UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS VII - PATOS
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CURSO DE GRADUAÇÃO EM BACHARELADO EM COMPUTAÇÃO

CALLYEL NUNES ALVES

INTEGRAÇÃO DE APLICAÇÕES WEB PROGRESSIVAS (PWA) COM ESP32

PATOS - PB
2025

CALLYEL NUNES ALVES

INTEGRAÇÃO DE APLICAÇÕES WEB PROGRESSIVAS (PWA) COM ESP32

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Computação do Departamento de Computação do Centro de Ciências Exatas e Sociais Aplicadas da Universidade Estadual da Paraíba, como requisito para obtenção do título de bacharel em Computação.

Orientador: Esp. Jaian Tales Gomes Santos

PATOS - PB
2025

A474i Alves, Callyel Nunes.
Integração de aplicações w*eb* progressivas (PWA) com ESP32 [manuscrito] / Callyel Nunes Alves. - 2025.
39 f. : il. color.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Ciência da computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas, 2025.

"Orientação : Prof. Esp. Jaian Tales Gomes Santos, Coordenação do Curso de Computação - CCEA".

1. Progressive Web App. 2. ESP32. 3. WebSocket. 4. Internet das coisas. I. Título

21. ed. CDD 004.678

Elaborada por Kaliane Eveny Martins de Oliveira - CRB - 15/986 BSC7

CALLYEL NUNES ALVES

INTEGRAÇÃO DE APLICAÇÕES WEB PROGRESSIVAS (PWA) COM ESP32

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação

Aprovada em: 03/06/2025.

BANCA EXAMINADORA

Documento assinado eletronicamente por:

- **Jaian Tales Gomes Santos** (***.796.864-**), em **10/06/2025 21:06:39** com chave **f281d15c465711f09cf92618257239a1**.
- **Lucas Henrique Oliveira de Araújo** (***.324.514-**), em **11/06/2025 00:15:03** com chave **44819d9c467211f0a12e1a1c3150b54b**.
- **José Aldo Silva da Costa** (***.862.334-**), em **11/06/2025 19:56:52** com chave **5d5dcf20471711f0957f06adb0a3afce**.

Documento emitido pelo SUAP. Para comprovar sua autenticidade, faça a leitura do QrCode ao lado ou acesse https://suap.uepb.edu.br/comum/autenticar_documento/ e informe os dados a seguir.
Tipo de Documento: Folha de Aprovação do Projeto Final
Data da Emissão: 12/06/2025
Código de Autenticação: f96baa

Dedico este trabalho à minha família, que sempre me apoiou e estiveram ao meu lado. Aos meu amigos que estiveram comigo durante toda essa jornada. Muito obrigado!

AGRADECIMENTOS

Primeiro, agradeço a Deus por me guiar, dar força e sempre estar comigo.

Agradeço a minha família, por todo o apoio e por sempre me incentivarem a seguir em frente e nunca desistir.

Agradeço também ao meu orientador, Jaian, pela contribuição com ideias, e pelas correções que ajudaram a tornar este trabalho melhor.

Aos colegas do curso de computação, que fizeram parte dessa caminhada e tornaram tudo mais leve.

E agradeço aos professores, que contribuíram para a minha evolução e fizeram possível esse trabalho.

“A paciência é um elemento fundamental do sucesso."

Bill Gates

RESUMO

Em um mundo onde conectividade está cada vez mais presente em nossas vidas, é comum as pessoas procurarem por agilidade e flexibilidade em tecnologias web e mobile. Nesse contexto, surge o desafio de integrar aplicações web modernas com dispositivos embarcados. Este trabalho tem como objetivo demonstrar a viabilidade de integração entre Aplicações Web Progressivas (PWAs) e o microcontrolador ESP32, por meio da implementação de uma solução para a conexão em tempo real via protocolo *WebSocket*. O presente estudo fundamenta-se em uma abordagem qualitativa e exploratória, utilizando a metodologia de Desenvolvimento Rápido de Aplicações (*RAD*), para implementação. Neste estudo foi desenvolvido um protótipo funcional que explora a comunicação entre uma PWA e ESP32. A arquitetura desenvolvida demonstrou-se genérica e adaptável a diversos contextos, como automação residencial e acionamento remoto de dispositivos. Os resultados indicam que a integração proposta é viável, eficiente e promissora.

Palavras-chaves: *Progressive Web App*, ESP32, *WebSocket*, Internet das Coisas.

ABSTRACT

In a world where connectivity is increasingly present in our lives, it is common for people to seek agility and flexibility in web and mobile technologies. In this context, the challenge of integrating modern web applications with embedded devices arises. This work aims to demonstrate the feasibility of integration between Progressive Web Applications (PWAs) and the ESP32 microcontroller through the implementation of a real-time connection solution using the WebSocket protocol. The present study is based on a qualitative and exploratory approach, using the Rapid Application Development (RAD) methodology for implementation. In this study, a functional prototype was developed to explore communication between a PWA and the ESP32. The developed architecture proved to be generic and adaptable to various contexts, such as home automation and remote device control. The results indicate that the proposed integration is feasible, efficient, and promising.

Keywords: Progressive Web App, ESP32, WebSocket, Internet of Things.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IoT	Internet of Things
Wi-Fi	Wireless Fidelity
PWA	Progressive Web App
HTTP	Hyper Text Transfer Protocol
RFID	Radio Frequency Identification
NFC	Near Field Communication
IP	Internet Protocol
IEEE	Institute of Electrical and Electronics Engineers
FPGAs	Field Programmable Gate Array
RDF	Resource Description Framework
OWL	Web Ontology Language
EXI	Efficient XML Interchange
XML	Extensible Markup Language
USB	Universal Serial Bus
SDK	Software Devolopment Kit
IDE	Integrated Development Environment
MQTT	Message Queuing Telemetry Transport
HTML	HyperText Markup Language
HTTPS	Hyper Text Transfer Protocol Secure
JSON	JavaScript Object Notation
PC	Personal Computer
URL	Uniform Resource Locator
TLS	Transport Layer Security
DOM	Document Object Model
SO	Sistema Operacional

IETF	Internet Engineering Task Force
TCP	Transmission Control Protocol
API	Application Programming Interface
RAD	Rapid Application Development
CSS	Cascading Style Sheets
SSID	Service Set Identifier

SUMÁRIO

1 INTRODUÇÃO

As *Progressive Web Apps* (PWAs) são aplicações web que buscam oferecer uma experiência semelhante a de aplicativos nativos. Essa abordagem vem se destacando no desenvolvimento de software por permitir que aplicações web funcionem de forma independente do navegador, com experiência próxima a de um aplicativo instalado.

Por outro lado, os microcontroladores têm desempenhado um papel fundamental no avanço da Internet das Coisas *(IoT)*. Entre eles, o ESP32 se destaca por ser um microcontrolador de baixo custo, com conectividade Wi-Fi e Bluetooth integrada. Para ter comunicação entre uma aplicação PWA e um dispositivo como o ESP32, é necessário utilizar protocolos leves e eficientes.

Nesse cenário, o *WebSocket* surge como uma tecnologia que permite comunicação bidirecional e em tempo real entre cliente e servidor, tornando-se uma alternativa promissora para integrar aplicações web com dispositivos físicos.

1.1 Problemática

É sabido que vivemos em um mundo que cada vez mais a rapidez, eficiência e o baixo custo tornam-se requisitos essenciais para qualquer solução de tecnologia. É nesse cenário que as *Progressive Web Apps* (PWAs) surgem como uma boa alternativa. Elas combinam a flexibilidade de uma página web com serviços nativos de um aplicativo.

Para realizar a integração entre uma PWA e um microcontrolador é preciso garantir que uma comunicação em tempo real que seja confiável e leve entre o navegador e o hardware. O *WebSocket*, protocolo de comunicação em tempo real, é uma das opções para fazer essa integração. Mas como fazer essa conexão de maneira eficiente? Quais são as limitações de desempenho? Em aplicações práticas, como na robótica, essas perguntas passam da teoria para o dia a dia do desenvolvedor e do usuário final.

1.2 Objetivo

Objetivo Geral:

Investigar a viabilidade da integração entre uma *Progressive Web Apps* (PWAs) e o microcontrolador ESP32, com o intuito de desenvolver uma solução de conexão entre ambas as partes.

Objetivos Específicos:

- Desenvolver uma PWA com interface intuitiva para controle de um robô com rodas;
- Implementar a comunicação via *WebSocket* entre a PWA e o ESP32.
- Propor uma arquitetura genérica e reutilizável para outros cenários de *IoT*.

1.3 Justificativa

A escolha do tema justifica-se pela crescente relevância das *Progressive Web Apps* no desenvolvimento de aplicações e pela necessidade de soluções mais acessíveis para automação e controle remoto de dispositivos.

A integração de PWAs com microcontroladores como o ESP32 representa uma área promissora e ainda pouco explorada, que pode ampliar as possibilidades de uso desses dispositivos em projetos de baixo custo e grande alcance. Além disso, a comunicação em tempo real via *WebSocket* oferece uma alternativa eficiente para o controle de aplicações, o que contribui para o avanço de tecnologias aplicadas à Internet das Coisas (*IoT*).

Este trabalho contribui para o desenvolvimento de soluções práticas e escaláveis, com potencial de aplicação em diversas áreas. Ao explorar a integração entre PWAs e microcontroladores, este estudo reforça o papel da inovação tecnológica utilizando recursos viáveis e de baixo custo.

1.4 Metodologia

Este trabalho adota uma abordagem qualitativa e exploratória, com ênfase na aplicação prática de tecnologias emergentes no contexto da Internet das Coisas. O desenvolvimento baseia-se na metodologia de Desenvolvimento Rápido de Aplicações (RAD), priorizando agilidade e funcionalidades essenciais. A comunicação entre a PWA e o microcontrolador ESP32 ocorre via *WebSocket*, e a interface da aplicação foi construída para o controle remoto de um robô com rodas com foco em simplicidade e usabilidade.

1.5 Estrutura do trabalho

Este trabalho está estruturado em cinco seções. A primeira apresenta a introdução ao tema, incluindo a problemática, os objetivos, a justificativa, a metodologia e a estrutura do trabalho. A segunda seção aborda os fundamentos teóricos, discutindo os conceitos de internet das coisas (*IoT*), ESP32, PWA, *Web App Minifest*, *Service Worker* e *WebSocket*. Na terceira seção, são apresentados os aspectos metodológicos. A quarta seção expõe os procedimentos experimentais, detalhando o desenvolvimento e a implementação da PWA. Por fim, a quinta seção traz as conclusões do estudo, destacando as contribuições da pesquisa e possíveis direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta os conceitos fundamentais para o desenvolvimento deste trabalho, abordando as tecnologias envolvidas na integração entre a PWA e o microcontrolador ESP32. São apresentados os princípios da Internet das Coisas *(IoT)*, as características do ESP32, os fundamentos das PWAs, bem como elementos essenciais para seu funcionamento, como *Web App Manifest*, *Service Worker* e o protocolo *WebSocket*. Por fim, discute-se a metodologia de Desenvolvimento Rápido de Aplicações *(RAD)*, adotada no projeto.

2.1 *Internet of Things*

A internet das coisas (*IoT*) pode ser aplicada em várias áreas, como casas inteligentes, cidades inteligentes, segurança, etc. De acordo com Santos et al. (2016), a internet das coisas emergiu dos avanços nas áreas como sistemas embarcados, microeletrônica, comunicação e sensoriamento.

Os dispositivos da Internet das Coisas (*IoT*), abrangendo desde sensores elementares até robôs de alta complexidade, têm experimentado avanços notáveis em aspectos como capacidade de processamento, eficiência energética e miniaturização (Miorandi et al., 2012).

Para Santos et al. (2016) a *IoT* pode ser entendida como a combinação de diversas tecnologias, as quais são complementares no sentido de viabilizar a integração dos objetos no ambiente físico ao mundo virtual. Ainda segundo Santos et al. (2016) existem alguns blocos básicos de construção da *IoT*, sendo eles:

Identificação: Nesta etapa, a identidade de um objeto conectado é definida para que ele possa ser individualmente reconhecido, monitorado e gerenciado. A identificação é possivel com a ajuda de tecnologias como Identificação por Rádio Frequência (*RFID*), Comunicação por Campo de Proximidade (*NFC*) e endereçamento IP. A identificação é a base para rastrear dispositivos, controlar o acesso e fornecer serviços personalizados.

Sensores e Atuadores: Sensores são os elementos capazes de perceber variáveis físicas ou ambientais, como temperatura, umidade, luminosidade e movimento, entre outros. Transformando os comprimentos das informações em sinais digitais que são aceitáveis para serem processadas por sistemas computacionais. Atuadores, por sua vez, são os elementos recebedores de comando que fazem ou continuam fazendo uma ação no ambiente, por exemplo: ligar motores, válvula e displays entre outros. A interação entre os sensores e atuadores permite que o sistema *IoT* reaja de forma dinâmica a seu meio.

Comunicação: O bloco que trata das tecnologias de comunicação, ou seja, das tecnologias que abrigam a troca de dados entre os dispositivos, os servidores e as plataformas de análise. A escolha do protocolo ou padrão é responsável pelo alcance e desempenho da solução *IoT*. Os protocolos mais escolhidos são *Wi-Fi*, *Bluetooth*, protocolos *IEEE 802.15.4* e novamente *RFID* no papel funcional.

Computação: refere-se às unidades de processamento embaladas, isto é, microcontroladores,

processadores e *Field Programmable Gate Array* (FPGA), que são dispositivos lógicos reconfiguráveis programáveis pelo usuário após a fabricação. Essas unidades são destinadas à execução de algoritmos locais nos dispositivos individuais.

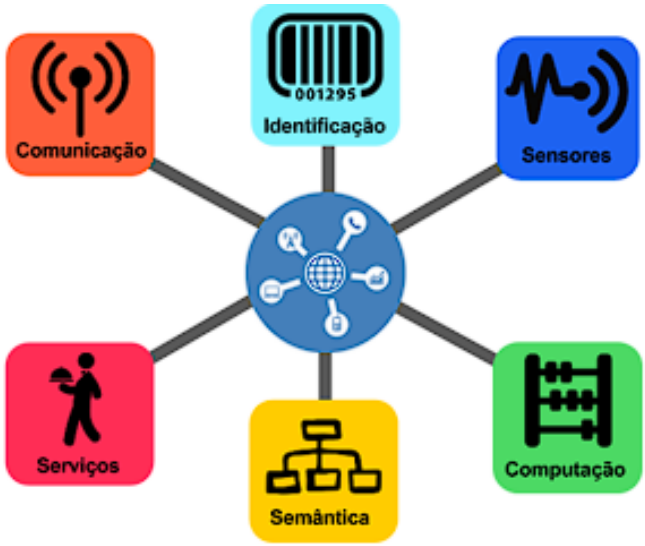
Serviços são oferecidos na *IoT*:

- Identificação: associa objetos físicos a representações digitais;
- Agregação de dados: coleta e resumem informações dos dispositivos;
- Colaboração e Inteligência: analisa os dados acumulados para tomada de decisões;
- Ubiquidade: impõe que os serviços estejam disponíveis a todo o tempo e em todo o lugar.

Semântica: Trata da interpretação dos dados para gerar conhecimento e oferecer serviços inteligentes. Utiliza tecnologias como *Resource Description Framework* (RDF), que estrutura dados facilitando sua integração e interpretação por máquinas. O *Web Ontology Language* (OWL), permite modelos formais que descreve conceitos e relações entre eles em um dominio específico. E o *Efficient XML Interchange* (EXI), um formato binário compactado para representação de dados XML, otimizado para dispositivos com restrições de memória e largura de banda, comum em ambientes de *IoT*.

A seguir a Figura 1 mostra cada um dos blocos básicos para a construção da *IoT*.

Figura 1 – Blocos básicos da *IoT*



Fonte: Santos et al. (2016).

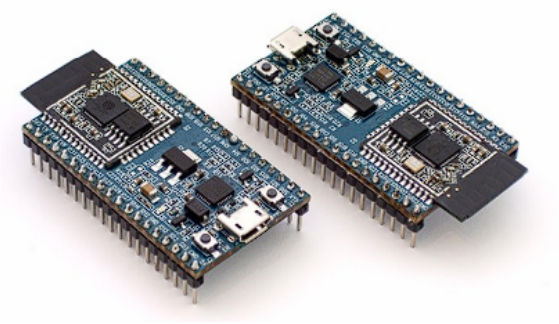
2.2 ESP32

O microcontrolador ESP32, desenvolvido pela empresa *Espressif Systems*, empresa chinesa com sede em Xangai, é considerado uma rede de WiFi independente, sendo como uma ponte entre os microcontroladores existentes (Kolban, 2018). O processador ESP32 é um

dispositivo que apresenta suporte a redes Wi-Fi e conexão Bluetooth, fazendo da mesma uma excelente alternativa para conexões entre dispositivos, facilitando a integração entre eles (Ibrahim, 2017).

O ESP32 possui um módulo exclusivo para desenvolvimento chamado “ESP32-DevKit”. De acordo com Kolban (2018) a placa contém conectores, bem como um adaptador micro USB e dois botões chamados enable e boot. Esses botões podem ser usados para "*flash*"ou "*download*"de um novo código.

Figura 2 – *ESP32-DevKitC*

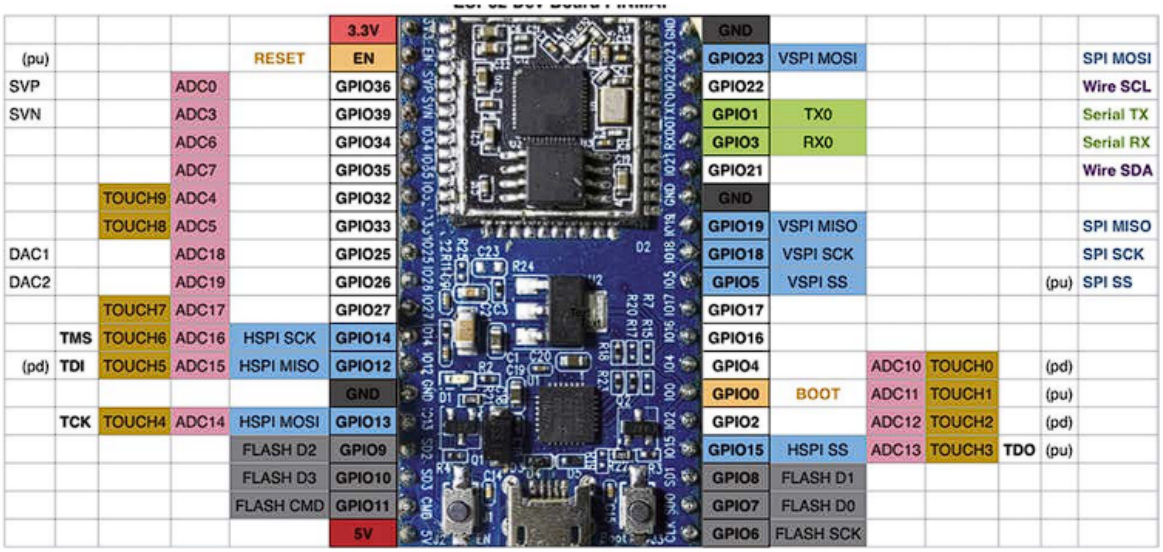


Fonte: Kolban (2018).

Observa-se na Figura 2, suas dimensões de pequeno porte pode ser facilmente incorporado em placas de circuitos eletrônicos, além disso, sua programação é feita através da linguagem C/C++, ela pode ser feita em seu próprio ambiente de programação o *Software Devolopment Kit* (SDK), o ESP32 também pode ser programado por meio da interface do Arduino, conhecida como Arduino IDE.

A placa recebe energia através da porta USB, contém vários pinos conectores que podem ser ligados a módulos para executar alguma ação. Conforme aponta Ibrahim (2017), os pinos na placa *ESP32 DevKitC* têm múltiplas funções. O pino 10 é compartilhado com as funções *GPIO* port 26, *DAC* channel 2, *ADC* channel 19, *RTC* channel 7 e RX01.

Figura 3 – Funções de cada pino



Fonte: Ibrahim (2017).

Na Figura 3 pode-se entender as funções de cada pino e mapear seus botões. Para garantir um funcionamento adequado da placa e evitar danos aos seus componentes, é importante entender sobre sua alimentação de energia. A placa é projetada para operar com uma tensão típica de 3,3 V, sendo 3,6 V o limite máximo suportado. Em relação à corrente elétrica, recomenda-se que cada pino não ultrapasse 6 mA. O valor absoluto máximo permitido é de 12 mA (Ibrahim, 2017).

A comunicação em sistemas de Internet das Coisas (*IoT*) pode ser realizada por meio de diferentes protocolos, cada um com características específicas. O protocolo *Hypertext Transfer Protocol* (HTTP) é amplamente utilizado para a transferência de dados na web, baseado em um modelo de requisição-resposta. O *Message Queuing Telemetry Transport (MQTT)*, por sua vez, é um protocolo projetado para dispositivos com restrições de recursos, utilizando uma arquitetura de publicação e assinatura para a troca de mensagens. Já o *WebSocket* é uma tecnologia que possibilita a comunicação bidirecional em tempo real entre cliente e servidor, mantendo uma conexão persistente sem a necessidade de múltiplas requisições.

Com a capacidade de se comunicar em tempo real e operar protocolos como HTTP, *MQTT* e *WebSocket*, o ESP32 oferece uma conexão confiável e essencial para muitas aplicações de *IoT*. O uso do protocolo *WebSocket* permite uma conexão bidirecional com o ESP32, exercendo um papel crucial na troca de mensagens entre o microcontrolador e a aplicação.

Pode-se entender que o ESP32 se destaca como uma opção versátil e eficiente para projetos de *IoT* e sistemas embarcados, combinando bom desempenho, baixo consumo de energia e ótimas capacidades de comunicação.

2.3 *Progressive Web Apps (PWAs)*

Os *Progressive Web Apps (PWAs)* são aplicativos criados a partir do uso de tecnologias baseadas na web e são usadas em navegadores de dispositivos multiplataforma. Segundo Pontes (2018) a aplicação deve ser acessível para todos, sejam microcomputadores ou smartphones, com *browsers* atualizados ou obsoletos e com conexão à internet ou não.

Com a evolução da tecnologia nos navegadores, pode-se obter algumas funcionalidades usando apenas tecnologias web como HTML, linguagem de marcação responsável por estruturar o conteúdo das páginas, e JavaScript, uma linguagem de programação voltada à interatividade e ao dinamismo das interfaces. Essas tecnologias permitem, a adaptação para ambientes desktop, caracterizando esse tipo de aplicação como uma PWA. Segundo a análise de Hume (2017), uma PWA deve apresentar as seguintes características:

- **Responsividade**

  Entende-se que é recomendado utilizar técnicas de responsividade no desenvolvimento de uma PWA. Um site com web design responsivo - ou *responsive web design* - pode ser acessado de um PC, notebook, smartphone, tablet, TV, geladeira, entre outras. (Zemel, 2015, p. 9).

- **Independente de conectividade**

  A maioria das aplicações web ainda depende totalmente da internet. Quando acessados sem internet, esses sites simplesmente não mostram conteúdo para os usuários. Visando melhorar a experiência dos usuários criou-se o *Offline First*. O conceito de *Offline First* é criar uma aplicação web começando no cenário mais limitado, ou seja, sem internet.

- **Interativa**

  Semelhante aos aplicativos nativos para os usuários, com interações e navegação de estilo de aplicativos porque é compilado no modelo para o *shell* de um aplicativo.

- **Sempre atualizada**

  Com o *Service Worker* é possível manter a aplicação em constante atualização, segundo Hume (2017) os *Service Workers* são scripts que rodam em segundo plano. Escritos na linguagem de programação JavaScript, eles permitem que um desenvolvedor intercepte requisições de rede, gerencie mensagens *push* e execute muitas outras tarefas.

- **Segura**

  Para a utilização do *Service Worker* é obrigatório que o site tenha um certificado de segurança HTTPS. Segundo Hume (2017), os Service Workers podem ser explorados por agentes maliciosos para interceptar e redirecionar conexões da aplicação, o que justifica a exigência de uso restrito a conexões HTTPS, como forma de garantir a integridade e a segurança das requisições.

- **Descoberta pelos motores de busca**

  É de extrema importância a criação de um manifesto de aplicativo web (*Web App Manifest*), um arquivo no formato *JSON* onde ficam definidas algumas informações da aplicação. Nesse arquivo deve possuir metadados que façam com que os motores de busca e os dispositivos reconheçam que aquele *website* é um PWA.

- **Capaz de reengajar o usuário**

  Facilitar o reengajamento com recursos como notificações *push*. Conforme aponta Hume (2017), aplicativos nativos normalmente eram os únicos capazes de enviar notificações *push* e acessar o sistema operacional de um dispositivo. É aqui onde os PWAs são cruciais. Eles podem receber notificações *push* do navegador, trazendo assim mais engajamento para o usuário.

- **Instalável**

  Como aponta, Trindade e Affini (2019) o PWA permite que os usuários incluam os aplicativos que consideram úteis em suas telas iniciais sem precisar buscar aplicativos na loja. Complementando LePage e Richard (2020) afirmam, a instalação de um PWA oferece novos recursos, como atalhos de teclado, que normalmente são reservados a um navegador. Além disso, os PWAs podem se registrar para aceitar conteúdo de outras aplicações ou servir como a aplicação padrão para processar certos tipos de arquivo.

- **Ter links compartilháveis**

  Os PWAs são executados diretamente nos mesmos navegadores que os sites básicos, navegando em dispositivos usando *URLs* padrão. Cada página da aplicação deve conter uma URL específica para que possa ser compartilhada e qualquer pessoa consiga ter acesso a essa página.

A integração da PWA atende a três requisitos essenciais: Conexão HTTPS, *Service Worker* e *Web App Manifest*. O Manifesto do Aplicativo Web é um arquivo no formato *JavaScript Object Notation (JSON)* que traz informações importantes, como o ícone, nome, autor e descrição do aplicativo.

O Manifesto garante que o *app* seja exibido de forma adequada na tela inicial, permitindo seu funcionamento offline e a recepção de notificações do tipo *push*. Os Service Workers são scripts executados em segundo plano pelo navegador, permitindo funcionalidades como notificações push e sincronizações em segundo plano, sem a necessidade de interação direta do usuário (Trindade; Affini, 2019, p. 12).

2.4 *Web App Manifest*

Para que uma aplicação web seja reconhecida como uma PWA de verdade pelos navegadores, é necessário incluir um arquivo chamado *Web App Manifest*. Segundo Trindade e Affini

(2019) o Manifesto do Aplicativo Web, conhecido como *Web App Manifest*, fornece informações sobre o aplicativo (como nome, autor, ícone e descrição) e como ele deve se comportar.

Mazzarolo et al. (2021) aponta que o navegador baseia-se na leitura do arquivo *manifest* para ter a informação se determinado site trata-se de um PWA. Com isso, entende-se que o arquivo *manifest* é essencial para os motores de busca identificarem o site como PWA.

2.5 *Service Worker*

O *Service Worker* é um script que permite com que a aplicação funcione de forma inteligente. Um *Service Worker* é um bloco de código JavaScript que um aplicativo web instala no navegador e é executado quando o navegador precisa (com base em eventos que são acionados (Wargo, 2020).

Ainda segundo Wargo (2020) o navegador só instala o *service worker* se:

- O navegador oferecer suporte aos *Service Workers*.
- O navegador carregou o aplicativo web usando uma conexão *TLS (HTTPS)* ou a partir do endereçamento local (localhost).
- O aplicativo web carrega o código *Service Worker* dentro do mesmo contexto (do mesmo servidor local).

O *Service Worker* é um dos responsáveis por ser possível utilizar alguns mecanismos que só poderiam ser utilizados baixando um aplicativo. Por meio dele, é possível implementar recursos offline e organizar o cache dos arquivos estáticos da página web (Pontes, 2018).

Para que uma aplicação PWA funcione sem conexão com a internet é preciso implementar um registro de um *Service Worker* (para salvar arquivos no *Cache Storage*). Entretanto, nem todo navegador suporta à tecnologia *Service Workers*. A Figura 4 mostra os navegadores que atualmente oferecem suporte à tecnologia *Service Workers*, destacando a compatibilidade entre diferentes versões. Esses dados foram obtidos por meio da plataforma "Can I use", que fornece informações atualizadas sobre suporte a recursos da web em navegadores.

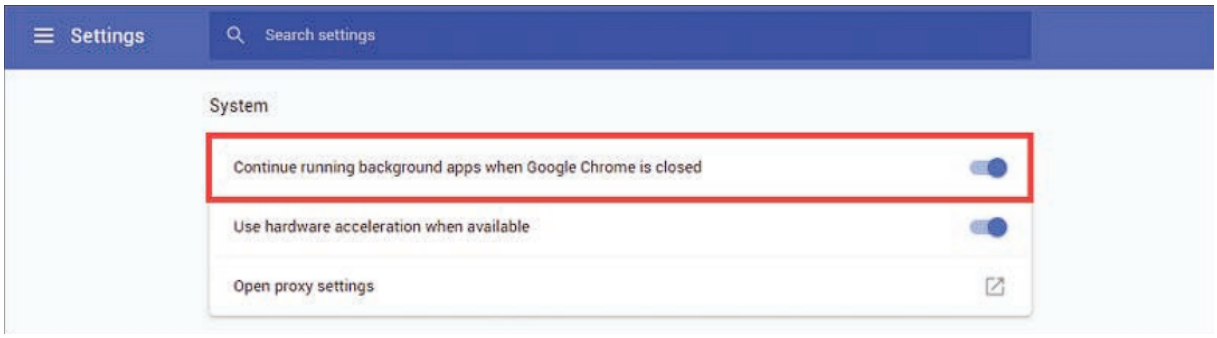
Figura 4 – Compatibilidade de *Service Workers* entre navegadores e suas versões

Fonte: Can I use (2025).

Os *Service Workers* também possuem limitações. Wargo (2020) aponta que os *Service Workers* não são executados a menos que o navegador esteja aberto e eles também não tem acesso ao modelo de acesso do documento *(DOM)*. A questão dele não funcionar quando o navegador não estiver aberto pode ser contornada dependendo do navegador e do sistema operacional (SO).

Alguns navegadores também são executados em segundo plano ou podem ser configurados para isso. Um exemplo é o *Chrome*, ele possui uma configuração avançada que continua executando tarefas em segundo plano quando o *Google Chrome* é fechado. A Figura 5 destaca essa configuração.

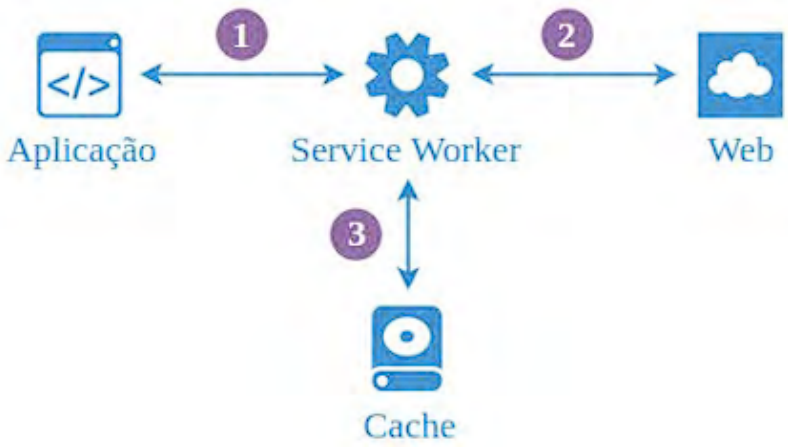
Figura 5 – Configurações avançadas do *Chrome*



Fonte: Wargo (2020).

Para o funcionamento offline da aplicação o *Service Worker* intercepta as requisições que o navegador faria pro servidor e guardar as respostas no cache. Ou seja, quando o usuário abrir a aplicação de novo (mesmo sem conexão), o *Service Worker* consegue devolver os arquivos direto do cache, fazendo com que tudo continue funcionando.

Figura 6 – Fluxo do *Service Worker*



Fonte: Pontes (2018).

A Figura 6 mostra o fluxo básico de funcionamento de um *Service Worker*. Ele atua como um intermediário entre a aplicação e a internet. Quando o app faz uma requisição (1), o *Service Worker* pode decidir buscar os dados na web (2) ou responder direto com algo que já está salvo no cache (3). Isso é útil principalmente quando o usuário está sem internet, pois permite que a aplicação continue funcionando com os arquivos que já foram armazenados.

2.6 *WebSocket*

O protocolo *WebSocket* teve origem em 2011 e foi lançado pelo *Internet Engineering Task Force (IETF)*, tendo como sua principal característica a comunicação em tempo real. O *WebSocket* reduz a latência, pois, uma vez que a conexão é estabelecida, o servidor pode enviar mensagens assim que estiverem disponíveis. O *WebSocket* faz apenas uma única requisição, o servidor não precisa aguardar uma solicitação do cliente (Wang; Salim; Moskovits, 2013).

O *WebSocket* funciona de forma bem simples, primeiro ele começa como se fosse uma conexão normal HTTP, só pra "abrir caminho" entre o cliente e o servidor usando um único *socket TCP*. Depois que essa conexão é feita, o cabeçalho HTTP é trocado por um cabeçalho específico do *WebSocket*. Com isso, a conversa entre cliente e servidor rola nos dois sentidos ao mesmo tempo, de forma contínua e dedicada.

O protocolo *WebSocket* é projetado para ser utilizado em navegadores da web, mas ele também pode ser utilizado para propósitos diferentes. Esse protocolo permite o envio de mensagens que podem conter texto com codificação *Unicode Transformation Format – 8 bits (UTF-8)* ou dados binários (Farias; Vasconcelos, 2019). A codificação *UTF-8* é um padrão amplamente utilizado para representar caracteres de texto, sendo capaz de codificar todos os caracteres do conjunto Unicode, o que garante maior compatibilidade e intercâmbio de dados entre diferentes sistemas e plataformas.

Além disso, a comunicação por meio do *WebSocket* é estruturada com o apoio de uma

Application Programming Interface (API). Uma API é um conjunto de definições e protocolos que facilita a comunicação entre diferentes softwares, permitindo que desenvolvedores utilizem funcionalidades prontas de forma padronizada e simplificada. No caso do *WebSocket*, a API disponibilizada define os eventos e métodos necessários para o estabelecimento, manutenção e encerramento da conexão, além do envio e recebimento de mensagens. É baseada em eventos e de fácil implementação. A Tabela 1 descreve os quatro eventos que compõem essa API (Farias; Vasconcelos, 2019):

Tabela 1 – Eventos *API Websocket*

Evento	**Descrição**
onopen	Chamado quando a conexão *Websocket* é estabelecida.
onmessage	Chamado quando uma mensagem é recebida.
onclose	Chamado quando a conexão *Websocket* é fechada devido ao recebimento de *handshake* ou falha.
onerror	Chamado quando ocorre erro devido à memória *buffer* cheia.

Fonte: Farias e Vasconcelos (2019).

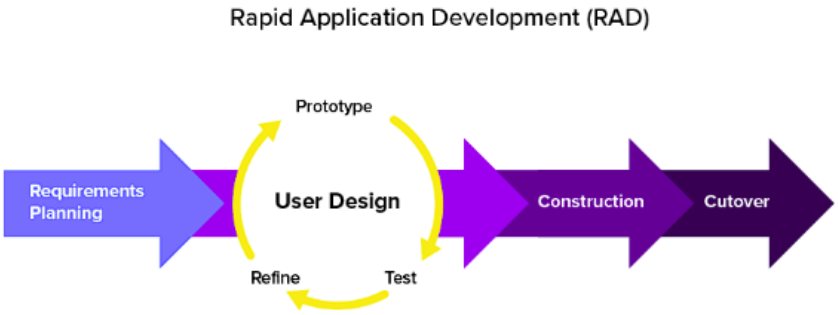
A introdução do protocolo *WebSocket* tem sido considerado um marco importante na comunicação em tempo real na web. Esse avanço no modelo de comunicação é atribuído a fatores essenciais como eficiência, flexibilidade e maior capacidade de interação.

2.7 Desenvolvimento Rápido de Aplicações (*RAD*)

A metodologia de Desenvolvimento Rápido de Aplicações (*RAD*) surgiu como uma alternativa aos métodos tradicionais de desenvolvimento de software, com o objetivo de acelerar a entrega de sistemas por meio de ciclos curtos e iterativos. Segundo Vasudevan e Wilemon (1997) o *RAD* é caracterizado como uma abordagem que visa à construção de sistemas computacionais combinando ferramentas e técnicas de engenharia de software assistida por computador (*CASE*), prototipação orientada ao usuário e ciclos curtos de desenvolvimento.

O ciclo de vida do *RAD* é composto por fases como planejamento de requisitos, design de usuário, prototipação, testes, refinamentos, desenvolvimento e implementação, conforme ilustrado na Figura 7.

Figura 7 – Ciclo de vida *RAD*



Fonte: Araújo e Tanaka (2024).

- **Planejamento de requisitos**: A fase de planejamento de requisitos no *RAD* reúne elementos das etapas de planejamento e análise de sistemas do ciclo de vida tradicional. Nessa fase, usuários, gestores e a equipe de TI colaboram para definir as necessidades do negócio, escopo do projeto, restrições e requisitos do sistema (Shelly; Rosenblatt, 2012).
- **Design de usuário**: Na fase de design do usuário, há uma interação constante entre os usuários e os analistas de sistemas, visando a criação de modelos e protótipos que representem os processos, entradas e saídas do sistema. Essa etapa é altamente iterativa, permitindo que os usuários compreendam, modifiquem e aprovem gradualmente um modelo funcional da aplicação (Shelly; Rosenblatt, 2012).
- **Construção**: A fase de construção no *RAD* envolve o desenvolvimento dos programas e aplicações, de forma semelhante ao ciclo de vida tradicional. No entanto, diferencia-se por manter a participação ativa dos usuários, que podem continuar sugerindo alterações ou melhorias mesmo durante o desenvolvimento das telas.
- ***Cutover***: Fase de transição que corresponde às etapas finais de implementação, incluindo a conversão de dados, testes, treinamento dos usuários e a substituição do sistema antigo pelo novo. De acordo com Shelly e Rosenblatt (2012) no *RAD*, esse processo é significativamente mais rápido do que nos métodos tradicionais, permitindo que o sistema seja entregue e colocado em operação em um prazo reduzido

O *RAD* apresenta vantagens e desvantagens em relação aos métodos tradicionais. Segundo Shelly e Rosenblatt (2012), sua principal vantagem está na agilidade no desenvolvimento de sistemas, o que pode gerar economia significativa de tempo e custos. Por outro lado, uma de suas limitações é o foco excessivo nos aspectos técnicos do sistema, em detrimento das necessidades estratégicas da organização. Isso pode resultar em soluções eficazes no curto prazo, mas desalinhadas com os objetivos corporativos de longo prazo.

Outra grande desvantagem é que o ciclo acelerado pode oferecer menos tempo para desenvolver padrões de qualidade, consistência e design. No entanto, o *RAD* pode ser uma excelente alternativa, se houver o entendimento dos riscos envolvidos ao optar por essa metodologia.

3 METODOLOGIA

Nesta seção, são apresentadas duas etapas que constituem a metodologia empregada para construção deste documento. Cada etapa descreve detalhadamente as fases do trabalho, divididas em metodologia científica e desenvolvimento de software.

3.1 Metodologia Científica

Este trabalho fundamenta-se em uma pesquisa qualitativa e possui aspectos de uma pesquisa exploratória, buscando compreender e apresentar uma nova perspectiva ou prática dentro do fenômeno estudado. Conforme descrito por Gil (2002) este tipo de pesquisas têm como objetivo proporcionar maior familiaridade com o problema, com vistas a torná-lo mais explícito ou a constituir hipóteses. Pode-se dizer que estas pesquisas têm como objetivo principal o aprimoramento de ideias ou a descoberta de intuições.

O uso de PWAs para esse tipo de aplicação ainda é pouco comum, o que reforça o caráter exploratório do estudo. Assim, além de propor uma solução funcional para o problema, esse estudo também busca contribuir para o entendimento desses tecnologias na prática.

Além disso, por se tratar de uma pesquisa que compreende o desenvolvimento de um software, objetiva a apresentação de algo diferente. Segundo Wazlawick (2009) este tipo de trabalho apresenta um método um pouco mais amadurecido, consiste na apresentação de uma forma diferente de resolver um problema. Como o desenvolvimento de controles usando PWA não é comum, por ser uma abordagem nova.

3.2 Desenvolvimento de Software

Por se tratar de uma aplicação de baixa complexidade para o desenvolvimento, optou-se por uma abordagem rápida de desenvolvimento. A ideia foi focar em velocidade e agilidade. Nesse sentido, foi adotada a metodologia de desenvolvimento conhecida como desenvolvimento rápido de aplicações (*Rapid Application Development – RAD)*, que segundo Stair e Reynolds (2015), RAD utiliza ferramentas, técnicas e metodologias projetadas, para acelerar o desenvolvimento de aplicativos.

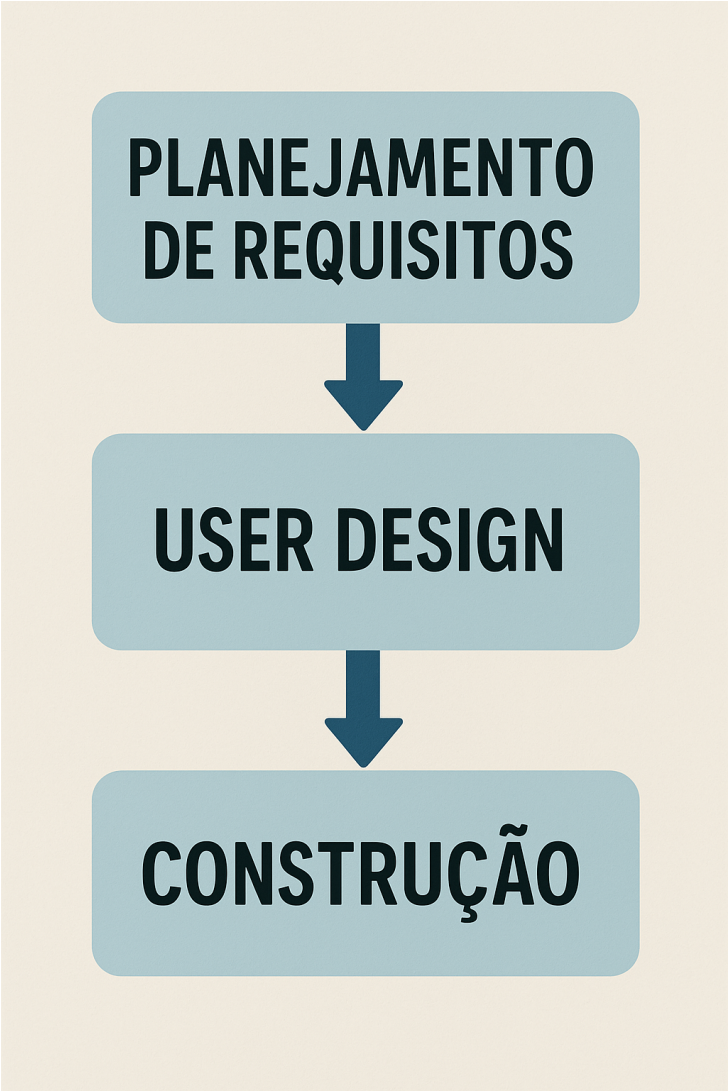
3.2.1 Aplicação do RAD no Projeto

A metodologia RAD foi aplicada por meio de três etapas principais: planejamento dos requisitos, design da interface e construção. Como o foco deste trabalho é a viabilidade da integração entre a PWA e o microcontrolador ESP32, optou-se por priorizar funcionalidades essenciais, como a interface de controle e a comunicação via *WebSocket*.

Durante o planejamento, foi definidos os requisitos básicos para o controle remoto de um carrinho, com foco na criação de comandos que pudessem ser enviados da PWA para o microcontrolador. A fase de design concentrou-se no desenvolvimento de uma interface simples

e leve, utilizando HTML e CSS puro. A construção da aplicação foi feita usando JavaScript para implementar a conexão com o ESP32 através do protocolo *WebSocket*. A Figura 8 ilustra de forma visual as etapas aplicadas no projeto segundo a abordagem RAD.

Figura 8 – Etapas aplicadas da metodologia *RAD* no desenvolvimento do projeto.



Fonte: Elaborado pelo autor (2025).

3.2.2 Potencial de Reaproveitamento da Solução

Embora o exemplo utilizado envolva a criação de uma PWA para o controle de um robô com rodas, a arquitetura proposta é genérica e pode ser aplicada a diversos cenários. A comunicação via *WebSocket* permite controle em tempo real de qualquer dispositivo que possua suporte a esse protocolo, como braços robóticos, sistemas de automação residencial, ou sensores industriais.

4 DESENVOLVIMENTO

Neste tópico serão apresentadas as ferramentas utilizadas para desenvolvimento e o passo a passo da criação da PWA até sua conexão com o ESP32.

4.1 Ferramentas utilizadas

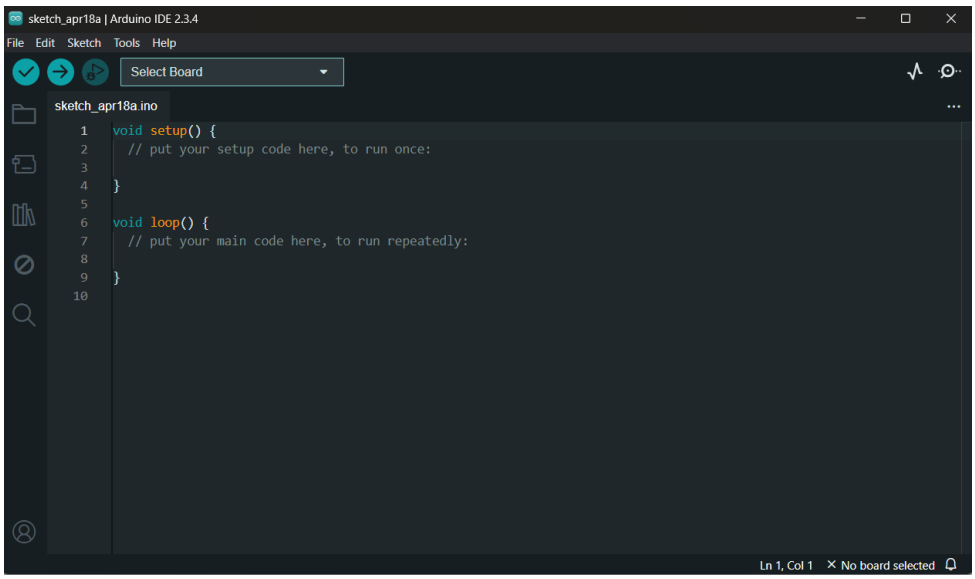
4.1.1 Visual Studio Code

O Visual Studio Code configura-se como um editor de texto e código-fonte de arquitetura leve e interface intuitiva, compatível com os sistemas operacionais Windows, macOS e Linux. Destaca-se pelo suporte nativo a diversas linguagens de programação, em especial ao JavaScript, que será utilizado no desenvolvimento da aplicação, em conjunto com as tecnologias HTML e CSS.

4.1.2 Arduino IDE

A plataforma Arduino utiliza uma *Integrated Development Environment* (IDE) própria. A IDE do Arduino é otimizada para simplificar o processo de programação e upload de código para placas de prototipagem, minimizando a complexidade encontrada em ambientes de desenvolvimento tradicionais. Como ilustrado na Figura 9, a interface inclui recursos essenciais como verificação de código, upload para a placa e monitoramento serial, tudo em um único ambiente integrado.

Figura 9 – Interface da IDE do Arduino



Fonte: Elaborado pelo autor (2025).

Para programar com a IDE do Arduino é recomendado conhecer linguagens como C/C++. Por ter uma sintaxe muito semelhante à de linguagens populares dentre os programadores, como

C/C++, escrita de forma linear e com linhas de código se utilizando de termos em inglês como, por exemplo, void setup() (Aquino et al., 2017).

O Arduino IDE permite a integração com o ESP32, sendo possibilitando escrever um código que funcione na placa do microcontrolador. O código escrito na IDE é compilado e transferido para o ESP32 através do cabo USB.

4.1.3 HTML, CSS e JavaScript

O desenvolvimento de uma PWA é possível por meio de algumas tecnologias fundamentais da web, como o HTML, CSS e JavaScript. Essas tecnologias, quando utilizadas em conjunto, formam a base da estrutura, aparência e comportamento da aplicação.

O *HyperText Markup Language* (HTML) é responsável por estruturar os elementos da página, como botões, textos, imagens e campos de formulário. Já o *Cascading Style Sheets* (CSS) cuida da parte visual, definindo cores, tamanhos, fontes e o layout geral. Por fim, o JavaScript adiciona interatividade e dinamismo, permitindo ações como clique em botões, animações, envio de dados e integração externa com o microcontrolador ESP32.

Essas três tecnologias juntas possibilitam a criação de uma PWA que é acessível via navegador e possui recursos avançados como funcionamento offline e instalação em dispositivos móveis.

4.2 Construção da PWA

A construção da PWA teve como foco principal permitir a comunicação com o ESP32 de forma simples, e para isso foi criado um arquivo chamando *manifest.json*, um arquivo que informa ao navegador os dados da aplicação, como nome, ícone e tema. Na Figura 10 pode-se ver a estrutura do arquivo.

Figura 10 – Arquivo *manifest.json*

Fonte: Elaborado pelo autor (2025).

Um outro arquivo muito importante na criação de uma PWA é o *service worker*. Esse arquivo contém scripts que rodam em segundo plano e permitem várias funcionalidades avançadas, como armazenar páginas e dados em cache, exibir notificações mesmo quando a PWA não está aberta, além de possibilitar que a aplicação funcione mesmo sem conexão com a internet.

Além disso, o *service worker* também ajuda a deixar o carregamento da aplicação mais rápido, pois evita que tudo precise ser baixado novamente toda vez que o usuário acessa. Ele é registrado no código JavaScript principal e começa a agir depois que a PWA é instalada ou visitada pela primeira vez.

A Figura 11 mostra como é a estrutura básica de um arquivo service worker.

Figura 11 – Arquivo *service-workers.js*

```
self.addEventListener("install", event => {
    event.waitUntil(
        caches.open("pwa-cache").then(cache => {
            return cache.addAll([
                "/",
                "/index.html",
                "/styles.css",
                "/manifest.json"
            ]);
        })
    );
});

self.addEventListener("fetch", event => {
    event.respondWith(
        caches.match(event.request).then(response => {
            return response || fetch(event.request);
        })
    );
});
```

Fonte: Elaborado pelo autor (2025).

Outro arquivo essencial na construção da PWA é o *index.html*, serve como ponto de partida da aplicação. É nele que toda a estrutura principal da página é montada. O HTML carrega o CSS para a parte visual, o manifesto da PWA e também o *service worker*, que é registrado automaticamente.

Além disso, dentro do próprio HTML, há um trecho de JavaScript responsável por chamar o arquivo JavaScript e abrir uma conexão *WebSocket* com o ESP32, usando o IP local do dispositivo. A partir dessa conexão, os botões são programados para enviar comandos simples, fazendo com que o microcontrolador possa se comunicar com a PWA de acordo com os cliques do usuário.

Figura 12 – Cabeçalho do arquivo *index.html*

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Controle do Carrinho</title>
    <link rel="stylesheet" href="styles.css">
    <link rel="manifest" href="manifest.json">
    <script>
        if ("serviceWorker" in navigator) {
            navigator.serviceWorker.register("sw.js");
        }
    </script>
</head>
```

Fonte: Elaborado pelo autor (2025).

A Figura 12 mostra como é feita a importação dos arquivos de estilo da página, arquivo *manifest* e carregamento do *service worker*. No *index.html* também é descrito o título da página, ele aparece na aba do navegador e nos resultados de pesquisa. No corpo do arquivo *index.html* é criado a estrutura do layout do controle de um carrinho, e no final é chamado o arquivo JavaScript para interagir com o HTML.

Na sequência, a Figura 13 apresenta com mais detalhes a estrutura da interface do controle do robô com rodas. Esse trecho do código HTML organiza os elementos visuais e interativos que permitem ao usuário enviar comandos ao sistema.

Figura 13 – Interface HTML para controle do carrinho

```
<body>
    <h2>Controle do Carrinho</h2>
    <button id="ligar" class="btn-start">Iniciar</button>
    <div class="bottom-controls">
        <div id="joystick-container"></div>
        <div class="speed-control">
            <label>Velocidade:</label>
            <input type="range" min="0" max="100" value="50" id="speed" oninput="sendSpeed()">
        </div>
        <div class="controls">
            <div></div>
            <button onclick="sendCommand('frente')">⬆️</button>
            <div></div>
            <button onclick="sendCommand('esquerda')">⬅️</button>
            <div></div>
            <button onclick="sendCommand('direita')">➡️</button>
            <div></div>
            <button onclick="sendCommand('re')">⬇️</button>
            <div></div>
        </div>
    </div>

    <script defer src="js/script.js"></script>
</body>
```

Fonte: Elaborado pelo autor (2025).

A Figura 13 mostra um trecho de código HTML responsável por estruturar a interface gráfica de controle do robô com rodas. O layout é composto por botões e controles que permitem ao usuário interagir diretamente com a aplicação.

O botão com o nome "Iniciar"abre conexão com a página PWA e o microcontrolador, enquanto os demais botões estão organizados para controlar a movimentação do carrinho: frente, ré, esquerda e direita, cada um acionando a função JavaScript *sendCommand()* com o parâmetro de direção. O controle de velocidade é representado por um *input* do tipo *range*, que varia de 0 a 100 e envia o valor atualizado ao sistema sempre que o usuário ajusta a barra, por meio da função *sendSpeed()*.

Além disso, é adicionado no código um arquivo externo de script (*script.js*) responsável pela lógica de funcionamento desses controles. Essa organização separa a lógica da estrutura visual.

O arquivo JavaScript contém métodos responsáveis pela manipulação do microcontrolador a Figura 14 apresenta esses métodos e como a conexão com o microcontrolador é instanciada.

Figura 14 – Arquivo JavaScript - Métodos de controle

```
const socket = new WebSocket("ws://192.168.1.10:81");

function sendCommand(command) {

    const podeEnviar = ["frente", "esquerda", "direita", "re"].includes(command);
    if (podeEnviar) {
        socket.send(command);
    }
}

function sendSpeed() {
    let speed = document.getElementById('speed').value;
    socket.send(speed);
}
```

Fonte: Elaborado pelo autor (2025).

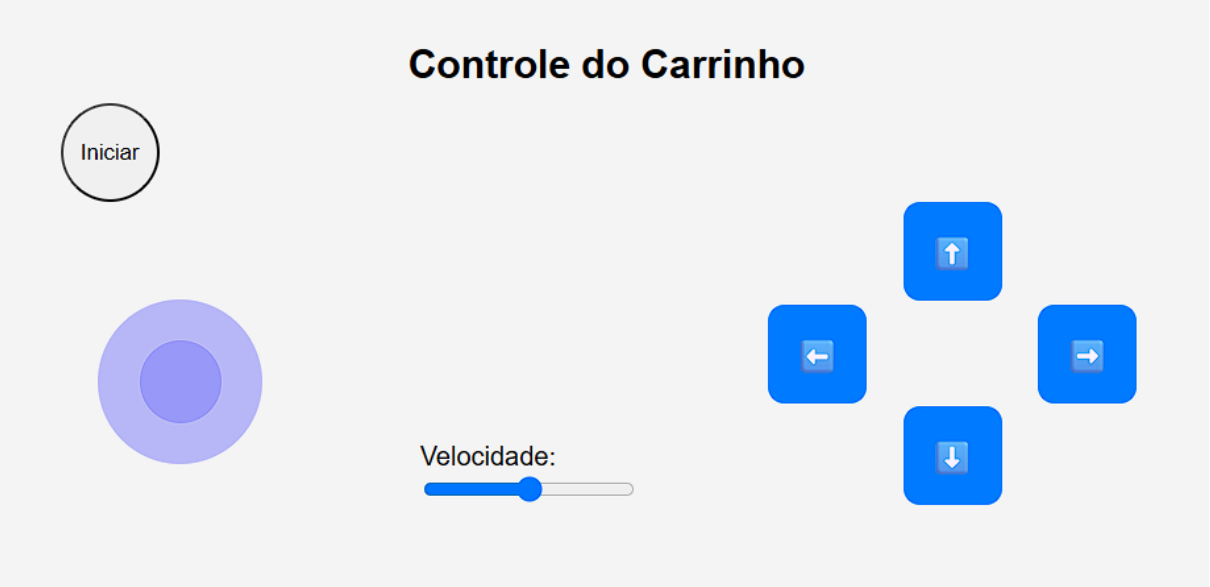
Observa-se na Figura 14 o trecho de código JavaScript responsável pela comunicação entre a PWA e o microcontrolador ESP32 por meio do protocolo *WebSocket*. Inicialmente, é estabelecida uma conexão com o servidor, localizado no endereço IP 192.168.1.10 na porta 81, através da instância *WebSocket*.

Em seguida, define-se a função *sendCommand(command)*, que verifica se o comando recebido está entre os valores permitidos ("frente", "esquerda", "direita"e "re") e, caso seja válido, o envia para o servidor utilizando o método *send()* da conexão *WebSocket*. Também é definida a função *sendSpeed()*, que recupera o valor atual de um controle deslizante de velocidade presente na interface (identificado pelo *ID speed*) e o transmite ao servidor. Com isso, o código possibilita

o controle remoto do carrinho, enviando tanto comandos de direção quanto ajustes de velocidade em tempo real.

Para que os comandos definidos no código JavaScript possam ser entendidos pelo usuário, foi desenvolvido um layout visual simples e funcional. Esse layout foi estruturado no arquivo *index.html*. A Figura 15 apresenta a organização desses elementos na interface, incluindo botões direcionais, controle de velocidade e o botão de inicialização.

Figura 15 – Layout do controle



Fonte: Elaborado pelo autor (2025).

O microcontrolador ESP32 pode receber comando via *WebSocket* e com isso controlar o carrinho de forma remota. Pra isso é necessário o desenvolvimento do código escrito na Arduino IDE, utilizando a linguagem de programação C++, é possível estabelecer a comunicação com a rede Wi-Fi e iniciar o servidor *WebSocket*. A seguir, apresenta-se o trecho do código implementado no ESP32 que viabiliza essa funcionalidade.

Figura 16 – Trecho do código para controle do ESP32 via *WebSocket*

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <WebSocketsServer.h>

const char* ssid = "SSID";
const char* password = "PASSWORD";

AsyncWebServer server(80);
WebSocketsServer webSocket(81);

void handleWebSocketMessage(uint8_t num, uint8_t *payload, size_t length) {
  String message = (char*)payload;

  if (message == "frente") {
    setDirecao("frente");
  } else if (message == "esquerda") {
    setDirecao("esquerda");
  } else if (message == "direita") {
    setDirecao("direita");
  } else if (message == "re") {
    setDirecao("re");
  }
}

void onWebSocketEvent(uint8_t num, WStype_t type, uint8_t *payload, size_t length) {
  if (type == WStype_TEXT) {
    handleWebSocketMessage(num, payload, length);
  }
}
```

Fonte: Elaborado pelo autor (2025).

A Figura 16 exibe parte do código desenvolvido na IDE do Arduino. Inicialmente, é feita as importações das bibliotecas necessárias para a comunicação da PWA com o ESP32 por meio do protocolo *WebSocket*. Em seguida, são definidas as credenciais da rede Wi-Fi, SSID e senha, que o ESP32 deve utilizar para se conectar. Logo após, é instanciado o servidor HTTP na porta 80 e o servidor *WebSocket* na porta 81, que será responsável por receber os comandos enviados pela interface web.

O método *handleWebSocketMessage()* é sempre chamada quando uma nova mensagem for recebida pelo *WebSocket*. Ela converte o conteúdo da mensagem recebida para o tipo *String*, e então verifica se o valor corresponde a algum dos comandos válidos, como "frente", "esquerda", "direita"ou "re". Conforme o comando recebido, a função *setDirecao()* é chamada passando a direção correta como parâmetro.

O método *onWebSocketEvent()* fica responsável por realizar o filtro dos tipos de mensagens que chegam. É chamada uma função de tratamento apenas se a mensagem recebido for do tipo "texto", isso evita que aconteça processamento de dados irrelevantes.

Para utilizar os métodos criado na Figura 16, é preciso utilizar uma função de inicialização. A Figura 17 ilustra a função de inicialização utilizada.

Figura 17 – Trecho de código de inicialização do ESP32 com conexão Wi-Fi e *WebSocket*

```
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando ao Wi-Fi...");
  }

  Serial.println(WiFi.localIP());

  Serial.println("Wi-Fi conectado!");

  server.begin();
  webSocket.begin();
  webSocket.onEvent(onWebSocketEvent);
}

void loop() {
  webSocket.loop();
}
```

Fonte: Elaborado pelo autor (2025).

A função *setup()* é executada uma única vez quando o dispositivo é iniciado. Nessa função é estabelecida a comunicação com o computador através da função *Serial.begin(115200)*. É iniciada posteriormente a conexão Wi-Fi com a rede local utilizando os parâmetros de nome (SSID) e senha.

Enquanto a conexão Wi-Fi não é estabelecida, o código entra em um laço de espera de 1 segundo e realizando a impressão da mensagem "Conectando ao Wi-Fi...". Uma vez conectada à rede, o endereço IP local é atribuído ao ESP32, permitindo assim que o dispositivo possa ser acessado.

Após a conexão ser estabelecida o servidor local é inicializado por meio da função *server.begin()*. Em seguida, a comunicação em tempo real com o *WebSocket* é configurada com o *webSocket.begin()*, permitindo que o ESP32 aceite conexões de clientes. Por fim, a função *webSocket.onEvent(onWebSocketEvent)* define um manipulador de eventos que será executado sempre que ocorrerem interações no canal *WebSocket*, como o recebimento de mensagens ou o estabelecimento de novas conexões. A função *loop()*, mantém o canal *WebSocket* ativo por meio de *webSocket.loop()*, permitindo a comunicação em tempo real com a aplicação web e o envio imediato de comandos da PWA.

5 CONCLUSÃO

Este trabalho buscou explorar de forma prática como é feita a integração de uma aplicação web progressiva (PWA) com o microcontrolador ESP32, utilizando a comunicação via *WebSocket* para o controle do ESP32. Vale ressaltar que, não foi implementado o código responsável pela movimentação do carrinho. O escopo do desenvolvimento limitou-se à construção da interface PWA e ao envio de comandos para o ESP32 via *WebSocket*. A proposta permitiu entender aspectos técnicos e sua aplicabilidade em cenários reais do dia a dia.

Embora o projeto tenha sido aplicado de maneira simples em uma estrutura para integrar com um carrinho, a ideia central vai muito além disso. A abordagem utilizada pode ser facilmente adaptada e aplicada há diferentes contextos como sistemas de automação residencial, monitoramento remoto, acionamento de dispositivos remotos, entre outros. A comunicação entre a PWA e o ESP32 apresenta uma ampla gama de soluções eficientes e inteligentes.

O desenvolvimento apresentado no trabalho reforça a importância de unir o desenvolvimento web com o mundo do controle eletrônico com dispositivos físicos. Essa conexão permite criar soluções mais interativas, modernas e acessíveis, e mostra como a tecnologia pode ser usada de forma eficiente para resolver problemas do nosso cotidiano.

Como sugestão para trabalhos futuros, recomenda-se a implementação completa do controle do robô com rodas, incluindo o código para seu controle de movimentação. Além disso, podem ser exploradas melhorias na interface da PWA e aplicações em diferentes contextos.

REFERÊNCIAS

Aquino, L. M. et al. **Proposta de um curso semipresencial de robótica educacional utilizando a plataforma Arduino**. Revista Principia, n. 34, p. 48–54, 2017. Acesso em: 20 mar. 2025. Disponível em: <https://repositorio.ufc.br/bitstream/riufc/62495/1/2017_art_lmaquino.pdf>. Citado na página 29.

ARAÚJO, L. de O.; TANAKA, S. S. **Metodologias de desenvolvimento de software: uma revisão bibliográfica das metodologias de desenvolvimento de software, demonstrando sua aplicabilidade e recomendações**. Revista Terra & Cultura: Cadernos de Ensino e Pesquisa, v. 40, n. especial, p. 22–45, 2024. Acesso em: 01 mai. 2025. Disponível em: <http://periodicos.unifil.br/index.php/Revistateste/article/view/3152>. Citado na página 25.

CAN I USE. Can I use Service Workers. 2025. Acesso em: 19 abr. 2025. Disponível em: <https://caniuse.com/serviceworkers>. Citado na página 22.

Farias, E. M. B.; Vasconcelos, F. H. **Protocolo Websocket em Hardware Restrito para Redes de Sensores sem Fio**. 2019. Acesso em: 21 abr. 2025. Disponível em: <https://www.researchgate.net/profile/Elany-Marinho-Branches-Farias/publication/337720805_Protocolo_Websocket_em_Hardware_Restrito_para_Redes_de_Sensores_sem_Fio/links/5de6e878a6fdcc2837034d3e/Protocolo-Websocket-em-Hardware-Restrito-para-Redes-de-Sensores-sem-Fio.pdf>. Citado 2 vezes nas páginas 23 e 24.

Gil, A. C. **Como elaborar projetos de pesquisa**. [S.l.]: Atlas São Paulo, 2002. v. 4. Citado na página 26.

Hume, D. **Progressive web apps**. [S.l.]: Simon and Schuster, 2017. Citado 2 vezes nas páginas 19 e 20.

Ibrahim, D. **The Complete ESP32 Projects Guide**. [S.l.]: Elektor Digital, 2017. Citado 2 vezes nas páginas 17 e 18.

Kolban, N. **Kolban's book on ESP32**. [S.l.]: Leanpub, 2018. Citado 2 vezes nas páginas 16 e 17.

LePage, P.; Richard, S. **O que são Progressive Web Apps?** 2020. Acesso em: 30 mar. 2025. Disponível em: <https://web.dev/articles/what-are-pwas?hl=pt-br>. Citado na página 20.

Mazzarolo, V. et al. **Progressive Web Apps uma nova abordagem no desenvolvimento de aplicações Web**. 2021. Acesso em: 30 mar. 2025. Disponível em: <https://repositorio.ifrs.edu.br/bitstream/handle/123456789/1275/1234567891275.pdf>. Citado na página 21.

Miorandi, D. et al. **Internet of things: Vision, applications and research challenges**. Ad hoc networks, Elsevier, v. 10, n. 7, p. 1497–1516, 2012. Acesso em: 13 mar. 2025. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S1570870512000674>. Citado na página 15.

Pontes, G. **Progressive Web Apps: Construa aplicações progressivas com React**. [S.l.]: Editora Casa do Código, 2018. Citado 3 vezes nas páginas 19, 21 e 23.

SANTOS, B. P. et al. **Internet das coisas: da teoria à prática**. Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos, v. 31, p. 16, 2016. Acesso em: 10 mar. 2025. Disponível em: <https://homepages.dcc.ufmg.br/~mmvieira/cc/papers/internet-das-coisas.pdf>. Citado 2 vezes nas páginas 15 e 16.

Shelly, G. B.; Rosenblatt, H. J. **System Analysis and Design, 9th Editio**. [S.l.]: Cengage Learning, 2012. Citado na página 25.

Stair, R. M.; Reynolds, G. W. **Princípios de sistemas de informação**. São Paulo: Cengage Learning, 2015. Citado na página 26.

Trindade, P. E.; Affini, L. P. **Apontamento a cerca do Progressive Web Apps**. In: III Jornada Internacional GEMInIS (JIG 2018) - São Paulo-SP. [s.n.], 2019. Acesso em: 08 set. 2024. Disponível em: <https://doity.com.br/anais/jig2018/trabalho/81964>. Citado 2 vezes nas páginas 20 e 21.

Vasudevan, S.; Wilemon, D. **Rapid application development: major issues and lessons learned**. In: IEEE. Innovation in Technology Management. The Key to Global Leadership. PICMET'97. [S.l.], 1997. p. 484. Citado na página 24.

Wang, V.; Salim, F.; Moskovits, P. **The definitive guide to HTML5 WebSocket**. [S.l.]: Springer, 2013. v. 1. Citado na página 23.

Wargo, J. M. **Learning progressive web apps**. [S.l.]: Addison-Wesley Professional, 2020. Citado 2 vezes nas páginas 21 e 22.

Wazlawick, R. S. **Engenharia de software: conceitos e práticas**. Rio de Janeiro: Elsevier, 2009. Citado na página 26.

Zemel, T. **Web Design Responsivo: páginas adaptáveis para todos os dispositivos**. [S.l.]: Editora Casa do Código, 2015. Citado na página 19.