Entendendo o padrão MVC

Nesta aula, exploraremos um dos pilares fundamentais do desenvolvimento de software moderno, o padrão Model-View-Controller (MVC). Este padrão é essencial para a construção de aplicações eficazes, eficientes e de fácil manutenção, organizando a lógica de negócios, a interface do usuário e o processamento de entradas em três componentes distintos e interconectados. Através de analogias claras e exemplos práticos, desvendaremos como o MVC facilita a separação de responsabilidades em um projeto, melhorando a interoperabilidade, performance e escalabilidade da aplicação. Esta introdução conceitual nos preparará para aprofundarmos nos intricados detalhes do MVC, entendendo seu funcionamento, aplicabilidade e como ele se encaixa nas diversas arquiteturas de software. Prepare-se para uma jornada enriquecedora pelo universo do MVC, um conhecimento indispensável para qualquer desenvolvedor aspirando a excelência em programação.

Definição de Arquitetura de uma aplicação

Este tema, amplamente discutido no mercado atual, é essencial para o desenvolvimento de sistemas eficazes e eficientes. A arquitetura de uma aplicação, muitas vezes confundida com o design pattern, é, na verdade, o alicerce sobre o qual o sistema é construído. Fazendo uma analogia ao projeto de uma casa, a arquitetura seria a planta que detalha a base do projeto, orientando sobre o que é necessário antes mesmo de iniciar a construção.

Este conceito é crucial, por definir a estrutura que suportará todas as funcionalidades da aplicação, garantindo que as decisões de design sejam

tomadas com uma compreensão clara do objetivo final. Ao planejar uma aplicação, é imprescindível entender suas necessidades - quantos "quartos" ou módulos serão necessários, por exemplo - para desenhar um sistema que atenda a esses requisitos de maneira eficaz.

A arquitetura de uma aplicação não só facilita o desenvolvimento e a manutenção do sistema como também define um caminho a ser seguido, um conjunto de práticas recomendadas que visam a criação de uma aplicação de alta performance e qualidade. Entre as características principais de uma boa arquitetura de aplicação, destacam-se a interoperabilidade, que permite a comunicação e compatibilidade com outros sistemas, e a performance, aspecto fundamental para garantir a eficiência das interações do sistema.

Além disso, a arquitetura deve contemplar o desempenho e a escalabilidade da aplicação. O desempenho se refere à capacidade de atender às expectativas dos usuários finais, enquanto a escalabilidade diz respeito à facilidade com que a estrutura do sistema pode ser ampliada ou replicada conforme necessário. Estes pontos são essenciais para assegurar que a aplicação possa crescer e se adaptar sem comprometer a qualidade ou incorrer em custos exorbitantes.

Neste contexto, entender a arquitetura de uma aplicação é o primeiro passo para desenvolver sistemas robustos e flexíveis. A seguir, avançaremos na compreensão desses conceitos, introduzindo nomes e imagens que ilustram de forma prática as diferenças entre arquitetura e design pattern, consolidando o aprendizado neste importante tópico da disciplina.

Diferença entre arquitetura e design pattern

Dando continuidade, nesta segunda parte, vamos nos aprofundar na distinção entre arquitetura e design pattern. Este tópico, essencial para o desenvolvimento de software, é muitas vezes fonte de confusão, mas com exemplos práticos e analogias, ficará mais claro.

Arquitetura de uma aplicação, como abordado anteriormente, é comparável à planta de uma casa. Ela define a estrutura sobre a qual o sistema é construído, englobando a organização dos seus componentes, suas interações e como eles se integram ao ambiente de execução. A arquitetura é o esqueleto da aplicação, responsável por aspectos de alto nível como requisitos, infraestrutura e definições estratégicas.

Por outro lado, o design pattern pode ser visto como o design interior dessa casa. Ele não aborda a estrutura global, mas foca em soluções pontuais, ao nível de código, para problemas comuns de design de software. O design pattern é um modelo ou um template que pode ser aplicado para resolver problemas específicos em uma parte do sistema, facilitando a manutenção e a escalabilidade ao promover soluções testadas e eficientes.

Para exemplificar a diferença, considere a arquitetura como a decisão de construir paredes, um teto e escolher onde cada quarto ficará. Em contraste, o design pattern se refere à escolha de um estilo particular de decoração para cada quarto, otimizando o uso do espaço e a funcionalidade.

Na prática, a arquitetura é a fundação que suporta a aplicação inteira, determinando como os grandes blocos funcionais se conectam e se comunicam, seja através de microsserviços, gateways de segurança ou bases de dados. Ela é visível no nível macro da aplicação, influenciando diretamente seu desempenho, segurança e capacidade de integração.

O design pattern, contudo, opera em um nível mais granular. Ele se manifesta como uma solução para um requisito específico em um serviço ou componente da arquitetura. Por exemplo, um padrão de design pode ser aplicado para implementar uma validação de entrada de dados de forma eficiente e reutilizável.

Utilizando ferramentas modernas de documentação e diagramação, é possível visualizar e comparar ambos os conceitos. Essas ferramentas

permitem a construção colaborativa de fluxogramas e outros diagramas que ajudam a planejar tanto a arquitetura quanto a aplicação de design patterns em uma aplicação, facilitando a compreensão e a implementação efetiva de ambos.

Neste contexto, compreender a distinção entre arquitetura e design pattern é crucial para o desenvolvimento de software, permitindo que desenvolvedores construam aplicações robustas, escaláveis e fáceis de manter. No próximo tópico, exploraremos exemplos concretos de arquiteturas e visualizaremos como identificar e diferenciar tipos de arquiteturas, assim como sua aplicabilidade e benefícios.

Exemplos de arquitetura de aplicações

Esta etapa é crucial para entendermos a aplicabilidade prática dos conceitos discutidos anteriormente, como a distinção entre arquitetura e design pattern, e como eles se manifestam no desenvolvimento de software.

A primeira arquitetura que exploraremos é a Serverless. Caracterizada pela abstração da gestão de servidores, esta arquitetura permite aos desenvolvedores focar no código e na lógica de negócios, deixando a infraestrutura para provedores de serviços na nuvem. Os recursos são utilizados e pagos conforme a demanda, proporcionando escalabilidade e eficiência sem a necessidade de manter servidores ativos continuamente. Este modelo é particularmente útil para aplicações que experienciam variações significativas de tráfego ou que requerem rápida escalabilidade.

Em seguida, discutiremos a Arquitetura Orientada a Eventos, que se baseia na reação a eventos específicos. Nesta arquitetura, componentes do sistema respondem a ações ou mudanças de estado, processando-as conforme ocorrem. Esta abordagem favorece a construção de sistemas altamente reativos e assíncronos, capazes de lidar com fluxos de dados em tempo real de maneira eficiente. Essencial para sistemas onde o tempo de

resposta e a capacidade de processamento de eventos em massa são críticos, a arquitetura orientada a eventos promove uma estrutura desacoplada e distribuída.

Por fim, abordaremos a Arquitetura de Microsserviços, uma das mais faladas e adotadas no desenvolvimento moderno de aplicações. Contrapondo-se ao modelo monolítico, essa arquitetura fragmenta a aplicação em pequenos serviços independentes, cada um responsável por uma parte específica da lógica de negócios. Isso permite escalabilidade granular, onde apenas os serviços com maior demanda necessitam de escalonamento. Além disso, promove a flexibilidade de desenvolvimento e implantação, pois cada microsserviço pode ser desenvolvido, implantado e gerenciado de forma independente.

Para cada uma dessas arquiteturas, utilizaremos a ferramenta DrawIO para ilustrar e documentar visualmente como elas são estruturadas. Esta abordagem não só facilita a compreensão dos conceitos, mas também serve como prática para documentação de arquiteturas de software, uma habilidade essencial para qualquer desenvolvedor.

Em síntese, estas arquiteturas oferecem diferentes benefícios e se adequam a variados cenários de aplicação. A escolha entre elas deve considerar as necessidades específicas do projeto, como demandas de escalabilidade, gerenciamento de infraestrutura e a natureza dos dados e interações que a aplicação irá processar. Nas próximas aulas, aprofundaremos no padrão MVC, sua estrutura, aplicação prática e como ele se encaixa no contexto das arquiteturas discutidas. Prepare-se para colocar a teoria em prática em nosso projeto. Nos vemos na próxima aula.

Detalhando o MVC

Na conclusão de nossa aula, vamos ao detalhamento deste importante padrão de design de software que será utilizado em nosso projeto. MVC, sigla para Model-View-Controller, é uma estrutura dividida em três

componentes principais que separa a lógica de negócios, a interface do usuário e a interação do usuário em partes independentes. Essa separação ajuda a gerenciar a complexidade de aplicações, facilitando a manutenção, a escalabilidade e a testabilidade.

O componente Model representa a camada de dados e a lógica de negócios. Ele é responsável por acessar o banco de dados, executar consultas e retornar os dados necessários. Em nosso projeto, por exemplo, ao definirmos entidades como consultas médicas, médicos e pacientes, estamos trabalhando na camada Model, que cuida de toda a gestão de informações sem se preocupar com a apresentação ou as ações do usuário.

Imagine uma funcionalidade em nosso projeto que gere relatórios médicos. O Model seria responsável por acessar o banco de dados e recuperar os dados dos pacientes, dos médicos e das consultas. Por exemplo, para gerar um relatório de consultas por médico, o Model consultaria o banco de dados para encontrar todas as consultas associadas a um determinado médico, tratando essas informações sem se preocupar com como serão apresentadas ou interagidas pelo usuário.

A camada View é o que o usuário vê: a interface gráfica. Ela exibe os dados recebidos do Model e envia as interações do usuário (como cliques e entradas de dados) ao Controller. Esta camada é crucial para a experiência do usuário por ser onde a interação acontece, mas ela não processa os dados recebidos, apenas os apresenta de forma legível e acessível.

Após o Model preparar os dados do relatório de consultas, a camada View se encarrega de apresentá-los. Isso pode ser feito por meio de uma tabela na interface do usuário que lista os detalhes das consultas, como data, horário, paciente e diagnóstico. A View é projetada para ser atraente e intuitiva, permitindo que o usuário visualize as informações de forma clara, mas sem executar lógicas de negócios ou acessar diretamente os dados.

O Controller atua como intermediário entre a View e o Model. Ele recebe as ações do usuário transmitidas pela View, processa essas ações (podendo incluir validações) e solicita dados ao Model. Após receber os dados solicitados, o Controller os encaminha de volta à View para serem exibidos. Essa camada de controle é essencial para a orquestração do fluxo de informações e para a lógica de interação no sistema.

Funcionando como o maestro dessa orquestra, o Controller recebe a solicitação do usuário para gerar o relatório de consultas por médico. Ele então solicita ao Model os dados necessários e, uma vez recebidos, encaminha esses dados à View para apresentação. Se o usuário decidir filtrar as consultas por data, por exemplo, ele interage com a View, que passa essa solicitação ao Controller. O Controller, por sua vez, ajusta a solicitação de dados ao Model conforme necessário e atualiza a View com as novas informações.

Exemplo Prático: Aplicativo de E-commerce:

No contexto do nosso aplicativo de e-commerce, o Model gerencia a lógica de negócios relacionada a produtos, pedidos e contas de usuários. Por exemplo, o Model define e manipula as informações de produtos, como nome, preço, descrição e estoque.

A View apresenta os produtos e informações de forma visual no site, permitindo que os usuários naveguem pelo catálogo, vejam detalhes dos produtos e adicionem itens ao carrinho de compras.

O Controller atua como o mediador entre o Model e a View, processando as entradas do usuário, como adicionar um produto ao carrinho, e fazendo as chamadas apropriadas ao Model para atualizar os dados conforme necessário.

Este exemplo prático não apenas ilustra o funcionamento do MVC, mas também demonstra a importância da separação de responsabilidades para

o desenvolvimento eficiente de aplicações web.

Um case de sucesso que demonstra a eficácia do padrão MVC é o sistema de gerenciamento de uma biblioteca online. Neste sistema, o MVC foi implementado para gerenciar usuários, livros e empréstimos.

- Model: gerencia o catálogo de livros, usuários e registros de empréstimos.
- View: proporciona uma interface amigável para que os usuários busquem livros, façam empréstimos e visualizem seus históricos de empréstimos.
- Controller: coordena as ações dos usuários, como buscar livros, fazer e devolver empréstimos.

Em suma, a arquitetura MVC é um padrão poderoso para o desenvolvimento de aplicações web, promovendo a separação de responsabilidades entre os componentes do sistema. Esta separação não apenas simplifica o desenvolvimento e a manutenção de aplicações complexas, mas também facilita a divisão do trabalho em equipes de desenvolvimento. Na próxima etapa do nosso curso, aplicaremos o padrão MVC ao nosso projeto prático, consolidando o aprendizado teórico através da experiência direta com a codificação e estruturação de um sistema seguindo esta arquitetura. Preparados para colocar a teoria em prática, nos veremos na próxima aula.

GitHub da Disciplina

Você pode acessar o repositório da disciplina no GitHub a partir do seguinte link: https://github.com/FaculdadeDescomplica/ProgramacaoII. Esse repositório tem como principal objetivo guardar os códigos das aulas práticas da disciplina para aprimorar suas habilidades em vários tópicos, incluindo a

criação e consumo de APIs com controle de autenticação utilizando Node.js e utilizando boas práticas de programação e mercado.

Conteúdo Bônus

Para aqueles interessados em aprofundar seus conhecimentos sobre o padrão Model-View-Controller (MVC) e como ele se aplica no desenvolvimento de software moderno, recomendo o vídeo "MVC // Dicionário do Programador". Este conteúdo, produzido pelo canal Código Fonte TV no YouTube, oferece uma visão clara e direta sobre o funcionamento do MVC, seus componentes e sua importância na estruturação de projetos de software eficazes.

Referências Bibliográficas

Bibliografia Básica:

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 7.ed. Pearson: 2018.

MEDEIROS, L. F. de. **Banco de dados:** princípios e prática. Intersaberes: 2013.

VICCI, C. (Org.). **Banco de dados**. Pearson: 2014.

Bibliografia Complementar:

CARDOSO, L. da C. **Design de aplicativos**. Intersaberes: 2022.

JOÃO, B. do N. **Usabilidade e interface homem-máquina**. Pearson: 2017.

LEAL, G. C. L. **Linguagem, programação e banco de dados:** guia prático de aprendizagem. Intersaberes: 2015.

PUGA, S.; FRANÇA, E.; GOYA, M. **Banco de dados**: implementação em SQL, PL/SQL e Oracle 11g. Pearson: 2013.

SETZER, V. W.; SILVA, F. S. C. **Bancos de dados**. Blucher: 2005.

Ir para exercício