UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GABRIEL BARROS DE PAULA

Versão Mobile do Sistema Aqua para Disponibilização de Dados da Qualidade das Águas

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Sérgio Luis Cechin

Porto Alegre
2022

AGRADECIMENTOS

Agradeço primeiramente a Deus pela vida e orientação do caminho a seguir. Também agradeço aos meus pais, minha esposa e minha família por todo apoio e suporte nesta caminhada, a qual não foi fácil tendo que enfrentar muitos desafios. Finalmente agradeço ao professor Sérgio Cechin pela oportunidade de ser orientado por ele neste trabalho de conclusão de curso, e a Universidade Federal do Rio Grande do Sul e Instituto de Informática por toda a sabedoria aprendida e pela excelente estrutura do curso de Ciência da Computação.

RESUMO

O trabalho realizado é o desenvolvimento de um aplicativo *mobile* o qual visa cadastrar e visualizar dados da qualidade das águas, dando continuidade ao Sistema Aqua Web, porém com as vantagens de uma aplicação *mobile*. O desenvolvimento foi realizado com o *framework* Flutter open source do Google e a linguagem de programação Dart, visando aplicações multi-plataformas como Android e iOS. A aplicação permite que os usuários logados consigam cadastrar dados de locais e coletas de águas com a praticidade de um aplicativo para celular. Além disso, será agregado aos locais de coletas de águas as funcionalidades de localização do *GPS*, visualização e escolha da posição no Google Maps e utilização da câmera do celular. Através da inclusão do *Global Positioning System* e escolha no Google Maps a aplicação fornece uma localização mais precisa, além da visualização de uma imagem real tirada pela câmera do celular do local de coleta.

Palavras-chave: Aplicação mobile. Multi-plataforma. Android. Flutter. Dart. Qualidade das águas.

Mobile version of the Aqua System for Water Quality Data Availability

ABSTRACT

The work carried out is the development of a mobile application which aims to register and visualize water quality data, giving continuity to the Aqua Web System, but with the advantages of a mobile application. The development was carried out with Google's open source Flutter *framework* and Dart programming language, targeting cross-platform applications such as Android and iOS. The application allows logged in users to be able to register data on locations and water collections with the convenience of a mobile application. In addition, the features of GPS location, visualization and choice of position on Google Maps and use of the cell phone camera will be added to the water collection sites. Through the inclusion of GPS and choice in Google Maps, the application provides a more accurate location, in addition to viewing a real image taken by the cell phone camera of the collection site.

Keywords: Mobile application. Multi-platform. Android. Flutter. Dart. Water quality.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

SDK	Software Development Kit
API	Application Programming Interface
JWT	JSON Web Token
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
MVP	Minimum Viable Product
GPS	Global Positioning System

SUMÁRIO

1 INTRODUÇÃO

O consumo de água é de vital importância para a sobrevivência de todos os seres vivos na Terra. É necessário controlarmos a qualidade da água para o consumo humano. Um exemplo é que a partir da década de 1970 o Ministério da Saúde através da portaria Nº 52 institui a norma de potabilidade em todo o território nacional (FREITAS; FREITAS, 2005). A quantidade total de água do planeta está dividida em 97% para mares e oceanos (água salgada) e apenas 3% para água doce. Porém retirando-se as gelerias desses 3%, restam menos de 1% para o consumo. Esta quantidade para uso comum vem diminuindo e tornando-se escassa, em média 20% a cada 10 anos (BARROS; AMIN, 2008).

Pesquisadores e companhias de abastecimento realizam um monitoramento das águas através de coleta de amostras. Um exemplo da forma como os dados das coletas são persistidos pode ser visto em Zerwes et al. (2015), onde os dados são salvos em planilhas no Microsoft Excel®2010. É possível utilizar uma planilha eletrônica para realizar esta tarefa, mas existe uma forma mais simples e ágil, através de um sistema Web ou dispositivo *mobile*, com o registro das informações em uma base de dados.

O presente trabalho é uma continuação do Sistema Aqua para Disponibilização de Dados da Qualidade das Águas que foi criado para a plataforma Web, o qual será referenciado como Aqua Web. O principal objetivo desse projeto é a criação de um sistema *mobile* para a plataforma Android que irá facilitar o acesso aos dados já coletados e a realização de novas coletas de amostras de água. O novo aplicativo será referenciado ao longo deste trabalho como Aqua Mobile. O aplicativo Aqua Mobile agregará ao Aqua Web a mobilidade, e a vantagem de utilizar o *GPS* e a câmera do celular nos locais de coletas. O trabalho visa preencher uma lacuna, pois não temos muitos aplicativos móveis relacionados com recursos hídricos (CANTELLE,).

O desenvolvimento do sistema Aqua Mobile utilizou, inicialmente a mesma base de dados do projeto Aqua Web. Porém, com o avanço do desenvolvimento do software foi necessário a realização de alterações em algumas tabelas da base para o acréscimo de informações, as quais serão detalhadas na seção 5.1. Apesar do Aqua Mobile ter sido programado e testado no Android, devido a sua criação ter sido com o *framework* Flutter e linguagem de programação Dart, os quais são multi-plataformas, ele pode ser executado no sistema operacional iOS.

O trabalho está organizado com a apresentação da escolha das tecnologias utilizadas, e logo após a apresentação da arquitetura global das integrações. Em seguida é

apresentado o aplicativo Aqua Mobile, o qual foi dividido nos ajustes realizados na base de dados, arquitetura e implementação do sistema e posteriormente a apresentação das telas do software. Concluímos o presente trabalho com uma análise dos objetivo alcançados e possíveis melhorias futuras.

1.1 Trabalho Relacionado

O Sistema Aqua Mobile está relacionado com o trabalho Sistema Aqua para Disponibilização de Dados da Qualidade das Águas, o qual foi realizado pelo ex-aluno Fernando (BOCK, 2021) para a plataforma Web. O Aqua Mobile é uma continuação deste trabalho, porém para dispositivo *mobile* e plataforma Android. Ambos os trabalhos utilizam a mesma base de dados MySQL.

Durante o desenvolvimento do Aqua Mobile foi necessário uma evolução incremental da base de dados para salvar informações do usuário, da imagem da câmera do celular e descrição do endereço da coleta, as quais serão detalhadas na seção 5.1. Utilizar a mesma base de dados possibilitou ao Aqua Mobile acessar todas as coletas já realizadas pelo sistema Aqua Web, o qual já está sendo utilizado no ambiente de produção.

2 ESCOLHA DAS TECNOLOGIAS UTILIZADAS

2.1 Flutter

O desenvolvimento para dispositivos móveis resume-se ao desenvolvimento para as duas grandes plataformas: Android e iOS. A escolha do Flutter justifica-se porque o desenvolvimento com este *framework* é multiplataforma, ou seja, o código desenvolvido pode ser executado em dispositivos Android, iOS, Desktop e Web. Essa possibilidade foi o fator mais importante na decisão da escolha do Flutter para este projeto.

O React Native é outra possibilidade para o desenvolvimento *mobile*, ele foi criado pelo Facebook em 2015. Porém o React Native não foi escolhido para o desenvolvimento deste projeto pelo fato de existirem alguns bugs e uma complexidade na funcionalidade da localização pelo *GPS*. Outro motivo que fortificou a decisão foi o bug da tela branca que ocorre no funcionamento do React Native.

Caso a escolha não tivesse sido utilizar o *framework* Flutter, haveria a necessidade de ser desenvolvido um código na linguagem Java/Kotlin para os dispositivos Android e outro código fonte na linguagem Object-C/Swift para o sistema operacional iOS. Essa escolha foi muito vantajosa, pois permitiu focar o desenvolvimento na criação de um único código. Outro grande ganho é evitar o retrabalho caso existam duas bases de códigos na manutenção de correção de erros que possam surgir.

O Flutter é um Software Development Kit criado pela Google no final de 2016 inspirado no React, o qual foi criado pelo Facebook. Uma das características de criação do Flutter é manter uma base de código única e implantar em várias plataformas, acelerando e simplificando o trabalho para criar aplicativos de alto desempenho (WU, 2018).

Através de sua interface do usuário o Flutter permite a construção de *widgets*, os quais são a configuração e visualização do estado atual de um componente em tela. Após uma mudança em um *widget*, é iniciado o processo de reconstrução da árvore de renderização para um novo estado. O *framework* inicia a construção da árvore com a execução da função principal main que pega o primeiro *widget* e o torna raiz da árvore de *widgets*. Ao escrever um aplicativo poderão ser usados outros componentes, os quais serão filhos do componente raiz. Assim o *framework* utiliza um conjunto de *widgets*, tornando possível a criação de *apps*. Abaixo estão listados alguns desses *widgets* (FLUTTER, 2022):

- *Text*: Permite a criação de texto estilizado;
- *Row, Column*: Ambos criam layouts flexíveis nas direções horizontal (*Row*) e ver-

tical (*Column*). Utilizam o *layout fexbox* permitindo inúmeras configurações de estilos na tela;

- *Stack*: Esse componente permite que seja colocado inúmeros *widgets* uns sobre os outros na tela; e
- *Container*: Cria um elemento visual retangular que pode agregar restrições a tela do aplicativo como margens e bordas.

Além dos *widgets* citados acima como exemplos, o *framework* possui muitos outros os quais podem ser encontrados no repositório de pacotes oficial de Flutter e Dart (GOOGLE, 2022).

2.2 Dart

Dart é a linguagem de programação desenvolvida pelo Google em 2011 e utilizada pelo *framework* Flutter. Esta linguagem de programação foi utilizada para desenvolver toda a parte *mobile* do projeto. A primeira impressão e durante todo o desenvolvimento do projeto Dart pareceu bem familiar e fácil de aprender. Devido a experiência com desenvolvimento *back-end* com a linguagem C# da Microsoft tive a impressão de ambas serem bem parecidas, então os comandos foram parecendo bem intuitivos, porém com alguns detalhes diferentes. A sintaxe segue o paradigma orientado a objetos e fortemente tipada, mas sendo opcional colocar tipos, pois eles são concluídos pelo Dart (ANDRADE, 2022).

Uma característica interessante introduzida na linguagem a partir da versão 2.12 é a opção *Null safety*, a qual pudemos utilizar no desenvolvimento do projeto. *Null safety* ou segurança nula garante que os tipos no código não podem ser *nullable* por *default*, ou seja, as variáveis não podem conter *null*, evitando erros de variáveis nulas em tempo de execução. Além de reduzir os bugs essa característica permite binários menores e execução mais rápida, pois otimizações do compilador são realizadas caso o sistema de tipos garanta que algo nunca será nulo (DART, 2022b).

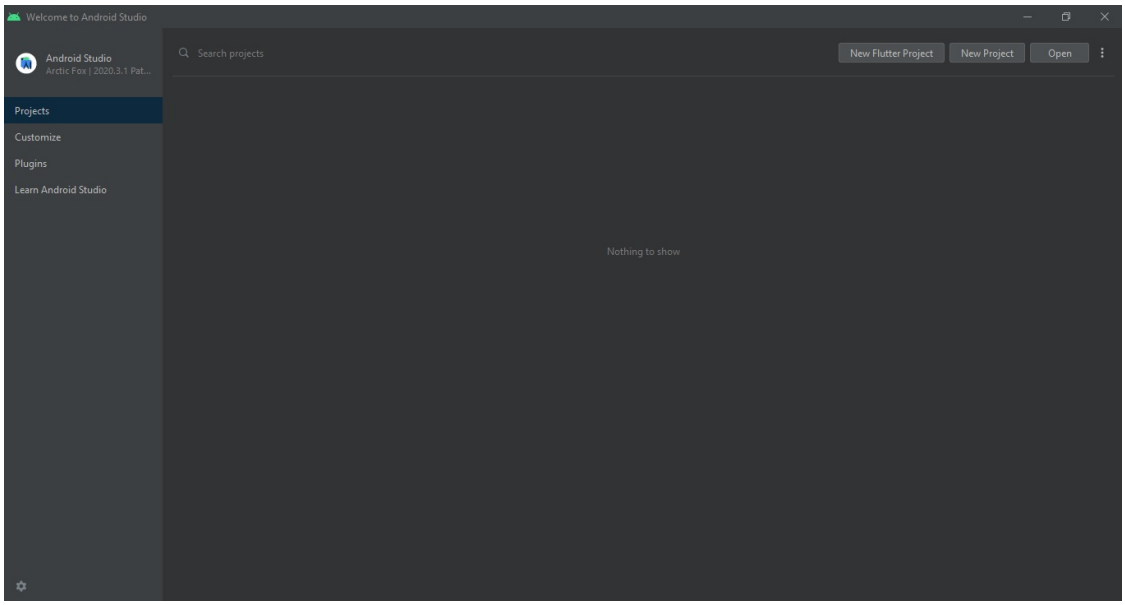
Dart utiliza uma combinação de verificação de tipos estático e verificação em tempo de execução, para garantir a segurança de tipo de uma variável. A verificação de tipos estático permite encontrar bugs em tempo de compilação. Todas as verificações servem para garantir que o programa não entre em estado inválido (DART, 2022c). Dart é uma linguagem de programação poderosa com inúmeras características e em constante

evolução pela comunidade. Mais informações sobre seus detalhes e versões pode ser encontrado no site oficial (DART, 2022a).

2.3 Android Studio - Virtual Device

Foi utilizado durante o desenvolvimento do projeto o Android Studio, para gerar um Virtual Device, e simular o dispositivo *mobile*. O Android Studio é uma ferramenta muito importante, pois permite a criação de qualquer tipo de dispositivo *mobile* Android, podendo ser escolhido o tamanho do aparelho e a versão Android desejada. Na figura 2.1 podemos visualizar a interface do Android Studio Arctic Fox versão 2020.3.1 patch 4 utilizado para o projeto.

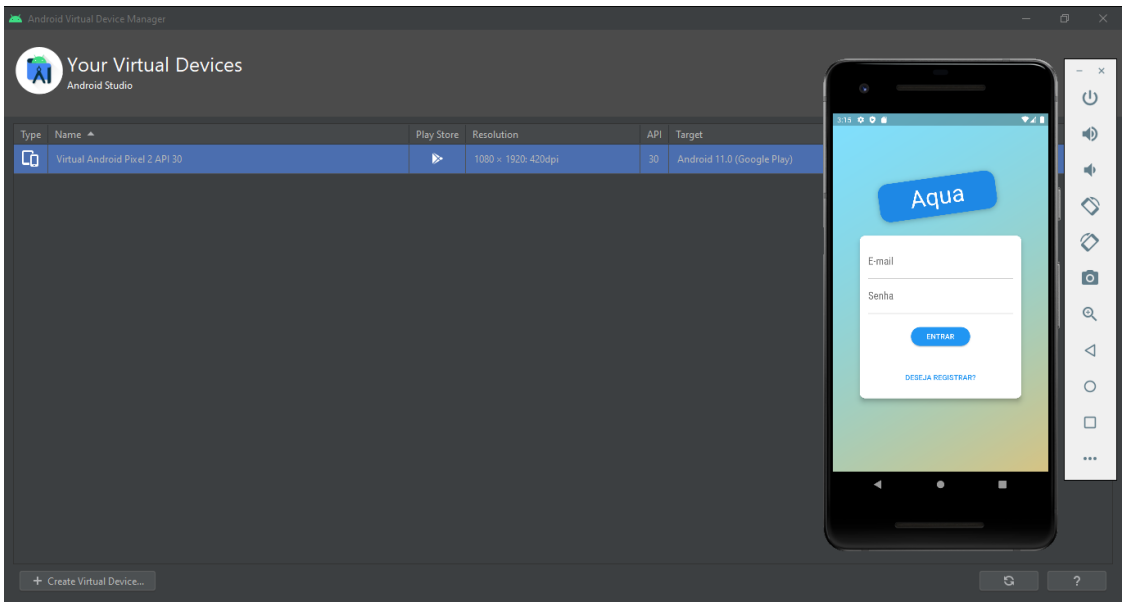
Figura 2.1 – Captura da tela do Android Studio.



Fonte: O Autor

O Android Virtual Device criado foi um Android 11.0 Pixel 2 e Application Programming Interface 30. Na figura 2.2 podemos visualizar o Android Virtual Device Manager e o Virtual Device criado com o aplicativo Aqua Mobile. Dessa maneira foi possível visualizar todas as funcionalidades no aparelho virtual, o qual simula um celular real, enquanto as mesmas eram desenvolvidas.

Figura 2.2 – Captura da tela do Android Virtual Device.



Fonte: O Autor

2.4 Google Maps - APIs e Serviços

Para a visualização dos mapas na tela do Aqua Mobile e localização dos locais das coletas foram utilizados as API's e serviços do Google Maps. Diferentemente do projeto Aqua Web que utilizou o OpenLayers API open-source e gratuita, no projeto Aqua Mobile foi adotado o serviço pago da Google. A escolha pelas APIs e serviços do Google foi pelo fato deles já serem integrados com o sistema operacional Android e *framework* Flutter. O que gerou uma complexidade menor para o desenvolvimento da funcionalidade do mapa e localização da posição de coleta no aplicativo *mobile*. Com isso foi possível obter a latitude, longitude e endereço das posições de coletas de forma nativa a plataforma. As APIs e serviços utilizados da Plataforma Google Cloud foram os seguintes:

- Maps SDK for Android: é o Software Development Kit que permite ao aplicativo Android adicionar mapas com base nos dados do Google Maps e lidar com os gestos do usuário como cliques e arrastos no mapa (DEVELOPERS, 2022a);
- Maps Static API: Permite a criação da imagem do Google Maps no aplicativo com base nos parâmetros (latitude e longitude) enviados na URL por meio de uma solicitação Hypertext Transfer Protocol (DEVELOPERS, 2022b); e
- Geocoding API: é a API utilizada para converter as coordenadas geográficas (latitude e longitude) em um endereço completo;

2.5 .NET Core e C#

O .NET Core é uma plataforma para desenvolvimento de software open-source criada e mantidade pela Microsoft. Ela permite a criação de diversos tipos de aplicativos como *microservices*, funções sem servidor, Aplicativos Web, Web APIs, etc. No projeto Aqua Mobile foi utilizado o .NET Core para a criação de uma Web API na linguagem de programação C#. A Web API tem o objetivo de criar e disponibilizar serviços HTTP para o Aqua Mobile consultar a base de dados MySQL, a qual contém todas as tabelas do projeto Aqua Web e as do próprio Aqua Mobile.

A vantagem de criar uma Web API com a plataforma .NET Core é que os aplicativos podem ser executados por muitos sistemas operacionais como Windows, macOS, Linux, Android, iOS, tvOS e watchOS (MICROSOFT, 2022). Foram criados na Web API serviços necessários para o aplicativo *mobile*, os quais permitem consultas de criação, edição, atualização e de exclusão dos dados nas tabelas do sistema. Também foi utilizado como autenticação o JSON Web Token para acesso aos métodos expostos.

A escolha da linguagem de programação C# foi devido a grande experiência do autor com a linguagem, além de ser uma linguagem da Microsoft, o que facilita o desenvolvimento e compatibilidade com a Web API e suas bibliotecas. A utilização dessa camada de serviço traz ao projeto grandes vantagens, pois encapsula o acesso aos dados do sistema. Com isso, além de expor o acesso aos dados para vários tipos de sistemas, independente da linguagem de programação ou sistema operacional, também foi agregada segurança ao projeto permitindo o acesso somente a quem está autenticado.

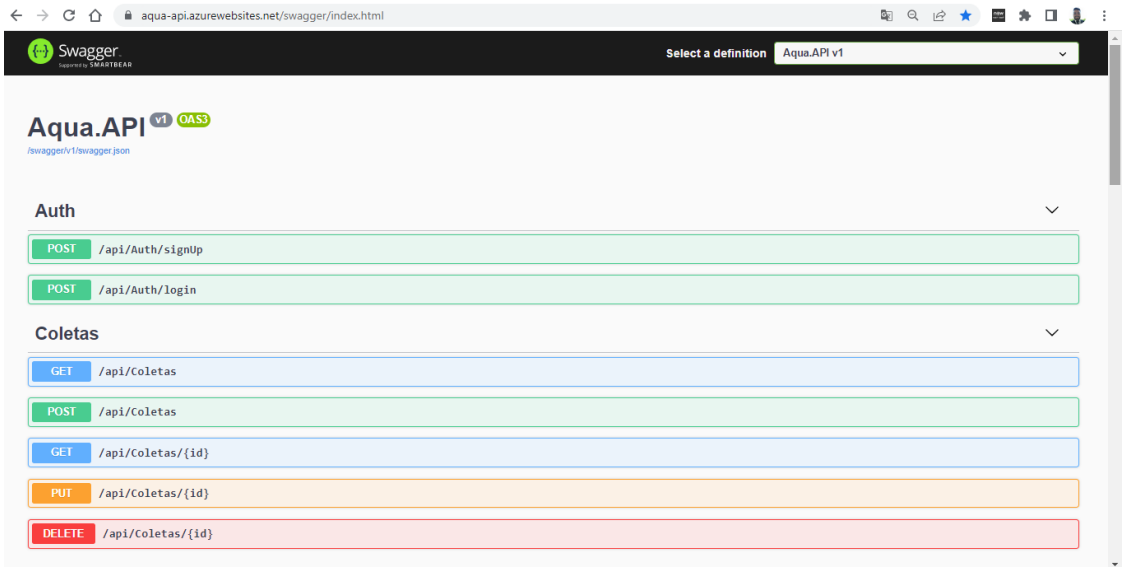
2.6 Swagger

O Swagger é uma ferramenta open-source criada em 2011 que permite documentar a interface de APIs Representational State Transfer com a utilização de JSON (SWAGGER, 2022). O Swagger foi utilizado para documentar a Web API do *back-end* de forma simples e eficiente, permitindo priorizar o desenvolvimento das funcionalidades. Essa ferramenta permite visualizar as alterações em tempo de desenvolvimento em uma página web. Com essa automatização o processo de documentação dos métodos se torna otimizado e com menos erros de digitação.

A API com a utilização do Swagger manteve o padrão de interface REST para todos os métodos expostos, gerando uma facilidade para os usuários que irão consumir

seus serviços. Na figura 2.3 pode-se verificar uma imagem da geração da documentação para o end-point Auth e Coletas.

Figura 2.3 – Captura da tela do Swagger.



Fonte: O Autor

2.7 MySQL

Para manter a compatibilidade com o projeto Aqua Web foi mantida a mesma base de dados com o sistema MySQL. Além de ser uma boa opção por ser open-source e gratuito, também já existiam todos os dados do sistema Aqua anterior. Para atender as necessidades do sistema *mobile* como *login* e imagem da câmera por exemplo, algumas tabelas sofreram alterações para a inclusão de novos campos.

Para acessar a base de dados local e a publicada no servidor remoto foi utilizada a ferramenta de interface gráfica do usuário MySQL Workbench. Essa ferramenta facilitou bastante o trabalho na manipulação de *queries* para validar os dados retornados e consistidos pelo aplicativo *mobile*.

2.8 Hospedagem Microsoft Azure - Service Application

O serviço de nuvem da Microsoft Azure foi utilizado para publicar a Web API. A escolha desse servidor foi pelo fato de já possuir grande experiência com ele em projetos

comerciais e também por ser gratuito para um Serviço de Aplicativo. O serviço de nuvem da Microsoft também possui integração com o repositório do git Azure DevOps, facilitando futuras automatizações como criação de pipelines para publicação. É um servidor de fácil publicação tanto por alguma Integrated Development Environment como o Visual Studio Code, e possui grande disponibilidade e segurança garantida pela Microsoft.

O Microsoft Azure também oferece de forma gratuita aos Serviços de Aplicativos um Application Insights, o qual ajuda muito a encontrar e solucionar possíveis bugs no sistema em tempo de execução. Após a publicação no servidor da Web API o serviço ficou com a seguinte url de endereço: <https://aqua-api.azurewebsites.net/>. Acessando esta url podemos visualizar a documentação do Swagger e realizar requisições HTTP ao serviço conforme a figura 2.3.

2.9 Hospedagem KingHost

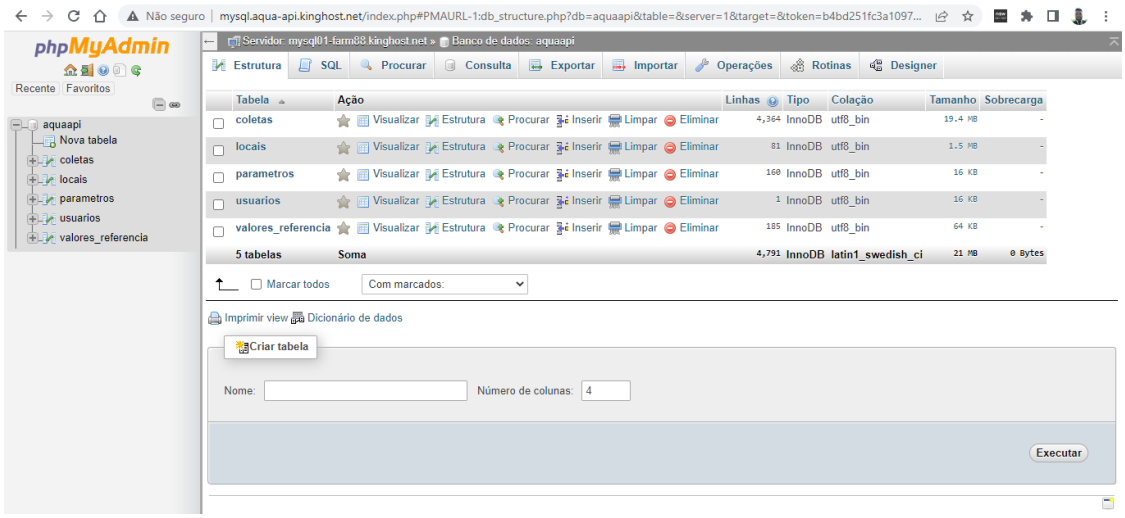
O serviço de hospedagem da KingHost foi utilizado para armazenar a base de dados MySQL em um servidor. Como já mencionado na seção 2.7 foi utilizado para esse projeto a mesma base de dados, porém até o início do desenvolvimento do presente projeto a base de dados utilizada no sistema Aqua Web não estava disponível em nenhum servidor para consulta. Devido a essa circunstância, para iniciarmos o projeto *mobile* foi instalado uma instância da base de dados MySQL no computador local para conseguirmos trabalhar com os dados e seguir com o desenvolvimento.

Ao término do desenvolvimento do sistema Aqua Mobile surgiu a necessidade de validarmos todas as integrações do *front-end* com o *back-end* em um sistema on-line, ou seja, todo o código publicado em servidores expostos na Internet. Foi tentado algumas opções para hospedar a base de dados de forma gratuita, porém nenhuma teve sucesso devido as restrições de tamanho de armazenamento oferecidas para uma versão gratuita. A base de dados do projeto *mobile* enfrentou essa limitação do tamanho de dados oferecido, pois na versão *mobile* passamos a salvar as imagens dos locais e das coletas o que gerou um armazenamento maior de dados para cada registro nas tabelas.

Foi analisada a questão de custo para hospedar uma base de dados MySQL e também a complexidade para realizar a importação dos dados existentes para a nova base de dados. Os serviços de hospedagem analisados foram o Google Cloud Platform (CLOUD, 2022), Microsoft Azure (AZURE, 2022) e KingHost (KINGHOST, 2022). A escolha do servidor KingHost foi pelo fato de ser o valor mensal mais baixo entre os três analisados

e também por eles possuirem um gerenciador on-line de base de dados MySQL chamado PhpMyAdmin, o qual é bem conhecido na comunidade de desenvolvedores MySQL. Na figura 2.4 pode-se verificar a interface gráfica on-line do PhpMyAdmin disponibilizado pela KingHost.

Figura 2.4 – Interface Gráfica on-line do PhpMyAdmin na KingHost.



Fonte: O Autor

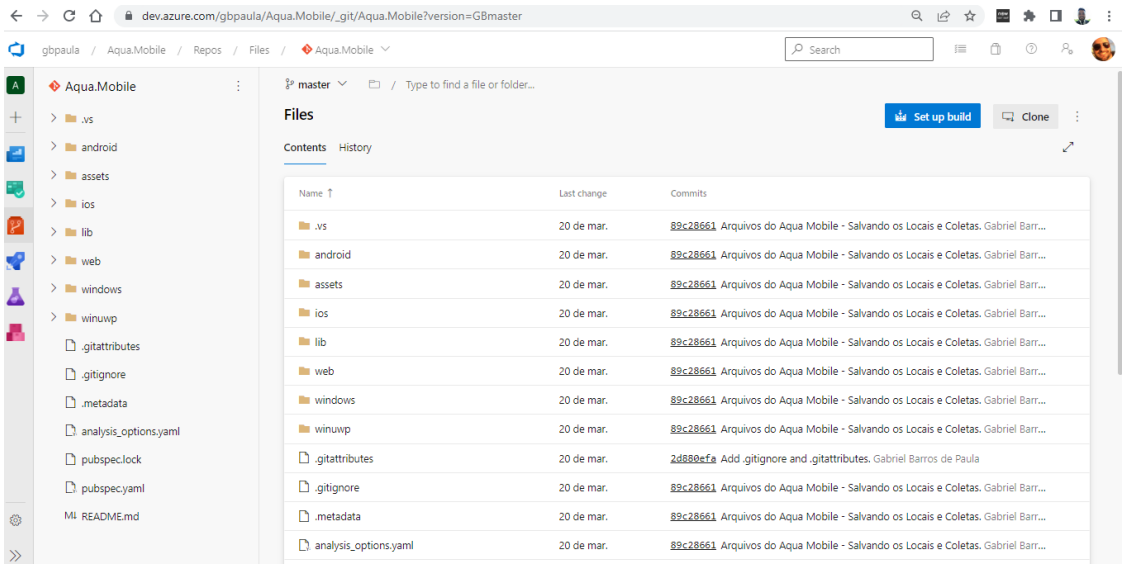
2.10 Microsoft Azure DevOps

O Microsoft Azure DevOps foi o repositório *git* dos códigos fontes utilizado no sistema Aqua Mobile. A escolha por esse repositório deveu-se a ser gratuito e pela simples integração com as IDEs do Visual Studio Code e Visual Studio 2019, utilizadas durante o desenvolvimento do *front-end* e *back-end* respectivamente. Além da simplicidade de integração, ele também oferece a possibilidade de criação de um Pipeline de publicação automatizado. A possibilidade de automatizar a publicação agiliza muito o processo de *deploy* e diminui a possibilidade de erros humanos. A automatização do *deploy* no sistema Aqua Mobile é muito útil pelo fato de termos dois repositórios, um para o código do *front-end* e outro para o *back-end*.

O fato do Microsoft Azure DevOps trabalhar com o versionamento do git foi essencial para a organização do código fonte do projeto. Isso se deve a característica da arquitetura utilizada, pois são dois projetos separados com códigos fonte diferentes, um escrito em Dart e outro em C#. Manter essa organização e rastreio das alterações é essencial em um projeto desse tamanho e com a quantidade de arquivos fonte desenvolvidos.

A figura 2.5 demonstra como a *branch master* do projeto *front-end* do Aqua Mobile está organizado no repositório DevOps da Microsoft. O mesmo ocorre para o projeto de *back-end* Web API.

Figura 2.5 – Branch master do Aqua Mobile no repositório da Microsoft Azure DevOps.



Fonte: O Autor

2.11 Google Play Store

A Google Play Store é a loja oficial da Google para distribuição de aplicativos para o sistema operacional Android. Como o ambiente de desenvolvimento escolhido pelo projeto Aqua Mobile foi o Android, então será publicado o aplicativo na Google Play Store. A grande vantagem é permitir que a última versão do Aqua Mobile possa ser instalado de forma simples e confiável através desse serviço.

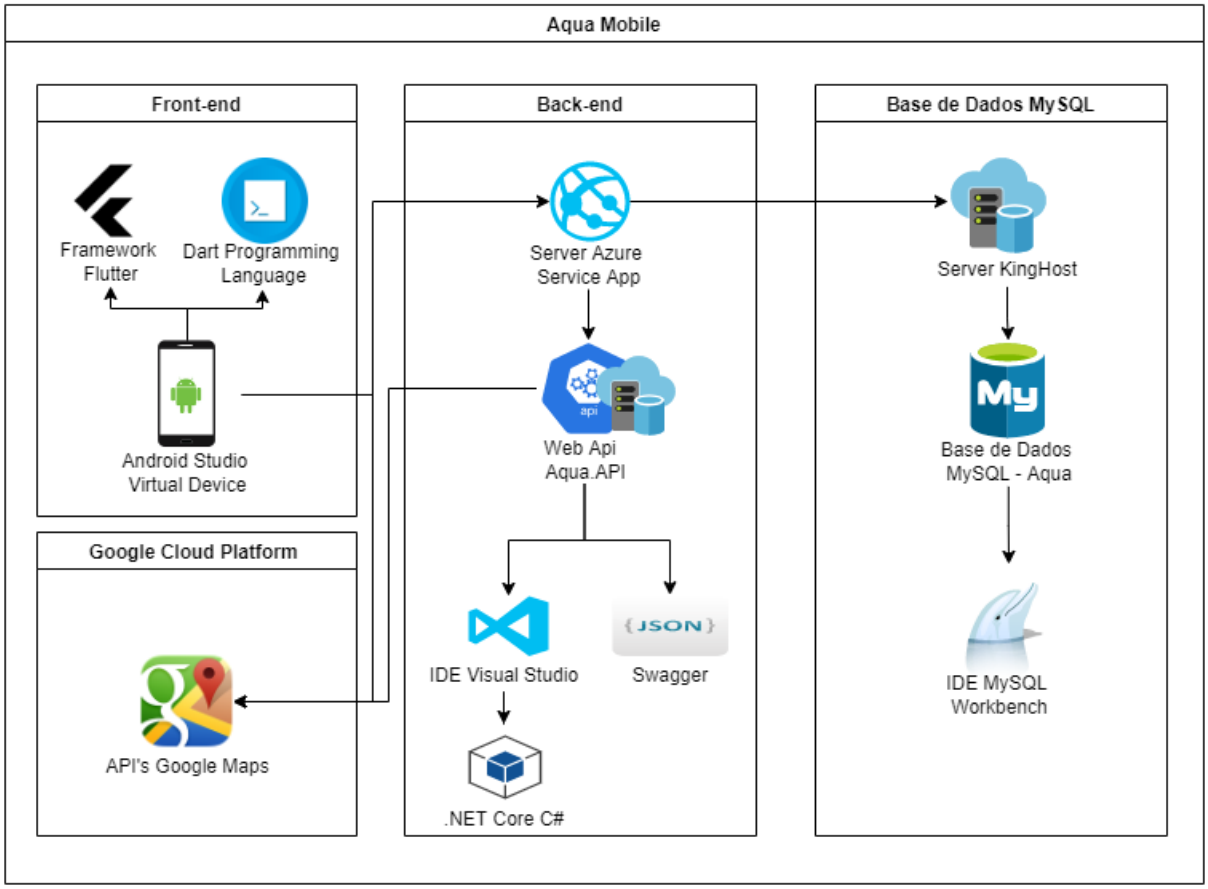
3 ARQUITETURA DA APLICAÇÃO

O desenvolvimento de um aplicativo envolve a criação de diversas funcionalidades que podem ser organizadas por área de utilização. O Aqua Mobile foi construído com a união de 4 grandes áreas, conforme a descrição:

- *Front-end*: possui duas integrações, sendo a primeira com o *back-end* para efetuar o *login*/cadastro e acesso as informações da base de dados, e a segunda com os serviços de localização do Google Maps; e
- Serviços do *Google Cloud Platform*: recebe todas as solicitações de localização do *front-end* e *back-end*; e
- *Back-end*: é responsável por responder todas as solicitações do *front-end* para autenticação e acesso a base de dados. Possui duas integrações, sendo a primeira com a base de dados e a segunda com os serviços de localização do Google; e
- Bases de Dados MySQL: só realiza a comunicação com o *back-end* para responder com as informações salvas na base de dados;

A figura 3.1 mostra todas as integrações realizadas pelas áreas citadas acima junto com as principais tecnologias utilizadas no desenvolvimento. O desenvolvimento destas 4 áreas separadas em camadas, possibilita um desacoplamento do sistema. Ou seja, a maneira como está estruturada a aplicação permite que a Web API Aqua API forneça o acesso a base de dados para qualquer tipo de sistema, como por exemplo uma aplicação *mobile*, web ou desktop. Outra possibilidade desse modelo de arquitetura onde a Web API Aqua API encapsula o acesso a base de dados, é a troca para qualquer outro tipo de banco de dados, como substituir de MySQL para SQL Server.

Figura 3.1 – Arquitetura da Aplicação.



Fonte: O Autor

4 COMPARAÇÃO DAS MUDANÇAS REALIZADAS

Essa seção irá discutir as principais mudanças de funcionalidades que foram realizadas em comparação com o sistema Aqua Web devido a necessidade de diferentes plataformas. A figura 4.1 mostra o resumo das alterações realizadas. De forma detalhada estão explicadas as mudanças e os motivos para sua realização:

- Tabelas da Base de Dados: a mudança na base de dados foi o salvamento da imagem capturada no Local de Coleta, pois o sistema Aqua Web não possibilitava salvar imagens.
- Visualização dos Mapas: O Aqua Web utilizou o serviço da api gratuita OpenLayers, porém o Aqua Mobile utilizou o serviço da api paga do Google Maps devido a fácil integração com o *framework* Flutter.
- Geração de Relatórios e Filtros de Pesquisas: o sistema Aqua Web realizou a implementação de relatórios e filtros para pesquisas, porém na primeira versão do Aqua Mobile ainda não foram implementados.
- Localização do Local de Coleta: a utilização do *GPS* do celular possibilitou ao usuário uma melhor experiência, pois ele abstrai a necessidade de digitar o município, a latitude e a longitude.

Figura 4.1 – Resumo das Mudanças Realizadas.

	AQUA WEB	AQUA MOBILE
Tabelas da Base de Dados	Não possui imagem.	Inserção das imagens da câmera na base de dados.
Visualização dos Mapas	OpenLayers api gratuita.	Google Maps api paga utilizada devido a facilidade de integração com o framework Flutter.
Geração de Relatórios	Desenvolvido.	Ainda não desenvolvido.
Filtros para Pesquisas	Desenvolvido.	Ainda não desenvolvido.
Localização do Local de Coleta	Em função da seleção no mapa.	Em função da seleção no mapa e também com a utilização do gps do celular. Abstrai a necessidade de precisar selecionar no mapa ou precisar digitar o município, a latitude e a longitude.

Fonte: O Autor

5 APLICATIVO AQUA MOBILE

O aplicativo Aqua Mobile é uma extensão para dispositivos móveis Android do sistema Aqua Web desenvolvido pelo ex-aluno Fernando Garcia Bock em sua monografia (BOCK, 2021). O objetivo de implementar o sistema em um dispositivo *mobile* é disponibilizar às pessoas que irão realizar as coletas as vantagens inerentes a um celular. Essa mobilidade traz vantagens tais como a possibilidade de realizarem a coleta e tirarem foto do material coletado com a câmera do celular, e utilizar o *GPS* do dispostivo para marcar com mais precisão a localização da coleta.

Visando o objetivo de desenvolver um sistema *mobile* para a disponibilização de dados da qualidade das águas, o projeto seguiu a metodologia *Minimum Viable Product*, a qual pode ser entendida como entregar ao usuário um produto mínimo viável. Por ser um sistema novo e *mobile* decidiu-se não desenvolver todas as características do projeto Aqua Web. Algumas dessas funcionalidades seriam de difícil implementação devido as características de um aplicativo para celular, como a restrição do tamanho da tela.

Analisando as peculiaridades de um sistema *mobile* decidiu-se aplicar o esforço de desenvolvimento em funcionalidades que agregariam mais valor para o usuário do sistema. Após uma análise do contexto, percebeu-se que a função principal seria a possibilidade de visualizar as coletas já realizadas e existentes na base de dados e poder criar uma coleta com a utilização da câmera e *GPS* do dispositivo.

A funcionalidade de coleta traz implícita a necessidade de realizar o cadastro de um local, devido as restrições de chaves estrangeiras da base de dados. Sendo assim, outro requisito inicial era a possibilidade de listar os locais já cadastrados na base de dados e poder realizar o cadastro de um novo local. Por existir a possibilidade de utilizar a câmera e o *GPS* do celular, tanto o cadastro de um local quanto de uma coleta receberam informações adicionais para permitir salvar uma foto no momento de criação de um registro. Também foi incluída a funcionalidade de poder selecionar o local atual ou escolher no mapa do Google em função da posição atual do *GPS* do celular ao criar um registro.

Em virtude da necessidade de registrar o usuário que realizou a coleta foi necessário criar uma tela de *login* ou cadastro ao iniciar o aplicativo. O desenvolvimento inicial desse projeto se resumiu nos seguintes requisitos em relação a quantidade de telas necessárias para a primeira versão do aplicativo:

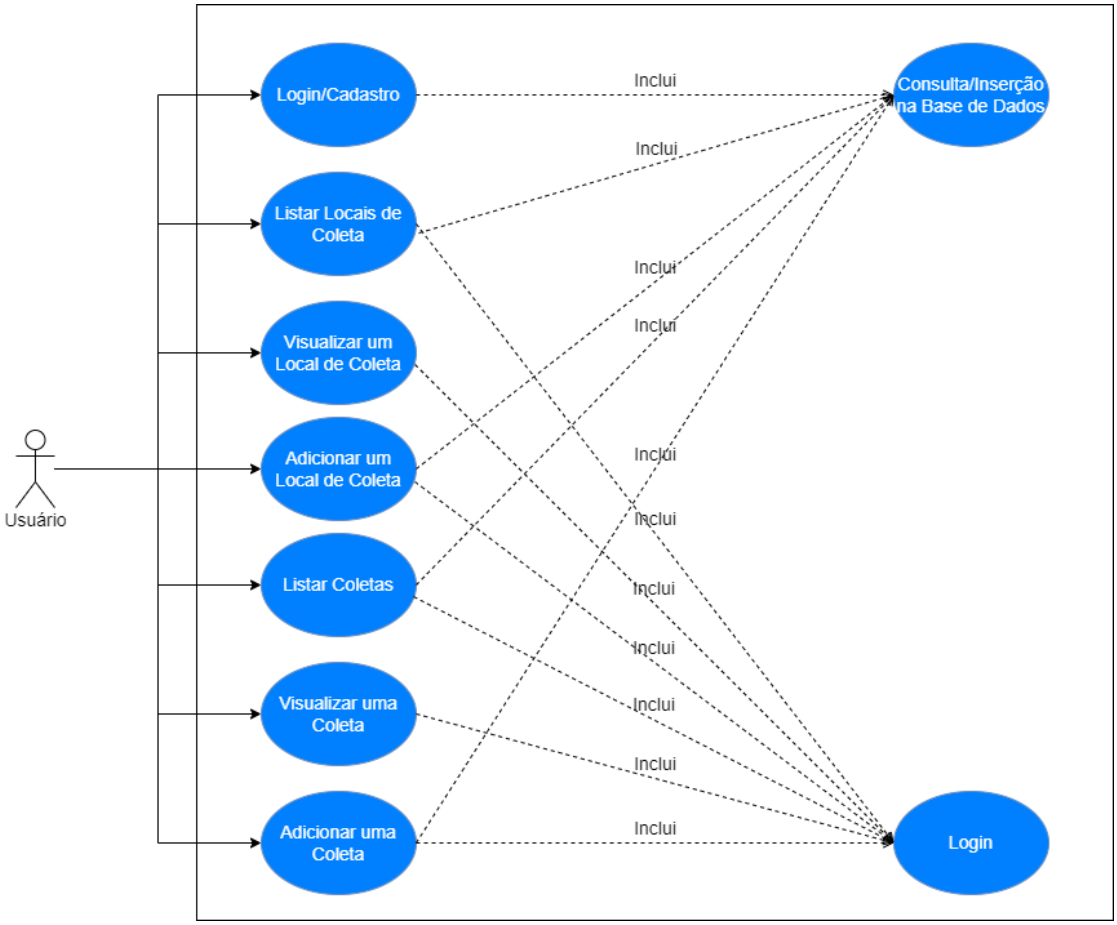
- Tela de *Login*/Cadastro: a primeira tela do sistema será para realizar o *login* de um usuário já existente no sistema ou cadastro de um novo usuário. A tela de *login* será

composta por e-mail e senha do usuário; e

- Tela de listagem dos Locais: essa tela irá listar todos os locais cadastrados e terá um botão para inserir um novo local; e
- Tela de visualizar um Local: essa tela irá permitir visualizar os dados de um local selecionado na listagem de locais; e
- Tela de listagem das Coletas: essa tela irá listar todos as coletas cadastradas e terá um botão para inserir uma nova coleta; e
- Tela de visualizar uma Coleta: essa tela irá permitir visualizar os dados de uma coleta selecionada na listagem de coletas; e
- Tela de visualizar ou selecionar no mapa: essa tela irá permitir visualizar uma posição já escolhida ou selecionar uma nova posição no mapa;
- Menu lateral: esse menu será lateral e terá a opção de expandir ou recolher conforme o clique do usuário. Ele terá as opções de navegar para a tela de listagem de locais ou listagem de coletas ou sair do sistema;

Na análise realizada foi possível identificar os casos de uso dos requisitos em relação a interação com o usuário. Esses casos de uso podem ser vistos na figura 5.1.

Figura 5.1 – Diagrama de casos de uso dos requisitos.



Fonte: O Autor

Diferente do sistema Aqua Web não será implementada na primeira versão do sistema *mobile* a geração de relatórios e possibilidade de filtros dinâmicos. De forma explicativa os casos de uso são contextualizados na listagem abaixo:

- *Login*/Cadastro: o usuário deve realizar o *login* no aplicativo através de seu e-mail e senha. Um novo usuário deve poder realizar o seu cadastro no sistema através da inserção de e-mail, senha e confirmação de senha. Após autenticado o usuário deve poder acessar o aplicativo.

- Visualizar Lista de Locais de Coleta: usuário deve poder visualizar todos os locais que estão cadastrados na base de dados com sua imagem quando houver sido cadastrada e descrição do local de coleta. Obs: no caso de locais antigos já cadastrados na base de dados foi inserido a imagem de mapa do Google.

- Visualizar um Local de Coleta selecionado: usuário deve poder selecionar um local da lista de locais e visualizar suas informações como a imagem, descrição e botão para poder visualizar a localização em um mapa.

- Inserir um novo Local de Coleta: usuário deve poder clicar no botão com o símbolo + no canto superior direito da tela de listagem de locais e com isso abrir uma tela para cadastro de um novo local de coleta.

- Visualizar Lista de Coletas: usuário deve poder visualizar todos as coletas que estão cadastrados na base de dados com sua imagem quando houver sido cadastrada e descrição da coleta. Obs: no caso de coletas antigas já cadastrados na base de dados foi inserido a imagem de mapa do Google.

- Visualizar uma Coleta selecionada: usuário deve poder selecionar uma coleta da lista de coletas e visualizar suas informações como a imagem, descrição e botão para poder visualizar a localização em um mapa.

- Inserir uma nova Coleta: usuário deve poder clicar no botão com o símbolo + no canto superior direito da tela de listagem de coletas e com isso abrir uma tela para cadastro de uma nova coleta.

5.1 Base de Dados

No desenvolvimento do projeto Aqua Mobile foram mantidas as mesmas 5 tabelas utilizadas no projeto Aqua Web. Porém, em função do desenvolvimento das novas funcionalidades *mobile*, foram necessárias alterações em algumas dessas tabelas. As alterações realizadas não excluíram nenhum dado ou alteraram algum campo existente, justamente para manter a compatibilidade com o sistema Web já existente. Somente foram adicionados campos novos o que não irá gerar nenhum erro de mapeamento ao software já existente.

A camada da base de dados é independente da camada de front-end. A base de dados é exposta pela Web API Aqua API através dos métodos de cada *controller*, permitindo ser consultada pelos projetos Aqua Web e Aqua Mobile. Como o trabalho realizado (BOCK, 2021) já explicou todas os campos das tabelas, por motivo de simplificação somente será explicado as mudanças realizadas para o funcionamento do sistema atual. As mudanças realizadas incluiram 3 tabelas: *locais*, *usuarios*, *coletas*. O objetivo dessas tabelas é salvar as novas informações de *login*, imagem e endereço.

5.1.1 Alterações na Tabela USUARIOS

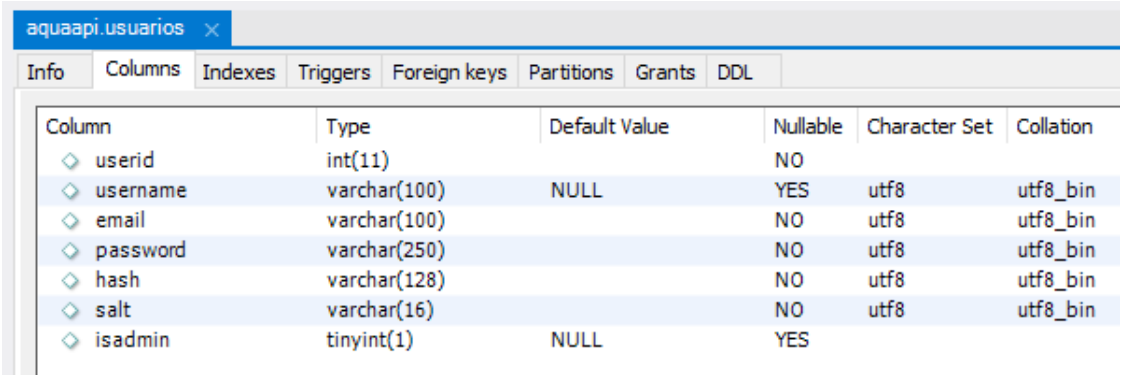
A primeira tabela que sofreu alteração foi a *usuarios*, com informações da tela de *login* e cadastro. Os seguintes campos abaixo foram adicionadas à tabela:

- *userid*: foi criado a coluna *userid* para o armazenamento do id do usuário. Essa coluna é representada por um inteiro que não permite *NULL*.
- *email*: foi criado a coluna *email* para salvar o e-mail do usuário no cadastro de um usuário novo. Essa coluna é consultada em todos os *logins* para validar o e-mail enviado. Essa coluna é representada por um tipo string com tamanho máximo de 100 caracteres que não permite *NULL*.
- *password*: foi criado a coluna *password* para salvarmos a senha do usuário no cadastro de um usuário novo. Essa coluna é consultada em todos os *logins* para validarmos a senha enviada. Essa coluna é representada por um tipo string com tamanho máximo de 250 caracteres que não permite *NULL*.
- *isadmin*: foi criado a coluna *isadmin* para o armazenamento da informação se usuá-

rio possui a permisão de aministrador. Caso o usuário seja administrador, então ele consegue adicionar locais e coletas, caso contrário somente tem a permissão de visualizar os registros. Essa coluna é representada por um tipo inteiro *default NULL*.

Após as alterações realizadas o aplicativo passou a ser capaz de gerenciar se um usuário que está tentando realizar o *login* existe e se a senha informada é válida. Após as alterações a tabela de *usuarios* ficou com a estrutura conforme a figura 5.2.

Figura 5.2 – Estrutura da tabela *usuarios*.



Fonte: O Autor

5.1.2 Alterações nas Tabelas LOCAIS e COLETAS

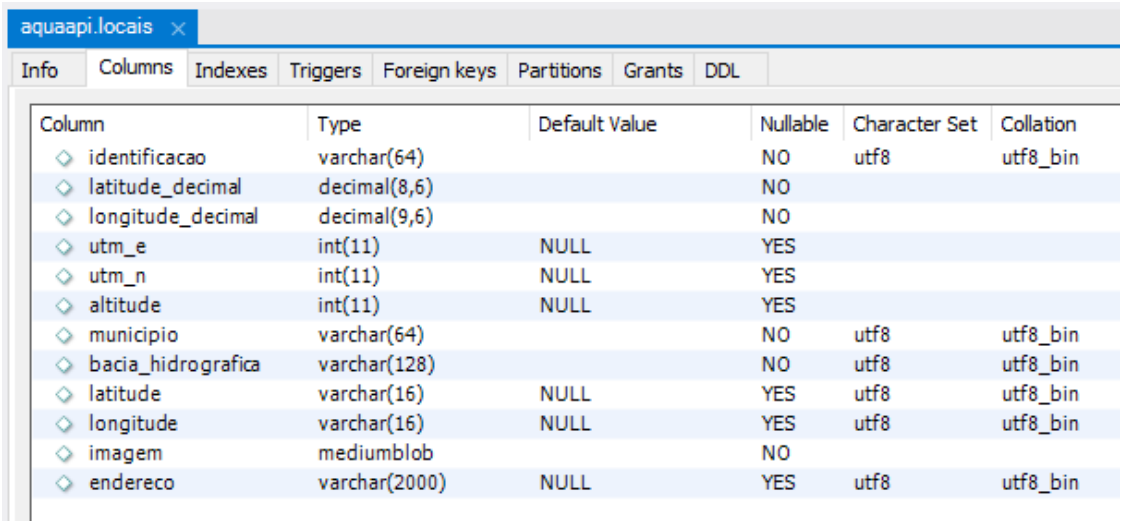
As tabelas *locais* e *coletas* estão em uma seção única, pois ambas as tabelas sofrerão as mesmas alterações. As seguintes colunas abaixo foram adicionadas à tabela:

- *imagem*: foi criada a coluna *imagem* para o armazenamento da foto tirada tanto no cadastro de um local quanto no de uma coleta. Essa coluna é representada por um tipo *mediumblob* que não permite *NULL*. O tipo *mediumblob* é um objeto binário para armazenar uma quantidade de dados binários na base de dados MySQL. A escolha pelo *mediumblob* foi em questão da sua capacidade que é de 16.777.215 bytes (16MB aproximadamente). Esse tamanho é razoável para armazenar uma imagem de qualidade e não sobrecarregar o tamanho do armazenamento dessa linha de registro na tabela de *locais* e *coletas*. Se cada registro dessas tabelas forem muito grandes em função da coluna *imagem*, irá afetar diretamente o tamanho da base de dados. Esse problema se não observado pode acarretar em consultas mais lentas.

- *endereco*: foi criado a coluna *endereco* para salvar essa informação na base de dados de uma forma mais amigável ao usuário humano, por exemplo: Av. Bento Gonçalves, 9500 - Agronomia, Porto Alegre. Essa ideia surgiu da observação que já existe a informação de latitude e longitude na criação de um local ou coleta e seria interessante ao usuário visualizar uma descrição do endereço. Essa coluna é representada por um tipo string com tamanho máximo de 2000 caracteres que permite *NULL*.

Após as alterações realizadas a base de dados ficou pronta para permitir salvar uma imagem no cadastro de um local ou uma coleta. A atualização também permitiu salvar a descrição do endereço. Após os ajustes, as tabelas de *locais* e *coletas* ficaram conforme as figuras 5.3 e 5.4 respectivamente.

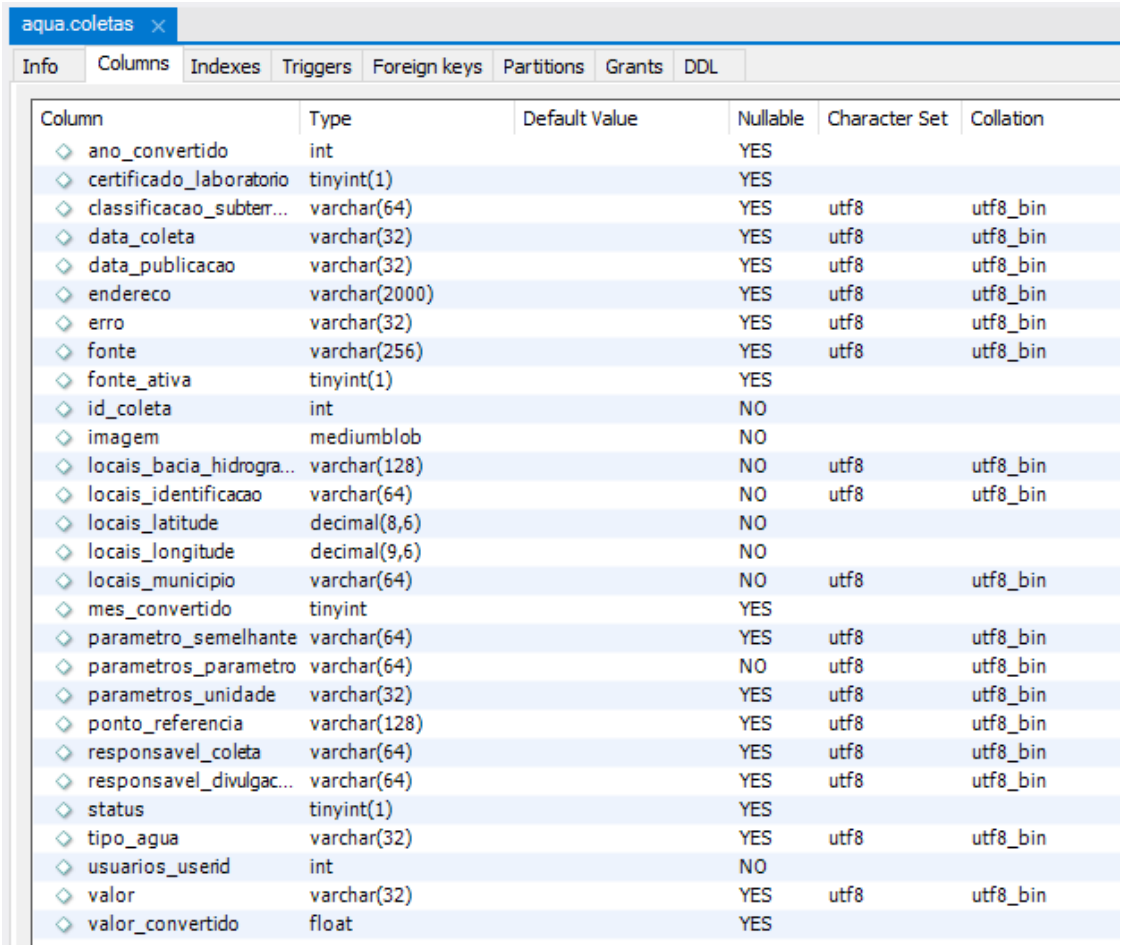
Figura 5.3 – Estrutura da tabela *locais*.



Column	Type	Default Value	Nullable	Character Set	Collation
identificacao	varchar(64)		NO	utf8	utf8_bin
latitude_decimal	decimal(8,6)		NO		
longitude_decimal	decimal(9,6)		NO		
utm_e	int(11)	NULL	YES		
utm_n	int(11)	NULL	YES		
altitude	int(11)	NULL	YES		
municipio	varchar(64)		NO	utf8	utf8_bin
bacia_hidrografica	varchar(128)		NO	utf8	utf8_bin
latitude	varchar(16)	NULL	YES	utf8	utf8_bin
longitude	varchar(16)	NULL	YES	utf8	utf8_bin
imagem	mediumblob		NO		
endereco	varchar(2000)	NULL	YES	utf8	utf8_bin

Fonte: O Autor

Figura 5.4 – Estrutura da tabela *coletas*.



aqua.coletas

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation
ano_convertido	int		YES		
certificado_laboratorio	tinyint(1)		YES		
classificacao_subterr...	varchar(64)		YES	utf8	utf8_bin
data_coleta	varchar(32)		YES	utf8	utf8_bin
data_publicacao	varchar(32)		YES	utf8	utf8_bin
endereco	varchar(2000)		YES	utf8	utf8_bin
erro	varchar(32)		YES	utf8	utf8_bin
fonte	varchar(256)		YES	utf8	utf8_bin
fonte_ativa	tinyint(1)		YES		
id_coleta	int		NO		
imagem	mediumblob		NO		
locais_bacia_hidrogra...	varchar(128)		NO	utf8	utf8_bin
locais_identificacao	varchar(64)		NO	utf8	utf8_bin
locais_latitude	decimal(8,6)		NO		
locais_longitude	decimal(9,6)		NO		
locais_municipio	varchar(64)		NO	utf8	utf8_bin
mes_convertido	tinyint		YES		
parametro_semelhante	varchar(64)		YES	utf8	utf8_bin
parametros_parametro	varchar(64)		NO	utf8	utf8_bin
parametros_unidade	varchar(32)		YES	utf8	utf8_bin
ponto_referencia	varchar(128)		YES	utf8	utf8_bin
responsavel_coleta	varchar(64)		YES	utf8	utf8_bin
responsavel_divulgac...	varchar(64)		YES	utf8	utf8_bin
status	tinyint(1)		YES		
tipo_agua	varchar(32)		YES	utf8	utf8_bin
usuarios_userid	int		NO		
valor	varchar(32)		YES	utf8	utf8_bin
valor_convertido	float		YES		

Fonte: O Autor

5.2 Arquitetura e Implementação do Sistema

A arquitetura do aplicativo Aqua Mobile pode ser dividida em duas partes. A primeira parte é *front-end*, escrito com a linguagem de programação Dart, com a utilização do *framework* Flutter; e a segunda parte é o *back-end*, escrito com a linguagem de programação C#, e com utilização da plataforma .NET Core. Para cada parte será descrita, de forma detalhada, a sua arquitetura, implementação e funcionamento, nas seções 5.2.1 e 5.2.2.

5.2.1 Arquitetura *Front-end*

5.2.1.1 Gerência de Estado

O Flutter fornece *widgets* para construir o aplicativo, os quais podem ser compostos por outros *widgets*, e assim criar uma interface para plataforma iOS e Android (DAGNE, 2019). O Aqua Mobile é composto pelo *widget* principal chamado MultiProvider, que permite gerenciar uma árvore de *widgets* que representa os estados no Flutter. É preciso gerenciar o estado de autenticação, locais, coletas e parâmetros. O objetivo do Provider é manter o estado de um objeto, e com isso poder verificar seu valor e notificar lugares que utilizam esse objeto, para atualizar o seu valor, caso seja atualizado.

Essa abordagem é necessária para o aplicativo, pois é necessário gerenciar os estados dos componentes abaixo e notificar os lugares que utilizam esses componentes pelos motivos explicados a seguir:

- *Auth*: deve ser gerenciado o estado da classe *Auth* para garantir quando o usuário está ou não autenticado. Assim é possível verificar se o usuário deixou de estar autenticado por expirar o token ou por realizar o logout, e direcionar o aplicativo de forma automática para a tela de *login*. É utilizado para essa funcionalidade a biblioteca provider versão 5.0.0 (PACKAGE, 2022f). Além de logout essa classe também é responsável por executar as funções de *login*, cadastro, auto-login e auto-logout. O auto-login permite verificar quando o usuário entra no sistema que ele ainda possui um token de autenticação válido, ou seja, sua data de expiração ainda é posterior a data atual e com isso realiza o *login* na aplicação de forma automática. O auto-logout cria uma classe Timer com a data de expiração do token, garantindo que o logout do sistema seja automático assim que o token expire. As funções de

login e cadastro realizam uma chamada HTTP POST para a Web API Aqua API *controller Auth*.

- *Locais*: deve ser gerenciado o estado da classe *Locais* para garantir que, assim que for alterado alguma informação de um local, seja refletido nas demais telas que estão usando esse objeto. Também é necessário a gerência do estado no caso de preencher a tela com a lista de locais, pois a alteração por inclusão ou exclusão de algum item, deve ser refletido na tela de listagem de locais. Essa classe é responsável por buscar a lista de locais e adicionar um novo local na Web API Aqua API. Estas duas funções realizam uma chamada HTTP a Web API Aqua API *controller Locals*.

- *Coletas*: deve ser gerenciado o estado da classe *Coletas* para garantir que, assim que for alterado alguma informação de uma coleta, seja refletido nas demais telas que estão usando esse objeto. Também é necessário a gerência do estado no caso de preencher a tela com a lista de coletas, pois a alteração por inclusão ou exclusão de algum item deve ser refletido na tela de listagem de coletas. Essa classe é responsável por buscar a lista de coletas e adicionar uma nova coleta na Web API Aqua API. Estas duas funções realizam uma chamada HTTP a Web API Aqua API *controller* Coletas.

- *Parametro*: deve ser gerenciado o estado da classe *Parametro* para garantir que, assim que for alterado alguma informação de um parâmetro, seja refletido nas demais telas que estão usando esse objeto. Também é necessário a gerência do estado ao inserir uma nova coleta, pois o formulário possui um campo DropdownButtonFormField que exibe a listagem de parâmetros para que o usuário possa escolher um item da lista. Essa funcionalidade é necessária para evitar a inserção de qualquer valor e garantir que os parâmetros permaneçam consistentes na base de dados. Essa classe é responsável por buscar a lista de parâmetros na Web API Aqua API para o cadastro de uma nova coleta. Esta função realiza uma chamada HTTP a Web API Aqua API *controller* Parametros.

5.2.1.2 Autenticação

O projeto utilizou para autenticação o padrão JWT, que é uma forma de autenticação segura e compacta, para a transferência de permissões entre duas partes (JONES;

BRADLEY; SAKIMURA, 2015). O JWT é um padrão de autenticação por meio de um token assinado que armazena um objeto JSON com os dados das permissões (DEVMEDIA, 2022). O JWT é criado na *controller Auth* da Web API Aqua API, após a validação do usuário na execução dos métodos de *login* ou cadastro. Assim que a resposta da requisição de *login* é avaliada como sucesso pelo aplicativo *mobile*, é realizada a decodificação da resposta e obtem-se o token, e-mail, id do usuário e data de expiração do token. Essas informações são salvas na memória local do dispositivo utilizando a biblioteca sqflite versão 2.0.0+3 (PACKAGE, 2022g). A partir desse momento toda a gerência de estado da autenticação é feita pelo aplicativo *mobile*. Para decodificar o token JWT recebido do servidor, foi utilizado a biblioteca jwt_decoder versão 2.0.1 (PACKAGE, 2022d).

5.2.1.3 Armazenamento Local

Foi possível utilizar o armazenamento local do dispositivo através da biblioteca SQLite. O armazenamento foi realizado para salvar as informações do usuário como token, e-mail, id de identificação e data de expiração da autenticação. Com isso toda vez que o aplicativo for fechado e aberto posteriormente será validado se há um usuário salvo e se o token ainda é válido. Assim foi possível manter a seção com o usuário e não ser preciso solicitar novamente que o usuário fosse identificado. Essa abordagem trouxe um grande ganho de otimização, pois a informação de usuário já está do lado do *front-end* evitando ter de consultar o *back-end*. Para implementar esse salvamento utilizou-se uma chave de identificação e o objeto json codificado.

5.2.1.4 Estilo Visual

A implementação utilizou o design ThemeData que faz parte do material.io do Google (MATERIAL, 2022). O ThemeData é uma classe que fornece propriedades para ajustar o tema do aplicativo, como por exemplo cor de fundo da tela, cores do texto, etc. Com essa utilização dos padrões do material.io foi possível agilizar o desenvolvimento aproveitando os layout prontos e fornecendo uma aparência expressiva e adaptável ao aplicativo.

5.2.1.5 Lista de Itens

Para melhorar a performance do aplicativo foi utilizado o *widget* ListView.builder. Esse componente, ao invés de criar todos os itens da lista de uma vez, cria os itens à

medida que é feito o *scroll* da tela. E para ficar uma aparência melhor ao usuário enquanto as listas estão sendo carregadas foi utilizado o componente CircularProgressIndicator que fica girando para informar que a busca está sendo realizada.

5.2.1.6 Utilização da Câmera

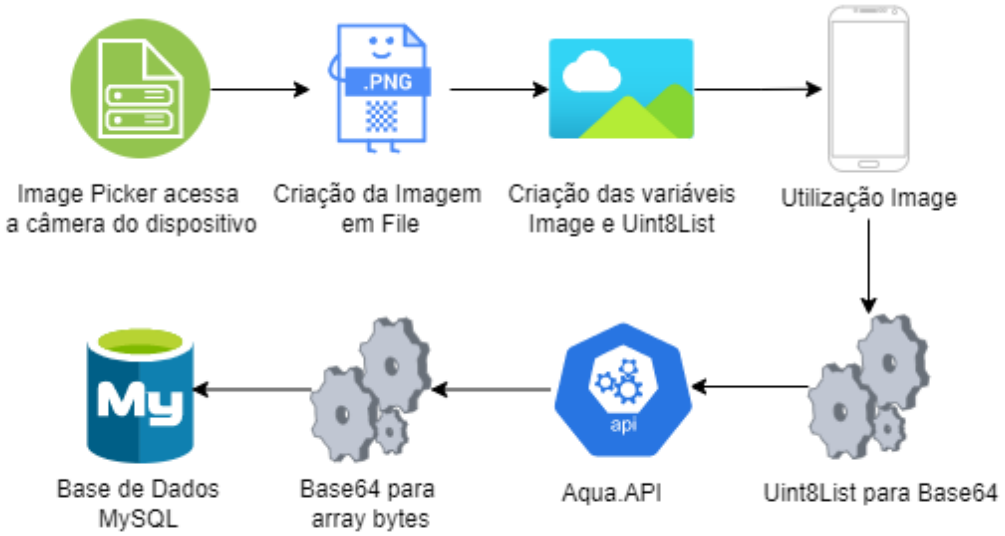
A captura da foto através da câmera do dispositivo foi realizada com a utilização da biblioteca image_picker versão 0.8.3+2. Nesse momento do desenvolvimento ocorreram grandes desafios em relação a salvar a imagem. O problema é que o Flutter trabalha com uma classe Image obtida através da classe ImagePicker que acessa a câmera. A classe Image é responsável por renderizar a imagem na tela do dispositivo. Porém, o MySQL salva a imagem como um array de bytes. Após inúmeras tentativas diferentes não foi possível obter do objeto Image os bytes da Imagem. Depois de algum tempo tentando outras soluções surgiu a ideia de criar um objeto File do Flutter a partir da imagem, e através desse objeto foi possível obter os bytes do arquivo da imagem.

A partir desse ponto foi necessário criar duas variáveis para salvar a imagem, uma do tipo Image e outra do tipo Uint8List (é a maneira como o Dart representa uma lista de bytes). Durante a execução do código do *front-end* utiliza-se a variável que representa o objeto Image para renderizar a imagem na tela do aplicativo. E quando necessário a comunicação com o *back-end* utiliza-se a variável do tipo Uint8List. Para finalizar o processo de envio da imagem para a Web API e salvar na base de dados foi necessário o último passo de converter de Uint8List para base 64. Foi necessário converter a imagem para o formato base 64 para conseguir enviá-la através de uma requisição HTTP. Ao chegar na Web API foi necessário realizar o processo inverso de conversão de base 64 para um array de bytes, e logo após salvar na base de dados.

Sempre que a imagem é transmitida do *front-end* para o *back-end* ou o contrário é realizado esse processo de conversão. E assim conseguir ter o objeto Image no *front-end* e array de bytes no *back-end*, além de utilizar o formato base 64 no envio através das requisições HTTP. A figura 5.5 ilustra o processo para obter a imagem, enviar para a Web API Aqua API e salvar na base de dados.

Para tornar o desenvolvimento do código reutilizável foi criado um componente *widget* ImageInput que encapsula toda a funcionalidade de obtenção da imagem e retorno das variáveis Image e Uint8List. Dessa forma, foi possível reutilizar esse componente nas telas de cadastro de local e coleta. A figura 5.6 ilustra o componente criado ImageInput com uma miniatura da imagem tirada pelo usuário e o botão para abrir a câmera do celular.

Figura 5.5 – Processo de obter a imagem e salvar na base de dados.



Fonte: O Autor

Figura 5.6 – Componente criado ImageInput.

Fonte: O Autor

5.2.1.7 Utilização da Localização e Mapas do Google

Para conseguir utilizar a localização do dispositivo foi necessário instalar a biblioteca location versão 4.3.0 (PACKAGE, 2022e), e para permitir que os mapas do Google fossem utilizados nas telas do Aqua Mobile foi necessário a biblioteca google_maps_flutter versão 2.0.6 (PACKAGE, 2022c). Além dessas bibliotecas foi necessário criar uma conta na Plataforma Google Cloud e acessar os serviços de localização, conforme mencionados na seção 2.4.

A localização implementada permite ao usuário selecionar nas telas de cadastro de local e coletas dois tipos de localização:

- *Localização Atual*: o tipo de localização atual faz utilização do *GPS* do dispositivo e pega a localização corrente do usuário. Após selecionar essa opção é adquirida

a latitude e longitude do celular e chamado o serviço do Google para gerar uma imagem do mapa estática em função da latitude e longitude passadas.

- *Localização a partir da Seleção no Mapa*: se o usuário clicar na opção Selecione no Mapa o aplicativo abre a tela de Mapa. Nessa tela é aberto o *widget* GoogleMap que apresenta um mapa com dados do serviço Google Maps, o qual o usuário pode rolar e escolher qualquer lugar desejado. Essa tela foi configurada para abrir com o mapa centralizado no endereço inicial do Instituto de Informática da UFRGS, latitude -30.068816 e longitude -51.120515. Após selecionado no mapa pelo usuário uma posição é habilitado um checkbox para a operação ser confirmada. Se confirmada a operação, a tela retorna para a tela anterior com a latitude e longitude. Nesse momento é chamado o serviço do Google para gerar uma imagem do mapa estática em função da latitude e longitude passadas.

Semelhante com o que foi realizado no componente ImageInput, a fim de tornar o desenvolvimento do código reutilizável, foi criado um componente *widget* LocationInput que encapsula toda a funcionalidade de obtenção da localização e retorno da latitude e longitude. Dessa forma, foi possível reutilizar esse componente nas telas de cadastro de local e coleta. A figura 5.7 demonstra as duas opções de localização implementadas no aplicativo Aqua Mobile.

Figura 5.7 – Componente criado LocationInput.

Fonte: O Autor

5.2.1.8 Biblioteca Intl

O projeto utilizou a biblioteca intl versão 0.17.0 (PACKAGE, 2022b) para a formatação de datas no padrão que seja usual ao usuário como por exemplo: dia/mês/ano.

5.2.1.9 Biblioteca Http

O projeto utilizou a biblioteca http versão 0.13.3 (PACKAGE, 2022a) para toda a comunicação HTTP realizada entre o aplicativo Aqua Mobile e os demais serviços, como o Google e a Web API Aqua API.

5.2.2 Arquitetura *Back-end*

5.2.2.1 Autenticação JSON Web Token

A autenticação JWT é criada na Web API Aqua API através de uma chave privada e utilização da classe SymmetricSecurityKey. À exceção da *controller* AuthController, que realiza a autenticação, todas as demais *controllers* somente são acessadas através da autenticação por um token válido. O token gerado está configurado para expirar em 2 horas.

5.2.2.2 Documentação com Swagger

Conforme descrito na seção 2.6 toda a Aqua API é documentada através do Swagger. O Swagger é amplamente utilizado em projetos de API's pela sua facilidade de utilização e grande benefício para documentar e testar uma API. Ele fornece inclusive a possibilidade de testar a API com a autenticação por um JWT conforme a figura 5.8.

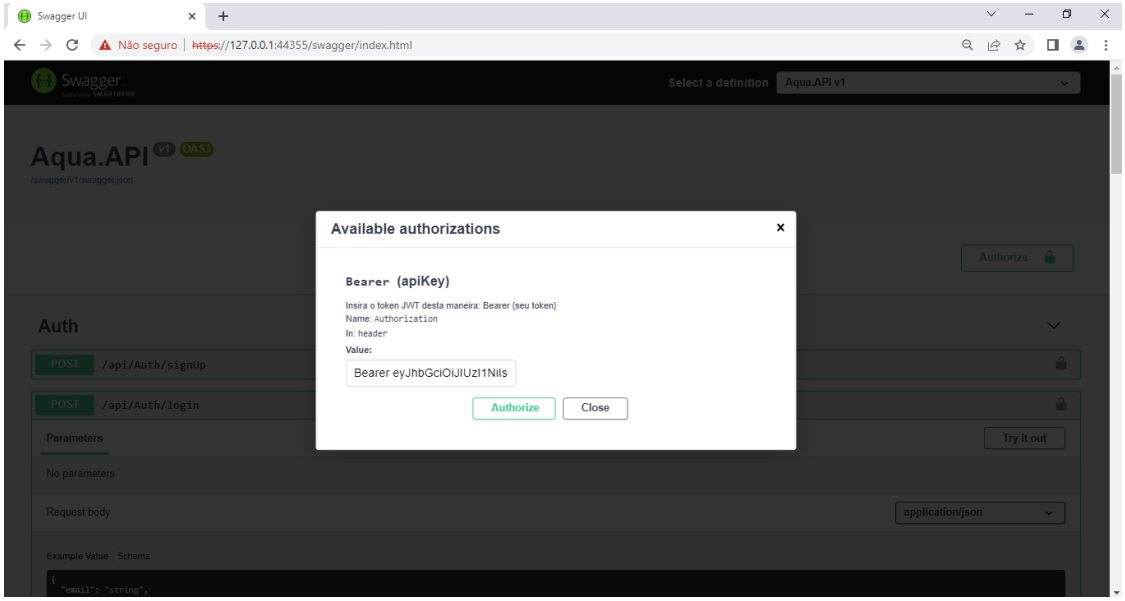
Nas figuras 5.9 e 5.10 podemos verificar toda a documentação gerada pelo Swagger para as *controllers* desenvolvidas. Nas figuras é possível analisar o nome da *controller*, os verbos HTTP utilizados e o caminho para acessar esse serviço. Por exemplo para realizar o *login* utilizar o verbo HTTP POST e caminho: <https://127.0.0.1:44355/api/auth/login> passando no body da requisição as informações do usuário.

5.2.2.3 Models

No projeto foram criadas 7 models para representar os modelos da base de dados e de requisições de serviços. Os modelos criados são:

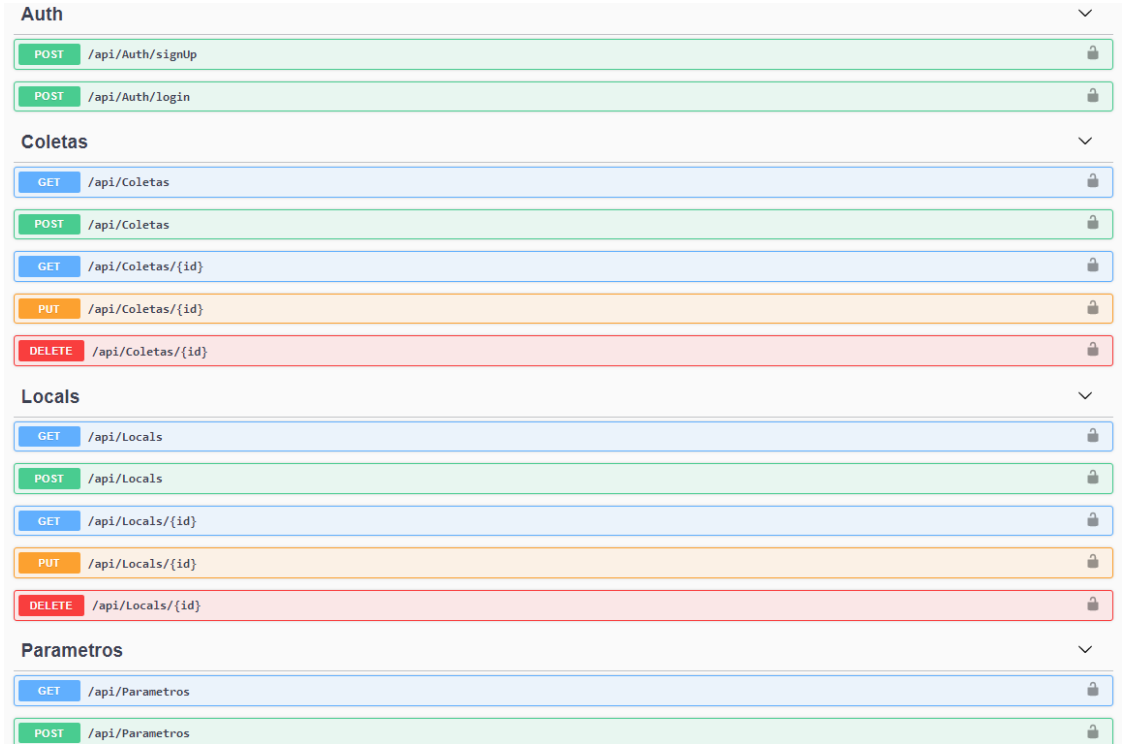
- *Coleta*: essa *model* foi criada para representar uma coleta da tabela da base de dados *coletas*.
- *Local*: essa *model* foi criada para representar um local da tabela da base de dados

Figura 5.8 – Autenticação JSON Web Token no Swagger.



Fonte: O Autor

Figura 5.9 – Documentação gerada pelo Swagger para a API.



Fonte: O Autor

Figura 5.10 – Continuação da documentação gerada pelo Swagger para a API.

Fonte: O Autor

locais.

- *Parametro*: essa *model* foi criada para representar um parâmetro da tabela da base de dados *parametros*.

- *Usuario*: essa *model* foi criada para representar um usuário da tabela da base de dados *usuarios*.

- *ValorReferencia*: essa *model* foi criada para representar um valor de referência da tabela da base de dados *valores_referencia*.

- *ResponseGoogleAddress*: essa *model* foi criada para representar os dados de retorno do serviço do Google de consultar informações de um endereço em função da latitude e longitude.

- *UserRequest*: essa *model* foi criada para representar os dados enviados pelo aplicativo Aqua Mobile ao efetuar o *login* e cadastro.

5.2.2.4 Controllers

Foram criadas na Aqua API 6 *controllers* para expor métodos que possam consultar, editar ou excluir dados de cada uma das tabelas da base de dados. As *controllers* criadas são:

- *AuthController*: essa *controller* é a única que não possui autenticação por JWT porque ela é a responsável por realizar a validação de *login* (HTTP POST), cadastro (HTTP POST) e criação do token.

- *ColetasController*: essa *controller* possui autenticação por JWT. Ela é responsável por fornecer os métodos de acesso a base de dados para a tabela *coletas*. Os métodos desse end-point são listar coletas (HTTP GET), buscar uma coleta (HTTP GET), alterar uma coleta (HTTP PUT), criar uma coleta (HTTP POST) e deletar uma coleta (HTTP DELETE). O end-point criar uma coleta (HTTP POST) possui uma validação extra necessária para a regra de negócio. Ao criar uma nova coleta é verificado se o local da coleta já existe na base de dados. Caso o local não exista, então é criado um novo local. Após isso o local é atribuído a coleta. O motivo dessa regra será melhor explicado na seção 5.3.5.

- *LocalsController*: essa *controller* possui autenticação por JWT. Ela é responsável por fornecer os métodos de acesso a base de dados para a tabela *locais*. Os métodos desse end-point são listar locais (HTTP GET), buscar um local (HTTP GET), alterar um local (HTTP PUT), criar um local (HTTP POST) e deletar um local (HTTP DELETE).

- *ParametrosController*: essa *controller* possui autenticação por JWT. Ela é responsável por fornecer os métodos de acesso a base de dados para a tabela *parametros*. Os métodos desse end-point são listar parâmetros (HTTP GET), buscar um parâmetro (HTTP GET), alterar um parâmetro (HTTP PUT), criar um parâmetro (HTTP POST) e deletar um parâmetros (HTTP DELETE).

- *UsuariosController*: essa *controller* possui autenticação por JWT. Ela é responsável por fornecer os métodos de acesso a base de dados para a tabela *usuarios*. Os métodos desse end-point são listar usuarios (HTTP GET), buscar um usuario (HTTP GET), alterar um usuario (HTTP PUT), criar um usuarios (HTTP POST) e deletar um usuario (HTTP DELETE).

- *ValorReferenciaController*: essa *controller* possui autenticação por JWT. Ela é responsável por fornecer os métodos de acesso a base de dados para a tabela *valores_referencia*. Os métodos desse end-point são listar valores de referência (HTTP GET), buscar um valor de referência (HTTP GET), alterar um valor de referência (HTTP PUT), criar um valor de referência (HTTP POST) e deletar um valor de referência (HTTP DELETE).

Atualmente no Aqua Mobile só estão sendo chamadas as *controllers*: AuthController, ColetasController, LocalsController e ParametrosController. Porém todas as demais já foram criadas para requisitos futuros.

5.2.2.5 Service

Durante o desenvolvimento do *back-end* foi necessário criar uma classe de serviço para consultar a API do Google e obter as informações de um endereço. A escolha por criação de uma *service* foi pensando na reutilização de código. A *service* foi criada utilizando injeção de dependência para evitar o acoplamento e poder utilizar a interface ao invés da classe concreta quando necessário.

5.3 Telas do Aqua Mobile

O aplicativo Aqua Mobile é composto por 9 telas que serão explicadas em detalhes nas próximas seções. Todas as telas foram testadas durante e após o desenvolvimento em um simulador Android 11 versão da api 30 e no aparelho físico Samsung Galaxy J5 Prime com um Android 8 versão da api 25. Para conseguir uma resolução melhor da imagem das telas do aplicativo foram realizados prints das telas do simulador ao invés de fotos das telas do aparelho Samsung.

5.3.1 Tela de Login

A tela de *login* possui a funcionalidade de realizar o *login* e cadastro do usuário. Ela realiza a lógica de trocar de *login* para cadastro através do clique do usuário no botão com a descrição 'DESEJA REGISTRAR?'. O fluxo para retornar ao *login* é através do clique do usuário no botão com a descrição 'JÁ POSSUI CONTA?'. Dessa maneira foi

possível manter as duas funcionalidades em uma única tela. A única diferença entre o *login* e o cadastro está na quantidade de campos em tela e na chamada de serviço diferente de ambos à Web API Aqua API. A tela de *login* pode ser visualizada na figura 5.11a e possui dois campos:

- *E-mail*: esse campo é responsável por receber o e-mail do usuário e realizar uma validação. A validação realizada é feita para verificar se o e-mail informado não está vazio e se contém o caracter '@'. Caso não passe na validação, então é apresentado ao usuário uma mensagem de erro 'Informe um e-mail válido.'.

- *Senha*: esse campo é responsável por receber a senha do usuário e realizar uma validação. A validação realizada é feita para verificar se a senha informada não está vazia e contém mais de 6 caracteres. Caso não passe na validação, então é apresentado ao usuário uma mensagem de erro 'Informe uma senha válida. Senha deve possuir no mínimo 6 caracteres.'.

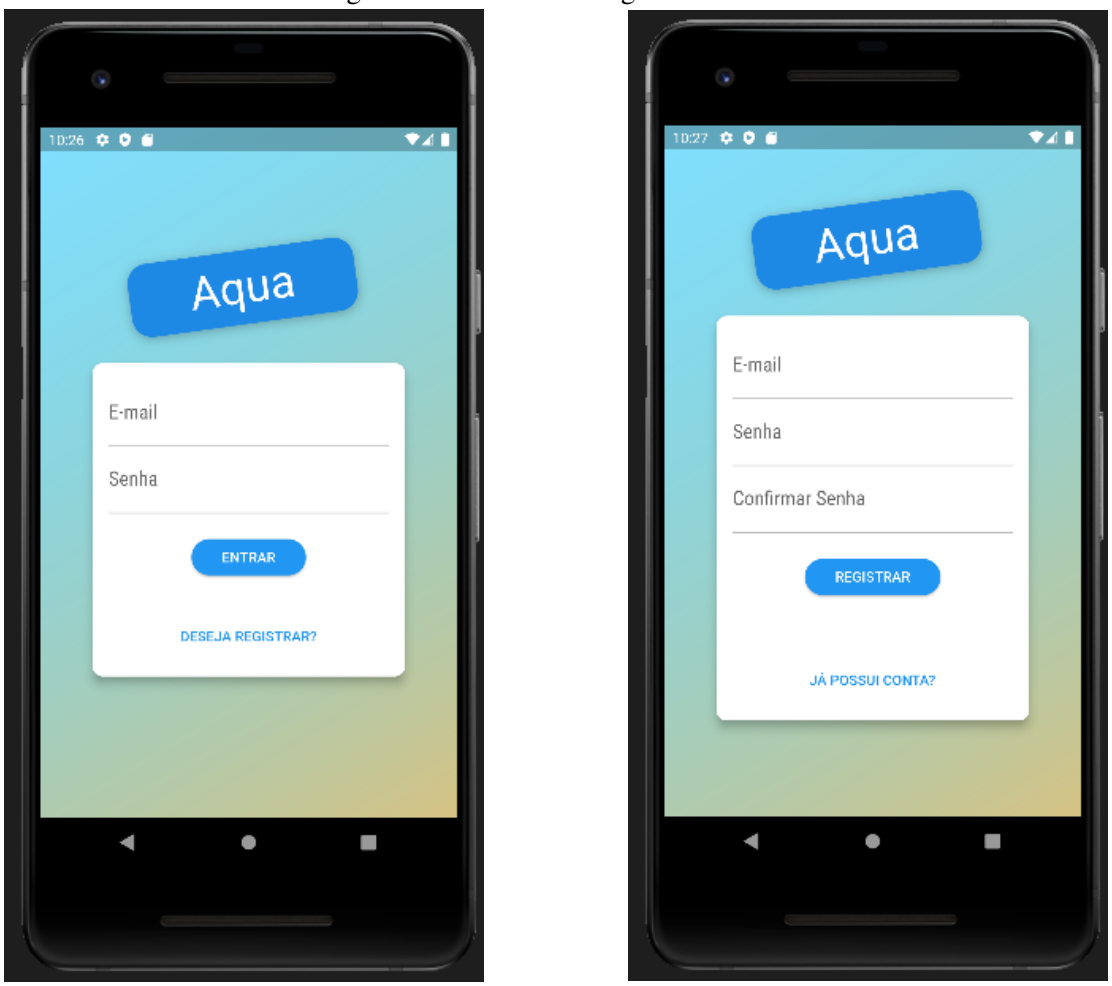
O *login* e cadastro realizam solicitações HTTP POST à Web API Aqua API controller Auth, porém para os métodos *login* e signUp respectivamente. Outra mudança em relação às duas telas é o botão 'ENTRAR' que altera sua descrição e ação para 'REGISTRAR' quando no modo de cadastro. A tela de cadastro está demonstrada na figura 5.11b e possui um campo a mais do que a tela de *login*. O campo extra é:

- *Confirmar Senha*: esse campo é responsável por receber a confirmação da senha do usuário e realizar uma validação. A validação realizada é feita para comparar se a senha de confirmação é igual a senha informada no campo anterior. Caso não passe na validação, então é apresentado ao usuário uma mensagem de erro 'Senhas informadas não conferem.'.

5.3.2 Tela de Listagem de Locais

A tela de listagem de locais realiza uma requisição HTTP GET na Web API Aqua API e busca todos os locais cadastrados na base de dados. Na lista apresentada no dispositivo, cada item é composto por uma imagem do local e ao lado o título e abaixo o subtítulo. O título é composto pela identificação do local concatenado com o município. O subtítulo é o endereço do local de coleta. A figura 5.12 apresenta a tela de listagem de locais. Além da lista citada acima, a tela apresenta no canto superior direito um botão

Figura 5.11 – Telas de Login e Cadastro



(a) Tela de Login

(b) Tela de Cadastro

Fonte: O Autor

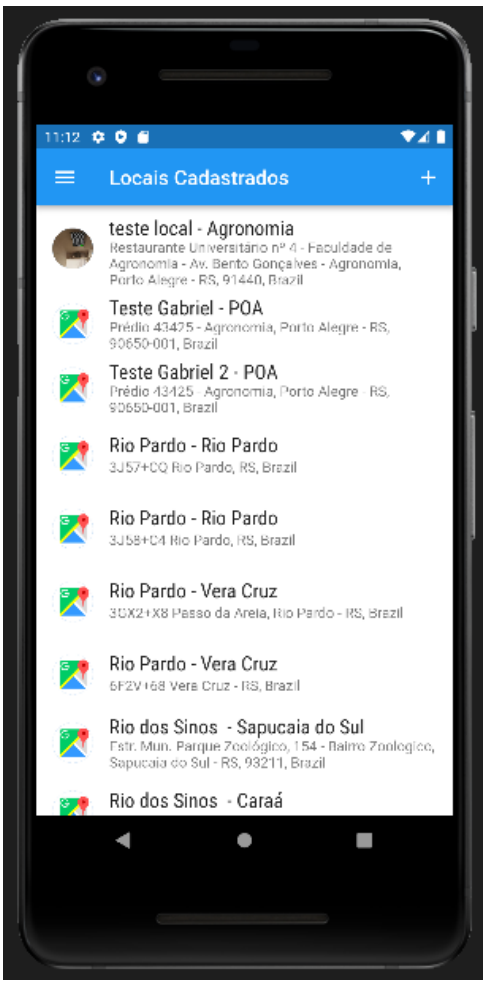
com o sinal '+' que permite adicionar um novo local. Ao clicar nesse botão o usuário é direcionado para a tela de criação de um novo local.

Os locais que foram cadastrados antes da criação do aplicativo Aqua Mobile não possuiam nenhuma imagem cadastrada, então foi utilizado nesse caso a imagem default do Google Maps conforme pode ser visto na 5.12. Caso não exista nehum local cadastrado, será apresentado nessa tela a mensagem 'Nenhum local cadastrado!'.

5.3.3 Tela de Cadastro de um Local

A tela de cadastro de um novo local é um formulário com os campos necessários para inserir um novo registro na base de dados. Diferentemente do sistema Aqua Web ela possibilita ao usuário tirar uma foto do local e escolher a localização do local entre a posição atual ou selecionando uma posição através do mapa do Google. Essa tela possui

Figura 5.12 – Tela de Listagem de Locais.



Fonte: O Autor

no canto superior esquerdo um botão que permite ao usuário retornar para a tela anterior, que no caso seria a listagem de locais. Os campos da tela de cadastro de um novo local são:

- *Descrição*: esse campo é responsável por receber a descrição do local e realizar uma validação. A validação realizada é para garantir que a descrição informada não está vazia. Caso não passe na validação, então é apresentado ao usuário uma mensagem de erro 'Descrição obrigatória'.

- *Bacia Hidrográfica*: esse campo é responsável por receber a bacia hidrográfica do local e realizar uma validação. A validação realizada é para garantir que a bacia hidrográfica informada não está vazia. Caso não passe na validação, então é apresentado ao usuário uma mensagem de erro 'Bacia Hidrográfica obrigatória'.

- *Tirar Foto*: esse campo é responsável por receber a imagem do local. Após o usuá-

rio clicar nele é aberto a câmera do dispositivo para ser tirada uma foto. Após a imagem ser tirada, a mesma é apresentada no lado esquerdo do botão em um container.

- *Localização*: esse campo é responsável por receber a localização do local. O usuário pode escolher entre dois botões que possibilitam selecionar a posição atual do usuário ou abrir o mapa do Google e escolher uma posição no mapa. Após a escolha do usuário através de um dos dois botões, então a latitude e longitude são conhecidas e através delas podemos buscar as demais informações necessárias na API do Google Maps. Esse campo no aplicativo *mobile* tem uma grande vantagem em relação ao sistema Aqua Web, pois ele abstrai a necessidade do usuário digitar o município, a latitude e a longitude. Tornando a experiência do usuário melhor com menos informações para digitar. Depois da localização escolhida ela é apresentada em um mapa do Google no container superior aos dois botões de escolha.

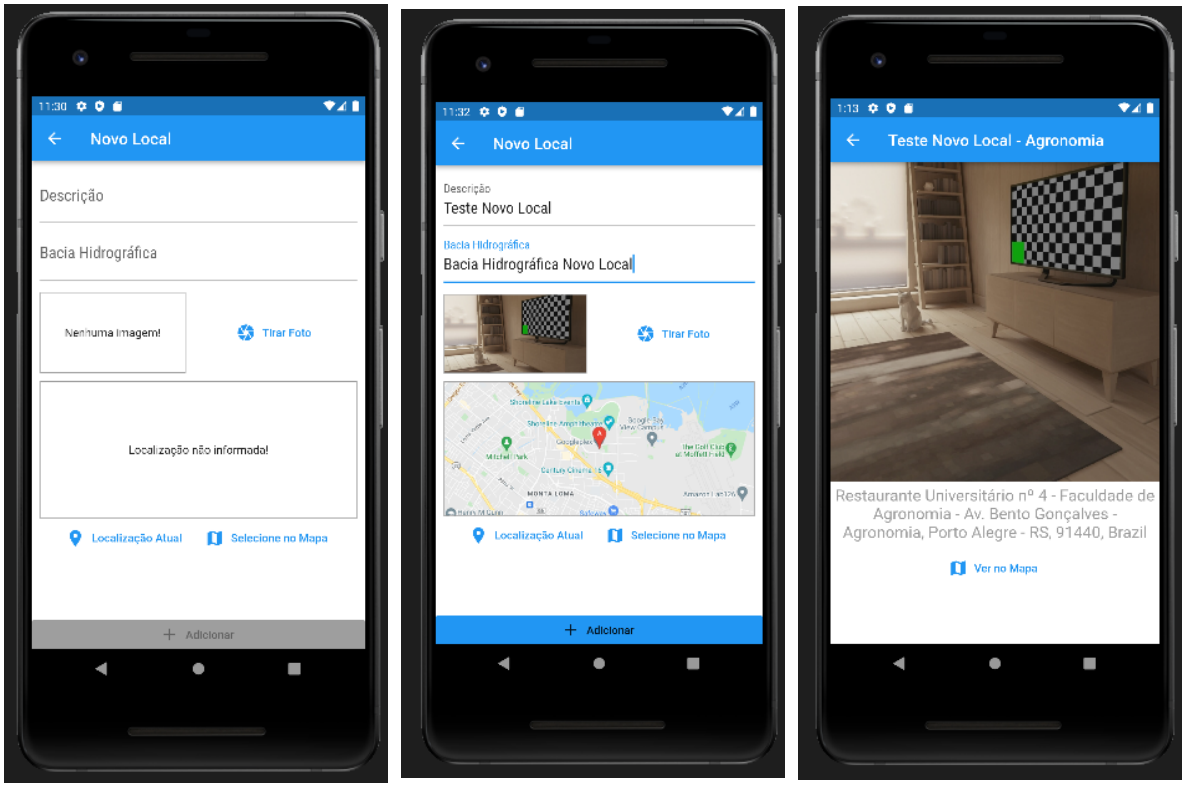
Para exemplificar a tela de cadastro de um novo local a figura 5.13a ilustra os campos vazios quando o usuário entrar na tela, e logo após a figura 5.13b mostra os campos preenchidos após a interação do usuário. Essa tela possui a validação de somente habilitar o botão '+ Adicionar' se todos os campos estiverem válidos, e for escolhido uma imagem e localização.

A tela para visualizar um local cadastrado permite ao usuário ver as informações de um local após ser selecionado na lista de locais. Ela apresenta a imagem cadastrada e logo abaixo a descrição do endereço e um botão com a descrição 'Ver no Mapa', o qual permite visualizar no Google Maps o endereço cadastrado. A figura 5.13c mostra a tela para visualizar um local.

5.3.4 Tela de Listagem de Coletas

A tela de listagem de coletas realiza uma requisição HTTP GET na Web API Aqua API e busca todas as coletas cadastradas na base de dados. Na lista apresentada no dispositivo, cada item é composto por uma imagem da coleta e ao lado o título e abaixo o subtítulo. O título é composto pela identificação do local da coleta. O subtítulo é o endereço do local de coleta. A figura 5.14 apresenta a tela de listagem de coletas. Além da lista citada acima, a tela apresenta no canto superior direito um botão com o sinal '+' que permite adicionar uma nova coleta. Ao clicar nesse botão o usuário é direcionado

Figura 5.13 – Telas de Cadastro de Local



(a) Cadastro de um novo Local vazia.

(b) Cadastro de um novo Local preenchida

(c) Tela para visualizar um Local cadastrado

Fonte: O Autor

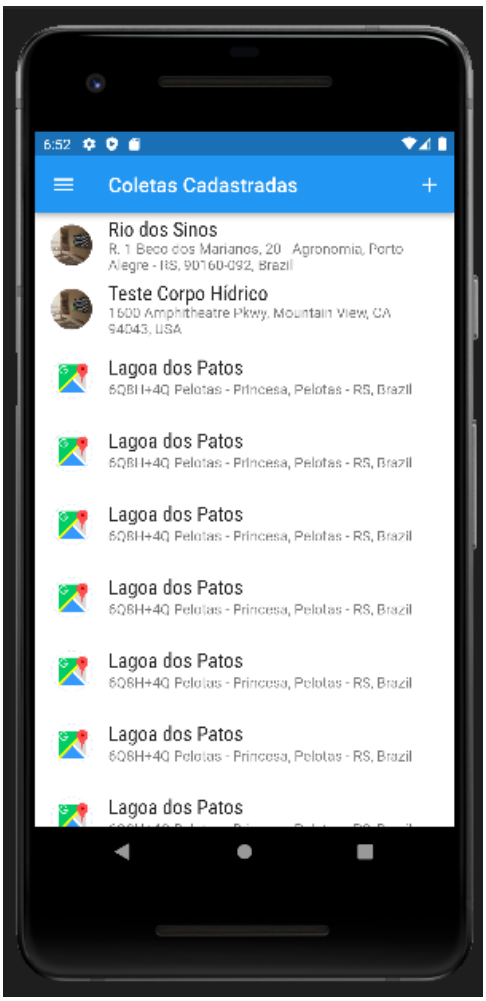
para a tela de criação de uma nova coleta.

As coletas que foram cadastradas antes da criação do aplicativo Aqua Mobile não possuiam nenhuma imagem cadastrada, então foi utilizado nesse caso a imagem default do Google Maps conforme pode ser visto na 5.14. Caso não exista nehuma coleta cadastrada, será apresentado nessa tela a mensagem 'Nenhuma coleta cadastrada!'.

5.3.5 Tela de Cadastro da Coleta

A tela de cadastro de uma coleta é um formulário com os campos necessários para inserir um novo registro na base de dados. Diferentemente do sistema Aqua Web ela possibilita ao usuário tirar uma foto da coleta e escolher a localização do local de coleta entre a posição atual ou selecionando uma posição através do mapa do Google. Essa tela possui no canto superior esquerdo um botão que permite ao usuário retornar para a tela anterior, que no caso seria a listagem de coletas. Os campos da tela de cadastro de uma nova coleta são:

Figura 5.14 – Tela de Listagem de Coletas.



Fonte: O Autor

- *Corpo Hídrico*: esse campo é responsável por receber o corpo hídrico e realizar uma validação. A validação realizada é para garantir que o corpo hídrico informado não está vazia. Caso não passe na validação, então é apresentado ao usuário uma mensagem de erro 'Corpo Hídrico obrigatório'.

- *Bacia Hidrográfica*: esse campo é responsável por receber a bacia hidrográfica do local e realizar uma validação. A validação realizada é para garantir que a bacia hidrográfica informada não está vazia. Caso não passe na validação, então é apresentado ao usuário uma mensagem de erro 'Bacia Hidrográfica obrigatória'.

- *Ponto de Referência*: esse campo é responsável por receber o ponto de referência da coleta.

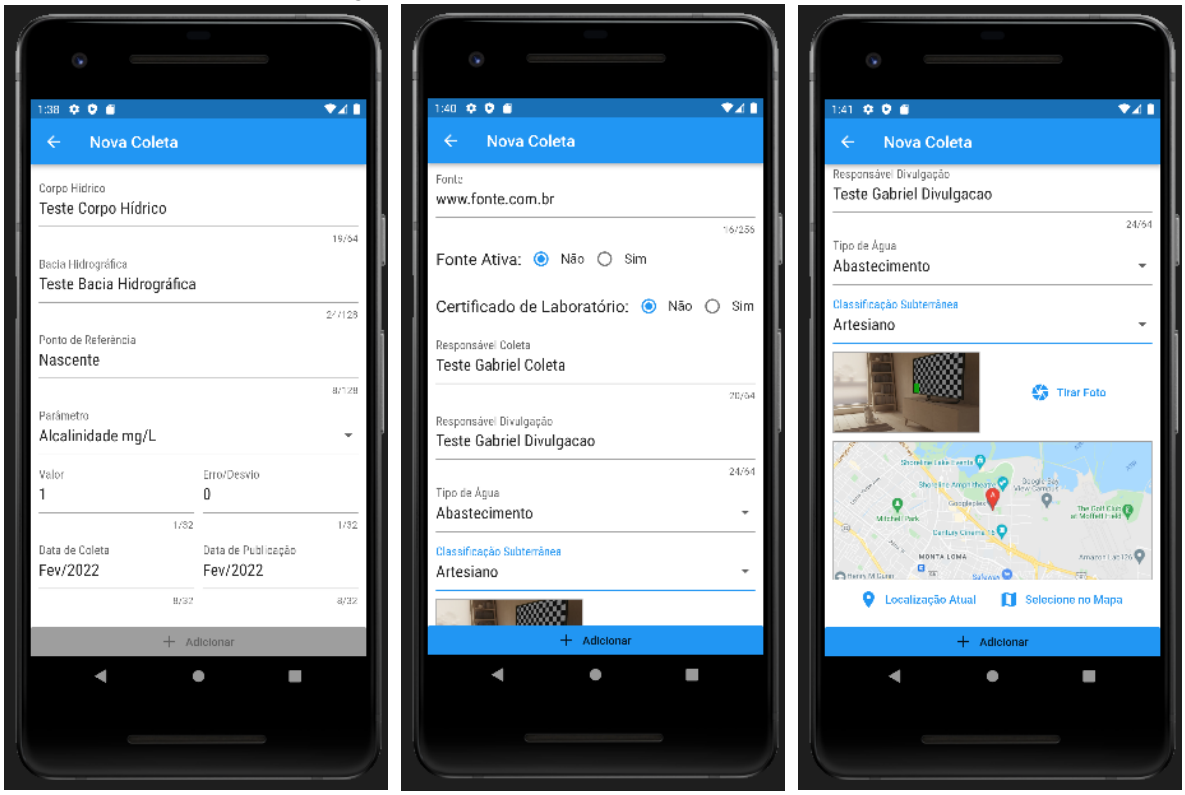
- *Parâmetro*: esse campo é responsável por receber o parâmetro da coleta e realizar

uma validação. A validação realizada é para garantir que o parâmetro da coleta não está vazio. Caso não passe na validação, então é apresentado ao usuário uma mensagem de erro 'Parâmetro obrigatório'. Nesse campo foi criado um DropdownButtonFormField para exibir as opções de parâmetros e evitar que o usuário insira uma valor incorreto.

- *Valor*: esse campo é responsável por receber o valor da coleta.
- *Erro/Desvio*: esse campo é responsável por receber o erro/desvio da coleta.
- *Data de Coleta*: esse campo é responsável por receber a data de coleta. Ele manteve o mesmo padrão do formato que já estava na base de dados que é mês/ano.
- *Data de Publicação*: esse campo é responsável por receber a data de publicação. Ele manteve o mesmo padrão do formato que já estava na base de dados que é mês/ano.
- *Fonte*: esse campo é responsável por receber a fonte da coleta.
- *Fonte Ativa*: esse campo é responsável por receber o valor se a fonte está ativa.
- *Certificado de Laboratório*: esse campo é responsável por receber o valor se o certificado é de laboratório.
- *Responsável Coleta*: esse campo é responsável por receber o responsável da coleta.
- *Responsável Divulgação*: esse campo é responsável por receber o responsável da divulgação.
- *Tipo de Água*: esse campo é responsável por receber o tipo de água.
- *Classificação Subterrânea*: esse campo é responsável por receber a classificação subterrânea.
- *Tirar Foto*: esse campo é responsável por receber a imagem da coleta. Foi reutilizado o mesmo componente da imagem do local.
- *Localização*: esse campo é responsável por receber a localização da coleta. Foi reutilizado o mesmo componente da localização do local.

Para exemplificar a tela de cadastro de uma nova coleta as figuras 5.15a, 5.15b e 5.15c mostram os campos preenchidos após a interação do usuário. Semelhante a tela de cadastro de um local, essa tela possui a validação de somente habilitar o botão '+ Adicionar' se todos os campos estiverem válidos, e for escolhido uma imagem e localização.

Figura 5.15 – Telas de Cadastro da Coleta



(a) Campos da coleta. (b) Demais campos da coleta. (c) Demais campos da coleta.

Fonte: O Autor

5.3.6 Tela para Visualizar uma Coleta

Essa tela permite ao usuário visualizar as informações de uma coleta após ser selecionada na lista de coletas. Ela apresenta a imagem cadastrada na coleta e logo abaixo a descrição do endereço e um botão com a descrição 'Ver no Mapa', o qual permite visualizar no Google Maps o endereço cadastrado. A figura 5.16 mostra a tela para visualizar uma coleta.

Figura 5.16 – Tela para visualizar uma Coleta.

Fonte: O Autor

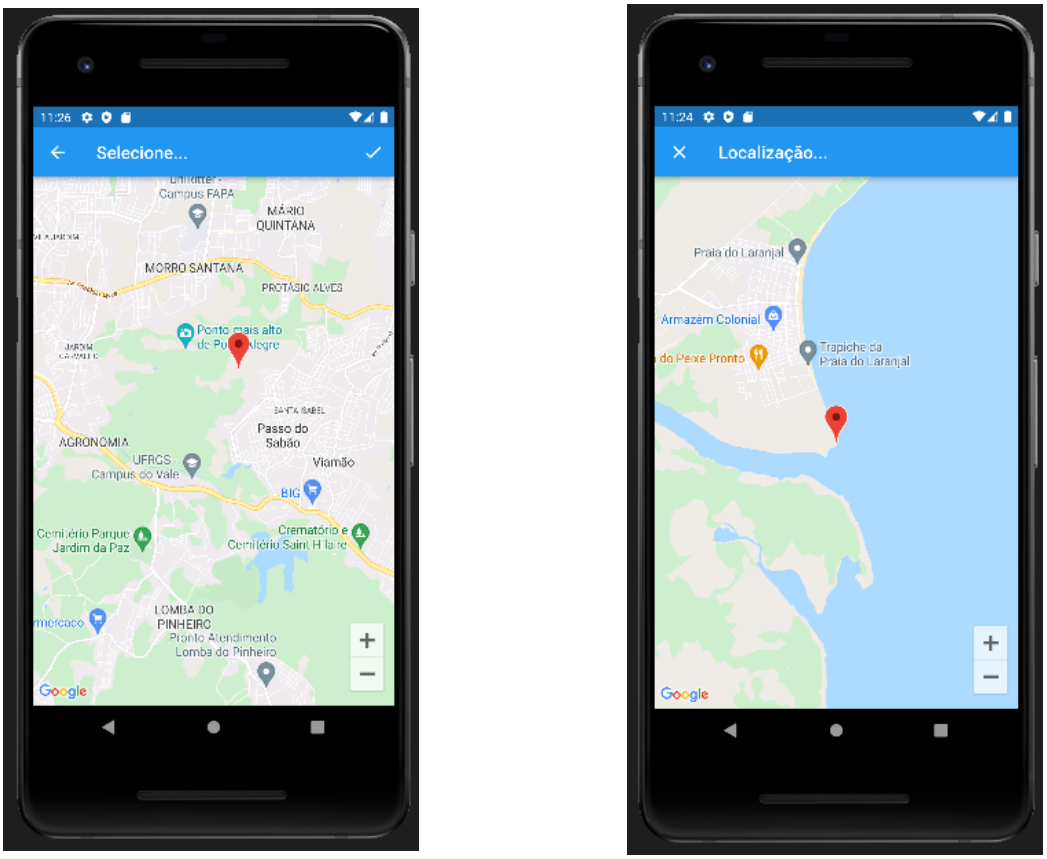
5.3.7 Tela de Visualizar ou Selecionar no Mapa

Essa tela foi desenvolvida para permitir realizar duas funcionalidades, a primeira é o usuário escolher uma posição no Google Maps, e a segunda é visualizar uma posição no Google Maps em função da latitude e longitude informadas no seu construtor. A ideia no seu desenvolvimento foi a reutilização de código e com isso aproveitar a chamada ao Google Maps. É realizada uma lógica que valida se a tela está em modo de escolha ou visualização de uma posição. Caso a tela esteja em escolha de uma posição então é habilitado o botão de check que fica no canto superior direito, e com isso é retornado a tela de origem da chamada a latitude e longitude escolhida.

A figura 5.17a ilustra o modo de escolha de uma posição e somente habilita o botão de check após o usuário clicar em uma posição do mapa. Esse modo de escolha é originado de alguma tela que possui o botão 'Selecione no Mapa', normalmente uma

tela de cadastro como pode ser visto na figura 5.15c por exemplo. A segunda figura 5.17b ilustra o modo de visualização de uma posição informada. Esse modo de escolha é originado de alguma tela que possui o botão 'Ver no Mapa', normalmente uma tela de visualização de um item cadastrado como pode ser visto na figura 5.16 por exemplo. No modo de escolha de posição essa tela foi desenvolvida para abrir com o mapa centralizado na região do Instituto de Informática da UFRGS.

Figura 5.17 – Telas de Visualizar ou Selecionar no Mapa



(a) Selecionar no mapa

(b) Visualizar no mapa

Fonte: O Autor

5.3.8 Menu Lateral

Foi utilizado o componente do Google Material Drawer para a criação do menu lateral. Na primeira versão do Aqua Mobile somente foram criados 3 itens no menu conforme pode ser visto na figura 5.18. Os itens do menu são:

- *Lista de Locais*: esse item direciona o aplicativo para a tela de listagem de locais.
- *Lista de Coletas*: esse item direciona o aplicativo para a tela de listagem de coletas.

- *Sair*: esse item realiza o logout do usuário do aplicativo e após redireciona para a tela de *login*.

Figura 5.18 – Menu Lateral.

Fonte: O Autor

6 CONCLUSÃO

Este trabalho foi realizado para suprir a necessidade de aplicativos móveis relacionados com recursos hídricos. Foi criado com este trabalho o aplicativo denominado Aqua Mobile, e seu desenvolvimento se deu para a plataforma Android e utilização do *framework* Flutter. Esse aplicativo facilitou o acesso aos dados já coletados e permitiu cadastrar novas coletas na palma da mão do usuário.

O cadastro de novos locais e coletas através do Aqua Mobile possibilita aos usuários a utilização das funcionalidades do *GPS* e da câmera do celular. Outra característica do novo software é a utilização do Google Maps para visualizar e escolher locais no mapa, com isso os dados coletados passaram a conter novas informações antes inexistentes, a descrição do endereço e imagem do local e da coleta. A partir dessas novas informações será possível enriquecer novas pesquisas com mais informações de amostras coletadas.

Durante o desenvolvimento do aplicativo foi possível descobrir uma solução para enviar a imagem obtida da câmera do celular no formato Image do Flutter, e salvá-la na base de dados MySQL no formato array de bytes. Também foi possível encontrar uma solução no cadastro de uma nova coleta, que não exige que o local já esteja cadastrado na base de dados. Foi criado uma lógica durante o cadastro da coleta, que ao verificar que o local ainda não possui cadastro, então cadastra automaticamente. Através do desenvolvido do aplicativo Aqua Mobile é possível concluir que os objetivos deste trabalho foram atingidos, com a criação de um aplicativo *mobile* para acessar e cadastrar dados de qualidade das águas.

6.1 Trabalhos Futuros

O aplicativo Aqua Mobile pode ser incrementado, com as seguintes funcionalidades:

- *Trabalhar off-line*: mesmo sem sinal de Internet o aplicativo irá permitir cadastrar novos locais e coletas na memória do celular. Assim que o aplicativo perceber que o sinal de Internet esteja disponível, então irá realizar a persistência dos dados na base de dados.
- *Editar e excluir*: será implementado a possibilidade de editar e excluir registros por usuários com permissão.

- *Imagem de fundo*: permitir que seja configurável a escolha da imagem de fundo da tela de *login*/cadastro.
- *Campo observação*: permitir a inclusão de um campo observação no cadastro de um local e coleta.
- *Copiar registros*: permitir criar novos registros a partir da cópia de registros já existentes.
- *Latitude e Longitude manuais*: permitir a criação de dois novos campos para o usuário poder escolher se quer efetuar a entrada da latitude e longitude de forma manual.
- *Análise da qualidade da rede*: em função da qualidade da rede será alterado a quantidade de registros trafegados.
- *Mapas gratuitos*: alterar os mapas para uma solução gratuita.
- *Possibilidade de diversas fotos para Locais e Coletas*: permitir ao usuário cadastrar mais que uma foto para Local e Coleta. Para esse desenvolvimento será necessário criar uma tabela para fotos na base de dados.

REFERÊNCIAS

ANDRADE, K. **Introdução a linguagem de programação Dart | by Kleber Andrade | Flutter—Comunidade BR | Medium**. 2022. <https://medium.com/flutter-comunidade-br/introdu%C3%A7%C3%A3o-a-linguagem-de-programa%C3%A7%C3%A3o-dart-b098e4e2a41e>. [Online; Acesso em: Fevereiro 2022].

AZURE, M. **Cloud Computing Services | Microsoft Azure**. 2022. <https://azure.microsoft.com>. [Online; Acesso em: Fevereiro 2022].

BARROS, F. G. N.; AMIN, M. M. Água: um bem econômico de valor para o brasil e o mundo. **Revista brasileira de gestão e desenvolvimento regional**, v. 4, n. 1, 2008.

BOCK, F. G. Sistema aqua para disponibilização de dados da qualidade das águas. 2021.

CANTELLE, T. D. Água e aplicativos para dispositivos móveis.

CLOUD, G. **Serviços de computação em nuvem | Google Cloud**. 2022. <https://cloud.google.com/>. [Online; Acesso em: Fevereiro 2022].

DAGNE, L. Flutter for cross-platform app and sdk development. 2019.

DART, G. **Language tour | Dart**. 2022. <https://dart.dev/guides/language/language-tour>. [Online; Acesso em: Fevereiro 2022].

DART, G. **Sound null safety**. 2022. <https://dart.dev/null-safety>. [Online; Acesso em: Fevereiro 2022].

DART, G. **The Dart type system | Dart**. 2022. <https://dart.dev/guides/language/type-system>. [Online; Acesso em: Fevereiro 2022].

DEVELOPERS, G. **Guia de início rápido do SDK do Maps para Android | Google Developers**. 2022. <https://developers.google.com/maps/documentation/android-sdk/start?hl=pt_BR>. [Online; Acesso em: Fevereiro 2022].

DEVELOPERS, G. **Overview | Maps Static API | Google Developers**. 2022. <https://developers.google.com/maps/documentation/maps-static/overview?hl=pt_BR>. [Online; Acesso em: Fevereiro 2022].

DEVMEDIA. **Como o JWT (JSON Web Token) funciona?** 2022. <https://www.devmedia.com.br/como-o-jwt-funciona/40265>. [Online; Acesso em: Março 2022].

FLUTTER. **Introduction to widgets**. 2022. <https://docs.flutter.dev/development/ui/widgets-intro>. [Online; Acesso em: Fevereiro 2022].

FREITAS, M. B.; FREITAS, C. M. d. A vigilância da qualidade da água para consumo humano: desafios e perspectivas para o sistema único de saúde. **Ciência & Saúde Coletiva**, SciELO Brasil, v. 10, n. 4, p. 993–1004, 2005.

GOOGLE. **The official package repository for Dart and Flutter apps**. 2022. <https://pub.dev/>. [Online; Acesso em: Fevereiro 2022].

JONES, M.; BRADLEY, J.; SAKIMURA, N. **Json web token (jwt)**. [S.l.], 2015.

KINGHOST. **A Melhor Hospedagem de Site | KingHost**. 2022. <https://king.host/>. [Online; Acesso em: Fevereiro 2022].

MATERIAL, G. **Homepage - Material Design**. 2022. <https://material.io/>. [Online; Acesso em: Março 2022].

MICROSOFT. **.NET (e .NET Core) – introdução e visão geral | Microsoft Docs**. 2022. <https://docs.microsoft.com/pt-br/dotnet/core/introduction>. [Online; Acesso em: Fevereiro 2022].

PACKAGE, D. **Http | Dart Package**. 2022. <https://pub.dev/packages/http>. [Online; Acesso em: Março 2022].

PACKAGE, D. **Intl | Dart Package**. 2022. <https://pub.dev/packages/intl>. [Online; Acesso em: Março 2022].

PACKAGE, F. **Google_maps_flutter | Flutter Package**. 2022. <https://pub.dev/packages/google_maps_flutter>. [Online; Acesso em: Março 2022].

PACKAGE, F. **Jwt_decoder | Dart Package**. 2022. <https://pub.dev/packages/jwt_decoder>. [Online; Acesso em: Março 2022].

PACKAGE, F. **Location | Flutter Package**. 2022. <https://pub.dev/packages/location>. [Online; Acesso em: Março 2022].

PACKAGE, F. **Provider | Flutter Package**. 2022. <https://pub.dev/packages/provider>. [Online; Acesso em: Março 2022].

PACKAGE, F. **Sqflite | Flutter Package**. 2022. <https://pub.dev/packages/sqflite>. [Online; Acesso em: Março 2022].

SWAGGER. **Swagger - About**. 2022. <https://swagger.io/about/>. [Online; Acesso em: Fevereiro 2022].

WU, W. React native vs flutter, cross-platforms mobile application frameworks. Metropolia Ammattikorkeakoulu, 2018.

ZERWES, C. M. et al. Análise da qualidade da água de poços artesianos do município de imigrante, vale do taquari/rs. **Ciência e Natura**, Universidade Federal de Santa Maria, v. 37, n. 3, p. 651–663, 2015.