UNIVERSIDADE ESTADUAL DA PARAÍBA

CAMPUS I

CENTRO DE CIÊNCIAS E TECNOLOGIAS - CCT

CURSO DE COMPUTAÇÃO

DIEGO VINÍCIUS LIMA FERREIRA

DESENVOLVIMENTO DE UM APLICATIVO MÓVEL PARA ACESSO A DADOS ACADÊMICOS DE UNIVERSITÁRIOS

CAMPINA GRANDE

2023

UNIVERSIDADE ESTADUAL DA PARAÍBA

CAMPUS I

CENTRO DE CIÊNCIAS E TECNOLOGIAS - CCT

DIEGO VINÍCIUS LIMA FERREIRA

DESENVOLVIMENTO DE UM APLICATIVO MÓVEL PARA ACESSO A DADOS ACADÊMICOS DE UNIVERSITÁRIOS

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dra. Kátia Elizabete Galdino.

CAMPINA GRANDE

2023

F383d Ferreira, Diego Vinicius Lima.
Desenvolvimento de um aplicativo móvel para acesso a dados acadêmicos de universitários [manuscrito] / Diego Vinicius Lima Ferreira. - 2023.
54 p. : il. colorido.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia, 2023.
"Orientação : Prof. Dr. Kátia Elizabete Galdino, Departamento de Computação - CCT. "

1. Usabilidade. 2. Aplicativos móveis. 3. Portal acadêmico. I. Título

21. ed. CDD 371.33

Elaborada por Geovani S. de Oliveira - CRB - 4/2314

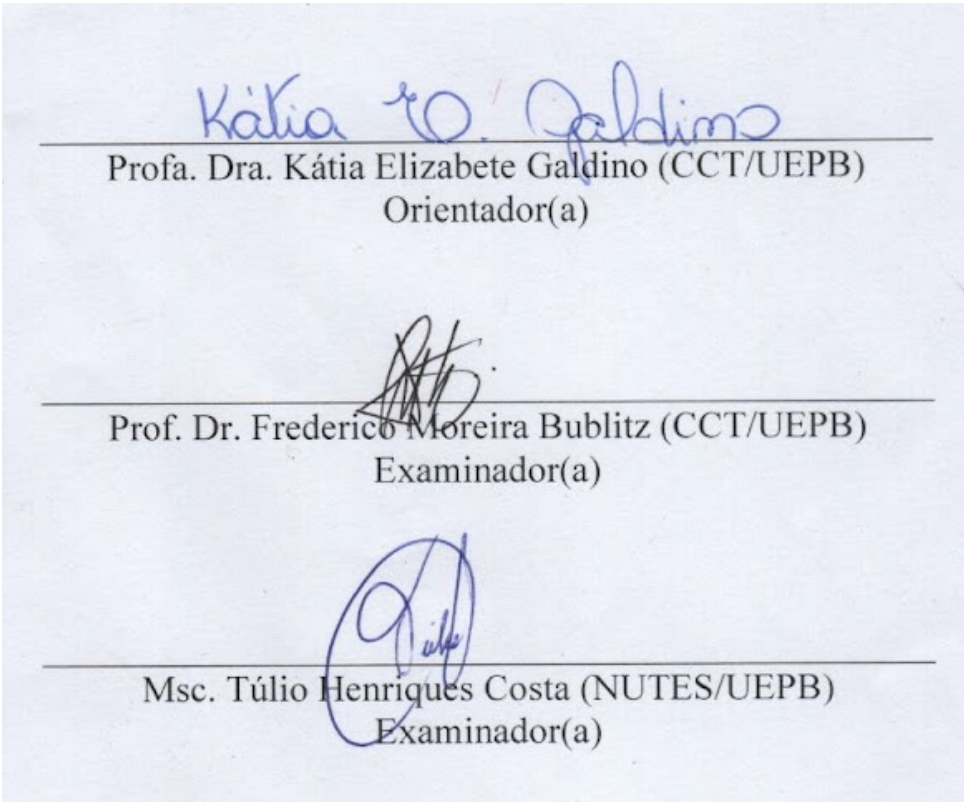
Biblioteca Central BC/UEPB

DIEGO VINÍCIUS LIMA FERREIRA

DESENVOLVIMENTO DE UM APLICATIVO MÓVEL PARA ACESSO A DADOS ACADÊMICOS DE UNIVERSITÁRIOS

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Ciência da Computação.

Aprovado em 23 de Agosto de 2023.



RESUMO

Este trabalho explora o desenvolvimento de um aplicativo móvel enquanto aborda as limitações de usabilidade e acessibilidade presentes nas interfaces dos portais acadêmicos universitários. O aplicativo tem como objetivo resolver as deficiências identificadas, tendo como foco principal a experiência do usuário. Para isso, a metodologia empregada consistiu em uma análise aprofundada da literatura, abrangendo princípios de arquitetura e padrões de projeto, além da avaliação de *design systems* e padrões de usabilidade. O intuito dessa abordagem é assegurar coesão tanto visual quanto funcional na aplicação. Após o levantamento teórico, os requisitos da aplicação foram definidos e as tecnologias que melhor se adequam a esses requisitos foram escolhidas. Com a finalização do desenvolvimento e, após validação interna, o *software* foi colocado em produção para ser avaliado por usuários reais. Com base no *feedback* recebido, iterações no projeto foram realizadas e planos para futuras melhorias foram estabelecidos.

Palavras-Chave: Aplicativo móvel; Usabilidade; Portais acadêmicos.

ABSTRACT

This study investigates the creation of a mobile app aimed at improving the usability and accessibility issues found in university academic portals. The app focuses on enhancing the user experience by addressing identified shortcomings. The research approach involved a thorough review of literature covering architectural principles, design patterns, and usability standards, including the assessment of design systems. This approach ensures both a consistent visual appearance and functional design. Following the theoretical exploration, application requirements were defined, and technologies best fitting these requirements were chosen. After the completion of initial development and internal validation, the software was released for real user testing. Feedback from users guided iterative project adjustments and laid out plans for future enhancements.

Keywords: Mobile application; Usability; Academic portals.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ASP	Atomic State Pattern
HIG	Human Interface Guidelines
HTTP	Hypertext Transfer Protocol
JSX	JavaScript XML
JWT	JSON Web Token
MVP	Minimum Viable Product
MVVM	Model-view-viewmodel
UI	User interface
RF	Requisitos Funcional
RNF	Requisitos Não Funcionais

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, os portais acadêmicos das universidades têm desempenhado um papel fundamental ao fornecer um meio eficaz para os estudantes acessarem informações relacionadas aos seus cursos, como horários, notas e documentação. No entanto, apesar dos benefícios oferecidos por essas plataformas online, elas frequentemente enfrentam desafios significativos em termos de usabilidade e acessibilidade. Barreiras tecnológicas, como interfaces complexas e falta de adaptação a diferentes dispositivos, resultam em dificuldades para os usuários ao acessar dados essenciais. Com isso, a experiência do usuário é afetada negativamente, resultando em frustrações, atrasos e perda de eficiência ao obter informações, como indica Hanchar em seu estudo sobre o impacto da complexidade visual na percepção estética e na impressão inicial de sites.

Tendo em vista os problemas encontrados em partes desses portais, mais especificamente nos objetos de estudos utilizados (UEPB e UFCG), surge a necessidade de uma abordagem mais eficiente e acessível para garantir que os usuários possam interagir de forma fluida e satisfatória com as plataformas acadêmicas. Nesse contexto, o desenvolvimento de uma aplicação móvel se mostra como uma solução promissora para enfrentar os desafios apresentados, dado que grande parte dos usuários acessam esses portais através de *smartphones*.

Diante deste cenário, o presente trabalho tem como propósito principal explorar e apresentar o desenvolvimento de uma aplicativo nativo para *smartphones.* Sendo uma aplicação nativa um *software* que é instalado no dispositivo do usuário, visto que esta abordagem pode oferecer recursos extras em comparação a um *website* que é executado em um navegador, como a exibição de notificações, por exemplo. Para alcançar esse objetivo, o estudo adotou uma abordagem que incluiu uma análise da literatura em busca de fundamentação para o desenvolvimento da aplicação levando em conta as boas práticas tanto no quesito estrutural como na criação da interface e experiência do usuário.

A pesquisa realizada buscou primeiramente identificar as abordagens para a criação de aplicativos e quais tecnologias se destacam para realização dessa tarefa. Após isso, foram analisadas padrões de arquitetura para aplicativos móveis com o objetivo de que o *software* criado oferecesse uma estrutura sólida e escalável. Seguidamente foram estudados os princípios para a criação da interface do usuário,

levando em consideração os dispositivos utilizados, padrões de uso e análise de projetos similares para entender como o mercado já resolve os problemas encontrados e tentar evoluir a partir disso.

Após a pesquisa realizada, foi feito o levantamento dos requisitos funcionais e não funcionais com o objetivo de determinar as funcionalidades do aplicativo e definir o seu escopo. Com os requisitos elencados, foi realizada a escolha das tecnologias utilizadas e que melhor se adequam às necessidades do projeto. Continuando com a etapa de planejamento, a arquitetura da aplicação foi definida junto com os componentes da interface e como funcionaria o fluxo de dados da aplicação.

A partir desse ponto, onde já foram definidos os objetivos do aplicativo e as ferramentas utilizadas, foi iniciado o processo de implementação. Durante essa etapa foram encontrados problemas no planejamento inicial, o que resultou em uma atualização nos requisitos. O processo de encontrar um possível problema, revisar a literatura e atualizar os requisitos foi muito presente durante o desenvolvimento do projeto, fazendo parte de uma abordagem incremental e iterativa que foi utilizada.

Ao alcançar um estágio considerável como estável durante validações e testes internos, o aplicativo foi colocado em produção para que fosse exposto a usuários reais a fim de se obter *feedback* e descobrir possíveis falhas na implementação. Com os dados obtidos, novas correções foram feitas e possíveis evoluções foram idealizadas.

Neste trabalho é apresentado todo o processo de desenvolvimento do aplicativo, desde sua idealização até o momento em que foi colocado em produção. As próximas seções abordam a literatura analisada, as decisões de projetos tomadas e, por fim, os resultados obtidos.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo serão abordados os principais temas que serviram de base para a realização deste trabalho e para o desenvolvimento do aplicativo.

2.1 Portais Acadêmicos

Os portais acadêmicos de universidades são plataformas digitais que desempenham um papel fundamental durante a vida acadêmica dos estudantes. Através desses portais os alunos têm acesso a recursos e informações essenciais para seu dia-a-dia como universitário, como horários e local das aulas, notas registradas pelos professores, calendários acadêmicos e demais registros educacionais. Ao disponibilizar esses dados de forma acessível e organizada, os estudantes podem acompanhar seu progresso acadêmico, planejar seus estudos e cumprir com os requisitos de seus cursos de maneira mais eficiente.

Além disso, muitos portais oferecem ferramentas adicionais para melhorar a experiência educacional. O que inclui fóruns de discussão, blogs e outras formas de interação social e colaborativa entre os membros da comunidade acadêmica. Essas funcionalidades promovem a troca de ideias e a colaboração entre os estudantes e professores.

Embora essas ferramentas tenham o objetivo de facilitar a vida acadêmica dos universitários, os usuários de parte desses portais enfrentam uma série de desafios em relação à usabilidade. Esses problemas, muitas vezes negligenciados, podem ter um impacto significativo na experiência do usuário e na eficiência geral do acesso às informações. Com o avanço das tendências de *design* e interação, os estudantes estão cada vez mais expostos a interfaces modernas, intuitivas e esteticamente agradáveis em outras plataformas digitais como as aplicações do Google e da Apple, e esperam ter a mesma experiência ao acessar o site de sua universidade. A seguir serão explorados alguns dos principais problemas de usabilidade enfrentados por portais acadêmicos atuais.

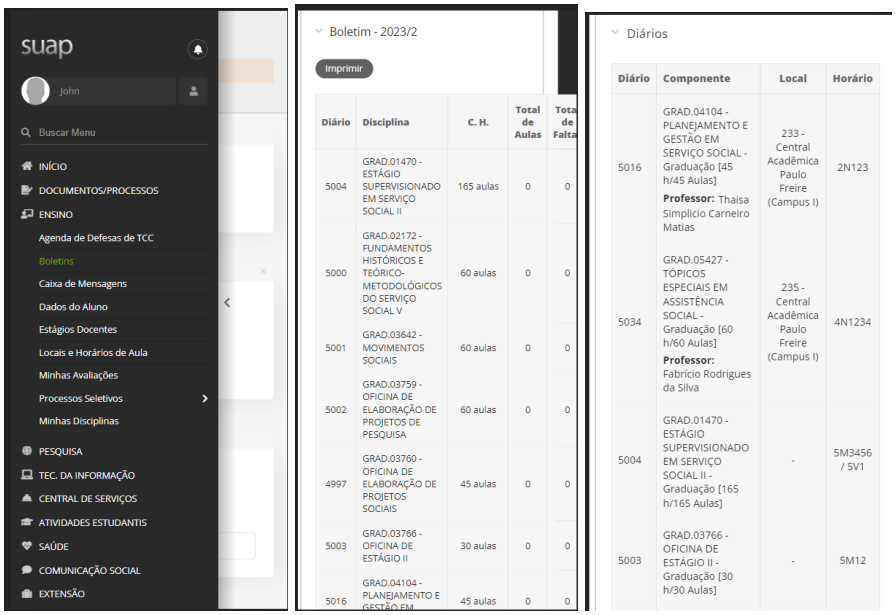
Um dos problemas recorrentes nesses portais diz respeito a falta de organização da interface do usuário. De acordo com a pesquisa de Hanchar (2012), a complexidade visual desempenha um papel fundamental na percepção estética

inicial de websites. Com uma grande quantidade de informações apresentadas de forma caótica, os usuários podem se sentir sobrecarregados e ter dificuldade em priorizar o conteúdo relevante para suas necessidades. Outro problema causado por essa desorganização é a complexidade de navegação, onde a falta de uma estrutura clara e intuitiva dificulta a localização de informações relevantes. Os menus complicados e a desorganização podem levar os usuários a percorrerem caminhos confusos para encontrar o que precisam.

Um quesito fundamental para a usabilidade e que afeta uma parcela da população acadêmica diz respeito a acessibilidade, questão crítica especialmente para estudantes com necessidades especiais. Segundo estudo realizado em 2017 pelo Movimento Web para Todos, onde foram analisados os sites das principais universidades do Brasil, os portais acadêmicos dessas instituições apresentavam barreiras tecnologias e/ou não estavam adequados às diretrizes de acessibilidade, o que pode dificultar o acesso às informações para pessoas com deficiência visual, auditiva ou outras limitações.

Outro problema comum enfrentado por portais acadêmicos é o uso de interfaces desatualizadas e que não refletem os padrões de *design* atuais, principalmente aqueles que não foram atualizados nos últimos anos. Muitos desses portais podem parecer obsoletos e pouco atraentes visualmente, o que afeta negativamente a experiência dos usuários. Em sua pesquisa, Hanchar (2012), apresenta dados que mostram a importância da beleza percebida em websites nas primeiras impressões do usuário e como a importância da interface aumenta ao passar do tempo em que o usuário utiliza a aplicação.

Figura 1 – Portal SUAP UEPB em dispositivos móveis



Fonte: Elaborado pelo autor.

Por fim, outro desafio significativo enfrentado pelos portais acadêmicos é a falta de responsividade em dispositivos móveis, como é mostrado no exemplo na Figura 1. Muitos desses portais não são otimizados para telas menores, o que resulta em uma experiência de usuário desconfortável e pouco prática em *smartphones*. A visualização inadequada de informações e a dificuldade de navegação podem desencorajar os estudantes a acessarem o portal por meio de seus dispositivos móveis, afetando a conveniência e a portabilidade que esses dispositivos oferecem.

2.2 Uso de Aplicativos

No Brasil, o uso de smartphones atingiu patamares impressionantes, transformando-os em uma ferramenta essencial para a vida cotidiana. Segundo pesquisa realizada pela Fundação Getúlio Vargas (2023), o Brasil tem em média mais de um smartphone por habitante. Com milhões de pessoas utilizando esses dispositivos para se conectar, aprender e acessar informações, adaptar-se a esse público é uma necessidade para qualquer plataforma *online*. Em um cenário em que a mobilidade é regra, universidades que investem em sites responsivos ou aplicativos nativos estão criando uma ponte direta com seus alunos, permitindo que eles acessem horários de aula, comunicados e materiais de estudo de maneira rápida e conveniente.

O desenvolvimento de um aplicativo nativo e exclusivo para os universitários é uma abordagem que vem se popularizando recentemente. Afinal, a geração de jovens, maioria do público que frequenta as instituições de ensino (Mapa do Ensino Superior, 2023) está familiarizada com o uso de aplicativos em seus smartphones. Essa abordagem oferece benefícios significativos em relação a um site convencional. Aplicativos podem proporcionar notificações instantâneas sobre atualizações importantes, uma interface mais amigável e funcionalidades otimizadas para a experiência móvel. Além disso, um aplicativo personalizado pode unir estudantes em um único espaço, fomentando o sentimento de comunidade e engajamento.

Ao direcionar os esforços para desenvolver um aplicativo agregador com foco na usabilidade, acessibilidade e responsividade em dispositivos móveis e com uma interface atraente, as instituições de ensino podem atender às expectativas desse público-alvo e promover um ambiente acadêmico mais eficiente e envolvente.

2.3 Desenvolvimento de Aplicativos

No contexto do desenvolvimento de aplicativos móveis, existem diferentes abordagens que os desenvolvedores podem adotar, cada uma com suas características distintas. Duas abordagens principais têm se destacado nesse contexto: o desenvolvimento nativo e o desenvolvimento multiplataforma (Abdal et al., 2013). A seguir serão exploradas em detalhes essas duas abordagens distintas de desenvolvimento, analisando suas características, vantagens e desafios.

Até esse momento do trabalho, o termo nativo foi utilizado para descrever um aplicativo que é instalado no smartphone do usuário, em oposição a um website que é acessado através do navegador. A partir de agora, o termo será utilizado para descrever um aplicativo desenvolvido para uma plataforma específica, em oposição ao desenvolvimento híbrido ou multiplataforma.

2.3.1 Desenvolvimento Nativo

O desenvolvimento nativo é a abordagem mais tradicional e amplamente utilizada para criar aplicativos móveis. Nesta abordagem, os aplicativos são projetados e desenvolvidos exclusivamente para uma plataforma específica, como iOS ou Android, utilizando as linguagens de programação e as ferramentas nativas de cada sistema operacional.

Uma de suas principais características é a alta integração com a plataforma. Isso significa que os aplicativos nativos são capazes de aproveitar totalmente os recursos e funcionalidades ofertadas do dispositivo em que estão sendo executados. Essa conexão direta com o sistema operacional permite que os aplicativos ofereçam um desempenho otimizado, uma experiência de usuário fluida e uma interface que harmoniza com a linguagem de *design* da plataforma.

No desenvolvimento nativo para Android, as linguagens de programação usadas são Java e Kotlin, enquanto que para iOS as linguagens Swift e Objective-C são as escolhas disponíveis. Essas linguagens são amplamente suportadas por suas plataformas (Abdal et al., 2013), e os desenvolvedores têm acesso a todas APIs nativas para criar aplicativos com funcionalidades avançadas e personalizadas, além de melhor acesso a recursos de *hardware*, como a câmera, o *GPS* e os sensores do dispositivo. Isso é especialmente relevante em aplicativos que exigem

um alto nível de interação com o *hardware*, como aplicativos de realidade aumentada, jogos e aplicativos de saúde e *fitness*.

Apesar de todos esses benefícios, o desenvolvimento nativo possui algumas desvantagens para determinantes tipos de projeto, principalmente aqueles que buscam atender aos dois sistemas operacionais. Uma delas é o custo e o esforço duplicados, já que é necessário desenvolver e manter códigos separados para cada plataforma que se pretende alcançar. Além disso, a possível necessidade de ter equipes de desenvolvimento especializadas para cada plataforma é um desafio, principalmente para pequenas empresas e *startups* com recursos limitados. Essa abordagem também pode necessitar de mais tempo para o desenvolvimento e a implementação de atualizações, dado que é necessário manter uma paridade de duas plataformas com estruturas diferentes.

Contudo, o desenvolvimento nativo continua sendo a escolha mais popular para aplicativos que demandam o máximo de desempenho, interação com *hardware* e *APIs* específicas de cada plataforma. É uma abordagem que oferece a possibilidade da criação de aplicativos sob medida para cada plataforma e, em geral, resultam em aplicações de alta qualidade com uma experiência refinada para cada ecossistema.

2.3.2 Desenvolvimento Cross-Platform

O desenvolvimento *cross-platform* tem como objetivo criar aplicativos que sejam compatíveis com múltiplas plataformas e com diferentes sistemas operacionais e arquiteturas. Essa abordagem visa oferecer aos desenvolvedores uma maneira de atender diferentes ecossistemas, sem a necessidade de reescrever o código para cada plataforma específica.

Ao optar por esse caminho os desenvolvedores podem atingir um número maior de usuários, independentemente dos dispositivos que utilizam. E com o fato de que é necessário manter apenas uma base de código para várias plataformas, o tempo e custo de desenvolvimento podem ser consideravelmente reduzidos, dado que com um código-fonte unificado, a manutenção do aplicativo torna-se mais simples e eficiente. Atualizações e correções podem ser implantadas de forma consistente em todas as plataformas alvo, facilitando o gerenciamento contínuo do

aplicativo. Isso é relevante principalmente para equipes pequenas e com recursos limitados.

Embora essas características parecem tornar essa a escolha ideal, não são todos os projetos que se adequam a essa abordagem. Uma das principais desvantagens é o acesso restrito a recursos nativos, prejudicando a capacidade de aproveitar totalmente funcionalidades específicas de cada plataforma. Além disso, o desempenho pode ser comprometido devido ao uso de camadas de abstração que impactam a performance e eficiência do aplicativo em comparação com aplicativos nativos.

Outro fator negativo é que em algum momento pode ser necessário acessar alguma funcionalidade do sistema operacional ou integrar com uma ferramenta de terceiros onde, provavelmente, será preciso realizar uma comunicação utilizando a linguagem nativa de cada plataforma como Kotlin ou Swift. Já no contexto do consumidor final, um aplicativo que não respeita o ecossistema, interface e usabilidade da plataforma em que está sendo executado é muito mal visto, sendo necessário ter um cuidado com a experiência oferecida aos usuários (Robinson, 2016).

A seguir são apresentados dois dos principais *frameworks* utilizados para o desenvolvimento *cross-platform* baseado em utilização e tamanho da comunidade de desenvolvedores segundo dados da pesquisa anual realizada pelo site Stack Overflow. Um *framework* é uma estrutura de desenvolvimento de *software* que fornece componentes e diretrizes pré-definidas para facilitar a criação e o desenvolvimento de aplicativos, promovendo a reutilização de código e a implementação de padrões de *design*.

- Flutter é um *framework* de desenvolvimento *cross-platform* criado pelo Google, lançado em 2017, que tem ganhado destaque no cenário de desenvolvimento de aplicativos. O principal diferencial oferecido pelo Flutter é a capacidade de criação de aplicativos para *Android*, *IOS*, *Web* e *Desktop* com uma única linguagem de programação. O *framework* utiliza a linguagem de programação Dart, também desenvolvida pelo Google, que se destaca por sua performance e capacidade de gerar binários para as principais arquiteturas como ARM e X86, além de gerar código JavaScript para ser executado em navegadores.

Uma das principais características técnicas do Flutter é o *hot reload*, um recurso que permite aos desenvolvedores verem as alterações no código imediatamente refletidas no aplicativo em execução, sem a necessidade de recompilar o aplicativo inteiro, o que acelera o processo de desenvolvimento e prototipação do projeto.

O Flutter é projetado para atender diferentes dispositivos com diferentes tamanhos de tela, mantendo uma interface adaptada em todas as plataformas, oferecendo suporte a gestos, *mouse* ou teclas físicas. Para isso o *framework* oferece componentes de interface pré configurados com suporte ao Material Design, o sistema de *design* do Google, e ao Cupertino, o sistema de *design* da Apple. Isso contribui para criar uma experiência familiar nas plataformas que executam o aplicativo, mas caso necessário os desenvolvedores podem criar seus próprios componentes e linguagem de *design* para atender seus requisitos.

Empresas de renome, como Alibaba, Google e Tencent, são exemplos de organizações que utilizam o Flutter para desenvolver aplicativos de alto desempenho e com grande base de usuários, o que traz uma credibilidade para o *framework*. Além disso, o Flutter conta com uma comunidade ativa e crescente, o que resulta em um rico ecossistema de pacotes de ferramentas e *plugins* para interação com recursos nativos, proporcionando aos desenvolvedores uma melhor experiência de desenvolvimento.

Devido a suas características técnicas, baixa curva de aprendizado e sua crescente adoção pela comunidade de desenvolvedores, o Flutter tem se destacado como uma das principais opções para projetos *cross-platform* que visam oferecer aplicativos de alta qualidade e excelente experiência de usuário em diversas plataformas.

- React Native é um *framework* de desenvolvimento móvel criado pelo Facebook que permite a criação de aplicativos para iOS e Android utilizando uma única base de código em JavaScript que é traduzida para código nativo em tempo de execução para cada plataforma. A ideia central por trás do React Native é possibilitar que os desenvolvedores criem aplicativos com uma experiência nativa, mas de forma mais rápida e eficiente do que o desenvolvimento tradicional para cada plataforma separadamente.

Uma das principais características do React Native é a utilização de componentes reutilizáveis. O código é dividido em pequenos componentes, que podem ser compartilhados e reaproveitados em diferentes partes do aplicativo. Isso torna o desenvolvimento mais organizado e permite que a equipe trabalhe de forma mais colaborativa, economizando tempo e esforço. Outro ponto forte é a sua abordagem declarativa para a criação de interfaces.

Com a utilização do *JSX*, os desenvolvedores podem descrever como a interface do aplicativo deve ser renderizada de forma mais intuitiva e concisa, o que facilita a manutenção do código.

O React Native possui diversos casos de uso, desde aplicativos corporativos até aplicativos de mídia social e jogos. Um exemplo famoso de aplicativo desenvolvido em React Native é o Facebook. Outras empresas conhecidas, como Instagram, Pinterest e Uber Eats, também utilizam o React Native em seus aplicativos.

Esse *framework* mostra-se como uma abordagem promissora para projetos que visam ambas as plataformas Android e IOS, mas que preferem manter uma única base de dados. Sendo o React Native ainda mais recomendado caso os desenvolvedores tenham experiência com JavaScript e React.

2.4 Arquitetura de Aplicativos Móveis

Segundo Perry e Wolf (1992), a arquitetura de *software* é a organização fundamental de um sistema, expressa por suas principais estruturas, seus componentes, as relações entre eles e as restrições que governam o seu *design* e evolução ao longo do tempo. No contexto de aplicações móveis, é a estrutura de alto nível que define como os diversos componentes do aplicativo, como a interface do usuário, lógica de negócio e acesso a dados, são organizados e interagem entre si para fornecer funcionalidades específicas aos usuários.

A arquitetura de um *software* desempenha um papel crucial no desenvolvimento de sistemas bem sucedidos. De acordo com Garlan e Shaw (1994), uma arquitetura bem definida pode trazer inúmeros benefícios para o desenvolvimento do *software*. Um dos principais benefícios é a separação de responsabilidades e reutilização de componentes, permitindo que partes do sistema

sejam usadas em diferentes contextos reduzindo duplicação de código. Além disso, uma arquitetura clara e organizada facilita a identificação e correção de problemas, o que agiliza a manutenção do sistema e torna o código mais sustentável a longo prazo.

Outro benefício importante da arquitetura de um *software* é a escalabilidade, dado que uma arquitetura bem planejada permite que o sistema cresça e se adapte facilmente a novas demandas e requisitos, tornando-o mais resiliente a mudanças. Além disso, a adoção de uma arquitetura apropriada ao escopo do projeto facilita a implementação de testes automatizados, o que traz maior qualidade ao *software* e auxilia na detecção precoce de problemas ainda durante o desenvolvimento.

No entanto, o tema de arquitetura de *software* também apresenta desafios e dificuldades. Conforme observado por Garlan e Shaw (1994), uma arquitetura mal projetada pode levar a uma complexidade desnecessária no projeto. A falta de clareza na estrutura ou abstrações demasiadas podem dificultar a compreensão do código, tornando a manutenção uma tarefa árdua.

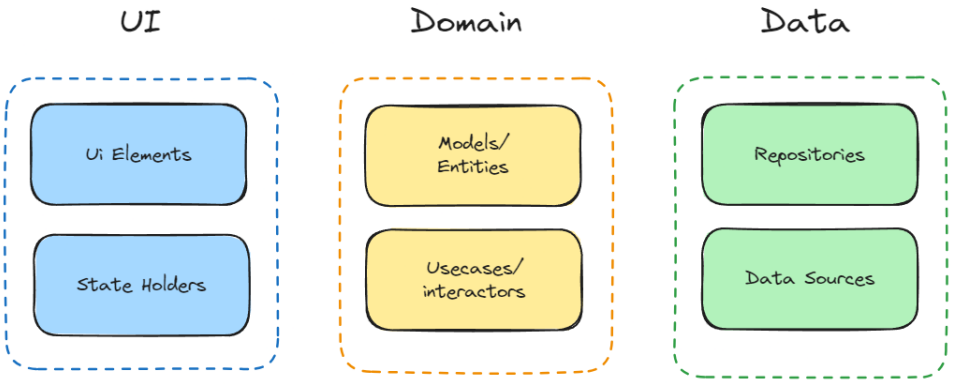
Outra dificuldade enfrentada em relação à arquitetura de *software* é a falta de flexibilidade. Em alguns casos, uma arquitetura pode ser excessivamente rígida e difícil de ser modificada para atender a novos requisitos. Isso pode limitar a capacidade do sistema de evoluir ao longo do tempo, podendo gerar altos custos de manutenção e adaptação.

Apesar dos desafios, a arquitetura de um *software* permanece como um elemento crítico para o sucesso dos projetos em desenvolvimento. Ao adotar uma abordagem cuidadosa e balanceada na definição da arquitetura, é possível aproveitar os benefícios mencionados anteriormente e superar as dificuldades. Garlan e Shaw (1994) ressaltam que a compreensão adequada dos princípios da arquitetura e a aplicação de padrões arquiteturais bem estabelecidos são fundamentais para maximizar os benefícios e minimizar as dificuldades no contexto da engenharia de software.

Uma arquitetura sólida é especialmente importante em aplicativos móveis devido aos desafios que essas plataformas apresentam, como recursos limitados, variação no desempenho do dispositivo e disponibilidade de rede.

As abordagens de arquiteturas modernas focadas em aplicativos móveis adotam uma estrutura reativa e em camadas. A Figura 3 exibe uma estrutura para arquitetura recomendada pela equipe de desenvolvimento do Android.

Figura 3 – Camadas de uma arquitetura e seus componentes



Fonte: Elaborado pelo autor.

Com o uso dessa estrutura o aplicativo é dividido em camadas distintas, cada uma com sua própria responsabilidade bem definida. As camadas se comunicam entre si de forma assíncrona, permitindo que o aplicativo reaja rapidamente às mudanças de estado e eventos. Essa abordagem facilita a escalabilidade e manutenção do aplicativo, tornando-o capaz de lidar com uma grande quantidade de dados e interações do usuário. A seguir são apresentadas as principais camadas que compõem a estrutura recomendada para o desenvolvimento de aplicações móveis.

2.4.1 UI

A camada de UI (User Interface) ou *Presentation* é responsável por descrever a experiência do usuário e as interações com o aplicativo. Essa camada é projetada para exibir visualmente os elementos da interface e responder às ações do usuário.

Uma abordagem moderna para a camada de UI inclui o código responsável por desenhar a tela do usuário e o uso de uma abordagem para gerenciamento de estado (dados da aplicação), como no caso do *Model View Viewmodel* (MVVM), onde o *Model* representa os dados da aplicação, a *View* descreve a interface e o *Viewmodel* faz a intermediação entre as duas partes. Esse padrão de interface permite que os dados sejam observados e atualizados de forma reativa, garantindo que a UI reflita automaticamente as mudanças de estado do aplicativo.

A separação clara entre a lógica de apresentação e os dados subjacentes é uma característica importante da camada de UI, permitindo que a interface do usuário permaneça independente dos detalhes da implementação da lógica de negócios e da camada de gerenciamento de dados. Isso torna o código mais organizado e fácil de manter, evitando a mistura de responsabilidades e melhorando a escalabilidade do aplicativo.

2.4.2 Domain

A camada *Domain* (ou *Business Logic*) é o coração da arquitetura de aplicativos móveis e que define as regras de negócio e implementa a lógica do aplicativo. Nesta camada, a funcionalidade central do aplicativo é definida de forma independente das preocupações técnicas ou da interface do usuário.

Uma abordagem moderna para a camada de domínio enfatiza a separação clara de responsabilidades e a modularidade. As regras de negócio são definidas em casos de uso ou *interactors*, que encapsulam as operações específicas que o aplicativo pode realizar. Cada caso de uso é uma unidade lógica e independente, facilitando o reuso e a manutenção do código.

Ao utilizar um padrão como *Unidirectional Data Flow*, a camada de domínio garante que a lógica de negócio seja tratada de forma previsível e transparente. As ações iniciadas pela camada de UI resultam em eventos ou comandos que são processados pela camada de domínio, que atualiza o estado do aplicativo de acordo com o resultado dos eventos.

Outro aspecto importante desta camada é a realização de testes. Ao manter a lógica de negócio desacoplada da interface do usuário e da camada de dados, os casos de uso podem ser facilmente testados de forma isolada, garantindo a integridade e a corretude das regras de negócio. É também nesta camada onde os modelos de dados, ou entidades, são definidos. Essas entidades representam os principais conceitos do domínio do aplicativo e refletem os objetos e suas interações no mundo real.

2.4.3 Data Layer

A **Data Layer** é responsável por gerenciar o acesso aos dados, gerenciando a persistência das informações e a comunicação com fontes externas. Para organizar de forma eficiente essa camada, são empregados conceitos como *Repositories* e *Data Sources*.

Repositories atuam como uma abstração entre a camada de negócios e as fontes de dados subjacentes, como bancos de dados locais ou serviços da web. Eles encapsulam a lógica para recuperar, armazenar e manipular dados e fornecem uma interface consistente para a camada de domínio do aplicativo. Por exemplo, ao buscar dados de um banco de dados local, o repositório será responsável por realizar a consulta e retornar os resultados para a camada de domínio. Se necessário, ele também deverá tratar os dados obtidos antes de entregá-los.

Por outro lado, os ***Data Sources*** são responsáveis pela implementação concreta das operações de acesso aos dados. Existem dois tipos principais de Data Sources: *Local Data Sources* e *Remote Data Sources*.

Os *Local Data Sources* interagem com bancos de dados internos, caches ou arquivos locais para buscar ou armazenar dados. Eles isolam os detalhes específicos do armazenamento de dados do resto do aplicativo e fornecem uma interface consistente para o Repositório. *Remote Data Sources*, por sua vez, são responsáveis por fazer chamadas de rede para recuperar dados de serviços remotos. Eles abstraem os detalhes de comunicação e formatação dos dados, tornando essa tarefa transparente para o Repositório.

A injeção de dependências (DI) é uma prática comum na construção da camada de dados para garantir a flexibilidade e a facilidade de testes. Através da DI, os Repositórios e *Data Sources* são injetados nas classes que deles dependem, em vez de serem instanciados diretamente. Isso permite que a configuração da Data Layer seja substituída por diferentes implementações durante a execução do aplicativo ou nos testes automatizados. Com o uso de DI, é possível trocar facilmente entre implementações locais e remotas dos Data Sources, facilitando a transição de armazenamento de dados e a simulação de cenários específicos nos testes unitários (Martin, 2008).

2.5 Interface e Experiência do Usuário

Durante a criação de uma interface para um aplicativo, surge a questão de utilizar ou não um *design system* (Fessenden, 2021). Um *design system* é um conjunto de componentes, diretrizes e princípios de *design* que visam fornecer consistência e eficiência na criação de interfaces. A decisão de usar um *design system* depende do escopo e das necessidades do projeto.

A adoção de um padrão de *design* durante o projeto oferece diversos benefícios significativos. Em primeiro lugar, proporciona consistência visual e funcional em toda a interface do aplicativo, resultando em uma experiência de usuário mais coesa. Além disso, um *design system* promove a eficiência no desenvolvimento, pois oferece um conjunto de componentes prontos para uso, poupando tempo e esforço na criação de elementos repetitivos. Outro benefício é a facilidade de manutenção, uma vez que qualquer atualização ou ajuste pode ser aplicado de forma centralizada em todo o sistema, refletindo automaticamente em todas as partes do aplicativo.

Por outro lado, a ausência de padrões consistentes pode resultar em uma interface fragmentada e confusa para os usuários. Além disso, a falta de um conjunto de componentes e diretrizes pode levar a uma abordagem inconsistente na criação de elementos de *design*, resultando em uma aparência desalinhada e dificultando a manutenção no longo prazo. Sem um *design system*, a equipe de *design* e desenvolvimento pode enfrentar desafios adicionais na garantia de uma experiência de usuário coesa e na eficiência do trabalho realizado.

Existem três alternativas para a implantação de um *design system* (Fessenden, 2021). A primeira opção é utilizar um *design system* existente, onde essa implementação já passou por um processo de pesquisa e desenvolvimento, garantindo a coerência visual e funcional dos componentes. Além disso, ele pode fornecer diretrizes claras para a criação de uma interface intuitiva e familiar aos usuários. Utilizar um *design system* existente também economiza tempo e esforço, pois grande parte do trabalho de pesquisa e desenvolvimento já foi realizado.

Outra possível abordagem é adaptar um *design system* existente. Isso envolve a personalização de componentes, cores e estilos para refletir a identidade da marca e as necessidades específicas do aplicativo. A vantagem dessa abordagem é a capacidade de aproveitar a base sólida de algo existente, mantendo

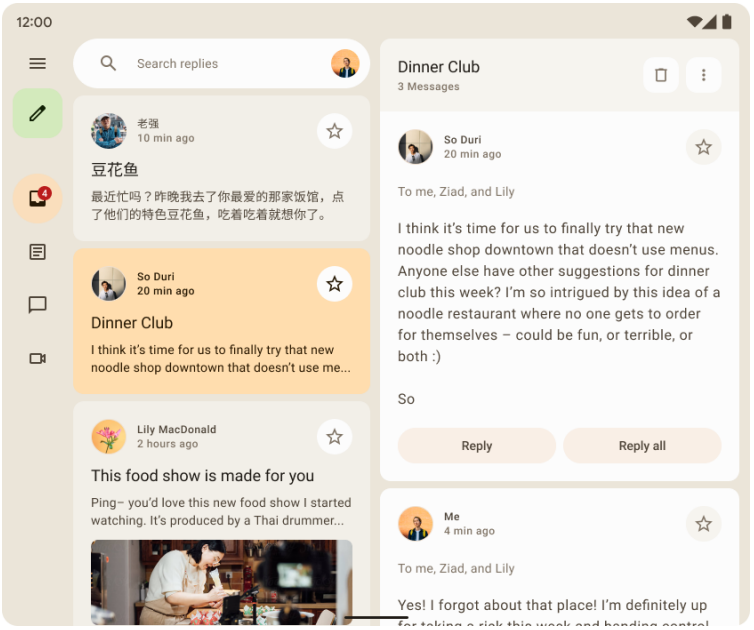
a consistência e a eficiência, e aplicar a originalidade de sua marca. Adaptar um *design system* requer uma análise cuidadosa dos componentes e diretrizes existentes, garantindo que eles atendam aos requisitos do projeto.

Por fim, também é possível criar um *design system* do zero. Essa abordagem oferece total liberdade criativa para desenvolver um sistema de *design* personalizado e exclusivo para o aplicativo. No entanto, criar uma linguagem de *design* requer um esforço significativo em pesquisa, planejamento e desenvolvimento. É importante garantir que o *design system* criado seja consistente e bem documentado, facilitando sua utilização ao longo do projeto (Babich, 2018).

Quando se trata de aplicativos móveis, o uso de um design system estabelecido, como os apresentados a seguir, pode ser particularmente benéfico.

- O Material Design foi desenvolvido pelo Google e é amplamente adotado na indústria de aplicativos móveis. Ele oferece uma linguagem de *design* consistente para dispositivos Android, garantindo uma experiência familiar e intuitiva para os usuários. O Material Design também abrange uma variedade de componentes e diretrizes específicas para aplicativos móveis, como tipografia, cores, formas e animações. Ao seguir o Material Design, é possível se beneficiar de uma interface que se alinha com as expectativas dos usuários e proporciona uma experiência de usuário coesa em todo o ecossistema principalmente em plataformas Android. A Figura 4 exibe uma interface com uma estrutura comum ao se utilizar o Material Design.

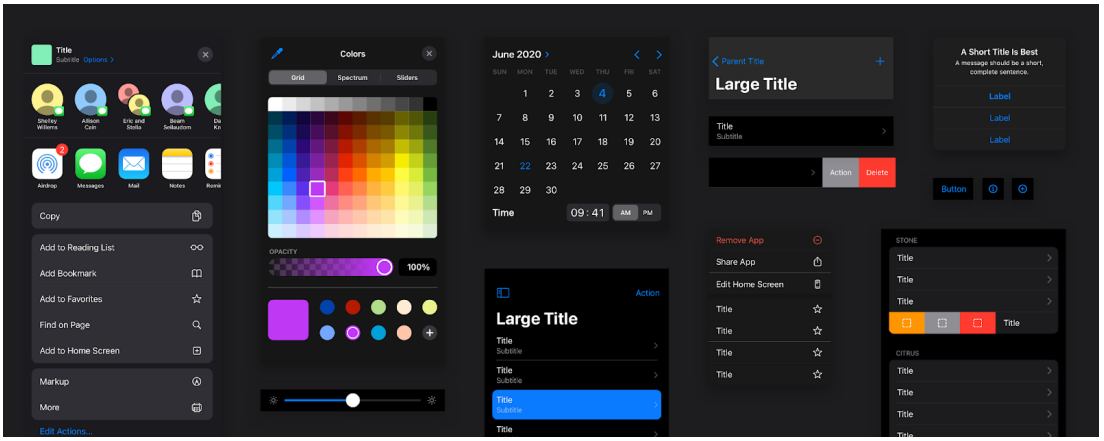
Figura 2 – Componentes do Material Design 3



Fonte: Material Design, 2023.

- O Human Interface Guidelines (HIG) é o *design system* da Apple, desenvolvido para fornecer diretrizes e padrões de *design* para a criação de interfaces de usuário em aplicativos iOS, macOS, watchOS e tvOS. O HIG é amplamente reconhecido por sua ênfase na clareza, simplicidade e usabilidade. Ele oferece orientações detalhadas sobre elementos de interface, como botões, barras de navegação, tipografia, ícones e gestos. Além disso, o HIG aborda princípios de *design*, comportamentos esperados e interações específicas para cada plataforma. Ao seguir o HIG, os *designers* e desenvolvedores podem criar interfaces coesas e intuitivas que se integram perfeitamente ao ecossistema da Apple, proporcionando uma experiência consistente e de alta qualidade para os usuários. A Figura 6 exibe os componentes que podem ser utilizados para construir a interface de aplicações que visam plataformas Apple.

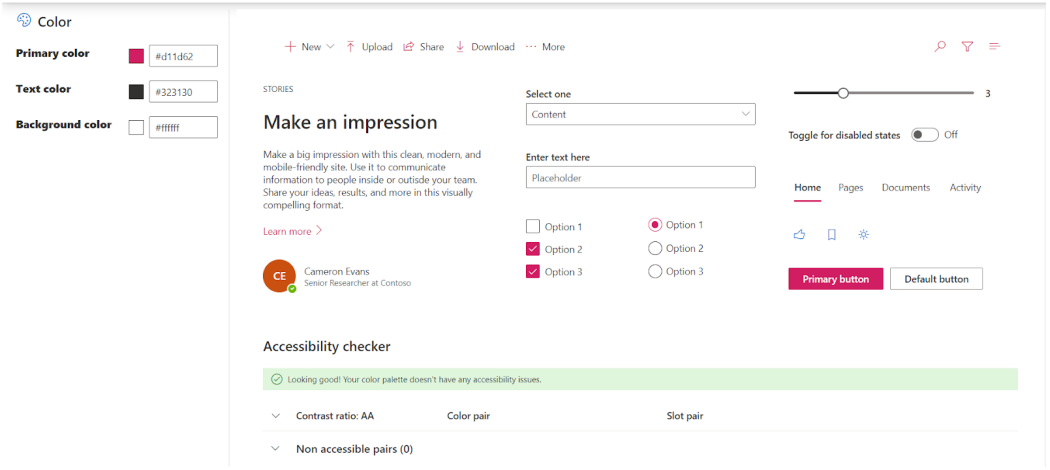
Figura 3 – Componentes do HIG para IOS



Fonte: Apple, 2023.

- O Fluent Design System é o *design system* da Microsoft, destinado a fornecer diretrizes e recursos de *design* para a criação de interfaces em aplicativos multiplataformas, mas que é amplamente reconhecido e utilizado no sistema Windows da própria Microsoft. O Fluent Design System baseia-se em cinco princípios de *design*: luz, profundidade, movimento, materiais e escala. Ele oferece uma abordagem moderna e envolvente, com foco em transições suaves, animações e efeitos visuais sutis. O Fluent Design System inclui uma variedade de componentes e estilos visuais (Figura 6), como acrílicos, sombras, ícones e tipografia, que ajudam a criar interfaces atraentes e funcionais. Ao seguir o Fluent Design System, os *designers* podem criar experiências imersivas e interativas em aplicativos Windows, proporcionando uma experiência coesa e visualmente rica para os usuários.

Figura 4 – Componentes do Fluent Design 2



Fonte: Microsoft, 2023.

É importante destacar a importância de o *design* se adequar à plataforma utilizada. Cada plataforma (como Android, iOS ou Windows) possui suas próprias diretrizes de *design* e padrões de interação. Ao aderir a essas diretrizes, você cria uma experiência mais consistente e familiar para os usuários, tornando o uso do seu aplicativo mais intuitivo.

Adaptar o *design* à plataforma escolhida também ajuda a garantir que o aplicativo se beneficie das convenções e funcionalidades específicas da plataforma. Isso pode incluir gestos de navegação, recursos de acessibilidade, integração com notificações do sistema e muito mais. Além disso, seguir as diretrizes de *design* da plataforma pode aumentar a aceitação do aplicativo nas lojas de aplicativos, pois as plataformas tendem a favorecer aplicativos que seguem suas diretrizes.

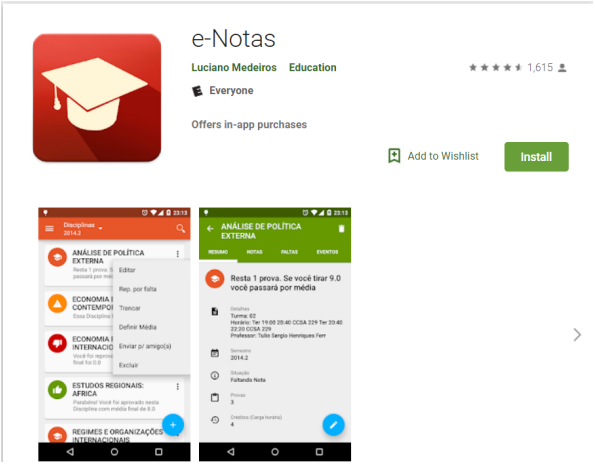
Em última análise, a escolha entre utilizar um padrão de *design* estabelecido ou criar um *design* original depende do contexto do projeto, dos objetivos do aplicativo e das preferências da equipe de *design*. Ambas as abordagens têm seus méritos e desafios. É importante considerar fatores como tempo, recursos disponíveis, identidade da marca, público-alvo e familiaridade dos usuários ao tomar essa decisão (Babich, 2018). Independentemente da abordagem escolhida, é essencial realizar testes de usabilidade e iterar continuamente o *design* para garantir uma experiência de usuário eficaz e satisfatória.

2.6 Projetos Relacionados

Nesta seção são apresentados alguns dos principais projetos com objetivos semelhantes ao aplicativo desenvolvido durante este trabalho. A análise de outros projetos é utilizada para validar a existência do aplicativo, buscar inspirações, aprender com erros e limitações de projetos anteriores, além de aprimorar e evoluir o que já existe no mercado. Essa abordagem enriquece o desenvolvimento do aplicativo, tornando-o mais sólido, inovador e capaz de atender eficazmente às necessidades dos usuários.

2.6.1 e-Notas

Figura 5 – Página de download do aplicativo e-Notas



Fonte: Internet Archive, 2019.

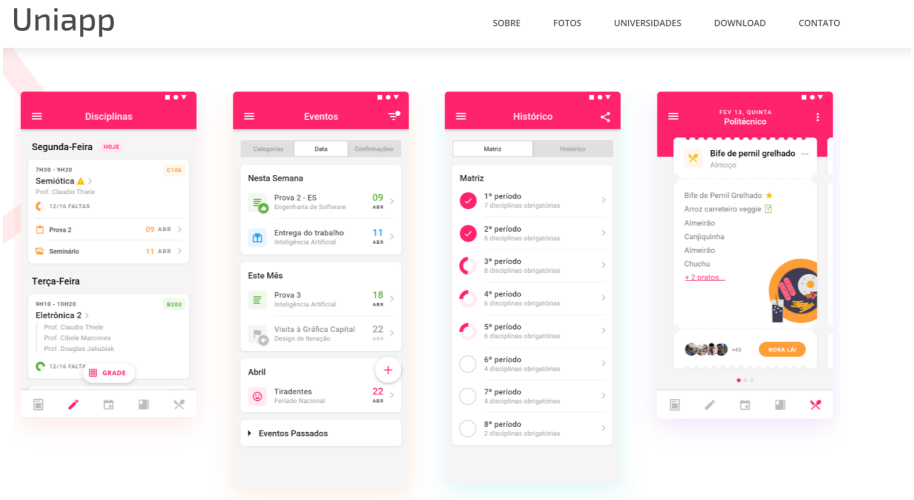
O primeiro projeto apresentado é o aplicativo e-Notas. Este aplicativo foi publicado pela conta de nome Luciano Medeiros na Google Play Store por volta de dezembro de 2015 com base no site Internet Archive. O desenvolvedor descreve o aplicativo como uma solução voltada para alunos universitários que visa auxiliar no registro dos seus históricos acadêmicos.

Em sua última versão publicada, datada em setembro de 2018 segundo o site Apkpure, o aplicativo dava suporte a cerca de 17 instituições, das quais se destacam a Universidade Estadual da Paraíba e a Universidade Federal de Campina Grande, sendo as mesmas objetos de estudo para o aplicativo desenvolvido neste trabalho. Durante a escrita deste trabalho (julho de 2023), o aplicativo não se encontra mais disponível para download através da Google Play Store.

Ao analisar o aplicativo, constatou-se que suas principais vantagens eram o suporte para múltiplas universidades, possibilitando um maior alcance de usuários. Entretanto, identificaram-se problemas significativos, como a indisponibilidade para download, uso de uma interface defasada e o suporte limitado apenas ao sistema operacional Android.

2.6.2 Uniapp

Figura 6 – Página de apresentação do aplicativo Uniapp



Fonte: Carbonaut, 2023.

O segundo projeto que serviu como base de estudo foi o Uniapp, desenvolvido pelo grupo Carbonaut. Segundo os desenvolvedores, esse aplicativo oferece grade horária organizada, calendário colaborativo, informações sobre biblioteca, notas e professores.

O projeto foi criado por alunos e é focado em instituições localizadas no sul do país como a Universidade Tecnológica Federal do Paraná, Universidade Federal do Paraná e Universidade Federal de Santa Catarina. Mas de acordo com a descrição do projeto é possível que novas instituições de outras partes do país sejam suportadas em futuras atualizações.

Algumas ideias foram obtidas ao analisar esse projeto, como a interface moderna, o suporte para múltiplas universidades e por ser um aplicativo multiplataforma. E, como possíveis evoluções, a adição de outras universidades de diferentes partes do país.

3 MATERIAIS E MÉTODOS

A metodologia adotada para o desenvolvimento do aplicativo proposto foi baseada em uma abordagem iterativa e incremental, seguindo princípios ágeis de desenvolvimento de *software* (Flora et al., 2014) . O processo foi dividido em etapas que abrangem desde a concepção inicial até a implantação final do aplicativo.

A primeira etapa consistiu na análise do contexto e na definição dos objetivos do projeto. Foi realizada uma revisão de trabalhos relacionados e estudos sobre portais acadêmicos, uso de dispositivos móveis no ambiente acadêmico e aplicativos voltados para estudantes. Essa revisão embasou a definição dos requisitos funcionais e não funcionais do aplicativo, considerando as necessidades dos usuários e as melhores práticas identificadas.

Em seguida, ocorreu o levantamento de requisitos, que envolveu conversas com estudantes que utilizam os portais suportados na aplicação e em avaliações de projetos similares nas lojas de aplicativo com o intuito de compreender as expectativas e demandas dos usuários. Essa etapa foi fundamental para identificar as principais funcionalidades e os fluxos necessários para o acesso e apresentação eficiente dos dados. E, com base nos requisitos levantados, foi realizado o processo de escolha das tecnologias e ferramentas adequadas.

O desenvolvimento do aplicativo seguiu uma abordagem ágil, com iterações curtas e frequentes, permitindo a validação contínua das funcionalidades implementadas. Foram realizados testes unitários e testes de integração para garantir a qualidade e a estabilidade do aplicativo. O feedback dos usuários e as revisões constantes foram essenciais para ajustes e refinamentos ao longo do processo de desenvolvimento.

Por fim, o aplicativo foi implantado em um ambiente de produção, após passar por um processo de validação e testes finais. A implantação envolveu a integração com os sistemas de instituições acadêmicas, garantindo a segurança e a privacidade dos dados dos usuários. A avaliação contínua do aplicativo foi realizada por meio de métricas de desempenho e testes de usabilidade, permitindo a identificação de possíveis melhorias em futuras atualizações.

3.1 Definição de Requisitos

O levantamento de requisitos foi realizado para compreender as necessidades dos usuários e as funcionalidades necessárias para o aplicativo proposto. Os requisitos elencados foram obtidos através de conversas com alunos, avaliações e comentários nas páginas de projetos similares e na análise dos projetos mais populares nas lojas de aplicativos.

3.1.1 Requisitos Funcionais (RF):

Quadro 1 – Requisitos Funcionais

RF	Descrição
RF01: Autenticação de Usuário	Os usuários devem poder fazer login no aplicativo usando credenciais fornecidas pela instituição acadêmica
RF02: Visualização de dados pessoais	Os usuários devem poder ter acesso a seus dados acadêmicos pessoais como matrícula, índices acadêmicos e carga horária/créditos.
RF03: Visualização de turmas em curso	Os usuários devem poder ter acesso a seus dados acadêmicos a respeito de turmas em curso, como professores, faltas e notas.
RF04: Acesso a Horários de Aula	Os usuários devem ter acesso aos horários de aula e local.
RF05: Notificações	Os usuários devem receber notificações quando novas notas, faltas e demais dados forem atualizados.
RF06: Acesso ao Histórico	Os usuários devem ter acesso ao seu histórico acadêmico.

Fonte: Elaborado pelo autor.

3.1.2 Requisitos Não Funcionais (RNF):

Quadro 1 – Requisitos Não Funcionais

RNF	Descrição
RNF01: Multiplataforma	O aplicativo deve ser compatível com múltiplas plataformas, como Android e iOS.
RNF02: *Offline First*	O aplicativo deve permitir que os usuários acessem suas informações mesmo sem conexão à internet.
RNF03: *Consistência*	Os dados devem ser sincronizados automaticamente quando houver uma conexão estável.
RNF04: Segurança	O aplicativo deve garantir a segurança dos dados dos usuários, implementando medidas de proteção, como criptografia e autenticação.
RNF05: Acessibilidade	O aplicativo deve possuir acessibilidade para pessoas com baixa visão.

Fonte: Elaborado pelo autor.

3.2 Materiais

Nesta seção, serão apresentados os materiais utilizados no desenvolvimento do aplicativo. Serão destacados o *framework* adotado, arquitetura implementada e escolhas de *design*.

3.2.1 Flutter

Após uma análise criteriosa das opções disponíveis e levando em consideração os requisitos elencados e o escopo do projeto, foi decidido pela utilização do Flutter para o desenvolvimento da aplicação. Uma das principais razões para escolher o Flutter é o foco em desenvolvimento multiplataforma, o que permitirá a criação de aplicativos tanto para dispositivos Android quanto iOS a partir de um único código-base sem abrir mão de performance e experiência do usuário.

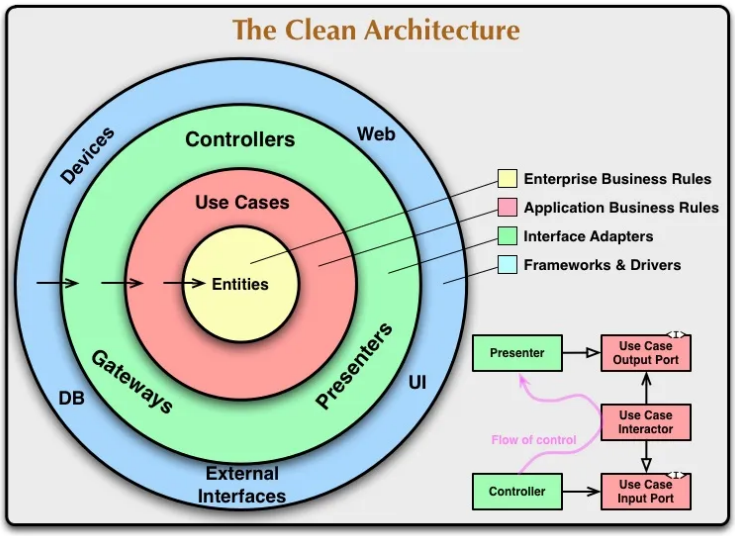
3.2.2 Material Design

A linguagem de *design* escolhida para o projeto foi o Material Design, especificamente na versão 3. A escolha foi feita levando em conta que esse é um Design System bem estabelecido, fundamento em anos de pesquisa pela equipe do Google e que é uma interface familiar, principalmente para usuários de Android e de aplicativos da própria Google. Além disso, esse padrão de *design* permite personalizações, o que facilita a adoção e adaptação para atender os requisitos de usabilidade do projeto desenvolvido.

3.2.3 Arquitetura Implementada

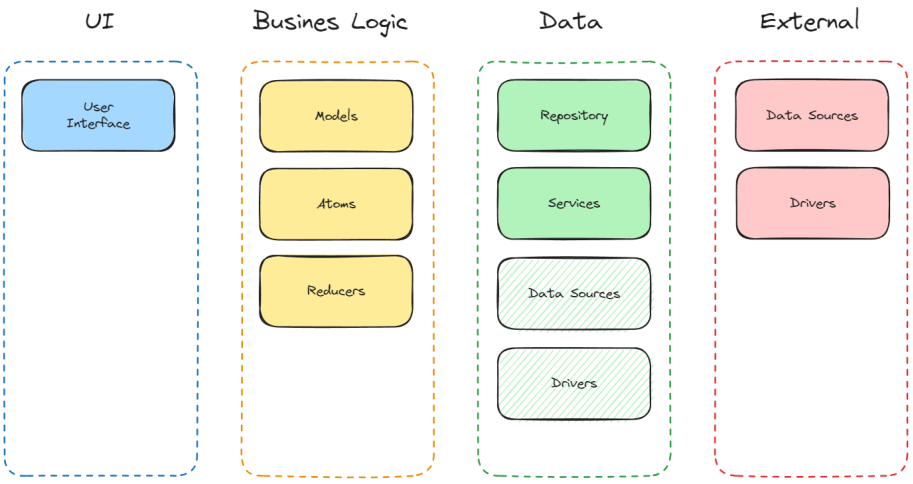
Após análise das abordagens arquiteturais mais utilizadas e recomendadas para aplicações móveis, foi definida uma estrutura que usa conceitos de *Clean Architecture* propostos por Uncle Bob (Figura 7), usando camadas para separação de responsabilidades, mas utilizando uma ideia mais simplificada que se adeque melhor ao escopo do projeto (Figura 8).

Figura 7 – Camadas de uma arquitetura limpa



Fonte: The Clean Code Blog, 2012.

Figura 8 – Camadas da arquitetura adotada e seus componentes



Fonte: Elaborado pelo autor.

A estrutura utilizada é composta por quatro partes. A camada de **UI** (*Presenter*), responsável por definir a estrutura da interface do usuário, como navegação de telas, componentes visuais e usabilidade em geral. A camada de Business Logic (*Domain*), atua como uma conexão entre a interface do usuário, as regras de negócio e os dados. Nessa camada são definidos os *Models* (*Entities*) para representar os dados da aplicação, *Atoms* que são propriedades reativas que armazenam o estado da aplicação e, por fim, os *Reducers* que implementam as regras de negócio que, superficialmente, lembram os *use cases*.

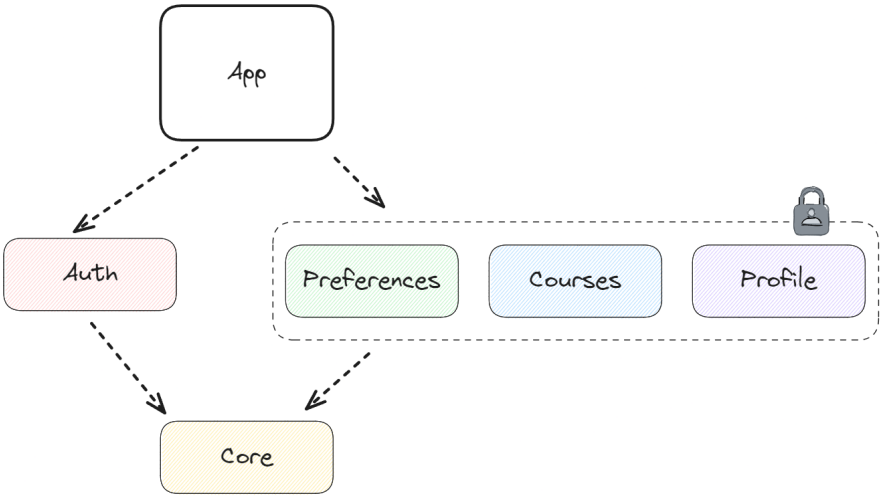
A camada Data é responsável por adaptar dados de fontes externas servindo de suporte para a camada de lógica de negócios. Essa camada é composta por *Repositories* que podem depender de *Services* e *Data Sources.* Aqui esses *Data Sources* são contratos (*Interfaces*) que devem ser implementados por camadas inferiores da arquitetura. De mesma forma são definidos os *Drivers* (*Adapters*) que, no contexto desta aplicação, são responsáveis por acesso a serviços nativos específicos de cada plataforma, como notificações e uso em segundo plano, esses adaptadores são utilizados nos *Services*, que são uma abstração para realizar a comunicação com os recursos independentemente da plataforma ou implementação utilizada.

A camada External, é responsável por implementar todo acesso a recursos externos que dependem de *hardware*, ferramentas de terceiros ou comunicação com recursos nativos da plataforma. As implementações nesta camada devem respeitar os contratos estabelecidos em camadas superiores e, qualquer mudança realizada nesta camada, não deve comprometer o funcionamento das demais camadas. Na aplicação desenvolvida, essa camada implementa dois tipos de contratos: *Data Sources* e *Drivers.* Primeiro, os *Data Sources* implementados podem ser locais, responsáveis por armazenamento local, e remotos, responsáveis por obter os dados dos portais acadêmicos. O segundo tipo de implementação são de *Drivers*, que são abstrações criadas para utilização de pacotes de terceiros para acesso a recursos nativos, nesse caso, acesso às notificações e agendamento de tarefas em segundo plano.

3.2.4 Estrutura Modular

Além da arquitetura que define a organização do aplicativo em camadas, também foi feita uma separação de funcionalidades da aplicação em módulos isolados, como mostra a Figura 9. Onde cada módulo conta com suas próprias camadas internas e oferece uma interface para expor seus recursos para outros módulos.

Figura 9 – Divisão de Módulos da aplicação



Fonte: Elaborado pelo autor.

Essa abordagem foi escolhida por facilitar a manutenção e a escalabilidade do projeto, dado que com a separação de responsabilidades é possível trabalhar em uma parte do projeto de forma isolada de outro módulo.

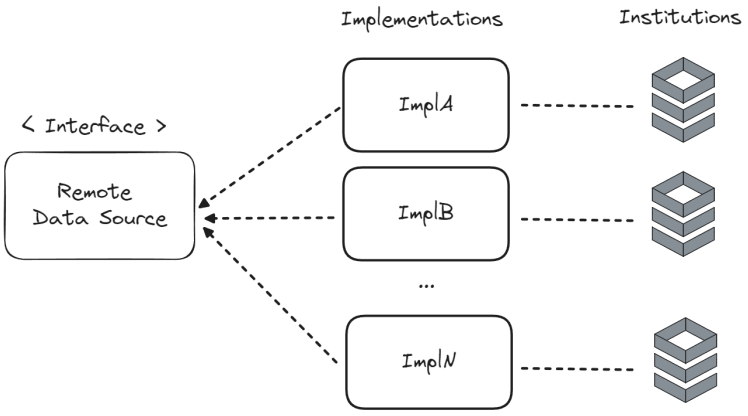
Para isso foram criados os seguintes módulos: *Core* que serve de suporte para os outros módulos fornecendo recursos compartilhados como acesso a persistência de dados, cliente *http* e sistema de notificações. O módulo *Auth* é responsável pela autenticação e autorização do usuário. Baseado nas informações desse módulo que a aplicação permite o acesso aos demais módulos. *Preferences* é responsável por gerenciar as configurações do aplicativo e personalizações feitas pelo usuário. O módulo *Course* gerencia as informações relacionadas aos cursos registrados, como horários e notas. E, por fim, o módulo *Profile* é responsável por gerenciar o acesso a informações pessoais do usuário.

Para auxiliar nessa abordagem foi utilizado o pacote *flutter_modular*, que fornece um sistema de injeção de dependências que é essencial para a arquitetura utilizada, além de oferecer um sistema de rotas para auxiliar na navegação entre módulos.

3.2.5 Remote Data Sources

O *Remote Data Source*, em sua essência, é uma camada de abstração entre o aplicativo e as instituições acadêmicas suportadas. Ele é responsável por buscar e fornecer os dados dos usuários de maneira uniforme, independentemente das nuances das diferentes estruturas de sistemas que as instituições possam empregar.

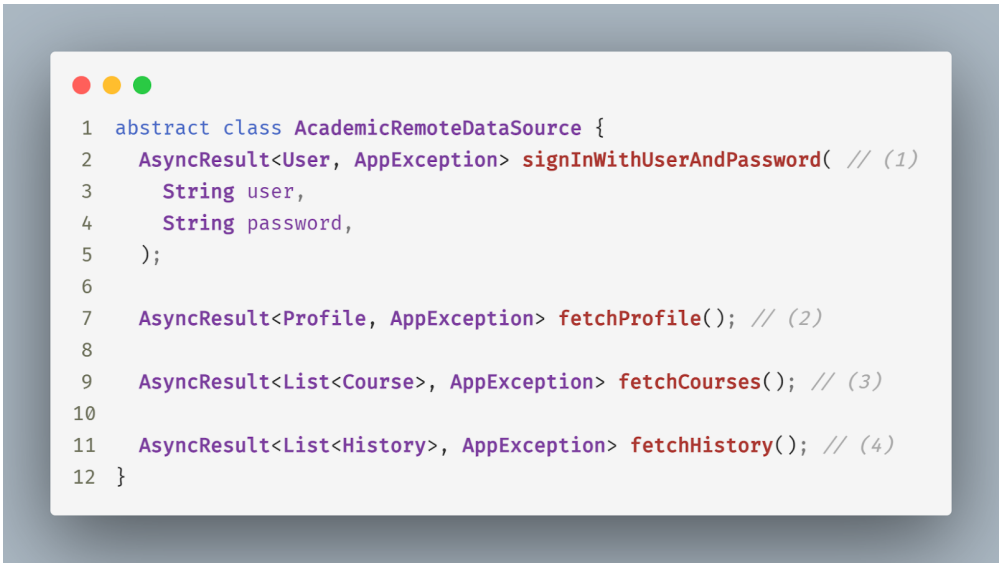
Figura 10 – Diagrama de comunicação de Data Sources



Fonte: Elaborado pelo autor.

Essa camada é responsável por consultar as instituições acadêmicas e tratar os dados recebidos para que atendam os contratos estabelecidos por camadas superiores da arquitetura. Cada instituição possui uma implementação específica de como fazer a requisição dos dados e como eles devem ser processados mas, independente da implementação, as camadas superiores esperam que o *Data Source* utilizado siga o contrato estabelecido através da interface apresentada na Figura 11.

Figura 11 – Código da interface do *Remote Data Source*



```dart
abstract class AcademicRemoteDataSource {
  AsyncResult<User, AppException> signInWithUserAndPassword( // (1)
    String user,
    String password,
  );

  AsyncResult<Profile, AppException> fetchProfile(); // (2)

  AsyncResult<List<Course>, AppException> fetchCourses(); // (3)

  AsyncResult<List<History>, AppException> fetchHistory(); // (4)
}
```

Fonte: Elaborado pelo autor.

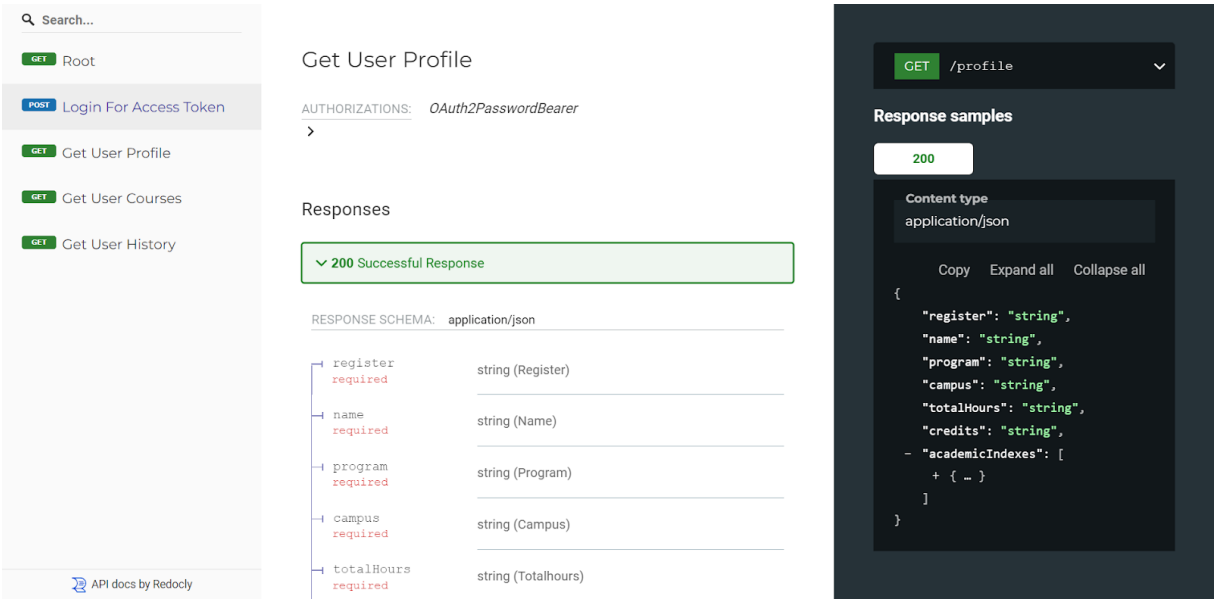
O contrato estabelece que toda implementação forneça um método para autenticar o usuário através de *username*, que pode ser a matrícula ou e-mail dependendo da instituição, e senha (1). Um método para consultar o perfil acadêmico do aluno (2) como nome, matrícula e demais dados fornecidos por cada instituição. Além disso, é preciso implementar um método que forneça as disciplinas matriculadas (3) e um método para consultar o histórico acadêmico do usuário (4).

Além disso, toda implementação deve retornar as informações como *AsyncResult<T, E>*, que é uma simplificação de *Future<Result<T, E>>*, isso quer dizer que o valor retornado é uma *Future,* resultado de uma execução assíncrona em Dart, que, quando concluído, pode ser tanto o dado esperado *T* quando bem sucedido, ou um erro *E* caso contrário.

A escolha por usar *async,* palavra reservada para métodos assíncronos em Dart, evita o bloqueio na execução principal do aplicativo enquanto os dados são processados. Já o uso do *Result*, é pensado para que os erros sejam tratados como valor, o que dispensa o disparo de exceções que alteram o fluxo de execução de maneira imprevisível. Além disso, usando *Result,* é possível obter uma sintaxe mais compreensiva para tratar os erros recebidos.

3.2.6 Protocolo Aberto

Figura 12 – Especificação da API do protocolo idealizado



Fonte: Elaborado pelo autor.

Como explicado em seções anteriores, o aplicativo desenvolvido acessa os dados dos usuários através do portal acadêmico da universidade em que eles estão matriculados. Para isso é preciso que se implemente um *Data Source* seguindo o contrato estabelecido pela aplicação. Isso pode ser visto como uma barreira técnica dado que essa implementação precisa ser feita em Dart, linguagem utilizada para desenvolver a aplicação, e que esse Data Source seja embarcado junto da aplicação principal, o que necessitaria lançar uma nova versão do aplicativo sempre que for preciso alterar algo da implementação de um Data Source. Com isso em mente foi desenvolvido uma implementação genérica que segue o contrato estabelecido, mas que age de maneira diferente.

A ideia dessa implementação é que o acesso aos dados acadêmicos é realizado através de um serviço *web* usando a arquitetura *REST* que atua como intermediário entre a aplicação e a instituição de ensino. Com isso, a instituição de ensino não precisará alterar seu sistema para fornecer os dados no formato esperado pela aplicação, e nem é necessário embarcar no aplicativo uma adaptação para tratar os dados recebidos.

O processamento dos dados é feito em uma aplicação externa que acessa os dados no portal acadêmico, faz as transformações necessárias e fornece os dados tratados através de uma API seguindo um protocolo que é suportado pelo *data source* genérico que, por contrato, fornece esses dados para as camadas internas do aplicativo móvel.

A definição de um protocolo para essa funcionalidade necessitaria de um estudo mais aprofundado mas, para o contexto deste trabalho, foi utilizada uma implementação simplificada como prova de conceito. Mesmo sendo uma ideia inicial, foram aplicados conceitos de segurança, como autenticação via JWT e uma especificação OpenAPI para exemplificar a ideia proposta.

3.2.7 Local Data Sources

O *Local Data Source* por sua vez, é uma camada de abstração entre a camada de dados e o sistema de arquivo do dispositivo. No contexto da aplicação desenvolvida, foram utilizados *data sources* para armazenar os dados acadêmicos e preferências do usuário e os dados de autenticação.

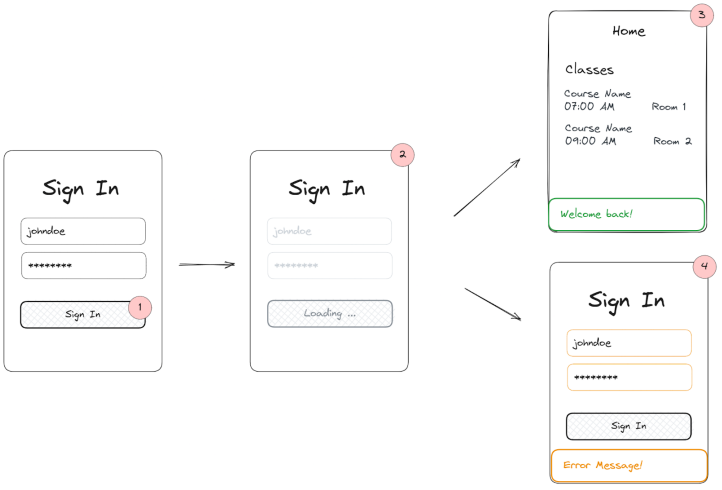
Os dados de configurações e os dados acadêmicos, como disciplinas cursadas e histórico, foram armazenadas utilizando uma implementação baseada no *plugin* Isar, que é uma base de dados local que utiliza o armazenamento convencional das plataformas. Já os dados de autenticação, que necessitam de um grau maior de segurança, foram armazenados utilizando uma implementação baseada no *plugin* flutter_secure_storage, que é uma abstração para o sistema de armazenamento encriptado das plataformas Android e IOS.

3.2.8 Gerenciamento de Estado

No contexto de desenvolvimento de *software*, o estado (*state*) é uma representação dos dados de uma aplicação em um determinado momento. No flutter especificamente, os *widgets* são responsáveis por descrever como a interface deve parecer dado uma configuração ou estado. Com isso, o gerenciamento do estado indicaria quando a interface deve ser reconstruída quando houver alguma alteração dos dados e como ela deve se comportar.

Em uma aplicação reativa, o estado pode ser alterado por diferentes ações, seja por uma interação do usuário, ou por um evento esperado ou inesperado no caso de um erro. Com isso, a interface do usuário deve sempre estar atualizada para apresentar as informações mais recentes, sendo necessário realizar um gerenciamento do estado para controlar e propagar essas alterações para componentes da aplicação que precisam ser informadas.

Figura 13 – Diagrama de transição de estados da interface de usuário



Fonte: Elaborado pelo autor.

Na Figura 12, é apresentada uma explicação em alto nível de um fluxo de estados de uma aplicação. Onde, em (1), são inseridos os dados de autenticação e o botão para realizar a ação é pressionado. Até esse momento a aplicação se encontra no estado inicial, quando o botão é pressionado é disparada uma alteração no estado da aplicação, apresentada em (2), o que deixa a aplicação em um estado de espera. Neste momento existem duas possibilidades de transição de estado: em

(3), é apresentada uma transição com estado de sucesso e, em (4), uma transição com estado de falha.

O gerenciamento de estados simples pode ser facilmente resolvido utilizando *setState*, recurso do flutter para reconstruir a interface com um novo estado. Os problemas começam a aparecer quando é necessário gerenciar estados com múltiplas variáveis e que precisam ser propagados para diferentes componentes da aplicação, sendo necessário um padrão de como as alterações devem ser feitas e de como notificar os interessados nessas alterações de estado.

Após o estudo das principais abordagens para auxiliar no gerenciamento e propagação do estado, como o BLoC, MobX e Triple, a abordagem do estado atômico se mostrou como a mais viável para o aplicativo desenvolvido dado a sua simplicidade e granularidade.

O estado atômico é um conceito de gerenciamento em que o estado é dividido em unidades independentes e atualizáveis individualmente, chamadas átomos. Cada átomo representa uma parte indivisível do estado, permitindo um controle granular sobre o acesso e a atualização do estado.

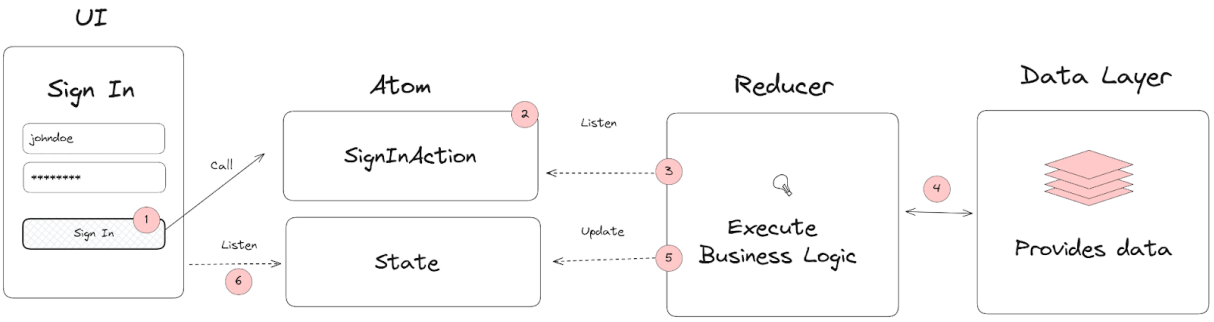
No contexto do Recoil, uma biblioteca de gerenciamento de estado para o React, os átomos são a base do estado atômico. Um átomo é um contêiner que armazena um valor de estado e pode ser acessado e atualizado por componentes diferentes. Cada átomo é independente, o que significa que as atualizações em um átomo não afetam diretamente outros átomos.

Além dos átomos, o Recoil também utiliza os *reducers* para gerenciar as atualizações do estado. Reducers são funções que recebem o estado atual e retornam o próximo estado com base em uma ação ou evento. Os reducers permitem que as alterações no estado aconteçam de forma previsível e consistente.

No Flutter, podemos adaptar o conceito de estado atômico com átomos e *reducers* utilizando os recursos nativos do *framework*, como o *ValueNotifier*. Mas, para auxiliar no desenvolvimento, foi utilizado o pacote ASP, que estende as funcionalidades nativas do Flutter para oferecer recursos e sintaxe mais coesa para essa abordagem.

Na Figura 13, é apresentado como o conceito de *atoms* e *reducers* foram adaptadas à arquitetura do projeto.

Figura 14 – Diagrama do fluxo do estado atômico da aplicação



Fonte: Elaborado pelo autor.

Em (1), ao pressionar o botão é disparada a reatividade no átomo (2), o *reducer* que observa quando essa reatividade é ativada (3) é acionado e realiza a lógica de negócio necessária para concluir a tarefa, que pode ou não depender de recursos da camada de dados (4). Após realizar o processamento, o *reducer* atualiza o estado da aplicação (5) com o resultado obtido. Por fim, a interface do usuário que escuta por alterações do estado (6) realiza uma atualização para apresentar o novo estado.

3.2.9 Ambiente de Desenvolvimento

O desenvolvimento do código foi realizado utilizando editor de Visual Studio Code, com o *plugin* Flutter e Dart. Também foi utilizado o pacote flutter_lints para definição de regras de estrutura do código, como sempre declarar o tipo de retorno nos métodos, por exemplo.

3.2.10 Testes

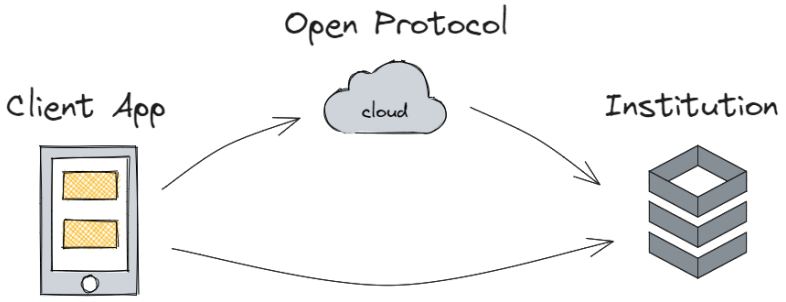
Para validação do código desenvolvido, foram criados testes unitários utilizando a ferramenta flutter_test que vem embarcada no conjunto de ferramentas de desenvolvimento do Flutter. Além disso, para auxiliar nos testes de injeção de dependências, foi utilizado o pacote mocktail que fornece uma sintaxe para a criação de mocks, que são objetos que simulam o comportamento de objetos reais de forma controlada.

4 RESULTADOS

Como foi descrito em capítulos anteriores, a solução final se comporta como um cliente que acessa os dados acadêmicos dos usuários através dos portais acadêmicos das universidades. O acesso pode ser realizado diretamente a partir do *Remote Data Source* específico de cada instituição, ou realizado através do protocolo aberto, onde a aplicação se comunica com o serviço *web* e esse serviço se comunica com a instituição.

A Figura 15 mostra um diagrama em alto nível da solução desenvolvida.

Figura 15 – Diagrama da Solução desenvolvida



Fonte: Elaborado pelo autor.

Neste capítulo serão apresentados os resultados obtidos da implementação do software idealizado e também informações iniciais sobre a publicação do aplicativo em ambiente de produção.

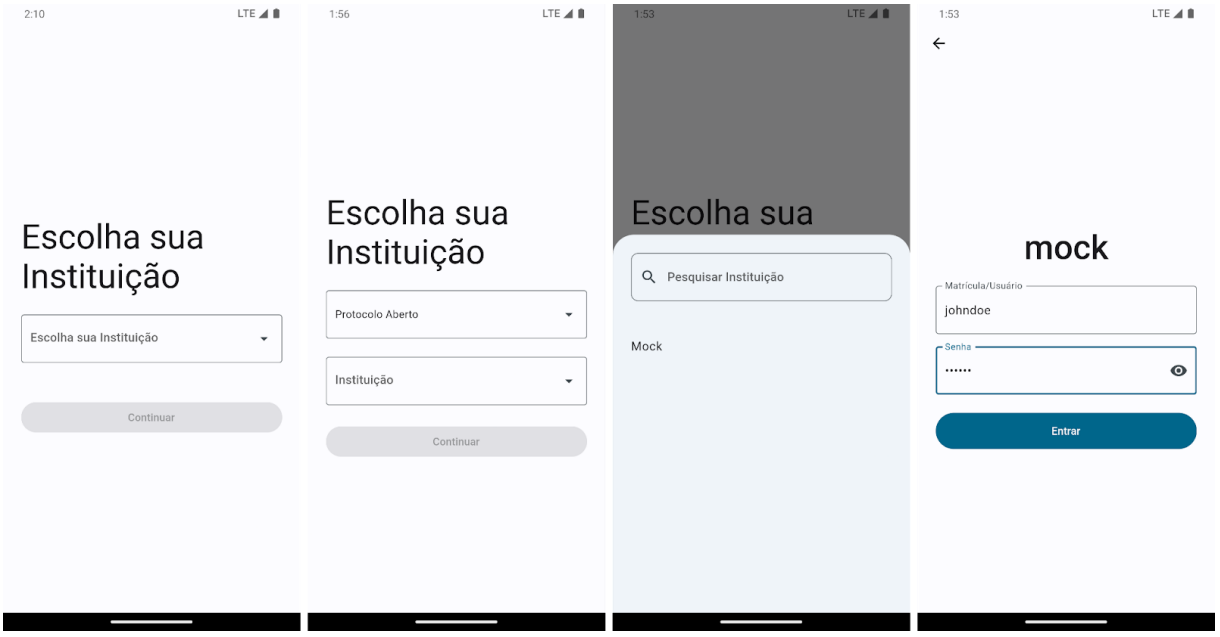
4.1 Interface de Usuário

A seguir são apresentadas as telas desenvolvidas que compõem a interface do usuário. A interface foi planejada de acordo com princípios de usabilidade e os padrões estabelecidos pelo Material Design 3.

4.1.1 Autenticação

O primeiro fluxo apresentado quando se acessa o aplicativo é a realização de autenticação do usuário (Figura 16). Nesse processo o usuário escolhe sua instituição de ensino e insere seus dados de acesso.

Figura 16 – Telas do fluxo de autenticação

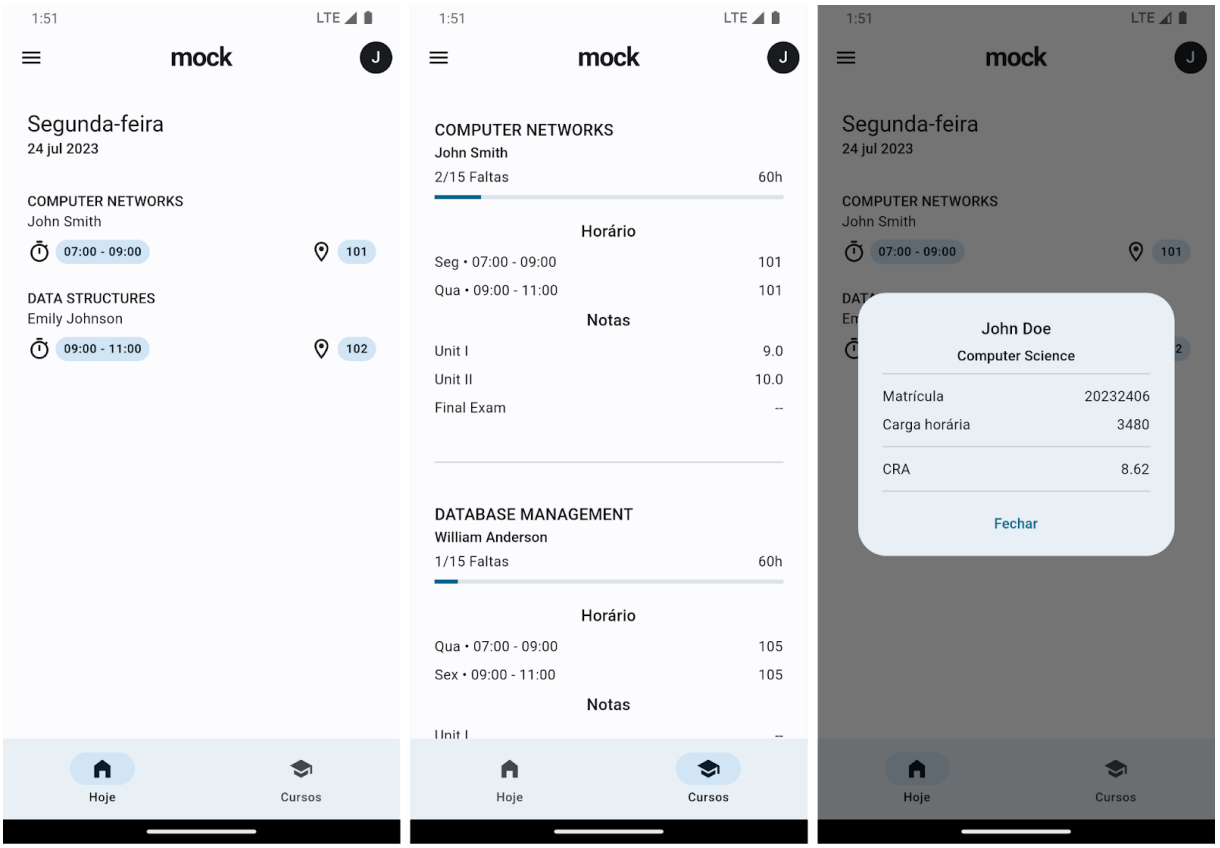


Fonte: Elaborado pelo autor.

4.1.2 Navegação Primária

Após se autenticar, o usuário é levado para a tela inicial da aplicação, que é apresentada na Figura 17.

Figura 17 – Navegação primária da aplicação



Fonte: Elaborado pelo autor.

A navegação primária no aplicativo é feita através da barra de navegação na parte inferior da tela, que oferece acesso rápido e conveniente às principais funcionalidades.

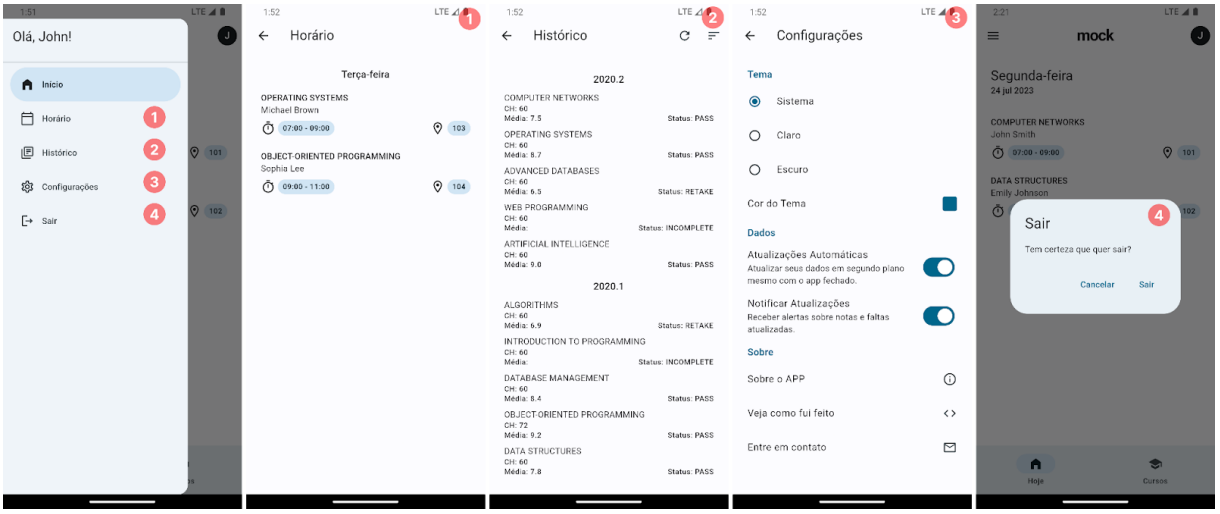
A primeira opção de navegação leva os usuários para as suas aulas do dia, sendo a aba principal do aplicativo. Essa escolha estratégica é baseada na premissa de que, ao abrir o aplicativo, a informação mais relevante e de maior interesse para a maioria dos usuários é a agenda diária das aulas. Essa interação foi projetada para levar em média menos de 15 segundos, entre um *cold start* (inicialização a partir de um estado inativo), carregamento e apresentação das dados, visualização das informações e encerramento do aplicativo.

A segunda opção de navegação exibe os cursos que o usuário está matriculado. A interação com essa tela é mais complexa em relação a primeira aba por oferecer informações mais detalhadas sobre cada curso. O usuário ao acessar essa aba tem como objetivo analisar um ou mais cursos para se informar se houve alguma alteração de notas ou faltas registradas, que são dados alterados com certa frequência, ao contrário do horário ou professor da disciplina que tende a ser o mesmo durante todo o semestre.

Por fim, a terceira interação na navegação primária é a exibição dos dados do usuário, como nome, curso, matrícula e índices acadêmicos. Essa interação é iniciada ao se tocar no ícone no canto superior direito da tela, chamado também de avatar. Essa interação é familiar para usuários habituados com o Material Design, principalmente em aplicativos do Google, como Gmail e Google Drive.

4.1.3 Navegação Secundária

Figura 18 – Navegação secundária da aplicação



Fonte: Elaborado pelo autor.

A navegação secundária (Figura 18) é feita utilizando um *navigation drawer*, que é uma técnica de *design* em aplicativos móveis e websites que serve para fornecer acesso rápido e organizado a funcionalidades ou seções menos prioritárias, mas ainda importantes. A seleção de páginas é realizada através de um painel que fica oculto na lateral da tela e pode ser aberto através do ícone de menu ou deslizado para revelar uma lista de opções que incluem recursos adicionais, configurações ou outras áreas acessadas com menor frequência.

Na aplicação desenvolvida, o *navigation drawer* conta com cinco opções, sendo a primeira o indicador que leva para a página inicial e mais quatro opções (indicadas de **1** a **4**). A opção **1** exibe a tela com o horário das aulas de toda a semana. A opção **2** apresenta a tela com o histórico acadêmico do usuário que exibe uma lista com as disciplinas cursadas com informações sobre nota, carga horária/ créditos e situação (aprovada, reprovada ou em curso), a lista é subdividida para agrupar os cursos de um mesmo semestre.

A opção **3** do menu leva o usuário para a tela de configurações do aplicativo. Nessa tela o usuário pode personalizar a aparência do aplicativo, configurar atualizações em segundo plano, acessar informações sobre o projeto e entrar em contato com os desenvolvedores.

Por fim, a opção **4** ativa a interação de *sign out,* que primeiro pergunta se o usuário deseja sair do aplicativo e, caso confirme, os dados do usuário são apagados e o aplicativo retorna a interação de autenticação.

4.2 Atualização de Dados

Figura 19 – Fluxo de ativação e recepção de alertas

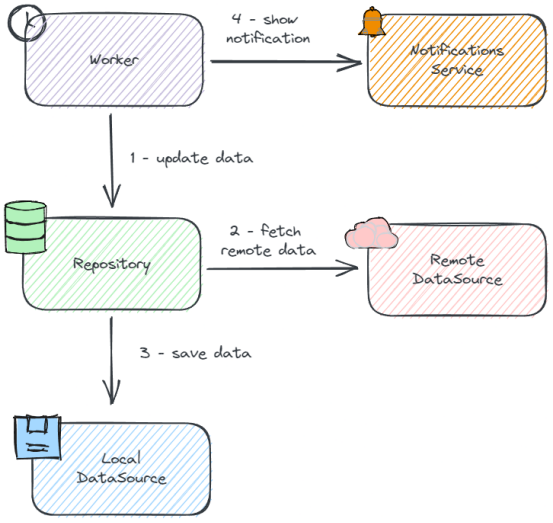
Fonte: Elaborado pelo autor.

Uma das principais funcionalidades da aplicação são as atualizações em segundo plano, onde o aplicativo fará consultas periódicas ao portal acadêmico do usuário para checar se existe alguma alteração dos dados, como faltas, notas ou mudança de professor de um um curso, por exemplo. Com isso os dados apresentados no aplicativo ficam sempre sincronizados com os dados remotos. O fluxo de notificação é apresentado na Figura 19.

Essa checagem acontece mesmo que o aplicativo não esteja em execução, onde são utilizados recursos do sistema operacional para agendar tarefas. Devido a essa característica, a funcionalidade é desativada por padrão, sendo escolha do usuário ativar ou não o recurso (indicação **1**). O usuário também pode escolher por autorizar as notificações(indicação **2** e **3**), autorizando que o aplicativo exiba um alerta caso novos dados sejam recebidos, como é apresentado nas indicações **4** e **5**.

A seguir é apresentado um diagrama (Figura 20) em alto nível da implementação da funcionalidade:

Figura 20 – Diagrama de componentes de serviços em segundo plano



Fonte: Elaborado pelo autor.

A funcionalidade foi implementada seguindo a arquitetura do projeto, onde ao se ativar o recurso através da interface do usuário, é executado o caso de uso que se comunica com o *Worker Manager* para registrar um *worker* programado para ser executado em um intervalo determinado. Esse *worker* se comunica com o repositório de cursos e de perfil do usuário (1) para atualizar seus dados armazenados. Cada repositório acessa os seus respectivos *data sources* para obter os dados mais recentes do portal acadêmico (2) que, ao serem recebidos, as atualizações são persistidas localmente (3).

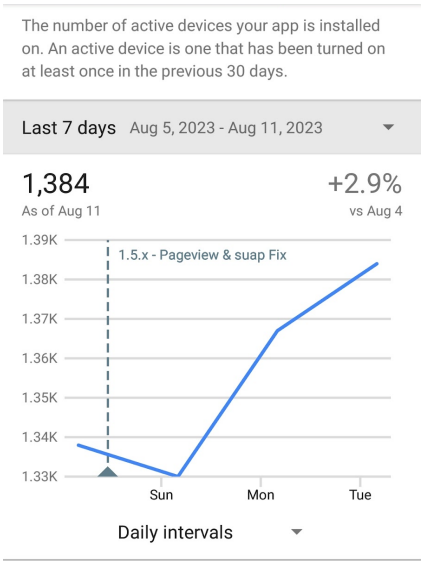
A seguir, as informações novas e antigas são comparadas e, caso haja alguma alteração, é executado o caso de uso para exibir notificações (4), que se comunica com o serviço de notificações que exibe um alerta para o usuário.

4.3 Visão Geral da Publicação

Após o desenvolvimento do MVP e da realização de validações internas, feita por meio de testes automatizados e com usuários alpha, o aplicativo foi publicado na loja de aplicativos Google Play Store. De início, a publicação foi feita em nível de acesso beta e, posteriormente, foi liberada para o público geral. Após coleta de *feedback* dos usuários e de detecção de erros, novas versões foram publicadas na loja, seguindo a prática de que atualizações críticas seriam enviadas diretamente para o público geral, e atualizações não críticas passariam primeiro por usuários beta.

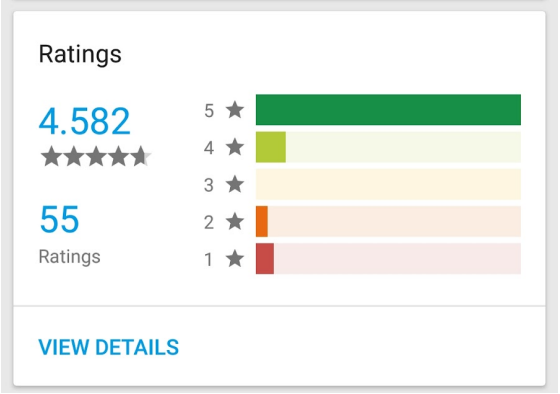
A seguir são exibidos estatísticas sobre a publicação do aplicativo. Os dados são limitados até 13 de agosto de 2023.

Figura 21 – Usuários ativos (5 a 11 de ago. 2023)



Fonte: Google Play Console, 2023.

Figura 22 – Avaliação do aplicativo



Fonte: Google Play Console, 2023.

Até o fim da escrita deste trabalho, o aplicativo contava com 1.384 usuários (Figura 21) ativos e com avaliação média de 4.5 de um máximo de 5.0 (Figura 22). As versões do aplicativo lançado contam com suporte a apenas duas universidades, o que pode limitar o público alvo inicial, além de possuir funcionalidades limitadas. Isso foi feito para que fosse possível obter algo concreto sobre tudo o que foi planejado durante a pesquisa e para validar a existência do projeto com usuários reais.

5 CONSIDERAÇÕES FINAIS

O resultado do desenvolvimento do aplicativo foi consideravelmente bem sucedido. Durante o processo de implementação, foram aplicados os princípios arquitetônicos que se mostraram mais eficientes para a construção e manutenção do projeto. Além da criação de uma interface de usuário levando em conta os padrões utilizados pela indústria tendo como o foco principal a usabilidade.

É importante destacar que, neste trabalho, optou-se por implementar apenas a versão para dispositivos Android, considerando a predominância dessa plataforma no cenário atual.

Ressalta-se também que o projeto foi desenvolvido com o código aberto, o que significa que seu código-fonte está disponível para acesso público e pode receber contribuições da comunidade de desenvolvedores. Essa abordagem de desenvolvimento colaborativo possibilita a evolução constante do aplicativo e a incorporação de novas ideias e melhorias.

Como possíveis trabalhos futuros, pretende-se explorar a versão do aplicativo para dispositivos iOS e a implementação de novas funcionalidades para aprimorar ainda mais a experiência dos usuários. Além disso, é possível que haja um refinamento na definição do protocolo aberto explorado durante este trabalho para que ele se torne mais robusto e funcional.

Dessa forma, com o desenvolvimento deste aplicativo, espera-se fornecer uma ferramenta valiosa para facilitar o acesso a dados acadêmicos de forma prática e satisfatória. A combinação da arquitetura sólida e a possibilidade de futuras contribuições tornam o aplicativo um projeto promissor, contribuindo para uma experiência acadêmica aprimorada.

REFERÊNCIAS

2023 Developer Survey. **Stack Overflow**, 2023. Disponível em: <https://survey.stackoverflow.co/2023>. Acesso em: 24 jul. 2023.

ABDAL, Mohamed; AHMED, Tarig; JAN, Sadeeq; KHAN, Fazal; KHATTAK, Amjad. A Comparative Analysis of Mobile Application Development Approaches. Proceedings of the Pakistan Academy of Sciences, v. 58, p. 35-45, 2021. DOI: 10.53560/PPASA(58-1)717.

App architecture. **Android Developers**, 2023. Disponível em: <https://developer.android.com/topic/architecture/intro>. Acesso em: 03, set. 2023.

Apple. **Human Interface Guidelines**. Disponível em: <https://developer.apple.com/design/human-interface-guidelines>. Acesso em: 18 jul. 2023.

BABICH, Nick. A Comprehensive Guide to Mobile App Design. **Smashing Magazine**, 2018. Disponível em: <https://www.smashingmagazine.com/2018/02/comprehensive-guide-to-mobile-app-design/>. Acesso em: 18, jul. 2023.

Build an offline-first app. **Android Developers**, 2023. Disponível em: <https://developer.android.com/topic/architecture/data-layer/offline-first>. Acesso em: 06, jun. 2023.

Dart. **Dart Documentation**. Disponível em: <https://dart.dev/guides>. Acesso em: 06 jun. 2023.

Data layer. **Android Developers**, 2023. Disponível em: <https://developer.android.com/topic/architecture/data-layer>. Acesso em: 09, jun. 2023.

FESSENDEN, Therese. Design Systems 101. **Nielsen Norman Group**, 2021. Disponível em: <https://www.nngroup.com/articles/design-systems-101/>. Acesso em: 18, jul. 2023.

Flora, H. K., Chande, S. V., & Wang, X. (2014). Adopting an agile approach for the development of mobile applications. International Journal of Computer Applications, 94(17).

Flutterando. **Flutter Modular Documentation**. Disponível em: <https://modular.flutterando.com.br/docs/intro/>. Acesso em: 09 jun. 2023.

GARLAN, David; SHAW, Mary. An Introduction to Software Architecture. Janeiro 1994.

Flutter. **Flutter Documentation**. Disponível em: <https://flutter.dev/docs>. Acesso em: 06 jun. 2023.

Flutter. **List of state management approaches**. Disponível em: <https://docs.flutter.dev/data-and-backend/state-mgmt/options>. Acesso em: 30 ago. 2023.

Flutter. **Testing Flutter apps**. Disponível em: <https://docs.flutter.dev/testing/overview>. Acesso em: 30 ago. 2023.

Google. **Material Design**. Disponível em: <https://m3.material.io/get-started>. Acesso em: 06 jun. 2023.

Guide to app architecture. **Android Developers**, 2023. Disponível em: <https://developer.android.com/topic/architecture>. Acesso em: 20, jul. 2023.

HANCHAR, Anna. Simple Beauty: The impact of visual complexity, prototypicality and color typicality on aesthetic perception in initial impression of websites. Novembro 2012.

Isar. **Isar Documentation**. Disponível em: <https://isar.dev/>. Acesso em: 09 ago. 2023.

MARTIN, Robert C.. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008.

Microsoft. **Fluent UI**. Disponível em: <https://developer.microsoft.com/fluentui>. Acesso em: 18 jul. 2023.

Microsoft. **Visual Studio Code**. Disponível em: <https://code.visualstudio.com/>. Acesso em: 09 ago. 2023.

Moura, Jacob A.. O Result e o problema não reconhecido pelos devs. **Flutterando**, 2022. Disponível em: <https://blog.flutterando.com.br/o-result-e-o-problema-n%C3%A3o-reconhecido-pelos-devs-5221d82eef19>. Acesso em: 13 ago. 2023.

PERRY, D. E.; WOLF, A. L. Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes, v. 17, n. 4, p. 40-52, 1992.

Pub. **ASP**. Disponível em: <https://pub.dev/packages/asp>. Acesso em: 09 ago. 2023.

Pub. **Flutter Lints**. Disponível em: <https://pub.dev/packages/asp>. Acesso em: 09 ago. 2023.

Pub. **Flutter Secure Storage**. Disponível em: <https://pub.dev/packages/asp>. Acesso em: 09 ago. 2023.

Pub. **Mocktail**. Disponível em: <https://pub.dev/packages/mocktail>. Acesso em: 09 ago. 2023.

Recoil. **Core Concepts**. Disponível em: <https://recoiljs.org/docs/introduction/core-concepts>. Acesso em: 09 ago. 2023.

Robinson, Landon. Why porting an iOS design to Android will not work. **MartianCraft**, 2016. Disponível em: <https://martiancraft.com/blog/2016/05/porting-ios-design-to-andoid/>. Acesso em: 03 set. 2023.

Web para Todos avalia sites das melhores universidades e escolas de ensino médio do Brasil. **Web para Todos**, 2017. Disponível em: <https://mwpt.com.br/web-para-todos-avalia-sites-das-melhores-universidades-e-escolas-de-ensino-medio-do-brasil/>. Acesso em: 06 ago. 2023.

Wang, Aoni. Cross-platform App Design: Why do some apps look different on iOS and Android?. **Medium**, 2017. Disponível em: <https://medium.com/@aoniwang/cross-platform-app-design-why-do-some-apps-look-different-on-ios-and-android-b75a1e3bf440>. Acesso em: 03 set. 2023.

XANTHOPOULOS, Spyridon; XINO GALOS, Stelios. A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. In: ACM International Conference Proceeding Series, 2013. DOI: 10.1145/2490257.2490292.