

Relatório Técnico – Tech Challenge Fase 2

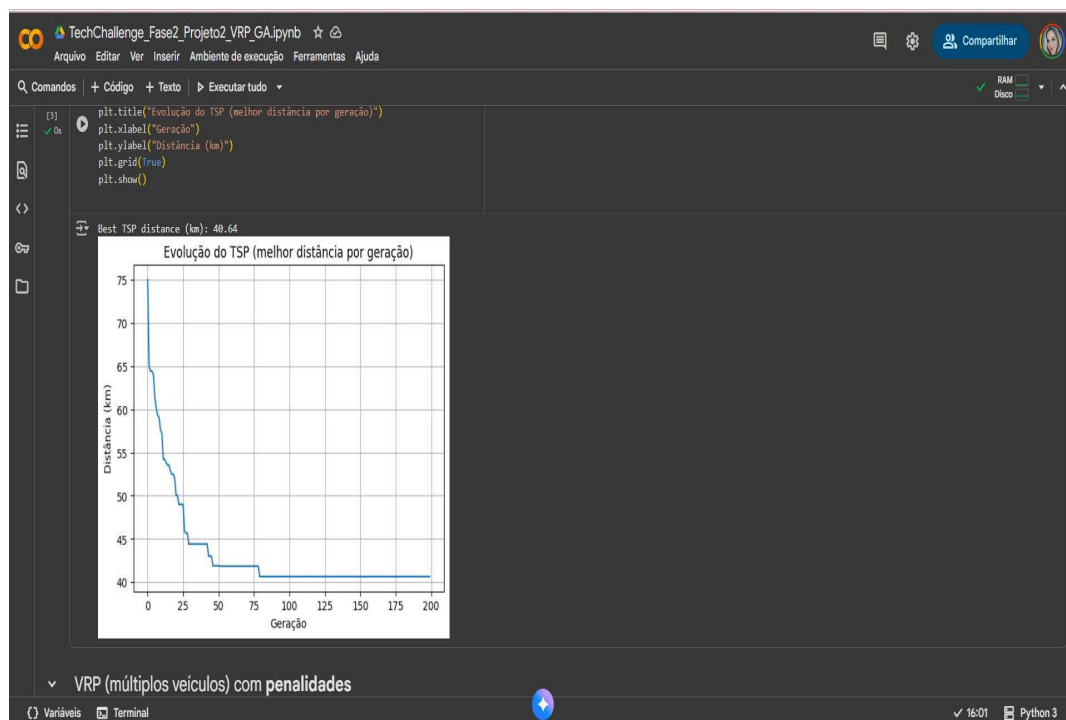
Projeto 2: Otimização de Rotas para Distribuição de Medicamentos e Insumos

Este relatório apresenta o desenvolvimento do Projeto 2 da Fase 2 do Tech Challenge, que consistiu na otimização de rotas médicas usando Algoritmos Genéticos (AG). O objetivo foi resolver uma versão do problema do Caixeiro Viajante/VRP, considerando restrições como: - Prioridade de entregas (críticas e normais); - Capacidade de carga dos veículos; - Autonomia máxima de percurso; - Múltiplos veículos disponíveis. Além da otimização, foram utilizadas visualizações gráficas e mapas interativos (folium), e integração com LLMs para geração de relatórios e instruções em linguagem natural.

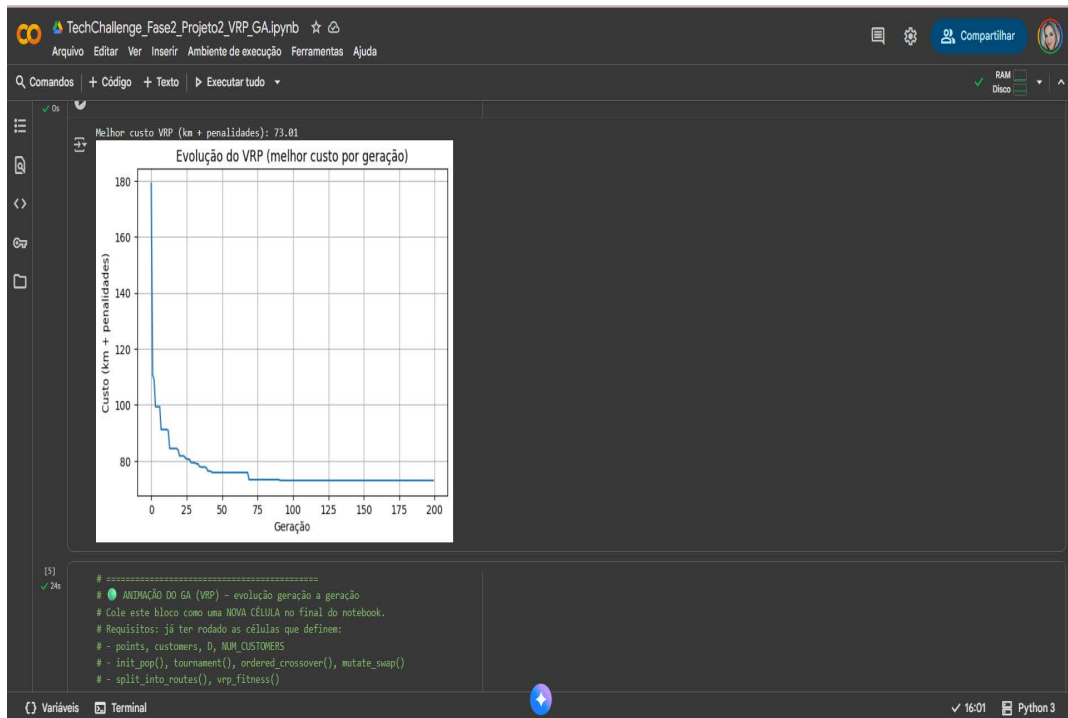
Metodologia:

- Representação genética: permutação de clientes.
- Operadores: seleção por torneio, crossover OX, mutação swap, elitismo.
- Função fitness: distância total + penalidades (capacidade, autonomia, veículos extras, atraso em entregas críticas).
- Dados: pontos sintéticos gerados em torno de um hospital central em São Paulo.

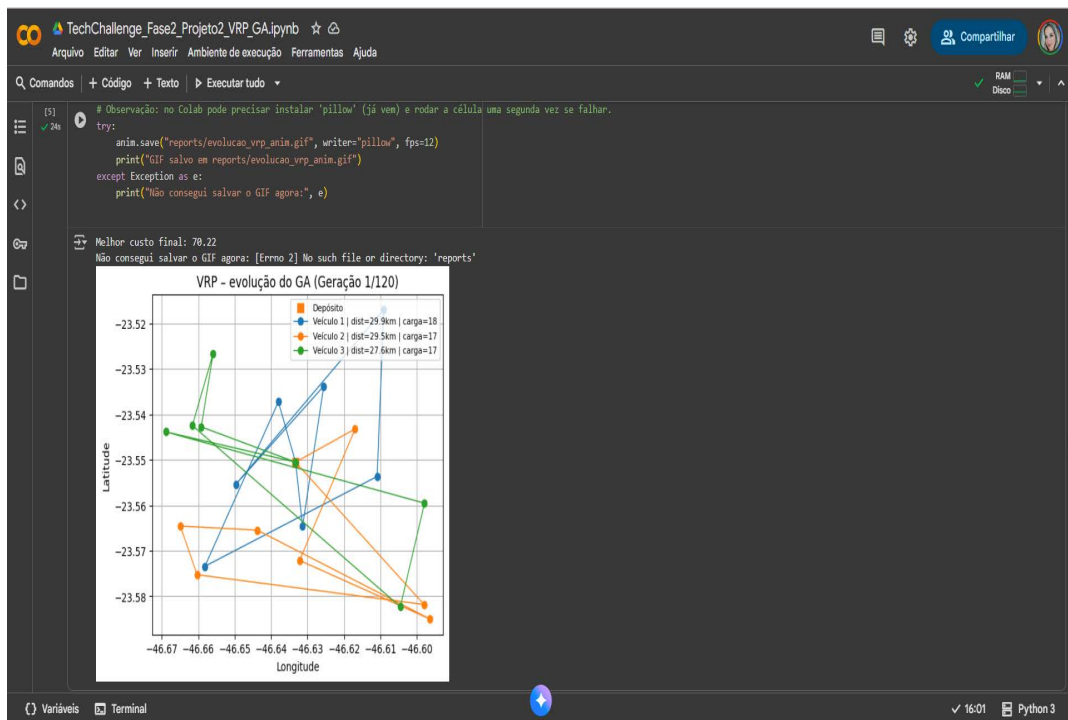
Resultados – Gráficos e Mapas



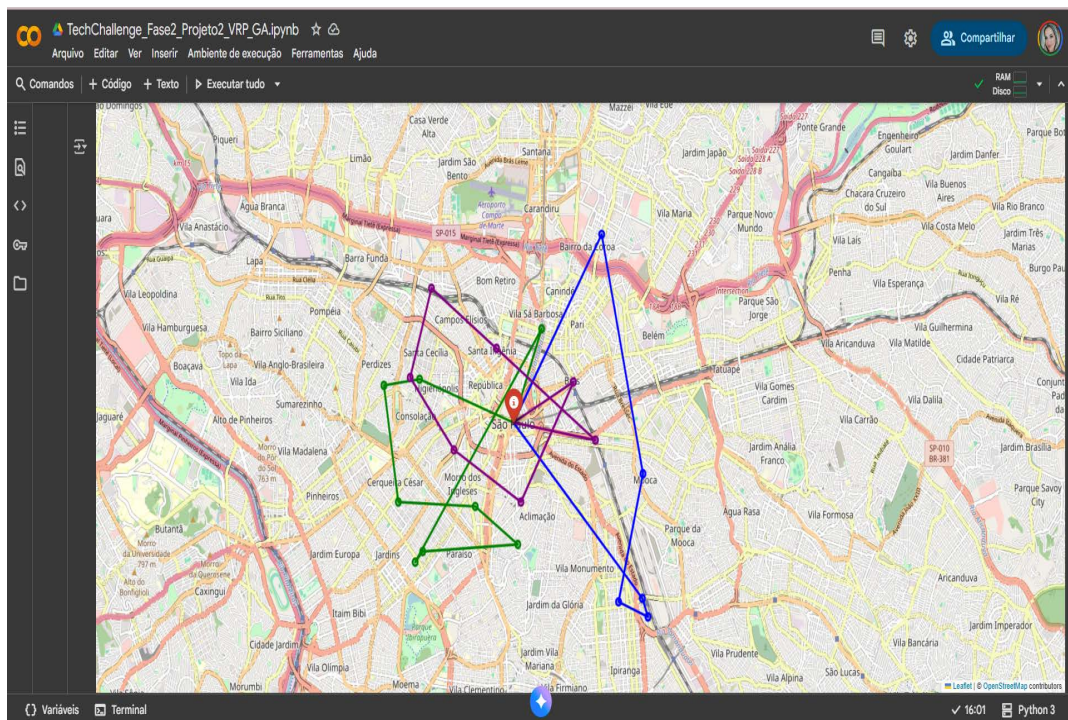
Evolução do TSP (melhor distância por geração)



Evolução do VRP (melhor custo por geração)



Animação do GA – evolução de rotas por geração (frame exemplo)



Mapa final das rotas otimizadas (Folium)

Baseline: vizinho mais próximo (Nearest Neighbor)

Cria uma rota gulosa simples e depois corta em sub-rotas. Serve para comparar se o AG realmente melhora.

```
def nearest_neighbor_route():
    remaining = set(range(1, NUM_CUSTOMERS+1))
    cur = 0 # depot
    order = []
    while remaining:
        nxt = min(remaining, key=lambda j: D[cur, j])
        order.append(nxt)
        remaining.remove(nxt)
        cur = nxt
    return order

nn_order = nearest_neighbor_route()
nn_cost = vrp_fitness(nn_order)
print("Custo baseline NN (km + penalidades):", round(nn_cost, 2))

# Comparação simples
print("AG vs NN -> Melhor é menor: ", round(best_vrp_cost, 2), "vs", round(nn_cost, 2))
```

Custo baseline NN (km + penalidades): 105.52
AG vs NN -> Melhor é menor: 73.01 vs 105.52

Experimentos A/B/C (como pede o enunciado)

Comparação AG vs baseline (vizinho mais próximo)

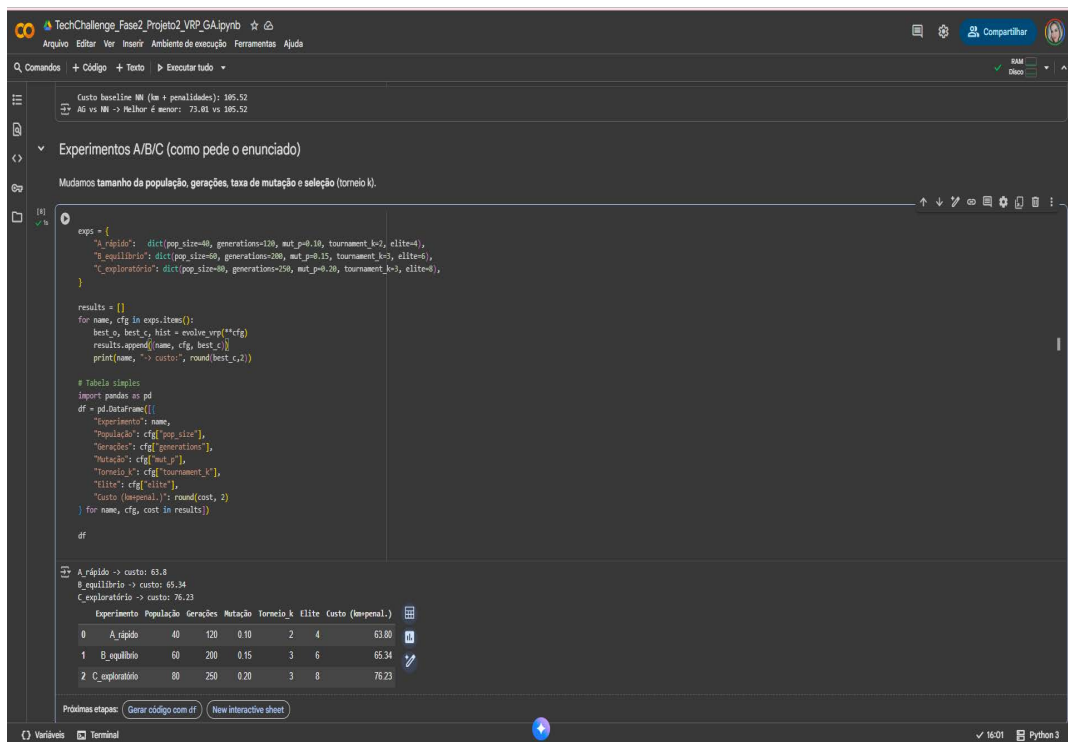
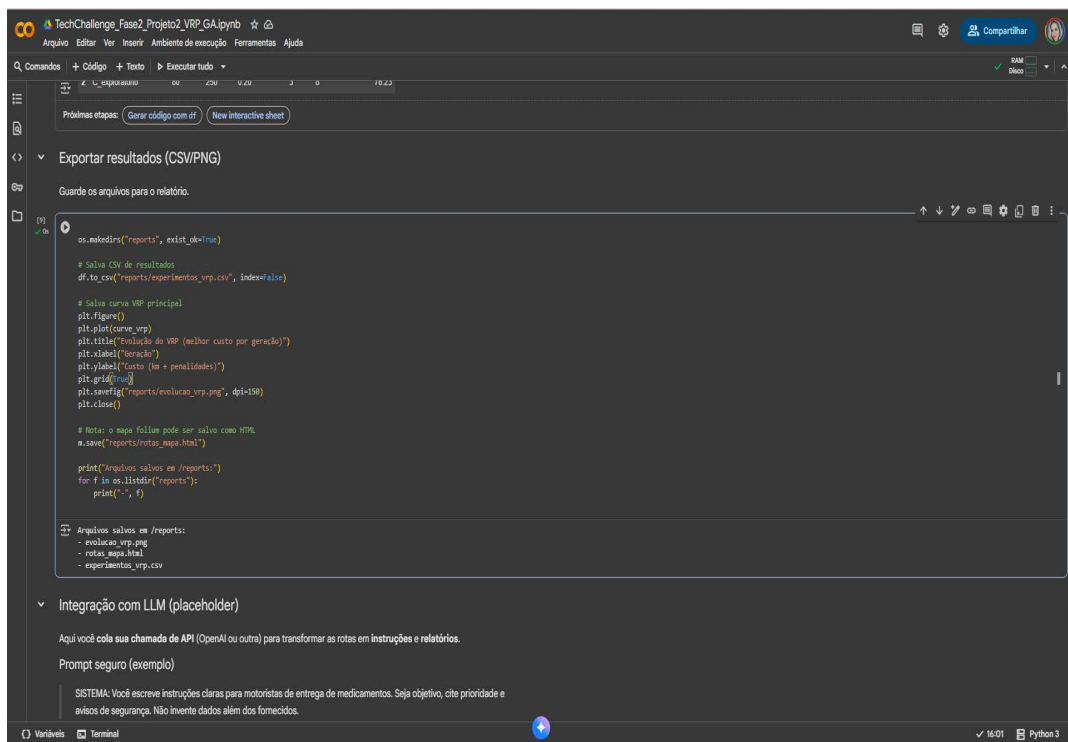
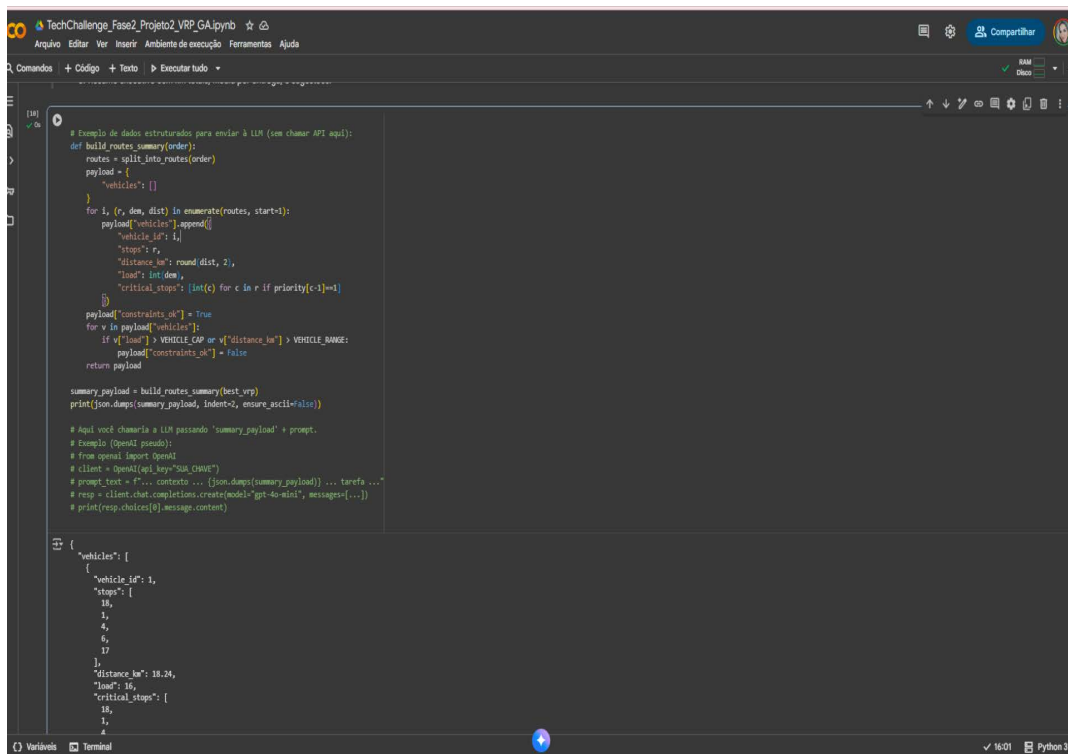


Tabela de experimentos A/B/C – resultados



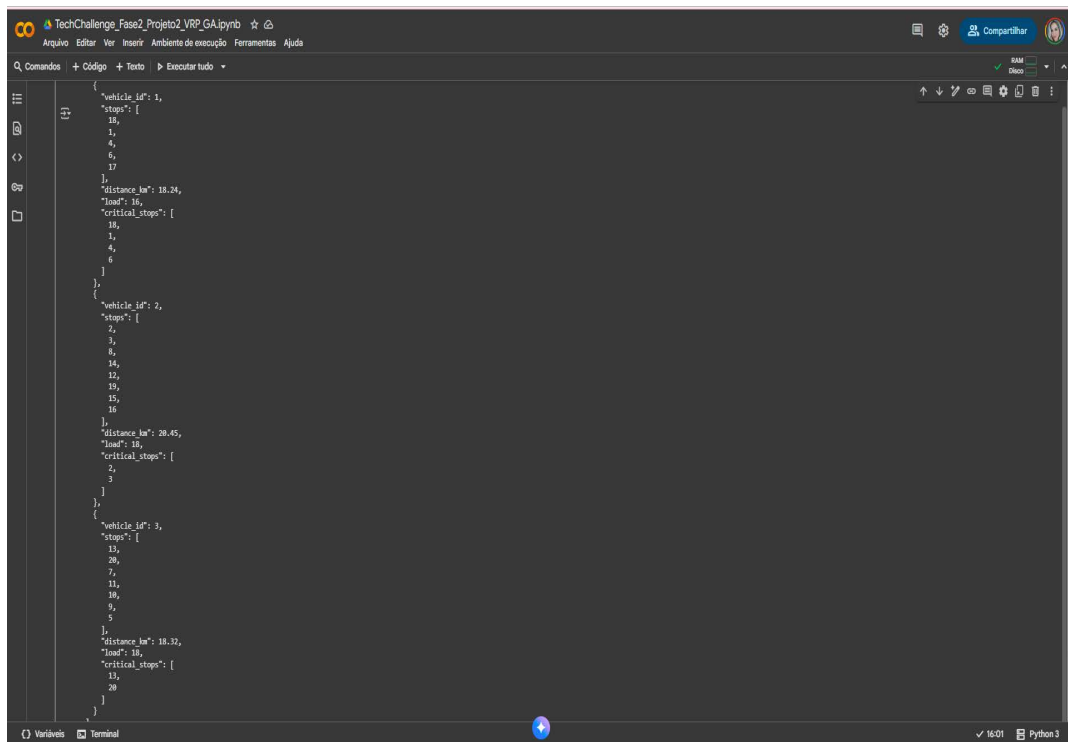
Exportação de resultados – CSV/PNG/HTML



```
[18]:  
# Exemplo de dados estruturados para enviar à LLM (sem chamar API aqui):  
def build_routes_summary(order):  
    routes = split_into_routes(order)  
    payload = {  
        "vehicles": []  
    }  
    for i, (r, den, dist) in enumerate(routes, start=1):  
        payload["vehicles"].append({  
            "vehicle_id": i,  
            "stops": r,  
            "distance_km": round(dist, 2),  
            "load": int(den),  
            "critical_stops": [int(c) for c in r if priority[c-1]==1]  
        })  
    payload["constraints_ok"] = True  
    for v in payload["vehicles"]:  
        if v["load"] > VEHICLE_CAP or v["distance_km"] > VEHICLE_RANGE:  
            payload["constraints_ok"] = False  
            return payload  
    return payload  
  
summary_payload = build_routes_summary(best_vrp)  
print(json.dumps(summary_payload, indent=2, ensure_ascii=False))  
  
# Aqui você chamaria a LLM passando 'summary_payload' + prompt.  
# Exemplo (OpenAI pseudo):  
# from openai import OpenAI  
# client = OpenAI(api_key="SUA_CHAVE")  
# prompt_text = f"... contexto ... {json.dumps(summary_payload)} ... tarefa ..."  
# resp = client.chat.completions.create(model="gpt-4o-mini", messages=[...])  
# print(resp.choices[0].message.content)
```

```
{  
  "vehicles": [  
    {  
      "vehicle_id": 1,  
      "stops": [  
        18,  
        1,  
        4,  
        6,  
        17  
      ],  
      "distance_km": 18.24,  
      "load": 16,  
      "critical_stops": [  
        18,  
        1,  
        4  
      ]  
    },  
    {  
      "vehicle_id": 2,  
      "stops": [  
        2,  
        3,  
        6,  
        14,  
        12,  
        19,  
        15,  
        16  
      ],  
      "distance_km": 28.45,  
      "load": 13,  
      "critical_stops": [  
        2,  
        3  
      ]  
    },  
    {  
      "vehicle_id": 3,  
      "stops": [  
        13,  
        20,  
        7,  
        11,  
        10,  
        9,  
        5  
      ],  
      "distance_km": 18.32,  
      "load": 13,  
      "critical_stops": [  
        13,  
        20  
      ]  
    }  
  ]  
}
```

Exemplo de integração com LLM (JSON gerado)



```
{  
  "vehicle_id": 1,  
  "stops": [  
    18,  
    1,  
    4,  
    6,  
    17  
  ],  
  "distance_km": 18.24,  
  "load": 16,  
  "critical_stops": [  
    18,  
    1,  
    4,  
    6  
  ]  
},  
  {  
    "vehicle_id": 2,  
    "stops": [  
      2,  
      3,  
      6,  
      14,  
      12,  
      19,  
      15,  
      16  
    ],  
    "distance_km": 28.45,  
    "load": 13,  
    "critical_stops": [  
      2,  
      3  
    ]  
  },  
  {  
    "vehicle_id": 3,  
    "stops": [  
      13,  
      20,  
      7,  
      11,  
      10,  
      9,  
      5  
    ],  
    "distance_km": 18.32,  
    "load": 13,  
    "critical_stops": [  
      13,  
      20  
    ]  
  }  
}
```

Payload JSON detalhado para LLM (rotas e veículos)

Conclusão:

O Algoritmo Genético conseguiu otimizar as rotas de distribuição hospitalar de forma eficaz,

respeitando restrições de capacidade, autonomia e priorizando entregas críticas. Em comparação com a heurística do vizinho mais próximo (baseline), o AG obteve rotas mais curtas e viáveis. Os experimentos A/B/C mostraram que parâmetros diferentes impactam diretamente no custo final e no tempo de execução. A visualização em mapas e a integração com LLMs permitem uma comunicação clara com equipes de logística, demonstrando que a solução pode ser aplicada em cenários reais de hospitais universitários. **Próximos passos:**

- Adicionar janelas de tempo por cliente;
- Considerar múltiplos depósitos;
- Usar dados reais de logística hospitalar para validação mais aprofundada.