JWT
JSON Web Token

Prof. Me. Hélio Esperidião

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 ***W**orld **W**ide **W**eb 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**.

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.

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.

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

Base64

- A codificação Base64 divide os dados de entrada em blocos de 3 bytes (24 bits).
- Cada bloco de 3 bytes é dividido em quatro blocos de 6 bits.
- Cada bloco de 6 bits é mapeado para um caractere no alfabeto Base64.

- Base64 é apenas uma forma diferente de representar texto

Base64Url

- Base64Url é uma variação do codificação Base64, adaptada para uso em URLs e nomes de arquivos.
- A codificação Base64 original utiliza caracteres que não são seguros para URLs e nomes de arquivos, como: + /
  - Pode existir um nome de arquivo com / ?

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)
```

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 de um JWT (*JSON Web Token*) é a segunda parte do token e contém as declarações (claims) sobre uma entidade (geralmente, o usuário) e metadados adicionais.
- Essas informações são representadas como um objeto JSON e são codificadas em Base64Url.

Componentes do Payload

- O payload pode conter três tipos de declarações (claims):
  - Registered claims
  - Public claims
  - Private claims

Registered claims

- São um conjunto de claims pré-definidos, recomendados mas não obrigatórios. Exemplos incluem:

1. iss (issuer): Identifica quem emitiu o token.
2. sub (subject): Identifica o assunto do token (geralmente o usuário).
3. aud (audience): Destinatário do token.
4. exp (expiration time): Tempo de expiração do token.
5. nbf (not before): O token não é válido antes desta data.
6. iat (issued at): Data em que o token foi emitido.
7. jti (JWT ID): Identificador único para o token.

iss (Issuer):

- Descrição: Identifica quem emitiu o token.
- Exemplo: "iss": "https://example.com"

sub (Subject):

- Descrição: Identifica o assunto do token. Este é geralmente um identificador único do usuário.
- Exemplo: "sub": "1234567890"

aud (Audience):

- Descrição: Identifica o destinatário do token.
- O destinatário deve verificar essa claim para garantir que o token foi destinado a ele.
- Exemplo: "aud": "https://myapi.example.com"

exp (Expiration Time):

- Descrição: Identifica a data e hora em que o token expira.
- Após essa data, o token não deve ser aceito.
- Exemplo: "exp": 1716239022
  - O valor 1716239022 está em segundos desde a época Unix (*Unix epoch time*), que é 00:00:00 UTC de 1 de janeiro de 1970.
  - Essa forma de representar o tempo é comumente usada em sistemas e protocolos computacionais para evitar ambiguidades relacionadas a fusos horários e formatos de data.

nbf (Not Before)

- Descrição: Identifica a data e hora antes da qual o token não é válido.
- Exemplo: "nbf": 1716239022

iat (Issued At):

- Descrição: Identifica a data e hora em que o token foi emitido.
- Exemplo: "iat": 1716239022

jti (JWT ID):

- Descrição: Identificador único do token.
- Pode ser usado para evitar a reutilização de um token (*proteção contra replay attacks*).
- Exemplo: "jti": "a-unique-id"

Exemplo Completo de Payload com Registered Claims


```
{
  "iss": "https://example.com",
  "sub": "1234567890",
  "aud": "https://myapi.example.com",
  "exp": 1716239022,
  "nbf": 1716239022,
  "iat": 1716239022,
  "jti": "a-unique-id",
  "name": "John Doe",
  "admin": true
}
```

Public claims:

- Public claims são declarações que são reconhecidas publicamente e podem ser usadas por qualquer aplicação que compreenda JWTs.
- Elas são definidas de maneira a evitar colisões de nome e devem ser registradas no *IANA JSON Web Token Claims Registry*.
  - Exemplo: name, email, role.

Exemplos de Public Claims

- name: Nome do usuário.
  - Exemplo: "name": "John Doe"
- email: Endereço de e-mail do usuário.
  - Exemplo: "email": "john.doe@example.com"
- role: Papel ou função do usuário no sistema.
  - Exemplo: "role": "admin"
- organization: Organização à qual o usuário pertence.
  - Exemplo: "organization": "Example Corp"
- permissions: Lista de permissões ou direitos do usuário.
  - Exemplo: "permissions": ["read", "write", "delete"]
- locale: Preferência de idioma do usuário.
  - Exemplo: "locale": "en-US“
- *O programador define conforme a necessidade da aplicação*

Public claims:


```
{
"name": "John Doe",
  "email": "john.doe@example.com",
  "role": "admin",
  "organization": "Example Corp",
  "permissions": ["read", "write", "delete"],
  "locale": "en-US"
}
```

Private claims

- Private claims são declarações definidas por acordo entre duas partes que compartilham o token.
- Elas são específicas a uma aplicação ou sistema e não são registradas publicamente.
- Por isso, não são padronizadas e só fazem sentido para as partes que concordaram em utilizá-las.
- Exemplo: userId, department, projectId.

resumo

Característica	Public Claims	Private Claims
Definição	Claims reconhecidas publicamente	Claims definidas por acordo entre duas partes
Registro	Podem ser registradas no IANA JWT Claims Registry	Não são registradas publicamente
Utilização	Usadas para informações comuns e amplamente reconhecidas	Usadas para informações específicas de uma aplicação
Exemplo	name, email, role	userId, department, projectId

Considerações finais sobre as claims

- Junto em único json as ***Registered claims***, ***Public claims*** e ***Private claims*** em um único json;

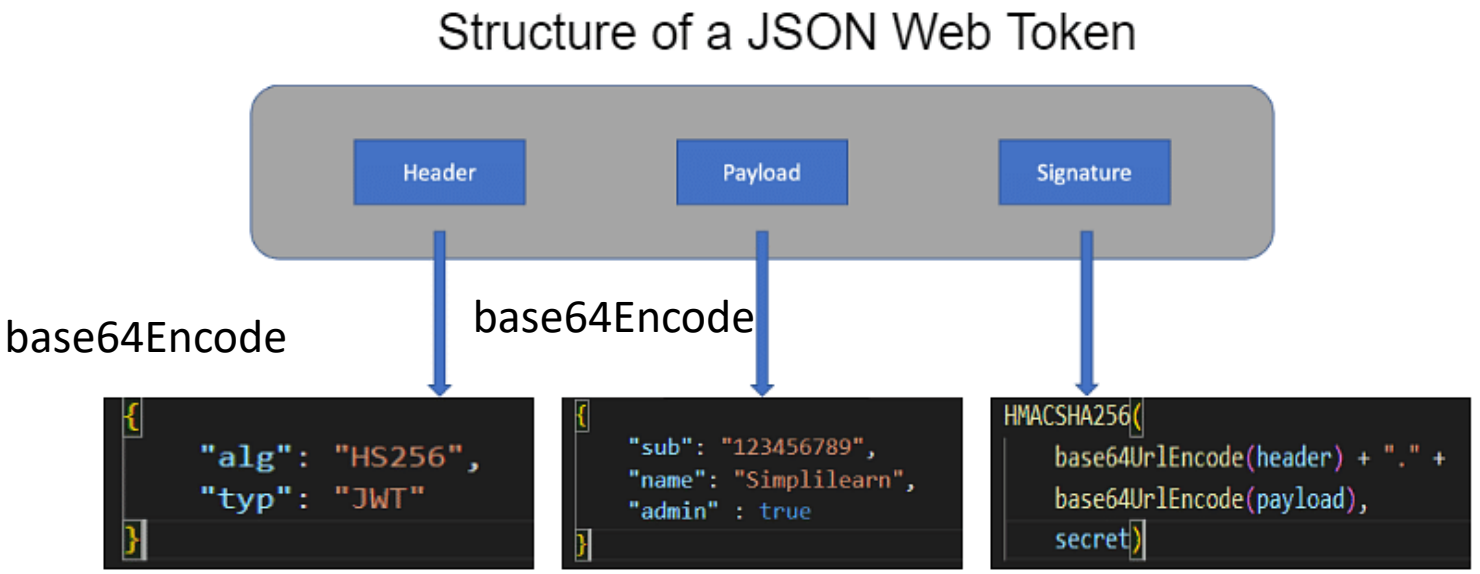
```
{
    "iss": "https://example.com", // Registered Claim: Emissor do token
    "sub": "1234567890", // Registered Claim: Identificador do sujeito (usuário)
    "aud": "https://your-app.com", // Registered Claim: Destinatário do token
    "exp": 1722470421, // Registered Claim: Tempo de expiração
    "nbf": 1622466821, // Registered Claim: Não válido antes de
    "iat": 1622466821, // Registered Claim: Emitido em
    "jti": "unique-jwt-id", // Registered Claim: Identificador do token
    "https://example.com/username": "john_doe", // Public Claim: Nome de usuário com namespace para evitar conflitos
    "https://example.com/user_email": "john.doe@example.com", // Public Claim: Email do usuário com namespace para evitar conflitos
    "https://example.com/user_role": "admin", // Public Claim: Papel do usuário com namespace para evitar conflitos
    "name": "John Doe", // Private Claim: Nome do usuário
    "department": "Engineering", // Private Claim: Departamento do usuário
    "employee_id": "EMP12345", // Private Claim: Identificador do funcionário
    "manager": "Jane Smith", // Private Claim: Gerente do usuário
    "location": "Building A, Floor 3", // Private Claim: Localização do escritório do usuário
    "preferences": { // Private Claim: Preferências do usuário
        "language": "en",
        "timezone": "GMT-5"
    },
    "custom_data": { // Private Claim: Dados personalizados específicos para a aplicação
        "projects": ["project1", "project2", "project3"],
        "access_level": 5,
        "tags": ["developer", "full-time"]
    }
}
```

Na prática

- O que vimos anteriormente é o rigor da teoria.
- Pode haver modificações para adequar a complexidade do sistema que está sendo desenvolvido.

Signature

- A assinatura do token (signature) é composta pela codificação base64 do header e do payload somada a uma chave secreta e é gerada pelo algoritmo especificado no cabeçalho.



Simplificando

token = base64Encode(header) + base64Encode(payload) + criptografia(base64Encode(header)+ base64Encode(payload));

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lIjoiSGVsaW8ifQ.MsTLetZtc05RBGb8ZQblYNVjmY58n7WGBJczZbDL010

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.
  - 2^256 = $1{,}16 \times 10^{77}$
- 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 256 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.

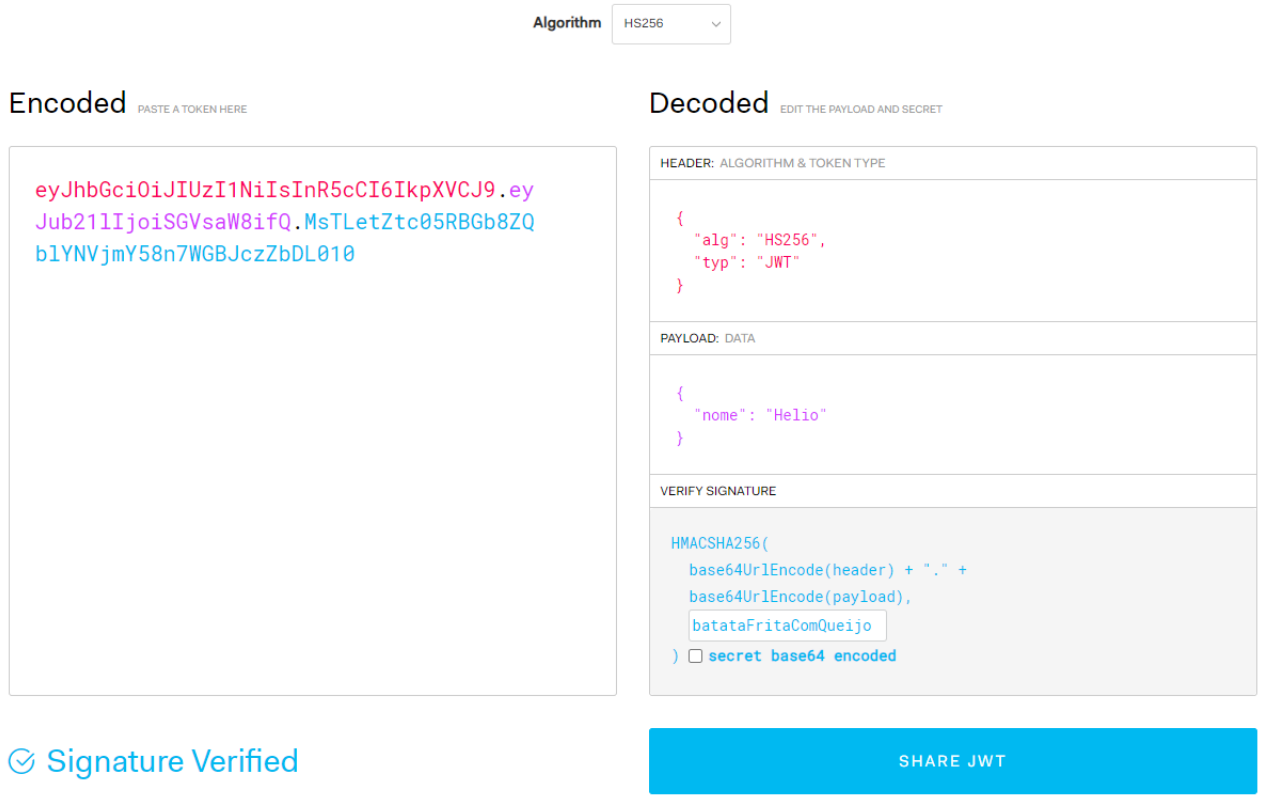
Vamos testar

https://jwt.io/

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

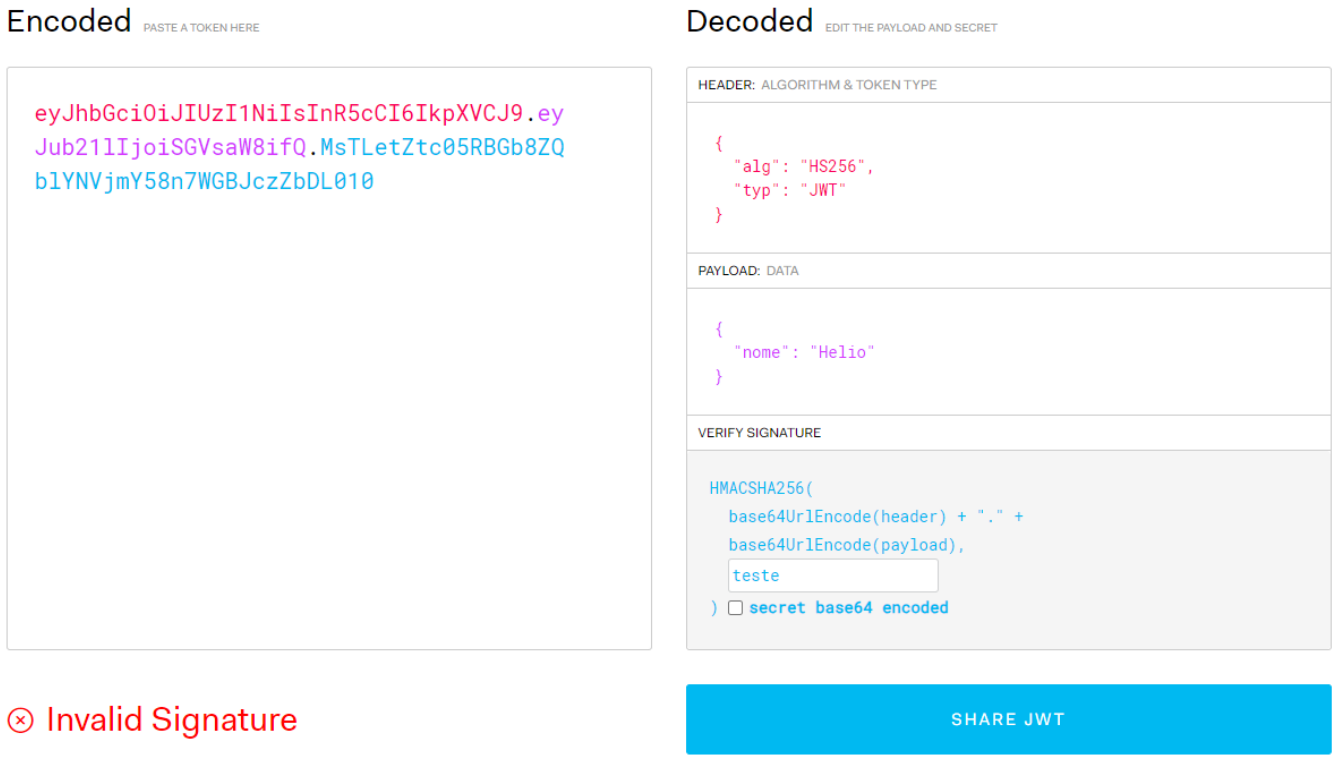
eyJhbGciOiJIUzI1NiIsInR5
cCI6IkpXVCJ9.eyJub21lIjo
iSGVsaW8ifQ.MsTLetZtc0
5RBGb8ZQblYNVjmY58n
7WGBJczZbDL010

Assinatura:atataFritaComQueijo



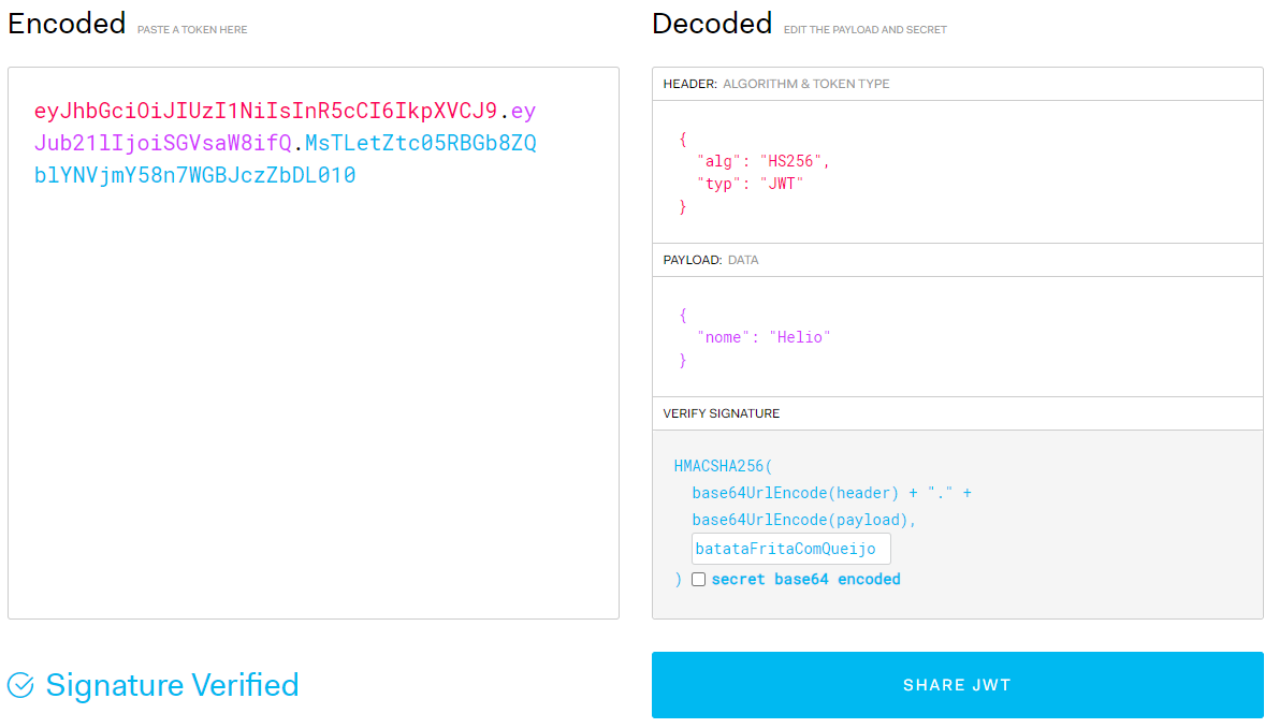
Observe

- 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

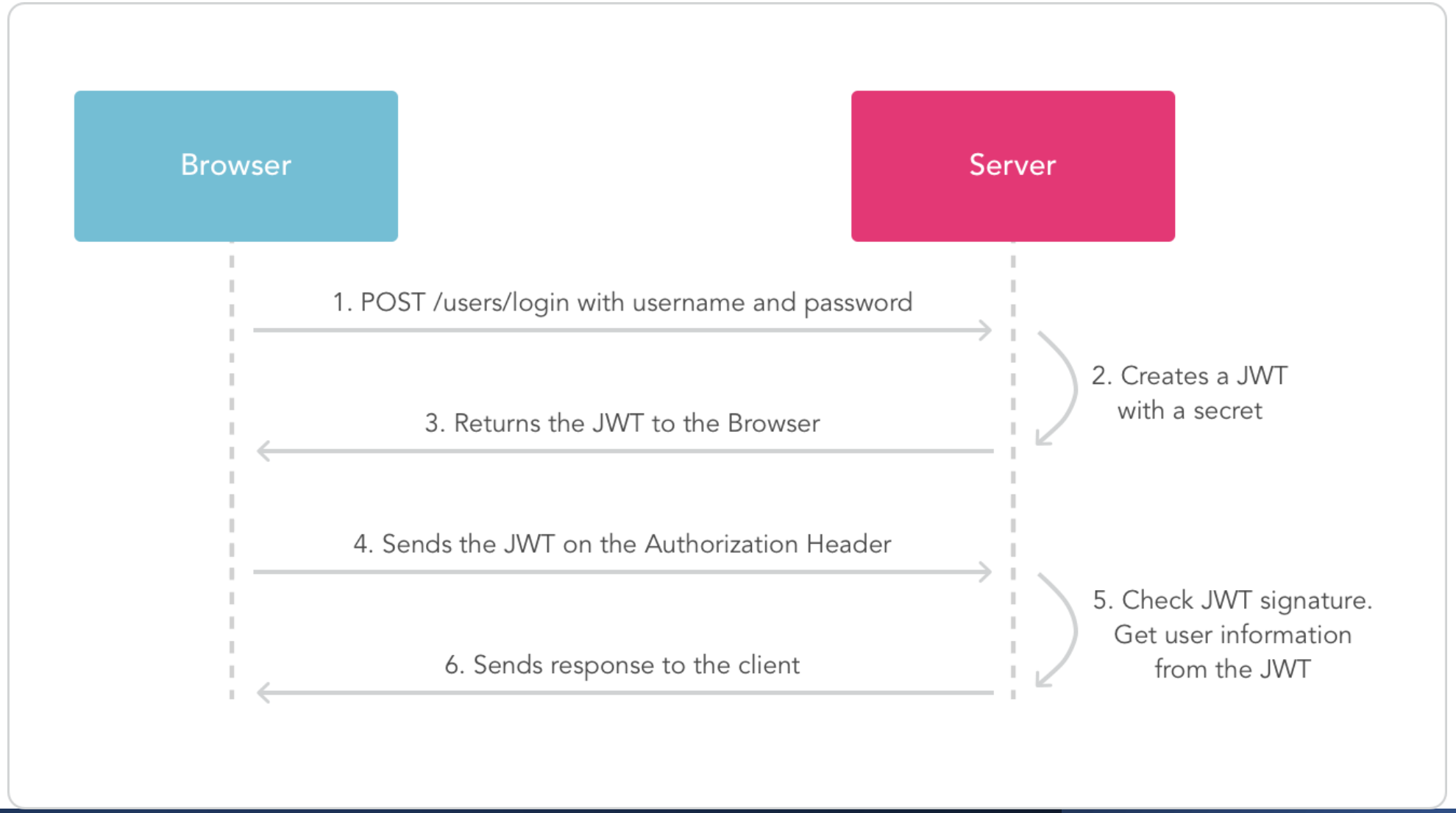
- Com a chave correta a assinatura é válida;



Tente você

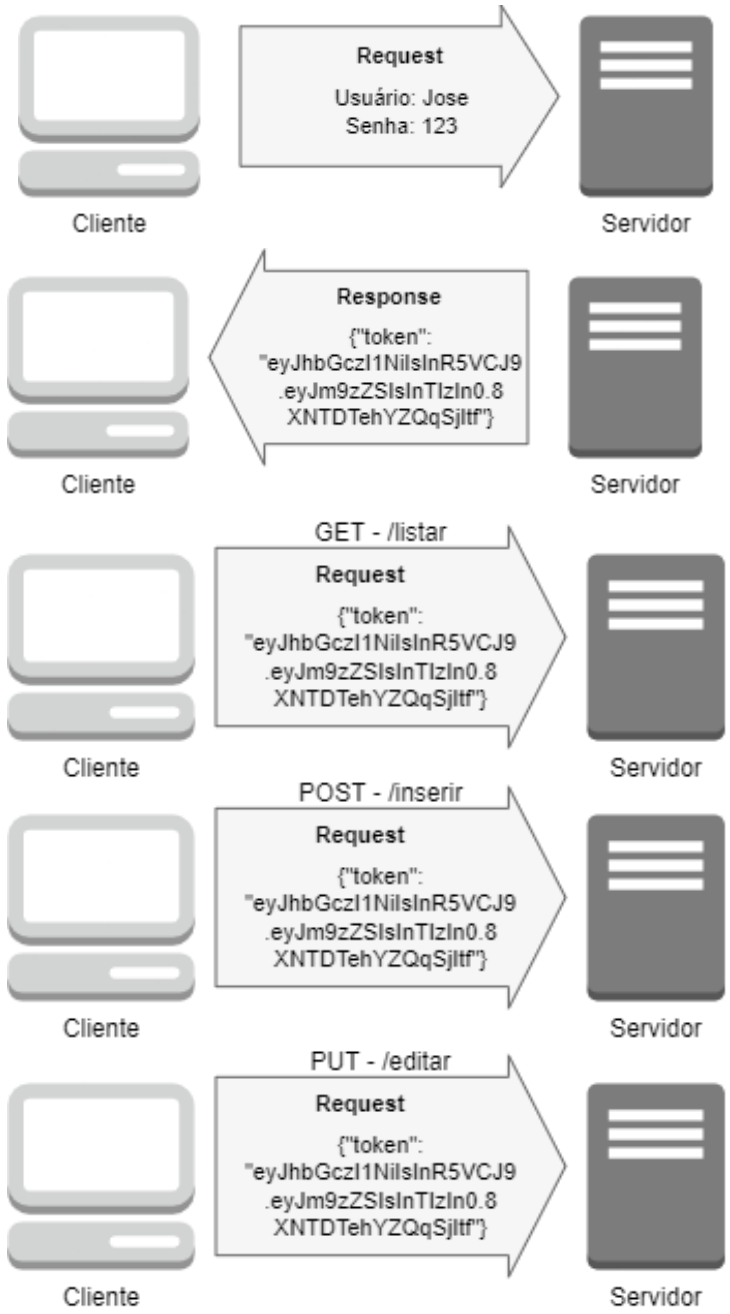
- Abra: https://jwt.io/
- Insira o token:
  - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lIjoiSGVsaW8ifQ.MsTLetZtc05RBGb8ZQblYNVjmY58n7WGBJczZbDL010.
- Insira a chave:
    - batataFritaComQueijo
- Tente outras chaves:
  - Teste007, batata com bacon.
- Verifique que é possível visualizar os dados do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

Uso em aplicações web



Arquitetura

Arquitetura



Implementação

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

MeuTokenJWT.php

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

MeuTokenJWT.php

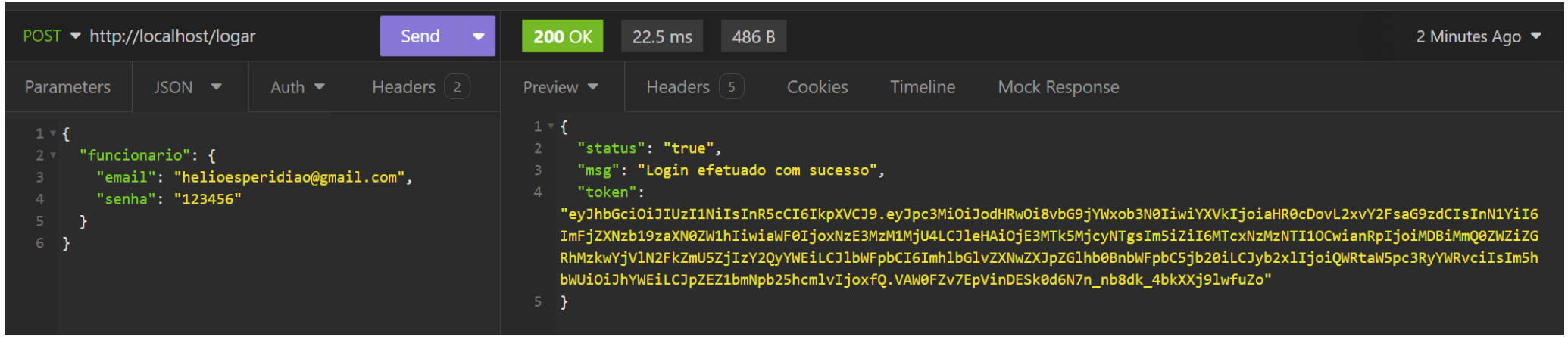
```php
class MeuTokenJWT{
    //chave de criptografia, defina uma chave forte e a mantenha segura.
    private $key = "x9S4q0v+V0IjvHkG20uAxaHx1ijj+q1HWjHKv+ohxp/oK+77qyXkVj/l4QYHHTF3";
    //algoritmo de criptografia para assinatura
    //Suportados: 'HS256' , 'ES384','ES256', 'ES256K', ,'HS384', 'HS512', 'RS256', 'RS384'
    private $alg = 'HS256';
    private $type = 'JWT';
    private $iss = 'http://localhost'; //emissor do token
    private $aud = 'http://localhost'; //destinatário do token
    private $sub = "acesso_sistema";   //assunto do token
    private $iat = "";  //momento de emissão
    private $exp = "";     //momento de expiração
    private $nbf = ""; //não é válido antes do tempo especificado
    private $jti = "";  //Identificador único
    private $payload; //claims
    //tempo de validade do token
    private $duracaoToken = 3600 * 24 * 30; //3600 segundos = 60 min
```

```php
public function gerarToken($parametro_claims) {

        // Criação dos headers como objeto da classe stdClass
        $objHeaders = new stdClass();
        $objHeaders->alg = $this->alg;
        $objHeaders->typ = $this->type;
        // Criação do payload como objeto da classe stdClass
        $objPayload = new stdClass();
        //Registered Claims
        $objPayload->iss = $this->iss;        // emissor do token
        $objPayload->aud = $this->aud;        // destinatário do token
        $objPayload->sub = $this->sub;        // assunto do token
        $objPayload->iat = time();            // momento de criação do token
        $objPayload->exp = time() + $this->duracaoToken;   // momento de expiração = tempo atual + duração
        $objPayload->nbf = time();        // momento em que o token torna-se valido.
        $objPayload->jti = bin2hex(random_bytes(16)); // gera um valor aleatório para jti;
        //Public Claims
        $objPayload->email = $parametro_claims->email;
        $objPayload->role = $parametro_claims->role;
        $objPayload->name = $parametro_claims->name;
        //Private claims
        $objPayload->idFuncionario = $parametro_claims->idFuncionario;
        // Utiliza a biblioteca do Firebase para gerar o token com os parâmetros
        $token = JWT::encode((array) $objPayload, $this->key, $this->alg, null, (array) $objHeaders);
        return $token;
    }
```

```php
public function validarToken($stringToken){
        if (isset($stringToken)) {
            if ($stringToken == "") {
                return false;
            } else {
                $remover = ["Bearer ", " "];
                $token = str_replace($remover, "", $stringToken);
                try {
                    $payloadValido = JWT::decode($token, new Key($this->key, $this->alg));
                    $this->setPayload($payloadValido);
                    return true;
                } catch (SignatureInvalidException $e) { // A assinatura do token é inválida.
                     return false;
                } catch (BeforeValidException $e) { // O token não é válido ainda (antes do tempo 'nbf').
                    return false;
                } catch (ExpiredException $e) { // O token expirou.
                    return false;
                } catch (InvalidArgumentException $e) { // Argumento inválido passado.
                            return false;
                } catch (DomainException $e) {  // Exceção de domínio genérica.
                    return false;
                } catch (UnexpectedValueException $e) { // Valor inesperado encontrado.
                    return false;
                } catch (Exception $e) { // Qualquer outra exceção genérica.
                    return false;
                }
            }
        }
        return false;
    }
```

Rota: /logar

- Envie um post para /logar com o seguinte json.
- O sistema deve retornar um token de acesso caso o login seja valido



Programando a rota /login

```php
// Define uma rota para efetuar login
$roteador->post("/logar", function () {
    // Requer o arquivo de controle responsável por fazer o login do funcionário
    require_once ("controle/funcionario/controle_funcionario_logar.php");
});
```

controle_funcionario_logar.php 1/2

```php
use Firebase\JWT\MeuTokenJWT; //utiliza o pacote do JWT

require_once "modelo/Funcionario.php"; //importa o arquivo Funcionario.php e
require_once "modelo/MeuTokenJWT.php"; //importa o arquivo MeuTokenJWT.php

$jsonRecebido = file_get_contents('php://input'); //recupera o json
$objJson = json_decode($jsonRecebido); // converte o texto json para um objeto
$objFuncionario = new Funcionario(); // cria um objeto de funcionario

$objFuncionario->setEmail($objJson->funcionario->email); //passa o email para o objeto funcionario
$objFuncionario->setSenha($objJson->funcionario->senha);//passa a senha para o objeto funcionario

$objResposta = new stdClass();
```

controle_funcionario_logar.php 1/2

```php
//chama o método de verificação do usuário

if ($objFuncionario->verificarUsuarioSenha() == true) {

    $tokenJWT = new MeuTokenJWT();
    $objClaimsToken = new stdClass();

    //gera as claims para a criação do token

    $objClaimsToken->email = $objFuncionario->getEmail();

    $objClaimsToken->role = $objFuncionario->getCargo()->getNomeCargo();

    $objClaimsToken->name = $objFuncionario->getNomeFuncionario();

    $objClaimsToken->idFuncionario = $objFuncionario->getIdFuncionario();


    //chama o método que gera um novo token

    //passa as claims para o método

    $novoToken = $tokenJWT->gerarToken($objClaimsToken);
    $objResposta->status = 'true';

    $objResposta->msg = 'Login efetuado com sucesso';

    $objResposta->token = $novoToken;

} else {

    $objResposta->status = 'false';

    $objResposta->msg = 'Login inválido';

}

//converte a resposta em um json

echo json_encode($objResposta);
```

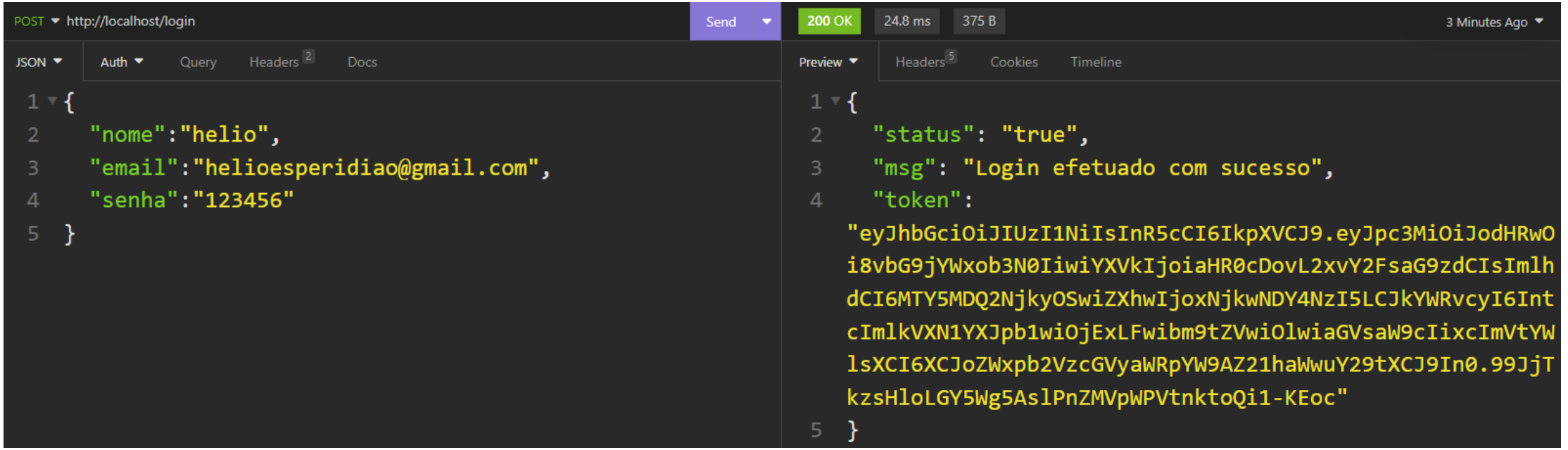
Classe funcionário

```
public function verificarUsuarioSenha()    {
    // Obtém a conexão com o banco de dados
    $conexao = Banco::getConexao();

    $SQL = "SELECT COUNT(*) AS qtd, idFuncionario,nomeFuncionario,email,idCargo, nomeCargo
    from funcionario join cargo on cargo_idCargo = idCargo WHERE email=? and senha =MD5(?)";

    $prepararSQL =  $conexao->prepare($SQL);

    $prepararSQL->bind_param("ss", $this->email, $this->senha);
     //executa a  instrução sql no sgbd
    $prepararSQL->execute();
    //recupera os dados referentes a execução do sql
    $matrizTuplas = $prepararSQL->get_result();
    //estrutura de repetição que passa por todos os dados
    while ($tupla = $matrizTuplas->fetch_object()) {
        //verifica se qtd é igual a 1, significa que existe 1 usuario om o email e senha fornecido.
        if ($tupla->qtd == 1) {
            $this->setIdFuncionario($tupla->idFuncionario);
            $this->setNomeFuncionario($tupla->nomeFuncionario);
            $this->setEmail($tupla->email);
            $this->getCargo()->setIdCargo($tupla->idCargo);
            $this->getCargo()->setNomeCargo($tupla->nomeCargo);
            return true;
        }
    }
    return false;
}
```

Rotas do exemplo

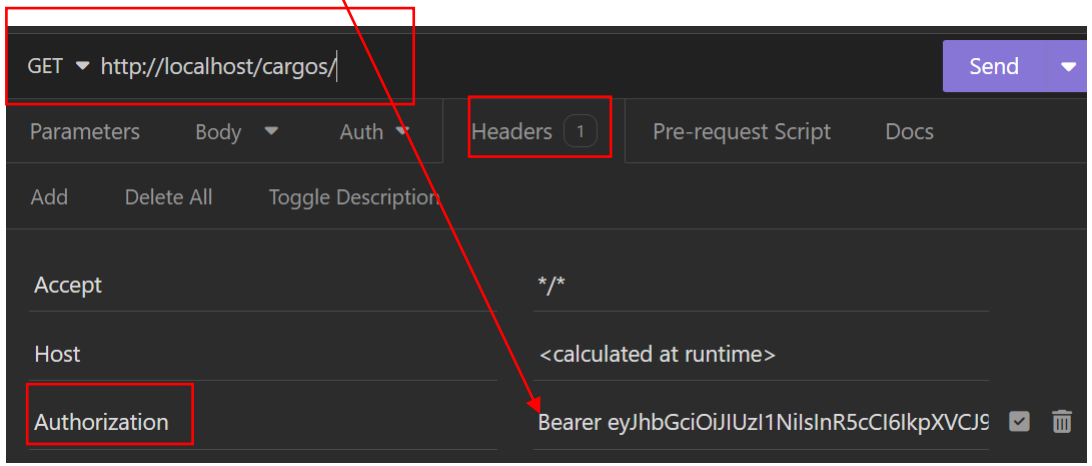
Verbo	Rota	envio	Funcionalidade/retorno

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



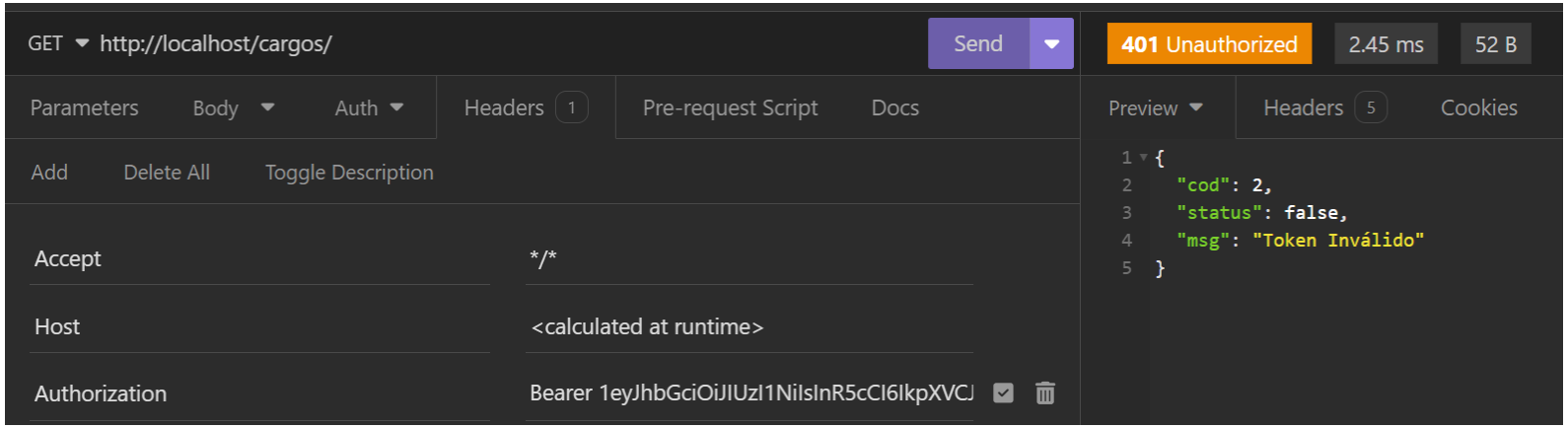
Adicione o atributo Authorization fornecer o valor: Bearer <token>

- Authorization: Bearer <
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0IiwiYXVkIjoiaHR0cDovL2xvY2FsaG9zdCIsInN1YiI6ImFjZXNzb19zaXN0ZW1hIiwiaWF0IjoxNzE3MzM1MjU4LCJleHAiOjE3MTk5MjcyNTgsIm5iZiI6MTcxNzMzNTI1OCwianRpIjoiMDBiMmQ0ZWZiZGRhMzkwYjVlN2FkZmU5ZjIzY2QyYWEiLCJlbWFpbCI6ImhlbGlvZXNwZXJpZGlhb0BnbWFpbC5jb20iLCJyb2xlIjoiQWRtaW5pc3RyYWRvciIsIm5hbWUiOiJhYWEiLCJpZEZ1bmNpb25hcmlvIjoxfQ.VAW0FZv7EpVinDESk0d6N7n_nb8dk_4bkXXj9lwfuZo >



Caso seja enviado um token errado

- O exemplo abaixo consiste em enviar um GET para rota /cargos:
- Para listar todos os cargos 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

- Caso token seja validado a operação é realizada