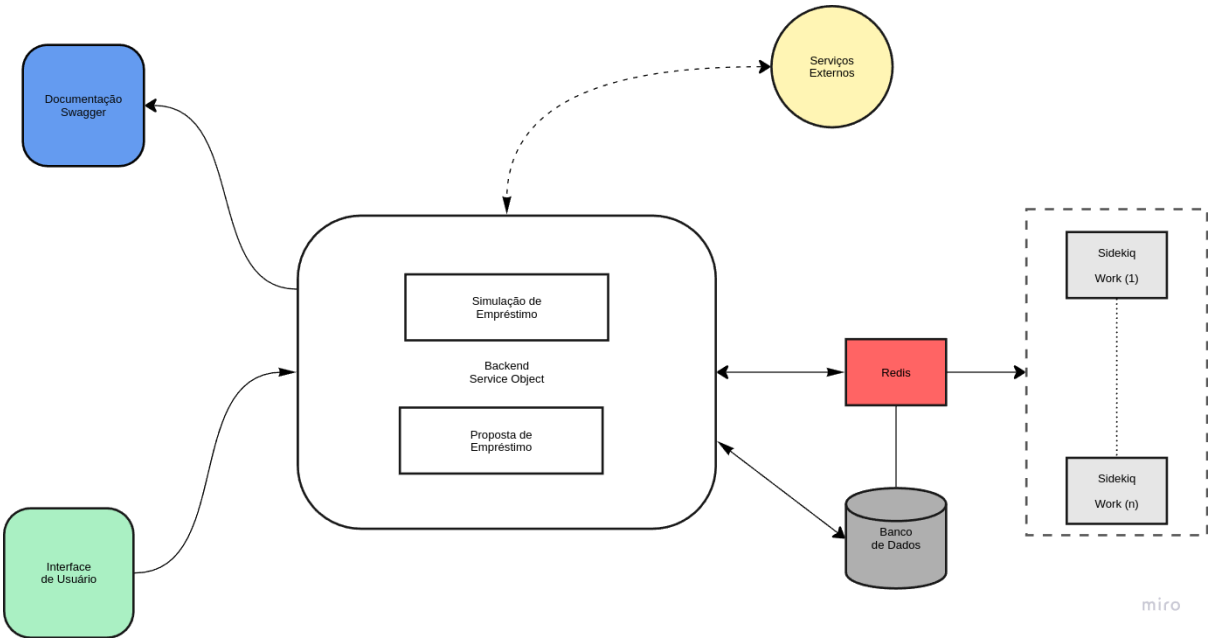
Desenho da Arquitetura



Interface do Usuário (Frontend)

O sistema foi concebido para ser acessado por qualquer tipo de interface — seja uma aplicação web, mobile ou desktop — através de uma API RESTful bem definida.

Isso garante total flexibilidade para que diferentes equipes ou plataformas possam consumir os serviços do backend, bastando realizar as requisições HTTP adequadas (como simulação ou geração de proposta).

Assim, o backend é totalmente desacoplado da camada de interface, permitindo que o frontend seja desenvolvido em qualquer tecnologia (React, Vue.js, Flutter, Swift, etc.), com comunicação feita exclusivamente via requisições à API.

Backend (API)

O backend será desenvolvido utilizando Ruby on Rails, um framework robusto, seguro e altamente produtivo para a criação de APIs RESTful.

Arquitetura e Organização da Lógica

- **Controllers**: Responsáveis por tratar as requisições HTTP, delegando a lógica para Service Objects. Mantêm-se leves, coesos e focados no fluxo da requisição.

- **Service Objects**: Encapsulam regras de negócio (simulação de empréstimo, cálculo de parcelas, geração de proposta), separando responsabilidades e facilitando testes.

- **Models (ActiveRecord)**: Representam entidades do sistema (Usuário, Cliente, Simulação, Proposta), com validações e persistência no banco PostgreSQL.

Processamento Assíncrono

Integração com **Sidekiq** para execução de tarefas demoradas (ex: envio de emails, geração de relatórios).

Performance e Escalabilidade

- **Redis** como cache, reduzindo tempo de resposta em simulações repetidas.

- Arquitetura com **Service Objects** permite escalabilidade modular.

Banco de Dados – PostgreSQL

Relacional, seguro e confiável, com suporte a transações e consistência, ideal para dados financeiros.

Redis

Utilizado como cache e como backend para filas do Sidekiq. Excelente desempenho para leitura/escrita.

Sidekiq

Executa tarefas em background, liberando o fluxo principal. Permite escalar apenas workers conforme a demanda.

Documentação – Swagger/Rswag

Gera documentação interativa da API para facilitar a integração com outras equipes e parceiros.

Padrões de Projeto e Boas Práticas

Service Objects

Cada regra de negócio é isolada em um serviço específico (ex: `Simulations::CalculateLoanService`,

`Proposals::GenerateOfferService`), evitando acoplamento e facilitando a manutenção.

Princípios SOLID

- **SRP (Responsabilidade Única)**: Cada classe tem uma responsabilidade clara e definida.
- **OCP (Aberto para extensão, fechado para modificação)**: Possibilidade de adicionar novas features sem alterar código existente.

Design Orientado a Objetos

Objetos com responsabilidades bem delimitadas, promovendo reuso e baixo acoplamento.

Validações e Segurança

- Validações robustas via ActiveRecord.
- Proteções contra SQL Injection, Mass Assignment e uso de HTTPS.

Considerações de Escalabilidade

Arquitetura Modular e Horizontal

- O backend é dividido em módulos independentes e pode ser escalado horizontalmente (subindo mais instâncias da aplicação conforme demanda).
- APIs REST permitem múltiplos clientes acessando simultaneamente.

Sidekiq para Escalabilidade de Processos

- Tarefas como Simulações e propostas são processadas por workers em background.
- É possível escalar workers separadamente, mantendo a fluidez da API.

Cache com Redis

- Simulações e propostas recorrentes podem ser armazenadas para acelerar a resposta.

- Suporte a altas taxas de leitura com baixa latência.

Banco com Read Réplicas

- Em cenários de alto tráfego, é possível utilizar réplicas para distribuir a carga de leitura, mantendo a performance.

Autenticação e Autorização

Devise

Utilizado para autenticação de usuários. Proporciona:

- Login seguro com criptografia de senhas.
- Recuperação de senha, confirmação de conta, timeout de sessão.
- Integração com JWT para acesso autenticado via API.

Pundit

Responsável pela autorização:

- Define quem pode acessar ou modificar recursos (ex: apenas o dono de uma proposta pode visualizá-la).
- Aplica as regras de negócio de forma clara e reutilizável por meio de *policies*.

API Design

A API segue padrões RESTful e está organizada sob o namespace `/api/v1`.

1. Simular Empréstimo

POST /api/v1/simulations

Request:

```
{
 "document": "12345678900",
 "amount": 10000,
 "installments": 12,
 "birth_date": "1995-08-10"
}
```

Response:

```
{
 "id": 1,
 "amount": 10000,
 "installments": 12,
 "interest_rate": 3.0,
 "monthly_payment": 850,
 "total_payable": 10200,
 "total_interest": 200
}
```

2. Gerar Proposta

POST /api/v1/proposals

Request:

```
{
 "simulation_id": 1
}
```

Response:


```
{
 "id": 1,
 "simulation_id": 1,
 "customer_id": 1,
 "amount": 10000,
 "installments": 12,
 "interest_rate": 3.0,
 "monthly_payment": 850,
 "total_payable": 10200,
 "total_interest": 200,
 "status": "pending",
 "created_at": "2025-04-30T17:30:00Z",
 "updated_at": "2025-04-30T17:30:00Z"
}
```


Todos os endpoints são protegidos com autenticação e/ou JWT (Devise + JWT) e autorização via Pundit.