

# Estándares y metodologías

Para el desarrollo correcto de la aplicación web “MatFi” se adoptarán los siguientes estándares y metodologías reconocidas para tecnologías de frontend y backend, esto con el objetivo de garantizar coherencia técnica, organización estructural y alineación con especificaciones formales vigentes.

## I. Tecnologías Front-End

Con el objetivo de garantizar calidad, mantenibilidad, escalabilidad y alineación con buenas prácticas de la industria, el desarrollo de la aplicación web seguirá estándares internacionales y metodologías reconocidas para CSS y JavaScript.

### HTML

#### I. HTML Living Standard

El desarrollo de los documentos HTML tomará en consideración el *HTML Living Standard*, mantenido por el Grupo de Trabajo sobre Tecnología de Aplicaciones de Hipertexto Web (WHATWG). Esto con el objetivo de mantener buenas prácticas en los documentos; aunque WHATWG es una comunidad de personas interesadas en desarrollar la web a través de estándares y pruebas se emplearán como buenas prácticas y recomendaciones; y su especificación es actualmente la referencia principal adoptada por los navegadores modernos y la industria del desarrollo web.

WHATWG. (2026, 19 de febrero). *HTML Living Standard*.  
<https://html.spec.whatwg.org/multipage/#toc-introduction>

### CSS

#### I. Metodología BEM (Block Element Modifier)

Se utilizará la metodología BEM como convención de nomenclatura para las clases CSS. Esta metodología define una estructura basada en tres conceptos principales que son bloque, elemento y modificador. El bloque representa una entidad independiente dentro de la interfaz, el elemento corresponde a una parte interna que pertenece a dicho bloque y el modificador se utiliza para indicar una variación o estado específico del bloque o del elemento. La estructura de nombrado sigue el patrón bloque, bloque\_\_elemento y bloque--modificador, permitiendo mantener una organización clara y consistente en la construcción de los estilos dentro del proyecto.

#### II. ITCSS (Inverted Triangle CSS)

Se implementará la arquitectura ITCSS para la organización interna de los archivos CSS. ITCSS establece una jerarquía estructurada en capas que van desde configuraciones globales hasta componentes específicos, organizando los estilos bajo un modelo de dependencia progresiva.

## JavaScript

### I. ECMAScript

El desarrollo en JavaScript se alineará con la especificación oficial de ECMAScript definida en el estándar ECMA-262, mantenido por el comité técnico TC39. Esta especificación define formalmente la sintaxis, semántica y comportamiento del lenguaje JavaScript y constituye el marco normativo para su implementación y uso en navegadores y entornos de ejecución modernos.

Ecma International. (2026, 17 de febrero). *ECMAScript® 2026 language specification (ECMA-262)*  
<https://tc39.es/ecma262/>

# Estándares de Codificación Backend - MatFi

## APIs REST con Node.js y Express

## 2. Tecnologías Back-End

Con el objetivo de garantizar calidad, mantenibilidad, seguridad y alineación con buenas prácticas de la industria, el desarrollo del backend de la aplicación web seguirá estándares internacionales y metodologías reconocidas para APIs REST, Node.js y Express.js.

### Arquitectura REST (Representational State Transfer)

#### 1. Principios REST de Roy Fielding

El diseño de las APIs seguirá los principios arquitectónicos REST definidos por Roy Thomas Fielding en su tesis doctoral "Architectural Styles and the Design of Network-based Software Architectures" (2000). Estos principios incluyen:

- **Cliente-Servidor:** Separación clara entre frontend y backend
- **Sin estado (Stateless):** Cada petición contiene toda la información necesaria
- **Cacheable:** Las respuestas indican si pueden ser almacenadas en caché

- **Sistema en capas:** Arquitectura jerárquica de componentes
- **Interfaz uniforme:** Uso consistente de métodos HTTP y recursos

**Referencia:** Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* [Tesis doctoral, Universidad de California, Irvine]. [https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

## 2. Protocolo HTTP/1.1

Las APIs REST implementarán el protocolo HTTP/1.1 según las especificaciones RFC 7230, RFC 7231, RFC 7232, RFC 7233, RFC 7234 y RFC 7235 del Internet Engineering Task Force (IETF).

### Métodos HTTP utilizados:

- **GET:** Obtener recursos (idempotente, seguro)
- **POST:** Crear nuevos recursos
- **PUT:** Actualizar recursos existentes (idempotente)
- **DELETE:** Eliminar recursos (idempotente)

### Códigos de estado HTTP:

- **2xx (Éxito):** 200 OK, 201 Created
- **4xx (Errores del cliente):** 400 Bad Request, 401 Unauthorized, 404 Not Found, 422 Unprocessable Entity
- **5xx (Errores del servidor):** 500 Internal Server Error

**Referencias:** Fielding, R., & Reschke, J. (Eds.). (2014). *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content* (RFC 7231). IETF. <https://www.rfc-editor.org/rfc/rfc7231>

Fielding, R., & Reschke, J. (Eds.). (2014). *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing* (RFC 7230). IETF. <https://www.rfc-editor.org/rfc/rfc7230>

## 3. Formato JSON (JavaScript Object Notation)

El intercambio de datos entre cliente y servidor se realizará exclusivamente mediante el formato JSON, siguiendo la especificación RFC 8259 del IETF. JSON proporciona un formato ligero, legible y ampliamente soportado para la representación estructurada de datos.

**Referencia:** Bray, T. (Ed.). (2017). *The JavaScript Object Notation (JSON) Data Interchange Format* (RFC 8259). IETF. <https://www.rfc-editor.org/rfc/rfc8259>

## Convenciones de Desarrollo Node.js

### 1. Node.js LTS (Long Term Support)

El desarrollo del backend utilizará versiones LTS de Node.js, garantizando estabilidad, seguridad y soporte extendido. La versión mínima requerida es Node.js 18.x LTS.

**Referencia:** OpenJS Foundation. (2026). *Node.js Release Working Group*.  
<https://nodejs.org/en/about/previous-releases>

## 2. Módulos CommonJS y ES Modules

Se utilizará el sistema de módulos ES Modules (ECMAScript Modules) para mantener consistencia con el estándar ECMAScript 2015+ utilizado en el frontend, permitiendo el uso de `import` y `export`.

**Configuración en package.json:**

```
{  
  "type": "module"  
}
```

## Framework Express.js

### 1. Convenciones de Express.js

Se seguirán las convenciones oficiales del framework Express.js 4.x para la estructura y organización del código backend.

**Estructura recomendada:**

```
backend/  
├── controllers/  # Lógica de negocio  
├── models/      # Acceso a datos  
├── routes/      # Definición de endpoints  
├── middleware/   # Funciones intermedias  
├── services/     # Servicios auxiliares  
├── config/       # Configuraciones  
└── utils/        # Utilidades
```

**Referencia:** OpenJS Foundation. (2026). *Express.js Documentation - Best Practices*.  
<https://expressjs.com/en/advanced/best-practice-performance.html>

## 2. Middleware Pattern

Se implementará el patrón de middleware de Express.js para procesar peticiones de forma secuencial:

1. **Middleware global:** CORS, body-parser, compression
2. **Middleware de autenticación:** Verificación de JWT
3. **Middleware de validación:** Validación de datos de entrada
4. **Middleware de manejo de errores:** Captura y formateo de errores

## Convenciones de Nomenclatura

### 1. Nomenclatura de Variables y Funciones

Se utilizará **camelCase** para nombres de variables, funciones y métodos, siguiendo las convenciones de JavaScript establecidas en el documento ERS.

#### Ejemplos:

```
// Variables y funciones

const getUserById = async (userId) => { }

let isAuthenticated = false;

const authToken = generateToken();


// Constantes (UPPER_SNAKE_CASE)

const MAX_LOGIN_ATTEMPTS = 5;

const JWT_SECRET_KEY = process.env.JWT_SECRET;
```

### 2. Nomenclatura de Archivos

- **Archivos de código:** camelCase (Ej: `UserController.js`, `AuthService.js`)
- **Archivos de configuración:** kebab-case (Ej: `database-config.js`)

### 3. Nomenclatura de Rutas API

Las rutas seguirán convenciones RESTful utilizando sustantivos en plural y kebab-case:

#### Correctas:

```
GET    /api/usuarios
GET    /api/usuarios/:id
POST   /api/usuarios/:id/rutinas
```

GET /api/recetas-favoritas

### Incorrectas:

GET /api/getUsers ❌ (verbo en URL)

GET /api/Usuario ❌ (singular, PascalCase)

POST /api/crear-usuario ❌ (verbo en URL)

## Seguridad

### 1. JSON Web Tokens (JWT)

Se implementará autenticación mediante JSON Web Tokens siguiendo el estándar RFC 7519.

**Referencia:** Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)* (RFC 7519). IETF. <https://www.rfc-editor.org/rfc/rfc7519>

### 2. Encriptación de Contraseñas

Las contraseñas se almacenarán utilizando el algoritmo bcrypt con un factor de trabajo (salt rounds) mínimo de 10, siguiendo las recomendaciones de OWASP (Open Web Application Security Project).

**Referencia:** OWASP Foundation. (2021). *Password Storage Cheat Sheet*. [https://cheatsheetseries.owasp.org/cheatsheets/Password\\_Storage\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html)

### 3. Validación y Sanitización

Toda entrada de usuario será validada y sanitizada para prevenir inyecciones SQL, XSS y otros ataques comunes, siguiendo las guías de OWASP.

**Referencia:** OWASP Foundation. (2021). *Input Validation Cheat Sheet*. [https://cheatsheetseries.owasp.org/cheatsheets/Input\\_Validation\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html)

## Documentación de APIs

### 1. Especificación OpenAPI 3.0

La documentación de las APIs REST seguirá la especificación OpenAPI 3.0 (anteriormente Swagger), mantenida por la OpenAPI Initiative.

**Referencia:** OpenAPI Initiative. (2021). *OpenAPI Specification v3.0.3*. <https://spec.openapis.org/oas/v3.0.3>

