Interface gráfica do usuário, Texto

Descrição gerada automaticamente

>> Em src/controllers/CategoryController.ts tenho:

import { Request, Response } from "express";

import Category from "../models/Category";

class CategoryController {

public async create(req: Request, res: Response): Promise<void> {

const { name } = req.body;

try {

const response = await Category.create({ name });

res.send(response);

} catch (e: any) {

if (e.code === 11000) {

res.send({ message: `O nome ${name} já está cadastrado` });

} else if (e.errors?.name) {

res.send({ message: e.errors.name.message });

} else {

res.send({ message: e.message });

}

}

}

public async list(\_: Request, res: Response): Promise<void> {

res.send(await Category.find({}, {}, { sort: { name: 1 } }));

}

public async delete(req: Request, res: Response): Promise<void> {

const { id } = req.body;

const response = await Category.findByIdAndDelete(id);

if (response) {

res.json(response);

} else {

res.json({ message: "Registro inexistente" });

}

}

public async update(req: Request, res: Response): Promise<void> {

const { id, name } = req.body;

try {

const response = await Category.findByIdAndUpdate(

id,

{ name },

{

new: true,

runValidators: true,

}

);

if (response) {

res.json(response);

} else {

res.json({ message: "Registro inexistente" });

}

} catch (e: any) {

if (e.code === 11000) {

res.send({ message: `O nome ${name} já está em uso` });

} else if (e.errors?.name) {

res.send({ message: e.errors.name.message });

} else {

res.send({ message: e });

}

}

}

}

export default new CategoryController();

>> Em src/controllers/ProductController.ts tenho:

import { Request, Response } from "express";

import Product from "../models/Product";

class ProductController {

public async create(req: Request, res: Response): Promise<void> {

const { name, category } = req.body;

try {

const response = await Product.create({ name, category });

res.send(response);

} catch (e: any) {

if (e.code === 11000) {

res.send({ message: `O nome ${name} já está cadastrado` });

} else if (e.errors?.name) {

res.send({ message: e.errors.name.message });

} else if (e.errors?.category) {

res.send({ message: e.errors.category.message });

} else {

res.send({ message: e.message });

}

}

}

public async list(\_: Request, res: Response): Promise<void> {

try {

const products = await Product.find({})

.select("name category")

.populate("category", "name")

.sort({ name: 1 })

.exec(); // Ensure the query is executed

const response = products.map((product) => ({

id: product.\_id,

name: product.name,

categoria: (product.category as any).name, // TypeScript workaround

}));

res.send(response);

} catch (e: any) {

res.send({ message: e.message });

}

}

public async delete(req: Request, res: Response): Promise<void> {

const { id } = req.body;

const response = await Product.findByIdAndDelete(id);

if (response) {

res.json(response);

} else {

res.json({ message: "Registro inexistente" });

}

}

public async update(req: Request, res: Response): Promise<void> {

const { id, name, category } = req.body;

try {

const response = await Product.findByIdAndUpdate(

id,

{ name, category },

{

new: true,

runValidators: true,

}

);

if (response) {

res.json(response);

} else {

res.json({ message: "Registro inexistente" });

}

} catch (e: any) {

if (e.code === 11000) {

res.send({ message: `O nome ${name} já está em uso` });

} else if (e.errors?.name) {

res.send({ message: e.errors.name.message });

} else if (e.errors?.category) {

res.send({ message: e.errors.category.message });

} else {

res.send({ message: e });

}

}

}

}

export default new ProductController();

>> Em src/onctrollers/SpentController.ts tenho:

import { Request, Response } from "express";

import Spent from "../models/Spent";

*class* SpentController {

  public async create(*req*: Request, *res*: Response): *Promise*<*void*> {

*const* { user, product, datetime, value } = *req*.body;

    try {

*const* response = await Spent.create({ user, product, datetime, value });

*res*.send(response);

    } catch (e: *any*) {

*res*.send({ message: e.message });

    }

  }

  public async list(*\_*: Request, *res*: Response): *Promise*<*void*> {

    try {

*const* spents = await Spent.find({})

        .select("product user datetime value")

        .populate("product", "name")

        .populate("user", "mail")

        .sort({ datetime: -1 })

        .exec();

*const* response = spents.map((*spent*) *=>* ({

        id: *spent*.\_id,

        name: (*spent*.product as *any*).name,

        userMail: (*spent*.user as *any*).mail,

        datetime: *spent*.datetime,

        value: *spent*.value,

      }));

*res*.send(response);

    } catch (e: *any*) {

*res*.send({ message: e.message });

    }

  }

  public async delete(*req*: Request, *res*: Response): *Promise*<*void*> {

*const* { id } = *req*.body;

    try {

*const* response = await Spent.findByIdAndDelete(id);

      if (response) {

*res*.json(response);

      } else {

*res*.json({ message: "Registro inexistente" });

      }

    } catch (e: *any*) {

*res*.send({ message: e.message });

    }

  }

  public async update(*req*: Request, *res*: Response): *Promise*<*void*> {

*const* { id, user, product, datetime, value } = *req*.body;

    try {

*const* response = await Spent.findByIdAndUpdate(

        id,

        { user, product, datetime, value },

        {

          new: true,

          runValidators: true,

        }

      );

      if (response) {

*res*.json(response);

      } else {

*res*.json({ message: "Registro inexistente" });

      }

    } catch (e: *any*) {

*res*.send({ message: e.message });

    }

  }

}

export default new SpentController();

>> Em src/controlers/UserController.ts tenho:

import { Request, Response } from "express";

import User from "../models/User";

import { tokenize } from "../middleware";

*class* UserController {

  public async login(*req*: Request, *res*: Response): *Promise*<*void*> {

*const* { mail, password } = *req*.body;

    if (!mail || !password) {

*res*.status(401).json({ erro: "Forneça o e-mail e senha" });

    } else {

      try {

*const* user = await User.findOne({ mail, password });

        if (user) {

*res*.json({ ...user.toObject(), token: tokenize(user.toObject()) });

        } else {

*res*.json({ erro: "Dados de login não conferem" });

        }

      } catch (e: *any*) {

*res*.status(500).json({ erro: e.message });

      }

    }

  }

  public async create(*req*: Request, *res*: Response): *Promise*<*void*> {

*const* { mail, password, profile } = *req*.body;

    try {

*const* newUser = new User({ mail, password, profile });

*const* response = await newUser.save();

*const* userObject = response.toObject();

*res*.json({ ...userObject, token: tokenize(userObject) });

    } catch (e: *any*) {

      if (e.code === 11000) {

*res*.json({ erro: `O e-mail ${mail} já está cadastrado` });

      } else {

*res*.json({ erro: e.message });

      }

    }

  }

  public async list(*\_*: Request, *res*: Response): *Promise*<*void*> {

    try {

*const* users = await User.find({}, {}, { sort: { mail: 1 } });

*res*.json(users);

    } catch (e: *any*) {

*res*.status(500).json({ erro: e.message });

    }

  }

  public async delete(*req*: Request, *res*: Response): *Promise*<*void*> {

*const* { id } = *req*.body;

    try {

*const* response = await User.findByIdAndDelete(id).exec();

      if (response) {

*res*.json(response);

      } else {

*res*.json({ erro: `Usuário não localizado` });

      }

    } catch (e: *any*) {

*res*.status(500).json({ erro: e.message });

    }

  }

  public async updateMail(*req*: Request, *res*: Response): *Promise*<*void*> {

*const* { mail } = *req*.body;

*const* { id } = *req*.body;

    try {

*const* response = await User.findByIdAndUpdate(

        id,

        { mail },

        { new: true, runValidators: true, context: "query" }

      ).exec();

      if (response) {

*res*.json(response);

      } else {

*res*.json({ erro: "Usuário não localizado" });

      }

    } catch (*e*: *any*) {

      if (e.code === 11000) {

        res.json({ erro: `O e-mail ${mail} já está em uso` });

      } else {

        res.json({ erro: e.message });

      }

    }

  }

  public async updatePassword(*req*: Request, *res*: Response): Promise<*void*> {

*const* { password } = req.body;

*const* { id } = req.body;

    try {

*const* response = await User.findByIdAndUpdate(

        id,

        { password },

        { new: true, runValidators: true, context: "query" }

      ).exec();

      if (response) {

        res.json(response);

      } else {

        res.json({ erro: "Usuário não localizado" });

      }

    } catch (*e*: *any*) {

      res.json({ erro: e.message });

    }

  }

  public async updateProfile(*req*: Request, *res*: Response): Promise<*void*> {

*const* { id, profile } = req.body;

    if (profile === "adm" || profile === "user") {

      try {

*const* response = await User.findByIdAndUpdate(

          id,

          { profile },

          { new: true, runValidators: true, context: "query" }

        ).exec();

        if (response) {

          res.json(response);

        } else {

          res.json({ erro: "Usuário não localizado" });

        }

      } catch (*e*: *any*) {

        res.json({ erro: e.message });

      }

    } else {

      res.json({ erro: `Perfil inexistente` });

    }

  }

}

export default new UserController();

>> Em src/database/connection.ts tenho:

import mongoose from "mongoose";

import dotenv from "dotenv";

dotenv.config();

// A URI indica o IP, a porta e BD a ser conectado

*const* uri: *string* = process.env.DB\_URI || "";

// Salva o objeto mongoose em uma variável

*const* db = mongoose;

export *function* connect() {

  // Utiliza o método connect do Mongoose para estabelecer a conexão com

  // o MongoDB, usando a URI

  db.connect(uri, {

    serverSelectionTimeoutMS: 20000,

    maxPoolSize: 10,

  })

    .then(() *=>* console.log("Conectado ao MongoDB"))

    .catch((*e*) *=>* {

      console.error("Erro ao conectar ao MongoDB:", *e*.message);

    });

  // o sinal SIGINT é disparado ao encerrar a aplicação, geralmente, usando Crtl+C

  process.on("SIGINT", async () *=>* {

    try {

      console.log("Conexão com o MongoDB fechada");

      await mongoose.connection.close();

      process.exit(0);

    } catch (error) {

      console.error("Erro ao fechar a conexão com o MongoDB:", error);

      process.exit(1);

    }

  });

}

export async *function* disconnect() {

  console.log("Conexão com o MongoDB encerrada");

  await db.disconnect();

}

>> Em src/middleware/index.ts tenho:

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

import jwt from "jsonwebtoken";

import dotenv from "dotenv";

dotenv.config();

*const* secret = process.env.JWT\_SECRET || "";

export *const* tokenize = (*object*: *any*) *=>* jwt.sign(*object*, secret);

export *const* validadeAcess = (

*req*: Request,

*res*: Response,

*next*: NextFunction

) *=>* {

  // o token enviado pelo cliente no header da requisição

*const* authorization: *string* | *undefined* = *req*.headers.authorization;

  if (!authorization) {

*res*.status(401).send({ error: "Efetue o login para continuar" });

  } else {

    try {

      // autorização no formato Bearer token

*const* [, token] = authorization.split(" ");

*const* decoded = <*any*>jwt.verify(token, secret);

      if (decoded) {

*res*.locals = decoded;

        next();

      } else {

*res*.status(401).send({ error: "Não autorizado" });

      }

    } catch (e: *any*) {

      if (e.message == "jwt malformed") {

*res*.status(401).send({ error: "Token inválido" });

      } else {

*res*.status(401).send({ error: e.message });

      }

    }

  }

};

export *const* checkAdm = (*\_*: Request, *res*: Response, *next*: NextFunction) *=>* {

*const* { profile } = *res*.locals;

  if (profile == "adm") {

    next();

  } else {

*res*.status(401).send({ error: "Acesso negado" });

  }

};

>> Em src/models/Category.ts tenho:

import mongoose, { Schema } from "mongoose";

*const* CategorySchema = new Schema(

  {

    name: {

      type: *String*,

      trim: true,

      lowercase: true,

      unique: true,

      required: [true, "Nome é obrigatório"],

    },

  },

  {

    toJSON: {

      transform: *function* (*doc*, *ret*, *options*) {

*ret*.id = *ret*.\_id;

        delete *ret*.\_id;

        delete *ret*.\_\_v;

      },

    },

  }

);

*const* Model = mongoose.model("Category", CategorySchema, "categories");

export default Model;

>> Em src/models/Product.ts tenho:

import mongoose, { ObjectId, Schema } from "mongoose";

import Category from "./Category";

*const* ProductSchema = new Schema(

  {

    name: {

      type: *String*,

      trim: true,

      lowercase: true,

      unique: true,

      required: [true, "Nome é obrigatório"],

    },

    category: {

      type: Schema.Types.ObjectId,

      ref: Category,

      required: [true, "Categoria é obrigatória"],

      validate: {

        validator: async *function* (*\_id*: ObjectId) {

*const* document = await mongoose.models.Category.findById(*\_id*);

          return !!document;

        },

        message: "Categoria inexistente",

      },

    },

  },

  {

    toJSON: {

      transform: *function* (*doc*, *ret*, *options*) {

*ret*.id = *ret*.\_id;

        delete *ret*.\_id;

        delete *ret*.\_\_v;

      },

    },

  }

);

*const* Product = mongoose.model("Product", ProductSchema, "products");

export default Product;

>> Em src/models/Spent.ts tenho:

import mongoose, { ObjectId, Schema } from "mongoose";

import User from "./User";

import Product from "./Product";

*const* SpentSchema = new Schema(

  {

    user: {

      type: Schema.Types.ObjectId,

      ref: User,

      required: [true, "Usuário é obrigatório"],

      validate: {

        validator: async *function* (*\_id*: ObjectId) {

*const* document = await mongoose.models.User.findById(*\_id*);

          return !!document;

        },

        message: "Usuário inexistente",

      },

    },

    product: {

      type: Schema.Types.ObjectId,

      ref: Product,

      required: [true, "Produto é obrigatório"],

      validate: {

        validator: async *function* (*\_id*: ObjectId) {

*const* document = await mongoose.models.Product.findById(*\_id*);

          return !!document;

        },

        message: "Produto inexistente",

      },

    },

    datetime: {

      type: *Date*,

      default: *Date*.now,

    },

    value: {

      type: *Number*,

      required: [true, "Valor é obrigatório"],

      min: [0, "Valor deve ser maior ou igual a 0"],

    },

  },

  {

    toJSON: {

      transform: *function* (*doc*, *ret*, *options*) {

*ret*.id = *ret*.\_id;

        delete *ret*.\_id;

        delete *ret*.\_\_v;

      },

    },

  }

);

*const* Spent = mongoose.model("Spent", SpentSchema, "spents");

export default Spent;

>> Em src/models/User.ts tenho:

import mongoose, { Schema } from "mongoose";

*const* UserSchema = new Schema(

  {

    mail: {

      type: *String*,

      trim: true,

      lowercase: true,

      unique: true,

      required: [true, "E-mail é obrigatório"],

      match: [

        /^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/,

        "E-mail inválido",

      ],

    },

    password: {

      type: *String*,

      trim: true,

      select: false,

      required: [true, "Senha é obrigatória"],

      minlength: [6, "Senha deve ter no mínimo 6 caracteres"],

      maxlength: [10, "Senha deve ter no máximo 10 caracteres"],

    },

    profile: {

      type: *String*,

      enum: ["user", "adm"],

      default: "user",

    },

  },

  {

    toJSON: {

      transform: *function* (*doc*, *ret*, *options*) {

*ret*.id = *ret*.\_id;

        delete *ret*.\_id;

        delete *ret*.\_\_v;

        return *ret*;

      },

    },

  }

);

*const* User = mongoose.model("User", UserSchema, "users");

export default User;

>> Em src/routes/category.ts tenho:

import { Router } from "express";

import controller from "../controllers/CategoryController";

*const* router = Router();

router.post("/", controller.create);

router.get("/", controller.list);

router.delete("/", controller.delete);

router.put("/", controller.update);

export default router;

>> Em src/routes/product.ts tenho:

import { Router } from "express";

import ProductController from "../controllers/ProductController";

*const* router = Router();

router.post("/", ProductController.create);

router.get("/", ProductController.list);

router.delete("/", ProductController.delete);

router.put("/", ProductController.update);

export default router;

>> Em src/routes/spents.ts tenho:

import { Router } from "express";

import SpentController from "../controllers/SpentController";

*const* router = Router();

router.post("/", SpentController.create);

router.get("/", SpentController.list);

router.delete("/", SpentController.delete);

router.put("/", SpentController.update);

export default router;

>> Em src/routes/user.ts tenho:

import { Router, Request, Response } from "express";

import controller from "../controllers/UserController";

import { validadeAcess, checkAdm } from "../middleware";

*const* routes = Router();

routes.get("/", validadeAcess, checkAdm, controller.list);

routes.post("/", controller.create);

routes.delete("/", validadeAcess, controller.delete);

routes.put("/mail", validadeAcess, controller.updateMail);

routes.put("/senha", validadeAcess, controller.updatePassword);

routes.put("/perfil", validadeAcess, checkAdm, controller.updateProfile);

//aceita qualquer método HTTP ou URL

routes.use((*\_*: Request, *res*: Response) *=>*

*res*.json({ error: "Operação desconhecida com o usuário" })

);

export default routes;

>> Em src/routes/index.ts tenho:

import { Router, Request, Response } from "express";

import user from "./user";

import product from "./product";

import userController from "../controllers/UserController";

import spent from "./spent";

import category from "./category";

import { validadeAcess } from "../middleware";

*const* routes = Router();

routes.post("/login", userController.login);

routes.use("/usuario", user);

routes.use("/categoria", validadeAcess, category);

routes.use("/produto", validadeAcess, product);

routes.use("/gasto", validadeAcess, spent);

//aceita qualquer método HTTP ou URL

routes.use((*\_*: Request, *res*: Response) *=>*

*res*.json({ error: "Requisição desconhecida" })

);

export default routes;

>> Em src/index.ts tenho:

import express from "express";

import cors from "cors";

import dotenv from "dotenv";

import routes from "./routes";

import { connect } from "./database/connection";

dotenv.config();

// será usado 3000 se a variável de ambiente não tiver sido definida

*const* PORT = process.env.PORT || 3000;

*const* app = express(); // cria o servidor e coloca na variável app

// suportar parâmetros JSON no body da requisição

app.use(express.json());

// configura o servidor para receber requisições de qualquer domínio

app.use(cors());

// conecta ao MongoDB no início da aplicação

connect();

// inicializa o servidor na porta especificada

app.listen(PORT, () *=>* {

  console.log(`Rodando na porta ${PORT}...`);

});

// define a rota para o pacote /routes

app.use(routes);

>> Fora de src tenho a variável de ambiente .env:

PORT = 3010

JWT\_SECRET = @tokenJWT

DB\_URI = mongodb://127.0.0.1:27017/bdatividade5

Me explique o que este programa faz.