UniFil
EDUCAÇÃO A DISTÂNCIA

Unidade 4

Sistemas baseados em componentes e prática web

Introdução

A grande demanda por softwares fez com que a forma de se desenvolver sistemas mudasse e que fosse de fácil acesso, com isso surge a necessidade de sistemas web baseados em componentes.

Na aula 01, conhecer os sistemas baseados em componentes e quais são as boas práticas que podem ser utilizadas.

Na aula 02, vamos visualizar códigos utilizando os conceitos de SOLID e como a prática desses conceitos relacionados ao patterns auxiliam no desenvolvimento de sistemas web.

O bom desenvolvimento de um sistema web impacta diretamente no cotidiano dos usuários; sistemas bem desenvolvidos geram valor e lucro para os clientes, além, claro, de satisfazerem o cliente com suas necessidades.

Objetivos

- Conhecer sistemas baseados em componentes.
- Conhecer o conceito de SOLID.
- Compreender o funcionamento de códigos web e sua arquitetura.

Conteúdo programático

Aula 01 — Sistemas baseados em componentes

Aula 02 — SOLID e exemplos de desenvolvimento web

Aula 01 — Sistemas baseados em componentes

Olá, estudante, tudo bem? Vamos começar agora um estudo muito importante de desenvolvimento web. Nesta aula, vamos conhecer como funcionam os sistemas baseados em componentes.

Com a aplicação correta de um sistema baseado em componentes, diversas empresas vêm obtendo resultados de sucesso, pois tratar os dados em locais diferentes faz com que um sistema seja mais preparado para futuras adaptações.

Para descrevermos um sistema baseado em componentes, é preciso relembrar a realidade dos sistemas web atuais e o modelo de arquitetura cliente/servidor.

Videoaula 1

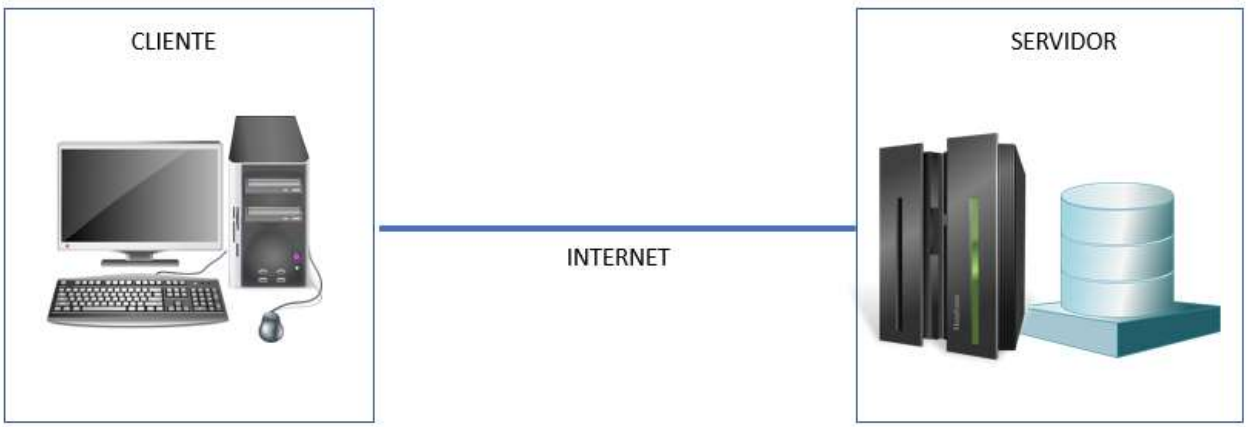
Assista ao vídeo para entender melhor o assunto.

Videoaula 1

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

Figura 1 — Estrutura cliente/servidor



Fonte: O próprio autor (2022).

A figura 1 demonstra a arquitetura cliente/servidor de forma simples, utilizando a internet, porém essa comunicação pode ser realizada via rede e sem internet. De um lado, temos o cliente, que é aquele que envia requisições; e do outro, o servidor, que recebe as requisições, processa e trata os dados de acordo com cada necessidade.

Visualizando esse modelo, é preciso compreender que o funcionamento de um sistema consiste, na verdade, em dois parâmetros diferentes: o cliente e o servidor. Porém, quando se fala de sistemas complexos e grandes, talvez seja necessário quebrar mais ainda esse modelo.

Pode ser que, para que o meu sistema funcione da melhor forma possível, seja preciso criar sistemas menores para cada um realizar operações como “pequenos sistemas”.

Normalmente, grandes sistemas, como aplicativos de bancos e redes sociais, utilizam esse modelo de “pequenos sistemas”. O desenvolvimento desses sistemas é baseado em componentes.

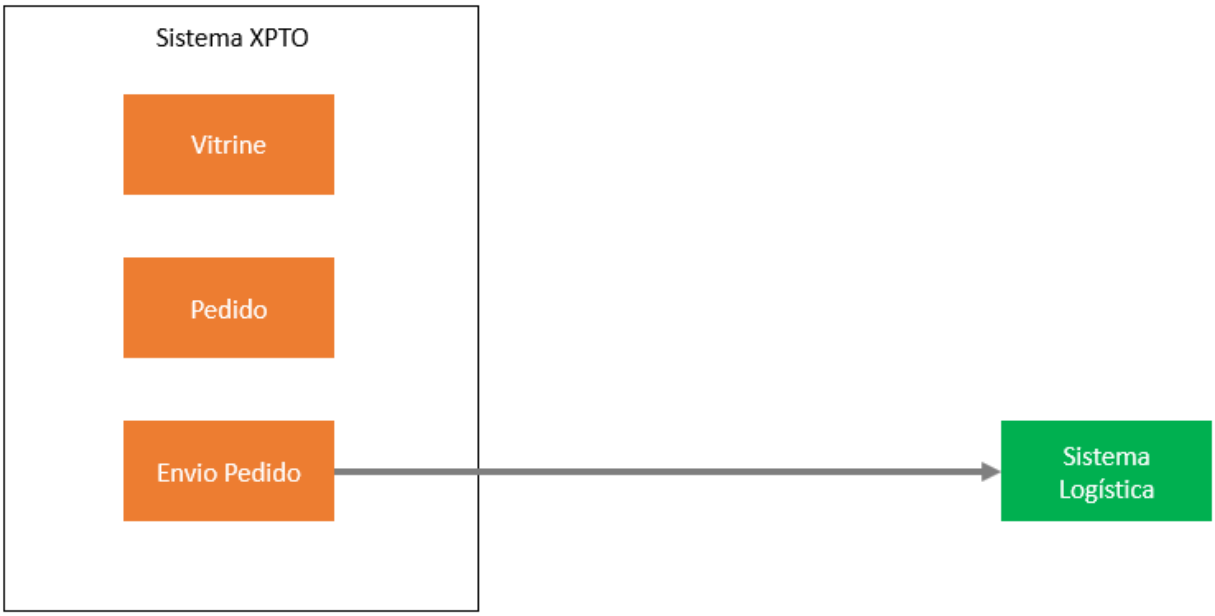
Vamos exercitar um pouco o pensamento em componentes.

Situação-problema

“Eu preciso de um sistema web para vender roupas. Nele, o cliente precisa visualizar as roupas que estou vendendo e depois poder selecionar a que deseja, fazer um pedido e, por fim, pagar o pedido. Após isso, o sistema deve mandá-lo para um sistema de logística que já possuo, para que envie ao cliente as roupas que ele comprou.”

Quando se visualiza essa situação-problema, pensando em construir pequenos sistemas, como ficaria?

Figura 2 — Situação problema minissistemas



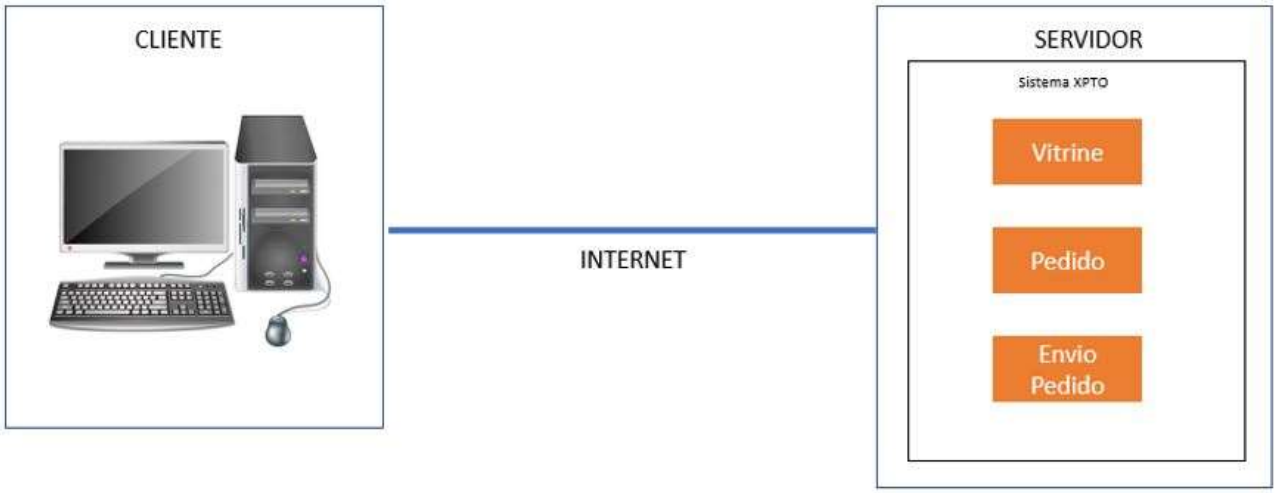
Fonte: O próprio autor (2022).

A figura 2 apresenta a quebra do sistema XPTO, de acordo com a situação-problema apresentada; desse modo, quebrando o sistema em três outros sistemas diferentes, sendo eles:

- Vitrine: espaço que vai retornar ao cliente as roupas que ele pode comprar com os dados: tamanho, modelo, tipo, fabricante etc.
- Pedido: vai cuidar dos dados dos pedidos, seu status, se está em andamento (carrinho), se está pago ou não etc.
- Enviar pedido: após a confirmação do pagamento, para integrar-se com o sistema externo de logística.

Observe que foi quebrado um sistema em minissistemas, em que cada um tem a sua responsabilidade e é separado de acordo com o seu domínio de operação. Olhando para o modelo cliente/servidor, o desenho com minissistemas ficaria diferente.

Figura 3 — Estrutura cliente/servidor/componentes



Fonte: O próprio autor (2022).

A figura 3 apresenta a estrutura cliente/servidor, pensando em componentes e minissistemas. Quando necessário, o cliente envia o request para o minissistema que deseja e recebe o retorno de acordo com os contratos de payload.

Desenvolver sistemas baseados em componentes favorece a agilidade do projeto, tornando mais fácil a construção de sistemas muito complexos, além de favorecer a manutenção dos projetos.

Os minissistemas ou, como alguns chamam, microsserviços são algo que ocorre a nível de back-end, dessa forma são transparentes para o usuário final. Para eles é um sistema só, mas para os desenvolvedores web é preciso ter a clareza de que são códigos em locais diferentes e que constroem o sistema como um todo.

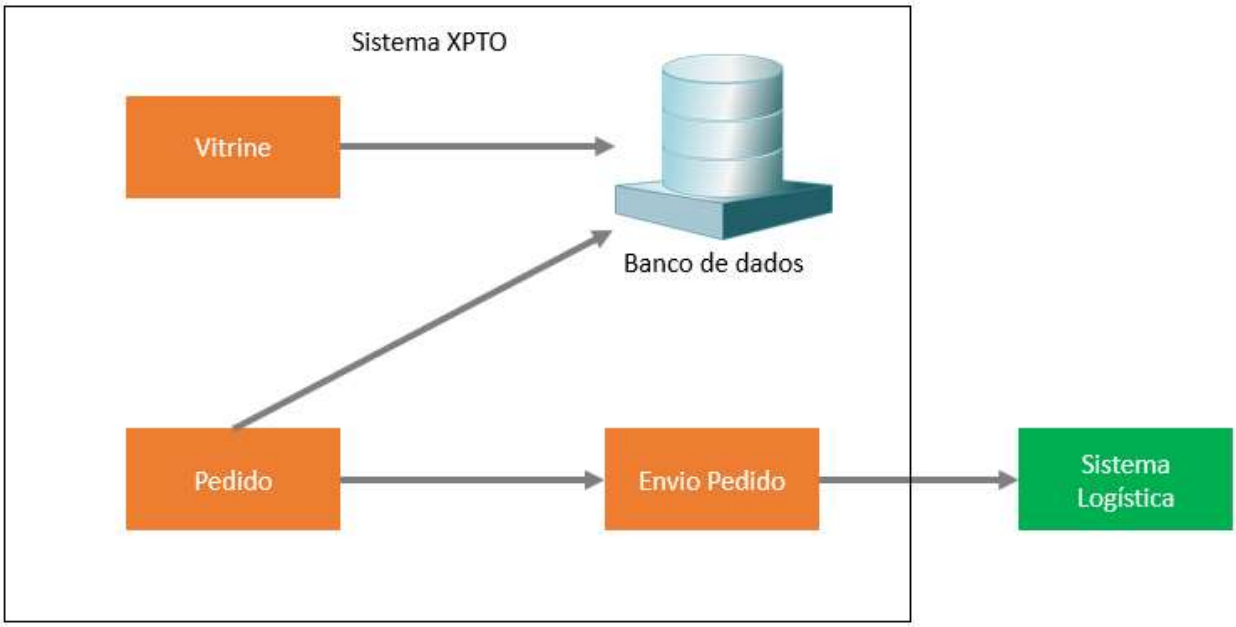
Videoaula 2

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

Porém é preciso se atentar a alguns pontos quando se utiliza o modelo de minissistemas (micro-services).

Figura 4 — Microservices BD único



Fonte: O próprio autor (2022).

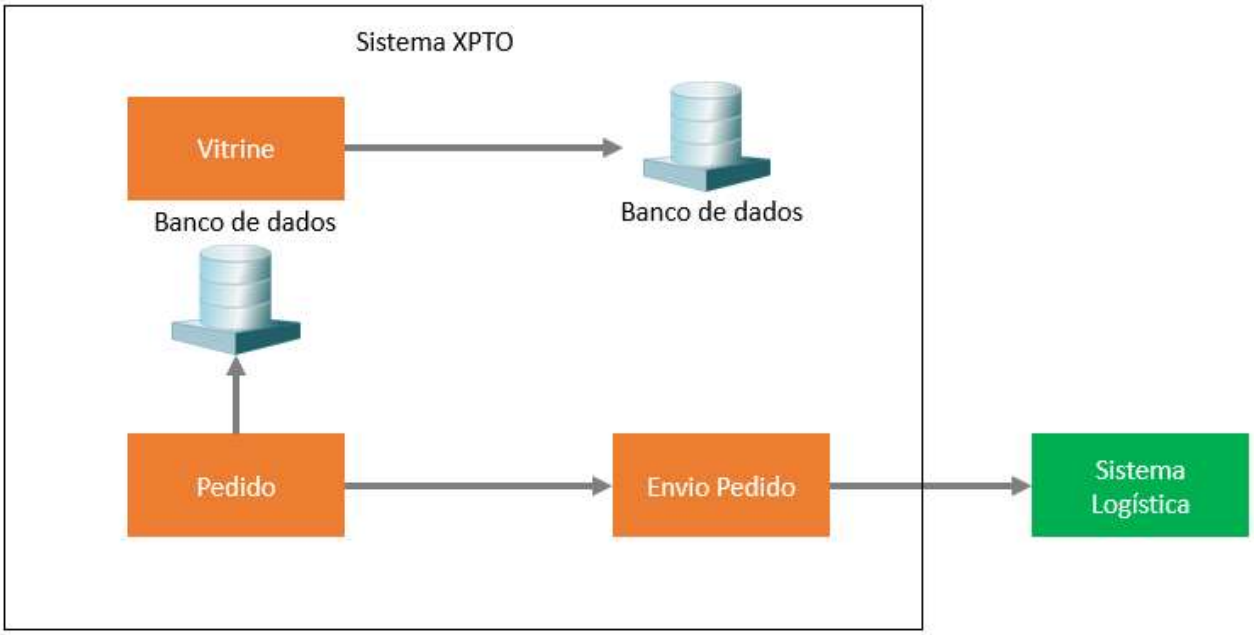
A figura 4 apresenta a comunicação entre os microsserviços, porém usando somente um banco de dados. Esse cenário não é o ideal; pois, se os minissistemas são independentes, não faz sentido utilizarem o mesmo banco de dados.

Quando isso ocorre, os microsserviços param de ser independentes, porque agora ambos precisam ter a conexão com o mesmo banco de dados. A representação da figura 4 é algo que muitas vezes ocorre em sistemas baseados em componentes, por isso é preciso se atentar.

Indicação de Vídeo

Microservices // Dicionário do Programador :(09:50). Disponível em: https://www.youtube.com/watch?v=_2bDOCTnbKc. Acesso em: 30 set. 2022.

Figura 5 — Microservices BD separados



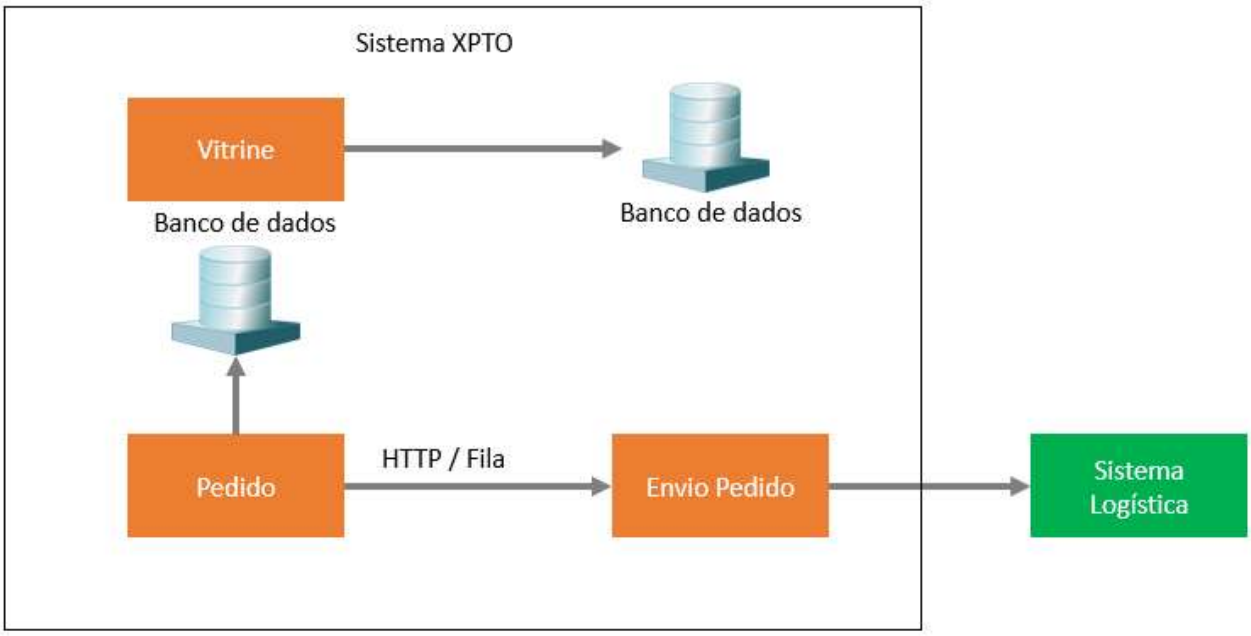
Fonte: O próprio autor (2022).

Observando a figura 5, visualiza-se uma estrutura mais próxima do ideal, que é quando cada microsserviço tem seu banco de dados, com isso dando maior liberdade para cada um deles.

Porém a forma de como montar essa estrutura depende muito da empresa e do quanto irá custar o recurso de possuir mais de um banco de dados, tudo isso normalmente é avaliado juntamente com o cliente que está contratando o desenvolvimento.

É preciso compreender também como funciona a comunicação entre os minissistemas e como eles podem se comunicar de diversas formas, pois é de acordo com essa comunicação que o fluxo de sistema vai se efetivar.

Figura 6 — Comunicação entre microservices



Fonte: O próprio autor (2022).

A figura 6 apresenta o fluxo de comunicação entre os microsserviços pedido e envio. Essa comunicação pode se estabelecer de diversos modos, porém os mais utilizados são HTTP e filas de mensagens.

Quando o sistema precisa ter um comportamento rápido e assíncrono, ou seja, processar diversas requisições ao mesmo tempo, normalmente se utilizam mensagerias para realizar essa comunicação.

Indicação de Leitura

AWS Amazon. Disponível em: https://aws.amazon.com/pt/microservices/. Acesso em: 30 set. 2022.

Até o momento, foram apresentados alguns recursos de sistemas baseados em componentes, porém vamos visualizar algumas vantagens da utilização dessa modelagem de sistemas web:

- Reutilização de dados;
- Independência de domínio;
- Fácil manutenção;
- Mitigação de quedas de sistemas;
- Customização;

- Produtividade;
- Escalabilidade flexível;
- Velocidade;
- Robustez;
- Distribuição dos recursos.

Desenvolvendo sistema no modelo de componentes, distribuem-se os recursos e as responsabilidades, com isso a principal vantagem é a **disponibilidade**, pois, caso ocorra um problema em um minissistema, o outro continua a funcionar normalmente.

Videoaula 3

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

A modelagem baseada em componentes é amplamente utilizada no mercado de tecnologia atual; com a sua boa implementação, um sistema web tem mais qualidade e, com isso, entrega mais rápido recursos para o usuário.

Porém é necessário compreender que, ao depender da internet, é sempre preciso assumir que pode dar errado, então nem sempre que ocorre um erro para o usuário isso é devido ao sistema web, pode ser qualquer tipo de falha na comunicação entre o cliente e o servidor, ou até mesmo entre os microsserviços participantes do sistema.

Aula 02 — Aplicando os conceitos de banco de dados relacional

Olá, estudante, tudo bem? Vamos começar agora um estudo bem diferente sobre desenvolvimento web. Nesta aula, vamos conhecer o conceito de SOLID e compreender exemplos de código web.

A construção de um código web pode ser carregada de informações e até mesmo, em algum momento, se perder durante o desenvolvimento. Com os avanços nas linguagens de programação, a maioria das linguagens utilizadas hoje é Orientada a Objetos (OO). Nesta aula utilizaremos exemplos na linguagem Java.

Visto os erros que ocorreram durante o tempo e a preocupação em sempre desenvolver sistemas da melhor forma possível, foi criado o SOLID, que é um acrônimo para cinco princípios do design OO.

SOLID:

- **S**: Single-Responsiblity Principle (Princípio de Responsabilidade Única).
- **O**: Open-Closed Principle (Princípio Aberto Fechado).
- **L**: Liskov Substitution Principle (Princípio de Substituição de Liskov).
- **I**: Interface Segregation Principle (Princípio de Segregação de Interfaces).
- **D**: Dependency Inversion Principle (Princípio de Inversão de Dependência).

Videoaula 1

Utilize o QR Code para assistir!

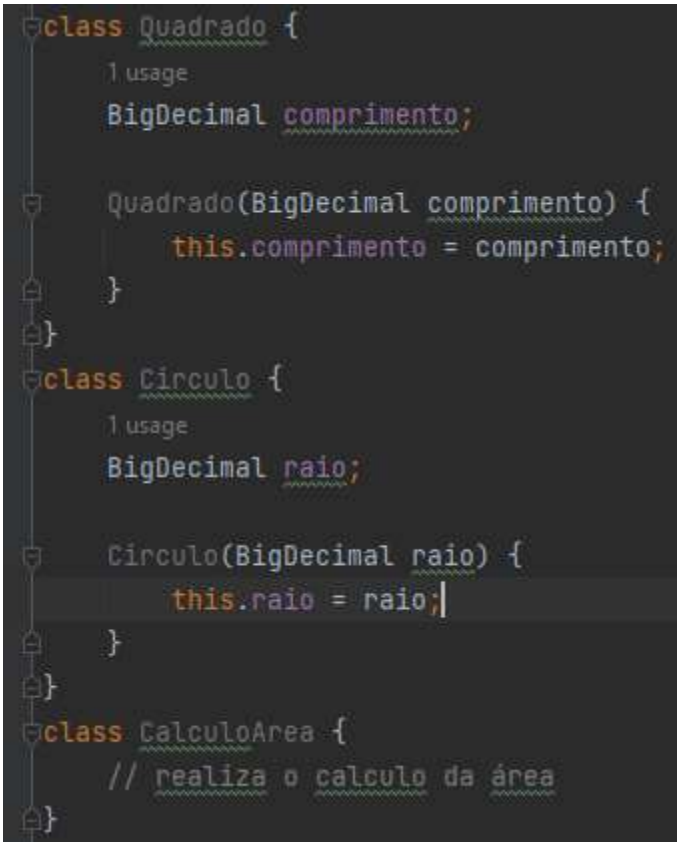
Assista ao vídeo para entender melhor o assunto.

Single-Responsiblity Principle (Princípio de Responsabilidade Única)

Esse princípio defende que uma classe deve ter apenas um motivo para mudar, com isso cada classe deve possuir apenas uma responsabilidade. Ou seja, se o seu objeto realiza mais de uma responsabilidade, você está em desacordo com esse princípio.

Imagine que você precise realizar o cálculo de área de uma forma, com isso você cria os objetos do seu sistema de acordo com as necessidades dele.

Figura 7 — Modelo responsabilidade única



Fonte: O próprio autor (2022).

A figura 7 apresenta como devem ser quebradas as classes de acordo com suas responsabilidades. Nesse caso temos duas formas (quadrado e círculo), cada uma com suas propriedades de acordo com o que precisam, mas o cálculo de área ocorre em outra classe (CalculoArea), pois é outra responsabilidade.

Utilizando esse princípio, a manutenção e correção do código, bem como sua organização, ficam mais fáceis, pois cada classe realiza somente aquilo que lhe convém, com isso os códigos ficam mais limpos e sem muitos métodos gigantes. Alterações podem ser feitas de forma clara e rápida, tornando o desenvolvimento mais ágil.

Open-Closed Principle (Princípio Aberto Fechado)

Esse princípio defende que objetos e entidades devem ser abertos para extensão, mas fechados para modificação, com isso podendo estender uma classe, mas sem modificá-la.

Para seguir esse princípio, normalmente se desenvolvem classes abstratas ou interfaces, para assim manter a integridade das classes, possibilitando que nunca seja modificada a estrutura da classe real.

Figura 8 — Modelo aberto/fechado

```
public abstract class Validacao {
    1 usage  2 implementations
    public abstract boolean isDocumentoValido();
}
class ValidacaoClientePF extends Validacao {

    1 usage
    @Override
    public boolean isDocumentoValido() {
        //implementa validação de acordo com a sua necessidade;
        return false;
    }
}
class ValidacaoClientePJ extends Validacao {

    1 usage
    @Override
    public boolean isDocumentoValido() {
        //implementa validação de acordo com a sua necessidade;
        return true;
    }
}
class AprovacaoDocumento {
    void validacaoDocumento(Validacao validacao) {
        validacao.isDocumentoValido();
    }
}
```

Fonte: O próprio autor (2022).

A figura 8 apresenta um exemplo de como seria o princípio aberto/fechado, em que temos uma classe abstrata chamada Validacao, que é estendida pelas classes ValidacaoClientePF e ValidacaoClientePJ, porém observe que existe também a classe

AprovacaoDocumento que possui um método validacaoDocumento, que recebe como parâmetro a classe abstrata Validacao, com isso essa classe validacaoDocumento está aberta para extensões, porém está fechada para modificação; pois, para adicionar um novo tipo de validação de documento, não é necessário modificar seu método, e sim criar uma nova classe, por exemplo, ValidacaoClienteMEI.

Esse é um exemplo simples e abstrato de como funciona o conceito aberto fechado, proposto pelo SOLID. Sempre que estiver desenvolvendo e visualizar que você vai precisar modificar a estrutura de um método de outra classe, atente-se, pois estará em desacordo com esse princípio.

Liskov Substitution Principle (Princípio de Substituição de Liskov)

Este princípio defende que objetos do tipo X são subtipos de um objeto do tipo Z, logo posso substituir objetos do tipo Z pelo seu subtipo X. Demonstrando assim, pode ser algo confuso, então vamos exemplificar.

Figura 9 — Exemplo de violação do princípio de Liskov

```
public class Veiculo {
    String nome;
    BigDecimal velocidade;
    2 usages  2 overrides
    void ligarMotor() {}
}
class Carro extends Veiculo {
    2 usages
    @Override
    void ligarMotor() {
        super.ligarMotor();
    }
}
class Bicicleta extends Veiculo {
    // Porém bicicleta não tem motor!
    2 usages
    @Override
    void ligarMotor() {
        super.ligarMotor();
    }
}
```

Fonte: O próprio autor (2022).

A figura 9 apresenta uma violação ao princípio de Liskov, pois o subtipo bicicleta não possui motor, dessa forma a construção da herança não atende a esse princípio; pois, caso se substitua o tipo Veiculo pelo subtipo Bicicleta, vamos ter um erro.

Figura 10 — Exemplo do princípio de Liskov

Fonte: O próprio autor (2022).

Analisando a figura 10, encontra-se a implementação da herança, de modo a atender ao princípio de Liskov; para que isso ocorresse, foram criadas duas classes que são subtipos de Veículo e herdadas pelas classes Carro e Bicicleta.

As figuras 9 e 10 apresentam exemplos simples de como refatorar um código aplicando o conceito do princípio da substituição de Liskov, porém é preciso se atentar para

o momento da aplicação desse princípio não acabar tornando o código mais complexo em sistemas web que são muito grandes.

Videoaula 2

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

Interface Segregation Principle (Princípio de Segregação de Interfaces)

A proposta desse princípio é que um cliente (classe que implementa) de uma interface nunca deve ser forçado a implementar algo que não utilize nem deve depender de algum método que não utilize.

Figura 11 — Exemplo de violação da segregação de interfaces

```
interface Impressora {
    void setValor();
    void setNome();
    void setTintaModelo();
    void setTonerModelo();
}
class ImpressoraTinta implements Impressora {

    @Override
    public void setValor() {}

    @Override
    public void setNome() {}

    @Override
    public void setTintaModelo() {}

    @Override
    public void setTonerModelo() {}
}
```

Fonte: O próprio autor (2022).

A figura 11 apresenta a violação do princípio de segregação de interfaces, isso porque a classe ImpressoraTinta é obrigada a implementar o método setTonerModelo, sendo que esse modelo de impressora não possui toner.

A proposta então é segregar em interfaces para que não seja obrigatório implementar um método que não vai ser utilizado e com isso deixar o código mais limpo (clean), ficando mais fácil de realizar manutenções e até mesmo de compreender o funcionamento do código.

Figura 12 — Exemplo de aplicação da segregação de interfaces

```
interface Impressora {
    void setValor();
    void setNome();
}
interface Tinta {
    void setTintaModelo();
}
interface Toner {
    void setTonerModelo();
}
class ImpressoraTinta implements Impressora, Tinta {
    @Override
    public void setValor() {}
    @Override
    public void setNome() {}
    @Override
    public void setTintaModelo() {}
}
class ImpressoraToner implements Impressora, Toner {
    @Override
    public void setValor() {}
    @Override
    public void setNome() {}
    @Override
    public void setTonerModelo() {}
}
```

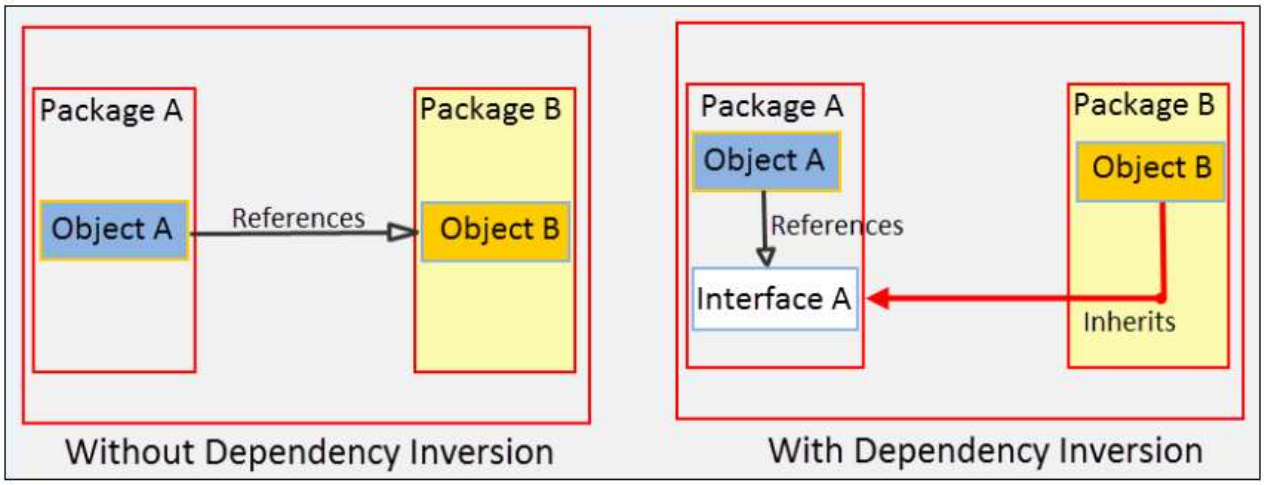
Fonte: O próprio autor (2022).

Observando a figura 12, analisa-se a aplicação da segregação de interfaces, em que cada classe de Impressora (ImpressoraTinta, ImpressoraToner) implementa somente os métodos que vão utilizar, para isso foi refatorado o código da figura 11, adicionando-se duas novas interfaces.

Dependency Inversion Principle (Princípio de Inversão de Dependência)

Esse princípio defende que uma entidade deve depender de uma abstração, e não da classe concreta, com isso módulos de alto nível não devem depender de módulos de baixo nível, mas sim de suas abstrações.

Figura 13 — Estrutura com Inversão de Dependência



Fonte: Guru Springframework (2022).

A figura 13 apresenta a estrutura com e sem a Inversão de Dependência; nesse caso, sem a aplicação desse princípio, um objeto chama o outro objeto concreto direto, mesmo sendo de pacotes (módulos) diferentes, em que um é de maior nível; e outro, de nível abaixo. Ao lado direito da figura 13, encontra-se como funciona a comunicação entre os objetos, sendo tratada por uma interface.

A principal vantagem de aplicar o princípio da Inversão de Dependência é que o sistema começa a possuir menos objetos sendo acoplados a outros e pode oferecer uma pequena vantagem em sistemas pequenos, mas, quando se fala de sistemas complexos, esse conceito pode ser crucial para o bom desenvolvimento do projeto.

Nem sempre a aplicação dos princípios SOLID são possíveis, além de ser raro um sistema que consegue aplicar todos esses conceitos juntamente com alguns patterns. Mas vale sempre lembrar que são princípios e boas práticas, e não regras de desenvolvimento.

O bom desenvolvimento de um sistema web depende de diversos recursos e conhecimentos e, em alguns casos, talvez não seja possível aplicar todas as boas práticas, patterns e princípios SOLID.

Por isso a proposta é sempre buscar desenvolver sistemas web da melhor forma possível naquele momento. Caso o código não fique limpo ou com a qualidade desejada, procure não se incomodar, mas pense em sempre refatorá-lo no futuro.

A grande chave por trás do desenvolvimento web, seja back-end ou front-end, é sempre pensar na melhoria contínua do sistema web. Projetos que sempre estão atualizados e que passam por melhorias constantemente resultam em sistemas bem organizados, longevos e rentáveis para o cliente.

Os princípios SOLID, as boas práticas e os patterns podem ser utilizados e implementados em qualquer sistema independente da tecnologia, basta se aprofundar no funcionamento de cada linguagem, claro que, para a aplicação do SOLID, é necessário que a linguagem seja orientada a objetos.

Com isso, vamos agora construir em conjunto uma API, utilizando a tecnologia Java com Spring Boot, em que iremos desenvolver o cadastro de produtos, armazenando os dados em um banco de dados H2.

Videoaula 3

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

Encerramento da Unidade

Chegamos ao final da nossa quarta unidade, na qual você teve contato com os conceitos práticos de desenvolvimento web. Com certeza, muito do que foi mostrado aqui você já conhecia ou até mesmo já havia utilizado em seu cotidiano, até mesmo sem saber que estava utilizando. Espero que tenha compreendido um pouco mais acerca do desenvolvimento web.

No mundo da tecnologia, existem diversas formas de desenvolver sistemas web, porém, quando realizada uma boa análise, implementando as tecnologias e práticas apresentadas nesta disciplina, as chances de obter um software de sucesso aumentam. Na aula 1, você conheceu e foi instigado a compreender como funcionam os sistemas baseados em componentes.

Na aula 2, foram discutidos e apresentados os princípios SOLID, com práticas de desenvolvimento e exemplo em códigos, além de criar uma API Java com Spring Boot e banco de dados em memória.

Juntando as duas aulas, teremos conhecimento conceitual e prático de como são desenvolvidos sistemas web na atualidade. Caro aluno, pense um pouco em como os conhecimentos desta unidade podem ajudá-lo nos desafios do seu cotidiano. Até a próxima.

Bons estudos!

Videoaula Encerramento

Utilize o QR Code para assistir!

Assista agora ao vídeo de encerramento de nossa disciplina.

Encerramento da disciplina

Olá, aluno, tudo bem?

No decorrer da disciplina de desenvolvimento web, conseguimos discutir como os conceitos e suas práticas auxiliam na elaboração de um sistema web sólido e eficaz.

Saber a forma correta de desenvolver sistemas web implica na construção de softwares mais resistentes e longevos, aumentando sua qualidade e, com um funcionamento rápido, resolvendo para o cliente problemas que antes o incomodavam.

Claro que a aplicação dos conhecimentos apresentados não será a solução de todas as adversidades que permeiam o desenvolvimento de uma aplicação web, porém construir um software web pensando nos aspectos demonstrados nesta disciplina auxilia no desenvolvimento web de qualidade.

Com os conhecimentos obtidos nesta disciplina, espera-se que você seja capaz de analisar, identificar, estruturar e aplicar o desenvolvimento web da melhor forma possível, de acordo com a necessidade de cada cliente.

Referências

GAMMA, Erich et al. Design patterns: elements of reusable object-oriented software. Boston: Addison-Wesley, 2000. 395 p. (Addison-Wesley professional computing series)

Springframework Guru. Disponível em: https://springframework.guru/solid-principles-object-oriented-programming/. Acesso em 29 de jul. de 2022.

Robert C. Martin. 2008. Clean Code: A Handbook of Agile Software Craftsmanship (1st. ed.). Prentice Hall PTR, USA.


UniFil
EDUCAÇÃO A DISTÂNCIA
UNIFIL.BR