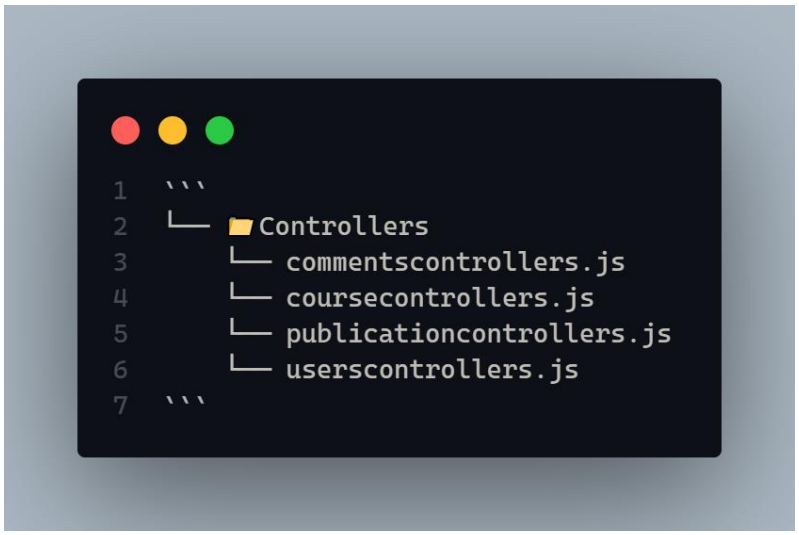
```js
const mysql = require('mysql')

/*Credenciales de su base de datos*/
const conexion = mysql.createConnection(
  {
    host: '127.0.0.1',
    user: 'root',
    password: '1234',
    database: 'usuarios'
  }
);

const Conectar = () => {
    conexion.connect( err => {
        if(err) throw err
        console.log("conectado a la DB");
    })
}


module.exports = {
    Conectar,
    conexion
}
```

Script.sql



Este archivo *SQL* contiene los comandos para crear la estructura de la base de datos, incluyendo tablas para *usuarios*, *catedráticos*, *cursos*, *publicaciones*, *comentarios* y *asignaciones*. Además, se incluyen comandos *SQL* para insertar datos de ejemplo en estas tablas. Asegúrese de ejecutar estos comandos en su sistema de gestión de bases de datos *MySQL* antes de utilizar la aplicación.

Controller



userscontrollers.js

Este archivo contiene controladores relacionados con las operaciones de *usuario*, como *registro*, *inicio de sesión*, *actualización de datos*, *listado de usuarios*, *obtención de datos de usuario* y *restablecimiento de contraseña*. Los controladores utilizan el módulo *mysql* para interactuar con la base de datos y responder a las solicitudes *HTTP*.

publicationcontrollers.js

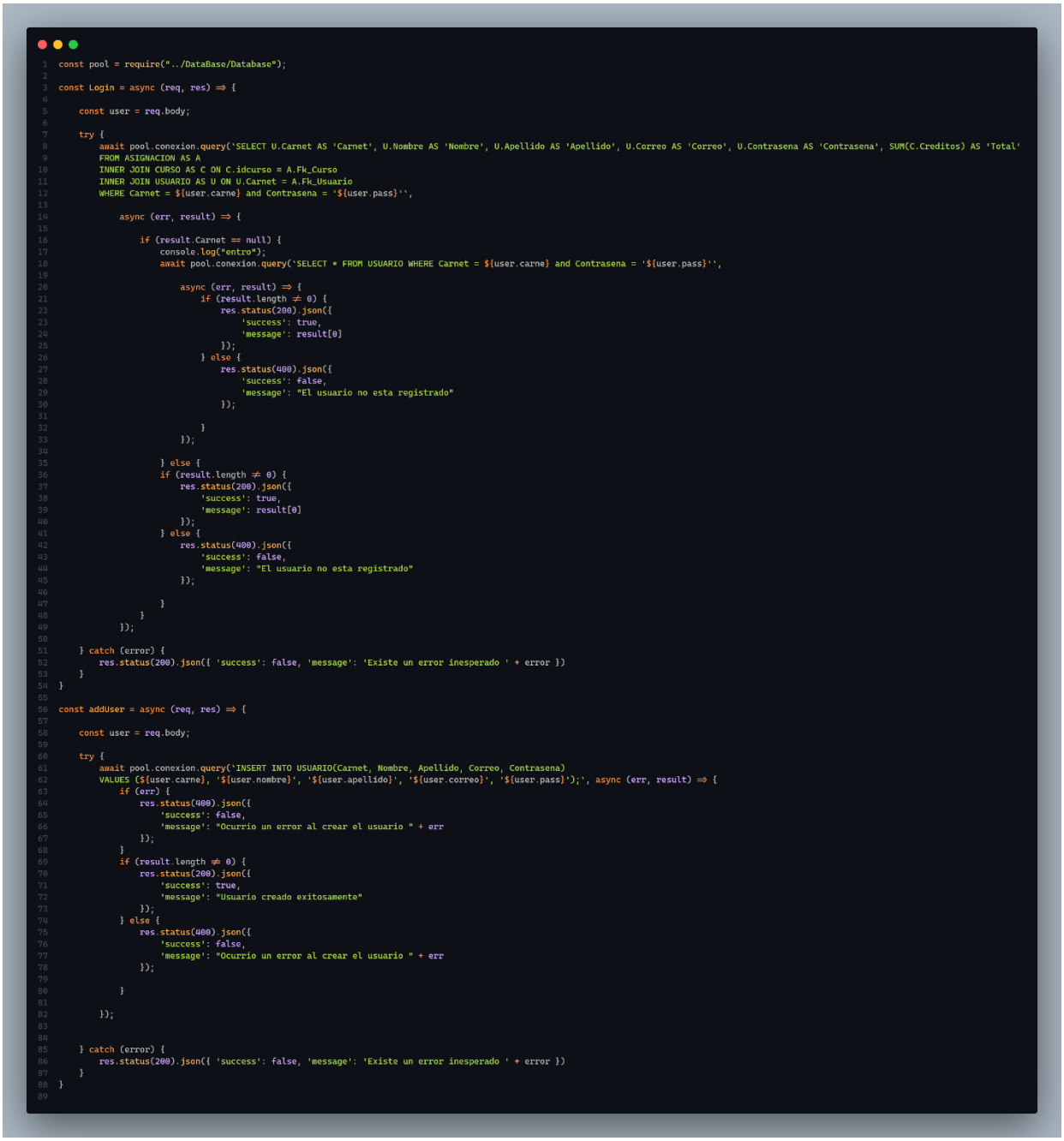
Este archivo contiene controladores relacionados con las operaciones de *publicación*, como *agregar una nueva publicación*, *listar todas las publicaciones* y *listar las publicaciones* de un usuario específico. También se incluye una función para *guardar catedráticos* en la base de datos cuando se realiza una publicación. Los controladores utilizan el módulo *mysql* para interactuar con la base de datos.

coursecontrollers.js

Este archivo contiene controladores relacionados con las operaciones de *cursos* y *catedráticos*, como *listar todos los cursos* y *listar todos los catedráticos*. Los controladores utilizan el módulo *mysql* para interactuar con la base de datos.

commentscontrollers.js

Este archivo contiene controladores relacionados con las operaciones de *comentarios*, como *agregar un comentario a una publicación* y *listar comentarios de una publicación específica*. Los controladores utilizan el módulo *mysql* para interactuar con la base de datos.



```
const pool = require("../DataBase/Database");

const Login = async (req, res) => {

    const user = req.body;

    try {
        await pool.conexion.query('SELECT U.Carnet AS 'Carnet', U.Nombre AS 'Nombre', U.Apellido AS 'Apellido', U.Correo AS 'Correo', U.Contrasena AS 'Contrasena', SUM(C.Creditos) AS 'Total'
        FROM ASIGNACION AS A
        INNER JOIN CURSO AS C ON C.idcurso = A.Fk_Curso
        INNER JOIN USUARIO AS U ON U.Carnet = A.Fk_Usuario
        WHERE Carnet = ${user.carne} and Contrasena = '${user.pass}'',

            async (err, result) => {

                if (result.Carnet == null) {
                    console.log("entro");
                    await pool.conexion.query('SELECT * FROM USUARIO WHERE Carnet = ${user.carne} and Contrasena = '${user.pass}'',

                        async (err, result) => {
                            if (result.length != 0) {
                                res.status(200).json({
                                    'success': true,
                                    'message': result[0]
                                });
                            } else {
                                res.status(400).json({
                                    'success': false,
                                    'message': "El usuario no esta registrado"
                                });

                            }
                        });

                } else {
                    if (result.length != 0) {
                        res.status(200).json({
                            'success': true,
                            'message': result[0]
                        });
                    } else {
                        res.status(400).json({
                            'success': false,
                            'message': "El usuario no esta registrado"
                        });

                    }
                }
            });

    } catch (error) {
        res.status(200).json({ 'success': false, 'message': 'Existe un error inesperado ' + error })
    }
}

const addUser = async (req, res) => {

    const user = req.body;

    try {
        await pool.conexion.query('INSERT INTO USUARIO(Carnet, Nombre, Apellido, Correo, Contrasena)
        VALUES (${user.carne}, '${user.nombre}', '${user.apellido}', '${user.correo}', '${user.pass}');', async (err, result) => {
            if (err) {
                res.status(400).json({
                    'success': false,
                    'message': "Ocurrio un error al crear el usuario " + err
                });
            }
            if (result.length != 0) {
                res.status(200).json({
                    'success': true,
                    'message': "Usuario creado exitosamente"
                });
            } else {
                res.status(400).json({
                    'success': false,
                    'message': "Ocurrio un error al crear el usuario " + err
                });

            }

        });


    } catch (error) {
        res.status(200).json({ 'success': false, 'message': 'Existe un error inesperado ' + error })
    }
}

```

Uso de la Aplicación

La aplicación proporciona una *API* para realizar operaciones de *registro*, *inicio de sesión*, *gestión de usuarios*, *gestión de publicaciones*, *gestión de cursos* y *gestión de comentarios*. Puede utilizar herramientas como *Postman* o aplicaciones *front-end* para interactuar con la *API*.

```sql
DROP DATABASE IF EXISTS Usuarios;
CREATE DATABASE Usuarios;
USE Usuarios;

-- Creación de la tabla 'USUARIO'
CREATE TABLE IF NOT EXISTS USUARIO (
    Carnet BIGINT PRIMARY KEY NOT NULL,
    Nombre VARCHAR(100),
    Apellido VARCHAR(100),
    Correo VARCHAR(50),
    Contrasena VARCHAR(50)
);

-- Creación de la tabla 'CATEDRATICO'
CREATE TABLE IF NOT EXISTS CATEDRATICO (
    idcatedratico INT AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(100)
);

-- Creación de la tabla 'CURSO'
CREATE TABLE IF NOT EXISTS CURSO (
    idcurso INT AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(100),
    Creditos INT
);

-- Creación de la tabla 'PUBLICACION'
CREATE TABLE IF NOT EXISTS PUBLICACION (
    idpublicacion INT AUTO_INCREMENT PRIMARY KEY,
    Descripcion VARCHAR(500),
    Fecha DATE,
    FK_Catedratico INT,
    Fk_Usuario BIGINT,
    Fk_Curso INT,
    FOREIGN KEY (Fk_Usuario) REFERENCES USUARIO(Carnet),
    FOREIGN KEY (FK_Catedratico) REFERENCES CATEDRATICO(idcatedratico),
    FOREIGN KEY (Fk_Curso) REFERENCES CURSO(idcurso)
);

-- Creación de la tabla 'COMENTARIO'
CREATE TABLE IF NOT EXISTS COMENTARIO (
    idcomentario INT AUTO_INCREMENT PRIMARY KEY,
    Fk_idpublicacion INT,
    Fk_Usuario BIGINT,
    Comentario VARCHAR(500),
    Fecha DATE,
    FOREIGN KEY (Fk_Usuario) REFERENCES USUARIO(Carnet),
    FOREIGN KEY (Fk_idpublicacion) REFERENCES PUBLICACION(idpublicacion)
);

-- Creación de la tabla 'ASIGNACION'
CREATE TABLE IF NOT EXISTS ASIGNACION (
    idasignacion INT AUTO_INCREMENT PRIMARY KEY,
    Fecha DATE,
    Fk_Usuario BIGINT,
    Fk_Curso INT,
    FOREIGN KEY (Fk_Usuario) REFERENCES USUARIO(Carnet),
    FOREIGN KEY (Fk_Curso) REFERENCES CURSO(idcurso)
);

-- Insertar datos de ejemplo
INSERT INTO USUARIO (Carnet, Nombre, Apellido, Correo, Contrasena) VALUES (20200045, 'LUIS', 'cubo de hielo', 'viejito@gmail.com', '123');
INSERT INTO CURSO (Nombre, Creditos) VALUES ('Logica de sistemas', 2);
INSERT INTO CURSO (Nombre, Creditos) VALUES ('Introduccion a la programacion y computacion 1', 4);
INSERT INTO CURSO (Nombre, Creditos) VALUES ('Lenguajes formales y de programacion', 3);
INSERT INTO CURSO (Nombre, Creditos) VALUES ('Introduccion a la programacion y computacion 2', 5);
INSERT INTO CURSO (Nombre, Creditos) VALUES ('Organizacion computacional', 3);
INSERT INTO CATEDRATICO (Nombre) VALUES ('Pepito');
INSERT INTO PUBLICACION (Descripcion, Fecha, FK_Catedratico, Fk_Usuario) VALUES ('Prueba 1', CURDATE(), 1, 20200045);

```