INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE RONDÔNIA
Vilhena - Código INEP: 11107804
Rodovia BR 174, KM 3, CEP 76982-270, Vilhena (RO)
CNPJ: 10.817.343/0003-69 - Telefone: 69 2101-0703

Luan Rafael Batista Ramos

Aplicativo para compartilhamento de arquivos em rede local - RB Share

Vilhena - RO

2025

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE RONDÔNIA
Vilhena - Código INEP: 11107804
Rodovia BR 174, KM 3, CEP 76982-270, Vilhena (RO)
CNPJ: 10.817.343/0003-69 - Telefone: 69 2101-0703

Luan Rafael Batista Ramos

Aplicativo para compartilhamento de arquivos em rede local - RB Share

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – campus Vilhena, realizado em cumprimento de requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas..

Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – IFRO
Campus Vilhena
Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas

Orientador: Prof. Esp. Erick Leonardo Weil.
Coorientador: Me. Marco Antonio Augusto de Andrade.

Vilhena - RO
2025

Ficha catalográfica elaborada pelo Sistema Gerador de Ficha Catalográfica do IFRO.

Ramos, Luan Rafael Batista.
Aplicativo para compartilhamento de arquivos em rede local - RB Share / Luan Rafael Batista Ramos. - Vilhena, 2025.
72 f.

Orientador(a): Prof. Esp. Erick Leonardo Weil.
Coorientador(a): Me. Marco Antonio Augusto de Andrade.

Trabalho de Conclusão de Curso (Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO, Vilhena, 2025.

1. Compartilhamento. 2. Arquivos. 3. Rede Local. I. Weil, Erick Leonardo (orient.). II. Andrade, Marco Antonio Augusto de (coorient.). III. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO. IV. Título.

Bibliotecário(a) Responsável: Rosilene Maria do Couto Marques, CRB-11/321

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE RONDÔNIA
Vilhena - Código INEP: 11107804
Rodovia BR 174, KM 3, CEP 76982-270, Vilhena (RO)
CNPJ: 10.817.343/0003-69 - Telefone: 69 2101-0703

ATA DE DEFESA DE MONOGRAFIA

Na data 30/05/2025 realizou-se a sessão pública de defesa da Monografia intitulada **Aplicativo para compartilhamento de arquivos em rede local - RB Share** apresentada pelo aluno **Luan Rafael Batista Ramos (2021103070014)** do Curso **Superior de Tecnologia em Análise e Desenvolvimento de Sistemas (Vilhena)**. Os trabalhos foram iniciados às **16:30** pelo Professor **Erick Leonardo Weil** presidente da banca examinadora, constituída pelos seguintes membros:

- **Erick Leonardo Weil** (Orientador)
- **Marco Antonio Augusto de Andrade** (Coorientador)
- **Wesley Jhonnes Ramos Rolim** (Examinador Interno)
- **Marco Antonio Augusto de Andrade** (Examinador Interno)

A banca examinadora, tendo terminado a apresentação do conteúdo da Monografia, passou à arguição do candidato. Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelo aluno, tendo sido atribuído o seguinte resultado:

[X] APROVADO **Nota: 94**

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu **Erick Leonardo Weil** lavrei a presente ata que assino juntamente com os demais membros da banca examinadora.

VILHENA / RO, 30/05/2025

Documento assinado eletronicamente por **Luan Rafael Batista Ramos**, Discente, em 30/05/2025, às 18:04, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Erick Leonardo Weil**, Orientador, em 30/05/2025, às 18:10, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Marco Antonio Augusto de Andrade**, Coorientador Interno, em 30/05/2025, às 18:08, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Wesley Jhonnes Ramos Rolim**, Examinador Interno, em 30/05/2025, às 20:15, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Marco Antonio Augusto de Andrade**, Examinador Interno, em 30/05/2025, às 18:07, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Dedico este trabalho a todos os que ousaram sonhar, tanto nos dias de céu claro quanto nas noites de código sem fim.

Aos colegas de estudo, que com risos e debates, transformaram cada desafio em uma conquista coletiva.

Às crianças que, com olhos curiosos e mentes inquietas, vislumbram o futuro e se inspiram a criar, não apenas o próximo grande aplicativo, mas um mundo melhor, onde a tecnologia é a ponte para realizações inimagináveis.

Que este trabalho seja um pequeno passo nesta longa jornada de transformar sonhos em realidades, códigos em experiências, e ideias em inovações que inspiram.

Agradecimentos

Os agradecimentos principais são direcionados a todos os alunos da graduação de Análise e Desenvolvimento de Sistemas que persistiram até o fim, mesmo com adversidades.

Agradeço também aos professores e orientadores que compartilharam seu conhecimento e forneceram suporte ao longo do curso, guiando-nos em nossa jornada acadêmica. Sua dedicação e paciência foram essenciais para o nosso desenvolvimento profissional e pessoal.

À minha família e amigos, que sempre estiveram ao meu lado, oferecendo encorajamento e apoio incondicional nos momentos de desafio. Sem vocês, esta conquista não teria sido possível.

Por fim, um agradecimento especial às empresas em que atuei durante a graduação, onde tive a oportunidade de aplicar na prática os conhecimentos adquiridos e de aprender com desafios reais do mercado de trabalho. Essas experiências complementaram a formação acadêmica, proporcionando um ambiente de aprendizado contínuo que foi crucial para o meu desenvolvimento como profissional.

A todos, o meu sincero agradecimento.

Resumo

Este projeto apresenta o desenvolvimento de um aplicativo de compartilhamento de arquivos, fotos e vídeos na rede local, intitulado “RB Share”, concebido para proporcionar uma experiência de usuário intuitiva e eficiente, facilitando a transferência de dados entre dispositivos conectados à mesma rede sem a necessidade de serviços de terceiros ou armazenamento em nuvem; ao longo do desenvolvimento, foram enfrentados diversos desafios técnicos, como a implementação de comunicação multicast para a descoberta automática de dispositivos e a configuração de servidores web locais para o compartilhamento seguro de arquivos via link; a aplicação foi desenvolvida utilizando Flutter, o que permitiu a criação de uma solução multiplataforma robusta e adaptável tanto para Android quanto para iOS; o processo de lançamento nas principais lojas de aplicativos envolveu a conformidade com diretrizes específicas e a preparação de todos os elementos necessários para garantir a aprovação e visibilidade do aplicativo, com suporte pós-lançamento estabelecido para monitorar o desempenho e implementar melhorias contínuas com base no feedback dos usuários; o RB Share oferece funcionalidades como a seleção de arquivos, pastas, texto, e conteúdo da área de transferência, além de modos de envio personalizados, incluindo a opção de compartilhamento via link com QR code; a segurança dos dados foi priorizada, com a possibilidade de adicionar criptografia SSL e gerenciar permissões de acesso; o projeto demonstrou a viabilidade e eficácia de uma solução de compartilhamento de arquivos que é ao mesmo tempo, poderosa e acessível, contribuindo para um avanço na usabilidade e na eficiência das transferências de arquivos em redes locais.

Palavras-chave: compartilhamento; arquivos; rede; aplicativo; multiplataforma;

Abstract

This project presents the development of a file, photo, and video sharing application on the local network, entitled "RB Share." The application was designed to provide an intuitive and efficient user experience, facilitating data transfer between devices connected to the same network without the need for third-party services or cloud storage. Throughout the development, several technical challenges were faced, such as the implementation of multicast communication for automatic device discovery and the configuration of local web servers for secure file sharing via link. The application was developed using Flutter, which allowed for the creation of a robust and adaptable multi-platform solution for both Android and iOS. The process of launching in the main app stores involved compliance with specific guidelines and the preparation of all necessary elements to ensure the application's approval and visibility. Post-launch support was established to monitor performance and implement continuous improvements based on user feedback. RB Share offers functionalities such as the selection of files, folders, text, and clipboard content, as well as customized sending modes, including the option to share via link with a QR code. Data security was prioritized, with the possibility of adding SSL encryption and managing access permissions. The project demonstrated the feasibility and effectiveness of a file-sharing solution that is both powerful and accessible, contributing to an advance in the usability and efficiency of file transfers on local networks.

Keywords: sharing; files; network; application; cross-platform;

Lista de ilustrações

Figura 1 – Visão geral do modelo C4 para visualização de arquitetura de software. . 22
Figura 2 – Diagrama ilustrando o fluxo GitFlow Simplificado. Fonte: ATLASSIAN . 30
Figura 3 – Controle de versão no GitHub com as branches `main` e `develop`, e o histórico de commits associado. 31
Figura 4 – Visualização das branches `main` e `develop` no GitHub (acima), e do histórico de commits dessas ramificações (abaixo). 31
Figura 5 – Diagrama de Contexto do C4 Model ilustrando interação do sistema com usuários e sistemas externos. 36
Figura 6 – Diagrama de Container do C4 Model ilustrando uma interação principal do sistema com usuários e sistemas externos. 37
Figura 7 – Diagrama de Componentes do C4 Model ilustrando uma interação principal do sistema com usuários e sistemas externos. 38
Figura 8 – Tela de receber, informações importantes, salvamento rápido e a tela de histórico 52
Figura 9 – Figura A: Tela de seleção de arquivos, onde o usuário escolhe o tipo de conteúdo a ser enviado. Figura B: Tela com os arquivos já selecionados, permitindo revisão antes do envio. 54
Figura 10 – Ilustração dos três modos de envio disponíveis no aplicativo: único destinatário, múltiplos destinatários e compartilhamento via link. 56
Figura 11 – Interface de dispositivos próximos: (A) lista de dispositivos disponíveis na rede local, (B) inserção manual de endereço IP, e (C) exibição da lista de dispositivos favoritos. 57
Figura 12 – Tela de solicitação de envio exibida no dispositivo destinatário, com informações detalhadas do remetente e opções de aceitar ou rejeitar a transferência. 58
Figura 13 – Solicitação de envio recebida pelo destinatário (Figura A) e opções de configuração antes da transferência (Figura B). 59
Figura 14 – Tela de recebimento de arquivos, com informações em tempo real sobre as transferências em andamento. 61
Figura 15 – Tela de compartilhamento via link. Figura A: link gerado com opções de aceitar solicitações automaticamente e exibir QR Code. Figura B: exibição de texto ao ativar a criptografia SSL. Figura C: link acessível via navegador e exibição de solicitações pendentes de acesso. 62

Figura 16 – Recebimento de arquivos via navegador web. Figura A: solicitação de acesso aguardando aprovação. Figura B: acesso autorizado com criptografia SSL ativada. Figura C: listagem de arquivos disponíveis para download após aprovação. 63

Lista de Quadros

3.1 Função para obtenção de sockets UDP multicast 39
3.2 Processamento de pacotes multicast recebidos 40
3.3 Resposta a anúncios de dispositivos na rede 40
4.1 Código do teste de favoritar dispositivo 47
4.2 Teste da conversão de JSON em *PrepareUploadResponseDto* 48
4.3 Código dos testes do histórico de recebimentos 49

Lista de tabelas

Tabela 1 – Requisitos Funcionais do Aplicativo de Compartilhamento de Arquivos na Rede Local (*RB Share*) 31
Tabela 1 – Requisitos Funcionais do Aplicativo de Compartilhamento de Arquivos na Rede Local (*RB Share*) 32
Tabela 1 – Requisitos Funcionais do Aplicativo de Compartilhamento de Arquivos na Rede Local (*RB Share*) 33
Tabela 2 – Requisitos Futuros do Aplicativo de Compartilhamento de Arquivos na Rede Local (*RB Share*) 33
Tabela 3 – Requisitos Não Funcionais do Aplicativo de Compartilhamento de Arquivos na Rede Local (*RB Share*) 34
Tabela 4 – Exemplo de testses manuais baseados nos requisitos funcionais 46
Tabela 5 – Casos de Teste Complementares 47

Sumário

Lista de Quadros 9

1 INTRODUÇÃO 13
1.1 Contexto e problema 13
1.2 Objetivos 14
1.2.1 Objetivo geral 14
1.2.2 Objetivos específicos 14
1.2.2.1 Analisar projetos similares Open Source ou Proprietários 14
1.2.2.2 Definir requisitos e planejar o sistema 14
1.2.2.3 Desenvolver a aplicação 15
1.2.2.4 Incorporar criptografia ponta a ponta (E2EE) 15
1.2.2.5 Testar e validar a solução 15
1.2.2.6 Publicar a aplicação nas lojas de aplicativos 15
1.3 Justificativa 15

2 FUNDAMENTAÇÃO TEÓRICA 18
2.1 Multicasting 19
2.2 Publicação de Aplicativos em Plataformas Mobile: Google Play Store e App Store 20
2.2.1 Preparação do Aplicativo 20
2.2.2 Criação de Contas de Desenvolvedor 20
2.2.3 Configuração de Informações e Metadados 21
2.2.4 Testes e Submissão para Revisão 21
2.2.5 Lançamento e Pós-Publicação 21
2.2.6 Conclusão da publicação 22
2.3 C4 Model para Arquitetura de Software 22
2.3.1 Diagrama de Contexto 23
2.3.2 Diagrama de Contêineres 23
2.3.3 Diagrama de Componentes 23
2.3.4 Diagrama de Código (ou Classes) 23
2.4 Benefícios do C4 Model 24
2.5 Trabalhos Similares 24
2.5.1 Aplicativos Proprietários 25
2.5.2 Aplicativos de Código Aberto 25
2.5.3 Diferenciais do RB SHARE 25

3 MATERIAIS E MÉTODOS 27
3.1 Flutter e Dart 27
3.2 Ferramentas Utilizadas 27

3.2.1 Visual Studio Code (VSCode) . . . 27
3.2.2 Xcode . . . 28
3.2.3 Android Studio . . . 28
3.2.4 Transporter . . . 28
3.3 Métodos . . . 28
3.3.1 Versionamento . . . 29
3.3.2 Git e GitHub . . . 29
3.3.3 GitFlow Simplificado . . . 29
3.3.4 Imagens . . . 30
3.4 Requisitos . . . 30
3.4.1 Requisitos Funcionais . . . 31
3.4.2 Requisitos Não Funcionais . . . 33
3.5 Arquitetura de Software . . . 34
3.5.1 Modelo MVC . . . 34
3.5.2 C4 Model . . . 35
3.6 Implementação do Multicast . . . 39
3.7 Licença de uso . . . 41
3.7.1 Detalhes da MIT License . . . 42
3.7.2 Vantagens da MIT License . . . 42
3.7.3 Texto Completo da Licença MIT . . . 42

4 RESULTADOS . . . 44
4.1 Gerenciamento de Tarefas . . . 44
4.1.1 Ferramentas Utilizadas . . . 44
4.1.2 Metodologia de Gerenciamento . . . 44
4.1.3 Resultados Obtidos . . . 44
4.2 Testes . . . 45
4.2.1 Testes Manuais . . . 45
4.2.1.1 Abordagem . . . 45
4.2.1.2 Casos de Teste . . . 46
4.2.2 Testes Automáticos . . . 47
4.2.2.1 Tipos de Testes Automáticos . . . 47
4.2.3 Resultados dos Testes . . . 50
4.3 Implantação . . . 50
4.3.1 Google Play Store . . . 50
4.3.1.1 Pontos-Chave . . . 50
4.3.2 Apple App Store . . . 51

4.3.2.1 Pontos-Chave 51
4.4 Demonstração 51
4.4.1 Visão Geral da Demonstração 51
4.4.2 Funcionalidades Principais 52
4.4.2.1 Receber 52
4.4.3 Identificação 53
4.4.3.1 Identificação adicional 53
4.4.4 Informações importantes 53
4.4.4.1 Enviar 54
4.4.5 Seleção 55
4.4.6 Modos de envio 55
4.4.7 Dispositivos próximos 57
4.4.8 Compartilhar via link 61
4.4.9 Compartilhar via link versão Web 63

5 CONSIDERAÇÕES FINAIS 64
5.1 Desafios Enfrentados 64
5.2 Aprendizados Adquiridos 65
5.3 Trabalhos Futuros 66
5.4 Conclusão 67

Referências 69

ANEXOS 71

ANEXO A – LICENÇA MIT 72

1 Introdução

1.1 Contexto e problema

A captura de fotos de momentos de lazer exerce um papel fundamental na forma como compartilhamos e experienciamos nossas vidas. As fotografias não apenas servem como lembranças pessoais valiosas, mas também como meio essencial para a socialização em ambientes digitais. Com a instauração das redes sociais e plataformas de compartilhamento, a facilidade com que compartilhamos esses momentos tornou-se uma parte integrante da nossa interação social. Tal fenômeno acentua a crescente necessidade de tecnologias e soluções que possibilitem o compartilhamento de arquivos e fotos de alta qualidade, sem comprometer sua integridade original. Dessa forma, investigar métodos eficientes e acessíveis para o compartilhamento de arquivos digitais não só atende a uma demanda tecnológica, mas também apoia a continuidade das práticas sociais que valorizam a preservação e o compartilhamento de memórias.

De acordo com REZENDE (2009), a troca de arquivos passou por uma notável evolução desde os primórdios dos computadores até o início do século XXI. Esse processo teve seu ponto de partida na centralização de dados em grandes sistemas computacionais. Posteriormente, com o surgimento de mídias removíveis como disquetes e CDs, observou-se uma descentralização gradual, que foi impulsionada mais adiante pelo crescimento das redes de computadores e da internet. Tal análise fornece um panorama histórico valioso sobre a transição do armazenamento físico para as redes digitais.

Complementando esse panorama, estudos mais recentes evidenciam como o compartilhamento de arquivos continuou evoluindo nas décadas seguintes. O surgimento de tecnologias peer-to-peer modernas, armazenamento em nuvem e soluções baseadas em blockchain permitiu o desenvolvimento de métodos descentralizados, seguros e altamente acessíveis para a transferência de dados DRIVEUPLOADER (2022). Paralelamente, abordagens que utilizam redes locais, como o DAPES (Data-Centric Peer-to-peer File Sharing), têm se destacado por oferecer alternativas viáveis mesmo em contextos com conectividade limitada ou ausente SILVA; SILVA; MORAES (2020).

Portanto, ao implementar a troca de arquivos por meio de redes locais, busca-se estabelecer uma solução prática e adaptável às necessidades atuais, garantindo que os usuários possam colaborar de maneira eficiente, mesmo em um ambiente restrito a uma localização específica. Isso amplia as possibilidades de interação e cooperação em diferentes cenários

locais, promovendo uma comunicação instantânea e descomplicada. Além disso, supera as limitações típicas de dependência da internet, oferecendo uma solução robusta para a troca de dados mesmo em situações onde a conectividade global não está disponível ou é restrita. Por consequência, a agilidade e a adaptabilidade desta aplicação local refletem o compromisso em proporcionar uma resposta adequada às demandas específicas do cenário em que ela opera.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver uma aplicação móvel multiplataforma, utilizando Flutter, para o compartilhamento eficiente e seguro de arquivos em redes locais entre dispositivos Android e iOS.

1.2.2 Objetivos específicos

Para atingir o objetivo geral de desenvolver uma aplicação de compartilhamento de arquivos eficiente e intuitiva para redes locais em dispositivos Android e iOS utilizando Flutter, estabelecem-se os seguintes objetivos específicos, que servirão de guia para o desenvolvimento e a implementação do projeto:

1.2.2.1 Analisar projetos similares Open Source ou Proprietários

Analisar comparativamente aplicações existentes de compartilhamento de arquivos, tanto de código aberto quanto proprietárias, a fim de identificar tendências, lacunas e oportunidades de inovação, além de estabelecer um ponto de partida para o desenvolvimento da solução proposta.

1.2.2.2 Definir requisitos e planejar o sistema

Levantar, documentar e validar os requisitos funcionais e não funcionais da aplicação, considerando aspectos como usabilidade, desempenho e segurança. Planejar o desenvolvimento técnico e estratégico com base nesses requisitos.

1.2.2.3 Desenvolver a aplicação

Implementar a aplicação utilizando o framework Flutter, empregando metodologias ágeis para garantir flexibilidade e adaptação contínua ao longo do processo. Priorizar a criação de uma interface intuitiva e responsiva para múltiplas plataformas móveis.

1.2.2.4 Incorporar criptografia ponta a ponta (E2EE)

Integrar mecanismos de criptografia ponta a ponta no compartilhamento de arquivos, visando garantir a segurança e a privacidade dos dados transmitidos entre dispositivos.

1.2.2.5 Testar e validar a solução

Executar testes de unidade, integração e aceitação com usuários para verificar o correto funcionamento da aplicação, identificar e corrigir falhas, e assegurar a qualidade, estabilidade e usabilidade do sistema.

1.2.2.6 Publicar a aplicação nas lojas de aplicativos

Realizar o processo de publicação da aplicação na Google Play Store e na Apple App Store, atendendo às diretrizes específicas de cada plataforma. Preparar materiais visuais e descritivos, bem como definir estratégias de lançamento e suporte pós-publicação.

1.3 Justificativa

Com o avanço da tecnologia móvel e o aumento da quantidade de dispositivos inteligentes em uso, tornou-se essencial dispor de soluções que possibilitem o compartilhamento rápido, eficiente, seguro e privado de arquivos entre diferentes plataformas. A proposta da aplicação *RB Share* surge para atender a essa demanda, com foco especial em ambientes de rede local e na interoperabilidade entre os sistemas operacionais iOS e Android.

Atualmente, muitos aplicativos que oferecem esse tipo de funcionalidade apresentam limitações significativas. É comum que o envio de arquivos dependa de conexões com servidores externos, exigindo que os dados sejam armazenados temporariamente em nuvem ou roteados por serviços de terceiros. Essa abordagem compromete diretamente a privacidade do usuário, uma vez que os arquivos passam por intermediários fora do seu controle. Além disso, esse tipo de solução demanda conexão estável com a internet, o que pode não estar disponível em todos os contextos, especialmente em locais mais remotos ou em redes corporativas restritas.

Nesse cenário, o *RB Share* apresenta uma proposta distinta: realizar todo o processo de compartilhamento diretamente entre os dispositivos, utilizando exclusivamente a rede local (como Wi-Fi ou hotspot). Isso significa que os arquivos são transferidos ponto-a-ponto, sem depender de qualquer servidor externo ou armazenamento em nuvem. Como resultado, o usuário mantém total controle sobre seus dados, reduzindo drasticamente os riscos relacionados ao vazamento de informações, acesso não autorizado ou uso indevido dos arquivos.

Um dos principais diferenciais da solução é sua capacidade de transferir arquivos de grande tamanho com alta velocidade. Como a comunicação ocorre diretamente entre os dispositivos, sem intermediários e sem necessidade de upload para servidores externos, o tempo de envio e recepção é consideravelmente reduzido, mesmo para arquivos pesados como vídeos em alta definição ou bibliotecas de fotos. Além disso, o *RB Share* garante a preservação total da qualidade dos arquivos transmitidos, ao contrário de muitos serviços populares que aplicam compressão automática, especialmente em fotos e vídeos, reduzindo sua resolução e fidelidade.

Outro aspecto relevante é a ausência completa de anúncios dentro do aplicativo. A maioria das soluções gratuitas disponíveis no mercado atualmente depende de publicidade para gerar receita, o que afeta diretamente a experiência do usuário. Pop-ups, banners e vídeos patrocinados não apenas tornam o uso do aplicativo mais lento e visualmente poluído, como também podem acarretar consumo excessivo de dados móveis e potenciais riscos à privacidade — já que muitos desses anúncios coletam informações comportamentais para fins de marketing. O *RB Share*, por sua vez, foi pensado para ser uma ferramenta simples, limpa e objetiva, eliminando completamente esse tipo de interferência.

Desenvolvido em Flutter, o aplicativo é multiplataforma e utiliza um único código-fonte para gerar versões nativas tanto para iOS quanto para Android. Isso garante maior eficiência no desenvolvimento, consistência na experiência do usuário e menor custo de manutenção, ao mesmo tempo em que assegura compatibilidade entre dispositivos com sistemas operacionais distintos — uma das grandes barreiras enfrentadas por usuários que precisam compartilhar arquivos entre Android e iOS.

O foco na simplicidade e eficiência também é um diferencial importante. O *RB Share* não tenta ser um aplicativo multifuncional cheio de recursos complexos; seu objetivo é claro: oferecer uma maneira prática, rápida e segura de compartilhar arquivos, sem distrações, propagandas ou interfaces complicadas. Essa abordagem favorece uma ampla gama de usuários, desde estudantes e professores em ambientes acadêmicos até profissionais em escritórios, participantes de eventos ou até mesmo famílias em casa.

Em resumo, o *RB Share* justifica-se como uma solução moderna, segura e adaptada às

necessidades do cotidiano digital. Ao eliminar a dependência de servidores externos, remover anúncios, preservar a qualidade dos arquivos e oferecer compatibilidade real entre plataformas, a aplicação promove um modelo de compartilhamento de arquivos mais inteligente, centrado na privacidade, praticidade e velocidade. Sua proposta atende diretamente à demanda por soluções mais confiáveis e acessíveis, tornando-se uma alternativa relevante frente às opções atuais disponíveis no mercado.

2 Fundamentação teórica

O compartilhamento de arquivos pode ser compreendido como o ato de distribuir livremente bens culturais, abrangendo desde entretenimento e arte até conteúdos gerados no meio acadêmico, por meio de uma estrutura tecnológica que se apoia na internet. De forma mais específica, envolve a duplicação e disseminação de arquivos digitais que carregam informações sobre uma ampla gama de conteúdos culturais, como livros, filmes e músicas, sem necessariamente ter autorização para tal. Esse processo é realizado gratuitamente e pode utilizar diferentes métodos de distribuição MIZUKAMI (2007).

Segundo BERGAMASCHI et al. (2012), as plataformas de mídia social na internet emergiram como um elemento significativo em várias esferas da sociedade contemporânea. Elas reúnem indivíduos com laços ou interesses compartilhados, facilitando a comunicação entre eles, a troca de uma ampla gama de informações, incluindo fotos e vídeos, e promovendo uma conexão contínua. A distribuição de informações por meio de arquivos digitais (em contraste com mensagens breves, como as do Twitter) consolidou-se como uma prática amplamente utilizada na internet. Essa distribuição pode ocorrer por meio do envio de arquivos via e-mail para múltiplos destinatários ou por meio de seu armazenamento em espaços compartilhados acessíveis pelos mesmos contatos — uma funcionalidade oferecida por serviços como Google Docs, Dropbox, OneDrive, entre outros.

Essa prática, embora controversa em alguns aspectos, reflete uma mudança significativa nas dinâmicas de acesso ao conhecimento e à cultura. Ela democratiza a distribuição de recursos culturais, permitindo que um número muito maior de pessoas tenha acesso a materiais educativos, artísticos e de entretenimento, que de outra forma poderiam ser restritos por barreiras financeiras ou geográficas. A acessibilidade é ampliada pela capacidade quase ilimitada de replicação e distribuição que a internet oferece, transformando a maneira como consumimos e interagimos com conteúdos culturais.

No entanto, o compartilhamento de arquivos também levanta questões importantes relacionadas aos direitos autorais e à sustentabilidade dos modelos econômicos tradicionais da indústria cultural. Para autores como LESSIG (2004), essa prática pode ser vista como uma forma de resistência a sistemas centralizados de distribuição de conteúdo, promovendo o acesso democrático à cultura por meio de tecnologias como o protocolo BitTorrent. Em contrapartida, instituições como a Recording Industry Association of America (2004) argumentam que a distribuição gratuita compromete a remuneração de artistas e produtores, ao permitir a circulação de obras sem a devida compensação financeira. Essa tensão tem impulsionado

o desenvolvimento de soluções tecnológicas, como sistemas de licenciamento mais flexíveis, exemplificados por iniciativas como a Creative Commons (2025), e tecnologias como o blockchain, que, segundo WATANABE et al. (2016), podem garantir a autoria e autenticidade de conteúdos digitais.

Em resumo, o compartilhamento de arquivos como fenômeno social e tecnológico destaca as tensões entre acesso livre à cultura e proteção dos direitos autorais, desafiando-nos a repensar as formas de produção, distribuição e consumo de bens culturais na era digital. À medida que avançamos, é crucial encontrar um equilíbrio que favoreça tanto a inovação quanto a justa compensação para os criadores de conteúdo, assegurando que o rico tecido cultural continue a se desenvolver e a ser acessível a todos.

2.1 Multicasting

De acordo com SAHASRABUDDHE (2000), Multicasting é a capacidade de uma rede de comunicação de aceitar uma única mensagem de um aplicativo e entregá-la a vários destinatários em diferentes locais. Um dos desafios é minimizar os recursos de rede utilizados. Por exemplo, um servidor de vídeo que deseja transmitir um filme para 1000 destinatários, se utilizasse 1000 conexões ponto a ponto, enviaria 1000 cópias do filme por um único link, desperdiçando largura de banda. Uma implementação eficiente de multicasting usa melhor a largura de banda, transmitindo uma única cópia do filme em cada link da rede. Recentemente, tem havido muita pesquisa sobre comunicação multicast. Embora existam muitas pesquisas e livros excelentes sobre vários aspectos do multicasting, há necessidade de um tutorial abrangente sobre os algoritmos de roteamento multicast e seu relacionamento com protocolos de roteamento multicast. Este trabalho apresenta um tutorial sobre dois tópicos importantes em multicasting: algoritmos de roteamento multicast e protocolos de roteamento multicast. As redes de comunicação podem ser classificadas em duas categorias: redes locais e redes de longa distância.

Multicasting tornou-se um foco importante na área de redes, especialmente com o crescimento da Internet e a demanda por aplicativos como videoconferências. Recentemente, surgiram redes ad hoc sem fio dinâmicas que interconectam usuários móveis para aplicações como recuperação de desastres e computação colaborativa. No entanto, protocolos multicast tradicionais, como DVMRP e PIM, não se adaptam bem a redes ad hoc dinâmicas, devido à necessidade de ajuste contínuo das estruturas de árvore multicast. A largura de banda limitada, a potência restrita e a mobilidade dos hosts tornam o design de protocolos multicast desafiador. LEE MARIO GERLA (2007)

2.2 Publicação de Aplicativos em Plataformas Mobile: Google Play Store e App Store

A publicação de aplicativos em plataformas mobile como a Google Play Store e a App Store é um processo crucial para desenvolvedores que desejam disponibilizar suas aplicações ao público global. Cada plataforma possui seus próprios procedimentos, requisitos e diretrizes, que devem ser seguidos rigorosamente para garantir que o aplicativo seja aceito e oferecido aos usuários. A seguir, discutimos os principais passos para publicar um aplicativo nessas duas plataformas, com base nas diretrizes fornecidas pelo Google, Google Play Support, (2024) e pela Apple. Apple, (2024a)

2.2.1 Preparação do Aplicativo

Tanto para a Google Play Store quanto para a App Store, é essencial que o aplicativo esteja completamente desenvolvido, testado e conforme as políticas de cada plataforma. Isso inclui assegurar que o aplicativo seja estável, funcional e proporcione uma experiência de usuário satisfatória. Além disso, o conteúdo do aplicativo deve respeitar as normas de segurança e privacidade, especialmente em relação à coleta e manejo de dados dos usuários.

2.2.2 Criação de Contas de Desenvolvedor

Para publicar um aplicativo em ambas as plataformas, o desenvolvedor precisa criar contas específicas:

- **Google Play Console:** Para publicar na Google Play Store, é necessário ter uma conta de desenvolvedor no Google Play Console. Há uma taxa única de $25 para a criação desta conta, que permite ao desenvolvedor publicar aplicativos ilimitadamente sob essa conta. Essa taxa cobre o acesso às ferramentas de desenvolvimento, suporte e a capacidade de gerenciar aplicativos na plataforma.
- **Apple Developer Program:** Para a App Store, é necessário estar inscrito no Apple Developer Program, que custa $99 por ano. Essa taxa deve ser renovada anualmente e dá ao desenvolvedor acesso às ferramentas de desenvolvimento da Apple, como TestFlight para testes beta, além da capacidade de publicar aplicativos na App Store. A Apple também oferece um programa específico para organizações, o Apple Developer Enterprise Program, que custa $299 por ano sendo destinado a empresas que desejam distribuir aplicativos internamente, sem passar pela App Store.

2.2.3 Configuração de Informações e Metadados

Após a criação das contas, o desenvolvedor deve configurar as informações do aplicativo em ambas as plataformas. Isso inclui:

- **Nome e Descrição:** Definir um título e uma descrição detalhada que expliquem as funcionalidades e os benefícios do aplicativo.
- **Imagens e Ícones:** Fornecer ícones de alta resolução, capturas de tela e, opcionalmente, vídeos promocionais que demonstrem o uso do aplicativo.
- **Classificação de Conteúdo:** Preencher questionários para determinar a classificação etária adequada do aplicativo.
- **Política de Privacidade:** Incluir uma política de privacidade clara, especialmente se o aplicativo coleta dados pessoais dos usuários.

2.2.4 Testes e Submissão para Revisão

É recomendável realizar testes beta antes do lançamento oficial do aplicativo. Na App Store, o TestFlight é amplamente utilizado para distribuir versões beta para testadores selecionados, que podem dar feedback valioso. Já na Google Play Store, é possível configurar canais de testes alfa e beta para um grupo limitado de usuários.

Após os testes, o aplicativo pode ser submetido para revisão. Ambos os sistemas possuem processos rigorosos de revisão, onde o aplicativo é avaliado para garantir que cumpre todas as diretrizes da plataforma. A Apple, em particular, é conhecida por sua revisão minuciosa, que pode levar alguns dias para ser concluída.

2.2.5 Lançamento e Pós-Publicação

Uma vez aprovado, o aplicativo é lançado nas respectivas lojas e fica disponível para download pelos usuários. Após o lançamento, é fundamental que o desenvolvedor monitore o desempenho do aplicativo, responda a avaliações e lance atualizações regulares para corrigir bugs e adicionar novas funcionalidades. Além disso, é importante estar sempre atento a quaisquer atualizações nas políticas da Google Play Store e da App Store para garantir a continuidade do aplicativo nessas plataformas.

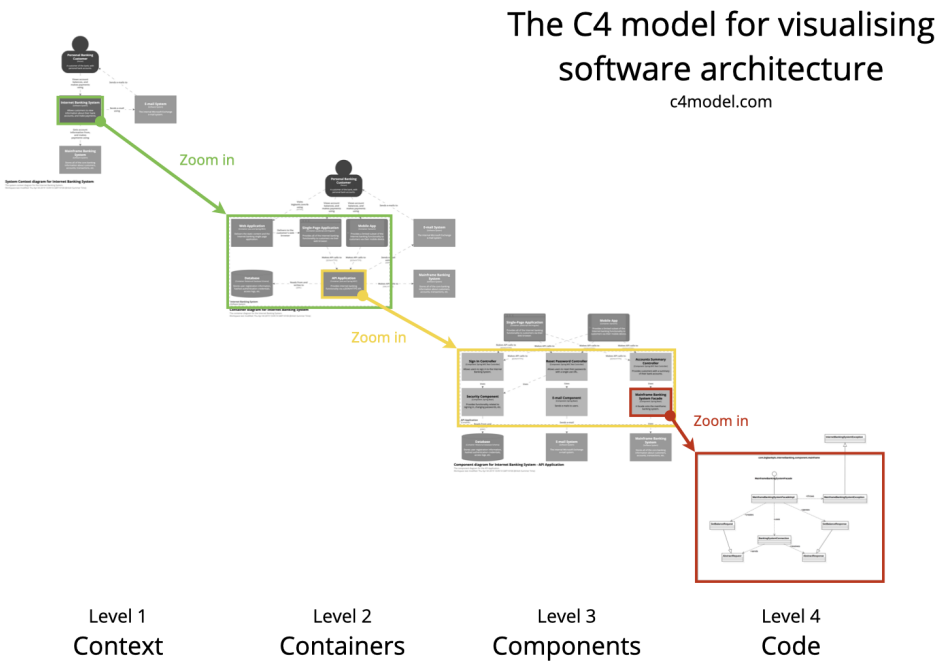
2.2.6 Conclusão da publicação

Publicar um aplicativo na Google Play Store e na App Store envolve um conjunto de etapas que exigem atenção aos detalhes e conformidade com as diretrizes específicas de cada plataforma. Além dos requisitos técnicos, os desenvolvedores devem estar cientes dos custos envolvidos em cada plataforma, sendo $25 como uma taxa única para a Google Play Store e $99 por ano para a App Store. Seguir rigorosamente os procedimentos recomendados e realizar testes abrangentes são passos cruciais para garantir o sucesso na publicação e na manutenção de um aplicativo nessas plataformas.

2.3 C4 Model para Arquitetura de Software

O C4 Model é uma abordagem estruturada para descrever a arquitetura de sistemas de software. Desenvolvido por Simon Brown, o modelo simplificará a comunicação e o entendimento de arquiteturas complexas por meio de uma série de diagramas que oferecem diferentes níveis de abstração. Esses níveis são Contexto, Contêineres, Componentes e Código (ou Classes), cada um focando em aspectos específicos do sistema. BROWN, (2020)

Figura 1 – Visão geral do modelo C4 para visualização de arquitetura de software.



Fonte: BROWN (2024). Disponível em: https://c4model.com/

2.3.1 Diagrama de Contexto

O Diagrama de Contexto é o nível mais alto de abstração no C4 Model. Ele oferece uma visão geral do sistema e seu ambiente, mostrando como o sistema interage com usuários externos, sistemas externos e outras entidades. O objetivo deste diagrama é comunicar a essência do sistema a um público não técnico, permitindo que todos compreendam o escopo geral e as interações principais.

- **Elementos:** Usuários finais, sistemas externos, e o sistema em foco.
- **Objetivo:** Fornecer uma visão geral de alto nível do sistema e suas interações externas.

2.3.2 Diagrama de Contêineres

O Diagrama de Contêineres foca nos principais contêineres de software que compõem o sistema, como aplicativos web, aplicativos móveis, bancos de dados, e microsserviços. Ele mostra como esses contêineres interagem entre si e com o ambiente externo. Este diagrama é útil para equipes técnicas, ao ajudar a entender a distribuição de responsabilidades e a comunicação entre diferentes partes do sistema.

- **Elementos:** Contêineres de software, interações entre contêineres e entidades externas.
- **Objetivo:** Descrever a arquitetura de alto nível, destacando como o sistema é decomposto em partes executáveis.

2.3.3 Diagrama de Componentes

O Diagrama de Componentes desce um nível de abstração para detalhar os principais componentes internos de cada contêiner. Ele ilustra como os componentes em um contêiner colaboram para fornecer a funcionalidade do sistema. Este diagrama é valioso para desenvolvedores e arquitetos que precisam entender a estrutura interna de cada contêiner.

- **Elementos:** Componentes dentro de contêineres e suas interações.
- **Objetivo:** Fornecer uma visão detalhada dos blocos de construção que compõem cada contêiner.

2.3.4 Diagrama de Código (ou Classes)

O Diagrama de Código é o nível mais baixo de abstração no C4 Model, focando nos detalhes de implementação. Ele pode incluir diagramas de classes, sequência de chamadas, ou

qualquer outro artefato que detalha a estrutura interna dos componentes. Este nível é mais relevante para desenvolvedores que trabalham diretamente no código-fonte.

- **Elementos:** Classes, interfaces, métodos, e relacionamentos.
- **Objetivo:** Fornecer detalhes específicos de implementação que suportam os componentes descritos no diagrama anterior.

2.4 Benefícios do C4 Model

O C4 Model traz uma série de benefícios ao processo de documentação e comunicação de arquiteturas de software:

- **Clareza:** Ao utilizar diferentes níveis de abstração, o C4 Model permite que arquitetos e desenvolvedores comuniquem as ideias complexas de forma clara e compreensível.
- **Flexibilidade:** Os diagramas podem ser adaptados para diferentes audiências, desde stakeholders não técnicos até desenvolvedores que precisam de detalhes específicos.
- **Padronização:** O uso de um conjunto padronizado de diagramas ajuda a manter a consistência na documentação e facilita a manutenção ao longo do tempo.

O C4 Model é uma ferramenta poderosa para arquitetos de software que buscam comunicar a estrutura e o funcionamento de sistemas complexos de forma clara e eficiente. Ao fornecer diferentes níveis de abstração, ele permite que informações relevantes sejam acessíveis a diferentes partes interessadas, desde a equipe técnica até os stakeholders não técnicos. Este modelo se tornou uma prática recomendada em muitas organizações por sua simplicidade e eficácia na documentação de arquiteturas de software.

2.5 Trabalhos Similares

Durante a pesquisa, foram identificadas diversas aplicações que oferecem funcionalidades de compartilhamento de arquivos entre dispositivos móveis. Algumas dessas soluções compartilham características em comum com o *RB Share*, como o uso de rede local ou a compatibilidade com múltiplas plataformas. No entanto, muitas ainda apresentam limitações que impactam negativamente a experiência do usuário, como a presença excessiva de anúncios, dependência de internet ou interfaces sobrecarregadas. A seguir, são apresentados alguns desses aplicativos, juntamente com suas similaridades e limitações:

2.5.1 Aplicativos Proprietários

- **SHAREit**[1]: Assim como o *RB Share*, o SHAREit permite transferências rápidas de arquivos entre dispositivos, utilizando redes locais. No entanto, apresenta uma quantidade excessiva de anúncios, incluindo pop-ups e conteúdos patrocinados, que comprometem a usabilidade. Além disso, sua interface incorpora funcionalidades não relacionadas ao compartilhamento de arquivos, como reprodutor de mídia e jogos, o que pode confundir o usuário.
- **Xender**[2]: Compartilha com o *RB Share* o suporte a múltiplas plataformas e o uso de redes locais para transferência de arquivos. Contudo, usuários relatam dificuldades de conexão entre dispositivos diferentes e a presença de anúncios e funcionalidades extras que desviam o foco da aplicação.
- **ShareMe**[3]: Similar ao *RB Share*, o ShareMe oferece uma interface limpa e é livre de anúncios, focando exclusivamente no compartilhamento de arquivos. Porém, sua limitação está na compatibilidade, sendo voltado principalmente para dispositivos Android, o que restringe o compartilhamento entre plataformas como iOS.

2.5.2 Aplicativos de Código Aberto

- **NetShare**[4]: Assim como o *RB Share*, utiliza rede local para o compartilhamento de arquivos e é desenvolvido em Flutter, promovendo compatibilidade entre plataformas. Entretanto, por estar em desenvolvimento ativo, ainda pode apresentar instabilidades e não oferece uma experiência tão estável quanto a do *RB Share*.
- **Bytes**[5]: Também desenvolvido em Flutter, compartilha a proposta multiplataforma do *RB Share*. No entanto, o projeto está depreciado e desatualizado, o que pode resultar em problemas de compatibilidade com versões recentes do Flutter e uma experiência inferior ao usuário.

2.5.3 Diferenciais do RB SHARE

O *RB Share* foi concebido para aproveitar as melhores características observadas nas soluções existentes e ao mesmo tempo superar suas principais limitações. Seus diferenciais incluem:

[1] <https://play.google.com/store/apps/details?id=com.lenovo.anyshare.gps>
[2] <https://www.xender.com/>
[3] <https://play.google.com/store/apps/details?id=com.xiaomi.midrop>
[4] <https://github.com/huynguyennovem/netshare>
[5] <https://github.com/alanrs2020/flutter-filesharing-app>

- **Ausência de anúncios**: Garante uma experiência limpa e fluida, sem distrações ou conteúdos patrocinados.
- **Compartilhamento via rede local**: Oferece transferências rápidas e seguras, sem necessidade de conexão com a internet.
- **Compatibilidade entre plataformas**: Desenvolvido em Flutter, funciona em dispositivos Android e iOS.
- **Foco na simplicidade**: Interface enxuta, com foco exclusivo no compartilhamento de arquivos, sem funcionalidades desnecessárias.

Em resumo, o *RB Share* incorpora os pontos fortes das soluções analisadas — como desempenho, simplicidade e compatibilidade — enquanto elimina aspectos negativos como anúncios invasivos, instabilidades e limitações de plataforma, consolidando-se como uma alternativa moderna, eficiente e centrada na experiência do usuário.

3 Materiais e métodos

3.1 Flutter e Dart

Flutter é um framework de código aberto desenvolvido pelo Google, que permite a criação de aplicações nativas para diversas plataformas a partir de uma única base de código Flutter, (2024). Ele se destaca por sua capacidade de gerar aplicações de alta performance, com uma experiência de usuário suave e consistente. Além disso, o Flutter oferece uma vasta gama de widgets personalizáveis, que facilitam a criação de interfaces modernas e intuitivas.

A linguagem Dart, também desenvolvida pelo Google, foi escolhida por ser a linguagem nativa do Flutter. Dart é uma linguagem otimizada para desenvolvimento de interfaces de usuário e é conhecida por sua facilidade de aprendizado, desempenho eficiente e forte tipagem. A escolha por Dart também se justifica pela sua capacidade de compilar para código nativo, o que contribui para a performance elevada das aplicações desenvolvidas em Flutter.

Outro fator relevante na escolha do Flutter é a sua comunidade ativa e crescente, além do suporte contínuo fornecido pelo Google. A documentação abrangente e os recursos disponíveis tornam o Flutter uma escolha atraente para desenvolvedores que buscam criar aplicações multiplataforma de maneira eficiente e com alto padrão de qualidade.

3.2 Ferramentas Utilizadas

Durante o desenvolvimento e distribuição do aplicativo, utilizamos diversas ferramentas que foram fundamentais para o sucesso do projeto. A seguir, discutimos as principais ferramentas utilizadas e os motivos para suas escolhas.

3.2.1 Visual Studio Code (VSCode)

O Visual Studio Code (VSCode) foi utilizado como o principal editor de código-fonte. Desenvolvido pela Microsoft, o VSCode é um editor leve e poderoso, com suporte para uma ampla gama de linguagens de programação e extensões, que facilitam a integração com tecnologias como Flutter e Dart. Visual Studio Code, (2024). A escolha do VSCode foi motivada pela sua interface amigável, suporte robusto para depuração e a vasta biblioteca de extensões que permite personalizar o ambiente de desenvolvimento conforme as necessidades

do projeto.

3.2.2 Xcode

Para o desenvolvimento de funcionalidades específicas para iOS e a integração com a App Store, utilizamos o Xcode. O Xcode é o ambiente de desenvolvimento integrado (IDE) oficial da Apple para o desenvolvimento de aplicativos para iOS, macOS, watchOS e tvOS Apple Developer, (2024). Ele oferece ferramentas avançadas para desenvolvimento, depuração e teste de aplicativos, além de ser essencial para a criação de builds e a submissão de aplicativos na App Store.

3.2.3 Android Studio

O Android Studio foi utilizado para o desenvolvimento de funcionalidades específicas para a plataforma Android. É o IDE oficial do Google para o desenvolvimento de aplicativos Android, oferecendo ferramentas completas para o desenvolvimento, depuração e teste de aplicativos Google, (2024). O Android Studio é amplamente adotado pela comunidade de desenvolvedores Android devido à sua integração nativa com o sistema operacional e as ferramentas de emulação e desempenho que ele oferece.

3.2.4 Transporter

O Transporter foi utilizado para a distribuição do aplicativo na App Store. É uma ferramenta fornecida pela Apple que facilita o envio de arquivos binários e outros conteúdos para a App Store Apple, (2024b). O Transporter é especialmente útil para gerenciar o upload de grandes arquivos e para automatizar partes do processo de distribuição, garantindo que os aplicativos estejam disponíveis na App Store de maneira eficiente.

3.3 Métodos

Nesta seção, abordaremos as práticas de versionamento e controle de código-fonte adotadas durante o desenvolvimento do projeto, bem como as ferramentas utilizadas para gerenciar o fluxo de trabalho.

3.3.1 Versionamento

O versionamento de código é uma prática essencial em projetos de software, permitindo o rastreamento de mudanças, a colaboração entre desenvolvedores e a manutenção de um histórico detalhado de todas as alterações realizadas no projeto. Durante o desenvolvimento deste projeto, utilizamos o sistema de controle de versão Git, juntamente com a plataforma GitHub, para hospedar e colaborar no código.

3.3.2 Git e GitHub

O Git é um sistema de controle de versão distribuído que permite que múltiplos desenvolvedores trabalhem simultaneamente em um projeto, facilitando a integração das mudanças de forma segura e organizada. O GitHub, por sua vez, é uma plataforma baseada na web que fornece repositórios Git, além de ferramentas adicionais para gerenciamento de projetos, colaboração e integração contínua.

Para gerenciar o repositório do projeto, seguimos as boas práticas de commits frequentes, utilização de branches para novas funcionalidades e correções de bugs, e integração das mudanças por meio de pull requests, garantindo revisões de código e discussões antes de qualquer alteração ser integrada à branch principal.

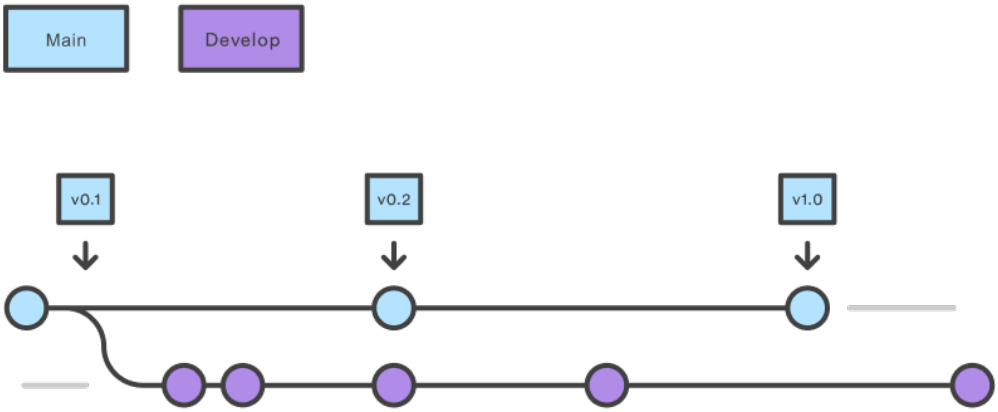
3.3.3 GitFlow Simplificado

Para estruturar o fluxo de desenvolvimento de forma eficiente, optei por uma estratégia de GitFlow simplificada, adequada para um ambiente de trabalho solo. Esse modelo define uma estrutura clara para gerenciar o desenvolvimento de software, separando-o em duas branches principais:

- **Main Branch:** Contém o código de produção, que deve sempre estar em um estado estável.
- **Development Branch:** É a branch onde o desenvolvimento ativo acontece, integrando novas funcionalidades antes de serem lançadas na main.

A Figura 2 ilustra o fluxo de trabalho GitFlow Simplificado, destacando as duas branches principais e o fluxo de integração de mudanças. Esta abordagem ajudou a manter o código organizado, minimizando conflitos e facilitando o gerenciamento de lançamentos, mesmo em um cenário de desenvolvimento solo.

Figura 2 – Diagrama ilustrando o fluxo GitFlow Simplificado. Fonte: ATLASSIAN



Fonte: Autoria própria (2024).

3.3.4 Imagens

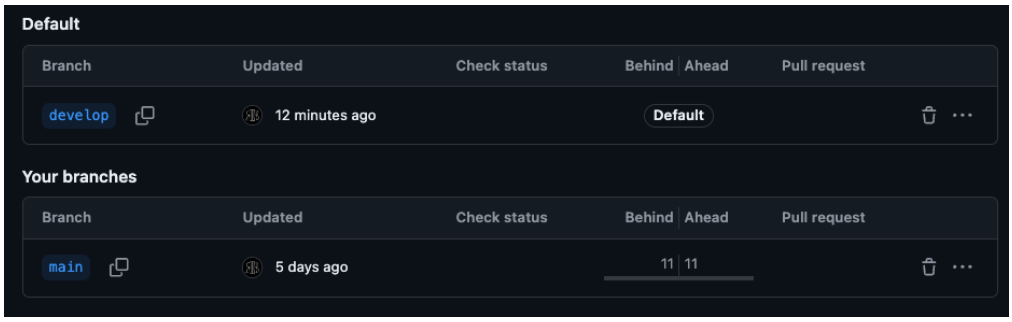
Para ilustrar os conceitos mencionados, incluímos imagens que representam o fluxo de trabalho com Git e GitHub, além do modelo GitFlow Simplificado. Essas imagens fornecem uma visualização clara de como o processo de desenvolvimento foi gerenciado.

O uso de Git e GitHub, aliado à estratégia de GitFlow Simplificado, permitiu uma gestão eficiente do código-fonte e facilitou o desenvolvimento do projeto. A prática de versionamento adequada é essencial para garantir a integridade do código e a capacidade de rastrear e reverter mudanças quando necessário.

3.4 Requisitos

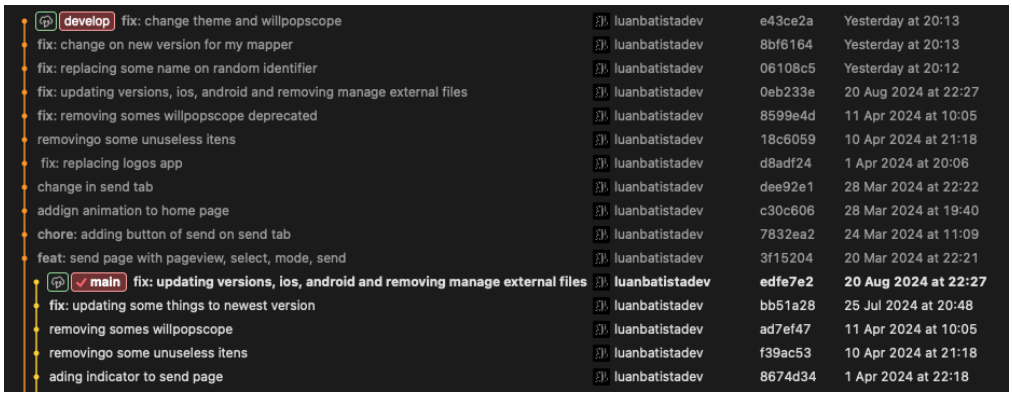
Nesta seção, apresentam-se os requisitos que fundamentaram o desenvolvimento da aplicação, organizados em duas categorias principais: requisitos funcionais e requisitos não funcionais. Os requisitos funcionais especificam as funcionalidades que o sistema deve oferecer para atender às necessidades dos usuários e permitir o cumprimento dos objetivos do projeto. Já os requisitos não funcionais descrevem as propriedades de qualidade que o sistema deve possuir, como desempenho, usabilidade, portabilidade e segurança. Essa distinção permite uma compreensão mais clara do escopo do sistema e serve como base para o planejamento, implementação e validação do software.

Figura 3 – Controle de versão no GitHub com as branches `main` e `develop`, e o histórico de commits associado.



Fonte: Autoria própria (2024).

Figura 4 – Visualização das branches `main` e `develop` no GitHub (acima), e do histórico de commits dessas ramificações (abaixo).



Fonte: Autoria própria (2024).

3.4.1 Requisitos Funcionais

Os requisitos funcionais descrevem as funcionalidades e comportamentos específicos que o sistema deve implementar. Eles são essenciais para garantir que o sistema atenda às necessidades dos usuários finais e alcance os objetivos propostos.

Tabela 1 – Requisitos Funcionais do Aplicativo de Compartilhamento de Arquivos na Rede Local (*RB Share*)

Identificador	**Requisito**
RF: 1	O aplicativo deve detectar automaticamente outros dispositivos conectados à mesma rede local que também tenham o aplicativo instalado.
RF: 2	A detecção deve ser em tempo real para facilitar a conexão imediata entre dispositivos.

Continua na próxima página

Tabela 1 – Requisitos Funcionais do Aplicativo de Compartilhamento de Arquivos na Rede Local (*RB Share*)

Identificador	**Requisito**
RF: 3	O aplicativo deve permitir que os usuários enviem arquivos (documentos, fotos, vídeos, etc.) para outros dispositivos na rede local.
RF: 4	O aplicativo deve suportar múltiplos tipos de arquivos e formatos.
RF: 5	Deve ser possível selecionar vários arquivos para envio simultâneo.
RF: 6	O aplicativo deve fornecer uma interface para os usuários visualizarem os arquivos recebidos.
RF: 7	Os arquivos de mídia (fotos e vídeos) devem ser exibidos diretamente na interface do aplicativo.
RF: 8	O aplicativo deve exibir o status de transferência dos arquivos em tempo real, incluindo progresso, tempo estimado e velocidade.
RF: 9	Deve ser possível pausar, retomar e cancelar transferências em andamento.
RF: 10	O aplicativo deve permitir o compartilhamento de arquivos grandes, como vídeos de alta qualidade, sem limites estritos de tamanho, respeitando as capacidades da rede.
RF: 11	O aplicativo deve garantir que as transferências de arquivos sejam seguras, utilizando criptografia para proteger os dados durante a transmissão.
RF: 12	O aplicativo deve notificar o usuário quando um novo dispositivo é detectado, quando um arquivo é enviado ou recebido, e quando uma transferência é concluída.
RF: 13	O aplicativo deve permitir que os usuários organizem os arquivos recebidos em pastas ou categorias.
RF: 14	O aplicativo deve ser compatível com diferentes sistemas operacionais, como Android, iOS, Windows, macOS e Linux, permitindo o compartilhamento entre dispositivos variados.
RF: 15	O aplicativo deve manter um registro das transferências realizadas, permitindo que os usuários revisitem ou reenviem arquivos facilmente.
RF: 16	O aplicativo deve permitir o compartilhamento de arquivos mesmo sem conexão com a internet, utilizando apenas a rede local.

Continua na próxima página

Tabela 1 – Requisitos Funcionais do Aplicativo de Compartilhamento de Arquivos na Rede Local (*RB Share*)

Identificador	**Requisito**
RF: 17	O aplicativo deve permitir que os usuários selecionem arquivos em outros aplicativos (como gerenciadores de arquivos, aplicativos de fotos, e-mail, etc.) e enviem esses arquivos diretamente para o aplicativo de compartilhamento para serem compartilhados com outros dispositivos na rede local.
RF: 18	Deve ser possível enviar múltiplos arquivos de uma só vez a partir de outros aplicativos.
RF: 19	O aplicativo deve oferecer suporte a extensões de compartilhamento nativas do sistema operacional, facilitando a transferência direta de arquivos.

Tabela 2 – Requisitos Futuros do Aplicativo de Compartilhamento de Arquivos na Rede Local (*RB Share*)

Identificador	**Requisitos Futuros**
RF: F1	Os usuários devem poder configurar manualmente aspectos da rede local, como selecionar um canal específico ou ajustar a largura de banda disponível para o compartilhamento.
RF: F2	Deve ser possível enviar arquivos diretamente para outros aplicativos instalados no dispositivo, como editores de imagem ou gerenciadores de arquivos.

3.4.2 Requisitos Não Funcionais

Os requisitos não funcionais referem-se às características de qualidade que o sistema deve possuir. Eles são cruciais para garantir o desempenho, a usabilidade e a segurança do sistema.

Tabela 3 – Requisitos Não Funcionais do Aplicativo de Compartilhamento de Arquivos na Rede Local (*RB Share*)

Identificador	**Requisito Não Funcional**
RNF: 1	O aplicativo deve suportar a transferência de arquivos de até 2 GB sem degradação significativa na performance.
RNF: 2	A interface do usuário deve ser intuitiva e fácil de navegar, permitindo que novos usuários compreendam como utilizar o aplicativo sem a necessidade de um tutorial extenso.
RNF: 3	O aplicativo deve ser compatível com as versões mais recentes dos principais sistemas operacionais: Android 10+, iOS 13+, Windows 10+, macOS 10.15+ e Linux (Kernel 5.4+).
RNF: 4	O aplicativo deve funcionar corretamente em dispositivos com diferentes tamanhos de tela, incluindo smartphones, tablets e desktops.
RNF: 5	O aplicativo deve ser projetado para permitir a adição de novas funcionalidades sem a necessidade de reestruturações significativas.
RNF: 6	O aplicativo deve ser facilmente portável para novas plataformas, com o mínimo de alterações necessárias no código.

A definição clara dos requisitos funcionais e não funcionais é essencial para orientar o desenvolvimento do projeto e garantir que o sistema final atenda às expectativas dos usuários e stakeholders. Esses requisitos foram cuidadosamente elaborados para cobrir tanto as funcionalidades esperadas quanto as qualidades de serviço necessárias para o sucesso do sistema.

3.5 Arquitetura de Software

A arquitetura de software define a estrutura e organização do sistema, determinando como os componentes se relacionam e interagem entre si. Neste projeto, optamos por seguir o padrão de arquitetura MVC (Model-View-Controller), que é amplamente utilizado no desenvolvimento de aplicações, especialmente naquelas com interfaces de usuário complexas.

3.5.1 Modelo MVC

O padrão de arquitetura MVC divide a aplicação em três componentes principais, cada um com responsabilidades distintas:

- **Model (Modelo):** O componente de modelo representa a lógica de negócios e os dados da aplicação. Ele é responsável por acessar o banco de dados, executar regras de negócio e notificar a View quando os dados são alterados. No contexto deste projeto, o Model

gerencia as informações sobre produtos, usuários e transações.

- **View (Visão):** A View é a camada responsável por apresentar os dados ao usuário. Ela observa o Model e atualiza a interface de usuário em resposta a mudanças nos dados. No projeto, a View está implementada utilizando Flutter, oferecendo uma interface de usuário intuitiva e responsiva.
- **Controller (Controlador):** O Controller atua como um intermediário entre o Model e a View. Ele processa as entradas do usuário, interage com o Model para atualizar os dados e seleciona a View apropriada para apresentar os resultados ao usuário. O Controller é responsável por coordenar a lógica de navegação e interação dentro da aplicação.

A arquitetura MVC oferece diversos benefícios, incluindo a separação clara de responsabilidades, facilitando a manutenção e escalabilidade da aplicação. Além disso, essa arquitetura permite o desenvolvimento paralelo das camadas, com equipes trabalhando simultaneamente no Model, View e Controller.

3.5.2 C4 Model

Para documentar a arquitetura de software de forma mais detalhada, também utilizamos o C4 Model, que oferece uma visão clara e organizada dos diferentes níveis de abstração da arquitetura:

- **Contexto:** Mostra o sistema em seu ambiente, destacando as interações com usuários e sistemas externos.

Figura 5 – Diagrama de Contexto do C4 Model ilustrando interação do sistema com usuários e sistemas externos.

Fonte: Autoria própria (2024).

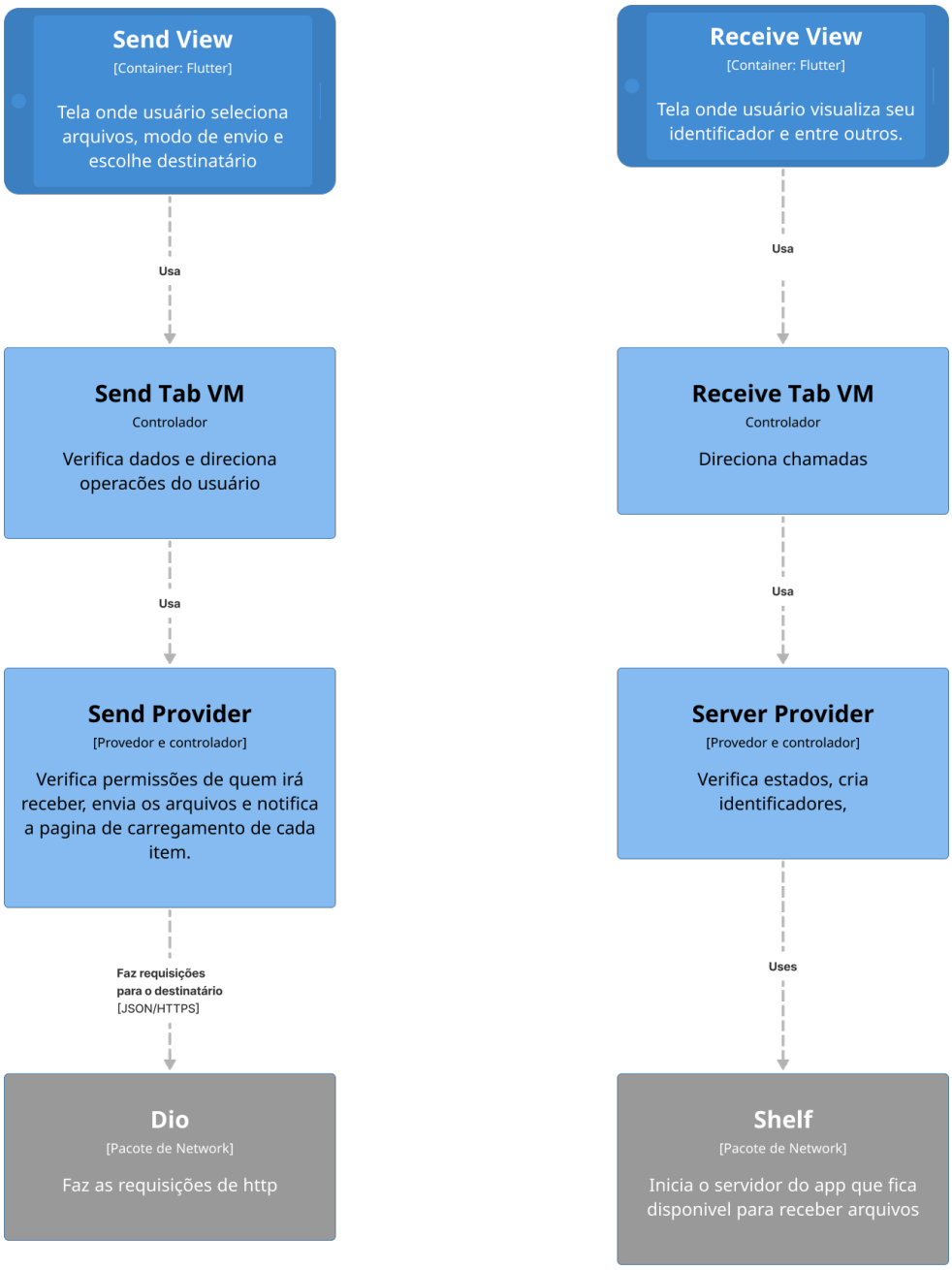
- **Contêineres:** Descreve os principais contêineres de software que compõem o sistema, como servidores, bancos de dados e microsserviços.

Figura 6 – Diagrama de Container do C4 Model ilustrando uma interação principal do sistema com usuários e sistemas externos.

Fonte: Autoria própria (2024).

- **Componentes:** Detalha os componentes internos de cada contêiner, mostrando como colaboram para fornecer funcionalidades específicas.

Figura 7 – Diagrama de Componentes do C4 Model ilustrando uma interação principal do sistema com usuários e sistemas externos.



Fonte: Autoria própria (2024).

- **Código (ou Classes):** Foca nos detalhes de implementação, como diagramas de classes e sequências de chamadas.

A escolha da arquitetura MVC, combinada com a documentação detalhada utilizando o C4 Model, proporcionou uma base sólida para o desenvolvimento do sistema. Essa combinação de abordagens permitiu uma separação clara de responsabilidades e uma visão estruturada da arquitetura, facilitando a manutenção e evolução da aplicação. ARCHITECTURE, (2022)

3.6 Implementação do Multicast

A implementação do multicast no aplicativo foi realizada utilizando as funcionalidades nativas do Flutter, que permite que dispositivos anunciem sua presença na rede local. O processo envolve o uso de *sockets* User Datagram Protocol (UDP) para enviar e receber pacotes de dados em um grupo multicast, facilitando a comunicação entre dispositivos na mesma rede.

Quadro 3.1 – Função para obtenção de sockets UDP multicast

```
Future<List<_SocketResult>> _getSockets(String multicastGroup, [int? port
   ]) async {
  final interfaces = await NetworkInterface.list();
  final sockets = <_SocketResult>[];
  for (final interface in interfaces) {
    try {
      final socket = await RawDatagramSocket.bind(InternetAddress.anyIPv4,
          port ?? 0);
      socket.joinMulticast(InternetAddress(multicastGroup), interface);
      sockets.add(_SocketResult(interface, socket));
    } catch (e) {
      _logger.warning(
        'Could not bind UDP multicast port (ip: ${interface.addresses.map
           ((a) => a.address).toList()}, group: $multicastGroup, port:
           $port)',
        e,
      );
    }
  }

  return sockets;
}
```

Inicialmente, o aplicativo configura *sockets* associados a cada interface de rede disponível, permitindo que cada socket se junte a um grupo multicast. Isso possibilita que os dispositivos na rede local possam se comunicar entre si de forma eficiente.

Quadro 3.2 – Processamento de pacotes multicast recebidos

```
final sockets = await _getSockets(settings.multicastGroup, settings.port);
for (final socket in sockets) {
  socket.socket.listen((_) {
    final datagram = socket.socket.receive();
    if (datagram == null) {
      return;
    }

    try {
      final dto = MulticastDto.fromJson(jsonDecode(utf8.decode(datagram.
          data)));
      if (dto.fingerprint == fingerprint) {
        return;
      }

      final ip = datagram.address.address;
      final peer = dto.toDevice(ip, settings.port, settings.https);
      streamController.add(peer);
      if ((dto.announcement == true || dto.announce == true) &&
          _ref.read(serverProvider) != null) {
        // only respond when server is running
        _answerAnnouncement(peer);
      }
    } catch (e) {
      _ref.notifier(discoveryLoggerProvider).addLog(e.toString());
    }
  });
}
```

O aplicativo então passa a escutar as mensagens multicast para detectar a presença de outros dispositivos. Quando uma mensagem é recebida, os dados são processados para identificar novos dispositivos na rede, que são então adicionados ao fluxo de peers ativos, permitindo interações com outros dispositivos.

Quadro 3.3 – Resposta a anúncios de dispositivos na rede

```
/// Responds to an announcement.
Future<void> _answerAnnouncement(Device peer) async {
  final settings = _ref.read(settingsProvider);

  try {
    // Answer with TCP
    await _ref.read(dioProvider).discovery.post(
```

```
          ApiRoute.register.target(peer),
          data: _getRegisterDto().toJson(),
        );
    _ref.notifier(discoveryLoggerProvider).addLog(
        '[RESPONSE/TCP] Announcement of ${peer.alias} (${peer.ip}, model:
           ${peer.deviceModel}) via TCP',);
  } catch (e) {
    // Fallback: Answer with UDP
    final sockets = await _getSockets(settings.multicastGroup);
    final dto = _getMulticastDto(announcement: false);
    for (final socket in sockets) {
      try {
        socket.socket.send(dto, InternetAddress(settings.multicastGroup),
           settings.port);
        socket.socket.close();
      } catch (e) {
        _ref.notifier(discoveryLoggerProvider).addLog(e.toString());
      }
    }
    _ref.notifier(discoveryLoggerProvider).addLog(
        '[RESPONSE/UDP] Announcement of ${peer.alias} (${peer.ip}, model:
           ${peer.deviceModel}) with UDP because TCP failed',);
  }
}
```

Periodicamente, o aplicativo envia anúncios na rede para informar outros dispositivos de sua presença. Esses anúncios garantem que todos os membros do grupo multicast estejam cientes dos dispositivos ativos na rede local, permitindo uma comunicação contínua e eficiente entre eles.

O uso do multicast facilita a descoberta automática de dispositivos na rede sem a necessidade de configuração manual, tornando o processo de comunicação mais fluido e intuitivo para o usuário final. A escolha do Flutter para essa implementação mostrou-se eficaz, permitindo a criação de uma solução robusta e integrada para o gerenciamento de dispositivos em uma rede local.

3.7 Licença de uso

Este projeto está licenciado sob a MIT License, uma das licenças de software mais permissivas e amplamente utilizadas na comunidade de desenvolvimento de software. A MIT License permite que o software seja reutilizado, modificado e distribuído, tanto em projetos

de código aberto quanto em projetos proprietários, com poucas restrições.

3.7.1 Detalhes da MIT License

A MIT License permite que qualquer pessoa que obtenha uma cópia do software e dos arquivos de documentação associados tenha permissão para usar o software sem restrições, incluindo os direitos de usar, copiar, modificar, mesclar, publicar, distribuir, sublicenciar e/ou vender cópias do software, bem como permitir que pessoas a quem o software seja fornecido façam o mesmo.

As únicas condições impostas pela MIT License são:

- O aviso de copyright original e esta permissão devem ser incluídos em todas as cópias ou partes substanciais do software.
- O software é fornecido "como está", sem garantia de qualquer tipo, expressa ou implícita, incluindo, mas não se limitando a, garantias de comercialização, adequação a um propósito específico e não infração. Em nenhum caso os autores ou detentores de direitos autorais serão responsáveis por qualquer reclamação, dano ou outra responsabilidade, seja em uma ação de contrato, delito ou de outra forma, decorrente de ou em conexão com o software ou o uso ou outras negociações no software.

3.7.2 Vantagens da MIT License

As principais vantagens de usar a MIT License incluem:

- Simplicidade e Clareza: A MIT License é curta e fácil de entender, tanto para desenvolvedores quanto para usuários.
- Flexibilidade: Ela permite que o software seja usado em projetos de código aberto e proprietários, sem exigir que o código derivado seja licenciado da mesma forma.
- Amplamente Aceita: É uma das licenças mais populares, aceita pela maioria das comunidades de software livre e open source, bem como por empresas comerciais.

3.7.3 Texto Completo da Licença MIT

O texto completo da licença MIT está disponível em anexo.

A MIT License oferece uma excelente opção para desenvolvedores que desejam compartilhar seu código com a comunidade, permitindo um amplo uso e modificação, ao mesmo tempo que limita a responsabilidade legal dos autores.

Sendo assim, ao adotar a MIT License, garantimos que nosso software possa ser amplamente utilizado e modificado, incentivando a inovação e a colaboração, ao mesmo tempo, em que protegemos os direitos dos desenvolvedores originais.

4 Resultados

Nesta seção, discutimos os resultados obtidos ao longo do desenvolvimento do projeto, com foco no gerenciamento de tarefas, testes, implantação e demonstração da aplicação proposta. Esses aspectos foram fundamentais para garantir o progresso contínuo, a qualidade do sistema e a validação de suas funcionalidades.

4.1 Gerenciamento de Tarefas

O gerenciamento de tarefas foi um desafio ao longo do desenvolvimento do projeto. Embora inicialmente tenha sido tentado o uso de ferramentas como Trello e GitHub Projects para organizar as atividades em um formato Kanban, a implementação dessas abordagens não foi plenamente realizada.

4.1.1 Ferramentas Utilizadas

Apesar dos esforços iniciais para utilizar ferramentas de gerenciamento de tarefas, como Trello e GitHub Projects, para organizar e rastrear o progresso, não foi possível manter um fluxo contínuo e estruturado de trabalho com essas ferramentas. Embora essas plataformas tenham sido configuradas, a prática cotidiana acabou se distanciando dos métodos formais de gerenciamento, como Kanban ou Scrum.

4.1.2 Metodologia de Gerenciamento

Embora houvesse a intenção de adotar uma metodologia ágil, com sprints e reuniões de planejamento, na prática, o desenvolvimento ocorreu de maneira mais flexível e adaptativa. O trabalho foi realizado conforme as necessidades surgiam, sem uma estrutura formal de gerenciamento. Isso permitiu um desenvolvimento mais espontâneo, embora tenha dificultado o monitoramento contínuo do progresso.

4.1.3 Resultados Obtidos

Durante o desenvolvimento do projeto, não foi adotada uma metodologia ágil formal, como Scrum ou Kanban. Essa ausência de estrutura metodológica específica proporcionou certa

flexibilidade no ajuste de escopo e prioridades, mas também evidenciou desafios significativos em termos de organização e acompanhamento do progresso.

- **Adaptação e Flexibilidade:** A condução sem uma metodologia estruturada permitiu decisões rápidas e mudanças de direção conforme necessário. No entanto, isso ocorreu de forma não sistematizada, o que pode ser menos eficiente do que o uso de metodologias ágeis, que também prezam pela adaptabilidade, mas com mecanismos de controle e melhoria contínua.
- **Desafios na Coordenação:** A falta de uma abordagem ágil ou planejada dificultou a coordenação eficiente das tarefas, bem como a visibilidade sobre o progresso e os próximos passos.
- **Cumprimento dos Objetivos:** Apesar das limitações no gerenciamento, os principais objetivos do projeto foram alcançados. No entanto, o uso de uma metodologia ágil — ainda que adaptada à realidade do projeto — poderia ter facilitado o planejamento, a colaboração e o controle das entregas.

Embora o gerenciamento de tarefas não tenha seguido um modelo estruturado, a experiência serviu como aprendizado para a importância de métodos organizados e a necessidade de ferramentas e práticas que possam ser mantidas ao longo de um projeto de desenvolvimento de software.

4.2 Testes

Nesta seção, detalhamos as abordagens de teste utilizadas no projeto, abrangendo tanto testes manuais quanto automáticos.

4.2.1 Testes Manuais

Os testes manuais foram uma parte essencial do processo de validação das funcionalidades da aplicação. Eles foram realizados principalmente durante as fases iniciais de desenvolvimento para garantir que as funcionalidades principais estivessem operando conforme o esperado antes de automatizar os testes.

4.2.1.1 Abordagem

A abordagem de testes manuais envolveu a execução de casos de teste detalhados, onde cada funcionalidade da aplicação foi testada isoladamente e em combinação com outras

funcionalidades. Isso incluiu a navegação pelo aplicativo, a verificação da resposta a diferentes inputs do usuário, e a validação das integrações com APIs externas.

4.2.1.2 Casos de Teste

Os casos de teste manuais foram elaborados com base nos requisitos funcionais do sistema listados na Seção 3.4.1. Cada caso definiu a ação a ser realizada, os dados de entrada, o resultado esperado e o resultado real obtido durante a execução. Esses testes permitiram validar o comportamento da aplicação em cenários-chave. Alguns exemplos incluem:

Tabela 4 – Exemplo de testses manuais baseados nos requisitos funcionais

Caso de Teste	**Descrição**
Detecção Automática de Dispositivos	**Objetivo:** Verificar se o aplicativo detecta automaticamente outros dispositivos na mesma rede local. **Passos:** 1. Conectar dois dispositivos à mesma rede local; 2. Abrir o aplicativo de compartilhamento em ambos os dispositivos; 3. Verificar a detecção automática do outro dispositivo. **Resultado Esperado:** O outro dispositivo deve ser exibido na interface, indicando disponibilidade para compartilhamento.
Envio de Arquivos Grandes	**Objetivo:** Verificar se o aplicativo suporta o envio de arquivos grandes sem comprometer a performance. **Passos:** 1. Selecionar um arquivo grande (ex: vídeo de 2 GB) no dispositivo A; 2. Enviar para o dispositivo B via rede local; 3. Observar a performance durante a transferência. **Resultado Esperado:** A transferência deve ocorrer sem interrupções, com progresso em tempo real.
Favoritar Dispositivo	**Objetivo:** Verificar se o usuário pode adicionar um dispositivo à lista de favoritos. **Passos:** 1. Conectar dois dispositivos à mesma rede local; 2. Adicionar o dispositivo B à lista de favoritos no dispositivo A; 3. Desconectar e reconectar o dispositivo B; 4. Verificar se ele ainda aparece na lista de favoritos. **Resultado Esperado:** O dispositivo deve permanecer na lista de favoritos e estar disponível para futuras transferências.

4.2.2 Testes Automáticos

Os testes automáticos foram implementados para garantir a repetibilidade e a eficiência na validação contínua das funcionalidades da aplicação. Eles foram especialmente úteis durante as fases de desenvolvimento contínuo, permitindo a detecção precoce de regressões e erros.

4.2.2.1 Tipos de Testes Automáticos

Foram implementados diferentes tipos de testes automáticos, cada um focado em um aspecto específico do sistema, alguns exemplos:

Tabela 5 – Casos de Teste Complementares

Caso de Teste	Descrição
Favoritar Dispositivo	Valida a funcionalidade de favoritar dispositivos. O teste assegura que, ao adicionar um dispositivo aos favoritos, o estado interno da aplicação é atualizado corretamente e que a persistência dessa informação ocorre conforme esperado, mantendo os dados entre sessões.
Preparação de Upload	Verifica se o processo de conversão e preparação de dados para upload está funcionando corretamente. Garante que as informações dos dispositivos e dos arquivos sejam interpretadas adequadamente antes do envio, assegurando a integridade dos dados tratados.
Histórico de Recebimentos	Avalia as funcionalidades relacionadas ao histórico de arquivos recebidos. Inclui testes para adicionar novas entradas, remover itens específicos e limpar todo o histórico, garantindo o comportamento esperado em cada operação.

Quadro 4.1 – Código do teste de favoritar dispositivo

```
test('Should add a favorite device', () async {
  final service = ReduxNotifier.test(
    redux: FavoritesService(persistenceService),
  );

  expect(service.state, []);

  final device = _createDevice('1');

  await service.dispatchAsync(AddFavoriteAction(device));

  expect(service.state, [device]);
  verify(persistenceService.setFavorites([device]));
```

```
});
```

O Quadro 4.1 apresenta um teste automatizado implementado em *Dart*, cujo objetivo é verificar o correto funcionamento da funcionalidade de favoritar dispositivos na aplicação. No início do teste, é criada uma instância do serviço de favoritos em um ambiente de teste, garantindo que a lista de favoritos esteja inicialmente vazia.

Em seguida, um dispositivo fictício é criado e adicionado à lista por meio da ação `AddFavoriteAction`. O teste então verifica se o estado foi corretamente atualizado para conter o dispositivo e se o método responsável pela persistência dos dados foi chamado com os parâmetros esperados.

Esse procedimento assegura que a lógica de gerenciamento dos favoritos está funcionando conforme o esperado, tanto em termos de estado interno quanto na integração com o serviço de persistência.

Quadro 4.2 – Teste da conversão de JSON em *PrepareUploadResponseDto*

```
test('PrepareUploadResponseDto', () {
  final parsed = PrepareUploadResponseDto.fromJson({
    'sessionId': 'some session id',
    'files': {
      'some id': 'some url',
      'some id 2': 'some url 2',
    },
  });

  expect(parsed.sessionId, 'some session id');
  expect(parsed.files.length, 2);
  expect(parsed.files['some id'], 'some url');
  expect(parsed.files['some id 2'], 'some url 2');
});
```

O Quadro 4.2 apresenta um teste automatizado para a verificação da correta conversão de um objeto JSON em uma instância da classe `PrepareUploadResponseDto`.

Inicialmente, um objeto JSON é definido com um identificador de sessão e um mapa de arquivos, representando URLs associados a identificadores únicos. Em seguida, esse objeto é convertido para uma instância da classe por meio do método `fromJson`.

O teste então verifica se o atributo `sessionId` foi corretamente interpretado e se o mapa de arquivos contém os dados esperados. Esse procedimento assegura que a desserialização está funcionando corretamente, o que é essencial para a comunicação entre o servidor e o cliente na aplicação.

Quadro 4.3 – Código dos testes do histórico de recebimentos

```
test('Should add an entry', () async {
  final service = ReduxNotifier.test(
    redux: ReceiveHistoryService(persistenceService),
  );

  final entry = _createEntry('1');

  await service.dispatchAsync(AddHistoryEntryAction(
    entryId: entry.id,
    fileName: entry.fileName,
    fileType: entry.fileType,
    path: entry.path,
    savedToGallery: entry.savedToGallery,
    fileSize: entry.fileSize,
    senderAlias: entry.senderAlias,
    timestamp: entry.timestamp,
  ));

  expect(service.state, [entry]);
  verify(persistenceService.setReceiveHistory([entry]));
});
```

O Quadro 4.3 apresenta um conjunto de testes automatizados responsáveis por verificar o correto funcionamento da funcionalidade de histórico de recebimentos na aplicação.

Inicialmente, são testados os cenários em que uma nova entrada é adicionada ao histórico, respeitando a configuração do usuário para o salvamento dessa informação. Também é testado o limite máximo de 200 entradas, assegurando que a entrada mais antiga seja removida ao se atingir esse limite.

Além disso, os testes contemplam a remoção individual de entradas, a tentativa de remoção de uma entrada inexistente (sem causar alteração no estado), e a exclusão total do histórico. Em cada operação, o estado final da lista e a persistência dos dados são validados.

Esses testes garantem que o histórico de arquivos recebidos seja gerenciado corretamente, com consistência e respeito às regras de negócio, promovendo uma experiência confiável para o usuário.

4.2.3 Resultados dos Testes

A implementação de uma abordagem combinada de testes manuais e automáticos resultou em uma alta cobertura de testes, minimizando bugs e garantindo a robustez da aplicação. Os testes manuais permitiram validações rápidas durante o desenvolvimento inicial, enquanto os testes automáticos garantiram a estabilidade contínua do sistema à medida que novas funcionalidades foram introduzidas.

Os testes desempenharam um papel fundamental na garantia da qualidade do software desenvolvido. A combinação de diferentes abordagens de teste permitiu uma validação abrangente do sistema, assegurando que todas as funcionalidades principais funcionassem conforme o esperado, tanto isoladamente quanto em conjunto, proporcionando uma experiência de usuário final consistente e confiável.

4.3 Implantação

Nesta seção, descrevemos o processo de implantação e publicação do aplicativo nas principais plataformas de distribuição de aplicativos móveis, incluindo a Google Play Store e a Apple App Store. A implantação bem-sucedida é um passo crucial para garantir que o aplicativo esteja disponível para os usuários finais.

4.3.1 Google Play Store

A publicação do aplicativo na Google Play Store envolveu alguns desafios, incluindo a questão de permissões específicas. Inicialmente, o aplicativo utilizava a permissão `MANAGE_EXTERNAL_STORA` o que resultou na recusa da publicação. Após a remoção dessa permissão, a publicação foi aceita.

4.3.1.1 Pontos-Chave

- Realização de testes finais e otimização do desempenho antes do envio.
- Inserção de informações essenciais, como título, descrição, ícone e capturas de tela.
- Ajuste das permissões solicitadas pelo aplicativo para conformidade com as políticas do Google Play.
- Envio para análise e aprovação final.

4.3.2 Apple App Store

A publicação na Apple App Store também apresentou um desafio relacionado a permissões. Foi necessário solicitar um entitlement de multicast da Apple para o aplicativo. No entanto, após a solicitação, o aplicativo foi aprovado na primeira tentativa de submissão.

4.3.2.1 Pontos-Chave

- Verificação da funcionalidade em todos os dispositivos iOS suportados.
- Configuração do aplicativo com informações detalhadas e capturas de tela no App Store Connect.
- Solicitação e obtenção do entitlement de multicast necessário para o aplicativo.
- Utilização do Transporter para agilizar o upload seguro de builds para a Apple.
- Análise detalhada da Apple e aprovação na primeira submissão.

A implantação do aplicativo em ambas as plataformas, Google Play Store e Apple App Store, exigiu não apenas o cumprimento das diretrizes gerais, mas também a resolução de problemas específicos relacionados a permissões e entitlements. Seguir as diretrizes e políticas de cada plataforma, juntamente com a adaptação necessária, garantiu a aprovação do aplicativo e sua disponibilidade para os usuários finais. Com a publicação bem-sucedida, o aplicativo se torna acessível globalmente, permitindo que os usuários aproveitem as funcionalidades desenvolvidas e proporcionando retorno sobre o investimento de desenvolvimento.

4.4 Demonstração

Nesta seção, apresentamos uma demonstração prática do aplicativo desenvolvido, destacando suas principais funcionalidades e a experiência do usuário. A demonstração foi planejada para cobrir os cenários mais comuns de uso, fornecendo uma visão abrangente do funcionamento do sistema.

4.4.1 Visão Geral da Demonstração

A demonstração do aplicativo foi estruturada em etapas para mostrar como um usuário interage com o sistema desde o início até a conclusão de uma tarefa. As principais funcionalidades demonstradas incluem o processo de envio e recebimento de arquivos, navegação pelas funcionalidades principais, e a execução de ações críticas, como a seleção de dispositivos e a

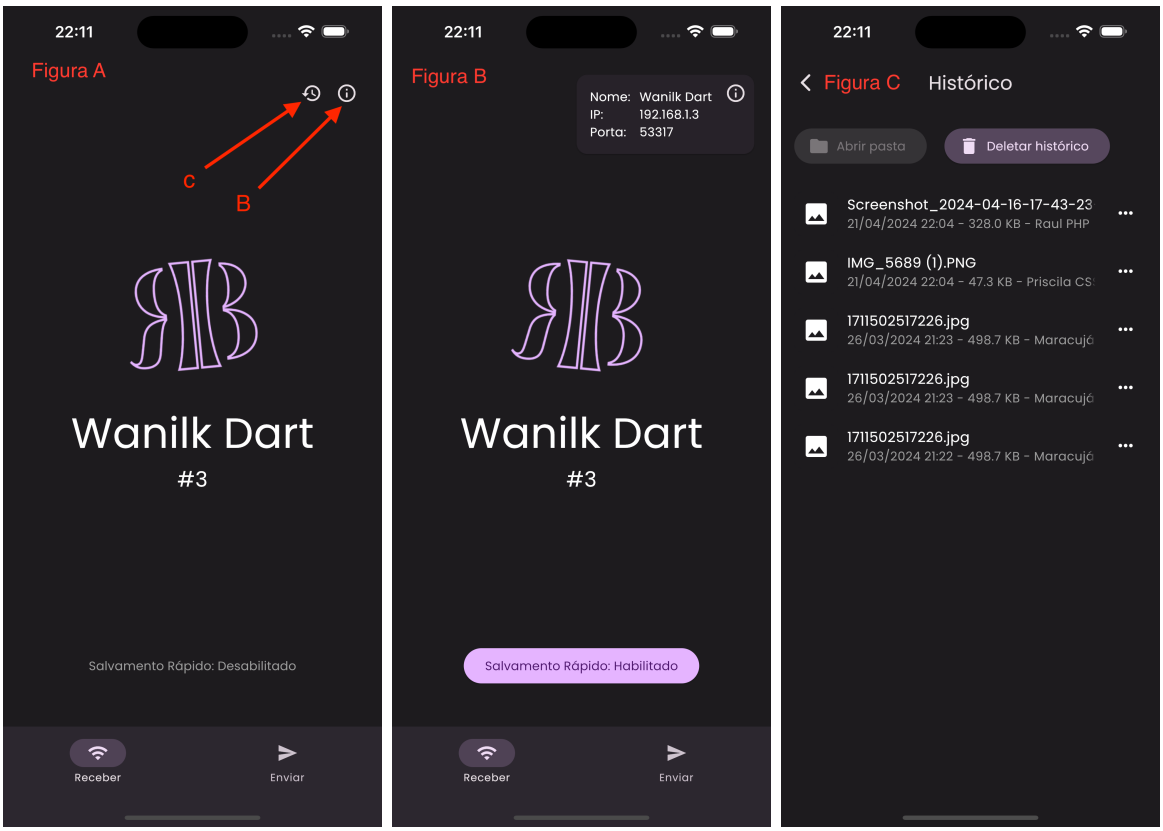
gestão de favoritos.

4.4.2 Funcionalidades Principais

A seguir, descrevemos as principais funcionalidades do aplicativo, tal como apresentadas na demonstração:

4.4.2.1 Receber

Figura 8 – Tela de receber, informações importantes, salvamento rápido e a tela de histórico



Fonte: Autoria própria (2024).

A tela de apresentação, denominada “Receber”, foi projetada para fornecer ao usuário informações claras sobre o dispositivo, incluindo seu nome identificador, que foi exibido nos outros dispositivos que receberam ou enviaram arquivos. Essa apresentação ajudou a identificar facilmente os dispositivos na rede e facilitou o processo de conexão e transferência de arquivos entre eles. Com uma abordagem centrada no usuário e focada na simplicidade e eficiência, a aplicação foi desenvolvida para atender às necessidades dos usuários de forma

rápida e intuitiva. Dessa forma, os usuários experimentaram uma interação mais fluida ao identificar e conectar-se a dispositivos na rede local, simplificando o processo de transferência de arquivos entre eles.

4.4.3 Identificação

A identificação foi projetada para funcionar de maneira simples e divertida, combinando um "Nome" e um "Sobrenome". O "Nome" foi selecionado aleatoriamente a partir de uma lista contendo diversos nomes de pessoas relevantes para o curso, incluindo alunos e amigos envolvidos no projeto. Já o "Sobrenome" foi escolhido a partir de uma lista que incluía nomes de linguagens de programação e frameworks. Por exemplo, foram geradas combinações como "Wanilk Dart" ou "Raul PHP", criando identidades únicas e interessantes para cada usuário no processo de transferência de arquivos.

4.4.3.1 Identificação adicional

Na tela de recebimento, além das informações padrão como o nome identificador do dispositivo e a situação da transferência, foi incluída uma identificação adicional representada pelo símbolo Hashtag. Esse símbolo Hashtag correspondia à última camada do endereço IP local do usuário na rede, proporcionando uma identificação única e específica para cada dispositivo na interface de recebimento de arquivos. Ao exibir essa informação, o usuário teve uma referência adicional para identificar e distinguir os dispositivos conectados à rede local, facilitando ainda mais o processo de recebimento e gerenciamento de arquivos entre eles. Essa identificação adicional foi apresentada de forma clara e intuitiva na interface da aplicação, contribuindo para uma experiência de usuário mais completa e informativa.

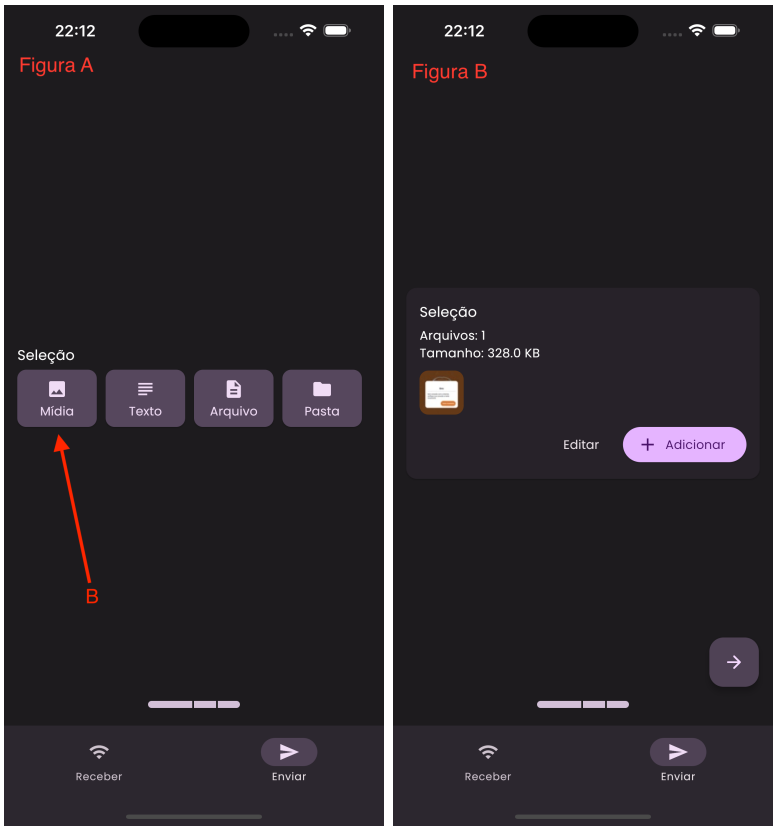
4.4.4 Informações importantes

A inclusão do alerta na tela inicial, apresentando informações importantes como o IP do dispositivo na rede local, seu aliás e a porta utilizada para a transferência de dados, foi projetada para fornecer uma experiência mais informativa e transparente para o usuário. Utilizando o pop-up menu do MaterialUI, mantido pela Google, foi garantida uma apresentação visual clara e integrada ao design da aplicação.

4.4.4.1 Enviar

Na tela de envio, foi implementada uma divisão em três partes utilizando o PageView, visando organizar o processo de envio de arquivos de forma clara e intuitiva para o usuário. A primeira parte consistiu na seleção dos arquivos a serem enviados, onde o usuário pôde escolher facilmente os documentos, imagens ou outros tipos de arquivos que desejava compartilhar. Em seguida, a segunda parte abordou os modos de envio disponíveis. Por fim, a terceira parte da tela de envio foi dedicada à identificação e seleção de dispositivos próximos, permitindo ao usuário visualizar e escolher os dispositivos disponíveis na rede local para enviar os arquivos selecionados. Essa divisão em partes facilitou a navegação e o entendimento do processo de envio de arquivos, garantindo uma experiência de usuário mais fluida e eficiente.

Figura 9 – Figura A: Tela de seleção de arquivos, onde o usuário escolhe o tipo de conteúdo a ser enviado. Figura B: Tela com os arquivos já selecionados, permitindo revisão antes do envio.



Fonte: Autoria própria (2024).

4.4.5 Seleção

Na interface de seleção de arquivos (Figura 9.A), o usuário pôde escolher entre diferentes tipos de conteúdo a serem enviados. No Android, as opções incluíam arquivos individuais (“file”), pastas inteiras (“folder”), texto (“text”) e conteúdo da área de transferência (“clipboard”), oferecendo flexibilidade e conveniência na escolha dos dados. No iOS, as opções foram adaptadas às características da plataforma, com foco em mídia (“media”), texto, arquivos e pastas. Essa diferenciação permitiu uma experiência de uso otimizada para cada sistema operacional.

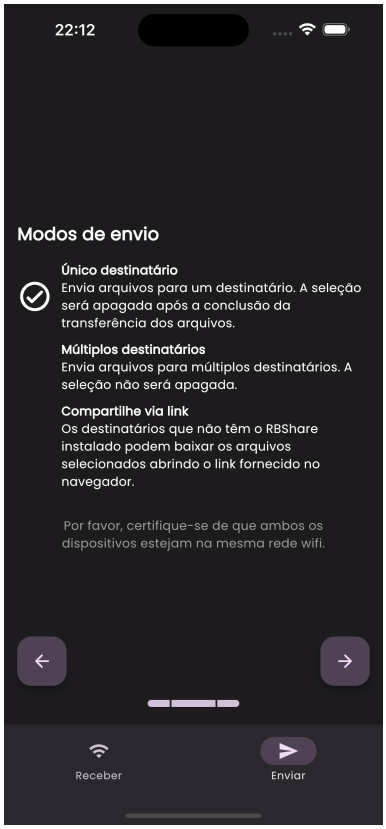
Após a seleção, os itens escolhidos eram exibidos de forma organizada na tela seguinte (Figura 9.B), permitindo ao usuário revisar o conteúdo antes do envio. Essa etapa conferiu maior controle e clareza ao processo de compartilhamento, contribuindo para uma experiência mais intuitiva e eficiente.

Ao adaptar a interface às particularidades de cada sistema e fornecer feedback visual claro, a aplicação garantiu uma experiência de seleção de arquivos consistente, independente da plataforma utilizada.

4.4.6 Modos de envio

Nos modos de envio, o usuário tem três opções disponíveis para escolher o método de compartilhamento mais adequado às suas necessidades. Essas opções são cuidadosamente desenvolvidas para oferecer flexibilidade e conveniência, permitindo que o usuário selecione o modo de envio que melhor se ajusta ao contexto e ao tipo de conteúdo a ser compartilhado. Essa variedade de opções garante que o processo de transferência seja adaptável às diferentes situações de uso, proporcionando uma experiência de usuário personalizada e eficiente.

Figura 10 – Ilustração dos três modos de envio disponíveis no aplicativo: único destinatário, múltiplos destinatários e compartilhamento via link.



Fonte: Autoria própria (2024).

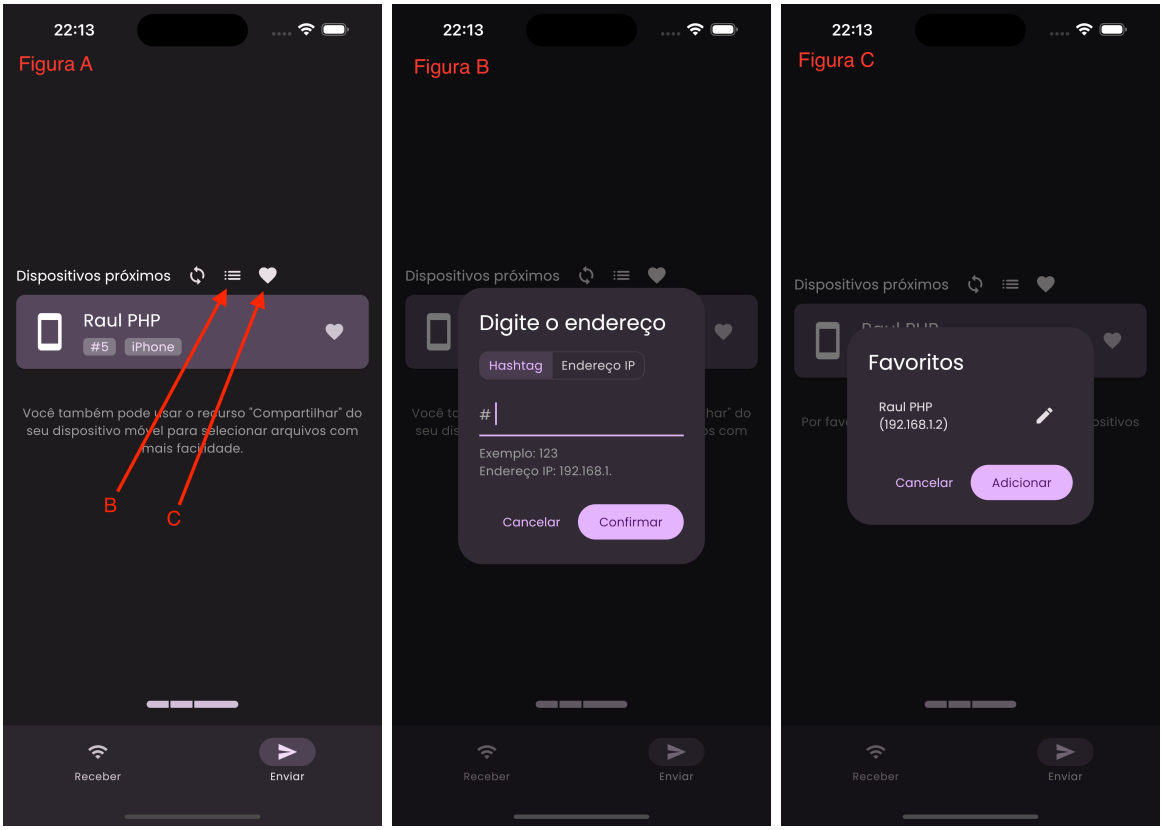
Conforme ilustrado na Figura 10, a aplicação oferece três modos distintos de envio, permitindo ao usuário escolher a forma mais adequada de compartilhamento de acordo com sua necessidade:

- **Único destinatário:** Permite enviar arquivos para apenas um destinatário por vez. Após a conclusão da transferência, a seleção de arquivos é apagada automaticamente, oferecendo uma abordagem simples e direta para compartilhamento individual.
- **Múltiplos destinatários:** Possibilita o envio simultâneo para vários dispositivos. Nesse modo, a seleção de arquivos é mantida após cada envio, facilitando o compartilhamento do mesmo conteúdo com diversas pessoas de forma eficiente.
- **Compartilhar via link:** Destinado a destinatários que não possuem o aplicativo *RB Share* instalado. O conteúdo é acessado por meio de um link gerado pela aplicação, que pode ser aberto em qualquer navegador, ampliando a acessibilidade do compartilhamento.

Essas opções proporcionam flexibilidade ao usuário, adaptando-se a diferentes cenários de uso e preferências, e contribuindo para uma experiência de compartilhamento mais personalizada e eficiente.

4.4.7 Dispositivos próximos

Figura 11 – Interface de dispositivos próximos: (A) lista de dispositivos disponíveis na rede local, (B) inserção manual de endereço IP, e (C) exibição da lista de dispositivos favoritos.



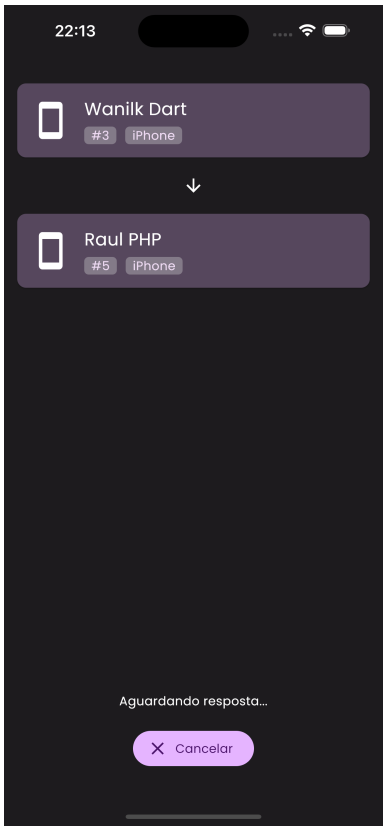
Fonte: Autoria própria (2024).

A Figura 11 apresenta a interface de dispositivos próximos, dividida em três partes. Na Figura 11.A, o usuário visualiza os dispositivos disponíveis na mesma rede local. Acima da lista, há um botão de recarregar que permite atualizar as informações dos dispositivos exibidos, garantindo que os dados estejam sempre atualizados. Na Figura 11.B, é mostrado o modal de inserção manual de endereço IP. Essa funcionalidade é útil para conectar-se diretamente a um dispositivo cujo IP já é conhecido, mesmo que ele não apareça automaticamente na lista. Por fim, a Figura 11.C exibe a lista de favoritos. Essa tela mostra os dispositivos salvos

previamente pelo usuário, permitindo acesso rápido aos contatos frequentes, sem depender da detecção automática na rede.

Esses recursos e funcionalidades são projetados para oferecer ao usuário uma experiência intuitiva e eficiente ao conectar-se com dispositivos na rede local, garantindo uma transferência de arquivos rápida e conveniente entre os dispositivos próximos.

Figura 12 – Tela de solicitação de envio exibida no dispositivo destinatário, com informações detalhadas do remetente e opções de aceitar ou rejeitar a transferência.



Fonte: Autoria própria (2024).

Conforme ilustrado na Figura 12, ao selecionar um dispositivo na lista de dispositivos próximos, uma solicitação de envio é exibida no dispositivo destinatário. Essa tela fornece ao usuário informações relevantes para decidir se deseja aceitar a transferência de arquivos.

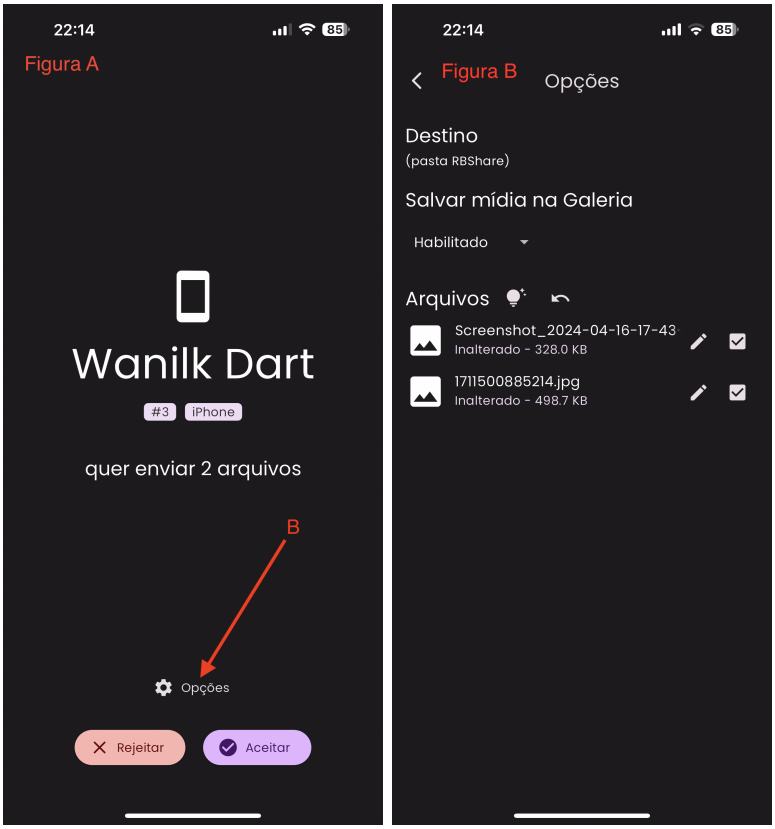
Entre os dados apresentados na solicitação de envio, estão: um ícone representando o tipo de dispositivo remetente, o nome identificador do dispositivo, informações sobre o modelo do celular e, em alguns casos, opções adicionais de configuração, como a escolha da pasta de destino ou ajustes de permissões específicas.

A interface também oferece dois botões de ação: “Rejeitar” e “Aceitar”. Ao clicar em

"Aceitar", a transferência dos arquivos é iniciada automaticamente. Caso o usuário opte por "Rejeitar", a solicitação é imediatamente cancelada. Esse mecanismo oferece maior controle e segurança, permitindo ao usuário decidir o que será recebido e de quem, aumentando a confiabilidade do processo de compartilhamento.

Conforme ilustrado na Figura 13, a imagem à esquerda apresenta a solicitação de envio recebida pelo destinatário, enquanto a imagem à direita exibe as opções de configuração disponíveis antes de aceitar a transferência. Essa abordagem proporciona uma experiência personalizada e segura, permitindo que o usuário revise e ajuste as configurações conforme necessário antes de confirmar o recebimento dos arquivos.

Figura 13 – Solicitação de envio recebida pelo destinatário (Figura A) e opções de configuração antes da transferência (Figura B).



Fonte: Autoria própria (2024).

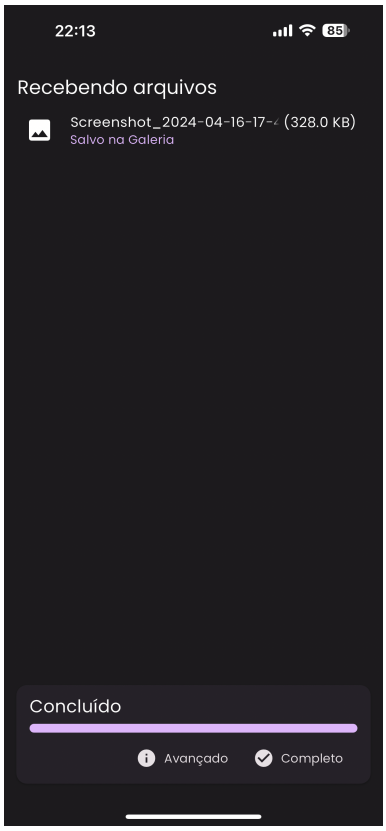
Após o destinatário aceitar a solicitação de transferência de arquivos, é exibida uma tela dedicada chamada "Recebendo Arquivos". Nessa tela, o usuário tem uma visão clara e detalhada de todos os arquivos que estão sendo recebidos, juntamente com o progresso individual de cada transferência. A lista de arquivos recebidos fornece informações como

o nome do arquivo e o status atual da transferência, permitindo que o usuário monitore facilmente o progresso e identifique qualquer problema que possa surgir durante o processo.

Além disso, uma barra de progresso total é exibida na parte inferior da tela, oferecendo uma visão geral do progresso de todos os arquivos. Essa barra de progresso fornece uma indicação visual clara do tempo estimado restante e do progresso geral da transferência. Para oferecer mais controle ao usuário, a tela também inclui uma opção "Avançado", que, quando clicada, exibe informações adicionais, como a quantidade de arquivos e o tamanho total dos arquivos que estão sendo recebidos. Isso permite que o usuário tenha uma compreensão mais detalhada do volume de dados sendo transferidos e do status geral da operação. Um botão "Concluir" está disponível na parte inferior da tela, permitindo que o usuário encerre o processo de recebimento de arquivos e retorne à tela principal ou de destino. Essa funcionalidade permite que o usuário finalize o recebimento de arquivos a qualquer momento, garantindo flexibilidade e controle durante todo o processo. Em conjunto, os elementos da interface exibidos durante o processo de recebimento oferecem ao usuário uma experiência intuitiva e eficiente. A tela permite acompanhar, em tempo real, o progresso de cada transferência, exibindo detalhes como o nome do arquivo, o remetente e o tempo estimado restante. Além disso, o usuário pode cancelar transferências individualmente, caso necessário, o que proporciona maior controle sobre o conteúdo recebido.

Conforme ilustrado na Figura 14, essa abordagem visual facilita o monitoramento e o gerenciamento de múltiplos arquivos sendo recebidos simultaneamente, tornando o processo mais transparente e seguro.

Figura 14 – Tela de recebimento de arquivos, com informações em tempo real sobre as transferências em andamento.



Fonte: Autoria própria (2024).

4.4.8 Compartilhar via link

Quando o usuário seleciona a opção “Compartilhar via Link” entre os modos de envio, a aplicação o direciona para uma tela específica, representada na Figura 15, que foi projetada para facilitar o compartilhamento de arquivos por meio de um link acessível via navegador web.

Nesta tela, o usuário encontra diversas funcionalidades. Uma delas é a possibilidade de abrir o link gerado diretamente no navegador padrão do dispositivo, permitindo visualizar e compartilhar o endereço com facilidade. O link é composto pelo endereço IP local do dispositivo remetente, seguido da porta utilizada para a transferência. Ao ser acessado, esse link direciona o navegador de outro dispositivo à interface web de um servidor local temporário, criado pelo aplicativo, permitindo o download direto dos arquivos compartilhados. Essa abordagem elimina a necessidade de serviços de terceiros ou armazenamento em nuvem, garantindo uma conexão mais rápida, privada e eficiente dentro da mesma rede local.

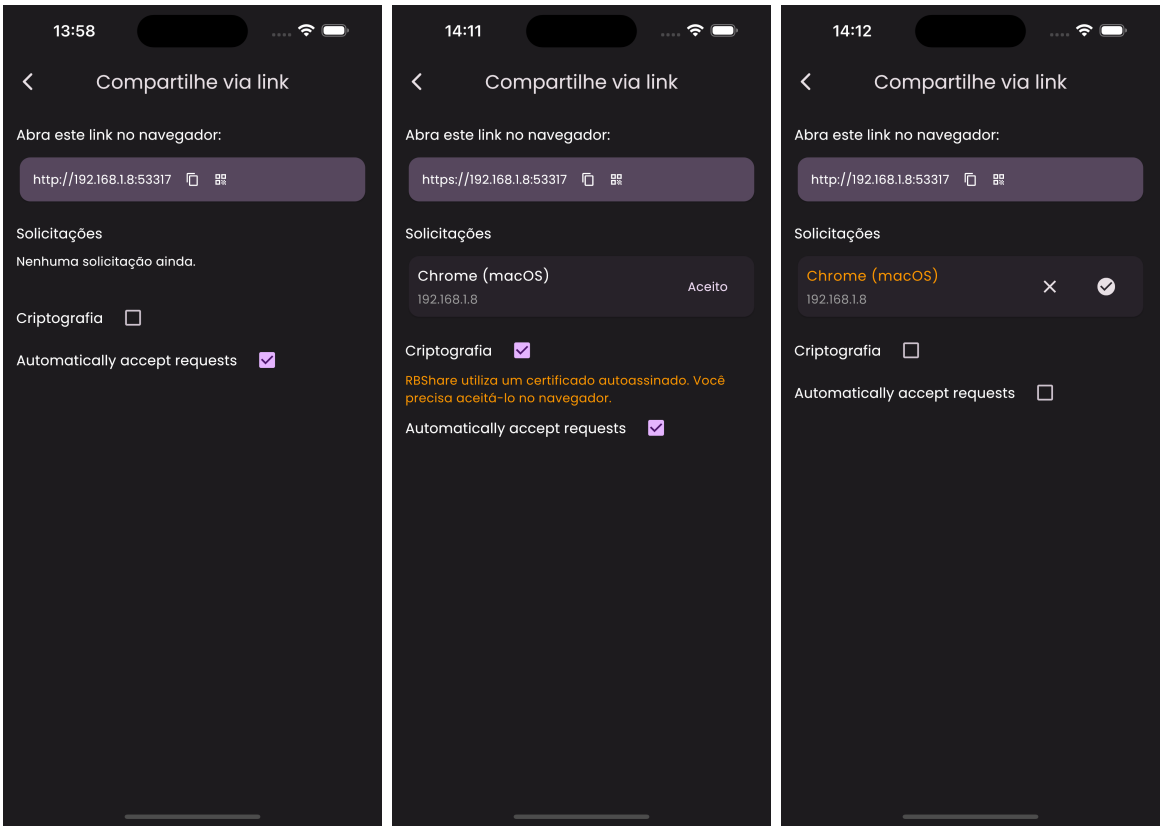
Outro recurso disponível é o botão de geração de QR Code. Ao ser clicado, um pop-up

exibe um código QR correspondente ao link de compartilhamento, o que facilita o acesso rápido ao conteúdo por meio de dispositivos móveis com câmera.

Abaixo do link e do QR Code, é exibida uma seção dedicada às solicitações de acesso pendentes. Esta funcionalidade é útil especialmente quando a aceitação automática de conexões não está ativada. O usuário pode visualizar essas tentativas e aprová-las manualmente, oferecendo maior controle sobre quem pode acessar os arquivos compartilhados.

Além disso, há duas opções adicionais que aprimoram a segurança e o gerenciamento do compartilhamento: um checkbox que permite ativar criptografia SSL (Secure Sockets Layer) para o link, protegendo os dados transmitidos; e outro que permite ativar a aceitação automática de novas conexões. Esses recursos conferem ao usuário um nível adicional de segurança e praticidade, garantindo que a experiência de compartilhamento seja ao mesmo tempo segura e fluida.

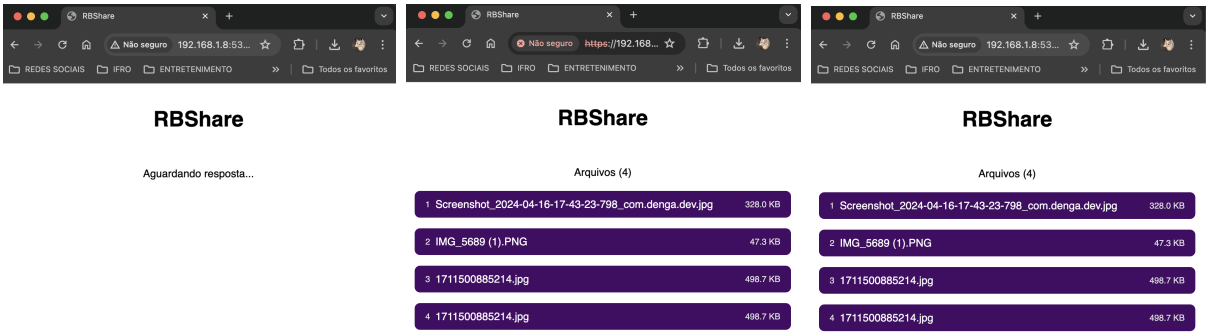
Figura 15 – Tela de compartilhamento via link. Figura A: link gerado com opções de aceitar solicitações automaticamente e exibir QR Code. Figura B: exibição de texto ao ativar a criptografia SSL. Figura C: link acessível via navegador e exibição de solicitações pendentes de acesso.



Fonte: Autoria própria (2024).

4.4.9 Compartilhar via link versão Web

Figura 16 – Recebimento de arquivos via navegador web. Figura A: solicitação de acesso aguardando aprovação. Figura B: acesso autorizado com criptografia SSL ativada. Figura C: listagem de arquivos disponíveis para download após aprovação.



Fonte: Autoria própria (2024).

Na versão web do *RB Share*, o título da aplicação é exibido como forma de identificação do projeto. Conforme ilustrado na Figura 16A, se a opção de aceitar solicitações automaticamente **não estiver ativada**, o destinatário verá uma tela solicitando permissão ao remetente. Somente após a aceitação, o acesso é liberado. A Figura 16B mostra a interface quando o acesso é autorizado com **criptografia SSL** ativada, indicando uma conexão segura. Por fim, a Figura 16C apresenta a **listagem dos arquivos disponíveis para download**, acessível assim que a solicitação é aceita. Essa abordagem garante segurança e controle durante o compartilhamento, mesmo ao utilizar um navegador.

5 Considerações Finais

Neste projeto, enfrentamos diversos desafios e adquirimos valiosos aprendizados que contribuíram significativamente para o desenvolvimento do sistema. Além disso, identificamos áreas para melhorias e expansão do trabalho em futuras iterações.

5.1 Desafios Enfrentados

Durante o desenvolvimento do aplicativo, diversos desafios técnicos e organizacionais foram enfrentados, exigindo soluções criativas, testes constantes e adaptabilidade ao longo do processo. A seguir, destacam-se os principais obstáculos superados:

- **Gerenciamento de Estado no Flutter:** Como o aplicativo foi desenvolvido utilizando Flutter, o gerenciamento de estado foi um aspecto crítico, especialmente em um projeto com múltiplas telas, atualizações em tempo real e interações simultâneas. A escolha de uma abordagem eficiente para lidar com o estado global do aplicativo — que impactava diretamente a navegação, as permissões e o status das transferências — foi fundamental para manter o código coeso e manutenível.
- **Descoberta e Comunicação entre Dispositivos:** Um dos maiores desafios técnicos foi a implementação da comunicação direta entre os dispositivos na rede local, sem utilizar servidores intermediários. A etapa de descoberta automática dos dispositivos na mesma rede enfrentou dificuldades devido a variações nas configurações de rede, firewalls e permissões do sistema. Foram realizados testes com servidores simulados e abordagens alternativas de broadcast e multicast até se alcançar uma solução confiável. No caso do iOS, a comunicação multicast exigiu uma configuração específica (entitlement) que precisou ser solicitada à Apple e validada durante o processo de publicação.
- **Publicação e Conformidade com Políticas de Loja:** O processo de submissão às lojas de aplicativos (Google Play Store e Apple App Store) envolveu a adequação a políticas rigorosas. Um dos obstáculos mais relevantes foi o uso da permissão `MANAGE_EXTERNAL_STORAGE` no Android, que gerou a recusa da primeira submissão à Google Play Store. Isso exigiu a reestruturação da lógica de acesso a arquivos e a adoção de alternativas mais seguras e restritas, em conformidade com as novas diretrizes de privacidade da plataforma.
- **Integração de Funcionalidades Multiplataforma:** Garantir que as funcionalidades

funcionassem uniformemente em dispositivos Android e iOS foi um desafio constante. A diferença na forma como cada sistema operacional trata permissões, rede e arquivos tornou necessária a criação de soluções específicas para cada plataforma em algumas partes do código, preservando, sempre que possível, a base compartilhada do Flutter.

- **Teste e Garantia de Qualidade:** Testar o aplicativo em diferentes marcas de dispositivos, versões do sistema operacional e cenários de rede exigiu uma abordagem diversificada de validação. Foram combinados testes manuais com testes automatizados para garantir estabilidade, consistência na interface e performance. A detecção e correção de problemas relacionados à compatibilidade de hardware e software foram etapas fundamentais antes da publicação final.
- **Otimização de Desempenho em Rede Local:** Como o *RB Share* é baseado em comunicação ponto-a-ponto via rede local, foi necessário garantir uma performance satisfatória mesmo em ambientes com redes instáveis ou congestionadas. Ajustes finos foram realizados para otimizar a velocidade de envio e recepção de arquivos grandes, reduzindo o consumo de memória e evitando travamentos ou perdas de conexão.
- **Adaptação a Mudanças de Requisitos:** Durante o ciclo de desenvolvimento, alguns requisitos mudaram com base em testes, feedbacks de usuários e orientações das lojas de aplicativos. Houve a necessidade de ajustar o escopo de funcionalidades, rever permissões, adaptar fluxos de navegação e reformular trechos da arquitetura original para garantir a entrega de uma solução funcional, segura e alinhada às exigências externas.
- **Conciliar Projeto com Atividades Profissionais:** O desenvolvimento do aplicativo ocorreu em paralelo às atividades profissionais do desenvolvedor na área de software. Conciliar os prazos e demandas do trabalho com o progresso do projeto pessoal exigiu uma gestão eficiente do tempo e disciplina. Essa experiência, embora desafiadora, proporcionou aprendizados valiosos sobre organização, priorização de tarefas e equilíbrio entre diferentes responsabilidades técnicas e pessoais.

Esses desafios demonstram a complexidade de desenvolver uma aplicação que alia privacidade, desempenho, usabilidade e compatibilidade multiplataforma em um único projeto. A superação desses obstáculos foi essencial para alcançar os objetivos do *RB Share* e oferecer uma solução robusta, confiável e alinhada às demandas reais dos usuários.

5.2 Aprendizados Adquiridos

Ao longo do projeto, foram adquiridos aprendizados valiosos que contribuíram para o crescimento profissional e o aperfeiçoamento das práticas de desenvolvimento:

- **Aprofundamento em Tecnologias Multiplataforma:** O uso de Flutter para desenvolver uma aplicação multiplataforma proporcionou um aprendizado profundo sobre essa tecnologia, incluindo suas vantagens e desafios. A experiência adquirida no gerenciamento de estado e na criação de interfaces de usuário responsivas foi particularmente enriquecedora.
- **Adaptação a Mudanças Rápidas:** Durante o desenvolvimento, surgiram necessidades de mudanças rápidas, tanto em requisitos quanto em abordagens técnicas. A capacidade de adaptação e a implementação ágil dessas mudanças foram habilidades desenvolvidas e aprimoradas ao longo do projeto.
- **Valorização da Segurança e Privacidade dos Dados:** Durante o desenvolvimento, ficou evidente a importância de incorporar práticas robustas de segurança e privacidade. A gestão adequada de permissões e a necessidade de adaptar funcionalidades para atender aos padrões de privacidade das plataformas reforçaram o compromisso com a proteção dos dados dos usuários.
- **Otimização de Desempenho para Melhor Experiência do Usuário:** Aprender a identificar e implementar otimizações de desempenho, especialmente em um aplicativo que lida com arquivos grandes e múltiplas conexões, foi crucial. Esse aprendizado não só melhorou a experiência do usuário, mas também proporcionou uma compreensão mais profunda sobre como equilibrar funcionalidade e eficiência.
- **Importância de Testes Abrangentes:** A combinação de testes manuais e automáticos foi essencial para garantir a qualidade do aplicativo. A experiência mostrou a importância de uma abordagem abrangente de testes para identificar problemas antes que eles afetem os usuários finais, melhorando a confiabilidade e a robustez do aplicativo.
- **Entendimento Profundo de Conformidade com Políticas de Lojas de Aplicativos:** A experiência adquirida ao lidar com as políticas rigorosas da Google Play Store e da Apple App Store foi significativa. Compreender as exigências de cada plataforma, desde permissões até entitlements específicos, como o multicast na Apple, destacou a importância de conformidade e de ajustes precisos para garantir a publicação bem-sucedida.

5.3 Trabalhos Futuros

Embora o aplicativo desenvolvido atenda aos requisitos iniciais, há várias oportunidades para expandir e aprimorar o sistema em trabalhos futuros:

- **Otimização de Performance:** Continuar a otimizar o desempenho do aplicativo,

especialmente em dispositivos de menor capacidade, é uma prioridade. Isso inclui a melhoria do tempo de carregamento e a redução do consumo de recursos.

- **Melhoria da Experiência do Usuário:** Trabalhos futuros incluirão a otimização da interface do usuário para torná-la mais intuitiva e acessível. Isso pode incluir a implementação de novos temas, suporte a diferentes idiomas e ajustes na navegabilidade do aplicativo para facilitar o uso por uma audiência ainda mais ampla.
- **Suporte a Transferências entre Redes Distintas:** Trabalhos futuros incluirão a adaptação do aplicativo para permitir a transferência de arquivos entre dispositivos que, embora geograficamente separados e em redes distintas, possam se conectar por meio de tecnologias peer-to-peer (P2P). Esse modelo de comunicação seria inspirado no funcionamento de sistemas como o BitTorrent e jogos online, como o GTA 5 Online, oferecendo uma solução eficiente e robusta para o compartilhamento de arquivos em diferentes condições de rede.
- **Análise e Monitoramento de Uso:** A introdução de ferramentas de análise e monitoramento pode fornecer insights valiosos sobre o comportamento dos usuários e o desempenho do aplicativo. Esse feedback permitirá ajustes contínuos e melhorias baseadas em dados concretos, otimizando a experiência do usuário e a eficiência do sistema.
- **Integração com Serviços em Nuvem:** Uma possível expansão do aplicativo inclui a integração com serviços de armazenamento em nuvem, permitindo que os usuários façam backup de arquivos diretamente do aplicativo ou compartilhem arquivos armazenados na nuvem com outros dispositivos na rede local.
- **Divisão de Arquivos Grandes:** Implementar a funcionalidade de divisão de arquivos grandes em múltiplos blocos para permitir a retomada da transferência em caso de falhas. Isso aumentaria a confiabilidade do sistema e evitaria a necessidade de reiniciar o envio do zero após interrupções.
- **Melhorias na Interface Web (Modo Link):** Quando utilizado via link em um navegador, o aplicativo poderá contar com um site mais moderno, bonito e intuitivo, oferecendo uma experiência visual mais agradável e funcionalidades acessíveis mesmo fora do aplicativo nativo.

5.4 Conclusão

O projeto foi concluído com sucesso, superando desafios técnicos e organizacionais que surgiram ao longo do caminho, como a conformidade com políticas de publicação e a otimização de desempenho. Essas barreiras foram enfrentadas com soluções criativas e

uma abordagem adaptável, resultando em um produto funcional e eficaz. Os aprendizados adquiridos e as habilidades desenvolvidas não apenas garantiram o sucesso deste projeto, mas também estabeleceram uma base sólida para futuros empreendimentos. A experiência acumulada servirá como guia para enfrentar novos desafios e explorar oportunidades de desenvolvimento e inovação. Com a conclusão deste projeto, abre-se caminho para futuras melhorias e expansões do aplicativo, assegurando sua relevância contínua e aprimorando a experiência do usuário.

Referências

Apple. *Submit your apps to the App Store*. 2024. Accessed: 2024-08-11. Disponível em: <https://developer.apple.com/ios/submit/>. Citado na página 20.

Apple. *Transporter*. 2024. Accessed: 2024-08-11. Disponível em: <https://apps.apple.com/br/app/transporter/id1450874784?mt=12>. Citado na página 28.

Apple Developer. *Xcode*. 2024. Accessed: 2024-08-11. Disponível em: <https://developer.apple.com/xcode/>. Citado na página 28.

ARCHITECTURE, C. M. for S. *C4 Model Diagrams in Figma*. 2022. <https://www.figma.com/community/file/1122907722147721168>. Accessed: 2024-08-25. Citado na página 39.

ATLASSIAN. *Gitflow Workflow*. <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>. Accessed: 2024-08-25. Citado 2 vezes nas páginas 8 e 30.

BERGAMASCHI, S. et al. Redes sociais na internet: uma investigação sobre práticas de compartilhamento de arquivos no facebook. *Revista Gestão Da Produção Operações E Sistemas*, v. 0, n. 4, p. 125, 2012. Disponível em: <https://doi.org/10.15675/gepros.v0i4.900>. Citado na página 18.

BROWN, S. *The C4 Model for Visualising Software Architecture*. 2020. Accessed: 2024-08-11. Disponível em: <https://gredos.usal.es/bitstream/handle/10366/143085/1396-pre.pdf?sequence=1>. Citado na página 22.

BROWN, S. *C4 model for visualising software architecture*. 2024. <https://c4model.com/>. Acesso em: 29 abr. 2025. Citado na página 22.

Creative Commons. *About CC Licenses*. 2025. Acesso em: 29 abr. 2025. Disponível em: <https://creativecommons.org/share-your-work/cclicenses/>. Citado na página 19.

DRIVEUPLOADER. *The Evolution of File Sharing: From Napster to Blockchain*. 2022. <https://driveuploader.com/blog/the-evolution-of-file-sharing-from-napster-to-blockchain/>. Acesso em: 29 de abril de 2025. Citado na página 13.

Flutter. *Beautiful native apps in record time*. 2024. Accessed: 2024-08-11. Disponível em: <https://flutter.dev/>. Citado na página 27.

Google. *Android Studio*. 2024. Accessed: 2024-08-11. Disponível em: <https://developer.android.com/studio?hl=pt-br>. Citado na página 28.

Google Play Support. *Prepare and roll out a release*. 2024. Accessed: 2024-08-11. Disponível em: <https://support.google.com/googleplay/android-developer/answer/9859152?hl=en>. Citado na página 20.

LEE MARIO GERLA, C.-C. C. S.-J. Mon-demand multicast routing protocol. *University of California*, 2007. Citado na página 19.

LESSIG, L. *Free Culture: How Big Media Uses Technology and the Law to Lock Down Culture and Control Creativity*. New York: Penguin Press, 2004. Citado na página 18.

MIZUKAMI, P. N. Função social da propriedade intelectual: compartilhamento de arquivos e direitos autorais na cf/88. *Pontifícia Universidade Católica de São Paulo*, 2007. Citado na página 18.

Recording Industry Association of America. *The Impact of Digital File Sharing on the Music Industry: An Empirical Analysis*. 2004. Acesso em: 29 abr. 2025. Disponível em: <https://www.riaa.com/reports/the-impact-of-digital-file-sharing-on-the-music-industry-an-empirical-analysis/>. Citado na página 18.

REZENDE, E. da S. Modelo estrutural para compartilhamento de arquivos peer-to-peer. *UNESP*, v. 0, n. 0, jun. 2009. Disponível em: <https://repositorio.unesp.br/handle/11449/98698>. Citado na página 13.

SAHASRABUDDHE, B. M. L. H. Multicast routing algorithms and protocols: A tutorial. *University of California*, 2000. Citado na página 19.

SILVA, M. D. d.; SILVA, F. A.; MORAES, R. M. de. *DAPES: A Data-centric Peer-to-peer file sharing framework for ad hoc networks*. 2020. <https://arxiv.org/abs/2006.01651>. Acesso em: 29 de abril de 2025. Citado na página 13.

Visual Studio Code. *Code editing. Redefined*. 2024. Accessed: 2024-08-11. Disponível em: <https://code.visualstudio.com/>. Citado na página 27.

WATANABE, H. et al. Blockchain for digital rights management. In: *Enabling the Internet of Things: From Integrated Circuits to Integrated Systems*. [S.l.]: Academic Press, 2016. p. 273–284. Citado na página 19.

Anexos

ANEXO A – Licença MIT