Arquitetura Limpa e TypeScript: Conceitos e Exemplos Práticos

Introdução

Este documento aborda os princípios da Arquitetura Limpa, proposta por Robert C. Martin ("Uncle Bob"), e como aplicá-los em projetos TypeScript para criar sistemas de software mais flexíveis, testáveis e fáceis de manter.

Arquitetura Limpa: Uma Visão Geral

A Arquitetura Limpa é um conjunto de princípios e práticas que visam separar as diferentes responsabilidades de um sistema de software em camadas distintas, cada uma com suas próprias regras e dependências. Essa abordagem promove a independência entre as camadas, tornando o sistema mais fácil de entender, modificar e testar.

Princípios Fundamentais da Arquitetura Limpa

Independência de Frameworks: A arquitetura não deve depender de detalhes de implementação de frameworks externos.
Testável: As regras de negócio devem ser testáveis independentemente de UI, banco de dados, frameworks web ou qualquer outro elemento externo.
Independência de UI: A UI pode ser facilmente alterada sem impactar o restante do sistema.
Independência de Banco de Dados: As regras de negócio podem ser testadas sem a necessidade de um banco de dados real.
Independência de Qualquer Agente Externo: As regras de negócio não devem conhecer detalhes sobre o mundo exterior.

Arquitetura Limpa e TypeScript: Conceitos e Exemplos Práticos

Camadas da Arquitetura Limpa

Entidades: Representam os objetos de negócio do sistema e encapsulam as regras de negócio mais importantes.

Casos de Uso: Orquestram o fluxo de dados entre as entidades e definem as operações que o sistema pode realizar.

Adaptadores de Interface: Convertem os dados da forma mais conveniente para as entidades e casos de uso (e vice-versa) para interagir com o mundo externo, como UI, banco de dados, etc.

Frameworks e Drivers: São as ferramentas e bibliotecas externas utilizadas pelo sistema, como frameworks web, ORMs, etc.

Regra da Dependência

As dependências devem sempre apontar para dentro, em direção às camadas mais internas. Isso significa que as camadas externas (adaptadores de interface, frameworks e drivers) dependem das camadas internas (casos de uso e entidades), mas nunca o contrário.

Exemplos Práticos com TypeScript

1. Criando uma Entidade

```typescript
// entities/user.ts
export class User {
```

```
  constructor(
    public id: string,
    public name: string,
    public email: string
  ) {}
}
```

2. Definindo um Caso de Uso

```typescript
// usecases/get-user-by-id.ts
import { User } from "../entities/user";
import { UserRepository } from "../repositories/user-repository";

export class GetUserByIdUseCase {
  constructor(private userRepository: UserRepository) {}

  async execute(userId: string): Promise<User> {
    return await this.userRepository.getById(userId);
  }
}
```

3. Implementando um Repositório (Adaptador de Interface)

```typescript
// repositories/user-repository.ts
import { User } from "../entities/user";

export interface UserRepository {
  getById(userId: string): Promise<User>;
}

// Implementação com banco de dados (por exemplo, usando TypeORM)
export class TypeORMUserRepository implements UserRepository {
  async getById(userId: string): Promise<User> {
    // código para obter usuário do banco de dados
  }
}
```

4. Criando um Controlador (Adaptador de Interface)

```typescript
// controllers/get-user-by-id-controller.ts
import { Request, Response } from "express";
import { GetUserByIdUseCase } from "../usecases/get-user-by-id";
```

```
import { TypeORMUserRepository } from "../repositories/user-repository";

export class GetUserByIdController {
  async handle(req: Request, res: Response): Promise<Response> {
    const { userId } = req.params;

    const userRepository = new TypeORMUserRepository(); // Injeção de dependência
    const getUserByIdUseCase = new GetUserByIdUseCase(userRepository);

    const user = await getUserByIdUseCase.execute(userId);
    return res.json(user);
  }
}
```

Conclusão

A Arquitetura Limpa, combinada com o uso de TypeScript, oferece uma base sólida para construir sistemas de software robustos, escaláveis e fáceis de manter. Ao seguir os princípios e exemplos apresentados neste documento, você estará no caminho certo para criar aplicações de alta qualidade que atendem às necessidades do seu negócio.