Contents

1 Documentación Técnica - Prueba Técnica GLocation

Candidato: Junior Rodriguez
Fecha de entrega: 20 de Enero, 2025
Repositorio: https://github.com/JUNIORRDSR/Prueba-Tecnica
Aplicación desplegada: Docker Hub - `juniorrdsr/prueba-tecnica:latest`

1.1 Tabla de Contenidos

1.2 Resumen Ejecutivo

ProjectInsight es una aplicación fullstack para la gestión de proyectos que cumple con todos los requerimientos de la prueba técnica. La solución incluye:

- **API REST completa** con operaciones CRUD sobre la entidad `Proyecto`
- **Base de datos PostgreSQL** con Prisma ORM
- **Documentación Swagger** accesible en `/docs`
- **Integración con IA generativa** (DeepSeek API) para análisis de proyectos
- **Frontend responsivo** con visualizaciones en tiempo real (Chart.js)
- **Despliegue con Docker** (Dockerfile + docker-compose.yml)
- **Imagen publicada en Docker Hub** para facilitar el despliegue

El proyecto fue desarrollado siguiendo las mejores prácticas de desarrollo, con una arquitectura modular, manejo centralizado de errores, validaciones robustas y código limpio y documentado.

1.3 Stack Tecnológico

1.3.1 Backend

- **Node.js 20+** - Runtime de JavaScript
- **Express 5.1** - Framework web minimalista
- **Prisma 6.17** - ORM moderno para PostgreSQL
- **PostgreSQL 16** - Base de datos relacional
- **swagger-jsdoc + swagger-ui-express** - Documentación automática de API
- **dotenv** - Gestión de variables de entorno
- **cors** - Middleware para habilitar CORS

1.3.2 Frontend

- **EJS** - Motor de plantillas para SSR
- **Bootstrap 5** - Framework CSS para diseño responsivo
- **Chart.js** - Librería para gráficos interactivos
- **Axios** - Cliente HTTP para consumir la API

1.3.3 IA Generativa

- **DeepSeek API** - Modelo de IA para generación de resúmenes
- **OpenAI SDK** - Cliente para integración con APIs de IA
- **SSE (Server-Sent Events)** - Streaming de respuestas en tiempo real

1.3.4 DevOps

- **Docker** - Contenedorización de la aplicación
- **Docker Compose** - Orquestación de servicios
- **Docker Hub** - Registro público de imágenes

1.4 Arquitectura de la Solución

1.4.1 Estructura del Proyecto

```
Prueba-Tecnica/
  src/
     app.js                          # Punto de entrada de la aplicación
     swagger.js                      # Configuración de Swagger
     controllers/                    # Controladores de las rutas
        proyectos.controller.js
        analisis.controller.js
        graficos.controller.js
     services/                       # Lógica de negocio
        proyectos.service.js
        analisis.service.js
        graficos.service.js
     routes/                         # Definición de rutas
        proyectos.routes.js
        analisis.routes.js
        graficos.routes.js
     middlewares/                    # Middlewares personalizados
        errorHandler.js
     db/
        prismaClient.js             # Cliente de Prisma
     views/                          # Plantillas EJS
        index.ejs
     public/                         # Assets estáticos
         assets/
             css/
             js/
  prisma/
     schema.prisma                   # Esquema de base de datos
     migrations/                     # Migraciones de Prisma
  docker/
     entrypoint.sh                   # Script de inicio para Docker
  docs/
     screenshots/                    # Capturas de pantalla
  Dockerfile                         # Imagen Docker de la aplicación
  docker-compose.yml                 # Orquestación de servicios
  package.json                       # Dependencias del proyecto
  README.md                          # Documentación del usuario
```

1.4.2 Arquitectura de Capas

La aplicación sigue una arquitectura de tres capas:

```
          Frontend (EJS + JS)
  - Dashboard responsivo
```

```
  - Formularios de gestión
  - Gráficos con Chart.js

                  HTTP/REST

        Backend (Express)

     Routes → Controllers
           ↓
        Services
           ↓
      Prisma Client


                  Prisma ORM

       PostgreSQL Database
  - Tabla: Proyecto



       DeepSeek API
       (IA Externa)
```

1.4.3 Modelo de Datos

```
model Proyecto {
  id          Int       @id @default(autoincrement())
  nombre      String
  descripcion String
  estado      String    // "EN_PROGRESO", "FINALIZADO", "PENDIENTE"
  fechaInicio DateTime
  fechaFin    DateTime?
}
```

1.5 Decisiones Técnicas

1.5.1 1. Elección de Prisma como ORM

Justificación: - **Type-safety**: Prisma genera tipos automáticamente para TypeScript/JavaScript moderno - **Developer Experience**: Prisma Studio facilita la visualización y gestión de datos - **Migraciones**: Sistema de migraciones robusto y fácil de usar - **Performance**: Consultas optimizadas automáticamente

Alternativas consideradas: Sequelize, TypeORM

1.5.2 2. Arquitectura Modular (MVC)

Justificación: - **Separación de responsabilidades**: Controllers manejan HTTP, Services contienen lógica de negocio - **Mantenibilidad**: Código organizado y fácil de encontrar - **Testabilidad**: Cada capa puede ser probada independientemente - **Escalabilidad**: Fácil agregar nuevas features sin afectar código existente

1.5.3 3. DeepSeek para IA Generativa

Justificación: - **API gratuita**: Permite pruebas sin costo - **Compatible con OpenAI SDK**: Fácil integración con librerías existentes - **Streaming SSE**: Respuestas en tiempo real para mejor UX - **Fallback local**: Sistema funciona sin API Key usando resumen básico

1.5.4 4. Bootstrap 5 para Frontend

Justificación: - **Diseño responsivo**: Grid system y componentes adaptativos - **Componentes pre-construidos**: Modales, cards, navbars, etc. - **Dark mode**: Tema oscuro profesional sin CSS adicional - **Compatibilidad**: Funciona en todos los navegadores modernos

1.5.5 5. EJS para Server-Side Rendering

Justificación: - **Simplicidad**: Sintaxis familiar similar a HTML - **Performance**: Renderizado del servidor reduce carga en cliente - **SEO-friendly**: Contenido renderizado en servidor - **No requiere build**: Desarrollo más rápido

1.5.6 6. Docker para Despliegue

Justificación: - **Portabilidad**: Funciona en cualquier sistema con Docker - **Consistencia**: Mismo entorno en desarrollo, staging y producción - **Fácil setup**: Un solo comando para levantar todo - **Aislamiento**: Dependencias no afectan el sistema host

1.5.7 7. Swagger para Documentación

Justificación: - **Documentación automática**: Generada desde comentarios JSDoc - **Interactive**: Permite probar endpoints desde el navegador - **Estándar de industria**: OpenAPI 3.0 ampliamente adoptado - **Mantenibilidad**: La documentación vive junto al código

---

1.6 Implementación

1.6.1 1. API REST - CRUD Completo

1.6.1.1 Endpoints Implementados

Método	Endpoint	Descripción	Validaciones
GET	`/api/proyectos`	Listar todos los proyectos	-
POST	`/api/proyectos`	Crear un nuevo proyecto	nombre, estado, fechaInicio requeridos
GET	`/api/proyectos/:id`	Obtener un proyecto específico	ID válido

Método	Endpoint	Descripción	Validaciones
PUT	`/api/proyectos/:id`	Actualizar un proyecto	fechaFin >= fechaInicio
DELETE	`/api/proyectos/:id`	Eliminar un proyecto	ID válido
GET	`/api/graficos`	Datos agregados para gráficos	-
GET	`/api/analisis`	Resumen IA de proyectos	-

1.6.1.2 Validaciones Implementadas

```
// Estados válidos
const estadosValidos = ['EN_PROGRESO', 'FINALIZADO', 'PENDIENTE'];

// Validación de fechas
if (fechaFin && new Date(fechaFin) < new Date(fechaInicio)) {
  throw new Error('fechaFin no puede ser menor a fechaInicio');
}

// Validación de ID
function parseId(param) {
  const id = Number(param);
  if (Number.isNaN(id)) {
    throw new Error('ID inválido');
  }
  return id;
}
```

1.6.1.3 Manejo de Errores Centralizado

```
// middlewares/errorHandler.js
export const errorHandler = (err, req, res, next) => {
  const status = err.status || 500;
  const message = err.message || 'Error interno del servidor';

  console.error(`[ERROR] ${status} - ${message}`);

  res.status(status).json({
    error: true,
    message,
    ...(process.env.NODE_ENV !== 'production' && { stack: err.stack })
  });
};
```

1.6.2 2. Base de Datos con Prisma

1.6.2.1 Configuración

```
// src/db/prismaClient.js
import { PrismaClient } from '@prisma/client';
```

```
const prisma = new PrismaClient({
  log: ['query', 'error', 'warn'],
});

export default prisma;
```

1.6.2.2 Operaciones CRUD en Service Layer

```
// services/proyectos.service.js
import prisma from '../db/prismaClient.js';

export async function crearProyecto(data) {
  return await prisma.proyecto.create({ data });
}

export async function listarProyectos() {
  return await prisma.proyecto.findMany({
    orderBy: { id: 'desc' }
  });
}

export async function obtenerProyectoPorId(id) {
  return await prisma.proyecto.findUnique({
    where: { id }
  });
}

export async function actualizarProyecto(id, data) {
  return await prisma.proyecto.update({
    where: { id },
    data
  });
}

export async function eliminarProyecto(id) {
  return await prisma.proyecto.delete({
    where: { id }
  });
}
```

1.6.3 3. Documentación Swagger

La documentación se genera automáticamente usando JSDoc comments:

```
/**
 * @swagger
 * /api/proyectos:
 *   get:
```

```
 *     summary: Listar todos los proyectos
 *     tags: [Proyectos]
 *     responses:
 *       200:
 *         description: Lista de proyectos
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 $ref: '#/components/schemas/Proyecto'
 */
```

Accesible en: **http://localhost:3000/docs**

1.6.4 4. Integración con IA (DeepSeek)

1.6.4.1 Implementación con SSE

```
export async function generarAnalisis(req, res, next) {
  try {
    const proyectos = await proyectosService.listarProyectos();

    if (!process.env.DEEPSEEK_API_KEY) {
      // Fallback sin IA
      return res.json({
        resumen: `Análisis de ${proyectos.length} proyectos...`,
        proyectos: proyectos.length,
        // ... más datos locales
      });
    }

    // Streaming con SSE
    res.setHeader('Content-Type', 'text/event-stream');
    res.setHeader('Cache-Control', 'no-cache');
    res.setHeader('Connection', 'keep-alive');

    const stream = await openai.chat.completions.create({
      model: 'deepseek-chat',
      messages: [
        { role: 'system', content: 'Eres un analista de proyectos...' },
        { role: 'user', content: prompt }
      ],
      stream: true
    });

    for await (const chunk of stream) {
      const content = chunk.choices[0]?.delta?.content || '';
      if (content) {
```

```
        res.write(`data: ${JSON.stringify({ chunk: content })}\n\n`);
      }
    }

    res.write(`data: ${JSON.stringify({ done: true })}\n\n`);
    res.end();
  } catch (err) {
    next(err);
  }
}
```

1.6.5 5. Frontend Responsivo

1.6.5.1 Dashboard Interactivo El frontend incluye: - **KPIs**: Total de proyectos, en progreso, finalizados - **Tabla dinámica**: Listado de proyectos con acciones (editar/eliminar) - **Gráfico de torta**: Distribución de proyectos por estado (Chart.js) - **Panel IA**: Generación de resumen con streaming en tiempo real - **Formularios**: Modal para crear/editar proyectos con validación

1.6.5.2 Código del Frontend

```
// Consumo de API con Axios
async function cargarProyectos() {
  const { data } = await axios.get('/api/proyectos');
  renderizarTabla(data);
  actualizarKPIs(data);
}

// Generación de resumen con SSE
function generarResumenIA() {
  const eventSource = new EventSource('/api/analisis');
  let contenidoCompleto = '';

  eventSource.onmessage = (e) => {
    const data = JSON.parse(e.data);
    if (data.done) {
      eventSource.close();
    } else if (data.chunk) {
      contenidoCompleto += data.chunk;
      document.getElementById('analisisText').innerHTML =
        marked.parse(contenidoCompleto);
    }
  };
}

// Gráfico con Chart.js
async function cargarGrafico() {
  const { data } = await axios.get('/api/graficos');
```

```
  new Chart(ctx, {
    type: 'doughnut',
    data: {
      labels: data.labels,
      datasets: [{
        data: data.data,
        backgroundColor: ['#28a745', '#ffc107', '#dc3545']
      }]
    }
  });
}
```

1.7 Instalación y Ejecución

1.7.1 Opción 1: Ejecución Local

1.7.1.1 Requisitos Previos

- Node.js 20+
- PostgreSQL 14+
- npm 10+

1.7.1.2 Pasos

1. **Clonar el repositorio**

```
git clone https://github.com/JUNIORRDSR/Prueba-Tecnica.git
cd Prueba-Tecnica
```

2. **Configurar variables de entorno**

```
# Crear archivo .env
cat > .env << EOF
PORT=3000
DATABASE_URL="postgresql://postgres:postgres@localhost:5432/pruebas?schema=public"
# Opcional: DEEPSEEK_API_KEY=sk-xxxxx
EOF
```

3. **Instalar dependencias**

```
npm install
```

4. **Configurar base de datos**

```
# Ejecutar migraciones
npx prisma migrate deploy

# Generar cliente Prisma
npx prisma generate
```

5. **Iniciar la aplicación**

```
npm run dev
```

6. **Acceder a la aplicación**

- Dashboard: http://localhost:3000
- Swagger Docs: http://localhost:3000/docs
- Health Check: http://localhost:3000/health

1.7.2 Opción 2: Docker Compose (Recomendado)

```
# Levantar servicios (PostgreSQL + App)
docker compose up --build

# La aplicación estará disponible en http://localhost:3000
```

1.7.3 Opción 3: Docker Hub (Pre-compilado)

```
# Crear red
docker network create projectinsight-net

# Levantar PostgreSQL
docker run -d \
  --name projectinsight-db \
  --network projectinsight-net \
  --network-alias postgres \
  -e POSTGRES_DB=pruebas \
  -e POSTGRES_USER=postgres \
  -e POSTGRES_PASSWORD=postgres \
  -v projectinsight-data:/var/lib/postgresql/data \
  postgres:16-alpine

# Levantar aplicación desde Docker Hub
docker run -d \
  --name projectinsight-app \
  --network projectinsight-net \
  -p 3000:3000 \
  -e PORT=3000 \
  -e DATABASE_URL="postgresql://postgres:postgres@postgres:5432/pruebas" \
  juniorrdsr/prueba-tecnica:latest
```

1.8 API REST - Endpoints

1.8.1 1. Crear Proyecto

Request:

```
POST /api/proyectos
Content-Type: application/json
```

```
{
  "nombre": "Sistema de Inventario",
  "descripcion": "Sistema web para gestión de inventario en tiempo real",
  "estado": "EN_PROGRESO",
  "fechaInicio": "2025-01-10",
  "fechaFin": "2025-03-15"
}
```

Response:


```
{
  "id": 1,
  "nombre": "Sistema de Inventario",
  "descripcion": "Sistema web para gestión de inventario en tiempo real",
  "estado": "EN_PROGRESO",
  "fechaInicio": "2025-01-10T00:00:00.000Z",
  "fechaFin": "2025-03-15T00:00:00.000Z"
}
```


1.8.2 2. Listar Proyectos

Request:

```
GET /api/proyectos
```

Response:


```
[
  {
    "id": 1,
    "nombre": "Sistema de Inventario",
    "descripcion": "Sistema web para gestión de inventario en tiempo real",
    "estado": "EN_PROGRESO",
    "fechaInicio": "2025-01-10T00:00:00.000Z",
    "fechaFin": "2025-03-15T00:00:00.000Z"
  },
  {
    "id": 2,
    "nombre": "App Móvil de Ventas",
    "descripcion": "Aplicación móvil para registro de ventas",
    "estado": "FINALIZADO",
    "fechaInicio": "2024-11-01T00:00:00.000Z",
    "fechaFin": "2025-01-05T00:00:00.000Z"
  }
]
```


1.8.3 3. Obtener Proyecto por ID

Request:

```
GET /api/proyectos/1
```

Response:


```
{
  "id": 1,
  "nombre": "Sistema de Inventario",
  "descripcion": "Sistema web para gestión de inventario en tiempo real",
  "estado": "EN_PROGRESO",
  "fechaInicio": "2025-01-10T00:00:00.000Z",
  "fechaFin": "2025-03-15T00:00:00.000Z"
}
```


1.8.4 4. Actualizar Proyecto

Request:

```
PUT /api/proyectos/1
Content-Type: application/json

{
  "estado": "FINALIZADO",
  "fechaFin": "2025-02-20"
}
```

Response:


```
{
  "id": 1,
  "nombre": "Sistema de Inventario",
  "descripcion": "Sistema web para gestión de inventario en tiempo real",
  "estado": "FINALIZADO",
  "fechaInicio": "2025-01-10T00:00:00.000Z",
  "fechaFin": "2025-02-20T00:00:00.000Z"
}
```


1.8.5 5. Eliminar Proyecto

Request:

```
DELETE /api/proyectos/1
```

Response:

```
204 No Content
```

1.8.6 6. Datos para Gráficos

Request:

```
GET /api/graficos
```

Response:


```
{
  "labels": ["EN_PROGRESO", "FINALIZADO", "PENDIENTE"],
```

```
  "data": [5, 12, 3],
  "total": 20
}
```

1.8.7 7. Análisis con IA

Request:

```
GET /api/analisis
```

Response (con API Key - SSE):

```
data: {"chunk": "# Análisis de Proyectos\n\n"}
data: {"chunk": "## Resumen General\n"}
data: {"chunk": "Actualmente hay 20 proyectos..."}
data: {"done": true}
```

Response (sin API Key - JSON):

```
{
  "resumen": "Análisis de 20 proyectos registrados",
  "proyectos": 20,
  "enProgreso": 5,
  "finalizados": 12,
  "pendientes": 3,
  "estadisticas": {
    "promedioDescripcion": 85,
    "proyectoMasReciente": "Sistema de Inventario"
  }
}
```

1.9 Integración con IA

1.9.1 Funcionamiento

1. **Con API Key (DeepSeek)**:
   - El usuario hace clic en "Generar resumen"
   - Frontend abre conexión SSE a `/api/analisis`
   - Backend envía prompt a DeepSeek con descripción de proyectos
   - Respuesta se transmite en chunks en tiempo real
   - Frontend renderiza markdown progresivamente
2. **Sin API Key (Fallback)**:
   - Backend devuelve JSON con estadísticas calculadas localmente
   - Frontend muestra resumen básico sin IA

1.9.2 Prompt Utilizado

```
const prompt = `
Analiza los siguientes proyectos y genera un resumen ejecutivo:
```

```
${proyectos.map(p => `
- ${p.nombre}: ${p.descripcion}
  Estado: ${p.estado}
  Inicio: ${p.fechaInicio} | Fin: ${p.fechaFin}
`).join('\n')}

Proporciona:
1. Resumen general
2. Proyectos destacados
3. Recomendaciones
`;
```

1.9.3 Configuración

```
# Obtener API Key gratuita en: https://platform.deepseek.com
DEEPSEEK_API_KEY=sk-xxxxxxxxxxxxxxxx
```

1.10 Frontend Responsivo

1.10.1 Diseño Desktop

El dashboard incluye: - **Header**: Logo y título de la aplicación - **KPIs Row**: 3 cards con métricas principales - **Content Row**: - Columna izquierda (8 cols): Tabla de proyectos - Columna derecha (4 cols): Gráfico de distribución - **AI Panel**: Sección para generar y mostrar resumen IA - **Modales**: Crear/Editar proyecto

1.10.2 Diseño Móvil

- **Layout de 1 columna**: Todo el contenido apilado verticalmente
- **KPIs**: Cards apiladas
- **Tabla**: Scroll horizontal en dispositivos pequeños
- **Gráfico**: Tamaño reducido pero legible
- **Modales**: Formularios adaptados a pantalla pequeña

1.10.3 Breakpoints Utilizados

```
/* Bootstrap 5 breakpoints */
- xs: <576px (móviles)
- sm:  576px (móviles grandes)
- md:  768px (tablets)
- lg:  992px (desktop pequeño)
- xl:  1200px (desktop)
- xxl:  1400px (desktop grande)
```

1.10.4 Características Responsive

- Grid system de Bootstrap
- Cards adaptativas

- Tabla con scroll horizontal
- Formularios con validación
- Modales fullscreen en móviles
- Gráficos responsivos con Chart.js

1.11 Docker y Despliegue

1.11.1 Dockerfile

```
FROM node:20-alpine
WORKDIR /app

# Instalar dependencias
COPY package*.json ./
RUN npm ci

# Instalar netcat para healthcheck
RUN apk add --no-cache netcat-openbsd

# Copiar código fuente
COPY prisma ./prisma
COPY src ./src
COPY docker/entrypoint.sh ./entrypoint.sh

# Generar cliente Prisma
ARG DATABASE_URL=postgresql://postgres:postgres@postgres:5432/pruebas
ENV DATABASE_URL=$DATABASE_URL
RUN npx prisma generate

# Configuración
ENV NODE_ENV=production
ENV PORT=3000
RUN chmod +x /app/entrypoint.sh

EXPOSE 3000
ENTRYPOINT ["/app/entrypoint.sh"]
```

1.11.2 Entrypoint Script

```
#!/bin/sh
set -e

# Esperar a que PostgreSQL esté listo
until nc -z -v -w30 postgres 5432; do
  echo "Esperando PostgreSQL..."
  sleep 1
done
```

```
# Ejecutar migraciones
echo "Ejecutando migraciones..."
npx prisma migrate deploy

# Iniciar aplicación
echo "Iniciando aplicación..."
exec node src/app.js
```

1.11.3 Docker Compose

```
services:
  postgres:
    image: postgres:16-alpine
    environment:
      POSTGRES_DB: pruebas
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - "5432:5432"
    volumes:
      - postgres-data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5

  app:
    build: .
    environment:
      PORT: 3000
      DATABASE_URL: postgresql://postgres:postgres@postgres:5432/pruebas
      DEEPSEEK_API_KEY: ${DEEPSEEK_API_KEY:-}
    ports:
      - "3000:3000"
    depends_on:
      postgres:
        condition: service_healthy

volumes:
  postgres-data:
```

1.11.4 Publicación en Docker Hub

```
# Build
docker build -t juniorrdsr/prueba-tecnica:latest .
```

```
# Login
docker login

# Push
docker push juniorrdsr/prueba-tecnica:latest
```

1.12 Evidencias Visuales

1.12.1 Dashboard Desktop

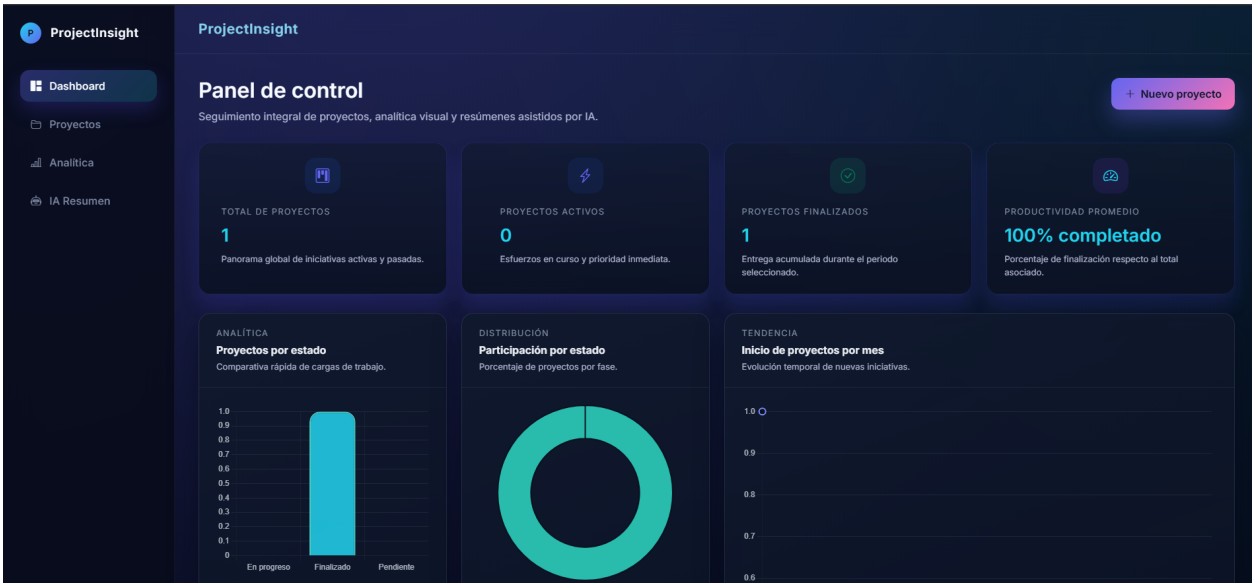

Figure 1: Vista principal del dashboard

Características visibles: - KPIs con métricas en tiempo real - Tabla de proyectos con acciones - Gráfico de distribución por estado - Panel de análisis con IA - Interfaz oscura profesional

1.12.2 Documentación Swagger

Características visibles: - Todos los endpoints documentados - Esquemas de datos definidos - Interfaz interactiva para probar API - Ejemplos de request/response

1.12.3 Modal de Creación

Características visibles: - Validación de campos - Selector de estado - Date pickers para fechas - Diseño consistente con el dashboard

1.12.4 Vista Móvil (Samsung S20 Ultra)

Características visibles: - Layout de 1 columna adaptativo - KPIs apilados - Tabla con scroll horizontal - Gráfico reducido pero legible - Navegación optimizada para táctil

---

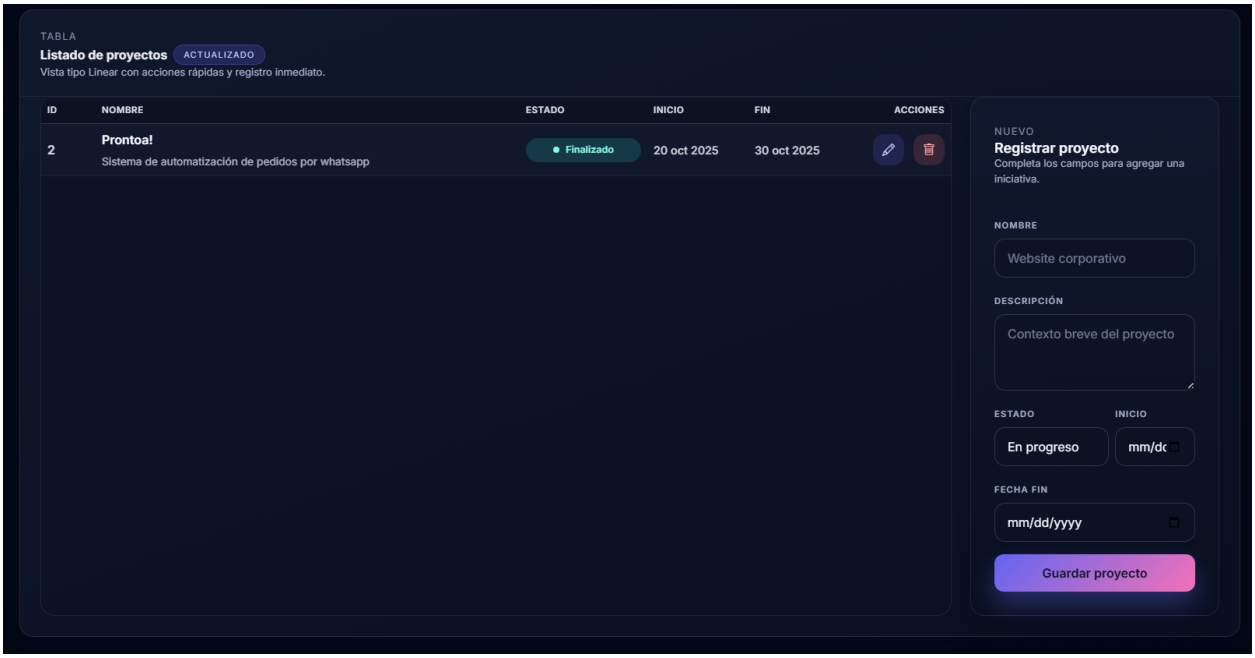

Figure 2: Swagger UI

Figure 3: Formulario de proyecto

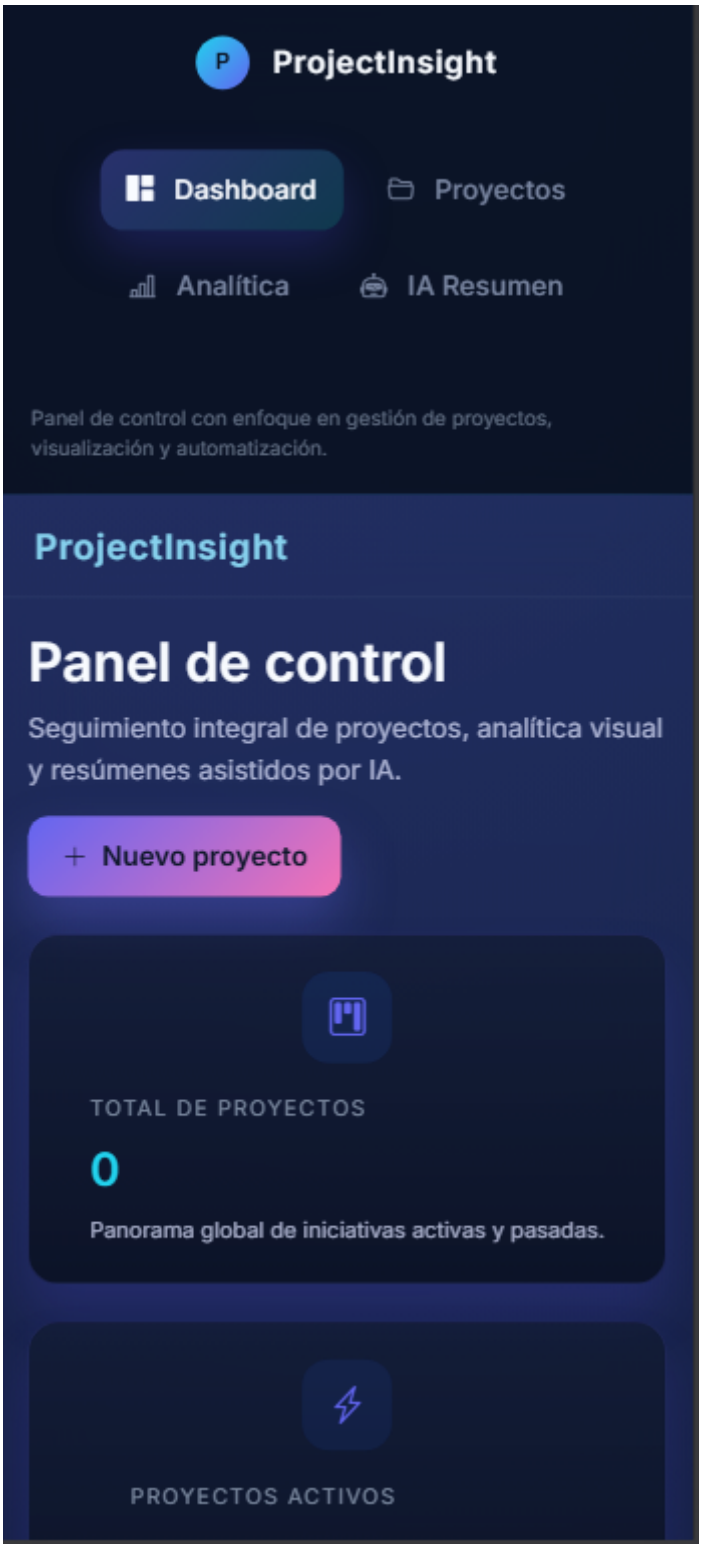

Figure 4: Vista móvil

1.13 Resultados y Cumplimiento

1.13.1 Checklist de Requerimientos

Requerimiento	Estado	Evidencia
API REST con CRUD	[X] Completo	5 endpoints + validaciones
PostgreSQL con ORM	[X] Completo	Prisma + Migraciones
Documentación Swagger	[X] Completo	`/docs` con OpenAPI 3.0
Integración IA	[X] Completo	DeepSeek + fallback
Frontend responsivo	[X] Completo	Desktop + Móvil
Tablas y gráficos	[X] Completo	Chart.js + tabla dinámica
Docker	[X] Plus	Dockerfile + Compose + Hub
Documentación completa	[X] Completo	README + Este documento

1.13.2 Plus Implementados

- ☒ **Docker completo**: Dockerfile, docker-compose.yml, imagen en Docker Hub
- ☒ **Arquitectura modular**: Separación clara de capas (controllers, services, routes)
- ☒ **Manejo de errores**: Middleware centralizado con logging
- ☒ **Validaciones robustas**: En todas las operaciones CRUD
- ☒ **SSE para IA**: Streaming en tiempo real de respuestas
- ☒ **Fallback sin IA**: Sistema funciona sin API Key
- ☒ **Dark theme**: Interfaz profesional con Bootstrap
- ☒ **Healthcheck**: Endpoint de monitoreo

1.14 Conclusiones

1.14.1 Logros Técnicos

1. **Arquitectura sólida**: El proyecto sigue patrones de diseño reconocidos (MVC, separación de capas)
2. **Código limpio**: Código legible, comentado y bien estructurado
3. **Escalabilidad**: Fácil agregar nuevas funcionalidades sin romper código existente
4. **Experiencia de usuario**: Frontend intuitivo y responsivo
5. **DevOps**: Despliegue simplificado con Docker

1.14.2 Decisiones Destacadas

1. **Prisma ORM**: Mejor DX que Sequelize, con type-safety
2. **SSE para IA**: Mejor UX que esperar respuesta completa
3. **Bootstrap 5**: Desarrollo rápido con diseño profesional
4. **Swagger**: Documentación viva y actualizada
5. **Docker Hub**: Facilita testing sin compilar localmente

1.14.3 Aprendizajes

1. **Integración con APIs de IA**: Manejo de streaming y fallbacks

2. **Server-Sent Events**: Implementación de comunicación unidireccional en tiempo real
3. **Prisma avanzado**: Migraciones, relaciones, validaciones
4. **Docker multi-stage**: Optimización de imágenes
5. **Frontend vanilla**: Desarrollo sin frameworks pesados

1.14.4 Tiempo de Desarrollo

- **Backend (API + DB)**: ~4 horas
- **Frontend (UI)**: ~3 horas
- **Integración IA**: ~2 horas
- **Docker**: ~1 hora
- **Documentación**: ~2 horas
- **Total**: ~12 horas

1.14.5 Posibles Mejoras Futuras

1. **Testing**: Agregar pruebas unitarias y de integración (Jest, Supertest)
2. **Autenticación**: Implementar JWT para proteger endpoints
3. **Paginación**: Para listado de proyectos cuando crezca la base de datos
4. **WebSockets**: Para actualización en tiempo real de múltiples clientes
5. **CI/CD**: Pipeline automático con GitHub Actions
6. **Monitoring**: Logs centralizados y métricas (Prometheus, Grafana)

1.15 Contacto

Junior Rodriguez
Email: [correo del candidato]
GitHub: https://github.com/JUNIORRDSR
LinkedIn: [perfil de LinkedIn]

1.16 Licencia

Este proyecto fue desarrollado como parte de una prueba técnica para GLocation.

Nota Final: Este documento evidencia la implementación completa de todos los requerimientos solicitados en la prueba técnica. El código está disponible en el repositorio de GitHub y la aplicación puede ser desplegada fácilmente usando Docker. Estoy disponible para una sesión técnica donde pueda demostrar el funcionamiento del sistema en localhost y responder cualquier pregunta sobre la implementación.