Modelo

Setembro de 2024

Sobre o arc42

arc42, o template para documentação de software e arquitetura de sistemas.

Versão do template 8.2 PT. (baseado na versão AsciiDoc), Setembro de 2024

Criado, mantido e © pelo Dr. Peter Hruschka, Dr. Gernot Starke e colaboradores. Veja https://arc42.org.

Equipe: Flávio José Cordeiro, Eduardo do Prado Bonacin, Erick de Menezes e Gabriel Trevisan.

Introdução e Objetivos

- **Visão geral**: O Prestus é uma plataforma digital que conecta estabelecimentos (empregadores) a trabalhadores temporários (garçons, DJs, bartenders, etc.), oferecendo gestão de plantões, segurança de pagamento e reputação.

- **Objetivo**: Reduzir a burocracia e aumentar a confiança no processo de contratação temporária em eventos e estabelecimentos.

Visão Geral dos Requisitos

- **Funcionais (RF)**: autenticação, perfis, publicação de vagas, candidatura/aceitação, check-in geolocalizado, chat, pagamentos via Stripe, taxas do sistema, avaliações, penalidades, notificações, segurança (logs, KYC/KYB), políticas de cancelamento/aceitação.

- **Não funcionais (RNF)**: segurança (LGPD,), disponibilidade (99,9%), escalabilidade (10.000 usuários simultâneos), performance (resposta <2s),

acessibilidade, manutenção (testes >80%, modularização), compatibilidade (web + mobile).

Objetivos de Qualidade

- **Segurança**: Custódia de pagamento e validação de documentos.
- **Escalabilidade**: Crescer horizontalmente sem afetar performance.
- **Disponibilidade**: Sistema sempre acessível em eventos noturnos (pico de uso).
- **Usabilidade**: App responsivo e intuitivo.
- **Transparência**: Histórico financeiro e reputacional confiável.

Partes Interessadas

Função/Nome	***Contato***	***Expectativas***
Trabalhador (Cliente)	*Cadastro no app / e-mail*	*Conseguir vagas rápidas, com segurança, pagamento garantido e histórico de reputação.*
Empregador (Estabelecimento)	*Cadastro no app / e-mail*	*Encontrar mão de obra confiável para plantões, com pagamento controlado e menos burocracia.*
Equipe de Negócio (Fundadores do Prestus)	*erickdemenezes@icloud.com*	*Monetizar o sistema via taxa de intermediação (30%), atrair usuários e garantir sustentabilidade.*
Equipe de Desenvolvimento	*Slack / GitHub*	*Ter arquitetura clara, modular, bem documentada e de fácil manutenção.*
Equipe de Operações / Suporte	*suporte@prestus.com*	*Resolver problemas de usuários, lidar com disputas (cancelamentos, penalidades) e manter o sistema estável.*
Gateway de Pagamento (Stripe)	*docs.stripe.com / API*	*Garantir integração estável, confiável e compliance com normas financeiras.*
Órgãos Reguladores (LGPD, Receita Federal, Prefeitura)	*canais oficiais*	*Que o sistema esteja em conformidade com exigências legais (proteção de dados, notas fiscais, licenças).*

Restrições Arquiteturais

- **Frontend**: React.js.
- **Backend**: Node.js.
- **Banco de dados**: Firebase
- **Hospedagem**: AWS
- **Pagamentos**: Stripe API.
- **Geolocalização**: Google Maps API.

Contexto e Escopo

Contexto Negocial

Ator Externo	***Interação com o Prestus***
Trabalhador	*Cria perfil, busca vagas, se candidata/aceita, faz check-in/out, recebe pagamento, avalia empregadores.*
Empregador	*Cria perfil do estabelecimento, publica vagas, aprova candidatos, efetua pagamentos, avalia trabalhadores.*
Gateway de Pagamento (Stripe)	*Custódia do valor, repasse para trabalhador, cobrança de taxas.*
Serviço de Geolocalização (Google Maps API)	*Permite check-in geolocalizado, busca por proximidade.*
Serviço de Notificações (Firebase / SMS / E-mail)	*Envia alertas de candidaturas, aprovações, lembretes e confirmações.*

Contexto Técnico

Componente Técnico	*Papel no Sistema*
Frontend (React.js)	*Interface web/mobile para trabalhadores e empregadores.*
Backend (Node.js/Express)	*Regras de negócio, autenticação, APIs REST.*
Banco de Dados (Firebase)	*Armazena usuários, vagas, candidaturas, avaliações, transações.*
Stripe API	*Pagamentos (custódia, repasse, recibos).*
Google Maps API	*Geolocalização para vagas e check-in.*

Estratégia de Solução

Decisões Tecnológicas Fundamentais

Frontend: React.js foi escolhido para garantir interfaces responsivas tanto web quanto mobile, com componentes reutilizáveis e desenvolvimento ágil.

Backend: Node.js/Express oferece escalabilidade horizontal, ecossistema robusto para APIs REST, facilita integração com serviços externos (Stripe, Google Maps) e permite desenvolvimento full-stack JavaScript.

Arquitetura: Microserviços com API Gateway centralizado para autenticação, rate limiting e roteamento. Separação clara entre camadas de apresentação, negócio e dados.

Pagamentos: Stripe API garante compliance PCI DSS, custódia segura, split automático e reduz complexidade de desenvolvimento financeiro.

Geolocalização: Google Maps API para validação precisa de check-in/out e busca por proximidade, essencial para confiabilidade do sistema.

Padrões Arquiteturais

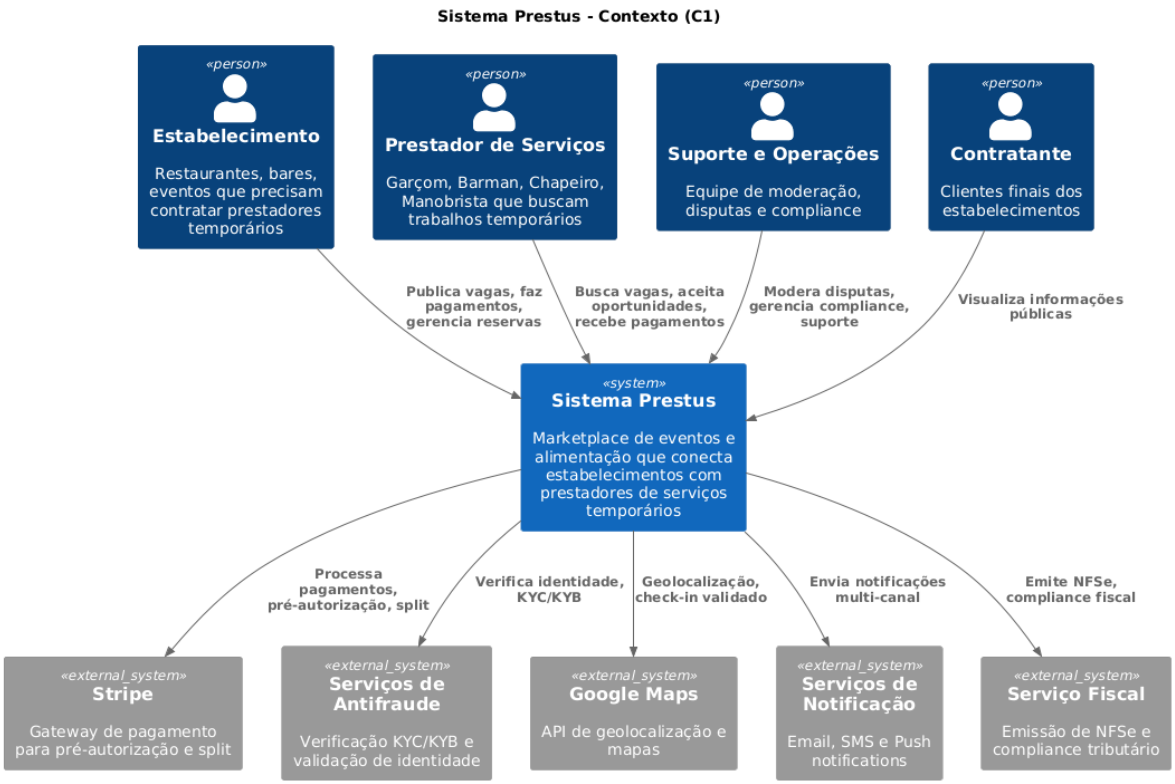
API Gateway Pattern: Centraliza cross-cutting concerns (autenticação, logging, rate limiting) e desacopla clientes dos serviços internos.

Repository Pattern: Isola lógica de acesso a dados, facilitando testes e mudanças de persistência.

Adapter Pattern: Para integração com serviços externos (Stripe, KYC, Maps), permitindo substituição sem impacto no core.

Visão de Blocos de Construção

Visão Sistêmica Geral de Caixa Branca



Motivação

O diagrama demonstra **como o Prestus interage com atores humanos e sistemas externos**, garantindo que os principais fluxos de negócio estejam representados.
O objetivo é fornecer uma **visão clara de fronteira**: o que pertence ao Prestus e o que está fora, mas influencia seu funcionamento (pagamentos, geolocalização, notificações e compliance).

Blocos de Construção Contidos

Dentro desta visão, o sistema Prestus é considerado uma **caixa preta** que:

- Gerencia vagas temporárias.
- Faz a ponte entre **Estabelecimentos** e **Prestadores de Serviços**.
- Lida com suporte, compliance e disputas.
- Conecta-se a serviços externos para pagamentos, geolocalização, notificações e obrigações fiscais.

Caixas pretas externas:

1. **Estabelecimento** → publica vagas, gerencia reservas, realiza pagamentos.
2. **Prestador de Serviços** → busca vagas, aceita oportunidades, recebe pagamentos.
3. **Suporte e Operações** → modera disputas e garante conformidade.
4. **Contratante** (clientes finais) → consome serviços dos estabelecimentos, visualiza informações.
5. **Stripe** → processa pagamentos com pré-autorização e split.
6. **Serviços de Antifraude** → valida KYC/KYB.
7. **Google Maps** → fornece geolocalização e check-in validado.
8. **Serviços de Notificação** → envia alertas por push, e-mail e SMS.
9. **Serviço Fiscal** → gera NFSe e cuida do compliance tributário.

Interfaces Importantes

1. ***Interface Usuário (Frontend Web/Mobile)*** *→ interação entre Prestus, Estabelecimento e Prestadores.*
2. ***Interface de Pagamento (Stripe API)*** *→ autorização, split, repasse de valores.*
3. ***Interface de Antifraude (KYC/KYB API)*** *→ validação de documentos e identidades.*
4. ***Interface de Geolocalização (Google Maps API)*** *→ validação de check-in/out e busca por proximidade.*
5. ***Interface de Notificação (Firebase)***
6. ***Interface Fiscal (NFSe API)*** *→ emissão de notas fiscais e integração tributária.*

Caixa Preta 1 – Estabelecimento

- **Propósito**: Criar e publicar vagas, contratar prestadores, realizar pagamentos.
- **Interfaces**: Web/Mobile → Prestus.

Caixa Preta 2 – Prestador de Serviços

- **Propósito**: Buscar oportunidades de trabalho temporário, aceitar plantões, receber pagamentos.
- **Interfaces**: Mobile → Prestus.

Caixa Preta 3 – Suporte e Operações

- **Propósito**: Moderar disputas, analisar justificativas de penalidades, garantir compliance.
- **Interfaces**: Painel administrativo → Prestus.

Caixa Preta 4 – Contratante (Cliente final do estabelecimento)

- **Propósito**: Consumir serviços do estabelecimento; visualizar informações públicas do prestador.

- **Interfaces**: Indireta, via informações publicadas no Prestus.

Caixa Preta 5 – Stripe (Gateway de Pagamento)

- **Propósito**: Processamento de pagamentos, custódia, split automático.
- **Interfaces**: API REST (Prestus ↔ Stripe).

Caixa Preta 6 – Serviços de Antifraude

- **Propósito**: Validar identidade de usuários e estabelecimentos (KYC/KYB).
- **Interfaces**: API REST (Prestus ↔ Antifraude).

Caixa Preta 7 – Google Maps API

- **Propósito**: Geolocalização de vagas, check-in e check-out validado.
- **Interfaces**: API REST (Prestus ↔ Google Maps).

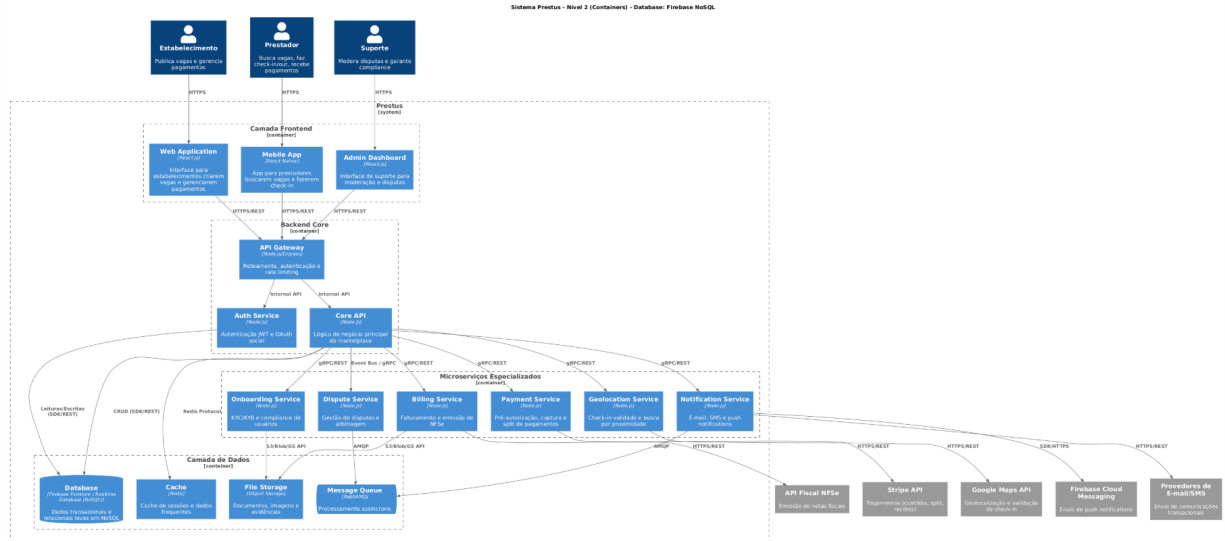
Caixa Preta 8 – Serviços de Notificação

- **Propósito**: Enviar notificações (push, SMS, e-mail) para usuários e empregadores.
- **Interfaces**: API REST (Firebase)

Caixa Preta 9 – Serviço Fiscal

- **Propósito**: Emissão de notas fiscais eletrônicas (NFSe) e compliance tributário.
- **Interfaces**: API REST (Prestus ↔ Prefeitura / Receita).

Nível 2



Caixa Branca 1 – Web Application

- **Propósito:** Interface para estabelecimentos criarem vagas e gerenciarem pagamentos.
- **Interfaces:** Web → Prestus.

Caixa Branca 2 – Mobile App

- **Propósito:** App para prestadores buscarem vagas e realizarem check-in.
- **Interfaces:** Mobile → Prestus.

Caixa Branca 3 – Admin Dashboard

- **Propósito:** Interface de suporte para moderação e resolução de disputas.
- **Interfaces:** Web → Prestus.

Caixa Branca 4 – API Gateway

- **Propósito:** Roteamento, autenticação e controle de tráfego (rate limiting).
- **Interfaces:** HTTPS/REST → Frontend, Internal API → Microserviços.

Caixa Branca 5 – Core API

- **Propósito:** Lógica de negócio principal do marketplace.
- **Interfaces:** Internal API → API Gateway, gRPC/Event Bus → Microserviços.

Caixa Branca 6 – Auth Service

- **Propósito:** Autenticação de usuários via JWT e OAuth social.
- **Interfaces:** Internal API → API Gateway, gRPC → Microserviços.

Caixa Branca 7 – Payment Service

- **Propósito:** Pré-autorização, captura e split de pagamentos.
- **Interfaces:** gRPC → Core API, HTTPS/REST → Stripe API.

Caixa Branca 8 – Geolocation Service

- **Propósito:** Validação de check-in e geolocalização.
- **Interfaces:** gRPC → Core API, HTTPS/REST → Google Maps API.

Caixa Branca 9 – Notification Service

- **Propósito:** Envio de notificações (e-mail, SMS, push).
- **Interfaces:** gRPC → Core API, HTTPS/REST → AWS SES.

Caixa Branca 10 – Dispute Service

- **Propósito:** Gestão de disputas e arbitragem.
- **Interfaces:** gRPC → Core API, AMQP → Message Queue.

Caixa Branca 11 – Billing Service

- **Propósito:** Faturamento e emissão de NFSe.
- **Interfaces:** gRPC → Core API, HTTPS/REST → API Fiscal NFSe.

Caixa Branca 12 – Onboarding Service

- **Propósito:** KYC/KYB e compliance de usuários.
- **Interfaces:** gRPC → Core API, HTTPS/REST → Serviços de Antifraude.

Caixa Branca 13 – Message Queue (RabbitMQ)

- **Propósito:** Processamento assíncrono de mensagens.
- **Interfaces:** AMQP → Microserviços.

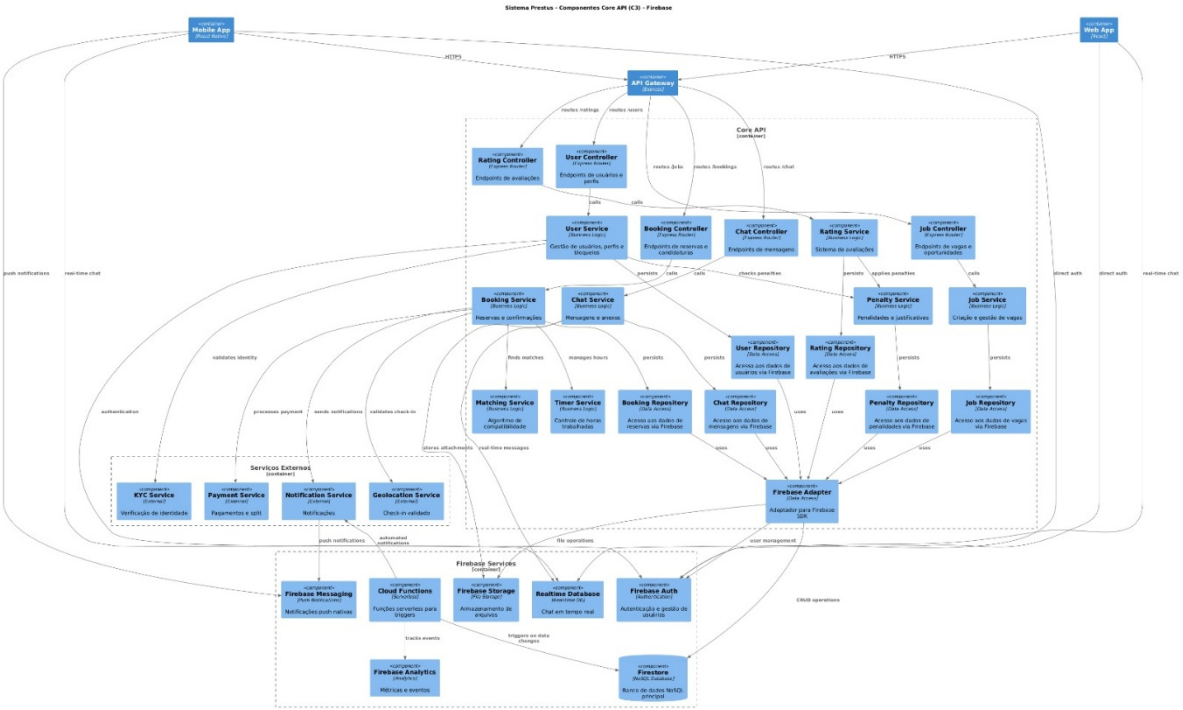
Caixa Branca 15 – Database (Firebase)

- **Propósito:** Armazenar os dados da aplicação, incluindo informações sobre usuários, vagas, candidaturas, avaliações e transações.
- **Interfaces:** O acesso é realizado através do protocolo nativo do Firebase SDK, consumido pelos microserviços do backend, como a Core API, Auth Service e Onboarding Service.

Caixa Branca 16 – Cache (Redis)

- **Propósito:** Cache de sessões e dados frequentes.
- **Interfaces:** Redis Protocol → Microserviços.

Nível 3



Nível 3 - Diagrama de Componentes (Component Level) - Sistema Prestus

Container: Core API (Node.js/Express)

Componente 1 – API Gateway

Propósito: Gerenciar autenticação, autorização e roteamento de todas as requisições HTTP para os serviços internos.
Interface: REST API (HTTP/HTTPS) ← Mobile App, Web App

Componente 2 – User Controller

Propósito: Processar requisições relacionadas a usuários (cadastro, login, perfil) e gerenciar sessões.
Interface: HTTP REST API ← API Gateway

Componente 3 – Rating Controller

Propósito: Gerenciar sistema de avaliações entre prestadores e estabelecimentos.
Interface: HTTP REST API ← API Gateway

Componente 4 – Booking Controller

Propósito: Controlar criação, aprovação, cancelamento e gestão de plantões/vagas.
Interface: HTTP REST API ← API Gateway

Componente 5 – Chat Controller

Propósito: Gerenciar sistema de mensagens entre prestadores e estabelecimentos.
Interface: WebSocket/HTTP REST API ← API Gateway

Componente 6 – Job Controller

Propósito: Processar operações relacionadas a vagas de trabalho e candidaturas.
Interface: HTTP REST API ← API Gateway

Container: Business Services Layer

Componente 7 – User Service

Propósito: Implementar regras de negócio para gestão de usuários, perfis e autenticação.
Interface: API interna ← Controllers

Componente 8 – Booking Service

Propósito: Executar lógica de negócio para reservas, confirmações e check-ins/check-outs.
Interface: API interna ← Booking Controller

Componente 9 – Chat Service

Propósito: Processar mensagens, notificações em tempo real e histórico de conversas.
Interface: API interna ← Chat Controller

Componente 10 – Rating Service

Propósito: Calcular e gerenciar sistema de reputação e avaliações.
Interface: API interna ← Rating Controller

Componente 11 – Penalty Service

Propósito: Aplicar penalidades, multas e justificativas do sistema.
Interface: API interna ← Job Controller, Booking Controller

Componente 12 – Job Service

Propósito: Gerenciar lógica de criação de vagas, candidaturas e gestão de oportunidades.
Interface: API interna ← Job Controller

Container: Data Access Layer

Componente 13 – User Repository

Propósito: Acessar e manipular dados de usuários na base de dados Firebase.
Interface: Database queries ← User Service

Componente 14 – Booking Repository

Propósito: Gerenciar persistência de dados de reservas e plantões.
Interface: Database queries ← Booking Service

Componente 15 – Rating Repository

Propósito: Armazenar e recuperar dados de avaliações e reputação.
Interface: Database queries ← Rating Service

Componente 16 – Chat Repository

Propósito: Persistir mensagens e histórico de conversas.
Interface: Database queries ← Chat Service

Componente 17 – Penalty Repository

Propósito: Armazenar dados de penalidades e justificativas.
Interface: Database queries ← Penalty Service

Componente 18 – Job Repository

Propósito: Gerenciar dados de vagas e candidaturas na base de dados.
Interface: Database queries ← Job Service

Container: External Adapters

Componente 19 – Firebase Adapter

Propósito: Adaptar comunicação com serviços Firebase (Auth, Storage, Realtime Database).
Interface: Firebase SDK ← Repositories, HTTPS → Firebase Services

Componente 20 – Matching Service

Propósito: Algoritmo de compatibilidade entre prestadores e vagas disponíveis.
Interface: API interna ← Job Service

Componente 21 – Timer Service

Propósito: Controlar horários de plantões, lembretes e timeouts do sistema.
Interface: API interna ← Booking Service

Componente 22 – Notification Service

Propósito: Gerenciar envio de notificações push, SMS e e-mail.
Interface: API interna ← Business Services, HTTP → Firebase Cloud Messaging

Componente 23 – Geolocation Service

Propósito: Validar check-ins gelocalizados e buscar vagas por proximidade.
Interface: HTTP REST API → Google Maps API

Componente 24 – Payment Service

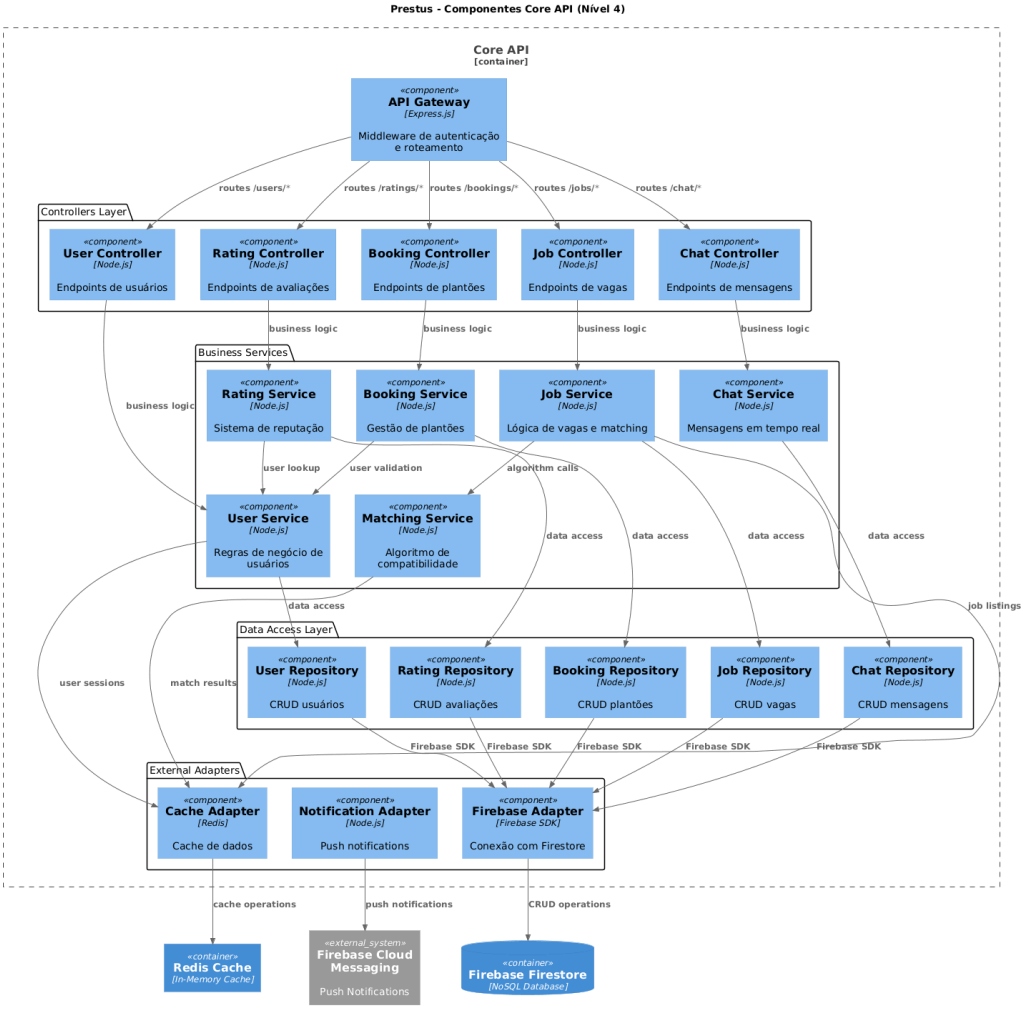
Propósito: Processar pagamentos, custódia de valores e repasses via Stripe.
Interface: HTTP REST API → Stripe API

Componente 25 – KYC Service

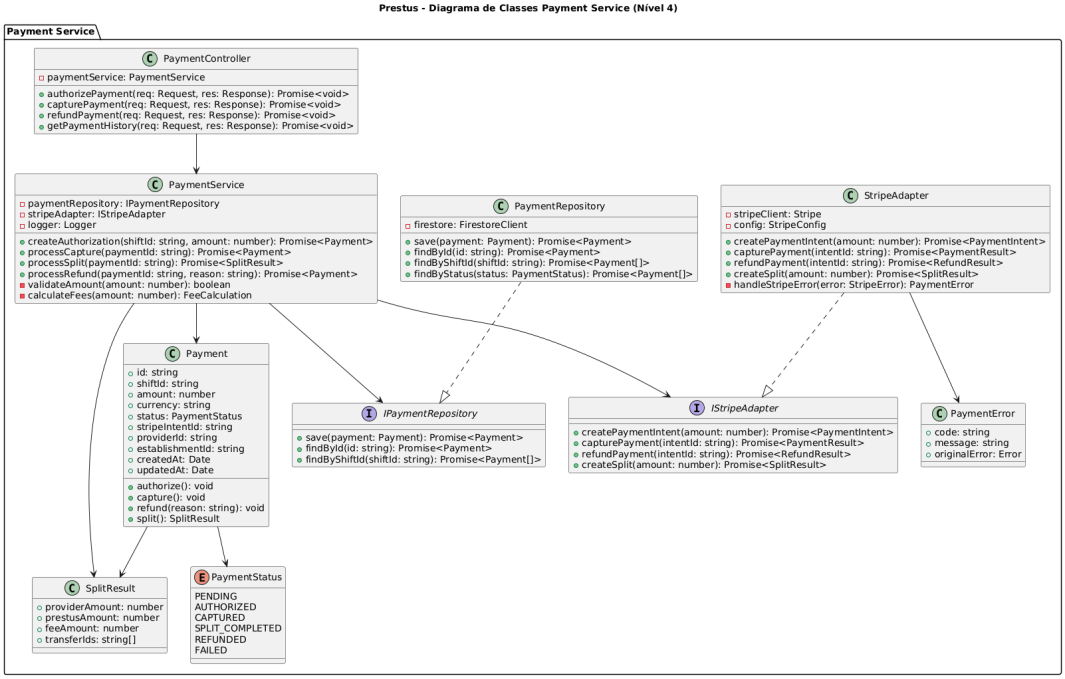
Propósito: Verificar identidade de usuários e estabelecimentos (Know Your Customer).
Interface: HTTP REST API → Serviços de Antifraude7

Nível 4

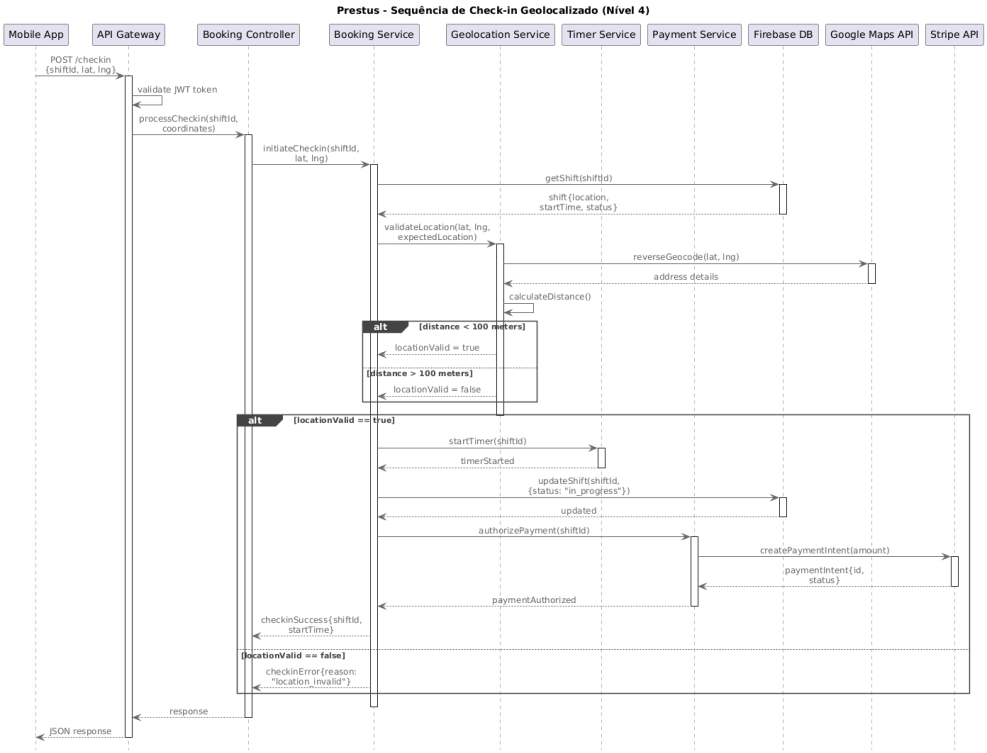
Componentes



Classes



Sequência



Visão de Tempo de Execução

Cenário de Tempo de Execução 1

Contexto: Estabelecimento publica vaga, prestador se candidata e é aprovado para plantão.

Fluxo:

1. **Estabelecimento** via Web App → publica vaga no **Core API**
2. **Core API** → persiste vaga no **Database** via **Job Repository**
3. **Notification Service** → envia push para prestadores qualificados
4. **Prestador** via Mobile App → busca vagas no **Core API**
5. **Core API** → **Matching Service** retorna vagas compatíveis
6. **Prestador** → se candidata via **Job Controller**
7. **Estabelecimento** → aprova candidato via **Booking Controller**
8. **Payment Service** → pré-autoriza pagamento no **Stripe**
9. **Notification Service** → confirma aprovação para ambas as partes

Cenário de Tempo de Execução 2

Contexto: Prestador chega ao local e faz check-in para iniciar plantão.

Fluxo:

1. **Prestador** → inicia check-in via Mobile App
2. **API Gateway** → valida autenticação e roteia para **Booking Controller**
3. **Geolocation Service** → verifica coordenadas via **Google Maps API**
4. **Booking Service** → valida horário e localização do plantão
5. **Timer Service** → inicia cronômetro do plantão
6. **Prestador** → realiza check-out ao final
7. **Booking Service** → calcula horas trabalhadas
8. **Payment Service** → captura valor pré-autorizado no **Stripe**
9. **Payment Service** → executa split (70% prestador, 30% Prestus)

10. **Notification Service** → confirma pagamento processado

Cenário de Tempo de Execução 3

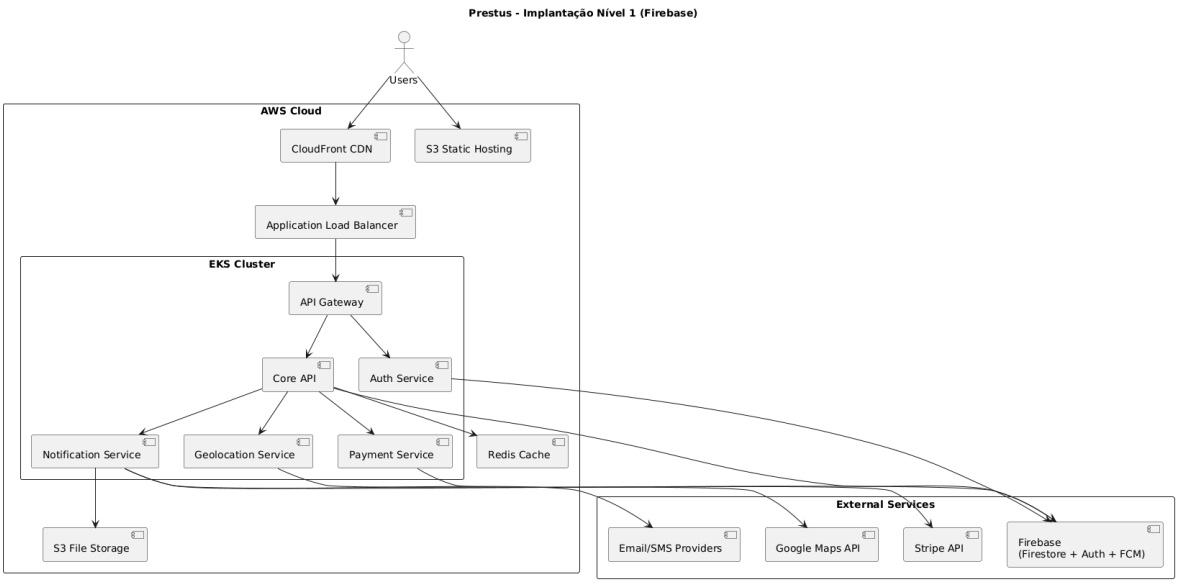
Contexto: Cancelamento de plantão com aplicação de penalidade.

Fluxo:

1. **Prestador/Estabelecimento** → solicita cancelamento
2. **Booking Controller** → verifica política de cancelamento
3. **Penalty Service** → calcula penalidade baseada em prazo
4. **Dispute Service** → registra disputa se houver contestação
5. **Message Queue** → processa notificação assíncrona
6. **Suporte** via Admin Dashboard → analisa justificativa
7. **Dispute Service** → resolve disputa e atualiza reputação
8. **Payment Service** → processa reembolso/cobrança conforme decisão

Visão de Implantação

Nível de Infraestrutura 1



Motivação: O Prestus opera como marketplace distribuído, exigindo alta disponibilidade, escalabilidade e segurança para transações financeiras.

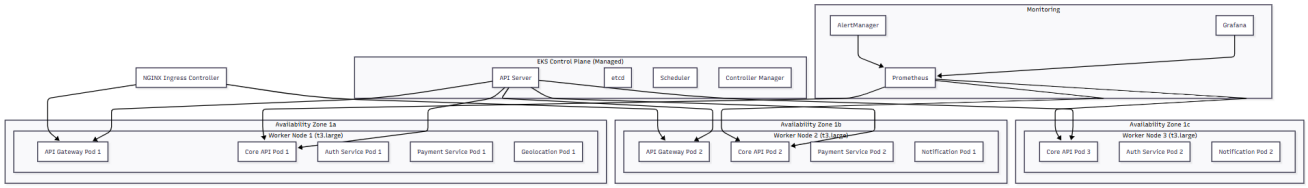
Características de Qualidade:

- **Disponibilidade**: 99.9% uptime com redundância geográfica
- **Escalabilidade**: Auto-scaling baseado em métricas de CPU/memória
- **Segurança**: HTTPS, WAF, DDoS protection, compliance LGPD

Mapeamento:

- **Frontend (React)** → CDN + Static Hosting (S3/CloudFront)
- **API Gateway + Core API** → Container Orchestration (EKS/AKS)
- **Microserviços** → Kubernetes pods com health checks
- **Cache** → ElastiCache/Azure Redis
- **File Storage** → S3/Azure Blob para documentos KYC

Nível de Infraestrutura 2



EKS Control Plane (Gerenciado pela AWS):

- **API Server**: Endpoint central para operações Kubernetes
- **etcd**: Armazenamento de configuração distribuído
- **Scheduler**: Alocação automática de pods nos worker nodes
- **Controller Manager**: Monitora e mantém estado desejado dos recursos

Worker Nodes (EC2 t3.large):

- **Distribuição**: 3 worker nodes em 3 Availability Zones
- **Especificação**: 2 vCPU, 8GB RAM por node
- **Auto Scaling**: Horizontal Pod Autoscaler baseado em CPU/memória

Pod Distribution Strategy:

- **API Gateway**: 2 réplicas distribuídas em AZ diferentes

- **Core API**: 3 réplicas (1 por AZ) para balanceamento
- **Auth Service**: 2 réplicas com anti-affinity rules
- **Payment Service**: 2 réplicas em AZs distintas
- **Notification Service**: 2 réplicas para redundância
- **Geolocation Service**: 1 réplica (pode escalar sob demanda)

Conceitos Transversais

Segurança e Compliance

Autenticação: JWT tokens com refresh mechanism, OAuth social login, MFA para operações críticas.

Autorização: RBAC (Role-Based Access Control) com diferentes níveis: Prestador, Estabelecimento, Admin, Suporte.

LGPD Compliance:

- Consentimento explícito para tratamento de dados
- Direito ao esquecimento com anonymização
- Logs de auditoria para todas as operações
- Criptografia de dados sensíveis (documentos KYC)

PCI DSS: Dados de cartão nunca armazenados, apenas tokens Stripe utilizados.

Tratamentos de Erros e Logging

Logging Estruturado: JSON format com correlationId para tracing distribuído.

Níveis de Log: ERROR (falhas críticas), WARN (situações degradadas), INFO (operações importantes), DEBUG (desenvolvimento).

Error Handling: Circuit breaker pattern para serviços externos, retry com backoff exponencial, fallback graceful.

Performance e Caching

Cache Strategy: Redis para sessões, dados frequentes (perfis, avaliações), resultados de matching.

Database Optimization: Indexes otimizados, query optimization, connection pooling.

CDN: Assets estáticos, imagens de perfil e documentos via CloudFront/Azure CDN.

Monitoramento e Observabilidade

Métricas: Prometheus/Grafana para infraestrutura, APM para aplicação

Health Checks: Endpoints /health em todos os serviços, incluindo dependências externas.

Alerting: Notificações automáticas para falhas críticas, degradação de performance, anomalias de uso.

Decisões Arquiteturais

1. **Frontend com React.js**: A escolha do React.js visa unificar o desenvolvimento de interfaces responsivas para web e mobile. Esta decisão permite a reutilização de componentes, o que acelera o processo de desenvolvimento e garante uma experiência de usuário consistente em diferentes plataformas.
2. **Backend com Node.js/Express**: Optou-se por Node.js com o framework Express para o backend. Essa tecnologia foi selecionada por sua capacidade de escalabilidade horizontal, pelo robusto ecossistema para a construção de APIs REST e por facilitar a integração com serviços de terceiros como Stripe e Google Maps. Além disso, padroniza a linguagem de programação em JavaScript em todo o stack de desenvolvimento (full-stack).
3. **Arquitetura de Microserviços com API Gateway**: A arquitetura do sistema é baseada em microserviços, com um API Gateway centralizando a autenticação, o controle de tráfego (rate limiting) e o roteamento de requisições. Essa abordagem promove a separação clara entre as camadas de apresentação, negócio e dados , desacoplando os clientes dos serviços internos e facilitando a manutenção e a evolução independente de cada serviço.
4. **Integração de Pagamentos via Stripe API**: Para o processamento de pagamentos, a escolha da Stripe API foi estratégica. A decisão se baseia na garantia de conformidade com o padrão PCI DSS, na segurança da

custódia de valores e na funcionalidade de divisão automática de pagamentos (split), o que reduz significativamente a complexidade do desenvolvimento financeiro interno.

Uso da Google Maps API para Geolocalização: A integração com a Google Maps API é essencial para as funcionalidades de check-in/out e busca de vagas por proximidade. A precisão e a confiabilidade deste serviço foram fatores determinantes para garantir a integridade e a confiança nas operações da plataforma.

5. **Banco de Dados Firebase**: O Firebase foi escolhido para o armazenamento de dados do sistema.
6. **Padrões de Projeto (Repository e Adapter)**: Foram adotados padrões arquiteturais para garantir um design de software robusto e de fácil manutenção.

   Repository Pattern é utilizado para isolar a lógica de acesso a dados, o que facilita a realização de testes e eventuais mudanças na tecnologia de persistência. O

   Adapter Pattern é aplicado na integração com serviços externos (Stripe, Google Maps, KYC), permitindo que esses serviços possam ser substituídos no futuro sem causar grande impacto no núcleo da aplicação.

Requisitos de qualidade

Árvore de qualidade

- **Segurança**
  - **Confidencialidade**:
    - Proteção de dados pessoais em conformidade com a LGPD.
    - Criptografia para dados sensíveis, como documentos de KYC/KYB.
    - Não armazenamento de dados de cartão de crédito, utilizando apenas tokens da Stripe.
  - **Integridade**:
    - Garantia da custódia de pagamento segura através da Stripe.
    - Validação de identidade de usuários e estabelecimentos (KYC/KYB) para mitigar fraudes.
  - **Autenticação e Autorização**:
    - Autenticação via tokens JWT e OAuth para login social.

        - Controle de acesso baseado em papéis (RBAC) para diferentes perfis de usuário (Prestador, Estabelecimento, Admin).
- **Escalabilidade**
    - **Horizontal**: A arquitetura deve permitir o crescimento horizontal para suportar o aumento de carga.
    - **Carga**: Suporte para até 10.000 usuários simultâneos sem degradação da performance.
    - **Componentes**: Uso de orquestração de contêineres (Kubernetes) e serviços gerenciados (RDS, ElastiCache) que suportam auto-scaling.
- **Disponibilidade**
    - **Uptime**: O sistema deve manter uma disponibilidade de 99,9%.
    - **Resiliência**: Alta disponibilidade, especialmente durante os horários de pico de uso em eventos noturnos.
    - **Infraestrutura**: Implementação com redundância geográfica para garantir a continuidade do serviço.
- **Performance**
    - **Tempo de Resposta**: As requisições da API devem ter um tempo de resposta inferior a 2 segundos.
    - **Otimização**:
        - Uso de cache (Redis) para dados acessados com frequência, como sessões e perfis.
        - Entrega de assets estáticos via CDN para reduzir a latência.
- **Manutenibilidade**
    - **Modularidade**: Arquitetura de microserviços que permite o desenvolvimento e a implantação independentes.
    - **Testabilidade**: Cobertura de testes superior a 80%.
    - **Documentação**: Arquitetura clara e bem documentada para facilitar a manutenção pela equipe de desenvolvimento.
- **Usabilidade**
    - **Intuitividade**: A interface do aplicativo deve ser intuitiva para os usuários.
    - **Responsividade**: O aplicativo deve ser responsivo, adaptando-se a diferentes dispositivos web e mobile.

Cenários de Qualidade

- **Cenário 1: Escalabilidade sob Carga em Horário de Pico**
    - **Estímulo**: Em uma noite de sábado, às 21h, o número de usuários ativos realizando buscas de vagas e check-ins aumenta para 10.000 conexões simultâneas.
    - **Sistema**: Prestus.
    - **Resposta**: O sistema escala horizontalmente os pods dos microserviços (Core API, Geolocation Service) automaticamente. O

tempo de resposta para as buscas de vagas permanece abaixo de 2 segundos, e os check-ins são processados sem falhas.
    - **Métrica**:
        - Tempo de resposta da API < 2s.
        - Taxa de sucesso das transações > 99,9%.
        - Uso de CPU dos contêineres se estabiliza abaixo de 80% após o auto-scaling.

- **Cenário 2: Disponibilidade durante Falha de um Componente**
    - **Estímulo**: O serviço de notificações (Notification Service) sofre uma falha inesperada e fica indisponível.
    - **Sistema**: Prestus.
    - **Resposta**: O API Gateway e o Core API continuam operando normalmente. Os usuários conseguem publicar vagas, candidatar-se e realizar check-ins. As notificações são enfileiradas na Message Queue para serem processadas assim que o serviço for restabelecido. O monitoramento alerta a equipe de operações sobre a falha em menos de 5 minutos.
    - **Métrica**:
        - Disponibilidade das APIs principais > 99,9%.
        - Tempo para detecção e alerta da falha < 5 minutos.
        - Nenhuma perda de dados de notificação.

- **Cenário 3: Segurança na Validação de um Novo Prestador**
    - **Estímulo**: Um novo prestador de serviços realiza o cadastro e envia seus documentos para validação (KYC).
    - **Sistema**: Prestus.
    - **Resposta**: Os documentos são enviados de forma segura via HTTPS para o File Storage (AWS S3), sendo criptografados. A Onboarding Service aciona o serviço externo de antifraude para validação da identidade. A comunicação com o serviço externo é protegida. O acesso aos documentos é restrito a administradores autorizados.
    - **Métrica**:
        - Todos os dados em trânsito e em repouso são criptografados.
        - Logs de auditoria registram todas as etapas do processo de validação.
        - Acesso aos documentos restrito conforme as políticas de RBAC.

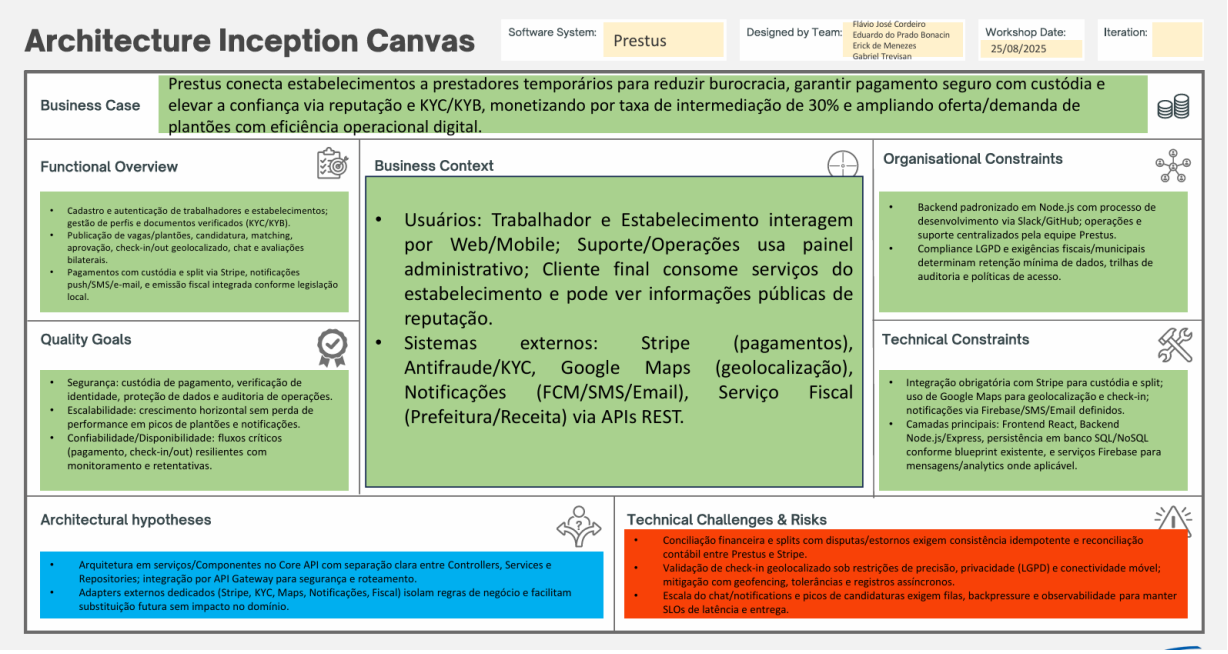
- **Cenário 4: Manutenibilidade ao Alterar uma Regra de Negócio**

    - **Estímulo**: A equipe de negócio decide alterar a política de penalidades por cancelamento, ajustando os prazos e valores.
    - **Sistema**: Prestus.
    - **Resposta**: Um desenvolvedor precisa modificar apenas o Penalty Service e o Dispute Service. A lógica está isolada e não afeta outros microserviços como Payment Service ou Booking Service. Novos testes unitários e de integração são criados, e a cobertura de testes do serviço modificado permanece acima de 80%. A nova versão do serviço é implantada de forma independente.
    - **Métrica**:
        - Número de microserviços que necessitam de alteração: 2.
        - Tempo de implantação da mudança em produção < 1 hora.
        - Nenhum impacto negativo (regressão) em outros serviços do sistema.

Riscos e Débitos Técnicos

- **Riscos Técnicos**:
    - **Dependência Excessiva de Serviços Externos**: A operação do Prestus depende criticamente de terceiros como Stripe, Google Maps e serviços de antifraude. Uma falha, mudança de API ou alteração nos preços desses provedores pode impactar diretamente a disponibilidade e o custo operacional do sistema.
    - **Complexidade da Arquitetura de Microserviços**: Embora escalável, uma arquitetura de microserviços introduz complexidade em monitoramento, depuração distribuída (tracing) e gerenciamento de transações entre serviços. Uma falha em cascata é um risco real se padrões como Circuit Breaker não forem implementados corretamente.
    - **Segurança em Múltiplos Pontos**: A natureza distribuída aumenta a superfície de ataque. Garantir a segurança em cada microserviço, na comunicação interna (gRPC, AMQP) e nas integrações externas é um desafio complexo.
- **Débitos Técnicos**:
    - Disputas pode se tornar extremamente complexa. A implementação inicial pode ser simplificada, mas acumulará um débito técnico à medida que novas regras e cenários de exceção surjam, exigindo refatoração futura para evitar que se torne um gargalo.

Canvas



Glossário

Termo	Definição
Prestador	Trabalhador temporário (garçon, DJ, bartender) que oferece serviços via plataforma
Estabelecimento	Empresa contratante que publica vagas e contrata prestadores
Plantão	Período de trabalho temporário com horário definido (ex: 19h-02h)
Check-in/Check-out	Processo de registro geolocalizado de início/fim de plantão
KYC/KYB	Know Your Customer/Business - validação de identidade de usuários/empresas
Split Payment	Divisão automática de pagamento (70% prestador, 30% Prestus)
Custódia	Retenção temporária do valor pelo Stripe até confirmação do serviço

Termo	Definição
Matching	Algoritmo que conecta prestadores a vagas compatíveis
Penalidade	Multa aplicada por cancelamento tardio ou não comparecimento
Reputação	Sistema de avaliações bidirecionais entre prestadores e estabelecimentos
NFSe	Nota Fiscal de Serviços Eletrônica para compliance fiscal
LGPD	Lei Geral de Proteção de Dados - regulamentação brasileira de privacidade