Solución Prueba Técnica – Líder Técnico de Desarrollo (ASISYA)

Arquitectura · Diseño · Implementación · Entregables

Angelica Duarte
2025

Objetivo de la Prueba

- Diseñar solución escalable para optimización de proveedores.
- Implementar microservicio crítico (.NET 8).
- Implementar frontend en React.
- Integrar base de datos PostgreSQL.
- Contenerizar con Docker.
- Documentar estándares técnicos y arquitectura.

Stack Tecnológico

BACKEND: .NET 8 (MINIMAL API)

BD: POSTGRESQL 14

FRONTEND: REACT + AXIOS

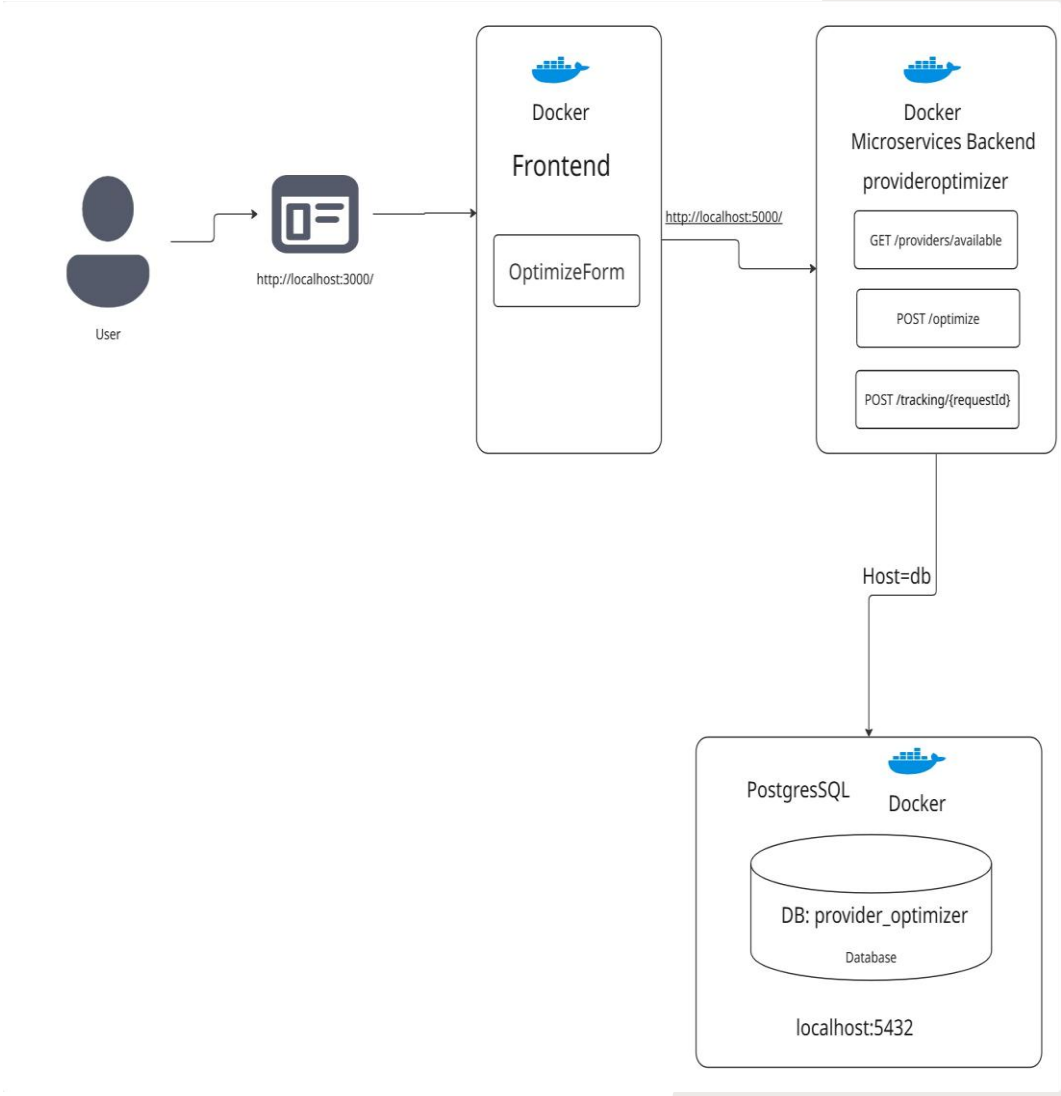
REALTIME: SIGNALR

INFRAESTRUCTURA: DOCKER, DOCKER COMPOSE

CONTROL DE VERSIÓN: GITHUB

Arquitectura General

- Microservicio Backend (.NET 8).
- Frontend en React + Nginx.
- PostgreSQL como base de datos relacional.
- Comunicación en tiempo real con SignalR.
- Infraestructura reproducible con Docker Compose.



Diseño del Microservicio

- Endpoints:
  - GET /providers/available
  - POST /optimize
  - GET /tracking/{requestId}
- Patrón Repository.
  - Servicio de optimización en capa Aplicacion
  - Uso de fórmula Haversine para calcular distancias mas cortas.

ProviderOptimizerService.cs

src > ProviderOptimizer.Application > Services > ProviderOptimizerService.cs

```
18  public async Task<Provider?> OptimizeAsync(OptimizeRequestDto request)
19  {
20      var providers = (await _repo.GetAvailableProvidersAsync()).ToList();
21
22      if (!providers.Any())
23          throw new Exception("No providers available");
24
25      var scored = providers
26          .Select(p =>
27          {
28              var distance = Haversine(request.Latitude, request.Longitude, p.Latitude, p.Longitude);
29              var score = ComputeScore(distance, p.Rating);
30              return new { Provider = p, Score = score };
31          })
32          .OrderByDescending(x => x.Score)
33          .ToList();
34
35      return scored.First().Provider;
36  }
```

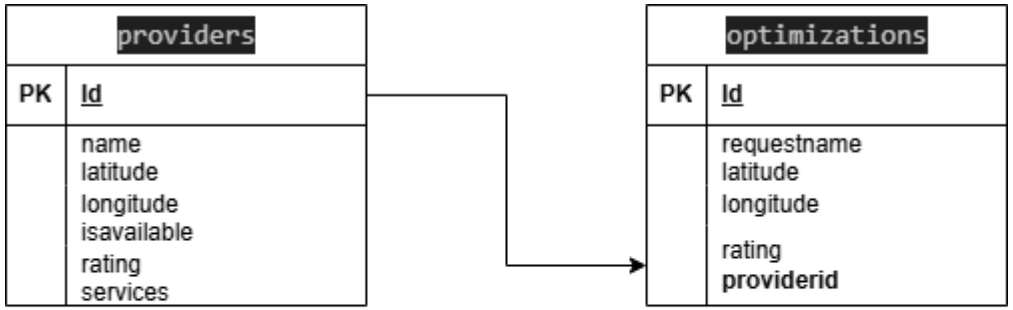
ProviderOptimizerService.cs

src > ProviderOptimizer.Application > Services > ProviderOptimizerService.cs

```
55
56      private static double ComputeScore(double distanceKm, double rating)
57      {
58          double nd = Math.Max(0, 1 - (distanceKm / 100.0));
59          double nr = Math.Clamp(rating / 5.0, 0.0, 1.0);
60          return nr * 0.4 + nd * 0.6;
61      }
62
63      private static double Haversine(double lat1, double lon1, double lat2, double lon2)
64      {
65          const double R = 6371;
66          double dLat = ToRad(lat2 - lat1);
67          double dLon = ToRad(lon2 - lon1);
68          double a = Math.Sin(dLat / 2) * Math.Sin(dLat / 2) +
69                     Math.Cos(ToRad(lat1)) * Math.Cos(ToRad(lat2)) *
70                     Math.Sin(dLon / 2) * Math.Sin(dLon / 2);
71
72          double c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
73          return R * c;
74      }
75
76      private static double ToRad(double deg) => deg * (Math.PI / 180.0);
77  }
```

Base de Datos

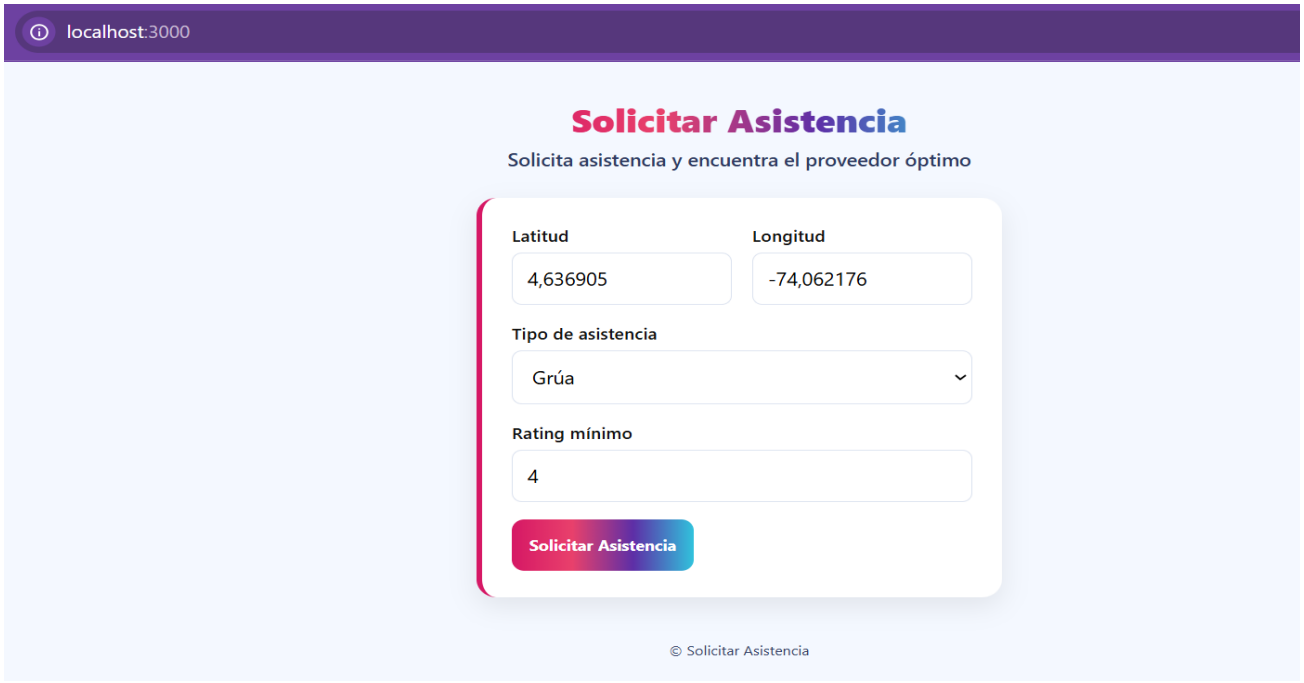
- Tabla principal: providers
  - id
  - name
  - latitude, longitude
  - service_type
  - rating



- Tabla principal: optimizations
  - id
  - name
  - latitude, longitude
  - requestname
  - rating
  - providerid

Frontend React

- Formulario de solicitud.
- Integración con backend vía Axios.
- UI moderna basada en diseño corporativo.
- Panel de estado en vivo con SignalR.



Seguimiento en Tiempo Real

- Implementado con SignalR.
- Canal de comunicación: /trackHub
- El backend emite estados de la solicitud:
  - 📡 Solicitud recibida
  - 📡 Validando ubicación
  - 📡 Buscando proveedor disponible
  - 📡 Proveedor encontrado
  - 📡 Finalizando solicitud
- El frontend escucha eventos en vivo.

Flujo Final:

React App → POST /optimize

↓

ASP.NET Core genera trackingId

↓

SignalR TrackingHub → envía eventos

↓

React muestra progreso en tiempo real

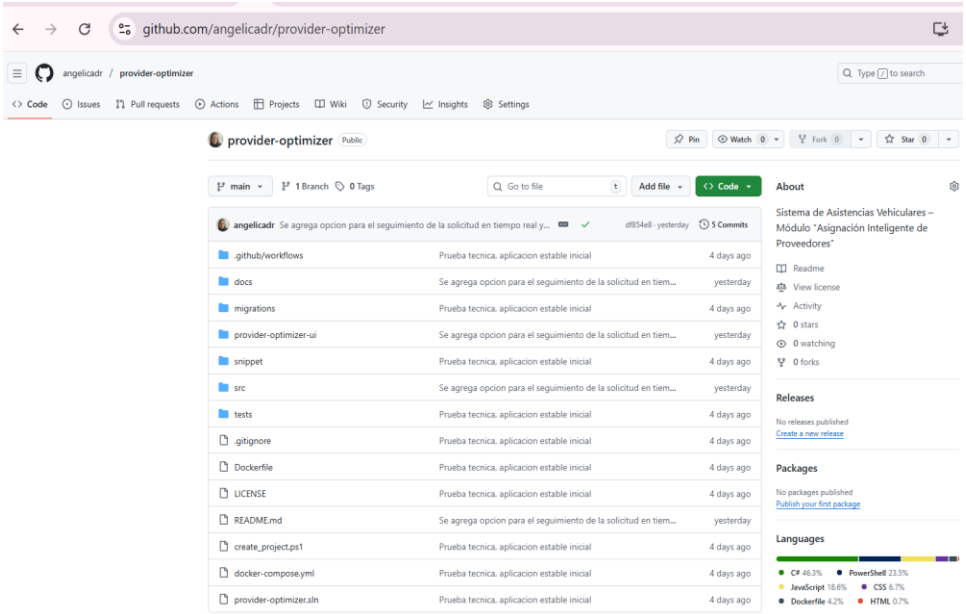
Docker Compose

- Servicios incluidos:
  - db → PostgreSQL
  - provideroptimizer-api → Backend .NET 8
  - provideroptimizer-frontend → React/Nginx
- Comando de despliegue:
  - docker compose up --build

Repositorio del Proyecto

- GitHub:
  - https://github.com/angelicadr/provider-optimizer

- Incluye:
  - Backend .NET
  - Frontend React
  - Docker
  - Scripts SQL
  - Documentación técnica



• Análisis del Code Review snippet defectuoso

Problemas encontrados:

- No hay verdadera asincronía.
- No se permite async/await para flujos vinculados a E/S.
- No se permite validación de entrada ni comprobación de valores nulos.
- Error de concurrencia.
- No se permite SOLID: la clase asume múltiples responsabilidades y expone el estado mutable.
- No se permiten DTO ni gestión de errores.
- No hay separación de responsabilidades

```
[HttpPost("assign")]
public async Task<IActionResult> AssignProvider(Request request)
{
    var providers = _db.Providers.ToList();
    var selected = providers.FirstOrDefault(); // escoger el primero nomas
    if(selected == null) return BadRequest("No providers");

    selected.IsBusy = true;
    _db.SaveChanges();

    return Ok(selected);
}
```

Recomendaciones:

- Control de concurrencia
- Validaciones robustas
- Arquitectura limpia y extensible
- Políticas de selección configurables
- Asincronía real
- Calidad y resiliencia

Conclusión

- La solución cumple completamente los requisitos de la prueba:
  - Diseño escalable.
  - Arquitectura limpia.
  - Trazabilidad en tiempo real.
  - Despliegues unificado.
  - Documentación Técnica y evidencia de pruebas incluida.
  - Análisis del Code Review snippet defectuoso