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.

JWT

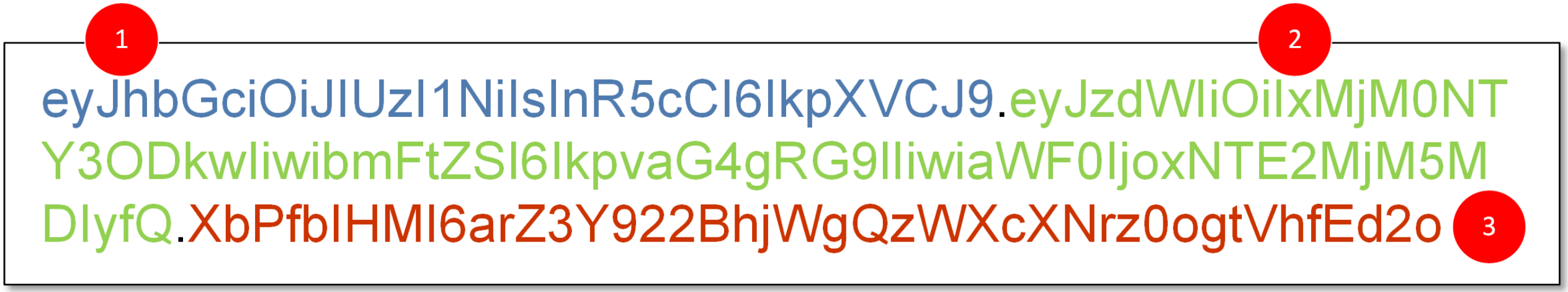
- É um padrão utilizado para **autenticação e troca segura de informações** entre cliente e servidor. Em aplicações web, normalmente é usado para manter o usuário autenticado após o login.
- **Como funciona**
  - O usuário faz login com usuário e senha.
  - O servidor valida as credenciais.
  - O servidor gera um JWT e envia para o cliente.
  - O cliente armazena esse token (localStorage, sessionStorage ou cookie).
  - A cada requisição protegida, o cliente envia o token.
  - O servidor valida o token e autoriza o acesso.

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.



1 Header

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

2 Payload

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

3 Signature

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

Simplificando

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

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lIjoiSGVsaW8ifQ.MsTLetZtc05RBGb8ZQblYNVjmY58n7WGBJczZbDL010

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:

- **iss** (issuer): Identifica quem emitiu o token.
- **sub** (subject): Identifica o assunto do token (geralmente o usuário).
- **aud** (audience): Destinatário do token.
- **exp** (expiration time): Tempo de expiração do token.
- **nbf** (not before): O token não é válido antes desta data.
- **iat** (issued at): Data em que o token foi emitido.
- **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***

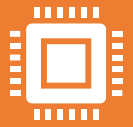
```
{
    "iss": "https://example.com", // Registered Claim: Emissor do token
    "sub": "1234567890", // Registered Claim: Identificador do sujeito (usuário)
    "aud": "https://your-app.com", // Registered Claim: Destinatário do token
    "exp": 1722470421, // Registered Claim: Tempo de expiração
    "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

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.

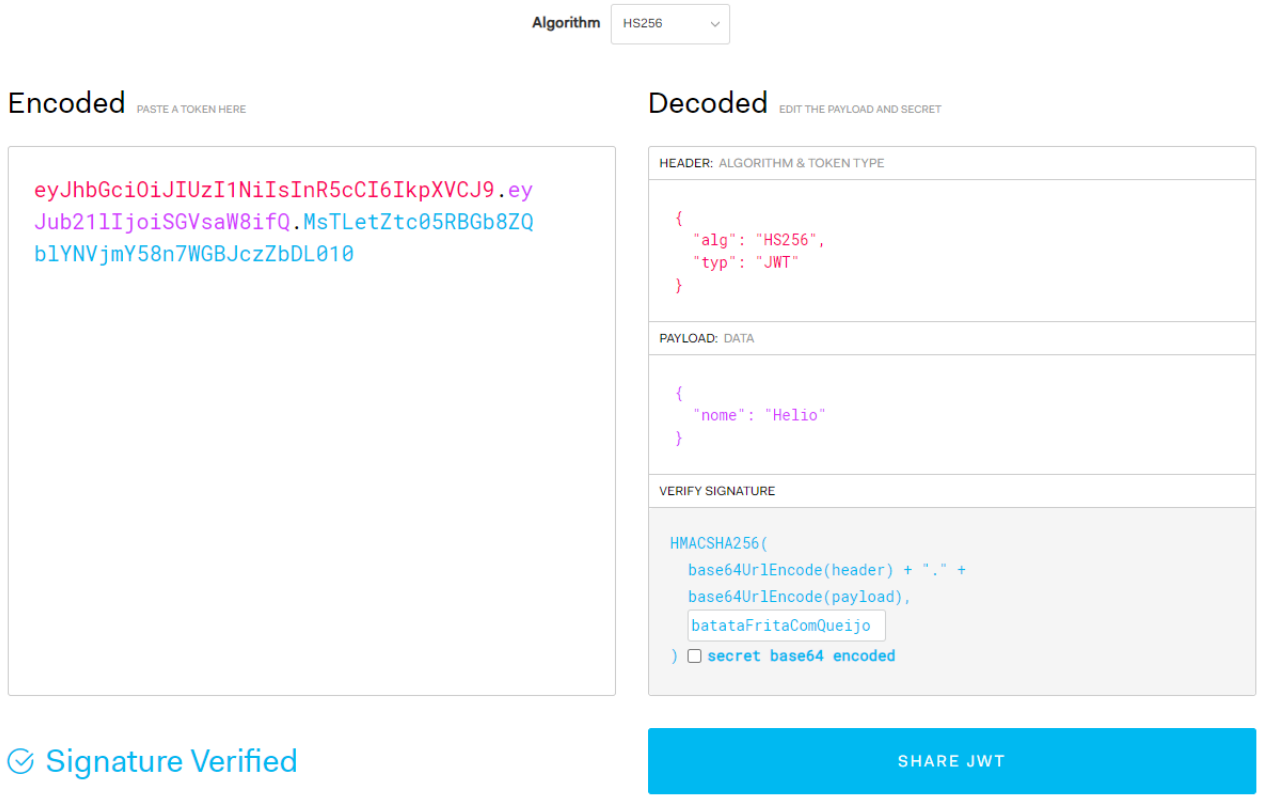
Uma chave secreta é usada para gerar uma assinatura digital.

https://jwt.io/

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

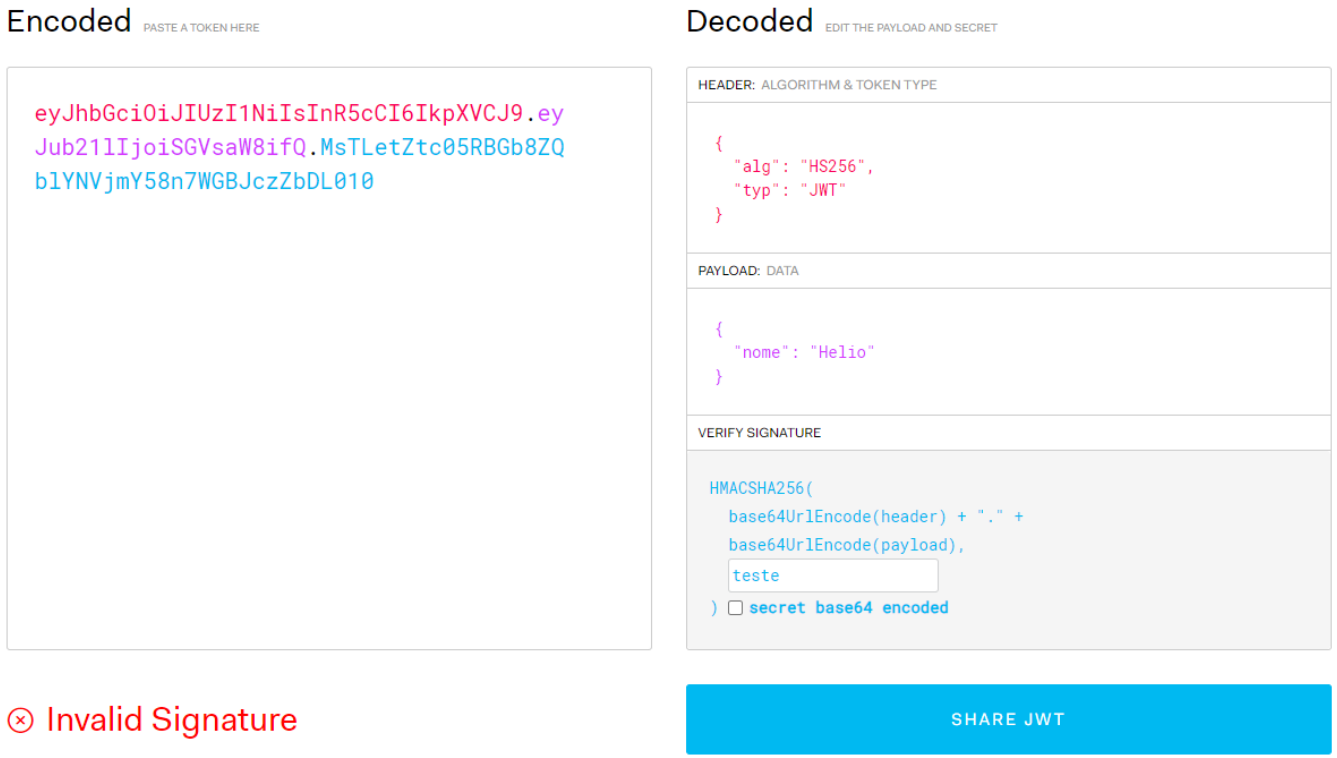
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lIjoiSGVsaW8ifQ.MsTLetZtc05RBGb8ZQblYNVjmY58n7WGBJczZbDL010

Assinatura: batataFritaComQueijo



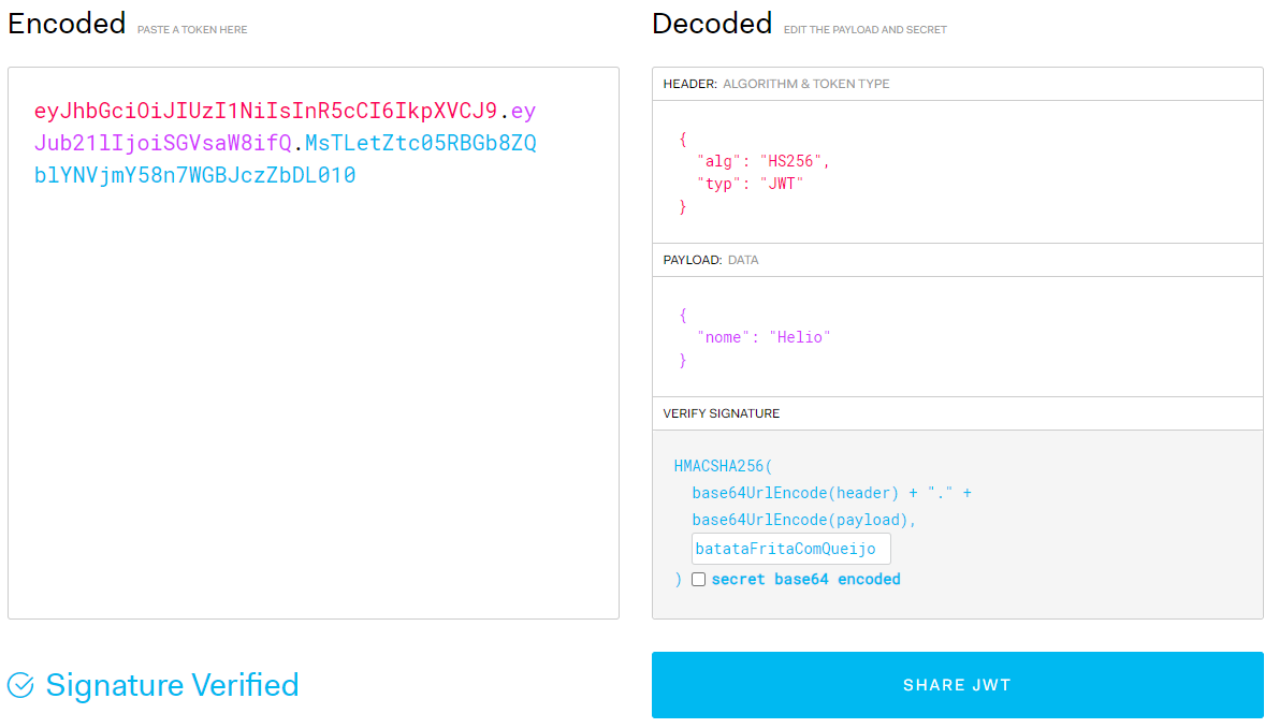
Observe

- 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

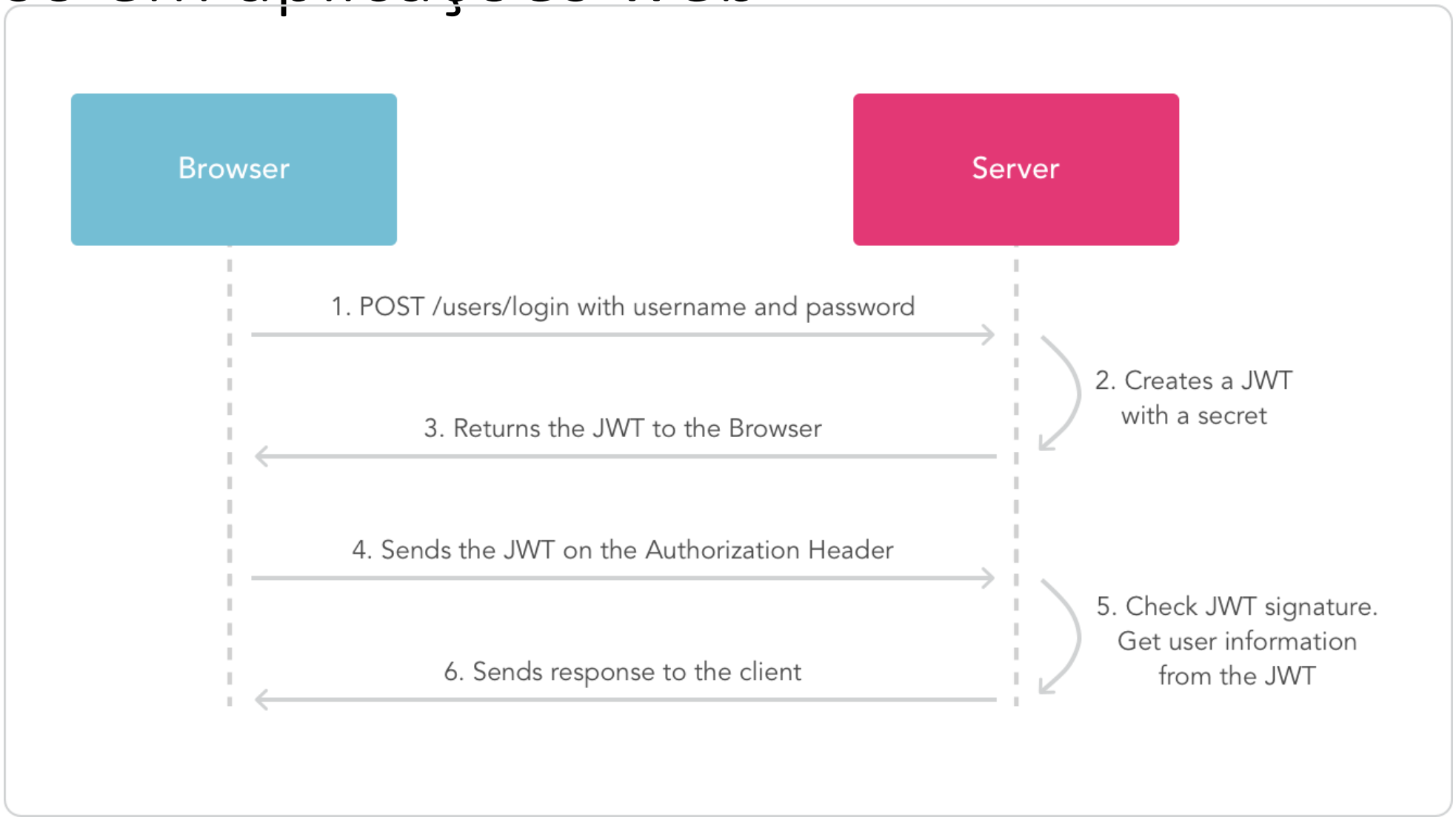
- Com a chave correta a assinatura é válida;



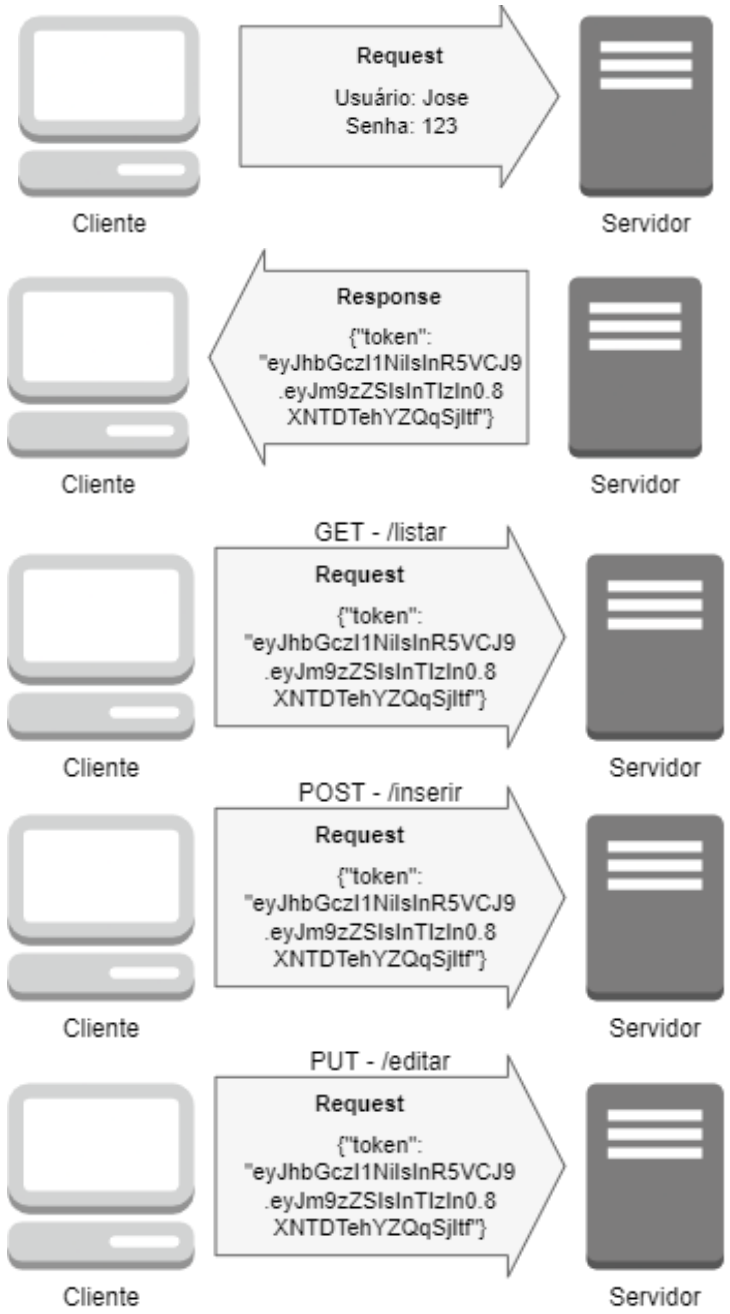
Tente você

- 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.

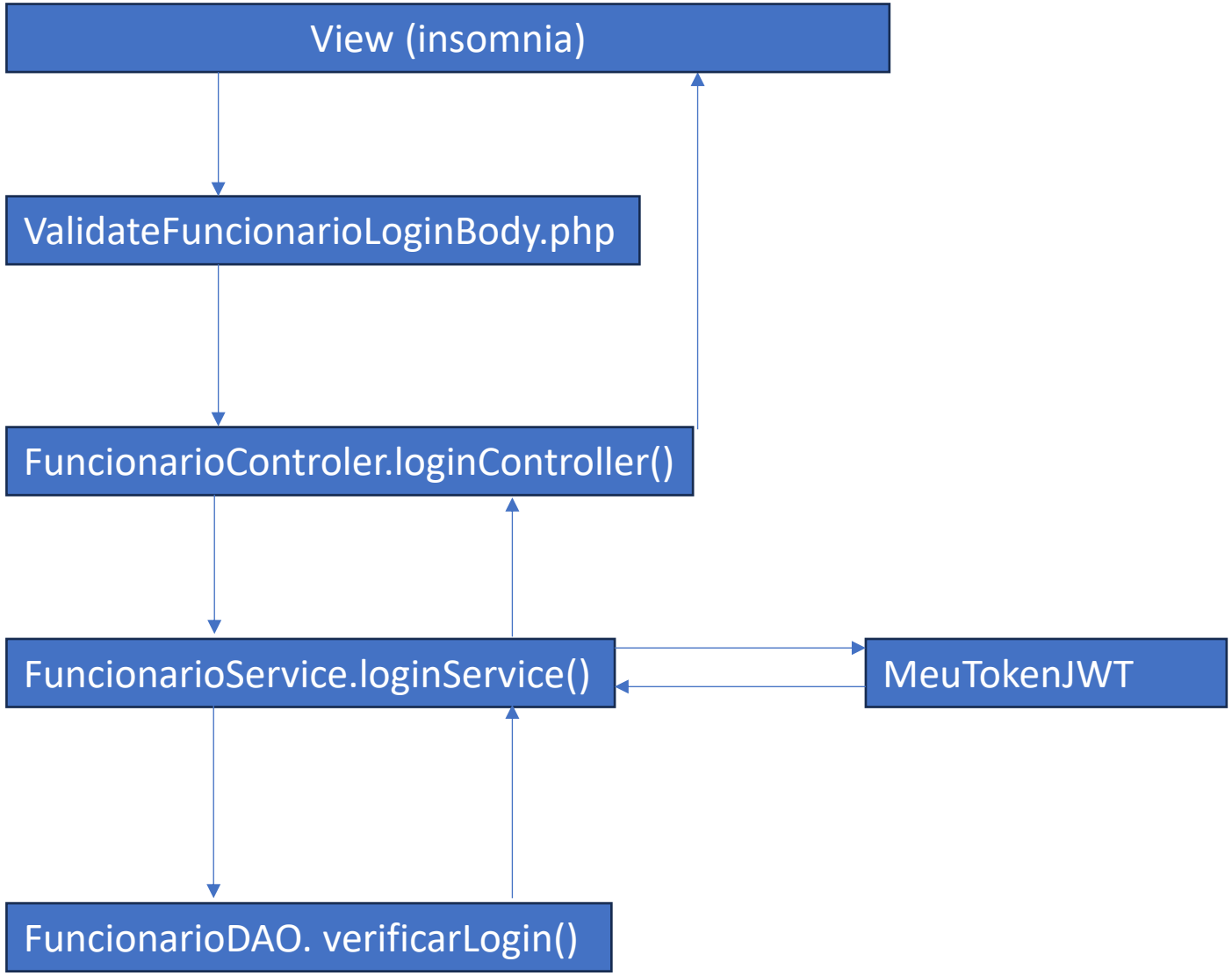
Uso em aplicações web



Arquitetura



Fluxo de login



Implementação

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

  - Para instalar utilize o composer
    - composer require firebase/php-jwt
  - No material de aula utilize o template que já está configurado

MeuTokenJWT.php

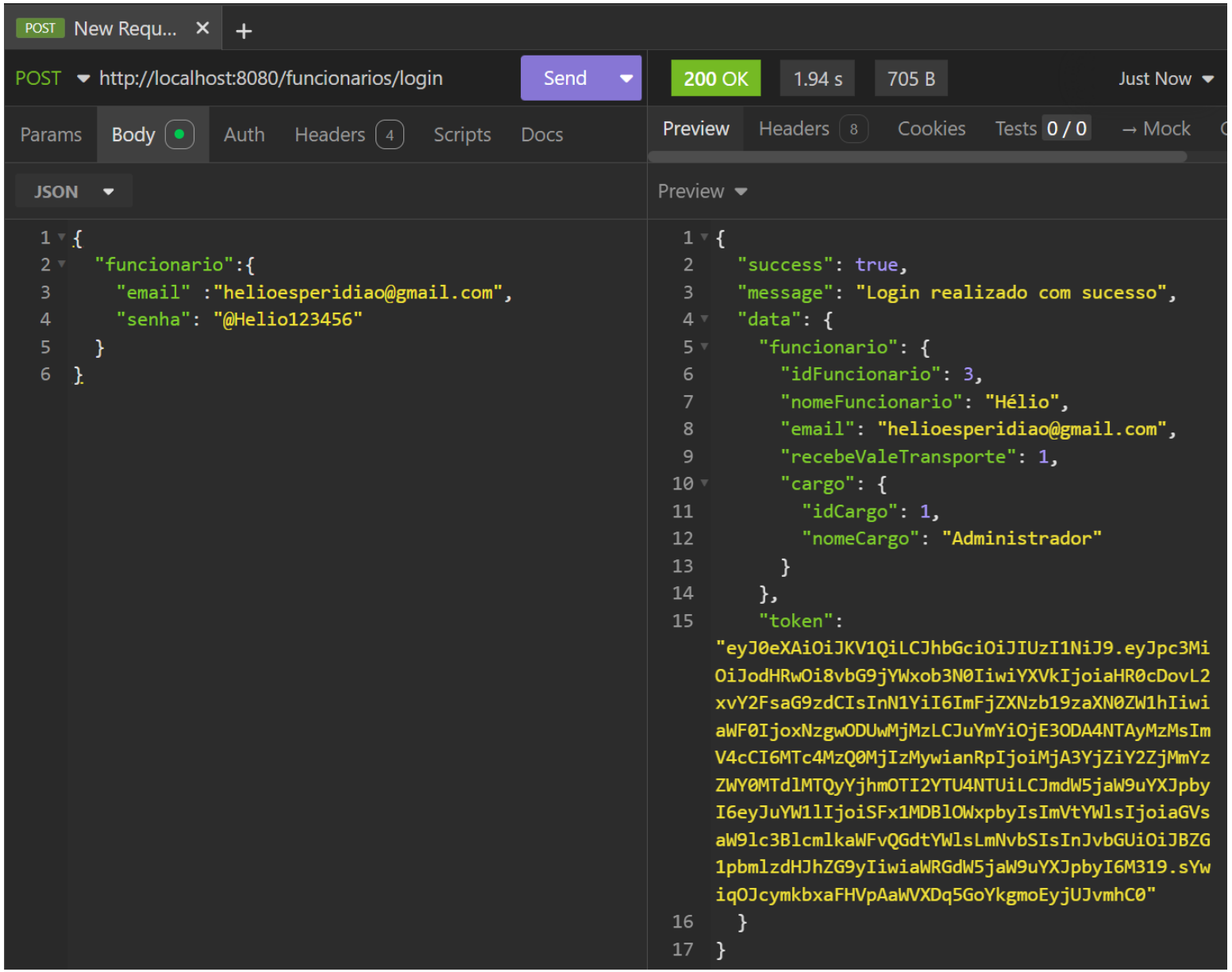
- Nessa classe modifique apenas os atributos necessários para a sua aplicação.
- Não modificar essa classe se não estiver certo do que está fazendo.

```php
namespace Api\Http;

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

```php
public function gerarToken(stdClass $claims): string {
        $headers = [
            'alg' => self::ALGORITHM,
            'typ' => self::TYPE
        ];

        $payload = [
            'iss' => $this->iss,
            'aud' => $this->aud,
            'sub' => $this->sub,
            'iat' => time(),
            'nbf' => time(),
            'exp' => time() + $this->duration,
            'jti' => bin2hex(random_bytes(16)),
            'funcionario' => [
                'name' => $claims->name ?? null,
                'email' => $claims->email ?? null,
                'role' => $claims->role ?? null,
                'idFuncionario' => $claims->idFuncionario ?? null
            ],
        ];

        return JWT::encode($payload, self::KEY, self::ALGORITHM, null, $headers);

    }
```

```php
public function validateToken(string $stringToken): bool {
        if (empty($stringToken)) {
            return false;
        }
        $token = trim($stringToken);
        if (str_starts_with($token, 'Bearer ')) {
            $token = substr($token, 7);
        }
        $padrao = /^[A-Za-z0-9\-_]+\.[A-Za-z0-9\-_]+\.[A-Za-z0-9\-_]+$/';
        if (preg_match($padrao, $token) !== 1) {
            return false;
        }
        try {
            $payloadValido = JWT::decode($token, new Key(self::KEY, self::ALGORITHM));
            if (!isset($payloadValido->iss) || $payloadValido->iss !== $this->iss) {
                return false;
            }
            if (!isset($payloadValido->aud) || $payloadValido->aud !== $this->aud) {
                return false;
            }
            if (!isset($payloadValido->sub) || $payloadValido->sub !== $this->sub) {
                return false;
            }
            $this->payload = $payloadValido;
            return true;
        } catch (     SignatureInvalidException |BeforeValidException |
            ExpiredException |InvalidArgumentException |DomainException |
            UnexpectedValueException |Exception $e
        ) {  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

Copie e cole no jwt.io

- eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0IiwiYXVkIjoiaHR0cDovL2xvY2FsaG9zdCIsInN1YiI6ImFjZXNzb19zaXN0ZW1hIiwiaWF0IjoxNzgwODUwMjMzLCJuYmYiOjE3ODA4NTAyMzMsImV4cCI6MTc4MzQ0MjIzMywianRpIjoiMjA3YjZiY2ZjMmYzZWY0MTdlMTQyYjhmOTI2YTU4NTUiLCJmdW5jaW9uYXJpbyI6eyJuYW1lIjoiSFx1MDBlOWxpbyIsImVtYWlsIjoiaGVsaW9lc3BlcmlkaWFvQGdtYWlsLmNvbSIsInJvbGUiOiJBZG1pbmlzdHJhZG9yIiwiaWRGdW5jaW9uYXJpbyI6M319.sYwiqOJcymkbxaFHVpAaWVXDq5GoYkgmoEyjUJvmhC0

Programando a rota /login

```php
public function setupRoutes(): void {

    $this->app->post(
        '/funcionarios/login',
         [$this->controller, 'loginController']
    )->add(ValidateFuncionarioLoginBody::class);
}
```

Middleware: ValidateFuncionarioLoginBody

```php
public function process(Request $request, RequestHandler $handler): Response {
    $body = $request->getBody()->getContents();
    $objPHP = json_decode($body);
    if (!isset($objPHP->funcionario)) {
        throw new ErrorResponse(
            httpCode: 400,
            message: "Erro na validação de dados",
            error: ["message" => "O campo 'funcionario' é obrigatório!"]);
    }
    $funcionario = $objPHP->funcionario;
    if (!isset($funcionario->email) || empty(trim($funcionario->email))) {
        throw new ErrorResponse(
            httpCode: 400,
            message: "Erro na validação de dados",
            error: ["message" => "O campo 'email' é obrigatório!"]);
    }if (!filter_var($funcionario->email,FILTER_VALIDATE_EMAIL)) {
        throw new ErrorResponse(
            httpCode: 400,
            message: "Erro na validação de dados",
            error: ["message" => "Email inválido!"]);
    } if (!isset($funcionario->senha) || empty(trim($funcionario->senha))    ) {
        throw new ErrorResponse(
            httpCode: 400,
            message: "Erro na validação de dados",
            error: [
                "message" => "O campo 'senha' é obrigatório!"
            ]
        );
    }
    return $handler->handle($request);
}
```

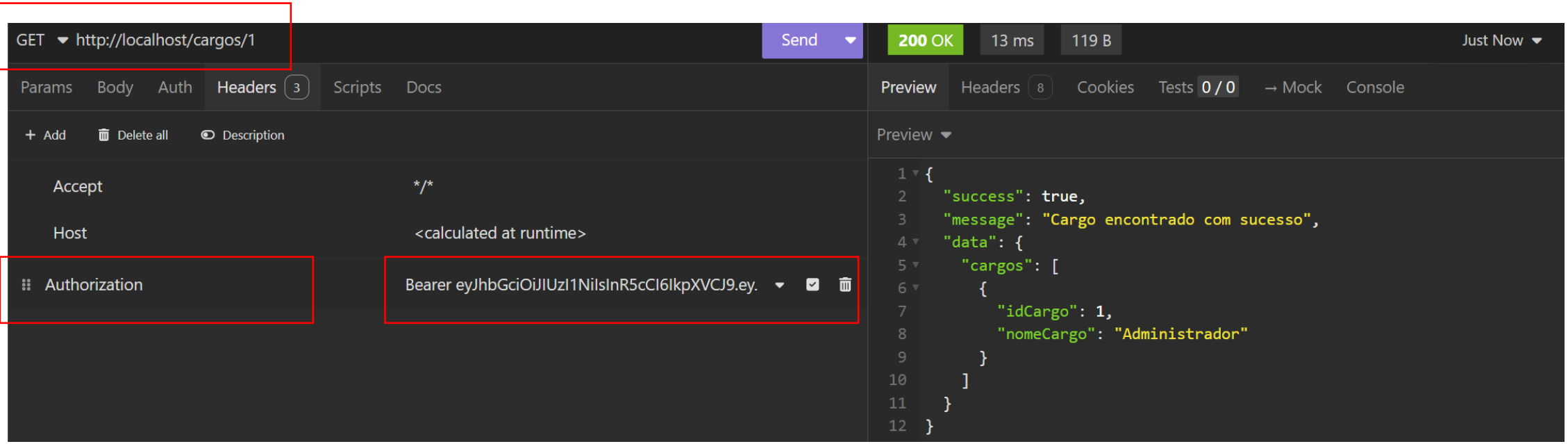
```php
public function loginController(Request $request, Response $response, array $args): Response  {
        $body = $request->getBody()->getContents();
        $objPHP = json_decode($body, true);
        $resultado = $this->funcionarioService->loginService($objPHP['funcionario']);
        $funcionario = $resultado['funcionario'];
        $token = $resultado['token'];
        $resposta = [
            'success' => true,
            'message' => 'Login realizado com sucesso',
            'data' => [
                'funcionario' => [
                    'idFuncionario' => $funcionario->getIdFuncionario(),
                    'nomeFuncionario' => $funcionario->getNomeFuncionario(),
                    'email' => $funcionario->getEmail(),
                    'recebeValeTransporte' => $funcionario->getRecebeValeTransporte(),
                    'cargo' => [
                        'idCargo' => $funcionario->getCargo()->getIdCargo(),
                        'nomeCargo' => $funcionario->getCargo()->getNomeCargo()
                    ]
                ],
                'token' => $token
            ]
        ];
        $response->getBody()->write(json_encode($resposta));
        return $response
            ->withHeader('Content-Type', 'application/json')
            ->withStatus(200);
    }
```

```php
public function loginService(array $jsonFuncionario): array {
        $funcionario = new Funcionario();
        $funcionario->setEmail($jsonFuncionario['email']);
        $funcionario->setSenha($jsonFuncionario['senha']);
        $funcionario = $this->funcionarioDAO->verificarLogin($funcionario);
        if (!$funcionario) {
            throw new ErrorResponse(
                401,
                "Usuário ou senha inválidos",
                [
                    "message" =>
                        "Não foi possível autenticar o funcionário"
                ]
            );
        }
        $jwt = new MeuTokenJWT();
        $claims = new \stdClass();
        $claims->idFuncionario = $funcionario->getIdFuncionario();
        $claims->name = $funcionario->getNomeFuncionario();
        $claims->email = $funcionario->getEmail();
        $claims->role = $funcionario->getCargo()?->getNomeCargo();
        $token = $jwt->gerarToken($claims);
        return [
            'funcionario' => $funcionario,
            'token' => $token
        ];
    }
```

```php
public function verificarLogin(Funcionario $funcionario): ?Funcionario {
        $sql = "SELECT * FROM funcionário JOIN cargo ON idCargo = Cargo_idCargo
        WHERE email = :email LIMIT 1";

        $pdo = $this->database->getConnection();
        $stmt = $pdo->prepare($sql);
        $stmt->execute([
            ':email' => $funcionario->getEmail()
        ]);
        $row = $stmt->fetch(PDO::FETCH_ASSOC);
        if (!$row) {
            return null;
        }
        if (!password_verify($funcionario->getSenha(), $row['senha'])) {
            return null;
        }
        $cargo = new Cargo();
        $cargo->setIdCargo((int) $row['idCargo']);
        $cargo->setNomeCargo($row['nomeCargo']);
        $funcionarioAutenticado = new Funcionario();
        $funcionarioAutenticado->setIdFuncionario((int) $row['idFuncionario']);
        $funcionarioAutenticado->setNomeFuncionario($row['nomeFuncionario']);
        $funcionarioAutenticado->setEmail($row['email']);
        $funcionarioAutenticado->setRecebeValeTransporte((int) $row['recebeValeTransporte']);
        $funcionarioAutenticado->setCargo($cargo);
        return $funcionarioAutenticado;
    }
```

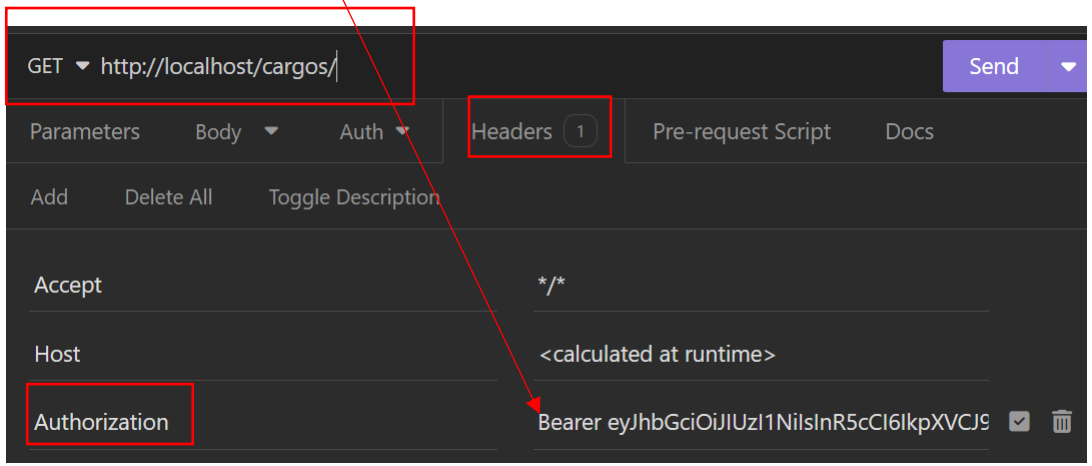
O que fazer com o token?

- 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.



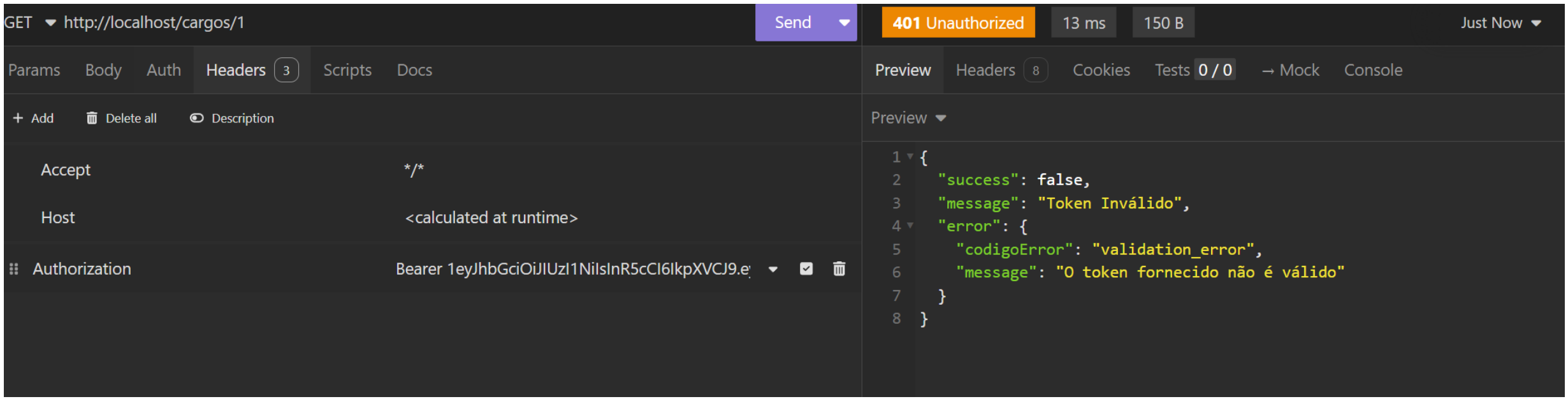
Adicione o atributo Authorization fornecer o valor: Bearer <token>

- 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.

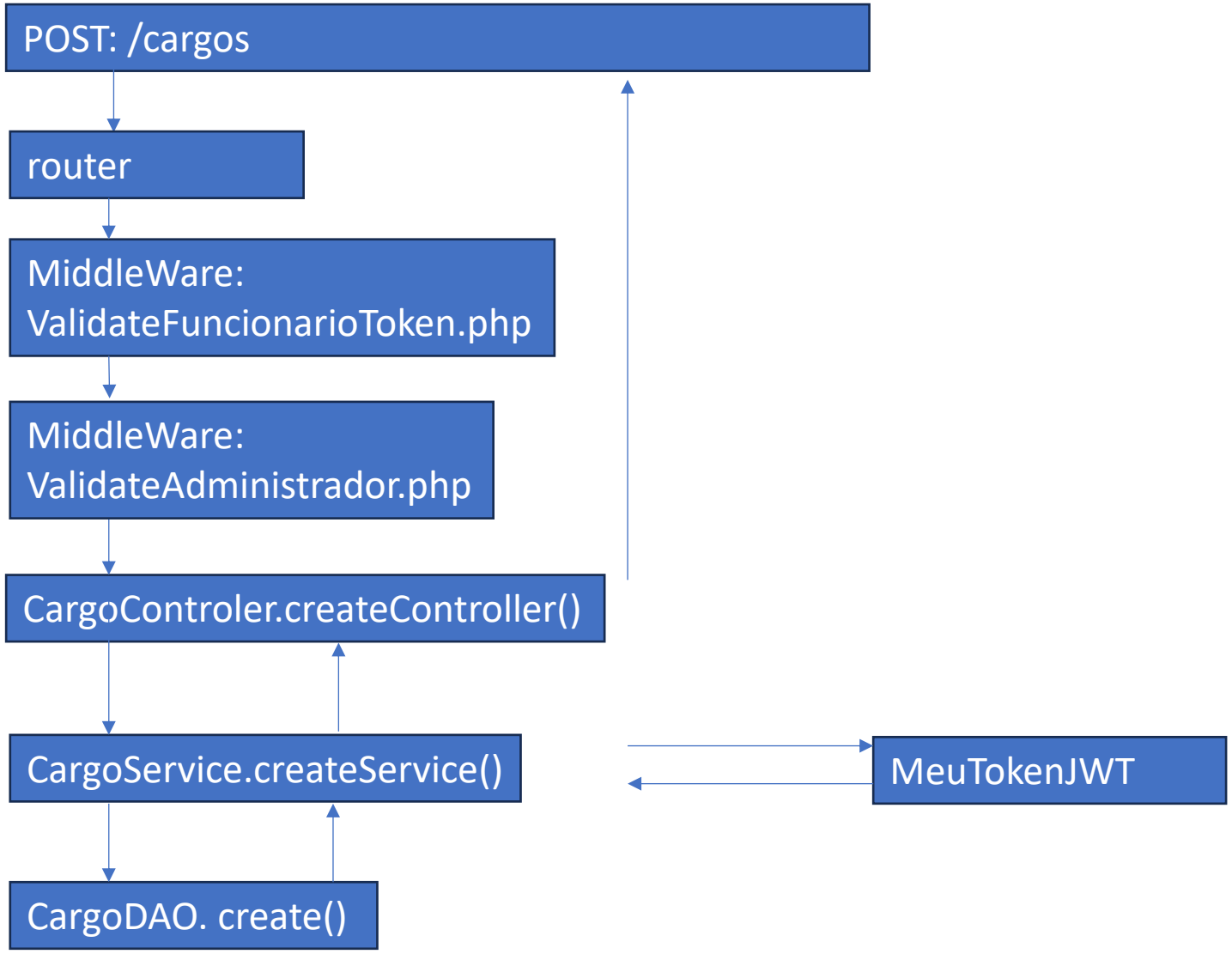

GET http://localhost/cargos/1
Send
401 Unauthorized
13 ms
150 B
Just Now
Params
Body
Auth
Headers 3
Scripts
Docs
Preview
Headers 8
Cookies
Tests 0 / 0
→ Mock
Console
+ Add
Delete all
Description
Preview
Accept */*
Host <calculated at runtime>
Authorization Bearer 1eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e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

Regras de funcionamento

- Para usar qualquer rota é necessário o envio do token
- Para excluir, atualizar e criar é necessário ser um administrador.

Fluxo para uso da API

Todos as rotas devem enviar o token



Router

```php
        $this->app->post(
            '/cargos',
            [CargoController::class, 'createController']
        )
            ->add(ValidateCargoBody::class)
            ->add(ValidateAdministrador::class)
            ->add(ValidateFuncionarioToken::class);
```

ValidateFuncionarioToken

```php
public function process(Request $request, RequestHandler $handler): Response{
        $authorization = $request->getHeaderLine('Authorization');
        if (empty($authorization)) {
            throw new ErrorResponse(
                httpCode: 401,
                message: "Acesso não autorizado",
                error: ["message" => "Token de autenticação não informado"]
            );
        }
        if (!str_starts_with($authorization, 'Bearer ')) {
            throw new ErrorResponse(
                httpCode: 401,
                message: "Acesso não autorizado",
                error: ["message" => "Formato do token inválido"]
            );
        }
        $jwt = new MeuTokenJWT();
        if (!$jwt->validateToken($authorization)) {
            throw new ErrorResponse(
                httpCode: 401,
                message: "Acesso não autorizado",
                error: ["message" => "Token inválido ou expirado"]
            );
        }
        //adiciona o payload na requisição
        $request = $request->withAttribute('jwtPayload', $jwt->getPayload());
        return $handler->handle($request);
    }
```

ValidateAdministrador

```php
public function process(Request $request, RequestHandler $handler): Response
    {
        $payload = $request->getAttribute('jwtPayload');

        if (!$payload) {
            throw new ErrorResponse(
                401,
                "Acesso não autorizado",
                ["message" => "Usuário não autenticado"]
            );
        }

        if (!isset($payload->funcionario->role) || $payload->funcionario->role !== 'Administrador' ) {
            throw new ErrorResponse(
                403,
                "Acesso negado",
                ["message" => "Apenas administradores possuem acesso"]
            );
        }

        return $handler->handle($request);
    }
```


```
{
  "iss": "http://localhost",
  "aud": "http://localhost",
  "sub": "acesso_sistema",
  "iat": 1780862436,
  "nbf": 1780862436,
  "exp": 1783454436,
  "jti": "08765b1cd4197db463c6d73079cd744f",
  "funcionario": {
    "name": "Hélio",
    "email": "helioesperidiao@gmail.com",
    "role": "Administrador",
    "idFuncionario": 3
  }
}
```

Adicione o token em todas as rotas!