IFSP - Instituto Federal de Educação, Ciência e Tecnologia
Câmpus São Paulo

Alkindar José Ferraz Rodrigues	SP3029956
Carolina de Moraes Josephik	SP3030571
Fabio Mendes Torres	SP3023184
Gabriely de Jesus Santos Bicigo	SP303061X
Leonardo Naoki Narita	SP3022498
Mariana da Silva Zangrossi	SP3030679

Lixt

São Paulo - SP - Brasil

Junho de 2021

IFSP - Instituto Federal de Educação, Ciência e Tecnologia
Câmpus São Paulo

Alkindar José Ferraz Rodrigues	SP3029956
Carolina de Moraes Josephik	SP3030571
Fabio Mendes Torres	SP3023184
Gabriely de Jesus Santos Bicigo	SP303061X
Leonardo Naoki Narita	SP3022498
Mariana da Silva Zangrossi	SP3030679

Lixt

Desenho de aplicação para desenvolvimento na disciplina de Projeto Integrado I no 1° semestre de 2021.

Professor: Ivan Francolin Martinez

Professor: José Braz de Araujo

IFSP - Instituto Federal de Educação, Ciência e Tecnologia
Câmpus São Paulo

Tecnologia em Análise e Desenvolvimento de Sistemas

PI1 - Projeto Integrado I

São Paulo - SP - Brasil

Junho de 2021

Resumo

O presente documento tem como objetivo apresentar o desenho do projeto Lixt, solução tecnólogica que visa facilitar a gestão de controle de gastos em mercado e gerenciamento das listas de compras individuais ou colaborativas. A plataforma dá autonomia para o usuário gerir listas, membros de uma lista, produtos, categorias dos produtos e compras. O Lixt ainda se propõe a oferecer a análise das compras realizadas e fornecer o histórico de compras para que o usuário consiga obter melhor controle sobre suas despesas de mercado. Para implementação da proposta utilizamos o banco de dados MySQL com back-end construído em Java aliado ao Spring e ao Hibernate. Na implementação mobile utilizamos Javascript com o framework React-Native. A hospedagem dos recursos ficará na plataforma Amazon Web Services. Para gerenciarmos a equipe durante o projeto optamos pela utilização do scrum e do kanban.

Palavras-chaves: lista de compras. compras. organização financeira.

Abstract

This document aims to present the design of the Lixt project, a technological solution that aims to facilitate the management of expenses control in groceries and management of individual or collaborative shopping lists. The platform eanbles the user to manage lists, list members, products, product categories and purchases. Lixt also proposes to offer an analysis of the purchases made and provide the purchase history so that the user can obtain better control over their groceries expenses. To implement the proposal, we decided to use the MySQL database and to develop the back-end in Java together with Spring and Hibernate. In the mobile implementation it uses Javascript with the React-Native framework. We chose to host the resources on the Amazon Web Services platform. To manage the team during the project, we chose to use scrum and kanban.

Keywords: shopping list. groceries. financial organization.

Lista de ilustrações

Figura 1 – Arquitetura Lixt 17
Figura 2 – Modelo Entidade-Relacionamento 25
Figura 3 – Detalhamento da Arquitetura 31

Lista de quadros

Quadro 1 – 1.5: Uma comparação dos aplicativos concorrentes. 12
Quadro 2 – Requisitos funcionais 21
Quadro 3 – Requisitos Não funcionais 22
Quadro 4 – Dicionário de Dados: tb_user 27
Quadro 5 – Dicionário de Dados: tb_list 27
Quadro 6 – Dicionário de Dados: tb_listMembers 27
Quadro 7 – Dicionário de Dados: tb_category 28
Quadro 8 – Dicionário de Dados: tb_product 28
Quadro 9 – Dicionário de Dados: tb_purchaseLocal 28
Quadro 10 – Dicionário de Dados: tb_purchase 29
Quadro 11 – Dicionário de Dados: tb_itemOfPurchase 29
Quadro 12 – Dicionário de Dados: tb_productOfList 30
Quadro 13 – Dicionário de Dados: tb_comment 31

Lista de abreviaturas e siglas

SDK	Software Development Kit — Kit de desenvolvimento de software. Citado em 2.3
HTML	HyperText Markup Language — Linguagem de Marcação de Hipertexto. Citado em 2.3
CSS	Cascading Style Sheets — Folhas de Estilo em Cascata. Citado em 2.3
API	Application Programming Interface — Interface de progragramação de Aplicação. Citado em 3.1
HTTPS	Hypertext Transfer Protocol — Protocolo seguro de transferência de hypertexto. Citado em 3.1
REST	Representational State Trasfer — Transferência de Representação deEstado: modelo de transferência de dados no qual o estado de um objeto é serializado e transferido entre aplicações. Citado em 3.1
ORM	Object–relational mapping — Mapeamento objeto-relacional. Citado em 3.3.2.1
MVP	*Minimum Viable Product* — Mínimo Produto Viável. Citado em 3.2.0.1
LGPD	Lei Geral de Proteção de Dados. — Citado em 3.7

Sumário

1 INTRODUÇÃO 9
1.1 Contextualização 9
1.2 Problematização 9
1.3 Justificativa 10
1.4 Objetivos 10
1.5 Análise da Concorrência 11

2 REVISÃO BIBLIOGRÁFICA 13
2.1 Organização financeira 13
2.2 Listas 13
2.3 Aplicativo *Mobile* 14

3 DESENVOLVIMENTO DA APLICAÇÃO 16
3.1 Arquitetura 16
3.2 Escopo do Projeto 17
3.2.0.1 Organização: POC e MVP 20
3.2.0.2 Requisitos Funcionais 20
3.2.0.3 Requisitos Não Funcionais 21
3.3 Tecnologias Utilizadas 22
3.3.1 Linguagens 22
3.3.1.1 Back-end 22
3.3.1.2 Mobile 23
3.3.2 Frameworks e ORMs 23
3.3.2.1 Back-end 23
3.3.2.2 Mobile 23
3.3.3 Banco de dados 23
3.3.4 Gerenciamento de tarefas 24
3.3.5 Versionamento 24
3.4 Modelagem e Definições Técnicas 24
3.4.1 Modelo Entidade-Relacionamento (MER) 24
3.4.1.1 Padrões de Banco de Dados 24
3.4.1.2 Apresentação do MER 25
3.4.1.3 Definição de Índices 26
3.4.1.4 Dicionário de Dados 26
3.4.2 Padrão Arquitetural 31

3.4.3 Serviços de Apoio . . . 32
3.5 Escalabilidade . . . 33
3.6 Manutenibilidade da aplicação . . . 33
3.6.1 *Logs* . . . 33
3.6.2 Integração Contínua . . . 34
3.6.3 *Code Conventions* . . . 34
3.6.4 *Designs Patterns* . . . 35
3.6.4.1 *Clean Code* . . . 35
3.6.4.2 *SOLID* . . . 36
3.6.4.3 *Gang Of Four* . . . 36
3.6.5 Testes . . . 37
3.7 Segurança, Privacidade e Legislação . . . 37
3.8 Viabilidade Financeira . . . 38
3.8.1 Gerenciamento de Custos . . . 38
3.8.1.1 Análise de Requisitos e Desenvolvimento . . . 38
3.8.1.2 Manutenções . . . 39
3.8.2 Custos de deploy e de Ambiente de Produção . . . 39
3.8.2.1 Frontend . . . 39
3.8.2.2 Backend . . . 39
3.8.2.3 Banco de Dados . . . 39
3.8.3 Medidas de Obtenção de Retorno Financeiro . . . 40
3.8.4 Conclusão . . . 40

4 PLANEJAMENTO E GERENCIAMENTO DO PROJETO . . . 41
4.1 Metodologia de Gestão e Desenvolvimento de Projeto . . . 41
4.2 Organização da Equipe . . . 41
4.3 Gestão de Tempo . . . 42

REFERÊNCIAS . . . 43

GLOSSÁRIO . . . 43

1 Introdução

1.1 Contextualização

Comprar é um ato essencial no cotidiano das pessoas. Não apenas por uma questão de sobrevivência, como a compra de alimentos, medicamentos, roupas, imóveis, móveis, entre outros, mas também para lazer. Quando trata-se de compras essencias, as compras de supermercado estão no topo da lista para a maioria das pessoas, pois costumam ser compras muito frequentes (semanalmente, 2 vezes por semana, mensalmente) e com alto volume de produtos, que não envolvem apenas alimentação em si, mas também produtos de higiene da casa e pessoal.

Devido a sua grande importância na vida das pessoas, as compras de supermercados devem ter sua devida atenção e gerenciamento, principalmente por questões financeiras e organização, pois o principal objetivo de muitos consumidores é comprar mais produtos de qualidade gastando menos.

1.2 Problematização

Por ser algo extremamente comum e cotidiano na vida de todas as famílias do país, a tendência é que as pessoas não saibam administrar seus gastos em supermercados, já que as compras ocorrem frequentemente. Atualmente, a forma universal de verificar os gastos de uma compra de um supermercado no Brasil é através do cupom fiscal, um documento frágil que contém todos os produtos comprados e seus respectivos preços. Sua fragilidade é devido a sua composição, o que ocasiona o desaparecimento do texto ao decorrer do tempo. Ao se depender do cupom fiscal para analisar e atualizar os gastos, ocorrem alguns dos seguintes problemas:

- Só é possível fazer uma análise de gastos após a compra, sendo que o consumidor teria grandes benefícios ao analisar os gastos e produtos durante a compra.
- Dificuldade de lembrar e guardar os preços dos alimentos e produtos durante a compra, para futura análise que auxilie o comprador a fazer compras mais econômicas e de qualidade.
- É necessário repassar manualmente todos os dados de cada cupom fiscal para um documento centralizado, se caso o consumidor queira manter um histórico de compra ou fazer uma análise geral.

- Dificuldade em analisar preços de diferentes estabelecimentos de modo a decidir qual o melhor custo-benefício julgado pelo comprador, já que essa informação vai provavelmente constar em algum meio não centralizado, ou seja, o consumidor irá perder a informação rapidamente.
- Dificuldade em analisar e calcular quais as categorias de produtos que possuem maior gasto, menor gasto, ou uma média, baseados em determinado período, que visa facilitar a descoberta de quais produtos são mais comprados, quais são mais caros, a fim de auxiliar o usuário em futuras compras.
- Dificuldade de gerenciar compras colaborativas, cada participante não sabe exatamente o que será comprado ou quais itens serão de sua responsabilidade, a não ser que essas informações sejam compartilhadas através de meios que a ação manual seria necessária, como mandar uma mensagem através de um aplicativo.

1.3 Justificativa

Como é possível perceber com base nos problemas descritos da seção acima, o gerenciamento de compras, quando feito, é uma atividade muito cansativa e manual. Com a frequência de compras somando a dificuldade de administrá-las, as pessoas não analisam e gerenciam suas compras, o que as fazem gastar cada vez mais e não saberem o motivo de tanto gasto, pois não conseguem fazer uma análise de compras detalhada.

Tendo em vista a economia do Brasil, muitas famílias tentam economizar com grande esforço, tanto para guardar o restante do dinheiro no final do mês, como também para sobreviver em casos em que a renda é muito baixa. Então, o gerenciamento e administração de compras é extremamente essencial para o presente e futuro de famílias e pessoas que se encontram com o dinheiro contado ou que precisam investir determinada parte do valor para os filhos, casa, pagamento de dívidas, entre outros.

1.4 Objetivos

Para solucionar todos os problemas citados e no que eles acarretam, entregando autonomia, controle de compras para os consumidores de maneira intuitiva e fácil, em que todas as informações que eles registrem sejam armazenadas em um local centralizado, criamos o Lixt, uma solução que facilite a vida do consumidor que reside no Brasil, para o gerenciamento e controle de suas compras através de criação e edição de listas de compras compartilhadas ou individuais. A solução será focada em facilitar as compras alimentícias, feitas em supermercados, feiras, lojas de conveniência, e até mesmo em aplicativos de alimentos, como IFood, Uber Eats, Rappi, e entre outros.

Além disso, o Lixt se propõe em oferecer análises de compras por determinado período escolhido pelo usuário, separar produtos em categorias e até mesmo apresentar análises quanto a variação de preço entre estabelecimentos, que poderão ser adicionados de acordo com a preferência do usuário, além de outras funcionalidades que estarão melhores descritas na seção de Escopo.

1.5 Análise da Concorrência

Auditamos soluções que existem atualmente no mercado e, ao verificar as aplicações existentes, conclui-se que há intersecções nas funções dentre os aplicativos analisados. As funções mais básicas, como gerenciamento de itens e gerenciamento de listas, estão presentes em todos, tendo em vista que são essenciais em qualquer aplicativo de lista. Outras funções básicas que deveriam ser incluídas em qualquer aplicação de lista, como gerenciamento de categorias e compartilhamento de listas, não estão presentes em todos os aplicativos analisados.

Contudo, as divergências ficam claras quando analisamos o mecanismo das aplicações, entre elas destacam-se o *Mealime* e o *Cozi Family Organizer* que, apesar de serem voltados para as compras, cumprem também outras funcionalidades. O *Mealime*, cujo foco é o planejamento de refeições, e o *Cozi Family Organizer*, cujo foco é o planejamento familiar, deixam a desejar nas funções relacionadas às compras.

Entre os outros aplicativos analisados, é perceptível que não possuem todas as funcionalidades propostas nesse documento, principalmente quando se trata de compartilhamento de listas, uma vez que cada software lida de modo diferente diante dessa feature. O *SoftList*, por exemplo, permite o compartilhamento de lista, porém não é capaz de ser gerenciada por mais de um usuário, sendo apenas importada para o usuário no qual a lista está sendo compartilhada.

Ao analisar os aplicativos mais populares da categoria, constatamos que o *Out Of Milk*, *Bring!* e o *OurGroceries*, que são destaques na área, não se propõem a exibir análise estatística das compras do usuário e nem manter um histórico do que foi comprado. O quadro 1.5 permite visualizar melhor as diferenças entre os concorrentes.

Quadro 1 – 1.5: Uma comparação dos aplicativos concorrentes.

	Cozi Family Organizer	OurGroceries	Softlist	Out of Milk	Mealime	Bring	Lixt
Login/Cadastro	x	x	x	x	x	x	x
Categorias		x	x	x		x	x
Compartilhamento de listas	x		x	x		x	x
Atribuição de itens							x
Gerenciamento de compras			x				x
Historico de compras			x				x
Análise de compras			x				x
Calculadora			x	x		x	x
Comentários							x

Fonte: os autores

2 Revisão Bibliográfica

Nesta seção, os tópicos conceituais e teóricos relacionados ao projeto a ser desenvolvido que irão auxiliar o entendimento do mesmo serão descritos de forma a esclarecer tanto as partes de negócio, como as técnicas. Dessa forma, a compreensão do projeto como todo será mais fácil para o leitor.

2.1 Organização financeira

De acordo com o dicionário Michaelis (MELHORAMENTOS, 2021), a palavra organização significa preparação de um projeto, com definição de procedimentos e metas. Assim, pode se dizer que organização financeira trata-se de cuidar das finanças (podendo elas ser empresarial, familiar, ou pessoal) afim de atingir um objetivo.

Quando trata-se de finanças pessoais ou familiares, ter uma boa organização financeira significa conhecer quais e quanto são suas despesas e receitas mensais (COLELLA, 2015). De acordo com o mesmo artigo, é necessário planejar como, quanto e onde a despesa será feita, além de fazer levantamento de preços.

Pode-se dizer então, que as compras, categorizadas como despesas fixas, devido a sua presença frequênte em ambientes pessoais e familiares, precisam ser planejadas e organizadas para impulsionar uma melhor organização financeira entre os envolvidos da compra.

2.2 Listas

As listas de afazeres, comumente chamadas de *to do list* em inglês, é uma lista de lembretes textuais, como por exemplo, "Ir ao médico", "Comprar caixa de 12 ovos", entre outros (CONLEY, 2007). Essas listas são encontradas em qualquer âmbito, e auxiliam a lembrar de tarefas ou coisas importantes que precisam sobrer uma ação do indívidio, por exemplo, comprar, completar, finalizar.

Seguindo o mesmo príncipio, uma lista de compras se refere à uma lista contendo nomes de produtos nos quais um indivíduo deseja adquirir, podendo conter ou não um limite de gasto total. As listas, além de ser um método eficaz para se organizar e lembrar do que foi escrito, é considerada uma ótima maneira de evitar que alguém ceda à produtos nos quais não quer comprar por serem prejudicias à saúde da pessoa (AU, 2013) ou por qualquer outro motivo.

Existem três tipos de compras (ANGELO, 2003), sendo que dois tipos podem ter a presença de uma lista de compras:

1. Completamente planejada, incluindo na lista de compras os produtos e suas respectivas marcas.
2. Parcialmente planejada, incluindo apenas os produtos a serem comprados mas a marca deles serão decididas no ato da compra.

As compras completamente planejadas são consideras com pouco envolvimento emocional do consumidor, enquanto as parcialmente planejadas, apesar de planejadas, podem ser manipuladas por algum critério exterior, como uma promoção (ANGELO, 2003). No aplicativo a ser desenvolvido, uma lista de compras pode ser tanto completamente planejada ou parcialmente planejada, de acordo com a necessidade do usuário que está utilizando a plataforma. Uma vez que a lista de compras existe, a compra em si se torna planejada.

2.3 Aplicativo *Mobile*

Um aplicativo *mobile* trata-se de uma aplicação de software que é desenvolvida exclusivamente ou acessável por um dispositivo móvel, como celulares e *smartphones*. O uso dos dispositivos móveis cresceu disparadamente nos últimos anos, e é o principal meio de acesso no Brasil desde 2017 (ADJUTO, 2018).

Atualmente, existem dois sistemas operacionais que são mais utilizados nas plataformas *mobile*: o *Android* e o *IOS*, tratados como concorrentes das marcas *Google* e *Apple*. Os aplicativos desenvolvidos para os dois sistemas operacionais são divididos em dois tipos:

- Aplicativos nativos.
- Aplicativos híbridos.

Os aplicativos nativos são desenvolvidos utilizando o SDK do sistema operacional em questão, e não pode ser executados em outros ambientes. Por exemplo, um aplicativo desenvolvido com o SDK da empresa *Apple* não pode ser executado em ambiente *Android* e vice-versa (SEUNG-HO, 2015).

Já os aplicativos híbridos são desenvolvidos utilizando tecnologias web, como HTML, CSS e *Javascript*, e por isso, podem ser executados em qualquer sistema operacionais móvel através de um container nativo. Apesar de poder ser executado em diversos dispositivos através de apenas um código, os aplicativos híbridos possuem certa limitações ao acessar recursos nativos do dispositivo, como câmeras, microfones e sistemas internos

de armazenamento e memória, porém, atualmente já existem soluções que facilitam a comunicação entre aplicativo híbrido e o dispositivo, como o *React Native* (SEUNG-HO, 2015).

3 Desenvolvimento da Aplicação

Neste capítulo descrevemos os conceitos, análises e ferramentas utilizadas pela equipe TGT para o desenvolvimento do porduto Lixt, incluindo os requisitos do projeto, as tecnologias utilizadas, e a arquitetura e a modelagem do produto. Isto é apresentado para que se possa estabelecer parâmetros e métricas que guiarão o desenvolvimento e a entrega final do projeto.

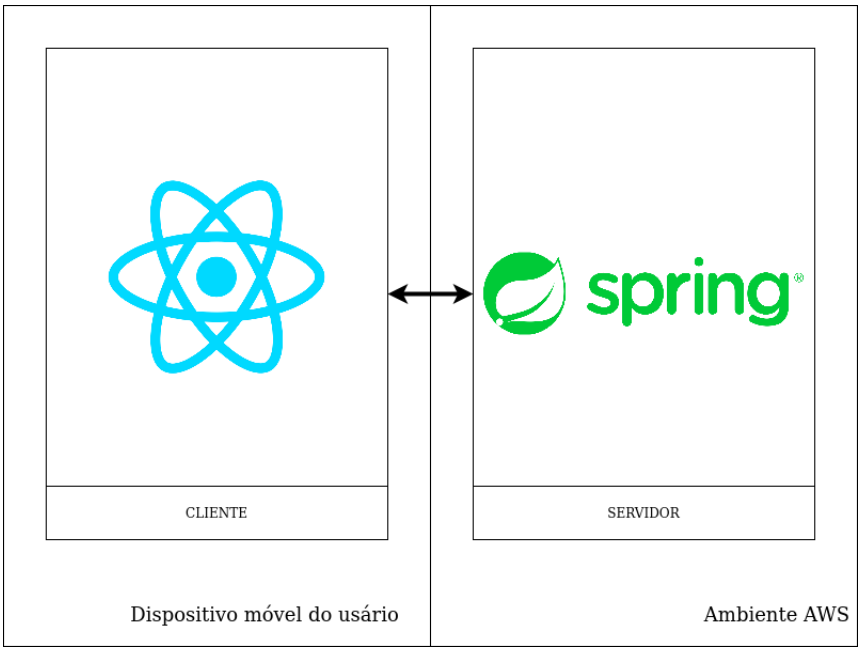
3.1 Arquitetura

Com base na análise do projeto, e nos requisitos que foram levantados como necessários, a arquitetura cliente-servidor é plausível como modelo para o produto que pretendemos entregar. Esta arquitetura é composta por duas aplicações distintas:

- Uma aplicação front-end, focada na interação com o usúario e apresentação de dados de uma forma agradável e intuitiva. Esta aplicação será implementada em JavaScript, com o framework React Native, e disponibilizada para as plataformas iOS e Android.
- Uma aplicação back-end, que será responsável por tratar os dados coletados no front-end e disponibilizar as informações que serão mostradas aos usuários. Como esta aplicação requer uma lógica de servidor, estabilidade e ampla disponiblidade, esta aplicação será implementada em Java, com uso do framework Spring Boot, que abstrai a criação de um servidor.

Podemos ver na Figura 1 uma respresentação desta arquitetura.

Figura 1 – Arquitetura Lixt



A comunicação entre estes serviços será feita com o uso do protocolo HTTPS, que permite a aplicação cliente realizar chamadas ao servidor através de urls, seja para buscar informações para apresentar ao usuário ou postar informações coletadas dele. O framework Spring, além de abstrair a implementação da lógica de um servidor, implementa *listeners* para estas urls, auxiliando a criação de pontos na aplicação do servidor focados na comunicação com a aplicação cliente.

O uso do protocolo HTTPS oferece alguamas vantagens a aplicação front-end, que não precisa esperar uma solicitação ao back-end ser finalizada antes de realizar outras solicitações, aumentando a repsonsividade da aplicação cliente. Para além disso, quando combinada ao modelo REST na construção da API, o protocolo HTTP oferece meios eficientes para que as aplicações se comuniquem.

Como plataforma de servidor, o serviço AWS será utilizado, uma vez que é oferecida de maneira gratuita para a realização deste projeto, e nele serão armazenadas algumas instâncias da aplicação back-end. Esta redundância é necessária como forma de garantir a estabilidade do sistema, de forma que sempre haja alguma disponível para o processamento de novas requisições, e, em caso de falha numa delas, o serviço não seja interrompido aos usuários.

3.2 Escopo do Projeto

Lixt é um aplicativo para gerenciamento de listas de compras compartilhadas ou não.

O aplicativo vai seguir a dinâmica de uso abaixo:

1. O usuário cria uma lista de compras e insere todos os itens antes da compra;
2. O usuário inicia um carrinho de compras quando chega ao mercado, nesse momento ele tem a opção de importar para aquele carrinho os itens das listas que ele possui em aberto (que ainda não foram finalizadas), marcando quais listas ele deseja que sejam incluídas. Com o carrinho de compras ele poderá anotar os preços dos itens, riscá-los e ver o total gasto. Ao finalizar a compra as listas são atualizadas, e já aparecem riscados os itens que já foram comprados;
3. Quando o usuário quiser reutilizar uma lista ele poderá clicar em desmarcar todos os itens.

A seguir listamos as principais funcionalidades como uma lista de tópicos para facilitar a visualização de quais funcionalidades dependem de outras de forma hierárquica:

- *Login*:
  - Criar conta;
  - Redefinir senha;
  - Autenticar-se no sistema;
- Manipular uma lista:
  - Criar lista
  - Adicionar item;
  - Remover itens;
  - Deletar uma lista;
  - Limpar uma lista;
- Configurar compartilhamento de lista: convidar pessoas para a lista (apenas quem criou a lista):
  - Enviar convite:
    - Acompanhar *status* (aceito ou pendente);
    - Remover convite;
- Iniciar um carrinho de compras:
  - Selecionar listas para compor o carrinho;

  - Informar o mercado onde a compra será realizada (automaticamente através da localização, se não estiver habilitada será solicitado que o usuário insira o nome do mercado);
  - Exibir o valor total do carrinho;
  - Informar a quantidade que será efetivamente comprada naquele momento (o usuário pode ter planejado 10 unidades e apenas comprar 5 naquele momento);
  - Riscar itens;
  - Finalizar um carrinho de compras;

- Ver estatísticas:
  - Selecionar uma lista e ver o total gasto naquela lista ao longo do tempo em um gráfico de linha:
    - Selecionar um dos pontos do gráfico e ver detalhes daquela lista;
  - Selecionar uma lista para ver um gráfico de pizza com os valores médios gastos por categorias naquela lista;
  - Verificar histórico de preços de um item (tabela com nome do produto, quantidade, marca, preço, mercado e data da compra):
    - Selecionar uma lista, dentro da lista selecionada selecionar o produto para ver o histórico;

- Cadastrar item
  - Caso um item não esteja disponível no banco de dados, será possível escanear o código de barras a fim de que, ao ler, consiga cadastrar os dados de modo automático;
  - Caso não queira escanear ou ainda o produto não haja código de barra, será possível cadastrar o produto manualmente através de um formulário:
    - Definir nome;
    - Definir quantidade;
    - Definir unidade de medida (un. ml, L etc.);
    - Definir medida;
    - Adicionar uma categoria:
      - Criar nova categoria;
    - Adicionar um comentário;
    - Atribuir a um usuário (caso a lista tenha sido compartilhada e pelo menos um convite já tenha sido aceito);

- Geolocalização
  - Caso o GPS esteja habilitado no celular do usuário, poderá registrar os dados básicos (latitude, longitude) a partir do GPS;
  - Caso o GPS esteja desabilitado, será perguntado manualmente os dados do local (cep, rua, cidade) para que seja possível buscar os dados de latitude e longitude.

3.2.0.1 Organização: POC e MVP

Para a Prova de Conceito (POC), serão desenvolvidas as funcionalidades de login e manipulação da lista.

Já para o *Minimum Viable Product* (MVP), serão implementadas as funcionalidades de lista compartilhada, cadastro de item por formulário e manipulação de carrinho de compras.

As demais funcionalidades de estatísticas, geolocalização e cadastro de item por código de barras ficarão alocadas para a entrega final do projeto.

3.2.0.2 Requisitos Funcionais

Os requisitos funcionais dizem respeito às funcionalidade que o sistema deve ter. O Quadro 2 lista os requisitos funcionais, suas dependências, a sigla e a prioridade de implementação.

Quadro 2 – Requisitos funcionais

Sigla	**Descrição**	**Prioridade**	**Dependências**
RF01	*Login*: o usuário deve ser capaz de criar sua conta no aplicativo, definir sua senha e realizar o *login* no sistema.	Alta	
RF02	O sistema deve possibilitar que o usuário crie suas listas de compras e possa atribuir um nome, uma descrição e ter a opção de importar uma lista existente.	Alta	RF01
RF03	Editar uma lista: Possibilita ao usuário o gerenciamento dos itens da lista, como adicionar itens, remover e enviar convites para a lista.	Alta	RF02
RF04	Iniciar um carrinho de compras: permitir que o usuário importe várias listas de compras, informe o local da compra, o total gasto, quantidade de itens a ser comprados, riscar itens e finalizar o carrinho.	Alta	RF03
RF05	Ver estatísticas: ver o histórico de valores pagos em uma lista ao longo do tempo, ver os valores gastos por categorias em uma lista, ver o histórico de preços de um determinado item ao longo do tempo.	Média	RF04

Fonte: Os Autores

3.2.0.3 Requisitos Não Funcionais

De maneira simplificada, os requisitos não funcionais não estão relacionados diretamente às funcionalidades do sistema, mas ao seu funcionamento de um modo geral, ou seja, como ele as funcionalidade serão executadas.

No Quadro 3 estão elencados os requisitos não funcionais, cada um com sua nomenclatura, categoria e descrição.

Quadro 3 – Requisitos Não funcionais

Nomenclatura	**Descrição**	**Categoria**
RNF01	Criptografia das senhas: Por uma questão de segurança as senhas não serão armazenadas diretamente no banco, serão criptografadas antes de serem armazenadas como um *hash*.	Segurança
RNF02	Comunicação: A comunicação entre as camadas da aplicação deverá ser feita utilizando o protocolo HTTPS, para garantir a segurança no envio dos dados através da rede.	Segurança
RNF03	Responsividade: O sistema deve exibir corretamente os elementos da interface gráfica nos mais variados tamanhos de celulares.	Usabilidade
RNF04	Internacionalização: O sistema deverá suportar dois idiomas (inglês e português) e suportar que futuramente seja possível adicionar outros idiomas.	Usabilidade
RNF05	Escalabilidade: O sistema deverá ser projetado para garantir que futuras melhoras e expansões sejam possíveis.	Desempenho
RNF06	Disponibilidade: O sistema deverá estar disponível aos usuários ininterruptamente	Disponibilidade

Fonte: Os Autores

3.3 Tecnologias Utilizadas

As tecnologias que decidimos utilizar foram escolhidas a partir do conhecimento prévio da equipe, da curva de aprendizado e levando em consideração também o tamanho das comunidades que já a utilizam, visando um maior apoio e material de pesquisa. Dito isso, escolhemos as seguintes tecnologias:

3.3.1 Linguagens

3.3.1.1 Back-end

Decidimos que a linguagem para o back-end seria o Java. A linguagem se adequa à nossa proposta e atende o paradigma de linguagem orientada a objetos do qual nos foi orientado a utilizar. A comunidade de Java é extensa e ativa, contribuindo com muitos materiais e recursos. Ainda podemos destacar que a utilização da linguagem previamente pelos integrantes da equipe também foi impactante na consolidação dessa decisão.

3.3.1.2 Mobile

Para o desenvolvimento da plataforma mobile decidimos utilizar o Javascript. A linguagem possui também uma comunidade ativa e uma variedade de materiais disponíveis e atualizados. Apesar de nem todos os integrantes terem tido contato prévio, por conta da facilidade de assimilação e necessidade de poucos recursos para a configuração do ambiente de desenvolvimento, optamos pelo Javascript.

3.3.2 Frameworks e ORMs

3.3.2.1 Back-end

Para o back-end decidimos utilizar o framework Spring, usando a ferramenta Spring Boot que proporciona agilidade na criação das aplicações pois segue a filosofia de Convention over Configuration(PADMANABHAN, 2020), nos poupando de depreender muito tempo nas configurações. Não obstante, a framework facilita o desenvolvimento pois nos propicia a utilização de módulos que julgarmos necessários (como Spring MVC e Spring Data JDBC). Além disso, há uma gama vasta de materiais para consultarmos. Como ferramenta ORM decidimos usar o Hibernate pela consolidação dele no mercado e o uso amplo em aplicações Java que necessitam de mapeamento relacional dos dados. Por conta da quantidade de modelos da aplicação, julgamos necessário utilizar uma ferramenta que facilitasse esse processo.

3.3.2.2 Mobile

Na aplicação mobile decidimos utilizar o framework React-Native. Esta framework gera aplicativos nativos, não necessita de muitos recursos e configurações para montar o ambiente de desenvolvimento e possibilita um conforto maior no desenvolvimento do código por ser uma framework Javascript. Não obstante, também é uma framework com larga quantidade de recursos para consulta além de uma comunidade muito ativa.

3.3.3 Banco de dados

O banco de dados que escolhemos foi o MySQL pois precisávamos para a nossa proposta de um banco de dados relacional e que fosse possível de ser hospedado na AWS. Verificamos que o MySQL cobria não apenas esses critérios mas também possui uma ferramenta gráfica (MySQL Workbench) que facilita a visualização e a operação do banco. Além disso os integrantes da equipe já tiveram experiências com a ferramenta anteriormente.

3.3.4 Gerenciamento de tarefas

Para o gerenciamento das tarefas optamos pela ferramenta Trello por ser gratuita, de fácil manuseio e visualização. Além disso, o Trello figura entre as ferramentas que foram utilizadas com sucesso nos semestres anteriores durante o desenvolvimento de projetos.

3.3.5 Versionamento

Para o controle de versão do desenvolvimento da aplicação optamos pelo Git com repositório no Github. Esta escolha foi realizada devido à experiência prévia da equipe com a ferramenta e pela quantidade de recursos oferecidos pela plataforma na gestão do desenvolvimento de aplicações. Para o versionamento do projeto utilizamos o Apache Subversion (SVN). Concentramos no repositório fornecido pelos professores todas as entregas previstas (incluindo as versões atualizadas dos códigos do repositório externo do Github).

3.4 Modelagem e Definições Técnicas

Nesse tópicos, serão abordados os diagramas e modelagens necessários para o desenvolvimento do projeto Lixt. Além disso, serão apresentadas as definições técnicas necessárias para o prosseguimento do projeto.

3.4.1 Modelo Entidade-Relacionamento (MER)

O Modelo Entidade-Relacionamento (MER) é a primeira abstração do banco de dados, no qual é modelado e estruturado a forma no qual os dados serão trabalhados, persistidos e buscados.

3.4.1.1 Padrões de Banco de Dados

Antes de tudo, é necessário definir os padrões utilizados para definir o banco de dados. O banco de dados foi modelado em inglês com o intuito de apresentar coerência com o desenvolvimento.

Todas as tabelas começam o prefixo “tb_”. Exemplo: tb_usuario. Todas as colunas possuem o prefixo de seu tipo de dado:

- Chaves Primárias e Estrangeiras: Começam com “id_”. Exemplo: id_user.
- Dados de Texto: Começam com “st_”, de string. Exemplo: st_name.
- Dados de Data: Começam com “dt_”, de date. Exemplo: dt_createdAt.

- Dados Numéricos: Começam com “nr_”, de number. Exemplo: nr_price.
- Dados Booleanos: Começam com “bl_”. Exemplo: bl_isActive.
- Enum: Começam com “en_”, sendo que os enums serão tratados no backend. Exemplo: en_status.

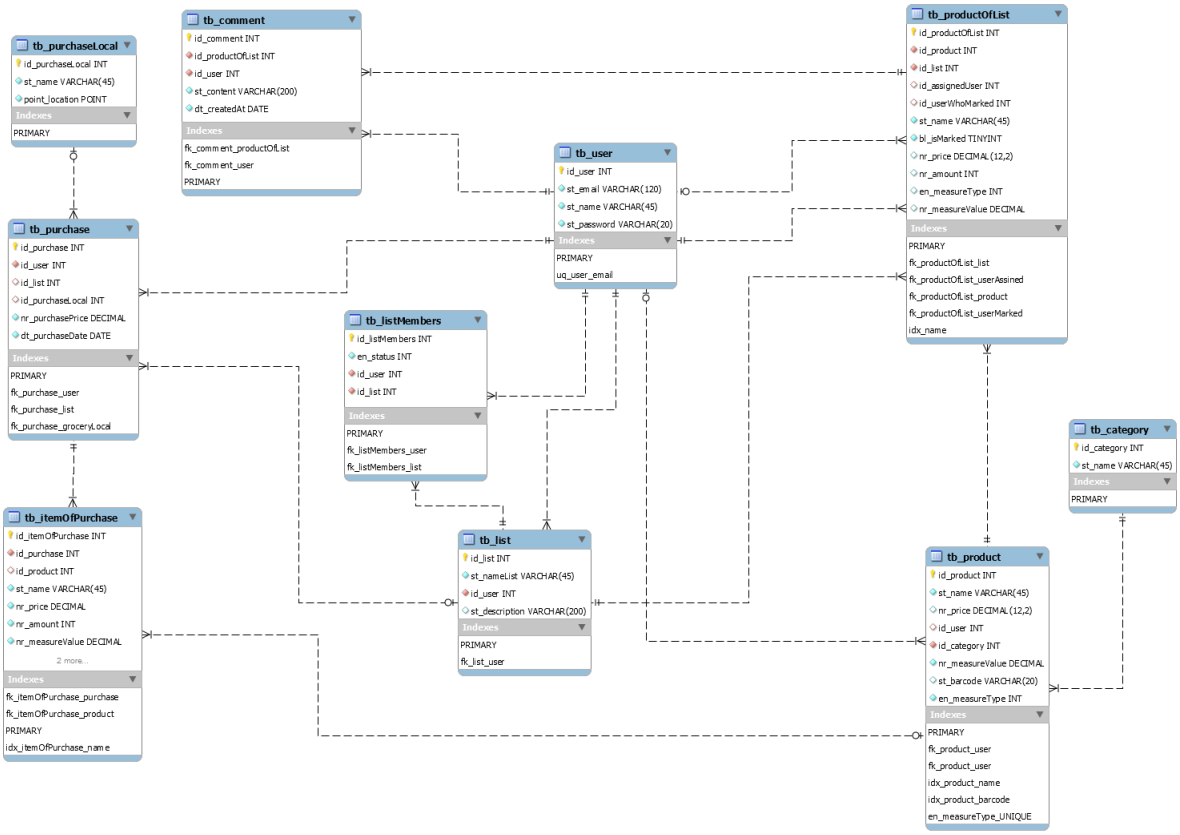
Quanto aos índices e restrições, é definido o uso dos seguintes prefixos precedidos pelo nome da tabela:

- Chave Estrangeira: Começam com “fk_”. Exemplo: fk_listMembers_user.
- Unique: Começam com “un_”. Exemplo: uq_user_email.
- Outros Índices: Começam com “idx_”. Exemplo: idx_user_name.

3.4.1.2 Apresentação do MER

Na Figura 2 está explicitado o MER proposto para a aplicação Lixt, sendo um diagrama normalizado e com as boas práticas aplicadas.

Figura 2 – Modelo Entidade-Relacionamento



Fonte: Os autores

No diagrama, uma lista (tb_list) possui vários usuário (tb_user), e um usuário pode estar em várias listas, surgindo assim a necessidade da tabela intermediária (tb_listMembers). Essa tabela intermediária possui um status, que pode ser "Aceitado", "Rejeitado" e "Aguardando Confirmação".

Um usuário pode fazer vários comentários (tb_comment) no produto da lista (tb_productOnList), e um comentário é de um usuário.

Uma lista pode ter várias compras (tb_purchase), pois uma lista pode ser feita ao decorrer de várias compras, onde uma compra é de um mercado (tb_purchaseLocal) e possui vários items (tb_itemOfPurchase).

Tanto um produto da lista quanto um item da compra é composto um produto (tb_product, sendo produto uma tabela mais genérica, pois assim será possível gerar dados estatísticos de produtos mais genéricos sem nenhuma especificidade como, por exemplo, marca do produto), sendo que o produto pode ter uma categoria (tb_category).

3.4.1.3 Definição de Índices

No MySQL, temos dois tipos de índices:

1. Índice Primário: Criado pelo MySQL automaticamente ao criar chaves primárias (primary keys - PK) ou campo UNIQUE.
2. Índice Secundário: Criado e manipulado durante a modelagem para otimização de queries.

Nesse sentido, foram definidos índices secundários para as chaves estrangeiras (foreign key - FK) e para as colunas que no qual serão buscada no frontend por meio de campos de busca.

3.4.1.4 Dicionário de Dados

O dicionário de dados é um suporte ao MER, descrevendo todas os dados, restrição e fornecendo breves descrições de cada campo de cada tabela.

Antes de tudo, foi criado siglas para simplificar os dicionários de dados apresentados: Chave Primária (PK), Chave Estrangeira (FK), Not Null (NN) e Unique (UQ). Lembrando também que toda PK é automaticamente NN, UQ e indexado.

Quadro 4 – Dicionário de Dados: tb_user

tb_user				
Coluna	**Tipo de Dado**	**Restrições de Domínio**	**Indexação**	**Descrição**
id_user	INT	PK	Primária	Campo identificador de usuário
st_name	VARCHAR(45)	NN		Campo obrigatório para guardar o nome do usuário
st_email	VARCHAR(120)	UQ, NN	Primária	Campo obrigatório para guardar o email do usuário
st_password	VARCHAR(120)	NN		Campo obrigatório para guardar o hash da senha do usuário

Fonte: Os autores

Quadro 5 – Dicionário de Dados: tb_list

tb_list				
Coluna	**Tipo de Dado**	**Restrições de Domínio**	**Indexação**	**Descrição**
id_list	INT	PK	Primária	Campo identificador da lista
st_nameList	VARCHAR(45)	NN		Campo obrigatório para conseguir apelidar uma lista
id_user	INT	NN	Secundária	Chave estrangeira obrigatória que referencia o usuário que criou a lista
st_description	VARCHAR(120)			Campo facultativo que possibilita a descrição da lista

Fonte: Os autores

Quadro 6 – Dicionário de Dados: tb_listMembers

tb_listMembers				
Coluna	**Tipo de Dado**	**Restrições de Domínio**	**Indexação**	**Descrição**
id_listMembers	INT	PK	Primária	Campo identificador da tabela intermediária entre lista e usuário
en_enum	INT	NN		Enum para verificar o status do convite para um usuário participar da lista
id_user	INT		Secundária	Chave estrangeira facultativa que referencia o usuário que criou o produto
id_list	INT	NN	Secundária	Chave estrangeira obrigatória que referencia a lista para o usuário participar

Fonte: Os autores

Quadro 7 – Dicionário de Dados: tb_category

tb_category				
Coluna	**Tipo de Dado**	**Restrições de Domínio**	**Indexação**	**Descrição**
id_category	INT	PK	Primária	Campo identificador da categoria
st_name	VARCHAR(45)	NN		Campo obrigatório para denominar a categoria

Fonte: Os autores

Quadro 8 – Dicionário de Dados: tb_product

tb_product				
Coluna	**Tipo de Dado**	**Restrições de Domínio**	**Indexação**	**Descrição**
id_product	INT	PK	Primária	Campo identificador da tabela produto
st_name	VARCHAR(45)	NN	Secundária	Campo obrigatório do nome do produto
nr_price	DECIMAL			Campo facultativo do preço do produto
id_user	INT		Secundária	Chave estrangeira facultativa que referencia o usuário que criou o produto
id_category	INT	NN	Secundária	Chave estrangeira obrigatória que referencia a categoria do produto
nr_measureValue	DECIMAL	NN		Valor quantitativo obrigatório da unidade de medida do produto
en_measureType	INT	NN		Enum obrigatório do tipo da unidade de medida do produto
st_barcode	VARCHAR(20)	NN	Secundária	Valor obrigatório do barcode do produto

Fonte: Os autores

Quadro 9 – Dicionário de Dados: tb_purchaseLocal

tb_purchaseLocal				
Coluna	**Tipo de Dado**	**Restrições de Domínio**	**Indexação**	**Descrição**
id_purchaseLocal	INT	PK	Primária	Campo identificador do local da compra
st_name	VARCHAR(45)	NN		Campo obrigatório para denominar o local da compra
point_location	POINT	NN		Campo obrigatório para guardar as informações relativo à geolocalização do local (latitude e longitude)

Fonte: Os autores

Quadro 10 – Dicionário de Dados: tb_purchase

tb_purchase				
Coluna	**Tipo de Dado**	**Restrições de Domínio**	**Indexação**	**Descrição**
id_purchase	INT	PK	Primária	Campo identificador Da compra
id_user	INT	NN	Secundária	Chave estrangeira obrigatória que referencia o usuário que realizou a compra
id_list	INT		Secundária	Chave estrangeira facultativa que referencia a lista na qual a compra foi baseada
id_purchaseLocal	INT		Secundária	Chave estrangeira facultativa que referencia o local em que a compra foi realizada
nr_purchasePrice	DECIMAL	NN		Campo obrigatório que registra o valor final da compra
dt_purchaseDate	DATE	NN		Campo obrigatório que registra a data da compra

Fonte: Os autores

Quadro 11 – Dicionário de Dados: tb_itemOfPurchase

tb_itemOfPurchase				
Coluna	**Tipo de Dado**	**Restrições de Domínio**	**Indexação**	**Descrição**
id_itemOf-Purchase	INT	PK	Primária	Campo identificador da tabela item da compra
id_purchase	INT	NN	Secundária	Chave estrangeira facultativa que referencia a compra no qual o item foi comprado
id_product	INT	NN	Secundária	Chave estrangeira facultativa que referencia os dados base do produto
st_name	VARCHAR(45)	NN	Secundária	Campo obrigatório do nome do item da compra
nr_price	DECIMAL	NN		Campo obrigatório do preço do item da compra
nr_amount	DECIMAL	NN		Campo obrigatório da quantidade do item comprado
nr_measureValue	DECIMAL	NN		Valor quantitativo obrigatório da unidade de medida do item da compra
en_measureType	INT	NN		Enum obrigatório do tipo da unidade de medida do item da compra

Fonte: Os autores

Quadro 12 – Dicionário de Dados: tb_productOfList

tb_productOfList				
Coluna	**Tipo de Dado**	**Restrições de Domínio**	**Indexação**	**Descrição**
id_productOfList	INT	PK	Primária	Campo identificador da tabela produto da lista
id_list	INT	NN	Secundária	Chave estrangeira obrigatória que referencia a lista no qual o produto está inserido
id_assignedUser	INT		Secundária	Chave estrangeira facultativa que referencia o usuário que comprará o produto
id_userWho-Marked	INT		Secundária	Chave estrangeira facultativa que referencia o usuário que comprou o produto
id_product	INT	NN	Secundária	Chave estrangeira facultativa que referencia os dados base do produto
st_name	VARCHAR(45)	NN	Secundária	Campo obrigatório do nome do item da compra
bl_isMarked	TINYINT	NN		Campo booleano a fim de verificar se o item foi marcado na lista de compras
nr_price	DECIMAL			Campo facultativo do preço do item da compra
nr_amount	DECIMAL			Campo facultativo da quantidade do item comprado
nr_measureValue	DECIMAL			Valor quantitativo facultativo da unidade de medida do item da compra
en_measureType	INT			Enum facultativo do tipo da unidade de medida do item da compra

Fonte: Os autores

Quadro 13 – Dicionário de Dados: tb_comment

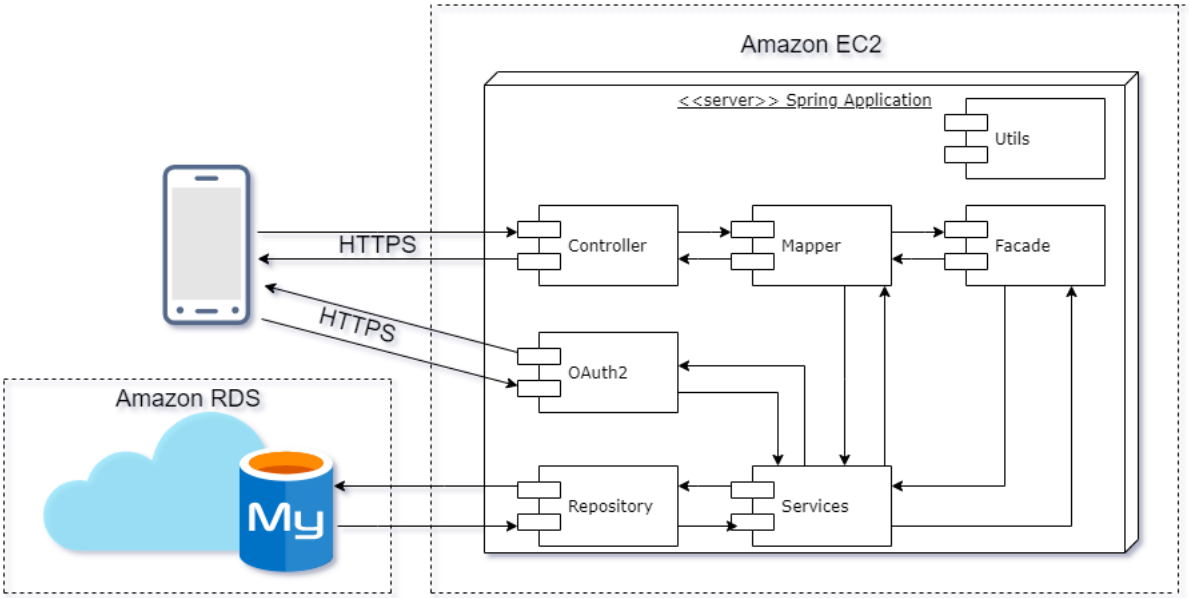
tb_comment				
Coluna	**Tipo de Dado**	**Restrições de Domínio**	**Indexação**	**Descrição**
id_comment	INT	PK	Primária	Campo identificador da tabela produto do comentário
id_productOfList	INT	NN		Chave estrangeira obrigatória que referencia o produto que recebeu o comentário
id_user	INT	NN	Secundária	Chave estrangeira obrigatória que referencia o usuário que realizou o comentário
st_content	VARCHAR(200)	NN		Campo obrigatório que armazena o comentário feito ao produto
dt_createdAt	INT	NN		Campo obrigatório que armazena a data no qual o comentário foi realizado

Fonte: Os autores

3.4.2 Padrão Arquitetural

A arquitetura escolhida para o desenvolvimento do Lixt foi cliente-servidor (com React-Native e Spring, respectivamente). Nesse tópico, iremos abordar detalhadamente o desenvolvimento de cada serviço.

Figura 3 – Detalhamento da Arquitetura



Fonte: Os autores

O cliente, que é uma aplicação mobile para Android, realizará requisições ao

backend e, assim, conseguir operar suas funcionalidades e exibir os dados no layout de modo coerente no aplicativo.

O servidor, por sua vez, será dividida em camadas lógicas. A camada que será exposta e acessível para o frontend são os endpoints dos controllers, recebendo e retornando DTOs (padrão Data Transfer Object) em formato JSON. Após acessar o endpoint, caso esteja devidamente autenticado no OAuth2 e com o token de autorização, será chamado o mapper, responsável por converter DTO em model.

Após essa conversão, dependendo da complexidade da regra de negócio, pode chamar o service - para regras de negócios simple - ou um facade - responsável por encapsular regras de negócios complexas, chamando os services a fim de evitar injeção de dependência cruzada entre os services.

Os services, por sua vez, encapsulam as operações de CRUD do hibernate que são definidas nas Repositories.

Após a realização da requisição, será enviada uma resposta para o frontend, onde o mapper, dessa vez, irá converter o model em DTO, de modo que consiga tratar o dado a partir de formatações, retirada de dados sensíveis e garantindo maior performance ao enviar apenas os dados que serão devidamente utilizados (e não dados em excesso), garantindo menor dados em tráfego na rede, menor consumo de internet e maior velocidade.

O OAuth2 é o serviço de autenticação e autorização adotado que irá se aproveitar dos services para realizar a verificação das credenciais fornecidas no momento do login.

O Utils é uma biblioteca global que pode ser reutilizado em todo o projeto como, por exemplo, validação de dados, formatação de dados, entre outros.

O backend será hospedado no Amazon EC2 e o banco de dados MySQL estará disponível no Amazon RDS.

3.4.3 Serviços de Apoio

No Lixt, haverá serviços de apoio que auxiliarão a cumprir as regras de negócios.

O primeiro serviço é o de autenticação e autorização, no qual será utilizado o OAuth2 para cumprir essa função. O OAuth2 garante a maior segurança, conseguindo permitir a utilização do sistema a partir do token que foi previamente registrado no sistema (impedindo a criação de token manualmente para acessar a plataforma). Para a autorização do usuário, será solicitado um bearer token. Além disso, garante a escalabilidade, visto que permite a autenticação de outras APIs a partir da geração o próprio token com as credenciais de clientID e secretID - gerando um basic token -, facilitando a integração entre APIs. Inclusive, para um uusário se autenticar com OAuth2, é necessário que o serviço que enviará a requisição possua seu próprio basic token, reforçando a segurança ao garantir

que o sistema não estará disponível para ser acessado facilmente.

O segundo serviço é o de email, que será implementado com javamail. Para esse serviço, será criada uma conta de gmail com as configurações habilitadas para enviar email de terceiros (ou seja, da nossa API).

O terceiro serviço é o serviço de swagger, que documenta os endpoints do API (método HTTP, DTO requerido, DTO retornado e headers).

3.5 Escalabilidade

A escalabilidade da aplicação é sua capacidade de se adequar a um amplo volume de requisições, mantendo a estabilidade do sistema e a velocidade de respostas. Um sistema escalável está apto para responder adequadamente nestes momentos, assim como liberar seus recursos em momentos com poucas requisições.

Por se tratar de algo relativo ao processamento de requisições, a escalabilidade diz respeito ao back-end, que centraliza os pedidos dos usúarios. A camada cliente por outro lado, por focar exclusivamente na lógica de visualização, e rodar em dispositivos mobile, não exige tanta capacidade de processamento, e a escalabilidade não é uma preocupação para esta camada.

Como foi mencionado, a aplicação back-end ficará armazenada na plataforma Amazon AWS. Existe um processo da ferramenta de integração contínua Jenkins para aumentar o número de instâncias ativas em produção, e este processo será acionado em momentos com intenso volume de requisições.

3.6 Manutenibilidade da aplicação

É fundamental para o desenvolvimento do projeto, tanto o previsto quanto em avanços posteriores, que a aplicação atinja um nível adequado de qualidade, e para tanto certos requisitos de manutenibilidade devem ser estabelecidos. Estes requisitos permitem a

3.6.1 *Logs*

Como forma de monitorar a aplicação em tempo de execução, especialmente na camada de servidor, *logs* serão usados para registrar o estado dos objetos. A ferramenta Log4j será usada, uma vez que os membros do time já tem mais familiaridade com ela. Esta ferramenta perimite o registro em diversos níveis, como

- *info*
- *debug*

- *warn*
- *error*

e pode ser configurada para que apenas os dois últimos sejam registrados no ambiente de produção. Desta forma a cada ponto de falha da aplicação um log de nível apropriado será colocado para que problemas sejam rapidamente identificados, analisados e resolvidos.

3.6.2 Integração Contínua

Visando manter o serviço sempre atualizado para o usuário, a ferramenta de integração contínua Jenkins foi selecionada para a implantação da aplicação back-end em produção. Ela permite, a partir do código no repositório git, a execução de testes, o build e o deploy para o ambiente de produção, automatizando tarefas complicadas de se executar em máquinas remotas.

Para o front-end, no entanto, não existem ferramentas de integração contínua, uma vez que as imagens do aplicativo mobile precisam ser aprovadas pelas lojas de aplicativo. Contudo, ainda é possível usar ferramentas de CI para a execução de testes e verificação da integridade do código a cada nova versão.

3.6.3 *Code Conventions*

As convenções de código são acordos internos ao time que visam estandartizar a forma como os diversos integrantes do time produzem seus códigos. Elas visam facilitar o entendimento mútuo entre os integrantes do time, de modo o estilo de progamação seja indistiguível e independente de seus autores. Geralmente, as convenções de código estabelecem estilos para se organizar o código textualmente, isto é, dizem respeito a forma como nomes de variáveis são escolhidas e comentários são posicionados, por exemplo.

As convenções adotadas são baseadas na especificação da SUN MICROSYSTEMS, de 1997. Esta é comumente usadas para o desenvolvimento na linguagem Java, e muito próxima do padão adotado em JavaScript, e vale destacar o seguintes pontos:

- Minimizar o uso de variáveis, funções e objetos globais.
- As declarações globais estarão preferencialmente no início do arquivo.
- Declarar as variáveis próximo do ponto onde elas serão inicializadas.
- A indentação é de 4 espaços.
- Linhas mais longas que 80 caracteres serão quebradas e indentadas a 8 espaços.
- Pacotes e variáveis com nomes curtos, em `camelCase` e substantivos.

- Classes e interfaces em `CamelCase` e substantivos.
- Métodos em `camelCase` e verbos.
- Constantes em `UPPER_CASE`.

No entanto, especificamente para a linguagem Java,

- Classes e métodos devem ser documentados com um comentário na seguinte forma, uma vez que as IDEs reconhecem este formato e formatam o text na forma de *pop-ups* quando o cursor está sobre uma referência a esta classe.

```
/**
 * Class ListService
 *
 * Implementar endpoints para as funcionalidades de lista.
 */
```

No backend, a estrutura de pacotes vai ser bem dividida, tendo o pacote controller para os controllers e endpoints, mapper para os mappers. O model, service e repository de uma entidade ficarão no mesmo pacote cujo nome é o mesmo nome da entidade.

3.6.4 *Designs Patterns*

Como padrões de projetos, serão usados 3 padrões amplamente utilizados pela comunidade de desenvolvimento: (I) Clean Code, (II) SOLID e (III) Gang Of Four. Esses 3 padrões podem ser utilizados tanto no backend quanto no frontend.

3.6.4.1 *Clean Code*

O Clean Code é um conjunto de boas práticas para melhorar o entendimento do código, de modo que seja mais fácil sua leitura. Segue abaixo uma lista das principais boas práticas:

- Nome Significativos para variáveis, classes, métodos, atributos e objetos, evitando abreviaturas desnecessárias e nomes não recorrentemente utilizados, sendo passíveis de busca.
- Evitar números mágicos, recomendando-se utilizar enums e constantes para que seja mais compreensível ao analisar o código.
- Evitar booleanos de forma implícita.

- Evitar condicionais negativos (por exemplo, uma função com nome "naoDevoExecutar()"), visto que dificulta a compreensão de código.
- Encapsular condicionais.
- Nunca comentar o óbio no código.
- Utilizar funções pequenas, tanto para melhorar a leitura quanto para respeitar os padrões do SOLID.

3.6.4.2 *SOLID*

O SOLID é um acrônimo de 5 príncipios da Programação Orientada a Objetos, sendo fundamental para o desenvolvimento e manutenção de software.

- **S**ingle Responsiblity Principle: Uma classe deve ter apenas um motivo para mudar.
- **O**pen-Closed Principle: Uma classe deve estar aberta para extensão, e fechada para modificação, recomendando sempre utilizar a herança e não modificar o código fonte original.
- **L**iskov Substitution Principle: Uma classe derivada deve ser substituível por sua classe base.
- **I**nterface Segregation Principle: Utilizar muitas interfaces específicas é melhor do que uma interface genérica.
- **D**ependency Inversion Principle: Dependa de abstrações e não de implementações.

3.6.4.3 *Gang Of Four*

O GoF (ou Gang of Four) se refere aos profissionais que criaram os 23 padrões de projetos, nos quais seus conceitos serão utilizados no desenvolvimento do Lixt.

Segue abaixo uma breve apresentação dos principais padrões utilizados:

- Builder: Será utilizado para criar instâncias legíveis de objetos complexos.
- Facade: Encapsula regras de negócios complexas.
- Observer: Observa e é capaz de reagir às mudanças de estado de um objeto.

3.6.5 Testes

Testes são uma ferramenta fundamental para o desenvolvimento da aplicação, uma vez que garenatem, em tempo de compilação, o comportamento correto do aplicativo. Para além disso, testes tem um papel de documentação, uma vez que encapsulam de forma breve o comportamento esperado das classes e métodos produzidos, e podem ser consultados em caso de dúvida quando ao uso destes. Este tipo de teste é chamado de teste unitário, em oposição aos testes de integração, que verificam o funcionamento da aplicação de ponta-a-ponta, isto é, a partir de uma chamada a um endpoint, apenas os serviços externos são mimetizados, garantindo o funcionamento correto de toda a aplicação.

Desta forma, a construção de testes, de ambos os tipos é de extrema importância para a elaboração do projeto visando a sua manutenibilidade, e será o primeiro passo de uma sprint, após o planejamento, a construção de testes relevantes para a tarefa, seguindo os princípios do TDD(ANICHE, 2014). Como as ferramentas de teste são específicas de cada linguagem, cada camada da aplicação fará uso de frameworks distintos.

O back-end será testado com o framework JUnit, fazendo uso da biblioteca Mockito quando necessário simular comportamentos de objetos que não são o alvo da suite de teste. Este framework ainda oferece ferramentas para testar o banco de dados, ou melhor, testar a conexão com o banco e verificar o comportamento das classes de acesso a ele. Já os teste de front-end serão feitos com a biblioteca Jest, que auxilia a construção de testes unitários, e por se tratar de uma GUI, as interções de usuário devem ser simuladas também. Para tanto, a blioteca React Testing Library será usada.

3.7 Segurança, Privacidade e Legislação

A principal lei brasileira que trata de tratamento de dados nos meios digitais é a lei N° 13.709, sancionada em 14 de Agosto de 2018 (BRASIL, 2018) e que entrou em vigor no ano de 2020, conhecida como Lei Geral de Proteção de Dados (LGPD).

Seguindo o disposto no Artigo 6°, inciso terceiro da LGPD, a aplicação vai coletar o mínimo de dados do usuário necessários para uso da aplicação, os dados serão: localização, endereço de email e nome do usuário. Sendo facultativo ao usuário ativar ou não a sua localização.

O design da aplicação vai seguir o princípio da transparência. O usuário do aplicativo será informado de quais as informações que serão coletadas. O próprio sistema Android, por padrão, solicita ao usuário permissão para uso da localização, um dos dados que será necessário coletar, caso o usuário opte pela detecção do local da compra automaticamente.

Como a maioria dos sistemas atuais, utilizaremos uma API (Application Program Interface) para realizar a comunicação e transferência de dados entre a interface de usuário

e o servidor. Essa arquitetura possui intrinsecamente vulnerabilidades conhecidas e quando não são bem projetadas as API's podem ser um dos pontos fracos do sistema quando se trata de segurança. A empresa de consultoria Gartner (LAMBA, 2019) prevê que a tendência é que em 2022 as API's se tornem o principal foco de ataques cibernéticos.

Cientes desse cenário, foi definido que o sistema deverá seguir algumas boas práticas de desenvolvimento, citadas brevemente a seguir:

- Autenticação: As requisições apenas serão aceitas se o usuário estiver logado no sistema;
- Criptografia: Para evitar ataques do tipo man-in-the-middle as mensagens entre cliente e servidor serão criptografadas, seguindo o protocolo HTTPS;
- Documentação dos *endpoints* sensíveis: Será feito um levantamento de todos os *endpoints* que acessam informações sensíveis, para garantir que apenas usuários autenticados tenham acesso;

Como medida de segurança, o banco de dados não irá armazenar as senhas das contas dos usuários e sim um *hash* da senha, e este será comparado com o *hash* da senha informado no momento do login. Ainda como uma forma de segurança, ao cadastrar a senha ou mudar a senha atual o usuário não poderá cadastrar senha que não possua menos de seis dígitos, e que não tenham letras maiúsculas e minúsculas e números, e deverá ter no mínimo um caractere especial.

3.8 Viabilidade Financeira

O projeto de análise de viabilidade financeira consiste em averiguar a garantia de lucro sobre as despesas do projeto. Portanto, nesse projeto será descrito cada processo a fim de fazer essa verificação.

3.8.1 Gerenciamento de Custos

Nesse tópico, serão abordados temas de investimento inicial e de desenvolvimento do projeto, incluindo tópicos de análise de requisitos, desenvolvimento, manutenções e imprevistos.

3.8.1.1 Análise de Requisitos e Desenvolvimento

Para iniciar o projeto, é necessário fazer os primeiros planejamentos, elicitação de requisitos, abstrair e concretizar as primeiras ideias e fazer os primeiros planejamen-

tos (diagramas, cronogramas e documentação). Logo após, o projeto chega na fase de desenvolvimento, onde é começado a se tornar real.

Contudo, o projeto não vai possuir nenhum custo de análise e implementação do sistema, devido ao fato de ser um projeto educacional.

3.8.1.2 Manutenções

Inevitavelmente, manutenções do sistema ocorrerão pós finalização do projeto e estar devidamente funcional em produção. Contudo, os custos de manutenções também não serão cobrados, devido a ser um projeto educacional.

3.8.2 Custos de deploy e de Ambiente de Produção

Nesse tópico, são apresentados os custos de manter o sistema funcional e disponível para os usuários. Desse modo, será feito uma previsão anual de cada plataforma utilizada:

3.8.2.1 Frontend

Tendo em vista que o projeto é *mobile* voltado para dispositivos Android, será publicado na PlayStore, estimando um valor de 25.00 USD anual.

3.8.2.2 Backend

Inicialmente gratuíto no Amazon EC2, sendo permitido 750h de instâncias por mês durante o período de 12 meses.

A partir do momento que for necessário grande porte, será indicado o plano Sob Demanda do Amazon EC2, que garante viabilidade econômica e estratégica (visto que o preço é calculado a partir do uso).

Utilizando a calculadora da AWS e optando por um servidor Linux da instância t4g.micro com 1 vCPU e 1GiB, com armazenamento SSD de uso geral de 10GB, será custeado o valor de 7,52 USD mensalmente para operar o mês inteiro.

3.8.2.3 Banco de Dados

Inicialmente gratuíto no Amazon RDS, sendo permitido 750h de instâncias durante o período de 12 meses. O Amazon RDS possui suporte a vários Sistemas Gerenciadores de Banco de Dados (SGBD), incluindo o MySQL, que foi o SBGD optado para desenvolver a aplicação Lixt.

A partir do momento que for necessário grande porte, será indicado o plano Sob Demanda do Amazon RDS, que garante viabilidade econômica e estratégica (visto que o preço é calculado a partir do uso).

Utilizando a calculadora da AWS e optando por um servidor da instância t2.micro de modelo Single-AZ OnDemand, com armazenamento SSD para cada instância de 10GB, será custeado o valor de 27,74 USD mensalmente para operar o mês inteiro.

3.8.3 Medidas de Obtenção de Retorno Financeiro

Para gerar uma receita positiva a fim de obter lucro, haverá duas formas principais de retorno financeiro:

- Cobrança do aplicativo: O aplicativo estará disponível gratuitamente na PlayStore, não gerando, portanto, retorno financeiro.
- Propaganda/Recomendação: Será utilizado mediador de anuncio AdMob (responsável por conectar aplicações e anunciantes), onde o valor varia por visualizações de anúncios e cliques neles. Contudo, no próprio site do admob, é citado um caso no qual houve 300.000 downloads e arrecadava, através do AdMob, 100 USD por dia. (**??**)

A partir da calculadora do AdMob, estimando-se uma quantidade de 250 visitantes por dia, onde há 2 paginas visualizadas por visitante, sendo que a taxa de cliques em anúncios é 1% e o custo do clique é 0.35 USD, o valor mensal será de 52.5 USD.

3.8.4 Conclusão

Concluindo que, ao utilizar os servidores de baixo porte detalhados acima, será desembolsado cerca de 35.00 USD por mês. Contudo, o valor calculador para 250 visitantes (estimando o valor com baixo engajamento) com os parâmetros detalhados arrecadará 52.5 USD, conseguindo custear os servidores e ainda garantindo margens de lucro.

Conforme o engajamento na aplicação for aumentando, será revisto os planos dos servidores para atender maiores níveis de requisições.

4 Planejamento e Gerenciamento do Projeto

As próximas seções possuem o objetivo de descrever mais sobre como a equipe e o projeto estão organizados quanto a metodologia, gestão de tempo e papéis de atuação.

4.1 Metodologia de Gestão e Desenvolvimento de Projeto

O grupo optou pelo uso de uma metodologia ágil para o projeto e, após algumas reuniões, foi decidido que o que guiaria o processo de desenvolvimento seria uma junção de frameworks extremamente úteis para a melhor performance do time, o scrum e o kanban. O scrum se baseia na divisão do projeto em vários ciclos de atividades — conhecidos como sprints - com diversas cerimônias e interações frequentes da equipe a fim de alinhar o que tem sido feito, tratar impedimentos e pensar em maneiras de otimizar o processo de trabalho, aumentando a agilidade no desenvolvimento. Já o kanban é um método de controle e gestão de forma visual, geralmente feito com o uso de post-its coloridos que reforçam a simbologia das tarefas e ações que precisam ser feitas, estão sendo feitas, ou foram concluídas.

Dentro das metodologias ágeis é muito comum - e essencial - a definição do product backlog contendo todas as funcionalidades desejadas na aplicação pelo cliente e elencadas por prioridade. Esse backlog é, posteriormente, dividido em tarefas que serão distribuídas em cada Sprint, gerando o sprint backlog. É nesse momento em que, no projeto, o Kanban será aplicado. Dessa forma, o scrum será utilizado para a determinação do método de desenvolvimento iterativo e incremental e o kanban será utilizado para métricas e fluxos de produção da equipe.

4.2 Organização da Equipe

As primeiras reuniões de alinhamento entre os membros da equipe foram essenciais para que os integrantes pudessem interagir e compartilhar mais sobre quais conhecimentos prévios possuem, quais são suas áreas de maior facilidade e interesse e dessa forma planejar uma estratégia de desenvolvimento que, apesar de desafiadora, também seja inclusiva a todos do grupo.

No Scrum, metodologia escolhida para o projeto, a definição de alguns papéis é necessária. São eles:

- **Product Owner**: É o responsável pelos interesses do cliente no projeto, captando,

Atividade	Alkindar	Carolina	Fabio	Gabriely	Leonardo	Mariana
Back-End	x		x		x	
Banco de dados	x		x		x	
Blog	x	x	x	x	x	x
Documentação	x	x	x	x	x	x
Front-End		x		x		x
Vídeos		x		x		x

Tabela 1 – Tabela de responsabilidades

interpretando e repassando ao time de desenvolvimento suas necessidades. Quem assumiu este papel foi o Fabio Mendes;

- **Scrum Master**: É o facilitador no desenvolvimento do projeto, cuidando para que as cerimônias do Scrum sejam devidamente cumpridas e ajudando a resolver os impedimentos que possam atrapalhar a equipe de desenvolvimento. Quem assumiu este papel foi a Carolina de Moraes;
- **Development Team**: Time que cuida do desenvolvimento técnico do projeto. Quem assumiu este papel foram Alkindar Rodrigues, Gabriely Bicigo, Leonardo Naoki e Mariana Zangrossi.

É importante salientar que, apesar da divisão de papéis, todos os participantes estão contribuindo no desenvolvimento com o objetivo de entregar o projeto na data estimada.

Para uma divisão mais organizada e formal de tarefas, foi montada uma tabela de responsabilidades dos integrantes referentes aos tópicos de desenvolvimento do projeto.

4.3 Gestão de Tempo

Cada sprint do projeto terá a duração de 2 semanas e respeitará todas as cerimônias do scrum. Antes de dar início a uma nova sprint ocorrerá a realização da sprint planning, reunião de planejamento onde serão definidos os entregáveis da Sprint a ser iniciada. Após o início da iteração, as dailies serão feitas através do grupo de mensagens instantâneas do time. Ao fim do período de 2 semanas, ocorrerá a sprint review, reunião com o objetivo de avaliar o que foi entregue com sucesso naquela Sprint, e também a sprint retrospective, reunião onde a equipe conversa sobre os pontos positivos e negativos da última iteração e o que pode ser melhorado para a próxima Sprint. Todas essas cerimônias foram incluídas no planejamento de gestão de tempo para o cálculo da quantidade de Sprints necessárias até a entrega final do projeto.

Referências

ADJUTO, G. Celular é o principal meio de acesso à internet no país. *Agência Brasil*, 2018. Citado na página 14.

ANGELO, C. As compras não planejadas em supermercados: a importância do tempo e da organização da loja na determinação dos gastos. *Revista de Administração Contemporânea*, 2003. Citado na página 14.

ANICHE, M. Test driven development. 2014. Citado na página 37.

AU, G. M. N. The cost-effectiveness of shopping to a predetermined grocery list to reduce overweight and obesity. *Macmillan Publishers Limited*, 2013. Citado na página 13.

BRASIL. *Lei Geral de Proteção de Dados*. 2018. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm>. Acesso em: 7 jun. 2021. Citado na página 37.

COLELLA, M. T. Planejamento financeiro familiar: A importância da organização e controle no orçamento familiar. *FAIT*, 2015. Citado na página 13.

CONLEY, K. Towel: Towards an intelligent to-do list. *AAAI ORGANIZATION*, 2007. Citado na página 13.

LAMBA, A. Api design principles and security best practices – accelerate your business without compromising security. *SSRN Electronic Journal*, 01 2019. Citado na página 38.

MELHORAMENTOS, E. Organização. *UOL*, 2021. Citado na página 13.

PADMANABHAN, A. Convention over configuration. *Devopedia*, 2020. Citado na página 23.

SEUNG-HO, L. Experimental comparison of hybrid and native applications for mobile systems. *International Journal of Multimedia and Ubiquitous Engineering*, 2015. Citado 2 vezes nas páginas 14 e 15.

SUN MICROSYSTEMS, I. Java coding conventions. 1997. Citado na página 34.

Glossário

API	Uma Interface de Progamação de Aplicação (*Application Programming Interface*), são pontos que a aplicação expoõe para permitir que usuários ou serviços externos executem tarefas dentro da aplicação. 17
back-end	Um sistema *back-end* é aquele que encontra na camada de servidor, em uma aplicação de duas camadas. Sua principal função é fornecer informações e capacidade de processamento a aplicação cliente. 16, 17, 22, 23, 33, 34, 37
deploy	Processo pelo qual a aplicação é implantada em ambiente de prpdução, e está disponível para os usuários finais. 34
framework	Conjunto de ações e estratégias que possuem um ojeto e objetivo em específico; Na programação, é um conjunto de pacotes e bibliotecas que abstarai alguma função complexa, geralmente de nível mais baixo, e sobre o qual uma aplicação pdoe ser contruída. Ex: o framework web Spring asbtrai a lógica de implementação de um servidor, auxiliando a criação de endpoints, rotas e processamento de HTTP. 16, 17, 23
front-end	Um sistema *front-end* é aquele que encontra na camada cliente, em uma aplicação de duas camadas. Sua principal função, no escopo deste projeto, é atuar como interface gráfica para o usário, coletar dados e enviá-los para o *back-end*. 16, 17, 34, 37
GUI	Interface gráfica de usuário é uma forma visual de se apresentar dados e coletar interações com o usuário, em oposição a linha de comando, que funciona por texto apenas. 37
kanban	Framework de gestão visual de fluxo de produção, que consiste no uso de cartões coloridos que separam as diferentes fases do desenvolvimento das tarefas (to do, doing, done). 41
product backlog	Lista de todos os requisitos e funcionalidades desejadas para um produto. 41
REST	A Tranfêrencia por Representação de Estado é uma forma de se transferir dados na qual os atirbutos de um objeto (seu estado) são serializados em um arquivo de formato específico, e o objeto pode ser reconstituído na aplicação que recebe o arquivo. 17

scrum	Framework ágil utilizado em desenvolvimentos iterativos e incrementais. 41, 42
sprint	Período de tempo limitado dentro do Scrum onde uma quantidade de histórias de usuário incrementáveis são desenvolvidas. 42
sprint retrospective	Cerimônia do Scrum que ocorre no fim da Sprint com o objetivo de avaliar os pontos positivos e negativos da Sprint, agrupando possibilidades de melhorias para a próxima Sprint. 42
sprint review	Cerimônia do Scrum que ocorre no fim da Sprint com o objetivo de validar as entregas daquele período. 42
sprint planning	Cerimônia do Scrum que ocorre antes do início de uma nova Sprint com o objetivo de planejar quais histórias de usuário serão desenvolvidas naquela iteração. 42
sprint backlog	Lista das histórias de usuário que serão desenvolvidas durante a Sprint a ser iniciada. 41