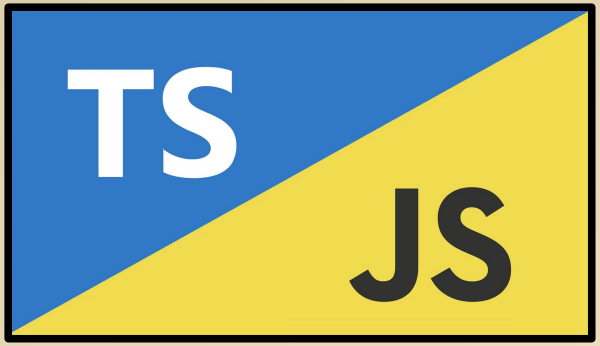
- Introdução
- REST
- Arquitetura
- Repositories
- Controllers
- Routes
- Swagger: {}



Express TS

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

Introdução

Ambiente

```
//gera package.json
npm init -y
//dependências ambiente produção
npm install @prisma/client express
//dependências ambiente local
npm install prisma --save-dev typescript tsx @types/node @types/express
```

```
{//package.json
 //modificar dev no scripts e adicionar prisma
 "scripts": {
  "dev": "tsx watch src/server.ts"
 },
 "prisma": {
  "seed": "npx tsx src/seed.ts"
 },
```

```
//para executar
npm run dev
```

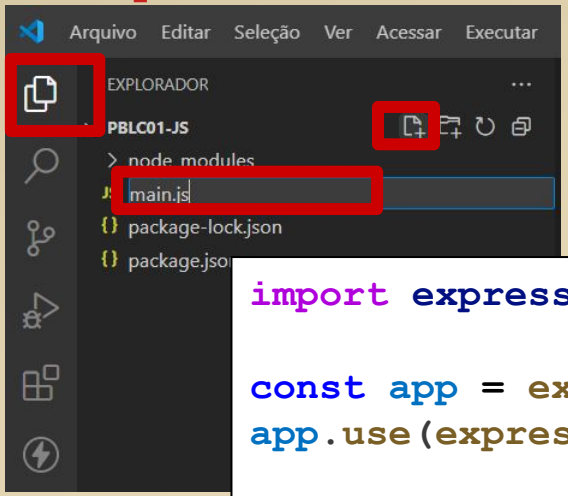
Ambiente

```
//gera tsconfig.json
npx tsc --init
//criar pasta
mkdir src
mkdir src\routes
mkdir src\controllers
mkdir src\repositories
```

```
{ //tsconfig.json
 "compilerOptions": {
 "target": "es2020",
 "rootDir": "./src", //descomentar dir código
 "outDir": "./generated", //descomentar dir geração js
```


Express: Arquivo ./src/server.ts

Introdução




```
import express from 'express';

const app = express();
app.use(express.json());

app.get('/hello', (req, res) => {
  res.json({ message: 'Hello World Express' });
});

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

Ambiente Docker e Prisma

Introdução

```
//inicializar docker com postgres
//download docker-compose.yaml
docker compose up -d
```

```
//prefixar o prisma com o executor de pacotes:
npx prisma
//./prisma/schema.prisma e .env
npx prisma init --datasource-provider postgresql
```

```
//atualizar arquivo .env
DATABASE_URL="postgresql://thor:thor@localhost
:5432/thor?schema=public"
```

TS JS

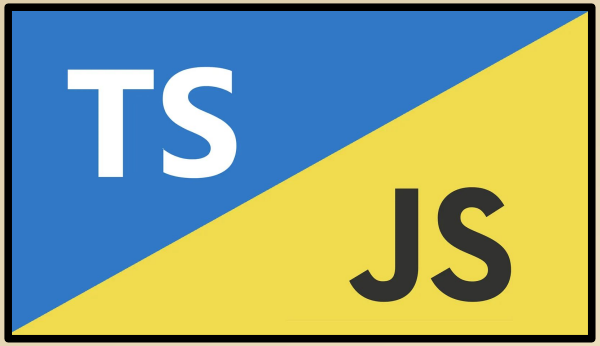
Ambiente Docker e Prisma

Introdução

```
//download seed.ts
//mover para ./src
//download schema.prisma
//mover substituindo ./prisma
```

```
//gera Prisma Client
npx prisma generate
//migrar banco de dados para ambiente dev
//gera arquivo de migração e executa SQL no banco
npx prisma migrate dev --name init
//executa inserção do seed.ts
npx prisma db seed
//Visualizar dados
npx prisma studio
```

- Introdução
- REST
- Arquitetura
- Repositories
- Controllers
- Routes
- Swagger: {}



Express TS

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

Express

Definições

- API → conjunto de definições e protocolos utilizados no desenvolvimento e integração de aplicações de software através de de requisição (por parte do consumidor) e sua resposta (por parte do provedor).
- REST → estilo de arquitetura para desenvolver sistemas de comunicação entre cliente e servidor. Define princípios e restrições de recursos para garantir uma comunicação eficiente e flexível.
- EndPoints → representam recursos em uma API através de uma URI única que são manipulados usando solicitações HTTP através dos: GET, POST, PUT, PATCH, DELETE.

TS JS

Métodos HTTP

Express

- ➔ GET: Recuperar um recurso (ex: dados de um cliente).
- ➔ POST: Criar um novo recurso (ex: criar um novo usuário).
- ➔ PUT: Substituir um recurso existente (ex: atualizar o endereço de entrega).
- ➔ DELETE: Remover um recurso (ex: remover um produto em uma compra).
- ➔ PATCH: Atualizar parcialmente um recurso (ex: alterar apenas o endereço de um usuário).

TS JS

Nomeação EndPoints

- ➔ Os endpoints devem focar no recurso e não na ação.
- ➔ Use letras minúsculas para endpoints
- ➔ Substantivos no plural para coleções de recursos. Ex. em vez da URI /getUsuarios, use a URI /usuarios.
- ➔ Substantivos no singular para um recurso individual. Ex. URI usuarios/enzo
- ➔ Evite verbos, já que são explicitados nos métodos HTTP
- ➔ Kebab case → convenção de nomenclatura onde palavras são separadas por hífens (-) e todas as letras são convertidas para minúsculas.

Express

Exemplo EndPoints

- ➔ GET /seres → Retorna uma lista de seres
- ➔ GET /seres/{nome} → Retorna o usuário nomeado
- ➔ POST /seres → Cria um usuário
- ➔ PUT /seres/{nome} → Atualiza todos os atributos do ser nomeado
- ➔ PATCH /seres/{nome} → Atualiza alguns atributos do ser nomeado
- ➔ DELETE /seres/{nome} → Remove o ser nomeado

https://www.restapitutorial.com/introduction/httpmethods

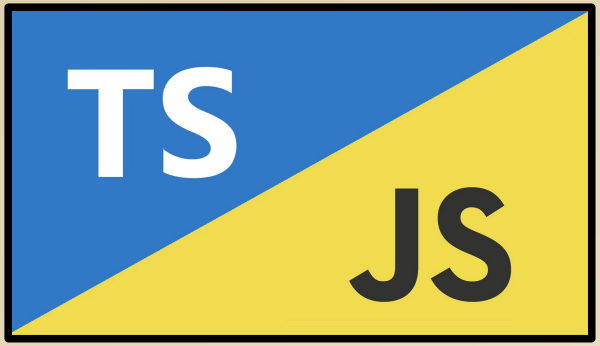
TS JS

Relacionamentos EndPoints

Express

- Alguns Relacionamentos (geralmente agregação) podem ser mapeados em hierarquias do mesmo endpoint
- GET /seres/{nome}/poderes → lista os poderes para ser nomeado
- GET /seres/{nome}/poderes/{id} → lista poder de id 6 para ser nomeado
- POST /seres/{nome}/poderes → cria um poder para o ser nomeado
- PUT /seres/{nome}/poderes/{id} → atualiza todos campos do poder identificado do ser nomeado
- PATCH /seres/{nome}/poderes/{id} → atualiza parte do do poder identificado do ser nomeado
- DELETE /seres/{nome}/poderes/{id} → remove poder identificado do ser nomeado

- Introdução
- REST
- Arquitetura
- Repositories
- Controllers
- Routes
- Swagger: {}



Express TS

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

Arquitetura

Routes+Controllers+Repositories

- Routes → define as rotas dos endpoints. Um classe de domínio tem várias rotas. Cada rota está associada a uma funcionalidade controllers.
- Controllers → define as funcionalidades do sistema, geralmente métodos CRUD. Um controlador pode estar associado a um ou mais funções do repositório.
- Repositories → define um método para cada ação de banco de dados

Exemplo 1 Hello: Rota GET /

```
//Criar pasta: src/routes/
//Criar um arquivo hello.route.ts

import { Router } from 'express';
import { helloController }
   from '../controllers/hello.controller';

const router = Router();
router.get('/', helloController);

export default router;
```

Exemplo 1 Hello: Controller

```
//Criar pasta: src/controllers/
//Criar um arquivo hello.controller.ts


import { Request, Response } from 'express';

export const helloController=(req: Request, res: Response) =>
{
  res.json({ message: 'Hello from controller!' });
};
```

Ajustar no server.ts

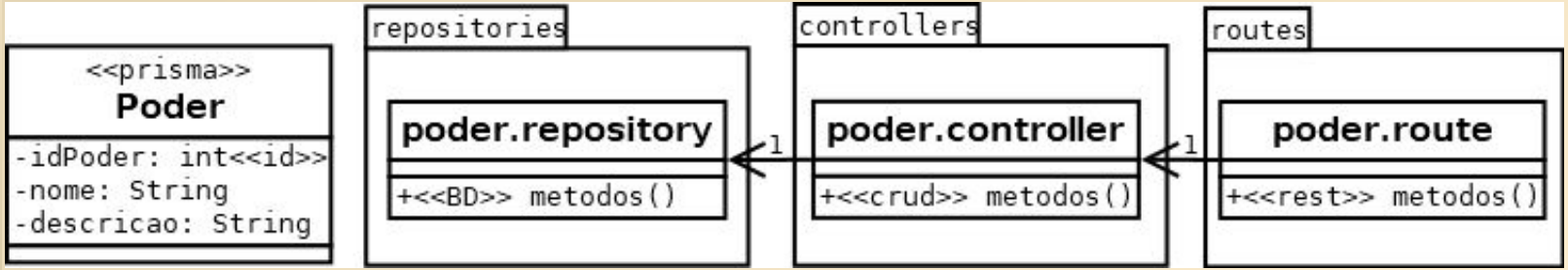
```ts
import express from 'express';
//adicionar o import das routes
import helloRoutes from './routes/hello.route';

const app = express();
app.use(express.json());

//adicionar o use das routes
app.use('/', helloRoutes);

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

npm run dev

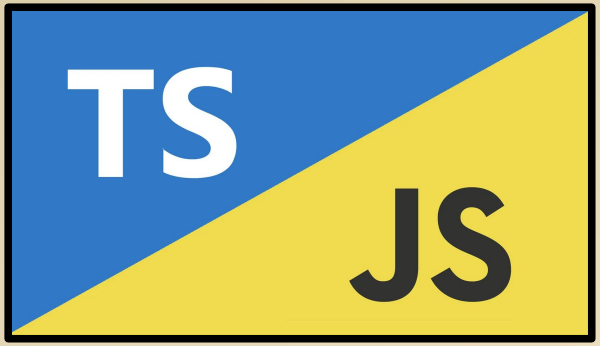
Routes+Controllers+Repositories

Arquitetura

- ➔ poder.repository → um método para cada ação de BD para a classe Poder
- ➔ poder.controller → Funcionalidades CRUD para Poder
- ➔ poder.route → Rotas do EndPoint Poder



- Introdução
- REST
- Arquitetura
- Repositories
- Controllers
- Routes
- Swagger: {}



Express TS

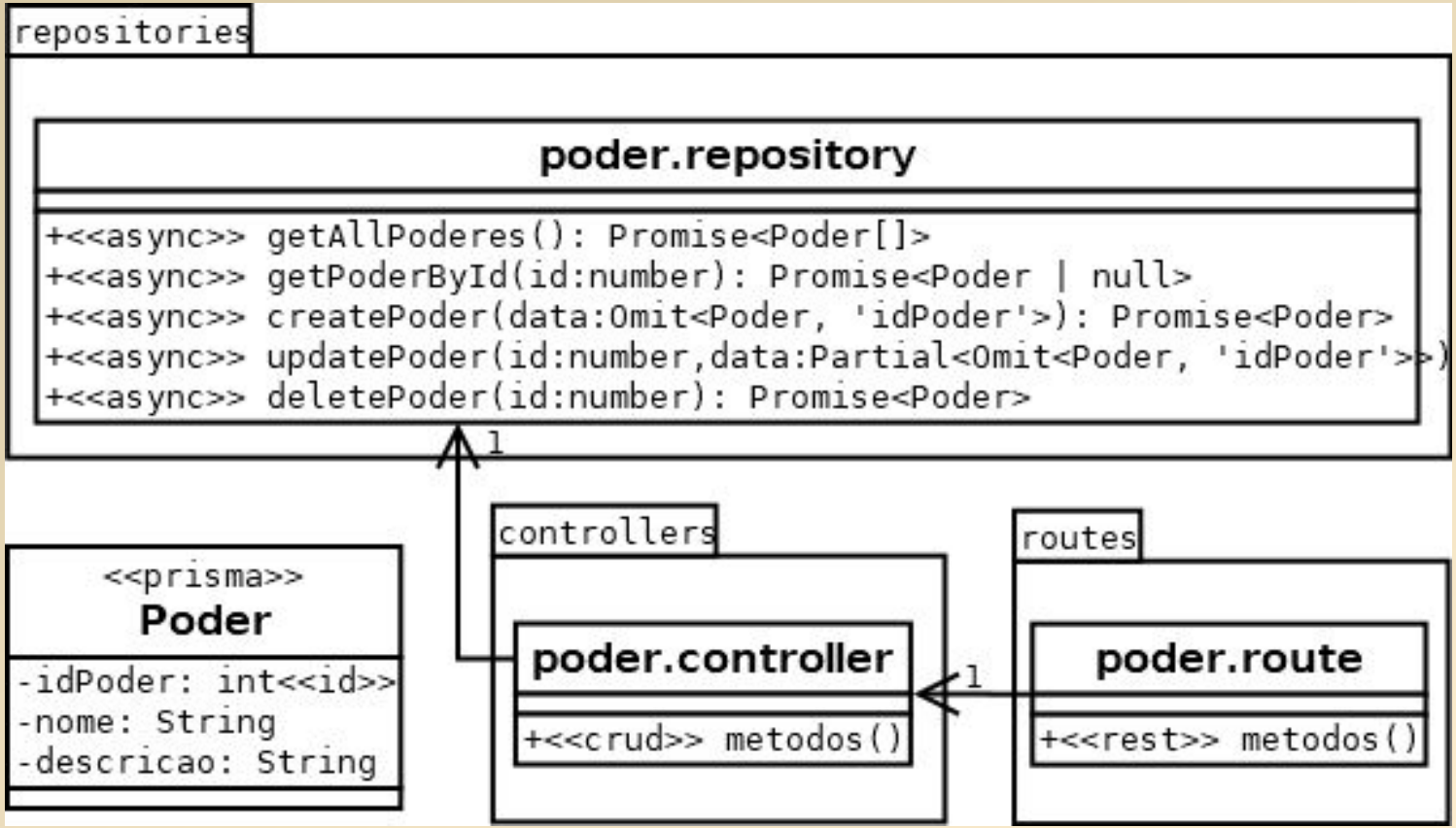
Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

Repository → Poder

Repositories



TS JS

Repository - poder.repository.ts

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

const prisma = new PrismaClient();

export const getAllPoderes = async (): Promise<Poder[]> => {
  return prisma.poder.findMany();
};

export const getPoderById=async(id: number):Promise<Poder | null> => {
  return prisma.poder.findUnique({ where: { idPoder: id } });
};

export const createPoder = async (data: Omit<Poder, 'idPoder'>):
Promise<Poder> => {
  return prisma.poder.create({ data });
};
```

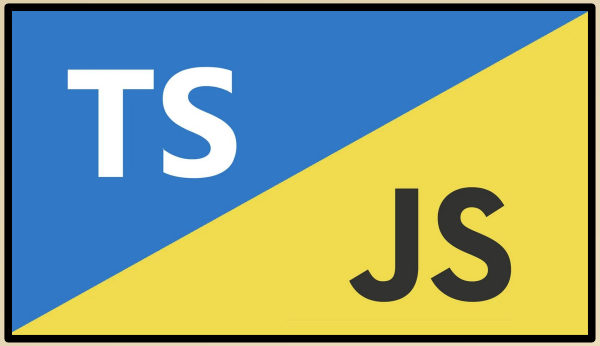
TS JS

Definições usadas Repositories

Repositories

- async → operador para definir um método assíncrono (não executados ao mesmo tempo)
- Promise<Type1> → tipo que representa a eventual conclusão ou falha de uma operação assíncrona (não executados ao mesmo tempo)
- Omit<Type,Keys> → constrói um tipo selecionando todas as propriedades de Type removendo-as Keys
- Partial<Type> → constrói um tipo com todas as propriedades de Typeset como opcionais.

- Introdução
- REST
- Arquitetura
- Repositories
- Controllers
- Routes
- Swagger: {}



Express TS

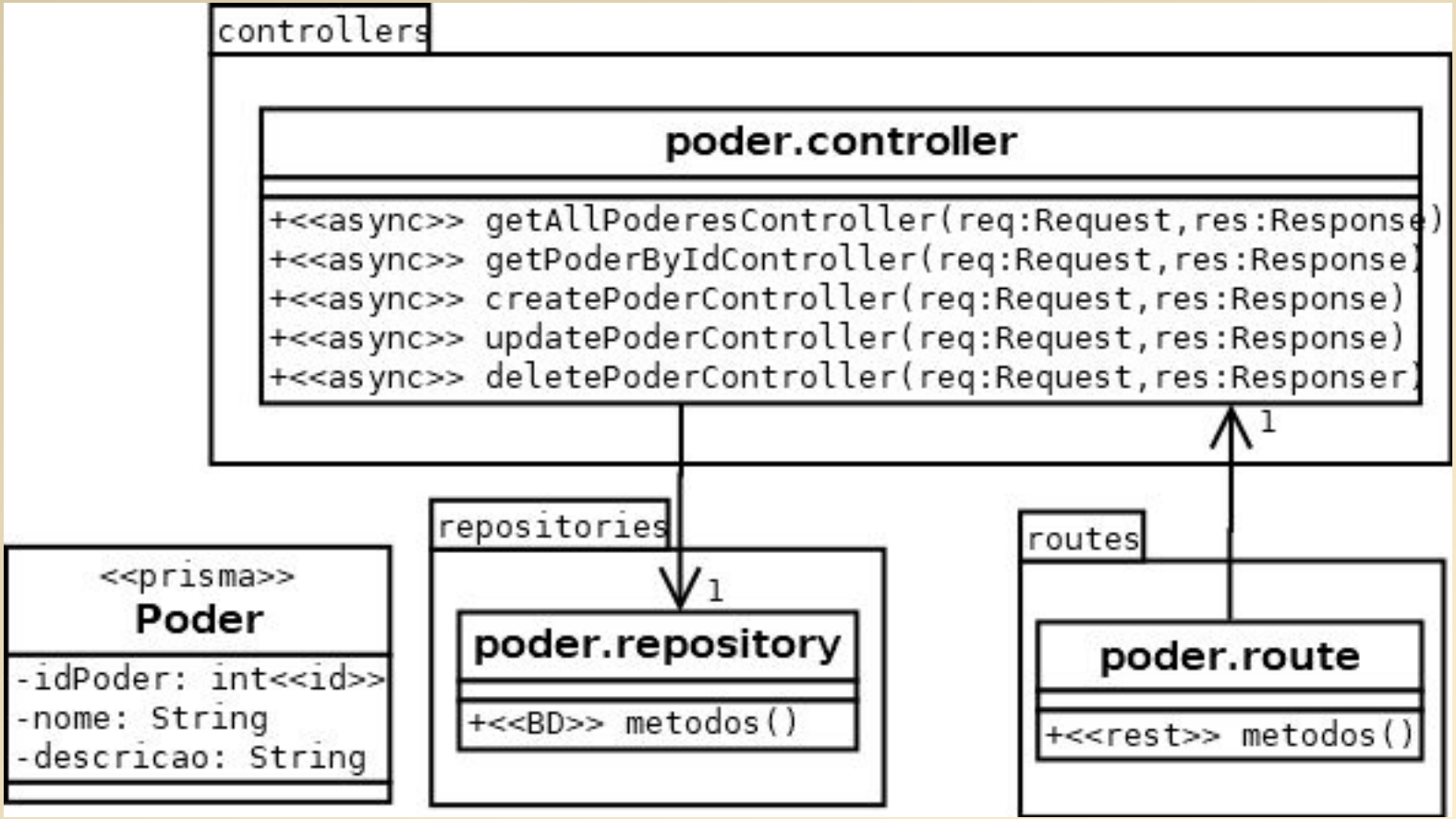
Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

Controller → Poder

Controllers



Controller - poder.controller.ts

```ts
import { Request, Response } from 'express';
import * as poderRepository from '../repositories/poder.repository';
export const getAllPoderesController = async (req: Request, res: Response) => {
  const poderes = await poderRepository.getAllPoderes();
  res.json(poderes);
};
export const getPoderByIdController = async (req: Request, res: Response) => {
  const id = parseInt(req.params.id);
  const poder = await poderRepository.getPoderById(id);
  if (poder) {
    res.json(poder);
  } else {
    res.status(404).json({ message: 'Poder not found' });
  }
};
export const createPoderController = async (req: Request, res: Response) => {
  const { nome, descricao } = req.body;
  const newPoder = await poderRepository.createPoder({ nome, descricao });
  res.status(201).json(newPoder);
};
```

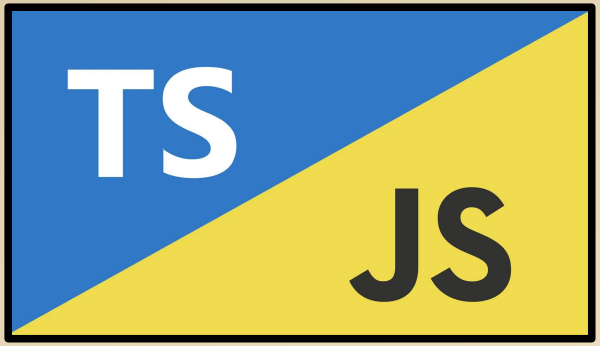
TS JS

Definições usadas Controllers

Controllers

➔ await → operador usado em chamada de método que para aguardar uma Promessa e obter seu valor de cumprimento. Somente usado dentro de uma função assíncrona.

- Introdução
- REST
- Arquitetura
- Repositories
- Controllers
- Routes
- Swagger: {}



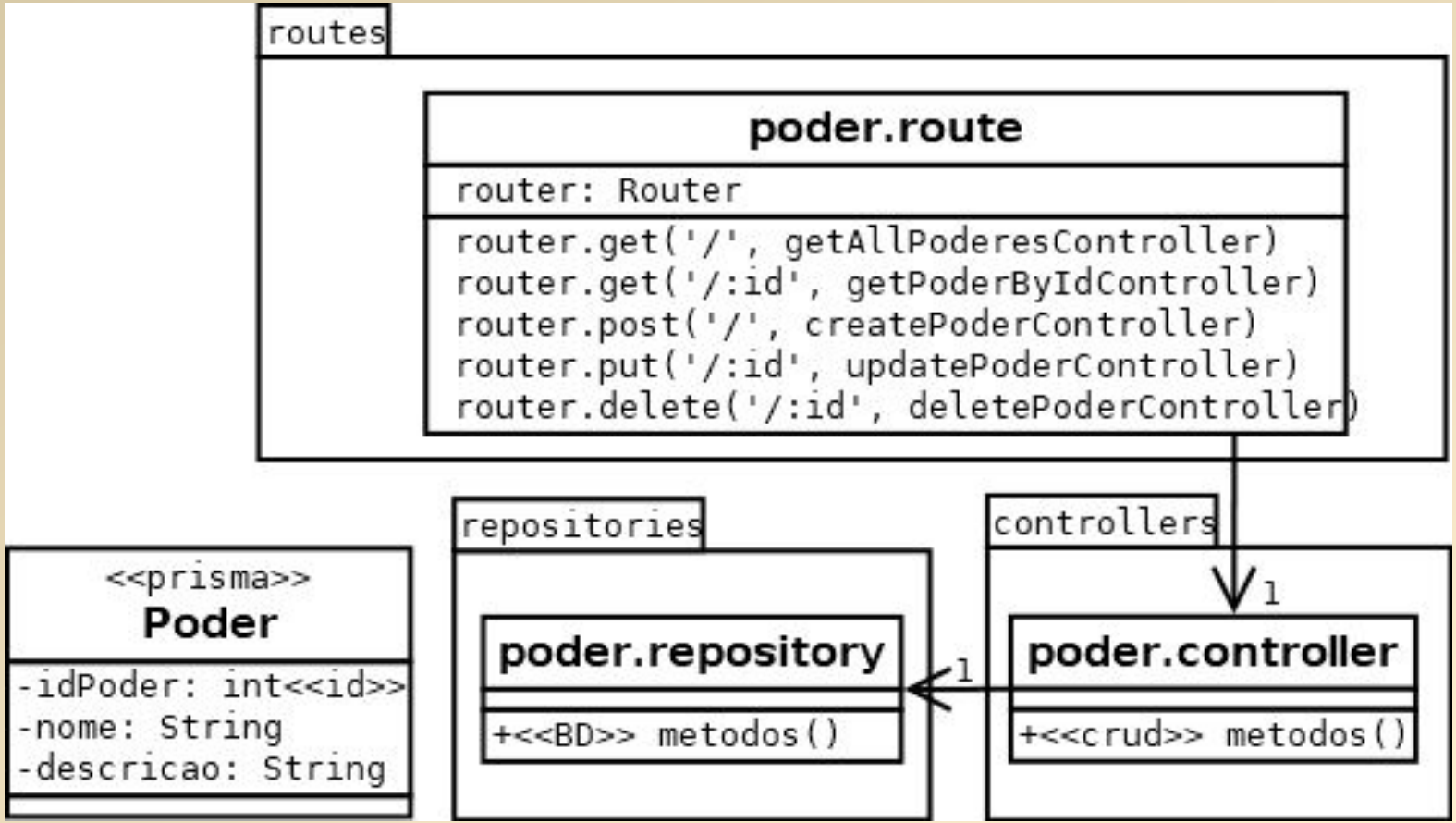
Express TS

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

Router → Poder

Routes



Router - poder.router.ts

```
import { Router } from 'express';
import {
    getAllPoderesController, getPoderByIdController,
    createPoderController
} from '../controllers/poder.controller';
const router = Router();
router.get('/', getAllPoderesController);
router.get('/:id', getPoderByIdController);
router.post('/', createPoderController);

export default router;
```

npm run dev

Ajustar o server.ts

```
import express from 'express';
import poderRoutes from './routes/poder.route';
import helloRoutes from './routes/hello.route';
const app = express();
app.use(express.json());
'app.use('/hello', helloRoutes)
app.use('/poderes', poderRoutes);
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {console.log(`Server running on port ${PORT}`);});
```

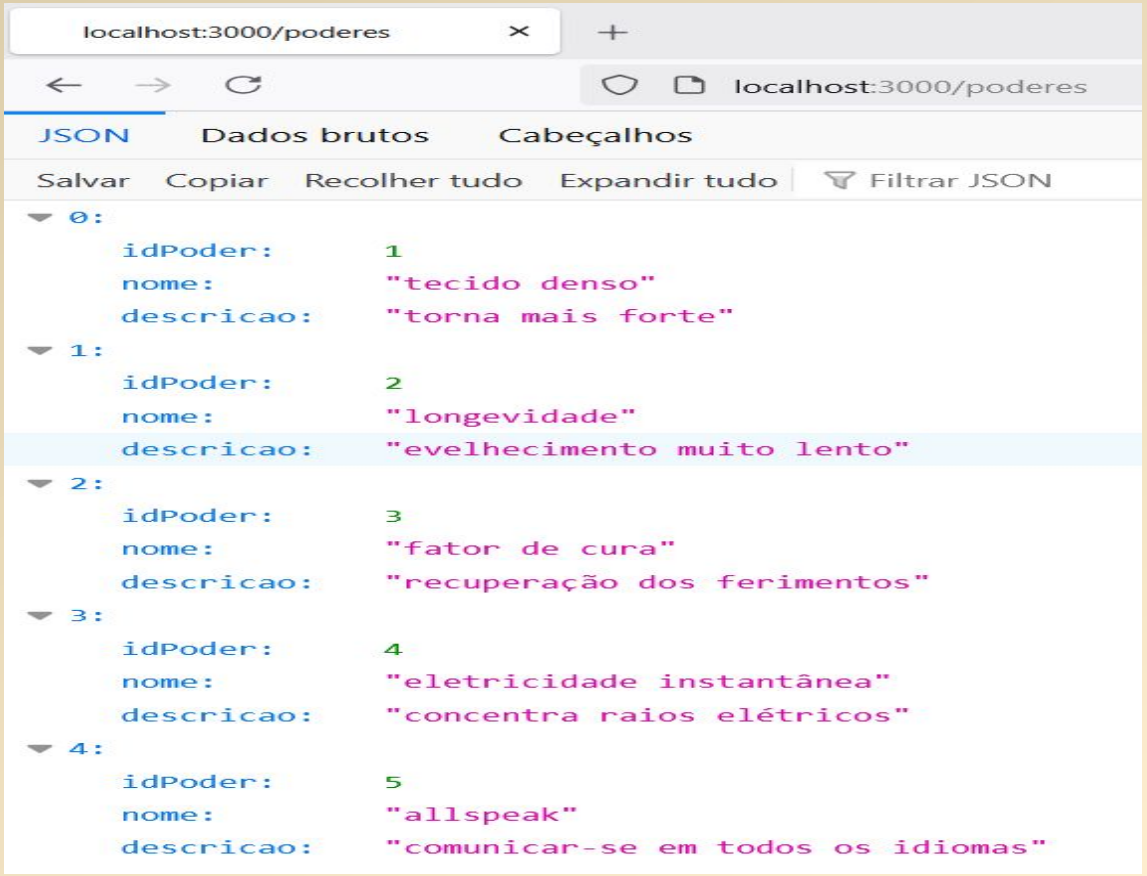
npm run dev

TS JS

Firefox

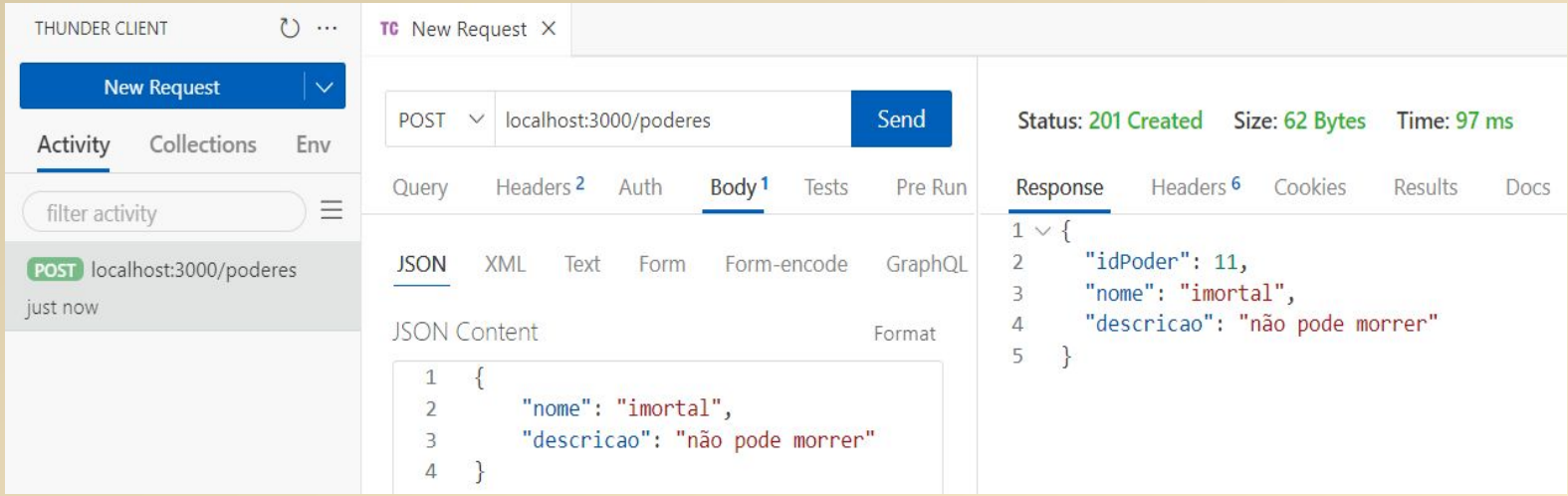
http://localhost:3000/poderes

Router



Thunder Client

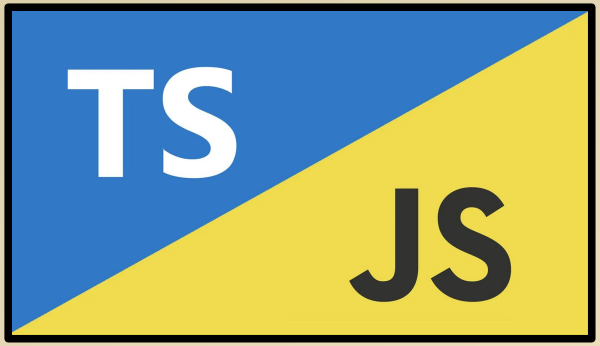
POST | **http://localhost:3000/poderes**



Router

```
{
  "nome": "imortal",
  "descricao": "não pode morrer"
}
```

- Introdução
- REST
- Arquitetura
- Repositories
- Controllers
- Routes
- Swagger: {}



Express TS

Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

TS JS

Swagger https://swagger.io/

Swagger

- Swagger é um conjunto de ferramentas para especificação, documentação e testes de APIs REST.
- Atualmente, o padrão de especificação se chama OpenAPI Specification (OAS). É mantido pela OpenAPI Initiative, ligada à Linux Foundation, garantindo padrões abertos e colaboração comunitária.
- Permite descrever os endpoints, parâmetros, respostas, erros e autenticação de uma API em formato legível por humanos e máquinas (normalmente usando um arquivo YAML ou JSON).
- O swagger possui Apache license 2.0

Swagger

```
//instalar as dependencias do swagger
npm install swagger-ui-express swagger-jsdoc
//dependências ambiente local
npm install --save-dev @types/swagger-ui-express @types/swagger-jsdoc
```

Swagger

- ➔ O swagger-jsdoc coleta os comentários JSDoc (/** @openapi ... */) do código e gera um JSON OpenAPI.
- ➔ Esse JSON é passado para o swagger-ui-express, que renderiza a interface visual no navegador.
- ➔ Ex.: https://petstore.swagger.io/

```
mkdir src\schemas
```

src\schema\poder.schema.ts

```
/**
 * @openapi
 * /poderes:
 *   get:
 *     tags: [Poderes]
 *     summary: Lista todos os poderes
 *     responses:
 *       200:
 *         description: Lista de poderes
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 $ref: '#/components/schemas/Poder'
 *   post:
 *     tags: [Poderes]
 *     summary: Cria um novo poder
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             $ref: '#/components/schemas/PoderInput'
 *     responses:
 *       201:
 *         description: Poder criado
```

```
 * /poderes/{id}:
 *   get:
 *     tags: [Poderes]
 *     summary: Busca um poder por ID
 *     parameters:
 *       - name: id
 *         in: path
 *         required: true
 *         schema:
 *           type: integer
 *     responses:
 *       200:
 *         description: Poder encontrado
 *         content:
 *           application/json:
 *             schema:
 *               $ref:
'#/components/schemas/Poder'
 *       404:
 *         description: Poder não encontrado
```

continua o próximo slide

Swagger - schemas

```
* components:
*   schemas:
*     Poder:
*       type: object
*       properties:
*         idPoder:
*           type: integer
*         nome:
*           type: string
*         descricao:
*           type: string
*     PoderInput:
*       type: object
*       properties:
*         nome:
*           type: string
*         descricao:
*           type: string
*/
```

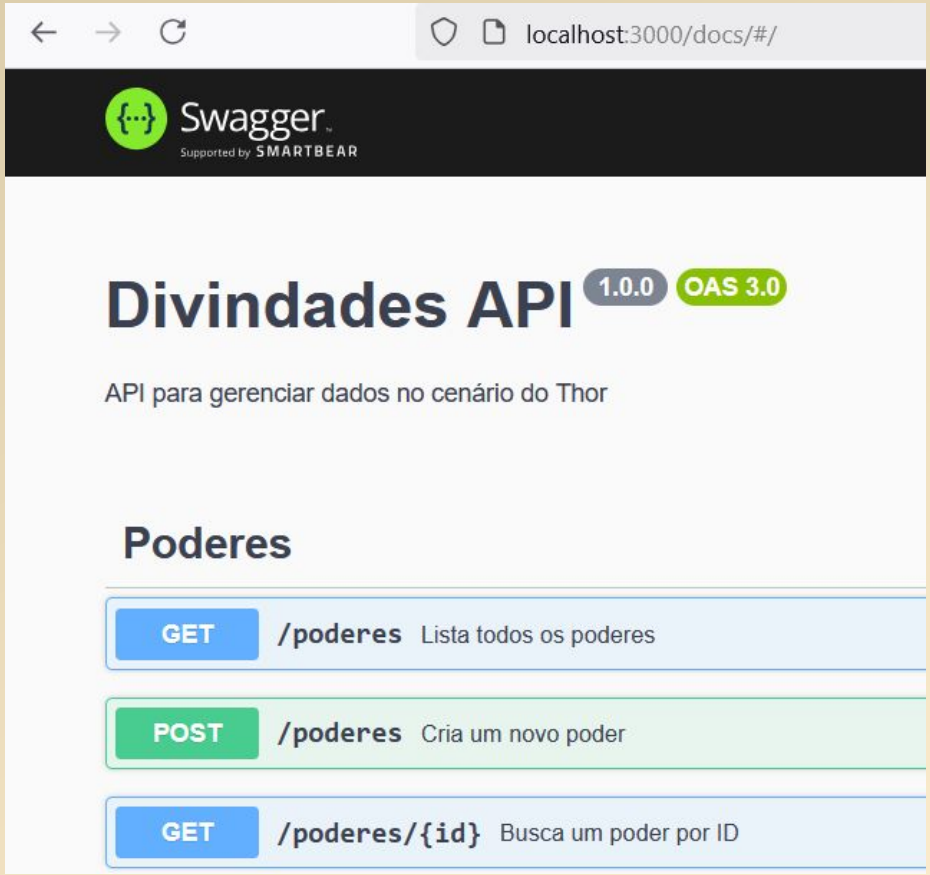
Swagger - ajustar o server.ts

```
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';
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 dados no cenário do Thor',},},
  apis: ['./src/schemas/*.ts'],
};
const swaggerSpec = swaggerJsdoc(swaggerOptions);
app.use('/docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));
app.use('/', helloRoutes)
app.use('/poderes', poderRoutes);
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {console.log(`Server running on port ${PORT}`);});
```

Swagger

http://localhost:3000/docs

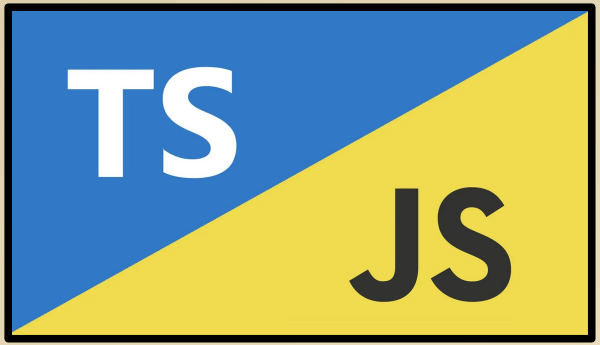
Swagger



Prof. Enzo Seraphim

Profa. Bárbara Pimenta Caetano

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



Express TS