03

ARQUITETURA EM CAMADAS

ARQUITETURA EM CAMADAS

01

Arquitetura em camadas é um estilo arquitetural que dentre vários objetivos, destaca-se o de organizar as responsabilidades de partes de um software, normalmente criando um isolamento e dando um propósito bem definido a cada camada de forma que a mesma possa ser reutilizável por um nível mais alto ou até substituível.

ARQUITETURA EM CAMADAS

01

Dentre exemplos conhecidos, pode-se citar:

Modelo OSI, criado ainda na década de 70 e que na década de 80 virou o modelo de rede padrão para protocolos de comunicação.

Modelo OSI

ARQUITETURA EM CAMADAS

01

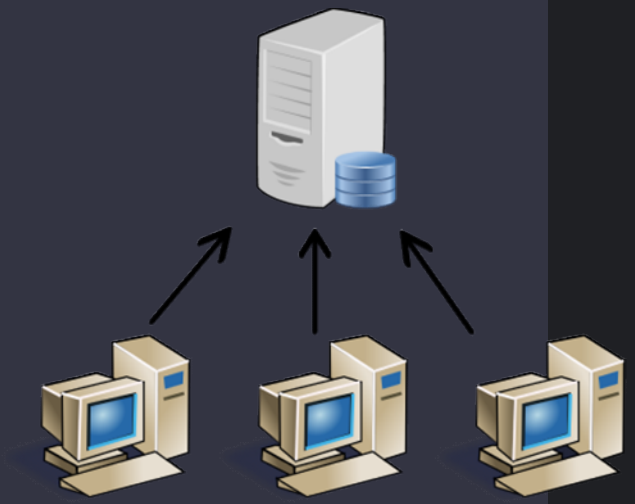
Sistemas operacionais: O uso de camadas por sistemas operacionais permitiu a padronização e evolução com relação a construção e uso de CPU, memória, dispositivos, kernel e aplicações.

Sistemas Operacionais

ARQUITETURA EM CAMADAS

01

Na década de 90, com a adoção de sistemas cliente-servidor, os quais são divididos em duas camadas, sendo o cliente com a interface do usuário e código de aplicação e no lado servidor um banco de dados para persistência.



Divisão física e lógica

01

No idioma inglês há dois termos que referentes a camadas, o Tier e Layer, os quais no idioma português possuem a mesma tradução: camadas.

- Tier refere-se a physical unit que é a divisão física.
- Layer refere-se a logical unit que é a divisão lógica.

Divisão física (Tier)

01

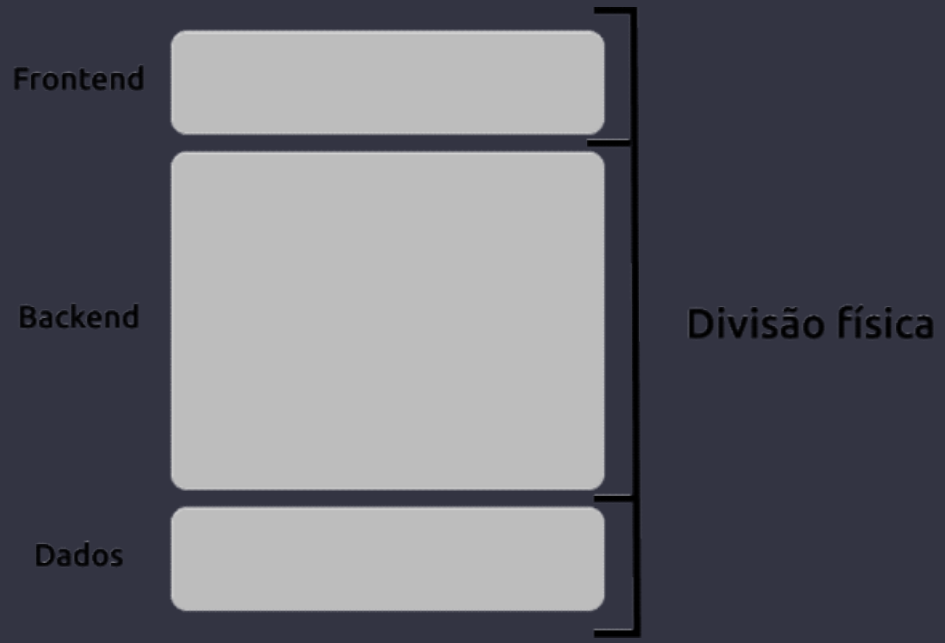
Uma camada pode ser considerada física quando ela é executada em processos ou máquinas diferentes. Se estiverem em máquinas diferentes os dados irão trafegar pela rede, ou seja, quanto mais distante, maior a chance de degradação de performance do software como um todo.

O modelo cliente-servidor, por exemplo, é considerado um modelo two-tier systems, pois rodam normalmente em máquinas diferentes.

Divisão física (Tier)

01

Aplicações web tradicionais também estão geralmente divididas em camadas físicas.



Divisão lógica (Layer)

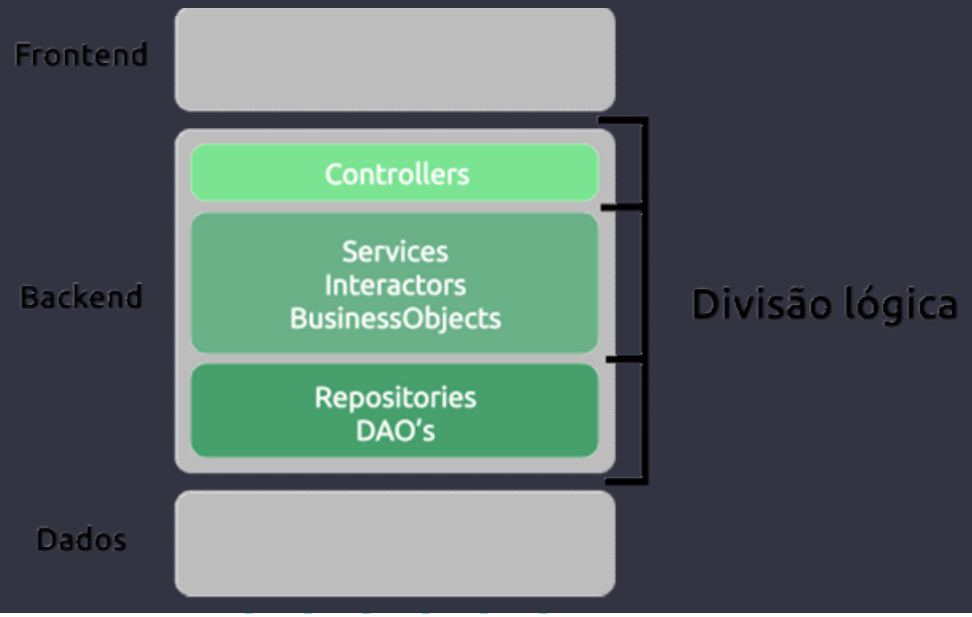
01

Uma camada pode ser considerada lógica quando ela é executada junto ao mesmo processo de outras camadas (superior ou inferior). Nestes casos 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. Poderá haver conversão de dados (tipo ou estrutura) entre as camadas, mas normalmente não há necessidade de serialização/deserialização de dados.

Divisão lógica (Layer)

01

Uma camada física pode conter várias camadas lógicas. O backend de soluções web, são um bom exemplo de camadas lógicas.



Modelos tradicionais

01

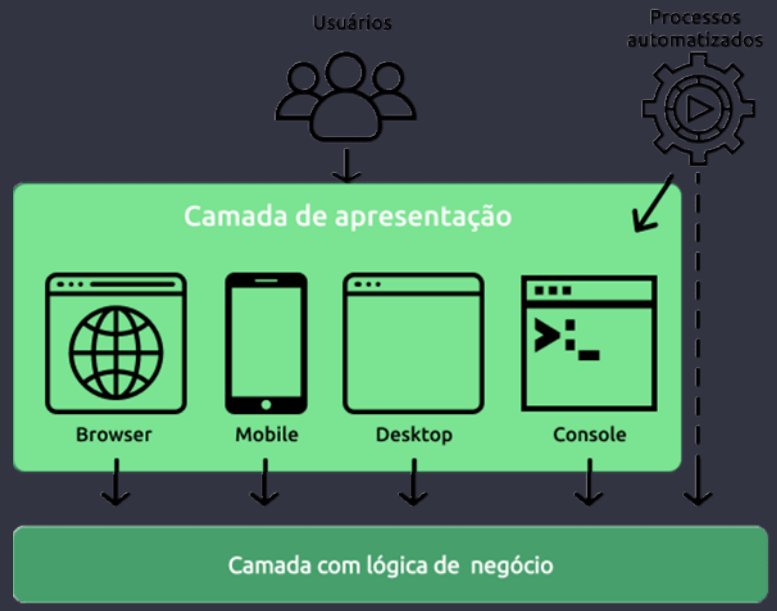
O modelo mais tradicional utilizado atualmente em arquiteturas de camadas para soluções corporativas é o de 3 camadas físicas.

Suas camadas são representadas por algumas nomenclaturas diferentes, mas tradicionalmente conhecidas por camadas de **apresentação**, **lógica de negócio** e **dados.**

Apresentação

01

A camada de apresentação, que inclui o frontend (termo amplamente utilizado atualmente), trata da interação do usuário com o software.



Apresentação

01

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.

Lógica de negócio

01

A camada de negócio, também popular pelo termo camada de domínio, é a que possui o código referente à lógica do negócio para qual o software foi concebido. Pode-se afirmar que dentre as camadas, é a mais relevante para uma empresa.

1. Os “contratos” relacionados aos dados de entrada para as funcionalidades esperadas.
2. As validações dos dados de entrada.
3. Os cálculos, regras e lógicas referentes a composição e estruturação dos dados seguindo o modelo de domínio.

Lógica de negócio

01

4. O envio dos dados para persistência.
5. A recuperação de dados, agrupando-os e organizando-os de acordo com modelo de domínio.
6. O retorno estruturado de dados para camadas superiores, mesmo que sejam erros relacionados a algum problema nas regras aplicadas.

Lógica de negócio

01

- Mudanças de versão de frameworks afetando código de negócio.
- Surgimento de novos frameworks e times de desenvolvimento reescrevendo código de negócio.
- Mudanças no código de negócio em função de mudanças em outras camadas, como melhorias de usabilidade na camada de apresentação.

Lógica de negócio

01

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?

Dados

01

A camada de dados geralmente é associada a persistência em um banco de dados, o que é uma verdade em grande parte das aplicações corporativas. Porém essa camada pode ser representada por qualquer outra ferramenta e/ou sistema que possa prover os dados para as demais camadas.

Sistemas de cache; Sistemas de mensageria; Buckets de arquivos; Sistemas terceiros; Arquivos de texto plano em um disco rígido (uma opção pouco válida para aplicações corporativas).

Dados

01

O grande benefício dela é isentar a lógica de negócio da responsabilidade de saber como persistir os dados em um local físico. Os bancos de dados predominaram nesta camada durante décadas, mas recentemente, apoiadas sobre o crescimento das infraestruturas de cloud, outras ferramentas ganharam espaço e têm se provado boas alternativas dependendo da necessidade do negócio.

ARQUITETURA MVC (MODEL-VIEW-CONTROLLER)

A arquitetura MVC é um padrão de design utilizado para organizar o código de aplicações, separando a lógica em três camadas: Model, View e Controller.

Model (Modelo)

Responsável pelos dados e regras de negócio.
Interage com o banco de dados.
Exemplo: Uma classe Usuario que lida com a persistência de dados.

ARQUITETURA MVC (MODEL-VIEW-CONTROLLER)

View (Visão)

- Responsável pela interface com o usuário.
- Exibe as informações e captura as interações.
- Exemplo: Arquivos HTML com JavaScript e CSS.

Controller (Controlador)

- Gerencia as requisições do usuário.
- Atualiza o modelo e escolhe qual visão será exibida.
- Exemplo: Um arquivo UsuarioController.php que recebe dados do formulário e chama o Model para salvar no banco.

ARQUITETURA MVC (MODEL-VIEW-CONTROLLER)

Prós da Arquitetura Mvc (Model-view-controller)

- Separação de responsabilidades melhora a organização do código.
- Facilita a manutenção e a escalabilidade do sistema.
- Permite que diferentes equipes trabalhem simultaneamente em cada camada.
- Reutilização de código, já que a camada de Model pode ser usada por diferentes Views.

ARQUITETURA MVC (MODEL-VIEW-CONTROLLER)

Contras da Arquitetura Mvc (Model-view-controller)

- Pode adicionar complexidade desnecessária para projetos pequenos.
- A comunicação entre camadas pode gerar um leve impacto na performance.
- Pode ser difícil de aprender e implementar corretamente para iniciantes.

ARQUITETURA REST (REPRESENTATIONAL STATE TRANSFER)

REST é um estilo de arquitetura para construção de APIs baseadas em HTTP. Ele define um conjunto de princípios que tornam a comunicação entre sistemas escalável e eficiente.
Principais Características do REST:

- **Baseado em Recursos:** Cada endpoint representa um recurso (exemplo: /usuarios, /produtos).
- **Uso de Métodos HTTP:** GET → Buscar dados, POST → Criar novos dados, PUT → Atualizar dados, DELETE → Remover dados.
- **Formato de Dados:** Normalmente JSON, mas pode ser XML.
- **Stateless:** O servidor não mantém estado entre requisições.

ARQUITETURA REST (REPRESENTATIONAL STATE TRANSFER)

Prós da Arquitetura Rest

- Simples e fácil de integrar com diferentes tecnologias e plataformas.
- Utiliza HTTP, que já é amplamente suportado e conhecido.
- Stateless (sem estado), o que melhora a escalabilidade.
- APIs RESTful podem ser consumidas por qualquer tipo de cliente (web, mobile, IoT).

ARQUITETURA REST (REPRESENTATIONAL STATE TRANSFER)

Contras da Arquitetura Rest

- A ausência de estado pode exigir mais requisições para obter informações contextuais.
- Nem todas as APIs seguem o padrão REST corretamente, gerando inconsistências.
- Pode ser ineficiente para operações em tempo real, onde WebSockets são mais recomendados.

ARQUITETURA REST (REPRESENTATIONAL STATE TRANSFER)

Comparação MVC vs REST

MVC	REST
Padrão de design de software	Estilo de arquitetura para APIs
Organiza código em camadas (Model, View, Controller)	Define princípios para comunicação entre sistemas
Muito usado em aplicações web (exemplo: Laravel, Django)	Muito usado para criar APIs (exemplo: Node.js com Express, Flask)

Exercício: Criando um Formulário com PHP

01

Objetivo: Criar um formulário simples onde o usuário poderá cadastrar seu nome, e-mail e senha. Esses dados serão enviados para um arquivo PHP, que os processará e os exibirá na tela. Além disso, a senha será armazenada de forma segura usando criptografia!

Criar um formulário em HTML (formulario.php)

O formulário deve conter os seguintes campos:

- ✓ Nome
- ✓ E-mail
- ✓ Senha
- ✓ Botão de envio

Exercício: Criando um Formulário com PHP

01

Criar um script PHP

O script deve:

✓ Receber os dados enviados pelo formulário

✓ Utilizar htmlspecialchars() para evitar ataques XSS

✓ Criptografar a senha usando password_hash()

✓ Exibir os dados na tela:

- Senha: Para exibir a senha Criptografada (Nunca faça isso em projeto real);
- Mensagem: Usuário cadastrado com sucesso!;
- Um link: Ir para Login.

Exercício: Criando um Formulário com PHP

01

Por que usar htmlspecialchars()?

A função htmlspecialchars() em PHP é usada para converter caracteres especiais em entidades HTML, ajudando a prevenir ataques de Cross-Site Scripting (XSS).

Se um usuário mal-intencionado inserir código HTML ou JavaScript em um formulário e o servidor exibir esse código sem proteção, ele pode ser interpretado pelo navegador, causando vulnerabilidades.

- ✓ Evita XSS
- ✓ Protege a exibição de dados dinâmicos
- ✓ Recomendado para saída de dados que serão exibidos em HTML

Exercício: Criando um Formulário com PHP

01

Como criptografar a senha corretamente?

O PHP fornece a função password_hash(), que utiliza um algoritmo seguro para proteger senhas.

- password_hash($_POST["senha"], PASSWORD_DEFAULT);
- Essa função criptografa a senha usando o algoritmo bcrypt por padrão.
- O bcrypt adiciona um "salt" automaticamente, dificultando ataques de força bruta.
- O bcrypt (via password_hash()) é a forma recomendada pelo PHP.

EXEMPLO DE FORMULÁRIO

```
<div class="container">
<h2>Cadastro de Usuário</h2>
<form action="" method="POST">
   <label for="nome">Nome:</label>
   <input type="text" id="nome" name="nome" required><br><br>
   <label for="email">E-mail:</label>
   <input type="email" id="email" name="email" required><br><br>
   <label for="senha">Senha:</label>
   <input type="password" id="senha" name="senha" required><br><br>
   <button type="submit">Cadastrar</button>
</form>
```

EXEMPLO DE CSS

```css
body {
    font-family: Arial, sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-color:rgb(167, 163, 173);
}
.container {
    background: white;
    padding: 100px;
    border-radius: 10px;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
}
```

EXEMPLO DO SCRIPT PHP

```php
<?php
session_start(); // Inicia a sessão

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $email = htmlspecialchars($_POST["email"]);
    $senha = password_hash($_POST["senha"], PASSWORD_DEFAULT); //
Criptografa a senha

    // Armazena os dados na sessão
    $_SESSION["email"] = $email;
    $_SESSION["senha"] = $senha;

    echo "$senha"; //Exibe a senha Criptografada
    echo "<h2>Usuário cadastrado com sucesso!</h2>";
    echo "<p><a href='login.php'>Ir para Login</a></p>";
}
?>
```

Exercício: Verificando Senhas com PHP

01

O que é session_start()?

O comando session_start() em PHP é utilizado para iniciar uma sessão ou continuar uma sessão existente. Ele é essencial para trabalhar com dados persistentes durante a navegação do usuário, como quando você quer armazenar informações de login ou preferências do usuário em várias páginas da sua aplicação.

Quando você usa sessões em PHP, o servidor cria um arquivo temporário (no servidor) para armazenar as variáveis de sessão. O session_start() permite acessar esse arquivo e, portanto, recuperar ou manipular as variáveis de sessão.

Exercício: Verificando Senhas com PHP

01

Como funciona?

1. Criação de Sessão: Quando você chama session_start(), o PHP verifica se existe uma sessão já ativa. Se não, ele cria uma nova sessão e gera um ID único de sessão (chamado de ID de Sessão), que será armazenado no navegador do usuário, geralmente em um cookie.

2. Armazenamento de Dados: Com a sessão iniciada, você pode armazenar dados temporários em um array superglobal chamado $_SESSION. Esse array pode ser usado para salvar variáveis como nome do usuário, carrinho de compras, preferências etc.

3. Persistência de Dados: A sessão mantém esses dados entre as páginas, ou seja, ao navegar por várias páginas, as informações armazenadas em $_SESSION serão preservadas.

Exercício: Verificando Senhas com PHP

01

Como funciona?

1. Iniciando a Sessão:

```php
<?php
session_start(); // Inicia a sessão ou continua a sessão existente
?>
```

2. Armazenando Dados na Sessão:

```php
<?php
session_start(); // Sempre coloque isso no início do script
$_SESSION["usuario"] = "João";  // Armazenando dados
?>
```

Exercício: Verificando Senhas com PHP

01

Como funciona?

3. Recuperando Dados da Sessão:

```
<?php
session_start();
echo "Usuário logado: " . $_SESSION["usuario"];  // Exibindo o dado armazenado
?>
```

4. Destruindo a Sessão:

Quando você quer encerrar a sessão (por exemplo, ao fazer o logout), use session_destroy() para apagar todos os dados da sessão. No entanto, para apagar as variáveis específicas, use unset().

Exercício: Verificando Senhas com PHP

01

Como funciona?

4. Destruindo a Sessão:

```
<?php
session_start();
session_unset(); // Limpa todas as variáveis de sessão
session_destroy(); // Destrói a sessão (no servidor)
?>
```

Exercício: Verificando Senhas com PHP

01

Objetivo: Criar uma página onde o usuário pode inserir um e-mail e uma senha para fazer login. O sistema deve verificar se a senha digitada corresponde à senha cadastrada previamente. Para isso, utilizaremos a função password_verify().

Criar um formulário de login (login.php)

O formulário deve conter os seguintes campos:

- ✓ **E-mail**
- ✓ **Senha**
- ✓ **Botão de login**

Exercício: Verificando Senhas com PHP

01

Criar um script PHP para verificar a senha (verificar.php)

O script deve:

✓ Simular uma senha cadastrada (criptografada)

✓ Comparar a senha digitada com a senha armazenada usando password_verify()

✓ Exibir uma mensagem informando se o login foi bem-sucedido ou não

EXEMPLO DO LOGIN

```
<div class="container">
   <h2>Login</h2>
   <form action="verificar.php" method="POST">
     <label for="email">E-mail:</label>
     <input type="email" id="email" name="email" required>
     <label for="senha">Senha:</label>
     <input type="password" id="senha" name="senha" required>
     <button type="submit">Entrar</button>
   </form>
</div>
```

EXEMPLO DO LOGIN

```php
<?php
session_start(); // Inicia a sessão

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $email_digitado = $_POST["email"];
    $senha_digitada = $_POST["senha"];

    // Verifica se o usuário já foi cadastrado
    if (isset($_SESSION["email"]) &&
isset($_SESSION["senha"])) {
        $email_cadastrado = $_SESSION["email"];
        $senha_armazenada = $_SESSION["senha"];
```

EXEMPLO DO LOGIN

```
// Verifica se o e-mail e a senha estão corretos
  if ($email_digitado === $email_cadastrado &&
      password_verify($senha_digitada, $senha_armazenada)) {
          echo "<h2>Login realizado com sucesso!</h2>";
        } else {
          echo "<h2>Erro: E-mail ou senha incorretos.</h2>";
        }
    } else {
        echo "<h2>Nenhum usuário cadastrado ainda.</h2>";
   }
}
?>
```

Desafio: Sistema de Cadastro e Login Simples

Objetivo: Criar um sistema simples de cadastro de usuário e login, onde os dados (e-mail e senha) são salvos de maneira segura. O usuário será redirecionado para uma página de bem-vindo caso faça o login corretamente.

Estrutura do Projeto:

cadastro.php – Página de cadastro de usuário.

login.php – Página de login de usuário.

bemvindo.php – Página exibida após o login bem-sucedido.

logout.php – Página para realizar o logout do usuário.

Desafio: Sistema de Cadastro e Login Simples

Dicas:

- ✓ Lembre-se de usar session_start() no início de todas as páginas que acessam ou alteram a sessão.
- ✓ Ao armazenar a senha, use password_hash() para garantir que a senha não fique visível.
- ✓ Use password_verify() para comparar a senha fornecida no login com a senha criptografada armazenada.

Até a Próxima Aula!