TEMA 01
Introdução ao Javascript
Habilidades

- Introdução as regras JavaScript
- Utilizar ambientes de desenvolvimento de software

Introdução ao JavaScript

Objetivos:
- Compreender as regras básicas do JavaScript.
- Utilizar ambientes de desenvolvimento de software.

Origem do JavaScript:
JavaScript foi criado em 1995 por Brendan Eich enquanto trabalhava para a Netscape. Na época, a web estava em seus primeiros passos e o HTML era a principal ferramenta para criar sites. A ideia de Eich foi criar uma linguagem de script simples e dinâmica para adicionar interatividade aos navegadores, o que levou ao desenvolvimento do JavaScript. Inicialmente chamado "Mocha", o JavaScript evoluiu rapidamente e se tornou uma parte essencial da web.

Evolução e Padronização:
Em 1996, a Netscape colaborou com a ECMA International para padronizar o JavaScript, resultando na criação do ECMAScript. Isso estabeleceu as bases para a linguagem e garantiu a sua evolução constante. Vale destacar que JavaScript e Java não são a mesma coisa e não têm relação direta entre si.

JavaScript Hoje:
JavaScript é uma linguagem de programação muito versátil e não se limita apenas à web. Ele é usado em várias plataformas, incluindo motores gráficos como Unity 3D. A linguagem é conhecida por sua flexibilidade e por permitir a criação de uma ampla gama de aplicações.

Componentes Principais do JavaScript:

1. Front-End vs. Back-End:
- **Front-End:** Refere-se à parte visível e interativa de um site ou aplicativo com a qual os usuários interagem diretamente. Utiliza HTML, CSS e JavaScript para criar interfaces e melhorar a experiência do usuário.
- **Back-End:** Refere-se à parte que lida com a lógica, processamento e gerenciamento de dados no servidor. Envolve a criação e manutenção de servidores, bancos de dados e APIs que suportam as funcionalidades do front-end.

2. Documentos JavaScript:
- São arquivos que contêm código JavaScript incorporado em páginas HTML para adicionar interatividade e funcionalidades dinâmicas. Eles permitem a manipulação do DOM, validação de dados, requisições assíncronas e muito mais.

Sintaxe JavaScript:

- Declaração de Variáveis: Utilize `var`, `let` ou `const` para declarar variáveis. As variáveis podem conter letras, números e sublinhados, mas não podem começar com um número.

- Comentários: Comentários ajudam a documentar o código. Use `//` para comentários de linha única e `/* */` para comentários de múltiplas linhas.

- Instruções e Expressões: Instruções executam ações e podem incluir expressões. Expressões produzem valores.

- Estruturas de Controle: `if`, `for` e `while` permitem controlar o fluxo do programa.

- Funções: Blocos de código reutilizáveis que podem receber parâmetros e retornar valores.

- Objetos e Arrays: Objetos são coleções de pares chave-valor e arrays são listas ordenadas de valores.

- Operadores: Utilizados para operações matemáticas, comparações e lógicas.

Comandos em Inglês:

Os comandos em JavaScript são em inglês devido à sua origem histórica e à predominância do inglês na tecnologia da informação. Isso facilita a colaboração global, integração com outras tecnologias e acesso a documentação e recursos educacionais.

Desenvolvimento de Aplicativos Móveis:

Para se tornar um desenvolvedor de aplicativos móveis, é essencial aprender:
- Linguagens de Programação: Swift (iOS), Kotlin (Android), JavaScript (React Native), Dart (Flutter).
- Ambientes de Desenvolvimento: Xcode (iOS), Android Studio (Android), Visual Studio Code (multiplataforma).
- Frameworks e Bibliotecas: React Native, Flutter, NativeScript.
- Design Responsivo e UI/UX: Princípios para criar interfaces eficazes.
- Consumo de APIs e Armazenamento de Dados: Integração com serviços externos e armazenamento local.
- Testes, Depuração e Publicação: Garantir a qualidade do aplicativo e distribuí-lo em lojas de aplicativos.

Ambiente de Desenvolvimento:

Para programar em JavaScript, é importante escolher um editor de código apropriado, como Sublime, Notepad++ ou Visual Studio Code (VSCode). Cada editor tem suas características, e a escolha pode depender das preferências pessoais e necessidades do projeto.

ECMA e JavaScript: Relação e Contexto

1. O que é a ECMA?

A ECMA International (European Computer Manufacturers Association) é uma organização internacional que trabalha para estabelecer padrões em tecnologia da informação e comunicação. Fundada em 1961, a ECMA é conhecida por criar e manter normas para várias tecnologias, incluindo linguagens de programação.

2. Origem do JavaScript e ECMA

JavaScript foi criado em 1995 por Brendan Eich enquanto trabalhava para a Netscape Communications Corporation. Inicialmente chamado "Mocha", o JavaScript evoluiu rapidamente e foi renomeado para "JavaScript" antes de seu lançamento oficial.

Na época de seu lançamento, a linguagem não tinha um padrão formal, o que significava que diferentes navegadores poderiam implementar JavaScript de maneiras diferentes, levando a inconsistências e problemas de compatibilidade.

3. O Papel da ECMA na Padronização do JavaScript

Para resolver as questões de compatibilidade e promover um padrão unificado, a Netscape decidiu colaborar com a ECMA International. Em novembro de 1996, a ECMA começou a trabalhar na padronização do JavaScript, resultando no ECMAScript, um padrão formal para a linguagem.

4. ECMAScript: O Padrão para JavaScript

- **ECMAScript:** É o nome dado ao padrão definido pela ECMA para JavaScript. O padrão é formalizado no documento chamado **ECMA-262**. O ECMAScript define a sintaxe, o comportamento e as APIs básicas da linguagem, garantindo que os implementadores (como navegadores e outros ambientes) sigam um conjunto comum de regras.

5. Importância da Padronização

A padronização pelo ECMAScript é crucial para:
- **Compatibilidade:** Garante que o código JavaScript funcione de forma consistente em diferentes ambientes e navegadores.
- **Desenvolvimento:** Facilita a adoção de novas funcionalidades e técnicas, sabendo que elas serão suportadas amplamente.
- **Interoperabilidade:** Permite que bibliotecas e frameworks trabalhem de maneira uniforme em diversos ambientes.

6. Conclusão

A ECMA International e seu padrão ECMAScript desempenham um papel fundamental na evolução e na padronização do JavaScript. Ao estabelecer um padrão formal, a ECMA garante que o JavaScript possa ser usado de maneira consistente e confiável em diversos navegadores e plataformas, ajudando a linguagem a se manter relevante e funcional à medida que a web e as tecnologias relacionadas evoluem.

Node.js

1. O que é Node.js?

Node.js é um ambiente de execução de JavaScript que permite executar código JavaScript fora do navegador. Baseado no motor V8 do Google Chrome, Node.js usa um modelo de I/O não-bloqueante e orientado a eventos, o que o torna ideal para aplicações de alta performance e escalabilidade.

2. Principais Características do Node.js

- **Assíncrono e Não-Bloqueante:** Permite que as operações de entrada/saída (I/O) sejam realizadas de forma assíncrona, sem bloquear o fluxo do programa. Isso é útil para aplicações que precisam lidar com várias conexões simultaneamente, como servidores web.

- **Event-Driven:** Utiliza um modelo orientado a eventos, onde as ações são tratadas por "event listeners" que respondem a eventos específicos.

- **Single-Threaded:** Embora use um único thread para a execução do código, Node.js pode lidar com múltiplas conexões ao usar um loop de eventos e operações de I/O assíncronas.

- **NPM (Node Package Manager):** Inclui um gerenciador de pacotes, o npm, que fornece acesso a uma vasta biblioteca de módulos e pacotes para estender as funcionalidades do Node.js.

3. Uso Comum

- **Desenvolvimento de Servidores e APIs:** Node.js é frequentemente usado para criar servidores web e APIs RESTful.

- **Aplicações em Tempo Real:** Ideal para aplicações que requerem comunicação em tempo real, como chats e jogos online.

- **Ferramentas de Desenvolvimento:** Usado em ferramentas de build e sistemas de gerenciamento de pacotes.

React Native com Expo

1. O que é React Native?

React Native é um framework para desenvolvimento de aplicativos móveis que permite criar aplicações para iOS e Android usando JavaScript e React. Em vez de renderizar código em HTML, como na web, o React Native compila os componentes para widgets nativos.

2. Principais Características do React Native

- **Desenvolvimento Multiplataforma:** Permite criar aplicativos nativos para diferentes plataformas com um único código base.

- **Componentes Nativos:** Oferece componentes que são renderizados como widgets nativos em dispositivos móveis, proporcionando uma experiência de usuário semelhante a aplicações nativas.

- **Hot Reloading:** Permite que desenvolvedores vejam as mudanças no código imediatamente sem reiniciar o aplicativo.

3. O que é Expo?

Expo é uma plataforma e um conjunto de ferramentas para o desenvolvimento de aplicativos React Native. Ele fornece uma série de bibliotecas e utilitários que facilitam a criação, o teste e o lançamento de aplicativos React Native.

4. Principais Características do Expo

- Ambiente de Desenvolvimento Simplificado: Fornece uma configuração inicial fácil e uma série de ferramentas prontas para uso, o que reduz a complexidade do processo de configuração.

- Expo Go: Um aplicativo móvel que permite testar e visualizar o seu aplicativo diretamente no dispositivo sem a necessidade de compilação adicional.

- APIs e Bibliotecas: Inclui uma série de APIs e bibliotecas para funcionalidades comuns, como acesso à câmera, localização e notificações, sem a necessidade de configuração nativa complexa.

- Desenvolvimento e Build: Oferece serviços para build e deploy de aplicativos, incluindo a criação de pacotes e distribuição nas lojas de aplicativos.

Relação Entre Node.js e React Native com Expo

Node.js e React Native:

- Back-End e Front-End: Node.js é frequentemente usado para construir o back-end de aplicações, enquanto o React Native é utilizado para criar o front-end (aplicações móveis). A comunicação entre o back-end (Node.js) e o front-end (React Native) pode ser feita via APIs RESTful ou GraphQL.

Expo e React Native:

- Facilidade de Desenvolvimento: Expo facilita o desenvolvimento de aplicativos React Native ao fornecer uma configuração simplificada e ferramentas adicionais que não estão presentes na configuração padrão do React Native. Com Expo, os desenvolvedores podem se concentrar mais na criação do aplicativo e menos na configuração e no gerenciamento de dependências.

Conclusão

- Node.js é uma solução poderosa para construir servidores e back-ends escaláveis usando JavaScript.
- React Native permite criar aplicativos móveis nativos usando JavaScript, proporcionando uma experiência de usuário rica e fluida.
- Expo simplifica o processo de desenvolvimento com React Native, oferecendo ferramentas e recursos que facilitam a criação, teste e lançamento de aplicativos móveis.

Ambos os frameworks, quando usados juntos, oferecem uma stack poderosa para o desenvolvimento de aplicações completas, desde o back-end até o front-end em dispositivos móveis.

APIs RESTful são uma abordagem popular para construir interfaces de programação de aplicativos (APIs) que seguem os princípios do **REST** (Representational State Transfer). Elas permitem que diferentes sistemas se comuniquem e troquem dados de forma eficiente e padronizada pela web. Vamos detalhar o que são e como funcionam:

O que é uma API RESTful?

API (Application Programming Interface) é um conjunto de definições e protocolos que permitem que softwares diferentes se comuniquem entre si. Uma **API RESTful** é uma API que segue os princípios do **REST** e utiliza HTTP como seu protocolo de comunicação.

Princípios do REST

1. Stateless (Sem Estado):
- Cada solicitação do cliente para o servidor deve conter todas as informações necessárias para que o servidor entenda e processe a solicitação. O servidor não armazena informações sobre o estado da sessão do cliente entre as solicitações.

2. Client-Server (Cliente-Servidor):
- A arquitetura REST separa o cliente do servidor, o que significa que o cliente e o servidor podem evoluir independentemente. O cliente faz solicitações para o servidor, e o servidor responde com os dados ou as ações solicitadas.

3. Cacheable (Cacheável):
- As respostas das APIs RESTful podem ser armazenadas em cache para melhorar o desempenho. Isso significa que as respostas de uma solicitação podem ser armazenadas temporariamente para que as solicitações subsequentes possam ser atendidas mais rapidamente.

4. Uniform Interface (Interface Uniforme):
- REST define uma interface uniforme entre os sistemas, o que simplifica a arquitetura e promove a interoperabilidade. Isso é alcançado através de padrões e convenções para as solicitações e respostas HTTP.

5. Layered System (Sistema em Camadas):
- A arquitetura REST permite que a API seja composta por várias camadas de servidores, como proxies e gateways. Cada camada só interage com a camada adjacente e não tem conhecimento direto da arquitetura além dela.

6. Code on Demand (Código Sob Demanda) (Opcional):
- REST permite que o servidor forneça código executável (como scripts JavaScript) que pode ser usado pelo cliente. No entanto, essa abordagem não é amplamente utilizada.

Métodos HTTP Comuns em APIs RESTful

- **GET:** Usado para recuperar dados do servidor.
- **POST:** Usado para enviar dados ao servidor para criar um novo recurso.
- **PUT:** Usado para atualizar ou substituir um recurso existente no servidor.
- **DELETE:** Usado para excluir um recurso do servidor.
- **PATCH:** Usado para fazer atualizações parciais em um recurso.

Recursos e URLs

Em APIs RESTful, recursos (dados ou objetos) são identificados por URLs (Uniform Resource Locators). Cada URL representa um recurso específico e as operações sobre esse recurso são realizadas através dos métodos HTTP.

Exemplo de URLs e Métodos:
- **GET /users:** Recupera uma lista de usuários.
- **GET /users/{id}:** Recupera um usuário específico com base no ID.
- **POST /users:** Cria um novo usuário.
- **PUT /users/{id}:** Atualiza um usuário específico com base no ID.
- **DELETE /users/{id}:** Exclui um usuário específico com base no ID.

Formatos de Dados

APIs RESTful frequentemente utilizam **JSON (JavaScript Object Notation)** como o formato padrão para troca de dados devido à sua simplicidade e facilidade de uso. No entanto, outros formatos como **XML (eXtensible Markup Language)** também podem ser utilizados.

Benefícios das APIs RESTful

- Simplicidade: A interface uniforme e a utilização de métodos HTTP padrão tornam as APIs RESTful fáceis de entender e usar.
- Escalabilidade: A separação entre cliente e servidor e a capacidade de cache ajudam a melhorar o desempenho e a escalabilidade.
- Interoperabilidade: O padrão REST facilita a comunicação entre diferentes sistemas e plataformas.
- Flexibilidade: APIs RESTful podem ser usadas com diferentes tipos de clientes e podem ser evoluídas de forma independente do cliente.

Conclusão

APIs RESTful são uma abordagem amplamente utilizada para criar interfaces de programação de aplicativos que são simples, escaláveis e compatíveis com uma ampla variedade de sistemas. Seguindo os princípios do REST e utilizando métodos HTTP padrão, elas permitem uma comunicação eficiente e padronizada entre diferentes partes de um sistema.