



Quem se prepara, não para.

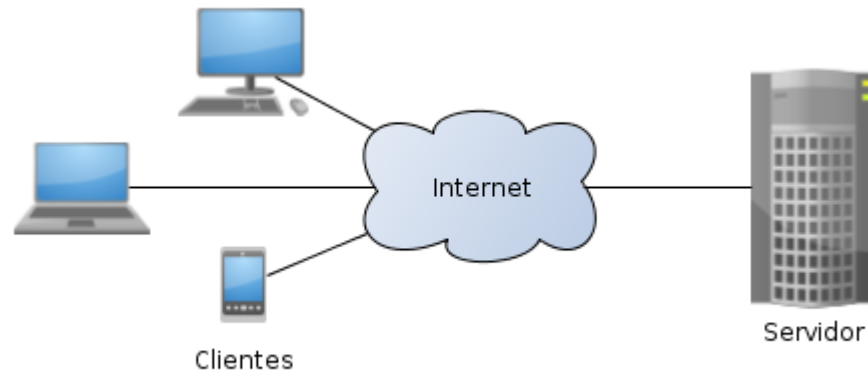
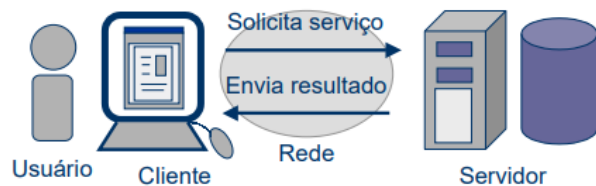
Arquitetura de Aplicações Web

5º período

Professora: Michelle Hanne

- Modelos de Arquitetura de Softwares
 - Arquitetura Cliente Servidor
 - Arquitetura em duas camadas
 - Arquitetura em três camadas
 - Arquitetura em “n” camadas
- Tecnologias para Sistemas Web
 - Webservice
 - SOAP vs REST
 - Middleware
- Documentação Exemplo de Arquitetura

Arquitetura Cliente Servidor



Arquitetura Cliente Servidor

- O modelo Cliente/Servidor, foi criado tendo como base a **descentralização** dos dados e recursos de processamento, em oposição ao modelo Centralizado (Mainframe).
- **A máquina servidor é um host** que está executando um ou mais programas de servidor que partilham os seus recursos com os clientes.
- Um cliente não compartilha de seus recursos, mas solicita o conteúdo de um servidor ou função de serviço.

Aplicações em duas camadas

- Neste modelo, um programa, normalmente desenvolvido em um ambiente como o **Visual Basic, Delphi ou Power Builder**, é instalado em cada **Cliente**. Este programa acessa dados em um **servidor de Banco de dados**.

Aplicações em duas camadas

- Camada cliente trata da lógica de negócio e da UI
- Camada servidor trata dos dados (usando um SGBD)
- **Falta de escalabilidade (conexões a bancos de dados)**
- **Enormes problemas de manutenção** (mudanças na lógica de aplicação forçava instalações)
- **Dificuldade de acessar fontes heterogêneas**

Aplicações em 3 Camadas

- Modelo em três camadas, derivado do modelo 'n' camadas, recebe esta denominação quando um **sistema cliente-servidor é desenvolvido retirando-se a camada de negócio do lado do cliente.**
- Respostas mais rápidas nas requisições, excelente performance tanto em sistemas que rodam na Internet ou em intranet e mais controle no crescimento do sistema.

Aplicações em 3 Camadas



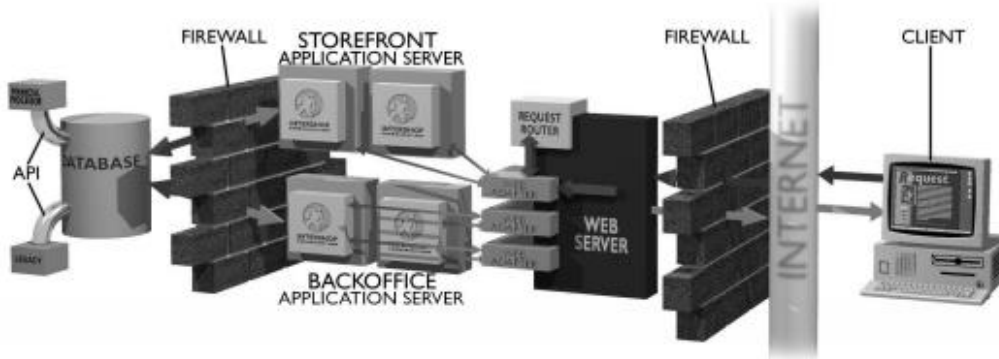
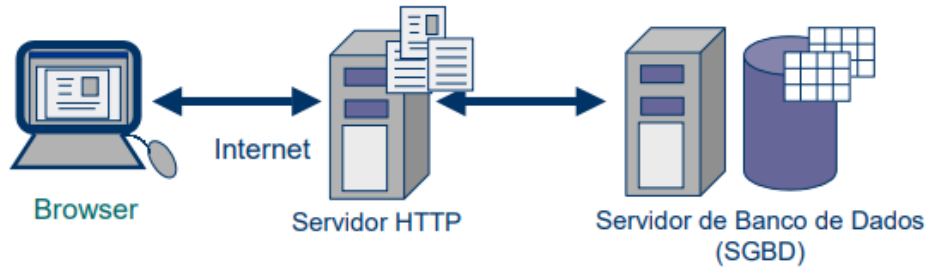
Aplicações em 3 Camadas

- **Camada de apresentação (UI)** - Continua no programa instalado no cliente.
- **Camada de aplicação (business logic)**- São as regras do negócio, as quais determinam de que maneira os dados serão utilizados. Esta camada foi deslocada para o Servidor de aplicações.
- **Camada de dados** - Nesta camada temos o servidor de Banco de dados, no qual reside toda a informação necessária para o funcionamento da aplicação.

Aplicações em 3 Camadas

- **Problemas de manutenção foram reduzidos**, pois mudanças às camadas de aplicação e de dados não necessitam de novas instalações no desktop.
- Observe que as camadas são lógicas:
 - Fisicamente, várias camadas podem executar na mesma máquina
 - Quase sempre, há separação física de máquinas
- Porém continuamos com o problema de **atualização da aplicação**, **cada vez que forem necessárias mudanças na Interface**.

Arquitetura Típica 3 Camadas



Web Services são uma forma padronizada de **integrar sistemas** onde uma aplicação oferece um **serviço HTTP (o servidor)** e a outra o **consome (cliente)**. É um método de comunicação entre máquinas (computador-computador) em uma rede. Os **Web Services** foram criados com o objetivo de possibilitar a **interoperabilidade** entre os sistemas.

Arquitetura Típica 3 Camadas

- Formulário no cliente
- Envio de dados para o servidor através da internet usando o protocolo HTTP
- Execução de programas no servidor – Existem diferentes alternativas
- Acesso a dados em um banco de dados através de um SGBD

Aplicações em 4 Camadas (*Web Based*)

O modelo de 4 camadas **retira a apresentação do cliente e centraliza em um determinado ponto**, o qual na maioria dos casos é um **servidor Web**. Com isso o próprio Cliente deixa de existir como um programa que precisa ser instalado, o acesso é feito por meio do navegador através da rede de computadores, como a Internet, por exemplo.

Aplicações em 4 Camadas (Web Based)



4 Camadas

Separa as camadas Web e Aplicação

Aplicações em 4 Camadas (Web Based)

- **Cliente:** O Cliente é o Navegador utilizado pelo usuário.
- **Apresentação:** Passa para o Servidor Web. A interface pode ser composta de páginas HTML, ASP, ou qualquer outra tecnologia capaz de gerar conteúdo para o Navegador.
- **Lógica:** São as regras do negócio, as quais determinam de que maneira os dados serão utilizados. Esta camada está no Servidor de aplicações.
- **Dados:** Nesta camada temos o servidor de Banco de dados, no qual reside toda a informação necessária para o funcionamento da aplicação.

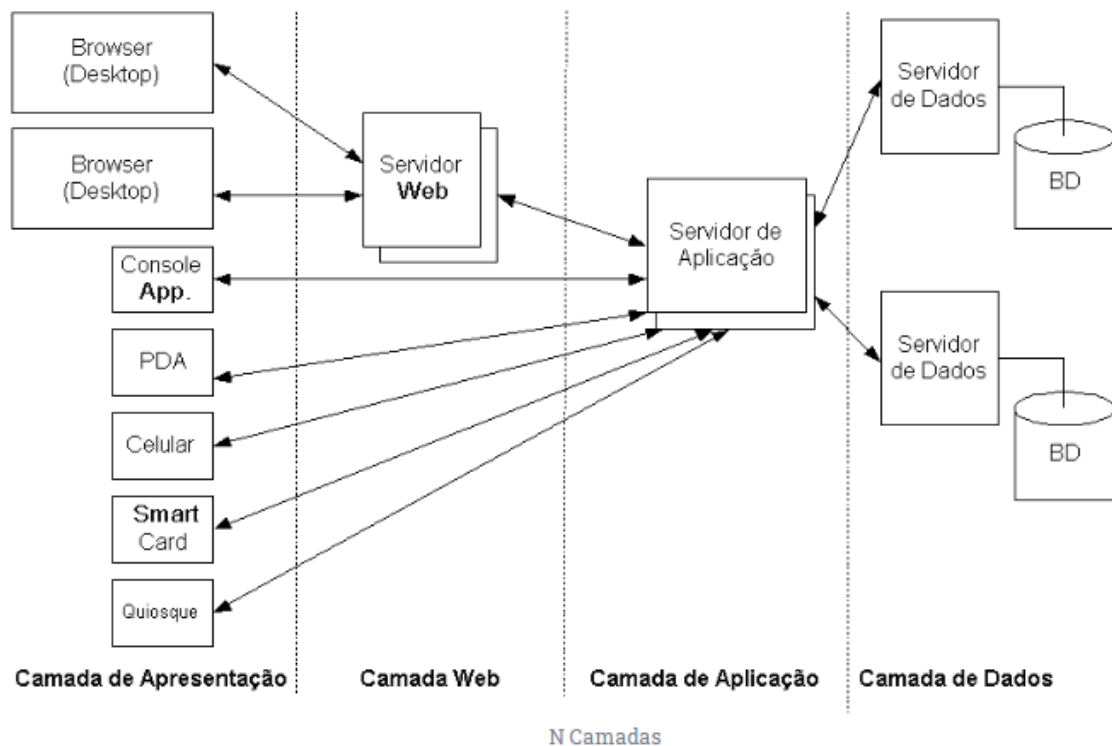
Arquitetura em N camadas

- **Os problemas remanescentes:** Não há suporte a Thin Clients (PDA, celulares, smart cards, quiosques, ...) pois preciso usar um browser (pesado) no cliente
- Dificuldade de criar software reutilizável: cadê a componentização?
- Fazer aplicações distribuídas multicamadas é difícil. Tem que:
 - Implementar persistência (*impedance mismatch* entre o mundo OO e o mundo dos BDs relacionais)
 - Implementar tolerância a falhas com fail-over
 - Implementar gerência de transações distribuídas
 - Implementar balanceamento de carga

Arquitetura em N camadas

- Uma alternativa é introduzir middleware num servidor de aplicação que ofereça esses serviços automaticamente.
- Além do mais, as soluções oferecidas (J2EE, .Net) são baseadas em componentes.
- As camadas podem ter vários nomes:
 - Apresentação, interface, cliente
 - Web
 - Aplicação, Business
 - Dados, Enterprise Information System (EIS)

Arquitetura em N camadas



Tecnologias que implementam este modelo

- **HyperText Markup Language (HTML)** – Linguagem que permite definir a estrutura de um documento a ser exibido por um browser
- **Uniform Resource Identifiers (URI)** – Esquema pelo qual os recursos da internet são endereçados
- **HyperText Transfer Protocol (HTTP)** – Protocolo que define a interação entre um browser (cliente) e um servidor de documentos hipertextuais

Tecnologias que implementam este modelo

- Uma URI identifica o mecanismo pelo qual um recurso pode ser acessado é geralmente referido como um URL (Uniform Resource Locator). **URIs HTTP são exemplos de URLs.**
- O HTTP é um protocolo da **camada de Aplicação do modelo OSI (Open System Interconnection)** utilizado para transferência de dados na Internet. É por meio deste protocolo que os recursos podem ser manipulados.

Tecnologias que implementam este modelo

Os verbos HTTP utilizados para interação com os recursos Web são:

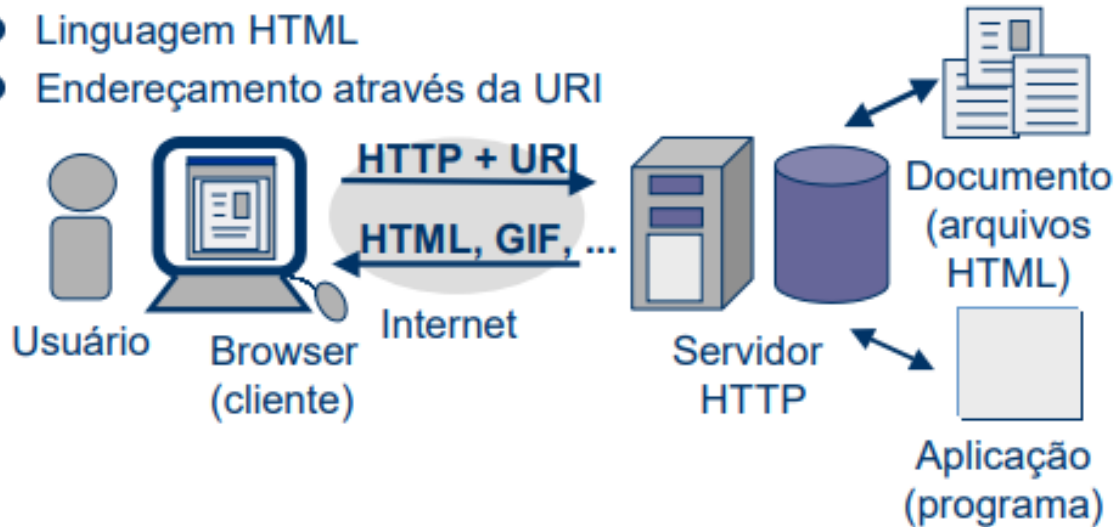
- **GET:** é utilizado para solicitar uma representação de um recurso específico e devem retornar apenas dados.
- **HEAD:** similar ao método GET, entretanto, não possui um corpo “body” contendo o recurso.
- **POST:** é utilizado para submeter uma entidade a um recurso específico, podendo causar eventualmente uma mudança no estado do recurso, ou ainda solicitando alterações do lado do servidor.
- **PUT:** substitui todas as atuais representações de seu recurso alvo pela carga de dados da requisição. **DELETE:** remove um recurso específico.
- **CONNECT:** estabelece um túnel para conexão com o servidor a partir do recurso alvo;
- **OPTIONS:** descreve as opções de comunicação com o recurso alvo.
- **TRACE:** executa uma chamada de loopback como teste durante o caminho de conexão com o recurso alvo;
- **PATCH:** aplica modificações parciais em um recurso específico

Tecnologias que implementam este modelo

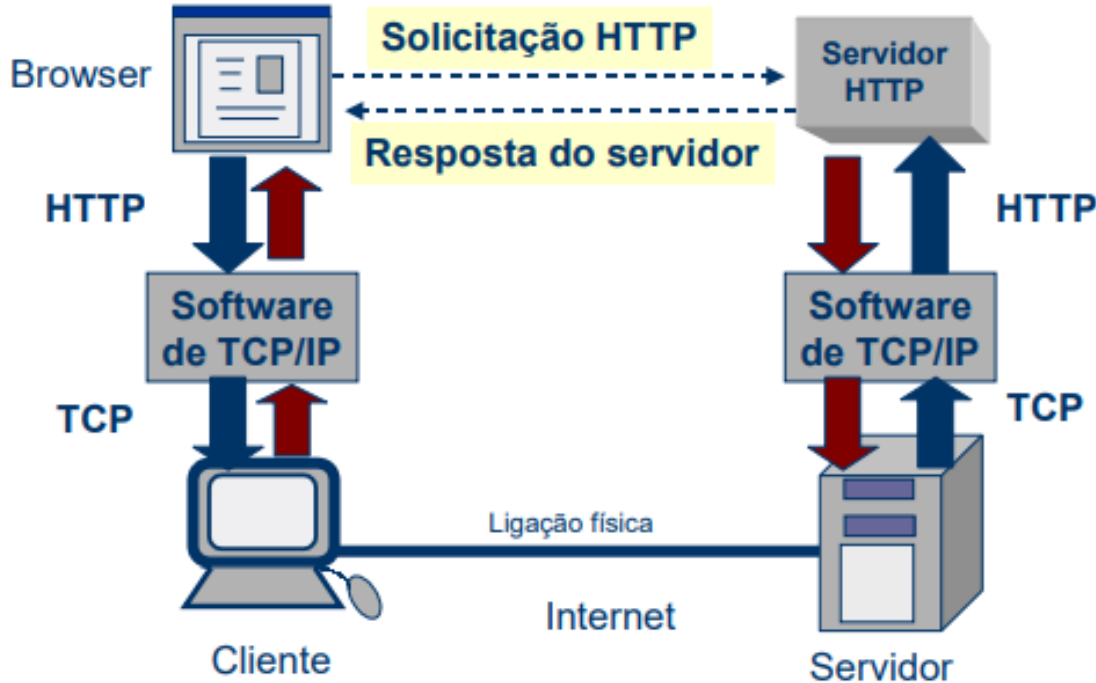
GET	URL	<i>http://localhost.8080/contacts</i>
	Descrição	Retorna uma lista com todos os contatos.
GET	URL	<i>http://localhost.8080/contacts/10</i>
	Descrição	Retorna o contato de ID 10 ou erro 404 caso o contato não seja encontrado.
GET	URL	<i>http://localhost.8080/contacts/paulo</i>
	Descrição	Retorna uma lista com os contatos que possuem a string "Paulo" no seu nome.
POST	URL	<i>http://localhost.8080/contacts</i>
	Descrição	Adiciona um novo contato. Os dados deverão ser passados no corpo da requisição no formato JSON.
PUT	URL	<i>http://localhost.8080/contacts/3</i>
	Descrição	Atualiza o contato de ID 3. Os dados deverão ser processados no corpo da requisição no formato JSON.
DELETE	URL	<i>http://localhost.8080/contacts/15</i>
	Descrição	Apaga o contato ID 15.

Modelos e tecnologias em Sistemas Web

- Arquitetura Cliente-Servidor
- Hipertexto
- Protocolo HTTP
- Linguagem HTML
- Endereçamento através da URI

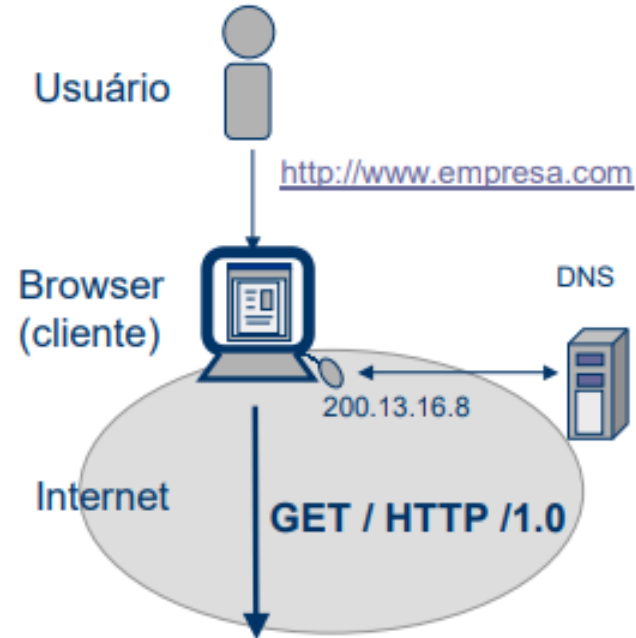


Transação cliente-servidor na Web



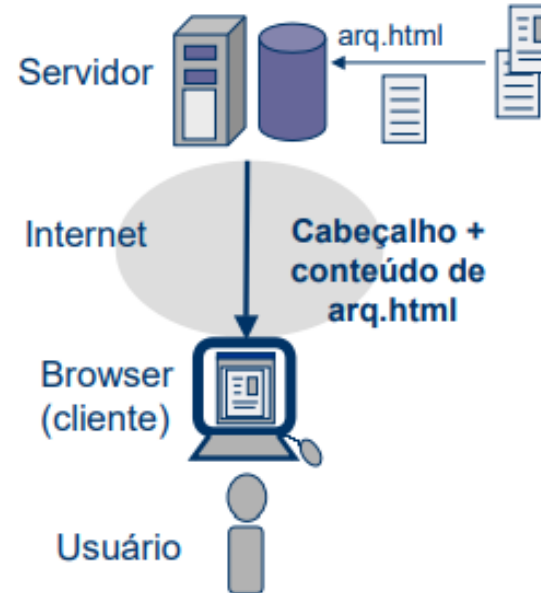
Exemplo 1 – sessão web

- 1 Usuário solicita
<http://www.empresa.com/arq.html>
- 2 DNS é consultado e fornece o endereço IP
 - 200.13.16.8
- 3 O browser faz a conexão e envia a solicitação em HTTP
 - GET /arq.html HTTP / 1.0
 - ... (seguem outras informações)

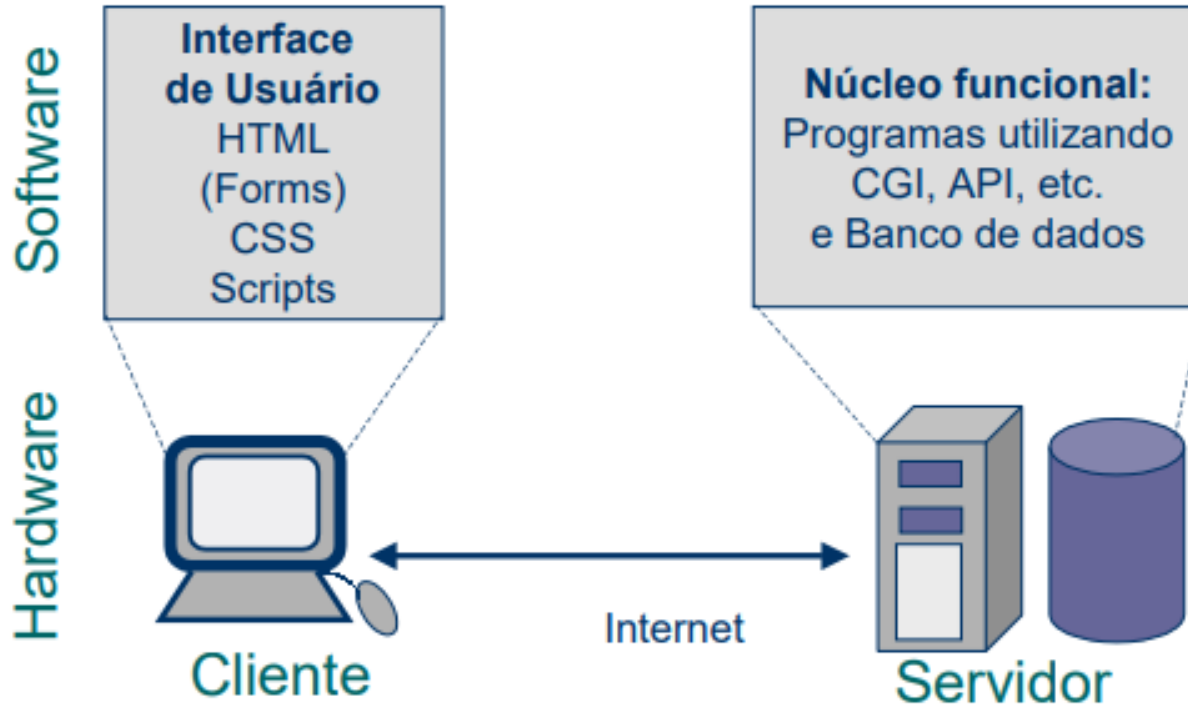


Exemplo 1 – sessão web

- 5 Servidor recebe solicitação e procura pelo recurso (arq.html)
- 6 Servidor:
HTTP/1.1 200 OK
Date: Thu, 23 Oct 1997
21:45:56 GMT
... (após o cabeçalho segue o conteúdo de arq.html)
- 7 Browser apresenta o resultado na tela



Tecnologias para Sistemas WEB



Tecnologias para Sistemas WEB- lado Cliente

- **Em 1995, a Netscape Communications** apresentou o **JavaScript**, uma linguagem de script do lado do cliente que permite aos programadores melhorar a interface e interatividade do usuário com os elementos dinâmicos.
- **Em 2005, Jesse James Garrett** propôs uma abordagem na construção de aplicações **Web** chamada **AJAX** (**Asynchronous Javascript + XML**).

Tecnologias do cliente para desenvolvimento de Sistemas Web

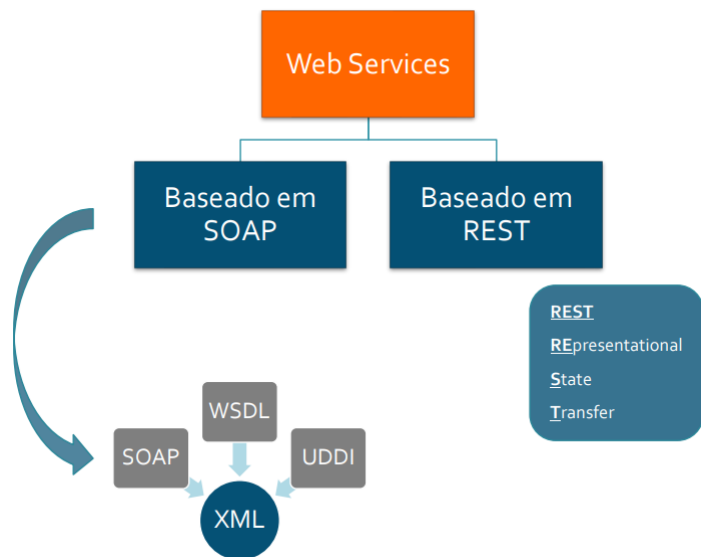
- **No modo tradicional:** interações do usuário efetuavam requisições HTTP para um servidor, que processava o pedido e retornava uma nova página HTML.
- **AJAX:** adicionar uma camada responsável por solicitar dados ao servidor e realizar todo o processamento sem a necessidade de atualizar toda a estrutura do documento HTML que estava em exibição, tornando assim a comunicação entre cliente e servidor assíncrona.

Tecnologias de servidor para desenvolvimento de Sistemas Web

- **Framework Spring** Spring fornece modelos de programação e configuração para aplicações Java EE modernas em qualquer tipo de plataforma.
- Seu principal foco é infraestrutura para o nível de aplicação, permitindo as equipes se concentrarem nas regras de negócios.

- Node.js é uma multi-plataforma um ambiente de execução de código aberto o Node.js é um ambiente de execução no servidor que permite que o [JavaScript](#) opere sem o cliente. O Node.js é de código aberto e compatível com diversas plataformas, o que o torna ideal para diferentes projetos – desde educacionais até de negócios.

Implementação de Web Services



Modelo SOAP (Simple Object Access Protocol) é o protocolo de mensagens que **especifica a forma de comunicação entre os web services e seus clientes**. Seu propósito é prover extensibilidade, neutralidade e independência. Utiliza frequentemente o protocolo HTTP (Hypertext Transfer Protocol) para intercâmbio de mensagens em formato XML.

Implementação de Web Services - SOAP

```
POST http://www.dominandoti.com.br/consultaCPF HTTP/1.1
Host: www.dominandoti.com.br
Content-Type: text/xml; charset="utf-8"
SOAPAction: "http://www.dominandoti.com.br/consultarCPF"
Content-Length: 314

<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2001/12/soap-envelope">
  <SOAP:Header>
    <!-- conteúdo do cabeçalho -->
  </SOAP:Header>
  <SOAP:Body xmlns:dti="http://www.dominandoti.com.br/tls/servicoCPF/wsdl">
    <dti:infoGestor>
      <dti:cpf>87598930104</dti:cpf>
    </dti:infoGestor>
  </SOAP:Body>
</SOAP:Envelope>
```

WSDL (**Web service Description Language**) É uma linguagem de descrição de web services em formato XML. É por meio do WSDL que o servidor especifica os serviços que deseja expor na rede para quem quiser consultá-los

UDDI (**Universal Description Discovery and Integration**) É um protocolo que foi criado para facilitar a arquitetura orientada a serviços via web services. O UDDI é um mecanismo que permite que os servidores registrem (ou publiquem) seus webservices em um diretório de forma que os clientes possam encontrá-los com facilidade.

Implementação de Web Services - Rest



O modelo REST (**Representational State Transfer**) é uma outra abordagem, uma alternativa ao SOAP, que estudamos até aqui. O REST é mais que um protocolo. É um estilo arquitetural. Trata-se de um padrão aberto, não proprietário de nenhuma empresa. As principais propriedades do REST são:

Implementação de Web Services

Comparativamente ao **SOAP**, as principais vantagens do **REST** são a sua **simplicidade**, **baixo overhead de comunicação**, **alta escalabilidade** (manter a boa performance mesmo quando o número de requisições aumenta muito)

XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```
{ "empinfo" :
  {
    "employees" : [
      {
        "name" : "James Kirk",
        "age" : 40,
      },
      {
        "name" : "Jean-Luc Picard",
        "age" : 45,
      },
      {
        "name" : "Wesley Crusher",
        "age" : 27,
      }
    ]
  }
}
```

Arquitetura SOAP vs REST

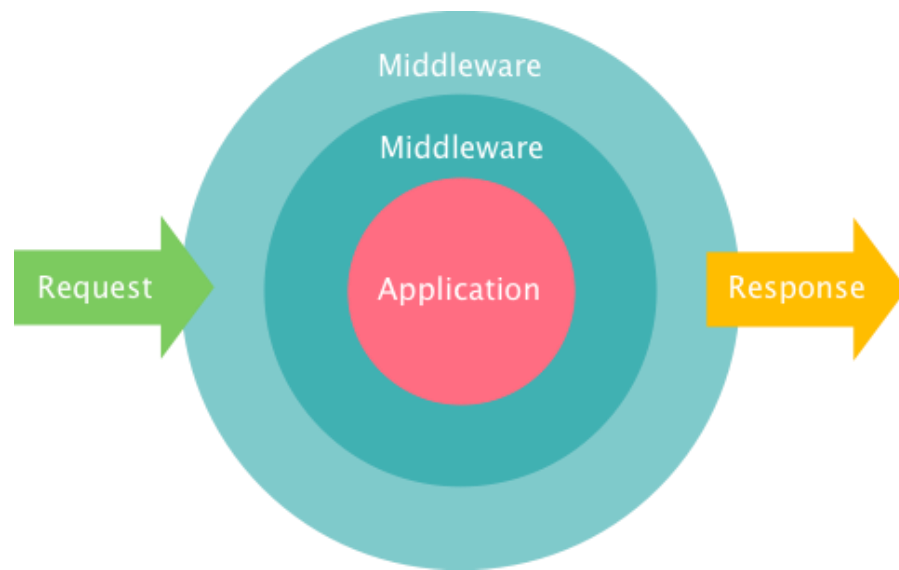
Restrições arquiteturais REST

REST	SOAP
+ simples	- simples
- <i>overhead</i> de comunicação	+ <i>overhead</i> de comunicação
<i>Uniform contract</i>	Cada serviço tem seu contrato WSDL específico
<i>Stateless</i>	<i>Stateful</i>
Sem controle de transação	Com controle de transação
- segurança	+ segurança

Restrição	Descrição
Arquitetura cliente-servidor	O servidor é o web service. O cliente pode ser um aplicativo qualquer como um app de smartphone, um sistema Windows, Linux, Mac ou até mesmo outro web service.
<i>Stateless</i>	No lado do servidor, não armazena estado entre as requisições. Cada requisição do cliente contém toda informação necessária para seu atendimento por parte do serviço.
<i>Cacheability</i>	Possibilidade de fazer cache de alguns tipos de resposta de forma a retornar o resultado para o cliente de forma mais ágil.
Sistema em Camadas	Construção de diversos níveis de abstração incluindo, além do cliente e do serviço, camadas intermediárias como <i>proxy</i> e balanceadores de carga.
Código sob demanda	Servidores podem estender funcionalidades dos clientes ao transferir, além de dados, códigos executáveis.
Interface uniforme	Forma padronizada e unificada de os clientes acessarem os serviços (<i>uniform contract</i>).

Middleware

Um middleware é uma camada de software, residente acima do sistema operacional e do substrato de comunicação, que oferece abstrações de alto nível, com objetivo de facilitar a programação distribuída.



O middleware API (interface de programação de aplicativos): fornece ferramentas que os desenvolvedores podem usar para criar, expor e gerenciar APIs para seus aplicativos

Middleware

4 camadas de Middleware no desenvolvimento de Aplicações:

Camada de containers

A camada de containers fornece recursos de [DevOps](#) com CI/CD (*continuous integration/continuous delivery*), gerenciamento de service mesh e de containers. Ela é quem **gerencia o aspecto de entrega de vida das aplicações de forma uniforme**.

Camada de ambientes de execução

A camada de ambientes de execução **permite a execução de código personalizado**.

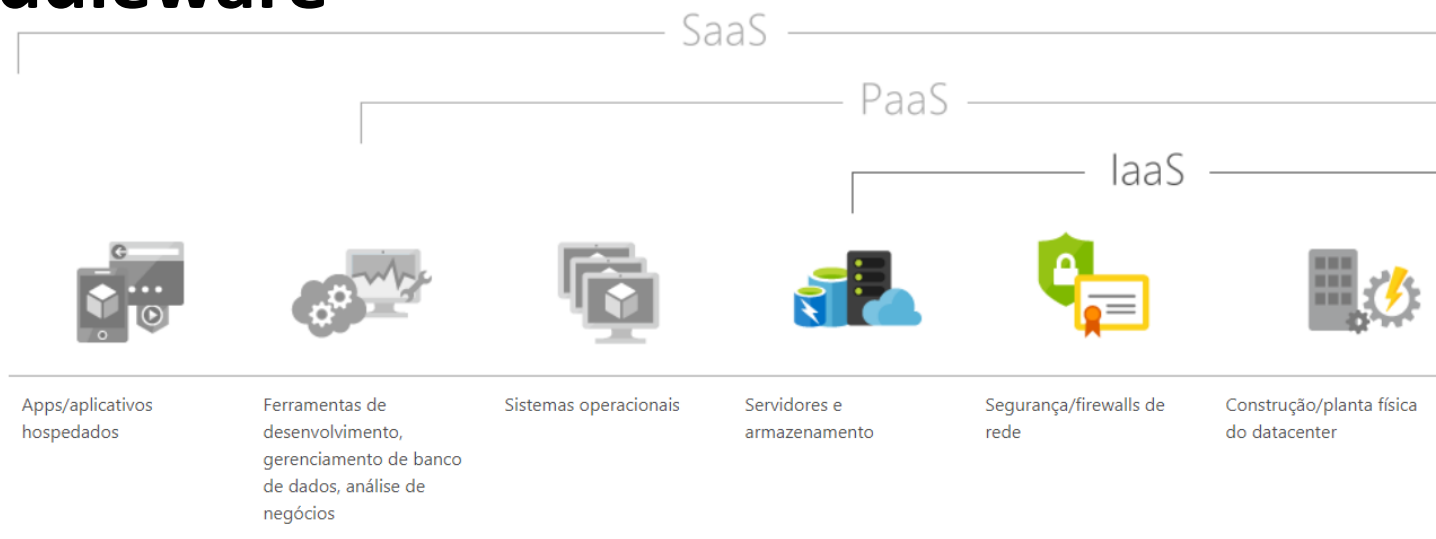
Camada de integração

É por meio de **integração de [APIs](#) e por meio de mensageria** que o middleware fornece serviços para conectar aplicações adquiridas e personalizadas, como ativos de [SaaS](#) (*Software as a Service*).

Camada de automação de processos e gerenciamento de decisões

A camada final introduz funções críticas de otimização, inteligência, gerenciamento de decisões e automação.

Middleware



Hoje em dia, um modelo hospedado em nuvem ***chamado plataforma de integração como serviço***, ou **iPaaS**, permite que uma organização conecte aplicativos, dados, processos e serviços em ambientes locais, **em nuvem privada e em nuvem pública** — sem o trabalho e despesas de adquirir, instalar, gerenciar e manter o middleware de integração (e o hardware no qual ele é executado) em seu próprio data center

Middleware

Praticamente falando, atualmente, os aplicativos nativos da nuvem são aplicativos desenvolvidos a partir de **microserviços** e implantados em **contêineres** orquestrados usando [Kubernetes](#).

Os **microserviços** são componentes de aplicativo **fracamente acoplados** que abrangem sua própria pilha e podem ser implementados e atualizados independentemente uns dos outros e se comunicam usando uma combinação de APIs REST, corretores de mensagens e fluxos de eventos.

Documentação de Arquitetura

Exemplo de Documentação de Arquitetura

- [https://github.com/LeonardoKalyn/Red-Button/wiki/Documento-de-Arquitetura-de-Software-\(DAS\)](https://github.com/LeonardoKalyn/Red-Button/wiki/Documento-de-Arquitetura-de-Software-(DAS))

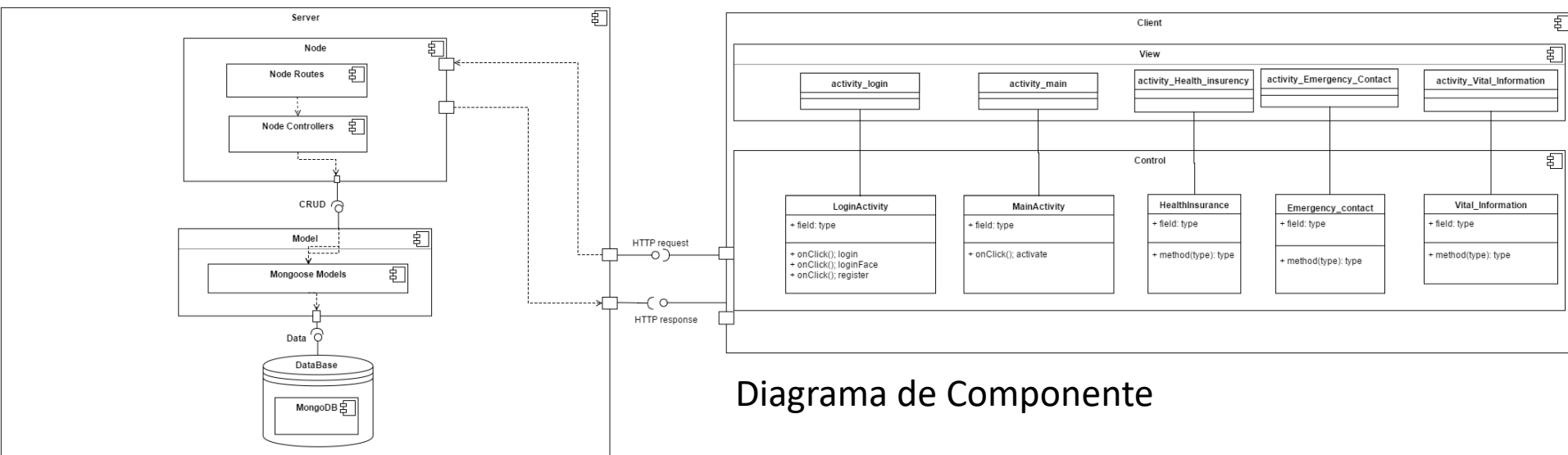


Diagrama de Componente