****

**PROJETO DE AUTOMAÇÃO**

**API REST**

**VICTOR GUERRA**

Sumário

[**Introdução**](#_gjdgxs) **3**

[**Firebase**](#_30j0zll) **3**

[Authentication](#_1fob9te) 3

[Realtime Database](#_3znysh7) 4

[**A API do Realtime Database**](#_2et92p0) **6**

[Obter Token de Acesso](#_tyjcwt) 6

[Parâmetros de Query](#_3dy6vkm) 7

[**Estrutura do Projeto**](#_1t3h5sf) **8**

[Estrutura de Arquivos](#_4d34og8) 8

[Estrutura de Requisições](#_2s8eyo1) 9

[Estrutura de Testes](#_3rdcrjn) 10

[**Testes x Regras de Negócio**](#_jdetvyp76vee) **11**

[Clientes](#_arws0lqohsqg) 11

[Produtos](#_7c6v5le4cip7) 14

[Fornecedores](#_mlhyxhgs7wqs) 14

[Vendas](#_turjvu83burx) 14

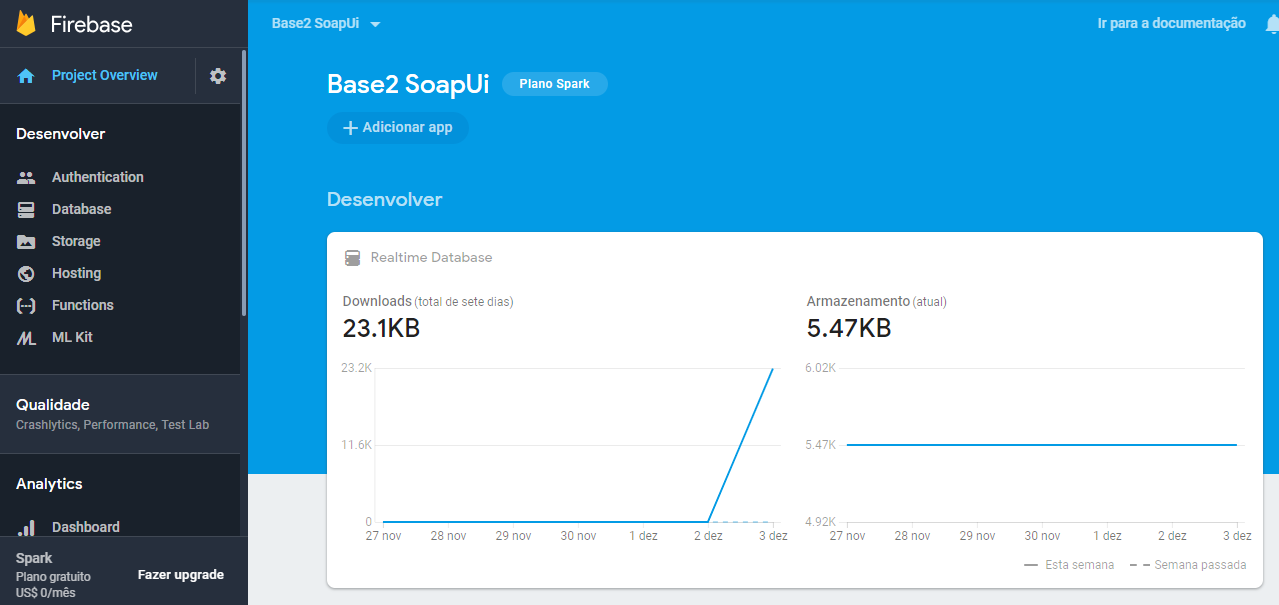
# Introdução

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 back-end para autenticação, armazenamento, hospedagem entre outros.



*Figura 1 - Console do Firebase*

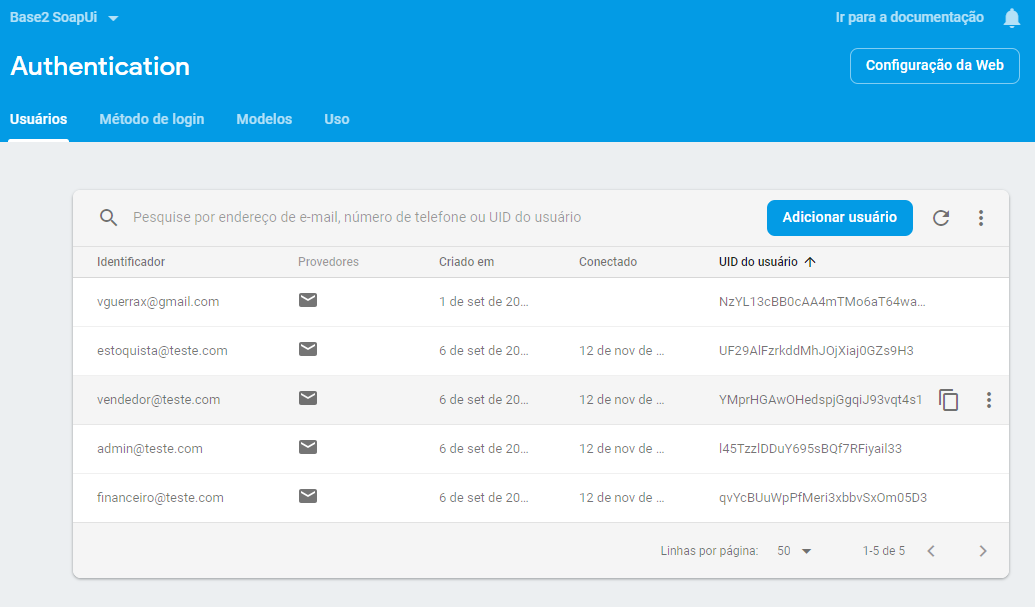
As ferramentas utilizadas no projeto são Authentication para autenticação e Realtime Database para gerenciamento de dados.

## Authentication

Através da ferramenta Authentication é possível gerenciar os métodos de autenticação da aplicação. Há vários métodos de autenticação disponíveis, tais como email 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.

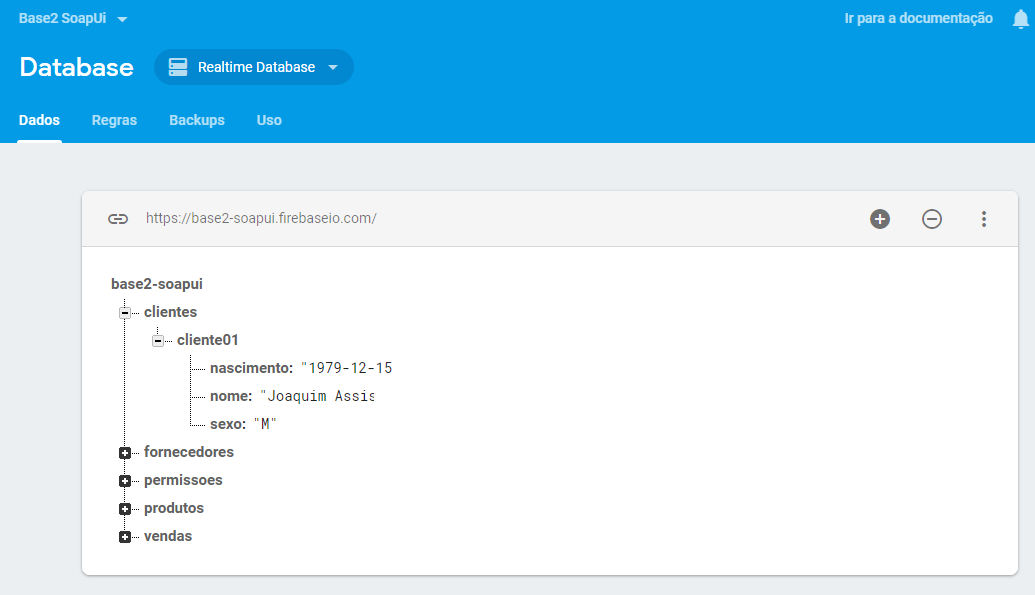


*Figura 2 - Usuários criados para autenticação*

## Realtime Database

O Realtime Database é a ferramenta 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.

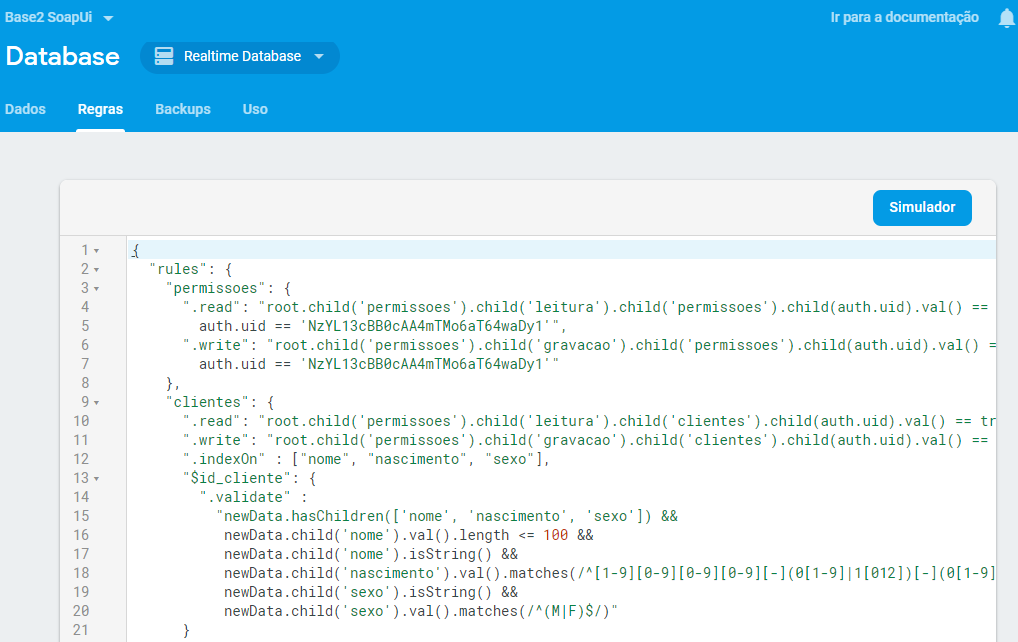
A ferramenta disponibiliza opções para estruturar a base, adicionando, alterando ou removendo dados manualmente, além de uma API rest para acesso aos dados. Os dados no Realtime Database são estruturados como em um JSON, sendo composto por vários nós.



*Figura 3 - Estrutura dos dados no Realtime Database*

Outra opção disponibilizada 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 utilizá-los:

* **orderBy**: Utilizado para ordenação, passando a chave do nó que será utilizado para ordenar 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”).

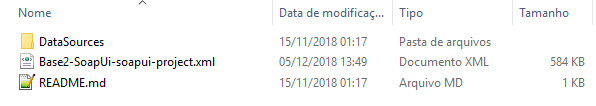
# Estrutura do Projeto

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. Contém os seguintes arquivos:
  + Clientes\_DataSource.xlsx;
  + Fornecedores\_DataSource.xlsx;
  + Produtos\_DataSource.xlsx;
  + Vendas\_DataSource.xlsx;
  + clientes.json;
  + fornecedores.json;
  + permissoes.json;
  + produtos.json.
  + vendas.json;
* **Base2-SoapUi-soapui-project.xml**: Arquivo do projeto;
* **README.md**: Arquivo de informações para GitHub.

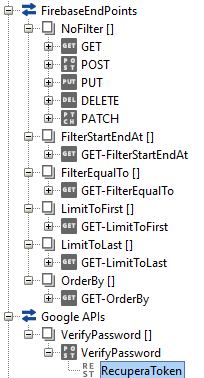


*Figura 5 - Estrutura de Arquivos do Projeto*

## 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.

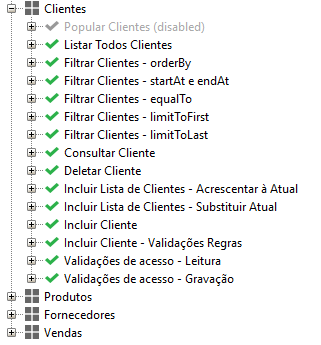


*Figura 6 - Estrutura das Requisições*

## Estrutura de Testes

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

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



*Figura 7- Estrutura dos Testes*

# Testes x Regras de Negócio

A seguir serão apresentados os testes e as regras de negócio validadas por estes, agrupados por suíte de teste.

## Clientes

As regras do nó *clientes* estão definidas da seguinte forma no Firebase:

* **Leitura**: Somente usuários autenticados cujo id esteja definido com valor true no caminho *permissoes > leitura > clientes*.

Definição da regra no Firebase:

*".read": "root.child('permissoes').child('leitura').child('clientes').child(auth.uid).val() == true*"

* **Escrita**: Somente usuários autenticados cujo id esteja definido com valor true no caminho *permissoes > gravacao> clientes*.

Definição da regra no Firebase:

*".write": "root.child('permissoes').child('gravacao').child('clientes').child(auth.uid).val() == true"*

* **Index**: Os campos indexados são *nome, nascimento* e *sexo*.

Definição da regra no Firebase:

*".indexOn" : ["nome", "nascimento", "sexo"]*

* **Validação**: Somente registros que contenham os campos *nome, nascimento* e *sexo* podem ser inseridos. O nome deve ter no máximo 100 caracteres e deve ser do tipo string. O nascimento deve ser uma data no formato ‘*yyyy-mm-dd*’ (Ex.: 1992-06-14). O sexo deve ser do tipo string e deve conter um dos seguintes valores: ‘*F*’ ou ‘*M*’.

Definição da regra no Firebase:

*".validate" :*

*"newData.hasChildren(['nome', 'nascimento', 'sexo']) &&*

*newData.child('nome').val().length <= 100 &&*

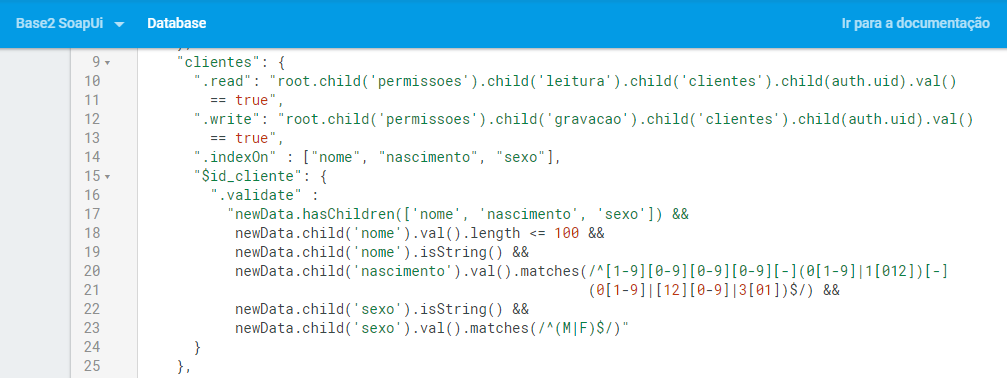
*newData.child('nome').isString() &&*

*newData.child('nascimento').val()*

*.matches(/^[1-9][0-9][0-9][0-9][-](0[1-9]|1[012])[-](0[1-9]|[12][0-9]|3[01])$/) &&*

*newData.child('sexo').isString() &&*

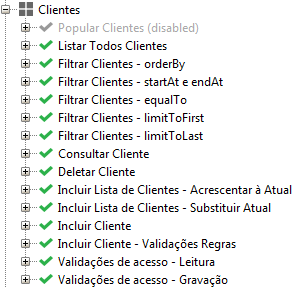
*newData.child('sexo').val().matches(/^(M|F)$/)"*

**

*Figura 8 - Definição das Regras do Nó Clientes*

Os casos de teste do nó *cliente* são os seguintes:

* **Popular Clientes**: Caso de teste que realiza inclusão de clientes na base para execução dos demais testes. Realiza uma chamada POST à requisição *GoogleAPIs* para obter o token de autenticação e em seguida realiza uma chamada PUT à requisição *FirebaseEndPoints*, nó clientes, passando o conteúdo do arquivo *clientes.json*. Está desabilitado e é chamado via script Groovy em cada caso de teste onde um cliente existente é necessário.
* **Listar Todos Clientes**: Caso de teste para executar um GET no nó *clientes* (https://base2-sopaui.firebaseio.com/clientes.json) e validar os registros retornados.
* **Filtrar Clientes - orderBy**: Caso de teste para executar um GET no nó *clientes* usando o parâmetro *orderBy* para ordenar os registros (<https://base2-sopaui.firebaseio.com/clientes.json?orderBy=>””) e validar os registros retornados.
* **Filtrar Clientes - startAt e endAt**: Caso de teste para executar um GET no nó *clientes* usando os parâmetros *startAt*  e *endAt* para limitar os registros ([https://base2-sopaui.firebaseio.com/clientes.json?startAt=](https://base2-sopaui.firebaseio.com/clientes.json?orderBy=)””&endAt=””) e validar os registros retornados.
* **Filtrar Clientes - equalTo**: Caso de teste para executar um GET no nó *clientes* usando o parâmetro *equalTo* para filtrar os registros ([https://base2-sopaui.firebaseio.com/clientes.json?equalTo=](https://base2-sopaui.firebaseio.com/clientes.json?orderBy=)””) e validar os registros retornados.
* **Filtrar Clientes - limitToFirst**: Caso de teste para executar um GET no nó *clientes* usando o parâmetro *limitToFirst* para limitar os registros ([https://base2-sopaui.firebaseio.com/clientes.json?](https://base2-sopaui.firebaseio.com/clientes.json?orderBy=)limitToFirst=””) e validar os registros retornados.
* **Filtrar Clientes - limitToLast**: Caso de teste para executar um GET no nó *clientes* usando o parâmetro *limitToLast* para limitar os registros ([https://base2-sopaui.firebaseio.com/clientes.json?](https://base2-sopaui.firebaseio.com/clientes.json?orderBy=)limitToLast=””) e validar os registros retornados.
* **Consultar Cliente**: Caso de teste para executar um GET em uma chave de cliente específica (https://base2-sopaui.firebaseio.com/clientes/cliente01.json) e validar os registros retornados.
* **Deletar Cliente**: Caso de teste para executar um DELETE em uma chave de cliente específica (https://base2-sopaui.firebaseio.com/clientes/cliente01.json) e validar a exclusão.
* **Incluir Lista de Clientes - Acrescentar à Atual**: Caso de teste para executar um PATCH no nó *clientes* (https://base2-sopaui.firebaseio.com/clientes.json) acrescentando dados ao nó e validar os registros inseridos.
* **Incluir Lista de Clientes - Substituir Atual**: Caso de teste para executar um PUT no nó *clientes* (https://base2-sopaui.firebaseio.com/clientes.json) substituindo os dados do nó e validar os registros inseridos.
* **Incluir Cliente**: Caso de teste para executar um POST no nó *clientes* (https://base2-sopaui.firebaseio.com/clientes.json) criando um novo registro e validar o registro inserido.
* **Incluir Cliente - Validações Regras**: Caso de teste para executar um POST no nó *clientes* (https://base2-sopaui.firebaseio.com/clientes.json) com dados que não atendendem às regras de validação do nó e validar o negativa de inserção.
* **Validações de acesso - Leitura**: Caso de teste para executar um GET no nó *clientes* (https://base2-sopaui.firebaseio.com/clientes.json) e validar os usuários que têm permissão de leitura.
* **Validações de acesso - Gravação**: Caso de teste para executar um POST no nó *clientes* (https://base2-sopaui.firebaseio.com/clientes.json) e validar os usuários que têm permissão de gravação.



*Figura 9 - Casos de Teste do Nó Clientes*