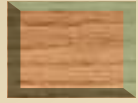


Introdução



Estrutura do JWT



Fluxo



Autenticação



Middleware



Requisição



JSON Web Token TS

Prof. Enzo
Seraphim

Profa. Bárbara
Pimenta Caetano



Introdução - O que é JWT?

- JWT: Json Web Token
- Formato compacto e seguro para transmitir informações como um token de autenticação
- Permite autenticação stateless (sem manter estado no servidor)
- Muito usado em APIs RESTful, microservices, SPAs, mobile

<https://jwt.io/>





Introdução - O que é JWT?

- O servidor te autentica uma vez (login) e te entrega um token assinado.
- Em cada nova requisição, você anexa esse token
- O servidor confere a assinatura para saber se é válido, sem precisar consultar o banco toda hora.
- Isso economiza tempo, processamento e deixa a aplicação mais escalável.





Introdução - O que é JWT?

- **Importante: o JWT não esconde dados!**
- Ele só garante: “ninguém alterou o que está aqui dentro”.
- Ele não protege dados confidenciais (porque qualquer um pode abrir e ver o payload).
- Ele protege a integridade, não o sigilo





Introdução - resumo JWT

→ Para que serve?

- Provar quem sou, de forma segura e rápida.

→ Ele guarda minhas senhas?

- Não! Só informações mínimas para validar.

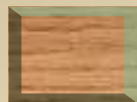
→ Ele esconde os dados?

- Não. Ele assina, mas não criptografa.

→ Por que usar?

- Porque evita revalidar no banco a cada ação.





Introdução



Estrutura do JWT



Fluxo



Autenticação



Middleware



Requisição



JSON Web Token TS

Prof. Enzo
Seraphim

Profa. Bárbara
Pimenta Caetano



Estrutura JWT - 3 componentes

→ Header

- Diz como o token foi gerado (qual algoritmo, qual tipo)

→ Payload

- Contém os dados (claims), por exemplo: quem é você, qual seu ID

→ Signature

- Um código gerado com segredo, que garante que ninguém mexeu nos dados





Estrutura

Estrutura JWT - Payload claims (padrões)

- iss Issuer: quem emitiu o token
- sub Subject: o dono do token (geralmente user ID)
- aud Audience: para quem o token é destinado
- exp Expiration: quando expira (timestamp UNIX)
- nbf Not before: antes disso não é válido
- iat Issued at: quando foi emitido (timestamp)
- jti JWT ID: identificador único do token

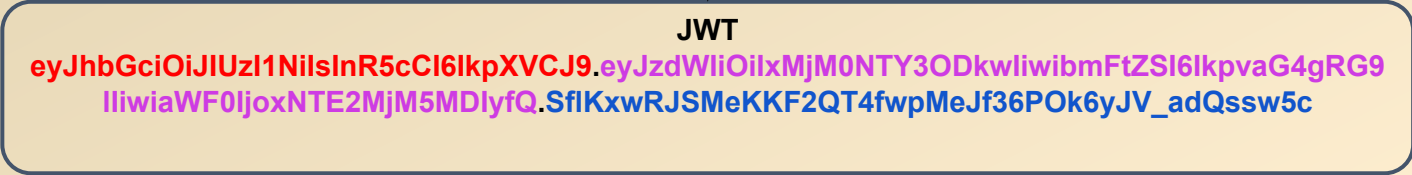
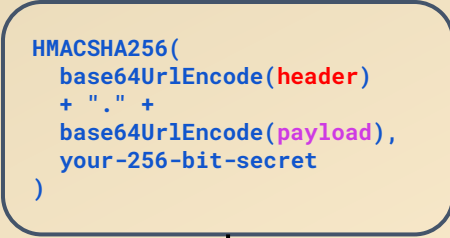
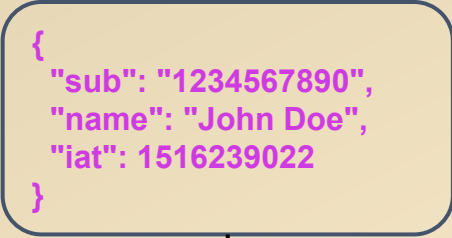
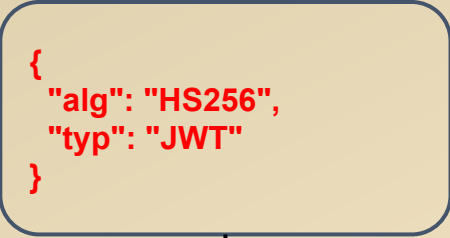
não é obrigado a usar todos, mas **exp**, **iat** e **sub** são muito comuns.



Estrutura JWT

Estrutura

XXXXXXXXXXXXXXXXXXXX . YYYYYYYYYYYYYYYYYY . ZZZZZZZZZZZZZZZZZZZZ





Introdução



Estrutura do JWT



Fluxo



Autenticação



Middleware



Requisição



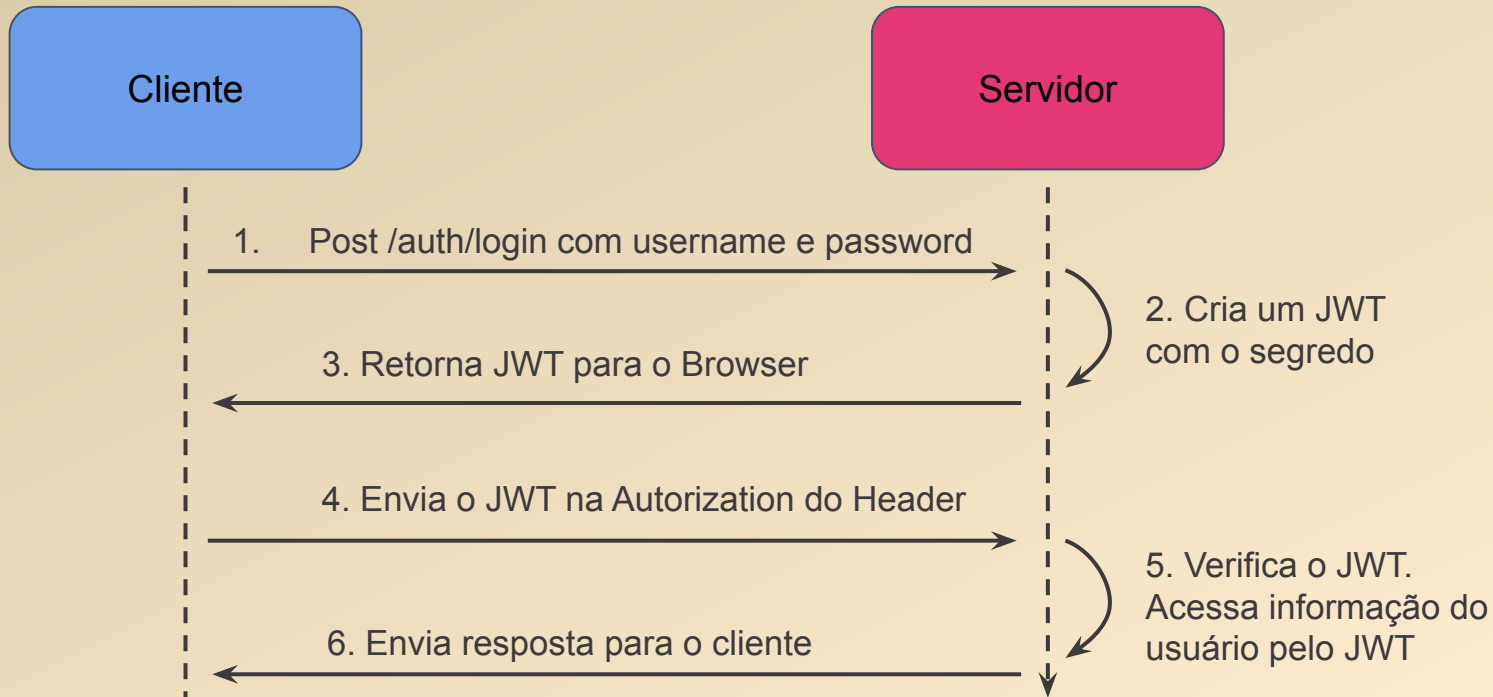
JSON Web Token TS

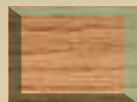
Prof. Enzo
Seraphim

Profa. Bárbara
Pimenta Caetano

Fluxo JWT

Fluxo

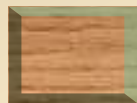




Introdução



Estrutura do JWT



Fluxo



Autenticação



Middleware



Requisição



JSON Web Token TS

Prof. Enzo
Seraphim

Profa. Bárbara
Pimenta Caetano



Autenticação

Ambiente - new

```
//Download Aula 12 - pblc12-12b.zip  
//Inicia o servidor de banco  
docker compose up -d  
//Instalar dependências e dependências de dev  
npm install include dev  
//gera Prisma Client  
npx prisma generate
```





Autenticação

Ajustar Prisma Schema (Ser)

```
model Ser {
  nome String @id
  sexo String
  morto Boolean
  tipo TipoSer
  fraqueza String? //Divindade
  designacao String? //Divindade
  restituirVita Boolean? //Divindade
  poderes PoderDivindade[] //Divindade
  nascimento DateTime? //Mortal
  raca String? //Mortal
  profissao String? //Mortal
  artefatos Artefato[]
  responsaveis SerSer[] @relation("serResponsavel")
  filhos SerSer[] @relation("serFilho")
  nomeMortal String? @unique //Divindade
  hospedeiro Ser? @relation("serMortal", fields: [nomeMortal],
references: [nome]) //Divindade
  divino Ser? @relation("serMortal") //Mortal
  senha String //Adicionar Senha <-----
}
```





Autenticação

Autenticação

```
//gera arquivo de migração e executa SQL no banco  
npx prisma migrate dev --name init
```

```
//Instalar dependência para criptografia de senha  
npm install argon2 jsonwebtoken  
//Instalar dependência de dev  
npm install --save-dev @types/argon2 @types/jsonwebtoken
```

Argon2 é um algoritmo moderno de hashing de senha, considerado um dos mais seguros atualmente.

Ele foi vencedor da Password Hashing Competition (PHC) em 2015, ou seja, foi escolhido como o melhor algoritmo para proteger senhas no lugar de outros mais antigos como bcrypt, PBKDF2 e scrypt.





Atualizar o seed.ts

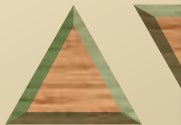
Autenticação

```
import { PrismaClient } from '../generated/prisma'
import argon2 from 'argon2'; //Adicionar import <-----
```

```
async function main() {
  console.log('Iniciando o seed...')
  const senhaHash = await argon2.hash('senha123'); //Gerar hash
```

```
//Adicionar atributo senha nos inserts de Ser
const thor = await prisma.ser.create({data: {nome: 'Thor',
sexo: 'Masculino', morto: false, tipo: 'DIVINDADE',
restituirVita: false, fraqueza: 'tornar não digno', designacao:
'deus do trovão', nomeMortal: blacke.nome, senha: senhaHash}}})
```

```
//Executa as inserções no banco
npx prisma db seed
```





Autenticação

Criar src/repositories/auth.repository.ts

```
import { PrismaClient } from '../../../generated/prisma';

const prisma = new PrismaClient();

export const findSerByNome = async (nome: string) => {
  return prisma.ser.findUnique({ where: { nome } });
};
```





Criar src/controllers/auth.controller.ts

```
import { Request, Response } from 'express';
import jwt from 'jsonwebtoken';
import argon2 from 'argon2';
import * as authRepository from '../repositories/auth.repository';
export const loginController = async (req: Request, res: Response): Promise<void> => {
  const { nome, senha } = req.body;
  if (!nome || !senha) {
    res.status(400).json({ message: 'Nome e senha são obrigatórios.' });
    return;
  }
  const ser = await authRepository.findSerByNome(nome);
  if (!ser) {
    res.status(401).json({ message: 'Usuário não encontrado.' });
    return;
  }
}
```



Criar src/controllers/auth.controller.ts

```
const senhaValida = await argon2.verify(ser.senha, senha);
if (!senhaValida) {
  res.status(401).json({ message: 'Senha inválida.' });
  return;
}
const token = jwt.sign(
  { nome: ser.nome, tipo: ser.tipo },
  process.env.JWT_SECRET || 'secret',
  { expiresIn: '1h' }
);
res.json({ message: 'Login realizado com sucesso!', token });
};
```

```
//Adicionar secret JWT no .env
JWT_SECRET=pblc12
```



Autenticação

Criar src/routers/auth.route.ts

```
import { Router } from 'express';  
import { loginController } from '../controllers/auth.controller';  
  
const router = Router();  
  
router.post('/login', loginController);  
  
export default router;
```



Ajustar src/server.ts

```
import express from 'express';
import swaggerUi from 'swagger-ui-express';
import swaggerJsdoc from 'swagger-jsdoc';
import poderRoutes from './routes/poder.route';
import helloRoutes from './routes/hello.route';
import authRoutes from './routes/auth.route';

const app = express();

app.use(express.json());
const swaggerOptions = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'Divindades API',
      version: '1.0.0',
      description: 'API para gerenciar poderes, seres e artefatos mitológicos',
    },
  },
  apis: ['./src/schemas/*.ts'],
};
```

```
const swaggerSpec = swaggerJsdoc(swaggerOptions);

app.use('/docs', swaggerUi.serve,
  swaggerUi.setup(swaggerSpec));

app.use('/hello', helloRoutes)
app.use('/poderes', poderRoutes);
app.use('/auth', authRoutes);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

```
/**
 * @openapi
 * tags:
 *   - name: Autenticação
 *     description: Endpoints relacionados ao login e
emissão de token
 * /auth/login:
 *   post:
 *     tags:
 *       - Autenticação
 *     summary: Realiza login e gera um token JWT
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             $ref: '#/components/schemas/LoginInput'
 *     responses:
 *       200:
 *         description: Login bem-sucedido, retorna o token
JWT
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/LoginResponse'
 *       400:
 *         description: Dados inválidos (nome ou senha
ausentes)
 *       401:
 *         description: Usuário não encontrado ou senha
inválida
```

```
* components:
*   schemas:
*     LoginInput:
*       type: object
*       required:
*         - nome
*         - senha
*       properties:
*         nome:
*           type: string
*           example: Don Blake
*         senha:
*           type: string
*           example: senha123
*     LoginResponse:
*       type: object
*       properties:
*         message:
*           type: string
*           example: Login realizado com sucesso!
*         token:
*           type: string
*           example: eyJhbGciOiJzI1NiIsInRCI6IkpX9...
```



Introdução



Estrutura do JWT



Fluxo



Autenticação



Middleware



Requisição



JSON Web Token TS

Prof. Enzo
Seraphim

Profa. Bárbara
Pimenta Caetano



Middleware

Criar src/middleware/auth.middleware.ts

```
//Criar pasta middleware
```

```
mkdir src/middleware
```

```
import jwt from 'jsonwebtoken';
import { Request, Response, NextFunction } from 'express';

export const authMiddleware = (req: Request, res: Response,
  next: NextFunction): void => {
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    res.status(401).json({ message: 'Token ausente' });
    return;
  }
  const token = authHeader.split(' ')[1];
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET || 'secret');
    (req as any).user = decoded;
    next();
  } catch {
    res.status(401).json({ message: 'Token inválido' });
    return;
  }
};
```



Ajustar src/server.ts

```
import express from 'express';
import swaggerUi from 'swagger-ui-express';
import swaggerJsdoc from 'swagger-jsdoc';
import poderRoutes from './routes/poder.route';
import helloRoutes from './routes/hello.route';
import authRoutes from './routes/auth.route';
import { authMiddleware } from
'./middlewares/auth.middleware';

const app = express();
app.use(express.json());
const swaggerOptions = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'Divindades API',
      version: '1.0.0',
      description: 'API para gerenciar poderes,
seres e artefatos mitológicos',
    },
  },
  apis: ['./src/schemas/*.ts'],
};
```

```
const swaggerSpec = swaggerJsdoc(swaggerOptions);

app.use('/docs', swaggerUi.serve,
swaggerUi.setup(swaggerSpec));
app.use('/hello', helloRoutes)
app.use('/poderes', authMiddleware, poderRoutes);
app.use('/auth', authRoutes);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```



Introdução



Estrutura do JWT



Fluxo



Autenticação



Middleware



Requisição



JSON Web Token TS

Prof. Enzo
Seraphim

Profa. Bárbara
Pimenta Caetano

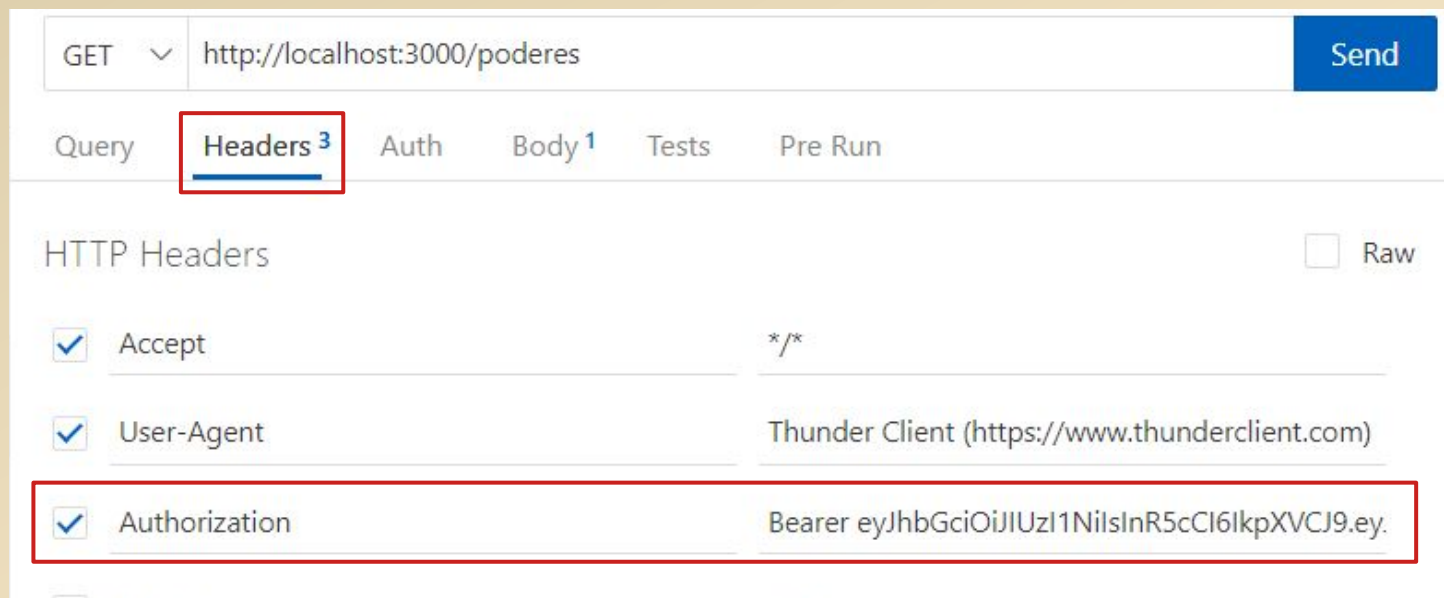


Requisição

→ Header da requisição

```
Authorization: Bearer <token jwt>
```

→ Ex: ThunderClient



The image shows the ThunderClient interface. At the top, there is a dropdown menu set to 'GET' and a text input field containing 'http://localhost:3000/poderes'. To the right of the input field is a blue 'Send' button. Below the input field is a tabbed interface with tabs for 'Query', 'Headers', 'Auth', 'Body', 'Tests', and 'Pre Run'. The 'Headers' tab is selected and highlighted with a red box. Below the tabs, the 'HTTP Headers' section is visible. It has a 'Raw' checkbox on the right. There are three header entries, each with a checked checkbox, a name, and a value. The first entry is 'Accept' with value '*/*'. The second entry is 'User-Agent' with value 'Thunder Client (https://www.thunderclient.com)'. The third entry is 'Authorization' with value 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ' and is highlighted with a red box.

Header Name	Header Value
Accept	*/*
User-Agent	Thunder Client (https://www.thunderclient.com)
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ

Requisição





Requisição Swagger

Ajustar swaggerOptions - src/server.ts

```
const swaggerOptions = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'Divindades API',
      version: '1.0.0',
      description: 'API para gerenciar poderes, seres e artefatos mitológicos',
    },
    components: {
      securitySchemes: {
        bearerAuth: {
          type: 'http',
          scheme: 'bearer',
          bearerFormat: 'JWT',
        },
      },
    },
    security: [
      {
        bearerAuth: [],
      },
    ],
  },
  apis: ['./src/schemas/*.ts'],
};
```



Ajustar src/schemas/auth.schema.ts

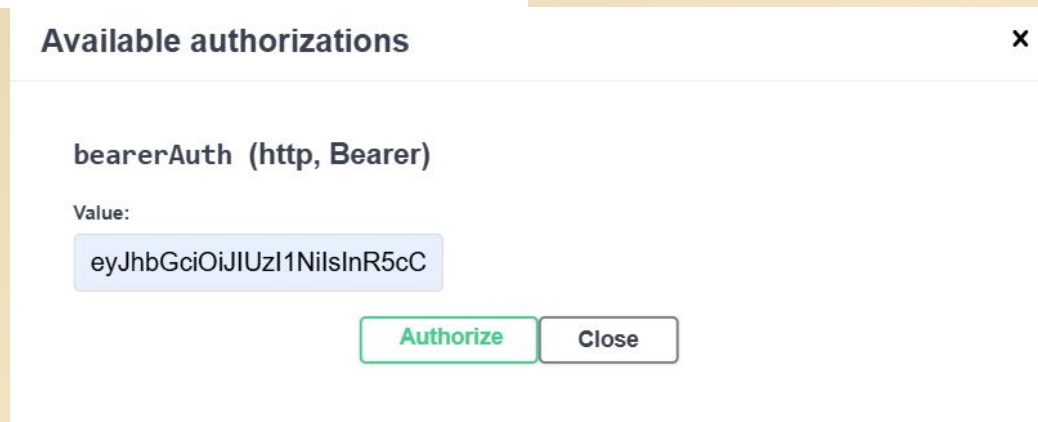
```
//Adicioar security vazio

* /auth/login:
*   post:
*     tags:
*       - Autenticação
*     security: []
*     summary: Realiza login e gera um token JWT
*     requestBody:
*       required: true
*       content:
*         application/json:
*           schema:
*             $ref: '#/components/schemas/LoginInput'
```



Requisição Swagger

Ajustar src/schemas/auth.schema.ts



**Prof. Enzo
Seraphim**

**Profa. Bárbara
Pimenta Caetano**

Os logotipos, marcas comerciais e nomes de produtos citados nesta publicação tem apenas o propósito de identificação e podem ser marcas registradas de suas respectivas companhias.



JSON Web Token TS