Universidad de San Carlos de Guatemala
Guatemala
Facultad de Ingeniería en
Ciencias y Sistemas
Prácticas iniciales
Ing. Igor Veliz
Sección F-

Grupo No. 7

Manual Técnico - Sistema de evaluación USAC

Integrantes:

No.	Carnet	Nombre	No. Celular
1	202402587	Angel Raúl Herrera Chilel	56214136
2	202402473	Kimberly Samantha Gómez Chávez	58330770
3	202106435	María Hercilia Flores Alvarez	47694272
4	202200013	Marcos Aarón Toledo Alvarez	45150096
5	202403929	Emiliana Elizabeth Pú Lara	41948533

Tutores:

No.	Nombre	No. Celular
1	Julio Ruano	58925434
2	Gabriel Melgar	43314235

ÍNDICE

MANUAL TÉCNICO - SISTEMA DE EVALUACIÓN USAC

1. Introducción

Este documento describe la arquitectura, componentes y funcionalidades del servidor desarrollado para la aplicación web en versión de prueba de reseñas de catedráticos y cursos de la Facultad de Ingeniería de la USAC. El backend está construido con Node.js y ofrece una API REST que sirve como interfaz entre el frontend (cliente) y la base de datos MySQL.

2. Aplicación web

Es un programa que funciona dentro del navegador, pero que te permite interactuar y hacer tareas complejas.
Se ejecuta en un servidor y el usuario solo necesita un navegador para usarla.

- Permite interacción: no solo lees información, también puedes escribir, modificar, enviar datos y recibir resultados.
- Puede conectarse a bases de datos para guardar y procesar información.
- Se actualiza automáticamente en línea (el usuario no tiene que descargar actualizaciones).

3. Proyecto

El proyecto es una aplicación web en versión de prueba, "Sistema de Evaluación de Catedráticos y Cursos de Ingeniería USAC". Su objetivo es que los estudiantes puedan:

- Publicar opiniones sobre catedráticos y cursos.
- Comentar publicaciones de otros.
- Buscar y filtrar publicaciones.
- Gestionar su perfil y cursos aprobados.

4. Arquitectura del Backend

El backend sigue una arquitectura por capas MVC (Modelo-Vista-Controlador) adaptada:

- Routes: Manejan las solicitudes HTTP y envían respuestas.
- Services: Contienen la lógica de negocio y reglas de aplicación.
- Models: Definen la estructura de datos y interactúan con la base de datos.
- Middleware: Funciones intermedias para autenticación y validación.
- Config: Configuración de la base de datos y otros servicios.
- Utils: Utilidades generales y helpers.

5. Stack Tecnologico

Tecnología	Versión	Propósito
Node.js	18+	Entorno de ejecución JavaScript
Express.js	4.18.2+	Framework web para API REST
MySQL2	3.6.5+	Controlador para MySQL
Sequelize	6.35.1+	ORM para base de datos
bcryptjs	3.0.2+	Encriptación de contraseñas
CORS	2.8.5+	Habilitar peticiones cruzadas

6. Estructura de Carpetas del Backend

```
Backend/
├── src/
│   ├── config/
│   │   ├── database.js      # Configuración de Sequelize
│   │   └── seed.js          # Datos iniciales para la BD
│   ├── middleware/
│   │   └── auth.js          # Middleware de autenticación JWT
│   ├── models/              # Modelos de datos (tablas)
│   │   ├── usuario.js       # Modelo de usuarios
│   │   ├── rol.js           # Modelo de roles
│   │   └── ...              # Otros modelos
│   ├── routes/              # Rutas de la API
│   │   ├── usuarios.routes.js
│   │   ├── asignacion.routes.js
│   │   └── ...
│   ├── service/             # Lógica
│   │   ├── usuarioService.js
│   │   ├── asignacionService.js
│   │   ├── filtroService.js
│   │   └── ...
│   └── utils/
│       └── asyncWrapper.js # Manejo de errores async
├── package.json
```

7. Funcionalidades del Servidor

7.1 Usuario Administrador

- Gestionar usuarios: Crear, modificar y eliminar usuarios mediante usuarioService.js
- Administrar tablas: Gestión completa de cursos, catedráticos y asignaciones
- Acceso a estadísticas generales: Mediante servicios específicos

7.2 Usuario Estudiante

- Publicar en el foro: mediante `publicacionService.js` y `comentarioService.js`.

- Modificar su perfil: mediante perfilService.js.
- Gestionar cursos aprobados: mediante cursoAprobadoService.js.

7.3 Usuario Catedrático

- Solo lectura en el foro: Acceso limitado a publicaciones
- Ver estadísticas personales: Número de alumnos y cursos impartidos
- Acceso restringido: Solo a endpoints específicos con autenticación

8. Endpoints Principales de la API

Método	Endpoint	Descripción	Acceso
POST	/api/auth/login	Inicio de sesión	Todos
POST	/api/auth/register	Registro de usuario	Público
GET	/api/publicaciones	Obtener publicaciones	Autenticados
POST	/api/publicaciones	Crear publicación	Estudiantes
GET	/api/catedraticos	Listar catedráticos	Todos
GET	/api/estadisticas	Ver estadísticas	Admin y Catedráticos
PUT	/api/perfil	Actualizar perfil	Propietario

9. Modelo Datos Principales (Modelo usuario)

```
3     import {Rol} from './rol.js';
4
5     export const Usuario = sequelize.define('Usuario', {
6         id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
7         carnet: { type: DataTypes.STRING(9), unique: true, allowNull: false },
8         nombre: { type: DataTypes.STRING, allowNull: false },
9         correo: { type: DataTypes.STRING, unique: true, allowNull: false },
10        password: { type: DataTypes.STRING, allowNull: false },
11        rolId: { type: DataTypes.INTEGER, allowNull: false }
12    }, { tableName: 'usuarios', timestamps: false });
13
14    Usuario.belongsTo(Rol, { foreignKey: 'rolId', as: 'rol' });
```

10. Servicios Principales

UsuarioService (service/usuarioService.js)

- registrar(dto): Crea nuevo usuario con validación de duplicados
- autenticar(correo, password): Verifica credenciales y retorna usuario
- restablecer(registro, correo, nuevaPass): Cambia contraseña con validación
- porId(id): Obtiene usuario por ID
- esEstudiante(id): Verifica si usuario es estudiante
- esCatedratico(id): Verifica si usuario es catedrático

FiltroService (service/filtroService.js)

- usuarios(query):Busca usuarios con filtros personalizados

AsignacionService (service/asignacionService.js)

- estadisticasCatedratico(id): Obtiene estadísticas para catedrático específico

11. Middleware de Autenticación y Autorización

```
import {UsuarioService} from '../services/usuarioService.js';

export function auth(rolPermitido) {
  return async (req, res, next) => {
    const id = req.headers['usuario-id'];
    if (!id) return res.status(401).json({ error: 'Falta usuario-id header' });
    const user = await UsuarioService.porId(id);
    if (!user) return res.status(404).json({ error: 'Usuario no existe' });
    if (rolPermitido && user.rol !== rolPermitido) return res.status(403).json({ error: 'Sin permisos' });
    req.usuario = user;
    next();
  };
}
```

12. Flujos de Autenticación

Registro de Usuario

1. Cliente envía POST a `/api/usuarios/registro` con datos
2. Servicio valida que no exista duplicado (carnet o correo)
3. Encripta password con bcryptjs
4. Crea usuario en base de datos
5. Retorna usuario creado

Inicio de Sesión

1. Cliente envía POST a `/api/usuarios/login` con credenciales
2. Servicio busca usuario por correo
3. Compara password con bcrypt.compareSync()
4. Retorna usuario autenticado

Autorización por Roles

1. Cliente incluye header `usuario-id` en peticiones
2. Middleware auth verifica existencia y permisos
3. Si tiene permisos, agrega usuario a `req.usuari`

13. Consideraciones de Seguridad

- Contraseñas encriptadas con bcryptjs (salt rounds: 10)
- Validación de duplicados en registro
- Autorización por roles en endpoints sensibles
- Validación de parámetros en servicios
- Manejo centralizado de errores

14. Base de Datos

En esta parte se describe el diseño, la implementación y la configuración de la base de datos utilizada en la aplicación web desarrollada en el curso. La base de datos fue creada en MySQL y gestionada en el proyecto mediante Sequelize como ORM en Node.js.

-MySQL

MySQL es un sistema de gestión de bases de datos relacionales (RDBMS) que permite crear, modificar y gestionar bases de datos. Utiliza el lenguaje de consulta estructurado (SQL) para interactuar con la base de datos. MySQL es una opción popular para aplicaciones web debido a su facilidad de uso, rendimiento y escalabilidad.

-XAMPP

XAMPP es un paquete de software que incluye Apache, MySQL, PHP y Perl, diseñado para proporcionar un entorno de desarrollo web integral. XAMPP permite crear un servidor web local en tu máquina, lo que facilita el desarrollo y prueba de aplicaciones web sin necesidad de una conexión a Internet.

-phpMyAdmin

phpMyAdmin es una herramienta de administración de bases de datos MySQL basada en web. Proporciona una interfaz gráfica de usuario (GUI) para interactuar con la base de datos, lo que facilita la creación, modificación y gestión de bases de datos MySQL. phpMyAdmin es una herramienta esencial para cualquier desarrollador web que trabaje con MySQL.

-Tablas relacionales

Las tablas relacionales son la base de datos relacionales. Una tabla relacional es una estructura de datos que consta de filas (registros) y columnas (campos). Cada fila representa un registro único, y cada columna representa un campo o atributo de ese registro. Las tablas relacionales se utilizan para almacenar datos relacionados entre sí, y se pueden vincular mediante claves primarias y foráneas para establecer relaciones entre ellas.

Ventajas al usar una Base de Datos local

-Control total: Al tener una base de datos local, se tiene el control total sobre la infraestructura y la configuración de la base de datos.

-Seguridad: Se puede implementar medidas de seguridad personalizadas y tener un mayor control sobre el acceso a la base de datos.

-Rendimiento: Las bases de datos locales pueden ofrecer un mejor rendimiento, ya que no dependen de la conexión a Internet y no hay latencia asociada con la comunicación con un servidor remoto.

-Privacidad: Los datos se almacenan en un servidor local, lo que puede ser beneficioso para empresas que manejan información sensible o confidencial.

-Costo: En algunos casos, mantener una base de datos local puede ser más rentable que pagar por un servicio de base de datos en la nube, especialmente para pequeñas empresas o proyectos con un tráfico bajo.

-Personalización: Se puede personalizar la configuración de la base de datos y el hardware según las necesidades específicas.

-No dependencia de Internet: Se puede acceder a la base de datos sin necesidad de una conexión a Internet.

Script de creación de la base de datos

El siguiente script SQL define todas las tablas necesarias para el funcionamiento de la aplicación:

```
-- ========================================

-- Tabla de usuarios

-- ========================================

CREATE TABLE usuario (

    id INT AUTO_INCREMENT PRIMARY KEY,

    carne VARCHAR(9) UNIQUE NOT NULL,

    nombres VARCHAR(60) NOT NULL,

    apellidos VARCHAR(60) NOT NULL,

    correo VARCHAR(60) NOT NULL,

    hash CHAR(64) NOT NULL, -- contraseña en hash

    rol ENUM('EST','CAT') DEFAULT 'EST' NOT NULL

);


-- ========================================

-- Tabla de cursos

-- ========================================

CREATE TABLE curso (

    id INT AUTO_INCREMENT PRIMARY KEY,

    codigo VARCHAR(10) UNIQUE NOT NULL,
```

```
    nombre VARCHAR(80) NOT NULL,
    creditos TINYINT NOT NULL,
    tieneLab TINYINT(1) DEFAULT 0 -- 0 = falso, 1 = verdadero
);

-- ========================================
-- Tabla de secciones
-- ========================================
CREATE TABLE seccion (
    id INT AUTO_INCREMENT PRIMARY KEY,
    idCurso INT NOT NULL,
    idCatedratico INT NOT NULL,
    codigoSeccion VARCHAR(10) NOT NULL,
    cupo INT NOT NULL,
    FOREIGN KEY (idCurso) REFERENCES curso(id) ON DELETE CASCADE,
    FOREIGN KEY (idCatedratico) REFERENCES usuario(id) ON DELETE CASCADE
);

-- ========================================
-- Tabla de publicaciones
-- ========================================
CREATE TABLE publicacion (
```

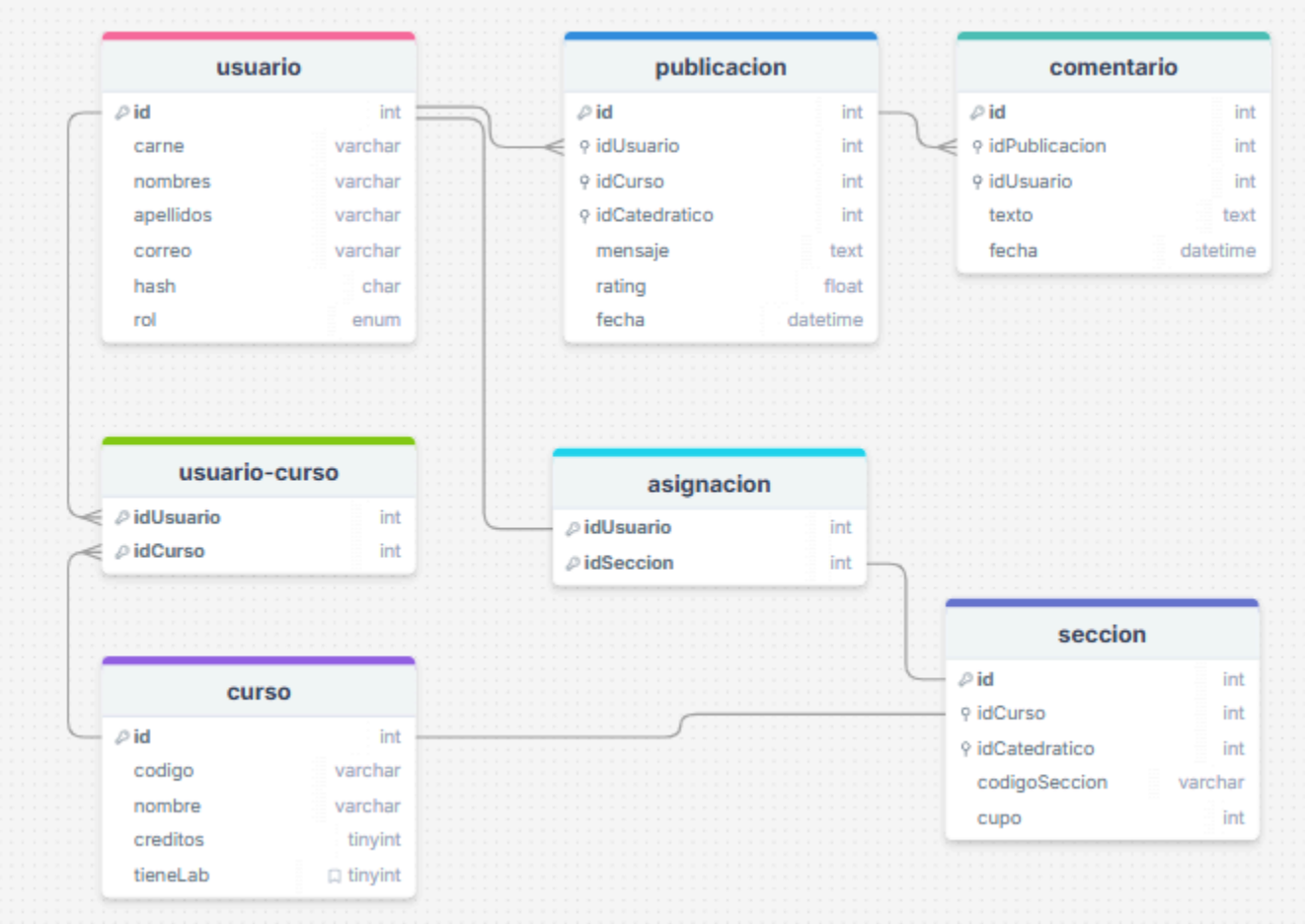
```
    id INT AUTO_INCREMENT PRIMARY KEY,
    idUsuario INT NOT NULL,
    idCurso INT,
    idCatedratico INT,
    mensaje TEXT NOT NULL,
    rating FLOAT,
    fecha DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (idUsuario) REFERENCES usuario(id) ON DELETE CASCADE,
    FOREIGN KEY (idCurso) REFERENCES curso(id) ON DELETE SET NULL,
    FOREIGN KEY (idCatedratico) REFERENCES usuario(id) ON DELETE SET NULL,
    CHECK (rating IS NULL OR (rating >= 1 AND rating <= 5))
);

-- ========================================
-- Tabla de comentarios
-- ========================================
CREATE TABLE comentario (
    id INT AUTO_INCREMENT PRIMARY KEY,
    idPublicacion INT NOT NULL,
    idUsuario INT NOT NULL,
    texto TEXT NOT NULL,
    fecha DATETIME DEFAULT CURRENT_TIMESTAMP,
```

```
    FOREIGN KEY (idPublicacion) REFERENCES publicacion(id) ON DELETE CASCADE,
    FOREIGN KEY (idUsuario) REFERENCES usuario(id) ON DELETE CASCADE
);

-- ========================================
-- Tabla de asignaciones (muchos a muchos usuario-seccion)
-- ========================================
CREATE TABLE asignacion (
    idUsuario INT NOT NULL,
    idSeccion INT NOT NULL,
    PRIMARY KEY (idUsuario, idSeccion),
    FOREIGN KEY (idUsuario) REFERENCES usuario(id) ON DELETE CASCADE,
    FOREIGN KEY (idSeccion) REFERENCES seccion(id) ON DELETE CASCADE
);
```

Relación entre tablas



Modelo Entidad-Relación (MER)

La base de datos está compuesta por las siguientes entidades principales:

- **Usuario**: representa a los estudiantes y catedráticos.
- **Curso**: cursos disponibles en el área de sistemas.
- **Sección**: secciones específicas de un curso, asignadas a un catedrático.
- **Publicación**: publicaciones realizadas por los usuarios sobre cursos o catedráticos.
- **Comentario**: comentarios dentro de las publicaciones.
- **Asignación**: relación de usuarios con secciones de cursos.

Diccionario de datos

Tabla: Rol
id: BIGINT, PK, autoincremental
nombre: ENUM('estudiante','catedratico','admin'), único, no nulo, define el rol del usuario

Tabla: Usuario
id: INT, PK, autoincremental
carnet: VARCHAR(9), único, no nulo
nombre: VARCHAR, no nulo
correo: VARCHAR, único, no nulo
password: VARCHAR, no nulo, se guarda en hash
rolId: INT, FK → Rol(id), no nulo

Tabla: Curso
id: INT, PK, autoincremental
codigo: VARCHAR(20), único, no nulo
nombre: VARCHAR, no nulo
creditos: INT, opcional

Tabla: Seccion
id: INT, PK, autoincremental
cursoId: INT, FK → Curso(id), no nulo
numero: INT, no nulo, define el número de sección
laboratorio: BOOLEAN, valor por defecto FALSE
catedraticoId: INT, FK → Usuario(id), puede ser nulo
Restricción única: combinación (cursoId, numero) debe ser única

Tabla: Publicacion
id: BIGINT, PK, autoincremental
autorId: BIGINT, FK → Usuario(id), no nulo
cursoId: BIGINT, FK → Curso(id), no nulo
contenido: TEXT, texto de la publicación
likes: BIGINT, valor por defecto 0
fecha: DATE, valor por defecto fecha actual

Tabla: Comentario
id: BIGINT, PK, autoincremental

publicacionId: BIGINT, FK → Publicacion(id), no nulo
autorId: BIGINT, FK → Usuario(id), no nulo
contenido: TEXT, texto del comentario
fecha: DATE, valor por defecto fecha actual

Tabla: UsuarioCurso
usuarioId: BIGINT, PK, FK → Usuario(id)
cursoId: BIGINT, PK, FK → Curso(id)
status: ENUM('activo','finalizado','reprobado'), valor por defecto 'activo'

Tabla: Asignacion
id: INT, PK, autoincremental
estudianteId: INT, FK → Usuario(id), no nulo
cursoId: INT, FK → Curso(id), no nulo
seccion: INT, número de sección, opcional
laboratorio: BOOLEAN, valor por defecto FALSE
vecesRepitiendo: INT, valor por defecto 0

Conexión con Sequelize

El proyecto utiliza Sequelize como ORM para abstraer la base de datos. La configuración se encuentra en database.js:

```
import {Sequelize} from 'sequelize';
import dotenv from 'dotenv';
dotenv.config();

export const sequelize = new Sequelize(
  process.env.MYSQL_DB,
  process.env.MYSQL_USER,
  process.env.MYSQL_PASS,
  {
    host: process.env.MYSQL_HOST,
    port: process.env.MYSQL_PORT,
    dialect: 'mysql',
    logging: false
  }
);
```

Modelos en Sequelize

Cada tabla está representada con un modelo.

Usuario

```
import {DataTypes} from 'sequelize';
import {sequelize} from '../config/database.js';
import {Rol} from './rol.js';

export const Usuario = sequelize.define('Usuario', {
    id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
    carnet: { type: DataTypes.STRING(9), unique: true, allowNull: false },
    nombre: { type: DataTypes.STRING, allowNull: false },
    correo: { type: DataTypes.STRING, unique: true, allowNull: false },
    password: { type: DataTypes.STRING, allowNull: false },
    rolId: { type: DataTypes.INTEGER, allowNull: false }
}, { tableName: 'usuarios', timestamps: false });

Usuario.belongsTo(Rol, { foreignKey: 'rolId', as: 'rol' });
```

UsuarioCurso

```
export const UsuarioCurso = sequelize.define('UsuarioCurso', {
    usuarioId: { type: DataTypes.BIGINT, primaryKey: true },
    cursoId: { type: DataTypes.BIGINT, primaryKey: true },
    status: { type: DataTypes.ENUM('activo', 'finalizado', 'reprobado'), defaultValue: 'activo' }
}, { tableName: 'usuario_curso', timestamps: false });
```

Sección

```
import {DataTypes} from 'sequelize';
import {sequelize} from '../config/database.js';
import {Curso} from './curso.js';
import {Usuario} from './usuario.js';

export const Seccion = sequelize.define('Seccion', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  cursoId: { type: DataTypes.INTEGER, allowNull: false },
  numero: { type: DataTypes.INTEGER, allowNull: false },
  laboratorio: { type: DataTypes.BOOLEAN, defaultValue: false },
  catedraticoId: { type: DataTypes.INTEGER }
}, { tableName: 'secciones', timestamps: false });

Seccion.uniqueKeys = { curso_seccion_unique: { fields: ['cursoId', 'numero'] } };

Seccion.belongsTo(Curso, { foreignKey: 'cursoId' });
Seccion.belongsTo(Usuario, { foreignKey: 'catedraticoId', as: 'catedratico' });
```

Rol

```
import {DataTypes} from 'sequelize';
import {sequelize} from '../config/database.js';

export const Rol = sequelize.define('Rol', {
    id: { type: DataTypes.BIGINT, primaryKey: true, autoIncrement: true },
    nombre: { type: DataTypes.ENUM('estudiante', 'catedratico', 'admin'), unique: true }
}, {tableName: 'roles', timestamps: false});
```

Publicación

```
export const Publicacion = sequelize.define('Publicacion', {
    id: { type: DataTypes.BIGINT, primaryKey: true, autoIncrement: true },
    autorId: { type: DataTypes.BIGINT, allowNull: false },
    cursoId: { type: DataTypes.BIGINT, allowNull: false },
    contenido: { type: DataTypes.TEXT },
    likes: { type: DataTypes.BIGINT, defaultValue: 0 },
    fecha: { type: DataTypes.DATEONLY, defaultValue: DataTypes.NOW }
}, { tableName: 'publicaciones', timestamps: false });
```

Curso

```
import {DataTypes} from 'sequelize';
import {sequelize} from '../config/database.js';

export const Curso = sequelize.define('Curso', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  codigo: { type: DataTypes.STRING(20), unique: true, allowNull: false },
  nombre: { type: DataTypes.STRING, allowNull: false },
  creditos: { type: DataTypes.INTEGER }
}, { tableName: 'cursos', timestamps: false });
```

Comentario

```
export const Comentario = sequelize.define('Comentario', {
    id: { type: DataTypes.BIGINT, primaryKey: true, autoIncrement: true },
    publicacionId: { type: DataTypes.BIGINT, allowNull: false },
    autorId: { type: DataTypes.BIGINT, allowNull: false },
    contenido: { type: DataTypes.TEXT },
    fecha: { type: DataTypes.DATEONLY, defaultValue: DataTypes.NOW }
}, { tableName: 'comentarios', timestamps: false });
```

Asignación

```
import {DataTypes} from 'sequelize';
import {sequelize} from '../config/database.js';
import {Usuario} from './usuario.js';
import {Curso} from './curso.js';

export const Asignacion = sequelize.define('Asignacion', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  estudianteId: { type: DataTypes.INTEGER, allowNull: false },
  cursoId: { type: DataTypes.INTEGER, allowNull: false },
  seccion: { type: DataTypes.INTEGER },
  laboratorio: { type: DataTypes.BOOLEAN, defaultValue: false },
  vecesRepitiendo: { type: DataTypes.INTEGER, defaultValue: 0 }
}, { tableName: 'asignaciones', timestamps: false });

Asignacion.belongsTo(Usuario, { foreignKey: 'estudianteId', as: 'estudiante' });
Asignacion.belongsTo(Curso, { foreignKey: 'cursoId' });
```

Relaciones entre modelos

- Un Usuario pertenece a un Rol.
- Un Curso tiene muchas Secciones.
- Una Sección pertenece a un Catedrático (Usuario).
- Un Usuario puede realizar muchas Publicaciones y Comentarios.
- Una Publicación puede tener muchos Comentarios.
- UsuarioCurso representa la relación muchos a muchos entre Usuarios y Cursos.
- Asignación vincula a estudiantes con secciones.

Datos iniciales (Seeds)

En el archivo seed.js se cargan los roles por defecto:

```
import {sequelize} from './database.js';
import {Rol} from '../models/rol.js';

await sequelize.sync();
await Rol.bulkCreate(
  [{ nombre: 'estudiante' }, { nombre: 'catedratico' }, { nombre: 'admin' }],
  { ignoreDuplicates: true }
);
```

15. Fuentes de documentación

Para el desarrollo y mantenimiento de este proyecto, se recomienda consultar las siguientes fuentes oficiales de documentación:

Node.js

- Documentación oficial: https://nodejs.org/en/docs/
- Guías y referencias de API: https://nodejs.org/api/

Express.js

- Documentación oficial: https://expressjs.com/
- Referencia de API: https://expressjs.com/en/4x/api.html

Sequelize ORM

- Documentación oficial: https://sequelize.org/

MySQL

- Documentación oficial: https://dev.mysql.com/doc/
- Referencia de lenguaje: https://dev.mysql.com/doc/refman/8.0/en/
- Introducción general: https://www.oracle.com/latam/mysql/what-is-mysql/
- Tutorial introductorio: https://www.hostinger.com/es/tutoriales/que-es-mysql

bcryptjs

- Documentación: https://www.npmjs.com/package/bcryptjs

CORS

- Documentación: https://www.npmjs.com/package/cors

dotenv

- Documentación: https://www.npmjs.com/package/dotenv
- Uso de variables de entorno: https://github.com/motdotla/dotenv

XAMPP

- Guía introductoria: https://draftdesignweb.com/2023/12/25/xampp-una-solucion-integral-para-desarrolladores/

Bases de datos locales

- Artículo explicativo: https://jhonmosquera.com/bases-de-datos-locales/

Arquitectura de datos

- Comparación On-premises vs Cloud: https://www.teradata.com/insights/data-architecture/on-premises-vs-cloud

16 Foto grupal