INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO AMAZONAS

Henrique Dolzane Mota

FERRAMENTA DE PROGRAMAÇÃO POR COMANDO DE VOZ PARA A GERAÇÃO DE CÓDIGO FRONT-END DE WEBSITES

Manaus, Amazonas - Brasil

2020

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO AMAZONAS

Henrique Dolzane Mota

FERRAMENTA DE PROGRAMAÇÃO POR COMANDO DE VOZ PARA A GERAÇÃO DE CÓDIGO FRONT-END DE WEBSITES

Orientador: Emmerson Santa Rita da Silva, Me.

Trabalho de Conclusão de Curso apresentado à banca examinadora do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Amazonas – IFAM Campus Manaus – Centro, como requisito para o cumprimento da disciplina TCC II – Desenvolvimento de Software.

Manaus, Amazonas - Brasil

2020

Dados Internacionais de Catalogação na Publicação (CIP)

M917f Mota, Henrique Dolzane.
Ferramenta de programação por comando de voz para a geração de código Front-End de websites. – Manaus: IFAM, 2020.
66p. : il. color.

Monografia (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia do Amazonas, C*ampus* Manaus Centro, 2020.
Prof. Me. Emmerson Santa Rita da Silva

1. Informática. 2. Desenvolvimento de Sistemas. 3. Software. I. Silva, Emmerson Santa Rita da. (Orient.) II. Instituto Federal de Educação, Ciência e Tecnologia do Amazonas III. Título.

CDD 005.43

Elaborado por Odimar Porto - CRB 11/496

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO MÉDIA E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA - AM
DEPARTAMENTO ACADÊMICO DE INFORMAÇÃO E COMUNICAÇÃO
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA AMAZONAS

TERMO DE APROVAÇÃO

A monografia, que tem como título: "**FERRAMENTA DE PROGRAMAÇÃO POR COMANDO DE VOZ PARA A GERAÇÃO DE CÓDIGO FRONT-END DE WEBSITES**" foi submetida à defesa pública, sob a avaliação de banca examinadora, como parte dos requisitos necessários para a obtenção do título de graduação do curso superior de Tecnologia em Análise e Desenvolvimento de Sistemas.

AUTOR (A): HENRIQUE DOLZANE MOTA

Monografia aprovada em: 07/10/2020

Emmerson Santa Rita da Silva

Orientador (a):

Primeiro (a) examinador (a):

Digitalizado com CamScanner

Segundo (a) examinador (a):

RESUMO

Este trabalho explora o desenvolvimento de software utilizando a voz como ferramenta de programação, como alternativa ao uso de mouse e teclado. O desenvolvimento se deu a partir de pesquisas sobre o estado-da-arte acerca dos conceitos envolvidos, assim como pela proposta de uma arquitetura modular que utiliza reconhecimento de voz. A pesquisa culminou em uma aplicação capaz de reconhecer comandos de voz, em um ambiente web, e permitir a criação e manipulação de código fonte para o desenvolvimento de páginas web. A demonstração desses princípios ilustra que é possível expandir as formas de desenvolvimento de software, permitindo maior inclusão de pessoas no universo da programação, além de evitar possíveis injúrias físicas por esforço repetitivo.

Palavras Chaves: reconhecimento de voz; desenvolvimento *front-end*; aplicações *web*;

ABSTRACT

Abstract*. Hereby we explore the software development by using voice as a tool, as an alternative for the use of mouse and keyboard. The development was based on research on the state-of-the-art about the concepts involved, as well as by the proposal of a modular architecture that uses voice recognition. The research resulted in an application capable of recognizing voice commands, in the web environment, and allowing the creation and manipulation of source code for the development of web pages. The demonstration of these principles illustrates that it is possible to expand the way how software is developed, allowing greater inclusion of people in the universe of programming, in addition to avoiding possible physical injuries due to repetitive strain.*

Keywords*: voice recognition; front-end development; web applications;*

LISTA DE FIGURAS

Figura 1 - Arquitetura simplificada cliente-servidor 14
Figura 2 – Requisição síncrona X Requisição assíncrona 15
Figura 3 - Escopo básico HTML 16
Figura 4 - Captura de tela da interface gráfica da ferramenta Repl.it 18
Figura 5 - Captura de tela da ferramenta PlayCode 18
Figura 6 - Captura de tela da ferramenta CodePen 19
Figura 7 - Captura de tela do ambiente CodeSandBox 19
Figura 8 - Captura de tela da interface de demonstração da API Cloud Speech-to-text 22
Figura 9 - Captura de tela da interface de demonstração da API IBM Watson Speech to text. 22
Figura 10 - Captura de tela da interface de demonstração da API Web Speech 23
Figura 11 - Módulos da arquitetura proposta 27
Figura 12 – Organização do módulo de reconhecimento de voz 27
Figura 13 - Organização do módulo de manipulação de texto 28
Figura 14 - Organização do módulo de renderização de resultado 29
Figura 15 - Diagrama de casos de uso 31
Figura 16 - Diagrama de atividades 31
Figura 17 – Estrutura do módulo Express Handlebars 40
Figura 18 – Arquivo container principal – Express Handlebars 41
Figura 19 – Instância de reconhecimento de voz 42
Figura 20 – Arquitetura geral da aplicação 43
Figura 21 – Diagrama de sequência dos módulos da aplicação 45
Figura 22 – Diagrama de classes 45
Figura 23 – Integração dos módulos da aplicação com serviços terceiros 46
Figura 24 – Front-end da aplicação 48
Figura 25 – Código do front-end da aplicação 49
Figura 26 – Uso de CSS Grid para definir disposição de layout 49
Figura 27 – Ilustração de disposição da propriedade grid-template-areas 50
Figura 28 – Atribuição de posição de layout 50
Figura 29 – entrypoins.js 51

Figura 30 – scripts.js 52
Figura 31 – Estrutura dos módulos da aplicação 53
Figura 32 – Diagrama de sequência – Iniciar reconhecimento de voz 56
Figura 33 – Diagrama de sequência – Finalizar reconhecimento de voz 57
Figura 34 - Diagrama de sequência – Manipulação de texto de reconhecimento de voz 58
Figura 35 – Diagrama de sequência – Renderizar resultado 59

LISTA DE QUADROS

Quadro 1 - Comparação de características entre ferramentas de trabalhos relacionados.....26
Quadro 2 - Requisitos funcionais 29
Quadro 3 - Requisitos não funcionais 30
Quadro 4 - Lista de palavras selecionadas para os testes 32
Quadro 5 - Descrição de siglas usadas nas tabelas de resultados dos testes 33
Quadro 6 - Resultados dos testes com palavras da API Cloud Speech-to-Text.....33
Quadro 7 - Acertos dos testes com palavras da API Cloud Speech-to-Text.....34
Quadro 8 - Resultados dos testes com palavras da API IBM Watson Speech to text.....35
Quadro 9 - Acertos dos testes com palavras da API IBM Watson Speech to text.....36
Quadro 10 - Resultados dos testes com palavras da API Web Speech 36
Quadro 11 - Acertos dos testes com palavras da API Web Speech 37
Quadro 12 – Cronograma de projeto de software (2019).....66
Quadro 13 - Cronograma de desenvolvimento de software (2020) 66

Sumário

Sumário 7
1 Introdução 8
1.1 Justificativa 9
1.2 Objetivos 9
1.2.1 Objetivo geral 9
1.2.2 Objetivos Específicos 9
1.3 Metodologia 10
2 Fundamentação teórica 11
2.1 LER/DORT 11
2.2 Aplicações WEB 13
2.3 Tecnologias *Front-End* 15
2.3.1 HTML, CSS e JavaScript 15
2.4 Ambientes de Desenvolvimento para WEB 17
2.5 Reconhecimento de voz 20
2.5.1 API's para reconhecimento de voz 21
3 Trabalhos Relacionados 24
3.1 Comparação entre as ferramentas 25
4 Proposta do anteprojeto 26
4.1 Arquitetura Proposta 26
4.1.1 Módulo de reconhecimento de voz 27
4.1.2 Módulo de manipulação de texto 28
4.1.3 Módulo de renderização de resultado 28
4.2 Especificação dos requisitos 29
4.2.1 Requisitos Funcionais 29
Quadro 2 – Requisitos funcionais 29
4.2.2 Requisitos não funcionais 29
Quadro 3 – Requisitos não funcionais 30
4.3 Modelagem do sistema 30
4.3.1 Diagrama de casos de uso 30
4.3.2 Diagrama de atividades 31
5 Experimentos 32
5.1 Experimentos de reconhecimento de voz 32
5.1.1 Google Cloud Speech-to-Text 33

5.1.2 IBM Watson Speech to text 34
5.1.3 Web Speech API 36
5.2 Conclusão sobre os experimentos das API's 37
6 Desenvolvimento 39
6.1 Tecnologias utilizadas 39
6.1.1 Node.js 39
6.1.2 Express 40
6.1.3 Express Handlebars 40
6.1.4 CodeMirror 41
6.1.5 Web Speech API 42
6.2 Arquitetura da aplicação 43
6.2.1 Funcionamento Geral 43
6.2.2 Arquitetura Modular 44
6.2.3 Diagrama de classes da aplicação 46
6.2.4 Contextos de comandos de voz 46
6.3 Código da aplicação 47
6.3.1 Front-end da aplicação 47
6.3.2 Pontos de entrada da aplicação 51
6.3.3 Módulos da aplicação 52
6.3.4 Diagramas de sequência da aplicação 56
7 Considerações Finais 60
7.1 Riscos e Dificuldades 61
8 Trabalhos futuros 62
9 Referências 63
10 Anexos 65
10.1 Descrição dos casos de uso 65
10.1.1 Caso de uso Gerar Código 65
10.1.2 Caso de uso Reconhecer comandos de voz 65
10.1.3 Caso de uso Converter comandos em código 65
10.1.4 Caso de uso Renderizar código 65
10.2 Cronogramas 66

1 Introdução

Na área de desenvolvimento de software é muito comum dedicar horas na atividade de programação e, dessa forma, fazer o uso intensivo de dispositivos como mouse e teclado. Dentre os muitos ambientes de desenvolvimento existentes no mercado, poucos oferecem alternativas para a diminuição do esforço repetitivo do uso desses dispositivos.

O esforço repetitivo pelo uso de mouse e teclado pode ocasionar lesões e até invalidez, cuja condição é aplicada a todo aquele que é considerado incapaz e insusceptível de reabilitação para o exercício de atividade que lhe garanta a subsistência (ASSUNÇÃO e ALMEIDA, 2003). Segundo Luttmann et al. (2003), a sobrecarga mecânica em músculos, tendões e ossos é uma das principais causas de desordens físicas, por isso é necessário que haja equilíbrio entre a carga de trabalho exercida e a carga suportada. Além dessas injúrias, o uso de mouse e teclado é limitado em atender pessoas que apresentam problemas motores, já que sua experiência de uso é prejudicada devido as suas limitações físicas.

Dentro desse contexto, o uso do reconhecimento de voz pode ser uma alternativa viável ao uso desses dispositivos físicos de entrada de dados. Este recurso pode ser usado por usuários com problemas motores, assim como a entrada de dados por voz pode ser uma terapia recomendada para lesões por esforço repetitivo (COHEN e OVIATT, 1993). O uso da linguagem tem sido reconhecido como o fluxo mais natural, conveniente e eficiente de compartilhamento de informação, já que diariamente 75% da comunicação humana é feita por voz (LIU, 2010). A comunicação por voz é uma das formas mais eficazes de comunicação utilizada pelo ser humano, e pode sanar questões relacionadas aos prejuízos à saúde e dificuldades de acessibilidade apresentados.

A evolução de metodologias de interação homem-máquina, tem permitido novas abordagens como as chamadas interfaces de usuário naturais, definida por Liu (2010) como “metodologias de interação com o computador que enfoca habilidades humanas como toque, visão, voz, movimentos e funções cognitivas superiores, como expressão e evocação”.

Nessa perspectiva, percebeu-se a necessidade de se buscar formas de atender a falta de medidas alternativas para o desenvolvimento de software que não sejam dependentes do uso de dispositivos clássicos de entrada de dados. A problematização tratada neste trabalho foi a seguinte: Como dinamizar a criação de programas de computador em ambientes de

desenvolvimento que ofereçam alternativas ao uso de dispositivos clássicos de entrada de dados como mouse e teclado?

De forma a delimitar o escopo do trabalho, foram investigadas alternativas para a tratativa do reconhecimento de voz na criação de um ambiente para o desenvolvimento de aplicações web a partir de tecnologias Front-End como HTML, CSS e JavaScript.

1.1 Justificativa

A partir do contexto exposto na seção anterior, nota-se que o uso da fala, com base nas características destacadas por Liu (2010), é uma alternativa viável visto que a fala é uma forma natural humana de interação, e dispensa o uso das mãos, principal fato relacionado à problemática apresentada do excesso de uso dispositivos de entrada de dados, como de mouse e teclado. Dessa forma, o presente trabalho visa demonstrar que é possível utilizar a fala pra desempenhar atividade de desenvolvimento de software e reduzir bastante a necessidade do uso de ferramentas como os dispositivos de entrada de dados citados, de forma a evitar danos físicos ou servir como auxílio a tratamentos em casos de lesões por esforço repetitivo, além de auxiliar pessoas já debilitadas fisicamente e que, por vezes, ficariam impedidas de realizar atividades de programação.

1.2 Objetivos

A seguir são definidos o objetivo geral e os objetivos específicos estabelecidos para o presente trabalho.

1.2.1 Objetivo geral

Desenvolver um protótipo de um ambiente de programação para a geração de código *front-end* de aplicações *web* que aceite comandos de voz como forma de entrada de dados.

1.2.2 Objetivos Específicos

- Implementar módulo de reconhecimento de voz.
- Implementar módulo de manipulação de texto.
- Implementar módulo de renderização de resultado.

- Integrar módulos para gerar aplicação *front-end.*

1.3 Metodologia

Conforme os objetivos estabelecidos, a execução do trabalho se deu a partir do cumprimento da seguinte metodologia:

- Levantamento bibliográfico que inclui arquivos e pesquisas referentes ao tema, assim como trabalhos relacionados.
- Análise dos trabalhos correlatos como forma de entender como se deu o desenvolvimento deles, a fim de buscar técnicas que poderão ser utilizadas neste trabalho.
- Realização de experimentação com ferramentas e técnicas de reconhecimento de voz, a fim de escolher qual tecnologia mais se aplica aos propósitos da proposta deste trabalho.
- Implementação da aplicação proposta no trabalho, com base nas técnicas e ferramentas escolhidas.

2 Fundamentação teórica

Este capítulo apresenta definições fundamentais para a compreensão do contexto abordado na proposta do presente trabalho.

2.1 LER/DORT

Segundo Kuorinka e Foncier (1995), por definição, LER (Lesões por Esforço Repetitivo) ou DORT (Distúrbios Osteomusculares Relacionados ao Trabalho) são siglas que definem quadros clínicos do sistema musculoesquelético adquiridos pelo trabalhador submetido a determinadas condições de trabalho. Além disso, caracterizam-se pela ocorrência, ou não, de sintomas concomitantes que podem se manifestar de maneira insidiosa, na maioria das vezes nos membros superiores por meio de dores, parestesia, sensação de peso e fadiga.

De acordo com Manual Técnico do Ministério da Saúde de 2001 sobre LER e DORT, os termos se referem de maneira abrangente aos distúrbios ou doenças do sistema músculo esquelético, principalmente de pescoço e membros superiores, relacionados ao trabalho. Ainda de acordo com o manual, são um grupo heterogêneo de distúrbios funcionais e/ou orgânicos que apresentam características, como:

- Fadiga neuromuscular causada por trabalho estático (posição fixa) ou com movimentos repetitivos.
- Quadros de dores, formigamento, dormência, choque, sensação de peso e fadiga
- Presença de quadros de tendinite, peritendinite, sinovite, principalmente nos ombros, cotovelos, punhos e mãos.

Alguns fatores de risco estão comumente relacionados ao surgimento dos males abordados, dentre os quais podem ser citados:

- Repetitividade
- Esforço e força
- Posturas inadequadas
- Trabalho muscular estático
- Invariabilidade da tarefa
- Choques e impactos
- Pressão mecânica
- Vibração
- Frio

- Fatores organizacionais

Complementando os fatores de risco, existe uma classificação de três moduladores que, combinados com os fatores de risco, caracterizam a aparição das lesões. São eles:

- Intensidade
- Duração
- Frequência

A repetitividade mostra-se como fator mais comumente referenciado para o acontecimento de LER/DORT. No entanto, não é o único fator determinante, já que a LER/DORT também é bastante associada a cargas de trabalho e posturas estáticas. Ainda que a repetitividade não seja, por si só, fator determinante para a ocorrência de LER/DORT, sua associação com outros fatores de risco e outras variáveis faz com que seus efeitos sejam aumentados.

O ciclo de operações de trabalho realizado diz respeito à quantidade de tarefa realizada em uma determinada faixa de tempo. Ciclos muito curtos podem sinalizar a realização de muitas operações em pouco tempo, implicando maior esforço muscular e articular. Ciclos mais longos podem indicar a realização de mais operações do que em ciclos curtos, resultando, da mesma forma, em esforço das estruturas física citadas. Ainda que as operações de trabalho possam ser variadas, não há garantia de que não ocorrerá sobrecarga musculoesquelética, já que ao fim da jornada de trabalho tais operações somam-se.

A ocorrência de LER/DORT pode ser acentuada se não houver tempo adequado de recuperação ao organismo do indivíduo. Luttmann et al. (2003) diz que é necessário equilíbrio entre a carga de trabalho exercido e a carga de trabalho suportada, visto que tal sobrecarga mecânica em músculos, tendões e ossos pode ser fator para a ocorrência das desordens físicas abordadas.

Para Guimarães et al. (2011), jornadas intensas de trabalho com uso do computador sujeitam trabalhadores a distúrbios musculoesqueléticos. Arnold et al (2000) também cita que longos períodos de trabalho de digitação têm sido relacionados à traumas físicos. Dentre os transtornos musculoesqueléticos mais comuns está a Síndrome do Túnel do Carpo (STC), descrita por Burton et al. (2014) como uma neuropatia sintomática compressiva do nervo mediano do punho.

Um estudo feito por Ali et al. (2006), demonstrou que a síndrome do túnel do carpo se apresentou em uma taxa de 13.1% dentre 648 profissionais de TI escolhidos em 21 companhias de tecnologia. O estudo revelou ainda que indivíduos com mais de oito anos de trabalho com

computadores, com média diária de trabalho de 12 horas, apresentavam maiores risco de contrair a síndrome.

2.2 Aplicações WEB

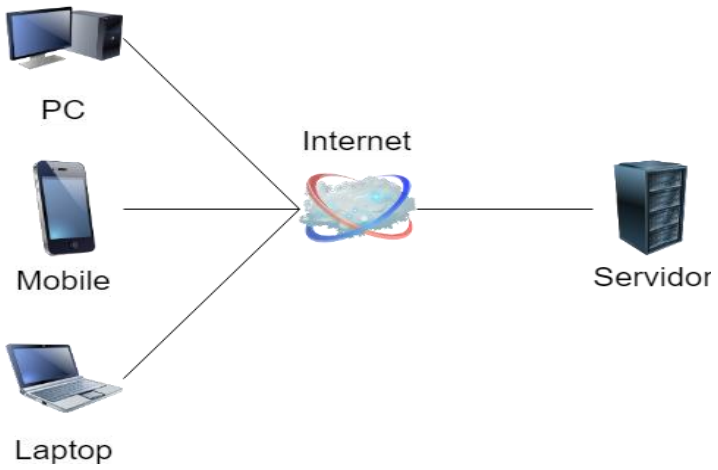
Aplicação web pode ser definida como qualquer programa de computador que execute alguma função por meio de um navegador web como cliente. Uma aplicação web permite que o processo de informações seja iniciado remotamente em um navegador web e executado em um servidor web, servidor de aplicação ou servidor de banco de dados (BRUNO, TAM e THOM, 2005).

Grande parte das aplicações web são baseadas em arquiteturas cliente-servidor, onde o cliente insere informações e gera requisições, e o servidor manipula e armazena essas informações, além de gerar respostas ao cliente. Por cliente podemos entender como qualquer aplicação que possa se comunicar com um servidor.

Para que a comunicação entre cliente e servidor seja possível é necessário o uso de regras, ou protocolos, como o HTTP (*Hypertext Transfer Procotol*), ou o FTP (*File Transfer Protocol*). Em se tratando de aplicações web, o protocolo HTTP é o mais utilizado na comunicação cliente-servidor.

Os clientes em um ambiente web, costumam se comunicar com servidores para consumir recursos presentes no mesmo. Um recurso web diz respeito a qualquer documento ou informação armazenada em um servidor web. Além disso, os recursos podem ser classificados como estáticos ou dinâmicos. Os recursos estáticos não costumam variar, e são entregues ao cliente na sua forma “in natura” a como estava armazenada no servidor. Já recursos dinâmicos são construídos conforme a requisição do cliente. Imaginando que um cliente deseje um arquivo XML contendo dados como data e hora da operação realizada, o servidor então manipula este arquivo e devolve ao cliente um dado dinâmico contendo as informações desejadas, específicas àquela requisição. A Figura 1 apresenta uma representação simplificada da arquitetura cliente-servidor.

Figura 1 - Arquitetura simplificada cliente-servidor



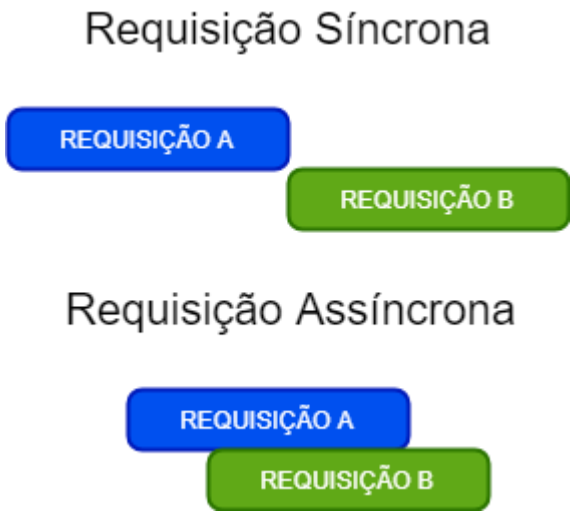
Fonte: Autor (2019)

Aplicações web, normalmente, combinam o uso de scripts do lado do servidor com scripts do lado do cliente. O lado cliente lida com a apresentação das informações ao usuário, enquanto que o lado do servidor é responsável pelo processamento pesado dos dados, como o armazenamento e retorno de informações. Este processo de requisição e resposta entre cliente e servidor é chamado de transação HTTP.

Dentre os benefícios do uso de aplicações web, destaca-se a sua capacidade multiplataforma, visto que o desenvolvedor não precisa se preocupar em construir as aplicações de maneira específica para determinado tipo de computador ou sistema operacional. A partir do momento em que o cliente utilize um navegador web, será possível acessar a aplicação não importando se o ambiente é Windows ou Linux, por exemplo.

Outra característica de aplicações web é a possibilidade de existência de requisições síncronas ou assíncronas. Este conceito é, geralmente, relacionado ao lado do cliente. Nas requisições síncronas o cliente aguarda a resposta do servidor, culminando num processo de requisição, espera e obtenção da resposta. Já em requisições assíncronas, quando o cliente faz uma requisição, ele não necessita aguardar a resposta do servidor, fato que não lhe impede de realizar outras tarefas simultaneamente. Assim, a aplicação não “trava” durante alguma transferência de arquivos grandes, por exemplo. A Figura 2 ilustra a diferença entre requisições síncronas e requisições assíncronas.

Figura 2 - Requisição síncrona X Requisição assíncrona



Fonte: Autor (2019)

2.3 Tecnologias *Front-End*

Front-end, no contexto de aplicações web, diz respeito ao ambiente *web* do lado cliente, composto, tipicamente, pelo uso de tecnologias como HTML, CSS e JavaScript, apresentadas com mais detalhes na próxima seção. É por meio do *front-end* que ocorrem as interações propriamente ditas com os usuários das aplicações.

2.3.1 HTML, CSS e JavaScript

HTML (*Hypertext Markup Language*) diz respeito à linguagem de marcação de hipertexto. A linguagem é estruturada com elementos chamados *tags*, delimitados pelos sinais < > e </>, que definem o conteúdo e função dos componentes da linguagem.

Com o HTML é possível estruturar documentos contendo títulos, parágrafos, listas, tabelas, formulários dentre diversos outros elementos que permitem a integração de imagens, animações ou vídeos (FLATSCHART, 2011). O escopo básico do HTML é comporto por três tags fundamentais: <*html*>, <*head*> e <*body*>.

A primeira parte, correspondendo a *tag* <*html*>, diz respeito ao elemento raiz do documento que atua como container para todos os demais elementos HTML, com exceção de <!DOCTYPE html>, que apenas indica ao navegador a versão do documento em questão. A segunda parte, *tag* <*head*>, caracteriza a primeira *tag* usada em um documento HTML, contendo outras *tags* que dizem ao navegador informações como o nome da página (*tag*

<*title*>), metadados (*tag* <*meta*>), como informações sobre a codificação de caracteres utilizada na página ou o layout padrão de exibição, além de poder referenciar documentos externos de estilo e script (*tags* <*link*> e <*script*>, respectivamente). Por fim, a *tag* <*body*> se apresenta como a *tag* corpo do documento, contendo praticamente todo o conteúdo apresentado ao usuário, como textos, botões, formulários, menus, listas ou tabelas. A Figura 3 ilustra o escopo básico de um documento HTML.

Figura 3 - Escopo básico HTML


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Escopo HTML</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <h1>Hello, World!</h1>
    <script src="script.js"></script>
  </body>
</html>
```

Fonte: Autor (2019)

CSS (*Cascading Style Sheet*), Folha de estilo em cascata, é uma linguagem responsável por definir a formatação e estilo do conteúdo em páginas HTML, como por tamanhos de fonte, cores ou posicionamento dos elementos, ou seja, como o conteúdo deve ser renderizado na página (GOODMAN, 2007).

JavaScript é uma linguagem de programação de scripting multiplataforma orientada a objetos que pode ser integrada a páginas HTML, com o objetivo de incorporar interatividade, como animações complexas, eventos de cliques em botões, dentre outros componentes. A linguagem possui uma biblioteca padrão de objetos de estruturas de dados como *Array*, além de objetos para tratamento de datas (*Date*) e manipulação de elementos matemáticos (*Math*), bem como outros elementos como operadores lógicos, estruturas de controle e declarações.

Cada uma das três tecnologias apresentadas atua como uma camada com responsabilidade própria no contexto de criação de páginas web, conferindo modularidade na construção de páginas no lado do cliente. Enquanto o HTML se encarrega de marcar o

conteúdo, o CSS apenas se responsabiliza em como este conteúdo se apresenta ao usuário, seja em termos de layouts, cores ou fontes, por exemplo. Por fim, o JavaScript adiciona interatividade e controle aos elementos HTML, além de permitir a manipulação do CSS dinamicamente.

2.4 Ambientes de Desenvolvimento para WEB

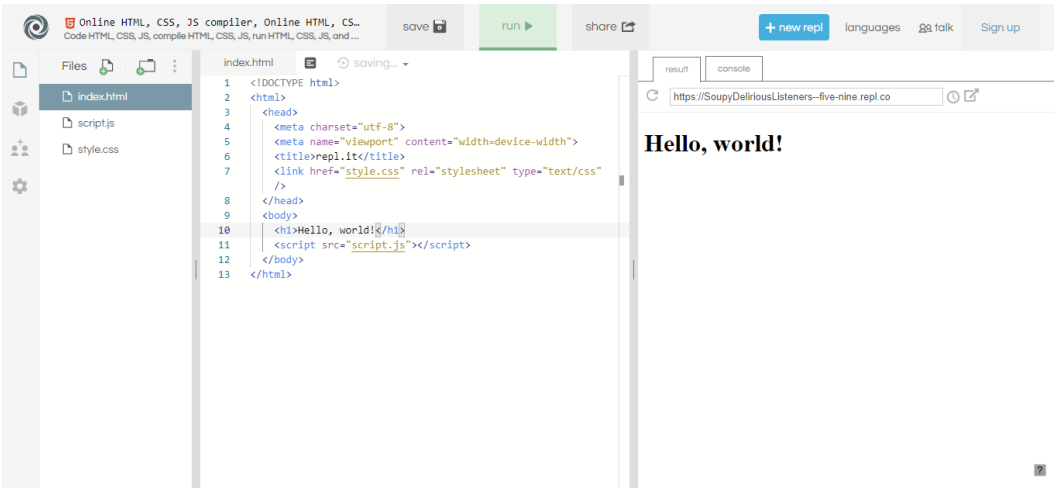
Ambientes de desenvolvimento existem para prover condições mais eficazes à concepção de componentes de software. Em geral, utilizam-se como ambientes editores de texto simples ou IDE (Ambientes de Desenvolvimento Integrado). Classifica-se IDE como um programa de computador utilizado para aumentar a produtividade de desenvolvedores de software. Os ambientes de desenvolvimento costumam apresentar um conjunto de ferramentas que vão desde editores de código fonte, compiladores, depuradores, a ferramentas para modelagem, testes e refatoração de código.

Com o avanço das tecnologias web e a disponibilidade de serviços de cloud, os ambientes para desenvolvimento têm sido providos totalmente online, facilitando o acesso e integração de equipes à projetos, e eliminando a necessidade de configurações locais dos ambientes. A seguir, são apresentados alguns exemplos de ambientes de desenvolvimento para ambiente web.

- Repl.it [1]– Este ambiente provê recursos para desenvolvimento em diversas linguagens de programação disponíveis para uso gratuitamente. Sua interface de programação principal consiste em um editor de código fonte, além de funcionalidades para adicionar, salvar e excluir arquivos, bem como uma região para exibir os resultados da execução do código fonte, conforme captura de tela exibida na Figura 4.

[1] Disponível em: https://repl.it/

Figura 4 – Captura de tela da interface gráfica da ferramenta Repl.it



Fonte: repl.it (2019)

- Playcode [2]– A ferramenta possui uma interface com funcionalidades de edição de código fonte, exibição de resultado, além da manipulação de arquivos e capacidade de integração com bibliotecas externas. Seu foco consiste em manipulação de código front-end ao permitir a manipulação primária de arquivos HTML, CSS e JavaScript. A Figura 5 apresenta uma captura de tela da ferramenta.

Figura 5 - Captura de tela da ferramenta PlayCode.io

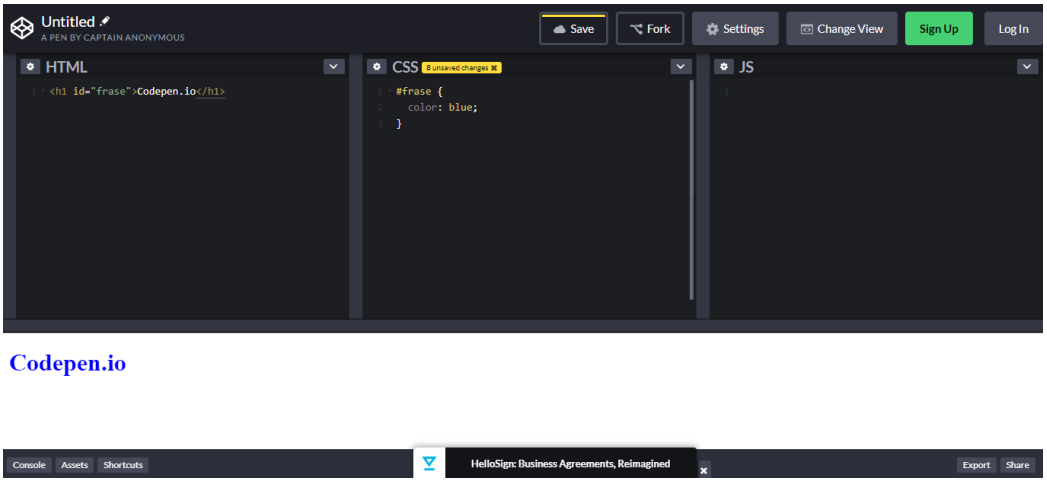
Fonte: playcode.io (2019)

- CodePen [3]– Assim como o PlayCode, o CodePen foca em desenvolvimento de páginas *front-end*, ao disponibilizar uma interface de programação simples, com campos para

[2] Disponível em: https://playcode.io/
[3] Disponível em: https://codepen.io/

edição de código HTML, CSS e JavaScript, de maneira intuitiva. A Figura 6 mostra uma captura de tela da ferramenta.

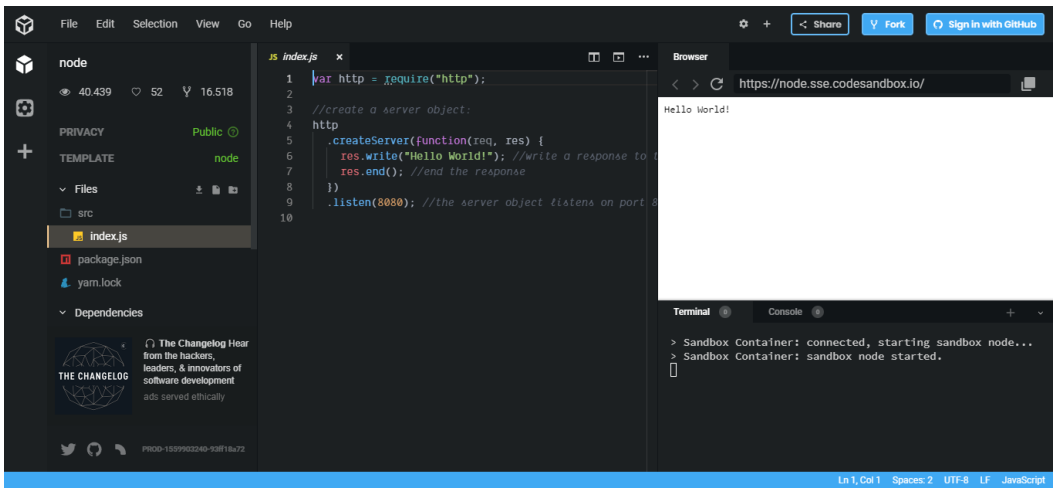
Figura 6 - Captura de tela da ferramenta CodePen



Fonte: codepen.io (2019)

- CodeSandBox [4]– O ambiente disponibilizado pela ferramenta CodeSandBox se assemelha à ferramenta Repl.it no tocante a ter uma proposta de desenvolvimento mais robusta ao permitir a manipulação de variados frameworks de desenvolvimento web, como Angular, Vue ou React. Apresenta uma interface gráfica dinâmica e com variadas funcionalidades de edição de código fonte, manipulação de arquivos da estrutura dos projetos, além de renderização do resultado do código criado, conforme pode ser observado na Figura 7.

Figura 7 - Captura de tela do ambiente CodeSandBox



Fonte: codesandbox.io (2019)

[4] Disponível em: https://codesandbox.io/

2.5 Reconhecimento de voz

Reconhecimento de voz diz respeito à tecnologia que torna possível a identificação de componentes da voz humana por computadores. De maneira geral e simplificada, o processo de reconhecimento pode ser descrito como tendo início pela captação de uma expressão por meio de um microfone e finalizado com o resultado das palavras reconhecidas sendo retornado pelo sistema (ENGLUND, 2004).

Ainda segundo Englund (2004), o processo de conversão da fala pode seguir diferentes abordagens, dentre as quais citam-se: Técnicas de correspondência de padrões, redes neurais e modelos ocultos de Markov.

A abordagem de correspondência de padrões consiste em comparar expressões de fala com templates pré-gravados de padrões acústicos que se relacionam com palavras de um vocabulário. Dessa forma é feita uma comparação entre a entrada de sinal de áudio da fala e os padrões armazenados, a fim de se determinar qual template apresenta maior similaridade com a palavra dita.

Redes neurais consistem em uma tentativa de modelar propriedades similares as do sistema nervoso humano (ENGLUND, 2004). As redes são formadas por nós organizados em camadas e interconectados com pesos de variadas forças. Seu funcionamento se dá a partir da entrada de dados pela camada inicial da rede, seguido pelo processamento de suas camadas internas, culminando no processamento pelas camadas finais da rede, responsáveis por retornar um resultado. A capacidade de classificação por uma rede neural depende da qualidade de seu treinamento. Em relação ao reconhecimento de voz, o treinamento pode se dar, por exemplo, a partir da entrada de um sinal acústico, como amplitudes do som a ser reconhecido, e ao final o valor de saída da rede é comparado a um valor esperado, como um fonema. Ainda que a abordagem de redes neurais seja promissora, em geral não são utilizadas isoladamente, mas integradas à sistemas baseados em modelos ocultos de Markov.

Modelos ocultos de Markov são um método estatístico para se modelar sinais de fala. Um modelo representa uma unidade de linguagem, como uma palavra ou fonema. O modelo possui uma quantidade limite de estados onde a transição entre estes ocorre uma vez a cada unidade de tempo. Cada estado tem como saída uma função probabilística que representa uma variável aleatória ou processos estocásticos (RABINER e JUANG, 1986).

2.5.1 API's para reconhecimento de voz

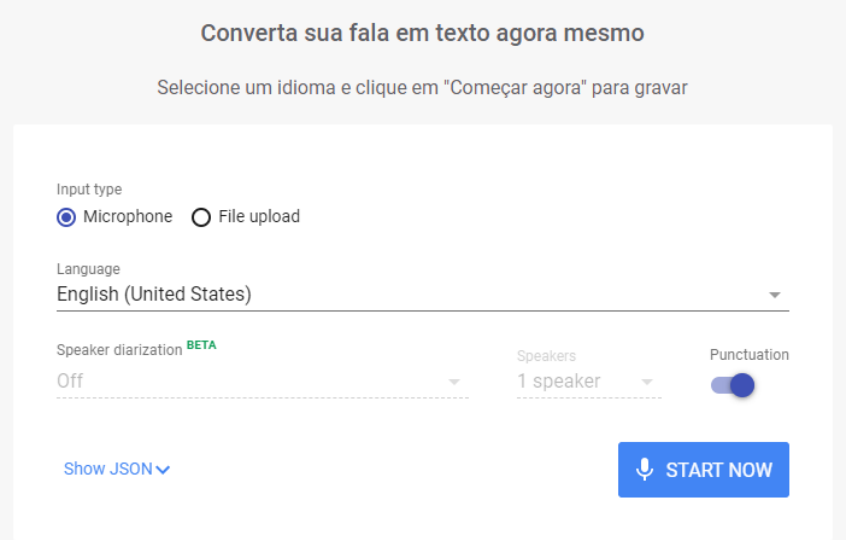
API's (*Application Programming Interfaces*) são como a porta de entrada para a comunicação entre clientes e serviços web, comumente expondo um conjunto de dados ou funcionalidades para que a integração entre programas de computador se faça possível (MASSE, 2011).

Várias são as API's de reconhecimento de voz existentes atualmente, devido o avanço das tecnologias e métodos utilizados para reconhecer sinais de áudio. Dentre as API's muito citadas por ter integração com ambientes web, estão a *Cloud Speech to Text* [5], da Google, a *Watson Speech to Text* [6], da IBM, e a *Web Speech API* [7], API de uso aberto mantida pelo W3C (*World Wide Web Consortium*).

A API de reconhecimento de fala da Google conta com a capacidade de reconhecer mais de 120 idiomas e variantes, além de poder processar streamings em tempo real ou áudio pré-gravado utilizando tecnologias de aprendizado de máquina. Pode-se utilizar uma ferramenta presente no próprio site da API. Com uma interface simples e de fácil uso, é possível selecionar qual o tipo de entrada de dados desejada (via Microfone ou envio de arquivo de áudio), bem como o idioma desejado ao reconhecimento. A Figura 8 apresenta captura de tela da interface da ferramenta de demonstração de uso da API.

[5] Disponível em: https://cloud.google.com/speech-to-text/?hl=pt-br
[6] Disponível em: https://www.ibm.com/watson/services/speech-to-text/
[7] Disponível em: https://w3c.github.io/speech-api/

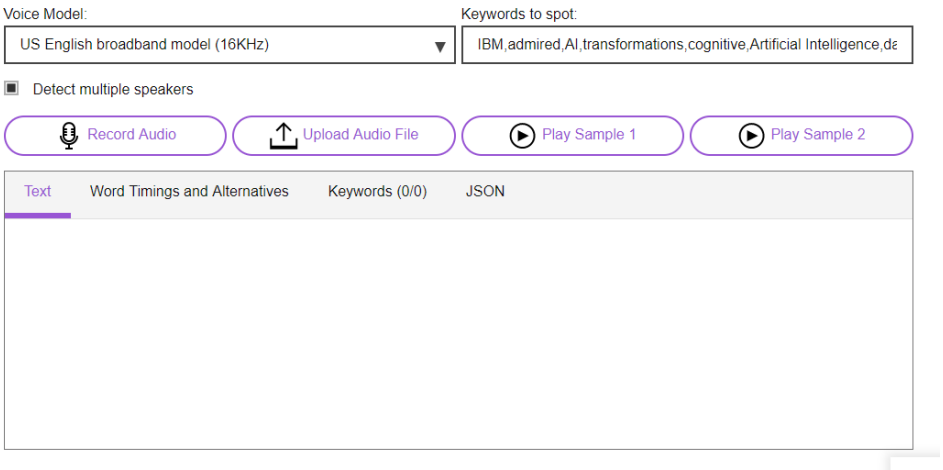
Figura 8 – Captura de tela da interface de demonstração da API *Cloud Speech-to-Text*



Fonte: cloud.google.com (2019)

A API de reconhecimento de fala da IBM conta com suporte para transcrição em tempo real para sete idiomas, por meio de uma variedade de interfaces de programação (HTTP REST, Websocket, HTTP Assíncrono). Existe uma página de testes disponível no site da ferramenta. A ferramenta possui uma interface de uso simples e intuitivo, permitindo a configuração do modelo de idioma a ser utilizado no reconhecimento, assim como realizar o reconhecimento em tempo real ou a partir do envio de arquivo de áudio pré-gravado. A Figura 9 apresenta uma captura de tela da interface de demonstração da API IBM Watson.

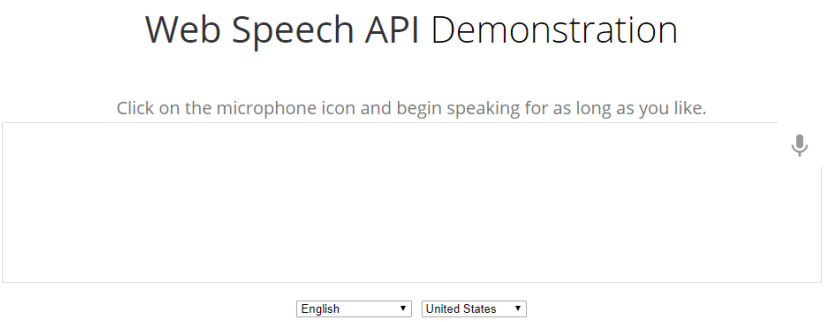
Figura 9 - Captura de tela da interface de demonstração da API IBM Watson Speech to text



Fonte: ibm.com (2019)

A *Web Speech API* visa permitir que desenvolvedores possam integrar em browsers características de reconhecimento de voz para texto, assim como a síntese de áudio a partir de texto. É possível fazer o uso de uma página de testes disponibilizada pelo Google. A ferramenta é bem simples, apenas possuindo opções para definir o idioma base do reconhecimento de voz, além de uma área de texto para mostrar o resultado da transcrição. A Figura 10 demonstra a interface da API Web Speech, a partir de uma captura de tela da aplicação.

Figura 10 – Captura de tela da interface de demonstração da API Web Speech



Fonte: google.com/chrome/demos/speech.html (2019)

3 Trabalhos Relacionados

Neste capítulo serão abordados trabalhos relacionados que apresentam similaridades ao contexto do que é proposto.

Leopold e Ambler (1997) propuseram um trabalho que aborda a programação visual por meio de voz, escrita manual e gestos sem o uso de teclado. Segundo os autores, muitos ambientes de desenvolvimento, ainda que visuais, são restritos pelo uso de teclado e mouse para a entrada de dados. Dessa forma, buscaram como alternativa estudar a aplicação de meios de interação mais naturais ao ser humano, como o uso da voz e gestos, pois acreditavam que o uso de interfaces de usuário multimodais teriam potencial para aumentar a produtividade em ambiente de programação não apenas para pessoas com alguma deficiência física, que lhes impossibilitasse utilizar um ambiente "comum", mas para um público geral. A proposta foi a de desenvolver uma interface multimodal de usuário para *Formulate* (Linguagem visual baseada em formulários) com o objetivo de melhorar a habilidade de visualizar, editar e interagir com programas visualmente orientados utilizando-se da combinação voz e caneta em vez de mouse e teclado.

Arnold et al. (2000) propuseram a construção de um gerador para ambientes de programação por reconhecimento de voz dirigidos à sintaxe. Tais ambientes tomariam vantagem da gramática formal de linguagens de programação para simplificar o processo de programação por meio da voz. O gerador proposto, denominado VocalGenerator, toma como entrada de dado uma gramática livre de contexto (CFG) de uma linguagem de programação, como por exemplo Java, C, C++ ou XML DTD, junto com um vocabulário de voz para aquela linguagem. Gramática livre de contexto (*Context-free grammar* - CFG), ou simplesmente gramática, define um conjunto finito de regras de produção que representam a definição recursiva de uma língua (HOPCRAFT, MOTWANI e ULLMAN, 1979). O vocabulário utilizado no gerador inclui literais de uma linguagem de programação, assim como nomes de classes e bibliotecas de funções disponíveis para o programador. Esses então são associados com uma lista de palavras pronunciáveis e frases que o programador usaria para editar o programa na linguagem. Dessa forma, como resultado, é gerado um ambiente de programação que permite o desenvolvimento por meio da voz.

Yui e Carro (2016) propuseram o uso de reconhecimento de voz para auxiliar a edição de código fonte em linguagem de programação Java. Seu trabalho consistiu em criar uma interface de software que auxiliasse pessoas com deficiências motoras a programar por meio do reconhecimento sintático de sentenças na linguagem de programação em questão. A partir de

pesquisa e testes de API's e ferramentas, foi desenvolvida uma aplicação em Java, denominada FJE (Fala Java Editor), que utiliza a API *Microphone Capture* para a captura de voz, o Google *Duplex Speech* API para o reconhecimento de linguagem natural.

Na Pycon US de 2013, Tavis Rudd, em sua apresentação *Using python to Code by Voice*, demonstrou o uso de Python para codificar por voz. Sua ideia se deu após ter ficado impossibilitado de trabalhar e fazer uso de dispositivos como mouse e teclado, devido a lesões físicas por esforço repetitivo. Tavis criou um pequeno sistema usando linguagem de programação Python, integrando-a com um software comercial de reconhecimento de fala para controle de aplicações chamado *Dragon Naturally Speaking*, emulado em um sistema Linux. Por meio do mapeamento de palavras monossilábicas, e integração com o editor de texto Emacs, demonstrou seu uso a partir de códigos simples implementados em linguagem LISP.

Em julho de 2016, David Williams-King apresentou na conferência *The Eleventh HOPE* uma palestra demonstrando um software, chamado Silvius, baseado na criação de Tavis Rudd, onde também é possível a programação por comando de voz. Assim como na aplicação de Tavis Rudd, David usou como base o software *Dragon Naturally Speaking*, integrado ao sistema Aenea, de Alex Roper, onde uma máquina virtual Windows, emulada em um sistema Linux, utiliza o framework *Dragonfly* e bibliotecas de python para converter entradas de fala em resultados que simulam a digitação no teclado físico.

3.1 Comparação entre as ferramentas

O Quadro 1 contém comparações das principais características dos trabalhos apresentados neste capítulo. Observa-se que o protótipo desde trabalho apresenta características que o diferencia dos demais trabalhos apresentados, como o fato de ser disponibilizado em plataforma web, fato que permite a facilidade de acesso. A usabilidade é outro ponto de destaque, já que a interação por meio de comandos de voz é facilitada em relação às ferramentas como a de Tavis Rudd, por apresentar comandos de voz que abstraem mais a escrita de código fonte, permitindo uma aproximação mais natural. A ferramenta deste trabalho foi desenvolvida utilizando-se de ferramentas e tecnologias gratuitas, permitindo o livre uso e eliminando a necessidade de compra de licenças e ferramentas proprietárias.

Quadro 1 – Comparação de características entre ferramentas de trabalhos relacionados

Características/Trabalhos	**VocalGenerator**	**FJE**	**Tavis Rudd *Code by Voice***	**Silvius**	**Protótipo desde trabalho**
Uso gratuito	não	sim	não	sim	sim
Ambiente web	não	não	não	não	sim
Facilidade de uso	não	sim	não	não	sim
Código aberto	não	não	não	sim	sim

Fonte: Autor (2019)

4 Proposta do anteprojeto

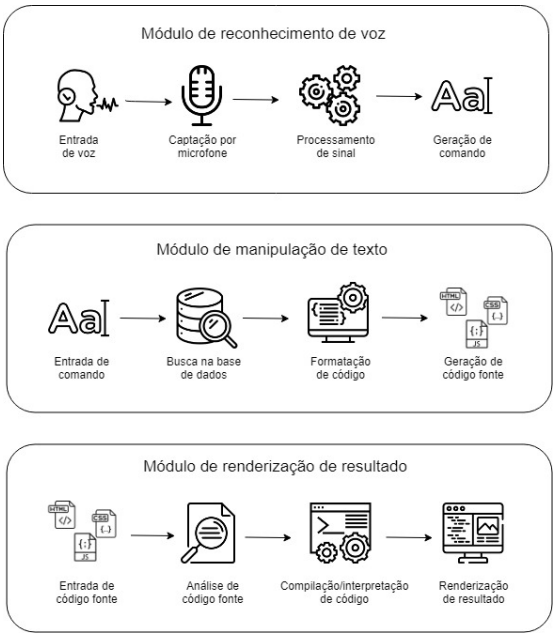
Este trabalho se propôs a desenvolver um ambiente protótipo de programação para a geração de código *front-end* de aplicações web que aceite de comandos de voz como forma de entrada de dados. Neste trabalho, o escopo foi limitado a permitir a criação de código HTML e CSS. Para alcançar o objetivo, foi proposta uma arquitetura modular visando dividir as responsabilidades previstas para o sistema, de modo a deixá-lo mais eficiente e com menor acoplamento.

4.1 Arquitetura Proposta

A arquitetura para a criação da ferramenta web de programação por comandos de voz proposta neste trabalho está organizada em 3 módulos básicos: Módulo de reconhecimento de voz, Módulo de manipulação de texto e Módulo de renderização de resultado. Cada módulo possui responsabilidades específicas que se integram para gerar uma solução.

Além disso, os módulos se estruturam em cadeia, interligando-se por meio da passagem de resultados de um módulo para outro ao longo de sua estrutura. As responsabilidades dos módulos são divididas em reconhecimento de voz, manipulação de texto e renderização de resultado. A Figura 11 apresenta a representação gráfica da arquitetura com os três módulos integrados.

Figura 11 – Módulos da arquitetura proposta



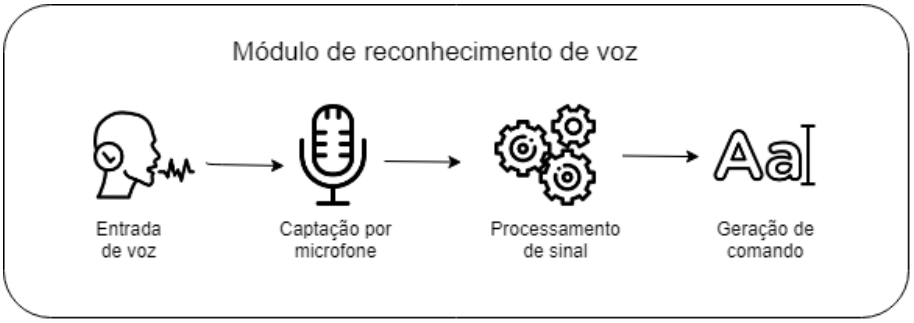
Fonte: Autor (2019)

Nos tópicos a seguir são apresentados, individualmente, os módulos da arquitetura proposta.

4.1.1 Módulo de reconhecimento de voz

O primeiro módulo presente no sistema é responsável por fazer o reconhecimento dos sinais de áudio captados por um microfone, e convertê-los em texto, para que se possa fazer o processamento no módulo seguinte. O processamento da voz pode ser feito por bibliotecas de reconhecimento de fala existentes, como a Web Speech API, selecionada para este trabalho, ou mesmo ser criada especificamente para este fim. A Figura 12 apresenta a representação gráfica das etapas pertinentes a este módulo.

Figura 12. Organização do módulo de reconhecimento de voz

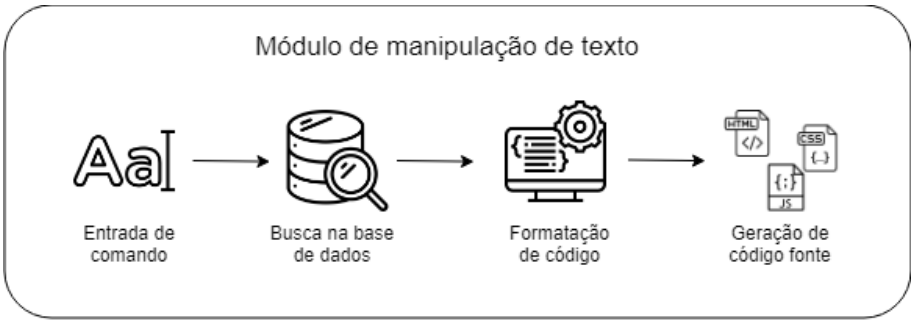


Fonte: Autor (2019)

4.1.2 Módulo de manipulação de texto

Após o processamento de voz, gerando como resultado um comando de texto, entra em ação o módulo de manipulação de texto. Seu funcionamento consiste em verificar o comando recebido dentre as bibliotecas de comando ou banco de dados, a fim de identificar a estrutura de código relacionada ao comando. Em caso de sucesso, o módulo realiza a estrutura do código relacionado ao comando, dentro do editor de texto, formatando-o dinamicamente em meio ao uso do sistema. Como resultado, o módulo pode produzir um arquivo de código fonte para uso na terceira camada da arquitetura. A Figura 13 apresenta a representação gráfica das etapas pertinentes a este módulo.

Figura 13. Organização do módulo de manipulação de texto

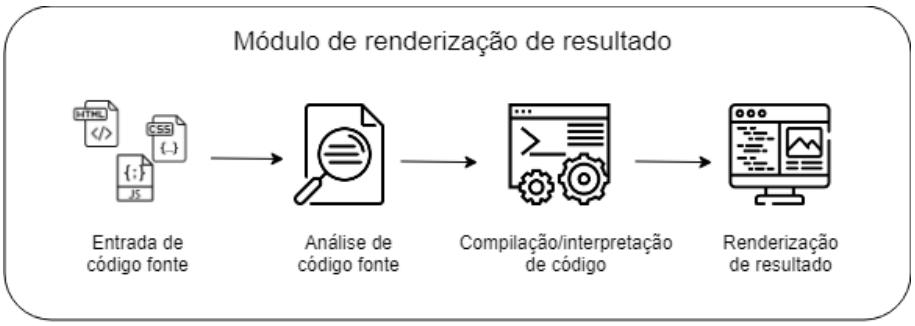


Fonte: Autor (2019)

4.1.3 Módulo de renderização de resultado

Como parte do último estágio, o módulo de renderização de resultado se utiliza do código fonte gerado pelo módulo anterior para apresentar ao usuário a conclusão do seu processo de programação. O módulo, inicialmente, verifica o arquivo para identificar qual é a linguagem ou o mecanismo de programação em questão. Em seguida, procede para a compilação ou interpretação do código presente no arquivo. A Figura 14 apresenta a representação gráfica das etapas pertinentes a esse módulo.

Figura 14. Organização do módulo de renderização de resultado.



Fonte: Autor (2019)

4.2 Especificação dos requisitos

Nesta seção serão apresentados os diagramas de casos de uso e de atividades que foram elaborados para orientar o desenvolvimento da aplicação proposta.

4.2.1 Requisitos Funcionais

No Quadro 2 são apresentados os requisitos funcionais definidos para o presente trabalho.

Quadro 2 – Requisitos funcionais

Código	Requisito	Descrição	Prioridade
RF01	Gerar Código	O sistema deve permitir que o usuário gere código fonte	Essencial
RF02	Utilizar microfone	O sistema deve ter acesso à entrada de áudio por meio de um microfone.	Essencial
RF03	Reconhecer comandos de voz	O sistema deve ser capaz de reconhecer comandos de voz inseridos pelo usuário por meio de um microfone	Essencial
RF04	Converter comandos em código	O sistema deve converter os comandos de voz inseridos pelo usuário em código fonte exibido em um editor de código	Essencial
RF05	Renderizar código	O sistema deve permitir que o usuário visualize o resultado do código fonte escrito	Importante

Fonte: Autor (2019)

4.2.2 Requisitos não funcionais

O Quadro 3 apresenta os requisitos não funcionais definidos para orientar o desenvolvimento do projeto proposto.

Quadro 3 – Requisitos não funcionais

Código	Requisito	Descrição	Prioridade
RNF01	Ambiente web	O sistema deve ser disponibilizado em ambiente web, a partir do uso de um navegador	Essencial
RNF02	Início rápido de geração de código	O sistema deve permitir que o usuário inicie o processo de programação a partir de poucas interações por meio de mouse, limitando-se a, no máximo, três cliques, ou via atalho de teclado.	Importante
RNF03	Responsividade	O sistema deve proporcionar responsividade à página da aplicação, permitindo que os componentes gráficos sejam adaptados conforme redimensionamento de tela feito pelo usuário.	Importante

Fonte: Autor (2019)

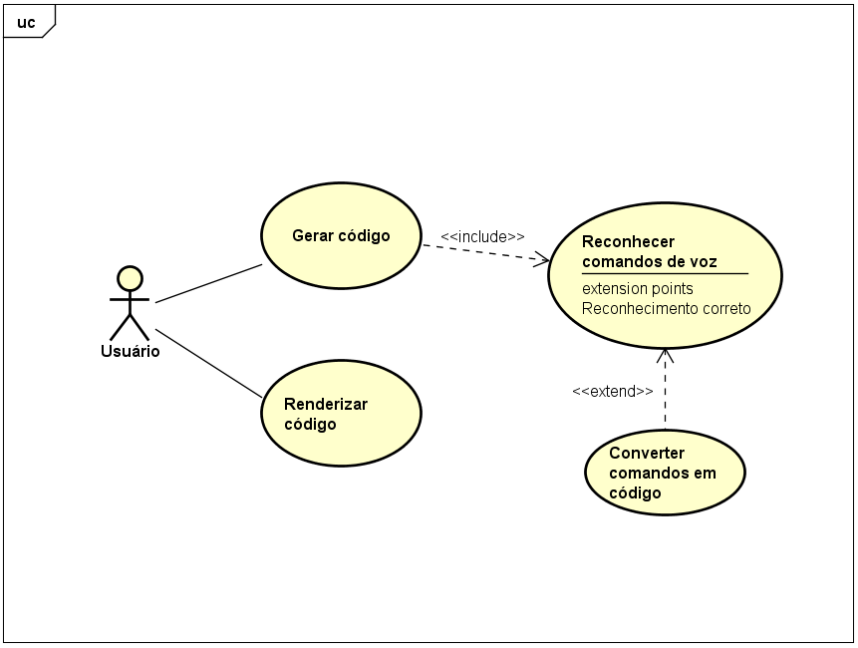
4.3 Modelagem do sistema

Nesta seção serão apresentados os diagramas de casos de uso e de atividades que foram elaborados para orientar o desenvolvimento da aplicação proposta.

4.3.1 Diagrama de casos de uso

O sistema proposto tem um ator, o usuário, responsável por manipular, diretamente, as funcionalidades de geração de código, bem como renderizar o resultado do código construído. Internamente, no processo de gerar código, é incluída a funcionalidade do módulo de reconhecimento de voz que, por sua vez, pode ou não utilizar a funcionalidade de conversão de comandos em código fonte, de acordo com o resultado do reconhecimento de voz. A Figura 15 apresenta o diagrama de casos de uso proposto para este trabalho. As descrições dos casos de uso encontram-se nos anexos deste trabalho.

Figura 15 - Diagrama de casos de uso

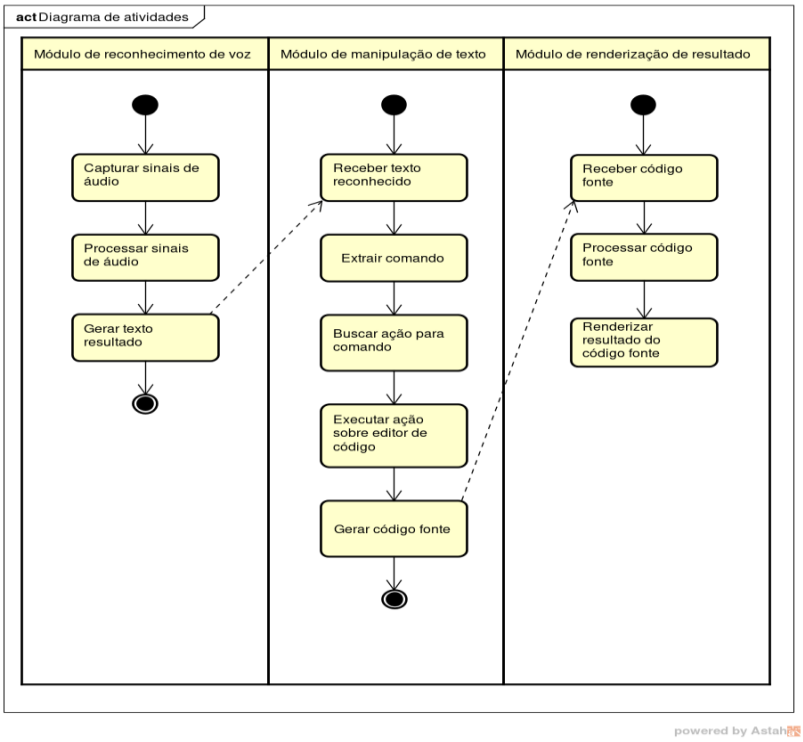


Fonte: Autor (2019)

4.3.2 Diagrama de atividades

O diagrama de atividades, conforme ilustrado na Figura 16, apresenta a ordem com que as atividades ocorrem no sistema, baseado nos módulos da arquitetura.

Figura 16 - Diagrama de atividades



Fonte: Autor (2020)

5 Experimentos

De modo a selecionar a API de reconhecimento de voz que melhor se adequa às necessidades deste trabalho, foram realizados experimentos com algumas API's de reconhecimento de voz e transcrição para texto para verificar sua capacidade de atender a demanda de transcrição de áudio. Para os experimentos foram selecionadas as API's: Google *Cloud Speech-to-Text* e IBM Watson *speech to text* e *WEB Speech* API. Este capítulo descreve os experimentos realizados com as API's selecionadas.

5.1 Experimentos de reconhecimento de voz

Para a realização dos experimentos foi definido um conjunto de palavras relacionadas à temática do projeto. Os experimentos se deram por meio da entrada de áudio diretamente na ferramenta testada, mediante o uso de um microfone, seguido pela verificação da resposta da API, documentação de seus resultados, concluindo pela análise e comparação com os resultados das demais ferramentas analisadas. Para cada palavra, foi realizada uma sequência de cinco testes de fala, variando-se a tonalidade e velocidade de pronúncia, a fim de se obter melhores análises das API's. No Quadro 4, são relacionadas 20 palavras utilizadas no contexto do desenvolvimento *front-end*, mais especificamente em HTML, para a realização dos experimentos das API's de reconhecimento de voz, de forma a criar um cenário similar ao que será utilizado na ferramenta proposta neste trabalho.

Quadro 4 – Lista de palavras selecionadas para os testes

Nº	Palavra	Nº	Palavra
1	Web	11	Valor
2	HTML	12	Título
3	Tabela	13	Nome
4	Parágrafo	14	Lista
5	Texto	15	Campo
6	Tag	16	Função
7	Linha	17	Estilo
8	Abrir	18	Comando
9	Fechar	19	Salvar
10	Executar	20	JavaScript

Fonte: (Autor, 2019)

5.1.1 Google Cloud Speech-to-Text

O Quadro 5 apresenta a descrição de algumas siglas usadas nas tabelas dos resultados dos experimentos realizados com as APIs.

Quadro 5 - Descrição de siglas usadas nas tabelas de resultados dos experimentos

SIGLA	DESCRIÇÃO
T1	Teste 1
T2	Teste 2
T3	Teste 3
T4	Teste 4
T5	Teste 5

Fonte: Autor (2019)

O Quadro 6 apresenta os registros dos experimentos realizados com a API *Cloud Speech-to-text*, contendo as palavras e resultados de cada um dos 5 experimentos realizados.

Quadro 6 - Resultados dos experimentos com palavras da API *Cloud Speech-to-Text*

n°	Palavra	T1	T2	T3	T4	T5
1	Web	Correto	Correto	Correto	Correto	Correto
2	HTML	Correto	Correto	Correto	Correto	Correto
3	Tabela	Correto	Correto	Correto	Correto	Correto
4	Parágrafo	Correto	Correto	Correto	Correto	Correto
5	Texto	Correto	Correto	Correto	Correto	Correto
6	Tag	Correto	Correto	Correto	Correto	Correto
7	Linha	Correto	Correto	Correto	Correto	Correto
8	Abrir	Correto	Correto	Correto	Correto	Correto
9	Fechar	Errado – Retornou “baixar”	Correto	Errado – Retornou “fechou”	Correto	Correto
10	Executar	Correto	Correto	Correto	Correto	Correto
11	Valor	Correto	Correto	Correto	Correto	Correto
12	Título	Correto	Correto	Correto	Correto	Correto
13	Nome	Correto	Correto	Correto	Correto	Correto
14	Lista	Correto	Correto	Correto	Correto	Correto
15	Campo	Correto	Correto	Correto	Correto	Correto
16	Função	Correto	Correto	Correto	Correto	Correto

17	Estilo	Correto	Correto	Errado – Retornou “Assistir”	Correto	Correto
18	Comando	Correto	Correto	Correto	Correto	Correto
19	Salvar	Correto	Correto	Correto	Correto	Correto
20	JavaScript	Errado – Retornou “Já vai ser”	Correto	Correto	Correto	Correto

Fonte: Autor (2019)

O Quadro 7 apresenta a taxa de acertos dos experimentos realizados com a API *Cloud Speech-to-Text*, identificando a quantidade de itens corretos dentre o total de palavras previstas, bem com uma média geral de aproveitamento.

Quadro 7 - Acertos dos experimentos com palavras da API *Cloud Speech-to-Text*

Item	Sigla	Acertos	Total de itens	Porcentagem
Teste 1	T1	18	20	90%
Teste 2	T2	20	20	100%
Teste 3	T3	18	20	90%
Teste 4	T4	20	20	100%
Teste 5	T5	20	20	100%
Média	-	19,2	-	96%

Fonte: Autor (2019)

Os experimentos demonstraram um bom desempenho alcançado pela API, embora tenha apresentado algumas variações e pequenos erros para reconhecer certas palavras como “JavaScript” e “Fechar”. A taxa de precisão de 96% alcançada no decorrer dos experimentos mostrou que a API pode ser uma boa candidata aos propósitos do projeto. Contudo, a ferramenta apresenta o contraponto de não ser disponibilizada gratuitamente. É necessário que haja um cadastro na plataforma e existe uma tabela de valores cobrados de acordo com o tempo de uso da API.

5.1.2 IBM Watson Speech to text

O Quadro 8 apresenta os registros dos experimentos realizados com a API IBM Watson *Speech to text*, contendo as palavras e resultados de cada um dos 5 experimentos realizados.

Quadro 8 - Resultados dos experimentos com palavras da API IBM Watson *Speech to text*

n°	Palavra	T1	T2	T3	T4	T5
1	Web	Errado – Retornou “prédio”	Errado – Retornou “reggae”	Errado – Retornou “a hebe”	Errado – Retornou “webb”	Errado – Retornou “a média”
2	HTML	Correto	Correto	Correto	Correto	Correto
3	Tabela	Correto	Correto	Correto	Correto	Correto
4	Parágrafo	Correto	Correto	Correto	Correto	Correto
5	Texto	Correto	Correto	Errado – Retornou “o texto”	Correto	Errado – Retornou “três de”
6	Tag	Errado – Retornou “pegue”	Errado – Retornou “pegue”	Errado – Retornou “pegue”	Errado – Retornou “peggy”	Errado – Retornou “pegue”
7	Linha	Correto	Correto	Correto	Correto	Correto
8	Abrir	Correto	Correto	Correto	Correto	Correto
9	Fechar	Errado – Retornou “já”	Errado – Retornou “para fechar”	Errado – Retornou “relaxar”	Correto	Correto
10	Executar	Correto	Correto	Correto	Correto	Correto
11	Valor	Errado – Retornou “o valor”	Correto	Correto	Errado – Retornou “valor da”	Correto
12	Título	Correto	Correto	Correto	Correto	Correto
13	Nome	Errado – Retornou “nome de cor”	Errado – Retornou “nome de”	Errado – Retornou “não de”	Errado – Retornou “nota de”	Errado – Retornou “lowell”
14	Lista	Correto	Correto	Correto	Correto	Correto
15	Campo	Correto	Correto	Correto	Correto	Correto
16	Função	Correto	Errado – Retornou “a função”	Correto	Correto	Correto
17	Estilo	Correto	Correto	Errado – Retornou “Assistir”	Correto	Correto
18	Comando	Correto	Correto	Correto	Correto	Correto
19	Salvar	Errado – Retornou “o salvar”	Errado – Retornou “A”	Errado – Retornou “Falar”	Errado – Retornou “tomar”	Errado – Retornou “ao bar”
20	JavaScript	Correto	Errado – Retornou “o java script”	Correto	Correto	Errado – Retornou “parava de”

Fonte: Autor (2019)

O Quadro 9 apresenta a taxa de acertos dos experimentos realizados com a API IBM Watson *Speech to text*, identificando a quantidade de itens corretos dentre o total de palavras previstas, bem com uma média geral de aproveitamento.

Quadro 9 - Acertos dos experimentos com palavras da API IBM Watson *Speech to text*

Item	Sigla	Acertos	Total de itens	Porcentagem
Teste 1	T1	14	20	70%
Teste 2	T2	13	20	65%
Teste 3	T3	13	20	65%
Teste 4	T4	15	20	75%
Teste 5	T5	15	20	75%
Média	-	14	-	70%

Fonte: Autor (2019)

Embora tenha tido uma taxa média de acertos de 70% durante os experimentos, a ferramenta demonstrou certa inconsistência para reconhecer palavras, apresentando erros ao reconhecer certas palavras, como por exemplo as palavras “Tag”, “Web” e “Nome” que não foram identificadas corretamente em nenhum dos 5 experimentos realizados para cada uma dessas palavras. Quando ao acesso à ferramenta, é necessário realizar cadastro no serviço de nuvem da IBM, o IBM Cloud, e seu uso pode ser limitado dependendo do plano do serviço utilizado.

5.1.3 Web Speech API

O Quadro 10 apresenta os registros dos testes realizados com a API *Web Speech*, contendo as palavras e resultados de cada um dos 5 experimentos realizados.

Quadro 10 - Resultados dos experimentos com palavras da API Web Speech

n°	Palavra	T1	T2	T3	T4	T5
1	Web	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
2	HTML	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
3	Tabela	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
4	Parágrafo	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
5	Texto	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
6	Tag	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
7	Linha	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
8	Abrir	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
9	Fechar	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
10	Executar	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
11	Valor	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**

12	Título	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
13	Nome	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
14	Lista	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
15	Campo	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
16	Função	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
17	Estilo	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
18	Comando	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
19	Salvar	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**
20	JavaScript	**Correto**	**Correto**	**Correto**	**Correto**	**Correto**

Fonte: Autor (2019)

O Quadro 11 apresenta a taxa de acertos dos experimentos realizados com a API Web Speech, identificando a quantidade de itens corretos dentre o total de palavras previstas, bem com uma média geral de aproveitamento.

Quadro 11 - Acertos dos experimentos com palavras da API Web Speech

Item	Sigla	Acertos	Total de itens	Porcentagem
Teste 1	T1	20	20	100%
Teste 2	T2	20	20	100%
Teste 3	T3	20	20	100%
Teste 4	T4	20	20	100%
Teste 5	T5	20	20	100%
Média	-	20	-	100%

Fonte: Autor (2019)

A API demonstrou excelente eficiência ao reconhecer todas as palavras apresentadas durante os experimentos, demonstrando boa capacidade para ser utilizada no projeto. Ainda que apresente a vantagem de ser livre para uso e simples de integrar com aplicações web, possui o contraponto de, por enquanto, apenas apresentar suporte nativo ao reconhecimento de voz em navegadores com base do Google Chrome.

5.2 Conclusão sobre os experimentos das API’s

Com base na observação dos resultados dos experimentos de reconhecimento de voz das API’s, notou-se que a API Web Speech apresentou melhor resultado geral em relação as demais API’s testadas, mostrando-se eficiente o bastante para os propósitos do presente

trabalho. Além da satisfação apresentada durante os experimentos, o fato de a API ter acesso gratuito também favoreceu sua escolha em relação as outras API’s que apresentam conteúdo pago dependendo da forma como forem utilizadas. Em termos de desvantagens, limita-se o uso do navegador Chrome para o funcionamento do reconhecimento de voz da API. Dessa forma, a API Web Speech foi integrada ao módulo de reconhecimento de voz da arquitetura proposta.

6 Desenvolvimento

Neste tópico, aborda-se o desenvolvimento da solução proposta neste trabalho, de maneira a esclarecer todas as tecnologias, técnicas e métodos utilizados para implementar a aplicação protótipo. O detalhamento dar-se-á apresentando, primeiramente, as tecnologias utilizadas na implementação do código. Segue-se, então, mostrando a arquitetura construída, seguido pela apresentação individual de seus módulos principais e como estes se integram para o funcionamento da proposta de software. Por fim, é mostrado como o código fonte da aplicação funciona.

6.1 Tecnologias utilizadas

A seguir são descritas as tecnologias utilizadas para implementar a aplicação protótipo desde trabalho. Todas as tecnologias abordadas são de uso público e gratuito, garantindo o aspecto *open-source* deste trabalho.

6.1.1 Node.js

A plataforma Node.js, que consiste em um ambiente de execução JavaScript dirigido a eventos, foi utilizada como base da aplicação. Este ambiente foi escolhido por utilizar linguagem JavaScript para criar aplicações do lado servidor, além da facilidade para servir aplicações web de maneira rápida, escalável e assíncrona. A versão 12.18.3 do Node foi utilizada por ser a versão LTS (*Long Term Support*) à época do desenvolvimento, e garantir estabilidade na execução da aplicação. A linguagem JavaScript foi fator chave, visto que o autor desde trabalho possui experiência de uso com esta linguagem, fato que facilitaria o desenvolvimento da solução proposta.

6.1.2 Express

Express é um módulo para Node.js, que consiste em um framework web que abstrai a implementação de serviços web, permitindo a construção de servidores HTTP de maneira simples, além de construção de APIs HTTP, integração com motores de renderização do lado servidor, dentre outras vantagens de implementação, como a construção de aplicações SPA (*Single Page Applications*). Este módulo foi escolhido por facilitar a implementação do serviço web do protótipo desde trabalho e permitir a integração com mecanismo de renderização partindo do servidor. Na aplicação utilizou-se a versão 4.17.1 do Express.

6.1.3 Express Handlebars

Express Handlebars é um módulo para Node.js, que consiste em um motor de renderização do lado servidor criado para ser utilizado com o módulo Express. Este módulo possui uma estrutura própria de funcionamento, conforme Figura 17, e se integra com uma instância do módulo Express, permitindo que um serviço web construído com Express possa retornar como resposta uma página web de maneira dinâmica.

Figura 17 – Estrutura do módulo Express Handlebars

Fonte: www.npmjs.com/package/express-handlebars (2020)

Sua estrutura é composta de um diretório *views*, onde são armazenadas as páginas a serem servidas, cuja extensão de arquivo é *.handlebars*. No diretório *views* existe outro diretório chamado *layouts*, que armazena um arquivo *main.handlebars*, que serve de container principal

para a renderização das páginas a serem servidas. Nota-se, conforme a Figura 18, que a sintaxe de um arquivo handlebars consiste, majoritariamente, de HTML puro, facilitando a escrita de código *front-end* da aplicação, já que se integra, normalmente, com código CSS e JavaScript.

Outro ponto presente neste arquivo, ainda conforme a Figura 18, diz respeito ao elemento <*body*> do HTML, que possui uma notação conhecida por *mustache*, composta por uma série de chaves abrindo e fechando, englobando uma palavra que atua como variável de algum contexto. Neste caso, a palavra englobada <*body*> representa o conteúdo do elemento HTML <*body*> a ser injetado na página da aplicação em tempo de execução.

Figura 18 – Arquivo container principal - Express Handlebars

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Programando por voz</title>
</head>
<body>
    {{{body}}}
</body>
</html>
```

Fonte: Autor (2020)

Na raiz do diretório “views”, mostrado na Figura 17, existe um arquivo nomeado *home.handlebars*. Este arquivo contém HTML comum, e seu conteúdo será injetado no *body* do arquivo *main*, citado anteriormente. Dessa forma, devido as vantagens de manipulação apresentadas por este módulo, além do fato de utilizar sintaxe HTML comum, o motor Express Handlebars foi utilizado para os propósitos de implementação do presente trabalho. Foi utilizado a versão 5.1.0 desde módulo.

6.1.4 CodeMirror

CodeMirror, conforme a própria página do projeto, é um editor de texto implementado em JavaScript para ser utilizado em navegadores web. Seu foco é em criar editores de código e já oferece suporte nativo para várias linguagens, além de possuir uma rica API que garante a

manipulação e customização de um editor de código para ser embarcado em um projeto web, de acordo com a necessidade de seus usuários.

De maneira simplificada, seu uso consiste em relacionar uma instância do CodeMirror com um elemento HTML <*textarea*>. Desta forma, são injetadas sobre este elemento, todas as funcionalidades providas pelo componente CodeMirror, permitindo a sua manipulação por meio da API de manipulação do editor, como inserção de conteúdo, navegação pelo conteúdo do editor, personalização com uso de CSS, dentre outras possibilidades.

Por estas funcionalidades oferecidas pelo CodeMirror, este componente foi escolhido para criar o editor de código necessário para a implementação da proposta do presente trabalho. Na implementação desse trabalho, foi utilizada a versão 5.43.0 do CodeMirror.

6.1.5 Web Speech API

A API *Web Speech*, de uso aberto mantida pelo W3C (*World Wide Web Consortium*), provê funcionalidades nativas de reconhecimento de voz para navegadores web. Para que a API funcione é necessário que o navegador suporte o kit de desenvolvimento *webkitSpeechRecognition*. Uma vez suportado o reconhecimento de voz, é possível criar uma instancia do objeto da classe *webkitSpeechRecognition* para utilizar a funcionalidade de reconhecimento de voz e conversão de voz para texto, utilizado neste trabalho.

A API permite configurações para o uso do reconhecimento de voz, como por exemplo definir o atributo "*continuous*" da instância de reconhecimento como "*true"*, como destacado de verde na Figura 19, definindo que o reconhecimento será contínuo e permitirá que o usuário faça pausas ao longo de sua fala.

Figura 19 – Instância de reconhecimento de voz

```
var recognition = new webkitSpeechRecognition();
recognition.continuous = true;
recognition.interimResults = true;

recognition.onstart = function() { ... }
recognition.onresult = function(event) { ... }
recognition.onerror = function(event) { ... }
recognition.onend = function() { ... }
```

Fonte: https://developers.google.com (2020)

Esta API também provém uma série de ouvidores de eventos, destacados em vermelho na Figura 19, que disparam ações como no início do reconhecimento de voz (*onstart*), durante os resultados de reconhecimento de voz (*onresult*), assim como em casos de erro de funcionamento da API (*onerror*) e ao finalizar o reconhecimento de voz (*onend*).

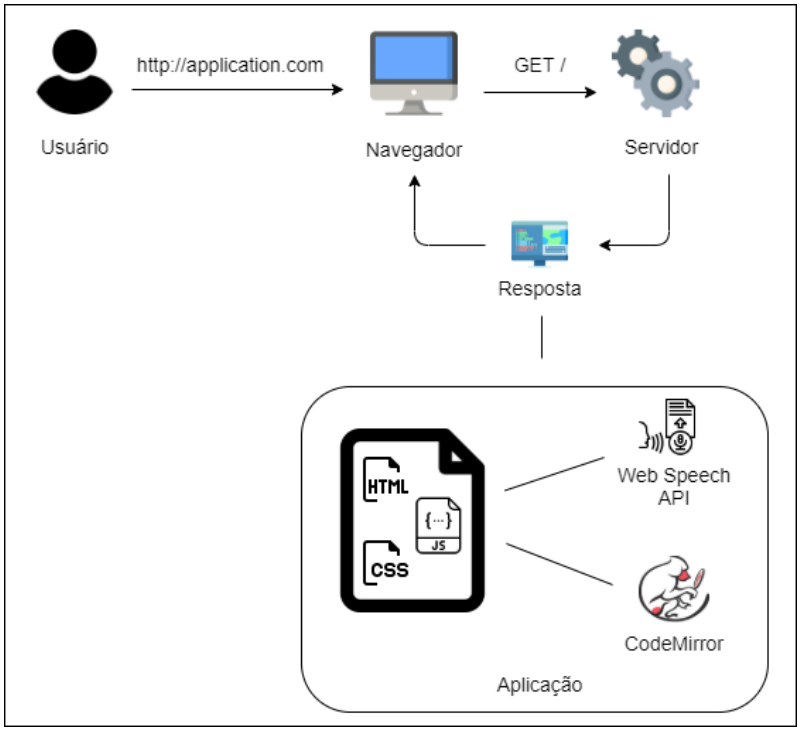
Para os propósitos do desenvolvimento do protótipo deste trabalho, foi utilizado apenas o ouvidor de eventos *onresult*, para capturar o reconhecimento de voz e sua consequente geração de texto. A simplicidade de uso desta API e sua disponibilidade para ambiente web foram os fatores que levaram a sua escolha para o uso neste trabalho.

6.2 Arquitetura da aplicação

Este tópico aborda a visão arquitetural da aplicação, mostrando como os pontos dela interagem entre si.

6.2.1 Funcionamento Geral

Figura 20 – Arquitetura geral da aplicação



Fonte: Autor (2020)

A Figura 20 apresenta a arquitetura geral da aplicação, para que se tenha uma visão macro de como o usuário interage com a aplicação e como esta realiza, modo simplificado, suas atividades.

Primeiramente, um usuário acessa a aplicação por meio de um navegador web. O usuário realiza uma chamada para o endereço da aplicação, aqui exemplificado por “http://application.com”. O navegador, realiza a chamada para o endereço em questão por meio de uma requisição HTTP Get. O servidor web da aplicação, ao receber a requisição para seu *endpoint* raiz “ / ”, responde com uma página web que é composta por um arquivo base HTML, além de uma série de arquivos JavaScript que compõem os algoritmos principais da aplicação, além de conteúdo CSS para estilização da página de resposta.

O código JavaScript retornado diz respeito a aplicação propriamente dita que controlará toda a interação do usuário, a lógica de todas as funcionalidades da aplicação, além do uso das tecnologias de controle de reconhecimento de voz (*Web Speech API*) e de controle do editor de código fonte (*CodeMirror*).

Nota-se que a arquitetura da aplicação tem por base o modelo cliente-servidor, visto que o usuário acessa a aplicação por meio da web. Destaca-se que, após a requisição inicial do usuário e da reposta do servidor, a aplicação é executada totalmente no lado cliente, já que as tecnologias principais utilizadas (*Web Speech API* e *CodeMirror*) foram concebidas para serem executadas em páginas web. Além disso, o fato de a aplicação executar do lado cliente, sem muitas requisições para o servidor da aplicação, garantem maior otimização e velocidade para a aplicação, já que não será necessário um uso intenso de rede (fato que não prejudica o uso da aplicação em caso de falhas na internet), pois os recursos necessários para a execução já foram buscados na primeira requisição realizada.

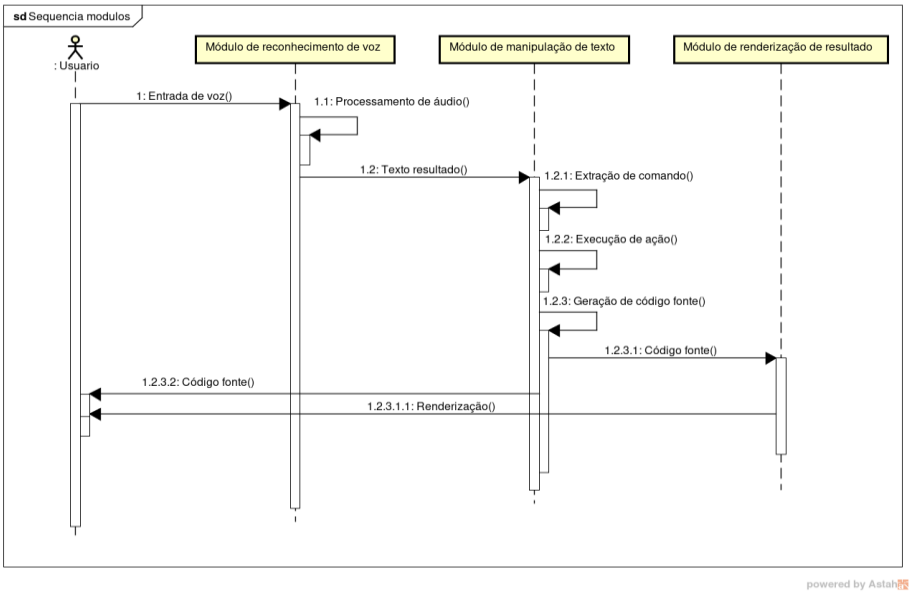
6.2.2 Arquitetura Modular

Ressalta-se a característica modular da aplicação, proposta para funcionar sob três módulos principais: módulo de reconhecimento de voz, módulo de manipulação de texto e módulo de renderização de resultado.

Os módulos interagem-se da seguinte maneira, conforme diagrama de sequência da Figura 21: Uma vez pronto para iniciar o processo de reconhecimento de voz, o usuário inicia

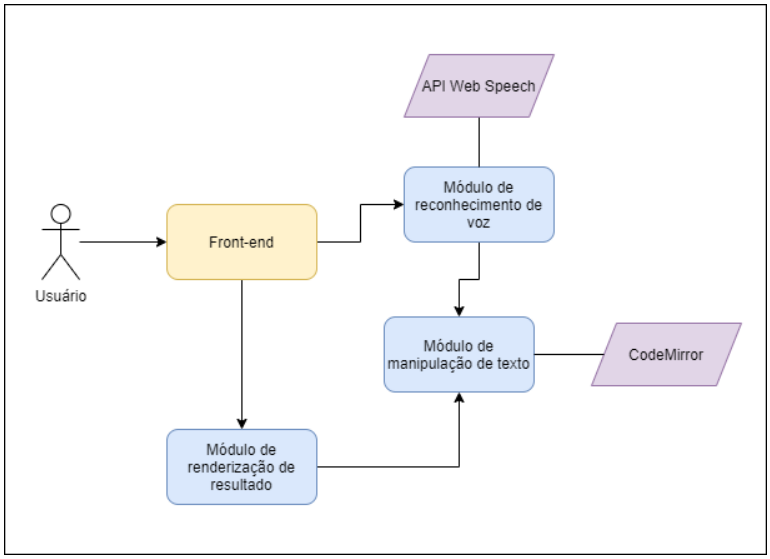
a fala. Essa entrada de voz é então capturada pelo módulo de reconhecimento de voz, que realiza o processamento dos sinais de áudio e transforma estes sinais em texto. O texto é, então, enviado ao módulo de manipulação de texto que, em um primeiro momento, realiza a extração de um comando no texto recebido. Em seguida o módulo de manipulação de texto executa uma ação programada para o comando em questão, resultando na geração de código fonte. Por fim, o módulo de renderização de conteúdo é responsável por obter este código fonte gerado e exibir o seu resultado para o usuário.

Figura 21 – Diagrama de sequência dos módulos da aplicação



Fonte: Autor (2020)

Figura 22 – Integração dos módulos da aplicação com serviços terceiros

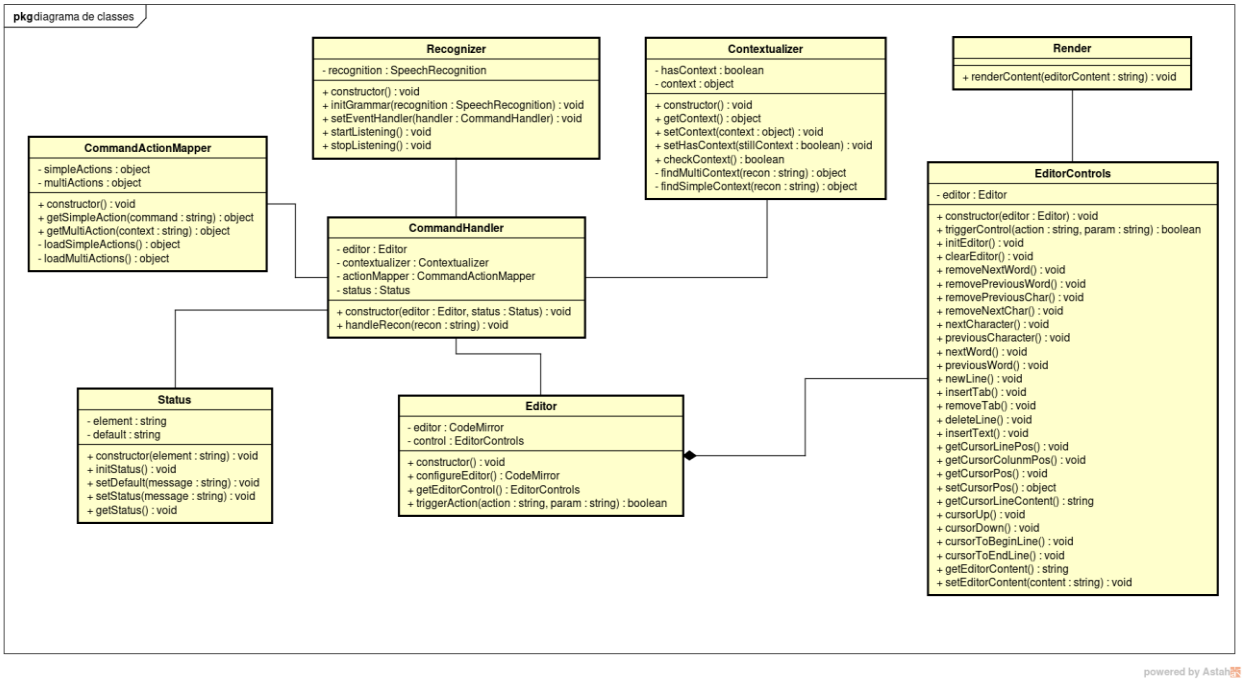


Fonte: Autor (2020)

É importante ressaltar que os módulos da arquitetura utilizam dois componentes terceiros aos implementados pelo autor: API *Web Speech* e CodeMirror. Conforme ilustrado pela Figura 22, nota-se os módulos representados por quadrados azuis, enquanto que os componentes externos, ou seja, aqueles que não foram desenvolvidos pelo autor deste trabalho, estão representados por uma espécie de losango de cor roxa. O módulo de reconhecimento de voz utiliza as funcionalidades oferecidas pela API *Web Speech* e o módulo de manipulação de texto utiliza, em sua estrutura interna, toda a estrutura do componente CodeMirror.

6.2.3 Diagrama de classes da aplicação

Figura 23 – Diagrama de classes



Fonte: Autor (2020)

Os módulos presentes na aplicação foram escritos em forma de classes, apresentados neste tópico pelo diagrama de classes mostrado na Figura 23. A interação entre as classes em questão será detalhada na seção 6.3.3.

6.2.4 Contextos de comandos de voz

Durante o reconhecimento de voz e posterior manipulação do texto do reconhecimento da fala, é feita a extração de comandos e a execução de ações para tal comando extraído. É importante destacar que a extração de comandos de voz leva em conta um sistema de contextos criados para a solução proposta neste trabalho, sendo estes contextos chamados de simples e

múltiplos. Destaca-se, também, que uma ação aqui citada diz respeito a uma função que é executada para determinado comando.

Um comando de voz é aqui classificado como de contexto simples quando este comando possui apenas uma ação direta, não sendo uma entrada para outros comandos e nem esperando um outro comando como argumento. De forma contrária, um comando de voz é classificado como de contexto múltiplo, quando este comando define um contexto para aguardar outros comandos de voz, sendo estes próximos comandos parâmetros de um comando anterior, ou apenas contextos destes comandos anteriores. Comandos de contexto múltiplo podem ou não executar ações, já que há casos em que apenas disparam o contexto de outros comandos.

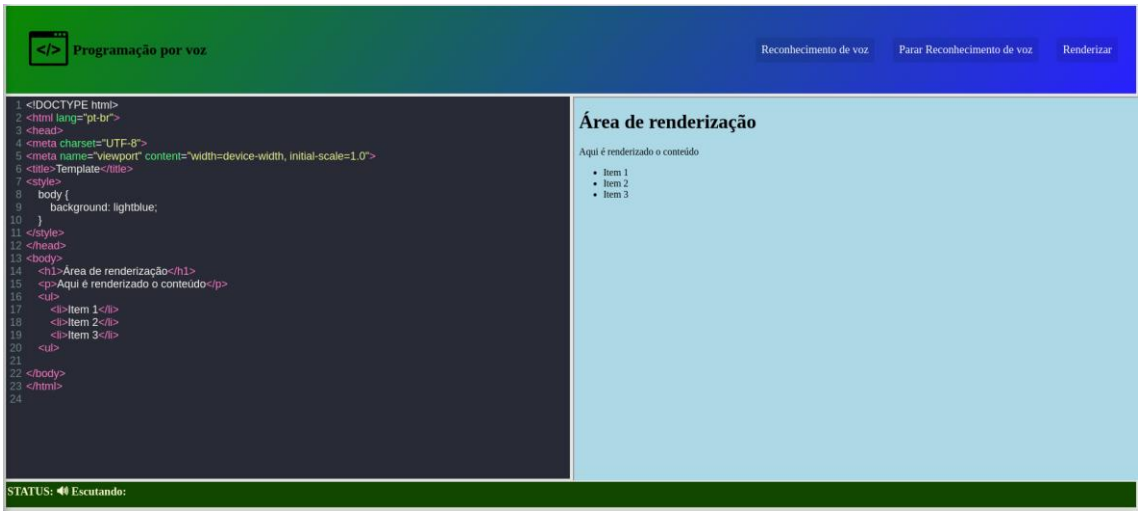
6.3 Código da aplicação

Este tópico aborda o funcionamento do código da aplicação. Para tal, vários tópicos destacam, partindo do *front-end*, como foram desenvolvidos os módulos até então apresentados, e como as tecnologias foram implantadas.

6.3.1 Front-end da aplicação

O *front-end* da aplicação é constituído por uma única página que possui um layout simples, como observado na Figura 24. O layout da página é composto por um cabeçalho, uma área central de conteúdo e um rodapé. O cabeçalho possui um *logo* e o nome da aplicação, além de três botões responsáveis por iniciar o reconhecimento de voz, finalizar o reconhecimento de voz e renderizar o resultado do código desenvolvido. A área central de conteúdo é formada por dois containers, sendo o da esquerda responsável por abrigar o editor de código fonte da aplicação, e o da direita responsável por exibir a renderização de resultado do código fonte desenvolvido. Por fim, cabe ao rodapé exibir os status de execução da aplicação.

Figura 24 – Front-end da aplicação



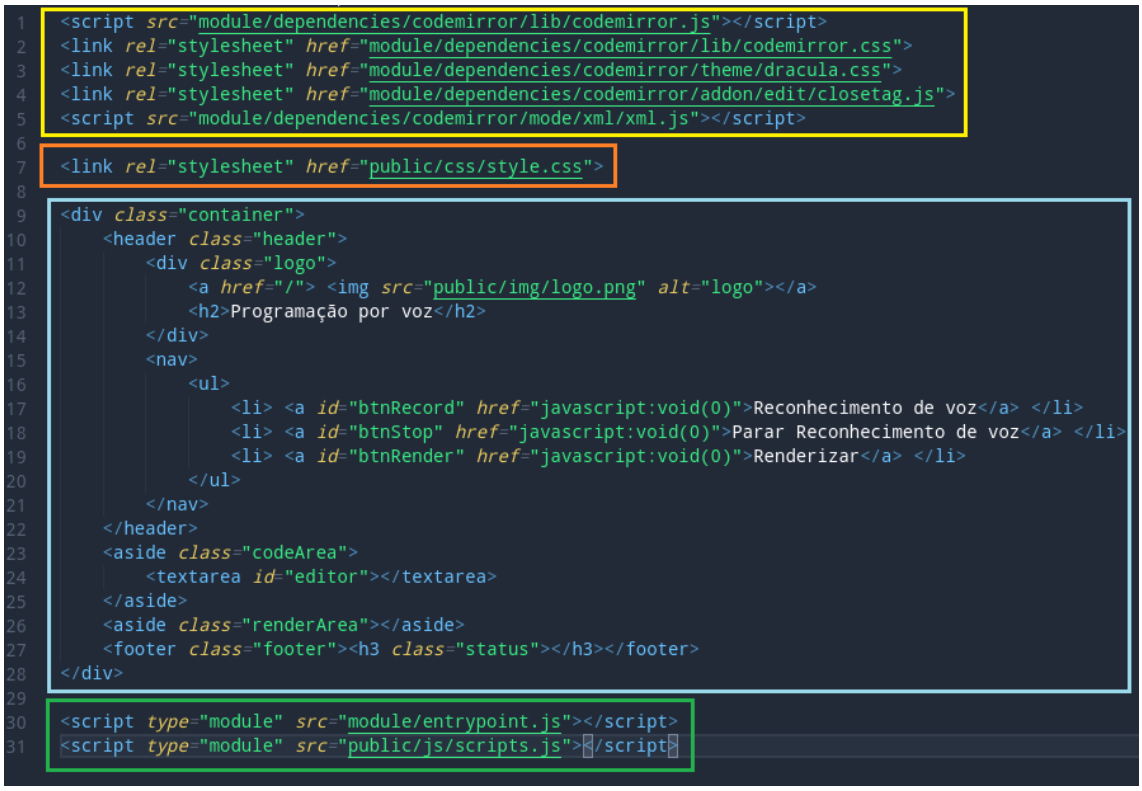
Fonte: Autor (2020)

A página foi escrita com HTML 5, utilizando o módulo Express Handlebars já citado. Buscou-se escrever um código simples, de modo a manter a característica semântica do HTML, uma vez que não foi definido marcação de estilo e nem scripts diretamente neste código. O estilo da página e seus scripts foram apenas referenciados para consumirem recursos externos.

Conforme a Figura 25, observa-se que todo o conteúdo apresentado na página (destacado em azul) está presente em uma <*div*> marcada pela classe *container*. O cabeçalho da página está contido no elemento <*header*>, os componentes da área central são representados pelos elementos <*aside*> e o rodapé pelo elemento <*footer*>. No cabeçalho há uma <*div*> marcada com a classe *logo,* que abriga a imagem do *logo* e o texto do nome da aplicação. Além disso há um elemento <*nav*> que cria uma pequena barra de navegação do lado superior direito que engloba uma lista de elementos <*a*> que são estilizados para criar os botões para controle de reconhecimento de voz e renderização.

No começo do código, destacado em amarelo na Figura 25, há várias referências para as dependências do CodeMirror que são injetados na página, dentre elas seus scripts e páginas de estilo. Destacado em laranja há uma referência para o CSS da página em si, nomeado *style.css*, e ao final do código, destacado em verde, são referenciados os scripts da aplicação (entrypoint.js e scripts.js) que atuam como pontos de entrada para os códigos que tratam das funcionalidades de reconhecimento de voz e manipulação do editor de código fonte, além da renderização, desempenhados pela aplicação proposta neste trabalho.

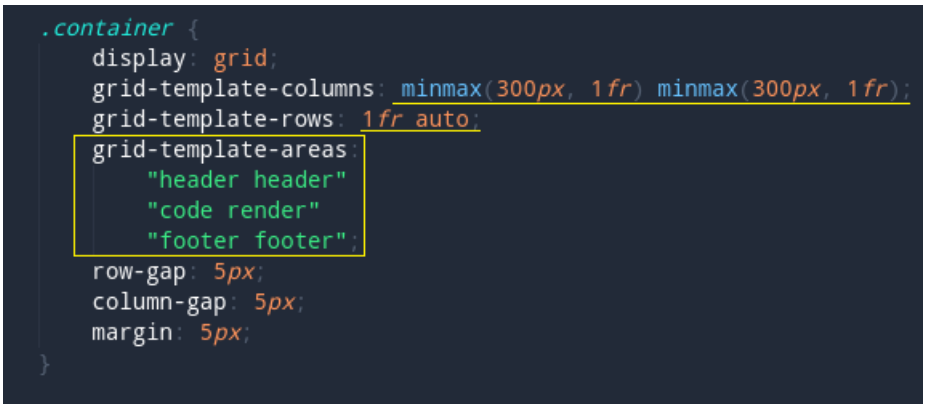
Figura 25 – Código do *front-end* da aplicação



Fonte: Autor (2020)

Conforme citado, o estilo da página está escrito em *style.css.* O layout da página (posicionamento de seus elementos) foi feito com base na tecnologia CSS Grid, nativa do CSS 3, além do uso, em alguns pontos, de Flexbox, outra tecnologia nativa do CSS 3.

Figura 26 – Uso de CSS Grid para definir disposição de layout



Fonte: Autor (2020)

O uso do CSS Grid facilita a disposição dos elementos na tela com o uso de sua propriedade *grid-template-areas* (Figura 26, destacado em amarelo), de forma que são definidos nomes que atuam como marcadores de posições, de acordo com a forma como se quer dividir a tela. Essa propriedade cria uma marcação de posições (ilustrada pela Figura 27), possibilitando que em outros elementos seja feita a referência à área desejada para se alocar aquele componente. Por exemplo, o elemento <*header*> presente no *front-end*, possui o estilo definido na classe CSS *.header*. Nesta classe, é feita a referência para a posição header, definida na classe *.container*, por meio do atributo *grid-area*, conforme a Figura 28. Desta forma, todo elemento marcado pela classe CSS *.header* ocupará a posição de tela definida por header (Figura 27), independentemente do tamanho de tela do dispositivo que executa a aplicação, garantindo a capacidade responsiva da aplicação.

Figura 27 – Ilustração de disposição da propriedade grid-template-areas

header	header
code	render
footer	footer

Fonte: Autor (2020)

Figura 28 – Atribuição de posição de layout

```
.header {
    grid-area: header;
    height: 10vh;
    background: linear-gradient(130deg, #0b8a00, #2921ff);
    padding: 30px;
    display: grid;
    grid-template-columns: 1fr auto;
    align-items: center;
}
```

Fonte: Autor (2020)

6.3.2 Pontos de entrada da aplicação

Quando a aplicação é carregada, a partir da resposta do servidor (conforme abordado na seção 6.2.1), dois scripts JavaScript, presentes na página front-end, são executados, respectivamente: entrypoint.js e scripts.js.

O primeiro carrega os módulos da aplicação (reconhecimento de voz, manipulação de texto, renderização de resultado), criando suas instâncias e definindo funções de alto nível para iniciar o reconhecimento de voz, finalizar o reconhecimento e renderizar o resultado, conforme ilustrado na Figura 29.

Figura 29 – entrypoint.js

```
import Recognizer from './voice/Recognizer.js'
import Editor from './editor/Editor.js'
import CommandHandler from './handler/CommandHandler.js'
import Render from './render/Render.js'
import Status from './status/Status.js'

const status = new Status('.status')
const editor = new Editor()
const rec = new Recognizer()
const commandHandler = new CommandHandler(editor, status)
const render = new Render()

export function startVoiceRecognition(){
    rec.setEventListener(commandHandler)
    rec.startListening()
    status.setDefault('STATUS: 🔊 Escutando: ')
}

export function stopVoiceRecognition(){
    rec.stopListening()
    status.setDefault('STATUS: ')
    status.setStatus('🔇 Reconhecimento de voz finalizado')
    setTimeout(()=>{
        status.setStatus('')
    }, 2500)
}

export function renderContent(){
    const control = editor.getEditorControl()
    render.renderContent(control.getEditorContent())
    status.setStatus('Conteúdo renderizado')
    setTimeout(()=>{
        status.setStatus('')
    }, 2000)
}
```

Fonte: Autor (2020)

O segundo, scripts.js, possui uma classe que define funções de controle da página da aplicação, como os ouvidores de evento dos botões de reconhecimento de voz e renderização, além de atalhos de teclado para utilização dos mesmos botões. A Figura 30 apresenta parte do conteúdo do código desde arquivo.

Figura 30 – scripts.js

```
import {startVoiceRecognition as start} from '../../../module/entrypoint.js'
import {stopVoiceRecognition as stop} from '../../../module/entrypoint.js'
import {renderContent as render} from '../../../module/entrypoint.js'

class Scripts {

    setRecordButtonListener(){
        document.querySelector("#btnRecord").addEventListener("click", ()=>{
            start()
        })
    }

    setStopButtonListener(){
        document.querySelector("#btnStop").addEventListener("click", ()=>{
            stop()
        })
    }

    setRenderButtonListener(){
        document.querySelector("#btnRender").addEventListener("click", ()=>{
            render()
        })
    }

    setMnemonics(){
        document.onkeydown = (event)=> {
            //ctrl + alt + r
            if(event.ctrlKey && event.altKey && event.key == 'r') start()
            //ctrl + alt + s
            if(event.ctrlKey && event.altKey && event.key == 's') stop()
            //ctrl + alt + p
            if(event.ctrlKey && event.altKey && event.key == 'l') render()
        }
    }

}
```

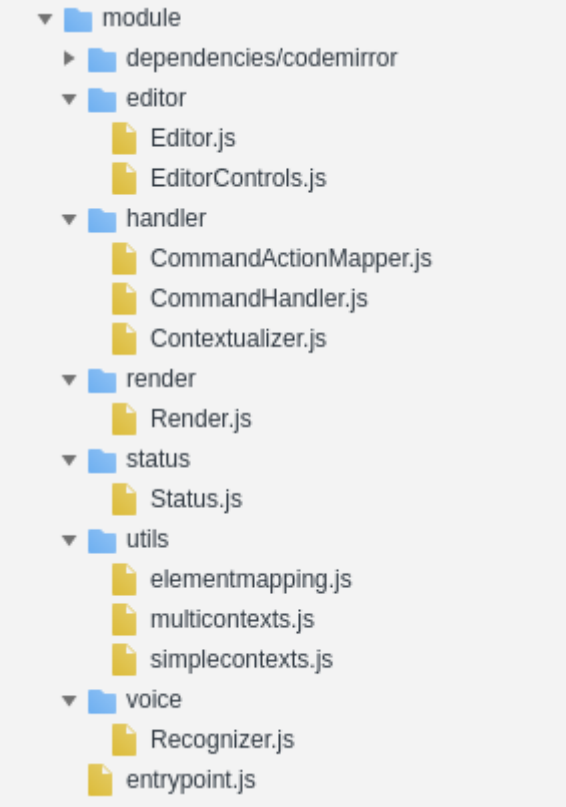
Fonte: Autor (2020)

Destaca-se, ainda conforme mostrado a Figura 30, que os eventos definidos para os botões da página, utilizam as funções alto nível definidas em entrypoint.js. Desta forma, tem-se uma divisão de responsabilidades entre os componentes da aplicação.

6.3.3 Módulos da aplicação

Os módulos da aplicação estão organizados em pastas, de acordo com suas responsabilidades (Figura 31). Observa-se que o ponto de entrada *entrypoint.js* localiza-se na raiz dessa estrutura. Todos os módulos são constituídos por classes, salvo o módulo *utils* que apresenta conteúdo consumido por outros módulos. A seguir, são apresentados os módulos individualmente, bem como suas classes constituintes e seu funcionamento.

Figura 31 – Estrutura dos módulos da aplicação



Fonte: Autor (2020)

6.3.3.1 Módulo de reconhecimento de voz

O módulo de reconhecimento de voz, presente no diretório *voice* (Figura 31) é formado, basicamente, pela classe Recognizer.js. Nessa classe temos um construtor que cria uma instância de reconhecimento de voz utilizando a API *Web Speech*. Além disso esta classe configura a instância de reconhecimento de voz, preparando-a para reconhecer língua portuguesa e definindo o reconhecimento contínuo de voz (conforme abordado na seção 6.1.5).

A classe possui três métodos utilizados externamente por outros contextos da aplicação: *setEventListener*(*handler*), *startListening*() e *stopListening*(). O primeiro método citado é responsável por iniciar o ouvidor de evento de reconhecimento de voz, obtendo o resultado em texto da fala. A partir de um objeto de manipulação de comandos recebido por parâmetro (*handler*), o resultado do reconhecimento de voz é transmitido ao módulo de manipulação de texto. Os métodos *startListening*() e *stopListening*() são responsáveis, respectivamente, por iniciar e terminar o reconhecimento de voz.

6.3.3.2 Módulo de manipulação de texto

O módulo de manipulação de texto é composto pelo componente de manipulação, presente no diretório *handler* (Figura 31), formado pelas classes CommandHandler.js, CommandActionMapper.js e Contextualizer.js, e pelo componente de editor, presente no diretório *editor* (Figura 31), constituído pelas classes Editor.js e EditorControls.js.

A classe CommandHandler.js é a manipuladora principal do texto que chega do módulo de reconhecimento de voz. É dessa classe a instância passada para o método *setEventListener()* da classe Recognizer.js citada anteriormente. Esta classe possui um construtor que recebe uma instância da classe Editor.js, e uma instância da classe Status.js, além de criar instâncias das classes Contextualizer.js e CommandActionMapper.js. O método *handleRecon(recon),* presente nesta classe, é o responsável por receber o texto do reconhecimento de voz passado ao módulo de manipulação de texto. Uma vez que este método recebe o texto de fala reconhecido (*recon*), é executada uma verificação de contexto de comando por meio do método *checkContext(recon)* da classe Contextualizer.js.

Na análise de contexto de comando da classe Contextualizer.js, é verificado se o atributo *hasContext* está definido como *true* ou *false.* Caso *hasContext* seja falso, assume-se que não há contexto múltiplo definido naquele momento da execução, dessa forma a aplicação prossegue para tentar extrair um comando de contexto simples por meio do método *findSimpleContext(),* desta mesma classe. Se não for encontrado contexto simples, prossegue-se para a busca de um possível contexto múltiplo, por meio do método *findMultiContext*(). Uma vez encontrado um contexto simples ou múltiplo, o resultado é retornado para *checkContext*(), que por sua vez retorna o resultado para CommandHandler.js. Caso o contexto encontrado em *checkContext*() for simples, o atributo *hasContext* permanece como *false*. Caso contrário, esse atributo é definido então como true, indicando que um contexto múltiplo está ocorrendo naquele momento. Ao se encontrar um contexto, o atributo *context* recebe o valor do contexto do comando em questão. Em caso de *hasContext* estar definido como *true* no começo da análise de *checkContext*(), a aplicação entende que já existe um contexto de comando múltiplo naquele momento da execução, então procede para retornar o contexto presente no atributo *context*.

Uma vez retornada a resposta da análise de contexto de comando para o método *handleRecon*(), na classe CommandHandler.js, é então feita uma verificação da resposta. Caso a resposta seja nula, é interpretado que o comando de voz recebido não está mapeado na

aplicação, então nenhuma ação é executada. Caso exista resposta, verifica-se se a resposta indica ou não existência de contexto. Se a resposta indicar existência, entende-se que o comando de voz reconhecido está mapeado na aplicação e que o mesmo possui contexto múltiplo. Caso contrário, é abstraído que o comando existe, porém seu contexto é simples.

Após a análise de contexto, e garantido que o comando está mapeado, é feita a busca pela ação a ser executada para aquele comando em questão. Em ambos os contextos o tratamento é semelhante. Primeiro, recupera-se o contexto daquele comando com a chamada do método *getContext*(), por meio da instância da classe Contextualizer.js. Em seguida, este contexto obtido é repassado como argumento para o método *getMultiAction*() ou *getSimpleAction*(), da classe CommandActionMapper.js, de acordo com o contexto do comando.

Na classe CommandActionMapper.js, é feito o mapeamento das ações dos comandos de voz. Quando um contexto chega em um de seus métodos de busca de ação (*getMultiAction*() ou *getSimpleAction*()), estes consultam os utilitários de recursos da aplicação, contidos no módulo *utils*. No módulo *utils,* representado pelo diretório utils (Figura 31), estão arquivos que mapeiam as relações de comandos de voz e suas ações a serem executadas, além de outros mapeamentos, como os elementos a serem exibidos no editor de código durante a utilização da aplicação.

Depois de encontrar a ação para o contexto de comando passado, esta ação é retornada para CommandHandler.js, que prossegue para a execução da ação de comando por meio da instância da classe Editor, recebida em seu construtor. É executado, então, o método *triggerAction()* da classe Editor.js, que por sua vez, executa o método *triggerControl*() da classe EditorControls.js. A classe Editor.js é responsável por configurar o editor de código fonte da aplicação, a partir da instanciação de um objeto do componente CodeMirror. A classe Editor.js é composta por um objeto da classe EditorControls.js, que diz respeito aos controles do editor de código, ou seja, às funções que são executadas para realizar ações sobre o editor. Todas as funções declaradas em EditorControls.js são correspondentes as ações emitidas pelo método *triggerControl*().

Ao fim da execução da ação do comando, o contexto de Contextualizer.js é atualizado, de acordo com o contexto retornado no mapeados de ações (CommandActionMapper.js), podendo ‘zerar’ o múltiplo contexto daquele momento de execução, ou definir um próximo contexto aninhado ao comando anterior. Ao final da manipulação do texto do reconhecimento,

por meio da instância da classe Status.js, é definido o status de execução da aplicação, exibido no rodapé da página do *front-end.*

6.3.3.3 Módulo de renderização de resultado

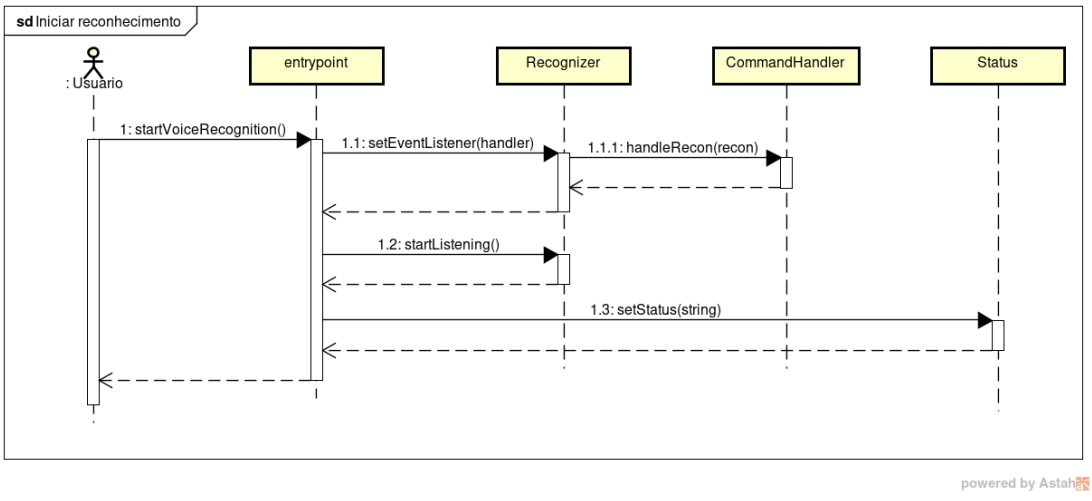
O módulo de renderização de resultado, presente no diretório *render* (Figura 31), é composto pela classe Render.js. Nela há apenas um método chamado *renderContent()*, que recebe como parâmetro um conteúdo de texto extraído por meio da uma instância de EditorControl.js, que retorna o conteúdo presente no editor de texto de uma instância de Editor.js. O método procede para a criação de um elemento <*iframe*> que será injetado na página do *front-end*, na posição de layout render, conforme ilustrado pela Figura 28. Neste *iframe* é inserido o conteúdo recebido do editor e realizada a exibição ao usuário.

6.3.4 Diagramas de sequência da aplicação

A seguir são apresentados diagramas de sequência dos pontos do código abordados até aqui. Os diagramas foram divididos pelo funcionamento de seus módulos, com vista a tornar sua legibilidade mais simples e reduzir o tamanho dos diagramas, evitando poluição visual.

A Figura 32 ilustra o diagrama referente ao processo de iniciar reconhecimento de voz, partindo da interação do usuário no front-end, que dispara a função startVoiceRecognition().

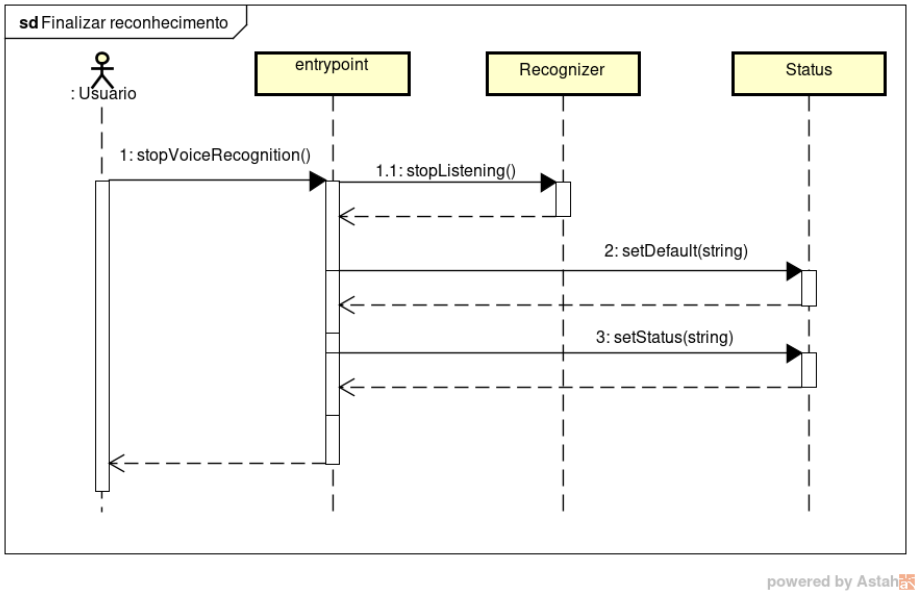
Figura 32 – Diagrama de sequência – Iniciar reconhecimento de voz



Fonte: Autor (2020)

A Figura 33 mostra se refere ao processo de finalizar reconhecimento de voz, partindo da interação do usuário, no front-end, ao disparar a função stopVoiceRecognition().

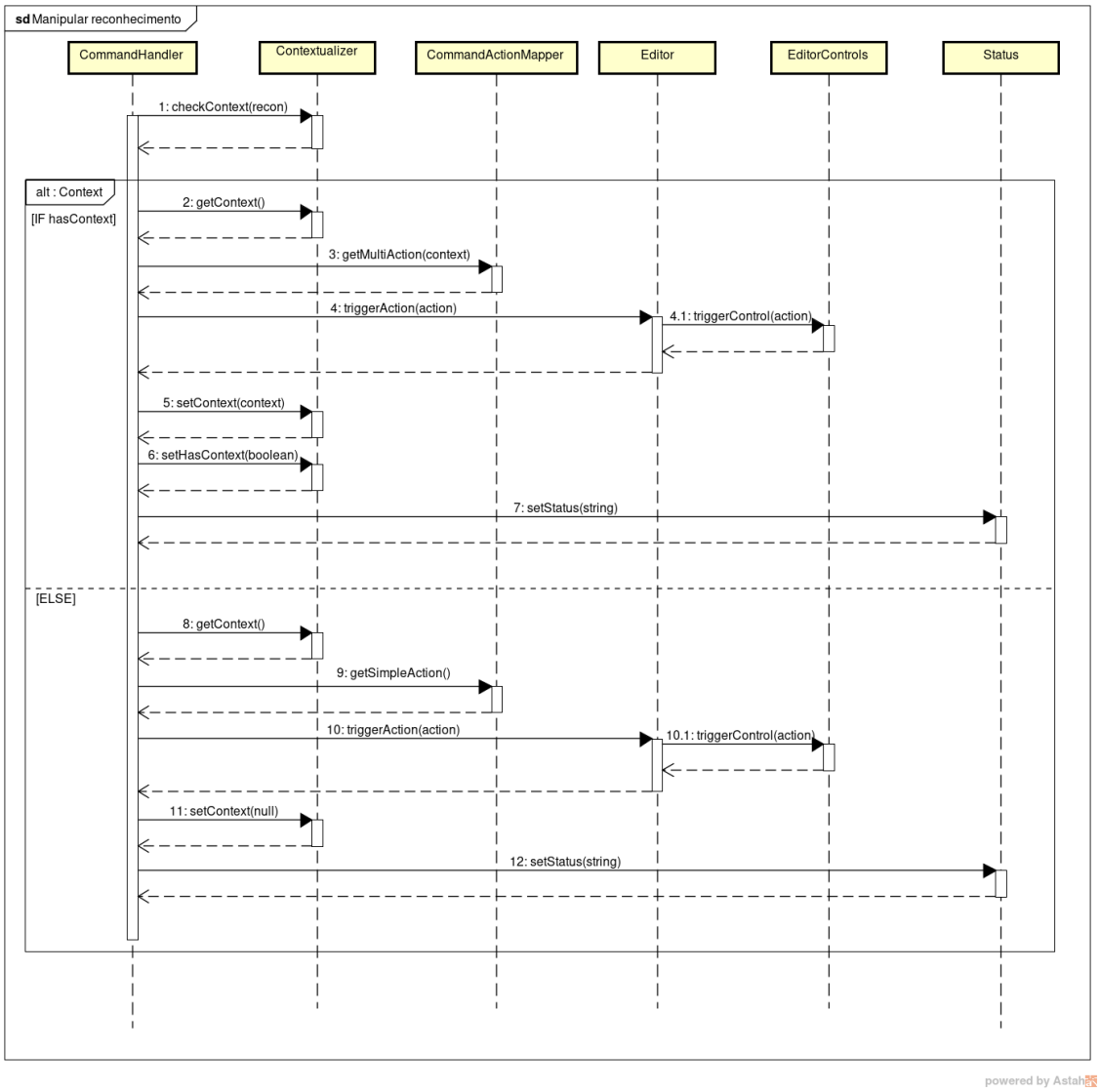
Figura 33 – Diagrama de sequência – Finalizar Reconhecimento de voz



Fonte: Autor (2020)

A Figura 34 apresenta o processo de manipulação de texto de reconhecimento de voz descrito na seção 6.3.3.2, onde a classe CommandHandler.js é chamada como ponto de entrada do módulo de manipulação de texto.

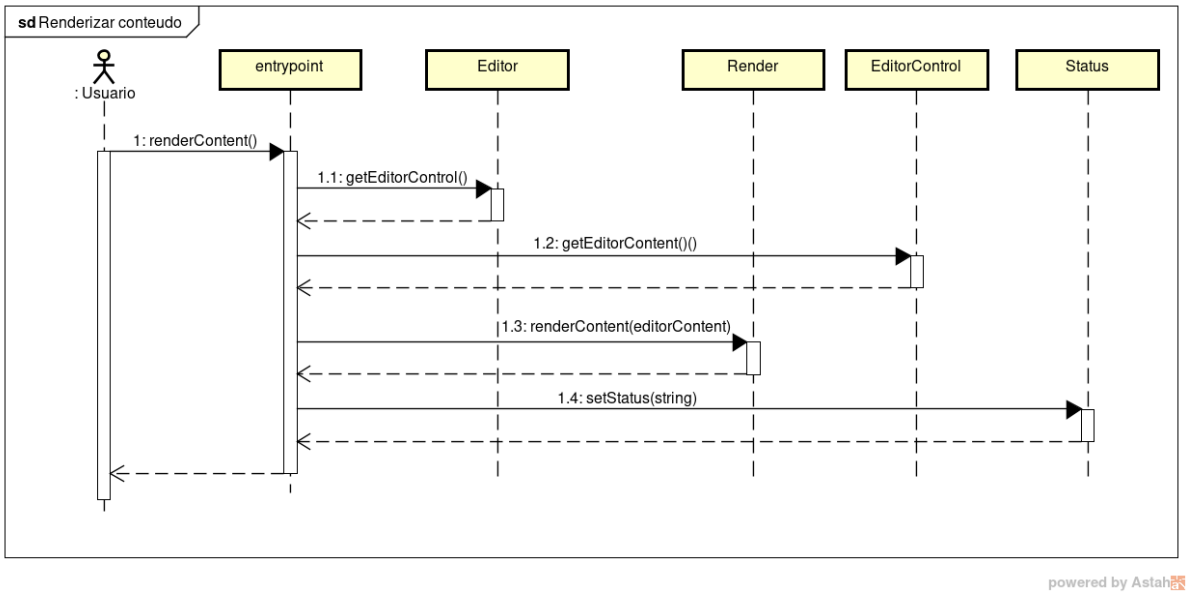
Figura 34 – Diagrama de sequência – Manipulação de texto de reconhecimento de voz



Fonte: Autor (2020)

A Figura 35 apresenta o processo de renderização de resultado para o usuário final, a partir de sua interação com o *front-end*, onde é executada a função *renderContent*(). O funcionamento do módulo foi descrito na seção 6.3.3.3.

Figura 35 – Diagrama de sequência – Renderizar resultado



Fonte: Autor (2020)

7 Considerações Finais

Ao início dos trabalhos de pesquisa, surgiu a problemática de como dinamizar o desenvolvimento de software em ambientes que apenas dispõem do uso de mouse e teclado. Dessa forma, com base em pesquisas iniciais foi observado que o uso da fala poderia ser uma medida viável como alternativa a esses dispositivos de entrada de dados.

A pesquisa teve como objetivo geral desenvolver um ambiente de programação para a geração de código *front-end* de aplicações *web* por meio de comandos de voz. Contata-se que o objetivo geral foi atendido pois a aplicação desenvolvida com base nas propostas, demonstrou que a partir do uso de reconhecimento de voz é possível gerar código fonte de aplicações utilizando-se apenas de comandos de voz, sem a necessidade massiva do uso de teclado e mouse para a escrita desse código.

Os objetivos específicos desta pesquisa também foram atingidos. O primeiro objetivo foi o de desenvolver uma arquitetura modular para a aplicação. Este objetivo foi atingido pois foi projetada e apresentada arquitetura correspondente aos módulos da aplicação, composto pelo módulo de reconhecimento de voz, o módulo de manipulação de texto e o módulo de renderização de resultado. O segundo objetivo, de implementar a aplicação com as tecnologias escolhidas para os módulos também foi alcançado, visto que cada módulo se utilizou das tecnologias analisadas para a sua finalizada, como por exemplo o módulo de reconhecimento de voz que utilizou a *API Web Speech.* O terceiro e último objetivo específico foi concretizado já que foi feita uma aplicação funcional a partir a integração de todos os módulos da arquitetura, o que permitiu o desenvolvimento de código *front-end* por meio de comandos de voz.

A hipótese de utilizar a fala como forma alternativa ao uso de mouse e teclado para o desenvolvimento de software se mostrou verdadeira, já que a aplicação desenvolvida permitiu utilizar apenas a fala como forma de repassar comandos ao sistema, que por sua vez interpretou estes comandos e gerou o código fonte. Assim, demonstrando que há possibilidade se aproveitar o uso da fala, sem que se dependa totalmente do uso de mouse e teclado para a escrita de código de softwares de computadores.

7.1 Riscos e Dificuldades

Durante o desenvolvimento da aplicação proposta neste trabalho, observaram-se alguns pontos quanto a possíveis riscos futuros, e dificuldades que desfavorecem a aplicação. Primeiramente, destaca-se que a API de reconhecimento de voz *Web Speech,* concebida para ser utilizada em navegadores web, por estar ainda em estágio de desenvolvimento, não é disponibilizada em uma ampla quantidade de navegadores. Seu kit de desenvolvimento e capacidade de voz para texto funciona, até o momento da construção desde trabalho, em versões modernas do navegador Google Chrome, limitando a utilização da ferramenta neste navegador.

Outro ponto a se destacar é quanto a capacidade de reconhecimento de voz da API *Web Speech.* A acurácia no reconhecimento de voz pela API é correspondente a qualidade de captação de áudio do microfone utilizado. Dessa forma, microfones de qualidade inferior geram maior interferência no sinal, resultando em erros de reconhecimento de voz. Além disso, ruídos externos durante o reconhecimento interferem na capacidade de reconhecimento, como por exemplo, se mais de uma pessoa estiver falando no momento de captura de voz pelo sistema. Nota-se, ainda, que o ouvidor de eventos gera um certo *delay* entre os comandos de voz, o que reduz a qualidade da usabilidade da aplicação.

Um ponto de dificuldade observado quanto ao uso do componente CodeMirror é que em alguns navegadores, é possível que alguns de seus pacotes internos não seja reconhecido, fato que gera erros em tempo de execução e prejudica na utilização do componente embarcado na página web da aplicação.

Com base nestes pontos destacados, é necessário que sejam feitos ajustes de arquitetura no futuro, como forma a otimizar a ferramenta. Algumas propostas de melhoria serão citadas na próxima seção.

8 Trabalhos futuros

Conforme destacado na seção anterior, alguns pontos de melhoria podem ser citados com vista a gerar futuras melhorias no uso da ferramenta, bem como no seu funcionamento.

O uso da API de reconhecimento de voz *Web Speech* se mostrou satisfatória para os propósitos deste trabalho, porém pode ser realizado incremento no módulo de reconhecimento de voz, de modo a otimizar esta funcionalidade. Visto que a API Web Speech ainda se encontra em desenvolvimento e não funciona em todos os navegadores, seria mais interessante que esta funcionalidade de reconhecimento de voz passasse para o contexto do *back-end* da aplicação, deixando no front-end apenas a responsabilidade de captar os sinais de áudio do microfone do usuário. Desta forma, a capacidade de execução em outros navegadores seria ampliada, e o controle sobre a forma como o reconhecimento de voz é realizado seria maior. Tendo o controle sobre como o reconhecimento de voz funciona, incrementos em sua capacidade não estariam ligadas ao desenvolvimento de terceiros.

Abstrair o modo de reconhecimento de voz para o servidor da aplicação, em vez de utilizar em navegador, poderia expandir seu uso além do ambiente web, como em aplicativos móveis e desktop. Em caso de mudança do módulo de reconhecimento de voz para o back-end, recomenda-se que o modelo cliente-servidor na transmissão dos dados seja evitado, uma vez que esse modelo depende das capacidades de rede e pode gerar muita latência no uso, prejudicando a usabilidade da aplicação. O mais recomendado para este caso seria a utilização de um modelo de web proxy ou alguma outra forma de realizar *streaming* dos dados de uma forma direta entre seu utilizador e o servidor da aplicação, aproveitando a velocidade e otimização deste tunelamento.

Em relação ao editor de código fonte da aplicação, recomenda-se que seja analisada a próxima versão (versão 6) do CodeMirror, que está em desenvolvimento, e propõe mudanças de utilização do componente, além de melhorias. Com essa versão seria possível realizar a utilização do módulo a partir do *back-end*, removendo a responsabilidade do *front-end* de manipular seus elementos.

No mais, fica livre à exploração das ideias e propostas de desenvolvimento apresentadas neste trabalho, visto que tudo o que foi utilizado tem caráter *open-source* e de uso público.

9 Referências

ASSUNÇÃO, A. A.; ALMEIDA, I. M. **Doenças osteomusculares relacionadas com o trabalho:** membro superior e pescoço. São Paulo: Atheneu, 2003.

LUTTMANN, A. **Preventing musculoskeletal disorders in the workplace. Protecting worker's health series:** no. 5. [S.l.]: World Health Organization, 2003. 32 p. Disponivel em: <https://www.who.int/occupational_health/publications/muscdisorders/en/>.

COHEN, P. R.; OVIATT, S. L. **The role of voice input for human-machine communication**, 1993.

LIU, W. **Natural user interface - Next mainstream product user interface**. IEEE 11th International Conference on Computer-AidedIndustrial Design & Conceptual Design 1, 2010.

KUORINKA, I.; FORCIER, L. **Work-related musculoskeletal disorders (WMSDs): a reference book for prevention**. London: Softcover, 1995.

BRASIL. MINISTÉRIO DA SAÚDE. DEPARTAMENTO DE AÇÕES PROGRAMÁTICAS E ESTRATÉGICAS. ÁREA TECNICA DE SAÚDE DO TRABALHADOR. **Lesões por Esforços Repetitivos (LER) e Distúrbios Osteomusculares Relacionados ao Trabalho (DORT).** Elaboração Maria Maeno. [et al]. Brasília: [s.n.], 2001. 36 p. Disponivel em: <http://bvsms.saude.gov.br/bvs/publicacoes/ler_dort.pdf>. Acesso em: 2019.

GUIMARÃES, B. M. D. et al. **Análise da carga de trabalho de analistas de sistemas e dos distúrbios osteomusculares**. Fisioter Mov, 2011. 115-124.

ARNOLD, S. C.; MARK, L.; GOLDTHWAITE, J. **Programming by Voice, VocalProgramming**. Proceedings of the Fourth Conference on Assistive Technologies, Arlington, 2000. Acesso em: 2019.

BURTON, C.; CHESTERTON, L. S.; DAVENPORT, G. **Diagnosing and managing carpal tunnel syndrome in primary care**. British Journal of General Practice, 2014. 262-263.

ALI, K. M.; SATHIYASEKARAN, B. W. C. **Computer Professionals and Carpal Tunnel Syndrome (CTS)**. International Journal of Occupational Satefy and Ergonomics, 2006. 319-325.

BRUNO, V.; TAM, A.; THOM, J. **Characteristics of web applications that affect usability: A review**. Proceedings of the 17th Australia conference on Computer-Human Interaction, 2005.

FLATSCHART, F. **HTML 5: Embarque Imediato**. 1°. ed. [S.l.]: Brasport, 2011.

GOODMAN, D. **Dynamic HTML: The Definitive Reference: A Comprehensive Resource for XHTML, CSS, DOM, JavaScript.** Third. ed. [S.l.]: O'Reilly Media, 2007. 1328 p.

ENGLUND, C. **Speech recognition in the JAS 39 Gripen aircraft - adaptation to speech at different G-loads**, Stockholm, 2004. Disponivel em: <http://www.speech.kth.se/publications/masterprojects/2004/christine_englund.pdf>. Acesso em: 2019.

RABINER, L. R.; JUANG, B. H. **An Introduction to Hidden Markov Models**. IEEE ASSP Magazine, p. 16, January 1986.

MASSE, M. **REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces**. 2°. ed. Sebastopol: O'Reilly Media, 2011.

LEOPOLD, J. L.; AMBER, A. L. **Keyboardless visual programming using voice, handwriting, and gesture**. IEEE Symposium on visual languages, Lawrence, 1997. Acesso em: 2019.

HOPCRAFT, J. E.; MOTWANI, R.; ULLMAN, J. D. **Introduction to automata theory, languages, and computation**, n. Second Edition, 1979.

YUI, F. S. M.; CARRO, S. A. **Reconhecimento de voz no auxílio à edição de código fonte em Java**. Colloquium Exactarum, 2017. 95-107. Disponivel em: <http://journal.unoeste.br/index.php/ce/article/view/1976/1888>. Acesso em: 2019.

TILKOV, S.; VINOSKI, S. **Node.js: Using JavaScript to Build High-Performance Network Programs**. IEEE Internet Computing, Dezembro 2010. 80-83.

CHODOROW, K. **MongoDB: The Definitive Guide**. 2°. ed. Sebastopol: O'Reilly Media, 2013.

ADORF, J. **Web Speech API**. Stockholm: KTH Royal Institute of Technology, 27 Maio 2013.

10 Anexos

10.1 Descrição dos casos de uso

A seguir são descritos os casos de uso apresentados na seção 5.3.1.

10.1.1 Caso de uso Gerar Código

Objetivo: O usuário deve ser capaz de gerar código fonte na aplicação.

Requisitos: RF01.

Pré-condições: Reconhecimento de voz deve estar em andamento.

Pós-condições: Editor de código fonte manipulado.

Fluxo Principal: 1 – Usuário inicia o reconhecimento de voz por meio de botão na página da aplicação ou atalho de teclado. 2 – Usuário inicia entrada de dados via fala. 3 – Editor de código fonte exibe código correspondente a seu comando de voz.

10.1.2 Caso de uso Reconhecer comandos de voz

Objetivo: O sistema deve ser capaz de reconhecer comandos de voz do usuário.

Requisitos: RF02, RF03.

Pré-condições: Usuário ter dado permissão para que o reconhecimento de voz seja iniciado.

Pós-condições: Comandos de voz convertidos em texto.

Fluxo Principal: 1 – Sistema capta os sinais de áudio inseridos pelo usuário e os converte em texto.

10.1.3 Caso de uso Converter comandos em código

Objetivo: O sistema deve gerar código fonte a partir de comandos de voz do usuário.

Requisitos: RF04.

Pré-condições: Reconhecimento de voz ter ocorrido com sucesso e ter gerado um comando válido pré-estabelecido no sistema.

Pós-condições: Código fonte inserido no editor de código fonte da aplicação.

Fluxo Principal: 1 – Sistema capta o resultado de texto reconhecido a partir da fala. 2 – Sistema identifica o comando de voz correspondente ao texto. 3 – Sistema realiza a ação correspondente ao comando de voz identificado.

10.1.4 Caso de uso Renderizar código

Objetivo: O usuário deve conseguir renderizar o resultado de seu código gerado no sistema.

Requisitos: RF05.

Pré-condições: Existir código no editor de código fonte da aplicação.

Pós-condições: Área de renderização da aplicação exibirá o resultado.

Fluxo Principal: 1 – Usuário inicia processo de renderização por meio de botão na página da aplicação ou atalho de teclado. 2 – Sistema extrai conteúdo do editor de código fonte da aplicação. 3 – Sistema renderiza conteúdo de código na área de renderização da página da aplicação.

10.2 Cronogramas

O Quadro 12 apresenta o cronograma das atividades desenvolvidas durante a construção do projeto de software.

Quadro 12 – Cronograma de projeto de software (2019)

Item/mês	Fev	Mar	Abr	Mai	Jun
Concepção da ideia					
Revisão bibliográfica					
Experimentos					
Escrita TCC					
Apresentação TCC1					

Fonte: Autor (2019)

O Quadro 13 apresenta o cronograma das atividades desenvolvidas durante o desenvolvimento do software deste projeto.

Quadro 13 – Cronograma de desenvolvimento de software (2020)

Item/mês	Jun	Jul	Ago	Set	Out
Implementação					
Testes da aplicação					
Escrita TCC2					
Apresentação TCC2					

Fonte: Autor (2020)