UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GEOVANE DE CAMPOS SOARES

REDE SOCIAL MOBILE FEDERADA PARA USO DE COMUNIDADES DE COSPLAYERS

PATO BRANCO

2025

GEOVANE DE CAMPOS SOARES

REDE SOCIAL MOBILE FEDERADA PARA USO DE COMUNIDADES DE COSPLAYERS

FEDERATED MOBILE SOCIAL NETWORK FOR COSPLAY COMMUNITIES

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Analise e Desenvolvimento de Sistemas do Curso Superior de Tecnologia em Analise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Robison Cris Brito

PATO BRANCO

2025

4.0 Internacional

Esta licença permite download e compartilhamento do trabalho desde que sejam atribuídos créditos ao(s) autor(es), sem a possibilidade de alterá-lo ou utilizá-lo para fins comerciais. Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

GEOVANE DE CAMPOS SOARES

REDE SOCIAL MOBILE FEDERADA PARA USO DE COMUNIDADES DE COSPLAYERS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Analise e Desenvolvimento de Sistemas do Curso Superior de Tecnologia em Analise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 16/junho/2025

Robison Cris Brito
Doutorado
Universidade Tecnológica Federal do Paraná - Campus Pato Branco

---

Bernardo Alves Villarinho Lima
Doutorado
Universidade Tecnológica Federal do Paraná - Campus Pato Branco

---

Rúbia Eliza de Oliveira Schultz Ascari
Doutorado
Universidade Tecnológica Federal do Paraná - Campus Pato Branco

PATO BRANCO
2025

Dedico este trabalho à minha família e amigos,
que me apoiaram em toda essa jornada.

AGRADECIMENTOS

Certamente estes parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase de minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre essas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

Agradeço ao meu orientador Prof. Dr. Robison Cris Brito, pela sabedoria com que me guiou nesta trajetória.

Aos meus colegas de sala.

À Secretaria do Curso, pela cooperação.

Gostaria de deixar registrado também, o meu reconhecimento à minha família, pois acredito que sem o apoio deles seria muito difícil vencer esse desafio.

RESUMO

A comunidade *Cosplay*, um subconjunto da cultura *Geek*, é conhecida por sua paixão em criar e interpretar personagens de jogos, séries e outras mídias. Dada a natureza específica dessa comunidade, existe uma lacuna no mercado para redes sociais dedicadas a esse nicho. Este trabalho apresenta uma plataforma móvel, desenvolvida utilizando o *framework* Kotlin Multi-Platform, para atender às necessidades específicas dos *cosplayers*. A plataforma aplica a linguagem de comunicação *AT-Protocol* para permitir a criação de uma rede social federada, promovendo a comunicação e o compartilhamento de experiências entre os usuários. O desenvolvimento foi realizado no Android Studio para macOS.

Palavras-chave: cosplay; android; kotlin multi-plataform; at-protocol.

ABSTRACT

The Cosplay community, a subset of Geek culture, is renowned for its passion in creating and portraying characters from games, series, and other media. Given the specific nature of this community, there is a market gap for dedicated social networks targeting this niche. This work presents a mobile platform developed using the Kotlin Multiplatform framework to address the specific needs of cosplayers. The platform implements the AT-Protocol communication language to enable a federated social network, facilitating communication and experience-sharing among users. Development was conducted in Android Studio for macOS.

Keywords: cosplay; android; kotlin multi-plataform; at-protocol.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	Interface de Programação de Aplicações, do inglês *Application Programming Interface*
BBS	Sistema de painel de boletins, do inglês *Bulletin Board System*
BGS	*Brasil Game Show*
CCXP	*ComicCon Experience*
HTC	*High-Tech Computer Corporation*
IDE	Ambiente de Desenvolvimento integrado, do inglês *Integrated Development Environment*
IRC	Relê de Chats via internet, do inglês *Internet Relay Chat*
JSON	Notação de Objeto JavaScript, do inglês *JavaScript Object Notation*
JVM	Máquina Virtual Java, do inglês *Java Virtual Machine*
JWT	*JSON Web Token*
KMP	Kotlin Multi-Plataform
PDS	Servidor Privado de Dados, do inglês *Private Data Server*
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais
SDK	Kit de Desenvolvimento de Software, do inglês *Software Development Kit*
SGBD	Sistema Gerenciador de Banco de Dados
SO	Sistema Operacional
XML	Linguagem Extensa de Marcação, do inglês *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

1.1 Considerações iniciais

Um elemento comum no cotidiano das pessoas, as redes sociais já se tornaram uma grande parte do dia a dia de todos. Conforme a Internet se popularizou, nos últimos anos, a troca de conhecimento e experiências se torna cada vez mais presente para todos, podendo explorar outras culturas e costumes, assim como seus entretenimentos também.

Com a facilidade de acesso a essas informações, cada vez se torna mais popular no ocidente a busca por jogos, animês e outros conteúdos de entretenimento originários do Japão e de outros países asiáticos, formando uma comunidade ampla com pessoas que compartilham esses interesses. Da mesma forma, também se popularizou uma outra comunidade, que tem como gosto em comum a atividade de se caracterizarem de acordo com seus personagens favoritos, conhecida como *cosplay*.

Cosplay é uma atividade que teve o inicio da sua popularidade em meados dos anos 70, nos Estados Unidos, quando uma convenção promoveu entrada gratuita para pessoas fantasiadas de super-heróis. No Brasil, o *cosplay* se iniciou na década de 80 de maneira pouco difundida, porém se intensificou nos anos 90 e em 2006 obteve o título de destaque entre as comunidades de *cosplay* do mundo (Dantas, 2016). Não há informações precisas em relação ao número de membros atualmente, estima-se estar acima de 55 mil membros, considerados a partir da maior comunidade de *cosplay* no Facebook[1] (COSPLAY BRASIL, 2024).

Atualmente, a comunidade de *cosplayers* utiliza das redes sociais para divulgar suas experiências e seus trabalhos, porém, entre as 10 redes sociais mais utilizadas no Brasil (MLABS, 2024), não há nenhuma rede social específica para essa comunidade. Dentre as redes que operam no Brasil, as que mais se destacam são as redes que permitem monetizar seus trabalhos, tais como FanFever, Patreon e também OnlyFans (Cortes, 2023).

Já com relação a aplicativos móveis, há uma rede chamada Incosplay (INCOSPLAY GLOBAL INC, 2023), que tem como objetivo substituir várias plataformas em apenas um aplicativo, porém a popularidade não é tão alta, considerando que na PlayStore consta apenas 5000+ *downloads*.

Com o lançamento da rede social BlueSky[2], também foi inserido um novo conceito de rede social para os dias atuais. O uso de um protocolo *open source* de comunicação, o *AT-Protocol* (BLUESKY SOCIAL PBC, 2022), permite que seus usuários possam hospedar seus próprios dados, mantendo a conexão com o restante da rede, o que a caracteriza como uma rede social federada.

[1] Facebook é uma rede social online lançada em 2004. Permite que seus usuários se conectem com amigos e familiares, compartilhem fotos, vídeos, mensagens e atualizações de status. Pertence à empresa Meta, Inc.

[2] Disponível em https://www.linkedin.com/ .Último acesso em junho de 2025.

Redes sociais federadas são grupos online independentes que se conectam, como bairros digitais interligados. As plataformas descentralizadas colocam o controle nas mãos dos usuários, não em uma única empresa. O *AT-Protocol* é um padrão tecnológico que viabiliza a criação dessas redes, priorizando a portabilidade da sua identidade e dados entre diferentes aplicativos. Isto permite que cada usuário leve sua conta e conexões para onde quiser, garantindo mais liberdade e controle sobre a experiência online.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver uma rede social federada, voltada para o público *cosplayer*, para promover o compartilhamento de experiências.

1.2.2 Objetivos específicos

- Projetar uma aplicação multiplataforma, utilizando tecnologia Kotlin Multi-Plataform (KMP) para acesso geral da comunidade *cosplayer*[3];
- Implementar o *AT-Protocol* como forma de gerenciar postagens na rede.

1.3 Justificativa

Atualmente, a comunidade de *cosplay* não tem rede social específica no Brasil. Dessa forma, com a rede social a ser desenvolvida, os usuários poderão explorar postagens dos usuários conectados em outras redes sociais que estejam na *Atmosphere* [4], em um *feed* específico, mas ainda tendo acesso a sua *timeline* padrão.

A decisão de realizar o desenvolvimento da aplicação para dispositivos móveis foi baseada na intenção de atingir a maioria da comunidade, pois atualmente é mais comum os usuários realizarem acessos a sites/aplicativos diretamente nos seus *smartphones*, dessa forma a aplicação terá maiores chances de se popularizar na comunidade.

O desenvolvimento voltado para a plataforma Android também é o foco, pois no Brasil, 82.72% dos usuários utilizam o sistema operacional Android em seus dispositivos (STATCOUNTER GLOBAL STATS, 2024), seguido pelo iOS, com 17.06%. Porém o *framework* Kotlin Multi-

[3] Termo que designa um indivíduo praticante de *cosplay*, uma forma de arte performática em que a pessoa utiliza fantasias, acessórios e maquiagem para representar fielmente um personagem específico. Além da aparência visual, a prática frequentemente envolve a imitação dos maneirismos e do comportamento do personagem em eventos de cultura pop.

[4] Termo utilizado pelo protocolo para descrever o seu ecossistema. Ver (BLUESKY SOCIAL PBC, 2025a).

Platform permitirá que com apenas uma base de código, seja possível utilizar em ambos os sistemas operacionais mais comuns.

1.4 Estrutura do Trabalho

O presente trabalho está organizado em cinco capítulos, que detalham as etapas do desenvolvimento do projeto. Inicialmente, o Capítulo 1 apresenta a introdução, contextualizando o problema, a relevância da aplicação e seus objetivos. Em seguida, o Capítulo 2 estabelece a fundamentação técnica, abordando os conceitos e as tecnologias que serviram de base para a implementação. O Capítulo 3, por sua vez, descreve a arquitetura do sistema e a metodologia de desenvolvimento adotada. O Capítulo 4 é dedicado à apresentação da aplicação desenvolvida, demonstrando suas principais telas e funcionalidades. Finalmente, o Capítulo 5 encerra o trabalho com as conclusões sobre o projeto, as dificuldades encontradas e sugestões para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão apresentados os conceitos básicos para compreender o público-alvo deste trabalho, assim como os conceitos fundamentais envolvidos no desenvolvimento deste trabalho.

2.1 Comunidade Cosplay e Cultura *Geek*

2.1.1 Definição

Segundo o Priberam Informática, S.A. (2024), o termo *cosplay* tem sua etimologia originária na palavra japonesa *kosupure*, que seria a combinação das palavras *costume* e *play*, que se entende como a atividade de se caracterizar fisicamente como algum personagem e também encenar seus trejeitos e características.

2.1.2 Cultura *Geek* e a comunidade

Os primeiros registros da atividade de *cosplay* datam o ano de 1939, na 1ª Convenção Mundial de Ficção Científica (WorldCon), em Nova Iorque. No evento, os fãs do filme Daqui a Cem Anos (Wells, 1936), foram caracterizados como os personagens do filme, com trajes *"futurecostume"* (traduzido livremente para trajes futuristas).

No Brasil, há uma comunidade bastante ativa, com vários eventos que ocorrem anualmente, sendo os principais eventos a Brasil Game Show (BGS), Anime Friends, realizada em Junho e a ComicCon Experience (CCXP), realizada no início de Dezembro de 2024, todas na cidade de São Paulo.

2.2 Redes Sociais e Comunidades *Online*

2.2.1 Definição

Conforme apresentado por Rodrigues (2024), redes sociais referem-se a ambientes *online* (sites ou aplicativos), que permitem realizar o compartilhamento de informações entre pessoas e/ou instituições.

2.2.2 Origem e história

Com o conceito de conectar pessoas *online*, os primeiros registros de redes sociais surgiram nos anos 1990 (junto com a popularização da internet), com o Sistema de painel de

boletins, do inglês *Bulletin Board System* (BBS) e Relê de Chats via internet, do inglês *Internet Relay Chat* (IRC), que eram plataformas que permitiam a comunicação entre os usuários conectados.

Considerando as redes sociais "modernas", a SixDegrees.com (Weinreich, 1997), criada em 1997 e encerrada em 2001, com um pico de usuários em 3,5 milhões de membros, é creditada como a primeira rede social moderna devido ao fato de ter a aparência de redes sociais como conhecemos hoje, com perfis para cada usuário e a possibilidade de adicionar outros membros (Rodrigues, 2024).

Nas últimas décadas, várias redes sociais surgiram. Dentre as que não estão mais ativas ou não são mais populares, pode-se citar o Orkut[1] e MySpace[2] como exemplos. Já com relação às que surgiram e ainda estão ativas, têm-se como exemplo as do grupo Meta (Facebook[3], Instagram[4] e WhatsApp)[5], LinkedIn[6] (rede social voltada para o ambiente corporativo) e X[7] (inicialmente chamada de Twitter). Entre as mais recentes que popularizaram uma nova forma de comunicação, têm-se o TikTok[8], voltado para o compartilhamento de vídeos curtos.

2.2.3 Redes sociais e comunidades específicas

De modo geral, há várias redes sociais com focos específicos, para exercícios, por exemplo, pode-se elencar o Strava[9], no qual pode-se compartilhar seus exercícios e afins. Um exemplo para outras comunidades, tem-se a TripAdvisor[10], voltado para viagens.

Para a comunidade em questão deste projeto, não há muitas opções de redes sociais específicas. As mais comuns entre membros da comunidade são Facebook, Instagram e, dependendo do foco do mesmo, também há o DevianArt[11].

---

1 A rede social foi descontinuada em setembro de 2014.
2 Disponível em https://myspace.com/ .Último acesso em fevereiro de 2025
3 Disponível em https://www.facebook.com/ .Último acesso em fevereiro de 2025
4 Disponível em https://www.instagram.com/ .Último acesso em fevereiro de 2025
5 Disponível em https://web.whatsapp.com/ .Último acesso em fevereiro de 2025
6 Disponível em https://www.linkedin.com/ .Último acesso em fevereiro de 2025
7 Disponível em https://x.com/ .Último acesso em fevereiro de 2025
8 Disponível em https://www.tiktok.com/ .Último acesso em fevereiro de 2025
9 Disponivel em https://www.strava.com/ .Último acesso em fevereiro de 2025
10 Disponivel em https://www.tripadvisor.com.br/ .Último acesso em fevereiro de 2025
11 Disponivel em https://www.deviantart.com/ .Último acesso em fevereiro de 2025

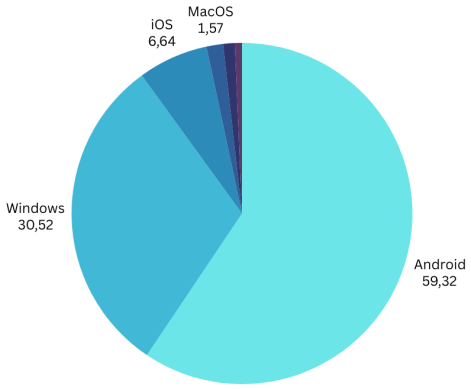
2.3 Aplicativos móveis

Nesta seção será apresentado os conceitos de dispositivos móveis, plataformas e linguagem utilizada para desenvolvimento da rede social proposta.

2.3.1 Plataformas móveis

Conforme apontando no estudo realizado por StatCounter Global Stats (2024), mais de 53% dos acessos realizados a internet são provenientes de dispositivos móveis. A Figura 1 representa graficamente a porcentagem dos acessos referidos, no período Outubro de 2023 a Novembro de 2024.

Figura 1 – *Sistemas Operacionais: Participação de Mercado no Brasil (out. 2023 – nov. 2024)*



Fonte: Adaptado de StatCounter Global Stats (2024).

Observando apenas os dispositivos móveis, a plataforma mais utilizada no Brasil é o Android, com aproximadamente 83%, seguido pelo iOS com 16%. A Figura 2 representa graficamente a porcentagem dos acessos referidos, no período Outubro de 2023 a Novembro de 2024.

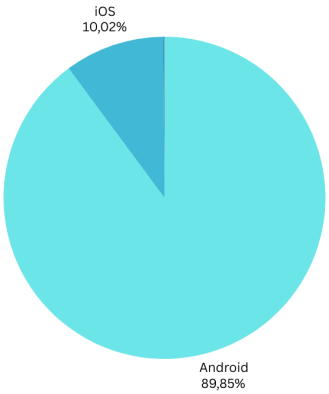
Considerando essas informações, para que uma rede social tenha como atingir o maior número de usuários, é ideal que seja realizado o desenvolvimento para dispositivos móveis.

2.3.1.1 Android

Com 83% dos dispositivos móveis no Brasil (STATCOUNTER GLOBAL STATS, 2024), o Android é um sistema operacional com seu *kernel open source* baseado em Linux mantido e aprimorado pela Google. Atualmente na sua versão 14, ele é encontrado nas principais marcas de *smartphones* comercializadas no território nacional (Lima, 2023).

Segundo Lima (2023), em 2008 o Android foi oficialmente lançado, porém a sua história teve início em 2003, na criação da Android Inc. O seu objetivo inicial era ser um sistema ope-

Figura 2 – *Sistemas Operacionais Móveis: Participação de Mercado no Brasil (out. 2023 – nov. 2024)*



Fonte: Adaptado de StatCounter Global Stats (2024)..

racional para câmeras digitais, porém esse mercado não apresentou potencial, o que fez com que a empresa mudasse seu foco.

Em 2005 a empresa foi comprada pelo Google, que assumiu o projeto, lançando seu primeiro dispositivo em parceria com a *High-Tech Computer Corporation* (HTC) em setembro de 2008 (Brito, 2017).

2.3.1.2 iOS

Presente nos dispositivos fabricados pela Apple, o iOS é um sistema operacional exclusivo para iPhones, atualmente na versão 18.1. Seu código-fonte é mantido pela própria Apple.

No artigo escrito por Coelho (2024), a história desse sistema operacional começou em 2007, com o lançamento do primeiro iPhone, mas em seu lançamento ainda era chamado iPhone OS 1. Como características iniciais, ele foi lançado com recursos básicos para uso do *smartphone*, como telefone, cliente de e-mail, navegador de internet entre outros aplicativos. Inicialmente ele não veio com uma loja de aplicativos, mas o recurso foi lançado na segunda versão, no ano de 2008, em conjunto com iPhone Kit de Desenvolvimento de Software, do inglês *Software Development Kit* (SDK) para os desenvolvedores.

O nome atual do Sistema Operacional (SO) foi lançado no ano de 2010, com mais alguns recursos importantes que estão presentes até hoje, como a possibilidade de multitarefas, pastas para organização dos aplicativos instalados e o FaceTime, o aplicativo para chamadas de vídeos.

2.3.2 Kotlin

Kotlin é uma linguagem de programação orientada a objetos, projetada pela Jetbrains, que iniciou o projeto em 2010, com o objetivo de criar uma linguagem de programação moderna,

concisa e segura para ser utilizada em Máquina Virtual Java, do inglês *Java Virtual Machine* (JVM) e que fosse interoperável com o Java (Alice, 2023).

Sua primeira versão oficial, a Kotlin 1.0, foi lançada ao público em fevereiro de 2016. Devido a popularidade da linguagem, em 2017 o Google anunciou que Kotlin seria uma linguagem oficial de desenvolvimento para Android (Alice, 2023).

2.3.2.1 Jetpack Compose

Anunciada para a comunidade de desenvolvedores em 2019, na Google I/O[12], o Jetpack Compose surgiu como um novo conceito para criar interfaces de usuários no Android, que até o momento utilizava a Linguagem Extensa de Marcação, do inglês *Extensible Markup Language* (XML) para desenhar tais interfaces. Seu lançamento oficial foi realizado em julho de 2021, após atingir sua versão estável, com sua segunda versão lançada em 2023, tendo também seu início como um *framework* para desenvolvimento multiplataforma.

As principais características do *framework*, no ponto de vista de desenvolvimento, é a abordagem de desenvolvimento declarativo de interfaces, de forma que o desenvolvedor possa usar de componentes criados no *framework*, para compor os elementos com os quais o usuário irá interagir. Devido a renderização da tela ser feita durante a execução, a interface do usuário atualiza conforme as informações mudam em tempo real, facilitando o desenvolvimento das aplicações (Tarasov, 2024).

2.3.2.2 Kotlin Multi-Platform

Kotlin Multi-Platform (KMP) surge como uma alternativa para desenvolvimento multiplataforma, estendendo as ferramentas familiares aos desenvolvedores Android para a criação de executáveis em diversas plataformas, como iOS. Seu objetivo é unificar o desenvolvimento ao permitir que a lógica de negócios seja escrita uma vez e compartilhada entre sistemas operacionais (JetBrains, 2024), agilizando o processo e promovendo consistência.

A arquitetura KMP baseia-se em módulos comuns (commonMain) para lógica de negócios e funcionalidades independentes de plataforma, e módulos específicos para cada sistema (androidMain, iosMain) (Kotlin Foundation, 2024).

[12] Conferência anual de tecnologia realizada pela Google, voltada principalmente para desenvolvedores de *software*. O nome "I/O"é uma sigla para *Input/Output* (Entrada/Saída), um conceito fundamental da computação, e também remete ao *slogan* do evento: *Innovation in the Open* (Inovação em Código Aberto).

A interação com APIs nativas é gerenciada pelo mecanismo *expect/actual*[13]: declarações *expect* no código comum são implementadas com *actual* nos módulos de plataforma, permitindo acesso a recursos nativos de forma abstrata e mantendo o código compartilhado agnóstico (Kotlin Foundation, 2024).

Os benefícios do KMP incluem reutilização de código, redução de custos e consistência funcional (JetBrains, 2024). Apesar de o framework ainda se encontrar em seu estado beta[14] para iOS durante a elaboração deste trabalho, já era considerado viável para produção em diversos cenários, conforme destacado pela Kotlin Foundation (2023). Desafios persistem, como a configuração por plataforma, a maturidade de bibliotecas de terceiros com suporte a KMP e a complexidade na gestão de dependências e *builds* específicos (JetBrains, 2024).

2.4 *AT-Protocol*

Conforme apresentado por Calixto (2023), o *Authenticated Transfer Protocol* (Protocolo de transferência autenticada) é uma linguagem de comunicação de código aberto, lançada em 2022. Esse padrão chamou a atenção com o lançamento da rede social Bluesky, que foi a primeira rede social que utilizou esse protocolo.

As redes que utilizam esse padrão podem ser consideradas redes sociais federadas, pois elas podem ser hospedadas em diferentes servidores, porém elas ainda conversam entre si dentro de um grande sistema. Outras características desse protocolo são:

- Os perfis podem ser transferidos entre servidores sem perda de dados, considerando que todas as contas são unificadas;
- Os dados de identidade pertencem ao usuário;
- Utilização do sistema Lexicon, um sistema global que facilita a inserção de novos esquemas e modificações nos códigos das plataformas.

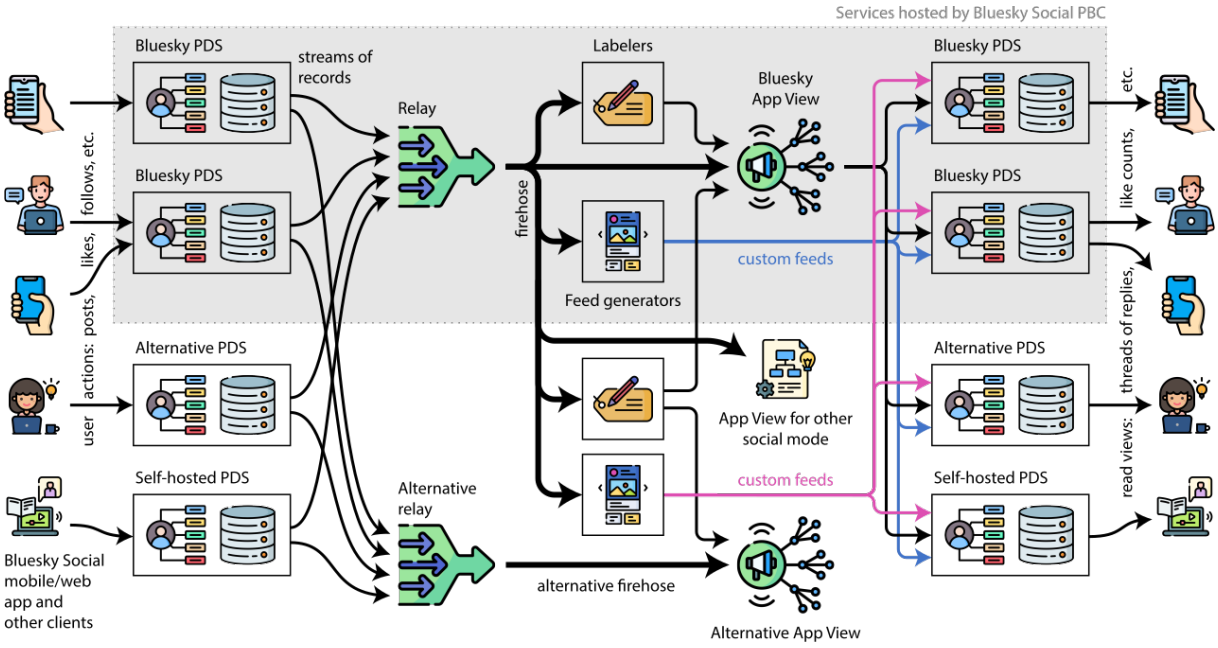
No ponto de vista técnico, Kleppmann *et al.* (2024) realizou a publicação de um artigo que explica em detalhes o funcionamento do protocolo. Baseado na Figura 3, é possivel verificar o fluxo de funcionamento do protocolo, assim como diferentes formas de visualização das postagens nas plataformas, nominadas como *Alternative App View*.

---

[13] O mecanismo `expect/actual` é o principal recurso do Kotlin Multiplatform para acessar APIs específicas de cada plataforma a partir do código compartilhado (`commonMain`). A palavra-chave `expect` é usada no módulo comum para declarar uma função, classe ou propriedade sem uma implementação, criando uma espécie de "promessa". Por sua vez, a palavra-chave `actual` é usada nos módulos de cada plataforma (ex: `androidMain`, `iosMain`) para fornecer a implementação concreta e específica daquela "promessa", utilizando as bibliotecas nativas da plataforma correspondente.

[14] Uma versão experimental ou de teste de um programa informático ou afim.

Figura 3 – Diagrama demonstrando a arquitetura do Bluesky e *AT-protocol*



Fonte: Kleppmann *et al.* (2024)

Os principais pontos a serem observados na estrutura é como os dados são armazenados e distribuídos.

- Os dados dos usuários são armazenados no Servidor Privado de Dados, do inglês *Private Data Server* (PDS);
- Os dados armazenados no PDS são incorporados ao restante da *Atmosphere*, pelos *Relay*[15];
- Após o processamento pelos *Relay*, é gerado os *feeds* que serão apresentados pelas *AppViews*, que irão permitir as interações nas postagens;
- As interações realizadas são armazenadas nos PDSs, que posteriormente passarão novamente pelos *Relays* para atualização dos dados da rede.

[15] Os *Relay*, são agregadores de conteúdo das PDS de maneira global, formando o que os desenvolvedores chamam de "mangueira de conteúdo", do inglês *firehose of content*.

3 MATERIAIS E MÉTODO

Neste capitulo serão apresentados os materiais e método utilizados na concepção e execução deste trabalho. A seção 3.1 apresenta as ferramentas utilizadas durante o processo de desenvolvimento. A seção 3.2 apresenta as principais etapas de desenvolvimento deste trabalho.

3.1 Materiais

O Quadro 1 apresenta a listagem de ferramentas utilizadas no desenvolvimento deste trabalho.

Quadro 1 – Ferramentas e tecnologias utilizadas no projeto

Ferramenta / Tecnologia	**Finalidade**
Android Studio	Ambiente de Desenvolvimento Integrado
AT Protocol	Protocolo de comunicação das postagens
Calf	Biblioteca para seleção de arquivos e operações de I/O no Jetpack Compose de forma multiplataforma
Coil3	Biblioteca para carregamento de imagens de forma assíncrona via internet
Kotlin Multi-Platform	*Framework* para desenvolvimento *front-end* e *back-end*
Ktor Client	Biblioteca para requisições HTTP de forma multiplataforma
Multiplatform Settings	Biblioteca para salvar pares de chave-valor de forma segura
Voyager	Biblioteca para navegação no Jetpack Compose de forma multiplataforma
XCode	Ambiente de Desenvolvimento Integrado

Fonte: Autoria própria (2025).

O *framework* de desenvolvimento escolhido foi o Kotlin Multi-Plataform, devido às ferramentas que ele oferece para o desenvolvimento de um *back-end* completo, assim como para o desenvolvimento de *front-end* para as principais plataformas de distribuição.

Considerando a linguagem de programação utilizada, a principal recomendação de ambiente integrado de desenvolvimento é o Android Studio, devido ao conjunto de ferramentas que facilitam o processo de desenvolvimento (GOOGLE, 2024).

Para a comunicação, o AT Protocol foi adotado em alinhamento com a natureza descentralizada do projeto, possibilitando que diferentes grupos operem seus próprios servidores, sem impedir a interconexão.

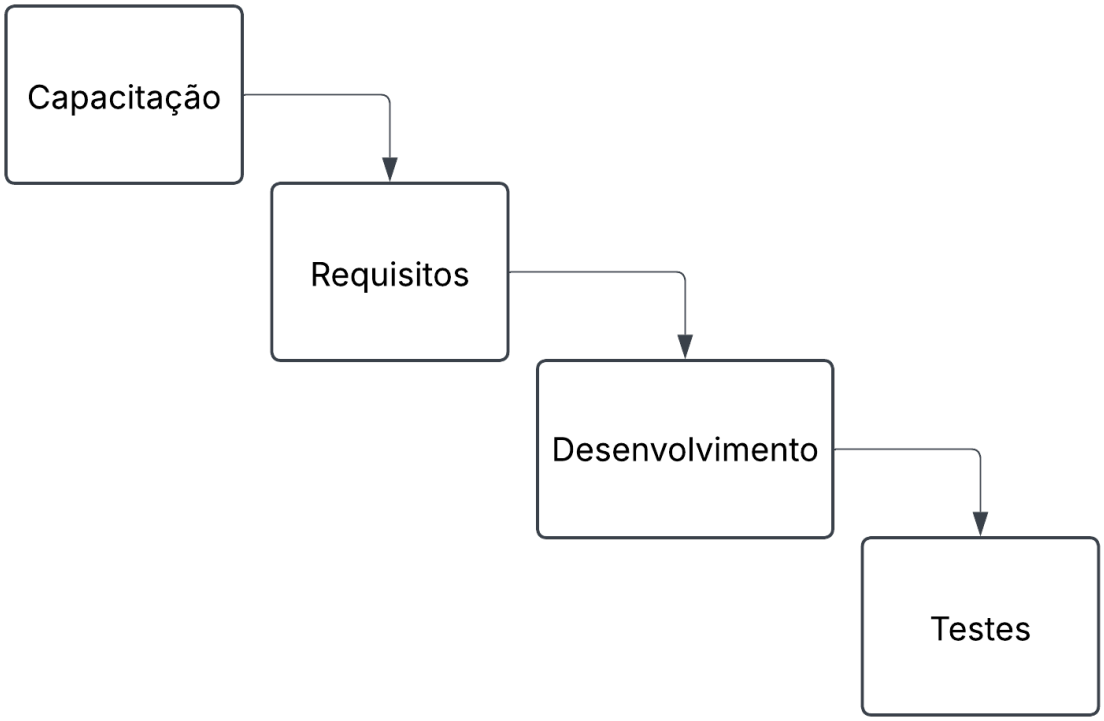
O XCode, Ambiente de Desenvolvimento integrado, do inglês *Integrated Development Environment* (IDE) nativo do MacOS, será utilizado para os testes e a publicação para iOS, devido à sua obrigatoriedade.

Devido as particularidades do protocolo, não haverá a necessidade de um Sistema Gerenciador de Banco de Dados (SGDB).

3.2 Método

O método selecionado para a execução desse trabalho é o Método Cascata, apresentado por Royce (1987). A Figura 4 representa a adaptação do método selecionado, visando os passos seguidos neste trabalho.

Figura 4 – Fluxo do trabalho



Fonte: Autoria própria (2025).

Inicialmente, aprofundou-se o conhecimento sobre o protocolo de comunicação das postagens, visto que este não havia sido abordado durante o período de graduação.

Com o entendimento desta etapa, deu-se início à fase de desenvolvimento, que começou com a definição detalhada dos requisitos da aplicação. Estes requisitos, por sua vez, nortearam a implementação da interface de usuário e de suas respectivas funcionalidades.

Para concluir o ciclo de desenvolvimento, realizou-se a etapa de testes. Nesta fase, validou-se o funcionamento da aplicação exclusivamente em ambientes de emulação, com o propósito de assegurar a correta implementação dos requisitos e o desempenho esperado do sistema.

4 RESULTADOS

4.1 Escopo do sistema

O sistema deverá ter as principais funções existentes em outras redes sociais, como realizar postagens e reagir a postagens dos demais membros da rede.

O sistema em questão terá apenas um ator, os membros. As funções a serem utilizadas serão a criação de postagens, interagir com as demais postagens (reagir, comentar, compartilhar ou reportar).

O acesso ao sistema é feito mediante login e senha, criado por meio da rede social Bluesky.

4.2 Modelagem do sistema

Nesta seção será apresentado os requisitos levantados para o sistema proposto, assim como a sua modelagem e casos de uso expandidos.

4.2.1 Requisitos Funcionais e Requisitos Não Funcionais

O Quadro 2 apresenta os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) elencados para a aplicação. A representação utilizada neste trabalho inspira-se nos modelos estudados durante a disciplina de Análise de Sistemas, da grade curricular da graduação. Esta abordagem, por sua vez, fundamenta-se nas recomendações de autores clássicos da área, como Pressman e Maxim (2016) e Sommerville (2019), e está em conformidade com as diretrizes do padrão ISO/IEC/IEEE (2018).

Quadro 2 – Requisitos Funcionais e Não-Funcionais do Sistema

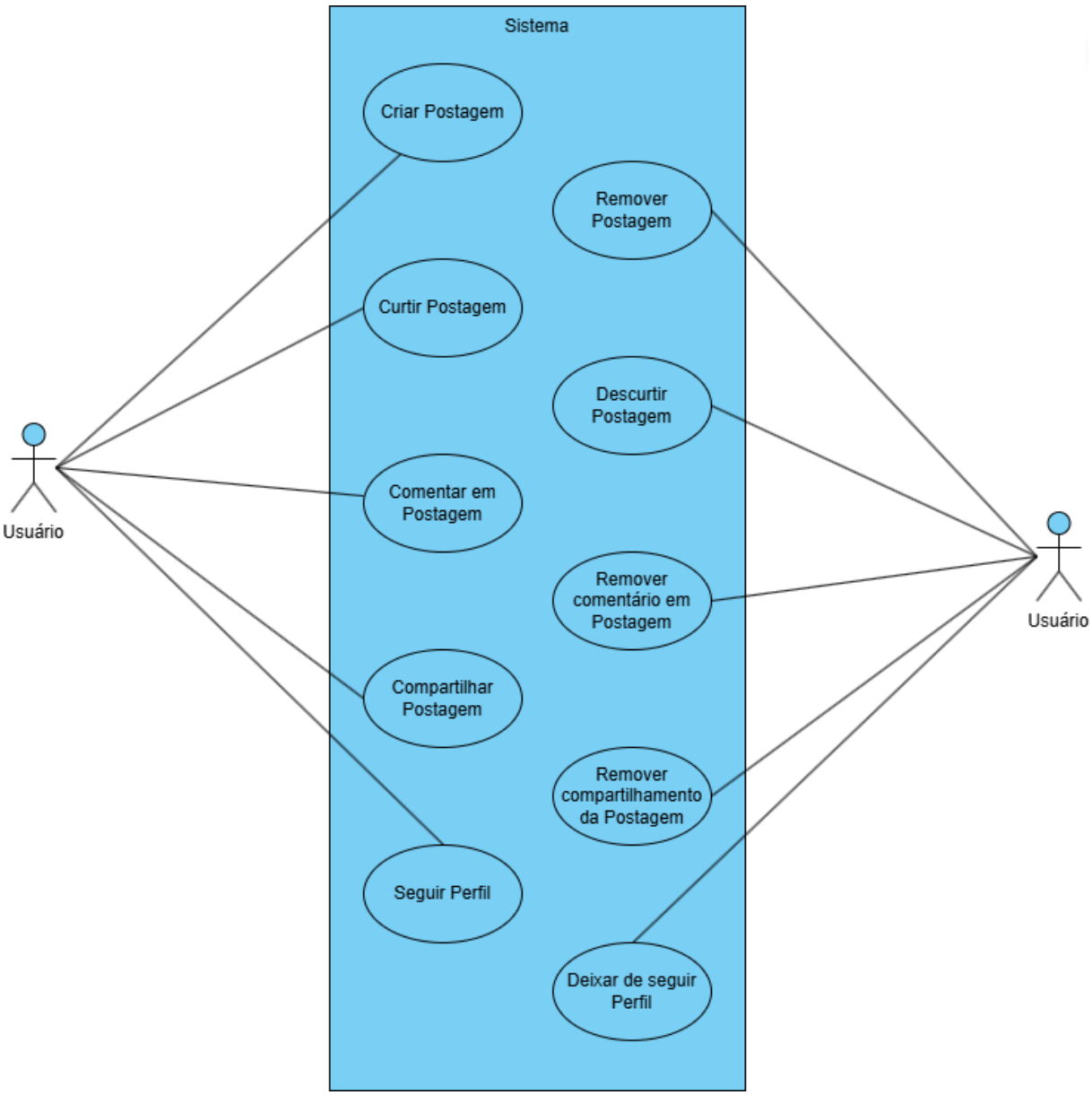
Requisitos Funcionais (RF)		
Identificador	**Descrição**	**Categoria**
RF1 - Postagens	O sistema deve permitir que o usuário realize criação de postagens com texto e/ou imagem.	Funcionalidade
RF2 - Interações	O sistema deve permitir que o usuário interaja com postagens via curtidas, comentários e compartilhamentos.	Funcionalidade
RF3 - Descoberta	O sistema deve oferecer uma interface para descoberta de conteúdos, com categorias pré-definidas.	Funcionalidade
Requisitos Não-Funcionais (RNF)		
Identificador	**Descrição**	**Categoria**
RNF1 - Desempenho de Imagem	As imagens do feed devem ter seu carregamento principal concluído em no máximo 2 segundos em uma conexão 4G.	Desempenho
RNF2 - Compatibilidade	A aplicação deve ser compatível com as versões do Android 8.0 (API 26) ou superior e iOS 14 ou superior.	Compatibilidade
RNF3 - Segurança de Dados	Os tokens de autenticação (JWT) devem ser armazenados de forma segura no dispositivo, utilizando a Keystore (Android) e a Keychain (iOS).	Segurança

Fonte: Autoria própria (2025).

4.2.2 Diagrama de casos de uso

A Figura 5 apresenta o diagrama de casos de uso, contendo as funcionalidades planejadas para o sistema e seus atores, que neste caso será apenas o usuário. O ator “Usuário” refere-se a todos os usuários do sistema, que serão os membros da comunidade cosplay.

Figura 5 – Casos de uso



Fonte: Autoria própria (2025).

4.2.3 Expansão de casos de uso

As tabelas numeradas de 3 a 5 demonstram a sequencia de passos executados para realizar algumas das ações dentro da plataforma.

Quadro 3 – Caso de uso expandido 1

Caso de Uso: Realizar uma postagem
Atores: Usuário.
Pré-condições: O usuário deve estar cadastrado na plataforma e autenticado.
Pós-condições: Não se aplica.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. [IN] O usuário abre o aplicativo e clica no botão de nova postagem. 2. [OUT] O Sistema apresenta ao usuário uma interface com um campo de texto e alguns botões para carregar arquivos. 3. [IN] O usuário seleciona o arquivo (foto ou video) que deseja publicar. 4. [IN] O usuário preenche uma descrição e/ou *hashtags* para a postagem. 5. [IN] O usuário clica no botão publicar. 6. [OUT] O sistema realiza o carregamento do arquivo para a plataforma (exceção). 7. [OUT] O sistema notifica que a publicação foi realizada com sucesso. 8. [OUT] O usuário é direcionado novamente para a tela de inicio do aplicativo. 9. Caso de uso encerrado.
Tratamento de Exceções e Variantes: **Exceção 6a:** Arquivo superior ao tamanho aceito. 6a.1 [OUT] O sistema apresenta uma notificação de tamanho excedido. 6a.2 [OUT] O sistema retorna no passo 2.

Fonte: Autoria própria (2025).

Quadro 4 – Caso de uso expandido 2

Caso de Uso: Reagir uma postagem
Atores: Usuário.
Pré-condições: O usuário deve estar cadastrado na plataforma, autenticado e na tela de *feed*.
Pós-condições: Não se aplica.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. [IN] O usuário clica no botão curtir. 2. [OUT] O sistema realiza um *feedback* visual para indicar que a postagem foi curtida. 3. [IN] O usuário desliza o dedo na tela para carregar a próxima postagem. 4. [OUT] O sistema apresenta a próxima postagem. 5. Caso de uso encerrado.
Tratamento de Exceções e Variantes: Não há exceções para esse caso.

Fonte: Autoria própria (2025).

Quadro 5 – Caso de uso expandido 3

Caso de Uso: Remover uma postagem
Atores: Usuário.
Pré-condições: O usuário deve estar cadastrado na plataforma, autenticado e com a postagem a ser removida na tela.
Pós-condições: Não se aplica.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. [IN] O usuário clica no ícone com 3 pontos verticais na postagem. 2. [OUT] O Sistema apresenta ao usuário um menu suspenso com a opção **Remover Postagem**. 3. [IN] O usuário clica no botão **Remover Postagem**. 4. [OUT] O sistema abrirá um *pop-up* para o usuário confirmar a ação de remover. 5. [IN] O usuário clica no botão **confirmar**. 6. [OUT] O sistema retorna um *feedback* postagem removida com sucesso. 7. [OUT] O sistema continua a exibição do *feed*. 8. Caso de uso encerrado.
Tratamento de Exceções e Variantes: Não há exceções para esse caso.

Fonte: Autoria própria (2025).

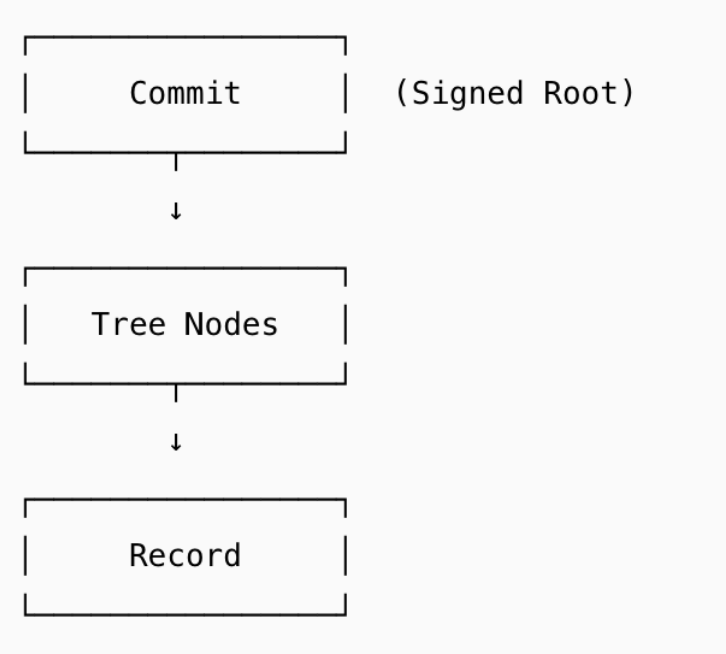
4.2.4 Arquitetura de persistência

Baseado na forma que o protocolo de comunicação utilizado na aplicação, alguns detalhes sobre como os dados serão armazenados funcionam de uma maneira única.

Os dados dentro do protocolo são armazenados no esquema Lexicon, que são salvos dentro de arquivos no formato de Notação de Objeto JavaScript, do inglês *JavaScript Object Notation* (JSON).

Esses arquivos ficam organizados de acordo com a Figura 6, que representa a estrutura dentro dos servidores privados de dados, onde o *Commit* é a raiz do diretório do usuário registrado, *Tree Nodes* são as coleções pertencentes ao usuário e *Record* é recurso a ser acessado.

Figura 6 – *Layout* dos dados



Fonte: Autoria própria (2025).

Utilizando um exemplo mais didático, com o padrão de acesso ao recurso no protocolo, a Figura 7 representa na sua primeira linha o usuário registrado, na segunda linha apresenta a coleção de *post* dentro da árvore de coleções e na ultima linha representa o *post* que está sendo acessado.

Figura 7 – Exemplo utilizando URI

```
Root       | at://alice.com
Collection | at://alice.com/app.bsky.feed.post
Record     | at://alice.com/app.bsky.feed.post/1234
```

Fonte: Autoria própria (2025).

Para a representação efetiva de como os dados são estruturados, a Listagem 1 apresenta tanto como é a requisição realizado pelo cliente do protocolo, assim como o corpo da resposta recebida do servidor. Foi representado o recurso de buscar os dados do perfil.

Listagem 1 – Exemplo de dados em Lexicon

```
{
  "lexicon": 1,
  "id": "com.example.getProfile",
  "type": "query",
  "parameters": {
    "user": {"type": "string", "required": true}
  },
  "output": {
    "encoding": "application/json",
    "schema": {
      "type": "object",
      "required": ["did", "name"],
      "properties": {
        "did": {"type": "string"},
        "name": {"type": "string"},
        "displayName": {"type": "string", "maxLength": 64},
        "description": {"type": "string", "maxLength": 256}
      }
    }
  }
}

```

Fonte: Autoria própria (2025).

4.3 Apresentação do sistema

Nesta seção, é apresentado o sistema desenvolvido no âmbito deste trabalho, denominado **GCSN**. As figuras a seguir ilustram as suas principais telas e o fluxo de interação para um usuário autenticado.

4.3.1 Tela de login

A tela de login é composta de 4 elementos visuais claros, utilizando o Material Design 3[1]. O conteúdo da tela é desenhado dentro do "Scaffold" (elemento não visível) que serve como base para todas as telas do sistema.

Dos elementos visíveis, há uma barra de navegação no topo da tela, com o título e um botão de ação para voltar a tela anterior.

[1] Sistema de design (ou linguagem de design) desenvolvido pela Google para unificar a experiência do usuário em suas plataformas. A versão 3, também conhecida como *Material You*, representa uma grande evolução ao focar na personalização. Sua característica mais notável é a tematização dinâmica, que adapta as cores dos componentes da interface de um aplicativo com base no papel de parede e nas preferências do usuário, criando uma experiência visual coesa e pessoal.

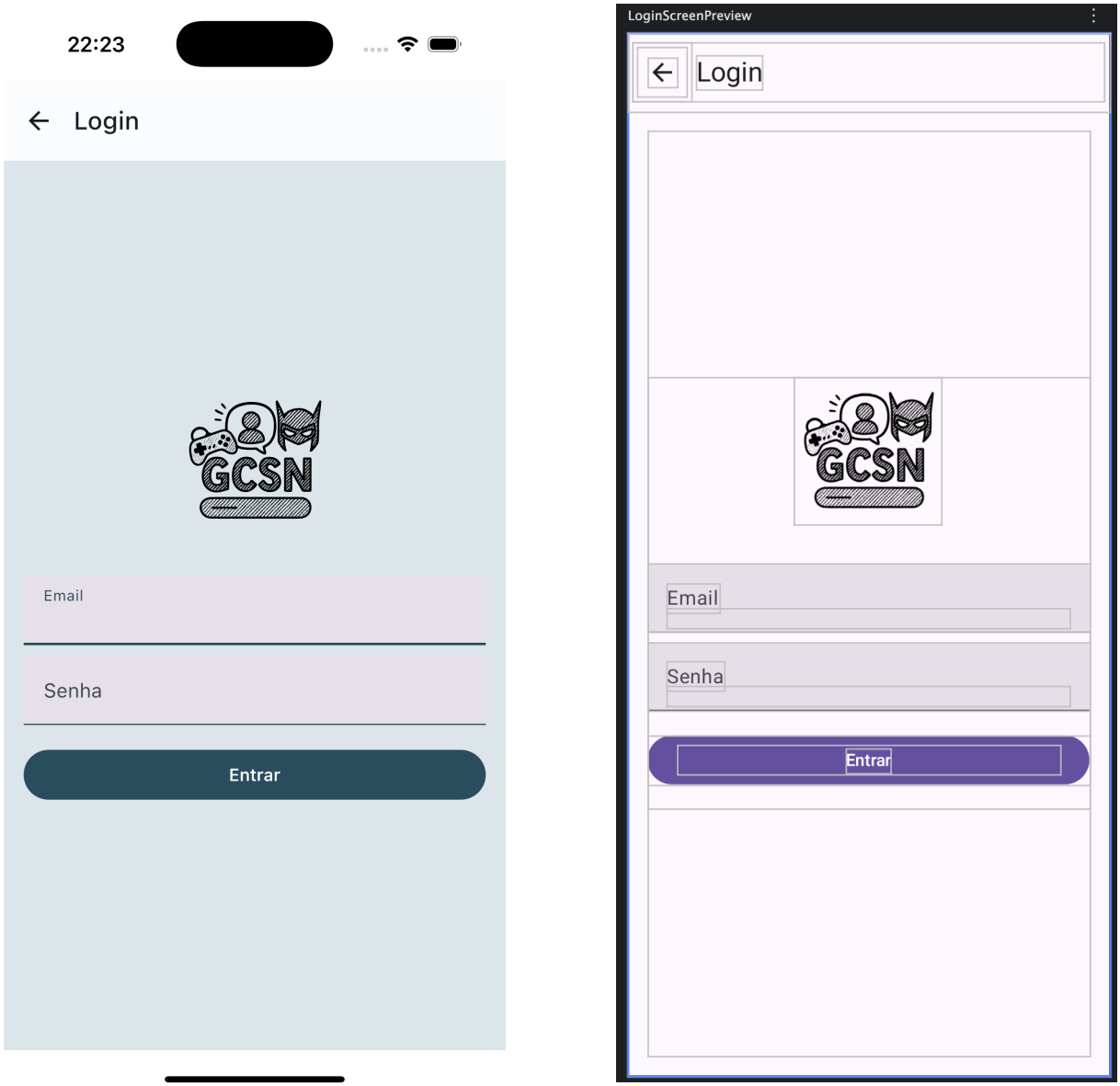
No centro da tela, encontra-se dois campos de texto, identificados como "email" e "senha", respectivamente, onde o usuário irá preencher suas credenciais de acesso.

Logo abaixo, há um botão com um texto inicial "Entrar", que ao ser clicado, executará a lógica de login com a API utilizada.

A composição da tela de login é detalhada na Figura 8, que exibe tanto a sua aparência final, como visto na subfigura (a), quanto a sua estrutura interna de componentes, representada na subfigura (b).

Figura 8 – Composição da tela de login de usuário

(a) Interface final renderizada **(b) Estrutura de componentes (Wireframe)**



Fonte: Autoria própria (2025).

4.3.2 Tela de inicio

A tela de início é a tela onde o usuário é direcionado após o login com sucesso na plataforma. Essa é composta dos seguintes elementos visuais:

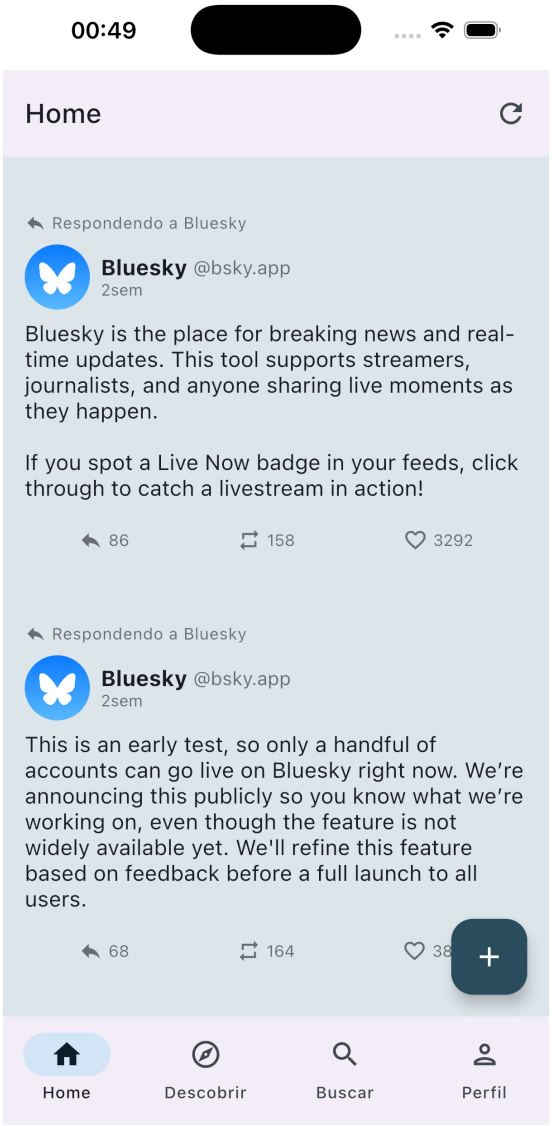
- No topo da tela há uma barra de navegação, onde há o título da página e um botão de ação para recarregar o *feed*;
- No centro da tela, há um componente reutilizável que realiza o carregamento das informações das postagens dos perfis seguidos pelo usuário. Os elementos que compõe esse item são:
  - Um *card* onde apresenta as informações básicas do perfil, como *avatar*, nome de exibição, *handle* (também conhecido como @ do usuário) e um indicador com o tempo que foi realizado a postagem. Nesse elemento, ao ser clicado, o usuário é direcionado ao perfil da postagem;
  - Caso a postagem possua imagens ou *link* externo, será apresentado um *card* com a imagem ou com informações gerais do *link* incorporado;
  - Botões de ações para visualizar respostas (ou comentar), compartilhar e curtir.

As telas que realizam carregamento de postagens utilizam um componente para carregamento lento, que em sua lógica, quando estiver próximo do fim, realiza uma nova requisição para buscar mais postagens.

Na parte inferior da tela, há uma barra de navegação que permite navegar para as diferentes funcionalidades do sistema.

A Figura 9 é a representação visual da tela descrita.

Figura 9 – Telas - Homepage



Fonte: Autoria própria (2025).

4.3.3 Descoberta

Ao utilizar a opção **descobrir**, na barra de navegação inferior, o usuário é direcionado para a tela de descoberta.

A tela de descoberta é similar a interface da tela de inicio, composta dos mesmos elementos, com o diferencial que é apresentado ao usuário uma seleção de postagens da plataforma que foram indicadas como ***Cosplay***.

A Figura 10 é a representação visual da tela descrita.

Figura 10 – Telas - Descoberta

Fonte: Autoria própria (2025).

4.3.3.1 Perfis

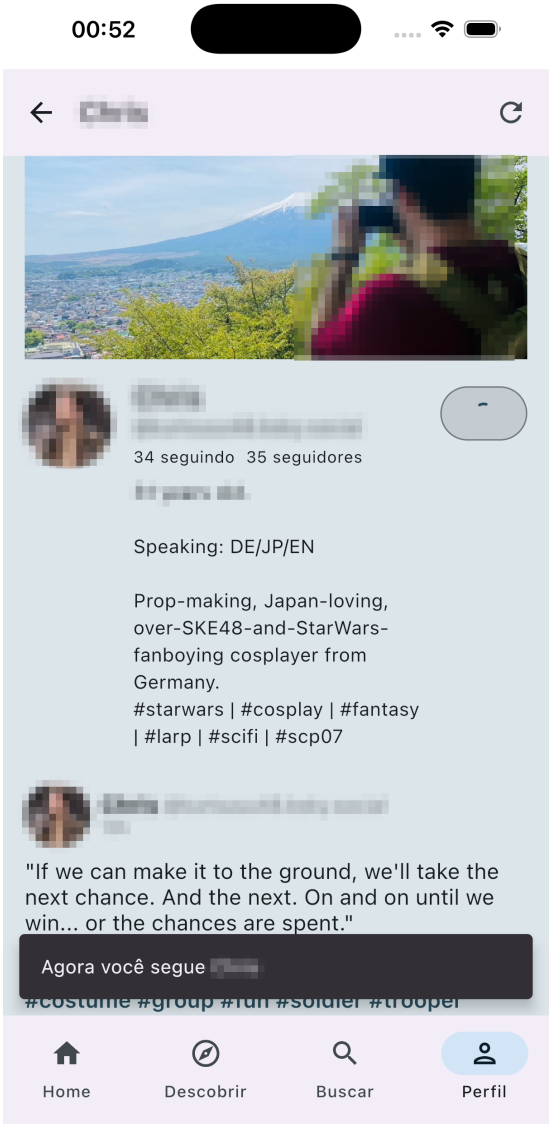
Ao clicar em um perfil, o usuário é direcionado ao perfil do autor da postagem. A tela apresentada ao usuário é composta da seguinte forma:

- No topo da tela, na barra de navegação, é apresentado o nome do perfil.
- Logo a baixo, há um *card* que apresenta as informações do usuário, como *banner*, imagem de perfil, nome e estatisticas do perfil. Nesse *card*, caso o usuário possua uma breve descrição, também será apresentada.
- Um botão de ação que, caso o usuário não siga o perfil, permite seguir. Caso já siga, o botão apresenta a opção para deixar de seguir.

- Caso o perfil possua postagens, apresenta uma listagem similar ao apresentado na tela de inicio.

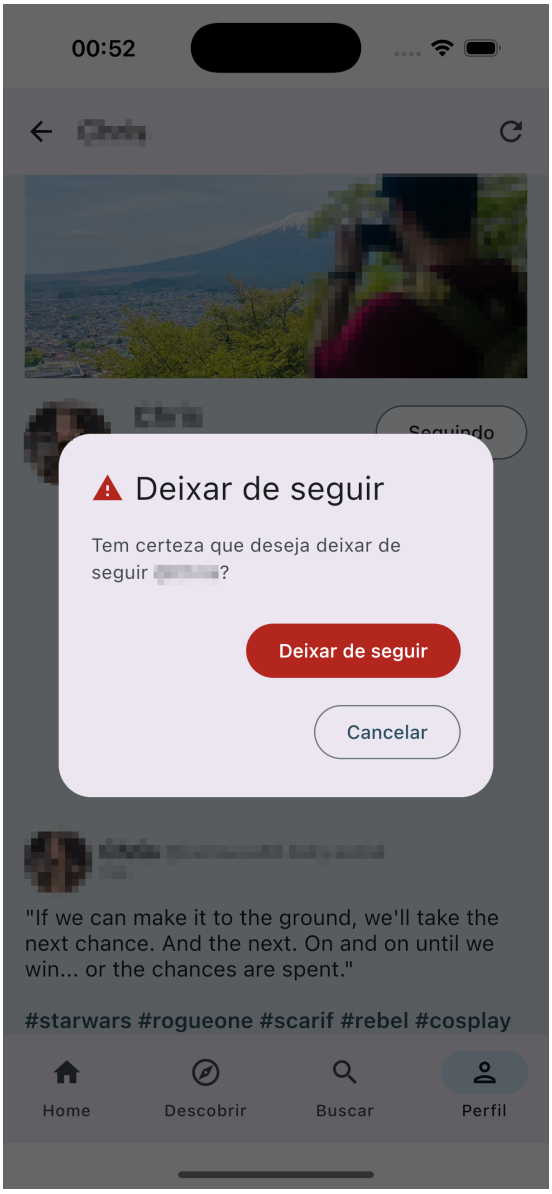
As Figuras 11 a 13 apresentam como essa tela se comporta durante as ações referidas.

Figura 11 – Telas - Perfil (Solicitar *follow*)



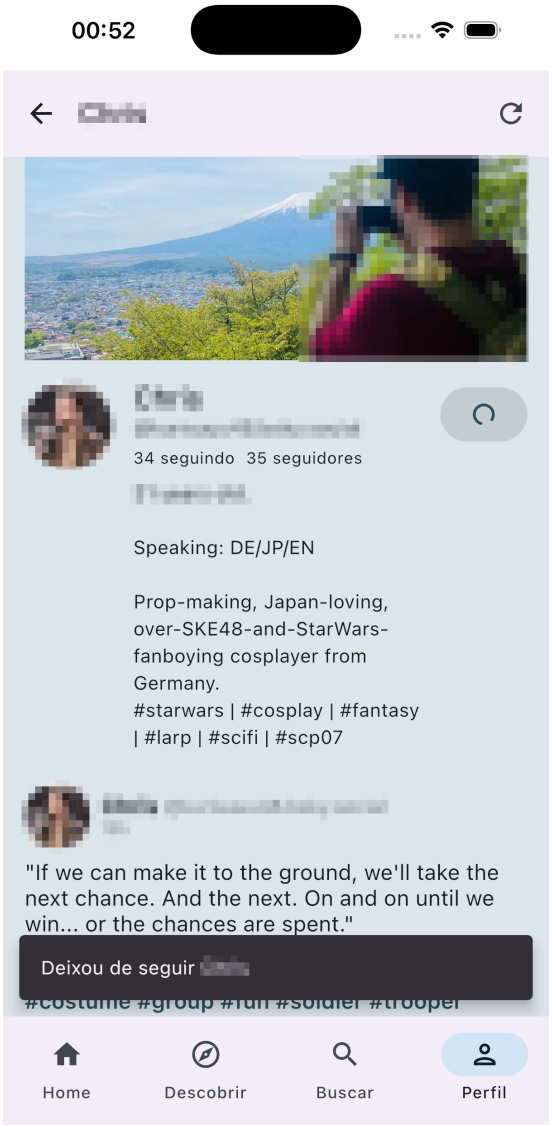
Fonte: Autoria própria (2025).

Figura 12 – Telas - Perfil (solicitar *unfollow*)



Fonte: Autoria própria (2025).

Figura 13 – Telas - Perfil (*unfollow* com sucesso)

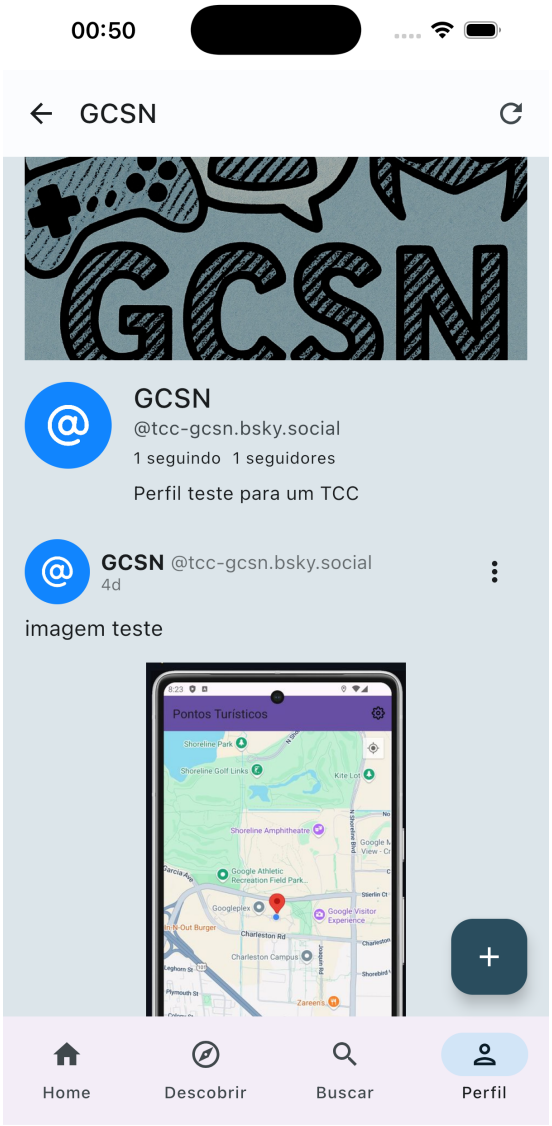


Fonte: Autoria própria (2025).

Caso o usuário não esteja na tela de perfil de algum usuário, ao clicar no botão **Perfil**, na barra de navegação inferior, será direcionado para o seu próprio perfil.

A Figura 14 é a representação visual da tela descrita.

Figura 14 – Telas - Perfil do usuário



Fonte: Autoria própria (2025).

4.3.4 Comentários

Ao usuário clicar no botão de ação **comentários**, em uma postagem, o usuário é direcionado para uma tela de comentários.

Essa tela apresentará um detalhamento da postagem original, assim como os comentários vinculados a essa postagem. Os elementos visuais são os mesmos das postagens apresentadas anteriormente, com o diferencial na parte inferior, onde há um campo de texto, para preenchimento do comentário, e um botão de enviar.

A Figura 15 é a representação visual da tela descrita.

Figura 15 – Telas - Comentários

Fonte: Autoria própria (2025).

4.3.5 Pesquisa

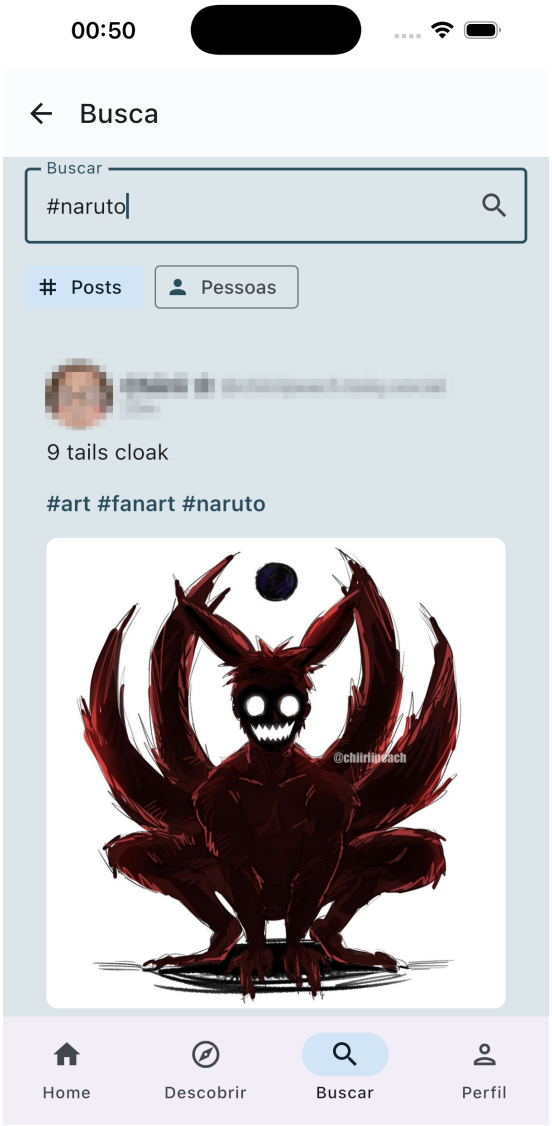
Esta tela pode ser apresentada ao usuário de duas forma: clicando no botão buscar ou clicando em *hashtags* nas postagens.

Os elementos visuais desta tela se destacam pela caixa de pesquisa, seguida por dois botões (postagem e perfil), para definir o tipo de pesquisa que será realizada.

Ao iniciar uma pesquisa, no centro da tela irá apresentar o resultado de acordo com a pesquisa realizada.

A Figura 16 é uma representação da tela, ao realizar a busca pelo termo **#naruto**.

Figura 16 – Telas - Busca



Fonte: Autoria própria (2025).

4.3.6 Nova postagem

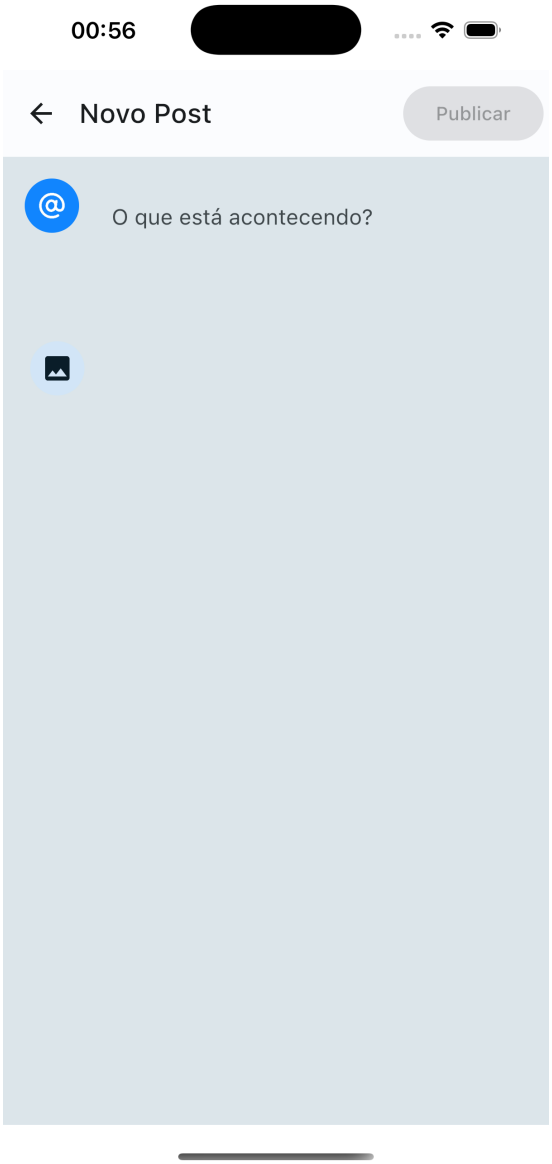
Nas telas de inicio, descoberta ou seu próprio perfil, na parte inferior da tela, há um botão de ação flutuante. Esse botão direciona o usuário para a tela de nova postagem.

Visualmente, esta tela é composta de uma barra de navegação no topo, onde há um botão de ação para realizar a postagem.

No corpo da tela, há um campo de texto onde o usuário colocará o texto da sua postagem e um botão de ação que permite buscar uma foto no dispositivo do usuário, para anexar à postagem.

A Figura 17 é uma representação da tela descrita.

Figura 17 – Telas - Nova postagem



Fonte: Autoria própria (2025).

4.4 Implementação do sistema

A implementação do sistema foi realizada apenas a criação da *AppView*. As funções de servidor não precisaram ser implementadas, pois a empresa que realizou o desenvolvimento do protocolo, a Bluesky Social PBC (2022) disponibiliza algumas ações via API, desde que seja respeitado alguns limites estabelecidos na documentação (BLUESKY SOCIAL PBC, 2025b).

4.4.1 Criação do projeto

A criação do projeto foi realizado com o auxilio da ferramenta oficial da Jetbrains para criação de projetos em Kotlin Multiplataforma[2].

4.4.2 Configuração do ambiente

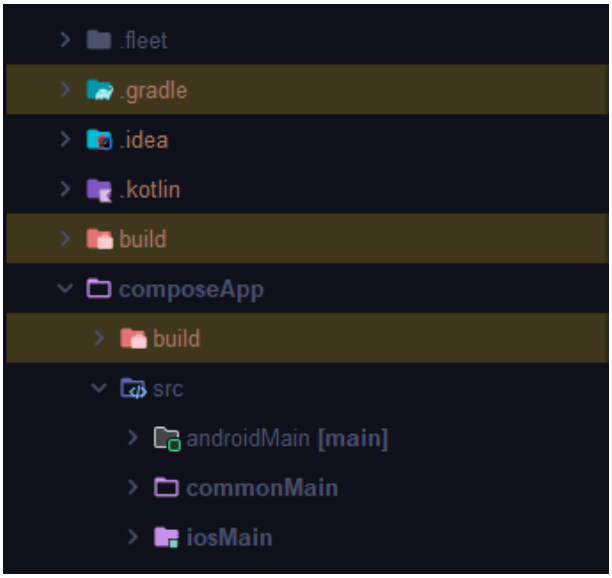
Após o *download* e instalação da IDE selecionada para o desenvolvimento, foi realizado a importação do projeto gerado pelo *wizard* e realizado a sincronização inicial do *Gradle*[3].

Neste momento também foi realizado a inclusão de algumas bibliotecas que foram utilizadas desde o inicio do desenvolvimento, como a biblioteca Ktor[4] e algumas dependências extras do Kotlin utilizadas no processo de serialização dos JSON recebidos como retorno das operações realizadas com a API do Bluesky. A biblioteca foi escolhida por possuir implementações disponiveis para uso multiplataforma.

4.4.3 Organização do projeto

Conforme apresentado na Figura 18, os projetos em KMP seguem um padrão onde cada plataforma que será desenvolvida tem seus arquivos de lógica agrupados e os arquivos de lógica compartilhada ficam dentro da pasta ***commomMain***.

Figura 18 – Estrutura do projeto



Fonte: Autoria própria (2025).

[2] Disponível em https://kmp.jetbrains.com/. Último acesso em Maio de 2025

[3] Gradle é uma ferramenta de automação de compilação (*build*) avançada que agiliza o desenvolvimento de software, gerenciando dependências e o processo de *build* de forma flexível e eficiente.

[4] Disponivel em https://ktor.io/docs/welcome.html. Último acesso em Maio de 2025

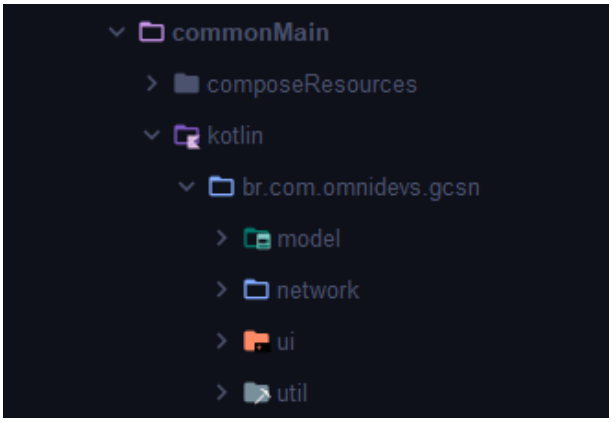
Nas pastas de plataformas específicas, há uma organização de arquivos de acordo com o esperado para um projeto de desenvolvimento na "linguagem nativa".

Como o projeto tem foco para uso multiplataforma, a maior parte do código é presente na ***commomMain***, exceto os códigos utilizado para acessar funcionalidades nativas que ainda não possuem uma biblioteca para uso.

Dentro das pastas comuns, há uma pasta de recursos compartilhados, onde ficam os arquivos fixos utilizados na aplicação. Dentro desta pasta também há uma pasta para os arquivo da linguagem Kotlin, onde a lógica compartilhada fica.

A organização dos arquivos do projeto seguem um padrão ***Model-View-Controller***[5], conforme apresentado na Figura 19.

Figura 19 – Estrutura do projeto - parte 2



Fonte: Autoria própria (2025).

- Na pasta **Model** está presente as *data classes* do Kotlin;
- Na pasta **Network** contém as classes que interagem com a API;
- Na pasta **UI** está presente as interfaces que os usuários usarão;
- Na pasta **Util** está as funções auxiliares para a aplicação.

4.4.3.1 *Model*

A Listagem 2 apresenta um exemplo de *model* utilizada no sistema, identificada como ***Actor***, que de acordo com as definições do protocolo, refere-se ao usuário na plataforma e contém as suas informações básicas.

[5] Padrão de arquitetura de *software* que promove a separação de preocupações em três camadas. O **Model** encapsula os dados da aplicação e notifica as Views sobre mudanças. A **View** renderiza a representação visual dos dados. O **Controller** recebe as interações do usuário (como cliques e entradas de teclado), manipula o Model conforme necessário e seleciona a View apropriada para responder ao usuário. Essa estrutura facilita a manutenção e a evolução do código.

Listagem 2 – Modelo de Dados para Atores (*Actor Data Class*)

```kotlin
package br.com.omnidevs.gcsn.model.actor

import br.com.omnidevs.gcsn.model.Label
import br.com.omnidevs.gcsn.model.post.Viewer
import kotlinx.serialization.Serializable

@Serializable
data class Actor(
    val did: String,
    val handle: String,
    val displayName: String? = null,
    val avatar: String? = null,
    val associated: Associated,
    val viewer: Viewer,
    val labels: List<Label> = emptyList(),
    val createdAt: String,
    val description: String? = null,
    val indexedAt: String,
    val banner: String? = null,
    val followersCount: Int = 0,
    val followsCount: Int = 0,
    val postsCount: Int = 0
)

```

Fonte: Autoria própria (2025).

A anotação **@Serializable** presente acima da definição de *data class*, pertence a uma das bibliotecas adicionais do Kotlin, onde permite a serialização do JSON recebido como resposta da API diretamente para um objeto.

4.4.3.2 *Network*

A Listagem 3 é um exemplo de chamada para API, realizada pelo sistema, para o *endpoint* que retorna as informações do *Actor*.

Listagem 3 – Modelo de chamada de API

```
package br.com.omnidevs.gcsn.network.api

import br.com.omnidevs.gcsn.network.HttpClientProvider
import br.com.omnidevs.gcsn.model.actor.Actor
import br.com.omnidevs.gcsn.util.AppDependencies


    class BlueskyApi {

    private val client = HttpClientProvider
    .getClient(AppDependencies.authService)

    suspend fun getProfile(actor: String): Actor {
        return
client.get("https://bsky.social/xrpc/app.bsky.actor.getProfile") {
            url.parameters.append("actor", actor)
        }.body()
    }
}

```

Fonte: Autoria própria (2025).

O uso da *keyword* **suspend** nessa função é um indicativo que, na tela onde essa função será chamada, utilizará de co-rotinas do Kotlin para evitar o travamento do aplicativo enquando a requisição é realizada e processada na aplicação.

A classe *HttpClientProvider*, localizada no pacote **Util**, foi criada para configurar o cliente HTTP com um interceptador (Interceptor) de requisições. A principal função deste interceptador é garantir a autenticação automática em todas as chamadas à API, seguindo a especificação do AT Protocol que adota o *JSON Web Token* (JWT) como padrão[6]. O processo ocorre em duas etapas: primeiramente, o interceptador recupera o JWT criptografado do armazenamento do dispositivo; em seguida, injeta-o em todas as requisições, adicionando o cabeçalho ‘Authorization: Bearer <token>‘.

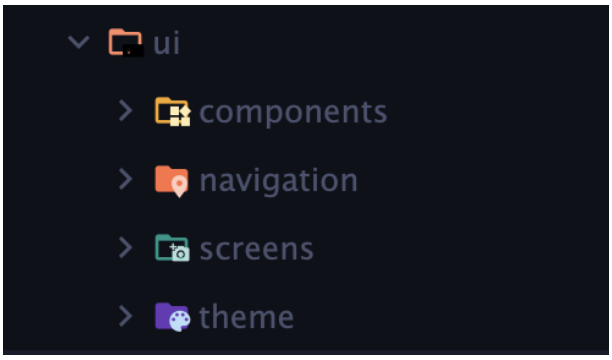
[6] O JWT (JSON Web Token) é utilizado como padrão de autenticação na API do Bluesky por ser uma tecnologia especialmente adequada para arquiteturas distribuídas e descentralizadas, como a do AT Protocol. Sua principal vantagem é ser *stateless* (sem estado): o token contém em si todas as informações necessárias para verificar a identidade e as permissões do usuário, eliminando a necessidade de o servidor armazenar dados de sessão. Cada token é assinado digitalmente para garantir sua autenticidade e integridade. O AT Protocol implementa um fluxo de segurança robusto utilizando dois tipos de JWT: um *access token* (de curta duração, para as requisições comuns) e um *refresh token* (de longa duração, usado apenas para obter um novo *access token*). Disponível em: <https://atproto.com/specs/auth>. Acesso em: 3 jul. 2025.

4.4.3.3 Interfaces do usuário (UI)

Neste pacote, se encontra as interfaces do usuário, onde é declarado como as telas são apresentadas para o usuário.

A Figura 20 apresenta a estrutura do pacote que é composto de 4 diretórios.

Figura 20 – Pacote de interface de usuário (UI)



Fonte: Autoria própria (2025).

- O diretório *components* ficam arquivos de elementos visuais que podem ser utilizados nas telas do sistema;
- O diretório *navigation* possui os arquivos necessários para a utilização da biblioteca *Voyager*, utilizada para a navegação entre telas para Compose Mulitplataforma;
- O diretório *screens*, onde fica o código fonte e lógicas das telas desenvolvidas nesse projeto;
- O diretório *theme* é onde são definidos os esquemas de cores e demais elementos do tema utilizado na aplicação

A Listagem 4 representa, parcialmente, o código escrito para a tela de *login*. Devido ao uso da biblioteca *Voyager*, a estrutura é um pouco diferente de como seria uma tela normal. Alguns detalhes foram removidos para facilitar a leitura do código.

Listagem 4 – Implementação parcial da tela de login

```
1   package br.com.omnidevs.gcsn.ui.screens
2   //import ocultados para melhor legibilidade
3
4   class LoginScreen : Screen {
5       @OptIn(ExperimentalMaterial3Api::class)
6       @Composable
7       override fun Content() {
8       //Variáveis locais ocultadas para melhor legibilidade
9           Scaffold(
10              topBar = {
11      //Conteúdo da topBar ocultado para melhor legibilidade
12              }
13          ) { paddingValues ->
14              Column(
15                  modifier = Modifier
16                      .fillMaxSize()
17                      .padding(paddingValues)
18                      .padding(16.dp),
19                  verticalArrangement = Arrangement.Center
20              ) {
21                  TextField(
22                      value = email,
23                      onValueChange = { email = it },
24                      label = { Text("Email") },
25                      modifier = Modifier.fillMaxWidth()
26                  )
27                  TextField(
28                      value = password,
29                      onValueChange = { password = it },
30                      label = { Text("Senha") },
31                      modifier = Modifier.fillMaxWidth(),
32                      visualTransformation =
    PasswordVisualTransformation()
33                  )
34                  Button(
35                      onClick = {
36      //lógica de login ocultada para melhor legibilidade
37                      },
38                      modifier = Modifier.fillMaxWidth()
39                  ) {
40                      Text(if (isLoading) "Carregando..." else
    "Entrar")
41                  }
42              }
43          }
44      }
45  }
46
```

Fonte: Autoria própria (2025).

As principais diferenças da implementação realizada e uma implementação tradicional com Jetpack Compose é que a classe da tela extende a interface *Screen* da biblioteca, que por sua vez faz a implementação do **@Composable**, que é a anotação onde traz os elementos visuais que serão criados ao compilar o projeto.

O Jetpack Compose utiliza de uma abordagem descritiva para fazer a composição da tela. No código apresentado temos a seguinte estrutura:

- O **Scaffold** é o "Esqueleto da tela". Ele possui alguns parâmetros como o apresentado **TopBar**, que é um elemento visual presente na parte superior da tela, onde fica o título da tela e também pode possuir alguns botões de ação;
- Dentro do **Scaffold** é desenhado o restante da tela, que no exemplo é composto por uma coluna, com dois campos de texto, para inserir o email e senha utilizado no login, e um botão com uma ação de clique que executa a lógica de login do protocolo.

4.4.3.4 Util

Neste pacote, encontra-se todas as classes de utilitários necessárias na aplicação, entre elas, como exemplo, foi elencado o **SecureStorage.kt**

A Listagem 5 representa parcialmente o código presente na classe, onde há uma peculiaridade do *framework* utilizado, que é o uso de *expect/actual* classes.

Listagem 5 – Implementação parcial do *Secure Storage*

```kotlin
//composeApp/src/commonMain/kotlin/../util/SecureStorage.kt
package br.com.omnidevs.gcsn.util

import com.russhwolf.settings.Settings

expect class SecureStorageProvider {
    fun getSecureSettings(): Settings
}

//composeApp/src/androidMain/kotlin/../util/SecureStorage.android.kt
package br.com.omnidevs.gcsn.util

import com.russhwolf.settings.Settings

actual class SecureStorageProvider(private val context: Context) {
    actual fun getSecureSettings(): Settings {
        try {
        //Lógica ocultada para melhor legibilidade
        } catch (e: Exception) {
            Log.e("SecureStorage", "Failed to create secure
   storage", e)
            throw e
        }
    }
}

//composeApp/src/iosMain/kotlin/../util/SecureStorage.ios.kt
package br.com.omnidevs.gcsn.util

import com.russhwolf.settings.ExperimentalSettingsImplementation
import com.russhwolf.settings.Settings

actual class SecureStorageProvider {
    @OptIn(ExperimentalSettingsImplementation::class)
    actual fun getSecureSettings(): Settings {
        //Lógica ocultada para melhor legibilidade
    }
}

```

Fonte: Autoria própria (2025).

Nestas classes foi utilizado essa abordagem pois não há uma biblioteca que consiga acessar diretamente os recursos de segurança de cada plataforma, para armazenar criptografada as credenciais de acesso, sendo necessário realizar a implementação nativa das funcionalidades de cada plataforma, o que traz uma complexidade maior, necessitando entender como cada plataforma manipula as informações de maneira segura.

4.4.4 Verificação e Validação do Sistema

A etapa de testes, prevista no método de desenvolvimento adotado (ver Figura 4 no Capítulo 3), foi conduzida com o objetivo de verificar se os requisitos funcionais e não-funcionais, especificados na subseção 4.2.1, foram implementados corretamente. Para isso, foi elaborado um plano de testes funcionais, apresentado no Quadro 6, que utiliza os cenários descritos nos casos de uso (subseção 4.2.3) como base para a execução.

Os testes foram realizados em ambientes de emulação para Android (Google Pixel 9 Pro, API 34) e iOS (iPhone 16 Pro, iOS 18), além de um dispositivo físico (Samsung Galaxy S23) para assegurar a conformidade e o desempenho da aplicação em um cenário de uso real.

Quadro 6 – Plano de Testes Funcionais

ID	Requisito	Passos para Execução	Resultado Esperado	Status
CT-01	RF1	Com o usuário autenticado, seguir os passos do caso de uso "Realizar uma postagem"(Quadro 3), utilizando um texto e uma imagem válida.	A aplicação deve exibir uma notificação de sucesso e a nova postagem deve ser visível no perfil do usuário e no feed principal.	Aprovado
CT-02	RF2	Seguir os passos do caso de uso "Reagir uma postagem"(Quadro 4). Adicionalmente, clicar no ícone de "comentar", inserir um texto e enviar.	O ícone de curtir deve mudar de estado visual. O comentário deve ser adicionado à postagem e exibido corretamente na tela de comentários (Figura 15).	Aprovado
CT-03	RF3	Na barra de navegação inferior, clicar em "Descobrir". Rolar a tela para verificar o carregamento de novas postagens.	O sistema deve exibir o feed de descoberta com postagens da categoria "Cosplay", conforme a interface da Figura 10.	Aprovado
CT-04	RNF2	1. Instalar e iniciar a aplicação em um emulador com Android 8.0 (API 26). 2. Executar o caso de teste CT-01.	A aplicação deve ser instalada e executada sem falhas ou *crashes*. A funcionalidade de postagem deve operar conforme esperado.	Aprovado
CT-05	RNF3	Após o login, verificar se as informações de sessão persistem ao fechar e reabrir o aplicativo.	O usuário deve permanecer autenticado no sistema, sem a necessidade de inserir novamente as credenciais.	Aprovado

Fonte: Autoria própria (2025).

Nota sobre a Execução em Emuladores

Para manter o foco deste capítulo nos resultados da verificação e validação do sistema, as instruções técnicas detalhadas para a configuração do ambiente e execução do projeto nos emuladores de Android e iOS estão documentadas no **Apêndice A**.

4.5 Avaliação Heurística da Interface

Além da verificação funcional do sistema, que assegura a correção de suas funcionalidades, considerou-se fundamental realizar uma avaliação da qualidade da interface sob a perspectiva da Interação Humano-Computador (IHC). Em projetos acadêmicos, onde a condução de testes de usabilidade com um grupo representativo de usuários finais é frequentemente inviável, as boas práticas da área recomendam a aplicação de métodos de inspeção.

Desta forma, optou-se pela **Avaliação Heurística** (Nielsen, 1994), um consolidado método de inspeção no qual um avaliador examina a interface e a julga com base em um conjunto de princípios e diretrizes de usabilidade (heurísticas). Este método permite identificar potenciais barreiras de uso de forma rápida e eficiente, gerando subsídios para o aprimoramento do sistema.

Para este trabalho, o próprio autor atuou como avaliador, inspecionando as principais telas da aplicação GCSN em relação às 10 heurísticas de usabilidade propostas por Jakob Nielsen. O objetivo foi identificar pontos fortes e potenciais problemas de design na interface que poderiam impactar a experiência do usuário. Os resultados e as recomendações extraídas desta análise estão consolidados na Tabela 1.

Tabela 1 – Resultados da Avaliação Heurística da Interface GCSN

Heurística	**Análise da Interface GCSN**	**Conformidade**	**Observações e Recomendações**
1. Visibilidade do estado do sistema	O sistema informa o usuário sobre o seu estado. Exemplos: o *snackbar* "Agora você segue"(ver Figura 11) e a indicação de carregamento no botão de login.	Sim	O feedback visual para ações críticas é claro e imediato, o que é um ponto positivo.
2. Correspondência com o mundo real	A aplicação utiliza ícones, termos e fluxos familiares a usuários de outras redes sociais (ex: curtir, comentar, compartilhar, *feed*).	Sim	A linguagem é adequada ao público-alvo e segue convenções de design consolidadas, reduzindo a curva de aprendizado.

(Continua na próxima página)

Tabela 1 – Continuação

Heurística	Análise da Interface GCSN	Conformidade	Observações e Recomendações
3. Controle e liberdade do usuário	O usuário possui controle sobre ações importantes. A caixa de diálogo para confirmar a ação de "Deixar de seguir"(ver Figura 12) é um bom exemplo de saída de emergência.	Parcial	A tela de "Nova Postagem"(ver Figura 17) não possui um botão "Cancelar". **Recomendação:** Adicionar um botão para descartar a postagem, com confirmação para evitar perdas acidentais.
4. Consistência e padrões	A interface demonstra consistência. A barra de navegação inferior é padrão nas telas principais, e o layout dos *cards* de postagem é o mesmo nas telas de Início, Descoberta e Perfil.	Sim	A consistência interna e externa (com padrões de plataforma) facilita a navegação e o uso do sistema.
5. Prevenção de erros	A confirmação para deixar de seguir previne um erro comum. O sistema guia o usuário em fluxos estruturados, como na criação de postagem.	Parcial	Poderiam ser implementados mais mecanismos, como um contador de caracteres ou desabilitar o botão "Publicar"se o campo de texto estiver vazio.
6. Reconhecimento em vez de memorização	Funções principais são mantidas visíveis na barra de navegação inferior. Os ícones são universais, não exigindo que o usuário memorize comandos ou localizações.	Sim	O design promove o reconhecimento, tornando a interface intuitiva para novos usuários.
7. Flexibilidade e eficiência de uso	O sistema oferece aceleradores, como a possibilidade de clicar em uma *hashtag* para iniciar uma busca por aquele termo, agilizando a navegação.	Sim	Esta funcionalidade permite um fluxo mais rápido para descobrir conteúdo relacionado.
8. Estética e design minimalista	As telas são limpas e focadas na tarefa, sem informações irrelevantes. A tela de "Nova Postagem"contém apenas os elementos essenciais para a ação.	Sim	O design não é sobrecarregado, o que ajuda a manter a clareza e a direcionar a atenção do usuário para o conteúdo.
9. Diagnóstico e recuperação de erros	O trabalho não apresenta telas de erro explícitas (ex: falha de login). Apenas o tratamento de exceção para tamanho de arquivo foi mencionado no caso de uso.	Não Avaliado	Esta heurística não pôde ser totalmente avaliada. **Recomendação:** Projetar mensagens de erro claras, como "Usuário ou senha inválidos"em vez de "Erro no login".

(Continua na próxima página)

Tabela 1 – Continuação

Heurística	Análise da Interface GCSN	Conformidade	Observações e Recomendações
10. Ajuda e documentação	Não foi identificada uma seção de ajuda ou FAQ na aplicação.	Não	A ausência é compreensível em um protótipo de TCC. **Recomendação para trabalhos futuros:** Implementar uma seção de ajuda para guiar os usuários.

5 CONCLUSÃO

Este trabalho teve como objetivo o desenvolvimento de uma rede social para dispositivos móveis, para comunidades *cosplay*, que permita a descoberta e compartilhamento de experiencias.

A rede foi implementada utilizando o *framework* Kotlin Multiplataforma para a criação da interface para ambas as plataformas, Android e iOS, e a lógica compartilhada. Mesmo com o estágio beta do *framework* para dispositivos que utilizam o iOS, não houve empecilhos para o desenvolvimento e testagem nesses dispositivos.

Para a comunicação das postagens, o uso da API fornecida pela mantenedora do protocolo, proporcionou uma comunicação mais fluida com a *Atmosphere*, fazendo com que uma futura adoção da comunidade à aplicação seja mais natural.

A validação do sistema foi conduzida em duas frentes complementares. Primeiramente, a **verificação funcional** foi realizada através da execução de um plano de testes sistemático (detalhado na subseção 4.4.4), que confirmou a correta implementação dos requisitos em emuladores (Google Pixel 9 Pro e iPhone 16 Pro) e em um dispositivo físico (Samsung Galaxy S23). Em segundo lugar, para analisar a qualidade da interface, foi conduzida uma **Avaliação Heurística** (detalhada na seção 4.5), que permitiu identificar pontos de conformidade com princípios de usabilidade e oportunidades de melhoria no design da interação.

Durante o desenvolvimento, os principais desafios enfrentados foram voltados ao detalhe que *framework* KMP ainda se encontra em fase beta para iOS, onde recursos necessários na plataforma ainda não possuem bibliotecas de uso compartilhado em código comum, necessitando o uso de implementações específicas por plataforma. Outro detalhe a ser apontado é a utilização de bibliotecas, onde nem todas estão devidamente atualizadas e pode resultar em comportamentos inesperados na utilização.

Com relação ao protocolo de comunicação das postagens, devido à natureza viva das definições utilizadas pelo protocolo (lexicons), mesmo durante a execução deste trabalho, ocorreram divergências no processo de serialização das respostas recebidas pelo servidor, mas foram ajustadas conforme surgiram.

A contribuição realizada por esse projeto visa incorporação a uma rede social aberta para comunidade específica, com funções voltadas para a comunidade, mas mantendo a usabilidade geral de rede social, aprimorando os conhecimentos em desenvolvimento para dispositivos móveis utilizando a linguagem Kotlin e introduzindo conhecimentos para desenvolvimento para diversas plataformas.

Como oportunidades para trabalhos futuros, e com base nos resultados da avaliação do sistema, sugere-se:

- **Aprimoramento da Interface com Base na Avaliação Heurística:** Implementar as melhorias de usabilidade identificadas na seção 4.5, como a inclusão de um botão

"Cancelar"na tela de nova postagem, o aprimoramento das mensagens de erro para o usuário e a adição de um contador de caracteres.

- **Implementação de Sistema de Notificações:** Desenvolver o sistema de notificações *push* no dispositivo, bem como uma tela dedicada na aplicação para visualizá-las.
- **Implementação da Funcionalidade de Chat:** Integrar a funcionalidade de chat entre usuários, recurso presente em versões atuais do protocolo.
- **Criação de Instâncias Particulares:** Explorar a criação de instâncias particulares do protocolo, o que permitiria o desenvolvimento de ferramentas de moderação e funcionalidades exclusivas para a aplicação GCSN.
- **Desenvolvimento de uma Seção de Ajuda:** Projetar e implementar uma seção de "Ajuda e Documentação"dentro do aplicativo para guiar os usuários, conforme recomendado pela análise da décima heurística.

Com base no estudo realizado para a execução deste trabalho, pode-se concluir que a aplicação foi desenvolvida de maneira funcional, ofertando uma opção para que as comunidades de *cosplayers* tenham um local para compartilhar suas experiencias.

REFERÊNCIAS

ALICE, A. F. A. **Linguagem Kotlin: o que é e um Guia para aprender | Alura — alura.com.br**. 2023. https://www.alura.com.br/artigos/kotlin. Acesso em: 09 dec. 2024.

BLUESKY SOCIAL PBC. **AT Protocol — atproto.com**. 2022. https://atproto.com/. [Accessed 20-02-2025].

BLUESKY SOCIAL PBC. **Glossary of terms - AT Protocol — atproto.com**. 2025. https://atproto.com/pt/guides/glossary#atmosphere. [Accessed 02-06-2025].

BLUESKY SOCIAL PBC. **Rate Limits | Bluesky — docs.bsky.app**. 2025. https://docs.bsky.app/docs/advanced-guides/rate-limits. [Accessed 02-06-2025].

BRITO, R. **Android Com Android Studio - Passo A Passo**. Ciência Moderna, 2017. ISBN 9788539907632. Disponível em: https://books.google.com.br/books?id=eDLFswEACAAJ.

CALIXTO, F. **O que é o AT Protocol? — canaltech.com.br**. 2023. https://canaltech.com.br/redes-sociais/o-que-e-o-at-protocol/. Acesso em: 09 dec. 2024.

COELHO Ândriu F. **O que é iOS: o sistema operacional dos dispositivos móveis da Apple | Alura — alura.com.br**. 2024. https://www.alura.com.br/artigos/o-que-e-ios. Acesso em: 09 dec. 2024.

CORTES, A. **Cosplayers: como ganhar dinheiro em plataformas de conteúdo**. 2023. Disponível em: https://www.remessaonline.com.br/blog/cosplayers-como-ganhar-dinheiro-em-plataformas-de-conteudo/. Acesso em: 22 oct. 2024.

COSPLAY BRASIL. **Sobre Nós**. 2024. Disponível em: https://cosplaybrasil.com.br/home/index.php/sobre-nos. Acesso em: 22 oct. 2024.

DANTAS, G. C. da S. **Cosplay - Brasil Escola**. 2016. Published at Brasil Escola. Disponível em: https://brasilescola.uol.com.br/artes/cosplay.htm. Acesso em: 22 oct. 2024.

GOOGLE. **Android Studio**. 2024. Software Development Kit. Disponível em: https://developer.android.com/studio.

INCOSPLAY GLOBAL INC. **Incosplay: rede de cosplay**. 2023. Version 1.1.0. Disponível em: https://play.google.com/store/apps/details?id=top.incosplay.app&hl=pt_BR. Acesso em: 22 oct. 2024.

ISO/IEC/IEEE. **Systems and software engineering – Life cycle processes – Requirements engineering**. Geneva, CH, 2018.

JetBrains. **Kotlin Multiplatform - Use cases and benefits**. 2024. https://www.jetbrains.com/kotlin-multiplatform/. Acessado em: 03 jun. 2025.

KLEPPMANN, M. *et al.* Bluesky and the at protocol: Usable decentralized social media. *In*: **Proceedings of the ACM Conext-2024 Workshop on the Decentralization of the Internet**. ACM, 2024. (CoNEXT '24), p. 1–7. Disponível em: http://dx.doi.org/10.1145/3694809.3700740.

Kotlin Foundation. **Kotlin Multiplatform Mobile for Cross-Platform Developers**. 2023. https://lp.jetbrains.com/kmm-for-crossplatform-developers/. Acessado em: 20 fev. 2025.

Kotlin Foundation. **Share code between platforms - Understand Kotlin Multiplatform project structure**. 2024. https://kotlinlang.org/docs/multiplatform-mobile-understand-project-structure.html. Acessado em: 03 jun. 2025.

LIMA, L. **O que é Android? Entenda a diferença para o iOS, do iPhone • Tecnoblog — tecnoblog.net**. 2023. https://tecnoblog.net/responde/o-que-e-o-android-entenda-a-diferenca-para-o-ios-do-iphone/. Acesso em: 06 dec. 2024.

MLABS. **Redes Sociais mais usadas no Brasil e no mundo em 2024**. 2024. MLabs. Disponível em: https://www.mlabs.com.br/blog/redes-sociais-mais-usadas. Acesso em: 21 oct. 2024.

NIELSEN, J. Heuristic evaluation. *In*: **Usability inspection methods**. [*S.l.*]: John Wiley & Sons, Inc., 1994. p. 25–62.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: Uma Abordagem Profissional**. 8ª. ed. Porto Alegre: AMGH Editora Ltda., 2016.

PRIBERAM INFORMÁTICA, S.A. **cosplay — dicionario.priberam.org**. 2024. https://dicionario.priberam.org/cosplay. Acesso em: 06 dec. 2024.

RODRIGUES, J. **Redes sociais: saiba o que são e para que servem? — rdstation.com**. 2024. https://www.rdstation.com/blog/marketing/redes-sociais/. Acesso em: 06 dec. 2024.

ROYCE, W. W. Managing the development of large software systems: concepts and techniques. *In*: **Proceedings of the 9th International Conference on Software Engineering**. Washington, DC, USA: IEEE Computer Society Press, 1987. (ICSE '87), p. 328–338. ISBN 0897912160.

SOMMERVILLE, I. **Engenharia de Software**. 10ª. ed. São Paulo: Pearson Education do Brasil, 2019.

STATCOUNTER GLOBAL STATS. **Mobile Operating System Market Share Brazil | Statcounter Global Stats**. 2024. StatCounter Global Stats. Disponível em: https://gs.statcounter.com/os-market-share/mobile/brazil. Acesso em: 22 oct. 2024.

TARASOV, V. **Jetpack Compose: A New Approach to Android UI Development — vptarasov**. 2024. https://medium.com/@vptarasov/jetpack-compose-a-new-approach-to-android-ui-development-4ca03d7ee261. Acesso em: 09 dec. 2024.

WEINREICH, A. **Six Degrees - Landing Page — SixDegrees.com**. 1997. SixDegrees.com. Acesso em: 06 dec. 2024.

WELLS, H. G. London Film Productions, 1936. Disponível em: https://www.imdb.com/title/tt0028358/. Acesso em: 06 dec. 2024.

APÊNDICE A – Instruções para Execução do Projeto em Emuladores

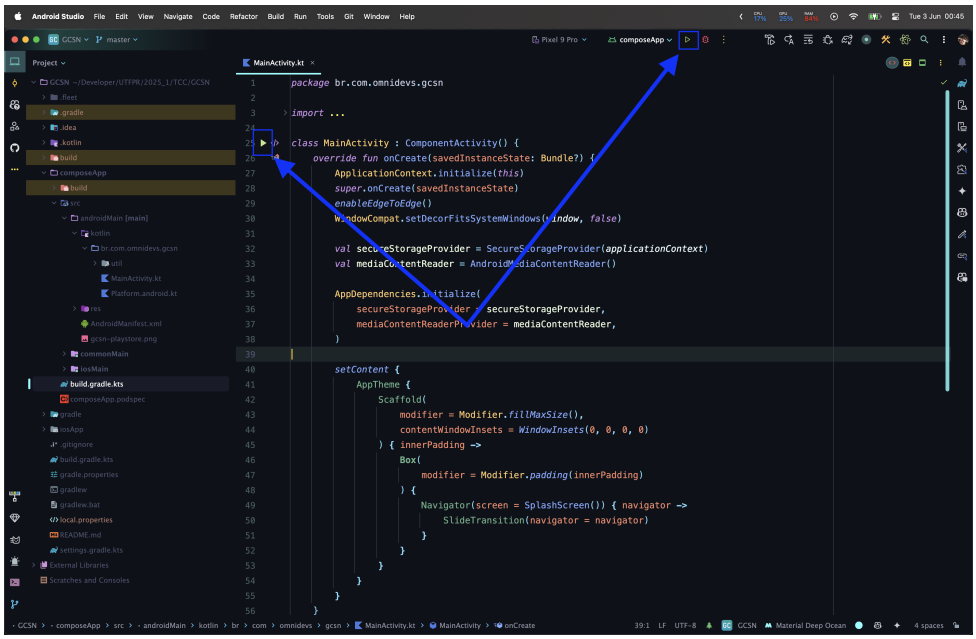
Este apêndice detalha os procedimentos técnicos necessários para compilar e executar a aplicação GCSN nos ambientes de emulação das plataformas Android e iOS, a partir do código-fonte do projeto.

A.1 Execução em Ambiente Android

O processo de iniciar a emulação no Android é simples. No Android Studio, basta abrir a classe ***MainActivity***, localizada dentro do diretório e pacote **androidMain**.

Ao lado da declaração da classe `MainActivity`, haverá um botão de execução (semelhante a um "play"), que compilará e iniciará a aplicação no emulador Android previamente configurado no IDE. A Figura 21 ilustra a localização do botão de execução na interface do Android Studio.

Figura 21 – Android Studio - Iniciar emulação



Fonte: Autoria própria (2025).

A.2 Execução em Ambiente iOS

O processo para emulação no iOS, a partir de um projeto KMP, depende da execução de uma sequência de procedimentos no terminal e na IDE **XCode**:

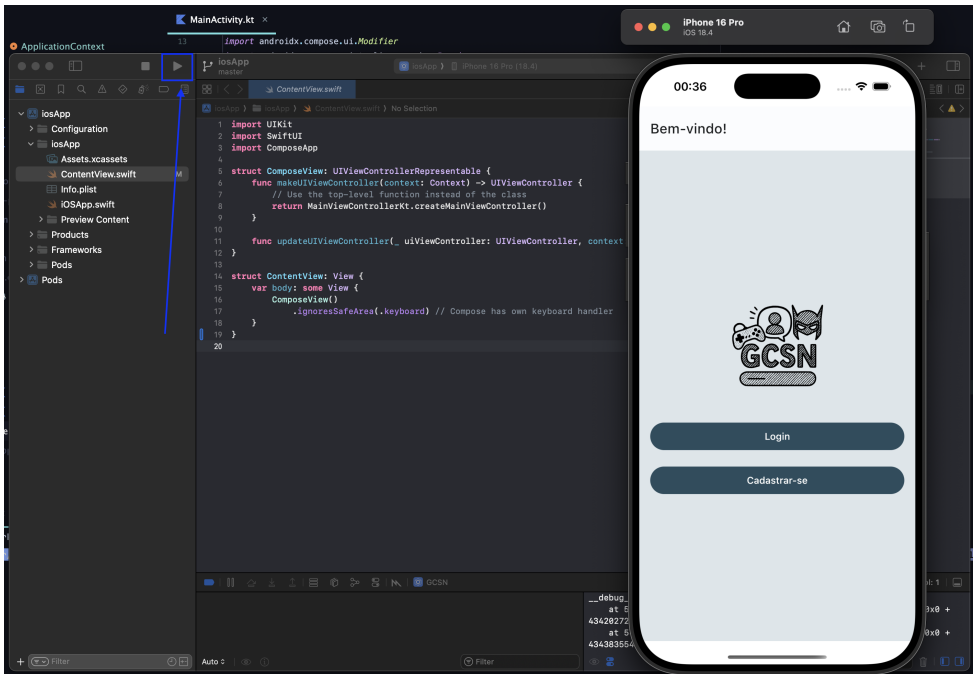
- Realizar o *download* e a instalação da IDE **XCode**, disponível na AppStore do macOS.
- Instalar o gerenciador de dependências **CocoaPods**[1], o que pode requerer uma versão atualizada do interpretador Ruby no sistema.

[1] Cocoapods é um gerenciador de dependências para projetos Cocoa (iOS e macOS) desenvolvidos em Swift e Objective-C.

- Verificar se no arquivo de *build* do projeto (**build.gradle.kts**) existe a declaração do *plugin* do CocoaPods, apontando para o arquivo **Podfile** localizado no diretório **iosApp**. Este arquivo contém as instruções para a compilação da aplicação no simulador de iPhone.
- No terminal integrado do Android Studio, na raiz do projeto, executar o comando `./gradlew podPublishDebug`. Este processo pode levar alguns minutos.
- Após a conclusão do comando anterior, executar a sequência de comandos `cd iosApp && pod install`. Este comando acessará o diretório do projeto iOS e instalará as dependências (*pods*).
- O último comando a ser executado no terminal é `open iosApp.xcworkspace`, que abrirá o projeto iOS já configurado no **XCode**.

Com o projeto aberto no **XCode**, basta clicar no botão de *build* e execução para que a aplicação seja compilada e iniciada no simulador de iOS. A Figura 22 demonstra a interface do XCode e o botão de execução.

Figura 22 – XCode - Iniciar emulação



Fonte: Autoria própria (2025).