QuickBite

Documentación Frontend — Aplicación React

Componentes · Páginas · Rutas · Autenticación · Framework de diseño

1. Introducción

QuickBite es una aplicación web de delivery de comida desarrollada con React. El frontend consume una API REST construida con Spring Boot y se comunica con ella a través de un proxy configurado en Vite. La arquitectura del cliente sigue el patrón de componentes reutilizables con gestión de estado mediante Context API de React.

La aplicación diferencia dos tipos de usuarios:

- Usuario (USER): puede explorar restaurantes, ver sus productos y realizar pedidos.
- Administrador (ADMIN): dispone de un panel completo para gestionar todas las entidades del sistema.

2. Framework de diseño: Tailwind CSS

El framework de diseño utilizado es Tailwind CSS, un framework de utilidades CSS (utility-first CSS framework) que permite construir interfaces directamente en el marcado HTML/JSX mediante clases predefinidas, sin necesidad de escribir CSS personalizado en archivos separados.

2.1 Configuración

Tailwind está configurado a través de tres archivos en la raíz del proyecto:

- tailwind.config.js: define los directorios a escanear para purgar clases no usadas.
- postcss.config.js: integra Tailwind como plugin de PostCSS.
- index.css: importa las capas base, components y utilities de Tailwind.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

2.2 Clases personalizadas (index.css)

Además de las utilidades estándar, el proyecto define clases de componente reutilizables en la capa @layer components:

Clase	Descripción
.btn	Base para todos los botones: padding, border-radius, transición y focus ring
.btn-primary	Botón negro con texto blanco
.btn-secondary	Botón blanco con borde gris
.btn-danger	Botón rojo para acciones destructivas
.input	Campo de formulario con borde gris y focus ring negro
.card	Contenedor blanco con sombra y padding
.label	Etiqueta de formulario pequeña y semibold
.badge	Pastilla de estado redondeada (base)
.badge-success / danger / warning / info	Variantes de color para badges de estado

2.3 Paleta de colores y estética

El tema visual de la aplicación usa una paleta centrada en el naranja (#ff6b35 / orange-500/600 de Tailwind) como color de marca, combinado con grises neutros para fondos y textos. La Navbar y los botones principales usan gradientes naranja. La tipografía utilizada es Outfit, cargada desde Google Fonts en index.html.

---

3. Estructura del proyecto

La aplicación sigue la estructura estándar de un proyecto Vite + React:

```
src/
  main.jsx              → Punto de entrada
  App.jsx               → Router principal y layout
  index.css             → Estilos globales Tailwind
  context/
    AuthContext.jsx     → Contexto de autenticación
  components/
    Navbar.jsx          → Barra de navegación
    PrivateRoute.jsx    → Guardia de rutas
  pages/
    Home.jsx            → Landing page
    Login.jsx           → Inicio de sesión
    Register.jsx        → Registro
    Restaurantes.jsx    → Listado de restaurantes
    RestauranteDetalle.jsx → Menú y carrito
    RealizarPedido.jsx  → Confirmar pedido
    MisPedidos.jsx      → Historial de pedidos
    admin/
      Admin.jsx         → Dashboard admin
      AdminRestaurantes.jsx
      AdminProductos.jsx
      AdminCategorias.jsx
      AdminPedidos.jsx
      AdminRepartidores.jsx
      AdminClientes.jsx
```

```
services/
  api.js              → Llamadas a la API REST
```

---

4. Autenticación

La autenticación está implementada mediante JWT (JSON Web Token) en el backend y gestionada en el frontend con React Context API. El flujo es el siguiente:

4.1 Flujo de autenticación

- El usuario envía sus credenciales (username + password) al endpoint POST /api/auth/login.
- El backend valida las credenciales y devuelve un objeto AuthResponse con el token JWT, el username, el email, el rol y un mensaje.
- El frontend almacena este objeto en localStorage para persistir la sesión entre recargas.
- En cada petición posterior a la API, el token se incluye en la cabecera Authorization: Bearer {token}.
- Al hacer logout, el objeto se elimina de localStorage y el estado del contexto se limpia.

4.2 AuthContext (src/context/AuthContext.jsx)

Es el corazón de la gestión de autenticación en el frontend. Implementa el patrón Provider de React Context y expone las siguientes funciones y estados a todos los componentes de la aplicación:

Propiedad / Función	Descripción
user	Objeto con los datos del usuario autenticado (username, email, role, token)
isAuthenticated	Booleano: true si hay usuario en sesión
login(username, pwd)	Llama a la API, guarda el token y actualiza el estado
register(userData)	Registra al usuario y lo autentica automáticamente
logout()	Borra el token del localStorage y limpia el estado
isAdmin()	Retorna true si user.role === 'ADMIN'

4.3 PrivateRoute (src/components/PrivateRoute.jsx)

Componente guardia que protege las rutas. Recibe una prop adminOnly (por defecto false) y aplica las siguientes reglas:

- Si el usuario NO está autenticado → redirige a /login.
- Si la ruta es adminOnly y el usuario NO es ADMIN → redirige a /.
- Si pasa ambas comprobaciones → renderiza el componente hijo normalmente.

4.4 Redirección tras login

Tras autenticarse correctamente, el componente Login.jsx redirige al usuario según su rol:

- Rol ADMIN → navega a /admin (panel de administración).
- Rol USER → navega a /restaurantes (listado de restaurantes).

---

5. Rutas (App.jsx)

El enrutamiento está implementado con React Router v6. Las rutas se organizan en tres grupos según su nivel de acceso:

5.1 Rutas públicas

Accesibles sin autenticación:

Ruta	Componente	Descripción
/	Home	Landing page pública de la aplicación
/login	Login	Formulario de inicio de sesión
/register	Register	Formulario de registro de nuevos usuarios
*	Navigate → /	Ruta 404: redirige al inicio

5.2 Rutas protegidas (requieren autenticación)

Envueltas en el componente <PrivateRoute>. Redirigen a /login si no hay sesión activa:

Ruta	Componente	Descripción
/restaurantes	Restaurantes	Listado paginado de restaurantes disponibles
/restaurantes/:id	RestauranteDetalle	Menú del restaurante y carrito de compra
/realizar-pedido	RealizarPedido	Resumen y confirmación del pedido
/mis-pedidos	MisPedidos	Historial de pedidos del usuario

5.3 Rutas de administración (requieren rol ADMIN)

Envueltas en <PrivateRoute adminOnly>. Redirigen a / si el usuario no tiene rol ADMIN:

Ruta	Componente	Descripción
/admin	Admin	Dashboard con acceso a todas las secciones admin
/admin/restaurantes	AdminRestaurantes	CRUD completo de restaurantes
/admin/productos	AdminProductos	CRUD completo de productos
/admin/categorias	AdminCategorias	CRUD completo de categorías
/admin/pedidos	AdminPedidos	Gestión de pedidos y repartidores
/admin/repartidores	AdminRepartidores	CRUD completo de repartidores
/admin/clientes	AdminClientes	Gestión y búsqueda de clientes

---

6. Componentes

6.1 Componentes globales

Navbar (src/components/Navbar.jsx)

Barra de navegación fija en la parte superior (sticky top-0). Su contenido varía según el estado de autenticación:

- Sin sesión: muestra los enlaces "Iniciar sesión" y "Registrarse".
- Con sesión de usuario: muestra "Restaurantes" y "Mis Pedidos", el nombre del usuario y el botón "Salir".
- Con sesión de administrador: añade además el icono/enlace de acceso al panel de administración.

Hace uso del hook useAuth() para leer el estado del contexto.

PrivateRoute (src/components/PrivateRoute.jsx)

Componente wrapper que actúa como guardia de rutas. Comprueba isAuthenticated e isAdmin() del contexto antes de renderizar los componentes hijo. Devuelve un <Navigate> en caso de acceso no autorizado.

6.2 Páginas públicas

Home (src/pages/Home.jsx)

Landing page de la aplicación. Presenta la propuesta de valor de QuickBite con secciones hero, características y llamada a la acción. No requiere autenticación y es accesible desde la ruta raíz /.

Login (src/pages/Login.jsx)

Formulario de inicio de sesión con campos de username y contraseña. Al autenticarse correctamente, llama a login() del contexto y redirige al usuario según su rol (ADMIN → /admin, USER → /restaurantes). Muestra mensajes de error en caso de credenciales incorrectas.

Register (src/pages/Register.jsx)

Formulario de registro con los campos definidos en RegisterRequest.java del backend: username, email, password, nombre, apellidos y teléfono. Tras el registro exitoso, redirige automáticamente a /restaurantes. El backend crea automáticamente un perfil Cliente vinculado al nuevo usuario.

6.3 Páginas de usuario (protegidas)

Restaurantes (src/pages/Restaurantes.jsx)

Muestra un listado paginado de todos los restaurantes disponibles. Cada tarjeta muestra el nombre, descripción, dirección, teléfono e imagen del restaurante. Al hacer clic, navega a /restaurantes/:id.

RestauranteDetalle (src/pages/RestauranteDetalle.jsx)

Página central de la experiencia de usuario. Carga los datos de un restaurante concreto mediante su ID y muestra sus productos filtrados. Incluye:

- Cabecera con información del restaurante (nombre, descripción, dirección, teléfono).
- Filtro por categoría: botones que permiten ver solo los productos de una categoría concreta.
- Listado de productos con nombre, descripción, precio y stock disponible.
- Carrito de compra lateral (sticky) con controles de cantidad (+/-), subtotales por producto y total general.

- Botón "Realizar Pedido" que navega a /realizar-pedido pasando el carrito y el restaurante por estado.

Llama al endpoint GET /api/productos/filtrar?restauranteId={id}&disponible=true para cargar únicamente los productos del restaurante activo.

RealizarPedido (src/pages/RealizarPedido.jsx)

Página de confirmación del pedido. Recibe el carrito y los datos del restaurante mediante el estado de navegación de React Router. Permite al usuario introducir o confirmar la dirección de entrega y envía el pedido al endpoint POST /api/pedidos.

MisPedidos (src/pages/MisPedidos.jsx)

Historial de pedidos del usuario autenticado. Muestra una lista paginada de pedidos con información de fecha, estado, total y productos. Los estados posibles son: PENDIENTE, EN_PREPARACION, EN_CAMINO, ENTREGADO y CANCELADO, cada uno con su propio color de badge.

6.4 Páginas de administración (solo ADMIN)

Admin (src/pages/admin/Admin.jsx)

Dashboard principal del panel de administración. Muestra seis tarjetas de acceso rápido a cada sección de gestión, con iconos de react-icons y colores diferenciados.

AdminRestaurantes (src/pages/admin/AdminRestaurantes.jsx)

Gestión completa de restaurantes. Incluye tabla con todos los restaurantes, modal de creación/edición con los campos del formulario (nombre, descripción, dirección, teléfono, URL de imagen, estado activo) y confirmación de eliminación.

AdminProductos (src/pages/admin/AdminProductos.jsx)

Gestión completa de productos. La tabla muestra nombre, restaurante, categoría, precio, stock y estado. El modal de creación/edición incluye selects dinámicos para restaurante y categoría cargados desde la API.

AdminCategorias (src/pages/admin/AdminCategorias.jsx)

Gestión de categorías en vista de tarjetas. Permite crear, editar y eliminar categorías con sus campos nombre y descripción.

AdminPedidos (src/pages/admin/AdminPedidos.jsx)

Vista de todos los pedidos del sistema. Permite cambiar el estado de cada pedido mediante un selector desplegable y asignar manualmente un repartidor. Incluye modal de detalle con información completa del cliente, dirección de entrega y productos del pedido.

AdminRepartidores (src/pages/admin/AdminRepartidores.jsx)

Gestión de repartidores. La tabla incluye nombre, apellidos, email, teléfono, vehículo, matrícula y estado activo/inactivo. Permite activar o desactivar repartidores con un toggle y realizar CRUD completo.

AdminClientes (src/pages/admin/AdminClientes.jsx)

Gestión de clientes con buscador por nombre, email o teléfono. Incluye paginación, modal de detalle con datos de contacto y dirección completa, y opción de eliminar cliente.

---

7. Servicios API (src/services/api.js)

Toda la comunicación con el backend se centraliza en el archivo api.js. Utiliza Axios con una instancia configurada con la URL base /api (redirigida al backend mediante el proxy de Vite en vite.config.js). El interceptor de peticiones añade automáticamente la cabecera Authorization: Bearer {token} a todas las llamadas cuando el usuario está autenticado.

Los servicios exportados son:

Servicio	Operaciones disponibles
authService	login, register, logout, getCurrentUser
restauranteService	getAll, getById, create, update, delete
productoService	getAll, getById, filtrar, create, update, delete
categoriaService	getAll, getById, create, update, delete
pedidoService	getAll, getMisPedidos, getById, create, updateEstado, asignarRepartidor
repartidorService	getAll, getActivos, getById, create, update, cambiarEstado, delete
clienteService	getAll, getById, buscar, create, update, delete

8. Resumen

Aspecto	Tecnología / Solución
Framework frontend	React 18 con Vite
Enrutamiento	React Router v6
Framework de diseño	Tailwind CSS (utility-first)
Tipografía	Outfit (Google Fonts)
Iconos	react-icons (FaUser, FaCog, etc.)
Autenticación	JWT gestionado con React Context API
Persistencia de sesión	localStorage
Protección de rutas	Componente PrivateRoute
Comunicación API	Axios con interceptor JWT
Proxy API	Vite proxy /api → localhost:8080
Roles	USER (cliente) / ADMIN (panel completo)