POO - 4

1. O que é o Single Responsability (Responsabilidade única)?

```
O princípio de responsabilidade única declara que um contexto deve ser responsável apenas por uma
 única responsabilidade.
```

2. O que é o Open closed (Fechado para modificação e aberto para extensão)?

```
A ideia central do princípio acima é que devemos ser capazes de adicionar
novas funcionalidades sem alterar o código existente.
```

3. O que é o Dependency Inversion Principle (Inversão de dependência)?

```
Não depender de classes concretas e sim de interfaces
```

Diagrama de Classe

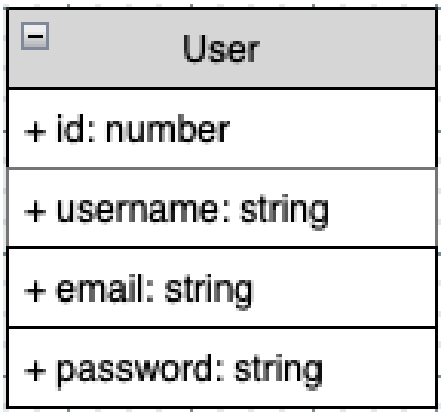
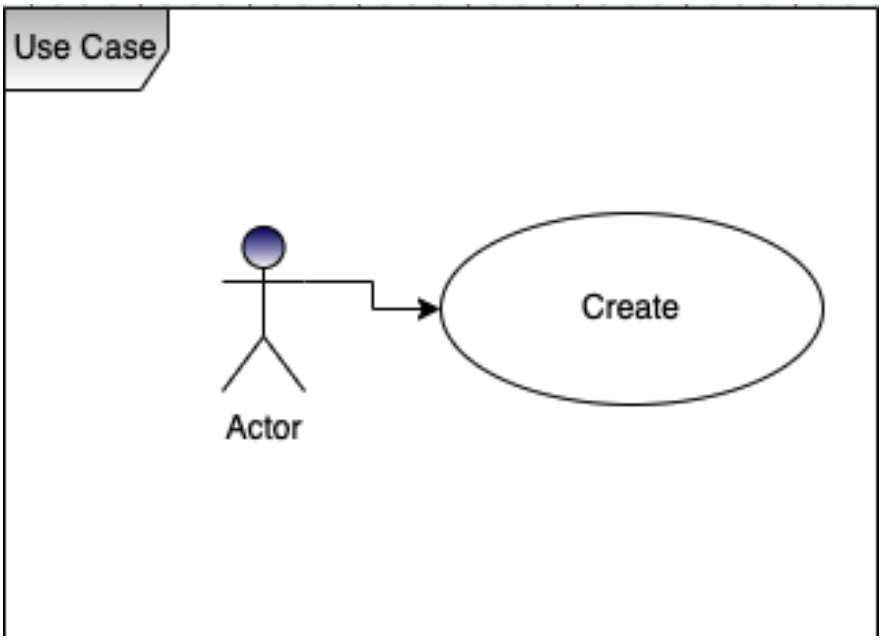


Diagrama de Casos de uso



Refatoração com SOLID

[X] Vamos separar! Muitos elementos! Muitas responsabilidades!

[X] TEM CONEXÃO! TEM QUERY! TEM ENTIDADE! OHHH MYYYY GOOOODDD!

```
import { IUser } from "../interfaces/IUser";
import { Pool, ResultSetHeader } from "mysql2/promise";

export default class UserModel {
    public connection: Pool;

    constructor(connection: Pool) {
        this.connection = connection;
    }

    public async create(): Promise<IUser[]> {
        const query = 'INSERT INTO express.users'
        const [users] = await this.connection
            .execute(query)

        return users as IUser[];
    }
}
```

[ ] Isolar nossa entidade (classe) do banco de dados

- Entitites
- Trazendo a ideia da Responsabilidade única

- /models/IUser.ts
- /models/User.ts

```
export interface IUser {
  id?: number;
  username: string;
  email: string;
  password: string;
}
```

```
import { IUser } from './IUser';

export default class User implements IUser {
  public readonly id: number
  public username: string
  public email: string
  public password: string

  constructor() {
    this.id = 0
    this.username = ''
    this.email = ''
    this.password = ''
  }

}
```

[ ] Isolar nossa connection

- Fechada para a modificação e aberta para a extensão
- /utils/AbstractConnection.ts
- /utils/MySQLConnection.ts
- .env

```
export default abstract class AbstractConnection<T> {
    abstract connect(): T
}
```

```
import mysql, { Pool } from 'mysql2/promise';
import AbstractConnection from './AbstractConnection';

import * as dotenv from 'dotenv'
dotenv.config({ path: __dirname+'/.env' })

export default class MySQLConnection extends AbstractConnection<Pool>{
```

```
    private static connection: Pool

    public connect(): Pool {
        const connection = MySQLConnection.connection = mysql.createPool({
            host: process.env.HOST,
            user: 'root',
            password: process.env.PASSWORD,
                database: process.env.DATABASE,
        })
    return connection
    }
}
```

```
HOST=localhost
USERNAME=root
PASSWORD=''
DATABASE=solid_example
```

[ ] Isolar nosso Repository

- Repository
- /models/repository/UserRepository.ts

```
import { ResultSetHeader} from "mysql2/promise";
import MySQLConnection from "../../utils/MySQLConnection";
import { IUser } from "../entitites/IUser";

export default class UserRepository {
    private persistence: MySQLConnection

    constructor() {
        this.persistence = new MySQLConnection()
    }

    public create = async (user: IUser): Promise<IUser> => {
        const { username, email, password } = user;

        const query = 'INSERT INTO solid_example.users (username, email, password) VALUES (?, ?, ?)';
        const values = [username, email, password];

        const [result] = await this.persistence.connect().execute<ResultSetHeader>(query, values);
        const { insertId: id } = result;

        const newUser: IUser = { id, username, email, password };
        return newUser;
    }
}
```

[ ] Adicionar o nosso Service/Use Case

- /services/UserService.ts

```
import { IUser } from "../models/entitites/IUser"
import User from "../models/entitites/User"
import UserRepository from "../models/repository/UserRepository"
import UserValidation from "../validations/UserValidation"

export default class UserService {

    constructor(private userRepository: UserRepository =
        new UserRepository()) {
    }

    public create = async (user: User): Promise<IUser> => {
        if(!UserValidation.validateUser(user.email, user.password))
            throw new Error("User or Password are invalid")
        const result = await this.userRepository.create(user)
        return result
    }
}
```

[ ] Adicionar validações no Service/Use Case

- /validations/UserValidation.ts

```
export default class UserValidation {
    private static validateEmail(email: string): boolean {
        const emailRegex = /\S+@\S+\.\S+/;
        return emailRegex.test(email);
    }

    private static validatePassword(password: string): boolean {
        const passwordRegex = /^\d+$/;
        return passwordRegex.test(password);
    }

    public static validateUser(email: string, password: string): boolean {
        return (
            this.validateEmail(email)
            && this.validatePassword(password)
        );
    }
}
```

[ ] Controller

```
import UserService from "../services/UserService";
import { NextFunction, Request, Response } from 'express';
import User from "../models/entitites/User";

export class UserController {

    constructor(private userService: UserService
        = new UserService()) {}
```

```
    public createUser = async(req: Request, res: Response, next: NextFunction): Promise<void> => {
        try {
            const result = await this.userService.create(req.body as User);
            res.status(200).json({
                message: result,
            });
        } catch (error) {
            next(error)
        }

    }
}
```

[ ] routes

```
import { Router } from "express";
import { UserController } from "../controllers/UserController";
import UserRepository from "../models/repository/UserRepository";
import UserService from "../services/UserService";

const router = Router()
const userRepository = new UserRepository()
const userService = new UserService(userRepository)
const controller = new UserController(userService)

router.post('/', controller.createUser)

export default router
```

☐ /middlewares/Error.ts

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

class ErrorHandler {

    public static handle(error: Error, _req: Request, res: Response, next: NextFunction) {
        res.status(500).json({ message: error.message });
        next()
    }
}

export { ErrorHandler }
```

[ ] app.ts

```
import express from "express";
import router from "./routes/userRoutes";
import { ErrorHandler } from "./middlewares/Error";
```

```
const app = express()

app.use(express.json())
app.use('/users', router)
app.use(ErrorHandler.handle)

export default app
```

[ ] server.ts

```
import app from "./app";
import MySQLConnection from "./utils/MySQLConnection";

const database = new MySQLConnection()
database.connect().getConnection().then( _result => {
    app.listen(6060)
    console.log("Database working..")
}).catch( error => {
    console.log(error)
})
```

[ ] package.json

```
"start": "ts-node-dev --respawn --transpile-only ./src/server.ts"
```