ProvedorNet

Sistema de Gestão para Provedores de Internet (ISP)

Kaiky França da Silva Matheus Fagundes Araujo Yago Almeida Melo

Disciplina de Desenvolvimento Web / Engenharia de Software

27 de novembro de 2025

Agenda

Contexto e Motivação

Contexto do Problema:

- Alta demanda por serviços de internet fibra óptica.
- Provedores locais sofrem com gestão manual de planos e suporte técnico repetitivo.

Motivação:

- Centralizar a gestão de assinaturas.
- Oferecer autoatendimento ao cliente (2ª via, upgrade de plano).
- Reduzir carga de suporte Nível 1 com Chatbot IA.

Público-Alvo

O sistema foi projetado para atender dois perfis distintos:

Administradores (ISP)

Gestores que precisam cadastrar planos, visualizar métricas financeiras (KPIs) e gerenciar a base de clientes.

Clientes Finais

Usuários residenciais que desejam consultar faturas, testar velocidade de conexão, alterar planos e obter suporte rápido via IA.

Principais User Stories

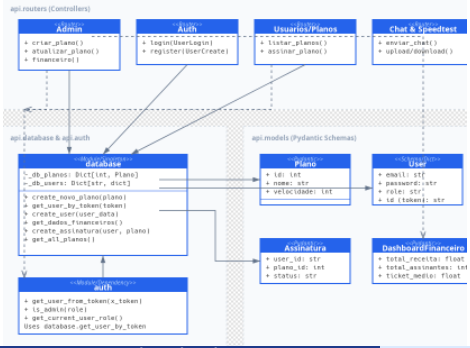
Utilizamos metodologias ágeis para definir o escopo:

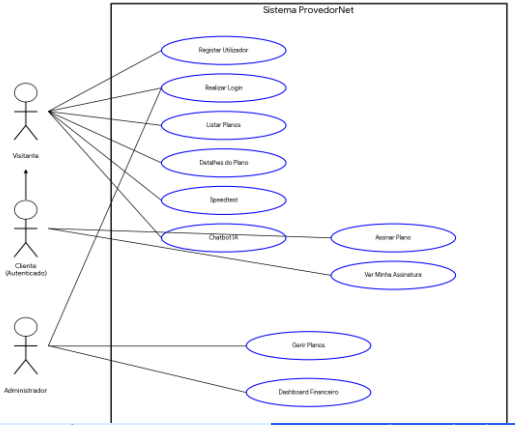
- **[Cliente]** Como cliente, quero visualizar meu consumo e faturas para manter o pagamento em dia.
- **[Cliente]** Como cliente, quero realizar um teste de velocidade (Speedtest) para validar minha conexão.
- **[Cliente]** Como cliente, quero interagir com uma IA para tirar dúvidas sobre planos sem ligar para o suporte.
- **[Admin]** Como administrador, quero um dashboard financeiro para ver a receita mensal recorrente.
- **[Admin]** Como administrador, quero criar, editar e excluir planos de internet disponíveis.

Modelagem do Sistema

Diagramas Utilizados:

- **Diagrama de Casos de Uso:** Mapeamento das permissões (Admin vs User).
- **Diagrama de Classes:** Estrutura dos modelos de dados (`Plano`, `Assinatura`, `Usuario`).





Arquitetura de Software

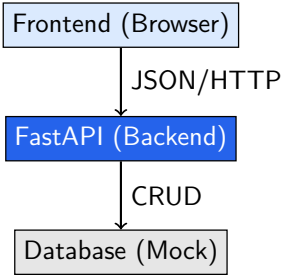
Adotamos uma arquitetura **Cliente-Servidor** moderna e desacoplada:

Frontend (SPA Artesanal):

- HTML5 + TailwindCSS.
- Vanilla JS (sem frameworks pesados).
- Roteamento dinâmico no cliente.

Backend (API REST):

- Python com FastAPI.
- Pydantic para validação de dados.
- In-Memory Database (Mock) para prototipagem rápida.



Padrões e Decisões de Design

- **Camadas (Layered Pattern):**
  - `routers`: Pontos de entrada HTTP.
  - `database/services`: Lógica de negócio e persistência.
  - `models`: Definição de esquemas (DTOs).
- **Injeção de Dependência:** Uso nativo do FastAPI (`Depends`) para gestão de autenticação e sessões.
- **Justificativa:** Escolha do FastAPI pela performance (Assíncrono) e geração automática de documentação (Swagger UI). TailwindCSS para agilidade na estilização responsiva.

Tecnologias e Frameworks

Componente	Tecnologia
Linguagem Backend	Python 3.10+
Framework API	FastAPI + Uvicorn
Frontend	HTML5, JavaScript (ES6+)
Estilização	Tailwind CSS (via CDN)
Testes	Pytest + TestClient
HTTP Client	HTTPX (para chamadas assíncronas)

Fluxo de Chamadas (Exemplo: Chatbot)

Fluxo de uma requisição para o assistente virtual:

1. **Frontend:** Captura input do usuário e faz `POST /chat/enviar`.
2. **Router (`chatbot.py`):** Recebe a mensagem e injeta contexto do sistema.
3. **Service:** Constrói o prompt com a lista de planos atualizada do banco.
4. **Mock/LLM:** Simula ou processa a resposta da IA.
5. **Response:** Retorna JSON para o Frontend renderizar.

```
@router.post("/enviar")
async def conversar(dados: models.ChatMensagem):
    prompt = gerar_prompt_sistema() # Busca planos no DB
    resposta = await chamar_llm(dados.mensagem, prompt)
    return {"resposta": resposta}
```

Estratégia de Testes

Garantia de qualidade focada em testes automatizados de integração:

- **Ferramentas:** `pytest` e `unittest.mock`.
- **Cobertura:**
  - Testes de Rotas (Status 200, 404, 403).
  - Testes de Segurança (Validação de Token Admin vs User).
  - Mocking de APIs Externas (Simulação da OpenAI e Speedtest para não gastar créditos/banda nos testes).

"Mockamos o objeto response para garantir que o teste do Chatbot não dependa da internet ou da API externa estar online."

Live Demo

1. Login e Autenticação (Admin vs User).
2. Dashboard Financeiro (Admin).
3. Speedtest com animação em tempo real.
4. Chatbot respondendo sobre planos (Context Aware).

Conclusões e Aprendizados

Principais Aprendizados:

- Construção de SPAs sem frameworks complexos (React/Vue) reduz a complexidade de build.
- Importância de testes assíncronos em APIs modernas.

Melhorias Futuras:

- Persistência real com PostgreSQL ou SQLite.
- Implementação de JWT com expiração real.
- Containerização com Docker para deploy.

O que faríamos diferente?

- Começaríamos com TypeScript no frontend para melhor tipagem dos dados consumidos da API.

Obrigado!

Perguntas?