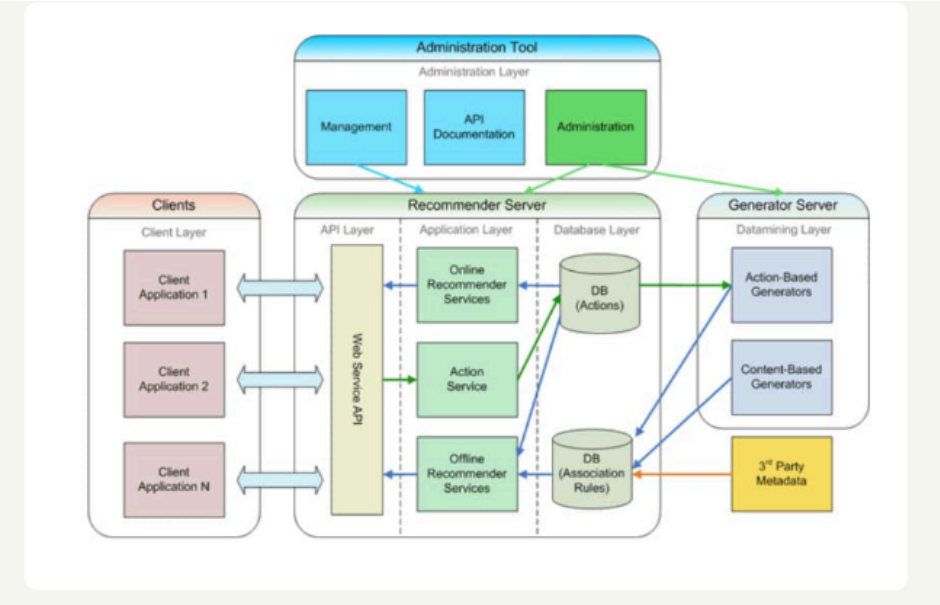
Arquitetura de aplicativos web

Site:	Digital College	Impresso por:	LEONARDO
Curso:	Desenvolvimento Web Full Stack - Turma 48 - Aldeota	Data:	terça-feira, 4 fev. 2025, 21:46
Livro:	Arquitetura de aplicativos web		

Índice

1. Arquitetura de aplicativos web

A palavra aplicativo tem um amplo significado no jargão tecnológico. Quando o aplicativo é um programa tradicional, executado localmente e auto-suficiente em sua finalidade, tanto a interface operacional do aplicativo quanto os componentes de processamento de dados são integrados em um único "pacote". Um aplicativo web é diferente porque adota o modelo cliente/servidor e sua parte cliente é baseada em HTML, obtido do servidor e, em geral, processado por um navegador.



1.1. Clientes e servidores

No modelo cliente/servidor, parte do trabalho é feito localmente no lado do cliente e parte do trabalho é feito remotamente, no lado do servidor. As tarefas realizadas por cada parte variam de acordo com a finalidade do aplicativo, mas em geral cabe ao cliente fornecer uma interface para o usuário e exibir o conteúdo de forma atraente. Cabe ao servidor executar a parte operacional do aplicativo, processando e respondendo às solicitações feitas pelo cliente. Em um aplicativo de compras, por exemplo, o aplicativo cliente apresenta uma interface para o usuário escolher e pagar pelos produtos, mas a fonte de dados e os registros da transação são mantidos no servidor remoto, acessado pela rede. Os aplicativos web realizam essa comunicação pela internet, geralmente por meio do Protocolo de Transferência de Hipertexto (HTTP).

Depois de carregado pelo navegador, o lado do cliente do aplicativo inicia a interação com o servidor sempre que necessário ou conveniente. Os servidores de aplicativos web oferecem uma interface de programação de aplicativos (API) que define as solicitações disponíveis e como devem ser feitas. Assim, o cliente constrói uma solicitação no formato definido pela API e a envia ao servidor, que verifica os pré-requisitos da solicitação e envia de volta a resposta apropriada.

Enquanto o cliente, na forma de um aplicativo móvel ou navegador de desktop, é um programa independente em relação à interface do usuário e às instruções para se comunicar com o servidor, o navegador deve obter a página HTML e os componentes associados — como imagens, CSS e JavaScript — que definem a interface e as instruções para comunicação com o servidor.

As linguagens de programação e as plataformas usadas por cliente e servidor são independentes, mas empregam um protocolo de comunicação mutuamente compreensível. A parte do servidor é quase sempre realizada por um programa sem interface gráfica, rodando em ambientes de computação de alta disponibilidade, de forma a estar sempre pronto para responder às solicitações. Já a parte do cliente é executada em qualquer dispositivo capaz de renderizar uma interface HTML, como os smartphones.

Além de ser imprescindível para determinadas finalidades, a adoção do modelo cliente/servidor permite que um aplicativo otimize diversos aspectos de desenvolvimento e manutenção, já que cada parte pode ser projetada para sua função específica. Um aplicativo que exibe mapas e rotas, por exemplo, não precisa ter todos os mapas armazenados localmente. São necessários apenas os mapas relativos à localização que interessa ao usuário e, portanto, somente esses mapas são solicitados ao servidor central.

Os desenvolvedores têm controle direto sobre o servidor; assim, eles também podem modificar o cliente fornecido por ele. Isso permite que os desenvolvedores aprimorem o aplicativo, em maior ou menor grau, sem que o usuário precise formalmente instalar novas versões.

1.2. O lado do cliente

Um aplicativo web deve ser executado da mesma maneira em todos os navegadores mais populares, desde que o navegador esteja atualizado. Alguns navegadores podem ser incompatíveis com as inovações recentes, mas apenas os aplicativos experimentais usam recursos ainda não amplamente adotados.

Os problemas de incompatibilidade eram mais comuns no passado, quando cada navegador tinha seu próprio motor de renderização e havia menos cooperação na formulação e adoção de padrões. O motor (ou mecanismo) de renderização é o principal componente do navegador, pois é responsável por transformar o HTML e outros componentes associados nos elementos visuais e interativos da

interface. Alguns navegadores, sobretudo o Internet Explorer, necessitavam de um tratamento especial no código para não quebrar o funcionamento esperado das páginas.

Hoje, as diferenças entre os navegadores principais são mínimas e as incompatibilidades, raras. Na verdade, os navegadores Chrome e Edge usam o mesmo mecanismo de renderização (chamado Blink). O navegador Safari e outros oferecidos na iOS App Store usam o mecanismo WebKit. O Firefox usa um mecanismo chamado Gecko. Esses três mecanismos são responsáveis por praticamente todos os navegadores usados hoje. Embora desenvolvidos separadamente, os três motores são projetos de código aberto e há cooperação entre seus desenvolvedores, o que facilita a compatibilidade, a manutenção e a adoção de padrões.

Como os desenvolvedores de navegadores se esforçaram muito para preservar a compatibilidade, o servidor normalmente não está vinculado a um único tipo de cliente. Em princípio, um servidor HTTP pode se comunicar com qualquer cliente que também seja capaz de se comunicar via HTTP. Em um aplicativo de mapa, por exemplo, o cliente pode ser um aplicativo móvel ou um navegador que carrega a interface HTML do servidor.

1.3. Variedades de clientes web

Existem aplicativos móveis e de desktop cuja interface é renderizada a partir de HTML e, como os navegadores, podem usar JavaScript como linguagem de programação. Porém, ao contrário do cliente carregado no navegador, o HTML e os componentes necessários para o funcionamento de um cliente nativo estão presentes localmente desde a instalação do aplicativo. Na verdade, um aplicativo que funciona dessa maneira é praticamente idêntico a uma página HTML (é provável que ambos sejam renderizados pelo mesmo mecanismo). Existem também os aplicativos web progressivos (Progressive Web Apps ou PWA), um mecanismo que permite empacotar clientes de aplicativos web para uso offline — limitado a funções que não requerem comunicação imediata com o servidor. Em relação às captabilidades do aplicativo, não há diferença entre rodá-lo no navegador ou empacotado em um PWA; porém, neste último, o desenvolvedor tem mais controle sobre o que é armazenado localmente.

Renderizar interfaces HTML é uma atividade tão recorrente que o mecanismo é geralmente um componente de software separado, presente no sistema operacional. Sua presença independente permite que diferentes aplicativos o incorporem sem precisar incluí-lo no pacote do aplicativo. Esse modelo também delega a manutenção do motor de renderização ao sistema operacional, facilitando as atualizações. É muito importante manter esse componente crucial sempre atualizado para evitar possíveis falhas.

Independentemente do método de fornecimento, os aplicativos escritos em HTML são executados em uma camada de abstração criada pelo mecanismo, que funciona como um ambiente de execução isolado. Em particular, no caso de um cliente que roda no navegador, o aplicativo tem à sua disposição apenas os recursos oferecidos pelo navegador. Recursos básicos, como a interação com elementos de página e solicitação de arquivos por HTTP, estão sempre disponíveis. Os recursos que podem conter informações confidenciais, como o acesso a arquivos locais, a localização geográfica, câmera e microfone, requerem uma autorização explícita do usuário antes que o aplicativo possa usá-los.

1.4. As linguagens de um cliente web

O elemento central de um cliente de aplicativo web executado no servidor é o documento HTML. Além de apresentar de forma estruturada os elementos da interface exibidos pelo navegador, o documento HTML contém os endereços de todos os arquivos necessários para a apresentação e funcionamento corretos do cliente.

O HTML sozinho não tem muita versatilidade para construir interfaces mais elaboradas, nem recursos de programação de finalidade geral. Por esse motivo, um documento HTML que deve funcionar como um aplicativo cliente vem sempre acompanhado por um ou mais conjuntos de CSS e JavaScript.

O CSS pode ser fornecido como um arquivo separado ou diretamente no próprio arquivo HTML. O principal objetivo do CSS é refinar a aparência e o layout dos elementos da interface HTML. Embora isso não seja estritamente necessário, as interfaces mais sofisticadas geralmente requerem modificações nas propriedades CSS dos elementos para atender às suas necessidades.

O JavaScript é um componente praticamente indispensável. Os procedimentos escritos em JavaScript respondem a eventos no navegador. Esses eventos podem ser causados pelo usuário ou ser nãointerativos. Sem o JavaScript, um documento HTML fica praticamente limitado a texto e imagens. O uso do JavaScript em documentos HTML permite estender a interatividade muito além de hiperlinks e formulários, transformando a página exibida pelo navegador em uma interface de aplicativo convencional.

O JavaScript é uma linguagem de programação de propósito geral, mas seu principal uso é em aplicativos web. Os recursos do ambiente de execução do navegador são acessíveis por meio de palavras-chave em JavaScript, utilizadas em um script para realizar a operação desejada. O termo document, por exemplo, é usado no código JavaScript para se referir ao documento HTML associado a ele. No contexto da linguagem JavaScript, document é um objeto global com propriedades e métodos que podem ser usados para obter informações de qualquer elemento no documento HTML. Além disso, podemos usar o objeto document para modificar seus elementos e associá-los a ações personalizadas escritas em JavaScript.

Um aplicativo cliente baseado em tecnologias web é multiplataforma, pois pode ser executado em qualquer dispositivo que possua um navegador compatível.

Porém, o fato de estarem confinados ao navegador impõe limitações aos aplicativos web em comparação com os aplicativos nativos. A intermediação realizada pelo navegador permite uma programação de alto nível e aumenta a segurança, mas também aumenta as exigências de processamento e o consumo de memória.

Os desenvolvedores estão continuamente aprimorando os navegadores para fornecer mais recursos e melhorar o desempenho dos aplicativos em JavaScript, mas existem aspectos intrínsecos à execução de scripts como o JavaScript que os deixam em desvantagem na comparação com programas nativos para o mesmo hardware.

Um recurso que melhora bastante o desempenho dos aplicativos JavaScript em execução no navegador é o WebAssembly. O WebAssembly é um tipo de JavaScript compilado que produz códigofonte escrito em uma linguagem mais eficiente de nível inferior, como a linguagem C. O WebAssembly pode acelerar principalmente as atividades de uso intensivo do processador, pois evita grande parte da tradução realizada pelo navegador ao executar um programa escrito em JavaScript convencional.

Independentemente dos detalhes de implementação do aplicativo, todos os códigos HTML, CSS, JavaScript e arquivos multimídia devem primeiro ser obtidos do servidor. O navegador obtém esses arquivos como se fosse uma página da internet, ou seja, por meio de um endereço acessado pelo navegador.

Uma página web que atua como uma interface para um aplicativo web é como um documento HTML simples, mas com comportamentos adicionais. Nas páginas convencionais, o usuário é direcionado para outra página ao clicar em um link. Os aplicativos web podem apresentar sua interface e responder aos eventos do usuário sem carregar novas páginas na janela do navegador. A modificação desse comportamento padrão das páginas HTML é feita por meio da programação em JavaScript.

Um cliente de webmail, por exemplo, exibe as mensagens e alterna entre as pastas sem sair da página. Isso é possível porque o cliente usa JavaScript para reagir às ações do usuário e fazer solicitações apropriadas ao servidor. Se o usuário clicar no assunto de uma mensagem na caixa de entrada, um código JavaScript associado a esse evento solicitará o conteúdo dessa mensagem ao servidor (usando a chamada API correspondente). Assim que o cliente recebe a resposta, o navegador exibe a mensagem na parte apropriada da mesma página. Clientes de webmail diferentes podem adotar estratégias diferentes, mas todos empregam o mesmo princípio.

Portanto, além de fornecer ao navegador os arquivos que compõem o cliente, o servidor também deve ser capaz de atender a solicitações como a do cliente de webmail quando solicita o conteúdo de uma determinada mensagem. Cada requisição que o cliente pode fazer está vinculada a um procedimento predefinido de resposta no servidor, cuja API pode definir diferentes métodos para identificar o procedimento ao qual a requisição se refere. Os métodos mais comuns são:

- Endereços, através de um Uniform Resource Locator (URL)
- Campos no cabeçalho HTTP
- Métodos GET/POST
- WebSockets

Um método pode ser mais adequado do que outro, dependendo da finalidade da solicitação e de outros critérios levados em consideração pelo desenvolvedor. Em geral, os aplicativos web usam uma combinação de métodos, cada um em uma circunstância específica.

O paradigma Representational State Transfer (REST) é amplamente utilizado para a comunicação nos aplicativos web, pois se baseia nos métodos básicos disponíveis em HTTP. O cabeçalho de uma solicitação HTTP começa com uma palavra-chave que define a operação básica a ser realizada: GET, POST, PUT, DELETE, etc., acompanhada por uma URL correspondente na qual a ação será aplicada. Se o aplicativo requer operações mais específicas, com uma descrição mais detalhada da operação solicitada, o protocolo GraphQL pode ser uma escolha mais adequada.

Os aplicativos desenvolvidos no modelo cliente/servidor estão sujeitos a instabilidades de comunicação. Por isso, o aplicativo cliente deve sempre adotar estratégias eficientes de transferência de dados para favorecer sua consistência e não prejudicar a experiência do usuário.

1.5. Controlando os caminhos de solicitações

Os servidores HTTP, como o Apache e o NGINX, costumam precisar de alterações específicas de configuração para atender às necessidades do aplicativo. Por padrão, os servidores HTTP tradicionais associam diretamente o caminho indicado na solicitação a um arquivo no sistema de arquivos local.

Se o servidor HTTP de um website mantiver seus arquivos HTML no diretório /srv/www, por exemplo, uma solicitação com o caminho /en/about.html receberá o conteúdo do arquivo /srv/www/en/about.html como resposta, se o arquivo existir. Os sites mais sofisticados, e os aplicativos web em especial, exigem tratamentos personalizados para diferentes tipos de solicitações. Nesse cenário, parte da implementação do aplicativo consiste em modificar as configurações do servidor HTTP para atender aos requisitos do aplicativo.

Como alternativa, existem frameworks (estruturas) que permitem integrar o gerenciamento das solicitações HTTP e a implementação do código do aplicativo em um só lugar, permitindo que o desenvolvedor se concentre mais na finalidade do aplicativo do que nos detalhes da plataforma. No Node.js Express, por exemplo, todo o mapeamento de solicitações e a programação correspondente são implementados usando JavaScript. Como a programação dos clientes geralmente é feita em JavaScript, muitos desenvolvedores consideram uma boa ideia, do ponto de vista da manutenção do código, usar a mesma linguagem para o cliente e o servidor. Outras linguagens comumente usadas para implementar o lado do servidor, seja em frameworks ou em servidores HTTP tradicionais, são PHP, Python, Ruby, Java e C#.