Armstrong Lohãns de Melo Gomes Quintino

ENZITECH: SISTEMA DE EXPERIMENTAÇÃO PARA ANÁLISE E CÁLCULO DE ATIVIDADES ENZIMÁTICAS DO SOLO

Trabalho de Graduação

Universidade Federal do Agreste de Pernambuco
coordenacao.bcc@ufape.edu.br
bcc.ufape.edu.br/curso

Garanhuns-PE
2022

Universidade Federal do Agreste de Pernambuco
Graduação em Ciência da Computação

Armstrong Lohãns de Melo Gomes Quintino

ENZITECH: SISTEMA DE EXPERIMENTAÇÃO PARA ANÁLISE E CÁLCULO DE ATIVIDADES ENZIMÁTICAS DO SOLO

Trabalho apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Agreste de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Rodrigo Gusmão de Carvalho Rocha*

Garanhuns-PE
2022

Armstrong Lohãns de Melo Gomes Quintino
Enzitech: Sistema de experimentação para análise e cálculo de atividades enzimáticas do solo/ Armstrong Lohãns de Melo Gomes Quintino. – Garanhuns-PE, 2022-
62 p. : il. (algumas color.) ; 30 cm.

Orientador Rodrigo Gusmão de Carvalho Rocha

Trabalho de Graduação – Universidade Federal do Agreste de Pernambuco, 2022.

1. Palavra-chave1. 2. Palavra-chave2. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

CDU 02:141:005.7

Trabalho de conclusão de curso apresentado por **Armstrong Lohãns de Melo Gomes Quintino** ao programa de Graduação em Ciência da Computação da Universidade Federal do Agreste de Pernambuco, sob o título **Enzitech: Sistema de experimentação para análise e cálculo de atividades enzimáticas do solo**, orientada pelo **Prof. Rodrigo Gusmão de Carvalho Rocha** e aprovada pela banca examinadora formada pelos professores:

Profª.
Centro de Informática/UFAPE

Prof.
Centro de Informática/UFAPE

Prof. Rodrigo Gusmão de Carvalho Rocha
Centro de Informática/UFAPE

Garanhuns-PE
2022

Lorem Ipsum

Agradecimentos

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

“Lorem Ipsum“

— JOHN

Resumo

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Palavras-chave: Cálculo de atividades enzimáticas do solo, Engenharia de Software, Desenvolvimento Mobile, Arquitetura Limpa, Flutter

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Keywords: Calculation of soil enzymatic activities, Software Engineering, Mobile Development, Clean Architecture, Flutter

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

Sumário

1

Introdução

A eficiente e incessante evolução da tecnologia traz consigo a grande oportunidade — como também a necessidade — de tornar as tarefas cotidianas dos seres humanos cada vez mais triviais e informatizadas, ou seja, realizar uma tarefa de forma que exija o menor esforço possível, dessa forma, algo que requisitaria tempo e atenção de alguém vem a se tornar uma tarefa simples para um computador.

As aplicações móveis, habitualmente conhecidas por Aplicativos (APPs), são exemplos claros e amplamente difundidos atualmente, visto que cada vez mais tarefas, desde o envio de mensagens até o pagamento de contas, podem ser realizadas nas palmas das nossas mãos, basicamente utilizando um *smartphone* com acesso à Internet. O que há pouco mais de uma década era visualizado como uma atividade exclusiva tecnologicamente de um Computador Pessoal (PC), fosse através de uma aplicação *web* ou *desktop*, hoje é facilmente feito por soluções adaptadas ou similares para as telas dos celulares, tornando-se uma atividade corriqueira na vida da maioria das pessoas.

[EM DESENVOLVIMENTO até 31/03/23]

1.1 Motivação

— Apresentar e discorrer sobre a motivação do projeto em si. — [EM DESENVOLVIMENTO até 31/03/23]

1.2 Objetivo

Este trabalho tem como objetivo detalhar a solução e o processo de desenvolvimento de uma aplicação para dispositivos móveis que informatize o método de fazer experimentos e análises do solo, aplicativo este desenvolvido usando padrões arquiteturais limpos, escaláveis, manuteníveis e testáveis. [EM DESENVOLVIMENTO até 31/03/23]

1.3 Visão geral da proposta

— Apresentar e discorrer sobre o objetivo do projeto em si. — [EM DESENVOLVIMENTO até 31/03/23]

1.4 Organização do texto

— Apresentar e discorrer sobre a organização do texto do projeto. — [EM DESENVOLVIMENTO até 31/03/23]

2

Fundamentação Teórica

Neste capítulo serão levantados dados sobre a necessidade de meios para combater a escassez global da produção de alimentos, sendo abordados detalhes técnicos da área de atuação deste *software*, que é voltado à pesquisas e experimentos na área da Agronomia, sobre a qualidade do solo e suas atividades enzimáticas para um melhor aproveitamento do solo na produção de alimentos. Além disso, serão discutidos pontos referentes ao crescimento da mobilidade no Brasil e no mundo, trazendo o contexto da solução apresentada por este projeto, bem como conceitos técnicos relacionados aos dispositivos móveis, desenvolvimento de *software*, metodologias ágeis e arquitetura do sistema.

2.1 Qualidade do solo

A Qualidade do Solo (QS) é um dos fatores fundamentais para o aumento da produtividade agrícola e, consequentemente, para a redução da escassez global de alimentos. Segundo a Organização das Nações Unidas para Agricultura e Alimentação (FAO), cerca de um terço dos solos do mundo estão degradados devido a práticas agrícolas insustentáveis e outros fatores como poluição, erosão, compactação e perda de matéria orgânica. Além disso, a Organização das Nações Unidada (ONU)[1] estima que a população mundial chegará a 9,7 bilhões em 2050, o que exigirá um aumento de 70% na produção de alimentos para suprir a demanda crescente FAO (2018).

Nesse contexto, é fundamental entender a QS, suas características físicas, químicas e biológicas, bem como as interações entre esses fatores. Isso permitirá o desenvolvimento de técnicas mais eficientes e sustentáveis de manejo do solo, contribuindo para a melhoria da produtividade agrícola e, consequentemente, para a redução da escassez global de alimentos.

Desta forma, trazendo o problema para o escopo do Brasil, surge a necessidade de conciliar nossa produção agrícola com a preservação ambiental, e isso têm despertado a atenção da comunidade científica e os produtores para a necessidade de associar produtividade e formas mais sustentáveis de produção LÓPEZ-SANTIAGO et al. (2019). Com este objetivo, as medidas

[1] ONU - População: `https://www.un.org/en/global-issues/population`

da Contribuição Nacionalmente Determinada (NDC) acordada pelo Brasil com a Convenção Quadro das Nações Unidas sobre mudanças climáticas, incluem como estratégia o fortalecimento do Plano de Agricultura de Baixa Emissão de Carbono (Plano ABC) através do desenvolvimento sustentável na agricultura até 2030 EMBRAPA (2018).

2.2 Cálculo de atividades enzimáticas do solo

Diversos métodos são utilizados para avaliar a qualidade do solo, como a análise físico-química, a análise microbiológica e a análise das Atividades Enzimáticas (AEs) do solo. Esta última pode ser utilizada como uma medida da atividade biológica do solo, pois as enzimas são produzidas pelos micro-organismos presentes no solo e são essenciais para a decomposição da matéria orgânica, a liberação de nutrientes e a formação de compostos benéficos para as plantas.

O cálculo de AEs do solo é uma importante ferramenta para avaliar a qualidade e saúde do solo, uma vez que a atividade enzimática está diretamente relacionada com a quantidade e diversidade de microrganismos presentes no solo. Alguns exemplos de enzimas comumente utilizadas para esse cálculo são a fosfatase ácida, a β-glucosidase, a urease e a protease. A atividade dessas enzimas pode ser afetada por vários fatores, como o pH do solo, a umidade, a temperatura e a presença de metais pesados. Portanto, a análise das AEs do solo é importante para identificar possíveis impactos ambientais e planejar práticas de manejo sustentável.

Entre as referências bibliográficas relevantes para o estudo de AEs em solo, podemos citar o trabalho de TABATABAI; BREMNER (1969), que desenvolveram uma metodologia para a determinação de fosfatase ácida em solo; o estudo de ALLISON; JASTROW; IVANS (2010), que comparou diferentes metodologias para a medição da AE em solos de diferentes ecossistemas; e o trabalho de SINSABAUGH et al. (2008), que avaliou a influência da qualidade do substrato e da umidade do solo na atividade enzimática.

2.3 Mobilidade

Segundo B'FAR (2004), um sistema de computação móvel é um sistema que pode ser facilmente movido fisicamente ou cuja funcionalidade pode ser usada durante o movimento. Como esses sistemas fornecem essa mobilidade, essas funcionalidades adicionadas são a razão para caracterizar separadamente os sistemas de computação móvel, eles geralmente oferecem capacidades e recursos não encontrados em sistemas normais, como:

- Armazenamento de dados local e/ou remoto via conexões com ou sem fio;
- Segurança para persistência de dados em caso de queda de energia ou pane;
- Sincronização de dados com outros sistemas;
- Etc.

Atualmente, pensamos em um sistema móvel como um sistema projetado para rodar em um computador de mão, seja ele celular, tablet ou qualquer outro dispositivo com tais características. Pela definição acima, os notebooks também são considerados plataformas para sistemas móveis, mas não são utilizados exatamente da mesma forma que os dispositivos citados acima, pois é necessário parar em algum lugar, abrir o notebook, esperar carregar, etc...

2.4 Dispositivos móveis

Para adentrar no universo da solução apresentada, os Dispositivos Móveis (DMs), é valioso abordar como estes *gadgets* lidam com a capacidade de armazenamento e processamento de dados, que, por possuírem sistemas móveis espera-se que os mesmos possuam menor eficiência para desenvolver atividades se comparados a um computador estacionário (PC), por exemplo, que possui muito mais disponibilidade energética para realizar suas tarefas, porém, nos dias de hoje esse *gap* para atividades cotidianas está mais estreito, devido aos grandes avanços da tecnologia voltados a este segmento que vem em forte alta — o qual, segundo DATA.AI (2021) cresceu 20% em 2020 comparado ao ano anterior, gerando um consumo no valor de 143 bilhões de dólares no mundo todo, somente com APPs para DMs, mesmo com as dificuldades enfrentadas pela pandemia da COVID-19, vale ressaltar que há uma expectativa que o mercado de DMs cresça ainda mais nos próximos cinco anos, com uma Taxa de Crescimento Anual Composta (CAGR) de 4% no período de previsão de 2023 a 2028 — de acordo com a projeção de INTELLIGENCE (2023) (veja a Figura 2.1 abaixo).

Já para processamentos mais robustos estamos caminhando em largos passos, atualmente é possível se deparar com sistemas móveis utilizando o processamento em nuvem (que são outra forma de representação de sistema estacionários, um exemplo disso são os servidores computacionais) para tais atividades, ao mesmo tempo em que outros DMs já contam com *chipsets* dedicados para isso — como dispositivos com processadores com núcleos desenvolvidos exclusivamente ao processamento de Aprendizado de Máquina (AM) e Inteligência Artificial (IA), por exemplo.

Para situar onde os DMs chegaram, é preciso olhar um pouco o passado para lembrar como os computadores eram: máquinas que ocupavam salas gigantes, os quais eram manuseados somente por setores importantes da sociedade, antes mesmo de ser um dispositivo doméstico como é hoje, limitando-se a órgãos do governo, instituições de ensino e poucas empresas, por exemplo ALECRIM (2013).

Com o desenvolvimento da tecnologia, os computadores tornaram-se cada vez mais compactos, eficientes, práticos e fáceis de usar, podendo ser levados para qualquer lugar, por qualquer pessoa. As tecnologias que fornecem essa maior flexibilidade são conhecidas como DMs.

Sintetizando, um DM é um tipo de dispositivo computacional que tem como principais características a portabilidade, a compactabilidade e fácil manuseio LEE; SCHNEIDER;

Figura 2.1: Mercado de Smartphones - crescimento, tendências, impacto da COVID-19 e previsões (2023 - 2028)
Fonte: INTELLIGENCE (2023)

SCHELL (2005), além de todos os aspectos citados na seção 2.3.

De acordo com LARICCHIA (2022), em 2021, o número de DMs operando em todo o mundo ficou em quase 15 bilhões, contra pouco mais de 14 bilhões no ano anterior. Espera-se que o número de DMs atinja 18,22 bilhões até 2025, um aumento de 4,2 bilhões de dispositivos (aproximadamente 30%) em comparação com os níveis de 2020. A Figura 2.2 mostra o gráfico desta previsão.

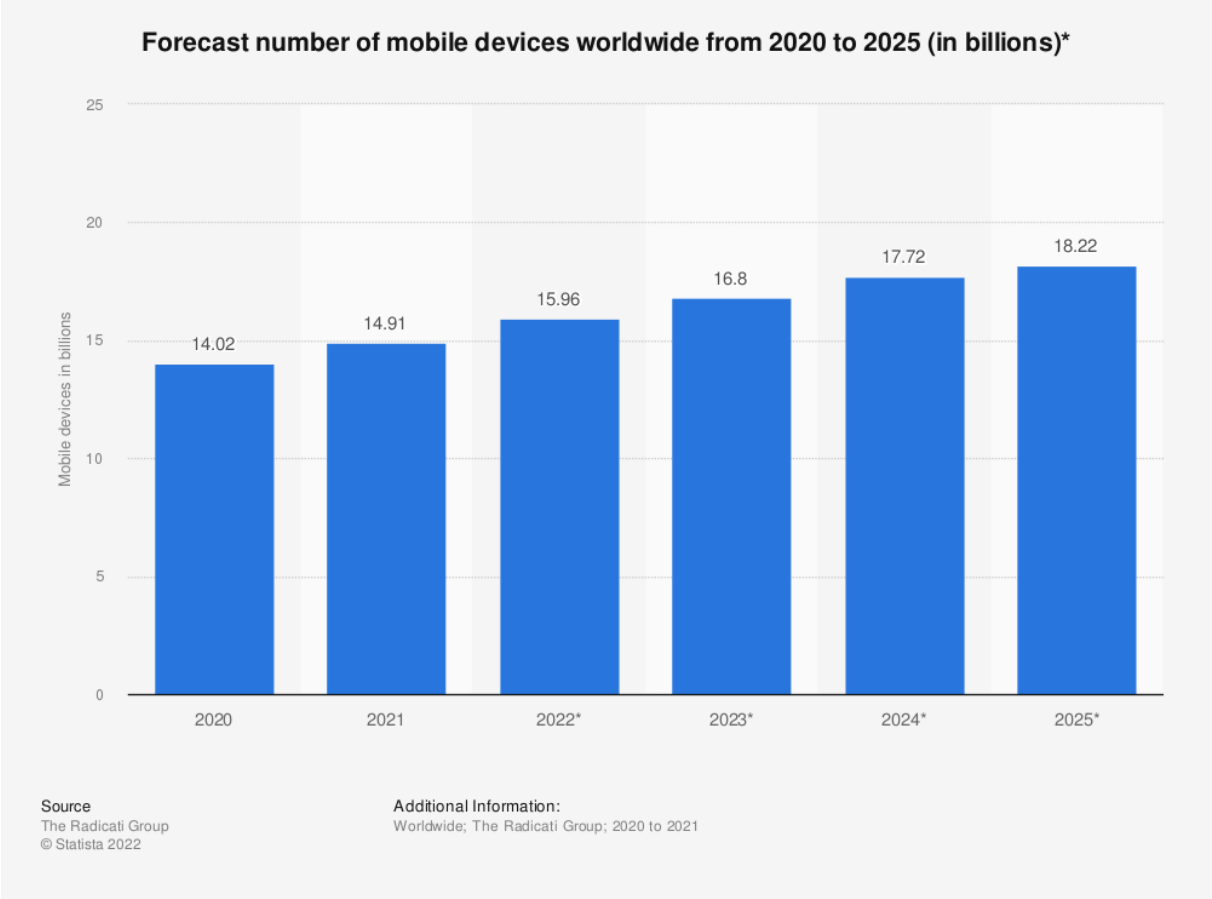

Figura 2.2: Previsão do número de dispositivos móveis em todo o mundo de 2020 a 2025 (em bilhões)

Fonte: LARICCHIA (2022)

Visto esse crescimento no uso de DMs, é perceptível que a demanda por **desenvolvimento de *software*** para esta tecnologia também aumentou. Essas soluções são chamadas de aplicativos móveis, tema que será abordado com mais detalhes na seção 2.5 e posteriormente detalhado sobre o mercado de desenvolvimento de APPs na subseção 2.5.1.

Falar sobre DMs é falar também de sobre seus Sistemas Operacionais (SOs), pois estes são responsáveis por gerenciar e controlar o *hardware* e *software* dos mesmos, atualmente, os principais são o iOS da Apple e o Android da Google. Ambos SOs tem características específicas e oferecem diferentes níveis de personalização e integração com outros dispositivos e serviços.

Desta forma, a existência destes dois grandes SOs de DMs é devido à competição entre as empresas de tecnologia para oferecer a melhor experiência do usuário e atrair o maior número de usuários para seus dispositivos. Além disso, os SOs também são importantes para garantir a compatibilidade com APPs e jogos, bem como para garantir a segurança e privacidade dos dados do usuário. Eles também possibilitam acesso a recursos como câmera, GPS, conectividade com a internet e outros dispositivos, além de gerenciar a bateria e os recursos de armazenamento; as peculiaridades e mercado de ambos SOs serão abordados brevemente a seguir, em suas respectivas subseções.

2.4.1 iOS

O iOS é o SO móvel desenvolvido pela Apple e utilizado em dispositivos como iPhones, iPads e iPods Touch. O iOS tem se tornado cada vez mais popular desde o seu lançamento em 2007, desde então tem sido atualizado regularmente com novas funcionalidades e melhorias de desempenho, ele é considerado por muitos acadêmicos como um dos SOs móveis mais avançados do mercado.

De acordo com BORGES (2017), o iOS é considerado mais fácil de usar e mais seguro que o Android. Os usuários também relataram uma melhor experiência de usuário com o iOS, incluindo uma interface mais limpa e intuitiva. Isso se deve principalmente ao fato de que o iOS é um sistema fechado, feito para *hardwares* próprios e com um controle de qualidade e desempenho arrojado, visto que os dispositivos que rodam o sistema são produzidos pela mesma empresa que desenvolve seu *software*, fazendo então com que hajam menos falhas causadas por incompatibilidades ou quebra de diretrizes, como pode acontecer em outros SOs que são livres para utilização em diversos dispositivos.

A segurança também é uma preocupação importante para o iOS, visto que o SO inclui várias medidas de segurança para proteger os dados do usuário, como a autenticação biométrica de toque ou rosto, criptografia de dados, configurações rígidas de privacidade e APPs de segurança integrados. Além disso, o iOS recebe atualizações regulares para corrigir vulnerabilidades de segurança e adicionar novos recursos de segurança. AHVANOOEY et al. (2020).

Segundo GLOBALSTATS (2020), como mostrado nos gráficos das Figura 2.3 e Figura 2.4 em destaque logo abaixo, o iOS tem um mercado geral que corresponde a cerca de ⅓ do mercado que tem o Android, no Brasil, no último ano, esse percentual aumentou, seguindo a tendência do resto do mundo. Muitas coisas influenciam essa diferença de mercado, mas o maior ponto é o preço, visto que para ter os dispositivos da Apple requer um investimento bem maior que um Android, que possui uma grande variedade de dispositivos e portanto preços, essa diferença pode aumentar ainda mais de acordo com a economia do país em questão, como o caso do Brasil, por exemplo.

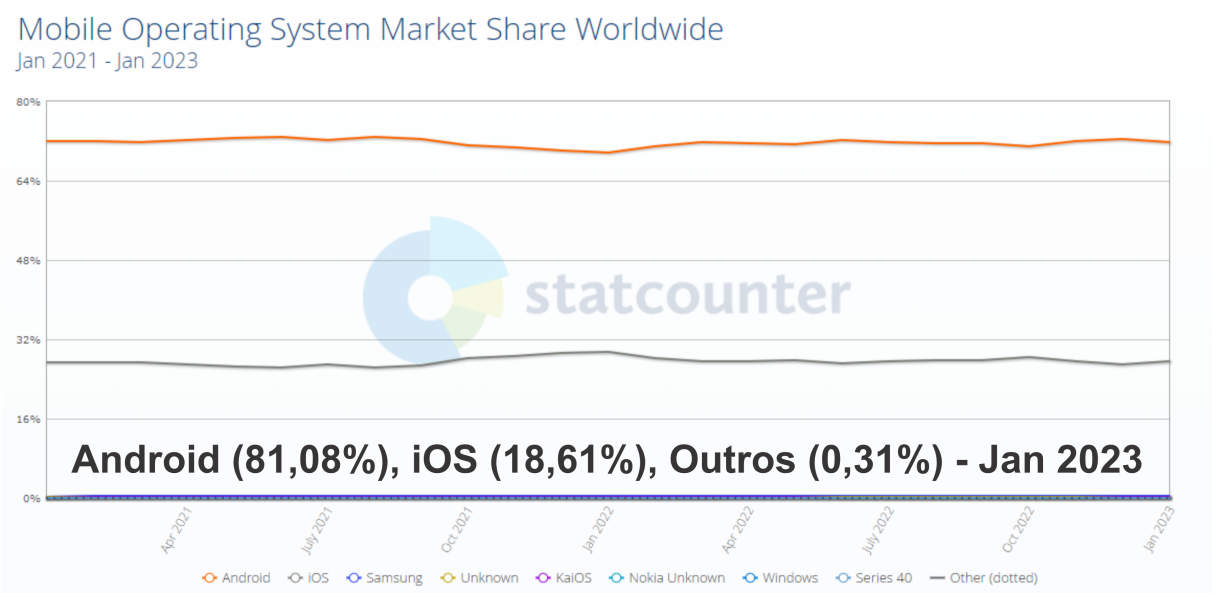

Figura 2.3: Participação no mercado de Sistemas Operacionais móveis em todo o mundo
Fonte: GLOBALSTATS (2020)

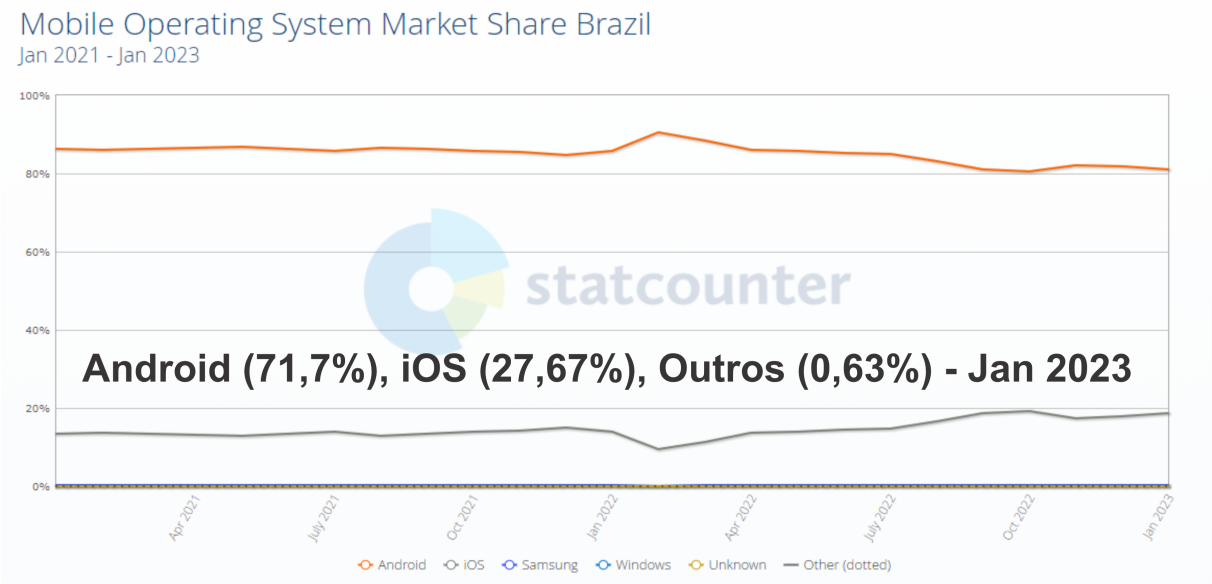

Figura 2.4: Participação no mercado de Sistemas Operacionais móveis no Brasil.
Fonte: GLOBALSTATS (2020)

2.4.2 Android

O SO Android é desenvolvido pela Google e é usado em DMs, como smartphones e tablets. Ele foi lançado em 2008 e tem sido atualizado regularmente com novas funcionalidades e melhorias de desempenho.

De acordo com a pesquisa "Comparativo entre Sistemas Operacionais móveis - Android x iOS" LEITE; MACEDO (2017), o Android tem uma maior flexibilidade e personalização do

que o iOS, permitindo que os usuários ajustem a interface de acordo com suas necessidades. Além disso, os dispositivos Android também tendem a ter uma melhor relação custo-benefício do que os dispositivos iOS.

Embora o Android tenha alguns problemas de segurança adicionais devido à sua abertura e personalização, a Google tem trabalhado para melhorar a confiabilidade do SO, incluindo a implementação de medidas de proteção como a verificação de segurança do Google Play Protect.

Em resumo, o SO Android é considerado flexível, personalizável e com uma boa relação custo-benefício, além de ser uma opção segura e com ampla variedade de APPs disponíveis na Google Play Store, ofertando um leque de APPs comparável à da App Store da Apple, o que a torna uma escolha atraente para muitos usuários. As pesquisas mencionadas acima (LEITE; MACEDO (2017) e BORGES (2017)) apoiam essa avaliação.

Assim como para o iOS, pode-se olhar os números de dispositivos Android no Brasil e no mundo, segundo os mesmos dados de GLOBALSTATS (2020), mostrado nos gráficos das Figura 2.3 e Figura 2.4 acima, o Android domina com grande folga em ambos os cenários, muito influenciado pelo preço e amadurecimento em relação às diversas funcionalidades agregadas ao SO.

2.5 Aplicativos móveis

É indiscutível que os aplicativos móveis tornaram-se uma parte fundamental da vida cotidiana das pessoas, permitindo que elas realizem uma generosa quantidade de tarefas em seus DMs. Com o crescimento contínuo do mercado de aplicativos móveis, o desenvolvimento de aplicativos móveis se tornou uma área em rápida expansão para empresas e desenvolvedores.

Nesta seção, serão tratadas algumas informações gerais sobre o desenvolvimento de APPs e uma visão sobre o desenvolvimento híbrido *versus* desenvolvimento nativo.

2.5.1 Desenvolvimento de aplicativos

O desenvolvimento de aplicativos, ou desenvolvimento de *software* para dispositivos móveis, é um campo em constante evolução, com novas tecnologias e abordagens surgindo a cada dia.

Segundo MARCONI et al. (2021), o desenvolvimento de aplicativos móveis requer um conjunto de habilidades técnicas e conhecimentos específicos em design de interface do usuário, programação e arquitetura de *software*. Além disso, é importante considerar as diferentes plataformas móveis, como as já citadas iOS e Android, e suas respectivas diretrizes de desenvolvimento.

Existem diversas ferramentas e *frameworks* disponíveis para facilitar o processo de desenvolvimento, como o Android Studio para desenvolvimento de APPs Android com *Java* ou *Kotlin*, o Xcode para iOS, usando *Swift* ou *Objective-C*, ou o VSCode — um editor de

código-fonte gratuito e de código aberto desenvolvido pela Microsoft — usado também para as alternativas anteriores, mas principalmente para o desenvolvimento multiplataforma com React Native ou Flutter, que vem ganhando popularidade por sua rapidez e facilidade de uso.

Um dos principais desafios no desenvolvimento de aplicativos móveis é a garantia de que o APP funcione corretamente em diferentes dispositivos e tamanhos de tela. Para isso, é fundamental realizar testes e validações em várias plataformas e dispositivos, a fim de garantir a qualidade do *software* desenvolvido BISWAL; RATH; MOHANTY (2020).

2.5.1.1 Desenvolvimento nativo

O desenvolvimento de *software* nativo para DMs consiste em criar APPs específicos para um determinado SO, utilizando as linguagens de programação e ferramentas oferecidas por esses sistemas, como os já citados Android com *Java* ou *Kotlin* e o iOS com *Swift* ou *Objective-C*. Essa abordagem permite que o APP seja otimizado para o SO em questão, garantindo maior desempenho e uma melhor experiência para o usuário final.

Segundo uma revisão sistemática realizada por SARKER; ALQAHTANI; ALQAHTANI (2021), o desenvolvimento nativo tem sido a escolha mais comum para desenvolvedores de aplicativos móveis, principalmente devido à sua eficiência e desempenho. Além disso, o desenvolvimento nativo permite o acesso completo às APIs e recursos do sistema operacional, o que possibilita a criação de APPs mais avançados e sofisticados.

No entanto, o desenvolvimento nativo também pode ser mais trabalhoso e exigir um conhecimento mais aprofundado das tecnologias envolvidas. De acordo com BISWAL; RATH; MOHANTY (2020), a complexidade do processo de desenvolvimento nativo pode ser amenizada pelo uso de ferramentas e *frameworks* especializados, como o Android Studio e o Xcode.

2.5.1.2 Desenvolvimento híbrido

O desenvolvimento de *software* híbrido para DMs pode ser realizado utilizando diversas tecnologias e *frameworks*, como as duas que estão em alta atualmente: Flutter e React Native. Em resumo, ambos são *frameworks* que permitem criar APPs nativos para Android e iOS a partir de um único código-base, claro que com algumas diferenças entre si, como o desempenho, linguagem, método de criação de componentes, etc.

Uma comparação entre as duas tecnologias foi realizada por GUHA; BANERJEE (2020), que avaliaram a performance e a experiência do usuário em diferentes cenários. Os resultados indicaram que o Flutter apresentou uma performance superior em termos de tempo de carregamento e suavidade de animações, enquanto o React Native foi mais eficiente em termos de uso de memória e processamento. Além disso, ambos os *frameworks* ofereceram uma experiência de usuário semelhante ao desenvolvimento nativo, com recursos como acesso a câmera, geolocalização e notificações *push*.

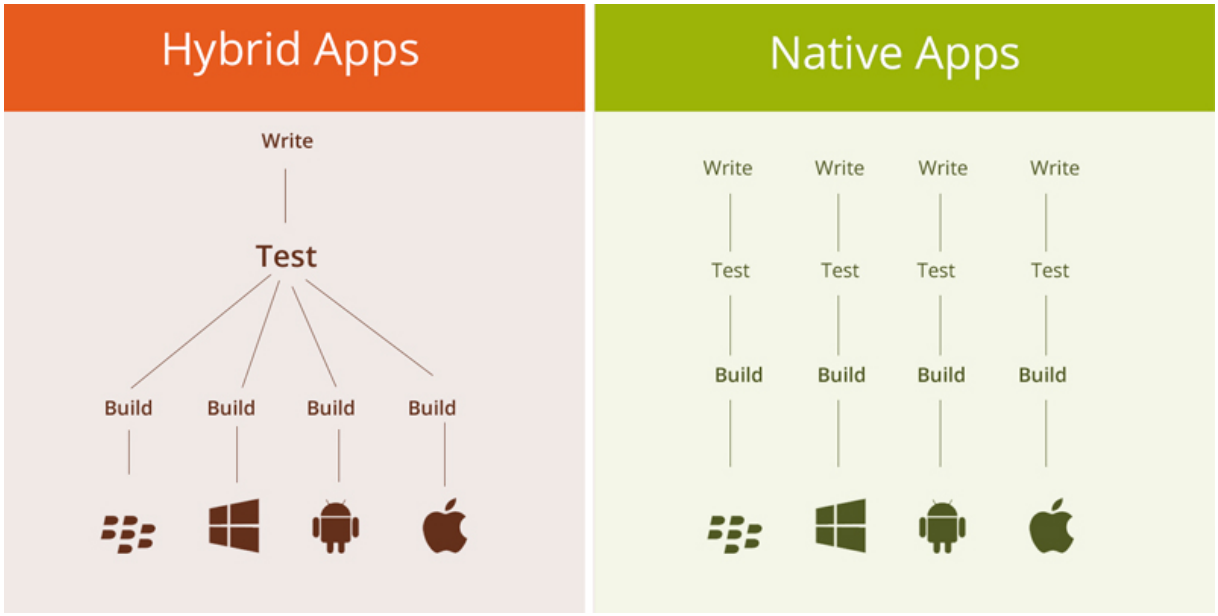

Figura 2.5: Passos do desenvolvimento de APPs híbridos e nativos
Fonte: ANGULARMINDS (2022)

A imagem acima mostra a diferença do desenvolvimento de um APP híbrido (multiplataforma) para um APP nativo, pode-se notar que o desenvolvimento híbrido economiza tempo, esforço e consequentemente dinheiro, por reduzir etapas, tornando-se uma escolha altamente viável e amplamente utilizada atualmente.

Portanto, a escolha entre as duas tecnologias pode depender de diferentes fatores, como a preferência da equipe de desenvolvimento, a complexidade do APP e a disponibilidade de recursos e ferramentas. É importante avaliar as características e limitações de cada tecnologia antes de decidir qual utilizar em um projeto. Neste projeto em particular — para o desenvolvimento do APP — a escolha foi o Flutter, o qual será destrinchado a seguir na seção 2.7.

2.6 APIs *Web*

Application Programming Interface (API), que em português significa "Interface de Programação de Aplicativos", é um conjunto de rotinas e padrões de programação que permitem a integração de diferentes sistemas ou plataformas. É através de uma API que é possível fazer a comunicação entre duas aplicações, permitindo que uma delas utilize os serviços ou dados oferecidos pela outra. As APIs são geralmente disponibilizadas por empresas ou organizações para que desenvolvedores possam criar novas aplicações ou integrar funcionalidades em sistemas existentes. Esse tipo de aplicação utiliza a internet para a transferência de dados e processa e armazena informações relevantes para o funcionamento de um produto ou serviço.

Segundo MARTIN (2008a), as APIs podem ser classificadas em duas categorias principais: as *Web* APIs e as APIs de *software*. As *Web* APIs são acessadas através da internet utilizando protocolos como *Hypertext Transfer Protocol* (HTTP) e geralmente são disponibi-

lizadas por meio de *Uniform Resource Locator*s (URLs) específicas — que em tradução livre significa "Localizador Uniforme de Recursos", ou seja, o endereço web, o texto digitado na barra do navegador para acessar uma determinada página ou serviço — que retornam dados em formatos padronizados como *JavaScript Object Notation* (JSON) ou *Extensible Markup Language* (XML), geralmente usando o padrão *Representational State Transfer* (REST), um estilo arquitetural que define como sistemas distribuídos devem se comunicar. Já as APIs de *software* são usadas para acessar serviços em nível de SO ou *software* e geralmente são escritas em linguagens de programação específicas.

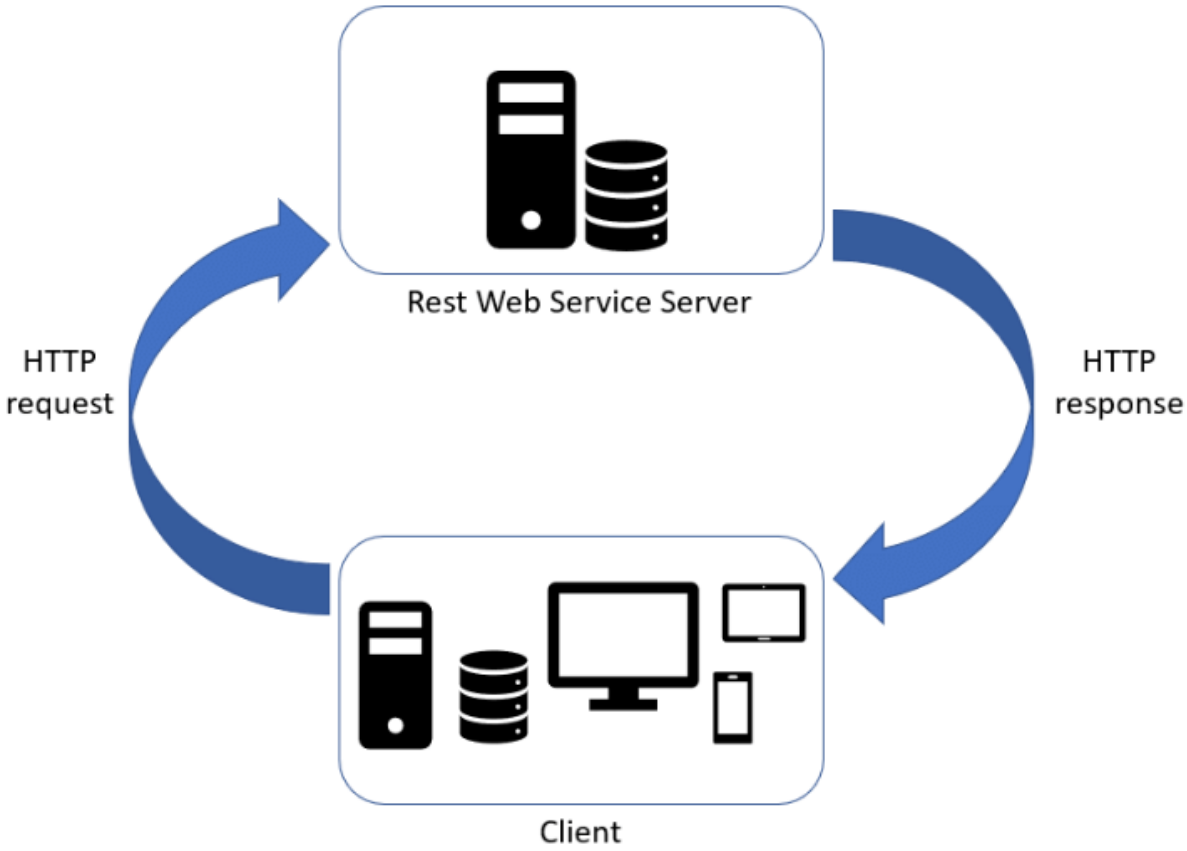

Figura 2.6: Esquema de comunicação síncrona da API REST
Fonte: BERTOLI et al. (2021)

Uma API funciona baseado na conversação entre cliente e servidor, através de um protocolo de comunicação, que fica responsável por transpor os dados entre um e outro, este processo envolve três passos, o primeiro é a geração de uma solicitação pelo cliente ao servidor, geralmente através do protocolo HTTP. Em seguida, se a solicitação exigir acesso ao banco de dados, o servidor realizará a consulta necessária para obter as informações solicitadas. Por fim, após obter a resposta, o servidor a encaminha para o cliente.

2.7 Flutter

O Flutter, principal tecnologia aplicada no desenvolvimento deste APP, é um framework de desenvolvimento de aplicativos móveis, desenvolvido pela Google, criado em 2014 (originalmente chamado de *Sky*), mas que só veio disputar como solução para desenvolvimento multiplataforma em 2018 quando teve sua primeira versão estável lançada, o Flutter permite criar APPs nativos para Android e iOS a partir de um único código-base. Ele utiliza a linguagem de programação *Dart* — também criada pelo Google — que é uma linguagem moderna e orientada a objetos, com recursos avançados como *garbage collection*, tipagem forte e *async/await*.

O Flutter tem ganhado popularidade entre os desenvolvedores devido à sua produtividade e flexibilidade, além de oferecer uma experiência de usuário fluida e rica em animações. Ele também possui um rico conjunto de *widgets* e bibliotecas que tornam a criação de interfaces de usuário complexas mais simples e intuitivas.

Uma avaliação do Flutter em relação a outras tecnologias de desenvolvimento de aplicativos móveis foi realizada por ZHOU; ZHANG; HUANG (2021), que analisaram a usabilidade, desempenho e eficiência do Flutter em comparação com o React Native e o desenvolvimento nativo. Os resultados indicaram que o Flutter ofereceu uma experiência de usuário superior em relação ao React Native e uma performance comparável ao desenvolvimento nativo, além de apresentar uma curva de aprendizado mais suave e uma maior eficiência no desenvolvimento de protótipos. Estes pontos reforçam a escolha do Flutter como principal *framework* escolhido para o projeto.

Dados de interesse do Google[2] corroboram o crescimento do Flutter em relação ao seu principal concorrente, como mostrado na figura abaixo.

Figura 2.7: Tendência mundial de popularidade do Flutter (vermelho) e do React Native (azul) (2018–2023).
Fonte: Google Trends[2]

[2]Google Trends: `https://trends.google.com/trends/explore?date=2018-01-02%202023-01-03&q=%2Fg%2F11h03gfxy9,%2Fg%2F11f03_rzbg`

Como explicado na página oficial sobre a arquitetura do Flutter[3], a mesma é dividida em três camadas: o *Framework*, *Engine* e o *Embedder*. Ele existe como uma série de bibliotecas independentes, cada uma dependendo da camada subjacente.

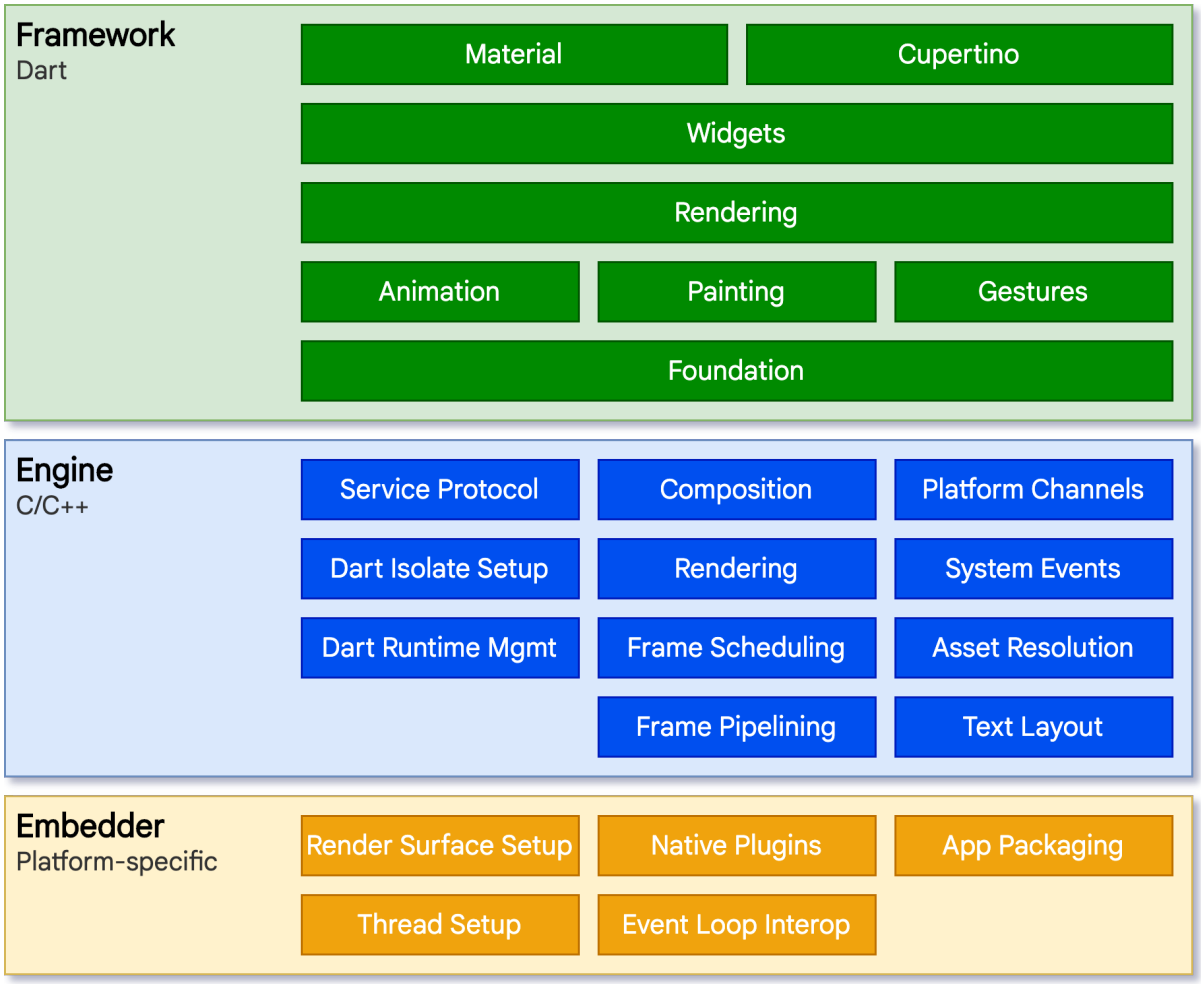

Figura 2.8: Visão geral da arquitetura do Flutter
Fonte: Flutter[3]

De forma resumida, a camada do *framework* é moderna e reativa, escrito na linguagem *Dart*, ela fornece uma API de nível superior para criar APPs de alta qualidade (por exemplo, *widgets*, teste de cliques, detecção de gestos, acessibilidade, entrada de texto). Já a camada de *engine* é escrita em *c++* e fornece implementação de baixo nível das principais APIs do Flutter. Por fim o *embedder* é o responsável por fornecer um ponto de entrada para aplicações e por coordenar acesso aos serviços dos SOs como superfícies de renderização, acessibilidade e entrada, gerenciando o *loop* de eventos de mensagens.

Outro ponto levado em consideração para a escolha do Flutter para o projeto, que também é um reflexo de seu crescimento, é que o mesmo se tornou um *framework* multiplataforma

[3]Visão geral da arquitetura do Flutter: `https://docs.flutter.dev/resources/architectural-overview`

muito poderoso, que, segundo o site oficial do Flutter[4] atualmente permite ser compilado para aplicações web, desktop (MacOS, Windows e Linux), dispositivos móveis (Android e iOS) e até dispositivos embarcados a partir de um mesmo código fonte. Desta forma, sendo possível expandir futuramente este projeto para uma ampla gama de dispositivos além dos DMs.

2.8 Código Limpo

O projeto foi desenvolvido usando as técnicas e referencias do livro "Código Limpo: Habilidades Práticas do *Agile Software*", publicado em 2008, por Robert C. Martin, também conhecido como "*Uncle Bob*" ou em português "Tio Bob", a seguir, uma breve definição: O Código Limpo é um conjunto de práticas e princípios para escrever código de alta qualidade, fácil de entender e manter. Essa abordagem de programação foi popularizada pelo Tio Bob, em seu livro acima citado.

Um código limpo deve ser legível, expressivo, ter baixa complexidade, ser fácil de testar e modular MARTIN (2008a). Além disso, ele deve seguir princípios citados como "S.O.L.I.D.", um acrônimo criado por Michael Feathers que representa os 5 princípios da programação orientada a objetos.

“Os programas escritos nessas linguagens podem parecer estruturados e orientados a objetos, mas a aparência pode enganar. Com muita frequência, os programadores de hoje desconhecem os princípios que são a base das disciplinas das quais suas linguagens foram derivadas. [...] Esses princípios expõem os aspectos de gerenciamento de dependência do design orientado a objetos em oposição aos aspectos de conceituação e modelagem. Isso não quer dizer que a OO seja uma ferramenta ruim para a conceituação do espaço do problema ou que não seja um bom local para a criação de modelos. Certamente muitas pessoas obtêm valor desses aspectos da OO. Os princípios, no entanto, concentram-se fortemente no gerenciamento de dependências.” MARTIN (2005).

A seguir, os cinco princípios abordados a serem seguidos para a criação de um bom código:

1. **SRP** - Princípio da Responsabilidade Única (*Single Responsibility Principle*): Uma classe deve ter um, e somente um, motivo para mudar.

2. **OCP** - Princípio do Aberto/Fechado (*Open/Closed Principle*): Você deve ser capaz de estender um comportamento de uma classe sem a necessidade de modificá-lo.

3. **LSP** - Princípio da Substituição de Liskov (*Liskov Substitution Principle*): As classes derivadas devem ser substituíveis por suas classes bases.

4. **ISP** - Princípio da Segregação de Interfaces (*Interface Segregation Principle*): Muitas interfaces específicas são melhores do que uma interface única geral.

[4]Flutter: `https://flutter.dev/multi-platform`

5. **DIP** - Princípio da Inversão de Dependência (*Dependency Inversion Principle*): Dependa de abstrações e não de implementações.

A aplicação desses princípios e práticas de código limpo pode levar a benefícios como redução do tempo de desenvolvimento, maior facilidade de manutenção, maior eficiência e redução de bugs. O que foi notado no desenvolvimento deste projeto.

2.9 Arquitetura Limpa

A chamada Arquitetura Limpa ou *Clean Architecture* é um tema bastante abordado atualmente na área de desenvolvimento de *software*. Esse conceito também foi criado por Robert C. Martin e tem como objetivo principal a criação de sistemas de *software* de alta qualidade, duráveis e que possam ser facilmente modificados e mantidos. A Arquitetura Limpa é baseada em três princípios fundamentais: separação de interesses, independência de *frameworks* e testabilidade MARTIN (2018).

A separação de interesses é um dos pilares da Arquitetura Limpa. Ela se baseia na ideia de que as diferentes camadas de uma aplicação devem ser isoladas umas das outras, permitindo que cada uma delas possa ser modificada sem afetar as demais. Essa separação pode ser feita de diversas formas, mas é comum a utilização de padrões como o MVC (Model-View-Controller), MVVM (Model-View-ViewModel) e MVP (Model-View-Presenter).

A independência de frameworks também é um conceito importante na Arquitetura Limpa. De acordo com o Tio Bob, os *frameworks* devem ser vistos como detalhes de implementação e não como parte da arquitetura. Isso significa que o código deve ser escrito de forma a não depender diretamente de nenhum *frameworks* específico, facilitando a migração para outros *frameworks* ou até mesmo para outras plataformas.

Por fim, a testabilidade é um princípio fundamental na Arquitetura Limpa. Segundo Martin, o código deve ser escrito de forma a ser facilmente testado, permitindo que os testes sejam executados com rapidez e eficiência. Para isso, é importante que as diferentes camadas da aplicação estejam bem separadas, permitindo que cada uma delas possa ser testada de forma independente.

2.10 Desenvolvimento de sistemas

O desenvolvimento de sistemas é um processo que envolve diversas etapas, desde a concepção do projeto até a entrega do produto final. Segundo PRESSMAN (2016), a fase de análise de requisitos é fundamental para entender as necessidades do cliente e garantir que o sistema desenvolvido atenda às expectativas.

Já SOMERVILLE (2015) destaca a importância da fase de design na criação de uma arquitetura que atenda aos requisitos do sistema, bem como na escolha de tecnologias e ferra-

mentas adequadas para a implementação. Além disso, o processo de teste é fundamental para garantir a qualidade do sistema desenvolvido, de acordo com MYERS; SMEDEMA; TURBITT (2012).

Por fim, é importante ressaltar que o desenvolvimento de sistemas é uma área em constante evolução e que novas metodologias e práticas surgem frequentemente, como o desenvolvimento ágil, conforme destacado por MARTIN (2008b).

2.10.1 Metodologias ágeis

As metodologias ágeis são uma abordagem de desenvolvimento de *software* que prioriza a interação contínua com o cliente, adaptação rápida às mudanças e entrega frequente de funcionalidades em pequenos incrementos. Essas metodologias se baseiam em valores e princípios que enfatizam a colaboração, o trabalho em equipe, a comunicação constante e a flexibilidade, em contraste com as abordagens mais tradicionais que tendem a ser mais burocráticas e hierárquicas.

As metodologias ágeis se tornaram cada vez mais populares nos últimos anos, principalmente devido à sua eficácia em projetos complexos e dinâmicos, bem como ao seu foco no valor entregue ao cliente. A seguir, uma breve explicação sobre duas metodologias que foram mutualmente aplicadas no desenvolvimento do sistema.

2.10.1.1 Scrum

O Scrum é uma metodologia ágil baseada em uma estrutura de trabalho em equipe que divide o projeto em sprints, ciclos de tempo definidos, com objetivos claros e bem definidos. Durante cada sprint, a equipe trabalha em conjunto para atingir esses objetivos, fazendo ajustes e melhorias ao longo do caminho. As equipes de desenvolvimento são auto-organizadas e multidisciplinares, o que permite que cada membro tenha uma visão ampla do projeto e possa contribuir com suas habilidades e experiências específicas. Além disso, a metodologia promove a comunicação constante entre as equipes, com reuniões diárias para acompanhar o progresso e identificar possíveis problemas.

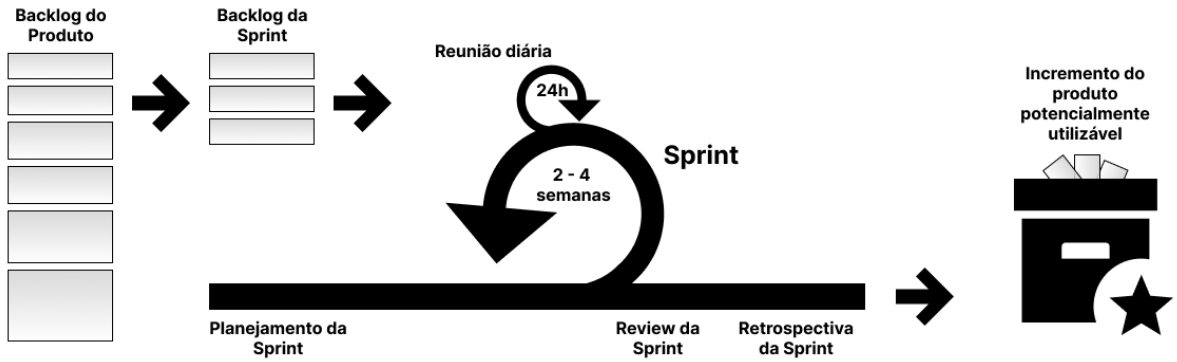

Figura 2.9: Representação de uma Sprint com o framework Scrum.
Fonte: Adaptação de SUTHERLAND (2014)

De acordo com o processo ágil do Scrum, existem cinco eventos (também chamados de cerimônias). São eles:

1. Reunião de planejamento (*Planning Meeting*): Para determinar o que pode ser entregue na Sprint que se inicia, e como o trabalho será feito.

2. Reunião diária (*Daily Scrum*): Realiza-se uma reunião de no máximo 15 minutos, idealmente no mesmo horário, nesta reunião o time inspeciona rapidamente seu trabalho e planeja o trabalho a ser feito para as próximas 24 horas.

3. Reunião de revisão da Sprint (*Sprint Review*): Ocorre tipicamente no último dia. Ela tem como objetivo inspecionar as entregas feitas pelo time durante a Sprint e adaptar o backlog do produto se necessário.

4. Reunião de retrospectiva da Sprint (*Sprint Retrospective*): Nesta reunião o time expõe de forma bastante transparente o que está funcionando e o que deve ser melhorado na maneira que o time está trabalhando.

5. A Sprint em si, a qual engloba os demais eventos acima.

2.10.1.2 Kanban

O Kanban é uma metodologia ágil que surgiu no Japão, em meados da década de 1940, com o objetivo de aumentar a eficiência do processo produtivo nas fábricas da Toyota. O método baseia-se na visualização do fluxo de trabalho por meio de um quadro Kanban, que mostra as tarefas a serem realizadas, em andamento e concluídas, permitindo que a equipe visualize o processo e identifique gargalos e desperdícios.

É comum acharem que o Kanban é uma ferramenta do Scrum, o que e não é verdade, ele nada mais é que uma ferramenta que vai ajudar o Scrum a ser mais eficaz no controle da quantidade de trabalho em progresso. Hoje ele vem sendo utilizando largamente no desenvolvimento de *software*, e com sua adoção em projetos de *software*, tem-se um processo mais transparente, colaborativo e ágil, possibilitando um melhor gerenciamento do trabalho em equipe.

O quadro geralmente é autoexplicativo, suas colunas são flexíveis, sendo adaptadas de acordo com o uso, mas existem pelo menos três, são elas:

1. Para fazer (*TODO*): Tarefas pendentes.

2. Fazendo (*Doing*): Tarefas em progresso.

3. Feito (*Done*): Tarefas concluídas.

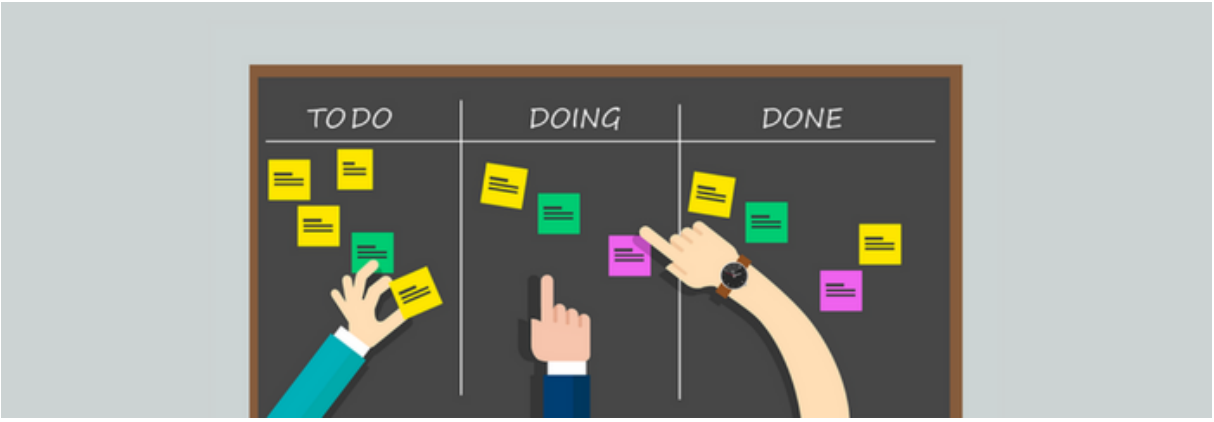

Figura 2.10: Representação de um quadro de desenvolvimento com o framework Kanban.
Fonte: ÁGIL (2017)

2.11 Prototipagem

A prototipagem é uma técnica importante no processo de desenvolvimento de *software*, permitindo aos desenvolvedores criar modelos iniciais do produto para avaliar e testar sua funcionalidade e usabilidade, essencial na construção de um Produto Viável Mínimo (MVP). Ao criar protótipos, os desenvolvedores podem obter feedback dos usuários e identificar problemas com antecedência, o que pode ajudar a evitar atrasos e aumentar a satisfação do cliente. De acordo com FIDEL (2003), a prototipagem é considerada uma das práticas mais importantes no design de interação, ajudando a reduzir os custos de desenvolvimento e melhorar a eficiência do processo de design.

Neste projeto, a ferramenta de prototipagem utilizada majoritariamente foi o Figma, que será explicado na subseção abaixo.

2.11.1 Figma

O Figma é uma das diversas ferramentas de prototipagem de interfaces de usuário existente e é amplamente utilizada em projetos de desenvolvimento de *software*. Sua interface intuitiva e recursos colaborativos facilitam a criação de protótipos interativos, que permitem aos designers e desenvolvedores visualizar como o produto final funcionará. Ele também oferece recursos de design, como a criação de ícones, gráficos e wireframes, permitindo que os designers criem designs personalizados e aprimorem a aparência geral do produto.

Além disso, o Figma suporta o compartilhamento de projetos, permitindo que vários membros da equipe colaborem em tempo real e facilitem a comunicação. Com esses recursos, ele tornou-se uma ferramenta valiosa para equipes de desenvolvimento de *software* que buscam agilidade e eficiência no processo de prototipagem e design.

3

Metodologia

Neste capítulo é apresentada a metodologia utilizada para o desenvolvimento do aplicativo. [EM DESENVOLVIMENTO até 31/03/23]

4

Desenvolvimento

Neste capítulo serão abordados diversos aspectos do desenvolvimento do sistema, desde a configuração do ambiente de desenvolvimento até o fluxo de trabalho utilizado durante o processo de criação da aplicação. Serão apresentados os desafios encontrados durante o desenvolvimento, bem como as soluções adotadas para superá-los, além de uma visão geral sobre as funcionalidades do sistema e as tecnologias utilizadas para sua criação.

4.1 Especificação de requisitos

O pontapé inicial do projeto foi uma reunião entre as partes envolvidas, cliente (docentes de Agronomia da Universidade Federal do Agreste de Pernambuco (UFAPE)), *Product Owner* (PO) (Proprietário do produto, em tradução livre) e desenvolvedores, a fim de levantar os requisitos funcionais do MVP. Nesta reunião foram explanados os pontos onde o APP auxiliaria no gerenciamento de experimentos e no processo dos cálculos enzimáticos, foram discutidas regras de negócios e requisitos necessários, identidade visual, além do início de um protótipo de baixa fidelidade para conduzir o *brainstorming* a respeito de como os requisitos se transformariam em funcionalidades, e claro, as tecnologias utilizadas em todo o ciclo de desenvolvimento.

O projeto surge com necessidade de um APP para celulares, facilitando a mobilidade dos usuários nos locais de coletas de dados para os experimentos, dessa forma, tudo foi pensado voltado ao desenvolvimento deste aplicativo móvel.

4.1.1 Requisitos funcionais

A partir disso, os requisitos funcionais do sistema foram definidos e especificados de acordo com a Tabela 4.1, abaixo.

Código	**Descrição**
RF01	O usuário deve ser capaz de se cadastrar no sistema
RF02	O usuário deve ser capaz de realizar *login* no sistema
RF03	O usuário deve ser capaz de recuperar sua senha no sistema
RF04	O usuário deve ser capaz de criar um tratamento
RF05	O usuário deve ser capaz de deletar um tratamento
RF06	O usuário deve ser capaz de visualizar todos os tratamentos disponíveis no sistema para seus experimentos
RF07	O usuário administrador deve ser capaz de criar uma nova enzima
RF08	O usuário administrador deve ser capaz de deletar uma enzima
RF09	O usuário deve ser capaz de visualizar todas as enzimas disponíveis no sistema para seus experimentos
RF10	O usuário deve ser capaz de criar um novo experimento
RF11	O usuário deve ser capaz de deletar um experimento
RF12	O usuário deve ser capaz de visualizar seus experimentos cadastrados
RF13	O usuário deve ser capaz de utilizar filtros de busca para visualizar seus experimentos cadastrados
RF14	O usuário deve ser capaz de visualizar detalhes dos seus experimentos cadastrados
RF15	O usuário deve ser capaz de inserir dados para o cálculo enzimático no seu experimento
RF16	O usuário deve ser capaz de visualizar resultados discrepantes do cálculo enzimático antes de salvar no seu experimento
RF17	O usuário deve ser capaz de editar o resultado do cálculo enzimático antes de salvar no seu experimento
RF18	O usuário deve ser capaz de salvar o resultado do cálculo enzimático no seu experimento
RF19	O usuário deve ser capaz de visualizar os resultados e o progresso do seu experimento
RF20	O usuário deve ser capaz de exportar os resultados do seu experimento

Tabela 4.1: Descrição dos requisitos funcionais do sistema

Segundo PRESSMAN (2016), os requisitos funcionais, que descrevem as funções e serviços que o software deve oferecer, descrevem como o software deve funcionar, quais entradas devem ser processadas e quais saídas devem ser geradas em resposta a essas entradas. Neste sentido, os requisitos funcionais descritos na Tabela 4.1 têm o objetivo de especificar as funcionalidades que o sistema proposto deve oferecer.

4.1.2 Casos de uso

Os Casos de Uso (*Use Cases*) são uma técnica de modelagem de requisitos de software que descrevem as interações entre o usuário e o sistema. Eles ajudam a identificar as funcionalidades e serviços que o software deve oferecer, além de fornecer um meio de comunicação claro entre as equipes de desenvolvimento e os *stakeholders*. A partir disto, foi elaborado um conjunto de casos de uso do sistema, demonstrado na Tabela 4.2, com o objetivo de centrar os requisitos funcionais descritos na Tabela 4.1 e relacionar os mesmos com os tipos de usuários do sistema.

Código	**Descrição**	**Tipo de usuário**	**Requisitos funcionais contemplados**
UC01	Fazer cadastro no sistema	Usuário comum, Administrador	RF01
UC02	Fazer *login* no sistema	Usuário comum, Administrador	RF02
UC03	Recuperar senha no sistema	Usuário comum, Administrador	RF03
UC04	Criar, deletar, visualizar e usar tratamentos	Usuário comum, Administrador	RF04, RF05 e RF06
UC05	Criar e deletar enzimas	Administrador	RF07 e RF08
UC06	Visualizar e usar enzimas	Usuário comum, Administrador	RF09
UC07	Criar, deletar, filtrar e visualizar experimentos	Usuário comum, Administrador	RF10, RF11, RF12 e RF13
UC08	Visualizar detalhes do experimento	Usuário comum, Administrador	RF14
UC09	Inserir dados, editar e salvar o cálculo enzimático no experimento	Usuário comum, Administrador	RF15, RF16, RF17 e RF18
UC10	Visualizar resultados do experimento	Usuário comum, Administrador	RF19
UC11	Compartilhar resultados do experimento	Usuário comum, Administrador	RF20

Tabela 4.2: Casos de Uso do sistema

Uma reprodução visual dos Casos de Uso apontados na Tabela 4.2 e as suas interações com os usuários foi elaborada, na Figura 4.1. Este diagrama utiliza *Unified Modeling Language* (UML) — em português: Linguagem de Modelagem Unificada — uma linguagem para visualização, especificação, construção e documentação de artefatos de um software em desenvolvimento, desta forma o Diagrama de *Use Cases* [1] utilizado abaixo tem o objetivo de auxiliar a comunicação entre os analistas e o cliente, descrevendo um cenário que mostra as

[1] Diagrama de *Use Cases*: `http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/uml/diagramas/usecases/usecases.htm`

funcionalidades do sistema do ponto de vista do usuário. O cliente deve ver neste diagrama as principais funcionalidades de seu sistema.

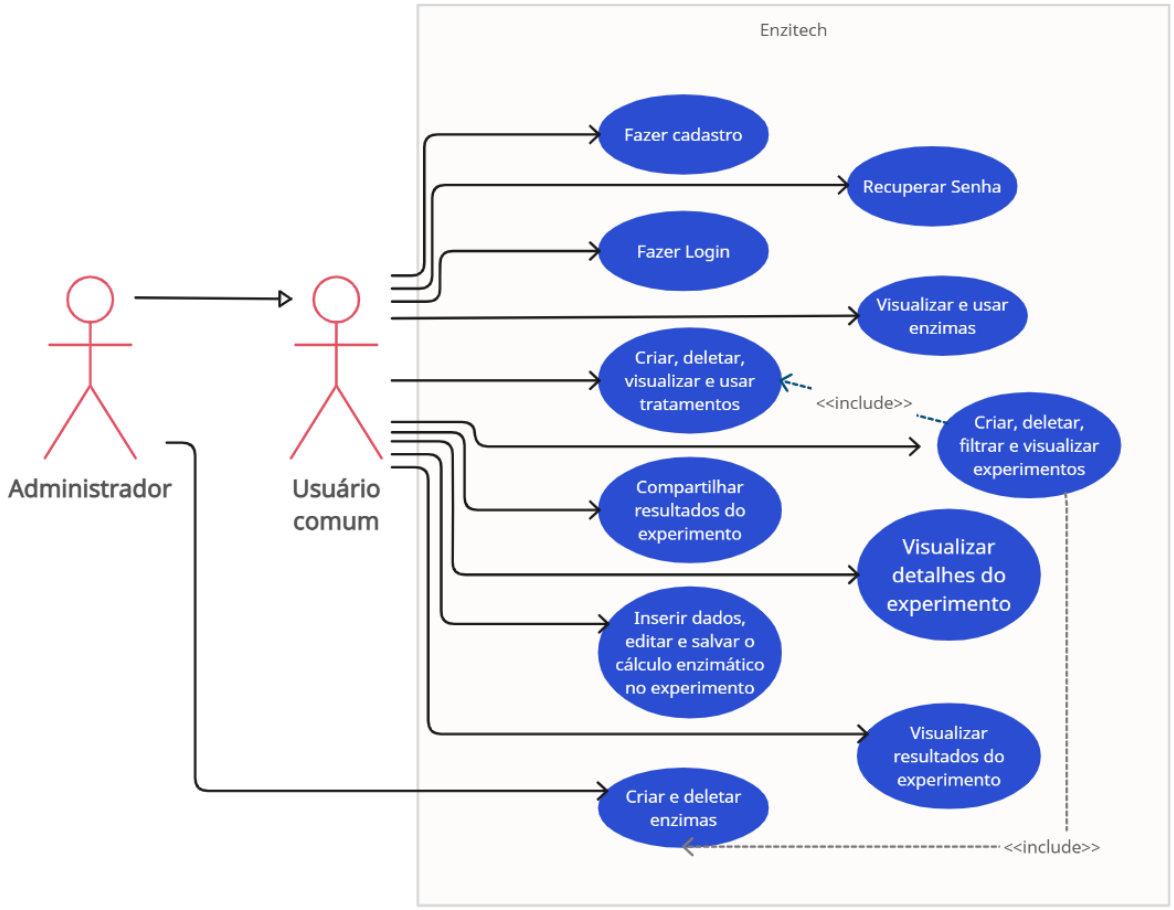

Figura 4.1: Diagrama de Casos de Uso do sistema.

Graças ao diagrama acima, é possível notar, de forma simples, que um usuário administrador possui um privilégio em comparação ao usuário comum: a criação e exclusão das enzimas, este Caso de Uso que ficou restrito à administração do sistema por ser compartilhada entre todos os usuários do Enzitech, podendo causar perdas de dados se sua alteração for realizada por um usuário comum, portando, surgiu a necessidade de diferenciar ambos os tipos de usuários no sistema e realizar estas validações para a disponibilização dessa *feature*.

4.2 Protótipo

WARFEL (2009) destaca que os protótipos de baixa fidelidade são rápidos e baratos de criar, permitindo que os *designers* testem e refinem rapidamente ideias iniciais. Já os protótipos

de alta fidelidade, embora mais caros e demorados, podem ajudar a obter *feedback* mais preciso e detalhado sobre o *design* e a usabilidade do software.

Dentre as diversas reuniões do projeto, houve a discussão das funcionalidades para gerar um protótipo de baixa fidelidade, para possíveis refinamentos, antes da elaboração de um protótipo de alta fidelidade, que demanda mais tempo e esforço. Junto desse protótipo, foi idealizado a identidade visual e o nome oficial para o projeto, que passou a se chamar "Enzitech".

Figura 4.2: Protótipo final de baixa fidelidade.

Como é possível ver na Figura 4.2, o protótipo de baixa fidelidade trás o desenho das principais funcionalidades do sistema, além de dar uma ideia do layout final da aplicação.

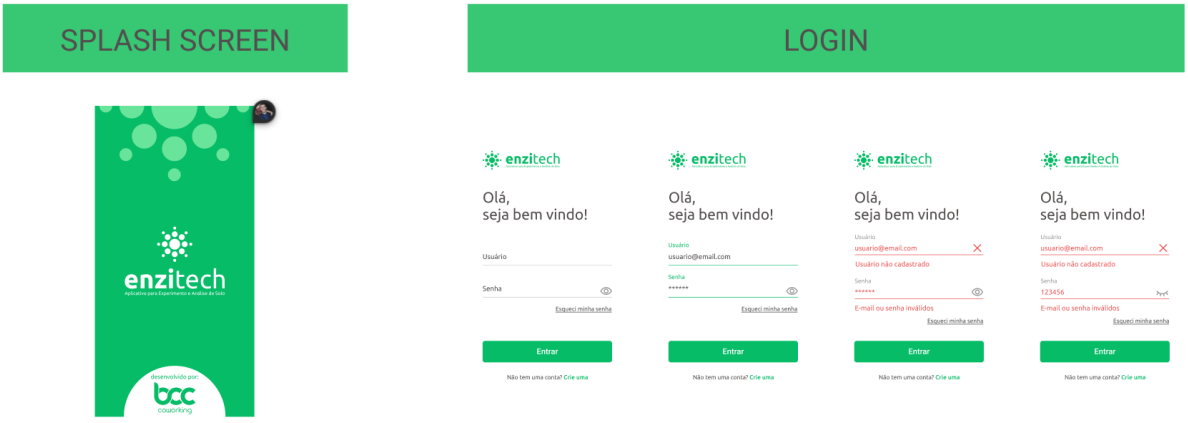

Figura 4.3: Parte do protótipo final de alta fidelidade.

Já nas Figura 4.3 e Figura 4.4, é possível verificar mais a fundo todo o *design-system*

aplicado, averiguar comportamentos das telas, fluxo de navegação, entre outras vantagens, trazendo uma versão muito mais refinada da aplicação, auxiliando o desenvolvimento e facilitando a visão do sistema para o cliente, onde antes era definido somente por Casos de Uso.

Figura 4.4: Visão geral do protótipo final de alta fidelidade.

4.3 Detalhes técnicos do aplicativo

Nesta seção, serão apresentados aspectos como tecnologias utilizadas, os passos necessários para a configuração do ambiente de desenvolvimento, a instalação do Flutter, ferramentas e bibliotecas utilizadas, integrações, gerenciamento de dependências, modelos de dados e o fluxo de trabalho.

4.3.1 Instalação do Flutter

Antes de tudo, é necessário elucidar os requisitos para montar um ambiente de desenvolvimento voltado à criação de APPs em Flutter, este *Software Development Kit* (SDK) está disponível nos principais SOs de PCs, ChromeOS, macOS, Linux e Windows, sendo este último o escolhido como ambiente para o desenvolvimento desta aplicação, o qual será explicado o processo de instalação aqui.

De acordo com o site oficial[2], para instalar e executar o Flutter, o ambiente de desenvolvimento deve atender a estes requisitos mínimos:

- Windows 10 ou posterior (64 bits), baseado em x86-64;

[2]Instalação do Flutter no Windows: `https://docs.flutter.dev/get-started/install/windows`

- 1,64 GB de espaço livre no disco (não inclui espaço em disco para IDE/ferramentas);
- Ferramentas no ambiente do Windows:
  - Windows PowerShell 5.0 ou mais recente (pré-instalado com o Windows 10);
  - Git para Windows 2.x, com a opção "Use o Git no prompt de comando do Windows";

Com todos os requisitos preenchidos, basta seguir o passo a passo detalhado como consta no site oficial[2].

4.3.2 Instalação do Android Studio

Como consta no tutorial oficial[2], a instalação do Android Studio é essencial, porém neste passo, gosto de utilizar o *JetBrains Toolbox App*[3], que faz a instalação e futuras atualizações em poucos cliques. Após instalar o *JetBrains Toolbox App*, basta achar o Android Studio e clicar em instalar.

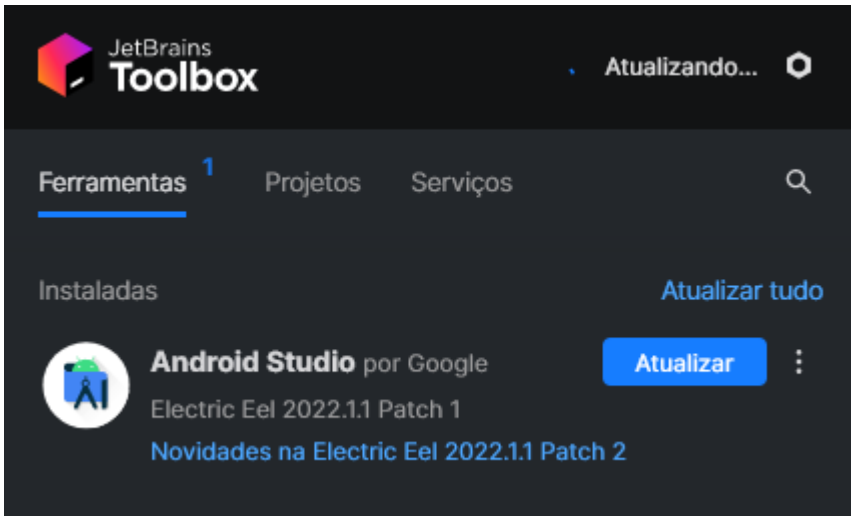

Figura 4.5: *JetBrains Toolbox App*

Após instalado, é necessário abrir o Android Studio, fazer sua configuração inicial e configurar mais alguns passos, dentre eles, fazer a instalação de um emulador Android para utilizar no desenvolvimento, caso não queira utilizar um dispositivo físico, veja a seguir:

1. Habilitar a "aceleração de VM" em sua máquina Windows;
2. Na tela inicial do Android Studio, clique no ícone *AVD Manager* e selecione *Create Virtual Device...*;

[3] *JetBrains Toolbox App*: `https://www.jetbrains.com/pt-br/toolbox-app/`

3. Escolha uma definição de dispositivo e selecione "Avançar";
4. Selecione uma ou mais imagens do sistema para as versões do Android que deseja emular e selecione Avançar. Uma imagem x86 ou x86_64 é recomendada;
5. Na aba de Performance do emulador, selecione Hardware - GLES 2.0 para habilitar a aceleração de hardware;
6. Verifique se tudo está correto e selecione "Concluir";
7. Por fim, basta executar o emulador e ele iniciará;

Outro passo imprescindível é ativar as ferramentas SDK necessárias para a compilação do projeto Flutter em um arquivo executável .APK para o Android, basta seguir este passo a passo:

1. Na tela inicial do Android Studio, clicar em *File* > *Settings* > *Appearance & Behavior* > *System Settings* > *Plugins*;
2. Na aba *Marketplace*, busque e instale os *plugins* "Dart" e "Flutter";
3. Nesta mesma tela, no menu lateral, clique em *Appearance & Behavior* > *System Settings* > *Android SDK*;
4. Na aba *SDK Tools*, verifique se existem versões instaladas para *Android SDK Build-Tools 34-rc2*, caso não, instale a correspondente à versão do Android escolhida na criação do emulador ou a do seu dispositivo físico;
5. Ainda na aba *SDK Tools*, verifique se existe pelo menos uma versão instalada para *Android SDK Command-line Tools*, caso contrário, instale a última versão disponível;
6. Ainda na aba *SDK Tools*, verifique se o *Android SDK Platform-Tools* está instalado, caso contrário, instale;
7. Opcional: Ainda na aba *SDK Tools*, verifique se o *Intel x86 Emulator Accelerator (HAXM installer)* está instalado, caso contrário, instale (esta ferramenta melhora a performance do emulador caso seu sistema suporte);

4.3.3 Instalação do VSCode

É possível seguir com o desenvolvimento no Android Studio, porém ele é um software que exige bastante processamento, o que pode "pesar" no sistema, portanto, existe a possibilidade de utilizar o VSCode[4] para seguir como software padrão de desenvolvimento, além disso, no VSCode é possível instalar uma grande quantidade de pacotes desenvolvidos pela comunidade

[4]VSCode: `https://code.visualstudio.com/`

que agilizam o desenvolvimento, para utilizar o VSCode basta instalá-lo pela loja do SO, no caso do Windows, a *Microsoft Store* ou pelo site oficial[4].

Após instalado e feito a configuração inicial, basta buscar e instalar os *plugins* "Dart" e "Flutter", outros de sua preferência também podem ser instalados. Por fim, basta criar (seguindo o tutorial oficial[5]) ou abrir um projeto Flutter já existente e começar a editar seu código.

4.3.4 Ferramentas e bibliotecas de suporte

Além dos *Integrated Development Environment*s (IDEs), outras ferramentas foram utilizadas para o desenvolvimento do sistema, nesta subseção abordo brevemente cada uma e sua aplicação.

4.3.4.1 Flutter DevTools

O DevTools[6] é um conjunto de ferramentas de desempenho e depuração para Dart e Flutter, vem integrado na instalação do Flutter e executa automaticamente na inicialização de *debugging*, suas principais funções são:

1. Inspecionamento do layout da interface do usuário e o estado de um aplicativo Flutter;
2. Diagnóstico de problemas de instabilidade no desempenho da interface do usuário em um aplicativo Flutter;
3. Visualização de informações gerais de *log* e diagnóstico sobre um aplicativo de linha de comando Flutter ou Dart em execução;

4.3.4.2 Postman

O Postman[7] é uma ferramenta de teste de API que permite aos desenvolvedores testar, documentar e compartilhar suas APIs. Com o Postman, é possível enviar solicitações HTTP, verificar as respostas da API, criar scripts de teste automatizados e compartilhar coleções de solicitações com outras pessoas. A ferramenta é amplamente utilizada no desenvolvimento de software e pode ser integrada a outras ferramentas, como o Swagger[8], para tornar o processo de teste e documentação de APIs mais eficiente.

4.3.4.3 *Packages* do pub.dev

Os *packages* do pub.dev[9] são bibliotecas de código-fonte aberto, desenvolvidas por membros da comunidade Flutter, que podem ser utilizadas para implementar recursos em APPs

[5]Escreva seu primeiro aplicativo Flutter: `https://docs.flutter.dev/get-started/codelab`
[6]DevTools: `https://docs.flutter.dev/development/tools/devtools/overview`
[7]Postman: `https://www.postman.com/`
[8]Swagger: `https://swagger.io/`
[9]pub.dev: `https://pub.dev/`

Flutter. Essas bibliotecas fornecem funcionalidades prontas para uso, como a integração com APIs, manipulação de imagens, gerenciamento de estado, entre outras.

Ao utilizar um *package* do pub.dev, os desenvolvedores podem economizar tempo e esforço na implementação de funcionalidades comuns, além de poderem se beneficiar de contribuições e correções de *bugs* feitas pela comunidade. Os *packages* podem ser facilmente adicionados ao projeto por meio do arquivo PUBSPEC.YAML e instalados por meio do comando `flutter pub get`.

4.4 Sprints

Como foi explicado na seção de Metodologias ágeis, o projeto seguiu duas metodologias que guiaram a criação do APP, o Scrum e o Kanban, muitas vezes referenciado por profissionais como "Scranban" ou algo semelhante, somente um nome genérico que aponte a junção dessas duas metodologias, portanto, o desenvolvimento do projeto foi dividido em sprints, houve um acordo entre a equipe sobre a flexibilidade das mesmas, visto que todos os envolvidos teriam que alinhar seus horários para a conclusão do projeto, ponderando entre sua vida acadêmica e profissional, o que acarretou em uma maior duração, mas não em menor desempenho ou qualidade, assim, impossibilitando manter uma sprint fixa de 2 a 4 semanas como sugerido pelo *framework* Scrum.

Como o uso do Kanban também estava presente, para a sprint ser iniciada havia um requisito: as atividades deveriam ser puxadas da coluna "TODO" na ordem de sua prioridade, portanto, com base na importância dos mesmos, tendo como foco a conclusão de um MVP. Logo que o APP atingiu os requisitos mínimos esperados, foi gerado uma versão *beta* e liberada para os usuários internos testarem, a fim de obter alguns *feedbacks*.

Abaixo é possível ver uma demonstração do quadro de atividades do projeto Enzitech no Github Project[10], plataforma utilizada como quadro adaptável, que se integra às *issues* e solicitações de *pull* no GitHub para ajudar no planejamento e acompanhar o projeto com eficiência.

[10]Github Project: `https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects`

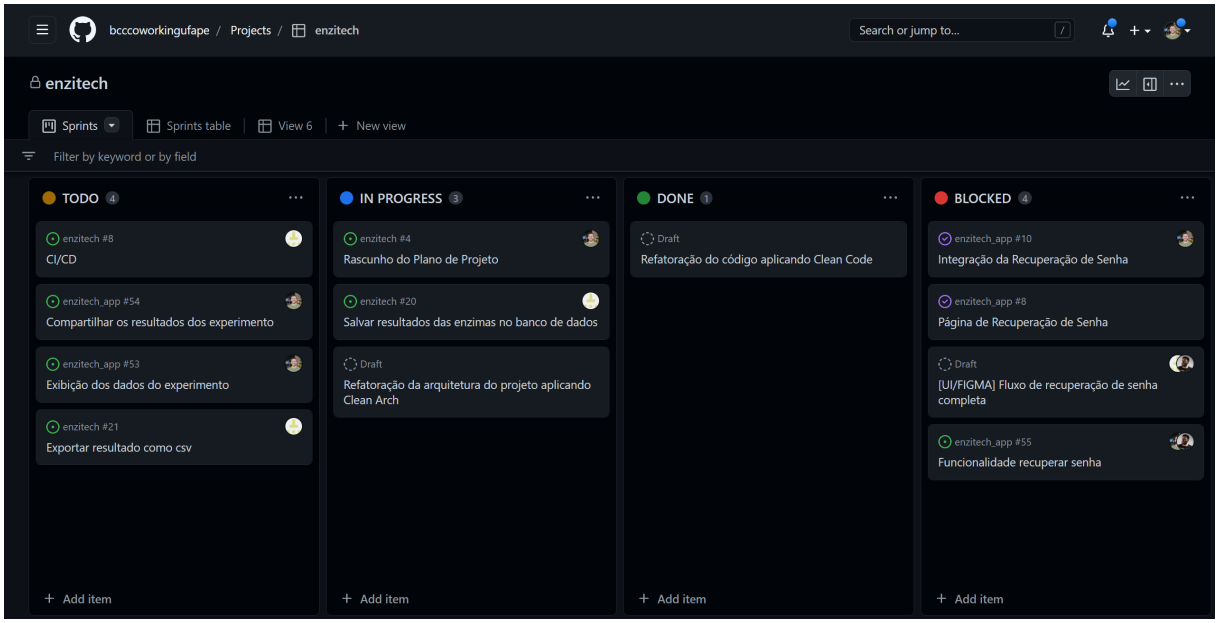

Figura 4.6: Quadro de atividades do Enzitech no Github Project[10]
Fonte: Github

Outro dado interessante para a gestão ágil do projeto é o gráfico de eficiência e bloqueios, algo que também é disponibilizado no Github Project[10], onde é possível ver o progresso do que está sendo feito para a conclusão do projeto, o fluxo de desenvolvimento e as mudanças de escopo ao longo do tempo. Sendo possível identificar gargalos e problemas que impedem o progresso da equipe. Abaixo está um exemplo do gráfico do projeto.

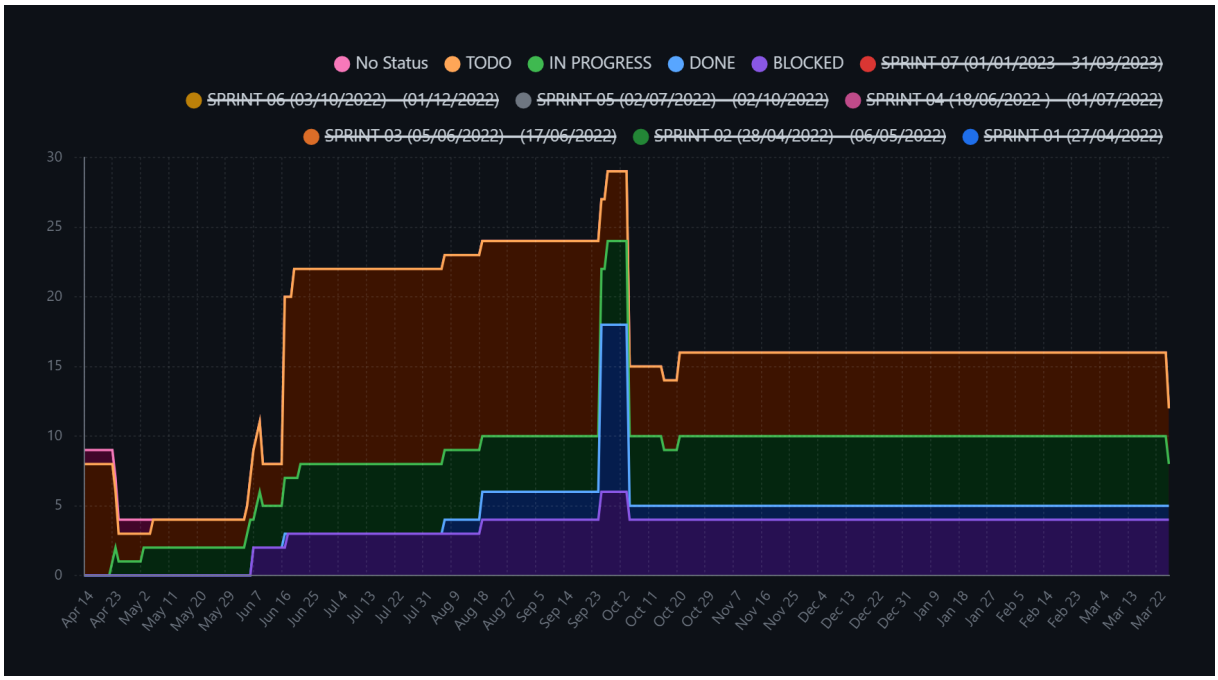

Figura 4.7: Gráfico de *Burn Up* das atividades do Enzitech no Github Project[10]
Fonte: Github

A seguir serão explicadas brevemente o planejamento, desenvolvimento e resultados

obtidos em cada sprint.

4.4.1 Sprint 1

Para a primeira sprint, foi planejado o estudo da última versão do protótipo, o desenvolvimento da identidade visual do APP, incluindo seu ícone/logotipo, o *back-end* tratou de desenvolver o *Create, Read, Update and Delete* (CRUD) de usuários, assim como a documentação da API que seria acessada pelo *front-end(mobile)*, além disso, houve a configuração inicial dos bancos de dados e servidores para os ambientes de teste e desenvolvimento.

Simultaneamente, houve um estudo para decidir qual arquitetura seria adotada no APP, seu sistema de gerenciamento de estado e outros detalhes técnicos, como a criação do arquivo de *design-system* para o APP.

Nesta primeira sprint não houveram muitos impedimentos, e tudo fluiu dentro do esperado, levando em consideração que o projeto estava sendo criado e configurado ainda dentro deste período.

4.4.2 Sprint 2

Nesta segunda sprint foi iniciado o desenvolvimento da tela de *home*, ainda sem integrações, somente para alinhar a estrutura do projeto com a arquitetura escolhida, o *Model–view–viewmodel* (MVVM), um padrão de arquitetura em *software* que facilita a separação do desenvolvimento da interface gráfica do usuário do desenvolvimento da lógica de negócios ou lógica de *back-end* (o *model*), de modo que a *view* não dependa de nenhuma plataforma de modelo específica.

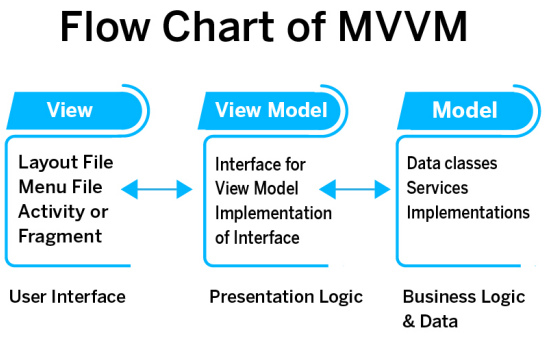

Figura 4.8: Descrição do padrão MVVM
Fonte: APPVENTUREZ (2021)

Além disso, foi desenvolvido a classe para requisições HTTP no APP, como também, mais telas foram desenhadas no protótipo de alta fidelidade. Ao fim, foi possível ter uma primeira versão executável do projeto, o qual já estava apto a se conectar à *Web* e fazer requisições na API que estava sendo desenvolvida em paralelo.

4.4.3 Sprint 3

Os objetivos da terceira sprint foram implementar as telas e realizar as integrações dos *endpoints* desenvolvidos para a API, portanto, como resultado, foram contempladas os seguintes requisitos funcionais:

- RF01 - Cadastro de usuário;
- RF02 - *Login*;
- RF03 - Recuperação de senha;
- RF12 - Listagem de experimentos;

Ademais, o desenvolvimento do protótipo seguiu avançando e correções no código e na estrutura do APP foram realizadas, atingindo a seguinte estrutura:

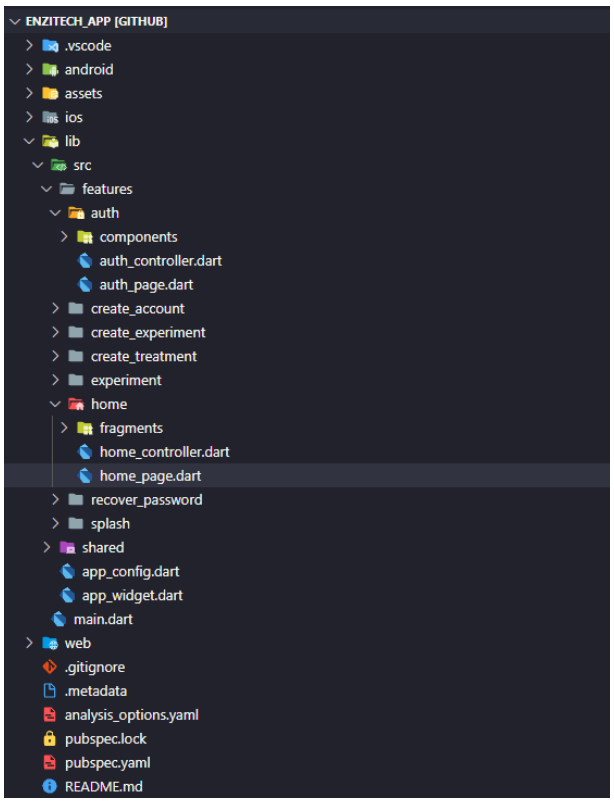

Figura 4.9: Estrutura do projeto ao fim da terceira sprint
Fonte: Autoria própria

Outro ponto importante é que neste momento já foram adicionadas algumas dependências ao projeto que facilitaram o desenvolvimento, entre elas, é válido destacar as seguintes:

- *dio*: Biblioteca utilizada para realizar requisições HTTP;
- *provider*: Biblioteca utilizada para lidar com o gerenciamento de estado do APP;

- *shared_preferences*: Biblioteca utilizada para armazenar dados simples de forma persistente no dispositivo;

Abaixo um trecho da *view* de *Home Page* desenvolvida até este ponto.

Código Fonte 4.1: Home Page

```
// imports suprimidos

class HomePage extends StatefulWidget {
  const HomePage({Key? key}) : super(key: key);

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  late final HomeController controller;
  late final ExperimentsController experimentsController;
  late final TreatmentsController treatmentsController;
  late final AccountController accountController;

  late List<Widget> _fragments;

  @override
  void initState() {
    super.initState();
    controller = context.read<HomeController>();
    experimentsController = context.read<ExperimentsController>();
    treatmentsController = context.read<TreatmentsController>();
    accountController = context.read<AccountController>();
    initFragements();
    if (mounted) {
      Future.delayed(Duration.zero, () async {
        await experimentsController.loadExperiments();
        await treatmentsController.loadTreatments();
        await accountController.loadAccount();
      });
      controller.addListener(() {
        if (controller.state == HomeState.error) {
          ScaffoldMessenger.of(context).showSnackBar(
```

```
            SnackBar(
              content: Text(controller.failure!.message),
            ),
          );
        }
      });
    }
  }

  initFragements() {
    _fragments = [
      ExperimentsPage(
        homeController: controller,
      ),
      TreatmentsPage(
        homeController: controller,
      ),
      AccountPage(
        homeController: controller,
      ),
    ];
  }

  Widget? get dealWithFloatingActionButton {
    if (controller.fragmentIndex == 0) {
      return FloatingActionButton.extended(
        onPressed: () {
          Navigator.pushNamed(
            context,
            RouteGenerator.createExperiment,
          );
        },
        label: Text(
          "Cadastrar\nexperimento",
          style: TextStyles.buttonBackground,
        ),
        icon: const Icon(
          PhosphorIcons.pencilLine,
          color: AppColors.white,
```

```
          size: 30,
        ),
      );
    }

    if (controller.fragmentIndex == 1) {
      return FloatingActionButton.extended(
        onPressed: () {
          Navigator.pushNamed(
            context,
            RouteGenerator.createTreatment,
          );
        },
        label: Text(
          "Cadastrar\ntratamento",
          style: TextStyles.buttonBackground,
        ),
        icon: const Icon(
          PhosphorIcons.pencilLine,
          color: AppColors.white,
          size: 30,
        ),
      );
    }

    return null;
  }

  @override
  Widget build(BuildContext context) {
    final controller = context.watch<HomeController>();

    return Scaffold(
      appBar: AppBar(
        title: SvgPicture.asset(
          'assets/images/logo.svg',
          fit: BoxFit.contain,
          alignment: Alignment.center,
        ),
```

```
      ),
      body: ChangeNotifierProvider(
        create: (BuildContext context) {},
        child: _fragments[controller.fragmentIndex],
      ),
      floatingActionButton: dealWithFloatingActionButton,
      bottomNavigationBar: BottomNavigationBar(
        items: const [
          BottomNavigationBarItem(
            icon: Icon(PhosphorIcons.flask),
            label: 'Experimentos',
          ),
          BottomNavigationBarItem(
            icon: Icon(PhosphorIcons.testTube),
            label: 'Tratamentos',
          ),
          BottomNavigationBarItem(
            icon: Icon(PhosphorIcons.userCircleGear),
            label: 'Conta',
          ),
        ],
        currentIndex: controller.fragmentIndex,
        selectedItemColor: AppColors.white,
        backgroundColor: AppColors.primary,
        onTap: (index) => controller.setFragmentIndex(index),
      ),
    );
  }
}
```

Desta forma, adiantando a estrutura para as próximas *features* a serem implementadas e obtendo como resultado um APP que já tomava o formato desejado, assim, entregando valor com a entrega da sprint.

4.4.4 Sprint 4

Nesta quarta sprint, os objetivos foram as implementações das telas e integrações dos *endpoints* desenvolvidos que restavam, abrangindo os seguintes requisitos funcionais:

- RF04 - Cadastro de tratamento;

- RF06 - Listagem de tratamentos;
- RF07 - Cadastro de enzima;
- RF09 - Listagem de enzimas;
- RF10 - Criação de experimento;

Neste ponto, o APP já possuía as principais *features* para seguir com a parte de experimentos e cálculos, alguns testes foram realizados para verificar o fluxo de informações entre as telas e se tudo estava integrado corretamente, apesar do projeto ser desenvolvido para multiplataformas, vale salientar que inicialmente o APP só estará disponível para o sistema Android, visto que a publicação de APPs neste sistema é muito mais simples e barata se comparada a APPs para iOS, cuja licença para publicar APPs na loja custa um valor considerável. Não é descartada a hipótese de lançar o APP para dispositivos iOS no futuro, contanto que exista demanda dos usuários para tal.

O projeto necessitou de uma configuração no *Firebase* [11] para o lançamento de versões internas para teste no *App Distribution*, o que foi realizado aqui nesta sprint, com isso, foi possível escolher alguns usuários envolvidos no projeto para realizarem os testes, instalando o mesmo em seus dispositivos pessoais.

Abaixo, na Figura 4.10, uma demonstração do painel do *App Distribution* para o APP, para utilizar este serviço foi preciso integrar o Firebase SDK no APP Flutter e configurar o processo de build para enviar o .APK ou .AAB (versões compiladas para o Android) para o Firebase. Em seguida, foi necessário criar um grupo de testadores e conceder acesso ao APP.

Os testadores receberam um convite por e-mail com um link para fazer o download do APP e começar a testá-lo. No painel, é possível controlar quem tem acesso às versões do APP, permitindo que apenas usuários específicos o baixem e o testem, uma vez que ele tenha sido distribuído, é possível monitorar as métricas de uso e coletar feedback dos testadores, o que ajuda a melhorar o produto antes de lançá-lo oficialmente.

[11] *Firebase*: `https://firebase.google.com/`

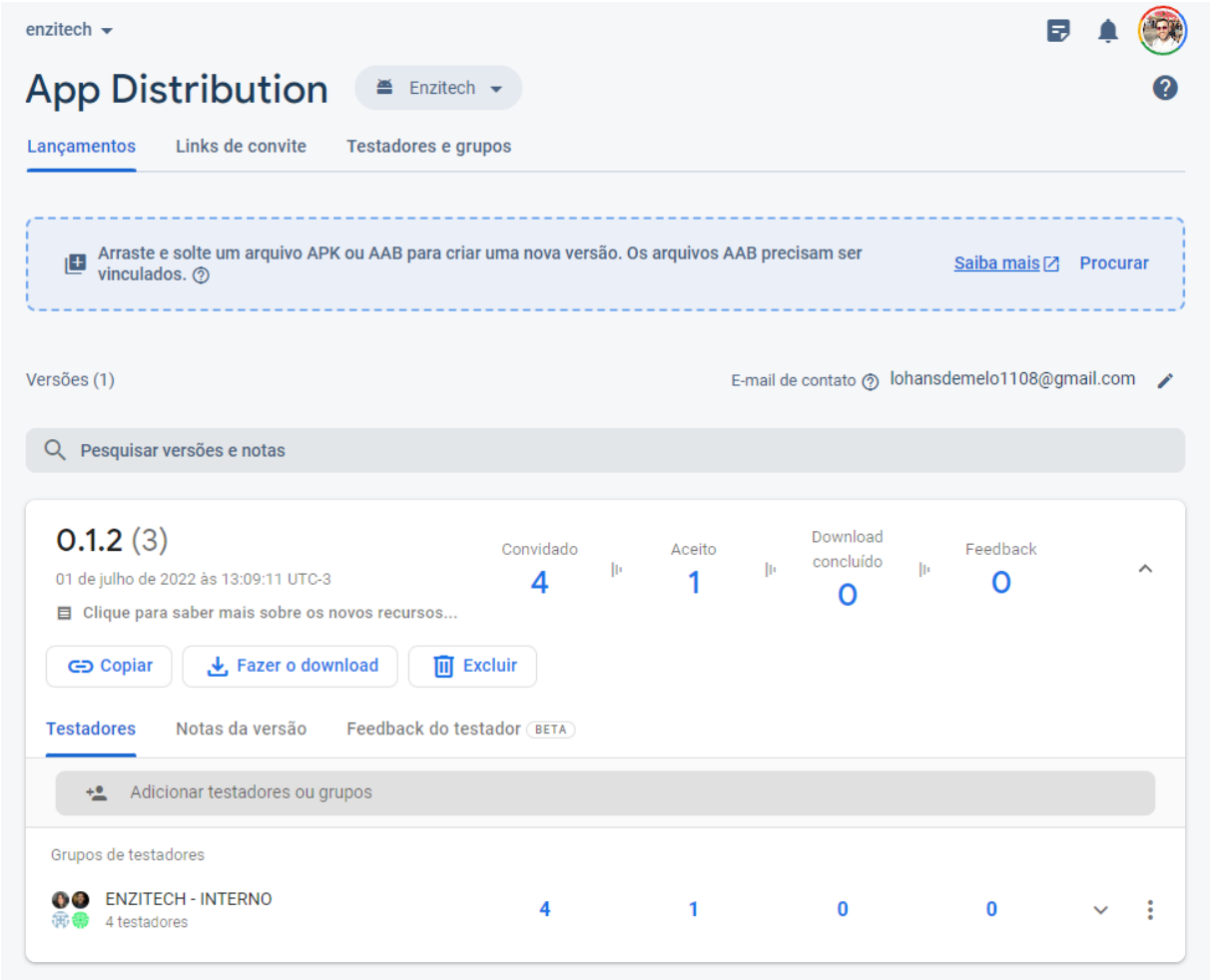

Figura 4.10: Aba do *Firebase App Distribution* do projeto Enzitech
Fonte: *Firebase*[11]

4.4.5 Sprint 5

Na quita sprint, o protótipo de alta fidelidade atingiu a maturidade que o projeto necessitava, desta forma o foco de desenvolvimento foi voltado para as implementações das *features* de deletar as entidades que já eram possíveis serem criadas, desta forma, os seguintes requisitos funcionais foram concluídos:

- RF05 - Deletar um tratamento;
- RF08 - Deletar uma enzima;
- RF11 - Deletar um experimento;
- RF13 - Utilizar filtros na listagem de experimentos;

A RF13 foi iniciada nesta sprint, mas não foi concluída, pois a história foi dividida em duas *spikes* — a definição de *"Spike"* vem do *Extreme Programming* e surgiu a partir da *"Spike Solution"*, que é um programa ou protótipo funcional usado para explorar possíveis

soluções para um problema específico — uma para o desenvolvimento de um *switch* entre experimentos concluídos e em andamento e outra para o desenvolvimento do filtro por ordenação de características de um experimento, portanto, nessa, o *switch* foi implementado, porém sem integrações com a API ainda.

Nesta sprint mais requisitos foram concluídos, houveram vários ajustes e correções de *bugs*, como os de fluxos após criação de um experimento, junto disso houveram alguns estudos para uma futura refatoração que será abordada na última sprint, foi notado que o escopo do projeto precisaria de uma arquitetura mais refinada, logo, este tempo foi reservado para estudo junto ao desenvolvimento.

4.4.6 Sprint 6

Nesta sexta sprint, os seguintes requisitos funcionais foram atacados:

- RF13 - Utilizar filtros na listagem de experimentos;
- RF14 - Visualizar detalhes de um experimento;
- RF15 - Inserção de dados para o cálculo enzimático no experimento;

Na Figura 4.11 é possível ver o *dialog* desenvolvido para a escolha de filtros na listagem de experimentos, nele é possível escolher a ordenação por ordem crescente ou decrescente, assim como, é possível mudar a ordenação de acordo com o nome, descrição, repetições, progresso, data de criação ou data de modificação. Desta forma, finalizando o requisito funcional 13 por completo.

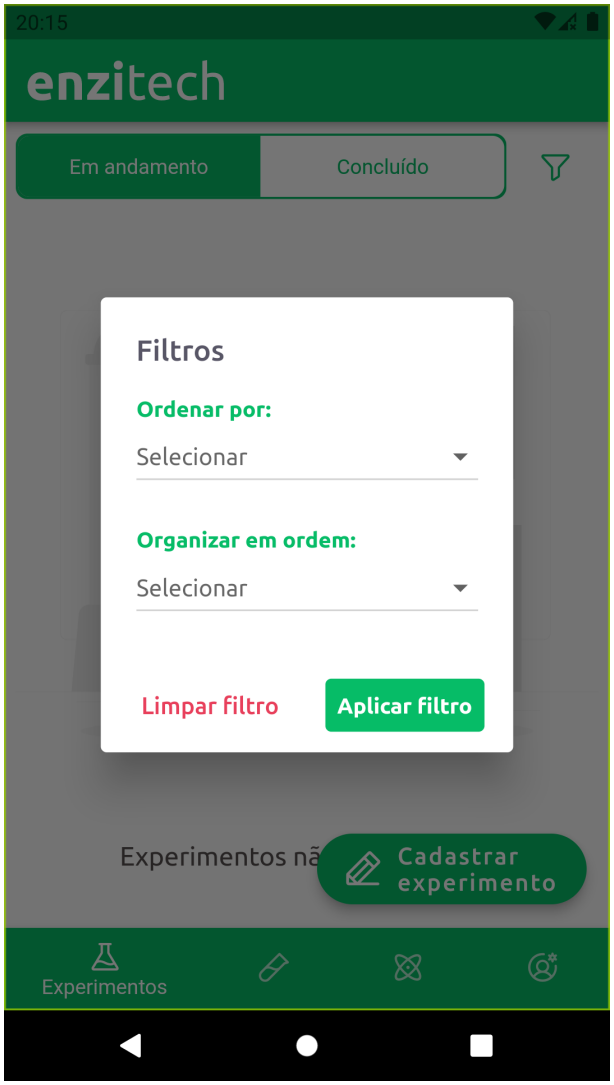

Figura 4.11: Filtros de experimento do Enzitech
Fonte: Autoria própria

Aqui também foi desenvolvido todo o fluxo para visualização dos detalhes de um experimento, o que dá acesso às opções de inserir dados e visualizar os mesmos. Nesta sprint também foi aplicado um estudo de performance feito no APP, o qual até este momento só carregava os dados após a *splashscreen* (tela antes da home/listagem de experimentos), o que não era o melhor cenário possível, então, após alguns artigos lidos e estudos de caso, a solução encontrada foi de fazer a primeira carga de informações do Enzitech ao APP ser aberto, ou seja, em sua inicialização, de forma assíncrona, onde, se preciso fosse, a tela de *splashscreen* faria sua responsabilidade, que é de segurar a navegação do usuário até que todos os dados estejam preparados para o uso.

Desta forma, houve um ganho de performance que diminuiu o tempo de carregamento do APP, trazendo uma sensação de maior fluidez.

Código Fonte 4.2: Trecho de código para realizar a checagem de autenticação e pré-carregamento dos dados

```
_checkAuth() async {
    Future.delayed(Duration.zero).then((_) async {
    String token = await getToken() ?? '';

    if (!mounted) return;

    if (token.isEmpty) {
        Navigator.pushReplacementNamed(context, RouteGenerator.auth);
    } else {
        await Provider.of<HomeViewmodel>(context, listen: false)
        .getContent()
        .then((value) =>
            Navigator.pushReplacementNamed(
                context,
                RouteGenerator.home,
            ),
        );
      }
    });
  }
```

Como podemos ver no Código Fonte 4.2 o sistema primeiro checa se há algum *token* de autenticação guardado, caso negativo ele segue da *splashscreen* de volta para a tela de *login* para o usuário realizar a autenticação novamente, caso positivo a navegação segue, os dados são pré-carregados e logo após o usuário é levado para a tela inicial do Enzitech.

Além disso, foi implementado uma *feature* para lidar com ações sensíveis, como a de exclusão de itens, desta forma, é possível ativar ou desativar nas configurações do APP um alerta de confirmação de exclusão de experimentos, tratamentos e enzimas, o qual, além deste passo de segurança implementado, oferece outra funcionalidade, independente desta anterior estar ativada ou não, que é a de desfazer ações destrutivas, ou seja, o usuário pode se arrepender da exclusão (ou tê-la feito por engano) e recuperar esta informação, assim, oferecendo um passo de segurança a mais para os dados, abaixo na Figura 4.12 é possível ver sua aplicação em um tratamento.

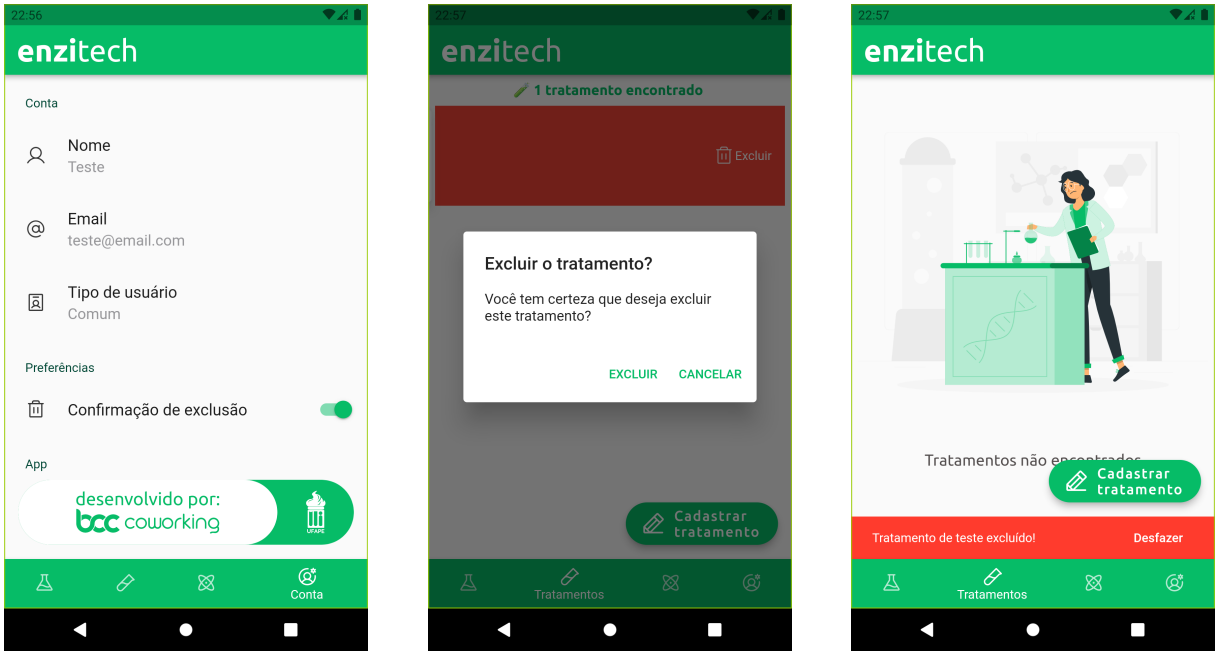

Figura 4.12: Fluxo de exclusão de um tratamento com a opção de confirmação de exclusão ativada

4.4.7 Sprint 7

Na sétima e última sprint do projeto o projeto houve uma enorme refatoração de arquitetura, como comentado na subseção 4.4.4, aqui, além dessa refatoração que não adiciona funcionalidades, mas garante a estabilidade do APP e permite que o mesmo possa receber novas funcionalidades de forma mais fácil e sem problemas, houve a conclusão dos requisitos funcionais restantes, são eles:

- RF16 - Visualização de resultados discrepantes do cálculo enzimático antes de salvar no experimento;
- RF17 - Edição de resultados discrepantes do cálculo enzimático antes de salvar no experimento;
- RF18 - Opção de salvar os resultados do cálculo enzimático no experimento;
- RF19 - Visualização de todos os resultados e o progresso do experimento;
- RF20 - Exportação dos resultados do experimento;

[EM DESENVOLVIMENTO até 31/03/23]

4.5 Back-end

O back-end do projeto foi desenvolvido pelo BCCCoworking, Laboratório de Pesquisa e Desenvolvimento, concebido no curso de Ciência da Computação da UFAPE, sendo assim, nesta

seção trago detalhes resumidos de sua implementação e como ele está ligado ao APP, já que o mesmo faz parte de todo o projeto do sistema Enzitech.

[EM DESENVOLVIMENTO até 31/03/23]

4.6 Banco de Dados

Descrever o Banco de Dados. [EM DESENVOLVIMENTO até 31/03/23]

4.7 Fluxo de informações

Descrever a comunicacao entre backend e app. [EM DESENVOLVIMENTO até 31/03/23]

4.8 Arquitetura do sistema

Descrever o processo de desenho da arquitetura do aplicativo.

4.9 Testes

Descrever o processo de testes da aplicação móvel, incluindo os tipos de testes realizados, como testes funcionais e de desempenho. [EM DESENVOLVIMENTO até 31/03/23]

4.10 Implantação

Descrever o processo de implantação da aplicação móvel, implantação do backend e do BD na ufape, como o app muda com isso, incluindo a distribuição nas lojas de aplicativos, atualizações e manutenção. [EM DESENVOLVIMENTO até 31/03/23]

5

Resultados obtidos

A ideia deste capítulo é trazer uma visão geral sobre todas as funcionalidades desenvolvidas, como o app ficou em sua ultima versão e a opinião dos usuários do sistema (talvez após eles usarem algumas funções e responderem um questionario).

5.1 Apresentação do Enzitech

[EM DESENVOLVIMENTO até 31/03/23]

5.2 Pesquisa de satisfação

[EM DESENVOLVIMENTO até 31/03/23]

6

Conclusão

[EM DESENVOLVIMENTO até 31/03/23]

6.1 Trabalhos futuros

[EM DESENVOLVIMENTO até 31/03/23]

Referências

AHVANOOEY, M. T. et al. A survey on smartphones security: software vulnerabilities, malware, and attacks. **arXiv preprint arXiv:2001.09406**, [S.l.], 01 2020.

ALECRIM, E. **O que é Tecnologia da Informação (TI)?** 2013.

ALLISON, S. D.; JASTROW, J. D.; IVANS, C. E. Measurement of enzymes in soil: a user's guide. **Soil Biology and Biochemistry**, [S.l.], v.42, n.12, p.1928–1935, 2010.

ANGULARMINDS. **Comparison Between Hybrid Vs Native App**. 2022.

APPVENTUREZ. **Introduction to MVVM Architecture in Flutter**. 2021.

BERTOLI, A. et al. Smart Node Networks Orchestration: a new e2e approach for analysis and design for agile 4.0 implementation. **Sensors**, [S.l.], v.21, p.1624, 02 2021.

B'FAR, R. **Mobile computing principles**: designing and developing mobile applications with uml and xml. [S.l.]: Cambridge University Press, 2004.

BISWAL, S.; RATH, S. K.; MOHANTY, S. Mobile application testing: a comprehensive review of techniques and tools. **Journal of Systems and Software**, [S.l.], v.167, p.110556, 2020.

BORGES, J. A análise de usuários de smartphones: android versus ios. **Revista de Tecnologia da Informação**, [S.l.], v.24, n.3, p.12–18, 2017.

DATA.AI. **The State of Mobile 2021**. 2021.

EMBRAPA. **Visão 2030 - O futuro da agricultura brasileira**. 2018.

FAO. **The Future of Food and Agriculture – Alternative Pathways to 2050**. Roma: FAO, 2018.

FIDEL, S. **User-centered design**: an integrated approach. [S.l.]: Pearson Education, 2003.

GLOBALSTATS, S. Mobile Operating System Market Share Brazil| Statcounter Global Stats. **Retrieved June**, [S.l.], v.23, p.2020, 2020.

GUHA, S.; BANERJEE, A. Flutter vs React Native: a comparative study. **Journal of Intelligent Systems**, [S.l.], v.29, n.2, p.255–270, 2020.

INTELLIGENCE, M. **Smartphones market - growth, trends, covid-19 impact, and forecasts (2023 - 2028)**. [S.l.]: Mordor Intelligence, 2023.

LARICCHIA, F. **Number of mobile devices worldwide 2020-2025**. 2022.

LEE, V.; SCHNEIDER, H.; SCHELL, R. **Aplicações móveis**: arquitetura, projeto e desenvolvimento. [S.l.]: Pearson Makron Books, 2005.

LEITE, A. C.; MACEDO, H. COMPARATIVO ENTRE SISTEMAS OPERACIONAIS MÓVEIS–ANDROID X IOS. , [S.l.], 2017.

LÓPEZ-SANTIAGO, J. G. et al. Carbon storage in a silvopastoral system compared to that in a deciduous dry forest in Michoacán, Mexico. **Agroforestry Systems**, [S.l.], v.93, n.1, p.199–211, 2019.

MARCONI, A. P. et al. A Systematic Literature Review of Mobile Application Development: trends, challenges, and opportunities. **IEEE Access**, [S.l.], v.9, p.122949–122965, 2021.

MARTIN, R. C. **Principles of OOD**. 2005.

MARTIN, R. C. **Código Limpo**: habilidades práticas do agile software. [S.l.]: Editora Bookman, 2008.

MARTIN, R. C. **Agile software development**: principles, patterns, and practices. [S.l.]: Prentice Hall Professional, 2008.

MARTIN, R. C. **Arquitetura Limpa**: o guia do artesão para estrutura e design de software. [S.l.]: Alta Books Editora, 2018.

MYERS, G. J.; SMEDEMA, L.; TURBITT, T. **The Art of Software Testing**. [S.l.]: John Wiley & Sons, 2012.

PRESSMAN, R. S. **Engenharia de Software**: uma abordagem profissional. 8.ed. [S.l.]: AMGH, 2016.

SARKER, A.; ALQAHTANI, F.; ALQAHTANI, M. Comparison between hybrid and native app development for mobile platform. **Journal of Information Security and Applications**, [S.l.], v.60, p.102634, 2021.

SINSABAUGH, R. L. et al. Stoichiometry of soil enzyme activity at global scale. **Ecology Letters**, [S.l.], v.11, n.11, p.1252–1264, 2008.

SOMERVILLE, I. **Engenharia de software**. 9.ed. [S.l.]: Pearson, 2015.

SUTHERLAND, J. **SCRUM**: a arte de fazer o dobro de trabalho na metade do tempo. [S.l.]: Leya, 2014.

TABATABAI, M. A.; BREMNER, J. M. Use of p-nitrophenyl phosphate for assay of soil phosphatase activity. **Soil Biology and Biochemistry**, [S.l.], v.1, n.4, p.301–307, 1969.

WARFEL, T. Z. **Prototyping**: a practitioner's guide. [S.l.]: Rosenfeld Media, 2009.

ZHOU, M.; ZHANG, Y.; HUANG, L. Flutter vs. React Native vs. Native: a comparative study. **Sensors**, [S.l.], v.21, n.2, p.354, 2021.

ÁGIL, M. **O que é Kanban?** 2017.