ARQUITETURA DE SOFTWARE

João Choma Neto
joao.choma@unicesumar.edu.br
https://github.com/JoaoChoma/arquitetura-software
Unicesumar – Maringá

ARQUITETURAS

- Arquitetura monolítica
- Arquitetura em camadas
- Arquitetura cliente-servidor

ARQUITETURA EM CAMADAS

CAMADAS

- A arquitetura em camadas é um padrão de design de software que organiza um sistema em camadas distintas, onde cada camada tem uma responsabilidade específica e se comunica apenas com camadas adjacentes.
- Essa abordagem visa separar as preocupações e promover a modularidade, flexibilidade e reusabilidade do código.

CAMADAS

- Uma das principais vantagens da arquitetura em camadas é o isolamento de funcionalidades relacionadas em camadas distintas.
- Isso facilita a reutilização de código, uma vez que as funcionalidades podem ser encapsuladas em módulos ou componentes que podem ser facilmente compartilhados entre diferentes partes do sistema.

CAMADAS

- A arquitetura em camadas está intimamente relacionada a outros padrões de design de software, como MVC (Model-View-Controller), MVVM (Model-View-ViewModel)

REDE DE COMUNICAÇÃO

- Um dos modelos mais conhecidos que utiliza a arquitetura em camadas é o Modelo de Referência OSI (Open Systems Interconnection) e o Modelo TCP/IP.

CLIENTE-SERVIDOR

CLIENTE-SERVIDOR

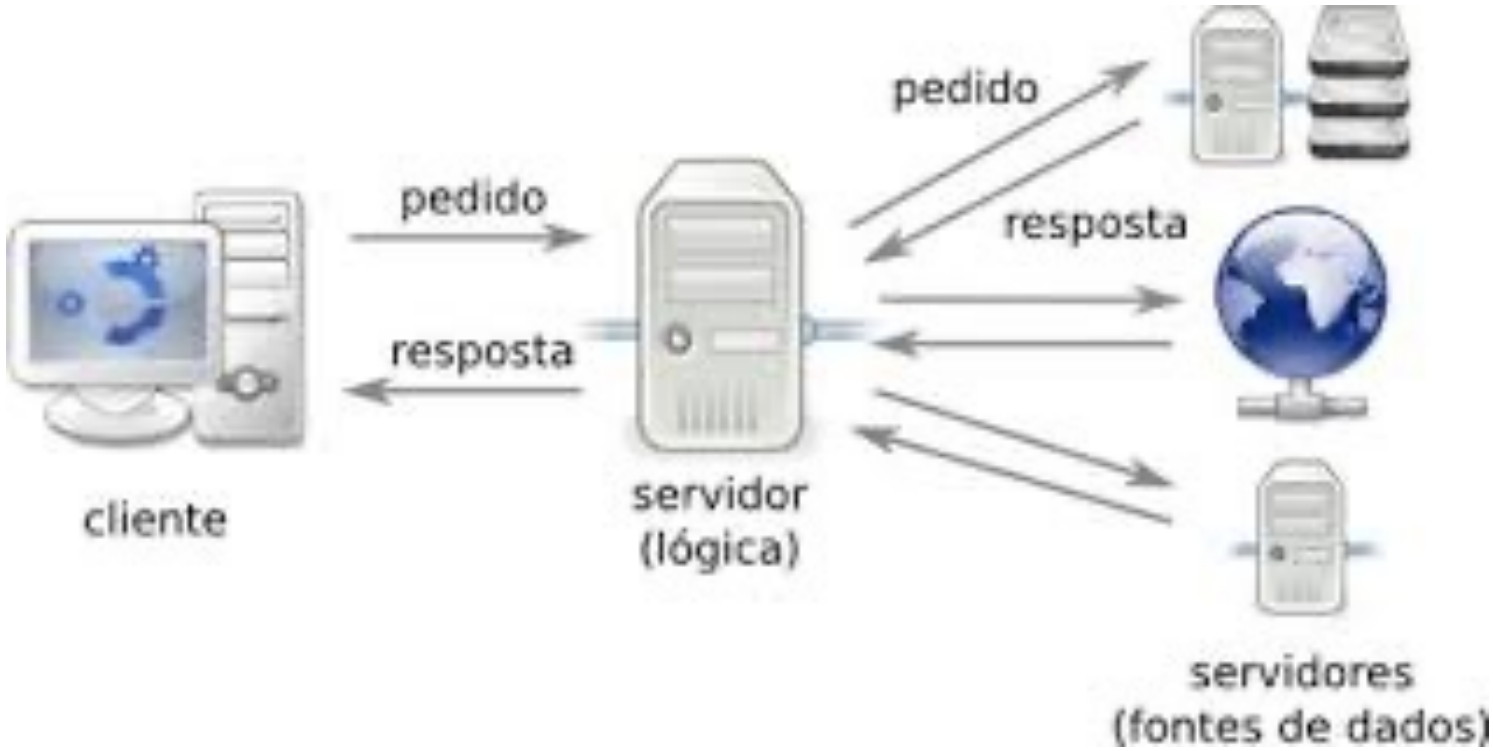
- A arquitetura cliente-servidor é um modelo de computação distribuída em que as responsabilidades e funcionalidades de um sistema de software são divididas entre dois tipos de entidades: o cliente e o servidor.
- O cliente e o servidor são programas ou dispositivos de computação que interagem entre si por meio de uma rede, como a internet.

CLIENTE-SERVIDOR

- A comunicação entre o cliente e o servidor ocorre por meio de protocolos de comunicação padrão, como HTTP, TCP/IP, WebSocket, etc.
- O cliente envia uma requisição ao servidor, especificando o serviço desejado e quaisquer parâmetros necessários.
- O servidor processa a requisição e retorna uma resposta adequada, que pode incluir dados solicitados, confirmação de operações, mensagens de erro, etc.

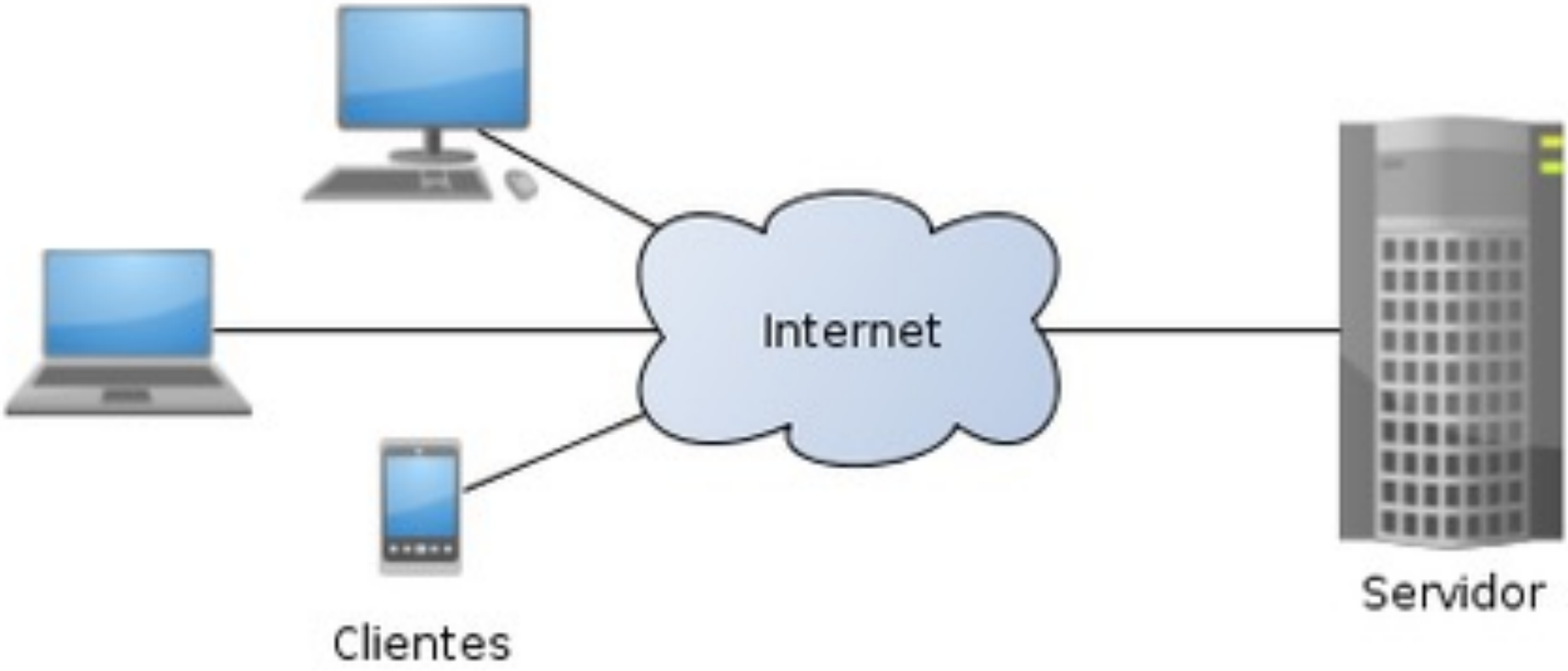
CLIENTE-SERVIDOR

- A arquitetura cliente-servidor é um modelo de design de software em que duas partes interagem em uma rede:
  - o **cliente**, que faz solicitações por serviços ou recursos, e
  - o **servidor**, que processa essas solicitações e fornece respostas


pedido
pedido
resposta
resposta
cliente
servidor
(lógica)
servidores
(fontes de dados)

CLIENTE

- **Definição**: Um cliente é uma entidade que solicita recursos ou serviços. Em um contexto web, um cliente geralmente é um navegador que solicita páginas da web.
- **Funções**:
  - Interface com o usuário: Fornece a interface através da qual os usuários interagem.
  - Solicitação de recursos: Envia pedidos ao servidor para acessar recursos como arquivos, dados, etc.
  - Recebimento e processamento de respostas: Recebe dados do servidor e os processa localmente.


Internet
Clientes
Servidor

SERVIDOR

- **Definição**: Um servidor é uma entidade que fornece recursos, dados ou serviços. Ele responde às solicitações dos clientes.
- **Funções**:
  - Gerenciamento de recursos: Mantém e gerencia recursos como bancos de dados, arquivos, etc.
  - Processamento de solicitações: Processa os pedidos recebidos dos clientes.
  - Envio de respostas: Envia as respostas para as solicitações dos clientes após o processamento.

FUNCIONAMENTO

- **Iniciação da Comunicação**: Tudo começa com o cliente enviando uma solicitação ao servidor. Essa solicitação pode ser por uma página da web, por informações de um banco de dados, ou para executar uma operação.
- **Processamento do Servidor**: O servidor recebe a solicitação, processa-a conforme necessário (que pode envolver consultar um banco de dados, executar um cálculo, etc.), e então prepara uma resposta.

FUNCIONAMENTO

- **Resposta ao Cliente**: O servidor envia a resposta de volta ao cliente. O cliente recebe esta resposta e, dependendo do tipo de solicitação, pode apresentar uma página da web, armazenar dados, ou tomar outra ação.
- **Sessão de Estado**: Em muitos casos, especialmente em aplicações web, pode ser necessário manter um estado entre o cliente e o servidor. Isso é frequentemente gerenciado por sessões ou cookies.

EMAIL

- Um exemplo comum de aplicação implementada com arquitetura cliente-servidor é um sistema de correio eletrônico (e-mail)
- O cliente seria o programa de e-mail instalado no dispositivo do usuário, como um aplicativo de e-mail em um computador ou dispositivo móvel. O cliente permite ao usuário enviar, receber, ler e gerenciar e-mails.
- O servidor é responsável por armazenar, gerenciar e distribuir os e-mails entre os clientes. Ele executa aplicativos de servidor de e-mail que lidam com o envio, recebimento, armazenamento e entrega de e-mails

TIPOS DE CLIENTES

- Navegadores web
- Aplicativos móveis
- Clientes de desktop
- Scripts e bots

TIPOS DE SERVIDORES

- Servidores web (Apache, Nginx)
- Servidores de aplicação (Tomcat, Node.js)
- Servidores de banco de dados (MySQL, MongoDB)
- Servidores de e-mail (Exchange, Postfix)

APLICAÇÕES WEB

- Os verbos HTTP, também conhecidos como métodos HTTP, são um componente crucial do protocolo HTTP (HyperText Transfer Protocol), usado para especificar a ação desejada em um recurso determinado.
- Cada método tem um propósito específico e é projetado para comunicar ao servidor o tipo de operação que o cliente deseja executar.

GET

- **Propósito**: O método GET é usado para solicitar dados de um recurso específico. Geralmente, é usado para recuperar a representação de um recurso sem causar qualquer efeito colateral (ou seja, não modifica o estado do recurso).
- GET é considerado um método seguro pois não altera o estado do recurso
- **Uso comum**: Carregar uma página web, solicitar uma imagem ou outros arquivos.

POST

- **Propósito**: POST é usado para enviar dados ao servidor para criar ou modificar um recurso. Por exemplo, quando você preenche um formulário em uma página web e o envia, seus dados são geralmente enviados ao servidor usando o método POST.
- POST não é seguro, pois modifica o estado do servidor (cria ou altera recursos).
- **Uso comum**: Enviar formulários, fazer upload de um arquivo, realizar uma operação que resulta em mudança de estado.

PUT

- **Propósito**: PUT é usado para atualizar um recurso existente ou criar um novo recurso.
- **Uso comum**: Atualizar um registro completo, como um perfil de usuário ou um post em um blog.

DELETE

- **Propósito**: DELETE é usado para remover um recurso especificado.
- **Uso comum**: Deletar um registro, como um usuário, uma postagem de blog, ou um arquivo.

emails

- O envio de e-mails na internet é primariamente gerenciado por um protocolo específico chamado SMTP (Simple Mail Transfer Protocol).
- Este protocolo é essencial para a operação dos sistemas de e-mail e desempenha um papel fundamental no processo de envio e encaminhamento de mensagens de e-mail entre remetentes e destinatários.

FUNCIONAMENTO

- **Conexão Inicial**: Quando um e-mail é enviado, o cliente de e-mail (também conhecido como agente de usuário de correio, ou MUA) começa por se conectar a um servidor SMTP configurado, geralmente oferecido pelo provedor de serviços de internet ou pelo serviço de hospedagem de e-mail.
- **Handshake SMTP**: Uma vez conectado, o cliente SMTP inicia um "handshake". Este processo inclui a saudação do servidor SMTP e a resposta do cliente, seguida pela troca de comandos que definem o remetente e o destinatário do e-mail.

FUNCIONAMENTO

- **Transmissão de Mensagem**: O corpo da mensagem e quaisquer anexos são transmitidos. O SMTP é responsável apenas pelo encaminhamento de mensagens de um servidor a outro. A entrega ao mailbox do destinatário final é geralmente gerida por outros protocolos, como IMAP ou POP3.
- **Encaminhamento**: Se o servidor SMTP receptor não for o destino final, ele encaminhará a mensagem para outro servidor SMTP mais próximo do destinatário final, utilizando uma série de consultas ao sistema de nomes de domínio (DNS) para encontrar o servidor adequado.

FUNCIONAMENTO

- **Finalização**: Uma vez que a mensagem alcança o servidor SMTP do domínio destinatário, ela é normalmente entregue à caixa de entrada do destinatário, processo que pode envolver outro protocolo, como mencionado anteriormente (IMAP ou POP3).

WebSocket

- O **WebSocket** é um protocolo de comunicação bidirecional e full-duplex baseado em TCP.
- Ele permite que um **cliente** e um **servidor** estabeleçam uma conexão persistente, na qual ambos podem enviar e receber mensagens em tempo real, sem a necessidade de novas requisições HTTP para cada atualização.

WebSocket

- Diferente do protocolo HTTP tradicional, que segue o modelo **requisição-resposta** (onde o cliente solicita e o servidor responde)
- WebSocket cria um **canal de comunicação contínuo** entre cliente e servidor, possibilitando interações mais eficientes e com menor latência.

Funcionamento WebSocket

1. O cliente inicia uma **conexão WebSocket** com o servidor enviando um handshake via HTTP.
2. O servidor aceita a conexão e estabelece um **canal bidirecional**.
3. Cliente e servidor podem enviar e receber mensagens **em tempo real**, sem precisar aguardar novas requisições.
4. A conexão permanece ativa até que seja **explicitamente fechada** por uma das partes.

Usos

- **Chat em tempo real** (WhatsApp Web, Slack, Discord).
- **Notificações instantâneas** (alertas de sistemas).
- **Streaming de dados financeiros** (cotações de ações em tempo real).
- **Multijogador online** (jogos que exigem resposta rápida).
- **Monitoramento de sensores IoT** (dispositivos inteligentes enviando dados contínuos).