Sistema web para farmácia solidária utilizando microsserviços

Bárbara Maria de Souza Lopes

JUIZ DE FORA

DEZEMBRO, 2018

Sistema web para farmácia solidária utilizando microsserviços

Bárbara Maria de Souza Lopes

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Sistemas de Informação

Orientador: Vânia de Oliveira Neves

JUIZ DE FORA
DEZEMBRO, 2018

Sistema web para farmácia solidária utilizando microsserviços

Bárbara Maria de Souza Lopes

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Aprovada por:

Vânia de Oliveira Neves
Doutora em Ciências da Computação e Matemática Computacional

Alessandreia Marta de Oliveira Julio
Doutora em Computação

Bárbara de Melo Quintela
Doutora em Modelagem Computacional

JUIZ DE FORA
03 DE DEZEMBRO, 2018

A Deus, pela força e coragem.

Ao meu noivo, por não me deixar desistir.

Resumo

Na correria do dia a dia, as pessoas normalmente se julgam sem tempo para lidar com questões que são cruciais para o meio ambiente e que, inclusive, podem ajudar outras pessoas. Medicamentos em desuso perdem sua validade e são descartados muitas das vezes de forma incorreta, sendo que ao mesmo tempo existem pessoas que necessitam e não possuem condições de adquiri-los. Nesse contexto, o presente trabalho traz o desafio de desenvolver uma aplicação web baseada na arquitetura de microsserviços. Esse estilo arquitetural vem sendo desenvolvido nos últimos anos, tendo como pioneiros a Amazon e a Netflix, e mostrou-se de grande valia devido a sua grande capacidade de reuso, uma vez que a aplicação desenvolvida poderá ser utilizada por outros sistemas da comunidade.

Palavras-chave: Microsserviços, Desenvolvimento de Software, Arquitetura.

Abstract

In everyday day-to-day running, people often feel they have no time to deal with issues that are crucial to the environment and can even help other people. Disused medicines lose their validity and are discarded many times incorrectly, while at the same time there are people who are in need and are not able to acquire them. In this context, the present work presents the challenge of developing a web application based on the microservice architecture. This architectural style has been developed in recent years, pioneered by Amazon and Netflix, and proved to be of great value because of its great reusability, since the application developed could be used by other systems in the community.

Keywords: Microservices, Software Development, Architecture.

Agradecimentos

Agradeço primeiramente a Deus, por abençoar o meu caminho durante toda a minha caminhada.

Ao meu noivo Frederico Alves, pela paciência nos momentos de estresse e compreensão nos momentos de ausência.

Sou grata à empresa Stefanini e ao Thiago Ribeiro, meu mentor, por me ensinar a prática dos conhecimentos adquiridos na faculdade.

À minha família pelo amor e apoio incondicional.

Obrigada a todos que direta ou indiretamente fizeram parte da minha vida acadêmica e em especial à minha orientadora, Vânia Neves pela oportunidade e apoio na elaboração deste trabalho.

"Tudo é do Pai, toda honra e toda glória

é Dele a vitória alcançada em minha vida".

Frederico Cruz

Sumário

Lista de Figuras

Lista de Abreviações

UFJF	Universidade Federal de Juiz de Fora
SOA	Service-Oriented Architecture
API	Application Programming Interface
HTTP	HyperText Transfer Protocol
REST	Representational State Transfer
ESB	Enterprise Service Bus
SGBD	Sistema Gerenciador de Banco de Dados
URI	Uniform Resource Identifier
JSON	JavaScript Object Notation
XML	Extensible Markup Language
HTML	HyperText Markup Language
SVN	Subversion
SQL	Structured Query Language
CSS	Cascading Style Sheets

1 Introdução

Nos últimos anos, diversas áreas como educação, medicina, comércio de bens e serviços e lazer têm se tornado cada vez mais dependentes de sistemas de informação. Nota-se a presença maciça de softwares que cada vez exigem-se que atendam diferentes objetivos e de maneira eficiente.

A partir disso, é possível identificar vários setores onde ainda há carência de sistemas de informação, principalmente setores que não possuem recursos para o seu desenvolvimento, como por exemplo, no trabalho social. Um dos papeis das universidades públicas, como a UFJF, é promover formas de melhorias e bem-estar para a população a sua volta e isso pode ser concretizado por meio desses trabalhos sociais. Um exemplo disso é encontrar soluções para o descarte correto de medicamentos. Uma vez que é comum adquirir mais medicamentos que o necessário, se o descarte for realizado de maneira inadequada poderá poluir solos e rios, por exemplo, prejudicando o meio ambiente. Por outro lado, se houver meios de recolher os medicamentos que não terão mais uso e repassar a quem necessita não apenas resolve o problema do descarte inadequado como também auxilia a população que não tem recursos para sua compra.

Nesse sentido, o presente trabalho tem o objetivo de projetar um sistema capaz de auxiliar a comunicação entre doadores de medicamentos e recebedores dessa doação. Com isso, espera-se que o meio ambiente deixe de receber na natureza medicamentos descartados de forma incorreta. Uma vez que pretende-se futuramente reutilizar parte do sistema em outros serviços e/ou sistemas, optou-se por desenvolvê-lo baseando-se em uma arquitetura orientada a microsserviços. Isso permite que softwares relacionados a outros contextos possam utilizá-lo como entrada de dados para outros fins, oferecendo ainda a capacidade de que cada microsserviço poder ser feito em uma linguagem diferente, e ser testado de forma individual, dando maior flexibilidade e aplicabilidade no mercado atual(LEWIS E FOWLER, 2014).

1.1 Apresentação do Tema

Tendo em mente a informatização de trabalhos sociais, chega-se ao departamento de Farmacologia da UFJF, o qual pretende iniciar um trabalho de arrecadação de medicamentos em bom estado de uso das pessoas que não necessitam dos mesmos para, posteriormente, identificar, por meio de profissional capacitado, as pessoas que necessitam.

Dessa forma, este projeto visa a construção de um sistema web que permita que: a) os possíveis doadores entrem em contato para que possam realizar suas contribuições; e b) quem necessite dos medicamentos disponíveis para doação se candidatem para recebê-los.

Funcionará assim: a pessoa que possui um medicamento em desuso, acessará o site e preencherá um formulário dizendo se quer levar o medicamento em algum dos pontos de apoio ou se deseja que ele seja recolhido por voluntários. O departamento de Farmacologia receberá esse medicamento e fará testes necessários verificando as características físicas, químicas, biológicas, e microbiológicas. Os que estiverem em boas condições de uso, estarão disponíveis no site para que pessoas que precisem possam fazer a solicitação. Ao fazer essa solicitação, o medicamento ficará reservado por alguns dias no ponto de apoio e para recebê-lo, será preciso apresentar a prescrição médica atestando sua necessidade de uso.

A arquitetura do sistema será construída baseando-se em microsserviços. Microsserviços são pequenos serviços operando via troca de mensagens, onde esses pequenos serviços podem ser módulos de um sistema maior, cada um podendo ou não ter sua linguagem, comportamento e estrutura distintas. Ou seja, cada microsserviço é independente e toda comunicação é feita via mensagens, dando liberdade a equipe de desenvolvimento de escolher a melhor estrutura (DRAGONI et al., 2017).

Baseado na arquitetura proposta, serão desenvolvidos pequenos serviços, onde cada um será implementado de forma independente, possibilitando que diferentes tipos de aplicação os consumam. Com isso, o sistema proposto além de cumprir seu objetivo inicial de recolher e doar medicamentos, também proporcionará a disponibilização de artefatos que poderão ser reutilizados futuramente em outros projetos ou para pesquisas acadêmicas tanto da UFJF quanto da comunidade em geral.

1.2 Justificativa

A escolha do tema para este trabalho de conclusão de curso surgiu pela possibilidade de se solucionar um grande problema social – a carência e desperdício de medicamentos, através do desenvolvimento de um sistema web.

Tendo em vista que o sistema a ser desenvolvido será baseado em uma arquitetura orientada a microsserviços, ele também trará a possibilidade de expandir o desenvolvimento de software para outros domínios e setores, uma vez que grande parte dos serviços implementados poderão ser reutilizados.

1.3 Objetivos

O objetivo principal deste trabalho é produzir um sistema para auxiliar o departamento de farmacologia no desenvolvimento de seu projeto social, recebendo medicamentos em desuso e distribuindo às pessoas que não têm condição de comprá-los, assim, reduzindo também o descarte incorreto no meio ambiente.

Os objetivos específicos do trabalho envolvem:

- Aprimorar o conhecimento no desenvolvimento de software;
- Avançar no estado da arte no desenvolvimento de software utilizando microsserviços;
- Aplicar os conceitos aprendidos em sala de aula em um projeto real;
- Aprimorar o conhecimento nas tecnologias envolvidas, como Java e desenvolvimento baseado em microsserviços;
- Disponibilizar os microsserviços desenvolvidos nesse software para que sejam reutilizados em projetos de desenvolvimento de software futuros;
- Conscientizar a sociedade quanto à importância da doação de medicamentos dentro do prazo de validade.

1.4 Metodologia

O estilo do presente trabalho é apresentação de um produto, sendo o método de pesquisa do tipo exploratório. O caráter exploratório se deve pela aquisição de conhecimento fundamentada em referências bibliográficas e pela apresentação de novo sistema aplicando técnicas aprendidas durante a graduação. Além disso, serão estudadas novas técnicas que estão surgindo no mercado, como microsserviços, que irá agregar no resultado final do trabalho trazendo a possibilidade de disponibilizar as informações para uso de outros produtos.

A pesquisa exploratória, segundo PRODANOV E FREITAS (2013), "se encontra na fase preliminar, tem como finalidade proporcionar mais informações sobre o assunto que vamos investigar, possibilitando sua definição e seu delineamento".

A proposta do projeto é o desenvolvimento de um sistema web que gerencie a doação e a distribuição de medicamentos. Desse modo, a pesquisa irá avançar através da metodologia descrita a seguir.

Inicialmente, foi feita uma revisão da literatura sobre os temas necessários para o desenvolvimento deste projeto. Durante essa atividade, foram realizadas pesquisas bibliográficas a respeito de diversos assuntos relacionados, procurando abordar os pontos fundamentais que serão utilizados como processo de desenvolvimento de software, projeto arquitetural, microsserviços, etc.

Na sequência, foi definido o processo de desenvolvimento de software a ser seguido. Esse processo contém a fase de análise - que se concentrou no levantamento de requisitos -, projeto - em que os conceitos de arquitetura de microsserviços foram aplicados -, implementação - envolveu a construção de um sistema web - e implantação - detalhamento da implantação de microsserviços através de contêineres. Ressalta-se que a contribuição deste projeto foi no desenvolvimento arquitetural e, por isso, apenas algumas funcionalidades do sistema foram implementadas.

1.5 Organização

O presente trabalho está estruturado em capítulos da seguinte forma:

- Capítulo 1: Introdução – breve apresentação do tema, objetivos e a metodologia deste trabalho;

- Capítulo 2: Fundamentação teórica - revisão de conceitos fundamentais para o entendimento deste trabalho como processo de desenvolvimento de software, microsserviços e arquitetura de software, além da apresentação das tecnologias utilizadas para o desenvolvimento do sistema;

- Capítulo 3: Trabalhos relacionados – apresenta uma comparação com outros sistemas web existentes que se assemelham ao sistema desenvolvido bem como uma comparação de outros sistemas desenvolvidos baseados em arquitetura de microsserviços;

- Capítulo 4: Sistema farmácia solidária - descreve todas as etapas para o desenvolvimento do sistema, desde o levantamento de requisitos até a implementação do mesmo;

- Capítulo 5: Conclusão - considerações finais relacionadas aos objetivos do trabalho e sua contribuição, além de indicar os trabalhos futuros.

2 Fundamentação teórica

2.1 Processo de desenvolvimento de software

Um processo de desenvolvimento de software é um conjunto de atividades onde define-se quem faz o que, como e quando com o objetivo final de obter um produto de software. Segundo PRESSMAN e MAXIM (2016), essas atividades constituem um conjunto mínimo para se obter um produto de software.

É uma área de estudo de Engenharia de Software que consiste de boas práticas para se obter um software de qualidade e com os requisitos definidos atingidos. Para desenvolver um projeto de software, é necessário seguir alguns passos, porém deve-se entender que não existe um processo perfeito, sendo recomendado cada empresa aperfeiçoar o seu. SOMMERVILLE et al. (2008) cita que, em casos de sistemas críticos, é necessário um processo bem estruturado, e para sistemas de negócios com requisitos que mudam rapidamente, um processo mais flexível e ágil é mais eficaz.

O primeiro passo consiste na viabilidade econômica, verificando se terá lucros suficientes para o projeto. Após isso, deve-se começar as reuniões com os clientes, para o levantamento de requisitos. Segundo SOMMERVILLE et al. (2008), requisitos são descrições ou restrições que definem as propriedades de um sistema. Com os requisitos é possível visualizar as necessidades do cliente e assim, identificar as regras de negócio, transformando-as em funcionalidades do sistema. Nessa fase, o analista e o usuário devem entender qual é o problema a ser solucionado e quais as prioridades do projeto, para então gerar a especificação do projeto.

A especificação do projeto tem a incumbência de descrever de forma precisa o software em questão, nos mínimos detalhes, construindo modelos a fim de representar o sistema a ser desenvolvimento. Um dos modelos é o projeto de arquitetura, descrito em mais detalhes na Subseção 2.1.1. Em seguida, vem a parte da implementação desta especificação, ou seja, sua codificação de fato. Segundo SOMMERVILLE et al. (2008), "o estágio de implementação do desenvolvimento de software é o processo de conversão

de uma especificação de sistema em um sistema executável". Escolhe-se uma linguagem de programação, banco de dados a utilizar, arquitetura, padrão de interface gráfica, etc. Com a codificação finalizada, é necessário testar o sistema já pronto, para garantir que todos os pontos da especificação estão aderentes. Dependendo da forma que foi conduzido o processo de desenvolvimento, se faz necessário elaborar o documento do software que será utilizado para entendimento do sistema e para futuras manutenções do mesmo.

Após todas as atividades concluídas, o sistema é liberado para produção assistida com os usuários. É nesta fase que eles fazem o teste do sistema e recebem um treinamento para uso e ao concluí-la, o sistema está entregue e em produção. Quando um sistema está entregue em produção, futuras modificações entram agora no ciclo de manutenção do software, podendo ser melhorias ou correções de erros.

É importante enfatizar que estas atividades acima descritas podem ser executadas de forma sequencial ou paralela ou em interações, dependendo do modelo de processo de desenvolvimento utilizado. Exemplos de modelos de processo de desenvolvimento de software incluem:

- Cascata:

  Sendo um dos modelos mais populares em engenharia de software, o modelo cascata ou top down, segundo ROYCE (1987) ocorre de forma linear, agrupadas em tarefas e executadas sequencialmente, onde a saída é entrada para outra.

  As fases são: Análise de requisitos, projeto, implementação, testes (validação), integração, e manutenção de software (SOMMERVILLE et al., 2008).

- RUP:

  É uma especialização do Processo Unificado que, por sua vez, é baseado no modelo incremental. O RUP (do inglês, Rational[1] Unified Process) adota algumas premissas para tentar minimizar riscos e problemas. VIANNA (s.d.) cita as seguintes premissas:

  – Uso de iterações para evitar o impacto de mudanças no projeto;

  – Gerenciamento de mudanças;

[1] http://www-01.ibm.com/software/rational/systems/index.html

– Abordagens dos pontos de maior risco o mais cedo possível;

As quatros fase do RUP são (VIANNA, s.d.):

– Concepção - entendimento da necessidade e visão do projeto;

– Elaboração - especificação e abordagem dos pontos de maior riscos;

– Construção - desenvolvimento principal do sistema;

– Transição - ajustes, implantação e transferência de propriedade do sistema;

O RUP costuma ser visto como um processo pesado devido ao grande número de atividades e artefatos e à rigidez e controle. No entanto, há uma variação chamada Processo Unificado Ágil, que visa desenvolver apenas um pequeno conjunto de atividades e artefatos, tornando-se um processo mais leve e adaptativo (LARMAN, 2004).

Outros exemplos de modelos de processo de software são:

- Modelo de Desenvolvimento Evolucionário;
- Modelo Espiral;
- Modelo Incremental;
- Modelo RAD;
- Modelo Desenvolvimento Formal de Sistemas;
- Modelo de Desenvolvimento Orientado a Reuso

2.1.1 Arquitetura de software

Por existir várias definições de arquitetura de software, o *Software Engineering Institute* compilou uma lista completa dessas definições(SEI, 2010). Para FIELDING e TAYLOR (2000), uma arquitetura de software é uma abstração dos elementos de execução durante alguma fase de sua operação. Ele considera ainda que um sistema pode ser composto de muitos níveis de abstração e muitas fases de operação, cada um com sua própria arquitetura. Assim, uma arquitetura pode representar uma abstração do comportamento do sistema em um determinado nível, de tal forma que os elementos arquiteturais são

traçados pelas interfaces abstratas que eles fornecem para os outros elementos neste nível (FIELDING e TAYLOR, 2000). Uma outra definição bastante utilizada é a de GARLAN (2000), que diz que arquitetura de software é uma "estrutura de componentes de um programa/sistema, os relacionamentos entre esses componentes, os princípios e diretrizes que governam os projetos e a evolução dos softwares".

Segundo SOMMERVILLE et al. (2008), a arquitetura de um sistema pode aderir a um ou mais estilos arquiteturais que definem meios de selecionar e apresentar blocos de construção de arquitetura. Exemplos de estilos arquiteturais conhecidos incluem cliente-servidor, camadas e fluxo de dados. Para BASS et al. (2003) (apud SOMMERVILLE, 2007) existem três vantagens de projetar e documentar uma arquitetura de software:

> 1. Comunicação com os *stakeholders*. A arquitetura é uma apresentação em alto nível do sistema que pode ser usada para enfocar a discussão entre os diferentes *stakeholders*.
> 2. Análise do sistema. Tornar a arquitetura do sistema explícita em um estágio inicial de desenvolvimento do sistema requer alguma análise. Decisões de projeto de arquitetura têm profundo efeito sobre se o sistema pode atender aos requisitos críticos, como desempenho, confiabilidade e facilidade de manutenção.
> 3. Reuso em larga escala. Um modelo de arquitetura de sistemas é uma descrição compacta e administrável de como um sistema está organizado e de como os componentes operam entre si. A arquitetura de sistemas é muitas vezes a mesma para requisitos similares e, assim pode apoiar o reuso do software em larga escala.

Uma vez que se pode olhar arquiteturas de sistemas sob várias perspectivas (SOMMERVILLE et al., 2008), este texto adota a convenção utilizada pela maior parte dos pesquisadores da área de microsserviços e considera como arquitetura monolítica toda arquitetura que, apesar de poder ser composta por vários componentes, gera apenas uma única aplicação. Por outro lado, considera-se como arquitetura de microsserviços uma arquitetura que é composta por vários componentes que trabalham juntos para fornecer serviços distribuídos. As subseções seguintes explicam essas e a arquitetura orientada a serviços com mais detalhes.

Arquitetura monolítica

Esta arquitetura é a mais antiga existente, formada por vários módulos que comunicam entre si. Entretanto toda a modularização é executada em uma única máquina, compar-

tilhando assim, recursos de processamento, memória, bancos de dados e arquivos (OPUS, s.d.).

Nesse formato, a medida que o tempo passa, vai se tornando cada vez mais custosa e dificultosa a manutenção, pois o sistema vai ficando mais complexo, e o uso de recursos da máquina aumenta. Há necessidade de compra de recursos de hardware, aumento de complexidade do código, o que acaba se tornando difícil de implementar novas funcionalidades, podendo ocasionar inconsistência no código já existente, e dificuldade de implantações em produção (CUNNINGHAM, 2014), ou seja, causa a degradação da arquitetura.

Arquitetura orientada a serviços - SOA

Arquitetura Orientada a Serviços, ou SOA (do inglês, *Service-Oriented Architecture*), é um conceito de arquitetura corporativa que "permite a criação de serviços de negócio interoperáveis que podem facilmente ser reutilizados e compartilhados entre aplicações e empresas"(ZONETTI, 2015).

Nessa arquitetura, informações que podem ser necessárias a outras áreas e sistemas são disponibilizadas em forma de serviços a serem consumidos por quem necessitar, podendo ter regras de negócios complexas, afim de solucionar algum problema de integração.

Existem processos e ferramentas que podem ser usados para implantar SOA, cada uma para um tipo de negócio, mas SOA em si não define alguma metodologia. SOA é puramente um conceito, não sendo algo que se possa comprar e instalar, nem um webservice em si, baseando-se no uso de serviços atômicos, independentes e com baixo acoplamento.

2.2 Microsserviços

Microsserviços são pequenos serviços que podem se comunicar entre si através de uma API HTTP com a responsabilidade de expor uma funcionalidade para o resto do sistema, cada uma independente das demais.

Podendo ser um serviço isolado na nuvem ou um processo no sistema operacional,

não existindo uma tecnologia padrão a ser aplicada. Desta forma, pode-se escolher a melhor forma de implementação, isto é, linguagem, plataforma de execução e banco de dados, com base em regras de negócio para que a funcionalidade agregue valor à organização.

De acordo com NEWMAN (2015), microsserviços possuem princípios e objetivos estratégicos junto às práticas de desenvolvimento de microsserviços. Esses princípios e objetivos são listados abaixo:

1. Utilização de automação;
2. Modelagem focada no domínio de negócio;
3. Implantação independente;
4. Descentralização;
5. Isolamento de falhas;
6. Abstração dos detalhes de implementação;

Com isso as falhas são reduzidas drasticamente pois diminui a complexidade tornando o desenvolvimento mais fácil, fazendo com que as equipes sejam independentes e as mudanças rápidas.

2.2.1 Arquitetura de microsserviços

Segundo LEWIS E FOWLER (2014), a arquitetura de microsserviços ainda não possui uma definição definida, porém ela possui características comuns que possibilita padronizar essa arquitetura. Essas características não são obrigatórias, mas se fazem presente na maioria dos sistemas que seguem essa arquitetura.

As características mais comuns presentes na arquitetura baseada em microsserviços, de acordo com LEWIS E FOWLER (2014), são:

- Componentização:

  Pode-se dizer que componente é uma unidade independente que pode ser atualizada ou substituída sem depender de nada.

A organização da arquitetura em microsserviços, busca dividir o software em serviços, componentes em processo diferentes que se comunicam através de protocolos simples como HTTP/REST.

A vantagem é a forma independente de modificação. Se necessitar alterar o serviço, basta republicar o mesmo com as alterações e não terá impacto algum diretamente na aplicação, a ponto de alterar vários pontos do código. Ao contrário de utilização de bibliotecas, que pode resultar na alteração de toda a aplicação.

- Organização por área de negócio:

  Fugindo do tradicional, onde os times são divididos por tecnologia, a abordagem de microsserviços tende a organizar os times por área de negócio. Sendo os times divididos dessa forma, eles podem explorar a tecnologia a favor do negócio, não se limitando a nada e trazendo benefícios para os mesmos.

- Tamanho do microsserviço:

  Ainda não se tem um tamanho definido apesar do termo "microsserviço".

- Produtos e não projetos:

  Tradicionalmente, ao construir uma aplicação, o desenvolvimento é tratado como projeto, onde ao fim deste a equipe é dissolvida, entregando a manutenção a outra equipe responsável pela mesma. Já em microsserviços, esse conceito é deixado de lado para seguir uma filosofia baseada na Amazon, que diz: "*You build, you run it*"(traduzido: "Você constrói, você executa"). Isto significa que a responsabilidade desse projeto, chamado de produto por esse conceito, é da equipe que o construiu, garantindo que todo o ciclo de vida do projeto será bem entendido. Além disso, a equipe que o desenvolveu também estará presente no dia a dia e em contato com os usuários, o que facilita a manutenção e desenvolvimento de novas funcionalidades.

- Comunicação simplificada:

  Aplicações em microsserviços devem ser coesas e desacopladas, onde se recebe uma requisição, processa-a e responde, utilizando os principais protocolos simples da web, como o HTTP/REST.

- Governança descentralizada:

  A vantagem de microsserviço é não ficar preso a uma tecnologia. Utilize a tecnologia que achar ideal para cada implementação.

- Infraestrutura automatizada:

  Devido a adoção de microsserviços ser diferenciada da tradicional, os aspectos de infraestrutura que a sustentam também devem ser fora da caixa, isto é, utilizam de novos conceitos, como deploy automatizado, conceitos de contêiner, testes automatizados, controles de versões entre outros.

- Projeto inovador:

  Um microsserviço pode ser visto como algo sempre em evolução. Por ser desacoplado, é possível enxergar melhorias contínuas e sempre em processo de aperfeiçoamento utilizando da melhor forma as tecnologias nele empregadas.

- Descentralização de banco de dados:

  Cada microsserviço pode ter seu próprio banco de dados. Não é preciso se limitar a um único banco de dados, visto que cada microsserviço pode demandar uma estrutura diferente das outras, sendo um modelo relacional ou no-sql por exemplo. Além do SGBD podendo ser diferente, de acordo com a necessidade ou o que julgar melhor a nível de adaptação e desempenho, seja em Oracle[2], Sql Server[3], PostgreSQL[4], etc. Isso complementa o desacoplamento dos microsserviços, onde a alteração de um banco não acarreta em nenhuma outra alteração na aplicação, a não ser naquele microsserviço.

2.2.2 SOA versus Microsserviços

Segundo ZONETTI (2015), muitos autores argumentam que arquitetura de microsserviços é apenas um sinônimo para a SOA. No entanto, eles apresentam algumas diferenças, como menciona WATTS (2017), tais como:

[2]https://www.oracle.com/br/index.html
[3]https://www.microsoft.com/pt-br/sql-server/sql-server-2017
[4]https://www.postgresql.org/

- SOA é baseado na ideia de compartilhar o máximo possível, quanto em microsserviços prega o compartilhamento do mínimo possível;

- SOA possui governança e padrões comuns, microsserviços foca em governança descentralizada e foco em colaboração e liberdade de escolha de acordo com o problema e situação;

- SOA utiliza comunicação via barramento ESB, que permite vários tipos de protocolos, enquanto microsserviços opta por protocolos mais simples como HTTP/REST;

- SOA utiliza plataforma comum para todos os serviços implantados, já em microsserviços, é possível que cada um esteja em uma plataforma diferente;

Em suma, SOA é adequado para integração de aplicações de negócios grandes e complexas, enquanto microsserviços é adequado para sistemas web bem particionados.

2.2.3 Tecnologias e ferramentas para desenvolvimento de microsserviços

A seguir, uma breve descrição das tecnologias que permitiram a construção do sistema 'farmácia solidária'.

Para o desenvolvimento de microsserviços é necessário definir uma linguagem de programação, uma ferramenta de virtualização, um protocolo de comunicação e um API Gateway. A seguir, uma breve descrição das tecnologias que foram utilizadas para a construção do sistema 'farmácia solidária'.

Java

Java é uma linguagem de programação orientada a objetos, desenvolvida pela Sun Microsystems[5] e agora pertencente a Oracle[6]. É uma linguagem de alto nível, simples, robusta, segura, extensível, bem estruturada e bem distribuída, além de possuir diversas características herdadas de outras linguagens. Ela possui diversas bibliotecas de classes que auxiliam a desenvolver sistemas rapidamente.

[5]https://www.oracle.com/sun/index.html
[6]https://www.oracle.com/br/java/

Sobre essa linguagem, o site oficial menciona que o Java "foi projetado para permitir o desenvolvimento de aplicações portáteis de alto desempenho para a mais ampla variedade possível de plataformas de computação"(JAVA, s.d.).

Spring Boot

Spring Boot é uma uma ferramenta para o desenvolvimento de aplicações que tem sido muito citado para o desenvolvimento de microsserviços em Java. Isso porque ele facilita o desenvolvimento, uma vez que fornece ferramentas para a configuração e adição de bibliotecas, sendo apenas necessário informar o que se deseja utilizar, para que o Spring Boot reconheça e faça a instalação.

De acordo com ANTONOV (2015), o Spring Boot "dará potência combinada com a flexibilidade que lhe permitirá produzir software de alta qualidade em um ritmo rápido".

Essa ferramenta soluciona a complexidade da inicialização e gerenciamento de dependências, além de resolver a questão de configuração de um projeto. Ao simplificar a execução do projeto, o desenvolvedor possui mais tempo para se dedicar nas regras de negócio.

BOAGLIO (s. d.) descreve essa ferramenta da seguinte maneira: "não se trata de um simples *framework*, mas de um conceito totalmente novo de criar aplicações web. Além de impulsionar o desenvolvimento para microsserviços, o Spring Boot ajuda na configuração importando e configurando automaticamente todas as dependências".

API REST

Microsserviços interagem entre si utilizando um mecanismo de comunicação entre processos e um dos mecanismos mais utilizados é o HTTP baseado em REST. REST é um estilo arquitetural simples e robusto cujo princípio básico é de divisão do cliente e servidor: o cliente não se preocupa com conexões em banco, gerenciamento de memória e cache; o servidor, por sua vez, também não se preocupa com interface, iteração humano-máquina e etc.

Dessa forma, até mesmo as requisições de um cliente são independentes, ou seja, pode-se mandar várias requisições para o servidor, porém cada uma delas deve conter

todas as informações necessárias. Outro fator importante é a capacidade de armazenar em cache requisições que possam responder a vários clientes, não necessitando ficar processando a mesma informação várias vezes, ganhando em desempenho.

O REST possui algumas regras básicas para que a comunicação cliente e servidor seja mais eficaz, mencionados por FIELDING e TAYLOR (2000):

1. Cada recurso, ou seja, cada elemento de informação, deve-se ter uma URI para ser acessado de forma rápida e prática.

2. Deve-se ter uma representação deste recurso para resposta ao cliente, que pode ser XML, JSON, HTML, etc;

3. Metadado tanto na requisição como na resposta, isto é, um cabeçalho de informações importantes para a comunicação, como *host* e HTTP da resposta;

4. Entregar todas as informações necessárias na resposta ao cliente;

5. Aplicação em camadas, permitindo que sejam facilmente alteradas, tanto para remoção quanto adição de novas camadas;

Spring Cloud Config

Fornecido pela Pivotal[7], o *Spring Cloud Config* é um recurso que permite que os microsserviços busquem suas propriedades em um servidor de configuração.

Segundo SILVA e SILVA (2017), os microsserviços consultam o *Spring Cloud Config* "para obter suas configurações na hora da inicialização. Podem ser desde configurações de acesso ao banco de dados até mesmo a porta em que desejamos que a aplicação suba".

Essa ferramenta permite gerenciar os arquivos de configuração de maneira externa e centralizada, facilitando os microsserviços que rodam em ambientes distintos. Esses arquivos de configuração podem ser armazenados de três formas: em repositório GIT ou SVN, em arquivo local ou em banco de dados.

[7]https://pivotal.io/

Eureka

Eureka[8] é uma solução de código aberto desenvolvida pela Netflix (SILVA e SILVA, 2017). Essa solução é responsável por registar informações de acesso dos microsserviços e por realizar checagens de *status* dos mesmo, verificando qual se encontra online para receber requisições. Outra funcionalidade dela é fornecer balanceamento de carga de instâncias da mesma aplicação.

O Eureka é composto pelos módulos Eureka Server e Eureka Client. SOUZA (s.d.) descreve esses módulos da seguinte forma:

- O Eureka Server consiste em uma aplicação que atua como um registrador de serviços (do inglês, *Service Registry*) permitindo que outras aplicações registrem suas instâncias. Com isso, ele controla os endereços registrados mantendo-os atualizados e sinalizando quando um serviço não está disponível;
- O Eureka Client é um componente Java que facilita a interação com Eureka Server;

Zuul

Segundo SOUZA (2018), Zuul[9] "é uma solução de roteamento dinâmico que possibilita monitoramento, resiliência e segurança para aplicações". Também foi desenvolvido pela Netflix e funciona como uma porta de entrada, onde toda requisição passa antes de ser direcionada para o microsserviço específico registrado no Eureka.

SILVA e SILVA (2017) explica o funcionamento dessa ferramenta da seguinte maneira: o Zuul recebe a requisição do cliente e consulta o Eureka qual a instância de microsserviço responde por aquela rota; se for uma rota segura, o zuul realizará também a autenticação antes de fazer o redirecionamento.

Spring Security OAuth

O Spring Security OAuth2[10] baseia-se no OAuth2[11] que, de acordo com SILVA e SILVA (2017), "é um *framework* de autenticação e autorização aberto, poderoso e flexível permi-

[8] http://spring.io/projects/spring-cloud-netflix
[9] https://github.com/Netflix/zuul
[10] https://spring.io/projects/spring-security-oauth
[11] https://oauth.net/2/

tindo que sua aplicação não fique manipulando diretamente as credenciais dos usuários". Esse *framework* define papéis e cria uma camada intermediária de autenticação e, com isso, a aplicação cliente solicita uma concessão de autorização ao usuário que é enviada ao servidor de autorização. Esse servidor de autorização autentica e valida o usuário e caso esteja tudo correto, ele emite um *token* para que o cliente possa acessar os outros serviços (SILVA e SILVA, 2017).

Docker

Docker[12] é uma plataforma de código aberto cujo objetivo é facilitar a criação, o desenvolvimento, o teste, a implantação e a execução de aplicações em ambientes isolados. Dessa forma, as aplicações terão alta disponibilidade e de maneira rápida. Ele foi escrito em Go[13], que é uma linguagem de programação de alto desempenho desenvolvida pela Google[14] (DIEDRICH, 2015).

Através do Docker é possível criar ambientes virtuais onde as aplicações são encapsuladas e iniciadas em um processo isolado do sistema operacional. Esses ambientes virtuais são chamados de contêineres, que é "um conjunto de processos isolados do restante do sistema, o que permite virtualização em nível de sistema operacional" (ROLLA, 2018). Para isso, ela possibilita a criação de pacotes contendo aplicações e suas dependências. Esses pacotes são chamados de imagens e é a partir delas que o contêiner será criado.

Para criar e construir uma imagem no Docker é preciso utilizar um arquivo conhecido como $Dockerfile$. É nesse arquivo que são definidas regras e instruções como adicionar bibliotecas, arquivos de configuração e parâmetros necessários para realizar a criação da imagem. Cada microsserviço de uma aplicação deverá ter um $Dockerfile$ e, dessa forma, será criado um contêiner para cada um. Com isso, se em uma aplicação existir vários microsserviços, é possível perceber o quão trabalhoso será executar um comando para cada $Dockerfile$ existente a fim de criar todos os contêineres.

Dado esse trabalho, foi criado a ferramenta Docker Compose que nada mais é que um "orquestrador de contêineres". O Docker compose é um arquivo usado para definir

[12]https://www.docker.com/
[13]https://docs.docker.com/samples/library/golang/
[14]https://golang.org/

como será o ambiente da aplicação, configurando todos os microsserviços e bancos de dados. Com ele, basta apenas um comando para que seja criado e executado os múltiplos contêineres que a aplicação possa conter (DIEDRICH, 2015).

MySQL

O Mysql é um SGBD relacional de código aberto pertencente a Oracle[15] e que utiliza a linguagem SQL como interface. Ele possui interface amigável e de fácil utilização, além de ser utilizado em vários sistemas operacionais, o que aumentou sua popularidade pelo mundo todo, sendo utilizado por instituições altamente reconhecidas como NASA[16], HP[17], Bradesco[18] e Sony[19] (OFICINA DA NET, 2010).

É uma boa opção tanto para pequenas aplicações por ser de código aberto e livre, quanto para grandes empresas, que possuem a habilidade de programação para editar o SGBD conforme suas necessidades.

Angular

Angular[20] é uma plataforma de código aberto, desenvolvida pela Google, utilizada no desenvolvimento de interfaces gráficas, ou seja, o *frontend* de aplicações. AFONSO (2018) menciona que o Angular possui elementos básicos como: "componentes, '*templates*, diretivas, roteamento, módulos, serviços, injeção de dependências e ferramentas de infraestrutura que automatizam tarefas", possilitando dessa forma a construção de *frontend*.

Além do HTML e CSS, essa plataforma utiliza o TypeScript[21] que é uma linguagem de programação criada pela Microsoft[22]. O TypeScript é um superconjunto do JavaScript, oferecendo funcionalidades que nativamente não são disponíveis como, por exemplo, recursos para o uso da programação orientada a objetos, tais com classes, interfaces e a tipagem de objetos (opcional).

[15]https://www.oracle.com/br/MySQL/
[16]https://www.nasa.gov/
[17]https://www8.hp.com/br/pt/home.html
[18]https://banco.bradesco/html/classic/index.shtm
[19]https://www.sony.com.br/
[20]https://angular.io/
[21]https://www.typescriptlang.org/
[22]https://www.microsoft.com/en-us/download/details.aspx?id=55258

3 Trabalhos relacionados

3.1 Sistemas de doação de medicamentos

No dia 23/11/2018 foi realizada uma busca na web a fim de obter outros sistemas que ofereçam funcionalidades ao que foi desenvolvido neste trabalho. As subseções a seguir descreve os principais sistemas encontrados com mais detalhes.

3.1.1 Farmácia do bem

Um dos trabalhos relacionados encontrado foi o sistema chamado 'farmácia do bem' de Sete Lagoas (MG). Esse sistema está em fase de lançamento e possui como objetivo "recolher medicamentos que não são mais usados e que estão dentro do prazo de validade e doar gratuitamente para quem precisa desse medicamento"(SIMI, 2017). É possível notar que o objetivo geral do sistema 'farmácia do bem' se iguala ao objetivo do sistema 'farmácia solidária' desenvolvido neste trabalho.

O sistema 'farmácia do bem' possui as seguintes características (Equipe desenvolvedora, s.d.):

- A população, médicos parceiros e indústria farmacêutica doam medicamentos;
- A população, através da plataforma solicita os medicamentos e com a receita médica buscam os medicamentos em sua estrutura física;
- Medicamentos vencidos são encaminhados para o descarte correto. O descarte correto dos medicamentos é feito usando uma rede de transportes e logística e incineradora;
- Utiliza da logísticas de estabelecimentos parceiros para arrecadar medicamentos na hora da solicitação de produtos destes estabelecimentos;

A diferença visível nessa descrição é que o sistema desenvolvido neste trabalho não trabalhará com medicamentos vencidos, sendo somente um ponto de informação para

aqueles que desejem saber onde fazer o descarte correto, visto que, "em Juiz de Fora existe uma lei municipal (Lei 13.442/2016) que responsabiliza as farmácias e as drogarias no descarte adequado dos medicamentos vencidos", de acordo com Marcelo Silvério (Portal UFJF, s.d.).

Outra particularidade do sistema relacionado 'farmácia do bem' é que além dos pontos de apoio para a coleta de medicamentos, eles possuem uma parceria com restaurantes onde o cliente aproveita o *motoboy* da entrega e faz a doação de um medicamento. Já no presente trabalho, o serviço para buscar o medicamento a ser doado em local escolhido pelo doador, será oferecido por meio de uma parceria entre o departamento de Farmacologia da UFJF e a Prefeitura Municipal de Juiz de Fora.

Ressalta-se, entretanto, que esse sistema ainda não está funcionando. No entanto, havia um sistema com o mesmo nome cuja área de funcionamento era em São Paulo (SP), mas que não está mais disponível e não se sabe se é da mesma empresa. Não se encontrou também nenhuma informação a respeito das tecnologias utilizadas para desenvolvimento e nem se o código fonte do sistema será disponibilizado para a comunidade.

3.1.2 SIRUM

Outro sistema relacionado encontrado é o chamado SIRUM. Esse sistema nasceu na Universidade de Standford e funciona em todo os Estados Unidos. Segundo informações de seu site oficial (SIRUM, s.d.), medicamentos não vencidos são coletados de fabricantes, atacadistas, farmácias e unidades de saúde, e levados para clínicas e farmácias onde são distribuídos para pacientes de baixa renda.

O SIRUM funciona como um canal de comunicação entre doadores e beneficiários. Apesar da essência ser a mesma para com o trabalho aqui em desenvolvimento, existe grandes diferenças: no SIRUM, os doadores de medicamentos são organizações como farmácias, atacadistas e fabricantes e os beneficiários são clínicas, e não indivíduos, como no caso do 'farmácia solidária'.

Além disso, a principal diferença fica a cargo de que a plataforma da SIRUM lida com toda a logística entre os doadores e receptores. A logística é descrita da seguinte forma: "As instalações dos doadores usam fax ou um sistema online para fazer *upload*

de seus excedentes, e as clínicas destinatárias enviam uma lista de medicamentos para criar um formulário de seus medicamentos comumente prescritos. Os doadores escolhem entre os destinatários correspondentes, analisando fatores como proximidade geográfica, porcentagem de correspondências e reconhecimento de nomes"(SIRUM, s.d.).

3.1.3 Doar Med

O aplicativo para smartphones, Doar Med, é gratuito e tem como objetivo "conectar pessoas ou empresas que possuem medicamentos a serem doados com pessoas que precisam destes medicamentos". Ele funciona da seguinte forma: o doador cadastra o medicamento no aplicativo e quando esse medicamento for solicitado por outra pessoa, o aplicativo dispara e-mails para que o doador e o solicitante possam entrar em contato e concluir a doação.

A grande diferença para com o sistema desenvolvido neste trabalho é que no Doar Med, os medicamentos a serem doados não passam por uma triagem, uma vez que a doação é feita de forma direta, entre o doador e o solicitante. Além disso, eles não possuem um ponto de apoio, assim a entrega/recebimento desses medicamentos são de responsabilidade do usuário.

A tabela apresentada na Figura 3.2 faz uma comparação dos sistemas similares encontrados para com o sistema desenvolvido 'farmácia solidária'.

3.2 Sistemas baseados em microsserviços

A fim de encontrar trabalhos relacionados a este, foi realizado uma busca na literatura[23] utilizando como strings de busca os termos *"microservice*"AND architecture* nas bases ACM e IEEE. A busca retornou 279 resultados, sendo 75 provenientes da ACM e 204 da IEEE. Muitos dos trabalhos retornados referiam-se a infraestrutura de microsserviços e, por isso, foram desconsiderados. Outros trabalhos descreviam o processo utilizado para transformar um sistema monolítico em microsserviços. De fato, conforme relatado por Zhou et al. (ZHOU et al., 2018), ainda que haja várias empresas que utilizam arquite-

[23]Busca realizada no dia 23/11/2018.

	SIRUM	Farmácia do bem	Doar Med	Farmácia solidária
Região	EUA	Sete Lagoas (MG) - Brasil	Brasil	Juiz de Fora (MG) - Brasil
Público alvo	Farmácias, atacadistas e fabricantes	População, médicos parceiros e indústria farmacêutica	População e empresas	População
Doação	Doadores escolhem para qual instituição conceder seus medicamentos	Doação de medicamentos são recebidos em um ponto de apoio que os disponibiliza para a população	Medicamentos para serem doados são cadastrador no aplicativo pelo doador	Doação de medicamentos são recebidos em um ponto de apoio que os disponibiliza para a população
Comunicação	Faz toda a logística entre doadores e beneficiários	Utiliza do site e do ponto de apoio para a ligação entre doadores e beneficiários	Utiliza do aplicativo e e-mails para conectar doadores e beneficiários	Utiliza do site e do ponto de apoio para a ligação entre doadores e beneficiários
Solicitação	Clinicas enviam uma lista de medicamentos comumente prescritos para o sistema	Indivíduos fazem a solicitação do medicamento pelo site e vão até o ponto de apoio para busca-lo	Indivíduos fazem a solicitação do medicamento pelo aplicativo, que envia e-mail para o doador com os dados do beneficiário e vice-versa.	Indivíduos fazem a solicitação do medicamento pelo site e vão até o ponto de apoio para busca-lo
Descarte de medicamentos vencidos	Não faz	Faz o descarte usando a própria rede de transporte e incineradora	Não faz	Não faz
Triagem	Não faz	Faz inspeção dos medicamentos doados antes de liberar para a população	Não faz	Faz inspeção dos medicamentos doados antes de liberar para a população

Figura 3.1: Comparação entre os sistemas de doação de medicamentos

tura orientada a microsserviços no desenvolvimento de seus produtos, há poucos artigos que relatem as características desses sistemas. Zhou et al. (ZHOU et al., 2018) relata ainda que muitos dos projetos encontrados que se dizem baseados em microsserviços, na verdade empregam uma arquitetura monolítica e apenas utilizam alguma técnica relacionada a microsserviços como, por exemplo, a implantação com Docker. Entretanto, alguns trabalhos retornados foram relevantes. Esses estudos são detalhados a seguir.

ZHOU et al. (2018) desenvolveram um sistema para emissão de bilhetes ferroviários chamado *TrainTicket* que contém 24 microsserviços relacionados à lógica de negócio. Sua arquitetura consiste de cinco camadas, sendo que na camada superior estão os microsserviços que dependem da camada inferior; a camada mais baixa contém os mi-

crosserviços que não dependem de nenhum outro. Para realizar a comunicação com o *frontend*, *TrainTicket* utiliza a camada API Gateway.

ADERALDO et al. (2017) e ZHOU et al. (2018) realizaram uma busca por projetos de sistemas baseados em microsserviços e de códigos abertos. Os primeiros selecionaram os projetos *Acme Air*, *Spring Cloud Demo Apps*, *Socks Shop* e *MusicStore*. Esse projetos também foram selecionados por ZHOU et al. (2018) que incluíram ainda *Bifrost Microservices Sample Application*, *Staffjoy* e *NServiceBus*.

Sistema	**Domínio**	**Comunicação**	**Virtualização**	**Linguagens**	**Esforço**	**#MS**
TrainTicket	Venda de bilhetes ferroviários	API Gateway	Docker	Java/Spring Boot Node.js	61.136 KLOC	24
AcmeAir	Loja online	Não encontrado	Docker	Java EE Node.js	Não encontrado	4
Spring Cloud Demos	Recomendação de filmes Processamento gráfico	Não encontrado	Docker	Java	Não encontrado	11 8
Socks Shop	Loja online	Não encontrado	Docker	Java / Spring Boot Go Node.js	Não encontrado	8
Music Store	Loja online	Não encontrado	Docker	.NET	Não encontrado	6

Figura 3.2: Comparação entre os sistemas baseados em microsserviços

4 Sistema farmácia solidária

4.1 Visão geral

O sistema farmácia solidária é uma aplicação web com o objetivo de facilitar o repasse de medicamentos que não são mais usados, contribuindo para apoiar o recebimento por pessoas que os necessitam de forma gratuita. Ele foi desenvolvido seguindo o processo unificado ágil que segue as fases do processo unificado, mas sem o rigor de produzir todos os artefatos.

4.2 Características do produto

O software deverá possibilitar que o Departamento de Farmacologia da UFJF ou outro setor público que se interesse pelo tema, como a Prefeitura Municipal de Juiz de Fora e/ou de outras cidades, trabalhe a arrecadação e a oferta de medicamentos de forma computadorizada. Ele permitirá que a pessoa possuidora do medicamento em desuso obtenha informações para fazer a doação e, também, que as pessoas que necessitem possam fazer a solicitação desse medicamento.

Através de um formulário, a pessoa que quiser doar algum tipo de medicamento irá dizer se quer levar o medicamento em algum dos pontos de apoio ou se deseja que ele seja recolhido por voluntários no endereço cadastrado no sistema.

O Departamento de Farmacologia, ou outro setor público de interesse, receberá os medicamentos doados e realizará a inspeção visual do mesmo conforme legislação vigente. Tal etapa visa avaliar a integridade física das embalagens, do produto contido, bem como o prazo de validade do medicamento. Aqueles que forem aprovados na inspeção ficarão disponíveis no sistema para solicitações.

O usuário, ao fazer uma solicitação, deve escolher um medicamento dentre aqueles na lista de disponíveis. Ao ser solicitado, o medicamento ficará reservado por até 3 dias úteis e para recebê-lo, será preciso ir até a sede da Farmácia Solidária e apresentar a

prescrição médica atestando a necessidade do uso.

4.3 Levantamento de requisitos

Para a primeira etapa do desenvolvimento do software, foram levantados os requisitos do sistema. Existem diversas técnicas para a pesquisa dos requisitos como entrevistas, *brainstorming*, questionários e prototipação. As técnicas utilizadas foram a entrevista e a prototipação, que foram realizadas com a professora Pâmela Souza Almeida Silva do Departamento de Farmacologia da UFJF. Diversas seções foram marcadas e sempre feitas de modo a conseguir o máximo de informações do cliente, no caso, a professora Pâmela.

Através das entrevistas e protótipos, foi possível especificar os requisitos do sistema, e com isso, desenvolver o documento de requisitos e os casos de uso do sistema. As subseções a seguir apresentam um resumo dos requisitos levantados. O documento completo pode ser visto no Anexo A.

4.3.1 Requisitos funcionais

1. Manter Controle de Estoque
2. Solicitar medicamento
3. Doar medicamento
4. Realizar cadastro de usuário
5. Manter informações
6. Manter ponto de apoio
7. Envio de mensagem
8. Efetuar login

4.3.2 Requisitos não funcionais

1. Mensagens de erro ou sucesso devem ser mostradas até 30 segundos após a interação com o usuário

2. O tempo de espera na busca de medicamentos no sistema não deve exceder 30 segundos, tempo esse que começa ao clicar no botão “pesquisar” até a visualização da listagem

3. O sistema deverá suportar processamento multiusuário, ou seja, vários usuários poderão utilizar o sistema simultaneamente

4. O sistema deverá rodar em qualquer navegador web

5. O sistema deverá fornecer uma interface amigável e prática

6. O site deverá estar disponível para o usuário por 24 horas, durante os 7 dias da semana, com não mais que 2% do tempo com o sistema fora do ar

7. Como o sistema será via WEB, ele deverá ser o mais seguro possível

4.3.3 Diagrama de caso de uso

O diagrama de caso de uso foi elaborado para representar como as funcionalidades do sistema se relacionam entre si e com os usuários (atores).

Conforme pode ser visto na Figura 4.1, foram identificados três atores: usuário, cliente e administrador; e 14 casos de uso, que são as funcionalidades que podem ser executadas por esses atores. Os casos de uso identificados foram ‘Doar medicamentos’, ‘Solicitar medicamentos’, ‘Levar medicamentos ao ponto e apoio’, ‘Requisitar o recolhimento do medicamento’, ‘Pesquisar medicamento’, ‘Verificar estoque’, ‘Enviar sugestões ou críticas’, ‘Gerenciar usuário’, ‘Inserir tipo de usuário’, ‘Efetuar login’, ‘Gerenciar notícias’, ‘Gerenciar medicamentos’, ‘Gerenciar doações’ e ‘Gerenciar solicitações’.

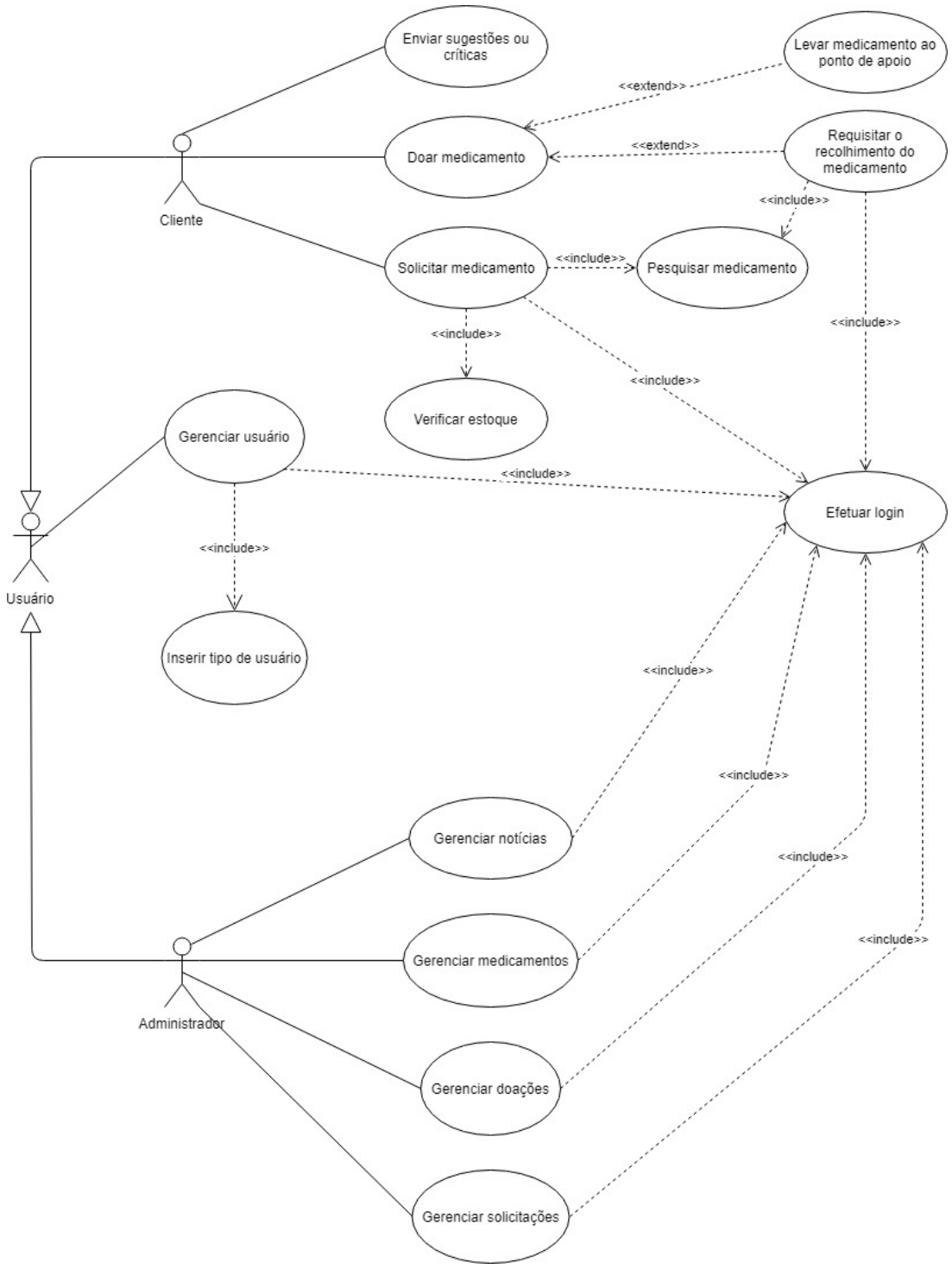


Figura 4.1: Diagrama de caso de uso

A seguir, será apresentado as descrições e o fluxo principal dos casos de uso desenvolvidos neste TCC, a descrição completa de todos os casos de uso pode ser vista no Anexo B.

UC01 – Solicitar medicamentos

- Objetivo:

  Permitir que o usuário possa solicitar um medicamento, dentre a lista de disponíveis;

- Ator:

  Usuário cliente;

- Condição de entrada:

  O ator deve entrar no menu “Solicitar medicamentos” e escolher dentre os submenus disponíveis (uso humano ou uso veterinário).

- Fluxo Principal:

  1. O usuário pesquisa pelo medicamento (nome genérico ou comercial) no filtro;

     Listagem

  2. O sistema lista os medicamentos encontrados pela pesquisa;
  3. O usuário clica no botão “Outras informações”;
  4. O sistema abre uma modal contendo várias informações do medicamento e um campo “quantidade” para preenchimento do usuário (caso a forma farmacêutica do medicamento seja comprimidos);
  5. O usuário preenche o campo “quantidade”, caso esteja habilitado;

     Confirmação de envio

  6. O usuário clica em “solicitar”;

     Validação de dados

  7. O sistema valida o login do usuário e a quantidade informada;
  8. O sistema salva a solicitação;
  9. O sistema abrirá uma tela contendo informações sobre o local e quais dias, o medicamento estará disponível para o usuário buscar.

UC11 – Cadastrar medicamento no estoque

- Objetivo:

  Permite o administrador cadastre medicamentos disponíveis no estoque;

- Ator:

  Usuário administrador;

- Condição de entrada:

  O ator deve estar logado e clicar no menu “estoque”.

- Fluxo Principal:

  1. O sistema abre a tela de pesquisar medicamentos;
  2. O administrador clica no botão “cadastrar medicamento”;
  3. O sistema abrirá uma modal com um formulário para preenchimento;

     Confirmação de envio
  4. O usuário clica em salvar;

     Validação de campos
  5. O sistema valida os campos;
  6. O sistema salva o medicamento;
  7. O sistema exibe uma mensagem de sucesso;
  8. O medicamento passará a ser exibido na listagem da tela do UC01 – Solicitar medicamento e no UC11 - Cadastrar medicamento.

UC12 – Remover medicamento do estoque

- Objetivo:

  Permite o administrador remova medicamentos do estoque;

- Ator:

  Usuário administrador;

- Condição de entrada:

  O ator deve estar logado e clicar no menu “estoque”.

- Fluxo Principal:

  1. O sistema abre a tela de pesquisar medicamentos;
  2. O ator pesquisa pelo medicamento (nome genérico ou comercial) ou pelo lote no filtro;

     Listagem
  3. O sistema lista os medicamentos encontrados pela pesquisa;
  4. O ator clica no botão “Remover”;
  5. O sistema abre uma modal contendo várias informações do medicamento e um campo “motivo” para preenchimento;
  6. O usuário preenche o campo “motivo”;

     Confirmação de envio
  7. O usuário clica em “remover”;

     Validação de campos
  8. O sistema valida os campos;
  9. O sistema faz uma exclusão lógica do medicamento (ou seja, o medicamento ainda estará no banco de dados, porém com status de “removido”);
  10. O sistema exibe uma mensagem de sucesso.

4.3.4 Protótipos

Para facilitar o entendimento dos requisitos e das funcionalidades do sistema, foi realizado a prototipação das telas. A prototipação é uma atividade para minimizar os riscos, visto que é apresentado uma solução apropriada para o negócio do cliente, além de identificar melhorias, erros e omissões dos requisitos.

Os protótipos foram feitos no software Balsamiq Mockups[24]. Como não apresentam interações de tela, eles são de baixa precisão; porém através deles, as regras de

[24]https://balsamiq.com/

negócio puderam ser validadas de forma rápida e eficiente. Nas subseções a seguir são apresentadas alguns dos protótipos de tela que foram criados para o projeto. A lista completa pode ser vista no Anexo A, pág. 7.

4.3.5 Tela para pesquisar medicamentos

Na tela de pesquisa apresentada na Figura 4.2, estarão apresentados os medicamentos com *status* de disponível, ou seja, os medicamentos ali listados poderão ser solicitados por usuários que precisem dos mesmos. O acesso à essa tela não necessita de login, podendo ser visualizada por qualquer pessoa que acessar o sistema.

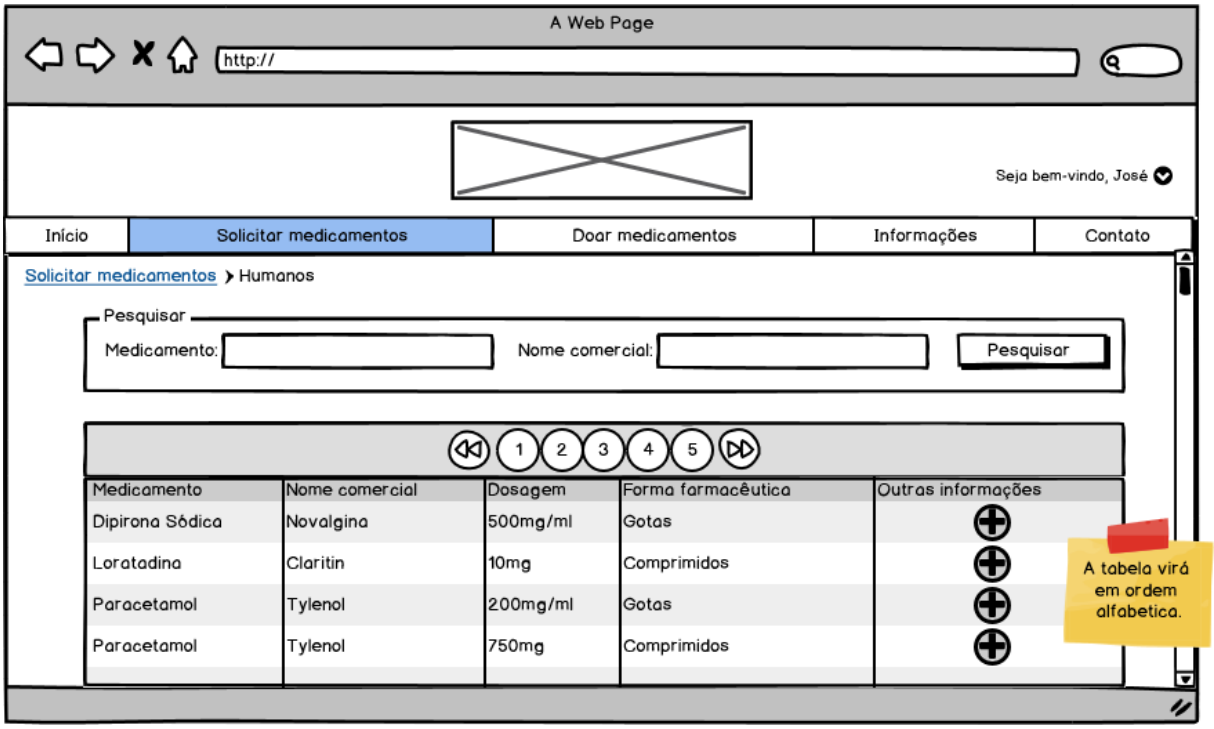


Figura 4.2: Tela para pesquisar medicamentos

4.3.6 Telas para solicitar medicamento

Ao clicar no botão solicitar medicamento presente na tela anterior, o sistema abrirá uma tela *modal*, conforme apresentado na Figura 4.3, contendo várias informações do medicamento e um campo 'quantidade' para preenchimento. Para solicitar um medicamento, o usuário deverá estar logado no sistema.

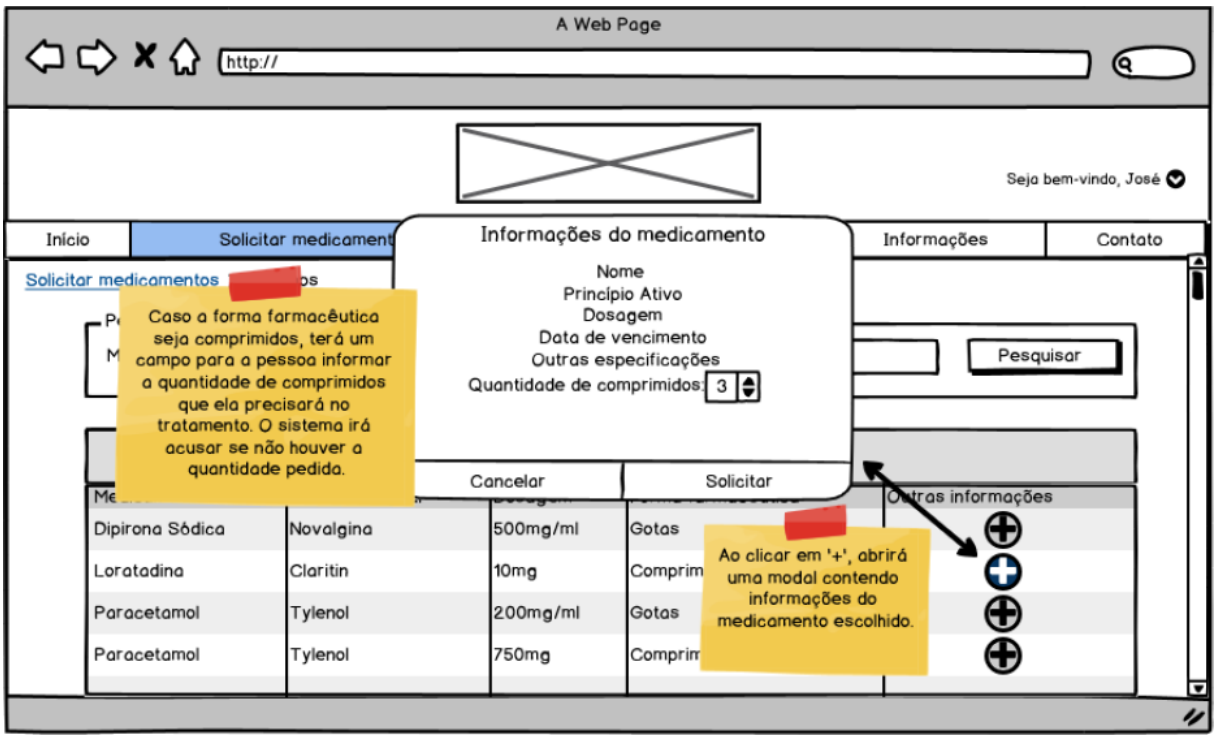


Figura 4.3: Tela para informar a quantidade de medicamento

Quando o sistema validar a requisição, uma mensagem aparecerá na tela, conforme apresentado na Figura 4.4, contendo informações sobre o local e quais dias o medicamento estará disponível para o usuário buscar.

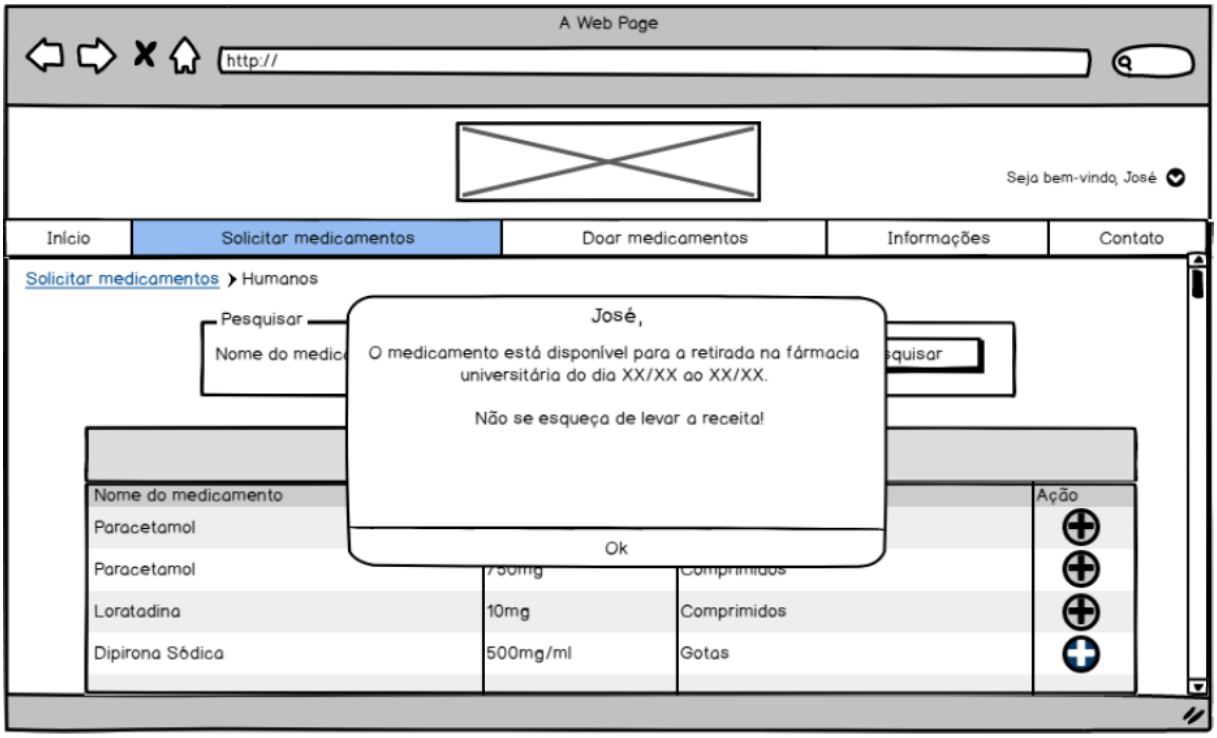


Figura 4.4: Mensagem sucesso

4.3.7 Tela para manter estoque

O controle de estoque será acessada somente pelo administrador do sistema. A tela de estoque, apresentada na Figura 4.5, exibirá uma listagem com todos os medicamentos cadastrados no sistema e seu *status*, sendo possível também, visualizar detalhes de especificação como o nome do medicamento, lote, dosagem, forma farmacêutica, quantidade de comprimidos, estado da embalagem, o ponto de apoio que este medicamento está estocado e a data de vencimento.

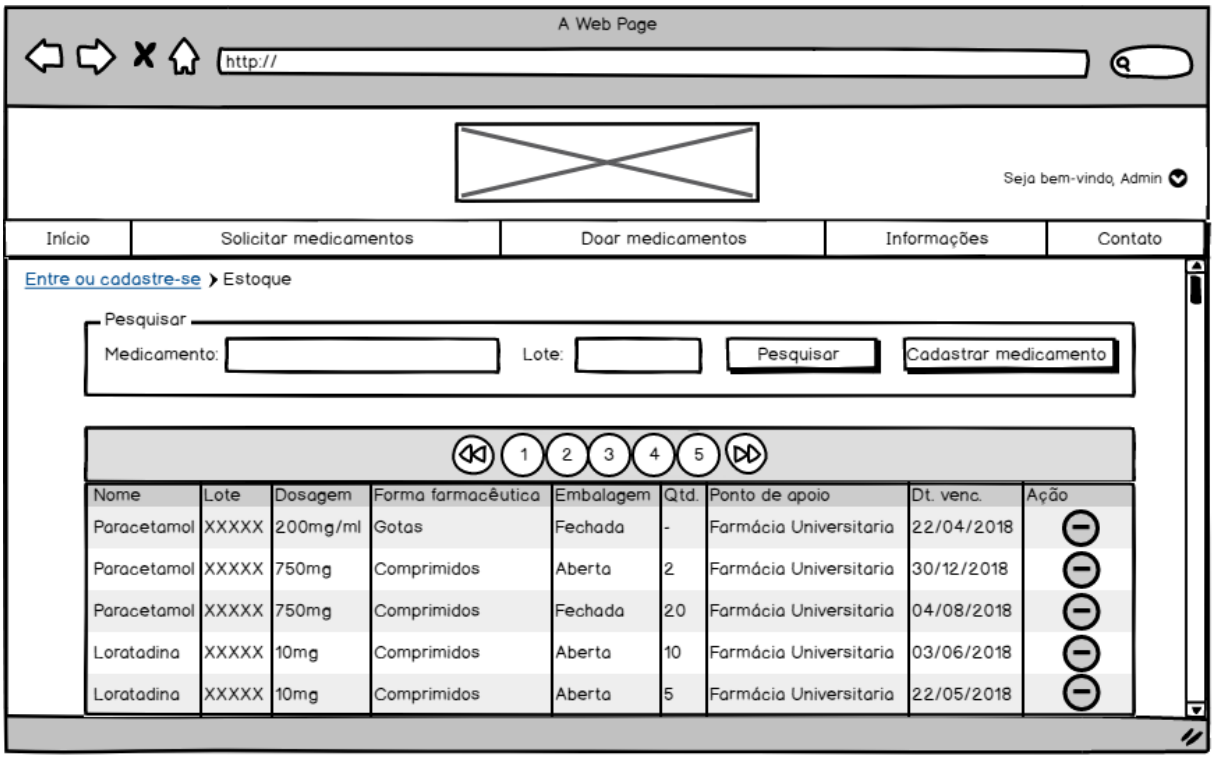


Figura 4.5: Tela listando todos os medicamentos cadastrados

Para dar baixa em algum medicamento, que pode acontecer, por exemplo, quando ele estiver vencido ou for repassado a uma pessoa, o administrador deverá clicar no botão remover e informar o motivo na tela *modal* que se abrirá, conforme representado na Figura 4.6.

A tela de listagem também deve possuir um botão 'Cadastrar medicamento' que, ao clicá-lo, deverá abrir uma tela *modal* exibindo um formulário contendo diversos campos para preenchimento, conforme pode ser vista na Figura 4.7.

Os documentos de requisitos, casos de uso e protótipos podem ser encontrados em anexo.

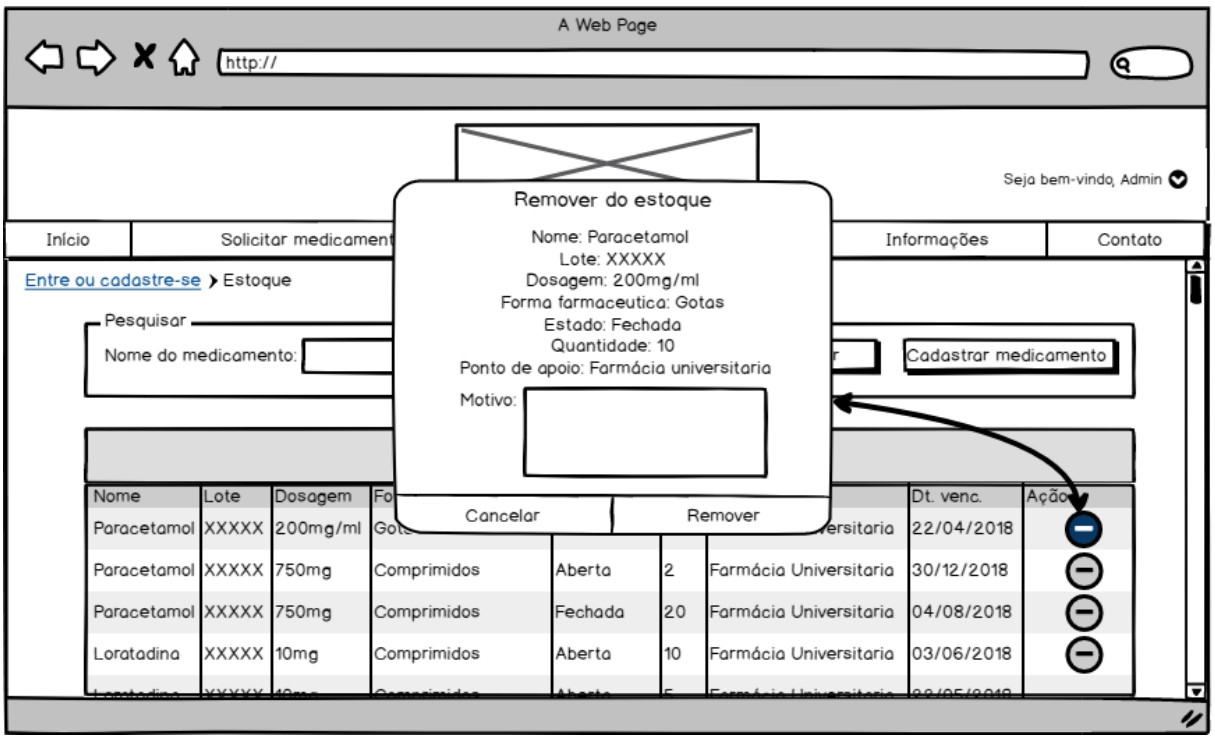


Figura 4.6: Tela para dar baixa no estoque

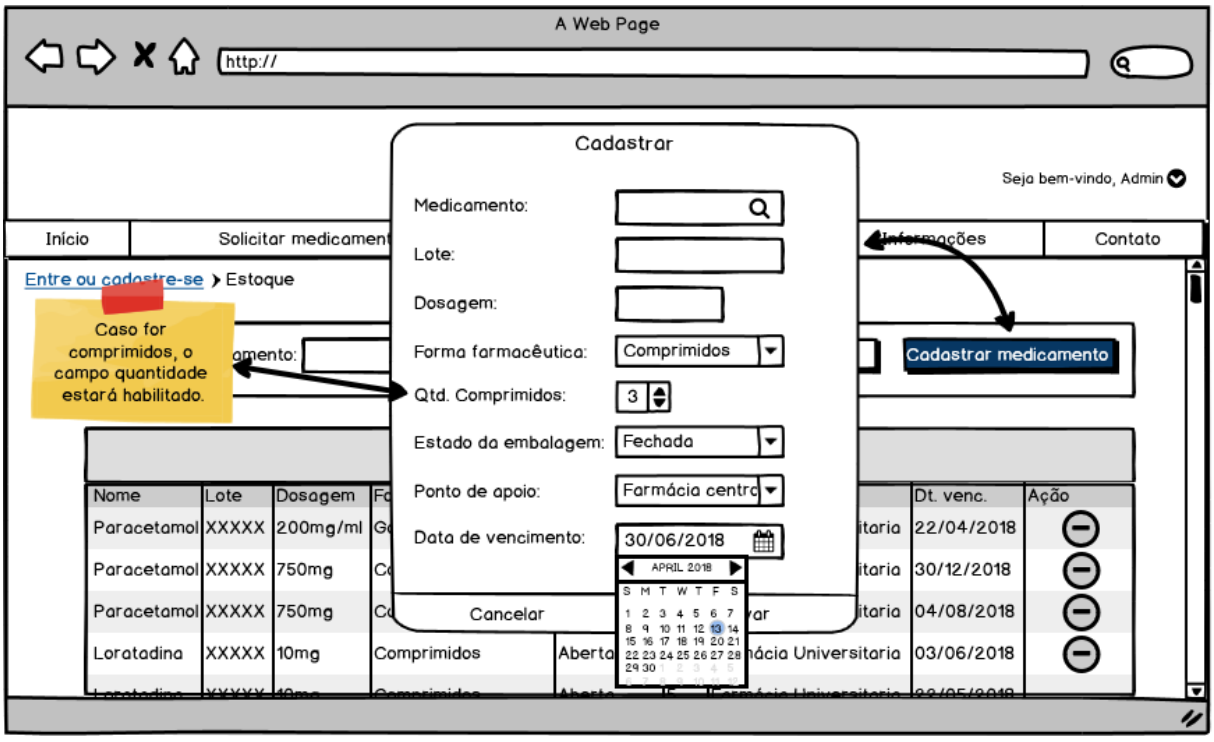


Figura 4.7: Tela para cadastrar medicamento

4.4 Arquitetura do sistema ‘farmácia solidária’

A arquitetura escolhida para o projeto, em uma perspectiva mais abstrata, foi a de microsserviços, visto que ela possui a capacidade de reuso, além de acelerar o desenvolvimento de novas soluções de software. Além disso, pretende-se expandir o desenvolvimento de software tanto para outros setores da UFJF quanto para a comunidade em geral, sendo que os microsserviços desenvolvidos para esse trabalho poderão ser reutilizados para tal fim.

Estabelecida a arquitetura mais abstrata, a próxima etapa realizado foi a de projetar o sistema, dividindo a aplicação em microsserviços independentes. Ao chegar nessa etapa, percebeu-se que não há uma definição de quão micro um microsserviço deve ser, o que levou a várias discussões de como essa divisão poderia ser feita. Optou-se por seguir a abordagem definida por LEWIS E FOWLER (2014), que leva em conta as funcionalidades do código, dividindo-os de forma mais significativa possível. Assim, os microsserviços identificados na aplicação ‘farmácia solidaria’ e uma breve descrição sobre a responsabilidade de cada um são listados a seguir:

- Controle de acesso:

  Responsável pela autenticação e permissão de acesso.

- Medicamento:

  Responsável por conter serviços de cadastrar, buscar, alterar, excluir, reservar e doar um medicamento.

  Consome dados do microsserviço de controle de acesso.

- Notícia:

  Encarregado de cadastrar, buscar e alterar uma notícia/informação.

  Consome dados do microsserviço controle de acesso.

- SAC:

  Encarregado de enviar o e-mail de contato do usuário para o administrador do sistema.

- Ponto de apoio:

  Responsável por cadastrar, alterar, buscar e excluir um ponto de apoio.

  Também consome dados do microsserviço controle de acesso.

4.4.1 Integração dos dados de microsserviços

Uma das diretrizes da arquitetura trabalhada diz que para garantir um baixo acoplamento, cada microsserviço deve administrar seus dados, mantendo a integridade e a consistência dos mesmos. Para que isso aconteça, cada microsserviço deverá ter seu próprio banco de dados. Mas como integrar os dados de diferentes microsserviços, visto que cada um possuirá sua base de dados separada? Baseado nisso, foi proposto pela autora três formas de como realizar essa integração.

A primeira abordagem possível seria que cada banco de dados tivesse duas instâncias, uma com permissão para leitura e escrita e outra somente para leitura. Assim, o microsserviço que precisasse consumir dados de outro microsserviço, acessaria diretamente a instância de leitura do banco de dados do segundo microsserviço. A Figura 4.8 representa essa abordagem cuja grande desvantagem ocorre pela duplicação de código, uma vez que uma mesma consulta estaria em diversos microsserviços diferentes.

A segunda possibilidade de integrar esses dados seria elaborar visões das tabelas compartilhadas no banco de dados de cada microsserviço, assim o microsserviço que precisar de dados de outro, conseguirá acessa-los através de sua própria base, conforme apresentado na Figura 4.9. Para essa proposta funcionar, deve-se utilizar somente um tipo de SGBD para que seja possível implementar as visões. Com isso, perde-se umas das vantagens da arquitetura de microsserviço, que é a possibilidade de se escolher a melhor forma de armazenar os dados a partir das necessidades de cada microsserviço.

As duas propostas anteriores mostram microsserviços acessando diretamente a base de dados de outros microsserviços, o que fere uma regra básica da arquitetura, onde se diz que cada microsserviço deve gerenciar seu próprio banco de dados e assim, garantir a integridade de seus dados. Neste projeto, por exemplo, os microsserviços de controle de acesso e o de medicamentos devem se comunicar, visto que um precisa dos dados que o outro possui. Caso o acesso aos dados fosse realizado de forma direta, como citado nas

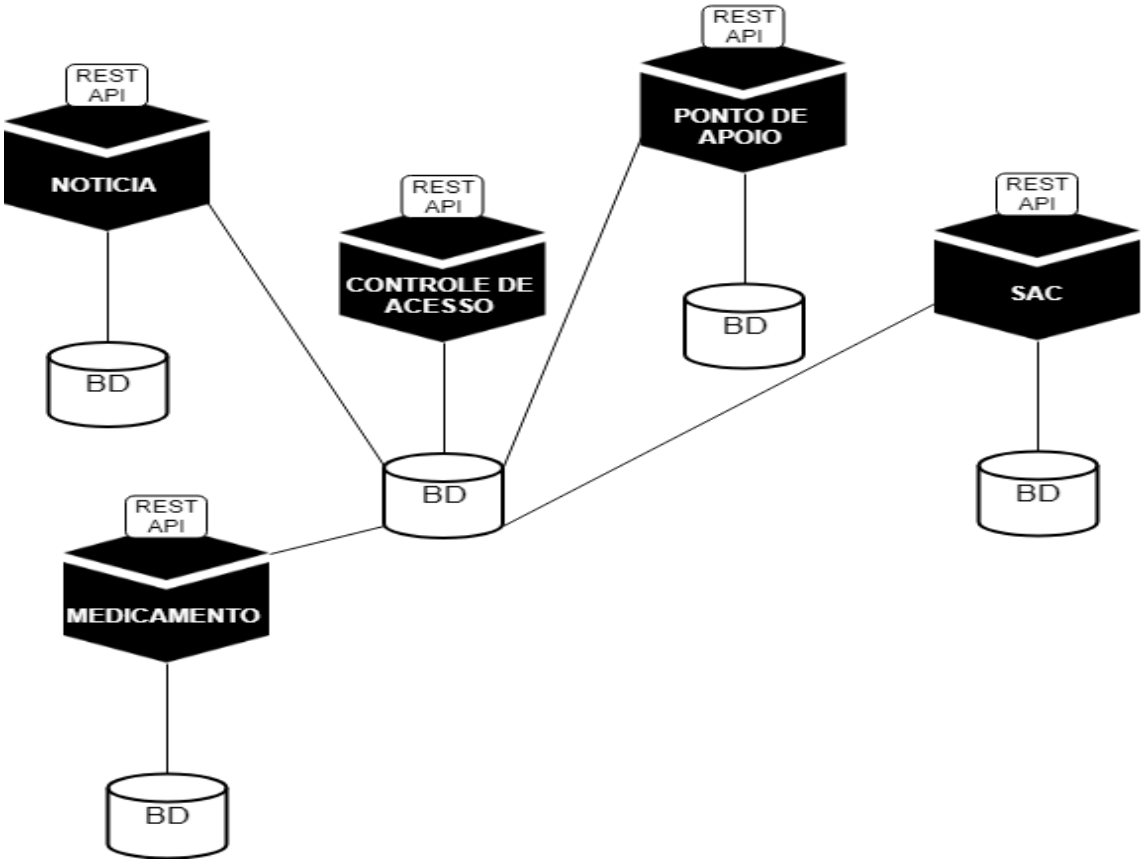


Figura 4.8: Abordagem 1 - Banco de dados com duas instâncias

abordagens 1 e 2, a lógica de programação de autenticação e autorização que está implementado no microsserviço de controle de acesso, deveria ser repetida no microsserviço de medicamento. Ou seja, sabendo que somente o microsserviço de controle de acesso poderá manipular seus dados, seu consumo deve ser feito apenas pelos recursos expostos por ele. Porém se os microsserviços se comunicarem diretamente via REST/HTTP, as especificidades de cada um se tornarão conhecidas, como os endereços onde cada microsserviço funciona. Além de que, seria muito complexo para cada microsserviço manter atualizados todos os endereços que utiliza.

Para resolver esse problema, a comunidade propõe uma terceira abordagem em que há uma camada de abstração entre os microsserviços. Essa camada de abstração é conhecida como *API Gateway*, que funciona como um intermediário na comunicação, não só entre os próprios microsserviços, mas também entre os microsserviços e o *frontend*. O API Gateway pode ser descrito como um único ponto de entrada dos serviços expostos.

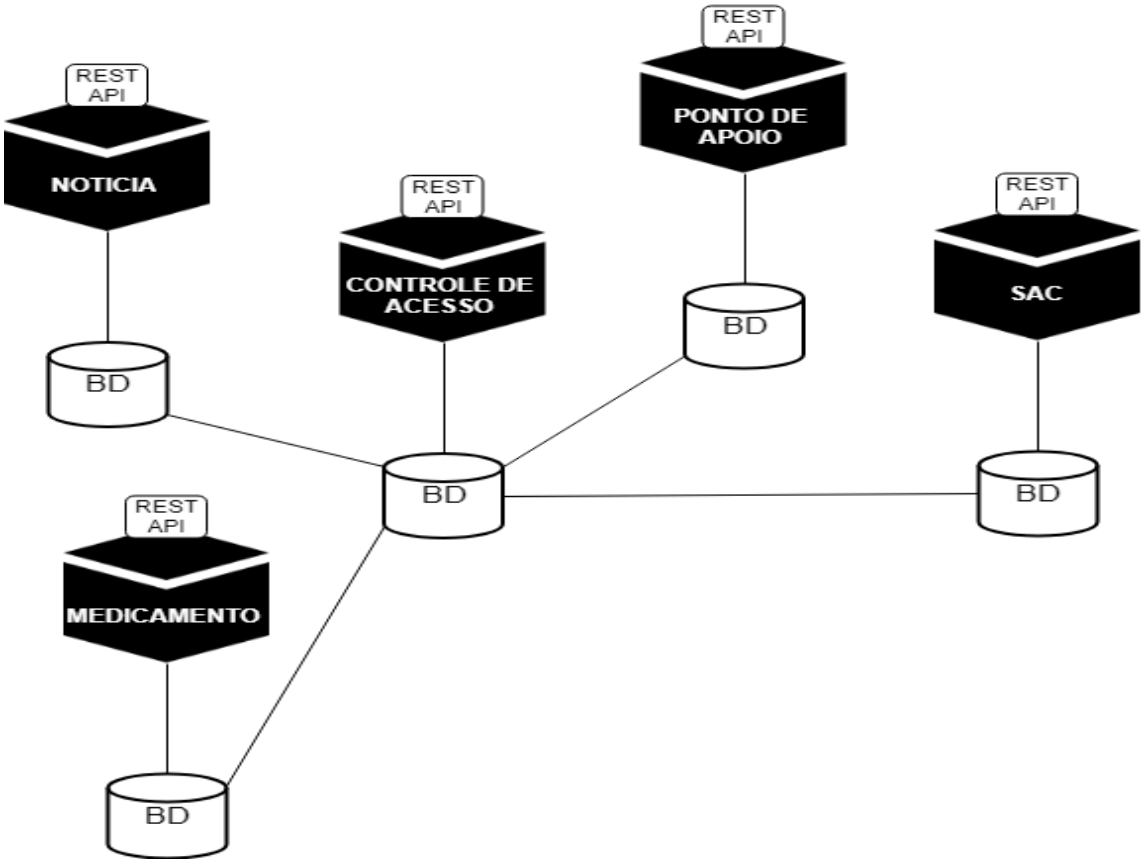


Figura 4.9: Abordagem 2 - Views das tabelas compartilhadas

Para consumir esses serviços, o cliente deverá chamar essa camada, que por sua vez, encaminhará a requisições para os serviços apropriados. Também é de responsabilidade do API Gateway administrar e monitorar o tráfego das transações, controlar a autorização de acesso, o armazenamento de cache, entre outros. Esse cenário está representado na Figura 4.10:

4.4.2 Comunicação

O *backend* do sistema é formado por um conjunto de microsserviços, sendo que cada microsserviço é composto de pequenos serviços que devem ser expostos através de interfaces bem definidas e protocolos independentes para que possam ser acessados. Esses pequenos serviços enviam e recebem dados de seus consumidores, porém conforme destacado por NEWMAN (2015), esses dados devem ser padronizados de tal forma que não reflita os detalhes das representações que o serviço utiliza.

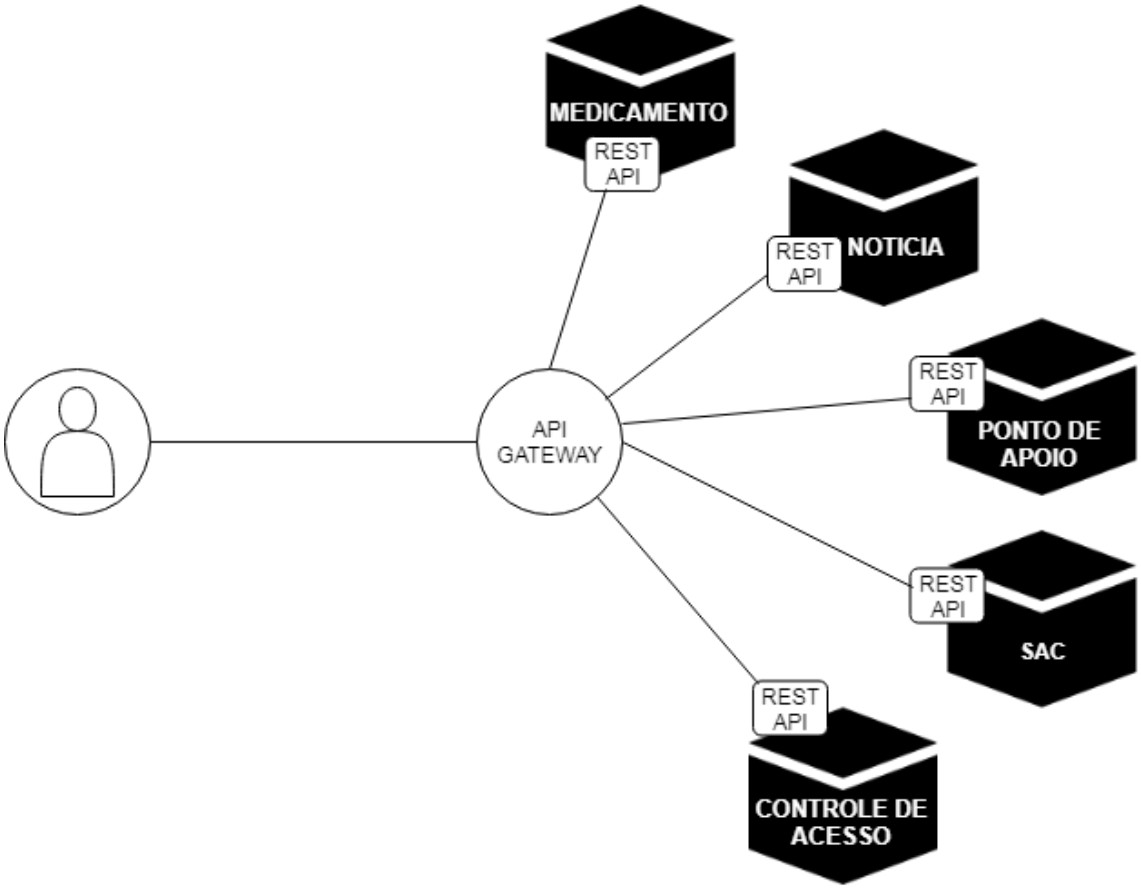


Figura 4.10: Abordagem 3 - Gateway

Apoiado nessa questão, o padrão escolhido para ser utilizado na comunicação foi o REST, visto que ele oferece como elemento principal a abstração do serviço. O padrão REST será garantido na integração do *frontend* com o *backend* por meio da API Gateway, citado na seção anterior. No REST, os métodos GET, POST, PUT e DELETE do HTTP mapeam as operações CRUD (do inglês Create, Read, Update, Delete), que são necessárias em aplicações que gerenciam dados. No microsserviço de medicamento, por exemplo, foi utilizado os métodos GET, POST e PUT como mostrado em Listing 4.1.

```
1 @PostMapping
2 public ResponseEntity<Object> criar(@Valid @RequestBody Medicamento medicamento,
      HttpServletResponse response) {
3     Optional<Medicamento> medicamentoOptional = medicamentoRepository.findById(medicamento.
      getLote());
4     if(medicamentoOptional.isPresent()) {
5         List<Erro> erros = Arrays.asList(new Erro("Lote já cadastrado.", "Lote já cadastrado."));
6         return ResponseEntity.badRequest().body(erros);
```

```
7       }
8       else {
9           medicamento.getEstoque().setMedicamento(medicamento);
10          medicamentoService.salvar(medicamento);
11      }
12      return ResponseEntity.status(HttpStatus.CREATED).body(medicamento);
13  }
14
15  @GetMapping("/{codigo}")
16  public ResponseEntity<Medicamento> buscarPeloCodigo(@PathVariable String codigo) {
17      Medicamento medicamento = new Medicamento();
18      medicamento.setLote(codigo);
19      Example<Medicamento> exampleMed = Example.of(medicamento);
20      Optional<Medicamento> medicamentoOptional = medicamentoRepository.findOne(exampleMed);
21      return medicamentoOptional.isPresent() ? ResponseEntity.ok().body(medicamentoOptional.get())
        : ResponseEntity.notFound().build();
22  }
23
24  @PutMapping("/{lote}")
25  public ResponseEntity<Medicamento> atualizar(@PathVariable String lote, @Valid @RequestBody
        Medicamento medicamento) {
26      Medicamento medicamentoSalvo = medicamentoService.atualizar(lote, medicamento);
27      return ResponseEntity.ok(medicamentoSalvo);
28  }
```

Listing 4.1: Métodos POST, GET e PUT do microsserviço de medicamento

4.5 Implementação

Para desenvolver o sistema 'farmácia solidária', foi montada uma equipe composta por 4 desenvolvedores, composta pela autora deste trabalho e mais três bolsistas e voluntários de Treinamento Profissional. Por esse motivo, o desenvolvimento do sistema foi divido da seguinte maneira:

- Alex: microsserviço de pontos de apoio;
- Bárbara: microsserviços de medicamento e controle de acesso;

- Maxwell: microsserviço de notícia;
- Vinícius: microsserviço de SAC;

Uma das vantagens da arquitetura adotada é a natureza poliglota, possibilitando que cada desenvolvedor escolha qual linguagem de programação usar para implementar seu microsserviço. A partir disso, obteve-se três linguagens diferentes sendo usada no projeto: o microsserviço de SAC foi feito na linguagem .NET, o microsserviços de notícias foi implementado em PHP, e os demais que foram desenvolvidos em Java.

Para o andamento do projeto, foram realizados encontros semanais com todos os desenvolvedores. O objetivo desses encontros eram disseminar o conhecimento, esclarecer dúvidas, discutir sobre microsserviços, além de transparecer o real estado do desenvolvimento. É importante mencionar que cada desenvolvedor ficou responsável por escolher a melhor forma de desenvolver para caracterizar ainda mais o estilo de desenvolvimento de uma aplicação baseada em microsserviços. O presente trabalho seguirá apresentando o desenvolvimento dos microsserviços de medicamento e controle de acesso, que foram os desenvolvidos pela autora.

4.5.1 Ferramentas e tecnologias utilizadas

Sistema Gerenciador de Banco de Dados:

o **MySQL** foi utilizado por ele ser um banco de dados de código aberto e o mais popular do mundo. A Figura 4.11 mostra o modelo de banco de dados para o microsserviço de medicamento:

Tecnologias para o desenvolvimento *backend*:

1. **Linguagem de programação:** a linguagem **Java** foi escolhido pois além de possuir uma vasta plataforma de ferramentas gratuitas, a autora possui uma experiencia de três anos programando softwares nessa linguagem. Foi utilizado a versão 8 do java para o desenvolvimento do projeto.

   Framework: O **Spring Boot** foi escolhido por estar evoluindo e facilitando o desenvolvimento de microsserviços em Java, além de ajudar na configuração do

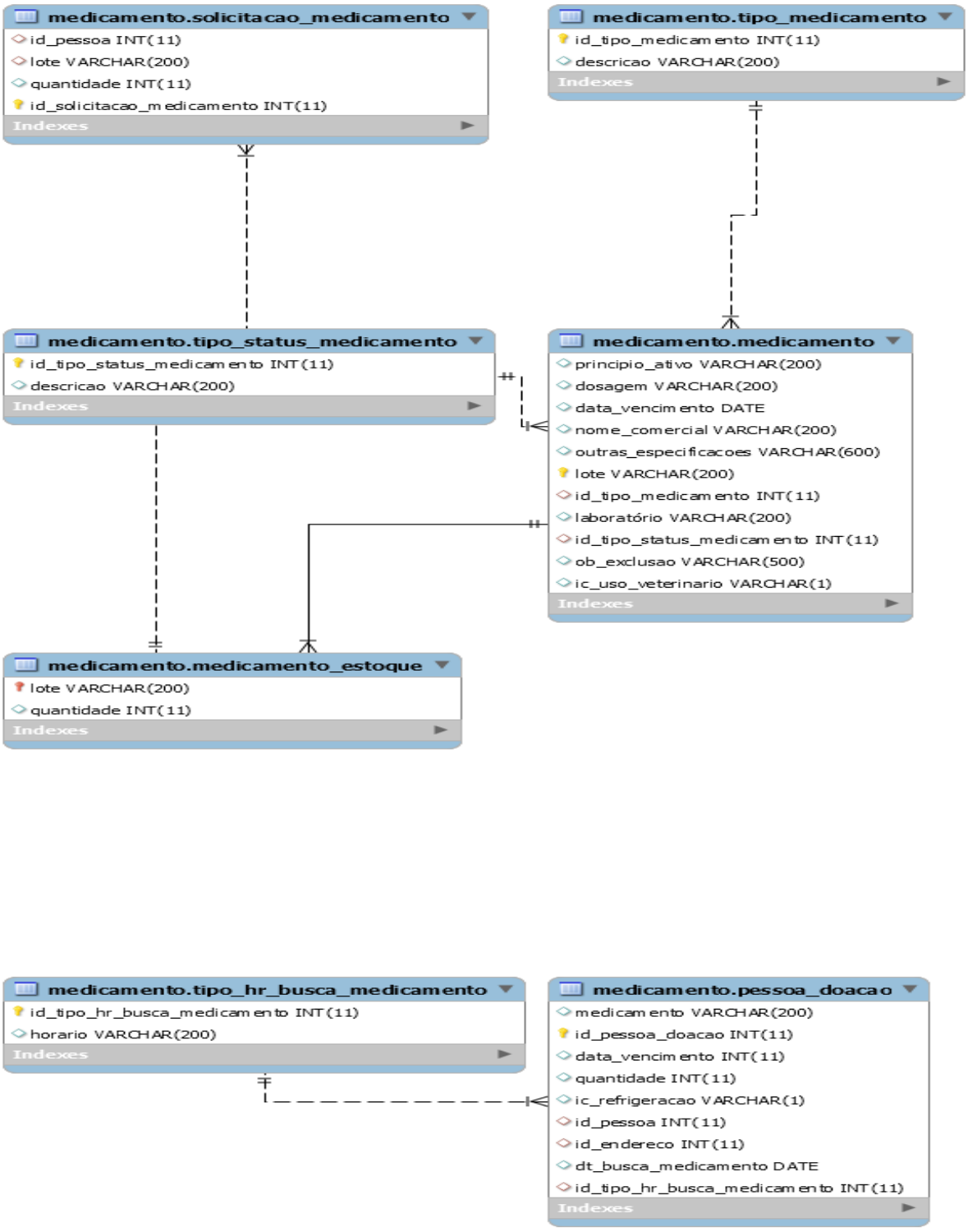


Figura 4.11: Modelo de banco de dados

projeto, importando automaticamente todas as dependências necessárias.

A seguir, são listadas as principais tecnologias de microsserviços consolidadas no mercado, como por exemplo, a utilização de soluções que a própria Netflix desenvolveu para o gerenciamento de seus microsserviços.

Gerenciamento de configurações:

a ferramenta **Spring Cloud Config** foi escolhida para centralizar a configuração de todos os microsserviços em um mesmo lugar pela facilidade de integração de todas as configurações dos microsserviços desenvolvidos com Spring Boot.

Para este trabalho, foi criado um projeto chamado 'Config Server' do Spring Cloud Config que foi apontado para um repositório git que contém todas as configurações de acesso ao banco de dados e qual porta utilizada pelos microsserviços de medicamento e controle de acesso. No momento que os microsserviços de medicamento e controle de acesso forem inicializados, eles consultarão suas propriedades fazendo uma requisição ao serviço do 'Config Server'. A Figura 4.12 representa esse funcionamento.

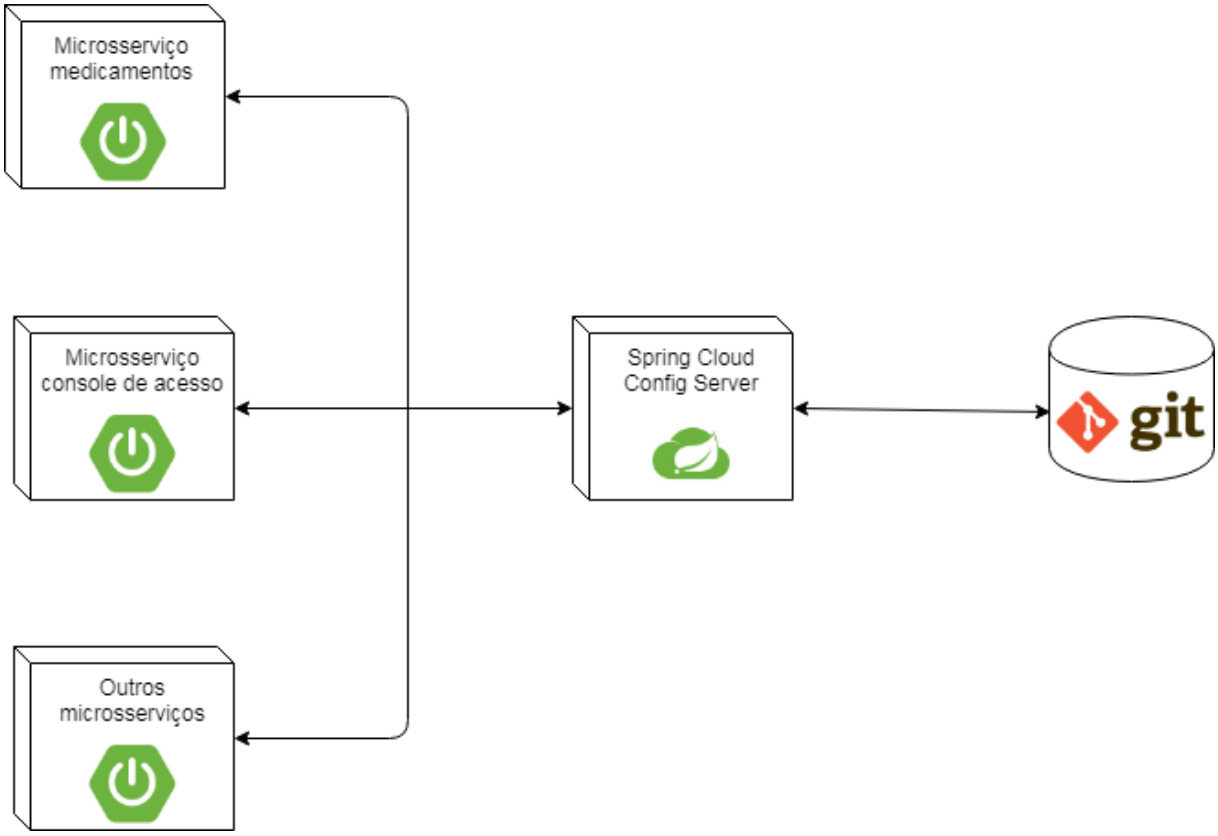


Figura 4.12: Spring Cloud Config

Descoberta e registro de serviços:

como o 'farmácia solidária' será composto por vários microsserviços que serão executados ao mesmo tempo, para facilitar o controle de endereço, portas e *status*, optou-se por

utilizar a ferramenta **Eureka**, uma vez que ela oferece fácil integração com a linguagem Java e o framework Spring Boot.

No sistema desenvolvido foi implementado o projeto 'Eureka Server' que gerencia todos os endereços e portas dos microsserviços registrados, além de indicar a indisponibilidade de algum microsserviço. Os microsserviços de medicamento e controle de acesso ao subirem no servidor, serão registrados no Eureka que, por sua vez, guarda as informações de acesso e também, quais estão disponíveis para receber requisições. Na Figura 4.13 é possível ver as instâncias registradas no 'Eureka Server' da aplicação, seus endereços e status. Os projetos 'controle-acesso', 'eureka-server', 'gateway-zuul' e 'medicamento-api' estão rodando localmente, nas portas 9092, 9091, 9999 e 9093, respectivamente. Nesta Figura também é possível observar que o status do microsserviço de medicamento está offline, enquanto dos outros estão online.

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CONTROLE-ACESSO	n/a (1)	(1)	UP (1) - DESKTOP-IO169JL:controle-acesso:9092
EUREKA-SERVER	n/a (1)	(1)	UP (1) - DESKTOP-IO169JL:eureka-server:9091
GATEWAY-ZUUL	n/a (1)	(1)	UP (1) - DESKTOP-IO169JL:gateway-zuul:9999
MEDICAMENTO-API	n/a (1)	(1)	DOWN (1) - DESKTOP-IO169JL:medicamento-api:9093

Figura 4.13: Instâncias registradas no Eureka

API Gateway:

a ferramenta **Zuul** possibilita criar a camada de abstração que irá funcionar como uma porta única de entrada, onde todas as requisições irão passar por ela antes de irem para o microsserviço desejado. Ela foi escolhida por possuir facilidade de integração com a ferramenta Eureka, conforme apresentado na Figura 4.14. Para a utilização dessa ferramenta, foi criado um projeto, chamado 'Zuul Gateway', onde foi configurado as rotas para os microsserviços de medicamento e controle de acesso. O gateway implementado, não precisa saber o endereço e a porta que estão funcionando os microsserviços, visto que essas informações estarão com o 'Eureka Server'. Em Listing 4.2, é possível visualizar as rotas implementadas no Zuul apontando para os microsserviços de controle de acesso e

medicamento. Observe que o campo 'serviceId' do Zuul, representa o campo 'application' do Eureka, mostrados na Figura 4.13.

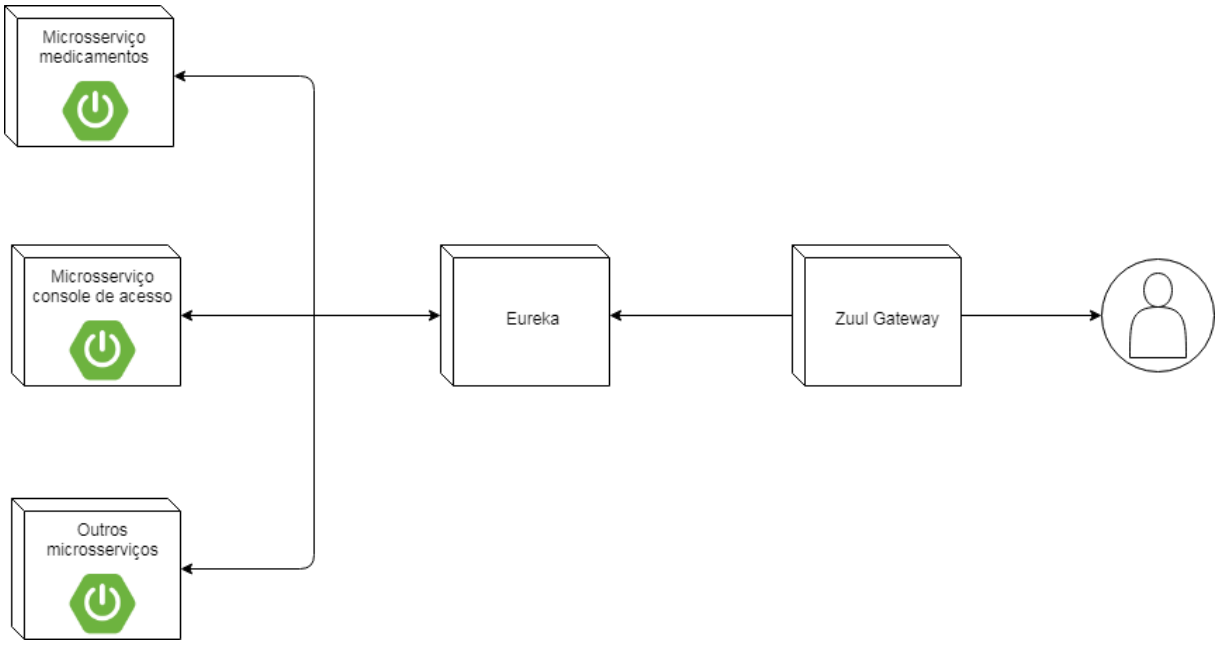


Figura 4.14: Eureka e Gateway

```
routes:
  oauth:
    stripPrefix: false
    serviceId: controle-acesso
  user:
    stripPrefix: false
    serviceId: controle-acesso
  tipos:
    stripPrefix: false
    serviceId: medicamento-api
  medicamentos:
    stripPrefix: false
    serviceId: medicamento-api
  estoque:
    stripPrefix: false
    serviceId: medicamento-api
  solicitacao:
    stripPrefix: false
    serviceId: medicamento-api
```

```
20    status:
21      stripPrefix: false
22      serviceId: medicamento-api
```

Listing 4.2: Rotas no Zuul

Autenticação:

o microsserviço de controle de acesso, foi implementado utilizado o **Spring Security OAuth2** que foi escolhido pela facilidade de integração com o Spring Boot. Ele funciona assim: o *frontend* envia os dados de usuário e senha para o microsserviço de controle de acesso. Se o usuário for válido, o 'Spring Security OAuth' autentica e emite um *token* de acesso na resposta. Esse *token* deverá ser enviado sempre que se fizer uma requisição aos microsserviços. Isto porque é através do *token* que se valida se o usuário está autorizado ou não a acessar o serviço.

Virtualização:

para virtualização dos microsserviços utilizou-se a ferramenta **Docker** por ser a mais popular. Cada um dos cinco microsserviços projetados neste trabalho, ou seja, os microsserviços de regras de negócio (controle-acesso e medicamento) e os de infraestrutura (config-server, eureka-server e gateway-zuul) irão atuar ao mesmo tempo e, para isso, eles foram "conteineirizados" a partir de uma imagem Docker. Isso também permite que os contêineres criado a partir dessas imagens possam ser instanciados em qualquer ambiente, desde o de desenvolvimento até o de produção.

É através do arquivo chamado $Dockerfile$ que se cria uma imagem. Esse arquivo está localizado no diretório raiz do projeto de cada microsserviço. Em Listing 4.3 é apresentado o $Dockerfile$ do microsserviço de medicamento com uma breve descrição de seu funcionamento. A primeira instrução $FROM$ indica de qual imagem será gerada a nova imagem, ou seja, a imagem do microsserviço de medicamento usará a imagem $java$: $alpine$ como base. A cláusula '$WORKDIR$' definirá o diretório trabalhado na imagem; o '$COPY$' copiará o arquivo '$config - client - entrypoint.sh$' e o pacote jar gerado do microsserviço para dentro da imagem; o comando RUN executa algum comando, por

exemplo, a linha '*RUN chmod 755 ./config − client − entrypoint.sh*' será executado o comando para alterar a permissão do arquivo '*config − client − entrypoint.sh*'. A última instrução, '*ENTRYPOINT*', executa o arquivo '*config − client − entrypoint.sh*' quando um contêiner for criado a partir dessa imagem.

```
1  FROM java:alpine
2
3  WORKDIR aplicacao
4
5  COPY ./config-client-entrypoint.sh ./
6  COPY ./target/*.jar ./
7  RUN mv ./*.jar app.jar
8
9  RUN chmod 755 ./config-client-entrypoint.sh
10
11 ENTRYPOINT ["./config-client-entrypoint.sh"]-cloud/bin/config-client-entrypoint.sh
```

Listing 4.3: Dockerfile do microsserviço de medicamento

Para que os microsserviços sejam implantados antes dos demais, foi desenvolvido o arquivo '*config − client − entrypoint.sh*' apresentado em Listing 4.4. Esse arquivo possui um comando '*while*' que verifica a todo momento se os contêineres de *config − server* e *eureka − server* estão inicializados para que os contêineres das imagens de *controle − acesso*, *medicamento* e *gateway − zuul* possam ser criados.

```
1  #!/bin/sh
2  while ! nc -z config-service 8080 ; do
3      echo "Waiting for upcoming Config Server"
4      sleep 2
5  done
6
7  while ! nc -z eureka-service 8080 ; do
8      echo "Waiting for upcoming Eureka Server"
9      sleep 2
10 done
11
```

```
12 java -jar -Dspring.profiles.active=prod -Dspring.cloud.config.uri=http://config-service:8080 app.jar
```

Listing 4.4: Arquivo 'config-client-entrypoint.sh' executado na criação dos contêineres

Para facilitar a criação e execução dos contêineres da aplicação, foi utilizado o arquivo *docker − compose*. Em Listing 4.5, o *docker − compose* da aplicação além de ter os cinco contêineres mencionados, também possuirá os contêineres do *frontend* e do banco de dados. Com isso, basta apenas um comando para que toda a aplicação (front, back e banco) seja utilizada em qualquer computador da mesma forma que é executada no servidor de produção.

```
1  version: "2"
2
3  services:
4
5      config.service:
6          image: farmacia/config-service:latest
7          restart: always
8          container_name: config-service
9          build: ./config-server/
10
11     eureka.service:
12         image: farmacia/eureka-service:latest
13         restart: always
14         container_name: eureka-service
15         build: ./eureka-server/
16         ports:
17             - 8081:8080
18
19     auth.service:
20         image: farmacia/auth-service:latest
21         restart: always
22         container_name: auth-service
23         build: ./auth-server/
24         ports:
25             - 8083:8080
26
```

```
27    medicamento.service:
28        image: farmacia/medicamento-service:latest
29        restart: always
30        container_name: medicamento-service
31        build: ./medicamento-api/
32
33    zuul.service:
34        image: farmacia/zuul-service:latest
35        restart: always
36        container_name: zuul-service
37        build: ./zuul/
38        ports:
39            - 8082:8080
40
41    frontend:
42        image: farmacia/frontend:latest
43        restart: always
44        container_name: frontend
45        build: ./frontend/
46        ports:
47            - 8080:80
48
49    memories:
50        image: mysql:5.6
51        container_name: memories
52        environment:
53            - MYSQL_ROOT_PASSWORD=root
```

Listing 4.5: Arquivo docker-compose

4.5.2 Microsserviço - controle de acesso

O microsserviço de controle de acesso é responsável pelo controle de autenticação e autorização da aplicação. Esse microsserviço foi implementado seguindo o artigo 'Arquitetura de MicroServices com Spring Cloud e Spring Boot - Construindo um Servidor de Autorização OAuth2' de SILVA e SILVA (2017).

O sistema 'farmácia solidária' possui dois tipos de permissão para usuários: *ROLE_USER* e *ROLE_ADMIN*. O usuário que possui somente a permissão *ROLE_USER* será o cliente que vai interagir com o sistema para requisitar ou doar um medicamento. Já o usuário administrador, possuirá as duas permissões, visto que ele é a pessoa responsável por gerenciar o sistema, como por exemplo, controlar o estoque dos medicamentos.

Para apresentar o funcionamento dos microsserviços, foram feitas telas simples somente com o objetivo de validá-los. O *frontend* final que será utilizado pelo sistema, ainda está em fase de desenvolvimento pelos demais bolsistas e voluntários. A Figura 4.15 apresenta a tela inicial do sistema em que o usuário digitará os campos usuário e senha. O *frontend* enviará uma requisição para o endereço 'http://localhost:9999/oauth/token'. Se o usuário for válido, o *backend* retornará um token, conforme exemplo apresentado na Figura 4.16.

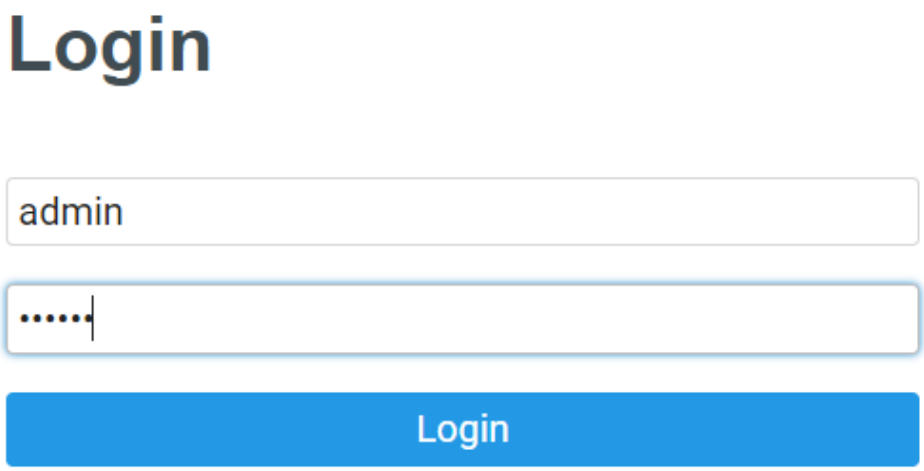


Figura 4.15: Tela login

Esse token será inserido no *Local Storage* do navegador pelo *frontend*. Com o token armazenado, ele poderá ser acessado por qualquer tela que necessite estar logado. O login só se fará necessário novamente em caso de logout ou caso o token expire, visto que ele tem um certo tempo de vida.

Depois de logado, o *frontend* fará uma segunda requisição internamente ao en-

```
"access_token": "c13fcb47-aa4b-4b80-99ba-6ef9cb8d1e93",
"token_type": "bearer",
"refresh_token": "72521e17-c636-43de-9e1e-4661aefe8294",
"expires_in": 1332,
"scope": "read write"
```


Figura 4.16: Exemplo de token que o microsserviço de controle de acesso retorna

dereço 'http://localhost:9999/user'. O serviço *user* é o que faz parte do microsserviço de controle de acesso, retornando os dados do usuário logado, como nome e permissões. Os dados do usuário também serão armazenados no *Local Storage*. Na Figura 4.17 é possível observar que o campo *'password'* está com valor *nulo*, visto que não seria nada seguro retornar a senha do usuário. Caso o usuário tente acessar uma página que não está autorizado, ele será redirecionado para a página principal.

Key	Value
token	245aa09c-a8ab-4f82-bf99-c5a9031c42f8
usuario	{"password":null,"username":"admin","authorities":[{"authority":"ROLE_ADMIN"},{"authority":"ROLE_USER"}],

Figura 4.17: Local Storage do navegador depois do login

4.5.3 Microsserviço - medicamento

O microsserviço de medicamento é responsável pelo controle de estoque e solicitações de medicamentos.

Solicitação de medicamento

A Figura 4.18 representa a tela utilizada para pesquisar um medicamento com a finalidade de se fazer uma solicitação. No momento que se entra nessa tela, é feita uma requisição para o serviço GET de medicamento que retorna uma listagem dos medicamentos que estão disponíveis.

Ao selecionar um medicamento e clicar no botão '+', se o usuário não estiver logado, o login será solicitado, visto que essa ação é restrita apenas a usuários logados. Para validar o login, o *frontend* recupera o token do *Local Storage* e envia para o microsserviço de controle de acesso autentica-lo. Caso o *Local Storage* esteja vazio, nenhum

Figura 4.18: Listagem de medicamentos disponíveis

login ainda foi realizado. Ao validarmos o usuário, é exibida a tela modal apresentada na Figura 4.19. Ao informar a quantidade e clicar em 'Solicitar', é feita uma requisição ao microsserviço de medicamento, que verificará se a quantidade pedida está disponível ou não no estoque. Em caso positivo, microsserviço vai alterar o status do medicamento para 'reservado', além de retornar uma mensagem de sucesso, conforme mostrado na Figura 4.20, informando até que dia deve-se buscar o medicamento (três dias uteis após a data de solicitação, sendo esse cálculo realizado pelo serviço). Caso contrário, uma mensagem de 'estoque insuficiente' é retornada e nenhuma ação é executada.

Estoque

A Figura 4.21 apresenta a tela de estoque. Ela contém uma listagem com todos os medicamentos cadastrados e seus respectivos status e só poderá ser acessada pelo administrador do sistema. Caso o status do medicamento esteja 'reservado' e o solicitante busque-o no ponto de apoio, o administrador deverá acionar o botão 'doar' e confirmar a doação, conforme mostrado na Figura 4.22. O microsserviço de medicamento salvará a solicitação no banco e mudará o *status* do medicamento. Caso acabe o medicamento do estoque, o *status* mudará para 'doado', caso contrário, o *status* continuará sendo 'disponível'. Sendo o status 'disponível', o administrador poderá excluir esse medicamento mediante ao preenchimento de um motivo, conforme mostra a Figura 4.23. O microsserviço fará uma

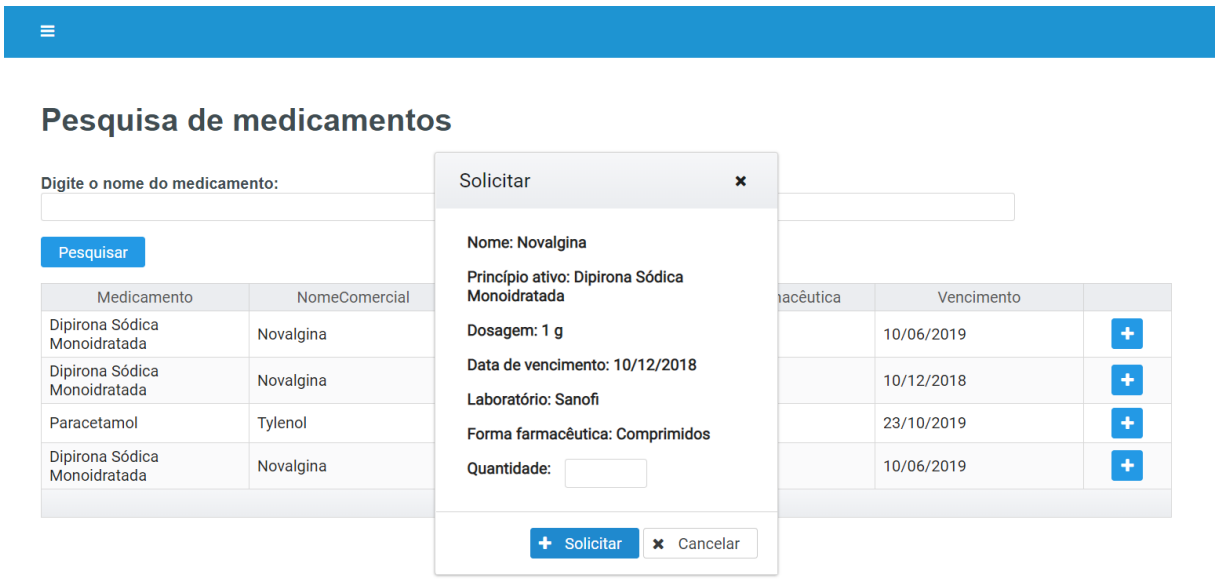


Figura 4.19: Modal para solicitação de medicamento

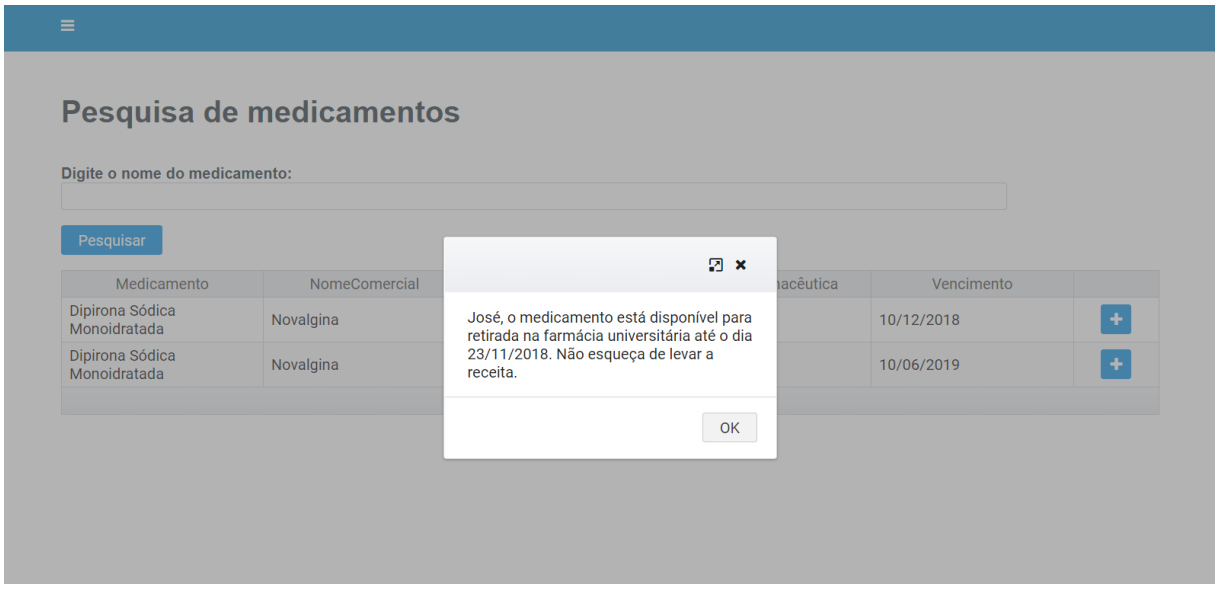


Figura 4.20: Modal com a mensagem de sucesso

exclusão lógica, alterando somente o *status* para 'excluído'.

A Figura 4.24 apresenta a tela para cadastro de medicamento e seus campos a serem preenchidos com informações sobre o medicamento. Ao cadastrar um lote pela primeira vez, o medicamento será salvo no banco de dados. Caso o lote já esteja cadastrado, será feita somente uma atualização sobre a quantidade no estoque para aquele medicamento.

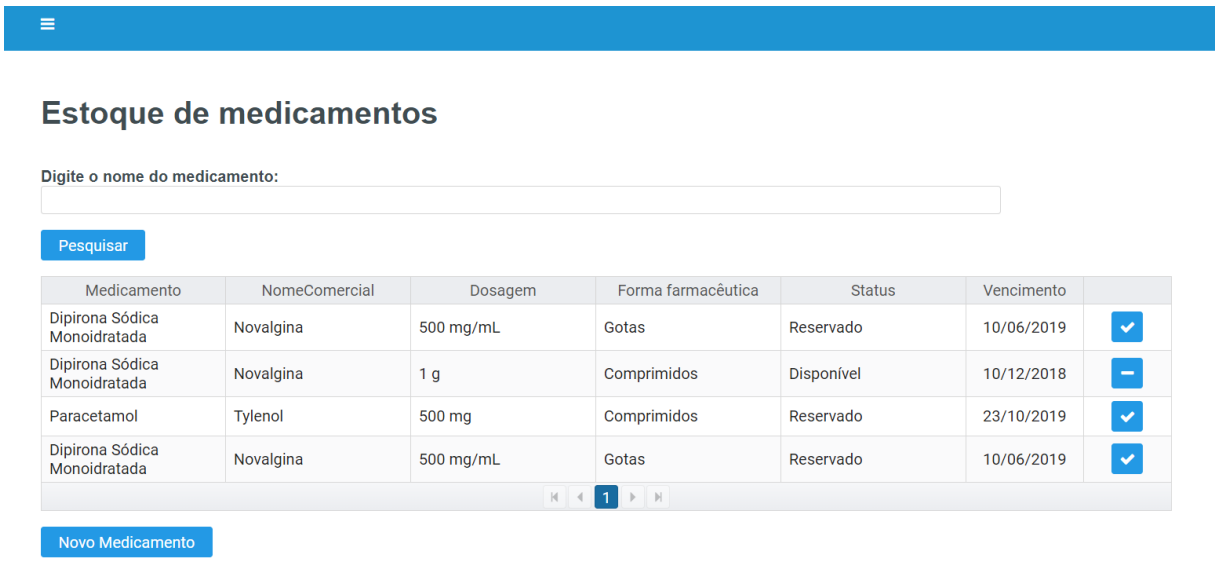


Figura 4.21: Listagem dos medicamentos cadastrados

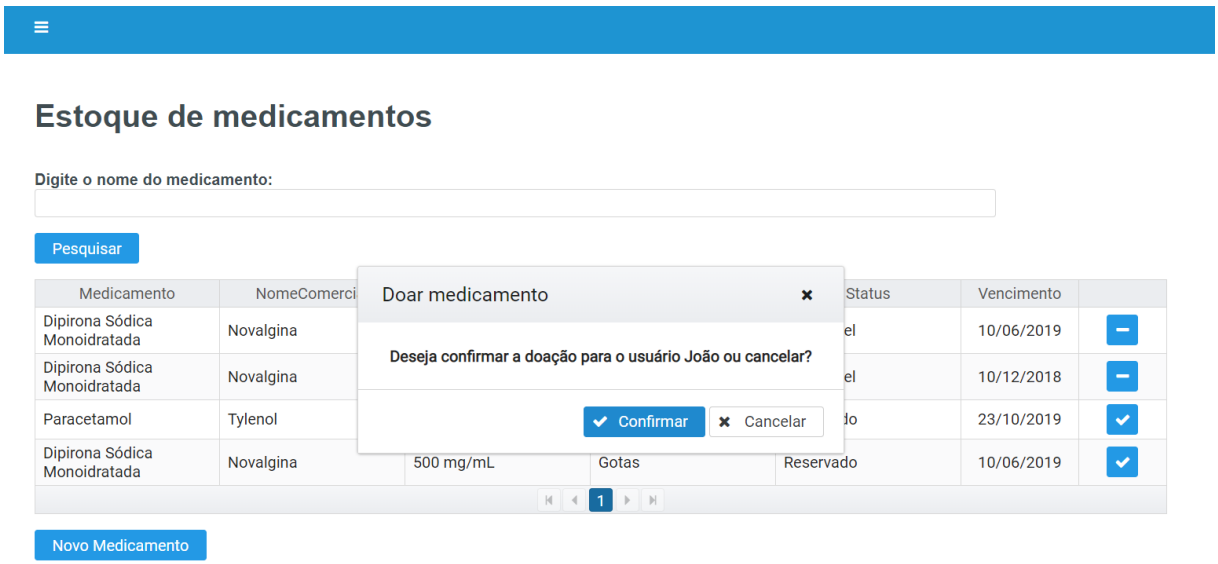


Figura 4.22: Modal para confirmar a doação de um medicamento

É importante frisar que ao fazer requisições a serviços que precisem de autenticação, o token sempre será enviado no header da requisição, para que assim, o microsserviço possa validar a permissão do usuário.

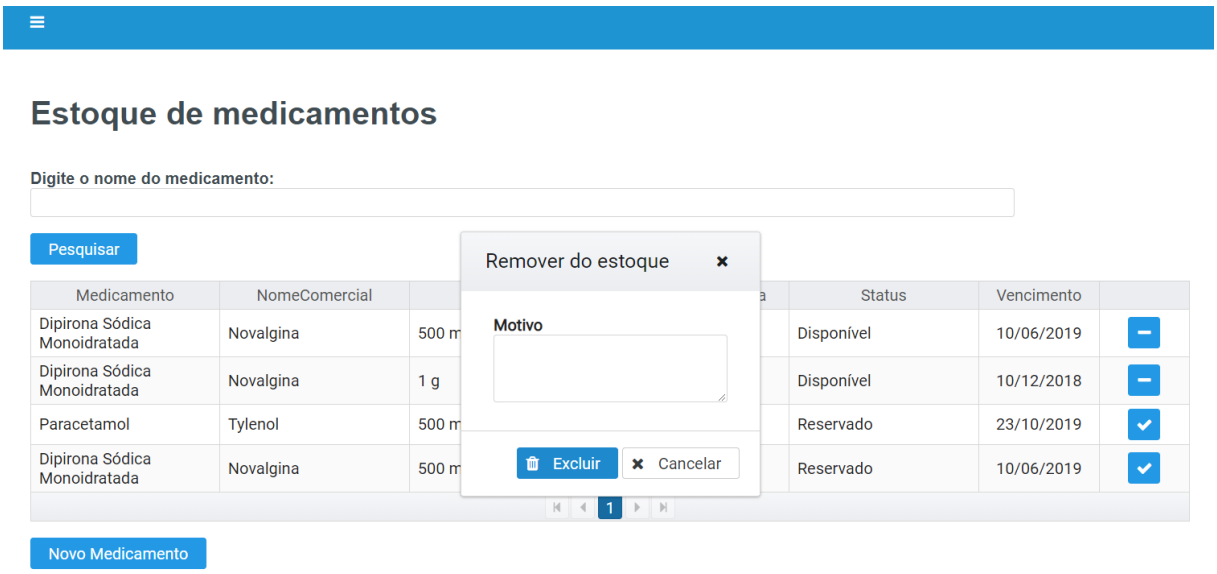


Figura 4.23: Modal para excluir medicamento

Figura 4.24: Tela para cadastro de medicamento

4.6 Sistema

Na figura 4.25, é possível visualizar o funcionamento do sistema. Observa-se os microsserviços controle de acesso e medicamento, cada um possuindo um banco de dados exclusivo e expondo seus recursos através de APIs REST. O serviço de configuração centralizando todas as configurações no repositório git, os serviços de descoberta e registro que contém os endereços e status dos microsserviços e o API GATEWAY, funcionando como uma porta única de entrada.

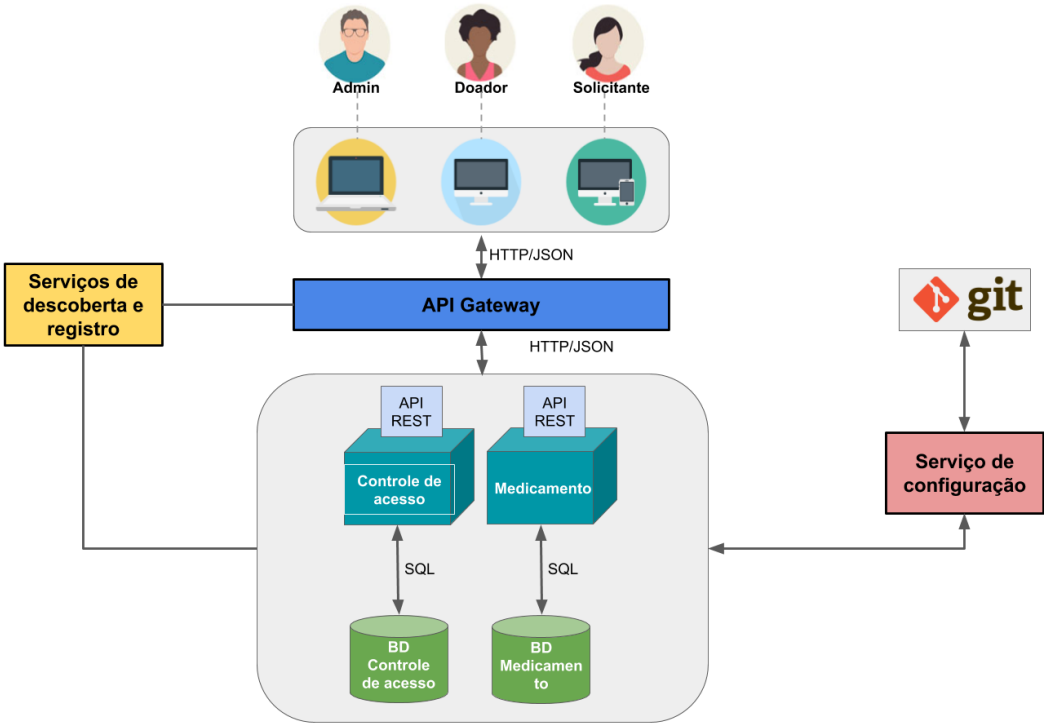


Figura 4.25: Sistema 'farmácia solidária'

5 Conclusão

Neste trabalho de conclusão de curso foi apresentado o processo de desenvolvimento de um sistema de informação utilizando uma arquitetura baseada em microsserviços e que seguiu o processo unificado ágil. Neste trabalho, foram realizados o levantamento de requisitos e o projeto arquitetural de toda aplicação e desenvolvidos os microsserviços de controle de acesso e de medicamento. O projeto apresentou o desenvolvimento baseado em uma arquitetura de microsserviços considerando os padrões mais utilizados na literatura. Além disso, foram empregadas diversas tecnologias e ferramentas para o desenvolvimento e gerenciamento de microsserviços que estão em destaques e se estabilizando no mercado.

Foi possível observar que por meio dessa arquitetura as seguintes vantagens:

- Resiliência: uma falha não compromete todo o sistema;
- Escalabilidade: possibilidade de desenvolver novos serviços sem que haja impacto aos existentes;
- Flexibilidade: cada microsserviço pode utilizar a linguagem e recursos que lhe convém;

O desenvolvimento deste sistema trará diversas facilidades. Do ponto de vista do desenvolvedor de software, pode-se citar a reutilização dos microsserviços em outros projetos uma vez que eles estão disponíveis para acesso público no GitHub, no seguinte repositório `https://github.com/BarbaraLopes/tcc-farmacia-solidaria`. Considerando também que há poucos sistemas baseados em arquitetura de microsserviços, este sistema poderá fazer parte de *benchmarking* e serem utilizados para pesquisas futuras na comunidade acadêmica. Do ponto de vista da comunidade em geral, o 'farmácia solidária' gerará impacto social uma vez que oferecerá apoio às pessoas que necessitam de medicamentos mas que não tem condições de adquiri-los, além do impacto ambiental, já que menos medicamentos serão descartados no ambiente.

Durante o desenvolvimento do projeto, foram encontradas algumas dificuldades. A primeira e maior delas foi o entendimento do que é um microsserviço, qual o tamanho

ideal e como ele distingue-se de um serviço. Para a montagem da arquitetura de microsserviços, foram necessárias muitas horas de estudo para conseguir se desvencilhar do padrão monolítico, que ainda é um padrão rotineiro usado em aplicações atuais. Outro obstáculo a ser superado foi a escassez de materiais em português e prático sobre o assunto, visto ser um tema ainda novo para a área acadêmica, uma vez que essa abordagem começou na indústria. Nesse sentido, este trabalho contribui também para a comunidade brasileira, uma vez que descreve todo o processo realizado e aponta tecnologias que poderão ser utilizadas e está escrito em português.

5.1 Contribuições

Destaca-se como contribuições deste trabalho:

- O desenvolvimento de parte de um sistema web;
- Definição de uma arquitetura baseada em microsserviços para um sistema de informação;
- Utilização de tecnologias e ferramentas voltadas para a arquitetura de microsserviços;
- Disponibilização de artefatos para a comunidade;
- Relato em português dos processos e ferramentas utilizadas.

5.2 Trabalhos Futuros

Como trabalhos futuros, pretende-se:

- Finalizar o do desenvolvimento dos microsserviços;
- Realizar a integração do gateway com os demais microsserviços;
- Utilizar outras abordagens para a definição da granularidade de microsserviços e realizar experimentos para saber o impacto;

- Utilizar outras abordagens para integração dos microsserviços e realizar experimentos para saber o impacto;
- Utilizar outras tecnologias para o desenvolvimento de microsserviços e analisar as vantagens e desvantagens;
- Definir abordagens e/ou técnicas para teste de microsserviços;
- Realizar o teste dos microsserviços bem como do sistema como um todo;
- Implantar tecnologias para integração contínua;
- Desenvolver aplicativos para smartphones baseado no presente projeto;

Referências Bibliográficas

Aderaldo, C. M.; Mendonça, N. C.; Pahl, C. ; Jamshidi, P. **Benchmark requirements for microservices architecture research**. In: 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE), p. 8–13, May 2017.

Afonso, A. **O que é angular?** https://blog.algaworks.com/o-que-e-angular/, 2018.

Antonov, A. **Spring Boot Cookbook**. Packt Publishing Ltd, 2015.

Bass, L.; Clements, P. ; Kazman, R. **Software architecture in practice**. Addison-Wesley Professional, 2003.

Boaglio, F. **Spring boot - acelere o desenvolvimento de microsserviços**. Editora Casa do Código.

Cunningham, W. **Monolithic design**. http://c2.com/cgi/wiki?MonolithicDesign, 2014.

Diedrich, C. **O que é docker?** https://www.mundodocker.com.br/o-que-e-docker/, 2015.

Diedrich, C. **Docker compose**. https://www.mundodocker.com.br/docker-compose/, 2015.

Dragoni, N.; Giallorenzo, S.; Lafuente, A. L.; Mazzara, M.; Montesi, F.; Mustafin, R. ; Safina, L. **Microservices: yesterday, today, and tomorrow**. In: Present and Ulterior Software Engineering, p. 195–216. Springer, 2017.

Duarte, L. **Microservices vs soa: entenda as diferenças**. http://www.luiztools.com.br/post/microservices-vs-soa-entenda-as-diferencas/.

desenvolvedora, E. **Farmácia do bem**. https://whizhealth.io/solucao/farmacia-do-bem/.

Fielding, R. T.; Taylor, R. N. **Architectural styles and the design of network-based software architectures**, volume 7. University of California, Irvine Irvine, USA, 2000.

Garlan, D. **Software architecture: a roadmap**. In: Proceedings of the Conference on the Future of Software Engineering, p. 91–101. ACM, 2000.

Java. **Java**. https://www.java.com/.

Larman, C. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

Lewis, J.; Fowler, M. Microservices: a definition of this new architectural term. **Martin-Fowler. com**, v.25, 2014.

Newman, S. **Building microservices: designing fine-grained systems**. "O'Reilly Media, Inc.", 2015.

OFICINA, R. **Mysql - o que é?** https://www.oficinadanet.com.br/artigo/2227/mysql, 2010.

Software, R. O. **Micro serviços: Qual a diferença para a arquitetura monolítica?** https://www.opus-software.com.br/micro-servicos-arquietura-monolitica/.

Pressman, R.; Maxim, B. **Engenharia de Software-8ª Edição**. McGraw Hill Brasil, 2016.

Prodanov, C. C.; de Freitas, E. C. **Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico-2ª Edição**. Editora Feevale, 2013.

UFJF, P. **Descarte de medicamentos no lixo comum pode contaminar o meio ambiente**. http://www.ufjf.br/limnologia/2017/05/25/descarte-de-medicamentos-no-lixo-comum-pode-contaminar-o-meio-ambiente/.

Rolla, C. **Hello world com docker**. https://www.devmedia.com.br/hello-world-com-docker/40174, 2018.

Royce, W. W. **Managing the development of large software systems: concepts and techniques**. In: Proceedings of the 9th international conference on Software Engineering. IEEE Computer Society Press, 1987.

SEI, S. E. I. **What is your definition of software architecture?**

Silva, D. B. d.; Silva, J. R. C. d. **Arquitetura de microservices com spring cloud e spring boot**. https://coderef.com.br/arquitetura-de-microservices-com-spring-cloud-e-spring-boot-parte-1-b5c9288df66d, 2017.

SIMI, R. **Startup incentiva doação de medicamentos por meio de aplicativo**. http://www.simi.org.br/noticia/Startup-incentiva-doacao-de-medicamentos-por-meio-de-aplicativo, 2017.

SIRUM. **Sirum**. https://www.sirum.org/.

Sommerville, I.; Arakaki, R. ; Melnikoff, S. S. S. **Engenharia de software**. Pearson Prentice Hall, 2008.

Souza, J. P. **Desenvolvendo microsserviços com spring cloud netflix**. http://www.matera.com/blog/post/desenvolvendo-microsservicos-spring-cloud-netflix.

Souza, E. N. d. **Criando proxy de apis com spring cloud, zuul e eureka**. https://emmanuelneri.com.br/2018/05/02/criando-proxy-de-apis-com-spring-cloud-zuul-e-eureka/, 2018.

Spring. **Router and Filter: Zuul**.

Vianna, M. **Conheça o rational unified process (rup)**. http://www.linhadecodigo.com.br/artigo/79/conheca-o-rational-unified-process-rup.aspx.

Watts, S. **Microservices vs soa: What's the difference?** https://www.bmc.com/blogs/microservices-vs-soa-whats-difference/, 2017.

Zhou, X.; Peng, X.; Xie, T.; Sun, J.; Xu, C.; Ji, C. ; Zhao, W. **Benchmarking microservice systems for software engineering research**. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE '18, p. 323–324, New York, NY, USA, 2018. ACM.

Zonetti, O. **Arquitetura de microserviços utilizando amqp**. http://tsdn.tecnospeed.com.br/blog-do-desenvolvimento-tecnospeed/post/arquitetura-de-microservicos-utilizando-amqp/sap/1, 2015.

A Documento de Requisitos do Farmácia Solidária

UNIVERSIDADE FEDERAL DE JUIZ DE FORA

Documento de requisitos
Farmácia solidária

Maio 2018

1. Introdução

1.1 Propósito

Este documento descreve os requisitos de software para o sistema de uma farmácia solidária que funcionará na cidade de Juiz de Fora - MG.

1.2 Escopo

A função do sistema é facilitar o acesso das pessoas aos medicamentos, contribuindo para apoiar o recebimento de medicamentos em desuso e repassá-los, de forma gratuita, às pessoas que não tem acesso aos mesmos.

1.3 Visão Geral

O restante deste documento está organizado como segue: inicialmente, definem-se alguns termos importantes para entendimento do documento. A Seção 2 contém uma descrição geral do sistema. A Seção 3 identifica os requisitos funcionais e não funcionais.

1.4 Definições

- Solicitar medicamento

Termo usado para requerer um medicamento online. A pessoa que requisitar um medicamento deverá apresentar receita no ato da retirada presencial do mesmo.

- Doação de medicamento

Termo usado para ceder o medicamento que não se utiliza mais. O medicamento a ser doado, deverá estar dentro do prazo de validade, já que não serão aceitos medicamentos vencidos.

- Informações

Serão notícias, dicas, informes e esclarecimentos sobre saúde e medicamentos, como por exemplo, onde deve-se descartar medicamento vencidos.

2. Descrição Geral

2.1 Perspectiva do Produto

O sistema deverá ser acessado por qualquer navegador web. Além disso, seu layout deverá ser responsivo, para que se possa adaptar ao formato de um tablet ou smartphone.

2.2 Funções do Produto

O software deverá permitir que o Departamento de Farmacologia da UFJF ou outro setor público que se interesse pelo tema, como a Prefeitura Municipal de Juiz de Fora, trabalhe a arrecadação e a oferta de medicamentos de forma computadorizada.

O sistema permitirá que a pessoa possuidora do medicamento em desuso obtenha informações para fazer a doação e, também, que as pessoas que necessitem possam fazer a solicitação desse medicamento.

Através de um formulário, a pessoa que quiser doar algum tipo de medicamento irá dizer se quer levar o medicamento em algum dos pontos de apoio ou se deseja que ele seja recolhido por voluntários no endereço cadastrado no sistema.

O Departamento de Farmacologia, ou outro setor público de interesse, receberá os medicamentos doados e realizará a inspeção visual do mesmo conforme legislação vigente. Tal etapa visa avaliar a integridade física das embalagens, do produto contido, bem como o prazo de validade do medicamento. Aqueles que forem aprovados na inspeção ficarão disponíveis no sistema para solicitações.

O usuário, ao fazer uma solicitação, deve escolher um medicamento dentre aqueles na lista de disponíveis. Ao ser solicitado, o medicamento ficará reservado por até 3 dias úteis e para recebê-lo, será preciso se direcionar até a sede da Farmácia Solidária e apresentar a prescrição médica atestando seu uso.

2.3 Características do Usuário

O cliente interage com o sistema para requisitar ou doar um medicamento, ou seja, haverá dois tipos de cliente: o solicitante e o doador.

O administrador é a pessoa responsável por gerenciar o sistema, como por exemplo, controlar o estoque dos medicamentos.

3. Requisitos Específicos

3.1 Requisitos Funcionais

F1. Manter Controle de Estoque

Será desenvolvido um módulo que controlará a entrada e saída dos medicamentos. Esses medicamentos terão detalhes de especificação como o nome do medicamento, lote, dosagem, forma farmacêutica, quantidade de comprimidos, estado da embalagem, o ponto de apoio que este medicamento está estocado e a data de vencimento.

Os medicamentos serão cadastrados de forma única e identificados por seu número de lote. Ao repassar um medicamento a uma pessoa, o administrador deverá remover o mesmo do estoque. Ao remover um medicamento, o sistema fará uma exclusão lógica (ou seja, o medicamento ainda estará no banco de dados, porém com status de "doado").

Diariamente, o sistema deverá verificar se existem medicamentos vencidos com status "disponivel", caso haja, deverá atualizar automaticamente o status para "vencido".

F2. Solicitar medicamento

O sistema deve permitir que o usuário faça a solicitação de um medicamento, dentre uma lista de disponíveis.

Ao escolher um medicamento, o usuário irá visualizar os detalhes do mesmo e indicará a quantidade necessária. O sistema verificará se a quantidade está disponível em estoque e se afirmativo, aparecerá na tela uma mensagem para que o usuário possa ir buscá-lo.

O status do medicamento passa a ser "reservado".

F3. Doar medicamento

O sistema deve permitir que o usuário escolha uma dentre duas alternativas para se doar um medicamento.

Se escolher levar até algum ponto de apoio, o sistema deve disponibilizar o endereço de todos os pontos de apoio que recebem doação de medicamentos. Se escolher a opção para que busquem em determinado endereço, o usuário deverá cadastrar o endereço e o horário para que os voluntários possam recolher o medicamento a ser doado no endereço e horário indicado.

F4. Realizar cadastro de usuário

O sistema permitirá realizar cadastro de usuários do sistema com os dados necessários para a criação de um perfil. Qualquer pessoa poderá realizar cadastro de um usuário cliente, porém somente um administrator, poderá realizar cadastro de outro usuário administrador.

F5. Manter informações

O sistema deve permitir a visualização e cadastro de informações. A visualização poderá ser acessada por qualquer usuário, enquanto que o cadastro, somente poderá ser feito por um administrador.

F6. Manter ponto de apoio

O sistema deve permitir a visualização, cadastro e exclusão de ponto de apoio. A visualização poderá ser acessada por qualquer usuário, enquanto que o cadastro e a exclusão, somente poderá ser feito por um administrador.

F7. Envio de mensagem

O sistema deve permitir que o usuário envie críticas, elogios, duvidas ou sugestões para o administrador. Terá um formulário onde o usuário deverá preencher o nome, e-mail, telefone e a mensagem. A mensagem irá para o e-mail do administrador responsável por responder essas mensagens.

F8. Efetuar login

Permitirá que os usuários tenham acesso a determinadas funcionalidade do sistema. O usuário cliente terá acesso as funcionalidades: solicitar medicamento e doar medicamentos (caso queira que o medicamento doado seja retirado em sua casa). Já o usuário administrador, terá acesso ao controle de estoque, cadastro de informações, listagem de todas as solicitações e doações cadastradas, e etc.

O usuário deverá fornecer os campos e-mail e senha.

3.2 Requisitos não funcionais

NF1. Mensagens de erro ou sucesso devem ser mostradas até 30 segundos após a interação com o usuário.

NF2. O tempo de espera na busca de medicamentos no sistema não deve exceder 30 segundos, tempo esse que começa ao clicar no botão “pesquisar” até a visualização da listagem.

NF3. O sistema deverá suportar processamento multiusuário, ou seja, vários usuários poderão utilizar o sistema simultaneamente.

NF4. O sistema deverá rodar em qualquer navegador web.

NF5. O sistema deverá fornecer uma interface amigável e prática.

3.3 Atributos

3.3.1 Disponibilidade

NF6. O site deverá estar disponível para o usuário por 24 horas, durante os 7 dias da semana, com não mais que 2% do tempo com o sistema fora do ar.

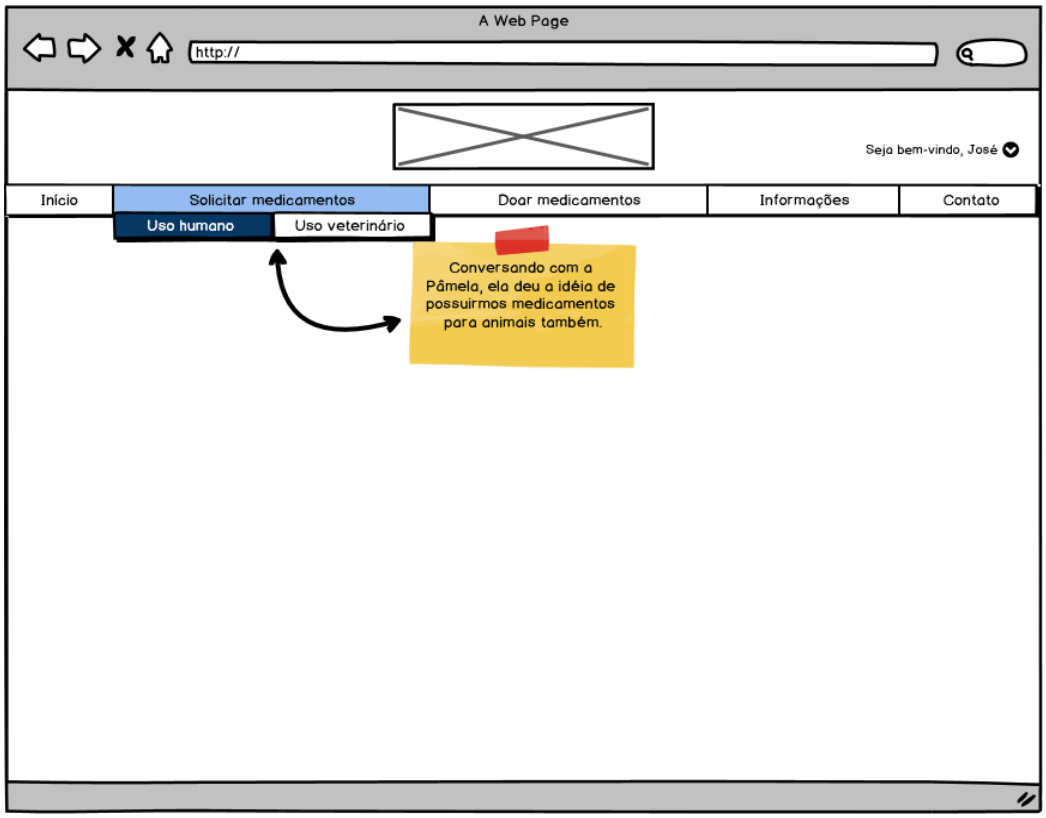
3.4.2 Segurança

NF7. Como o sistema será via WEB, ele deverá ser o mais seguro possível.

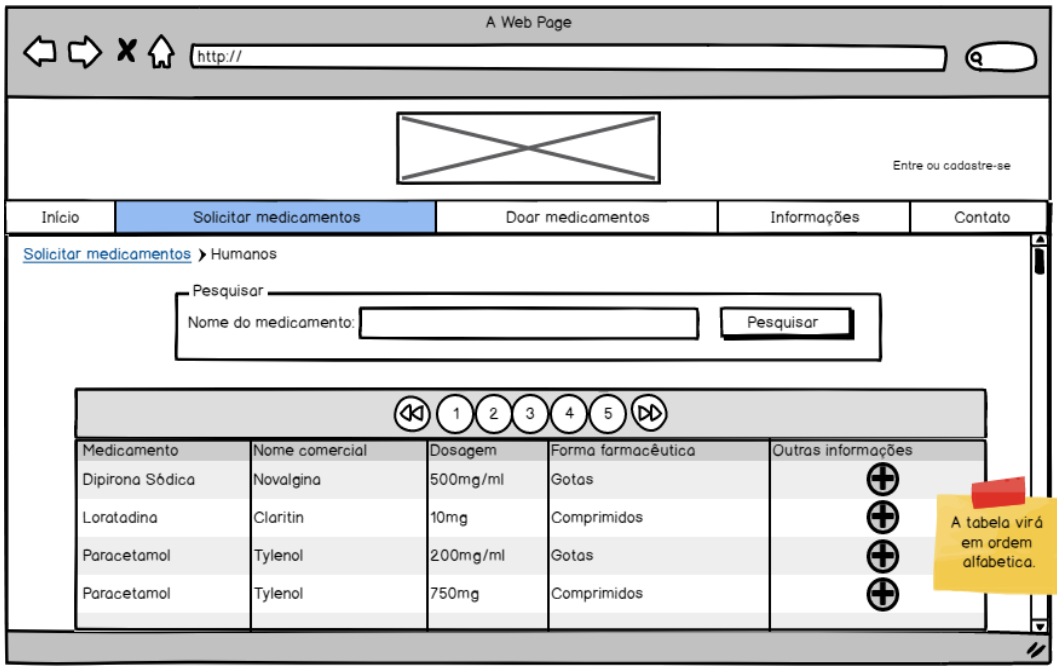
Todos os dados devem ser armazenados em um banco de dados e apenas terão acesso a esses dados, usuários cadastrados no sistema mediante um login com a solicitação de uma senha de acesso.

4. Protótipos

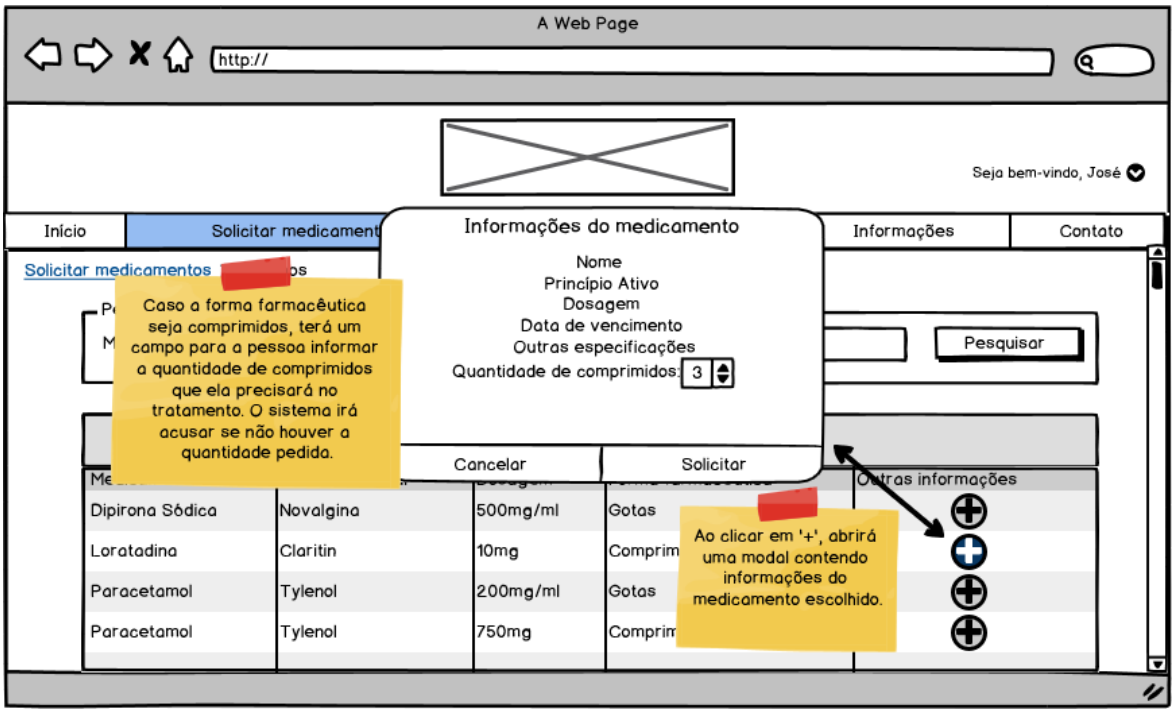
<Submenus de solicitor medicamentos>



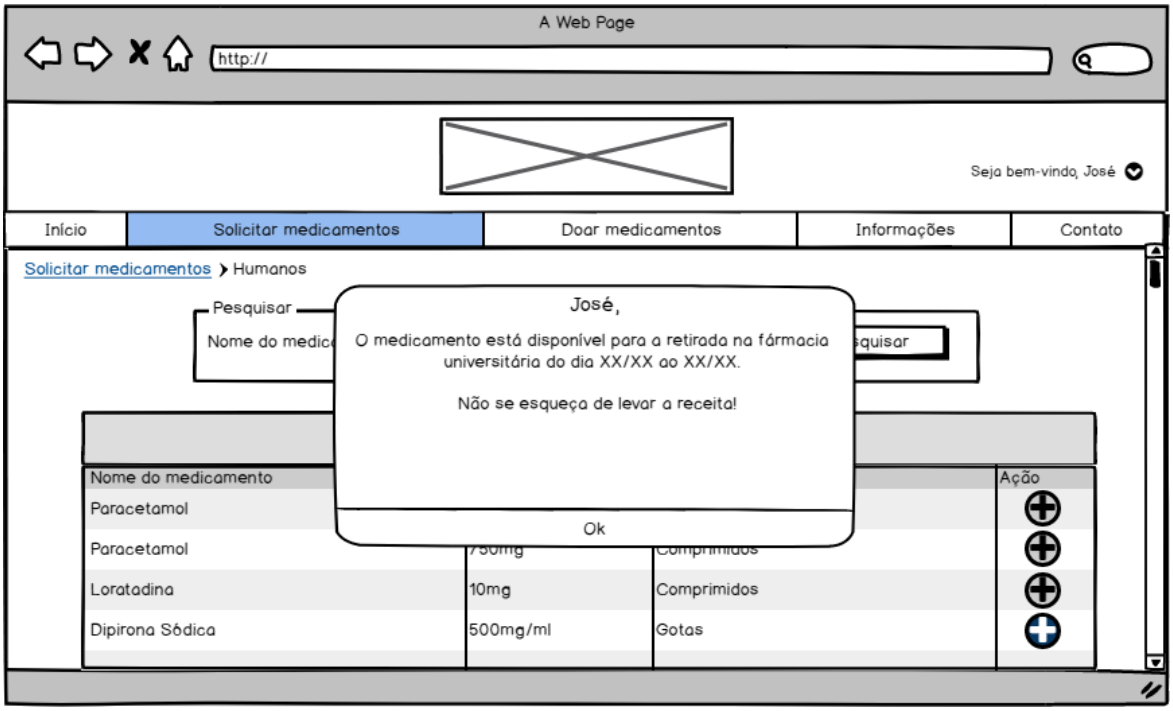
<Pesquisa de medicamentos para solicitação>



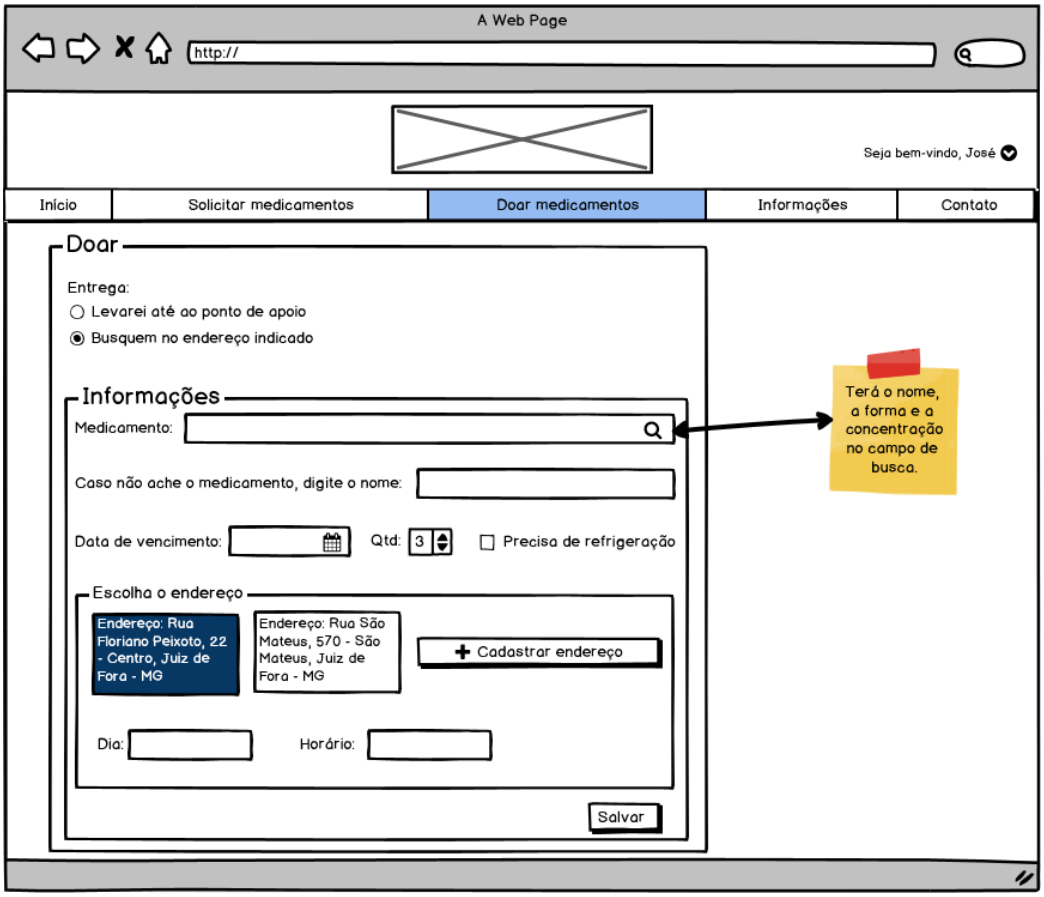
<Solicitar medicamento>



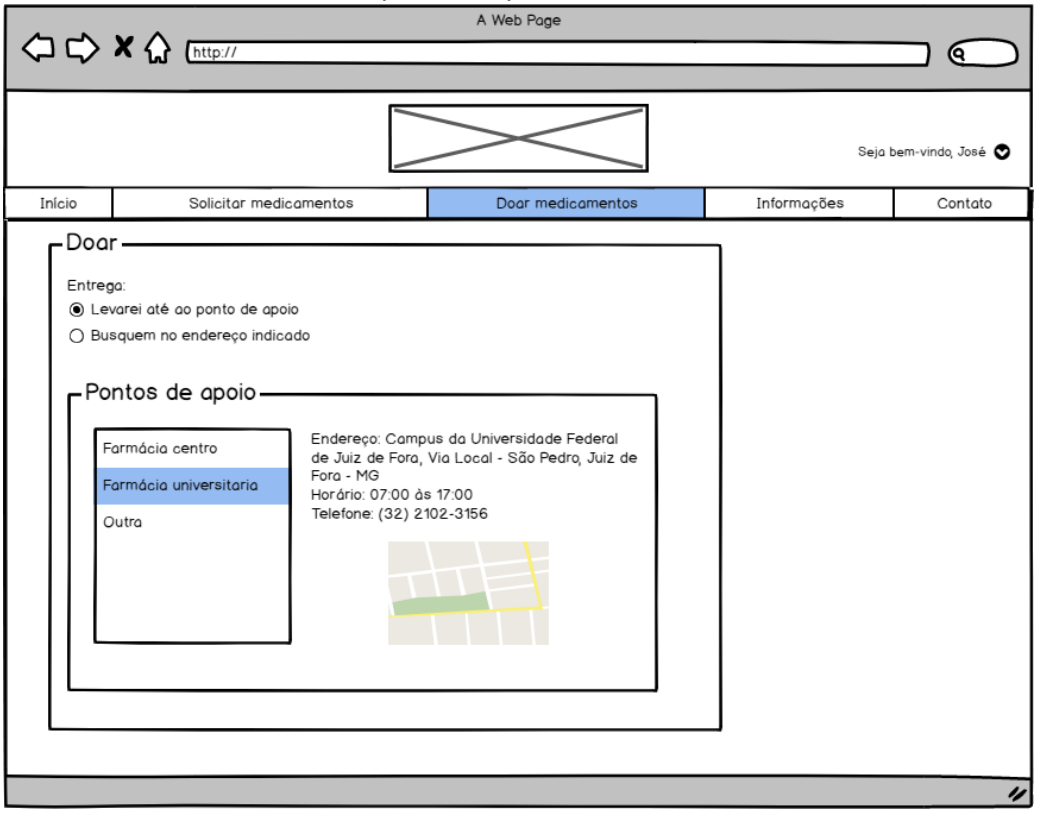
<Mensagem de sucesso da solicitação>



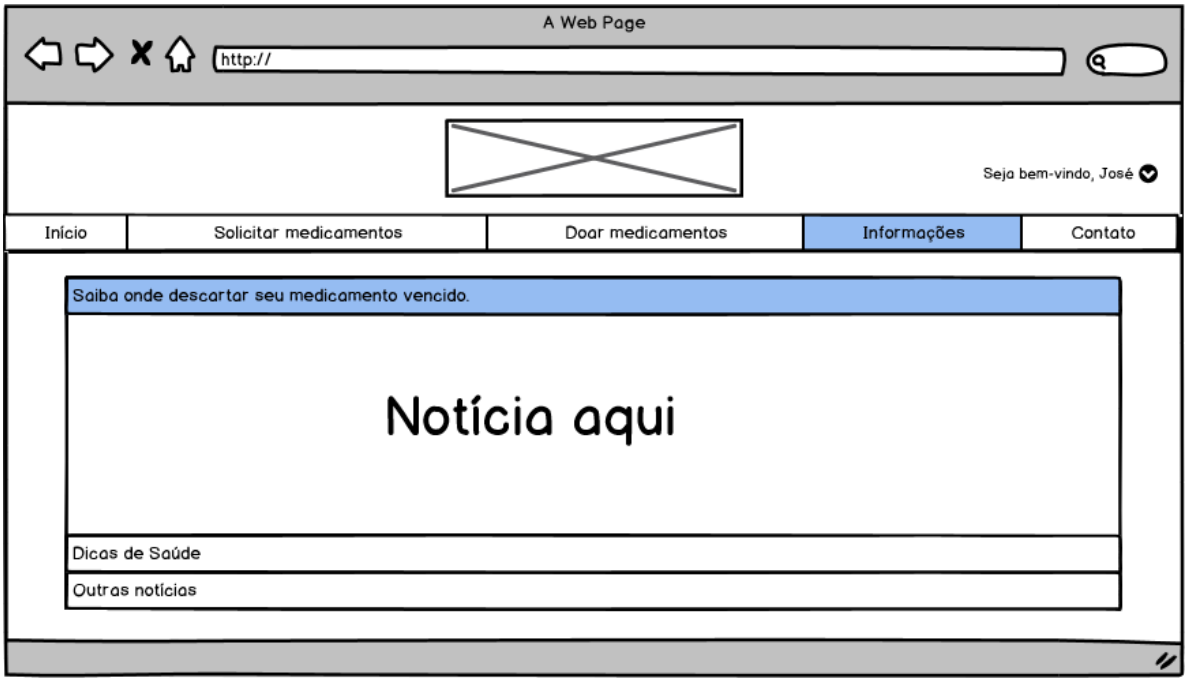
<Doar medicamento - Buscar no endereço indicado>



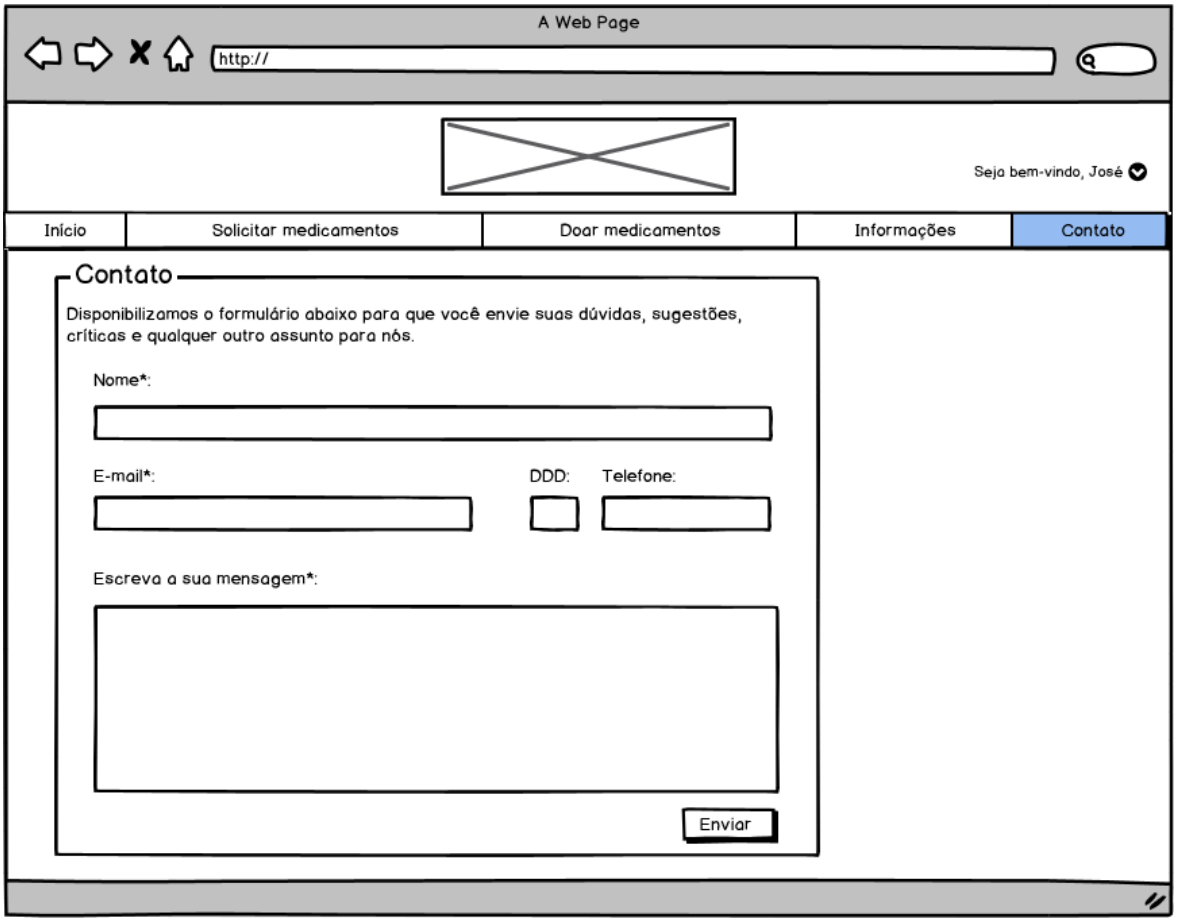
<Doar medicamento - Levar ao ponto de apoio>



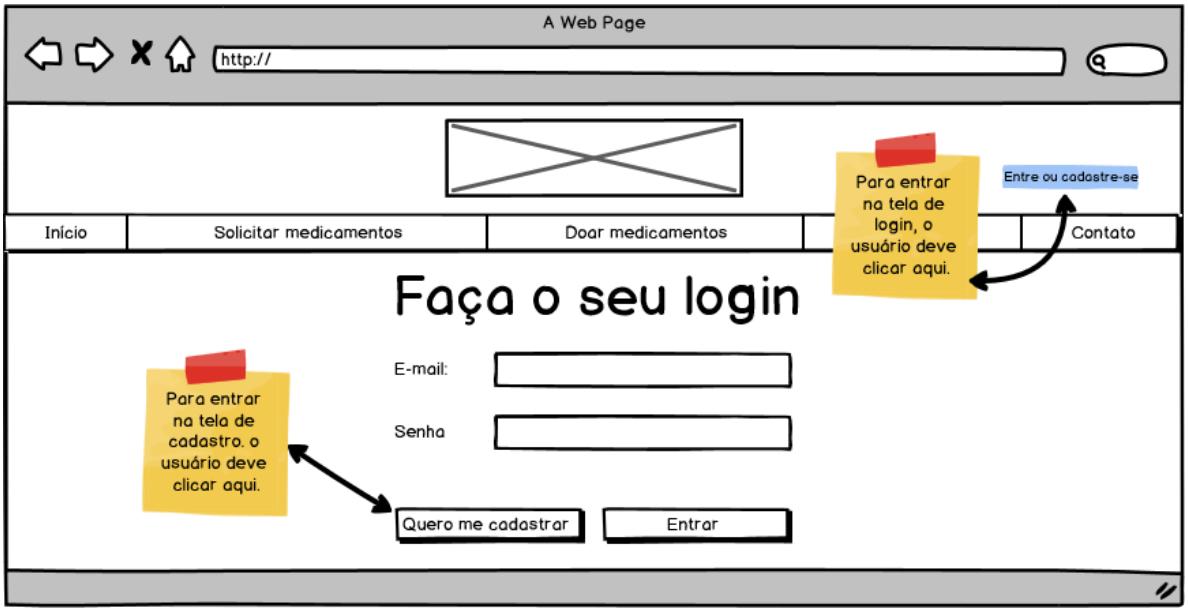
<Informações>



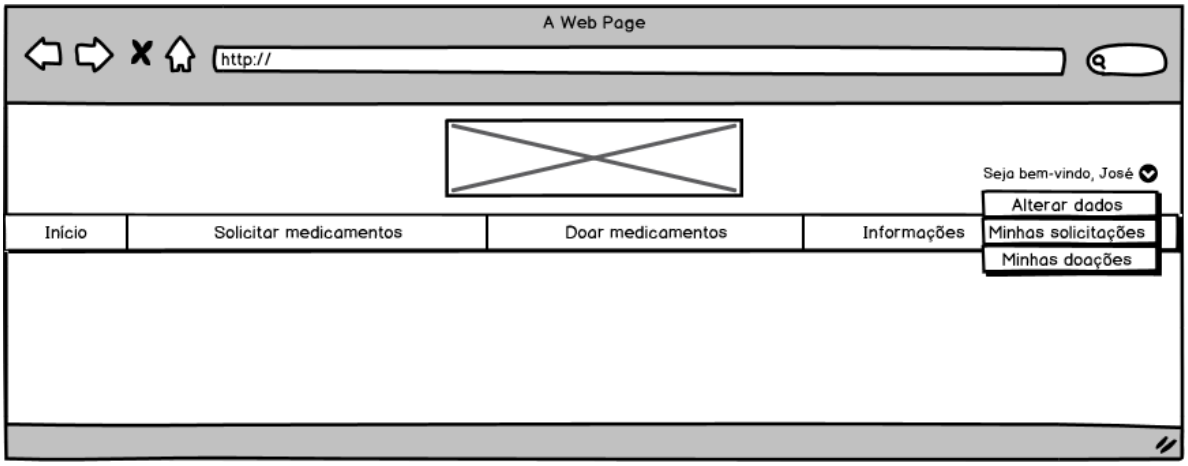
<Contato>



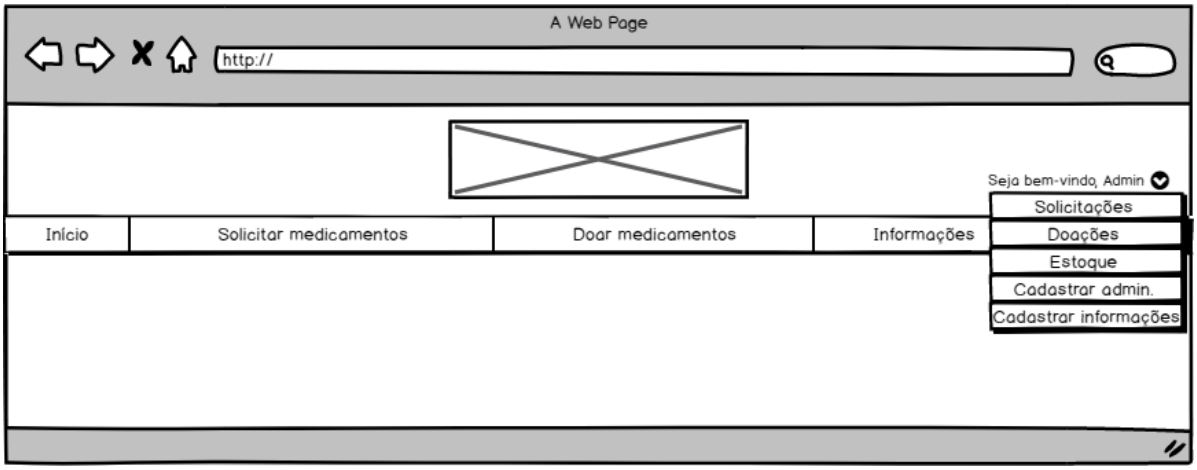
<Login>



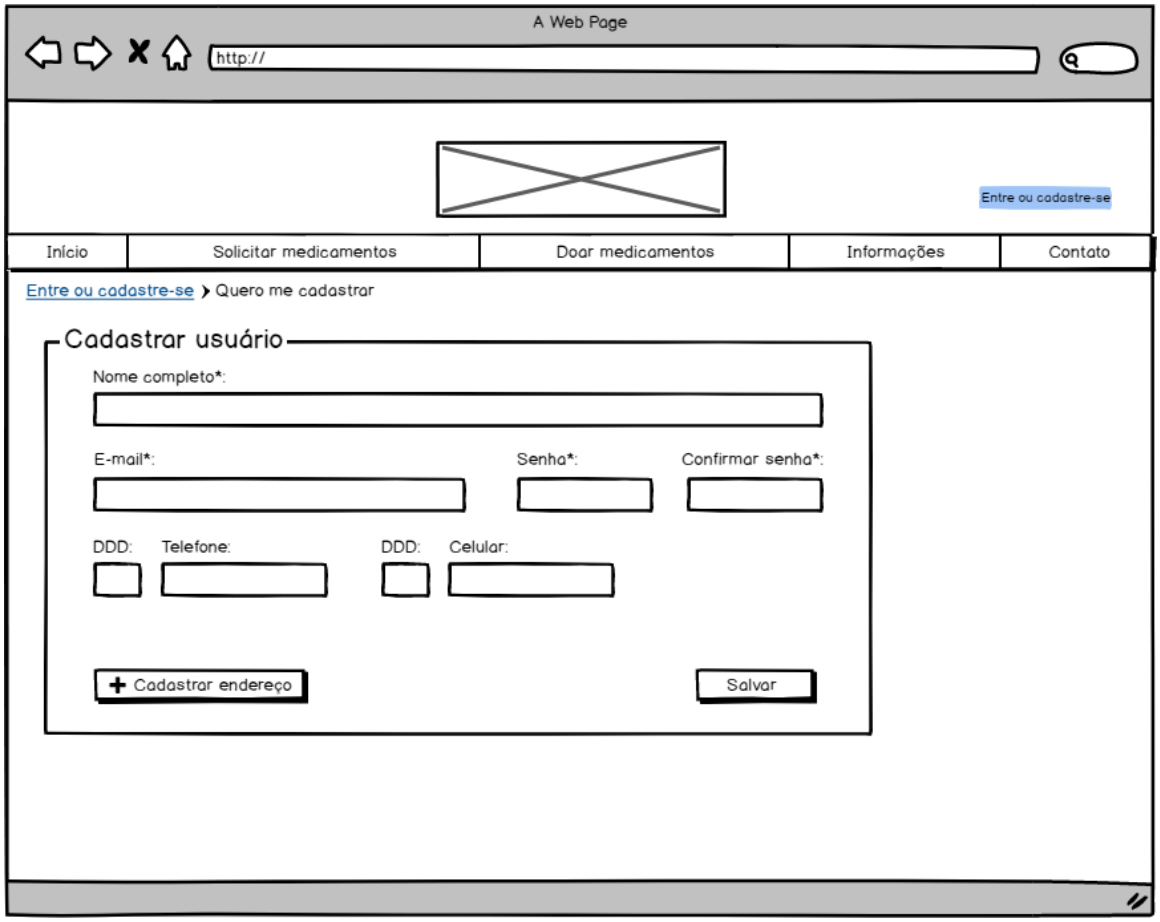
<Menu usuário comum>



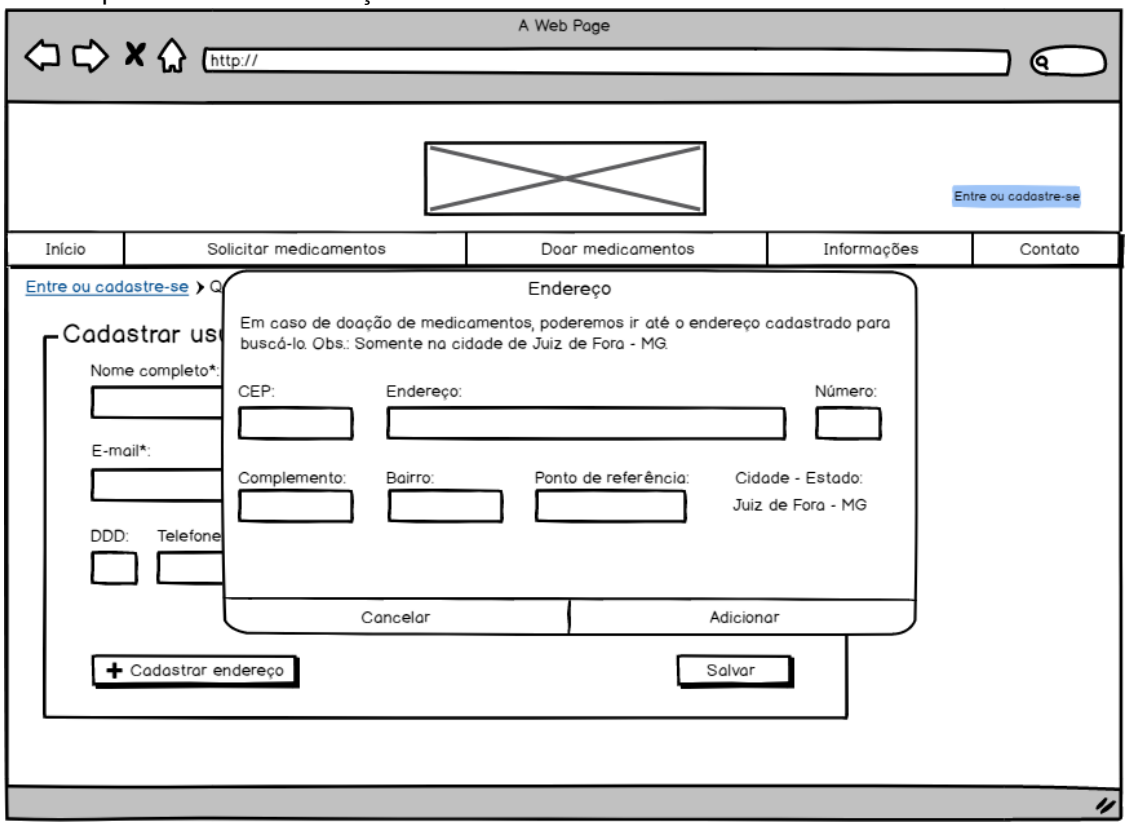
<Menu usuário administrador>



<Tela de cadastro de usuário comum>

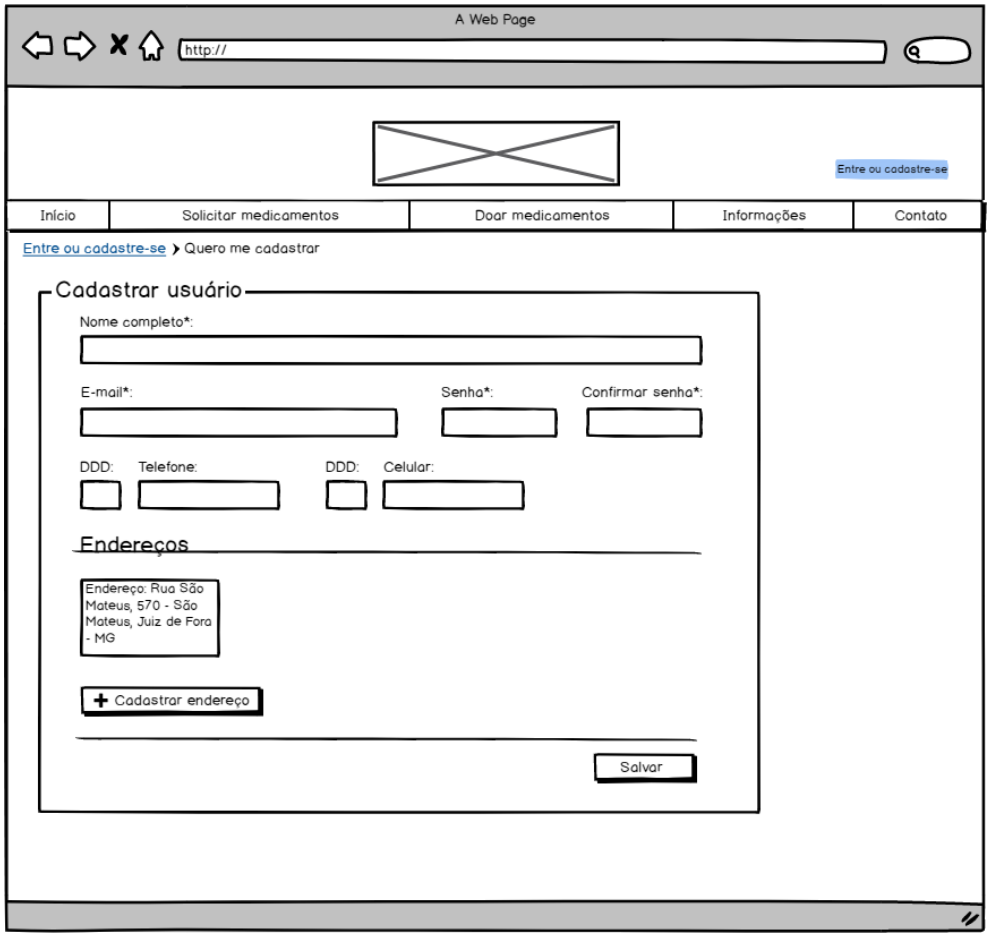


<Modal para adicionar endereço>

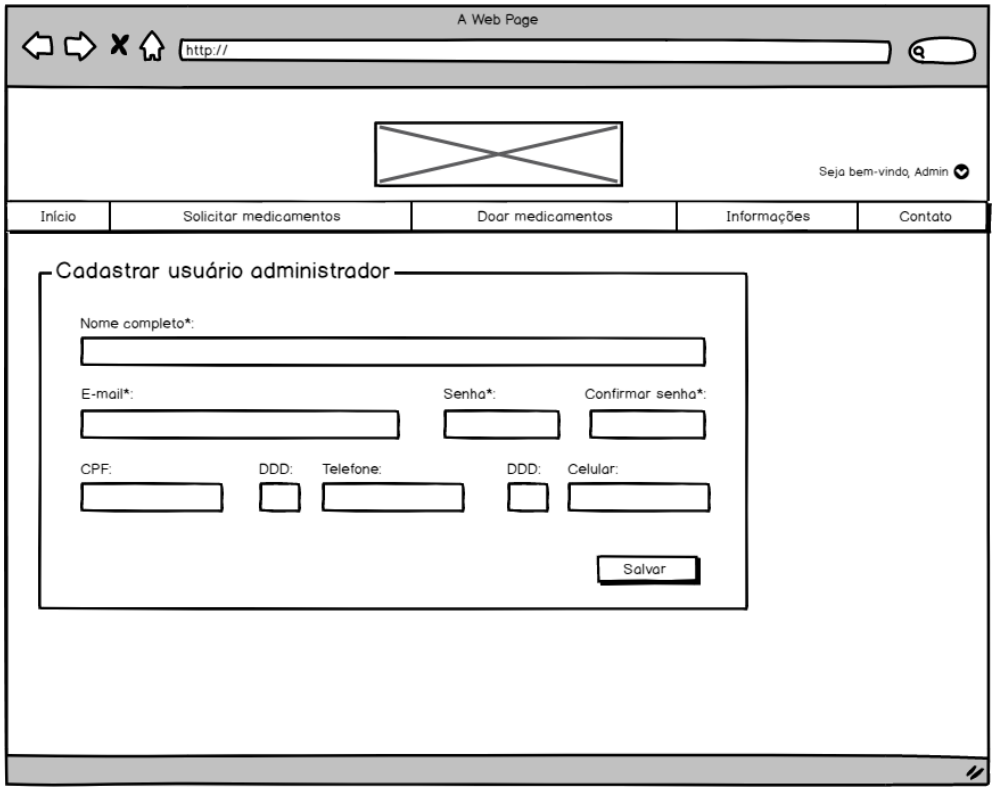


Documento de requisitos

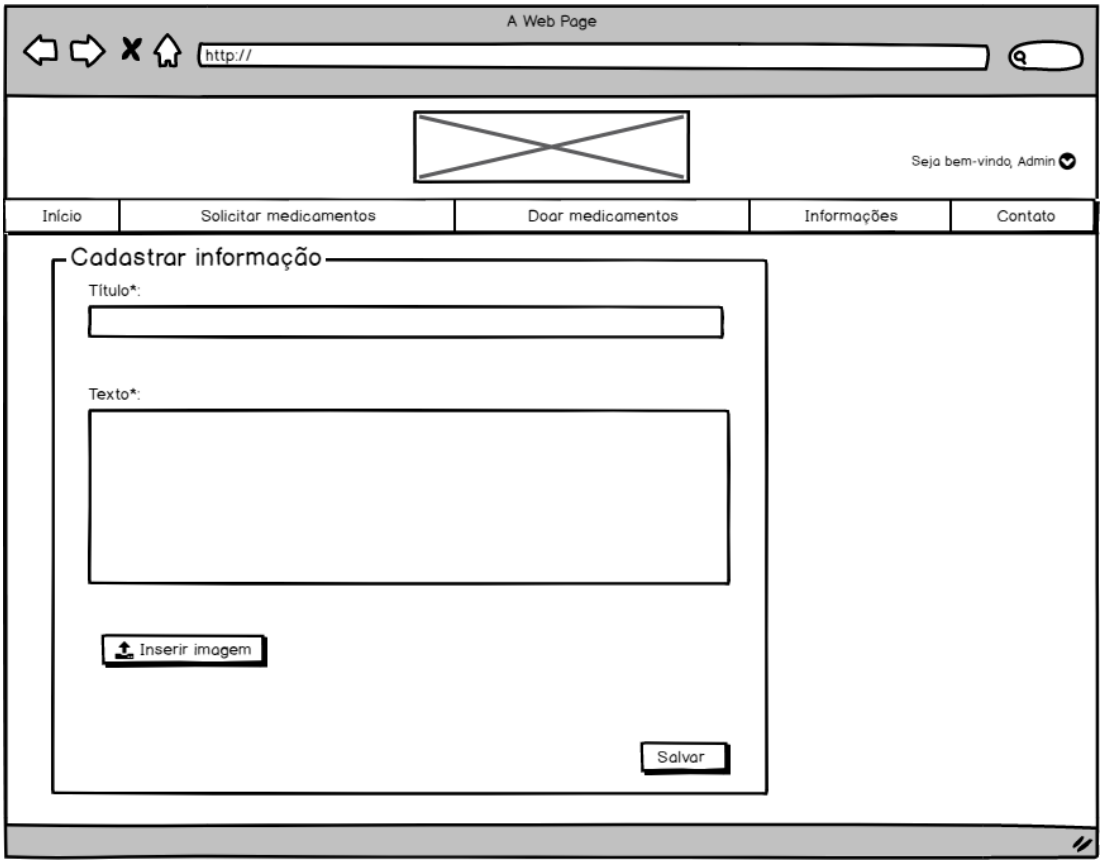
<Tela de cadastro de usuário comum com endereço adicionado>



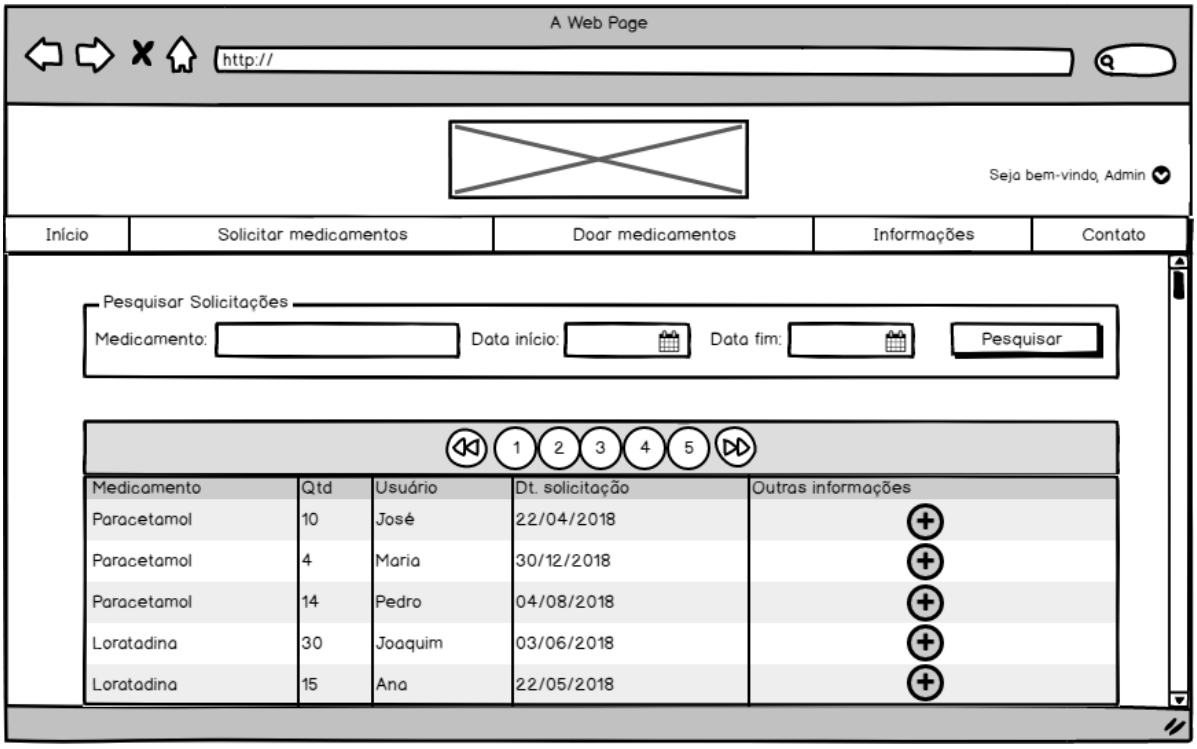
<Tela de cadastro de usuário administrador>



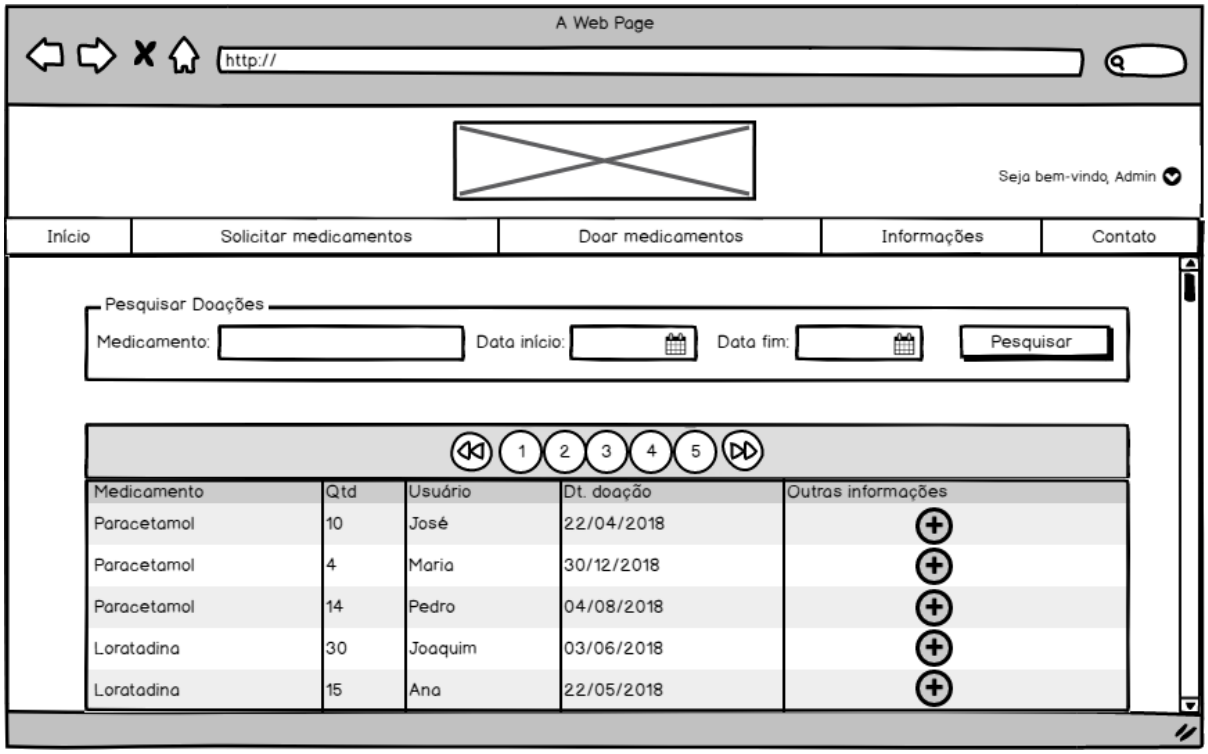
<Tela de cadastro de informação>



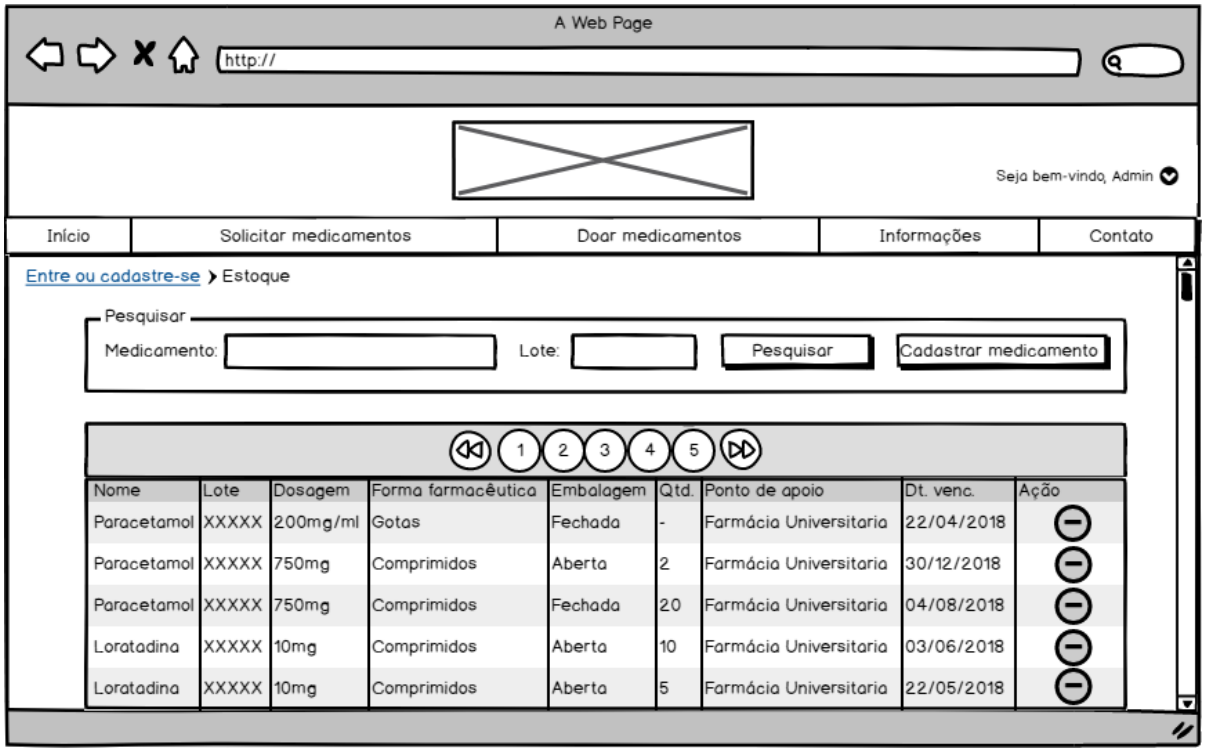
<Tela para visualizer solicitações>



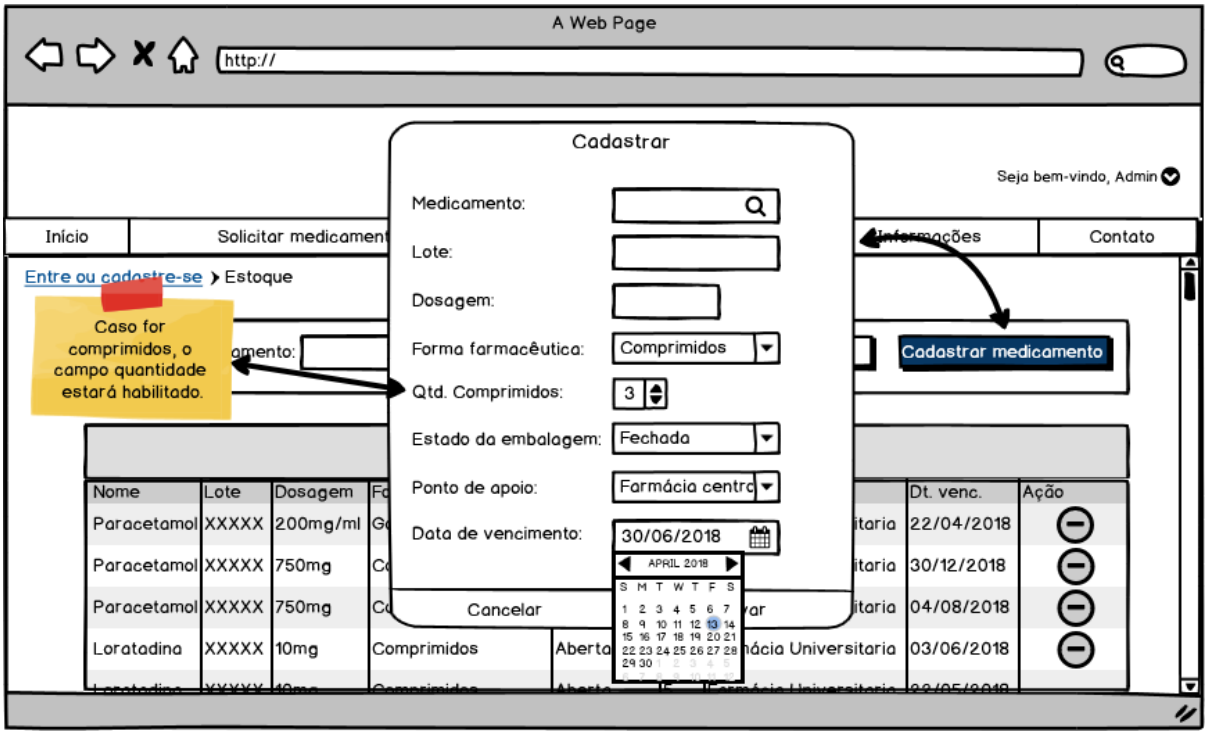
<Tela para visualizer doações>



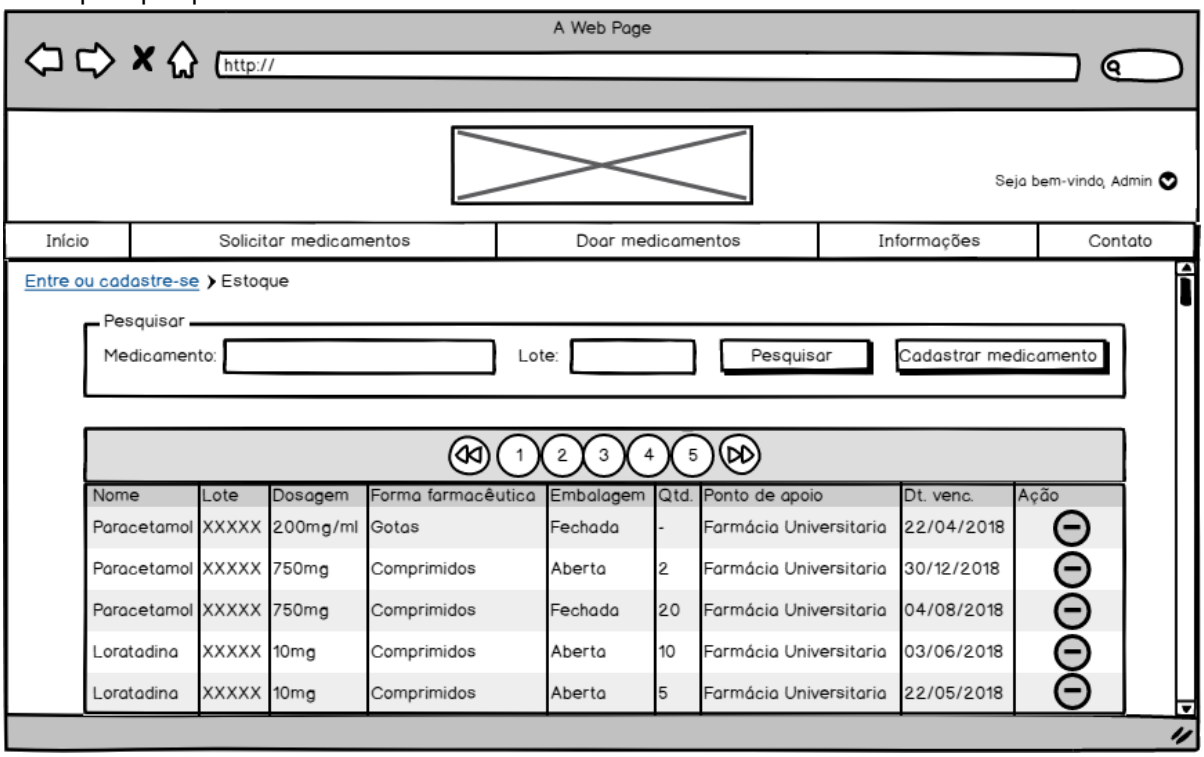
<Tela para pesquisar medicamento no estoque>



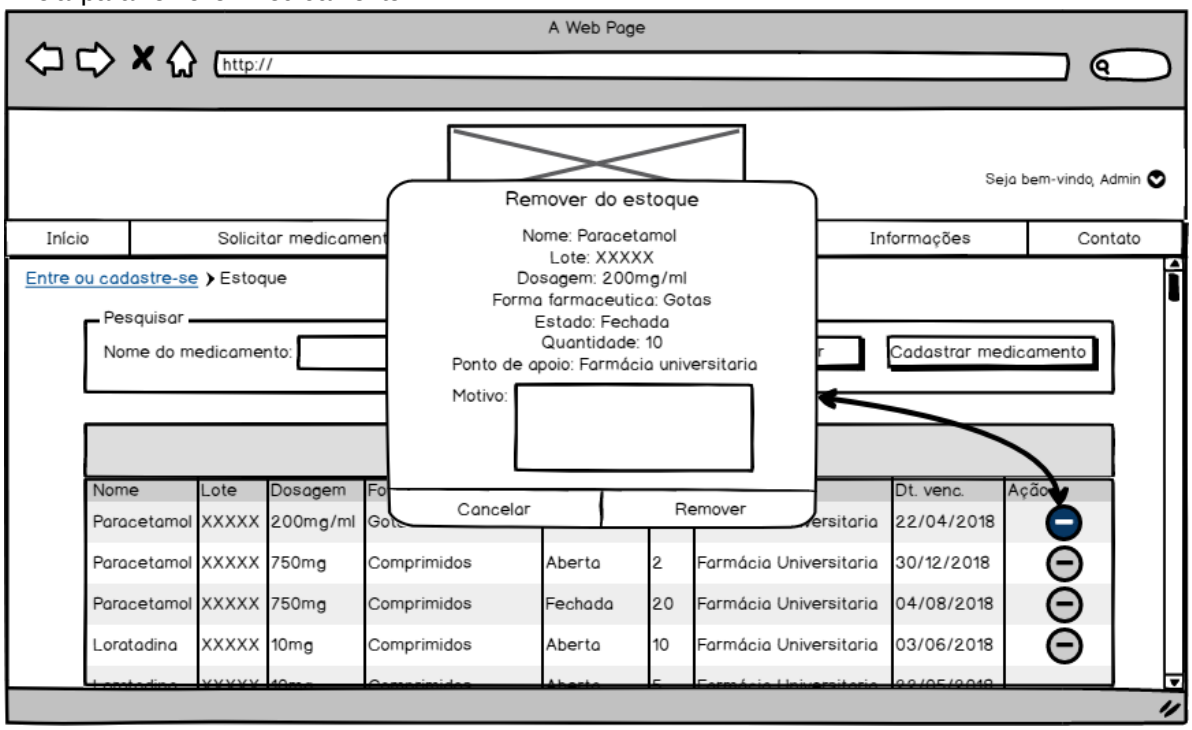
<Tela para cadastrar medicamento>



<Tela para pesquisar medicamento>



<Tela para remover medicamento>



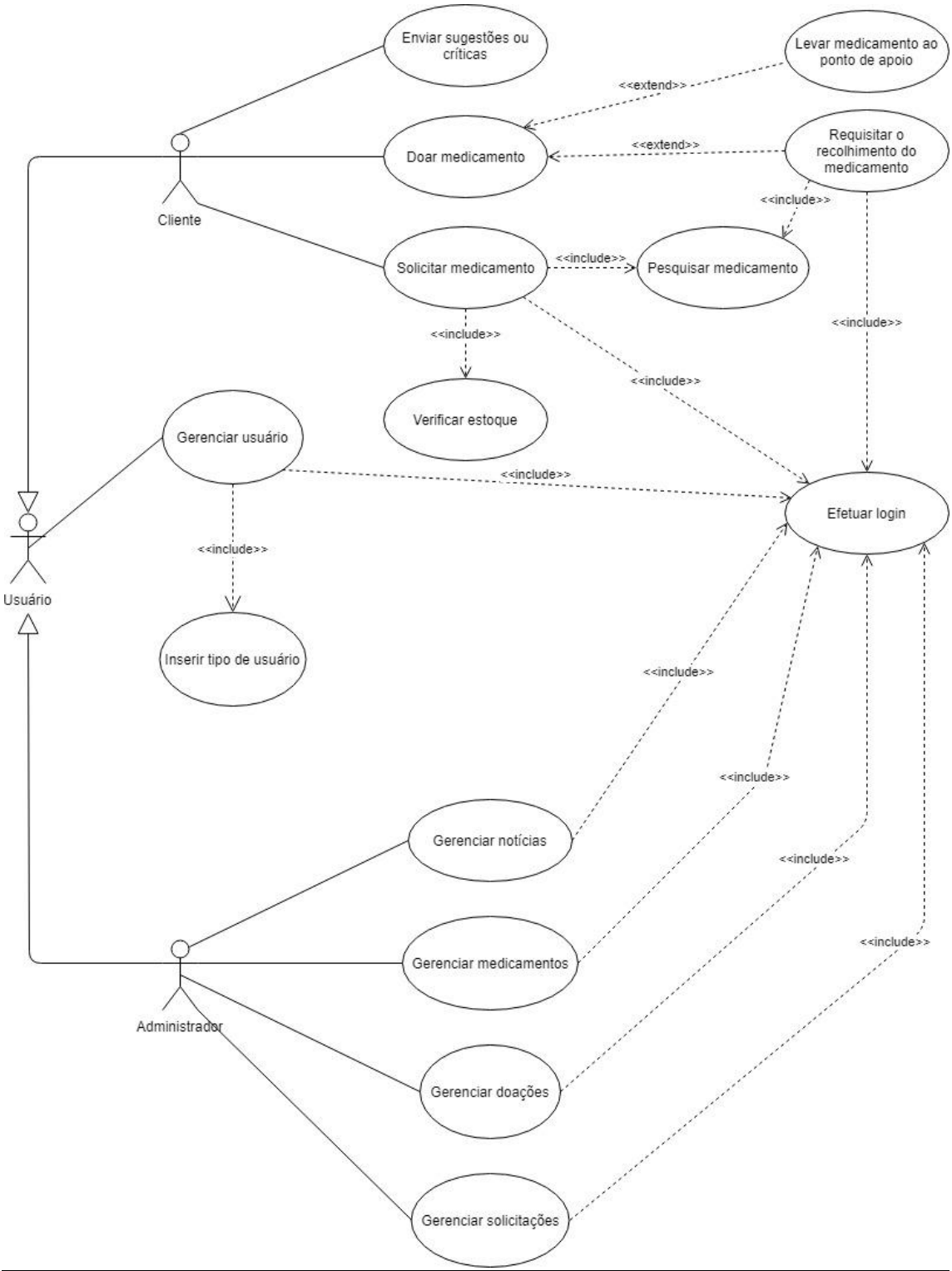
B Documento de Casos de Uso do Farmácia Solidária

UNIVERSIDADE FEDERAL DE JUIZ DE FORA

Casos de uso
Farmácia solidária

Maio 2018

Diagrama de caso de uso



Descrição dos casos de uso

UC01 – <Solicitar medicamentos>

Objetivo: Permitir que o usuário possa solicitar um medicamento, dentre a lista de disponíveis.

Casos de uso

Atores: Usuário

Condição de entrada: O ator deve entrar no menu "Solicitar medicamentos" e escolher dentre os sub-menus disponíveis (uso humano ou uso veterinário).

Fluxo Principal:

1. **O usuário pesquisa pelo medicamento (nome genérico ou comercial) no filtro;**
   {Listagem}
2. **O sistema lista os medicamentos encontrados pela pesquisa;**
3. **O usuário clica no botão "Outras informações";**
4. **O sistema abre uma modal contendo várias informações do medicamento e um campo "quantidade" para preenchimento do usuário (caso a forma farmacêutica do medicamento seja comprimidos);**
5. **O usuário preenche o campo "quantidade", caso esteja habilitado;**
   {Confirmação de envio}
6. **O usuário clica em "solicitar";**
   {Validação de dados}
7. **O sistema valida o login do usuário e a quantidade informada;**
8. **O sistema salva a solicitação;**
9. **O sistema abrirá uma tela contendo informações sobre o local e quais dias, o medicamento estará disponível para o usuário buscar.**

Fluxos Alternativos:

O sistema não retorna registros.
Em {Listagem}, se consulta não retornar registros que atendam o filtro informado:

1. **O sistema informa ao usuário com uma mensagem que a consulta não retornou registros.**

O usuário desiste da solicitação.
Em {Confirmação de envio}, se o usuário desistir:

1. **O usuário cancela a solicitação;**
2. **O sistema apaga o campo preenchido;**
3. **O sistema volta para a tela de pesquisa.**

O sistema não consegue validar campo obrigatório.
Em {Validação de dados}, se o usuário deixou de inserir o campo quantidade:

1. **O sistema informa que o campo é obrigatório;**
2. **O usuário preenche o campo;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

O sistema não valida login.
Em {Validação de dados}, se o usuário não está logado:

1. **O sistema abre a tela de login;**
2. **O usuário loga no sistema;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

O estoque não possui a quantidade solicitada.
Em {Validação de dados}, se a quantidade informada for maior do que o estoque disponível:

Casos de uso

1. **O sistema informa ao usuário com uma mensagem que a quantidade solicitada não está disponível.**

UC02 – <Doar de medicamentos>

Objetivo: Permite que o usuário doe medicamentos a farmácia solidaria.

Atores: Usuário

Condição de entrada: O ator deve entrar no menu “Doar medicamento”.

Fluxo Principal:

1. **O sistema exibe duas opções para o usuário escolher {Escolher entrega}**
2. **O usuário escolhe a opção “Busquem no endereço indicado”; {Validação de login}**
3. **O sistema valida o login do usuário;**
4. **O sistema exibirá campos para serem preenchidos;**
5. **O usuário preenche os campos do medicamento; {Escolher endereço};**
6. **O usuário escolhe um endereço, dentre os cadastrados em seu login;**
7. **O usuário preenche o dia e hora; {Confirmação de envio}.**
8. **O usuário clica em salvar; {Validação de campos}**
9. **O sistema valida os campos e salva a doação;**
10. **O sistema exibe mensagem de sucesso.**

Fluxos Alternativos: O usuário escolhe outra opção.
Em {Escolher entrega}, se o usuário escolhe a opção “Levarei até ao ponto de apoio”:

1. **O sistema exibe uma lista contendo todos os pontos de apoio disponíveis;**
2. **O usuário clica em um ponto de apoio;**
3. **O sistema exibe o endereço, horário de funcionamento e telefone.**

O sistema não valida login.
Em {Validação de login}, se o usuário não está logado:

1. **O sistema abre a tela de login;**
2. **O usuário loga no sistema;**
3. **O sistema retorna ao fluxo básico {Escolher opção}.**

O usuário não tem endereço cadastrado.
Em {Escolher endereço}, se o usuário não possui endereços cadastrados:

1. **O usuário clica em “cadastrar endereço”;**
2. **O sistema abre a tela para cadastrar endereço;**
3. **O usuário salva o endereço;**
4. **O sistema retorna ao fluxo básico {Escolher endereço}.**

O sistema não consegue validar campos obrigatórios.
Em {Validação de campos}, se o usuário deixou de inserir algum

Casos de uso

campo obrigatório:

1. **O sistema informa qual é o campo em branco;**
2. **O usuário preenche o campo;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

UC03 – <Visualizar informações>

Objetivo: Permite que o usuário visualize notícias e outras informações.

Atores: Usuário

Condição de entrada: O ator deve entrar no menu “Informações”.

Fluxo Principal:

1. **O sistema exibe uma lista contendo títulos de diversas notícias;**
2. **O usuário clica na notícia escolhida;**
3. **O sistema exibe o texto da notícia.**

Fluxos Alternativos: Não há.

UC04 – <Enviar sugestões ou críticas>

Objetivo: Permite que o usuário envie sugestões ou críticas para o administrador do sistema.

Atores: Usuário

Condição de entrada: O ator deve entrar no menu “Contato”.

Fluxo Principal:

1. **O sistema exibe um formulário para preenchimento;**
2. **Caso o usuário esteja logado, o sistema deverá trazer os campos nome, e-mail e telefone preenchidos;**
3. **O usuário escreve a mensagem;**
   {Confirmação de envio}
4. **O usuário clica em “enviar”;**
   {Validação de campos}
5. **O sistema valida os campos;**
6. **O sistema envia um email para o administrador com os dados do formulário;**
7. **O sistema exibe uma mensagem de sucesso;**

Fluxos Alternativos: O sistema não consegue validar campos obrigatórios.
Em {Validação de campos}, se o usuário deixou de inserir algum campo obrigatório:

1. **O sistema informa qual é o campo em branco;**
2. **O usuário preenche o campo;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

UC05 – <Efetuar login>

Objetivo: Permite que o usuário efetue login para poder navegar em algumas funcionalidades do sistema.

Atores: Usuário e Administrador

Condição de entrada: O ator deve clicar em "Entre ou cadastre-se" e já ser cadastrado no sistema.

Fluxo Principal:

1. **O sistema exibe um formulário contendo login e senha;**
2. **O usuário preenche os campos;**
   {Confirmação de envio}
3. **O usuário clica em "Entrar";**
   {Validação de dados}
4. **O sistema valida os campos;**
   {Verificação de usuário}
5. **Caso o usuário for um usuário comum, o sistema abrirá as seguintes opções: alterar dados, minhas solicitações e minhas doações.**

Fluxos Alternativos: O sistema não consegue validar campos.
Em {Validação de campos}, se o usuário digitou o login ou senha errados:

1. **O sistema informa o campo inserido erroneamente;**
2. **O usuário substitui o dado inválido;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

O usuário é do tipo administrador.
Em {Verificação de usuário}, se o usuário for do tipo administrador:

1. **O sistema abrirá um menu com as seguintes opções: solicitações, doações, estoque, cadastrar administradores e cadastrar informações.**

UC06 – <Cadastrar usuário comum>

Objetivo: Permite o cadastro de usuário comum.

Atores: Usuário

Condição de entrada: O ator deve entrar na tela de "login" e clicar no botão "Quero me cadastrar".

Fluxo Principal:

1. **O sistema exibe um formulário para preenchimento;**
   {Preencher dados cadastrais}
2. **O usuário preenche os campos;**
   {Confirmação de envio}
3. **O usuário clica em salvar;**
   {Validação de dados}
4. **O sistema valida os dados;**
5. **O sistema salva o usuário;**
6. **O sistema exibe uma mensagem de sucesso;**

Fluxos Alternativos: O usuário clica no botão “cadastrar endereço”.
Em {Preencher dados cadastrais}, se o usuário quiser cadastrar um endereço:

1. **O sistema abrirá uma modal com campos para preencher;**
2. **O usuário preenche os campos;**
3. **O usuário clica em “adicionar”;**
4. **O sistema exibe o endereço adicionado na tela de cadastrar usuário.**

O sistema não consegue validar campos obrigatórios.
Em {Validação de dados}, se o usuário deixou de inserir algum campo obrigatório:

1. **O sistema informa qual é o campo em branco;**
2. **O usuário preenche o campo;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

Os campos “senha” e “confirmar senha” são diferentes.
Em {Validação de dados}, se o usuário digitou caracteres diferentes nos campos “senha” e “confirmar senha”:

1. **O sistema informa que os campos estão divergindo;**
2. **O usuário preenche novamente os campos;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

UC07 – <Cadastrar usuário administrador>

Objetivo: Permite o cadastro de usuário administrador.

Atores: Administrador

Condição de entrada: O ator deve estar logado e clicar no menu “cadastrar admin”.

Fluxo Principal:

1. **O sistema exibe um formulário para preenchimento; {Preencher dados cadastrais}**
2. **O ator preenche os campos; {Confirmação de envio}**
3. **O ator clica em salvar; {Validação de dados}**
4. **O sistema valida os dados;**
5. **O sistema salva o usuário;**
6. **O sistema exibe uma mensagem de sucesso;**

Fluxos Alternativos: O sistema não consegue validar campos obrigatórios.
Em {Validação de dados}, se o usuário deixou de inserir algum campo obrigatório:

1. **O sistema informa qual é o campo em branco;**
2. **O usuário preenche o campo;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

Os campos “senha” e “confirmar senha” são diferentes.
Em {Validação de dados}, se o usuário digitou caracteres diferentes nos campos “senha” e “confirmar senha”:

Casos de uso

1. **O sistema informa que os campos estão divergindo;**
2. **O usuário preenche novamente os campos;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

UC08 – <Cadastrar informação>

Objetivo: Permite o cadastro de informação.

Atores: Administrador

Condição de entrada: O ator deve estar logado e clicar no menu “cadastrar informação”.

Fluxo Principal:

1. **O sistema exibe um formulário para preenchimento;**
2. **O ator preenche os campos;**
   {Confirmação de envio}
3. **O ator clica em salvar;**
   {Validação de dados}
4. **O sistema valida os dados;**
5. **O sistema salva a informação;**
6. **O sistema exibe uma mensagem de sucesso;**
7. **A informação cadastrada é exibida na listagem da tela do UC03 - <Visualizar informações>.**

Fluxos Alternativos: O sistema não consegue validar campos obrigatórios.
Em {Validação de dados}, se o usuário deixou de inserir algum campo obrigatório:

1. **O sistema informa qual é o campo em branco;**
2. **O usuário preenche o campo;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

UC09 – <Visualizar solicitações>

Objetivo: Permite a visualização de todas as solicitações feitas.

Atores: Administrador

Condição de entrada: O ator deve estar logado e clicar no menu “solicitações”.

Fluxo Principal:

1. **O sistema exibe uma tela de pesquisa;**
2. **O administrador pesquisa por algum campo do filtro;**
   {Listagem}
3. **O sistema lista as solicitações encontradas pela pesquisa;**

Fluxos Alternativos: O sistema não retorna registros.
Em {Listagem}, se consulta não retornar registros que atendam o filtro informado:

1. **O sistema informa ao usuário com uma mensagem que a consulta não retornou registros.**

UC10 – <Visualizar doações>

Objetivo: Permite a visualização de todas as doações salvas.

Atores: Administrador

Condição de entrada: O ator deve estar logado e clicar no menu “doações”.

Fluxo Principal:

1. **O sistema exibe uma tela de pesquisa;**
2. **O administrador pesquisa por algum campo do filtro; {Listagem}**
3. **O sistema lista as solicitações encontradas pela pesquisa;**

Fluxos Alternativos: O sistema não retorna registros.
Em {Listagem}, se consulta não retornar registros que atendam o filtro informado:

1. **O sistema informa ao usuário com uma mensagem que a consulta não retornou registros.**

UC11 – <Cadastrar medicamento no estoque>

Objetivo: Permite o administrador cadastre medicamentos disponíveis no estoque.

Atores: Administrador

Condição de entrada: O ator deve estar logado e clicar no menu “estoque”.

Fluxo Principal:

1. **O sistema abre a tela de pesquisar medicamentos;**
2. **O administrador clica no botão “cadastrar medicamento”;**
3. **O sistema abrirá uma modal com um formulário para preenchimento; {Confirmação de envio}**
4. **O usuário clica em salvar; {Validação de campos}**
5. **O sistema valida os campos;**
6. **O sistema salva o medicamento;**
7. **O sistema exibe uma mensagem de sucesso;**
8. **O medicamento passará a ser exibido na listagem da tela do UC01 – <Solicitar medicamento> e no UC11 - <Cadastrar medicamento>.**

Fluxos Alternativos: O usuário desiste do cadastro.
Em {Confirmação de envio}, se o usuário desistir:

1. **O usuário cancela o cadastro;**
2. **O sistema apaga os campos preenchidos;**
3. **O sistema volta para a tela de pesquisa.**

O sistema não consegue validar campos obrigatórios.
Em {Validação de campos}, se o usuário deixou de inserir algum campo obrigatório:

1. **O sistema informa qual é o campo em branco;**
2. **O usuário preenche o campo;**
3. **Retorna ao fluxo básico em {Confirmação de envio}.**

Casos de uso

UC12 – <Remover medicamento do estoque>

Objetivo: Permite o administrador remova medicamentos do estoque.

Atores: Administrador

Condição de entrada: O ator deve estar logado e clicar no menu “estoque”.

Fluxo Principal:

1. **O sistema abre a tela de pesquisar medicamentos;**
2. **O ator pesquisa pelo medicamento (nome genérico ou comercial) ou pelo lote no filtro;**
   {Listagem}
3. **O sistema lista os medicamentos encontrados pela pesquisa;**
4. **O ator clica no botão “Remover”;**
5. **O sistema abre uma modal contendo várias informações do medicamento e um campo “motivo” para preenchimento;**
6. **O usuário preenche o campo “motivo”;**
   {Confirmação de envio}
7. **O usuário clica em “remover”;**
   {Validação de campos}
8. **O sistema valida os campos;**
9. **O sistema faz uma exclusão lógica do medicamento (ou seja, o medicamento ainda estará no banco de dados, porém com status de “removido”);**
10. **O sistema exibe uma mensagem de sucesso.**

Fluxos Alternativos:

O sistema não retorna registros.
Em {Listagem}, se consulta não retornar registros que atendam o filtro informado:

1. **O sistema informa ao administrador com uma mensagem que a consulta não retornou registros.**

O administrador desiste da remoção.
Em {Confirmação de envio}, se o usuário desistir:

4. **O ator cancela a remoção;**
5. **O sistema apaga o campo preenchido;**
6. **O sistema volta para a tela de pesquisa.**

O sistema não consegue validar campos obrigatórios.
Em {Validação de campos}, se o administrador deixou de inserir algum campo obrigatório:

4. **O sistema informa qual é o campo em branco;**
5. **O usuário preenche o campo;**
6. **Retorna ao fluxo básico em {Confirmação de envio}.**