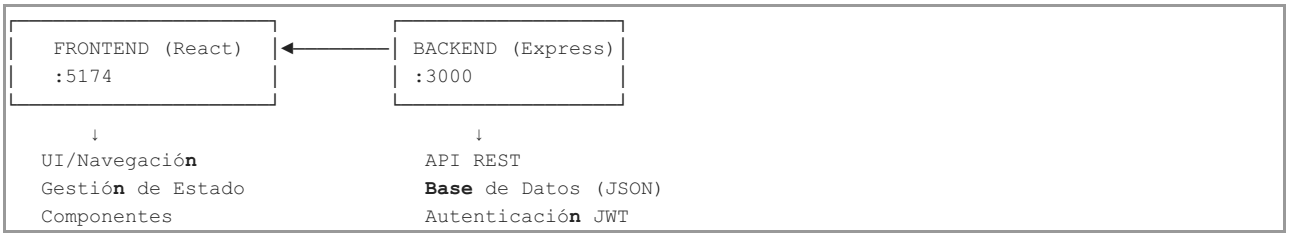
📐 Arquitectura del Proyecto - Menú Digital para Restaurante

🏛️ Visión General

Este es un proyecto **Full Stack** con arquitectura **Cliente-Servidor**:



🗂️ Estructura de Carpetas

BACKEND (`/backend`)

```
backend/
├── server.js                 # Punto de entrada. Configura Express, CORS, rutas
├── package.json              # Dependencias: express, cors, jwt, multer
├── .env                      # Variables de entorno (PORT, etc)
│
├── data/                     # Base de datos en archivos JSON
│   ├── users.json            # Usuarios registrados (admin, cliente)
│   ├── categories.json       # Categorías del menú (Cocina, Bar, etc)
│   └── products.json         # Productos con precios y descripciones
│
├── routes/                   # Rutas de la API
│   ├── auth.js               # POST /login, GET /me
│   ├── categories.js         # CRUD de categorías y subcategorías
│   └── products.js           # CRUD de productos
│
├── middleware/               # Funciones intermedias
│   └── auth.js               # Verificar JWT token, validar admin
│
├── uploads/                  # Carpeta donde se guardan imágenes subidas
└── images/assets/            # Imágenes estáticas del proyecto
```

FRONTEND (`/frontend/src`)

```
frontend/src/
├── main.jsx                # Entrada React, monta App en DOM
├── App.jsx                 # Componente raíz, rutas principales
├── index.css               # Estilos globales, variables CSS
│
├── pages/                  # Páginas (pantallas completas)
│   ├── Home.jsx            # Página inicial
│   ├── Menu.jsx            # Visualización del menú para clientes
│   ├── Login.jsx           # Formulario de autenticación
│   ├── AdminDashboard.jsx  # Panel de administración
│   └── QRGenerator.jsx     # Generador de códigos QR
│
├── components/             # Componentes reutilizables
│   ├── Header.jsx          # Cabecera con logo e icono
│   ├── Footer.jsx          # Pie de página
│   ├── ProductCard.jsx     # Tarjeta individual de producto
│   └── CategoryFilter.jsx  # Selector de categorías
│
├── context/                # Estado global
│   └── AuthContext.jsx     # Contexto de autenticación (usuario, login, logout)
│
├── services/               # Servicios de API
│   └── api.js              # Cliente Axios configurado, métodos GET/POST/PUT/DELETE
│
├── utils/                  # Funciones auxiliares
│   └── imageCompression.js # Redimensiona y comprime imágenes
│
└── assets/                 # Imágenes y archivos estáticos
    ├── logo.svg            # Icono de la app (tenedor + cuchara)
    ├── barra/              # Imágenes de bebidas
    └── cocina/             # Imágenes de platos
```

🔄 Flujo de Datos

1. AUTENTICACIÓN (Login)

```
Usuario escribe credenciales en Login.jsx
        ↓
[form onSubmit] → handleSubmit()
        ↓
api.post('/auth/login', {usuario, contraseña})  ← services/api.js
        ↓
BACKEND: routes/auth.js → Valida en data/users.json
        ↓
Si es correcto → Genera JWT token
        ↓
FRONTEND: Guarda token en localStorage
        ↓
AuthContext actualiza state (user, isAdmin)
        ↓
Redirige a /admin o /menu según rol
```

Archivos involucrados:

- Frontend: `Login.jsx`, `AuthContext.jsx`, `api.js`
- Backend: `routes/auth.js`, `middleware/auth.js`, `data/users.json`

---

2. VISUALIZAR MENÚ (Cliente)

```
Cliente abre /menu
        ↓
Menu.jsx: useEffect → Carga datos
        ↓
productsAPI.getAll() ← services/api.js
        ↓
GET /api/products → BACKEND routes/products.js
        ↓
Lee data/products.json
        ↓
Devuelve array de productos
        ↓
FRONTEND: useState(products) guarda datos
        ↓
Mapea productos → ProductCard.jsx por cada uno
        ↓
Renderiza grid de tarjetas con imagen, nombre, precio
```

Archivos involucrados:

- Frontend: `Menu.jsx`, `ProductCard.jsx`, `api.js`, `index.css` (grid-3)
- Backend: `routes/products.js`, `data/products.json`

---

3. AGREGAR PRODUCTO (Admin)

```
Admin abre AdminDashboard → /admin
        ↓
Valida autenticación (AuthContext.isAdmin())
        ↓
Admin rellena formulario y selecciona imagen
        ↓
handleProductImageChange() → Comprime imagen
        ↓
imageCompression.js:
  - Lee archivo con FileReader
  - Redimensiona en canvas (mantiene proporción)
  - Convierte a JPEG comprimido
        ↓
Admin hace click en "Guardar"
        ↓
handleSubmitProduct() forma FormData con:
  - name, description, price, categoryId
  - image (archivo comprimido)
        ↓
productsAPI.create(formData) ← services/api.js
        ↓
POST /api/products + JWT token en header
        ↓
BACKEND: Verifica token y role=admin (middleware auth.js)
        ↓
routes/products.js:
  - Multer maneja subida (guarda en /uploads)
  - Crea objeto newProduct
  - Agrega a products.json
        ↓
Responde con 201 + producto creado
        ↓
FRONTEND: Actualiza lista (loadData)
        ↓
Modal cierra, tabla se refresca
```

Archivos involucrados:

- Frontend: `AdminDashboard.jsx`, `imageCompression.js`, `api.js`
- Backend: `routes/products.js`, `middleware/auth.js`, `data/products.json`

---

4. EDITAR PRODUCTO (Admin)

```
Admin hace click en ícono editar
        ↓
openModal('product', productData)
        ↓
Formulario se llena con datos existentes
        ↓
Admin modifica y hace click "Guardar"
        ↓
handleSubmitProduct() verifica si existe formData.id
        ↓
Si existe: productsAPI.update(id, formData)
        ↓
PUT /api/products/:id
        ↓
BACKEND: Busca en products.json por ID
        ↓
Actualiza campos (merge con datos existentes)
        ↓
Guarda cambios en products.json
```

---

5. ELIMINAR PRODUCTO (Admin)

```
Admin hace click en icono basura
        ↓
handleDeleteProduct(id) → setConfirmDelete
        ↓
Modal de confirmación aparece
        ↓
Admin confirma
        ↓
confirmDeleteProduct()
        ↓
productsAPI.delete(id)
        ↓
DELETE /api/products/:id
        ↓
BACKEND: Filter out product con ese ID
        ↓
Guarda array actualizado en products.json
        ↓
FRONTEND: loadData() recarga lista
```

Sistema de Autenticación

JWT Token

```
1. User login → Backend genera JWT
   JWT = { userId, role, iat, exp }

2. Frontend guarda en localStorage

3. Cada solicitud incluye token:
   Headers: { Authorization: 'Bearer <token>' }

4. Backend valida en middleware/auth.js
   - Si token es válido → next()
   - Si token es inválido → 401 Unauthorized

5. Para rutas admin, también valida:
   if (user.role !== 'admin') → 403 Forbidden
```

Archivos clave:

- Backend: `middleware/auth.js`
- Frontend: `services/api.js` (interceptor añade token)
- Frontend: `context/AuthContext.jsx` (guarda en localStorage)

Gestión de Estado (Context API)

AuthContext.jsx

```
const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);        // Usuario actual
  const [loading, setLoading] = useState(true);  // Cargando autenticación

  // Métodos
  const login = async (credentials) => { ... }
  const logout = () => { ... }
  const isAdmin = () => user?.role === 'admin'
  const checkAuth = async () => { ... }  // Verifica token al cargar
}
```

Cómo se usa:

```
import { useAuth } from '../context/AuthContext';

function MyComponent() {
  const { user, login, logout, isAdmin } = useAuth();

  if (isAdmin()) {
    // Mostrar panel admin
  }
}
```

API REST Endpoints

Autenticación

```
POST   /api/auth/login          → { usuario, contraseña } → JWT token
GET    /api/auth/me             → Obtiene usuario actual (requiere token)
```

Categorías (todos requieren token, POST/PUT/DELETE solo admin)

```
GET    /api/categories          → Array de categorías
GET    /api/categories/:id      → Una categoría con subcategorías
POST   /api/categories          → Crear categoría (admin)
PUT    /api/categories/:id      → Actualizar categoría (admin)
DELETE /api/categories/:id      → Eliminar categoría (admin)
PUT    /api/categories/reorder  → Reordenar categorías (admin)
POST   /api/categories/:id/subcategories        → Agregar subcategoría
PUT    /api/categories/:id/subcategories/:subId → Actualizar subcategoría
DELETE /api/categories/:id/subcategories/:subId → Eliminar subcategoría
```

Productos (GET público, POST/PUT/DELETE solo admin)

```
GET    /api/products                    → Array de productos (filtrable)
GET    /api/products?categoryId=123     → Filtrar por categoría
GET    /api/products/:id                → Un producto específico
POST   /api/products                    → Crear producto (admin, multipart)
PUT    /api/products/:id                → Actualizar producto (admin)
DELETE /api/products/:id                → Eliminar producto (admin)
```

Componentes Principales

ProductCard.jsx

- **Props:** `{ product, isAdmin }`
- **Renderiza:** Tarjeta con imagen, nombre, descripción, precio
- **Estilos:** `ProductCard.css` (tamaño 300px mín)
- **Dinámico:** Si isAdmin=true, muestra botones editar/eliminar

Menu.jsx

- **Estado:** categories, products, selectedCategory, selectedSubcategory
- **Flujo:** Carga datos → Usuario selecciona categoría → Filtra productos → Mapea ProductCard
- **Tamaño grid:** grid-3 (300px mín por columna)

AdminDashboard.jsx

- **Tabs:** Products / Categories
- **Modal:** Para crear/editar
- **Imagen:** Compresión automática antes de subir
- **Drag & Drop:** Reordenar categorías (opcional)

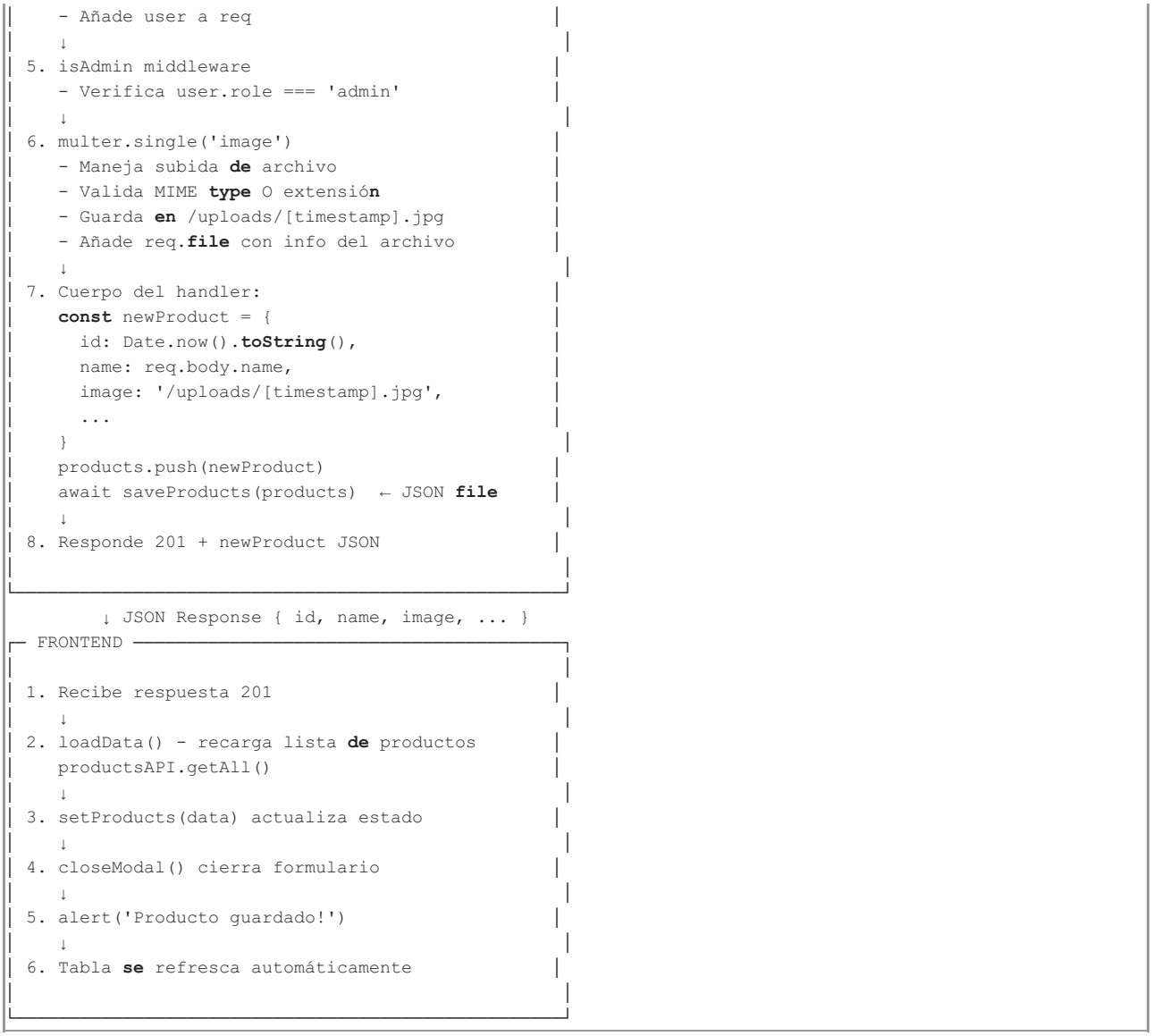
Header.jsx

- **Logo:** Icono SVG rotando (logo.svg)
- **Navegación:** Links a /menu, /admin, /qr
- **Responsive:** Menú hamburguesa en mobile

Ciclo de Vida de una Solicitud

Ejemplo: Crear Producto con Imagen

```
┌─ FRONTEND ──────────────────────────────────────┐
│                                                 │
│ 1. Usuario selecciona archivo imagen            │
│    ↓                                            │
│ 2. handleProductImageChange(event)              │
│    - Extrae file de input                       │
│    - Llama compressImage(file)                  │
│    ↓                                            │
│ 3. imageCompression.js                          │
│    - FileReader lee archivo                     │
│    - Canvas redimensiona (400x300 máx)          │
│    - Mantiene proporción                        │
│    - Comprime a JPEG 90%                        │
│    - Devuelve File comprimido                   │
│    ↓                                            │
│ 4. setFormData({ ...formData, imageFile })      │
│    ↓                                            │
│ 5. Usuario hace click "Guardar"                 │
│    ↓                                            │
│ 6. handleSubmitProduct(event)                   │
│    - Crea FormData                              │
│    - Agrega campos: name, description, etc      │
│    - Agrega imagen comprimida                   │
│    ↓                                            │
│ 7. productsAPI.create(formData)                 │
│    (services/api.js)                            │
│    - axios.post('/products', formData)          │
│    - Headers con JWT token                      │
│    - Content-Type: multipart/form-data          │
│                                                 │
└─────────────────────────────────────────────────┘
        ↓ HTTP POST /api/products
┌─ BACKEND ───────────────────────────────────────┐
│                                                 │
│ 1. Express recibe solicitud                     │
│    ↓                                            │
│ 2. Middleware CORS valida origen                │
│    ↓                                            │
│ 3. routes/products.js:                          │
│    POST / handler                               │
│    ↓                                            │
│ 4. authenticateToken middleware                 │
│    - Extrae token de header                     │
│    - Verifica JWT válido                        │
│                                                 │
```



🛠 Tecnologías por Capa

Frontend

- **React 18:** UI library
- **React Router:** Navegación entre páginas
- **Axios:** Cliente HTTP
- **Context API:** Gestión estado global
- **Vite:** Build tool (dev server rápido)
- **CSS Custom Properties:** Variables de diseño
- **FileReader API:** Lectura de archivos
- **Canvas API:** Redimensionamiento de imágenes

Backend

- **Express.js:** Framework web
- **Multer:** Manejo de subidas
- **jsonwebtoken:** Autenticación JWT
- **bcryptjs:** Hashing de contraseñas
- **CORS:** Política de origen cruzado
- **dotenv:** Variables de entorno
- **fs/promises:** Lectura/escritura JSON

Base de Datos

- **JSON Files:** Almacenamiento de datos (simple, sin BD)
    - `users.json`
    - `categories.json`
    - `products.json`

🔒 Seguridad

Validaciones

1. **Frontend:**
    - Validación HTML5 (required, type)
    - Validación de tipo de archivo imagen
    - Comprobación de usuario logueado
2. **Backend:**
    - Verificación JWT token
    - Validación role=admin
    - Multer: fileFilter (MIME type O extensión)
    - Validación de campos requeridos

Protección Rutas

```
Cliente: GET /menu      → Sin autenticación ✓
Cliente: POST /products → Error 401 Unauthorized
Admin:   POST /products → Verifica JWT + isAdmin ✓
```

📊 Flujo de Roles

```
┌─ USUARIO NO LOGUEADO ─┐
│ - Ver menú (lectura)  │
│ - Login / Registro    │
└───────────────────────┘

          ↓
      Ingresa credenciales
          ↓
┌─ CLIENTE ──────────────┐
│ - Ver menú (lectura)   │
│ - Ver detalles productos │
│ - No puede crear/editar  │
└──────────────────────┘

          ↓
   (datos: role='cliente')
          ↓
┌─ ADMINISTRADOR ──────────────┐
│ - Ver menú (lectura)         │
│ - CRUD productos             │
│ - CRUD categorías            │
│ - Ver/descargar código QR    │
│ - Acceso panel AdminDashboard │
└──────────────────────────────┘

   (datos: role='admin')
```

🚀 Inicio de la Aplicación

Secuencia de Carga

```
1. npm start (backend)
   → server.js configura Express
   → Carga rutas
   → Escucha en :3000

2. npm run dev (frontend)
   → Vite levanta dev server en :5174
   → main.jsx renderiza App

3. Usuario abre http://localhost:5174
   → App.jsx carga
   → AuthProvider envuelve todo
   → AuthContext ejecuta checkAuth()
   → Si hay token en localStorage:
     - Llama GET /api/auth/me
     - Actualiza state user
   → Header renderiza
   → Por defecto muestra Home

4. Usuario navega /menu
   → Menu.jsx se monta
   → useEffect llama productsAPI.getAll()
   → GET /api/products
   → Renderiza ProductCard por cada uno

5. Usuario hace login
   → Envía credenciales
   → Backend valida y genera JWT
   → localStorage guarda token
   → AuthContext actualiza user
   → Redirige a /admin o /menu
```

🎓 Resumen Conceptual

Este proyecto demuestra:

- ✅ **Full Stack:** Frontend React + Backend Express
- ✅ **REST API:** CRUD con JSON como BD
- ✅ **Autenticación JWT:** Protección de rutas
- ✅ **Roles:** Admin vs Cliente
- ✅ **Context API:** Estado global sin Redux
- ✅ **Manejo de Archivos:** Subida, compresión, almacenamiento
- ✅ **Responsive:** Mobile-first design
- ✅ **Componentes Reutilizables:** ProductCard, etc
- ✅ **Error Handling:** Try-catch en Frontend y Backend

¿Preguntas específicas sobre alguna parte? 🤔