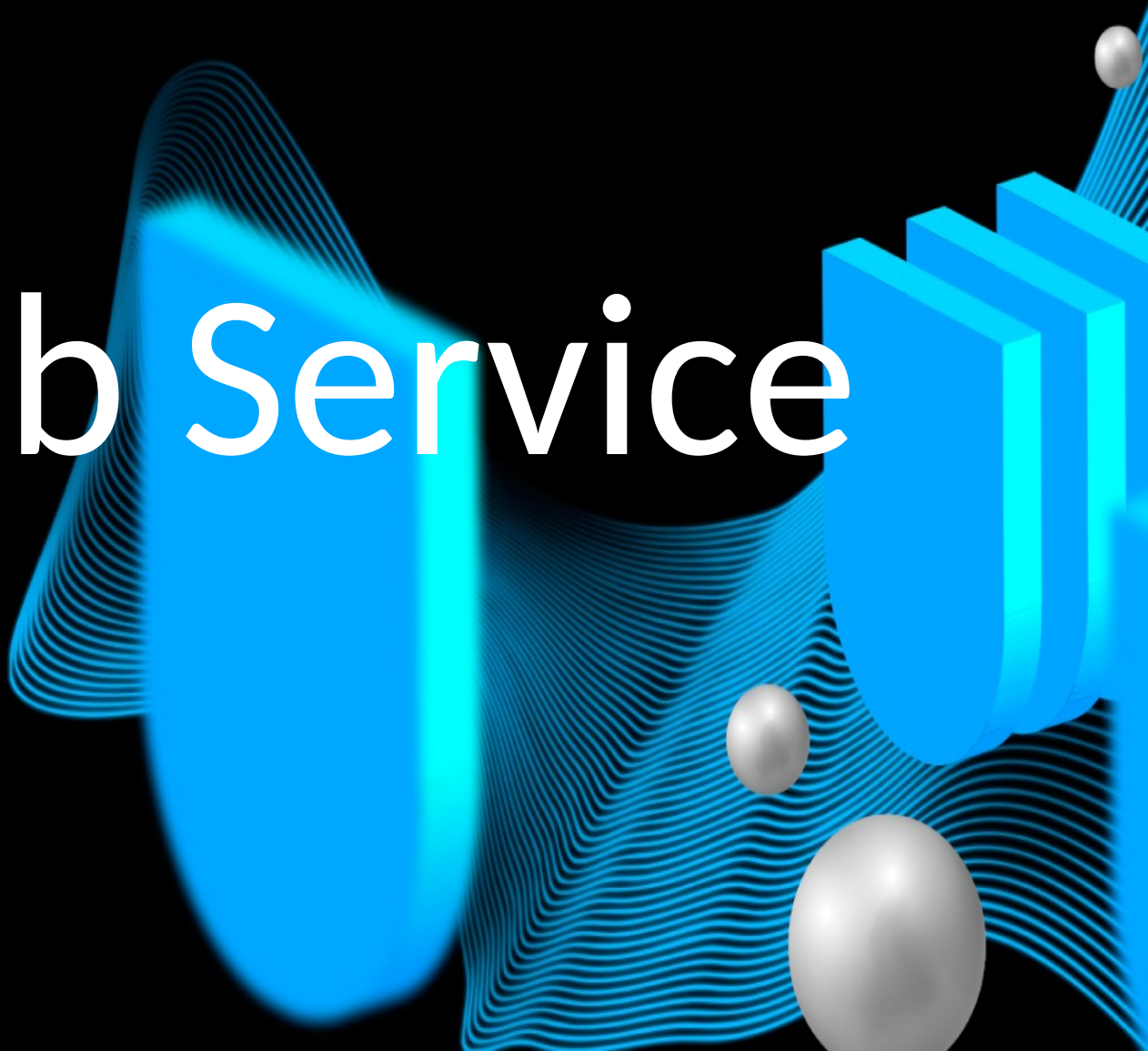
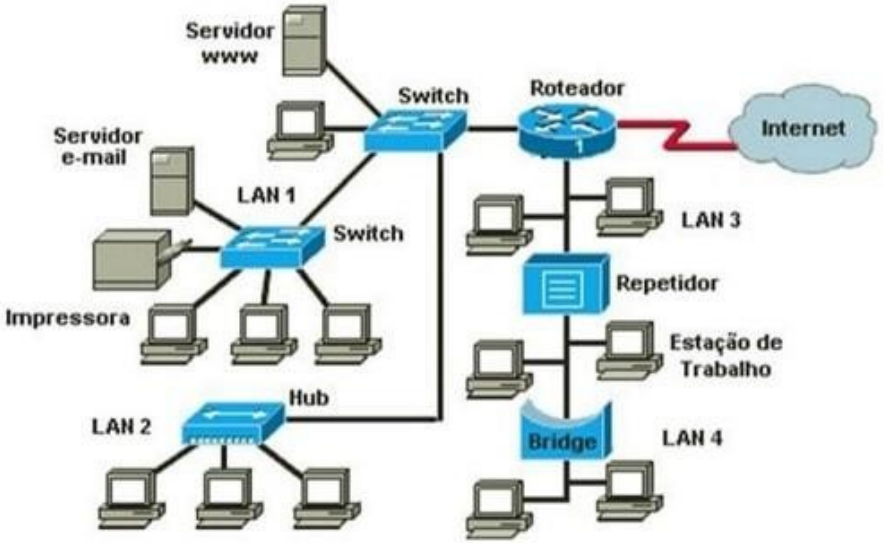

UNITAU
Universidade de Taubaté

Web Service



Redes

Uma rede de computadores é formada por um conjunto de máquinas eletrônicas com processadores capazes de trocar informações e partilhar recursos, interligados por um sub-sistema de comunicação, ou seja, é quando há pelo menos dois ou mais computadores, e outros dispositivos interligados entre si de modo a poderem compartilhar recursos físicos e lógicos, estes podem ser do tipo: dados, impressoras, mensagens,entre outros.



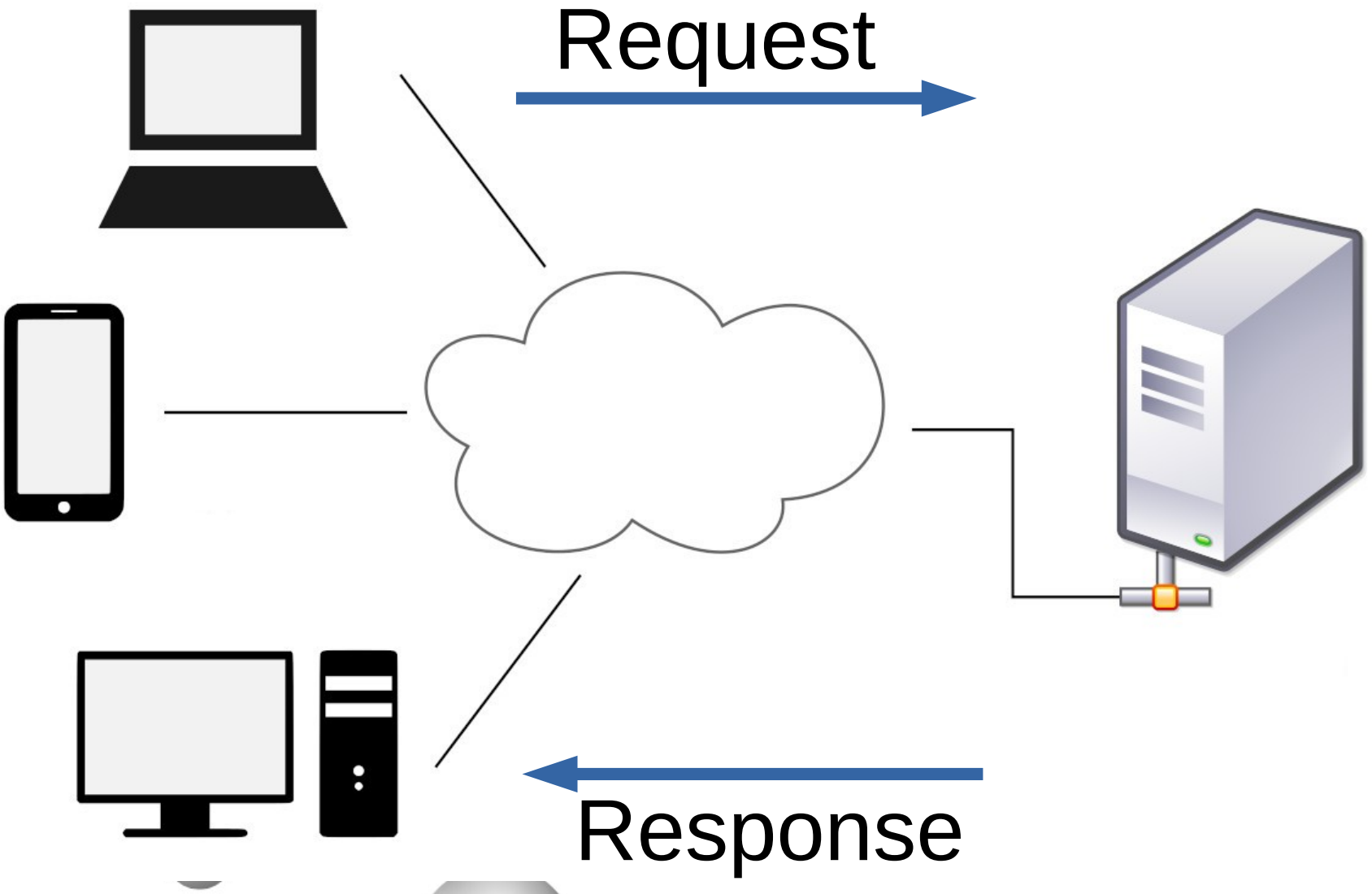
https://www.tiflux.com.br/blog/redes-ethernet-um-briefing/

Serviço

O modelo cliente-servidor (client/server) é uma estrutura de aplicação que distribui as tarefas e cargas de trabalho entre os fornecedores de um recurso ou serviço, designados como servidores, e os requerentes dos serviços, designados como clientes. Geralmente os clientes e servidores comunicam através de uma rede em computadores distintos, mas tanto o cliente quanto o servidor podem residir no mesmo computador.

Um servidor é um host que está executando um ou mais serviços ou programas que compartilham recursos com os clientes. Um cliente é quem solicita um conteúdo ou função do servidor. Os clientes iniciam sessões de comunicação com os servidores que aguardam requisições de entrada, processam as requisições e enviam o resultado das requisições de volta para os clientes.

Serviço



Protocolo

“Na ciência da computação, um protocolo é uma convenção que controla e possibilita uma conexão, comunicação, transferência de dados entre dois sistemas computacionais.

De maneira simples, um protocolo pode ser definido como "as regras que governam" a sintaxe, semântica e sincronização da comunicação. Os protocolos podem ser implementados pelo hardware, software ou por uma combinação dos dois.”

https://pt.wikipedia.org/wiki/Protocolo_(ciência_da_computação)

Cada serviço tem um protocolo próprio que define como deve ser a requisição do cliente e como deve ser a resposta do servidor. Usualmente os serviços são identificados pelo nome do protocolo que os define.

Porta

Para o sistema operacional do servidor, cada serviço é identificado por uma porta, um número usado para identificar para qual programa as requisições daquele serviço devem ser encaminhados. O número da porta de cada serviço é definido na configuração do programa servidor, não sendo possível dois programas distintos “ouvirem” a mesma porta do sistema operacional. Serviços amplamente utilizados e já tradicionais, usualmente (mas não obrigatoriamente) utilizam um número de porta já consagrado para esse serviço e os programas clientes desses serviços utilizam essa porta por padrão para enviar as requisições.

A faixa entre 0 a 1023 é chamada de “portas bem conhecidas” ou “portas do sistema” e normalmente são usadas por serviços amplamente utilizados.

Porta

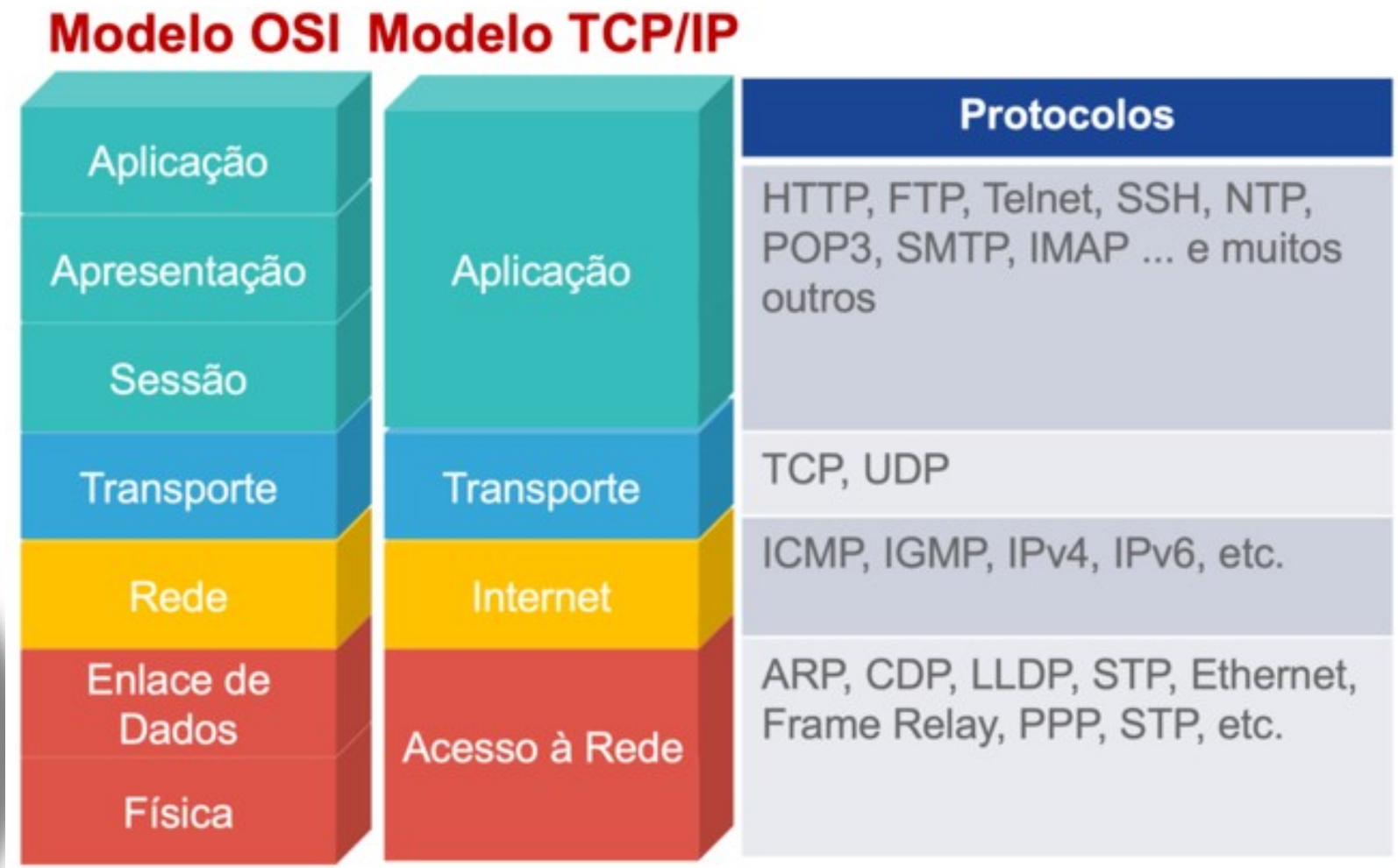
Exemplos de serviços e porta padrão usada:

- **HTTP - Hypertext Transfer Protocol (porta 80)**
- **HTTPS - HTTP Protocol over TLS/SSL (porta 443)**
- **FTP - File Transfer Protocol (portas 20/21)**
- **SMTP - Simple Mail Transfer (porta 25)**
- **POP3 - Post Office Protocol - Version 3 (porta 110)**
- **DNS - Domain Name Server (porta 53)**
- **SSH - Secure Shell (porta 22)**

Camadas de Rede

O conceito de layers ou camadas de software separa os módulos de um software em camadas, cada uma com sua contribuição para a execução do software. O software é dividido em camadas hierárquicas, ou seja, cada camada usa as funções da própria ou da camada anterior, para esconder a complexidade e transparecer as operações ao usuário, seja ele um programa ou uma outra camada. As redes de computadores são divididas em camadas para simplificar a implementação dos softwares e hardwares da rede. Existem diferentes modelos de rede com diferentes estruturas de camadas. O modelo **OSI**, que divide as redes em 7 camadas, é um modelo de referência da **ISO** que tinha como principal objetivo ser um modelo standard, para protocolos de comunicação entre os mais diversos sistemas. A **Internet** usa um modelo de 4 camadas.

Camadas de Rede

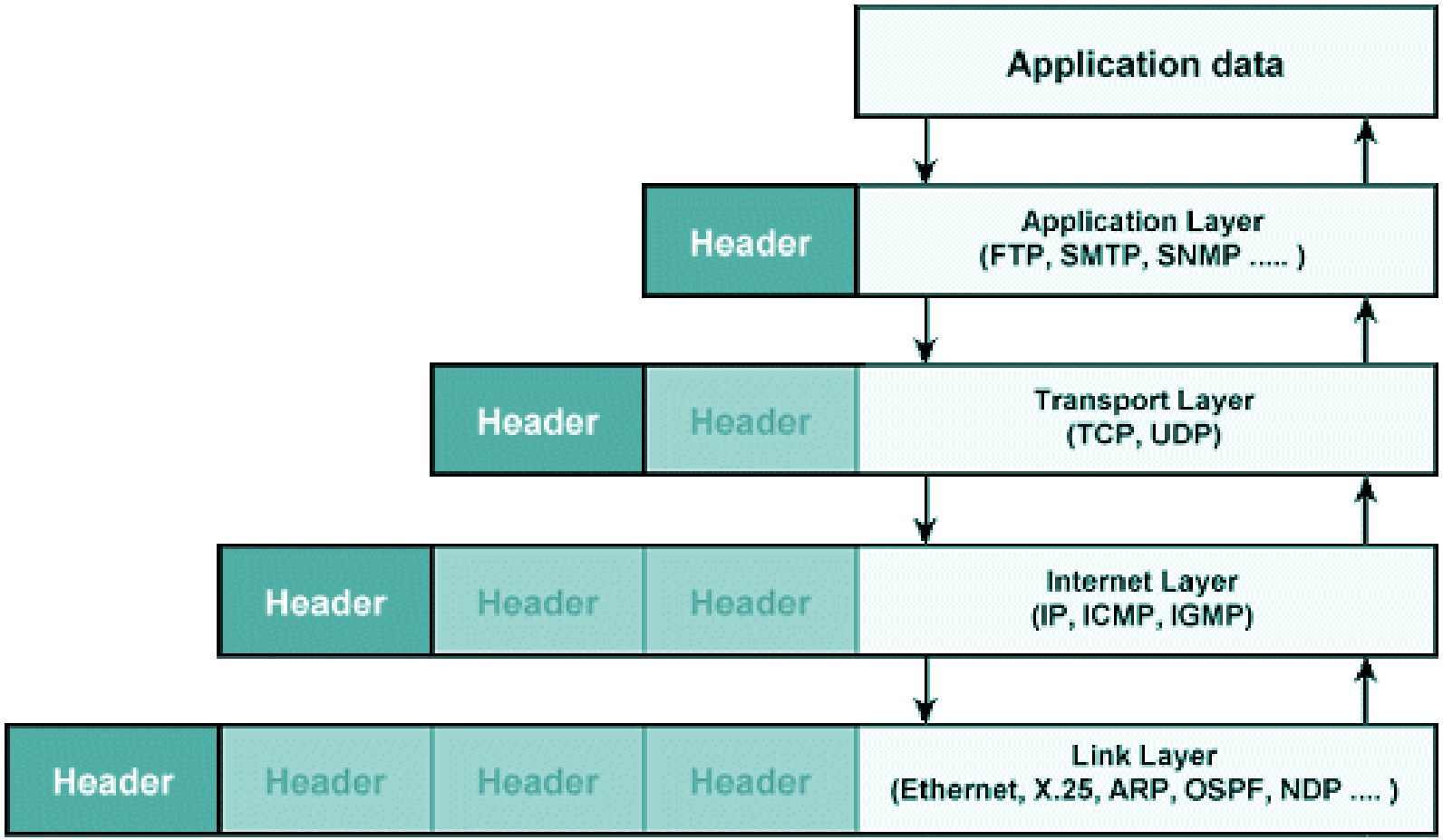


https://wiki.brasilpeeringforum.org/w/Fundamentos_de_Roteamento_para_Provedores

Camadas de Rede

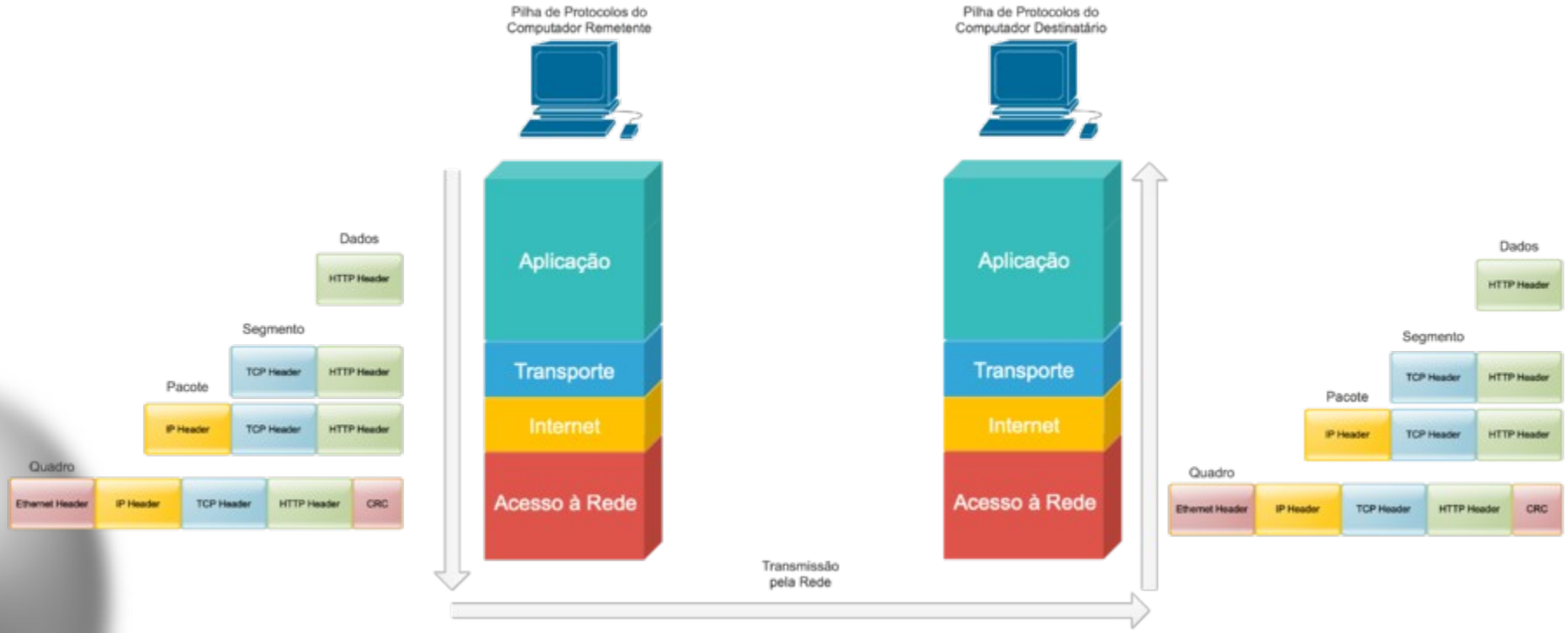
Os serviços de rede (aplicações) transferem os dados (mensagem) entres os computadores utilizando o protocolo de transporte, que por sua vez, utiliza o protocolo de rede para encaminhar a mensagem para o destino e esse por sua vez utiliza o link de rede para repassar a mensagem para o próximo nó do caminho. Essa cadeia é seguida pelo encapsulamento da mensagem de uma camada em pacotes da camada subsequente.

Camadas de Rede



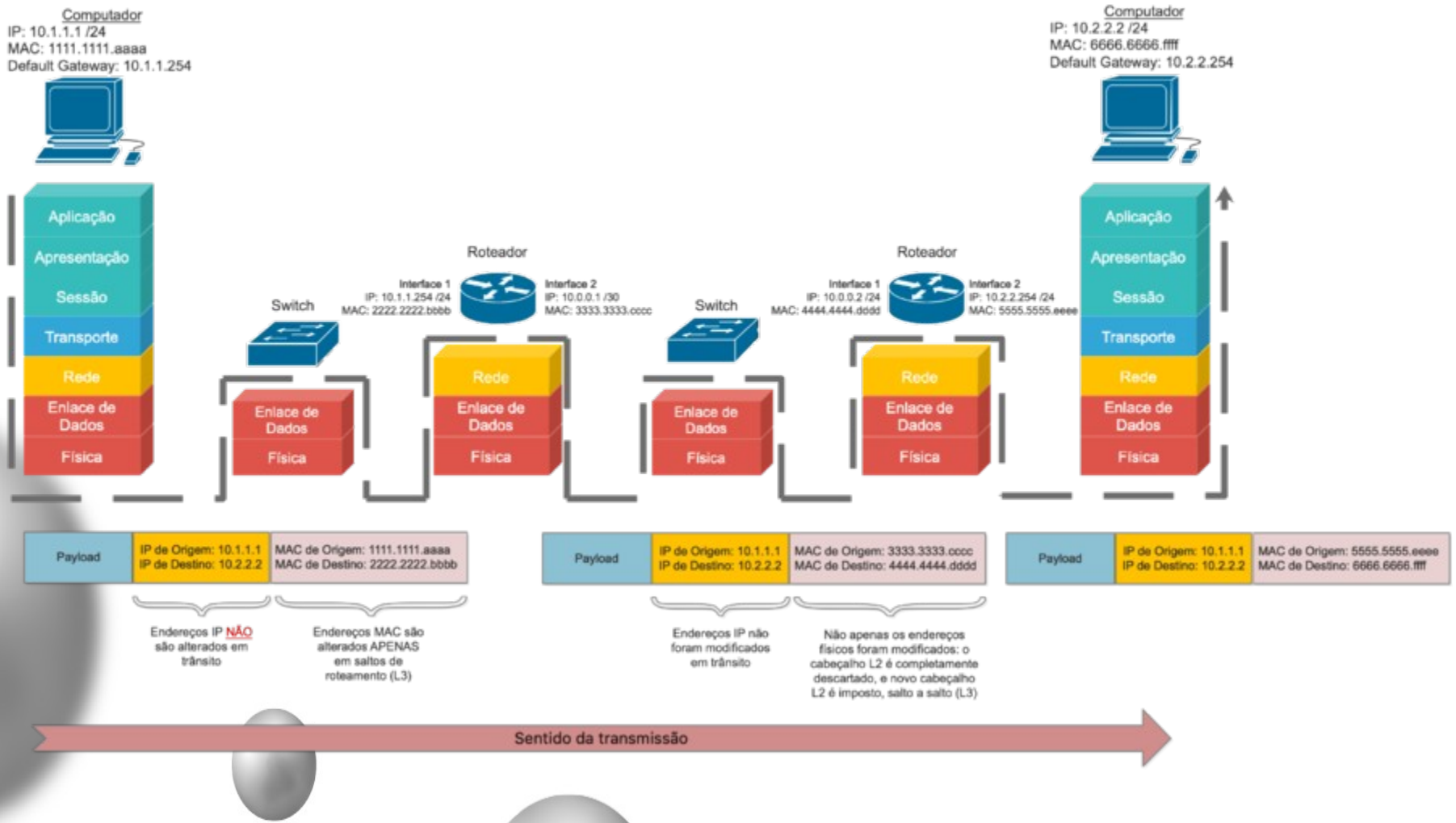
https://www.technologyuk.net/computing/computer-networks/internet/tcp-ip-stack.shtml

Camadas de Rede



https://wiki.brasilpeeringforum.org/w/Fundamentos_de_Roteamento_para_Provedores

Camadas de Rede



https://wiki.brasilpeeringforum.org/w/Fundamentos_de_Roteamento_para_Provedores

HTTP

O protocolo HTTP, Hypertext Transfer Protocol ou Protocolo de Transferência de Hipertexto, é um protocolo da camada de aplicação do TCP/IP cuja função é de proporcionar a transferência de hipertexto. Este protocolo é usado desde 1990. Características do Protocolo HTTP:

- É um protocolo de camada de aplicação da WEB
- É implementado em dois programas: Cliente e Servidor
- O HTTP é quem define a estrutura da mensagem que o cliente vai trocar com o servidor e utiliza TCP como seu protocolo de transporte
- Protocolo sem estado. O que significa que ele não mantém memória sobre suas ações. Ou seja se um cliente fizer uma requisição idêntica a uma anterior a qualquer momento, o HTTP não sabe informar sobre esse histórico.

https://pt.wikiversity.org/wiki/Introdução_às_Redes_de_Computadores/WWW_e_HTTP

HTTPS

HTTPS (**Hyper Text Transfer Protocol Secure** - protocolo de transferência de hipertexto seguro) é uma implementação do protocolo HTTP sobre uma camada adicional de segurança que utiliza o protocolo **SSL/TLS**. Essa camada adicional permite que os dados sejam transmitidos por meio de uma conexão criptografada e que se verifique a autenticidade do servidor e do cliente por meio de certificados digitais. A porta TCP usada por norma para o protocolo HTTPS é a 443.

O protocolo HTTPS é utilizado, em regra, quando se deseja evitar que a informação transmitida entre o cliente e o servidor seja visualizada por terceiros, como por exemplo no caso de compras online.

https://pt.wikipedia.org/wiki/Hyper_Text_Transfer_Protocol_Secure

URI

Um Uniform Resource Identifier (URI) é uma sequência única de caracteres que identifica um recurso lógico ou físico usado por tecnologias da web. URIs podem ser usados para identificar qualquer coisa, incluindo objetos do mundo real, como pessoas e lugares, conceitos ou recursos de informação, como páginas da web e livros. Alguns URIs fornecem um meio de localizar e recuperar recursos de informação em uma rede (na Internet ou em outra rede privada, como um sistema de arquivos de computador ou uma Intranet), são Uniform Resource Locators (URLs). Outros URIs fornecem apenas um nome exclusivo, sem um meio de localizar ou recuperar o recurso ou informações sobre ele. São URNs (Uniform Resource Names). Identificador de Recurso Uniforme -

https://pt.qaz.wiki/wiki/Uniform_Resource_Identifier

URL

Um Uniform Resource Locator (URL) é um URI que especifica os meios de agir sobre ou obter a representação de um recurso, ou seja, especificando seu mecanismo de acesso primário e localização de rede. Por exemplo, a URL http://example.org/wiki/Main_Page refere-se a um recurso identificado como /wiki/Main_Page , cuja representação, na forma de HTML e código relacionado, pode ser obtida por meio do Protocolo de Transferência de Hipertexto (http :) de um host de rede cujo nome de domínio é example.org . Um URN pode ser comparado ao nome de uma pessoa, enquanto um URL pode ser comparado ao seu endereço. Em outras palavras, um URN identifica um item e um URL fornece um método para localizá-lo. Identificador de Recurso Uniforme -

https://pt.qaz.wiki/wiki/Uniform_Resource_Identifier#URLs_and_URNs

Web Service

Inicialmente destinado a transferir informações aos usuários em formatos que são exibidos por um browser, o HTTP passou a ter cada vez mais importância a medida que o crescimento da Internet e da World Wide Web (WWW) criaram uma plataforma tão abrangente para troca de informações, quase onipresente na sociedade moderna, que passou a assumir papeis cada vez mais importantes, muito além da simples “exibição de hipertextos”.

A Internet tornou-se algo amplamente disseminado, constante, indispensável. Incorporada a vida das pessoas, empresas e instituições como algo vital e imperativo. Com esse crescimento, a WWW assumiu o papel de principal plataforma para desenvolvimento de aplicações da atualidade.

Web Service

Para facilitar a integração de diferentes sistemas, é utilizado o conceito de **Web Service** para substituir mecanismos e protocolos próprios utilizados anteriormente.

Usualmente a implementação de Web Services segue o padrão **SOAP** ou a arquitetura **REST**.

Web Service

“Web Service é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Os Web Services são componentes que permitem às aplicações enviar e receber dados. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, um formato intermediário como XML, Json, CSV, etc. “

“Essencialmente, o Web Service faz com que os recursos da aplicação do software estejam disponíveis sobre a rede de forma normalizada.“

https://pt.wikipedia.org/wiki/Web_service

Web Service

“O SOAP (Simple Object Access Protocol) baseia-se numa invocação remota de um método e para tal necessita especificar o endereço do componente, o nome do método e os argumentos para esse método. Estes dados são formatados em XML com determinadas regras e enviados normalmente por HTTP para esse componente.”

“Representational State Transfer (REST), em português Transferência Representacional de Estado, é um estilo de arquitetura de software que define um conjunto de restrições a serem usados para a criação de web services (serviços Web). Os Web Services que estão em conformidade com o estilo arquitetural REST, denominados Web services RESTful, fornecem interoperabilidade entre sistemas de computadores na Internet.”

https://pt.wikipedia.org/wiki/Web_service

SOAP

O SOAP (Simple Object Access Protocol) é um protocolo mais rígido, voltado a disponibilizar métodos bem definidos que podem ser utilizados na aplicação cliente. O SOAP utiliza a a linguagem WSDL (Web Services Description Language) para definir os métodos que o Web Service disponibiliza.

“O WSDL descreve os serviços disponibilizados à rede através de uma semântica XML, este providencia a documentação necessária para se chamar um sistema distribuído e o procedimento necessário para que esta comunicação se estabeleça. Enquanto que o SOAP especifica a comunicação entre um cliente e um servidor, o WSDL descreve os serviços oferecidos. “

https://pt.wikipedia.org/wiki/Web_service

REST

REST (Representational State Transfer) é mais um estilo de arquitetura de software do que um protocolo claramente definido. Propõe que o Web Service disponibilize operações padrão a serem executadas sobre os recursos do sistema, que podem ser acessados baseados na URI do recurso.

A proposta do REST é ser uma forma simples de interação entre cliente e servidor. Facilitando a implementação de aplicações web. Embora não seja exclusivo para o protocolo HTTP, a implementação de Web Services RESTful é muito mais comum com o protocolo HTTP, utilizado os métodos do HTTP para definir as operações disponíveis sobre os recursos. Por exemplo:

- **[GET] http://www.exemplo.com/alunos deve retornar um XML ou JSON com a relação de alunos**
- **[DELETE] http://www.exemplo.com/alunos/1 deve deletar o aluno com ID=1**

REST

Principais métodos do protocolo HTTP e o cenário de utilização de cada um deles:

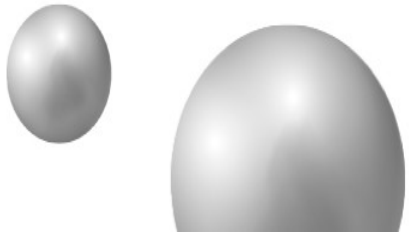
GET	Obter os dados de um recurso
POST	Criar um novo recurso
PUT	Substituir os dados de um determinado recurso
PATCH	Atualizar parcialmente um determinado recurso
DELETE	Excluir um determinado recurso
HEAD	Similar ao GET, mas utilizado apenas para se obter os cabeçalhos de resposta, sem os dados em si
OPTIONS	Obter quais manipulações podem ser realizadas em um determinado recurso

https://blog.caelum.com.br/rest-principios-e-boas-praticas/

Exemplo

Para exemplo de uma aplicação com um Web Service REST, utilizaremos um Web Service criado com Spring, disponível em:

https://student-rest.herokuapp.com



Exemplo

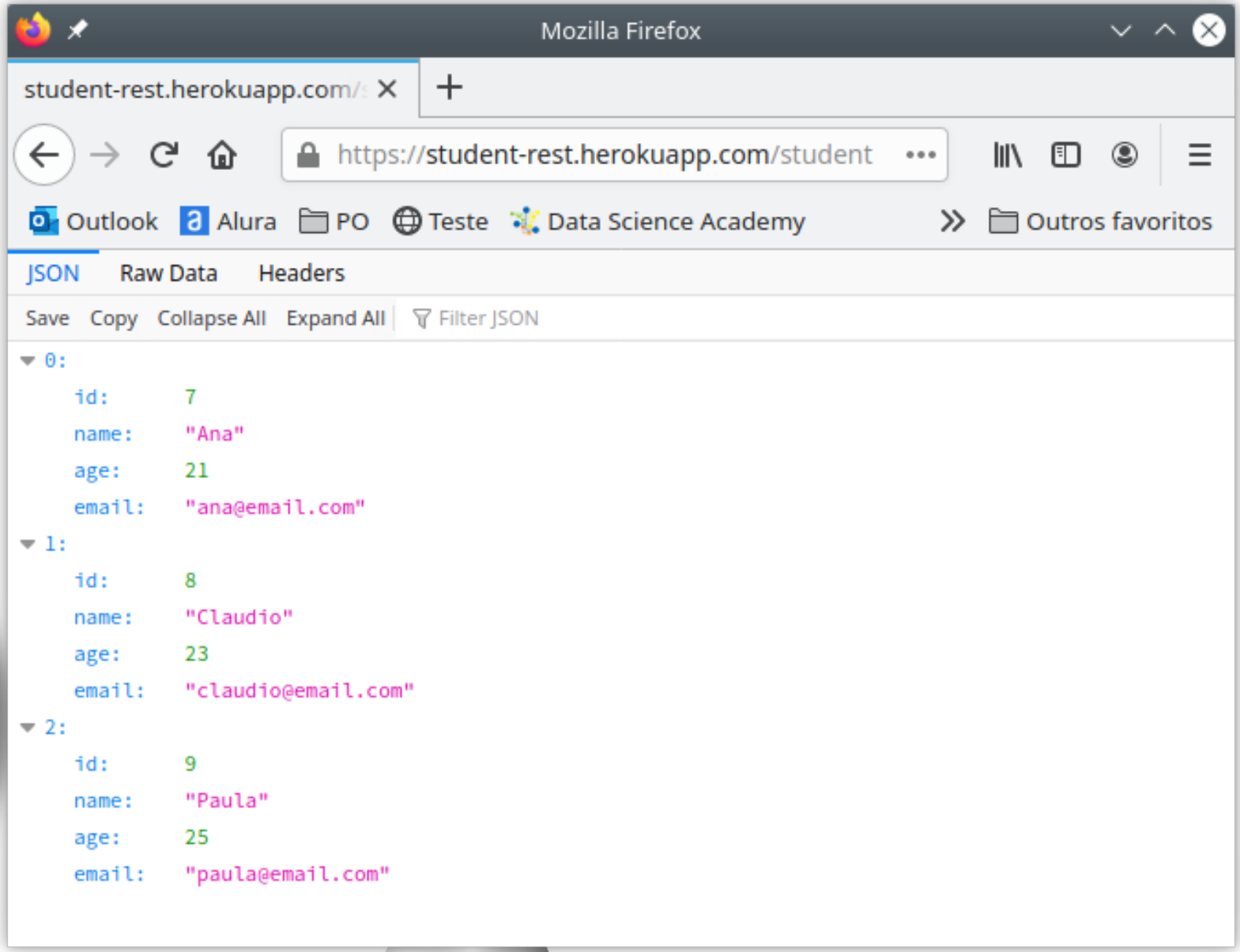
Este Web Service disponibiliza uma API aberta para testes, sem autenticação, que implementa as seguintes funções:

GET	`https://student-rest.herokuapp.com/student`	Obter a lista dos estudantes
GET	`https://student-rest.herokuapp.com/student/{id}`	Obter os dados de um estudante pelo id
POST	`https://student-rest.herokuapp.com/student`	Inserir um novo estudante
PUT	`https://student-rest.herokuapp.com/student/{id}`	Alterar um estudante pelo id
DELETE	`https://student-rest.herokuapp.com/student/{id}`	Deletar um estudante pelo id

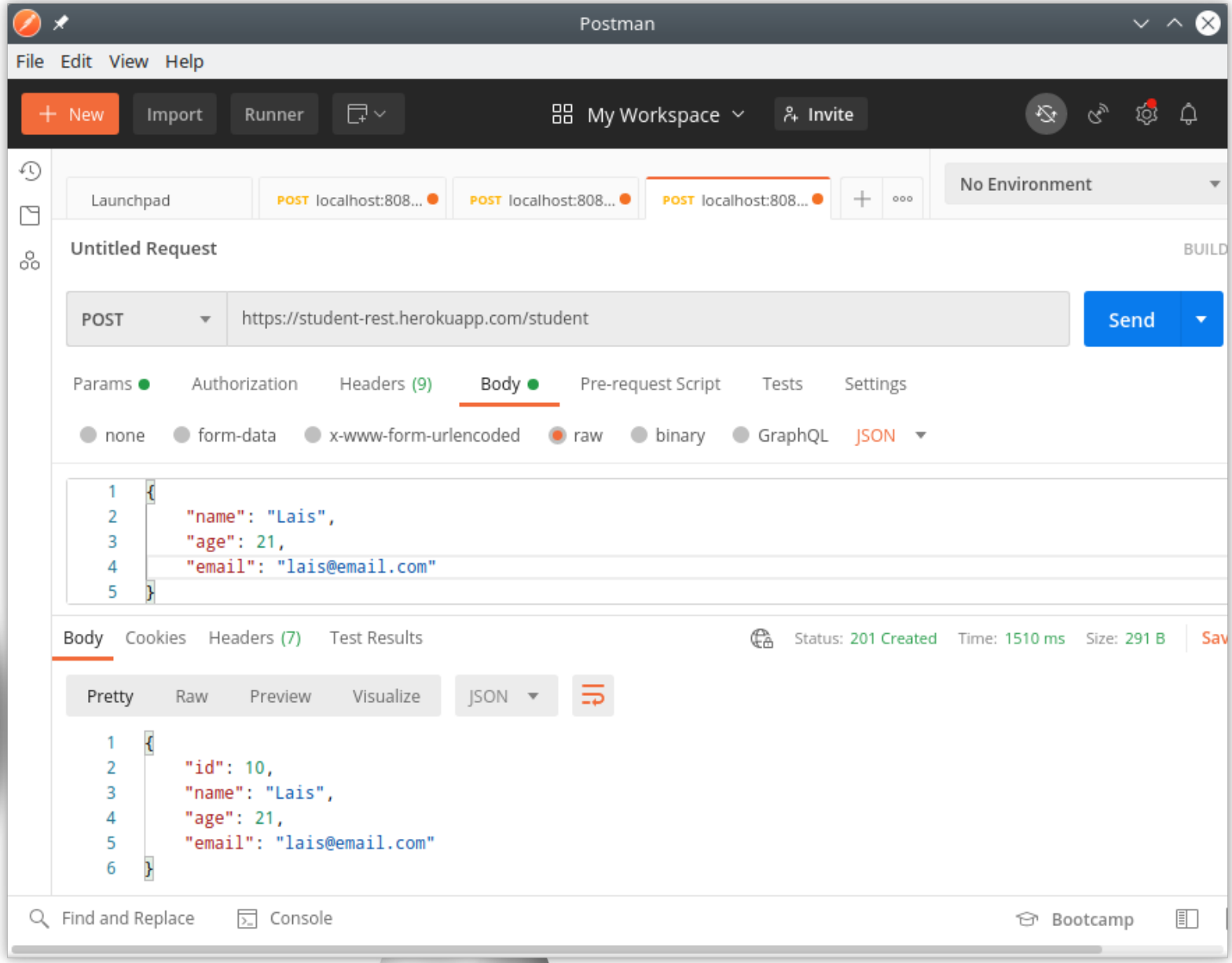
Exemplo

O Web Service pode ser testado de várias formas, usando o browser, programas como o Postman, usando Curl e outras formas.

Browser


Mozilla Firefox
student-rest.herokuapp.com/
https://student-rest.herokuapp.com/student
Outlook
Alura
PO
Teste
Data Science Academy
Outros favoritos
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
0:
id: 7
name: "Ana"
age: 21
email: "ana@email.com"
1:
id: 8
name: "Claudio"
age: 23
email: "claudio@email.com"
2:
id: 9
name: "Paula"
age: 25
email: "paula@email.com"

Postman


Postman
File Edit View Help
New
Import
Runner
My Workspace
Invite
Launchpad
POST localhost:808...
POST localhost:808...
POST localhost:808...
No Environment
Untitled Request
BUILD
POST
https://student-rest.herokuapp.com/student
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"name": "Lais",
"age": 21,
"email": "lais@email.com"
}
Body
Cookies
Headers (7)
Test Results
Status: 201 Created
Time: 1510 ms
Size: 291 B
Pretty
Raw
Preview
Visualize
JSON
{
"id": 10,
"name": "Lais",
"age": 21,
"email": "lais@email.com"
}
Find and Replace
Console
Bootcamp

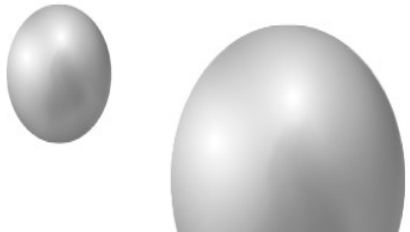
Php+Curl

student_get.php:

```
<?php
$url = 'https://student-rest.herokuapp.com/student';
if ($argc > 1) {
  $url = $url.'/'.(string)$argv[1];
}
$curl = curl_init($url);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
$result = curl_exec($curl);
$httpcode = curl_getinfo($curl, CURLINFO_HTTP_CODE);
curl_close($curl);
echo 'HTTP code: ' . $httpcode;
echo "\n";
$json =json_decode($result);
print_r(json_encode($json,JSON_PRETTY_PRINT));
echo "\n";
?>
```

Php+Curl

```
> php student_get.php
HTTP code: 200
[
    {
        "id": 7,
        "name": "Ana",
        "age": 21,
        "email": "ana@email.com"
    },
    {
        "id": 11,
        "name": "Lais",
        "age": 21,
        "email": "lais@email.com"
    }
]
```



Php+Curl

```
> php student_get.php 7
HTTP code: 200
{
    "id": 7,
    "name": "Ana",
    "age": 21,
    "email": "ana@email.com"
}
```

Php+Curl

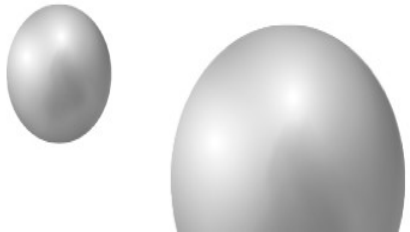
student_post.php:

```
<?php
$url = 'https://student-rest.herokuapp.com/student';
$data = array(
  "name" => "Paula",
  "age" => "20",
  "email" => "paula@email.com",
);
$postdata = json_encode($data);
$curl = curl_init($url);
curl_setopt($curl, CURLOPT_POST, 1);
curl_setopt($curl, CURLOPT_POSTFIELDS, $postdata);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($curl, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
$result = curl_exec($curl);
$httpcode = curl_getinfo($curl, CURLINFO_HTTP_CODE);
curl_close($curl);
```

Php+Curl

```
echo 'HTTP code: ' . $httpcode;
echo "\n";
$json =json_decode($result);
print_r(json_encode($json,JSON_PRETTY_PRINT));
echo "\n";
?>
```

```
> php student_post.php
HTTP code: 201
{
    "id": 12,
    "name": "Paula",
    "age": 20,
    "email": "paula@email.com"
}
```



Php+Curl

student_put.php:

```
<?php
$url = 'https://student-rest.herokuapp.com/student/'.
(string)$argv[1];
$data = array(
  "name" => "Carla",
  "age" => "30",
  "email" => "carla@email.com",
);
$postdata = json_encode($data);
$curl = curl_init($url);
curl_setopt($curl, CURLOPT_POST, 1);
curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PUT");
curl_setopt($curl, CURLOPT_POSTFIELDS, $postdata);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($curl, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
$result = curl_exec($curl);
$httpcode = curl_getinfo($curl, CURLINFO_HTTP_CODE);
curl_close($curl);
```

Php+Curl

```
echo 'HTTP code: ' . $httpcode;
echo "\n";
$json =json_decode($result);
print_r(json_encode($json,JSON_PRETTY_PRINT));
echo "\n";
?>
```

```
> php student_put.php 12
HTTP code: 200
{
    "id": 12,
    "name": "Carla",
    "age": 30,
    "email": "carla@email.com"
}
```

Php+Curl

student_delete.php:

```
<?php
$url = 'https://student-rest.herokuapp.com/student/'.
(string)$argv[1];
$curl = curl_init($url);
curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "DELETE");
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
$result = curl_exec($curl);
$httpcode = curl_getinfo($curl, CURLINFO_HTTP_CODE);
curl_close($curl);
echo 'HTTP code: ' . $httpcode;
echo "\n";
$json =json_decode($result);
print_r(json_encode($json,JSON_PRETTY_PRINT));
echo "\n";
?>
```

```
> php student_delete.php 12
HTTP code: 200
null
```

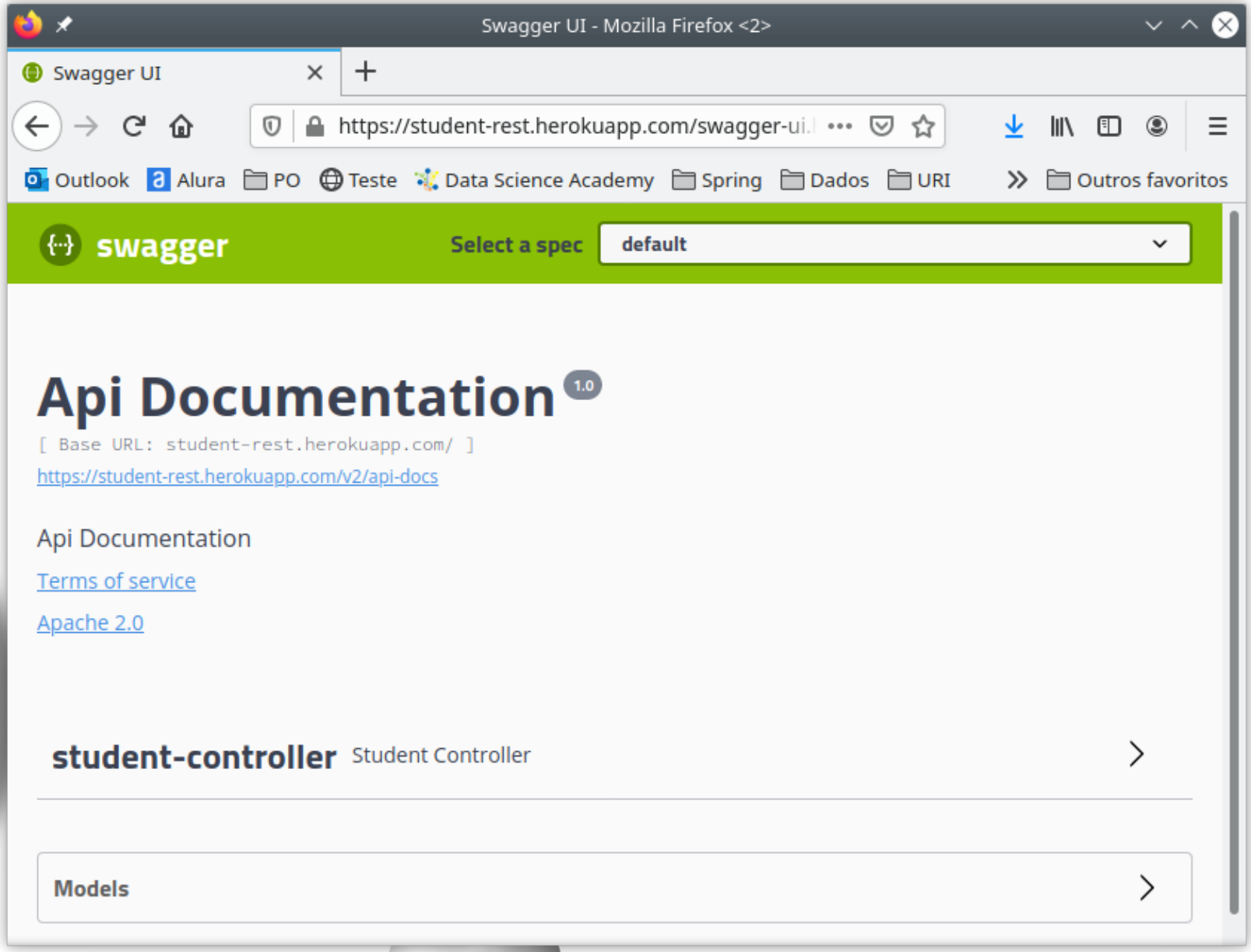
Swagger

Swagger é uma ferramenta que permite documentar a API de um Web Service REST, disponibilizando uma documentação online que também permite testar a API do Web Service.

Para o Web Service de exemplo, a documentação está disponível em:

https://student-rest.herokuapp.com/swagger-ui.html

Swagger


Swagger UI - Mozilla Firefox <2>
Swagger UI
https://student-rest.herokuapp.com/swagger-ui.
Outlook
Alura
PO
Teste
Data Science Academy
Spring
Dados
URI
Outros favoritos
swagger
Select a spec
default
Api Documentation
1.0
[Base URL: student-rest.herokuapp.com/]
https://student-rest.herokuapp.com/v2/api-docs
Api Documentation
Terms of service
Apache 2.0
student-controller
Student Controller
Models

Rest

Crie uma nova aplicação Flutter com o nome **rest**.

Altere o arquivo **pubspec.yaml** para incluir as dependências de **http** e **preferences**:

```
dependencies:
  http: ^0.13.0
  preferences: ^5.1.0
  flutter:
    sdk: flutter
```

Instale os pacotes adicionados nas dependências.

Rest

Copie toda a estrutura de subdiretórios e arquivos do subdiretório **lib** da aplicação **persisitencia_postgresql**.

Altere o nome da aplicação no arquivo **home.dart** no subdiretório **lib/pages**:

```
appBar: AppBar(
  title: Text("Rest"),
```

Rest

Altere a função main no arquivo main.dart:

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await PrefService.init(prefix: 'pref_');
  PrefService.setDefaultValues({'host': 'student-
rest.herokuapp.com'});

  runApp(MyApp());
}
```

Rest

Altere o arquivo configuration.dart no subdiretório lib/page:

```
import 'package:flutter/material.dart';
import 'package:preferences/preferences.dart';

class Configuration extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Configuracao"),
      ),
      body: PreferencePage([
        PreferenceTitle('Web Service'),
        TextFieldPreference(
          'Host',
          'host',
        ),
      ]),
    );
  }
}
```

Rest

Altere o arquivo database.dart no subdiretório lib/database:

```
import 'package:http/http.dart' as http;
import 'package:http/http.dart';
import 'package:preferences/preferences.dart';
import 'dart:convert';

// TimeOut para as requicoes
const _TIMEOUT = Duration(seconds: 10);

class DatabaseHelper {
  static final DatabaseHelper _instance =
DatabaseHelper.internal();
  factory DatabaseHelper() => _instance;
  DatabaseHelper.internal();

  static String _host;

  static _setHost() {
    _host = PrefService.getString('host').trim();
  }
```

Rest

```
static Future<bool> initDb() async {
  _setHost();
  return true;
}
```

Rest

```
  static Future<List<Map>> getAll(String _table) async {
    print("Database getAll");
    List<Map> ret;
    String _errorMsg = "";
    final _response = await http.get(Uri.https("$
{_host}","/${_table}"))
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 200) {
        final _parsed = json.decode(_response.body);
        ret = List<Map>();
        for (int i = 0; i < _parsed.length; i++)
          ret.add(_parsed[i]);
      } else
        _errorMsg = responseError(_response, null);
    }
```

Rest

```
  if (ret != null)
    return ret;
  else
    return Future<List<Map>>.error(_errorMsg);
}
```

Rest

```
  static Future<Map> getByID(String _table, int _id) async {
    print("Database getByID");
    Map ret;
    String _errorMsg = "";
    final _response = await http.get(Uri.https("$
{_host}","/${_table}/${_id}"))
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 200)
        ret = json.decode(_response.body);
      else
        _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else
      return Future<Map>.error(_errorMsg);
  }
```

Rest

```
  static Future<int> insert(String _table, Map _map) async {
    print("Database insert");
    int ret;
    String _errorMsg = "";
    _map.remove('id');
    final _response = await http.post(Uri.https("$
{_host}","/${_table}"),
        body: json.encode(_map),
        headers: {
          "Accept": "application/json",
          "Content-Type": "application/json"
        },
        encoding: Encoding.getByName("utf-8")
      )
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
```

Rest

```
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 201)
        ret = 1;
      else
        _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }
```

Rest

```
  static Future<int> update(String _table, Map _map) async {
    print("Database update");
    int ret;
    String _errorMsg = "";
    final _response = await http.put(Uri.https("$
{_host}","/${_table}/${_map['id']}"),
        body: json.encode(_map),
        headers: {
          "Accept": "application/json",
          "Content-Type": "application/json"
        },
        encoding: Encoding.getByName("utf-8")
      )
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
```

Rest

```
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 200)
          ret = 1;
      else
        _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }
```

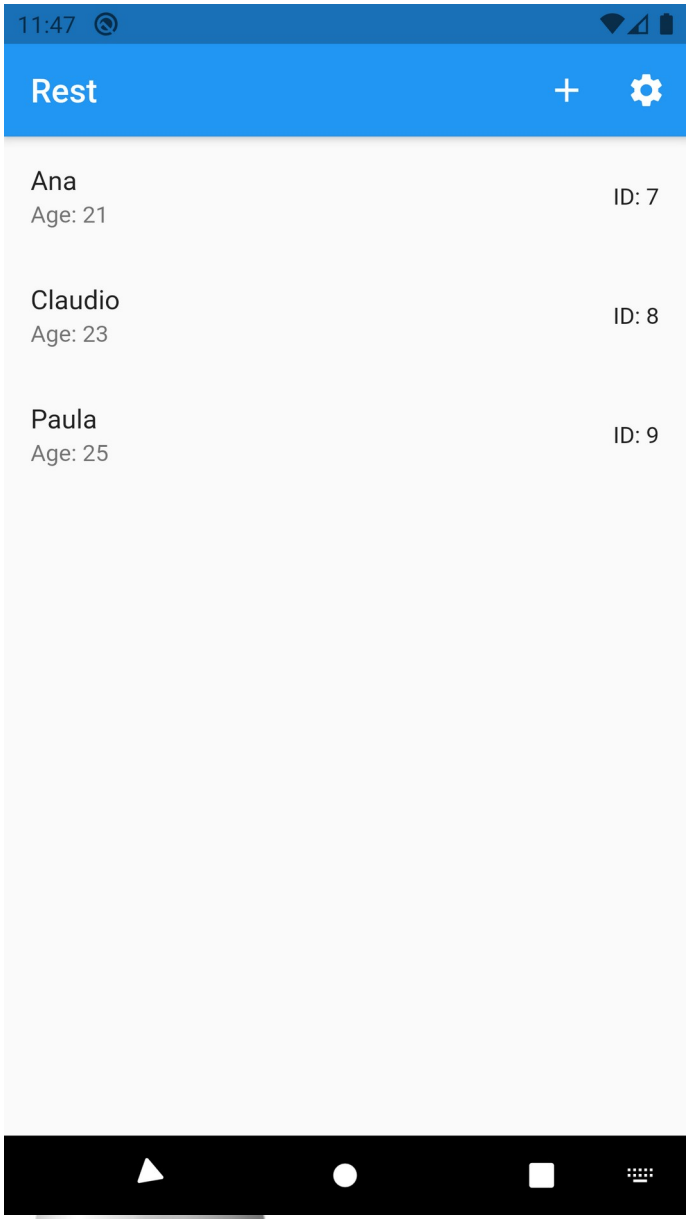
Rest

```
  static Future<int> delete(String _table, int _id) async {
    print("Database delete");
    int ret;
    String _errorMsg = "";
    final _response = await http.delete(Uri.https("$
{_host}","/${_table}/${_id}"))
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 200)
          ret = 1;
      else
        _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }
```

Rest

```
  static String responseError(var _response, String _body) {
    String ret = 'Bad status code ${_response.statusCode}.';
    _body ??= _response.body;
    if (_body != null )
      ret = ret+'\n${_body}';
    return ret;
  }

  void close() {
    print("CloseDB");
  }
}
```

Rest



Autenticação

Um dos principais problemas da área de informática é a segurança. Ao implementar um Web Service, deve se determinar se é necessário alguma restrição de acesso.

Além de mecanismos de controle de acesso à URI do Web Service, também é possível implementar algum método de autenticação para acessar o Web Service, impedindo que qualquer pessoa tenha acesso aos recursos do Web Service.

É desejável que o Web Service seja stateless, ou seja, o processamento de uma requisição não deve depender de informações armazenadas no servidor em requisições anteriores. Toda informação necessária para o processamento da requisição deve ser enviada com a requisição. Web Service stateless não devem armazenar informações em sessões do servidor.

Autenticação

Desta forma, Web Services devem utilizar métodos de autenticação stateless.

Existem vários métodos de autenticação que podem ser utilizados. Alguns dos métodos possíveis são:

- HTTP Basic
- HTTP Digest
- Tokens
- OAuth

Autenticação HTTP Basic

O esquema "Basic" de autenticação HTTP é definido em RFC 7617, transmitindo credenciais como pares de ID/senhas de usuários, codificadas usando base64.”

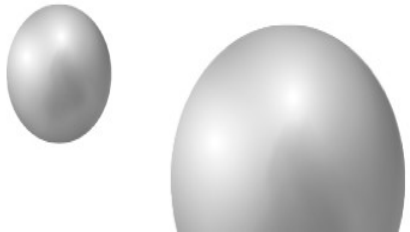
“Como o ID e senha do usuário são transmitidos através da rede como texto claro (é codificado em base64, mas base64 é uma codificação reversível), o esquema básico de autenticação não é seguro. HTTPS / TLS devem ser usados em conjunto com autenticação básica. Sem esses aprimoramentos de segurança adicionais, a autenticação básica não deve ser usada para proteger informação sensível ou valiosa.”

https://developer.mozilla.org/pt-BR/docs/Web/HTTP/authentication#Esquema_Basic_de_autenticação

Autenticação HTTP Digest

A autenticação Digest é um esquema de desafio / resposta que se destina a substituir a autenticação básica. O servidor envia uma cadeia de caracteres de dados aleatórios chamados um nonce ao cliente como um desafio. O cliente responde com um hash que inclui o nome de usuário, senha e nonce entre informações adicionais. A complexidade que introduz essa troca e o hash de dados torna mais difícil roubar e reutilizar as credenciais do usuário com esse esquema de autenticação.

https://docs.microsoft.com/pt-br/dotnet/framework/wcf/feature-details/understanding-http-authentication

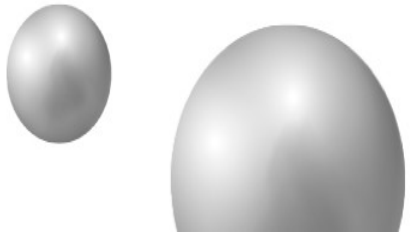


Autenticação por Token

Na autenticação por token, a aplicação envia as credenciais do usuário para o servidor. Se as credenciais forem validadas, o servidor envia um token para a aplicação.

Ao fazer requisições para o servidor, a aplicação envia o token para autenticar a identidade do solicitante. O servidor valida o token para verificar se a requisição será aceita.

O servidor pode utilizar um algoritmo próprio para gerar e validar o token ou utilizar um padrão já estabelecido como o **JWT** (**Json Web Token**).



JWT

“O JWT é um padrão (RFC-7519) de mercado que define como transmitir e armazenar objetos JSON de forma compacta e segura entre diferentes aplicações. Os dados nele contidos podem ser validados a qualquer momento pois o token é assinado digitalmente. Ele é formado por três seções: Header, Payload e Signature.

- **O Header é um objeto JSON que define informações sobre o tipo do token (typ), nesse caso JWT, e o algorítmo de criptografia usado em sua assinatura (alg), normalmente HMAC SHA256 ou RSA.**
- **O Payload é um objeto JSON com as Claims (informações) da entidade tratada, normalmente o usuário autenticado.**
- **A assinatura é a concatenação dos hashes gerados a partir do Header e Payload usando base64UrlEncode, com uma chave secreta ou certificado RSA.”**

https://medium.com/tableless/entendendo-tokens-jwt-json-web-token-413c6d1397f6

JWT

1 2

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o 3

1 Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

2 Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

3 Signature

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

https://research.securitum.com/jwt-json-web-token-security/

OAuth

”OAuth é um padrão aberto para autorização, comumente utilizado para permitir que os usuários da Internet possam fazer logon em sites de terceiros usando suas contas do Google, Facebook, Microsoft, Twitter, etc.—mas, sem expor suas senhas. Geralmente, o OAuth fornece aos clientes um "acesso seguro delegado" aos recursos do servidor em nome do proprietário do recurso. Ele especifica um processo para proprietários de recursos para autorizar o acesso de terceiros aos seus recursos de servidor sem compartilhar suas credenciais. [...] o OAuth permite essencialmente tokens de acesso a ser emitidos para clientes de terceiros, mediante autorização do servidor, com a aprovação do proprietário do recurso. O terceiro, em seguida, usa o token de acesso para recursos protegidos hospedados pelo servidor.”

https://pt.wikipedia.org/wiki/OAuth

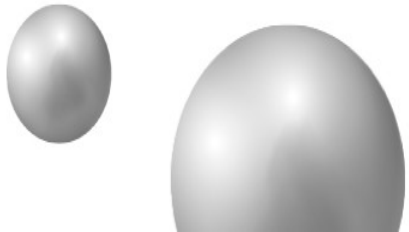
Exemplo

Para a aplicação de exemplo com um Web Service REST utilizando autenticação com JWT, utilizaremos um Web Service criado com Spring, disponível em:

https://student-jwt.herokuapp.com

A documentação está disponível em:

https://student-jwt.herokuapp.com/swagger-ui.html



Exemplo

Este Web Service disponibiliza uma API que implementa as mesmas funções do Web Service anterior, acrescentando a autenticação com JWT e acrescenta as seguintes funções:

POST	`https://student-jwt.herokuapp.com/usuario`	Inserir um usuário
POST	`https://student-jwt.herokuapp.com/password/forgot`	Enviar e-mail para reset da senha do usuário
POST	`https://student-jwt.herokuapp.com/auth`	Autenticar o usuário

As demais funções adicionais implementadas no Web Service são destinadas a uso interno e não devem ser acessadas pela aplicação.

JWT

Crie uma nova aplicação Flutter com o nome **rest_jwt**.

Altere o arquivo **pubspec.yaml** para incluir as dependências de **http** e **preferences**:

```
dependencies:
  http: ^0.13.0
  preferences: ^5.1.0
  jwt_decoder: 2.0.0
  flutter:
    sdk: flutter
```

Instale os pacotes adicionados nas dependências.

Copie toda a estrutura de subdiretórios e arquivos do subdiretório **lib** da aplicação **rest**.

JWT

Altere o arquivo main.dart no subdiretório lib:

```
import 'package:flutter/material.dart';
import 'package:preferences/preferences.dart';
import 'database/database.dart';
import 'pages/signin.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await PrefService.init(prefix: 'pref_');
  PrefService.setDefaultValues({'host': 'student-
jwt.herokuapp.com'});
  final DatabaseHelper _dbHelper = DatabaseHelper();
  DatabaseHelper.initDb();

  runApp(MyApp());
}
```

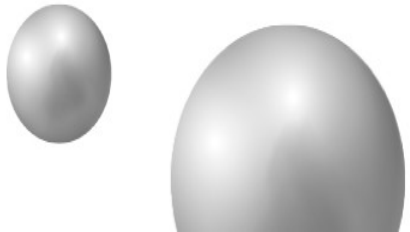
JWT

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity:
VisualDensity.adaptivePlatformDensity,
      ),
      home: SignIn(),
    );
  }
}
```

JWT

Crie o arquivo user.dart no subdiretório lib/models:

```
//
// Modelo para dados do usuario
//
class User {
  int id;
  String name;
  String email;
  String password;
  String confirm;

  // Construtor padrao
  User({this.id, this.name, this.email, this.password,
this.confirm});
```



JWT

```
// Construtor de um usuario vazio
User.empty() {
  id = 0;
  name = "";
  email = "";
  password = "";
  confirm = "";
}

// Construtor baseando em JSON
factory User.fromJson(Map<String, dynamic> json) {
  return User(
    id: json['id'],
    name: json['name'],
    email: json['email']
  );
}
```

JWT

```
  // Converte um usuario para JSON
  Map<String, dynamic> toJson() {
    var map = {
      'id': id,
      'name': name,
      'email': email,
      'password': password,
      'confirm': confirm
    };
    return map;
  }
}
```

JWT

Crie o arquivo session.dart no subdiretório lib/models:

```
//
// Modelo para dados da sessao
//
class Session {
  String email;
  String password;

  // Construtor padrao
  Session({this.email, this.password});

  // Construtor de uma sessao vazia
  Session.empty() {
    email = "";
    password = "";
  }
```

JWT

```
  // Construtor baseando em JSON
  factory Session.fromJson(Map<String, dynamic> json) {
    return Session(
      email: json['email'],
      password: json['password']
    );
  }

  // Converte uma sessao para JSON
  Map<String, dynamic> toJson() {
    var map = {
      'email': email,
      'password': password
    };
    return map;
  }
}
```

JWT

Crie o arquivo user_dao.dart no subdiretório lib/dao:

```
import 'dart:async';
import '../models/user.dart';
import '../database/database.dart';

class UserDao {
  String _errorMsg = "";

  String get errorMsg => _errorMsg;

  Future<int> insert(User _user) async {
    print("DAO insert");
    int ret;
    _errorMsg = "";
    ret = await DatabaseHelper.insert('usuario',
      _user.toJson()
    ).catchError((_error) {
      _errorMsg = _error;
    });
    return ret;
  } }
```

JWT

Crie o arquivo user_bloc.dart no subdiretório lib/bloc:

```
import 'dart:async';
import '../models/user.dart';
import '../dao/user_dao.dart';

class UserBLoC {
  static final UserBLoC _singleton = UserBLoC._internal();
  final UserDao _dao = UserDao();

  factory UserBLoC() => _singleton;

  UserBLoC._internal();

  Future<int> insert(User user) async {
    print("BLoC insert");
    int ret = await _dao.insert(user);
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_dao.errorMsg);
  } }
```

JWT

No arquivo student_dao.dart no subdiretório lib/dao:

Remover:

```
  final DatabaseHelper _dbHelper = DatabaseHelper();
```

Remover:

```
  Future<bool> init() async {
    _errorMsg = "";
    bool ret = await
DatabaseHelper.initDb().catchError((_error) {
      _errorMsg = _error;
    });
    ret ??= false;
    return ret;
  }
```

Remover:

```
  close() {
    _dbHelper.close();
  }
```

JWT

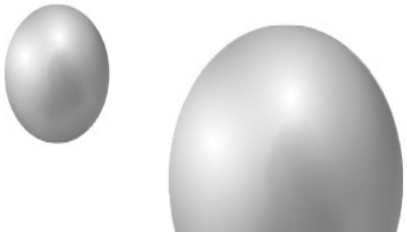
No arquivo student_bloc.dart no subdiretório lib/bloc:

Remover:

```
void firstCall() async {
  bool _init = await _dao.init();
  if ( _init )
    getAll();
  else
    _controller.addError(_dao.errorMsg);
}
```

Alterar:

```
void dispose(){
  _controller.close();
}
```



JWT

Copie o arquivo **home.dart** do subdiretório **lib/pages** da aplicação **persistencia_sqlite**.

Altere no arquivo **home.dart** no subdiretório **lib/pages**:

O nome da aplicação:

```
appBar: AppBar(
  title: Text("JWT"),
```

A chamada:

```
} else {
  _bloc.firstCall();
  return _waitWidget;
}
```

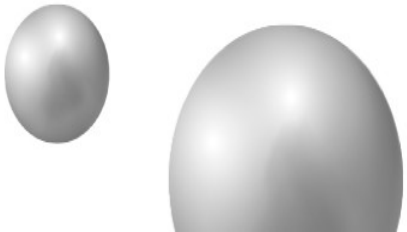
Para:

```
} else {
  _bloc.getAll();
  return _waitWidget;
}
```

JWT

Crie o arquivo showalert.dart no subdiretório lib/utils:

```
import 'package:flutter/material.dart';

void showAlert(BuildContext context, String _msg) async {
  return await showDialog<void>(
    context: context,
    barrierDismissible: false,
    builder: (BuildContext context) {
      return AlertDialog(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(20.0),
        ),
        title: Row(
            children: [
              Icon(Icons.info,color:
Colors.yellow[600],size: 48.0,),
```



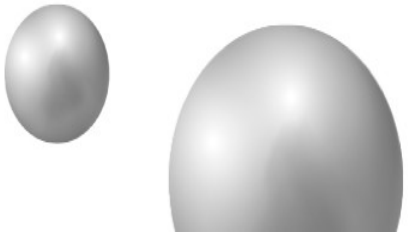
JWT

```
                Expanded(
                  child: Text(
                    'Alerta',
                    style: TextStyle(fontWeight:
FontWeight.bold),
                    textAlign: TextAlign.center,
                  ),
                ),
              ]
          ),
          content: Text(_msg),
          actions: <Widget>[
            FlatButton(
              child: Text('Ok'),
              onPressed: () => Navigator.of(context).pop(),
            ),
          ],
        );
      },
    );
}
```

JWT

Crie o arquivo signin.dart no subdiretório lib/pages:

```
import 'package:flutter/material.dart';
import 'package:jwt_decoder/jwt_decoder.dart';
import '../models/session.dart';
import '../models/user.dart';
import '../database/database.dart';
import 'configuration.dart';
import 'password.dart';
import 'register.dart';
import 'home.dart';
import '../utils/showerror.dart';

// Classe para sigin no Web Service
class SignIn extends StatelessWidget {
  final GlobalKey<FormState> _formStateKey =
GlobalKey<FormState>();
  Session _session = Session.empty();
```



JWT

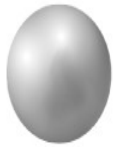
```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Sign In"),
        actions: <Widget>[
          IconButton(
            icon: Icon(Icons.settings),
            tooltip: 'Configuracao',
            onPressed: () async {
              await Navigator.push(context,
                MaterialPageRoute(builder: (context) =>
Configuration()),
              );
              DatabaseHelper.initDb();
            },
          ),
        ],
      ),
```

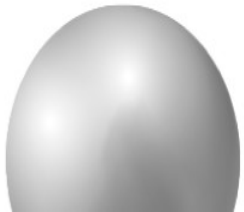
JWT

```
      body: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: <Widget>[
            Form(
              key: _formStateKey,
              autovalidate: true,
              child: Padding(
                padding: EdgeInsets.all(16.0),
                child: Column(
                  children: <Widget>[
                    TextFormField(
                      initialValue: _session.email,
                      decoration: InputDecoration(labelText:
'Email',),
                      validator: (value) =>
_validateEmail(value),
                      onSaved: (value) => _session.email =
value,
                    ),
```

JWT

```
                    TextFormField(
                      initialValue: _session.password,
                      decoration: InputDecoration(labelText:
'Password',),
                      obscureText: true,
                      validator: (value) =>
_validatePassword(value),
                      onSaved: (value) => _session.password
= value,
                    ),
                  ],
                ),
              ),
            ),
            RaisedButton(
              onPressed: () => _signin(context),
              color: Colors.blue[300],
              child: Text("Log In")
            ),
```





JWT

```
            FlatButton(
              onPressed: () => _password(context),
              child: Text("Esqueceu sua Senha?")
            ),
            FlatButton(
              onPressed: () => _register(context),
              child: Text("Novo Usuario")
            )
          ],
        ),
      )
    );
}
```

JWT

```
  // Valida o email
  String _validateEmail(String value) {
    String ret = null;
    value = value.trim();
    if (value.isEmpty)
      ret = "Email e obrigatorio";
    else if (!RegExp(r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*+-/=?
^_`{|}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+").hasMatch(value))
      ret = "Email inválido";
    return ret;
  }

  // Valida a password
  String _validatePassword(String value) {
    String ret = null;
    value = value.trim();
    if (value.isEmpty)
      ret = "Pasword e obrigatoria";
    else if(value.length < 4)
      ret = "Minimo de 4 caracteres";
    return ret;
  }
```

JWT

```
// Autenticar o usuario
void _signin(BuildContext context) async {
  if (_formStateKey.currentState.validate()) {
    _formStateKey.currentState.save();
    await _getJwt(context);
  }
}

// Resetar senha
void _password(BuildContext context) async {
  await Navigator.push(context,
    MaterialPageRoute(builder: (context) => Password()),
  );
}

// Registrar um novo usuario
void _register(BuildContext context) async {
  await Navigator.push(context,
    MaterialPageRoute(builder: (context) => Register()),
  );
}
```

JWT

```
  // Obtem o token de autenticacao
  void _getJwt(BuildContext context) async {
    String _errorMsg = "";
    var _jwt = await DatabaseHelper.signin(_session.toJson()
    ).catchError((_error) {
      _errorMsg = _error;
    });
    if(_jwt != null) {
      print("JWT ${_jwt}");
      Map<String, dynamic> decodedToken =
JwtDecoder.decode(_jwt);
      decodedToken.forEach((k,v) => print('${k}: ${v}'));
      DatabaseHelper.setToken(_jwt);
      Navigator.pushReplacement(context,
        MaterialPageRoute(
          builder: (BuildContext context) => Home()
        )
      );
    }
    else showError(context, _errorMsg);
  }
}
```

JWT

Crie o arquivo password.dart no subdiretório lib/pages:

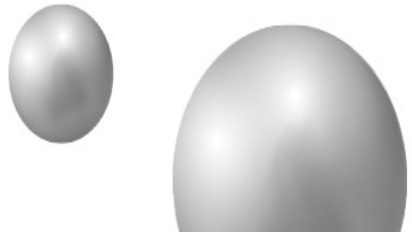
```
import 'package:flutter/material.dart';
import '../models/session.dart';
import '../database/database.dart';
import '../utils/showerror.dart';
import '../utils/showalert.dart';

// Classe para resetar a senha
class Password extends StatelessWidget {
  final GlobalKey<FormState> _formStateKey =
GlobalKey<FormState>();
  Session _session = Session.empty();
  String _password;


  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Password Reset"),),
      body: Padding(
        padding: const EdgeInsets.all(8.0),
```

JWT

```
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: <Widget>[
            Form(key: _formStateKey,
              autovalidate: true,
              child: Padding(
                padding: EdgeInsets.all(16.0),
                child: Column(
                  children: <Widget>[
                    TextFormField(
                      initialValue: _session.email,
                      decoration: InputDecoration(labelText:
'Email',),
                      validator: (value) =>
_validateEmail(value),
                      onSaved: (value) => _session.email =
value,
                    ),
                  ],
                ),
              ),
            ),
```

JWT

```
            RaisedButton(
              onPressed: () => _reset(context),
              color: Colors.blue[300],
              child: Text("Reset")
            ),
          ],
        ),
      )
    );
}
```



JWT

```
  // Valida o email
  String _validateEmail(String value) {
    String ret = null;
    value = value.trim();
    if (value.isEmpty)
      ret = "Email e obrigatorio";
    else if (!RegExp(r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*+-/=?
^_`{|}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+").hasMatch(value))
      ret = "Email inválido";
    return ret;
  }
```

JWT

```
  // Reseta a senha
  void _reset(BuildContext context) async {
    print("Reset");
    String _errorMsg;
    if (_formStateKey.currentState.validate()) {
      _formStateKey.currentState.save();
      var _done = await DatabaseHelper.passwordReset(
          _session.toJson()
      ).catchError((_error) {
        _errorMsg = _error;
      });
      _done ??= false;
      if (_done) {
        await showAlert(context, "Antes de prosseguir,
acesse o e-mail indicado e altere a senha");
        Navigator.pop(context);
      } else
        showError(context, _errorMsg);
    }
  }
}
```

JWT

Crie o arquivo register.dart no subdiretório lib/pages:

```
import 'package:flutter/material.dart';
import '../models/user.dart';
import '../blocs/user_bloc.dart';
import '../utils/showerror.dart';
import '../utils/showalert.dart';

// Classe para registrar um novo usuario
class Register extends StatelessWidget {
  final GlobalKey<FormState> _formStateKey =
GlobalKey<FormState>();
  UserBLoC _bloc = UserBLoC();
  User _user = User.empty();
  String _password;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Registration"),),
      body: Padding(
        padding: const EdgeInsets.all(8.0),
```

JWT

```
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: <Widget>[
            Form(
              key: _formStateKey,
              autovalidate: true,
              child: Padding(
                padding: EdgeInsets.all(16.0),
                child: Column(
                  children: <Widget>[
                    TextFormField(
                      initialValue: _user.name,
                      decoration: InputDecoration(labelText:
'Nome',),
                      validator: (value) =>
_validateName(value),
                      onSaved: (value) => _user.name =
value,
                    ),
```

JWT

```
                    TextFormField(
                      initialValue: _user.email,
                      decoration: InputDecoration(labelText:
'Email',),
                      validator: (value) =>
_validateEmail(value),
                      onSaved: (value) => _user.email =
value,
                    ),
                    TextFormField(
                      initialValue: _user.password,
                      decoration: InputDecoration(labelText:
'Password',),
                      obscureText: true,
                      validator: (value) =>
_validatePassword(value),
                      onSaved: (value) => _user.password =
value,
                    ),
```

JWT

```
                    TextFormField(
                      initialValue: _user.confirm,
                      decoration: InputDecoration(labelText:
'Confirmacao da Passsword',),
                      obscureText: true,
                      validator: (value) =>
_validateConfirm(value),
                      onSaved: (value) => _user.confirm =
value,
                    ), ],
                ),
              ),
            ),
            RaisedButton(
              onPressed: () => _register(context),
              color: Colors.blue[300],
              child: Text("Register")
            ), ],
        ),
      )
    );
  }
```

JWT

```
  // Valida o nome
  String _validateName(String value) {
    String ret = null;
    value = value.trim();
    if (value.isEmpty)
      ret = "Nome e obrigatorio";
    return ret;
  }

  // Valida o email
  String _validateEmail(String value) {
    String ret = null;
    value = value.trim();
    if (value.isEmpty)
      ret = "Email e obrigatorio";
    else if (!RegExp(r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*+-/=?
^_`{|}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+").hasMatch(value))
      ret = "Email inválido";
    return ret;
  }
```

JWT

```
// Valida a password
String _validatePassword(String value) {
  String ret = null;
  value = value.trim();
  _password = value;
  if (value.isEmpty)
    ret = "Pasword e obrigatoria";
  else if(value.length < 4)
    ret = "Minimo de 4 caracteres";
  return ret;
}

// Valida a confirmacao da password
String _validateConfirm(String value) {
  String ret = null;
  value = value.trim();
  if (value.compareTo(_password)!=0)
    ret = "Pasword nao confere";
  return ret;
}
```

JWT

```
  // Cadastra um novo usuario
  void _register(BuildContext context) async {
    print("Register");
    String _errorMsg;
    if (_formStateKey.currentState.validate()) {
      _formStateKey.currentState.save();
      var value = await
_bloc.insert(_user).catchError((_error) {
        _errorMsg = _error;
      });
      value ??= 0;
      if (value > 0) {
        await showAlert(context, "Antes de prosseguir,
acesse o e-mail indicado e faça a verificacao do cadastro");
        Navigator.pop(context);
      } else
        showError(context, _errorMsg);
    }
  }
}
```

JWT

Altere o arquivo database.dart no subdiretório lib/database:

```
import 'package:http/http.dart' as http;
import 'package:http/http.dart';
import 'package:preferences/preferences.dart';
import 'dart:convert';

// TimeOut para as requicoes
const _TIMEOUT = Duration(seconds: 10);

class DatabaseHelper {
  static final DatabaseHelper _instance =
DatabaseHelper.internal();
  factory DatabaseHelper() => _instance;
  DatabaseHelper.internal();

  static String _host;
  static String _token;
```

JWT

```
static _setHost() {
  _host = PrefService.getString('host').trim();
}

static setToken(String _value) {
  _token = _value;
}

static Future<bool> initDb() async {
  _setHost();
  return true;
}

static String getAuthorization() {
  return "Bearer ${_token}";
}
```

JWT

```
  static Future<List<Map>> getAll(String _table) async {
    print("Database getAll");
    List<Map> ret;
    String _errorMsg = "";
    final _response = await http.get(Uri.https("$
{_host}","/${_table}"),
      headers: {"Authorization": getAuthorization()})
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 200) {
        final _parsed = json.decode(_response.body);
        ret = List<Map>();
        for (int i = 0; i < _parsed.length; i++)
          ret.add(_parsed[i]);
      } else _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else return Future<List<Map>>.error(_errorMsg); }
```

JWT

```
  static Future<Map> getByID(String _table, int _id) async {
    print("Database getByID");
    Map ret;
    String _errorMsg = "";
    final _response = await http.get(Uri.https("${_host}","/${_table}/${_id}"),
      headers: {"Authorization": getAuthorization()})
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
    if (_response != null) {
      print("Response ${_response.statusCode} ${_response.body}");
      if (_response.statusCode == 200)
        ret = json.decode(_response.body);
      else
        _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else
      return Future<Map>.error(_errorMsg);
  }
```

JWT

```
  static Future<int> insert(String _table, Map _map) async {
    print("Database insert");
    int ret;
    String _errorMsg = "";
    _map.remove('id');
    final _response = await http.post(Uri.https("$
{_host}","/${_table}"),
      body: json.encode(_map),
      headers: {
        "Accept": "application/json",
        "Content-Type": "application/json",
        "Authorization": getAuthorization()
      },
      encoding: Encoding.getByName("utf-8"))
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
```

JWT

```
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 201)
        ret = 1;
      else
        _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }
```

JWT

```
  static Future<int> update(String _table, Map _map) async {
    print("Database update");
    int ret;
    String _errorMsg = "";
    final _response = await http.put(Uri.https("$
{_host}","/${_table}/${_map['id']}"),
      body: json.encode(_map),
      headers: {
        "Accept": "application/json",
        "Content-Type": "application/json",
        "Authorization": getAuthorization()
      },
      encoding: Encoding.getByName("utf-8"))
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
```

JWT

```
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 200)
          ret = 1;
      else
        _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }
```

JWT

```
  static Future<int> delete(String _table, int _id) async {
    print("Database delete");
    int ret;
    String _errorMsg = "";
    final _response = await http.delete(Uri.https("$
{_host}","/${_table}/${_id}"),
      headers: {"Authorization": getAuthorization()})
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 200)
          ret = 1;
      else
        _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }
```

JWT

```
  static Future<String> signin(Map _map) async {
    print("Database signin");
    String ret;
    String _errorMsg = "";
    final _response = await http.post(Uri.https("$
{_host}","/auth"),
      body: json.encode(_map),
      headers: {
        "Accept": "application/json",
        "Content-Type": "application/json"
      },
      encoding: Encoding.getByName("utf-8"))
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
```

JWT

```
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 200) {
        final _parsed = json.decode(_response.body);
        if (_parsed["tipo"].compareTo("Bearer")==0)
          ret = _parsed["token"];
        else
          _errorMsg = "Falha na autenticacao";
      }
      else
        _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else
      return Future<String>.error(_errorMsg);
  }
```

JWT

```
  static Future<bool> passwordReset(Map _map) async {
    print("Password Reset");
    bool ret;
    String _errorMsg = "";
    final _response = await http.post(Uri.https("${_host}","/password/forgot"),
      body: json.encode(_map),
      headers: {
        "Accept": "application/json",
        "Content-Type": "application/json"
      },
      encoding: Encoding.getByName("utf-8"))
      .timeout(_TIMEOUT)
      .catchError((_error) {_errorMsg = "${_error}";});
```

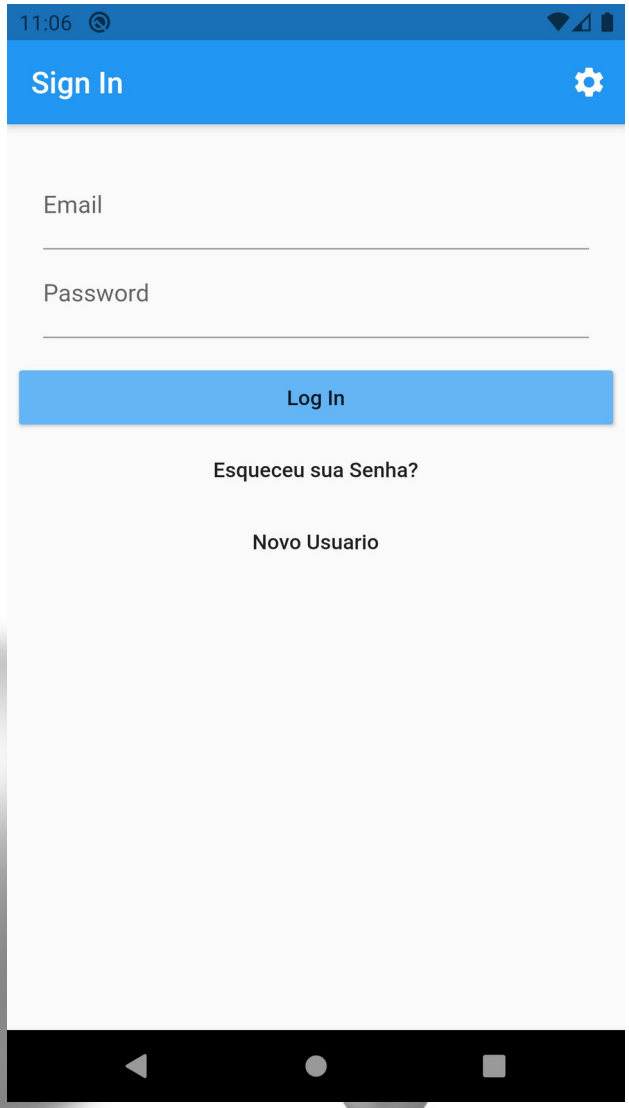
JWT

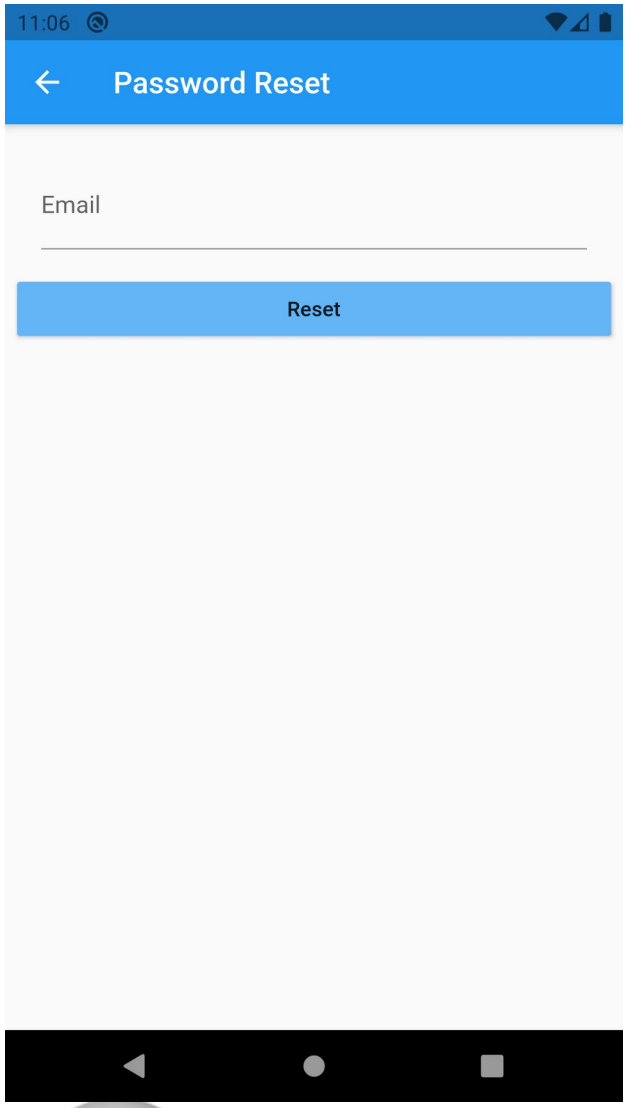
```
    if (_response != null) {
      print("Response ${_response.statusCode} $
{_response.body}");
      if (_response.statusCode == 200)
        ret = true;
      else
        _errorMsg = responseError(_response, null);
    }
    if (ret != null)
      return ret;
    else
      return Future<bool>.error(_errorMsg);
  }
```

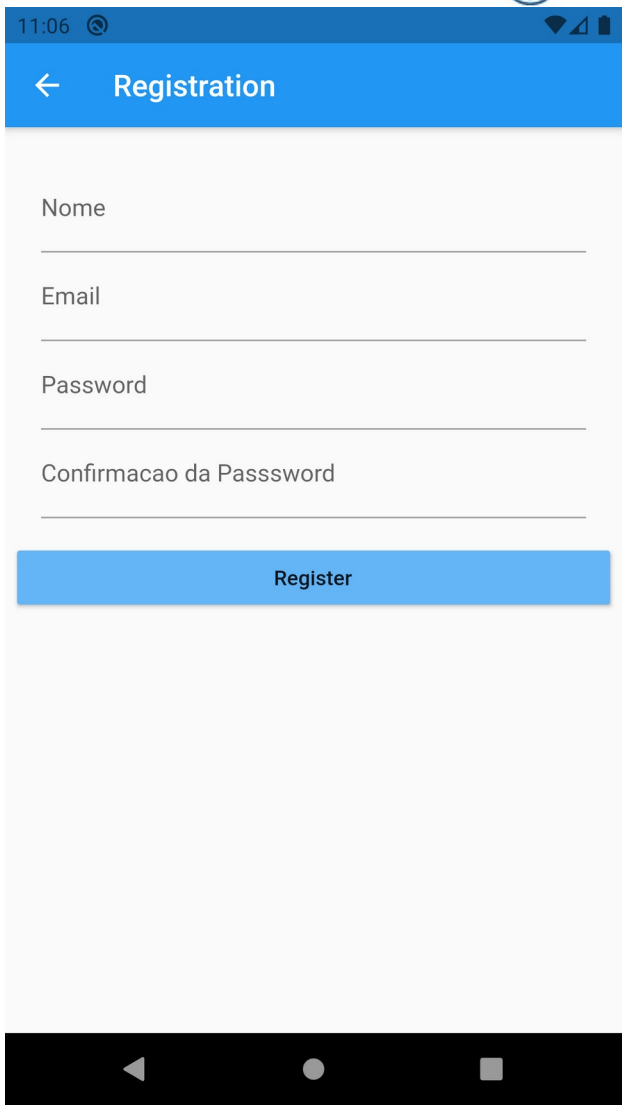
JWT

```
  static String responseError(var _response, String _body) {
    String ret = 'Bad status code ${_response.statusCode}.';
    _body ??= _response.body;
    if (_body != null )
      ret = ret+'\n${_body}';
    return ret;
  }
}
```

JWT


11:06
Sign In
Email
Password
Log In
Esqueceu sua Senha?
Novo Usuario


11:06
Password Reset
Email
Reset


11:06
Registration
Nome
Email
Password
Confirmacao da Passsword
Register

JWT