

UNIVERSIDADE PAULISTA – UNIP EaD

Projeto Integrado Multidisciplinar

Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas

VINICIUS MANOEL RODRIGUES DA SILVA – 2256756

Sistema de streaming de conteúdo multimídia

Senador Canedo - GO

2025

VINICIUS MANOEL RODRIGUES DA SILVA – 2256756

Sistema de streaming de conteúdo multimídia

Projeto Integrado Multidisciplinar em
Análise e Desenvolvimento de Projetos

Projeto Integrado Multidisciplinar para obtenção do título de tecnólogo em (SUP TEC EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS), apresentado à Universidade Paulista – UNIP EaD.

Orientador (a): Prof. MSc. Tarcísio Peres

Senador Canedo - GO

2025

SUMÁRIO

1. INTRODUÇÃO.....	5
2. A NINELINKS.....	6
3. TECNOLOGIAS UTILIZADAS.....	7
3.1. A LINGUAGEM C#.....	7
3.2. ASP.NET CORE.....	8
3.3. ENTITY FRAMEWORK CORE.....	8
3.4. BASE DE DADOS: SQL SERVER EXPRESS.....	9
3.5. SWAGGER (OPENAPI + SWAGGER UI).....	10
3.6. NET MAUI.....	10
3.7. JAVA (ANDROID).....	11
3.8. GITHUB.....	11
4. FUNCIONALIDADES DO SISTEMA DE STREAMING.....	12
4.1. AUTENTICAÇÃO JWT.....	12
4.2. ORGANIZAÇÃO DOS ENDPOINTS.....	12
5. VISUALIZAÇÃO DA API COM SWAGGER.....	14
5.1. ENDPOINTS DA ENTIDADE USUARIO.....	14
5.2. ENDPOINTS DA ENTIDADE CONTEUDO.....	19
5.3. ENDPOINTS DA ENTIDADE CRIADOR.....	24
5.4. ENDPOINTS DA ENTIDADE CURTIDA.....	26
5.5. ENDPOINTS DA ENTIDADE PLAYLIST.....	32
5.6. ENDPOINTS DA ENTIDADE ITEMPLAYLIST.....	36
5.7. ENDPOINTS DA ENTIDADE VISUALIZACAO.....	39
5.8. ENDPOINTS DA ENTIDADE COMENTÁRIO.....	43
5.9. CONSIDERAÇÕES FINAIS SOBRE A DOCUMENTAÇÃO DOS ENDPOINTS.....	45
6. ESTRUTURA DO APLICATIVO MOBILE.....	46
6.1. DEPENDÊNCIAS UTILIZADAS.....	46
6.2. NAVEGAÇÃO E INTERFACE.....	47
6.3. COMPATIBILIDADE E COMUNICAÇÃO.....	47
6.4. APRESENTAÇÃO DAS TELAS DO APLICATIVO MOBILE.....	48
6.5. Diagrama de caso de uso.....	67
6.6. CONSIDERAÇÕES FINAIS SOBRE O APLICATIVO MOBILE.....	69
7. PROTÓTIPO DA INTERFACE DESKTOP COM .NET MAUI.....	70
7.1. CONSIDERAÇÕES FINAIS SOBRE O APLICATIVO EM .NET MAUI.....	74
8. ESTRUTURA RELACIONAL DO BANCO DE DADOS.....	74
9. PLANILHA DE TESTES – INTERAÇÃO DO USUÁRIO.....	77
10. CONCLUSÃO.....	80
REFERÊNCIAS.....	81

RESUMO

Este projeto tem como foco o desenvolvimento de um sistema de streaming de conteúdo multimídia, com funcionalidades voltadas tanto para criadores quanto para consumidores de mídia. A plataforma foi desenvolvida pela **NineLinks**, com o objetivo de permitir que criadores publiquem vídeos, músicas e podcasts, enquanto usuários finais acessam e interagem com esse conteúdo de forma dinâmica. O sistema inclui autenticação segura, controle de playlists, comentários, curtidas e métricas de visualização, permitindo uma experiência completa de navegação e gerenciamento de conteúdo.

A API foi construída em C# utilizando o ASP.NET Core e Entity Framework, com endpoints bem definidos, autenticação via JWT e documentação completa por meio do Swagger. Durante o desenvolvimento, o uso de C# exigiu domínio de conceitos como injeção de dependência, organização em camadas e manipulação segura de dados relacionais. Esses desafios contribuíram significativamente para a evolução técnica de **DevViniNine**, resultando em maior domínio sobre a linguagem e sobre boas práticas de desenvolvimento modernas.

A interface gráfica voltada ao criador de conteúdo foi prototipada utilizando .NET MAUI, enquanto o aplicativo Android foi implementado em Java, garantindo acessibilidade multiplataforma. O projeto é um exemplo prático e funcional de como integrar diferentes tecnologias para entregar soluções completas e escaláveis no segmento de streaming.

Palavras-Chave: Streaming, Multimídia, C#, .NET MAUI, Swagger, Java, Android, API, JWT

ABSTRACT

This project focuses on the development of a multimedia content streaming system, designed to serve both content creators and media consumers. Developed by **NineLinks**, the platform allows creators to publish videos, music, and podcasts, while end users can access and interact with this content through an intuitive and dynamic experience. The system includes secure authentication, playlist control, user comments, likes, and content view metrics, offering a complete environment for managing and consuming multimedia.

The backend API was developed in C# using ASP.NET Core and Entity Framework, featuring well-structured endpoints, JWT-based authentication, and full documentation provided via Swagger. The use of C# posed several technical challenges, such as mastering dependency injection, layered architecture, and safe handling of relational data. These aspects significantly contributed to the technical growth of **DevViniNine**, resulting in a stronger command of the language and modern development practices.

The user interface for content creators was prototyped using .NET MAUI, while the Android mobile application was implemented in Java, ensuring cross-platform accessibility. This project stands as a practical and functional example of how to integrate multiple technologies to deliver scalable and comprehensive solutions in the streaming market.

Keywords: Streaming, Multimedia, C#, .NET MAUI, Swagger, Java, Android, API, JWT

1. INTRODUÇÃO

Este trabalho tem como foco o desenvolvimento de um sistema de streaming de conteúdo multimídia, idealizado e implementado pela empresa **NineLinks**, com o objetivo de oferecer uma plataforma virtual robusta e moderna para publicação e consumo de vídeos, músicas e podcasts. O sistema foi planejado para atender dois públicos principais: os criadores de conteúdo, que precisam de ferramentas práticas para gerenciar suas produções, e os usuários finais, que buscam uma experiência intuitiva e completa de acesso e interação com a mídia.

Durante o desenvolvimento, foram aplicadas boas práticas de arquitetura de software, visando a escalabilidade, a segurança e a integração entre tecnologias distintas. A Web API foi construída em C# com ASP.NET Core, contando com autenticação via JWT e operações CRUD estruturadas com Entity Framework. A documentação da API foi gerada com Swagger, permitindo testes e visualização clara dos endpoints. Para os criadores de conteúdo, foi desenvolvido um protótipo semi funcional em .NET MAUI, simulando ações como upload de conteúdo, criação de playlists e visualização de métricas. Já para o usuário final, o acesso à plataforma é realizado por meio de um aplicativo Android desenvolvido em Java.

O sistema foi projetado para manter o controle sobre interações como curtidas, visualizações e comentários, além de permitir a associação de conteúdos a playlists personalizadas. A implementação dessas funcionalidades promove um ambiente digital organizado, interativo e adaptável a diferentes perfis de usuários. Este projeto representa uma solução prática e completa para plataformas de streaming, integrando diferentes camadas e tecnologias em um mesmo ecossistema.

2. A NINELINKS

A **NineLinks** é uma empresa com foco no desenvolvimento de aplicações para desktop, especializada em soluções sob medida para diferentes segmentos do mercado. Ao longo de sua trajetória, a empresa concentrou seus esforços na criação de sistemas robustos e funcionais voltados ao ambiente Windows. Este projeto marca um novo capítulo para a organização, sendo o primeiro a adotar uma arquitetura multiplataforma, com tecnologias como .NET MAUI, ASP.NET Core e Java, integrando diferentes dispositivos e ambientes operacionais. A conclusão do sistema de streaming multimídia representará não apenas um avanço técnico para a empresa, mas também a consolidação de um portfólio moderno, versátil e alinhado às exigências do mercado atual.

O projeto é conduzido por **Vinicius Manoel**, conhecido como **DevViniNine**, desenvolvedor da NineLinks com experiência prévia em uma startup voltada ao setor de soluções ambientais. Durante sua atuação anterior, participou ativamente do desenvolvimento de um sistema em linguagem C para o gerenciamento de cadastros e geração de relatórios sobre as atividades ambientais de indústrias. Essa vivência proporcionou uma base sólida em lógica de programação, estrutura de dados e controle de processos.

Com o novo desafio de criar uma plataforma completa de streaming multimídia, Vinicius teve a oportunidade de expandir significativamente seus conhecimentos, enfrentando e superando dificuldades técnicas com a linguagem C#, a modelagem de bancos relacionais e a integração entre diferentes camadas do sistema. Esse projeto representa um marco tanto para o desenvolvedor quanto para a NineLinks, destacando a capacidade de adaptação e a evolução tecnológica de ambos.

3. TECNOLOGIAS UTILIZADAS

Aplicações desktop são programas projetados para serem instalados e executados diretamente em computadores pessoais, como desktops e notebooks. Elas operam localmente, utilizando os recursos do hardware e do sistema operacional onde estão instaladas, como memória, CPU e disco rígido.

Esses aplicativos geralmente têm uma interface gráfica de usuário (GUI) que permite interação fácil por meio de menus, botões e outras ferramentas visuais. Diferentemente das aplicações web, que dependem de um navegador e conexão à internet, as aplicações desktop funcionam de forma independente, embora possam oferecer funcionalidades online, como sincronização de dados.

Exemplos incluem editores de texto, ferramentas de design gráfico, softwares de produtividade e até jogos. Sua principal vantagem é a velocidade e o acesso total aos recursos do sistema, tornando-as ideais para tarefas que demandam alto desempenho ou segurança.

3.1. A LINGUAGEM C#

A linguagem C# (pronuncia-se “C Sharp”) foi criada pela Microsoft no início dos anos 2000, como parte da plataforma .NET. Desde então, ela tem evoluído constantemente e hoje é uma das linguagens mais versáteis e utilizadas no desenvolvimento de software profissional, principalmente em ambientes Windows. O poder do C# está justamente na sua flexibilidade: com ela é possível criar desde aplicativos de linha de comando, aplicações web robustas, serviços de API, até jogos 3D completos.

A primeira experiência do desenvolvedor **DevViniNine** com a linguagem foi justamente nessa última área, estudando desenvolvimento de jogos com a **Unity Engine**, uma das maiores plataformas de criação de jogos da atualidade — que utiliza C# como linguagem principal de scripts. Essa experiência inicial foi fundamental para criar familiaridade com a sintaxe, a estrutura e a lógica da linguagem. Com o tempo, essa base foi ampliada para o uso do C# em aplicações

comerciais, como neste projeto de streaming, onde a linguagem foi essencial para a construção da API e a modelagem de entidades com segurança e organização.

3.2. ASP.NET CORE

O ASP.NET Core é o framework da Microsoft para desenvolvimento de aplicações web modernas, modulares e de alta performance. Ele foi projetado para ser multiplataforma, o que significa que os sistemas desenvolvidos podem ser executados no Windows, Linux ou macOS sem modificações profundas. No projeto da NineLinks, o ASP.NET Core foi utilizado como base para a criação da Web API, responsável por controlar os acessos, executar as operações CRUD e manter a comunicação entre banco de dados e os aplicativos front-end.

Além da sua estrutura bem definida, o ASP.NET Core se destaca pela facilidade de integração com segurança via JWT, uso de serviços e injeção de dependência — todos recursos utilizados neste projeto para garantir segurança, escalabilidade e organização do código.

3.3. ENTITY FRAMEWORK CORE

O **Entity Framework Core** (EF Core) é a tecnologia utilizada no projeto para fazer a ponte entre a aplicação em C# e o banco de dados relacional. Ele funciona como um ORM (Object-Relational Mapper), permitindo que os dados sejam tratados como objetos de C#, sem a necessidade de escrever comandos SQL diretamente para cada operação.

No sistema da NineLinks, o EF Core foi essencial para mapear as entidades centrais, como **Usuario**, **Conteudo**, **Playlist**, **Comentario**, **Curtida**, **Criador**, **ItemPlaylist** e **Visualizacao**, tornando o acesso e a manipulação de dados mais seguros e organizados. A ferramenta também foi utilizada para aplicar **migrations**, possibilitando alterações na estrutura do banco de dados ao longo do projeto de forma controlada e sem a perda de dados.

3.4. BASE DE DADOS: SQL SERVER EXPRESS

Para o gerenciamento e armazenamento dos dados da aplicação, foi adotado o **SQL Server Express**, uma edição gratuita e oficial do sistema de gerenciamento de banco de dados relacional da **Microsoft**. Essa versão é voltada especialmente para aplicações de pequeno a médio porte, oferecendo um conjunto robusto de funcionalidades essenciais, com algumas limitações de recursos em relação às versões pagas (como limite de tamanho de banco de dados e uso de memória), mas que não impactaram negativamente no escopo deste projeto.

O **SQL Server Express** possui como principais características:

- **Gratuito e leve**: Ideal para ambientes de desenvolvimento, testes ou aplicações menores.
- **Compatibilidade total com EF Core**: Permite a integração fluida com o Entity Framework Core, utilizado neste projeto para o mapeamento objeto-relacional (ORM), facilitando a criação, consulta e manipulação de dados diretamente via código C#.
- **Ambiente confiável**: Mesmo sendo uma versão gratuita, o SQL Server Express oferece estabilidade, segurança e suporte aos principais recursos relacionais, como chaves estrangeiras, procedimentos armazenados, índices e transações.

Durante o desenvolvimento, a escolha do SQL Server Express proporcionou um ambiente eficaz para **modelagem de dados, testes com dados reais e validação da estrutura relacional**, além de garantir **alto desempenho nas operações CRUD (Create, Read, Update, Delete)**.

Essa base de dados foi fundamental para sustentar as funcionalidades da API e manter a integridade das informações manipuladas tanto no aplicativo móvel quanto na interface .NET MAUI.

3.5. SWAGGER (OPENAPI + SWAGGER UI)

O **Swagger**, atualmente integrado ao ecossistema **OpenAPI**, surgiu em 2010 com o objetivo de padronizar e facilitar a documentação de APIs REST. No projeto da **NineLinks**, ele teve um papel fundamental ao gerar automaticamente uma documentação interativa e acessível a partir do próprio código-fonte da Web API.

Através do **Swagger UI**, os endpoints podem ser visualizados e testados diretamente pelo navegador, o que foi extremamente útil tanto para o desenvolvedor quanto para a validação geral da estrutura da API. A interface permite realizar requisições reais (como GET, POST, PUT, DELETE), além de suportar autenticação com token JWT, facilitando o teste de rotas protegidas.

Ter o Swagger implementado no projeto proporcionou **clareza na organização dos recursos**, agilidade durante os testes e uma visão geral das funcionalidades disponíveis — especialmente nas fases de integração entre o backend e os aplicativos front-end.

3.6. .NET MAUI

O **.NET MAUI** (Multi-platform App UI) é a evolução do **Xamarin.Forms** e permite a criação de interfaces gráficas multiplataforma com código único. Com ele, é possível desenvolver aplicativos para Windows, Android, iOS e macOS a partir de uma mesma base em C#. Embora ainda em processo de amadurecimento, o MAUI já é considerado uma solução poderosa para projetos que exigem prototipação rápida e visual consistente entre plataformas.

Neste projeto, a interface voltada aos criadores de conteúdo foi prototipada com **.NET MAUI**. Ela simula as principais funções de gestão de mídia, como upload de arquivos, análise de métricas e gerenciamento de playlists. Mesmo que o foco da aplicação ainda seja o ambiente Windows, o uso do MAUI abre portas para futuras versões em outras plataformas, mostrando a versatilidade e o futuro multiplataforma da NineLinks.

3.7. JAVA (ANDROID)

O aplicativo voltado ao usuário final foi desenvolvido em Java, utilizando o Android Studio como ambiente principal. Java continua sendo uma das linguagens mais populares no ecossistema Android, e sua estabilidade e compatibilidade com diferentes dispositivos foram determinantes para sua escolha neste projeto.

A versão mobile da plataforma permite que os usuários naveguem por vídeos, músicas e imagens, consumam o conteúdo diretamente do repositório online, curtam, comentem e salvem conteúdos em playlists. A comunicação com a API foi realizada com Retrofit, uma biblioteca eficiente para lidar com requisições HTTP, tornando o app leve e responsivo mesmo em conexões instáveis.

3.8. GITHUB

O GitHub foi utilizado como repositório principal para o controle de versão e organização do projeto de streaming desenvolvido pela NineLinks. Sua utilização foi essencial para registrar o histórico de alterações, realizar backups automáticos e manter o código-fonte acessível e seguro ao longo do desenvolvimento. A plataforma também facilitou o gerenciamento das etapas de evolução do sistema, permitindo que os arquivos fossem organizados por pastas e versões, algo indispensável para manter a consistência em projetos com múltiplos módulos como este.

Além de armazenar o código da Web API, do protótipo em .NET MAUI e do aplicativo Android, o GitHub também foi usado como repositório de mídia para testes do app. Foram hospedados arquivos de vídeos, músicas e imagens dentro da estrutura do repositório, mais especificamente na pasta Conteudos, que foi acessada diretamente pelos aplicativos em tempo real. Isso possibilitou que o sistema fosse testado com arquivos reais durante o desenvolvimento, sem a necessidade de configurar servidores de mídia externos, agilizando os testes e garantindo resultados próximos ao uso final.

Essa prática demonstrou como o GitHub pode ir além do versionamento de código, servindo também como apoio para armazenamento temporário e distribuição de arquivos estáticos durante a fase de prototipação e validação funcional.

4. FUNCIONALIDADES DO SISTEMA DE STREAMING

O sistema de streaming de conteúdo multimídia desenvolvido pela NineLinks foi dividido em três camadas principais: a Web API, responsável por toda a lógica e persistência de dados; um protótipo de interface gráfica voltada aos criadores de conteúdo, construída com .NET MAUI; e o aplicativo Android, destinado aos usuários finais. A seguir, serão apresentados os principais componentes da aplicação, com capturas de tela das interfaces, descrição das funcionalidades e exemplos de uso.

4.1. AUTENTICAÇÃO JWT

Para garantir segurança e controle de acesso, foi implementada **autenticação baseada em JWT (JSON Web Token)**. Ao realizar o login com e-mail e senha válidos, o servidor retorna um token codificado que deve ser enviado nas próximas requisições, através do cabeçalho **Authorization**, no formato:

Bearer {token}

Isso permite restringir rotas específicas apenas para usuários autenticados e vincular ações (como curtir um conteúdo ou criar uma playlist) ao usuário que realizou a requisição.

4.2. ORGANIZAÇÃO DOS ENDPOINTS

Os endpoints da API estão organizados de forma lógica e modular, agrupados por área de responsabilidade. Isso facilita a manutenção e a navegação no Swagger. A seguir, os principais agrupamentos:

Usuário : Login, cadastro, listagem, alteração e exclusão de usuários, Busca por id, Busca por email.

Conteúdo : Cadastro de vídeos, músicas, podcasts, alteração, exclusão e listagem geral.

Criador : Cadastro como criador de conteúdo, listagem de criadores, alteração e exclusão.

Playlist : Criação, edição, exclusão e consulta de playlists do usuário logado.

ItemPlaylist : Inserção e listagem de conteúdos vinculados a uma playlist.

Curtida : Curte e descurte conteúdos, listagem de curtidas por conteúdo e por usuário.

Comentário : Inserção e visualização de comentários vinculados a um conteúdo.

Visualização : Registro de visualizações, listagem de conteúdos populares e recentes.

Exemplo de Requisição

Abaixo está um exemplo real de requisição feita ao endpoint de login:

Requisição:

```
POST /api/Usuario/login
Content-Type: application/json

{
  "email": "usuario@email.com",
  "password": "senha123"
}
```

Figura 1 – Requisição de login

Resposta:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Figura 2 – Resposta da requisição de login

Esse token pode então ser utilizado nas próximas requisições protegidas, como o endpoint:

```
GET /api/Playlist/MinhasPlaylist
Authorization: Bearer {token}
```

Figura 3 – Exemplo requisição protegidas

5. VISUALIZAÇÃO DA API COM SWAGGER

A seguir, são apresentadas capturas de tela retiradas diretamente da interface do Swagger UI, que demonstram como a Web API do sistema de streaming foi estruturada. As imagens ilustram os principais endpoints disponíveis, incluindo exemplos de requisições e suas respectivas respostas.

Essas capturas têm como objetivo tornar mais clara a organização da API, evidenciando os métodos criados para o gerenciamento de usuários, conteúdos, playlists, interações e autenticação. Também são exibidos exemplos práticos de envio de dados (Request Body) e os formatos de retorno (Response Body), como ocorrem durante a comunicação entre o frontend e o backend.

Além de servirem como documentação técnica, os prints mostram que os endpoints estão funcionalmente operacionais e prontos para integração com diferentes plataformas, alidando o uso da API em ambientes reais.

5.1. ENDPOINTS DA ENTIDADE USUARIO

Esta seção apresenta os principais endpoints relacionados ao gerenciamento de usuários da plataforma. Eles permitem a realização de operações como cadastro, login, listagem, busca por ID ou e-mail, atualização e exclusão de registros. Esses recursos são essenciais para controlar o acesso ao sistema, autenticar usuários por meio de token JWT e definir permissões administrativas.

Os endpoints foram implementados utilizando os métodos HTTP adequados (GET, POST, PUT e DELETE), seguindo o padrão REST, e retornam respostas no formato JSON, com estrutura clara e objetiva. Ao longo das figuras seguintes, são apresentados exemplos reais de requisições e respostas obtidas diretamente pela interface Swagger da API.

POST /api/Usuario/cadastrar

Parameters

No parameters

Request body application/json

```
{
  "id": 0,
  "nome": "DevViniNine",
  "email": "devvininine@gmail.com",
  "password": "Admin123",
  "admin": 1
}
```

Responses

Curl

```
curl -X 'POST' \
'http://localhost:5127/api/Usuario/cadastrar' \
-H 'accept: text/plain' \
-H 'Content-Type: application/json' \
-d '{
  "id": 0,
  "nome": "DevViniNine",
  "email": "devvininine@gmail.com",
  "password": "Admin123",
  "admin": 1
}'
```

Request URL

http://localhost:5127/api/Usuario/cadastrar

Server response

Code	Details
200	Response body <pre>{ "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjUwMjUiLCJ1bWFpbCI6ImRldnZpbmluaW5lQGdtYWlsLmNvbSIsp0aSI6ImY2ZDE0NTFjLTA1MzAtNGFhMy05NGExLTliYmE0ZGU4ZTA1ZSIsImV4cCI6MTc0ODE5NjkxMCwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdCisImF1ZCI6Imh0dHA6Ly9sb2Nhbgvhc3QifQ.OcFXC87jzIelwjYxp1c70Vfji4NSJaVgaRo1EWPyNKA" }</pre> Response headers <pre>content-type: application/json; charset=utf-8 date: Sun, 25 May 2025 17:15:09 GMT server: Kestrel transfer-encoding: chunked</pre>

Figura 4 – Execução do endpoint POST /api/Usuario/cadastrar

A imagem demonstra a execução do endpoint responsável pelo cadastro de um novo usuário. Os dados foram enviados no corpo da requisição no formato JSON, incluindo nome, e-mail, senha e o nível de permissão (admin = 1). Após o envio, a API retornou com sucesso (HTTP 200) um token de autenticação JWT, que poderá ser utilizado para autorizar futuras requisições do usuário recém-cadastrado.

The screenshot shows a REST API endpoint for fetching user data. The URL is `GET /api/Usuario/email/{email}`. The parameters section shows a required parameter `email` with value `admin@admin.com`. The responses section shows a 200 OK status with a JSON response body containing user data and response headers.

```

{
  "id": 18,
  "name": "admin",
  "email": "admin@admin.com",
  "password": null,
  "admin": 0
}
  
```

Responses

Server response

Code	Details
200	Response body <pre> { "id": 18, "name": "admin", "email": "admin@admin.com", "password": null, "admin": 0 } </pre> Response headers <pre> content-type: application/json; charset=utf-8 date: Sun, 25 May 2025 17:35:29 GMT server: Kestrel transfer-encoding: chunked </pre>

Figura 5 – Execução do endpoint GET /api/Usuario/email/{email}

A imagem exibe a execução do endpoint que busca os dados de um usuário a partir de seu e-mail. Neste exemplo, foi informado o e-mail **admin@admin.com**, e a resposta retornou um objeto JSON contendo o ID do usuário, nome, e-mail e o valor null para a senha (por segurança), além da indicação de que o usuário não possui privilégios administrativos (**admin: 0**).

The screenshot shows the Swagger UI interface for the `GET /api/Usuario/{id}` endpoint. At the top, the method `GET` and the URL `/api/Usuario/{id}` are specified. Below this, the **Parameters** section shows a required parameter `id` of type `integer($int32)` with a description of `(path)`. The **Responses** section shows a 200 OK response with a media type of `text/plain`, which is highlighted with a green border. An example value is provided as a JSON object:

```
{
  "id": 0,
  "nome": "string",
  "email": "string",
  "password": "string",
  "admin": 0
}
```

Figura 6 – Endpoint GET /api/Usuario/{id}

Este endpoint permite consultar os dados de um usuário a partir do seu identificador numérico (ID). O ID é passado como parâmetro na URL, e a API retorna as informações correspondentes ao usuário, como nome, e-mail, senha e status de administrador.

The screenshot shows the Swagger UI interface for the `GET /api/Usuario/listar` endpoint. At the top, the method `GET` and the URL `/api/Usuario/listar` are specified. Below this, the **Parameters** section indicates **No parameters**. The **Responses** section shows a 200 OK response with a media type of `text/plain`, which is highlighted with a green border. An example value is provided as a JSON array:

```
[
  {
    "id": 0,
    "nome": "string",
    "email": "string",
    "password": "string",
    "admin": 0
  }
]
```

Figura 7 – Endpoint GET /api/Usuario/listar

Este endpoint retorna a lista completa de usuários cadastrados na plataforma. A resposta contém um array de objetos JSON com os dados de cada usuário, incluindo id, nome, e-mail, senha e status de administrador.

PUT /api/Usuario/alterar/{id}

Parameters

Name	Description
id * required	integer(\$int32) (path)

Request body

Example Value | Schema

```
{
  "id": 0,
  "name": "string",
  "email": "string",
  "password": "string",
  "admin": 0
}
```

Responses

Code	Description	Links
200	OK	No links

Media type

text/plain

Controls Accept header.

Example Value | Schema

```
{
  "id": 0,
  "name": "string",
  "email": "string",
  "password": "string",
  "admin": 0
}
```

Figura 8 – Endpoint PUT /api/Usuario/alterar/{id}

Este endpoint permite atualizar os dados de um usuário existente. O identificador do usuário é informado na URL e os novos dados são enviados no corpo da requisição em formato JSON. A resposta exibe os dados atualizados do usuário.

POST /api/Usuario/login

Parameters

No parameters

Request body

application/json

Example Value | Schema

```
{
  "email": "user@example.com",
  "password": "string"
}
```

Responses

Code	Description	Links
200	OK	No links

Media type

text/plain

Controls Accept header.

Example Value | Schema

```
{
  "token": "string"
}
```

Figura 9 – Endpoint POST /api/Usuario/login

Este endpoint realiza o processo de autenticação de um usuário. Os dados de login, compostos por e-mail e senha, são enviados no corpo da requisição em

formato JSON. Se as credenciais estiverem corretas, a API retorna um token de autenticação JWT para uso nas próximas requisições protegidas

The screenshot shows a REST API documentation page for the `DELETE /api/Usuario/deletar/{id}` endpoint. The top navigation bar has a lock icon and an upward arrow icon. Below the URL, there is a "Parameters" section with a table:

Name	Description
id * required	integer(\$int32) id (path)

Below the parameters is a "Responses" section with a table:

Code	Description	Links
200	OK Media type <code>text/plain</code>	No links

Under the "text/plain" media type, there is a note: "Controls Accept header." followed by "Example Value | Schema". The schema is shown as a JSON object:

```
{
  "id": 0,
  "name": "string",
  "email": "string",
  "password": "string",
  "admin": 0
}
```

Figura 10 – Endpoint DELETE /api/Usuario/deletar/{id}

Este endpoint permite excluir um usuário do sistema a partir de seu identificador (ID). O ID é informado como parâmetro na URL e, após a exclusão, os dados do usuário removido são retornados na resposta.

caso o usuário esteja vinculado a dados de curtida, visualização ou tenha playlists criadas a exclusão será impedida.

5.2. ENDPOINTS DA ENTIDADE CONTEUDO

A seguir, são apresentados os endpoints responsáveis pelo gerenciamento de conteúdos na plataforma, que incluem arquivos de vídeo, música, imagem e podcast. Esses recursos são fundamentais para o funcionamento do sistema de streaming, permitindo o cadastro, listagem, alteração e exclusão dos conteúdos disponibilizados por criadores. Além disso, há um endpoint de upload, incluído apenas como elemento visual, já que o sistema não possui um repositório de armazenamento real configurado.

Cada operação é acessada por meio de uma rota específica na API, com métodos HTTP apropriados (GET, POST, PUT e DELETE), garantindo a integridade e o controle das informações manipuladas.

The screenshot shows a POST request to the endpoint `/api/Conteudo/cadastrar`. The 'Parameters' section indicates 'No parameters'. The 'Request body' is set to `application/json`, and its example value is:

```
{
  "id": 0,
  "nome": "string",
  "tipo": "string",
  "url": "string",
  "nomeCriador": "string"
}
```

The 'Responses' section shows a successful response (200 OK) with the message 'OK'.

Figura 11 – Endpoint POST /api/Conteudo/cadastrar

Este endpoint permite cadastrar um novo conteúdo na plataforma. Os dados são enviados no corpo da requisição em formato JSON e devem incluir o nome do conteúdo, tipo (como vídeo, música, imagem ou podcast), URL do arquivo e o nome do criador. A resposta indica o sucesso da operação com o código HTTP 200.

The screenshot shows a DELETE request to the endpoint `/api/Conteudo/deletar/{id}`. The 'Parameters' section includes a required parameter `id` of type integer(\$int32) with a value of 2040. The 'Responses' section shows a successful response (200 OK) with the message 'Content deleted successfully!'.

Figura 12 – Execução do endpoint DELETE /api/Conteudo/deletar/{id}

A imagem apresenta a execução do endpoint utilizado para excluir um conteúdo da plataforma, a partir de seu identificador. Neste exemplo, foi informado o ID 2040, e a resposta da API confirmou a operação com a mensagem "Conteúdo deletado com sucesso!", juntamente com o status HTTP 200.

The screenshot shows a REST API testing interface. At the top, it displays a blue bar with 'GET' and the endpoint '/api/Conteudo/listar'. Below this, there's a 'Parameters' section with a 'No parameters' message, an 'Execute' button, and a 'Cancel' button. Under the 'Responses' section, there's a 'Curl' block containing a command to run a curl request to the specified URL, and a 'Request URL' block showing the full URL. In the 'Server response' section, there are tabs for 'Code' (set to 200) and 'Details'. The 'Code' tab shows the status code 200, and the 'Details' tab shows the response body, which is a JSON array with one item. The JSON data is as follows:

```
[
  {
    "id": 1016,
    "nome": "ASIAN KUNG-FU GENERATION - After Dark (Video Clip)",
    "tipo": "Video",
    "url": "https://github.com/DevViniNine/Unip-PIM-VIII/raw/refs/heads/main/Conteudos/Videos/ASIAN%20KUNG-FU%20GENERATION%20-%20After%20Dark%20(Video%20Clip).mp4",
    "nomeCriador": "Asian Kung Fu"
  }
]
```

Figura 13 – Execução do endpoint GET /api/Conteudo/listar

A imagem apresenta a execução do endpoint responsável por listar todos os conteúdos cadastrados na plataforma. A requisição foi realizada com sucesso e retornou um array de objetos JSON, contendo informações como ID, nome do conteúdo, tipo (ex: vídeo), URL do arquivo hospedado (neste caso no GitHub) e nome do criador. Esse recurso é essencial para exibir o acervo multimídia disponível aos usuários.

POST /api/Conteudo/upload

Parameters

No parameters

Request body

arquivo * required
string(\$binary)
nome * required
string
tipo * required
string

Escolher Arquivo streaming-de-video.png

Streaming de video icone

Imagen

Responses

Curl

```
curl -X 'POST' \
'http://localhost:5127/api/Conteudo/upload' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC' \
-H 'Content-Type: multipart/form-data' \
-F 'arquivo=@streaming-de-video.png;type=image/png' \
-F 'nome=Streaming de video icone' \
-F 'tipo=Imagen'
```

Request URL

<http://localhost:5127/api/contenudo/upload>

Server response

Code	Details
200	Response body

```
{
  "mensagem": "Upload realizado com sucesso!",
  "conteudo": {
    "id": 2041,
    "nome": "Streaming de video icone",
    "tipo": "Imagen",
    "url": "http://192.168.1.11:5127/uploads/imagem/23db02e0-a7de-44b7-a30b-8424d2f3252a.png",
    "criadorID": 1,
    "criador": {
      "id": 1,
      "nome": "admin",
      "conteudos": [
        null
      ]
    }
  }
}
```

Figura 14 – Execução do endpoint POST /api/Conteudo/upload

A imagem mostra a execução do endpoint destinado ao upload de arquivos multimídia na plataforma. Contendo o arquivo **streaming-de-video.png**, o nome “Streaming de vídeo icone” e o tipo “**Imagen**”. A API respondeu com sucesso (HTTP 200), retornando os dados do conteúdo armazenado, incluindo a URL gerada para acesso e as informações do criador responsável pela submissão.

The screenshot shows a REST API endpoint configuration for a PUT request to `/api/Conteudo/alterar/{id}`. The interface includes fields for parameters, request body, and responses.

Parameters:

Name	Description
id * required integer(\$int32) (path)	2041

Request body: application/json

Example Value | Schema:

```
{  
  "id": 2041,  
  "nome": "string",  
  "tipo": "string",  
  "url": "string",  
  "nomeCriador": "string"  
}
```

Responses:

Code	Description	Links
200	OK	No links

Figura 15 – Execução do endpoint PUT /api/Conteudo/alterar/{id}

Esta imagem apresenta a execução do endpoint responsável por atualizar os dados de um conteúdo existente. O identificador do conteúdo (neste caso, 2041) é passado na URL, e os novos dados são enviados no corpo da requisição em formato JSON. A operação é concluída com sucesso, retornando o status HTTP 200.

5.3. ENDPOINTS DA ENTIDADE CRIADOR

Nesta seção são apresentados os endpoints responsáveis pela gestão dos criadores de conteúdo da plataforma. Esses usuários têm permissão para cadastrar conteúdos multimídia, como vídeos, músicas e imagens. Os endpoints disponíveis permitem realizar o cadastro, listagem, alteração e exclusão de criadores.

As operações seguem o padrão REST e utilizam os métodos HTTP adequados (POST, GET, PUT e DELETE), garantindo uma comunicação eficiente com a API. A seguir, são apresentados exemplos reais de requisições feitas por meio da interface Swagger.

The screenshot shows the Swagger UI interface for a POST request to the endpoint `/api/Criador/cadastrar`. The request body is a JSON object with the key `"nome": "DevViniNine"`. The response shows a successful 200 status with the message `"Criador cadastrado com sucesso."` and the created creator object, which includes `"id": 2017`, `"nome": "DevViniNine"`, and `"conteudos": null`.

```

curl -X 'POST' \
'http://localhost:5127/api/Criador/cadastrar' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC\
-H 'Content-Type: application/json' \
-d '{
  "nome": "DevViniNine"
}'

```

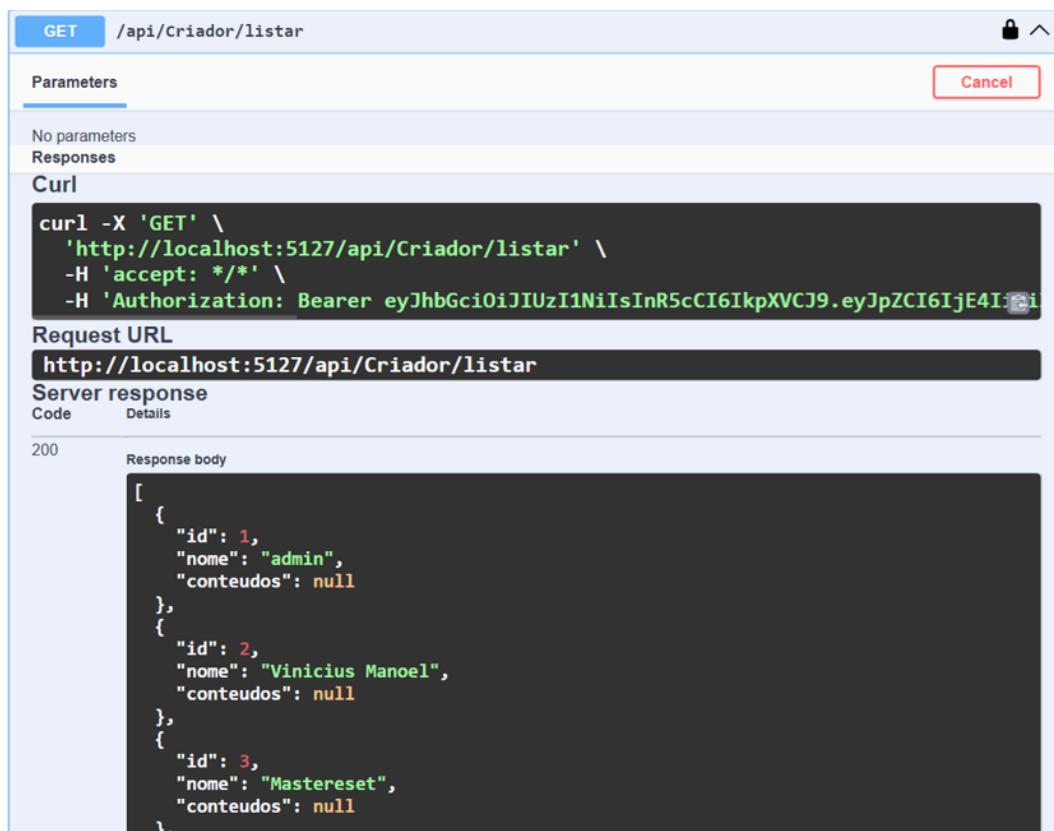
Request URL: `http://localhost:5127/api/Criador/cadastrar`

Code	Details
200	Response body <code>{ "mensagem": "Criador cadastrado com sucesso.", "criador": { "id": 2017, "nome": "DevViniNine", "conteudos": null } }</code>

Figura 16– Execução do endpoint POST /api/Criador/cadastrar

A imagem mostra a execução do endpoint responsável por cadastrar um novo criador de conteúdo. Foi enviado um corpo JSON com o nome "**DevViniNine**", e a resposta da API confirmou o sucesso da operação com a mensagem "**Criador cadastrado com sucesso.**". A resposta também retorna o objeto do criador

recém-cadastrado, incluindo seu identificador (**id: 2017**) e o campo de conteúdos vinculado, que está inicialmente como **null**.



```

GET /api/Criador/listar
curl -X 'GET' \
  'http://localhost:5127/api/Criador/listar' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4IiwiYXV0aF9pZCI6IjEwMTciLCJleHAiOjE2NjQyMjUxOTB9.eyJpZCI6IjE4IiwiYXV0aF9pZCI6IjEwMTciLCJleHAiOjE2NjQyMjUxOTB9'
Request URL
http://localhost:5127/api/Criador/listar
Server response
Code Details
200 Response body
[
  {
    "id": 1,
    "nome": "admin",
    "conteudos": null
  },
  {
    "id": 2,
    "nome": "Vinicius Manoel",
    "conteudos": null
  },
  {
    "id": 3,
    "nome": "Mastereset",
    "conteudos": null
  }
]

```

Figura 17 – Execução do endpoint GET /api/Criador/listar

A imagem mostra a execução do endpoint responsável por listar todos os criadores de conteúdo cadastrados na plataforma. A resposta da API, com status HTTP 200, retorna um array de objetos contendo o ID, Entre os criadores listados, estão nomes como “admin”, “Vinicius Manoel” e “Mastereset”.

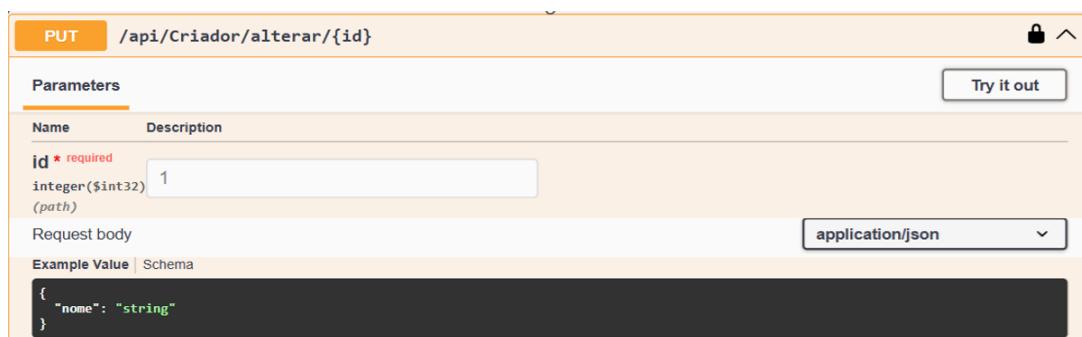


Figura 18 – Endpoint PUT /api/Criador/alterar/{id}

Este endpoint permite atualizar o nome de um criador de conteúdo já cadastrado. O ID do criador é informado como parâmetro na URL e o novo nome é enviado no corpo da requisição em formato JSON. Ao ser executada corretamente, a API retorna um status HTTP 200, confirmando a alteração dos dados.

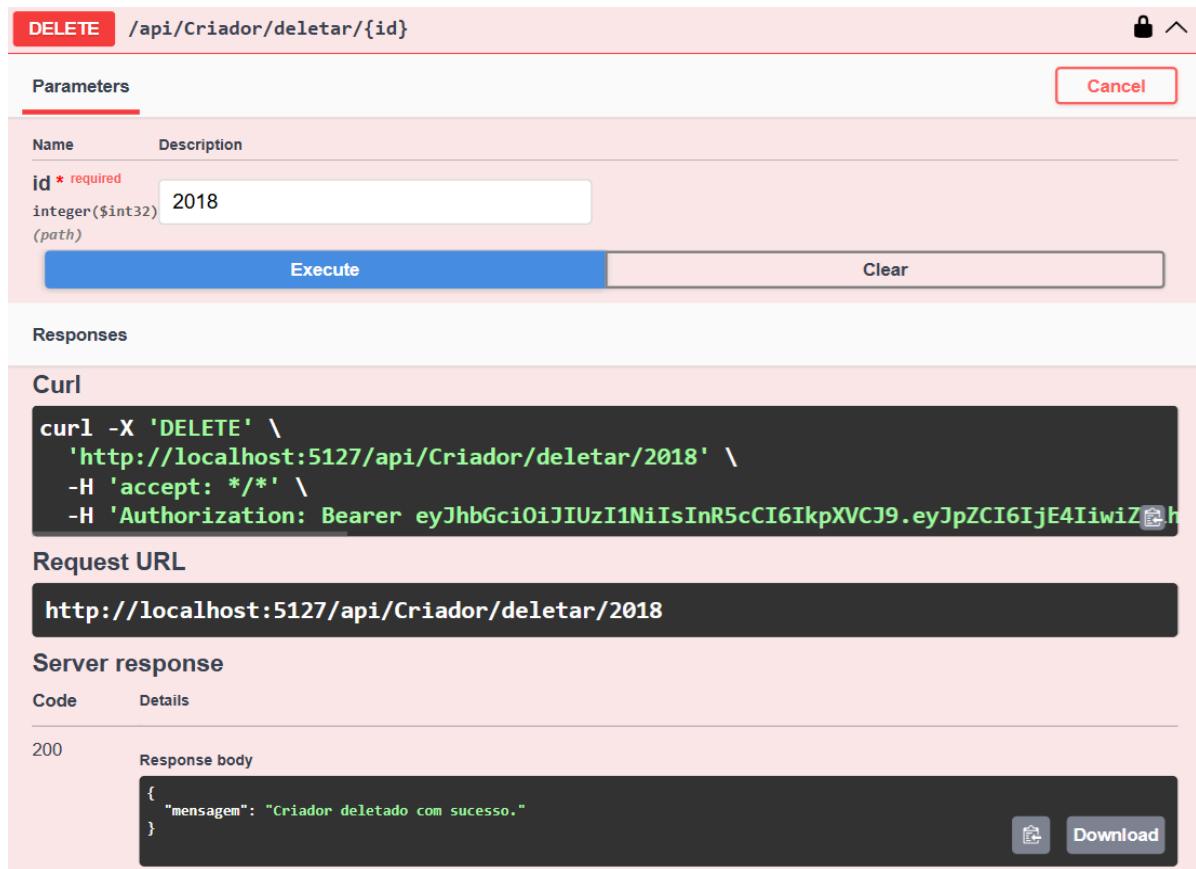


Figura 19 – Execução do endpoint DELETE /api/Criador/deletar/{id}

A imagem mostra a execução do endpoint utilizado para excluir um criador de conteúdo. O ID **2018** foi informado na URL e, ao processar a requisição, a API retornou uma resposta com status HTTP 200 e a mensagem "**Criador deletado com sucesso!**", confirmando a operação.

5.4. ENDPOINTS DA ENTIDADE CURTIDA

Nesta seção são apresentados os endpoints relacionados ao sistema de curtidas da plataforma, que permite aos usuários interagirem com os conteúdos por

meio de avaliações positivas. A funcionalidade de curtida é essencial para indicar popularidade, preferências e engajamento com vídeos, músicas, imagens ou podcasts.

Os endpoints disponíveis permitem curtir e descurtir um conteúdo, verificar se um usuário já curtiu determinado item, listar todas as curtidas de um conteúdo e também recuperar todos os conteúdos curtidos por um usuário. Essas operações são executadas utilizando os métodos HTTP adequados (POST, GET e DELETE), seguindo os princípios REST.

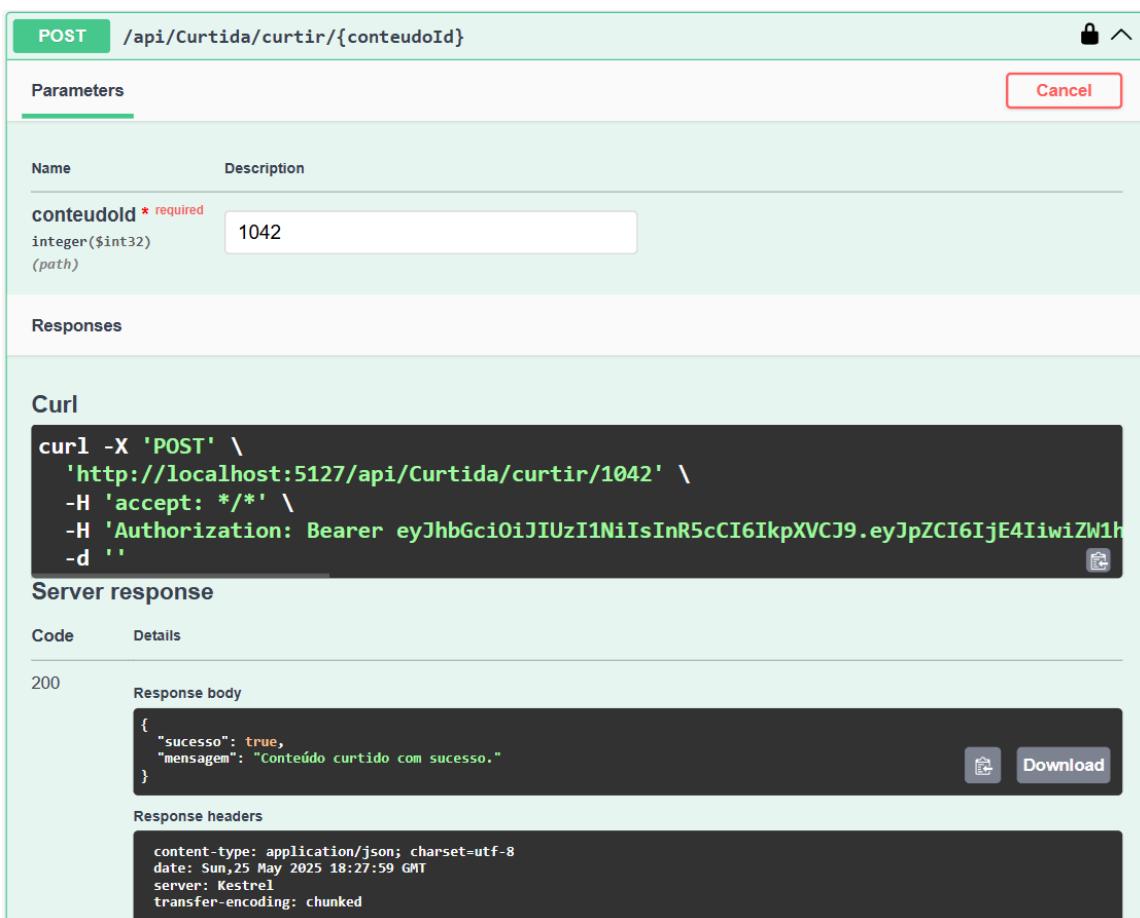


Figura 20 – Execução do endpoint POST /api/Curtida/curtir/{conteudoid}

A imagem apresenta a execução do endpoint responsável por registrar uma curtida em um conteúdo. O identificador do conteúdo (neste exemplo, **1042**) é informado na URL, e a requisição é autenticada por meio de token JWT. Caso o

conteúdo ainda não tenha sido curtido pelo usuário, a API retorna uma resposta de sucesso ("**Conteúdo curtido com sucesso.**").

Se o conteúdo já tiver sido curtido anteriormente, a API responde com sucesso: `false` e a mensagem "**Usuário já curtiu esse conteúdo**", evitando duplicidade de interações.

GET /api/Curtida/existe

Parameters

Name	Description
usuarioid	integer(\$int32) (query)
conteudolid	integer(\$int32) (query)

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5127/api/Curtida/existe?usuarioid=18&conteudolid=1016' \
  -H 'accept: text/plain' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4IiwiZH...' --compressed
```

Request URL

```
http://localhost:5127/api/Curtida/existe?usuarioid=18&conteudolid=1016
```

Server response

Code	Details
200	Response body <code>true</code>

Download

Figura 21 – Execução do endpoint GET /api/Curtida/existe

Este endpoint verifica se um determinado usuário já curtiu um conteúdo específico. Os parâmetros **usuarioid** e **conteudolid** são informados na URL como query string. No exemplo exibido, a API retorna **true**, indicando que o usuário de ID **18** curtiu o conteúdo de ID **1016**.

Caso a curtida não exista, o retorno será **false**. Esse recurso é essencial para a interface do aplicativo mobile, pois permite alternar dinamicamente entre os

botões "Curtir" e "Descurtir", garantindo uma experiência de interação adequada ao estado atual da curtida.

The screenshot shows a REST API endpoint configuration for a DELETE request to `/api/Curtida/descurtir/{conteúdoId}`. The endpoint has one required parameter, `conteúdoId`, which is an integer (int32) path parameter with the value `2054`. Below the endpoint details, there is a "Responses" section showing a "Curl" example and a "Request URL". The "Server response" section shows a successful HTTP 200 status with a JSON response body containing the message `"sucesso": true, "mensagem": "Curtida removida com sucesso."`.

```

DELETE /api/Curtida/descurtir/{conteúdoId}
Parameters
Name Description
conteúdoId * required
integer($int32)
(path)
2054
Responses
Curl
curl -X 'DELETE' \
  'http://localhost:5127/api/Curtida/descurtir/2054' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4IiwidHlwIiwi
Request URL
http://localhost:5127/api/Curtida/descurtir/2054
Server response
Code Details
200 Response body
{
  "sucesso": true,
  "mensagem": "Curtida removida com sucesso."
}

```

Figura 22 – Execução do endpoint `DELETE /api/Curtida/descurtir/{conteúdoId}`

A imagem mostra a execução do endpoint utilizado para remover uma curtida previamente registrada por um usuário. O identificador do conteúdo é informado como parâmetro na URL (neste exemplo, **2054**). A API retorna status HTTP 200 juntamente com a mensagem **"Curtida removida com sucesso."**, indicando que a operação foi concluída corretamente.

GET /api/Curtida/listar/{conteudoId}

Parameters

Name	Description
conteudoid * required	integer(\$int32) (path)
1016	

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5127/api/Curtida/listar/1016' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4IiwidjI0IjoiMSJ9.ewJyL2FzY2lzb2xvZ2F0aW9ucy5jb20='
```

Request URL

```
http://localhost:5127/api/Curtida/listar/1016
```

Server response

Code	Details
200	Response body <pre>{ "totalCurtidas": 2, "usuarios": [{ "userId": 2023, "conteudoId": 1016 }, { "userId": 18, "conteudoId": 1016 }] }</pre>

Figura 23 – Execução do endpoint GET /api/Curtida/listar/{conteudoId}

Este endpoint retorna a quantidade total de curtidas associadas a um conteúdo específico, além da lista de usuários que realizaram a ação. No exemplo, ao consultar o conteúdo de ID **1016**, a API respondeu com **totalCurtidas: 2** e uma lista contendo os IDs dos usuários que curtiram. Esse recurso é especialmente utilizado para alimentar o contador de curtidas exibido na interface do aplicativo ou sistema web.

GET /api/Curtida/curtidos

Cancel

Parameters

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5127/api/Curtida/curtidos' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4I:clip'
```

Request URL

```
http://localhost:5127/api/Curtida/curtidos
```

Server response

Code Details

200 Response body

```
{
  "total": 5,
  "conteudos": [
    {
      "conteudoId": 1042,
      "nome": "Pízza",
      "tipo": "Imagen",
      "url": "https://images.pexels.com/photos/2147491/pexels-photo-2147491.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1",
      "nomeCriador": "pexels.com"
    },
    {
      "conteudoId": 1016,
      "nome": "ASIAN KUNG-FU GENERATION - After Dark (Video Clip)",
      "tipo": "Video",
      "url": "https://github.com/DevViniNine/Unip-PIM-VIII/raw/refs/heads/main/Conteudos/Videos/ASIAN%20KUNG-FU%20GENERATION%20-%20After%20Dark%20(Video%20Clip).mp4",
      "nomeCriador": "Asian Kung Fu"
    }
  ]
}
```

Download

Figura 24 – Execução do endpoint GET /api/Curtida/curtidos

Este endpoint retorna todos os conteúdos que foram curtidos pelo usuário autenticado. A resposta da API inclui um total de curtidas (**"total": 5**) e uma lista de objetos com as informações dos conteúdos, como **conteudoid**, nome, tipo, URL e nome do criador. Esse recurso é utilizado, por exemplo, para exibir a lista personalizada de curtidos do usuário logado no aplicativo.

5.5. ENDPOINTS DA ENTIDADE PLAYLIST

A seção a seguir apresenta os endpoints responsáveis pelo gerenciamento das playlists personalizadas dos usuários. As playlists permitem agrupar conteúdos diversos, como músicas, vídeos, imagens e podcasts.

Os recursos disponíveis incluem a criação, listagem, edição e exclusão de playlists. Para o funcionamento completo deste recurso, é essencial utilizar também os endpoints das entidades **Conteúdo** e **ItemPlaylist**.

POST /api/Playlist/criar

Parameters

No parameters

Request body

Example Value | Schema

```
{  
  "id": 0,  
  "nome": "Playlist Rock Supremo!",  
  "usuarioId": 0  
}
```

Responses

Curl

```
curl -X 'POST' \  
  'http://localhost:5127/api/Playlist/criar' \  
  -H 'accept: */*' \  
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4Iiwi' \  
  -H 'Content-Type: application/json' \  
  -d '{  
    "id": 0,  
    "nome": "Playlist Rock Supremo!",  
    "usuarioId": 0  
}'
```

Request URL

<http://localhost:5127/api/Playlist/criar>

Server response

Code	Details
200	Response body

```
{  
  "id": 3013,  
  "nome": "Playlist Rock Supremo!",  
  "usuarioId": 18  
}
```

Try it out

Reset

application/json

Download

Figura 25 – Execução do endpoint POST /api/Playlist/criar

Este endpoint permite ao usuário criar uma nova playlist na plataforma. No exemplo apresentado, foi enviada uma requisição contendo o nome "**Playlist Rock Supremo!**" e o **usuarioid**. A API processou a solicitação e retornou status HTTP 200, junto com o objeto da playlist criada, contendo o novo **id** e os dados fornecidos. Essa funcionalidade é o ponto de partida para organizar conteúdos, sendo indispensável a utilização conjunta dos endpoints de **Conteúdo** e **ItemPlaylist** para que a playlist cumpra sua função.

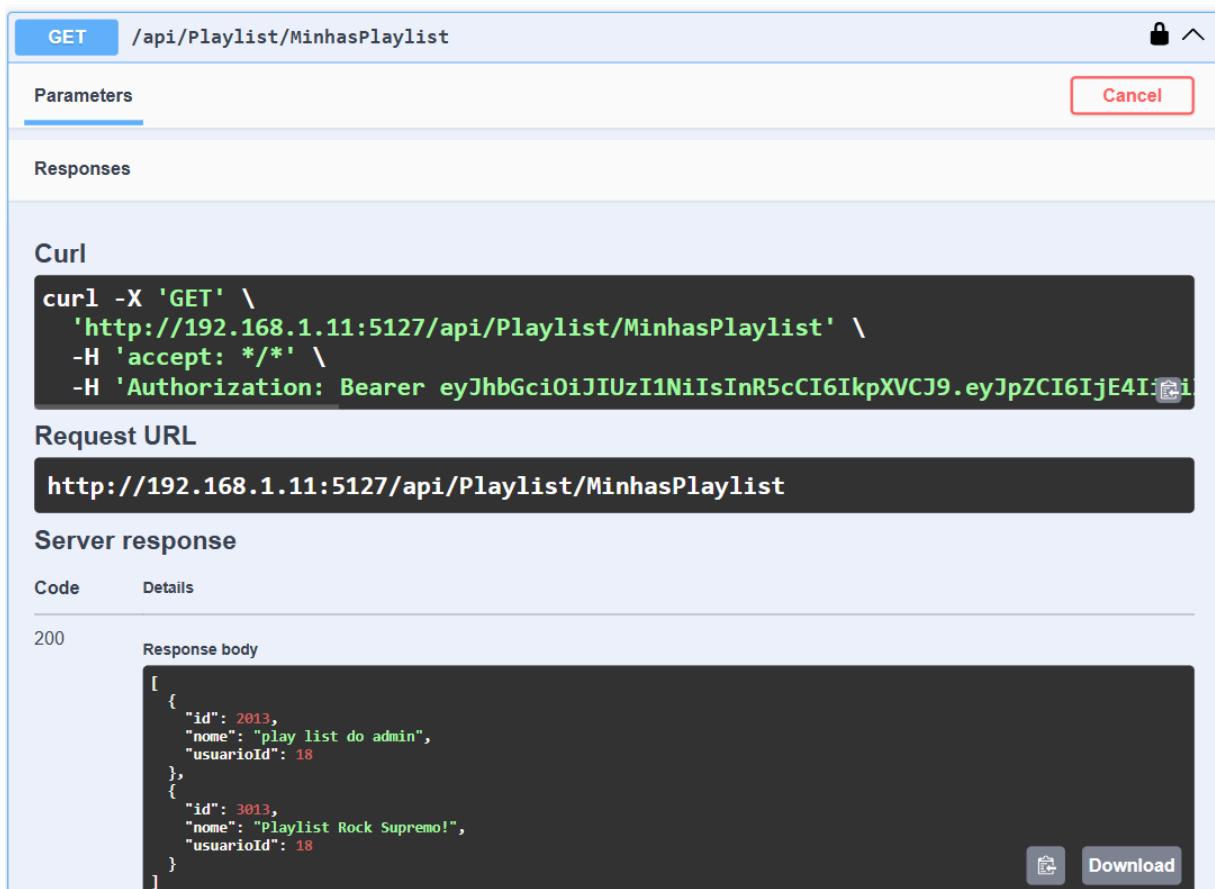


Figura 26 – Execução do endpoint GET /api/Playlist/MinhasPlaylist

Este endpoint retorna todas as playlists associadas ao usuário logado. A autenticação é feita via token JWT, e a API responde com um array de objetos JSON contendo o **id**, **nome** e **usuarioId** de cada playlist criada. No exemplo, o usuário de ID **18** possui duas playlists cadastradas: "**play list do admin**" e "**Playlist Rock Supremo!**". Este recurso é fundamental para que o usuário possa visualizar e gerenciar suas próprias coleções de conteúdo.

PUT /api/Playlist/alterar/{id}

Parameters

Name	Description
id * required	integer(\$int32) 3013 (path)

Request body application/json

```
{
  "id": 3013,
  "nome": "Playlist de rock SUper sUPREmo!",
  "usuarioId": 0
}
```

Responses

Curl

```
curl -X 'PUT' \
'http://192.168.1.11:5127/api/Playlist/alterar/3013' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4IiwidWIh
-d '{
  "id": 3013,
  "nome": "Playlist de rock SUper sUPREmo!",
  "usuarioId": 0
}'
```

Request URL

```
http://192.168.1.11:5127/api/Playlist/alterar/3013
```

Server response

Code	Details
200	Response body

```
{
  "id": 3013,
  "nome": "Playlist de rock SUper sUPREmo!",
  "usuarioId": 18
}
```

Download

Figura 27 – Execução do endpoint PUT /api/Playlist/alterar{id}

Este endpoint permite alterar as informações de uma playlist existente. No exemplo exibido, o **id** da playlist **3013** foi passado na URL, e um novo nome foi fornecido no corpo da requisição: "**Playlist de rock SUper sUPREmo!**". A API processou a solicitação e retornou os dados atualizados da playlist, confirmando a modificação com status HTTP 200. Esse recurso é útil para renomear playlists conforme a organização ou preferência do usuário.

DELETE /api/Playlist/deletar/{id}

Cancel

Parameters

Name	Description
id * required integer(\$int32) (path)	3013

Responses

Curl

```
curl -X 'DELETE' \
'http://192.168.1.11:5127/api/Playlist/deletar/3013' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4IiwidHlwIiwi...
```

Request URL

```
http://192.168.1.11:5127/api/Playlist/deletar/3013
```

Server response

Code	Details
200	Response body Playlist deletada com sucesso.

Download

Figura 28 – Execução do endpoint DELETE /api/Playlist/deletar/{id}

Este endpoint é utilizado para excluir uma playlist criada pelo usuário. A operação é realizada informando o identificador da playlist na URL (neste exemplo, **3013**). A resposta da API indica sucesso com o status HTTP 200 e a mensagem **"Playlist deletada com sucesso."**.

Contudo, caso existam conteúdos associados à playlist por meio da entidade **ItemPlaylist**, a exclusão não será permitida. Para que a operação seja concluída corretamente, é necessário primeiro **remover ou desvincular todos os itens da playlist**, utilizando os endpoints adequados da entidade **ItemPlaylist**.

5.6. ENDPOINTS DA ENTIDADE ITEMPLAYLIST

A entidade *ItemPlaylist* tem como função estabelecer o relacionamento entre os conteúdos cadastrados na plataforma e as playlists criadas pelos usuários. Por meio desses endpoints, é possível adicionar conteúdos a uma determinada playlist, além de listar os itens associados.

Durante a fase atual do desenvolvimento, foram implementadas as operações de inserção de novos itens em playlists e de listagem geral ou específica por *playlistId*. No entanto, destaca-se que **não foi desenvolvido, até o momento, um endpoint para exclusão de itens da playlist**, impossibilitando a remoção individual de conteúdos associados. Essa limitação será considerada em versões futuras do sistema, com o objetivo de garantir maior flexibilidade na gestão de playlists por parte do usuário.

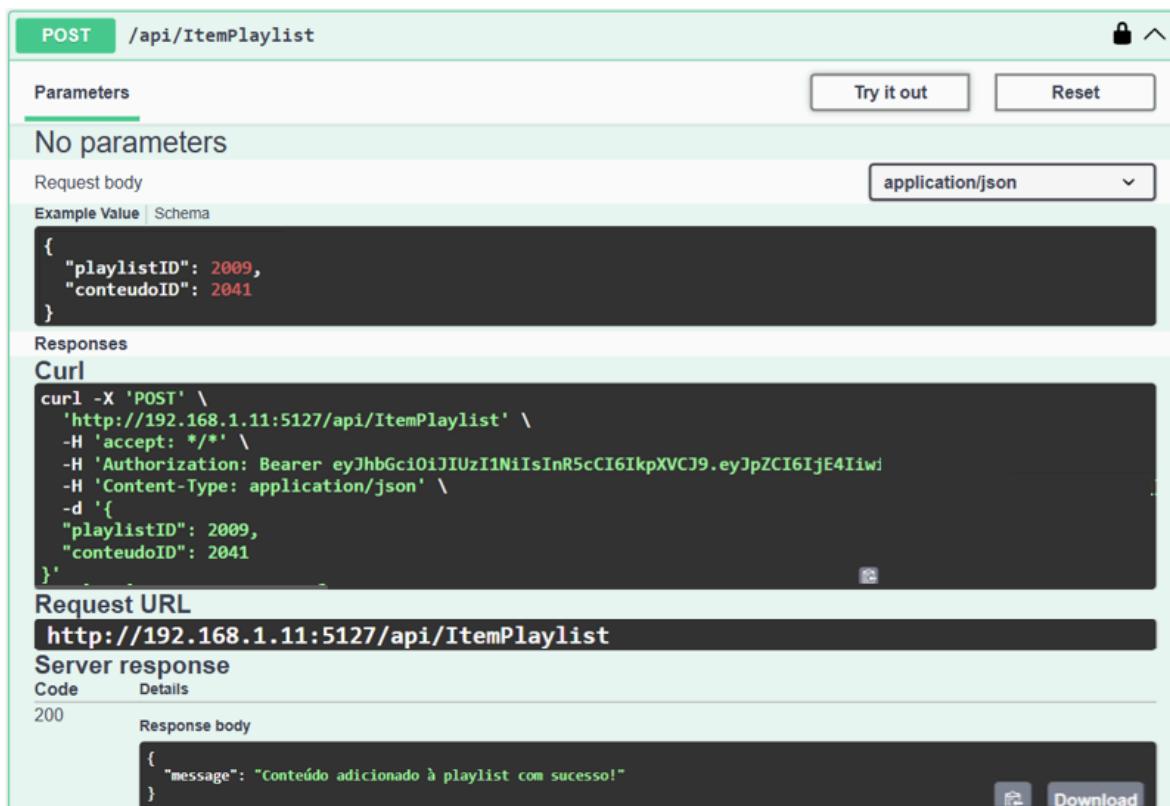


Figura 29 – Execução do endpoint POST /api/ItemPlaylist

O endpoint acima é responsável por associar um conteúdo previamente cadastrado a uma playlist existente. Para isso, o corpo da requisição deve conter os

identificadores da playlist (**playlistID**) e do conteúdo (**conteudoID**), conforme ilustrado no exemplo.

Ao realizar a operação com sucesso, a API retorna o status HTTP 200 e a mensagem "**Conteúdo adicionado à playlist com sucesso!**". No entanto, o sistema realiza uma verificação interna para evitar duplicidade. Caso o conteúdo já esteja vinculado à playlist informada, a API retorna a mensagem "**Este conteúdo já está na playlist**", impedindo que o item seja adicionado novamente.

The screenshot shows the execution of the `GET /api/ItemPlaylist` endpoint. The interface includes:

- Parameters:** No parameters.
- Responses:** A red "Cancel" button.
- Curl:**

```
curl -X 'GET' \
'http://192.168.1.11:5127/api/ItemPlaylist' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4Iiwi...'
```
- Request URL:** `http://192.168.1.11:5127/api/ItemPlaylist`
- Server response:**

Code	Details
200	Response body: <pre>[{"playlistID": 1008, "conteudoID": 1016}, {"playlistID": 2009, "conteudoID": 1021}, {"playlistID": 2009, "conteudoID": 1016}, {"playlistID": 2009, "conteudoID": 1028}, {"playlistID": 2012, "conteudoID": 1031}]</pre>
- Buttons:** Download, Print.

Figura 30 – Execução do endpoint GET /api/ItemPlaylist

Este endpoint tem como finalidade listar todas as associações existentes entre conteúdos e playlists no sistema. A resposta da API é composta por uma

coleção de objetos contendo os identificadores **playlistID** e **conteudoid**, representando o vínculo entre cada item e sua respectiva playlist.

Tal funcionalidade é útil principalmente para fins administrativos, como auditoria e conferência geral das relações entre playlists e conteúdos cadastrados, permitindo uma visualização completa de todas as conexões existentes na base de dados.

The screenshot shows the execution of a GET request to the endpoint `/api/ItemPlaylist/{playlistId}/conteudos`. The parameter `playlistId` is set to `1008`. The request URL is `http://192.168.1.11:5127/api/ItemPlaylist/1008/conteudos`. The response body is a JSON array containing one item:

```
[
  {
    "id": 0,
    "nome": "ASIAN KUNG-FU GENERATION - After Dark (Video Clip)",
    "tipo": "Video",
    "url": "https://github.com/DevViniNine/Unip-PIM-VIII/raw/refs/heads/main/Conteudos/Videos/ASIAN%20KUNG-FU%20GENERATION%20-%20After%20Dark%20(Video%20Clip).mp4",
    "nomeCriador": "Asian Kung Fu"
  }
]
```

Figura 31 – Execução do endpoint GET `/api/ItemPlaylist/{playlistId}/conteudos`

Este endpoint retorna todos os conteúdos associados a uma playlist específica, identificada pelo parâmetro **playlistId** na URL. A resposta consiste em um array de objetos JSON, cada um contendo informações detalhadas do conteúdo relacionado, como **id**, **nome**, **tipo**, **url** e **nomeCriador**.

Essa funcionalidade é essencial para a exibição personalizada de playlists no ambiente do usuário, permitindo o carregamento dinâmico dos itens que compõem cada lista. É com base neste endpoint que se possibilita, por exemplo, a reprodução de faixas ou vídeos armazenados em uma mesma playlist dentro do aplicativo.

5.7. ENDPOINTS DA ENTIDADE VISUALIZACAO

A entidade *Visualizacao* tem por finalidade registrar e fornecer informações relacionadas ao acesso dos conteúdos na plataforma. Esse módulo é essencial para o monitoramento do comportamento dos usuários, permitindo a coleta de métricas que orientam a exibição personalizada de conteúdos e o ranqueamento por popularidade.

Os endpoints possuem funcionalidades como o registro de uma nova visualização ao acessar determinado conteúdo, a listagem dos últimos conteúdos visualizados, a contagem total de visualizações por conteúdo e a recuperação dos itens mais populares com base na frequência de acessos.

Essas operações visam ampliar o controle sobre a interação dos usuários com a plataforma, estatísticas e análise de engajamento.

The screenshot shows a REST API interface for a 'Visualizacao' endpoint. The method is 'POST' and the URL is '/api/Visualizacao/registrar/{conteudoid}'. A parameter 'conteudoid' is defined as an integer (\$int32) with a value of 1016. Below the parameters, there's a 'Responses' section labeled 'Curl' containing a 'curl' command to execute the request. The 'Request URL' is listed as 'http://192.168.1.11:5127/api/Visualizacao/registrar/1016'. The 'Server response' section shows a status code of 200 with a JSON response body: { "sucesso": true, "mensagem": "Visualização registrada com sucesso." }. There are 'Download' and 'Copy' buttons next to the response body.

Figura 32 – Execução do endpoint POST /api/Visualizacao/registrar/{conteudoid}

O endpoint acima é utilizado para registrar uma nova visualização em um conteúdo específico. Sempre que um usuário acessa um conteúdo, esse endpoint é acionado para contabilizar a visualização, contribuindo para os dados estatísticos do sistema. Essa funcionalidade é essencial para contador e análise de engajamento e desempenho dos conteúdos disponibilizados na plataforma.

GET /api/Visualizacao/ultimos

Parameters

No parameters

Responses

Curl

```
curl -X 'GET' \
  'http://192.168.1.11:5127/api/Visualizacao/ultimos' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4IiA6ICi
```

Request URL

```
http://192.168.1.11:5127/api/Visualizacao/ultimos
```

Server response

Code	Details
200	Response body <pre>{ "total": 63, "visualizacoes": [{ "usuarioId": 18, "conteudoId": 1016, "nomeConteudo": "ASIAN KUNG-FU GENERATION - After Dark (Video Clip)", "tipoConteudo": "Video", "dataVisualizacao": "2025-05-25T19:51:14.83" }, { "usuarioId": 18, "conteudoId": 1018, "nomeConteudo": "Briga de carneiros impala", "tipoConteudo": "Video", "dataVisualizacao": "2025-05-25T16:02:50.977" }, { "usuarioId": 18, "conteudoId": 1016,</pre>

Download

Figura 33 - Execução do endpoint GET /api/Visualizacao/ultimos

Este **endpoint** tem como finalidade retornar as últimas visualizações registradas para o usuário logado. A resposta da API fornece uma lista cronológica contendo o **identificador do conteúdo**, **nome do conteúdo**, **tipo**, **identificador do usuário** e a **data/hora** em que a visualização ocorreu. Essa funcionalidade é essencial para a implementação de recursos como **histórico de atividades**,

permitindo ao sistema oferecer uma experiência personalizada com base no comportamento recente do usuário.

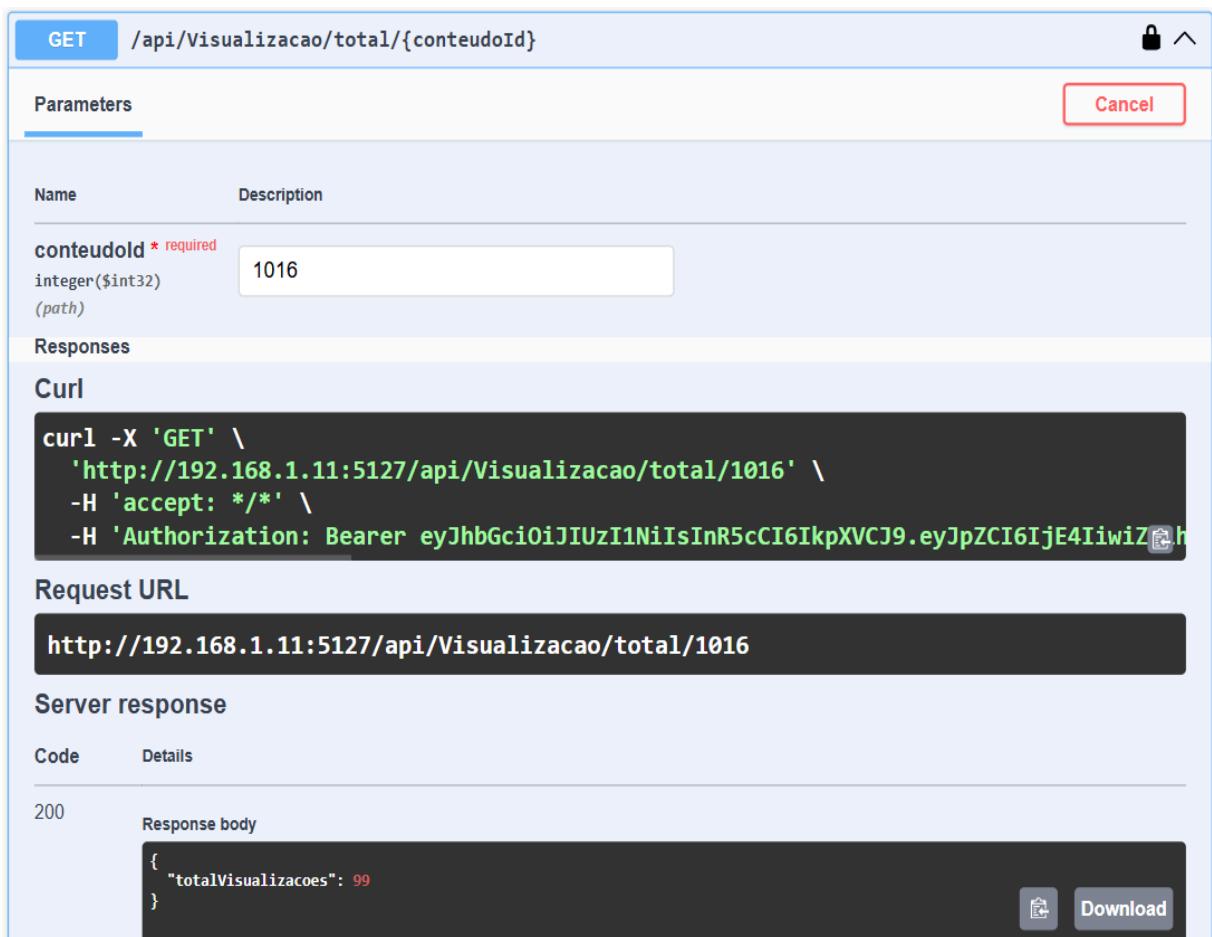


Figura 34 – Execução do endpoint GET /api/Visualizacao/total/{conteudoId}

Este **endpoint** tem como finalidade retornar o número total de visualizações registradas para um conteúdo específico da plataforma. Ao receber uma requisição contendo o **identificador do conteúdo** no caminho da URL, a API realiza uma consulta no banco de dados e contabiliza todas as visualizações associadas àquele conteúdo. A resposta retorna um valor numérico no campo `totalVisualizacoes`, refletindo diretamente o engajamento do público com a mídia. Esse recurso é crucial para a implementação de contadores de **views**, geração de métricas de desempenho e análise de popularidade de conteúdos multimídia publicados na plataforma.

The screenshot shows a REST API endpoint configuration. At the top, it specifies a **GET** method and the URL `/api/Visualizacao/populares`. Below this, there are sections for **Parameters** (No parameters) and **Responses**. Under **Curl**, a command is provided:

```
curl -X 'GET' \
  'http://192.168.1.11:5127/api/Visualizacao/populares' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4IiA='
```

The **Request URL** is listed as `http://192.168.1.11:5127/api/Visualizacao/populares`. The **Server response** section shows a **Code** of 200 and a **Response body** containing the following JSON data:

```
[
  {
    "id": 1016,
    "nome": "ASIAN KUNG-FU GENERATION - After Dark (Video Clip)",
    "tipo": "Video",
    "url": "https://github.com/DevViniNine/Unip-PIM-VIII/raw/refs/heads/main/Conteudos/Videos/ASIAN%20KUNG-FU%20GENERATION%20-%20After%20Dark%20(Video%20Clip).mp4",
    "criador": "Asian Kung Fu",
    "totalVisualizacoes": 99
  },
  {
    "id": 1017,
    "nome": "Briga de carneiros impala",
    "tipo": "Video",
    "url": "https://github.com/DevViniNine/Unip-PIM-VIII/raw/refs/heads/main/Conteudos/Videos/Briga%20de%20carneiros%20impala.mp4",
    "criador": "Vinicius Manoel",
    "totalVisualizacoes": 20
  }
]
```

At the bottom right of the response body area are two buttons: a clipboard icon labeled **Copy** and a **Download** button.

Figura 35 – Execução do endpoint GET /api/Visualizacao/populares

Este **endpoint** tem como finalidade retornar uma lista com os conteúdos mais populares da plataforma, ordenados de acordo com o número total de visualizações registradas. A resposta contém os dados essenciais de cada conteúdo, como nome, tipo, URL, nome do criador e a quantidade total de visualizações.

Esse recurso é especialmente útil para a criação de rankings dinâmicos, sugestões personalizadas de mídia, e para permitir ao usuário o acesso rápido ao que está sendo mais consumido na plataforma. Além disso, é uma ferramenta importante de análise de engajamento, permitindo que criadores de conteúdo acompanhem o alcance de suas publicações de forma objetiva.

5.8. ENDPOINTS DA ENTIDADE COMENTÁRIO

A seção de comentários permite a interação direta dos usuários com os conteúdos disponíveis na plataforma. Com os endpoints apresentados a seguir, é possível registrar novos comentários em conteúdos específicos e também listar todos os comentários já publicados.

Esses endpoints desempenham um papel essencial no aplicativo mobile e na plataforma do criador de conteúdo que irá ser desenvolvida para windows, permitindo que os usuários expressem opiniões e acompanhem os comentários de outras pessoas. A autenticação é obrigatória para comentar, garantindo assim a associação do comentário com o usuário autenticado.

POST /api/Comentario/comentar/{conteudoId}

Parameters

Name	Description
conteudoId <small>* required</small>	integer(\$int32) (path)
1016	

Request body application/json

```
{
  "usuarioId": 0,
  "conteudoId": 1016,
  "usuarioNome": "Admin",
  "texto": "Musica top!",
  "dataComentario": "2025-05-25T20:15:36.183Z"
}
```

Responses

Curl

```
curl -X 'POST' \
'http://192.168.1.11:5127/api/Comentario/comentar/1016' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4IiwidWIi \
-d '{
  "usuarioId": 0,
  "conteudoId": 1016,
  "usuarioNome": "Admin",
  "texto": "Musica top!",
  "dataComentario": "2025-05-25T20:15:36.183Z"
}'
```

Request URL http://192.168.1.11:5127/api/Comentario/comentar/1016

Server response

Code	Details
200	Response body

```
{
  "sucesso": true,
  "mensagem": "Comentado com sucesso!"
}
```

Download

Figura 36 – Execução do endpoint POST /api/Comentario/comentar/{conteudoId}

O **endpoint** apresentado na **figura 36** é utilizado para registrar um novo comentário associado a um conteúdo específico, identificado pelo parâmetro **{conteudoid}** na URL. Para sua execução, é necessário enviar um corpo JSON contendo os dados do usuário que comentou, o texto do comentário e a data/hora da publicação. Esse recurso permite ampliar a interação dos usuários com os conteúdos da plataforma, possibilitando feedback direto, discussão e engajamento. A resposta da API confirma se a operação foi realizada com sucesso, retornando uma mensagem de confirmação em caso positivo.

GET /api/Comentario/listar/{conteudoId}

Cancel

Parameters

Name	Description
conteudoId * required	integer(\$int32) (path)

Responses

Curl

```
curl -X 'GET' \
  'http://192.168.1.11:5127/api/Comentario/listar/1016' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE4Iiwizh...
```

Request URL

<http://192.168.1.11:5127/api/Comentario/listar/1016>

Server response

Code	Details
200	Response body

```
{
  "totalComentarios": 23,
  "comentarios": [
    {
      "usuarioId": 18,
      "conteudoId": 1016,
      "usuarioNome": "admin",
      "texto": "Musica top!",
      "dataComentario": "2025-05-25T17:16:17.213"
    },
    {
      "usuarioId": 3024,
      "conteudoId": 1016,
      "usuarioNome": "Kenshin",
      "texto": "asian toperson",
      "dataComentario": "2025-05-22T20:32:15.317"
    },
    ...
  ]
}
```

Figura 37 – Execução do endpoint GET /api/Comentario/listar/{conteudoid}

O **endpoint** em questão tem como finalidade listar todos os comentários relacionados a um conteúdo específico, identificado pelo parâmetro **{conteudoid}**. A resposta da API fornece uma estrutura contendo o número total de comentários, e uma lista com os dados de cada um: nome do usuário, texto do comentário e data/hora em que foi realizado.

Esse recurso é fundamental para fomentar a interação entre os usuários da plataforma, contribuindo para a construção de um ambiente participativo, onde é possível debater, opinar e compartilhar percepções sobre os conteúdos acessados.

5.9. CONSIDERAÇÕES FINAIS SOBRE A DOCUMENTAÇÃO DOS ENDPOINTS

Com a conclusão desta seção, apresentamos toda a estrutura de endpoints desenvolvida até o momento para a API do sistema de streaming multimídia, devidamente documentada utilizando o Swagger. Cada endpoint possui seus respectivos códigos de resposta HTTP para indicar o resultado das operações, como, por exemplo:

- Erro 200 : requisição realizada com sucesso;
- Erro 400 : Bad Request: erro de validação ou parâmetros inválidos;
- Erro 401 : Unauthorized: quando o token de autenticação está ausente ou inválido;
- Erro 404 : Not Found: recurso não encontrado;
- Erro 500 : Internal Server Error: erro interno do servidor.

Em alguns endpoints, foram implementadas mensagens de erro personalizadas para tornar o retorno mais claro e orientativo ao desenvolvedor ou usuário. Em outros casos, optou-se por não inserir essas mensagens, pois a própria estrutura de erro e os códigos de status já são suficientes para indicar o problema ocorrido.

Vale destacar que esta é apenas uma parte do sistema já funcional. O projeto ainda está em constante desenvolvimento, e novos recursos serão adicionados conforme a evolução do aplicativo. A versão atual é uma base sólida, mas com foco principal na construção da infraestrutura e nos fluxos iniciais. A interface voltada ao criador de conteúdo, por exemplo, encontra-se em estágio inicial, com um esboço básico formado. É previsto o desenvolvimento de novas funcionalidades, melhorias na segurança, validação de dados e, inclusive, a criação do endpoint de remoção de itens da playlist, que ainda não foi implementado nesta etapa.

Com isso, encerramos a parte teórica e de testes dos endpoints. A seguir, no próximo tópico, veremos como esses serviços funcionam na prática com a integração ao **aplicativo mobile**, onde cada funcionalidade poderá ser visualizada em tempo real na experiência do usuário final.

6. ESTRUTURA DO APLICATIVO MOBILE

O aplicativo mobile desenvolvido como parte do sistema de streaming multimídia foi inteiramente construído utilizando **Java** como linguagem principal dentro do ambiente **Android Studio**. Embora os testes práticos tenham sido realizados através do **emulador**, todas as funcionalidades foram pensadas para rodar perfeitamente em **dispositivos físicos**, já que o projeto foi validado com base em um celular real durante o processo de desenvolvimento.

6.1. DEPENDÊNCIAS UTILIZADAS

Durante a implementação do aplicativo, foram utilizadas bibliotecas específicas para garantir uma comunicação eficiente com a API e uma experiência fluida de mídia e navegação. Abaixo, segue uma breve descrição das principais dependências utilizadas no projeto:

Retrofit: biblioteca responsável por simplificar a comunicação HTTP com APIs REST. O Retrofit permite mapear endpoints da API em interfaces Java de forma clara e segura.

Converter-Gson: utilizada para converter automaticamente os dados JSON retornados pela API em objetos Java, facilitando a manipulação e exibição dessas informações no aplicativo.

Media3 ExoPlayer: responsável por executar arquivos de mídia (áudio e vídeo). O ExoPlayer é uma solução poderosa e flexível para reprodução de mídia, oferecendo suporte a streaming adaptativo e controle detalhado do player.

Media3 UI: fornece a interface visual do player de mídia, como botões de play/pause, barra de progresso e opções de controle diretamente integradas à interface do usuário.

6.2. NAVEGAÇÃO E INTERFACE

O aplicativo foi desenvolvido com **navegação entre múltiplas interfaces**, facilitando o acesso a funcionalidades como playlists, favoritos, configurações e perfil. O design foi inspirado em plataformas populares de streaming, como o **YouTube**, especialmente no uso de uma **barra inferior de navegação** que permite alternar entre seções como:

- Início (feed de conteúdos)
- Biblioteca (playlists e curtidos)
- Perfil (perfil do usuário e algumas configurações extra)

Essa abordagem torna a experiência do usuário mais familiar e intuitiva.

6.3. COMPATIBILIDADE E COMUNICAÇÃO

O aplicativo foi pensado para funcionar com a **Sdk 35**, alinhando-se às práticas mais atuais do Android em termos de compatibilidade, segurança e desempenho.

A comunicação com o backend é realizada por meio da API REST, desenvolvida em .NET. Com a devida configuração do Retrofit (definindo o endereço base da API), o aplicativo pode ser utilizado de **qualquer local**, desde que haja conectividade com a internet e que o servidor esteja devidamente configurado com um endereço de acesso válido (IP local, hostname ou domínio público).

Com isso, temos uma aplicação robusta, modular e escalável, capaz de consumir os recursos desenvolvidos na Web API. A seguir, apresentaremos capturas

de tela de cada etapa do aplicativo mobile, desde o login até o consumo das funcionalidades, demonstrando na prática o funcionamento do sistema e a integração efetiva entre frontend e backend.

6.4. APRESENTAÇÃO DAS TELAS DO APLICATIVO MOBILE

A seguir, serão apresentadas as telas do aplicativo mobile desenvolvido para consumo da API de streaming multimídia. Os prints foram obtidos durante o processo de desenvolvimento, utilizando o emulador do Android Studio.

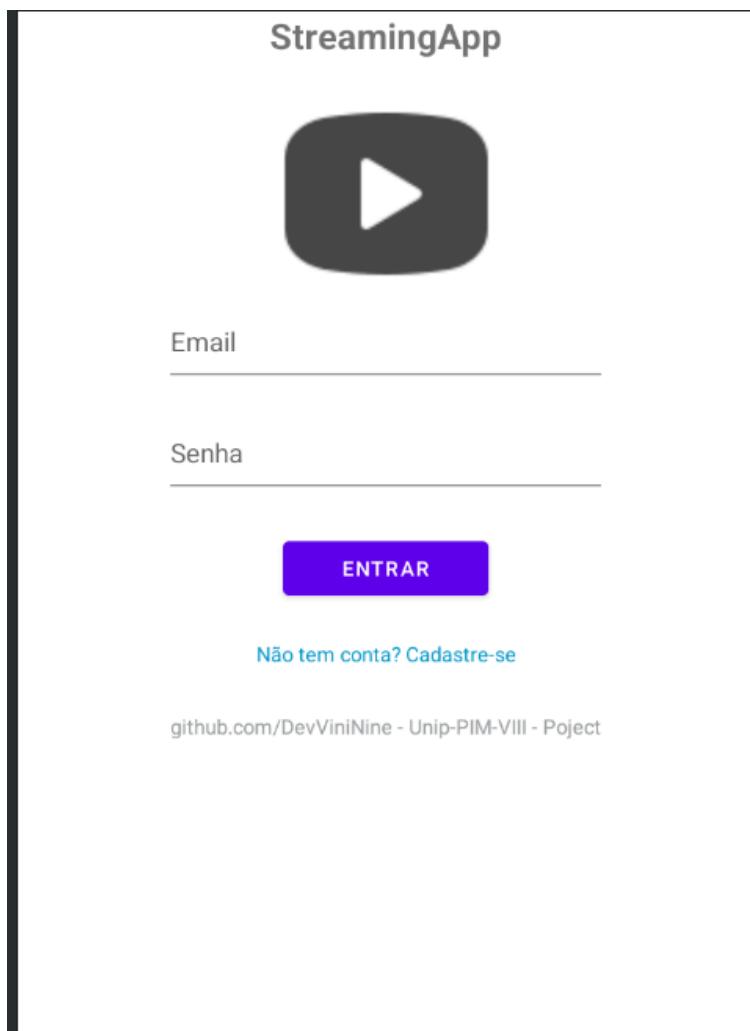


Figura 38 – Tela de Login

A tela de login é a primeira interface exibida ao iniciar o aplicativo. Sua função principal é validar as credenciais de acesso do usuário por meio da

comunicação com a API desenvolvida. Ao informar o e-mail e a senha, o aplicativo realiza uma requisição de autenticação; caso os dados estejam corretos, o usuário é autenticado com sucesso e o token JWT gerado é armazenado localmente no dispositivo. Esse token é utilizado automaticamente em futuras requisições que exigem autenticação.

Se o e-mail informado não estiver cadastrado ou a senha estiver incorreta, o aplicativo retorna uma mensagem informativa ao usuário, indicando a falha na autenticação. Além disso, a interface também disponibiliza a opção “Não tem conta? Cadastre-se”, permitindo que novos usuários realizem o auto cadastro diretamente no aplicativo. Na figura 39 temos o fluxograma desse processo.

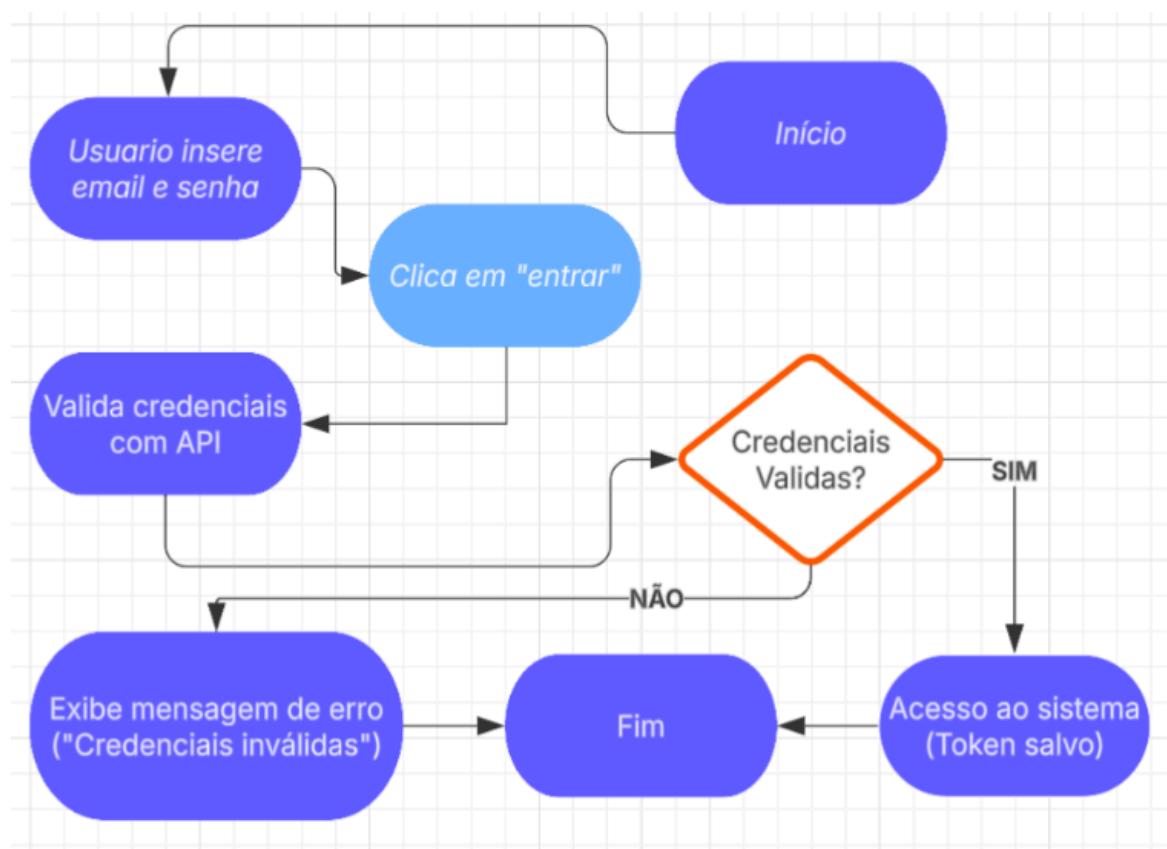


Figura 39 – Fluxograma de Login

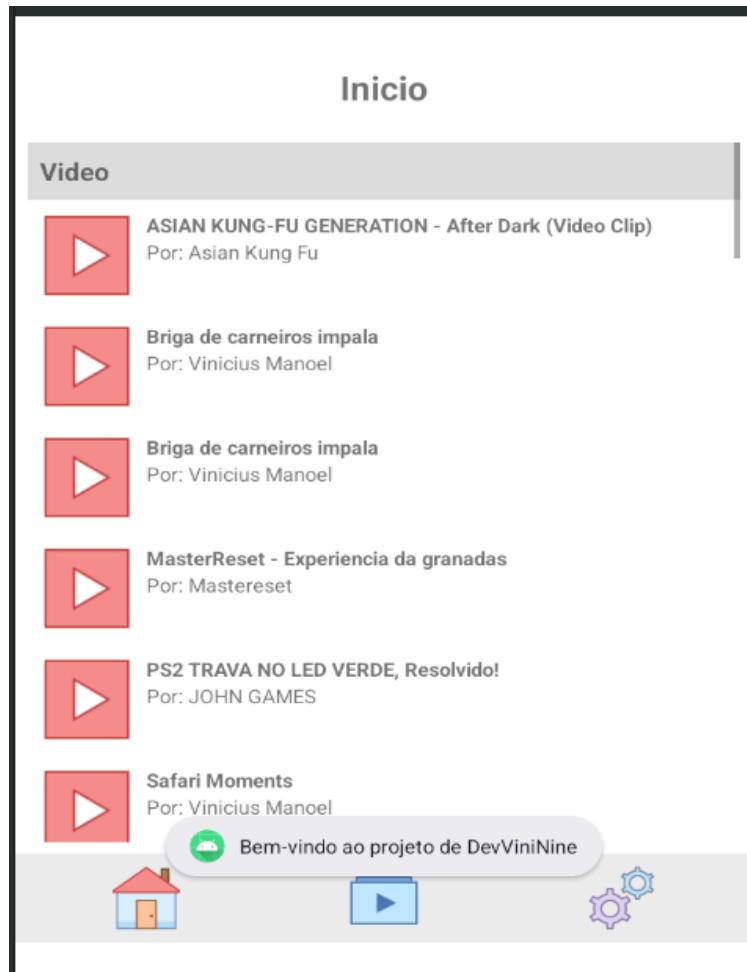


Figura 40 – Tela de inicio (Feed de conteúdos)

Após a autenticação bem-sucedida do usuário, o aplicativo exibe a tela inicial, responsável por apresentar um feed dinâmico e contínuo de conteúdos disponíveis na plataforma. Esse feed pode ser considerado “infinito”, à medida que exibe, de forma rolável, todos os conteúdos retornados pela API, sem limitação fixa de itens visíveis inicialmente.

Os conteúdos são organizados por tipo, como vídeos, músicas, imagens e podcasts, e exibidos com seus respectivos nomes e criadores. Essa estrutura facilita a navegação e o consumo do material de forma intuitiva, permitindo ao usuário identificar rapidamente os conteúdos do seu interesse.

Os dados exibidos são obtidos por meio do endpoint de listagem da entidade Conteúdo. A resposta da API é interpretada e renderizada no aplicativo,

possibilitando a visualização contínua de novos itens conforme o usuário navega pela tela.

Na parte inferior da interface, destaca-se a presença de uma barra de navegação inspirada na organização do YouTube, que permite o acesso rápido às demais seções do aplicativo. O token de autenticação do usuário permanece armazenado, garantindo que todas as operações protegidas continuem acessíveis sem a necessidade de novo login.

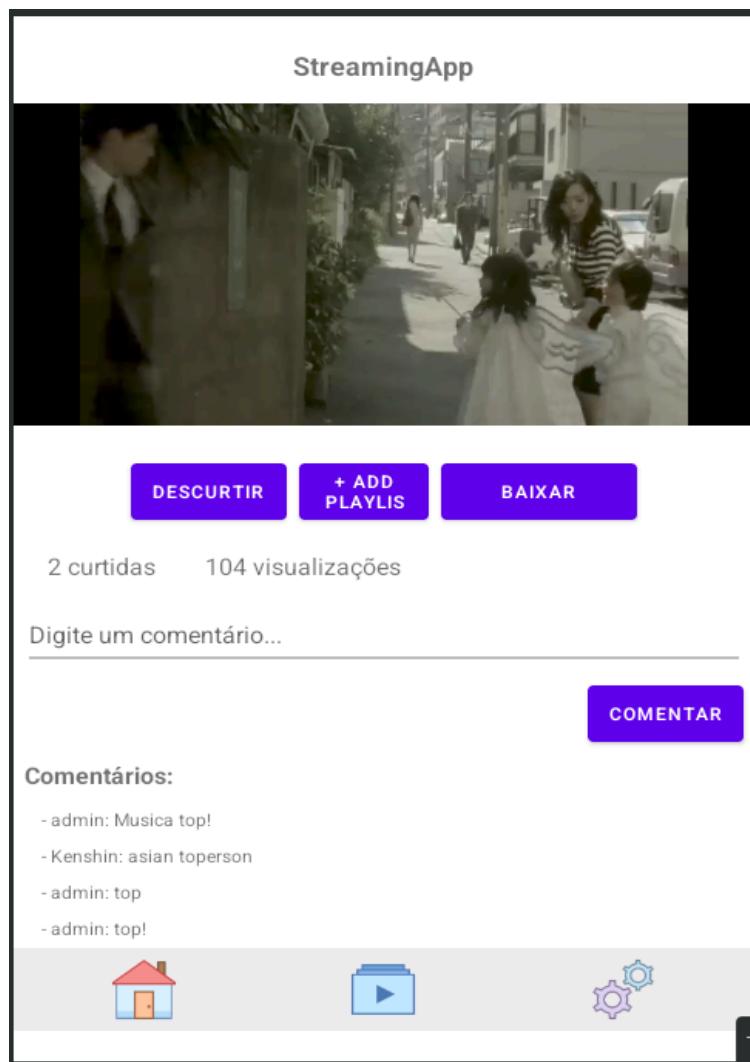


Figura 41 – Reprodução de conteúdo

A tela de reprodução é acionada sempre que o usuário seleciona um conteúdo do tipo vídeo no feed principal. Nela, o vídeo é exibido na parte superior da

interface, utilizando o player nativo do Android configurado com a biblioteca ExoPlayer, garantindo compatibilidade e desempenho na reprodução.

Abaixo do vídeo, há um conjunto de funcionalidades que reforçam a interatividade do sistema: é possível curtir ou descurtir o conteúdo, adicioná-lo a uma playlist previamente criada e realizar o download do arquivo, que abre o link direto em um navegador. Essas ações são vinculadas aos endpoints correspondentes da API, que são consumidos utilizando o token de autenticação salvo localmente.

Além disso, a tela exibe o número total de curtidas e visualizações, atualizados dinamicamente conforme os endpoints de estatísticas. Logo abaixo, encontra-se o campo para inserção de comentários. O usuário pode escrever e publicar novas mensagens, que são imediatamente exibidas na lista de comentários recuperada da API para aquele conteúdo.

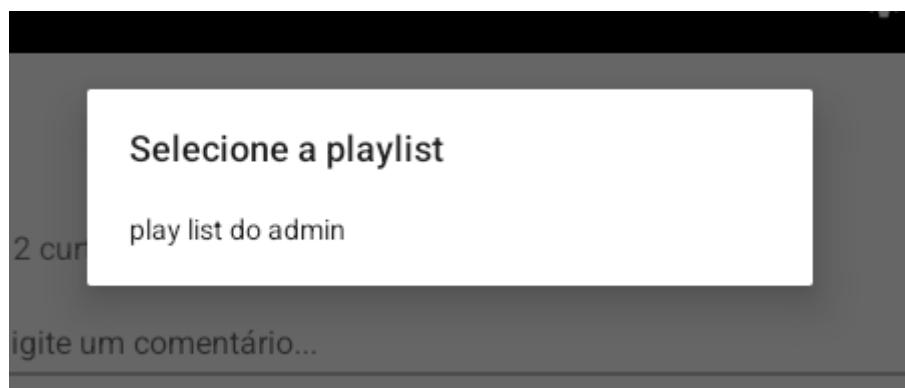


Figura 42 – Dialog de adicionar a playlist

Ao clicar no botão “**+ ADD PLAYLIST**” durante a visualização de um conteúdo, o aplicativo exibe uma janela sobreposta (do tipo **AlertDialog**) com a lista de playlists previamente criadas pelo usuário autenticado.

A lista é preenchida dinamicamente por meio de uma requisição à API que retorna as playlists associadas ao usuário logado. Ao selecionar uma das opções, o app realiza uma nova chamada à API para vincular o conteúdo atual à playlist escolhida. Caso o conteúdo já esteja na playlist, a resposta do servidor retorna uma mensagem de aviso, informando que o item já foi adicionado anteriormente.

Essa funcionalidade permite ao usuário construir e organizar suas playlists de maneira prática, diretamente da interface de reprodução, sem a necessidade de navegação por diferentes telas.

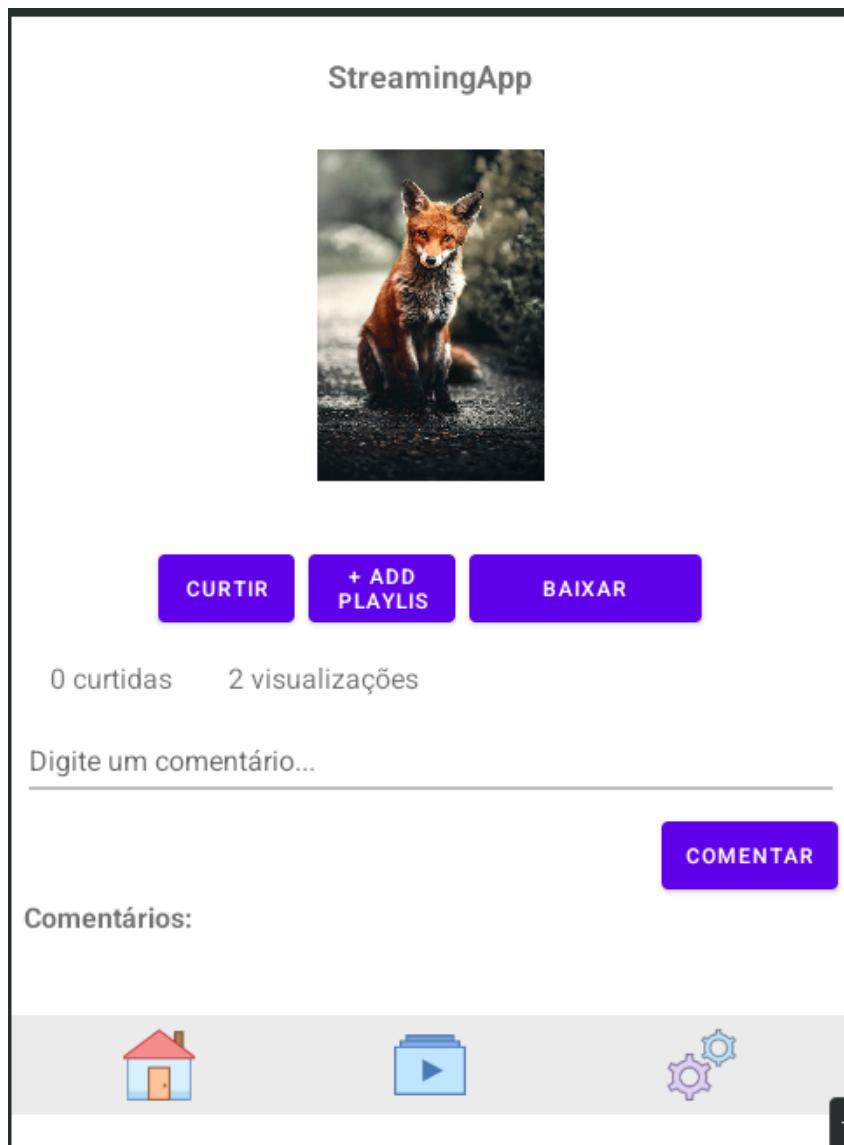


Figura 43 – Visualização de imagens

Além da reprodução de vídeos, o aplicativo também oferece suporte à visualização de imagens. A interface segue o mesmo padrão funcional da tela de vídeos, com botões para curtir, adicionar à playlist, comentar e baixar o conteúdo.

A imagem é centralizada no topo da tela e exibida em tamanho proporcional ao espaço disponível. As interações com curtidas, visualizações e comentários são registradas normalmente na API, garantindo a uniformidade do sistema para

diferentes tipos de mídia. Essa abordagem reforça a proposta multimídia da plataforma.

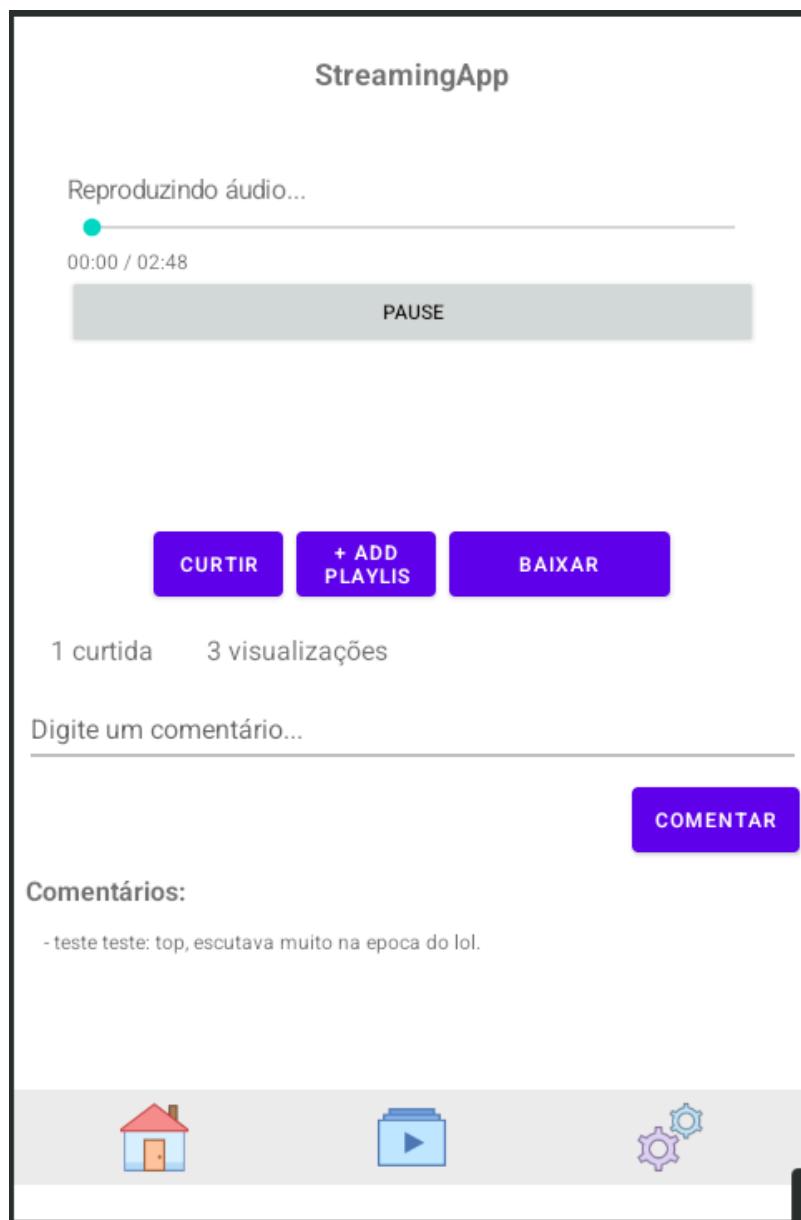


Figura 44 - Reprodução de Áudio

A tela de reprodução de áudios foi projetada para fornecer uma experiência simples e eficiente ao usuário. Nela é possível escutar conteúdos como músicas, podcasts ou qualquer outro tipo de áudio hospedado na plataforma.

O player exibe o tempo decorrido e o tempo total da faixa, com botões funcionais para pausar ou retomar a reprodução. Assim como nas demais mídias, o usuário pode curtir o conteúdo, adicionar à playlist, comentar e também realizar o

download. O sistema registra automaticamente a visualização e a interação com a API. Essa abordagem garante consistência entre os diferentes formatos de mídia oferecidos no aplicativo.

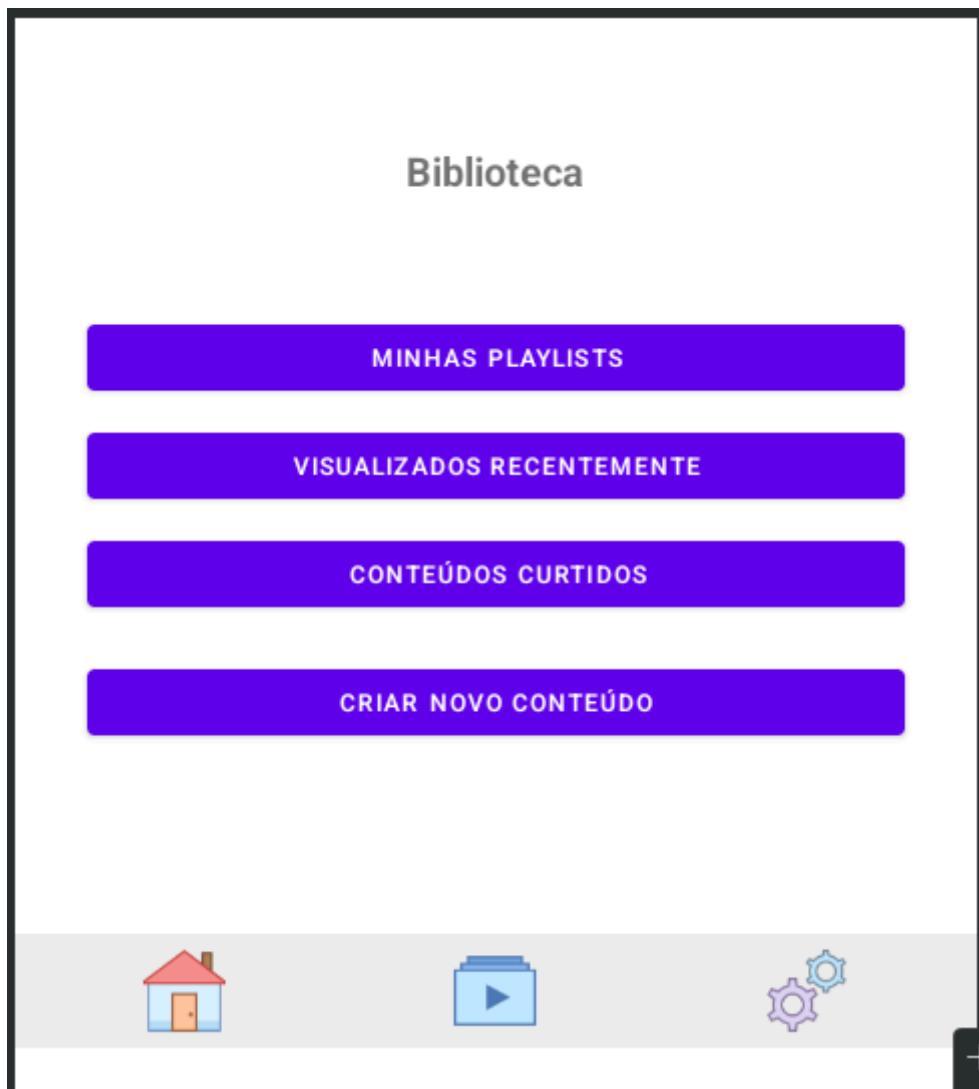


Figura 45 – Biblioteca de Conteúdos

A biblioteca é acessada por meio do botão central da barra inferior de navegação. Essa interface reúne funcionalidades voltadas à organização e gerenciamento dos conteúdos com os quais o usuário interagiu.

Na tela, são disponibilizadas quatro opções de minhas playlists, visualizados recentemente, conteúdos curtidos e criar Novo Conteúdo.

Essa tela centraliza o acesso às interações do usuário com a plataforma, promovendo uma navegação mais intuitiva e funcional.



Figura 46 – Tela “Minhas Playlists”

Ao acessar a opção **Minhas Playlists** dentro da Biblioteca, o usuário é direcionado para uma interface dedicada à visualização de todas as playlists que ele criou.

No topo da tela, há o botão **Criar Nova Playlist**, que permite que o usuário adicione uma nova lista personalizada à sua conta. Abaixo, são exibidas as playlists existentes, com seus respectivos nomes organizados em formato de lista.

Cada item da lista é apresentado com um ícone representativo e o nome definido pelo usuário no momento da criação. Ao clicar sobre uma das playlists, o sistema carrega todos os conteúdos associados a ela, permitindo ao usuário explorar, curtir ou reproduzir diretamente os itens salvos.

Visualizados recentemente

ASIAN KUNG-FU GENERATION - After Dark (Video Clip) (Video)
22/05/25

ASIAN KUNG-FU GENERATION - After Dark (Video Clip) (Video)
22/05/25

ASIAN KUNG-FU GENERATION - After Dark (Video Clip) (Video)
22/05/25

ASIAN KUNG-FU GENERATION - After Dark (Video Clip) (Video)
22/05/25

ASIAN KUNG-FU GENERATION - After Dark (Video Clip) (Video)
22/05/25

ASIAN KUNG-FU GENERATION - After Dark (Video Clip) (Video)
22/05/25

ASIAN KUNG-FU GENERATION - After Dark (Video Clip) (Video)
22/05/25

ASIAN KUNG-FU GENERATION - After Dark (Video Clip) (Video)
22/05/25

Safari Moments (Video)
21/05/25

ASIAN KUNG-FU GENERATION - After Dark (Video Clip) (Video)
21/05/25



Figura 47 – Tela “Visualizados Recentemente”

A tela **Visualizados Recentemente** apresenta ao usuário uma lista dos conteúdos acessados por ele em momentos anteriores, funcionando como um histórico de visualizações. Os itens são organizados em ordem cronológica, permitindo que o usuário tenha uma visão clara de sua atividade dentro da plataforma.

Cada item exibe o título do conteúdo, seu tipo (vídeo, áudio, imagem ou podcast) e a data da visualização. No entanto, nesta fase do desenvolvimento, a funcionalidade de reprodução direta a partir dessa tela ainda não foi

implementada, sendo necessário voltar ao feed principal ou à área de playlists para executar o conteúdo desejado.

Essa interface visa fornecer uma referência rápida ao usuário sobre seu uso recente da plataforma, sendo uma funcionalidade útil para relembrar ou localizar materiais previamente consumidos. Futuramente, pretende-se adicionar a reprodução direta a partir desta tela.



Figura 48 – Tela “Conteúdos Curtidos”

A tela **Conteúdos Curtidos** tem como objetivo apresentar todos os conteúdos marcados com curtida pelo usuário autenticado. Nela, são exibidos

diferentes tipos de mídia como vídeos, músicas, imagens e podcasts, acompanhados de seus respectivos títulos, tipos e criadores.

Diferente da tela de visualizações recentes, esta interface permite a **reprodução direta dos conteúdos**. Ao clicar em qualquer item listado, o aplicativo encaminha o usuário automaticamente para a **tela de reprodução de conteúdo**.

Essa funcionalidade reforça a proposta de navegação facilitada e personalizada do aplicativo, possibilitando acesso rápido aos materiais que o usuário demonstrou interesse.



Figura 49 – Tela “Criar Conteúdo”

Na interface **Criar Conteúdo** é possível informar manualmente o **nome do conteúdo**, seu **tipo** (como vídeo, imagem, música ou podcast) e uma **URL direta** do conteúdo hospedado, preferencialmente no formato raw. A funcionalidade está compatível com links de repositórios como o **GitHub** ou serviços de mídia direta como o **media.istockphoto.com**.

A opção **Upload**, embora presente na interface, **ainda não está implementada no aplicativo Android**. Essa funcionalidade foi inserida apenas como **ilustração e teste de interface**, com o botão exibindo uma mensagem informativa ao usuário: “Função não implementada”.

Caso o usuário autenticado ainda não possua um cadastro como criador de conteúdo na plataforma, o aplicativo bloqueia o envio e impede a criação ou upload de conteúdos, respeitando as restrições da API.

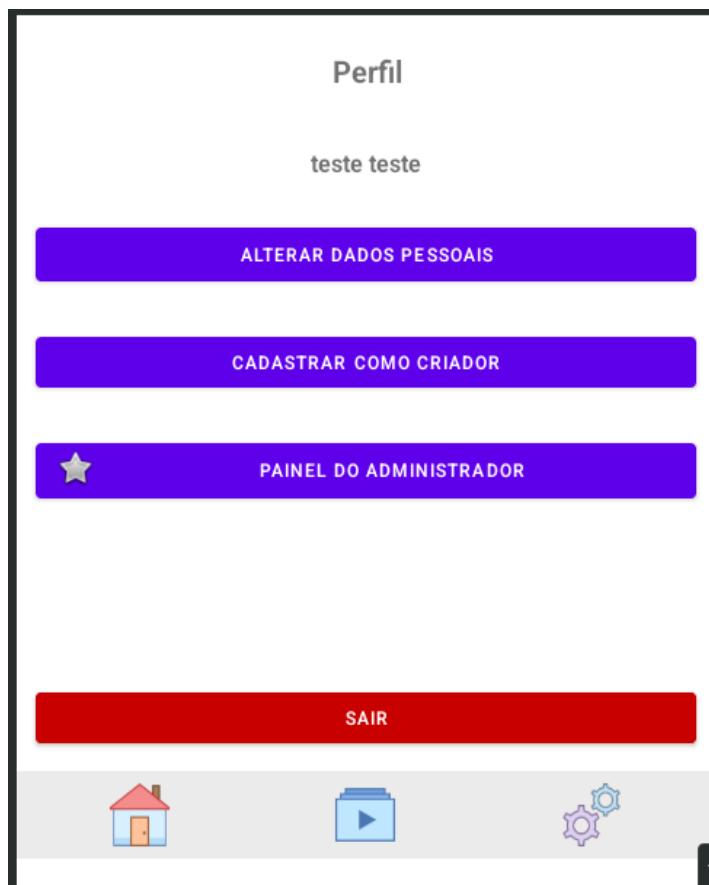


Figura 50 – Tela “Perfil”

A seção de perfil do usuário centraliza ações relacionadas à identidade, permissões e administração da conta. A tela exibe o nome do usuário logado e oferece os recursos de Alterar Dados, Cadastrar como Criador, Painel do Administrador e Sair.

Essa tela tem como foco principal permitir ao usuário controlar sua identidade e suas permissões, além de facilitar a navegação entre funcionalidades mais sensíveis como o gerenciamento administrativo.

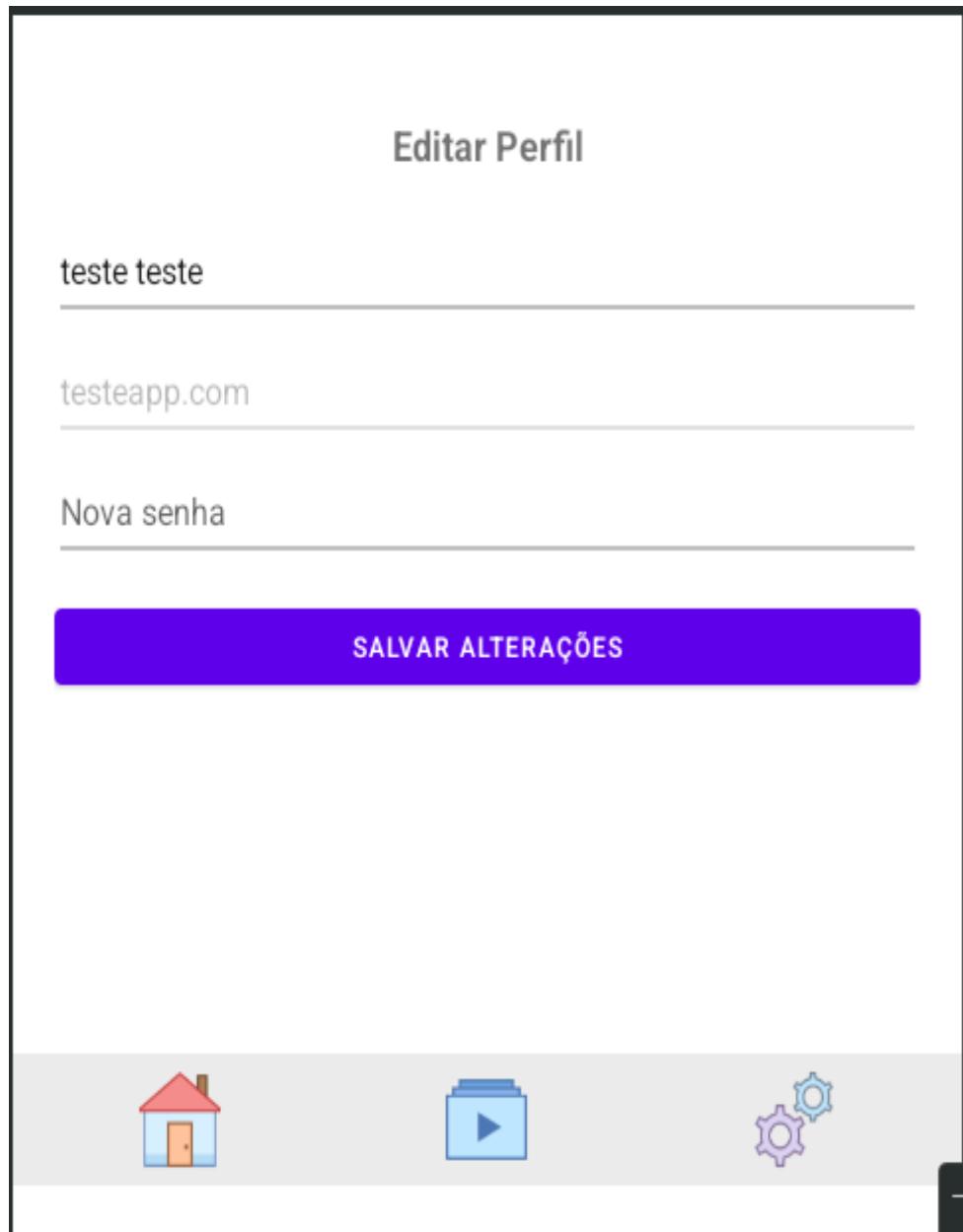


Figura 51 – Tela “Editar Perfil”

A tela de edição de perfil permite que o usuário altere seus dados pessoais de forma simples e segura. São exibidos os campos, nome e email e senha.

O campo de e-mail aparece desabilitado, indicando que não pode ser alterado, somente administradores poderão alterar email do usuário. Após preencher os dados desejados, o usuário pode tocar no botão **SALVAR ALTERAÇÕES**, que envia a solicitação para a API atualizar as informações no banco de dados.

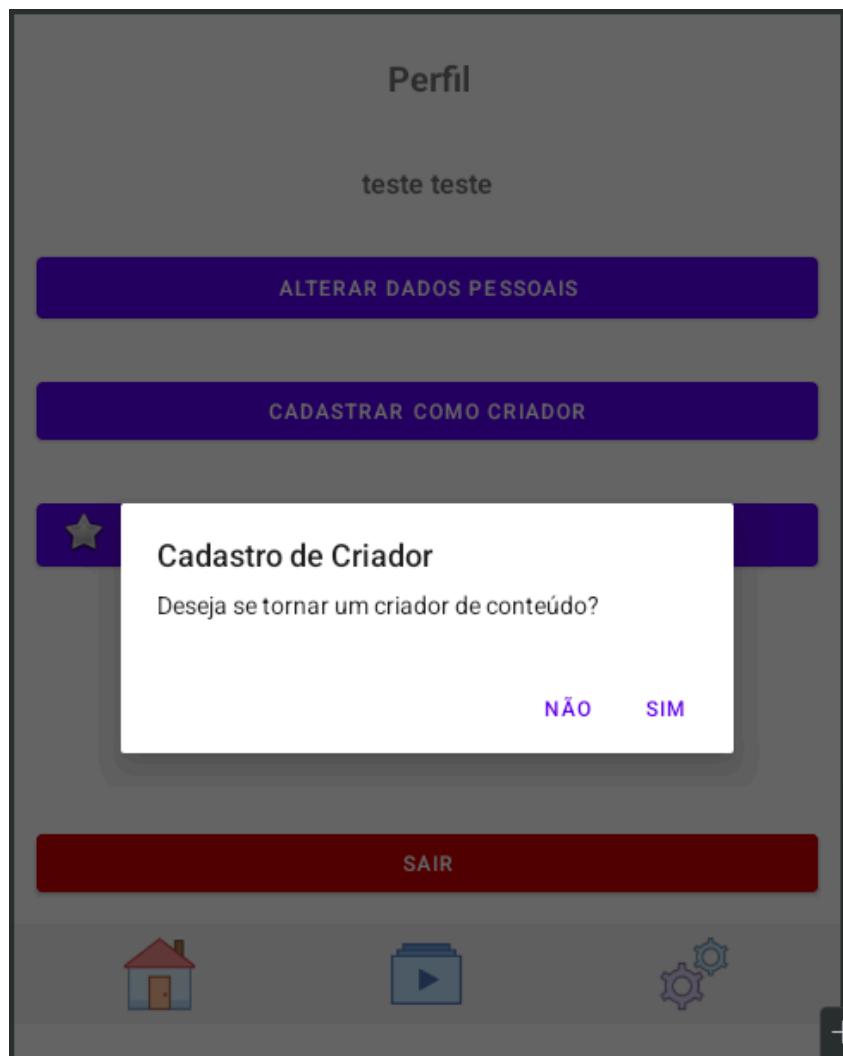


Figura 52 – Cadastro como Criador de Conteúdo

Para facilitar o processo de adesão à criação de conteúdos na plataforma, o aplicativo apresenta uma interface objetiva para os usuários interessados em se tornar criadores. Ao selecionar a opção “Cadastrar como Criador”, é exibida uma janela de confirmação com a pergunta: “*Deseja se tornar um criador de conteúdo?*”.

Caso o usuário confirme a ação, o sistema verifica se ele já está cadastrado como criador. Se positivo, uma mensagem é exibida informando que o usuário já possui esse perfil. Caso contrário, o cadastro é realizado automaticamente com base no usuário logado.

Vale destacar que o cadastro como criador é irreversível — uma vez feito, o sistema não permite desfazer a ação. Essa abordagem foi adotada para manter a integridade dos conteúdos associados ao criador.

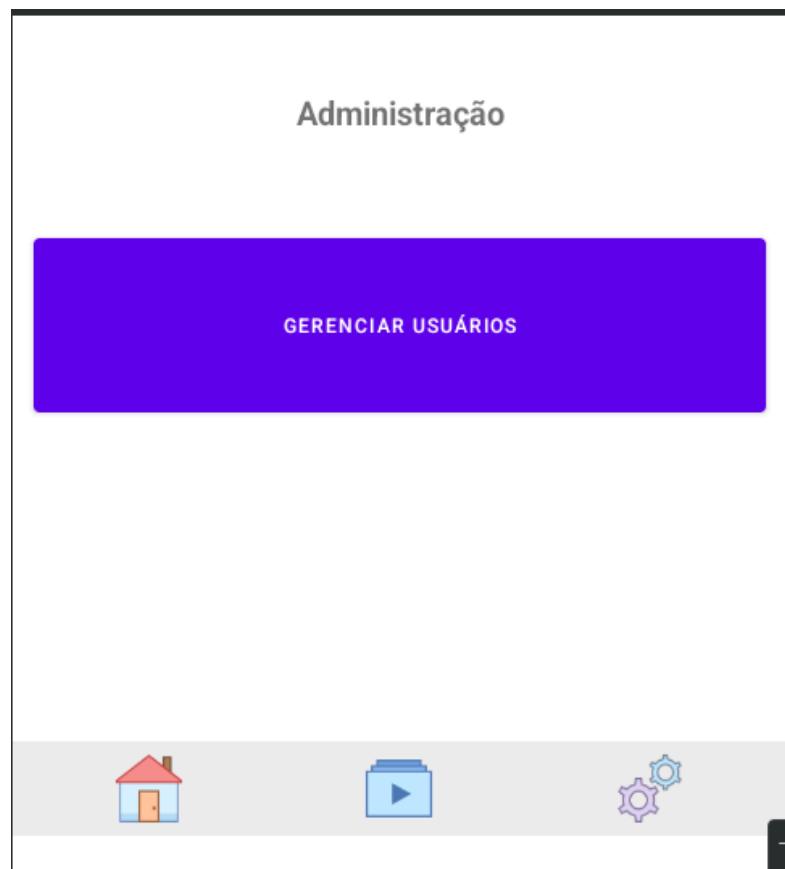


Figura 53 – Tela de Administração

A tela de administração é acessível exclusivamente para usuários com privilégio de administrador, identificado no banco de dados pelo campo **admin = 1**. Quando um usuário comum (com **admin = 0**) tenta acessar esta área, o aplicativo exibe uma mensagem informativa destacando que apenas administradores têm permissão.

Atualmente, a tela exibe apenas o botão “Gerenciar Usuários”, que direciona para o gerenciamento básico dos cadastros. Essa funcionalidade foi implementada com foco em facilitar os testes administrativos durante a fase de desenvolvimento da aplicação.

É importante ressaltar que este painel foi projetado com possibilidade de expansão. Novos recursos administrativos poderão ser incorporados futuramente, como: controle de conteúdos, estatísticas de acesso, moderação de comentários, entre outros, ampliando a gestão da plataforma diretamente pelo aplicativo.

Usuários cadastrados

- 3 - Marcos testes
- 18 - admin (admin)
- 20 - 2025teste (admin)
- 21 - teste api pronta (admin)
- 1023 - Vinicius Manoel (admin)
- 2022 - strin123g
- 3024 - Kenshin
- 3025 - TESTE MAUI
- 4025 - tessssste



Figura 54 – Tela de Usuários Cadastrados

A tela de gerenciamento de usuários, acessível apenas por administradores, exibe uma lista com todos os usuários registrados na plataforma. Cada item da lista apresenta, identificador único (ID) do usuário, nome cadastrado.

A indicação explícita caso o usuário possua status de administrador (exibido como “(admin)”. Essa visualização tem como objetivo

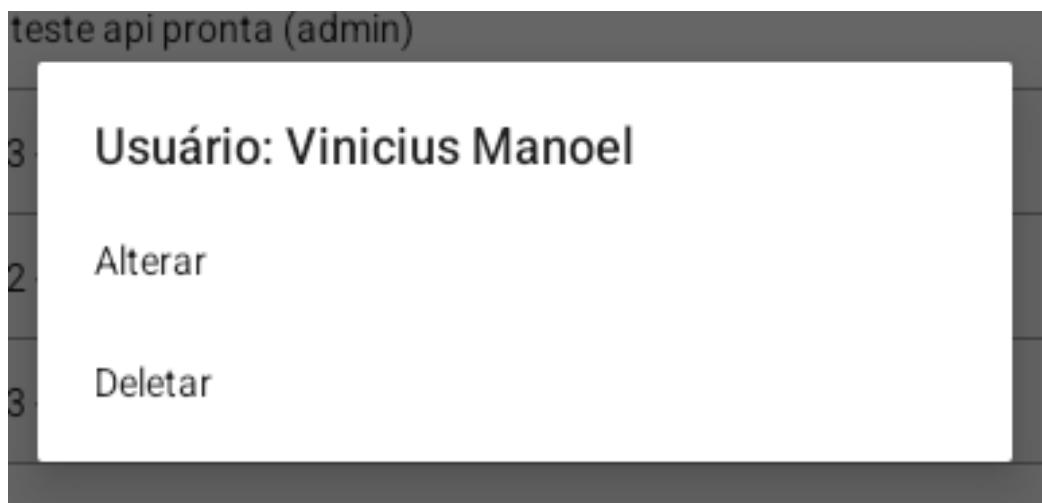


Figura 55 – Dialog de alterar usuário

Ao selecionar um usuário na tela de gerenciamento, o aplicativo exibe uma janela de ação com duas opções disponíveis: “Alterar” e “Deletar”. Esta abordagem foi projetada para permitir que administradores editem ou removam usuários com facilidade.

Entretanto, o sistema impõe restrições importantes: usuários que estejam relacionados como criadores de conteúdo, ou que possuam interações registradas na plataforma (como curtidas, playlists, ou visualizações), não podem ser deletados diretamente pelo aplicativo. Essa limitação foi implementada como medida de integridade de dados, uma vez que a remoção direta poderia comprometer vínculos importantes dentro do sistema.

A exclusão definitiva de usuários com dependências só é possível por meio de ações diretas no banco de dados, ou por meio de futuros endpoints

administrativos que serão desenvolvidos e integrados à API, conforme a plataforma evolua e novas funcionalidades sejam incorporadas.



Figura 56 – Edição de usuários (Admin)

A tela de edição de usuário permite que administradores realizem modificações nos dados cadastrais de qualquer usuário registrado no sistema. Nela, é possível alterar o nome, a senha e, diferentemente do painel do próprio usuário, também o e-mail da conta.

Um recurso exclusivo dessa interface é o switch “**Administrador**”, que permite conceder ou remover o status de administrador a um usuário. Essa funcionalidade está visível apenas para quem já possui privilégio administrativo, garantindo controle de acesso seguro dentro da aplicação.

Após as alterações, o botão “**Salvar**” executa a atualização, enquanto o botão “**Cancelar**” retorna à tela anterior sem aplicar modificações. Essa tela reforça a autonomia do administrador sobre a base de usuários e centraliza o gerenciamento de permissões de forma prática.

6.5. DIAGRAMA DE CASO DE USO

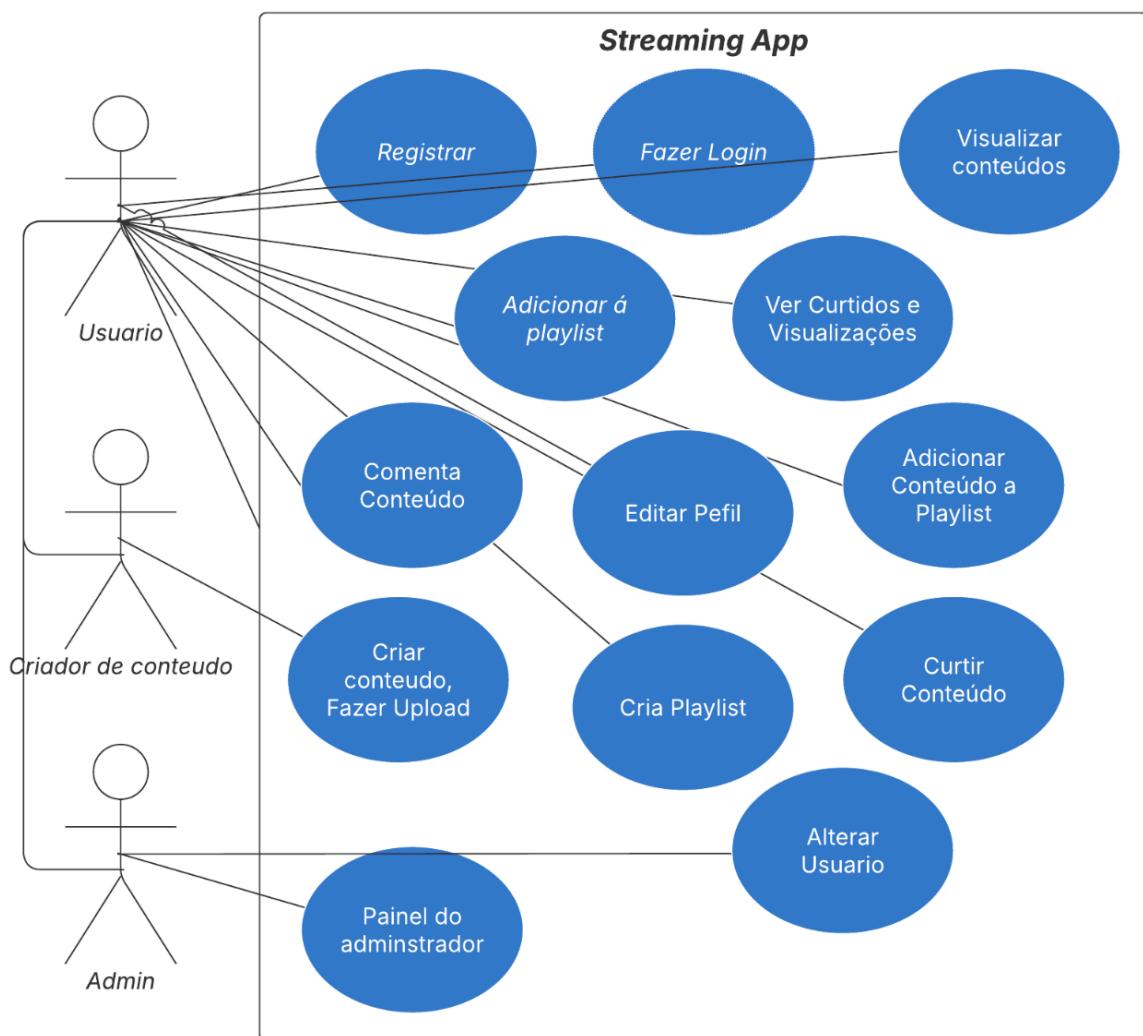


Figura 57 – Diagrama de caso de uso

O diagrama de casos de uso acima representa as **principais funcionalidades do sistema de streaming multimídia**, destacando a interação dos três principais tipos de atores: **Usuário, Criador de conteúdo e Administrador**.

- **Usuário:** representa qualquer pessoa que acessa o sistema com um login válido.

- **Criador de conteúdo:** especialização do ator “Usuário”, com permissões adicionais voltadas à publicação e gerenciamento de conteúdo.
- **Admin:** especialização do ator “Usuário”, com acesso a funções administrativas exclusivas, como gerenciamento de usuários.

Relações de Generalização

- O **Criador de conteúdo** herda os casos de uso do **Usuário**, pois ele também realiza ações comuns como login, visualização e curtidas.
- O **Admin** também herda os casos de uso do **Usuário**, estendendo suas permissões para atividades administrativas.

Casos de Uso Associados

Para Usuário

- **Registrar:** cria uma nova conta no sistema.
- **Fazer Login:** autentica o usuário e libera acesso às funcionalidades do sistema.
- **Visualizar conteúdos:** acessa a lista de conteúdos disponíveis.
- **Ver curtidos e visualizações:** consulta as interações realizadas no app.
- **Adicionar à playlist:** insere conteúdos em playlists já criadas.
- **Comentar conteúdo:** publica comentários nos conteúdos visualizados.
- **Curtir conteúdo:** interage com o conteúdo atribuindo curtidas.
- **Editar perfil:** permite alteração de nome ou senha.
- **Criar playlist:** permite a organização personalizada dos conteúdos.
- **Adicionar conteúdo à playlist:** funcionalidade complementar à manipulação de playlists.

Para Criador de conteúdo

- **Criar conteúdo, Fazer upload:** permite o envio de conteúdos por meio de links diretos ou (futuramente) por upload interno.

Para Admin

- **Painel do administrador:** interface exclusiva com funções administrativas.

- **Alterar usuário:** edita ou remove dados de usuários cadastrados no sistema.

6.6. CONSIDERAÇÕES FINAIS SOBRE O APLICATIVO MOBILE

Com o desenvolvimento do aplicativo mobile concluído em sua fase inicial, foi possível integrar com sucesso a Web API e garantir funcionalidades essenciais para o consumo e interação com conteúdos multimídia. A aplicação, desenvolvida em Java com bibliotecas modernas como Retrofit, Gson Converter, OkHttp Logging Interceptor e Android Media3, demonstrou-se funcional e adaptável para diferentes tipos de mídia, como vídeos, áudios, imagens e podcasts.

Além disso, a navegação entre telas foi cuidadosamente planejada, adotando uma interface intuitiva com barra inferior inspirada no YouTube, o que facilita a usabilidade e o acesso rápido às principais funções do sistema. Reforça-se que o aplicativo pode ser executado tanto em emuladores quanto em dispositivos físicos, contanto que a API esteja acessível.

A arquitetura modular da aplicação e a separação clara entre responsabilidades permitem que novas funcionalidades sejam implementadas de forma contínua, como é o caso da futura função de upload real de arquivos e melhorias nos recursos administrativos.

Com isso, encerramos o ciclo de apresentação do aplicativo Android. No entanto, o sistema vai além do mobile. Na próxima seção, será apresentada a plataforma desktop desenvolvida com .NET MAUI, voltada especialmente para criadores de conteúdo que desejam uma interface mais robusta para gerenciamento, publicação e controle de suas produções multimídia no ambiente Windows.

Essa nova aplicação complementa o ecossistema do projeto e representa um passo importante rumo à consolidação de uma plataforma multiplataforma completa e funcional.

7. PROTÓTIPO DA INTERFACE DESKTOP COM .NET MAUI

Com o objetivo de proporcionar aos criadores de conteúdo uma experiência mais completa, eficiente e compatível com fluxos de trabalho profissionais, foi iniciado o desenvolvimento de uma aplicação desktop utilizando a tecnologia .NET MAUI. Diferentemente do aplicativo mobile, que tem foco no consumo e interação com o conteúdo, esta plataforma é voltada especificamente para o gerenciamento, criação, análise e acompanhamento de métricas por parte dos produtores.

A escolha de concentrar as ferramentas do criador no ambiente de computador se justifica pela praticidade que esse tipo de dispositivo oferece em tarefas que exigem maior controle, como inserção de dados, organização de mídias, leitura de relatórios e acompanhamento de desempenho. Tarefas como criar playlists, editar descrições, monitorar visualizações e curadorias de conteúdo são melhor desempenhadas em interfaces maiores e com suporte a múltiplas janelas, comuns em desktops ou notebooks.

O uso do .NET MAUI como base do projeto permite que essa aplicação seja multiplataforma, funcionando em sistemas operacionais como Windows, macOS e futuramente até Linux, ampliando o alcance e a flexibilidade da solução. Através dessa tecnologia, o criador poderá criar, editar, acompanhar e visualizar suas métricas a partir de qualquer sistema compatível, mantendo a integração com a mesma API utilizada pelo app mobile.

Nesta etapa do desenvolvimento, foi construído um protótipo funcional com foco no fluxo de autenticação. O sistema já conta com telas de login e cadastro de usuários plenamente integradas à Web API, garantindo o controle de acesso seguro por meio de token JWT. Após a autenticação, o criador é direcionado à tela inicial da aplicação, uma dashboard que será o ponto de partida para todas as funcionalidades futuras.

A seguir, serão apresentados os prints da interface construída com .NET MAUI, demonstrando o funcionamento das telas implementadas e como elas se comunicam diretamente com a API já desenvolvida.

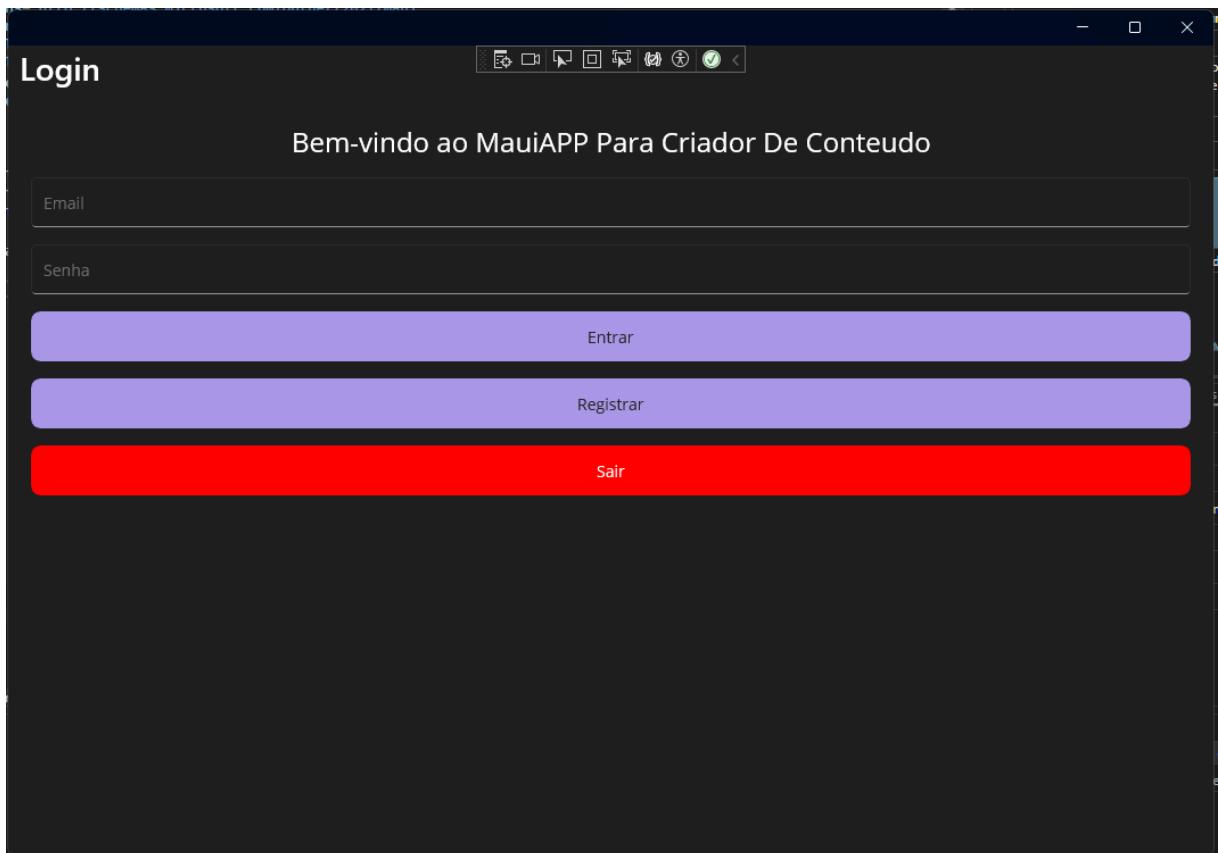


Figura 58 – Tela de Login do MauiAPP

A primeira interface apresentada ao iniciar o MauiAPP é a tela de **Login**, essencial para garantir a segurança e a personalização do acesso ao ambiente do criador de conteúdo. Nela, o usuário deve inserir seu e-mail e senha previamente cadastrados. Caso as credenciais estejam corretas, o sistema autentica o acesso por meio de um **token JWT**, permitindo a navegação segura pelas demais funcionalidades da aplicação.

Além da opção de login, a tela também conta com o botão "**Registrar**", possibilitando que novos usuários realizem o auto cadastro diretamente na plataforma. O botão "**Sair**", por sua vez, encerra a aplicação. Toda essa lógica está completamente **integrada à Web API**, garantindo a consistência e a centralização das informações, da mesma forma que ocorre no aplicativo mobile.

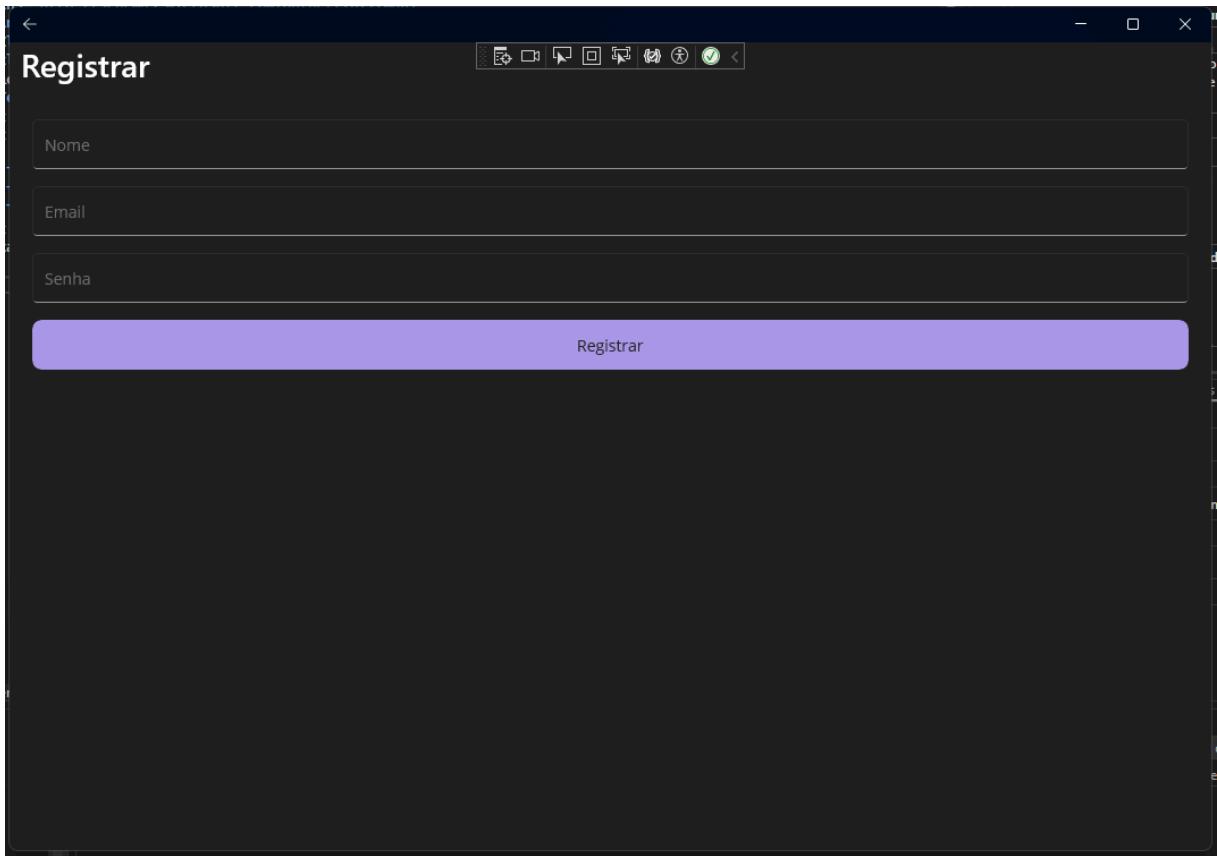


Figura 59 - Tela de Registro de Usuário

A tela de **registro** do MauiAPP permite que novos usuários se cadastrem diretamente na plataforma de forma simples e eficiente. Nela, o usuário deve informar o **nome**, **e-mail** e **senha** para concluir o processo.

Essa interface está completamente integrada à API do sistema, sendo responsável por enviar os dados ao endpoint de criação de usuário. Caso o e-mail já esteja em uso ou haja algum erro no preenchimento, a interface exibe mensagens informativas ao usuário.

O registro bem-sucedido habilita o acesso às funcionalidades da plataforma voltadas ao criador de conteúdo, desde que posteriormente ele se cadastre como criador.

Na próxima tela, será apresentada a **dashboard inicial**, que aparece após o login, fornecendo um panorama geral para o criador.

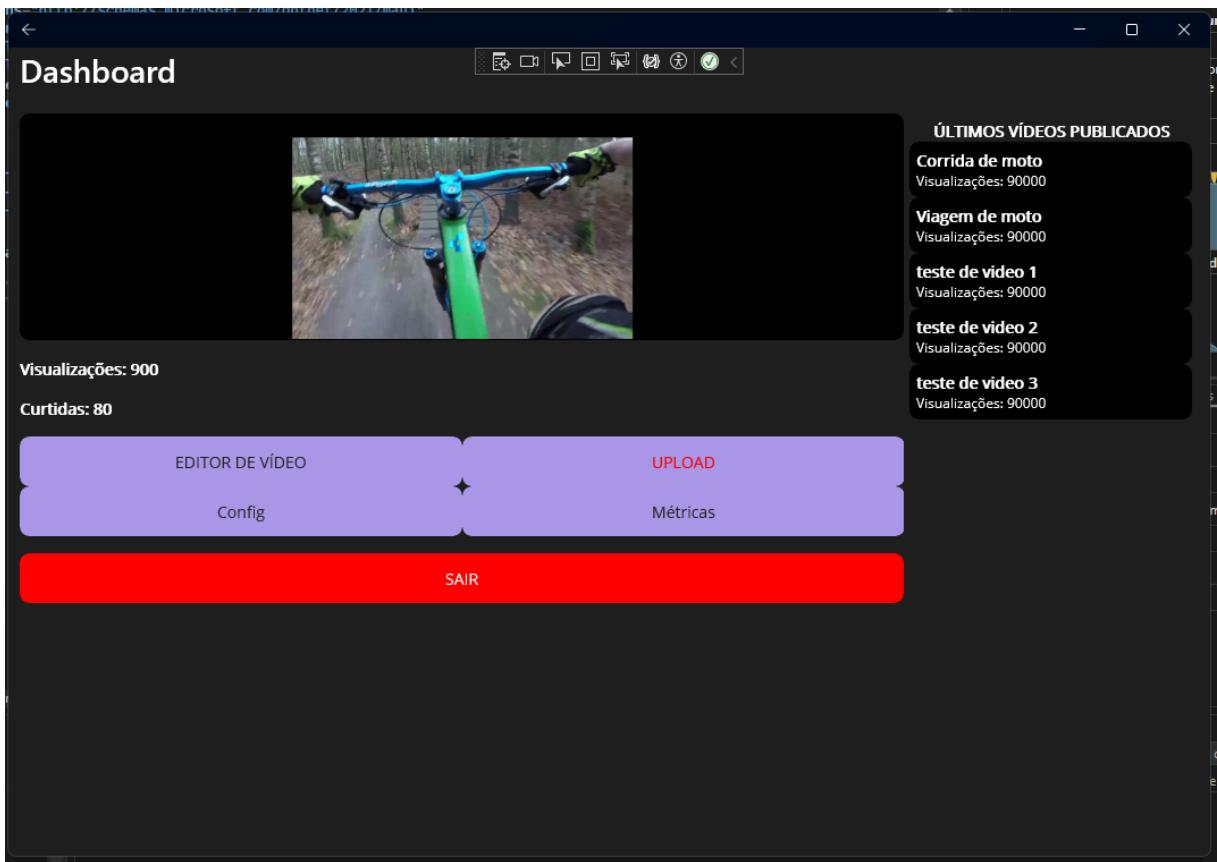


Figura 60 – Protótipo Tela de Dashboard do Criador de Conteúdo

A tela de **dashboard** é a primeira interface exibida ao criador de conteúdo após o login no aplicativo MauiAPP. Embora ainda esteja em fase de prototipagem, essa tela já oferece uma visão geral das publicações mais recentes e de algumas métricas essenciais, como o total de **visualizações** e **curtidas** do último conteúdo publicado.

No painel lateral direito, uma **lista com os últimos vídeos publicados** é exibida, servindo como um atalho visual e prático para acompanhamento rápido. A tela também antecipa a ideia de uma **integração com um editor de vídeo interno**, que futuramente permitirá cortes simples, ajustes de resolução e outras edições básicas.

Além disso, há seções dedicadas a **configurações da conta** e ao acesso às **métricas completas** de todos os conteúdos vinculados ao criador. Esse protótipo visa facilitar a gestão e acompanhamento do desempenho diretamente do ambiente desktop, trazendo funcionalidades estratégicas em um único lugar.

7.1. CONSIDERAÇÕES FINAIS SOBRE O APLICATIVO EM .NET MAUI

O desenvolvimento do protótipo do aplicativo **MauiAPP** representa um passo importante na construção de uma ferramenta voltada especialmente para **criadores de conteúdo**. Com foco no ambiente **desktop** e utilizando a tecnologia **.NET MAUI**, a proposta é oferecer uma plataforma robusta, multiplataforma e adaptável às rotinas de quem precisa gerenciar publicações com maior controle e praticidade.

Ainda em estágio inicial, o sistema já permite **login e registro totalmente integrados à API**, mostrando o potencial da aplicação para evoluir com rapidez. A ideia central é que o criador possa, em um único local, **acompanhar o desempenho de seus conteúdos, editar vídeos, realizar uploads**, além de configurar sua conta e visualizar métricas relevantes — tudo isso com uma experiência mais confortável e completa que só um ambiente de computador pode proporcionar.

Embora muitos recursos estejam apenas esboçados, o MauiAPP já estabelece as **bases de uma interface moderna e funcional**, preparada para futuras expansões. Entre os planos está a implementação de um sistema completo de edição de vídeos e imagens, novas formas de visualização de dados analíticos, além de uma integração mais profunda com o aplicativo mobile, formando um ecossistema unificado e responsivo.

8. ESTRUTURA RELACIONAL DO BANCO DE DADOS

Para sustentar as operações do sistema de streaming multimídia desenvolvido, foi projetado um banco de dados relacional que organiza os dados de forma lógica e eficiente. O diagrama de Entidade-Relacionamento (DER) representa visualmente essa estrutura, evidenciando como os dados estão interligados e quais são as dependências entre as entidades. Essa modelagem foi fundamental para garantir a integridade referencial, a escalabilidade e a correta associação entre usuários, conteúdos, playlists e interações como curtidas, comentários e visualizações.

A seguir, será apresentada uma imagem com a estrutura do banco e, logo abaixo, uma explicação detalhada de cada uma das relações entre as tabelas envolvidas.

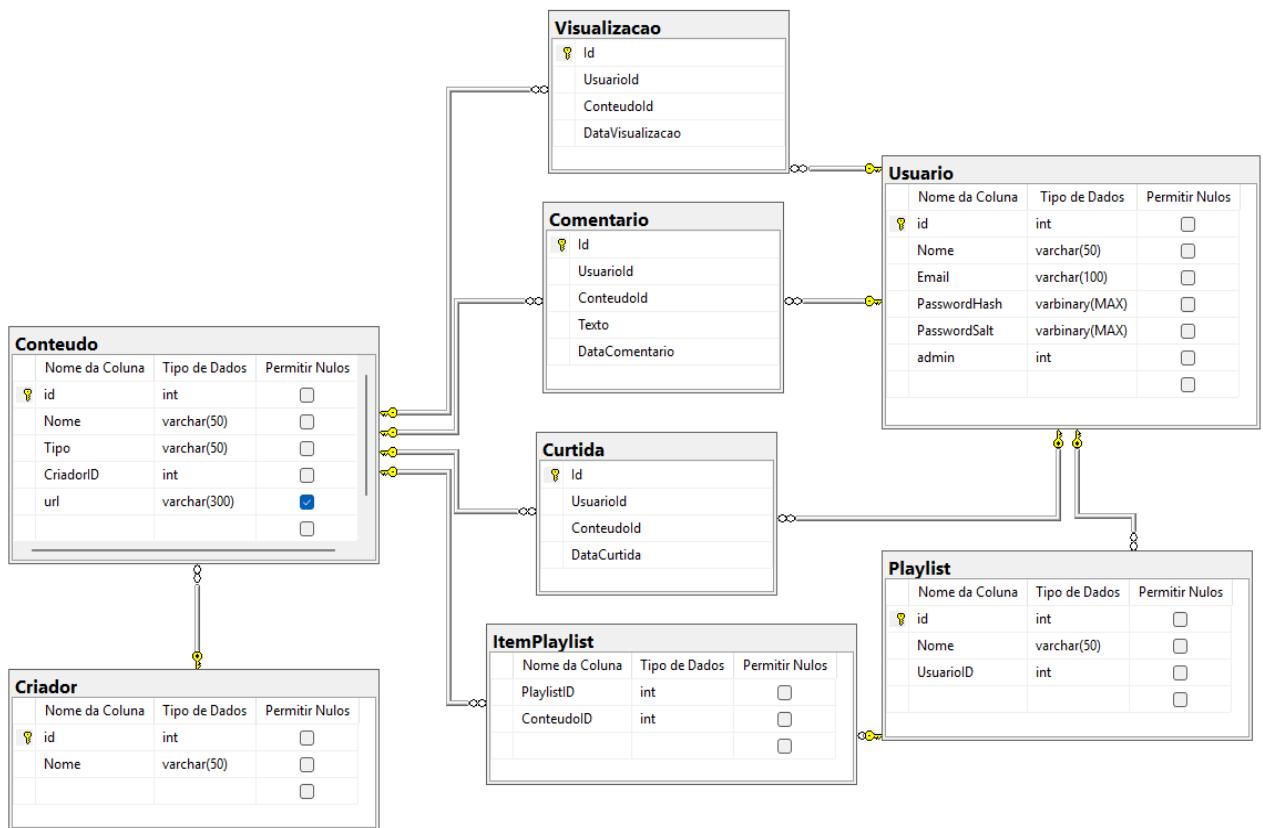


Figura 61 - Diagrama de Entidade-Relacionamento

Relacionamentos do Diagrama

1. Usuário ↔ Playlist

- Relação: Um para muitos (1:N)

- Explicação: Cada usuário pode criar várias playlists, mas cada playlist pertence a apenas um usuário. A foreign key Usuariold na tabela Playlist referencia a tabela Usuario.

2. Usuário ↔ Curtida

- Relação: Um para muitos (1:N)

Explicação: Um usuário pode curtir diversos conteúdos, e cada curtida está vinculada a um único usuário. A foreign key Usuariold em Curtida representa essa ligação.

3. Usuário ↔ Comentário

- Relação: Um para muitos (1:N)
- Explicação: Um usuário pode comentar em diversos conteúdos. Cada comentário está relacionado a um único usuário via Usuariold.

4. Usuário ↔ Visualização

- Relação: Um para muitos (1:N)

Explicação: Cada usuário pode visualizar vários conteúdos. A foreign key Usuariold na tabela Visualizacao representa essa associação.

5. Conteúdo ↔ Comentário

- Relação: Um para muitos (1:N)

Explicação: Um conteúdo pode receber diversos comentários. Cada comentário pertence a apenas um conteúdo via Conteudold.

6. Conteúdo ↔ Curtida

- Relação: Um para muitos (1:N)
- Explicação: Um conteúdo pode receber várias curtidas de diferentes usuários. A foreign key Conteudold em Curtida realiza essa associação.

7. Conteúdo ↔ Visualização

- Relação: Um para muitos (1:N)
- Explicação: Um conteúdo pode ser visualizado por vários usuários. Cada visualização é registrada com Conteudold.

8. Conteúdo ↔ ItemPlaylist

- Relação: Um para muitos (1:N)

Explicação: Um conteúdo pode estar presente em várias playlists. A tabela ItemPlaylist funciona como tabela associativa entre Conteudo e Playlist.

9. Playlist ↔ ItemPlaylist

- Relação: Um para muitos (1:N)
- Explicação: Uma playlist pode conter diversos conteúdos. Essa relação é estabelecida por PlaylistID na tabela ItemPlaylist.

10. Criador ↔ Conteúdo

- Relação: Um para muitos (1:N)

Explicação: Cada conteúdo é publicado por um criador. A foreign key CriadorID na tabela Conteudo aponta para a tabela Criador.

9. PLANILHA DE TESTES – INTERAÇÃO DO USUÁRIO

Esta planilha tem como objetivo organizar os testes realizados com foco na **interação do usuário dentro do aplicativo**. Ela registra as principais ações feitas pelos usuários, como login, visualização de conteúdos, curtidas, comentários, criação de playlists, entre outras.

Cada teste descreve o que o usuário faz, o que o sistema deve retornar e se o comportamento foi o esperado. Dessa forma, é possível identificar falhas, confirmar se tudo está funcionando corretamente e garantir uma boa experiência para quem usa o app.

Os testes estão organizados por ID, com campos para identificar a funcionalidade, descrição do teste, entrada, resultado esperado, resultado obtido.

TESTES STREAMING API					
ID DO TESTE	FUNCIONALIDADE	DESCRÍÇÃO DO TESTE	ENTRADA / AÇÃO	RESULTADO ESPERADO	RESULTADO OBTIDO
TC001	Login	Teste de login	Nome de usuário: "teste", Senha: "teste".	Acesso permitido, redirecionamento para o feed	Funcionou
TC002	Cadastro de Usuário	Teste de cadastro de usuário	Nome de usuário: "admin", Senha: "admin123", Email:"admin@admin.com"	Mensagem : "Usuario cadastrado com sucesso"	Função Cadastro de usuario funciona
TC003	Cadastro de Usuário	Cadastro de Usuário já existente	Nome de usuário: "admin", Senha: "admin123", Email:"admin@admin.com"	Mensagem: "Ocorreu um erro ao cadastrar"	Evitou cadastro em duplicidade
TC004	Login	Login com dados Inválidos	Nome de usuário: "administrador", Senha: "admin123", Email:"admin@admin.com"	Mensagem de erro: "Usuário ou senha inválido."	Usuário com dados inválidos ou não cadastrados.
TC005	Reprodução de conteúdo	Teste de reprodução de conteúdo no feed	Usuário clica em algum conteúdo no feed	Mensagem: "Carrregando conteúdo"	Função de reprodução funcionando
TC006	Reprodução de conteúdo	Teste de Curtir e Descurtir de conteúdo	Usuário clica em curtir e em seguida descurtir	Mensagem de erro: "Conteúdo curtido" e "Conteúdo discutido"	Função Curtida funcionando.

TC007	Reprodução de conteúdo	Teste de adição de conteúdo a uma playlist existente	Usuário clica em add Playlist, android exibe dialog contento lista de playlists, usuário escolhe uma playlist.	Mensagem: "Conteúdo adicionando a playlist com sucesso"	Adição de conteúdo a playlist funcionando.
TC008	Reprodução de conteúdo	Teste de comentar um conteúdo	Usuário insere um comentário e clica em comentar	Mensagem de erro: "Comentado com sucesso"	Comentário adicionado ao conteúdo com sucesso.
TC009	Playlists	Verificação de itens previamente adicionados a playlist	Usuário entra em uma playlist contendo conteúdo adicionado previamente	Exibe todos os conteúdos adicionados a playlist	Exibiu o conteúdo em uma lista
TC010	Cadastro como criador de conteúdo	Teste do cadastro de usuário como criador de conteúdo	Usuário clica na opção de cadastrar como criador, escolhe a opção "Sim"	Exibe a mensagem "Cadastrado com sucesso."	Cadastrado com sucesso, agora usuário tem acesso a função de upload.
TC011	Alteração de dados pessoais	Teste de alteração de dados pessoais, Usuário comum	Usuário altera nome e senha, campo email desabilitado.	Exibe a mensagem "Informações alteradas com sucesso"	Usuário consegue alterar nome e senha da conta.
TC012	Logout	Teste de botão logout, onde "esquece" o token do usuário e redireciona para tela de login.	Usuário clicar em logout	Redirecionar usuário para tela de login	Redirecionado com sucesso, sendo necessário refazer o login.

Tabela 1- Planilha de testes

10. CONCLUSÃO

O projeto apresentado demonstra parte da construção de um sistema de streaming multimídia funcional, dividido em três principais componentes: uma API robusta desenvolvida em .NET, um aplicativo mobile para usuários finais (Android) e um protótipo desktop voltado aos criadores de conteúdo (utilizando .NET MAUI).

Durante o desenvolvimento, foram implementadas funcionalidades essenciais como **cadastro e login de usuários**, **visualização e reprodução de conteúdos**, **interações via curtidas e comentários**, **gerenciamento de playlists** e **registro de visualizações**. A API foi documentada utilizando o Swagger, o que facilitou os testes e a validação das rotas REST. Já o aplicativo Android demonstrou, na prática, a integração entre a interface do usuário e os endpoints da API, reproduzindo vídeos, áudios, imagens e podcasts com sucesso.

Além disso, o sistema conta com permissões específicas para **criadores de conteúdo** e **administradores**, garantindo uma hierarquia clara e segura para a gestão de dados. O protótipo para desktop com .NET MAUI já contempla o fluxo de login e início da dashboard, com previsão de evolução para publicação de conteúdos, visualização de métricas e edição simplificada.

No aspecto técnico, também foram produzidos fluxogramas, um diagrama de casos de uso e o modelo relacional do banco de dados, além de planilhas de teste que auxiliaram no controle da qualidade das funcionalidades.

Por fim, é importante destacar que **o sistema ainda está em fase de desenvolvimento**, e muitas funcionalidades estão previstas para as próximas versões. Entre elas, destacam-se o **upload real de arquivos**, o **controle avançado de conteúdo pelo criador**, a **exclusão de itens da playlist** e a **ampliação dos recursos administrativos**. A estrutura atual, no entanto, já estabelece uma base sólida e escalável para o crescimento contínuo da plataforma.

O projeto segue em evolução, com foco na **experiência do usuário**, **segurança dos dados** e **expansão das funcionalidades**, visando atender tanto consumidores finais quanto produtores de conteúdo de maneira prática, acessível e eficiente.

REFERÊNCIAS

- MICROSOFT. Documentação do .NET. Disponível em:
<https://learn.microsoft.com/pt-br/dotnet/>. Acesso em: 26 maio 2025.
- MICROSOFT. Introdução ao .NET MAUI. Disponível em:
<https://learn.microsoft.com/pt-br/dotnet/maui/what-is-maui>. Acesso em: 26 maio 2025.
- GOOGLE DEVELOPERS. Desenvolvendo apps com Android Studio. Disponível em:
<https://developer.android.com/studio>. Acesso em: 26 maio 2025.
- RETROFIT. Retrofit: A type-safe HTTP client for Android and Java. Disponível em:
<https://square.github.io/retrofit/>. Acesso em: 26 maio 2025.
- POSTMAN. O que é uma API REST?. Disponível em:
<https://www.postman.com/what-is-rest-api/>. Acesso em: 26 maio 2025.
- MEDIA3. Android Media3 Documentation. Disponível em:
<https://developer.android.com/media/media3>. Acesso em: 26 maio 2025.
- MICROSOFT. Documentação do Entity Framework Core. Disponível em:
<https://learn.microsoft.com/pt-br/ef/core/>. Acesso em: 26 maio 2025.
- SWAGGER.IO. Documentação do Swagger (OpenAPI). Disponível em:
<https://swagger.io/docs/specification/about/>. Acesso em: 26 maio 2025.
- GITHUB DOCS. Working with raw content and files. Disponível em:
<https://docs.github.com/en/repositories/working-with-files/using-files/viewing-raw-content>. Acesso em: 26 maio 2025.