



**UNIVERSIDADE FEDERAL DO
AGRESTE DE PERNAMBUCO**

Armstrong Lohãns de Melo Gomes Quintino

**SISTEMA DE EXPERIMENTOS 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
2023



Universidade Federal do Agreste de Pernambuco
Graduação em Ciência da Computação

Armstrong Lohãns de Melo Gomes Quintino

**SISTEMA DE EXPERIMENTOS 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
2023

Armstrong Lohãns de Melo Gomes Quintino

Sistema de experimentos para análise e cálculo de atividades enzimáticas do solo/
Armstrong Lohãns de Melo Gomes Quintino. – Garanhuns-PE, 2023-

94 p. : il. (algumas color.) ; 30 cm.

Orientador Rodrigo Gusmão de Carvalho Rocha

Trabalho de Graduação – Universidade Federal do Agreste de Pernambuco, 2023.

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 **Sistema de experimentos 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:

Ícaro Lins Leitão da Cunha
Universidade Federal do Agreste de Pernambuco
Membro Interno

Jamilly Alves de Barros
Universidade Federal do Agreste de Pernambuco
Membro Externo

Jean Carlos Teixeira de Araujo
Universidade Federal do Agreste de Pernambuco
Membro Interno

Rodrigo Gusmão de Carvalho Rocha
Universidade Federal do Agreste de Pernambuco
Professor Orientador

Garanhuns-PE
2023

*Eu dedico este trabalho a todos os meus familiares, amigos
e colegas que sempre acreditaram em mim, dedico em
especial ao meu avô paterno, Vovô Zezinho, que não está
mais aqui, e minha avó materna, Vó Augusta, a quem tenho
grande apreço e orgulho de poder compartilhar esta
conquista.*

Agradecimentos

Primeiramente a Deus, por ter me concedido o conhecimento, a saúde e a força de vontade necessários para alcançar até aqui.

Aos meus pais, Ricarte Gomes Quintino e Ivanize Alves de Melo Quintino, que batalharam duro, nunca mediram esforços para minha educação e sempre me ensinaram a lutar para atingir meus objetivos, meus primeiros amigos, que me ensinaram valores que levarei para toda a vida. Essa conquista é nossa, amo vocês!

A minha irmã, Roane Lohaynne, a qual sempre foi um espelho como irmã mais velha, que sempre esteve ao meu lado. Você é uma fonte de inspiração e admiração para mim, e tenho muito orgulho de ser seu irmão. Dedico este trabalho a você com todo o meu amor e gratidão.

A minha namorada, Ana Beatriz, que me apoiou em todas as fases dessa jornada. Agradeço por ter estado ao meu lado em todos os momentos, sempre acreditando e me incentivando a buscar o meu melhor. Pude iniciar e concluir esta graduação com você ao meu lado, e espero que possamos continuar juntos nos próximos ciclos da vida. Eu te amo muito, você é um exemplo de amor e companheirismo!

Aos meus queridos amigos do OBDGUS e amigos agregados, em especial Alessandro, Andrei, Cézar, Elyson, Erik e Morgan, por estarem juntos comigo sempre que precisei. Todos vocês participaram de algo na minha graduação — me apoiaram com todas as dificuldades que encontrei no caminho e nunca desacreditaram — mas acima de tudo, participaram da minha vida, como grandes amigos que levarei comigo para sempre. Que venham mais peladas, acampamentos, brejas e partidas de CS! *Tamo junto!*

Aos meus amigos de turma e do nosso grupo "O Núcleo", formado lá no começo do curso, em especial meus amigos Anderson, André, Antônio, Baltazar, Daniel, Igor, Júnior (CJ), Laisy, Luis, Machado, Saú, Valdir e Victor, que contribuíram com minha formação, e apesar de seguirmos um pouco distantes, sei que posso contar com muitos, obrigado pelos anos de conversas e conhecimentos trocados. Carrego todos vocês no peito.

Aos demais amigos e colegas que fiz na graduação, aqui em especial Matheus Noronha e Weverton Cintra, vocês me ajudaram muito nessa fase final do curso, devo muito a vocês, agradeço demais pelas horas disponibilizadas em meio a tanta coisa, sem a ajuda de vocês isso aqui não seria possível.

A todos os professores que me fizeram ser um ser humano melhor, desde o ensino infantil até aqui, principalmente aos professores que fazem parte da nossa universidade e que me instruíram de forma excelente para os desafios da vida profissional.

E claro, agradeço ao professor orientador, Dr. Rodrigo Gusmão de Carvalho Rocha por todo o apoio ao longo do curso, em especial, nesta reta final, pegando na minha mão e me ajudando a alcançar este feito, me auxiliando com todos os contratemplos enfrentados.

Para finalizar, a todos os meus colegas e amigos, que não mencionei aqui, mas que de alguma forma participaram deste ciclo maravilhoso. Obrigado!

“As circunstâncias do nascimento de alguém são irrelevantes; é o que você faz com seu dom da vida que determina quem você é.”

— MEWTWO

Resumo

Diante da projeção do crescimento da população mundial, a Qualidade do Solo (QS) é um dos fatores fundamentais para o aumento da produtividade agrícola e para a redução da escassez global de alimentos. Nesse contexto, é fundamental entender a QS, suas características físicas, químicas e biológicas, bem como as interações entre esses fatores, permitindo o desenvolvimento de técnicas mais eficientes e sustentáveis de manejo do solo. Diversos métodos são utilizados para avaliar a QS, entre eles, o cálculo de Atividades Enzimáticas (AEs) do solo é uma importante ferramenta para avaliar a qualidade e saúde do solo, uma vez que a AEs está diretamente relacionada com a quantidade e diversidade de microrganismos presentes no solo. Desta forma, é de fundamental importância a disponibilidade de informação precisa e também de sistemas computacionais para cálculo das AEs e para auxílio na tomada de decisão. Dessa maneira, este trabalho propõe um sistema para o gerenciamento de experimentos e resultados das análises das AEs do solo. Esta abordagem foi implementada através de conceitos como métodos ágeis e com arquitetura limpa, contemplando inicialmente o cálculo das enzimas fosfatase ácida e alcalina, a β -glucosidase, a urease e a protease. Esta ferramenta auxilia pesquisadores, laboratórios e profissionais da área, que podem realizar experimentos de solo com embasamento científico de forma prática e precisa, aperfeiçoando as práticas de manejo, identificação de problemas de contaminação, tratamento e remediação.

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

Abstract

Given the projection of world population growth, Soil Quality (SQ) is one of the key factors for increasing agricultural productivity and reducing global food shortages. In this context, it is essential to understand SQ, its physical, chemical and biological characteristics, as well as the interactions between these factors, allowing the development of more efficient and sustainable soil management techniques. Several methods are used to evaluate the SQ, among them, the calculation of Enzymatic Activities (AEs) of the soil is an important tool to evaluate the quality and health of the soil, since the AEs is directly related to the quantity and diversity of microorganisms present in the soil. In this way, the availability of accurate information and also of computational systems for calculating the AEs and to aid in decision-making is of fundamental importance. Thus, this work proposes a system for managing experiments and results of soil AEs analyses. This approach was implemented through concepts such as agile methods and clean architecture, initially contemplating the calculation of acid and alkaline phosphatase enzymes, β -glucosidase, urease and protease. This tool helps researchers, laboratories and professionals in the area, who can carry out scientifically based soil experiments in a practical and precise way, improving management practices, identifying contamination problems, treatment and remediation.

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

Listas de Figuras

2.1	Mercado de Smartphones - crescimento, tendências, impacto da COVID-19 e previsões (2023 - 2028)	24
2.2	Previsão do número de dispositivos móveis em todo o mundo de 2020 a 2025 (em bilhões)	25
2.3	Participação no mercado de Sistemas Operacionais móveis em todo o mundo . .	27
2.4	Participação no mercado de Sistemas Operacionais móveis no Brasil.	27
2.5	Passos do desenvolvimento de Aplicativos (APPs) híbridos e nativos	30
2.6	Esquema de comunicação síncrona da <i>Application Programming Interface</i> (API) <i>Representational State Transfer</i> (REST)	31
2.7	Tendência mundial de popularidade do Flutter (vermelho) e do React Native (azul) (2018–2023).	32
2.8	Visão geral da arquitetura do Flutter	33
2.9	Representação de uma Sprint com o framework Scrum.	36
2.10	Representação de um quadro de desenvolvimento com o framework Kanban. .	38
3.1	Diagrama de Casos de Uso do sistema.	47
3.2	Parte do protótipo final de baixa fidelidade.	48
3.3	Parte do protótipo final de alta fidelidade.	49
3.4	Visão geral do protótipo final de alta fidelidade.	49
3.5	<i>JetBrains Toolbox App</i>	51
4.1	Representação do Processo Ágil de Desenvolvimento de Software para o Enzitech	55
4.2	Quadro de atividades do Enzitech no Github Project ¹¹ ^{47.41}	56
4.3	Gráfico de <i>Burn Up</i> das atividades do Enzitech no Github Project ¹¹ ¹	57
4.4	Descrição do padrão <i>Model–view–viewmodel</i> (MVVM)	58
4.5	Estrutura do projeto ao fim da terceira sprint	59
4.6	Aba do <i>Firebase App Distribution</i> do projeto Enzitech	61
4.7	Filtros de experimento do Enzitech	63
4.8	Fluxo de exclusão de um tratamento com a opção de confirmação de exclusão ativada	64
4.9	Estrutura final do aplicativo em Flutter para o Enzitech	65
4.10	Diagrama de Entidade-Relacionamento do Enzitech	68
4.11	Camadas de uma Arquitetura Limpa	70
4.12	Sugestão da estrutura das camadas de uma Arquitetura Limpa no Flutter	71
4.13	Arquitetura simplificada da infraestrutura do Enzitech	72

5.1	Fluxo de login do Administrador previamente cadastrado	75
5.2	Fluxo de listagem, exclusão e criação de enzimas	76
5.3	Fluxo de listagem, exclusão e criação de tratamentos	77
5.4	Fluxo de criação de um experimento	78
5.5	Fluxo de detalhamento e cálculo enzimático de um experimento	79
5.6	Fluxo de visualização e compartilhamento de resultados de um experimento . . .	81
5.7	Tela de configurações do APP do Enzitech	82

Lista de Tabelas

3.1	Relação de concentração x absorbância da solução	43
3.2	Descrição dos requisitos funcionais do sistema	45
3.3	Casos de Uso do sistema	46

Lista de Acrônimos

AE	Atividade Enzimática	18
AM	Aprendizado de Máquina	23
APP	Aplicativo	18
API	<i>Application Programming Interface</i>	30
BCC	Bacharelado em Ciência da Computação	40
CAGR	Taxa de Crescimento Anual Composta	23
CRUD	<i>Create, Read, Update and Delete</i>	57
DTO	<i>Data Transfer Object</i>	66
DM	Dispositivo Móvel	23
FAO	Organização das Nações Unidas para Agricultura e Alimentação	21
HTTP	<i>Hypertext Transfer Protocol</i>	30
IA	Inteligência Artificial	23
IDE	<i>Integrated Development Environment</i>	52
JSON	<i>JavaScript Object Notation</i>	31
LEMA	Laboratório de Enzimologia e Microbiologia Agrícola/Ambiental	40
MVP	Produto Viável Mínimo	38
MVVM	<i>Model–view–viewmodel</i>	57
NDC	Contribuição Nacionalmente Determinada	22
ONU	Organização das Nações Unidas	21
PC	Computador Pessoal	18
Plano ABC	Plano de Agricultura de Baixa Emissão de Carbono	22
REST	<i>Representational State Transfer</i>	31
PO	<i>Product Owner</i>	44
QS	Qualidade do Solo	21
SDK	<i>Software Development Kit</i>	50
SO	Sistema Operacional	25
UFAPE	Universidade Federal do Agreste de Pernambuco	40
UI	<i>User Interface</i>	57

UML	<i>Unified Modeling Language</i>	46
URL	<i>Uniform Resource Locator</i>	31
XML	<i>Extensible Markup Language</i>	31

Sumário

1	Introdução	18
1.1	Motivação	19
1.2	Objetivo Geral	19
1.3	Objetivos específicos	19
1.4	Organização do texto	20
2	Fundamentação Teórica	21
2.1	Qualidade do solo	21
2.2	Cálculo de atividades enzimáticas do solo	22
2.3	Mobilidade	22
2.4	Dispositivos móveis	23
2.4.1	iOS	26
2.4.2	Android	27
2.5	Aplicativos móveis	28
2.5.1	Desenvolvimento de aplicativos	28
2.5.1.1	Desenvolvimento nativo	29
2.5.1.2	Desenvolvimento híbrido	29
2.6	APIs <i>Web</i>	30
2.7	Flutter	32
2.8	Código Limpo	34
2.9	Arquitetura Limpa	35
2.10	Desenvolvimento de sistemas	35
2.10.1	Metodologias ágeis	36
2.10.1.1	Scrum	36
2.10.1.2	Kanban	37
2.11	Prototipagem	38
2.11.1	Figma	38
3	Métodos Utilizados	39
3.1	Análise de Viabilidade	39
3.2	Definições do Sistema: Abordagens, Requisitos e Recursos	39
3.2.1	Laboratório BCC Coworking	40
3.2.2	Explorando as etapas e protocolos envolvidos nas AEs do solo	40
3.2.2.1	Reagentes e soluções	41
3.2.2.2	Procedimento	42

3.2.2.3	Curva de calibração	42
3.2.2.4	Cálculo para determinação da atividade	42
3.3	Especificação de requisitos	44
3.3.1	Requisitos funcionais	45
3.3.2	Casos de uso	46
3.4	Protótipo	48
3.5	Tecnologias e ferramentas envolvidas	49
3.5.1	Instalação do Flutter	50
3.5.2	Instalação do Android Studio	50
3.5.3	Instalação do VSCode	52
3.5.4	Ferramentas e bibliotecas de suporte	52
3.5.4.1	Flutter DevTools	52
3.5.4.2	Postman	53
3.5.4.3	<i>Packages</i> do pub.dev	53
4	Proposta e Desenvolvimento da Aplicação	54
4.1	Processo de desenvolvimento de software	54
4.2	Sprints	55
4.3	Back-end	66
4.4	Modelagem dos dados	66
4.4.1	Diagrama de Entidade-Relacionamento	67
4.5	Arquitetura do sistema	69
4.6	Testes	73
4.7	Implantação	73
5	Análise dos resultados	74
5.1	Apresentação do Enzitech	74
6	Conclusão	83
6.1	Principais contribuições	84
6.2	Principais limitações	84
6.3	Trabalhos Futuros	85
Referências		86
Apêndice		88
A Códigos Flutter		89
A.1	Código da tela <i>Home Page</i> desenvolvida na Terceira Sprint	89
A.2	Trecho de código para realizar a checagem de autenticação e pré-carregamento dos dados desenvolvido na Sprint 6	93

B Testes	94
B.1 Trecho de código do teste de unidade para enzima	94

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 outra esfera, a análise de Atividades Enzimáticas (AEs) do solo é um importante indicador da qualidade e saúde do solo, pois essas atividades estão diretamente relacionadas com a disponibilidade de nutrientes e a capacidade do solo em sustentar a vida vegetal. Porém, as técnicas convencionais para a medição dessas atividades são trabalhosas, caras e requerem equipamentos sofisticados.

Segundo [NASCIMENTO et al. \(2017\)](#), o uso de tecnologias nas instituições de pesquisa científica em agronomia tem sido amplamente adotado nas últimas décadas, com o objetivo de aumentar a eficiência e a precisão dos experimentos, bem como de facilitar a coleta e análise de dados. A implementação de sistemas informatizados e a utilização de equipamentos avançados de análise, como espectrômetros e microscópios eletrônicos, são exemplos dessas tecnologias que vêm sendo empregadas em estudos agronômicos.

Nesse contexto, o desenvolvimento de um aplicativo móvel para a realização de experimentos de análise e cálculo de AEs do solo pode ser uma solução eficaz e prática para essa demanda. O objetivo deste trabalho é apresentar o desenvolvimento de um aplicativo móvel que permite a realização destes experimentos, com o intuito de facilitar e tornar mais acessível a

análise dessas atividades, buscando harmonizar o crescimento econômico, a equidade social e a preservação ambiental, visando à conservação do meio ambiente e à garantia do bem-estar humano a longo prazo.

1.1 Motivação

Os experimentos do solo com cálculo de AEs são uma técnica utilizada em estudos de ciência do solo para avaliar a saúde e qualidade do solo, estes experimentos apresentam alguns desafios e lacunas, como a dificuldade na padronização dos métodos de análise, a falta de informações sobre a relação entre as AEs e a fertilidade do solo e a dificuldade em avaliar a dinâmica temporal das AEs. Além disso, há pouco conhecimento sobre as AEs em diferentes tipos de solos e em diferentes regiões do mundo, e a identificação de microrganismos e enzimas específicas pode ser desafiadora. Esses desafios podem afetar a interpretação dos resultados dos experimentos e limitar a aplicação das técnicas em diferentes contextos [TABATABAI \(1994\)](#).

Este tema é importante para a comunidade acadêmica e para a sociedade em geral, pois permite avaliar a saúde e qualidade do solo, identificar microrganismos e enzimas benéficos para o solo e para as plantas, promovendo práticas agrícolas mais sustentáveis e eficientes, além de monitorar a qualidade do solo em áreas urbanas e industriais. A pesquisa e desenvolvimento do sistema proposto para essa área contribui para o desenvolvimento sustentável e para a conservação do meio ambiente.

1.2 Objetivo Geral

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.

1.3 Objetivos específicos

Com base no objetivo geral, correspondem os objetivos específicos indicados a seguir:

- Implementar um sistema, acessível via aplicativo móvel, que permite criar e gerenciar experimentos que realizem cálculos das AEs do solo;
- Gerar resultados e planilhas de forma automática;
- Possibilitar a análise dos resultados;
- Proporcionar uma experiência fluida de navegação no aplicativo, ao utilizar boas práticas de desenvolvimento de *software*.

1.4 Organização do texto

O presente trabalho foi assim constituído:

- O Capítulo 2 discorre sobre conceitos básicos dos principais fundamentos adotados neste trabalho e relacionados ao tema científico, em que são abordado os temas de AEs do solo, dispositivos móveis, desenvolvimento de aplicativos móveis e arquitetura limpa;
- No Capítulo 3 é realizada uma apresentação dos métodos utilizados no desenvolvimento da solução, bem como os requisitos do sistema, tecnologias e ferramentas envolvidas;
- No Capítulo 4, são apresentados os processos do desenvolvimento, arquiteturas utilizadas, informações sobre os testes e implantação do aplicativo e do sistema em si;
- No Capítulo 5, é apresentado a versão final do aplicativo desenvolvido, o fluxo de navegação e como todo o sistema se comporta;
- No Capítulo 6, são apresentadas as considerações finais, principais contribuições, dificuldades encontradas e trabalhos futuros.

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 Unidas (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

¹ONU - 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 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.

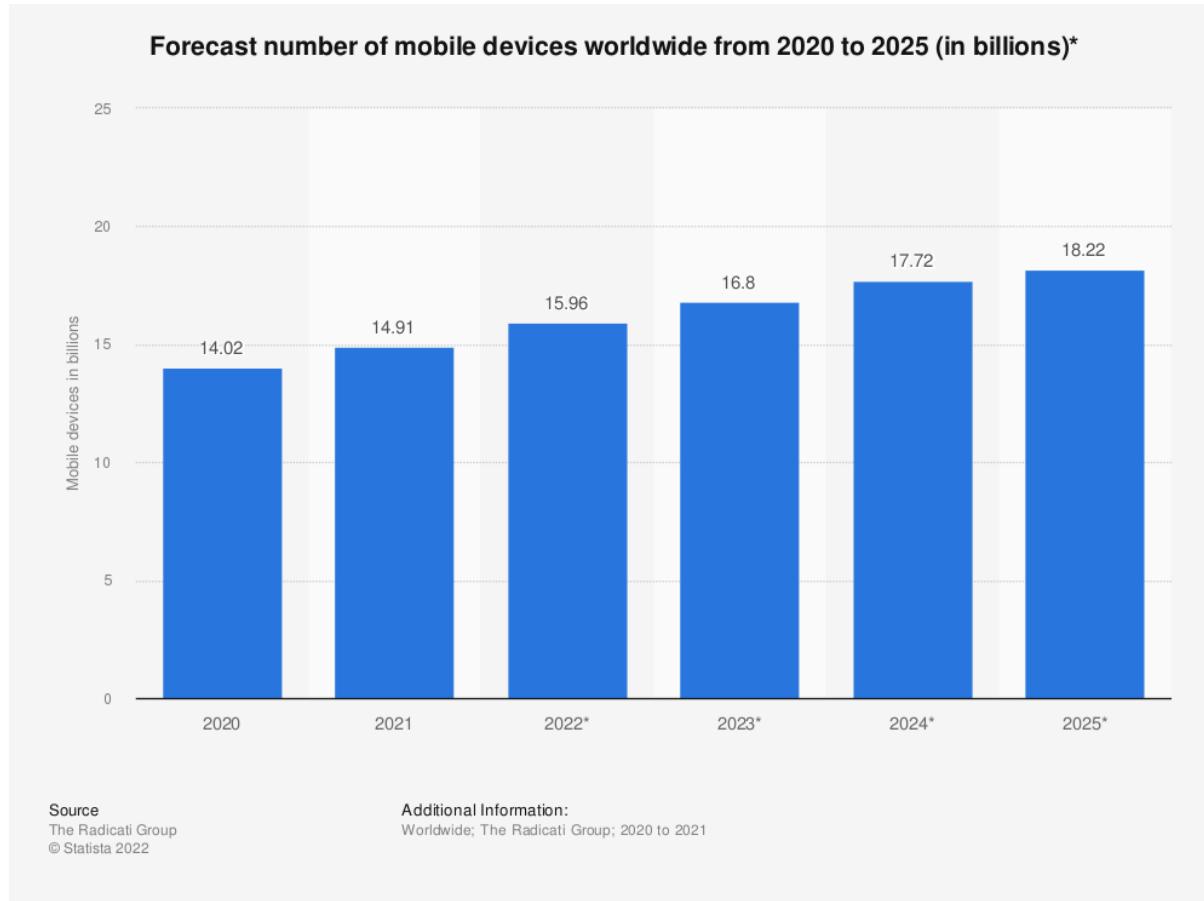


Figura 2.2: Previsão do número de dispositivos móveis em todo o mundo de 2020 a 2025 (em bilhões)

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 $\frac{1}{3}$ 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.

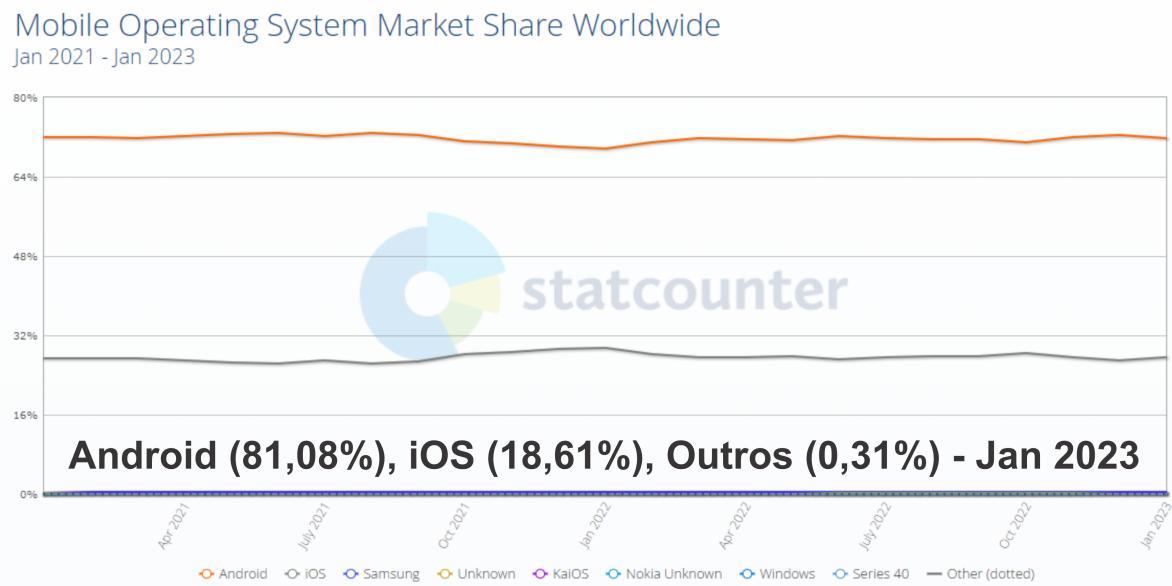


Figura 2.3: Participação no mercado de Sistemas Operacionais móveis em todo o mundo

Fonte: [GLOBALSTATS \(2020\)](#)

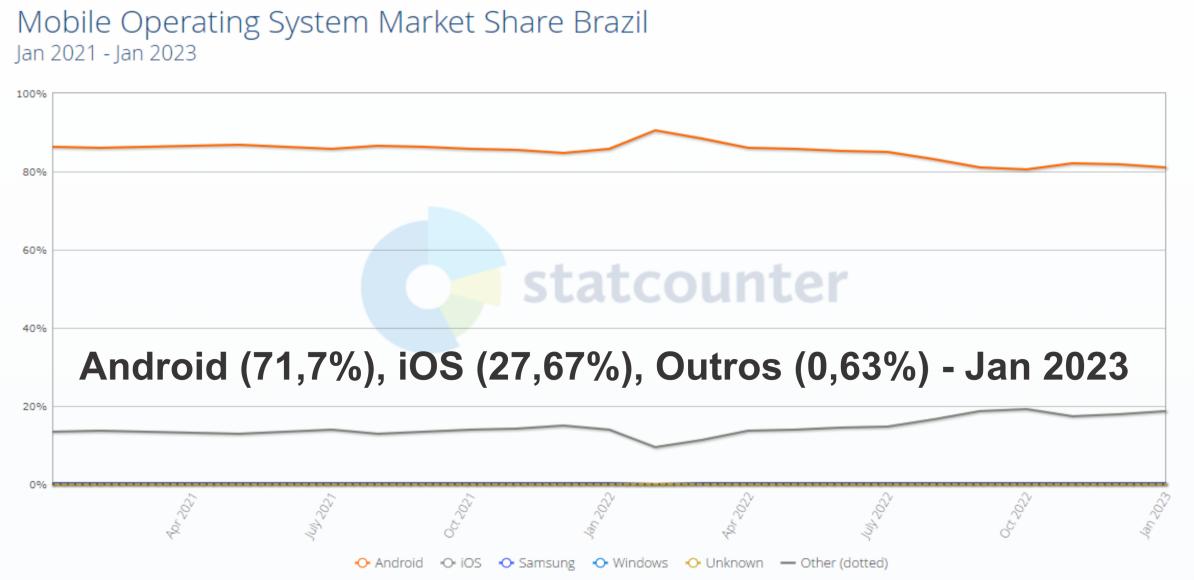


Figura 2.4: Participação no mercado de Sistemas Operacionais móveis no Brasil.

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*.

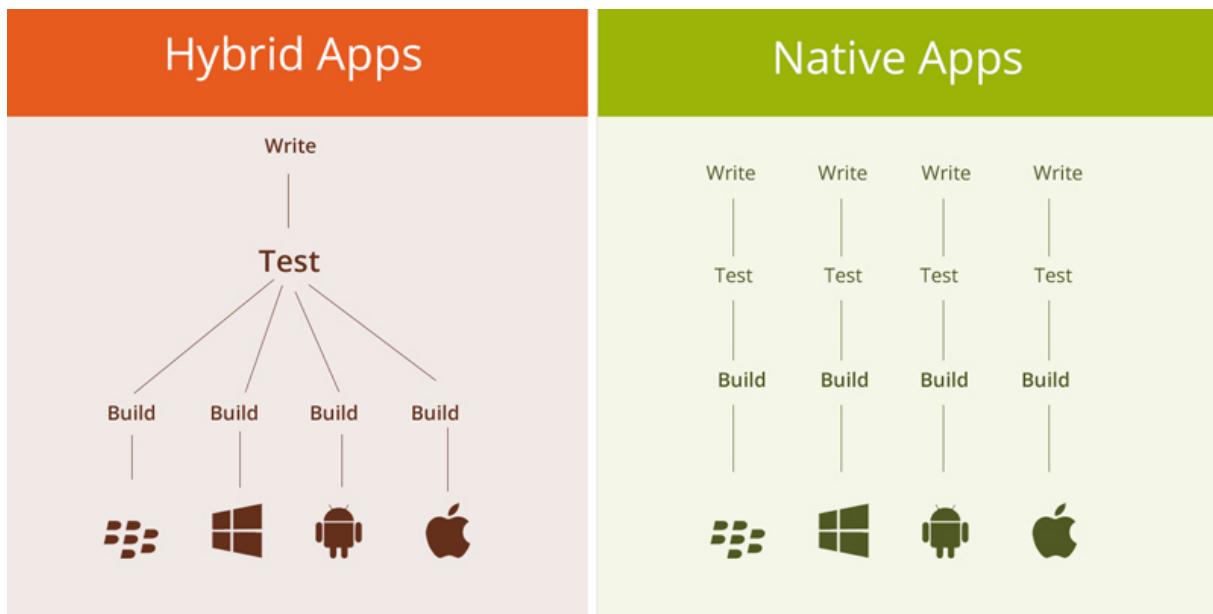


Figura 2.5: Passos do desenvolvimento de APPs híbridos e nativos

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 destchinhado 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 Locators* (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.

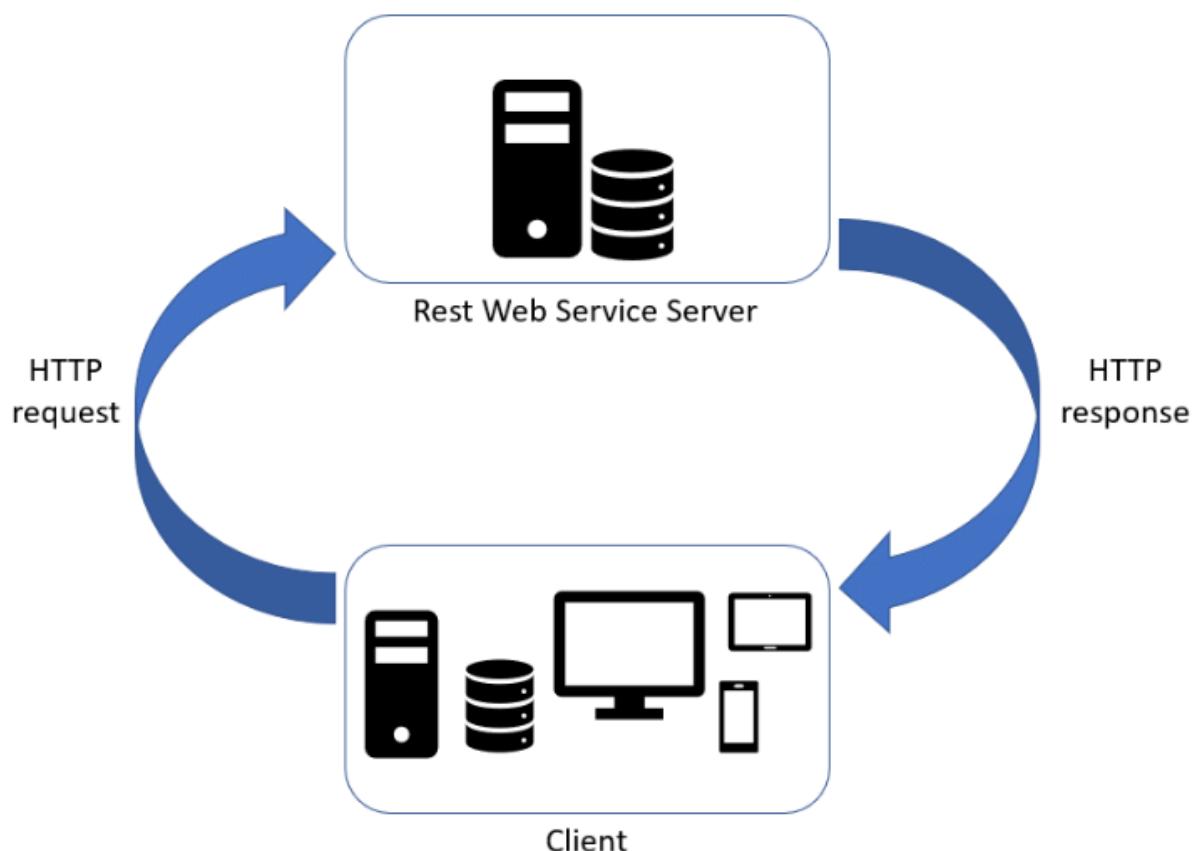


Figura 2.6: Esquema de comunicação síncrona da API REST

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 Google² corroboram o crescimento do Flutter em relação ao seu principal concorrente, como mostrado na figura abaixo.

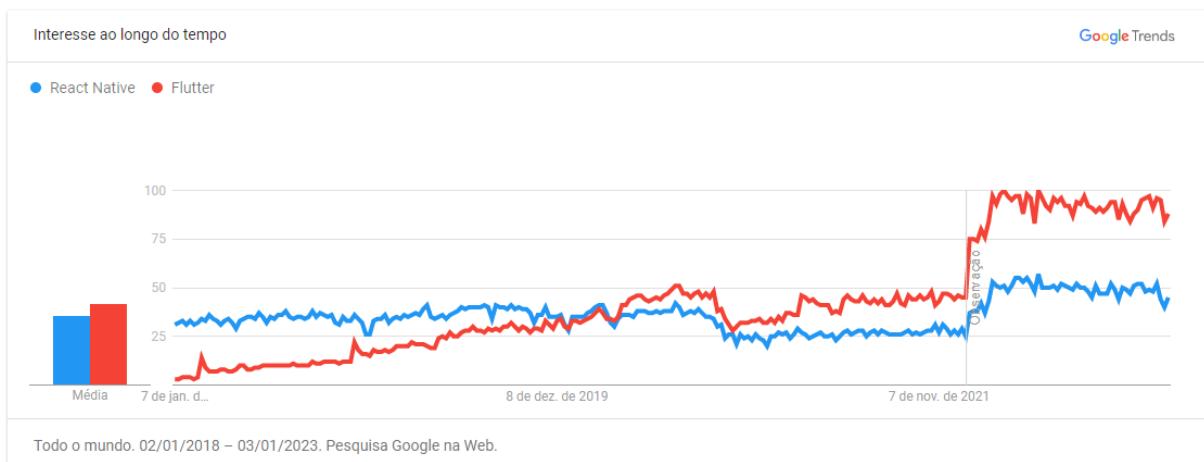


Figura 2.7: Tendência mundial de popularidade do Flutter (vermelho) e do React Native (azul) (2018–2023).

Fonte: Google Trends²

²Google 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 Flutter³, 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.

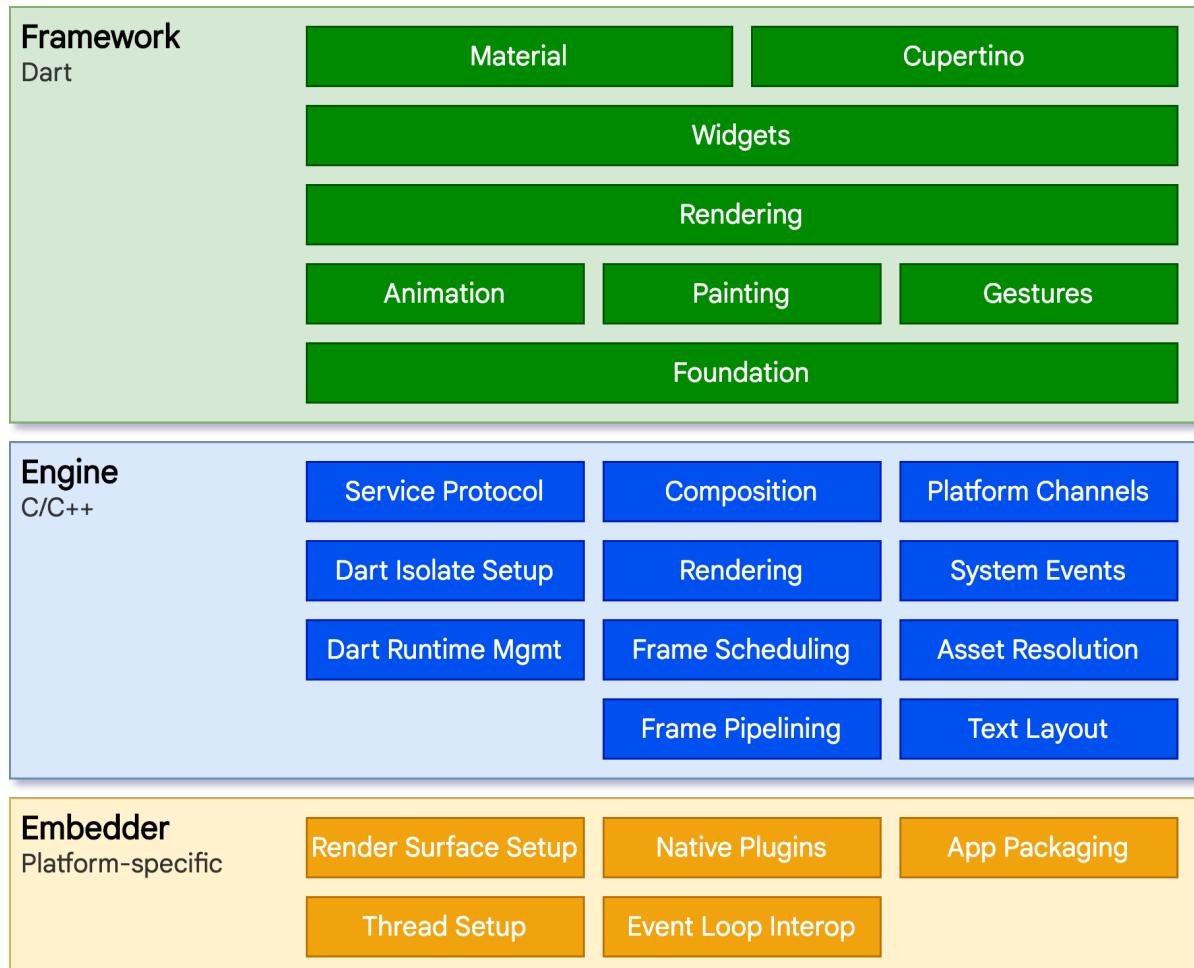


Figura 2.8: Visão geral da arquitetura do Flutter

Fonte: Flutter³

De forma resumida, a camada do *framework* é moderna e reativa, escrita 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

³Visão geral da arquitetura do Flutter: <https://docs.flutter.dev/resources/architectural-overview>

muito poderoso, que, segundo o site oficial do Flutter⁴ 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 frequênciа, os programadores de hoje desconhecem os princípios que são a base das disciplinas das quais suas linguagens foram derivadas. [...] Esse principio 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.

⁴Flutter: <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 isoladasumas 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.

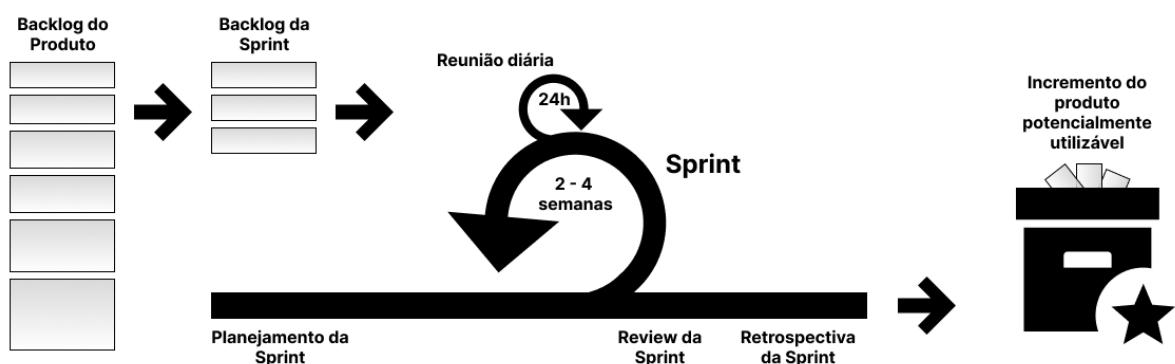


Figura 2.9: Representação de uma Sprint com o framework Scrum.

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 utilizado 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.

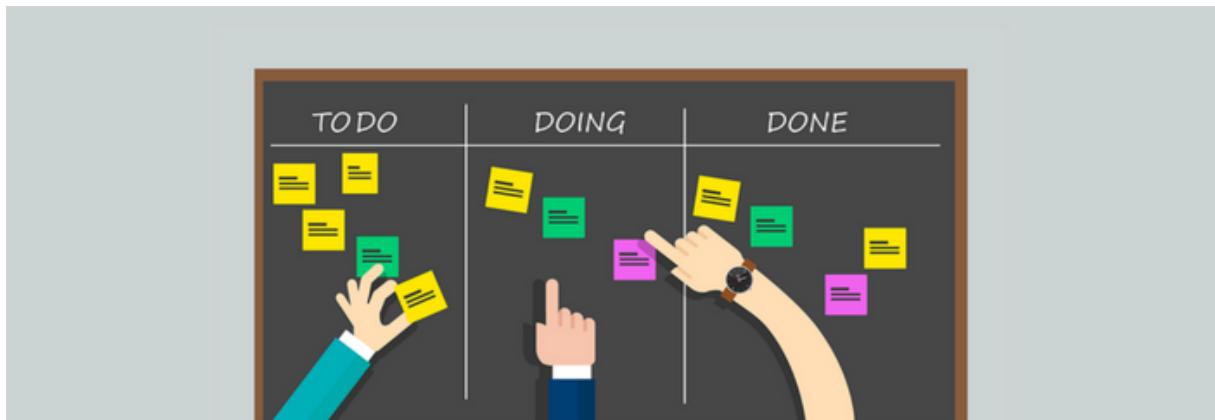


Figura 2.10: Representação de um quadro de desenvolvimento com o framework Kanban.

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

Métodos Utilizados

Este capítulo descreve a metodologia aplicada no desenvolvimento do trabalho, que tem como objetivo uma solução para dispositivos móveis para a realização de experimentos das atividades enzimáticas do solo. Para realizar o trabalho foram utilizados os conceitos de engenharia de requisitos como parte inicial do projeto, a modelagem do sistema por meio de requisitos funcionais e casos de uso, e a definição dos processos de desenvolvimento do software que serão apresentados nas próximas seções.

3.1 Análise de Viabilidade

Nesta fase foi feita uma estimativa das diversas maneiras de suprir as necessidades do cliente, identificando tanto a parte tecnológica como a parte de técnica. Foi verificado o retorno acadêmico da ferramenta, os avanços e melhorias no processo de experimentação do solo que seriam possíveis obter com a mesma.

3.2 Definições do Sistema: Abordagens, Requisitos e Recursos

As AE são as informações chave das funcionalidades do sistema que foi desenvolvido, desta forma, é importante entender nesta etapa suas características, aplicações e importância, portanto, é relevante salientar que o estudo das AEs do solo tem ganhado cada vez mais notoriedade devido à sua relação com a QS e com a produtividade das culturas agrícolas. O cálculo dessas atividades torna-se fundamental para a compreensão dos processos bioquímicos que ocorrem no solo e para a avaliação do impacto das práticas de manejo no solo. No entanto, o cálculo das AEs do solo é um processo complexo e demorado, que envolve a utilização de várias técnicas e equipamentos específicos.

Nesse contexto, o desenvolvimento deste sistema para o cálculo das AEs do solo torna-se uma solução promissora para aumentar a eficiência e a precisão desse processo. O aplicativo tem este objetivo, permitir o cálculo de forma automatizada das atividades de diversas enzimas do solo, como fosfatase ácida, β -glucosidase e urease.

3.2.1 Laboratório BCC Coworking

O projeto teve início na disciplina de Desenvolvimento Distribuído de Software, no curso de Bacharelado em Ciência da Computação (BCC) da Universidade Federal do Agreste de Pernambuco (UFAPE), o qual, logo em seguida, teve seu escopo de desenvolvimento integrado com o Laboratório BCC Coworking, da mesma instituição, este que é um Laboratório de Pesquisa e Desenvolvimento projetado por docentes deste curso, o propósito deste espaço é fornecer um ambiente apropriado para o desenvolvimento de projetos reais, com a supervisão de profissionais da área para garantir o aprimoramento do conhecimento e experiência técnica. Este espaço é destinado aos estudantes que buscam autonomia, estímulo e desenvolvimento de projetos, visando melhorar sua produtividade e expandir seu conhecimento prático.

A proposta do Enzitech foi concebida através de uma observação de melhoria dentre os docentes e discentes de agronomia, ao realizarem as atividades de campo, onde dados eram coletados em papéis e depois passados para uma planilha para o cálculo e coleta de resultados, assim, o Laboratório de Enzimologia e Microbiologia Agrícola/Ambiental (LEMA) da UFAPE identificou a necessidade de desenvolver uma solução que pudesse aprimorar seus processos, portanto, recorreu à expertise do Laboratório BCC Coworking, reconhecido por suas soluções inovadoras e de qualidade, para que juntos pudessem desenvolver uma solução que atendesse a essas demandas específicas.

A equipe foi composta por Armstrong Lohãns, Matheus Noronha, Weverton Cintra, José Vieira, Eduarda Interaminense, as professoras Erika Valente e Jamilly Barros — do curso de Agronomia — que respectivamente propuseram o sistema. A equipe do Enzitech realizou várias reuniões para obter um conjunto inicial de requisitos e elaborar uma breve apresentação da ideia do sistema, além de discutir como o sistema poderia melhorar os processos na área da Agronomia.

3.2.2 Explorando as etapas e protocolos envolvidos nas AEs do solo

Consequentemente, uma análise das etapas sobre os materiais usados, procedimentos de coleta, transporte, conservação e os protocolos que regem todo o processo das AEs do solo foi realizada, explicada e compreendida pela equipe, afim de facilitar a criação das regras de negócio que regem o funcionamento do sistema, ou seja, as regras que definem as principais operações e processos que ocorrem dentro do sistema para atender às necessidades de negócio, de forma clara e precisa.

Em vista disso, o procedimento de coleta de amostras de solo para análise da atividade enzimática é semelhante ao utilizado para avaliação das propriedades químicas do solo. A coleta deve ser realizada preferencialmente na camada de 0 a 10 cm de profundidade do solo, que é a camada diagnóstica para enzimas do solo, ou de 20 a 40 cm, dependendo do tipo de solo, da cultura e finalidade do estudo, após o período chuvoso e coincidindo com a amostragem para análise química. É importante que a amostragem seja representativa da área avaliada, realizando

múltiplas subamostras em áreas de cultivos anuais. Para evitar contaminação e perda de atividade biológica, é recomendado utilizar instrumentos esterilizados ou desinfestados, acondicionar as amostras em sacos plásticos ou tubos de polietileno bem fechados, identificá-las com informações sobre a área, a profundidade, a data e o responsável pela coleta, secá-las ao ar e peneirá-las.

A análise da atividade enzimática deve ser realizada o mais breve possível após a coleta, preferencialmente dentro de 24 horas, utilizando métodos colorimétricos padronizados para medir a atividade de enzimas como arilsulfatase, β -glucosidase, fosfatase ácida e alcalina, urease e outras.

O solo coletado para análises enzimáticas deve ser seco e peneirado, com malha de 2mm. No entanto, para algumas enzimas, é necessário refrigerar as amostras imediatamente após a coleta, a fim de evitar perdas em quantidade de enzimas. A refrigeração das amostras de solo é uma prática recomendada para preservar a atividade enzimática do solo e evitar alterações causadas por fatores como temperatura, umidade e contaminação. No entanto, a refrigeração pode afetar negativamente algumas enzimas do solo, como a fosfatase.

A determinação da atividade de enzimas no solo é um desafio para os pesquisadores, devido à heterogeneidade espacial e temporal do solo e à influência das condições ambientais na expressão e estabilidade das enzimas. Os métodos analíticos devem ser padronizados e validados para cada tipo de solo e de enzima, e os resultados devem ser interpretados com cautela e integrados com outros indicadores biológicos do solo.

A seguir, a demonstração das etapas do protocolo para a Fosfatase. O princípio da atividade das fosfomonoesterases, estando inclusas nesse grupo as fosfatases ácida e alcalina, versa determinar o teor de p-nitrofenol liberado após a incubação do solo com o substrato específico da enzima (p-nitrofenilfosfato).

3.2.2.1 Reagentes e soluções

- **Modified universal buffer (MUB estoque):** dissolver 12,1 g de THAM (Tris), 11,6 g de ácido maleico, 14,0 g de ácido cítrico, 6,3 g de ácido bórico em 500 mL de NaOH 1 mol/L e completar para 1.000 mL com água destilada em balão volumétrico. (Estocar em refrigerador);
- **Modified universal buffer (MUB pH 6,5 fosfatase ácida ou pH 11 fosfatase alcalina):** pegar 200 ml da solução de MUB estoque e ajustar o pH desejado (pH 6,5 ou pH 11) com HCl 0,1 mol/L ou NaOH 1 mol/L, respectivamente, e completar para 500 mL em balão volumétrico com água destilada. Obs.: Preparar o volume desta solução de acordo com a quantidade de amostras realizadas;
- **Solução de p-nitrofenilfosfato de sódio PNP 0,05 mol/L (Sigma):** dissolver 0,06 mL de p-nitrofenilfosfato substrato líquido em 10 mL de MUB pH desejado (pH 6,5 ou pH 11). Preparar o volume desta solução de acordo com a quantidade de amostras realizadas;

- **Solução padrão de p-nitrofenol (Sigma):** dissolver 1 g de p-nitrofenol em 700 mL e completar para 1.000 mL com H₂O destilada. (Estocar em refrigerador);
- **Solução padrão diluída de p-nitrofenol (10 mg/L):** diluir 1 mL da solução padrão de p-nitrofenol em 100 mL de água destilada;
- Solução de CaCl₂ 0,5 mol/L;
- Solução de NaOH 0,5 mol/L;
- Tolueno p.a. (esse reagente pode ser omitido quando o tempo de incubação for de 1 a 2 horas);

3.2.2.2 Procedimento

- Pesar 1 g de solo (passado em peneira de 2 mm) e colocar em tubo de ensaio (3 repetições e 2 brancos);
- Pipetar: 0,25 mL de tolueno; 4 mL de MUB (pH 6,5 ou pH 11); 1 mL de solução de substrato PNP 0,05 mol/L (nos brancos a solução de PNP só é adicionada após a incubação), agitar em vórtex, vedar os tubos com papel filme e incubar em banho-maria a 37 °C por 1 hora;
- Após incubação adicionar 1 mL de PNP (apenas nos brancos), 1 mL de CaCl₂ 0,5 mol/L e 4 mL de NaOH 0,5 mol/L. Agitar em vórtex e filtrar em papel de filtro com auxílio de um funil em tubos de ensaio. Proceder à leitura da atividade em espectrofotômetro em absorbância no comprimento de onda de 400 nm;

3.2.2.3 Curva de calibração

Em seis tubos de ensaio pipetar alíquotas de 0, 1, 2, 3, 4 e 5 mL da solução padrão diluída de p-nitrofenol (10 mg/L), ajustar o volume para 5 mL com água destilada para cada ponto e proceder pipetando 1 mL de CaCl₂ 0,5 mol/L e 4 mL de NaOH 0,5 mol/L. Em seguida agitar em vórtex, vedar os tubos com papel filme e incubar a 37 °C por 1 hora. Proceder à leitura em espectrofotômetro à 400 nm. Obs.: O sobrenadante deve ser colocado em cubeta sem misturar com o precipitado. (Fazer a curva em triplicata).

3.2.2.4 Cálculo para determinação da atividade

Utilizando alguns valores de exemplo, o cálculo é realizado da seguinte forma:

- **Etapa 1 - Calibração com os valores da curva:**

Concentração de p-nitrofenol $\mu\text{g/mL}$	Absorbância
0	0
1	0,102
2	0,149
3	0,223
4	0,312
5	0,382

Tabela 3.1: Relação de concentração x absorbância da solução

O modelo da equação da reta é $y = ax + b$ onde Y é a absorbância, X é a concentração de p-nitrofenol, a e b são parâmetros do ajuste do modelo. No exemplo, $a = 0,0747$ e $b = 0,008$.

Resolvendo a equação para X (concentração de p-nitrofenol):

$$x = \frac{Y - b}{a}$$

Para calcular a atividade da Fosfatase, usando o exemplo da curva de calibração:

$$p-nitrofenol(\mu\text{g/mL}) = \frac{Y - 0,008}{0,0747}$$

O valor de absorbância da leitura da amostra (Y) para uso na equação (2) é a média das leituras de absorbância das duas repetições da amostra de solo, subtraída da absorbância da prova em branco. Considerando, como exemplo, a leitura de absorbância de duas repetições de uma amostra de solo e seu branco:

Leitura da repetição 1: 0,181

Leitura da repetição 2: 0,192

Leitura da prova em Branco: 0,02

$$Y = (0,181 + 0,192) / 2 - 0,02$$

$$Y = 0,1865 - 0,02$$

$$Y = 0,1665$$

Substituindo o Y na equação:

$$p-nitrofenol(\mu\text{g/mL}) = \frac{0,1665 - 0,008}{0,0747}$$

$$p-nitrofenol(\mu\text{g/mL}) = 2,122$$

- **Etapa 2** - Para expressar a atividade da enzima em massa de solo, a partir da curva padrão:

$$p-nitrofenol(\mu\text{g g}^{-1}\text{h}^{-1}) = \frac{CxV}{gxt}$$

Onde, **C** = concentração de p-nitrofenol na suspensão de solo, em

μ

g/mL; **V**= volume das soluções pipetadas (10,25 mL); **g** = peso do solo seco ao ar (g) e **t** = tempo de incubação (horas).

Ou C x 10,25. Na equação completa:

$$p-nitrofenol(\mu\text{g g}^{-1}\text{h}^{-1}) = \left(\frac{2,122}{mL} \right) x \left(\frac{10,25mL}{1gx1h} \right) = 21,75\mu\text{g g}^{-1}\text{h}^{-1}$$

- Vale salientar que esse cálculo é utilizado para as enzimas fosfatases, arilsulfatase e betaglucosidase.

3.3 Especificação de requisitos

A engenharia de requisitos é um processo fundamental na concepção e construção de sistemas, pois busca definir os requisitos do sistema em conjunto com os *stakeholders*, produzindo uma série de artefatos em suas fases de concepção. Segundo KOTONYA; SOMMERVILLE (1998), a engenharia de requisitos é o processo de descobrir, analisar, documentar e verificar serviços e restrições para um sistema. Embora não haja uma maneira irrefutável de garantir que as especificações construídas pela engenharia de requisitos sejam totalmente compatíveis com as necessidades dos *stakeholders* e atendam às suas necessidades, este é o maior desafio encontrado na engenharia de requisitos PRESSMAN (2016). As etapas da engenharia de requisitos incluem a concepção e análise de requisitos, levantamento e compreensão dos requisitos junto ao cliente, negociação dos requisitos, especificação e modelagem desses requisitos, tendo como objetivo documentar as necessidades e os propósitos do software.

O pontapé inicial do projeto em si foi uma reunião, como proposto nos *frameworks* ágeis, entre as partes envolvidas, cliente (docentes de Agronomia da 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 a 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.

3.3.1 Requisitos funcionais

A partir disso, os requisitos funcionais do sistema foram definidos e especificados de acordo com a Tabela 3.2, abaixo.

Código	Descrição
RF01	O usuário deve ser capaz de se cadastrar no sistema
RF02	O usuário deve ser capaz de realizar <i>login</i> 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 3.2: 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 3.2 têm o objetivo de especificar as funcionalidades que o sistema proposto deve oferecer.

3.3.2 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 3.3, com o objetivo de centrar os requisitos funcionais descritos na Tabela 3.2 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 <i>login</i> 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 3.3: Casos de Uso do sistema

Uma reprodução visual dos Casos de Uso apontados na Tabela 3.3 e as suas interações com os usuários foi elaborada, na Figura 3.1. 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 funcionalidades do sistema do ponto de vista do usuário. O cliente deve ver neste diagrama as principais funcionalidades de seu sistema.

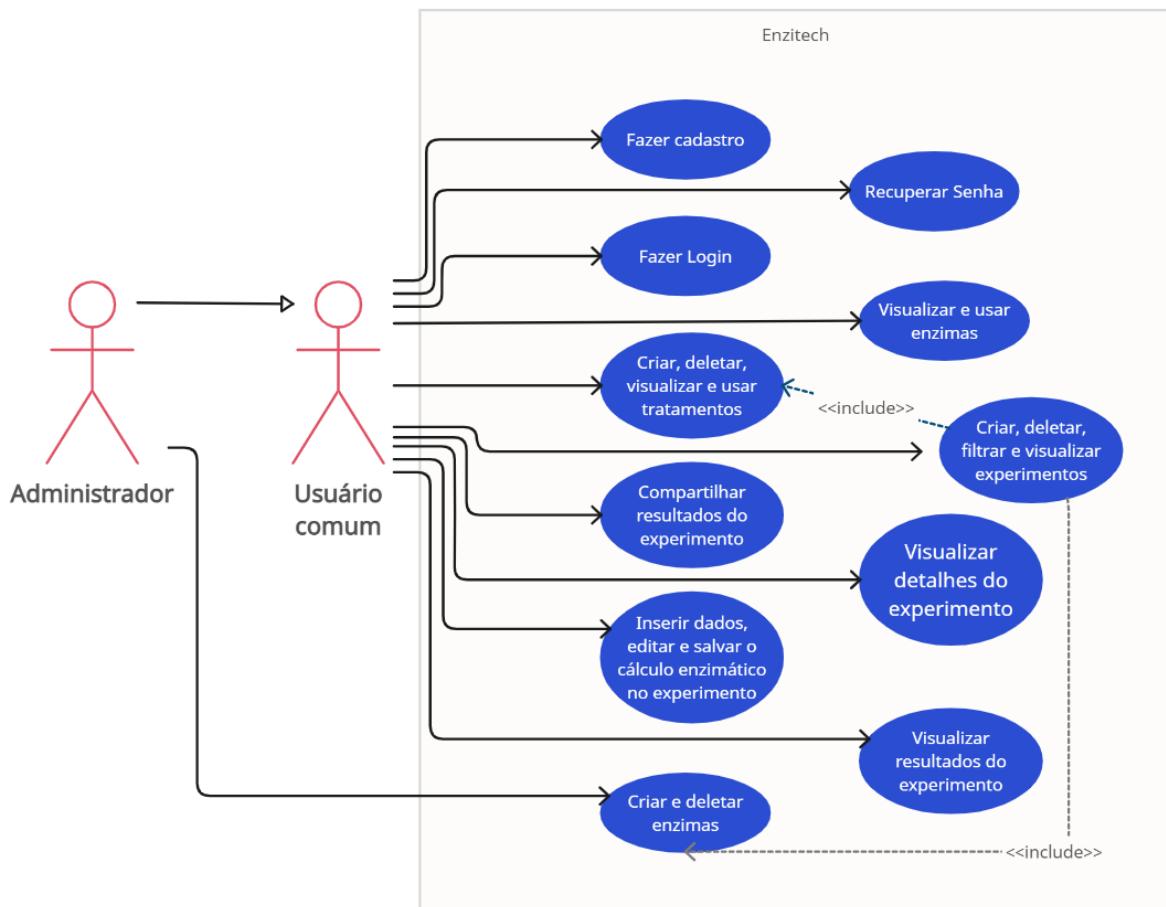


Figura 3.1: Diagrama de Casos de Uso do sistema.

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, portanto, surgiu a necessidade de diferenciar ambos os tipos de usuários no

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

sistema e realizar estas validações para a disponibilização dessa *feature*.

3.4 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 3.2: Parte do protótipo final de baixa fidelidade.

Como é possível ver na Figura 3.2, o protótipo de baixa fidelidade trás o desenho das principais funcionalidades do sistema, além de dar uma ideia do layout final da aplicação.

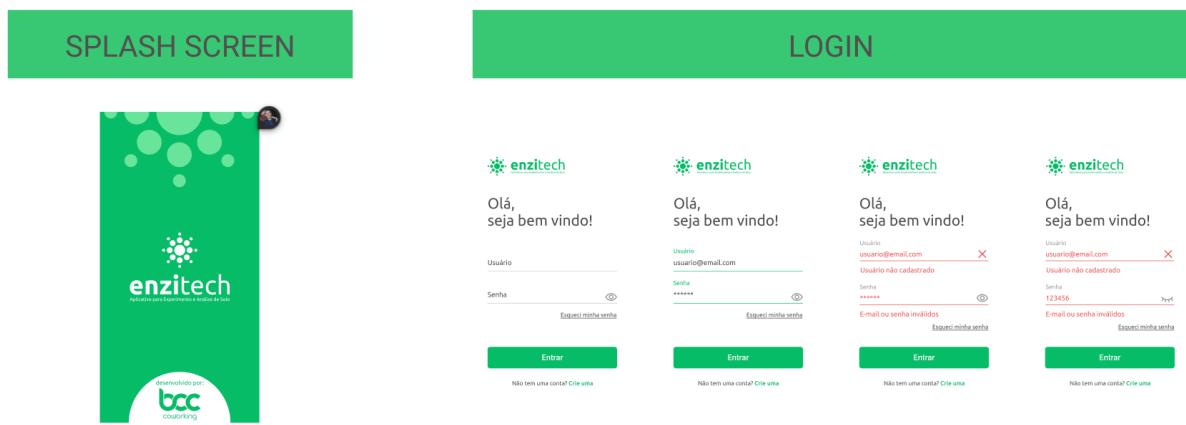


Figura 3.3: Parte do protótipo final de alta fidelidade.

Já nas Figura 3.3 e Figura 3.4, 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 3.4: Visão geral do protótipo final de alta fidelidade.

3.5 Tecnologias e ferramentas envolvidas

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.

3.5.1 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 oficial², 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;
- 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 oficial².

3.5.2 Instalação do Android Studio

Como consta no tutorial oficial², 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.

²Instalação do Flutter no Windows: <https://docs.flutter.dev/get-started/install/windows>

³JetBrains Toolbox App: <https://www.jetbrains.com/pt-br/toolbox-app/>

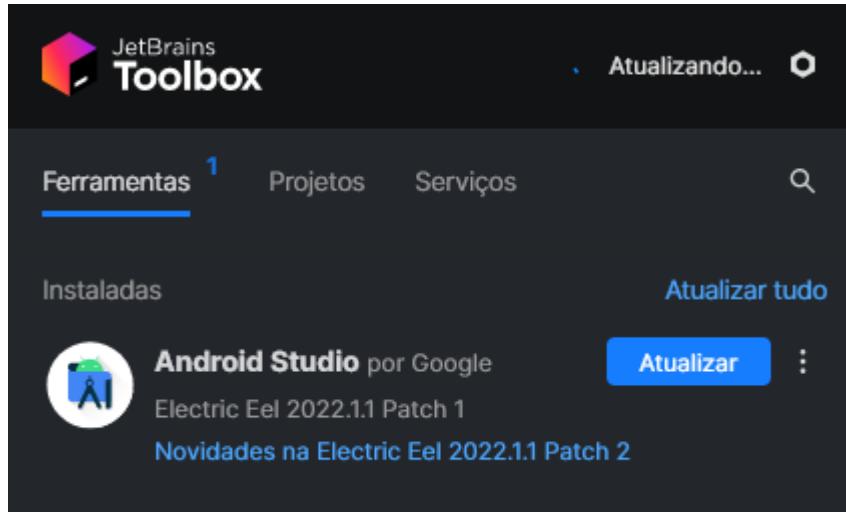


Figura 3.5: *JetBrains Toolbox App*

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. Habilite 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. 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);

3.5.3 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, principalmente se utilizado junto de um emulador Android, portanto, existe a possibilidade de utilizar o VSCode⁴ para seguir como software padrão de desenvolvimento, além disso, no VSCode é possível instalar uma grande quantidade de pacotes desenvolvidos pela comunidade 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 oficial⁴.

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 oficial⁵) ou abrir um projeto Flutter já existente e começar a editar seu código.

3.5.4 Ferramentas e bibliotecas de suporte

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

3.5.4.1 Flutter DevTools

O DevTools⁶ 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;

⁴VSCode: <https://code.visualstudio.com/>

⁵Escreva seu primeiro aplicativo Flutter: <https://docs.flutter.dev/get-started/codelab>

⁶DevTools: <https://docs.flutter.dev/development/tools/devtools/overview>

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;

3.5.4.2 Postman

O Postman⁷ 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 Swagger⁸, para tornar o processo de teste e documentação de APIs mais eficiente.

3.5.4.3 Packages do pub.dev

Os *packages* do pub.dev⁹ são bibliotecas de código-fonte aberto, desenvolvidas por membros da comunidade Flutter, que podem ser utilizadas para implementar recursos em APPs 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.
```

⁷Postman: <https://www.postman.com/>

⁸Swagger: <https://swagger.io/>

⁹pub.dev: <https://pub.dev/>

4

Proposta e Desenvolvimento da Aplicação

Neste capítulo serão abordados diversos aspectos do desenvolvimento do sistema, desde a configuração de todo o 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 Processo de desenvolvimento de software

De acordo com [SOMMERVILLE \(2011\)](#): "O processo de desenvolvimento de software é um conjunto de atividades que visam transformar requisitos de software em um sistema de software. Essas atividades são divididas em fases, cada uma com objetivos específicos e entregas definidas. O processo é composto por atividades de elicitação, análise, projeto, implementação, teste e manutenção. Cada atividade é composta por técnicas, métodos e ferramentas que visam aprimorar a qualidade do software produzido. A escolha do processo de desenvolvimento de software a ser utilizado depende do tipo de projeto, das necessidades do cliente e das características da equipe de desenvolvimento".

Como será elucidado logo após, na seção de Sprints, o processo do projeto conta com variações por questões particulares de todos os envolvidos, um dos principais motivos é por conta do mesmo ser realizado em um ambiente acadêmico. O modelo final do desenvolvimento do software deste projeto está representado na Figura 4.1 abaixo.

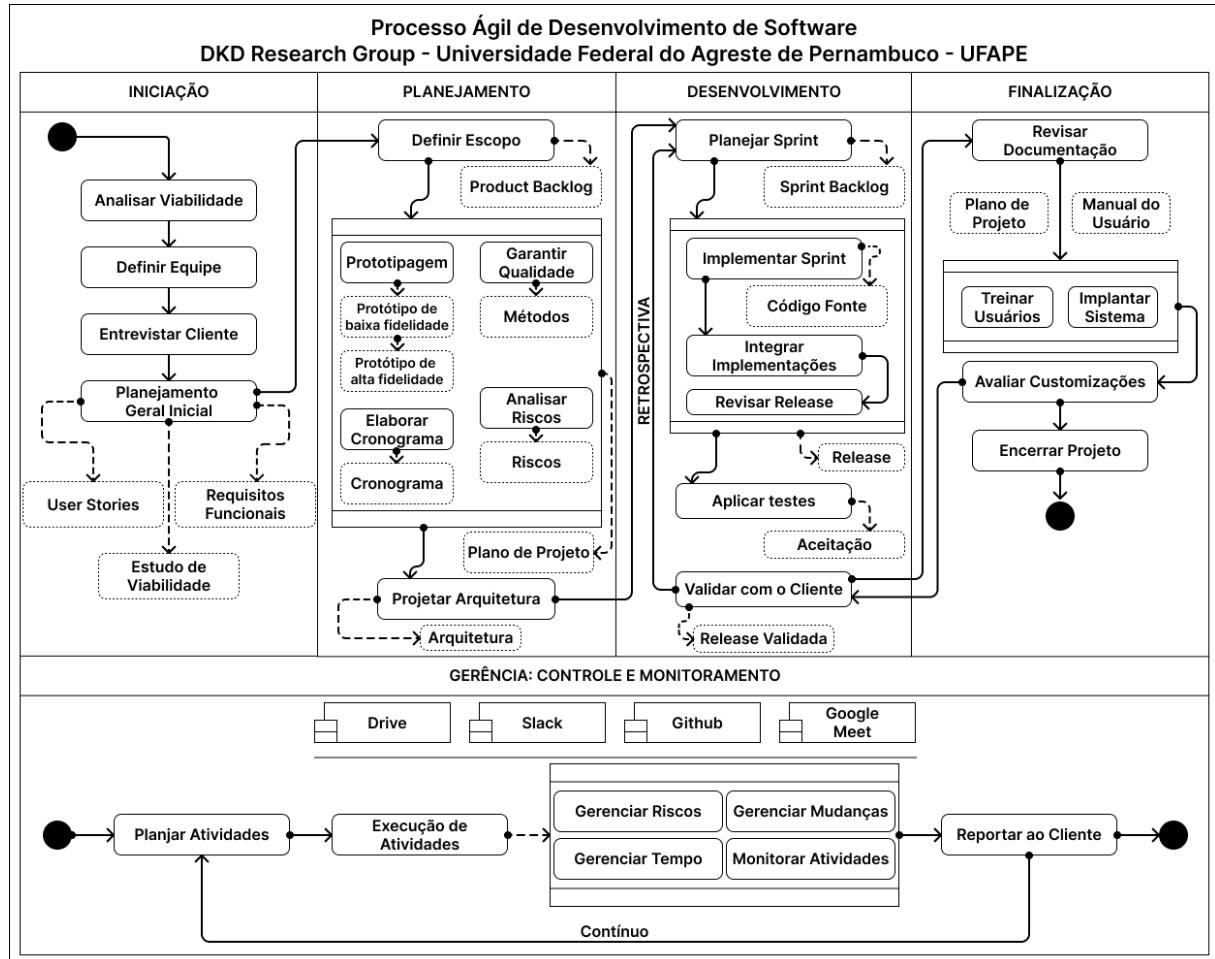


Figura 4.1: Representação do Processo Ágil de Desenvolvimento de Software para o Enzitech

Fonte: Adaptação de DKDGroup

4.2 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 Project¹, 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.

TODO	IN PROGRESS	DONE	BLOCKED
enzitech #8 CI/CD	enzitech #4 Rascunho do Plano de Projeto	Draft Refatoração do código aplicando Clean Code	enzitech_app #10 Integração da Recuperação de Senha
enzitech_app #54 Compartilhar os resultados dos experimentos	enzitech #20 Salvar resultados das enzimas no banco de dados		enzitech_app #8 Página de Recuperação de Senha
enzitech_app #53 Exibição dos dados do experimento	Draft Refatoração da arquitetura do projeto aplicando Clean Arch		Draft [UI/FGMA] Fluxo de recuperação de senha completa
enzitech #21 Exportar resultado como csv			enzitech_app #55 Funcionalidade recuperar senha

Figura 4.2: Quadro de atividades do Enzitech no Github Project¹

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 Project¹, 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.

¹Github Project: <https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects>

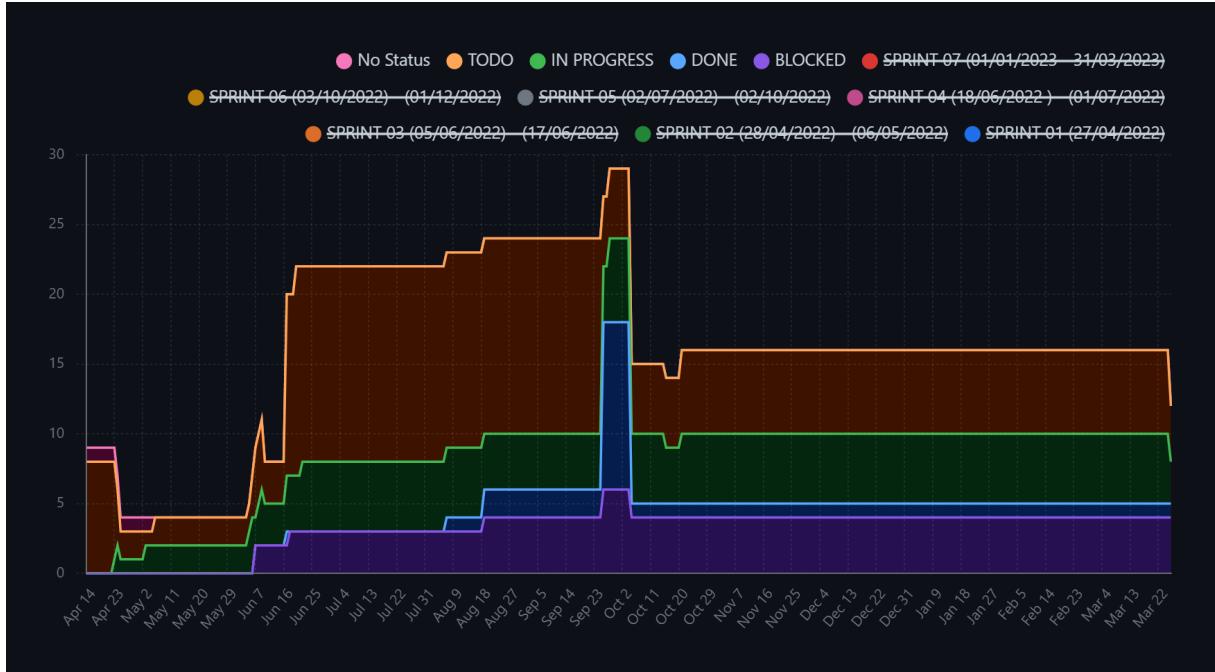


Figura 4.3: Gráfico de *Burn Up* das atividades do Enzitech no Github Project¹

Fonte: Github

A seguir serão explicadas brevemente o planejamento, desenvolvimento e resultados obtidos em cada sprint.

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.

Na segunda sprint foi iniciado o desenvolvimento da tela de *home*, ainda sem integrações, somente para alinhar a estrutura do projeto com a arquitetura inicialmente escolhida, o *Model–view–viewmodel* (MVVM), um padrão de arquitetura em *software* que facilita a separação do desenvolvimento da *User Interface* (UI) – em português, 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.

Flow Chart of MVVM

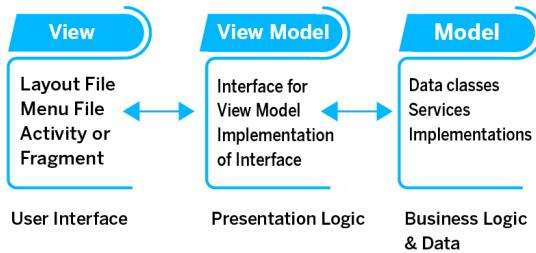


Figura 4.4: Descrição do padrão MVVM

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.

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:

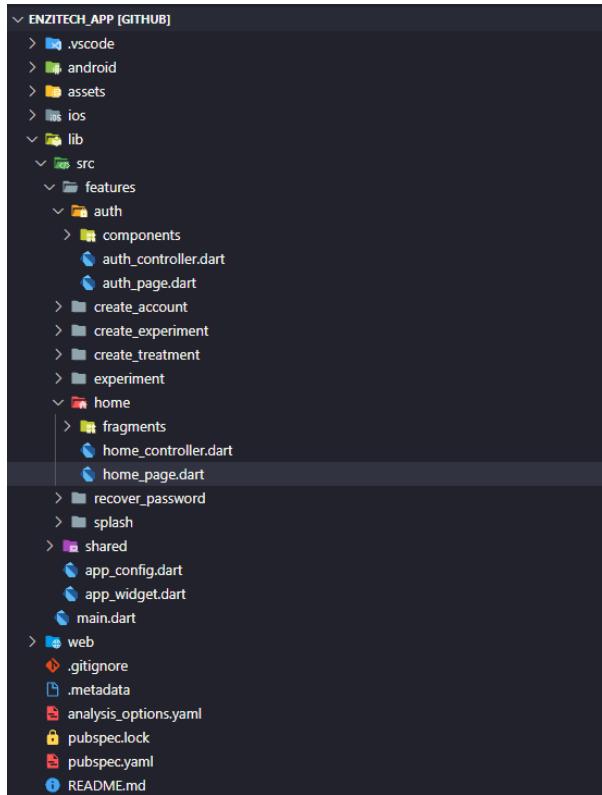


Figura 4.5: Estrutura do projeto ao fim da terceira sprint

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;

No fim deste arquivo, na seção de Apêndice, em "Código da tela *Home Page* desenvolvida na Terceira Sprint", um trecho da *view* da *Home Page* desenvolvida até este ponto está disponível.

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 conclusão da sprint.

Já na 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*² 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 4.6, 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.

²*Firebase*: <https://firebase.google.com/>

The screenshot shows the Firebase App Distribution interface for the Enzitech project. At the top, there's a header with the project name "enzitech" and user profile icons. Below the header, the title "App Distribution" is displayed, along with tabs for "Lançamentos", "Links de convite", and "Testadores e grupos". A prominent dashed blue box contains instructions: "Arraste e solte um arquivo APK ou AAB para criar uma nova versão. Os arquivos AAB precisam ser vinculados." with a help icon, and links to "Saiba mais" and "Procurar". Below this, the "Versões (1)" section shows a single release:

- 0.1.2 (3)** - released on 01 de julho de 2022 às 13:09:11 UTC-3.
- Statistics: Convidado (4), Aceito (1), Download concluído (0), Feedback (0).
- Actions: Copiar, Fazer o download, Excluir.
- Sub-sections: Testadores, Notas da versão, Feedback do testador (BETA).
- Testers: ENZITECH - INTERNO (4 testadores).

Figura 4.6: Aba do *Firebase App Distribution* do projeto Enzitech

Fonte: *Firebase*²

Na quinta 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.

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 4.7 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.

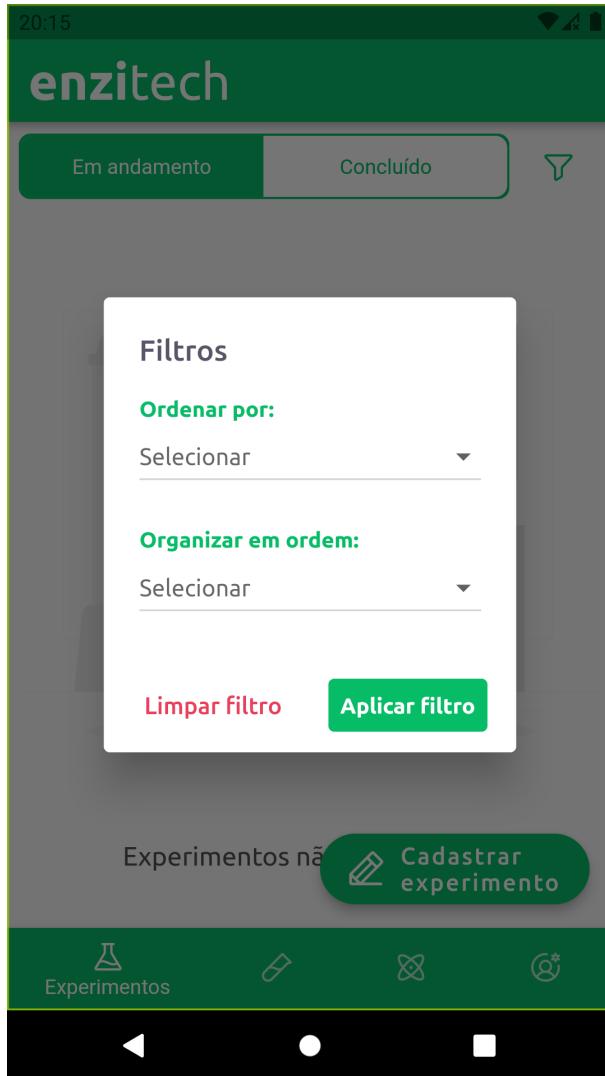


Figura 4.7: Filtros de experimento do Enzitech

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.

Como podemos ver no Código Fonte Trecho de código para realizar a checagem de autenticação e pré-carregamento dos dados desenvolvido na Sprint 6 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 4.8 é possível ver sua aplicação em um tratamento.

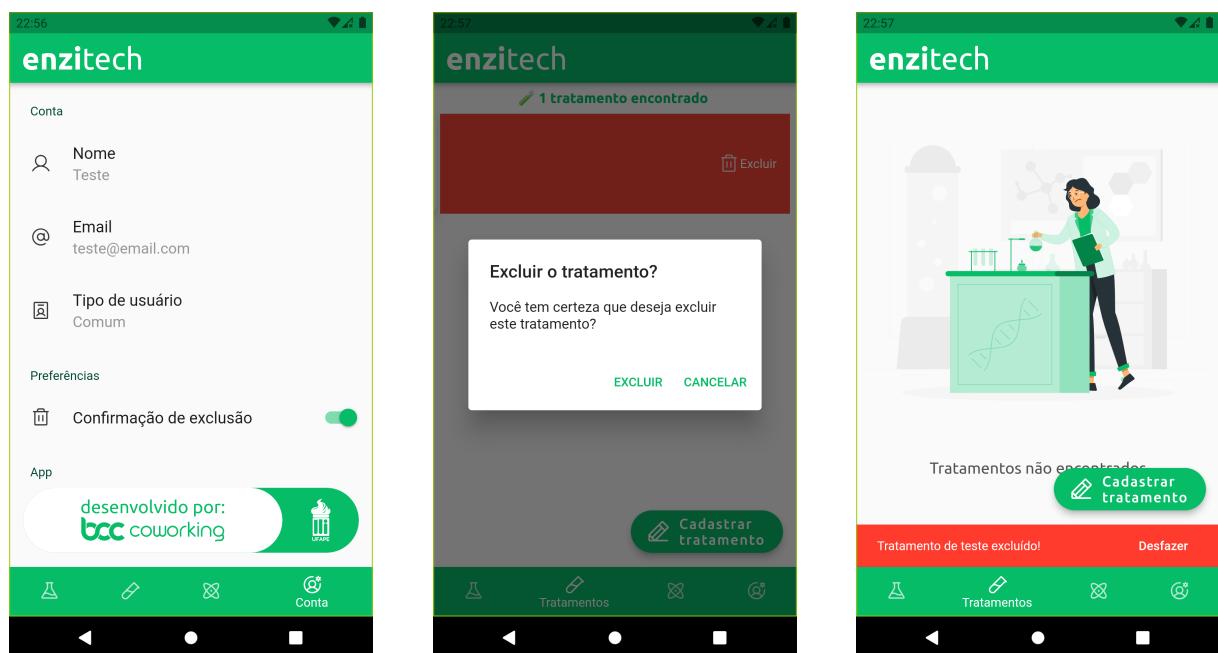


Figura 4.8: Fluxo de exclusão de um tratamento com a opção de confirmação de exclusão ativada

Na sétima e última sprint do projeto o projeto houve uma enorme refatoração de arquitetura, como comentado anteriormente, 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;

Nesta refatoração foi aplicado o conceito de Arquitetura Limpa proposto pelo Uncle Bob, que já foi explicado no capítulo de ?? e será aprofundada na seção Arquitetura do sistema deste capítulo, assim como houve a troca de alguns pacotes do aplicativo, portanto, com a aplicação da arquitetura limpa, foi possível isolar os pacotes adicionados ao projeto utilizando contratos/interfaces para abstrair sua implementação, permitindo uma troca mais fácil, caso necessário, no futuro... A estrutura do projeto ficou da seguinte forma:



Figura 4.9: Estrutura final do aplicativo em Flutter para o Enzitech

Fonte: Autoria própria

Sintetizando toda essa estrutura, temos a separação dos requisitos funcionais baseadas em *features*, como mostrado na Figura 4.9, tudo relacionado a enzimas no aplicativo está na *feature "enzyme"*, são elas: as telas, *widgets* e os *ViewModels* de cadastrar, deletar e obter listagem das enzimas, isto para a camada de *Presentation*, seguindo o que a arquitetura exige, na camada de *Domain* estão as entidades, interfaces e implementações dos Casos de Uso e a interface para os Repositórios, por fim, na camada de *Data* estão o restante das classes necessárias para concluir uma *feature*, são as interfaces e implementações do *DataSource*, as implementações dos Repositórios e os *Data Transfer Objects* (DTOs), que fazem a conversão dos dados externos para uma entidade interna do aplicativo. Além desse escopo, estão na pasta denominada *core* todas as implementações para os arquivos estruturais do APP, como as rotas, as injecções de dependência, falhas, entre outros, já na pasta *shared*, como o nome já diz, estão todos os outros arquivos e implementações para os utensílios em comum do APP, como os *widgets* comuns a todo o sistema, as configurações do *design-system*, *validators*, *extensions*, dentre outros.

O *back-end* também teve seus ajustes finais para disponibilizar todas as APIs de acordo com o que o app esperava e como os requisitos funcionais pediam, portanto, todo o sistema do Enzitech ficou pronto para ser utilizado ao final desta sprint.

4.3 Back-end

Como já abordado, o sistema foi feito juntamente com o Laboratório BCC Coworking, portanto, a parte encarregada do mesmo foi o desenvolvimento do *back-end* que forneceria uma API para o aplicativo consumir, sendo assim, nesta seção trago detalhes resumidos de sua implementação e como ele está ligado ao APP.

O *back-end* é responsável por gerenciar as solicitações feitas pelo cliente (APP), manipulá-las, e realizar as operações necessárias no Banco de Dados, inclusive, é no *back-end* onde os cálculos são realizados, retirando a responsabilidade do cliente de realizar possíveis grandes quantidades de cálculo, podendo desencadear problemas de performance no aplicativo a depender do dispositivo em que o APP está sendo executado, além disso, manter estas responsabilidades no *back-end* permite que outros clientes possam realizar as mesmas coisas, bastando apenas a consulta em sua API, evitando reescrita de código.

O *back-end* foi feito totalmente em TypeScript utilizando o Nest.js (*framework back-end* que auxilia o desenvolvimento de aplicações eficientes, escaláveis e confiáveis), o qual, ao ser executado, fornece uma API acessível através de uma URL específica para clientes/dispositivos acessarem suas funcionalidades.

4.4 Modelagem dos dados

Assim como o *back-end*, a configuração do Banco de Dados foi responsabilidade dos desenvolvedores envolvidos do Laboratório BCC Coworking, porém, a fase de modelagem dos

dados é de suma importância também para o APP, pois influencia no desenvolvimento do mesmo, sendo assim, houve uma integração multidisciplinar entre as equipes para esta fase, foi criado um modelo lógico para organizar a camada de dados de todo o sistema, incluindo atributos e relacionamentos de entidades. Esse modelo foi utilizado como guia para o desenvolvimento do Enzitech, utilizando ferramentas para construção do diagrama de entidade-relacionamento.

4.4.1 Diagrama de Entidade-Relacionamento

O Diagrama Entidade-Relacionamento (ER) é uma ferramenta de modelagem utilizada para representar entidades, seus atributos e relacionamentos em um sistema de banco de dados. Ele é composto por elementos gráficos como entidades, atributos e relacionamentos, que são utilizados para representar objetos do mundo real e suas interações.

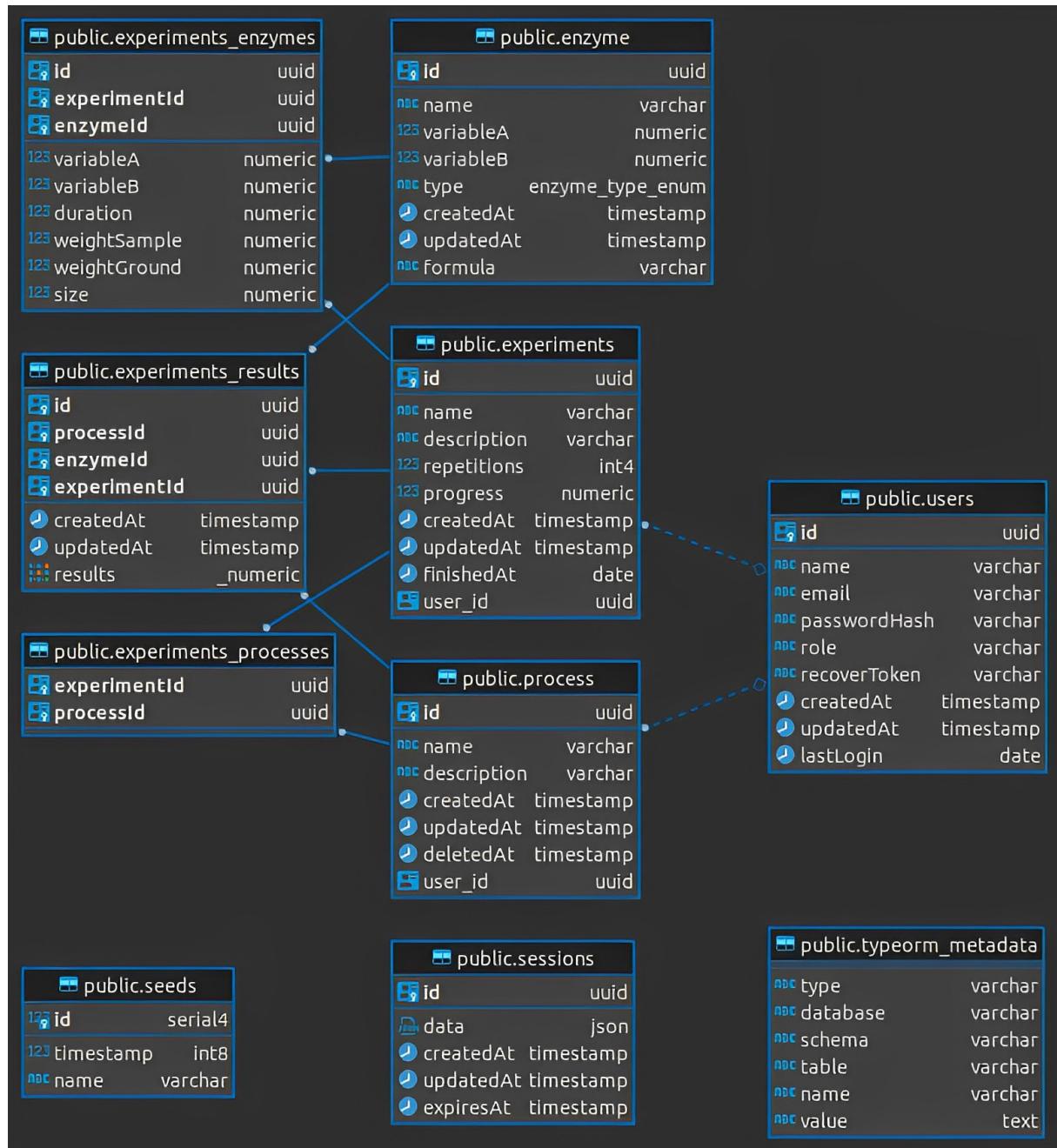


Figura 4.10: Diagrama de Entidade-Relacionamento do Enzitech

Fonte: DBeaver³

Através da modelagem com o diagrama ER, é possível criar um modelo lógico para o banco de dados, abstraindo as características específicas da tecnologia a ser utilizada, facilitando a implementação do modelo físico do banco de dados. Na Figura 4.10 é possível ver como ficou o diagrama ER do Enzitech, após consulta do mesmo no software para gerenciamento de banco de dados, DBeaver³.

³DBeaver: <https://dbeaver.com/docs/wiki/ER-Diagrams/>

4.5 Arquitetura do sistema

A Arquitetura do sistema por completo pode ser considerada uma arquitetura híbrida, pois a mesma utiliza princípios de quatro padrões arquiteturais, sendo eles:

1. MVVM: Arquitetura de UI para evitar o acoplamento das camadas de UIs das de negócios;
2. Cliente-Servidor: Arquitetura para comunicação entre o *front-end* e o *back-end/API*;
3. Monolítico: Arquitetura em que todo o sistema é desenvolvido em um único bloco de código, uma só instancia que gerencia todos os estados da aplicação, sem divisão clara de módulos ou serviços;
4. Arquitetura Limpa: Arquitetura com o foco em promover a implementação de sistemas que favorecem reusabilidade de código, coesão, independência de tecnologia e testabilidade;

O *back-end* gerencia todas as solicitações feitas pelo APP, manipulando os dados por meio de operações CRUD em uma instância de banco de dados Postgresql⁴.

A comunicação entre o cliente e o *back-end* é estabelecida por meio de requisições HTTP, que são processadas pela API do *back-end* e resultam em respostas adequadas, formatadas em JSON. O cliente, por sua vez, é responsável por exibir as informações ao usuário final e coletar as interações do usuário para posteriormente enviar ao *back-end*.

A arquitetura adotada busca garantir a escalabilidade e a segurança do sistema, bem como a facilidade de manutenção. O *back-end* é projetado seguindo os padrões RESTful (capacidade de determinado sistema aplicar os princípios de REST), o que permite a integração com outros sistemas e garante a interoperabilidade. O uso do Postgresql⁴ como banco de dados permite a manipulação de grandes volumes de dados e garante a consistência dos dados armazenados. Além disso, a utilização de boas práticas de segurança, como a autenticação e autorização de usuários, garante que apenas usuários autorizados possam acessar os dados do sistema.

Por fim, a arquitetura do sistema é projetada para atender às necessidades do cliente *mobile*, oferecendo uma experiência de usuário agradável e uma interface responsiva, que garante o bom desempenho do APP. A seguir, serão destrinchadas as arquiteturas do aplicativo e da infraestrutura do sistema.

⁴Postgresql: <https://www.postgresql.org/>

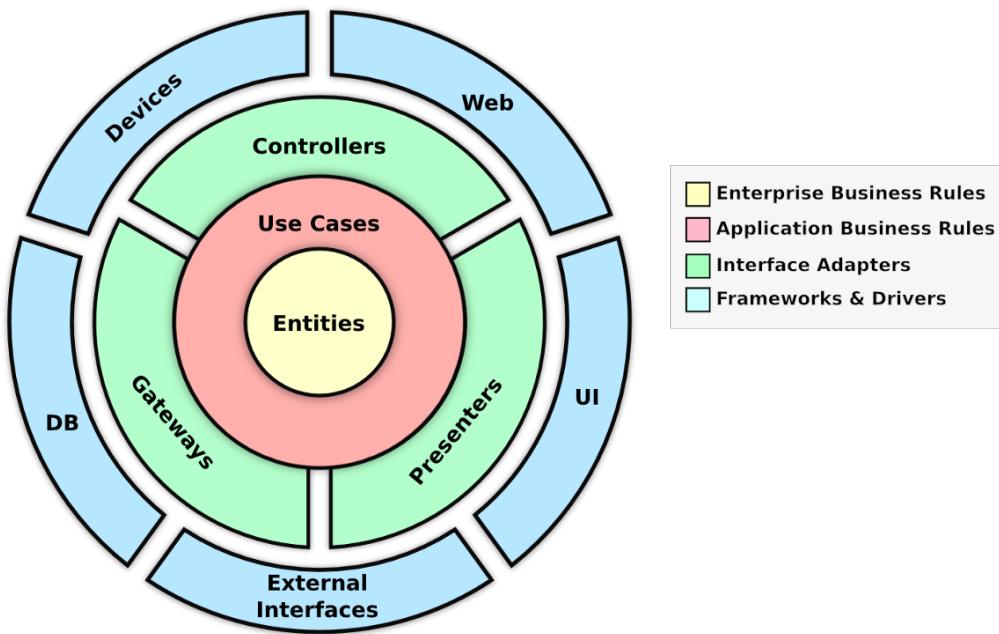


Figura 4.11: Camadas de uma Arquitetura Limpa

Fontes: [MARTIN \(2018\)](#) e [FLUTTERANDO \(2021\)](#)

Como apontado por [MARTIN \(2018\)](#), uma arquitetura deve conter pelo menos estas quatro camadas principais e independentes para ser considerada “limpa”:

- **Regras de Negócio Corporativas:** As entidades são modelos de dados que representam as regras de negócio mais importantes de um sistema e estão no topo das camadas. Elas devem ser puras, não conhecendo nenhuma outra camada, mas são conhecidas pelas outras camadas;
- **Regras de Negócio da Aplicação:** Os casos de uso representam as ações que um usuário pode fazer na aplicação. Os casos de uso conhecem apenas as entidades, não sabendo sobre as implementações de camadas de baixo nível. Se precisarem acessar uma camada superior, devem fazê-lo por meio de contratos definidos por interfaces, seguindo o Princípio de Inversão de Dependências do SOLID;
- **Adaptadores de Interface:** Camada responsável por auxiliar as Regras de Negócios, convertendo os dados externos em um formato que atenda aos contratos de interface definidos por ela;
- **Frameworks & Drivers (Externos):** Abstrações feitas para aumentar a facilidade da conexão dos artefatos externos como um banco de dados, protocolo de requisições web ou uma interface gráfica, facilitando a troca sem alterar o comportamento do sistema;

Desta forma, a aplicação da arquitetura limpa sugerida no Flutter/Dart pela comunidade [FLUTTERANDO \(2021\)](#) segue o fluxo da Figura 4.12 a seguir:

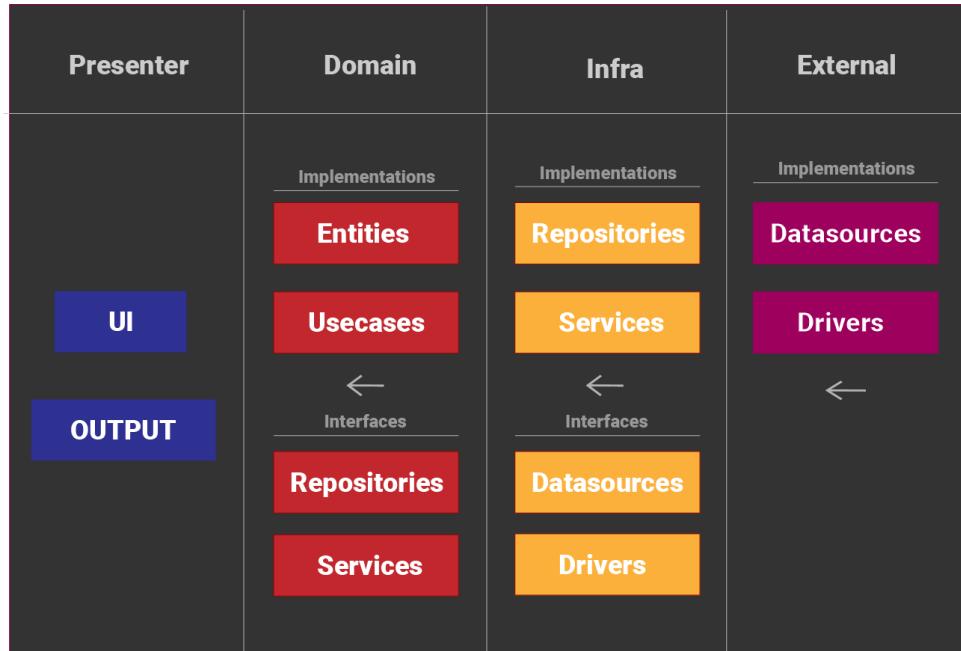


Figura 4.12: Sugestão da estrutura das camadas de uma Arquitetura Limpa no Flutter

Fonte: [FLUTTERANDO \(2021\)](#)

Para aplicação *mobile* desenvolvida, esta proposta de arquitetura foi abstraída em três camadas maiores, mas seguindo o padrão e as peculiaridades da proposta acima, sendo assim, a arquitetura seguiu com as seguintes camadas:

- **Presentation:** Camada responsável por declarar as entradas, saídas e interações da aplicação. onde ficarão os *Widgets*, *Pages*, *ViewModels* das respectivas *Pages* e também a Gerência de Estado;
- **Domain:** Camada que hospeda as Entidades (Regras de Negócio Corporativa), os Casos de Uso (Regras de Negócio da Aplicação) e os contratos dos Repositórios (a responsabilidade de implementação desse objeto é repassado para outra camada mais baixa), fazendo com que esta camada seja responsável apenas pela execução da lógica de negócio, não havendo implementações de outros objetos fora da mesma;
- **Data:** Camada que dá suporte à camada Domain, implementando suas interfaces. Para isso, adapta os dados externos para que possa cumprir os contratos do domínio. Nesta camada é possível implementar alguma interface de um Repositório ou Serviço que pode ou não depender de dados externos como uma API ou acesso a algum *Hardware* como por exemplo GPS. Para que o Repositório possa processar e adaptar os dados externos foram criados contratos para esses serviços visando passar a responsabilidade de implementação para a camada mais baixa dessa arquitetura: o

DataSource, esta camada interna em Data permite acessar um dado externo, como a API do Enzitech ou um Cache Local usando *SharedPrefs* por exemplo, como há uma redução de camadas, é aqui onde os acessos externos são implementados seguindo o mesmo padrão de contratos como acontece entre Domain e Data, realizado na aplicação fora das camadas de *features* que possuem a mesma estrutura, assim, ainda preservando a separação de responsabilidades e tornando os pacotes externos desacoplados da nossa aplicação, como deve ser na arquitetura limpa;

Para concluir, um exemplo abaixo da arquitetura de toda a infraestrutura do Enzitech:

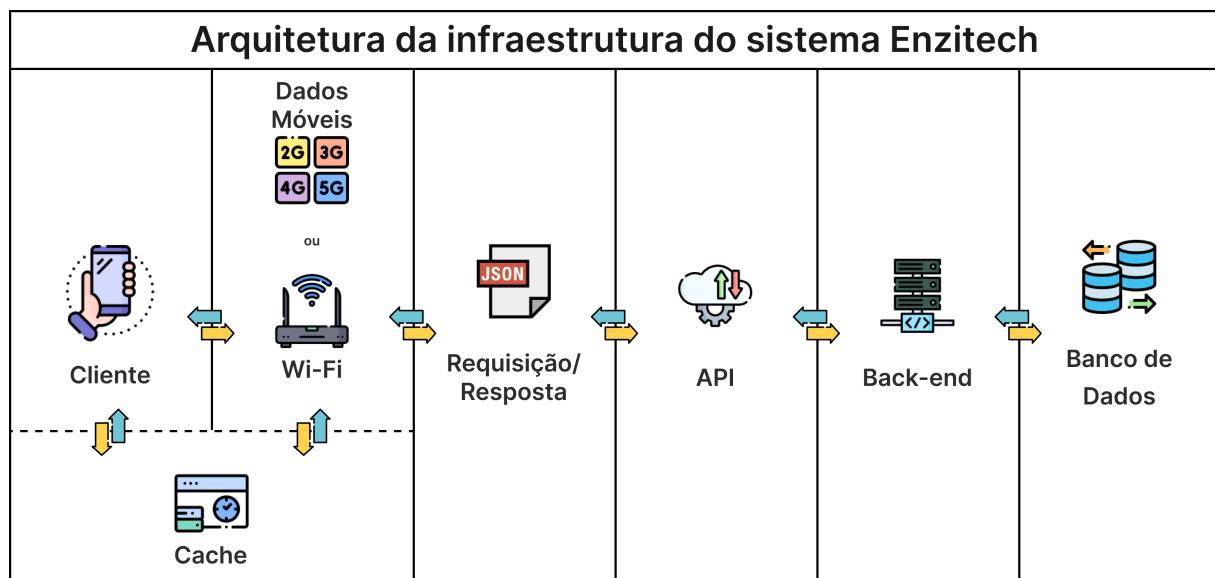


Figura 4.13: Arquitetura simplificada da infraestrutura do Enzitech

A arquitetura da aplicação ilustrada na Figura 4.13 foi desenvolvida baseado na proposta de arquitetura de MARTIN (2018), também do seu livro sobre Arquitetura Limpa, arquitetura esta que é separada em camadas, com o objetivo de simplificar a codificação, possibilitando sua manutenção e atualização de forma prática, reduzindo a dependência do código. Com base nesta arquitetura descrita, é possível destacar alguns pontos adicionais:

- O aplicativo *mobile* é responsável por fornecer uma interface de usuário amigável e eficiente para interação com o sistema. Isso inclui a exibição de informações, a coleta de dados de entrada e a navegação entre diferentes telas e funcionalidades;
- A escalabilidade do sistema é uma preocupação importante, uma vez que o número de usuários e a quantidade de dados podem crescer significativamente ao longo do tempo. Para garantir a escalabilidade, é possível adotar soluções como a utilização de servidores em nuvem, a implementação de técnicas de cache e a otimização de consultas no banco de dados;

- A arquitetura do sistema foi projetada para facilitar a manutenção e o desenvolvimento de novas funcionalidades. Isso inclui a utilização de padrões de codificação consistentes, a documentação adequada do código-fonte e a realização de testes unitários e de integração para garantir a qualidade do software;

4.6 Testes

Antes de ser implantado, o aplicativo foi testado em suas *releases* pela equipe interna do Enzitech, além disso, no desenvolvimento, foram utilizados técnicas de teste, como integração, aceitação e de unidade, no fim deste arquivo, no capítulo Testes é possível verificar esta etapa.

Dentro da seção "Trecho de código do teste de unidade para enzima", é possível ver um exemplo de teste de unidade escrito na linguagem de programação Dart, usando o *framework* de testes padrão do Flutter, o teste é uma função anônima (sem nome) que verifica se a instância da classe *EnzymeEntity* não é nula. O nome do teste é "*Should not return null on Enzyme entity*".

Dentro do corpo da função, é criada uma nova instância da classe *EnzymeEntity* e atribuída à variável *enzyme*. O construtor da classe *EnzymeEntity* é chamado com argumentos específicos para inicializar as propriedades da instância. Em seguida, é usada a função *expect()* do *framework* de teste para verificar se a instância da classe *EnzymeEntity* criada anteriormente não é nula. Se a expectativa for cumprida, o teste passa. Caso contrário, o teste falha.

Este teste serve como uma verificação básica para garantir que a classe *EnzymeEntity* esteja sendo inicializada corretamente e não esteja retornando um valor nulo.

4.7 Implantação

O *back-end* e o banco de dados do sistema Enzitech foram implantados na infraestrutura da UFAPE, utilizando-se dos recursos de processamento e armazenamento disponibilizados pela instituição. A API desenvolvida para o projeto foi disponibilizada através de uma rota pública, permitindo o acesso aos dados e funcionalidades do sistema a partir do aplicativo desenvolvido para os usuários finais. A escolha pela hospedagem na infraestrutura da UFAPE teve como objetivo garantir a segurança, estabilidade e escalabilidade da aplicação, além de possibilitar a utilização de recursos computacionais locais e adequados para as necessidades do projeto.

O aplicativo móvel foi projetado para alterar facilmente entre os ambientes de desenvolvimento, teste ou produção, desta forma, em sua versão final, o APP foi distribuído internamente via *Firebase App Distribution*, porém sendo possível também a sua publicação na *Google Play Store*, loja de APPs do Android.

5

Análise dos resultados

A ideia deste capítulo é trazer uma visão geral sobre todas as funcionalidades desenvolvidas, o fluxo de navegação entre as *features* e como o APP ficou em sua última versão.

5.1 Apresentação do Enzitech

Após concluir todos os requisitos funcionais, a versão final do Enzitech lida com dois tipos de usuário, Comum ou Administrador. O primeiro deles é referente ao perfil de todos os usuários, pessoas que utilizarão o sistema para criar e calcular seus experimentos. Já o segundo é destinado ao administrador geral de todo o sistema, normalmente uma única pessoa que ficará encarregada de gerir o Enzitech disponibilizando os dados corretamente.

A diferença entre os dois perfis está na possibilidade de criação de enzimas, funcionalidade restrita ao Administrador, devido a necessidade de ajuste também no *back-end*, ou seja, o Administrador fica responsável por gerir (criar e excluir) enzimas e solicitar a inclusão de novos cálculos para outros tipos, outro detalhe importante é que as enzimas criadas pelo Administrador ficam disponíveis para todos os usuários do sistema, as demais funcionalidades ficam disponíveis para ambos os tipos de usuário.

Sendo assim, para ter acesso ao sistema, o usuário terá que inserir seus dados de acesso, e-mail e senha, na tela de login (Figura 5.1). A autenticação é fundamental para que o usuário tenha acesso às funcionalidades do APP, o login satisfaz o caso de uso UC02.



Figura 5.1: Fluxo de login do Administrador previamente cadastrado

Fonte: Aplicativo Enzitech desenvolvido pelo autor

Na Figura 5.1 é possível ver o fluxo inicial do APP, ao abri-lo, é feita uma verificação na *splashscreen* (primeira imagem da sequência) para determinar se existe usuário logado ou não, caso negativo, o APP redireciona para a tela de login, onde é possível criar uma conta (caso de uso UC01), recuperar senha (caso de uso UC03) ou logar com suas credencias, após o login, ou, caso o usuário já estivesse logado, o app redireciona para a *homepage*, onde estão todas as funcionalidades disponíveis, a primeira delas é a listagem de experimentos (caso de uso UC07), nesta tela é possível além da listagem, filtrar, excluir ou criar, como será mostrado em breve.

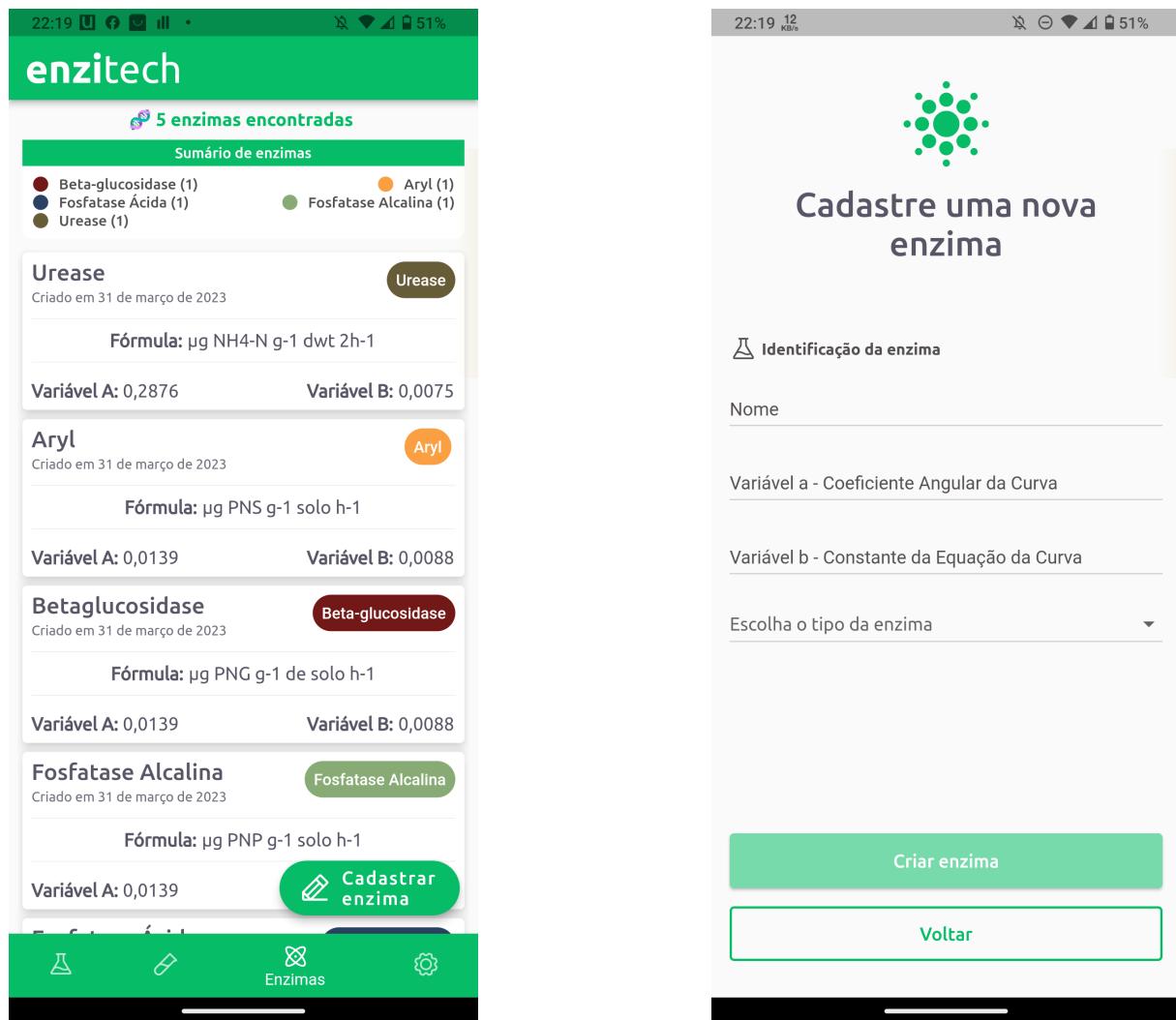


Figura 5.2: Fluxo de listagem, exclusão e criação de enzimas

Fonte: Aplicativo Enzitech desenvolvido pelo autor

Na Figura 5.2 estão as funcionalidades de listagem, exclusão e criação de enzimas, esta última restrita ao administrador, satisfazendo os casos de uso UC05 e UC06. Criar uma enzima no sistema é fundamental para a criação de um experimento.

Abaixo, na Figura 5.3, está o fluxo de listagem, criação e exclusão dos tratamentos (caso de uso UC04), a criação de um tratamento também é fundamental para a criação de um experimento.

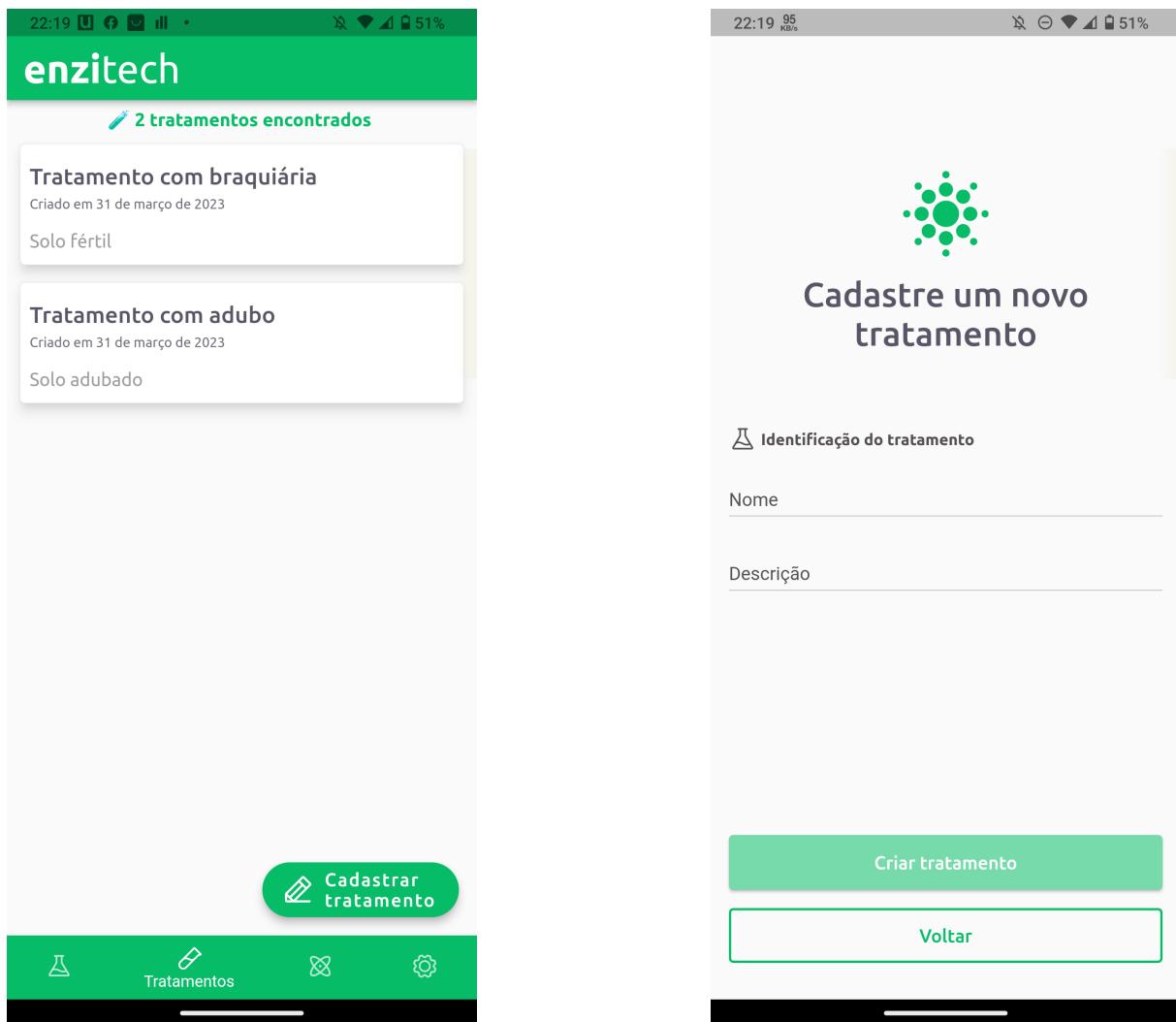


Figura 5.3: Fluxo de listagem, exclusão e criação de tratamentos

Fonte: Aplicativo Enzitech desenvolvido pelo autor

Após criado pelo menos uma enzima e um tratamento, o usuário consegue criar seu primeiro experimento, como mostrado no fluxo da Figura 5.4 abaixo.

É possível ver na primeira imagem, na tela de experimentos, o filtro de "experimentos concluídos" aplicado, seguindo o fluxo, na segunda imagem é solicitada as informações de identificação do experimento, nome e descrição, a seguir, os tratamentos daquele experimento e quantas repetições serão realizadas, logo após, surge o seletor de enzimas, na quarta imagem (terceira etapa do processo de criação de experimento), nele é possível escolher quais enzimas farão parte do experimento, logo após surge a última etapa para o preenchimento dos valores variáveis daquele experimento, após criado (concluindo o caso de uso UC07), o usuário é levado para a funcionalidade de visualizar o experimento, explicado na Figura 5.5 a seguir.

Após o experimento criado, o usuário pode ver seus detalhes, como a quantidade de enzimas, tratamentos e repetições, seu progresso (caso de uso UC08) e a possibilidade de excluí-lo, além disso, o acesso à outras duas funcionalidades, a de cálculo enzimático, para inserção de dados no experimento, e a de resultados, para a visualização e compartilhamento desses dados

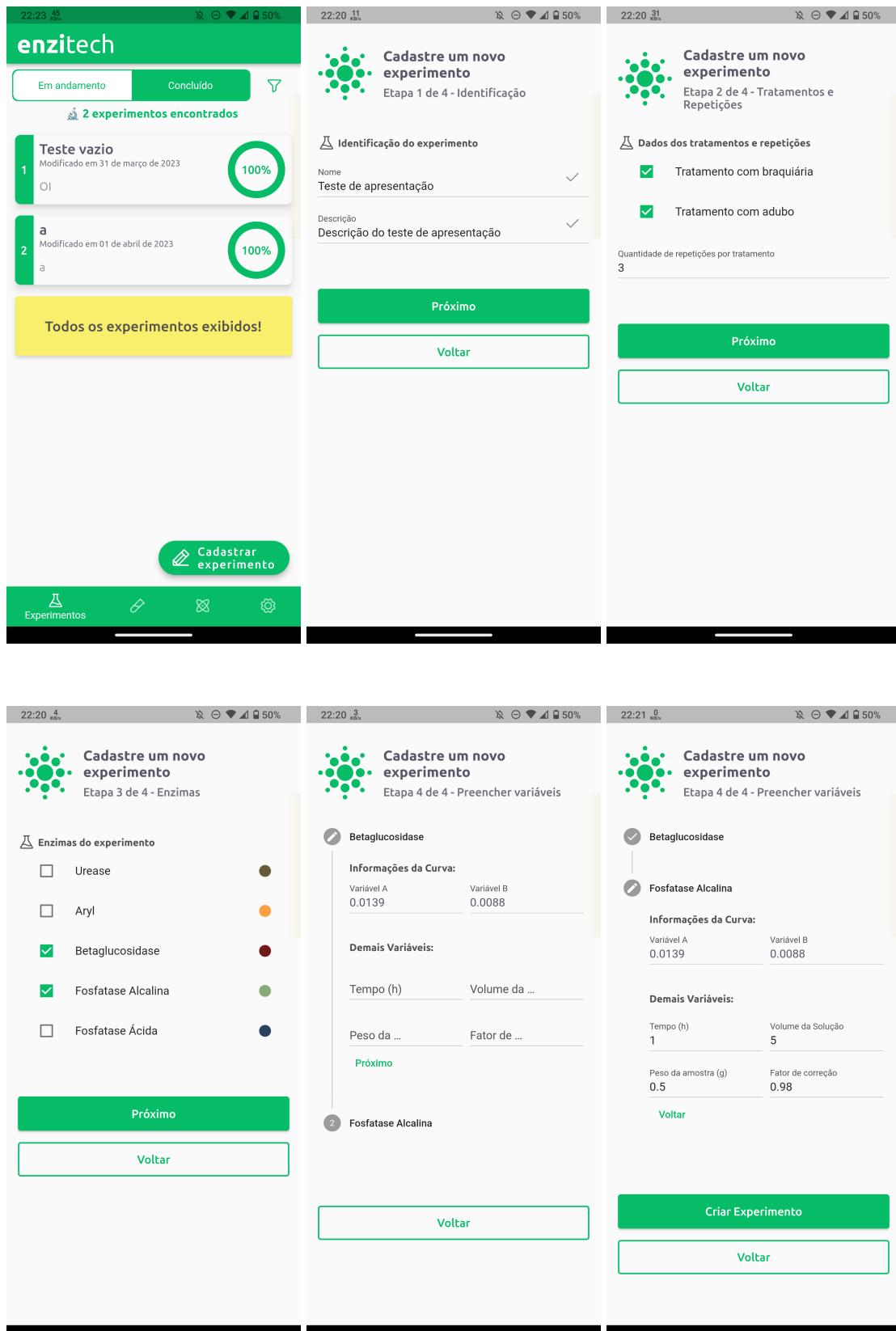


Figura 5.4: Fluxo de criação de um experimento
Fonte: Aplicativo Enzitech desenvolvido pelo autor

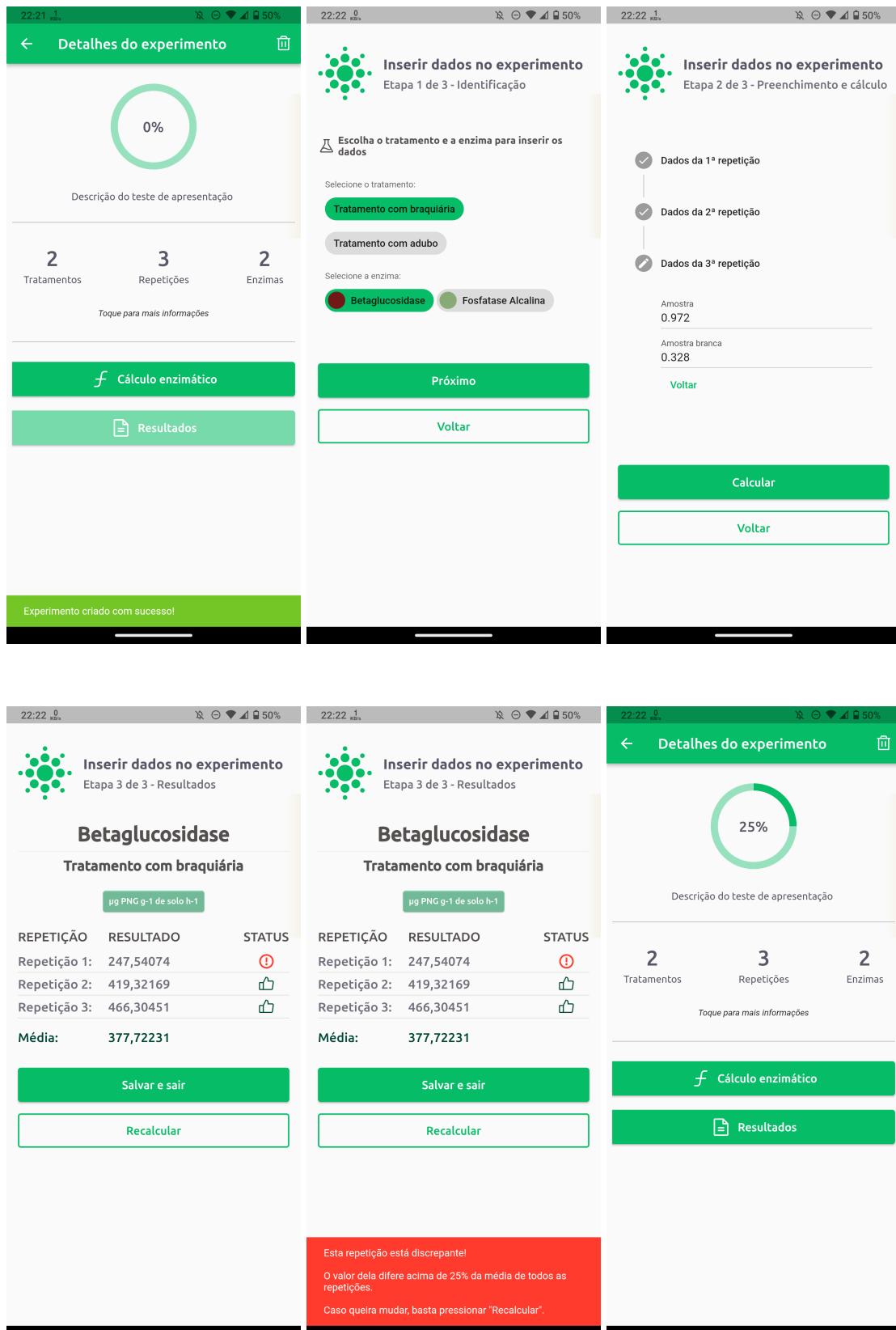


Figura 5.5: Fluxo de detalhamento e cálculo enzimático de um experimento
Fonte: Aplicativo Enzitech desenvolvido pelo autor

também estão contidos nesta tela, todas elas serão explicadas a seguir.

Acompanhando novamente a Figura 5.5, para o preenchimento do experimento é necessário escolher um tratamento e uma enzima, assim, gerando um conjunto de informações para prosseguir com a inserção dos valores em suas respectivas repetições, após todos os valores inseridos, o usuário clica em carregar e é levado para uma tela de resultados deste cálculo e a média (quarta e quinta imagem), nesta tela, é possível ver os resultados e receber um *feedback* sobre a discrepância dos valores em comparação com a média de todos os resultados das repetições, assim, sendo possível perceber algum valor que pode estar inserido incorretamente, resultando em dados errôneos, desta forma, o usuário pode recalcular, corrigindo com novos valores ou prosseguir, salvando e saindo da tela de cálculo (caso de uso UC09).

Em seguida, quando um experimento já tem dados suficientes (progresso maior ou igual a 1%), é possível ver e compartilhar os resultados obtidos (casos de uso UC10 e UC11), mostrados a seguir na Figura 5.6.

As duas primeiras imagens do fluxo da Figura 5.6 são de experimentos diferentes, a primeira é de um que já foi concluído, nele é possível ver que o usuário tem a ação de cálculo enzimático bloqueado, já na segunda imagem é o experimento que seguirá o fluxo de visualização e compartilhamento dos resultados aqui, este experimento possui um tratamento, duas repetições, e cinco enzimas cadastradas.

Ao entrar nos resultados do experimento, o usuário consegue visualizar de forma organizada todos os dados preenchidos como uma listagem com tabelas para cada combinação de enzima e tratamento feita, nesta tela, é possível salvar o resultado em uma planilha Excel no formato NOME-DO-EXPERIMENTO.XLSX no armazenamento do dispositivo, como é possível ver na terceira, quarta e quinta imagem do fluxo da Figura 5.6, a planilha é montada seguindo o mesmo padrão, para cada enzima é criado uma página, e dentro de cada página os resultados são montados com suas repetições para cada tratamento do experimento (caso de uso UC10).

Além disso, o usuário pode compartilhar diretamente o arquivo para qualquer APP externo que suporte esta ação, ao compartilhar um experimento, o salvamento dele também é realizado (caso de uso UC11).

Por fim, o usuário tem acesso à uma tela de configurações na *home* do APP (Figura 5.7), nela é possível ter acesso às seguintes funcionalidades adicionais: informações do APP, seus dados de login, uma configuração para a ativação e desativação do *AlertDialog* para confirmação da exclusão de itens no APP, informação da quantidade de resultados de experimentos salvos localmente, informações sobre ambiente e versão do APP e a opção de deslogar do sistema. Além, disso, como mostrado na última imagem, o APP tem uma funcionalidade que informa quando o aplicativo fica sem acesso ao servidor, seja por falha no servidor ou por problemas de conexão com a internet. Por fim, o APP também contempla a funcionalidade de armazenar e consumir dados em *cache* quando não há conexão com a internet disponível, tornando possível a visualização de algumas informações resumidas.

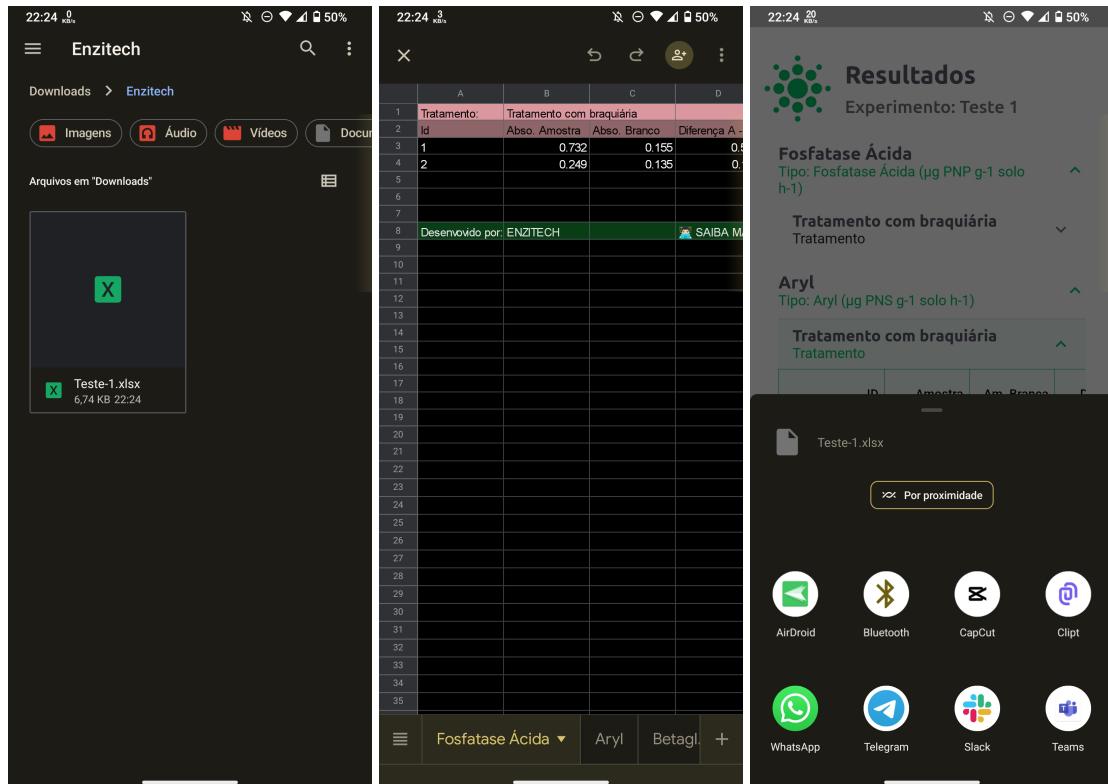
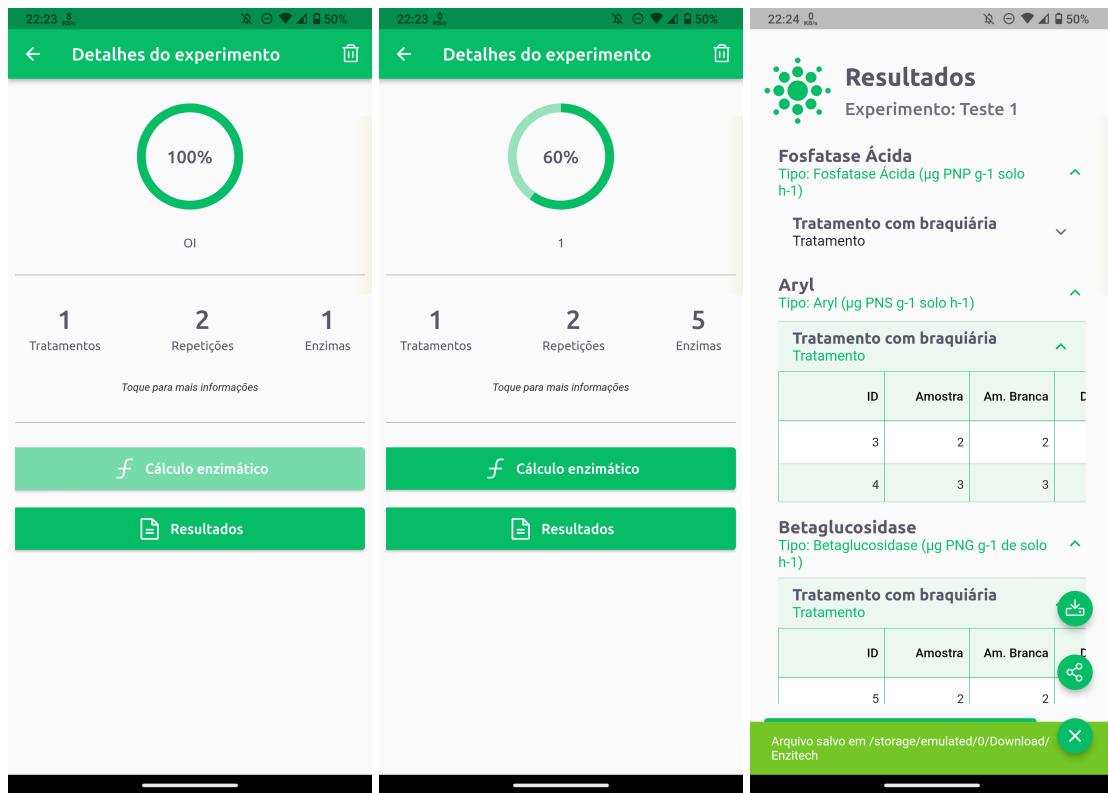


Figura 5.6: Fluxo de visualização e compartilhamento de resultados de um experimento
Fonte: Aplicativo Enzitech desenvolvido pelo autor

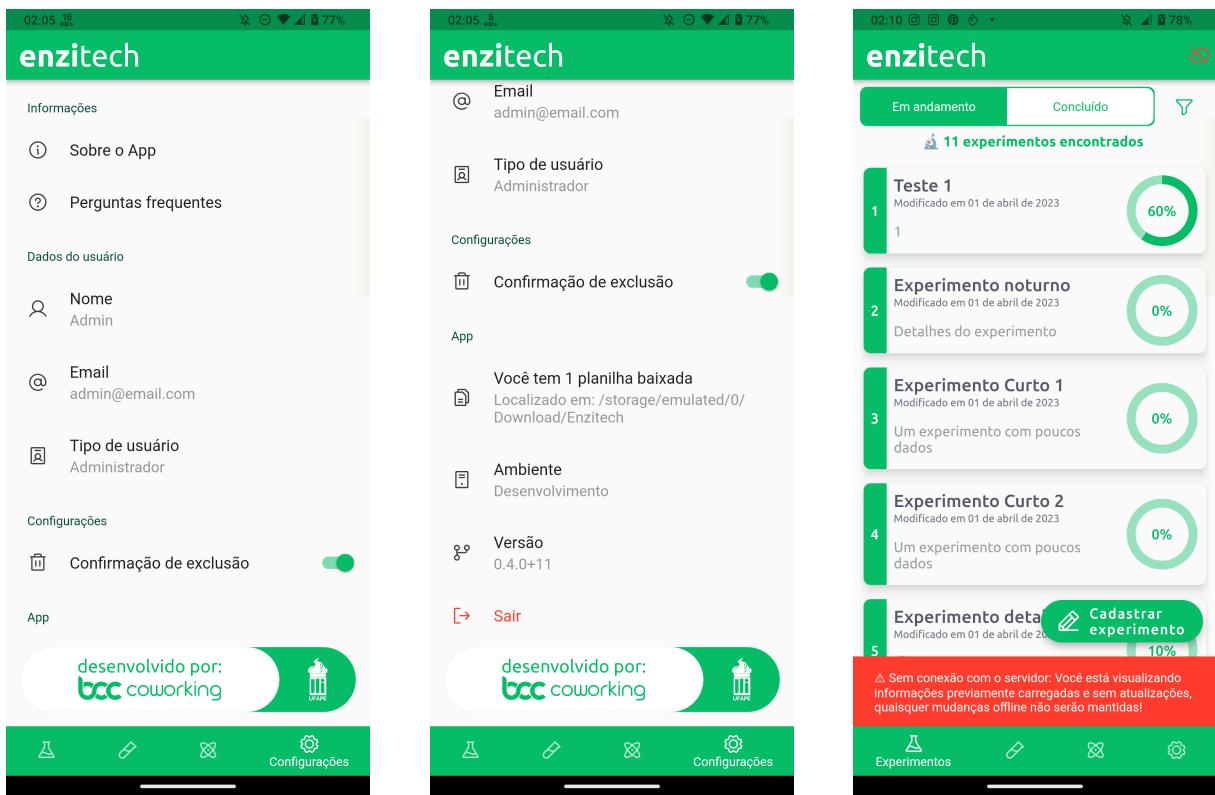


Figura 5.7: Tela de configurações do APP do Enzitech

Fonte: Aplicativo Enzitech desenvolvido pelo autor

6

Conclusão

Durante a realização deste estudo acadêmico, foram identificados e analisados desafios contemporâneos relacionados à qualidade do solo para a produção de alimentos no contexto do agronegócio brasileiro. Entre os aspectos destacados, destaca-se o uso de ferramentas para experimentos e análises do solo, juntamente com a inovação digital para gerenciar melhor a produção agrícola e, portanto, utilizar essas inovações como meio de preservação ambiental.

Nesse sentido, ao investigar os métodos de avaliação da qualidade do solo para o cultivo de culturas específicas, sugere-se a integração do desenvolvimento de software a um método de avaliação do solo que calcule os dados coletados nesses experimentos, buscando maior eficácia e praticidade, bem como uma solução prática para esses desafios, apoiando estudantes e pesquisadores do setor.

Em vista disso, o projeto denominado Enzitech visa apresentar uma solução prática para a administração dos experimentos realizados para análise e cálculo de atividades enzimáticas do solo, sendo, na prática, um sistema disponibilizado via APP *mobile*, o qual permite acompanhar todas as etapas da criação de tratamentos, enzimas, experimentos e seus resultados.

Uma das etapas mais importantes do projeto foi estabelecer a metodologia a ser seguida, incluindo quais processos seriam utilizados para desenvolver o software. Para isso, foram realizadas análises de requisitos em conjunto com os clientes e, em seguida, iniciou-se a criação dos modelos, por meio de protótipos, diagramas e planos, para orientar o desenvolvimento do projeto.

A ideia do sistema consistiu em criar modelos para análise e coleta de requisitos, além de documentos que seriam utilizados durante o desenvolvimento. Depois que o protótipo do sistema foi criado, foram elaborados os diagramas UML, bem como a modelagem de dados e o plano de gerenciamento, que servirá de referência tanto para o gerente do projeto quanto para guiar os desenvolvedores.

Os estudantes, cientistas e corpo docente de Agronomia da UFAPE, ao qual o sistema foi criado inicialmente para atender, buscam um melhor gerenciamento dos experimentos que dizem respeito às análises do solo, eliminando dezenas de planilhas, papéis e outros artefatos que podem ser facilmente alterados, ou preenchidos incorretamente, podendo gerar resultados que

atrapalhem os estudos daquele solo em questão, desta forma, esse produto tem a possibilidade de expandir seu alcance para inúmeras instituições que buscam otimizações nestes processos.

Por último, com base no processo de desenvolvimento, é importante ressaltar que a construção deste sistema e, portanto, deste trabalho, pode ser vista como um ponto de partida para novas investigações, a fim de acrescentar e contribuir para o conhecimento científico. Na seção, serão apresentadas sugestões para trabalhos futuros.

6.1 Principais contribuições

As principais contribuições com o desenvolvimento do Enzitech serão apresentadas a seguir:

- Organização de experimentos sobre o solo com base no usuário, permitindo a separação em categorias como "em andamento" e "concluídos", disponibilizando um acesso fácil e centralizado às informações.
- Redução de erros oriundos de dados inconformes.
- Fácil preenchimento e visualizações de dados relativos à cada experimento.
- Geração de resultados na forma tradicional em planilhas, para o acompanhamento e fácil compartilhamento das informações.
- Possibilidade de manuseio dos experimentos em ambientes com difícil acesso a dispositivos não-móveis, como um computador pessoal, por exemplo.

6.2 Principais limitações

As maiores dificuldades para desenvolvimento do sistema Enzitech, estão basicamente no tempo disponível para desenvolvimento, na infraestrutura disponibilizada para o desenvolvimento do sistema e na equipe reduzida, principalmente. Estão listadas a seguir as principais limitações:

- O sistema não lê planilhas já existentes e transforma em dados para o APP.
- O aplicativo não funciona 100% na arquitetura *offline-first*, que visa disponibilizar o acesso à todas as ferramentas com dados baixados previamente, possibilitando que o usuário envie as atualizações posteriormente.
- Existem limitações quanto ao SO Android, o qual desde a versão 11 (API Level 30) não permite o fácil gerenciamento das planilhas salvas no dispositivo, por medidas de segurança e privacidade¹, assim gerando possíveis erros (já tratados) caso o usuário mantenha planilhas baixadas, reinstale o aplicativo e tente baixá-las novamente.

¹ Atualizações de armazenamento no Android 11: <https://developer.android.com/about/versions/11/privacy/storage?hl=pt-br>

6.3 Trabalhos Futuros

Há alguns aspectos neste trabalho que podem ser melhorados, com base neste objetivo, serão apresentadas algumas sugestões para dar continuidade ao trabalho realizado:

- A realização de testes com usuários reais para validação pode abrir novas possibilidades de estudos na área de interface e experiência do usuário.
- Melhorar a exibição de resultados tanto no APP, quanto nas planilhas, utilizando mais dados que possam ser gerados e disponibilizados pela API.
- Um estudo profundo sobre o novo sistema de gerência de arquivos no SO Android pode solucionar a limitação previamente citada.
- A possibilidade de leitura de planilhas em determinado formato para a inserção destes dados no sistema Enzitech.
- Implementação de mais funcionalidades seguindo a arquitetura *offline-first*.
- Disponibilização do aplicativo para plataformas do SO iOS, da Apple, visto que o aplicativo foi desenvolvido utilizando o *framework* Flutter, que permite a geração do mesmo app para diversas plataformas, sendo necessário somente alguns ajustes específicos de cada SO.

Referências

- AHVANOOEY, M. T. et al. A survey on smartphones security: software vulnerabilities, malware, and attacks. **arXiv preprint arXiv:2001.09406**, [S.I.], 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.I.], 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.I.], v.21, p.1624, 02 2021.
- B'FAR, R. **Mobile computing principles:** designing and developing mobile applications with uml and xml. [S.I.]: 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.I.], v.167, p.110556, 2020.
- BORGES, J. A análise de usuários de smartphones: android versus ios. **Revista de Tecnologia da Informação**, [S.I.], 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.I.]: Pearson Education, 2003.
- FLUTTERANDO. **Clean Dart.** 2021.
- GLOBALSTATS, S. Mobile Operating System Market Share Brazil Statcounter Global Stats. **Retrieved June**, [S.I.], v.23, p.2020, 2020.
- GUHA, S.; BANERJEE, A. Flutter vs React Native: a comparative study. **Journal of Intelligent Systems**, [S.I.], v.29, n.2, p.255–270, 2020.
- INTELLIGENCE, M. **Smartphones market - growth, trends, covid-19 impact, and forecasts (2023 - 2028).** [S.I.]: Mordor Intelligence, 2023.
- KOTONYA, G.; SOMMERVILLE, I. **Requirements engineering:** processes and techniques. [S.I.]: John Wiley & Sons, 1998.
- LARICCHIA, F. **Number of mobile devices worldwide 2020-2025.** 2022.
- LEE, V.; SCHNEIDER, H.; SCHELL, R. **Aplicações móveis:** arquitetura, projeto e desenvolvimento. [S.I.]: Pearson Makron Books, 2005.

- LEITE, A. C.; MACEDO, H. COMPARATIVO ENTRE SISTEMAS OPERACIONAIS MÓVEIS–ANDROID X IOS. , [S.I.], 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.I.], 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.I.], v.9, p.122949–122965, 2021.
- MARTIN, R. C. **Principles of OOD**. 2005.
- MARTIN, R. C. **Código Limplo**: habilidades práticas do agile software. [S.I.]: Editora Bookman, 2008.
- MARTIN, R. C. **Agile software development**: principles, patterns, and practices. [S.I.]: Prentice Hall Professional, 2008.
- MARTIN, R. C. **Arquitetura Limpa**: o guia do artesão para estrutura e design de software. [S.I.]: Alta Books Editora, 2018.
- MYERS, G. J.; SMEDEMA, L.; TURBITT, T. **The Art of Software Testing**. [S.I.]: John Wiley & Sons, 2012.
- NASCIMENTO, E. L. et al. Impacto das tecnologias nas instituições de pesquisa científica em agronomia. **Revista Brasileira de Ciências Agrárias**, [S.I.], v.12, n.2, p.143–150, 2017.
- PRESSMAN, R. S. **Engenharia de Software**: uma abordagem profissional. 8.ed. [S.I.]: 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.I.], v.60, p.102634, 2021.
- SINSABAUGH, R. L. et al. Stoichiometry of soil enzyme activity at global scale. **Ecology Letters**, [S.I.], v.11, n.11, p.1252–1264, 2008.
- SOMERVILLE, I. **Engenharia de software**. 9.ed. [S.I.]: Pearson, 2015.
- SOMMERVILLE, I. **Software engineering**. [S.I.]: Pearson Education Limited, 2011.
- SUTHERLAND, J. **SCRUM**: a arte de fazer o dobro de trabalho na metade do tempo. [S.I.]: Leya, 2014.
- TABATABAI, M. A. Soil enzymes. In: **Methods of soil analysis**: part 2. microbiological and biochemical properties. [S.I.]: Soil Science Society of America, Inc, 1994. p.775–833.
- TABATABAI, M. A.; BREMNER, J. M. Use of p-nitrophenyl phosphate for assay of soil phosphatase activity. **Soil Biology and Biochemistry**, [S.I.], v.1, n.4, p.301–307, 1969.
- WARFEL, T. Z. **Prototyping**: a practitioner's guide. [S.I.]: Rosenfeld Media, 2009.
- ZHOU, M.; ZHANG, Y.; HUANG, L. Flutter vs. React Native vs. Native: a comparative study. **Sensors**, [S.I.], v.21, n.2, p.354, 2021.
- ÁGIL, M. **O que é Kanban?** 2017.

Apêndice

A

Códigos Flutter

A.1 Código da tela *Home Page* desenvolvida na Terceira Sprint

```
// 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>();
    initFragments();
    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),
            ),
        );
    }
}) ;

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

Widget? 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),  
    ),  
);  
}  
}
```

A.2 Trecho de código para realizar a checagem de autenticação e pré-carregamento dos dados desenvolvido na Sprint 6

```
_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,
                    ),
                );
        }
    });
}
```

B

Testes

B.1 Trecho de código do teste de unidade para enzima

```
main() {  
    test(  
        "Should not return null on Enzyme entity",  
        () {  
            EnzymeEntity enzyme = EnzymeEntity(  
                id: "1",  
                name: "Enzima",  
                type: "Tipo",  
                formula: "a+b=c",  
                variableA: 0.52,  
                variableB: 0.32,  
            );  
  
            expect(enzyme, isNotNull);  
        },  
    );  
}
```