

# Guia para Criar o Backend da Pizzaria

Este guia passo a passo explica como recriar o backend da aplicação de pizzaria desenvolvido em Node.js com TypeScript, Express, Prisma e PostgreSQL.

## Pré-requisitos

Antes de começar, certifique-se de ter instalado:

- **Node.js** (versão 18 ou superior)
- **PostgreSQL** (banco de dados)
- **npm** ou **yarn** (gerenciador de pacotes)
- Uma conta no **Cloudinary** (para upload de imagens)

## 1. Configuração Inicial do Projeto

### 1.1 Criar o diretório do projeto

```
mkdir backend-pizzaria  
cd backend-pizzaria
```

### 1.2 Inicializar o projeto Node.js

```
npm init -y
```

### 1.3 Instalar dependências

```
npm install express @prisma/client @prisma/adapters pg bcryptjs jsonwebtoken zod cors dotenv multer  
cloudinary tsx  
npm install -D typescript @types/express @types/cors @types/jsonwebtoken @types/multer @types/node  
@types/pg prisma
```

### 1.4 Configurar TypeScript

Crie o arquivo `tsconfig.json`:

```

{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "lib": ["ES2020"],
    "moduleResolution": "node",
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "strictFunctionTypes": true,
    "strictBindCallApply": true,
    "strictPropertyInitialization": true,
    "noImplicitThis": true,
    "alwaysStrict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": true,
    "noUncheckedIndexedAccess": true,
    "noImplicitOverride": true,
    "allowUnusedLabels": false,
    "allowUnreachableCode": false,
    "exactOptionalPropertyTypes": true,
    "noImplicitReturns": true,
    "noPropertyAccessFromIndexSignature": false,
    "noUncheckedIndexedAccess": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "forceConsistentCasingInFileNames": true,
    "resolveJsonModule": true,
    "isolatedModules": true,
    "skipLibCheck": true,
    "declaration": true,
    "declarationMap": true,
    "sourceMap": true,
    "removeComments": false,
    "noEmitOnError": true,
    "incremental": true,
    "tsBuildInfoFile": ".tsbuildinfo",
    "experimentalDecorators": false,
    "emitDecoratorMetadata": false,
    "allowJs": false,
    "checkJs": false,
    "maxNodeModuleJsDepth": 1,
    "disableSizeLimit": false,
    "disableSourceOfProjectReferenceRedirect": false,
    "disableSolutionSearching": false,
    "disableReferencedProjectLoad": false
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "dist", "**/*.test.ts"]
}

```

## 1.5 Criar arquivo de configuração do Prisma

Crie `prisma.config.ts`:

```
import "dotenv/config";
import { defineConfig } from "prisma/config";

export default defineConfig({
  schema: "prisma/schema.prisma",
  migrations: {
    path: "prisma/migrations",
  },
  datasource: {
    url: process.env["DATABASE_URL"],
  },
});
```

## 1.6 Configurar variáveis de ambiente

Crie o arquivo .env:

```
DATABASE_URL="postgresql://username:password@localhost:5432/pizzaria_db"
JWT_SECRET="your_jwt_secret_here"
CLOUDINARY_NAME="your_cloudinary_name"
CLOUDINARY_API_KEY="your_cloudinary_api_key"
CLOUDINARY_API_SECRET="your_cloudinary_api_secret"
PORT=3333
```

## 2. Configuração do Banco de Dados com Prisma

### 2.1 Inicializar Prisma

```
npx prisma init
```

### 2.2 Criar o schema do banco

Substitua o conteúdo de prisma/schema.prisma:

```
generator client {
  provider = "prisma-client"
  output   = "../src/generated/prisma"
}

datasource db {
  provider = "postgresql"
}

enum Role {
  STAFF
  ADMIN
}

model User {
  id      String @id @default(uuid())
  name    String
  email   String @unique
  password String
  role    Role @default(STAFF)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  @@map("users")
}

model Category {
  id      String @id @default(uuid())
  name    String
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  products Product[]
  @@map("categories")
}

model Product {
```

```

    id String @id @default(uuid())
    name String
    description String
    price Int
    banner String
    disabled Boolean @default(false)
    category_id String
    category Category @relation(fields: [category_id], references: [id], onDelete: Cascade)
    items Item[]
    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt
    @@map("products")
}

model Order{
    id String @id @default(uuid())
    table Int
    status Boolean @default(false)
    draft Boolean @default(true)
    name String?
    items Item[]
    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt
    @@map("orders")
}

model Item{
    id String @id @default(uuid())
    amount Int
    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt

    order_id String
    order Order @relation(fields: [order_id], references: [id], onDelete: Cascade)

    product_id String
    product Product @relation(fields: [product_id], references: [id], onDelete: Cascade)

    @@map("items")
}

```

## 2.3 Gerar cliente Prisma e criar migração

```

npx prisma generate
npx prisma migrate dev --name first_migration

```

## 3. Estrutura de Pastas

Crie a estrutura de pastas conforme mostrado:

```

backend/
├── prisma/
│   ├── migrations/
│   └── schema.prisma
├── src/
│   ├── @types/
│   │   └── express/
│   │       └── index.d.ts
│   ├── controllers/
│   │   ├── user/
│   │   ├── category/
│   │   ├── product/
│   │   └── order/
│   ├── services/
│   │   ├── user/
│   │   ├── category/
│   │   ├── product/
│   │   └── order/
│   ├── middlewares/
│   ├── schemas/
│   ├── config/
│   ├── generated/
│   │   └── prisma/
│   ├── prisma/
│   │   └── index.ts
│   ├── routes.ts
│   └── server.ts
├── .env
├── package.json
└── tsconfig.json

```

## 4. Configurações Básicas

### 4.1 Extensão de tipos do Express

Crie `src/@types/express/index.d.ts`:

```

declare global {
  namespace Express {
    interface Request {
      user_id?: string;
    }
  }
}

```

### 4.2 Instância do Prisma Client

Crie `src/prisma/index.ts`:

```

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

const prismaClient = new PrismaClient();

export { prismaClient };

```

### 4.3 Configuração do Multer

Crie `src/config/multer.ts`:

```
import multer from 'multer';
import path from 'path';
import crypto from 'crypto';

export default {
  storage: multer.diskStorage({
    destination: path.resolve(__dirname, '..', '..', 'tmp'),
    filename: (req, file, cb) => {
      const hash = crypto.randomBytes(6).toString('hex');
      const fileName = `${hash}-${file.originalname}`;
      cb(null, fileName);
    },
  }),
};
```

## 4.4 Configuração do Cloudinary

Crie src/config/cloudinary.ts:

```
import { v2 as cloudinary } from 'cloudinary';

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

export { cloudinary };
```

## 5. Middlewares

### 5.1 Middleware de Autenticação

Crie src/middlewares/isAuthenticated.ts:

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

interface JwtPayload {
  id: string;
}

export const isAuthenticated = (req: Request, res: Response, next: NextFunction) => {
  const authHeader = req.headers.authorization;

  if (!authHeader) {
    return res.status(401).json({ message: 'Token não fornecido' });
  }

  const [, token] = authHeader.split(' ');

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET!) as JwtPayload;
    req.user_id = decoded.id;
    return next();
  } catch (error) {
    return res.status(401).json({ message: 'Token inválido' });
  }
};
```

### 5.2 Middleware de Admin

Crie src/middlewares/isAdmin.ts:

```
import { NextFunction, Request, Response } from 'express';
import { prismaClient } from '../prisma';

export const isAdmin = async (req: Request, res: Response, next: NextFunction) => {
  const userId = req.user_id;

  const user = await prismaClient.user.findUnique({
    where: { id: userId },
  });

  if (!user) {
    return res.status(404).json({ message: 'Usuário não encontrado' });
  }

  if (user.role !== 'ADMIN') {
    return res.status(403).json({ message: 'Acesso negado. Apenas administradores.' });
  }

  return next();
};
```

### 5.3 Middleware de Validação de Schema

Crie src/middlewares/validateSchema.ts:

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

export const validateSchema = (schema: ZodSchema) => {
  return (req: Request, res: Response, next: NextFunction) => {
    try {
      schema.parse(req.body);
      next();
    } catch (error: any) {
      return res.status(400).json({
        message: 'Dados inválidos',
        errors: error.errors,
      });
    }
  };
};
```

## 6. Schemas de Validação

### 6.1 Schema de Usuário

Crie src/schemas/userSchema.ts:

```
import { z } from 'zod';

export const createUserSchema = z.object({
  name: z.string().min(1, 'Nome é obrigatório'),
  email: z.string().email('Email inválido'),
  password: z.string().min(6, 'Senha deve ter pelo menos 6 caracteres'),
});

export const authUserSchema = z.object({
  email: z.string().email('Email inválido'),
  password: z.string().min(1, 'Senha é obrigatória'),
});
```

### 6.2 Schema de Categoria

Crie src/schemas/categorySchema.ts:

```
import { z } from 'zod';

export const createCategorySchema = z.object({
  name: z.string().min(1, 'Nome é obrigatório'),
});
```

### 6.3 Schema de Produto

Crie `src/schemas/productSchema.ts`:

```
import { z } from 'zod';

export const createProductSchema = z.object({
  name: z.string().min(1, 'Nome é obrigatório'),
  description: z.string().min(1, 'Descrição é obrigatória'),
  price: z.number().positive('Preço deve ser positivo'),
  category_id: z.string().min(1, 'Categoria é obrigatória'),
});
```

### 6.4 Schema de Pedido

Crie `src/schemas/orderSchema.ts`:

```
import { z } from 'zod';

export const createOrderSchema = z.object({
  table: z.number().int().positive('Mesa deve ser um número positivo'),
  name: z.string().optional(),
});

export const addItemSchema = z.object({
  order_id: z.string().min(1, 'ID do pedido é obrigatório'),
  product_id: z.string().min(1, 'ID do produto é obrigatório'),
  amount: z.number().int().positive('Quantidade deve ser positiva'),
});

export const sendOrderSchema = z.object({
  order_id: z.string().min(1, 'ID do pedido é obrigatório'),
});

export const finishOrderSchema = z.object({
  order_id: z.string().min(1, 'ID do pedido é obrigatório'),
});
```

## 7. Serviços (Services)

### 7.1 Serviço de Usuário

Crie `src/services/user/CreateUserService.ts`:

```

import { prismaClient } from '../prisma';
import bcrypt from 'bcryptjs';

interface CreateUserRequest {
  name: string;
  email: string;
  password: string;
}

export class CreateUserService {
  async execute({ name, email, password }: CreateUserRequest) {
    const userAlreadyExists = await prismaClient.user.findFirst({
      where: { email },
    });

    if (userAlreadyExists) {
      throw new Error('Email já cadastrado');
    }

    const hashedPassword = await bcrypt.hash(password, 8);

    const user = await prismaClient.user.create({
      data: {
        name,
        email,
        password: hashedPassword,
      },
      select: {
        id: true,
        name: true,
        email: true,
        role: true,
      },
    });

    return user;
  }
}

```

Crie src/services/user/AuthUserService.ts:

```

import { prismaClient } from '../..//prisma';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';

interface AuthUserRequest {
  email: string;
  password: string;
}

export class AuthUserService {
  async execute({ email, password }: AuthUserRequest) {
    const user = await prismaClient.user.findFirst({
      where: { email },
    });

    if (!user) {
      throw new Error('Email ou senha incorretos');
    }

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

    if (!passwordMatch) {
      throw new Error('Email ou senha incorretos');
    }

    const token = jwt.sign(
      { id: user.id },
      process.env.JWT_SECRET!,
      { expiresIn: '30d' }
    );

    return {
      id: user.id,
      name: user.name,
      email: user.email,
      role: user.role,
      token,
    };
  }
}

```

Crie src/services/user/DetailUserService.ts:

```

import { prismaClient } from '../..//prisma';

export class DetailUserService {
  async execute(user_id: string) {
    const user = await prismaClient.user.findFirst({
      where: { id: user_id },
      select: {
        id: true,
        name: true,
        email: true,
        role: true,
        createdAt: true,
      },
    });

    return user;
  }
}

```

## 7.2 Serviço de Categoria

Crie src/services/category/CreateCategoryService.ts:

```
import { prismaClient } from '../..prisma';

interface CreateCategoryRequest {
  name: string;
}

export class CreateCategoryService {
  async execute({ name }: CreateCategoryRequest) {
    const category = await prismaClient.category.create({
      data: { name },
      select: {
        id: true,
        name: true,
      },
    });

    return category;
  }
}
```

Crie src/services/category/ListCategoryService.ts:

```
import { prismaClient } from '../..prisma';

export class ListCategoryService {
  async execute() {
    const categories = await prismaClient.category.findMany({
      select: {
        id: true,
        name: true,
      },
    });

    return categories;
  }
}
```

### 7.3 Serviço de Produto

Crie src/services/product/CreateProductService.ts:

```

import { prismaClient } from '../..prisma';

interface CreateProductRequest {
  name: string;
  description: string;
  price: number;
  banner: string;
  category_id: string;
}

export class CreateProductService {
  async execute({ name, description, price, banner, category_id }: CreateProductRequest) {
    const product = await prismaClient.product.create({
      data: {
        name,
        description,
        price,
        banner,
        category_id,
      },
      select: {
        id: true,
        name: true,
        description: true,
        price: true,
        banner: true,
        category_id: true,
      },
    });

    return product;
  }
}

```

Crie src/services/product/ListProductService.ts:

```

import { prismaClient } from '../..prisma';

export class ListProductService {
  async execute() {
    const products = await prismaClient.product.findMany({
      where: { disabled: false },
      select: {
        id: true,
        name: true,
        description: true,
        price: true,
        banner: true,
        category: {
          select: {
            id: true,
            name: true,
          },
        },
      },
    });

    return products;
  }
}

```

Crie src/services/product/DeleteProductService.ts:

```
import { prismaClient } from '../..//prisma';

interface DeleteProductRequest {
  product_id: string;
}

export class DeleteProductService {
  async execute({ product_id }: DeleteProductRequest) {
    const product = await prismaClient.product.update({
      where: { id: product_id },
      data: { disabled: true },
    });

    return product;
  }
}
```

Crie src/services/product/ListProductByCategoryService.ts:

```
import { prismaClient } from '../..//prisma';

interface ListProductByCategoryRequest {
  category_id: string;
}

export class ListProductByCategoryService {
  async execute({ category_id }: ListProductByCategoryRequest) {
    const products = await prismaClient.product.findMany({
      where: {
        category_id,
        disabled: false,
      },
      select: {
        id: true,
        name: true,
        description: true,
        price: true,
        banner: true,
        category: {
          select: {
            id: true,
            name: true,
          },
        },
      },
    });

    return products;
  }
}
```

## 7.4 Serviço de Pedido

Crie src/services/order/CreateOrderService.ts:

```
import { prismaClient } from '../..//prisma';

interface CreateOrderRequest {
  table: number;
  name?: string;
}

export class CreateOrderService {
  async execute({ table, name }: CreateOrderRequest) {
    const order = await prismaClient.order.create({
      data: {
        table,
        name,
      },
      select: {
        id: true,
        table: true,
        status: true,
        draft: true,
        name: true,
      },
    });

    return order;
  }
}
```

Crée src/services/order/ListOrdersService.ts:

```
import { prismaClient } from '../..//prisma';

export class ListOrdersService {
  async execute() {
    const orders = await prismaClient.order.findMany({
      where: { draft: false },
      select: {
        id: true,
        table: true,
        status: true,
        draft: true,
        name: true,
        createdAt: true,
        items: {
          select: {
            id: true,
            amount: true,
            product: {
              select: {
                id: true,
                name: true,
                price: true,
              },
            },
          },
        },
      },
      orderBy: { createdAt: 'desc' },
    });

    return orders;
  }
}
```

Crée src/services/order/AddItemOrderService.ts:

```

import { prismaClient } from '../..prisma';

interface AddItemOrderRequest {
  order_id: string;
  product_id: string;
  amount: number;
}

export class AddItemOrderService {
  async execute({ order_id, product_id, amount }: AddItemOrderRequest) {
    const item = await prismaClient.item.create({
      data: {
        order_id,
        product_id,
        amount,
      },
      select: {
        id: true,
        amount: true,
        product: {
          select: {
            id: true,
            name: true,
            price: true,
          },
        },
        order: {
          select: {
            id: true,
            table: true,
            name: true,
          },
        },
      },
    });

    return item;
  }
}

```

Crie src/services/order/RemoveItemService.ts:

```

import { prismaClient } from '../..prisma';

interface RemoveItemRequest {
  item_id: string;
}

export class RemoveItemService {
  async execute({ item_id }: RemoveItemRequest) {
    const item = await prismaClient.item.delete({
      where: { id: item_id },
    });

    return item;
  }
}

```

Crie src/services/order/DetailOrderService.ts:

```

import { prismaClient } from '../..//prisma';

interface DetailOrderRequest {
  order_id: string;
}

export class DetailOrderService {
  async execute({ order_id }: DetailOrderRequest) {
    const order = await prismaClient.order.findFirst({
      where: { id: order_id },
      select: {
        id: true,
        table: true,
        status: true,
        draft: true,
        name: true,
        createdAt: true,
        items: {
          select: {
            id: true,
            amount: true,
            product: {
              select: {
                id: true,
                name: true,
                price: true,
              },
            },
          },
        },
      },
    });

    return order;
  }
}

```

Create src/services/order/SendOrderService.ts:

```

import { prismaClient } from '../..//prisma';

interface SendOrderRequest {
  order_id: string;
}

export class SendOrderService {
  async execute({ order_id }: SendOrderRequest) {
    const order = await prismaClient.order.update({
      where: { id: order_id },
      data: { draft: false },
      select: {
        id: true,
        table: true,
        status: true,
        draft: true,
        name: true,
      },
    });

    return order;
  }
}

```

Create src/services/order/FinishOrderService.ts:

```
import { prismaClient } from '../..//prisma';

interface FinishOrderRequest {
  order_id: string;
}

export class FinishOrderService {
  async execute({ order_id }: FinishOrderRequest) {
    const order = await prismaClient.order.update({
      where: { id: order_id },
      data: { status: true },
      select: {
        id: true,
        table: true,
        status: true,
        draft: true,
        name: true,
      },
    });

    return order;
  }
}
```

Crie src/services/order/DeleteOrderService.ts:

```
import { prismaClient } from '../..//prisma';

interface DeleteOrderRequest {
  order_id: string;
}

export class DeleteOrderService {
  async execute({ order_id }: DeleteOrderRequest) {
    const order = await prismaClient.order.delete({
      where: { id: order_id },
    });

    return order;
  }
}
```

## 8. Controladores (Controllers)

### 8.1 Controlador de Usuário

Crie src/controllers/user/CreateUserController.ts:

```
import { Request, Response } from 'express';
import { CreateUserService } from '../..//services/user/CreateUserService';

export class CreateUserController {
  async handle(req: Request, res: Response) {
    const { name, email, password } = req.body;

    const createUserService = new CreateUserService();

    const user = await createUserService.execute({
      name,
      email,
      password,
    });

    return res.json(user);
  }
}
```

Crie src/controllers/user/AuthUserController.ts:

```
import { Request, Response } from 'express';
import { AuthUserService } from '../../services/user/AuthUserService';

export class AuthUserController {
  async handle(req: Request, res: Response) {
    const { email, password } = req.body;

    const authUserService = new AuthUserService();

    const auth = await authUserService.execute({
      email,
      password,
    });

    return res.json(auth);
  }
}
```

Crie src/controllers/user/DetailUserController.ts:

```
import { Request, Response } from 'express';
import { DetailUserService } from '../../services/user/DetailUserService';

export class DetailUserController {
  async handle(req: Request, res: Response) {
    const user_id = req.user_id!;

    const detailUserService = new DetailUserService();

    const user = await detailUserService.execute(user_id);

    return res.json(user);
  }
}
```

## 8.2 Controlador de Categoria

Crie src/controllers/category/CreateCategoryController.ts:

```
import { Request, Response } from 'express';
import { CreateCategoryService } from '../../services/category/CreateCategoryService';

export class CreateCategoryController {
  async handle(req: Request, res: Response) {
    const { name } = req.body;

    const createCategoryService = new CreateCategoryService();

    const category = await createCategoryService.execute({
      name,
    });

    return res.json(category);
  }
}
```

Crie src/controllers/category/ListCategoryController.ts:

```
import { Request, Response } from 'express';
import { ListCategoryService } from '../../services/category/ListCategoryService';

export class ListCategoryController {
  async handle(req: Request, res: Response) {
    const listCategoryService = new ListCategoryService();

    const categories = await listCategoryService.execute();

    return res.json(categories);
  }
}
```

### 8.3 Controlador de Produto

Crie src/controllers/product/CreateProductController.ts:

```
import { Request, Response } from 'express';
import { CreateProductService } from '../../../services/product/CreateProductService';
import { cloudinary } from '../../../config/cloudinary';
import fs from 'fs';
import path from 'path';

export class CreateProductController {
  async handle(req: Request, res: Response) {
    const { name, description, price, category_id } = req.body;
    const file = req.file;

    if (!file) {
      return res.status(400).json({ message: 'Imagem é obrigatória' });
    }

    const resultFile = await cloudinary.uploader.upload(file.path, {
      folder: 'pizzaria',
    });

    fs.unlinkSync(file.path);

    const createProductService = new CreateProductService();

    const product = await createProductService.execute({
      name,
      description,
      price: Number(price),
      banner: resultFile.url,
      category_id,
    });

    return res.json(product);
  }
}
```

Crie src/controllers/product/ListProductController.ts:

```
import { Request, Response } from 'express';
import { ListProductService } from '../../../services/product/ListProductService';

export class ListProductController {
  async handle(req: Request, res: Response) {
    const listProductService = new ListProductService();

    const products = await listProductService.execute();

    return res.json(products);
  }
}
```

Crie src/controllers/product/DeleteProductController.ts:

```
import { Request, Response } from 'express';
import { DeleteProductService } from '../../../services/product/DeleteProductService';

export class DeleteProductController {
  async handle(req: Request, res: Response) {
    const product_id = req.query.product_id as string;

    const deleteProductService = new DeleteProductService();

    const product = await deleteProductService.execute({
      product_id,
    });

    return res.json(product);
  }
}
```

Crie src/controllers/product/ListProductByCategoryController.ts:

```
import { Request, Response } from 'express';
import { ListProductByCategoryService } from '../../../services/product/ListProductByCategoryService';

export class ListProductByCategoryController {
  async handle(req: Request, res: Response) {
    const category_id = req.query.category_id as string;

    const listProductByCategoryService = new ListProductByCategoryService();

    const products = await listProductByCategoryService.execute({
      category_id,
    });

    return res.json(products);
  }
}
```

## 8.4 Controlador de Pedido

Crie src/controllers/order/CreateOrderController.ts:

```
import { Request, Response } from 'express';
import { CreateOrderService } from '../../../services/order/CreateOrderService';

export class CreateOrderController {
  async handle(req: Request, res: Response) {
    const { table, name } = req.body;

    const createOrderService = new CreateOrderService();

    const order = await createOrderService.execute({
      table,
      name,
    });

    return res.json(order);
  }
}
```

Crie src/controllers/order/ListOrdersController.ts:

```
import { Request, Response } from 'express';
import { ListOrdersService } from '../../../services/order/ListOrdersService';

export class ListOrdersController {
  async handle(req: Request, res: Response) {
    const listOrdersService = new ListOrdersService();

    const orders = await listOrdersService.execute();

    return res.json(orders);
  }
}
```

Crie src/controllers/order/AddItemController.ts:

```
import { Request, Response } from 'express';
import { AddItemOrderService } from '../../../services/order/AddItemOrderService';

export class AddItemController {
  async handle(req: Request, res: Response) {
    const { order_id, product_id, amount } = req.body;

    const addItemOrderService = new AddItemOrderService();

    const item = await addItemOrderService.execute({
      order_id,
      product_id,
      amount,
    });

    return res.json(item);
  }
}
```

Crie src/controllers/order/RemoveItemController.ts:

```
import { Request, Response } from 'express';
import { RemoveItemService } from '../../../services/order/RemoveItemService';

export class RemoveItemController {
  async handle(req: Request, res: Response) {
    const item_id = req.query.item_id as string;

    const removeItemService = new RemoveItemService();

    const item = await removeItemService.execute({
      item_id,
    });

    return res.json(item);
  }
}
```

Crie src/controllers/order/DetailOrderController.ts:

```
import { Request, Response } from 'express';
import { DetailOrderService } from '../../../services/order/DetailOrderService';

export class DetailOrderController {
  async handle(req: Request, res: Response) {
    const order_id = req.query.order_id as string;

    const detailOrderService = new DetailOrderService();

    const order = await detailOrderService.execute({
      order_id,
    });

    return res.json(order);
  }
}
```

Crie src/controllers/order/SendOrderController.ts:

```
import { Request, Response } from 'express';
import { SendOrderService } from '../../../services/order/SendOrderService';

export class SendOrderController {
  async handle(req: Request, res: Response) {
    const { order_id } = req.body;

    const sendOrderService = new SendOrderService();

    const order = await sendOrderService.execute({
      order_id,
    });

    return res.json(order);
  }
}
```

Crie src/controllers/order/FinishOrderController.ts:

```
import { Request, Response } from 'express';
import { FinishOrderService } from '../../../services/order/FinishOrderService';

export class FinishOrderController {
  async handle(req: Request, res: Response) {
    const { order_id } = req.body;

    const finishOrderService = new FinishOrderService();

    const order = await finishOrderService.execute({
      order_id,
    });

    return res.json(order);
  }
}
```

Crie src/controllers/order/DeleteOrderController.ts:

```
import { Request, Response } from 'express';
import { DeleteOrderService } from '../services/order/DeleteOrderService';

export class DeleteOrderController {
  async handle(req: Request, res: Response) {
    const order_id = req.query.order_id as string;

    const deleteOrderService = new DeleteOrderService();

    const order = await deleteOrderService.execute({
      order_id,
    });

    return res.json(order);
  }
}
```

## 9. Rotas

Crie src/routes.ts:

```
import { Router } from "express";
import multer from "multer";
import uploadConfig from "../config/multer";
import { CreateUserControlller } from "../controllers/user/CreateUserController";
import { validateSchema } from "../middlewares/validateSchema";
import { createUserSchema, authUserSchema } from "../schemas/userSchema";
import { createProductSchema } from "../schemas/productSchema";
import { createCategorySchema } from "../schemas/categorySchema";
import { addItemSchema, createOrderSchema, finishOrderSchema, sendOrderSchema } from
  "../schemas/orderSchema";
import { AuthUserController } from "../controllers/user/AuthUserController";
import { DetailUserController } from "../controllers/user/DetailUserController";
import { isAuthenticated } from "../middlewares/isAuthenticated";
import { isAdmin } from "../middlewares/isAdmin";
import { CreateCategoryController } from "../controllers/category/CreateCategorytController";
import { ListCategoryController } from "../controllers/category/ListCategoryController";
import { CreateProductController } from "../controllers/product/CreateProductController";
import { ListProductController } from "../controllers/product/ListProductController";
import { DeleteProductController } from "../controllers/product/DeleteProductController";
import { ListProductByCategoryController } from "../controllers/product/ListProductByCategoryController";
import { CreateOrderController } from "../controllers/order/CreateOrderController";
import { ListOrdersController } from "../controllers/order/ListOrdersController";
import { AddItemController } from "../controllers/order/AddItemController";
import { RemoveItemController } from "../controllers/order/RemoveItemController";
import { DetailOrderController } from "../controllers/order/DetailOrderController";
import { SendOrderController } from "../controllers/order/SendOrderController";
import { FinishOrderController } from "../controllers/order/FinishOrderController";
import { DeleteOrderController } from "../controllers/order/DeleteOrderController";

const router = Router();
const upload = multer(uploadConfig);

//Rotas users
router.post("/users", validateSchema(createUserSchema),
  new CreateUserControlller().handle)
router.post("/session", validateSchema(authUserSchema),
  new AuthUserController().handle)
router.get("/me", isAuthenticated, new DetailUserController().handle)

//Rotas categories
router.post("/category", isAuthenticated, isAdmin, validateSchema(createCategorySchema),
  new CreateCategoryController().handle)
router.get("/category", isAuthenticated,
  new ListCategoryController().handle)

//Rotas products
router.post("/product", isAuthenticated, isAdmin,
  upload.single("file"), validateSchema(createProductSchema),
```

```

    new CreateProductController().handle)
router.get("/products", isAuthenticated,
    new ListProductController().handle)
router.delete("/product", isAuthenticated, isAdmin,
    new DeleteProductController().handle)
router.get("/category/product", isAuthenticated,
    new ListProductByCategoryController().handle)

//Rotas orders
router.post("/order", isAuthenticated, validateSchema(createOrderSchema),
    new CreateOrderController().handle)
router.get("/orders", isAuthenticated, new ListOrdersController().handle)
router.post("/order/add", isAuthenticated, validateSchema(addItemSchema),
    new AddItemController().handle)
router.delete("/order/remove", isAuthenticated, new RemoveItemController().handle)
router.get("/order/detail", isAuthenticated, new DetailOrderController().handle)
router.put("/order/send", isAuthenticated, validateSchema(sendOrderSchema), new
SendOrderController().handle)
router.put("/order/finish", isAuthenticated, validateSchema(finishOrderSchema), new
FinishOrderController().handle)
router.delete("/order", isAuthenticated, new DeleteOrderController().handle)

export { router }

```

## 10. Servidor Principal

Crie src/server.ts:

```

import cors from 'cors'
import "dotenv/config"
import express, { NextFunction, Request, Response } from 'express';
import { router } from "../routes"

const app = express();

app.use(express.json());
app.use(cors());
app.use(router);

app.use((error: Error, req: Request, res: Response, next: NextFunction) => {
    if (error instanceof Error) {
        return res.status(400).json({
            message: error.message
        })
    }

    return res.status(500).json({
        message: "Internal server error"
    })
})

const PORT = process.env.PORT! || 3333;

app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});

```

## 11. Executar a Aplicação

### 11.1 Criar diretório temporário para uploads

```
mkdir tmp
```

### 11.2 Executar o servidor

```
npm run dev
```

O servidor estará rodando em <http://localhost:3333>.

## 12. Testando os Endpoints

Use ferramentas como Postman ou Insomnia para testar os endpoints. Consulte o arquivo `endpoints.md` para detalhes sobre cada endpoint, parâmetros e exemplos de uso.

## Conclusão

Seguindo este guia, você terá recriado completamente o backend da aplicação de pizzaria. O projeto utiliza boas práticas de desenvolvimento, como separação de responsabilidades (MVC), validação de dados com Zod, autenticação JWT, e integração com banco de dados PostgreSQL via Prisma ORM.

<parameter name="filePath">c:\Users\ks\Desktop\Estudos\Udemy\SUJEITO\_PROGRAMADOR\fullstack-reactjs\secao7\pizzaria\backend\guia-criacao-app.md