### **Cronograma Detalhado de Desenvolvimento do Projeto PDV em 6 Dias**

#### **Dia 1: Configuração Inicial e Planejamento**

1. **Configuração do Ambiente de Desenvolvimento**
   * **Instalação de Node.js e npm:**
     + Baixar e instalar a versão LTS do Node.js, que inclui o npm (gerenciador de pacotes). Verificar a instalação usando os comandos node -v e npm -v.
   * **Instalação do PostgreSQL:**
     + Baixar e instalar o PostgreSQL. Configurar uma senha para o usuário postgres e criar um banco de dados inicial.
   * **Criação do Repositório no GitHub:**
     + Criar um novo repositório no GitHub para versionamento do código. Clonar o repositório localmente e configurar o Git (git config --global user.name "Seu Nome" e git config --global user.email "seu.email@exemplo.com").
   * **Configuração do Ambiente de Desenvolvimento:**
     + Instalar e configurar o Visual Studio Code com as extensões necessárias (ESLint, Prettier, etc.).
   * **Configuração de ESLint e Prettier:**
     + Instalar ESLint e Prettier como dependências de desenvolvimento e configurar arquivos .eslintrc.json e .prettierrc para padronizar o código.
2. **Planejamento do Projeto**
   * **Análise de Requisitos:**
     + Ler e compreender os requisitos do projeto, focando nas funcionalidades principais e nos endpoints necessários.
   * **Definição da Estrutura de Diretórios:**
     + Estruturar o projeto com diretórios para src (código fonte), config (configurações), controllers (lógica de negócio), models (modelos de dados), e routes (rotas da API).
   * **Documentação Inicial:**
     + Criar um documento com a descrição do projeto, objetivos, público-alvo, e um esboço dos fluxos de trabalho e funcionalidades principais.

#### **Dia 2: Banco de Dados e Estruturação do Projeto**

1. **Criação do Banco de Dados**
   * **Configuração do PostgreSQL:**
     + Acessar o PostgreSQL usando pgAdmin ou linha de comando e criar o banco de dados pdv. Definir a codificação como UTF-8.
   * **Desenvolvimento do Script SQL:**
     + Criar um script SQL para a criação das tabelas usuarios, categorias, clientes, pedidos, e produtos. Incluir comandos para criar índices e constraints, como chaves primárias e estrangeiras.
   * **Inserção de Dados Iniciais:**
     + Inserir categorias iniciais no banco de dados, como Informática, Celulares, Beleza e Perfumaria, etc., usando o script SQL.
2. **Início do Desenvolvimento Backend**
   * **Início do Projeto Node.js:**
     + Inicializar um novo projeto Node.js com o comando npm init -y e instalar dependências básicas como express, pg (cliente PostgreSQL), e dotenv (gerenciamento de variáveis de ambiente).
   * **Estruturação dos Diretórios:**
     + Criar diretórios para src/config (configurações de ambiente), src/controllers (lógica de rotas), src/models (esquemas de banco de dados), e src/routes (definição de rotas).
   * **Configuração do Servidor:**
     + Configurar o servidor Express no arquivo principal (ex: index.js ou server.js), definir middleware básicos como express.json() e cors(), e configurar a porta de escuta.

#### **Dia 3: Implementação de Funcionalidades Básicas e API**

1. **Desenvolvimento de Endpoints Iniciais**
   * **Listar Categorias (GET /categoria):**
     + Criar uma rota para listar todas as categorias no banco de dados. Implementar a lógica no controlador para buscar categorias e retornar uma lista em formato JSON.
   * **Cadastrar Usuário (POST /usuario):**
     + Criar uma rota para cadastrar novos usuários. Implementar validações de entrada (ex: email único, senha segura), e salvar os dados no banco de dados.
   * **Validação de Dados:**
     + Usar Joi ou outra biblioteca de validação para garantir que os dados recebidos estejam no formato correto e atender às regras de negócio.
2. **Autenticação e Autorização**
   * **Implementação de JWT:**
     + Configurar JSON Web Tokens (JWT) para autenticação de usuários. Criar uma função para gerar tokens após o login e middleware para validar tokens em rotas protegidas.
   * **Middleware de Autenticação:**
     + Desenvolver middleware que verifica a presença e validade de um token JWT nos cabeçalhos das requisições. Se inválido ou ausente, retornar um status de erro apropriado (ex: 401 Unauthorized).

#### **Dia 4: Funcionalidades de Usuário e Segurança**

1. **Gerenciamento de Usuários**
   * **Login de Usuários (POST /login):**
     + Criar uma rota para login de usuários. Implementar validação de credenciais, gerar e retornar um token JWT em caso de sucesso.
   * **Redefinição de Senha (PATCH /usuario/redefinir):**
     + Criar uma rota para redefinir senhas. Implementar lógica para verificar a senha antiga, validar a nova senha, atualizar o banco de dados, e enviar um email de confirmação.
2. **Segurança e Boas Práticas**
   * **Proteção contra Ataques:**
     + Implementar medidas de segurança como proteção contra SQL Injection usando consultas parametrizadas, e Cross-Site Scripting (XSS) através da sanitização de entradas.
   * **Configuração de Segurança HTTP:**
     + Configurar cabeçalhos de segurança HTTP, incluindo Content Security Policy (CSP), e CORS para limitar acessos indesejados.

#### **Dia 5: Testes e Integração**

1. **Testes**
   * **Testes Unitários:**
     + Escrever testes unitários para funções críticas, como validação de dados e manipulação de tokens JWT, usando Jest.
   * **Testes de Integração:**
     + Criar testes de integração para validar a comunicação entre componentes, como verificações de endpoints com Supertest.
   * **Testes Manuais:**
     + Realizar testes manuais de fluxo completo (end-to-end) para garantir que todas as funcionalidades estejam funcionando conforme esperado.
2. **Integração com Vercel e GitHub**
   * **Deploy com Vercel:**
     + Configurar Vercel para o deploy contínuo da aplicação. Verificar variáveis de ambiente e configurações de build.
   * **Configuração de CI/CD com GitHub Actions:**
     + Configurar pipelines no GitHub Actions para automação de testes e deploy. Incluir steps para instalação de dependências, execução de testes, e deploy no Vercel.

#### **Dia 6: Documentação, Refatoração e Entrega**

1. **Documentação e Refatoração**
   * **Documentação da API com Swagger:**
     + Criar documentação Swagger para a API, detalhando todos os endpoints, parâmetros de entrada, respostas esperadas e códigos de status.
   * **Escrita de README:**
     + Elaborar um README detalhado no GitHub, incluindo descrição do projeto, instruções de instalação, configuração, uso, exemplos de requisição/resposta, e informações de contribuição.
   * **Refatoração de Código:**
     + Revisar o código para melhorias de legibilidade, eficiência e limpeza. Garantir que todos os comentários estejam atualizados e que o código siga as melhores práticas de desenvolvimento.
2. **Deploy Final e Revisão**
   * **Deploy Final:**
     + Realizar o deploy final da aplicação no Vercel, garantindo que todas as funcionalidades estejam operacionais e que a aplicação esteja pronta para uso em produção.
   * **Testes em Produção:**
     + Executar testes finais em produção para verificar a integridade e o funcionamento do sistema. Monitorar logs e métricas de desempenho para identificar e corrigir quaisquer problemas.
   * **Entrega e Feedback:**
     + Preparar a documentação final do projeto e entregar para avaliação. Solicitar feedback de usuários ou avaliadores para futuras melhorias.

### **Ferramentas e Bibliotecas Utilizadas**

* **Linguagem e Frameworks:** Node.js, Express
* **Banco de Dados:** PostgreSQL
* **Autenticação:** JSON Web Tokens (JWT)
* **Validação de Dados:** Joi
* **Criptografia:** bcrypt
* **Testes:** Jest, Supertest
* **Deploy:** Vercel
* **Controle de Versão:** Git (GitHub)
* **Documentação:** Swagger, README no GitHub

Este manual detalhado fornece uma orientação passo a passo para o desenvolvimento do projeto PDV, garantindo uma abordagem organizada e eficiente para cada fase do projeto. Boa sorte com o desenvolvimento, Fábio! Se precisar de mais alguma coisa, estarei aqui para ajudar. 😊

4o

Conseguiria enriquecer mais esta documentação?

ChatGPT

Claro, Fábio! Vamos enriquecer ainda mais esta documentação, fornecendo detalhes adicionais e orientações para cada etapa do desenvolvimento. Isso inclui exemplos de código, dicas práticas e explicações mais detalhadas para ajudar no processo de criação do projeto PDV.

### **Cronograma Detalhado e Enriquecido de Desenvolvimento do Projeto PDV em 6 Dias**

#### **Dia 1: Configuração Inicial e Planejamento**

1. **Configuração do Ambiente de Desenvolvimento**
   * **Instalação de Node.js e npm:**

Baixe a versão LTS de [Node.js](https://nodejs.org/). Verifique a instalação com os comandos:  
bash  
Copiar código  
node -v

npm -v

* + **Instalação do PostgreSQL:**
    - Baixe o PostgreSQL de [PostgreSQL.org](https://www.postgresql.org/download/). Durante a instalação, configure uma senha para o usuário postgres e crie um banco de dados inicial.
  + **Criação do Repositório no GitHub:**

Crie um novo repositório no GitHub. Localmente, execute:  
bash  
Copiar código  
git clone <URL do repositório>

cd <nome do repositório>

git config --global user.name "Seu Nome"

git config --global user.email "seu.email@exemplo.com"

* + **Configuração do Ambiente de Desenvolvimento:**
    - Instale o [Visual Studio Code](https://code.visualstudio.com/). Adicione extensões úteis como:
      * ESLint
      * Prettier - Code formatter
      * PostgreSQL (para interagir com o banco de dados)
  + **Configuração de ESLint e Prettier:**

Instale as dependências:  
bash  
Copiar código  
npm install eslint prettier eslint-config-prettier eslint-plugin-prettier --save-dev

Crie um arquivo .eslintrc.json:  
json  
Copiar código  
{

"env": {

"browser": true,

"es2021": true

},

"extends": ["eslint:recommended", "plugin:prettier/recommended"],

"parserOptions": {

"ecmaVersion": 12,

"sourceType": "module"

},

"rules": {}

}

Crie um arquivo .prettierrc:  
json  
Copiar código  
{

"semi": true,

"singleQuote": true,

"trailingComma": "all"

}

1. **Planejamento do Projeto**
   * **Análise de Requisitos:**
     + Estude o documento do desafio para entender as funcionalidades essenciais, como o sistema de PDV, gestão de usuários, e segurança.
   * **Definição da Estrutura de Diretórios:**
     + Planeje a estrutura com:
       - /src: Código fonte
       - /src/config: Configurações de ambiente e banco de dados
       - /src/controllers: Lógica de negócio e manipulação de dados
       - /src/models: Esquemas de banco de dados
       - /src/routes: Definição de rotas da API
   * **Documentação Inicial:**
     + Crie um documento com a descrição do projeto, especificações técnicas e fluxos de trabalho. Use diagramas para ilustrar a arquitetura do sistema e o fluxo de dados.

#### **Dia 2: Banco de Dados e Estruturação do Projeto**

1. **Criação do Banco de Dados**
   * **Configuração do PostgreSQL:**

Use pgAdmin ou a linha de comando para criar o banco pdv:  
sql  
Copiar código  
CREATE DATABASE pdv;

* + **Desenvolvimento do Script SQL:**

Crie o script create\_tables.sql com comandos como:  
sql  
Copiar código  
CREATE TABLE usuarios (

id SERIAL PRIMARY KEY,

nome VARCHAR(255) NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

senha VARCHAR(255) NOT NULL

);

CREATE TABLE categorias (

id SERIAL PRIMARY KEY,

descricao VARCHAR(255) NOT NULL

);

-- Tabelas adicionais

* + **Inserção de Dados Iniciais:**

Adicione categorias iniciais:  
sql  
Copiar código  
INSERT INTO categorias (descricao) VALUES

('Informática'),

('Celulares'),

('Beleza e Perfumaria'),

('Mercado'),

('Livros e Papelaria'),

('Brinquedos'),

('Moda'),

('Bebê'),

('Games');

1. **Início do Desenvolvimento Backend**
   * **Início do Projeto Node.js:**

Inicialize o projeto:  
bash  
Copiar código  
npm init -y

npm install express pg dotenv

* + **Estruturação dos Diretórios:**
    - Crie os diretórios conforme planejado e comece com o básico, como src/index.js para iniciar o servidor.
  + **Configuração do Servidor:**

Configure o servidor no index.js:  
javascript  
Copiar código  
const express = require('express');

const app = express();

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

app.use(express.json());

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

#### **Dia 3: Implementação de Funcionalidades Básicas e API**

1. **Desenvolvimento de Endpoints Iniciais**
   * **Listar Categorias (GET /categoria):**

Defina a rota e o controlador:  
javascript  
Copiar código  
// src/routes/categoryRoutes.js

const express = require('express');

const router = express.Router();

const { listCategories } = require('../controllers/categoryController');

router.get('/', listCategories);

module.exports = router;

javascript  
Copiar código  
// src/controllers/categoryController.js

const pool = require('../config/db');

const listCategories = async (req, res) => {

try {

const result = await pool.query('SELECT \* FROM categorias');

res.status(200).json(result.rows);

} catch (error) {

res.status(500).json({ error: 'Erro ao listar categorias' });

}

};

module.exports = { listCategories };

* + **Cadastrar Usuário (POST /usuario):**

Implemente a lógica para criar novos usuários:  
javascript  
Copiar código  
// src/controllers/userController.js

const bcrypt = require('bcrypt');

const pool = require('../config/db');

const registerUser = async (req, res) => {

const { nome, email, senha } = req.body;

try {

const hashedPassword = await bcrypt.hash(senha, 10);

await pool.query('INSERT INTO usuarios (nome, email, senha) VALUES ($1, $2, $3)', [nome, email, hashedPassword]);

res.status(201).json({ message: 'Usuário criado com sucesso' });

} catch (error) {

res.status(500).json({ error: 'Erro ao criar usuário' });

}

};

module.exports = { registerUser };

* + **Validação de Dados:**

Use Joi para validar entradas:  
javascript  
Copiar código  
const Joi = require('joi');

const userSchema = Joi.object({

nome: Joi.string().min(3).required(),

email: Joi.string().email().required(),

senha: Joi.string().min(6).required(),

});

const validateUser = (req, res, next) => {

const { error } = userSchema.validate(req.body);

if (error) return res.status(400).json({ error: error.details[0].message });

next();

};

module.exports = { validateUser };

1. **Autenticação e Autorização**
   * **Implementação de JWT:**

Configure JWT para autenticação:  
javascript  
Copiar código  
const jwt = require('jsonwebtoken');

const generateToken = (userId) => {

return jwt.sign({ id: userId }, process.env.JWT\_SECRET, { expiresIn: '1h' });

};

module.exports = { generateToken };

* + **Middleware de Autenticação:**

Desenvolva middleware para validar JWT:  
javascript  
Copiar código  
const jwt = require('jsonwebtoken');

const authenticateToken = (req, res, next) => {

const token = req.headers.authorization?.split(' ')[1];

if (!token) return res.status(401).json({ error: 'Acesso negado' });

jwt.verify(token, process.env.JWT\_SECRET, (err, user) => {

if (err) return res.status(403).json({ error: 'Token inválido' });

req.user = user;

next();

});

};

module.exports = { authenticateToken };

#### **Dia 4: Funcionalidades de Usuário e Segurança**

1. **Gerenciamento de Usuários**
   * **Login de Usuários (POST /login):**

Valide credenciais e gere token:  
javascript  
Copiar código  
const loginUser = async (req, res) => {

const { email, senha } = req.body;

try {

const user = await pool.query('SELECT \* FROM usuarios WHERE email = $1', [email]);

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

const validPassword = await bcrypt.compare(senha, user.rows[0].senha);

if (!validPassword) return res.status(400).json({ error: 'Senha incorreta' });

const token = generateToken(user.rows[0].id);

res.json({ token });

} catch (error) {

res.status(500).json({ error: 'Erro ao realizar login' });

}

};

module.exports = { loginUser };

* + **Redefinição de Senha (PATCH /usuario/redefinir):**

Implemente a lógica de redefinição:  
javascript  
Copiar código  
const resetPassword = async (req, res) => {

const { email, senha\_antiga, senha\_nova } = req.body;

try {

const user = await pool.query('SELECT \* FROM usuarios WHERE email = $1', [email]);

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

const validPassword = await bcrypt.compare(senha\_antiga, user.rows[0].senha);

if (!validPassword) return res.status(400).json({ error: 'Senha antiga incorreta' });

const hashedNewPassword = await bcrypt.hash(senha\_nova, 10);

await pool.query('UPDATE usuarios SET senha = $1 WHERE email = $2', [hashedNewPassword, email]);

res.json({ message: 'Senha redefinida com sucesso' });

} catch (error) {

res.status(500).json({ error: 'Erro ao redefinir senha' });

}

};

module.exports = { resetPassword };

1. **Segurança e Boas Práticas**
   * **Proteção contra Ataques:**

Use consultas parametrizadas e sanitize inputs:  
javascript  
Copiar código  
// Exemplo de consulta segura

const getUserById = async (id) => {

const result = await pool.query('SELECT \* FROM usuarios WHERE id = $1', [id]);

return result.rows[0];

};

* + **Configuração de Segurança HTTP:**

Configure o Helmet para segurança:  
javascript  
Copiar código  
const helmet = require('helmet');

app.use(helmet());

#### **Dia 5: Testes e Integração**

1. **Testes**
   * **Testes Unitários:**

Use Jest para testes unitários:  
javascript  
Copiar código  
const { validateUser } = require('../src/middlewares/validateUser');

test('Validar usuário com dados corretos', () => {

const req = { body: { nome: 'Teste', email: 'teste@exemplo.com', senha: 'senha123' } };

const res = {};

const next = jest.fn();

validateUser(req, res, next);

expect(next).toHaveBeenCalled();

});

* + **Testes de Integração:**

Use Supertest para testar endpoints:  
javascript  
Copiar código  
const request = require('supertest');

const app = require('../src/index');

describe('Testar rota de login', () => {

it('Deve logar usuário com credenciais corretas', async () => {

const response = await request(app)

.post('/login')

.send({ email: 'teste@exemplo.com', senha: 'senha123' });

expect(response.statusCode).toBe(200);

expect(response.body).toHaveProperty('token');

});

});

* + **Testes Manuais:**
    - Teste a API manualmente usando ferramentas como Postman ou Insomnia. Verifique todos os fluxos de usuário e cenários de erro.

1. **Integração com Vercel e GitHub**
   * **Deploy com Vercel:**
     + Conecte o repositório GitHub ao Vercel e configure variáveis de ambiente (por exemplo, DATABASE\_URL, JWT\_SECRET).
   * **Configuração de CI/CD com GitHub Actions:**

Configure o GitHub Actions:  
yaml  
Copiar código  
name: CI/CD Pipeline

on: [push]

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Node.js

uses: actions/setup-node@v2

with:

node-version: '14'

- name: Install dependencies

run: npm install

- name: Run tests

run: npm test

- name: Deploy to Vercel

run: npx vercel --prod

env:

VERCEL\_TOKEN: ${{ secrets.VERCEL\_TOKEN }}

#### **Dia 6: Documentação, Refatoração e Entrega**

1. **Documentação e Refatoração**
   * **Documentação da API com Swagger:**

Configure Swagger para documentar a API:  
javascript  
Copiar código  
const swaggerUi = require('swagger-ui-express');

const swaggerDocument = require('./swagger.json');

app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));

* + - * Inclua descrições detalhadas para cada endpoint, parâmetros de entrada e saída, e exemplos de respostas.
  + **Escrita de README:**
    - Inclua no README:
      * **Descrição do Projeto:** Breve descrição e propósito do sistema.
      * **Instalação e Configuração:** Instruções para instalar dependências, configurar variáveis de ambiente e iniciar o servidor.
      * **Uso:** Exemplos de requisições e respostas para os principais endpoints.
      * **Contribuição:** Guia de contribuição, incluindo como criar branches, fazer pull requests e padrões de codificação.
      * **Licença:** Informações sobre a licença do projeto.
  + **Refatoração de Código:**
    - Revise todo o código para identificar e corrigir duplicação, melhorar a legibilidade e garantir a conformidade com as melhores práticas de desenvolvimento. Utilize ferramentas de análise estática de código para identificar possíveis problemas.

1. **Deploy Final e Revisão**
   * **Deploy Final:**
     + Certifique-se de que todas as dependências e configurações estão corretas para o ambiente de produção. Verifique se todas as variáveis de ambiente estão configuradas no Vercel.
   * **Testes em Produção:**
     + Realize testes completos no ambiente de produção, incluindo testes de carga e segurança. Use ferramentas como o OWASP ZAP para identificar vulnerabilidades.
   * **Entrega e Feedback:**
     + Prepare uma apresentação ou relatório final detalhando o projeto, as decisões de design, as funcionalidades implementadas e os resultados dos testes. Solicite feedback dos usuários ou avaliadores e documente sugestões para melhorias futuras.

### **Ferramentas e Bibliotecas Utilizadas**

* **Linguagem e Frameworks:** Node.js, Express
* **Banco de Dados:** PostgreSQL
* **Autenticação:** JSON Web Tokens (JWT)
* **Validação de Dados:** Joi
* **Criptografia:** bcrypt
* **Testes:** Jest, Supertest
* **Deploy:** Vercel
* **Controle de Versão:** Git (GitHub)
* **Documentação:** Swagger, README no GitHub
* **Outras Ferramentas:** pgAdmin, Visual Studio Code, Postman/Insomnia, GitHub Actions, OWASP ZAP