🧬 Metodología de Desarrollo del Sistema de Laboratorio Bioquímico con IA

📋 Documento de Trabajo en Equipo

🎯 OBJETIVO

Establecer una metodología clara para el desarrollo colaborativo del sistema de laboratorio bioquímico utilizando asistentes de IA (Claude, ChatGPT), manteniendo la consistencia y funcionalidad del código existente.

🏗️ ARQUITECTURA DEL PROYECTO

📁 Estructura de Carpetas

```
C:\Users\che-g\Desktop\BQ\
├── laboratorio-backend/
│   ├── src/
│   │   ├── controllers/     ← Lógica de negocio
│   │   ├── routes/          ← Definición de rutas
│   │   ├── services/        ← Servicios auxiliares (opcional)
│   │   └── index.ts         ← Servidor principal
│   ├── .env                 ← Variables de entorno
│   └── package.json
└── laboratorio-frontend/
    ├── src/
    │   ├── components/      ← Componentes reutilizables
    │   ├── pages/           ← Páginas/vistas principales
    │   ├── contexts/        ← Contextos de React
    │   ├── services/        ← Servicios de API
    │   ├── types/           ← Tipos de TypeScript
    │   ├── App.tsx          ← Configuración de rutas
    │   └── main.tsx         ← Punto de entrada
    └── package.json
```

🔌 Conexiones Clave

- **Base de Datos**: MySQL → `src/routes/db.ts`

- **API Endpoints**: `index.ts` → `routes/` → `controllers/`
- **Frontend Routes**: `App.tsx` → `pages/`
- **State Management**: `contexts/` → `pages/`

🚨 REGLAS CRÍTICAS (LEER ANTES DE CODIFICAR)

⚠️ REGLA #1: NUNCA RECORTAR CÓDIGO

```
// ❌ INCORRECTO - Recortar código existente
const getDashboard = async () => {
  // ... código recortado
}

// ✅ CORRECTO - Mantener TODO el código existente
const getDashboard = async (req: Request, res: Response) => {
  const id_medico = parseInt(req.params.id_medico);

  try {
    console.log('📊 Generando dashboard para médico ID:', id_medico);

    // ... TODO el código existente debe mantenerse
    // ... AGREGAR nuevas funcionalidades AL FINAL

  } catch (error: any) {
    console.error('💥 ERROR al generar dashboard:', error);
    return res.status(500).json({
      success: false,
      message: 'Error al obtener datos del dashboard'
    });
  }
};
```

⚠️ REGLA #2: NO MODIFICAR VARIABLES EXISTENTES

```
// ❌ INCORRECTO - Cambiar nombres de variables
const usuario = req.body; // Si ya existe, NO cambiar

// ✅ CORRECTO - Agregar nuevas variables
const usuario = req.body; // Mantener existente
const nuevaFuncionalidad = req.body.nuevoCampo; // Agregar nueva
```

⚠️ REGLA #3: PRESERVAR FUNCIONALIDADES

```
// ❌ INCORRECTO - Eliminar funcionalidades existentes
```

```
// const handleLogout = () => { ... } // Comentar o eliminar

// ✅ CORRECTO - Mantener y extender
const handleLogout = () => {
  // ... código existente COMPLETO
  localStorage.removeItem("usuario");
  navigate("/login");

  // ... AGREGAR nuevas funcionalidades aquí
};
```

📂 GUÍA DE UBICACIÓN DE ARCHIVOS

🎛️ BACKEND

📁 `/src/controllers/`

- **QUÉ VA**: Lógica de negocio, procesamiento de datos
- **EJEMPLO**: `medico.controller.ts`, `paciente.controller.ts`

```
// Estructura típica de un controlador
export const getFuncionNueva = async (req: Request, res: Response) => {
  try {
    // 1. Validaciones
    // 2. Consultas a BD
    // 3. Procesamiento
    // 4. Respuesta
  } catch (error) {
    // Manejo de errores
  }
};
```

📁 `/src/routes/`

- **QUÉ VA**: Definición de rutas, solo endpoints
- **EJEMPLO**: `medico.routes.ts`

```
import { getFuncionNueva } from '../controllers/medico.controller';

router.get('/nueva-funcionalidad', getFuncionNueva);
```

📁 `/src/index.ts`

- **QUÉ VA**: Configuración del servidor, registro de rutas

```
// Importar rutas
import medicoRoutes from './routes/medico.routes';

// Registrar rutas
app.use('/api/medico', medicoRoutes);
```

🎨 FRONTEND

📁 `/src/pages/`

- **QUÉ VA**: Componentes principales, vistas completas
- **EJEMPLO**: `MedicoPanel.tsx`, `PacienteDetalle.tsx`

📁 `/src/components/`

- **QUÉ VA**: Componentes reutilizables, UI elements
- **EJEMPLO**: `Button.tsx`, `Modal.tsx`

📁 `/src/App.tsx`

- **QUÉ VA**: Configuración de rutas de React Router

```
<Route path="/nueva-funcionalidad" element={<NuevaFuncionalidad />} />
```

🤖 PROMPTS ESPECÍFICOS PARA IA

📝 PROMPT PARA NUEVA FUNCIONALIDAD

Necesito implementar [DESCRIPCIÓN DE LA FUNCIONALIDAD] para el sistema de laboratorio bioquímico.

REQUISITOS CRÍTICOS:
1. ✅ NO recortar código existente - mantener TODAS las funciones previas
2. ✅ NO modificar variables existentes que ya funcionan
3. ✅ NO cambiar nombres de funciones existentes
4. ✅ Usar TypeScript en todo el código
5. ✅ Mantener el estilo de logging con emojis (console.log('📊 Mensaje'))
6. ✅ Preservar el formato de respuesta JSON existente
7. ✅ Usar la misma conexión a BD (pool de db.ts)

ESTRUCTURA DEL PROYECTO:
- Backend: Node.js + Express + TypeScript + MySQL
- Frontend: React + TypeScript + Tailwind CSS

- Conexión BD: src/routes/db.ts

ARCHIVOS A MODIFICAR:
- Controller: src/controllers/[MÓDULO].controller.ts
- Route: src/routes/[MÓDULO].routes.ts
- Frontend: src/pages/[MÓDULO]/[COMPONENTE].tsx
- App.tsx: Agregar nueva ruta

CÓDIGO EXISTENTE QUE DEBO MANTENER:
[PEGAR AQUÍ EL CÓDIGO EXISTENTE QUE NO DEBE MODIFICARSE]

Por favor, genera el código manteniendo absolutamente todo lo existente y agregando solo la nueva funcionalidad solicitada.

📝 PROMPT PARA CORRECCIÓN DE ERRORES

Tengo un error en el sistema de laboratorio bioquímico:

ERROR: [DESCRIPCIÓN DEL ERROR]

REGLAS PARA LA CORRECCIÓN:
1. ✅ NO eliminar código existente que funciona
2. ✅ NO modificar lógica existente sin afectar funcionalidad
3. ✅ Mantener compatibilidad con el código actual
4. ✅ Solo corregir lo específico del error
5. ✅ Preservar nombres de variables y funciones
6. ✅ Usar TypeScript

CÓDIGO ACTUAL:
[PEGAR EL CÓDIGO CON ERROR]

Por favor, corrige solo el error específico manteniendo toda la funcionalidad existente.

📝 PROMPT PARA INTEGRACIÓN DE COMPONENTES

Necesito integrar un nuevo componente en el sistema de laboratorio bioquímico.

COMPONENTE: [DESCRIPCIÓN]
UBICACIÓN: src/pages/[RUTA]/[COMPONENTE].tsx

REQUISITOS:
1. ✅ Mantener el diseño existente con Tailwind CSS
2. ✅ Usar las mismas interfaces TypeScript del proyecto
3. ✅ Conservar el estilo de navegación existente
4. ✅ Mantener la estructura de contextos (MedicoContext, etc.)
5. ✅ Preservar el manejo de estado existente
6. ✅ Usar axios para llamadas a API (como en otros componentes)

CONTEXTO EXISTENTE:
[PEGAR CÓDIGO DE CONTEXTO/ESTADO EXISTENTE]

DISEÑO BASE:
[PEGAR ESTRUCTURA DE COMPONENTE SIMILAR EXISTENTE]

Por favor, crea el componente manteniendo consistencia con el resto del sistema.

📊 WORKFLOW DE DESARROLLO

🔄 Proceso Paso a Paso

1. 📋 **PLANIFICACIÓN**
    - Definir funcionalidad específica
    - Identificar archivos a modificar
    - Revisar código existente relacionado
2. 🔍 **ANÁLISIS**
    - Copiar código existente que NO debe modificarse
    - Identificar interfaces/tipos necesarios
    - Verificar conexiones de BD necesarias
3. 🤖 **CONSULTA A IA**
    - Usar los prompts específicos de arriba
    - Incluir SIEMPRE el código existente
    - Especificar la regla de NO recortar código
4. ✅ **VALIDACIÓN**
    - Verificar que el código existente se mantiene
    - Probar funcionalidades previas
    - Validar nuevas funcionalidades
5. 🚀 **INTEGRACIÓN**
    - Actualizar rutas en App.tsx
    - Registrar rutas en index.ts
    - Probar integración completa

🎯 RECOMENDACIONES ESPECÍFICAS

💡 Para ChatGPT

Siempre incluir al final del prompt:
"Recuerda: Este es un proyecto real en producción. Es CRÍTICO mantener todo el código existente funcionando. No recortes, no elimines, no modifiques funcionalidades existentes. Solo agrega lo nuevo preservando lo anterior."

💡 Para Claude

Siempre incluir al final del prompt:
"IMPORTANTE: Sé absolutamente respetuoso del código existente. Mantén todas las funcionalidades previas intactas. Este es un sistema en uso real."

⚠️ Señales de Alerta

- Si la IA responde con "// ... código existente" → ❌ RECHAZAR
- Si elimina imports existentes → ❌ RECHAZAR
- Si cambia nombres de funciones → ❌ RECHAZAR
- Si modifica interfaces existentes → ❌ RECHAZAR

📱 EJEMPLOS PRÁCTICOS

✅ EJEMPLO CORRECTO - Agregar Funcionalidad

```
// ANTES - Código existente
const getDashboard = async (req: Request, res: Response) => {
  // ... 50 líneas de código existente
};

// DESPUÉS - Con nueva funcionalidad
const getDashboard = async (req: Request, res: Response) => {
  // ... LAS MISMAS 50 líneas de código existente SIN CAMBIOS

  // NUEVA funcionalidad agregada al final
  const nuevaFuncionalidad = await obtenerNuevosDatos();

  // Mantener la respuesta existente pero agregar nuevos datos
  return res.status(200).json({
    ...dashboardData, // Datos existentes
    nuevaFuncionalidad // Nuevos datos
  });
};
```

❌ EJEMPLO INCORRECTO - Recortar Código

```
// ANTES - Código existente
```

```
const getDashboard = async (req: Request, res: Response) => {
  // ... 50 líneas de código existente
};

// DESPUÉS - Código recortado (INCORRECTO)
const getDashboard = async (req: Request, res: Response) => {
  // ... código recortado o simplificado
  // ESTO ROMPE LA FUNCIONALIDAD EXISTENTE
};
```

🔧 HERRAMIENTAS Y CONFIGURACIÓN

🛠️ Extensiones Recomendadas (VS Code)

- TypeScript and JavaScript Language Features
- ES7+ React/Redux/React-Native snippets
- Tailwind CSS IntelliSense
- Auto Rename Tag

📦 Dependencias Clave

```
// Backend
{
  "express": "^4.18.0",
  "mysql2": "^3.6.0",
  "bcrypt": "^5.1.0",
  "cors": "^2.8.5",
  "dotenv": "^16.0.0"
}

// Frontend
{
  "react": "^18.2.0",
  "react-router-dom": "^6.8.0",
  "axios": "^1.3.0",
  "tailwindcss": "^3.2.0"
}
```

📞 CONTACTO Y SOPORTE

🤝 Coordinación del Equipo

- **Comunicar SIEMPRE** antes de modificar código existente

- **Compartir prompts** utilizados con la IA
- **Documentar cambios** realizados
- **Probar integración** antes de commit

🆘 Si Algo Sale Mal

1. **PARAR** inmediatamente
2. **DOCUMENTAR** el error específico
3. **CONTACTAR** al equipo
4. **NO INTENTAR** arreglos rápidos que puedan romper más código

---

📚 RECURSOS ADICIONALES

📖 Documentación Técnica

- Documentación de Express.js
- Documentación de React
- Documentación de TypeScript
- Documentación de Tailwind CSS

🔍 Debugging

- Usar `console.log()` con emojis para logging consistente
- Verificar Network tab en DevTools para errores de API
- Revisar terminal del backend para errores de servidor

✅ CHECKLIST ANTES DE ENVIAR CÓDIGO

- [ ] ✅ Todo el código existente se mantiene sin cambios
- [ ] ✅ Nuevas funcionalidades agregadas correctamente
- [ ] ✅ No se modificaron variables/funciones existentes
- [ ] ✅ TypeScript sin errores
- [ ] ✅ Rutas agregadas en App.tsx (si es necesario)
- [ ] ✅ Endpoints registrados en index.ts (si es necesario)
- [ ] ✅ Funcionalidades existentes probadas y funcionando
- [ ] ✅ Nuevas funcionalidades probadas
- [ ] ✅ Código documentado con comentarios
- [ ] ✅ Logging consistente con emojis

🚀 ¡Éxito en el desarrollo colaborativo!

Este documento debe ser leído completamente antes de comenzar cualquier desarrollo con IA.