

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Segundo Semestre 2023
Prácticas Iniciales
Sección: “ F- ”



Manual Técnico

Grupo No.9

Nombre de los estudiantes:

María Patricia Serrano Ramírez	202201989
Valery Nicolle Gálvez García	202200141
Ariel Josué López Gálvez	202200185
Sergio Joel Rodas Valdez	202200271

Nombre de los tutores:

Justin Josué Aguirre Román	202004734
Ronaldo Javier Posadas Guerra	202004770

Guatemala, 21 de septiembre de 2023

Introducción

Este manual técnico proporciona una guía detallada y completa para entender la estructura, el funcionamiento y la implementación de nuestra aplicación. Está diseñado para desarrolladores y cualquier persona interesada en comprender en profundidad cómo funciona nuestra aplicación, desde la configuración del *servidor* hasta la gestión de datos en la *base de datos*.

En este manual, se describen en detalle todos los componentes clave de la aplicación, incluyendo el servidor *Express*, las *rutas*, los *controladores*, la autenticación de usuarios con *tokens JWT* y la conexión a la base de datos *MySQL*. Además, se proporciona información sobre la estructura de la base de datos y los procedimientos almacenados utilizados para gestionar los datos.

La aplicación que se describe en este manual está diseñada para permitir a los usuarios registrar *cuentas*, *crear publicaciones*, *agregar comentarios* y *gestionar cursos*. Para garantizar la seguridad de la aplicación, se ha implementado un sistema de autenticación de usuarios y se han desarrollado funciones para gestionar la información de los usuarios de manera segura.

Este manual está diseñado para ser una herramienta útil tanto para aquellos que están familiarizados con el desarrollo web como para aquellos que desean aprender sobre los conceptos y la implementación detrás de una aplicación web moderna.

Manual Técnico del Servidor

Archivo index.js

El archivo *index.js* es el punto de entrada de la aplicación y se encarga de configurar y ejecutar el servidor.

Variables:

- *port*: Define el puerto en el que se ejecutará el servidor.
- *app*: Importa el módulo de la aplicación desde el archivo *app.js*.

Funciones:

app.listen(port, callback): Inicia el servidor en el puerto especificado y muestra un mensaje en la consola cuando el servidor está corriendo.

```
'use strict'

var port = 3800; // Puerto de la aplicación
var app = require('./app');

app.listen(port, ()=>{
  console.log('Servidor corriendo con express', port);
});
```

Figura 1. *index.js*

Archivo app.js

El archivo *app.js* es el módulo principal de la aplicación y configura la instancia de *Express.js*.

Librerías y Rutas:

- Importa las librerías *Express* y *Body Parser*.
- Define una instancia de *Express* llamada *app*.
- Importa las rutas desde los archivos correspondientes (*users-routes*, *publication-routes*, *comment-routes*, *course-routes*).
- Configura *Body Parser* para procesar las solicitudes JSON.

Middleware CORS

Configura un *middleware* para manejar los encabezados CORS y permitir las solicitudes desde cualquier origen.

Rutas

Asocia las rutas importadas a rutas específicas en la aplicación, como: */users*, */publication*, */comment* y */course*.

```
// Import Libraries and routes
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
var userRoutes = require('./routes/users-routes');
var publicationRoutes = require('./routes/publication-routes');
var commentRoutes = require('./routes/comment-routes');
var courseRoutes = require('./routes/course-routes');

app.use(bodyParser.urlencoded({extended: false}));
app.use(bodyParser.json());

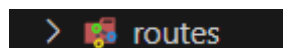
//Declaration of CORS and Headers
app.use((req, res, next) => {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', 'Authorization, X-API-KEY, Origin, X-Requested-With, Content-Type,');
  res.header('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, DELETE');
  res.header('Allow', 'GET, POST, OPTIONS, PUT, DELETE');
  next();
});

//Routes
app.use('/users', userRoutes);
app.use('/publication', publicationRoutes);
app.use('/comment', commentRoutes);
app.use('/course', courseRoutes);
module.exports = app;
```

Figura 2. *app.js*

Carpeta Routes

La carpeta Routes contiene archivos que definen las rutas y las funciones controladoras para diferentes partes de la aplicación.



Archivo users-routes.js

Define un router de *Express* llamado *api*. Importa el controlador de usuarios (*user-controller*) y el middleware de autenticación (*mdAuth*).

Define las siguientes rutas:

- *POST /registerUser*: Registrar un usuario.

- *GET /findOneUser*: Buscar un usuario.
- *GET /findAllUsers*: Listar todos los usuarios.
- *PUT /updatePassword*: Modificar la contraseña del usuario.
- *POST /login*: Iniciar sesión.
- *GET /view_profile_users/:carnet*: Ver perfil de usuario (requiere autenticación de administrador).

Archivo **publication-routes.js**

Define un router de Express llamado *api*. Importa el controlador de publicaciones (*publication-controller*) y el middleware de autenticación (*mdAuth*).

Define las siguientes rutas:

- *POST /addPublication*: Crear una publicación (requiere autenticación de administrador).
- *DELETE /deletePublication/:id*: Eliminar una publicación (requiere autenticación de administrador).
- *PUT /updatePublication/:id*: Actualizar una publicación (requiere autenticación de administrador).
- *GET /getPublications*: Obtener todas las publicaciones (requiere autenticación de administrador).
- *GET /getPublicationsFilterByCurso*: Obtener publicaciones filtradas por curso (requiere autenticación de administrador).
- *GET /getPublicationsFilterByCatedratico*: Obtener publicaciones filtradas por catedrático (requiere autenticación de administrador).
- *GET /getPublicationsFilterByNameCurso*: Obtener publicaciones filtradas por nombre de curso (requiere autenticación de administrador).
- *GET /getPublicationsFilterByNameCatedratico*: Obtener publicaciones filtradas por nombre de catedrático (requiere autenticación de administrador).

Archivo **comment-routes.js**

Define un router de *Express* llamado *api*. Importa el controlador de comentarios (*comment-controller*) y el middleware de autenticación (*mdAuth*).

Define las siguientes rutas:

- *GET /getComments/:id*: Obtener todos los comentarios de una publicación (requiere autenticación de administrador).
- *GET /get_Onecoment/:id*: Obtener un comentario (requiere autenticación de administrador).
- *POST /addComment/:id*: Añadir un comentario (requiere autenticación de administrador).
- *DELETE /deleteComment/:id*: Eliminar un comentario (requiere autenticación de administrador).

- *PUT /updateComment/:id*: Actualizar un comentario (requiere autenticación de administrador).

Archivo `course-routes.js`

Define un router de *Express* llamado *api*. Importa el controlador de cursos (*course-controller*) y el middleware de autenticación (*mdAuth*).

Define las siguientes rutas:

- *GET /getCourse/:code*: Obtener cursos aprobados por un código de usuario.
- *POST /addCourse/:code*: Agregar un curso (requiere autenticación de administrador).
- *DELETE /deleteCourse/:id*: Eliminar un curso (requiere autenticación de administrador).


```
// Import libraries and controller
var express = require('express');
const app = require('../app');
var userController = require('../controllers/user-controller');
var api = express.Router();
var mdAuth = require('../middlewares/authenticated');

// Routes and function to use for queries
api.post('/registerUser', userController.registerUser); // Registrar un usuario
api.get('/findOneUser', userController.findOneUser); // Buscar un usuario
api.get('/findAllUsers', userController.findUsers); // Listar todos los usuarios
api.put('/updatePassword', userController.updatePasswordUser); // Modificar contraseña
api.post('/login', userController.login); // Login
api.get('/view_profile_users/:carnet', mdAuth.ensureAuthAdmin, userController.view_profile_users);
// Export variable
module.exports = api;
```

Figura 3. *users-routes.js*

Carpeta controllers

La carpeta *controllers* contiene archivos que definen las funciones controladoras para las rutas de la aplicación.

>  controllers

Archivo `user-controller.js`

- Define funciones controladoras para las operaciones relacionadas con usuarios, como *registro*, *búsqueda*, *listado*, *modificación de contraseña*, *inicio de sesión* y *actualización de perfil*.

Archivo publication-controller.js

Define funciones controladoras para las operaciones relacionadas con publicaciones, como *creación*, *actualización*, *eliminación* y *obtención de publicaciones*. También incluye funciones para filtrar publicaciones por *curso*, *catedrático* y *nombre de curso* o *catedrático*.

Archivo comment-controller.js

Define funciones controladoras para las operaciones relacionadas con comentarios, como *creación*, *actualización*, *eliminación* y *obtención de comentarios*. También incluye funciones para obtener *un solo comentario* y *todos los comentarios* de una publicación.

Archivo course-controller.js


Define funciones controladoras para las operaciones relacionadas con cursos, como *obtener cursos aprobados por un código de usuario*, *agregar un curso* y *eliminar un curso*.

```
function registerUser(req, res){
  ... var {registroAcademico, nameU, lastnameU, email, passwordU} = req.body
  ... mysqlFunctions.query('call sp_registerUser(?,?,?,?)', [registroAcademico, nameU, lastnameU, email, passwordU], (err, rows, fields)=>{
  ... if(err){
  ... res.status(500).send({message: 'Error general en el servidor', err});
  ... }else if(rows){
  ... res.send({user: rows});
  ... }else{
  ... res.status(402).send({message: 'No se pudo realizar la peticion'});
  ... }
  ... });
}
```

Figura 4. user-controller.js

Carpeta middlewares

La carpeta *middlewares* contiene archivos que definen *middlewares*, específicamente, el *middleware de autenticación*.

>  middlewares

Archivo authenticated.js

Define un *middleware de autenticación* que verifica la presencia de un *token de autenticación* en las solicitudes entrantes y valida su vigencia. Si el *token* es válido,

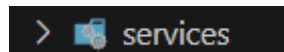
el *middleware* agrega los datos del usuario autenticado a la solicitud para su posterior procesamiento.

```
exports.ensureAuthAdmin = (req, res, next)=>{
  ....if(!req.headers.authorization){
  ....  return res.status(403).send({message: 'Petición sin autenticación'});
  ....}else{
  ....  var token = req.headers.authorization.replace(/["]+/g, '');
  ....  try{
  ....    var payload = jwt.decode(token, key);
  ....    if(payload.exp <= moment().unix()){
  ....      return res.status(401).send({message: 'Token expirado'});
  ....    }
  ....  }catch(ex){
  ....    return res.status(404).send({message: 'Token no válido'});
  ....  }
  ....
  ....  req.user = payload;
  ....  next();
  ....}
}
```

Figura 5. Autenticación de token

Carpeta services

La carpeta *services* contiene archivos que definen funciones de servicios, en este caso, la creación y codificación de *tokens JWT*.



Archivo jwt.js

Define una función para crear *tokens JWT* que contienen información del usuario autenticado, como su *código de registro académico*, *nombre*, *apellidos* y *correo electrónico*. Los *tokens* tienen una fecha de expiración de 30 días.

```
'use strict'

var jwt = require('jwt-simple');
var moment = require('moment');
var key = 'admin123';

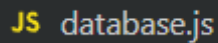
exports.createToken = (users)=>{
  ....var payload = {
  ....  sub: users.registroAcademico,
  ....  nameU: users.nameU,
  ....  lastnameU: users.lastnameU,
  ....  email: users.email,
  ....  iat: moment().unix(),
  ....  exp: moment().add(30, "days").unix() // 30 days expiration
  ....}
  ....return jwt.encode(payload, key);
}
```

Figura 6. jwt.js

Conexión a la Base de Datos

Archivo database.js

El archivo *database.js* se encarga de la conexión entre el servidor y la base de datos MySQL.

A logo for the file database.js, featuring the letters 'JS' in yellow and 'database.js' in white on a dark background.

Librería y Configuración:

- Importa la librería *mysql* para interactuar con la base de datos.
- Define una conexión con la base de datos utilizando la función *mysql.createConnection*.
- Especifica los detalles de conexión, como el host (*localhost*), la base de datos (*Usuarios*), el usuario (*root*), y la contraseña (*admin123*).

Conexión:

- Utiliza el método *conexion.connect* para establecer una conexión a la base de datos.
- Si la conexión es exitosa, muestra un mensaje en la consola indicando que la conexión se realizó con éxito.

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

const conexion = mysql.createConnection({
  ... host: 'localhost',
  ... database: 'Usuarios',
  ... user: 'root',
  ... password: 'admin123',
  ...
});

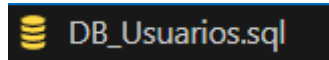
conexion.connect(function(error){
  ... if(error){
  ...   ... throw error;
  ... }else{
  ...   ... console.log('Conexion exitosa...!');
  ... }
});

module.exports = conexion;
```

Figura 7. *database.js*

Archivo SQL DB_Usuarios.sql

El archivo *DB_Usuarios.sql* contiene las instrucciones SQL para crear la estructura de la base de datos.



Creación de la Base de Datos

- Utiliza la sentencia *CREATE DATABASE* para crear una base de datos llamada *Usuarios*.
- Utiliza la sentencia *USE* para seleccionar la base de datos *Usuarios* para su uso.

Creación de Tablas

Define varias tablas en la base de datos, incluyendo:

- *Users*: Almacena información de los usuarios, como el *código de registro académico*, *nombre*, *apellidos*, *correo electrónico* y *contraseña*.
- *Publications*: Almacena publicaciones con detalles como *el usuario que la creó*, *el curso*, *el catedrático*, *el mensaje* y *la fecha de creación*.
- *Comments*: Almacena comentarios con detalles como *la publicación a la que están asociados*, *el usuario que los hizo*, *el mensaje* y *la fecha de creación*.
- *Courses*: Almacena información de cursos, incluyendo el *código de usuario*, *el nombre del curso* y *los créditos*.

Creación de Procedimientos

Define una serie de procedimientos almacenados que permiten realizar operaciones en la base de datos, como *registrar usuarios*, *buscar usuarios*, *eliminar usuarios* y *actualizar usuarios*.

Uso de Procedimientos

- Se incluye un ejemplo de llamada a un procedimiento almacenado para registrar un usuario utilizando la sentencia *CALL sp_registerUser(...)*.
- También se incluye una serie de consultas para mostrar información de las tablas *Users*, *Publications* y *Comments*.

Conclusiones

1. *Estructura Organizada*: El manual técnico presenta una estructura organizada y detallada de todos los componentes de la aplicación, desde el *servidor* hasta las *rutas* y *controladores*, lo que facilita la comprensión y el mantenimiento del código.
2. *Seguridad de Usuarios*: Se ha implementado un sistema de autenticación de usuarios mediante *tokens JWT*, lo que garantiza la seguridad de las rutas y recursos de la aplicación, permitiendo el acceso solo a *usuarios autorizados*.
3. *Conexión a la Base de Datos*: El manual explica cómo se establece la conexión a la base de datos *MySQL* y proporciona el archivo *SQL* necesario para *crear las tablas y procedimientos almacenados* requeridos por la aplicación.
4. *Gestión de Publicaciones, Comentarios, Usuarios, Cursos*: Se han desarrollado controladores y rutas para gestionar *publicaciones, comentarios, usuarios, cursos* lo que permite a los usuarios *crear, actualizar, eliminar y buscar* estos recursos de manera efectiva.