TECNOLÓGICO
NACIONAL DE MÉXICO


INGENIERÍA EN SISTEMAS
COMPUTACIONALES

**Instituto Tecnológico Superior de Uruapan**

# Manual Técnico

``Web Gestión TEC´´

**Alumnos:**
Saucedo Bautista Rafael Benjamin
Servin Franco Brandon Alexis
Vazquez Sanchez Jose Antonio
Hernandez Garcia Jose Luis
Gonzalez Orozco Juan Carlos

**Docente:**
Vizcaino Paz Graciela Alicia

# Contenido

# 1. INTRODUCCIÓN

## 1.1 Visión General

**Web-Gestion-TEC** es una solución integral de software diseñada para la administración académica y la gestión de expedientes docentes. El sistema nace de la necesidad de digitalizar y centralizar procesos que anteriormente se realizaban de manera manual o fragmentada, permitiendo a la institución tener un control preciso sobre la plantilla docente, el cumplimiento de requisitos administrativos y el historial de actividades (bitácora).

## 1.2 Propósito del Documento

Este manual técnico tiene como propósito documentar exhaustivamente la arquitectura, lógica y estructura del sistema **Web-Gestion-TEC**. Está dirigido al equipo de desarrollo, administradores de sistemas y auditores técnicos que requieran comprender el funcionamiento interno para realizar tareas de mantenimiento, escalabilidad o integración con otros sistemas.

## 1.3 Ámbito del Sistema

El sistema abarca los siguientes módulos operativos:

- Autenticación y seguridad de usuarios.
- Gestión CRUD (Crear, Leer, Actualizar, Eliminar) de Docentes.
- Administración de Semestres Académicos.
- Control de Requisitos y Documentación.
- Auditoría de movimientos mediante Bitácora del Sistema.

# 2. OBJETIVOS DEL SISTEMA

## 2.1 Objetivo General

Desarrollar e implementar una plataforma web robusta y escalable que optimice la gestión de la información docente y administrativa, garantizando la integridad de los datos y facilitando la toma de decisiones mediante el acceso rápido a la información.

## 2.2 Objetivos Específicos

1. **Centralización de Datos:** Unificar la información dispersa de los docentes (datos personales, laborales y académicos) en una base de datos relacional única.
2. **Automatización de Control de Requisitos:** Proveer mecanismos visuales y lógicos para verificar rápidamente el estado de cumplimiento de documentos por parte del personal.
3. **Trazabilidad y Seguridad:** Implementar un sistema de bitácora inmutable que registre quién, cuándo y qué cambios se realizaron en el sistema para fines de auditoría.
4. **Interoperabilidad:** Exponer una API RESTful interna que permita al frontend (y a futuros sistemas) consumir los datos de manera estandarizada mediante JSON.

# 3. ARQUITECTURA GENERAL

## 3.1 Patrón Arquitectónico: MVC (Modelo-Vista-Controlador)

El proyecto se fundamenta en el patrón de diseño **MVC**, el cual desacopla la lógica de negocio de la interfaz de usuario. Esta arquitectura facilita el mantenimiento del código y permite que múltiples desarrolladores trabajen en diferentes capas simultáneamente sin conflictos.

### Componentes de la Arquitectura

- **Modelo (app/models):** Representa la estructura de los datos y las reglas de negocio. Es la única capa que se comunica directamente con la base de datos MySQL.
- **Vista (public/*.html):** La capa de presentación. En este proyecto, las vistas son archivos HTML5 independientes que actúan como "esqueletos" que son llenados dinámicamente mediante JavaScript.
- **Controlador (app/controllers):** Actúa como intermediario. Recibe las peticiones HTTP del usuario, procesa la entrada, invoca a los modelos necesarios y devuelve una respuesta estructurada (generalmente JSON).

## 3.2 Diagrama de Flujo de Datos

1. **Petición:** El navegador (Cliente) envía una solicitud HTTP (ej. GET /docentes).
2. **Enrutamiento:** El archivo index.php y config/Router.php interceptan la solicitud y determinan qué Controlador debe atenderla.
3. **Procesamiento:** El DocenteController recibe la orden.
4. **Consulta:** El controlador llama a DocenteModel para obtener datos.
5. **Respuesta:** El modelo retorna un array de datos; el controlador lo transforma a JSON y lo envía al navegador.
6. **Renderizado:** El JavaScript del cliente (docentes.js) recibe el JSON y actualiza el DOM del HTML.

# 4. TECNOLOGÍAS DETALLADAS

El stack tecnológico se seleccionó priorizando la compatibilidad universal en servidores de hosting compartido y el rendimiento nativo sin dependencias pesadas.

| Capa | Tecnología | Versión / Características |
|---|---|---|
| **Backend Core** | PHP | 7.4 / 8.x (POO, Strict Types) |
| **Base de Datos** | MySQL / MariaDB | Motor InnoDB, UTF-8mb4 |
| **Servidor Web** | Apache | Mod_Rewrite habilitado para URLs amigables |
| **Frontend Estructura** | HTML5 | Semántico, Modular |
| **Frontend Estilos** | CSS3 | Flexbox, Grid, Variables CSS (Custom Properties) |
| **Frontend Lógica** | JavaScript (ES6+) | Vanilla JS (Sin frameworks), Fetch API, Async/Await |
| **Intercambio de Datos** | JSON | Formato estándar para respuestas de API |
| **Control de Versiones** | Git | Gestión de código fuente |

# 5. DOCUMENTACIÓN COMPLETA DE ENDPOINTS API

El sistema expone los siguientes endpoints REST. Todos los endpoints retornan respuestas en formato application/json.

## 5.1 Módulo de Autenticación (AuthController)

- **POST** /auth/login
  - **Descripción:** Valida credenciales de usuario e inicia sesión.
  - **Parámetros Body:** { "usuario":String, "password":String }
  - **Respuesta:** { "status": "success", "token": "..." }

## 5.2 Módulo de Docentes (DocenteController)

- **GET** /docentes
  - **Descripción:** Obtiene el listado de todos los docentes activos.
- **GET** /docentes/get/{id}
  - **Descripción:** Obtiene los datos detallados de un docente específico por ID.
- **POST** /docentes/create
  - **Descripción:** Registra un nuevo docente en la base de datos.
  - **Body:** Objeto JSON con datos del docente (nombre, rfc, curp, etc.).
- **POST** /docentes/update
  - **Descripción:** Actualiza la información de un docente existente.
- **POST** /docentes/delete
  - **Descripción:** Elimina (o desactiva) un registro de docente.

## 5.3 Módulo de Requisitos (RequisitoController)

- **GET** /requisitos/getByDocente/{id}
  - **Descripción:** Lista los requisitos asignados a un docente y su estado (cumplido/pendiente).
- **POST** /requisitos/updateStatus
  - **Descripción:** Cambia el estado de un requisito (toggle checkbox).

## 5.4 Módulo de Semestres (SemestreController)

- **GET** /semestres/actual
  - **Descripción:** Retorna el semestre marcado como "Actual" en el sistema.

# 6. MODELO DE BASE DE DATOS

La persistencia de datos se maneja en MySQL. A continuación, se detalla el esquema relacional inferido del archivo Dump20251009.sql.

## 6.1 Tabla: usuarios

Almacena las credenciales de los administradores del sistema.

- **id_usuario** (INT, PK, AUTO_INCREMENT): Identificador único.
- **nombre_usuario** (VARCHAR 50): Username para login.
- **password** (VARCHAR 255): Hash encriptado de la contraseña.
- **rol** (ENUM): Nivel de acceso ('admin', 'visor').

## 6.2 Tabla: docentes

Tabla central que almacena la información del personal.

- **id_docente** (INT, PK, AUTO_INCREMENT)
- **nombre** (VARCHAR 100): Nombre(s) del docente.
- **apellidos** (VARCHAR 100): Apellidos.
- **rfc** (VARCHAR 13): Registro Federal de Contribuyentes (Debe ser único).
- **curp** (VARCHAR 18): Clave Única de Registro de Población.
- **telefono** (VARCHAR 15): Número de contacto.
- **correo** (VARCHAR 100): Email institucional o personal.

## 6.3 Tabla: requisitos

Catálogo de documentos que los docentes deben entregar.

- **id_requisito** (INT, PK)
- **nombre_requisito** (VARCHAR): Nombre del documento (ej. "Acta de Nacimiento").
- **descripcion** (TEXT): Detalles adicionales.

## 6.4 Tabla: estado_requisitos (Tabla Pivote)

Relaciona Docentes con Requisitos para controlar el cumplimiento.

- **id_relacion** (INT, PK)
- **fk_docente** (INT, FK -> docentes.id_docente)
- **fk_requisito** (INT, FK -> requisitos.id_requisito)
- **estado** (BOOLEAN): 1 (Entregado), 0 (Pendiente).
- **fecha_actualizacion** (DATETIME): Última vez que se modificó.

## 6.5 Tabla: bitacora

Historial de auditoría.

- **id_bitacora** (INT, PK)
- **accion** (VARCHAR): Descripción breve (ej. "CREATE DOCENTE").
- **usuario** (VARCHAR): Usuario que realizó la acción.
- **fecha** (TIMESTAMP): Momento exacto del evento.
- **detalles** (JSON/TEXT): Data completa del cambio realizado.

# 7. EJEMPLOS DE CÓDIGO COMPLETO DEL BACKEND

A continuación, se presentan fragmentos clave del código fuente que demuestran la implementación de la arquitectura MVC.

## 7.1 Conexión a Base de Datos

Archivo: config/Database.php. Garantiza una única conexión eficiente.

```
class Database {
    private $host = "localhost";
    private $db_name = "webnahim"; // Nombre de la DB
    private $username = "root";
    private $password = "";
    public $conn;

    public function getConnection() {
        $this->conn = null;
        try {
            $this->conn = new PDO(
                "mysql:host=" . $this->host . ";dbname=" . $this->db_name,
                $this->username,
                $this->password
            );
            $this->conn->exec("set names utf8");
            $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        } catch(PDOException $exception) {
            echo "Error de conexión: " . $exception->getMessage();
        }
        return $this->conn;
    }
}
```

## 7.2 Controlador Base (Herencia)

Archivo: app/controllers/BaseController.php. Provee funcionalidades comunes a todos los controladores.

```
class BaseController {
    // Método para cargar modelos dinámicamente
    public function model($model) {
        require_once '../app/models/' . $model . '.php';
        return new $model();
    }

    // Método para obtener datos del POST body (JSON)
    public function getJsonInput() {
        return json_decode(file_get_contents('php://input'), true);
    }
}
```

## 7.3 Modelo de Docentes (Lógica de Datos)

Archivo: app/models/DocenteModel.php.

```
class DocenteModel extends BaseModel {
    private $table = 'docentes';

    public function getAll() {
        // Uso de PDO Prepared Statements implícito en la clase padre o directa
        $query = "SELECT * FROM " . $this->table . " ORDER BY apellidos ASC";
        $stmt = $this->db->prepare($query);
        $stmt->execute();
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }

    public function create($data) {
        $query = "INSERT INTO " . $this->table . " (nombre, apellidos, rfc, curp, telefono, correo)
                  VALUES (:nombre, :apellidos, :rfc, :curp, :telefono, :correo)";
        $stmt = $this->db->prepare($query);
        // Bindeo de parámetros para seguridad
        $stmt->bindParam(':nombre', $data['nombre']);
        $stmt->bindParam(':apellidos', $data['apellidos']);
        // ... (resto de bindeos)
        return $stmt->execute();
    }
}
```

# 8. DOCUMENTACIÓN DETALLADA DEL FRONTEND

*Nota Técnica: El proyecto utiliza una arquitectura **SPA (Single Page Application)** construida con **JavaScript Vanilla (ES6+)**. Aunque funcionalmente es similar a frameworks como React (componentización, manejo de estado asíncrono), se ha optado por JS nativo para reducir la sobrecarga y dependencias.*

## 8.1 Cliente API (api-client.js)

Este módulo actúa como una capa de abstracción para fetch, similar a lo que haría Axios en React. Centraliza la configuración de cabeceras y manejo de errores.

```
// public/assets/js/api-client.js
const ApiClient = {
    baseUrl: 'http://localhost/Web-Gestion-TEC', // URL base configurable

    async get(endpoint) {
        try {
            const response = await fetch(`${this.baseUrl}${endpoint}`);
            if (!response.ok) throw new Error('Network response was not ok');
            return await response.json();
        } catch (error) {
            console.error('Fetch error:', error);
            throw error;
        }
    },

    async post(endpoint, data) {
        try {
            const response = await fetch(`${this.baseUrl}${endpoint}`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(data)
            });
            return await response.json();
        } catch (error) {
```

```
            console.error('Fetch error:', error);
            throw error;
        }
    }
};
```

## 8.2 Lógica de Vistas (docentes.js)

Manejo del DOM y consumo de la API.

```
// public/assets/js/docentes.js
document.addEventListener('DOMContentLoaded', () => {
    loadDocentes();
});

async function loadDocentes() {
    const tableBody = document.getElementById('docentes-table-body');
    tableBody.innerHTML = '<tr><td colspan="5">Cargando...</td></tr>';

    try {
        const response = await ApiClient.get('/docentes');

        if (response.data && response.data.length > 0) {
            renderTable(response.data);
        } else {
            tableBody.innerHTML = '<tr><td colspan="5">No hay docentes registrados.</td></tr>';
        }
    } catch (error) {
        alert('Error al cargar docentes');
    }
}

function renderTable(docentes) {
    const tableBody = document.getElementById('docentes-table-body');
    tableBody.innerHTML = '';

    docentes.forEach(docente => {
        const row = `
            <tr>
                <td>${docente.rfc}</td>
```

```
            <td>${docente.nombre} ${docente.apellidos}</td>
            <td>${docente.correo}</td>
            <td>
              <button onclick="editDocente(${docente.id_docente})" class="btn-
edit">Editar</button>
              <button onclick="deleteDocente(${docente.id_docente})" class="btn-
delete">Eliminar</button>
            </td>
          </tr>
        `;
        tableBody.insertAdjacentHTML('beforeend', row);
    });
}
```

# 9. SEGURIDAD DEL SISTEMA

La seguridad es transversal a toda la aplicación, implementada tanto en el servidor web, el código PHP y la base de datos.

## 9.1 Middleware de Autenticación

El archivo app/middleware/AuthMiddleware.php intercepta cada petición a las rutas protegidas. Verifica si existe una sesión válida antes de permitir que el controlador procese la solicitud. Si la verificación falla, retorna un error 401 Unauthorized inmediatamente.

## 9.2 Prevención de SQL Injection

El uso estricto de **PDO (PHP Data Objects)** con sentencias preparadas (prepare y bindParam) en todos los modelos (app/models/*) neutraliza cualquier intento de inyección SQL. Los datos del usuario nunca se concatenan directamente en las cadenas de consulta SQL.

## 9.3 Protección de Archivos Sensibles

Se utiliza un archivo .htaccess en el directorio raíz y en public/ para:

1. **Redirección de Tráfico:** Forzar que todo el tráfico pase por index.php, centralizando el punto de entrada.
2. **Bloqueo de Directorios:** Evitar el listado de directorios (Directory Browsing) para que los atacantes no puedan ver la estructura de archivos.
3. **Protección de Core:** Bloquear el acceso directo HTTP a las carpetas app/ y config/.

## 9.4 Sanitización de Salida (XSS)

Aunque el frontend renderiza datos dinámicamente, el sistema asegura que los datos sensibles se traten como texto plano al inyectarlos en el DOM (usando textContent o plantillas controladas), reduciendo el riesgo de Cross-Site Scripting (XSS).