



## UNIVERSIDAD TECNOLOGICA DE TEHUACAN

### PROGRAMA EDUCATIVO EN:

Tecnologías de la Información en Área Desarrollo  
de Software Multiplataforma

### NOMBRE DEL PROFESOR:

JOSÉ MIGUEL CARRERA PACHECO

### INTEGRANTES DEL EQUIPO:

Ciro Julian Cervantes Zamora

### MATERIA:

Desarrollo Web Profesional

### TEMA:

Arquitectura de información y navegación accesible

TEHUACÁN, PUEBLA, ENERO-ABRIL 2026

## 1. Arquitectura de Software

### 1.1 ¿Qué es la Arquitectura de Software?

La arquitectura de software es la estructura fundamental de un sistema, que comprende sus componentes, las relaciones entre ellos y los principios que guían su diseño y evolución. Define cómo se organizan y comunican las diferentes partes del sistema para cumplir con los requisitos funcionales y no funcionales.

### 1.2 Tipos de Arquitectura

Tipo	Características	Uso Ideal
Monolito	Todo en un solo bloque. Fácil de desarrollar inicialmente. Difícil de escalar.	Proyectos pequeños, equipos pequeños, MVPs simples
3 Capas	Presentación, Lógica, Datos. Separación clara. Escalabilidad moderada.	Aplicaciones web de tamaño medio, equipos organizados
Microservicios	Servicios independientes. Altamente escalable. Complejidad alta.	Sistemas grandes, múltiples equipos, alta disponibilidad

### 1.3 Justificación: Arquitectura de 3 Capas para Rapiti

Para el proyecto Rapiti elegimos una arquitectura de 3 capas por las siguientes razones:

- **Separación clara de responsabilidades:** El frontend (React) maneja la presentación, el backend (Node.js/Express) gestiona la lógica de negocio, y PostgreSQL almacena los datos.
- **Escalabilidad adecuada:** Podemos escalar cada capa independientemente. Por ejemplo, añadir más instancias del backend sin tocar la base de datos.
- **Mantenibilidad:** Un equipo de 5 personas puede trabajar en paralelo: unos en frontend, otros en backend, sin conflictos constantes.
- **Complejidad justificada:** Es más compleja que un monolito pero menos que microservicios. Para un MVP con 3-5 tiendas y posibilidad de crecer a múltiples ciudades, es el balance perfecto.

- **Tecnologías maduras:** Existe amplia documentación y experiencia de la industria con este patrón.

## 2. Stack Tecnológico

### 2.1 Node.js

**¿Qué es?** Node.js es un entorno de ejecución de JavaScript del lado del servidor, construido sobre el motor V8 de Chrome.

**¿Por qué lo elegimos?**

- **JavaScript en todo el stack:** El equipo usa el mismo lenguaje en frontend y backend.
- **Ecosistema NPM:** Más de 1 millón de paquetes disponibles.
- **Arquitectura asíncrona:** Excelente para manejar múltiples peticiones simultáneas.
- **Rendimiento:** El motor V8 es muy rápido para APIs REST.

### 2.2 Express.js

**¿Qué es?** Framework web minimalista y flexible para Node.js.

**Ventajas:**

- API intuitiva para definir rutas y middleware
- Sistema modular para autenticación, logging, manejo de errores
- Fácil integrar JWT, Bcrypt, CORS

### 2.3 PostgreSQL

**¿Qué es?** Sistema de gestión de bases de datos relacional open-source.

**¿Por qué SQL y no NoSQL?**

- **Relaciones claras:** Producto → Precios → Tienda. SQL es ideal para esto.
- **Integridad de datos:** Foreign keys y constraints garantizan consistencia.
- **Consultas complejas:** JOINs para obtener productos con todos sus precios.
- **Esquema definido:** Evita errores de tipos de datos.

## 2.4 JWT (JSON Web Tokens)

**¿Qué es?** Estándar abierto para transmitir información de forma segura entre partes como objeto JSON.

### Flujo en Rapiti:

1. Tienda envía email/password a /api/auth/login
2. Backend verifica credenciales
3. Si correctas, genera JWT con {id, email, rol, tienda\_id}
4. Frontend guarda token en localStorage
5. Cada petición envía JWT en header Authorization
6. Middleware verifica token antes de procesar

## 2.5 Alternativas Consideradas

Framework	Lenguaje	Razón de Descarte
Django	Python	Mayor complejidad inicial. El equipo tiene más experiencia con JavaScript.
Spring Boot	Java	Verboso y pesado. Curva de aprendizaje alta. Overkill para MVP.
Laravel	PHP	Menos moderno. Ecosistema menos robusto para SPAs.

### 3. APIs REST

#### 3.1 ¿Qué es una API REST?

REST (Representational State Transfer) es un estilo arquitectónico para diseñar servicios web. Una API REST utiliza HTTP para la comunicación y sigue principios específicos de diseño.

#### 3.2 Principios REST

- **Recursos como sustantivos:** URLs representan recursos, no acciones.  
Ejemplo: /api/productos ✓ vs /api/obtenerProductos X
- **Verbos HTTP para acciones:**
  - GET: Obtener recursos
  - POST: Crear recursos
  - PUT/PATCH: Actualizar recursos
  - DELETE: Eliminar recursos
    - **Stateless:** Cada petición contiene toda la información necesaria (token JWT).
    - **Formato JSON:** Ligero y fácil de parsear para request/response.
    - **Códigos de estado HTTP:**
      - 200: OK, 201: Created
      - 400: Bad Request, 401: Unauthorized
      - 404: Not Found, 500: Server Error

#### 3.3 Ejemplo Request/Response

##### Request:

```
GET /api/productos?q=leche HTTP/1.1 Host: api.rapiti.com Content-Type:  
application/json
```

## 4. Sistema de Autenticación

### 4.1 ¿Qué es JWT?

JSON Web Token es un estándar para crear tokens de acceso. Un JWT consta de tres partes:

- **Header:** Tipo de token y algoritmo (HS256)
- **Payload:** Datos del usuario (id, email, rol, tienda\_id)
- **Signature:** Firma digital para verificar integridad

### 4.2 Roles de Usuario

Rol	Permisos	Ejemplo
admin	Crear tiendas, usuarios, productos. Ver todo el sistema.	Equipo Rapiti
tienda	Ver sus productos. Actualizar precios.	Don Juan, Papelería Mary
público	Buscar productos, ver comparaciones, ver mapa (sin login)	Compradores de Tehuacán

### 4.3 Bcrypt para Contraseñas

Bcrypt es un algoritmo de hashing diseñado específicamente para contraseñas:

- **One-way hash:** Imposible revertir
- **Salt automático:** Datos aleatorios únicos
- **Lento por diseño:** Dificulta ataques de fuerza bruta

## **5. Problemas que Resuelve el Backend**

### **5.1 Problemas de Negocio**

#### **Comparación de precios**

Agrega precios de múltiples tiendas y los devuelve ordenados del más barato al más caro.

#### **Actualización en tiempo real**

Las tiendas actualizan precios y los cambios son visibles inmediatamente para todos los compradores.

#### **Búsqueda inteligente**

Permite buscar productos por nombre con resultados relevantes usando consultas SQL con LIKE o ILIKE.

#### **Localización geográfica**

Almacena coordenadas de tiendas para mostrarlas en el mapa. El comprador ve qué tan cerca está cada tienda.

### **5.2 Problemas Técnicos**

#### **Seguridad de autenticación**

JWT garantiza que solo tiendas autenticadas pueden actualizar precios.

#### **Integridad de datos**

Foreign keys aseguran que no existan precios para productos o tiendas inexistentes.

#### **Validación de datos**

El backend valida que los precios sean números positivos, emails válidos, coordenadas correctas.

#### **Manejo de errores**

Middleware centralizado captura errores y retorna mensajes útiles al frontend.

## 6. Endpoints del MVP

### 6.1 Rutas Públicas

Método	Ruta	Descripción	Response
GET	/api/productos	Buscar productos	Array de productos con precios
GET	/api/tiendas	Listar tiendas	Array de tiendas con ubicaciones
POST	/api/auth/login	Iniciar sesión	{token, user}

### 6.2 Rutas Privadas - Tiendas

Método	Ruta	Descripción	Auth
GET	/api/tienda/productos	Ver productos de mi tienda	JWT tienda
PUT	/api/tienda/precios/:id	Actualizar precio	JWT tienda

### 6.3 Rutas Privadas - Admin

Método	Ruta	Descripción	Auth
POST	/api/admin/tiendas	Crear nueva tienda	JWT admin
POST	/api/admin usuarios	Crear usuario para tienda	JWT admin
POST	/api/admin/productos	Agregar producto al catálogo	JWT admin

## Descripción de Carpetas

Carpeta	Propósito
config/	Configuración de servicios: PostgreSQL, JWT
routes/	Definición de endpoints de la API
controllers/	Lógica de negocio de cada ruta
middlewares/	Funciones que procesan requests: authMiddleware verifica JWT
models/	Interacción con BD: queries SQL
database/	Scripts SQL para crear/modificar esquema

## 8. Referencias

### Documentación Oficial

#### Node.js

<https://nodejs.org/docs/>

#### Express.js

<https://expressjs.com/>

#### PostgreSQL

<https://www.postgresql.org/docs/>

#### JWT.io

<https://jwt.io/introduction>

#### Bcrypt.js

<https://github.com/kelektiv/node.bcrypt.js>

### Recursos Adicionales

- REST API Tutorial: <https://restfulapi.net/>
- OWASP API Security: <https://owasp.org/www-project-api-security/>
- Node.js Best Practices: <https://github.com/goldbergoni/nodebestpractices>
- Railway Deployment: <https://docs.railway.app/>