1. **Resumo Executivo**

O sistema desenvolvido é um gerenciador de recursos para o ambiente hospitalar. Se trata de um MVP de um backend que pode ser implementado em uma rede interna, ou de um sistema escalável com arquitetura em nuvem.

A ideia é fornecer uma ferramenta que possa ser usada tanto para gerir os recursos disponíveis, verificando a disponibilidade, cadastrando novos recursos ou gerindo o consumo.

Espera-se, com a sua implementação, uma racionalização do consumo de recursos disponíveis, e uma agilidade nas atividades dos profissionais no cotidiano, servindo para reduzir o risco de desabastecimento, e para orientar para quais os recursos possíveis a cada instante.

O sistema visa facilitar o controle e rastreabilidade de processos críticos em ambientes hospitalares, promovendo agilidade, segurança e transparência nas operações de cadastro, atualização, busca e movimentação de recursos.

1. **Problema Identificado**

A escassez de recursos, seja de colaboradores, seja de estrutura física como leitos, ou por desabastecimento de medicamentos, afeta cotidianamente a rotina de quem depende do sistema público de saúde.

Gerir o consumo desses recursos, e prover uma ferramenta capaz de otimizar o uso e reduzir o risco de desabastecimento por falta de gestão é o principal objetivo desse sistema.

1. **Descrição da Solução**

Este projeto é uma aplicação Java baseada em Spring Boot, desenvolvida para apoiar operações de saúde, incluindo o gerenciamento de pacientes, colaboradores, coletas, descartes e entregas de equipamentos e insumos hospitalares.

Dentre as funcionalidades implementadas, podemos citar:

- Cadastro e atualização de pacientes e colaboradores;

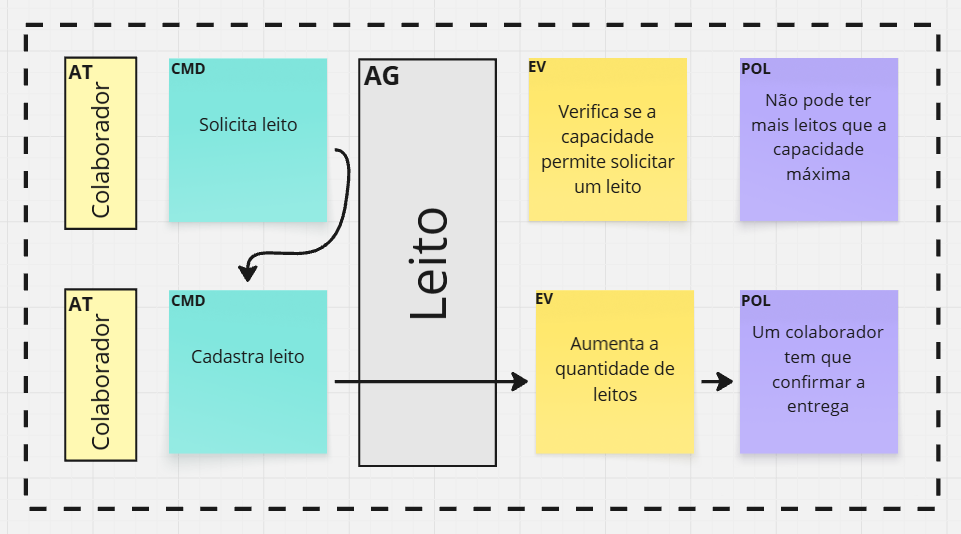
- Gestão de coletas, descartes e entregas de equipamentos e insumos;

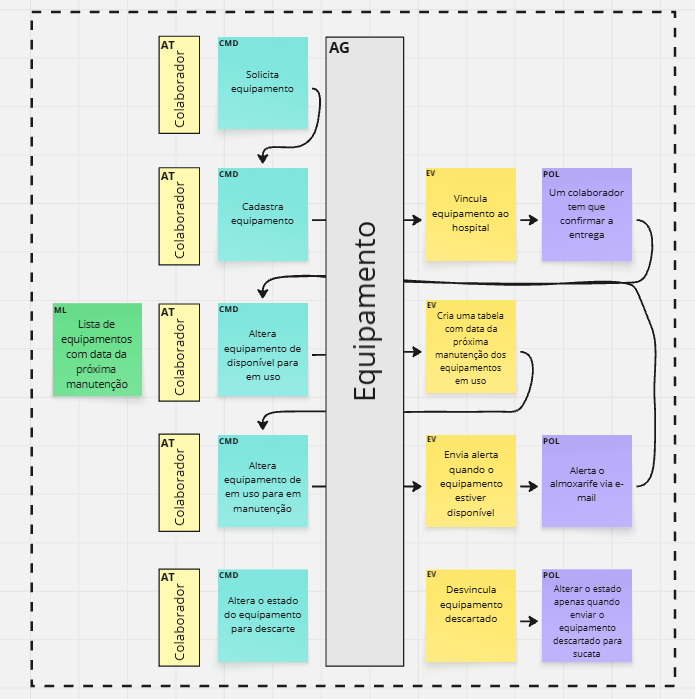
- Busca e listagem de registros;

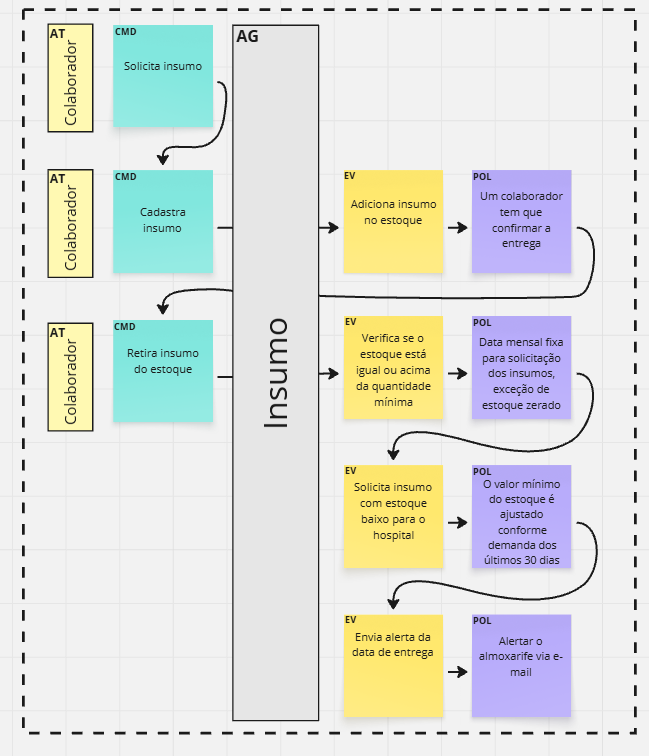
1. **Processo de desenvolvimento**

Foi adotado o Domain Drive Desing (DDD) para facilitar a delimitação dos contextos da aplicação desde o momento da Ideação. Promovendo o desenvolvimento, desde o começo, com reduzido acoplamento entre classes.

No momento de ideação, buscou-se separar em 3 contextos:

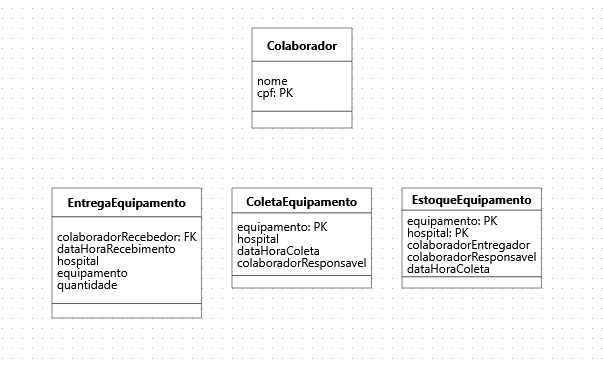






As imagens acima ilustram o resulto da event storming promovido para a definição dos contextos. Mais detalhes do processo estão no Miro, com link disponível no capítulo 6.

O exemplo da modelagem final das classes do contexto de equipamento pode ser observado no diagrama do modelo a seguir:



1. **Detalhes técnicos**

O projeto adota uma arquitetura baseada em Clean Arch, com a separação em camadas, segregando as responsabilidade de negócios, de persistência e de disponibilização do endpoint.

Nas classes *controller*, dentro da camada *infraestructure*, estão implementadas as validações dos dados de entrada. Evitando que dados incompletos sem campos obrigatórios sejam processados pela aplicação. As classes *controller* tem a responsabilidade de disponibilizar os endpoints da aplicação. Ainda na camada *infraestructure*, encontram-se as classes *dto* (data transfer objetcts) que se destinam a transferência dos dados ao longo da aplicação, evitando a propagação das *model* das *entity* que tem relação estrita com as entidades das tabelas, evitando modificações indesejadas nas tabelas.

Foram implementados tratamentos de exceções, com *exceptions* personalizadas implementadas em cada domínio na camada *entity*. Ainda na camada *entity* tem-se os models, que representam as entidades da tabela, e a interface gateway, que será implementada na camada *infraestructure*.

Na camada *usecase* estão presentes as regras de negócio da aplicação.

Os endpoints, exemplos de uso e documentação de uso estão no vídeo disponível no link do capítulo 6.

Para a integração com o banco de dados foi utilizado o Spring Data JPA, o que tornou transparente maiores configurações de conexão com o banco.

Foram realizados testes unitários automatizados, bem como o BDD e testes de performance. Para os testes unitários, foram adotados o Mockit e o Junit. A cobertura pode ser observada na imagem a seguir:



Já os testes BDD (Behavior-Driven Development), que são testes que permitem alinhar o comportamento do sistema com o que é esperado, e os testes de Performance estão com os resultados disponíveis no repositório do Git, com link disponível no capítulo 6.

1. **Links**

Repositório Git

<https://github.com/TayMike/hackaton>

Repositório Vídeo

<https://drive.google.com/drive/folders/11xgd4SweIuJgn3ykt9f9uUc8BbuHGzTu?usp=sharing>

Miro

<https://miro.com/app/board/uXjVJezNcEM=/>

1. **Próximos Passos**

Integrar à aplicação em um sistema que automatize a criação de um relatório financeiro que permita uma análise a nível monetário dos recursos. Além disso, automatizar o inventário por cada tipo de equipamento e de insumo.