RELATÓRIO LABDSOFT

P3

**André Sacramento** - 1161107

**Luís Vigário** – 1180990

**Maria Pereira** – 1181130

**Carlos Ferreira** – 1181486

**Professor** – António Rocha AJO

Instituto Superior de Engenharia do Porto

Mestrado em Engenharia Informática – Engenharia de Software

Conteúdo

[Introdução 5](#_Toc124114370)

[Âmbito do projeto 5](#_Toc124114371)

[Organização 5](#_Toc124114372)

[Processo de Análise e Organização dos Requisitos 6](#_Toc124114373)

[Regras do Processo do Desenvolvimento 6](#_Toc124114374)

[Feature Branch 6](#_Toc124114375)

[Pipeline 7](#_Toc124114376)

[Regras do projeto 8](#_Toc124114377)

[Arquitetura 9](#_Toc124114378)

[Arquitetura Onion 9](#_Toc124114379)

[Domain Entities 9](#_Toc124114380)

[Repository Layer 9](#_Toc124114381)

[Service Layer 9](#_Toc124114382)

[UI Layer 10](#_Toc124114383)

[Abordagem Arquitetural 10](#_Toc124114384)

[Padrão MVC 11](#_Toc124114385)

[Abordagem Arquitetural 11](#_Toc124114386)

[Tecnologias para Implementação 12](#_Toc124114387)

[Landbot 12](#_Toc124114388)

[Google Cloud Vision 12](#_Toc124114389)

[Análise de Imagem 13](#_Toc124114390)

[Implementação 14](#_Toc124114391)

[Modelo Relacional da Base de Dados 14](#_Toc124114392)

[API e Simulador Health Tracker 14](#_Toc124114393)

[Requisitos 16](#_Toc124114394)

[Conclusões do trabalho 18](#_Toc124114395)

Conteúdo

[Figura 1 - User Stories no Jira 6](#_Toc124114396)

[Figura 2 - Excerto do ficheiro de configuração das pipelines referente ao step da LabdsoftCore 7](#_Toc124114397)

[Figura 3 - Excerto do ficheiro de configuração das pipelines referente ao step da API Equipment 7](#_Toc124114398)

[Figura 4 - Excerto do ficheiro de configuração das pipelines referente ao step da API Health Tracker 7](#_Toc124114399)

[Figura 5 - Excerto do ficheiro de configuração das pipelines referente ao step da Labdsoft UI 8](#_Toc124114400)

[Figura 6 - Excerto do ficheiro de configuração das pipelines referente ao step da Security Scan 8](#_Toc124114401)

[Figura 7 - Designação dos Commits 8](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114402)

[Figura 8 - Designação dos Commits 8](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114403)

[Figura 9 - Arquitetura Onion 9](#_Toc124114404)

[Figura 10 - Vista Lógica Nível 3 10](#_Toc124114405)

[Figura 11 - Vista Lógica Nível 3 Frontend 11](#_Toc124114406)

[Figura 12 - Depêndencia Spring Cloud GCP 13](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114407)

[Figura 13 - Depêndencia Spring Cloud GCP 13](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114408)

[Figura 14 - Injeção de dependência CloudVisionTemplate 13](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114409)

[Figura 15 - Estrutura de dados recebida na API Health Tracker 14](#_Toc124114410)

[Figura 16 - Mensagem e Assunto do email alerta 15](#_Toc124114411)

[Figura 17 - Exemplo email alerta recebido 15](#_Toc124114412)

[Figura 18 - Proposta de funcionalidades para a 2º Iteração 16](#_Toc124114413)

[Figura 19 - Diagrama genérico pedido create 16](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114414)

[Figura 20 - Diagrama genérico pedido read 17](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114415)

[Figura 21 - Diagrama genérico pedido update 17](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114416)

[Figura 22 - Distribuição de Tarefas 19](#_Toc124114417)

[Figura 23 - Modelo relacional 1 20](#_Toc124114418)

[Figura 24 - Modelo Relacional 2 21](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114419)

[Figura 25 - Modelo Relacional 3 22](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114420)

[Figura 26 - Landbot Diagrama 1 23](#_Toc124114421)

[Figura 27 - Landbot Diagrama 2 24](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114422)

[Figura 28 - Landbot Diagrama 3 25](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114423)

[Figura 29 - Landbot SSD 26](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114424)

[Figura 30 - Diagrama de Sequência do Simulador com integração à API Health Tracker 27](https://myisepipp-my.sharepoint.com/personal/1180990_isep_ipp_pt/Documents/LABDSOFT_Relatorio_P3.docx#_Toc124114425)

# Introdução

O presente relatório foi desenvolvido no âmbito da unidade curricular de Laboratório Desenvolvimento Software (LABDSOFT) do Mestrado em Engenharia Informática (MEI) do Instituto Superior de Engenharia do Porto (ISEP).

No presente documento vão ser descritos o produto e as suas funcionalidades com suporte a diagramas desenvolvidos pela equipa. Para além disto serão descritas as decisões tomadas e as suas justificações, artefactos relevantes para o projeto, as medidas de qualidade usadas.

# Âmbito do projeto

O Centro de Inovação e do Transplante de Órgãos (CITO) é uma clínica caracterizada por gerir a atribuição de órgãos humanos entre possíveis recetores e possíveis dadores através de critérios complexos que envolvem por um lado análises genéticas (DNA), análises víricas, análises de imunidade, análises serológicas e, por outro lado, critérios de prioridade para os possíveis recetores, nomeadamente: a sua urgência, a idade, o tempo de espera, etc.

A CITO pretende uma aplicação que automatize e simplifique os processos associados à sua atividade.

# Organização

O projeto foi feito em duas grandes etapas. A primeira esta relacionada com o planeamento do projeto. Nesta fase foi necessário avaliar os requisitos e planear os sprints. Numa segunda parta procedeu-se ao desenvolvimento propriamente dito. Para isso, usou-se todo o planeamento tanto de análise de requisitos como de design feito na primeira iteração.

# Processo de Análise e Organização dos Requisitos

Após a análise dos requisitos surgiram algumas dúvidas, às quais foram esclarecidas com o Product Owner. Após este processo foram distribuídas responsabilidades referente a cada requisito e foram anotados num documento Word (anexo 1).

Depois foram criadas User Stories mais pequenas a partir das tarefas distribuídas e o grupo escolheu usar o plug-in do Jira no Bitbucket para gerir o estado de cada User Story.

Uma imagem com texto

Descrição gerada automaticamente

Figura 1 - User Stories no Jira

# Regras do Processo do Desenvolvimento

## Feature Branch

A principal branch para o desenvolvimento da aplicação é precisamente a dev, onde o código criado em cada feature é merged, e é sujeita a pipeline runs a cada commit, testando a sua capacidade de executar corretamente a app, a interface gráfica, a componente dedicada à integração da API de equipamentos e o scan de segurança.

Dado que esta se tratou da primeira iteração de desenvolvimento, a equipa negligenciou a utilização de um master branch, comummente observada em projetos desta tipologia e estratégia, aspeto que será objeto de discussão entre os membros para uma potencial inclusão nas próximas iterações.

## Pipeline

A implementação de pipelines no projeto passa pelo trigger de uma Bitbucket Pipeline quando alterações são committed na dev, onde a correta compilação do código dos componentes de aplicação, interface gráfica, API de equipamentos e API de Health Tracker é verificada, assim como o scan de segurança, sendo a execução bem-sucedida se nenhum componente falhar.

Step da LabdsoftCore

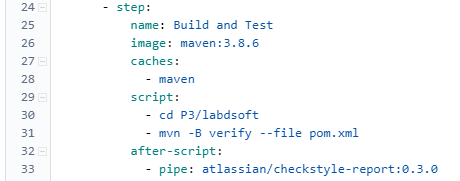


Figura 2 - Excerto do ficheiro de configuração das pipelines referente ao step da LabdsoftCore

Step da API Equipment

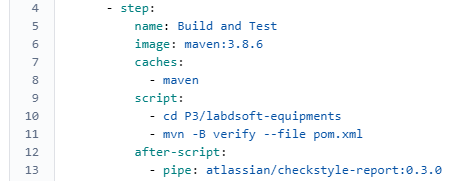


Figura 3 - Excerto do ficheiro de configuração das pipelines referente ao step da API Equipment

Step da API Health Tracker

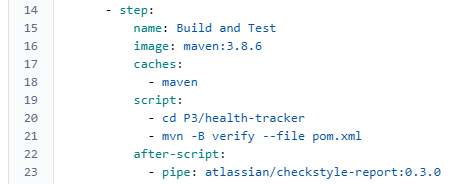


Figura 4 - Excerto do ficheiro de configuração das pipelines referente ao step da API Health Tracker

Step da Labdsoft UI

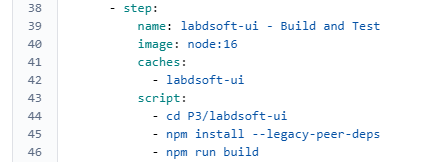


Figura 5 - Excerto do ficheiro de configuração das pipelines referente ao step da Labdsoft UI

Security Scan

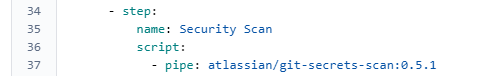


Figura 6 - Excerto do ficheiro de configuração das pipelines referente ao step da Security Scan

A equipa discutiu também a possível implementação de triggered scheduling e mesmo verificações de critérios adicionais para um melhor entendimento do comportamento e desempenho atual, mas dados constrangimentos de tempo optou por não realizar.

## Regras do projeto

De forma a poder garantir algum tipo de consistência dentro do processo de desenvolvimento e passagem das alterações para o branch principal foram estipulados um conjunto de nomenclaturas e regras para serem cumpridas por todos os elementos da equipa:

1. Através do Issue do Jira criar o branch para o User Story.

2. Todos os commits dados são dados para o branch criado.

3. Cada commit terão o seguinte formato: [Jira Issue] [Implementação/Documentação] DESCRIÇÃO DO PROBLEMA

Uma imagem com texto

Descrição gerada automaticamente4. Quando a User Story estiver completa fazer o Pull Request para o master e esperar pelo review dos colegas de equipa.

Figura - Designação dos Commits

Figura 8 - Designação dos Commits

# Arquitetura

A solução implementada possui dois módulos distintos: um que grava a informação, obtém dados e realiza ações lógicas e de negócio, normalmente designado como Labdsoft Core, e outro que mostra a informação ao utilizador final e permite a interação do mesmo com os vários componentes, designado como LabdsoftUI.

## Arquitetura Onion

Após uma análise do problema, o grupo decidiu utilizar a Arquitetura Onion como solução. A Arquitetura Onion aborda os desafios das arquiteturas de 3 camadas ou mais, e fornece uma solução para problemas inerentes. As camadas da Arquitetura Onion interagem entre si através da utilização das Interfaces. A Arquitetura Onion é composta por múltiplas camadas concêntricas que interagem entre si em direção ao núcleo que representa o domínio. A arquitetura não depende da data layer, mas sim dos modelos de domínio. No centro da Arquitetura Onion está o Domain Layer, que representa os objetos de negócio. À volta da camada do Domain Layer encontram-se outras camadas, com outros comportamentos. A imagem seguinte é um exemplo da Arquitetura Onion:

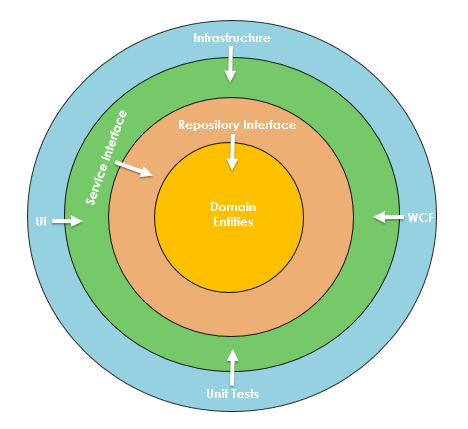


Figura - Arquitetura Onion

### Domain Entities

No centro encontra-se o Domain Layer e é responsável por armazenar todos os objetos e as suas respetivas regras de negócio.

### Repository Layer

Esta camada cria uma abstração entre as Domain Entities e as regras de negócio de uma aplicação. Nesta camada, acrescenta-se tipicamente interfaces que proporcionam um comportamento de persistência de objetos, envolvendo uma base de dados. É criado um repositório genérico, e adiciona-se querys para obter os dados, mapear os dados, e persistir alterações na base de dados.

### Service Layer

A Service Layer possui interfaces com operações comuns, tais como Add, Save, Edit, e Delete. Além disso, esta camada é utilizada para comunicar entre a UI Layer e a Repository Layer.

### UI Layer

É a camada mais externa, e é responsável pelas preocupações periféricas como a IU e os testes. Para uma aplicação Web, representa o Web API ou Unit Test.

### Abordagem Arquitetural

Para esta solução foi utilizada a arquitetura Onion, e a imagem seguinte mostra as várias camadas do projeto desenvolvido:

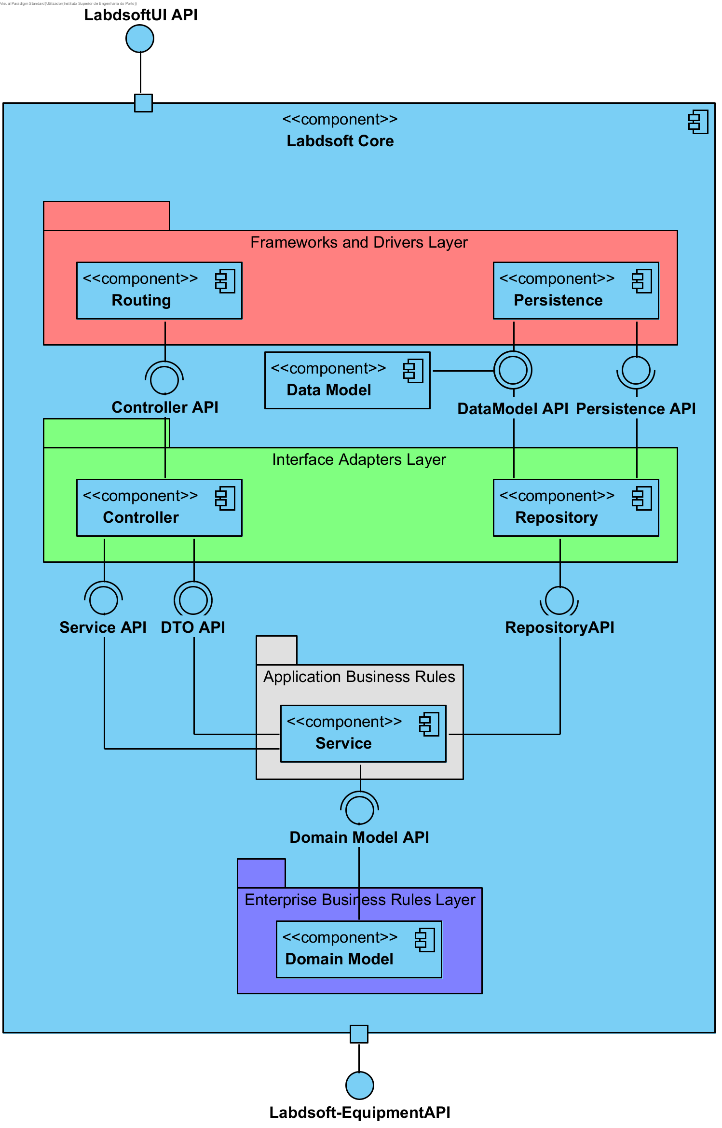


Figura - Vista Lógica Nível 3

O componente Domain Model possuí as Entities. Existem Service Interfaces e Repositoy Interfaces que ligam as diferentes camadas. Há também o componente de base de dados que o grupo decidiu usar Postgre, uma vez que é uma base de dados relacional. O padrão DTO também foi utilizado para simplificar a entrada de dados no sistema. O componente controlador é responsável por receber pedidos CRUD (Create, Read, Update, Delete) do cliente.

## Padrão MVC

O MVC é um padrão de arquitetura de software. O MVC sugere uma forma de dividir as responsabilidades, principalmente dentro de software web. O princípio do MVC é a divisão da aplicação em três camadas: a camada de interação com o utilizador (view), a camada de manipulação dos dados (model) e a camada de controlo (controller). Com o MVC, é possível separar o código relativo à interface do utilizador das regras de negócio, o que sem dúvida traz muitas vantagens.

Model: A responsabilidade dos models é representar o negócio da aplicação. Também é responsável pelo acesso e manipulação dos dados da aplicação.

View: A view é responsável pela interface que será apresentada ao utilizador e mostra as informações do model para o utilizador.

Controller: É a camada de controlo, responsável por ligar o model e a view.

### Abordagem Arquitetural

Para esta solução foi utilizado padrão MVC, e a imagem seguinte mostra como foi desenvolvido no projeto:

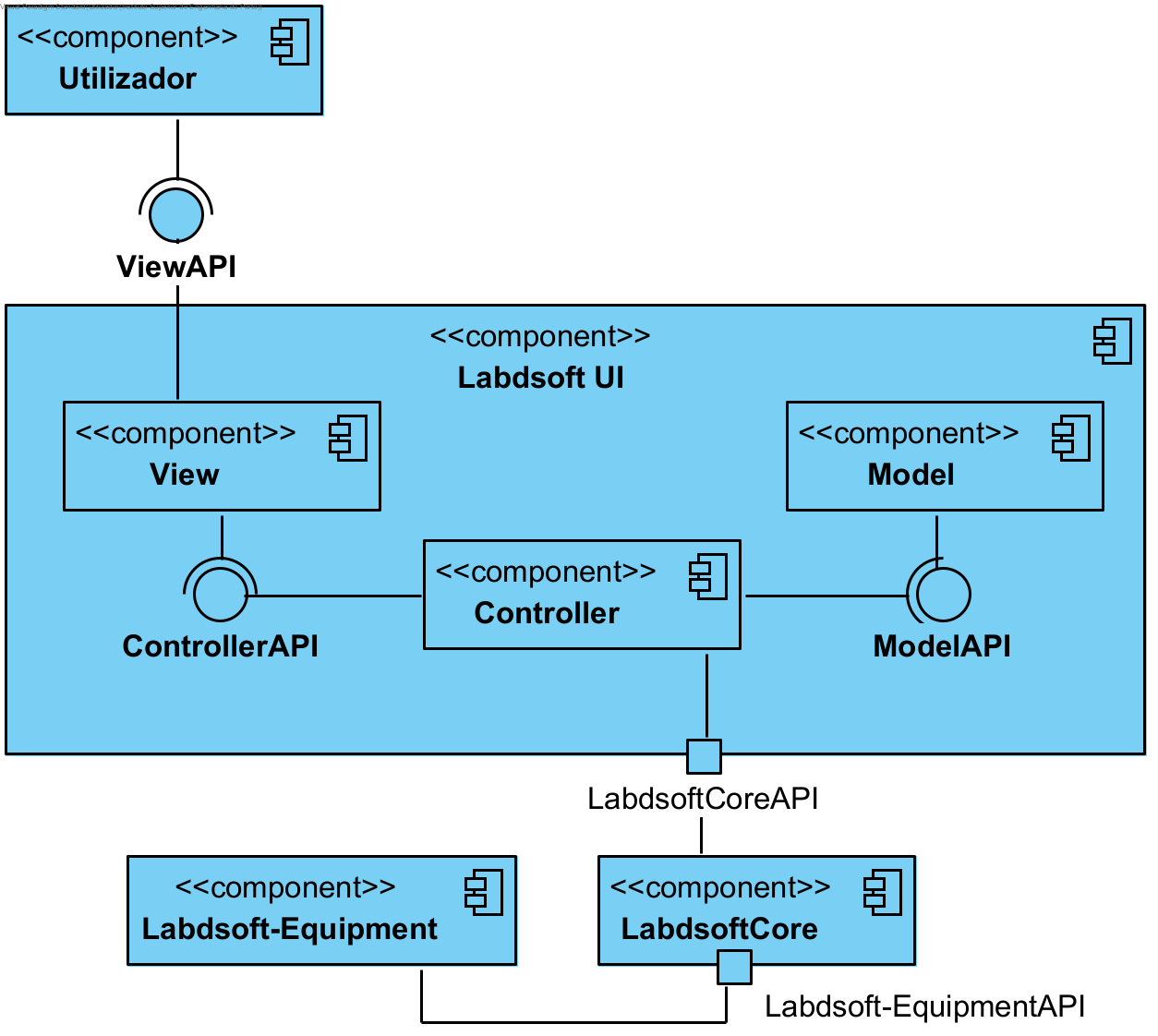


Figura - Vista Lógica Nível 3 Frontend

# Tecnologias para Implementação

## Landbot

Chatbots são ferramentas utilizadas para facilitar comunicação com clientes e aumentar a disponibilidade de oferta de serviços e apoio, com a capacidade de dialogar em linguagem natural por meio de mensagens. Podem ser pré-programados para seguir apenas certos fluxos de conversa e permitir escolhas limitadas, ou utilizar inteligência artificial para entender melhor necessidades dos utilizadores e para que estes mesmos tenham maior liberdade ao exploras as funcionalidades. Chatbots são comummente utilizados em áreas como o apoio ao cliente, campanhas de marketing, venda de produtos.

Entre as razões para o crescimento desta tecnologia dentro de várias indústrias, está a sua capacidade de oferecer resposta imediata, precisa e correta às necessidades e dúvidas dos clientes, disponível a toda a hora, reduzindo drasticamente o tempo de resposta assim como a necessidade de funcionários para efeitos automatizáveis.

Posto isto, a clínica CITO requisitou o desenvolvimento de um chatbot para funções como as acima descritas. Sendo que nenhum membro da equipa tinha experiência prévia com tecnologias como esta, optou-se pela utilização de algo que disponibilizasse a construção de um bot pré-programado, relativamente simples como ferramenta introdutória. No final, o grupo acabou por escolher utilizar o Landbot, um chatbot builder com muito pouca intervenção de programação, que oferece uma interface que facilita a construção e visualização do fluxo de informação. O produto final do desenvolvimento da equipa está presente no anexo 3.

Foi também idealizado um Simple Sequence Diagram expondo o *flow* de uma conversa entre o utilizador e o chatbot, onde estão presentes as funcionalidades que podem ser acedidas, e a comunicação entre o próprio sistema e o chatbot (anexo 4).

Primeiramente o chatbot pergunta ao utilizador o que pretende fazer, consultar lugar na lista de espera ou gerir marcações. Se escolher consultar a lista de espera, o chatbot pergunta-lhe de que órgão quer saber o lugar na lista de espera. Se o utilizador estiver na lista de espera é lhe indicado o seu lugar se não, o chatbot notifica o utilizador que não se encontra na lista de espera daquele órgão e encerra a conversa. No caso de escolher gerir marcações, é lhe pedido que insira o NIF para efeitos de autenticação, e posteriormente é pedido ao utilizador se quer marcar ou consultar as marcações. Se escolher marcar, será pedido para escolher uma data e depois é enviado ao médico um email sobre a solicitação da consulta. Se escolher consultar, será apresentado ao utilizador a lista das suas marcações, e o chatbot pergunta se quer cancelar alguma marcação. Se quiser cancelar uma marcação, o utilizador seleciona a marcação e o médico é notificado por email com a solicitação de cancelamento. Se não quiser cancelar nenhuma marcação o chatbot encerra a conversa.

## Google Cloud Vision

O grupo decidiu utilizar para os requisitos REQ\_23 e REQ\_24 a tecnologia Google Cloud Vision pela sua compatibilidade com as tecnologias de desenvolvimento previamente definidas (Springboot). A abundante documentação e compatibilidade tecnológica traduz-se em otimização de tempo e menor curva de aprendizagem.  
O Spring Cloud, fornecido pela Google Cloud, oferece bibliotecas convenientes para interagir com a API Vision através de um aplicativo Spring. Essas bibliotecas incluem classes de configuração automática e auxiliares e classes de modelo do Spring Boot Template para permitir que os desenvolvedores comecem a usar a API Vision rapidamente.

A API Vision oferece modelos avançados de machine learning pré-treinados pelas APIs REST e RPC. Potencia a rotulação de imagens, a sua classificação rápida em milhões de categorias predefinidas, Deteção de objetos, leitura de textos impressos e manuscritos e criação meta dados em imagens.

Uma imagem com texto

Descrição gerada automaticamenteDe forma a Integrar as funcionalidades API Vision numa aplicação Spring é necessário aceder ao ficheiro *pom.xml* e adicionar a dependência Spring Cloud GCP.

Figura - Depêndencia Spring Cloud GCP

Figura 13 - Depêndencia Spring Cloud GCP

É também necessário criar conta na plataforma do serviço e usar uma chave de serviço de forma a autenticar no Google Cloud.

### Análise de Imagem

Depois de configurar as dependências da Vision do Spring Cloud do Google Cloud no classpath, é possível começar imediatamente a processar imagens conseguindo uma instância de CloudVisionTemplate usando a injeção de dependências do Spring.

O CloudVisionTemplate é um wrapper em torno das bibliotecas de cliente da API Vision e permite processar imagens facilmente por meio da API Vision.

Figura - Injeção de dependência CloudVisionTemplate

Entre as várias análises possíveis a imagens e vídeos, destacamos a Deteção de Rostos e a Deteção de conteúdo explícito (SafeSearch)

Face ao primeiro, a Google permite deteção de vários rostos dentro de uma imagem com os principais atributos faciais associados, como estado emocional ou wearing headwear.

A Detecção do SafeSearch identifica conteúdo adulto ou violento em uma imagem. Esse recurso usa cinco categorias (adult, spoof, medical, violence e racy) e retorna a probabilidade de que cada uma esteja presente em determinada imagem. Veja detalhes sobre esses campos na página SafeSearchAnnotation.

A API Vision pode realizar a deteção de recursos em um arquivo de imagem local enviando o conteúdo do arquivo de imagem como uma string codificada em base64 no corpo da solicitação. Esta pode também realizar a deteção de recursos em um arquivo de imagem remoto localizado no Cloud Storage ou na Web. Ao enviar uma solicitação de arquivo remoto, você especifica o URL do arquivo e não precisa enviar o conteúdo do arquivo de imagem no corpo da solicitação.

# Implementação

## Modelo Relacional da Base de Dados

Na hiperligação seguinte encontra-se o modelo relacional da base de dados da solução gerado pelo Intellij.

Neste modelo relacional (anexo 2) estão representadas todas as relações entre cada elemento do programa e também como é persistida cada informação usada pelo software desenvolvido e as API’s.

## API e Simulador Health Tracker

O nosso sistema permite associar dispositivos Health Tracker a pacientes e atribuir-lhes também um protocolo médico, que irá definir os valores de referência para os sinais vitais.

Para interpretar e avaliar a estabilidade do paciente, foi criada uma API que retorna o estado dos sinais comparativamente às configurações dos protocolos. Esta API é simples e apenas necessita do identificador do dispositivo, os valores dos sinais vitais reais e os valores protocolares:

Uma imagem com mesa

Descrição gerada automaticamente

Figura - Estrutura de dados recebida na API Health Tracker

Cada parâmetro dos sinais vitais é comparado com o seu valor protocolar, e caso o valor esteja fora do intervalo desejado é adicionada a seguinte informação ao estado do paciente:

* Pressão arterial:



* Temperatura corporal:



* Batimentos cardíacos:



Um dos requisitos desejados envolve um simulador que teste o comportamento da API descrita anteriormente. A equipa decidiu utilizar o Postman como simulador pois é possível criar *cron jobs* através da opção *scheduled runs*. Infelizmente, por incompatibilidade não foi possível fazer a simulação síncrona de 20 em 20 segundos.

Efetuando o pedido à URL do simulador, a API Core analisa os dados de entrada e efetua a comunicação com a API Health Tracker. O fluxo do simulador está descrito diagrama de sequência referente ao anexo 5.

Consoante a resposta recebida pela API, é enviado um email ao médico responsável com os pormenores do alerta detalhados.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Mensagem e Assunto do email alerta

Uma imagem com texto

Descrição gerada automaticamente

Figura - Exemplo email alerta recebido

## Requisitos

Neste subcapítulo serão apresentados os diagramas de sequência CRUD dos requisitos. Uma vez que os requisitos se mantêm com a mesma lógica de implementação foram reutilizados os diagramas genéricos desenhados na iteração anterior. Nesta 2º iteração, a equipa não entregou o desenvolvimento dos requisitos REQ\_21, REQ\_23, REQ\_24, e REQ\_32. De forma a ser possível implementar os requisitos dependentes do Health Tracker e do algoritmo match, foram desenvolvidos requisitos da iteração anterior, nomeadamente o REQ\_12 e REQ\_14.

Para esta iteração o *product owner* sugeriu os seguintes requisitos:

Uma imagem com mesa

Descrição gerada automaticamente

Figura - Proposta de funcionalidades para a 2º Iteração

**Diagrama genérico CREATE**

**Uma imagem com texto

Descrição gerada automaticamente**O Diagrama de Create é utilizado principalmente pelo REQ\_19, REQ\_22, REQ\_28.

Figura - Diagrama genérico pedido create

**Diagrama genérico READUma imagem com texto

Descrição gerada automaticamente**

Figura - Diagrama genérico pedido read

O Diagrama de Read é utilizado principalmente pelo REQ\_20, REQ\_25.

**Diagrama genérico UPDATE**

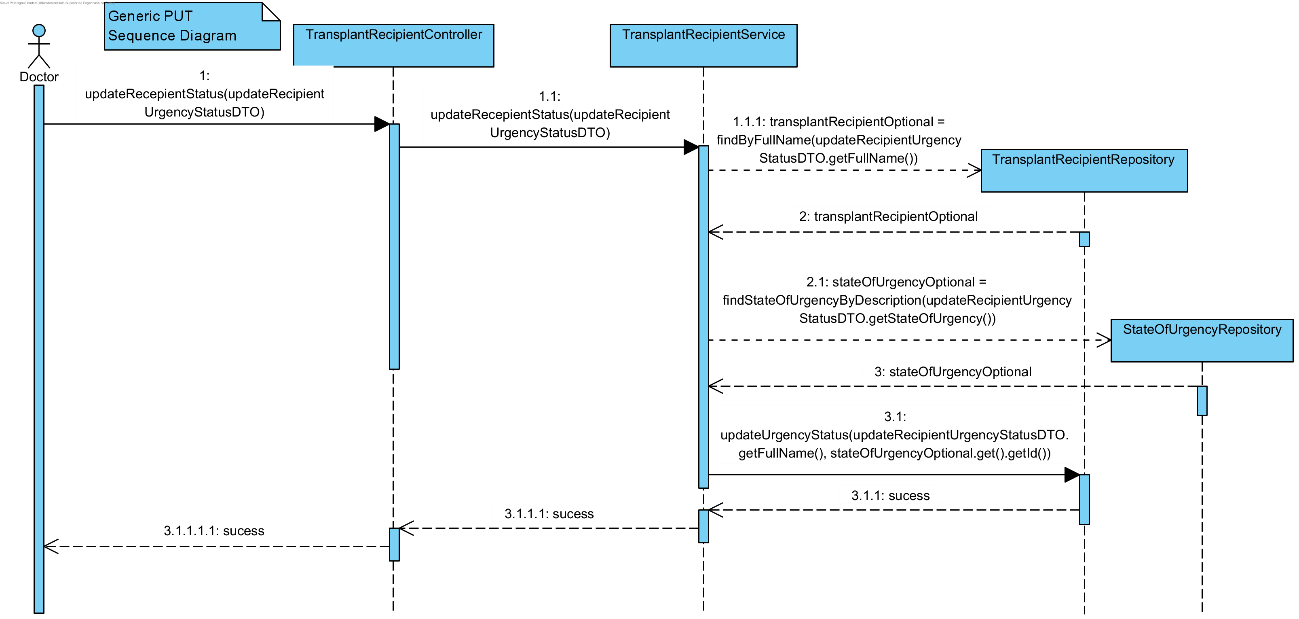
****O Diagrama de Update é utilizado principalmente pelo REQ\_27.

Figura - Diagrama genérico pedido update

# Conclusões do trabalho

O presente trabalho, no seu âmbito global, debruçou-se sobre a temática do levantamento de requisitos, comunicação direta com o cliente, engenharia, arquitetura e desenvolvimento de software. Esta última iteração baseou-se na arquitetura e desenvolvimento dos últimos requisitos.

Destes, destacam-se os requisitos envolventes no departamento de marketing, pela necessidade de integração com tecnologias de Inteligência Artificial e Machine Learning. Bem como o Chatbot e o desenvolvimento da API do Health Tracker pelos protocolos de comunicação SMTP.

Por adversidades e contratempos ao longo do processo, os requisitos REQ\_21, REQ\_23, REQ\_24 e REQ\_32 não foram totalmente implementados.

Assim sendo, conclui-se que o trabalho é passível de melhoria permitindo ao grupo implementar os restantes requisitos posteriormente.

ANEXO 1

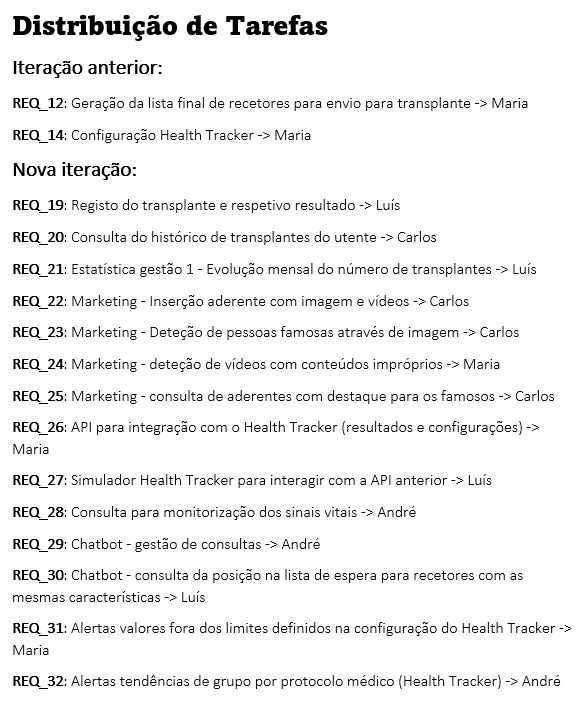


Figura - Distribuição de Tarefas

Anexo 2

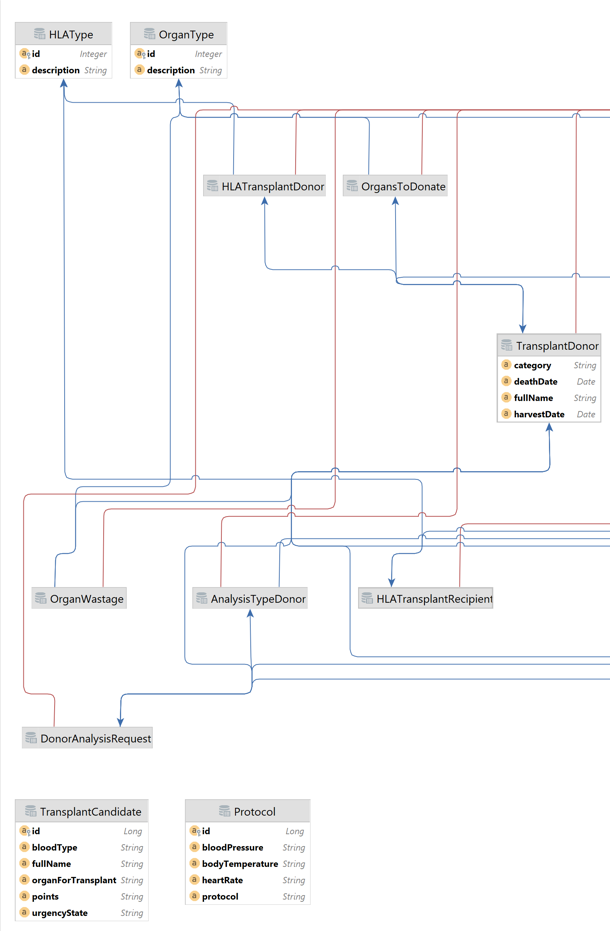


Figura - Modelo relacional 1

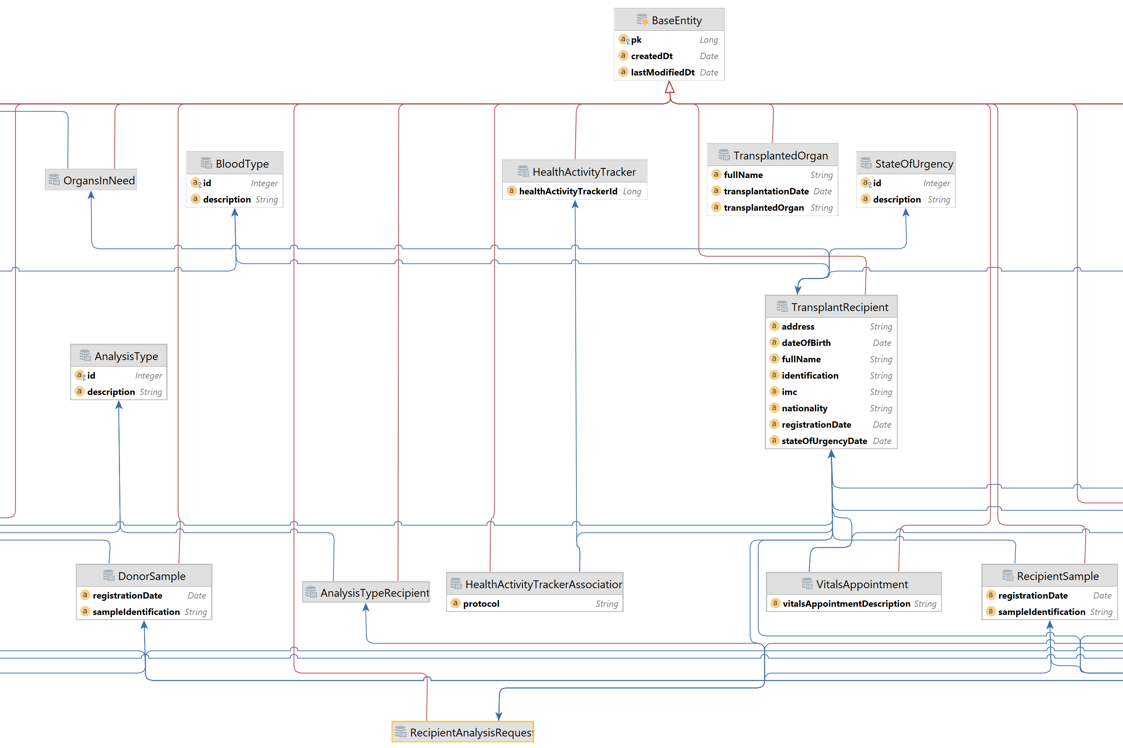


Figura - Modelo Relacional 2

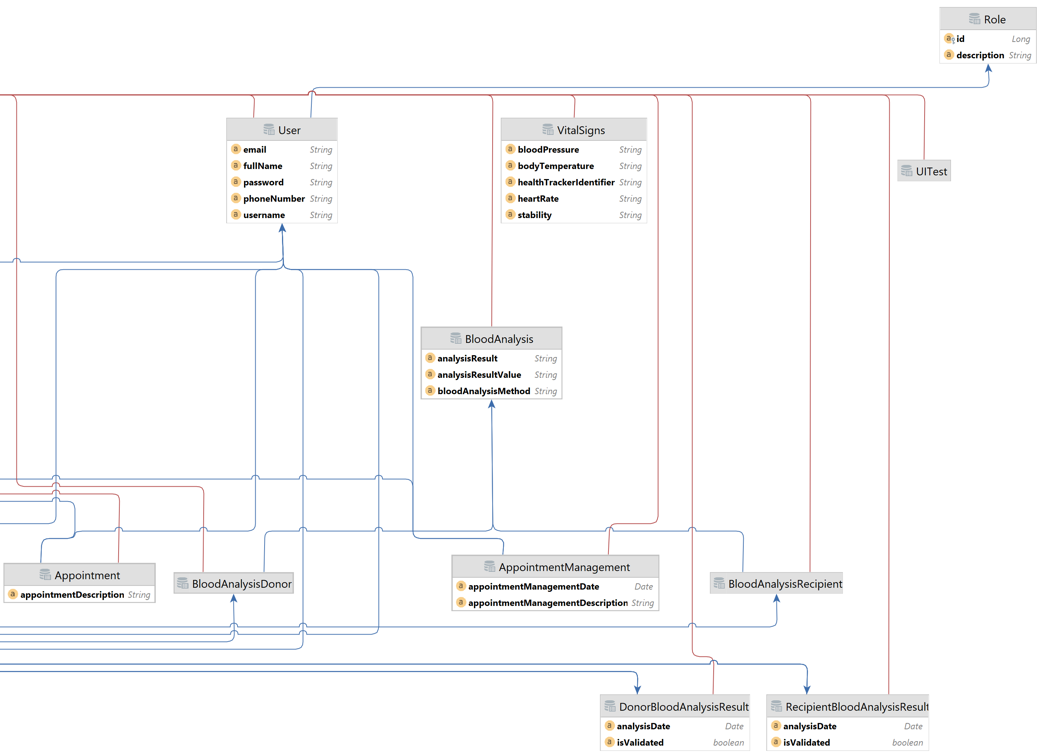


Figura - Modelo Relacional 3

ANEXO 3

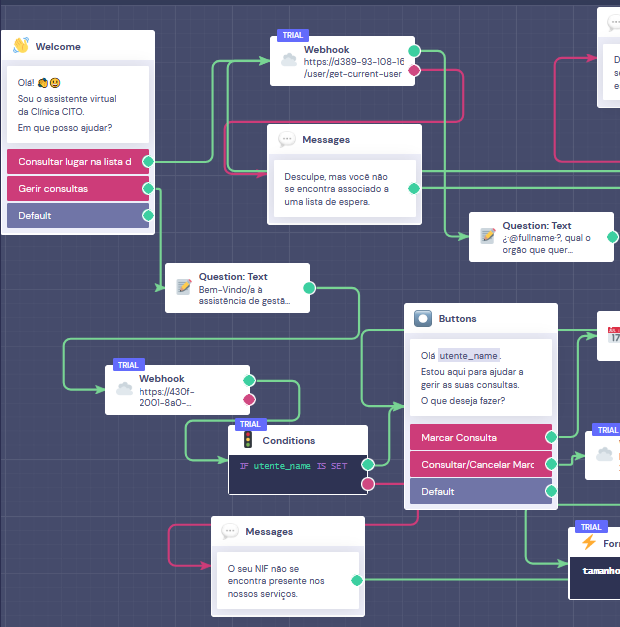


Figura - Landbot Diagrama 1

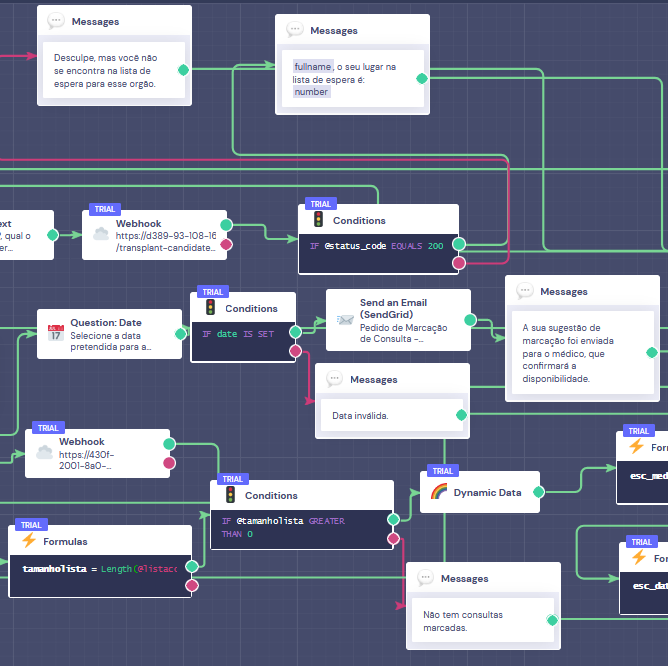


Figura - Landbot Diagrama 2

Uma imagem com texto, interior, captura de ecrã

Descrição gerada automaticamente

Figura - Landbot Diagrama 3

Anexo 4

Uma imagem com texto

Descrição gerada automaticamente

Figura - Landbot SSD

Anexo 5

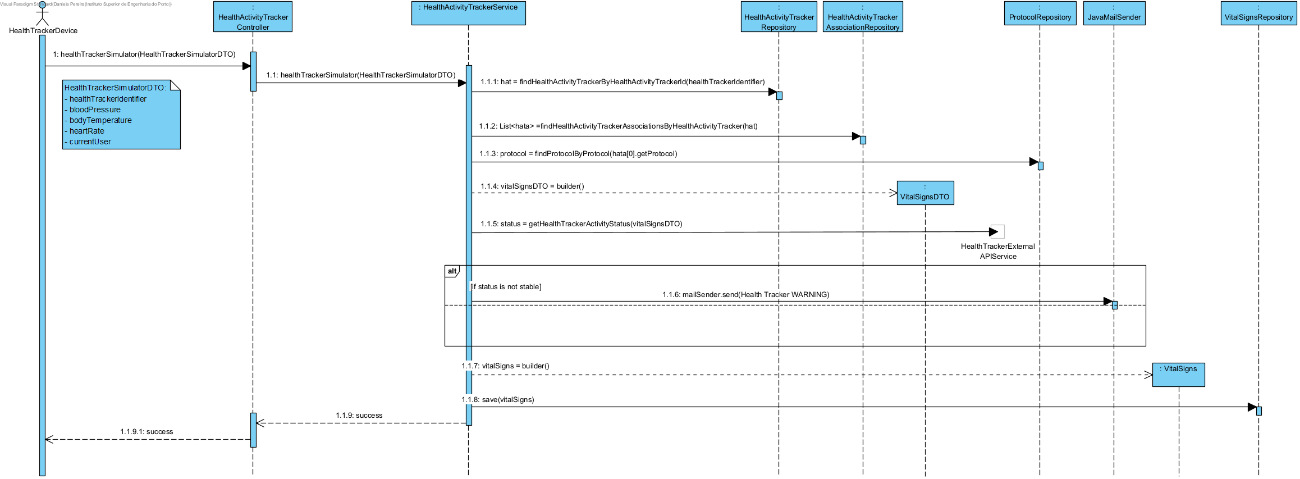


Figura - Diagrama de Sequência do Simulador com integração à API Health Tracker