Caritá

Projeto Integrador

Idealizadores

Ariele Peres, 3011392310000

Fernando Ribeiro, 3011392310000

Guilherme Pazetti, 3011392413008

Guilherme Piva, 3011392413043

Mayara Barros, 3011392413017

Vanessa Capuano, 3011392413031

Orientador: CLAUDIO ROBERTO CORREDATO

Introdução

O projeto Caritá surgiu como uma iniciativa pensando na dificuldade de se obter informações sobre ações de doação de caridade e como as ajudar.

Idealizamos o site como uma forma de instituições de caridade se conectarem com pessoas interessadas em realizarem doações

Motivação

O número de pessoas em situação de necessidade é alarmantemente alto (considerando que o ideal seria 0), por isso pensamos que qualquer forma de facilitar o auxílio a quem se encontra em situação de vulnerabilidade alimentar

No Brasil, **21,6 milhões de pessoas (27,6% da população) ainda têm dificuldade na aquisição de alimentos**

Diagrama de classe

Nosso diagrama de classe foi elaborado para organizar a visão da estrutura e dos relacionamentos da arquitetura do nosso projeto

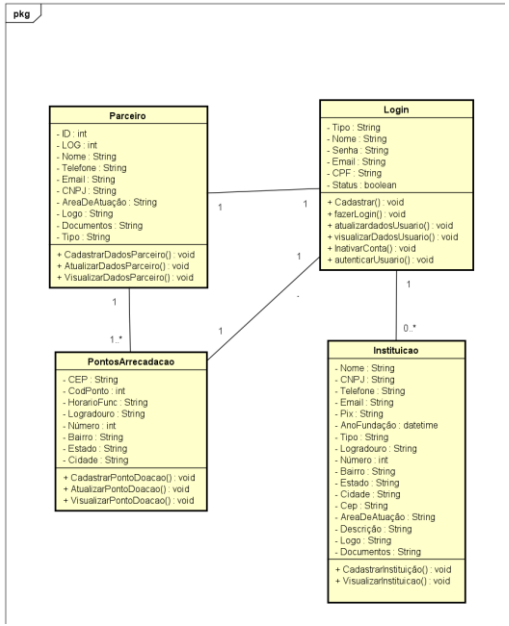
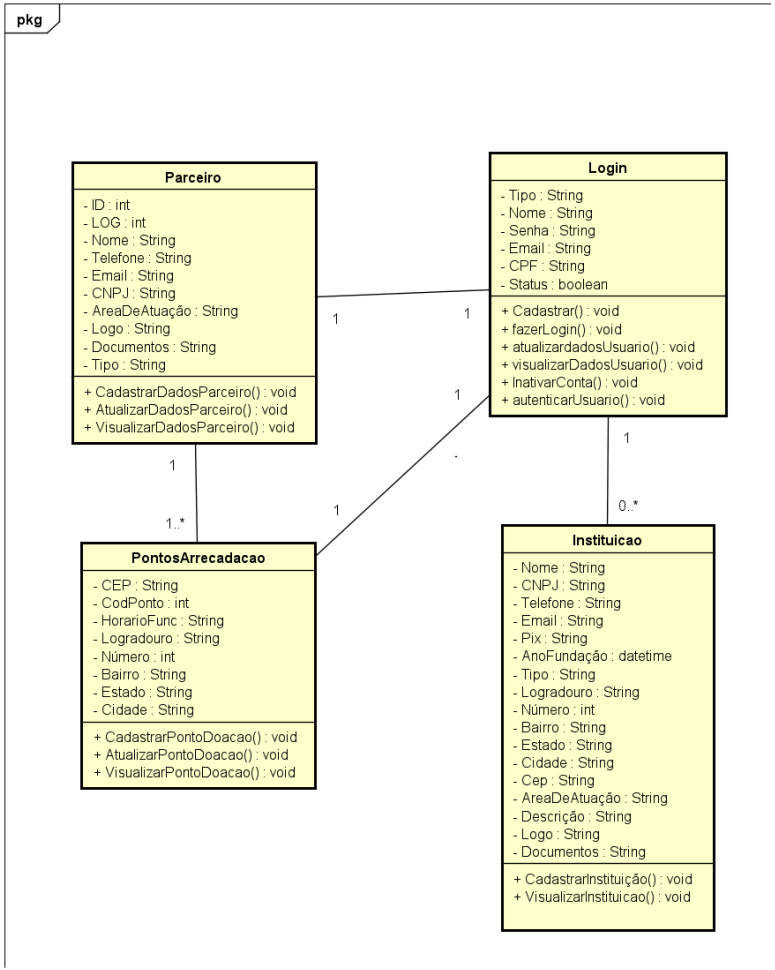
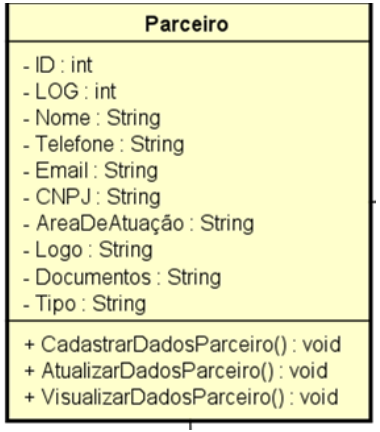


Diagrama de classe



Atributos



ID: Int que armazena o id único de cada parceiro

LOG: Int que armazena o momento de acesso do parceiro

Nome: String que armazena o nome do parceiro

Telefone: String que armazena o número de contato do parceiro

Email: String que armazena o E-mail de contato do parceiro

CNPJ: String que armazena o CNPJ do parceiro

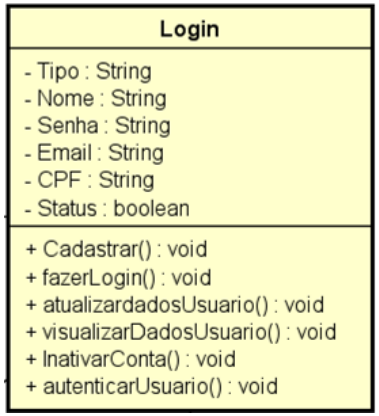
Área de atuação: String que recebe as informações da área de atuação do parceiro

Logo: String que recebe o link da logo do parceiro

Documentos: String que recebe os documentos para validação do parceiro

Tipo: String que recebe o tipo de parceiro

Atributos



Tipo: String que especifica se o login é de parceiro ou organização

Nome: String que especifica o nome de quem esta logando

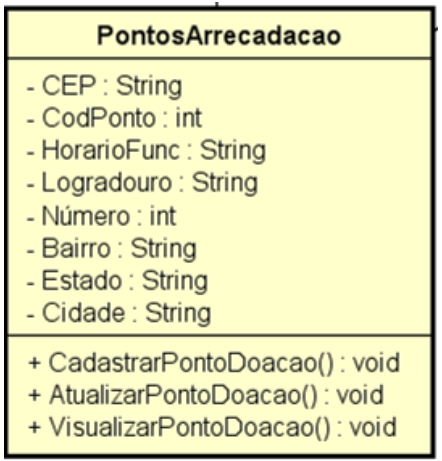
Senha: String que recebe a senha de login

Email: String que especifica o email de quem esta logando

CPF: String que especifica o CPF de quem esta logando

Status: Boolean que identifica se o login está ativo ou não

Atributos



CEP: String que identifica o cep do endereço

CodPonto: Int que identifica o código único do ponto

HorarioFunc: String que identifica o horário de funcionamento do ponto

Logradouro: String que armazena o logradouro do ponto

Numero: Int que armazena o número físico do ponto

Bairro: String que armazena o bairro em que se localiza o ponto

Estado: String que armazena o estado em que o ponto se localiza o ponto

Cidade: String que armazena a cidade em que se localiza o ponto

Atributos

Nome: String que identifica o nome da instituição

CNPJ: String que identifica o CNPJ da instituição

Telefone: String que identifica o número de contato da instituição

Email: String que identifica o endereço de e-mail da

instituição

Pix: String que identifica a chave PIX da instituição

AnoDeFundação: Datetime que identifica o ano de fundação da empresa

Tipo: String que identifica o que a instituição faz

Logradouro: String que identifica o logradouro da instituição

Numero: Int que identifica o número físico da instituição

Bairro: String que identifica o bairro onde se localiza a instituição

Estado: String que identifica o estado de localização da instituição

Cidade: String que identifica a cidade em que a instituição se localiza

CEP: String que identifica o cep

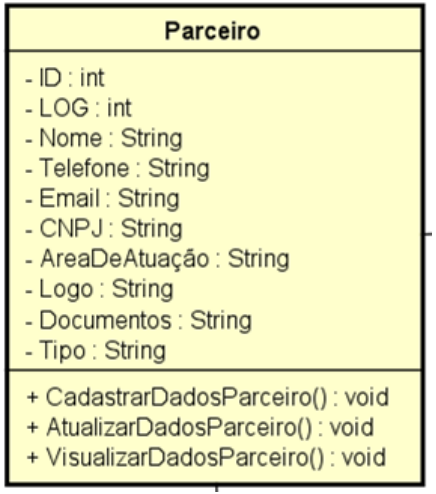
AreaDeAtuação: String que detalha a área de atuação da instituição

Descrição: String que recebe uma breve descrição da instituição

Logo: String que recebe o link da logo da instituição

Documentos: String que recebe os documentos para validação da instituição

Definição das funções

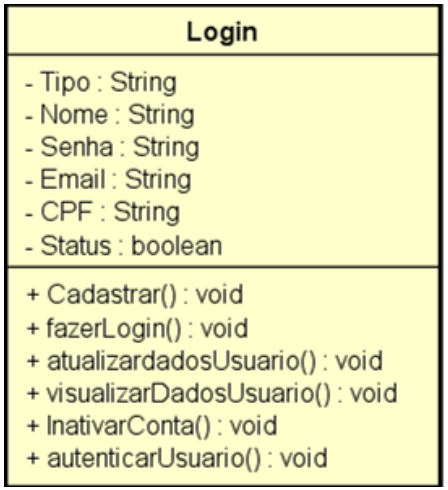


CadastrarDadosParceiro(): Função para cadastrar o parceiro no site

AtualizarDadosParceiro(): Função para atualizar os dados do parceiro já cadastrado

VisualizarDadosParceiro(): Função de visualização dos dados do parceiro já cadastrado

Definição de funções



Cadastrar(): Processo do qual o usuário cadastra suas informações para formar um perfil no site

fazerLogin(): Processo do qual o usuário faz login com suas credenciais cadastradas no site

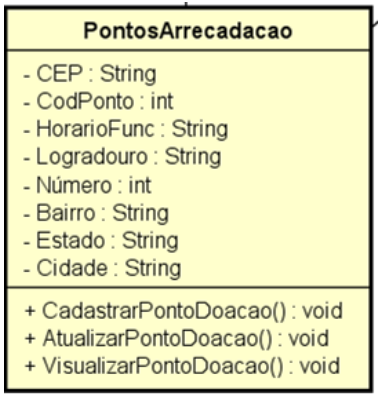
autenticarUsuario(): Processo do qual o sistema faz os processos de validação de um usuário para formalizar seu cadastro

inativarConta(): Inativa a conta cadastrada

atualizardadosUsuario(): Função para atualizar os dados cadastrados do usuário

visualizarDadosUsuario(): Função para visualizar os dados cadastrados do usuário

Definição de funções



CadastrarPontoDoacao(): Processo do qual o usuário cadastra um ponto de doação

VisualizarPontoDoacao(): Visualização do ponto de doação no site

AtualizarPontoDoacao(): Processo para atualização dos dados cadastrados

Definição de funções

VisualizarInstituicao():Processo para visualização da instituição no site

AlterarInstituicao():Processo para alteração dos dados cadastrados

CadastrarInstituição(): Processo de cadastro da instituição no site

VisualizarInstituicao()

```
export const listAll = async (): Promise<Organizacao[]> => {
    const organizacoes = await OrganizacaoModel.findAll();
    return organizacoes;
};
```

Retorna a lista completa de organizações registradas no sistema.

```
export const getByUsuarioId = async (idUsuario: number): Promise<Organizacao | null> => {
    const organizacao = await OrganizacaoModel.findOne({
        where: { idUsuario: idUsuario }
    });
    return organizacao;
};
```

Busca e retorna a organização vinculada a um determinado usuário (via idUsuario).

AlterarInstituicao()

```
export const create = async (dadosOrganizacao: Organizacao): Promise<Organizacao |
undefined> => {


    try{
    console.log(JSON.stringify(dadosOrganizacao));
        const novaOrganizacao = await OrganizacaoModel.create(dadosOrganizacao);
        return novaOrganizacao;
    } catch(error){
        console.log(">>> ", error)
        return undefined;
    }
};
```

Cria uma nova organização com os dados fornecidos. Caso ocorra erro na criação, retorna undefined.

CadastrarInstituição()

```
export const update = async (id: number, data: Partial<Organizacao>): Promise<Organizacao |
null> => {
    const organizacao = await OrganizacaoModel.findByPk(id);
    if (!organizacao) return null;

    await organizacao.update(data);
    return organizacao;

};
```

Atualiza os dados de uma organização com base no id informado. Retorna a organização atualizada ou null caso ela não exista.

VisualizarDadosParceiro()

```
export const listAll = async (): Promise<Parceiro[]> => {
    const parceiros = await ParceiroModel.findAll({
    include: [
      {
        model: UsuarioModel,
        where: { status: true },
      }
    ]
  });
    return parceiros;
};
```

Atualiza um parceiro existente com os dados fornecidos. Retorna o parceiro atualizado ou null se não for encontrado.

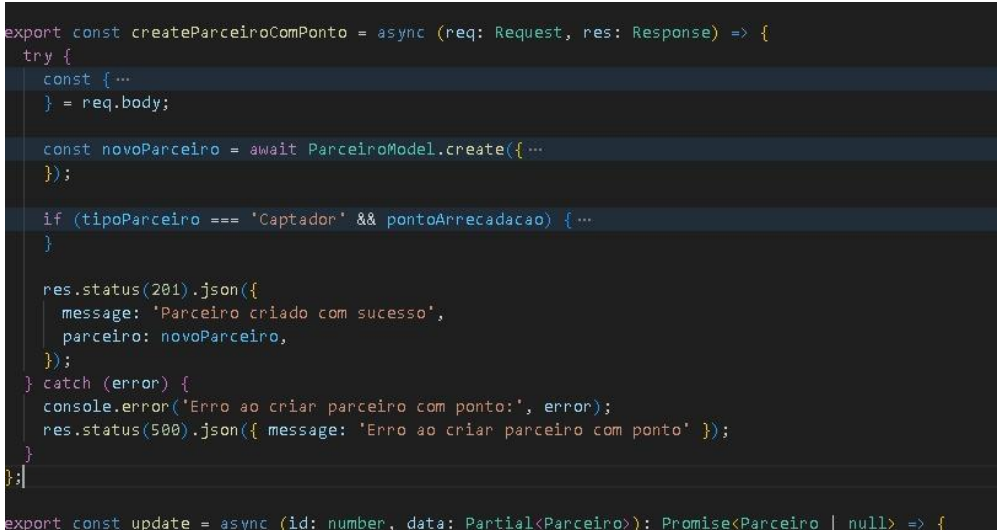
AtualizarDadosParceiro()

```
export const update = async (id: number, data: Partial<Parceiro>): Promise<Parceiro | null> => {
    const parceiro = await ParceiroModel.findByPk(id);
    if (!parceiro) return null;

    await parceiro.update(data);
    return parceiro;
};
```

Atualiza um parceiro existente com os dados fornecidos. Retorna o parceiro atualizado ou null se não for encontrado.

CadastrarDadosParceiro()



```
export const createParceiroComPonto = async (req: Request, res: Response) => {
  try {
    const {···
    } = req.body;

    const novoParceiro = await ParceiroModel.create({···
    });

    if (tipoParceiro === 'Captador' && pontoArrecadacao) {···
    }

    res.status(201).json({
      message: 'Parceiro criado com sucesso',
      parceiro: novoParceiro,
    });
  } catch (error) {
    console.error('Erro ao criar parceiro com ponto:', error);
    res.status(500).json({ message: 'Erro ao criar parceiro com ponto' });
  }
};

export const update = async (id: number, data: Partial<Parceiro>): Promise<Parceiro | null> => {
```

Cria um novo parceiro com os dados recebidos no corpo da requisição. Se o tipo for “Captador” e existir um ponto de arrecadação, ele associa os dados e retorna o novo parceiro. Em caso de erro, envia mensagem de erro com status 500.

VisualizarPontoDoacao()

```
export const listAll = async (): Promise<pontoArrecadacao[]> => {
  const pontosArrecadacao = await pontoArrecadacaoModel.findAll({
    include: [{
      model: ParceiroModel,
      as: 'parceiro',      // tem que ser o alias da associação
      attributes: ['nome'] // seleciona só o que precisa para economizar
    }]
  });
  return pontosArrecadacao;
};

export const getById = async (id:number): Promise<pontoArrecadacao | null> => {
    const pontoArrecadacao = await pontoArrecadacaoModel.findOne({
        where:{
            id: id
        }
    });
    return pontoArrecadacao;
};
```

listAll: Lista todos os pontos de arrecadação, incluindo o nome do parceiro associado.

getById: Busca um ponto de arrecadação específico pelo id.

CadastrarPontoDoacao()

```
export const create = async (dadosPontoArrecadacao: pontoArrecadacao): Promise<pontoArrecadacao> => {
    const novoPonto = await pontoArrecadacaoModel.create(dadosPontoArrecadacao);
    return novoPonto;
};
```

Cria um novo ponto de arrecadação com os dados fornecidos e retorna o objeto criado.

AtualizarPontoDoacao()

```
export const update = async (id: number, data: Partial<pontoArrecadacao>):
Promise<pontoArrecadacao | null> => {
    const PontoArrecadacao = await pontoArrecadacaoModel.findByPk(id);
    if (!PontoArrecadacao) return null;

    await PontoArrecadacao.update(data);
    return PontoArrecadacao;
};
```

Atualiza um ponto de arrecadação existente com os dados fornecidos. Se o ponto não for encontrado pelo id, retorna null. Caso contrário, atualiza e retorna o ponto.

autenticarUsuario()

```
authRouter.post("/", async (req: Request, res: Response) => {
  const { email, senha } = req.body;


  let usuario = await getByEmail(email)// achar no banco
  if(!usuario){
    res.status(401).json("E-mail ou senha inválidos")
    return
  }


  if(!await bcrypt.compare(senha, usuario.senha)){
    res.status(401).json("E-mail ou senha inválidos")
    return
  }
  const secret = process.env.AUTH_SECRET || "";
  const token = jwt.sign({ email: usuario.email, id: usuario.id }, secret);
  res.status(200).json({ token, id: usuario.id });


});
```

Esta rota autentica usuários, verificando email e senha no banco de dados. Se válidos, gera um token JWT com informações do usuário e retorna o token e ID. Caso contrário, retorna erro 401.

inativarConta()

```
export const desativar = async(id:number):Promise<boolean> =>{
    const [nro_linhas_alteradas] = await UsuarioModel.update({
        status: false
    }, {
        where:{
            id:id
        }
    })
    return !!nro_linhas_alteradas;
}
```

Função que desativa um usuário no banco, alterando seu status para false com base no ID fornecido. Retorna o número de linhas afetadas pela atualização.

atualizardadosUsuario()

```
export const update = async (id: number, data: Partial<Usuario>): Promise<Usuario| null> => {
    const usuario = await UsuarioModel.findByPk(id);
    if (!usuario) return null;

    await usuario.update(data);
    return usuario;
};
```

Atualiza os dados de um usuário existente, buscando-o pelo ID. Se encontrado, aplica as alterações parciais e retorna o usuário atualizado; caso contrário, retorna null.

Cadastrar()

```
export const create = async (dadosUsuario: Usuario): Promise<Usuario> => {
  try {
    const novoUsuario = await UsuarioModel.create(dadosUsuario);
    return novoUsuario;
  } catch (error) {
    console.error("❌ Erro no controller ao criar usuário:", error); // <-- Mostra erro detalhado
    throw error;
  }
};
```

Cria um novo usuário no banco de dados com os dados fornecidos. Retorna o registro criado ou exibe um erro detalhado no console em caso de falha.

visualizarDadosUsuario()

```
export const listAll = async (): Promise<Usuario[]> => {
    const usuarios = await UsuarioModel.findAll();
    return usuarios;
};

export const getById = async (id:number): Promise<Usuario| null> => {
    const usuario = await UsuarioModel.findOne({
        where:{
            id: id
        }
    });
    return usuario;
};
```

listAll retorna todos os usuários cadastrados, enquanto getById busca um usuário específico pelo ID, retornando seus dados ou null se não existir.