Desenvolvimento de um sistema de gestão de armazém para dispositivos móveis em Flutter

Eigon Noboru Ribeiro Kimura - 39048

Relatório Final de Estágio apresentado à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação no âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná.

Trabalho realizado sob a orientação de:

Prof. Doutor Paulo Alexandre Vara Alves

Prof. Doutor José Eduardo Moreira Fernandes

Marcin Włodarczyk

Bragança

2020

Desenvolvimento de um sistema de gestão de armazém para dispositivos móveis em Flutter

Eigon Noboru Ribeiro Kimura - 39048

Relatório Final de Estágio apresentado à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação no âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná.

Trabalho realizado sob a orientação de:
Prof. Doutor Paulo Alexandre Vara Alves
Prof. Doutor José Eduardo Moreira Fernandes
Marcin Włodarczyk

Bragança
2020

Agradecimentos

Primeiramente, agradeço à minha família que sempre esteve do meu lado e apoiou as minhas decisões, aos meus amigos que me deram grande motivação e forças todo o caminho da minha vida acadêmica.

Agradeço também aos meus orientadores, Professor Doutor José Eduardo Moreira Fernandes e o Professor Doutor Paulo Alexandre Vara Alves por aceitarem este estágio com a Riskivector e auxiliarem todo o desenvolvimento do projeto.

Agradeço ao Marcin Włodarczyk que me ajudou na concretização deste projeto, ao Vitor Laranjeira, fundador e CEO da empresa Riskivector, que me deu todo o apoio para tornar o estágio em uma ótima experiência e aos funcionários da Riskivector que me ajudou na etnografia.

Por último, quero agradecer também à Universidade Tecnológica Federal do Paraná por todo o conhecimento e oportunidades que me deram e ao Instituto Politécnico de Bragança que me abriu novas portas para minha vida acadêmica.

Resumo

A aplicação da tecnologia da informação no ambiente de trabalho otimiza a eficiência e agilidade das atividades da empresa. A motivação deste trabalho é auxiliar e melhorar o gerenciamento do armazém de utensílios da empresa Riskivector. O objetivo foi implementar um sistema de gestão de armazém que se adaptasse com a forma de trabalho e gerenciamento do armazém. O sistema é composto por uma aplicação para dispositivos móveis que é utilizado dentro do armazém e uma API REST que atende as solicitações do aplicativo através de operações de consulta e mudança no banco de dados próprio para gerenciar o armazém. As vantagens da aplicação para empresa é o aumento da precisão sobre as informações dos estoques e a melhora no controle de entrada e saída de utensílios no armazém. Para compreender todo o trabalho relacionado ao armazém, foi feito um estudo etnográfico e foram levantados vários requisitos funcionais, os quais foram mapeados para diagramas. O sistema visa ser utilizados em dispositivos móveis que utilizam o sistema operacional Android ou iOS. Portanto, optou-se por desenvolver a aplicação com o framework Flutter que possibilita gerar uma aplicação para ambas as plataformas com o mesmo código escrito na linguagem Dart. Como o sistema pode se tornar complexo, escolheu-se a arquitetura de software chamada Arquitetura Limpa para prover melhor manutenibilidade e testabilidade do sistema. Entre as funcionalidades identificadas nos requisitos funcionais, somente algumas funcionalidades consideradas essenciais foram implementadas como gerenciamento de utensílios, categorias destes utensílios, armazéns e registração de estoques de utensílios. Por outro lado, a arquitetura implementada facilita a extensibilidade das funcionalidades.

Palavras-chave: Sistema de Gestão de Armazém, Flutter.

Abstract

The application of information technology in the work environment optimizes the efficiency and agility of the company's activities. The purpose of this work is to assist and improve the management of the Riskivector utensils warehouse. The objective was to implement a warehouse management system that was adapted to the way of working and managing the warehouse. The system consists of an application for mobile devices that is used inside the warehouse and a REST API that meets the application's requests through query and change operations in the database itself to manage the warehouse. The advantages of the application for the company are the increase in the accuracy of the stock information and the improvement in the control of entry and exit of utensils in the warehouse. To understand all the work related to the warehouse, an ethnographic study was made and several functional requirements were raised, which were mapped to diagrams. The system aims to be used on mobile devices using the Android or iOS operating system. Therefore, it was decided to develop the application with the Flutter framework that makes it possible to generate an application for both platforms with the same code written in the Dart language. As the system can become complex, the software architecture called Clean Architecture was chosen to provide better maintainability and testability of the system. Among the functionalities identified in the functional requirements, only some functionalities considered essential have been implemented such as utensil management, categories of utensils, warehouses and registration of utensil stocks. On the other hand, the implemented architecture facilitates the extensibility of the functionalities.

Keywords: Warehouse Management System, Flutter.

Conteúdo

1 Introdução **1**

1.1 A empresa 1

1.1.1 Kit de utensílios 2

1.2 Problema e Desafios 2

1.3 Objetivos 2

1.4 Estrutura do Documento 2

2 Revisão da Literatura **5**

2.1 Arquitetura de Software 5

2.1.1 Arquitetura de camadas 6

2.1.2 Arquitetura de portas e adaptadores 6

2.1.3 Arquitetura Limpa 9

2.2 Tecnologias de gerenciamento de desenvolvimento 11

2.2.1 Git 11

2.2.2 Gitlab 11

2.2.3 Docker 12

2.3 Tecnologias para desenvolvimento do aplicativo 12

2.3.1 Android 12

2.3.2 Java 13

2.3.3 Kotlin 13

2.3.4 iOS 14

2.3.5 Objective-C 14
2.3.6 Swift 14
2.3.7 Flutter 15
2.3.8 Dart 17
2.4 Tecnologias para desenvolvimento da API REST 18
2.4.1 TypeScript 18
2.4.2 Node.js 18
2.4.3 Express 19
2.4.4 MySQL 19
2.4.5 ORM 19
2.4.6 Sequelize 19

3 Análise de Requisitos e Modelação 21
3.1 Proposta de Solução 21
3.2 Mapeamento das atividades do armazém 21
3.2.1 Trabalho com kit de utensílios 22
3.2.2 Preparo de apartamentos 27
3.2.3 Compra de produtos e utensílios para o armazém 30
3.2.4 Armazenamento de bagagens dos clientes 31
3.3 Modelação do Projeto 32
3.3.1 Requisitos 32
3.3.2 Diagrama de caso de uso 37
3.3.3 Diagrama de Entidade Relacionamento 39
3.3.4 Diagrama de Domínio 40
3.4 Arquitetura do Sistema 41
3.5 Representação de Árvore com Banco de Dados Relacional 42
3.5.1 Modelo de lista de adjacências 42
3.5.2 Modelo de conjunto aninhado 43

4 Desenvolvimento e Implementação **45**

4.1 Desenvolvimento da API 45

4.1.1 Implementação da camada de interface de adaptadores 46

4.1.2 Implementação da camada de regras de negócio da aplicação 49

4.1.3 Implementação da camada de regras de negócio da empresa 49

4.1.4 Implementação da camada de frameworks e drivers 52

4.1.5 Implementação da Estrutura Árvore com Banco de Dados Relacional 52

4.2 Desenvolvimento do Aplicativo 53

4.2.1 Funcionalidades do aplicativo 54

5 Conclusões e Trabalhos Futuros **81**

Lista de Tabelas

3.1 Exemplo da tabela “Categoria” no modelo de lista de adjacências 43
3.2 Exemplo da tabela “Categoria” no modelo de conjunto aninhado 44

Lista de Figuras

2.1 Arquitetura hexagonal com adaptadores, adaptada da fonte: Alistair Cockburn (2005). 8
2.2 Arquitetura limpa. Adaptada. Fonte: Martin (2018) 9

3.1 Fluxo geral do trabalho com kit de utensílios 22
3.2 Fluxo de trabalho da coleta de itens 23
3.3 Fluxo de trabalho da montagem do kit 24
3.4 Fluxo de trabalho da entrega do kit 24
3.5 Fluxo de trabalho da confirmação de entrega do kit 25
3.6 Fluxo de trabalho para recolher o kit 26
3.7 Fluxo de trabalho para restaurar os utensílios recolhidos 27
3.8 Fluxo geral para preparar um apartamento 28
3.9 Fluxo de trabalho para verificar o que precisa 28
3.10 Fluxo de trabalho para preparar kit de cozinha 29
3.11 Fluxo de trabalho para preparar móveis 29
3.12 Fluxo de trabalho para recolher os móveis e o kit de cozinha 30
3.13 Fluxo de trabalho para compra de produtos e utensílios 31
3.14 Fluxo de trabalho para armazenar as bagagens dos clientes 31
3.15 Fluxo de trabalho para retirar as bagagens dos clientes 32
3.16 Casos de uso do administrador do armazém 37
3.17 Casos de uso do mantedor do armazém parte 01/02 38
3.18 Casos de uso do mantedor do armazém parte 02/02 39

3.19 Diagrama de entidade relacionamento 40
3.20 Diagrama de domínio do sistema 41
3.21 Mapa do sistema de gerenciamento de armazém 42
3.22 Ideia do modelo de conjunto aninhado 44

4.1 Arquitetura limpa. Adaptada. Fonte: Martin (2018) 46
4.2 Relação entre as classes através de um caso de uso que “cria uma categoria” 48
4.3 Ideia do modelo de conjunto aninhado 51
4.4 Tabela “Category” criada no MySQL 53
4.5 Tela de login 54
4.6 Tela de login em um armazém 55
4.7 Tela de menu 56
4.8 Tela de catálogo de itens 57
4.9 Feedbacks da tela de catálogo de itens 58
4.10 Menu de ações da tela de catálogo de itens 59
4.11 Tela de criação de subcategoria 59
4.12 Menu de ações da tela de catálogo de itens 60
4.13 Primeira etapa de deleção de categoria 61
4.14 Segunda etapa de deleção de categoria 62
4.15 Terceira etapa de deleção de categoria 63
4.16 Tela de escolha ou mudança de categoria 64
4.17 Menu de ações da tela de seleção de categoria 65
4.18 Tela de criar item 66
4.19 Telas de detalhes de um item 67
4.20 Menu de ações da tela de detalhes de um item 67
4.21 Tela de excluir um item 68
4.22 Tela de editar item 69
4.23 Tela de lista de armazéns 70
4.24 Tela de criação de um armazém 71

4.25 Menu de ações da tela de criação de um armazém 72
4.26 Tela de editar um armazém 73
4.27 Tela de excluir um armazém 74
4.28 Etapa de escolher um armazém 75
4.29 Etapa de escolher um item 76
4.30 Etapa de inserir informações do estoque 77
4.31 Etapa de confirmação das informações escolhidas e inseridas 78
4.32 Telas de erros 79

Siglas

API Application Programming Interface. 11, 12, 18, 21, 25, 41, 45, 47, 49, 52, 81, 82

ARM Advanced RISC Machine. 15

BCE Boundary-Control-Entity. 9

CRUD Create, Read, Update and Delete. 47

DCI Data, Context, and Interaction. 9

DTO Data Transfer Object. 46–48

DVCS Distributed Version Control System. 11

FPS Frames per Second. 15

HTTP HyperText Transfer Protocol. 47

IPB Instituto Politécnico de Bragança. 1

JSON JavaScript Object Notation. 7

ORM Object Relational Mapping. 19, 47, 52, 81

OSS Open Source Software. 13

RDBMS Relational Database Management System. 19, 81

RDBS Relational Database System. 19

SQL Structured Query Language. 19, 43

UUID Universally Unique Identifier. 49, 51

Capítulo 1

Introdução

O Capítulo 1 é dedicado à apresentação da empresa que propôs este projeto, a abordagem do problema e sua importância, os objetivos do trabalho e estrutura do relatório.

1.1 A empresa

A Riskivector é uma empresa do ramo imobiliário fundada em 23 de Fevereiro de 2009 e incubada no Instituto Politécnico de Bragança (IPB) em 2010. Sua sede está em Bragança e possui escritórios em Vila Real e Mirandela. Sua principal atividade é o aluguel de imóveis ou parte destes para os clientes e toda manutenção do imóvel e gestão dos pagamentos fica por conta da Riskivector. Este serviço traz benefícios tanto para o dono do imóvel como para os inquilinos. Para o dono do imóvel não há preocupação em procurar e relacionar com os inquilinos. Para os inquilinos, não há preocupação em dividir as contas, realizar contratos ou negociações com o dono do imóvel e provedoras de Internet, procurar por novas pessoas para dividir o apartamento caso um morador saia, etc. Além do serviço imobiliário, a Riskivector oferece manutenção gerais, apoio de reservas de alojamento e alugamento de kit de utilitários.

1.1.1 Kit de utensílios

O kit de utensílios é um conjunto de itens essenciais para iniciar uma morada. Grande parte dos clientes que alugam um quarto com a Riskivector, são pessoas que ficam um curto período em Bragança. Um dos trabalhos que estes clientes têm são a compra de utensílios como talheres, copos, panelas, pratos, etc. e a revenda destes após o término da estada. Para evitar este trabalho, a Riskivector aluga este kit de utensílios para os clientes.

1.2 Problema e Desafios

Toda a gestão do armazém é feita manualmente. Um dos principais trabalhos é a montagem de kits de utensílios que são entregue ao cliente. Por isso, no armazém ocorrem várias vezes a entrada e saída de utensílios. No final do dia é feita a contagem dos utensílios que saíram e sobraram para ter um controle dos estoques e estimativas de quantos kits poderão ser feitos para o dia seguinte. Caso perceba-se que faltará utensílios para os kits, é feito a compra destes por um funcionário. Diante disso, o sistema poderia diminuir certos trabalhos e auxiliar outros, provendo uma eficiência e agilidade no trabalho do armazém.

1.3 Objetivos

O objetivo deste estágio é implementar um sistema de gerenciamento de armazém para gerenciar os produtos que formam os *kits* de utensílios da Riskivector. Espera-se que o sistema contribua para melhora da gestão do armazém, provendo precisão sobre as informações dos estoques do armazém e melhor controle de entrada e saída dos utensílios.

1.4 Estrutura do Documento

Este relatório está organizado da seguinte forma: no Capítulo 2 é apresentado a revisão da literatura, descrevendo as tecnologias e conceitos relacionados ao projeto; o Capítulo 3

apresenta as análises de requisito e modelação; o Capítulo 4 demonstra o desenvolvimento do projeto; e por fim, o Capítulo 5 apresenta a conclusão e os trabalhos futuros.

Capítulo 2

Revisão da Literatura

Este capítulo apresenta os conceitos e tecnologias relacionadas ao projeto, com o intuito de auxiliar a compreensão.

2.1 Arquitetura de Software

Nesta seção é explicados sobre os conceitos relacionados à arquitetura de software, principalmente sobre a Arquitetura Limpa, a qual foi escolhida para este projeto.

Segundo Shaw and Garlan (1996) e Martin (2018), pode-se dizer que a arquitetura de software é um conjunto de descrições sobre:

- Divisão do sistema em componentes;
- Comunicação e interação entre os componentes;
- Organização dos componentes;
- Padrões de composição e restrições sobre os padrões;

O objetivo da arquitetura em um sistema é facilitar o desenvolvimento, implantação, operação e manutenção do sistema, para que se tenha muitas opções de melhoria e extensão do sistema por um maior tempo possível Martin (2018).

Este projeto escolheu utilizar a "Arquitetura Limpa" de Martin (2018), pois o potencial de crescimento do sistema, do ponto de vista da empresa, é elevado. Portanto, sua manutenibilidade e testabilidade deve ser facilitada. Outro motivo que contribuiu para a escolha é a grande quantidade de exemplos de projetos que utilizam esta arquitetura. A seguir estão algumas das arquiteturas que estão relacionadas com a Arquitetura Limpa.

2.1.1 Arquitetura de camadas

É uma arquitetura que divide o software em várias camadas com funções e responsabilidades específicas. Não há um determinado número de camadas, porém, é comum a representação com quatro camadas: apresentação, negócios, persistência e banco de dados. A seguir estão os conceitos chave desta arquitetura.

- **Camada fechada**: Quando uma camada é fechada, significa que ela só pode comunicar somente com as camadas adjacentes.
- **Camada aberta**: É igual a camada fechada mas as camadas adjacentes podem comunicar entre elas, pulando a camada aberta.
- **Isolamento de camadas**: As modificações feitas em uma camada deve impactar pouco ou nada sobre outras camadas. Esta restrição reduz ou evita os impactos de modificações. Se permitisse a comunicação com outras camadas, tornaria difícil e custoso fazer mudanças no software. Este conceito significa também que uma camada precisa conhecer pouco ou nada sobre as outras camadas.

2.1.2 Arquitetura de portas e adaptadores

É uma arquitetura de software elaborada por Cockburn (2005) e também é chamada de "Arquitetura Hexagonal". A ideia desta arquitetura é construir um software que consiga ter o mesmo comportamento independente dos utilizadores, programas e testes automáticos. O software também deve ser desacoplado dos dispositivos e banco de dados que

são utilizados na produção. Resumidamente, a Arquitetura Hexagonal tem como objetivo construir um software que consiga ser executado mesmo estando totalmente isolado (Cockburn, 2005).

O significado de **Portas** possui a mesma ideia de "portas" de protocolo das redes de computadores ou "portas" para dispositivos de computadores. É uma entrada de dados externos e saída para os dados processados pela aplicação.

Os **Adaptadores** vão converter os dados recebidos pela porta em formatos mais adequados para o sistema. Por exemplo, se os dados vieram em formato JavaScript Object Notation (JSON), o adaptador converterá para uma instância de uma classe. Os adaptadores normalmente estão ligados a uma porta, e uma porta pode ter vários adaptadores.

A "fronteira de casos de uso" indica que os casos de uso devem ser escritos dentro do aplicativo e não depender das tecnologias externas. A Figura 2.1 ilustra a Arquitetura Hexagonal.

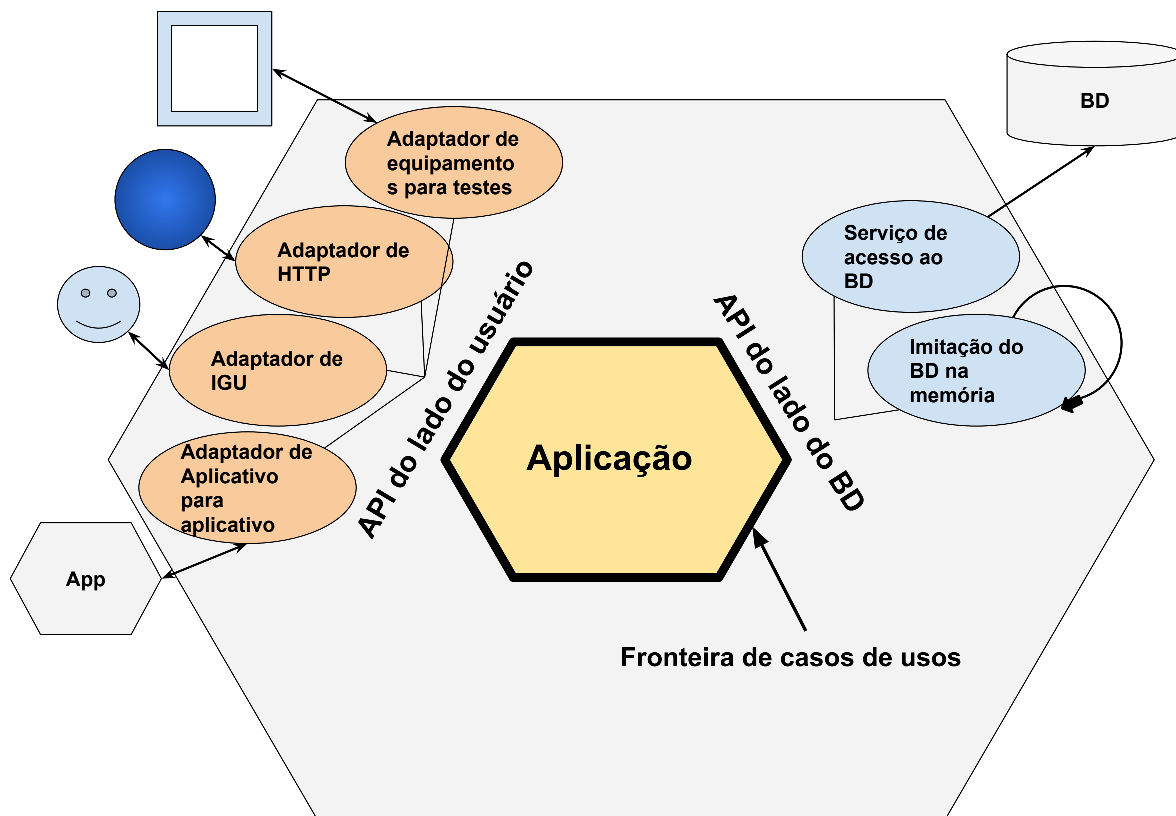


Figura 2.1: Arquitetura hexagonal com adaptadores, adaptada da fonte: Alistair Cockburn (2005).

Como mencionado anteriormente, esta arquitetura possui dois nomes: **"Portas e Adaptadores"** e **"Hexagonal"**.

A "Portas e Adaptadores" deixa explícito os componentes elementares desta arquitetura.

A nome "Hexagonal" traz a ideia de 6 camadas ou lados. Normalmente a arquitetura de camadas possui entre 3 a 4 camadas por ser o necessário. Com base nisso, a arquitetura Hexagonal possui camadas de sobra, o que possibilita inserir mais adaptadores caso precise. O que dá uma ideia de extensibilidade do sistema. Vale ressaltar que tanto a arquitetura de camadas como a hexagonal, não obriga ter um número fixo de camadas. Pode ter uma arquitetura de camadas com 2 ou 6 camadas como uma arquitetura hexagonal com 3 ou 7 camadas. Não há uma regra e sim uma ideia.

2.1.3 Arquitetura Limpa

A Arquitetura Limpa foi elaborada por Martin (2018) que se baseou em arquiteturas que possuem alta **independência** e **testabilidade** como características em comum. Segundo Martin (2018) essas arquiteturas são:

- arquitetura hexagonal de Cockburn (2005);
- Data, Context, and Interaction (DCI) de Trygve Reenskaug e Coplien and Bjørnvig (2010);
- Boundary-Control-Entity (BCE) de Jacobson et al. (1992).

A "independência" significa não depender de frameworks, interface do usuário, banco de dados e agentes externos.

A "testabilidade" significa o quão fácil é testar algo como funções, componentes, telas, funcionalidades, etc. Esta característica é uma consequência benéfica da independência (Martin, 2018), dado que é possível substituir os componentes por outros que o software executará. Com isso, pode-se colocar ou trocar por componentes que simulam o ambiente real, chamados de mock (imitação). A Figura 2.2 ilustra a Arquitetura Limpa.

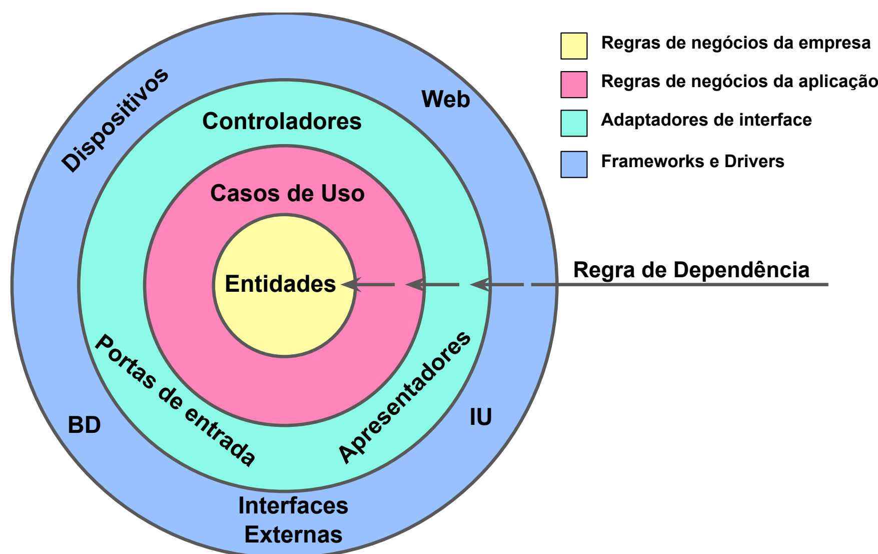


Figura 2.2: Arquitetura limpa. Adaptada. Fonte: Martin (2018)

A Arquitetura Limpa é composta por 4 camadas básicas: Frameworks e drivers, adaptadores de interfaces, casos de uso e entidades. Porém, Martin (2018) diz que a arquitetura permite adotar mais camadas de acordo com a necessidade. O mais importante é obedecer a regra de dependência. A seguir serão explicados os conceitos que estão relacionados à Arquitetura Limpa.

Regra de dependência

No centro da Figura 2.2 há uma seta pontilhada que indica a direção que um componente pode depender. Por exemplo, o componente que representa a web pode depender do controlador. O controlador pode depender do caso de uso, que por sua vez pode depender da entidade. Mas nunca o contrário. Os componentes mais interiores da arquitetura não devem depender dos componentes mais externos.

Entidades

Entidades encapsulam as regras de negócios da empresa. Podem ser um objeto que possui um método ou conjunto de estruturas de dados e funções. Como as entidades estão no centro da arquitetura, não devem sofrer efeitos colaterais devido as alterações feitas nas camadas mais externas. Exemplos de mudanças seria: mudança no banco de dados, forma de paginação, segurança, etc.

Casos de uso

Casos de uso encapsulam regras de negócios em nível de projeto que normalmente são mais complexas. Se após criar um usuário é preciso enviar um e-mail para o usuário, toda esta lógica está dentro do caso de uso.

Adaptadores

Possuem a mesma ideia da arquitetura Hexagonal 2.1.2. Os adaptadores são responsáveis por converter os dados em formatos úteis para os outros componentes.

Frameworks e drivers

Nesta camada é realizada a comunicação com os agentes externos como web frameworks, Application Programming Interface (API), banco de dados, etc.

2.2 Tecnologias de gerenciamento de desenvolvimento

Este projeto utilizou ferramentas para facilitar o gerenciamento e replicação do projeto em outros ambientes. A seguir estão descritos sobre estas tecnologias.

2.2.1 Git

Segundo Ingeno (2018), o software precisa de versões que podem ser números ou um nome exclusivo. Isto ajuda o software ter um estado definido e fornecer conhecimento sobre este estado às pessoas que estão relacionadas com o software. Exemplos de conhecimento são as mudanças realizadas, andamento de funcionalidades, dependências de bibliotecas ou softwares, etc.

Com base na importância do software possuir versões, uma das necessidades que surge é o controle de versões. Os sistemas de controle de versão registram alterações em um ou vários arquivos ao longo de sua vida com o intuito de poder recuperar versões específicas mais tarde (Chacon and Straub, 2014).

O Git é um Distributed Version Control System (DVCS) desenvolvido por Linus Torvalds em 2005. Os usuários podem fazer cópias completas de um projeto incluindo até o histórico de modificações e distribuí-las. Isto aumenta a disponibilidade e a capacidade de restaurar o projeto (Chacon and Straub, 2014).

2.2.2 Gitlab

O GitLab é uma plataforma online baseada em Git criada em 2011 à 2012 por Dmitriy Zaporozhets e Valery Sizov (GitLab, 2020b). O GitLab permite o desenvolvimento completo do software. Pode-se criar projetos, hospedar os códigos, rastrear problemas,

colaborar com o código, testar e implantar continuamente o software. Provê e facilita o desenvolvimento colaborativo através de muitas ferramentas e funcionalidades integradas (GitLab, 2020a).

2.2.3 Docker

Docker é um software de código aberto lançado em março de 2013 para encapsular, executar e distribuir um programa ou um ambiente necessário para executar um aplicativo. Uma analogia para o Docker é um provedor de logística de software que economizará tempo na construção de ambientes de desenvolvimento e produção, permitindo que os desenvolvedores de software concentrem nos principais trabalhos (Nickoloff and Kuenzli, 2019).

O Docker utiliza uma tecnologia chamada contêiner que é comparada com as máquinas virtuais. Uma das diferenças é que o contêiner compartilha recursos com o sistema operacional principal. Esta capacidade evita a sobrecarga ao sistema operacional principal e torna o contêiner mais leve e eficiente comparado com a máquina virtual.

Mesmo comparado com os aplicativos que são executados diretamente no sistema operacional principal, possuem pouco ou nenhuma sobrecarga. A leveza dos contêineres exige menos capacidade do computador, possibilita executar vários contêineres e facilita a portabilidade destes (Mouat, 2015).

2.3 Tecnologias para desenvolvimento do aplicativo

O projeto foi desenvolvido em duas partes intercaladamente: o aplicativo e a API. Nesta subseção serão apresentadas as tecnologias relacionadas ao aplicativo.

2.3.1 Android

O Android é um conjunto de software de código aberto composto por um sistema operacional, interface do usuário e aplicativos fundamentais para executar dispositivos móveis

(Rubin, 2007). Em uma visão mais abrangente, o Android é composto por documentações sobre compatibilidade, um Kernel do sistema operacional Linux e bibliotecas de código aberto para desenvolvimento de aplicativos (Meier and Lake, 2018). O desenvolvimento nativo de aplicações para Android é feito em linguagens de programação Java e Kotlin.

2.3.2 Java

Segundo Deitel and Deitel (2015), Java é uma linguagem de programação baseada em C++ que resultou de um projeto financiado pela Sun Microsystems e liderado por James Gosling em 1991. A Java tem como um dos principais objetivos escrever programas que podem ser executados em grande parte dos sistemas e dispositivos computacionais. Segundo Deitel et al. (2014), Java é uma das linguagens de programação mais usadas no mundo e isso é um dos motivo que contribuiu para a linguagem ser escolhida para desenvolver aplicativos para a plataforma Android, além de ser poderoso, gratuito e de código aberto.

2.3.3 Kotlin

Kotlin é uma linguagem de programação Open Source Software (OSS) desenvolvida pela JetBrains que teve sua primeira versão lançada em 2016. Uma das características da Kotlin é a interoperabilidade com o código Java, podendo ser usado na maioria dos cenários onde Java está sendo aplicado. Além disso, esta linguagem é concisa e segura Jemerov and Isakova (2017). Uma característica da linguagem que torna mais segura, é o rastreio de variáveis que não podem ser nulas e proibir que estas variáveis realizem operações que podem gerar uma exceção de ponteiro nulo (NullPointerException). Segundo Jemerov and Isakova (2017), a Google anunciou que Kotlin seria oficialmente suportada para sistemas operacional Android. Isso torna a Koltin uma linguagem alternativa para desenvolver aplicativos nativos para a plataforma Android.

2.3.4 iOS

O iOS é o sistema operacional móvel desenvolvido pela Apple. É um sistema operacional proprietário e seu gerenciamento de memória é muito rigoroso. Os programadores precisam adaptar às restrições sobre a memória. Somente os aplicativos podem ter acesso a todas as funcionalidades do sistema operacional. Logo, os aplicativos que são publicados na App Store são avaliados minuciosamente para estarem dentro dos padrões da Apple (Levin, 2015). O desenvolvimento nativo de aplicações para iOS é feito em linguagens de programação Objective-C e Swift.

2.3.5 Objective-C

Segundo Kochan (2011) Objective-C é uma linguagem de programação criada no início dos anos 80 por Brad J. Cox, que adicionou extensões à linguagem C para poder desenvolver softwares com programação orientada à objeto. A inspiração da orientação à objeto foi pela linguagem de programação SmallTalk-80. Mais tarde ela se torna a linguagem de programação base e oficial do OS X e para desenvolvimento de aplicações para as plataformas da empresa Apple.

2.3.6 Swift

Swift é uma linguagem de programação desenvolvida pela Apple para substituir as linguagens baseadas em C, inclusive o Objective-C, incorporando funcionalidades e ideias de uma linguagem moderna. Uma dessas funcionalidades é o tipo Optional que força o programador a tratar o valor nulo, evitando possíveis problemas futuros devido ao valor nulo inesperado. Caso não utilize o tipo Optional, o compilador não aceita variáveis com o valor nulo.

2.3.7 Flutter

Flutter é um framework que possibilita criar aplicativos nativos para dispositivos Android e iOS programando na linguagem Dart. O Flutter compila código Advanced RISC Machine (ARM) nativo para as duas plataformas, que resulta em um aplicativo com alto desempenho (Napoli, 2019) (Clow, 2019).

A renderização do Flutter executa a 60 Frames per Second (FPS) e pode chegar a 120 FPS em dispositivos que suportam essa frequência. Flutter usa o motor de renderização chamada Skia 2D. Este pode trabalhar com diferentes tipos de hardware e software. Além de ser usado nos produtos da Google, é também usado no Mozilla FireFox, FireFoxOS e outros. Ele também é capaz de compilar as telas em tempo real, agilizando o desenvolvimento de aplicativos (Napoli, 2019).

Vantagens e desvantagens

As vantagens de utilizar o Flutter são:

- criar aplicativos nativos para dispositivos Android e iOS apenas com código Dart;
- tempo de compilação rápido;
- devido a ideia de widgets (componentes), há uma grande capacidade de reutilização de códigos;
- biblioteca de testes para seus widgets (componentes).

A desvantagem de utilizar o Flutter é aprender sobre a linguagem Dart. Ela não é considerada complicada mas é um trabalho que se pode considerar.

Conceitos relacionados ao Flutter

O Flutter trabalha com a ideia de widgets que podem ser visto como componentes. Uma lista, um botão, uma tela, um texto e até mesmo uma animação é considerado como

widget, que é uma classe em Dart que sabe como descrever sua exibição (Windmill, 2019). No Flutter há dois tipos de widgets: com estado e sem estado.

- **Widgets sem estado**: são os que não precisam armazenar informações ou mudar seu estado durante sua vida útil. Todos os dados que ele precisa exibir, são passados no momento de instanciar. Sua vida depende dos widgets que estão usando-o como parte. Um exemplo de widget sem estado é um simples botão e o widget que sua vida dependerá, seria uma tela que possui este botão. Se a tela for descartada, o botão também será descartado.

- **Widgets com estado**: além de ter um estado, eles possuem um ciclo de vida. Um exemplo de widget com estado poderia ser um contador, que deverá alterar seu valor após ser incrementado ou decrementado pelo usuário. Seu ciclo de vida, de forma simplificada, pode ser:

  1. ao navegar para tela que contém o contador, o "objeto contador" é criado;
  2. o "objeto contador" inicializa seu estado tendo o valor inicial como 0;
  3. o "widget contador" é criado pelo Flutter;
  4. se o usuário realizar uma ação que incrementa o contador, o "objeto contador" chama uma função para atualizar seu valor (no caso de 0 para 1);
     (a) o Flutter reconstruirá o "widget contador", com um novo valor (neste caso 1);
  5. se o usuário fechar a tela que se encontra o contador, o "widget contador" será descartado.

Funcionamento

O Flutter gera uma árvore de widgets que possuem informações para renderizar na tela.

O Flutter percorre esta árvore da raiz para as folhas em tempo linear, coletando as restrições de posição e tamanho que cada widget possui. Estas restrições são ditas pelos widget parente.

Após coletar as restrições, dessa vez é feito o caminho inverso, para definir os valores reais sobre a renderização (como por exemplo, cores, tamanhos e posições) de cada widget.

Em outras palavras, primeiro é feito as restrições e depois a definições dos valores que darão forma aos widgets na tela.

Finalmente, Flutter fornece aos widgets as coordenadas reais na tela e diz ao sistema operacional o que deve ser exibido na tela.

O Flutter foi escolhido para o desenvolvimento da aplicação devido a capacidade de criar aplicativo nativo para Android e iOS. Isso evita o trabalho de manter dois códigos distintos e o custo de ter desenvolvedores para cada plataforma.

2.3.8 Dart

Dart é uma linguagem de programação de código aberto criada pela Google. Segundo Walrath and Ladd (2012), a motivação para criar o Dart foram as necessidades como:

- Desenvolver aplicações web complexas de forma mais simples e rápida;
- Carregar aplicações rapidamente;
- Executar suavemente para prover uma boa experiência aos usuários;

Dart é uma linguagem puramente orientada a objeto (Bracha, 2016) e é possível executá-la nos navegadores e na linha de comando, permitindo ser utilizada tanto no lado do cliente como no servidor. Pode ser compilada para JavaScript e chamar suas bibliotecas (Buckett, 2013).

Dart por ser elaborada para facilitar o desenvolvimento de aplicações web, possui muitas bibliotecas principais. Biblioteca para matemática, programação assíncrona, codificação e descodificação de dados, manipulação de datas, soquetes, programação concorrente, programação web, etc. (Walrath and Ladd, 2012).

2.4 Tecnologias para desenvolvimento da API REST

A API fará a comunicação intermediária entre o banco de dados e o aplicativo. A seguir serão abordados as tecnologias utilizadas na API.

JavaScript

O JavaScript é uma linguagem de que inicialmente foi criada com o propósito para programação web. É uma linguagem interpretada de alto nível, dinâmica e sem tipagem com capacidade para programação orientada a objeto e programação funcional (Flanagan, 2011). Por estar associada aos navegadores no seu início, é uma das linguagens mais populares do mundo (Crockford, 2008).

2.4.1 TypeScript

TypeScript é uma linguagem de programação que fornece funcionalidades complementares do JavaScript. Uma das principais funcionalidades do TypeScript é a verificação estática de tipos. O sistema de tipo do TypeScript permite restringir os acessos a estruturas de dados, diminuindo *bugs* e melhorando a manutenibilidade do programa. Entre as funcionalidades que complementam o JavaScript, estão o Enum, Interface, Namespace, tipagem de funções, tipo genérico entre outros (Microsoft, 2016).

2.4.2 Node.js

O JavaScript foi desenvolvido para ser executado nos navegadores. Para executar o JavaScript no lado do servidor, foi desenvolvido o Node.js. Este é um software que permite executar o JavaScript no servidor. Isto permite que bibliotecas e estruturas escritas em JavaScript, como o Express, sejam usadas no servidor (Brown, 2019).

2.4.3 Express

Express é um framework de aplicativos web para Node.js criado por TJ Holowaychuk. Tem como filosofia ser mínimo e flexível, fornecendo um conjunto robusto de recursos para a criação de aplicativos web e móveis (Brown, 2019). Com o Express é possível manipular requisições recebidas pelos clientes através de rotas e devolver uma resposta de uma maneira mais simples para o programador.

2.4.4 MySQL

O MySQL é um Relational Database Management System (RDBMS) baseado na arquitetura cliente-servidor. O cliente comunica com o servidor através do Structured Query Language (SQL). Esta comunicação é feita via rede, independentemente se o servidor está executando localmente ou em outra máquina (DuBois, 2014).

2.4.5 ORM

Segundo Kouraklis (2019), Object Relational Mapping (ORM) representa um conjunto de técnicas em programação de computadores, que tentam fazer com que sistemas incompatíveis cooperem, se comuniquem e troquem informações. Além disso, facilita o mapeamento de um objeto para a tabela do banco de dados e o trabalho inverso também.

2.4.6 Sequelize

Sequelize é um ORM para Node.js baseado no JavaScript. Suporta a comunicação com os Relational Database System (RDBS) Postgres, MySQL, MariaDB, SQLite e Microsoft SQL Server. Possui funcionalidades úteis como o “paranoid” que realiza o soft delete de um registro, suporte a transações, validação de dados, etc. (Sequelize, 2020).

Capítulo 3

Análise de Requisitos e Modelação

Neste capítulo é apresentado a proposta para a solução do problema. Em seguida serão apresentados os requisitos e a modelagem realizada para o desenvolvimento do sistema.

3.1 Proposta de Solução

A proposta de solução é inserir a tecnologia de informação no ambiente do armazém. Para isso, foi feito a construção de um sistema que gerencie os armazéns da empresa. Este sistema é constituído por uma API que faz a comunicação intermediária entre o banco de dados e o aplicativo que auxilia o trabalho do armazém e fornece dados ao servidor. Como é previsto utilizar tablets e smartphones para trabalhar no armazém, foi escolhido o desenvolvimento de um aplicativo móvel, ao invés de um software de computador de mesa.

3.2 Mapeamento das atividades do armazém

A primeira etapa do projeto foi a compreensão de todo o trabalho que é realizado no armazém. Para compreender os trabalhos realizados no armazém e aqueles que estão relacionados, foi utilizada a técnica de etnografia. A identificação dos trabalhos que podem ser feitos ou auxiliado pelo aplicativo, foi através do diagrama de atividades que

mapeia os trabalhos do armazém.

Os trabalhos relacionados ao armazém podem ser divididos em: trabalho com kit de utensílios, preparamento de apartamentos, compra de produtos para o armazém e armazenamento de bagagem dos clientes.

3.2.1 Trabalho com kit de utensílios

Um dos principais trabalhos realizados no armazém é o trabalho com os **kits de utensílios**. Apesar das etapas deste trabalho estarem interligadas, não há uma etapa que indica o início de todo o fluxo de etapas, porém para facilitar a explicação, inicia-se pela coleta dos utensílios que compõe um kit. A Figura 3.1 mostra de forma simplificada as etapas do trabalho.

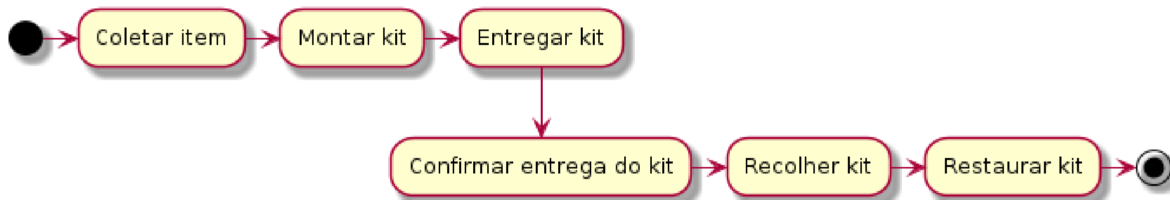


Figura 3.1: Fluxo geral do trabalho com kit de utensílios

Coleta dos utensílios

A primeira etapa do fluxo é a coleta de utensílios que comporá um kit. A Figura 3.2 demonstra as etapas deste trabalho. Uma das suas etapas pode gerar um novo trabalho chamado **Adquirir utensílio**. Nesta subetapa, o funcionário gera uma ordem de compra dos utensílios que estão em falta, compra-os e traz até o armazém.

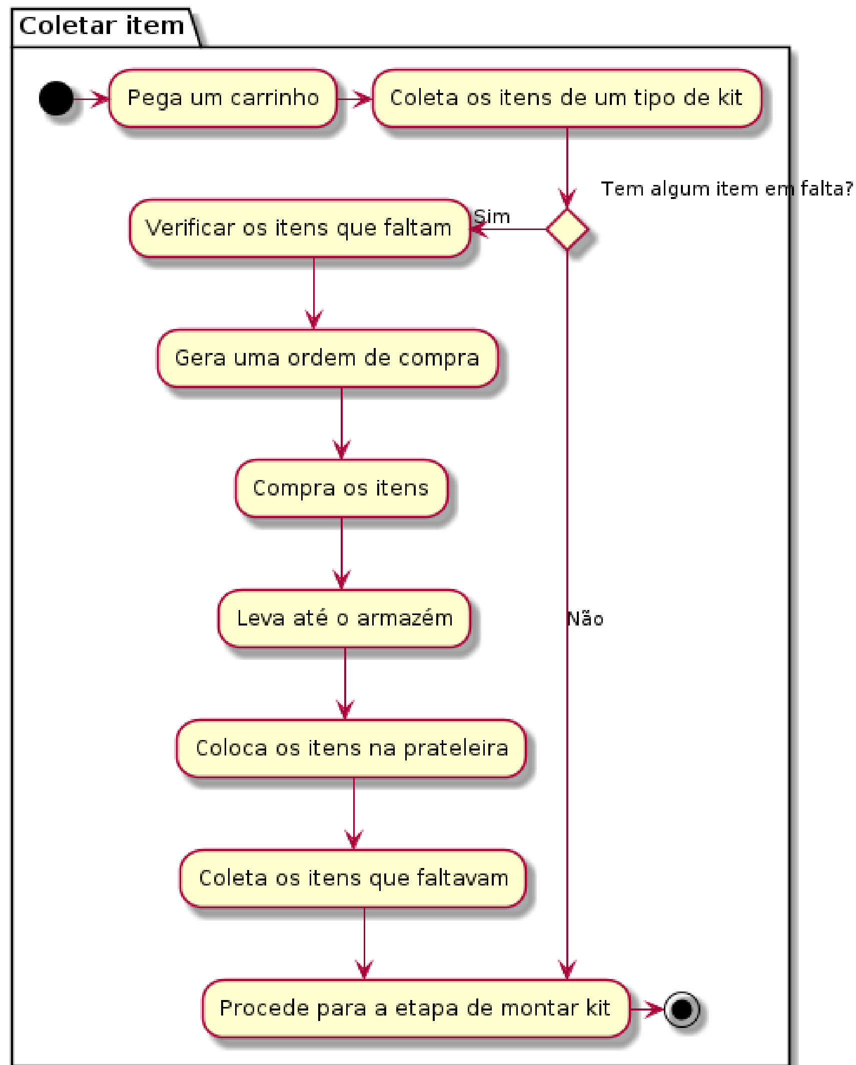


Figura 3.2: Fluxo de trabalho da coleta de itens

Nesta etapa pode-se utilizar o aplicativo como uma lista de verificação dos utensílios necessários para montar o kit e marcar os utensílios coletados. Já na subetapa para adquirir os utensílios em falta, o aplicativo pode gerar essa ordem de compra.

Montagem do kit de utensílios

A segunda etapa é a montagem do kit de utensílios. A Figura 3.3 exemplifica as etapas deste trabalho.

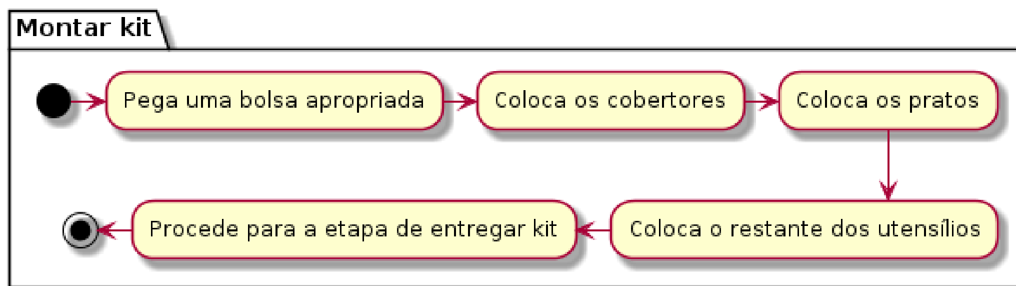


Figura 3.3: Fluxo de trabalho da montagem do kit

Nesta etapa o aplicativo pode auxiliar na verificação dos produtos coletados e inseri-los em um saco ou bolsa que será levado até o cliente.

Entrega do kit e preparo do quarto

A terceira etapa é a entrega do kit de utensílios e o preparo do quarto. A Figura 3.4 instrui as etapas deste trabalho.

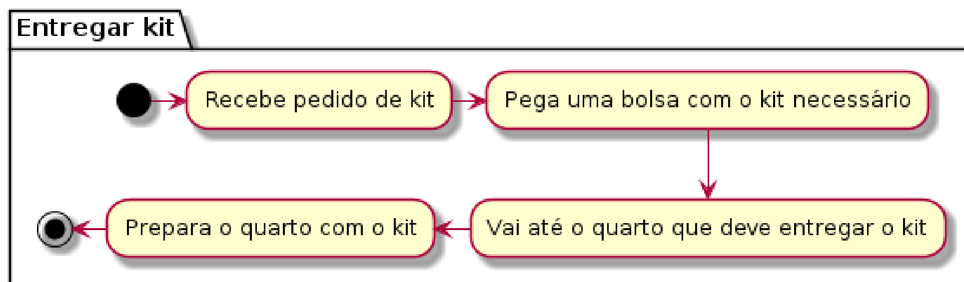


Figura 3.4: Fluxo de trabalho da entrega do kit

Nesta etapa o aplicativo pode auxiliar o trabalho, informando quantos kits de utensílios estão disponíveis para serem entregues.

Confirmação de entrega de um kit e *check in* do cliente

A quarta etapa é a confirmação de entrega de um kit de utensílios e o *check in* do cliente no quarto. A Figura 3.5 ilustra as etapas deste trabalho.

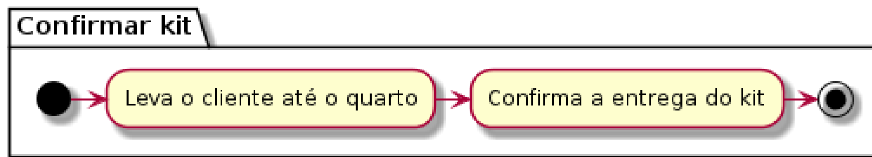


Figura 3.5: Fluxo de trabalho da confirmação de entrega do kit

Nesta etapa o aplicativo pode auxiliar para confirmar o recebimento do kit, como por exemplo, retirando uma foto da assinatura do cliente e enviar para API que armazenará no banco de dados.

Recolhimento do kit e *check out* do cliente

A quinta etapa é o recolhimento do kit e o *check out* do cliente de sua morada. A Figura 3.6 demonstra as etapas deste trabalho.

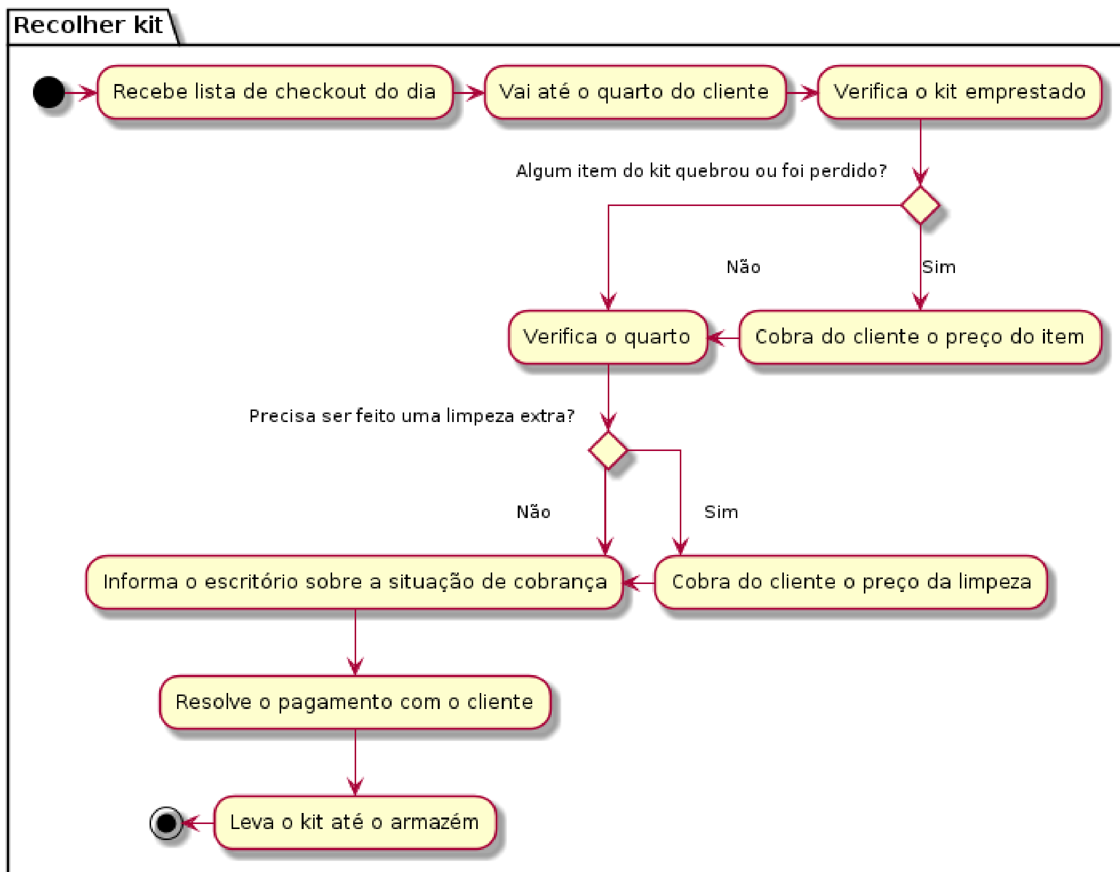


Figura 3.6: Fluxo de trabalho para recolher o kit

Nesta etapa o aplicativo pode auxiliar para confirmar o recolhimento do kit ao realizar o *check out* do cliente ou quando o kit recolhido chegar no armazém.

Restaurar utensílios recolhidos

A sexta e última etapa é a restauração dos utensílios recolhidos. A Figura 3.7 exemplifica as etapas deste trabalho.

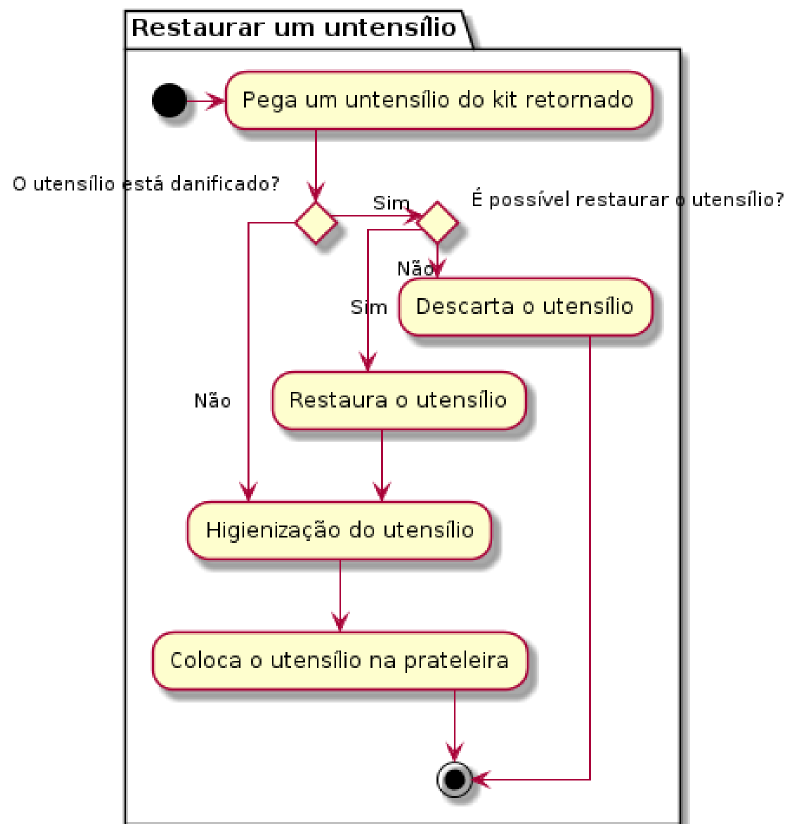


Figura 3.7: Fluxo de trabalho para restaurar os utensílios recolhidos

Nesta etapa o aplicativo pode auxiliar mostrando os passos necessários para cada utensílio e ao terminar a restauração com sucesso, adicionar o produto restaurado de volta ao estoque do armazém.

3.2.2 Preparo de apartamentos

Após firmar um contrato com o dono de um apartamento, é necessário preparar o apartamento para que os clientes possam hospedar. A Figura 3.8 instrui as etapas deste trabalho de forma simplificada.

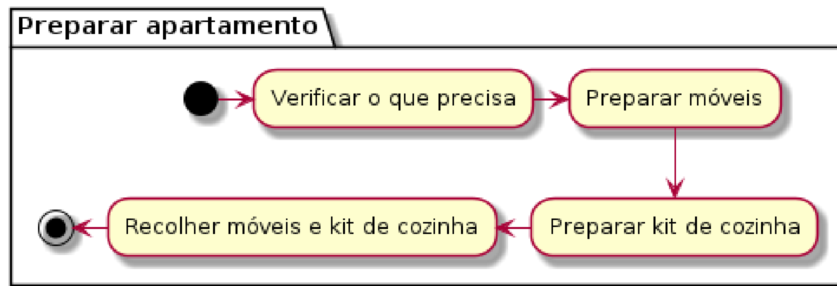


Figura 3.8: Fluxo geral para preparar um apartamento

Verificar o que precisa

A primeira etapa é verificar o que é necessário para o apartamento ser hospedável. A Figura 3.9 ilustra os detalhes desta etapa.

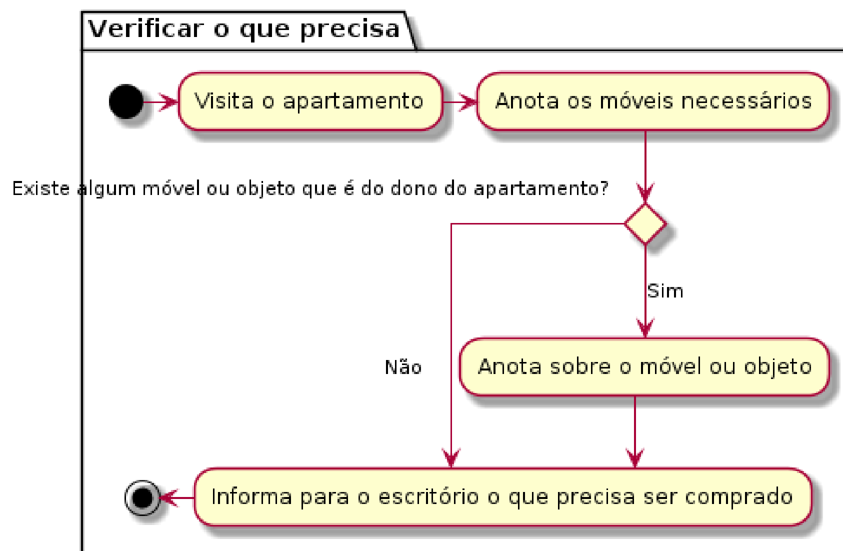


Figura 3.9: Fluxo de trabalho para verificar o que precisa

Há apartamentos que possuem móveis que são do dono da propriedade. O aplicativo pode auxiliar esta etapa, tirando uma foto dos móveis que pertencem ao apartamento e os móveis que pertencem à empresa. Também criar uma ordem de compra é uma possível funcionalidade.

Preparar kit de cozinha

A segunda etapa é preparar o kit de cozinha. A Figura 3.10 demonstra os detalhes desta etapa.

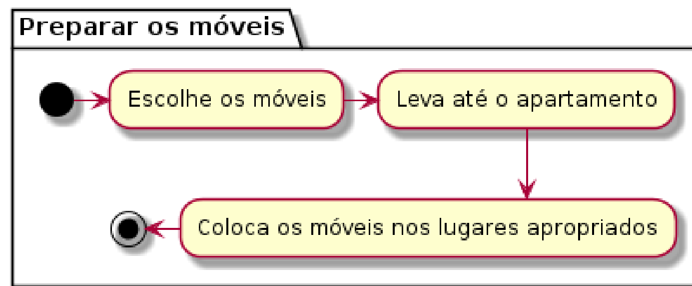


Figura 3.10: Fluxo de trabalho para preparar kit de cozinha

Nesta etapa pode-se utilizar o aplicativo como uma lista de verificação dos utensílios de cozinha necessários para montar o kit de cozinha.

Preparar móveis

A terceira etapa é preparar os móveis que serão colocados no apartamento. A Figura 3.11 mostra os detalhes desta etapa.

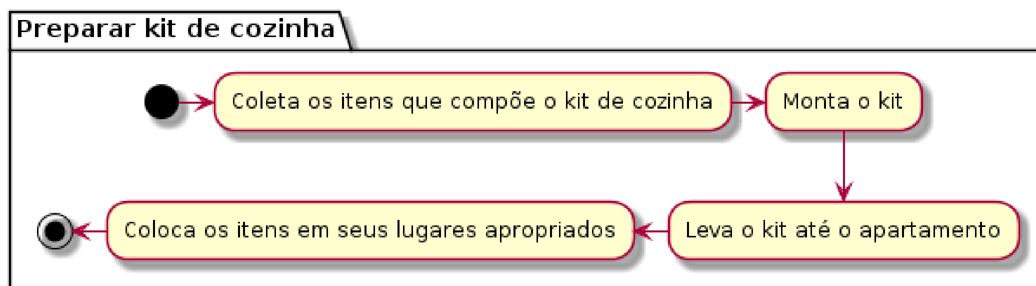


Figura 3.11: Fluxo de trabalho para preparar móveis

Nesta etapa pode-se utilizar o aplicativo como uma lista de verificação dos móveis que foram escolhidos.

Recolher os móveis e o kit de cozinha

A quarta e última etapa consiste em recolher os móveis. A Figura 3.12 exemplifica os detalhes desta etapa. Esta etapa acontece quando o apartamento deve ser devolvido para o dono. São recolhidos os móveis e o kit de cozinha.

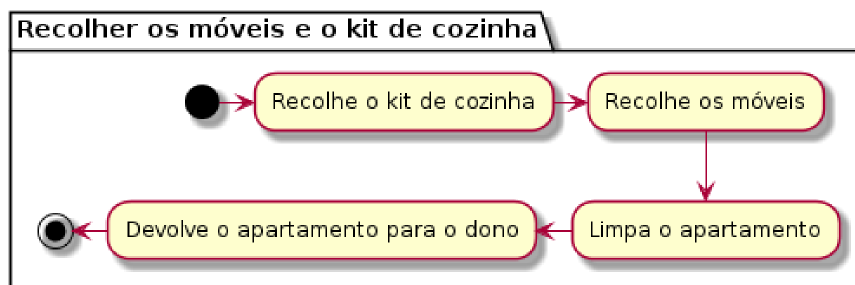


Figura 3.12: Fluxo de trabalho para recolher os móveis e o kit de cozinha

Nesta etapa o aplicativo pode auxiliar mostrando as fotos dos móveis que devem ser recolhidos, possibilitando a verificação de danificações nos móveis durante a hospedagem dos clientes.

3.2.3 Compra de produtos e utensílios para o armazém

O armazém também precisa de produtos não duráveis como sabão em pó, saco de lixos e utensílios como vassoura, linhas para costuras, etc. A Figura 3.13 instrui as etapas deste trabalho.

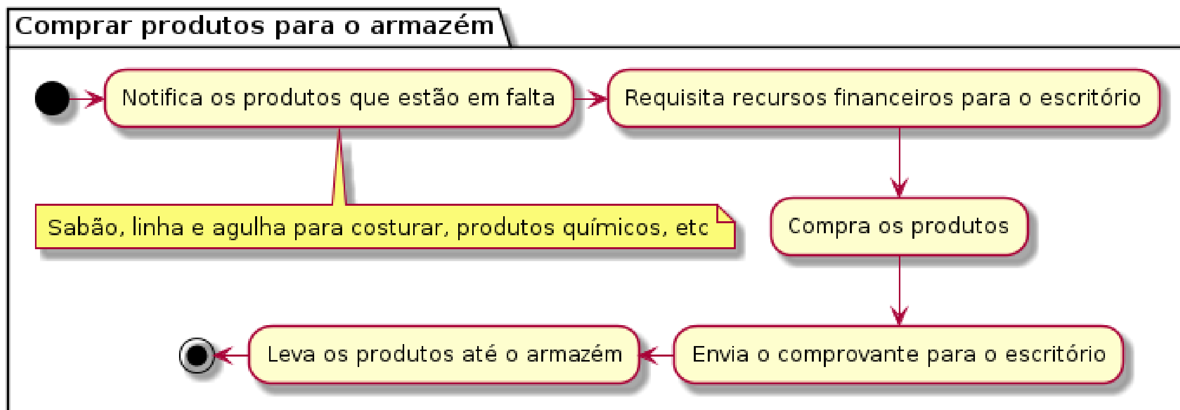


Figura 3.13: Fluxo de trabalho para compra de produtos e utensílios

Neste trabalho, o aplicativo pode auxiliar gerando uma ordem de compra dos produtos que estão em falta e utensílios necessários.

3.2.4 Armazenamento de bagagens dos clientes

No armazém pode-se armazenar as bagagens dos clientes. A Figura 3.14 ilustra as etapas para armazenar as bagagens dos clientes.

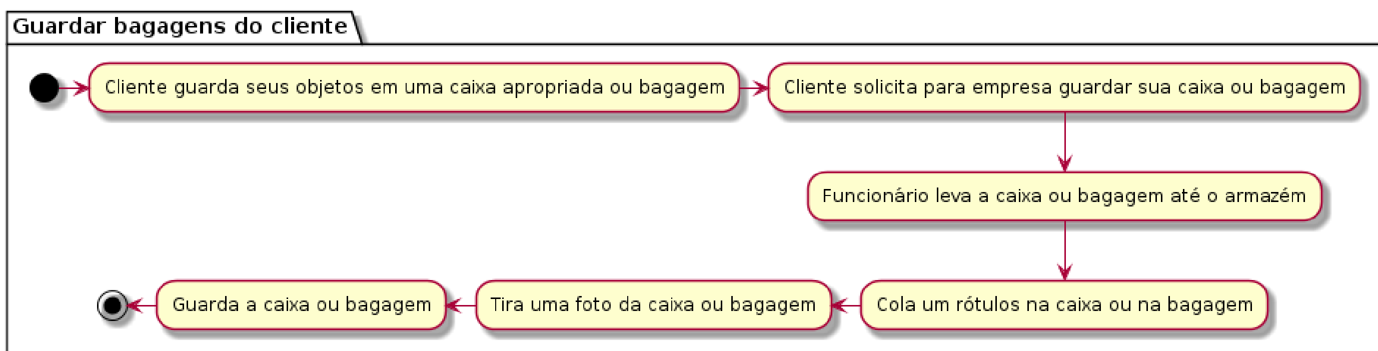


Figura 3.14: Fluxo de trabalho para armazenar as bagagens dos clientes

Quando o cliente solicitar a devolução de sua bagagem, é feito uma busca da bagagem pelo armazém. Ao encontrar, a bagagem é levada para o cliente. A Figura 3.15 ilustra as etapas deste trabalho.

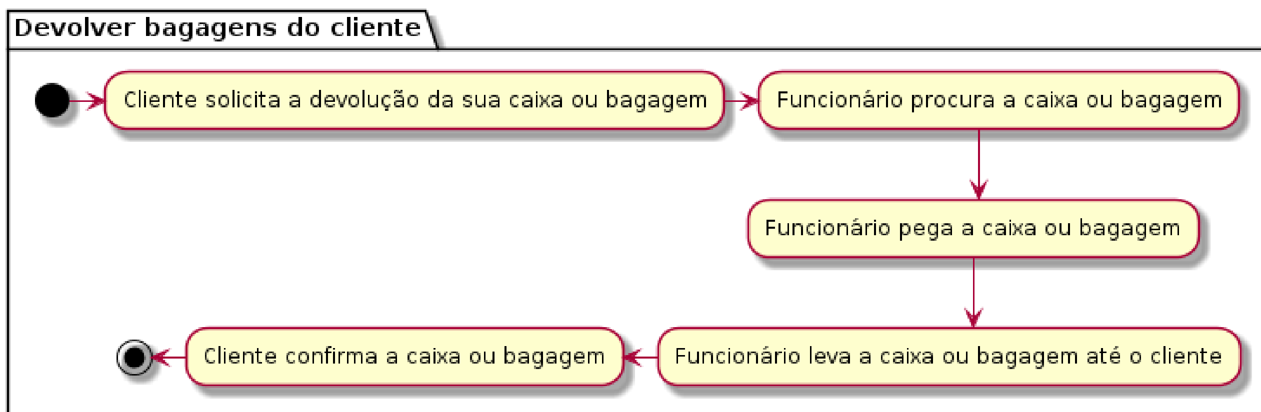


Figura 3.15: Fluxo de trabalho para retirar as bagagens dos clientes

Neste trabalho, o aplicativo pode auxiliar tirando uma foto da bagagem do cliente e registrar no banco de dados. Assim, facilita o reconhecimento da bagagem quando o funcionário for procurar a bagagem para devolver ao cliente. O local onde será armazenado, pode ser salvo pelo aplicativo. Isso funcionará similar ao armazenamento dos estoques de utensílios.

3.3 Modelação do Projeto

Feito o mapeamento dos trabalhos realizados no armazém, iniciou-se a modelação do projeto. Foram criados diagramas para a modelação: caso de uso e entidade relacionamento.

3.3.1 Requisitos

Nesta subseção serão apresentados os requisitos funcionais (RF) e não funcionais (RFN).

Requisitos funcionais

Estes requisitos são funcionalidades que o sistema deve prover.

Foram identificadas várias funcionalidades desejáveis para o sistema. Porém, somente algumas delas foram implementadas. A seguir serão apresentadas as funcionalidades

implementadas e uma lista das funcionalidades não implementadas.

RF-1. Gerenciar armazéns:

Uma das entidades do sistema é o **Armazém**. O gerenciamento deste é fundamental pois é uma entidade que engloba grande parte das outras entidades que fazem parte do sistema.

RF-1.1. Listar armazéns:

O aplicativo deverá listar todos os armazéns da empresa.

RF-1.2. Detalhes de um armazém:

O aplicativo deverá mostrar os detalhes de um armazém.

RF-1.3. Criar um armazém:

O aplicativo deverá permitir criar um armazém.

RF-1.4. Atualizar informações de um armazém:

O aplicativo deverá permitir atualizar as informações de um armazém.

RF-1.5. Arquivar um armazém:

O aplicativo deverá permitir excluir um armazém.

RF-2. Gerenciar itens:

Um item pode ser um utensílio, móvel ou até mesmo um produto consumível.

RF-2.1. Exibir catálogo de itens:

O aplicativo deverá exibir um catálogo de itens que estão registrados no sistema da empresa.

RF-2.2. Detalhes de um item:

O aplicativo deverá mostrar os detalhes de um item.

RF-2.3. Registrar um item:

O aplicativo deverá permitir registrar um item.

RF-2.4. Atualizar informações de um item:

O aplicativo deverá permitir atualizar as informações de um item.

RF-2.5. Arquivar um item:

O aplicativo deverá permitir arquivar um item.

RF-3. Gerenciar categorias de itens:

As categorias são para catalogar os itens.

RF-3.1. Listar categorias:

O aplicativo deverá listar as categorias dos itens.

RF-3.2. Criar uma categoria:

O aplicativo deverá permitir criar uma categoria.

RF-3.3. Atualizar informações de uma categoria:

O aplicativo deverá permitir atualizar as informações de uma categoria.

RF-3.4. Excluir uma categoria:

O aplicativo deverá permitir excluir uma categoria.

RF-4. Gerenciar estoques de um item:

O estoque de um item seria o item que está armazenado em um armazém. Este estoque pode ter estados como novo, usado, restaurado e quebrado. Cada estado terá uma quantidade e um local de armazenamento. O local de armazenamento indica em que prateleira está armazenado o estoque.

RF-4.1. Registrar um estoque:

O aplicativo deverá permitir registrar um estoque.

RF-4.2. Procurar um estoque por item:

O aplicativo deverá permitir buscar um estoque por um item.

A seguir estão os requisitos funcionais identificados porém não implementados.

- Item
  - Estatísticas sobre um item.

- Estoque
  - Registrar itens comprados que estão na ordem de compra.
  - Transferir um estoque de um item entre armazéns.
  - Visualizar informações de entrada e saída de itens.
- Kit
  - Associar quarto ou cliente ou ambos com o kit.
  - Auxiliar montagem de kits.
  - Confirmar recebimento do kit.
  - Confirmar recolhimento do kit.
  - Entregar kits.
  - Enviar kit para cliente externo.
  - Inserir um item no kit.
  - Selecionar kit que recolherá.
  - Receber pedido de kit.
  - Receber kit do cliente externo.
  - Visualizar estatísticas de kits.
  - Visualizar informações do kit ao realizar checkout.
- Template de kit
  - CRUD do Template de kit.
- Ordem de compra
  - CRUD de ordem de compra.
  - Comprar itens da ordem de compra.
  - Gerar PDF de ordem de compra.

- Bagagem do cliente
  - Armazenar bagagem do cliente.
  - Retirar bagagem do cliente.
  - Modificar informações sobre a bagagem do cliente.
  - Transferir bagagem do cliente entre armazéns.
  - Visualizar detalhes da bagagem do cliente.
- Ordem de preparo de apartamento
  - Criar ordem de preparo de apartamento.
  - Confirmar preparação do quarto.
  - Consultar estado do quarto.
  - Listar ordens de preparo.

Requisitos não funcionais

A seguir estão os requisitos não funcionais, os quais são especificações sobre o sistema em si.

RNF-1. Linguagem de programação do backend:
O backend deverá ser desenvolvido em TypeScript e deverá compilar para JavaScript.

RNF-2. Banco de dados:
O banco de dados deve ser o MySQL de preferência versão 5.7.

RNF-3. Aplicativo multiplataforma:
O aplicativo deve ser tanto para Android como para o iOS.

RNF-4. Framework para o aplicativo:
A aplicação móvel deve ser desenvolvida com o framework Flutter.

3.3.2 Diagrama de caso de uso

O diagrama de caso de uso é composto por dois elementos: atores e casos de uso. A seguir serão apresentados em detalhes os atores e casos de uso identificados.

Atores do sistema

Os atores identificados que utilizarão o sistema são os administradores e mantedores do armazém.

- **Administrador do Armazém**: funcionário que possui todas as permissões sobre o gerenciamento do armazém.
- **Mantedor do Armazém**: funcionário que trabalha no armazém mas não possui todas as permissões sobre o gerenciamento.

Casos de usos

O administrador do armazém possui qualquer tipo de permissão sobre as entidades relacionadas ao sistema de gerenciamento de armazém. A Figura 3.16 mostra os casos de uso do administrador do armazém.

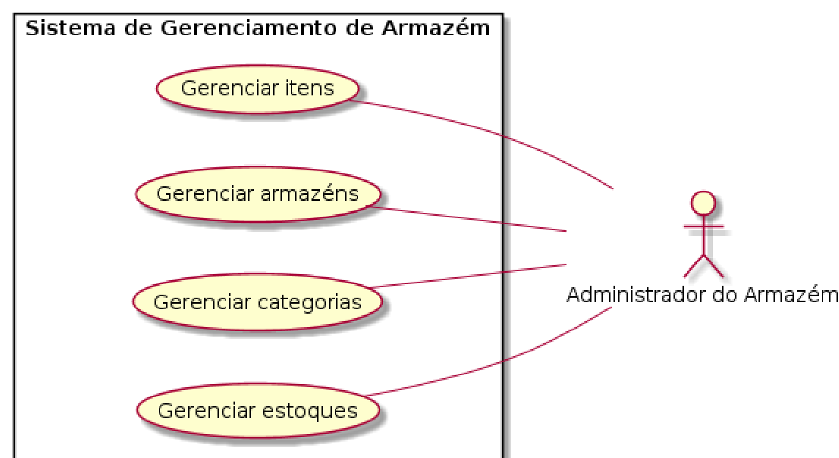


Figura 3.16: Casos de uso do administrador do armazém

O mantedor do armazém possui menos permissões comparado com o administrador do armazém. Para uma melhor visualização, o diagrama de caso de uso do mantedor de armazém foi particionado em pequenas partes. O mantedor possui apenas permissão para consultas sobre as entidades armazéns e itens. A Figura 3.17 ilustra o caso de uso relacionado às estas entidades.

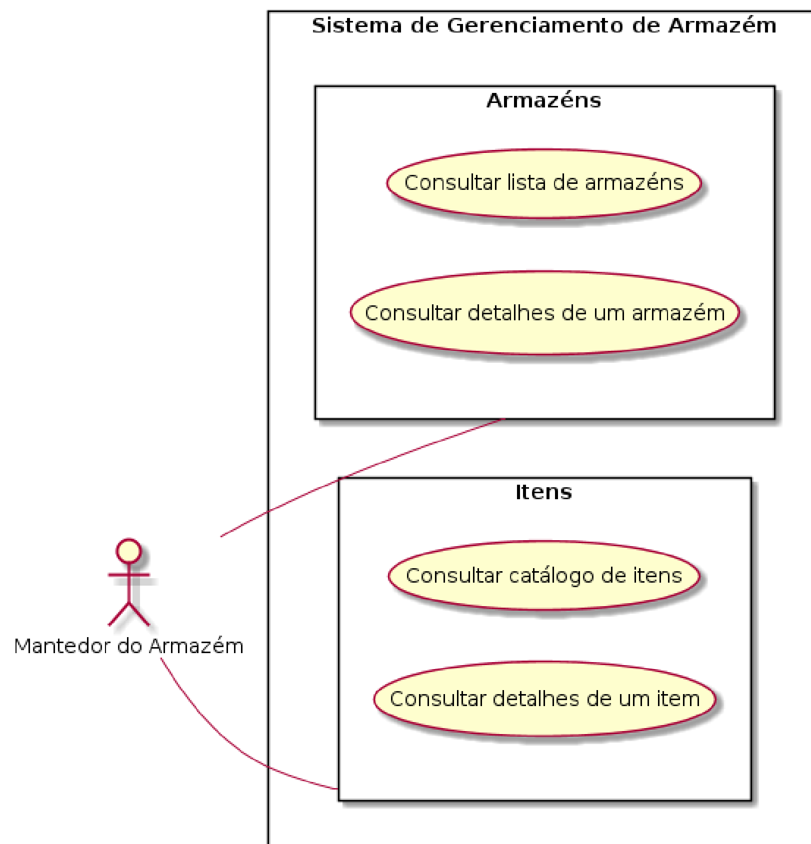


Figura 3.17: Casos de uso do mantedor do armazém parte 01/02

A Figura 3.18 demonstra os casos de uso relacionados aos estoques. Sobre os estoques, é permitido registrar um estoque e procurá-lo por um item. O motivo de permitir o mantedor realizar ações de mudança no banco, é por ser trabalhoso para um administrador do armazém ser responsável por todas as alterações do estoque.

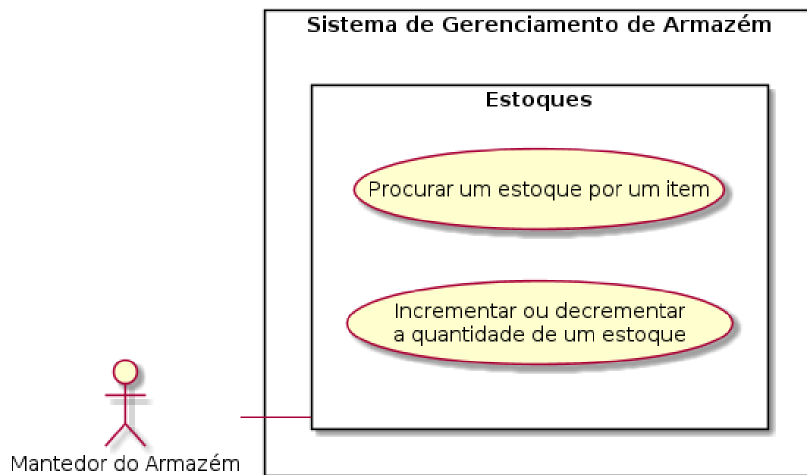


Figura 3.18: Casos de uso do mantedor do armazém parte 02/02

3.3.3 Diagrama de Entidade Relacionamento

A identificação das entidades foram feitas com base nos requisitos funcionais e casos de usos. Este diagrama não é o que foi elaborado inicialmente e sim o mais atual. Houve várias mudanças com relação a estrutura do banco de dados e foram retirados várias entidades que não foram implementadas. A Figura 3.19 ilustra o estado atual do sistema.

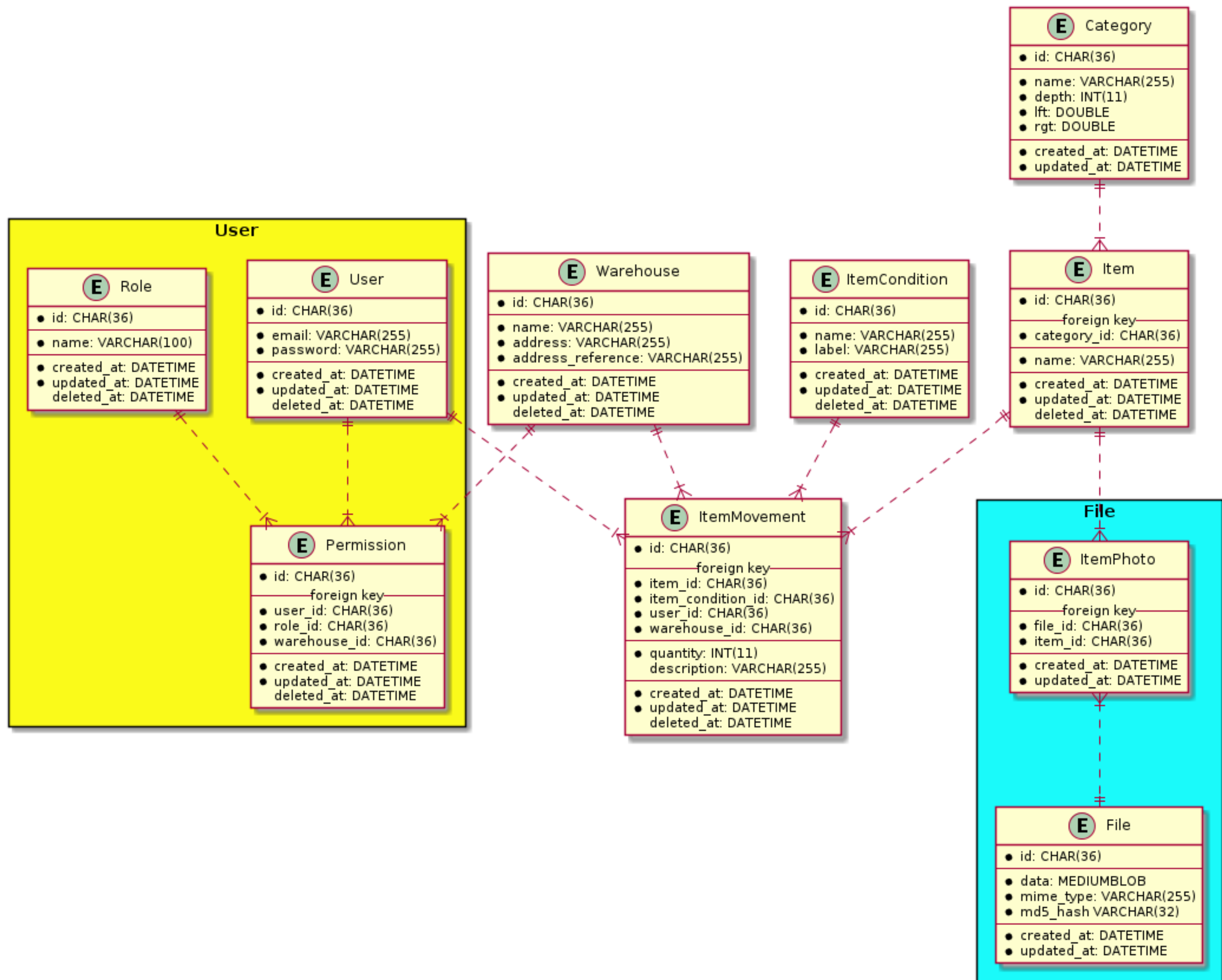


Figura 3.19: Diagrama de entidade relacionamento

3.3.4 Diagrama de Domínio

Com base no diagrama de entidade relacionamento, pode-se ter o diagrama de domínio que é um diagrama de classe de alto nível, retratando as principais entidades do domínio e seus relacionamentos. A Figura 3.19 ilustra este diagrama.

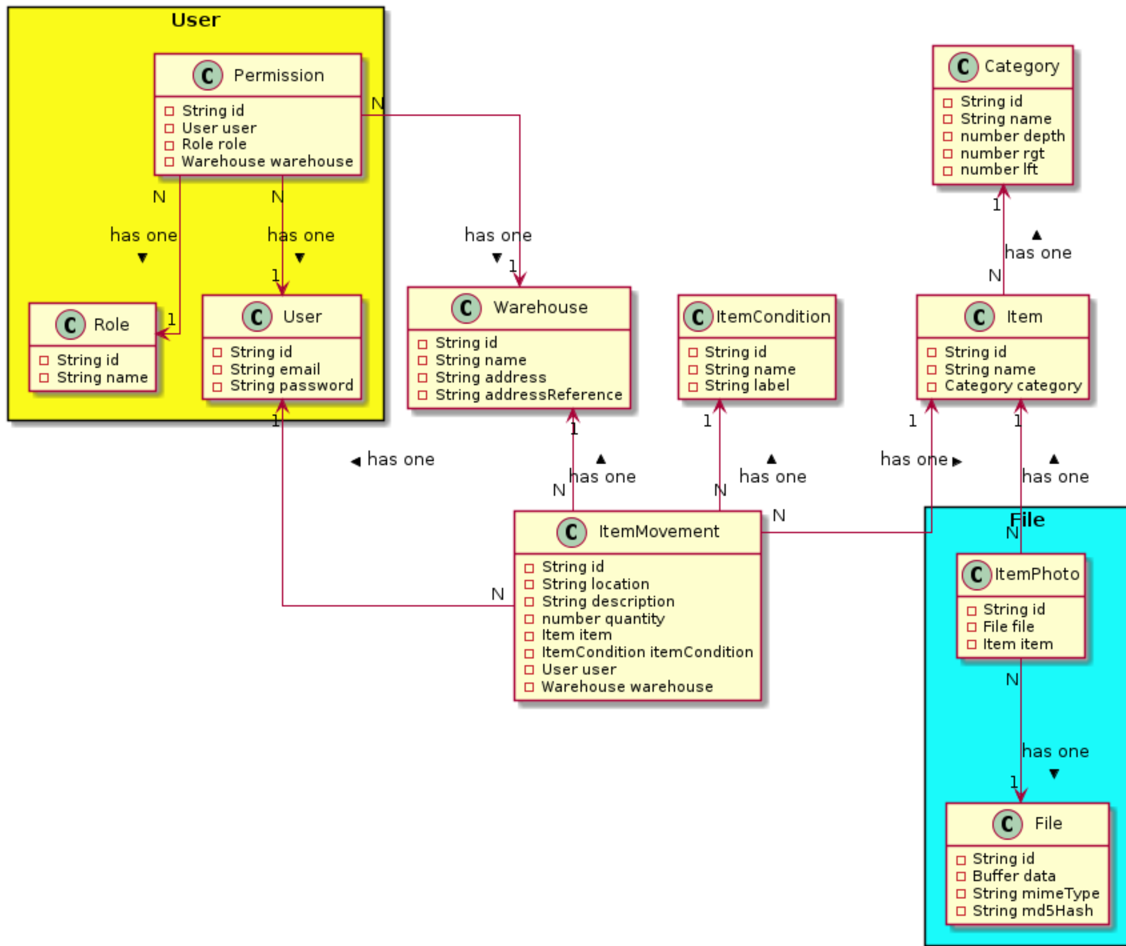


Figura 3.20: Diagrama de domínio do sistema

3.4 Arquitetura do Sistema

O sistema de gerenciamento de armazém é composto pela aplicação para dispositivos móveis, pela API e pelo banco de dados. A Figura 3.21 ilustra a arquitetura do sistema.

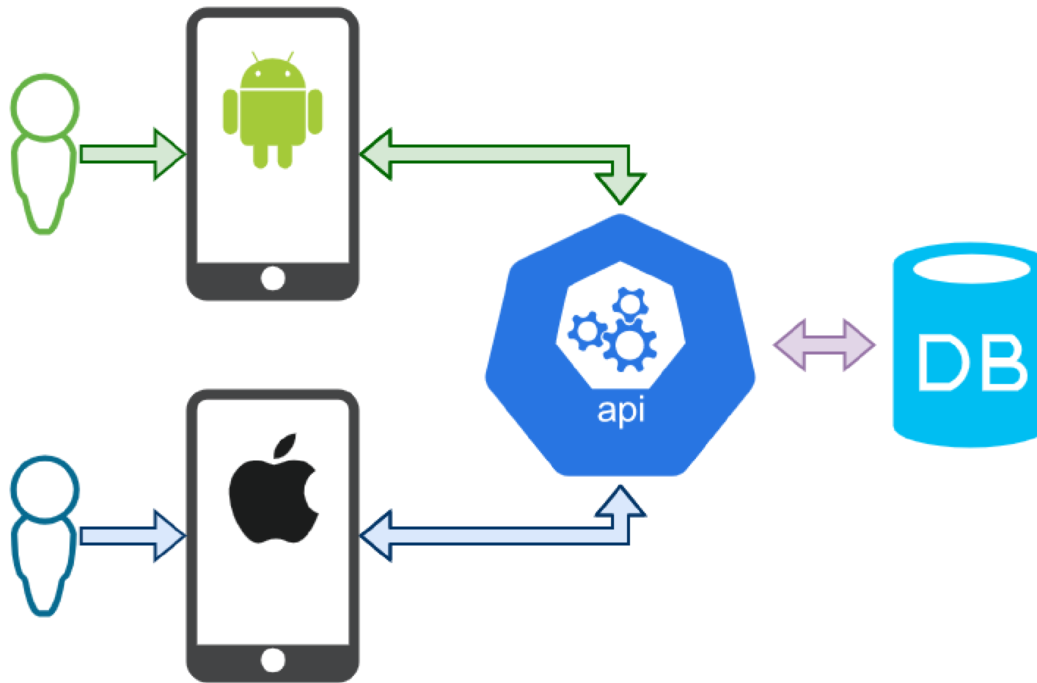


Figura 3.21: Mapa do sistema de gerenciamento de armazém

3.5 Representação de Árvore com Banco de Dados Relacional

O banco de dados utilizado no projeto é o MySQL. Neste projeto, há uma entidade chamada "Categoria" que exige representar uma estrutura de árvore. Em seu livro, Celko (2014) apresenta dois modelos para representar uma árvore no banco de dados relacional: o modelo de lista de adjacências e o modelo de conjunto aninhado.

O modelo escolhido para este projeto é o modelo de conjunto aninhado. A seguir serão explicados brevemente sobre os dois modelos.

3.5.1 Modelo de lista de adjacências

O modelo de lista de adjacências foi introduzido por Dr. E. F. Codd (Celko, 2014). Neste modelo, cada registro possui a informação de quem é seu parente, como se fosse um ponteiro. A Tabela 3.1 exemplifica a tabela "Categoria" do banco de dados. A categoria

raiz é chamada de "catalog". Por ser a raiz, ela não possui um parente.

id	**name**	**parent**
1	catalog	null
2	bedroom	1
3	kitchen	1
4	furniture	2
5	beds	4
6	utilities	3
7	plates	6
8	cuttlery	6

Tabela 3.1: Exemplo da tabela "Categoria" no modelo de lista de adjacências

Este modelo tem como vantagem de ser simples e intuitivo. Suas desvantagens são a alta complexidade e baixo desempenho das consultas, principalmente se for manipular dados utilizando somente o SQL padrão.

3.5.2 Modelo de conjunto aninhado

Segundo Celko (2014), este modelo é um modelo mais adequado para representar estruturas de árvore devido o SQL ser uma linguagem orientada a conjuntos.

Neste modelo, cada nó da árvore é considerado como um círculo que possui uma área. A Figura 3.22 ilustra a ideia do modelo. Um nó pai envolve os nós filhos. Assim, o maior círculo é a raiz e os círculos que não possuem nenhum círculo no seu interior, são as folhas.

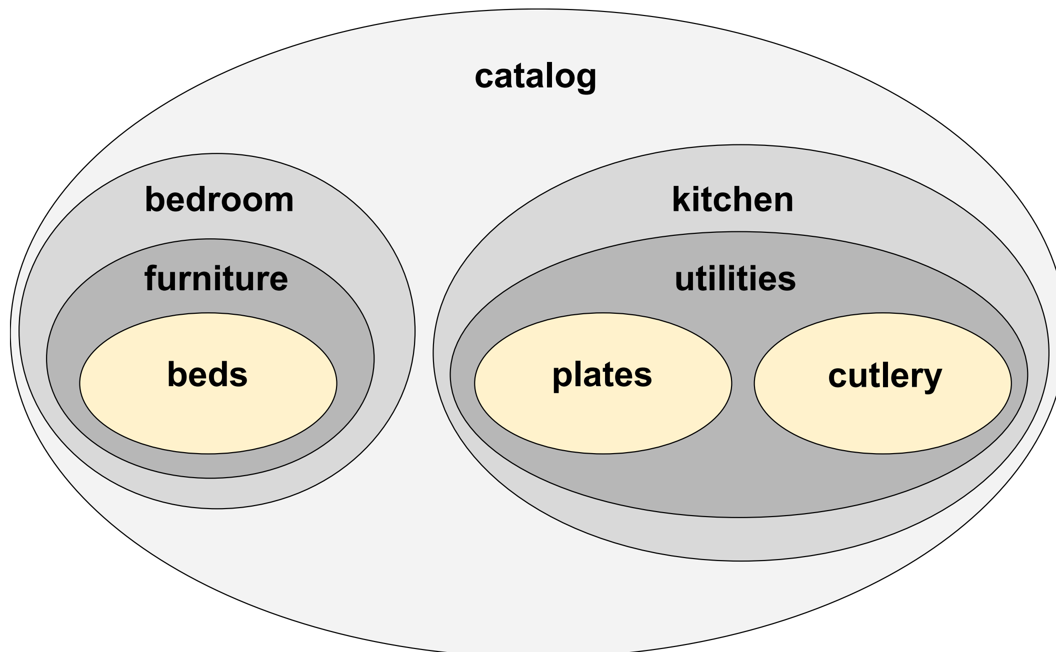


Figura 3.22: Ideia do modelo de conjunto aninhado

Para representar o aninhamento, são utilizados dois valores: esquerda e direita. Esses valores equivalem as extremidades de cada círculo. A Tabela 3.2 exemplifica a tabela "Categoria" neste modelo. Pode-se observar que não há sobreposições de valores que evita de um valor "lft" (esquerda) ter o mesmo valor que um "rgt" (direita).

id	**name**	**lft**	**rgt**
1	catalog	1	100
2	bedroom	20	40
3	kitchen	60	80
4	furniture	22	28
5	beds	24	26
6	utilities	62	68
7	plates	63	65
8	cuttlery	66	67

Tabela 3.2: Exemplo da tabela "Categoria" no modelo de conjunto aninhado

Capítulo 4

Desenvolvimento e Implementação

Neste capítulo é descrito o trabalho de desenvolvimento e implementação do aplicativo e da API.

4.1 Desenvolvimento da API

A API fará a comunicação intermediária entre o aplicativo e o banco de dados. As solicitações feitas pelo aplicativo passarão pela API, serão validadas e consultadas ou aplicadas no banco de dados. A resposta do banco de dados passará novamente pela API que converterá os dados em formato adequado para o aplicativo.

A arquitetura utilizada na API foi a Arquitetura Limpa de Martin (2018) como foi mencionado na Seção 2.1. Esta arquitetura possui várias camadas que serão implementadas. A camada mais externa chamada de "Camada de drivers e frameworks" terá menos código comparada com as outras. Isso se deve à sua natureza, pois sua maior parte são bibliotecas e frameworks de terceiros. O que será implementado são classes e interfaces que são necessárias para prover das funcionalidades destas bibliotecas. As camadas de adaptadores de interface, regras de negócios de aplicações e a regras de negócios da empresa terão a maior parte do código. A Figura 4.1 ilustra a ideia das camadas e a regra de dependência (é a mesma mostrada na Subseção 2.1.3). A regra de dependência indica que a camada mais externa pode ter conhecimento da camada mais interna, porém, o

contrário não se aplica.

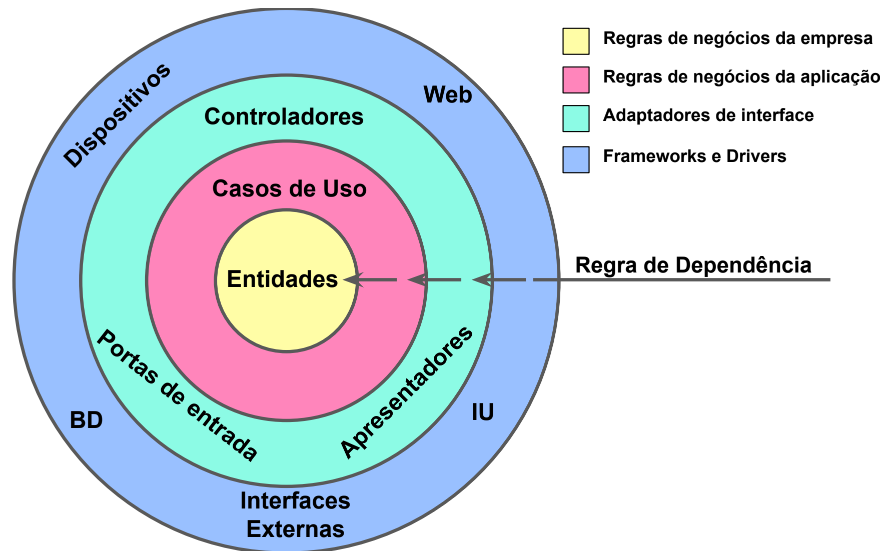


Figura 4.1: Arquitetura limpa. Adaptada. Fonte: Martin (2018)

4.1.1 Implementação da camada de interface de adaptadores

Os componentes da camada de interface de adaptadores são responsáveis por converter os dados no formato mais conveniente para os casos de uso e entidades, ou para o banco de dados e outras interfaces externas. As classes implementadas para representar a camada de adaptadores de interface são: controladores e repositórios. A seguir estão os detalhes sobre cada uma delas.

Controladores

Os controladores tem como responsabilidade a manipulação de requisições. A lógica do controlador segue os seguintes passos:

1. receber uma requisição;
2. adaptar a requisição para um Data Transfer Object (DTO);
3. passar o DTO para um caso de uso;

4. enviar o resultado do caso de uso como uma resposta adequada para o cliente.

Cada controlador recebe como parâmetro um “caso de uso”. Esse caso de uso fará o tratamento necessário sobre a requisição. O retorno do caso de uso pode ser o sucesso da execução do caso de uso ou um erro que pode ser devido ao cliente ou o servidor. Os erros do lado dos clientes são os que violam as regras de negócios, como por exemplo, uma requisição que possui dados inválidos. Um dado que será o nome da categoria pode ser inválido devido ser muito curto ou muito longo. Os erros que ocorrem no servidor são erros inesperados ou por algum problema no próprio sistema. Por exemplo, um erro que ocorre quando não consegue comunicar com o banco de dados. Na resposta está explicado sobre o erro que ocorreu (HyperText Transfer Protocol (HTTP) 5XX). Para erros do tipo do cliente, como um nome inválido para uma categoria, nas respostas também estão explicadas os motivos do erro (HTTP 4XX).

Repositórios

Os repositórios são responsáveis por comunicar com o banco de dados. Por esse motivo, os repositórios possuem vários métodos além do Create, Read, Update and Delete (CRUD). Os métodos são criados de acordo com a necessidade do aplicativo e casos de uso. Normalmente os repositórios comunicam com o banco de dados através de ORM que facilitam a geração de consultas e alteração dos dados no banco. Em geral, a lógica dos repositórios é:

1. mapear os parâmetros de seus métodos para uma consulta;
2. gerar uma consulta adequada para o banco de dados;
3. retornar o resultado da consulta.

Data Transfer Object

Os DTOs são objetos para comunicação entre as partes do sistema (como controladores e repositórios) ou até mesmo entre sistemas (como a própria API com o aplicativo). Eles

possuem somente os dados necessários para a comunicação. O benefício de utilizar os DTOs é a padronização da comunicação entre as partes. Ambos são obrigados a processar os dados para que satisfaça o DTO.

Com as classes explicadas até agora, pode-se dar um exemplo da relação entre as classes. A Figura 4.2 exemplifica a comunicação das classes controlador, caso de uso e repositório quando há uma requisição para criar uma categoria. A classe "CreateCategoryController" recebe uma instância de "CreateCategoryUseCase". Esta instância do caso de uso recebe uma instância do repositório "SequelizeCategoryRepository".

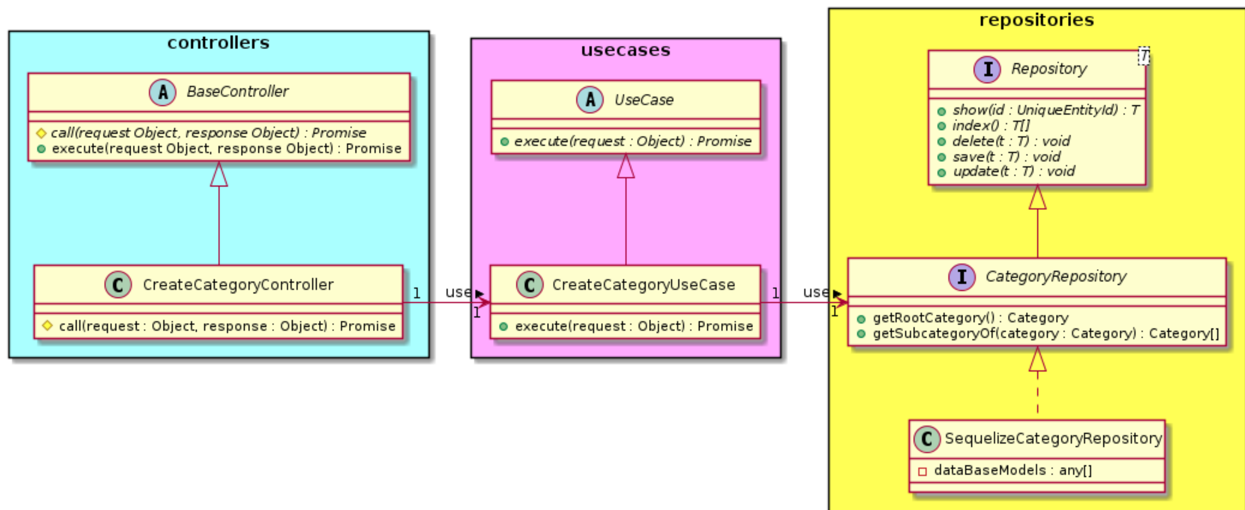


Figura 4.2: Relação entre as classes através de um caso de uso que "cria uma categoria"

Aparentemente, há uma violação da regra de dependência quando uma classe de "Caso de Uso" recebe uma instância de um "Repositório". Pois uma classe da camada interna (nesse caso a camada de regras de negócios da aplicação) não deve depender de uma classe da camada externa. Mais especificamente, não pode depender de uma classe concreta. Porém, a classe "Caso de Uso" precisa da classe "Repositório" devido a necessidade de comunicar com o banco de dados. Para resolver esse problema de violação da regra de dependência, utiliza-se o "Princípio da Inversão de Dependência". Com isso, não há violação da regra de dependência.

4.1.2 Implementação da camada de regras de negócio da aplicação

A camada de regras de negócios da aplicação é responsável por comunicar com as entidades necessárias para atingir o objetivo do caso de uso. É nesta camada que são implementados os casos de uso do sistema. Portanto, a classe que representará esta camada é o "caso de uso".

Casos de Uso

A responsabilidade do caso de uso é executar alguma regra de negócios da aplicação. Se olhar pelo ponto de vista de quem utilizará a API, os casos de uso são os serviços oferecidos pela API. Em geral, a lógica dos casos de uso é:

1. recuperar dados necessários para executar com sucesso o caso de uso;
2. criar os objetos de domínio necessários para realizar o caso de uso;
3. executar o caso de uso;
4. retornar o resultado para o controlador que efetuou a sua invocação.

4.1.3 Implementação da camada de regras de negócio da empresa

A camada de regras de negócios da empresa é responsável por validar as regras da empresa. Aqui encontra-se as entidades que o sistema irá trabalhar. As classes que representam esta camada são: entidades e objetos de valores. A seguir estão os detalhes de cada classe.

Entidades

As entidades são classes que representam os objetos que o sistema trabalhará, como por exemplo itens, armazéns, kits, etc. Mais especificamente, são objetos que podem ser identificados por um identificador como Universally Unique Identifier (UUID). Estas

entidades encapsulam a lógica de validação dos objetos. Por exemplo, se a empresa possui uma regra que as categorias não podem possuir um determinado nome, essa verificação é feita na hora que será criado o objeto. A lógica de verificação estará escrita junto com a classe da entidade.

Objetos de Valores

Os objetos de valores são objetos que não possuem um identificador. Normalmente estão associados a uma entidade como uma propriedade. No exemplo de uma entidade "categoria", o objeto de valor que ela pode ter é o "nome da categoria". A propriedade 'nome da categoria" em vez de ser uma simples "string", será um "objeto de valor" que encapsula regras de validações e possui um valor que é exatamente o nome da categoria.

A Figura 4.3 ilustra o diagrama de classe da camada de regras de negócios da empresa com base na entidade "Category". No diagrama as associações da classe "Category" sobre as classes que estão dentro do retângulo "Category Properties" está resumida em somente 1 associação.

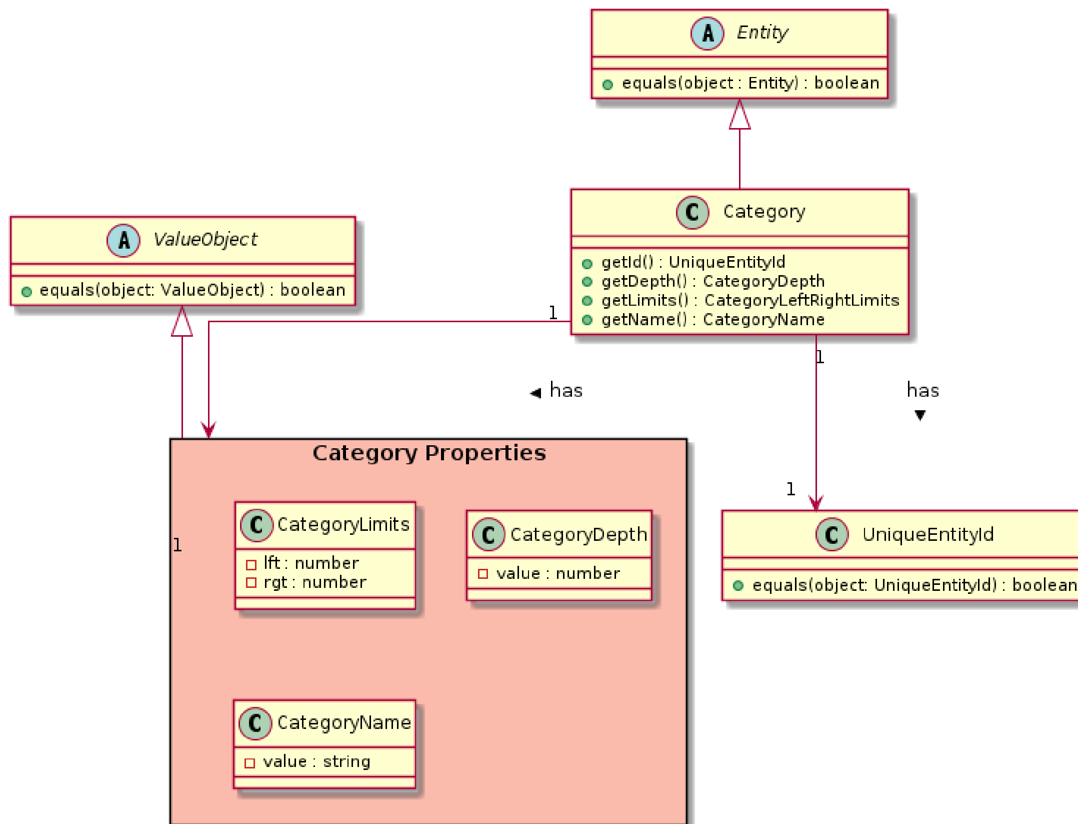


Figura 4.3: Ideia do modelo de conjunto aninhado

A classe “Category” estende a classe abstrata “Entity”. Suas propriedades são todas associadas à alguma classe que validará os dados. As validações estão dentro do construtor da classe.

A classe “UniqueEntityId” representa os UUIDs. Semelhante as classes abstratas “Entity” e “ValueObject”, possui uma função que verifica a igualdade.

As outras entidades seguem o mesmo padrão. O que difere são as propriedades de cada entidade. Como pode-se observar, a classe que equivale o nome da entidade “Category” é chamada de “CategoryName”. Se a entidade for “Item”, o nome da classe que equivale o nome será “ItemName”.

4.1.4 Implementação da camada de frameworks e drivers

Na camada de frameworks e drivers estão localizados classes e interfaces que são necessárias para realizar a comunicação com o banco de dados ou outras API externas. Exemplos dessas classes são ORM e framework de web. Para esta camada uma classe implementada foi a "modelo".

Modelos

Os modelos são a representação de uma tabela do banco de dados em forma de classe. Sua instância equivale à linha da tabela. A diferença entre modelo e entidade, estão na sua representação e seus métodos. Enquanto o modelo representa a tabela do banco de dados, a entidade representa os objetos que o negócio (neste caso o armazém) trabalha. Os métodos do modelo estão relacionados às especificações das tabelas, enquanto os métodos das entidades estão relacionados às regras de negócio da empresa.

4.1.5 Implementação da Estrutura Árvore com Banco de Dados Relacional

Como mencionado na seção 3.5, este projeto precisou representar a tabela "Category" em forma de árvore e o modelo para representação escolhido foi o modelo de conjunto aninhado. A Figura 4.4 mostra a tabela "Category" criada no banco de dados. O nome dado para os limites do conjunto é "lft" e "rgt". O nome desses campos estão abreviados por serem uma palavra reservada no MySQL.

```
mysql> desc category;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| id         | char(36)      | NO   | PRI | NULL    |       |
| name       | varchar(255)  | NO   |     | NULL    |       |
| lft        | double(14,10) | NO   |     | NULL    |       |
| rgt        | double(14,10) | NO   |     | NULL    |       |
| depth      | int(11)       | NO   |     | NULL    |       |
| created_at | datetime      | NO   |     | NULL    |       |
| updated_at | datetime      | NO   |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
7 rows in set (0.13 sec)
```

Figura 4.4: Tabela "Category" criada no MySQL

Para facilitar algumas buscas, adicionou-se o campo "depth" que representa a profundidade da árvore. Este valor é um número inteiro que se inicia do valor 0.

Entre as buscas que foram facilitada devido ao campo "depth", está a busca de subcategorias. Com este campo, é possível evitar uma operação que é necessária para determinar quais são as subcategorias. Para calcular as profundidades de cada categoria, é necessário uma operação como a seguinte:

```
SELECT node.name, COUNT(parent.id) AS depth
FROM category AS parent, category AS node
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.id;
```

4.2 Desenvolvimento do Aplicativo

O desenvolvimento do aplicativo foi feito com o framework Flutter. A seguir estão as telas com base nas funcionalidades do aplicativo.

4.2.1 Funcionalidades do aplicativo

Login

O aplicativo possui uma tela de login para ter acesso ao sistema do armazém. A Figura 4.5 ilustra a tela de login.

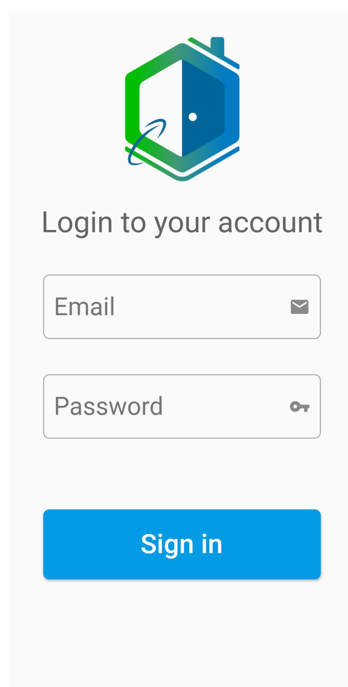


Figura 4.5: Tela de login

Login no Armazém

O sistema visa ter dois tipos básicos de usuários: administrador e mantedor do armazém. O mantedor do armazém possui menos permissões comparado ao administrador o que implica em menos funcionalidades que podem serem feitas pelo aplicativo. Por esse motivo, é feito mais um login com base na permissão. A permissão é uma combinação de um papel (administrador ou mantedor) e um armazém. Portanto, um usuário pode ter várias permissões. Por exemplo, um usuário é administrador do armazém de Bragança mas é um mantedor no armazém de Mirandela. A Figura 4.6 ilustra a tela de login em um armazém.

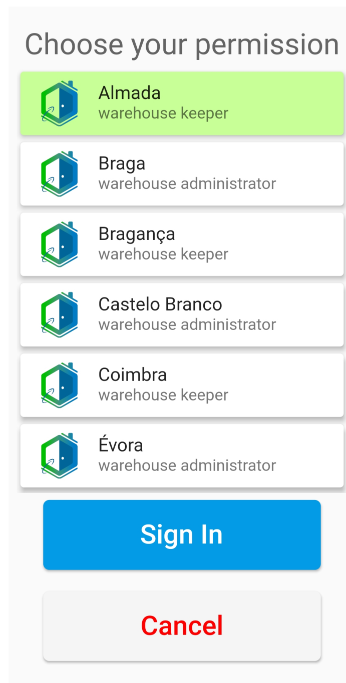


Figura 4.6: Tela de login em um armazém

Menu

Após feito os passos de login, o usuário é encaminhado para tela de menu. Como aplicativo possui várias funcionalidades, necessita de uma tela que lista as possíveis funcionalidades. A tela de menu tem como objetivo dividir as funcionalidades de acordo com o interesse do usuário. As funcionalidades relacionadas aos itens, estão no botão “Item Catalog”. As funcionalidades relacionadas aos armazéns, estão no botão “Warehouses” e assim segue o padrão. A Figura 4.7 ilustra a tela menu.

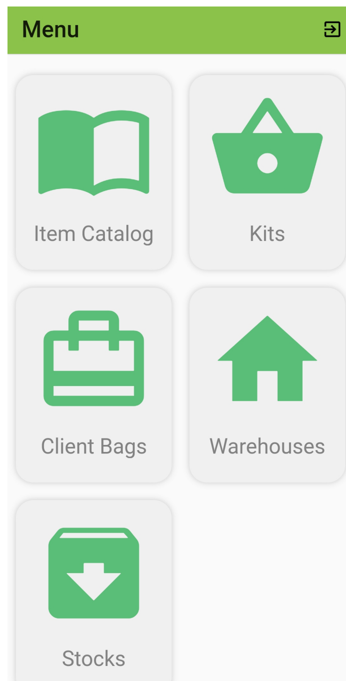


Figura 4.7: Tela de menu

Catálogo de itens

Ao apertar o botão “Item Catalog”, o usuário é encaminhado para a tela que lista os itens de acordo com as categorias. A Figura 4.8 ilustra esta tela.

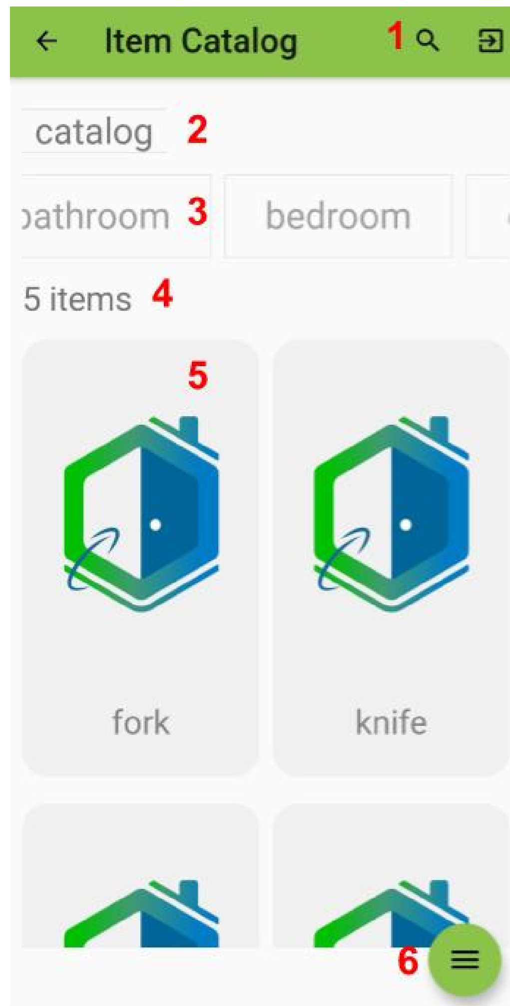


Figura 4.8: Tela de catálogo de itens

A tela é composta por: a busca de um item pelo nome, caminho até a categoria atual, lista de subcategorias, quantidades de itens que serão listados e lista de itens.

- O número 1 indica o botão que ao apertar, o usuário é movido para a tela de busca por nome dos itens. Esta tela será mostrada mais adiante.
- O número 2 designa o caminho atual até a categoria atual.
- O número 3 aponta as subcategorias que a categoria atual apresenta.
- O número 4 indica a quantidade de itens que serão listados.

- O número 5 designa a lista de itens que estão relacionados à categoria atual. O significado de "estar relacionado" é que pode estar ligado diretamente com a categoria atual ou está ligado à alguma subcategoria que está abaixo da categoria principal. Por exemplo, os itens "garfo" e "faca" não estão ligados diretamente com a categoria "catalog" e sim uma outra categoria que está abaixo de "catalog". Os itens são listados de forma paginada e estão em ordem alfabética. Ao quando está sendo carregado mais itens, é dado um feedback para o usuário. Da mesma forma, é dado um feedback que chegou no fim da lista. A Figura 4.9 ilustra os feedbacks mencionados.

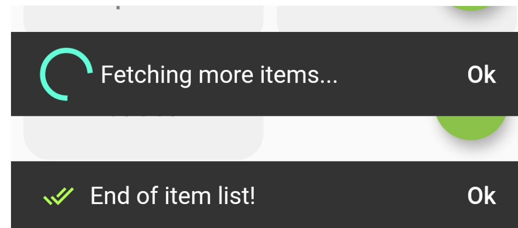
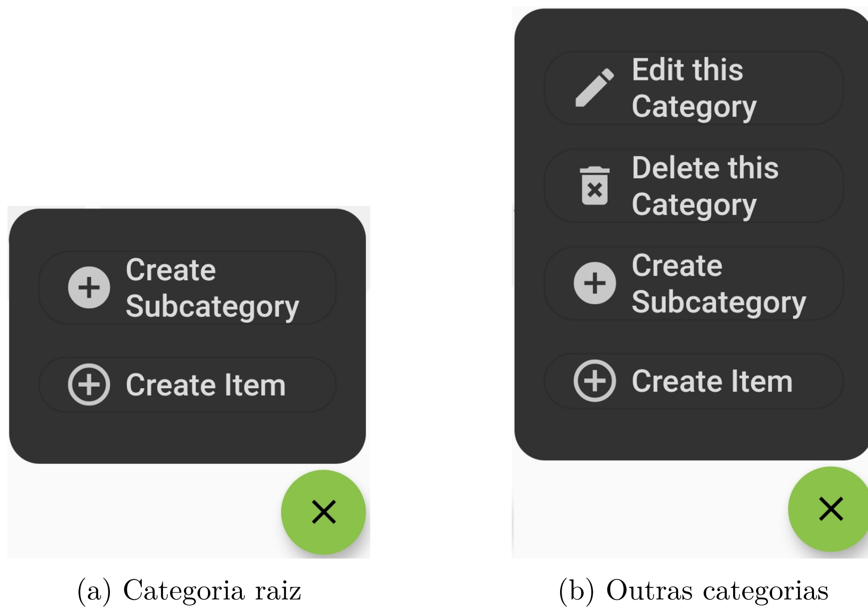


Figura 4.9: Feedbacks da tela de catálogo de itens

- O número 6 aponta o botão que mostra ações (funcionalidades) que podem ser feitas na tela de catálogo de itens. As funcionalidades são: criar uma subcategoria, criar um item, editar a categoria atual e deletar a categoria atual. Há uma exceção das ações disponíveis quando está na categoria raiz que é o "Catalog". Nesta categoria, não se pode realizar a edição e deleção da categoria atual. A Figura 4.10 ilustra quando é apertado este botão nos dois casos.



(a) Categoria raiz (b) Outras categorias

Figura 4.10: Menu de ações da tela de catálogo de itens

Criar nova subcategoria

Ao apertar o botão da ação "Create subcategory", o usuário é encaminhado para a tela de criar uma nova subcategoria que é ilustrada pela Figura 4.11. Após inserir o nome da categoria e apertar o botão "Create", é criado uma nova subcategoria que estará abaixo da categoria principal que neste caso é a categoria "utilities".

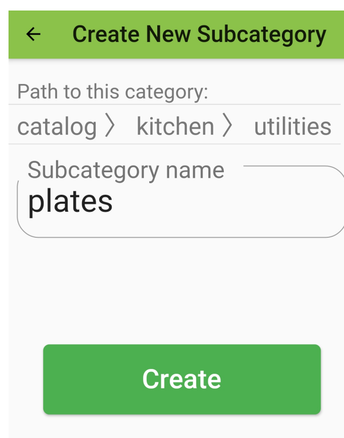
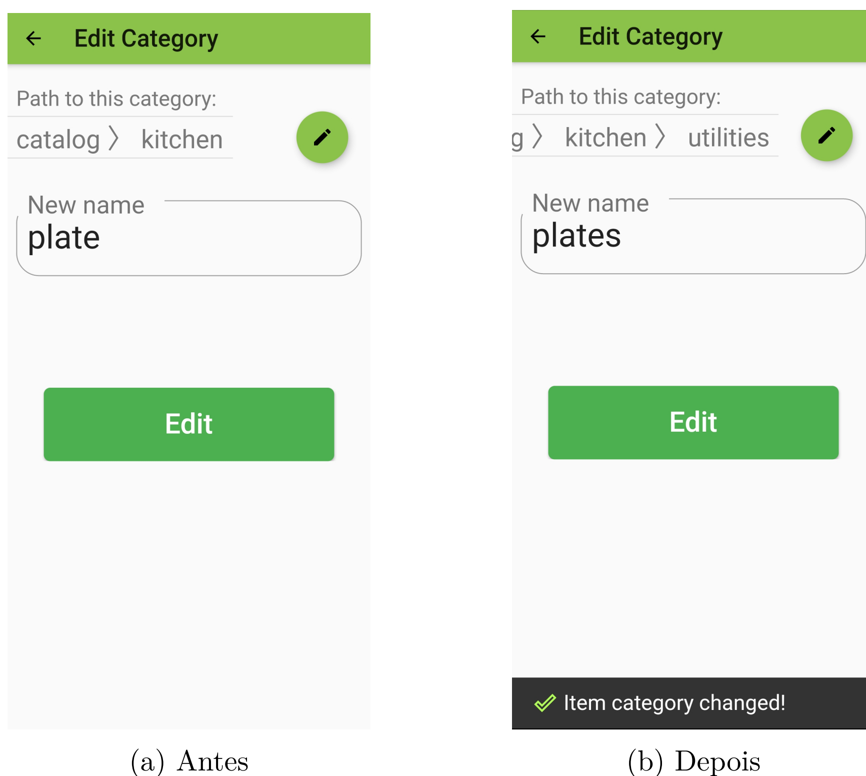


Figura 4.11: Tela de criação de subcategoria

Editar categoria atual

Ao apertar o botão de "Edit this category", o usuário é encaminhado para a tela de editar a categoria atual, que é ilustrada pela Figura 4.12a. A tela possui um botão para alterar o caminho para esta categoria. Ao apertar este botão, o usuário é encaminhado para tela que permite definir um outro caminho para esta categoria. Em outras palavras, permite o usuário escolher uma nova categoria parente (que a categoria que será editada pertencerá). A Figura 4.12b mostra após feitos as alterações necessárias.



(a) Antes (b) Depois

Figura 4.12: Menu de ações da tela de catálogo de itens

Deletar categoria atual

A deleção de uma categoria é composta por três etapas: gerenciamento das subcategorias, gerenciamento de itens da categoria que será deletada e deleção da categoria. Estas etapas são importantes devido a estrutura das categorias ser uma árvore e a relação entre categoria e itens.

A primeira etapa é o gerenciamento das subcategorias que é ilustrada pela Figura 4.13. Se uma categoria for removida da estrutura de árvore, os seus nós filhos serão vinculados à categoria parente. No caso, a categoria "pillows" será excluída, logo, a subcategoria "red pillows" será vinculada à categoria parente "utilities".

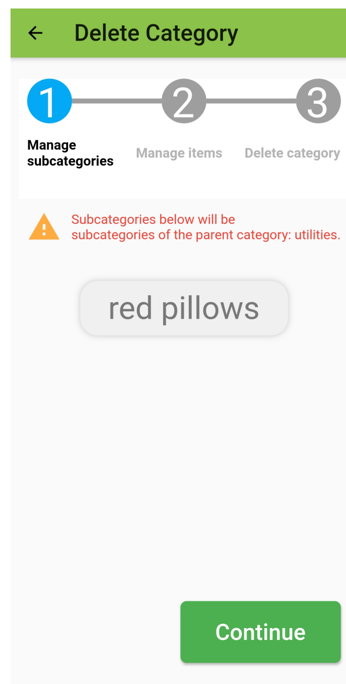


Figura 4.13: Primeira etapa de deleção de categoria

A segunda etapa é o gerenciamento de itens que pertencem a categoria que será excluída, como é mostrado na Figura 4.14. Neste caso, há somente um item que pertence a categoria "pillows". Ela será vinculada à categoria parente "utilities", caso não altere a nova categoria que este item pertencerá.

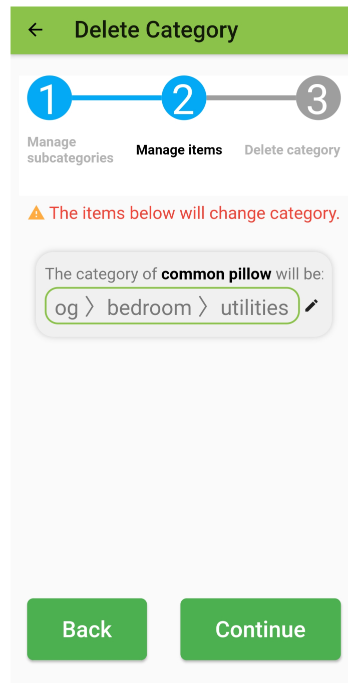


Figura 4.14: Segunda etapa de deleção de categoria

A terceira etapa é a confirmação. Caso confirme, será feito as mudanças de categorias nos itens e em seguida a remoção da categoria. A Figura 4.15 ilustra esta ultima etapa.

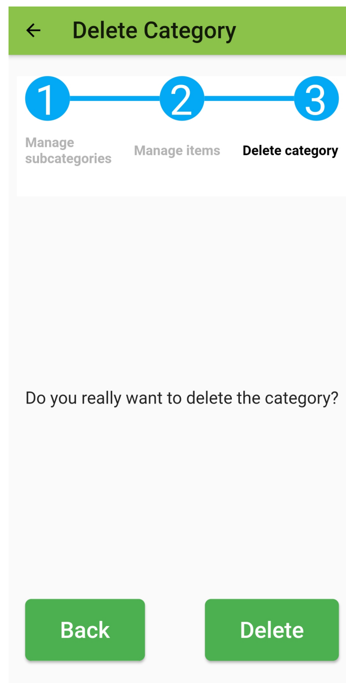


Figura 4.15: Terceira etapa de deleção de categoria

Escolher/mudar categoria

Um item possui uma categoria. As próprias categorias também possuem uma categoria parente. A escolha ou mudança de categoria é feita pela tela que a Figura 4.16 ilustra. Nesta tela, há o caminho para a categoria atual, a lista de subcategorias que estão ligadas diretamente à categoria atual e o botão de ações.

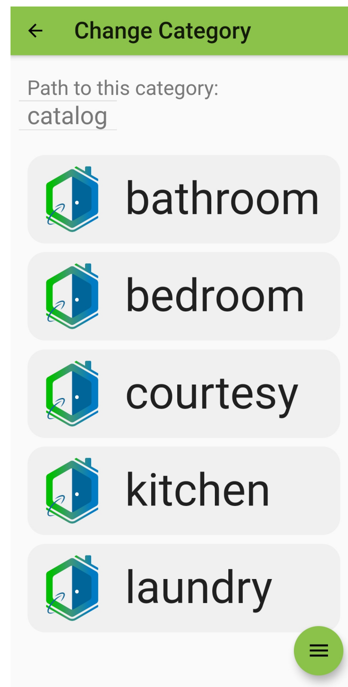
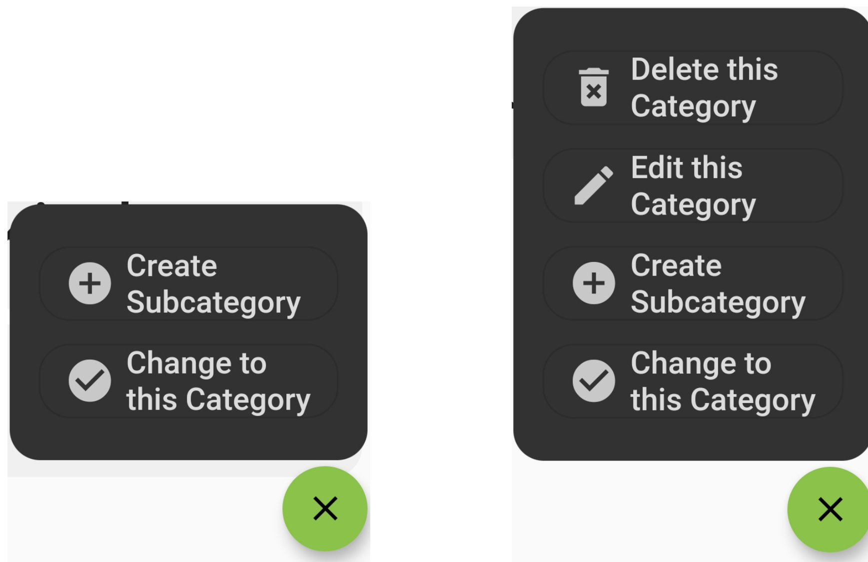


Figura 4.16: Tela de escolha ou mudança de categoria

O botão que mostra o menu de ações segue a mesma ideia da tela de catálogo de itens. Quando a categoria principal é a categoria raiz, o qual neste sistema é a categoria "catalog", as ações que podem ser tomadas são limitadas. Para as demais categorias, a quantidade de ações são maiores. A Figura 4.17 exemplifica esta diferença.

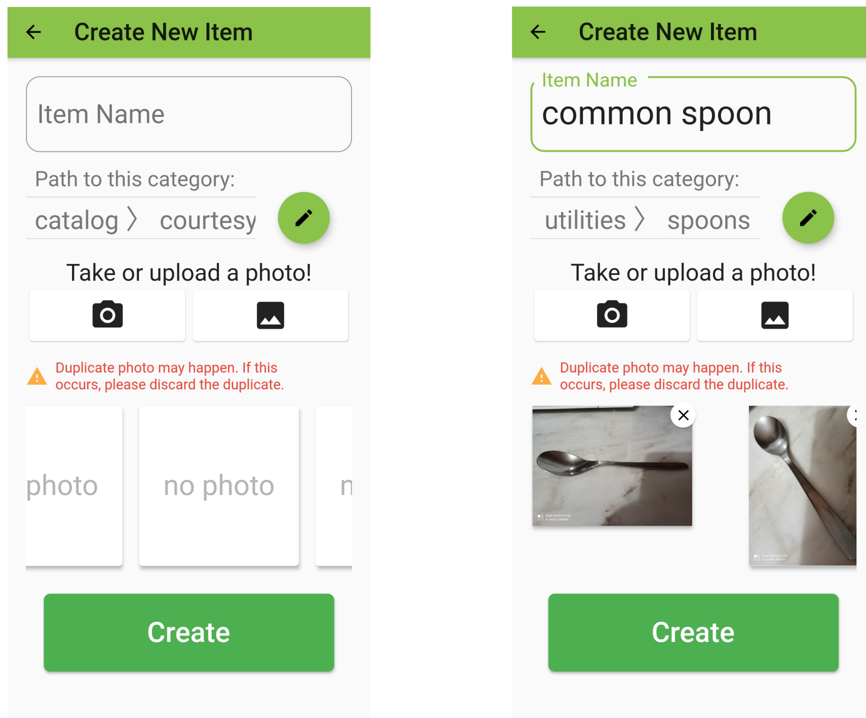


(a) Categoria raiz (b) Outras categorias

Figura 4.17: Menu de ações da tela de seleção de categoria

Criar item

Um item deve ter um nome e uma categoria. Opcionalmente, pode possuir até 5 fotos. A tela de criar um item possibilita colocar fotos que estão no dispositivo ou tirar fotos com a câmera do dispositivo. A Figura 4.18 ilustra antes e depois de inserir as informações necessárias para criar um item.

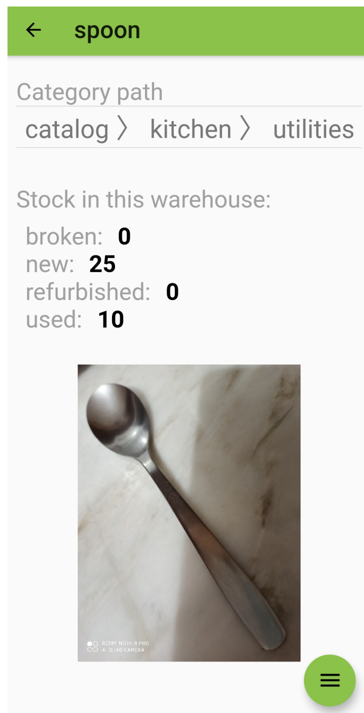


(a) Antes (b) Depois

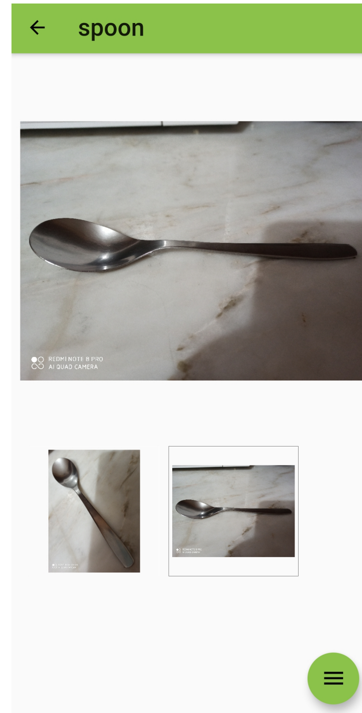
Figura 4.18: Tela de criar item

Detalhes do item

Ao apertar em um item na tela de catálogo de itens, o usuário é encaminhado para tela que detalha as informações do item. As Figuras 4.19a e 4.19b ilustram esta tela.



(a) Parte 01 de 02



(b) Parte 02 de 02

Figura 4.19: Telas de detalhes de um item

Esta tela é composta por: nome do item, fotos do item, caminho para a categoria que este item pertence e um botão de ações. O botão de ações possui opções para editar ou deletar o item. A Figura 4.20 ilustra as opções do botão de ações.

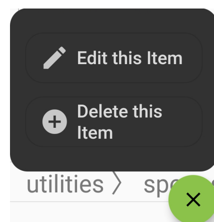
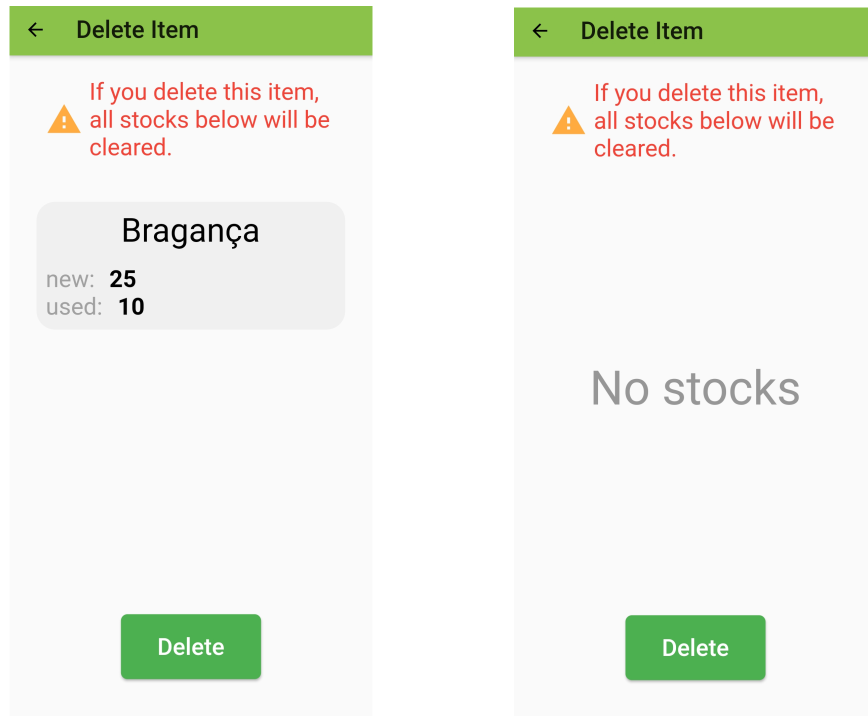


Figura 4.20: Menu de ações da tela de detalhes de um item

Excluir um item

Ao escolher a ação “Delete this item” na tela de detalhes de um item, o usuário é direcionado para a tela de exclusão de um item. O usuário é alertado que os estoques do item que será excluído, será limpado, ou seja, terá seus estoques zerados. A Figura 4.21a

exemplifica o caso que há estoques e a Figura 4.21b o caso que não há nenhum estoque.



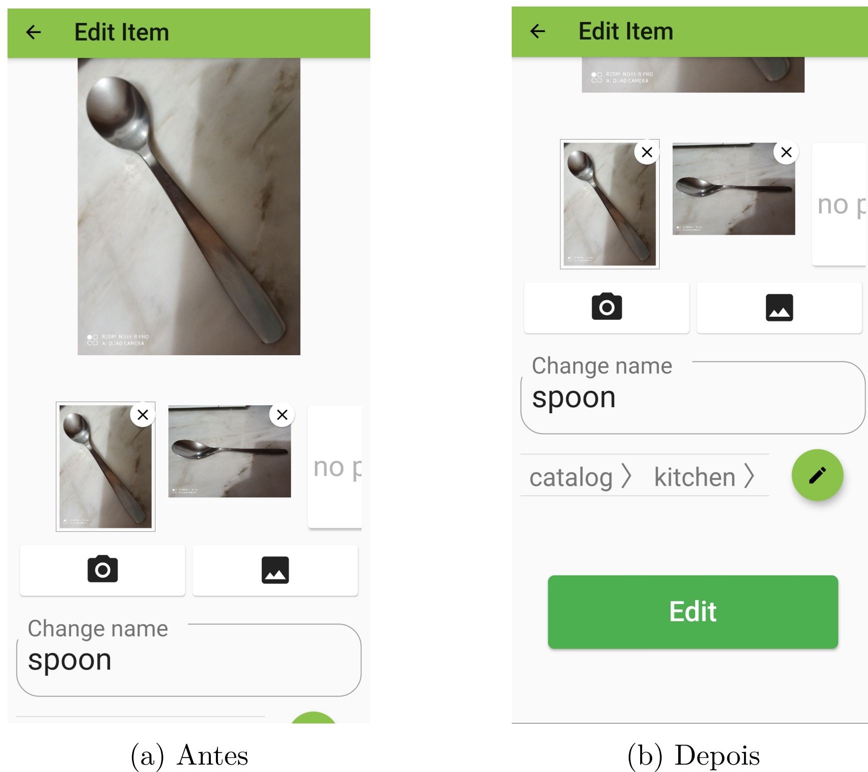
(a) Quando há estoques (b) Quando não há estoques

Figura 4.21: Tela de excluir um item

O item não é excluído totalmente do banco de dados. É feito uma exclusão suave que é registrado a data que foi feito a exclusão e este passa a não aparecer em uma busca comum no banco de dados. Mais especificamente, o registro do item que será excluído tem o campo "deleted_at" preenchido com a data atual.

Editar um item

Ao escolher a ação "Edit this item" na tela de detalhes de um item, o usuário é direcionado para a tela que é ilustrada pela Figura 4.22. Nesta tela o usuário pode alterar o nome, categoria e inserir ou deletar fotos do item.



(a) Antes

(b) Depois

Figura 4.22: Tela de editar item

Listar armazéns

Ao apertar o botão "Warehouse" na tela de menu, o usuário é encaminhado para a tela que lista os armazéns registrados. A Figura 4.23 ilustra esta tela. Se apertar um armazém que está na lista, pode-se ver os detalhes desse armazém. O botão que está no canto inferior direito da tela, é para criar um novo armazém.

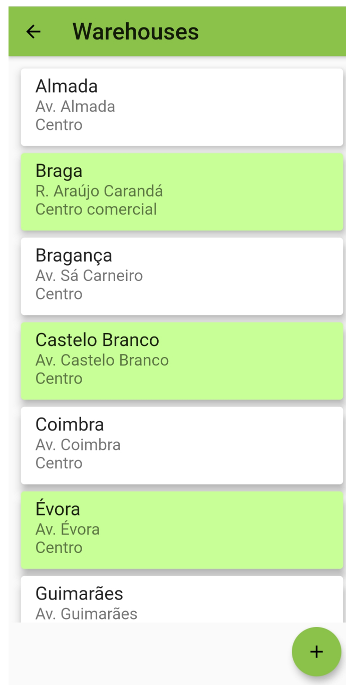


Figura 4.23: Tela de lista de armazéns

Criar um armazém

Um armazém deve possuir um nome, endereço e uma referência. A Figura 4.24 ilustra a tela de criação de um armazém.

Figura 4.24: Tela de criação de um armazém

Detalhes de um armazém

Ao apertar em um armazém da lista, o usuário é direcionado para a tela que mostra os detalhes de um armazém. A Figura 4.25 ilustra esta tela com exibição do menu de ações. As ações são: editar e excluir um armazém.

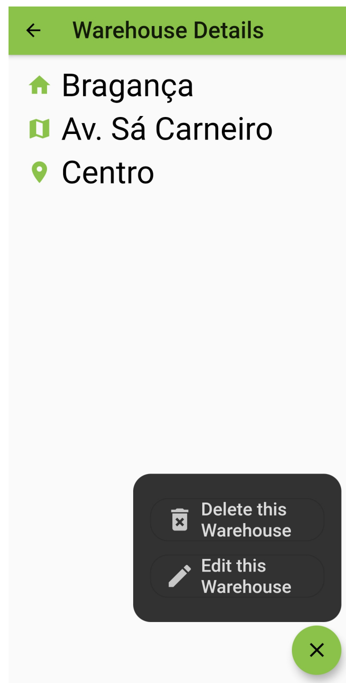


Figura 4.25: Menu de ações da tela de criação de um armazém

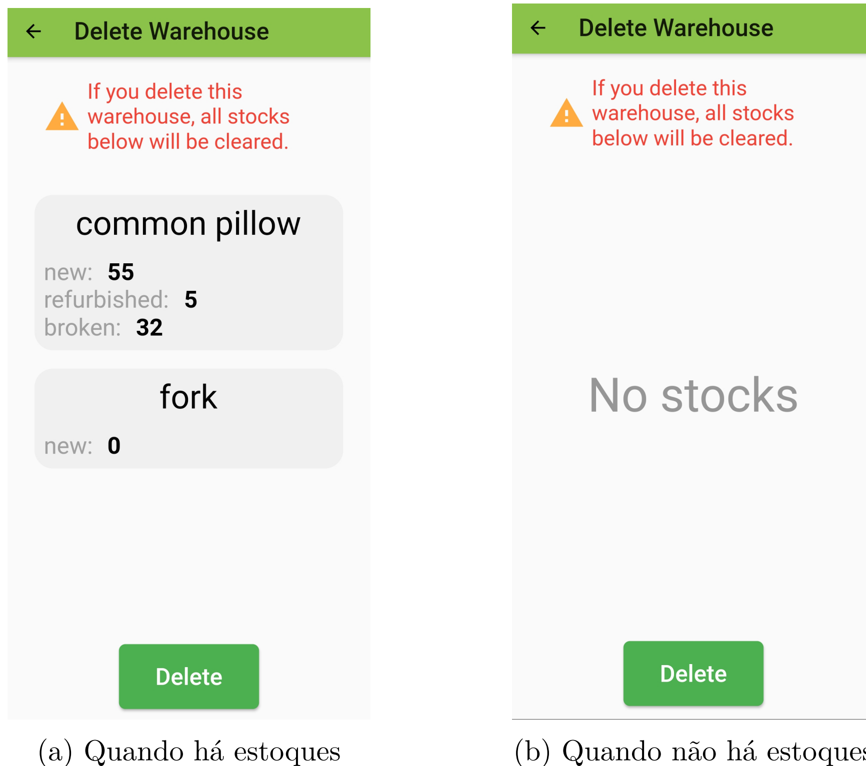
Editar um armazém

Ao apertar na ação “Edit this warehouse” na tela de detalhes de um armazém, o usuário é direcionado para tela de editar um armazém. Os mesmos campos para criar um armazém, podem ser editados. A Figura 4.26 ilustra esta tela.

Figura 4.26: Tela de editar um armazém

Excluir um armazém

Ao apertar na ação "Delete this warehouse" na tela de detalhes de um armazém, o usuário é direcionado para tela de excluir um armazém. A Figura 4.27a ilustra o caso em que há estoques de itens e a Figura 4.27b o caso que não há estoques de itens. A lógica da exclusão de um armazém é a mesma de excluir um item. Os estoques são zerados e o armazém não é excluído totalmente do banco de dados.



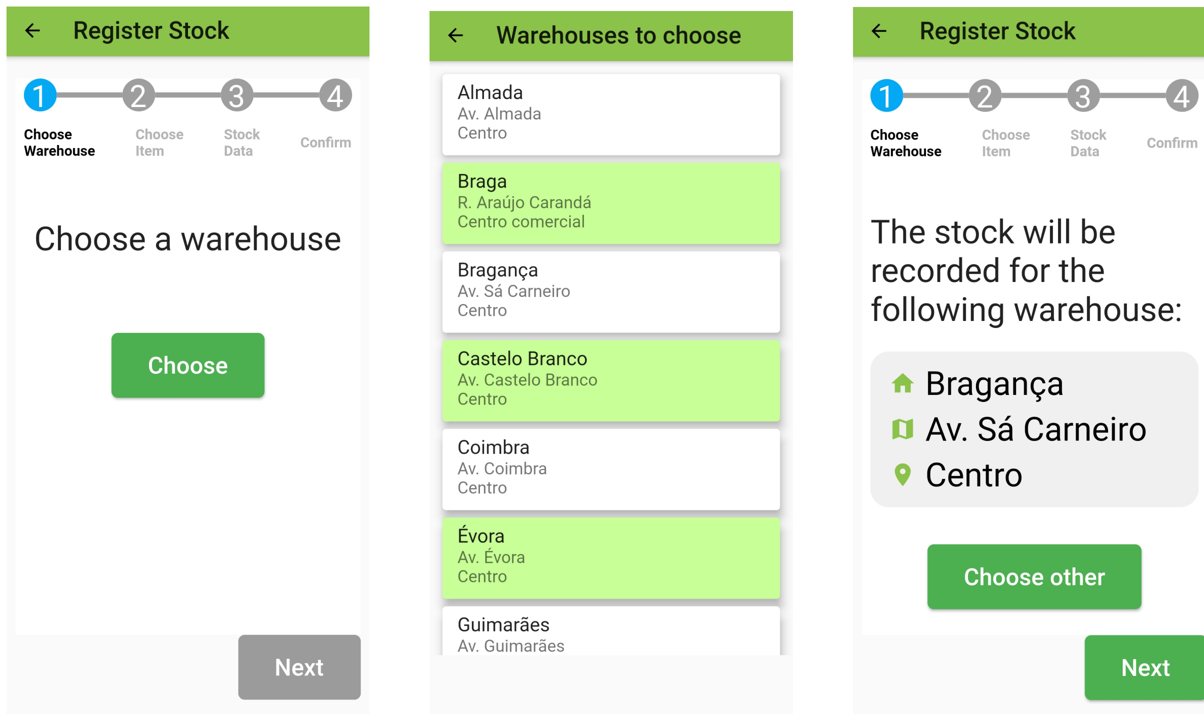
(a) Quando há estoques

(b) Quando não há estoques

Figura 4.27: Tela de excluir um armazém

Registrar um estoque de um item

Ao apertar o botão "Register stock" do menu de estoques, o usuário é encaminhado para uma tela que auxilia registrar um estoque. A tela possui 4 etapas: escolha de um armazém, escolha do item, inserir informações do estoque e confirmação. A Figura 4.28a ilustra a primeira etapa antes de escolher um armazém. Ao apertar em "Choose", é mostrada a tela que é exemplificada pela Figura 4.28b que é a lista de armazéns. E após escolher um armazém, mostra-se os dados do armazém escolhido, como demonstra na Figura 4.28c. Somente após a escolha de um armazém, o usuário pode passar para a próxima etapa.



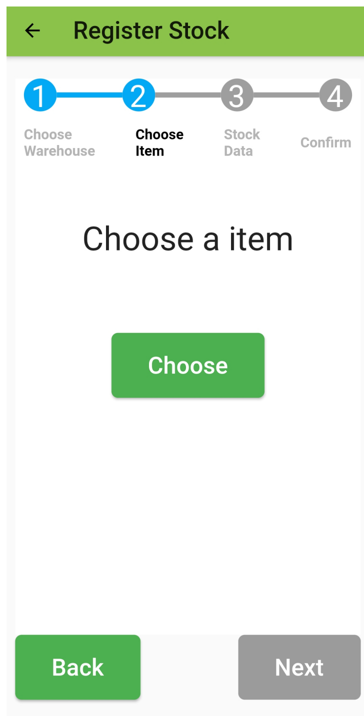
(a) Antes de escolher um armazém

(b) Durante a escolha de um armazém

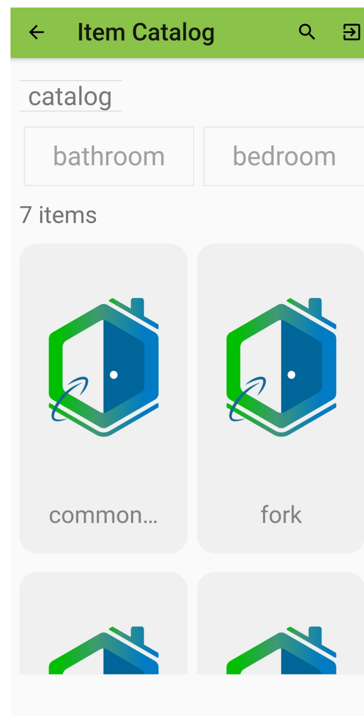
(c) Após escolher um armazém

Figura 4.28: Etapa de escolher um armazém

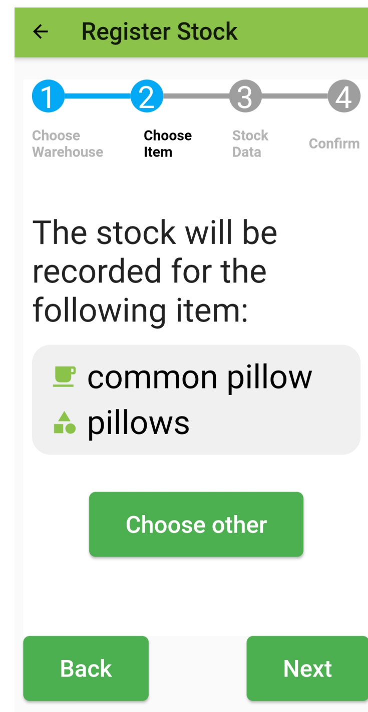
O segundo passo é escolher um item. A Figura 4.29a ilustra a tela antes de escolher um item. Ao apertar em “Choose”, o usuário irá para a tela de catálogo de item para escolher um item. A ação que ocorre ao apertar um item é diferente neste caso. Ao apertar, será escolhido o item e não mostrar os detalhes de um item. A Figura 4.29b mostra a tela durante a escolha de um item. No final desta etapa, o item escolhido é mostrado como na Figura 4.29c. Assim, o usuário pode seguir para a próxima etapa.



(a) Antes de escolher um item



(b) Durante a escolha de um item



(c) Após escolher um item

Figura 4.29: Etapa de escolher um item

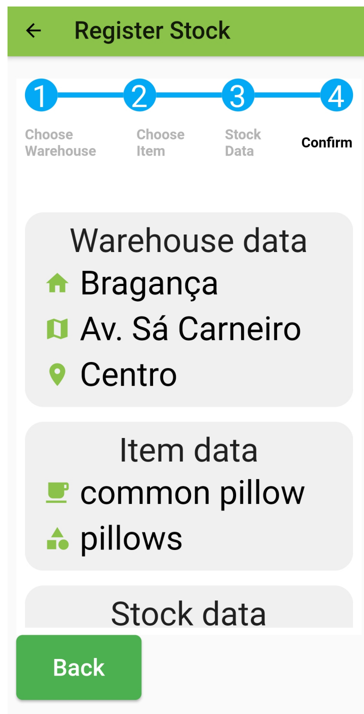
O terceiro passo é inserir as informações do estoque que são: condição do estoque (se é novo, usado, restaurado ou quebrado), quantidade, localização e descrição. A localização e descrição são opcionais. A Figura 4.30a mostra a tela antes de inserir os dados e a Figura 4.30b com os dados inseridos.

(a) Antes de escolher inserir as informações

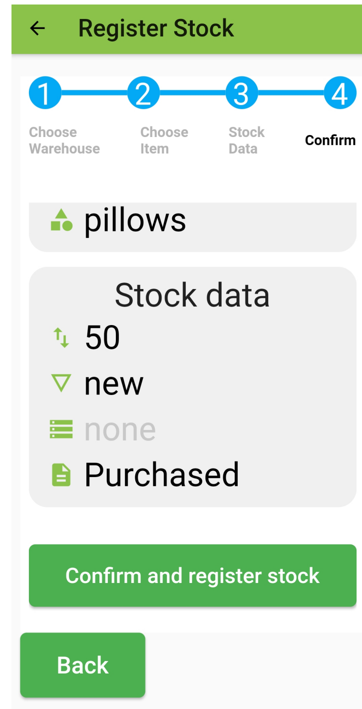
(b) Depois de inserir as informações

Figura 4.30: Etapa de inserir informações do estoque

O quarto e último passo é confirmar os dados inseridos. A Figura 4.31 mostra esta última etapa.



(a) Parte 01 de 02

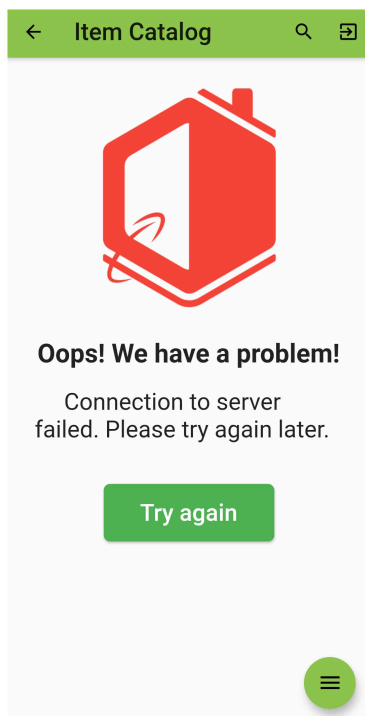


(b) Parte 02 de 02

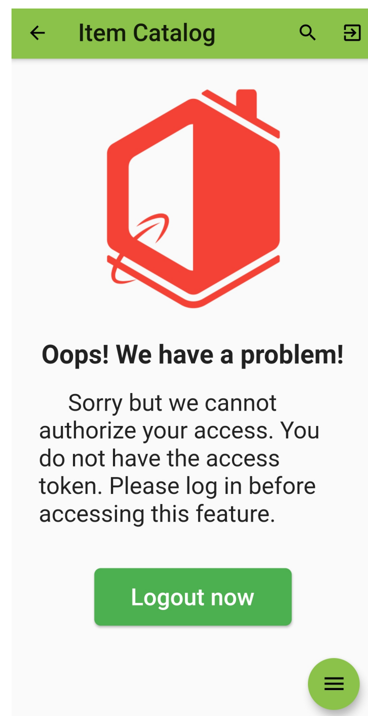
Figura 4.31: Etapa de confirmação das informações escolhidas e inseridas

Tela de erros

Quando ocorre algum erro pelo lado do servidor mostra-se uma tela de erro junto com um botão de ação. A Figura 4.32a ilustra quando ocorreu um erro de conexão com um botão de "tentar novamente", e a Figura 4.32b mostra a tela quando há problema de autenticação. Normalmente as mensagens de erro são enviadas pelo servidor.



(a) Quando há problema com conexão



(b) Quando há problema com autenticação

Figura 4.32: Telas de erros

Capítulo 5

Conclusões e Trabalhos Futuros

Um meio para otimizar os trabalhos de uma empresa é implantar a tecnologia da informação. O ambiente de um armazém pode desfrutar dos benefícios destes implantes de tecnologias. Com este intuito foi implementado um sistema de gerenciamento de armazém que é composto por uma aplicação para dispositivos móveis, uma API REST e um banco de dados. A aplicação que é utilizado nos celulares e tablets, oferece as funcionalidades de gerenciamento para os funcionários do armazém. A API recebe as solicitações da aplicação e realiza operações de consulta ou mudança no banco de dados.

Com o objetivo de compreender melhor os trabalhos do armazém e realizar os levantamentos de funcionalidades que o sistema deve prover, foi feito uma etnografia na empresa. Entre as funcionalidades identificadas nos requisitos funcionais, somente algumas funcionalidades que consideradas essenciais foram implementadas como: gerenciamento de utensílios, categorias destes, armazéns e registração de estoques de utensílios.

O aplicativo foi implementado utilizando o framework Flutter que possui a vantagem de gerar aplicativos de código nativo para Android e iOS. A API REST foi implementada em TypeScript que traz funcionalidades complementares ao JavaScript como interfaces, tipos genéricos, etc. e melhorando a detecção de erros de tipos ao compilar para código JavaScript. Para facilitar a configuração e comunicação com o banco de dados, foi utilizado o Sequelize como ORM e o RDBMS foi escolhido o MySQL.

Como o sistema tem um potencial de tornar complexo com base nos requisitos funcionais detectados, escolheu-se a Arquitetura Limpa para prover manutenibilidade e testabilidade do sistema.

Diante disso, o sistema não está completo, porém há uma base sólida que é capaz de estender e implementar novas funcionalidades. Como trabalhos futuros são recomendados:

- adicionar as funcionalidades que não foram implementadas como: gerenciamento de kit e seus templates, armazenamento de bagagens dos clientes, auxílio na montagem de kit, etc;
- implementação de uma versão web com o Flutter para não depender somente de tablets e celulares;
- melhorar o design das telas do aplicativo para trazer uma experiência melhor aos funcionários;
- controle de permissões sobre a gerência do armazém através de APIs que facilitam esta funcionalidade.

Bibliografia

Alistair Cockburn (2005). Hexagonal-architecture-with-adapters. [Online; accessed April 27, 2020].

Bracha, G. (2016). *The Dart Programming Language*. Always learning. Addison-Wesley.

Brown, E. (2019). *Web Development with Node and Express: Leveraging the JavaScript Stack*. O'Reilly Media.

Buckett, C. (2013). *Dart in Action*. Manning Publications.

Celko, J. (2014). *Joe Celko's SQL for Smarties: Advanced SQL Programming*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science.

Chacon, S. and Straub, B. (2014). *Pro Git*. Apress.

Clow, M. (2019). *Learn Google Flutter Fast: 65 Example Apps*. Independently Published.

Cockburn, A. (2005). Hexagonal architecture. `https://alistair.cockburn.us/hexagonal-architecture/`. [Online; acessado 27 de Janeiro de 2020].

Coplien, J. and Bjørnvig, G. (2010). *Lean Architecture: for Agile Software Development*. Wiley.

Crockford, D. (2008). *JavaScript: The Good Parts: The Good Parts*. O'Reilly Media.

Deitel, P. and Deitel, H. (2015). *Java how to Program: Early objects*. Always learning. Pearson.

Deitel, P., Deitel, H., and Deitel, A. (2014). *Android: How to Program*. How to program series. Pearson.

DuBois, P. (2014). *MySQL Cookbook*. O'Reilly.

Flanagan, D. (2011). *JavaScript: The Definitive Guide: Activate Your Web Pages*. O'Reilly Media.

GitLab (2020a). Gitlab documentation. `https://docs.gitlab.com/ee/user/index.html`. [Online; acessado 20 de Janeiro de 2020].

GitLab (2020b). History of gitlab. `https://about.gitlab.com/company/history/`. [Online; acessado 20 de Janeiro de 2020].

Ingeno, J. (2018). *Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts*. Packt Publishing.

Jacobson, I., Jacobson, I., Christerson, M., Staff, A. P., Jonsson, P., and Övergaard, G. (1992). *Object-oriented Software Engineering: A Use Case Driven Approach*. ACM Press Series. ACM Press.

Jemerov, D. and Isakova, S. (2017). *Kotlin in Action*. Manning Publications.

Kochan, S. (2011). *Programming in Objective-C*. Developer's Library. Pearson Education.

Kouraklis, J. (2019). *Introducing Delphi ORM: Object Relational Mapping Using TMS Aurelius*. Apress.

Levin, J. (2015). *Mac Os X and Ios Internals: To the Apple's Core*. John Wiley & Sons.

Martin, R. (2018). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Martin, Robert C. Prentice Hall.

Meier, R. and Lake, I. (2018). *Professional Android*. Wiley.

Microsoft (2016). Typescript language specification. https://github.com/microsoft/TypeScript/blob/master/doc/spec.md. [Online; accessado 16 de Dezembro de 2019].

Mouat, A. (2015). *Using Docker: Developing and Deploying Software with Containers*. O'Reilly Media.

Napoli, M. (2019). *Beginning Flutter: A Hands On Guide to App Development*. Wiley.

Nickoloff, J. and Kuenzli, S. (2019). *Docker in Action*. Manning Publications.

Rubin, A. (2007). Where's my gphone? https://googleblog.blogspot.com/2007/11/wheres-my-gphone.html. [Online; accessado 16 de Dezembro de 2019].

Sequelize (2020). Sequelize orm. https://sequelize.org/. [Online; acessado 20 de Janeiro de 2020].

Shaw, M. and Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. An Alan R. Apt book. Prentice Hall.

Walrath, K. and Ladd, S. (2012). *Dart: Up and Running*. Nutshell handbook. O'Reilly.

Windmill, E. (2019). *Flutter in Action*. Manning Publications.