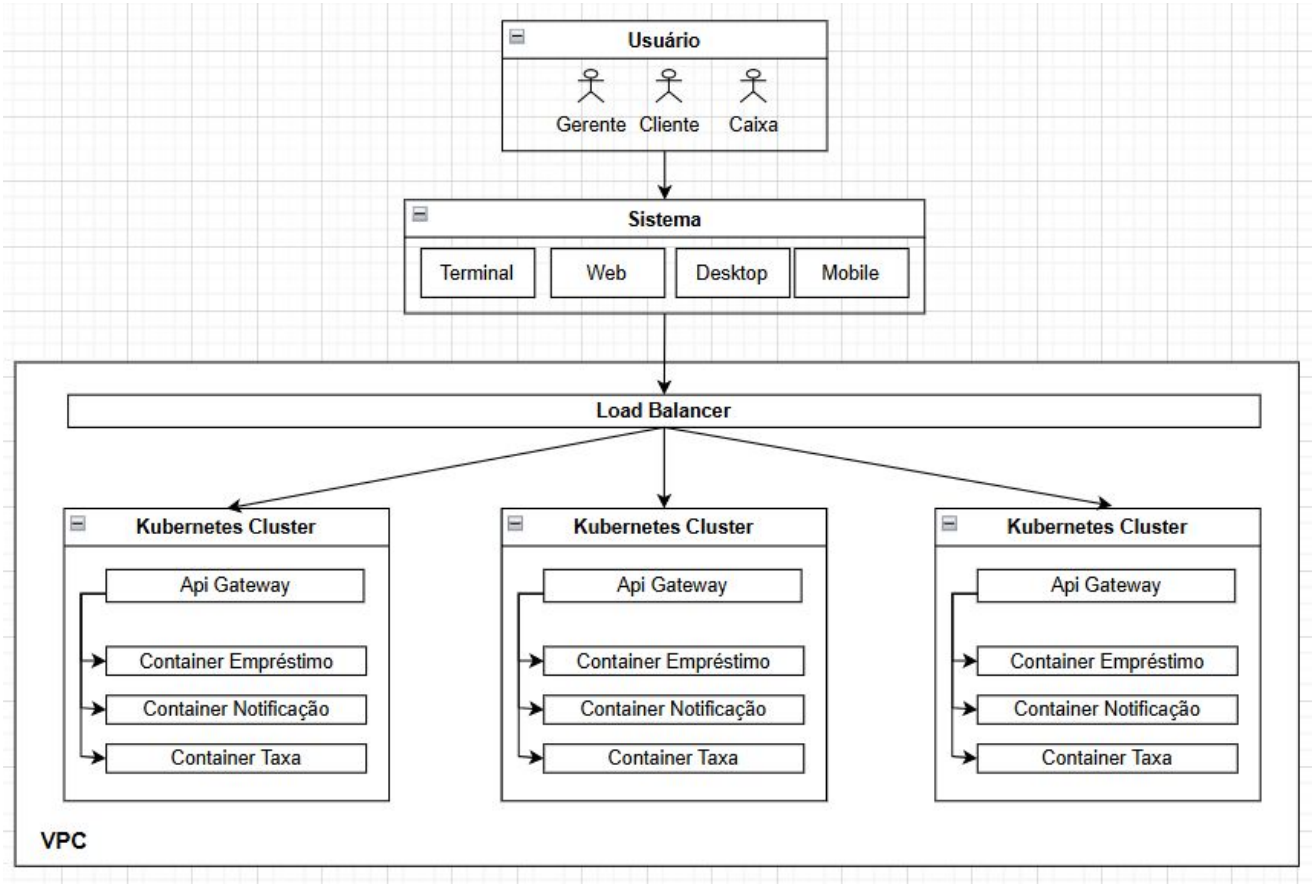
Teste de Arquitetura

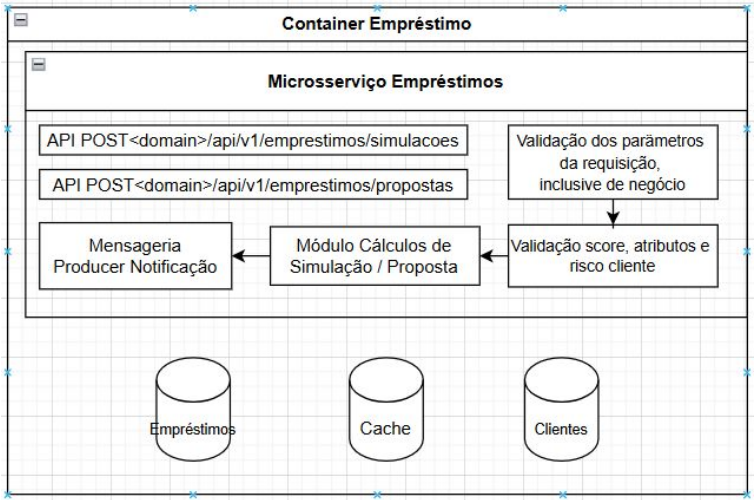
Sistema de Simulação e Proposta de Empréstimos

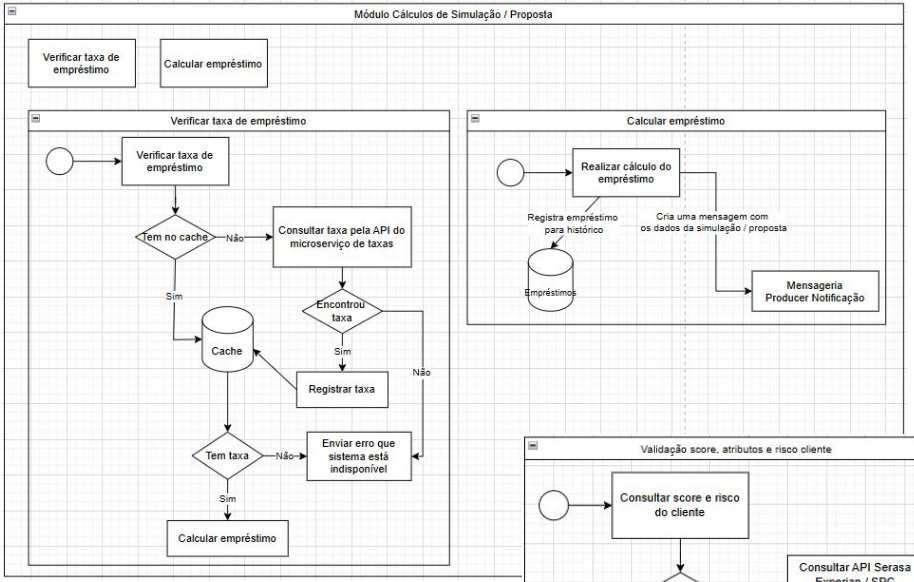
Autor: Guilherme Tiago Bulla

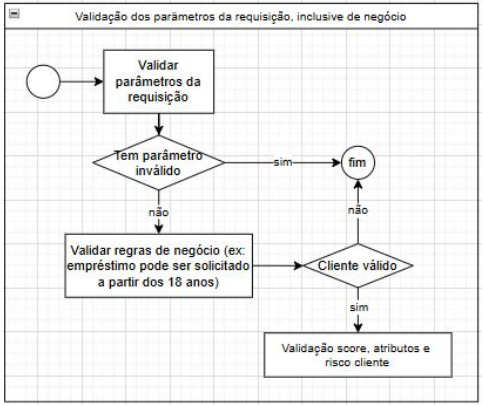
Desenho da Arquitetura - Geral

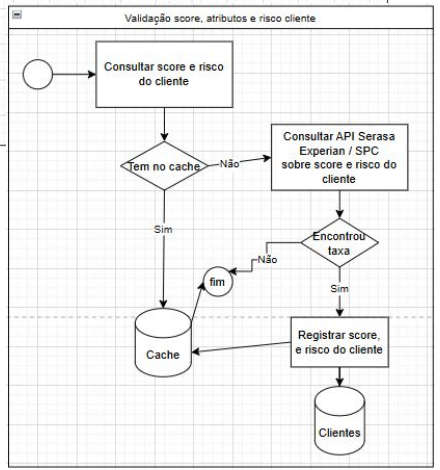


Desenho da Arquitetura - Container Empréstimo



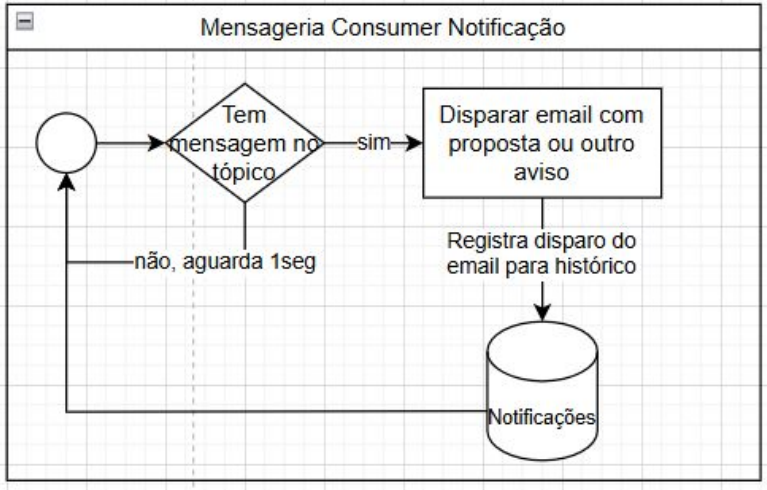




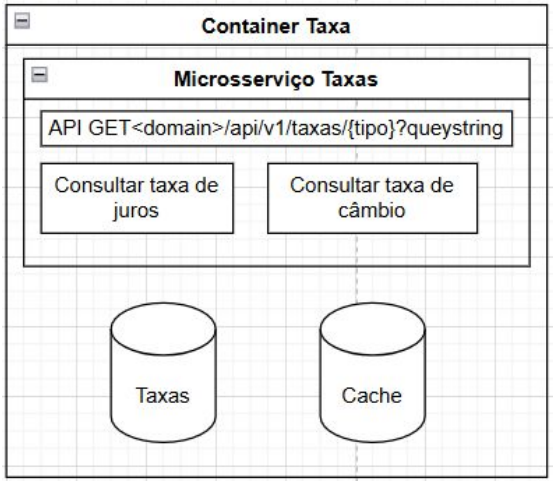


Desenho da Arquitetura - Container Taxa

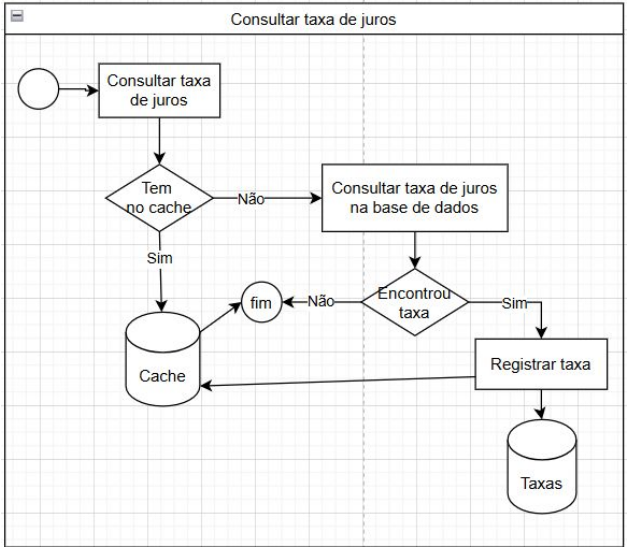
PathParameter "id" representa um hash da simulação de empréstimo

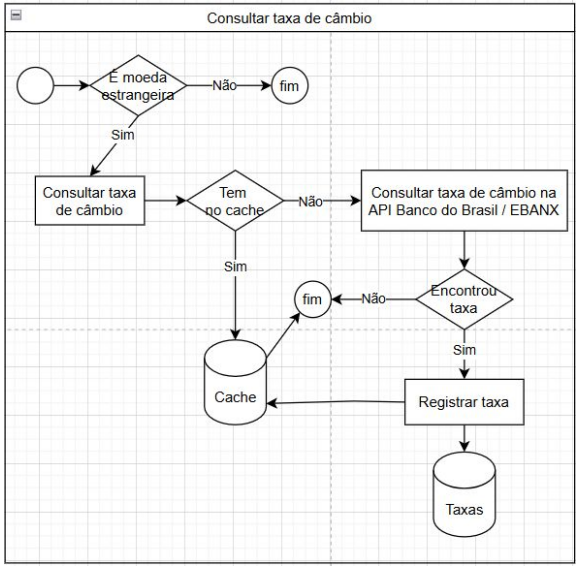


Desenho da Arquitetura - Container Notificação



PathParameter "tipo" representa se a taxa é anual, mensal, etc
Na querystring da URL podemos passar a idade e outros parâmetros condicionais





Desenho da Arquitetura - Componentes Principais

1. **Interface de Usuário (Sistema):**
    a. Terminal: Totens de atendimento com Android
    b. Web: Aplicativo web responsivo em React.
    c. Desktop: Sistema desenvolvido em Java
    d. Mobile: Aplicativo desenvolvido com Kotlin Multi Plataform ou nativamente com Kotlin e Swift
2. **Load Balancer:** distribuição da carga de trabalho conforme escalabilidade
3. **API Gateway:** gerenciamento das requisições de APIs
4. **Backend:** construção de microserviços com framework Ktor
    a. Microserviços:
        i. Empréstimos: APIs Restful de simulação e propostas
        ii. Notificações: API Restful de envio de simulação ou proposta de empréstimo por email
        iii. Taxas: API Restful de consulta de taxas de juros e câmbio
5. **Banco de Dados:** PostgreSQL para armazenamento transacional e Redis para cache
    a. Empréstimos: histórico de empréstimos
    b. Clientes: armazena dados de score, atributos e risco financeiro do cliente
    c. Notificações: status de envio dos emails de proposta / simulação
    d. Taxas: taxas de juros e câmbio de acordo com parâmetros condicionais (idade, score da pontuação, etc)
6. **Serviços Externos:** Integração com serviços de validação de crédito e câmbio.
    a. Consulta API do SPC e Serasa Experian: análise de score e risco
    b. Consulta API câmbio (Banco Central do Brasil ou EBANX): taxas diárias
7. **Motor de Simulação:** Módulo da simulação de empréstimo responsável pelos cálculos.
8. **Infraestrutura:** Kubernetes para orquestração de contêineres e AWS para infraestrutura em nuvem.

Desenho da Arquitetura - Justificativas Tecnológicas

Ktor com Kotlin: Permite escrita de código conciso, suporte às melhores práticas de API e alta performance.
PostgreSQL: Banco relacional, robusto e com bom suporte a consultas complexas.
Redis: Minimiza latências em dados acessados com alta frequência.
Docker: Promove a IaC com consistência, portabilidade e eficiência no gerenciamento de ambiente
Kubernetes: Escalabilidade automática e gestão eficiente de recursos. O orquestrador pode ser o EKS.
Kafka: Processa grandes volumes de mensagens, com alta disponibilidade e particionamento para escalabilidade.
JWT: Autenticação e autorização com tokenização
Lettuce: Biblioteca para manipulação de cache com o Redis
ContentNegotiation: Suporte a serialização e deserialização automática
Koin: Injeção de dependência para melhorar a testabilidade e desacoplamento
CallLogging: Facilitar a auditoria e monitoramento
SLF4: Logs mais limpos e consistentes

LogBack: Logging muito flexível e configurável de alta performance
Routing: Definição e organização das rotas dos microserviços
Exposed: ORM flexível para bancos de dados relacionais, suporte a transação e segurança contra SQL Injection
Detekt e Ktlint: Linters para análise estática de código
Eureka: Serviço de descoberta de microserviços
Hikari: Gerenciamento de conexões rápido e eficiente
Flyway: Controle de versão e migrações do banco de dados
Swagger: Documentação das APIs
StatusPages: Plugin para estruturar melhor as mensagens de retorno das requisições
JUnit, Mockito, PowerMockito e MockK: Testes unitários, inclusive para testes de dependências e funções privadas
VPC: Isolamento, controle de roteamento e segurança

Padrões de Projeto e Boas Práticas

Padrões de Projeto Avançados

- **Domain-Driven Design (DDD):** Para separar responsabilidades em camadas (Domain, Application, Infrastructure).
- **Clean Architecture:** Traz escalabilidade, desacoplamento de frameworks, modularidade, testabilidade e manutenibilidade.
- **Event-Driven Architecture (EDA):** Arquitetura orientada a eventos entre microserviços por meio de eventos ou mensagens.
- **Database per Service:** Cada microserviço tem o seu banco de dados
- **Strangler Fig Pattern:** Caso fosse necessário migrar a arquitetura de um projeto
- **CQRS (Command Query Responsibility Segregation):** Para separar consultas e comandos.
- **Versionamento de API:** Aplique um padrão de versionamento
- **Factory e Strategy:** Para criar diferentes cenários de simulação de empréstimos.
- **Saga Coreografia:** Comunicações por eventos, não exigindo um orquestrador central. É escalável e resiliente.
- **Rate limiting:** Configuração para evitar sobrecarga de serviços ou filas, protegendo o sistema contra picos de tráfego.
- **Timeout:** Limitar o tempo de processamento das requisições que não podem ser processadas dentro de um prazo específico.
- **Service Discovery:** Automação da descoberta dos microsserviços em arquitetura distribuída.
- **Circuit Breaker:** Implementação com Resilience4j para tratar falhas em integrações externas.
- **Retry:** Tentativas automáticas de reprocessamento

Autenticação e Autorização

- **JWT (JSON Web Tokens):** Para autenticação segura por tokenização.
- **OAuth 2.0:** Para autorização granular com diferentes papéis (usuários e administradores).
- **Segurança na API:** HSTS, validação de input e logs auditáveis para requisições sensíveis.

Considerações de Escalabilidade e Resiliência

Escalabilidade

- **Cache Distribuído (Redis):** Reduz cargas no banco de dados, ganho no tempo de resposta.
- **Auto-Scaling:** Configurações em Kubernetes para escalabilidade horizontal automática.
- **Partitioning e Sharding:** Dividir filas e tópicos por partições, permitindo que múltiplos consumidores processem simultaneamente.
- **Fila de mensagens:** Gerencia grandes volumes de requisições de maneira assíncrona
- **Paginação e Limitação de Resultados:** Para evitar que grandes volumes de dados sejam transferidos de uma vez, implemente paginação nas APIs para retornar resultados em pedaços menores.
- **Rate Limiting:** Implemente mecanismos de limitação de taxa para proteger seus serviços de sobrecarga.
- **Monitoramento de Performance:** Por meio de ferramentas CloudWatch ou Datadog pode ser feito o monitoramento dos microserviços

Resiliência

- **Failover para Banco de Dados:** Replicação no PostgreSQL para alta disponibilidade.
- **Failover de Serviços:** Kubernetes gerencia o estado de um pod ou contêiner. Se falhar, outro será reiniciado automaticamente
- **Redundância de Servidores e Serviços:** Distribuir serviços em múltiplas regiões e zonas de disponibilidade
- **Circuit Breakers:** Para tratar falhas temporárias em APIs externas.
- **Backup e Recuperação:** Backups automáticos e testes periódicos de recuperação.
- **Testes de Carga e Stress:** Fazer testes de carga e stress com JMeter podem ser usadas para testar a performance
- **Load Balancer:** Configure ELB para redirecionar automaticamente o tráfego para instâncias ativas e saudáveis.

API Design - Endpoints Principais

Simulação de Empréstimo

Método	EndPoint	Request	Response

POST	/api/v1/emprestimos/simulacoes	{
		"amount": "10000",
		"term": 12,
		"birthDate": "1988-06-10",
		"customerId": "12345"
		}
			{
			"totalPayment": "10300",
			"monthlyPayment": "858.33",
			"totalInterest": "300",
			"simulationId": "hash"
			}

API Design - Endpoints Principais

Proposta de Empréstimo

Método	EndPoint	Request	Response

POST	/api/v1/emprestimos/propostas	{
		"customerId": "12345",
		"simulationId": "hash"
		}
			{
			"proposalId": "prop5678",
			"terms": [
			{
			"amount": "10000",
			"term": 12,
			"rate": 0.05,
			"monthlyPayment": "858.33"
			"totalInterest": "300"
			}
			]
			}

API Design - Endpoints Principais

API Serasa Experian / SPC (consultar scores, riscos, etc)

Método	EndPoint	Request	Response
GET	/api/v1/credits/customers/{id}	PathParameter "12345"	{ "data": { "customer_id": "123456789", "credit_score": 720, "score_range": { "min_score": 300, "max_score": 850 }, "financial_risks": { "delinquency_risk": "low", "default_risk": "medium", "fraud_risk": "low" }, "credit_attributes": { "payment_history": "good", "credit_utilization": "low", "credit_inquiries": "moderate", "open_credit_lines": 5, "recent_activity": "stable" } } }

API Design - Endpoints Principais

API Banco do Brasil / EBANX (consultar taxa de câmbio)

Método	EndPoint	Request	Response
GET	/api/v1/taxes/currencies/{base_currency}/{target_currency}	PathParameter "BRL" e "USD"	{ "data": { "base_currency": "USD", "target_currency": "BRL", "exchange_rate": 5.25, "timestamp": "2024-12-23T15:30:00Z" } }

API Design - Versionamento e Segurança

- **Versionamento:** Versionar os endpoints (api/v1).
- **HTTPS**: Comunicação entre cliente e servidor com HTTPS garante que os dados sejam criptografados
- **Controle de acesso baseado em IP (ACL)**: Limitar quais IPs ou faixas de IP podem acessar a API
- **Criptografia de payload:** Aumenta segurança encapsulando o body da requisição.
- **OAuth 2.0**: Permite que usuários concedam acesso de forma segura a seus recursos sem compartilhar credenciais diretamente.
- **JWT (JSON Web Tokens)**: Transmitir informações entre as partes (client-server) como tokens de acesso.
- **SQL Injection**: Usar consultas parametrizadas para evitar que entradas maliciosas sejam executadas diretamente no banco de dados.
- **XSS (Cross-Site Scripting)**: Implementar técnicas de validação e sanitização de entradas para evitar injeções de scripts maliciosos.
- **CSRF (Cross-Site Request Forgery)**: Utilizar tokens CSRF em formulários e APIs para garantir que as requisições sejam feitas pelo usuário legítimo.
- **Rate Limiting:** Proteção contra abuso, evitando ataques de DDoS.

Motor de Simulação de Empréstimos

O motor de simulação de empréstimos poderia ser um módulo apartado no backend. Dessa forma poderia ser aproveitado por outros sistemas. Ele calcularia diferentes cenários de empréstimos com base nas entradas fornecidas, por exemplo, poderíamos ter além dos parâmetros de entrada valor do empréstimo, data de nascimento do cliente e prazo, alguns outros como tipo de juros (simples ou composto) e forma de pagamento (parcelado ou à vista). Já a resposta traria um objeto contendo pagamentos mensais, juros totais, pagamento total e valor da simulação.

Diretrizes para implementação:

1. **Validação dos dados de entrada:** Realizar validações rigorosas para não ter erros nos cálculos.
2. **Valores pré-calculados:** Certos valores, como taxas de juros anuais ou mensais não precisam ser calculados sempre, podemos armazenar em um cache como o Redis.
3. **Cache de resultados:** Resultados comuns podem ser cacheados para ganharmos agilidade.
4. **Tipos de Dados:** Utilizar BigDecimal nos cálculos pois fornecem maior precisão do que os tipos flutuantes.
5. **Casas decimais:** Arredondamento para duas casas decimais.
6. **Paralelização:** Incluir chunks para paralelizar processamento e assim diminuir o uso de memória e distribuir a carga de trabalho.
7. **Processamento Assíncrono:** Executar em segundo plano os cálculos.
8. **Testes de carga e performance:** Garantir que o sistema pode lidar com um número elevado de simulações simultâneas sem afetar a performance.