Proyecto de Implementación de Sitios Web Dinámicos

ACTIVIDAD 11: Instalación, Configuración y Utilización de Node.js para Backend con JavaScript

Sistema de Gestión de Tareas (Todo App)

Materia: Diseño e Implementación de Sitios Web Dinámicos
Escuela: EEST N.º 1 - .Eduardo Ader"Vicente López
Curso: 7° Año 2° Grupo A
Día y horario: Miércoles de 17:35 a 21:45 Hs
Modalidad: Presencial/Virtual Asincrónico

Profesor: Jorge Fabián Siles Guzmán
Estudiante: Santiago Ibarra
1° Cuatrimestre 2025

16 de julio de 2025

Índice

1 Introducción

Este informe documenta la implementación completa de un sistema backend utilizando Node.js, Express.js y MySQL para la gestión de tareas (Todo App). El proyecto fue desarrollado como parte de la Actividad 11 de la materia "Diseño e Implementación de Sitios Web Dinámicos.en la EEST N.º 1 .Eduardo Ader".

El sistema implementa funcionalidades completas de autenticación de usuarios, gestión de tareas con operaciones CRUD, y una API RESTful bien estructurada. Se incluye también un frontend básico para interactuar con el backend.

2 Link Github

3 Arquitectura del Sistema

3.1 Estructura del Proyecto

El proyecto sigue una arquitectura MVC (Model-View-Controller) con la siguiente estructura:

3.2 Tecnologías Utilizadas

- **Node.js**: Runtime de JavaScript para el servidor
- **Express.js**: Framework web para Node.js
- **MySQL**: Base de datos relacional
- **MySQL2**: Driver de MySQL para Node.js
- **bcryptjs**: Encriptación de contraseñas
- **jsonwebtoken**: Autenticación JWT
- **cors**: Middleware para CORS
- **dotenv**: Gestión de variables de entorno
- **nodemon**: Reinicio automático en desarrollo

4 Configuración del Proyecto

4.1 Dependencias del Proyecto

El archivo `package.json` contiene las siguientes dependencias:

```
{
  "name": "trabajon12",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
```

```
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "nodemon app.js",
    "start": "node app.js"
  },
  "dependencies": {
    "bcryptjs": "^3.0.2",
    "cors": "^2.8.5",
    "dotenv": "^17.2.0",
    "express": "^5.1.0",
    "jsonwebtoken": "^9.0.2",
    "mysql2": "^3.14.2"
  },
  "devDependencies": {
    "nodemon": "^3.1.10"
  }
}
```

4.2 Variables de Entorno

El archivo `entorn.env` contiene la configuración de la base de datos:

```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=
DB_NAME=todo_app
PORT=3000
JWT_SECRET=tu_secreto_jwt_super_seguro
```

5 Implementación del Backend

5.1 Configuración del Servidor Principal

El archivo `app.js` configura el servidor Express con todos los middlewares necesarios:

```
const express = require('express');
const cors = require('cors');
require('dotenv').config({ path: './entorn.env' });

const authRoutes = require('./routes/authRoutes');
const taskRoutes = require('./routes/taskRoutes');

const app = express();

// Middlewares
app.use(cors());
app.use(express.json());
app.use(express.static('public'));

// Health check route
app.get('/api/health', (req, res) => {
```

```
    res.json({ status: 'OK', message: 'Server is running' });
});

// Rutas
app.use('/api/auth', authRoutes);
app.use('/api/tasks', taskRoutes);

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {
    console.log(`Servidor corriendo en http://localhost:${PORT}`);
});
```

5.2 Configuración de la Base de Datos

El archivo `config/db.js` establece la conexión con MySQL:

```
const mysql = require('mysql2');
require('dotenv').config();

const pool = mysql.createPool({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
    waitForConnections: true,
    connectionLimit: 10,
    queueLimit: 0
});

module.exports = pool.promise();
```

6 Modelos de Datos

6.1 Modelo de Usuario

El archivo `models/user.js` define las operaciones de base de datos para usuarios:

```
const db = require('../config/db');

class User {
    static async create(username, email, password) {
        const sql = 'INSERT INTO users (username, email, password) VALUES (?, ?, ?)';
        const [result] = await db.execute(sql, [username, email, password]);
        return result.insertId;
    }

```

```
    static async findByEmail(email) {
        const sql = 'SELECT * FROM users WHERE email = ?';
        const [rows] = await db.execute(sql, [email]);
        return rows[0];
    }

    static async findById(id) {
        const sql = 'SELECT id, username, email, created_at FROM
           users WHERE id = ?';
        const [rows] = await db.execute(sql, [id]);
        return rows[0];
    }
}

module.exports = User;
```

6.2 Modelo de Tarea

El archivo `models/task.js` define las operaciones CRUD para tareas:

```
const db = require('../config/db');

class Task {
    static async create(title, description, userId) {
        const sql = 'INSERT INTO tasks (title, description,
           user_id) VALUES (?, ?, ?)';
        const [result] = await db.execute(sql, [title,
           description, userId]);
        return result.insertId;
    }

    static async findByUserId(userId) {
        const sql = 'SELECT * FROM tasks WHERE user_id = ? ORDER
           BY created_at DESC';
        const [rows] = await db.execute(sql, [userId]);
        return rows;
    }

    static async findById(id) {
        const sql = 'SELECT * FROM tasks WHERE id = ?';
        const [rows] = await db.execute(sql, [id]);
        return rows[0];
    }

    static async update(id, title, description, completed) {
        const sql = 'UPDATE tasks SET title = ?, description = ?,
            completed = ? WHERE id = ?';
        const [result] = await db.execute(sql, [title,
           description, completed, id]);
        return result.affectedRows;
    }
```

```
27
28      static async delete(id) {
29          const sql = 'DELETE FROM tasks WHERE id = ?';
30          const [result] = await db.execute(sql, [id]);
31          return result.affectedRows;
32      }
33  }
34
35  module.exports = Task;
```

7 Controladores

7.1 Controlador de Autenticación

El archivo `controllers/authController.js` maneja el registro y login de usuarios:

7.2 Controlador de Tareas

El archivo `controllers/taskController.js` implementa las operaciones CRUD para tareas:

8 Middlewares

8.1 Middleware de Autenticación

El archivo `middlewares/authMiddleware.js` verifica tokens JWT:

9 Rutas

9.1 Rutas de Autenticación

El archivo `routes/authRoutes.js` define los endpoints de autenticación:

```
1  const express = require('express');
2  const router = express.Router();
3  const authController = require('../controllers/authController');
4
5  router.post('/register', authController.register);
6  router.post('/login', authController.login);
7
8  module.exports = router;
```

9.2 Rutas de Tareas

El archivo `routes/taskRoutes.js` define los endpoints protegidos para tareas:

```
const express = require('express');
const router = express.Router();
const taskController = require('../controllers/taskController');
const authMiddleware = require('../middlewares/authMiddleware');

router.get('/', authMiddleware, taskController.getAllTasks);
router.post('/', authMiddleware, taskController.createTask);
router.put('/:id', authMiddleware, taskController.updateTask);
router.delete('/:id', authMiddleware, taskController.deleteTask);

module.exports = router;
```

10 API Endpoints

10.1 Endpoints de Autenticación

Método	Endpoint	Descripción	Parámetros
POST	`/api/auth/register`	Registrar nuevo usuario	{`username, email, password`}
POST	`/api/auth/login`	Iniciar sesión	{`email, password`}

10.2 Endpoints de Tareas (Protegidos)

Método	Endpoint	Descripción	Parámetros
GET	`/api/tasks`	Obtener todas las tareas del usuario	Header: `Authorization` `Bearer <token>`
POST	`/api/tasks`	Crear nueva tarea	{`title, description`} + Hea der: `Authorization`
PUT	`/api/tasks/:id`	Actualizar tarea existente	{`title, description, completed`} + Heade `Authorization`
DELETE	`/api/tasks/:id`	Eliminar tarea	Header: `Authorization`

10.3 Endpoints de Sistema

Método	Endpoint	Descripción	Respuesta
GET	`/api/health`	Verificar estado del servidor	{`status: .ºK", message: "Server is running"`}

11 Frontend

11.1 Interfaz de Usuario

El proyecto incluye un frontend básico en `public/index.html` que permite:

- Registro e inicio de sesión de usuarios
- Creación, edición y eliminación de tareas
- Visualización de tareas en tiempo real
- Interfaz responsive y moderna

11.2 Estilos CSS

Los estilos están definidos en `public_css/style.css` con un diseño moderno y responsive.

11.3 JavaScript del Cliente

La lógica del frontend está en `public_js/main.js` e incluye:

- Gestión de tokens JWT
- Llamadas a la API REST
- Manipulación del DOM
- Validación de formularios

12 Base de Datos

12.1 Esquema de la Base de Datos

El sistema utiliza MySQL con las siguientes tablas:

12.1.1. Tabla users

```
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

12.1.2. Tabla tasks

```
CREATE TABLE tasks (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    completed BOOLEAN DEFAULT FALSE,
    user_id INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
       CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

13 Instalación y Configuración

13.1 Requisitos Previos

1. Node.js (versión 14 o superior)
2. MySQL (versión 5.7 o superior)
3. npm (incluido con Node.js)

13.2 Pasos de Instalación

1. **Clonar o descargar el proyecto**
2. **Instalar dependencias:**

```
    npm install
```

3. **Configurar la base de datos:**

```
    CREATE DATABASE todo_app;
    USE todo_app;
```

4. **Ejecutar los scripts SQL para crear las tablas**
5. **Configurar variables de entorno** en `entorn.env`
6. **Ejecutar el servidor:**

```
    npm run dev
```

14 Pruebas y Validación

14.1 Pruebas de la API

Se realizaron pruebas exhaustivas de todos los endpoints:

14.2 Errores Solucionados

Durante el desarrollo se identificaron y corrigieron los siguientes errores:

1. **Errores de sintaxis:** Eliminación de comentarios `[cite: ...]` inválidos
2. **Problemas de case-sensitive:** Corrección de imports de archivos

3. **Configuración de variables de entorno:** Especificación correcta de la ruta del archivo
4. **Rutas faltantes:** Implementación de endpoint de health check

15 Características de Seguridad

15.1 Autenticación JWT

- Tokens con expiración de 1 hora
- Verificación automática en rutas protegidas
- Manejo seguro de errores de autenticación

15.2 Encriptación de Contraseñas

- Uso de bcryptjs con salt rounds de 10
- Almacenamiento seguro de contraseñas en la base de datos
- Comparación segura durante el login

15.3 Validación de Datos

- Validación de campos obligatorios
- Sanitización de entradas de usuario
- Manejo de errores de base de datos

16 Optimizaciones Implementadas

16.1 Base de Datos

- Pool de conexiones para mejor rendimiento
- Índices en campos de búsqueda frecuente
- Consultas optimizadas con prepared statements

16.2 Código

- Estructura modular y reutilizable
- Manejo asíncrono de operaciones
- Separación clara de responsabilidades (MVC)

17 Conclusiones

Este proyecto demuestra la implementación exitosa de un backend completo utilizando Node.js y Express.js. Se han logrado todos los objetivos planteados en la consigna académica:

El sistema desarrollado es escalable, mantenible y sigue las mejores prácticas de desarrollo web moderno. La arquitectura MVC implementada permite una fácil extensión de funcionalidades y la separación clara de responsabilidades facilita el mantenimiento del código.

18 Anexos

18.1 Comandos Útiles

18.2 Recursos Adicionales

- Documentación oficial de Node.js
- Documentación oficial de Express.js
- Documentación oficial de MySQL
- Información sobre JWT