UNIVERSIDADE ESTADUAL DO MARANHÃO
CENTRO DE CIÊNCIAS TECNOLÓGICAS
CURSO DE ENGENHARIA DE COMPUTAÇÃO

SISTEMA DE AUTOMAÇÃO PARA PRODUÇÃO DE CERVEJAS ARTESANAIS

MATHEUS PEREIRA DA SILVA

SÃO LUÍS - MA
2021

UNIVERSIDADE ESTADUAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE ENGENHARIA DA COMPUTAÇÃO

MATHEUS PEREIRA DA SILVA

SISTEMA DE AUTOMAÇÃO PARA PRODUÇÃO DE CERVEJAS ARTESANAIS

Trabalho apresentado ao curso de Graduação em Engenharia da Computação na Universidade Estadual do Maranhão, como pré-requisito necessário para obtenção do título de Bacharel em Engenharia da Computação

Orientador: Prof.(a) Elton de Sousa e Silva.

SÃO LUÍS - MA
2021

Silva, Matheus Pereira da.

Sistema de automação para produção de cervejas artesanais / Matheus Pereira da Silva. – São Luís, 2021.

57 f

Monografia (Graduação) – Curso de Engenharia de Computação, Universidade Estadual do Maranhão, 2021.

Orientador: Prof. Elton de Sousa e Silva.

1.Cerveja artesanal. 2.Sistema automatizado. 3.Esp8266. I.Título.

Matheus Pereira da Silva

SISTEMA DE AUTOMAÇÃO PARA PRODUÇÃO DE CERVEJAS ARTESANAIS

Trabalho apresentado ao curso de Graduação em Engenharia da Computação na Universidade Estadual do Maranhão, como pré-requisito necessário para obtenção do título de Bacharel em Engenharia da Computação

Aprovado em: 05/04/2021

BANCA EXAMINADORA

Elton de Sousa e Silva

Prof. Elton de Sousa e Silva (Orientador)
Universidade Estadual do Maranhão

Pedro Brandão nto

Prof. Me. Pedro Brandão Neto
Mestre em Engenharia de Eletricidade
Universidade Estadual do Maranhão

Lacampos

Prof. Dr. Lúcio Flávio de Albuquerque Campos
Doutor em Biotecnologia
Universidade Estadual do Maranhão

AGRADECIMENTOS

Agradeço primeiramente a Deus, que me deu forças para enfrentar as dificuldades durante a graduação e iluminou o meu caminho e deu a oportunidade de poder defender minha monografia.

Agradeço também aos meus pais, Lucas Vieira da Silva Filho e Ilzenir Sousa Pereira da Silva que sempre incentivaram meus estudos e me propiciaram toda estrutura necessária para que eu pudesse realizar minha graduação da melhor forma possível.

Agradeço aos meus irmãos Lucas Victor Pereira da Silva e Lorena Victoria Pereira da Silva, que estiveram presentes durante o período do desenvolvimento do trabalho, me motivando e servindo como exemplos por terem concluído esta etapa enquanto eu ainda iniciava minha graduação.

Agradeço aos meus amigos que estiveram presentes em todo o meu período de graduação, me incentivando e propiciando momentos de descontração para poder passar pelos momentos difíceis com mais leveza.

Agradeço a todos professores, que passaram seu conhecimento técnico e ensinamentos extra classe, possibilitando a minha formação não só como bacharel, mas em todas etapas em que eu passei, desde o jardim de infância até hoje.

Agradeço a minha namorada Glenda Rebeca Reis França, que sempre acreditou em mim e me deu todo apoio para que eu persistisse em minha formação.

Agradeço também a todos aqueles que contribuíram direta ou indiretamente na minha formação, seja com uma palavra amiga ou a simples presença em minha vida.

Agradeço àqueles que já não se fazem presentes hoje entre nós, mas que contribuíram com seus ensinamentos e se fazem presentes por meio da memória que propiciaram.

RESUMO

De acordo com dados de 2019 do Ministério da Agricultura, Pecuária e Abastecimento (MAPA), nos últimos 10 anos o número de cervejarias artesanais no Brasil obteve um crescimento superior a 1000%. Este mercado já movimenta cerca de 2,4 bilhões de reais. Tendo em vista o cenário de grande expansão deste nicho, surge a necessidade da criação de sistemas que automatizem a produção, reduzindo custos e aumentando a eficiência da produção. Este trabalho propõe a criação de um sistema automatizado de baixo custo que otimize o processo de produção de cerveja artesanal com foco nos cervejeiros caseiros. O protótipo é composto pela placa Wemos d1 mini que possui o microcontrolador esp8266, um sensor de temperatura, relés, e outros componentes eletrônicos responsáveis por controlar uma resistência elétrica responsável por aquecer o mosto no processo de brassagem. O sistema possui um servidor integrado que hospeda uma interface web responsável por gerenciar o processo de brassagem, podendo ser acessado por celular, computador ou qualquer dispositivo que possui um navegador.

Palavras-chave: Cerveja artesanal; Sistema automatizado; esp8266.

ABSTRACT

According to 2019 data from the Ministry of Agriculture, Livestock and Supply (Mapa), in the last 10 years the number of artisanal breweries in Brazil has grown over 1000%. This market already moves about 2.4 billion reais. In view of the scenario of great expansion of this niche, there is a need to create systems that automate production, reducing costs and increasing production efficiency. This work proposes the creation of a low cost automated system that optimizes the craft beer production process with a focus on home brewers. The prototype is composed by the Wemos d1 mini plate that has the esp8266 microcontroller, a temperature sensor, relays, and other electronic components responsible for controlling an electrical resistance responsible for heating the wort in the brazing process. The system has an integrated server that hosts a web interface responsible for managing the brazing process, which can be accessed by cell phone, computer or any device that has a browser.

Keywords: Craft beer; Automated system; esp8266.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BIAB	*Brew In A Bag*
MAPA	Ministério da Agricultura, Pecuária e Abastecimento
IDE	Integrated Development Environment
MQTT	*Message Queuing Telemetry Transport*
IOT	*Internet of Things*

SUMÁRIO

1. INTRODUÇÃO

Os sistemas automatizados permitem que tarefas sejam realizadas com expressivo aumento de produtividade. Tarefas que outrora necessitavam de um trabalhador atuando ativamente, podem ser realizados por um sistema automatizado com maior eficiência.

O processo de automatização já é utilizado a muito tempo em grandes indústrias cervejeiras em todo mundo, porém nas últimas décadas tornou-se cada vez mais comum a produção de cervejas artesanais, seja para consumo próprio ou para comercialização. Em decorrência do aumento da produção de cervejas artesanais, tornou-se necessário a criação de sistemas que aumentassem a eficiência do processo, mas ao mesmo tempo tivessem baixo custo.

O processo de produção da cerveja artesanal é composto por diversas etapas que exigem do produtor bastante atenção para que os ingredientes sejam misturados corretamente, no momento certo e na temperatura adequada, do contrário a cada brassagem a cerveja terá características diferentes, prejudicando a padronização da mesma.

O objetivo deste trabalho é desenvolver um sistema de automação de baixo custo voltado para produção de cerveja artesanal caseira (*homebrew*). Neste tipo de produção são utilizadas de uma a três panelas aquecidas por fogareiro a gás ou resistência elétrica.

1.1. Objetivos

1.1.1. Objetivo Geral

Melhorar a eficiência da produção de cervejas artesanais por meio da criação de um sistema de automação.

1.1.2. Objetivos Específicos

Desenvolver um sistema composto de hardware e software capaz de automatizar, monitorar e gerenciar o processo de produção de cerveja artesanal em micro cervejarias.

- Montar um dispositivo que automatize a produção de cerveja artesanal.
- Desenvolver um software que gerencie a produção.
- Desenvolver um relatório detalhado sobre o funcionamento do sistema de automação para produção de cerveja artesanal.

1.2. Metodologia

A pesquisa documental foi realizada com base em livros, revistas, sites e teses que abordem tanto o tema de automação e controle quanto o processo de produção cervejeira artesanal e também sobre desenvolvimento de software. Como recursos técnicos foram utilizados componentes eletrônicos, microcontroladores, computador, celular e equipamentos voltados para a produção de cerveja artesanal, como: panelas cervejeiras, resistência elétrica, sensores de temperatura, etc. Foram também utilizados softwares como Visual Studio Code, Platformio e Arduino IDE. As linguagens de programação utilizadas serão C++ para o desenvolvimento no esp8266 e Javascript em conjunto com framework Vuejs para desenvolvimento da interface do sistema.

1.3. Justificativa

A motivação para a realização desse trabalho, veio do estudo do processo de produção de cerveja artesanal, que apesar de ser realizado em casa por diversas pessoas, é um processo manual e bastante trabalhoso. Logo, existe a necessidade de sistemas de baixo custo que auxiliem pequenos cervejeiros a melhorar a eficiência e padronização da brassagem.

1.4. Aplicabilidade

O sistema de automação desenvolvido pode ser utilizado em micro cervejarias caseiras, sendo inicialmente voltado para o método Single *Vessel*, onde todo processo é realizado apenas em uma panela. O método de filtragem do mosto pode ser *Brew in a Bag ou BIAB* (Método em que brassagem é feita utilizando apenas uma panela e um saco de grãos), bazzoka (O qual utiliza uma pequena malha metálica em um furo na parte inferior da panela) ou fundo falso (Que utiliza um fundo falso de alumínio ou inox com pequenos furos que permitem a passagem do líquido).

1.5. Estrutura do documento

Este relatório está estruturado da forma como segue. No Capítulo 2 é apresentada a fundamentação teórica. O Capítulo 3 apresenta a desenvolvimento do trabalho. Finalmente no Capítulo 4 são apresentadas as conclusões e os trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. Automação

"Um sistema de controle consiste em subsistemas e processos (ou plantas) construídos com o objetivo de se obter uma saída desejada com um desempenho desejado, dada uma entrada especificada" [1]. No caso do sistema de controle aqui apresentado o foco será em um processo da produção de cerveja artesanal, a etapa de brassagem.

> Os sistemas de controle possibilitaram diversos avanços em processos industriais, ao melhorar produtividade, precisão e realização de tarefas repetitivas. Uma tendência cada vez mais presente nas empresas que buscam o sucesso é automatizar as diversas atividades desenvolvidas. A automação melhora o dinamismo dos serviços oferecidos, reduzindo filas, tempo de espera, agilizando a emissão de notas fiscais, entre outros. Os fornecedores de equipamentos para micro cervejarias vêm se aprimorando a cada dia, e com isto, os processos produtivos estão sendo integrados e controlados por meio de sensores e equipamentos eletrônicos, de forma que, praticamente, todo o processo produtivo está automatizado. Existem softwares, por exemplo, que fazem o controle da temperatura da adega e da fermentação da cerveja, controle da temperatura-pressão para o processo de fermentação, etc. (Sebrae, 202-)

"Os grandes grupos conseguem isenções de impostos, como o ICMS, que as pequenas não conseguem, além de ter automação e economizar com a produção em escala", afirma Carlo Bressiani, diretor-geral da Escola Superior de Cerveja e Malte (ESCM). [2]

Tendo em vista este cenário, pequenos produtores buscam cada vez mais alternativas para diminuir os custos de produção. Como não conseguem isenções fiscais, o uso de tecnologias de automação de baixos custos podem mitigar esta discrepância nos custos de produção, ao reduzir a necessidade de empregar capital humano em tarefas com facilmente realizáveis por sistemas automatizados.

Uma tecnologia que vêm ganhando relevância nos últimos anos, por conta dos seus inúmeros benefícios é a internet das coisas. Segundo a RedHat, "A IoT se refere a qualquer sistema de dispositivos físicos que recebem e transferem dados em redes sem fio e sem intervenção humana. Isso é possível ao integrar dispositivos de computação simples com sensores em todos os tipos de objetos."

Sua utilização em pequenas cervejarias pode melhorar ainda mais o processo de automação ao monitorar e analisar o processo de produção para garantir que estejam funcionando dentro das tolerâncias exigidas

2.2. Cerveja

2.2.1. História

Embora a cerveja, como é reconhecida nos dias modernos, tenha sido desenvolvida na Alemanha, a bebida fermentada tem suas origens na antiga Mesopotâmia.

O início da produção de cerveja coincide com a época em que o ser humano pré-histórico abandona a vida de nômade e desenvolve a agricultura. Com o início do cultivo de cereais, como trigo e cevada a cerca de 12000 anos na Ásia. A tese mais aceita sobre o processo de fabricação de cerveja é que tenha sido descoberta por acaso, através de cereais que sofreram com a umidade e logo após foram secos, interrompendo o processo de germinação e, logo após, sofreram, fermentação espontânea por fungos presentes na atmosfera.

A partir da mesopotâmia a cerveja teve diversas rotas de difusão, uma delas levou aos povos celtas e germanos, tornando esta bebida amplamente produzida e consumida nestas regiões, tornando-se tradicional na Alemanha e Bélgica.

> “Os alemães estavam fabricando cerveja (que eles chamavam de ol, para "cerveja") já em 800 AC, como é conhecido por grandes quantidades de jarros de cerveja, ainda contendo evidências da cerveja, em uma tumba na vila de Kasendorf no norte da Baviera, perto de Kulmbach. Que a prática continuou na era cristã é evidenciado por outros achados arqueológicos e registros escritos. No início, como na Mesopotâmia e no Egito, o ofício do cervejeiro era proveniente de mulheres e os Hausfrau fabricavam sua cerveja em casa para complementar as refeições diárias. [...] Com o tempo, porém, o ofício foi assumido por monges cristãos, principalmente, e a fabricação de cerveja tornou-se parte integrante da vida monástica. O Kulmbacher Monchshof Kloster, um mosteiro fundado em 1349 dC em Kulmbach, ainda hoje produz sua famosa Schwartzbier, entre outras cervejas. Em 1516 dC, a Reinheitsgebot (lei de pureza) alemã foi instituída, regulamentando os ingredientes que poderiam ser legalmente usados na fabricação de cerveja (apenas água, cevada, lúpulo e, posteriormente, fermento) e, ao fazê-lo, continuou a prática da legislação relativa à cerveja o que os babilônios sob Hammurabi haviam feito cerca de três mil anos antes. Os alemães, como os

> que os precederam, também instituíram uma ração diária de cerveja e consideravam a cerveja um alimento básico necessário de sua dieta." (Mark, 2011)

2.2.2. Matérias-primas

Promulgada em 23 de abril de 1516, pelo então Duque da Baviera Guilherme IV a lei da pureza alemã (Reinheitsgebot) permitia a utilização de apenas três ingredientes, são eles: água, malte de cevada e lúpulo. Apenas no século XX, após a descoberta da fermentação por Louis Pasteur, o fermento (leveduras) foi introduzido à lei.

Além dos ingredientes já citados, são utilizados outros tipos de grãos além da cevada como: trigo, arroz, milho, centeio, aveia e sorgo. Outros produtos, como açúcar, frutas e especiarias, são utilizadas em cervejas que não seguem a lei da pureza alemã.

> “Somente Bebidas alcoólicas feitas a partir do açúcar de grãos que contenham pelo menos 20% de malte de cevada podem ser consideradas cerveja. Bebidas resultantes de fermentação de açúcares não originários de grãos, ou até mesmo que usem cereais, mas não utilizem a quantidade mínima de malte, não podem ser chamadas de cerveja, ainda que apliquem o mesmo processo bioquímico. Assim o mel fermentado é chamado de hidromel, o suco de maçã ou pera é chamado de sidra o suco de uva fermentado resulta em vinho e a fermentação do arroz dá origem ao saquê japonês e ao jiu chinês.” (MORADO, 2018)

2.2.3. Etapas da produção de cerveja artesanal

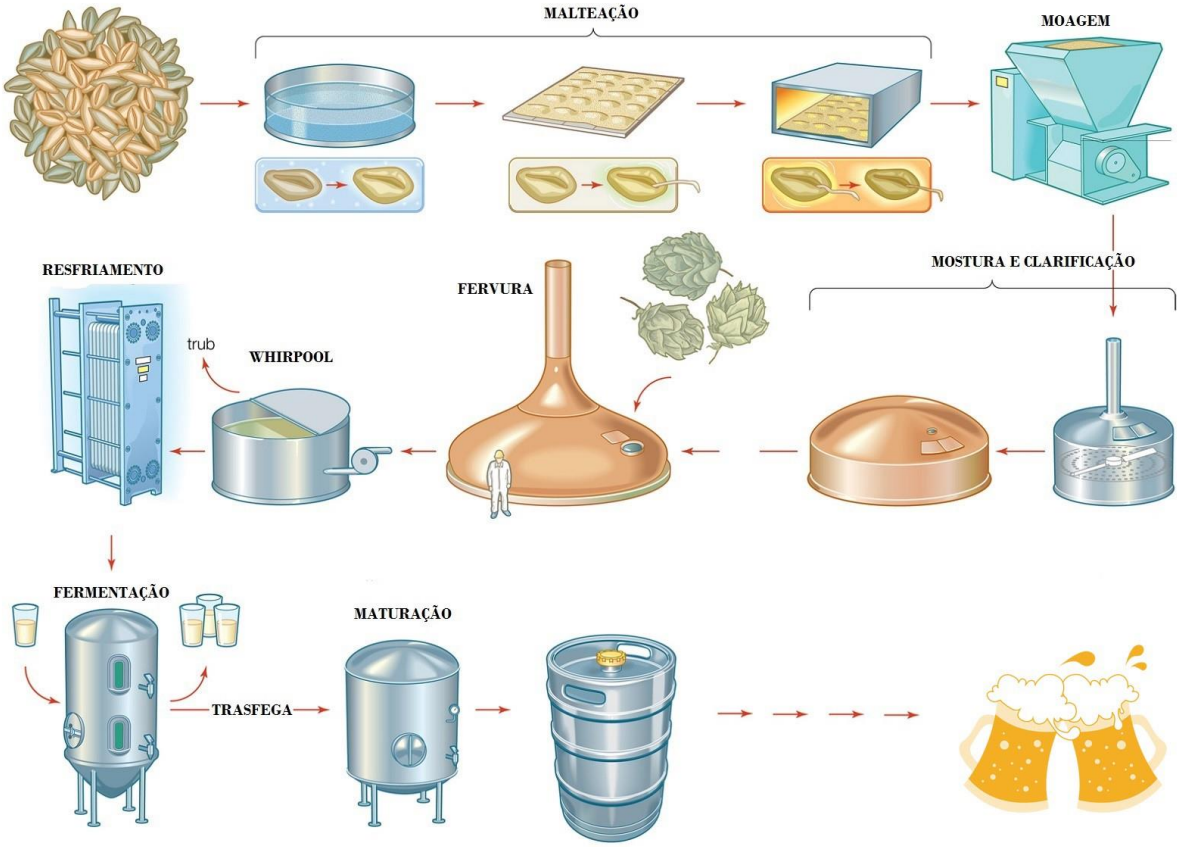
A produção de cerveja artesanal caseira é dividida em diversas etapas, sendo as principais: maltagem, moagem dos grãos, brassagem, fervura, fermentação, maturação e envase. A figura 1 representa um diagrama da produção de cerveja artesanal.

Maltagem: É o processo que transforma um cereal em malte. Esse processo ocorre quando se inicia o processo de germinação do grão e logo após interrompe-o processo através de secagem e/ou torra do mesmo. Este processo é responsável por fazer com que as sementes produzam enzimas que vão quebrar o amido do malte em açúcares menores. Dependendo do grau de torra deste grão o malte recebe sua classificação.

Moagem: Durante a etapa da moagem os grãos de cevada devem ser quebrados para expor o endosperma e assim melhorar a ação das enzimas sobre o amido contido neles. É

importante que o nível da moagem seja controlado para que se tenha o resultado esperado. Manter a casca da cevada auxilia na filtragem enquanto uma moagem muito fina pode prejudicar o processo de filtragem ao causar entupimentos.

Figura 1. Diagrama do processo de produção de cerveja artesanal



Fonte: Cinema e Cerveja, 2017

Mostura: Esse processo é responsável pela conversão do amido presente no malte em açúcares menores. Durante esse processo o malte é cozido em água em diversas rampas de temperatura. No contexto dos cervejeiros caseiros, os sistemas normalmente são compostos de uma a três panelas. Nos sistemas com 3 panelas(*three-vessel*), cada uma das panelas tem uma função específica, sendo uma para mostura, a outra para a fervura e a terceira para aquecer a água de lavagem. Já nos sistemas que possuem apenas uma panela *(single-vessel)* a mesma panela realiza todas as funções, reduzindo o espaço necessário.

Filtragem: Após o processo de mostura o malte deve ser filtrado do líquido (mosto). Em relação ao método de filtragem do mosto, existem diversas opções, tais como *Brewing in a Bag* (BIAB), bazooka ou fundo falso.

Fervura: A fervura acontece quando o mosto entra em ebulição, e tem como função esterilizar o mosto, aumentar a densidade e eliminar substâncias indesejadas.

Resfriamento: Este processo é responsável por resfriar o mosto até a temperatura ideal para inocular o fermento, pois as leveduras são micro-organismos sensíveis a altas temperaturas.

Fermentação: No processo de fermentação as leveduras convertem os açúcares provenientes do grão maltado em álcool e gás carbônico. Durante esta etapa a temperatura deve ser controlada para garantir a conversão. Dura em média duas a três semanas.

Maturação: A maturação é uma segunda fermentação da cerveja que tem como objetivo eliminar sabores indesejados, clarificar a cerveja e melhorar características sensoriais em geral. Dura em média 10 dias.

Envase: O envase é a etapa em que a cerveja está pronta para consumo e é armazenada para distribuição, podendo ser realizado em garrafas, barris e latas de acordo com a logística do cervejeiro.

2.3. Trabalhos Correlatos

Nesta seção serão apresentados alguns trabalhos correlatos que fazem uso da automatização para melhorar o processo de produção de cerveja artesanal.

2.3.1. Desenvolvimento de máquina semi-automatizada de fabricação de cerveja artesanal

O TCC da Natália Santos Mendes orientado pelo Professor Dr. Diogo Caetano Garcia intitulado “Desenvolvimento de máquina semi-automatizada de fabricação de cerveja artesanal”, submetido ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília em 2018 propõe a criação de uma máquina semiautomatizada de fabricação de cerveja artesanal.

O protótipo, nomeado por MakeBeer, é composto por estrutura em aço 1020 que comporta duas panelas monitoradas por sensores de temperatura infravermelho (sistema *double vessel*). Este projeto possui compartimentos exclusivos para os lúpulos controlado por motores DC automaticamente, utiliza uma peneira como elemento de filtragem do mosto e

possui mini-bombas para realizar a troca de líquido entre as duas panelas. A interface com o usuário será uma tela LCD, com botões para serem escolhidas as opções disponíveis. O controle deste sistema é feito pelo microcomputador Raspberry Pi.

3. DESENVOLVIMENTO

3.1. Proposta do Sistema

A proposta do sistema é controlar e automatizar a produção da cerveja nas etapas de brassagem e fervura. Durante estas etapas, que compõem a fase quente do processo cervejeiro o sistema será responsável por acionar a fonte de calor na temperatura e tempo especificados pela receita.

> "Os fornecedores de equipamentos para microcervejarias vêm se aprimorando a cada dia, e com isto, os processos produtivos estão sendo integrados e controlados por meio de sensores e equipamentos eletrônicos, de forma que, praticamente, todo o processo produtivo está automatizado. Existem softwares, por exemplo, que fazem o controle da temperatura da adega e da fermentação da cerveja, controle da temperatura-pressão para o processo de fermentação, etc." (Sebrae,201-)

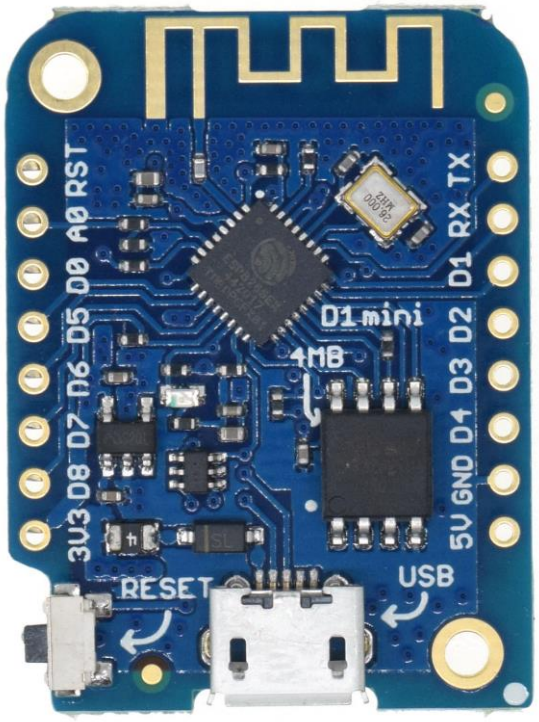
O sistema nomeado de EspBrew é composto por um equipamento eletrônico com software embarcado montado em uma caixa plástica feita em material termoplástico de alta resistência. O dispositivo conta com sensor de temperatura a prova d'água que será inserido dentro da panela cervejeira no momento da brassagem. A temperatura será lida pelo microcontrolador esp8266 que será responsável por controlar o acionamento de uma resistência elétrica de acordo com as rampas de temperatura inseridas pelo usuário.

A interface do sistema será feita por meio de dispositivo eletrônico com acesso a Wifi que se conectara em um servidor hospedado no microcontrolador. A interface será minimalista e de intuitiva, possibilitando que usuários não familiarizados com sistemas de automação possam manusear o sistema sem complicações e ao mesmo tempo poupe a memória limitada do esp8266. Além disso, o sistema conta com uma tela LCD de 20X4 com o objetivo de mostrar os status da brassagem, como: temperatura, etapa atual e tempo decorrido desde o início do processo.

3.2. Montagem do dispositivo

O Hardware foi escolhido com base na proposta do sistema, visando o baixo custo e facilidade de aquisição, o esp8266 une diversas funcionalidades como, por exemplo: Wifi integrado com suporte aos modos Station, modo SoftAP e SoftAP+Station, comunicação serial via cabo micro-usb.

Figura 2. Wemos D1 mini



Fonte: LOLIN D1 mini - WEMOS documentation, 201-

A placa wemos d1 mini, mostrada na figura 2, foi utilizada pelo seu baixo custo e tamanho reduzido. A mesma possui 11 pinos de entrada e saída, entrada micro usb, além de possuir o microcontrolador esp8266 integrado. Este microcontrolador conta com, CPU que opera em 80MHz até 160MHz, 32KBytes de RAM para instruções, 96KBytes de RAM para dados, e 64KBytes de ROM para boot. Além disso, possui documentação detalhada e grande suporte da comunidade.

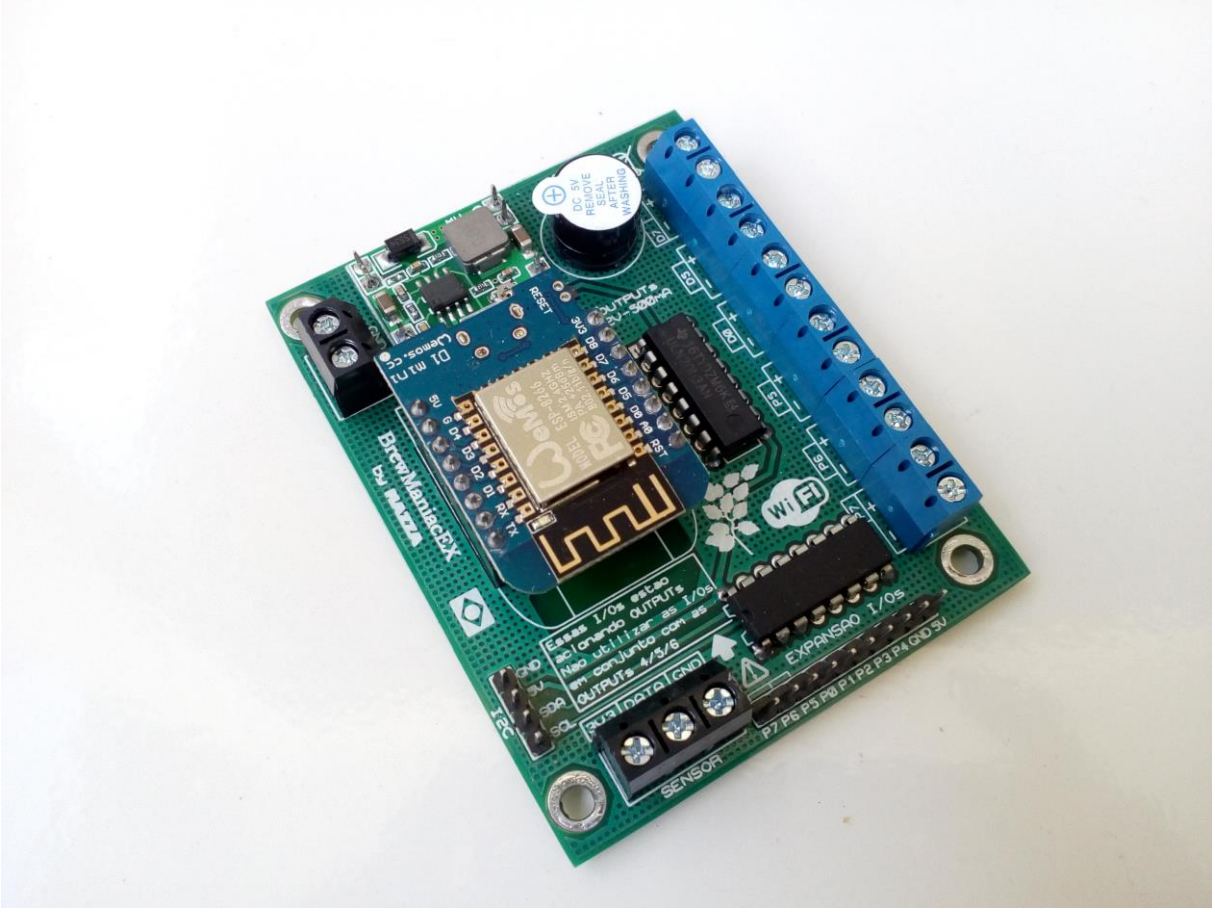
Tabela 1. Especificação Técnica Wemos D1 mini

Operating Voltage	3.3V
Digital I/O Pins	11
Analog Input Pins	1(3.2V Max)
Clock Speed	80/160MHz
Flash	4M Bytes
Size	34.2*25.6mm

Weight	3g

O sistema foi montado sobre a interface mazza handmade, mostrada na figura 3, que amplia os recursos da placa D1 mini. A placa possui buzzer, saída i2c para conexão de telas LCD, Bloco de terminais de parafuso, Módulo Conversor De Tensão ajustável e expansão de I/O.

Figura 3. Interface controladora de brassagem mazza handmade



Fonte: mazzahandmade, 201-

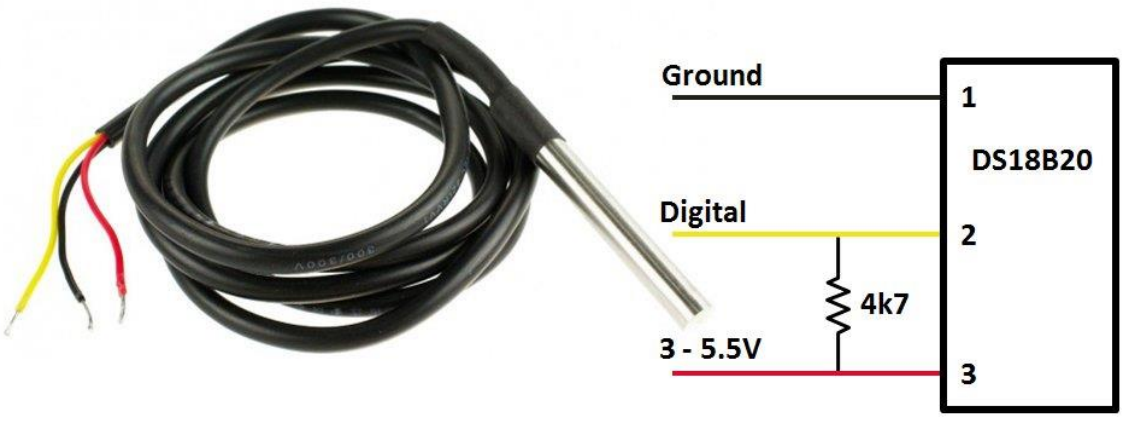
Para alimentar o sistema foi utilizado uma fonte chaveada de 40 watts mostrada na figura 4. Que apresenta voltagem de entrada bivolt alternada, voltagem de saída 12V de corrente contínua e 3 amperes. Como a tensão de operação do wemos D1 mini é de 3.3v e a fonte possui saída de 12v, o modulo step down realiza a correção para a tensão adequada.

Figura 4. Fonte de tensão chaveada de 40 watts

Fonte: Mercado Livre, Fonte Chaveada 40w X 12v Bivolt 3ª, 201-

Para realizar a medição das temperaturas foi utilizado o sensor de temperatura DS18B20, mostrado na figura 5, que se comunica através barramento Onewire. Além disso, cada DS18B20 possui o um código serial único de 64-bits, permitindo que vários sensores funcionem no mesmo barramento. O sensor trabalha em temperaturas entre -55°C até +125°C e tem precisão de 0.5ºC de -10ºC até 85ºC. A tensão de operação é de 3 a 5,5V.

Figura 5. Sensor de temperatura DS18B20



Fonte: nshopvn, 2019

A interface controladora de brassagem mazza handmade já possui bloco de terminais para a instalação dos sensores, além da resistência elétrica de 4,7 k ohms, necessária para que o sensor trabalhe na voltagem de 3,3V. Desta forma, possibilitando o acionamento do sistema de aquecimento a partir das temperaturas medidas.

O dispositivo de aquecimento utilizado para realizar o processo de mostura e fervura foi uma a RESISTÊNCIA ELÉTRICA PARA AQUECIMENTO ÁGUA OU MOSTO CERVEJEIRO EZbrew Booster 2150W – 220V – 10ª, mostrada na figura 6. Essa foi a resistência escolhida pois é uma resistência de baixa densidade. Essa característica significa que a resistência tem uma superfície de contato grande o bastante para espalhar toda sua potência em uma grande área de contato, distribuindo o calor apropriadamente em toda a panela. Logo, o mosto pode ser aquecido diretamente sem se preocupar em queimar caramelizar o mosto ou queimar cerveja.

Figura 6. Resistência elétrica 2150W

Fonte: Ezbrew, 201-

Outro aspecto a ser considerado na escolha da resistência é quantidade de água e malte das receitas que se deseja produzir, pois a mesma possui capacidade limitada de

aquecimento de acordo com a potência fornecida. Alguns cálculos aproximados podem ser feitos para estimar a potência necessária. A comunidade de cervejeiros caseiros costuma utilizar 1kwatts a casa 10 litros de água/mosto para ter uma fervura intensa, levando em conta essa proporção, a resistência utilizada tem capacidade de aquecer 21,5 litros de água/mosto de forma intensa.

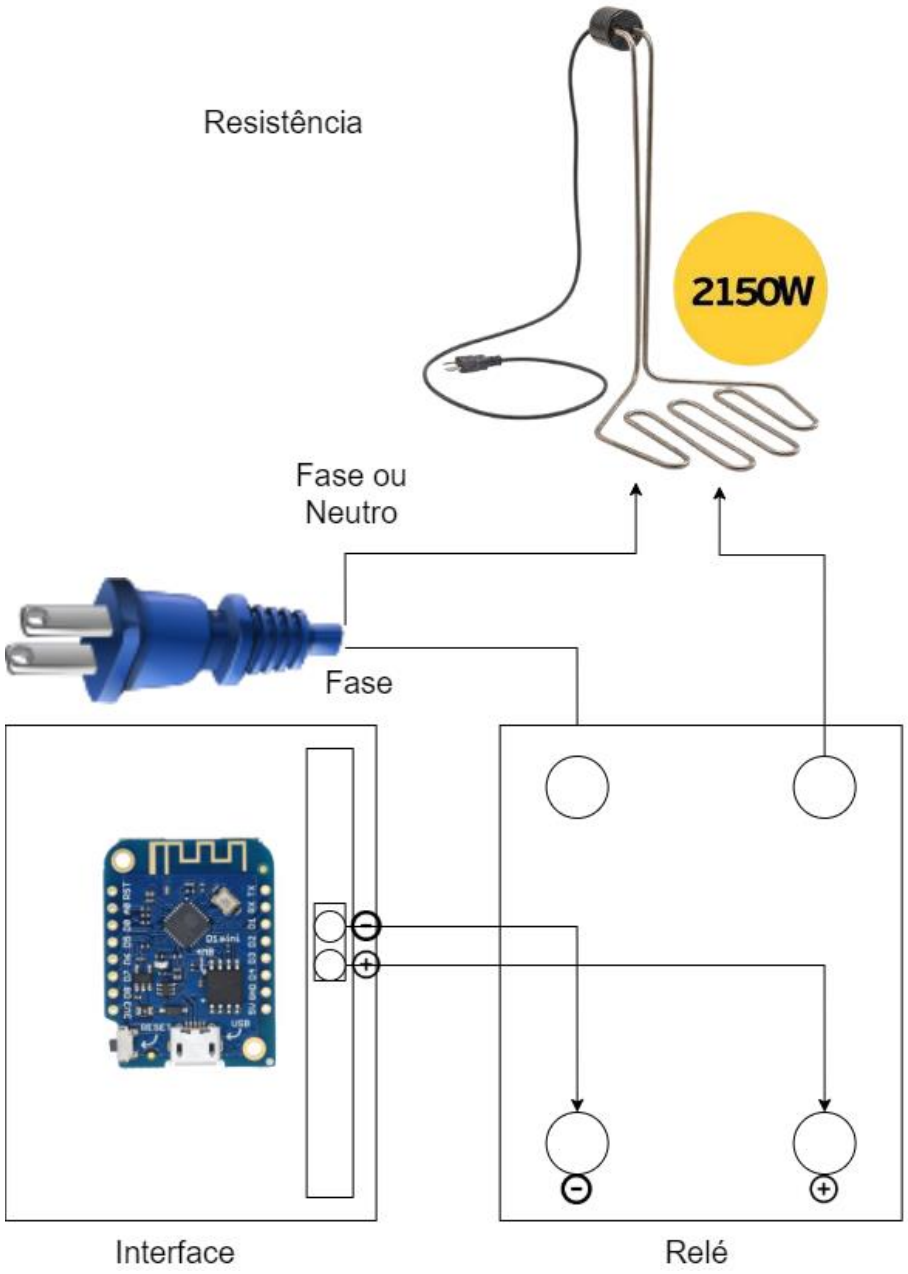
Para realizar o acionamento da resistência foi utilizado um relé SSR (Relé de estado sólido) de 25ª, mostrado na figura 7. O contato deste relé é normalmente aberto, possuindo tensão de entrada de 3-32VDC e tensão de saída de 24-380VAC. Tendo em vista que a resistência possui uma potência muito elevada em relação a potência de trabalho da interface de brassagem, o relé tem a função de acionar o interruptor da resistência a partir de um sinal de baixa potência completamente isolado do resto do sistema.

Figura 7. Relé de estado sólido 25A

Fonte: Amazon, 2013

Os terminais de entrada do relé foram conectados aos outputs D5 da interface de brassagem. Já nos terminais de saída foi realizada uma conexão em série entre a resistência e a fonte de energia, como mostrado na figura 8. Esse acionamento é realizado por meio do software que foi desenvolvido e embarcado no Wemos D1 mini, possibilitando a ativação de forma remota.

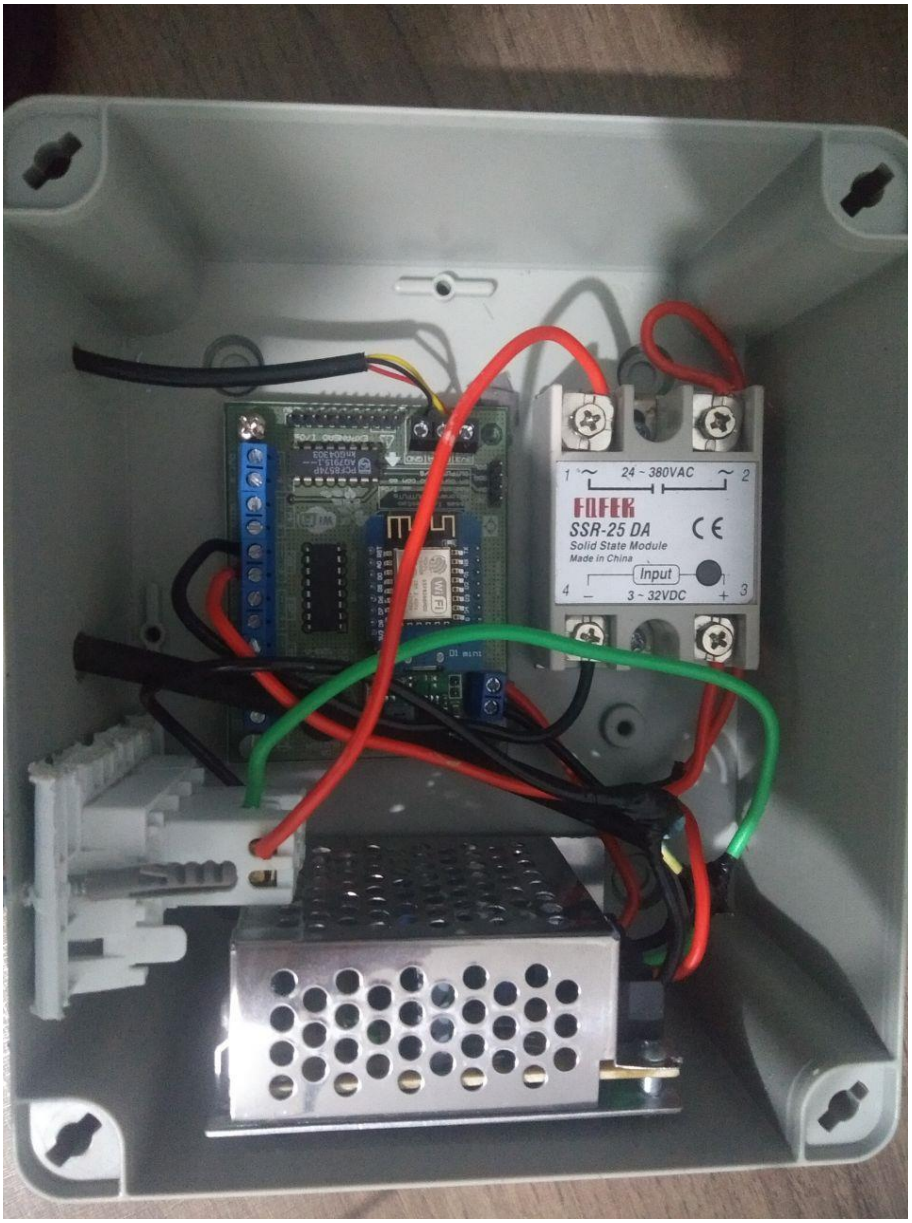
Figura 8. Esquema de ligação da resistência



Fonte: Autoral

A montagem final do circuito foi realiza em uma caixa de montagem eletrônica da marca steck fabricada em material termoplástico, como mostrado na figura 9. Esta caixa possui dimensões 170mm x 145mm x 84,5 mm e possui grau de proteção IP55. Na tampa da caixa foram montados o display LCD 20x4 e o teclado de 4 botões que servem como interface auxiliar para o usuário já que o sistema conta com a interface web como principal.

Figura 9. Montagem do controlador



Fonte: Autoral

3.3. Software

Os softwares vêm ganhando cada vez mais importância na sociedade atual. Presentes nas mais variadas áreas, são empregados desde a distribuição de energia, telecomunicação, nanotecnologia até as indústrias. Esses programas surgem como soluções para as mais diversas necessidades dos clientes. No caso do sistema aqui proposto não é diferente. As grandes industrias cervejeiras implementaram sistemas que automatizam sua produção, reduzindo custos e melhorando a produção. São sistemas complexos e de custo altíssimo, inviáveis economicamente para cervejeiros caseiros. Visando atender esse público que deseja sistemas, simples, de fácil usabilidade e custo baixo, este sistema foi proposto.

Para desenvolver um software que efetivamente resolve o problema do usuário, existe um campo de estudo chamado engenharia de software caracterizado por conjuntos de regras, processos e práticas que orientam o desenvolvimento do mesmo.

Segundo Bauer (1969). Engenharia de software é o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter software de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais.

Na engenharia de software existem diversos modelos de processo de software. Estes modelos podem ser entendidos como frameworks que descrevem as atividades durante o processo de software. Dentre os mais utilizados estão o modelo cascata e desenvolvimento incremental e Engenharia de software orientada a reuso.

Para determinar qual modelo de processo de software orientarão o desenvolvimento do software, é necessário primeiramente determinar a sua aplicação. As aplicações destes programas são vastas. Dentre os campos de aplicação de softwares, existem aqueles voltados para sistemas, aplicações específicas, científicos, engenharia, web, IA e muitos outros.

O software desenvolvido neste trabalho se trata de um software embutido. Segundo Sommervile (2011). Sistemas de controle embutidos. São sistemas de controle que controlam e gerenciam dispositivos de hardware. Numericamente, é provável que haja mais sistemas embutidos do que de qualquer outro tipo. Exemplos de sistemas embutidos incluem softwares em telefone celular, softwares que controlam antitravamento de freios em um carro e software em um micro-ondas para controlar o processo de cozimento.

Neste software será utilizado o desenvolvimento incremental pois é baseado na ideia de desenvolver várias versões até que um sistema adequado seja desenvolvido. Especificação, validação e desenvolvimento são realizados simultaneamente.

> "Desenvolvimento incremental de software, que é uma parte fundamental das abordagens ágeis, é melhor do que uma abordagem em cascata para a maioria dos sistemas de negócios, e-commerce sistemas pessoais. Desenvolvimento incremental reflete a maneira como resolvemos os problemas. Raramente elaboramos uma completa solução do problema com antecedência; geralmente movemo-nos passo a passo em direção a uma solução, recuando quando percebemos que cometemos um erro. Ao desenvolver um software de forma incremental, é mais barato e mais fácil fazer mudanças no software durante seu desenvolvimento." (Sommervile, 2011).

Após a definição que o tipo de aplicação será um sistema embutido e o modelo de processo de software será desenvolvimento incremental, segue-se com a análise inicial de requisitos do sistema.

Requisitos funcionais:

- [RF001] O Sistema deve exibir a temperatura atual.
- [RF002] O Sistema deve exibir a receita atual.
- [RF003] O Sistema deve exibir a etapa da brassagem.
- [RF004] O Sistema deve exibir o tempo decorrido.

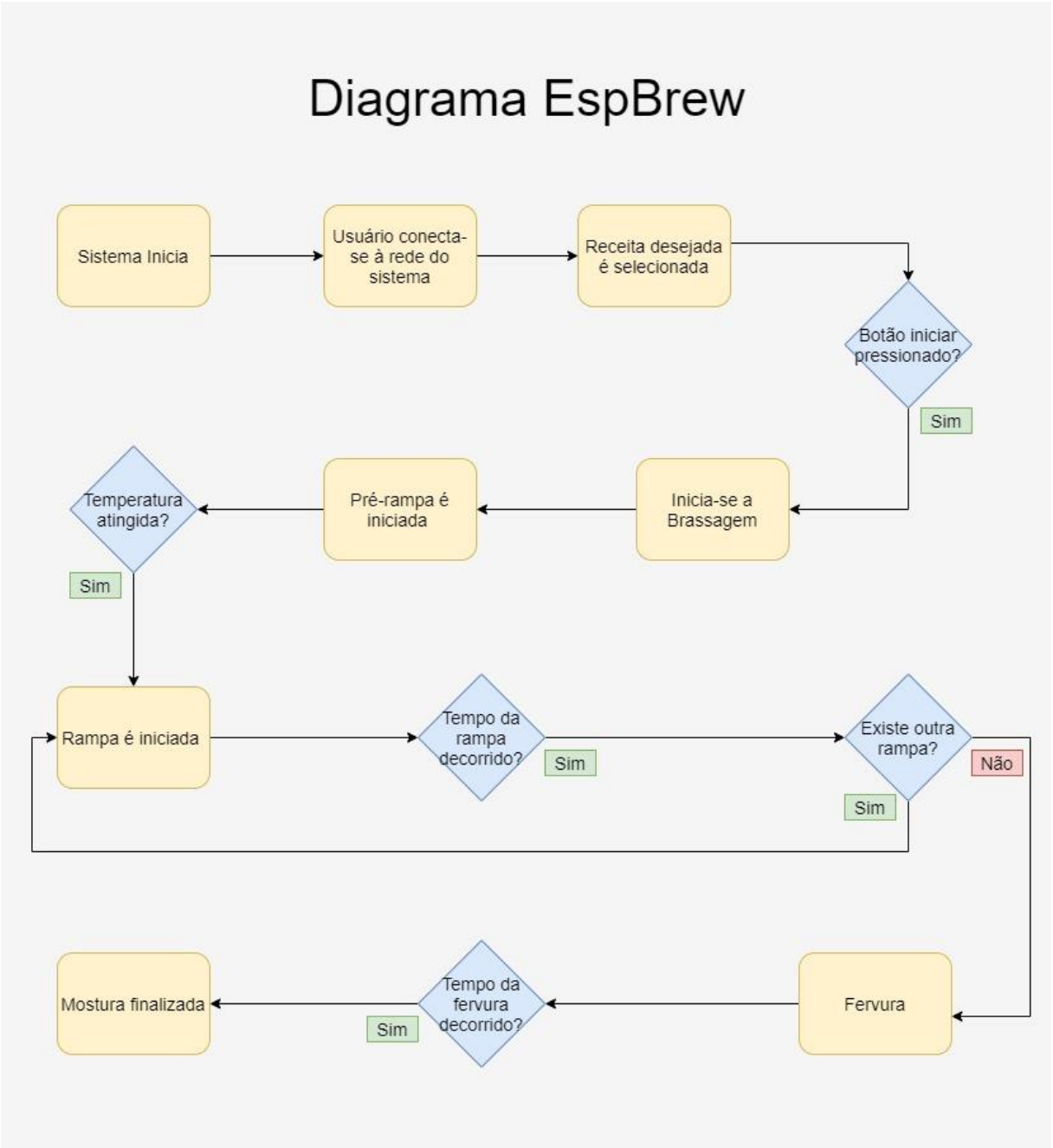
Requisitos não funcionais:

- [RNF001] O sistema não poderá ter mais de 4mb.
- [RNF002] A atualização da interface deve ocorrer a cada 15 segundos ou menos.
- [RNF003] O sistema deve ser implementado em C++.

Após a definição dos requisitos, as etapas realizadas em cada brassagem foram definidas, como mostrado na figura 10.

1. O usuário colocará água na panela.
2. Irá se conectar no sistema por meio do wifi roteado pelo esp8266
3. Selecionará a receita que deseja produzir através da interface
4. Ao clicar no botão “Iniciar” o brassagem será iniciada e a água aquecerá até a temperatura determinada.
5. Após a água atingir a temperatura adequada o sistema irá manter a temperatura da rampa com precisão de 1 grau (Podendo ser ajustado para atender os parâmetros desejados) até que percorra o tempo indicado.
6. Em seguida, o sistema verificará se existe outra rampa de temperatura, executando a etapa 5 respeitando os parâmetros de cada rampa de temperatura da receita.
7. Ao acabar todas as rampas da mostura, o sistema se aquece até a fervura e inicia a contagem do tempo de fervura.
8. Ao final da fervura, o sistema indicará que o processo foi concluído.

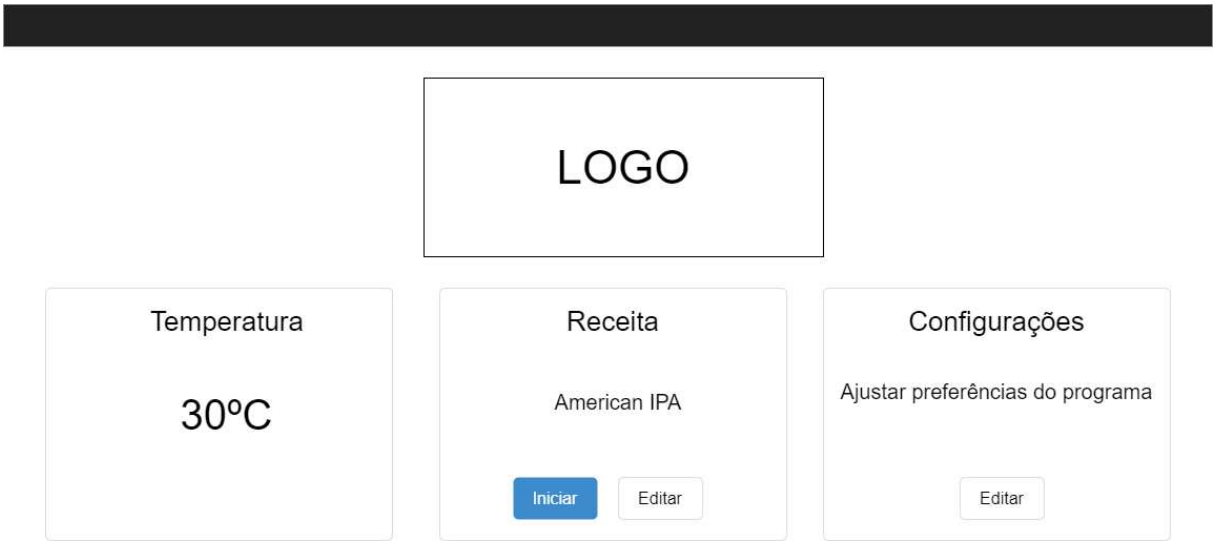
Figura 10. Fluxograma das etapas do sistema



Fonte: Autoral

Após a definição das etapas e análise de requisitos foi feito um protótipo da interface inicial do sistema de baixa fidelidade, a partir da ferramenta Figma, como mostrado na figura 11.

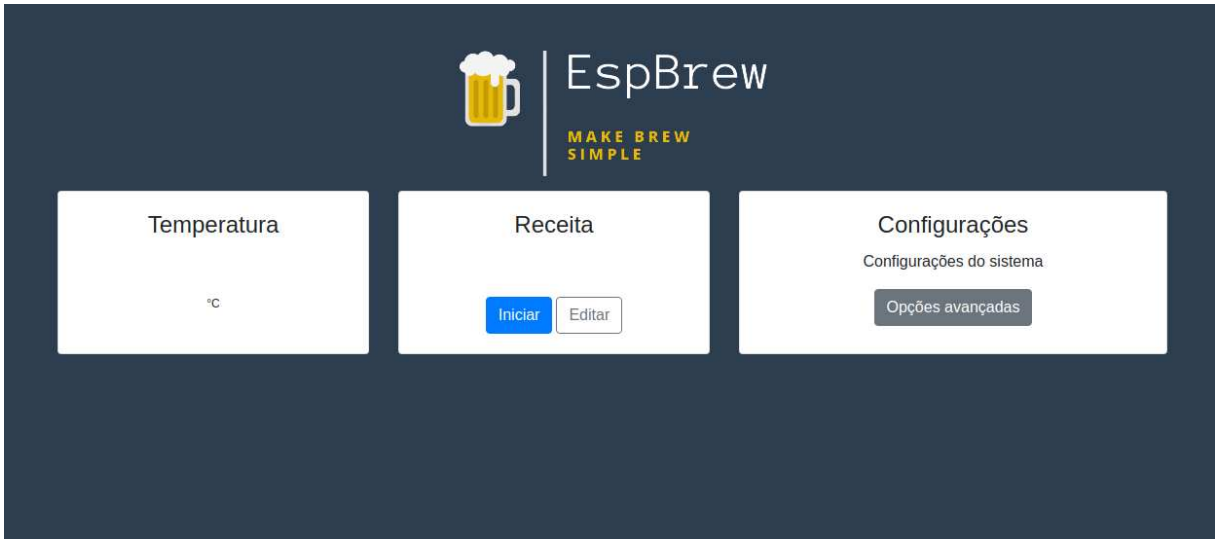
Figura 11. Protótipo de baixa fidelidade



Fonte: Autoral

Após a criação do protótipo de baixa fidelidade prosseguiu-se com o desenvolvimento de uma interface responsiva, mostrada na figura 12, que foi desenvolvida em para rodar em navegador web. As linguagens utilizadas no frontend foram a HTML para marcação, CSS para folha de estilo em conjunto com Javascript para possibilitar dinamismo.

Figura 12. Interface página principal



Fonte: Autoral

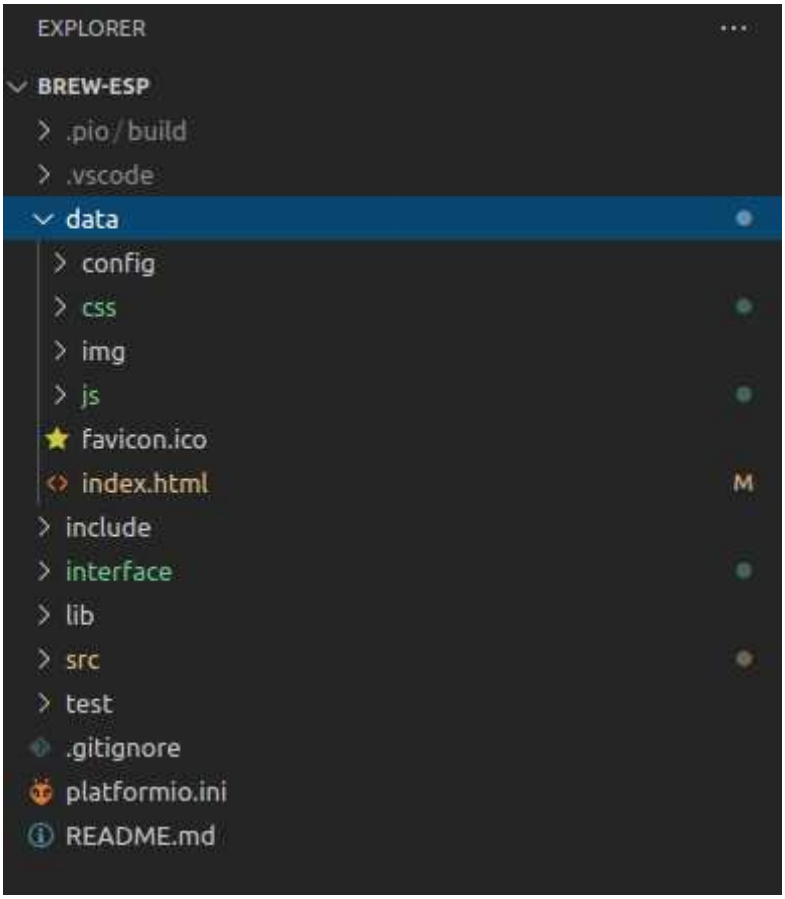
Além das tecnologias já apresentadas o framework Bootstrap, que possibilita a responsividade por meio da sua grande biblioteca de componentes e Vuejs, que permite composição de componentes, foram utilizados para o desenvolvimento se tornar mais rápido e fluido. A página web foi desenvolvida levando em conta as especificações do

microcontrolador esp8266, que possui poder de processamento e memória interna limitadas, logo o uso de bibliotecas frontend foi restringido a essas.

A linguagem utilizada no backend foi o C++, responsável por executar as funções no microcontrolador, como o servidor, tratamento da temperatura. A primeira etapa da configuração do ambiente, ocorreu no sistema operacional Linux Mint Ulyssa, em que foi utilizado a com o editor de códigos Visual Studio Code em Conjunto com a IDE Platformio.

Dentre as possibilidades para realizar a exibição da página web no microcontrolador, a mais simples é armazenar todo código HTML em uma String dentro do código em C++, porém é um método que dificulta o desenvolvimento e manutenção do software. A forma utilizada neste trabalho foi a utilização do sistema de arquivos do esp (*Serial Peripheral Interface Flash File System* ou SPIFFS) para realizar o acesso e gravação dos arquivos na memória flash do dispositivo.

Figura 13. Organização de diretórios



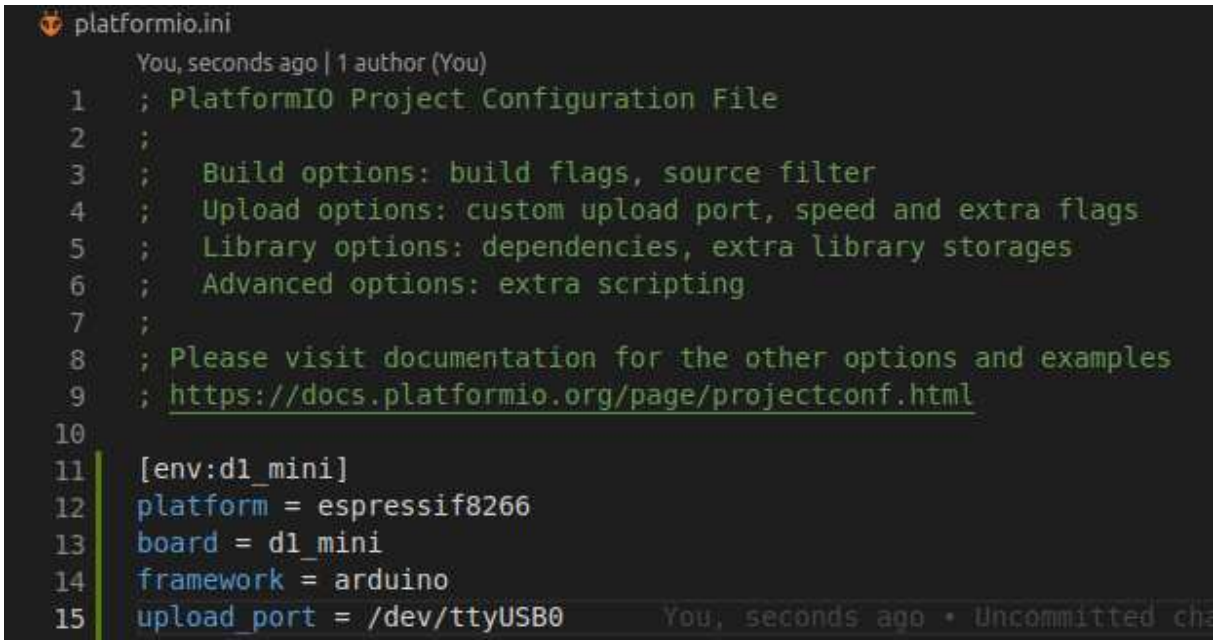
Fonte: Autoral

Por meio da IDE platformio, instalada como extensão no visual studio code, é possível organizar os diretórios do projeto para fazer o upload da interface de forma simplificada, como é mostrado na figura 13. A platformio é um ambiente de desenvolvimento que dispõe

de um conjunto de ferramentas que tornam o desenvolvimento pra sistemas embarcados mais eficiente.

As configurações do platformio devem ser preenchidas no arquivo platformio.ini, como mostrado na figura 14, que contém informações sobre a plataforma, placa, framework e a porta de upload utilizada no projeto. Preenchida estas configurações, basta criar a pasta "data" na raiz do diretório e mover os arquivos da interface para esta pasta e selecionar a opção "*upload system image*".

Figura 14. Arquivo de configuração platformio



Fonte: Autoral

Os arquivos da interface, que são gerados pelo framework vuejs, são separados em arquivos css, imagens e arquivos javascript. Além destes arquivos, a pasta config possui um arquivo de configuração da brassagem em json, que será explicado mais à frente.

Com os arquivos gravados na memória é necessário criar um servidor de arquivos para exibir a página web. Para isto, foi utilizada a biblioteca EspAsyncWebServer, que é um servidor HTTP Assíncrono e WebSocket para ESP8266. Por meio deste servidor podemos enviar diversas requisições em paralelo, e cada resposta retorna quando estiver pronta.

Dando progresso com o desenvolvimento, é necessário gravar um sketch para controlar o esp. Neste projeto foi utilizada a biblioteca Arduino.h, que traz suporte para o chip ESP8266 para o ambiente Arduino. Ele permite escrever esboços, usando funções e bibliotecas familiares do Arduino, e executá-los diretamente no ESP8266, sem a necessidade de microcontrolador externo.

A biblioteca EspAsyncWebServer e seus requisitos foram baixadas e movidas para o diretório lib, assim como todas as outras bibliotecas do esp neste projeto.

Figura 15. Inicializar leitura do sistema de arquivos

```
//  Start SPIFFS read      You, a month ago • add all files
if(!SPIFFS.begin()){
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
}
```

Fonte: Autoral

Em seguida, inclui-se a biblioteca SPIFFS e inicia-se a leitura do sistema de arquivos, retorna-se uma exceção com uma mensagem de erro caso a operação não tenha sucesso, como mostrado na figura 15.

Figura 16. Envio da página web

```
// Send web page to client
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(SPIFFS, "/index.html");
});
```

Fonte: Autoral

Posteriormente a biblioteca foi importada no código de fonte foi criado uma instância do servidor assíncrono utilizando a porta 8080 e a página index.html foi enviada por meio do sistema de arquivo do esp, como mostrado na figura 16.

Para que o usuário tenha acesso ao servidor do dispositivo foi incluída a biblioteca ESP8266WIFI. Essa biblioteca provê rotinas wifi que permitem a conexão com a internet, que permitem três modos de conexão wifi, que são: *Station* (STA), *Acess Point* (AP) e o modo híbrido (APSTA).

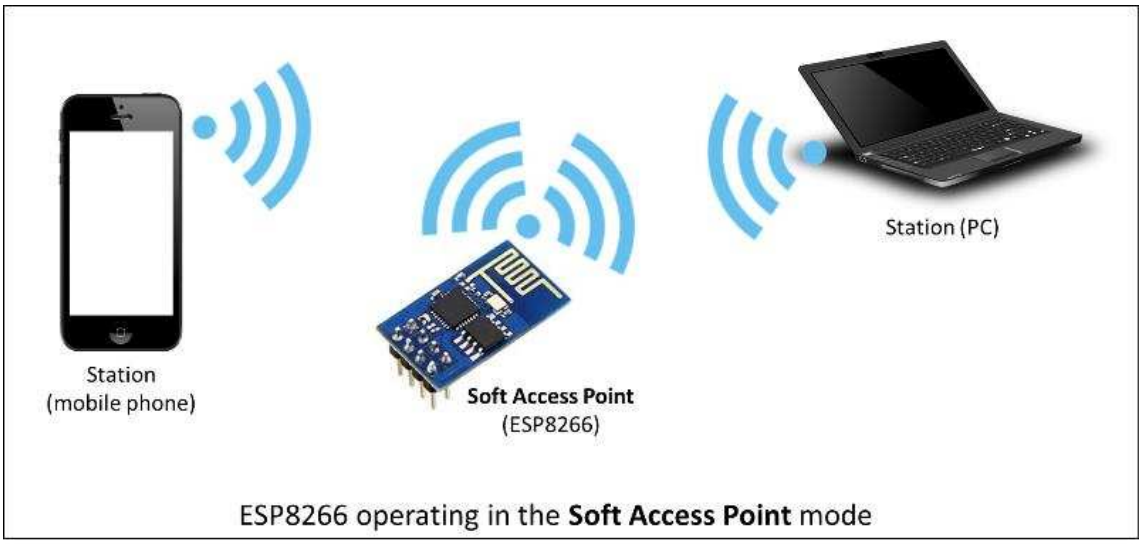
Figura 17. Configuração wifi

```
// Start access point
WiFi.mode(WIFI_AP);
WiFi.softAP(ssid, password);
```

Fonte: Autoral

Neste projeto, o dispositivo foi configurado no modo WIFI_AP. Este modo permite que o esp funcione como ponto de acesso de rede wifi para outros dispositivos, como mostrado na figura 18. O SSID da rede foi definido como espbrew e foi protegido por meio de senha para que usuários indesejados não tenham acesso à rede, como mostrado na figura 17.

Figura 18. Esp em modo AP



Fonte: https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html

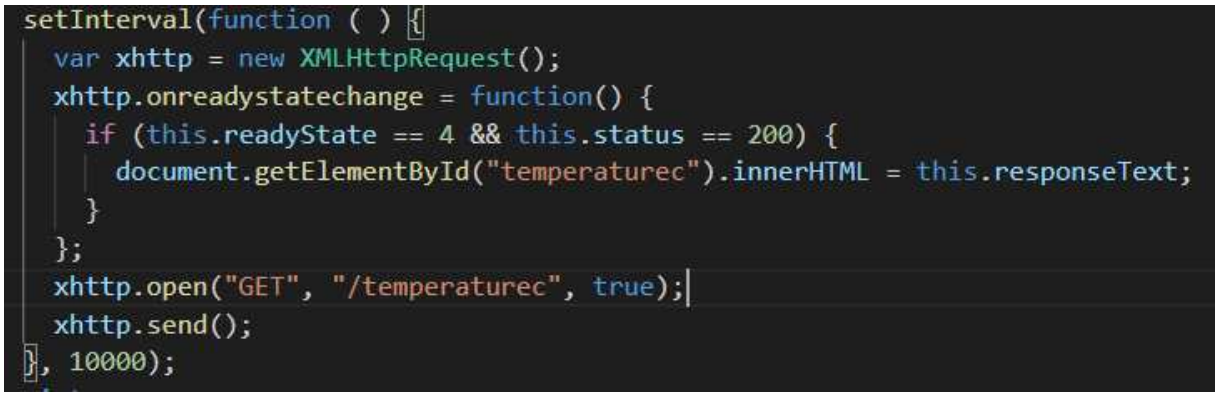
Após realizar o acesso a rede, o usuário está apto a acessar o servidor web ao digitar o endereço de rede 192.168.4.1 no navegador de seu dispositivo. Quando essa ação é realizada o usuário será redirecionado para a página principal do sistema, na qual ele terá a possibilidade de iniciar a brassagem e acompanhar o progresso da mesma.

Entretanto, a interface ainda não está configurada para receber os valores de temperatura e os dados da receita por meio do microcontrolador, pois para isso necessita que estes valores sejam enviados pelo microcontrolador. O fato do código *frontend* ter sido gravado no sistema de arquivos do esp, torna este processo um tanto quanto complicado, pois a página será carregada antes de receber as informações necessárias. Portanto, será necessário utilizar um *placeholder* que receberá o valor da temperatura em tempo de execução.

No código HTML da interface desenvolvido anteriormente, foi adicionado a *string* %TEMPERATUREC% no card de temperatura. Este é um espaço reservado para os valores de temperatura. Isso significa que este texto %TEMPERATUREC% é como uma variável que será substituída pelo valor real da temperatura do sensor. Os espaços reservados no texto HTML devem ficar entre os sinais%.

A Função processor (), será responsável por substituir os espaços reservados em nosso texto HTML pelos valores reais de temperatura. Essa função verifica se o HTML possui marcadores de posição quando a página da web é solicitada. Se encontrar o marcador% TEMPERATUREC%, retornamos a temperatura em Celsius chamando a função de leitura da temperatura. Para realizar a requisição da temperatura foi desenvolvido um script na interface que utiliza XMLHttpRequest, como mostrado na figura 19.

Figura 19. Requisição



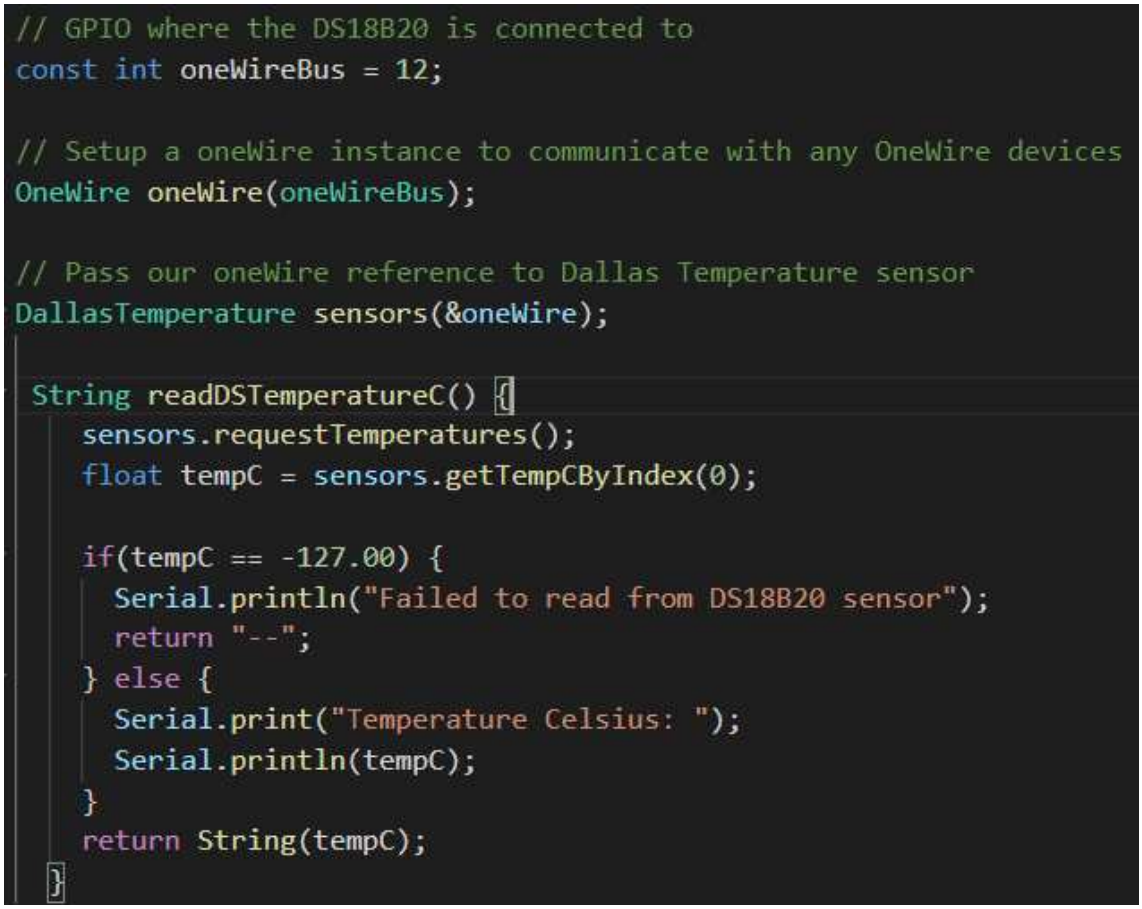
Fonte: Autoral

XMLHttpRequest (XHR) é uma API disponível em linguagens de script para navegadores web tais como JavaScript. É utilizada para enviar requisições HTTP ou HTTPS diretamente para um servidor web e carregar os dados de resposta do servidor diretamente de volta ao script.[6]

Esta abordagem permite recuperar dados de um URL sem ter que fazer uma atualização de página inteira. Isso permite que uma página da Web atualize apenas uma parte do conteúdo sem interromper o que o usuário esteja fazendo. Desta forma, a temperatura pode ser atualizada sem que a interface que tenha que recarregar. Como mostrado na figura 18, a cada 10000 milissegundos é feita uma requisição pela temperatura, e a mesma é atualizada sem que a página inteira tenha que ser recarregada.

Para realizar a medição de temperatura foram utilizadas as bibliotecas OneWire.h e DallasTemperature.h. Foi definido o GPIO ao qual o pino de dados DS18B20 está conectado. Neste caso, ele está conectado ao GPIO 12. O pino 12 é passado como referência para a criação do objeto oneWire, que configura uma instância para comunicar com outros objetos do tipo, caso existam. Esse objeto é passado como parâmetro para a instância de um sensor da biblioteca DallasTemperatura, que possui funções para facilitar a leitura da temperatura.

Figura 20. Função leitura temperatura



```
// GPIO where the DS18B20 is connected to
const int oneWireBus = 12;

// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);

// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

String readDSTemperatureC() {
    sensors.requestTemperatures();
    float tempC = sensors.getTempCByIndex(0);

    if(tempC == -127.00) {
      Serial.println("Failed to read from DS18B20 sensor");
      return "--";
    } else {
      Serial.print("Temperature Celsius: ");
      Serial.println(tempC);
    }
    return String(tempC);
  }
```

Fonte: Autoral

Posteriormente, foi definida a função readDSTemperaturec() que lê a temperatura do sensor e a retorna o valor “- -“ em caso de falha. Em caso de sucesso a temperatura obtida pelo sensor é retornada em graus celsius, vide figura 19.

A parte principal do software é a responsável pelo gerenciamento das rampas de temperatura, pois nessa etapa é fundamental que o sistema tenha limiares bem definidos de quando a resistência deve ser acionada e quando deve ser desligada.

> Depois de criar uma função setup(), a qual inicializa e atribui os valores iniciais, a função loop() faz precisamente o que o seu nome sugere, e repete-se consecutivamente enquanto a placa estiver ligada, permitindo o seu programa mudar e responder a essas mudanças. Use-a para controlar ativamente uma placa Arduino.[7]

O esp utiliza essa mesma funcionalidade do Arduino, permitindo que nele sejam desenvolvidos os códigos que devem se repetir durante cada ciclo de execução do programa. Dentro desta rotina foi escrito o trecho de software (Presente no apêndice A) que é

responsável por fazer o controle do acionamento da resistência de acordo com a etapa da brassagem.

Estas etapas acontecem de acordo com as rampas de temperatura. Vamos supor que esteja sendo criada uma receita do estilo Cream Ale com rendimento de 20 litros. Esta receita precisa alcançar 66 ºC para executar a primeira rampa, logo definimos uma etapa chamada de pré-rampa 1 que será responsável por elevar a temperatura até o limiar de 66ºC.

No momento em que este limiar é atingido o sistema alterna para a próxima etapa, que neste caso é a rampa 1. Neste momento se inicia a contagem da rampa 1, que deve manter a temperatura em 66ºC durante 60 minutos sem que ajam variações bruscas na temperatura.

Para manter a temperatura estável o software possui uma margem de 1ºC para o acionamento da resistência, por exemplo: Se a temperatura da mostura chegar a 67ºC (Temperatura da rampa + margem de temperatura) a resistência deve ser desligada, e caso a temperatura chegue a 65ºC (Temperatura da rampa - margem de temperatura) a resistência será acionada. Para cada etapa do processo de brassagem esta lógica se repetirá até que a última rampa chegue ao fim.

Na primeira versão do protótipo os dados da receita serão gravados em um arquivo json. Os dados serão a temperatura de cada rampa (em graus celsius), a duração de cada rampa (em segundos), e os momentos de adição de lúpulo e outros adjuntos. Este arquivo é gravado no SPIFFS do esp8266 e é enviado para a interface por meio do servidor web criado anteriormente.

Para que o usuário tenha maior controle sobre a situação da receita, a interface LCD exibirá informações sobre o status atual da brassagem, como mostrado na figura 21. As informações principais são: tempo de brassagem, temperatura e etapa atual. Para isso foi realizado a inclusão da biblioteca LiquidCrystal_I2C.h.

Figura 21. Informações LCD

Fonte: Autoral

Para informar ao usuário o momento certo para adicionar lúpulos e outros adjuntos o sistema acionará um alarme sonoro por meio de um buzzer além de ativar avisos visuais tanto na interface web quanto no display lcd.

3.4. Teste do protótipo

O intuito desta etapa é avaliar se o software e o hardware atendem as necessidades do usuário.

> Até mesmo os desenvolvedores de software mais experientes concordarão que software de alta qualidade é um objetivo importante. Mas como definir a qualidade de software? No sentido mais geral, a qualidade de software pode ser definida como: uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam. (Sommervile, 2011).

Para garantir a qualidade do protótipo, durante esta etapa foi feito a aferição do mesmo para verificar o funcionamento como um todo. Hardware e software são analisados em funcionamento para que sejam identificados erros e pontos de melhorias para serem corrigidos em versões posteriores do sistema.

Foi realizada a produção de uma receita do estilo Cream Ale com rendimento de 20(l), como mostrado na tabela 2.

Tabela 2. Receita Cream Ale 20l

RECEITA				
Descrição		Rendimento (litros)	Peso malte (kg)	Água(litros)
Cream Ale		20	4,6	32
ESTILO (BJCP 2015)				
OG	FG	IBU	SRM	ABV
1,042 – 1,055	1,006 – 1,012	8,0 – 20,0	2,5 – 5,0	4,2 – 5,6
INGREDIENTES				
Matéria-prima			Quantidade	UN

Caramel Pils	0,100	Kg
Malte Pilsen	4,500	Kg
Lúpulo Cascade	15,0	g
Lúpulo Magnum	9,0	g
Fermento US-05	1,0	UN
Whirfloc Tablet	1,0	UN
MOSTURA		
Processo	Temperatura(C)	Tempo(min)
Rampa de Temperatura 1	66	60
Mash Out	75-77	10
FERVURA (60 minutos)		
Processo	Descrição	
Lúpulo Magnum	Adicionar 9g (60 min)	
Lúpulo Cascade	Adicionar 5g (30 min)	
Whirfloc Tablet	Adicionar 1UN (15 min)	
Lúpulo Cascade	Adicionar 10g (0 min)	
FERMENTAÇÃO		
Temperatura ideal	Entre 15º e 20ºC	
Tempo recomendado	7 dias	
MATURAÇÃO		
Temperatura ideal	Entre 0º e 4ºC	
Tempo recomendado	Entre 7 e 15 dias	
PRIMMING		
CO2		

Fonte: Autoral

Inicialmente foi executada a limpeza de todos os equipamentos para remoção da sujeira superficial e impedir a proliferação de microrganismos durante a produção. Entretanto

só a limpeza não é suficiente para ter alta eficiência. Portanto prosseguimos com a etapa de sanitização dos equipamentos.

Sanitizar significa reduzir microrganismos críticos para saúde pública em níveis considerados seguros, com base em parâmetros estabelecidos, sem prejudicar nem a qualidade do produto nem a sua segurança.[8]

Esse processo é empregado principalmente em indústrias alimentícias para obter-se uma maior redução da presença microbiana. Nesta etapa foi utilizado uma solução de água e iodofor 2,25% na proporção de 1 litro de água para 0,5ml de iodofor que foi aplicada em todos equipamentos durante 5 minutos.

Após a sanitização, o equipamento foi montado e a resistência elétrica foi conectada ao mesmo. O saco de grãos foi colocado na panela e em seguida adicionou-se a água. O sensor de temperatura e a resistência foram dispostos na panela, como mostrado na figura 22, e o sistema foi iniciado. A interface foi acessada por meio de notebook conectado a rede do dispositivo. Em seguida iniciou-se a brassagem.

Figura 22. Início da etapa de testes

Fonte: Autoral

A mosturação foi iniciada com o aquecimento dos 32 litros de água até a temperatura de 66ºC, na etapa que foi chamada de pré-rampa 0. O sistema demorou cerca de 1 hora para

aquecer a água, tempo que poderia ser menor caso fosse utilizada resistência elétrica de maior potência, mas que não afetou o resultado final do processo.

Logo após a rampa 0 iniciou-se, o malte foi adicionado e a temperatura se manteve entre 65 e 67ºC durante uma hora.

Em alguns momentos o sensor apresentou falha ao medir a temperatura. Como a temperatura é requisitada a cada 10 segundos não afetou substancialmente o processo, mas, de toda forma, essa etapa foi sinalizada para correção posterior.

Dando continuidade ao processo, prosseguiu-se com o *MashOut* que é uma rampa de temperatura que ocorre após a mosturação com objetivo de inativar enzimas e diminuir a viscosidade de mosto.

Após essa etapa o saco de grãos é escorrido para realizar a filtragem dos resíduos sólidos do malte. O líquido decorrente da filtragem é aquecido até iniciar a etapa de fervura, que perdura por 60 minutos.

Como resultado, a densidade foi medida por meio de um refratômetro, como mostrado na figura 23 e o mosto apresentou densidade de 1,043, alcançando assim, a faixa de densidade original indicada pela receita. O mosto então foi resfriado e as leveduras foram inoculadas em temperatura ambiente, dando início a etapa de fermentação. Durante os 7 dias seguintes a cerveja foi mantida a uma temperatura de 18ºC em um balde fermentador dentro de uma geladeira.

Figura 23. Refratômetro

Fonte: Autoral

Após a fermentação, foi realizada nova medição com o refratômetro, mostrado na figura 23, a receita obteve 1,010 de densidade final. Por meio da calculadora de volume alcoólico da yeastlabs foi realizado o cálculo do volume alcoólico obtendo o resultado de 4,33%, como mostrado na figura 24.

Figura 24. Volume alcoólico obtido

Fonte: Autoral

A Cerveja foi mantida mais duas semanas em temperatura de 4ºC em maturação, para aperfeiçoar as características sensoriais e para que as leveduras que estavam em suspensão decantassem. Ao final, a cerveja foi transferida para um barril postmix e sofreu carbonatação forçada, que é aquela em que se aplica o CO_2 diretamente na cerveja.

A cerveja apresentou perfil sensorial, de acordo com o esperado. Cor límpida, coloração amarelo claro, levemente lupulada, refrescante e com espuma persistente. Indo de acordo com a classificação indicada pela BJCP. Comprovando assim, que o equipamento é capaz de fornecer as condições necessárias para a produção de cerveja artesanal em pequena escala.

3.5. Correções e aprimoramentos

Por estar sendo utilizada uma abordagem iterativa e incremental, as correções e aprimoramentos foram feitos no decorrer do desenvolvimento de software. Foram atribuídas às tarefas ordem de prioridade e níveis de complexidade para priorização da próxima sprint, como mostrado na tabela 3.

Tabela 3. Tarefas futuras

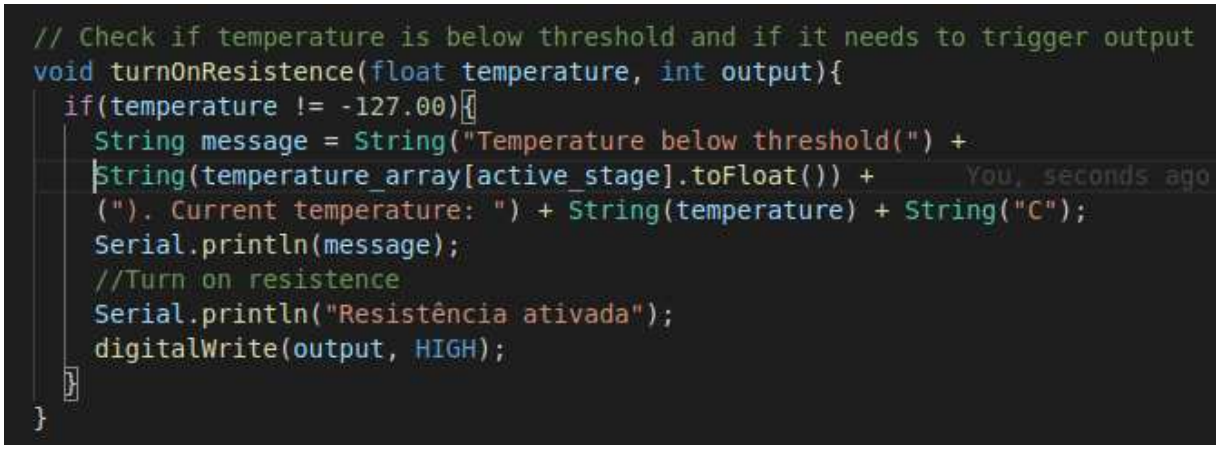
Tarefa	Prioridade	Complexidade
Correção Leitura temperatura	Alta	Baixa
Acionamento manual da resistência	Baixa	Média
Pausa/continuação da mostura	Baixa	Alta
Alarmes para adição de lúpulo/adjuntos	Média	Média

Fonte: Autoral

O processo de melhoria foi iniciado pela correção da leitura de temperatura, pois é um processo de prioridade alta. Medições erradas na temperatura influenciam diretamente no resultado da produção da cerveja. Além disso, esta tarefa possui complexidade baixa, podendo assim ser resolvida rapidamente.

Ao apresentar falha na leitura, o sensor de temperatura retorna -127 ºC. Anteriormente o software acionava a resistência, pois a temperatura era menor que todas as rampas de temperatura. Para resolver esta falha, foi adicionada a função condicional que só é executada se a temperatura for diferente de -127ºC. Quando a temperatura retornada pelo sensor for esta, a resistência não será ativada, fazendo com que o estado anterior seja mantido até que o sensor leia a temperatura corretamente, como mostrado na figura 25.

Figura 25. Correção da função de acionamento da resistência



```
// Check if temperature is below threshold and if it needs to trigger output
void turnOnResistence(float temperature, int output){
  if(temperature != -127.00){
    String message = String("Temperature below threshold(") +
    String(temperature_array[active_stage].toFloat()) +      You, seconds ago
    ("). Current temperature: ") + String(temperature) + String("C");
    Serial.println(message);
    //Turn on resistence
    Serial.println("Resistência ativada");
    digitalWrite(output, HIGH);
  }
}
```

Fonte: Autoral

A criação do alarme para adição de lúpulos e adjuntos foi desenvolvida em seguida por ter prioridade média. Como a adição destes produtos é um processo manual, o usuário pode abrir brecha para falhas humanas. Tendo em vista este cenário foi desenvolvida a funcionalidade de emitir alerta sonoro nos momentos em que devem ser adicionados os ingredientes.

4. CONCLUSÕES

O sistema aqui apresentado, cumpriu sua função de auxiliar do processo de produção de cerveja artesanal, especificamente na etapa de brassagem. A partir da etapa de testes, que contou com a produção de uma cerveja do estilo Cream Ale, foi possível verificar que o protótipo possibilitou a fabricação de forma mais simples e eficiente. Todos estes fatores contribuem para que a bebida mantenha um padrão elevado e consequentemente mitigar possíveis erros cometidos por falha humana.

Melhorias futuras podem ser implementadas para que o sistema seja disponibilizado comercialmente. Para isso, é necessário a implementação de testes que garantam ainda mais a confiabilidade do sistema. Outra possível melhoria é a utilização do protocolo MQTT (Message Queuing Telemetry Transport), por ser simples e leve, acarretando em inúmeras vantagens a um sistema IOT.

Apesar das ressalvas o protótipo apresentou desempenho satisfatório ao cumprir seu objetivo com exito, tendo em vista a limitação em relação a tempo, suporte financeiro e capital humano.

5. REFERÊNCIAS

[1] Norman Nise. **Engenharia de Sistemas de Controle**, sexta ed.

[2] Você s/a. **Cerveja artesanal tem mercado quente para empreender e carreiras em alta**. Disponível em: https://vocesa.abril.com.br/empreendedorismo/mercado-de-cervejas-artesanais/. Acesso em: 20 out. 2020.

[3] Sebrae. **Como montar uma microcervejaria**. 202-. Disponível em: https://www.sebrae.com.br/sites/PortalSebrae/ideias/como-montar umamicrocervejaria,8f387a51b9105410VgnVCM1000003b74010aRCRD. Acesso em: 28 out. 2021.

[4] MARK, Joshua. **Beer in the Ancient World**. 2011. Disponível em: https://www.ancient.eu/article/223/beer-in-the-ancient-world/. Acesso em: 01 fev. 2021.

[5] MORADO, Ronaldo. **Larousse da cerveja**: a história e as curiosidades de uma das bebidas mais populares do mundo. [S. L.]: Alaúde, 2018. 444 p.

[6] **XMLHttpRequest Level 1**, 6 set. 2016. Disponível em: https://www.w3.org/TR/XMLHttpRequest/. Acesso em: 28 jan. 2020.

[7] **LOOP() - Documentação de Referência do Arduino**. 1 jun. 2019. Disponível em: https://www.arduino.cc/reference/pt/language/structure/sketch/loop/. Acesso em: 28 fev. 2021.

[8] PFUNTNER, Allan. **Sanitizers and Disinfectants: The Chemicals of Prevention**. 1 ago. 2011. Disponível em: https://www.food-safety.com/articles/6707-sanitizers-and-disinfectants-the-chemicals-of-prevention. Acesso em: 1 mar. 2021.

[9] BASSOLI, Monali. **Cinema e Cerve**ja, 2017. Disponível em: https://cinemaecerveja.com.br/descomplicando-a-cerveja-03-conhe%C3%A7a-o-processo-de-fabrica%C3%A7%C3%A3o-da-cerveja-4d47b0881c9b . Acesso em: 03 fev. 2021.

[10] **LOLIN D1 mini - WEMOS documentation**. Disponível em: https://www.wemos.cc/en/latest/d1/d1_mini.html . Acesso em: 15 fev. 2021.

[11] **Fonte Chaveada 40w X 12v Bivolt 3a**. Disponível em: https://produto.mercadolivre.com.br/MLB-1103364598-fonte-chaveada-40w-x-12v-bivolt-3a-_JM. Acesso em: 04 fev. 2021.

[12] **Sensor de temperatura DS18B20 de fio macio**. Disponível em: https://nshopvn.com/product/cam-bien-nhiet-do-ds18b20-day/ Acesso em: 10 fev. 2021.

[13] **Resistência elétrica para aquecimento água ou mosto cervejeiro Ezbrew Booster 2150w – 220v 10a | Panela automatizada para fazer cerveja**. Disponível em: https://ezbrew.com.br/featured_item/resistencia-eletrica-para-aquecimento-agua-ou-mosto-cervejeiro-ezbrew-booster-2150w-220v-10a/ . Acesso em: 10 fev. 2021.

[14] **YKS Solid State Relay Module SSR-25DA 25A /250V 3-32V DC Input 24-380VAC Output.** Disponível em: https://www.amazon.co.uk/Solid-Module-SSR-25DA-24-380VAC-Output/dp/B00F998JSM. Acesso em: 10 fev. 2021.

[15] **Interface Controlador Brassagem Wi-Fi BrewManiacEX - BrewUNO.** Disponível em: https://www.mazzahandmade.com.br/produtos/interface-controlador-brassagem-wi-fi-brewmaniacex-brewuno. Acesso em: 10 fev. 2021.

[16] **O que é Internet das Coisas?.** Disponível em: https://www.redhat.com/pt-br/topics/internet-of-things/what-is-iot. Acesso em: 07 abr. 2021.

APÊNDICE

Apêndice A – Código esp8266

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <Hash.h>
#include <FS.h>
#include <ESPAsyncWebServer.h>
#include <OneWire.h>
#include <Wire.h>
#include <DallasTemperature.h>
#include <ArduinoJson.h>
#include <TimeLib.h>
#include <LiquidCrystal_I2C.h>

#define MAX_RECIPE_FILE_SIZE 2024
#define RECIPE_FILE "/config/config.json"
LiquidCrystal_I2C lcd(0x27, 20, 4); // set the LCD address to 0x0 for a 16 cha
rs and 2 line display


// FS *_fs;
String Quantidade_rampas;
String Temperatura_1;
String Tempo_1;
String time_array[5];
String temperature_array[5];
String lastTemperature;
String recipeName;
boolean brew_started = 0;
boolean init_clock = 0;
boolean display_brew_msg = 1;
boolean preRamp = 1;
unsigned long brew_init_time = 0;
unsigned long ramp_start_time = 0;
int active_stage = 0;

// Interval between sensor readings.
unsigned long previousMillis = 0;
const long interval = 5000;
//Network Credentials
const char* ssid = "NewBrewSystem";
const char* password = "12345678";

AsyncWebServer server(80);
WiFiServer server1(81);
```

```
void notFound(AsyncWebServerRequest *request) {
  request->send(404, "text/plain", "Not found");
}

void showWelcome(){

  lcd.init();// initialize the lcd
  lcd.backlight();
  lcd.setCursor(1, 0);
  lcd.print("WELCOME");
  lcd.setCursor(1, 1);
  lcd.print("ESPBREW");
}
void showTemperature(float temperature){

  lcd.clear();
  lcd.setCursor (0,0); //character zero, line 1
  lcd.print("Temperatura: "); // print text
  lcd.setCursor (14,0); //character zero, line 1
  lcd.print(temperature);
}

void showActiveStage(int active_stage, boolean preRamp){

  lcd.setCursor (0,1); //character 4, line 2
  if(preRamp){
    lcd.print("PRE-RAMPA "); // print text
    Serial.println("Pré rampa: " + String(active_stage));

  }else{
    lcd.print("RAMPA "); // print text
    Serial.println("Rampa: " + String(active_stage));

  }
  lcd.print(active_stage); // print text
}

 String returnActiveStage(int active_stage, boolean preRamp){
   String $msg;
  if(preRamp){
    $msg = "Pré rampa " + String(active_stage);

  }else{
    $msg = "Rampa " + String(active_stage);
  }
  return $msg;
 }
```

```
void showTime(unsigned long currentMillis, unsigned long brew_init_time){

  lcd.setCursor (0,2); //character 0, line 3
  lcd.print("Tempo: "); // print text

  //Conversão do tempo para ser exibido
  unsigned long seconds = (currentMillis - brew_init_time) / 1000;
  unsigned long minutes = seconds / 60;
  seconds %= 60;
  lcd.setCursor (9,2); //character 0, line 3
  lcd.print(minutes);
  lcd.print(':');
  if (seconds < 10) {
    lcd.print('0');
  }
  lcd.print(seconds);
}

// Check if temperature is below threshold and if it needs to trigger output
void turnOnResistence(float temperature, int output){
  if(temperature != -127.00){
    String message = String("Temperature below threshold(") + String(temperatu
re_array[active_stage].toFloat()) + ("). Current temperature: ") +
    String(temperature) + String("C");
    Serial.println(message);
    //Turn on resistence
    Serial.println("Resistência ativada");
    digitalWrite(output, HIGH);
  }
}

// Check if temperature is above threshold and if it needs to trigger output
void turnOffResistence(float temperature, int output){
  String message = String("Temperature above threshold(") + String(temperature
_array[active_stage].toFloat()) + ("). Current temperature: ") +
            String(temperature) + String("C");
  Serial.println(message);
  //Turn off resistence
  Serial.println("Resistência desativada");
  digitalWrite(output, LOW);
}

void initRamp(float temperature){
  //Inicia contagem de tempo da rampa
  init_clock = 0;
  //Desativa pré rampa
  preRamp = 0;
  Serial.println("Iniciando rampa: " + String(active_stage));
```

```
  //Se a temperatura ambiente for maior ou igual a temperatura alvo e se a ram
pa não foi iniciada, a rampa inicia
  while (init_clock == 0)
  {
      ramp_start_time = millis();
      Serial.println("ramp_start_time: " + String(ramp_start_time));
      init_clock = 1;
  }
}

void finishBrew(int output){
    brew_started = 0;
    Serial.println("Brassagem finalizada");
    Serial.println("Resistência desativada");
    lcd.setCursor (0,4); //character 4, line 2
    lcd.print("Brassagem finalizada"); // print text


    digitalWrite(output, LOW);
}

boolean loadRecipeSettings()
{
    File configFile = SPIFFS.open(RECIPE_FILE, "r");

    if (configFile)
    {
      Serial.println("Arquivo lido com sucesso");
        size_t size = configFile.size();
        if (size <= MAX_RECIPE_FILE_SIZE)
        {
            DynamicJsonDocument _activeStatusJsonDocument = DynamicJsonDocumen
t(MAX_RECIPE_FILE_SIZE);
            DeserializationError error = deserializeJson(_activeStatusJsonDocu
ment, configFile);
            if (error == DeserializationError::Ok && _activeStatusJsonDocument
.is<JsonObject>())
            {
              JsonObject _activeStatus = _activeStatusJsonDocument.as<JsonObje
ct>();
              Quantidade_rampas = _activeStatus["quantidade_rampas"].as<String
>();
              recipeName = _activeStatus["nome"].as<String>();

              for (size_t i = 0; i < Quantidade_rampas.toInt(); i++)
              {
                temperature_array[i] = _activeStatus["temperatura_" + String(i
+1)].as<String>();
```

```
                time_array[i] = _activeStatus["tempo_" + String(i+1)].as<Strin
g>();
                Serial.println("temperatura_" + String(i) + " = " + String(tem
perature_array[i]) + " tempo_" + String(i) + " = " + String(time_array[i]));
              }
                configFile.close();
            }
        }
        configFile.close();
    }
    return true;
}

static const char ACTIVE_STATUS[] PROGMEM = "{\"nome\":{{nome}},\"quantidade_r
ampas\":{{quantidade_rampas}},\"tempo_1\":{{tempo_1}},\"temperatura_1\":\"{{te
mperatura_1}}\" }";

String getJson()
{
    String active_status = FPSTR(ACTIVE_STATUS);
    active_status.replace("{{nome}}", String(recipeName));
    active_status.replace("{{quantidade_rampas}}", String(Quantidade_rampas));
    active_status.replace("{{tempo_1}}", String(time_array[0]));
    active_status.replace("{{temperatura_1}}", String(temperature_array[0]));

    return active_status;
}

String getRecipe(){
    String htmlRecipe;
    htmlRecipe =  "<h1>" + recipeName + "</h1>";
    htmlRecipe += "<h3>" + returnActiveStage(active_stage, preRamp) + "</h3>";
    htmlRecipe += "<ul style=\"list-style: none; text-
align:left;\" > <li> Quantidade de rampas: " + Quantidade_rampas + "</li>";
    htmlRecipe += "<li> Tempo rampa 1: " + time_array[0] + " min</li>";
    htmlRecipe += "<li> Temperatura rampa 1: " + temperature_array[0] + " ºC</
li></ul>";

    return htmlRecipe;
}
// GPIO where the output is connected to
const int output = 14;

// GPIO where the DS18B20 is connected to
const int oneWireBus = 12;

// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);
```

```
// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

 String readDSTemperatureC() {
    // Call sensors.requestTemperatures() to issue a global temperature and Re
quests to all devices on the bus
    sensors.requestTemperatures();
    float tempC = sensors.getTempCByIndex(0);

    if(tempC == -127.00) {
      Serial.println("Failed to read from DS18B20 sensor");
      return "--";
    } else {
      Serial.print("Temperature Celsius: ");
      Serial.println(tempC);
    }
    return String(tempC);
  }

  // Replaces placeholder with variable values
  String processor(const String& var){
    //Serial.println(var);
    if(var == "TEMPERATUREC"){
      return readDSTemperatureC();
    }
    else if(var == "RECEITA"){
      return getRecipe();
    }
    return String();
  }
void setup(){

  showWelcome();

  // Starts Serial Monitor with 9600 baud rate
  Serial.begin(9600);

  // Starts DS18B20 Sensor
  sensors.begin();

  //  Start SPIFFS read
  if(!SPIFFS.begin()){
     Serial.println("An Error has occurred while mounting SPIFFS");
     return;
  }

  //Load recipe config
  loadRecipeSettings();
```

```
  Serial.println("Configuring access point...");

  // Start access point
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);

  // IP Address of our access point
  IPAddress ip_address = WiFi.softAPIP();

  // Start web server
  server.begin();
  Serial.print("AP IP address: ");
  Serial.println(ip_address);

  pinMode(output, OUTPUT);
  digitalWrite(output, LOW);

  // Send web page to client
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html",String(), false, processor);
  });

    // Send a GET request to <IP>/get?message=<message>
  server.on("/get", HTTP_GET, [] (AsyncWebServerRequest *request) {
      String message;
      if (request->hasParam("brew_started")) {
          brew_started = request->getParam("brew_started")->value();
          brew_init_time = millis();
          message = "Brew started";
          // message =  getJson();
      } else {
          message = "Brew not started";
      }
      request->redirect("/");
      // request-
>send(200, "text/html", "Brew Started: " + message + "<br><a href=\"/\">Return
 to Home Page</a>");
  });

  server.on("/css/app.bb028a70.css", HTTP_GET, [](AsyncWebServerRequest *reque
st){
    request->send(SPIFFS, "/css/app.bb028a70.css", "text/css");
  });

  server.on("/css/chunk-
vendors.367538c2.css", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/css/chunk-vendors.367538c2.css", "text/css");
  });
```

```
  server.on("/js/app.5509a63a.js", HTTP_GET, [](AsyncWebServerRequest *request
){
    request->send(SPIFFS, "/js/app.5509a63a.js");
  });

  server.on("/js/chunk-
vendors.334450e7.js", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/js/chunk-vendors.334450e7.js");
  });

  server.on("/favicon.ico", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/favicon.ico");
  });

  server.on("/img/logo.2a84def0.png", HTTP_GET, [](AsyncWebServerRequest *requ
est){
    request->send(SPIFFS, "/img/logo.2a84def0.png");
  });

  //  Read Celsius Temperature
  server.on("/temperaturec", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", readDSTemperatureC().c_str());
  });

  server.on("/receita", HTTP_GET, [](AsyncWebServerRequest *request)
  {
      // request->send(200, "application/json", getJson());
      request->send(200, "text/plain", getRecipe());
  });

}

void loop(){
  // Starts if
  if(brew_started){
    while(display_brew_msg)
    {
        Serial.println("Brew started");
        lcd.clear();
        lcd.print("Brew started");
        display_brew_msg = 0;
    }
    unsigned long currentMillis = millis();
    //Mensure temperature each 5 seconds
    if (currentMillis - previousMillis >= interval) {
      previousMillis = currentMillis;

      // Temperature in Celsius degrees
      sensors.requestTemperatures();
```

```
      float temperature = sensors.getTempCByIndex(0);

      showTemperature(temperature);
      showTime(currentMillis, brew_init_time);

      //Se estiver em pré-rampa, acionar a resistência até chegar ao threshold
      if(preRamp){
        showActiveStage(active_stage, preRamp);
        digitalWrite(output, HIGH);
        if(temperature >= (temperature_array[active_stage].toFloat())){
          initRamp(temperature);
        }
      }else{
        showActiveStage(active_stage, preRamp);
        // if Horário atual >= Horário do fim da rampa
        if(currentMillis >= ((time_array[active_stage].toFloat() * 1000) + ram
p_start_time)){
          active_stage ++;
          preRamp = 1;
        }else if(temperature >= (temperature_array[active_stage].toFloat() + 1
)){
          turnOffResistence(temperature, output);
        }else if(temperature < (temperature_array[active_stage].toFloat() -
 1)){
          turnOnResistence(temperature, output);
        }
        if(active_stage == (Quantidade_rampas.toInt())){
          finishBrew(output);
        }
      }
    }
  }
}
```