JWT
JSON Web Token

Prof. Me. Hélio Esperidião

Web Services

- Os Web Services são componentes que possibilitam que diferentes aplicações se comuniquem, enviando e recebendo dados de forma padronizada.
- Cada aplicação pode utilizar sua própria linguagem de programação e efetuar comunicação por meio de um formato intermediário universal, como XML, JSON ou CSV.

Token – contexto de autenticação

- É uma string de caracteres que, caso cliente e servidor estejam sob **HTTPS**, permite que somente o servidor que conhece o ‘segredo’ possa validar o conteúdo do token e assim confirmar a autenticidade do cliente.
- O token não é "criptografado", mas sim "assinado".
- Essa assinatura é verificada usando um "secret" (segredo), o que garante que somente aqueles que possuem o segredo correto possam validar a assinatura.
- Essa abordagem impede que atacantes tenham a capacidade de "criar" tokens fraudulentos por conta própria.

O que são tokens?

- O token é uma assinatura digital ou uma chave.

IETF - Internet Engineering Task Force

- Grupo internacional aberto composto de técnicos, fabricantes, agências, fornecedores e pesquisadores que desenvolvem os padrões da Internet.
- Isso é feito em parceria com a World Wide Web Consortium e ISO/ IEC
- A principal missão do IETF é identificar e propor soluções para o uso da Internet e padronizações das redes baseadas em IP

RFC - Request for Comments

- Os Request for Comments (RFC) são documentos usados pela comunidade online há décadas para definir os padrões da web e compartilhar informações técnicas.
- Atualmente, os RFCs são gerenciados pela IETF (Internet Engineering Task Force).
- Os RFCs geralmente são identificados por números, como por exemplo, o RFC 1945
  - Regulação do protocolo http

RFCs

Protocolo	RFC
ARP	826
DHCP	2131
DNS	1034 e 1035
FTP	959
HTTP	1945
ICMP	792
IP	791
IPv6	2460
MD5	1321

RFCs

NAT	3022
POP3	1939
SMTP	5321
SSH	4251
TCP	793
UDP	768

JSON Web Tokens (JWT)

- JSON Web Token, é um padrão aberto e bem documentado, definido pela **RFC-7519**, que estabelece uma forma simples, compacta e segura de transmitir e armazenar objetos JSON entre diferentes aplicações.
  - Veja o padrão: https://datatracker.ietf.org/doc/html/rfc7519
- Sua ampla utilização ocorre principalmente na validação de serviços em Web Services.
- A razão para isso é que os dados contidos em um token podem ser verificados a qualquer momento devido à sua assinatura digital.

Quando e onde eu posso usar um JWT?

- Você pode usar, por exemplo, em um cenário de autorização. Depois que o usuário estiver conectado, é possível que cada solicitação verifique se está incluso o JWT, permitindo que o usuário acesse rotas, serviços e outros recursos.
- Outro cenário de utilização de JWTs são as trocas de informações pois, como eles são assinados, é possível ter certeza de que os remetentes são quem dizem ser quem são.

- Podemos identificar se o conteúdo da assinatura foi alterado ou não devido à composição de um JWT.

Terminologias

- JOSE (JSON Object Signing and Encryption) significa Assinatura e Criptografia de Objetos JSON.
- O JWT é uma das especificações dessa família e representa o formato de token.
- Além do JWT, existem outras especificações relacionadas, como:
  - JWT (JSON Web Tokens): representa o token propriamente dito;
  - JWS (JSON Web Signature): representa a assinatura do token;
  - JWE (JSON Web Encryption): representa a assinatura para criptografia do token;
  - JWK (JSON Web Keys): representa as chaves para a assinatura;
  - JWA (JSON Web Algorithms): representa os algoritmos para assinatura do token.

Componentes básicos de um JSON Web Token

- Um JWT possui uma estrutura básica composta por:
  - header
  - payload
  - signature.
- Essas três partes são separadas por pontos (.).
- É Representado por: header.payload.signature.

Exemplo de um token

- Para realizar operações no sistema o usuário precisa enviar um token valido.
  - header
  - payload
  - signature
- Token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkjDqWxpbyBFc3BlcmlkacOjbyIsImVtYWlsIjoiaGVsaW9lc3BlcmlkaWFvQGdtYWlsLmNvbSIsImlhdCI6MTUxNjIzOTAyMn0.cQhpa4tI02HXOQaCYlWVHkMQGjtaZwSYDdN8c4fXfgo

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)
```

Estrutura

- A estrutura de um JWT é composta por 3 partes:
  - Header,
  - Payload e
  - Signature;
- As partes são separadas por um ponto (.) e cada parte é individualmente encodada em Base64Url.

Base64Url/Base64

- Base64Url é uma variação do esquema de codificação Base64, projetada para ser segura e amigável para ser usada em URLs/URIs e formatos que aceitam apenas caracteres seguros.
- O esquema Base64 é comumente usado, mas ele pode conter alguns caracteres que não são seguros para URLs, como o sinal de igual (=) ou o símbolo de adição (+), o que pode causar problemas em alguns contextos.
- O Base64Url substitui os caracteres "padrão" do esquema Base64 que podem causar problemas em URLs. Especificamente, ele substitui o sinal de igual (=) por underline (_), o símbolo de adição (+) por hífen (-) e remove os caracteres de barra (/)

Header

- O header especifica o algoritmo que foi utilizado para gerar a assinatura do Token.

Header

- O cabeçalho (headers) do token é onde passamos basicamente duas informações: o "alg", que informa qual algoritmo é usado para criar a assinatura, e o "typ", que indica o tipo de token utilizado.

- Exemplo:

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

Payload

- O Payload é uma estrutura em formato JSON que contém as reivindicações (claims) relacionadas à entidade em questão.
- É utilizado para armazenar dados do usuário autenticado durante processos de autenticação.
- Essas reivindicações podem ser classificadas em três tipos:
  - Reserved claims (reivindicações reservadas),
  - Public claims (reivindicações públicas)
  - Private claims (reivindicações privadas).

- As reivindicações reservadas contêm informações padrão e bem definidas pelo padrão JWT (JSON Web Token) e possuem um significado pré-definido e são amplamente reconhecidas.
- As reivindicações públicas referem-se a informações que são compartilhadas publicamente e podem ser utilizadas entre diferentes partes sem preocupações com confidencialidade.
- Por fim, as reivindicações privadas são informações específicas de uma aplicação ou uso particular, destinadas apenas para partes específicas que acordam seu significado.

Payload

- O payload é o componente onde encontramos os dados relacionados à autenticação, email, `nome` por exemplo.


```
{
  "sub": "1234567890",
  "name": "Hélio Esperidião",
  "email": "helioesperidiao@gmail.com",
  "iat": 1516239022
}
```

Payload:
claims RFC 7519

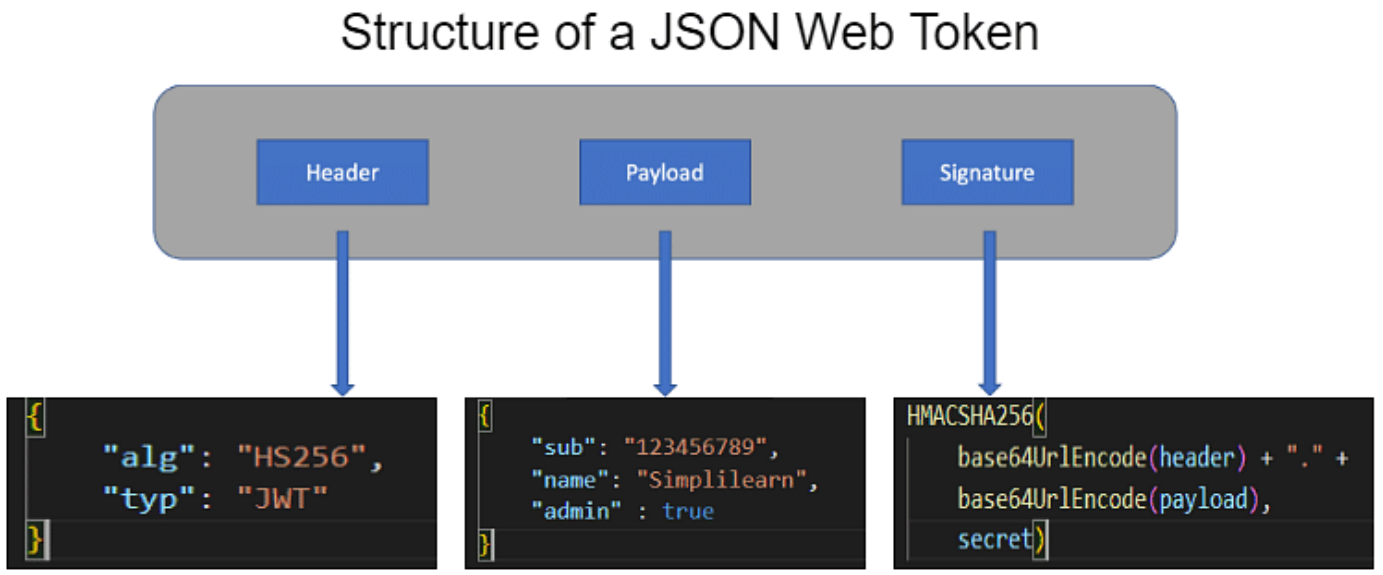
- “iss” – Issuer (Emissor) :
  - Responsável pela emissão do token.
- “sub” – Subject (Sujeito)
  - A quem pertence o token
- “aud” – Audience (Destinatário)
  - Aplicação que irá utilizar o token
- “exp” – Expiration Time (Tempo de expiração)
  - Data de validade do token
- “nbf” – Not Before (Não antes)
  - Data de início de validade do token, antes dela o token não deve ser aceito
- “iat” – Issued At (Emitido em)
  - Data de emissão do token
- “jti” – JWT ID
  - Identificador único do token

resumo

- Em resumo, o Payload em um token JWT é o local onde as informações relevantes (reivindicações) são armazenadas, e sua classificação em reivindicações reservadas, públicas ou privadas permite uma organização e compartilhamento eficiente de dados entre diferentes aplicações e serviços.

Signature

- A assinatura do token (signature) é composta pela codificação do header e do payload somada a uma chave secreta e é gerada pelo algoritmo especificado no cabeçalho.



HS256

- O algoritmo HS256 (HMAC-SHA256) é um dos algoritmos comumente utilizados na criação da assinatura do JSON Web Token (JWT).
- Ele faz parte dos algoritmos de assinatura HMAC (Hash-based Message Authentication Code) e utiliza a função de hash SHA-256.

- Uma chave secreta é usada para gerar uma assinatura digital.
- Essa assinatura é calculada aplicando o algoritmo HMAC-SHA256 ao Header e ao payload do JWT, juntamente com a chave secreta. O resultado é uma sequência de bytes que representa a assinatura.

Algoritmos de criptografia.

- HS256 (HMAC-SHA256):
  - O algoritmo HS256 utiliza HMAC (Hash-based Message Authentication Code) com SHA-256 como função hash. Isso significa que a chave secreta compartilhada entre o emissor e o receptor é usada para assinar e verificar a integridade do token. A mesma chave é usada para gerar e validar a assinatura.

- RS256 (RSA-SHA256):
  - O algoritmo RS256 utiliza RSA (Rivest-Shamir-Adleman) com SHA-256 como função hash. Nesse caso, o emissor possui um par de chaves: uma chave privada para assinar o token e uma chave pública para que os receptores possam verificar a assinatura. A chave privada é usada para gerar a assinatura, enquanto a chave pública é usada para validar a assinatura.

- ES256 (ECDSA-SHA256):
  - O algoritmo ES256 utiliza ECDSA (Elliptic Curve Digital Signature Algorithm) com SHA-256 como função hash. Esse algoritmo também utiliza um par de chaves, semelhante ao RS256, mas baseado em curvas elípticas. O emissor usa uma chave privada para assinar o token, e os receptores usam a chave pública correspondente para verificar a autenticidade da assinatura.

Os algoritmos de criptografia.

- Os algoritmos de criptografia segura são conhecidos.
  - Tanto desenvolvedores quanto “hackers” conhecem muito bem os algoritmos.
- A segurança não está na capacidade de criar um algoritmo que ninguém conheça.
- A segurança está na quantidade de esforço computacional para conseguir “quebrar” a chave.
- Construir um algoritmo de criptografia sem conhecimento adequando é uma grave falha de segurança.

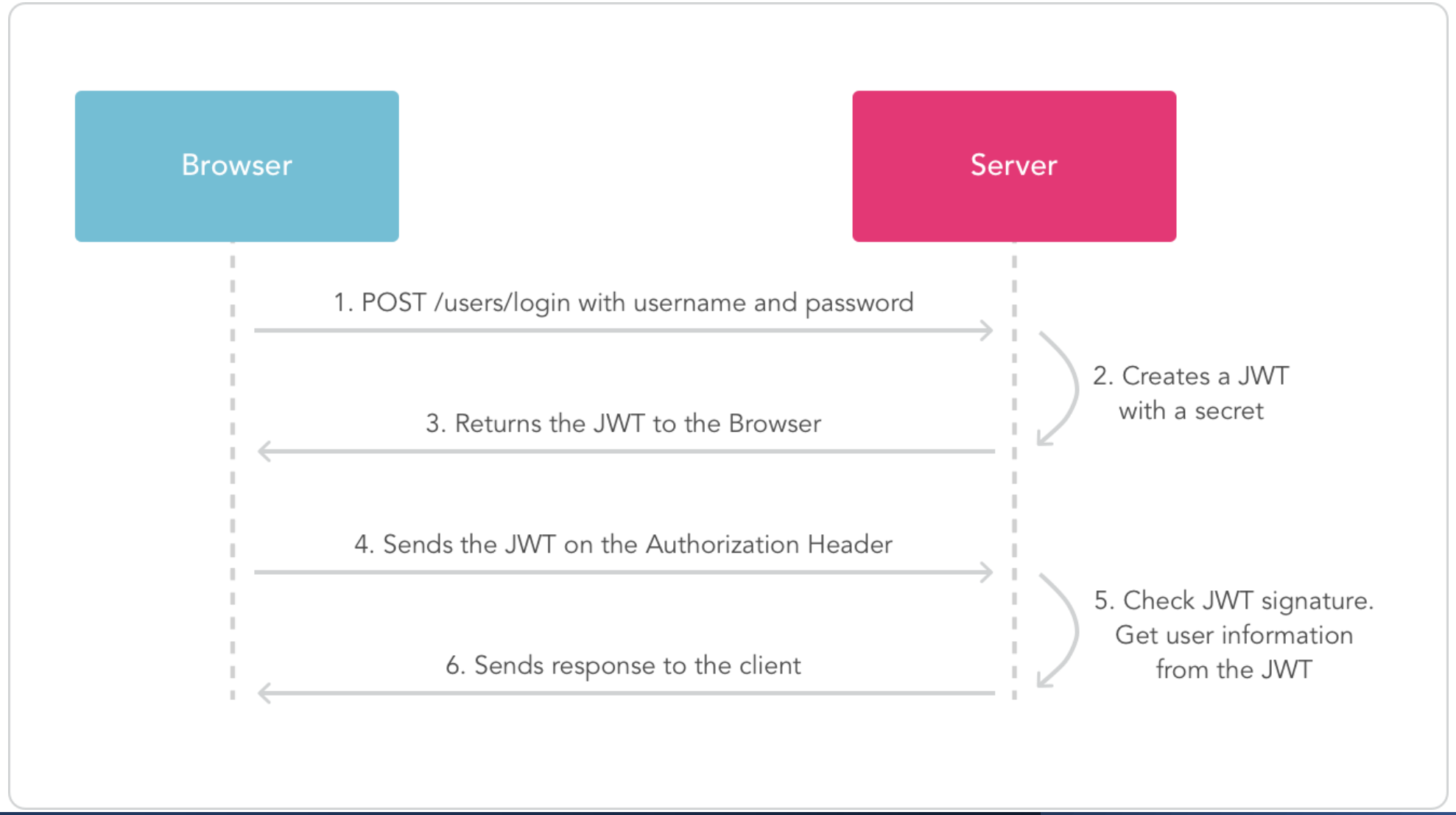
Segurança em ataque de força bruta

- Em termos de força bruta, quebrar o HS256 por tentativa e erro envolveria tentar todas as possíveis combinações de chave secreta até encontrar uma que resulte na mesma assinatura do token. O SHA-256 tem um espaço de saída de 256 bits, o que significa que há 2^256 possíveis valores para o resultado do hash.

- Se considerarmos um cenário hipotético com um espaço de chaves secreto de 128 bits (ou seja, 2^128 possíveis chaves), o número de tentativas necessárias para forçar a quebra seria, em média, metade do espaço de chaves, o que seria 2^127 tentativas.

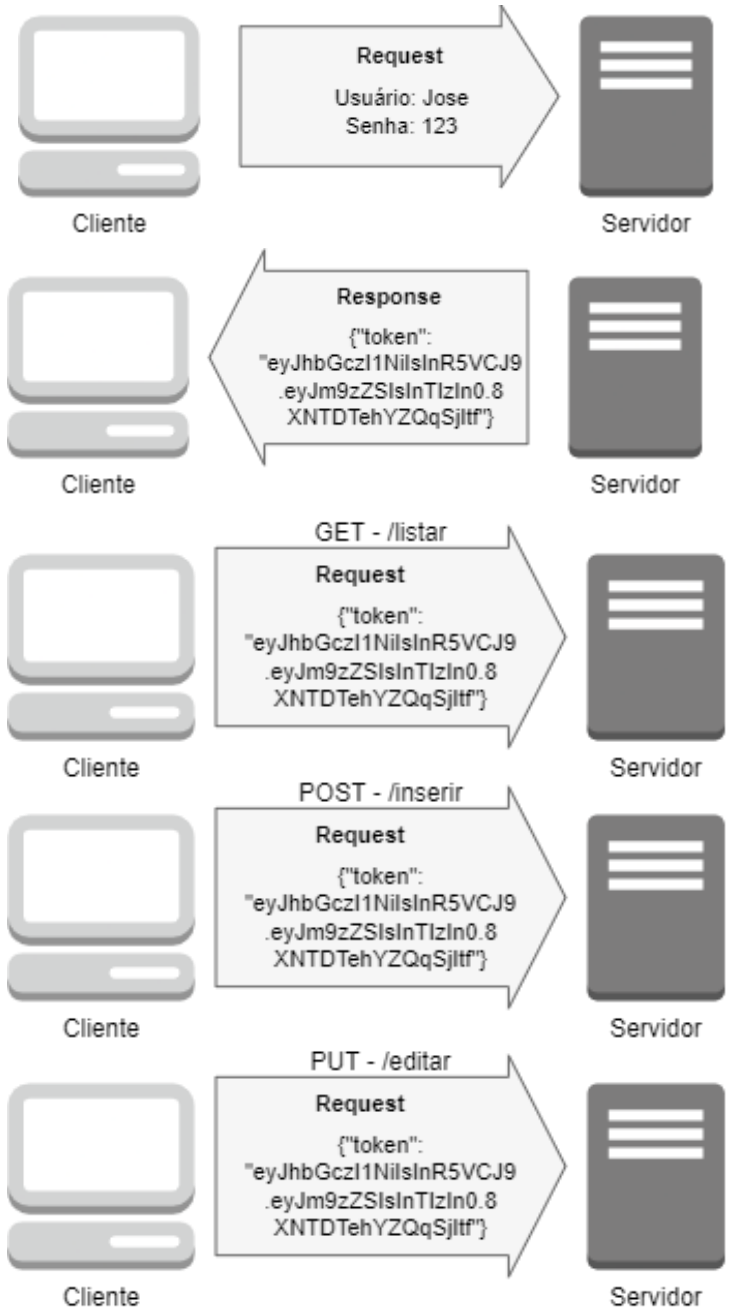
- No entanto, é importante notar que um ataque de força bruta nesse nível é extremamente inviável, mesmo para computadores superpoderosos. Atualmente, não existem recursos computacionais suficientes no mundo para realizar um ataque de força bruta bem-sucedido em um algoritmo com chave de 128 bits ou mais. Além disso, a computação distribuída também enfrentaria desafios significativos nesse contexto.

- Portanto, o HS256 é considerado seguro contra ataques de força bruta, desde que a chave secreta seja forte e suficientemente longa, o que geralmente é recomendado utilizar chaves com pelo menos 256 bits de tamanho para garantir a segurança do algoritmo. A principal preocupação em relação à segurança do HS256 está relacionada à proteção adequada da chave secreta e à prevenção de ataques como vazamento de chaves ou ataques de injeção de código que possam comprometer a integridade do sistema.



Arquitetura

Arquitetura



Arquitetura

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkjDq
WxpbyBFc3BlcmlkacOjbyIsImVtYWlsIjoiaGVs
aW9lc3BlcmlkaWFvQGdtYWlsLmNvbSIsImlhdCI
6MTUxNjIzOTAyMn0.cQhpa4tI02HXOQaCYlWVHk
MQGjtaZwSYDdN8c4fXfgo
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


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


PAYLOAD: DATA


```
{
  "sub": "1234567890",
  "name": "Hélio Esperidião",
  "email": "helioesperidiao@gmail.com",
  "iat": 1516239022
}
```


VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  BATATA-FRITA-123
) ☐ secret base64 encoded
```

Qual vantagem na utilização do JWT? *Segurança*

- O JWT é totalmente autossuficiente, pois inclui uma assinatura que pode ser utilizada para verificar a integridade do token.
- Essa característica é fundamental para garantir a segurança e a confiabilidade do JWT.
- Se porventura o conteúdo de um JWT for modificado ou adulterado, a sua assinatura torna-se inválida, o que resulta na rejeição do token.
- Qualquer tentativa de utilizar um JWT alterado para acessar recursos protegidos será impedida, garantindo que apenas tokens válidos e íntegros possam ser utilizados com sucesso no processo de autenticação e autorização.

Qual vantagem na utilização do JWT? *Portabilidade*

- Devido ao formato compacto e auto suficiente, o JWT pode ser facilmente transportado e utilizado entre diversos sistemas e tecnologias. Sua estrutura é projetada para ser independente, o que significa que ele pode ser compartilhado e interpretado sem a necessidade de dependências externas ou recursos adicionais.
- É uma escolha popular para a comunicação e integração entre diferentes aplicações e plataformas. Ele pode ser transmitido através de URLs, inserido em cabeçalhos HTTP, ou mesmo armazenado em cookies.
- A facilidade de decodificação e validação do JWT permite que aplicativos escritos em diferentes linguagens de programação ou que operem em diferentes ambientes possam facilmente interpretar o token e utilizar as informações nele contidas. **(Utiliza pouco trabalho computacional)**

Qual vantagem na utilização do JWT? **Escalabilidade**

- Com o uso do JWT, é possível que o servidor de autenticação delegue a responsabilidade da autenticação para o cliente. Essa abordagem permite que o servidor de autenticação se concentre principalmente no gerenciamento de identidades, enquanto a tarefa específica de autenticação é realizada pelos próprios clientes.
- Essa delegação de autoridade torna o processo de autenticação mais distribuído e eficiente, pois os clientes podem validar as credenciais dos usuários localmente, sem a necessidade de constantes interações com o servidor de autenticação. Ao receber um JWT válido, o servidor pode confiar nas informações contidas no token, sem a necessidade de consultar uma base de dados ou executar etapas adicionais de autenticação.
- Essa abordagem descentralizada traz vantagens significativas em termos de escalabilidade e redução da carga no servidor de autenticação central, além de proporcionar uma experiência mais ágil para os usuários finais, já que a autenticação pode ser realizada localmente e com menos latência. No entanto, é essencial que a segurança do sistema seja devidamente implementada, garantindo que apenas clientes confiáveis e autorizados possam gerar e utilizar os tokens JWT para autenticação.

Qual vantagem na utilização do JWT? Desempenho

- Devido ao seu formato compacto, o JWT pode ser transmitido de forma rápida e eficiente entre o cliente e o servidor, o que resulta em uma melhoria significativa no desempenho geral do sistema.
- A leveza do JWT, em comparação com outros formatos mais pesados de token ou autenticação, reduz a sobrecarga de comunicação durante o processo de troca de informações entre as partes. Essa economia de dados é particularmente benéfica em cenários de alta demanda e largura de banda limitada, como em aplicações web, serviços de API e ambientes móveis.
- O rápido envio e processamento do JWT ajudam a minimizar o tempo de resposta e o tempo de carregamento, proporcionando uma experiência mais ágil e responsiva para os usuários. Além disso, a agilidade do JWT é especialmente valiosa em comunicações assíncronas, como no uso de WebSockets ou em situações que envolvam várias trocas de tokens em um curto período de tempo.
- Em resumo, a natureza compacta do JWT é uma vantagem significativa para a eficiência e o desempenho do sistema, permitindo uma comunicação mais rápida e uma experiência mais fluída para os usuários finais.

Qual vantagem na utilização do JWT? **Flexibilidade**

- O JWT, por ser baseado em JSON, oferece grande flexibilidade e personalização para atender às necessidades específicas de um aplicativo ou sistema. A possibilidade de adicionar informações personalizadas ao token, além das reivindicações padrão, é especialmente útil para fins de autorização, permitindo a inclusão de dados específicos do usuário ou contexto da aplicação.
- Essa característica, aliada à segurança, portabilidade, escalabilidade, desempenho e flexibilidade do JWT, faz dele um padrão aberto amplamente adotado para autenticação e autorização em aplicativos web e APIs. Sua capacidade de transmitir informações de maneira segura e compacta, além de possibilitar uma fácil integração com diferentes tecnologias, torna o JWT uma escolha popular e confiável.
- Compreender as partes e o funcionamento do JWT é fundamental para utilizar esse padrão de forma adequada e eficaz em sistemas e aplicações, garantindo a segurança e o bom desempenho da autenticação e autorização em ambientes distribuídos.

Quais as desvantagens?

- A segurança do JWT está baseada na sua chave secreta.
- Se por qualquer motivo essa chave for exposta, alguém mal intencionado poderá acessar todos os dados da aplicação.

Quais as desvantagens?

- Como todas as requisições devem conter o token de validação, isso pode causar um aumento desnecessário no tamanho dos pacotes transmitidos.

Quais as desvantagens?

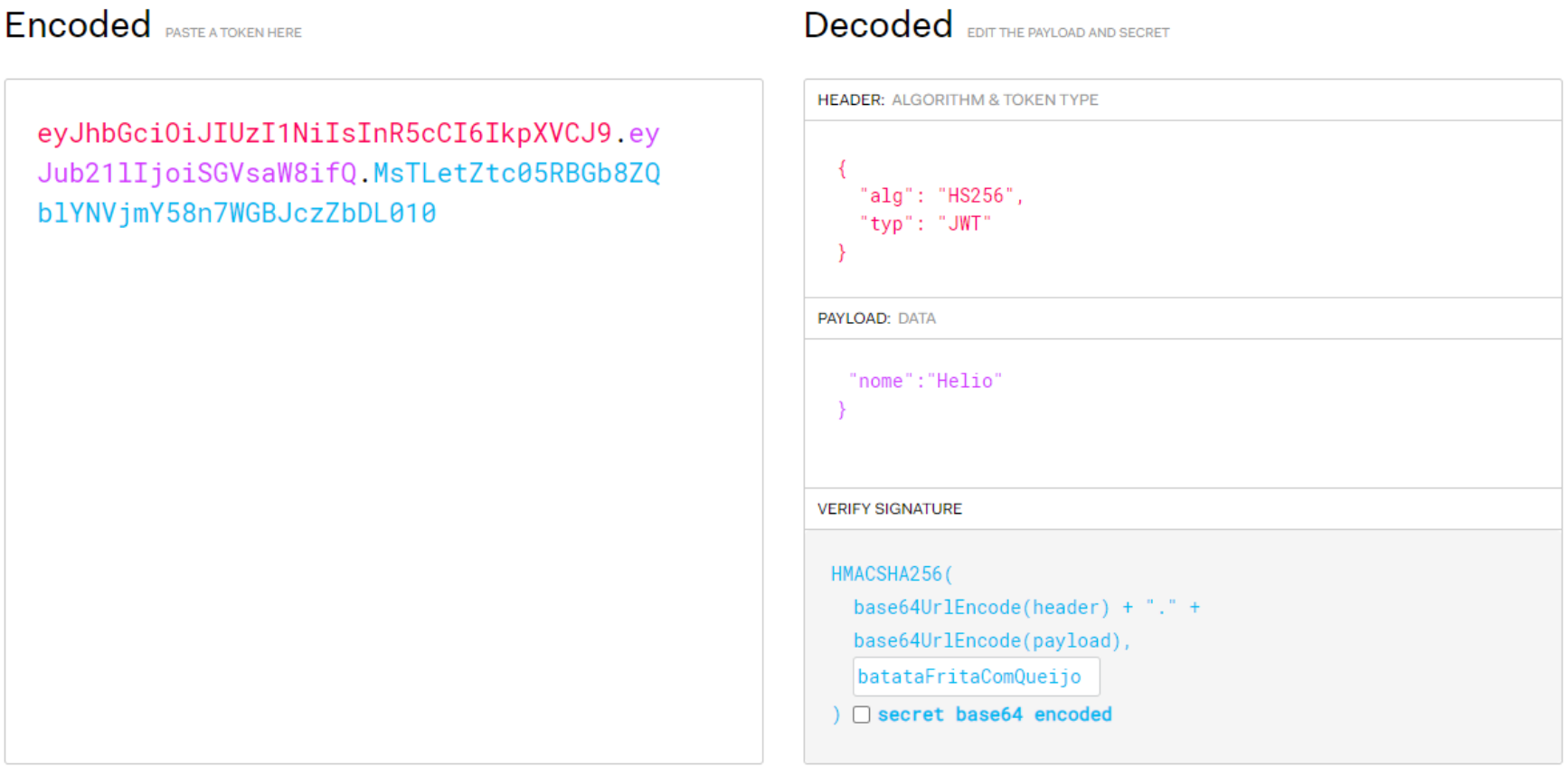
- Como não existe uma sessão, não é possível realizar um logout de outro dispositivo.

https://jwt.io/

- Pode ser utilizado para testes de criação e verificação de chaves.

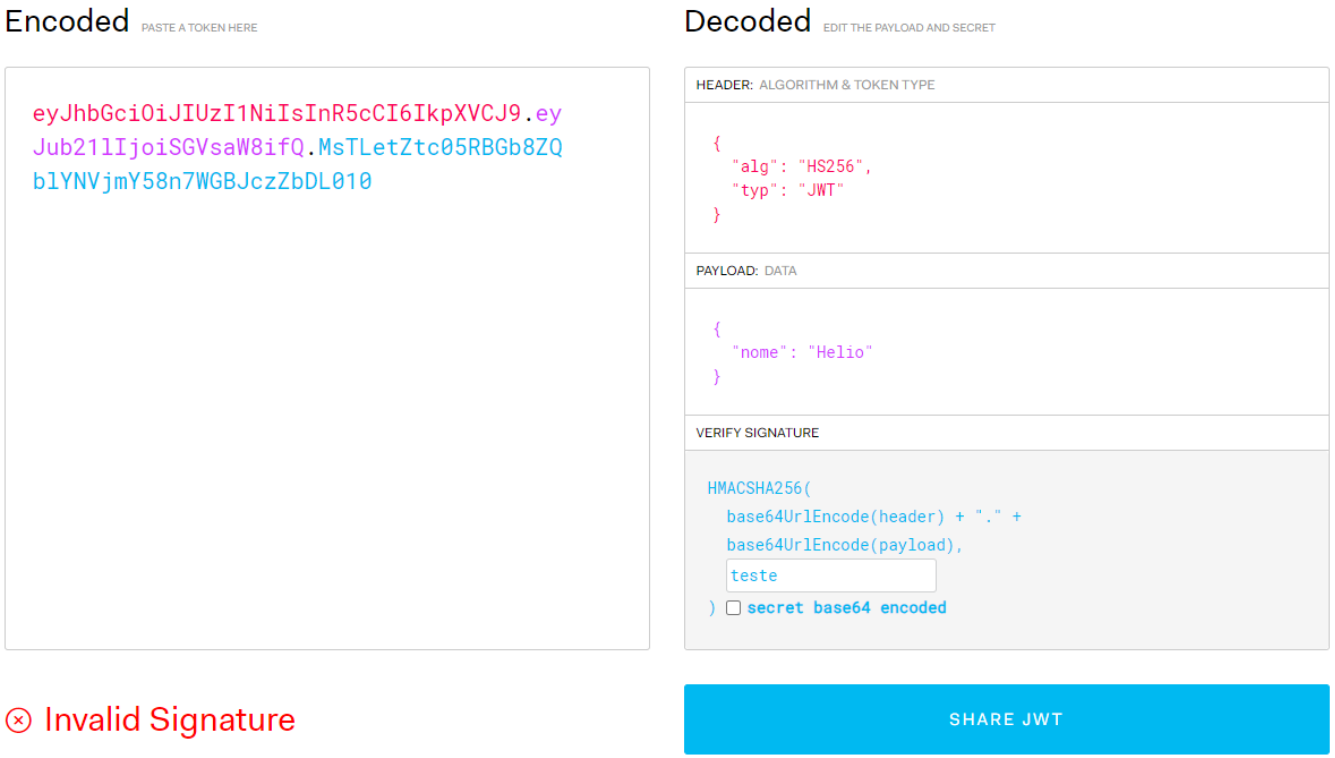
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lIjoiSGVsaW8ifQ.MsTLetZtc05RBGb8ZQblYNVjmY58n7WGBJczZbDL010

batataFritaComQueijo



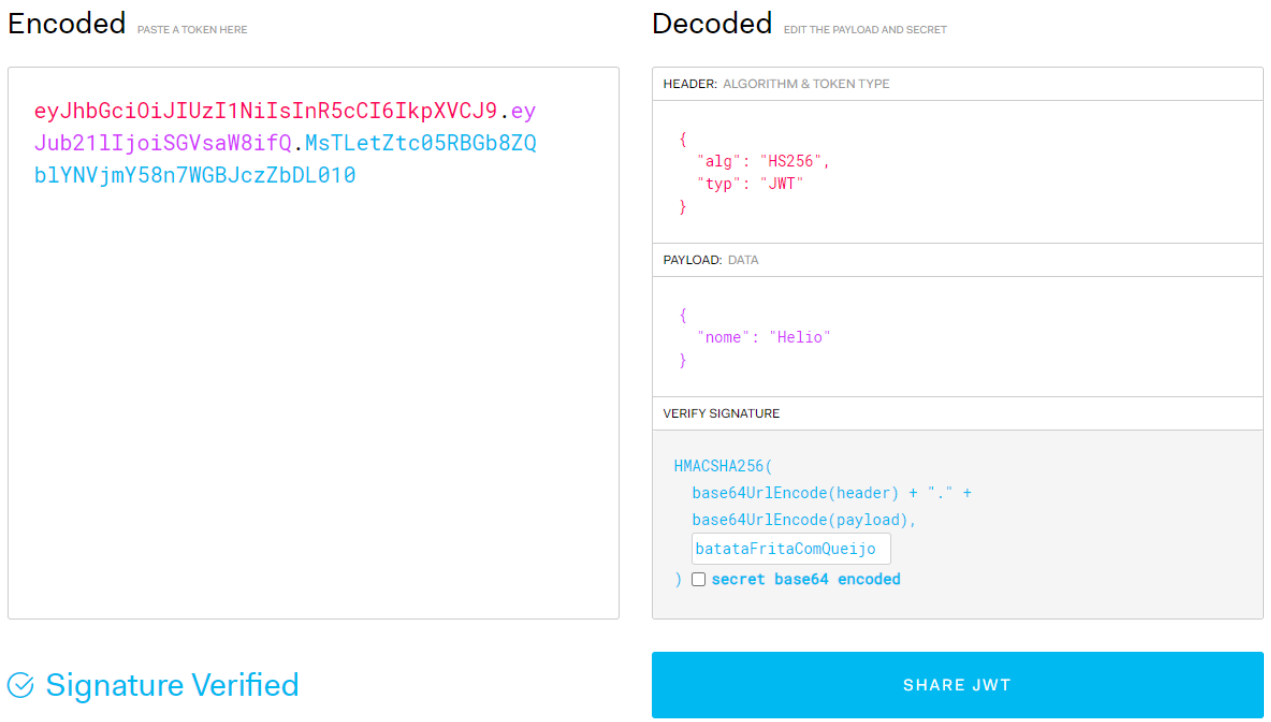
Observe

- Foi utilizado o mesmo token, porem a chave de assinatura foi modificada.
- É possível recuperar e ver os dados.
- Como a chave foi modificada a assinatura é inválida.
- Não é possível garantir que os dados não foram modificados.



Observe

- Com a chave correta a assinatura é válida;



Tente você

- Abra: https://jwt.io/
- Insira o token:
  - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lIjoiSGVsaW8ifQ.MsTLetZtc05RBGb8ZQblYNVjmY58n7WGBJczZbDL010.
- Insira a chave:
    - batataFritaComQueijo
- Tente outras chaves:
  - Teste007, batata com bacon.
- Verifique que é possível visualizar os dados do payload, mas não é possível verificar a assinatura.
  - A verificação da assinatura garante que os dados do payload não foram alterados.

Implementação

- Vamos utilizar uma implementação que é amplamente utilizada no mercado:
  - https://github.com/firebase/php-jwt

Esquema e arquivos

- Modelo
  - **jwt** – pacote de classes https://github.com/firebase/php-jwt
  - **Banco.php** – Classe de conexão com banco de dados;
  - **Route**r – Classe de roteamento.
  - **TokenJWT.php** – classe desenvolvida pelo prof. Hélio para facilitar o uso do pacote jwt
  - **Usuário.php** – classe para manipular ações do usuário.
- **.htaccess** – arquivo de configuração do apache.
- **Index.php** – Arquivo de gestão de rotas.

TokenJWT.php

- Nessa classe modifique apenas os atributos necessários para a sua aplicação.
- Não modificar essa classe se não estiver certo do que está fazendo.

TokenJWT.php
Atributos

```
//chave de criptografia, defina uma chave forte e a mantenha segura.
private $key = "x9S4q0v+V0IjvHkG20uAxaHx1ijj+q1HWjHKv+ohxp/oK+77qyXkVj/l4QYHHTF3";
private $alg = 'HS256';     //algoritmo de criptografia
private $iss = 'http://localhost';  //emissor do token
private $aud = 'http://localhost'; //destinatário do token
private $payload = array();
private $erro;
//tempo de validade do token
private $duracaoToken=1800; //1800 segundos = 30 min
```

TokenJWT.php gerarToken($jsonDados)

```
public function gerarToken($jsonDados){
    $headers = [
        'alg' => $this->alg,
        'typ' => 'JWT'
    ];
    $payload = [
        'iss' => $this->iss,  // emissor do token
        'aud' => $this->aud,  // destinatário do token
        'iat' => time(),          // data de criação do tolen
        'exp' => time() + $this->duracaoToken,   //data de expiracao 1800 segundos do atual, 30 minutos
        'dados' => ($jsonDados),
    ];
    //utiliza a biblioteca do firebase para gerar o token com os parâmetros
    $token = JWT::encode($payload, $this->key, $this->alg, null, $headers);
    return $token;
  }
```

TokenJWT.php validarToken($headers)

```
public function validarToken($headers){
    $token =  TokenJWT::getAuthorizationToken($headers);
    if (isset($token)) {
       if ($token == "") {
          return false;
       } else {
          try {
             $decoded = JWT::decode($token,  new key($this->key, $this->alg));
             $this->setPayload(($decoded));
             return true;
          }
          catch (SignatureInvalidException $e) { return false;}
          catch (DomainException $e) { return false;}
          catch (InvalidArgumentException $e) { return false;}
          catch (ExpiredException $e) { return false; }
          catch (UnexpectedValueException $e) { return false;}
          catch (Exception $e) { return false; }
       }
    }
    return false;
 }
```

Usuarios.php

```
<?php
include "Banco.php";
class Usuario implements JsonSerializable {
    private $idUsuario;
    private $nome;
    private $email;
    private $senha;
    private $banco;
```

jsonSerialize()

```
/*
    Quando for chamado o json_decode($objUsuario)
    o método  jsonSerialize() será chamado
    Um json no formato definido em  jsonSerialize()
    será criado
 */
 public function jsonSerialize()
 {
    $json = array();
    $json['idUsuario'] = $this->getIdUsuario();
    $json['nome'] = $this->getNome();
    $json['email'] = $this->getEmail();
    return $json;
 }
```

create()

```
*    método chamado pelo arquivo index.php, quando é recebido um post para cadastro de novo usuário. */
  public function create() {
     //Instancia a classe banco
     $this->banco = new Banco();
     //cria um hash da senha usando o algoritmo md5
     $this->senha = md5($this->senha);
     //prepara a instrução de insert no banco
     $stmt = $this->banco->getConexao()->prepare("insert into usuario (nome, email, senha) values (?,?,?)");
     //substitui os "s" pelos interrogações e valore de variáveis.
     $stmt->bind_param("sss", $this->nome, $this->email, $this->senha);
     //execura a instrução no banco de dados.
     $resposta = $stmt->execute();
     //retorna o id do registro que foi inserido no banco.
     $idCadastrado = $this->banco->getConexao()->insert_id;
     //passa para a instancia da classe o id que foi cadastrado
     $this->setIdUsuario($idCadastrado);
     return $resposta;
  }
```

usuarioExiste()

```
public function usuarioExiste(){
    $this->banco = new Banco();  //faz uma instância de banco
    //prepara a instrução de select no banco
    $sql = "select count(*) as qtd from usuario where email = ?";
    $stmt = $this->banco->getConexao()->prepare($sql);
    //víncula os parametros substituindo os "?" pelos valores correspondentes.
    $stmt->bind_param("s", $this->email);
    $stmt->execute();  //executa a  instrução sql no sgbd
    //recupera os dados referentes a execução do sql
    $resultado = $stmt->get_result();
    //estrutura de repetição que passa por todos os dados
    while ($linha = $resultado->fetch_object()) {
      //verifica se qtd é igual a 1, igual a 1 significa que existe 1 usuário, com o email e senha fornecido.
      if ($linha->qtd == 1) {
        return true;
      }
    }
    return false;
  }
```

delete()

```
/*  método chamado pelo arquivo index.php
    quando é recebido um Delete para exclusão
    de usuário.*/
  public function delete() {
    $this->banco = new Banco();  //Instancia a classe banco
    //prepara a instrução de delete no banco
    $stmt = $this->banco->getConexao()->prepare("delete from usuario where idUsuario = ?");
    //substitui o ? pelo "i"(inteiro) que corresponde ao idUsuario
    $stmt->bind_param("i", $this->idUsuario);
    //retorna verdadeiro se a instrução foi executa com sucesso no sgbd
    return $stmt->execute(); //executa a instrução no sgbd
  }
```

update()

```
*   método chamado pelo arquivo index.php
    quando é recebido um PUT para atualização
    de usuário. */
public function update(){
    $this->banco = new Banco(); //Instancia a classe banco
    $this->senha = md5($this->senha);  //cria um hash da senha usando o algoritmo md5
    //prepara a instrução de update no banco
    $stmt = $this->banco->getConexao()->prepare("update usuario
                        set nome=?,
                        email=?,
                        senha=?
                        where idusuario = ?");
    //substitui os "s,i" pelos interrogações e valores de variáveis.
    $stmt->bind_param("sssi", $this->nome, $this->email, $this->senha, $this->idUsuario);
    return $stmt->execute();  //executa a instrução no sgbd
}
```

read()

```
/*   método chamado pelo arquivo index.php , quando é recebido um GET com id do Usuario
     para retornar dados do usuário com id correspondente.     */
  public function read(){
      $this->banco = new Banco(); //Instancia a classe banco
     //prepara a instrução de select no banco
     $stmt = $this->banco->getConexao()->prepare("select *  from usuario where idUsuario=?");
     //substitui os "i" pelo interrogação e valor da variável.
     $stmt->bind_param("i", $this->idUsuario);
     $stmt->execute(); //executa a instrução no sgbd
     $resultado = $stmt->get_result(); //recupera os dados referentes a execução da instrução.
     //cria um vetor de objetos com os do usuário do id.
     $usuario= array();
     while ($linha = $resultado->fetch_object()) {
        //é utilizado apenas o índice zero pois , só deve existir um único usuário com o id especifico.
        $usuario[0] = new Usuario();
        $usuario[0]->setIdUsuario($linha->idUsuario);
        $usuario[0]->setNome($linha->nome);
        $usuario[0]->setEmail($linha->email);
     }
     return   $usuario; //retorna um vetor com os dados do usuário.
  }
```

readAll()

```
// método chamado pelo arquivo index.php quando é recebido um GET sem o id do Usuario para retornar dados de todos os usuario.
public function readAll(){
    $this->banco = new Banco(); //Instancia a classe banco
    //prepara a instrução de select no banco
    $stmt = $this->banco->getConexao()->prepare("select *  from usuario ");
    $stmt->execute();  //executa a instrução no sgbd
    $resultado = $stmt->get_result();  //recupera os dados referentes a execução da instrução.
    $usuario = array(); $i = 0;
    //cria um vetor com os dados de todos os usuários
    while ($linha = $resultado->fetch_object()) {
        $usuario[$i] = new Usuario(); //instância um novo usuário em cada posição do vetor
        //recupera os dados que vieram do sgbd  e faz o "set" dos dados
        $usuario[$i]->setIdUsuario($linha->idUsuario);
        $usuario[$i]->setNome($linha->nome);
        $usuario[$i]->setEmail($linha->email);
        $i++;
    }
    return   $usuario;  //retorna um vetor de usuários
}
```

```
//método que verifica se o usuário e senha estão corretos
    public function verificarUsuarioSenha(){
        $this->banco = new Banco(); //faz uma instância de banco
        $this->senha = md5($this->senha); //cria um hash utilizando o algoritmo md5 para a senha
        $sql = "select count(*) as qtd ,nome,idUsuario  from usuario where email = ? and senha =?";
        $stmt = $this->banco->getConexao()->prepare($sql);
        $stmt->bind_param("ss", $this->email, $this->senha);
        $stmt->execute();  //executa a  instrução sql no sgbd
        $resultado = $stmt->get_result();   //recupera os dados referentes a execução do sql
        //estrutura de repetição que passa por todos os dados
        while ($linha = $resultado->fetch_object()) {
            //verifica se qtd é igual a 1, significa que existe 1 usuario com o e-mail e senha fornecido.
            if ($linha->qtd == 1) {
                $this->setIdUsuario($linha->idUsuario);
                $this->setNome($linha->nome);
                return true;
            }
        }
        return false;
    }
```

verificarUsuarioSenha()

Index.php

```
use Firebase\JWT\TokenJWT;

//https://github.com/bramus/router
require_once "modelo/Router.php";
require_once "modelo/Usuario.php";
require_once "modelo/TokenJWT.php";

//Cria uma instância da classe Router
$router  = new Router();
```

Index.php

```php
// trata a rota: POST: /usuarios, a rota cadastra um cliente.
$router->post('/usuarios', function () {
$jsonRecebido = file_get_contents('php://input'); //recupera os dados enviados no corpo da requisição.
  $obj = json_decode($jsonRecebido);   //converte os dados em um objeto json.
  $u1 = new Usuario();   //cria uma instância de Usuario
 $u1->setNome($obj->nome); //faz o "set" dos dados dentro da classe.
  $u1->setEmail($obj->email); //faz o "set" dos dados dentro da classe.
  $u1->setSenha($obj->senha); //faz o "set" dos dados dentro da classe.
if ($u1->usuarioExiste() == false) { //verifica se existe algum usuário com o mesmo e-mail
    //executa o método create da classe Usuário e pepara o array resposta que será convertido em json
    $resposta['status'] = $u1->create();
    $resposta['msg'] = 'cadastrado com sucesso';
    $resposta['dados'] = $u1;
  } else {
    $resposta['status'] = false;
    $resposta['msg'] = 'já existe um usuário com o e-mail fornecido';
    $resposta['dados'] = $u1;
  }
  echo json_encode($resposta);
});
```

```
// trata a rota: GET /usuario, retorna todos os usuários
$router->get('/usuarios', function () {
$headers     = apache_request_headers(); //recupera o header contendo o token de autorização
$tokenJWT = new TokenJWT(); //instância a classe TokenJWT()
 //verifica se o token é valido ou não
 if ($tokenJWT->validarToken($headers) == true) {
   $u1 = new Usuario();
   $resposta['status'] = true;
   $resposta['dados'] =  $u1->readAll();
   echo json_encode($resposta);
   exit;
 } else {
   $resposta['status'] = false;
   $resposta['msg'] =  'Token inválido';
   echo json_encode($resposta);
 }
});
```

Index.php

```
//trata a rota: GET /usuario/idUsuario, retorna usuário com id especifico
$router->get('/usuarios/(\d+)', function ($v1) {
$headers     = apache_request_headers(); //recupera o header contendo o token de autorização
$tokenJWT = new TokenJWT(); //instância a classe TokenJWT()
if ($tokenJWT->validarToken($headers) == true) { //verifica se o token é valido ou não
    $u1 = new Usuario();
    $u1->setIdUsuario($v1);
    $resposta['status'] = true;
    $resposta['dados'] =  $u1->read();
    echo json_encode($resposta);
  } else {
    $resposta['status'] = false;
    $resposta['msg'] =  'Token inválido';
    echo json_encode($resposta);
  }
});
```

Index.php

Index.php

```
//trata a rota: PUT /usuario/idUsuario
$router->put('/usuarios/(\d+)', function ($v1) {
$headers     = apache_request_headers(); //recupera o header contendo o token de autorização
$tokenJWT = new TokenJWT();  //instância a classe TokenJWT()
if ($tokenJWT->validarToken($headers) == true) { //verifica se o token é valido ou não
    $u1 = new Usuario();
    $jsonRecebido = file_get_contents('php://input'); //recupera os dados enviado no corpo da requisição.
    $obj = json_decode($jsonRecebido); //converte os dados em um objeto json.
    $u1->setIdUsuario($v1); //faz o set dos dados para update
    $u1->setNome($obj->nome); //faz o set dos dados para update
    $u1->setEmail($obj->email); //faz o set dos dados para update
    $u1->setSenha($obj->senha); //faz o set dos dados para update
    $resposta['status'] = $u1->update();
    $resposta['msg'] = "Atualizado com sucesso";
    $resposta['dados'] = $u1;
    echo json_encode($resposta);
  } else {
    $resposta['status'] = false;
    $resposta['msg'] = 'Usuário não logado';
    echo json_encode($resposta);
  }
});
```

Index.php

```
//trata a rota: DELETE /usuario/idUsuario
$router->delete('/usuarios/(\d+)', function ($v1) {
  //recupera o header contendo o token de autorização
  $headers     = apache_request_headers();
  //instância a classe TokenJWT()
  $tokenJWT = new TokenJWT();
  //verifica se o token é valido ou não
  if ($tokenJWT->validarToken($headers) == true) {
    $u1 = new Usuario();
    $u1->setIdUsuario($v1);
    $resposta['status'] = $u1->delete();
    $resposta['msg'] = "excluído com sucesso";
    echo json_encode($resposta);
  } else {
    $resposta['status'] = false;
    $resposta['msg'] = 'Usuário não logado';
    echo json_encode($resposta);
  }
});
```

Index.php

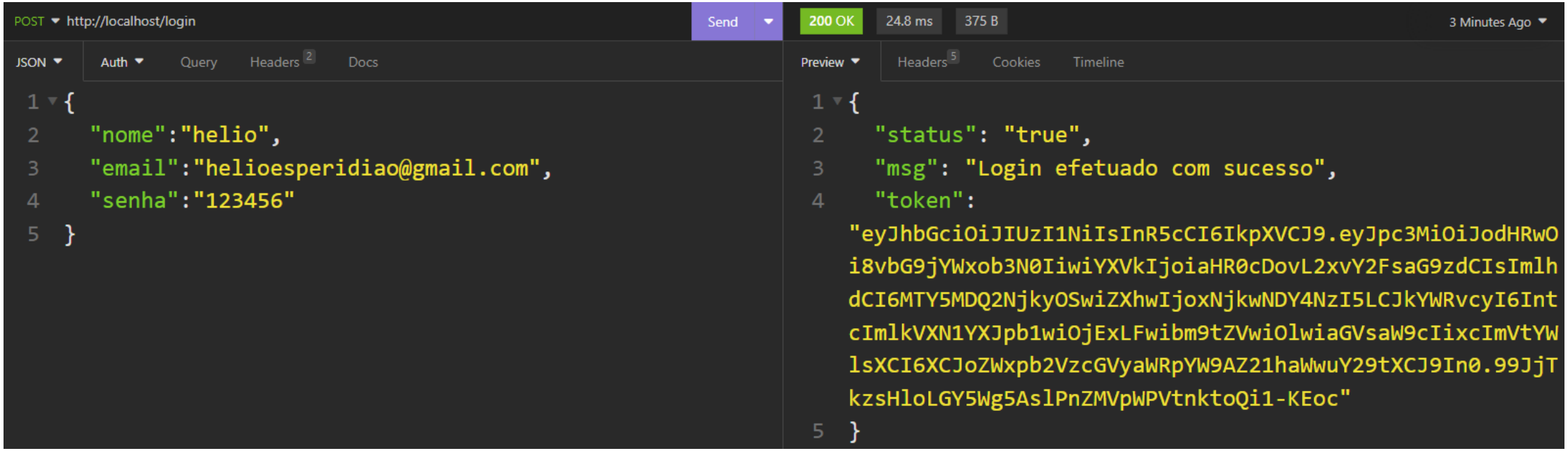
```
//trata a rota POST: / login, verifica se usuário e senha estão corretos, cria uma sessão para o usuário
$router->post('/login', function () {
  $jsonRecebido = file_get_contents('php://input'); //recupera os dados enviado no corpo da requisição.
  $obj = json_decode($jsonRecebido);
  $u1 = new Usuario();
  $u1->setEmail($obj->email);
  $u1->setSenha($obj->senha);
  $resposta = array();
  if ($u1->verificarUsuarioSenha() == true) {
    $tokenJWT = new TokenJWT();
    $novoToken = $tokenJWT->gerarToken(json_encode($u1));
    $resposta['status'] = 'true';
    $resposta['msg'] = "Login efetuado com sucesso";
    $resposta['token'] =  $novoToken;
  }else{
    $resposta['status'] = 'false';
    $resposta['msg'] = "Login inválido";
  }
  echo json_encode($resposta);
});

$router->run();
```

Rotas do exemplo

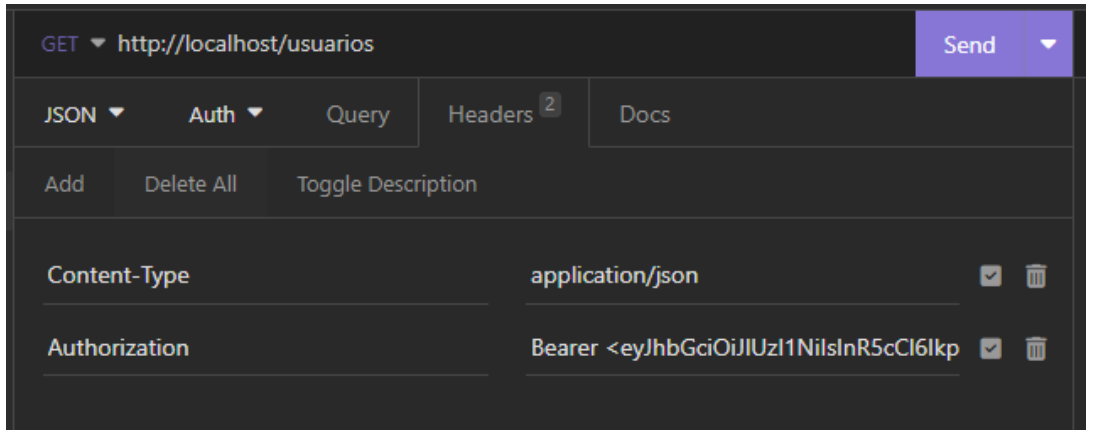
Verbo	Rota	envio	Funcionalidade/retorno

POST	/login	Corpo:
		{
		"nome":"helio",
		"email":"helioesperidiao@gmail.com",
		"senha":"123456"
		}
			Verificar usuário e senha e **retornar token de acesso.**



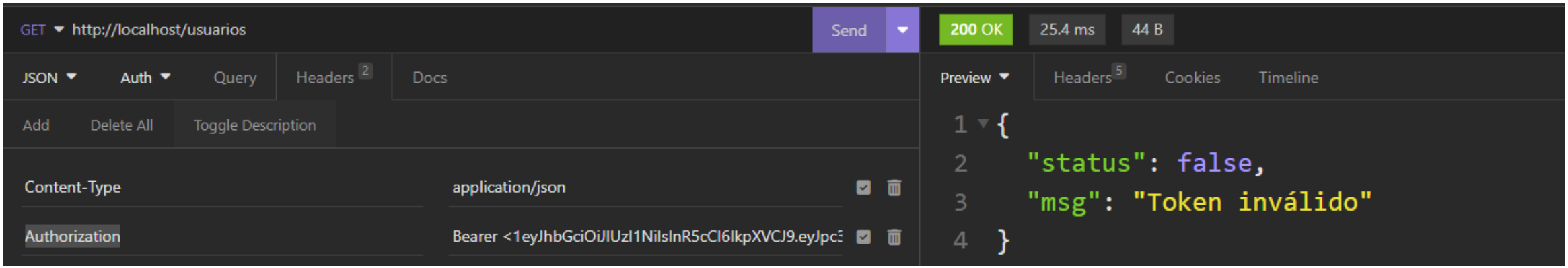
Adicione o atributo Authorization fornecer o valor: Bearer <token>

- Authorization: Bearer <eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0IiwiYXVkIjoiaHR0cDovL2xvY2FsaG9zdCIsImlhdCI6MTY5MDU3OTU2OSwiZXhwIjoxNjkwNTgxMzY5LCJkYWRvcyI6IntcImlkVXN1YXJpb1wiOjExLFwibm9tZVwiOlwiaGVsaW9cIixcImVtYWlsXCI6XCJoZWxpb2VzcGVyaWRpYW9AZ21haWwuY29tXCJ9In0.uvPZdKajZfJSHJTZRNdK3J-MrKQP8_-aKzPjbtl0EM0>

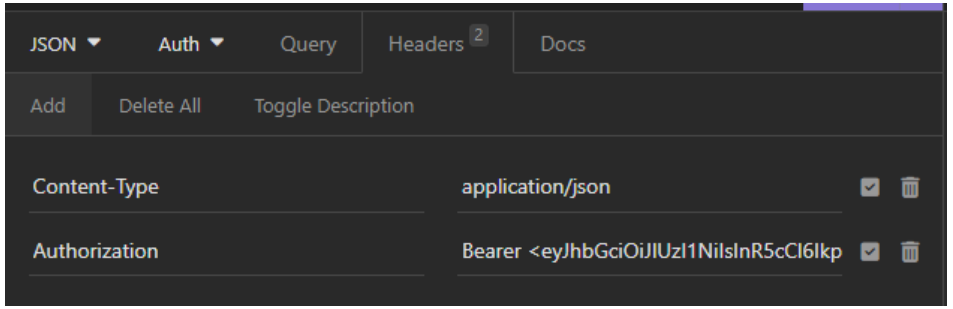


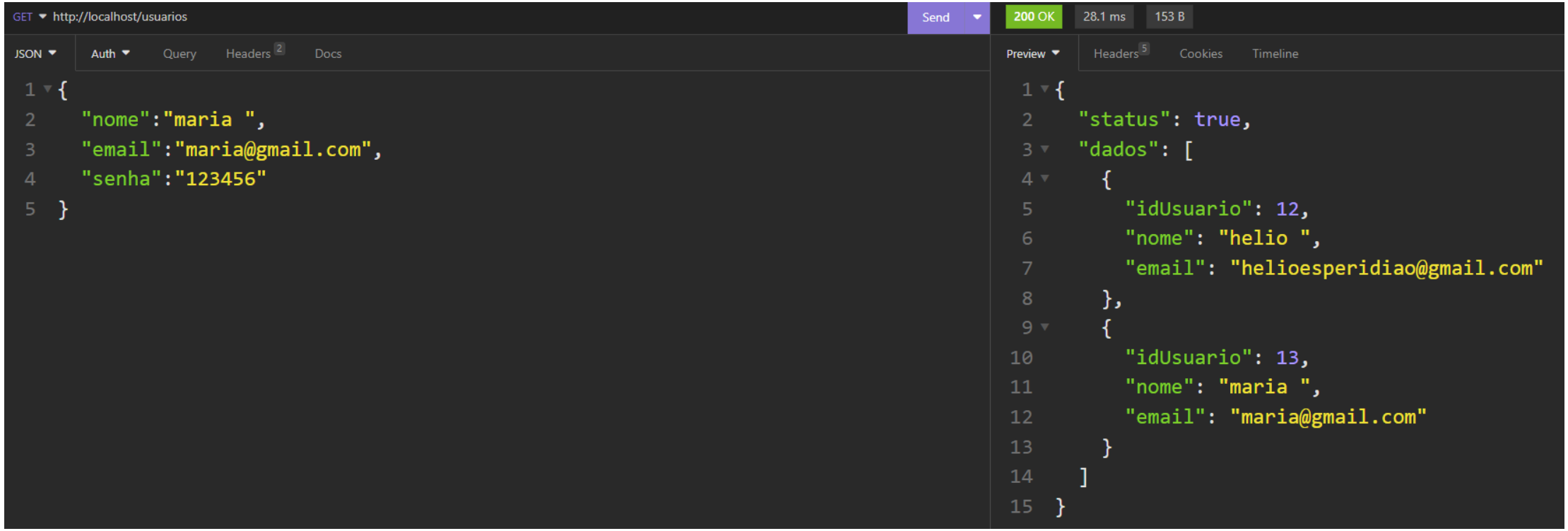
Caso seja enviado um token errado

- O exemplo abaixo consiste em enviar um GET para rota /usuarios:
- Para listar todos os usuários o sistema precisa validar o token
- Caso não seja fornecido um token ou ele seja inválido a transação não é concluída.
  - O sistema deve validar o token. Caso não seja válido o requisitante não terá acesso ao recurso.



Não esqueça de enviar o token no cabeçalho

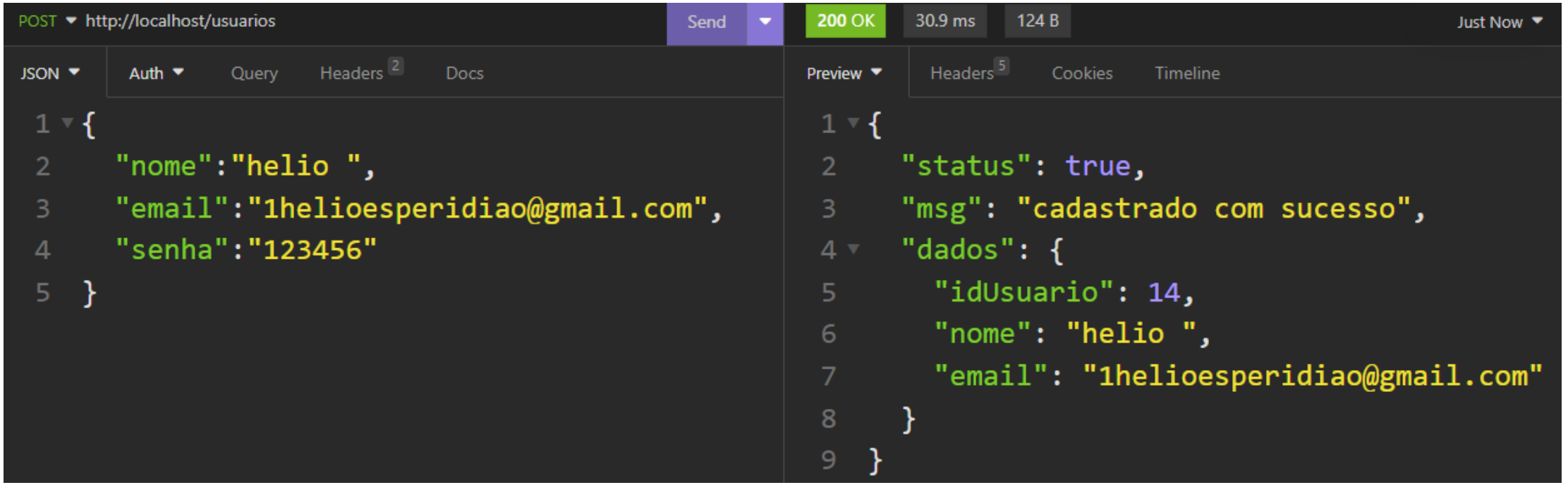




Rotas do exemplo

Verbo	Rota	envio	Funcionalidade/retorno

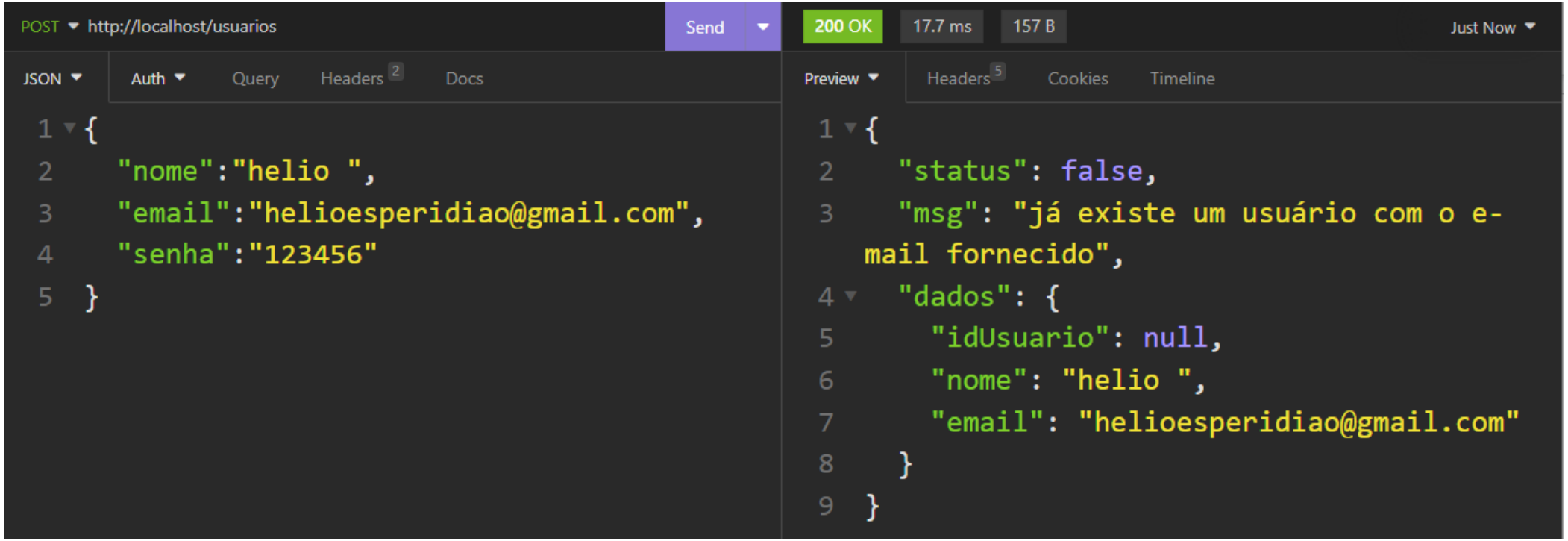
POST	/usuarios	{
		"nome":"helio ",
		"email":"helioesperidiao@gmail.com",
		"senha":"123456"
		}
			Cadastra usuários no sistema.
			Não é preciso enviar o token de autorização.
			Neste Exemplo É possível cadastrar novos usuários sem a necessidade de ser autenticado.



Rotas do exemplo

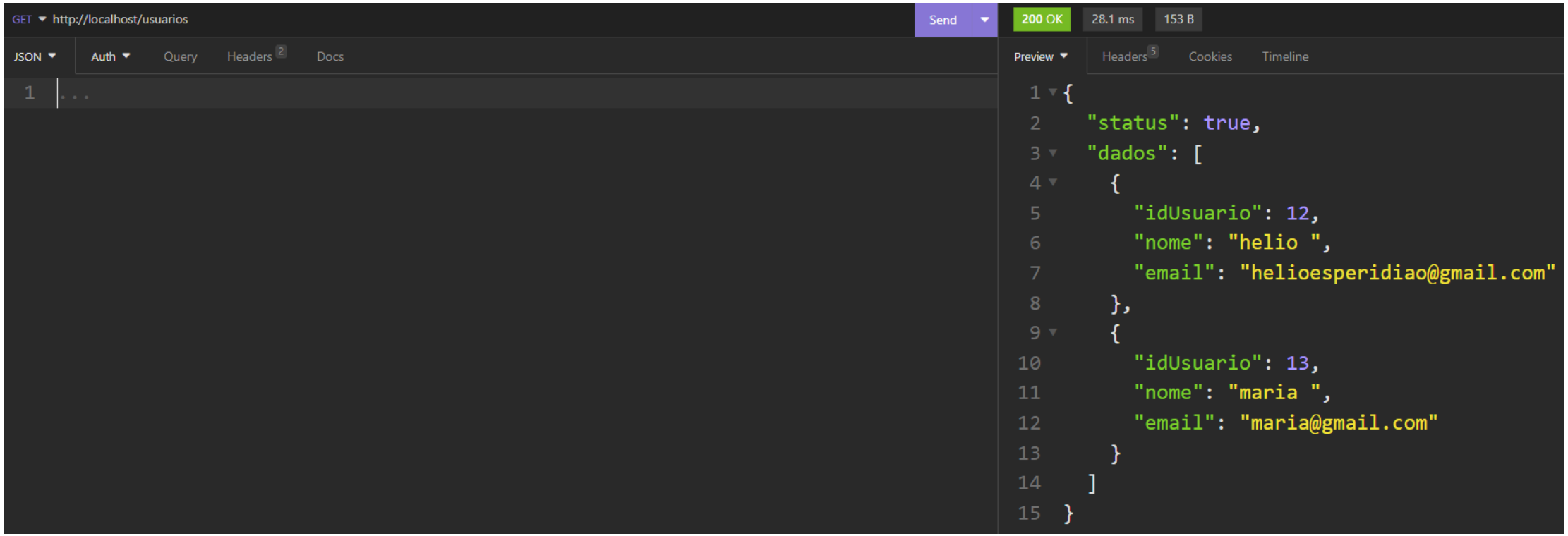
Verbo	Rota	envio	Funcionalidade/retorno

POST	/usuarios	{
		"nome":"helio ",
		"email":"helioesperidiao@gmail.com",
		"senha":"123456"
		}
			Cadastra usuários no sistema.
			Não é preciso enviar o token de autorização.
			Neste Exemplo É possível cadastrar novos usuários sem a necessidade de ser autenticado.



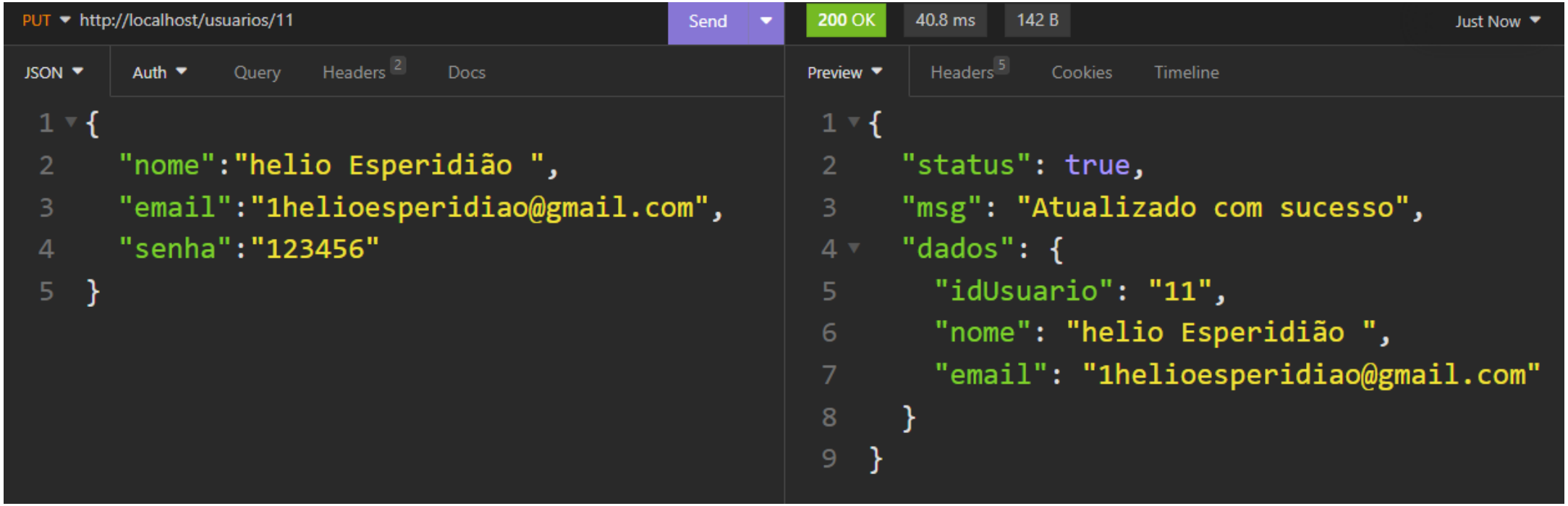
Rotas do exemplo

Verbo	Rota	envio	Funcionalidade/retorno
GET	/usuarios	Header/Authorization: Bearer <token>	Verificar usuário e senha e retornar token de acesso.



Rotas do exemplo

Verbo	Rota	envio	Funcionalidade/retorno
put	/usuarios/{idUsuario}	Header/Authorization: Bearer <token>	Atualiza usuário se o token for valido.



Rota delete

Verbo	Rota	envio	Funcionalidade/retorno
DELETE	/usuarios/{idUsuario}	Header/Authorization: Bearer <token>	Exclui o usuário se o token for valido.