Padroes de Projeto

Criado por: Júnior Silva
Data: 30/11/2024

Sumario

- Design (Figma, Penpot)
- Gestão de projetos (GitHub, Taiga)
- Integração e deploy contínuos (CircleCI, GitHub Workflows)

Introdução

Este documento tem como objetivo direcionar os membros do **Oiticica Valley** envolvidos no setor de desenvolvimento para adotar ferramentas e práticas desejáveis na criação de sistemas web, desktop, mobile, backend, infraestrutura e outros.

Embora este documento recomende várias ferramentas e práticas, **não é esperado que todas sejam utilizadas em um único projeto**. Em geral, será necessário utilizar apenas um subconjunto de ferramentas, definido de acordo com as necessidades do projeto.

Os padrões de nomenclatura, estrutura de pastas e boas práticas descritos aqui são obrigatórios para todos os projetos. Entretanto, a aplicação de ferramentas específicas será limitada para evitar complexidade desnecessária, especialmente em projetos menores e mais simples.

Exemplo de Aplicação Prática

Vamos imaginar o desenvolvimento de um sistema básico de estoque para uma mercearia. As práticas e ferramentas selecionadas poderiam incluir:

Padrões gerais para todos os setores:

1. Sistemas operacionais utilizados (ex.: Linux Ubuntu).
2. Editores de código recomendados (ex.: VS Code).
3. Ferramentas de comunicação utilizadas (ex.: Discord).
4. Nomenclaturas (arquivos, pastas, classes, variáveis).
5. Gerenciamento de versão (ex.: GitHub).
6. Padrões de commit (ex.: Commitizen).
7. Documentação (ex.: Notion).
8. Infraestrutura (ex.: Docker).

Específicos para Frontend:

1. Estrutura de projetos Frontend.
2. Tecnologia (ex.: Next.js, TailwindCSS).
3. Testes (ex.: Jest, Cypress).

Específicos para Backend:

1. Estrutura de projetos Backend.
2. Tecnologia (ex.: NestJS, TypeORM).
3. Gerenciamento de banco de dados (ex.: PostgreSQL).
4. Testes (ex.: Jest).

Tecnologias e Ferramentas Utilizadas

Nesta seção, definimos as tecnologias e ferramentas essenciais para o desenvolvimento, abrangendo sistemas operacionais, editores de código e ferramentas de comunicação. Essas diretrizes garantem uniformidade, eficiência e uma base sólida para o trabalho em equipe.

Sistemas Operacionais

Os sistemas operacionais recomendados são:

- **Windows**: Deve atender aos seguintes requisitos mínimos:
    - 20 GB de RAM
    - SSD
    - Processador Intel Core i3 (10ª geração ou superior)
    - Versão original e licenciada
- **MacOS**: Qualquer versão que atenda às necessidades do desenvolvedor.
- **Linux**: Qualquer distribuição com domínio comprovado. Caso contrário, o padrão recomendado é o **Ubuntu**, com as configurações mínimas:
    - 8 GB de RAM
    - SSD

A escolha do sistema operacional deve considerar desempenho, confiabilidade e conformidade com licenças.

Editores de Código Recomendados

O editor de código padrão é o **Visual Studio Code (VSCode)**, devido à sua ampla gama de extensões, suporte e eficiência.

No entanto, o desenvolvedor pode optar por qualquer outro editor de código, desde que:

- Seja original e licenciado para uso, ou
- Seja gratuito e legal para utilização sem implicações jurídicas.

A flexibilidade na escolha do editor permite que cada desenvolvedor trabalhe com a ferramenta que melhor atenda às suas preferências e produtividade.

Ferramentas de Comunicação

A comunicação do time será centralizada no **Discord**, especialmente no grupo da comunidade **Oiticica Valley**.

- **Calls e Reuniões**: Todas as reuniões e chamadas deverão ser realizadas no Discord, promovendo organização e integração no ambiente da comunidade.
- **Mensagens e Documentos**: Para troca de mensagens rápidas, documentação e outros arquivos, podem ser utilizados:
    - **WhatsApp**
    - **Telegram**

- **E-mail**

Essa abordagem híbrida mantém o foco principal no Discord enquanto oferece alternativas para necessidades específicas de comunicação.

Ao seguir estas diretrizes, asseguramos que a equipe utilize ferramentas confiáveis e apropriadas, otimizando o fluxo de trabalho e minimizando problemas técnicos.

Servidores

Nesta seção, detalhamos as soluções de hospedagem e armazenamento de arquivos utilizadas no desenvolvimento de projetos, com recomendações para backend, frontend e upload de arquivos, além de informações sobre bancos de dados.

Hospedagem de Backend

- **Padrão**: A maioria dos projetos backend será hospedada na **Vercel**, pela sua simplicidade e integração eficiente com tecnologias modernas.
- **Alternativa**: Em casos específicos que demandem maior controle ou infraestrutura personalizada, utilizaremos a **DigitalOcean**, que oferece maior flexibilidade e recursos robustos.

Hospedagem de Frontend

O **frontend** também será hospedado, preferencialmente, na **Vercel**, devido ao suporte nativo para frameworks como Next.js e excelente desempenho em deploy contínuo.

Upload de Arquivos

Para gerenciamento de uploads e armazenamento de arquivos, a ferramenta recomendada é o **Cloudinary**, por sua:

- Integração fácil com múltiplos frameworks e linguagens
- Otimização automática de imagens e vídeos
- Escalabilidade para atender projetos de diferentes tamanhos

Bancos de Dados

- **Padrão**: A maioria dos bancos de dados será configurada no **UOLHost**, devido à sua acessibilidade e suporte técnico em português.
- **Alternativa**: Em casos que exijam maior desempenho, escalabilidade ou personalização, utilizaremos **DigitalOcean**, que oferece soluções gerenciadas para bancos de dados como PostgreSQL e MySQL.

Resumo das Ferramentas

Tipo	Ferramenta	Situação de Uso
Backend Hosting	Vercel	Padrão
Backend Hosting	DigitalOcean	Casos específicos
Frontend Hosting	Vercel	Padrão
Upload de Arquivos	Cloudinary	Padrão
Banco de Dados	UOLHost	Padrão
Banco de Dados	DigitalOcean	Casos específicos

Com essas definições, garantimos flexibilidade e eficiência, adaptando cada solução às necessidades específicas do projeto.

Infraestrutura

A infraestrutura é um componente fundamental para a criação de sistemas escaláveis, confiáveis e fáceis de gerenciar. Nesta seção, abordaremos as principais ferramentas que utilizamos para gerenciar, orquestrar e padronizar a infraestrutura de projetos.

Terraform

O **Terraform** é a ferramenta recomendada para gerenciamento de infraestrutura como código (IaC). Com ele, é possível:

- Criar, atualizar e versionar a infraestrutura de forma declarativa.
- Automatizar a criação de recursos em nuvens públicas (como AWS, Azure, Google Cloud) e serviços como DigitalOcean.
- Garantir consistência entre ambientes (desenvolvimento, staging, produção).

> **Recomendação:** Sempre versionar os arquivos de configuração do Terraform no repositório do projeto para controle de mudanças.

Docker

O **Docker** é utilizado para criar e gerenciar contêineres, permitindo que aplicações sejam executadas de forma isolada e consistente em qualquer ambiente. Ele é essencial para:

- Padronizar o ambiente de desenvolvimento, evitando problemas como "funciona na minha máquina".
- Reduzir o tempo de configuração e implantação de sistemas.
- Facilitar o escalonamento horizontal em ambientes de produção.

> **Exemplo de uso comum:** Criar um contêiner para rodar o backend com todas as dependências pré-instaladas.

Docker Compose

O **Docker Compose** complementa o Docker, permitindo a orquestração de múltiplos contêineres de forma simples. Ele é usado para:

- Configurar e gerenciar serviços interdependentes (ex.: backend, frontend, banco de dados).
- Automatizar o início de todo o ecossistema do projeto com um único comando.
- Simplificar o desenvolvimento local e o teste de sistemas complexos.

> **Exemplo de uso comum:** Um arquivo docker-compose.yml que inicia o backend, o banco de dados e o Redis em contêineres separados, configurados para interagir entre si.

Resumo das Ferramentas

Ferramenta	Propósito	Situação de Uso
Terraform	Gerenciamento de infraestrutura como código	Ambientes na nuvem
Docker	Criação e execução de contêineres	Desenvolvimento e produção
Docker Compose	Orquestração de múltiplos contêineres	Desenvolvimento e testes

Com o uso dessas ferramentas, garantimos uma infraestrutura robusta e adaptável, alinhada às melhores práticas de DevOps e engenharia de software.

Tecnologias por Tipo de Projeto

Esta seção apresenta as tecnologias recomendadas para cada tipo de projeto, com foco em garantir eficiência, boas práticas e facilidade de manutenção.

Backend

Para o desenvolvimento backend, as tecnologias recomendadas são:

- **NestJS**
  Um framework Node.js progressivo, ideal para criar aplicações escaláveis e altamente testáveis.
    - Utiliza conceitos como injeção de dependências e módulos, facilitando a organização do código.
    - Oferece suporte nativo a Express ou Fastify como servidores HTTP.
    - Indicado para projetos de média a grande complexidade.
- **Node.js**
  Utilizado em projetos que não necessitam de toda a estrutura do NestJS.
    - **Express**: Um framework minimalista e flexível, ideal para APIs simples e rápidas.
    - **Fastify**: Uma alternativa ao Express, focada em performance e menor consumo de recursos.

Escolha do servidor:

- Use **Express** para projetos onde a simplicidade é uma prioridade.
- Use **Fastify** quando a performance for essencial.

Frontend

Para o desenvolvimento de interfaces web, as tecnologias recomendadas incluem:

- **Next.js**
  Um framework React que oferece renderização híbrida (SSR e SSG), ideal para projetos modernos.
    - Suporte nativo a roteamento, otimização de imagens e SEO.
    - Excelente escolha para aplicações web rápidas e otimizadas.
- **TailwindCSS**
  Uma biblioteca de classes utilitárias que facilita o desenvolvimento de interfaces responsivas e personalizáveis.
    - Reduz a dependência de CSS personalizado, acelerando o tempo de entrega de layouts.
- **JavaScript e HTML**
  Usados em conjunto para criar funcionalidades e estruturar o layout.
    - Essenciais para qualquer projeto frontend, servindo como base de desenvolvimento.

Recomendação adicional: Sempre que possível, siga as melhores práticas de acessibilidade (WCAG) ao criar interfaces.

Mobile

Para o desenvolvimento mobile, a principal tecnologia recomendada é:

- **React Native (Expo)**
  Uma plataforma robusta para o desenvolvimento de aplicativos móveis multiplataforma (iOS e Android).
    - **Expo**: Facilita a configuração e execução de projetos, especialmente para desenvolvedores iniciantes ou equipes pequenas.
    - Componentes reutilizáveis e integração com APIs nativas.
    - Indicado para projetos que precisam de consistência entre plataformas.

Resumo das Tecnologias

Tipo de Projeto	Ferramenta	Propósito
Backend	NestJS, Node.js	APIs robustas e serviços de backend
Frontend	Next.js, TailwindCSS	Aplicações web modernas e responsivas
Mobile	React Native (Expo)	Desenvolvimento de aplicativos móveis multiplataforma

Com essas tecnologias, podemos criar projetos de alto desempenho, bem estruturados e alinhados às necessidades de cada tipo de aplicação.

Estruturas e Nomenclaturas

Nesta seção, definimos padrões para a organização de projetos e nomenclatura, garantindo consistência, legibilidade e facilidade de manutenção em todos os projetos desenvolvidos.

Estrutura de Projetos

Backend (NestJS)

Para projetos backend, utilizaremos a estrutura padrão oferecida pelo **NestJS**. Essa estrutura modular facilita a escalabilidade e organização do código.

Exemplo de estrutura de pastas:

```
src/
├── modules/          # Módulos organizados por funcionalidade
│   ├── user/         # Exemplo de módulo
│   │   ├── dto/      # Data Transfer Objects
│   │   ├── entities/ # Entidades
│   │   ├── user.controller.ts
│   │   ├── user.service.ts
│   │   └── user.module.ts
│   └── ...
├── config/           # Configurações do projeto
├── common/           # Filtros, interceptadores, guards, pipes e utilitários
├── main.ts           # Ponto de entrada da aplicação
├── app.module.ts     # Módulo principal
└── ...
```

Frontend (Next.js)

Para projetos frontend, utilizaremos a estrutura padrão do **Next.js**, que organiza pastas e arquivos baseados em páginas e componentes.

Exemplo de estrutura de pastas:

```
src/
├── pages/            # Páginas da aplicação
│   ├── index.tsx     # Página inicial
│   ├── about.tsx     # Exemplo de página
│   └── api/          # Rotas de API
├── components/       # Componentes reutilizáveis
├── styles/           # Estilos globais e CSS modules
├── utils/            # Funções utilitárias
├── hooks/            # Hooks personalizados
├── public/           # Arquivos estáticos (imagens, fontes, etc.)
└── ...
```

Nomenclaturas

Arquivos e Pastas

- Utilizar o padrão **kebab-case** para nomes de arquivos e pastas.

- Exemplo: user-controller.ts, order-service.ts.

Classes

- Utilizar o padrão **PascalCase** para nomes de classes.
  - Exemplo: UserService, OrderController.

Variáveis

- Utilizar o padrão **camelCase** para nomes de variáveis.
  - Exemplo: userName, orderId.

Bancos de Dados

- Utilizar o padrão **snake_case** para nomes de tabelas e colunas no banco de dados.
  - Exemplo: user_table, order_id.

JSON de Resposta e Validators

- Utilizar o padrão **camelCase** para os campos de resposta JSON e validações.
  - Exemplo:

```
{
  "userName": "John Doe",
  "orderId": 123
}
```

Padrões Resumidos

Item	Padrão	Exemplo
Arquivos e Pastas	kebab-case	user-service.ts
Classes	PascalCase	UserService, OrderEntity
Variáveis	camelCase	userName, orderId
Bancos de Dados	snake_case	user_table, order_id
JSON e Validators	camelCase	{ "userName": "John Doe" }

Com essas definições, asseguramos que todos os projetos mantêm uma linguagem universal, promovendo integração e colaboração eficiente entre os membros da equipe.

Ferramentas

Nesta seção, apresentamos as ferramentas utilizadas para o gerenciamento de banco de dados, monitoramento de logs e desempenho, e gerenciamento de backend, assegurando eficiência e controle em nossos projetos.

Gerenciamento de Banco de Dados

DBeaver

O **DBeaver** é a ferramenta padrão para gerenciar bancos de dados. Ele oferece suporte a diversas plataformas (PostgreSQL, MySQL, SQLite, etc.), com uma interface intuitiva para executar consultas, visualizar esquemas, e gerenciar dados de forma eficiente.

Funcionalidades principais:

- Suporte a múltiplos bancos de dados.
- Interface gráfica para execução de queries e visualização de tabelas.
- Ferramentas para exportação e importação de dados.
- Compatível com plataformas locais e remotas.

Log e Monitoramento

Para garantir a confiabilidade dos sistemas, utilizamos um conjunto robusto de ferramentas para logs e monitoramento:

Datadog

- Utilizado para monitoramento de performance, rastreamento de métricas em tempo real, e detecção de anomalias em aplicações e infraestrutura.
- Oferece integração com ferramentas de CI/CD, APIs e microserviços.

Sentry

- Ideal para rastrear erros e exceções em aplicações frontend e backend.
- Permite identificar, categorizar, e priorizar problemas críticos para resolução.

OpenTelemetry

- Utilizado para rastreamento distribuído e coleta de métricas em sistemas complexos.
- Integração com diversas plataformas de monitoramento, incluindo o Datadog.

Grafana

- Ferramenta de visualização de dados e criação de dashboards personalizados.
- Utilizado para analisar logs e métricas em tempo real de diferentes fontes de dados.

Gerenciamento de Backend

PM2

O **PM2** é a ferramenta utilizada para gerenciar processos de backend. Ela simplifica o gerenciamento de aplicações Node.js em ambientes de produção, garantindo alta disponibilidade e estabilidade.

Funcionalidades principais:

- Gerenciamento de processos com suporte a balanceamento de carga.
- Logs detalhados para análise de erros e desempenho.
- Monitoramento de métricas em tempo real.
- Suporte a reinicialização automática em caso de falhas.

Resumo das Ferramentas

Categoria	Ferramenta	Finalidade
Gerenciamento de Banco	DBeaver	Visualização e gerenciamento de dados
Log e Monitoramento	Datadog	Monitoramento de métricas e performance
	Sentry	Rastreamento de erros
	OpenTelemetry	Rastreio distribuído
	Grafana	Dashboards personalizados
Gerenciamento de Backend	PM2	Controle de processos Node.js

Estas ferramentas são fundamentais para o desenvolvimento e operação de aplicações robustas, garantindo estabilidade e monitoramento eficaz.

Outros Tópicos

Nesta seção, detalhamos ferramentas, padrões e práticas utilizadas em diferentes aspectos do desenvolvimento de software, garantindo organização, qualidade e escalabilidade nos projetos.

DNS

- **Cloudflare**
  Utilizamos o **Cloudflare** para gerenciamento de DNS, com benefícios como:
  - Proteção contra ataques DDoS.
  - Aceleração de aplicações com cache de conteúdo estático.
  - Certificados SSL gratuitos para comunicação segura.

Gateway de Pagamento

- **Stripe**
  Gateway de pagamento utilizado por sua robustez e suporte a múltiplos métodos de pagamento em escala global.
- **Pagar.me**
  Solução preferida para operações no Brasil, com integração simplificada e suporte ao PIX, boleto bancário e cartões de crédito.

Bancos de Dados

- **MySQL**: Utilizado em sistemas legados ou projetos que exigem alta compatibilidade.
- **PostgreSQL**: Banco de dados principal devido à sua performance e suporte avançado a tipos de dados.
- **SQLite**: Utilizado em testes e aplicações locais leves.

ORMs

- **TypeORM**: Usado como padrão para modelagem e consultas em bancos SQL, especialmente com NestJS.
- **Knex**: Preferido para projetos que requerem maior flexibilidade em queries SQL.

Mensageria e Comunicação em Tempo Real

Ferramentas para entrega de mensagens assíncronas e comunicação em tempo real:

- **Redis**: Para mensagens rápidas em filas e caching.
- **Socket.io** ou **WebSocket**: Para implementações de comunicação em tempo real.

Cache, Cron Jobs e Filas

- **Redis**: Solução principal para caching de dados e gerenciamento de sessões.
- **Bull**: Utilizado para filas de trabalho, especialmente em projetos Node.js.
- **Cron Jobs**: Gerenciados com bibliotecas como node-cron ou pelo próprio sistema operacional.

Documentação

- **Swagger/OpenAPI**: Padrão para documentação de APIs REST.
- **Obsidian** e **Notion**: Usados para documentação interna e organização de conhecimento.

Testes

Frontend

- **Jest** e **Vitest**: Para testes unitários.
- **Cypress**: Ferramenta principal para testes end-to-end.

Backend

- **Jest** e **Vitest**: Para testes unitários e de integração.

Gerenciamento de Versão

- **GitHub**: Plataforma principal para versionamento e colaboração em projetos.

Padrões de Commit

- **Commitizen**: Ferramenta para padronizar mensagens de commit.
- **Husky**: Para execução de hooks do Git, como linting pré-commit.
- **GitHub Workflow**: Para automação de tarefas como lint, testes e deploy.

Design

- **Figma**: Ferramenta principal para criação de interfaces e prototipagem.
- **Penpot**: Alternativa open-source para design colaborativo.

Gestão de Projetos

- **GitHub Projects**: Usado para gestão de tarefas e acompanhamento de progresso.
- **Taiga**: Ferramenta adicional para gerenciamento de projetos ágeis.

Integração e Deploy Contínuos (CI/CD)

- **CircleCI**: Utilizado para pipelines mais complexos com múltiplas etapas.
- **GitHub Workflows**: Preferido para automação e deploy contínuos integrados ao GitHub.

Resumo

Categoria	Ferramentas/Práticas
DNS	Cloudflare
Gateway de Pagamento	Stripe, Pagar.me
Bancos de Dados	MySQL, PostgreSQL, SQLite
ORMs	TypeORM, Knex
Mensageria	Redis, Socket.io, WebSocket
Cache, Filas e Cron Jobs	Redis, Bull, Cron Jobs
Documentação	Swagger/OpenAPI, Obsidian, Notion

Categoria	Ferramentas/Práticas
Testes	Jest, Vitest, Cypress
Gerenciamento de Versão	GitHub, Commitizen, Husky, GitHub Workflows
Design	Figma, Penpot
Gestão de Projetos	GitHub Projects, Taiga
CI/CD	CircleCI, GitHub Workflows

Estas ferramentas e práticas fortalecem os processos de desenvolvimento, garantindo eficiência, qualidade e colaboração contínua.

Finalização

Agradecemos por consultar esta documentação. Nosso objetivo é fornecer um guia claro e detalhado para garantir que os processos de desenvolvimento, infraestrutura, ferramentas e práticas estejam bem alinhados às necessidades do projeto e à cultura da equipe.

Próximos Passos

1. **Revisão Contínua**:
    - A documentação deve ser revisada regularmente para refletir atualizações em ferramentas, processos ou tecnologias.
    - Sugestões de melhorias podem ser enviadas via nossos canais de comunicação oficiais.
2. **Adaptação e Crescimento**:
    - Projetos podem evoluir, e ajustes nas práticas e ferramentas documentadas devem ser feitos para acompanhar novas demandas ou inovações.
3. **Colaboração**:
    - Toda a equipe é incentivada a contribuir com feedback, ideias e boas práticas que enriqueçam este guia.

Contato e Suporte

- **Canal de Comunicação Principal**: Discord da comunidade.
- **E-mails de Suporte**: Utilize os e-mails corporativos para questões mais específicas.
- **Documentação Auxiliar**: Reforce a consulta a documentos complementares no **Notion**, **Obsidian** ou ferramentas relevantes.

Agradecimento

Esta documentação foi criada com o intuito de fortalecer a comunicação e alinhamento entre os membros da equipe.
Agradecemos a todos os envolvidos pelo esforço e dedicação, e esperamos que esta guia seja um recurso útil em cada etapa do desenvolvimento.

Versão da Documentação

- **Versão Atual**: 1.0
- **Última Atualização**: 01/12/2024
- **Responsável pela Manutenção**: [Junior Silva/Oiticica Valley Dev Team]

Continuamos à disposição para esclarecer dúvidas e auxiliar em qualquer necessidade. Vamos juntos construir soluções incríveis!