

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 – Brazil

Resumo. *O presente trabalho apresenta o desenvolvimento do Prism, uma aplicação mobile voltada à recomendação personalizada de filmes, com o objetivo de otimizar o processo de descoberta de conteúdo audiovisual em plataformas de streaming. A solução foi construída com base em tecnologias modernas e escaláveis, utilizando o framework **Flutter** para o front-end e o **Firebase** como infraestrutura de back-end, integrando serviços como Authentication, Cloud Firestore e Cloud Functions em **Node.js**. Para enriquecer a experiência do usuário, o sistema estabelece conexão com a **IMDb API** para obtenção de metadados e com uma **API de Inteligência Artificial**, como o **Gemini**, responsável por interpretar preferências e gerar recomendações personalizadas por meio de processamento de linguagem natural. O Prism busca oferecer uma navegação intuitiva, com telas desenvolvidas no **Figma**, e uma experiência de uso consistente entre diferentes dispositivos. O resultado é uma aplicação funcional, segura e com potencial de expansão, que demonstra a viabilidade do uso combinado entre IA generativa e tecnologias serverless na criação de sistemas de recomendação modernos e eficientes.*

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

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, *Cloud Firestore* para persistência de dados e *Cloud Functions*, desenvolvidas em *Node.js*, para lógica de negócios *serverless*.

Para enriquecer a capacidade de recomendação, o *Prism* estabelece integração com *APIs* externas. A *IMDb API* é utilizada para a coleta de metadados abrangentes sobre filmes e séries, enquanto uma *API* de Inteligência Artificial (IA), como *Gemini* ou *Claude*, é empregada para processamento de linguagem natural. Esta integração permite a interpretação de *prompts* de contexto e solicitações do usuário, viabilizando a geração de recomendações altamente personalizadas. A interface do usuário é estruturada em seções intuitivas como “Favoritos”, “Perfeitos para Você” e “Você Também Pode Gostar”, buscando organizar as recomendações em diferentes níveis de relevância percebida. Os protótipos de design da aplicação foram desenvolvidos em *Figma*, conforme as melhores práticas de *User Experience (UX)*.

A crescente capacidade dos Grandes Modelos de Linguagem (*LLMs*), como *Gemini* ou *Claude*, em 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 (*Figma*, 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. NodeJS

O *Node.js* é um ambiente de execução de código *JavaScript* construído sobre o motor V8 do *Google*, permitindo que aplicações sejam desenvolvidas e executadas fora do navegador, em servidores ou sistemas *back-end* (*Node.js*, 2025). Essa capacidade amplia o alcance da linguagem, tornando possível utilizar o mesmo conhecimento de *JavaScript* tanto no *front-end* quanto no *back-end*, promovendo maior produtividade e padronização no desenvolvimento.

Uma das principais características do *Node.js* é seu modelo de arquitetura orientada a eventos e *I/O* não bloqueante (*non-blocking I/O*). Esse modelo permite que o ambiente gerencie múltiplas requisições simultaneamente sem a necessidade de criar novas *threads*, utilizando o conceito de *Event Loop*. Dessa forma, aplicações que demandam alto volume de conexões, como *APIs*, sistemas de chat ou plataformas em tempo real, conseguem operar com baixo consumo de recursos e alta escalabilidade (*Node.js*, 2025).

Além disso, o *Node.js* fornece um conjunto robusto de *APIs* que abrangem manipulação de arquivos, operações de rede, criptografia, execução de processos filhos e controle de *streams*, permitindo o desenvolvimento de soluções completas e seguras. A modularidade proporcionada pelos módulos do *Node* também facilita a manutenção do código, a reutilização de componentes e a integração com bibliotecas externas, tornando-o uma escolha estratégica para projetos que exigem flexibilidade e eficiência (*Node.js*, 2025).

A adoção do *Node.js* neste projeto justifica-se pela capacidade de unificar o desenvolvimento utilizando uma única linguagem, agilizar o processo de construção do *back-end* e permitir o desenvolvimento de aplicações escaláveis e de alto desempenho. Sua arquitetura eficiente, combinada com o amplo suporte de *APIs* e módulos, garante produtividade, facilidade de manutenção e robustez, atendendo às necessidades técnicas e práticas do projeto (*Node.js*, 2025).

2.3. 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.

2.3.1. 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).

2.3.2. 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).

2.3.3. Cloud Functions

Segundo o *Firebase* (2025), as *Cloud Functions* são funções *serverless* que permitem executar código em resposta a eventos disparados por outros serviços do *Firebase* ou de terceiros. No *Firestore*, essas funções podem ser acionadas por alterações em documentos ou coleções, possibilitando automações como envio de notificações, atualização de registros relacionados ou execução de cálculos em segundo plano. Elas eliminam a necessidade de gerenciar servidores, oferecendo escalabilidade automática e processamento seguro de eventos, sendo fundamentais para o desenvolvimento de funcionalidades avançadas.

A integração entre *Authentication*, *Cloud Firestore* e *Cloud Functions* cria um ecossistema completo para gerenciamento de usuários, armazenamento e processamento de dados de forma segura e eficiente. Essa combinação permite a criação de aplicações escaláveis, seguras e de fácil manutenção, alinhadas às melhores práticas de desenvolvimento moderno. O uso conjunto desses serviços proporciona infraestrutura robusta, facilita a implementação de funcionalidades avançadas e otimiza a gestão de recursos, contribuindo para o sucesso do projeto.

2.4. APIs Externas

A integração com *APIs* externas é um componente essencial para o desenvolvimento de aplicações modernas, permitindo a ampliação das funcionalidades e o acesso a dados e serviços especializados. No contexto deste projeto, a utilização da *IMDb API* e de uma *API* de Inteligência Artificial (IA), como *Gemini* ou *Claude*, desempenha um papel crucial na oferta de recomendações personalizadas e na melhoria da experiência do usuário.

2.4.1. IMDb API

Segundo a *IMDb* (2025), a *API IMDb* é uma interface de programação baseada em *GraphQL*, disponibilizada por meio do *AWS Data Exchange*, que permite acesso a dados atualizados sobre filmes, séries, atores e outros conteúdos do entretenimento. De

acordo com a documentação oficial, a *API* fornece dados em formato *JSON*, permitindo consultas precisas sem a necessidade de manipular parâmetros complexos, facilitando a integração com sistemas externos e o desenvolvimento de aplicações interativas (*IMDb*, 2025).

A *IMDb* (2025) também descreve que a *API* utiliza o *GraphQL*, uma linguagem de consulta que possibilita requisitar exatamente os dados necessários, evitando sobrecarga de informações desnecessárias. Isso otimiza o desempenho das aplicações, principalmente em cenários de alto volume de consultas. A *API* também suporta requisições que combinam múltiplos títulos ou nomes em uma única consulta, tornando o acesso às informações mais eficiente e ágil (*IMDb*, 2025).

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.

Além disso, a documentação da *IMDb* destaca que é necessário possuir uma conta na *AWS* e assinar o produto correspondente no *AWS Data Exchange* para utilizar a *API*. Após a assinatura, o desenvolvedor recebe chaves de acesso para realizar requisições, podendo configurar o ambiente e autenticar-se em diferentes linguagens de programação, como *Java*, *Python* e *TypeScript* (*IMDb*, 2025).

Segundo a *IMDb* (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 (*IMDb*, 2025).

2.4.2. APIs de Inteligência Artificial (IA)

A Inteligência Artificial (IA) é um dos pilares centrais do sistema *Prism*, sendo responsável pelo processamento de linguagem natural e pela geração de recomendações personalizadas para cada usuário. O projeto utiliza a *API Google Gemini*, uma plataforma de IA generativa desenvolvida pela *Google DeepMind*, projetada para interpretar texto, compreender contextos e gerar respostas e análises contextualizadas com base em grandes volumes de dados.

O modelo *Gemini*, lançado oficialmente em 2024, representa a evolução dos modelos de linguagem de grande porte (*Large Language Models — LLMs*), capazes de processar texto, código e dados multimodais. Sua arquitetura baseia-se em redes neurais profundas treinadas em um extenso conjunto de dados, o que permite à *API*

compreender intenções, sentimentos e preferências expressas pelos usuários, fornecendo recomendações mais precisas e relevantes (DeepMind, 2025).

No contexto do *Prism*, a *API Gemini* é integrada ao *backend* por meio das *Cloud Functions* do *Firebase*, utilizando requisições *HTTP* seguras para envio de *prompts* e retorno das respostas processadas. Essa integração possibilita que a aplicação interprete mensagens de texto enviadas pelos usuários, analise suas preferências de filmes e séries e gere sugestões personalizadas com base no histórico de interação e nas avaliações anteriores. Dessa forma, a IA atua como um mecanismo dinâmico de recomendação contextual, indo além dos métodos tradicionais de filtragem colaborativa ou de conteúdo.

Entre as principais vantagens da adoção da *API Gemini*, destacam-se:

- **Compreensão de Linguagem Natural:** a IA é capaz de interpretar descrições subjetivas fornecidas pelos usuários, como “filmes com finais surpreendentes” ou “séries parecidas com *Breaking Bad*”, identificando padrões semânticos e contextuais;
- **Recomendações Personalizadas:** o modelo gera listas de filmes adaptadas às preferências do usuário, combinando dados da *IMDb API* com análises comportamentais;
- **Aprendizado Contínuo:** as respostas da IA podem ser refinadas com base nas interações, aprimorando a acurácia das recomendações ao longo do tempo;
- **Integração Serverless:** a execução é feita de forma totalmente automatizada via *Cloud Functions*, eliminando a necessidade de servidores dedicados e reduzindo custos operacionais.

A utilização da *API Gemini* no projeto justifica-se por sua capacidade de combinar o poder dos modelos de linguagem com a praticidade do ecossistema *Firebase*, resultando em um sistema de recomendação inteligente, escalável e de fácil manutenção. Essa integração reforça o caráter inovador do *Prism*, que não apenas exibe dados estáticos, mas interpreta preferências subjetivas, tornando a experiência de descoberta de conteúdo mais fluida, personalizada e contextualizada.

2.5. 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.

2.6. 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 *multi-plataforma* 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*), banco de dados em tempo real e *NoSQL* (*Cloud Firestore*) e funções *serverless* (*Cloud Functions*) escritas em *Node.js*;
- **IMDb 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 tomadas ao longo do processo de construção.

4.1. Arquitetura do Sistema

O sistema adota uma arquitetura em camadas, baseada na separação entre as responsabilidades de *frontend*, *backend* e serviços externos. Essa estrutura visa garantir escalabilidade, modularidade e facilidade de manutenção.

A aplicação é composta por três principais camadas:

- **Camada de Apresentação (Frontend)** — desenvolvida em *Flutter*, responsável pela interface gráfica e pela interação direta com o usuário;
- **Camada de Lógica e Processamento (Backend)** — implementada sobre a infraestrutura do *Firebase*, utilizando *Cloud Functions* escritas em *Node.js* para o processamento de dados e execução das regras de negócio;
- **Camada de Integração** — responsável pela comunicação com APIs externas, como a *IMDb API* e as APIs de Inteligência Artificial (IA), que enriquecem a base de dados e aprimoram as recomendações de filmes.

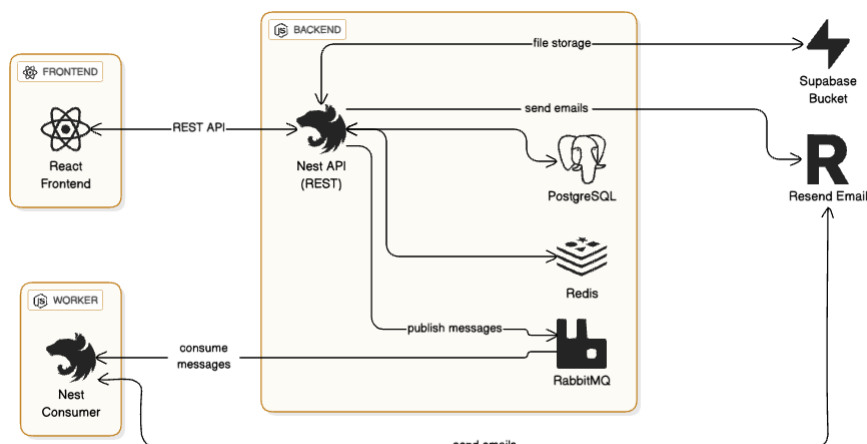


Figure 1. Arquitetura do Sistema Prism

Essa arquitetura proporciona um fluxo de dados contínuo e seguro entre o cliente e o servidor, aproveitando os recursos *serverless* do *Firebase* para reduzir custos e simplificar a escalabilidade. A escolha dessa estrutura se deve à necessidade de integrar diferentes serviços em nuvem de forma eficiente, mantendo desempenho elevado e baixo acoplamento entre os módulos.

4.2. Estrutura do *Backend*

A organização interna do *backend* segue os princípios de modularização e reutilização de código, com foco na integração nativa entre os serviços do *Firebase*. As principais tecnologias utilizadas incluem o *Firebase Authentication*, o *Cloud Firestore* e as *Cloud Functions*.

Cada módulo contém:

- Registro do módulo com dependências e provedores (**.module.ts*);
- Funções assíncronas para manipulação de dados e integração com o *Firestore*;
- *Triggers* automáticas que respondem a eventos do banco de dados, como inserções ou atualizações de registros;
- *Middleware* de autenticação que valida o *token* do usuário antes de permitir qualquer operação sensível.

A implementação em *Node.js* aproveita o modelo de I/O não bloqueante (*non-blocking I/O*) para lidar com múltiplas requisições simultâneas, mantendo o desempenho e reduzindo a latência. As *Cloud Functions* atuam como o principal ponto de processamento das recomendações e de sincronização dos dados, sendo disparadas automaticamente conforme as interações do usuário.

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.

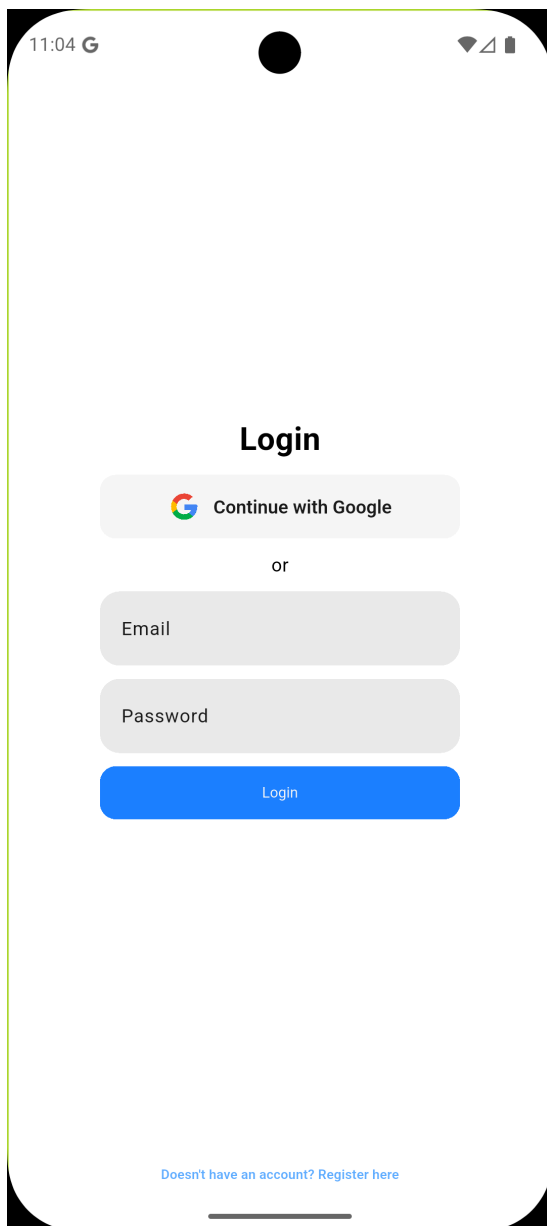


Figure 2. *
(a) Tela de Login e Cadastro

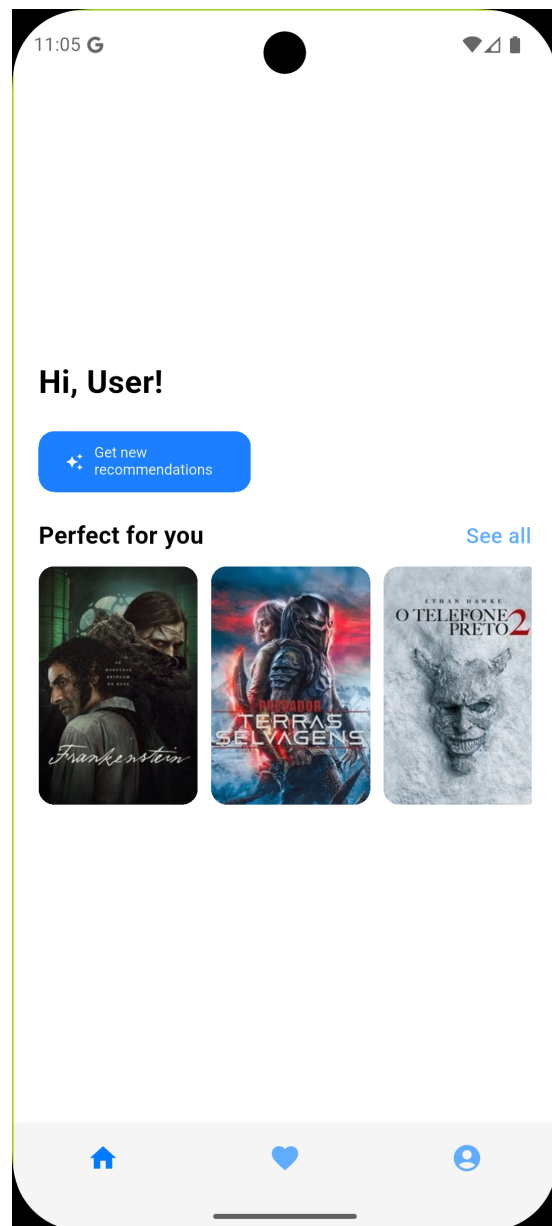


Figure 3. *
(b) Tela Inicial

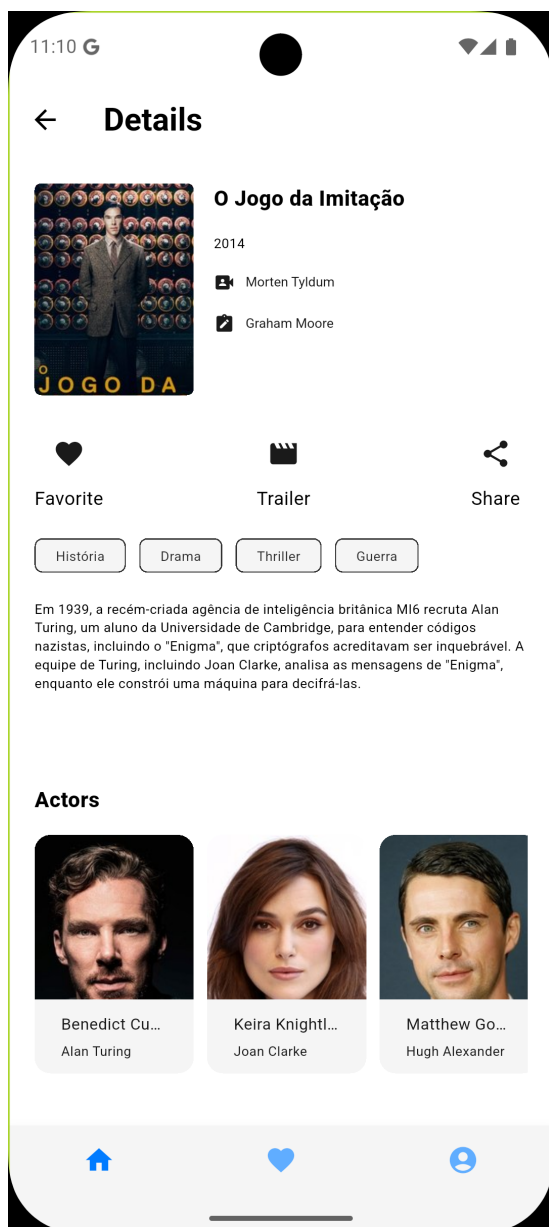


Figure 4. *
(a) Tela de Detalhes do Filme

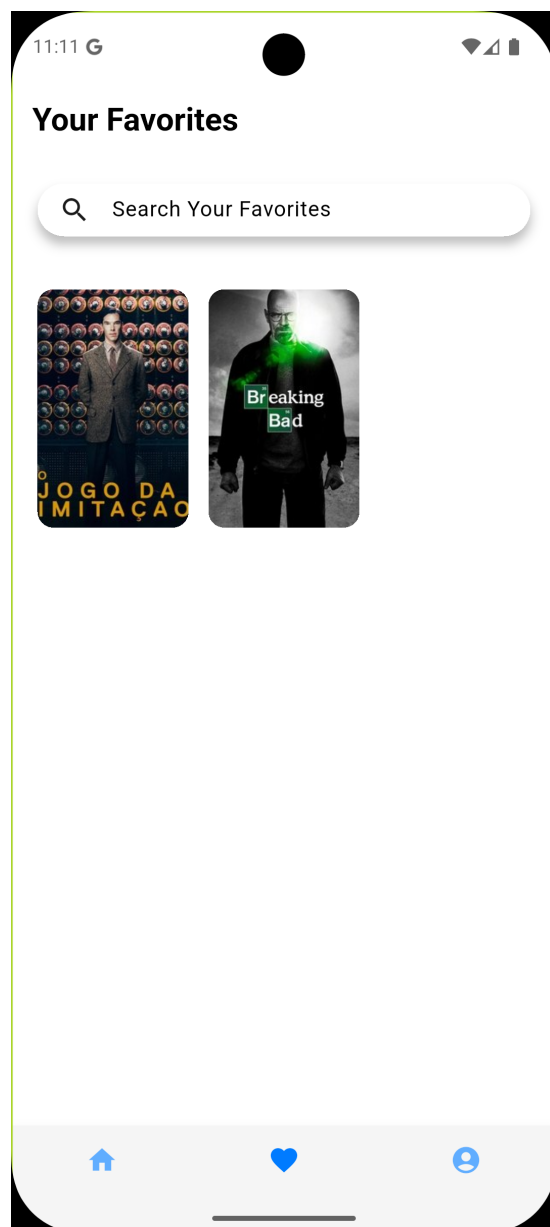


Figure 5. *
(b) Tela de Favoritos

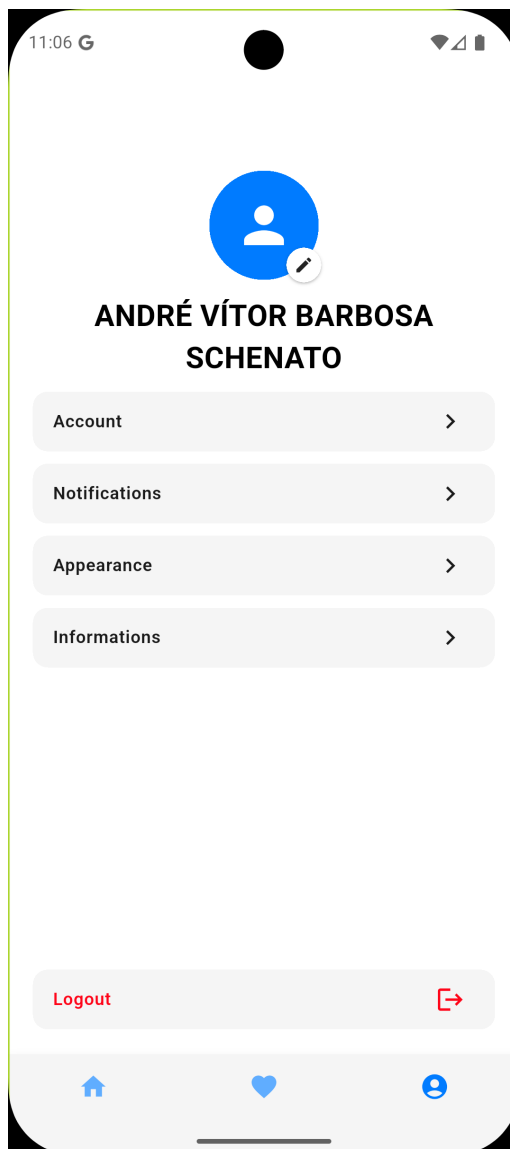


Figure 6. Tela de Perfil do Usuário.

4.4. Integração com APIs Externas

A integração com *APIs* externas constitui um dos principais diferenciais do *Prism*. O sistema se conecta à **API IMDb**, que fornece metadados de filmes e séries, e a uma **API de Inteligência Artificial** (*Gemini* ou *Claude*), utilizada para processar linguagem natural e compreender o contexto das solicitações do usuário.

Essa integração é feita por meio de requisições *HTTPS* autenticadas e tratadas nas *Cloud Functions*, que interpretam as respostas e estruturam os dados para exibição no aplicativo. Com isso, o sistema consegue gerar recomendações contextuais, identificar padrões de preferência e exibir informações detalhadas e atualizadas diretamente das fontes oficiais.

4.5. Banco de Dados e Persistência

A persistência de dados é gerenciada via **Cloud Firestore**, um banco de dados *NoSQL* em tempo real oferecido pelo **Firebase**. Os dados são organizados em coleções e documentos, permitindo a estruturação dinâmica das informações de usuários, filmes e recomendações.

Além disso, o **Firestore** oferece sincronização automática e suporte *offline*, garantindo que o aplicativo continue funcional mesmo sem conexão estável com a internet. A modelagem ER apresentada na Figura 7 demonstra a relação entre as coleções do sistema e a forma como os dados são interligados.

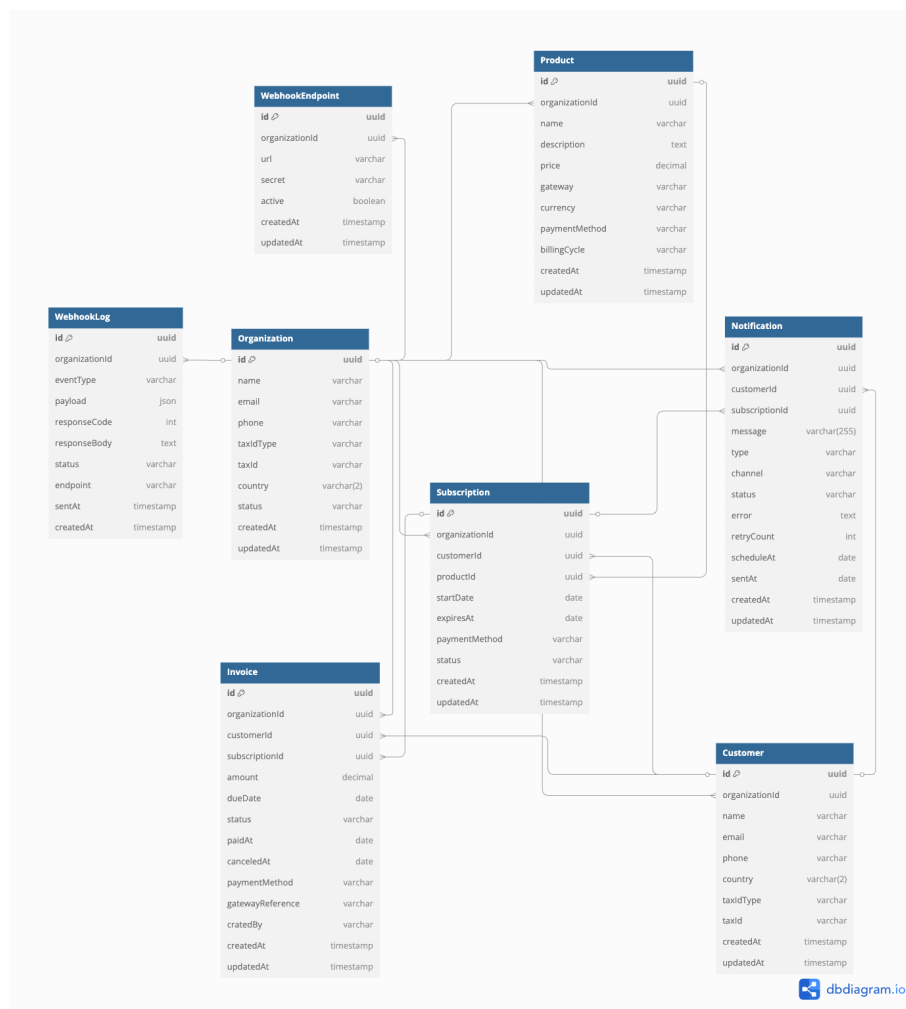


Figure 7. Modelo ER do Banco de Dados

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 **IMDb 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 **Firestore**, 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*, *Firestore* 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/>.

`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.

IMDb. IMDb API: Visão geral e aplicações. Disponível em: <https://developer.imdb.com/documentation/api-documentation/>. Acesso em: 5 set. 2025.

Node.js Foundation. Documentação oficial da API Node.js. Disponível em: <https://nodejs.org/docs/latest/api/>. Acesso em: 5 set. 2025.

Google DeepMind. Gemini API: Documentação oficial. Disponível em: <https://ai.google.dev/docs>. Acesso em: 10 out. 2025.