Capítulo 1

Desenvolvimento de extensão para o Chrome com persistência em nuvem utilizando Firebase

Letícia D. N. Ximenes, Marcos Antônio B. P. Tenório de Oliveira, Francisco Kellviny Cruz Feitosa e Mayllon Veras da Silva

Abstract

In this chapter, we will cover the fundamental concepts of HTML, CSS, and JavaScript, followed by an introduction to cloud data persistence. The chosen tool for this purpose is Visual Studio Code, and we will utilize Cloud Firestore, a free service provided by Google. The goal is to create a simple extension for the Chrome browser, aiming to assist in the initial learning of web development.

Resumo

Neste capítulo, serão abordados os conceitos básicos de HTML, CSS e JavaScript, seguidos pela introdução à persistência de dados em nuvem. A ferramenta escolhida para este propósito é o Visual Studio Code, e utilizaremos o Cloud Firestore, um serviço gratuito fornecido pelo Google. O objetivo é criar uma extensão simples para o navegador Chrome, visando auxiliar no aprendizado inicial do desenvolvimento web.

1.1. Introdução

Neste capítulo, o principal objetivo é fornecer uma introdução acessível e prática às tecnologias fundamentais do desenvolvimento web. Explorando os conceitos básicos do HTML, CSS e JavaScript, que são as linguagens essenciais para construir páginas da web.

Além disso, o armazenamento de informações se dará por meio da nuvem utilizando o Cloud Firestore, um serviço gratuito oferecido pelo Google. Este que permite guardar e recuperar dados online, o que é de muita utilidade em projetos web.

Para uma melhor organização, o capítulo será dividido em três partes principais. Contendo cada uma delas os tópicos de HTML, CSS, JavaScript, o banco de dados em núvem Cloud Firestore e os passos necessários para o desenvolvimento de uma extensão para o navegador Chrome.

1.1.1. Construir uma base sólida

As primeiras páginas desse capítulo se dão à exploração dos alicerces fundamentais do **HTML, CSS e JavaScript**. Será fornecida uma base sólida para a construção de páginas web, estabelecendo a base de conhecimento para seu desenvolvimento. À medida que se adquire esses fundamentos, torna-se possível uma progressão de conhecimento web que é fundamental para o desenvolvimento de aplicações de maior complexidade.

Este conhecimento não é somente uma etapa inicial, como também uma preparação para a criação de uma extensão para o navegador Chrome. O início deste capítulo oferece a base essencial que servirá como um guia ao desenvolvimento de extensões e à expansão de habilidades na programação em geral.

1.1.2. Explorar o banco de dados em nuvem

Após estabelecida as bases sólidas no desenvolvimento web, é necessário um entendimento do armazenamento de dados na nuvem. Nesta parte, será conduzida uma análise do banco de dados em nuvem que servirá como diferencial de uma extensão para o navegador Chrome. A ferramenta escolhida foi o Cloud Firestore, uma solução oferecida pelo Google.

Aqui será detalhado como essa ferramenta permite armazenar e acessar dados de forma simples e intuitiva, tornando-a a escolha ideal para a introdução ao armazenamento de dados na nuvem. Começando com uma visão geral do Cloud Firestore, abordando seus principais conceitos e recursos. Em seguida, será fornecido o conhecimento de como criar e gerenciar bancos de dados, armazenar informações e acessá-las. Após a edificação base do assunto, será possível atrelá-lo ao desenvolvimento de uma extensão.

1.1.3. Entender o que é uma extensão e seu processo de desenvolvimente

Para concluir este capítulo, depois de abordados em detalhes os tópicos essenciais, será desenvolvido o entendimento das extensões. Desde o que exatamente é uma extensão, como funciona, o seu desenvolvimento específico para o navegador Chrome e as maneiras pelas quais o Cloud Firestore pode ser integrado a esse processo.

Além disso, será oferecido um exemplo de extensão que será construído, proporcionando uma visão concreta e tangível dos conceitos discutidos anteriormente. Este fim de capítulo servirá como um guia abrangente para a criação de extensões para o navegador Chrome, destacando algumas estratégias de integração com o Cloud Firestore.

1.2. HTML: Introdução e Conceito

Esta seção concentra-se nos princípios fundamentais do HTML (Hypertext Markup Language). O HTML é a principal linguagem usada para criar páginas da web e serve como a base essencial para qualquer desenvolvedor. Este conhecimento é fundamental, pois será aplicado posteriormente na criação de uma extensão simples para o navegador Chrome.

O HTML é uma linguagem de marcação que estrutura o conteúdo de uma página da web, fornecendo a estrutura básica para todas as páginas na internet. Com o HTML, você pode definir a estrutura de uma página, criar links, inserir imagens e muito mais. Além disso, todo documento HTML segue uma estrutura padrão, como demonstrado no

código abaixo.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
     initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

Listing 1.1. Estrutura básica HTML

1.2.1. HTML: <!DOCTYPE html>

O termo <DOCTYPE html> é uma instrução fundamental no HTML que especifica a versão e o tipo de documento que está sendo usado em uma página da web.

A tag <!DOCTYPE html> é uma forma de declarar que o documento é um documento HTML. Ela informa ao navegador que o conteúdo da página deve ser interpretado como um documento HTML.

O <!DOCTYPE html> não especifica uma versão específica do HTML, como o HTML5. Em vez disso, ele indica ao navegador que a página deve ser tratada de acordo com os padrões mais recentes e compatíveis com HTML.

A declaração <!DOCTYPE html> também desempenha um papel crucial na determinação do modo de renderização do navegador. Os navegadores têm modos diferentes de renderização para lidar com páginas mais antigas e com páginas compatíveis com os padrões modernos. O <!DOCTYPE html> instrui o navegador a usar o modo de renderização mais recente e compatível.

O uso do <!DOCTYPE html> é altamente recomendado em todos os documentos HTML. Ele ajuda a garantir que sua página seja renderizada corretamente nos navegadores modernos e seja compatível com os padrões atuais da web.

1.2.2. HTML: Tags de construção web

As tags HTML são as bases fundamentais da web. Elas são os elementos mais básicos de uma página web, pois tudo é desenvolvido a partir delas, incluindo os caracteres apresentados, informações de vídeo e imagem, ou campos de dados dos quais serão retiradas informações necessárias, que podem ou não ser usadas posteriormente.

Em resumo, elas definem a estrutura e o conteúdo de uma página. Elas são cercadas por colchetes angulares, como por exemplo, ’<tag>’, e geralmente vêm em pares, consistindo em uma tag de abertura e outra de fechamento com uma barra entre o primeiro colchete e a tag ’</tag>’, como a tag ’<h1>’, que tem a seguinte estrutura ’<h1>’

Título '</h1>'. Também existem aquelas que não necessitam de fechamento, como a tag '<input>'.

Tabela 1.1. Tags exemplo, que também serão utilizadas na extensão

Tags	Funcionalidade
<html>	**É o elemento raiz de uma página HTML. Ela engloba todo o conteúdo da página.**
<head>	**É usada para incluir informações sobre o documento que não são visíveis para os visitantes da página, como metadados, folhas de estilo CSS e meta tags.**
<body>	**Contém o conteúdo visível da página da web, incluindo texto, imagens, links e outros elementos que os visitantes veem e interagem.**
<button>	**Cria um botão clicável que pode ser usado para acionar ações ou eventos em uma página web.**
<iframe>	**Permite incorporar uma janela de navegador aninhada em uma página, útil para incorporar conteúdo externo, como mapas ou vídeos.**
<script>	**É usada para incluir código JavaScript diretamente em uma página, permitindo interatividade e funcionalidade dinâmica.**
<link>	**É usada para vincular folhas de estilo externas (CSS) a uma página, definindo a aparência e o layout do conteúdo.**
<input>	**Cria campos de entrada, como caixas de texto, botões de opção ou caixas de seleção, que permitem aos usuários inserir dados em um formulário.**
	É usada para criar listas não ordenadas, com itens listados com marcadores, como pontos ou símbolos.
	É usada dentro de elementos ou para definir itens de lista, como elementos de uma lista de itens.
<a>	**Cria links para outras páginas ou recursos, permitindo aos usuários navegar para diferentes partes do site ou para outros sites.**

1.2.2.1. HTML: Atributos, Classes e Ids

Atributos de tag em HTML são informações adicionais que podem ser incluídas dentro das tags HTML para fornecer detalhes ou instruções específicas sobre como um elemento deve ser processado ou exibido em um navegador da web.

Sintaxe: Os atributos de tag são especificados dentro das tags HTML e geralmente têm a seguinte sintaxe: nome-do-atributo = "valor". O nome do atributo é sempre colocado dentro do nome da tag, seguido por um sinal de igual (=) e, em seguida, o valor do atributo, que é colocado entre aspas duplas (") ou aspas simples (').

Validação e Acessibilidade: A escolha e o uso adequados de atributos de tag são importantes para garantir que seu código HTML seja válido e acessível. A atribuição de valores significativos a atributos, como alt em imagens e aria-label em elementos acessíveis, melhora a experiência do usuário.

Os atributos de tag em HTML desempenham um papel essencial na estruturação e no funcionamento das páginas da web, permitindo que os desenvolvedores controlem como os elementos são exibidos e interagem com os usuários. Também vale-se ressaltar que existem diversos tipos de atributos para HTML, porém alguns necessitam de uma tag específica para serem utilizados. Na seguinte tabela há um detalhamento desses atributos:

Além disso, é fundamental destacar que as Classes e IDs são atributos HTML

Tabela 1.2. Tipos de atributos HTML

Tipos de Atributos	Explicação
Atributos Globais	**Existem atributos que são considerados "globais"e podem ser usados em praticamente qualquer elemento HTML. Alguns exemplos de atributos globais incluem id, class, style, title, lang, data-*, entre outros.**
Atributos Específicos de Elementos	**Cada elemento HTML pode ter atributos próprios específicos que controlam seu comportamento e aparência. Por exemplo, a tag <img> tem o atributo src para especificar a fonte da imagem e o atributo alt para fornecer um texto alternativo.**
Atributos de Hiperlink	**As tags de hiperlink, como <a> e <area>, têm atributos especiais, como href para especificar o destino do link e target para determinar onde o link deve ser aberto (por exemplo, em uma nova janela ou na mesma janela).**
Atributos de Formulário	**Os formulários HTML, criados com as tags <form>, <input>, <select>, entre outras, utilizam atributos como name, type, value, placeholder e outros para coletar informações do usuário e enviar dados para o servidor.**
Atributos Booleanos	**Alguns atributos têm valores booleanos, o que significa que sua presença indica verdadeiro e sua ausência indica falso. Por exemplo, o atributo checked em uma caixa de seleção indica se ela está marcada.**

essenciais para a identificação e seleção de elementos específicos, principalmente com o propósito de estilização e manipulação por meio de CSS e/ou JavaScript.

O que são Classes: Classes são atributos que podem ser adicionados a elementos HTML para identificá-los de maneira específica. Diferentes elementos podem compartilhar a mesma classe, e um elemento pode ter várias classes.

Sintaxe: Para adicionar uma classe a um elemento HTML, utiliza-se o atributo "class"e atribui-se um nome de classe. Como por exemplo no seguinte código:

```
<p class="destaque">destaque.</p>
```

Listing 1.2. Sintaxe do atributo "class"

O que são IDs: IDs são atributos usados para identificar exclusivamente um único elemento em uma página HTML. Cada ID deve ser único na página.

Sintaxe: Para adicionar um ID a um elemento HTML, usa-se o atributo id e associa-se um nome de ID. Como por exemplo no código:

```
<div id="cabecalho">Div principal.</p>
```

Listing 1.3. Sintaxe do atributo "ID"

1.2.3. HTML: Inclusão de bibliotecas

As bibliotecas são conjuntos de código pré-escrito que fornecem funcionalidades específicas para facilitar o desenvolvimento de sites e aplicativos. Sua inclusão em HTML é uma prática importante na construção de sites mais complexos, principalmente quando se trata de desenvolvimento com banco de dados em nuvem, pois somente com essas bibliotecas se torna possível utilizar diversos dos serviços disponibilizados. Algumas informações sobre bibliotecas HTML são:

Reutilização de Código: As bibliotecas contêm código pré-escrito que aborda

diversas tarefas, como manipulação do DOM (Document Object Model), animações, validação de formulários, gráficos e, principalmente, a aplicação de um sistema de banco de dados em nuvem. Incluir bibliotecas permite que os desenvolvedores reutilizem esse código em seus projetos, economizando tempo e esforço.

Desenvolvimento Rápido: Com o uso de bibliotecas, é possível desenvolver sites e aplicativos mais rapidamente. Em vez de escrever código do zero para cada funcionalidade, os desenvolvedores podem aproveitar as bibliotecas existentes para acelerar o processo de desenvolvimento.

Confiabilidade e Testes: Bibliotecas amplamente utilizadas são frequentemente testadas pela comunidade de desenvolvedores, o que aumenta a confiabilidade do código. Isso reduz a probabilidade de erros e problemas de segurança.

1.2.3.1. HTML: Como Incluir Bibliotecas

Usando <script> para JavaScript: Para incluir uma biblioteca JavaScript em seu HTML, você pode usar a tag <script> no elemento <head> ou <body>, essas bibliotecas podem ser tanto na nuvem, quanto locais. Respectivamente como exemplo de inclusão em nuvem e local observa-se a Listing 1.4 e Listing 1.5:

```
<script src="https://www.gstatic.com/firebasejs/9.14.0/
   firebase-compat.js"><script>
```

Listing 1.4. Inclusão de biblioteca da nuvem.

```
<script src="sandbox.js"><script>
```

Listing 1.5. Inclusão de biblioteca da local

Ordem de Inclusão: A ordem em que você inclui bibliotecas no HTML pode ser importante, especialmente no caso de bibliotecas JavaScript. Normalmente, é uma boa prática incluir bibliotecas antes da utilização do código JavaScript para garantir que as dependências estejam disponíveis.

1.3. CSS: Introdução e Conceito

O tópico a ser retratado se dedica na explosão dos conceitos de CSS (Cascading Style Sheets). Representa uma linguagem de estilo (en-US) usada para a apresentação de um documento que está estruturado em HTML ou XML (abrangendo diversos tipos de XML, como SVG, MathML ou XHTML). O CSS determina a maneira como os elementos são exibidos em diferentes tipos de mídia, incluindo tela, papel, saída de áudio ou outros meios de apresentação.

CSS é considerada umas das principais liguagens da open web e incorporada nos navegadores da web de acordo com as espicificações da WC3 (World Web consortium). Possui diversas versões como CSS1, CSS2 e CSS3, a recomendada é a CSS2, porém a mais atual e que está em processo de padronização se trata da versão CSS3, que será discutida nessa parte do capítulo.

Um dos principais usos do CSS é se trata da estilização de elementos HTML, permitindo que os desenvolvedores determinem as cores, fontes, tamanhos de texto e espaçamentos. Isso garante que o conteúdo seja apresentado de forma atraente e legível, tornando a experiência do usuário mais agradável. Com a capacidade de adaptar a aparência de um site a diferentes tamanhos de tela, permitindo que os programadores construam interfaces que funcionem corretamente em dispositivos móveis, tablets e desktops, mantendo a consistência visual.

1.3.1. CSS: Estrutura do Codigo

A estrutura de um código CSS é fundamental para manter o projeto organizado e fácil de entender. Uma boa estrutura ajuda a evitar problemas de manutenção, facilitando a colaboração com outros desenvolvedores e melhora a eficiência do desenvolvimento. A estrutura de um código CSS segue uma série de elementos e regras que organizam as instruções de estilo para definir como os elementos HTML devem ser apresentados. A estrutura básica de um código CSS se baseia em:

Seletor: Um seletor é usado para identificar qual elemento HTML ou grupo de elementos será estilizado. Pode ser um elemento HTML específico, uma classe, um ID, ou até mesmo um seletor universal que se aplica a todos os elementos.

Exemplos de seletores:

Seletor de elemento HTML: 'p' estiliza todos os parágrafos.

Seletor de classe: '.destaque' estiliza todos os elementos com a classe "destaque".

Seletor de ID: '#cabecalho' estiliza o elemento com o ID cabecalho.

Propriedades: As propriedades são os atributos de estilo que você deseja aplicar ao(s) elemento(s) selecionado(s). Cada propriedade tem um nome que descreve a característica do estilo e um valor que define como essa característica deve ser aplicada, podem mudar desde a cor, a tamanho, margem, entre outras mudanças.

Exemplo de propriedades:

'color': Define a cor do texto.

'font-size': Define o tamanho da fonte.

'margin': Define margens externas.

'padding': Define espaçamento interno.

'background-color': Define a cor de fundo

Declaração: Uma declaração CSS é composta por um seletor e um conjunto de propriedades e valores. Ela define como um elemento selecionado deve ser estilizado.

Bloco de Regras: Um bloco de regras CSS é um conjunto de uma ou mais declarações CSS que são agrupadas entre chaves. Este bloco é associado a um seletor, que define a quais elementos HTML as declarações dentro do bloco se aplicam. Os blocos de regras permitem estilizar elementos HTML de acordo com as propriedades e valores definidos nas declarações CSS. Eles são a base da estilização em CSS, permitindo que você controle o layout e a aparência de sua página da web.

```
p {
    color: blue;
    font-size: 160px;
}
```

Listing 1.6. Exemplo de declaração e Bloco de regras

Comentários: Você pode incluir comentários no seu código CSS para documentação e esclarecimentos. Comentários começam com /* e terminam com */.

```
/*Exemnplo de um comentario CSS*/
```

Listing 1.7. Exemplo de comentário CSS

Arquivo externo: O código CSS também pode ser colocado em um arquivo externo com a extensão .css e, em seguida, referenciado no documento HTML usando a tag <link>.

```
<link rel="stylesheet" type="text/css" href="estilos.css">
```

Listing 1.8. Exemplo de referência de um arquivo CSS externo

Em resumo, a estrutura de um código CSS consiste em seletores que definem os elementos a serem estilizados, propriedades que descrevem o estilo e os valores que definem como esse estilo deve ser aplicado. As declarações são agrupadas em blocos de regras e podem ser organizadas em um arquivo externo para facilitar a manutenção e a reutilização dos estilos em um site. Comentários também podem ser usados para fornecer informações adicionais sobre o código CSS.

1.3.2. CSS: Demais funções

Além do que foi citado anteriormente o CSS ainda possui diversas outras funções que são do interesse de qualquer desenvolvedor web.

1.3.2.1. CSS: Especificidade

A especificidade CSS refere-se à maneira como os navegadores determinam qual regra de estilo deve ser aplicada a um elemento quando várias regras conflitam entre si. É um conceito fundamental em CSS e é usado para resolver conflitos e determinar a prioridade das regras de estilo em uma página da web. A especificidade é crucial para garantir que o design de uma página seja consistente e previsível.

- **Hierarquia de Seletores:** A hierarquia de seletores determina a ordem de prioridade das regras CSS. Quanto mais específico for o seletor, maior será a prioridade da regra. Por exemplo, uma regra aplicada a um seletor de classe (.classe) terá mais prioridade do que uma regra aplicada a um seletor de tipo (elemento) porque é mais específica.

- **ID vs. Classes vs. Elementos:** A especificidade também depende do tipo de seletor usado. Os seletores ID (#id) têm mais peso do que seletores de classe (.classe) e seletores de tipo (elemento). Portanto, uma regra com um seletor ID terá prioridade sobre regras com seletores de classe ou elementos.

- **Inline Styles:** As regras de estilo definidas diretamente em um elemento HTML usando o atributo style têm a maior prioridade e substituem todas as outras regras. Isso significa que os estilos inline prevalecerão sobre qualquer outra definição de estilo.

- **!important:** O valor !important pode ser anexado a uma declaração de estilo em uma regra CSS para dar a ela a mais alta prioridade. No entanto, o uso excessivo de !important não é uma prática recomendada, pois pode tornar o código CSS mais difícil de manter e depurar.

- **Especificidade de Combinação:** Quando há várias partes em um seletor, como combinação de seletores de classe, ID, elementos e pseudo-classes, a especificidade é calculada somando os valores correspondentes a cada parte. Por exemplo, um seletor como div#id .classe terá uma especificidade maior do que um seletor como .classe .classe.

- **Cascata:** A ordem em que as regras CSS são declaradas também é importante. As regras que aparecem posteriormente no código têm precedência sobre as regras anteriores, desde que tenham a mesma especificidade. Isso é conhecido como ordem de cascata.

1.3.2.2. CSS: Herança

A herança em CSS é um princípio fundamental que determina como as propriedades CSS são aplicadas a elementos HTML dentro de uma página da web. A herança permite que você defina estilos para elementos pai e permita que esses estilos sejam herdados por elementos filhos, economizando assim muito tempo e esforço no desenvolvimento de páginas web consistentes e bem projetadas.

- **Como Funciona:** A herança em CSS segue a hierarquia de elementos HTML. Isso significa que, quando você aplica um estilo a um elemento pai, os elementos filhos desse pai herdarão automaticamente esse estilo, a menos que um estilo específico seja aplicado diretamente a eles, substituindo a herança.

- **Propriedades Herdadas:** Nem todas as propriedades CSS são herdadas. As propriedades herdadas são aquelas que fazem sentido serem passadas dos elementos pai para os elementos filhos. Exemplos de propriedades herdadas incluem font-family, color, font-size, line-height e text-align.

- **Propriedades Não Herdadas:** Algumas propriedades CSS não são herdadas, o que significa que elas não são transmitidas aos elementos filhos. Exemplos de propriedades não herdadas incluem width, height, margin, padding e border.

1.4. JavaScript: Introdução e Conceito

Nesta seção será explorada os fundamentos do JavaScript. Esta que é uma linguagem de programação amplamente utilizada no desenvolvimento web para criar funcionalidades em páginas da web. Diferentemente do HTML, que é usado para estruturar o conteúdo de uma página, e do CSS, que é usado para estilizar a página, o JavaScript é usado para adicionar comportamento e interatividadde à web.

- **LIguaguem de script:** JavaScript é uma linguagem de script, o que significa que é executada diretamente no navegador da web do usuário. Isso permite que os desenvolvedores criem funcionalidades que respondem às ações dos usuários sem a necessidade de recarregar a página ou fazer solicitações ao servidor.

- **Interatividade:** Uma das principais funções do JavaScript é tornar as páginas da web interativas. Isso inclui validar formulários, criar animações, responder a cliques de botões etc..

- **Cliente-Side:** JavaScript é executado no lado do cliente, o que significa que o código JavaScript é baixado e executado no navegador do usuário. Isso o torna adequado para tarefas que não requerem acesso a recursos do servidor.

- **Manipulação do DOM:** O DOM (Document Object Model) é uma representação hierárquica da estrutura de uma página da web. JavaScript é usado para acessar e manipular o DOM, permitindo que os desenvolvedores alterem o conteúdo e a aparência de uma página.

- **Ampla Comunidade e Ecossistema:** JavaScript possui uma grande comunidade de desenvolvedores e um ecossistema rico de bibliotecas e frameworks que facilitam o desenvolvimento de aplicativos web complexos. Alguns dos frameworks populares incluem React, Angular e Vue.js.

- **Versatilidade:** Além de ser usado no desenvolvimento web, JavaScript também é usado em outras áreas, como desenvolvimento de jogos (usando bibliotecas como Phaser), desenvolvimento de aplicativos móveis híbridos (usando frameworks como o Apache Cordova) e até mesmo em extenções para o navegador chrome.

1.4.1. JavaScript: Linguagem de Script

Uma linguagem de script é uma linguagem de programação que é projetada para ser executada por um interpretador ou máquina virtual, em oposição a linguagens compiladas, que são traduzidas para código de máquina antes de serem executadas.

- **Interpretação:** As linguagens de script são geralmente interpretadas, o que significa que o código é lido linha por linha e executado diretamente por um interpretador, sem a necessidade de um processo de compilação separado. Isso torna a escrita e a execução de código mais flexíveis e rápidas.

- **Facilidade de Uso:** As linguagens de script são frequentemente projetadas para serem de fácil leitura e escrita, o que as torna acessíveis a programadores de diferentes níveis de habilidade. Elas geralmente têm uma sintaxe mais simples e abstrações de alto nível que simplificam tarefas comuns.
- **Tempo de Execução:**Como as linguagens de script são interpretadas em tempo de execução, elas permitem uma execução dinâmica do código, o que significa que as decisões podem ser tomadas com base em condições que só são conhecidas durante a execução.
- **Tempo de Execução:** Como as linguagens de script são interpretadas em tempo de execução, elas permitem uma execução dinâmica do código, o que significa que as decisões podem ser tomadas com base em condições que só são conhecidas durante a execução.
- **Linguagens de Script na Web:** Linguagens de script desempenham um papel crucial na criação de interatividade em páginas da web. JavaScript é a linguagem de script mais comumente usada na web para validar formulários, criar animações, manipular o DOM e muito mais.

1.4.2. JavaScript: Interativisdade

A interatividade em JavaScript refere-se à capacidade de criar interações em seu código. Isso é fundamental para realizar tarefas como processar listas de dados, criar animações, validar entradas do usuário, etc.

Variáveis: Em JavaScript são utilizadas para armazenar dados e valores que podem ser usados ao longo do programa. Elas são um conceito fundamental em programação e desempenham um papel crucial na manipulação de informações. Por se tratar de uma linguagem dinâmica, não é necessário declarar explicitamente o tipo de uma variável. já que o mesmo é determinado em tempo de execução com base no seu valor atribuído.

Existem três palavras-chave para declarar variáveis em JavaScript. Essas que são:

Tabela 1.3. Tipos de variáveis JavaScript

Variáveis	Explicação
var	**Foi a forma tradicional de declarar variáveis em JavaScript, mas possui escopo de função, o que pode levar a problemas de escopo. É raramente usado hoje em dia.**
let	**Introduzido no ES6 (ECMAScript 2015), o let permite declarar variáveis com escopo de bloco, o que significa que elas são visíveis apenas dentro do bloco onde foram declaradas.**
const	**Também introduzido no ES6, o const é usado para criar variáveis de somente leitura, ou seja, uma vez atribuído um valor, ele não pode ser alterado.**

```
let saldo = 1000;
const nome = "Paulo";
var idade = 25;
```

Listing 1.9. Exemplo das variáveis JavaScript

Há também algumas regras para a nomeclatura de variáves, como por exemplo, deve-se começar com uma letra, sublinhado (_) ou cifrão ($), pode-se conter letras, números, sublinhados e cifrões, sensíveis a maiúsculas e minúsculas (ex.: minhaVariavel e minhavariavel são diferentes).

Funções:A criação de funções em JavaScript é um conceito estrutural que permite o agrupamento de um conjunto de instruções em um bloco nomeado e reutilizável. Funções em JavaScript são usadas para organizar e modular o código, tornando-o mais legível e fácil de manter

A sintaxe básica para criar uma função em JavaScript é a seguinte. 'nomeDaFuncao:' Um identificador que representa o nome da função, 'parametro1', 'parametro2', etc.: Parâmetros opcionais que a função pode receber. Eles são usados para passar informações para a função, 'Código da função': O conjunto de instruções que a função executa quando é chamada.

```
function nomeDaFuncao(parametro1, parametro2, ...) {
  // C digo da function
}
```

Listing 1.10. Declaração de uma função em JavaScript

Para executar o código dentro de uma função, sendo assim necessário chamá-la usando o nome e, opcionalmente, passar argumentos para os parâmetros. Por exemplo:

```
function saudacao(nome) {
  console.log("Ol , " + nome + "!");
}
saudacao("Alice");
```

Listing 1.11. Chamando uma função em JavaScript

As funções podem retornar valores usando a palavra-chave return. Isso permite que seja obtido um resultado da função e o utilize em outras partes do código.

```
function soma(a, b) {
  return a + b;
}
let resultado = soma(5, 3);
```

Listing 1.12. Retorno de uma função em JavaScript

As variáveis declaradas dentro de uma função têm escopo local, o que significa que somente são visíveis e acessíveis dentro da função. Isso ajuda a evitar colisões de nomes de variáveis em diferentes partes do código.

If e Else: Em JavaScript, 'if' e 'else' são estruturas de controle de fluxo que permitem tomar decisões com base em condições. Elas são fundamentais para criar lógica condicional em programas.

A estrutura 'if' permite que seja executado um bloco de código se uma determinada condição for verdadeira. Se a condição não for verdadeira, o bloco de código não

será executado. A sintaxe geral é a seguinte:

```
let idade = 18;
if (idade >= 18) {
  console.log("alguma_coisa");
}
```

Listing 1.13. Exemplo do uso do if

A estrutura if...else permite que seja efetuado um bloco de código se a condição for verdadeira (if) e outro bloco de código se a condição for falsa (else). A sintaxe é a seguinte:

```
let idade = 15;
if (idade >= 18) {
  console.log("alguma_coisa");
} else {
  console.log("alguma_outra_coisa");
}
```

Listing 1.14. Exemplo do uso do if...else

Em resumo, as estruturas if e else são cruciais para criar lógica condicional em programas JavaScript, permitindo uma tomada de decisões com base nas condições especificadas. São amplamente utilizadas em scripts para tornar o código mais flexível e responsivo.

Manipulação do DOM: O gerenciamento do Document Object Model ou DOM, com Java=Script é um aspecto fundamental do desenvolvimento web. O DOM é uma representação em hierarquia da estrutura de uma página web, sua manipulação permite que os desenvolvedores acessem e modifiquem os elementos HTML e sua estrutura dinamicamente.

Para acessar elementos HTML no DOM, pode-se utilizar os métodos 'getElementById', 'getElementsByClassName', 'getElementsByTagName', 'querySelector', 'querySelectorAll' e outros. Aqui estão alguns exemplos:

- **getElementById:** Acessa um elemento pelo seu ID.
- **getElementsByClassName:** Acessa elementos por sua classe.
- **getElementsByTagName:** Acessa elementos por suas tags.
- **querySelector:** Acessa o primeiro elemento de um seletor CSS.
- **querySelectorAll** Acessa todos os elementos de um seletor CSS.

A manipulação do DOM com JavaScript é uma habilidade essencial para criar páginas da web interativas e dinâmicas. Com os comandos corretos, pode-se criar, modificar e remover elementos, tornando aplicações web mais flexíveis e responsivas. É importante

lembrar de usar essa manipulação com cuidado para evitar problemas de desempenho e acessibilidade.

addEventListener: É um método fundamental em JavaScript que permite adicionar ouvintes de eventos a elementos HTML. Eventos são ações que ocorrem em uma página web, como cliques do mouse, pressionamentos de teclas, carregamento da página, entre outros. O 'addEventListener' permite que sejam registradas funções (ouvintes) que serão executadas quando um evento específico ocorrer em um elemento HTML.

```
elemento.addEventListener(evento, function, opcional);
```

Listing 1.15. Sintaxe do addEventListener

elemento: É o elemento HTML ao qual deseja-se adicionar o ouvinte de evento.

evento: É uma string que especifica o tipo de evento que será ouvido (por exemplo, "click", "keydown", "load", "submit", etc.).

função: É a função que será executada quando o evento ocorrer.

opcional: Pode ser um objeto de opções que permite personalizar o comportamento do ouvinte, como usar a captura de eventos ou uma vez.

```
// Seleciona um elemento pelo ID
let meuBotao = document.getElementById("meu-botao");
// Adiciona um ouvinte de evento de clique ao bot o
meuBotao.addEventListener("click", function() {
  alert("clicado!");
});
```

Listing 1.16. Exemplo de Uso Básico

1.5. Banco de dados em nuvem: Cloud Firestore

Nesse segmento do capítulo, será introduzido o conceito fundamental de bancos de dados em nuvem, bem como informações essenciais sobre o Cloud Firestore, incluindo como integrá-lo ao seu projeto. Para incorporar o Cloud Firestore em aplicativos da web JavaScript, é necessário utilizar o Firebase, que é a plataforma que hospeda e fornece acesso ao Firestore. Você pode incluir o Firebase em seu projeto da web com um CDN (Content Delivery Network) ou instalá-lo via npm (Node Package Manager) para ter acesso às bibliotecas necessárias.

Uma vez integrado, você poderá utilizar as APIs e métodos fornecidos pelo Firebase para interagir com o Firestore no JavaScript da web. Essas funcionalidades permitem que você leia, escreva, atualize e exclua dados no Firestore, além de permitir a escuta de eventos de sincronização em tempo real. Essas capacidades tornam o Cloud Firestore uma ferramenta para o desenvolvimento de aplicativos web responsivos e dinâmicos na nuvem.

1.5.1. Introdução a banco de dados

Um banco de dados é um sistema organizado para armazenar, gerenciar e recuperar informações de maneira eficiente, que desempenha um papel crucial na maioria das aplicações de software, pois permite que os desenvolvedores armazenem e acessem dados de forma estruturada e confiável.

Bancos de dados são usados para armazenar uma variedade de informações, desde simples listas de tarefas até informações complexas em sistemas de gerenciamento de clientes, registros de vendas, registros médicos e muito mais. Desempenham um papel fundamental na persistência de dados, o que significa que os dados podem ser mantidos mesmo após a aplicação ser encerrada.

1.5.2. O que é o Cloud Firestore

O Cloud Firestore é um banco de dados NoSQL (Not Only SQL), ou seja, faz parte de uma categoria de bancos de dados que não seguem o modelo de bancos de dados relacionais tradicionais (SQL), que são baseados em tabelas com esquemas rigorosos. Em vez disso, os bancos de dados NoSQL adotam modelos de dados mais flexíveis e escaláveis, projetados para atender a diferentes necessidades de armazenamento e recuperação de informações.

Oferecido pela Google como parte da plataforma Google Cloud. Diferentemente dos bancos de dados tradicionais, o Firestore adota um modelo de dados flexível, onde os dados são organizados em documentos e coleções. Se destaca por sua escalabilidade, capacidade de sincronização em tempo real e integração com a infraestrutura de nuvem da Google, tornando-o uma escolha preferêcial para desenvolvedores que buscam construir aplicativos modernos e responsivos na nuvem. Com o Firestore, é possível armazenar, consultar e sincronizar dados de maneira eficaz, tornando-o uma ferramenta para uma variedade de aplicações em todo o mundo.

- **Modelo de Dados Flexível:** Os dados são armazenados em documentos que podem conter campos de diferentes tipos e serem organizados em coleções. Isso oferece flexibilidade significativa na estruturação de dados, permitindo que os desenvolvedores adaptem o banco de dados às necessidades de seus aplicativos.

- **Escalabilidade e Desempenho:** O Firestore é altamente escalável e pode lidar com grandes volumes de dados e tráfego de aplicativos, escalando automaticamente para atender às demandas crescentes.

- **Sincronização em Tempo Real:** Uma das características mais marcantes é a capacidade de sincronizar dados em tempo real entre os clientes e o banco de dados. Qualquer alteração nos dados é refletida imediatamente em todos os clientes conectados, tornando-o adequado para aplicativos que exigem colaboração em tempo real, como aplicativos de chat.

- **Consultas e Indexação:** O Firestore suporta consultas ricas que permitem buscar e filtrar dados com base em vários critérios, tornando-o adequado para aplicativos

que precisam de consultas complexas.O Firestore suporta consultas ricas que permitem buscar e filtrar dados com base em vários critérios, tornando-o adequado para aplicativos que precisam de consultas complexas.

- **Integração com Plataforma Google:** Se integra perfeitamente com outras ferramentas e serviços do Google Cloud, oferecendo uma solução completa para desenvolvimento de aplicativos na nuvem.

1.5.3. Cloud Firestore: Organização em documentos

A estrutura de organização de dados do Cloud Firestore é baseada em documentos, coleções e campos, e é projetada para ser altamente escalável e adaptável às necessidades dos desenvolvedores. Aqui está uma explicação mais detalhada:

- **Documentos:** O documento é a unidade básica de armazenamento de dados no Cloud Firestore. Cada documento é um registro individual que contém um conjunto de campos e seus valores associados. Os documentos são armazenados em coleções e podem ser pensados como equivalentes a registros ou linhas em uma tabela de banco de dados relacional.

- **Coleções:** As coleções são grupos de documentos relacionados. Podem ser criadas e nomeadas de acordo com as necessidades do aplicativo. Uma coleção pode conter vários documentos, e os documentos dentro de uma coleção não precisam ter a mesma estrutura de campos. Isso oferece uma grande flexibilidade na organização dos dados.

- **Campos:** Os campos são pares chave-valor que representam os dados em um documento. Cada campo tem um nome único (chave) e um valor associado. Os valores podem ser de diferentes tipos, como strings, números, datas, arrays ou mesmo outros documentos aninhados. O Cloud Firestore não impõe um esquema rígido, permitindo que diferentes documentos dentro da mesma coleção tenham campos diferentes.

- **Hierarquia:** Os documentos e coleções são organizados em uma hierarquia. Onde épossível aninhar coleções dentro de documentos ou documentos dentro de documentos, criando assim uma estrutura de dados hierárquica. Isso é especialmente útil para representar relacionamentos complexos entre os dados, como um usuário com várias postagens, onde cada usuário é um documento e as postagens podem ser coleções aninhadas dentro desse documento.

1.5.4. Cloud Firestore: Como incluir na web e códigos em JavaScript

Para incluir a biblioteca do Firebase (Cloud Firestore é um serviço do Firebase) em um projeto da web, é necessário seguir alguns passos. A biblioteca do Firebase é usada para interagir com o Cloud Firestore e outros serviços Firebase em aplicativos web. A versão utilizada será a do Firebase (versão 9.14.0).

- Primeiro, é necessário acessar o **Console Firebase**, após deve-se criar um novo projeto ou selecionar um projeto existente onde é desejado utilizar o Cloud Firestore.

- Dentro do projeto Firebase, acesse as configurações do projeto na guia "Geral". Na sequência, role a página até a seção "Seus aplicativos"e proceda com a adição de um novo aplicativo web para realizar a configuração adequada.
- Opte por um nome para o aplicativo da web. O Firebase gerará um objeto de configuração que conterá as informações essenciais para estabelecer a conexão com o projeto. Este objeto seguirá um formato semelhante ao exemplo a seguir:

```
    const firebaseConfig = {
      apiKey: "SUA_API_KEY",
      authDomain: "SEU_DOMNIO.firebaseapp.com",
      projectId: "SEU_ID_DO_PROJETO",
      storageBucket: "SEU_BUCKET.appspot.com",
      messagingSenderId: "SEU_SENDER_ID",
     appId: "SEU_APP_ID"
    };
```

Listing 1.17. Objeto exemplo do firebase

- Adicione a biblioteca do Firebase à sua página HTML, incluindo o seguinte código em seu arquivo HTML, normalmente dentro da tag <head>:

```
    <script src="https://www.gstatic.com/firebasejs
       /9.14.0/firebase-compat.js"></script>
```

Listing 1.18. Inclusão da biblioteca do firebase(9.14.0)

- Após incluir a biblioteca, você precisará inicializar o Firebase com as configurações do seu projeto. Coloque o seguinte código JavaScript em um arquivo ou script dentro do seu projeto:

```
import firebase from 'firebase/compat/app';\

import 'firebase/compat/firestore';

const firebaseConfig = {
  apiKey: "SUA_API_KEY",
  authDomain: "SEU_DOMNIO.firebaseapp.com",
  projectId: "SEU_ID_DO_PROJETO",
  storageBucket: "SEU_BUCKET.appspot.com",
  messagingSenderId: "SEU_SENDER_ID",
  appId: "SEU_APP_ID"
};

firebase.initializeApp(firebaseConfig);

const firestore = firebase.firestore();
```

Listing 1.19. Exemplo de inicialização do firebase

- Para adicionar um novo documento ao Cloud Firestore, utiliza-se o método 'add()' ou 'set()'. O 'add()' cria automaticamente um ID único para o documento, enquanto o 'set()' permite que você defina um ID personalizado.

```
const minhaColecao = collection(db,
'nome-da-sua-colecao');

const novoDocumento = { campo1: 'valor1',
campo2: 'valor2' };
const novoDocumentoRef = await addDoc(minhaColecao,
novoDocumento);

const documentoPersonalizadoRef = await
setDoc(minhaColecao, 'meu-id-personalizado',
novoDocumento);
```

Listing 1.20. Exemplo de como adicionar um documento a nuvem

- Para recuperar documentos do Cloud Firestore, pode-se usar o método get() para obter os dados de um documento específico ou onSnapshot() para ouvir atualizações em tempo real nos documentos.

```
const meuDocumentoRef = doc(db, 'nome-da-sua-colecao',
'id-do-documento');

const documento = await getDoc(meuDocumentoRef);
if (documento.exists()) {
  console.log('Dados do documento:', documento.data());
} else {
  console.log('O documento nao existe.');
}

const unsubscribe = onSnapshot(meuDocumentoRef,
(doc) => {console.log('Dados atualizados do documento:',
doc.data());
});

unsubscribe();
```

Listing 1.21. Exemplo de como vizualizar um documento na nuvem

- Para deletar documentos, utiliza-se o método delete().

```
const meuDocumentoRef = doc(db, 'nome-da-sua-colecao',
'id-do-documento');

await deleteDoc(meuDocumentoRef);
```

Listing 1.22. Exemplo de como deletar um documento da nuvem

1.6. Extensão Chrome: Conceito e Desenvolvimento

Nesta seção final do capítulo,com base no conhecimento adquirido sobre HTML, CSS, JavaScript e Cloud Firestore, será detalhado o conceito de uma extensão e como funciona seu processo de desenvolvimento para que a mesma consiga trabalhar ao lado do banco de dados do firebase. A forma utilizada para a implementação do Firebase em uma extensão será baseada no método do autor Aniket Das no site [1].

- **Conceito de uma extensão Chrome:** Uma extensão do Chrome é um pequeno programa de software que pode ser instalado no navegador Google Chrome. Essas extensões são projetadas para adicionar funcionalidades extras ao navegador, personalizar a experiência do usuário e melhorar a produtividade.

  As extensões do Chrome podem realizar uma variedade de tarefas, como bloquear anúncios, traduzir páginas da web, gerenciar senhas, verificar a ortografia, capturar capturas de tela, adicionar marcadores e muito mais. São criadas por desenvolvedores e estão disponíveis na Chrome Web Store, onde os usuários podem procurar, encontrar e instalar as extensões que desejam.

  Essas extensões geralmente aparecem na barra de ferramentas do navegador como ícones e podem ser ativadas ou desativadas de acordo com as necessidades do usuário. Ajudam a personalizar e melhorar a experiência de navegação no Chrome de acordo com as preferências individuais.

- **Definir o Objetivo da Extenção:** Antes de começar a escrever código, é importante ter uma compreensão clara do que a extensão deve fazer. deve-se ter um objetivo definido para a extensão, seja fornecer funcionalidades adicionais a sites, melhorar a experiência de navegação ou realizar outras tarefas específicas.

- **Cofigurar o Ambiente de Desenvolvimento:** Certificar-se de ter o ambiente de desenvolvimento adequado configurado. Isso geralmente inclui um editor de código, como o VS Code (Visual Studio Code), e o navegador Chrome instalado.

- **Criar a estrutura da extensão:** Uma extensão pode possuir diversos arquivos formadores da mesma dependendo de sua complexidade, porém devem ter no mínimo 4 arquivos principais: manifest.json‘: Este é o arquivo de manifesto da extensão, que contém informações sobre a extensão, como nome, descrição, versão e permissões, script.js : Este é um arquivo de código da extensão, onde está contido o JavaScript do programa, style.css : É um arquivo de folha de estilo que define a aparência visual da extensão, incluindo formatação, cores, fontes e layout, popup.html : Define o conteúdo de uma pequena janela pop-up que pode ser aberta pelo usuário clicando no ícone da extensão na barra de ferramentas do navegador. Esse arquivo HTML contém a interface do usuário que aparece quando a extensão é ativada e fornece uma maneira de interagir com a extensão ou acessar suas funcionalidades principais de forma rápida e conveniente.

- **Definir as permissões necessárias:** No arquivo ‘manifest.json‘, especifique as permissões necessárias para sua extensão funcionar. Por exemplo, se a extensão precisa acessar a página atual, precisará especificar a permissão ‘"activeTab"‘.

- **Criar a interface do usuário:** Usando HTML, CSS e JavaScript para criar a interface da extensão. Isso pode incluir botões, pop-ups, barras de ferramentas ou qualquer outra interface necessária para a interação com o usuário.
- **Escrever o código JavaScript:** Utiliza-se JavaScript para implementar a lógica da sua extensão. pode interagir com as APIs do Chrome para acessar funcionalidades específicas do navegador, como guias, histórico de navegação, cookies, entre outros.
- **Testar a extensão:** Carregar a extensão no Chrome para testá-la. Para fazer isso, é necessário acessar a página de extensões (chrome://extensions/), habilita o modo de desenvolvedor e carreguar a pasta da extensão.

1.6.1. Extensão Chrome: Desenvolvimento junto ao Cloud Firestore

Primeiro é necessário entender que para se utilizar o Firebase dentro de uma extensão é preciso uma série de permisões que só são adquiridas ao comprar o pacote de desenvolvedor exclusivo do Chrome, porém existe outra maneira de criar essa extensão, que se trata basicamente de utilizar um script como background do código, ou seja, o uso de um arquivo HTML e JavaScript extra permite que a extensão execute código em um ambiente controlado e isolado, útil para tarefas específicas que não devem afetar a página da web principal ou para garantir a segurança das operações da extensão.

Criando os arquivos da extensão: Ao abrir o VS Code (Visual Studio Code) se torna possível escolher uma pasta já existente no computador e utiliza-la para criação de novos arquivos, ao clicar em ´New File´, é importante saber que as terminações nos nomes dos arquivos indicam seu tipo, sendo respectivamente ´.js´,´.css´,´.html´,´.json´ para arquivos JavaScript, CSS, HTML e manifest. Para uma extensão básica com Cloud Firestore, serão 7 arquivos essenciais: ccs1.css, css2.css, manifest.json, popup.html, popup.js, sandbox.html e sandbox.js. Cada um será explicado de forma mais detalhada a seguir:

1.6.1.1. Extensão Chrome: manifest.json

O manifest.json é um arquivo de manifesto de extensão do Chrome, usado para configurar e definir as propriedades e comportamentos de uma extensão.

```
{
  "manifest_version": 3,
  "name": "My extension",
  "version": "1.0",
  "permissions": [
    "activeTab",
    "tabs"
  ],
  "action": {
    "default_popup": "popup.html"
  },
  "sandbox": {
```

```
    "pages": ["sandbox.html"]
  },
  "content_security_policy": {
    "sandbox": "sandbox allow-scripts allow-popups; script-
       src 'self' https://www.gstatic.com/ https://*.
       firebaseio.com https://www.googleapis.com"
  }
}
```

Listing 1.23. Arquivo manifest.json de uma extensão com Cloud Firestore

- **manifest_version (Versão do Manifesto):** Indica qual versão o manifesto segue. Cada versão tem suas próprias especificações e recursos, porém somente o manifest V3 é aceito pelo Chrome atualmente.

- **name (Nome):** Define o nome da extensão, que será exibido aos usuários na lista de extensões do Chrome e em outras interações com a extensão.

- **version (Versão):** Especifica a versão atual da extensão. Isso é útil para controlar e rastrear diferentes versões da extensão à medida que ela é atualizada.

- **permissions (Permissões):** Lista as permissões necessárias para a extensão. Neste caso, a extensão requer acesso às guias ativas ("activeTab") e acesso às guias em geral ("tabs"). Isso permite que a extensão interaja com as guias do navegador.

- **action:**Define as ações da extensão. Aqui, a ação padrão é configurada para abrir a página ´popup.html´ quando o ícone da extensão na barra de ferramentas do Chrome é clicado.

- **sandbox:** Especifica que a extensão utiliza uma página ´sandbox.html´ que é executada em um ambiente isolado. Isso ajuda a garantir que o código dentro dessa página não interfira na página da web atual e funcione de forma segura e isolada.

- **content_security_policy:** Define as políticas de segurança de conteúdo para a extensão. Aqui, é configurado para permitir a execução de scripts no ambiente sandbox (´"sandbox allow-scripts allow-popups"´) e também permite a execução de scripts a partir de fontes específicas, como o próprio domínio da extensão, ´https://www.gstatic.com´, ´https://*.firebaseio.com´, e ´https://www.googleapis.com´.

Em resumo, este arquivo manifest.json descreve os detalhes essenciais da extensão, incluindo seu nome, versão, permissões necessárias, comportamento padrão e configurações de segurança para a execução de scripts. Ele é fundamental para configurar e distribuir a extensão no navegador Chrome.

1.6.1.2. Extensão Chrome: popup.html e sandbox.html

O seguinte código HTML cria uma página da web simples com a inclusão de bibliotecas JavaScript (Firebase), um arquivo de script personalizado (sandbox.js), um arquivo de folha de estilo (css1.css), elementos de entrada de texto e botões, e uma lista não ordenada. A interação e funcionalidade específica da página dependerá do conteúdo do arquivo sandbox.js, que provavelmente contém lógica de JavaScript para manipular os elementos da página e realizar ações quando os botões são clicados.

```
<html>
<head>
  <script src="https://www.gstatic.com/firebasejs/9.14.0/
     firebase-compat.js"></script>

  <script src="sandbox.js"></script>

  <link rel="stylesheet" href="css1.css">
</head>
<body>
  <input id="addList" type="text"><br>
  <button id="addDocument">Adicionar</button>
  <button id="removeAll">Remover</button>
  <ul id="list"></ul>
</body>
</html>
```

Listing 1.24. Estrutura do sandbox.html

Este próximo código HTML cria uma página da web simples com um botão, um iframe para incorporar outra página chamada (sandbox.html), um arquivo JavaScript externo chamado "popup.js"e um arquivo de folha de estilo externo chamado (css2.css). É uma estrutura básica para criar uma página que pode ser interativa e estilizada com a ajuda de JavaScript e CSS.

```
<html>
        <body>
                <button id="init">Initialize Firebase</
                   button><br/>
                <iframe id="iframe" src="sandbox.html"></
                   iframe>
                <script src="popup.js"></script>
                <link rel="stylesheet" href="css2.css">
        </body>
</html>
```

Listing 1.25. Estrutura do popup.html

1.6.1.3. Extensão Chrome: css1.css e css2.css

Os arquivos CSS são puramente estilizações e, como tal, não contêm lógica de programação ou funcionalidades específicas. Eles são a ferramenta que permite aos desenvolvedores moldar a estética de uma página, desde o posicionamento de elementos até a escolha das cores, tipografia e animações. A flexibilidade inerente ao CSS significa que os desenvolvedores podem traduzir suas visões criativas em designs únicos, cativantes e, acima de tudo, funcionais, ou seja, dependem somente da preferência do desenvolvedor que cria a aplicação.

1.6.1.4. Extensão Chrome: popup.js e sandbox.js

Agora há dois arquivos JavaScript, cada um pro html de mesmo nome. Enquanto o popup.js somente inicializa o Firebase para que seja possível utiliza-lo na extensão, o sandbox.js faz todo o resto, tendo a função de adicionar, remover e mostrar todos os dados que estão presentes na nuvem.

```
const iframe = document.getElementById("iframe");

document.getElementById("init").onclick = () => {
        iframe.contentWindow.postMessage("init", "*");
};
```

Listing 1.26. Javascript do popup.js

Vale-se ressaltar também, que devemos incluir a configuração do Cloud Firestore no inicio do código dentro do sandbox.js e que após isso pode-se fazer uma extensão que inclua todas as funcionalidas do banco de dados em nuvem do Firebase.

```
const config = {
        projectId: "SEU_ID_DO_PROJETO",
        apiKey: "SUA_API_KEY",
        storageBucket: "SEU_BUCKET.appspot.com",
};

const app = firebase.initializeApp(config);
const db = firebase.firestore();

let isFirebaseInitialized = false;
```

Listing 1.27. Estrutura JavaScript básica do sandbox.js

1.7. Conclusão

Neste capítulo, foram explorados os fundamentos essenciais de desenvolvimento web, abordando os conceitos básicos de HTML, CSS e JavaScript. Além disso, foi introduzido o conceito de persistência de dados em nuvem, utilizando o Visual Studio Code como a

ferramenta de desenvolvimento e o Cloud Firestore, um serviço gratuito fornecido pelo Google, como a plataforma de armazenamento de dados em nuvem.

O principal objetivo deste capítulo é fornecer uma base sólida para aqueles que estão iniciando no desenvolvimento web, com foco na criação de uma extensão simples para o navegador Chrome que tenha como diferencial a utilização de um banco de dados em nuvem. Essa extensão tem a finalidade de auxiliar os iniciantes a compreenderem os princípios fundamentais dessa área da tecnologia.

Referências

[1] [10] Das, A. (2022). How to add firebase to a chrome extension on manifest v3.

[2] Documentação do Firebase sobre ler dados do Cloud Firestore, https://firebase.google.com/docs/firestore/query-data/get-data?hl=pt-br, Acessado em 20 de setembro de 2023.

[3] Mozilla Developer Network (MDN) para HTML, https://developer.mozilla.org/en-US/docs/Web/HTML, Acessado em 20 de setembro de 2023.

[4] Mozilla Developer Network (MDN) para JavaScript, https://developer.mozilla.org/en-US/docs/Web/JavaScript, Acessado em 20 de setembro de 2023.

[5] Mozilla Developer Network (MDN) para CSS, https://developer.mozilla.org/en-US/docs/Web/CSS, Acessado em 20 de setembro de 2023.

[6] Mozilla Developer Network (MDN) para HTML, https://developer.mozilla.org/en-US/docs/Web/HTML, Acessado em 20 de setembro de 2023.

[7] Documentação do Firebase sobre adicionar o Firebase ao projeto JavaScript, https://firebase.google.com/docs/web/setup?hl=pt-br, Acessado em 20 de setembro de 2023.

[8] Documentação do Firebase de introdução ao Cloud Firestore, https://firebase.google.com/docs/firestore?hl=pt-br, Acessado em 20 de setembro de 2023.

[9] Documentação do Firebase sobre adicionar dados ao Cloud Firestore, https://firebase.google.com/docs/firestore/manage-data/add-data?hl=pt-br, Acessado em 20 de setembro de 2023.

[10] Documentação do Firebase sobre excluir dados ao Cloud Firestore, https://firebase.google.com/docs/firestore/manage-data/delete-data?hl=pt-br, Acessado em 20 de setembro de 2023.