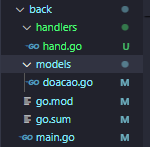
**ETAPA 7**

Vamos começar pelo Back, onde iremos fazer as funções do CRUD e fazer a interligação com o front:



Começando pela pasta handlers, foi criado um arquivo hand.go, onde inicialmente não usaremos por agora, mas deixaremos algumas informações por lá por enquanto:

hand.go

package *handlers*

import (

    "database/sql"

    "net/http"

    "root/models"

    "github.com/gin-gonic/gin"

)

type *Handler* struct {

    DB \**sql*.*DB*

}

func NewHandler(*db* \**sql*.*DB*) \**Handler* {

    return &*Handler*{DB: db}

}

func (*h* \**Handler*) InserirDado(*c* \**gin*.*Context*) {

    var novoDado *models*.*Doador*

    if err := c.BindJSON(&novoDado); err != nil {

        c.JSON(http.StatusBadRequest, *gin*.*H*{"error": err.Error()})

        return

    }

    // Inserir dados no banco de dados

    result, err := h.DB.Exec("INSERT INTO doador (nome,cpf,contato,tipo\_sanguineo,rh,tipo\_rh\_corretos,situacao) VALUES ($1,$2,$3,$4,$5,$6,$7)", novoDado.Nome, novoDado.CPF, novoDado.Contato, novoDado.TipoDoSangue, novoDado.Rh, novoDado.TipoRHCorretos, novoDado.Situacao)

    if err != nil {

        c.JSON(http.StatusInternalServerError, *gin*.*H*{"error": err.Error()})

        return

    }

    // Obter o ID do dado inserido

    id, \_ := result.LastInsertId()

    // Retornar o ID do dado inserido

    c.JSON(http.StatusCreated, *gin*.*H*{"id": id})

}

func (*h* \**Handler*) BuscarDoacoes(*c* \**gin*.*Context*) {

    // Realizar a consulta ao banco de dados para buscar as doações

    rows, err := h.DB.Query("SELECT codigo, nome, cpf, contato, tipo\_sanguineo, rh, tipo\_rh\_corretos, situacao FROM doador")

    if err != nil {

        c.JSON(http.StatusInternalServerError, *gin*.*H*{"error": err.Error()})

        return

    }

    defer rows.Close()

    // Converter os resultados da consulta em uma lista de doadores

    var doadores []*models*.*Doador*

    for rows.Next() {

        var doador *models*.*Doador*

        err := rows.Scan(&doador.Codigo, &doador.Nome, &doador.CPF, &doador.Contato, &doador.TipoDoSangue, &doador.Rh, &doador.TipoRHCorretos, &doador.Situacao)

        if err != nil {

            c.JSON(http.StatusInternalServerError, *gin*.*H*{"error": err.Error()})

            return

        }

        doadores = append(doadores, doador)

    }

    // Fechar a consulta

    if err := rows.Err(); err != nil {

        c.JSON(http.StatusInternalServerError, *gin*.*H*{"error": err.Error()})

        return

    }

    // Retornar os doadores como uma resposta JSON

    c.JSON(http.StatusOK, *gin*.*H*{

        "mensagem": "Lista de doadores",

        "doadores": doadores,

    })

}

Na pasta models, foi criado o arquivo doação.go que foi mostrado no tutorial passado. Onde basicamente ele seria sua classe.

Doação.go

package *models*

import (

    "time"

)

type *RH* *int64*

type *TipoSanguineo* *int64*

const (

    A  *TipoSanguineo* = 0

    B  *TipoSanguineo* = 1

    AB *TipoSanguineo* = 2

    O  *TipoSanguineo* = 3

)

const (

    POSITIVO *RH* = 1

    NEGATIVO *RH* = 0

)

type *Doador* struct {

    Codigo         *int*    `json:"codigo"`

    Nome           *string* `json:"nome"`

    CPF            *string* `json:"cpf"`

    Contato        *string* `json:"contato"`

    TipoDoSangue   *string* `json:"tipodosangue"`

    Rh             *string* `json:"rh"`

    TipoRHCorretos *bool*   `json:"tiporhcorretos"`

    Situacao       *string* `json:"situacao"`

}

type *Doacao* struct {

    Codigo *int*

    Data   *time*.*Time*

    Hora   *time*.*Duration*

    Volume *float64*

}

E por fim, na parte do back, temos o main.go, onde é ele que colocaremos nosso CRUD, e a conexão entre o banco também.

Durante o processo o código pedirá algumas importações para fazer no terminal e outras que o próprio VsCode faz automático, segue abaixo algumas:

go get github.com/rs/cors

go get github.com/gorilla/mux

go get github.com/lib/pq

Caso queira a versão mais atualizada, tente desta forma:

go get -u github.com/rs/cors

go get -u github.com/gorilla/mux

go get -u github.com/lib/pq

main.go

package *main*

import (

    "database/sql"

    "encoding/json"

    "fmt"

    "log"

    "net/http"

    "strconv"

    "github.com/gorilla/mux"

    \_ "github.com/lib/pq"

    "github.com/rs/cors"

)

const (

    host     = "172.17.0.3"

    port     = 5432

    user     = "postgres"  // Usuário PostgreSQL

    password = "Digite\_sua\_senha\_:D" // Senha PostgreSQL

    dbname   = "web"       // Banco de dados PostgreSQL

)

type *Doador* struct {

    Codigo         *int*    `json:"codigo"`

    Nome           *string* `json:"nome"`

    CPF            *string* `json:"cpf"`

    Contato        *string* `json:"contato"`

    TipoDoSangue   *string* `json:"tipoSanguineo"`

    Rh             *string* `json:"fatorRh"`

    TipoRHCorretos \**bool*  `json:"tiporhcorretos"`

    Situacao       *string* `json:"situacao"`

}

var db \**sql*.*DB*

func main() {

    // Estabelece conexÃ£o com o banco de dados

    dbinfo := fmt.Sprintf("host=%s port=%d user=%s "+

        "password=%s dbname=%s sslmode=disable",

        host, port, user, password, dbname)

    var err *error*

    db, err = sql.Open("postgres", dbinfo)

    if err != nil {

        log.Fatal(err)

    }

    defer db.Close()

    // Inicia o roteador

    router := mux.NewRouter()

    // Define as rotas CRUD

    router.HandleFunc("/informacoes", getTodos).Methods("POST")

    router.HandleFunc("/dados/{codigo}", getTodo).Methods("GET")

    router.HandleFunc("/dados", createTodo).Methods("POST")

    router.HandleFunc("/dados/{codigo}", updateTodo).Methods("PUT")

    router.HandleFunc("/dados/{codigo}", deleteTodo).Methods("DELETE")

    // Inicia o servidor na porta 8080

    //log.Fatal(http.ListenAndServe(":8081", router))

    // Adiciona o middleware CORS ao roteador

    c := cors.New(*cors*.*Options*{

        AllowedOrigins: []*string*{"http://localhost:8080"},

        AllowedMethods: []*string*{"GET", "POST", "PUT", "DELETE", "OPTIONS"},

        AllowedHeaders: []*string*{"\*"},

    })

    // Adiciona o middleware CORS ao roteador

    handler := c.Handler(router)

    // Inicia o servidor na porta 8081

    log.Fatal(http.ListenAndServe(":8081", handler))

}

func getTodos(*w* *http*.*ResponseWriter*, *r* \**http*.*Request*) {

    var dadosNovos *Doador*

    \_ = json.NewDecoder(r.Body).Decode(&dadosNovos)

    // Construção da consulta SQL

    query := "SELECT \* FROM doador WHERE situacao =$1"

    parametros := []interface{}{"Ativo"} // Parâmetros para a consulta SQL

    // Verifica se as variáveis de pesquisa não estão vazias

    if dadosNovos.Nome != "" {

        query += " AND nome ILIKE $" + strconv.Itoa(len(parametros)+1)

        parametros = append(parametros, dadosNovos.Nome+"%")

    }

    if dadosNovos.CPF != "" {

        query += " AND cpf ILIKE $" + strconv.Itoa(len(parametros)+1)

        parametros = append(parametros, dadosNovos.CPF+"%")

    }

    if dadosNovos.Contato != "" {

        query += " AND contato ILIKE $" + strconv.Itoa(len(parametros)+1)

        parametros = append(parametros, dadosNovos.Contato+"%")

    }

    if dadosNovos.TipoDoSangue != "" {

        query += " AND tipo\_sanguineo ILIKE $" + strconv.Itoa(len(parametros)+1)

        parametros = append(parametros, dadosNovos.TipoDoSangue+"%")

    }

    if dadosNovos.Rh != "" {

        query += " AND rh=$" + strconv.Itoa(len(parametros)+1)

        parametros = append(parametros, dadosNovos.Rh)

    }

    if dadosNovos.TipoRHCorretos != nil {

        query += " AND tipo\_rh\_corretos= $" + strconv.Itoa(len(parametros)+1)

        parametros = append(parametros, dadosNovos.TipoRHCorretos)

    }

    if dadosNovos.Codigo != -1 {

        query += " AND codigo=$" + strconv.Itoa(len(parametros)+1)

        parametros = append(parametros, dadosNovos.Codigo)

    }

    // Executa a consulta SQL

    rows, err := db.Query(query, parametros...)

    if err != nil {

        log.Fatal(err)

    }

    defer rows.Close()

    // Itera sobre os resultados

    var todos []*Doador*

    for rows.Next() {

        var novoDado *Doador*

        err := rows.Scan(&novoDado.Codigo, &novoDado.Nome, &novoDado.CPF, &novoDado.Contato, &novoDado.TipoDoSangue, &novoDado.Rh, &novoDado.TipoRHCorretos, &novoDado.Situacao)

        if err != nil {

            log.Fatal(err)

        }

        todos = append(todos, novoDado)

    }

    // Codifica os resultados como JSON e envia a resposta

    json.NewEncoder(w).Encode(todos)

}

func getTodo(*w* *http*.*ResponseWriter*, *r* \**http*.*Request*) {

    params := mux.Vars(r)

    codigo := params["codigo"]

    var todo *Doador*

    err := db.QueryRow("SELECT \* FROM doador WHERE codigo=$1%", codigo).Scan(&todo.Codigo, &todo.Nome, &todo.CPF, &todo.Contato, &todo.TipoDoSangue, &todo.Rh, &todo.TipoRHCorretos, &todo.Situacao)

    if err != nil {

        log.Fatal(err)

    }

    json.NewEncoder(w).Encode(todo)

}

func createTodo(*w* *http*.*ResponseWriter*, *r* \**http*.*Request*) {

    var novoDado *Doador*

    \_ = json.NewDecoder(r.Body).Decode(&novoDado)

    \_, err := db.Exec("INSERT INTO doador (nome,cpf,contato,tipo\_sanguineo,rh,tipo\_rh\_corretos,situacao) VALUES ($1,$2,$3,$4,$5,$6,$7)", novoDado.Nome, novoDado.CPF, novoDado.Contato, novoDado.TipoDoSangue, novoDado.Rh, novoDado.TipoRHCorretos, novoDado.Situacao)

    if err != nil {

        log.Fatal(err)

    }

    json.NewEncoder(w).Encode(novoDado)

}

func updateTodo(*w* *http*.*ResponseWriter*, *r* \**http*.*Request*) {

    params := mux.Vars(r)

    codigo := params["codigo"]

    var novoDado *Doador*

    \_ = json.NewDecoder(r.Body).Decode(&novoDado)

    \_, err := db.Exec("UPDATE  doador SET nome=$1, cpf=$2, contato=$3, tipo\_sanguineo=$4, rh=$5, tipo\_rh\_corretos=$6 WHERE codigo=$7", novoDado.Nome, novoDado.CPF, novoDado.Contato, novoDado.TipoDoSangue, novoDado.Rh, novoDado.TipoRHCorretos, codigo)

    if err != nil {

        log.Fatal(err)

    }

    json.NewEncoder(w).Encode(novoDado)

}

func deleteTodo(*w* *http*.*ResponseWriter*, *r* \**http*.*Request*) {

    params := mux.Vars(r)

    codigo := params["codigo"]

    \_, err := db.Exec("UPDATE doador SET situacao =$1 WHERE codigo=$2", "Inativo", codigo)

    if err != nil {

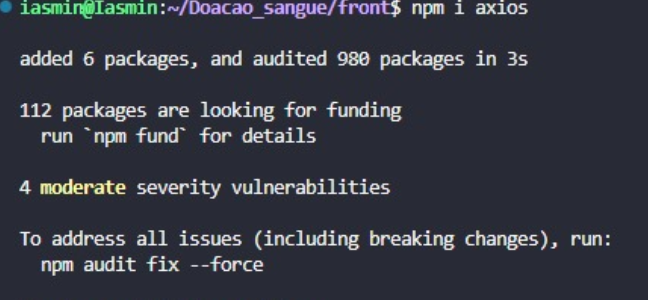
        log.Fatal(err)

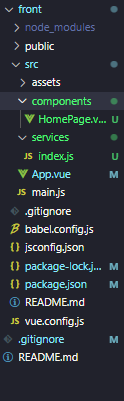
    }

    w.WriteHeader(http.StatusNoContent)

}

Partindo agora pra o FRONT, primeiramente iremos fazer uma instalação no terminal, da seguinte forma:





Verifique se foi instalado seu axios, no package.json:

  "name": "front",

  "version": "0.1.0",

  "private": true,

  "scripts": {

    "serve": "vue-cli-service serve",

    "build": "vue-cli-service build",

    "lint": "vue-cli-service lint"

  },

  "dependencies": {

    "axios": "^1.6.8",

    "core-js": "^3.8.3",

    "vue": "^3.2.13"

  },

  "devDependencies": {

    "@babel/core": "^7.12.16",

    "@babel/eslint-parser": "^7.12.16",

    "@vue/cli-plugin-babel": "~5.0.0",

    "@vue/cli-plugin-eslint": "~5.0.0",

    "@vue/cli-service": "~5.0.0",

    "eslint": "^7.32.0",

    "eslint-plugin-vue": "^8.0.3"

  },

  "eslintConfig": {

    "root": true,

    "env": {

      "node": true

    },

    "extends": [

      "plugin:vue/vue3-essential",

      "eslint:recommended"

    ],

    "parserOptions": {

      "parser": "@babel/eslint-parser"

    },

    "rules": {}

  },

  "browserslist": [

    "> 1%",

    "last 2 versions",

    "not dead",

    "not ie 11"

  ]

}

Foi criado uma pasta chama services, onde foi colocado a configuração do axios, onde o arquivo chama index.js:

import axios from "axios";

const api = axios.create({

  baseURL: 'http://localhost:8081',

});

export default api;

E por fim, foi criado uma pasta chamada: componentes, e foi movido o arquivo que já foi criado pela própria versão do Vue, para pasta componentes. Nela, terá o vue,css e a conexão com o CRUD e banco.

HomePage.vue

<template>

  <div>

    <!-- Formulário de cadastro de doador -->

    <div *role*="form">

      <h2>Cadastro de Doador</h2>

      <input *type*="text" *v-model*="nome" *placeholder*="Nome"><br>

      <input *type*="text" *v-model*="cpf" *placeholder*="CPF"><br>

      <input *type*="text" *v-model*="contato" *placeholder*="Contato"><br>

      <select *v-model*="tipoSanguineo">

        <option *disabled* *value*="">Selecione o tipo sanguíneo</option>

        <option *value*="A">A</option>

        <option *value*="B">B</option>

        <option *value*="AB">AB</option>

        <option *value*="O">O</option>

      </select><br>

      <select *v-model*="fatorRh">

        <option *disabled* *value*="">Fator RH</option>

        <option *value*="POSITIVO">Positivo</option>

        <option *value*="NEGATIVO">Negativo</option>

      </select><br>

      <button @*click*="adicionarDoador">Adicionar Doador</button>

    </div>

    <!-- Tabela de doadores -->

    <div>

      <h2>Lista de Doadores</h2>

      <button @*click*="acaoSelecionada(doador, 'buscar')">Buscar</button>

      <br>

      <table *border*="1">

        <thead>

          <tr>

            <th>Código</th>

            <th>Nome</th>

            <th>CPF</th>

            <th>Contato</th>

            <th>Tipo Sanguíneo</th>

            <th>RH</th>

            <th>Ações</th>

          </tr>

        </thead>

        <tbody>

          <tr v-for="doador in doadores" :*key*="doador.codigo">

            <td>{{ doador.codigo  }}</td>

            <td>{{ doador.nome }}</td>

            <td>{{ doador.cpf }}</td>

            <td>{{ doador.contato }}</td>

            <td>{{ doador.tipoSanguineo }}</td>

            <td>{{ doador.fatorRh }}</td>

            <td>

              <button @*click*="acaoSelecionada(doador, 'editar')">Editar</button>

              <button @*click*="acaoSelecionada(doador, 'inativar')">Remover</button>

            </td>

          </tr>

        </tbody>

      </table>

    </div>

    <!-- Modal de Edição -->

    <div v-if="mostrarDropdownEditar" *class*="modal">

      <div *class*="modal-content">

      <span *class*="close" @*click*="fecharDropdown">&times;</span>

        <h3>Editar Doador</h3>

        <input *type*="text" *v-model*="doadorEditado.nome"  *placeholder*="Nome"><br>

        <input *type*="text" *v-model*="doadorEditado.cpf"  *placeholder*="CPF"><br>

        <input *type*="text" *v-model*="doadorEditado.contato"  *placeholder*="Contato"><br>

        <select *v-model*="doadorEditado.tipoSanguineo"  *placeholder*="Tipo Sanguineo">

          <option *value*="A">A</option>

          <option *value*="B">B</option>

          <option *value*="AB">AB</option>

          <option *value*="O">O</option>

        </select><br>

        <select *v-model*="doadorEditado.fatorRh"  *placeholder*="Fator RH">

          <option *value*="POSITIVO">Positivo</option>

          <option *value*="NEGATIVO">Negativo</option>

        </select><br>

        <button @*click*="editarDoador">Salvar</button>

      </div>

    </div>

    <!-- Popup de Inativação -->

    <div v-if="mostrarPopupInativar" *class*="popup-inativar">

      <div *class*="popup-content">

        <span *class*="close" @*click*="fecharDropdown">&times;</span>

        <h3>Confirmar Inativação</h3>

        <p>Deseja realmente inativar o doador {{ doadorInativar.nome }}?</p>

        <button @*click*="inativarDoador">Sim</button>

        <button @*click*="fecharPopupInativar">Não</button>

      </div>

    </div>

  </div>

  <div v-if="mostrarDropdownBuscar" *class*="modal">

      <div *class*="modal-content">

      <span *class*="close" @*click*="fecharDropdown">&times;</span>

        <h3>Buscar Doador</h3>

        <input *type*="text" *v-model*="doadorBuscavel.nome" *placeholder*="Nome"><br>

        <input *type*="text" *v-model*="doadorBuscavel.cpf"  *placeholder*="CPF"><br>

        <input *type*="text" *v-model*="doadorBuscavel.contato" *placeholder*="Contato"><br>

        <select *v-model*="doadorBuscavel.tipoSanguineo">

          <option *disabled* *value*="">Tipo Sanguineo</option>

          <option *value*="A">A</option>

          <option *value*="B">B</option>

          <option *value*="AB">AB</option>

          <option *value*="O">O</option>

          <option *value*="">TODOS</option>

        </select><br>

        <select *v-model*="doadorBuscavel.fatorRh">

          <option *disabled* *value*="">Fator RH</option>

          <option *value*="POSITIVO">Positivo</option>

          <option *value*="NEGATIVO">Negativo</option>

          <option *value*="">TODOS</option>

        </select><br>

        <button @*click*="buscarTodosDoadores">Salvar</button>

      </div>

    </div>

</template>

<script>

import api from '../services';

export default {

  data() {

    return {

      nome: '',

      cpf: '',

      contato: '',

      tipoSanguineo: '',

      fatorRh: '',

      filtro: '',

      doadores: [],

      mostrarDropdownEditar: false,

      doadorEditado: {

        codigo:'',

        nome: '',

        cpf: '',

        contato: '',

        tipoSanguineo: '',

        fatorRh: ''

      },

      mostrarPopupInativar: false,

      doadorInativar: {},

      mostrarDropdownBuscar: false,

      doadorBuscavel: {

        codigo: -1,

        nome: '',

        cpf: '',

        contato: '',

        tipoSanguineo: '',

        fatorRh: ''

      },

    };

  },

  mounted(){

*this*.buscarTodosDoadores()

  },

  methods: {

      async adicionarDoador() {

      // Simulação de adição de doador

      // this.doadores.push({

      //   nome: this.nome,

      //   cpf: this.cpf,

      //   contato: this.contato,

      //   tipoSanguineo: this.tipoSanguineo,

      //   fatorRh: this.fatorRh

      // });

      //api.get("/ping")

    //  const response = await api.post(api.defaults.baseURL + 'dados', this.doadores);

     console.log(*this*.tipoSanguineo);

      await api.post('dados', {

      nome: *this*.nome,

      cpf: *this*.cpf,

      contato: *this*.contato,

      tipoSanguineo: *this*.tipoSanguineo,

      fatorRh: *this*.fatorRh,

      situacao: "Ativo"

}).then(*response* => {

*this*.buscarTodosDoadores()

  console.log(response);

});

  // Limpe os campos do formulário após a adição

*this*.nome = '';

*this*.cpf = '';

*this*.contato = '';

*this*.tipoSanguineo = '';

*this*.fatorRh = '';

    },

    acaoSelecionada(*doador*, *acao*) {

      if (acao === 'editar') {

*this*.mostrarDropdownEditar = true;

*this*.doadorEditado = Object.assign({}, doador); // Clonar o objeto para evitar mutações indesejadas

      } else if (acao === 'inativar') {

*this*.mostrarPopupInativar = true;

*this*.doadorInativar = doador;

      } else if (acao == 'buscar') {

*this*.mostrarDropdownBuscar = true;

*this*.LimpaDepois()

      }

    },

    fecharPopupInativar() {

*this*.mostrarPopupInativar = false;

    },

    fecharDropdown() {

*this*.mostrarDropdownEditar = false;

    if(*this*.mostrarDropdownBuscar){

*this*.mostrarDropdownBuscar = false;

*this*.LimpaDepois()

    }

*this*.mostrarPopupInativar = false;

  },

  LimpaDepois() {

*this*.doadorBuscavel.codigo = - 1

*this*.doadorBuscavel.nome = ''

*this*.doadorBuscavel.cpf = ''

*this*.doadorBuscavel.contato = ''

*this*.doadorBuscavel.tipoSanguineo = ''

*this*.doadorBuscavel.fatorRh = ''

  },

  async editarDoador() {

      await api.put('dados/'+*this*.doadorEditado.codigo, {

      nome: *this*.doadorEditado.nome,

      cpf: *this*.doadorEditado.cpf,

      contato: *this*.doadorEditado.contato,

      tipoSanguineo: *this*.doadorEditado.tipoSanguineo,

      fatorRh: *this*.doadorEditado.fatorRh

})

      location.reload();

    },

   async inativarDoador() {

      await api.delete('/dados/'+*this*.doadorInativar.codigo)

      location.reload()

    },

    async buscarTodosDoadores() {

      await api.post('informacoes',*this*.doadorBuscavel).then(*response* => {

*this*.doadores = []

        response.data?.map(*item* => *this*.doadores.push(item))

      })

*this*.mostrarDropdownBuscar = false;

    },

  }

};

</script>

<style>

body {

  font-family: Arial, sans-serif;

  margin: 0;

  padding: 0;

  background-color: #f0f0f0;

}

div[*role*="form"] {

  max-width: 600px;

  margin: 20px auto;

  padding: 20px;

  background-color: #fff;

  box-shadow: 0 0 10px rgba(0,0,0,0.1);

}

table {

  width: 100%;

  border-collapse: collapse;

}

table, th, td {

  border: 1px solid #ddd;

}

th, td {

  padding: 8px;

  text-align: left;

}

*.modal*,*.popup-inativar* {

  position: fixed;

  z-index: 1;

  left: 0;

  top: 0;

  width: 100%;

  height: 100%;

  overflow: auto;

  background-color: rgba(0,0,0,0.4);

}

*.modal-content*,*.popup-content* {

  position: relative;

  background-color: #fefefe;

  margin: 15% auto;

  padding: 20px;

  border: 1px solid #888;

  width: 80%;

  box-shadow: 0 4px 8px rgba(0,0,0,0.2);

}

button {

  background-color: #4CAF50;

  color: white;

  padding: 10px 20px;

  border: none;

  cursor: pointer;

  margin: 5px;

  width: 100%;

}

button*:hover* {

  background-color: #45a049;

}

input, select {

  width: 100%;

  padding: 12px 20px;

  margin: 8px 0;

  display: inline-block;

  border: 1px solid #ccc;

  box-sizing: border-box;

}

h2 {

  text-align: center;

  margin-bottom: 20px;

}

*.modal* {

  position: fixed;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  background-color: rgba(0, 0, 0, 0.5); /\* Fundo semi-transparente para destacar o modal \*/

  display: flex;

  align-items: center;

  justify-content: center;

}

*.modal-content* {

  position: relative;

  background-color: white;

  padding: 20px;

}

*.close* {

  position: absolute;

  top: 10px;

  right: 10px;

  font-size: 24px;

  color: #000000;

  cursor: pointer;

}

*.close:hover*,

*.close:focus* {

  color: black;

  text-decoration: none;

  cursor: pointer;

}

</style>

Lembrando que para fazer os testes, deve entrar nas pastas respectivas, uma para o front e outro para o back, inserindo o seguinte comando:  
**npm run serve – para o Front**

**go run main.go – para o Back**

Ressalto que, além de iniciar o front e o back, ativei a URL no pontainer, para que meu banco funcionasse.