



SIDIS

Turma 3DA _ Grupo 6

1231479_José Figueiras

1221432_Luís Martins

1231690_Pedro Cunha

1230773_Pedro Sousa

11/1/2026

Índice

1.	Introdução e Visão Geral.....	3
2.	Contexto do Sistema e Stakeholders.....	4
3.	Arquitetura de Alto Nível.....	4
4.	Arquitetura e Padrões de Dados.....	5
5.	Comunicação e Transações Distribuídas.....	6
6.	Resiliência e Tolerância a Falhas.....	7
7.	Segurança Avançada (Zero-Trust).....	8
8.	Observabilidade.....	9
9.	Governação, SLAs e Evolução	10
10.	Detalhamento dos Microsserviços	10
10.1.	hap-auth	10
10.2.	hap-patients	11
10.3.	hap-physicians	11
10.4.	hap-appointmentrecords	11
11.	Infraestrutura e Deployment	11
12.	Testes e Qualidade.....	12
13.	Fluxos de Negócio Principais	13
14.	Conclusão	13
	Figura 1 - SystemContextDiagram	3
	Figura 2 - ContainersDiagram	5

1. Introdução e Visão Geral

Objetivo do projeto: Plataforma distribuída de saúde (HAP) com 4 microsserviços para gestão de pacientes, médicos, autenticação e registos clínicos.

Contexto: Sistema crítico para ambiente de saúde, com requisitos de segurança, disponibilidade e conformidade RGPD.

Arquitetura geral: Microsserviços independentes comunicando via HTTP/REST e AMQP, com isolamento de dados e padrões de resiliência.

Tecnologias base: Spring Boot, RabbitMQ, MongoDB, H2, Docker.

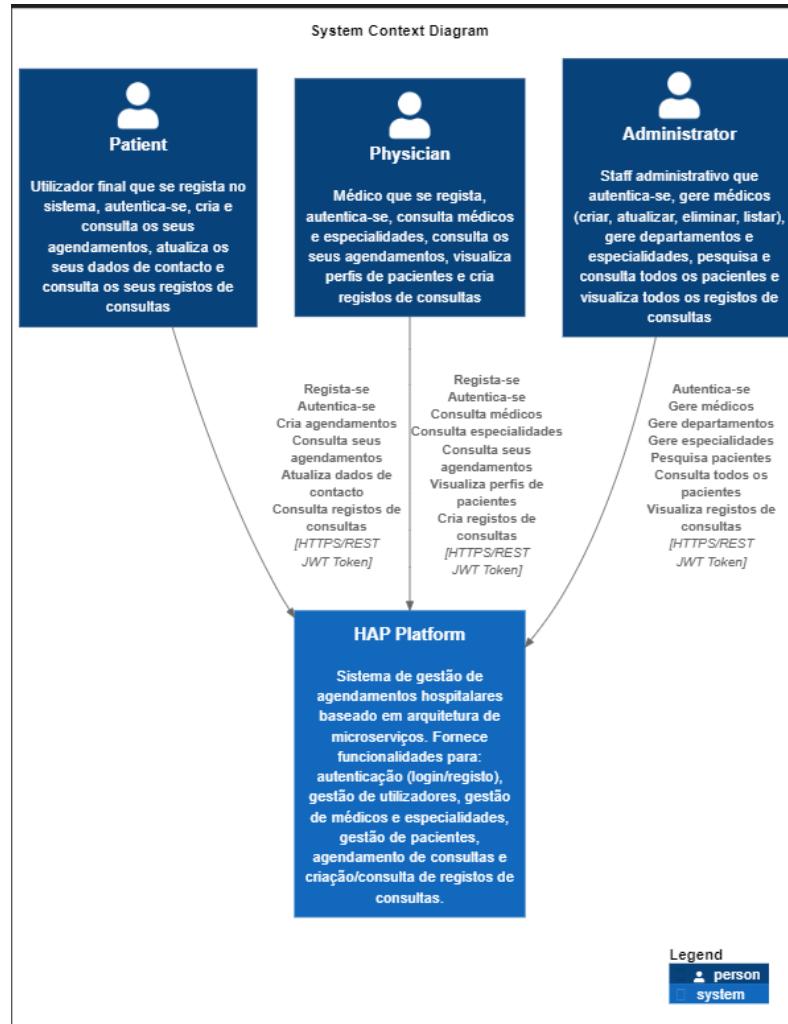


Figura 1 - SystemContextDiagram

2. Contexto do Sistema e Stakeholders

Stakeholders:

- Pacientes: registo, consulta de perfil, agendamento.
- Médicos: gestão de consultas, acesso a registos.
- Administradores: gestão completa, auditoria.
- Sistema externo: integração via APIs REST.

Requisitos funcionais: Registo de utilizadores, agendamento de consultas, gestão de registo clínico, autenticação/autorização.

Requisitos não funcionais: Alta disponibilidade (99.5%), segurança (mTLS, OAuth2), escalabilidade, observabilidade, conformidade RGPD.

Restrições: Dados sensíveis de saúde, isolamento por serviço, comunicação assíncrona para desacoplamento.

3. Arquitetura de Alto Nível

Visão de containers (C2):

- 4 microserviços: hap-auth (8084/8089), hap-patients (8082/8088), hap-physicians (8081/8087), hap-appointmentrecords (8083/8090).
- Infraestrutura: RabbitMQ (5672), MongoDB (27017), Prometheus (9090), Grafana (3000), Zipkin (9411).

Princípios arquiteturais:

- DDD: 4 Bounded Contexts (Auth, Patients, Physicians, AppointmentRecords).
- Database-per-Service: isolamento total de dados.
- API-Led Architecture: System/Process/Experience APIs.

Comunicação: HTTP/REST (síncrono) e AMQP (assíncrono).

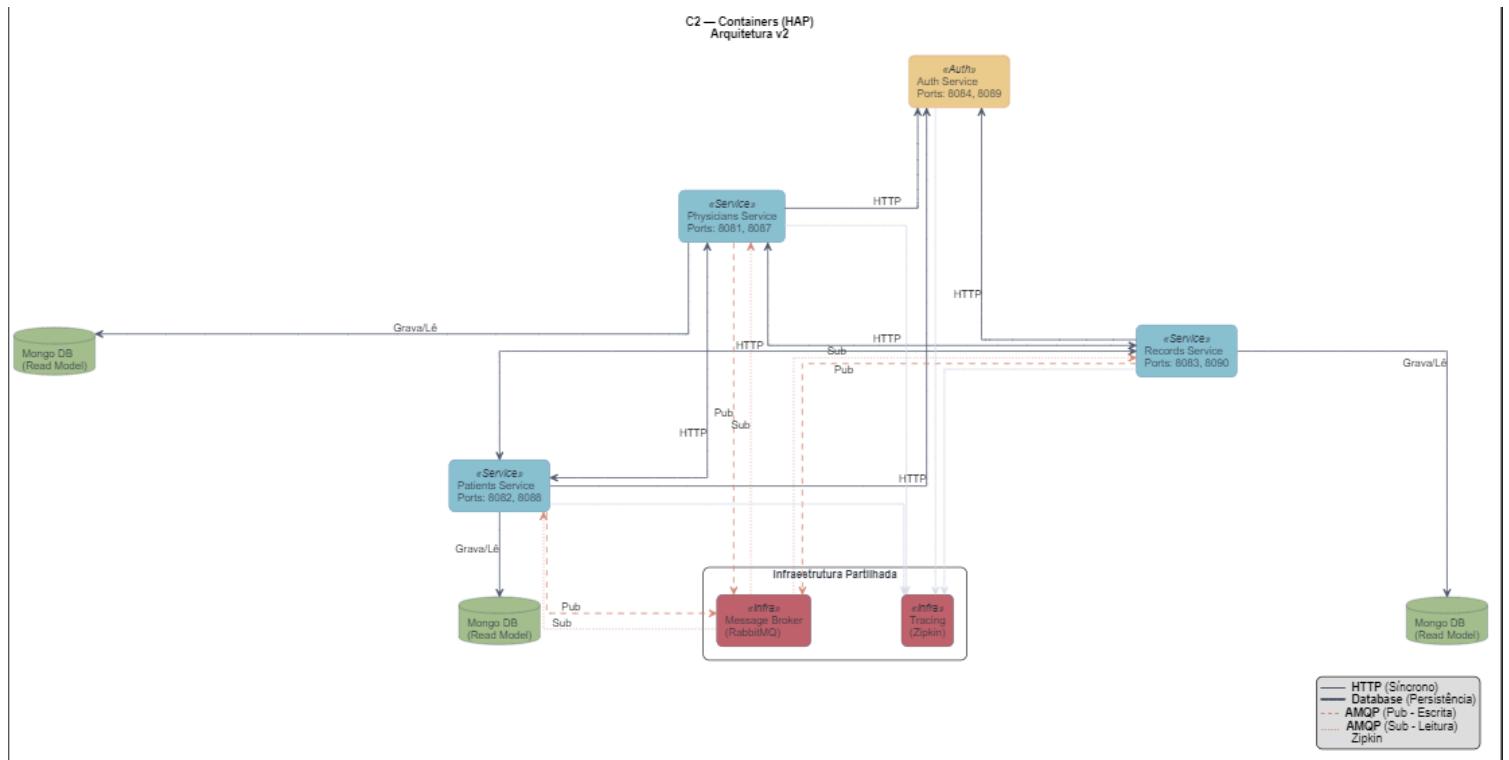


Figura 2 - ContainersDiagram

4. Arquitetura e Padrões de Dados

Database-per-Service:

- Cada serviço tem base de dados própria (H2 em dev, PostgreSQL/SQL Server em produção).
- Sem acesso direto entre bases de dados.
- Comunicação apenas via HTTP/REST ou AMQP.

CQRS (Command Query Responsibility Segregation):

- Write Model: H2/JPA para comandos e transações.
- Read Model: MongoDB para queries otimizadas.
- Exemplos: POST /appointments (Command) vs GET /appointments (Query).
- Implementação: CommandService e QueryService separados.

Polyglot Persistence:

- Relacional (H2): dados transacionais, integridade referencial.
- NoSQL (MongoDB): projeções de leitura, histórico de consultas.
- Justificativa: adequar tecnologia ao modelo de dados.

Event Sourcing:

- Event Store (H2) no hap-physicians para consultas.
- Eventos imutáveis: AppointmentCreated, AppointmentCanceled, etc.
- Audit trail completo para RGPD.
- Replay de eventos para reconstrução de estado.

Modelos de dados principais:

- Auth: User, Role.
- Patients: Patient, PatientProfile.
- Physicians: Physician, Appointment.
- AppointmentRecords: AppointmentRecord, ClinicalNote.

5. Comunicação e Transações Distribuídas

Comunicação síncrona (HTTP/REST):

- Cliente -> Serviço: feedback imediato, validações em tempo real.
- Serviço -> Auth: validação de credenciais obrigatória.
- Propagação de headers: Authorization, X-User-Id, X-User-Role, X-Correlation-Id.
- Exemplos: POST /api/v2/patients/register (chama hap-auth síncrono).

Comunicação assíncrona (AMQP/RabbitMQ):

- Exchange: hap-exchange (topic).
- Eventos principais: appointment.created, patient, physician.registered, etc.
- Desacoplamento temporal: fire-and-forget.
- Suporte ao CQRS: sincronização Write -> Read Model.

Padrão Saga (Orquestração):

- Saga baseada em Coreografia: eventos publicados, serviços reagem.
- Exemplo cancelamento: hap-physicians cancela -> publica AppointmentCanceled -> hap-appointmentrecords atualiza estado.
- Compensações: ações de rollback em caso de falha.
- Timeout: 30s para operações de Saga completas.

Peer Forwarding:

- Mecanismo: instância tenta localmente, depois tenta peers.
- Alta disponibilidade: distribuição de dados entre instâncias.
- Exemplo: GET /patients/{id} tenta instance1, depois instance2.
- Configuração: lista estática de peers por perfil.

6. Resiliência e Tolerância a Falhas

Circuit Breaker (Resilience4j):

- **Proteção:** chamadas a serviços externos e publicações AMQP.
- **Configuração:** sliding window 10, failure rate threshold 50%, wait duration 5-10s.
- **Estados:** CLOSED -> OPEN -> HALF_OPEN -> CLOSED.
- **Integração:** exposto como health indicator, métricas Prometheus.
- **Aplicação:** hap-auth, hap-appointmentrecords, publicações AMQP.

Exponential Backoff Retry:

- Configuração: max 3 tentativas, wait duration 500ms, exponential backoff habilitado.
- Casos: operações de Saga, publicações AMQP.
- Benefício: lida com falhas temporárias de rede.

Timeout (TimeLimiter):

- HTTP: 5-15s para chamadas a serviços externos.
- Saga: 30s para transações distribuídas completas.
- Benefício: evita bloqueios indefinidos.

Bulkhead:

- Isolamento: pool de threads separado para compensações.
- Aplicação: operações de compensação de Saga isoladas.
- Benefício: problemas em compensações não afetam operações normais.

Integração com Saga:

- Publicações de eventos com Circuit Breaker e Retry.
- Timeout em operações de Saga.
- Métricas: saga.step.duration, contadores de compensações.
- Métricas de resiliência: Expostas via /actuator/metrics, integração Prometheus/Grafana.

7. Segurança Avançada (Zero-Trust)

OAuth2 e JWT:

- Padrão: OAuth2 Resource Server com Spring Security.
- hap-auth como Authorization Server: emite tokens JWT.
- Outros serviços como Resource Servers: validam tokens.
- Claims: userId, email, roles (ADMIN, PHYSICIAN, PATIENT).
- Stateless: validação sem comunicação com Authorization Server.
- Fluxo: POST /api/public/login -> recebe JWT -> Authorization: Bearer <token>.

mTLS (Mutual TLS):

- Encriptação mútua: todas as comunicações service-to-service.
- Certificados: keystore (próprio) e truststore (serviços confiáveis).
- Endpoints internos: /internal/** com server.ssl.client-auth=need.
- Endpoints públicos: Swagger/H2 com server.ssl.client-auth=want.
- Geração: script generate_certs.sh para certificados X.509.
- Benefício: proteção contra eavesdropping e man-in-the-middle.

RBAC (Role-Based Access Control):

- @PreAuthorize com roles específicas.
- Exemplos: POST /appointments (ADMIN/PHYSICIAN/PATIENT), GET /appointments (ADMIN/PHYSICIAN), PUT /appointments/{id}/cancel (ADMIN).
- Proteção de dados sensíveis: apenas roles autorizadas.

Conformidade RGPD:

- Audit trails: Event Store imutável com metadados (timestamp, user ID, correlation ID).
- Endpoints de auditoria: GET /appointments/{id}/audit-trail (ADMIN/PHYSICIAN).
- Encriptação em trânsito: mTLS.
- Event Sourcing: histórico completo para direito de acesso e portabilidade.

8. Observabilidade

Logging estruturado:

- Correlation IDs: X-Correlation-Id propagado via HTTP e AMQP.
- Trace IDs: Micrometer Tracing (Brave) injeta TraceID e SpanID via MDC.
- Formato: timestamp, nível, thread, trace ID, correlation ID, mensagem.
- Integração: pronto para ELK Stack (Elasticsearch, Logstash, Kibana).
- Benefício: rastrear operação completa através de múltiplos serviços.

Métricas (Prometheus + Grafana):

- Métricas: saga.step.duration, saga.compensation.count, amqp.messages.published, amqp.messages.consumed, circuit.breaker.state.
- Expostas via: /actuator/prometheus.
- Dashboards Grafana: saúde do sistema, latências, taxas de erro.
- Alertas: configuráveis no Grafana (ex: "alerta se >10% mensagens AMQP falharem").

Distributed Tracing (Zipkin):

- Propagação automática: Trace IDs via headers HTTP e AMQP.
- Visualização: percurso completo de transação, latências por componente.
- Integração: Micrometer Tracing Bridge com Brave.
- Endpoint: http://localhost:9411.
- Benefício: depuração de problemas que envolvem múltiplos serviços.

Health Checks:

- Liveness: /actuator/health/liveness (processo vivo).
- Readiness: /actuator/health/readiness (pronto para tráfego).
- Health indicators customizados: Circuit Breakers expostos como health.
- Integração: Docker/Kubernetes usa para reiniciar/remover instâncias.

9. Governação, SLAs e Evolução

SLAs (Service Level Agreements):

- Disponibilidade: 99.5% (meta).
- Latência: <500ms para queries, <2s para transações Saga.
- Monitorização: Prometheus + Grafana para métricas em tempo real.
- Health checks: integração com orquestradores para auto-recuperação.

Testes de Contrato (Pact):

- Implementação: consumer tests em hap-appointmentrecords, provider tests em hap-patients.
- Fluxo: consumer gera contrato → provider valida contra contrato.
- Garantia: evolução de APIs não quebra serviços dependentes.
- Localização: target/pacts/ (contratos gerados).

Estratégia de Depreciação:

- Versionamento por URL: /api/v1/... vs /api/v2/...
- Exemplo: POST /api/v2/patients/register (versão melhorada).
- Ciclo de vida: v1 mantido durante período de transição, depois deprecado.
- Documentação: Swagger/OpenAPI documenta versões disponíveis.

Blue-Green Deployment:

- Implementação: hap-patients-blue (8082) e hap-patients-green (8088).
- Estratégia: alternar tráfego entre versões sem downtime.
- Configuração: Docker Compose com ambas as instâncias.

10. Detalhamento dos Microsserviços

10.1. hap-auth

- **Responsabilidade:** Gestão de identidades, credenciais, roles, emissão de tokens JWT.
- **Endpoints:** POST /api/public/login, POST /api/public/register.
- **Persistência:** H2 (Users, Roles).
- **Comunicação:** Recebe chamadas síncronas de outros serviços para validação.

10.2. hap-patients

- **Responsabilidade:** Gestão do ciclo de vida e perfil de pacientes (dados sensíveis/PII).
- **Endpoints:** GET /patients/{id}, POST /api/v2/patients/register, PATCH /patients/me.
- **CQRS:** Write (H2/JPA), Read (MongoDB).
- **Comunicação:** HTTP (síncrono), AMQP (publica PatientRegisteredEvent).
- **Peer Forwarding:** Implementado para alta disponibilidade.

10.3. hap-physicians

- **Responsabilidade:** Gestão de médicos, especialidades, agendamento de consultas futuras.
- **Endpoints:** POST /physicians/register, POST /appointments, GET /appointments, PUT /appointments/{id}/cancel.
- **CQRS + Event Sourcing:** Write (H2), Read (MongoDB), Event Store (H2).
- **Comunicação:** HTTP (valida paciente), AMQP (publica eventos de consultas).
- **Padrões:** Saga, Circuit Breaker, Retry, Bulkhead.

10.4. hap-appointmentrecords

- **Responsabilidade:** Arquivo histórico e clínico de consultas realizadas.
- **Endpoints:** POST /api/appointment-records/{id}/record, GET /api/appointment-records/{id}.
- **Persistência:** MongoDB (registros clínicos).
- **Comunicação:** HTTP (obtém dados de pacientes/médicos), AMQP (consome eventos de cancelamento).

11. Infraestrutura e Deployment

Docker Compose:

- Serviços: 4 microserviços + RabbitMQ + MongoDB + Prometheus + Grafana + Zipkin.
- Networking: hap-net (bridge network).
- Volumes: mongo_data para persistência MongoDB.
- Health checks: configuração por serviço com start_period.

Perfis Spring Boot:

- instance1: portas 8081-8084.
- instance2: portas 8087-8090.
- docker: configuração para ambiente Docker.

Portas e endpoints:

- Tabela completa: serviço -> porta instance1 -> porta instance2
- Swagger UI: <http://localhost:{porta}/swagger-ui.html>.

Configuração SSL/mTLS:

- Certificados: gerados via generate_certs.sh.
- Keystore/Truststore: .p12 files em cada serviço.
- HTTPS: todas as comunicações service-to-service.

12. Testes e Qualidade

Testes unitários:

- Cobertura: Controllers, Services, Repositories, Event Handlers.
- Ferramentas: JUnit 5, Mockito, AssertJ.
- Documentação: TEST_DOCUMENTATION.md em cada módulo.

Testes de integração:

- Comunicação entre serviços: RestTemplate mockado.
- Eventos AMQP: testes de publicação/consumo.
- Persistência: H2 in-memory para testes.

Testes de contrato (Pact):

- Consumer: PactConsumerTest em hap-appointmentrecords.
- Provider: PactProviderTest em hap-patients.
- Contratos: gerados em target/pacts/.

Testes de resiliência:

- Circuit Breaker: testes de abertura/fechamento.
- Retry: testes de tentativas com backoff.
- Timeout: testes de cancelamento após timeout.

13. Fluxos de Negócio Principais

Registo de paciente:

1. Cliente chama POST /api/v2/patients/register.
2. hap-patients valida dados e cria perfil (transação local).
3. Chama hap-auth síncrono para criar credenciais.
4. Publica PatientRegisteredEvent via AMQP.
5. Atualiza Read Model (MongoDB) via event handler.

Agendamento de consulta:

1. Cliente chama POST /appointments.
2. hap-physicians valida médico/paciente (HTTP síncrono).
3. Cria consulta (Write Model H2).
4. Salva evento no Event Store.
5. Publica AppointmentCreatedEvent (com Circuit Breaker + Retry).

Cancelamento de consulta (Saga):

1. Cliente chama PUT /appointments/{id}/cancel.
2. hap-physicians cancela localmente.
3. Publica AppointmentCanceledEvent.
4. hap-appointmentrecords consome evento e atualiza estado.
5. Compensação: se falhar, ação de rollback isolada (Bulkhead).

14. Conclusão

A Plataforma HAP implementa padrões de microsserviços de nível industrial. Aplicando padrões como CQRS, Event Sourcing, Saga, Circuit Breaker, mTLS, OAuth2. Com isto, alcançamos os seguintes benefícios:

- Alta disponibilidade (peer forwarding, múltiplas instâncias).
- Segurança (Zero-Trust com mTLS + OAuth2).
- Observabilidade completa (logging, métricas, tracing).
- Resiliência (Circuit Breaker, Retry, Timeout, Bulkhead).
- Conformidade RGPD (audit trails, Event Sourcing).

Com isto, a Plataforma HAP, encontra-se preparada para produção, uma vez que é um sistema robusto, monitorizável e seguro, em que retiramos como lições a importância de observabilidade, resiliência desde o início e segurança em camadas.