Trabalho Final

ARQUITETURA DE SOFTWARE

Grupo:
Arthur Nucada Félix de Souza - 202201683
José Alves de Oliveira Neto - 202201699
Lucas Gabriel Nunes Alves - 202201703
Sophia Fernandes Magalhães Almeida - 202201713
Victor Martins Vieira - 202204532

Visão Geral do Sistema

- Plataforma digital integrada para bem-estar no trabalho
- Foco em escuta ativa, suporte emocional e cultura organizacional positiva
- Resposta à crescente demanda por ambientes mais saudáveis e humanizados
- Pandemia reforçou a urgência por equilíbrio e canais empáticos
- Atua além do RH: promove saúde integral e sustentabilidade organizacional
- Gera dados úteis para decisões de gestão e prevenção de riscos

Objetivos Estratégicos e Público-Alvo

- Promover escuta segura e anônima dos colaboradores
- Facilitar ações de bem-estar como consultas e eventos
- Aumentar engajamento com gamificação e personalização
- Gerar inteligência organizacional com relatórios e insights
- Integrar especialistas com confidencialidade e controle
- Reduzir custos com saúde via prevenção e intervenções
- Público-alvo: empresas médias e grandes, RHs, psicólogos e consultores
- Acesso multiplataforma e modelo modular para adoção progressiva

Requisitos Arquiteturalmente Significativos (ASRs)e Decisões Arquiteturais

- **10 Requisitos Funcionais** priorizados
- **4 Requisitos de Qualidade** críticos
- Abordagem **iterativa** de implementação
- Foco em **segurança** e **escalabilidade**

- **Microserviços** para flexibilidade
- **API Gateway** centralizado
- **PostgreSQL** + **Redis** para dados
- **Docker** para containerização

Requisitos Funcionais Prioritários

Alta Prioridade

- **RF-01:** Feedback e Reportes Anônimos
- **RF-02:** Agendamento de Reuniões de Bem-estar
- **RF-03:** Consultas Virtuais
- **RF-04:** Recomendações Personalizadas

Média Prioridade

- **RF-05:** Desafios Gamificados
- **RF-06:** Ofertas de Serviços Externos

Baixa Prioridade

- **RF-07:** Meditação e Respiração
- **RF-08:** Sugestões de Pausas
- **RF-09:** Dicas de Ergonomia
- **RF-10:** Lembretes Personalizados

Requisitos de Qualidade

Disponibilidade

- **Meta:** 99,5% no horário comercial
- **Limite:** Máximo 2h de downtime/mês

Confiabilidade

- **RPO:** 4 horas (backup)
- **RTO:** 2 horas (recuperação)

Segurança

- **Foco:** Conformidade com a LGPD
- **Crítico:** Dados anônimos protegidos

Escalabilidade

- **Crescimento:** 500 → 5.000 usuários
- **Performance:** ≤ 3s para 95% requisições

Esses quatro atributos **moldaram/definiram** nossas decisões arquiteturais.

Principais Decisões Arquiteturais

Arquitetura de Microserviços

Divisão em 3 serviços: **User Service, Wellness Service** e **Scheduling Service**

API Gateway Centralizado

Nginx como ponto de entrada único para autenticação, rate limiting e roteamento

PostgreSQL + Redis

PostgreSQL com schemas isolados por serviço + Redis para cache de sessões

Containerização com Docker

Docker Compose para orquestração e deploy simplificado

As decisões arquiteturais estabelecem uma base sólida para um **sistema escalável e seguro!**

Por que essas decisões?

Atendimento aos ASRs

- **Disponibilidade:** Health checks + restart automático
- **Segurança:** Gateway centralizado + JWT + criptografia
- **Escalabilidade:** Microserviços independentes + cache
- **Confiabilidade:** Backup automatizado + transações

Trade-offs Considerados

- **Complexidade vs Flexibilidade:** Microserviços para evolução independente
- **Performance vs Segurança:** Gateway adiciona latência mas centraliza controle
- **Consistência vs Disponibilidade:** Schemas isolados para maior resiliência

Visões arquiteturais do sistema

Visão de módulos

Decomposição modular (Microserviços)

- **User service:** Autenticação, perfis
- **Wellnes service:** Feedbacks anônimos, recomendações, gamificação
- **Scheduling Service:** Agendamentos, integrações externas

Módulos de Infraestrutura

- **Gateway:** nginx para roteamento, autenticação, rate limiting.
- **Data Layer:** PostgreSQL (persistência isolada por serviço), Redis (cache).

Visão de Componentes e Conectores

Componentes Principais:

- **API Gateway:** Proxy/Load Balancer**.**
- **User/Wellness/Scheduling Service Components:** Microserviços com APIs REST, conexões com DB e Cache.

Fluxos de Dados:

- **Feedback Anônimo:** Cliente → Gateway → User Service (autenticação) → Wellness Service (processamento) → PostgreSQL (armazenamento criptografado).
- **Recomendações:** Cliente → Gateway → Wellness Service (cache/DB) → Retorno.

Visões arquiteturais do sistema

Visão de Alocação

Arquitetura de Deployment

- **Camada de Containers:** nginx-gateway, user-service, wellness-service, scheduling-service (múltiplas instâncias).
- **Camada de Dados:** postgresql-primary/replica, redis-cache.
- **Orquestração:** Docker Compose (restart automático, health checks).

Arquitetura de Rede

- **Rede Interna:** Docker Network isolada para comunicação entre serviços.
- **Rede Externa:** API Gateway exposto via HTTPS, Rate Limiting.

Visão de Informação

Modelo de Dados Conceitual:

- **Usuários:** Entidades User, Profile, Role, Session (dados sensíveis anonimizados).
- **Bem-estar:** Entidades Feedback (anônimo), Recommendation, Challenge (dados comportamentais).
- **Agendamentos:** Entidades Appointment, Specialist, Calendar (integrações externas).

Fluxos de Informação Críticos:

- **Anonimização de Feedbacks:** Hash irreversível antes do armazenamento.
- **Geração de Recomendações:** Coleta de dados comportamentais (não identificáveis), análise, personalização e entrega.

Táticas e Padrões Arquiteturais Aplicados

Atributos de Qualidade Garantidos

Disponibilidade

- **Detecção:** Health Checks automáticos (/health), Heartbeat Monitoring.
- **Recuperação:** Restart automático, Circuit Breaker Pattern (integrações externas).
- **Prevenção:** Load Balancing (nginx), Rate Limiting (DDoS, abuso).

Confiabilidade

- **Detecção:** Logging Estruturado, Validação de Dados.
- **Recuperação:** Estratégia de Backup, Gerenciamento de Transaçõe**.**
- **Prevenção:** Input Sanitization, Database Constraints**.**

Segurança

- **Resistência a Ataques:** JWT, RBAC, HTTPS obrigatório, AES-256, Hash (anonimização).
- **Detecção:** Audit Logging, Rate Limiting Inteligente.
- **Recuperação:** Invalidação/Logout forçado de sessões

Escalabilidade

- **Gerenciar Recursos:** Caching (Redis), Connection Pooling (HikariCP).
- **Gerenciar Demanda:** Horizontal Scaling (múltiplas instâncias), Data Partitioning (schemas isolados).

Táticas e Padrões Arquiteturais Aplicados

Padrões Arquiteturais Aplicados

Microservices Pattern

- Decomposição por domínio, comunicação REST, DB independente por serviço.

API Gateway Pattern

- Nginx como ponto de entrada centralizado (autenticação, logging, rate limiting).

Database per Service Pattern

- PostgreSQL com schemas isolados por serviço, sem compartilhamento direto.

Layered Architecture Pattern

- (Dentro de cada Microserviço) Controller, Service, Repository, Entity.

Repository Pattern

- Spring Data JPA para abstração e acesso a dados

Análise de Qualidade

Disponibilidade

- Meta: ≥ 99,5% de uptime
- Health checks e reinício automático de containers
- Gateway com rate limiting e serviços em containers independentes
- Banco com réplica para tolerância a falhas

Confiabilidade

- Meta: < 1% de falhas em transações críticas
- Validações, constraints e logging estruturado
- Transações com rollback e idempotência
- Backups com RPO e RTO definidos

Segurança

- 100% das requisições autenticadas e autorizadas
- JWT, RBAC e criptografia AES-256
- Anonimização de dados e audit logging
- Gateway centralizado e bloqueio por IP suspeito

Análise de Qualidade

Escalabilidade

- Escala horizontal por microserviços
- Redis para cache e Nginx com balanceamento de carga
- Particionamento lógico do banco por schema
- Suporte a picos de 10x a carga média

Testabilidade

- Microserviços independentes facilitam testes unitários
- REST APIs padronizadas permitem automação de testes
- Riscos: falta de testes end-to-end e ambientes inconsistentes
- Solução: ambientes de staging e CI com testes automatizados

Diagramas

Diagrama de Casos de Uso

O objetivo do diagrama de caso de uso é demonstrar as diferentes maneiras que o **usuário** pode **interagir com o sistema.**

Fornece uma visão geral do relacionamento entre casos de uso, atores e sistemas.

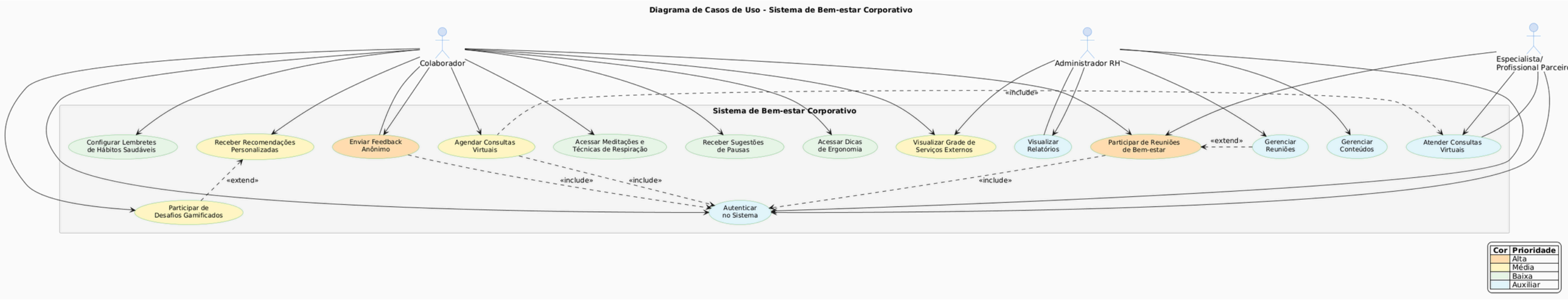

Diagrama de sequência

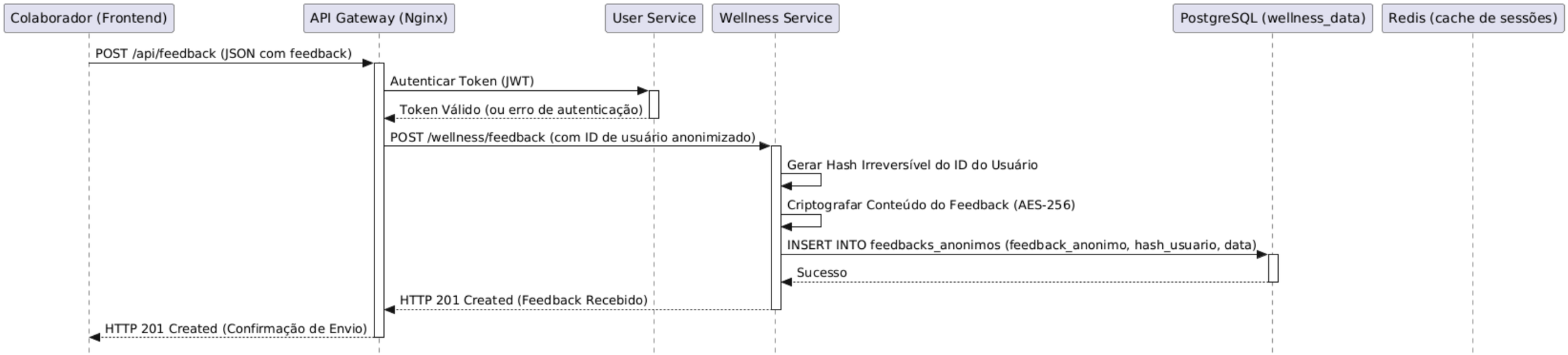

Diagrama de componentes

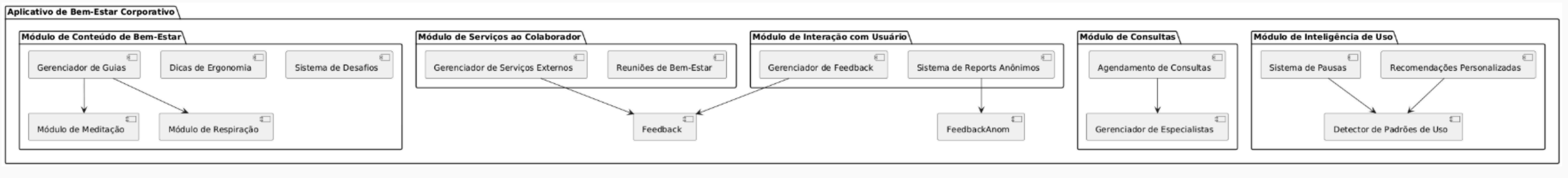

Diagrama Arquitetural