UNIVERSIDADE FEDERAL DO MARANHÃO

Centro de Ciências Exatas e Tecnologia

Otavio Cesar da Costa dos Santos

Um ambiente para integração e gerenciamento de conteúdo entre aplicativos de redes sociais

São Luís - MA

2022

Otavio Cesar da Costa dos Santos

Um ambiente para integração e gerenciamento de conteúdo entre aplicativos de redes sociais

Trabalho de Conclusão de Curso II apresentado ao curso Bacharelado em Engenharia da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia da Computação

Centro de Ciências Exatas e Tecnologia

Universidade Federal do Maranhão

Orientador: Prof. Dr. Rafael Fernandes Lopes

São Luís - MA

2022

Otavio Cesar da Costa dos Santos

Um ambiente para integração e gerenciamento de conteúdo entre aplicativos de redes sociais

Trabalho de Conclusão de Curso II apresentado ao curso Bacharelado em Engenharia da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia da Computação

Monografia aprovada em São Luís - MA, 03 de fevereiro de 2022:

Prof. Dr. Rafael Fernandes Lopes
Orientador
Universidade Federal do Maranhão

Prof. Me. Alana de Araujo Oliveira Meireles Teixeira
Universidade Federal do Maranhão
Professor

Prof. Dr. Davi Viana dos Santos
Universidade Federal do Maranhão
Professor

São Luís - MA
2022

Agradecimentos

Este trabalho não poderia ser terminado sem a ajuda de diversas pessoas dentre as quais agradeço.

Aos meus pais, pelo apoio incondicional em todos os momentos difíceis da minha trajetória acadêmica.

A minha esposa, pelo apoio, incentivo a realização da minha graduação.

Ao meu orientador, que me mostrou os caminhos a serem seguidos e pela confiança depositada.

A todos os professores e colegas do curso de Engenharia da Computação, que compartilharam essa fase de intensa aquisição de conhecimento.

Por fim, a todos os que de alguma forma contribuíram para a realização deste trabalho.

Resumo

As redes sociais são ferramentas cruciais para a integração de pessoas na era da informação. Entende-se como redes sociais web sites e aplicativos que permitem o compartilhamento de conteúdo e troca de mensagens entre pessoas de todo o mundo. Percebendo o valor e o alcance desses meios de comunicação, chegamos à conclusão que qualquer nova oportunidade de negócio deve ser acompanhada de uma estratégia de marketing digital bem estruturada. Para que esse objetivo seja alcançado é necessário estar presente em diferentes canais de comunicação que essas redes sociais disponibilizam para alcançar a maior gama de usuários e clientes. No entanto, a grande dificuldade dessa tarefa refere-se à gestão do conteúdo em diferentes interfaces de redes sociais, o que ocorre de forma individualizada. Desse modo, este trabalho tem como objetivo desenvolver um ambiente para integração e gerenciamento de conteúdo entre aplicativos de redes sociais utilizando o headless CMS Strapi para construção de uma API, com suporte a arquitetura REST e GraphQL, que funcionará como o back-end e o framework Flutter para o front-end. O ambiente será capaz de centralizar o envio de textos, imagens e vídeos em diferentes canais de comunicação. Além disso, será disponibilizada uma página Web no estilo Landing Page que será construída utilizando a linguagem Typescript juntamente com tecnologias modernas, como Next.js e React.

Palavras-chave: Flutter, Strapi, API REST, GraphQL, Next.js, React, TypeScript, Sistema de gerenciamento de conteúdo, Redes Sociais.

Abstract

Social networks are crucial tools for the integration of people in the information age. It is understood as social networks, websites, and applications that allow the sharing of content and exchange of messages between people from all over the world. Realizing the value and reach of these media, we came to the conclusion that any new business opportunities must be accompanied by a well-structured digital marketing strategy. For this objective to be achieved, it is necessary to be present in different communication channels that these social networks make available to reach the widest range of users and customers. However, the great difficulty of this task refers to the management of content in different interfaces of social networks, which occurs individually. Thus, this work aims to develop an environment for integration and content management between social network applications using the headless CMS Strapi to build an API, with support for the REST and GraphQL architecture, which will work as the back-end and the Flutter framework for the frontend. The environment will be able to centralize the sending of texts, images, and videos in different communication channels. In addition, a Landing Page style web page will be available, which will be built using the Typescript language along with modern technologies such as Next.js and React.

Keywords: Flutter, Strapi, REST API, GraphQL, Next.js, React, TypeScript, Content Management System, Social Networks.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CMS	*Content Management System (Sistema de gerenciamento de conteúdo)*
API	*Application Programming Interface (Interface de Programação de Aplicações)*
REST	*Representational State Transfer (Transferência Representacional de Estado)*
CRM	*Customer Relationship Management (Gestão de Relacionamento com o Cliente)*
HTML	*Hypertext Markup Language (Linguagem de Marcação de Hypertexto)*
URL	*Uniform Resource Locator (Localizador Uniforme de Recursos)*
AWS	*Amazon Web Services*
UML	*Unified Modeling Language (Linguagem de Modelagem Unificada)*

Sumário

1 Introdução

Vivemos na era da informação e a comunicação é uma das principais estratégias para o sucesso das organizações. O conceito de comunicação pode ser entendido como forma de transmissão de informações entre emissor e receptor com o objetivo de facilitar a criação de relacionamentos, o compartilhamento de ideias e as interações sociais. Enquanto a informação tem caráter unilateral, ou seja, não existe a preocupação ou obrigatoriedade de retorno, a comunicação pressupõe a necessidade de uma interação bilateral entre emissor e receptor. Nesse sentido, um meio de comunicação em massa difunde a informação a um público imenso e heterogêneo. Sob esse viés, a Internet pode ser considerada um meio de comunicação em massa como o rádio e TV, mas difere destas por possuir ferramentas que possibilitam alavancar o direcionamento da informação, a interatividade, relacionamento com o público, o alcance e mensuração de resultados com menos custos envolvidos (OLIVEIRA, 2020).

Segundo a pesquisa "Global Digital Overview 2020", realizada pelo site "We Are Social", o Brasil está em terceiro no ranking dos países que mais utilizam internet no mundo. A pesquisa diz ainda que 66% dos brasileiros já estão nas redes sociais gastando em média 3 horas e 31 minutos conectados por dia. (WE ARE SOCIAL, 2020).

Tendo em vista esse mercado, as redes sociais têm recebido crescente investimento organizacional. Esses investimentos são voltados principalmente para o monitoramento e gestão direta da interação de perfis comerciais com seus potenciais clientes. Nesse contexto, empreendedores utilizam diversos veículos de comunicação com o objetivo de fortalecer a imagem e o relacionamento perante o seu público alvo. (VERGILI, 2014).

Tendo em vista esta realidade, para uma estratégia de comunicação bem sucedida é notória a necessidade de estar presente nos mais principais canais de mídias e redes sociais existentes, já que esses canais apresentam uma oportunidade de escalar a comunicação exponencialmente. Sob essa lógica, o principal desafio é gerir diferentes contas individuais nos canais de comunicação, tendo em vista a necessidade de uma interação contínua com o aplicativo ou ferramenta para garantir a comunicação com os usuários.

Deste modo, para otimizar a gestão e a consistência nos diversos canais de mídias sociais é necessário que haja ferramentas que propiciem a gestão integrada desses canais, fornecendo também aos administradores de perfis comerciais, a possibilidade de criação de conteúdo que possa adaptar-se aos diferentes cenários das redes como Instagram, Youtube e Twitter. Além disso, é essencial gerir a interação dos usuários com o conteúdo e mensurar o engajamento em cada tipo de canal a fim de gerar métricas que possam auxiliar esses geradores de conteúdo a otimizar as suas estratégias de comunicação.

1.1 Objetivos

1.1.1 Objetivo Geral

O presente trabalho tem como objetivo geral disponibilizar um ambiente centralizado para integração e gerenciamento de conteúdo em diferentes canais de mídias sociais, e dessa forma, auxiliar empreendedores na gestão de suas contas em diferentes mídias sociais otimizando o tempo gasto para disponibilização de conteúdo e agilizando o atendimento a potenciais clientes que interagem com o perfil comercial da marca.

1.1.2 Objetivos Específicos

- Analisar e especificar uma API, com suporte a REST e GraphQL, para integração entre diferentes APIs de mídias sociais.
- Integrar a API com uma aplicação móvel para agir como interface com o usuário final.
- Disponibilizar uma página Web no estilo Landing Page para agir como porta de entrada para a utilização do aplicativo.

1.2 Organização do Trabalho

Este trabalho está estruturado da seguinte forma: O Capítulo 2 descreve as tecnologias utilizadas para solução do problema proposto. O Capítulo 3 se encarrega de apresentar as metodologias utilizadas no desenvolvimento da ferramenta. O Capítulo 4 apresenta a modelagem, a descrição do funcionamento dos Frameworks utilizados, o passo a passo do desenvolvimento da aplicação e os resultados obtidos. E por fim, o Capítulo 5 apresenta as considerações finais sobre os resultados e trabalhos futuros.

2 Fundamentação Teórica e Tecnologias Utilizadas

O presente capítulo tem como objetivo apresentar a importância da presença em redes e mídias sociais e as justificativas para o desenvolvimento de um sistema de gerenciamento de conteúdo que possibilite a administração integrada de diferentes canais de comunicação. Nesse contexto, apresentar as principais ferramentas que incentivaram a elaboração desse tipo de sistema.

2.1 Integração de mídias sociais para consolidação da presença digital de empreendedores

A construção de estratégias para estabelecer uma boa presença digital em um empreendimento é uma atitude considerada indispensável nos dias atuais. Prova disso são os contantes investimentos das empresas em profissionais especializados com o objetivo de gerir diferentes contas em ambientes digitais, demonstrando dessa forma a seriedade com que o assunto é tratado hoje em dia. Essa presença digital pode ser fundamentada na construção de relacionamentos com o público através de respostas rápidas e personalizadas em cada canal de mídia social, além do compartilhamento de conteúdo que gere valor e engajamento. (TELLES, 2010).

> "A comunicação integrada de marketing tem como finalidade unificar a comunicação da empresa para assegurar que a essência da mensagem seja a mesma, independente do formato do conteúdo ou os canais escolhidos (ANNIBAL, 2018).

Um CMS *(Content Management System)* é uma ferramenta que facilita a inserção de novos conteúdos em um web site ou blog, possibilitando o gerenciamento de conteúdo através de uma camada entre o usuário e o código fonte. O wordpress é um dos casos mais populares de CMS no mercado. Nele é possível criar integrações com redes sociais através de plugins de terceiros que possibilitam o compartilhamento de uma publicação diretamente nas redes sociais e também o disparo de e-mails para inscritos do site. (HOSTINGER, 2021).

Paralelamente existe outra ferramenta chamada de CRM *(Customer Relationship Management)* que é dedicada ao relacionamento com o cliente, que registra e organiza as origens e pontos de contato que um consumidor tem com o vendedor de um empreendimento. A grande vantagem dessa ferramenta é o registro do histórico do cliente, possibilitando ao

empreendedor entender mais sobre o seu público alvo e oferecer melhores soluções. Muitas vezes a origem desse registro inicial do cliente pode ocorrer através de uma mídia social ou interação com alguma publicação do empreendimento nas redes sociais. (RESULTADOS. . . , 2021).

Nos dias atuais, o atendimento ao cliente não está restrito a e-mails formais ou a um chat online próprio. Hoje ferramentas como Instagram e WhatsApp já são utilizadas como contato principal com o cliente e até mesmo comentários em vídeos da marca no Youtube podem ser o início de um relacionamento de negócio.

Nesse contexto, podemos inferir que o conceito de gestão de conteúdo disponível nas mídias sociais e o atendimento ao cliente estão interligados e, dessa forma, podemos unir os conceitos de CMS e CRM para gerar uma ferramenta que possibilite a gestão do conteúdo de um empreendimento em diversos canais de mídias sociais e o atendimento ao cliente através da interação centralizada com as mensagens de usuários nesses canais.

2.2 Sistema de gerenciamento de conteúdo

Pelo exposto, decidiu-se que um Sistema de gerenciamento de conteúdo é uma excelente opção para gestão de mídias sociais, visto que, por intermédio dessa ferramenta é possível integrar diferentes aplicativos e ferramentas de mídias sociais para disponibilização de conteúdo adaptável para cada canal e que alcance a maior quantidade de pessoas. Além disso, agregar a esse sistema conceitos de um sistema de gerenciamento de relacionamento com o cliente para permitir a comunicação bilateral nas redes, centralizando as interações de usuários nos diferentes canais em uma única plataforma possibilita um atendimento ágil e centralizado, unindo, dessa forma, a presença em todos os canais e a fidelização do cliente.

2.2.1 CMS Tradicional x Headless CMS

Um CMS tradicional consiste de um sistema de gerenciamento que possibilita a edição, armazenamento e apresentação do conteúdo. Já um Headless CMS, assim como o tradicional, possui uma base de dados para armazenamento centralizado, diferindo-se do primeiro pelo fato de que sua estrutura é baseada em APIS, possibilitando dessa forma que diversos canais digitais com suas próprias interfaces possam consumir e incluir conteúdos. O Headless CMS não possui interface para o usuário final, mas possibilita a integração com diferentes interfaces através das APIs que ele disponibiliza (LUMIS, 2021).

A utilização de um Headless CMS faz sentido por que temos o objetivo de integrar esse sistema de gerenciamento com outras aplicações, além de ter a possibilidade de capturar a jornada de interação com o cliente através de diferentes canais de comunicação.

Com esse escopo definido será mostrado a seguir as diversas linguagens, ferramentas, artefatos e conceitos que possibilitam a construção de um sistema de gerenciamento de conteúdo voltado para mídias sociais e atendimento ao cliente.

2.3 API

API *(Application Program Interface)* é um sistema back-end, ou seja, o sistema que contém as regras de negócio e a comunicação direta com o banco de dados, que possibilita a comunicação e o tráfego de dados entre diversos dispositivos (smartphones, tablets e notebooks). Estes utilizam diferentes aplicações (Web ou Mobile) que são escritas em diferentes linguagens de programação. (PEREIRA, 2016b)

Em outras palavras, as APIs possibilitam a integração entre sistemas que foram desenvolvidos separadamente. É através de tecnologias como esta que é possível compartilhar um artigo de um blog diretamente em uma rede social. Para que essa comunicação seja possível, é necessário que as aplicações utilizem o protocolo HTTP (Hypertext Transfer Protocol) juntamente com um estilo de arquitetura que define um conjunto de restrições que permitem a criação de uma API. Uma arquitetura amplamente utilizada é a REST (Representational State Transfer). (DEVMEDIA, 2021)

2.3.1 REST

A arquitetura REST utiliza uma interface de operações padronizadas do protocolo HTTP que tem como verbos principais o POST, GET, PUT e DELETE, que possibilitam criar, obter, atualizar e remover. Esses verbos são executados sob um recurso disponibilizado através de uma URL. A execução de uma operação que significa a utilização de um verbo sob um recurso obtém respostas do servidor que são classificadas em famílias de códigos de respostas dos quais podemos destacar as principais 2XX (Sucesso), 4XX (Erro no cliente) e 5XX (Erro HTTP no servidor). (ALGAWORKS, 2021)

2.3.2 GraphQL

Apesar da REST apresentar uma maneira sólida de trabalhar com APIs, novas tecnologias como o GraphQL estão surgindo para tentar otimizar esse processo. O GraphQL utiliza apenas um verbo HTTP que é o POST e apenas um endereço de recurso precisa ser chamado, que seria usualmente o /graphql. A forma com que essa linguagem de consulta para APIs consegue rotear para os recursos reais é através da passagem de parâmetros no corpo da requisição. Dentre os benefícios que essa abordagem proporciona destacam-se a eliminação da infinidade de urls de recursos que são apresentados no REST e uma consulta mais otimizada, já que no GraphQL os clientes têm o poder de pedir exatamente o que eles precisam, evitando o fluxo de dados desnecessários.(GAPHQL, 2021)

2.4 JavaScript

Segundo (ZAKAS, 2020) o JavaScript surgiu como uma forma de melhorar a experiência do usuário ao utilizar páginas Web, incorporando inicialmente, funcionalidades simples como por exemplo uma validação de formulário que poupava um acesso desnecessário ao servidor. Aos poucos, a linguagem foi ganhando cada cada vez mais espaço e acabou definindo a lógica de como as páginas Web devem se comportar.

O que inicialmente era uma linguagem de apoio no front-end, separado do HTML por suas tags <script> e </script>, hoje tornou-se uma linguagem muito completa e que possui diversos ecossistemas que possibilitam a criação de sistemas complexos que vão do front-end ao back-end.

2.5 TypeScript

O TypeScript foi criado pela Microsoft como um superset (ou superconjunto) do JavaScript. De acordo com (BANKS; PORCELLO, 2020), o TypeScript adiciona recursos na linguagem JavaScript como a tipagem estática de dados, que é uma característica essencial para detecção de erros durante o desenvolvimento e manutenção de aplicações modernas de grande escala.

2.6 Node.js

De acordo com (PEREIRA, 2016a) o Node.js é um ecossistema que permite a execução do JavaScript do lado do servidor, ou seja, fora de um navegador Web. Com ele é possível trabalhar com programação assícrona e construir APIs modernas com uma baixa curva de aprendizagem. Possui um gerenciador de pacotes, que vêm junto com a sua instalação original, que possui, em seu leque de possibilidades, milhões de módulos, possibilitando resolver rapidamente uma ampla gama de problemas no desenvolvimento de sistemas. Além disso, está sendo utilizado por gigantes da tecnologia como Uber, Netflix e Microsoft.

2.7 Strapi

O Strapi é uma ferramenta Headless CMS de código livre desenvolvida em Node.js que disponibiliza um painel administrativo simples e intuitivo para a construção de serviços REST, além de gerar uma documentação automática das operações. Além dessas vantagens, a ferramenta é totalmente personalizável, Self-hosted (permite o armazenamento local dos dados) e suporta o consumo da API via REST ou GraphQL. Outra vantagem é que o

Strapi possui nativamente toda a gestão de permissões e autorização de aplicativos clientes que pode ser facilmente configurada em seu painel administrativo (STRAPI, 2021).

2.8 Dart

Dart é uma linguagem moderna desenvolvida pelo Google em 2011 com o intuito de disponibilizar um ambiente multiplataforma aberto com alto desempenho e que possibilite uma alta produtividade. Inicialmente ela foi pensada para que fosse similar às principais linguagens conhecidas como Java, C-sharp e JavaScript para que fosse possível diminuir a sua curva de aprendizagem. Foi desenvolvida para rodar do lado do cliente e do servidor e além disso é compatível com uma ampla gama de navegadores, já que ela é compilada para o JavaScript, porém conta com uma performance consideravelmente melhor. (MITCHELL; AKOPKOKHYANTS; BALBAERT, 2017)

2.9 Flutter

O Flutter é um framework idealizado pelo Google para preencher uma lacuna que as tecnologias nativas e hibridas deixavam. Até então era necessário desenvolver de forma nativa para ganhar desempenho, ou seja desenvolver diversos aplicativos cada um com sua própria linguagem nativa e respectiva plataforma. Por outro lado, existem frameworks que sacrificam certo desempenho utilizando um tipo de mecanismo híbrido que mescla atributos de um aplicativo nativo e tecnologias web por exemplo. (MARINHO, 2020)

Segundo (MARINHO, 2020), a solução que o Flutter trouxe foi possibilitar o desempenho de uma aplicação nativa e, ao mesmo tempo, ser uma aplicação híbrida na qual é necessário escrever apenas um código fonte para poder executá-lo nas diferentes plataformas existentes como Android, iOS, Desktop e Web. Somado a isso temos ainda uma compilação mais rápida, fluidez, simplicidade na configuração e ganho considerável na velocidade de desenvolvimento.

2.10 Next.js

Next.js é um framework para a biblioteca React disponível para linguagem a Javascript e com suporte à linguagem TypeScript. O framework possibilita a construção de interfaces e adiciona poderosas funcionalidades além das disponíveis na biblioteca React. O Next.js trabalha com renderização estática e pelo lado do servidor, além de possuir um serviço de tratamento de rotas otimizado. O framework ainda melhora significativamente as pontuações na indexação de conteúdo pelos motores de busca. (KONSHIN, 2018).

3 Metodologias e Ferramentas

Este capítulo discorrerá sobre as metodologias adotadas neste projeto e os motivos que levaram às suas escolhas. Além disso, discorrerá sobre a arquitetura de desenvolvimento, bibliotecas e ferramentas utilizadas.

3.1 Metodologia de gerenciamento do Projeto

3.1.1 Kanban

Duas metodologias muito utilizadas no desenvolvimento de projetos são o Scrum e o Kanban. (ENGHOLM, 2017) aponta que o Scrum é uma metodologia ágil que possui ciclos de desenvolvimento de duas a quatro semanas chamadas de sprints. Essa metodologia prevê também a figura do Product Owner que é responsável de realizar o levantamento de requisitos junto ao cliente final e realizar reuniões periódicas de acompanhamento do projeto com um time de desenvolvedores.

Por outro lado, o Kanban é uma metodologia ágil que estrutura-se sobre um cartaz com cartões do tipo post-it. Nele o processo de trabalho é contínuo e é disposto em colunas agregadoras de tarefas. Cada coluna possui uma fila de entregáveis que pode ser migrada para a coluna posterior assim que os itens vão ficando prontos. (WAZLAWICK, 2019).

Segundo (WAZLAWICK, 2019), as tarefas podem receber prioridades diferentes. Nesse contexto, os itens com maior prioridade devem aparecer mais ao topo da coluna. Para essa priorização ainda podemos utilizar cartões coloridos para melhorar a visualização como, por exemplo, a cor vermelha para prioridades mais elevadas, amarelo para médias e verde para as mais baixas.

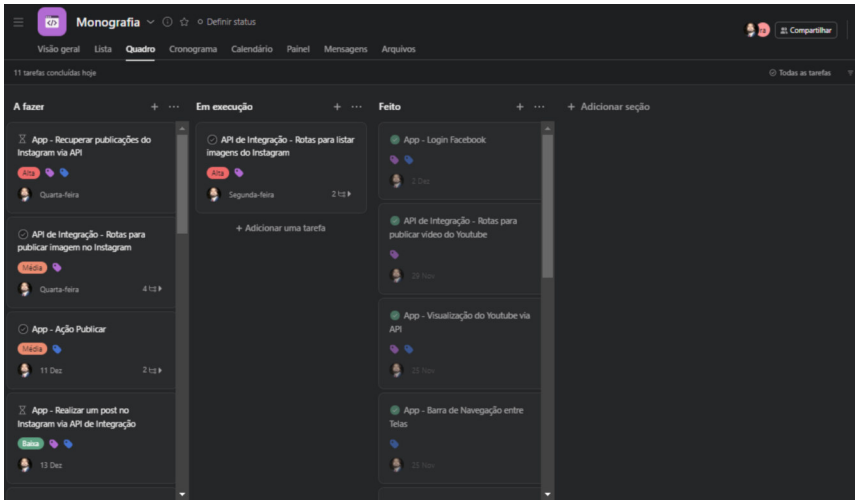
Um dos principais benefícios que essa metodologia proporciona é a visualização do fluxo de trabalho, já que é possível verificar o que há por fazer, o que está sendo feito e o que já foi feito. Além disso, é possível limitar os itens por coluna, evitando o acúmulo de tarefas e tornando o processo de desenvolvimento público para acompanhamento, mensuração, recebimento de feedbacks e implementação de melhorias contínuas. Optou-se por utilizar o Kanban por ser uma metodologia mais simples onde não há necessidade de reuniões diárias, sprints, ou papéis bem definidos como no Scrum e pelas características do projeto que conta apenas com um desenvolvedor.

3.1.2 Aplicativo de Gerenciamento de Projeto

A ferramenta online Asana foi utilizada para apoiar a gestão deste projeto. Ela utiliza majoritariamente os conceitos do Kanban. O fluxo de desenvolvimento foi organizado da seguinte forma:

- A Fazer - Lista de tarefas à fazer baseada nos requisitos.
- Em Execução - Lista de tarefas que estão em desenvolvimento.
- Feito - Lista de tarefas finalizadas.

Figura 1 – Utilização da Ferramenta Asana como Aplicativo de Gestão do Projeto

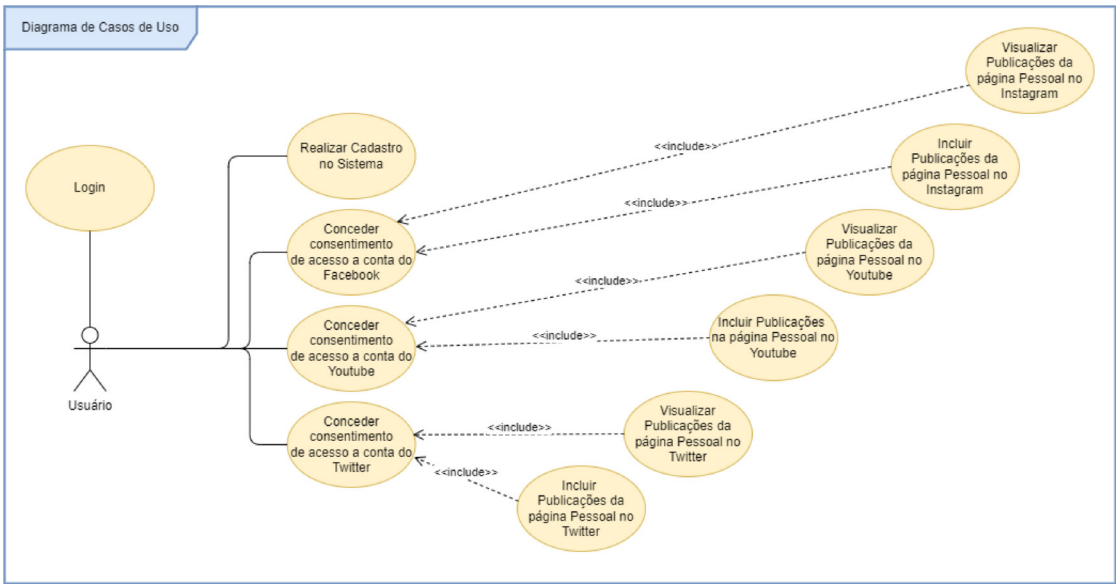


Fonte: O Autor

3.2 Diagrama Casos de Uso

O Diagrama de Casos de Uso possibilita a compreensão geral dos sistemas através da representação gráfica das funcionalidades que o sistema deve apresentar quando o mesmo interage com outro sistema externo ou com atores. O ator é aquele que espera um valor observável ao realizar uma interação e possui um determinado papel dentro do sistema.(GUEDES, 2018). Para a representação dos casos de uso utilizaremos um diagrama, conforme é especificado pela UML (Unified Modeling Language), que pode ser visualizado na Figura 2 e o detalhamento de cada um deles pode ser encontrado nas Tabelas 7-16 no apêndice A.

Figura 2 – Diagrama de Casos de Uso

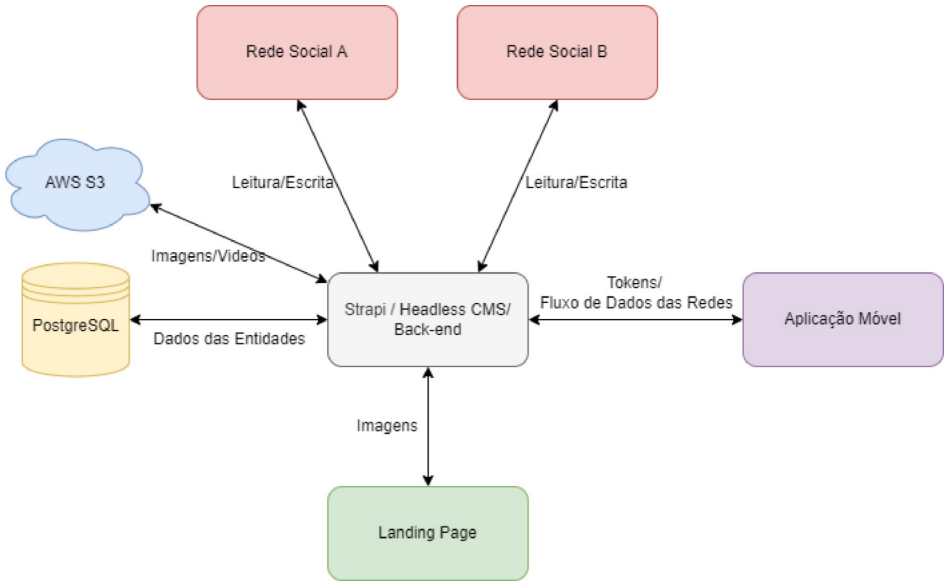


Fonte: O Autor

3.3 Arquitetura do Projeto

Nesta seção, iremos apresentar como foi organizado o fluxo de informações do projeto e discorrer sobre a organização de sua arquitetura interna. Além disso, iremos apresentar as bibliotecas, ferramentas utilizadas e as configurações principais que antecedem o processo de codificação da aplicação. A Figura 3 apresenta a arquitetura macro da solução e o fluxo de dados entre os sistemas.

Figura 3 – Arquitetura do Projeto



Fonte: O Autor

Tabela 1 – Descrição dos Componentes da Arquitetura

Componente	**Descrição**
Strapi	Headless CMS que será o back-end capaz de realizar a comunicação entre as APIs das redes sociais, banco de dados e provedor de arquivos estáticos.
Aplicação Móvel	Aplicação móvel onde será possível gerir as redes sociais. Possibilita a comunicação com o back-end para envio de tokens de acesso às redes sociais, envio e recebimento de dados.
Landing Page	Página no estilo Landing Page. As imagens apresentadas no site devem ser recuperadas do provedor de estáticos através do back-end Strapi.
PostgreSQL	Banco de Dados do Strapi. Serve para armazenar os dados das entidades criadas.
AWS S3	Provedor de arquivos estáticos. A sua finalidade no projeto é armazenar imagens e vídeos.
Rede Social A e B	Redes sociais que serão integradas ao sistema.

3.4 Frameworks/Bibliotecas e Ferramentas

Nesta subseção serão apresentadas as ferramentas, frameworks e bibliotecas que são utilizadas no projeto.

3.4.1 Editor de Código Fonte - Visual Studio Code

Para criação do código fonte desse projeto foi utilizado o Visual Studio Code. Esse editor de código fonte é gratuito e possui diversas funcionalidades que garantem uma significativa melhoria de desempenho durante o desenvolvimento de aplicações como depuração de código, comandos git integrados, realce de sintaxe e preenchimento automático (VISUAL. . . , 2021).

3.4.2 API Client - Postman

O Postman é um software gratuito que facilita os testes de consumo de APIs. Além disso ele pode ser utilizado também para documentar as APIs. Com ele é possível realizar chamadas HTTP e HTTPs de todos os níveis de complexidade e visualizar as suas respostas.(POSTMAN, 2021).

3.4.3 Frameworks e Bibliotecas

A Tabela 12 traz a lista de todos os frameworks e bibliotecas utilizadas na construção do back-end. Foi utilizado o gerenciador de pacotes "yarn"para a instalação.

Tabela 2 – Frameworks e Bibliotecas : Back-end Strapi

Biblioteca/Framework	**Versão**	**Função**
strapi	3.6.8	Pacote do headless CMS.
strapi-admin	3.6.8	Módulo de administração do Strapi,
strapi-plugin-ckeditor	1.13.0	Inserir textos com formatação e imagens.
strapi-plugin-content-manager	3.6.8	Módulo de gestão de conteúdo.
strapi-content-type-builder	3.6.8	Construção das entidades da API.
strapi-plugin-documentation	3.6.8	Criar um documentação no padrão OpenAPI e a interface do Swagger.
strapi-plugin-email	3.6.8	Possibilita o envio de e-mails com templates.
strapi-plugin-entity-relationship-chart	3.1.0	Gera um gráfico das relações entre as entidades geradas.
strapi-plugin-graphql	3.6.8	Possibilita o consumo da API no padrão GraphQL.
strapi-plugin-upload	3.6.8	Possibilita o upload de arquivos e imagens através do módulo de administração.
strapi-plugin-users-permissions	3.6.8	Possibilita a gestão de permissões dos usuários.
strapi-provider-upload-aws-s3	3.6.8	Gera a conexão com o provedor AWS S3.
pg	8.5.1	Possibilita a integração com o banco de dados PostgreSQL.
googleapis	92.0.0	Biblioteca que facilita o consumo da API do Google.
crypto	1.0.1	Possibilita a manipulação de dados criptografados.
oauth-1.0a	2.2.6	Facilita as requisições a operações que utilizam oAuth 1.0.
axios	0.24.0	Cliente HTTP para realização de chamadas a uma API.

A Tabela 13 traz a lista de todos os frameworks e bibliotecas utilizadas na construção do front-end Flutter. Foi utilizado o comando nativo do flutter para a instalação.

Tabela 3 – Frameworks e Bibliotecas : Front-end Flutter

Biblioteca/Framework	**Versão**	**Função**
cupertino_icons	1.0.4	Pacote de ícones.
cached_network_image	3.2.0	Possibilita mostrar imagens através de uma URL e armazená-las em cache.
material_design_icons_flutter	5.0.695	Pacote de ícones.
http	0.13.4	Biblioteca para realizar requisições HTTP.
youtube_player_flutter	8.0.0	Plugin para reproduzir vídeos do Youtube.
flutter_facebook_auth	3.5.7	Possibilita a integração com o Login do Facebook.
get	4.6.1	Pacote utilizado para gerência de estados.
image_picker	0.8.4+4	Possibilita a captura de imagem através da câmera do dispositivo ou da galeria.
google_sign_in	5.2.1	Possibilita a integração com o Login do Google.
provider	6.0.2	Pacote utilizado para gerência de estados de objetos.
video_player	2.2.11	Possibilita a execução de vídeos em segundo plano através de uma URL.
flutter_screenutil	5.0.3	Utilizado para adaptar o tamanho da fonte de acordo com o tamanho da tela.
twitter_login	4.0.1	Possibilita a integração com o Login do Twitter.

A Tabela 14 traz a lista de todos os frameworks e bibliotecas utilizadas na construção do front-end Next.js. Foi utilizado o gerenciador de pacotes "yarn"para a instalação.

Tabela 4 – Frameworks e Bibliotecas : Front-end Next.js

Biblioteca/Framework	**Versão**	**Função**
next	11.1.2	Framework React desenvolvido pela Vercel.
next-pwa	5.3.1	Plugin que adiciona funcionalidades de PWA para aplicação.
react	17.0.2	Biblioteca Javascript para construção de interfaces.
react-bootstrap	2.1.1	Componentes Bootstrap 5 para aplicações React.
storybook-addon-next-router	2.0.4	Integração entre as rotas do Next.js e o Storybook.

Tabela 4 continuação da página anterior

Biblioteca/Framework	Versão	Função
styled-components	5.3.1	Biblioteca base para construção de componentes personalizados.
typescript	4..4.2	Possibilita a utilização do Typescript no projeto.
prettier	2.4.0	Ferramenta para formatação automática de código.
husky	7.0.0	Ferramenta que possibilita a criação de hooks no Github.
jest	27.1.0	Ferramenta de testes de código.
jest-styled-components	7.0.5	Ferramenta de testes de componentes.
lint-staged	11.1.2	Ferramenta que impede que código não testado ou com problemas seja inserido no Github.

3.5 Sistema de Controle de Versão

Para o controle de versões do projeto utilizamos o gerenciador de repositórios Github, que utiliza o sistema de controle de versões Git (GITHUB..., 2021). Um repositório remoto possibilita que vários desenvolvedores atuem em diferentes partes de um mesmo projeto. Após uma modificação as atualizações realizadas no código fonte são enviadas ao repositório e todos os outros desenvolvedores terão acesso ao código mais atual. O primeiro passo é instalar o Git (GIT..., 2021a) no sistema operacional. Posteriormente foi instalado a ferramenta Git Bash (GIT..., 2021b) que foi o terminal utilizado para trabalhar com linha de comando no sistema Windows. Também foi utilizada a ferramenta TortoiseGit (TORTOISEGIT..., 2021) que é uma interface gráfica que facilita a execução de vários comandos comuns do git com poucos cliques no diretório do projeto. Com essas ferramentas à disposição chega o momento de configurar os acessos necessários para trabalhar com um repositório existente no Github. Para isso é necessário a definição de um usuário e e-mail e posteriormente gerar uma chave ssh para adicionar na nossa conta do Github e assim concluir a verificação de credenciais de acesso.

4 Desenvolvimento e Resultados Obtidos

Este capítulo é dividido em quatro seções áreas. Primeiramente iremos tratar do desenvolvimento do back-end utilizando o Strapi. Em seguida será apresentado o desenvolvimento do aplicativo Flutter. Posteriormente será tratado do desenvolvimento da Landing Page e por fim iremos tratar dos resultados obtidos. A finalidade do capítulo é apresentar os resultados obtidos ao adotar as referências teóricas, o modelo de desenvolvimento, a utilização das ferramentas e as metodologias que foram apresentadas nos Capítulos 2 e 3.

4.1 Desenvolvimento

Nessa sessão iremos apresentar o desenvolvimento prático e codificação do back-end Strapi, aplicativo Flutter, e para finalizar, o desenvolvimento da Landing Page em Next.js. Serão apresentadas também suas telas e funcionalidades em ação.

4.1.1 Strapi

Nessa subseção será abordada a construção da API que funcionará como back-end da aplicação Flutter. Para isso foi utilizado o Strapi, que possibilita a disponibilização instantânea de APIS para leitura e escrita de tabelas de um banco de dados. Além disso, já possui toda a segurança necessária nesse tipo de aplicação como OAuth 2.0 e Perfis de acesso.

4.1.1.1 Banco de Dados PostgreSQL

O Banco de dados utilizado para a conexão com o back-end foi o PostgreSQL. (POSTGRES, 2021). Após a instalação podemos abrir o Query Tool da página de Administração do PostgreSQL e executar os comandos de criação de usuário, database e conceder os privilégios do banco de dados local conforme podemos visualizar na Figura 4.

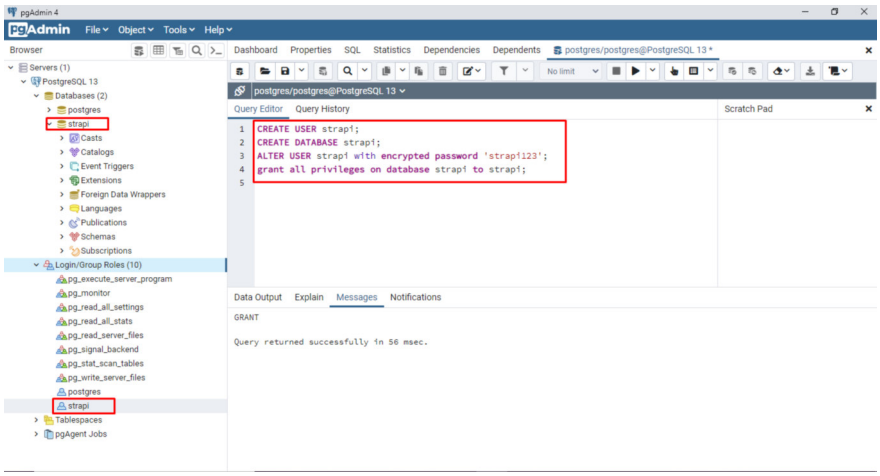
4.1.1.2 Configurando o Strapi

Para criação do projeto Strapi devemos possuir o Node.js e o gerenciador de pacotes npm ou yarn instalados para que seja possível rodar no terminal o comando para instalação do pacote do strapi. (STRAPI. . . , 2021a)

```
$ yarn add strapi
```

Agora podemos rodar o comando que irá gerar a estrutura de diretórios do projeto base.

Figura 4 – pgAdmin do PostgreSQL



Fonte: O Autor

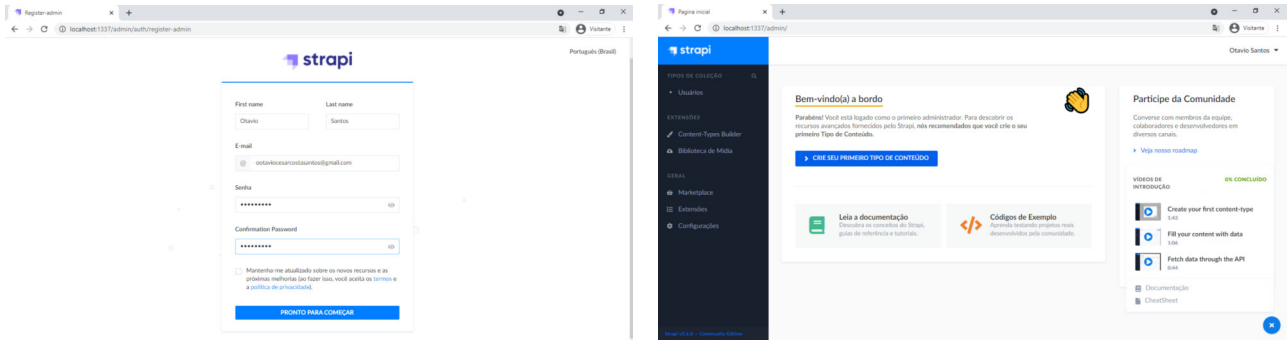
```
$ yarn create strapi-app streaming-api
```

Em seguida aparecerá a opção de escolha do banco de dados PostgresSQL que foi previamente configurado. Após a confirmação podemos rodar a aplicação e navegar na página de administração do Strapi.

```
$ yarn develop
```

Para o primeiro acesso é necessário criar um usuário para navegar na página inicial do dashboard de administração do Strapi, como podemos visualizar na Figura 5.

Figura 5 – a) Criando Usuário Strapi e b) Página inicial do Strapi.

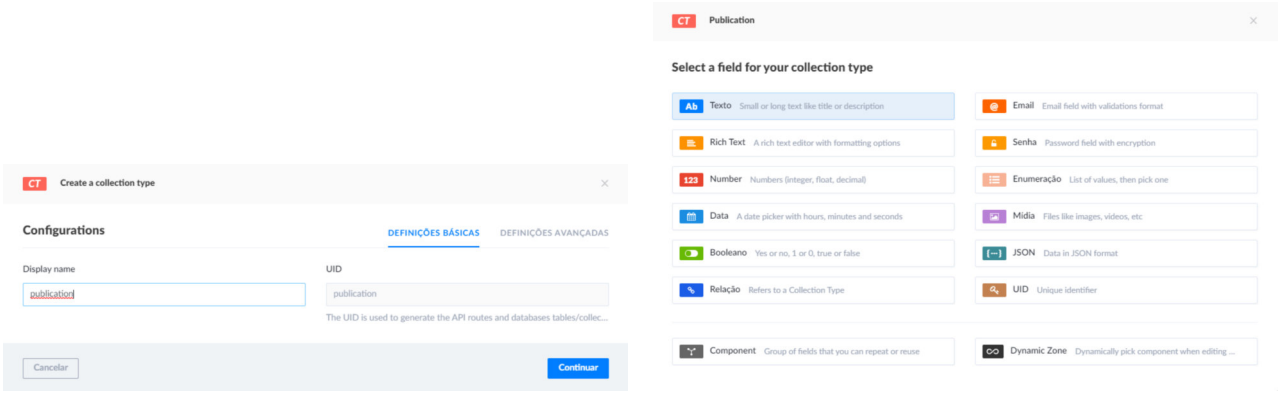


(*a*) (*b*)

Fonte: O autor

Agora podemos iniciar a modelagem das entidades do banco de dados, conforme mostrado nas Figuras 6, 7 e 8.

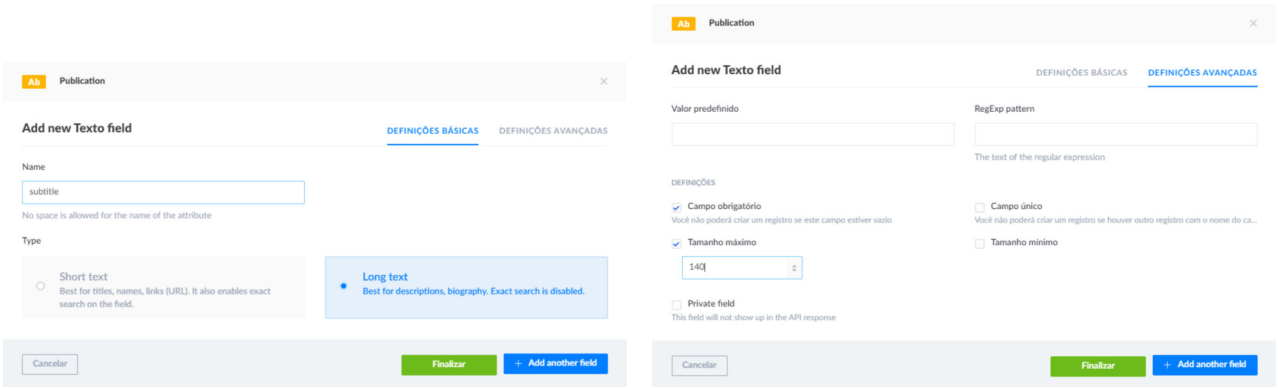
Figura 6 – a) Criando Entidades e b) Atributos disponíveis.



(*a*) (*b*)

Fonte: O autor

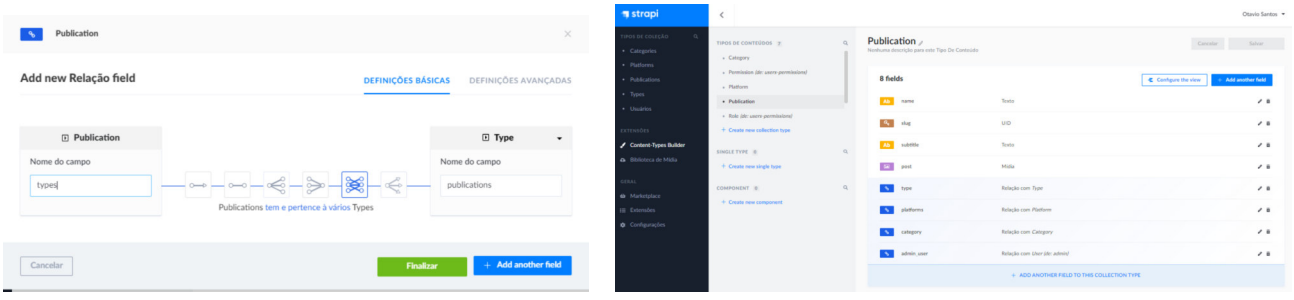
Figura 7 – a) Criando um atributo e b) Configurações avançadas do atributo.



(*a*) (*b*)

Fonte: O autor

Figura 8 – a) Criando Relacionamentos entre Entidades e b) Entidade Finalizada.



(*a*) (*b*)

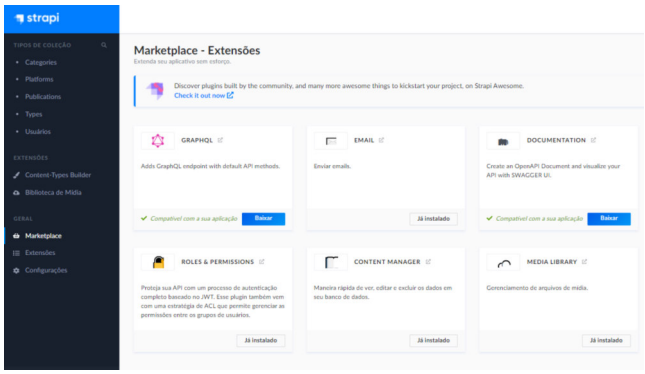
Fonte: O autor

4.1.1.3 GraphQL

Na instalação inicial do Strapi conseguimos visualizar o menu Marketplace. Nesse menu estão disponíveis algumas extensões úteis. Uma delas é a que fornece a funcionalidade

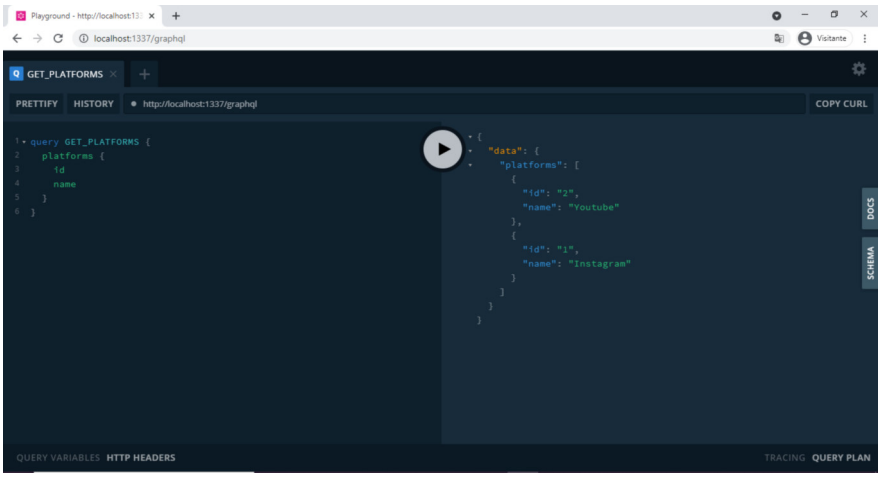
de GraphQL aos endpoints. Após a instalação da extensão todas as operações REST agora também atenderão ao padrão GraphQL. É disponibilizada ainda uma nova rota /graphql que pode ser utilizada para os testes do consumo da API GraphQL como mostram as Figuras 9 e 10.

Figura 9 – Habilitando funcionalidade de GraphQL



Fonte: O Autor

Figura 10 – Consumindo a API GraphQL



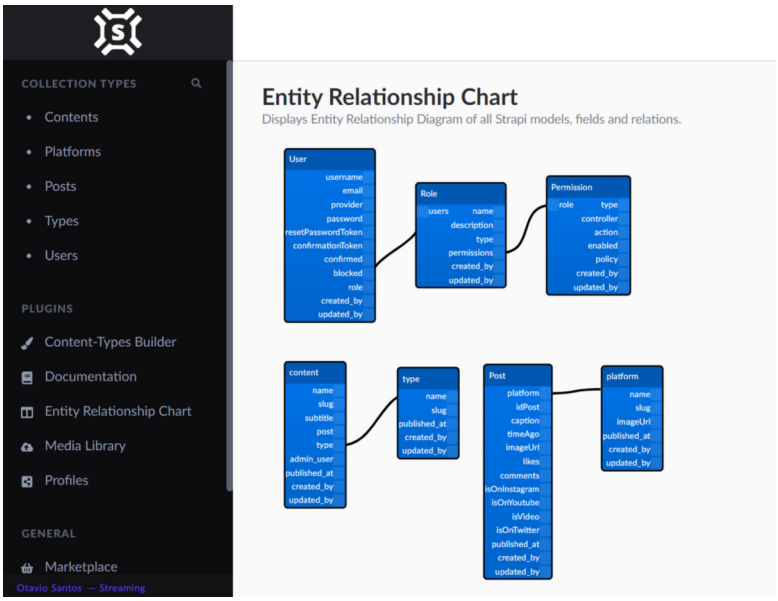
Fonte: O Autor

4.1.1.4 Personalizando o CMS

Para expandir as funcionalidades do framework, é possível instalar plugins desenvolvidos pela comunidade. O primeiro plugin que utilizamos foi o Entity Relationship Chart, que possibilita visualizar graficamente as entidades criadas e os seus respectivos relacionamentos como mostrado na Figura 11.

```
$ yarn add strapi-plugin-entity-relationship-chart
```

Figura 11 – Plugin Entity Relationship Chart

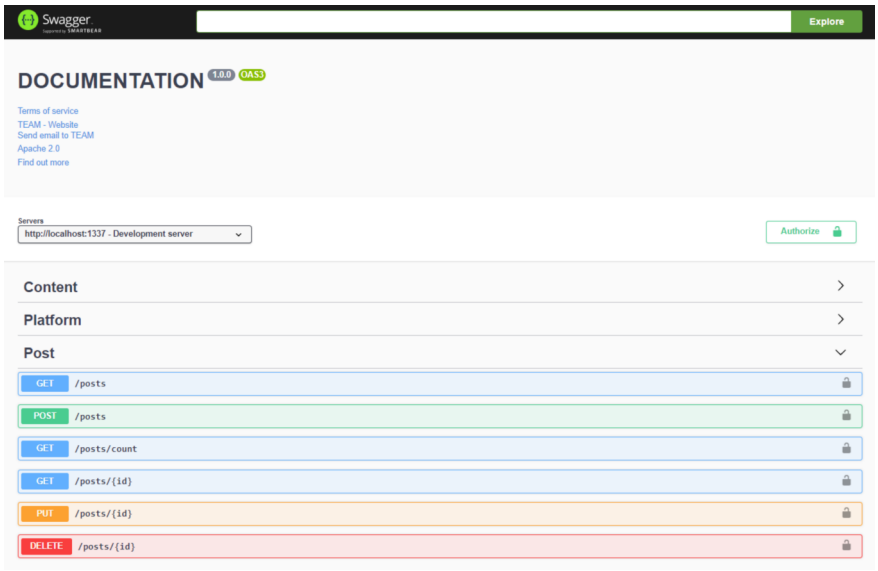


Fonte: O Autor

O Segundo plugin instalado foi o Documentation. Esse plugin disponibiliza a documentação da API utilizando a especificação OpenAPI e possibilita visualizar as operação com a interface Swagger UI como mostrado na Figura 12.

```
$ yarn strapi install documentation
```

Figura 12 – Documentação da API



Fonte: O Autor

4.1.1.5 Implementação da camada de Regras de Negócio

Conforme podemos perceber, o Strapi possibilita a construção de uma API de forma extremamente rápida com todas as atribuições esperadas. Os seus verbos HTTP GET, POST, PUT, PATCH e DELETE estão disponíveis para utilização via REST e as mesmas entidades podem ser consumidas via GraphQL. Mas, na maioria dos casos, isso não é suficiente e o desenvolvedor necessita de mais controle durante essas interações. Para isso a documentação do Strapi explica a forma que podemos realizar a personalização das camadas de Modelo, Controladores e de Serviços.

4.1.1.5.1 Customização de Controllers

O controller é a porta de entrada da rota que foi chamada. Inicialmente é aqui que mantemos parte da nossa lógica de negócio. Caso a complexidade aumente e comecem a aparecer métodos que podem ser reutilizáveis, podemos migrar parte da lógica para a camada de serviços.

Para realizar esse tipo de personalização, de acordo com a documentação do Strapi, basta navegar até o diretório da entidade desejada, acessar o diretório de controllers e adicionar no arquivo que dá nome a entidade o exemplo do método que deseja-se manipular de acordo com as instruções da Figura 13.

Figura 13 – Customização de Controllers

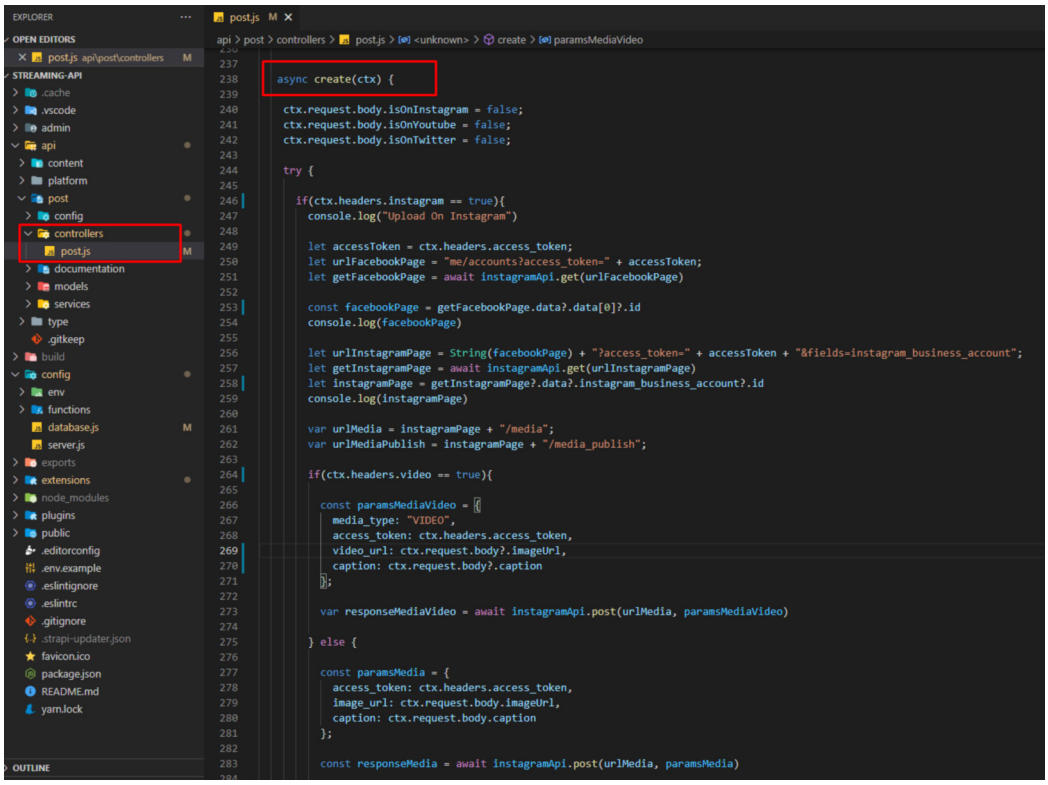
Fonte: (STRAPI. . . , 2021b)

Ainda de acordo com a Figura 13, podemos perceber que o método possui o parâmetro "ctx". Esse parâmetro contêm as variáveis de contexto da operação e através dele podemos acessar as informações contidas no header, body e queryParams, possibilitando dessa forma, manipulações dos dados da requisição e execução de outras tarefas (inclusive chamadas a outras APIs externas e internas) antes de seguir com o fluxo principal.

Posteriormente, temos a variável response que será o retorno que o back-end irá apresentar e, da mesma forma, podemos acessar os status de resposta da chamada e o possível payload de resposta, ou até mesmo se preferirmos montar uma resposta final

totalmente. Na Figura 14 podemos observar em ação a utilização dos recursos dessa personalização do nosso back-end para interagir com as diversas APIs de Redes Sociais previstas no projeto.

Figura 14 – Personalização de um Controller



Fonte: O Autor

4.1.1.6 Deploy da API

A última etapa do desenvolvimento do back-end é a implantação no ambiente de produção. Para realizar tal tarefa, foi escolhida a plataforma de serviço em nuvem Heroku. O Strapi possui em sua documentação os requisitos necessários para implantação em diversos provedores incluindo o Heroku (STRAPI. . . , 2021c). O primeiro passo é realizar a instalação do Heroku CLI. Em seguida executar o comando de login. Uma página será então aberta para realizarmos o login na plataforma do Heroku. Uma vez autenticados na máquina podemos rodar o comando e criar a instância que vai servir a nossa aplicação no Heroku.

```
$ heroku login
```

```
$ heroku create api-streaming-integration
```

O Heroku possui um plano gratuito para criação de uma instância de banco de dados do PostgreSQL. Para realizar a configuração devemos rodar o seguinte comando.

```
$ heroku addons:create heroku-postgresql:hobby-dev
```

Em seguida executamos o comando heroku config para verificar a URL do banco de dados criado.

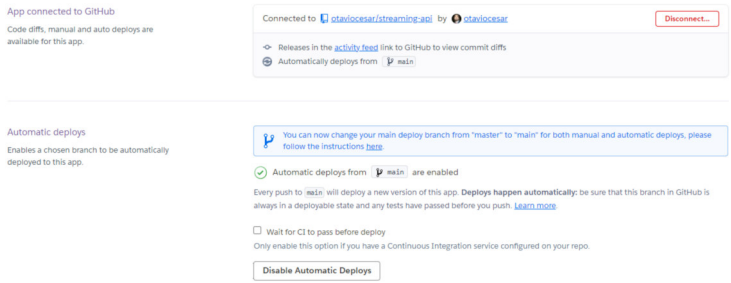
```
$ heroku config --app api-streaming-integration
=== api-streaming-integration Config Vars
DATABASE_URL: postgres:// USERNAME : PASSWORD @ HOST : PORT / DATABASE_NAME
```

Por fim, podemos executar a sequência de comandos abaixo para configurar as variáveis de ambiente para acesso ao banco de dados do projeto com todas as suas definições que foram retornadas na etapa anterior.

```
$ heroku config:set DATABASE_USERNAME=USERNAME --app api-streaming-integration

$ heroku config:set DATABASE_PASSWORD=PASSWORD --app api-streaming-integration

$ heroku config:set DATABASE_HOST=HOST --app api-streaming-integration

$ heroku config:set DATABASE_PORT=PORT --app api-streaming-integration

$  heroku config:set DATABASE_NAME=DATABASE_NAME --app api-streaming-integration
```

No Heroku temos a possibilidade de vincular a nossa conta no Github e associar o código fonte do projeto disponível no Github para deploy automático na instância criada no Heroku. Essa etapa foi realizada de acordo com a Figura 15.

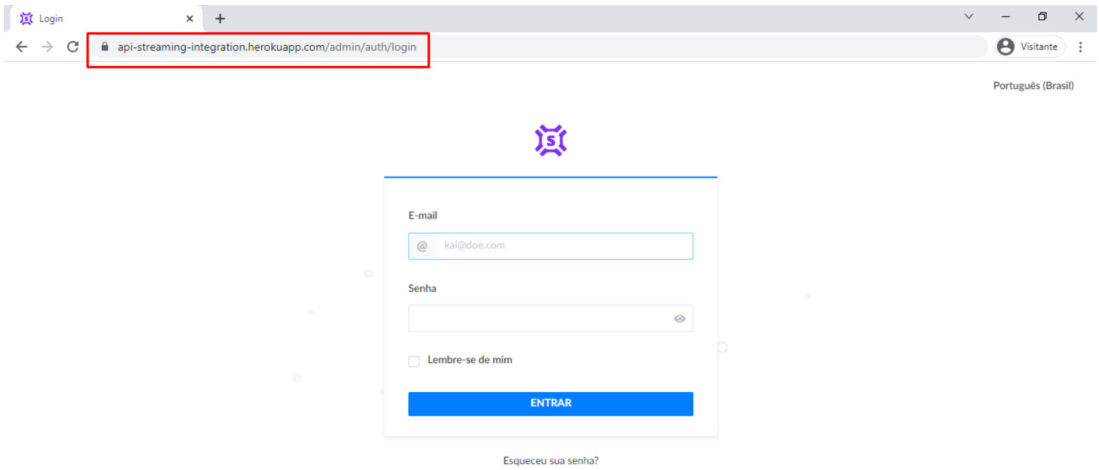
Figura 15 – Conectando Github para deploy automático



Fonte: O Autor

Após o deploy realizado com sucesso, o nosso back-end ficou disponível em um domínio terminado em .herokuapp.com como podemos visualizar na Figura 16.

Figura 16 – Deploy Realizado com sucesso



Fonte: O Autor

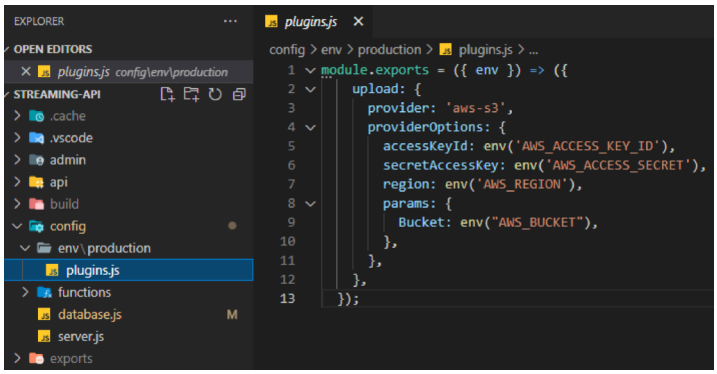
4.1.1.7 Provedor de Estáticos

Durante o desenvolvimento da aplicação Strapi, o módulo de gestão de conteúdo grava os arquivos localmente. Dessa forma, após o deploy da aplicação, notou-se a necessidade de adicionar o provedor de arquivos Amazon S3 (Amazon Simple Storage Service) para disponibilizar os arquivos em forma de URL para posterior manipulação. Para isso, foi criada uma conta na Amazon Web Services (AMAZON. . . , 2021) e rodamos os seguintes comandos para configurar no Heroku as variáveis de ambiente de acesso ao serviço criado na Amazon.

```
$ yarn add strapi-provider-upload-aws-s3

$ heroku config:set AWS_ACCESS_KEY_ID=XXXX --app api-streaming-integration

$ heroku config:set AWS_ACCESS_SECRET=XXXX --app api-streaming-integration

$ heroku config:set AWS_REGION=sa-east-1 --app api-streaming-integration

$ heroku config:set AWS_BUCKET=XXXX --app api-streaming-integration
```

Para finalizar devemos criar o arquivo plugins.js dentro do diretório env/production e adicionar a configuração apresentada na Figura 17.

Figura 17 – Configuração do Provedor AWS no projeto



Fonte: O Autor

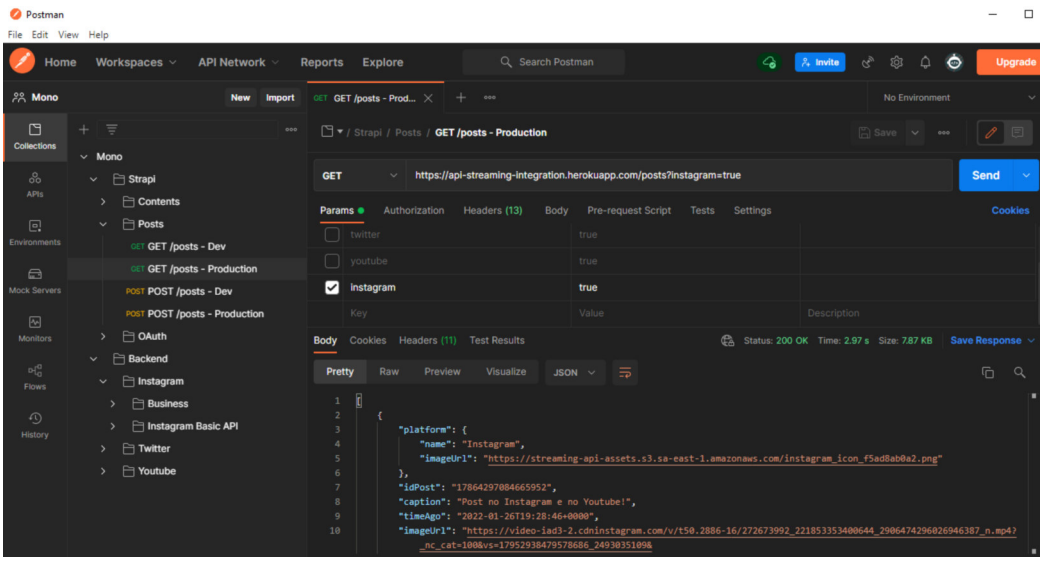
A partir de agora as URLs dos arquivos que forem incluidos via API ficaram com o seguinte formato:

`https://streaming-api-assets.s3.sa-east-1.amazonaws.com/[nome-do-arquivo]`

4.1.1.8 Testes de APIs com Postman

Foi criada uma collection no sofware Postman para testar as operações que foram desenvolvidas na API Strapi. Paralelamente foram criados testes de chamadas às APIs das redes sociais separadamente para entender o modelo de dados que é retornado em cada caso. Com esses dados em mãos foi possível personalizar o back-end para chamar as APIs das redes sociais de acordo com os parâmetros informados na requisição, conforme podemos visualizar na Figura 18.

Figura 18 – Testes de APIs com Postman



Fonte: O Autor

4.1.2 App Flutter

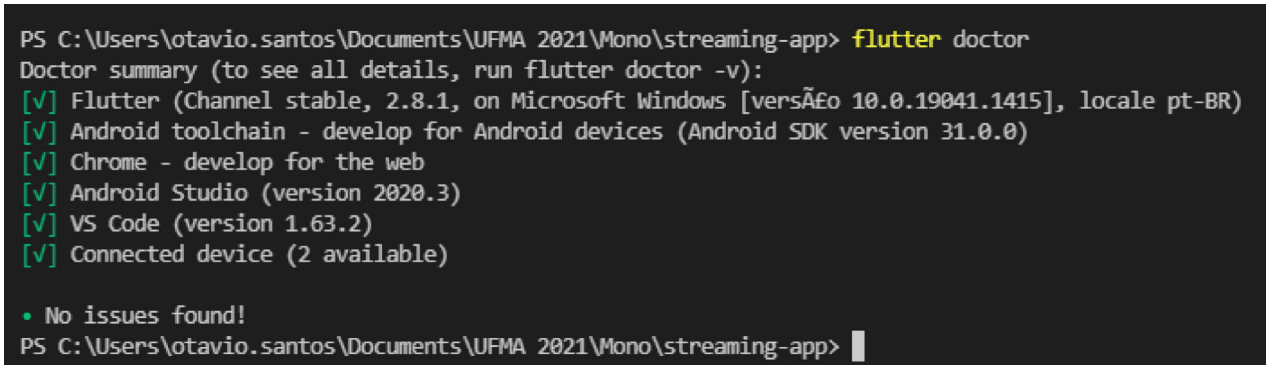
Nessa subseção iremos tratar do desenvolvimento do Aplicativo utilizando o kit de desenvolvimento Flutter e a linguagem de programação Dart. A primeira etapa para criar o ambiente de programação do aplicativo é instalar o Flutter na máquina através do link disponível no site oficial (FLUTTER. . . , 2021).

Após a instalação, teremos disponível no terminal o comando "flutter". Então podemos executar os comandos abaixo para aceitar os termos de licença do android e em seguida verificarmos o que está faltando configurar para iniciarmos o projeto

```
$ flutter doctor --android-licenses
$ flutter doctor
```

Como podemos verificar na lista de dependências da Figura 19, é recomendado instalarmos também o navegador do Google Chrome(CHROME, 2021), o Android Studio(ANDROID. . . 2021). e o Visual Studio Code (VISUAL. . . , 2021).

Figura 19 – Resultado da execução do comando flutter doctor



Fonte: O Autor

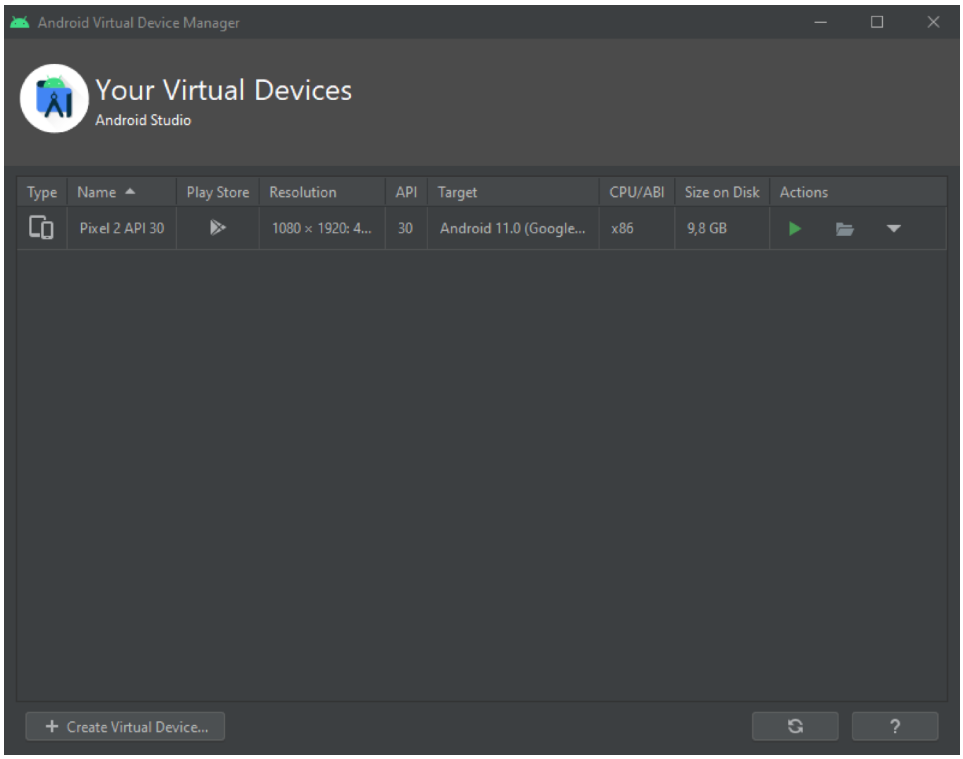
Ao realizar a instalação do Android Studio podemos incluir um emulador do Android acessando o AVD Manger conforme mostra a Figura 20.

Para executar o emulador criado basta executar o comando abaixo o veremos em ação conforme a Figura 21.

```
$ flutter emulators --launch Pixel_2_API_30
```

Para criar o projeto podemos utilizar o comando create seguido do nome do projeto. Na Figura 22 podemos observar a estrutura de diretórios do projeto Flutter que foi gerada através do comando anteriores e na Tabela 15 a descrição dos principais.

Figura 20 – Criando Emulador do Android



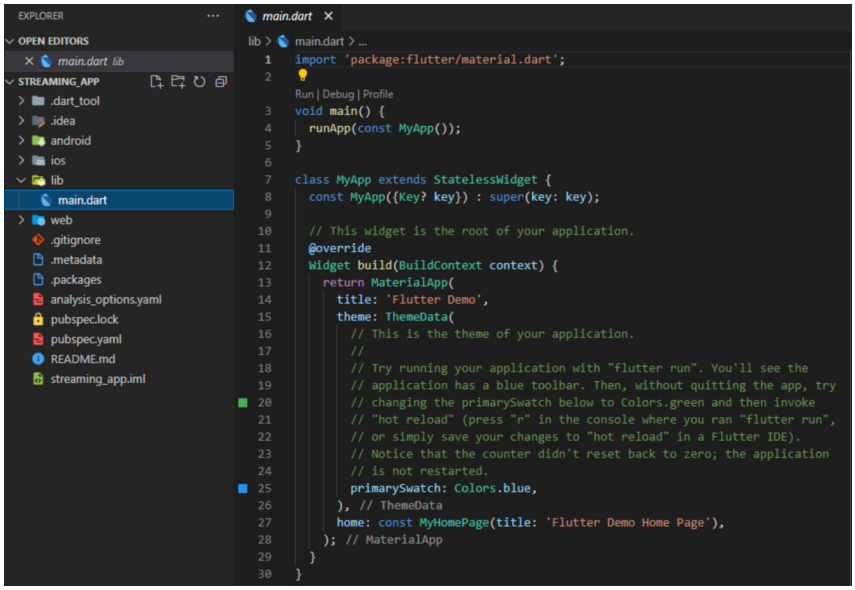
Fonte: O Autor

Figura 21 – Executando o Emulador Android

Fonte: O Autor

```
$ flutter create streaming_app
```

Figura 22 – Estrutura de Diretórios de um Projeto Flutter



Fonte: O Autor

Tabela 5 – Descrição dos Diretórios do Projeto Flutter

Diretório	**Descrição**
android	Contém arquivos para o funcionamento do app na plataforma Android.
ios	Contém arquivos para o funcionamento do app na plataforma iOS.
lib	Contém os arquivos .dart Temos a liberdade de criar a estrutura de diretórios que desejarmos porém o arquivo main.dart precisa estar na raiz.
main.dart	Aqui é o ponto de partida da aplicação e deve conter a chamada ao método main().
pubspec.yaml	Arquivo que contêm diversas informações e configurações do app além dos pacotes/dependências.

Ao criar o projeto inicialmente podemos verificar que no arquivo main.dart já é gerado um código de demonstração. Para executar a aplicação no emulador devemos executar o comando abaixo.

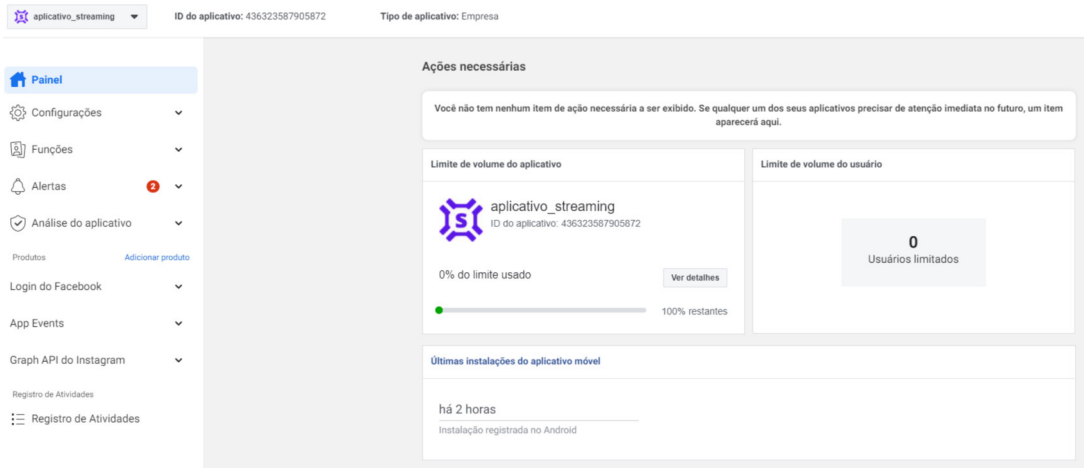
```
$ flutter run
```

Agora podemos realizar pequenas alterações no código e verificar as alterações em tempo real utilizando o comando r para hot reload (comando utilizado para pequenas alterações, mantém o estado atual) e R para hot restart (o código é compilado novamente e retorna ao estado inicial).

4.1.2.1 Integração com o Instagram

Temos o objetivo de conectar uma conta do Instagram ao nosso aplicativo através de Login e posterior consentimento. O Instagram é uma rede social que pertence ao Facebook então precisamos criar uma conta no Facebook for Developers e criar um novo aplicativo para consumo das APIs da empresa como é mostrado na Figura 23.

Figura 23 – Criação do aplicativo na plataforma Facebook for Developers



Fonte: O Autor

Na plataforma Facebook for Developers podemos configurar o aplicativo adicionando as funcionalidades de Login do Facebook, opção de utilização da Graph API do Instagram e as plataformas que o aplicativo irá funcionar como Web, Android e iOS e associá-lo ao nosso projeto Flutter.

Para a implementação da classe que possibilitará o Login com o Facebook utilizaremos a seguinte biblioteca:

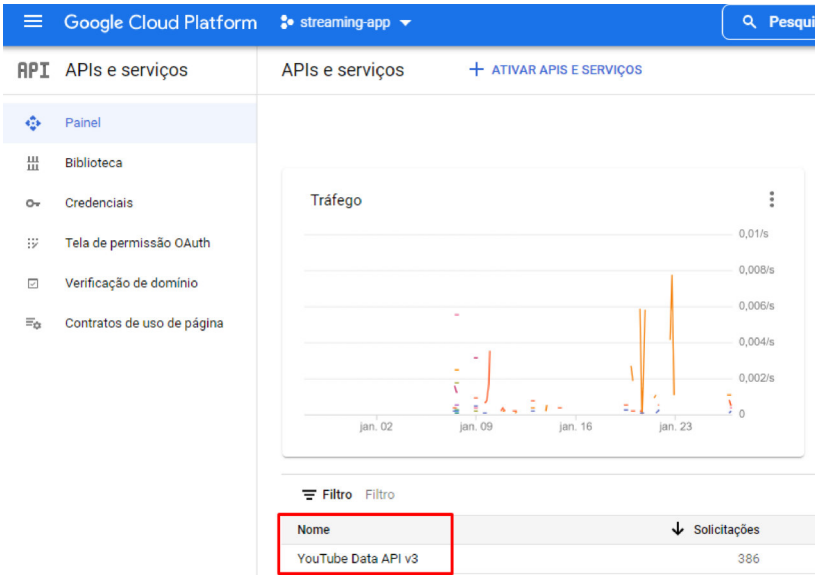
```
$ flutter pub add flutter_facebook_auth
```

Ao importar a biblioteca em um arquivo .dart, teremos à nossa disposição os métodos de realizar login, logout e obtenção de tokens de acesso a conta de um usuário do Facebook. Desse modo, conseguiremos gerir suas páginas e a sua conta de Instagram associada.

4.1.2.2 Integração com o Youtube

Temos o objetivo de conectar uma conta do Youtube ao nosso aplicativo através de Login e posterior consentimento. O Youtube é uma rede social que pertence ao Google então precisamos criar uma conta no Google Cloud Platform e criar um novo aplicativo para consumo das APIs da empresa como é mostrado na Figura 24.

Figura 24 – Criação do aplicativo na Google Cloud Platform



Fonte: O Autor

Na plataforma Google Cloud Platform podemos configurar o aplicativo adicionando as funcionalidades de Login do Google, opção de utilização da Youtube API e as plataformas as quais o aplicativo irá funcionar, como Web, Android, iOS, e associá-lo ao nosso projeto Flutter.

Para a implementação da classe que possibilitará o Login com o Google utilizaremos a seguinte biblioteca:

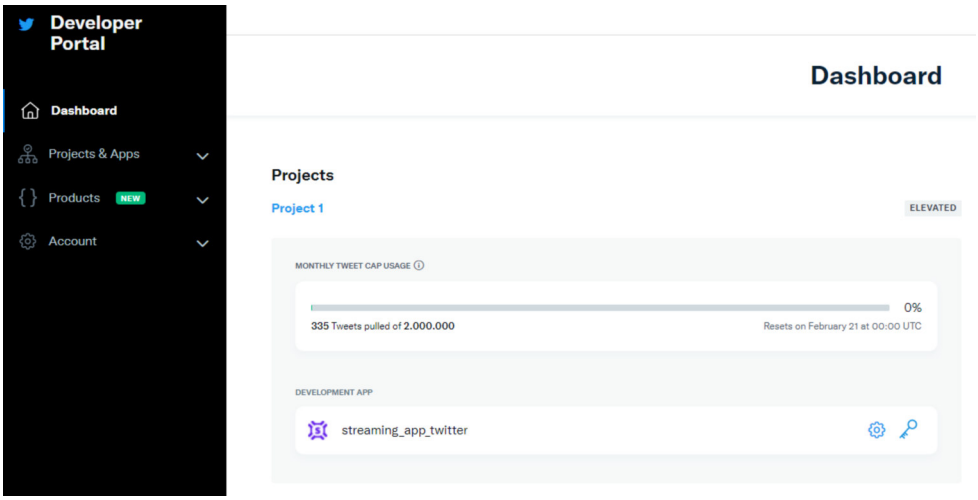
```
$ flutter pub add google_sign_in
```

Agora, ao importar a biblioteca em um arquivo .dart, teremos à nossa disposição os métodos de realizar login, logout e obtenção de tokens de acesso a conta de um canal do Youtube.

4.1.2.3 Integração com o Twitter

Temos o objetivo de conectar uma conta do Twitter ao nosso aplicativo através de Login e posterior consentimento. Senso assim precisamos criar uma conta no Twitter

Figura 25 – Criação do aplicativo no Twitter Developer Portal



Fonte: O Autor

Developer Portal e criar um novo aplicativo para consumo das APIs da empresa e associá-lo ao nosso projeto Flutter como é mostrado na Figura 25.

Para a implementação da classe que possibilitará o Login com o Twitter utilizaremos a seguinte biblioteca:

```
$ flutter pub add twitter_login
```

Agora, ao importar a biblioteca em um arquivo .dart, teremos à nossa disposição os métodos de realizar login, logout e obtenção de tokens de acesso a conta de um usuário do Twitter.

4.1.2.4 Consumo à API do Strapi

Para consumir o nosso back-end que são as operações desenvolvidas na API do Strapi devemos instalar uma biblioteca que possibilite realizar requisições HTTP. A biblioteca escolhida foi a seguinte:

```
$ flutter pub add http
```

Utilizando a biblioteca HTTP podemos implementar métodos para consumir o back-end Strapi que foi construído anteriormente. Devemos lembrar que o back-end serve como fachada para as demais APIs das redes sociais e suas chamadas foram implementadas diretamente no back-end.

O aplicativo Flutter foi encarregado de capturar os tokens de acesso de cada rede social conectada e manter esse token em memória durante a sessão do usuário. Então

podemos passar parâmetros via interface do usuário para o nosso método se desejarmos. Optou-se em enviar os tokens de acesso das redes sociais via header e possíveis parâmetros de consulta via queryParam conforme podemos visualizar na Figura 26. É importante destacar que o Heroku disponibiliza o protocolo HTTPS ao nosso back-end possibilitando que a comunicação seja criptografada.

Figura 26 – Exemplo de consumo de publicações do back-end Strapi

```
class PostRepository implements IPostRepository {
  final String _baseUrl = BASE_URL;
  final String authorization = 'Bearer ' + STRAPI_AUTHORIZATION;

  @override
  Future<List<Post>> findAllPosts() async {
    final accessToken = await FacebookAuth.instance.accessToken;
    final String accessTokenFacebook = accessToken!.token;
    final String accessTokenYoutube = await GoogleLogin.getAcessTokenYoutube();

    final queryParameters = {
      'instagram': 'true',
      'youtube': 'true',
      'twitter': 'true'
    };

    Uri uri = Uri.http(_baseUrl, '/posts', queryParameters);

    Map<String, String> headers = {
      HttpHeaders.contentTypeHeader: 'application/json',
      "access_token": accessTokenFacebook,
      "access_token_youtube": accessTokenYoutube,
      "Authorization": authorization,
      "access_token_twitter": TWITTER_AUTHORIZATION,
    };

    final response = await http.get(uri, headers: headers);
    final List<dynamic> responseMap = jsonDecode(response.body);
    return responseMap.map<Post>((resp) => Post.fromMap(resp)).toList();
  }
}
```

Fonte: O Autor

4.1.2.5 Trabalhando com Imagens e Videos

A biblioteca Cached Network Image possibilita a apresentação de imagens na tela que estão disponíveis na Internet através de uma URL. Além disso a biblioteca realiza o cache da imagem para melhorar a performance. A biblioteca Image Picker possibilita o acesso à câmera do dispositivo para captura de imagens e vídeos e a também realiza a busca dos mesmos itens na Galeria do dispositivo. A biblioteca Video Player possibilita reproduzir videos em segundo plano que estão disponíveis na Internet através de uma URL. Por fim, a biblioteca Youtube Player Flutter possibilita reproduzir vídeos que estão hospedados no Youtube.

```
$ flutter pub add cached_network_image
$ flutter pub add image_picker
$ flutter pub add video_player
$ flutter pub add youtube_player_flutter
```

4.1.3 Landing Page Next.js

Nessa subseção iremos tratar do desenvolvimento da Landing Page utilizando o framework Next.js. Para criar o projeto podemos utilizar o gerenciador de pacotes "yarn"e rodar o seguinte comando:

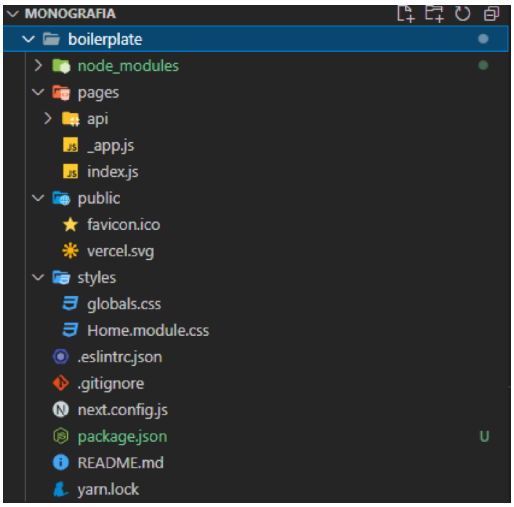
```
$ yarn create next-app
```

Após essa etapa será necessário adicionar o nome do projeto. Nesse caso podemos criar um projeto que sirva como base para projetos futuros, visto que as configurações que serão efetuadas posteriormente serão as mesmas ao criar um projeto Next.js e poderemos reaproveitar tais configurações.

```
$ What is your project named? ... nome_do_projeto_base
```

Na Figura 27 podemos observar a estrutura de diretórios do Next.js que foi gerada através dos comandos anteriores.

Figura 27 – Estrutura de Diretórios de um Projeto Next.js



Fonte: O Autor

Tabela 6 – Descrição dos Diretórios do Projeto Next.js

Diretório	**Descrição**
pages	Diretório de páginas.
public	Arquivos estáticos: imagens, favicons.
styles	Arquivos de estilo das páginas.
index.js	Página principal Home.
package.json	Arquivo de configurações e lista de dependências

Agora devemos acessar o diretório principal do projeto.

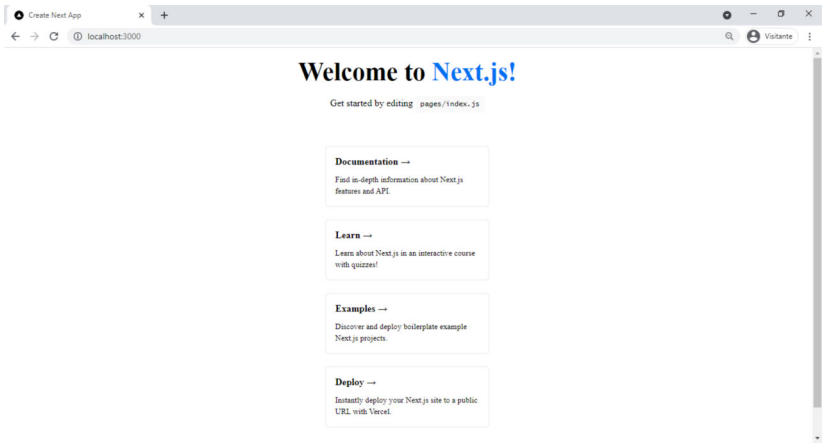
```
$ cd nome_do_projeto_base
```

Para executar a aplicação gerada podemos rodar o comando abaixo.

```
$ yarn dev
```

Agora podemos acessar o endereço local disponível em http://localhost:3000 e uma página de boas-vindas do Framework será exibida, como podemos visualizar na Figura 28.

Figura 28 – Página de boas-vindas do Next.js



Fonte: O Autor

4.1.3.1 Integração com TypeScript

Para realizar a integração com o TypeScript devemos criar arquivo tsconfig.json e adicionar a sua dependência para react executando o comando abaixo:

```
$ yarn add --dev typescript @types/react
```

Após essa etapa ao executarmos o comando yarn dev o arquivo tsconfig.json será populado automaticamente com os valores padrões. Além disso, é criado o arquivo next-env.d.ts que é um arquivo que declara os tipos do next.js, e por fim, são adicionadas três dependências no arquivo package.json: node, react e typescript.

4.1.3.2 Adicionando ESLint

A ferramenta de análise de código ESLint foi adicionada ao projeto para ajudar a identificar possíveis padrões problemáticos no código executando o comando abaixo. (ESLINT, 2021)

```
$ npx eslint --init
```

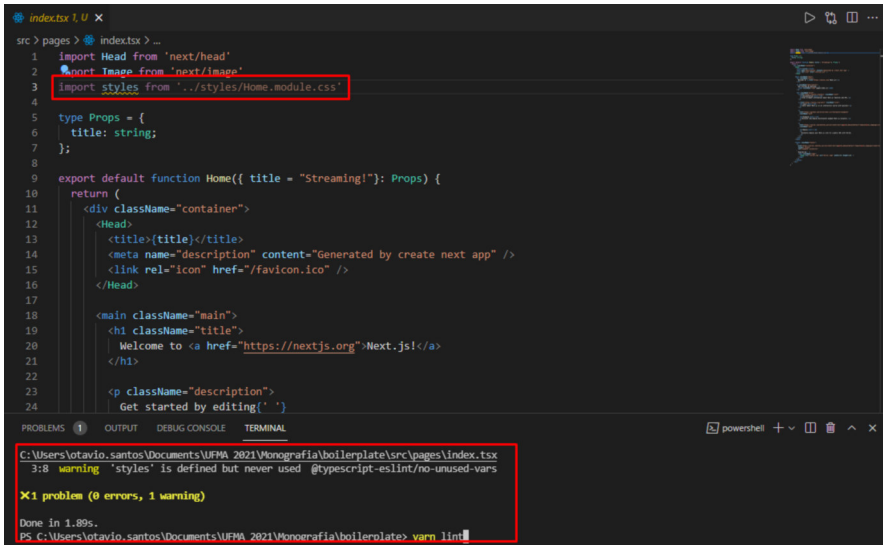
Após a execução do comando acima o arquivo .eslintrc.json foi gerado. Agora o próximo passo é instalar os plugins do eslint com o seguinte comando:

```
$ yarn add --dev eslint-plugin-react@latest
@typescript-eslint/eslint-plugin@latest @typescript-eslint/parser@latest:
```

No arquivo package.json podemos acessar a estrutura scripts e alterar o parâmetro lint para "eslint src". Dessa forma, quando rodarmos o comando abaixo, todo o código fonte será analisado conforme a Figura 29.

```
$ yarn lint
```

Figura 29 – Resultado da análise do código fonte com Eslint



Fonte: O Autor

4.1.3.3 Adicionando o Prettier

A ferramenta Prettier foi utiliza para ajudar na formatação automática do código fonte no Visual Studio Code. Para adicionar ao nosso projeto executamos o comando abaixo. (ESLINT, 2021)

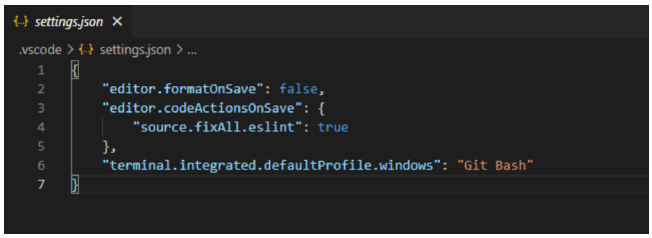
```
$ yarn add --dev --exact prettier
```

Após a execução do comando acima o arquivo .prettierrc.json foi gerado. Agora o próximo passo é instalar os plugins de integração do eslint com o prettier executando os seguintes comandos:

```
$ yarn add --dev eslint-plugin-prettier
$ yarn add --dev eslint-config-prettier
```

E para finalizar a integração abrimos o arquivo .eslintrc.json e adicionamos na estrutura extends o parâmetro plugin:prettier/recommended. Como passo adicional podemos incluir uma configuração no arquivo /settings.json para formatação automática do código ao salvar. Aproveitando a alteração foi adicionado uma configuração para utilizar o Git Bash integrado ao terminal do VSCode como mostrado na Figura 30.

Figura 30 – Configurando o arquivo setting.json no VScode



Fonte: O Autor

4.1.3.4 Adicionando o Husky

A ferramenta Husky foi utilizada para adicionar a funcionalidade de git hook que tem o intuito impedir que erros de lint sejam incluídos no repositório do GitHub. Essa prática é muito útil para manter a qualidade do código (GITHOOKS. . . , 2021). Executando o comando abaixo instalamos o Husky e a pasta .husky é criada.

```
$ yarn add husky --dev
```

Executamos o comando abaixo para gerar o arquivo pre-commit:

```
$ npx husky-init && yarn
```

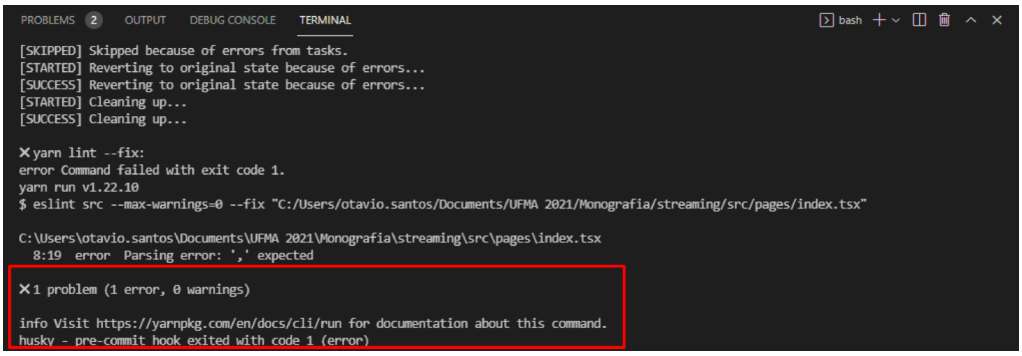
Agora podemos instalar o lint-staged para possibilitar a validação de cada commit conforme a Figura 31.

```
$ yarn add lint-staged --dev
```

4.1.3.5 Testes com Jest

A framework Jest foi utilizado como a nossa ferramenta de testes. Para isso executamos o comando abaixo e criamos arquivo de configuração jest.config.js conforme a Figura 32.

Figura 31 – Husky e lint-staged realizando validação do commit.



Fonte: O Autor

```
$ yarn add --dev jest @babel/preset-typescript @types/jest
```

Figura 32 – Configuração do arquivo jest.config.js

Fonte: O Autor

Em seguida criamos na raiz do projeto o diretório .jest e dentro da pasta criamos o arquivo setup.js com as informações do Jest. Por fim no package.json adicionamos em scripts a configuração "test": "jest".

Agora adicionamos o React Testing Library com o seguinte comando:

```
$ yarn add --dev @testing-library/react @testing-library/jest-dom
```

Importamos então o "React Test Library"no arquivo setup.js do jest. No diretório src criamos um novo diretório components e dentro dele criamos um diretório chamado "Main". Dentro da diretório Main criamos um arquivo chamado index.tsx e outro chamado test.tsx.

Agora estamos aptos a escrever o nosso primeiro teste de componente e em seguida criar o componente em si como podemos ver nas Figuras 33 e 34.

Agora com essas configurações antes de efetuar um novo commit no Git serão executados os testes e validações de código e só será possível concretizar o commit se todos os testes estiverem passando e não existir nenhum erro no código.

Figura 33 – Criando o primeiro teste de componente.

Fonte: O Autor

Figura 34 – Implementando o componente.

Fonte: O Autor

4.1.3.6 Styled Components

O Styled Components permite que sejam adicionadas funcionalidades de CSS ao código TypeScript. Para instalar executamos o comando abaixo. (STYLED. . . , 2021)

```
$ yarn add styled-components
```

Agora precisamos criar o arquivo documents.tsx dentro do diretório /pages para integrar com o Next.js. Em seguida precisamos realizar uma modificação para integrar os nossos testes com o "styled-components"executando o seguinte comando:

```
$ yarn add --dev jest-styled-components
```

No arquivo de setup.js importamos o jest-styled-components e depois podemos executar o seguinte comando de execução dos testes e o resultado pode ser visto na Figura 35.

```
$ yarn test
```

Figura 35 – Executando os testes com yarn test.

```
$ yarn test
yarn run v1.22.10
$ jest
 PASS  src/components/Main/test.tsx
  <Main />
    √ should render the heading (65 ms)

-----------------|---------|----------|---------|---------|-------------------
File             | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------|---------|----------|---------|---------|-------------------
All files        |   66.67 |        0 |      50 |   66.67 |
 components/Main |     100 |      100 |     100 |     100 |
  index.tsx      |     100 |      100 |     100 |     100 |
 pages           |       0 |        0 |       0 |       0 |
  index.tsx      |       0 |        0 |       0 |       0 | 9
-----------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        5.692 s
Ran all test suites.
Done in 7.49s.
```

Fonte: O Autor

Na Figura 36 é apresentado como foi configurado o arquivo styles.ts e na Figura 37 é apresentado como podemos utilizá-lo na nossa página principal.

Figura 36 – Configurando styles.ts

```
styles.ts U
src > components > Main > styles.ts > LogoConnect
1   import styled from 'styled-components'
2
3   export const Wrapper = styled.main`
4     background-color: #1b1b1b;
5     color: #fff;
6     width: 100%;
7     height: 100%;
8     padding: 3rem;
9     text-align: center;
10    display: flex;
11    flex-direction: column;
12    align-items: center;
13    justify-content: center;
14  `
15
16  export const Logo = styled.img`
17    width: 10rem;
18    margin-bottom: 2rem;
19  `
20  export const LogoConnect = styled.img`
21    width: 70rem;
22    margin-bottom: 2rem;
23  `
```

Fonte: O Autor

Figura 37 – Utilizando Styled Components na página inicial

```
index.tsx M
src > components > Main > index.tsx > ...
1   import * as S from './styles'
2
3   const Main = () => (
4     <S.Wrapper>
5       <S.Logo src="/img/logo.svg" alt="Um logotipo com a letra S" />
6       <S.Title>Streaming</S.Title>
7       <S.Description>Integração de Mídias Sociais</S.Description>
8       <S.LogoConnect
9         src="/img/connect-midia.svg"
10        alt="Arte que mostra a conexão com várias mídias sociais"
11      />
12    </S.Wrapper>
13  )
14
15  export default Main
```

Fonte: O Autor

4.1.3.7 Storybook Para React

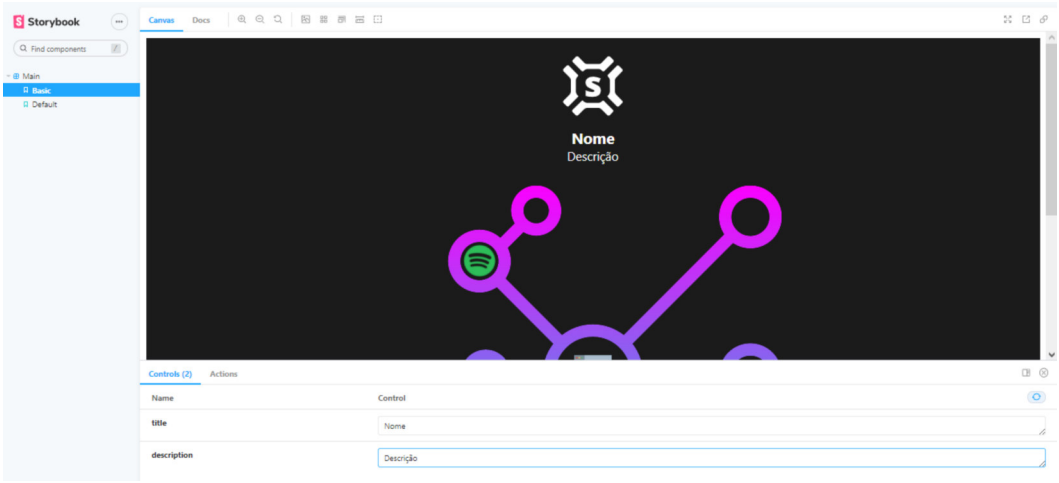
Segundo o site official (STORYBOOK, 2021), o Storybook é uma ferramenta de código aberto para criar componentes de interface do usuário e páginas. Ele simplifica o desenvolvimento, o teste e a documentação da interfaces. Para adicionarmos o Storybook ao nosso projeto executamos o comando abaixo:

```
$ npx -p @storybook/cli sb init --type react
```

O diretório .storybook será então criado e podemos inicilizar o storybook com o comando abaixo e podemos acessar a página da Figura 38 e verificar a integração com os componentes criados.

```
$ yarn storybook
```

Figura 38 – Utilizando o Storybook.



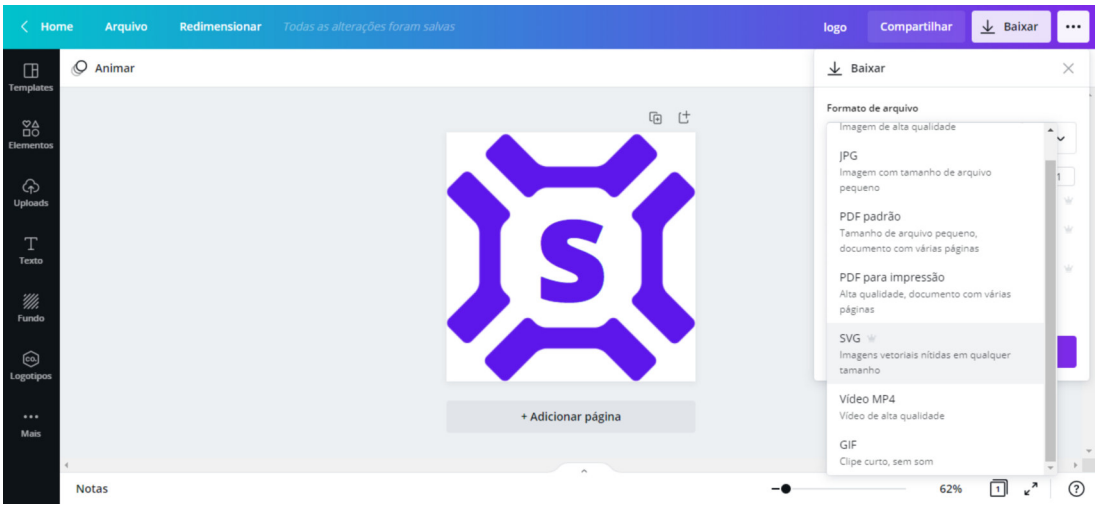
Fonte: O Autor

O software Canva foi utilizado para construção de imagens que podem ser exportadas no formato .svg para utilizarmos na construção dos nossos componentes como mostrado na Figura 39. (CANVA, 2021)

4.1.3.8 Progressive Web App (PWA)

Um PWA é uma aplicação híbrida entre web e mobile (PWA..., 2021). É possível adicionar funcionalidades de PWA ao nosso projeto next.js melhorando o desempenho do armazenando dados em cache e adicionando a estratégia de "offline fisrt"por exemplo que tira a responsabilidade da conexão com a internet no desempenho da aplicação. Para adicionar esse recurso executamos o seguinte comando:

Figura 39 – Canva.



Fonte: O Autor

```
$ yarn add next-pwa
```

O próximo passo é adicionar o arquivo next.config.js com seguinte configuração.

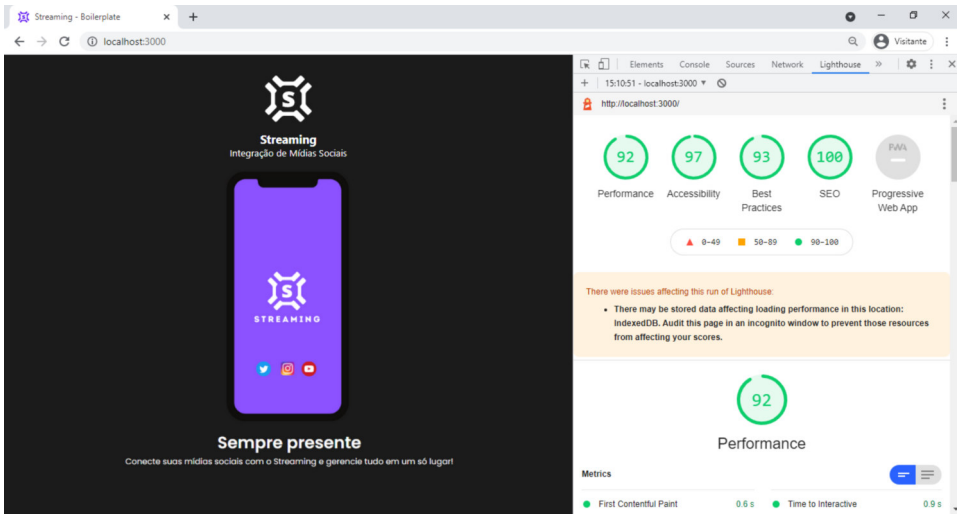
```
const withPWA = require('next-pwa')
const isProd = process.env.NODE_ENV === 'production'

module.exports = withPWA({
  pwa: {
    dest: 'public',
    disable: !isProd
  }
})
```

Para testar o funcionamento do PWA podemos rodar o Lighthouse do navegar que é uma ferramenta que avalia o desempenho e a qualidade de aplicativos da web conforme podemos visualizar na Figura 40.

4.1.3.9 Integração contínua

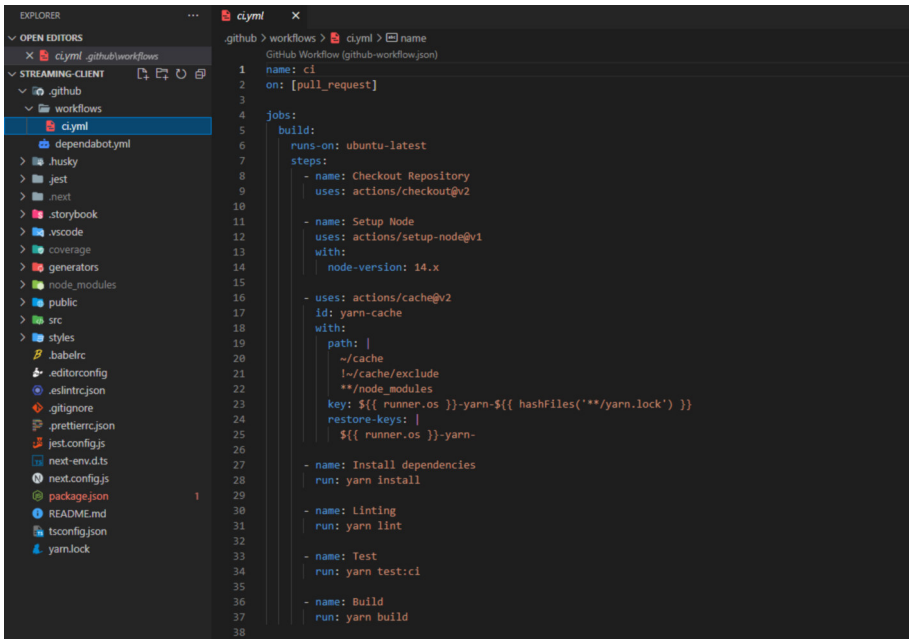
A integração contínua é uma união de esforços para garantir que seja possível realizar atualizações constantes da ramificação principal do projeto, mantendo o código sempre atualizado e livre de bugs. Para isso devemos criar o diretório .github, adicionar a ele o diretório "workflows"e dentro dele o arquivo ci.yml. E dentro dele adicionar o nosso fluxo de integração contínua conforme a Figura 41.

Figura 40 – PWA.



Fonte: O Autor

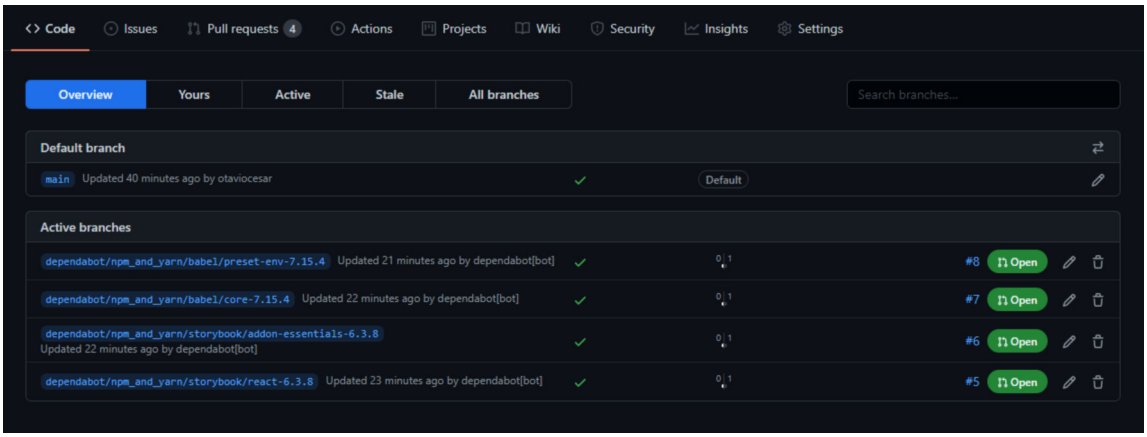
Figura 41 – Configurando a Integração Contínua



Fonte: O Autor

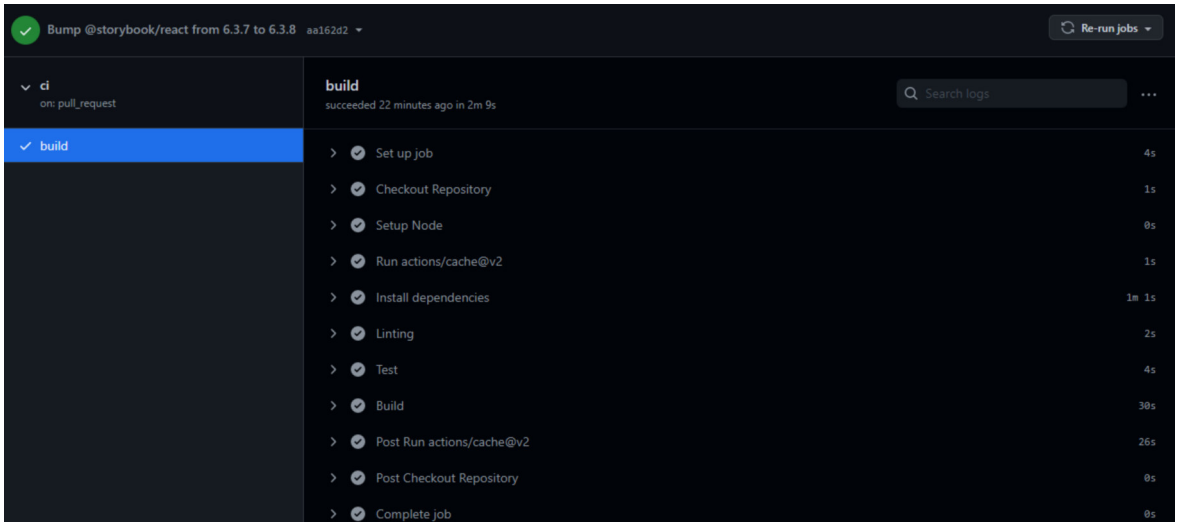
Ainda conforme a Figura 41, adicionamos também o arquivo dependabot.yml do github para verificação automática das atualizações de dependências. Essa funcionalidade realiza verificações diárias e caso exista uma atualização de biblioteca é criada automaticamente uma branch contendo a atualização, e em seguida, um pull request é gerado conforme podemos visualizar na Figura 42. Durante o pull request todos os testes e builds rodam para certificar que nada quebre. Na Figura 43 podemos visualizar as branches criadas automaticamente pelo dependabot.

Figura 42 – Branches criadas pelo dependabot.



Fonte: O Autor

Figura 43 – Fluxo de Integração Contínua.



Fonte: O Autor

4.1.3.10 Criando novos projetos através do Boilerplate

O Boilerplate é um projeto base, ou seja, a partir dessas configurações iniciais podemos gerar os futuros projetos sem que haja retrabalho. Para criar uma nova aplicação a partir do projeto boilerplate disponível no Github basta executarmos o seguinte comando:

```
$ yarn create next-app -e https://github.com/sua-conta/boilerplate-nextjs
```

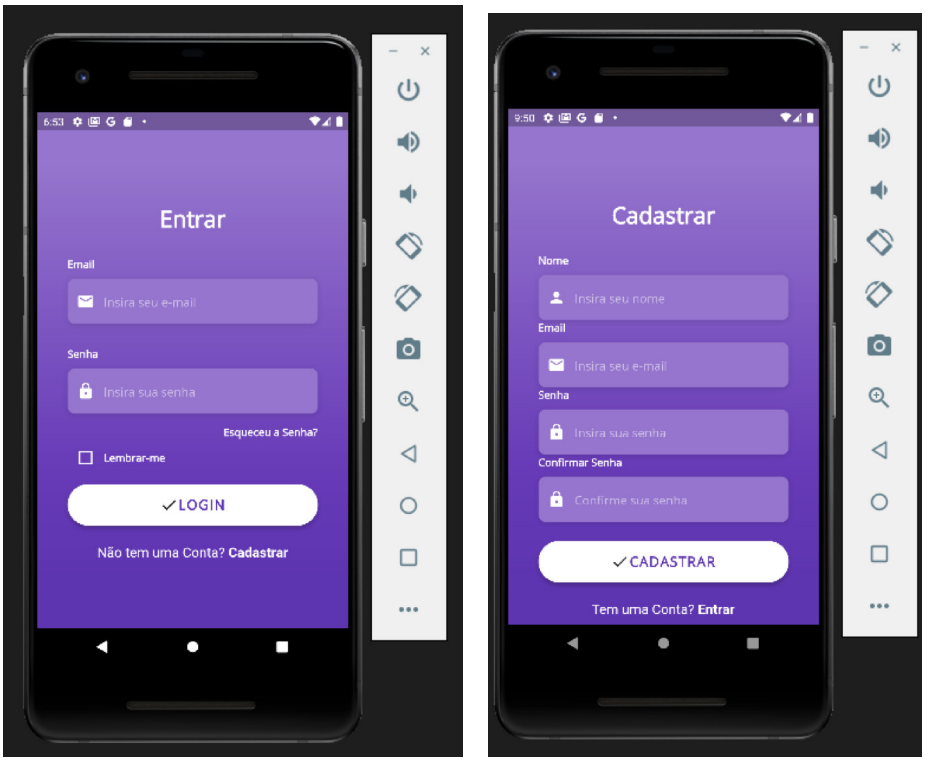
4.2 Resultados Obtidos

4.2.1 Aplicação Flutter

4.2.1.1 Autenticação de Usuário

Ao abrir o aplicativo somos direcionados para a tela de Login. Caso o usuário não possua uma conta ele poderá navegar para tela de cadastro.

Figura 44 – a) Tela de Login e b) Tela de Cadastro de Usuário



(a) (b)

Fonte: O autor

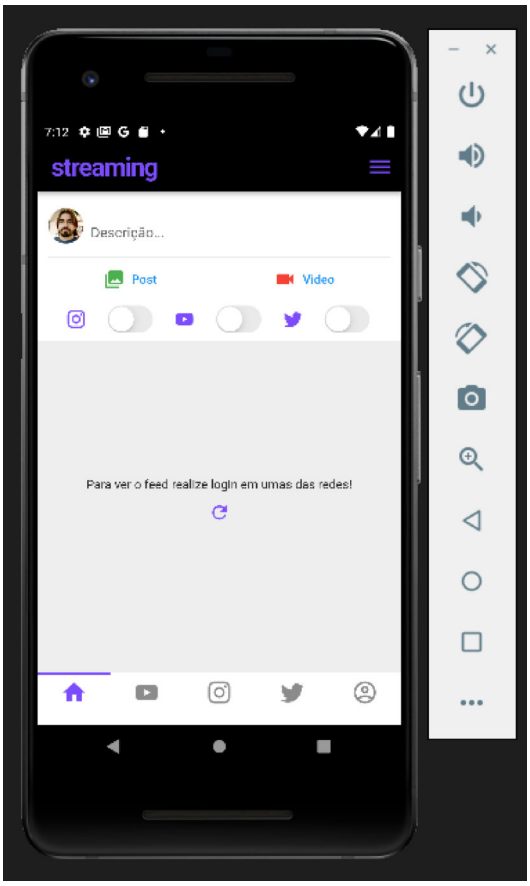
4.2.2 Tela de Feed das Redes Sociais

A nossa Home é a tela do feed unificado das redes sociais. Quando não temos nenhuma rede conectada uma mensagem é informada no meio da Tela.

4.2.3 Telas de Login/Consentimento das Redes Sociais e Perfis

Podemos navegar para as telas adjacentes com os ícones das redes sociais compatíveis com o aplicativo. Em cada aba temos uma tela que permite o login na rede social e a posterior concessão do consentimento de acesso. Após o consentimento o estado de cada tela muda e agora podemos visualizar os dados principais de cada conta conectada ao aplicativo.

Figura 45 – Tela de Feed das Redes Sociais.



Fonte: O Autor

Na última aba temos a tela de perfil do usuário logado. A imagem de perfil é carregada automaticamente caso exista no provedor de e-mail de origem. Ainda na mesma tela de perfil é possível Sair do aplicativo. Após a ação de sair, em um próximo acesso, será solicitado as credenciais do usuário novamente.

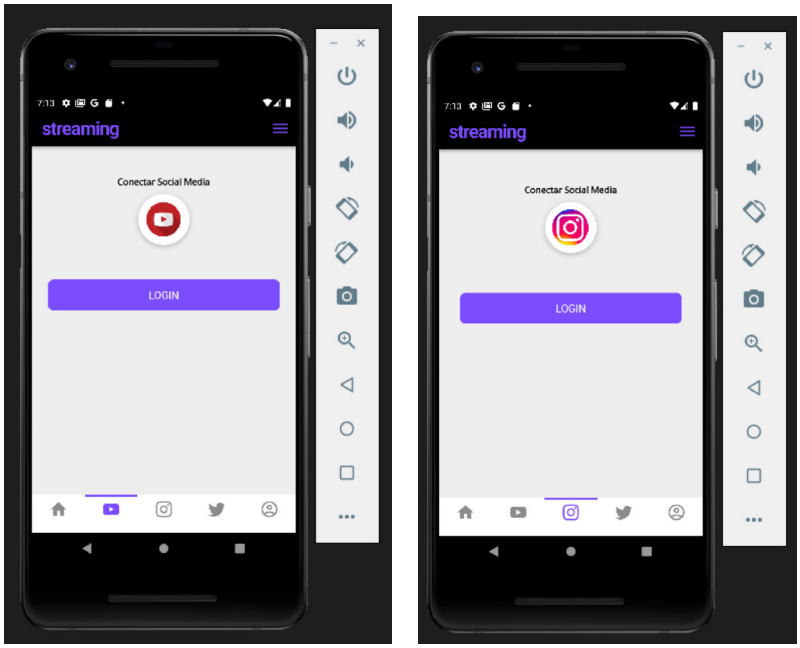
4.2.4 Tela de Feed após Consentimentos

Após a integração das contas de pelo menos uma das redes sociais disponíveis, a tela de feed é carregada com as publicações de cada rede conectada.

4.2.5 Criando publicações simultâneas nas Redes Sociais

É possível acionar através dos botões em quais redes sociais será realizada a publicação da imagem ou vídeo.

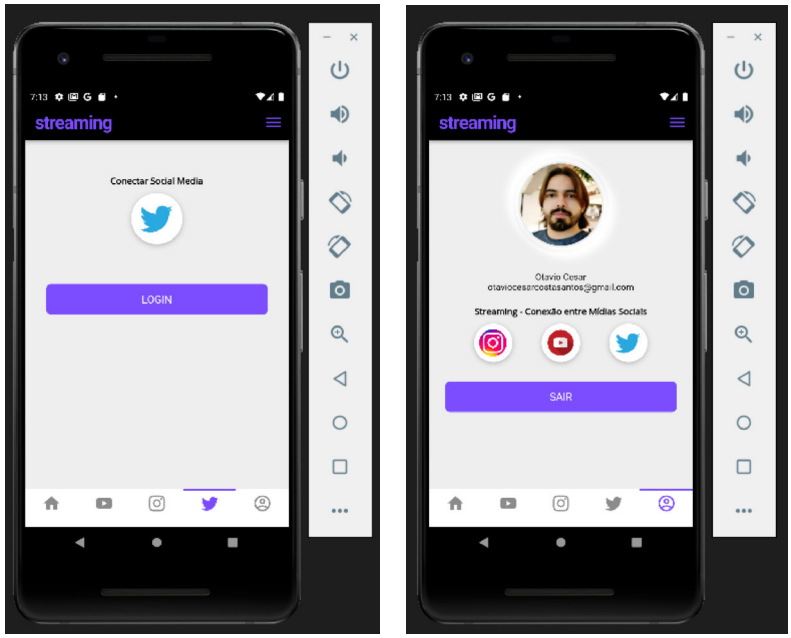
Figura 46 – a) Tela de Login/Consentimento do Youtube e b) Tela de Login/Consentimento do Instagram



(*a*) (*b*)

Fonte: O autor

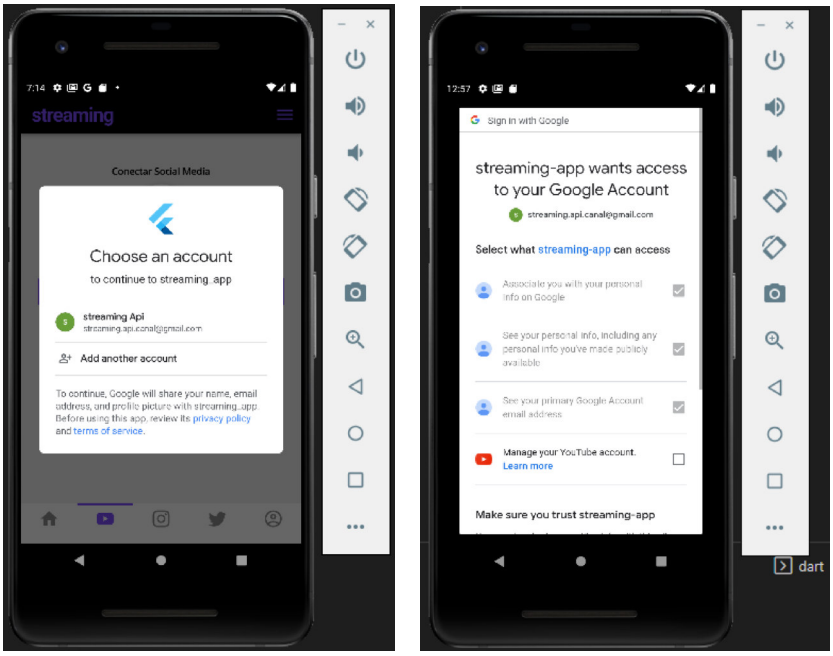
Figura 47 – a) Tela de Login/Consentimento do Twitter e b) Tela de Perfil/Logout



(*a*) (*b*)

Fonte: O autor

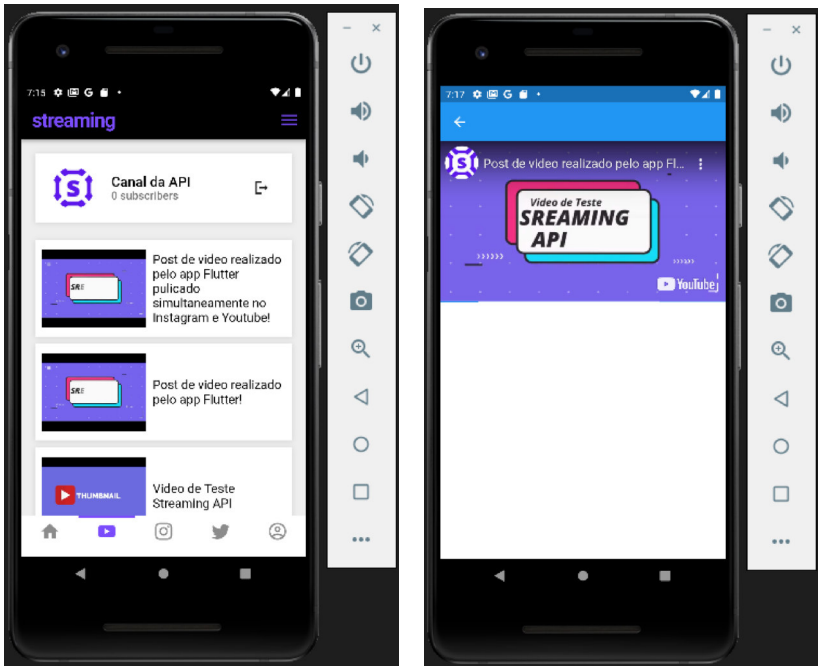
Figura 48 – a) Realizando Login com a Conta do Google e b) Concedendo Consentimentos para Acesso a Conta do Youtube



(*a*) (*b*)

Fonte: O autor

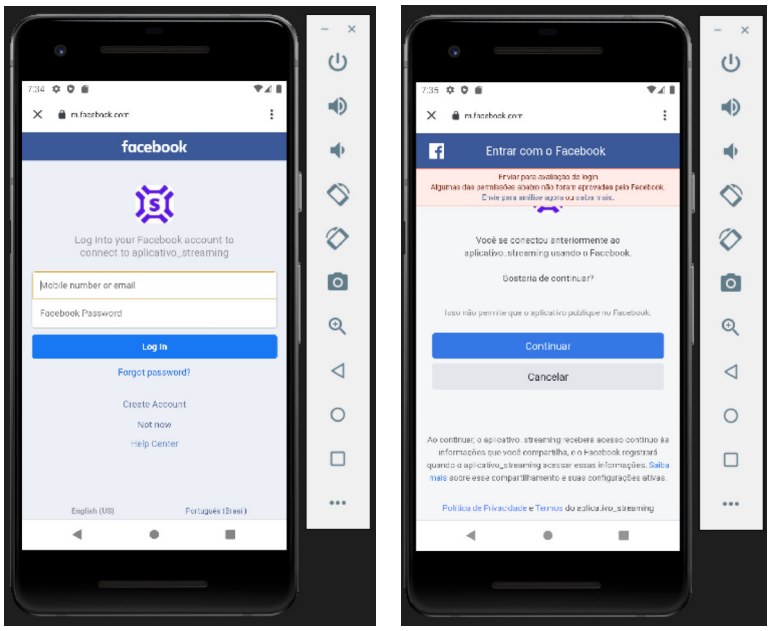
Figura 49 – a) Dados do Canal do Youtube carregados após o consentimento e b) Carregando um Video da Lista



(*a*) (*b*)

Fonte: O autor

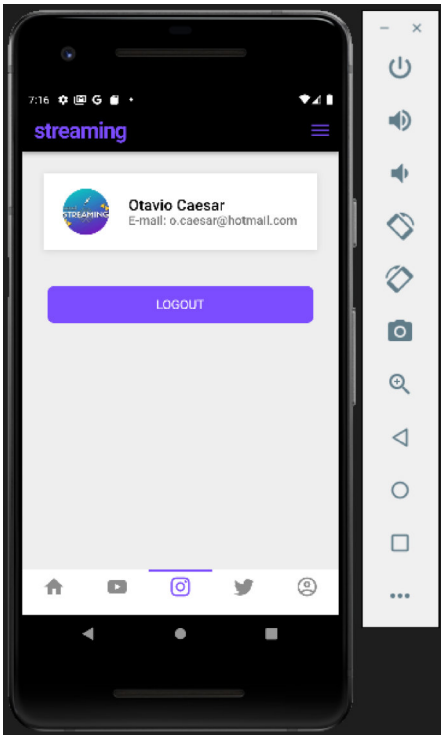
Figura 50 – a) Realizando Login com a Conta do Facebook e b) Concedendo Consentimentos para Acesso a Página do Facebook vinculada a conta do Instagram



(*a*) (*b*)

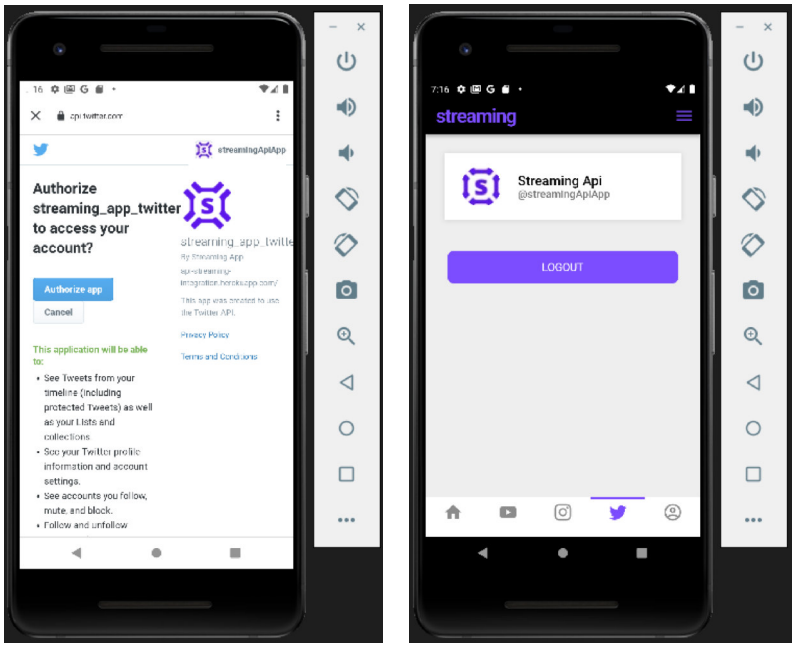
Fonte: O autor

Figura 51 – Dados da conta do Facebook carregados após o consentimento



Fonte: O Autor

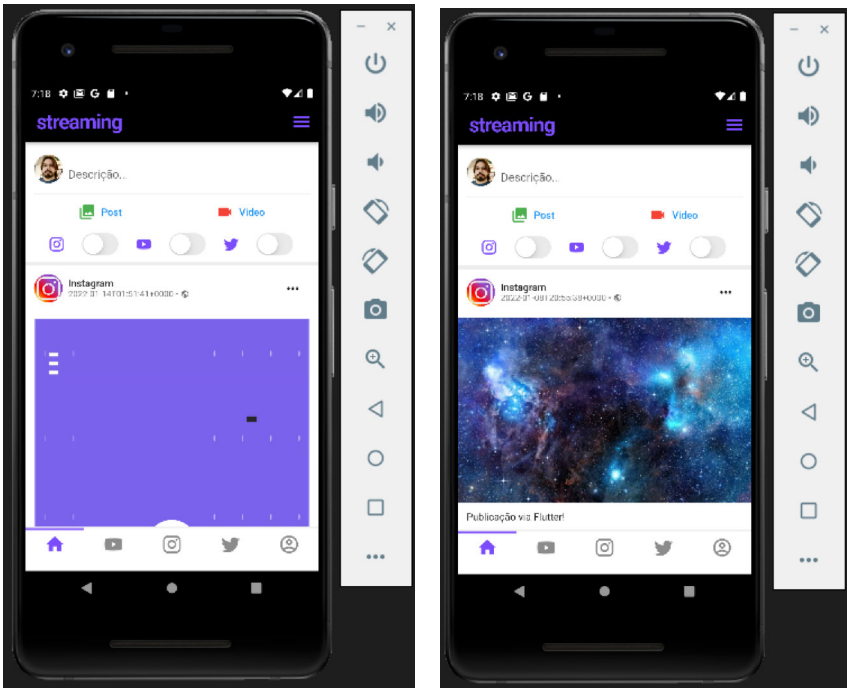
Figura 52 – a) Concedendo Consentimentos para Acesso a Conta do Twitter e b) Dados da conta do Twitter carregados após o consentimento



(*a*) (*b*)

Fonte: O autor

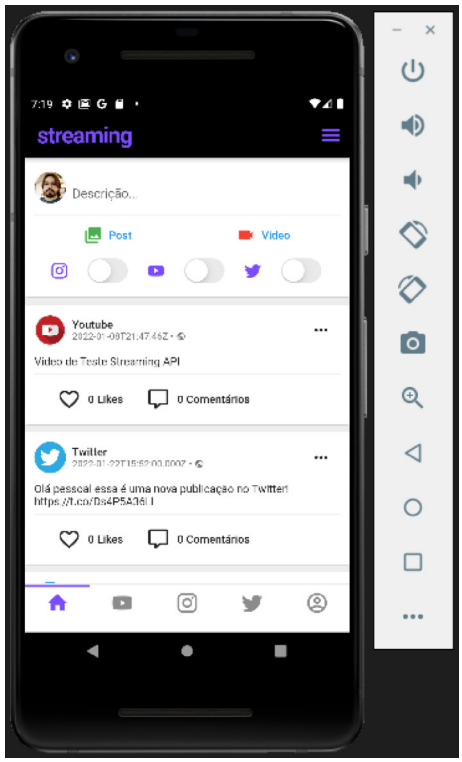
Figura 53 – a) Video do Instagram carregado no Feed e b) Imagem do Instagram carregada no Feed



(*a*) (*b*)

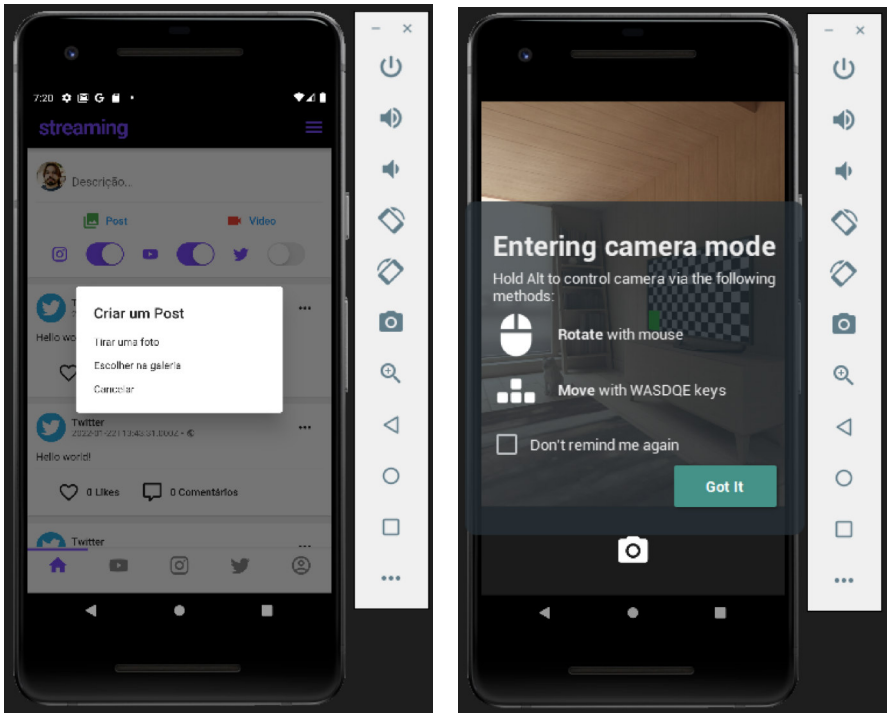
Fonte: O autor

Figura 54 – Navegando pelas publicações das outras contas.



Fonte: O Autor

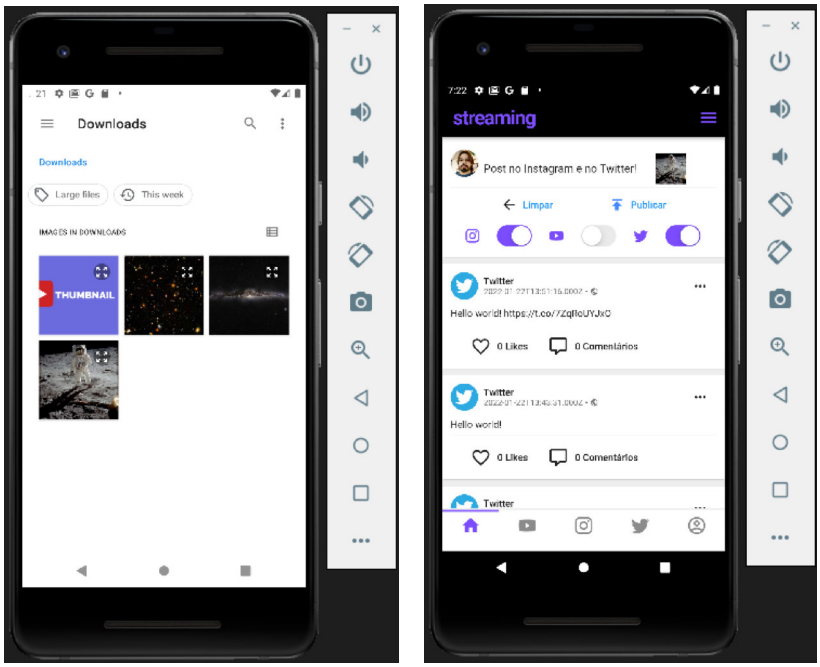
Figura 55 – a) Criando uma publicação com Imagem e b) Acesso a câmera do dispositivo



(*a*) (*b*)

Fonte: O autor

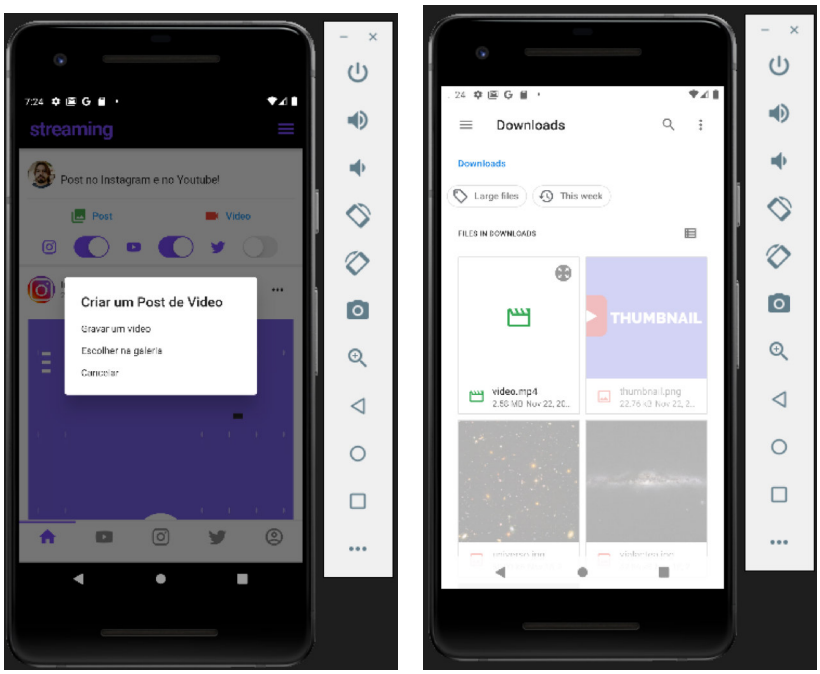
Figura 56 – a) Acesso aos arquivos de imagens salvas na galeria do dispositivo e b) Formulário de publicação preenchido



(*a*) (*b*)

Fonte: O autor

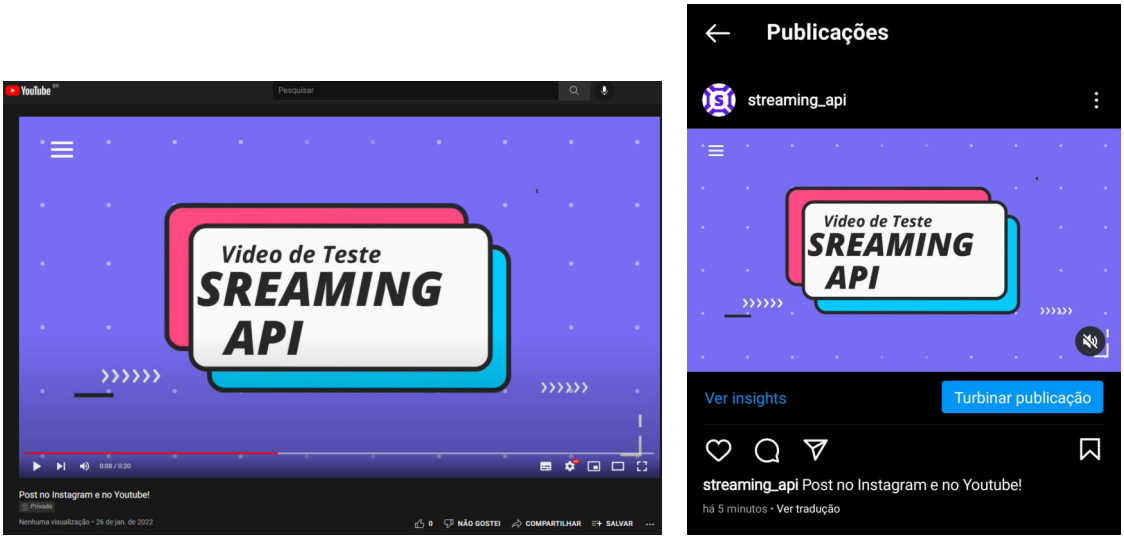
Figura 57 – a) Criando uma publicação com Video e b) Acesso aos arquivos de video salvos na galeria do dispositivo



(*a*) (*b*)

Fonte: O autor

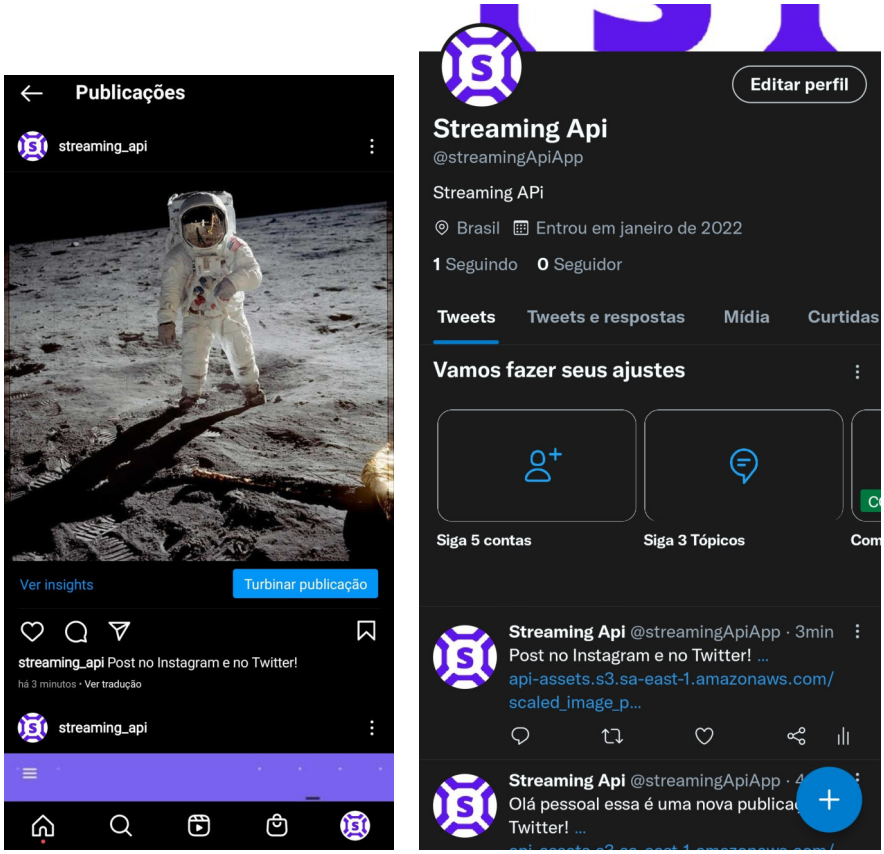
Figura 58 – a) Video publicado no Youtube e b) Video publicado no Instagram



(a) (b)

Fonte: O autor

Figura 59 – a) Imagem publicada no Instagram e b) Link da Imagem Publicada no Twitter



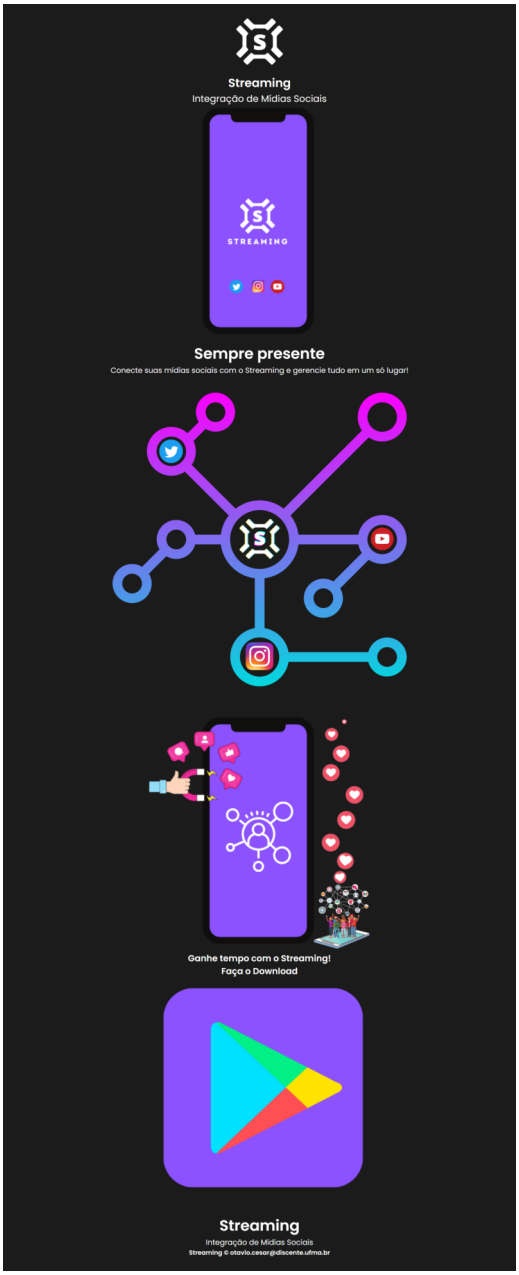
(a) (b)

Fonte: O autor

4.2.6 Landing Page

Na Figura 60 temos a aplicação Landing Page finalizada. O Objetivo desse tipo de site é ser uma porta de entrada para um produto ou serviço. No nosso caso ele tem o objetivo de convencer o cliente a realizar o download do aplicativo. O intuito é que ele seja rápido, simples e que tenha uma boa indexação nos mecanismos de busca.

Figura 60 – Landing Page Completa



Fonte: O Autor

5 Conclusão e Trabalhos Futuros

O presente trabalho apresenta um conjunto de ferramentas capazes de realizar a gestão de conteúdo em três ferramentas de mídia social consolidadas no mercado. Ao todo foram disponibilizados três artefatos construídos com tecnologias modernas e distintas, incluindo uma API, um aplicativo mobile e uma página Web.

O Strapi demonstrou ser uma ferramenta que possibilita a construção de APIs com extrema velocidade e qualidade, além de permitir uma fácil manutenibilidade. Sendo assim, torna-se trivial expandir as suas funcionalidades, regras de negócio e adicionar ou substituir redes sociais de acordo com as necessidades dos usuários.

Foi possível perceber ainda, o poder que as APIs proporcionam ao permitir a comunicação entre sistemas que foram implementados em diferentes linguagens de programação e com tecnologias distintas. Além de proporcionar o fluxo leve de dados através da notação JSON (JavaScript Object Notation). O desenvolvimento do backend da aplicação como uma API possibilitou também a liberdade de consumo do modelo de negócio por diferentes clientes e aplicativos.

Conseguinte, a aplicação Flutter foi pensada de forma a contribuir como uma ferramenta que auxilie empreendedores na gestão de seus canais de comunicação com potenciais clientes. Desta forma, o principal objetivo é direcionar o conteúdo para todos os canais onde o público esteja presente. Assim, o sistema prevê o acompanhamento das publicações em uma tela única dando ao usuário a visão geral de onde tais conteúdos geram mais engajamento.

Além disso, na construção da Landing Page, por ser uma aplicação mais simples, a proposta foi utilizar ao máximo as melhores práticas de desenvolvimento de software e ferramentas. Houve tempo hábil para realizar os testes dos componentes e configurar um fluxo de entrega contínua.

Por fim, para os trabalhos futuros, a intenção é adicionar novas funcionalidades e melhorarias de usabilidade de acordo com o feedback recebido dos usuários, migrar o sistema de login para o Strapi, ajustar o fluxo de autenticação das redes sociais nas plataformas Web e iOS, possibilitar a interação com os comentários das publicações e disponibilizar o aplicativo Flutter na Play Store e App Store.

Referências

ALGAWORKS. 2021. Disponível em: <https://blog.algaworks.com/4-conceitos-sobre-rest-que-qualquer-desenvolvedor-precisa-conhecer/>. Acesso em: 21 abr. 2021. Citado na página 17.

AMAZON AWS. 2021. Disponível em: <https://aws.amazon.com/pt/free>. Acesso em: 21 abr. 2021. Citado na página 35.

ANDROID Studio. 2021. Disponível em: <https://developer.android.com/studio>. Acesso em: 04 nov. 2021. Citado na página 37.

ANNIBAL, A. *As mídias sociais na estratégia de comunicação integrada de marketing*. 2018. Disponível em: <https://professorannibal.com.br/2018/04/03/as-midias-sociais-na-estrategia-de-comunicacao-integrada-de-marketing/>. Acesso em: 21 abr. 2021. Citado na página 15.

BANKS, A.; PORCELLO, E. *Learning React: Modern Patterns for Developing React Apps*. [S.l.]: O'Reilly Media, 2020. Citado na página 18.

CANVA. 2021. Disponível em: <https://www.canva.com/>. Acesso em: 14 set. 2021. Citado na página 51.

CHROME. 2021. Disponível em: <https://www.google.pt/intl/pt-PT/chrome/browser-tools/>. Acesso em: 04 nov. 2021. Citado na página 37.

DEVMEDIA. 2021. Disponível em: <https://www.devmedia.com.br/servicos-restful-verbos-http/37103>. Acesso em: 21 abr. 2021. Citado na página 17.

ENGHOLM, *Análise e Design Orientados a Objetos*. 1. ed. [S.l.]: Novatec, 2017. 376 p. Citado na página 20.

ESLINT. 2021. Disponível em: <https://eslint.org/>. Acesso em: 14 set. 2021. Citado 2 vezes nas páginas 45 e 46.

FLUTTER - Get Started. 2021. Disponível em: <https://docs.flutter.dev/get-started/install/windows>. Acesso em: 04 nov. 2021. Citado na página 37.

GAPHQL. 2021. Disponível em: <https://graphql.org/>. Acesso em: 21 abr. 2021. Citado na página 17.

GIT: –fast-version-control. 2021. Disponível em: <https://git-scm.com>. Acesso em: 3 abr. 2021. Citado na página 26.

GIT For Windows. 2021. Disponível em: <https://gitforwindows.org>. Acesso em: 3 abr. 2021. Citado na página 26.

GITHOOKS com Husky em React JS. 2021. Disponível em: <https://danieldcs.com/eslint-prettier-e-githooks-com-husky-em-react-js/>. Acesso em: 14 set. 2021. Citado na página 47.

GITHUB.COM. Github. 2021. Disponível em: <https://github.com/>. Acesso em: 3 abr. 2021. Citado na página 26.

GUEDES, G. *UML 2 - Uma Abordagem Prática - 3ª Edição*. Novatec Editora, 2018. ISBN 9788575226469. Disponível em: <https://books.google.com.br/books?id=RUdLDwAAQBAJ>. Citado na página 21.

HOSTINGER. 2021. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-cms/>. Acesso em: 21 abr. 2021. Citado na página 15.

KONSHIN, K. *Next.js Quick Start Guide: Server-side rendering done right*. [S.l.]: Packt Publishing, 2018. 164 p. Citado na página 19.

LUMIS. 2021. Disponível em: <https://www.lumis.com.br/blog/o-que-e-headless-cms-e-como-ele-pode-ajudar-na-gestao-da-experiencia-dos-seus-clientes.htm>. Acesso em: 21 abr. 2021. Citado na página 16.

MARINHO, L. *Iniciando com Flutter Framework: Desenvolva aplicações móveis no Dart Side!* Casa do Código, 2020. ISBN 9786586110289. Disponível em: <https://books.google.com.br/books?id=EcPxDwAAQBAJ>. Citado na página 19.

MITCHELL, D.; AKOPKOKHYANTS, S.; BALBAERT, I. *Dart: Scalable Application Development*. Packt Publishing, 2017. ISBN 9781787289116. Disponível em: <https://books.google.com.br/books?id=IHg5DwAAQBAJ>. Citado na página 19.

OLIVEIRA, C. *Comunicação Integrada*. São Paulo: Editora Senac São Paulo, 2020. 236 p. (Série Universitária). Citado na página 13.

PEREIRA, C. *Construindo APIs REST com Node.js: Caio Ribeiro Pereira*. Casa do Código, 2016. ISBN 9788555191510. Disponível em: <https://books.google.com.br/books?id=byCjCwAAQBAJ>. Citado na página 18.

PEREIRA, C. R. *Contruindo APIs REST com Node.j*. [S.l.]: Casa do Código, 2016. 180 p. Citado na página 17.

POSTGRES. 2021. Disponível em: <https://www.postgresql.org/download/>. Acesso em: 14 set. 2021. Citado na página 27.

POSTMAN. 2021. Disponível em: <https://www.postman.com/downloads/>. Acesso em: 14 set. 2021. Citado na página 23.

PWA - O que é e quando utilizar. 2021. Disponível em: <https://blog.rocketseat.com.br/pwa-o-que-e-quando-utilizar/>. Acesso em: 14 set. 2021. Citado na página 51.

RESULTADOS Digitais. 2021. Disponível em: <https://resultadosdigitais.com.br/especiais/crm/>. Acesso em: 21 abr. 2021. Citado na página 16.

STORYBOOK. 2021. Disponível em: <https://storybook.js.org/>. Acesso em: 14 set. 2021. Citado na página 51.

STRAPI. 2021. Disponível em: <https://strapi.io//>. Acesso em: 21 abr. 2021. Citado na página 19.

STRAPI Developer Docs. 2021. Disponível em: <https://strapi.io/documentation/developer-docs/latest/getting-started/introduction.html>. Acesso em: 14 set. 2021. Citado na página 27.

STRAPI Developer Docs. 2021. Disponível em: <https://docs.strapi.io/developer-docs/latest/development/backend-customization/controllers.html#extending-core-controllers>. Acesso em: 11 dez. 2021. Citado na página 32.

STRAPI Developer Docs. 2021. Disponível em: <https://docs-v3.strapi.io/developer-docs/latest/setup-deployment-guides/deployment/hosting-guides/heroku.html>. Acesso em: 11 dez. 2021. Citado na página 33.

STYLED Components. 2021. Disponível em: <https://styled-components.com/>. Acesso em: 14 set. 2021. Citado na página 49.

TELLES, A. *A Revolução das Mídias Sociais: Estratégias de Marketing Digital para Você e Sua Empresa Terem Sucesso nas Mídias Sociais - Cases, Conceitos, Dicas e Ferramentas*. [S.l.]: M. Books, 2010. 200 p. (Série Universitária). Citado na página 15.

TORTOISEGIT: Windows Shell Interface to Git. 2021. Disponível em: <https://tortoisegit.org>. Acesso em: 3 abr. 2021. Citado na página 26.

VERGILI, R. *Relações públicas, mercado e redes sociais*. Summus Editorial, 2014. Disponível em: <https://books.google.com.br/books?id=AlalAwAAQBAJ>. Citado na página 13.

VISUAL Studio Code. 2021. Disponível em: <https://code.visualstudio.com/>. Acesso em: 18 jun. 2021. Citado 2 vezes nas páginas 23 e 37.

WAZLAWICK, S. *Engenharia de software: Conceitos e Práticas*. 2. ed. Rio de Janeiro: Elsevier, 2019. 368 p. Citado na página 20.

WE ARE SOCIAL. *Digital In 2020*. 2020. Disponível em: <https://wearesocial.com/digital-2020>. Acesso em: 20 abr. 2020. Citado na página 13.

ZAKAS, N. *JavaScript de Alto Desempenho: Construa interfaces mais rápidas para aplicações web*. Novatec Editora, 2020. ISBN 9788575227756. Disponível em: <https://books.google.com.br/books?id=5t3bDwAAQBAJ>. Citado na página 18.

A Casos de Uso

Tabela 7 – Caso de Uso: Login

Caso de Uso	Login
Descrição	Realizar a autenticação do usuário na aplicação móvel.
Ator	Usuário.
Entradas e pré-condições	-
Saídas pós-condições	Deverá ser encaminhado para a tela principal do aplicativo.
Fluxo Principal	Deverá acessar a tela de Login. Deverá preecher os campos e-mail e senha. Deverá pressionar o botão de Login. O sistema deverá autenticar o usuário. Deverá encaminhar para página principal do aplicativo.
Fluxo Secundário	Deverá preecher os campos e-mail e senha. Deverá pressionar o botão de Login. O sistema deverá autenticar o usuário. Deverá informar falha na autenticação.

Tabela 8 – Caso de Uso: Conceder Consentimento de acesso à conta do Facebook

Caso de Uso	**Conceder Consentimento de acesso à conta do Facebook.**
Descrição	Realizar a autenticação do usuário na conta do Facebook que deseja gerir através do aplicativo e permitir a concessão do consentimento de acesso à conta.
Ator	Usuário.
Entradas e pré-condições	O usuário deverá ter realizado Login no aplicativo.
Saídas pós-condições	Dados da conta do usuário do Facebook devem ser carregados na Tela.
Fluxo Principal	Deverá acessar a tela de consentimento do Facebook. Deverá pressionar o botão de Login. Deverá preencher os dados solicitados pelo Facebook. O sistema deverá autenticar o usuário. Deverá carregar os dados da conta autenticada.

Tabela 8 continuação da página anterior

Caso de Uso	**Conceder Consentimento de acesso à conta do Facebook.**
Fluxo Secundário	Deverá acessar a tela de consentimento do Facebook. Deverá pressionar o botão de Login. Deverá preencher os dados solicitados pelo Facebook. O sistema deverá autenticar o usuário. Deverá informar falha na autenticação com o Facebook.

Tabela 9 – Caso de Uso: Conceder Consentimento de acesso à conta do Youtube

Caso de Uso	**Conceder Consentimento de acesso à conta do Youtube.**
Descrição	Realizar a autenticação do usuário na conta do Youtube que deseja gerir através do aplicativo e permitir a concessão do consentimento de acesso à conta.
Ator	Usuário.
Entradas e pré-condições	O usuário deverá ter realizado Login no aplicativo.
Saídas pós-condições	Dados da conta do usuário do Canal do Youtube devem ser carregados na Tela.
Fluxo Principal	Deverá acessar a tela de consentimento do Youtube. Deverá pressionar o botão de Login. Deverá preencher os dados solicitados pelo Youtube. O sistema deverá autenticar o usuário. Deverá carregar os dados da conta autenticada.
Fluxo Secundário	Deverá acessar a tela de consentimento do Youtube. Deverá pressionar o botão de Login. Deverá preencher os dados solicitados pelo Google. O sistema deverá autenticar o usuário. Deverá informar falha na autenticação com o Youtube.

Tabela 10 – Caso de Uso: Conceder Consentimento de acesso à conta do Twitter

Caso de Uso	**Conceder Consentimento de acesso à conta do Youtube.**
Descrição	Realizar a autenticação do usuário na conta do Twitter que deseja gerir através do aplicativo e permitir a concessão do consentimento de acesso à conta.
Ator	Usuário.
Entradas e pré-condições	O usuário deverá ter realizado Login no aplicativo.
Saídas pós-condições	Dados da conta do usuário do Twitter devem ser carregados na Tela.
Fluxo Principal	Deverá acessar a tela de consentimento do Twitter. Deverá pressionar o botão de Login. Deverá preencher os dados solicitados pelo Twitter. O sistema deverá autenticar o usuário. Deverá carregar os dados da conta autenticada.
Fluxo Secundário	Deverá acessar a tela de consentimento do Twitter. Deverá pressionar o botão de Login. Deverá preencher os dados solicitados pelo Twitter. O sistema deverá autenticar o usuário. Deverá informar falha na autenticação com o Twitter.

Tabela 11 – Caso de Uso: Visualizar publicações da página pessoal do Instagram

Caso de Uso	**Visualizar publicações da página pessoal do Instagram**
Descrição	Visualizar publicações de uma página pessoal do Instagram previamente conectada ao aplicativo.
Ator	Usuário.
Entradas e pré-condições	O usuário deverá ter realizado Login no aplicativo. O usuário deverá ter realizado a concessão do consentimento de acesso a conta do Facebook.
Saídas pós-condições	Publicações da página pessoal do Instagram previamente conectada ao aplicativo devem ser carregadas na Tela.
Fluxo Principal	Deverá acessar a tela de Feed. Deverá carregar as publicações da conta autenticada.
Fluxo Secundário	Deverá acessar a tela de Feed. Deverá informar falha no carregamento das publicações.

Tabela 12 – Caso de Uso: Visualizar publicações da página pessoal do Youtube

Caso de Uso	**Visualizar publicações da página pessoal do Youtube**
Descrição	Visualizar publicações de um canal do Youtube previamente conectado ao aplicativo.
Ator	Usuário.
Entradas e pré-condições	O usuário deverá ter realizado Login no aplicativo. O usuário deverá ter realizado a concessão do consentimento de acesso a conta do Youtube.
Saídas pós-condições	Publicações do canal do Youtube previamente conectado ao aplicativo devem ser carregadas na Tela.
Fluxo Principal	Deverá acessar a tela de Feed. Deverá carregar as publicações da conta autenticada.
Fluxo Secundário	Deverá acessar a tela de Feed. Deverá informar falha no carregamento das publicações.

Tabela 13 – Caso de Uso: Visualizar publicações da página pessoal do Twitter

Caso de Uso	**Visualizar publicações da página pessoal do Twitter**
Descrição	Visualizar publicações de uma página pessoal do Twitter previamente conectada ao aplicativo.
Ator	Usuário.
Entradas e pré-condições	O usuário deverá ter realizado Login no aplicativo. O usuário deverá ter realizado a concessão do consentimento de acesso a conta do Twitter.
Saídas pós-condições	Publicações da página pessoal do Twitter previamente conectada ao aplicativo devem ser carregadas na Tela.
Fluxo Principal	Deverá acessar a tela de Feed. Deverá carregar as publicações da conta autenticada.
Fluxo Secundário	Deverá acessar a tela de Feed. Deverá informar falha no carregamento das publicações.

Tabela 14 – Caso de Uso: Incluir publicações na página pessoal do Instagram

Caso de Uso	**Incluir publicações na página pessoal do Instagram**
Descrição	Incluir publicações na página pessoal do Instagram previamente conectada ao aplicativo.
Ator	Usuário.
Entradas e pré-condições	O usuário deverá ter realizado Login no aplicativo. O usuário deverá ter realizado a concessão do consentimento de acesso a conta do Instagram.
Saídas pós-condições	A nova publicação deverá ser carregada na Tela.
Fluxo Principal	Deverá acessar a tela de Feed. Deverá ativar o botão de incluir no Instagram. Deverá pressionar o botão de Post ou Video. Deverá adicionar uma mídia através da câmera ou galeria. Deverá pressionar em publicar.
Fluxo Secundário	-

Tabela 15 – Caso de Uso: Incluir publicações na página pessoal do Youtube

Caso de Uso	**Incluir publicações na página pessoal do Youtube**
Descrição	Incluir publicações no canal do Youtube previamente conectado ao aplicativo.
Ator	Usuário.
Entradas e pré-condições	O usuário deverá ter realizado Login no aplicativo. O usuário deverá ter realizado a concessão do consentimento de acesso a conta do Youtube.
Saídas pós-condições	A nova publicação deverá ser carregada na Tela.
Fluxo Principal	Deverá acessar a tela de Feed. Deverá ativar o botão de incluir no Youtube. Deverá pressionar o botão de Post ou Video. Deverá adicionar uma mídia através da câmera ou galeria. Deverá pressionar em publicar.
Fluxo Secundário	-

Tabela 16 – Caso de Uso: Incluir publicações na página pessoal do Twitter

Caso de Uso	**Incluir publicações na página pessoal do Twitter**
Descrição	Incluir publicações na página pessoal do Twitter previamente conectada ao aplicativo.
Ator	Usuário.
Entradas e pré-condições	O usuário deverá ter realizado Login no aplicativo. O usuário deverá ter realizado a concessão do consentimento de acesso a conta do Twitter.
Saídas pós-condições	A nova publicação deverá ser carregada na Tela.
Fluxo Principal	Deverá acessar a tela de Feed. Deverá ativar o botão de incluir no Twitter. Deverá pressionar o botão de Post ou Video. Deverá adicionar uma mídia através da câmera ou galeria. Deverá pressionar em publicar.
Fluxo Secundário	-