# JWT
# JSON Web Token

Prof. Me. Hélio Esperidião

# JSON Web Token

- JWT significa **JSON Web Token**. É um padrão aberto (RFC 7519) usado para **autenticação** e **transmissão segura de informações** entre partes como um **objeto JSON compactado e assinado** digitalmente.

# 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.

# 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

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.
- 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.

# Payload: iss (Issuer):

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

# Payload: sub (Subject):

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

  "sub": "autenticacao"

# Payload: 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"

# Payload: 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.

# Payload: nbf (Not Before)

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

# Payload: iat (Issued At):

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

# Payload: 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": “autenticacao",
  "aud": "https://myapi.example.com",
  "exp": 1716239022,
  "nbf": 1716239022,
  "iat": 1716239022,
  "jti": "a-unique-id",
}
```

# Registered claims

| Claim | Nome (em português) | Significado | Tipo de Dado |
|---|---|---|---|
| iss | Emissor | Quem emitiu o token (ex: URL ou nome do servidor) | string |
| sub | Sujeito | Identificador do usuário ou entidade dona do token | string |
| aud | Público-alvo | Destinatário(s) do token (ex: URL da API ou sistema consumidor) | string ou array |
| exp | Expiração | Momento em que o token expira (não deve ser aceito depois disso) | integer (timestamp) |
| nbf | Não antes de | Momento antes do qual o token **não é válido** | integer (timestamp) |
| iat | Emitido em | Data/hora de emissão do token | integer (timestamp) |
| jti | ID do token | Identificador único do token (útil para rastreamento ou revogação) | string |

# 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

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

# 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**.

# Considerações finais sobre as claims

- Podem existir em conjunto em um único json as ***Registered claims***, ***Public claims*** e ***Private claims***

```
{
    "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
    “name": "john_doe", // Public Claim: Nome de usuário com namespace para evitar conflitos
    “email": "john.doe@example.com", // Public Claim: Email do usuário com namespace para evitar conflitos
    "role": "admin", // Public Claim: Papel do usuário com namespace para evitar conflitos
    "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
    "custom_data": { // Private Claim: Dados personalizados específicos para a aplicação
      "projects": ["project1", "project2", "project3"],
      "access_level": 5,
      "tags": ["developer", "full-time"]
    }
  }
```

Registered Claims

Public Claim

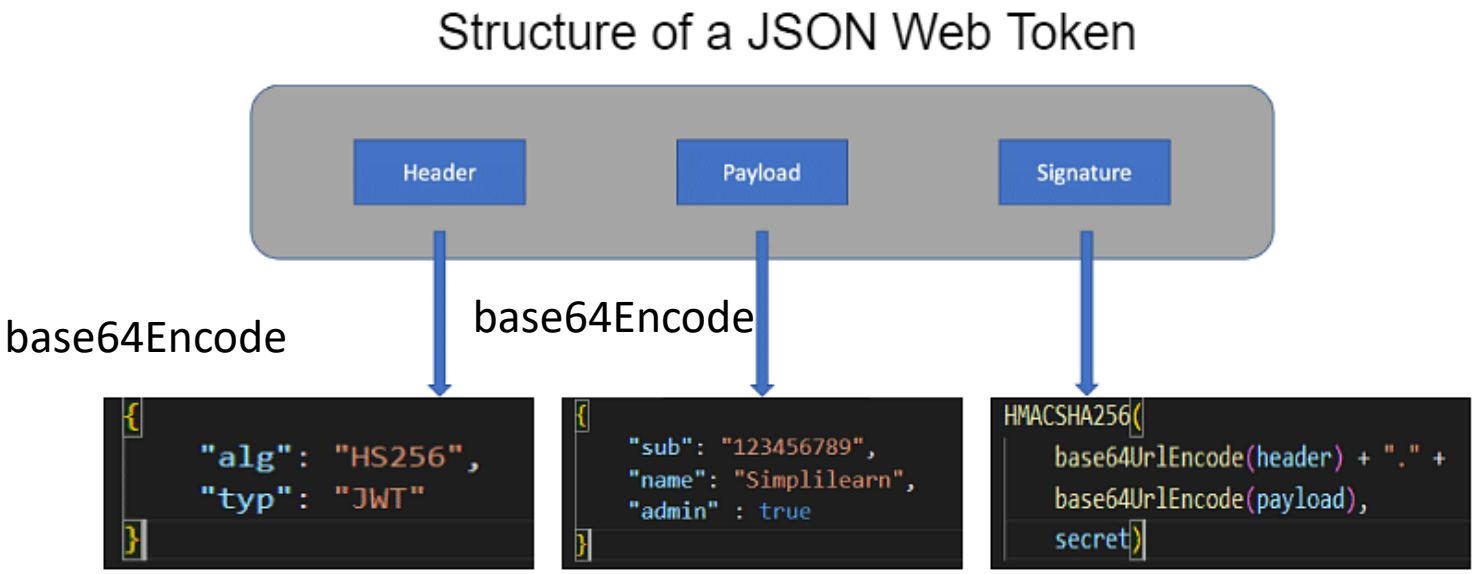
Private Claim

# 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.

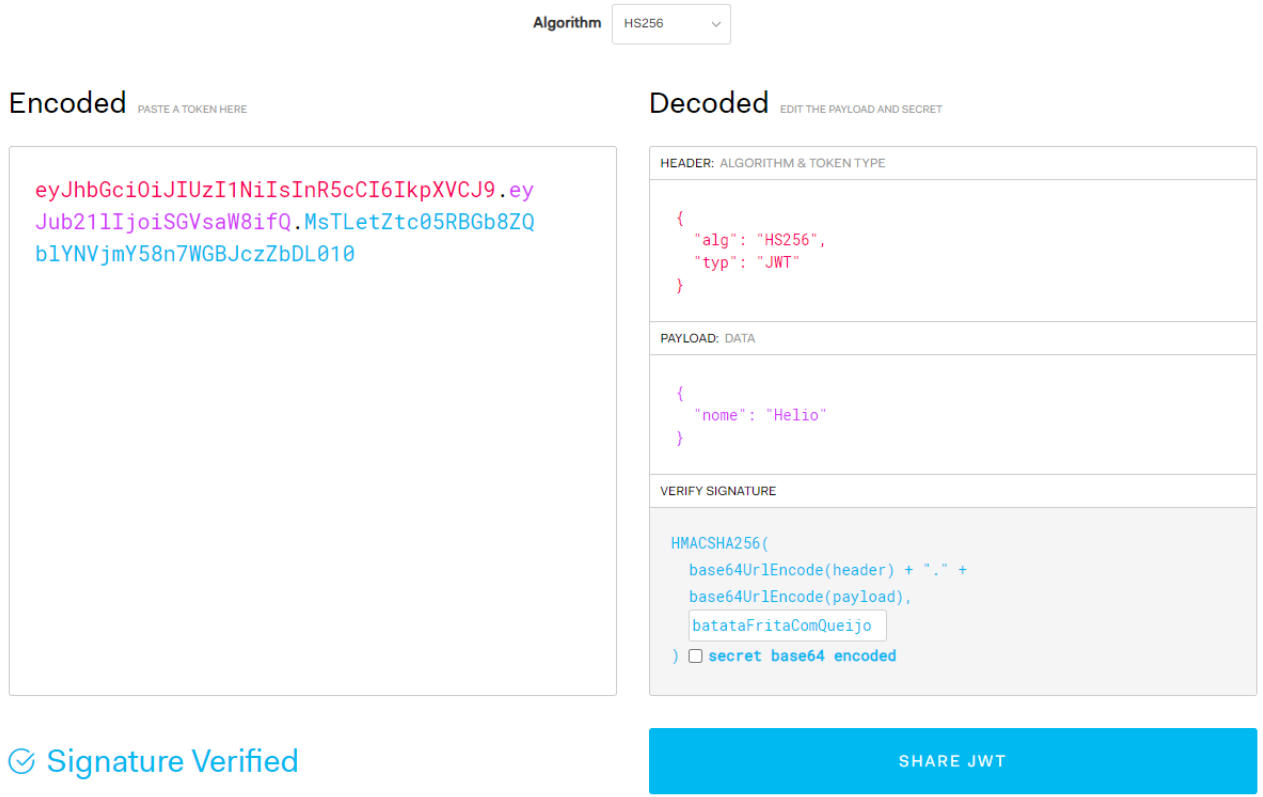
- Uma chave secreta é usada para gerar uma assinatura digital.
- Essa assinatura é calculada aplicando o algoritmo HMAC-SHA256 ao Header e ao payload do JWT, juntamente com a chave secreta.
  - O resultado é uma sequência de bytes que representa a assinatura.

# https://jwt.io/

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

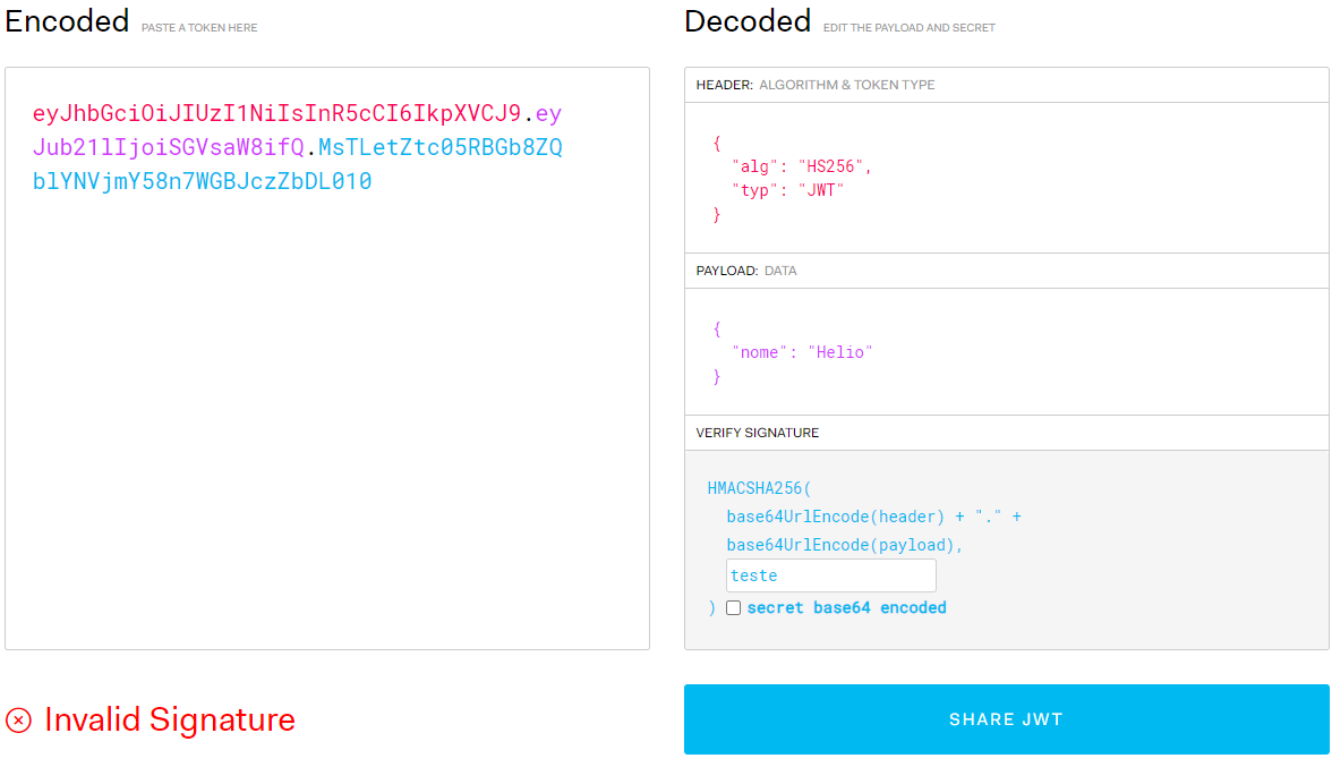
eyJhbGciOiJIUzI1NiIsInR5
cCI6IkpXVCJ9.eyJub21lIjo
iSGVsaW8ifQ.MsTLetZtc0
5RBGb8ZQblYNVjmY58n
7WGBJczZbDL010

Assinatura:batataFritaComQueijo



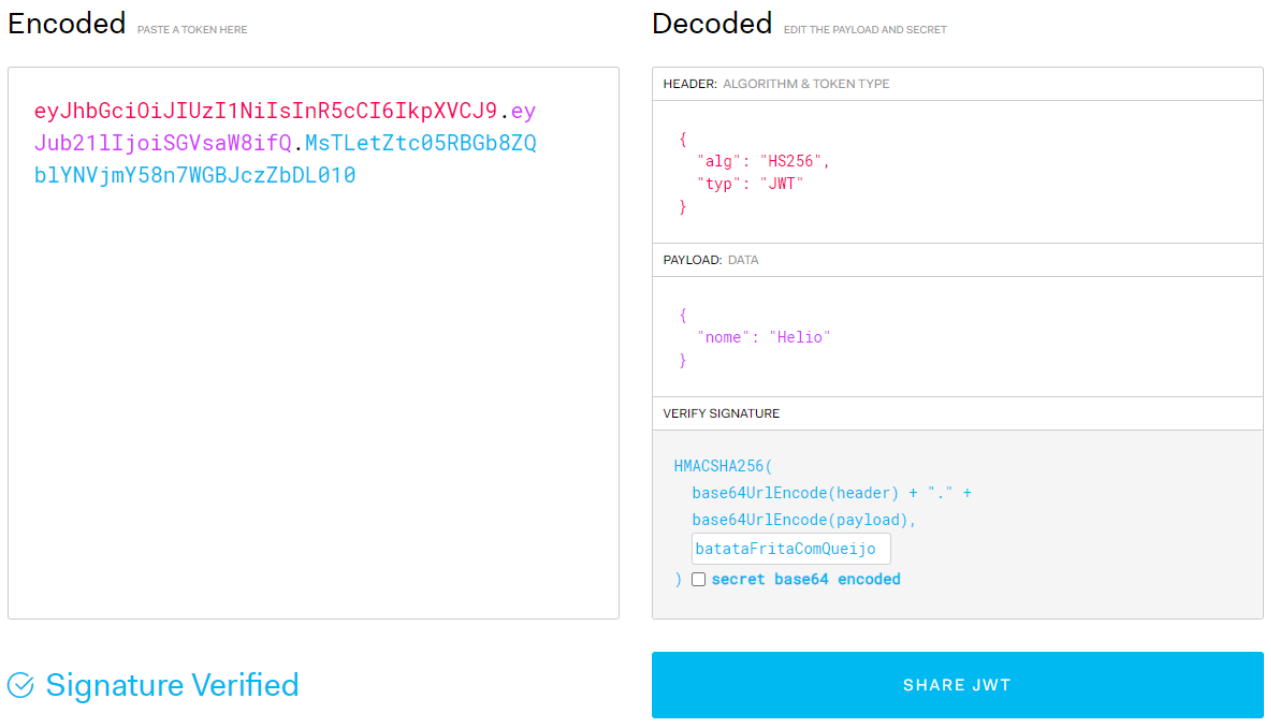
# Observe

- 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

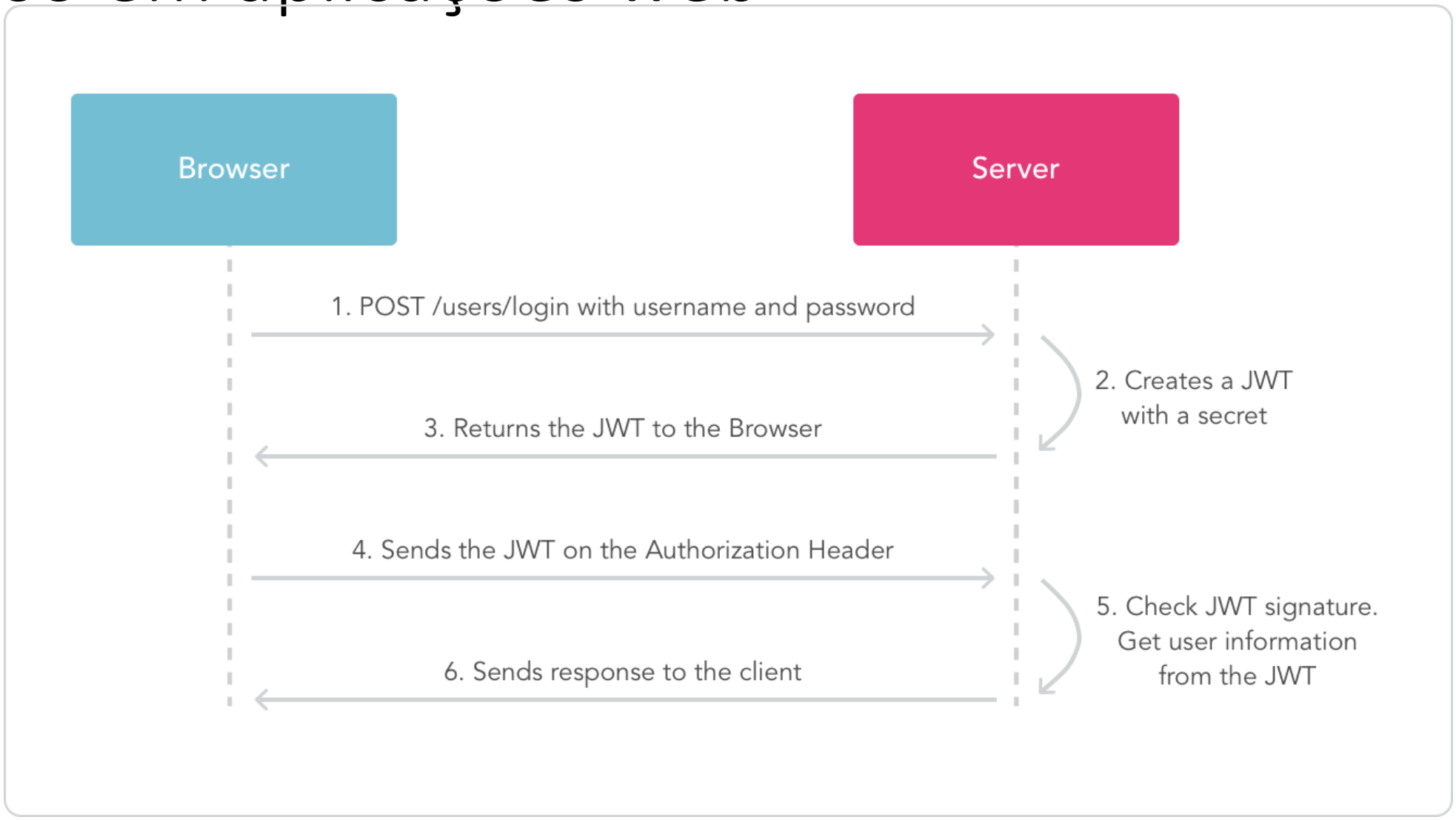
- Com a chave correta a assinatura é válida;



# Tente você

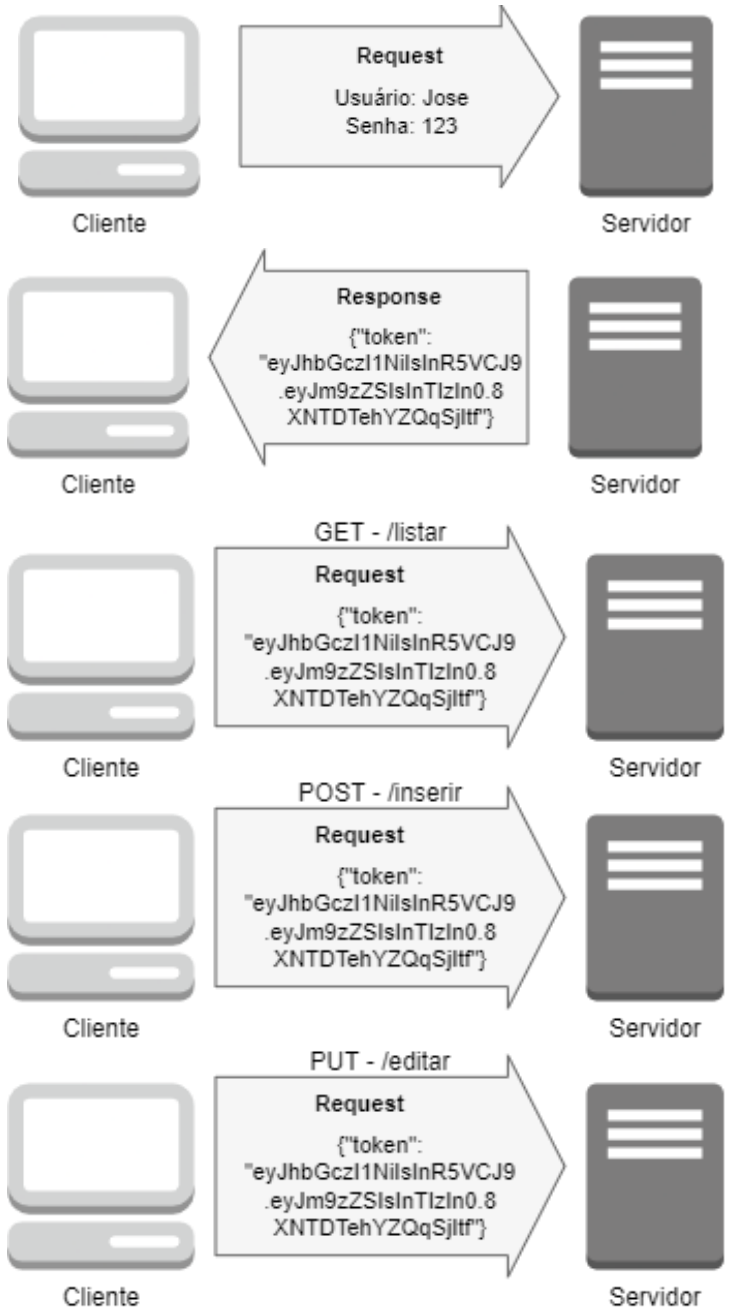
- Abra: https://jwt.io/
- Insira o token:
  - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lIjoiSGVsaW8ifQ.MsTLetZtc05RBGb8ZQblYNVjmY58n7WGBJczZbDL010.
- Insira a chave:
    - batataFritaComQueijo
- Tente outras chaves:
  - Teste007, batata com bacon.
- Verifique que é possível visualizar os dados do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

# 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

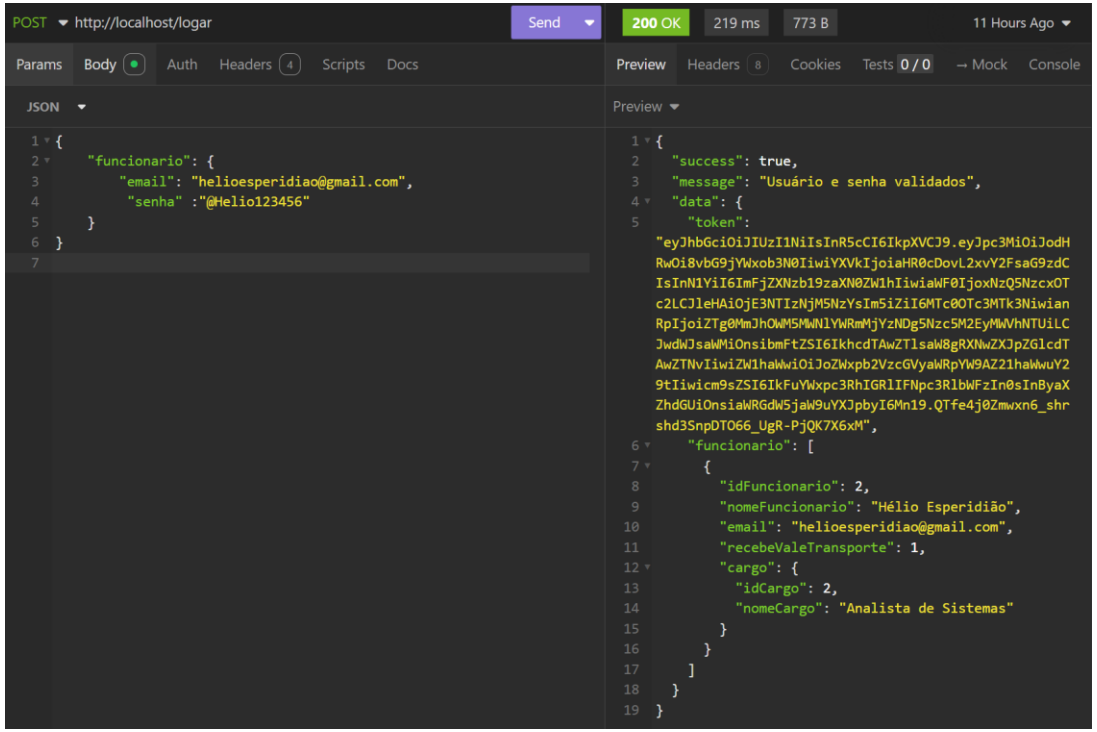
```php
class MeuTokenJWT {
    // Constantes estáticas
    private const KEY = "x9S4q0v+V0IjvHkG20uAxaHx1ijj+q1HWjHKv+ohxp/oK+77qyXkVj/l4QYHHTF3";
    private const ALGORITHM = 'HS256';
    private const TYPE = 'JWT';
    public function __construct(
        private stdClass $payload = new stdClass(),
        private string $iss = 'http://localhost',
        private string $aud = 'http://localhost',
        private string $sub = 'acesso_sistema',
        private int $duration = 3600 * 24 * 30 // 30 dias
    ) { }
```

```php
public function gerarToken(stdClass $claims): string
{
    $objHeaders = new stdClass();
    $objHeaders->alg = MeuTokenJWT::ALGORITHM;
    $objHeaders->typ = MeuTokenJWT::TYPE;
    $objPayload = new stdClass();
    $objPayload->iss = $this->iss;
    $objPayload->aud = $this->aud;
    $objPayload->sub = $this->sub;
    $objPayload->iat = time();
    $objPayload->exp = time() + $this->duration;
    $objPayload->nbf = time();
    $objPayload->jti = bin2hex(random_bytes(16));
    // Public Claims
    $objPayload->public = new stdClass();
    $objPayload->public->name = $claims->name ;
    $objPayload->public->email = $claims->email ;
    $objPayload->public->role = $claims->role ;
    // Private Claims
    $objPayload->private = new stdClass();
    $objPayload->private->idFuncionario = $claims->idFuncionario;
    return JWT::encode(
        payload: (array) $objPayload,
        key: MeuTokenJWT::KEY,
        alg: MeuTokenJWT::ALGORITHM,
        keyId: null,
        head: (array) $objHeaders
    );
}
```

```php
    public function validateToken($stringToken): bool     {
        if (empty($stringToken)) {
            return false;
        }
        //esse padrão verifica se o token tem o padrao "caracteres.caracteres.caracteres";
        $padrao = '/^[A-Za-z0-9-_]+\.[A-Za-z0-9-_]+\.[A-Za-z0-9-_]+$/';
        if (!preg_match($padrao, $stringToken) === 1) {
            return false;
        }
        $token = str_replace(["Bearer ", " "], "", $stringToken);
        try {
            $payloadValido = JWT::decode(jwt: $token, keyOrKeyArray: new Key(keyMaterial: MeuTokenJWT::KEY, algorithm:
MeuTokenJWT::ALGORITHM));
            $this->setPayload($payloadValido);
            return true;
        } catch (
            SignatureInvalidException | \Firebase\JWT\BeforeValidException | ExpiredException |  InvalidArgumentException |
            DomainException |
            UnexpectedValueException |
            Exception $e
        ) {
            return false;
        }
    }
```

```php
public function getPayload(): stdClass|null
{
    return $this->payload;
}

public function setPayload(stdClass $payload): self
{
    $this->payload = $payload;
    return $this;
}
```

# 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
private function setupLoginRoutes()    {
    $this->router->post(pattern: '/logar', fn: function (): never {
        try {
            $requestBody = file_get_contents(filename: "php://input");
            $loginMiddleare = new LoginMiddleware();
            $stdLogin = $loginMiddleare->stringJsonToStdClass($requestBody);
            $loginMiddleare
                ->isValidEmail($stdLogin->login->email)
                ->isvalidSenha($stdLogin->login->senha);
            $LoginControl = new LoginControl();
            $LoginControl->autenticar($stdLogin);
        } catch (Throwable $throwable) {
            $this->sendErrorResponse(
                throwable: $throwable,
                message: 'Erro ao efetuar login'
            );
        }

        exit();
    });
}
```

# LoginControl - 1

```php
    public function login(stdClass $stdLogin): never
    {
        // Cria uma instância do DAO para acessar os dados no banco
        $loginDAO = new LoginDAO();

        $funcionario = new Funcionario();


        $funcionario->setEmail($stdLogin->funcionario->email);
        $funcionario->setSenha($stdLogin->funcionario->senha);

        // Obtém todos os cargos cadastrados
        $funcionarioLogado = $loginDAO->verificarLogin($funcionario);

        if (empty($funcionarioLogado)) {
            // Envia a resposta JSON com os dados encontrados
            (new Response(
                success: false,
                message: 'Usuário e senha inválidos',

                httpCode: 401
            ))->send();
        } else {
```

# LoginControl - 2

```
            $claims = new stdClass();

            $claims->name = $funcionarioLogado->getNomeFuncionario();
            $claims->email = $funcionarioLogado->getEmail();
            $claims->role = $funcionarioLogado->getCargo()->getNomeCargo();
            $claims->idFuncionario = $funcionarioLogado->getIdFuncionario();


            $meuToken = new MeuTokenJWT();

            $token = $meuToken->gerarToken($claims);

            (new Response(
                success: true,
                message: 'Usuário e senha validados',
                 data: [
                    'token' => $token,
                    'funcionario' => $funcionarioLogado
                ],

                httpCode: 200
            ))->send();
        }

        exit();
    }
```

# LoginDAO 1

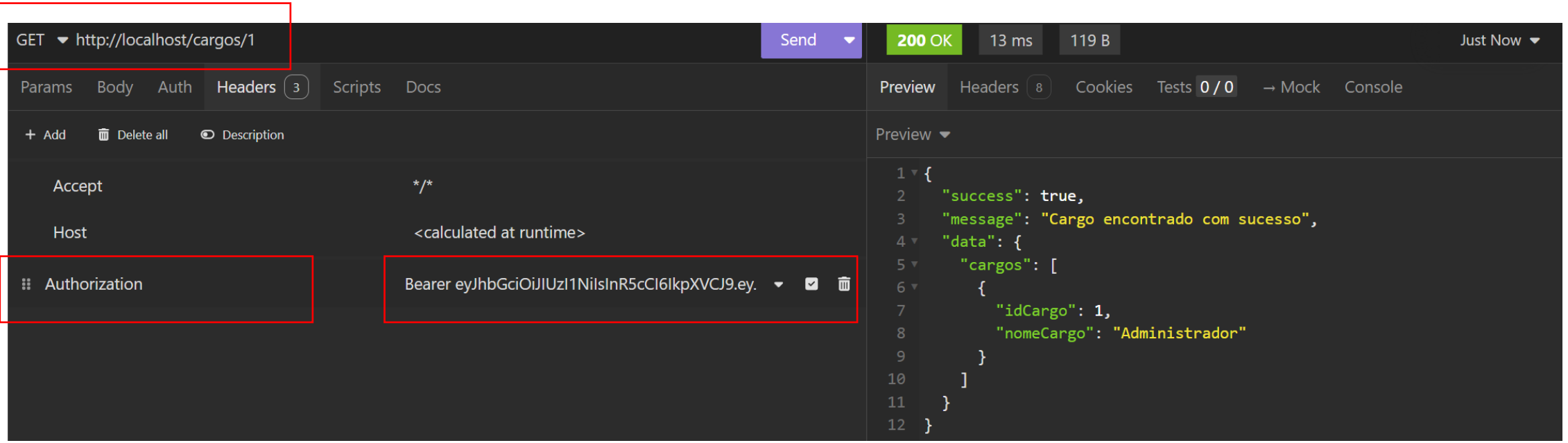
```php
public function login(Funcionario $funcionario): array
   {
       $query = '  SELECT idFuncionario, nomeFuncionario, email, idCargo, nomeCargo FROM funcionario
                   JOIN cargo ON Cargo_idCargo = idCargo
                   WHERE
                       email = :email AND senha = md5(:senha)
                   ORDER BY nomeFuncionario ASC ';

       // Prepara a instrução SQL, protegendo contra SQL Injection
       $statement = Database::getConnection()->prepare(query: $query);

       $statement->bindValue(
           param: ':email',
           value: $funcionario->getEmail(),
           type: PDO::PARAM_STR
       );

       $statement->bindValue(
           param: ':senha',
           value: $funcionario->getSenha(),
           type: PDO::PARAM_STR
       );
       // Busca a única linha esperada da consulta como um objeto genérico (stdClass)
       $statement->execute();
```

# LoginDAO 2

```
$funcionario = new Funcionario();

        // Busca a única linha esperada da consulta como um objeto genérico (stdClass)
        $linha = $statement->fetch(mode: PDO::FETCH_OBJ);

        if (!$linha) {
                return null; // Retorna array vazio caso não encontre nenhum funcionário com esse idFuncionario
        }
        // Preenche os dados básicos do funcionário no objeto Funcionario
        $funcionario
            ->setIdFuncionario(idFuncionario: $linha->idFuncionario)                  // ID do funcionário
            ->setNomeFuncionario(nomeFuncionario: $linha->nomeFuncionario)            // Nome do funcionário
            ->setEmail(email: $linha->email);                                          // E-mail


        // Acessa o objeto Cargo dentro do Funcionario e preenche os dados do cargo
        $funcionario
            ->getCargo()
            ->setIdCargo(idCargo: $linha->idCargo)              // ID do cargo associado
            ->setNomeCargo(nomeCargo: $linha->nomeCargo);       // Nome do cargo

        return $funcionario;
    }
```

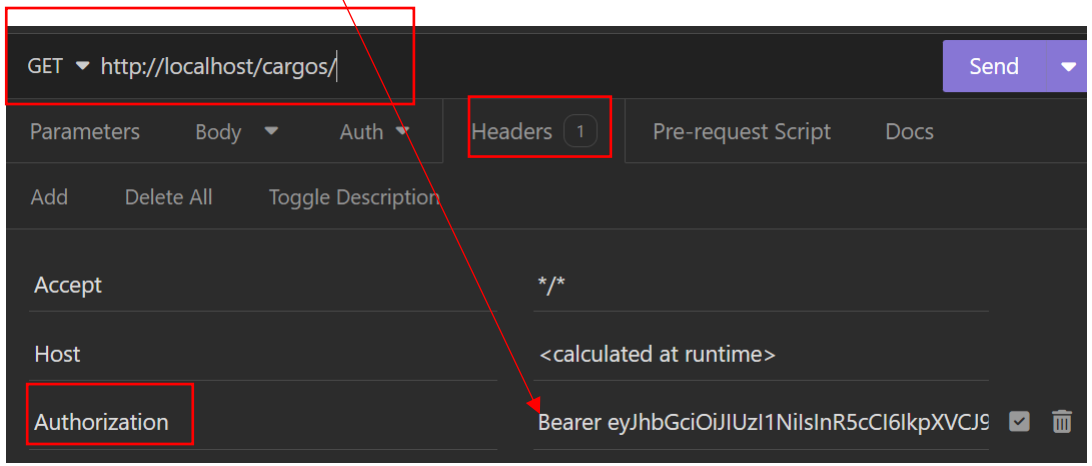
# O que fazer com o token?

- Ao confirmar o login o solicitante recebe um token.
- O token deve ser enviado junto a todas as requisições
- A aplicação php deve validar o token enviado, caso o token seja válido a solicitação é concluída.



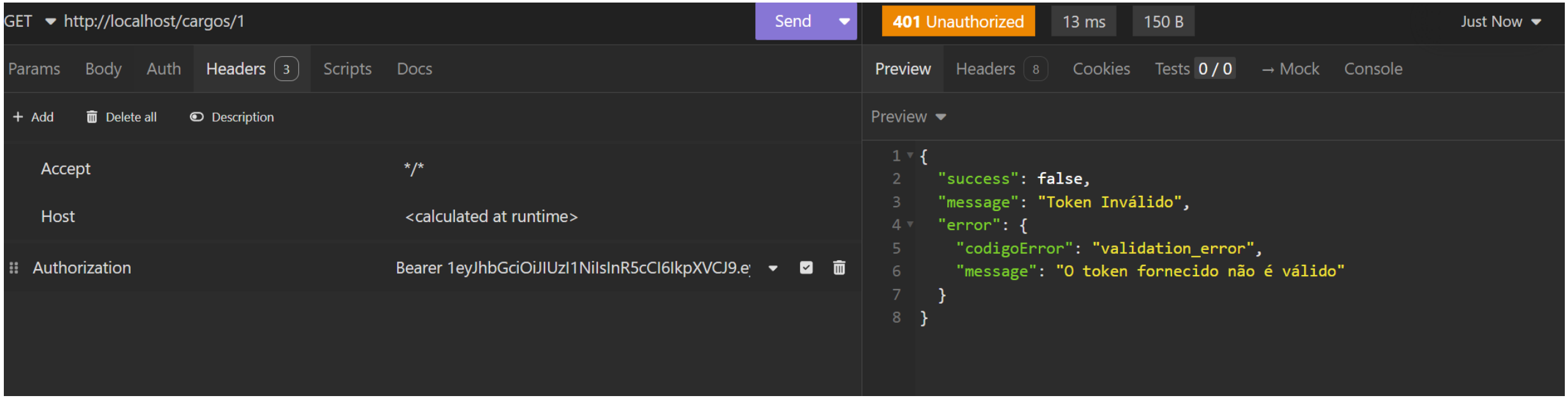
# Adicione o atributo Authorization fornecer o valor: Bearer <token>

- 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.

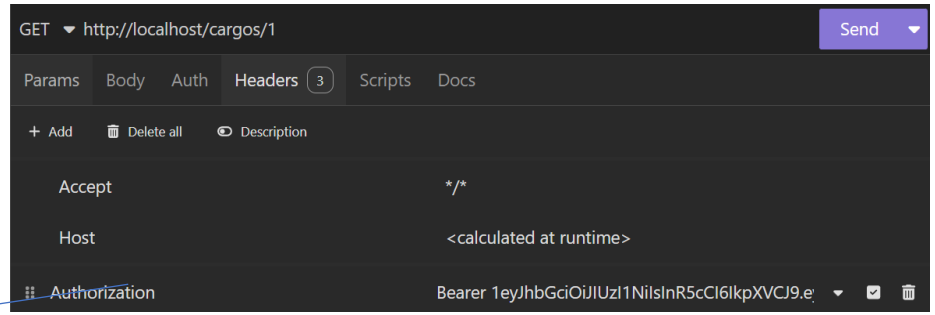

GET http://localhost/cargos/1
Send
401 Unauthorized
13 ms
150 B
Just Now
Params
Body
Auth
Headers 3
Scripts
Docs
Preview
Headers 8
Cookies
Tests 0 / 0
→ Mock
Console
+ Add
Delete all
Description
Preview
Accept */*
Host <calculated at runtime>
Authorization Bearer 1eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e
{
"success": false,
"message": "Token Inválido",
"error": {
"codigoError": "validation_error",
"message": "O token fornecido não é válido"
}
}

# Como fazer a validação?

- Crie um middleware para receber e validar o token
- A validação do token deve ser o primeiro middleware a ser executado

O método getAuthorizationHeader() recuperar o valor enviado do authorization enviado pelo insominia

JWTMiddlware.php




```php
class JWTMiddlware{

    function getAuthorizationHeader()
    {
        $headers = null;

        if (isset($_SERVER['Authorization'])) {
            $headers = trim($_SERVER["Authorization"]);
        } elseif (isset($_SERVER['HTTP_AUTHORIZATION'])) {
            // Para Nginx ou servidores que passam como HTTP_*
            $headers = trim($_SERVER["HTTP_AUTHORIZATION"]);
        } elseif (function_exists('apache_request_headers')) {
            $requestHeaders = apache_request_headers();
            // Faz o header ser insensível a maiúsculas/minúsculas
            $requestHeaders = array_change_key_case($requestHeaders, CASE_LOWER);
            if (isset($requestHeaders['authorization'])) {
                $headers = trim($requestHeaders['authorization']);
            }
        }

        return $headers;
    }
```

JWTMiddlware.php

```php
public function isValidToken(): stdClass
{
    $token = $this->getAuthorizationHeader();
    //verifica se existe algum token enviado
    if (!isset($token)) {
        (new Response(
            success: false,
            message: 'Token Inválido',
            error: [
                'codigoError' => 'validation_error',
                'message' => 'Não foi fornecido um token',
            ],
            httpCode: 401
        ))->send();
        exit();
    }
```

JWTMiddlware.php

```php
$Jwt = new MeuTokenJWT();
  //verifica se o token é valido
  if ($Jwt-> validateToken(stringToken: $token) == true) {
      return $Jwt->getPayload(); //retorna o payload publico e privado
  } else {
      (new Response(
          success: false,
          message: 'Token Inválido',
          error: [
              'codigoError' => 'validation_error',
              'message' => 'O token fornecido não é válido',
          ],
          httpCode: 401
      ))->send();
      exit();
  }
```

# E no roteador?

```
$this->router->get(pattern: "/cargos/(\d+)", fn: function ($idCargo): never {
    try {

        $tokenMiddleware = new JWTMiddlware();
        $payload = $tokenMiddleware->isValidToken();

        $cargoMiddleware = new CargoMiddleware();
        $cargoMiddleware
            ->isValidId(idCargo: $idCargo);

        $cargoControl = new CargoControl();
        $cargoControl->show(idCargo: $idCargo);

    } catch (Throwable $throwable) {

        $this->sendErrorResponse(
            throwable: $throwable,
            message: 'Erro na seleção de dados'
        );
    }

    exit();
});
```

```php
$this->router->post(pattern: "/cargos", fn: function (): never {
    try {
        $ claims = (new JWTMiddlware())-> $tokenMiddleware->isValidToken();
        if ($claims->public->role == "Administrador") {//verifica se o token possui o claim publica role igual a administrador
            $requestBody = file_get_contents(filename: "php://input");
            $cargoMiddleware = new CargoMiddleware();
            $stdCargo = $cargoMiddleware->stringJsonToStdClass(requestBody: $requestBody);
            $cargoMiddleware
                ->isValidNomeCargo(nomeCargo: $stdCargo->cargo->nomeCargo) // Valida o nome do cargo
                ->hasNotCargoByName(nomeCargo: $stdCargo->cargo->nomeCargo); // Verifica se o cargo já existe no banco
            $cargoControl = new CargoControl();
            $cargoControl->store(stdCargo: $stdCargo);
       } else {
            (new Response(
                success: false,
                message: 'Você não possui autorização para executar a operação',
                error: ['codigoError' => 'validation_error’,       'message' => 'Credencial de acesso inválida',  ],
                httpCode: 401
            ))->send();
        }
    } catch (Throwable $throwable) {
            $this->sendErrorResponse(throwable: $throwable,message: 'Erro ao inserir um novo cargo');
    }
});
```