Roadmap de Implementación — Sistema Restaurante (v1 → v1.1)

> Proyecto: Plataforma Web de Restaurantes con Gestión de Sucursales\ Stack: React + Vite + Tailwind v4 · Node.js + Express · MongoDB Atlas · JWT\ Arquitectura: Clean Architecture (capas domain / application / infrastructure / interfaces)

---

0) Preparación y cimientos (Día 0)

Objetivo: tener el entorno listo, repos preparado y un “Hello API/Hello UI” funcionando.

Tareas clave

- Crear repos `resto-sys` con subcarpetas `/frontend` y `/backend` (monorepo simple).
- Frontend: Vite + React + Tailwind v4 + React Router + Axios + Context + Mongo.
- Backend: Express + Mongoose + cors + dotenv + express-validator + bcrypt + jsonwebtoken.
- Conexión a MongoDB Atlas y variables de entorno (`.env` en backend y `VITE_API_URL` en frontend).
- Scripts útiles: `dev` , `build` , `preview` (FE) y `dev` (BE con nodemon).
- Ramas Git: `main` (estable), `develop` (integración), `feature/*` (unidades funcionales).
- **Smoke test:** `GET /api/health` responde 200 y el frontend muestra pantalla inicial.

Entregables

- Repositorio con estructura base + `.gitignore` + README con pasos de arranque.
- Endpoint `/api/health` y página `/` con “OK UI”.

DoD

- Levanta `FE:5173` y `BE:3000` local, FE consume `GET /api/health`.

---

1) Autenticación (v1.0 — Sprint 1)

Objetivo: registro/login de empresa y protección básica del panel admin.

Alcance

- **Modelos**: `Usuario/Empresa` (email, password hash, nombre_empresa, logo, sucursales[]).
- **Endpoints**: `POST /api/register` , `POST /api/login` , `GET /api/me` (token).
- **Seguridad**: JWT (Bearer), middleware `auth` , hashing con bcrypt, validaciones con express-validator.
- **UI**: `/admin/login` , `/admin/register` (opcional), guard de rutas privadas, almacenamiento del token.

Tests mínimos

- Backend: validación de email duplicado, login con credenciales válidas/invalidas, acceso a ruta protegida.
- Frontend: flujo completo login → acceso a dashboard; persistencia del token; logout.

Entregables

- Pantallas de Auth + contexto/estado de usuario + guardado de token.
- Colección de pruebas (Thunder Client/Postman) para auth.

DoD

- No es posible acceder a `/admin/*` sin token válido.

---

2) Gestión de Sucursales (v1.0 — Sprint 1)

Objetivo: CRUD de sucursales por empresa.

Alcance

- **Modelo**: `Sucursal` (nombre, ubicación, teléfono, empresa_id, configuracion {}).
- **Endpoints**: `GET/POST /api/sucursales`, `PUT/DELETE /api/sucursales/:id` (reglas: no borrar con reservas activas).
- **UI**: `/admin/sucursales` + componente `SucursalSelector` persistente (sucursal activa en estado global).

Reglas de negocio

- Solo ver/alterar sucursales de la empresa dueña del token.

Tests mínimos

- Crear/editar/borrar sucursal y visibilidad filtrada por empresa.

DoD

- Selector de sucursal actualiza el contexto global y persiste en `localStorage`.

---

3) Carta — Categorías y Platos (v1.0 — Sprint 2)

Objetivo: definir categorías y platos por sucursal, con disponibilidad y orden.

Alcance

- **Modelos**: `Categoria` (nombre, sucursal_id, orden), `Plato` (nombre, precio, descripción, imagen_url, disponible, categoria_id, sucursal_id).

- **Endpoints**: `GET/POST/PUT/DELETE /api/categorias`, `GET/POST/PUT/DELETE /api/platos` (filtrar por `sucursal_id`).
- **Reglas**: nombres de categoría únicos por sucursal; plato pertenece a una sola categoría y sucursal.
- **UI Admin**: `/admin/carta` con lista + `PlatoFormModal` y `CategoriaList`, filtros por categoría, toggle de disponibilidad.

Tests mínimos

- No permitir categorías duplicadas dentro de la misma sucursal.
- Listados correctos por sucursal y por categoría.

DoD

- CRUD completo operando; feedback con toasts; revalidación de cache con React Query.

---

4) Reservas (v1.0 — Sprint 2)

Objetivo: permitir crear reservas públicas y gestionarlas en admin.

Alcance

- **Modelo**: `Reserva` (sucursal_id, nombre_cliente, email, fecha, hora, personas, estado, mensaje?).
- **Endpoints**: `GET /api/reservas?sucursal_id=...` (admin), `POST /api/reservas` (público), `PUT/DELETE /api/reservas/:id` (cambiar estado: pendiente/confirmada/cancelada).
- **Reglas**: no permitir reservas en fechas pasadas; impedir borrar sucursales con reservas activas.
- **UI**: `/admin/reservas` (filtros por fecha/estado), `/[sucursal]/reserva` (form público con validaciones).

Tests mínimos

- Creación pública valida formatos, límites de fecha y sucursal existente.
- Cambio de estado desde admin con trazabilidad (updated_at).

DoD

- Flujo público → creación de reserva; admin puede confirmar/cancelar; feedback visible.

---

5) Carta Pública (v1.0 — Sprint 3)

Objetivo: publicar la carta por sucursal para clientes.

Alcance

- **Rutas públicas FE**: `/[sucursal]/carta` (lista categorías + platos), SEO básico.
- **UI**: `CartaItemCard`, filtros por categoría, estado disponible, precios.

- **Performance**: cache en React Query, skeleton loaders.

DoD

- Cualquier visitante puede navegar la carta pública de una sucursal válida.

---

6) Configuración Visual por Sucursal (v1.0 — Sprint 3)

Objetivo: personalizar la carta pública (color primario, mensaje de bienvenida, logo).

Alcance

- **Modelo**: `Sucursal.configuracion` (color_primario, mostrar_mensaje, mensaje_bienvenida, logo opcional).
- **Endpoints**: `PUT /api/sucursales/:id` (patch de configuración validado).
- **UI**: `/admin/configuracion` con previsualización live (Tailwind v4 CSS variables).

DoD

- Cambios guardados se reflejan en la carta pública en tiempo real (revalidación).

---

7) Seguridad y Endurecimiento (v1.0 hardening)

Objetivo: elevar el baseline de seguridad.

Alcance

- Validaciones exhaustivas (express-validator/Joi) y sanitización.
- Rate limiting por IP en endpoints públicos (reservas/login).
- CORS restringido por ambiente; headers seguros; manejo de errores centralizado.
- Logs de auditoría mínimos para Auth/Reservas (console + estructura para futuro LogRepo).

DoD

- Pruebas de abuso básicas superadas (inputs grandes, XSS simple, fuerza bruta rudimentaria mitigada).

---

8) Observabilidad y Métricas (v1.0+)

Objetivo: agregar trazas y métricas mínimas.

Alcance

- Middleware de request logging (método, ruta, latencia, status).
- Contadores: reservas creadas por día/sucursal; errores 4xx/5xx.
- Tab de métricas simple en admin (solo lectura, opcional en v1.0; recomendado v1.1).

DoD

- Archivo de logs (dev) y consola estructurada; panel simple si se incluye.

---

9) Deploy inicial (v1.0)

Objetivo: publicar MVP funcional.

Alcance

- **Backend**: Render/Railway con variables `.env` seguras; health check.
- **Frontend**: Vercel/Netlify apuntando a la API deployeada.
- **Base de datos**: MongoDB Atlas (usuario restringido, IPs permitidas, deshabilitar 0.0.0.0/0 en prod).
- **Seed**: script con empresa demo + sucursal + categorías/platos dummy.
- **Colección de API** (Postman/Thunder) versionada en repo.

DoD

- URL pública FE sirve carta y formulario de reservas; admin puede loguear y operar.

---

10) v1.1 — Mejoras y backlog

Objetivo: pulir UX, rendimiento y gobernanza.

Ideas priorizadas

- Paginación y búsqueda en lista de platos/reservas.
- Orden drag&drop de categorías y platos; campo `orden` persistente.
- Roles (multi-admin por empresa) y recuperación de contraseña.
- Envío de email al crear/confirmar reserva (hook + servicio externo, idempotente básico).
- Tests E2E livianos (Playwright) para flujos críticos (login, CRUD plato, crear reserva).
- Modo “carta QR”: URL corta por sucursal y diseño full móvil.

Tablas de referencia rápidas

Endpoints núcleo (v1)

- Auth: `POST /api/register`, `POST /api/login`, `GET /api/me`.
- Sucursales: `GET|POST /api/sucursales`, `PUT|DELETE /api/sucursales/:id`.
- Categorías: `GET|POST|PUT|DELETE /api/categorias` (query `sucursal_id`).
- Platos: `GET|POST|PUT|DELETE /api/platos` (query `sucursal_id`).
- Reservas: `GET /api/reservas?sucursal_id=...` (admin), `POST /api/reservas` (público), `PUT|DELETE /api/reservas/:id`.

Definiciones de hecho (DoD) comunes

- Validaciones y errores manejados; respuestas 2xx/4xx/5xx consistentes (JSON).
- Estado en FE con React Query: loaders, errores y revalidación tras mutaciones.
- Protecciones de ruta en FE y middlewares en BE.
- README de módulo actualizado + colección de pruebas sincronizada.

Estructura de ramas

- `main` (prod), `develop` (integración), `feature/{modulo}` (por ejemplo, `feature/auth`, `feature/sucursales`).

Guía de implementación capa a capa (resumen)

- **domain**: entidades puras y reglas (Reserva, Plato, etc.).
- **application**: casos de uso (RegisterUser, CreateBranch, CreateDish, CreateReservation…).
- **infrastructure**: repositorios (Mongoose), mailer futuro, cache.
- **interfaces**: controladores Express, validaciones HTTP, mappers request↔domain.

  Nota: aunque Express MVC clásico es válido, mantener esta separación facilitará testing y escalabilidad.

Comandos rápidos (recordatorio)

Frontend

```
cd frontend
npm install
npm run dev
```

Backend

```
cd backend
npm install
npm run dev
```
