PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

IMPLEMENTAÇÃO DE UMA FERRAMENTA DE APOIO AO ENSINO DE SQL UTILIZANDO GOOGLE BLOCKLY

JANILSON ELIAS DE ALMEIDA LIMA JÚNIOR

GOIÂNIA
2021

JANILSON ELIAS DE ALMEIDA LIMA JÚNIOR

IMPLEMENTAÇÃO DE UMA FERRAMENTA DE APOIO AO ENSINO DE SQL UTILIZANDO GOOGLE BLOCKLY

Trabalho de Conclusão de Curso apresentado à Escola Politécnica, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Ciências da Computação.

Orientador(a):

Prof. Me. Rafael Leal Martins

GOIÂNIA

2021

JANILSON ELIAS DE ALMEIDA LIMA JÚNIOR

IMPLEMENTAÇÃO DE UMA FERRAMENTA DE APOIO AO ENSINO DE SQL UTILIZANDO GOOGLE BLOCKLY

Este Trabalho de Conclusão de Curso julgado adequado para obtenção o título de Bacharel em Ciências da Computação, e aprovado em sua forma final pela Escola Politécnica, da Pontifícia Universidade Católica de Goiás, em _____/_____/_____.

_____

Prof. Ma. Ludmilla Reis Pinheiro dos Santos

Coordenador(a) de Trabalho de Conclusão de Curso

Banca Examinadora:

_____

Orientador(a): Me. Rafael Leal Martins

_____

Prof. Me. Fernando Gonçalves Abadia

_____

Prof. Me. Gustavo Siqueira Vinhal

GOIÂNIA

2021

RESUMO

O objetivo deste trabalho foi construir uma ferramenta de auxílio a estudos da Linguagem de Consulta Estruturada (*Structured Query Language* – SQL) através da linguagem visual Blockly. Foi realizada uma pesquisa bibliográfica e experimental para realizar o desenvolvimento de uma aplicação *web*, a partir da Interface de Programação de Aplicação (*Application Programming Interface* - API) Google Blockly, com intuito de auxiliar alunos a melhor compreender a sintaxe e as funcionalidades da linguagem SQL através do uso de blocos virtuais que se encaixam para construir instruções de acesso e manipulação de um Banco de Dados (BD). A construção da ferramenta resultou na elaboração do site BlocklySQL, no qual o usuário possuí blocos visuais, da API Blockly, com mecânicas e funções da linguagem SQL suficientes para a elaboração de um banco. Além de uma área de exercícios em que o usuário e submetido corrigir os erros presentes no código ou apresentar a tabela desejada.

Palavras Chaves: *Structured Query Language*, Banco de Dados. Blockly. API. Estudos.

ABSTRACT

The goal of this work was to build a tool to help the study of Structured Query Language (SQL) through the Blockly visual language. A bibliographical and experimental research was carried out to develop a web application, using the Application Programming Interface (API) Google Blockly, in order to help students better understand the syntax and functionalities of the SQL language through the use of virtual blocks that fit together to build instructions for accessing and manipulating a Database (DB). The construction of the tool resulted in the elaboration of the BlocklySQL website, in which the user has visual blocks, from the Blockly API, with mechanics and functions of the SQL language sufficient for the elaboration of a database. In addition to an area of exercises in which the user is submitted to correct errors present in the code or present the desired table.

Keywords: Structured query language, database. Blockly. API. Studies.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API	*Application Programming Interface*
BD	Banco de Dados
CSS	*Cascading Style Sheets*
DBMS	*Database Management System*
GB	GigaByte
GHz	GigaHertz
HTML	*Hypertext Markup Language*
JSON	*JavaScript Object Notation*
NPM	*Node Package Manager*
RAM	*Random Access Memory*
RDBMS	*Relational Database Management System*
SQL	*Structured Query Language*
TCC	Trabalho de Conclusão de Curso
TI	Tecnologia da Informação
UI	*User Interface*
URL	*Uniform Resource Locator*
VPL	*Visual Programming Language*
VSCode	*Visual Studio Code*
W3C	*World Wide Web Consortium*
XML	*Extensible Markup Language*
YARN	*Yet Another Resource Negotiator*

SUMÁRIO

1 INTRODUÇÃO

Em cursos técnicos e superiores voltados a Tecnologia da Informação (TI), disciplinas de programação são essenciais para a formação do estudante. Estudantes de programação precisam aprender novas estruturas lógicas, semânticas e sintaxes das linguagens.

> Iniciar a atividade da programação de computadores sempre foi um desafio para professores e alunos. Além dos conceitos de lógica e algoritmos, o programador iniciante precisa também agregar o conhecimento de uma linguagem de programação com suas características próprias e uma série de estruturas e comandos em língua estrangeira (TRINDADE, 2015, p. 66).

Com o crescimento e aprimoramento das Linguagens de Programação Visuais (*Visual Programming Languages* - VPLs), os estudos iniciais para diversas linguagens de programação podem se tornar menos árduos.

Blockly é uma VPL desenvolvida em 2012 pela Google que permite realizar programação através de blocos visuais, possuindo uma ideia semelhante a montagem de peças Lego. Seu objetivo principal é funcionar como uma ferramenta para ensino de programação para iniciantes. Blockly é de uso inteiramente livre e totalmente customizável às necessidades do desenvolvedor. Sua *Application Programming Interface* (API) foi empregada no desenvolvimento de sites diversos com objetivos distintos como Wonder Workshop (makewonder.com), KodeKLIX (kodeklix.com.au), OzoBlockly (ozoblockly.com) e outros.

Blockly fornece aos usuários uma maneira alternativa de escrever os próprios scripts e configurações para um Bando de Dados (BD), por exemplo. Dessa forma, cada bloco representa uma parte do código, que será “empilhado” e traduzido em código. Portanto, mesmo que o estudante não conheça nenhuma linguagem de programação, será possível criar scripts.

A programação em bloco está mudando a maneira que as empresas trabalham com a criação de programas, pois fornece uma maneira mais simples para os usuários a compreender as principais funcionalidades na geração de um software. Além disso, por ser compatível com diversas linguagens, as organizações podem maximizar o alcance dos projetos, exportando os aplicativos para mais plataformas, por exemplo (CRONAPP, 2021).

1.1 JUSTIFICATIVA

Disciplinas iniciais de computação tendem à altos índices de evasão e reprovação, dificultando a permanência de alunos no curso (Kamiya, 2009). Dessa forma, estudos iniciais de programação tendem a ser mais complexos, porém recursos e ferramentas atuais são capazes auxiliar nesse aprendizado.

1.2 HIPÓTESE

Diante deste contexto, esse projeto visa responder a seguinte questão de pesquisa: **É possível construir uma ferramenta de auxílio a estudos da linguagem SQL através da VPL Blockly?**

1.3 OBJETIVO GERAL

O objetivo geral deste trabalho é, por meio da API Blockly, desenvolver uma aplicação web que auxilie no aprendizado ou memorização de comandos e mecânicas presentes na linguagem *Structured Query Language* (SQL).

1.4 OBJETIVOS ESPECÍFICOS

- Demostrar funcionalidades da API Blockly;
- Desenvolver uma aplicação web, com a ferramenta Blockly, para estudos de SQL a partir de blocos visuais geradores de código.

1.5 METODOLOGIA

Essa pesquisa quanto a sua natureza é um resumo de assunto, analisando informações em artigos e *web*sites publicados na área. Realizando o levantamento, como resultado das informações obtidas, de suas causas e explicações (WAZLAWICK, 2014).

Também será realizado, quanto aos procedimentos técnicos, uma pesquisa bibliográfica. A pesquisa bibliográfica, é um estudo de livros, *websites*, artigos, publicações, entre outros, já publicados.

A aplicação será uma tela com a API Blockly e uma área de código. A área da API terá os artefatos necessários para que o usuário final crie um banco de dados a partir de blocos geradores de código. A área de código apresentará ao usuário um código em formato MySQL ou *Web*SQL produzido de acordo com os blocos dispostos. A aplicação também poderá executar o código disposto em formato *Web*SQL.

Está monografia está estruturada em 4 capítulos, sendo neste Capítulo é apresentando o contexto do trabalho, justificativa, hipótese, os objetivos e a metodologia. O Capítulo 2 traz o referencial teórico com conceitos e definições de temas relacionados na monografia. No Capítulo 3 é demonstrado as etapas do desenvolvimento e explicação do uso da aplicação. Finalmente, o Capítulo 4 traz as considerações finais deste trabalho e as sugestões para trabalhos futuros.

2 REFERENCIAL TEÓRICO

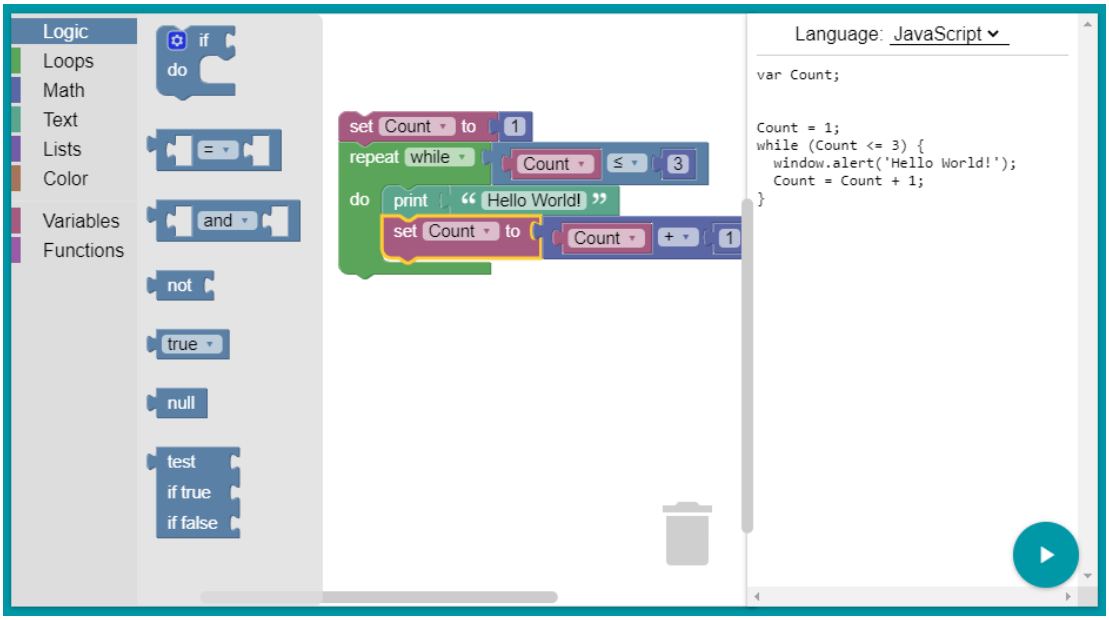
Este capítulo apresentará conceitos, definições e configurações de uso das três ferramentas centrais do projeto: Blockly, aplicações *web* e Bancos de dados (BD).

2.1 BLOCKLY

Blockly, uma biblioteca escrita totalmente em JavaScript e *open source,* adiciona a aplicações *web* e móveis um editor de código com blocos visuais. Sua arquitetura suporta mais de 40 idiomas incluindo versões árabes e hebraicas com escrita da direita para esquerda. A aplicação executada totalmente no lado cliente pelo navegador, dispensando conexão contínua com o servidor. Seu modelo *open source* concede ao desenvolvedor total manipulação, adaptação e distribuição de projetos e afins que utilizam da biblioteca. Com isso, projetos que utilizam da API podem ser totalmente únicos com blocos e geradores de códigos próprios.

Blockly usa do método de arrastar e soltar blocos gráficos que representam variáveis, expressões lógicas, *loops* e demais funções pré-determinadas da programação. Os blocos se encontram em uma *toolbox* (caixa de ferramentas) lateral e são posicionados pelo usuário em uma *workspace* (área de trabalho) (Figura 1). Nessa *workspace* blocos interligados criam uma sequência lógica de código que é interpretada e transformada em saídas textuais, visuais ou mecânicas com base no algoritmo construído. Em seu código fonte, Blockly já possuí a funcionalidade de exportar os códigos visuais em saídas textuais nas linguagens JavaScript, Python, PHP, Lua e Dart.

Figura 1 - Exemplo de interface da *toolbox* e *workspace* Blockly.



Fonte: Blockly, 2020

Com Blockly o usuário final fica restrito a criar uma sequência lógica correta com os blocos necessitando apenas do seu conhecimento sobre a sintaxe da programação sem se preocupar com erros de sintaxe. “Da perspectiva do usuário, Blockly é uma maneira intuitiva e visual de construir código. Da perspectiva do desenvolvedor, Blockly é uma *User Interface* (UI) pronta para a criação de uma linguagem visual que emite código gerado pelo usuário sintaticamente correto.” (BLOCKLY API, 2020).

2.1.1 Incorporando Blockly em aplicações *Web*

Para que o editor de códigos do Blockly seja incorporado em projetos e aplicações, sua API deve ser importada da máquina ou referenciada por meio de *scripts*. *Scripts* são uma série de comandos em um arquivo usados para automatizar processos em uma máquina ou gerar páginas *web*. Em seu guia Blockly lista três principais meios de uso da sua biblioteca (BLOCKLY API, 2020).

Uma opção de uso da biblioteca é importando-a diretamente para a máquina do desenvolvedor. Para isso, ela deve ser instalada por meio dos gerenciadores de pacotes *Node Package Manager* (Npm) ou *Yet Another Resource Negotiator* (Yarn). Esses gerenciadores são repositórios usados para a publicação de projetos e ferramentas prontas em *open source*, onde constantemente os códigos são atualizados por seus colaboradores. Usando deste método apenas pacotes padrões serão incialmente importados. Os demais pacotes podem ser

pesquisados nos gerenciadores. Blockly recomenda que se use dos gerenciadores para que sua biblioteca seja mais facilmente atualizada por seus programadores.

Outro meio é pelo uso da Unpkg (UNPKG, 2021), que é um site de distribuição de códigos abertos para Npm. Unpkg será um substituto para caso não tenha gerenciadores de pacotes instalado na máquina e seu uso é por *scripts* referenciados por *Uniform Resource Locator* (URL) (Figura 2). Unpkg sempre manterá a versão mais recente do código publicado, sendo recomendado para demonstração de projetos ou experimentos.

Figura 2 - Exemplo uso do Blockly por Unpkg.

Fonte: Elaborado pelo autor

O último método citado pelo guia é a de exportar o código fonte disponibilizado no GitHub (GITHUB, 2021). Sendo esse o modo menos recomendado tendo o programador que sincronizar manualmente com o repositório regularmente. Seu uso é semelhante ao do Unpkg, sendo diferenciado pelo fato de os *scripts* serem pegos diretamente da máquina e não por URLs.

2.1.2 Configurações do Blockly

As configurações do Blockly são diversas, desde a interface alterando cores da escrita, até as funcionalidades dos blocos. Como por exemplo: alterando: alterando comandos pré-existentes ou desenvolvendo novos, fazendo com que a ferramenta seja capaz ser totalmente customizável de acordo com as vontades de seu aplicador.

Entre as configurações mais diretas disponibilizadas estão as opções de injeção da *workspace*. Elas são adicionadas e configuradas quando se cria a *workspace* (Figura 3). São mais de vinte configurações de injeção descritas pelo guia que alteram e facilitam o uso da área de trabalho. Entre as opções estão:

- *Toolbox*, a principal ferramenta de injeção, com ela adiciona-se o menu lateral de blocos que ficará disponível para o usuário final. No código ela é especificada a partir de um *Extensible Markup Language* (XML);
- *Grid*, é uma malha de pontos que ajuda na visualização da pilha de blocos. Os pontos podem ser espaçados e coloridos conforme a necessidade do desenvolvedor;
- *HorizontalLayout*, atribui à *toolbox* uma visualização horizontal caso especifique como verdadeiro, caso falso manterá o padrão vertical.
- *ReadOnly*, previne que o usuário faça modificações na *workspace* ocultando a *toolbox*;
- *Scrollbars*, remove barras de rolagem horizontais e verticais da *workspace*;
- *MaxTrashcanContents*, determina um máximo de itens que ficarão no histórico da lixeira, por padrão são 32 itens;

Figura 3 – Personalização de injeção na criação da *workspace*.

```
<script>
    var blocklyArea = document.getElementById('blocklyArea');
    var blocklyDiv = document.getElementById('blocklyDiv');
    var workspace = Blockly.inject(blocklyDiv, {
        toolbox: document.getElementById('toolbox'),
        scrollbars: true,
        grid:{
            spacing: 20,
            length: 3,
            colour: '#ccc',
            snap: true
        },
        zoom:{
            controls: true,
            wheel: true,
            startScale: 1.0,
            maxScale: 3,
            minScale: 0.5,
            scaleSpeed: 1.2,
            pinch: true
        },
    });
```

Fonte: Elaborado pelo autor

2.1.3 Configuração dos blocos

As configurações de injeção são as mais claras de se aplicar em projetos. Além delas também é possível alterar o tamanho da *workspace* (tornando-a recursiva), aplicar novos temas, adicionar traduções, navegar por teclado, entre outras opções. Entre esses e outras configurações listadas no guia a mais importante é a adição e personalização de blocos e geração de código.

Os blocos padrão presentes na API nem sempre suprem as necessidades do desenvolvedor cabendo a ele modificar ou desenvolver novos. Esses novos blocos são elaborados em arquivos JavaScript e escritos ou na própria linguagem JavaScript ou em JavaScript *Object Notation* (JSON). Esse último é um modo de notação mais simples de entender para máquinas e humanos. “Embora o Blockly defina uma série de blocos padrão, a maioria dos aplicativos precisa definir e implementar pelo menos alguns blocos de domínio relevantes” (BLOCKLY API, 2020).

A elaboração de um novo bloco é definida, no arquivo JavaScript, pela chamada do objeto Blockly.Blocks e seguida pelo nome do novo bloco entre colchetes. Após a chamada do objeto atribui-se a ele uma função entre chaves. Dentro dessa função aplicasse textos, entrada de dados com nome e validações, tipo de saída, coloração, informações sobre o bloco, links de ajuda e demais necessidades do desenvolvedor (Figura 4).

Figura 4 – Definição de um novo bloco usando JSON.

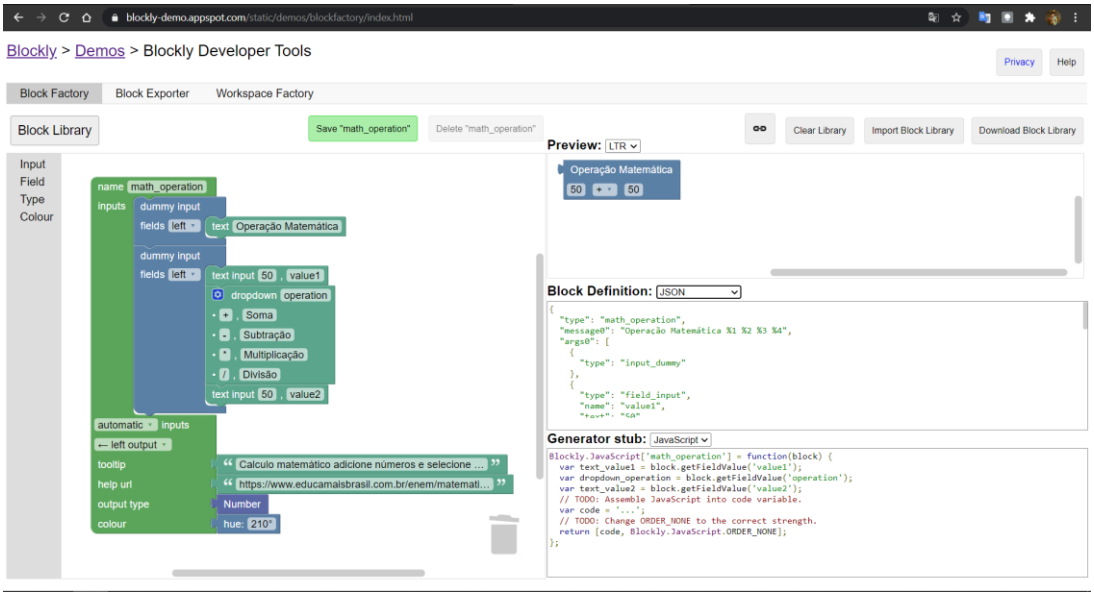
JSON JavaScript

```
Blockly.Blocks['string_length'] = {
  init: function() {
    this.appendValueInput('VALUE')
        .setCheck('String')
        .appendField('length of');
    this.setOutput(true, 'Number');
    this.setColour(160);
    this.setTooltip('Returns number of letters in the provided text.');
    this.setHelpUrl('http://www.w3schools.com/jsref/jsref_length_string.asp');
  }
};
```

Fonte: Blockly, 2020

Para facilitar a escrita dos novos blocos o próprio Blockly fornece o site *Blockly Developer Tools* (BLOCKLY API, 2020),, onde o desenvolvedor pode elaborar a interface do bloco desejado (Figura 4). O site utiliza da própria API Blockly para a criação deles (Figura 5).

Figura 5 – Criação de um bloco customizado



Fonte: Blockly Developer Tools, 2020

2.1.4 Geração de códigos

Os blocos Blockly geralmente devem ser traduzidos para uma linguagem previamente determinada gerando códigos totalmente corretos para a linguagem. Por padrão em seu código fonte algumas linguagens como JavaScript, Python e PHP já possuem essa geração de código, porém novas linguagens podem ser descritas e adaptadas pelo desenvolvedor.

Para que os blocos visuais gerem códigos executáveis, Blockly, disponibiliza da função *workspaceToCode*, que ao ser chamada, é atribuída a *workspace* como parâmetro. Essa função realiza uma leitura dos blocos dispostos pelo usuário e os traduz na linguagem previamente especificada (Figura 6). Contudo nem todas as ações da linguagem ou as ações desejadas pelo desenvolvedor terão disponibilidade, necessitando do aplicador criar suas próprias gerações.

Figura 6 – Transformando blocos da *workspace* em código

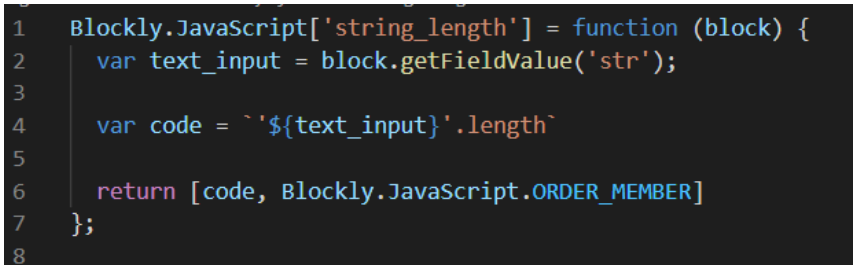
```
182
183     function gerarCodigo() {
184         var linguagem = document.getElementById("linguagens");
185
186         switch (linguagem.value) {
187             case "js":
188                 var code = Blockly.JavaScript.workspaceToCode(this.workspace);
189                 break;
190             case "python":
191                 var code = Blockly.Python.workspaceToCode(this.workspace);
192                 break;
193             case "php":
194                 var code = Blockly.PHP.workspaceToCode(this.workspace);
195                 break;
196             case "lua":
197                 var code = Blockly.Lua.workspaceToCode(this.workspace);
198                 break;
199             case "dart":
200                 var code = Blockly.Dart.workspaceToCode(this.workspace);
201                 break;
202             default:
203                 break;
204         }
205
206         if (code)
207             return code;
208     }
209
```

Fonte: Elaborado pelo autor

Não a uma documentação clara para que linguagens, além das já presentes, sejam usadas, necessitando que o desenvolvedor utilize as pré-existentes como base para outras. O site *Blockly Developer Tools*, além de auxiliar na produção dos blocos visuais customizados, também demostra o modo correto de se gerar o código na linguagem desejada (Figura 5). Utilizando dessa estrutura lógica, sobrará ao desenvolvedor criar a sintaxe 100% correta que o bloco deverá descrever.

A geração de código de um bloco é definida pela chamada do objeto Blockly seguido pela linguagem desejada e o nome do bloco que gerara o código entre colchetes. Após a chamada do objeto atribui-se a ele uma função com o parâmetro block. Dentro dessa função pega-se os valores inseridos no bloco com base nos nomes definidos no mesmo. Tendo as variáveis, basta o programador desenvolver o código que deverá ser descrito e retorná-lo (Figura 7).

Figura 7 – Geração JavaScript de um bloco.



```
Blockly.JavaScript['string_length'] = function (block) {
  var text_input = block.getFieldValue('str');

  var code = `'${text_input}'.length`

  return [code, Blockly.JavaScript.ORDER_MEMBER]
};
```

Fonte: Elaborado pelo autor

2.2 APLICAÇÕES *WEB*

Blockly, por ser uma aplicação mais dinâmica, tem com seu maior foco aplicações *web*, se diferenciando de um site *web*, que fornecem páginas mais estáticas, com o objetivo der ser mais informativo. Conforme Noleto (2020), “a aplicação *web* diz respeito a uma solução que é executada diretamente no browser (ou navegador), não sendo preciso realizar uma instalação na máquina do usuário.”.

Essas aplicações, de modo geral, são divididas entre *front-end* e *back-end*. *Front-end* é parte visual do site, responsável pela comunicação com o usuário. Geralmente utiliza de *Hypertext Markup Language* (HTML), *Cascading Style Sheets* (CSS) e JavaScript para a organização de suas estruturas, que será explicado adiante. Já o *back-end* é a parte lógica do site, sendo responsável por suas regras de negócio. Para o seu desenvolvimento utilizam de linguagens como Python, Ruby, Java, entres outras.

2.2.1 HTML, CSS e JavaScript

HTML é uma estrutura para a criação de páginas *web*. Seu objetivo é ditar ao navegador como a página deve ser exibida, descrevendo a ele seções como títulos, cabeçalhos, corpo principal, containers, parágrafos e links. O HTML não é uma linguagem de programação, isto é, não é capaz de elaborar funções dinâmicas, sendo seu objetivo a organização e formatação de documentos, similar ao Microsoft Word (ANDREI, 2021).

CSS é uma linguagem de estilização de documentos HTML e XML. CSS é uma adição ao HTML, ele incrementa elementos descritos no mesmo com mudança de fontes, cores, posicionamentos, comprimentos, *backgrounds* e muito mais. Sua especificação é padronizada em navegadores *web* de acordo com as especificações da *World Wide Web Consortium* (W3C).

JavaScript, umas das tecnologias mais padrões de desenvolvimento *web,* é uma linguagem leve, interpretada e baseada em objetos. Sua principal funcionalidade é a implementação de funcionalidades complexas em páginas HTML.

> Toda vez que uma página da *web* faz mais do que simplesmente mostrar a você informação estática — mostrando conteúdo que se atualiza em um intervalo de tempo, mapas interativos ou gráficos 2D/3D animados etc. — você pode apostar que o JavaScript provavelmente está envolvido (MDN, 2021).

2.3 BANCO DE DADOS

A API Blockly é capaz de se adaptar a infinitas linguagens, assim como linguagens SQL para criação de BD. Um BD é definido por um agrupamento de dados ou informações inter-relacionados ou não, que são armazenados de forma eletrônica em uma máquina, no intuito de serem posteriormente conferidos ou melhor protegidos. Um bom BD resulta diversos benefícios para a empresa, impactando positivamente no entrosamento e produtividade da equipe e, por conseguinte, nos resultados alcançados (SOUZA, 2020).

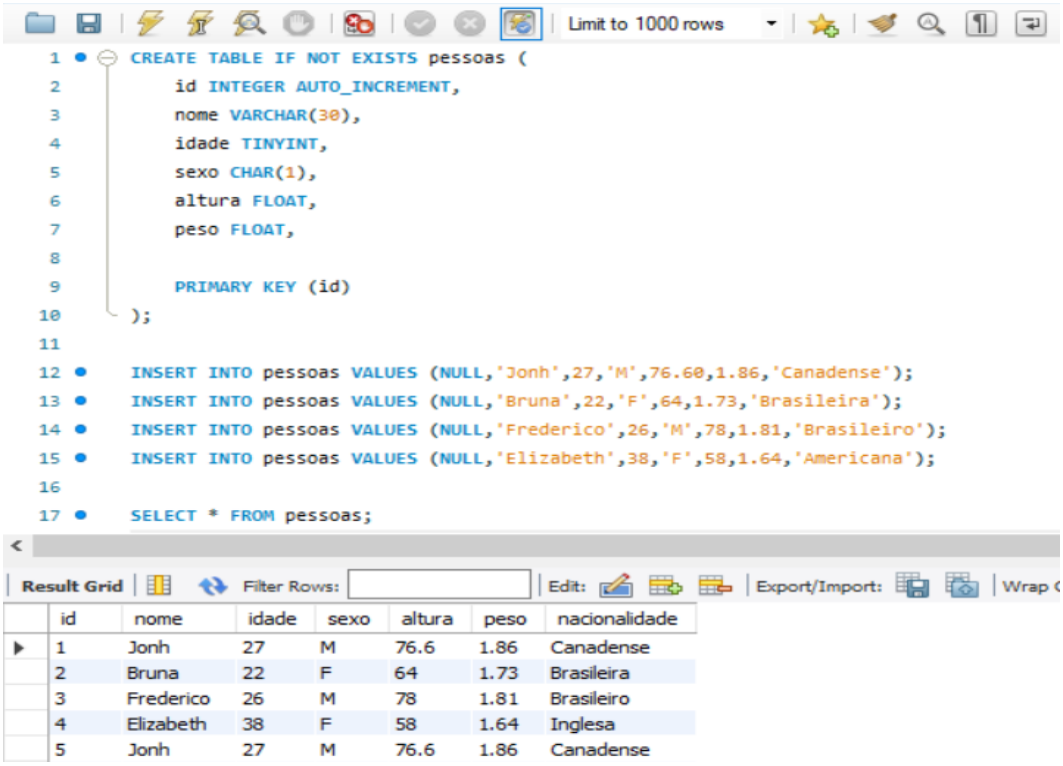
> Os dados nos tipos mais comuns de bancos de dados em operação hoje são normalmente modelados em linhas e colunas em uma série de tabelas para tornar o processamento e a consulta de dados eficientes. Os dados podem então ser facilmente acessados, gerenciados, modificados, atualizados, controlados e organizados. A maioria dos bancos de dados usa *Structured Query Language* (SQL) para escrever e consultar dados (ORACLE, 2021).

Para a melhor manipulação dos BDs, existem sistemas moderadores que geralmente são os *Database Management System* (DBMS) ou *Relational Database Management System* (RDBMS). Esses sistemas possuem uma interface entre o banco e o usuário final, com objetivo de melhorar a experiência na manipulação de bancos, facilitando a criação de bancos, o gerenciamento dados e tabelas e muito mais. Entre RDBMS mais populares estão MySQL, PostgreSQL, Microsft Access e Oracle, sendo todos esses gerenciadores cliente-servidor. Na necessidade de uso apenas no lado cliente existem as DBMS *Web*SQL, *Indexed Database* e *LocalStorage*.

2.3.1 MySQL

MySQL (Figura 8), uma das RDBMS mais populares, é gerenciada pela Oracle e totalmente *open source*. Seu BD é do tipo relacional e seu modelo é cliente-servidor. Sua alta popularização advém da sua flexibilidade, facilidade de uso, alto desempenho e segurança dos dados, além de grandes empresas como Tesla, Youtube, Facebook e Netflix utilizarem da mesma.

Figura 8 – Criação de um BD pelo MySQL.

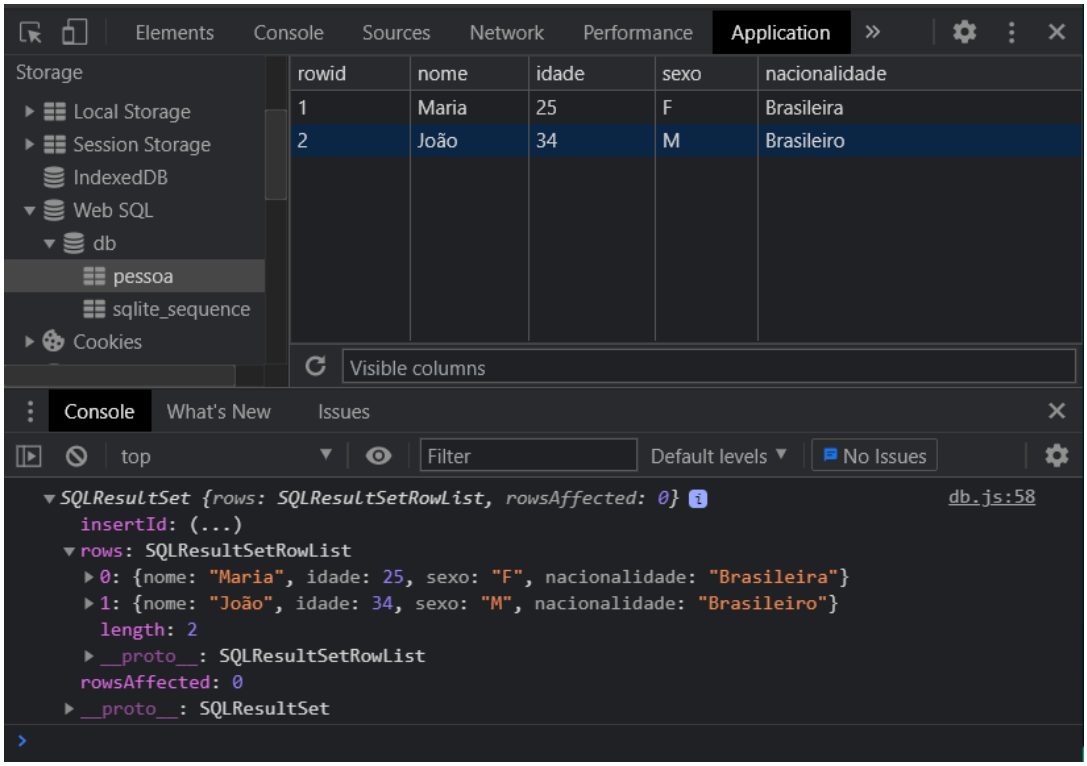


Fonte: Elaborado pelo autor

2.3.2 *Web*SQL

*Web*SQL (Figura 9) é uma DBMS que utiliza um conjunto de APIs para manipulação de banco no lado cliente da aplicação. *Web*SQL dispõe da mesma linguagem suportada pela versão 3.6.19. do Sqlite. Mesmo sendo descontinuado em 2010 pela W3C, a DBMS, ainda possuí suporte nas versões mais recentes dos navegadores Opera, Safari e Chrome, sendo este último com suporte *desktop* e *mobil*e.

Figura 9 – Visualização de dados do *Web*SQL pelo navegador.



Fonte: Elaborado pelo autor

A visualização do banco gerado no WebSQL e visível pelo navegado ao inspecionar a página e selecionar a aba de aplicação, como visto na Figura 9. Nessa aba em armazenamento tem o campo WebSQL, no qual é possível ver os bancos criados e as tabelas presentes em cada banco.

3 DESENVOLVIMENTO DA APLICAÇÃO

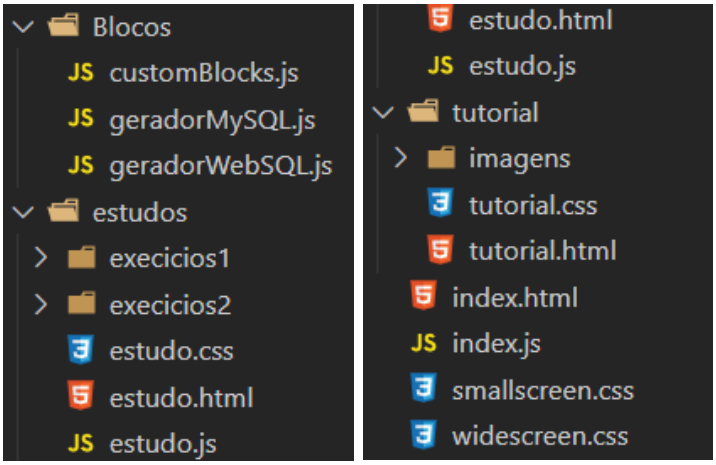
O desenvolvimento da aplicação será no editor de código *Visual Studio Code* (VSCode). O computador utilizado será um notebook Acer, com Sistema Operacional Microsoft Windows 10 Home, processador Intel Core i7-9750H 2.60 GigaHertz (GHz), 16 GigaByte (GB) de Memória de Acesso Randômico (RAM).

A aplicação *web* foi totalmente desenvolvida ao redor da API Blockly juntamente com os mecanismos de desenvolvimento *web* HTML, CSS e JavaScript, além do auxílio do framework Bootstrap para melhor usabilidade do usuário. A partir da API foram desenvolvidos blocos capazes de gerar códigos nos dialetos *Web*SQL e MySQL. Para a execução dos códigos gerados é utilizado o *Web*SQL, seu uso é devido ser um executor SQL *client side* não necessitando de um servidor para a execução de seus comandos, sendo-os executados totalmente no navegador do usuário. A aplicação foi hospedada pelo *hosting* da Google, Firebase.

3.1 ESTRUTURA DOS ARQUIVOS

A aplicação foi totalmente elaborada no editor de código-fonte VSCode, desenvolvido pela Microsoft. A partir do VSCode foram criados arquivos HTML, CSS, JavaScript e as pastas Blocos, onde possuí as customizações dos blocos, a pasta Estudos onde possuí os exercícios e a pasta Tutorial. A pasta Estudos é subdividida em exercicos1 e exercicios2, com 5 outras pastas cada onde a os arquivos HTML e JavaScript (Figura 10).

Figura 10 – Arquivos utilizados na aplicação *web*.



Fonte: Elaborado pelo autor

3.2 CRIAÇÃO DOS BLOCOS

O desenvolvimento dos blocos foi realizado no arquivo JavaScript customBlocks.js. Neste arquivo é definido, para cada bloco de textos, campos de entrada de dados, coloração, conexões, *tooltips* e links de ajuda (Figura 11).

Figura 11 – Criação do bloco start_sql.

```
26  Blockly.Blocks['start_sql'] = {
27      init: function () {
28          this.appendDummyInput()
29              .appendField("Conectar ao banco")
30              .appendField(new Blockly.FieldTextInput("database", validator), "db_name");
31          this.setNextStatement(true, "create_table");
32          this.setColour(120);
33          this.setTooltip("Iniciando projeto SQL");
34          this.setHelpUrl("https://www.w3schools.com/sql/default.asp");
35      }
36  }
```

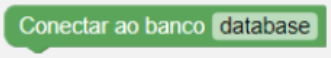
Fonte: Elaborado pelo autor

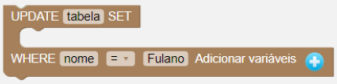
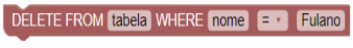
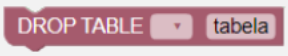
Os blocos criados envolvem conexão em bancos, criação, exclusão e alteração de dados e tabelas e apresentação de tabelas (Tabela 1). Cada bloco possuí uma restrição de conexão evitando erros na execução do SQL. Por exemplo, o bloco relacionado ao comando *PRIMARY KEY*, usando na criação de uma tabela, só pode ser conectado ao bloco relacionado a criação de variável na tabela, ao bloco relacionado ao comando *FOREING KEY* e a ele mesmo. O bloco *PRIMARY KEY* também não pode receber nenhuma outra conexão que não seja a dele (Figura 12).

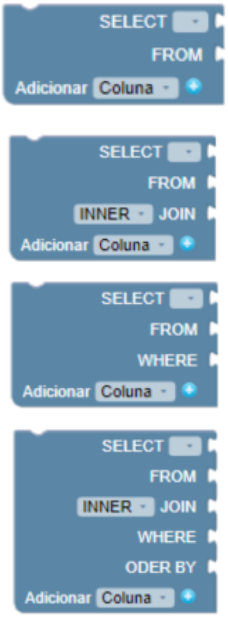
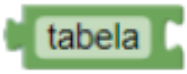
Figura 12 – Tentativa de conexão de um bloco em local inapropriado.

Fonte: Elaborado pelo autor

Tabela 1 – Blocos criados e suas funções.

BLOCO	FUNÇÃO	REPRESENTAÇÃO
start_sql	Responsável pelo acesso ou criação de um banco de dados. Obrigatoriamente o primeiro bloco da pilha e necessário para utilizar qualquer comando SQL posterior. Deve-se passar o nome do banco a ser acessado.	
create_table	Cria uma tabela. Possuí um campo de seleção que pode ser vazio ou IF EXISTS, além de três botões que facilitam a adição dos blocos table_var, table_var_pk e table_var_fk. Deve-se passar o nome desejado para a tabela.	
table_var	Bloco responsável por adicionar novos dados no bloco create_table. Deve-se informar o nome da variável, tipo e possível atributo.	
table_var_pk	Bloco responsável por definir uma variável como Primary Key no bloco create_table. Deve-se informar a variável desejada	
table_var_fk	Bloco responsável por definir uma variável como Foreign Key no bloco create_table. Deve-se informar a variável desejada, a tabela a ser referenciada e a variável relacionada	
insert_table	Insere novos dados a uma tabela. Possui dois botões que facilitam a adição dos blocos insert_var e insert_var_default. Deve-se passar o nome da tabela em que serão	

	adicionados os dados e caso necessário a ordem ou quais dados serão inseridos.	
insert_var	Bloco responsável por inserir novos dados a uma tabela no bloco insert_table. Deve-se passar o valor do dado.	
insert_var_default	Bloco responsável por inserir dados padrões a uma tabela no bloco insert_table. Deve-se selecionar o valor do dado.	
update_table	Altera os dados de uma tabela. Possui um botão que facilita a adição do bloco update_var. Deve-se passar o nome da tabela e a condição para a alteração do dado.	
update_var	Bloco responsável por determinar qual variável terá o valor alterado e seu novo valor no bloco update_table. Deve-se passar o nome da variável e seu novo valor.	
alter_table	Adiciona ou remove uma variável de uma tabela. Deve-se passar o nome da tabela, selecionar a ação desejada, o nome da variável a ser adicionada e seu tipo.	
delete_from_table	Excluí um dado inserido a uma tabela. Deve-se passar o nome da tabela e a restrição de exclusão do dado.	
drop_table	Deleta totalmente uma tabela. Possuí um campo de seleção que pode ser vazio ou *IF EXISTS*. Deve-se passar o nome da tabela.	

select	Seleciona uma tabela para apresentação de seus dados. Possuí um campo de seleção que pode ser vazio ou *DISTINCT*, além de um botão que quando apertado identifica a opção selecionada e adiciona os blocos select_var, select_from, select_join, select_where e select_orderby. O bloco possuí 4 diferentes versões que apenas restringem alguns comandos deixando com uma melhor usabilidade e entendimento.	
select_var	Restringe quais dados serão apresentados ao realizar uma pesquisa no bloco select. Deve-se informar os dados que serão pesquisados ou se apresentara todos os dados.	
select_from	Determina a tabela que será pesquisada no bloco select. Deve-se informar o nome da tabela desejada.	
select_join	Realiza uma junção de tabelas ao pesquisar no bloco select. Pode ser definida entre *inner*, *left*, *right* e *full join*. Deve-se informar a qual tabela terá a união e os dados semelhantes as tabelas.	
select_where	Cria uma restrição de pesquisa no bloco select. Deve-se informar o nome da variável, o operador e sua restrição.	

select_orderby	Altera a ordem de apresentação dos dados. Deve-se informar a variável e sua ordem de apresentação.	

Fonte: Elaborado pelo autor

3.3 GERAÇÃO DE CÓDIGO SQL

A geração dos códigos é realizada nos arquivos JavaScript geradorMySQL e gerador*Web*SQL. Esses arquivos utilizam dos geradores de linguagem JavaScript e Lua já presentes na API Blockly, respectivamente, para sustentar a geração de código MySQL e *Web*SQL. Isso advém do fato de que não à uma documentação clara para a criação de uma própria linguagem, sendo recomendado pelos próprios desenvolvedores e usuários da biblioteca Blockly o uso de uma das cinco linguagens pré-existentes como suporte para uma nova.

Para a geração dos códigos, foi escrito a semântica correta de cada ação dos blocos presentes para o usuário final para cada uma das linguagens desenvolvidas (Figura 13). Alguns geradores para a linguagem *Web*SQL apresentam um adicional no código para apresentação e manipulação dos dados, para que eles sejam visivelmente dispostos em uma tabela na aplicação *web*.

Figura 13 – Geração de código do bloco create_table.

```
Blockly.Lua['create_table'] = function (block) {
    var table_name = block.getFieldValue('table_name');
    var option = block.getFieldValue('option');
    var table_var = Blockly.Lua.statementToCode(block, 'table_var');

    // Organizar semântica PRIMARY KEY
    if (table_var.includes("PRIMARY KEY")) {
        let array = table_var.split("PRIMARY KEY")
        table_var = array[0] + 'PRIMARY KEY (';
        for (let i = 1; i < array.length; i++) {
            while (array[i].includes(' '))
                array[i] = array[i].replace(' ', '');

            table_var += array[i].replace('\n', '').replace('(', '').replace(')', '');
        }
    }

    var code = `\nCREATE TABLE${option} ${table_name} (${table_var}`;
    if(code.includes('PRIMARY KEY'))
        code = code.slice(0, -1) + `)\n);\n`;
    else
        code = code.slice(0, -1) + `\n);\n`;

    return code;
};
```

Fonte: Elaborado pelo autor

As gerações de código dos blocos não apenas traduzem os blocos e retornam código, muitos blocos também possuem funcionalidades a mais que organizam a geração do código

para que seja a mais correta. O bloco insert_var, por exemplo, realiza uma verificação se o valor fornecido é um número ou não, caso seja não seja numeral a descrição do valor terá aspas duplas, indicando ser uma *string*. Este bloco também possuí ações que substituem os primeiros e os últimos caracteres, deixando apenas os necessários (Figura 14).

Figura 14 – Geração de código do insert_var.

```
Blockly.JavaScript['insert_var'] = function (block) {
    var var_input = block.getFieldValue('var_input');
    var insert_var = Blockly.JavaScript.valueToCode(block, 'insert_var', Blockly.JavaScript.ORDER_ATOMIC);

    if (isNumber(var_input))
        variaveis[count] = var_input;
    else
        variaveis[count] = `"${var_input}"`;

    insert_var = insert_var.replace('(', ', ')
    insert_var = insert_var.slice(0, -2)

    var code = `\n  let var${count} = `;
    code += var_input + insert_var;

    count++;

    return [code, Blockly.JavaScript.ORDER_NONE];
};
```

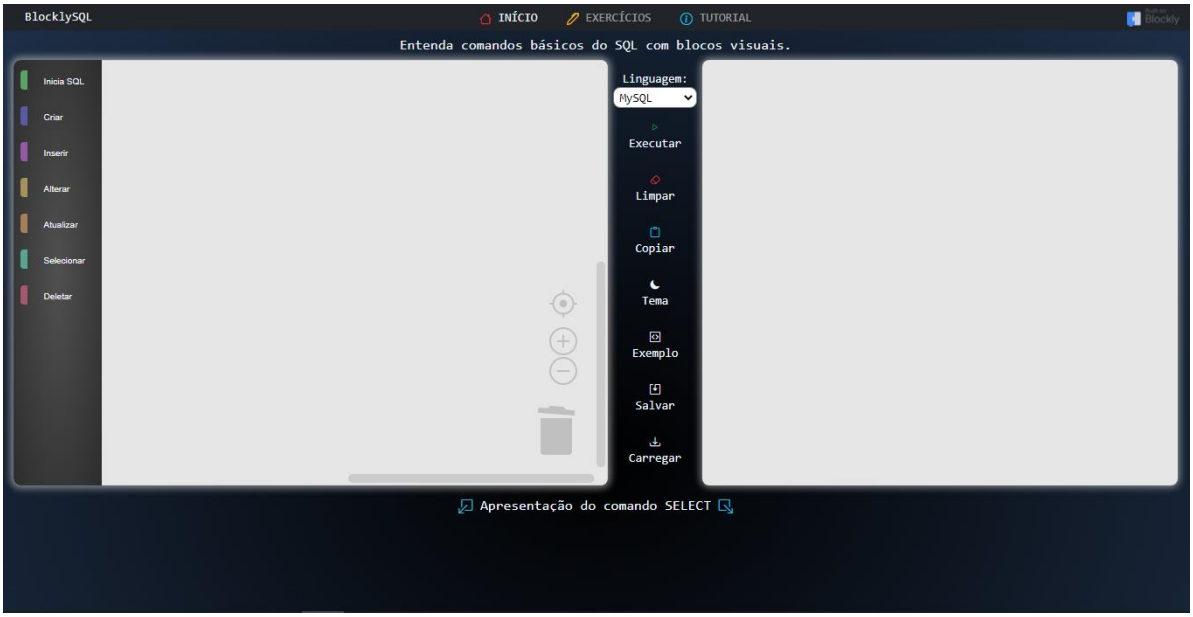
Fonte: Elaborado pelo autor

3.4 TELA INÍCIAL

A página inicial do site é representada por index.html, conforme a Figura 10. Essa página tem como objetivo ser uma área *sandbox*, ou seja, para uso livre dos blocos geradores de SQL. Com ela o usuário pode criar códigos SQL, a partir dos blocos visuais, nas linguagens MySQL e *Web*SQL para desenvolver um BD totalmente funcional ajudando-o assim a conhecer e entender melhor uma variedade de comandos SQL.

A tela inicial é composta por um menu superior que concede acesso as demais páginas da aplicação, Exercícios e Tutorial. No canto direito do menu há o logo Blockly que, ao clicar, abre a página oficial do Blockly. O restante da página e representado pela *workspace* do Blockly, um campo de texto no qual é apresentado o código gerado e uma tabela onde é apresentada a tabela gerada ao se usar o comando SELECT do SQL (Figura 15).

Figura 15 – Página inicial Blockly SQL.



Fonte: Elaborado pelo autor

Na página Início à também, entre a *workspace* e a área de código, um campo para seleção da linguagem desejada e sete botões com ações especificas, esses botões são:

- Executar: Executa, na linguagem *Web*SQL, o código gerado;
- Limpar: Apaga todos os blocos da *workspace;*
- Copiar: Copia o código presente na área de código para a área de transferência;
- Tema: Troca a cor do *background* da workspace e da área de código entre claro e escuro;
- Exemplo: Gera na *workspace* um código exemplo com conexão ao banco, criações e exclusões de tabelas, inserção de dados e apresentação de dados;
- Salvar: Salva no navegador do usuário os blocos presentes na *workspace;*
- Carregar: Pega do navegador os blocos que foram salvos pelo usuário é os colocar na *workspace*;

3.4.1 Workspace e Toolbox

Para a página inicial do Blockly SQL a workspace não foi muito modificada perante a configuração padrão fornecido pelo Blockly. Dessa forma, além da definição da toolbox, foi especificado a forma de renderização Geras para os blocos, que realiza uma leve modificação

visual nos blocos. Sendo também, habilitado as barras de rolagem, redução visual dos blocos e lixeira, configurações de zoom e visão vertical da toolbox (Figura 16).

Figura 16 – Configuração da *workspace* na tela inicial.

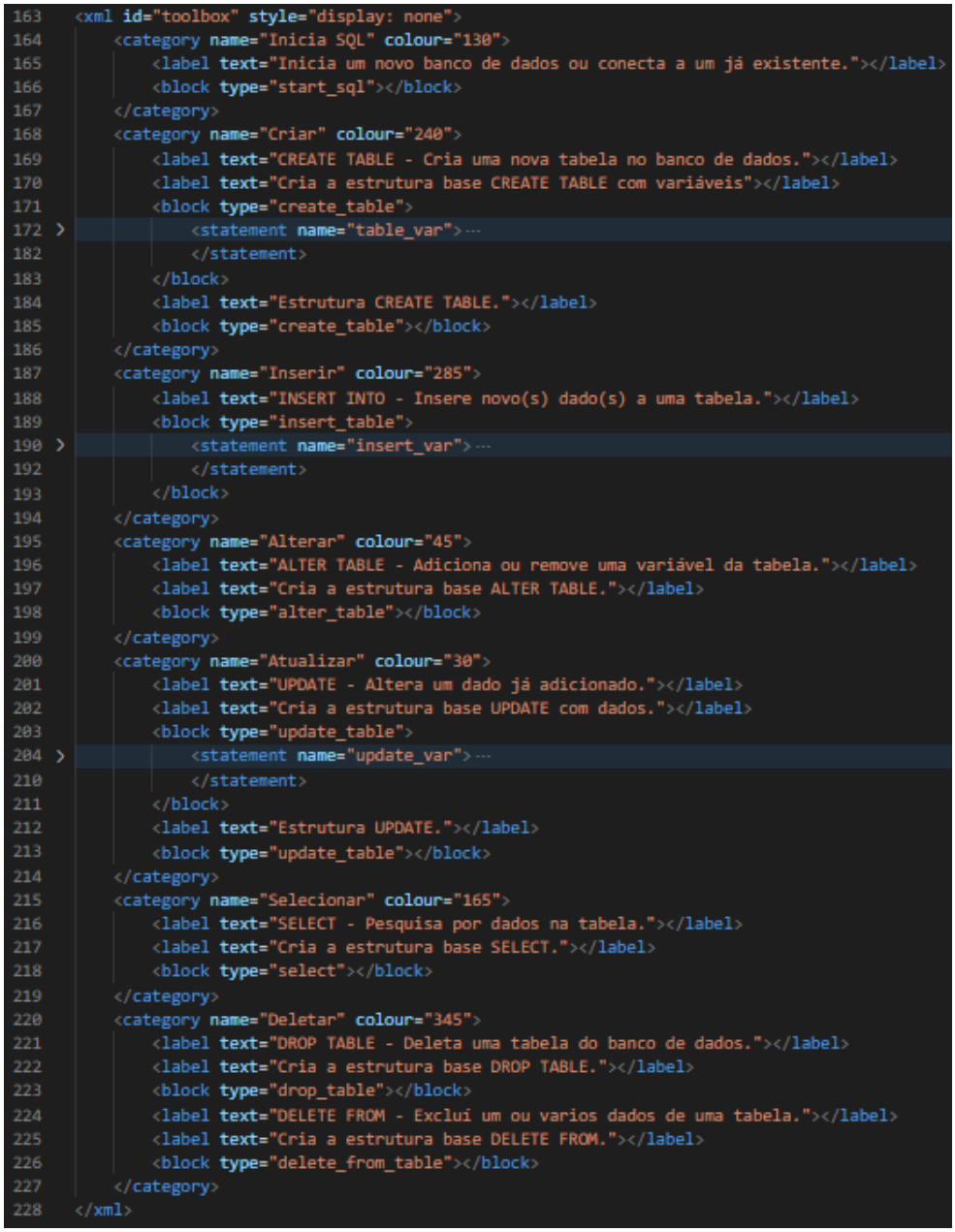
```
var blocklyDiv = document.getElementById('blocklyDiv');
var workspace = Blockly.inject(blocklyDiv, {
    toolbox: document.getElementById('toolbox'),
    scrollbars: true,
    renderer: 'geras',
    collapse: true,
    horizontalLayout: false,
    trashcan: true,
    zoom: {
        controls: true,
        wheel: true,
        startScale: 0.6,
        maxScale: 3,
        minScale: 0.5,
        scaleSpeed: 1.2,
        pinch: true
    }
});
```

Fonte: Elaborado pelo autor

Para a *toolbox* foi aplicado um XML onde os blocos foram separados por cores e nas categorias (Figura 17):

- Inicia SQL: Possui o bloco start_sql;
- Criar: Possui o bloco create_table e uma variação com três blocos table_var já acoplados;
- Inserir: Possui o bloco insert_table;
- Alterar: Possui os blocos alter_table;
- Atualizar: Possui o bloco update_table e uma variação com três blocos update_var já acoplados;
- Selecionar: Possui o bloco select;
- Deletar: Possui os blocos drop_table e delete_from_table.

Figura 17 – *Toolbox* da aplicação.



```
<xml id="toolbox" style="display: none">
    <category name="Inicia SQL" colour="130">
        <label text="Inicia um novo banco de dados ou conecta a um já existente."></label>
        <block type="start_sql"></block>
    </category>
    <category name="Criar" colour="240">
        <label text="CREATE TABLE - Cria uma nova tabela no banco de dados."></label>
        <label text="Cria a estrutura base CREATE TABLE com variáveis"></label>
        <block type="create_table">
            <statement name="table_var">···
            </statement>
        </block>
        <label text="Estrutura CREATE TABLE."></label>
        <block type="create_table"></block>
    </category>
    <category name="Inserir" colour="285">
        <label text="INSERT INTO - Insere novo(s) dado(s) a uma tabela."></label>
        <block type="insert_table">
            <statement name="insert_var">···
            </statement>
        </block>
    </category>
    <category name="Alterar" colour="45">
        <label text="ALTER TABLE - Adiciona ou remove uma variável da tabela."></label>
        <label text="Cria a estrutura base ALTER TABLE."></label>
        <block type="alter_table"></block>
    </category>
    <category name="Atualizar" colour="30">
        <label text="UPDATE - Altera um dado já adicionado."></label>
        <label text="Cria a estrutura base UPDATE com dados."></label>
        <block type="update_table">
            <statement name="update_var">···
            </statement>
        </block>
        <label text="Estrutura UPDATE."></label>
        <block type="update_table"></block>
    </category>
    <category name="Selecionar" colour="165">
        <label text="SELECT - Pesquisa por dados na tabela."></label>
        <label text="Cria a estrutura base SELECT."></label>
        <block type="select"></block>
    </category>
    <category name="Deletar" colour="345">
        <label text="DROP TABLE - Deleta uma tabela do banco de dados."></label>
        <label text="Cria a estrutura base DROP TABLE."></label>
        <block type="drop_table"></block>
        <label text="DELETE FROM - Excluí um ou varios dados de uma tabela."></label>
        <label text="Cria a estrutura base DELETE FROM."></label>
        <block type="delete_from_table"></block>
    </category>
</xml>
```

Fonte: Elaborado pelo autor

3.5 PÁGINA DE EXERCÍCIOS

A página de exercícios tem como objetivo fornecer ao usuário um meio simples e mais desafiador de entender alguns mecanismos da linguagem SQL. Nessa página o usuário tem dois tipos de exercícios diferentes com 5 etapas cada (Figura 18).

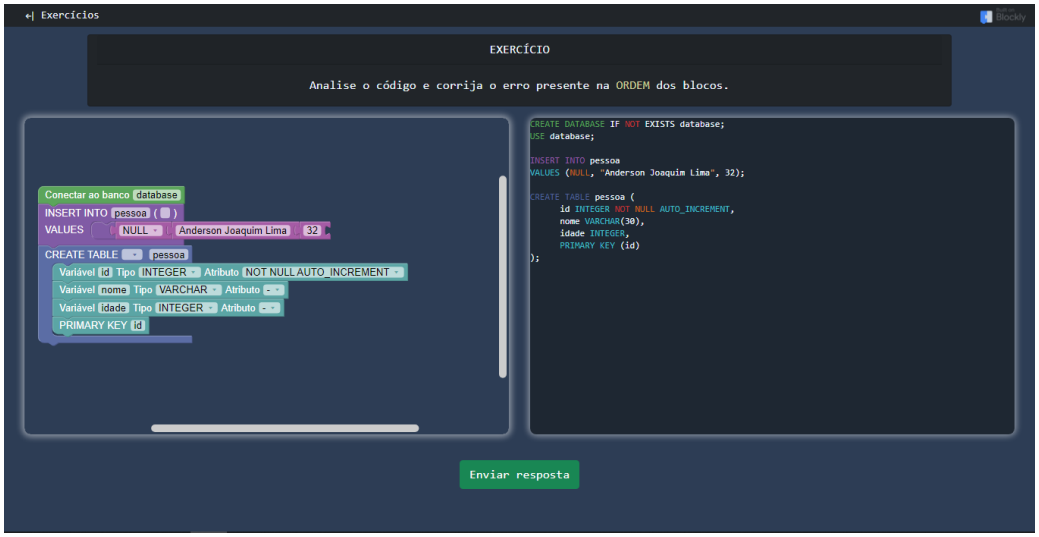
Figura 18 – Página de Exercícios.

Fonte: Elaborado pelo autor

3.5.1 Exercícios 1 – Organize os blocos

O primeiro exercício consiste em encontrar os erros presentes na ordem dos blocos ou acrescentar os blocos necessários para que a geração do código seja correta (Figura 19). A cada etapa desse exercício é apresentado ao usuário erros simples de semântica com o objetivo de que ele entenda e evite erros semelhantes no futuro.

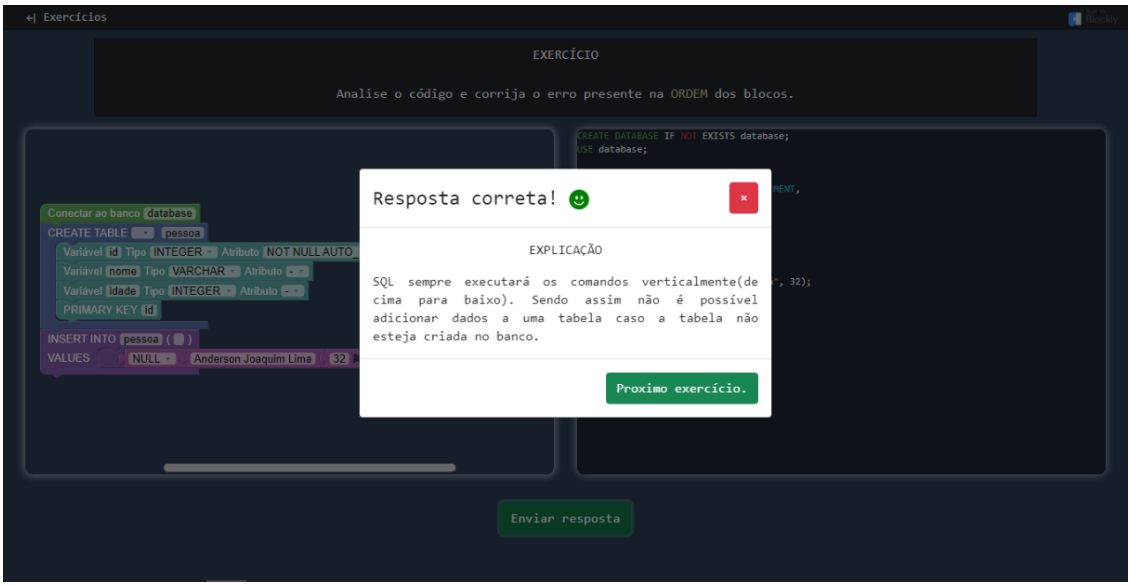
Figura 19 – Etapa 1 do primeiro exercício.



Fonte: Elaborado pelo autor

As páginas dos exercícios são padrões, havendo um campo com a *workspace* do Blockly, no qual o usuário realizará as modificações do código, um campo de texto onde apresenta o código gerado pela *workspace* e um botão que ao ser apertado irá analisar a resposta dada e retornará. Caso a resposta seja correta, uma mensagem de correto e uma breve explicação ou análise do motivo do erro (Figura 20). Para a verificação da resposta é realizada uma comparação do código gerado pelo usuário e o código considerado certo que é descrito em uma *string* no arquivo JavaScript do exercício (Figura 21).

Figura 20 – Mensagem de resposta correta.



Fonte: Elaborado pelo autor

Figura 21 – Verificação da resposta do exercício 1.

```
function verificarResposta() {
    let resposta = `CREATE DATABASE IF NOT EXISTS database;
USE database;

CREATE TABLE pessoa (
      id INTEGER NOT NULL AUTO_INCREMENT,
      nome VARCHAR(30),
      idade INTEGER,
      PRIMARY KEY (id)
);

INSERT INTO pessoa
VALUES (NULL, "Anderson Joaquim Lima", 32);`

    let codigo = document.getElementById('Codigo').textContent;

    if (codigo.includes(resposta)) {
        document.getElementById('respostaCorreta').style.display = 'block'
        document.getElementById('respostaErrada').style.display = 'none'

        let prog = JSON.parse(localStorage.getItem('progressoEstudos'));

        if (prog[0] < 2)
            localStorage.setItem('progressoEstudos', JSON.stringify([2,prog[1],prog[2],prog[3]]));

    } else {
        document.getElementById('respostaErrada').style.display = 'block'
        document.getElementById('respostaCorreta').style.display = 'none'
    }
}
```

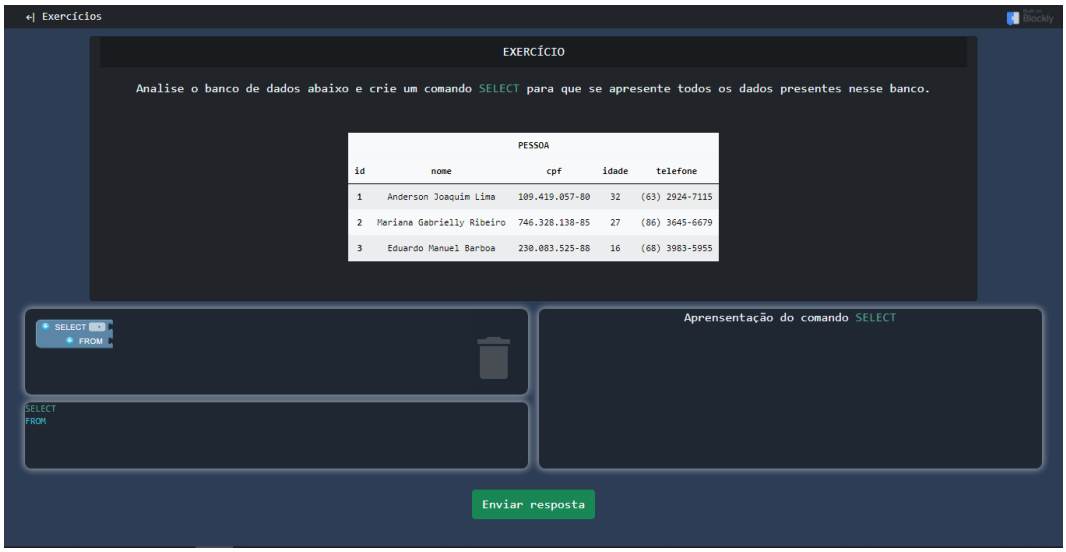
Fonte: Elaborado pelo autor

Em todos os exercícios, ao analisar que a resposta está correta, a aplicação armazenará no armazenamento local do navegador a próxima fase deverá realizar. Além de apresentar a mensagem correta sobre a resposta do usuário.

3.5.2 Exercícios 2 – Monte o Select

Para o segundo exercício, dado uma ou duas tabelas, o usuário deverá montar um select, a partir dos blocos visuais, que apresente todos dados pedidos pela etapa em questão. Uma tabela com todos os dados será gerada automaticamente a cada modificação realizada nos blocos (Figura 22).

Figura 22 – Etapa 1 do segundo exercício.



Fonte: Elaborado pelo autor

Assim como no exercício 1, as páginas dos exercícios 2 também são padrões entre si, diferindo somente na adição do campo de apresentação da tabela que o código está gerando. Para a verificação da resposta é inicialmente realizada uma conversão da tabela gerada pelo usuário para uma tabela JSON, utilizando a função tableToObj fornecida por um usuário do GitHub (Figura 23). A partir dessa conversão o JSON gerado é comparado com o JSON definido anteriormente, no qual possui os dados pedidos pelo exercício (Figura 24).

Figura 23 – Função tableToObj.

JavaScript: HTML table to object

gistfile1 Raw

```
//Short, jQuery-independent function to read html table and write them into an Array.
//Kudos to RobG at StackOverflow

function tableToObj(table) {
  var rows = table.rows;
  var propCells = rows[0].cells;
  var propNames = [];
  var results = [];
  var obj, row, cells;

  // Use the first row for the property names
  // Could use a header section but result is the same if
  // there is only one header row
  for (var i=0, iLen=propCells.length; i<iLen; i++) {
    propNames.push(propCells[i].textContent || propCells[i].innerText);
  }

  // Use the rows for data
  // Could use tbody rows here to exclude header & footer
  // but starting from 1 gives required result
  for (var j=1, jLen=rows.length; j<jLen; j++) {
    cells = rows[j].cells;
    obj = {};

    for (var k=0; k<iLen; k++) {
      obj[propNames[k]] = cells[k].textContent || cells[k].innerText;
    }
    results.push(obj)
  }
  return results;
}
```

Fonte: GitHub

Figura 24 – Verificação da resposta do exercício 2.

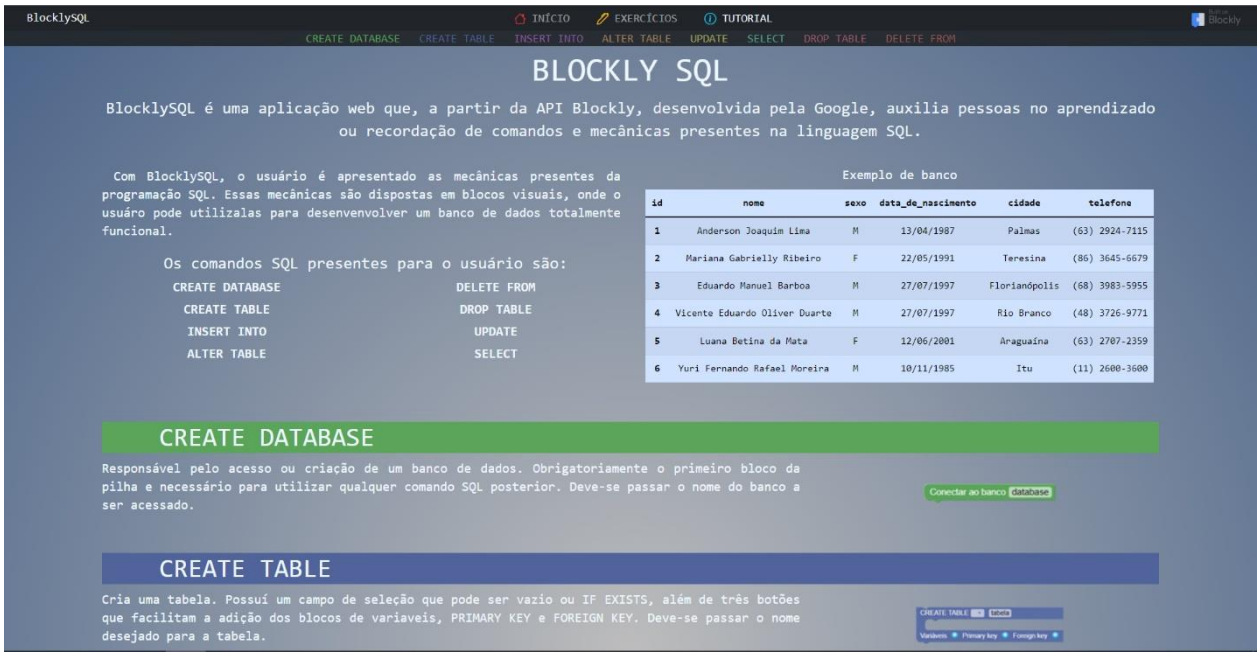
```
function verificarResposta() {
    var respostas = [
        {
            Id: "1",
            Nome: "Anderson Joaquim Lima",
            Cpf: "109.419.057-80",
            Idade: "32",
            Telefone: "(63) 2924-7115"
        },
        {
            Id: "2",
            Nome: "Mariana Gabrielly Ribeiro",
            Cpf: "746.328.138-85",
            Idade: "27",
            Telefone: "(86) 3645-6679"
        },
        {
            Id: "3",
            Nome: "Eduardo Manuel Barbosa",
            Cpf: "230.083.525-88",
            Idade: "16",
            Telefone: "(68) 3983-5955"
        }
    ]

    try {
        var tabela = tableToObj(document.getElementById('tabelaSelect2'));
    } catch(e) {}

    if (isContainedIn(respostas, tabela)) {
        document.getElementById('respostaCorreta').style.display = 'block'
        document.getElementById('respostaErrada').style.display = 'none'

        let prog = JSON.parse(localStorage.getItem('progressoEstudos'));
        if (prog[1] < 2)
            localStorage.setItem('progressoEstudos', JSON.stringify([prog[0], 2, prog[2], prog[3]]));

    } else {
        document.getElementById('respostaErrada').style.display = 'block'
        document.getElementById('respostaCorreta').style.display = 'none'
    }
}
```

Fonte: Elaborado pelo autor

3.6 PÁGINA TUTORIAL

Na página tutorial foi utilizado apenas HTML e CSS para a sua elaboração. Essa página é um guia de apresentação e documentação sobre as funcionalidades dos blocos. Nessa página o usuário pode entender mais sobre funcionalidades gerais dos blocos e do site (Figura 25).

Figura 25 – Página Tutorial.



Fonte: Elaborado pelo autor

3.7 AÇÕES GERAIS DA APLICAÇÃO

Após a configuração da *workspace* é passada a função addChangerListener, conforme a linha 19 da Figura 26. Essa é uma função ouvinte da *workspace* que sempre é executada quando uma ação na mesma é realizada. Essa função é passada uma vez para desabilitar os blocos sem uma conexão pai, ou seja, qualquer bloco não conectado ao bloco start_sql, e outra para ativar a função mirrorEvent. A função mirrorEvent faz com que todos os blocos presentes na *workspace* sejam automaticamente lidos e transformados em código. O código gerado também é colorido chamando a função aplicarColor (Figura 26).

Figura 26 – Função mirrorEvent.

```
workspace.addChangeListener(Blockly.Events.disableOrphans);
workspace.addChangeListener(this.mirrorEvent);

//Carrega a ultima workspace ou o bloco de conexção ao banco
try {
    if (localStorage.getItem('lastWorkspace').length > 61)
        Blockly.Xml.domToWorkspace(Blockly.Xml.textToDom(localStorage.getItem('lastWorkspace')), workspace);
    else
        Blockly.Xml.domToWorkspace(Blockly.Xml.textToDom('<xml><block type="start_sql"><field name="db_name">db</field></block></xml>'),
} catch (err) {
    console.log(err);
}

function mirrorEvent() {
    let linguagem = document.getElementById("linguagens").value;

    switch (linguagem) {
        case "WebSQL":
            var code = Blockly.JavaScript.workspaceToCode(this.workspace);
            break;
        case "MySQL":
            var code = Blockly.Lua.workspaceToCode(this.workspace);
            break;
    }

    document.getElementById("Codigo").innerHTML = aplicarColor(code);

    //Configuração SELECT JOIN
    try {
        var bloco_join = Blockly.mainWorkspace.getBlocksByType('select_join')[0];
        var nomeTabela = Blockly.mainWorkspace.getBlocksByType('select_from')[0].getFieldValue('table_name');
        bloco_join.setFieldValue(bloco_join.getFieldValue('table_name'), 'tabela1')
        bloco_join.setFieldValue(nomeTabela, 'tabela2')
    } catch (error) {}

    // Salva a workspace no armazenamento do browser
    localStorage.setItem('lastWorkspace', Blockly.Xml.domToText(Blockly.Xml.workspaceToDom(this.workspace)));
}
```

Fonte: Elaborado pelo autor

A função mirrorEvent também possuí o adicional de identificar para qual linguagem os blocos devem ser traduzidos, executando assim a função workspaceToCode para a linguagem desejada. A função também salva a última workspace no armazenamento local do navegador.

Em campos de apresentação de código a aplicação também aplica uma coloração que ajuda o usuário a melhor relacionar os blocos com o código gerado (Figura 27).

Figura 27 – Coloração do campo de código.

```
function aplicarColor(code){
    code = code.replace(/CREATE DATABASE/g, "<span class='color_create_db'>CREATE DATABASE</span>");
    code = code.replace(/USE/g, "<span class='color_create_db'>USE</span>");
    code = code.replace(/CREATE TABLE/g, "<span class='color_create_table'>CREATE TABLE</span>");
    code = code.replace(/INSERT INTO/g, "<span class='color_insert'>INSERT INTO</span>");
    code = code.replace(/DROP TABLE/g, "<span class='color_drop_table'>DROP TABLE</span>");
    code = code.replace(/UPDATE/g, "<span class='color_update'>UPDATE</span>");
    code = code.replace(/SELECT/g, "<span class='color_select'>SELECT</span>");
    code = code.replace(/ALTER TABLE/g, "<span class='color_alter_table'>ALTER TABLE</span>");

    code = code.replace(/INTEGER/g, "<span class='color_var'>INTEGER</span>");
    code = code.replace(/VARCHAR/g, "<span class='color_var'>VARCHAR</span>");
    code = code.replace(/CHAR/g, "<span class='color_var'>CHAR</span>");
    code = code.replace(/DATE,/g, "<span class='color_var'>DATE,</span>");
    code = code.replace(/CURRENT_TIMESTAMP/g, "<span class='color_var'>CURRENT_TIMESTAMP</span>");
    code = code.replace(/TIMESTAMP/g, "<span class='color_var'>TIMESTAMP</span>");
    code = code.replace(/PRIMARY KEY/g, "<span class='color_var'>PRIMARY KEY</span>");
    code = code.replace(/FOREIGN KEY/g, "<span class='color_var'>FOREIGN KEY</span>");
    code = code.replace(/REFERENCES/g, "<span class='color_var'>REFERENCES</span>");
    code = code.replace(/UNIQUE/g, "<span class='color_var'>UNIQUE</span>");
    code = code.replace(/AUTO_INCREMENT/g, "<span class='color_var'>AUTO_INCREMENT</span>");

    code = code.replace(/ASC/g, "<span class='color_var'>ASC</span>");
    code = code.replace(/DESC/g, "<span class='color_var'>DESC</span>");
    code = code.replace(/AND/g, "<span class='color_var'>AND</span>");
    code = code.replace(/ON/g, "<span class='color_var'>ON</span>");
    code = code.replace(/ADD/g, "<span class='color_var'>ADD</span>");
    code = code.replace(/DROP COLUMN/g, "<span class='color_var'>DROP COLUMN</span>");

    code = code.replace(/DISTINCT/g, "<span class='color_var'>DISTINCT</span>");
    code = code.replace(/SET/g, "<span class='color_var'>SET</span>");
    code = code.replace(/SET/g, "<span class='color_var'>SET</span>");
    code = code.replace(/VALUES/g, "<span class='color_var'>VALUES</span>");

    code = code.replace(/FROM/g, "<span class='color_select'>FROM</span>");
    code = code.replace(/WHERE/g, "<span class='color_select'>WHERE</span>");
    code = code.replace(/ORDER BY/g, "<span class='color_select'>ORDER BY</span>");
    code = code.replace(/JOIN/g, "<span class='color_select'>JOIN</span>");
    code = code.replace(/INNER/g, "<span class='color_select'>INNER</span>");
    code = code.replace(/LEFT/g, "<span class='color_select'>LEFT</span>");
    code = code.replace(/RIGHT/g, "<span class='color_select'>RIGHT</span>");
    code = code.replace(/FULL/g, "<span class='color_select'>FULL</span>");

    code = code.replace(/%/g, "<span style='color: rgb(197, 147, 97)'>%</span>");
    code = code.replace(/"/g, "<span style='color: rgb(197, 147, 97)'>" + '"' + "</span>");

    code = code.replace(/ > /g, "<span style='color: rgb(203, 107, 26)'> > </span>");
    code = code.replace(/ >= /g, "<span style='color: rgb(203, 107, 26)'> >= </span>");
    code = code.replace(/ < /g, "<span style='color: rgb(203, 107, 26)'> < </span>");
    code = code.replace(/ <= /g, "<span style='color: rgb(203, 107, 26)'> <= </span>");
    code = code.replace(/ = /g, "<span style='color: rgb(203, 107, 26)'> = </span>");
    code = code.replace(/LIKE/g, "<span style='color: rgb(203, 107, 26)'>LIKE</span>");
    code = code.replace(/BETWEEN/g, "<span style='color: rgb(203, 107, 26)'>BETWEEN</span>");
    code = code.replace(/ IN /g, "<span style='color: rgb(203, 107, 26)'> IN </span>");

    code = code.replace(/NOT/g, "<span style='color: rgb(192, 53, 53)'>NOT</span>");
    code = code.replace(/NULL|null|"null"/g, "<span style='color: rgb(203, 107, 26)'>NULL</span>");

    return code;
}
```

Fonte: Elaborado pelo autor

4 CONSIDERAÇÕES FINAIS

A linguagem SQL pode não ser de fácil compreensão para todos os seus usuários, visto que possui maneiras próprias de executar seus comandos e suas funcionalidades. Sabendo disso, foi desenvolvido, a partir da API Google, Blockly, uma aplicação web que fosse capaz de auxiliar as pessoas nos estudos da linguagem ou que fosse capaz de recordar usuários que já estudaram sua sintaxe e semântica.

Desenvolvi para o site três páginas principais, a página inicial é uma área *sandbox* para o livre uso dos blocos pelo usuário, essa página possui uma área com os blocos que representam as mecânicas da linguagem SQL. Uma área de texto no qual é disposto o código gerado pelos blocos e uma área de apresentação do comando SELECT. A segunda página do site é a de exercícios, nela o usuário é submetido a alguns desafios que o força a analisar e corrigir os erros presentes em um trecho de código ou construir a consulta adequada ao que se pede. A página final é uma apresentação dos blocos, que ajuda o usuário a entender melhor a funcionalidade de cada bloco, juntamente com o seu modo de uso.

Para o desenvolvimento da aplicação foi necessário realizar diversas pesquisas em sites de perguntas e respostas voltados a programação para ter uma melhor compreensão do Blockly. A documentação oficial disponibilizada nem sempre era de fácil entendimento ou não demonstrava como acessar certas funcionalidades dos blocos como, por exemplo, a criação de uma nova linguagem ou a edição de uma *label* pré-determinada.

Tive também pequenas dificuldades na criação do site, desde a escrita do HTML até a estilização do CSS, sendo posteriormente utilizado o Bootstrap, facilitando personalizações gerais.

Apesar da finalização da aplicação, nem todos os comandos presentes na linguagem SQL foram implementados. No bloco referente ao comando *SELECT*, por exemplo, é possível adicionar os comandos *GROUP BY* e *HAVING*, fornecendo ainda mais funcionalidades SQL aos usuários. A aplicação também não foi submetida a teste práticos de usabilidade ou na efetividade para estudos, seja com iniciadores da linguagem ou usuários de longa data. Recomendando-se assim a implementação de novas funcionalidades e testes práticos para trabalhos futuros.

REFERÊNCIAS

ANDREI, L. **O Que é HTML? Guia Básico Para Iniciantes**. Hostinger. 11 mai. 2021. Disponível em: https://www.hostinger.com.br/tutoriais/o-que-e-html-conceitos-basicos. Acesso em: 18 mai. 2021.

BLOCKLY API. ***Blockly: A JavaScript library for building visual programming editors***. Google. 06 nov. 2020. Disponível em: https://developers.google.com/blockly. Acesso em: 27 abr. 2021.

BLOCKLY API. ***Blockly Developer Tools***. Google. 09 set. 2020. Disponível em: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html. Acesso em: 27 abr. 2021.

BLOCKLY API. ***Introduction to Blockly***. Google. 06 nov. 2020. Disponível em: https://developers.google.com/blockly/guides/overview. Acesso em: 27 abr. 2021.

BLOCKLY API. ***Get Started***. Google. 04 mai. 2021. Disponível em: https://developers.google.com/blockly/guides/get-started/web. Acesso em: 29 nov. 2021.

CRONAPP. **Programação Blockly: saiba mais sobre essa linguagem visual**. 22 set. 2021. Disponível em: https://blog.cronapp.io/programacao-blockly-saiba-mais-sobre-essa-linguagem-visual/. Acesso em: 14 dez. 2021

GIST, Mattheo. **JavaScript: HTML *table to Object***. GitHub. 2012. Disponível em: https://gist.github.com/mattheo-gist/4151867. Acesso em: 13 set. 2021.

GITHUB, Blockly. ***Getting Started with Blockly***. Disponível em: https://github.com/google/blockly. Acesso em: 27 abr. 2021.

HICKSON, Ian. ***Web SQL Database***. W3. 18 nov. 2010. Disponível em: https://www.w3.org/TR/*web*database/. Acesso em: 17 mai. 2021.

JAVATPOINT. **DBMS Tutorial**. Javatpoint. 2018. Disponível em: https://www.javatpoint.com/dbms-tutorial. Acesso em: 16 mai. 2021.

KAMIYA, Reginaldo Rideaki; BRANDÃO, Leônidas O. **iVProg-um sistema para introdução à Programação através de um modelo Visual na Internet**. Anais do XX Simpósio Brasileiro de Informática na Educação. Florianópolis, SC, 2009.

KODEKLIX. **KodeKLIX: *Coding, Electronics and Fun!***, c2021. Disponível em: http://www.kodeklix.com.au/. Acesso em: 29 nov. 2021.

MDN. **CSS**. MDN *Web* Docs. 16 mai. 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/*Web*/CSS. Acesso em: 18 mai. 2021.

MDN. **Estrutura de documento e sites**. MDN *Web* Docs. 24 mai. 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/HTML/Introduction_to_HTML/Document_and_*web*site_structure. Acesso em: 25 mai. 2021.

MDN. **O que é JavaScript?**. MDN *Web* Docs. 16 mai. 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_is_JavaScript. Acesso em: 18 mai. 2021.

NOLETO, Cairo. **Aplicações *web*: entenda o que são e como funcionam!**. Betrybe. 18 mar. 2020. Disponível em: https://blog.betrybe.com/desenvolvimento-*web*/aplicacoes-*web*/. Acesso em 17 mai. 2021.

ORACLE. ***What Is a Database?***. Oracle. Disponível em: https://www.oracle.com/database/what-is-database/. Acesso em: 15 mai. 2021.

OZO EDU. **OzoBlockly**, c2021. Disponível em: https://ozobot.com/create/ozoblockly. Acesso em: 23 nov. 2021.

PRATES, Lucas de Matos. **A ferramenta blockly como apoio ao processo de ensino de algoritmos**. 2020. Disponível em: https://cepein.femanet.com.br/BDigital/arqTccs/1711420031.pdf. Acesso em: 14 dez. 2021.

SOUZA, Ivan De. **Banco de dados: saiba o que é, os tipos e a importância para o site da sua empresa**. Rockcontent. 2020. Disponível em: https://rockcontent.com/br/blog/banco-de-dados/. Acesso em: 16 mai. 2021.

TUTORIALSPOINT. ***HTML5 – Web SQL Database***. Tutorialspoint. 2021. Disponível em: https://www.tutorialspoint.com/html5/html5_*web*_sql.htm. Acesso em: 16 mai. 2021

TRINDADE, Andrea Garcia. **Linguagem de programação visual: uma nova forma de apresentar a programação de computadores**. Revista Processando o Saber, Praia Grande, v.7, n.7, p.65-79, 01 out. 2015.

UNPKG. **Unpkg is a fast, global content delivery network for everything on npm**. Disponível em: https://unpkg.com/. Acesso em: 27 abr. 2021.

WAZLAWICK, R. S. **Metodologia da Pesquisa para Ciência da Computação**. 2ª. ed. [S.l.]: Campus, 2014.

WONDER WORKSHOP. **Wonder Workshop**, c2019. Página inicial. Disponível em: https://www.makewonder.com/. Acesso em: 29 nov. 2021.