Plan Detallado – Entrega 1 (LicitAgil + Cypress)

Pila: React (frontend) + Node.js (backend) + PostgreSQL + AWS (más adelante).
Testing: **Cypress** (E2E/UI).
Equipo tipo: 3 personas.

---

1) Objetivo y Alcance del MVP (Entrega 1)

Construir un **CRUD de Licitaciones** con búsqueda simple, vista de detalle y pruebas automatizadas con Cypress. Se incluye CI en GitHub Actions y documentación mínima obligatoria.

Fuera de alcance (para siguiente entrega): autenticación/roles, subidas reales de PDF, chat P&R, auditoría completa.

---

2) Backlog Inicial (Historias + Criterios de Aceptación)

H1 – Listar licitaciones
Como usuario quiero ver un listado de licitaciones para conocer las disponibles.
Criterios: - Dado que ingreso al home, entonces veo una tabla con columnas: Título, Estado, Fecha de cierre. - Si no hay registros, muestra mensaje “No hay licitaciones”. - La tabla pagina a 10 ítems.

H2 – Buscar licitación
Como usuario quiero buscar por título para encontrar rápidamente una licitación.
Criterios: - Input de búsqueda filtra por `titulo` (contiene, case-insensitive). - El filtro aplica tanto en frontend como en endpoint `GET /api/licitaciones?search=`.

H3 – Ver detalle
Como usuario quiero ver el detalle de una licitación para revisar su información.
Criterios: - `GET /api/licitaciones/:id` retorna `id, titulo, descripcion, estado, fecha_cierre, fecha_creacion`. - En la UI se muestran todos los campos.

H4 – Crear licitación
Como usuario quiero crear una licitación.
Criterios: - Form con validaciones: `titulo` requerido (3–120), `descripcion` requerido (10–10k), `estado` $\in$ {Abierta, En revisión, Cerrada}, `fecha_cierre` $\geq$ hoy. - `POST /api/licitaciones` persiste y redirige al detalle con toast “Creada con éxito”.

H5 – Editar licitación
Como usuario quiero editar una licitación existente.
Criterios: - `PUT /api/licitaciones/:id` actualiza y vuelve al detalle con toast “Actualizada”. - Validaciones iguales a crear.

H6 – Eliminar licitación
Como usuario quiero eliminar una licitación.

Criterios: - Confirmación modal. - `DELETE /api/licitaciones/:id` y la fila desaparece del listado + toast “Eliminada”.

Definición de Listo (DoD) por historia - UI funcional, validaciones, estados de carga/empty/error. - Pruebas Cypress E2E (feliz + borde principal). - Endpoint documentado en README/Wiki. - Sin errores de ESLint/Prettier.

---

3) Arquitectura MVP

Capas: - **Web (React + Vite)**: Router, React Query (o fetch simple), componentes UI con Tailwind o MUI. - **API (Node + Express)**: Endpoints REST, validación con `zod` / `express-validator`, logs `morgan`. - **DB (PostgreSQL)**: ORM recomendado: **Prisma** (alternativa: Sequelize).

Modelo de Datos (mínimo)

```
-- PostgreSQL (versión simple; si usas Prisma ver sección de migraciones)
CREATE TABLE licitaciones (
  id SERIAL PRIMARY KEY,
  titulo VARCHAR(120) NOT NULL,
  descripcion TEXT NOT NULL,
  estado VARCHAR(20) NOT NULL CHECK (estado IN ('Abierta','En
revisión','Cerrada')),
  fecha_creacion TIMESTAMP NOT NULL DEFAULT NOW(),
  fecha_cierre TIMESTAMP NOT NULL
);
CREATE INDEX idx_licitaciones_titulo ON licitaciones USING gin
(to_tsvector('spanish', titulo));
```

API (contratos) - `GET /api/licitaciones?search=&page=&pageSize=` → `{ items:[...], total }` - `GET /api/licitaciones/:id` → `{ id, titulo, descripcion, estado, fecha_creacion, fecha_cierre }` - `POST /api/licitaciones` → body `{ titulo, descripcion, estado, fecha_cierre }` - `PUT /api/licitaciones/:id` → body idem POST - `DELETE /api/licitaciones/:id` → `{ ok: true }`

---

4) Estructura de Repositorio y Setup

```
/licitagil
  /api            # Node/Express
    /src
      /routes
      /controllers
      /services
      /db
      /schemas
      app.ts
```

```
  prisma/       # si usas Prisma: schema.prisma, migrations
  package.json
/web            # React/Vite
  /src
    /components
    /pages
    /api        # fetchers
    /hooks
  index.html
  package.json
/docs           # assets para README/Wiki
docker-compose.yml
.github/workflows/ci.yml
README.md
```

Scripts sugeridos - Root: `"dev": "concurrently -n API,WEB -c auto \"npm --prefix api run dev\" \"npm --prefix web run dev\""` - API: `dev` (ts-node-dev/nodemon), `migrate`, `seed`, `start`. - Web: `dev`, `build`, `preview`.

Docker Compose (dev) - Servicio `db` (Postgres 14+), volúmenes, puerto 5432. - Variables `.env` (ver sección 9).

---

5) Frontend (React)

Páginas/Componentes - `/` Lista + barra de búsqueda + paginación. - `/licitaciones/:id` Detalle. - `/licitaciones/nueva` Form crear. - `/licitaciones/:id/editar` Form editar. - Componentes: `LicList`, `LicRow`, `LicForm`, `SearchBar`, `DeleteDialog`, `Toast`.

Estado de datos - React Query (o fetch + SWR). Manejar loading/empty/error.

UX/Accesibilidad - Botones con `aria-label`, formularios con labels reales. - Selectores para tests: `data-testid="lic-row-<id>"`, `data-testid="create-btn"`, etc.

---

6) Backend (Node.js + Express)

Puntos clave - Validación de entrada (`zod` / `express-validator`). - CORS (permitir `http://localhost:5173`). - Logs `morgan` en dev. - Rutas separadas y controladores delgados. - Servicio DB con Prisma/pg-promise.

Semillas - Script `seed` que cree 15 licitaciones con estados variados y fechas de cierre futuras.

---

7) Pruebas con Cypress (E2E)

Estrategia - Correr contra app real levantada en `localhost`. - Reset de datos: comando `npm run seed:test` antes de cada suite (o endpoint protegido solo en `NODE_ENV=test`: `POST /test/reset`).

Suites mínimas 1. **listado.cy.ts**
- Renderiza tabla con N filas iniciales. - Paginación funciona. 2. **busqueda.cy.ts**
- Escribe término, aparecen solo coincidencias. 3. **crear.cy.ts**
- Completa formulario válido → aparece en listado y en detalle; validaciones de requeridos. 4. **editar.cy.ts**
- Cambia `titulo` / `estado` → persiste. 5. **eliminar.cy.ts**
- Confirma modal → fila desaparece; estado del backend.

Buenas prácticas - Usar `data-testid` estables. - Evitar dependencias en layout frágil. - `cy.intercept` para esperar respuestas si hace falta.

---

8) CI en GitHub Actions (mínimo viable)

Objetivo: en cada PR/push a `main` / `develop` ejecutar: lint, build, migraciones + seed, levantar API y Web, correr Cypress en headless.

Plantilla `/.github/workflows/ci.yml` **(resumen)**

```
name: CI
on: [push, pull_request]
jobs:
  e2e:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:14
        env:
          POSTGRES_USER: postgres
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: licitagil_test
        ports: ["5432:5432"]
        options: >-
          --health-cmd "pg_isready -U postgres" --health-interval 10s --
health-timeout 5s --health-retries 5
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: { node-version: '20' }
      - run: npm ci --prefix api && npm ci --prefix web
      - run: npm run migrate:test --prefix api
      - run: npm run seed:test --prefix api
      - run: npm run build --prefix web
```

```
      - run: npm run start:ci --prefix api &
      - run: npx http-server web/dist -p 5173 &
      - run: npx wait-on http://localhost:3000/healthz http://localhost:5173
      - run: npx cypress run --config baseUrl=http://localhost:5173
```

Ajustar comandos a tus scripts reales. Usa `wait-on` para sincronizar.

---

9) Variables de Entorno (.env)

API

```
DATABASE_URL=postgres://postgres:postgres@localhost:5432/licitagil
PORT=3000
CORS_ORIGIN=http://localhost:5173
NODE_ENV=development
```

Web

```
VITE_API_URL=http://localhost:3000
```

Test: duplicar con DB `licitagil_test`.

---

10) Gestión del Proyecto (Jira + GitFlow)

Roles sugeridos (3 personas) - **Líder/BE**: arquitectura API/DB, CI. - **FE/UX**: páginas y componentes, accesibilidad, estilos. - **QA/E2E**: Cypress, datos de prueba, documentación.

GitFlow - Ramas: `main`, `develop`, `feature/<id-jira>-<slug>`. - PRs con plantilla: descripción, screenshots, checklist de pruebas.

Jira - Proyecto Kanban: *Backlog* → *In Progress* → *Code Review* → *Testing* → *Done* . - Historias H1–H6 + tareas técnicas (CI, seed, docs, video).

Definición de Done (global) - PR aprobado (≥1), CI verde, cobertura E2E básica, docs actualizadas.

---

11) Documentación y Video

README (raíz) - Resumen, requisitos, instalación (API/Web), variables de entorno, comandos dev/test/build. - Enlaces: Wiki, video, integrantes.

Wiki - Arquitectura, contratos de API, estrategia de pruebas, supuestos y dependencias.

Video (3–5 min) - 1) Qué resuelve LicitAgil. 2) Demo: CRUD + búsqueda. 3) Cypress corriendo. 4) CI mostrando pasadas. 5) Lecciones.

---

12) Cronograma sugerido (10 días hábiles)

Día 1: Repo, plantillas PR, ESLint/Prettier, Docker Postgres, Prisma/ORM, esquema `licitaciones`.
Día 2: API Express: rutas CRUD, validaciones, healthcheck, seed.
Día 3: Web scaffolding (Vite), routing, layout, listado básico.
Día 4: Búsqueda + paginación; estado de carga/empty/error.
Día 5: Form Crear (validaciones UI) → POST.
Día 6: Form Editar → PUT; Detalle completo.
Día 7: Eliminar con modal; toasts; pulir UX.
Día 8: Cypress suites (listado, búsqueda, crear).
Día 9: Cypress (editar, eliminar) + CI en GitHub Actions.
Día 10: README + Wiki + guion y grabación video; buffer.

Plan Express 48h (si aprieta el tiempo) - **Día 1**: API CRUD + Seed + Web (lista/detalle) + Crear.
- **Día 2**: Editar/Eliminar + Búsqueda + 3 suites Cypress + README básico.

---

13) Riesgos y Mitigaciones

- **Tiempo**: recortar estilos; priorizar flujo feliz.
- **Datos inestables en tests**: usar seed/reset por suite.
- **Flakiness Cypress**: `wait-on`, `cy.intercept`, selectores `data-testid`.
- **CI lento**: cache `~/.npm`, construir solo lo necesario, paralelizar si es posible.

---

14) Camino a Entrega 2 (preview)

- Autenticación y roles (Departamentos/Adquisiciones/Postulantes).
- Subida y validación básica de PDF (tipo y tamaño).
- Auditoría (tabla `eventos_auditoria`).
- Métricas y exportación CSV.

Checklist final de Entrega 1

- [ ] CRUD + búsqueda funcionando.
- [ ] Seeds y script de reset.
- [ ] 5 suites Cypress pasan local y en CI.
- [ ] README y Wiki completos.
- [ ] Video grabado y enlazado.