**Manual de Arquitectura - GastroVerse**

1. **Descripción General**

GastroVerse es una plataforma web pensada para personas interesadas en el mundo de la cocina. Permite buscar recetas, ver ingredientes, e incluso guardar recetas favoritas en una cuenta personal. Su público objetivo es todo aquel que quiera aprender a cocinar o ampliar su repertorio gastronómico. El siguiente documento busca exponer los procesos, funcionalidades y compones que denotan la estructuración de la plataforma implementada de forma integrada por completo.

1. **Tecnologías Utilizadas (Backend)**

* Lenguaje: JavaScript
* Entorno de ejecución: Node.js (v22.16.0)
* Framework web: Express.js
* Base de datos: PostgreSQL + Supabase
* Cliente SQL: pg
* Autenticación: JSON Web Tokens (JWT), bcrypt
* Dependencias principales:
  + - express, cors, dotenv, express-validator
    - jsonwebtoken, bcrypt, pg, axios
    - nodemon (desarrollo)
* Supabase CLI (instalado globalmente)

1. **Tecnologias Utilizadas( Frontend)**

* Librería Principal: React (v18.2.0)
* Lenguaje: JavaScript (ES6+)
* Entorno de Desarrollo: Vite
* Gestión de Rutas: React Router DOM (v6)
* Cliente HTTP: Axios
* Gestión de Estado: React Context API
* Estilos: CSS3 con Variables (Custom Properties) y metodología BEM
* Dependencias principales:
  + - react, react-dom
    - react-router-dom
    - axios
    - react-icons

1. **Arquitectura Implementada: MVC**

**Descripción del Patrón**

La arquitectura seleccionada para el backend es MVC (Modelo-Vista-Controlador), ya que permite una clara separación de responsabilidades, organización del código y escalabilidad. Esta división es ideal para equipos de trabajo o proyectos que evolucionan con el tiempo.

**Ventajas:**

* Facilitó el separar la lógica de negocio de las rutas y la base de datos.
* Ayudó a mantener el código legible y organizado.
* Hace más sencilla la reutilización de componentes.

1. **Estructura del Proyecto (Solo Backend)**

Raíz del proyecto:

├── backend/

│ ├── config/

│ │ └── database.js

│ ├── controllers/

│ │ ├── busquedaController.js

│ │ ├── favoritosController.js

│ │ ├── ingredientesController.js

│ │ ├── recetasController.js

│ │ └── usuariosController.js

│ ├── database/

│ │ ├── crearTablas.sql

│ │ ├── funciones.sql

│ │ ├── indicesRendimiento.sql

│ │ ├── insercionRecetas.sql

│ │ └── vistasUtiles.sql

│ ├── middlewares/

│ │ ├── authMiddleware.js

│ │ ├── manejoErroresValidacion.js

│ │ └── validaciones/

│ │ └── usuariosValidator.js

│ ├── models/

│ │ ├── busquedaTextoModel.js

│ │ ├── favoritosModel.js

│ │ ├── ingredientesModel.js

│ │ ├── recetasModel.js

│ │ ├── tokenBlacklistModel.js

│ │ ├── usuariosModel.js

│ │ └── vistas/

│ │ ├── recetasConIngredientesModel.js

│ │ └── recetasPopularesModel.js

│ ├── routes/

│ │ ├── busquedaRoutes.js

│ │ ├── favoritosRoutes.js

│ │ ├── ingredientesRoutes.js

│ │ ├── recetasRoutes.js

│ │ └── usuariosRoutes.js

│ ├── services/

│ │ ├── authService.js

│ │ ├── busquedaService.js

│ │ ├── favoritosService.js

│ │ ├── ingredientesService.js

│ │ ├── recetasService.js

│ │ └── usuariosService.js

└── frontend/ (a implementar)

└── shared/  
.env

node\_modules.js

app.js

package.json

package-lock.json

1. **Estructura del Proyecto (Solo Frontend)**

src/

└── frontend/

├── public/

│ └── vite.svg (y otros assets públicos)

├── src/

│ ├── assets/

│ │ └── logo.png

│ ├── components/

│ │ ├── common/

│ │ │ ├── FilterControls.jsx

│ │ │ ├── PopularRecipes.jsx

│ │ │ ├── ProtectedRoute.jsx

│ │ │ ├── RecipeCard.jsx

│ │ │ └── SearchBar.jsx

│ │ └── layout/

│ │ └── Navbar.jsx

│ ├── context/

│ │ └── AuthContext.jsx

│ ├── pages/

│ │ ├── CreateRecipePage.jsx

│ │ ├── EditIngredientsPage.jsx

│ │ ├── EditRecipePage.jsx

│ │ ├── FavoritesPage.jsx

│ │ ├── HomePage.jsx

│ │ ├── LoginPage.jsx

│ │ ├── ProfilePage.jsx

│ │ ├── RecipeDetailPage.jsx

│ │ └── RegisterPage.jsx

│ ├── services/

│ │ └── api.js

│ ├── App.css

│ ├── App.jsx

│ ├── index.css

│ └── main.jsx

├── .env

├── index.html

├── package.json

└── vite.config.js

1. **Clases, Objetos y Artefactos que Sustentan la Arquitectura (Backend)**

**Modelos (src/backend/models)**

* Encargados de acceder a la base de datos.
* Ejemplos: usuariosModel.js, recetasModel.js, favoritosModel.js
* Se conectan usando pg y ejecutan SQL puro (o funciones Supabase cuando aplica).

**Controladores (src/backend/controllers)**

* Manejan la solicitud entrante desde la ruta.
* Llaman a servicios o modelos según necesidad.
* Ejemplo: usuariosController.js maneja registro/login de usuarios.

**Rutas (src/backend/routes)**

* Definen los endpoints de la API REST.
* Conectan URL + verbo HTTP con un controlador.

**Servicios (src/backend/services)**

* Contienen lógica de negocio específica.
* Permiten que el controlador se mantenga limpio.
* Ejemplo: authService.js maneja emisión y validación de JWTs.

**Middlewares (src/backend/middlewares)**

* Interceptan peticiones para validación, autenticación y manejo de errores.
* authMiddleware.js protege rutas privadas.

1. **Componentes, Hooks y Artefactos que Sustentan la Arquitectura (Frontend)**

**Páginas (src/frontend/src/pages/)**

* Responsabilidad: Representan una vista completa de la aplicación asociada a una ruta específica (ej. la página de inicio, la página de perfil).
* Función: Orquestan la presentación de datos y la interacción del usuario, componiendo varios componentes más pequeños.
* Ejemplos: HomePage.jsx, LoginPage.jsx, RecipeDetailPage.jsx.

**Componentes (src/frontend/src/components/)**

* Responsabilidad: Son bloques de construcción de la UI, reutilizables y autocontenidos.
* Función: Encapsulan una parte específica de la interfaz y su lógica (ej. una tarjeta, un botón, un formulario). Se dividen en common (genéricos) y layout (estructurales).
* Ejemplos: Navbar.jsx, RecipeCard.jsx, SearchBar.jsx.

**Contextos (src/frontend/src/context/)**

* Responsabilidad: Gestionar y proveer estado global a través de todo el árbol de componentes.
* Función: Evitan el "prop drilling" al permitir que cualquier componente anidado acceda a datos compartidos, como la información del usuario autenticado.
* Ejemplo: AuthContext.jsx maneja el estado de autenticación, el token y los datos del usuario.

**Servicios (src/frontend/src/services/)**

* Responsabilidad: Centralizar y configurar la comunicación con APIs externas.
* Función: Contienen la configuración del cliente HTTP (Axios), incluyendo la URL base y los interceptores para añadir automáticamente tokens de autorización a las peticiones.
* Ejemplo: api.js exporta una instancia de Axios pre-configurada para interactuar con el backend de GastroVerse.

**Hooks Personalizados**

* Responsabilidad: Encapsular y reutilizar lógica de estado dentro de los componentes.
* Función: Simplifican el acceso al estado global. En lugar de usar useContext(AuthContext) directamente, se crea un hook más legible.
* Ejemplo: useAuth() es un hook personalizado dentro de AuthContext.jsx que proporciona una forma limpia de acceder al contexto de autenticación.

1. **Flujo de Comunicación MVC (Lógica que obedece al patrón)**
2. El usuario hace una solicitud HTTP (ej: GET /recetas).
3. La ruta correspondiente (recetasRoutes.js) redirige la solicitud al controlador.
4. El controlador (recetasController.js) valida los datos y delega la acción al servicio o modelo.
5. El modelo (recetasModel.js) ejecuta la consulta en PostgreSQL.
6. La respuesta regresa por el mismo camino hasta el cliente.

Este flujo claro permite identificar errores rápidamente y probar unidades de forma aislada.

1. **Seguridad y Autenticación**

* Se utiliza JWT para la autenticación.
* bcrypt cifra las contraseñas antes de guardarlas.
* authMiddleware.js protege rutas privadas comprobando el token.
* Se implementa una blacklist de tokens para cerrar sesión.
* dotenv mantiene las claves y configuraciones sensibles fuera del repositorio.

1. **Conclusiones del Equipo**

Implementar el patrón MVC nos permitió estructurar correctamente el proyecto desde sus primeras etapas. Aunque al inicio puede parecer más complejo, a medida que el sistema creció, la organización y claridad del código resultaron ser una gran ventaja. Recomendamos este enfoque a cualquier equipo que desee aplicar buenas prácticas desde la universidad.

Para futuras versiones del proyecto, consideramos que sería valioso:

* Incorporar Swagger para una mejor documentación de la API.
* Mejorar la separación de responsabilidades entre los servicios y los modelos.
* Implementar pruebas unitarias automatizadas.
* Integrar la lógica del frontend siguiendo una arquitectura modular coherente con el backend.