Embora um PWA de verdade precise de JavaScript para funcionalidades como o **Service Worker** (que permite o uso offline), você pode construir a base visual e a estrutura do seu projeto usando apenas HTML e CSS. Depois, é fácil adicionar o JavaScript.

Vamos começar com um projeto simples: um aplicativo de lista de tarefas.

### **1. Estrutura do HTML**

Crie um arquivo chamado index.html. Este será o esqueleto do nosso aplicativo.

HTML

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Minha Lista de Tarefas PWA</title>

<link rel="stylesheet" href="style.css">

<link rel="manifest" href="manifest.json">

<link rel="apple-touch-icon" href="icons/apple-icon-180.png">

<link rel="icon" type="image/png" sizes="32x32" href="icons/favicon-32x32.png">

<link rel="icon" type="image/png" sizes="16x16" href="icons/favicon-16x16.png">

<meta name="theme-color" content="#3498db">

</head>

<body>

<header>

<h1>Lista de Tarefas</h1>

</header>

<main>

<div class="task-input">

<input type="text" id="task-text" placeholder="Adicione uma nova tarefa...">

<button id="add-task-btn">+</button>

</div>

<ul id="task-list">

</ul>

</main>

</body>

</html>

* **meta name="viewport"**: Essencial para que seu aplicativo seja responsivo e se ajuste à tela de qualquer dispositivo.
* **link rel="manifest"**: Este é um arquivo de texto simples que descreve o seu aplicativo, como nome, ícones e cores, permitindo que ele seja "instalável" em dispositivos. Vamos criar esse arquivo no próximo passo.
* **meta name="theme-color"**: Define a cor da barra de status do navegador no Android, dando uma aparência mais nativa.
* **Links para ícones**: Crie uma pasta icons e adicione os ícones com os tamanhos especificados. Eles são usados quando o usuário adiciona o PWA à tela inicial.

### **2. O Arquivo Manifest**

Crie um arquivo chamado manifest.json na mesma pasta do index.html.

JSON

{

"name": "Minha Lista de Tarefas",

"short\_name": "Tarefas",

"start\_url": "./",

"display": "standalone",

"background\_color": "#ffffff",

"theme\_color": "#3498db",

"description": "Um aplicativo de lista de tarefas simples, feito como um PWA.",

"icons": [

{

"src": "icons/icon-192x192.png",

"sizes": "192x192",

"type": "image/png"

},

{

"src": "icons/icon-512x512.png",

"sizes": "512x512",

"type": "image/png"

}

]

}

* **name e short\_name**: Nome completo e um nome mais curto para a tela inicial.
* **start\_url**: A página inicial do seu PWA.
* **display: "standalone"**: Diz ao navegador para exibir o aplicativo como uma janela independente, sem a barra de endereço e outros elementos do navegador, como um aplicativo nativo.
* **icons**: Caminhos para os ícones que seu PWA usará.

### 

### 

### **3. Estilização com CSS**

Crie um arquivo chamado style.css. Aqui, vamos dar vida ao nosso aplicativo com um design limpo e moderno.

CSS

/\* Estilos gerais \*/

body {

font-family: Arial, sans-serif;

background-color: #f0f2f5;

margin: 0;

padding: 20px;

color: #333;

}

header {

text-align: center;

padding: 15px;

background-color: #3498db;

color: white;

border-radius: 8px 8px 0 0;

margin-bottom: 20px;

}

h1 {

margin: 0;

}

main {

max-width: 600px;

margin: 0 auto;

background-color: white;

padding: 20px;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0,0,0,0.1);

}

/\* Estilos para a entrada de tarefas \*/

.task-input {

display: flex;

margin-bottom: 20px;

}

#task-text {

flex-grow: 1;

padding: 10px;

border: 1px solid #ccc;

border-radius: 4px;

font-size: 16px;

outline: none;

}

#add-task-btn {

padding: 10px 15px;

background-color: #27ae60;

color: white;

border: none;

border-radius: 4px;

font-size: 20px;

cursor: pointer;

margin-left: 10px;

}

#add-task-btn:hover {

background-color: #2ecc71;

}

/\* Estilos para a lista de tarefas \*/

#task-list {

list-style: none;

padding: 0;

}

#task-list li {

background-color: #ecf0f1;

padding: 15px;

margin-bottom: 10px;

border-radius: 4px;

display: flex;

justify-content: space-between;

align-items: center;

}

#task-list li.completed {

text-decoration: line-through;

color: #888;

background-color: #e0e0e0;

}

### **4. Próximos Passos (Adicionando a Funcionalidade PWA)**

Com esses três arquivos, você já tem a base visual. Para que ele funcione como um PWA completo, você precisa adicionar um **Service Worker**. O Service Worker é um script JavaScript que fica entre a sua página e a rede, permitindo que o navegador armazene em cache os arquivos do seu aplicativo para que ele funcione offline.

**Para adicionar o Service Worker:**

1. Crie um arquivo chamado sw.js.

No seu index.html, adicione um bloco de script JavaScript para registrar o Service Worker:

<script>

if ('serviceWorker' in navigator) {

window.addEventListener('load', () => {

navigator.serviceWorker.register('/sw.js').then(registration => {

console.log('Service Worker registrado com sucesso:', registration);

}).catch(error => {

console.log('Falha ao registrar Service Worker:', error);

});

});

}

</script>

2- No arquivo sw.js, adicione o código para armazenar em cache os arquivos:

const CACHE\_NAME = 'pwa-task-list-v1';

const urlsToCache = [

'/',

'/index.html',

'/style.css',

'/manifest.json',

'/icons/icon-192x192.png',

'/icons/icon-512x512.png'

];

self.addEventListener('install', event => {

event.waitUntil(

caches.open(CACHE\_NAME)

.then(cache => {

return cache.addAll(urlsToCache);

})

);

});

self.addEventListener('fetch', event => {

event.respondWith(

caches.match(event.request)

.then(response => {

return response || fetch(event.request);

})

);

});

3- Depois de criar esses arquivos, você precisa de um servidor web local para testar a funcionalidade PWA (o Service Worker não funciona em arquivos locais, como file:///). Você pode usar ferramentas simples como o **Live Server** no VS Code para isso.