# Documento de Orientação ao Desenvolvimento do MVP do e-Store

## 1. Introdução

### 1.1 Propósito

O propósito deste documento é fornecer diretrizes claras e concisas para o desenvolvimento do MVP do sistema e-Store. Ele aborda as melhores práticas de codificação, ferramentas a serem utilizadas, padrões de projeto, e considerações para garantir a qualidade e manutenção do código.

### 1.2 Escopo

Este documento é destinado a todos os desenvolvedores envolvidos no projeto, incluindo desenvolvedores back-end, front-end e de banco de dados. A aplicação das diretrizes aqui apresentadas contribuirá para um desenvolvimento mais coeso, eficiente e sustentável.

## 2. Estrutura do Projeto

### 2.1 Organização de Diretórios

A estrutura de diretórios deve seguir um padrão organizado que facilite a navegação e manutenção do código. Abaixo está uma sugestão de estrutura de diretórios:

e-store/

│

├── backend/

│ ├── app/

│ ├── config/

│ ├── database/

│ ├── routes/

│ ├── tests/

│ └── ...

│

├── frontend/

│ ├── src/

│ ├── public/

│ ├── tests/

│ └── ...

│

├── docs/

├── .gitignore

├── README.md

└── …

## 3. Tecnologias e Ferramentas

### 3.1 Linguagens de Programação

* **Back-end:** PHP (Laravel)
* **Front-end:** JavaScript (React ou Vue.js)
* **Banco de Dados:** MySQL

### 3.2 Ferramentas de Desenvolvimento

* **Controle de Versão:** Git (GitHub ou GitLab)
* **Gerenciamento de Pacotes:** Composer (PHP) e npm/yarn (JavaScript)
* **IDE:** PHPStorm, Visual Studio Code ou outra de preferência
* **Ambiente de Desenvolvimento:** Docker (para ambientes consistentes)

## 4. Padrões de Codificação

### 4.1 Princípios de Design

* **SOLID:** Aplicar os princípios SOLID para garantir um design de software robusto e manutenível.
* **DRY (Don't Repeat Yourself):** Evitar duplicação de código sempre que possível.
* **KISS (Keep It Simple, Stupid):** Manter o código simples e direto.

### 4.2 Convenções de Nomenclatura

* Usar camelCase para variáveis e funções.
* Usar PascalCase para classes.
* Usar snake\_case para nomes de tabelas e colunas no banco de dados.

### 4.3 Documentação de Código

* Comentar o código sempre que necessário, explicando a lógica complexa e decisões de design.
* Utilizar ferramentas de geração de documentação, como PHPDoc e JSDoc, para manter a documentação atualizada.

## 5. Testes

### 5.1 Práticas de Testes

* Implementar testes unitários e de integração para garantir que cada componente funcione como esperado.
* Utilizar ferramentas de testes adequadas:
  + **Back-end:** PHPUnit para testes em PHP.
  + **Front-end:** Jest ou Cypress para testes em JavaScript.

### 5.2 Integração Contínua

* Configurar um pipeline de integração contínua (CI) para executar testes automaticamente em cada commit ou pull request.
* Usar ferramentas como GitHub Actions, Travis CI ou Jenkins.

## 6. Controle de Versão

### 6.1 Fluxo de Trabalho

* Utilizar o modelo Git Flow para gerenciar branches:
  + main: branch de produção.
  + develop: branch de desenvolvimento.
  + feature/\*: branches para novas funcionalidades.
  + bugfix/\*: branches para correções de bugs.

### 6.2 Mensagens de Commit

* Escrever mensagens de commit claras e concisas. Seguir o padrão:

tipo: mensagem

Exemplo:

feat: adicionar funcionalidade de login

fix: corrigir erro de validação de e-mail

## 7. Segurança

### 7.1 Boas Práticas de Segurança

* **Criptografia:** Utilizar algoritmos de hash seguros (como bcrypt) para senhas.
* **Validação de Entrada:** Validar e sanitizar todas as entradas do usuário para evitar injeção de SQL e outros ataques.
* **Controle de Acesso:** Implementar autenticação e autorização adequadas.

## 8. Considerações Finais

### 8.1 Revisão de Código

* Realizar revisões de código (code reviews) para melhorar a qualidade do código e compartilhar conhecimentos entre a equipe.

### 8.2 Feedback e Iteração

* Incentivar a equipe a fornecer feedback e iterar sobre as funcionalidades, melhorando continuamente o produto.