****

**PROETO DE AUTOMAÇÃO**

**API REST**

**VICTOR GUERRA**

Sumário

[1. Introdução 3](#_Toc531795991)

[2. Firebase 3](#_Toc531795992)

[Authentication 3](#_Toc531795993)

[Realtime Database 4](#_Toc531795994)

[3. A API do Realtime Database 6](#_Toc531795995)

[Obter Token de Acesso 6](#_Toc531795996)

[Parâmetros de Query 7](#_Toc531795997)

[4. Estrutura do Projeto 8](#_Toc531795998)

[Estrutura de Arquivos 8](#_Toc531795999)

[Estrutura de Requisições 8](#_Toc531796000)

[Estrutura de Testes 9](#_Toc531796001)

# Introdução

Este documento tem como objetivo apresentar o Projeto de Automação de API Rest com SoapUI, tratando a estrutura e composição do projeto, a API sob testes e seus endpoints.

Para o projeto, foi utilizada a API disponibilizada pela plataforma Firebase para gerenciamento de um banco de dados NoSQL. A seguir farei uma breve explicação do que é o Firebase e como funciona esta API.

# Firebase

O Firebase é uma plataforma para desenvolvimento web e mobile que disponibiliza diversas ferramentas de beckend para autenticação, armazenamento, hospedagem entre outros.

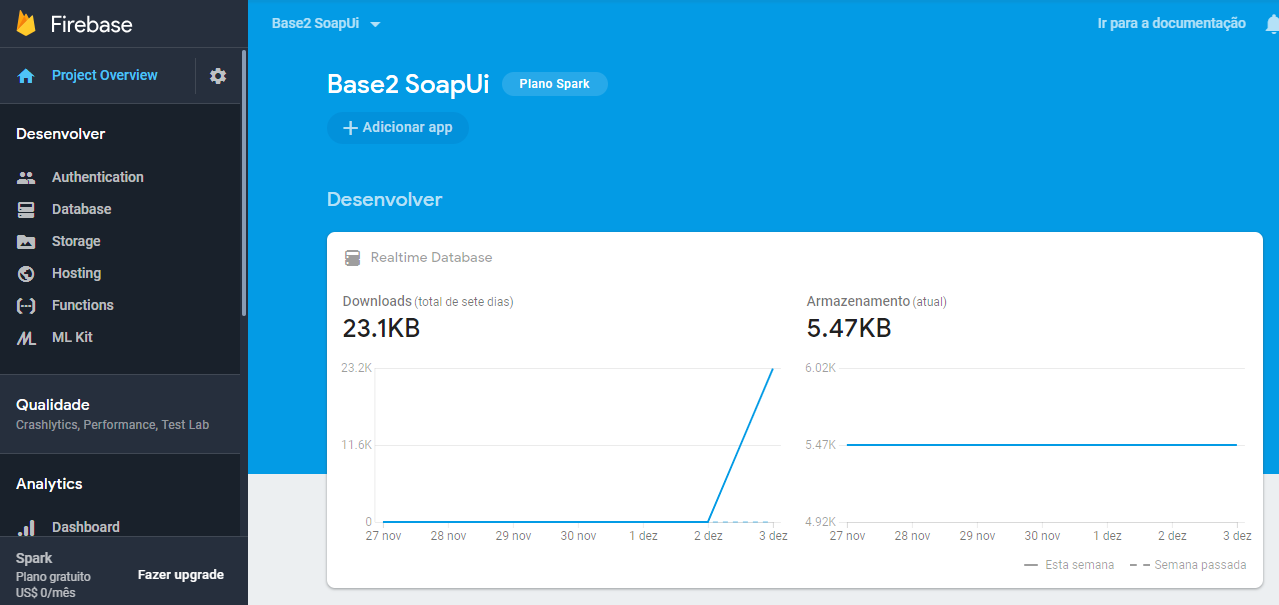


Figura 1 - Console do Firebase

As ferramentas utilizadas no projeto são Authentication para autenticação e Realtime Database para gerenciamento de dados.

## Authentication

Através da ferramenta Authentication é possível gerenciar os métodos de autenticação da aplicação. Há vários métodos de autenticação disponíveis, tais como emial e senha, Facebook, Google, Twitter, etc.

Para tornar a autenticação o mais simples possível, o método utilizado para o projeto foi email e senha. Para tal foram criados quatro usuários, cada um com um perfil de acesso, sendo eles:

* Administrador;
* Financeiro;
* Estoquista;
* Vendedor.

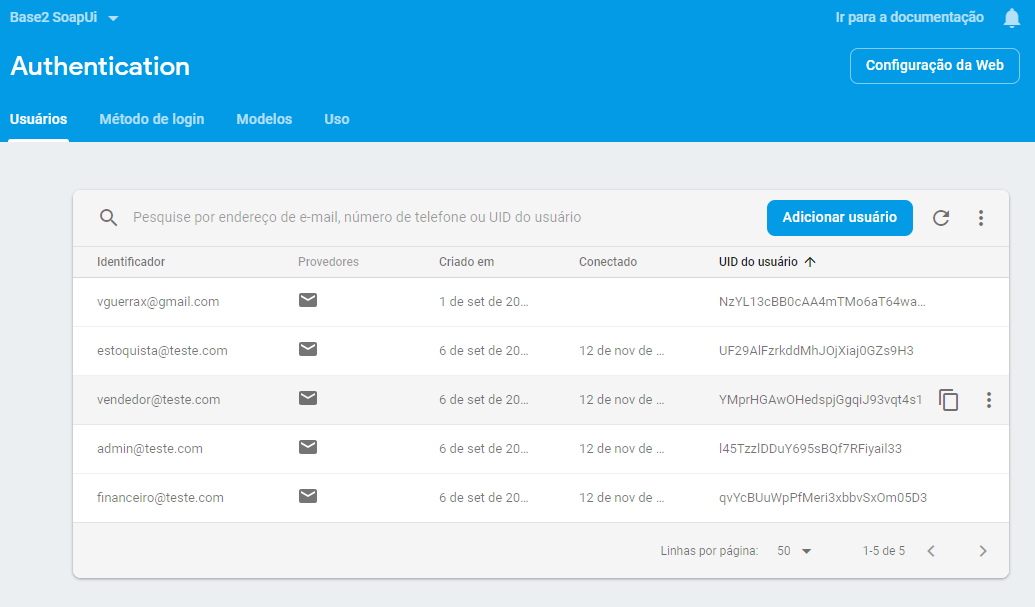


Figura 2 - Usuários criados para autenticação

## Realtime Database

O Realtime Database é a ferramente de banco de dados do Firebase. Através dele é disponibilizado um banco de dados NoSQL atualizado em tempo real na aplicação e com possibilidade de acesso off-line.

A ferramenta disponibiliza opções para estruturar a base, adicionando, alterando ou removendo dados manualmente, além de uma API rest para acesso aos dados. Os dados no Realtime Database são estruturados como em um JSON, sendo composto por vários nós.

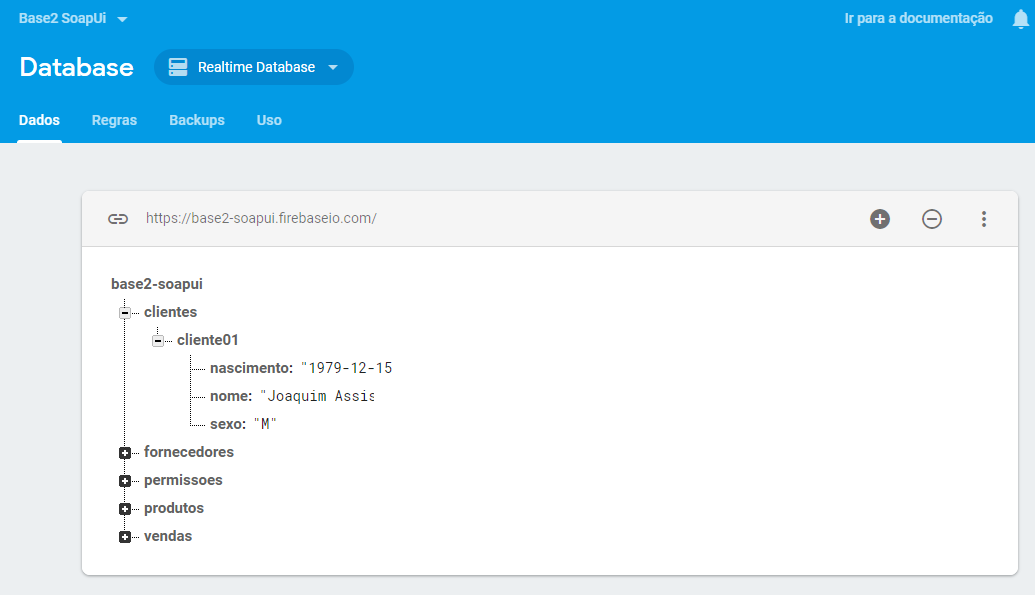


Figura 3 - Estrutura dos dados no Realtime Database

Outra opção disponilizada pelo Realtime Database são as rules, regras de acesso aos dados. As regras podem ser implementadas universalmente para todos os nós ou individualmente para cada nó e a sintaxe das regras é parecida com a do Javascript. São quatro tipos de regras:

* Read – para conceder permissão de leitura aos nós;
* Write – para conceder permissão de gravação aos nós;
* Validade – Para validação dos dados inseridos;
* IndexOn – Para definir os índices dos nós.

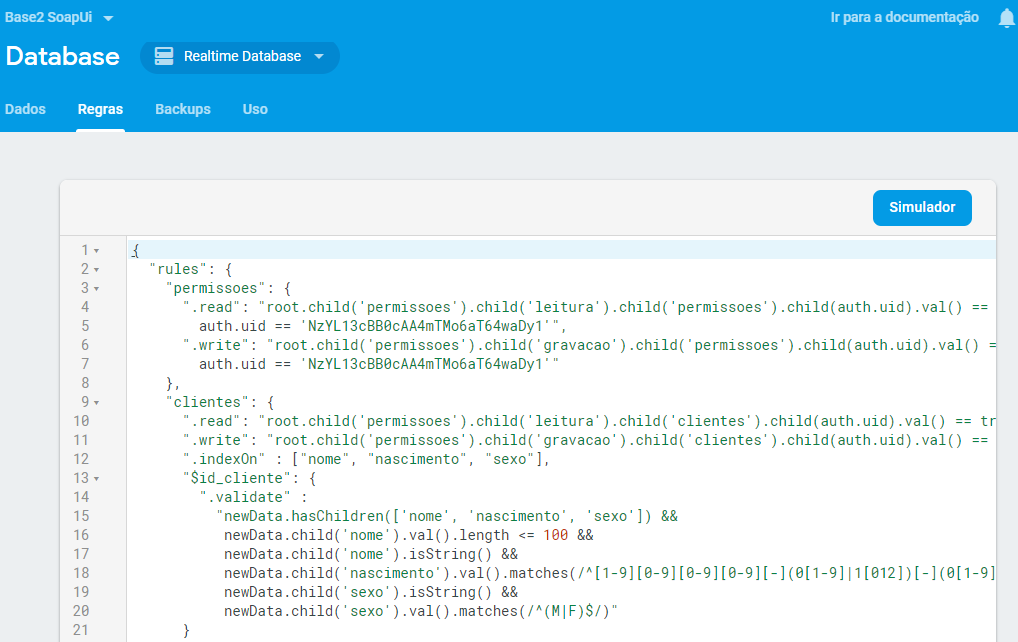


Figura 4 - Criação das regras de acesso aos dados

# A API do Realtime Database

Como citado anteriormente, o Realtime Database disponibiliza uma API para gerenciamento dos dados gravados. Esta api pode ser acessada através da URL https://<codigo\_do\_projeto>.firebaseio.com/<caminho\_do\_no>.json. Logo para acessar os dados de clientes basta realizar uma chamada ao endpoint <https://base2-sopaui.firebaseio.com/clientes.json>.

As requisições possíveis para a API são:

* GET – para realizar consulta aos dados;
* PUT – para gravar ou substituir os dados (se já existirem dados no caminho passado, serão substituídos);
* PATCH – para adicionar dados;
* POST – para gravar novos dados (sempre que chamada irá gerar uma nova chave única);
* DELETE – para deletar dados.

Caso existam regras de leitura/gravação nos nós do banco de dados, é necessário autenticar a requisição através do parâmetro *acess\_token.* Através deste parâmetro deve ser passado para a requisição o token de acesso do usuário responsável pelos dados.

## Obter Token de Acesso

Para obter o token de acesso de um usuário basta realizar uma chamada ao endpoint <https://www.googleapis.com/identitytoolkit/v3/relyingparty/verifyPassword> passando o parâmetro *key* com a chave do projeto no Firebase e o email e senha para autenticação no corpo da requisição, conforme exemplo a seguir:

<https://www.googleapis.com/identitytoolkit/v3/relyingparty/verifyPassword?key=AIzaSyDrFtY6FOFqFln_Bw917GnWQ3ylca-BVuA>

Body:

{

email : "teste@teste.com",

password : "teste123",

returnSecureToken : true

}

O retorno dessa requisição será parecido com o JSON a seguir, onde *idToken* é o token de acesso.

{

"kind": "identitytoolkit#VerifyPasswordResponse",

"localId": "YMprHGAwOHedspjGgqiJ93vqt4s1",

"email": "vendedor@teste.com",

"displayName": "",

"idToken": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjIzNTBiNWY2NDM0Zjc2Y2NiM2IxMTlmZGQ4OGQxMzhjOWFjNTVmY2UiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vYmFzZTItc29hcHVpIiwiYXVkIjoiYmFzZTItc29hcHVpIiwiYXV0aF90aW1lIjoxNTQzOTYzNzU4LCJ1c2VyX2lkIjoiWU1wckhHQXdPSGVkc3BqR2dxaUo5M3ZxdDRzMSIsInN1YiI6IllNcHJIR0F3T0hlZHNwakdncWlKOTN2cXQ0czEiLCJpYXQiOjE1NDM5NjM3NTgsImV4cCI6MTU0Mzk2NzM1OCwiZW1haWwiOiJ2ZW5kZWRvckB0ZXN0ZS5jb20iLCJlbWFpbF92ZXJpZmllZCI6ZmFsc2UsImZpcmViYXNlIjp7ImlkZW50aXRpZXMiOnsiZW1haWwiOlsidmVuZGVkb3JAdGVzdGUuY29tIl19LCJzaWduX2luX3Byb3ZpZGVyIjoicGFzc3dvcmQifX0.Nrxgcp4kVltlotl0Uvm8zhwuejARGX9fxgBX3QKreSyEPg\_e0h0N18fwL9FAL5IU3xi7pL24p8Qf-tkdrDqgvq9IOrujd9b2qiCw7TIO8AXApxixnzkjdZTkCiKFv7oweNAIrAb6PoMKmab983BVr9iPNAQn94LttF1t8ou7eznRZY7U\_iVK3DBmDEZFwOdTuKIkuu3uLowLwfVwL3MrIPrdgB0NuG8iKxL6l62\_rAgjOSJ-qxfqwFgLm9COhyOyTKdaRT-vqi7yR6wkFLCWy1ojd60heNma2EyTAh1D34SIKECXishqWhgx4-G1XXrVOjSwVKdFtTDnWyO9WLPe4g",

"registered": true,

"refreshToken": "AGK09ANVn187e9lrzKzYgIS\_qiimm1XxmCKVq1Z-4ScaJZv4WLRM5F1jGFdPqa7EoxtkLh8IBuB\_3cggX2nymaEwnQZSzsjbVZrgGXbc-o2LIBzGOPqwEMINaaR-v4rZ443nkHTVQiVoxL-0BHIRAVPM40TdUA19jQerlnLhDicz0abNelqt2IAhtunkAMObJsXcI-bOi8R0cOPRJLJFu1BZedZRO2pjoA",

"expiresIn": "3600"

}

## Parâmetros de Query

A API do Realtime Database aceita diversos parâmetros para filtrar e ordenar os dados retornados. Segue a lista dos parâmetros e como utiliza-los:

* orderBy – Utilizado para ordenação, passando a chave do nó que será utilizado para ordernar os dados ou “$key” para ordenar pela chave.

Exemplo: [https://base2-sopaui.firebaseio.com/clientes.json?orderBy=”nome](https://base2-sopaui.firebaseio.com/clientes.json?orderBy=)”;

* startAt e endAt – Utilizados combinados com o orderBy para limitar os dados retornados. O startAt indica o valor para o primeiro elemento e o endAt indica o valor para o último elemento da lista de retorno e a chave para comparação é definida no orderBy.

Exemplo: [https://base2-sopaui.firebaseio.com/clientes.json?orderBy=”nome”&startAt=”A”&endAt=”M](https://base2-sopaui.firebaseio.com/clientes.json?orderBy=)” (lista os clientes cujos nomes estão entre ‘A’ e ‘M’);

* limitToFirst – Utilizado para limitar a quantidade de elementos retornados. Retorna apenas os primeiros *n* elementos da lista, sendo *n* o valor passado no parâmetro.

Exemplo: https://base2-sopaui.firebaseio.com/clientes.json?orderBy=”nome”&limitToFirst=5 (retorna apenas os 5 primeiros clientes, ordenados por nome);

* limitToLast – Utilizado para limitar a quantidade de elementos retornados. Retorna apenas os últimos *n* elementos da lista, sendo *n* o valor passado no parâmetro.

Exemplo: https://base2-sopaui.firebaseio.com/clientes.json?orderBy=”nome”&limitToFirst=5 (retorna apenas os 5 últimos clientes, ordenados por nome);

* equalTo – Utilizado para filtrar os dados retornados. Filtra os dados para aqueles cujo valor da chave definida pelo orderBy é igual ao valor passado no parâmetro.

Exemplo: [https://base2-sopaui.firebaseio.com/clientes.json?orderBy=”nome”&equalTo=”João](https://base2-sopaui.firebaseio.com/clientes.json?orderBy=) da Silva” (retorna apenas clientes cujo nome seja “João da Silva”).

# Estrutura do Projeto

Serão apresentadas a seguir as estruturas de arquivos, de requisições e de testes do projeto.

## Estrutura de Arquivos

A estrutura de arquivos do projeto contém os seguintes elementos:

* DataSources – Pasta para armazenar todos os arquivos para implementação de data-driven e criação de massa de testes;
* Base2-SoapUi-soapui-project.xml – Arquivo do projeto;
* README.md – Arquivo de informações para GitHub.

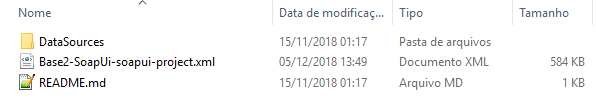


Figura 5 – Estrutura de Arquivos do Projeto

## Estrutura de Requisições

As requisições foram organizadas dentro de dois serviços, da seguinte forma:

* FirebaseEndPoints – concentra todas as requisições à URL <https://base2-soapui.firebaseio.com/> e suas variações. Possui chamadas com os métodos: GET, POST, PUT, DELETE, PATCH. As requisições deste serviço são:
  + GET – Definição de uma requisição GET simples;
  + POST – Definição de uma requisição POST simples;
  + PUT – Definição de uma requisição POST simples;
  + DELETE – Definição de uma requisição POST simples;
  + PATCH – Definição de uma requisição PATCH simples;
  + GET-FilterStartEndAt – Definição de uma requisição GET com os parâmetros *startAt* e *endAt*;
  + GET-FilterEqualTo – Definição de uma requisição GET com o parâmetro *equalTo*;
  + GET-LimitToFirst – Definição de uma requisição GET com o parâmetro *limitToFirst*;
  + GET-LimitToLast – Definição de uma requisição GET com o parâmetro *limitToLast*;
  + GET-OrderBy – Definição de uma requisição GET com o parâmetro *orderBy*;
* GoogleAPIs – contém a requisição de autenticação, uma requisição POST feita à URL <https://www.googleapis.com/identitytoolkit/v3/relyingparty/verifyPassword> conforme citado anteriormente.

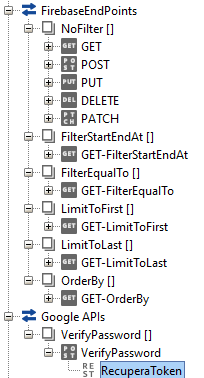


Figura 6 - Estrutura das Requisições

## Estrutura de Testes

Os testes foram organizados em quatro suítes de teste, uma para cada nó tradado pelos testes, sendo estas:

* Clientes;
* Produtos;
* Fornecedores;
* Vendas.

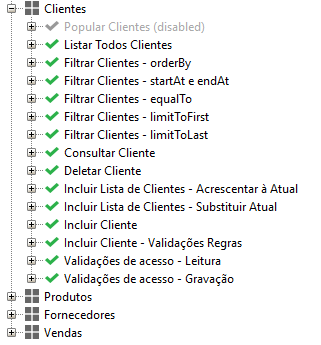


Figura 7- Estrutura dos Testes