14/08/2022

Oque é o NodeJs

Plataforma open-source permite execução da linguagem Javascript do lado do servidor.

V8+libuv_conjuto de módulos.

Arquitetura Event Loop: Call Stack.

Single Thread.

Non-blocking I/O.

API: Interface de Programação de aplicativos.

Conjunto de especificações de possíveis interações entre aplicações.

Documentação para desenvolvedor.

Rest: Transferência representacional de estado. 6 regras.

1 - Client Server.

2- Stateless.

3 – Cache.

4 – Interface Uniform:

Identificação dos recursos.

Representação dos recursos.

Mensagens auto-descritivas.

HATEOAS { “id”: 1 };

5 – Camadas.

6 Códigos Sob Demanda.

Métodos de requisições http verbs:

GET – leitura.

POST – Criação.

PUT – Atualização.

DELETE – Deleção.

PATCH – Atualização Parcial.

HTTP Codes:

1xx Informativo – a solicitação foi aceita ou o processo continua em andamento.

2xx Confirmação – 200 Requisição bem sucedida / 201 Created.

3xx Redirecionamento – 301 Moved Permanenttly / 302 Moved.

4xx Erro do cliente – 400 bad request / 401 Unauthorized / 403 forbiden / 404 not found / unprocessable entity.

5xx erro no servidor – 500 internal server error / 502 bad gateway.

Parâmetros das requisições

Header Params – Parametros que vão no cabeçalho.

Query Params – no final de url, paginação, chave, valor, separação.

Route Params – parâmetros que vão no meio da rota, fazer busca.

Body Params – quando envia no corpo da requisição.

Boas práticas de API REST

A utilização correta dos métodos HTTP.

A utilização correta dos status no retorno das respostas.

Padrão de nomenclatura.

Reduce(); transforma o valores que passarem em um só valor.

Módulo 02

TypeScript: é o Javascript, tipagem estática

```
Class User {
  Name: string;
  Username: string;
  Document: string;
}


Const user: User = {
  Name: “José”,
  Username: “JrSs”,
  Document: “3333”
}
```

Typescript:

Yarn init -y, yarn add express, yarn add @types/express -D, yarn add typescript, yarn tsc –int,

Converter ts para Js instalar: Yarn add ts-node-dev -D

```
"scripts": {
  "dev": "ts-node-dev --transpile-only --ignore-watch node_mdules --respawn src/server.ts"
},
```

E dentro do tsconfig.json desabilitar “strict”: true.

```
// "strict": true,
options. */
```

Criando categoria linha 15 (201).send(). falta

src > routes > categories.routes.ts > categoriesRoutes.post("/categories") callback

```
import { Router } from "express";

const categoriesRoutes = Router();

const categories = [];

categoriesRoutes.post("/categories", (request, response) => {
  const { name, description } = request.body;

  categories.push({
    name,
    description,
  });

  return response.status(201);
});

export { categoriesRoutes };
```

src > server.ts > …

```
import express from "express";
import { categoriesRoutes } from './routes/categories.routes';

const app = express();

app.use(express.json());

app.use(categoriesRoutes)

app.listen(3333, () => console.log("Server is running!"));
```

Inserido Id com uuid yarn add uuid e yarn add @types/uuid

```
import { v4 as uuidV4 } from 'uuid'  1.5K (gzipped: 754)
```

```
const category = {
  name,
  description,
  id: uuidV4(),
};
```

E NO SERVER.TS

```
app.use(express.json());

app.use("/categories", categoriesRoutes);
```

Inserindo tipagem para categoria

```
src > modules > cars > model > Category.ts > ...
import { v4 as uuidV4 } from "uuid";

// Modelo da classe Category
class Category {

  id?: string; // id opcional
  name: string;
  description: string;
  created_at: Date;

  // constructor um método que é chamado quando a classe é instânciada, new () iniciar
  constructor() {
    if (!this.id) { // se ñ tiver nem um id desse Category
      this.id = uuidV4(); // this id recebe um uuid
    }
  }
}

export { Category }
```

```
  const { name, description } = request.body;

  const category = new Category();

  Object.assign(category, {
    name,
    description,
    created_at: new Date()
  })

  categories.push(category);

  return response.status(201).json({ category });
});
```

Criando repositório de categoria é como classe responsável pela manipulação de db

```
// DTO = Data Transfer Object
export interface ICreateCategoryDTO {
  name: string;
  description: string;
}
```

```
class CategoriesRepository {
  private categories: Category[];

  constructor() {
    this.categories = [];
  }

  create({ description, name }: ICreateCategoryDTO): void {
    const category = new Category();

    Object.assign(category, {
      name,
      description,
      created_at: new Date(),
    });

    this.categories.push(category);
  }
}
```

E nas rotas

```
import { CategoriesRepository } from "../repositories/CategoriesRepository";

const categoriesRoutes = Router();
const categoriesRepository = new CategoriesRepository();

categoriesRoutes.post("/", (request, response) => {
  const { name, description } = request.body;

  categoriesRepository.create({ name, description });

  return response.status(201).send();
});

export { categoriesRoutes };
```

Listando as categorias no category repository

```
list(): Category[] { // função q retorna a lista de Category
  return this.categories;
}
```

Na rotas

```
categoriesRoutes.get("/", (request, response) => {
  const all = categoriesRepository.list();

  return response.json(all);
});
```

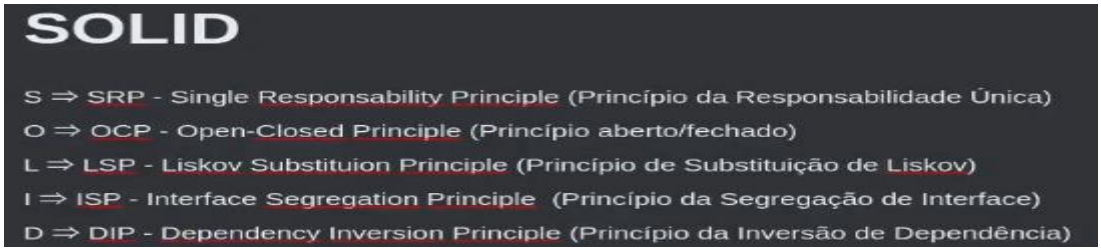
Validado o cadastro de categorias no category repository

```
findByName(name: string): Category { // função de verificação de duplicação do name
  // find percorre o array procurando pelo name e retorna
  const category = this.categories.find(category => category.name === name);
  return category;
}
```

Nas rotas

```
categoriesRoutes.post("/", (request, response) => {
  const { name, description } = request.body;

  const categoryAlreadyExists = categoriesRepository.findByName(name);

  if (categoryAlreadyExists) {
    return response.status(400).json({ error: "Category Already exists!" });
  }

  categoriesRepository.create({ name, description });

  return response.status(201).send();
```

Repositório: é uma classe, uma camada responsável pela manipulação de dados da aplicação, com banco de dados.



SRP Single Responsability Principle Princípio de responsabilidade única

Rota só é responsável para receber a requisição e retorna as informações recebida.

Services: É um intermediário entre as rotas e os repositórios, responsável pela regra de negócio.

As rotas existem apenas unicamente para atender as requisições.

Os repositórios existem apenas unicamente para fazer operações para guardar e prover só dados persistidos.

DIP Dependency inversion príncipe Princípio de inversão de dependência.

O código que implementa uma política de alto nível não deve depender de código que depende do código que implementa código de baixo nível.

O servisse não tem que reconhecer qual o tipo do repositório, servisse o alto nível.

Rotas baixo nível.

LSP Liskov Substituion Principle Princípio de substituição de liskov.

Utilizando o princípio de responsabilidade

src > modules > cars > useCases > createCategory > CreateCategoryUseCase.ts > ...

```ts
import { ICategoriesRepository } from "../../repositories/ICategoriesRepository";

interface IRequest {
  name: string;
  description: string;
}

class CreateCategoryUseCase {
  // precisa do private para acessar
  constructor(private categoriesRepository: ICategoriesRepository) {}

  execute({ name, description }: IRequest): void {
    // pega o categoriesRepository e procura pelo name
    const categoryAlreadyExists = this.categoriesRepository.findByName(name);

    if (categoryAlreadyExists) { // se já existir da um error
      throw new Error("Categoy already exists!"); // sempre q tiver error dentro do service
    }

    this.categoriesRepository.create({ name, description }); // aqui chama o
    categoriesRepository
  }
}

export { CreateCategoryUseCase };
```

Nas rotas

```
const categoriesRoutes = Router();
const categoriesRepository = new CategoriesRepository();

categoriesRoutes.post("/", (request, response) => {
  const { name, description } = request.body;

  const createCategoryService = new CreateCategoryService(categoriesRepository);

  createCategoryService.execute({ name, description });

  return response.status(201).send();
});
```

Utilizando o princípio da substituição de Liskov

```
src > modules > cars > repositories > ICategoriesRepository.ts > ICategoriesRepository > list
1   import { Category } from "../model/Category";
2
3   // DTO = Data Transfer Object
4   export interface ICreateCategoryDTO {
5     name: string;
6     description: string;
7   }
8
9   interface ICategoriesRepository {
10    findByName(name: string): Category;
11    list(): Category[];
12    create({ name, description }: ICreateCategoryDTO):void ;
13  }
14
15  export { ICategoriesRepository };
```

Em categories repository

```
// Responsável por cuidar das funcionalidade ao banco de dados etc
class CategoriesRepository implements ICategoriesRepository {

  private categories: Category[]; // do tipo array de Category, só o categories tem acesso
```

Criando servisse de especificação e separando em módulos

```
src > modules > cars > model > Specifications.ts > ...
1   import { v4 as uuidV4} from "uuid";
2
3   class Specifications {
4     id?: string; // id opcional
5     name: string;
6     description: string;
7     created_at: Date;
8
9     // constructor um método que é chamado quando a classe é instânciada, new () iniciar
10    constructor() {
11      if (!this.id) { // se ñ tiver nem um id desse Category
12        this.id = uuidV4(); // this id recebe um uuid
13      }
14    }
15  }
16
17  export { Specifications };
```

Módulos: pequenos blocos da aplicação.

Criando repostório de especificação

```
src > modules > cars > repositories > ISpecificationsRepository.ts > ...
1   import { Specifications } from "../model/Specifications";
2
3   interface ICreateSpecificationsDTO {
4     name: string;
5     description: string;
6   }
7
8   interface ISpecificationsRepository {
9     create({ name, description}: ICreateSpecificationsDTO): void;
10    findByName(name: string): Specifications;
11  }
12
13  export { ISpecificationsRepository, ICreateSpecificationsDTO };
```

Specification repositor

```
src > modules > cars > repositories > implementations > SpecificationsRepository.ts > ...
1   import { Specifications } from "../../model/Specifications";
2   import { ICreateSpecificationsDTO, ISpecificationsRepository } from "../ISpecificationsRepository";
3
4   class SpecificationRepository implements ISpecificationsRepository {
5     private specifications: Specifications[];
6
7     constructor() {
8       this.specifications = [];
9     }
10
11    create({ name, description }: ICreateSpecificationsDTO): void {
12      const specification = new Specifications();
13
14      Object.assign(specification, {
15        name,
16        description,
17        created_at: new Date(),
18      });
19
20      this.specifications.push(specification);
21    }
22
23    findByName(name: string): Specifications {
24      const specification = this.specifications.find(specification => specification.name === name);
25
26      return specification;
27    }
28  }
29
30  export { SpecificationRepository };
```

CreateSpecification Service

```
src > modules > cars > services > CreateSpecificationService.ts > CreateSpecificationService > execute
1   import { ISpecificationsRepository } from "../repositories/ISpecificatiosRepository";
2
3   interface IRequest {
4     name: string;
5     description: string;
6   }
7
8   class CreateSpecificationService {
9     constructor(private specificationsRepository: ISpecificationsRepository) {}
10    execute({ name, description }: IRequest): void {
11      const specificationAlreadyExists = this.specificationsRepository.findByName(
12        name
13      );
14
15      if (specificationAlreadyExists) {
16        throw new Error("Specification already exists!");
17      }
18
19      this.specificationsRepository.create({
20        name,
21        description,
22      });
23    }
24  }
25
26  export { CreateSpecificationService };
27
```

Rotas specifications

```ts
import { CreateSpecificationService } from "modules/cars/services/CreateSpecificationService";

const specificationsRoutes = Router();

const specificationsRepository = new SpecificationsRepository();

specificationsRoutes.post("/", (request, response) => {
  const { name, description } = request.body;
  const createSpecificationService = new CreateSpecificationService(
    specificationsRepository
  );

  createSpecificationService.execute({ name, description });

  return response.status(201).send();
});

export { specificationsRoutes };
```

Index da rotas

```ts
import { Router } from "express";

import { categoriesRoutes } from "./categories.routes";
import { specificationsRoutes } from "./specifications.routes";

const router = Router();

router.use("/categories", categoriesRoutes); // cria a rota categories path inicia com categories
router.use("/specifications", specificationsRoutes);
```

Criando Usecade de categoria

UseCases: as regras de negócios da aplicação.

Controllers: são classes que recebe a requisição, e retorna a resposta para quem está chamando. Responsável pelas requisições e respostas.

```ts
class CreateCategoryUseCase {
  // precisa do private para acessar
  constructor(private categoriesRepository: ICategoriesRepository) {}

  execute({ name, description }: IRequest): void {
    // pega o categoriesRepository e procura pelo name
```

```ts
import { Request, Response } from "express";

import { CreateCategoryUseCase } from "./CreateCategoryUseCase";

class CreateCategoryController {
  constructor(private createCategoryUseCase: CreateCategoryUseCase) {}

  handle(request: Request, response: Response): Response {
    const { name, description } = request.body; // pega o nome, descrição do body

    this.createCategoryUseCase.execute({ name, description });

    // status 201 algo criado com sucesso, se não manda json manda send
    return response.status(201).send();
  }
}

export { CreateCategoryController };
```

```ts
import { CategoriesRepository } from "../../repositories/implementations/CategoriesRepository";
import { CreateCategoryController } from "./CreateCategoryController";
import { CreateCategoryUseCase } from "./CreateCategoryUseCase";

const categoriesRepository = CategoriesRepository.getInstance();
const createCategoryUseCase = new CreateCategoryUseCase(categoriesRepository);
const createCategoryController = new CreateCategoryController(createCategoryUseCase);

export { createCategoryController };
```

```
// rota de criar categorias
categoriesRoutes.post("/", (request, response) => {
  return createCategoryController.handle(request, response);
});

// rota de listar categoria
categoriesRoutes.get("/", (request, response) => {
  return listcategoriesController.handle(request, response)
});
```

Refatorando a listagem de categoria

Conhecendo Singleton Pattern instância de uma classe que vai ser uma instância global. No categoriesRepository

```ts
class CategoriesRepository implements ICategoriesRepository {

  private categories: Category[]; // do tipo array de Category, só o categories tem acesso

  private static INSTANCE: CategoriesRepository; // para listar os produtos

  private constructor() {
    this.categories = []; // aqui que cria o categories
  }

  public static getInstance(): CategoriesRepository {
    if (!CategoriesRepository.INSTANCE) { //se não tem valores atribuido
      CategoriesRepository.INSTANCE = new CategoriesRepository();
    }

    return CategoriesRepository.INSTANCE; //se já tem valores manda ele
  }
```

No index do useCase do createcategory e no list

```ts
import { CategoriesRepository } from "../../repositories/implementations/CategoriesRepository";

const categoriesRepository = CategoriesRepository.getInstance();
```

Criando use case de especificação

src > modules > cars > useCases > createSpecifications > CreateSpecificationsController.ts > ...

```ts
import { Request, Response } from "express";

import { CreateSpecificationsUseCase } from "./CreateSpecificationsUseCase";

class CreateSpecificationsController {
  constructor(private createSpecificationsUseCase: CreateSpecificationsUseCase) {}

  handle(request: Request, response: Response): Response {
    const { name, description } = request.body;

    this.createSpecificationsUseCase.execute({ name, description });

    return response.status(201).send();
  }
}

export { CreateSpecificationsController };
```

src > modules > cars > useCases > createSpecifications > CreateSpecificationsUseCase.ts > ...

```ts
import { ISpecificationsRepository } from "../../repositories/ISpecificationsRepository";

interface IRequest {
  name: string;
  description: string;
}

class CreateSpecificationsUseCase {
  constructor(private specificationsRepository: ISpecificationsRepository) {}

  execute({ name, description }: IRequest): void {
    const specificationsAlreadyExists = this.specificationsRepository.findByName(name);

    if (specificationsAlreadyExists) {
      throw new Error("Specification already exists!");
    }

    this.specificationsRepository.create({
      name,
      description,
    });
  }
}

export { CreateSpecificationsUseCase };
```

```
src > modules > cars > useCases > createSpecifications > index.ts > ...
1  import { CreateSpecificationsUseCase } from "./CreateSpecificationsUseCase";
2  import { CreateSpecificationsController } from "./CreateSpecificationsController";
3  import { SpecificationRepository } from "../../repositories/implementations/SpecificationsRepository";
4
5  const specificationsRepository = new SpecificationRepository();
6  const createSpecificationsUseCase = new CreateSpecificationsUseCase(specificationsRepository);
7  const createSpecificationsController = new CreateSpecificationsController(createSpecificationsUseCase);
8
9  export { createSpecificationsController };
```

Refatorando as rotas criar um index dentro das rotas

```
src > routes > index.ts > ...
1  import { Router } from "express";
2
3  import { categoriesRoutes } from "./categories.routes";
4  import { specificationsRoutes } from "./specifications.routes";
5
6  const router = Router();
7
8  router.use("/categories", categoriesRoutes); // cria a rota categories path inicia com categories
9  router.use("/specifications", specificationsRoutes);
10
11 export { router };
```

Trabalhando com Upload

UpLoad de arquivos. Usa o multer é como um middleware dentro da rota.

Yarn add multer

Yarn add @types/multer -D

Criar pasta tmp. E nas rotas de categories coloca a configuração do multer.

```
const upload = multer({
  dest: "./tmp",
});
```

Single por quê só quer upload de um arquivo, e dentro do single coloca um nome.

Ler o arquivo.

```
categoriesRoutes.post("/import", upload.single("file"), (request, response) => {
  const { file } = request;
  console.log(file);
  return response.send();
});
```

Criando o use case para importar categorias

```
src > modules > cars > useCases > importCategory > index.ts > ...
1  import { CategoriesRepository } from "../../repositories/implementations/CategoriesRepository";
2  import { ImportCategoryController } from "./ImportCategoryController";
3  import { ImportCategoryUseCase } from "./ImportCategoryUseCase";
4
5  const categoriesRepository = CategoriesRepository.getInstance();
6  const importCategoryUseCase = new ImportCategoryUseCase(categoriesRepository);
7  const importCategoryController = new ImportCategoryController(importCategoryUseCase);
8
9  export { importCategoryController };
```

src > modules > cars > useCases > importCategory > ImportCategoryController.ts > ...

```ts
import { Request, Response } from "express";

import { ImportCategoryUseCase } from "./ImportCategoryUseCase";

class ImportCategoryController {
  constructor(private importCategoryUseCase: ImportCategoryUseCase) {}

  handle(request: Request, response: Response): Response {
    const { file } = request;

    this.importCategoryUseCase.execute(file);

    return response.send();
  }
}

export { ImportCategoryController };
```

Conhecendo conceito de streaming

READFILE faz a leitura tudo de uma vez do arquivo, strimming: permite leia arquivos por partes, sem precisar consumir muita memória.

FS, a função creatReadStream( ), permite que faça a leitura do arquivo em ponto.

E recebe o path do arquivo. Usa o **PIPI**, pega o streaming que está sendo lido e dentro dele, ele joga oque foi lido para o lugar que a gente determinar.

LIB CSV PARSE: Yarn add csv-parse

src > modules > cars > useCases > importCategory > ImportCategoryUseCase.ts > ImportCategoryUseCase > loadCategories > <function>

```ts
import fs from "fs";
import { parse } from "csv-parse";

import { ICategoriesRepository } from "../../repositories/ICategoriesRepository";

interface IImportCategory {
  name: string;
  description: string;
}

class ImportCategoryUseCase {
  constructor(private categoriesRepository: ICategoriesRepository) {}


  loadCategories(file: Express.Multer.File): Promise<IImportCategory[]> {
    return new Promise((resolve, reject) => {
      const stream = fs.createReadStream(file.path); // faz leitura em parte
      const categories: IImportCategory[] = [];

      const parseFile = parse();

      // pipe pega oque foi lido e joga pra onde nos quer
      stream.pipe(parseFile);

      parseFile.on("data", async (line) => {
        const [name, description] = line;
        categories.push({ name, description });
      })
      .on("end", () => {
        fs.promises.unlink(file.path); // para remover os arquivos do tmp
        resolve(categories);
      })
      .on("error", (err) => {
        reject(err);
      });
    });
  }
```

```
async execute(file: Express.Multer.File): Promise<void> {
  const categories = await this.loadCategories(file);

  categories.map(async (category) => {
    const { name, description } = category;

    const existCategory = this.categoriesRepository.findByName(name); // existe um categoria

    if (!existCategory) { // se ñ tem cria
      this.categoriesRepository.create({
        name,
        description,
      });
    }
  });
 }
}

export { ImportCategoryUseCase };
```

SWAGGER DOCUMENTAÇÃO: yarn add swagger-ui-express

Yarn add @types/swagger-ui-express -D

Criar dentro do server, 1 a rota da documentação, 2 chama o server do Swagger, 3 o setup é o arquivo json onde está as informações da aplicação.

Para resolver o problema de importação do Swagger.

```
 resolving a module. */
"resolveJsonModule": true,
// "noResolve": true,
```

```
import swaggerUi from "swagger-ui-express";

import swaggerFile from "./swagger.json";

import { router } from "./routes";

const app = express();

app.use(express.json());

app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerFile))

app.use(router);
```

DOCKER

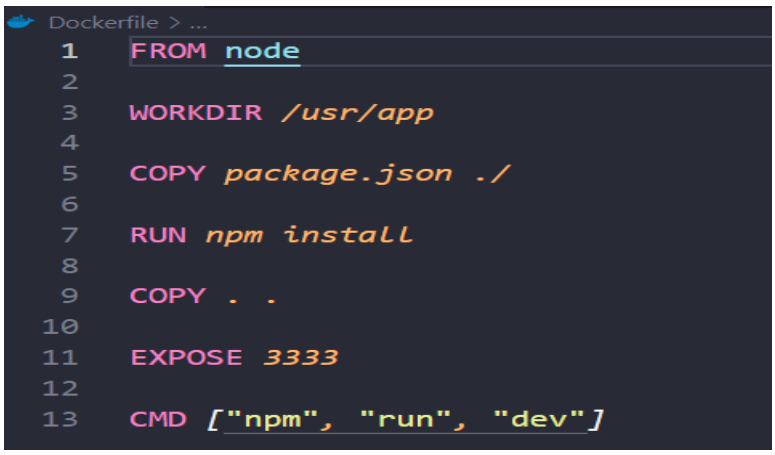
- Ferramenta para criação de containers
- Container: Ambiente isolado
- Imagens: Instruções para criação de um container
- O que “roda” localmente “roda” em produção
- Mesmo SO, compartilhando recursos da máquina host

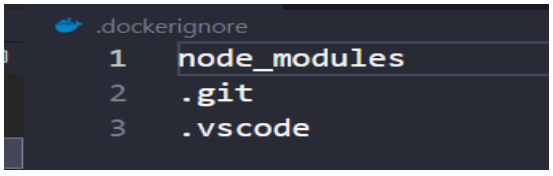
Workdir: define uma pasta onde as informações estejam contidas, /usr/app

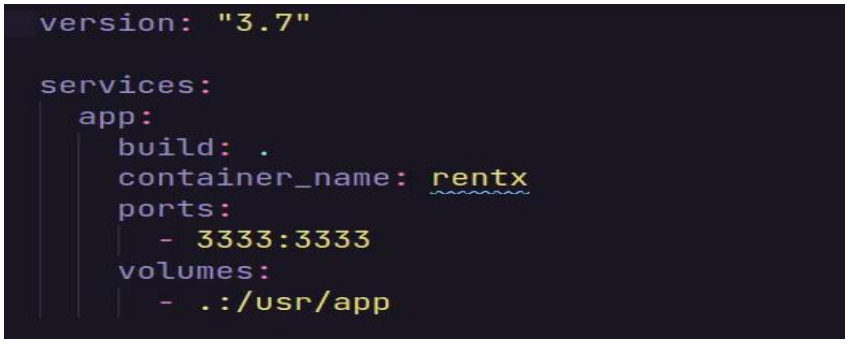
COPY para copiar as dependências

RUN instala as dependências no Docker

Comando de rodar o Docker: **Docker build -t rentx .**







Docker ps = ver os container rodadando

Docker run -p 3333:3333 rentx = rodar a imagem

Docker compose é um orquestador de container

Docker-compose up = para builda e executa em real time, e de da -d roda em background

Docker logs nome -f = mostra toda execução em real time.

Comandos docker

Docker ps = mostra lista dos container de pé

Docker ps -a = lista todo container de pé e que não estão

Docker rm name-id = remover container

Docker start name = inicia o container

Docker stop id-container = para o container

Docker-compose up -D = sobe o container em background

Docker-compose stop = para o Docker compose

Docker-compose down = remove tudo que tiver criado dentro do serviço

Docker exec -it namecontainer /bin/bash = acessa o container.

Docker logs nameimage -f = acompanhar os logs

Docker-compose up –force-recreate -d = força recria novamente

Banco de dados Postgrees

Yarn add typeorm reflect-metadata

Yarn add pg

Desabilitar no ts-config

```
// "jsx": "preserve",
"experimentalDecorators": true,
draft decorators. */
"emitDecoratorMetadata": true,
declarations in source files. */
```

E criano src database uma pasta, e um index.ts

Criar no Docker a image postgres

```
import { DataSource } from "typeorm";
const dataSource = new DataSource({
  type: "postgres",
  host: "localhost",
  port: 5432,
  username: "docker",
  password: "ignite",
  database: "rentx",
});

dataSource.initialize();
```

$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' database_ignite

ormconfig.json

```json
{
  "type": "postgres",
  "port": 5432,
  "host": "localhost",
  "username": "docker",
  "password": "ignite",
  "database": "rentx",
  "migrations": ["./src/database/migrations/*.ts"],
  "entities": ["./src/modules/**/entities/*.ts"],
  "cli": {
    "migrationsDir": "./src/database/migrations"
  }
}
```

docker-compose.yml

```yaml
version: "3.9"

services:
  database:
    image: postgres
    container_name: database_ignite
    restart: always
    ports:
      - 5432:5432
    environment:
      - POSTGRES_USER=docker
      - POSTGRES_PASSWORD=ignite
      - POSTGRES_DB=rentx
    volumes:
      - pgdata:/data/postgres


  app:
    build: .
    container_name: rentx
    restart: always
    ports:
      - 3333:3333
      - 9229:9229
    volumes:
      - .:/usr/app
    links:
      - database
    depends_on:
      - database

volumes:
  pgdata:
    driver: local
```

typeORM

https://www.notion.so/Refatora-o-Docker-com-TypeORM-4500fc0d075349ac9b97d670e734d41b

https://efficient-sloth-d85.notion.site/Atualizando-o-TypeORM-no-Rentx-7988bcb23f9f417197fcf2113a74161a

github: https://github.com/rocketseat-education/ignite-nodejs-rentx

```
"typeorm": "^0.2.31",
"pg": "^8.5.1",
```

```
"reflect-metadata": "^0.1.13",
```

Migrations

Yarn typeorm migration:create -n CreateCategories

Criar pasta migrations dentro de database e no ormconfig e no package em script

```
ts ,
"typeorm": "ts-node-dev ./node_modules/typeorm/cli"
```

```
  "cli": {
    "migrationsDir": "./src/database/migrations"
  }
}
```

```
import {MigrationInterface, QueryRunner, Table } from "typeorm";

export class CreateCategories1669381715993 implements MigrationInterface {

  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.createTable(
      new Table({
        name: "categories",
        columns: [
          {
            name: "id",
            type: "uuid",
            isPrimary: true,
          },
          {
            name: "name",
            type: "varchar",
          },
          {
            name: "description",
            type: "varchar",
          },
          {
            name: "created_at",
            type: "timestamp",
            default: "now()"
          },
        ],
      })
    );
  }

  public async down(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.dropTable("categories");
  }

}
```

Down se da errado ele deleta a tabela

Depois no terminal

Yarn typeorm migration:run

Criando entidade de categoria

modules > cars > entities > Category.ts > ...

```ts
import { v4 as uuidV4 } from "uuid";
import { Entity, Column, PrimaryColumn, CreateDateColumn } from "typeorm";

@Entity("categories")
// Modelo da classe Category
class Category {

  @PrimaryColumn()
  id?: string; // id opcional

  @Column()
  name: string;

  @Column()
  description: string;

  @CreateDateColumn()
  created_at: Date;

  // constructor um método que é chamado quando a classe é instânciada, new () iniciar
  constructor() {
    if (!this.id) { // se ñ tiver nem um id desse Category
      this.id = uuidV4(); // this id recebe um uuid
    }
  }
}

export { Category }
```

No seu **package.json**, adicione o script do typeorm:

"typeorm": "ts-node-dev -r tsconfig-paths/register ./node_modules/typeorm/cli.js -d src/database/index.ts"

No arquivo **index.ts** da pasta **database**, altere o conteúdo para:

```ts
import { DataSource } from "typeorm";



const AppDataSource = new DataSource({

 type: "postgres",

 host: "localhost",

 port: 5432,

 username: "docker",

 password: "ignite",

 database: "rentx",

});
```

```
export function createConnection(host = "database_ignite"): Promise<DataSource> {
  return AppDataSource.setOptions({ host }).initialize();
}

export default AppDataSource;
```

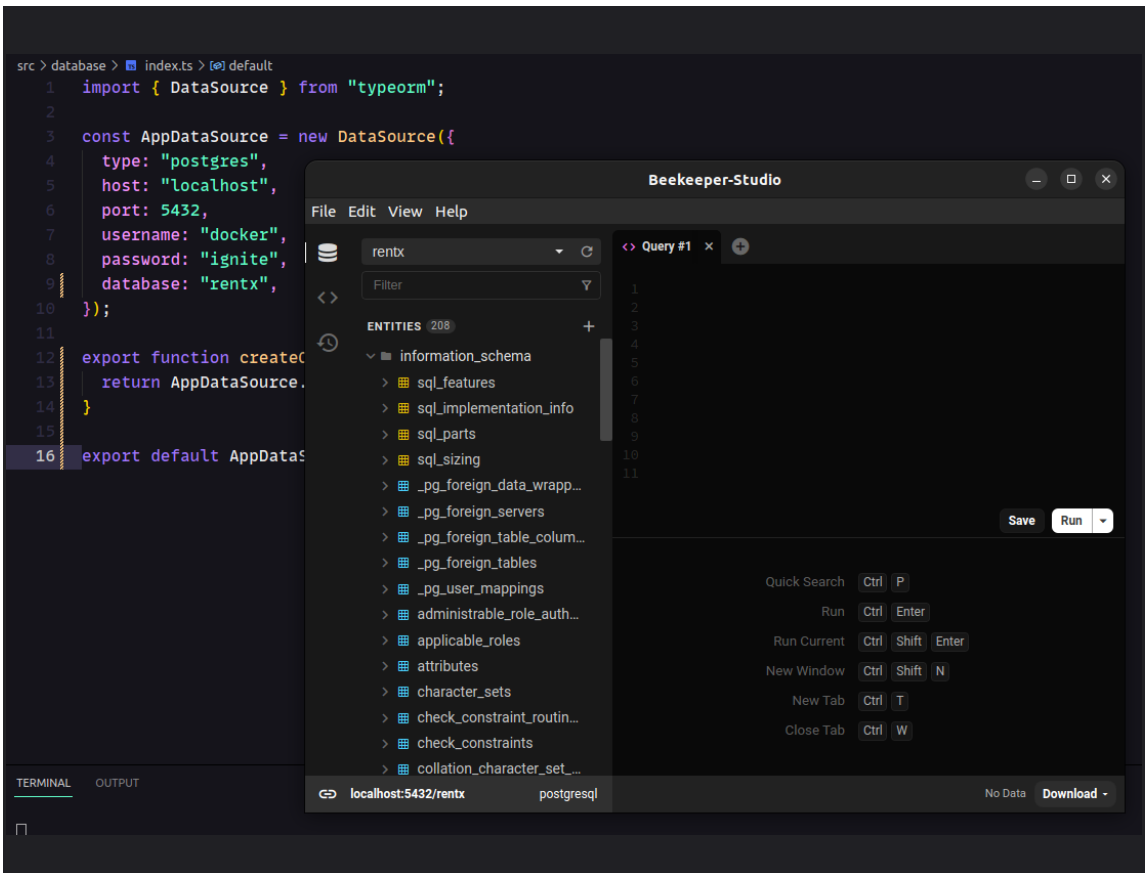
No arquivo **server.ts**, mude a importação do banco para:

```
import { createConnection } from "./database";

createConnection();
```

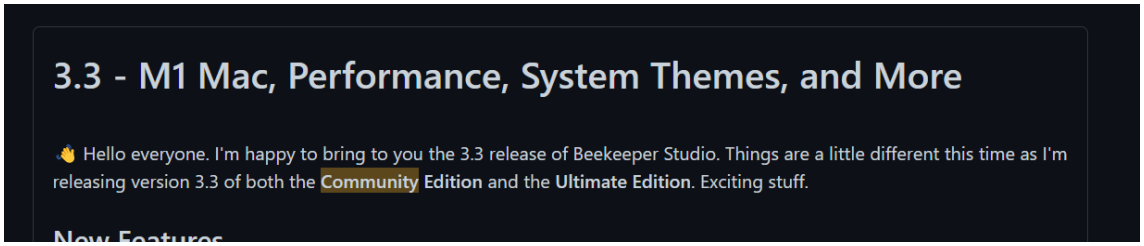
Por fim, atualize o seu **docker-compose.yml** para:

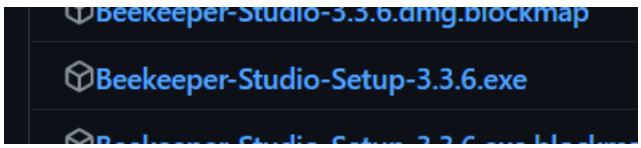
```
version: "3.9"

services:
  database_ignite:
    image: postgres
    container_name: database_ignite
    restart: always
    ports:
      - 5432:5432
    environment:
      - POSTGRES_USER=docker
```

```
    - POSTGRES_PASSWORD=ignite
    - POSTGRES_DB=rentx
  volumes:
    - pgdata:/data/postgres

app:
  build: .
  container_name: rentx
  restart: always
  ports:
    - 3333:3333
  volumes:
    - .:/usr/app
  links:
    - database_ignite
  depends_on:
    - database_ignite

volumes:
 pgdata:
  driver: local
```



https://github.com/beekeeper-studio/beekeeper-studio/releases?q=Community+&expanded=true





```typescript
@Entity("categories")
// Modelo da classe Category
class Category {

  @PrimaryColumn()
  id?: string; // id opcional

  @Column()
  name: string;

  @Column()
  description: string;

  @CreateDateColumn()
  created_at: Date;

  // constructor um método que é char
  constructor() {
    if (!this.id) { // se ñ tiver ner
      this.id = uuidV4(); // this id
```

```typescript
private repository: Repository<Category>; // acesso só internamente

private static INSTANCE: CategoriesRepository; // para listar os produtos

private constructor() {
  this.repository = getRepository(Category);
}

public static getInstance(): CategoriesRepository {
  if (!CategoriesRepository.INSTANCE) {
    CategoriesRepository.INSTANCE = new CategoriesRepository();
  }

  return CategoriesRepository.INSTANCE;
}
```

```typescript
async create({ name, description}: ICreateCategoryDTO): Promise<void> { // função sem retorno
  const category = this.repository.create({
    description,
    name,
  }); // cria a entidade para poder salvar

  // manda pro categories name, descrição, uuid
  await this.repository.save(category)
}

async list(): Promise<Category[]> { // função q retorna a lista de Category
  const categories = await this.repository.find(); // retorna uma lista
  return categories;
}

async findByName(name: string): Promise<Category> { // função de verificação de duplicação do name
  // SELECT * FROM categories WHERE name = "name" LIMIT 1
  const category = await this.repository.findOne({ name })
  return category;
}
}
```

```ts
interface ICategoriesRepository {
  findByName(name: string): Promise<Category>;
  list(): Promise<Category[]>;
  create({ name, description }: ICreateCategoryDTO): Promise<void> ;
}

export { ICategoriesRepository };
```

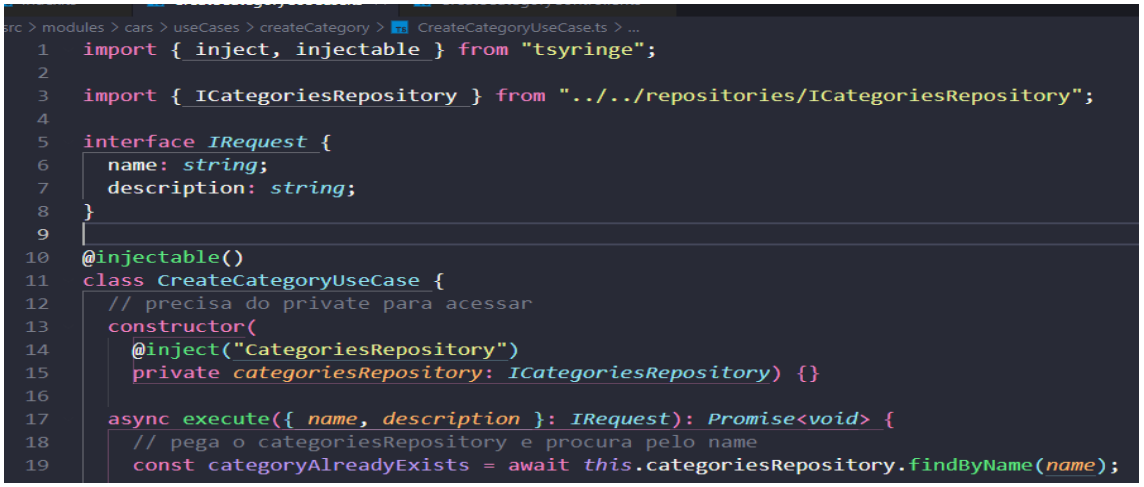
Yarn typeorm migration:run / cria as tabelas

Yarn typeorm migration:revert / desfaz as tabelas

Injeção de dependência baixar o [yarn add tsyringe]

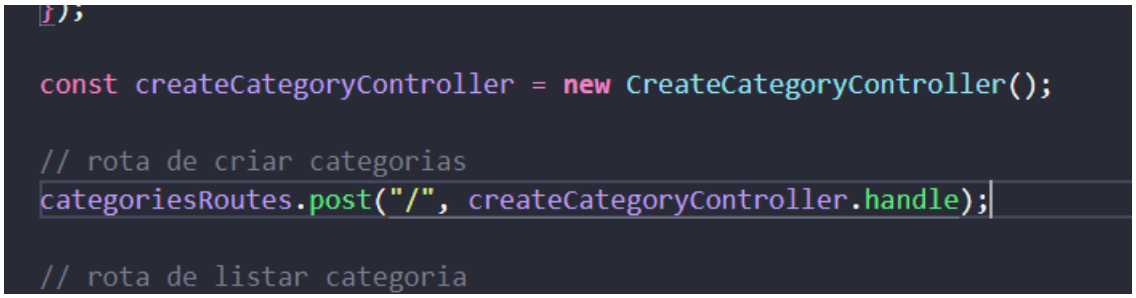
Cria pasta em src shared e container outra pastas dentro de shred.

No useCase

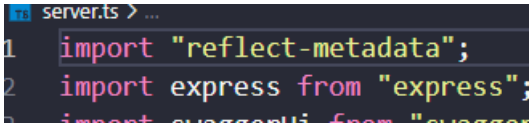


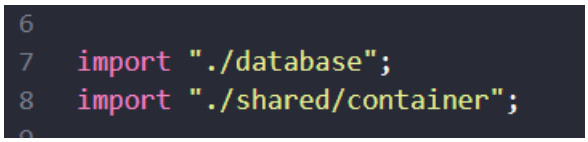
No controller, e remove o index.ts

E nas rotas passa assim



E chama no serve o container





Criando Users

Yarn typeorm migration: create -n aidaonome

```typescript
export class CreateUsers1669976709139 implements MigrationInterface {

  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.createTable(
      new Table({
        name: "users",
        columns: [
          {
            name: "id",
            type: "uuid"
          },
          {
            name: "name",
            type: "varchar"
          },
          {
            name: "username",
            type: "varchar",
            isUnique: true
          },
          {
            name: "isAdmin",
            type: "boolean",
            default: false
          },
          {
            name: "created_at",
            type: "timestamp",
            default: "now()"
          },
        ],
      })
    );
  }

  public async down(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.dropTable("users");
  }

}
```

Depois yarn typeorm migration:run para criar a tabela

Criar pasta account com a entities user.ts

Criando repositórios. Dentro IUserrespository e pasta implementations e dentro UserRepository

```ts
import { Repository, getRepository } from "typeorm";

import { ICreateUserDTO } from "../../dtos/ICreateUserDTO";
import { IUsersRepository } from "../IUsersRepository";
import { User } from "../../entities/User";


class UsersRepository implements IUsersRepository {
  private repositoriy: Repository<User>

  constructor() {
    this.repositoriy = getRepository(User);
  }

  async create({ name, password, username, email, driver_license }: ICreateUserDTO): Promise<void> {
    const user = this.repositoriy.create({
      name, password, username, email, driver_license
    });

    await this.repositoriy.save(user);
  }
}

export { UsersRepository }
```

Cria pasta DTO

```ts
export interface ICreateUserDTO {
  name: string;
  username: string;
  password: string;
  email: string;
  driver_license: string;
}
```

Cria pasta useCases e dentr outra pasta createUser e dentro a createUserUsercase.ts e createusercontroller.ts

```ts
import { inject, injectable } from "tsyringe";

import { ICreateUserDTO } from "../../dtos/ICreateUserDTO";
import { IUsersRepository } from "../../repositories/IUsersRepository";


@injectable()
class CreateUserUseCase {

  constructor(
    @inject("UsersRepository")
    private usersRepository: IUsersRepository
  ) {}

  async execute({ name, username, password, email, driver_license }: ICreateUserDTO): Promise<void> {
    await this.usersRepository.create({
      name, username, password, email, driver_license
    });
  }
}

export { CreateUserUseCase }
```

Controller

```
modules > accounts > useCases > createUser > CreateUserController.ts > CreateUserController > handle
import { Request, Response } from "express";
import { container } from "tsyringe";

import { CreateUserUseCase } from "./CreateUserUseCase";


class CreateUserController {
  async handle(request: Request, response: Response): Promise<Response> {
    const { name, username, email, password, driver_license } = request.body;
    const createUserUseCase = container.resolve(CreateUserUseCase);

    await createUserUseCase.execute({
      name, username, email, password, driver_license
    });

    return response.status(201).send();
  }
}

export { CreateUserController }
```

E no container cria

```
container.registerSingleton<IUsersRepository>(
  "UsersRepository",
  UsersRepository
);
```

Rotas agora

```
src > routes > users.routes.ts > ...
import { Router } from "express";
import { CreateUserController } from "../modules/accounts/useCases/createUser/CreateUserController";

const usersRoutes = Router();

const createUserController = new CreateUserController();

usersRoutes.post("/", createUserController.handle);

export { usersRoutes }
```

E no index da rota

```
src > routes > index.ts > ...
import { Router } from "express";

import { categoriesRoutes } from "./categories.routes";
import { specificationsRoutes } from "./specifications.routes";
import { usersRoutes } from "./users.routes";

const router = Router();

router.use("/categories", categoriesRoutes); // cria a rota categories path inicia com categories
router.use("/specifications", specificationsRoutes);
router.use("/users", usersRoutes);

export { router };
```

Alterar tabelas yarn typeorm migration:create -n ALterUserDeleteUsername

```
import { MigrationInterface, QueryRunner, TableColumn } from "typeorm";

export class AlterUserDeleteUsername1669979467860 implements MigrationInterface {

  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.dropColumn("users", "username");
  }

  public async down(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.addColumn("users",
      new TableColumn({
        name: "username",
        type: "varchar"
      })
    )
  }

}
```

Criptografia de senha. Yarn add bcryptjs yarn add @types/bcryptjs -D

Recebe a senha e o salt que o salt melhor e o 8

```
16
17    async execute({ name, password, email, driver_license }: ICreateUserDTO): Promise<void> {
18      const passwordHash = await hash(password, 8);
19
20      await this.usersRepository.create({
21        name, password: passwordHash, email, driver_license
22      });
23    }
24  }
```

No authentication importa compare do bcriptjs

```
const passwordMatch = await compare(password, user.password);

if (!passwordMatch) {
  throw new Error("Email or password incorrect!");
}
```

Validando email

```
src > modules > accounts > repositories > IUsersRepository.ts > ...
1   import { ICreateUserDTO } from "../dtos/ICreateUserDTO";
2   import { User } from "../entities/User";
3
4
5   interface IUsersRepository {
6     create(data: ICreateUserDTO): Promise<void>;
7     findByEmail(email: string): Promise<User>;
8   }
```

userrepository

```
  async findByEmail(email: string): Promise<User> {
    const user = await this.repositoriy.findOne({ email });

    return user;
  }
}
```

Usecase

```
async execute({ name, password, email, driver_license }: ICreateUserDTO): Promise<void> {
  const userAlreadyExists = await this.usersRepository.findByEmail(email);

  if (userAlreadyExists) {
    throw new Error("User already exists!");
  }
```

JWT JSON WEB TOKEN yarn add jasonwebtoken yarn add @types/jsonwebtoken -D

Dentro do usercase cria pasta authenticateUser e cria o UseCase e o Controller.

No useCase

No Controller

```
interface IResponse {
  user: {
    name: string;
    email: string;
  },
  token: string;
}
```

Usecase authentication

```ts
async execute({ email, password }: IRequest): Promise<IResponse> {
  // usuario existe
  const user = await this.usersRepository.findByEmail(email);

  // senha está correta
  if (!user) {
    throw new Error("Email or password incorrect!");
  }

  const passwordMatch = await compare(password, user.password);

  if (!passwordMatch) {
    throw new Error("Email or password incorrect!");
  }

  // 1 = payload info n critica, 2 = palavrasecreta 'jromarioss', 3 = obj
  const token = sign({}, "c82fdf0ebf03c6e8494030621c6d21b3", {
    subject: user.id,
    expiresIn: "1d"
  });

  const TokenReturn: IResponse = {
    token,
    user: {
      name: user.name,
      email: user.email
    }
  }

  return TokenReturn;

  // gerar o jsonwebtoken
}
```

Controller

```ts
import { Request, Response } from "express";
import { container } from "tsyringe";

import { AuthenticateUserUseCase } from "./authenticateUserUseCase";


class AuthenticateUserController {
  async handle(request: Request, response: Response): Promise<Response> {
    const { password, email } = request.body;

    const authenticateUserUsercase = container.resolve(AuthenticateUserUseCase);

    const token = await authenticateUserUsercase.execute({ email, password });

    return response.json(token);
  }
}

export { AuthenticateUserController }
```

E cria rotas authotenticaton

```ts
import { Router } from "express";
import { AuthenticateUserController } from "../modules/accounts/useCases/authenticateUser/AuthenticateUserController";

const authenticateRoutes = Router();

const authenticateUserController = new AuthenticateUserController();


authenticateRoutes.post("/sessions", authenticateUserController.handle);

export { authenticateRoutes };
```

E no idnex de rotas

```
router.use(authenticateRoutes);
```

Rotas autenticadas criar um middleware para verificar se é um token valido.

Funciona com o Beare Token

```
  async findById(id: string): Promise<User> {
    const user = await this.repositoriy.findOne(id);

    return user;
  }
}
```

```
interface IUsersRepository {
  create(data: ICreateUserDTO): Promise<void>;
  findByEmail(email: string): Promise<User>;
  findById(id: string): Promise<User>;
}

export { IUsersRepository }
```

Cria pasta middleware

```
> middlewares > ensureAuthenticated.ts > ensureAuthenticated > user
import { NextFunction, Request, Response } from "express";
import { verify } from "jsonwebtoken";

import { UsersRepository } from "../modules/accounts/repositories/implementations/UsersRepository";

interface IPayload {
  sub: string;
}

export async function ensureAuthenticated(request: Request, response: Response, next: NextFunction) {
  // Bearer token dawdad vem dentro do headers o token vem dentro do headers
  const authHeader = request.headers.authorization;

  if (!authHeader) { // se o ehader vem vazio
    throw new Error("Token missing");
  }

  // Bearer token adawdawfsefse o split divide pelo espaço, [bearer, token] pega posicao 2
  const [, token] = authHeader.split(" ");

  try {
    // sub é o id do user
    const { sub: user_id } = verify(token, "c82fdf0ebf03c6e8494030621c6d21b3") as IPayload;

    // verificar se o usuário existe
    const userRepository = new UsersRepository();

    const user = await userRepository.findById(user_id);

    if (!user) {
      throw new Error("User does not exists!");
    }

    next();
  } catch(error) {
    throw new Error("Invalid token!");
  }
}
```

Alterar tabelas

```
> database > migrations > 1669979467860-AlterUserDeleteUsername.ts > AlterUserDeleteUsername1669979467860
import { MigrationInterface, QueryRunner, TableColumn } from "typeorm";

export class AlterUserDeleteUsername1669979467860 implements MigrationInterface {

    public async up(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.dropColumn("users", "username");
    }

    public async down(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.addColumn("users",
            new TableColumn({
                name: "username",
                type: "varchar"
            })
        )
    }
}
```

Cria pasta erros

```
> errors > AppError.ts > AppError > constructor
export class AppError {
    public readonly message: string;

    public readonly statusCode: number;

    constructor(message: string, statusCode = 40
        this.message = message;
        this.statusCode = statusCode;
    }
}
```

E coloca appero em tudo e o statuscode

```
if (!authHeader) { // se o ehader vem vazio
    throw new AppError("Token missing", 401);
}
```

E no server dps ds rotas

```
app.use((err: Error, request: Request, response: Response, next: NextFunction) => {
    if (err instanceof AppError) {
        return response.status(err.statusCode).json({
            message: err.message
        })
    }

    return response.status(500).json({
        status: "error",
        message: `Internal server error - ${err.message}`
    });
});

app.listen(3333, () => {
    console.log("Servis is running!");
});
```

E instalar [yarn add express-async-errors]

```
import "reflect-metadata";
import express, { NextFunction, Request, Response } from "express";
import "express-async-errors";
import swaggerUi from "swagger-ui-express";

import { router } from "./routes";
```

Avatar

database > migrations > 1670062489492-AlterUserAddAvatar.ts > AlterUserAddAvatar1670062489492 > down

```
import {MigrationInterface, QueryRunner, TableColumn} from "typeorm";

export class AlterUserAddAvatar1670062489492 implements MigrationInterface {

    public async up(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.addColumn("users",
            new TableColumn({
                name: "avatar",
                type: "varchar",
                isNullable: true
            })
        )
    }

    public async down(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.dropColumn("users", "avatar");
    }

}
```

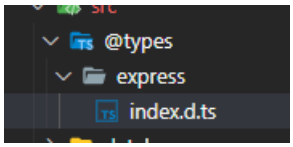
Usecase

```ts
import { inject, injectable } from "tsyringe";
import { IUsersRepository } from "../../repositories/IUsersRepository";

interface IRequest {
  user_id: string;
  avatar_file: string;
}

@injectable()
class UpdateUserAvatarUseCase {
  constructor(
    @inject("UsersRepository")
    private usersRepository: IUsersRepository
  ) {}

  async execute({ user_id, avatar_file }: IRequest): Promise<void> {
    const user = await this.usersRepository.findById(user_id);

    user.avatar = avatar_file;

    await this.usersRepository.create(user);
  }
}

export { UpdateUserAvatarUseCase }
```

Controller

```ts
modules > accounts > useCases > updateUserAvatar > UpdateUserAvatarController.ts > ...
import { Request, Response } from "express";
import { container } from "tsyringe";

import { UpdateUserAvatarUseCase } from "./UpdateUserAvatarUseCase";


class UpdateUserAvatarController {
  async handle(request: Request, response: Response): Promise<Response> {
    const { id } = request.user;

    // receber arquivo
    const avatar_file = null;

    const updateUserAvatarUseCase = container.resolve(UpdateUserAvatarUseCase);

    await updateUserAvatarUseCase.execute({ user_id: id, avatar_file});

    return response.status(200).send();
  }
}

export { UpdateUserAvatarController }
```

Criar a pasta, sobreescrever types




```ts
declare namespace Express {
  export interface Request {
    user: {
      id: string;
    }
  }
}
```

Criar pasta config e upload.ts

```ts
import multer from "multer";
import { resolve } from "path";
import crypto from "crypto";

export default {
  upload(folder: string) {
    return {
      storage: multer.diskStorage({
        destination: resolve(__dirname, "..", "..", folder),
        filename: (request, file, callback) => {
          const fileHash = crypto.randomBytes(16).toString("hex");
          const fileName = `${fileHash}-${file.originalname}`;

          return callback(null, fileName);
        }
      })
    }
  }
}
```

Nas rotas

```ts
import { UpdateUserAvatarController } from "../modules/accounts/useC
import uploadConfig from "../config/upload";

const usersRoutes = Router();

const uploadAvatar = multer(uploadConfig.upload("./tmp/avatar"))

const createUserController = new CreateUserController();
```

```ts
usersRoutes.patch("/avatar", uploadAvatar.single("avatar"), updateUserAvatarControler.handle);

export { usersRoutes }
```

E no controller

```
class UpdateUserAvatarController {
  async handle(request: Request, response: Response): Promise<Response> {
    const { id } = request.user;
    // receber arquivo
    const avatar_file = request.file.filename;
```

Deletando arquivo existente

```
src > config > file.ts > deleteFile
1   import fs from "fs";
2
3   export const deleteFile = async(filename: string) => {
4     // statverifica se um arquivo existe
5     try {
6       await fs.promises.stat(filename);
7
8     } catch(err) {
9       return;
10    }
11
12    // responsavel por remover o arquivo
13    await fs.promises.unlink(filename);
14  }
```

E no usercase do updateavatar

```
async execute({ user_id, avatar_file }: IRequest): Promise<void> {
  const user = await this.usersRepository.findById(user_id);

  if (user.avatar) {
    await deleteFile(`./tmp/avatar/${user.avatar}`);
  }
```

Testes

Instalar jest [yarn add jest @types/jest -D yarn ts-jest -D]

Yarn jest –init

```
// A preset that is
preset: "ts-jest",
// The glob patterns Jest us
testMatch: ["**/*.spec.ts"],
// Stop runn
bail: true,
```

```
// agrupa os teste
describe("Criar categoria", () => {

  it("Espero que 2 + 2 seja 4", () => {
    const soma = 2 + 2;
    const resultado = 4;

    expect(soma).toBe(resultado);
  });

  it("Espero que 2 + 2 não seja 5", () => {
    const soma = 2 + 2;
    const resultado = 5;

    expect(soma).not.toBe(resultado);
  })
});
```

No repositories cria pasta in-memory para faze criação na memoria para n fazer teste no banco de dados pq n é a responsabilidade dos teste unitários.

```
modules > cars > repositories > in-memory > CategoriesRepositoryInMemory.ts > CategoriesRepositoryInMemory > findByName
import { Category } from "../../entities/Category";
import { ICategoriesRepository, ICreateCategoryDTO } from "../ICategoriesRepository";

class CategoriesRepositoryInMemory implements ICategoriesRepository {

  categories:Category[] = [];

  async findByName(name: string): Promise<Category> {
    const category = this.categories.find(category => category.name === name);
    return category;
  }

  async list(): Promise<Category[]> {
    const list = this.categories;
    return list
  }

  async create({ name, description }: ICreateCategoryDTO): Promise<void> {
    const category = new Category();

    Object.assign(category, {
      name, description
    });

    this.categories.push(category);
  }
}

export { CategoriesRepositoryInMemory }
```

```typescript
import { AppError } from "../../../../errors/AppError";
import { CategoriesRepositoryInMemory } from "../../repositories/in-memory/CategoriesRepositoryInMemory";
import { CreateCategoryUseCase } from "./CreateCategoryUseCase";

let createCategoryUseCase: CreateCategoryUseCase;
let categoriesRepositoryInMemory: CategoriesRepositoryInMemory;

// agrupa os teste
describe("Create Category", () => {

  // faça antes do teste
  beforeEach(() => {
    categoriesRepositoryInMemory = new CategoriesRepositoryInMemory();
    createCategoryUseCase = new CreateCategoryUseCase(categoriesRepositoryInMemory);
  })

  it("should be able to create a new category", async () => {
    const category = {
      name: "Category Test",
      description: "Category description Test",
    }

    await createCategoryUseCase.execute({
      name: category.name,
      description: category.description
    });

    const categoryCreated = await categoriesRepositoryInMemory.findByName(category.name);

    // se tiver a propriedade id foi criada
    expect(categoryCreated).toHaveProperty("id");
  });
```

```typescript
  it("should not be able to create a new category with the same name", async () => {

    expect(async () => {
      const category = {
        name: "Category Test",
        description: "Category description Test",
      }

      await createCategoryUseCase.execute({
        name: category.name,
        description: category.description
      });

      await createCategoryUseCase.execute({
        name: category.name,
        description: category.description
      });
    }).rejects.toBeInstanceOf(AppError)
  });
```

Teste user

```
odules > accounts > repositories > in-memory > UsersRepositoryInMemory.ts > ...
import { ICreateUserDTO } from "../../dtos/ICreateUserDTO"
import { User } from "../../entities/User"
import { IUsersRepository } from "../IUsersRepository"


class UsersRepositoryInMemory implements IUsersRepository {
  users: User[] = [];


  async create({ email, name, password, driver_license }: ICreateUserDTO): Promise<void> {
    const user = new User();

    Object.assign(user, {
      email, name, password, driver_license
    });

    this.users.push(user)
  }

  async findByEmail(email: string): Promise<User> {
    return this.users.find((user) => user.email === email);
  }

  async findById(id: string): Promise<User> {
    return this.users.find((user) => user.id === id);
  }
}

export { UsersRepositoryInMemory }
```

Teste de criar o usuário

```
let authenticateUserUseCase: AuthenticateUserUseCase;
let usersRepositoryInMemory: UsersRepositoryInMemory;
let createUserUserCase: CreateUserUseCase;

describe("Authenticate User", () => {
  beforeEach(() => {
    usersRepositoryInMemory = new UsersRepositoryInMemory();
    authenticateUserUseCase = new AuthenticateUserUseCase(usersRepositoryInMemory);
    createUserUserCase = new CreateUserUseCase(usersRepositoryInMemory);
  });

  it("should be able to authenticate a user", async () => {
    // criando usuario
    const user: ICreateUserDTO = {
      driver_license: "000123",
      email: "user@teste.com",
      password: "12345",
      name: "User Teste",
    }

    await createUserUserCase.execute(user);

    const result = await authenticateUserUseCase.execute({
      email: user.email,
      password: user.password,
    });

    // mostra oque irá retornar, que é o token
    // console.log(result);

    expect(result).toHaveProperty("token");
  });
```

Verificação se n existe o email e a senha errado.

```
it("should not be able to authentocate an none existent user", () => {
  expect(async () => {
    await authenticateUserUseCase.execute({
      email: "false@email.com",
      password: "1234",
    });
  }).rejects.toBeInstanceOf(AppError);
});

it("should not be able to authenticate with incorrect password", () => {
  expect(async () => {
    const user: ICreateUserDTO = {
      driver_license: "99999",
      email: "user@user.com",
      password: "22123",
      name: "User teste error",
    }

    await createUserUserCase.execute(user);

    await authenticateUserUseCase.execute({
      email: user.email,
      password: "incorrectpassword",
    });
  }).rejects.toBeInstanceOf(AppError);
});
```

Automatizando os imports no tsconfig procure por:

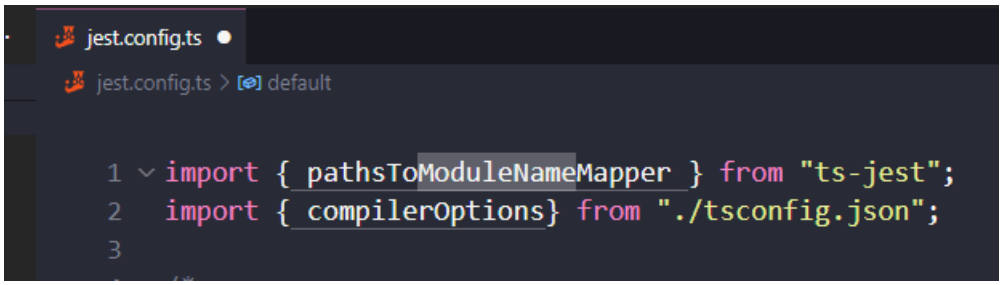
```
/* Modules */
"module": "commonjs",
// "rootDir": "./",
// "moduleResolution": "node",
"baseUrl": "./src",
"paths": {
  "@modules/*": ["modules/*"],
  "@config/*": ["config/*"],
  "@shared/*": ["shared/*"],
  "@errors/*": ["errors/*"],
},
// "rootDirs": [],
```

Baixar para traduzir as importações [yarn add tsconfig-paths -D]

E no package.json colocar -r tsconfig-paths/register

```
▷ Debug
"scripts": {
  "dev": "ts-node-dev -r tsconfig-paths/register --inspect --
  "typeorm": "ts-node-dev -r tsconfig-paths/register ./node_m
  "test": "jest"
```

Consigurando os teste vai no jestconfig:




```
moduleNameMapper: pathsToModuleNameMapper(compilerOptions.paths, {
  prefix: "<rootDir>/src/",
}),
```

RF => Requesitos funcionais

RNF => Requesitos não funcionais

RN => Regra de negócio

Chave estrangeira: fecha as columas

```
  name: "cars",
  columns: [...
  ],
  foreignKeys: [
    {
      name: "FKCategoryCar",
      referencedTableName: "categories",
      referencedColumnNames: ["id"],
      columnNames: ["category_id"],
      onDelete: "SET NULL",
      onUpdate: "SET NULL"
    }
  ]
})
```

No entitites em car.

```typescript
@Column()
brand: string;

@ManyToOne(() => Category)
@JoinColumn({ name: "category_id"})
category: Category;

@Column()
category_id: string;
```

Criando usuário adm

```typescript
import { hash } from "bcryptjs";
import { v4 as uuidV4} from "uuid";

import createConnection from "shared/infra/typeorm/index";

async function create() {
  const connection = await createConnection("localhost");

  const id = uuidV4();
  const password = await hash("admin", 8);

  await connection.query(
    `INSERT INTO USERS(id, name, email, password, "isAdmin", created_at, driver_license)
    values('${id}', 'admin', 'admin@admin.com.br', '${password}', 'true', 'now()', 'XXXXXX')
    `
  );

  await connection.close();
}

create().then(() => console.log("User admin created!"));
```

```
"seed:admin": "ts-node-dev -r tsconfig-paths/register src/shared/infra/typeorm/seed/admin.ts"
```

Middle de admin

```typescript
import { NextFunction, Request, Response } from "express";

import { AppError } from "@shared/errors/AppError";
import { UsersRepository } from "@modules/accounts/infra/typeorm/repositories/UsersRepository";

export async function ensureAdmin(request: Request, response: Response, next: NextFunction) {
  const { id } = request.user;

  const usersRepository = new UsersRepository();
  const user = await usersRepository.findById(id);

  if (!user.isAdmin) {
    throw new AppError("User isn't admin!");
  }

  return next();
}
```

Criando tabela com chave estrangeira

```typescript
export class CreateSpecificationsCars1670495130974 implements MigrationInterface {

  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.createTable(
      new Table({
        name: "specifications_cars",
        columns: [
          {
            name: "card_id",
            type: "uuid"
          },
          {
            name: "specification_id",
            type: "uuid"
          },
          {
            name: "created_at",
            type: "timestamp",
            default: "now()"
          }
        ]
      })
    );

    await queryRunner.createForeignKey(
      "specifications_cars",
      new TableForeignKey({
        name: "FKSpecificationCar",
        referencedTableName: "specifications",
        referencedColumnNames: ["id"],
        columnNames: ["specification_id"],
        onDelete: "SET NULL",
        onUpdate: "SET NULL"
      })
    );
```

```typescript
  public async down(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.dropForeignKey("specifications_cars", "FKSpecificationCar");

    await queryRunner.dropForeignKey("specifications_cars", "FKCarSpecification");

    await queryRunner.dropTable("specifications_cars")
  }
```

Quando tem tabela de relacionamento é many to many.

Caso der esse error, Cannot read properties of undefined (reading 'findByIds'), oque vem antes do metodo vem nulo.

```typescript
beforeEach(() => {
  carsRepositoryInMemory = new CarsRepositoryInMemory()
  specificationRepositoryInMemory = new SpecificationsRepositoryInMemory();
```

Trabalhar com data [yarn add dayjs]

```
> modules > rentals > usecase > createRental > CreateRentalsUse
1  import dayjs from "dayjs";
2  import utc from "dayjs/plugin/utc";
3
```

```
8  dayjs.extend(utc);
9
```

```ts
import dayjs from "dayjs";
import utc from "dayjs/plugin/utc";

import { IDateProvider } from "../IDateProvider";

dayjs.extend(utc);


class DayjsDateProvider implements IDateProvider {

  compareInHours(start_date: Date, end_date: Date): number {
    const end_date_utc = this.convertToUtc(end_date);
    const start_date_utc = this.convertToUtc(start_date);

    return dayjs(end_date_utc).diff(start_date_utc, "hours"); // converted a comparação e
  }

  convertToUtc(date: Date): string {
    return dayjs(date).utc().local().format(); // gera data baseada utc
  }

  dateNow() {
    return dayjs().toDate();
  }
}

export { DayjsDateProvider }
```

```ts
interface IDateProvider {
  compareInHours(start_date: Date, end_date: Date): number;
  convertToUtc(date: Date): string;
  dateNow(): Date;
}

export { IDateProvider }
```

```ts
// - O aluguel deve ter duração mínima de 24 horas.
const dateNow = this.dateProvider.dateNow();

const compare = this.dateProvider.compareInHours(dateNow, expected_return_date);

if (compare < minimumHour) {
  throw new AppError("Invalid return time!");
}
```

SUPER TEST yarn add supertest e o @types

```ts
"scripts": {
  "dev": "ts-node-dev -r tsconfi
  infra/http/server.ts",
  "typeorm": "ts-node-dev -r tsc
  "test": "NODE_ENV=test jest",
```

```
src > shared > infra > typeorm > index.ts > default
15  export default async(host = "database"): Promise<Connection> => {
16    const defaultOptions = await getConnectionOptions();
17
18    return createConnection(
19      Object.assign(defaultOptions, {
20        host: process.env.NODE_ENV === "test" ? "localhost" : host,
21        database: process.env.NODE_ENV === "test" ? "rentx_test" : defaultOptions.database,
22      })
23    );
24  }
```

Criar um database de teste

```
create database rentx_test
```

```
modules > cars > useCases > createCategory > CreateCategoryController.spec.ts > describe("Create Category Controller") callback
import request from "supertest";
import { Connection } from "typeorm";
import { v4 as uuidV4 } from "uuid";
import { hash } from "bcryptjs";

import { app } from "@shared/infra/http/app";

import createConnection from "@shared/infra/typeorm";

let connection: Connection;

describe("Create Category Controller", () => {

  beforeAll(async () => {
    connection = await createConnection();
    await connection.runMigrations();

    const id = uuidV4();
    const password = await hash("admin", 8);

    await connection.query(
      `INSERT INTO USERS(id, name, email, password, "isAdmin", created_at, driver_license)
      values('${id}', 'admin', 'admin@admin.com.br', '${password}', 'true', 'now()', 'XXXXXX')
      `
    );
  });
```

```
  afterAll(async () => {
    await connection.dropDatabase();
    await connection.close();
  })

  it("should be able to create a new category", async () => {
    const responseToken = await request(app).post("/sessions")
    .send({
      email: "admin@admin.com.br",
      password: "admin"
    });

    const { token } = responseToken.body;

    const response = await request(app)
    .post("/categories")
    .send({
      name: "Category supertest",
      description: "supertest category"
    }).set({
      Authorization: `Bearer ${token}`,
    });

    expect(response.status).toBe(201);
  })
});
```

```
    it("should be able to list all categories", async () => {
      const responseToken = await request(app).post("/sessions")
      .send({
        email: "admin@admin.com.br",
        password: "admin"
      });

      const { token } = responseToken.body;

      await request(app)
      .post("/categories")
      .send({
        name: "Category supertest",
        description: "supertest category"
      }).set({
        Authorization: `Bearer ${token}`,
      });

      const response = await request(app).get("/categories");

      console.log(response.body);

      expect(response.status).toBe(200);
      expect(response.body.length).toBe(1);
      expect(response.body[0]).toHaveProperty("id");
      expect(response.body[0].name).toEqual("Category supertest");
    });
  });
```