

Prism

Sistema de recomendação de filmes

André Vítor Barbosa Schenato – 103586@aluno.uricer.edu.br¹

Henri A. Grzegozeski – 104114@aluno.uricer.edu.br²

Levi da Rosa Gomes – 103495@aluno.uricer.edu.br³

Ruan Dalla Rosa – 103423@aluno.uricer.edu.br⁴

¹Universidade Regional Integrada do Alto Uruguai e das Missões – Campus Erechim
Erechim – RS – Brasil

Resumo. O presente trabalho descreve o desenvolvimento do *Prism*, uma aplicação mobile voltada à recomendação personalizada de filmes, projetada para facilitar a descoberta de conteúdo em plataformas de streaming. A solução utiliza o framework **Flutter** para a construção da interface e adota o **Firebase** como infraestrutura de back-end, integrando serviços como Authentication e Cloud Firestore. Para ampliar a qualidade das sugestões, o sistema consome metadados da **TMDb API** e emprega a LLM **Gemini Flash 2.5**, acessada por meio do **OpenRouter**, responsável por interpretar preferências do usuário e gerar recomendações com técnicas de linguagem natural. As telas foram projetadas no **Figma**, garantindo uma navegação clara e consistente entre dispositivos. O resultado é uma aplicação funcional, segura e escalável, que demonstra a viabilidade da integração entre IA generativa e tecnologias serverless na construção de sistemas modernos de recomendação.

Palavras-chave: Recomendação de filmes; Inteligência Artificial; Flutter; Firebase;

Link do Repositório: <https://github.com/andreschenato/prism>

1. Introdução

Este artigo descreve o desenvolvimento do *Prism*, uma aplicação mobile projetada para otimizar o processo de recomendação personalizada de filmes para usuários. O sistema visa mitigar os desafios inerentes à vasta quantidade de conteúdo disponível em plataformas de *streaming*, proporcionando uma experiência de descoberta de conteúdo audiovisual mais eficiente e alinhada às preferências individuais.

A consolidação das plataformas de *streaming* transformou o consumo de conteúdo audiovisual, disponibilizando vastos catálogos de filmes e séries. Contudo, essa abundância gerou um desafio para os usuários: a dificuldade em encontrar conteúdo relevante e personalizado em meio a inúmeras opções, levando à sobrecarga de escolha. Sistemas de recomendação atuais, muitas vezes, não conseguem atender plenamente à demanda por sugestões altamente alinhadas às preferências individuais. O *Prism* justifica-se pela necessidade de um sistema que refine a experiência de descoberta, oferecendo recomendações de filmes personalizadas e eficientes.

A arquitetura do *Prism* integra tecnologias modernas para garantir escalabilidade, desempenho e uma experiência de usuário rica. A interface da aplicação é construída

utilizando o *framework Flutter*, permitindo o desenvolvimento *multiplataforma* e uma interface de usuário responsiva. O *backend* é implementado sobre a infraestrutura *Firebase*, compreendendo serviços como *Firebase Authentication* para gerenciamento de usuários e *Cloud Firestore* para persistência de dados.

Para aprimorar a capacidade de recomendação, o *Prism* integra-se a serviços externos especializados. A *TMDb API* é responsável pelo fornecimento de metadados detalhados sobre filmes e séries, enquanto a *API OpenRouter* disponibiliza modelos de linguagem, como *Gemini* e *Claude*, utilizados para processamento de linguagem natural. Essa combinação permite interpretar *prompts* e preferências do usuário, resultando em recomendações mais precisas e personalizadas.

A interface do aplicativo organiza essas sugestões em seções como “Favoritos”, “Perfeitos para Você” e “Você Também Pode Gostar”, facilitando a navegação por diferentes níveis de relevância. Os protótipos de interface foram desenvolvidos no *Figma*, seguindo diretrizes de boas práticas em *User Experience (UX)*.

A crescente capacidade dos Grandes Modelos de Linguagem (*LLMs*) de processar e interpretar linguagem natural é um fator determinante para a relevância do *Prism*. Ao contrário dos algoritmos de recomendação baseados unicamente em filtragem colaborativa ou de conteúdo, a integração de *LLMs* permite ao sistema interpretar *prompts* de contexto e requisições complexas do usuário. Essa capacidade habilita a geração de recomendações mais sofisticadas, contextuais e adaptadas dinamicamente, transformando a interação do usuário com o sistema em um processo mais intuitivo e preciso de descoberta de conteúdo.

2. Referencial Teórico

O referencial teórico tem como finalidade apresentar os conceitos e fundamentos que sustentam o desenvolvimento deste projeto. Essa seção busca contextualizar as ferramentas e metodologias utilizadas, evidenciando sua importância para a construção de soluções eficientes, colaborativas e alinhadas às demandas atuais da área de tecnologia. Por meio desse embasamento, garante-se não apenas a fundamentação técnica, mas também a justificativa da escolha dos recursos aplicados ao trabalho.

2.1. Flutter

O *Flutter* é um *framework* de código aberto desenvolvido pelo *Google* para a criação de aplicações *multiplataforma* a partir de uma única base de código. Com ele é possível desenvolver para *iOS*, *Android*, *Web*, *Windows*, *macOS* e *Linux*, garantindo consistência visual e desempenho nativo. Seu diferencial está no uso da linguagem *Dart* e na arquitetura baseada em *widgets*, que permitem a construção de interfaces altamente personalizáveis (*Flutter*, 2025).

Entre suas funcionalidades, destacam-se o *Hot Reload*, que possibilita visualizar alterações no código em tempo real sem reiniciar a aplicação, acelerando o ciclo de desenvolvimento, além de bibliotecas ricas de componentes e suporte integrado para design responsivo. Essas características tornam o *Flutter* uma solução moderna para desenvolvimento ágil e *multiplataforma* (*Flutter*, 2025).

A escolha do *Flutter* para este projeto se justifica pela produtividade gerada pela reutilização de código em múltiplas plataformas, pelo alto desempenho proporcionado

pelo motor gráfico próprio e pela facilidade de colaboração em equipes que precisam entregar soluções visuais consistentes e escaláveis. Assim, o *framework* contribui para reduzir custos, agilizar prazos e assegurar qualidade no desenvolvimento (*Flutter*, 2025).

2.2. Firebase

O *Firebase* é uma plataforma de desenvolvimento de aplicativos móveis e *web* oferecida pelo *Google*, que fornece uma variedade de ferramentas para facilitar a criação, o gerenciamento e a escalabilidade de aplicativos (*Google*, 2025). Entre seus principais serviços, destacam-se o *Firebase Authentication*, o *Cloud Firestore* e as *Cloud Functions*, que, quando integrados, permitem o desenvolvimento de soluções robustas, seguras e eficientes.

2.2.1. Firebase Authentication

Segundo o *Firebase* (2025), o *Firebase Authentication* permite autenticar usuários de forma segura e simplificada, oferecendo suporte a métodos de login como e-mail e senha, autenticação por número de telefone e provedores de identidade federada, como *Google*, *Facebook*, *GitHub* e *Apple*. Além disso, disponibiliza recursos de autenticação anônima e multifatorial, garantindo maior segurança no acesso aos aplicativos. Dessa forma, o serviço simplifica o gerenciamento de credenciais e validação de usuários, proporcionando uma experiência confiável para os usuários (*Google*, 2025).

2.2.2. Cloud Firestore

O *Cloud Firestore* é um banco de dados *NoSQL* flexível e escalável, projetado para armazenar e sincronizar dados entre clientes e servidores em tempo real. Ele organiza os dados em documentos e coleções, permitindo consultas complexas, filtragem, ordenação e suporte a transações. O *Firestore* oferece sincronização automática entre dispositivos, operação *offline* e atualizações em tempo real, garantindo que os usuários tenham acesso imediato às alterações de dados sem necessidade de atualização manual da aplicação (*Google*, 2025).

2.3. APIs Externas

A integração com *APIs* externas é um componente essencial no desenvolvimento de aplicações modernas, pois amplia as funcionalidades disponíveis e possibilita o acesso a dados e serviços especializados. No contexto deste projeto, a utilização da *TMDb API* e da *API OpenRouter*, que fornece acesso a modelos de Inteligência Artificial, desempenha um papel fundamental na geração de recomendações personalizadas e na melhoria da experiência do usuário.

2.3.1. TMDb API

A *TMDb API* oferece acesso estruturado a um amplo catálogo de metadados sobre filmes, séries, artistas e outros elementos do universo audiovisual. De acordo com sua

documentação oficial, a API disponibiliza dados em formato *JSON* por meio de endpoints REST, permitindo consultas diretas a informações como sinopses, avaliações, gêneros, imagens promocionais e tendências globais. Essa abordagem simplificada facilita a integração e reduz a necessidade de manipulação complexa de parâmetros, tornando o consumo das informações mais ágil e eficiente.

A API também permite combinar diferentes tipos de dados em uma única requisição — como detalhes de um título, elenco, trailers e recomendações relacionadas — o que contribui para otimizar o desempenho da aplicação e reduzir a quantidade de chamadas externas. Dessa forma, a *TMDb API* viabiliza a construção de experiências interativas e atualizadas, fundamentais para sistemas de recomendação baseados em preferências do usuário.

As principais funcionalidades da *API* incluem:

- Consulta de títulos e nomes: permite obter informações detalhadas sobre filmes, séries, episódios e profissionais do entretenimento, incluindo sinopses, classificações, elenco e dados complementares;
- Pesquisa avançada: possibilita buscar conteúdos com base em termos específicos, facilitando a localização de informações relevantes;
- Dados de bilheteira: disponibiliza informações financeiras sobre filmes, como receitas de bilheteira em diferentes períodos e regiões;
- Rankings e métricas: oferece acesso a indicadores como *STARmeter* e *TITLEmeter*, que refletem a popularidade de profissionais e títulos.

Segundo a *TMDb* (2025), a integração da *API* em projetos permite enriquecer aplicações com dados confiáveis e atualizados do universo do entretenimento, proporcionando aos usuários experiências mais completas e interativas. A utilização da *API* facilita a manutenção e atualização das informações, uma vez que os dados são obtidos diretamente da fonte oficial em tempo real (*TMDb*, 2025).

2.3.2. APIs de Inteligência Artificial (IA)

A Inteligência Artificial (IA) constitui um dos pilares centrais do sistema *Prism*, sendo responsável pelo processamento de linguagem natural e pela geração de recomendações personalizadas. Para esse propósito, o projeto utiliza a *API OpenRouter*, uma plataforma que funciona como um gateway para múltiplos modelos de IA generativa, incluindo *Gemini*, *Claude*, *Llama* e outros modelos de última geração. O OpenRouter permite selecionar diferentes provedores de modelos, oferecendo flexibilidade, custo-benefício e diversidade de capacidades linguísticas.

Esses modelos pertencem à categoria de *Large Language Models* (LLMs), capazes de interpretar texto, compreender preferências e gerar análises contextualizadas com base em grandes volumes de dados. Sua arquitetura se fundamenta em redes neurais profundas treinadas em conjuntos extensivos de dados multimodais, o que possibilita identificar intenções, estilos narrativos, temas recorrentes e outras nuances relevantes às preferências do usuário.

No *Prism*, a integração com o *OpenRouter* é realizada por meio de requisições *HTTP* seguras enviadas pelo *backend* no próprio cliente *mobile*, que repassa *prompts*

estruturados e recebe como retorno recomendações interpretadas pelo modelo escolhido. Essa abordagem permite analisar o histórico de interação do usuário, interpretar descrições subjetivas sobre preferências e produzir sugestões alinhadas ao perfil individual. A IA passa, assim, a atuar como um mecanismo dinâmico de recomendação contextual, complementando os dados fornecidos pela *TMDb API* com uma camada semântica avançada.

Entre as principais vantagens do uso da *API OpenRouter*, destacam-se:

- **Flexibilidade de Modelos:** possibilidade de utilizar diferentes LLMs, escolhendo aquele mais adequado em termos de custo, desempenho ou estilo de resposta;
- **Compreensão de Linguagem Natural:** interpretação de descrições subjetivas, como “filmes com antagonistas complexos” ou “obras com atmosfera semelhante a *Blade Runner*”;
- **Recomendações Personalizadas:** geração de listas de filmes alinhadas ao perfil do usuário, combinando metadados da *TMDb API* com análise semântica das preferências;
- **Integração Simples e Serverless:** comunicação direta via *Cloud Functions*, sem necessidade de servidores dedicados;
- **Evolução Contínua:** melhoria na qualidade das respostas à medida que os modelos disponibilizados pelo OpenRouter são atualizados.

A adoção do *OpenRouter* no projeto justifica-se por sua capacidade de oferecer múltiplos modelos de IA generativa em uma única interface, garantindo flexibilidade, escalabilidade e facilidade de manutenção. Essa integração amplia o potencial do *Prism*, permitindo que o sistema vá além da exibição de metadados e interprete preferências subjetivas com precisão, tornando a experiência de descoberta de conteúdo mais fluida, personalizada e contextualizada.

2.4. Git e GitHub

O *GitHub* é uma plataforma em nuvem que permite armazenar, compartilhar e colaborar no desenvolvimento de códigos em repositórios. Seu diferencial está no trabalho colaborativo, viabilizado pelo sistema de versionamento *Git*, que garante o controle das alterações e a integração segura de modificações no código (*Docs*, 2025).

O *Git* é um sistema de controle de versão que rastreia mudanças em arquivos e facilita o trabalho simultâneo de vários desenvolvedores. Ele possibilita criar ramificações independentes, editar de forma segura e mesclar atualizações à versão principal do projeto, assegurando consistência e organização.

A integração entre *Git* e *GitHub* proporciona não apenas o versionamento do código, mas também um ambiente colaborativo para revisão, integração e sincronização de alterações. Essa combinação é essencial em projetos que demandam rastreabilidade, segurança e produtividade, justificando seu uso neste trabalho.

2.5. Figma

O *Figma* é uma ferramenta de design de interface gráfica (*UI/UX*), colaborativa e baseada na *web*, que permite criar, compartilhar, prototipar e construir interfaces visuais com outros usuários em tempo real. Seu funcionamento é facilitado por recursos de edição

simultânea, armazenamento automático na nuvem e controle de versões integrado, assegurando que todos vejam sempre a versão mais atual do design (*Figma*, 2025).

Além disso, o *Figma* reúne diversas funcionalidades em um único ambiente: criação de designs *UI* (*Figma Design*), prototipagem interativa (*Figma Prototipação*), estruturação de sistemas de design (*Figma para Design Systems*), e ainda outras ferramentas colaterais como *FigJam* (quadro branco colaborativo), *Figma Slides* e *Figma Make* (para gerar protótipos com IA).

A adoção do *Figma* no projeto justifica-se pela sua capacidade de centralizar processos de design e promover colaboração eficaz. Ele oferece suporte a *workflows* integrados — desde o desenho inicial até a prototipagem e entrega ao desenvolvedor — com *feedback* imediato e histórico de alterações. Esse alinhamento entre designers, desenvolvedores e *stakeholders* acelera a tomada de decisão, aumenta a consistência visual via bibliotecas compartilhadas e reduz retrabalhos, sendo especialmente valioso em projetos que envolvem interfaces interativas e equipes distribuídas (*Figma*, 2025).

3. Metodologia

O desenvolvimento deste projeto seguiu uma sequência estruturada de etapas, garantindo planejamento adequado e organização durante todo o processo. Inicialmente, foi definido o tema do projeto, estabelecendo o objetivo principal e as necessidades que seriam atendidas pela solução a ser desenvolvida.

Em seguida, realizou-se a seleção das tecnologias mais adequadas para implementação, levando em consideração critérios como escalabilidade, compatibilidade, facilidade de uso e integração entre ferramentas. As tecnologias escolhidas incluíram ferramentas de versionamento e colaboração, design de interfaces, desenvolvimento *multiplataforma* e gestão de *back-end*.

Com as tecnologias definidas, iniciou-se a concepção do protótipo de interface no *Figma*, permitindo estruturar visualmente as telas e funcionalidades do sistema. Essa etapa foi essencial para mapear a experiência do usuário e ajustar fluxos de navegação antes da implementação do código.

Paralelamente, todo o ambiente de desenvolvimento foi configurado e anexado ao *GitHub*, garantindo controle de versão, colaboração eficiente e registro das alterações realizadas durante o desenvolvimento. Com o ambiente pronto, a equipe passou a programar o sistema, dividindo as tarefas entre os integrantes de acordo com suas especialidades e responsabilidades. Essa divisão permitiu trabalhar em paralelo nas diferentes partes do projeto, assegurando que as funcionalidades fossem desenvolvidas, testadas e integradas de forma organizada e eficiente.

3.1. Tecnologias e Ferramentas

O sistema foi desenvolvido com uma *stack* moderna e coerente com os requisitos de escalabilidade e automação. Destacam-se:

- **Flutter:** linguagem e *framework* para desenvolvimento de aplicações *mobile multiplataforma*, permitindo criar interfaces responsivas e performáticas;
- **Firebase:** plataforma de *backend* como serviço (*BaaS*), utilizada para autenticação de usuários (*Firebase Authentication*) e banco de dados em tempo real *NoSQL* (*Cloud Firestore*);

- **TMDb API:** API externa utilizada para obtenção de metadados sobre filmes e séries, enriquecendo as recomendações com informações detalhadas;
- **APIs de IA (Gemini, Claude):** utilizadas para processamento de linguagem natural, permitindo interpretar *prompts* de contexto e solicitações complexas dos usuários;
- **Figma:** ferramenta de design colaborativo utilizada para criar protótipos de interface, facilitando a visualização e iteração sobre o design da aplicação;
- **GitHub:** plataforma de hospedagem de código-fonte e controle de versão, utilizada para gerenciar o desenvolvimento colaborativo do projeto.

4. Desenvolvimento

A seguir, são apresentados os aspectos técnicos da implementação do sistema, abrangendo sua arquitetura, estrutura interna, funcionalidades desenvolvidas e decisões adotadas ao longo do processo de construção.

4.1. Arquitetura do Sistema

O *Prism* adota uma arquitetura modular estruturada em camadas, com divisão clara entre apresentação, processamento e integração externa. Essa abordagem favorece a escalabilidade, o isolamento de responsabilidades e a manutenção contínua do projeto.

A aplicação é organizada da seguinte forma:

- **Camada de Apresentação (*Frontend*)** — implementada em *Flutter*, responsável pela interface visual, navegação e interação direta com o usuário. Essa camada se comunica com os serviços de autenticação e banco de dados do *Firebase* para acessar perfis, preferências e listas personalizadas;
- **Camada de Dados e Autenticação** — composta pelo *Firebase Authentication* e pelo *Cloud Firestore*. Essa camada gerencia informações como perfis de usuário, preferências, histórico e favoritos, atuando como a base de persistência da aplicação;
- **Camada de Processamento (*Backend*)** — formada pelo mecanismo interno de recomendação, o *Recommender Engine*, responsável por interpretar os dados armazenados, consolidar preferências e organizar o fluxo de consulta aos serviços externos;
- **Camada de Integração** — responsável pela comunicação com a *TMDb API*, utilizada para coleta de metadados de filmes e séries, e com a *OpenRouter API*, que fornece modelos de IA capazes de interpretar preferências em linguagem natural e gerar sugestões personalizadas.

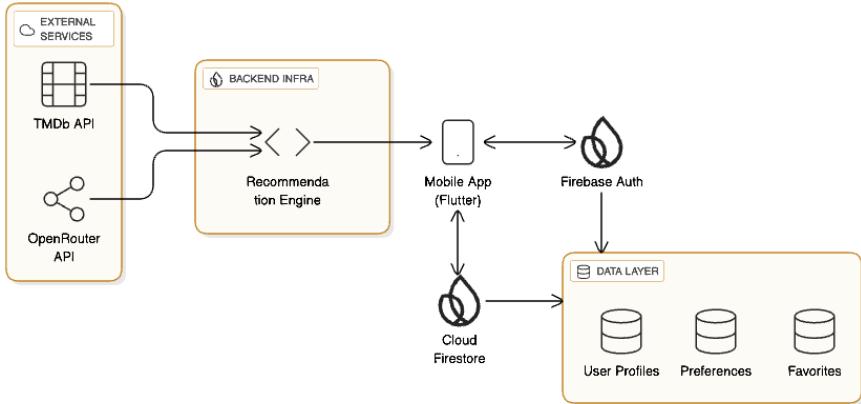


Figure 1. Arquitetura do Sistema *Prism*

Essa arquitetura permite um fluxo contínuo entre cliente, serviços de dados e mecanismos de recomendação. O aplicativo móvel interage com o *Firebase* e com o *Recommender Engine*, enquanto a camada de integração fornece respostas enriquecidas obtidas a partir das *APIs* externas. Dessa forma, o sistema combina o ecossistema *Firebase* com fontes externas especializadas, permitindo recomendações contextualizadas, baixo acoplamento entre módulos e expansão futura sem comprometer o desempenho.

4.2. Estrutura da Aplicação

A estrutura interna do projeto segue uma organização modular, típica de aplicações *Flutter* de médio porte, separando de forma explícita as responsabilidades de configuração, navegação, serviços, componentes visuais e funcionalidades. Essa divisão reduz acoplamento, facilita testes e permite evolução incremental dos módulos.

A pasta `core/` concentra os elementos centrais da aplicação: o cliente HTTP usado para comunicação com as *APIs* externas (`api/api_client.dart`), as configurações de inicialização do *Firebase*, o gerenciador de dependências via *service locator*, o roteamento global e o tema visual. Também abriga o conjunto de *widgets* reutilizáveis, que padronizam a interface e reduzem duplicações.

As funcionalidades de domínio são agrupadas em `features/`, cada uma contendo suas camadas de *data*, *domain* e *presentation*, seguindo um modelo inspirado em *clean architecture*. Entre os módulos mais relevantes destacam-se:

- **Auth** — responsável por autenticação, integrando o *Firebase Authentication* via fontes de dados e repositórios dedicados;
- **Complete Profile** — gerencia preferências iniciais do usuário, como gêneros, idiomas e país, utilizando fontes do *Firebase*;
- **Media List e Details** — realizam consultas à *TMDb API*, tratam modelos de mídia e expõem as informações para as telas de listagem e detalhes;
- **Recommendations** — módulo que interage com a *OpenRouter API* e implementa a lógica do mecanismo de recomendação;
- **Settings** — responsável pelas telas de configurações e informações da conta.

Arquivos como `main.dart`, `app.dart` e `firebase_options.dart` compõem o ponto de entrada da aplicação, inicializando provedores, configurando o ambiente do *Firebase* e montando a estrutura básica do aplicativo.

4.3. *Frontend* e Experiência do Usuário

A interface do usuário foi implementada como uma aplicação móvel *multiplataforma*, desenvolvida com o *framework Flutter* e a linguagem *Dart*. Essa escolha permitiu que o sistema fosse executado tanto em dispositivos *Android* quanto *iOS*, mantendo uma experiência consistente.

O design das telas foi prototipado no *Figma*, seguindo boas práticas de *UX/UI* e aplicando o conceito de interface intuitiva e responsiva. O uso do *Hot Reload* do *Flutter* acelerou o desenvolvimento e os testes, permitindo ajustes visuais em tempo real.

As principais telas do sistema *Prism* estão apresentadas a seguir.

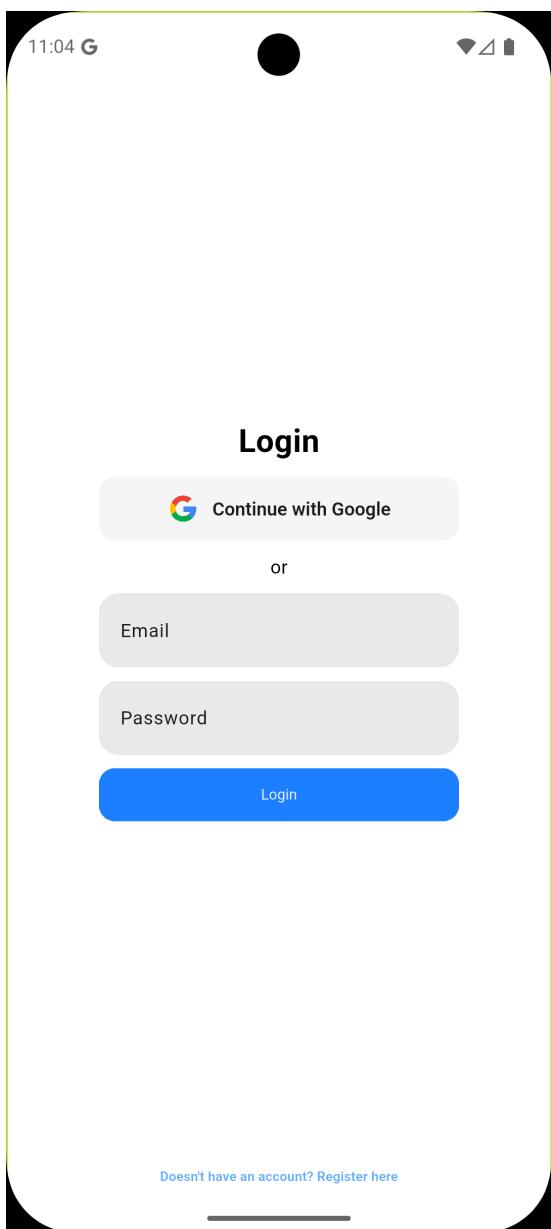


Figure 2. *
(a) Tela de Login e Cadastro

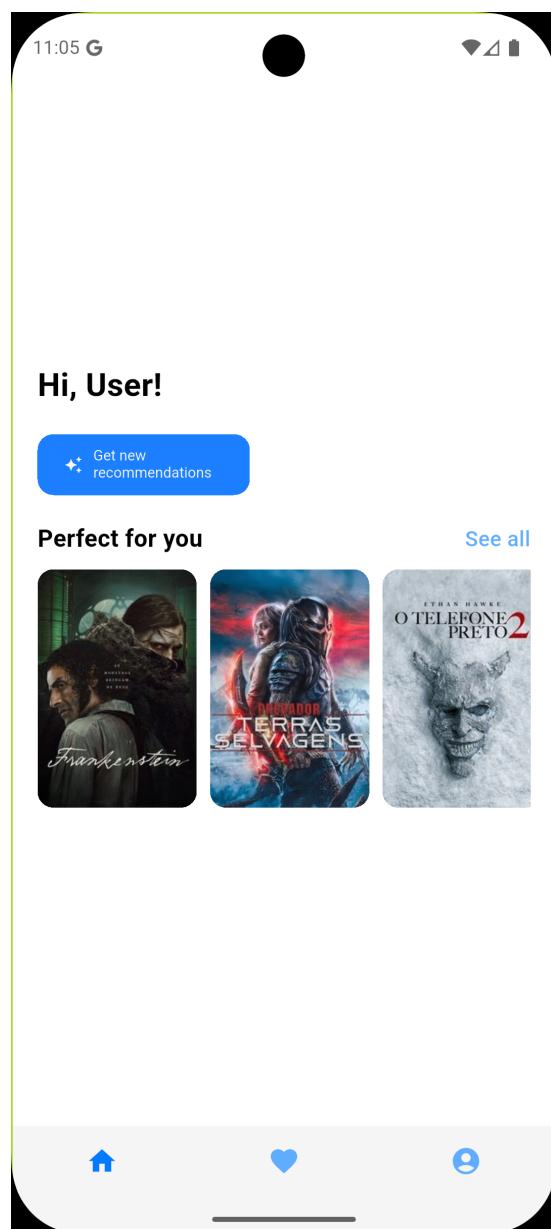


Figure 3. *
(b) Tela Inicial

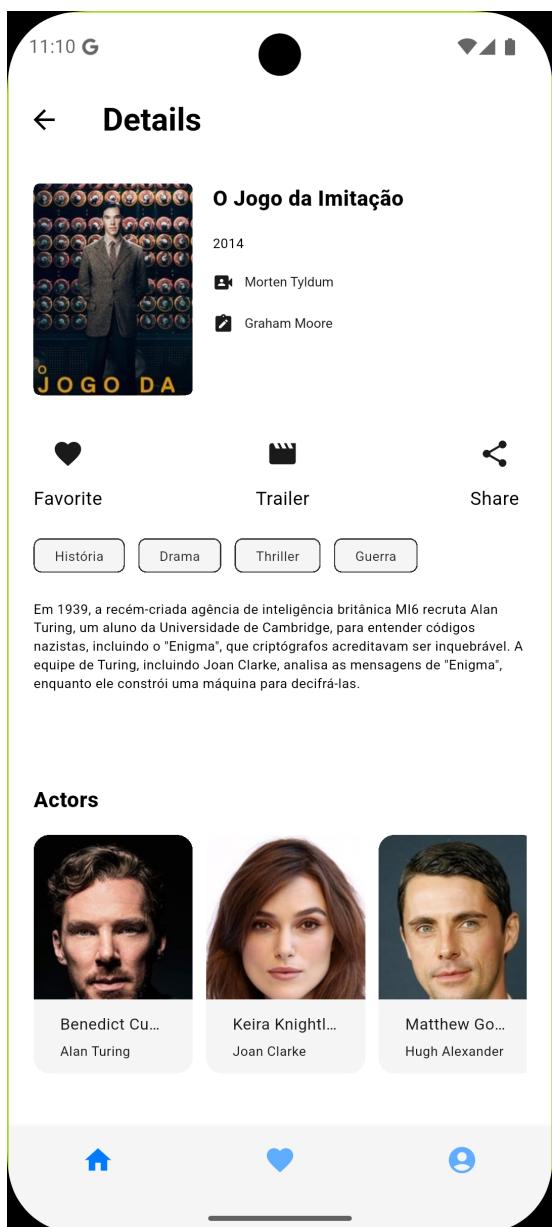


Figure 4. *
(a) Tela de Detalhes do Filme

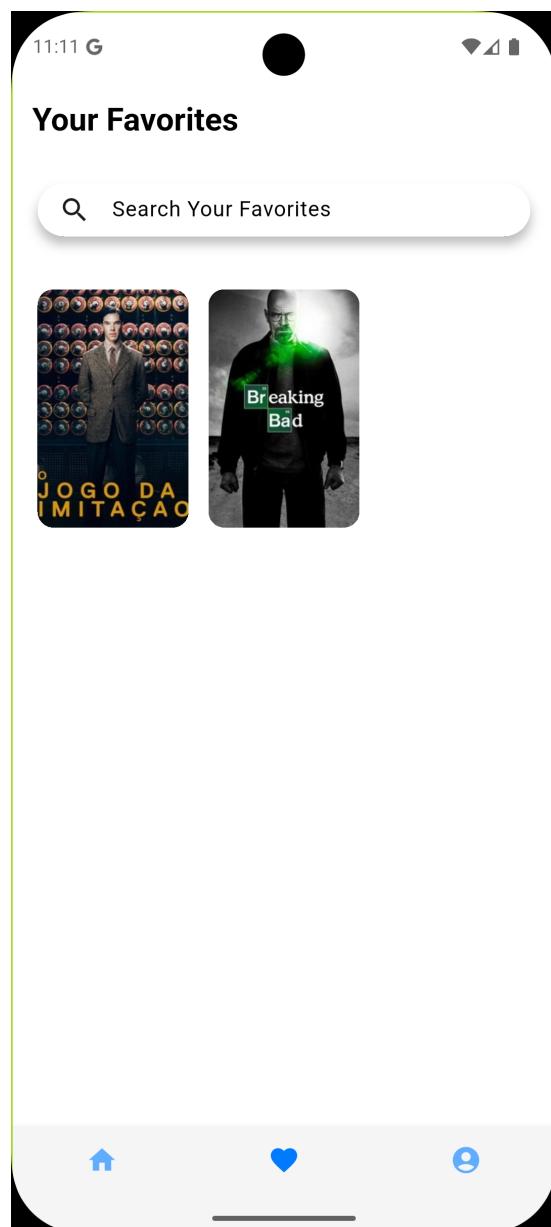


Figure 5. *
(b) Tela de Favoritos

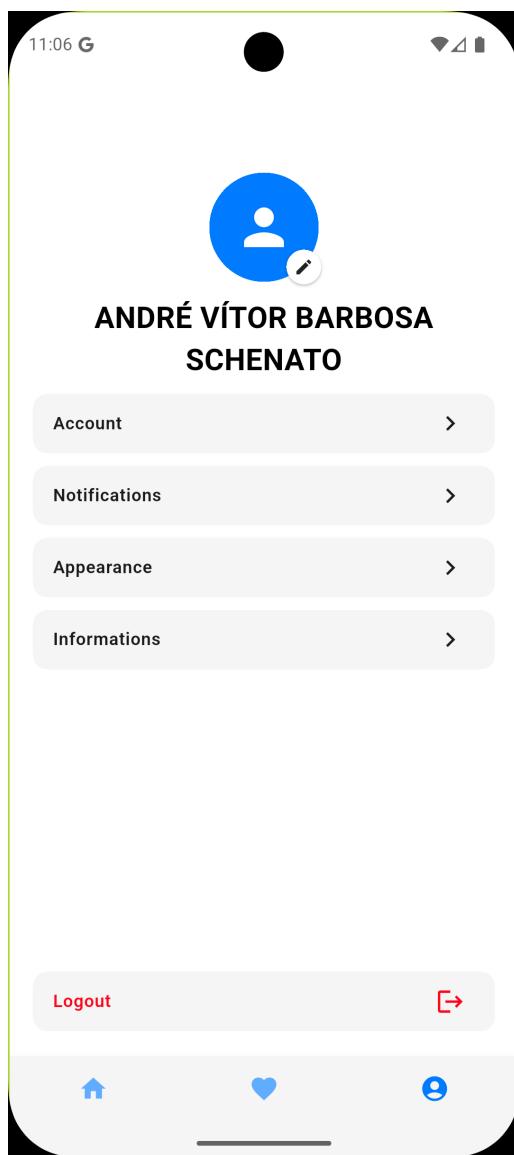


Figure 6. Tela de Perfil do Usuário.

4.4. Integração com APIs Externas

O *Prism* consome duas fontes externas principais: uma **API de catálogo audiovisual** (para obtenção de metadados de filmes, séries e artistas) e uma **API de Inteligência Artificial**, responsável por interpretar linguagem natural e auxiliar na geração de recomendações personalizadas.

Essas integrações ocorrem diretamente no aplicativo, por meio do módulo `core/api/api_client.dart`, que centraliza as requisições *HTTPS*, gerencia autenticação, cabeçalhos e parse das respostas. Os módulos de *features* utilizam fontes específicas (*sources*) para acessar endpoints distintos, mantendo o isolamento entre camadas e facilitando testes e evolução futura.

O serviço de IA é utilizado para analisar descrições inseridas pelo usuário, interpretar preferências declaradas e construir um contexto unificado de recomendação.

Já o serviço de catálogo retorna dados estruturados sobre mídias, temporadas, atores e informações relacionadas.

4.5. Banco de Dados e Persistência

A persistência é realizada pelo *Cloud Firestore*, que armazena perfis, preferências, listas de itens favoritos e registros produzidos durante o fluxo de recomendações. Cada módulo do aplicativo acessa o banco por meio de suas respectivas fontes de dados dedicadas, como `profile.firestore_source.dart` ou `favorite_source.dart`.

O modelo segue a estrutura de coleções típica do Firestore, permitindo organização flexível dos dados e sincronização em tempo real. O suporte nativo a operações *offline* garante continuidade do uso mesmo quando não há conexão estável. Todas as coleções e seus relacionamentos lógicos já estão representados no diagrama geral do sistema.

5. Resultados Obtidos

Esta seção apresenta os principais resultados obtidos com o desenvolvimento do projeto, bem como os desafios enfrentados durante sua execução. Também são discutidas as contribuições técnicas e as possibilidades de evolução da solução proposta.

5.1. Resultados

O desenvolvimento do *Prism* resultou em uma aplicação funcional capaz de realizar recomendações personalizadas de filmes, com autenticação de usuários, armazenamento em nuvem e integração dinâmica com *APIs* externas. O sistema apresenta uma interface moderna e responsiva, permitindo ao usuário cadastrar-se, favoritar títulos e visualizar sugestões personalizadas conforme suas preferências.

A integração com a *TMDb API* possibilitou o acesso a informações atualizadas e detalhadas sobre filmes e séries, enquanto o uso da *API Gemini* trouxe um diferencial na personalização das recomendações, permitindo que o sistema compreendesse solicitações em linguagem natural e sugerisse conteúdos de forma contextualizada.

Durante o desenvolvimento, destacaram-se os ganhos de produtividade obtidos pelo uso do *Flutter*, com recursos como o *Hot Reload*, e pela adoção do *Firebase*, que simplificou a estrutura do *backend* e reduziu custos de infraestrutura. O uso do *GitHub* garantiu versionamento e colaboração eficiente entre os membros da equipe, contribuindo para um processo de desenvolvimento mais ágil e organizado.

5.2. Discussões

Os principais desafios encontrados envolveram o gerenciamento da comunicação entre as *APIs* externas e o tratamento das respostas da inteligência artificial, além da definição de critérios para priorização de recomendações. Apesar dessas dificuldades, o projeto demonstrou ser tecnicamente viável e com alto potencial de escalabilidade.

A combinação entre tecnologias *serverless* e inteligência artificial provou-se eficaz para construir um sistema ágil e personalizável, abrindo caminho para futuras evoluções. Entre as possíveis melhorias estão a inclusão de análises de comportamento do usuário, recomendações baseadas em histórico de visualização e integração com plataformas de *streaming* reais. Dessa forma, o *Prism* pode se consolidar como uma ferramenta

completa para personalização de conteúdo audiovisual, aliando inovação tecnológica e experiência do usuário.

6. Conclusão

O projeto *Prism* cumpriu o objetivo de desenvolver um sistema de recomendação de filmes inteligente e *multiplataforma*, unindo usabilidade, escalabilidade e inovação tecnológica. A integração entre *Flutter*, *Firebase* e *APIs* de Inteligência Artificial demonstrou ser uma solução robusta e moderna, capaz de oferecer recomendações mais precisas e personalizadas aos usuários.

Além de aprimorar a experiência de descoberta de conteúdo, o trabalho contribuiu para o aprendizado prático sobre desenvolvimento de aplicações *full stack* baseadas em nuvem e reforçou a importância da colaboração e do versionamento de código em equipe.

Como perspectivas futuras, propõe-se expandir o sistema com novos algoritmos de recomendação híbrida, suporte à análise de sentimentos e integração com múltiplas fontes de dados, consolidando o *Prism* como uma ferramenta completa para personalização de conteúdo audiovisual.

Referências

- Figma.** Página inicial do Figma. Disponível em: <https://www.figma.com/pt-br/>. Acesso em: 1 set. 2025.
- Flutter.** Documentação oficial do Flutter. Disponível em: <https://docs.flutter.dev/>. Acesso em: 1 set. 2025.
- GitHub Docs.** Sobre o GitHub e o Git. Disponível em: <https://docs.github.com/pt/get-started/start-your-journey/about-github-and-git>. Acesso em: 20 ago. 2025.
- Google.** Firebase: Ampliar o Cloud Firestore com o Cloud Functions. Disponível em: <https://firebase.google.com/docs/firestore/extend-with-functions?hl=pt-br>. Acesso em: 5 set. 2025.
- The Movie Database.** TMDB API: Visão geral e aplicações. Disponível em: <https://developer.themoviedb.org/reference/intro/getting-started>. Acesso em: 5 set. 2025.
- Google DeepMind.** Gemini API: Documentação oficial. Disponível em: <https://ai.google.dev/docs>. Acesso em: 10 out. 2025.