Módulo 4: Docker e React

1. Docker para Desenvolvimento

1.1 Introdução ao Docker

Docker é uma plataforma de código aberto que automatiza o processo de implantação de aplicações dentro de contêineres. Esses contêineres são leves, portáteis e autocontidos, permitindo que as aplicações sejam executadas de maneira consistente em diferentes ambientes.

Conceitos Fundamentais:

- **Contêiner**: Unidade padronizada de software que empacota código e todas as suas dependências para que a aplicação seja executada de forma rápida e confiável em diferentes ambientes computacionais.
- **Imagem**: Template somente leitura com instruções para criar um contêiner. Uma imagem contém o código, runtime, bibliotecas, variáveis de ambiente e arquivos de configuração necessários para executar uma aplicação.
- **Dockerfile**: Arquivo de texto que contém todas as instruções necessárias para construir uma imagem Docker.
- **Docker Hub**: Repositório público de imagens Docker, onde é possível encontrar imagens oficiais e da comunidade.
- **Docker Compose**: Ferramenta para definir e executar aplicações Docker multi-contêiner.
- **Volume**: Mecanismo para persistir dados gerados e utilizados por contêineres Docker.

Vantagens do Docker:

- **Consistência**: Garante que a aplicação funcione da mesma forma em qualquer ambiente.
- **Isolamento**: Cada contêiner opera independentemente, sem interferir em outros processos.
- **Portabilidade**: Contêineres podem ser executados em qualquer sistema que tenha o Docker instalado.
- **Eficiência**: Contêineres compartilham o kernel do sistema operacional, tornando-os mais leves que máquinas virtuais.
- **Escalabilidade**: Facilita a criação e gerenciamento de múltiplas instâncias da aplicação.
- **Versionamento**: Permite controlar versões de imagens e reverter facilmente quando necessário.

1.2 Instalação e Configuração do Docker

Instalação no Ubuntu:

```
# Atualizar os pacotes
sudo apt-get update

# Instalar pacotes necessários
sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common

# Adicionar a chave GPG oficial do Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

# Adicionar o repositório do Docker
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

# Atualizar os pacotes novamente
sudo apt-get update

# Instalar o Docker CE
sudo apt-get install -y docker-ce

# Verificar a instalação
sudo docker run hello-world
```

Instalação no macOS:

1. Baixar e instalar o Docker Desktop para Mac a partir do site oficial: https://www.docker.com/products/docker-desktop
2. Iniciar o Docker Desktop
3. Verificar a instalação: docker run hello-world

Instalação no Windows:

1. Baixar e instalar o Docker Desktop para Windows a partir do site oficial: https://www.docker.com/products/docker-desktop
2. Garantir que o WSL 2 (Windows Subsystem for Linux 2) esteja instalado e configurado
3. Iniciar o Docker Desktop
4. Verificar a instalação: docker run hello-world

Configuração Pós-instalação:

```
# Adicionar seu usuário ao grupo docker (Linux)
sudo usermod -aG docker $USER

# Aplicar as mudanças (necessário fazer logout e login novamente)
newgrp docker

# Configurar o Docker para iniciar com o sistema
sudo systemctl enable docker

# Verificar a versão do Docker
docker --version

# Verificar informações detalhadas
docker info
```

1.3 Comandos Básicos do Docker

Gerenciamento de Imagens:

```
# Listar imagens
docker images

# Baixar uma imagem do Docker Hub
docker pull node:14

# Construir uma imagem a partir de um Dockerfile
docker build -t minha-app:1.0 .

# Remover uma imagem
docker rmi node:14

# Inspecionar uma imagem
docker image inspect node:14

# Salvar uma imagem como arquivo tar
docker save -o node-image.tar node:14

# Carregar uma imagem a partir de um arquivo tar
docker load -i node-image.tar
```

Gerenciamento de Contêineres:

```
# Criar e iniciar um contêiner
docker run -d -p 8080:80 --name meu-servidor nginx

# Listar contêineres em execução
docker ps

# Listar todos os contêineres (incluindo os parados)
docker ps -a

# Parar um contêiner
docker stop meu-servidor

# Iniciar um contêiner parado
docker start meu-servidor

# Reiniciar um contêiner
docker restart meu-servidor

# Remover um contêiner
docker rm meu-servidor

# Remover um contêiner em execução
docker rm -f meu-servidor

# Ver logs de um contêiner
docker logs meu-servidor

# Acompanhar logs em tempo real
docker logs -f meu-servidor

# Executar um comando dentro de um contêiner em execução
docker exec -it meu-servidor bash
```

Opções Comuns do Comando docker run:

```
# Mapear portas (host:contêiner)
docker run -p 8080:80 nginx

# Executar em modo interativo com terminal
docker run -it ubuntu bash

# Executar em segundo plano (modo detached)
docker run -d nginx

# Definir nome do contêiner
docker run --name meu-nginx nginx

# Montar volume (host:contêiner)
docker run -v /caminho/local:/caminho/contêiner nginx

# Definir variáveis de ambiente
docker run -e VARIAVEL=valor nginx

# Limitar recursos (CPU e memória)
docker run --cpus=0.5 --memory=512m nginx

# Remover contêiner automaticamente após finalização
docker run --rm nginx
```

Redes Docker:

```
# Listar redes
docker network ls

# Criar uma rede
docker network create minha-rede

# Conectar um contêiner a uma rede
docker network connect minha-rede meu-servidor

# Desconectar um contêiner de uma rede
docker network disconnect minha-rede meu-servidor

# Inspecionar uma rede
docker network inspect minha-rede

# Remover uma rede
docker network rm minha-rede
```

Volumes Docker:

```
# Listar volumes
docker volume ls

# Criar um volume
docker volume create meu-volume

# Inspecionar um volume
docker volume inspect meu-volume

# Remover um volume
docker volume rm meu-volume

# Remover volumes não utilizados
docker volume prune
```

1.4 Criando Dockerfiles

Um Dockerfile é um arquivo de texto que contém todas as instruções necessárias para construir uma imagem Docker. Cada instrução cria uma camada na imagem.

Estrutura Básica de um Dockerfile:

```
# Imagem base
FROM node:14-alpine

# Metadados
LABEL maintainer="seu.email@exemplo.com"
LABEL version="1.0"
LABEL description="Aplicação Node.js em contêiner"

# Definir diretório de trabalho
WORKDIR /app

# Copiar arquivos de dependências
COPY package*.json ./

# Instalar dependências
RUN npm install

# Copiar o restante dos arquivos da aplicação
COPY . .

# Expor porta
EXPOSE 3000

# Definir variáveis de ambiente
ENV NODE_ENV=production

# Comando a ser executado quando o contêiner iniciar
CMD ["npm", "start"]
```

Instruções Comuns em Dockerfiles:

- **FROM**: Define a imagem base
- **WORKDIR**: Define o diretório de trabalho dentro do contêiner
- **COPY/ADD**: Copia arquivos do host para o contêiner
- **RUN**: Executa comandos durante a construção da imagem
- **ENV**: Define variáveis de ambiente
- **EXPOSE**: Informa quais portas o contêiner escuta
- **VOLUME**: Define pontos de montagem para volumes
- **USER**: Define o usuário que executará os comandos
- **CMD**: Define o comando padrão a ser executado quando o contêiner iniciar
- **ENTRYPOINT**: Define o executável principal do contêiner

Boas Práticas para Dockerfiles:

1. **Use imagens oficiais e específicas**: Prefira imagens oficiais e especifique a versão exata.

```
FROM node:14.17.0-alpine3.13
```

2. **Minimize o número de camadas**: Combine comandos RUN relacionados usando && e \.

```
RUN apt-get update && \
    apt-get install -y --no-install-recommends curl && \
    rm -rf /var/lib/apt/lists/*
```

3. **Não instale pacotes desnecessários**: Use variantes alpine ou slim das imagens.

4. **Use .dockerignore**: Exclua arquivos desnecessários da construção.

```
node_modules
npm-debug.log
.git
.env
```

5. **Ordene as instruções por frequência de mudança**: Coloque as instruções que mudam com menos frequência no início do Dockerfile.

6. **Use multi-stage builds**: Reduza o tamanho final da imagem.

```
# Estágio de build
FROM node:14-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Estágio de produção
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

7. **Não execute como root**: Use um usuário não privilegiado.

```
RUN addgroup -g 1000 node && \
    adduser -u 1000 -G node -s /bin/sh -D node
USER node
```

8. **Use HEALTHCHECK**: Verifique se a aplicação está funcionando corretamente.

```
HEALTHCHECK --interval=30s --timeout=3s \
  CMD curl -f http://localhost/ || exit 1
```

1.5 Docker Compose

Docker Compose é uma ferramenta para definir e executar aplicações Docker multi-contêiner. Com um único arquivo YAML, você configura todos os serviços da sua aplicação e, com um único comando, cria e inicia todos os serviços.

Instalação do Docker Compose:

```
# Linux
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

# Verificar a instalação
docker-compose --version
```

No Windows e macOS, o Docker Compose já vem incluído no Docker Desktop.

Estrutura Básica do docker-compose.yml:

```yaml
version: '3.8'

services:
  web:
    build: ./web
    ports:
      - "8080:80"
    depends_on:
      - api
    environment:
      - API_URL=http://api:3000

  api:
    build: ./api
    ports:
      - "3000:3000"
    depends_on:
      - db
    environment:
      - DB_HOST=db
      - DB_USER=postgres
      - DB_PASSWORD=secret
      - DB_NAME=mydb

  db:
    image: postgres:13
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=secret
      - POSTGRES_DB=mydb

volumes:
  postgres_data:
```

Comandos Básicos do Docker Compose:

```
# Criar e iniciar contêineres
docker-compose up

# Criar e iniciar contêineres em segundo plano
docker-compose up -d

# Parar contêineres
docker-compose stop

# Parar e remover contêineres
docker-compose down

# Parar e remover contêineres, redes, imagens e volumes
docker-compose down --rmi all --volumes

# Visualizar logs
docker-compose logs

# Visualizar logs de um serviço específico
docker-compose logs api

# Executar um comando em um serviço
docker-compose exec api npm test

# Listar contêineres
docker-compose ps

# Construir ou reconstruir serviços
docker-compose build

# Verificar a configuração
docker-compose config
```

Recursos Avançados do Docker Compose:

1. **Redes Personalizadas**:

```
networks:
  frontend:
  backend:
    driver: bridge

services:
  web:
    networks:
      - frontend
  api:
    networks:
      - frontend
      - backend
  db:
    networks:
      - backend
```

2. **Volumes Nomeados e Bind Mounts**:

```
services:
  web:
    volumes:
      - ./web:/app  # Bind mount
      - node_modules:/app/node_modules  # Volume nomeado

volumes:
  node_modules:
```

3. **Variáveis de Ambiente e Arquivos .env**:

```
services:
  api:
    env_file:
      - .env.api
    environment:
      - NODE_ENV=development
```

4. **Healthchecks**:

```
services:
  api:
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
```

5. **Escala de Serviços**:

```
docker-compose up -d --scale api=3
```

6. **Profiles para Ambientes Diferentes**:

```
services:
  web:
    profiles: ["dev", "prod"]
  db:
    profiles: ["dev", "prod"]
  adminer:
    profiles: ["dev"]
```

```
docker-compose --profile dev up
```

1.6 Criando uma Imagem Docker para NodeJS

Vamos criar uma imagem Docker otimizada para desenvolvimento com Node.js:

Estrutura de Diretórios:

```
projeto-node/
├── Dockerfile
├── docker-compose.yml
├── .dockerignore
├── package.json
├── src/
│   └── index.js
└── .env
```

Dockerfile para Desenvolvimento:

```
# Imagem base
FROM node:16-alpine

# Instalar dependências globais
RUN apk add --no-cache python3 make g++ git

# Criar diretório da aplicação
WORKDIR /app

# Instalar dependências
COPY package*.json ./
RUN npm install

# Copiar código-fonte
COPY . .

# Expor porta
EXPOSE 3000

# Comando para iniciar em modo de desenvolvimento
CMD ["npm", "run", "dev"]
```

docker-compose.yml:

```yaml
version: '3.8'

services:
  node-app:
    build: .
    container_name: node-dev
    ports:
      - "3000:3000"
    volumes:
      - .:/app
      - /app/node_modules
    environment:
      - NODE_ENV=development
    command: npm run dev
```

.dockerignore:

```
node_modules
npm-debug.log
.git
.gitignore
.env
.DS_Store
```

package.json:

```json
{
  "name": "node-docker-dev",
  "version": "1.0.0",
  "description": "Node.js development with Docker",
  "main": "src/index.js",
  "scripts": {
    "start": "node src/index.js",
    "dev": "nodemon src/index.js",
    "test": "jest"
  },
  "dependencies": {
    "express": "^4.17.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.12",
    "jest": "^27.0.6"
  }
}
```

src/index.js:

```
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

app.get('/', (req, res) => {
  res.json({ message: 'Hello from Docker Node.js app!' });
});

app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

Construir e Executar:

```
# Usando Docker diretamente
docker build -t node-dev .
docker run -p 3000:3000 -v $(pwd):/app -v /app/node_modules --name node-dev node-dev

# Usando Docker Compose
docker-compose up -d
```

Características desta Configuração:

1. **Hot Reloading**: Alterações no código são detectadas automaticamente pelo Nodemon
2. **Volumes**: O código-fonte é montado do host para o contêiner, permitindo edição em tempo real
3. **Isolamento de node_modules**: As dependências ficam isoladas dentro do contêiner
4. **Ambiente de Desenvolvimento**: Configurado especificamente para desenvolvimento

1.7 Melhores Práticas para Docker em Desenvolvimento

1. **Use Volumes para Código-fonte**:
    - Monte o código-fonte como volume para permitir edição em tempo real
    - Isole node_modules dentro do contêiner
2. **Otimize para Reconstrução**:
    - Organize o Dockerfile para aproveitar o cache
    - Coloque as instruções que mudam com menos frequência no início
3. **Ambiente de Desenvolvimento vs. Produção**:
    - Use arquivos docker-compose diferentes para cada ambiente
    - Use variáveis de ambiente para configurar comportamentos específicos
4. **Segurança**:
    - Não armazene segredos no Dockerfile ou imagem
    - Use variáveis de ambiente ou secrets para informações sensíveis
5. **Performance**:
    - Use imagens base leves (alpine)
    - Limpe caches e arquivos temporários após instalações

6. **Debugging**:
    - Configure ferramentas de debugging dentro do contêiner
    - Exponha portas para depuração remota
7. **Integração com IDEs**:
    - Configure sua IDE para trabalhar com contêineres Docker
    - Use extensões como Remote - Containers para VS Code
8. **Persistência de Dados**:
    - Use volumes nomeados para dados que precisam persistir
    - Implemente estratégias de backup para volumes importantes
9. **Rede**:
    - Crie redes Docker dedicadas para isolar serviços relacionados
    - Use nomes de host baseados no nome do serviço para comunicação entre contêineres
10. **Monitoramento**:
    - Implemente healthchecks para verificar a saúde dos serviços
    - Configure logs adequados para facilitar o diagnóstico de problemas

2. React com TypeScript

2.1 Introdução ao React

React é uma biblioteca JavaScript de código aberto para construir interfaces de usuário, especialmente para aplicações de página única (SPAs). Desenvolvida e mantida pelo Facebook (agora Meta), o React permite criar UIs complexas a partir de componentes pequenos e isolados.

Conceitos Fundamentais do React:

1. **Componentes**: Blocos de construção reutilizáveis que encapsulam lógica e UI.
2. **JSX**: Extensão de sintaxe que permite escrever HTML dentro do JavaScript.
3. **Virtual DOM**: Representação em memória do DOM real, que otimiza as atualizações de UI.
4. **Fluxo de Dados Unidirecional**: Os dados fluem de componentes pais para filhos.
5. **Estado (State)**: Dados específicos do componente que podem mudar ao longo do tempo.
6. **Props**: Dados passados de um componente pai para um filho.
7. **Ciclo de Vida**: Métodos que são chamados em diferentes estágios da existência de um componente.
8. **Hooks**: Funções que permitem usar estado e outros recursos do React em componentes funcionais.

Por que usar React?

- **Componentização**: Facilita a reutilização de código e manutenção
- **Performance**: O Virtual DOM otimiza as atualizações de UI

- **Ecossistema Rico**: Grande comunidade e muitas bibliotecas complementares
- **Suporte Corporativo**: Mantido pelo Facebook/Meta e usado por grandes empresas
- **Flexibilidade**: Pode ser integrado gradualmente em projetos existentes
- **React Native**: Permite reutilizar conhecimento para desenvolvimento mobile

2.2 TypeScript com React

TypeScript adiciona tipagem estática ao JavaScript, o que traz vários benefícios ao desenvolvimento React:

Benefícios do TypeScript com React:

1. **Detecção de Erros em Tempo de Compilação**: Identifica erros antes da execução.
2. **Melhor Intellisense e Autocompletar**: Facilita o desenvolvimento com sugestões precisas.
3. **Documentação Integrada**: Os tipos servem como documentação.
4. **Refatoração Mais Segura**: Mudanças em tipos são refletidas em todo o código.
5. **Melhor Manutenção**: Especialmente útil em projetos grandes e equipes.

Configuração do TypeScript em um Projeto React:

```
# Criar um novo projeto React com TypeScript
npx create-react-app meu-app --template typescript

# Ou adicionar TypeScript a um projeto existente
npm install --save typescript @types/node @types/react @types/react-dom @types/jest
```

tsconfig.json para React:

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx"
  },
  "include": ["src"]
}
```

2.3 Criando um Projeto React com TypeScript utilizando Vite

Vite é uma ferramenta de build moderna que oferece uma experiência de desenvolvimento mais rápida comparada ao Create React App, graças ao seu servidor de desenvolvimento com Hot Module Replacement (HMR) extremamente rápido.

Criando um Projeto com Vite:

```
# Criar um novo projeto
npm create vite@latest meu-app-react -- --template react-ts

# Navegar para o diretório do projeto
cd meu-app-react

# Instalar dependências
npm install

# Iniciar o servidor de desenvolvimento
npm run dev
```

Estrutura de Diretórios Gerada pelo Vite:

```
meu-app-react/
├── node_modules/
├── public/
│   └── vite.svg
├── src/
│   ├── assets/
│   │   └── react.svg
│   ├── App.css
│   ├── App.tsx
│   ├── index.css
│   ├── main.tsx
│   └── vite-env.d.ts
├── .eslintrc.cjs
├── .gitignore
├── index.html
├── package.json
├── package-lock.json
├── README.md
├── tsconfig.json
├── tsconfig.node.json
└── vite.config.ts
```

Configuração do Vite (vite.config.ts):

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
  server: {
    port: 3000,
    open: true
  },
  resolve: {
    alias: {
      '@': '/src'
    }
  }
})
```

Exemplo de Componente React com TypeScript:

```
// src/components/Greeting.tsx
import React, { useState } from 'react';

interface GreetingProps {
  name: string;
  initialCount?: number;
}

const Greeting: React.FC<GreetingProps> = ({ name, initialCount = 0 }) => {
  const [count, setCount] = useState<number>(initialCount);

  return (
    <div>
      <h1>Olá, {name}!</h1>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Incrementar
      </button>
    </div>
  );
};

export default Greeting;
```

Usando o Componente:

```tsx
// src/App.tsx
import React from 'react';
import './App.css';
import Greeting from './components/Greeting';

function App() {
  return (
    <div className="App">
      <Greeting name="Mundo" initialCount={5} />
    </div>
  );
}

export default App;
```

Vantagens do Vite sobre Create React App:

1. **Velocidade**: Inicialização e recarregamento muito mais rápidos
2. **ESM Nativo**: Usa ES modules nativamente durante o desenvolvimento
3. **Configuração Flexível**: Mais fácil de personalizar
4. **Suporte Moderno**: Focado em navegadores modernos
5. **Tamanho do Pacote**: Geralmente produz bundles menores
6. **Plugins**: Sistema de plugins poderoso e extensível

2.4 Componentes React com TypeScript

Tipos de Componentes:

1. **Componentes Funcionais:**

```tsx
// Componente funcional básico
const Button: React.FC = () => {
  return <button>Clique-me</button>;
};

// Componente funcional com props
interface ButtonProps {
  text: string;
  onClick: () => void;
  disabled?: boolean;
}

const Button: React.FC<ButtonProps> = ({ text, onClick, disabled = false }) => {
  return (
    <button onClick={onClick} disabled={disabled}>
      {text}
    </button>
  );
};
```

2. **Componentes de Classe:**

```tsx
interface CounterProps {
  initialValue: number;
}

interface CounterState {
  count: number;
}

class Counter extends React.Component<CounterProps, CounterState> {
  constructor(props: CounterProps) {
    super(props);
    this.state = {
      count: props.initialValue
    };
  }

  increment = () => {
    this.setState(prevState => ({
      count: prevState.count + 1
    }));
  };

  render() {
    return (
      <div>
        <p>Contagem: {this.state.count}</p>
        <button onClick={this.increment}>Incrementar</button>
      </div>
    );
  }
}
```

Tipagem de Props Avançada:

```tsx
// Props com children
interface CardProps {
  title: string;
  children: React.ReactNode;
}

const Card: React.FC<CardProps> = ({ title, children }) => {
  return (
    <div className="card">
      <h2>{title}</h2>
      <div className="card-content">{children}</div>
    </div>
  );
};

// Props com funções específicas
interface InputProps {
  value: string;
  onChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  onBlur?: (e: React.FocusEvent<HTMLInputElement>) => void;
}

const Input: React.FC<InputProps> = ({ value, onChange, onBlur }) => {
  return <input value={value} onChange={onChange} onBlur={onBlur} />;
};

// Props com union types
type ButtonVariant = 'primary' | 'secondary' | 'danger';

interface ButtonProps {
  text: string;
  variant: ButtonVariant;
  onClick: () => void;
}

const Button: React.FC<ButtonProps> = ({ text, variant, onClick }) => {
  return (
    <button className={`btn btn-${variant}`} onClick={onClick}>
      {text}
    </button>
  );
};
```

Componentes Genéricos:

```tsx
interface ListProps<T> {
  items: T[];
  renderItem: (item: T) => React.ReactNode;
}

function List<T>({ items, renderItem }: ListProps<T>) {
  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{renderItem(item)}</li>
      ))}
    </ul>
  );
}

// Uso
interface User {
  id: number;
  name: string;
}

const users: User[] = [
  { id: 1, name: 'Alice' },
  { id: 2, name: 'Bob' }
];

<List
  items={users}
  renderItem={(user) => <span>{user.name}</span>}
/>;
```

2.5 Hooks com TypeScript

useState:

```tsx
// Tipo inferido
const [count, setCount] = useState(0);

// Tipo explícito
const [count, setCount] = useState<number>(0);

// Com tipo complexo
interface User {
  id: number;
  name: string;
  email: string;
}

const [user, setUser] = useState<User | null>(null);

// Com tipo union
const [status, setStatus] = useState<'idle' | 'loading' | 'success' | 'error'>('idle');
```

useEffect:

```
// Efeito básico
useEffect(() => {
  document.title = `Contagem: ${count}`;
}, [count]);

// Com limpeza
useEffect(() => {
  const timer = setInterval(() => {
    setCount(c => c + 1);
  }, 1000);

  // Função de limpeza
  return () => clearInterval(timer);
}, []);

// Com async/await
useEffect(() => {
  const fetchData = async () => {
    try {
      setStatus('loading');
      const response = await fetch('https://api.example.com/data');
      const data: User[] = await response.json();
      setUsers(data);
      setStatus('success');
    } catch (error) {
      setStatus('error');
      setError(error instanceof Error ? error.message : 'Erro desconhecido');
    }
  };

  fetchData();
}, []);
```

useRef:

```tsx
// Ref para elemento DOM
const inputRef = useRef<HTMLInputElement>(null);

// Uso
<input ref={inputRef} type="text" />

// Acessando o elemento
const focusInput = () => {
  inputRef.current?.focus();
};

// Ref para valor mutável
const countRef = useRef<number>(0);

// Atualizando o valor (não causa re-render)
const incrementCount = () => {
  countRef.current += 1;
  console.log(`Contagem atual: ${countRef.current}`);
};
```

useReducer:

```
// Definir tipos de estado e ação
interface State {
  count: number;
  loading: boolean;
  error: string | null;
}

type Action =
  | { type: 'INCREMENT' }
  | { type: 'DECREMENT' }
  | { type: 'RESET'; payload: number }
  | { type: 'SET_LOADING'; payload: boolean }
  | { type: 'SET_ERROR'; payload: string };

// Definir estado inicial
const initialState: State = {
  count: 0,
  loading: false,
  error: null
};

// Implementar reducer
const reducer = (state: State, action: Action): State => {
  switch (action.type) {
    case 'INCREMENT':
      return { ...state, count: state.count + 1 };
    case 'DECREMENT':
      return { ...state, count: state.count - 1 };
    case 'RESET':
      return { ...state, count: action.payload };
    case 'SET_LOADING':
      return { ...state, loading: action.payload };
    case 'SET_ERROR':
      return { ...state, error: action.payload };
    default:
      return state;
  }
};

// Usar o reducer
const [state, dispatch] = useReducer(reducer, initialState);

// Exemplos de uso
dispatch({ type: 'INCREMENT' });
dispatch({ type: 'RESET', payload: 0 });
dispatch({ type: 'SET_LOADING', payload: true });
```

useContext:

```tsx
// Definir tipo do contexto
interface ThemeContextType {
  theme: 'light' | 'dark';
  toggleTheme: () => void;
}

// Criar contexto com valor padrão
const ThemeContext = React.createContext<ThemeContextType | undefined>(undefined);

// Criar provider
const ThemeProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [theme, setTheme] = useState<'light' | 'dark'>('light');

  const toggleTheme = () => {
    setTheme(prevTheme => prevTheme === 'light' ? 'dark' : 'light');
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

// Hook personalizado para usar o contexto
const useTheme = (): ThemeContextType => {
  const context = useContext(ThemeContext);
  if (context === undefined) {
    throw new Error('useTheme must be used within a ThemeProvider');
  }
  return context;
};

// Uso em componente
const ThemedButton: React.FC = () => {
  const { theme, toggleTheme } = useTheme();

  return (
    <button
      onClick={toggleTheme}
      style={{
        backgroundColor: theme === 'light' ? '#fff' : '#333',
        color: theme === 'light' ? '#333' : '#fff'
      }}
    >
      Alternar Tema
    </button>
  );
};
```

Custom Hooks:

```tsx
// Hook personalizado para formulários
interface UseFormProps<T> {
  initialValues: T;
  onSubmit: (values: T) => void;
  validate?: (values: T) => Partial<Record<keyof T, string>>;
}

function useForm<T extends Record<string, any>>({
  initialValues,
  onSubmit,
  validate
}: UseFormProps<T>) {
  const [values, setValues] = useState<T>(initialValues);
  const [errors, setErrors] = useState<Partial<Record<keyof T, string>>>({});
  const [isSubmitting, setIsSubmitting] = useState(false);

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const { name, value } = e.target;
    setValues({
      ...values,
      [name]: value
    });
  };

  const handleSubmit = (e: React.FormEvent<HTMLFormElement>) => {
    e.preventDefault();

    if (validate) {
      const validationErrors = validate(values);
      setErrors(validationErrors);

      if (Object.keys(validationErrors).length === 0) {
        setIsSubmitting(true);
        onSubmit(values);
        setIsSubmitting(false);
      }
    } else {
      setIsSubmitting(true);
      onSubmit(values);
      setIsSubmitting(false);
    }
  };

  return {
    values,
    errors,
    isSubmitting,
    handleChange,
    handleSubmit
  };
}
```

```tsx
// Uso do hook personalizado
interface LoginFormValues {
  email: string;
  password: string;
}

const LoginForm: React.FC = () => {
  const {
    values,
    errors,
    isSubmitting,
    handleChange,
    handleSubmit
  } = useForm<LoginFormValues>({
    initialValues: {
      email: '',
      password: ''
    },
    validate: (values) => {
      const errors: Partial<Record<keyof LoginFormValues, string>> = {};

      if (!values.email) {
        errors.email = 'Email é obrigatório';
      } else if (!/\S+@\S+\.\S+/.test(values.email)) {
        errors.email = 'Email inválido';
      }

      if (!values.password) {
        errors.password = 'Senha é obrigatória';
      } else if (values.password.length < 6) {
        errors.password = 'Senha deve ter pelo menos 6 caracteres';
      }

      return errors;
    },
    onSubmit: (values) => {
      console.log('Form submitted:', values);
      // Lógica de autenticação
    }
  });

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label htmlFor="email">Email</label>
        <input
          type="email"
          id="email"
          name="email"
          value={values.email}
          onChange={handleChange}
```

```
        />
        {errors.email && <p className="error">{errors.email}</p>}
      </div>

      <div>
        <label htmlFor="password">Senha</label>
        <input
          type="password"
          id="password"
          name="password"
          value={values.password}
          onChange={handleChange}
        />
        {errors.password && <p className="error">{errors.password}</p>}
      </div>

      <button type="submit" disabled={isSubmitting}>
        {isSubmitting ? 'Enviando...' : 'Entrar'}
      </button>
    </form>
  );
};
```

2.6 Gerenciamento de Estado

Estado Local com useState:

```
const Counter: React.FC = () => {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Contagem: {count}</p>
      <button onClick={() => setCount(count + 1)}>Incrementar</button>
      <button onClick={() => setCount(count - 1)}>Decrementar</button>
      <button onClick={() => setCount(0)}>Resetar</button>
    </div>
  );
};
```

Estado Complexo com useReducer:

```
interface TodoItem {
  id: number;
  text: string;
  completed: boolean;
}

interface TodoState {
  todos: TodoItem[];
  filter: 'all' | 'active' | 'completed';
}
```

```
type TodoAction =
  | { type: 'ADD_TODO'; payload: string }
  | { type: 'TOGGLE_TODO'; payload: number }
  | { type: 'REMOVE_TODO'; payload: number }
  | { type: 'SET_FILTER'; payload: 'all' | 'active' | 'completed' };

const initialState: TodoState = {
  todos: [],
  filter: 'all'
};

const todoReducer = (state: TodoState, action: TodoAction): TodoState => {
  switch (action.type) {
    case 'ADD_TODO':
      return {
        ...state,
        todos: [
          ...state.todos,
          {
            id: Date.now(),
            text: action.payload,
            completed: false
          }
        ]
      };
    case 'TOGGLE_TODO':
      return {
        ...state,
        todos: state.todos.map(todo =>
          todo.id === action.payload
            ? { ...todo, completed: !todo.completed }
            : todo
        )
      };
    case 'REMOVE_TODO':
      return {
        ...state,
        todos: state.todos.filter(todo => todo.id !== action.payload)
      };
    case 'SET_FILTER':
      return {
        ...state,
        filter: action.payload
      };
    default:
      return state;
  }
};

const TodoApp: React.FC = () => {
  const [state, dispatch] = useReducer(todoReducer, initialState);
```

```
const [text, setText] = useState('');

const handleSubmit = (e: React.FormEvent) => {
  e.preventDefault();
  if (text.trim()) {
    dispatch({ type: 'ADD_TODO', payload: text });
    setText('');
  }
};

const filteredTodos = state.todos.filter(todo => {
  if (state.filter === 'active') return !todo.completed;
  if (state.filter === 'completed') return todo.completed;
  return true;
});

return (
  <div>
    <form onSubmit={handleSubmit}>
      <input
        value={text}
        onChange={e => setText(e.target.value)}
        placeholder="Adicionar tarefa"
      />
      <button type="submit">Adicionar</button>
    </form>

    <div>
      <button onClick={() => dispatch({ type: 'SET_FILTER', payload: 'all' })}>
        Todas
      </button>
      <button onClick={() => dispatch({ type: 'SET_FILTER', payload: 'active' })}>
        Ativas
      </button>
      <button onClick={() => dispatch({ type: 'SET_FILTER', payload: 'completed' })}>
        Concluídas
      </button>
    </div>

    <ul>
      {filteredTodos.map(todo => (
        <li key={todo.id}>
          <span
            style={{
              textDecoration: todo.completed ? 'line-through' : 'none'
            }}
            onClick={() => dispatch({ type: 'TOGGLE_TODO', payload: todo.id })}
          >
            {todo.text}
          </span>
          <button onClick={() => dispatch({ type: 'REMOVE_TODO', payload: todo.id })}>
            X
```

```
          </button>
        </li>
      ))}
    </ul>
  </div>
 );
};
```

Estado Global com Context API:

```
// types.ts
export interface User {
  id: number;
  name: string;
  email: string;
}

export interface AuthState {
  user: User | null;
  isAuthenticated: boolean;
  loading: boolean;
  error: string | null;
}

export type AuthAction =
  | { type: 'LOGIN_REQUEST' }
  | { type: 'LOGIN_SUCCESS'; payload: User }
  | { type: 'LOGIN_FAILURE'; payload: string }
  | { type: 'LOGOUT' };

// authContext.tsx
import React, { createContext, useReducer, useContext } from 'react';
import { AuthState, AuthAction, User } from './types';

const initialState: AuthState = {
  user: null,
  isAuthenticated: false,
  loading: false,
  error: null
};

const authReducer = (state: AuthState, action: AuthAction): AuthState => {
  switch (action.type) {
    case 'LOGIN_REQUEST':
      return {
        ...state,
        loading: true,
        error: null
      };
    case 'LOGIN_SUCCESS':
      return {
        ...state,
```

```
        user: action.payload,
        isAuthenticated: true,
        loading: false,
        error: null
      };
    case 'LOGIN_FAILURE':
      return {
        ...state,
        user: null,
        isAuthenticated: false,
        loading: false,
        error: action.payload
      };
    case 'LOGOUT':
      return {
        ...state,
        user: null,
        isAuthenticated: false,
        loading: false,
        error: null
      };
    default:
      return state;
  }
};

interface AuthContextType {
  state: AuthState;
  login: (email: string, password: string) => Promise<void>;
  logout: () => void;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [state, dispatch] = useReducer(authReducer, initialState);

  const login = async (email: string, password: string) => {
    try {
      dispatch({ type: 'LOGIN_REQUEST' });

      // Simulação de chamada de API
      const response = await new Promise<User>((resolve, reject) => {
        setTimeout(() => {
          if (email === 'user@example.com' && password === 'password') {
            resolve({
              id: 1,
              name: 'Usuário Teste',
              email: 'user@example.com'
            });
          } else {
            reject(new Error('Credenciais inválidas'));
```

```
        }
      }, 1000);
    });

    dispatch({ type: 'LOGIN_SUCCESS', payload: response });
  } catch (error) {
    dispatch({
      type: 'LOGIN_FAILURE',
      payload: error instanceof Error ? error.message : 'Erro desconhecido'
    });
  }
};

const logout = () => {
  dispatch({ type: 'LOGOUT' });
};

return (
  <AuthContext.Provider value={{ state, login, logout }}>
    {children}
  </AuthContext.Provider>
);
};

export const useAuth = (): AuthContextType => {
const context = useContext(AuthContext);
if (context === undefined) {
  throw new Error('useAuth must be used within an AuthProvider');
}
return context;
};

// App.tsx
import React from 'react';
import { AuthProvider } from './authContext';
import LoginPage from './LoginPage';
import Dashboard from './Dashboard';

const App: React.FC = () => {
return (
  <AuthProvider>
    <AppContent />
  </AuthProvider>
);
};

const AppContent: React.FC = () => {
const { state } = useAuth();

return (
  <div>
    {state.isAuthenticated ? <Dashboard /> : <LoginPage />}
```

```
    </div>
  );
};

// LoginPage.tsx
import React, { useState } from 'react';
import { useAuth } from './authContext';

const LoginPage: React.FC = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const { state, login } = useAuth();

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    await login(email, password);
  };

  return (
    <div>
      <h1>Login</h1>
      <form onSubmit={handleSubmit}>
        <div>
          <label htmlFor="email">Email</label>
          <input
            type="email"
            id="email"
            value={email}
            onChange={e => setEmail(e.target.value)}
            required
          />
        </div>
        <div>
          <label htmlFor="password">Senha</label>
          <input
            type="password"
            id="password"
            value={password}
            onChange={e => setPassword(e.target.value)}
            required
          />
        </div>
        <button type="submit" disabled={state.loading}>
          {state.loading ? 'Entrando...' : 'Entrar'}
        </button>
        {state.error && <p className="error">{state.error}</p>}
      </form>
    </div>
  );
};

// Dashboard.tsx
```

```
import React from 'react';
import { useAuth } from './authContext';

const Dashboard: React.FC = () => {
  const { state, logout } = useAuth();

  return (
    <div>
      <h1>Dashboard</h1>
      <p>Bem-vindo, {state.user?.name}!</p>
      <button onClick={logout}>Sair</button>
    </div>
  );
};
```

Bibliotecas de Gerenciamento de Estado:

1. **Redux com TypeScript:**

```
// store/types.ts
export interface Todo {
  id: number;
  text: string;
  completed: boolean;
}

export interface TodoState {
  todos: Todo[];
  loading: boolean;
  error: string | null;
}

// store/actions.ts
import { Todo } from './types';

export enum ActionTypes {
  ADD_TODO = 'ADD_TODO',
  TOGGLE_TODO = 'TOGGLE_TODO',
  REMOVE_TODO = 'REMOVE_TODO',
  FETCH_TODOS_REQUEST = 'FETCH_TODOS_REQUEST',
  FETCH_TODOS_SUCCESS = 'FETCH_TODOS_SUCCESS',
  FETCH_TODOS_FAILURE = 'FETCH_TODOS_FAILURE'
}

interface AddTodoAction {
  type: ActionTypes.ADD_TODO;
  payload: string;
}

interface ToggleTodoAction {
  type: ActionTypes.TOGGLE_TODO;
  payload: number;
```

```
}

interface RemoveTodoAction {
  type: ActionTypes.REMOVE_TODO;
  payload: number;
}

interface FetchTodosRequestAction {
  type: ActionTypes.FETCH_TODOS_REQUEST;
}

interface FetchTodosSuccessAction {
  type: ActionTypes.FETCH_TODOS_SUCCESS;
  payload: Todo[];
}

interface FetchTodosFailureAction {
  type: ActionTypes.FETCH_TODOS_FAILURE;
  payload: string;
}

export type TodoActionTypes =
  | AddTodoAction
  | ToggleTodoAction
  | RemoveTodoAction
  | FetchTodosRequestAction
  | FetchTodosSuccessAction
  | FetchTodosFailureAction;

// Action creators
export const addTodo = (text: string): AddTodoAction => ({
  type: ActionTypes.ADD_TODO,
  payload: text
});

export const toggleTodo = (id: number): ToggleTodoAction => ({
  type: ActionTypes.TOGGLE_TODO,
  payload: id
});

export const removeTodo = (id: number): RemoveTodoAction => ({
  type: ActionTypes.REMOVE_TODO,
  payload: id
});

export const fetchTodosRequest = (): FetchTodosRequestAction => ({
  type: ActionTypes.FETCH_TODOS_REQUEST
});

export const fetchTodosSuccess = (todos: Todo[]): FetchTodosSuccessAction => ({
  type: ActionTypes.FETCH_TODOS_SUCCESS,
  payload: todos
```

```
});

export const fetchTodosFailure = (error: string): FetchTodosFailureAction => ({
  type: ActionTypes.FETCH_TODOS_FAILURE,
  payload: error
});

// store/reducers.ts
import { TodoState, Todo } from './types';
import { TodoActionTypes, ActionTypes } from './actions';

const initialState: TodoState = {
  todos: [],
  loading: false,
  error: null
};

export const todoReducer = (
  state = initialState,
  action: TodoActionTypes
): TodoState => {
  switch (action.type) {
    case ActionTypes.ADD_TODO:
      return {
        ...state,
        todos: [
          ...state.todos,
          {
            id: Date.now(),
            text: action.payload,
            completed: false
          }
        ]
      };
    case ActionTypes.TOGGLE_TODO:
      return {
        ...state,
        todos: state.todos.map(todo =>
          todo.id === action.payload
            ? { ...todo, completed: !todo.completed }
            : todo
        )
      };
    case ActionTypes.REMOVE_TODO:
      return {
        ...state,
        todos: state.todos.filter(todo => todo.id !== action.payload)
      };
    case ActionTypes.FETCH_TODOS_REQUEST:
      return {
        ...state,
        loading: true,
```

```
      error: null
    };
  case ActionTypes.FETCH_TODOS_SUCCESS:
    return {
      ...state,
      loading: false,
      todos: action.payload,
      error: null
    };
  case ActionTypes.FETCH_TODOS_FAILURE:
    return {
      ...state,
      loading: false,
      error: action.payload
    };
  default:
    return state;
 }
};
```

2. **Zustand:**

```
import create from 'zustand';

interface Todo {
 id: number;
 text: string;
 completed: boolean;
}

interface TodoStore {
 todos: Todo[];
 loading: boolean;
 error: string | null;
 addTodo: (text: string) => void;
 toggleTodo: (id: number) => void;
 removeTodo: (id: number) => void;
 fetchTodos: () => Promise<void>;
}

export const useTodoStore = create<TodoStore>((set, get) => ({
 todos: [],
 loading: false,
 error: null,

 addTodo: (text: string) => {
  set(state => ({
   todos: [
    ...state.todos,
    {
     id: Date.now(),
     text,
```

```
        completed: false
      }
    ]
  }));
 },

 toggleTodo: (id: number) => {
  set(state => ({
   todos: state.todos.map(todo =>
    todo.id === id ? { ...todo, completed: !todo.completed } : todo
   )
  }));
 },

 removeTodo: (id: number) => {
  set(state => ({
   todos: state.todos.filter(todo => todo.id !== id)
  }));
 },

 fetchTodos: async () => {
  set({ loading: true, error: null });

  try {
   // Simulação de API
   const response = await fetch('https://jsonplaceholder.typicode.com/todos?_limit=5');

   if (!response.ok) {
    throw new Error('Failed to fetch todos');
   }

   const data = await response.json();

   const todos: Todo[] = data.map((item: any) => ({
    id: item.id,
    text: item.title,
    completed: item.completed
   }));

   set({ todos, loading: false });
  } catch (error) {
   set({
    error: error instanceof Error ? error.message : 'An unknown error occurred',
    loading: false
   });
  }
 }
}));

// Uso em componente
const TodoList: React.FC = () => {
 const { todos, loading, error, addTodo, toggleTodo, removeTodo, fetchTodos } = useTodoStore();
```

```
  const [text, setText] = useState('');

  useEffect(() => {
    fetchTodos();
  }, [fetchTodos]);

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (text.trim()) {
      addTodo(text);
      setText('');
    }
  };

  if (loading) return <p>Carregando...</p>;
  if (error) return <p>Erro: {error}</p>;

  return (
    <div>
      <form onSubmit={handleSubmit}>
        <input
          value={text}
          onChange={e => setText(e.target.value)}
          placeholder="Adicionar tarefa"
        />
        <button type="submit">Adicionar</button>
      </form>

      <ul>
        {todos.map(todo => (
          <li key={todo.id}>
            <span
              style={{
                textDecoration: todo.completed ? 'line-through' : 'none'
              }}
              onClick={() => toggleTodo(todo.id)}
            >
              {todo.text}
            </span>
            <button onClick={() => removeTodo(todo.id)}>X</button>
          </li>
        ))}
      </ul>
    </div>
  );
};
```

2.7 Roteamento com React Router

Configuração Básica:

```tsx
// Instalação
// npm install react-router-dom @types/react-router-dom

import React from 'react';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

// Componentes de página
const Home: React.FC = () => <h1>Página Inicial</h1>;
const About: React.FC = () => <h1>Sobre</h1>;
const Contact: React.FC = () => <h1>Contato</h1>;
const NotFound: React.FC = () => <h1>Página não encontrada</h1>;

// Componente de navegação
const Navigation: React.FC = () => (
  <nav>
    <ul>
      <li><Link to="/">Início</Link></li>
      <li><Link to="/about">Sobre</Link></li>
      <li><Link to="/contact">Contato</Link></li>
    </ul>
  </nav>
);

// Componente principal
const App: React.FC = () => {
  return (
    <BrowserRouter>
      <div>
        <Navigation />
        <hr />
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/contact" element={<Contact />} />
          <Route path="*" element={<NotFound />} />
        </Routes>
      </div>
    </BrowserRouter>
  );
};

export default App;
```

Rotas com Parâmetros:

```tsx
import React from 'react';
import { BrowserRouter, Routes, Route, Link, useParams } from 'react-router-dom';

// Interface para parâmetros
interface UserParams {
  id: string;
}
```

```tsx
// Componente que usa parâmetros
const UserProfile: React.FC = () => {
  const { id } = useParams<UserParams>();

  return (
    <div>
      <h1>Perfil do Usuário</h1>
      <p>ID do usuário: {id}</p>
    </div>
  );
};

// Lista de usuários
const UserList: React.FC = () => {
  const users = [
    { id: 1, name: 'Alice' },
    { id: 2, name: 'Bob' },
    { id: 3, name: 'Charlie' }
  ];

  return (
    <div>
      <h1>Usuários</h1>
      <ul>
        {users.map(user => (
          <li key={user.id}>
            <Link to={`/users/${user.id}`}>{user.name}</Link>
          </li>
        ))}
      </ul>
    </div>
  );
};

// Componente principal
const App: React.FC = () => {
  return (
    <BrowserRouter>
      <div>
        <nav>
          <ul>
            <li><Link to="/">Início</Link></li>
            <li><Link to="/users">Usuários</Link></li>
          </ul>
        </nav>
        <hr />
        <Routes>
          <Route path="/" element={<h1>Página Inicial</h1>} />
          <Route path="/users" element={<UserList />} />
          <Route path="/users/:id" element={<UserProfile />} />
          <Route path="*" element={<h1>Página não encontrada</h1>} />
```

```
      </Routes>
    </div>
  </BrowserRouter>
 );
};
```

Rotas Aninhadas:

```tsx
import React from 'react';
import { BrowserRouter, Routes, Route, Link, Outlet } from 'react-router-dom';

// Layout para o dashboard
const DashboardLayout: React.FC = () => {
 return (
  <div>
   <h1>Dashboard</h1>
   <nav>
    <ul>
     <li><Link to="/dashboard">Visão Geral</Link></li>
     <li><Link to="/dashboard/stats">Estatísticas</Link></li>
     <li><Link to="/dashboard/settings">Configurações</Link></li>
    </ul>
   </nav>
   <hr />
   <Outlet /> {/* Renderiza as rotas filhas */}
  </div>
 );
};

// Componentes para as rotas do dashboard
const DashboardOverview: React.FC = () => <h2>Visão Geral do Dashboard</h2>;
const DashboardStats: React.FC = () => <h2>Estatísticas</h2>;
const DashboardSettings: React.FC = () => <h2>Configurações</h2>;

// Componente principal
const App: React.FC = () => {
 return (
  <BrowserRouter>
   <div>
    <nav>
     <ul>
      <li><Link to="/">Início</Link></li>
      <li><Link to="/dashboard">Dashboard</Link></li>
     </ul>
    </nav>
    <hr />
    <Routes>
     <Route path="/" element={<h1>Página Inicial</h1>} />
     <Route path="/dashboard" element={<DashboardLayout />}>
      <Route index element={<DashboardOverview />} />
      <Route path="stats" element={<DashboardStats />} />
      <Route path="settings" element={<DashboardSettings />} />
     </Route>
     <Route path="*" element={<h1>Página não encontrada</h1>} />
    </Routes>
   </div>
  </BrowserRouter>
 );
};
```

Navegação Programática:

```
import React from 'react';
import { BrowserRouter, Routes, Route, useNavigate } from 'react-router-dom';

// Componente de login
const Login: React.FC = () => {
  const navigate = useNavigate();
  const [username, setUsername] = React.useState('');
  const [password, setPassword] = React.useState('');

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();

    // Simulação de autenticação
    if (username === 'admin' && password === 'password') {
      // Armazenar token ou estado de autenticação
      localStorage.setItem('isAuthenticated', 'true');

      // Redirecionar para o dashboard
      navigate('/dashboard');
    } else {
      alert('Credenciais inválidas');
    }
  };

  return (
    <div>
      <h1>Login</h1>
      <form onSubmit={handleSubmit}>
        <div>
          <label htmlFor="username">Usuário:</label>
          <input
            type="text"
            id="username"
            value={username}
            onChange={e => setUsername(e.target.value)}
            required
          />
        </div>
        <div>
          <label htmlFor="password">Senha:</label>
          <input
            type="password"
            id="password"
            value={password}
            onChange={e => setPassword(e.target.value)}
            required
          />
        </div>
        <button type="submit">Entrar</button>
      </form>
```

```
    </div>
  );
};

// Componente de dashboard
const Dashboard: React.FC = () => {
  const navigate = useNavigate();

  const handleLogout = () => {
    // Remover token ou estado de autenticação
    localStorage.removeItem('isAuthenticated');

    // Redirecionar para a página de login
    navigate('/login');
  };

  return (
    <div>
      <h1>Dashboard</h1>
      <p>Bem-vindo ao dashboard!</p>
      <button onClick={handleLogout}>Sair</button>
    </div>
  );
};

// Componente principal
const App: React.FC = () => {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<h1>Página Inicial</h1>} />
        <Route path="/login" element={<Login />} />
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="*" element={<h1>Página não encontrada</h1>} />
      </Routes>
    </BrowserRouter>
  );
};
```

Proteção de Rotas:

```
import React from 'react';
import { BrowserRouter, Routes, Route, Navigate, useLocation } from 'react-router-dom';

// Hook para verificar autenticação
const useAuth = () => {
  const [isAuthenticated, setIsAuthenticated] = React.useState<boolean>(() => {
    return localStorage.getItem('isAuthenticated') === 'true';
  });

  const login = (username: string, password: string): boolean => {
    if (username === 'admin' && password === 'password') {
```

```
      localStorage.setItem('isAuthenticated', 'true');
      setIsAuthenticated(true);
      return true;
    }
    return false;
  };

  const logout = () => {
    localStorage.removeItem('isAuthenticated');
    setIsAuthenticated(false);
  };

  return { isAuthenticated, login, logout };
};

// Contexto de autenticação
const AuthContext = React.createContext<ReturnType<typeof useAuth> | null>(null);

// Provider de autenticação
const AuthProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const auth = useAuth();

  return (
    <AuthContext.Provider value={auth}>
      {children}
    </AuthContext.Provider>
  );
};

// Hook para usar o contexto de autenticação
const useAuthContext = () => {
  const context = React.useContext(AuthContext);
  if (!context) {
    throw new Error('useAuthContext must be used within an AuthProvider');
  }
  return context;
};

// Componente para proteger rotas
interface ProtectedRouteProps {
  children: React.ReactNode;
}

const ProtectedRoute: React.FC<ProtectedRouteProps> = ({ children }) => {
  const { isAuthenticated } = useAuthContext();
  const location = useLocation();

  if (!isAuthenticated) {
    // Redirecionar para login, preservando a rota original
    return <Navigate to="/login" state={{ from: location }} replace />;
  }
```

```
  return <>{children}</>;
};

// Componente de login
const Login: React.FC = () => {
  const { login } = useAuthContext();
  const location = useLocation();
  const [username, setUsername] = React.useState('');
  const [password, setPassword] = React.useState('');
  const [error, setError] = React.useState('');

  // Obter a rota original
  const from = (location.state as any)?.from?.pathname || '/dashboard';

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    setError('');

    const success = login(username, password);

    if (!success) {
      setError('Credenciais inválidas');
    }
  };

  // Se autenticado, redirecionar para a rota original
  const { isAuthenticated } = useAuthContext();
  if (isAuthenticated) {
    return <Navigate to={from} replace />;
  }

  return (
    <div>
      <h1>Login</h1>
      {error && <p style={{ color: 'red' }}>{error}</p>}
      <form onSubmit={handleSubmit}>
        <div>
          <label htmlFor="username">Usuário:</label>
          <input
            type="text"
            id="username"
            value={username}
            onChange={e => setUsername(e.target.value)}
            required
          />
        </div>
        <div>
          <label htmlFor="password">Senha:</label>
          <input
            type="password"
            id="password"
            value={password}
```

```
          onChange={e => setPassword(e.target.value)}
          required
        />
      </div>
      <button type="submit">Entrar</button>
    </form>
  </div>
 );
};

// Componente de dashboard
const Dashboard: React.FC = () => {
 const { logout } = useAuthContext();

 return (
  <div>
   <h1>Dashboard</h1>
   <p>Bem-vindo ao dashboard!</p>
   <button onClick={logout}>Sair</button>
  </div>
 );
};

// Componente principal
const App: React.FC = () => {
 return (
  <AuthProvider>
   <BrowserRouter>
    <Routes>
     <Route path="/" element={<h1>Página Inicial</h1>} />
     <Route path="/login" element={<Login />} />
     <Route
      path="/dashboard"
      element={
       <ProtectedRoute>
        <Dashboard />
       </ProtectedRoute>
      }
     />
     <Route path="*" element={<h1>Página não encontrada</h1>} />
    </Routes>
   </BrowserRouter>
  </AuthProvider>
 );
};
```

Recursos Adicionais

Documentação Oficial

- Docker Documentation (https://docs.docker.com/)

- React Documentation (https://reactjs.org/docs/getting-started.html)
- TypeScript Documentation (https://www.typescriptlang.org/docs/)
- Vite Documentation (https://vitejs.dev/guide/)
- React Router Documentation (https://reactrouter.com/docs/en/v6)

Tutoriais e Cursos

- Docker for Beginners (https://docker-curriculum.com/)
- React TypeScript Cheatsheet (https://react-typescript-cheatsheet.netlify.app/)
- Full Stack Open (https://fullstackopen.com/) - Curso gratuito sobre React, TypeScript e mais

Ferramentas

- Docker Hub (https://hub.docker.com/) - Repositório de imagens Docker
- Docker Compose Explorer (https://github.com/microsoft/vscode-docker) - Extensão VS Code para Docker
- React Developer Tools (https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi) - Extensão para Chrome/Firefox
- ESLint (https://eslint.org/) - Linter para JavaScript/TypeScript
- Prettier (https://prettier.io/) - Formatador de código

Exercícios Práticos

1. **Configuração de Ambiente Docker:**
    - Crie um Dockerfile para uma aplicação Node.js
    - Configure um arquivo docker-compose.yml com múltiplos serviços (frontend, backend, banco de dados)
    - Implemente volumes para persistência de dados
    - Configure variáveis de ambiente para diferentes ambientes (dev, prod)
2. **Aplicação React com TypeScript:**
    - Crie um novo projeto React com TypeScript usando Vite
    - Implemente componentes funcionais com props tipadas
    - Utilize hooks (useState, useEffect, useContext)
    - Crie pelo menos um custom hook reutilizável
3. **Gerenciamento de Estado:**
    - Implemente gerenciamento de estado local com useState e useReducer
    - Crie um contexto global para compartilhar estado entre componentes
    - Adicione persistência de estado usando localStorage
    - Implemente carregamento assíncrono de dados de uma API
4. **Roteamento e Navegação:**
    - Configure React Router com múltiplas rotas
    - Implemente rotas aninhadas e layouts compartilhados
    - Adicione proteção de rotas com autenticação
    - Implemente redirecionamentos e navegação programática
5. **Projeto Completo:**

- Crie uma aplicação CRUD completa (Create, Read, Update, Delete)
- Utilize Docker para configurar o ambiente de desenvolvimento
- Implemente autenticação e autorização
- Adicione testes unitários para componentes e hooks
- Configure um pipeline de CI/CD com GitHub Actions