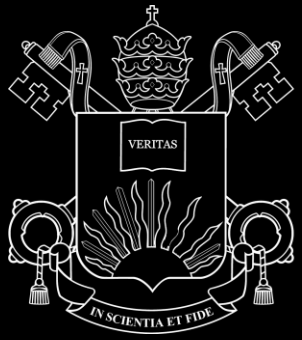


PUC GOIÁS

CURSO: ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CMP1550

Padrões (software dev)
- de projeto
- de arquitetura

Prof Fabricio Schlag

Design pattern

- Desenho / padrão de projeto
- Oferecer solução para um problema que acontece em projeto de software.
- Visa melhores práticas

Padrão arquitetural de software

- Oferece solução para o software que já é madura nas ocorrências de projeto
  - Foca área geral de engenharia de software
  - Projeto pensando no desempenho – equipamentos, na questão de disponibilidade e acesso e voltado á gestão do negócio.
  - No geral estão já inseridos em frameworks (plugin) para o desenvolvedor

Como fazer?

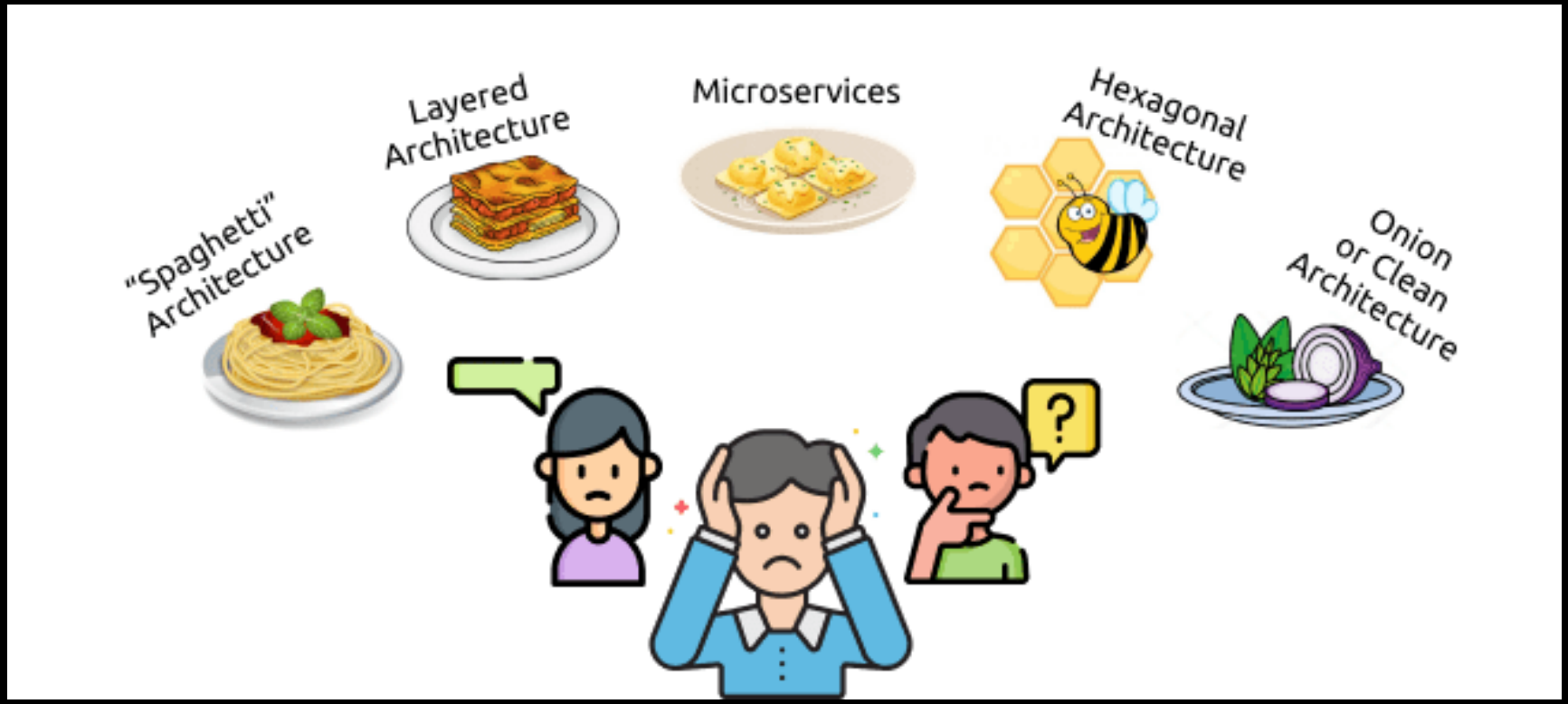
- Projetar sua aplicação visual de forma que as classes do modelo fiquem desacopladas da visualização.

Ainda

- Permitir que diferentes formas de interação possam ser feitas de modo intercambiável entre a visualização e o modelo

E...

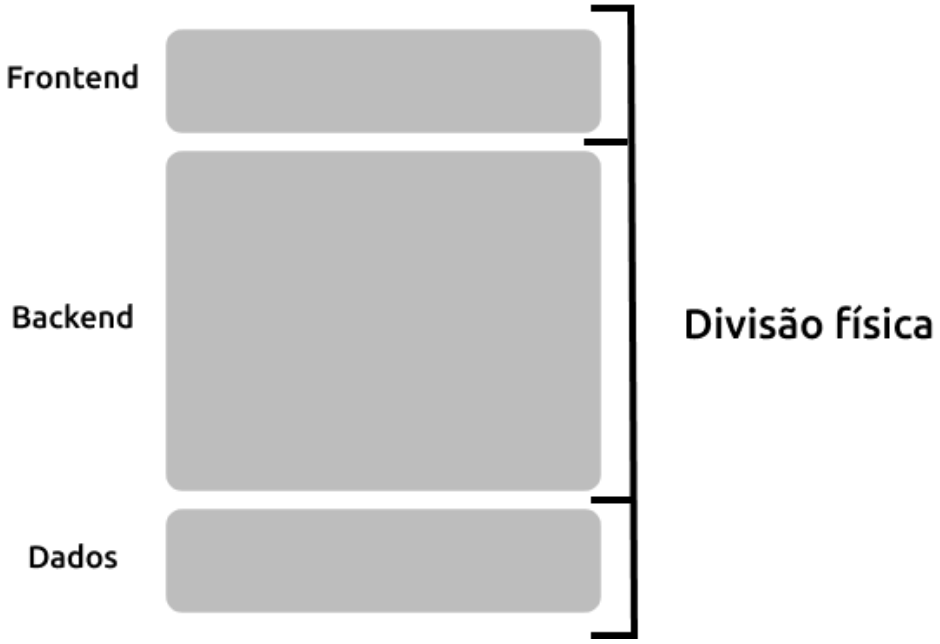
- Permitir que a atualização da interface (apresentação) seja feita de modo uniforme independentemente dos elementos que a compõem.


"Spaghetti" Architecture
Layered Architecture
Microservices
Hexagonal Architecture
Onion or Clean Architecture
?

guia.dev

No geral (web) – Divisão em camadas Física

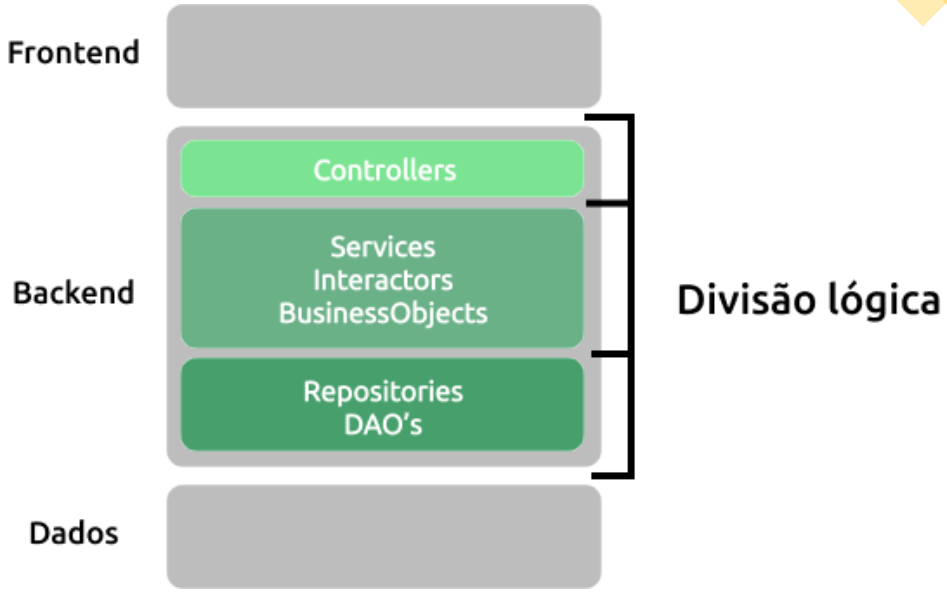
- Pois é quando se executa em processos ou máquinas distintas



☹ desempenho

Camada Lógica

- considerada lógica quando executada junto ao mesmo processo de outras camadas (superior ou inferior).
- Normalmente as chamadas e trocas de dados entre as camadas são através da invocação de métodos dentro de um mesmo processo em execução da máquina virtual utilizada.



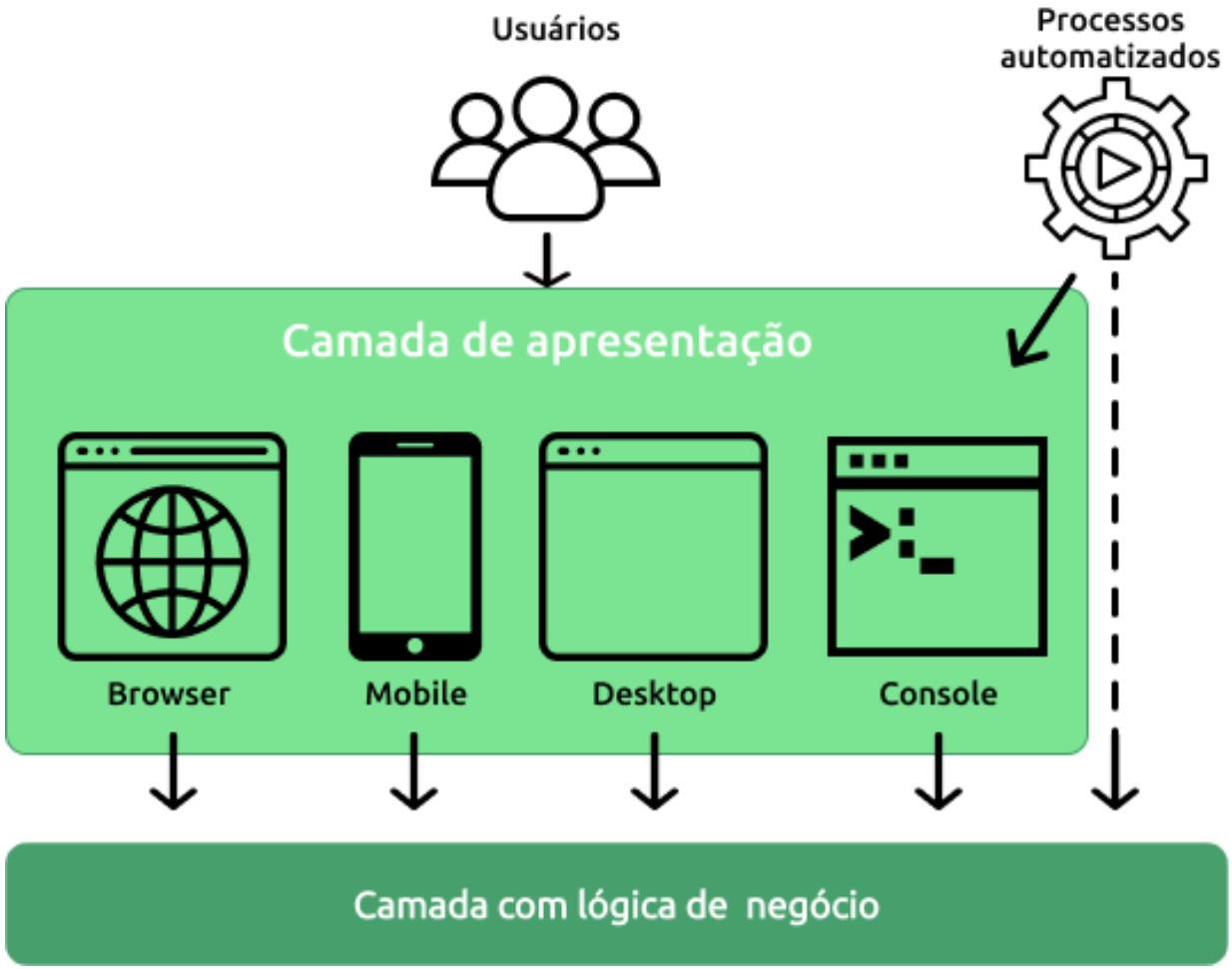
camada física pode ter várias camadas lógicas.

Camada Apresentação

- Frontend – interface – *rick cliente*
  - Interação – usuário – máquina.
  - Interfaces web
    - Html / javascript (responsividade)
    - Linguagens para Android – IOS
  - Solução de interface de interação para usuário => linha de comando (script) no SO.

Camada Apresentação

não conhecer lógica de negócio



https://guia.dev/

A camada de apresentação precisa:

1. Mostrar ao usuário que informações de entrada são necessárias.
2. Capturar essas informações.
3. Agrupar, formatar e converter as informações conforme a camada de negócio espera receber.
4. Enviar as informações para a camada de negócio.
5. Processar o resultado formatando e agrupando as informações da melhor forma para exibição ao usuário.
6. Exibir o resultado, seja ele de sucesso ou erros.

Para padrão de projeto

camada de apresentação precisa:

Quanto à codificação:

- Condicionais ou lógicas com relação à validação dos dados (camada de negócio)
- Conversões de dados (camada de negócio)
- Chamadas a vários métodos ue representam um mesmo caso de uso ou regra de negócio.
- Pode ter acesso a camadas inferiores a de negócio
- Integrações com serviços externos sem passar por uma camada inferior
- Ações como notificações a clientes – processos paralelos etc.

Assim, podemos ter camadas lógicas – segue próx

Camada lógica de negócio

Ou de domínio = código referente à lógica do negócio.

- “contratos” relacionados aos dados de entrada
- validações dos dados de entrada.
- cálculos, regras e lógicas
- envio dos dados para persistência
- recuperação de dados, agrupando-os e organizando-os.
- retorno estruturado de dados para camadas superiores

Tudo de acordo com modelo de domínio

Camada lógica de negócio

- Se bem estruturada e implementada – sofre menor manutenção.

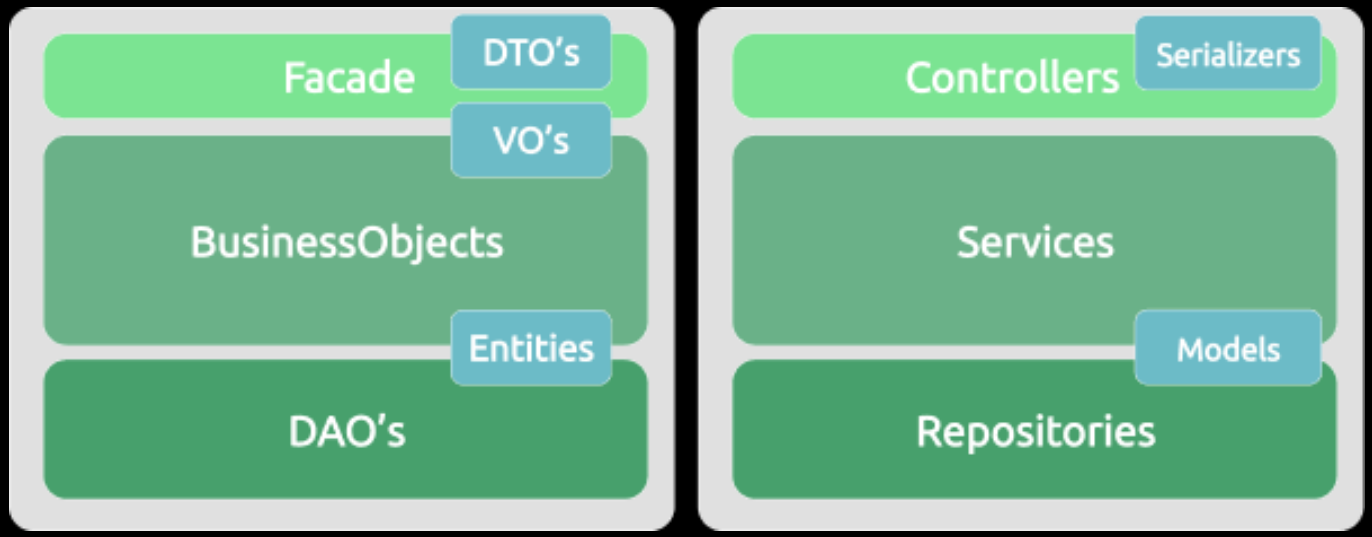
- Problemas!!

Podem ocorrer!!

Lógica de negócio

- Alteração de versão de frameworks afetando código de negócio.
- Novos frameworks e equipes de desenvolvimento reescrevendo código de negócio.
- Mudanças no código de negócio em função de mudanças em outras camadas - usabilidade na camada de apresentação.
- Incertezas com relação a mudanças no código de negócio quando necessário, como:
  - A mudança quebrará outras camadas?
  - A mudança será o suficiente naquele ponto ou será necessário revisar as demais camadas?
  - Qual o impacto de uma mudança nas demais regras de negócio?

Camada de negócio - física ou lógica



Camada de Dados

- Maioria - persistência em um banco de dados
- Pode ser Ferramenta ou(e) sistema.
- Oferecer dados as demais camadas.
    - Sistemas de cache.
    - Sistemas de mensageria.
    - Buckets de arquivos.
    - Sistemas terceiros.
    - Arquivos de texto plano

Agora?!!! Infra nas Nuvens

Camada física (no geral)

- ☹ número de chamadas que são realizadas à camada de dados.

Analise o problema – Loja virtual

Exemplo: Usuário faz consulta em sua lista de pedidos

- Sistema vai executar:

1. A apresentação envia o filtro utilizado junto a requisição feita para consultar os dados.
2. O negócio recebe o filtro, valida e faz a requisição dos dados à camada de dados.
3. A camada de dados, aplicando o filtro, retorna 25 pedidos em aberto.
4. O negócio agora tem os dados dos pedidos, mas precisa também dos dados do cliente e itens de cada pedido, logo faz uma nova requisição a camada de dados para cada item.
5. No retorno para o usuário será necessário apresentar também o endereço do cliente, então para cada cliente será necessário uma nova requisição para os dados de endereço.

Exemplo das requisições de consulta

- 76 requisições a camada de dados.
    - CONCLUSÃO!!

Soluções em Software

- Rever Modelagem de dados
- Rever as consultas em tabelas.
- Criar uma tabela com técnica de mapeamento objeto relacional
- Lógicas no código que priorizem ir menos vezes a camada de dados, fazendo uso de caches ou outras estratégias.

Volume alto de dados

- Tamanho do pacote em única requisição

Solução:

- Revisões sobre quais dados realmente são necessários no retorno. Toda informação não utilizada pode ser reconsiderada.
- Fragmentar consulta aos dados, porém mantendo atenção a quantidade de requisições a serem realizadas.
- Manter cache na camada de negócio de dados que estejam repetidos no retorno, de forma que não precisem ser solicitados a camada de dados.

Camadas com Isolamento

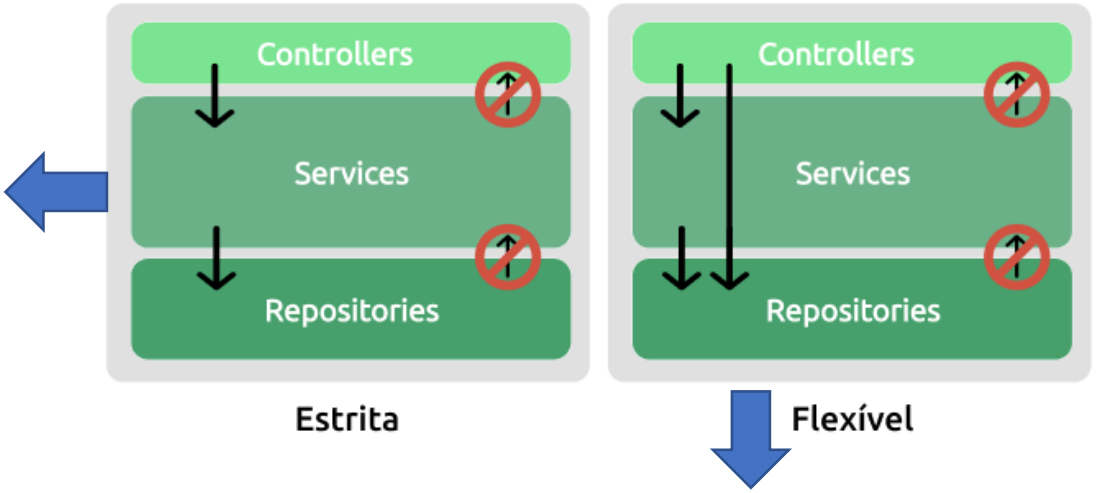
- Posso isolar a complexidade e os detalhes de uma camada para as demais camadas.
  - Manter a ordem no código
  - Manter fluxo de execução do software.
- Ver quais linguagens e frameworks que permitem o isolamento

Via de regra das camadas

- Uma camada pode conhecer camadas inferiores, porém não pode conhecer camadas superiores.
  - Não deve fazer chamadas a recursos de uma camada superior
  - O que é:?
    - Chamada de métodos.
    - Importação de classes e/ou outros artefatos.
    - Recebimento de dados em estruturas definidas ou(e) implementadas por uma camada superior.
    - Retorno de dados em estruturas definidas por uma camada superior.

Isolamento estrito ou flexível

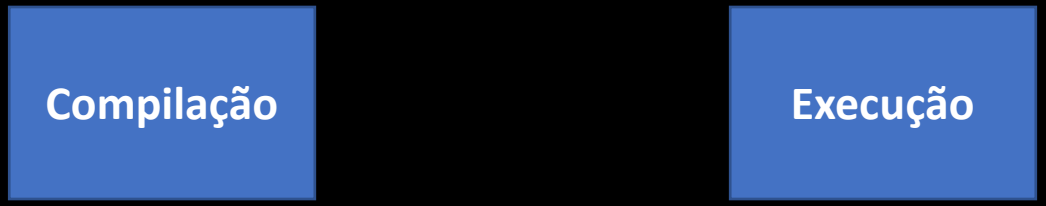
uma camada pode conhecer apenas a camada inferior



uma camada conhecer várias inferiores.

Isolamento e Dependências

- Uma camada não pode depender de uma camada superior



variar quando são estritas ou não estritas.

Uso de Inversão de dependência

- Módulos de alto nível NÃO devem depender de módulos de baixo nível - ambos devem depender de abstrações.
  - Uso em arquitetura modular
  - Boa prática em uso de camadas

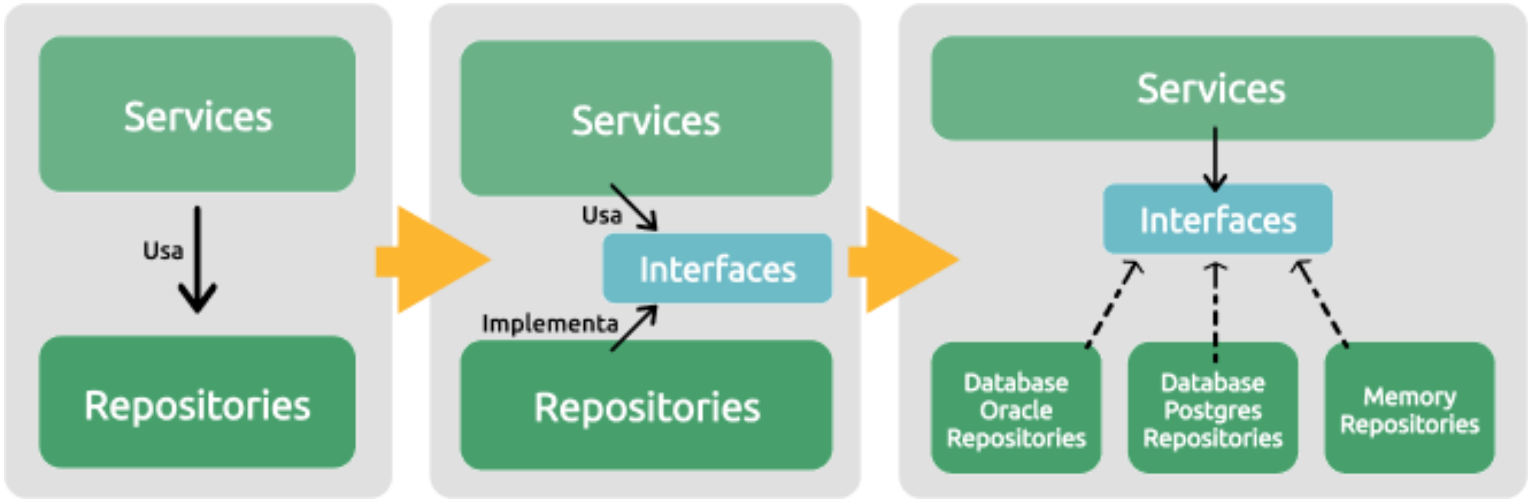
- Caso uma camada NÃO possa depender de uma superior?

Ver sobre pricípio SOLID – em OO -

Inversão

- Criar uma abstração dos contratos estabelecidos entre as camadas => uso de interfaces ou classes abstratas

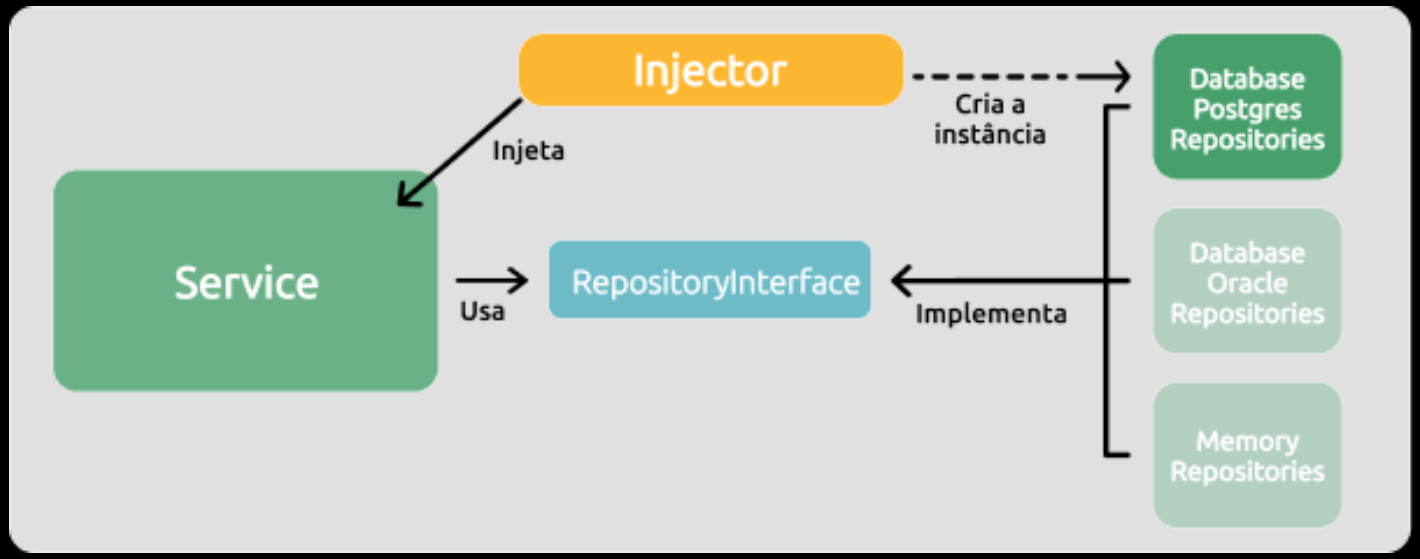
Substituição de camada



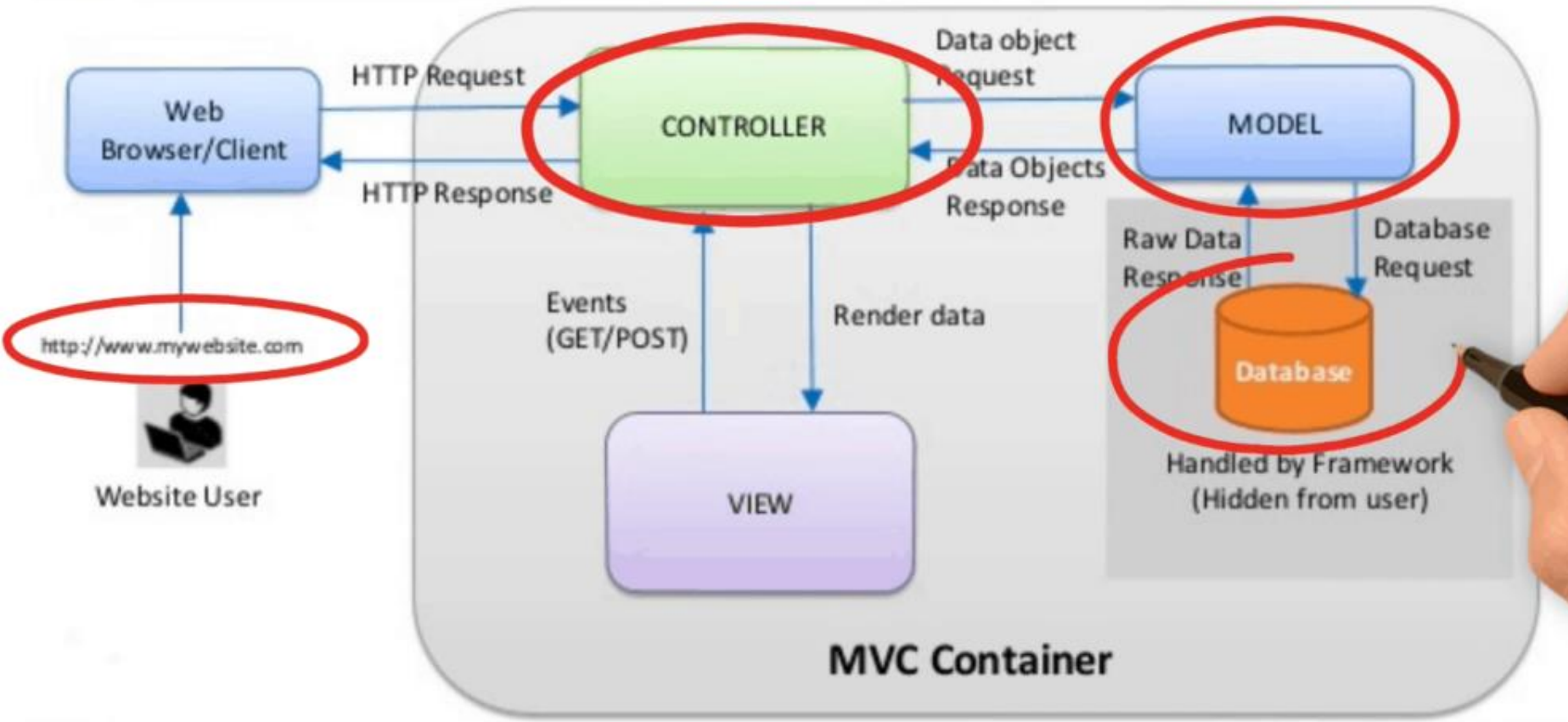
Precisa de algo a mais?

Inversão – com injeção de dependência

Cria uma instância da implementação concreta do Repository



MVC


Web Browser/Client
HTTP Request
HTTP Response
CONTROLLER
Data object Request
Data Objects Response
MODEL
Raw Data Response
Database Request
Database
Handled by Framework (Hidden from user)
Events (GET/POST)
Render data
VIEW
http://www.mywebsite.com
Website User
MVC Container

Modelo, visão e controle

- Solução para um problema dentro de um contexto.
- O que acontece com requisitos de um software... Sofrem alterações
- Separar a lógica de negócio da apresentação favorece a manutenção.

Característica MVC

- Modelo
  - Representa a lógica do negócio
  - Armazena as entidades do sistema
  - Pode interagir com a base de dado
- Visão
  - Responsável por exibir as informações ao usuário
- Controle
  - Faz a ligação entre Visão e Modelo

Modelo (model)

- Parte do sistema que gerencia todas as tarefas relacionadas aos dados
  - E.g.: a validação e estrutura de dados (banco de dados)
- Diminui a complexidade do código que o desenvolvedor escreve.
- Responsável pela lógica de negócio da aplicação.
- O Modelo encapsula um conjunto de rotinas de acesso de dados no geral são armazenados em banco de dados
- Inclui código que reforce as regras de negócio sobre os dados da aplicação

Visão (View)

- Responsável pelo gerenciamento da interface com o usuário.
- Isso inclui o gerenciamento de todos os elementos gráficos da aplicação
  - E.g.: botões, formulários etc.
- Ela controla a forma na qual os dados são apresentados aos usuários e como eles interagem com ela.
- Em aplicações web, o HTML, CSS e JavaScript são tecnologias associadas à View.
- Via de regra, na view não deve conter código elementos que pertencem à lógica da aplicação facilitando o trabalho do projetista.

Controle (controller)

- Responsável pelo gerenciamento de eventos da aplicação.
- Estes eventos podem ser disparados pela interação dos usuários com a aplicação ou por processos do sistema.
- Recebe a requisição e prepara o dado para a resposta. É responsável também pelo formato da resposta.
- O controller gerencia o relacionamento entre a view e o model.
- Responde as requisições do usuário, interage com o model e decide qual view deve ser gerada e apresentada.

Benefício

- Separa a apresentação e a interação dos dados do sistema
  - Os três componentes tem responsabilidades distintas mas interagem entre si
- Quando é recomendado? – Quando existem várias maneiras de visualizar e interagir com os dados
- São desconhecidos (ou são voláteis) os requisitos de interação com os dados

Benefícios MVC

- Permite que os dados sejam alterados de forma independente de sua representação (e vice versa) – Apoia a apresentação dos mesmos dados de maneiras diferentes
- Permite o desenvolvimento de várias visões utilizando o mesmo modelo.
- Permite testar a lógica de domínio sem ter que gerar scripts para interface gráfica, pois é muito mais fácil testar objetos não visuais do que visuais.
- Facilita a distribuição do componente de visão – Os dados são mantidos centralizados e protegidos

Desvantagem

- Complexidade alta quando o modelo de dados e de interações é muito simples

- Estrutura do padrão pode impor código adicional desnecessário

Como é?

1. O usuário interage com a Visualização (interface gráfica) - A Visualização é a janela para o Modelo. A Visualização informa o Controlador sobre todo tipo de interação feita sobre ela.
2. O Controlador pede ao Modelo para modificar seu estado - O Controlador obtém as interações ocorridas na Visualização e as interpreta traduzindo em ações que irão manipular o modelo.
3. O Controlador pede para a Visualização se modificar - O Controlador em alguns casos pede para a Visualização se atualizar (por exemplo, desabilitar um botão ou menu), após receber um vento causado por uma interação.
4. O Modelo notifica a Visualização quando seu estado é modificado - As mudanças de estado podem ocorrer devidos a eventos externos ou internos.
5. A Visualização requisita um estado ao Modelo - A Visualização busca diretamente do modelo o estado requisitado. Ela também pode requisitar um estado do modelo como resultado da ação do Controlador sobre si mesma.

Então...

Aumentar a qualidade de um software e garantir o desenvolvimento dos projetos de forma organizada, a preocupação com a arquitetura da aplicação é fundamental.

Definir a plataforma, os componentes que serão desenvolvidos e como serão organizados, além de seguir os requisitos do sistema e focar no desenvolvimento, é de suma importância.

Quais os padrões arquiteturais que podem atender suas necessidades de organização e de estruturação de código?