ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

# Projeto de Programação em Ambiente Web

Licenciatura em Engenharia Informática

2023/2024


Grupo 20

Bruno Miguel Magalhães Soares Pontvianne Ferreira - 8220191

Miguel Correia Ribeiro Rangel Tavares - 8220229

Francisco Jose Pinto Costa - 8220805

# Índice

## 1. Introdução

A Recicla Têxtil tem como objetivo desenvolver uma plataforma de doações de produtos têxtil a diversas entidades.

Este sistema consiste em dois componentes principais: BackOffice (portal de administração do sistema) e FrontOffice (aplicação web pública que permite o acesso ao sistema por parte de doadores ou entidades).

Qualquer doador tem acesso às diversas entidades registadas, sendo então possível realizar uma doação, têxtil ou monetária, para a entidade desejada. As doações têxteis realizadas recompensam o doador que a realizou, e/ou um possível doador que o convidou para utilizar a plataforma, com a atribuição de pontos que mais tarde podem ser usados para resgatar vales afetos a diversas empresas conhecidas a nível nacional.

Todo e qualquer tipo de validação pode ser realizada a partir do BackOffice, onde os funcionários da nossa empresa são responsáveis pela correta verificação das doações realizadas, entidades que submetem o seu registo para validação e registo de vales de empresas com parceria.

## 2. Credenciais de acesso

### Acesso ao BackOffice:

- Administrador:
  - Email: projetopaw2324082203@gmail.com
  - Password: admin082203
- Funcionário:
  - Email: testerFuncionario@gmail.com
  - Password: tester

### Acesso ao FrontOffice:

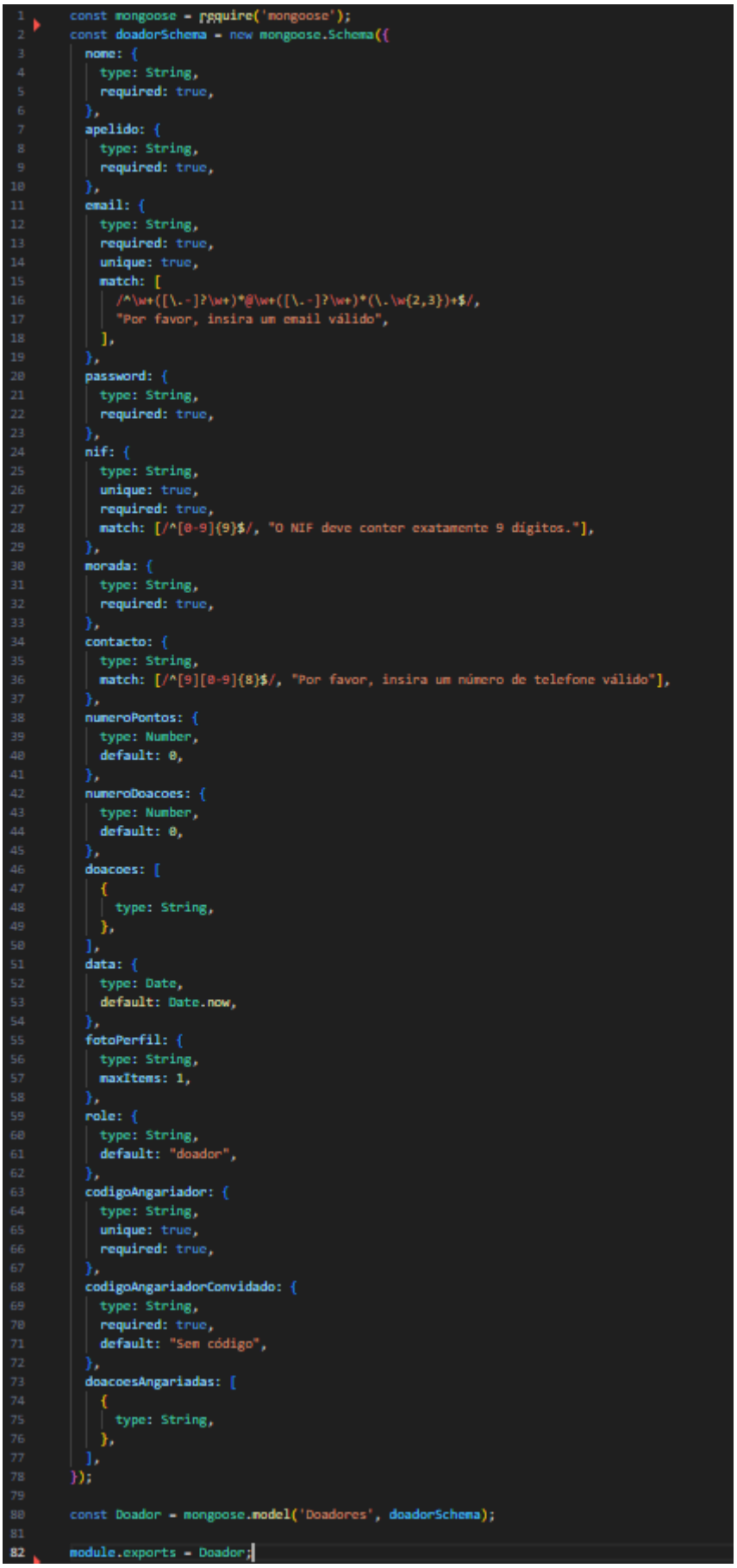
- Utilizador sem convite:
  - Email: 8220191@estg.ipp.pt
  - Password: 123
- Utilizador convidado:
  - Email: jrmr@estg.ipp.pt
  - Password: 123
- Entidade:
  - Email: defesa@gmail.com
  - Password: 123

## Acesso à base de dados:

- Connect:
  - mongodb+srv://BrunoFerreira:wPc1l5M7rtJUBAux@reciclatextil.lufifqt.mongodb.net/

## 3. Modelos

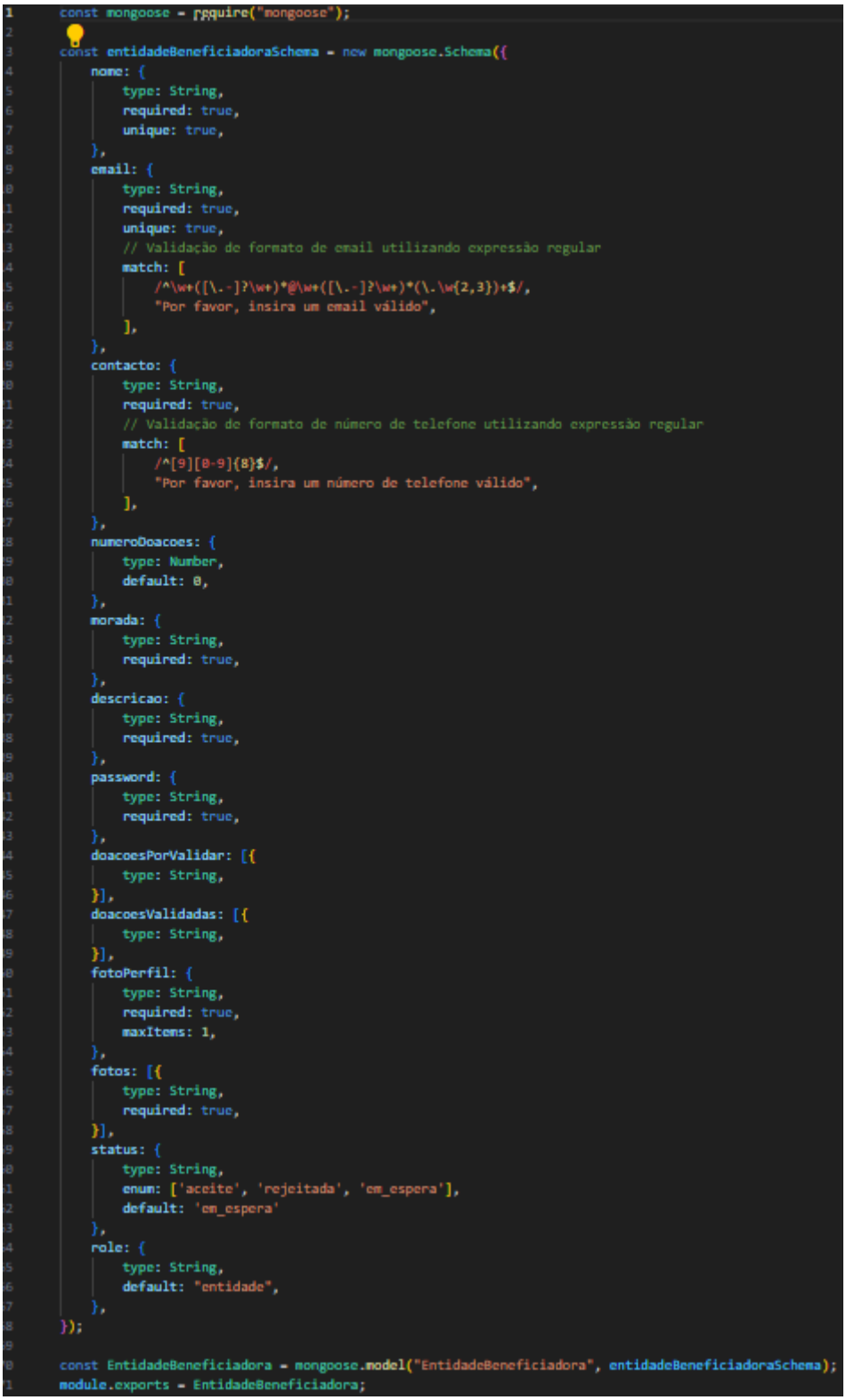
### 2.1 Doador



```
const mongoose = require('mongoose');
const doadorSchema = new mongoose.Schema({
  nome: {
    type: String,
    required: true,
  },
  apelido: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
    match: [
      /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/,
      "Por favor, insira um email válido",
    ],
  },
  password: {
    type: String,
    required: true,
  },
  nif: {
    type: String,
    unique: true,
    required: true,
    match: [/^[0-9]{9}$/, "O NIF deve conter exatamente 9 dígitos."],
  },
  morada: {
    type: String,
    required: true,
  },
  contacto: {
    type: String,
    match: [/^[9][0-9]{8}$/, "Por favor, insira um número de telefone válido"],
  },
  numeroPontos: {
    type: Number,
    default: 0,
  },
  numeroDoacoes: {
    type: Number,
    default: 0,
  },
  doacoes: [
    {
      type: String,
    },
  ],
  data: {
    type: Date,
    default: Date.now,
  },
  fotoPerfil: {
    type: String,
    maxItems: 1,
  },
  role: {
    type: String,
    default: "doador",
  },
  codigoAngariador: {
    type: String,
    unique: true,
    required: true,
  },
  codigoAngariadorConvidado: {
    type: String,
    required: true,
    default: "Sem código",
  },
  doacoesAngariadas: [
    {
      type: String,
    },
  ],
});

const Doador = mongoose.model('Doadores', doadorSchema);

module.exports = Doador;
```

## 2.2 Entidade


_id: ObjectId('6665d247b054fe34e30b9ab0')
nome : "EntidadeABC"
email : "entidadeabc@gmail.com"
contacto : "911123123"
numeroDoacoes : 0
morada : "R. Benjamim Pinto dos Santos"
descricao : "Entidade de teste e utilizada no relatório."
password : "$2a$10$qgrXdICYt8FWNd6at1ieQeZpxBKf74cuokEoIkEWqZwu1QUeeNYMe"
doacoesPorValidar : Array (empty)
doacoesValidadas : Array (empty)
fotoPerfil : "uploads\a5a1e8b52dca572cde082cbda79f8a6c"
fotos : Array (empty)
status : "em_espera"
role : "entidade"
__v : 0



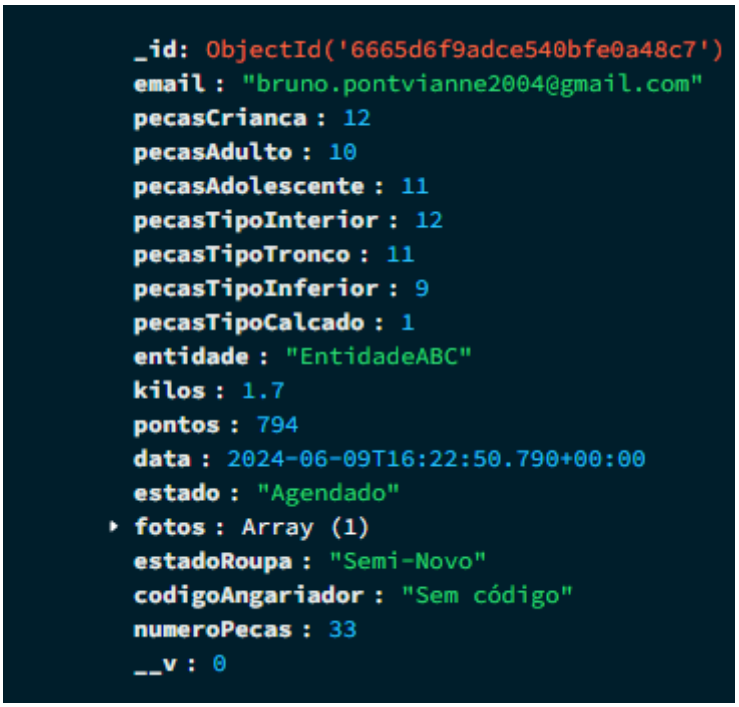
```
const mongoose = require("mongoose");

const entidadeBeneficiadoraSchema = new mongoose.Schema({
    nome: {
        type: String,
        required: true,
        unique: true,
    },
    email: {
        type: String,
        required: true,
        unique: true,
        // Validação de formato de email utilizando expressão regular
        match: [
            /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/,
            "Por favor, insira um email válido",
        ],
    },
    contacto: {
        type: String,
        required: true,
        // Validação de formato de número de telefone utilizando expressão regular
        match: [
            /^[9][0-9]{8}$/,
            "Por favor, insira um número de telefone válido",
        ],
    },
    numeroDoacoes: {
        type: Number,
        default: 0,
    },
    morada: {
        type: String,
        required: true,
    },
    descricao: {
        type: String,
        required: true,
    },
    password: {
        type: String,
        required: true,
    },
    doacoesPorValidar: [{
        type: String,
    }],
    doacoesValidadas: [{
        type: String,
    }],
    fotoPerfil: {
        type: String,
        required: true,
        maxItems: 1,
    },
    fotos: [{
        type: String,
        required: true,
    }],
    status: {
        type: String,
        enum: ['aceite', 'rejeitada', 'em_espera'],
        default: 'em_espera'
    },
    role: {
        type: String,
        default: "entidade",
    },
});

const EntidadeBeneficiadora = mongoose.model("EntidadeBeneficiadora", entidadeBeneficiadoraSchema);
module.exports = EntidadeBeneficiadora;
```

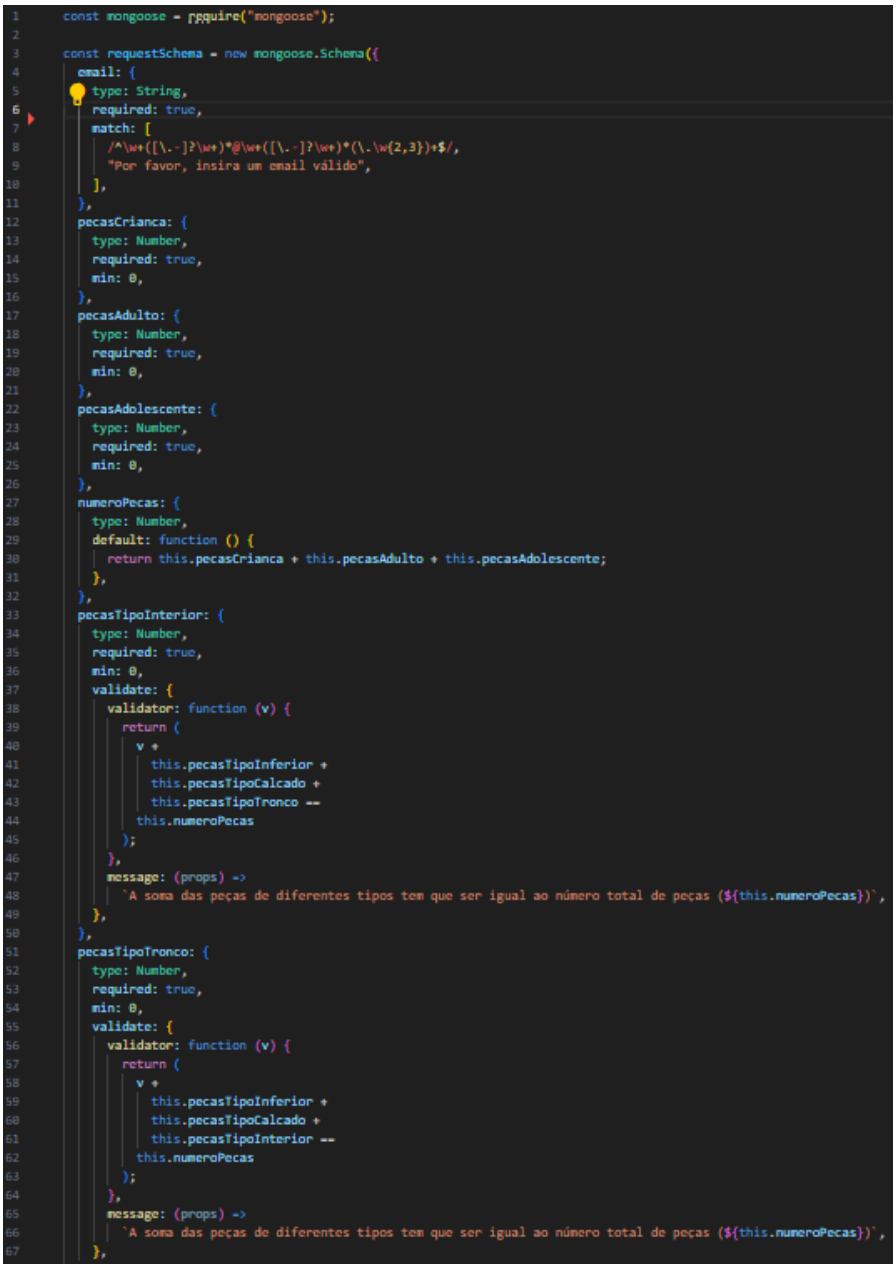
## 2.3 Funcionário


```
_id: ObjectId('6634a7ccbbce1e48a4315d38')
nome : "Miguel"
apelido : "Tavares"
email : "projetopaw2324082202@gmail.com"
numero_empresa : "4930"
password : "$2a$10$kAODav3DPkHl3Yqsy8WJQeJWgJnKo6mSgMnvVn/MO6/WeF0WXVc7S"
admin : true
contacto : "999999999"
morada : "Maia"
role : "admin"
getsEmails : true
__v : 0
```

```
const mongoose = require("mongoose");

const funcionarioSchema = new mongoose.Schema({
    nome: {
        type: String,
        required: true,
    },
    apelido: {
        type: String,
        required: true,
    },
    email: {
        type: String,
        required: true,
        unique: true,
        // Validação de formato de email utilizando expressão regular
        match: [
            /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/,
            "Por favor, insira um email válido",
        ],
    },
    numero_empresa: {
        type: String,
        required: true,
        unique: true,
    },
    password: {
        type: String,
        required: true,
    },
    admin: {
        type: Boolean,
        default: false,
    },
    contacto: {
        type: String,
        required: true,
        match: [
            /^[9][0-9]{8}$/,
            "Por favor, insira um número de telefone válido",
        ],
    },
    morada: {
        type: String,
        required: true,
    },
    role:{
        type: String,
        enum: ['funcionario', 'admin'],
        default: "funcionario",
    }, getsEmails: {
        type: Boolean,
        default: false,
    }
});

const Funcionario = mongoose.model("Funcionario", funcionarioSchema);
module.exports = Funcionario;
```

## 2.4 Doação


_id: ObjectId('6665d6f9adce540bfe0a48c7')
email : "bruno.pontvianne2004@gmail.com"
pecasCrianca : 12
pecasAdulto : 10
pecasAdolescente : 11
pecasTipoInterior : 12
pecasTipoTronco : 11
pecasTipoInferior : 9
pecasTipoCalcado : 1
entidade : "EntidadeABC"
kilos : 1.7
pontos : 794
data : 2024-06-09T16:22:50.790+00:00
estado : "Agendado"
fotos : Array (1)
estadoRoupa : "Semi-Novo"
codigoAngariador : "Sem código"
numeroPecas : 33
__v : 0

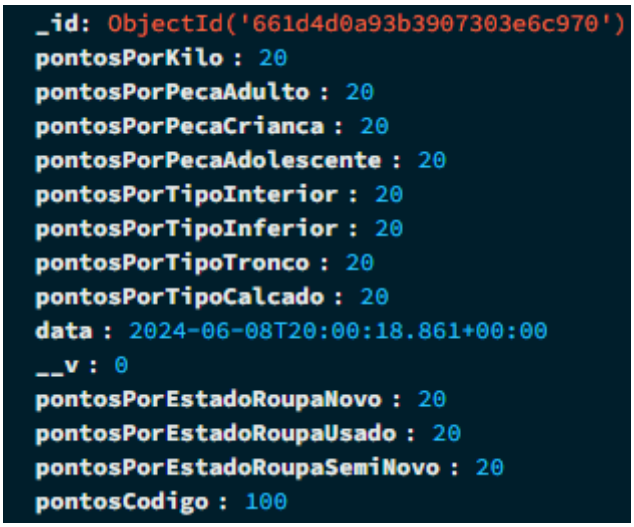


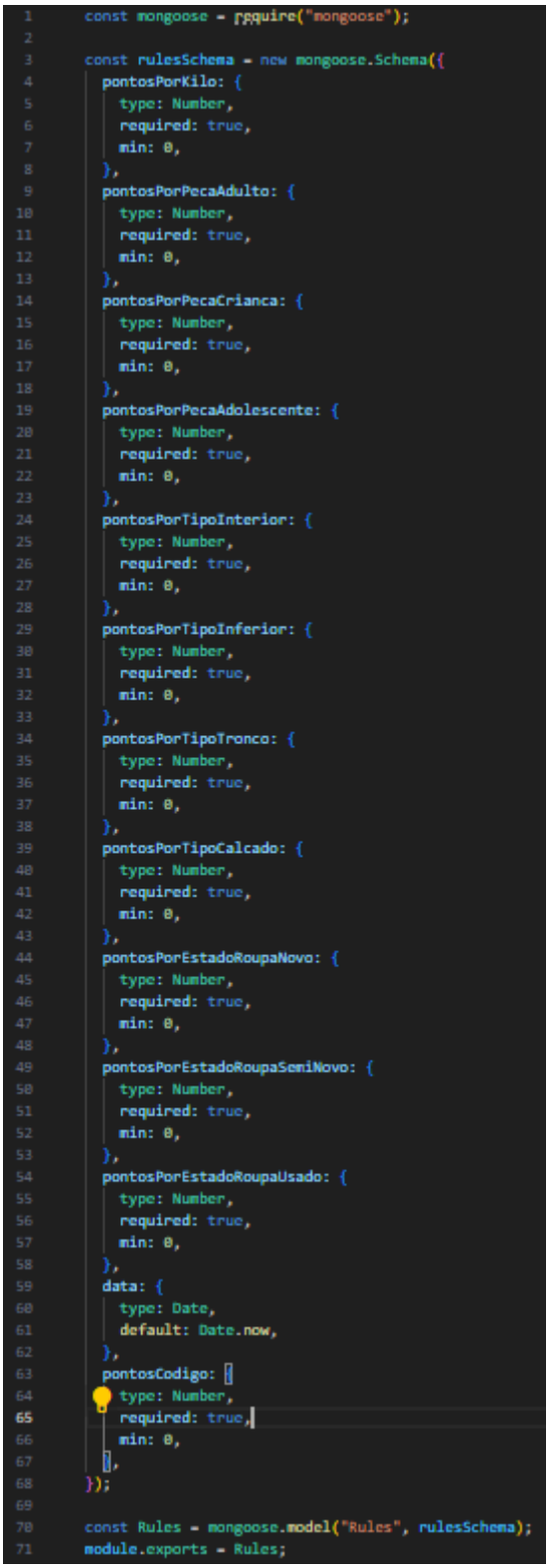
```
const mongoose = require("mongoose");

const requestSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
    match: [
      /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/,
      "Por favor, insira um email válido",
    ],
  },
  pecasCrianca: {
    type: Number,
    required: true,
    min: 0,
  },
  pecasAdulto: {
    type: Number,
    required: true,
    min: 0,
  },
  pecasAdolescente: {
    type: Number,
    required: true,
    min: 0,
  },
  numeroPecas: {
    type: Number,
    default: function () {
      return this.pecasCrianca + this.pecasAdulto + this.pecasAdolescente;
    },
  },
  pecasTipoInterior: {
    type: Number,
    required: true,
    min: 0,
    validate: {
      validator: function (v) {
        return (
          v +
            this.pecasTipoInferior +
            this.pecasTipoCalcado +
            this.pecasTipoTronco ==
          this.numeroPecas
        );
      },
      message: (props) =>
        `A soma das peças de diferentes tipos tem que ser igual ao número total de peças (${this.numeroPecas})`,
    },
  },
  pecasTipoTronco: {
    type: Number,
    required: true,
    min: 0,
    validate: {
      validator: function (v) {
        return (
          v +
            this.pecasTipoInferior +
            this.pecasTipoCalcado +
            this.pecasTipoInterior ==
          this.numeroPecas
        );
      },
      message: (props) =>
        `A soma das peças de diferentes tipos tem que ser igual ao número total de peças (${this.numeroPecas})`,
    },
```

```
  },
  pecasTipoInferior: {
    type: Number,
    required: true,
    min: 0,
    validate: {
      validator: function (v) {
        return (
          v +
            this.pecasTipoInterior +
            this.pecasTipoCalcado +
            this.pecasTipoTronco ==
          this.numeroPecas
        );
      },
      message: (props) =>
        `A soma das peças de diferentes tipos tem que ser igual ao número total de peças (${this.numeroPecas})`,
    },
  },
  pecasTipoCalcado: {
    type: Number,
    required: true,
    min: 0,
    validate: {
      validator: function (v) {
        return (
          v +
            this.pecasTipoInferior +
            this.pecasTipoInterior +
            this.pecasTipoTronco ==
          this.numeroPecas
        );
      },
      message: (props) =>
        `A soma das peças de diferentes tipos tem que ser igual ao número total de peças (${this.numeroPecas})`,
    },
  },
  entidade: {
    type: String,
    required: true,
  },
  kilos: {
    type: Number,
    required: true,
    min: 0.1,
  },
  pontos: {
    type: Number,
    default: 0,
  },
  data: {
    type: Date,
    required: true,
    validate: {
      validator: function (value) {
        const today = new Date();
        return value.getDay() >= today.getDay();
      },
      message: "A data deve ser estabelecida após hoje",
    },
  },
  estado: {
    type: String,
    enum: [
      "Agendado",
      "Por iniciar",
      "Aprovado",
      "A realizar",
      "Validado",
      "Cancelado",
    ],
    default: "Agendado",
  },
```

```
  fotos: [
    {
      type: String,
      required: true,
    },
  ],
  estadoRoupa: {
    type: String,
    enum: ["Novo", "Semi-Novo", "Usado"],
    default: "Usado",
  },
  codigoAngariador: {
    type: String,
  },
});

const Requests = mongoose.model("Requests", requestSchema);
module.exports = Requests;
```

## 2.5 Pontos


_id: ObjectId('661d4d0a93b3907303e6c970')
pontosPorKilo : 20
pontosPorPecaAdulto : 20
pontosPorPecaCrianca : 20
pontosPorPecaAdolescente : 20
pontosPorTipoInterior : 20
pontosPorTipoInferior : 20
pontosPorTipoTronco : 20
pontosPorTipoCalcado : 20
data : 2024-06-08T20:00:18.861+00:00
__v : 0
pontosPorEstadoRoupaNovo : 20
pontosPorEstadoRoupaUsado : 20
pontosPorEstadoRoupaSemiNovo : 20
pontosCodigo : 100



```
const mongoose = require("mongoose");

const rulesSchema = new mongoose.Schema({
  pontosPorKilo: {
    type: Number,
    required: true,
    min: 0,
  },
  pontosPorPecaAdulto: {
    type: Number,
    required: true,
    min: 0,
  },
  pontosPorPecaCrianca: {
    type: Number,
    required: true,
    min: 0,
  },
  pontosPorPecaAdolescente: {
    type: Number,
    required: true,
    min: 0,
  },
  pontosPorTipoInterior: {
    type: Number,
    required: true,
    min: 0,
  },
  pontosPorTipoInferior: {
    type: Number,
    required: true,
    min: 0,
  },
  pontosPorTipoTronco: {
    type: Number,
    required: true,
    min: 0,
  },
  pontosPorTipoCalcado: {
    type: Number,
    required: true,
    min: 0,
  },
  pontosPorEstadoRoupaNovo: {
    type: Number,
    required: true,
    min: 0,
  },
  pontosPorEstadoRoupaSemiNovo: {
    type: Number,
    required: true,
    min: 0,
  },
  pontosPorEstadoRoupaUsado: {
    type: Number,
    required: true,
    min: 0,
  },
  data: {
    type: Date,
    default: Date.now,
  },
  pontosCodigo: [
    type: Number,
    required: true,
    min: 0,
  ],
});

const Rules = mongoose.model("Rules", rulesSchema);
module.exports = Rules;
```

## 2.6 Vales


```
const mongoose = require("mongoose");

const valeSchema = new mongoose.Schema({
  empresa: {
    type: String,
    required: true,
  },
  custoPontos: {
    type: Number,
    required: true,
    min: 0,
  },
  numeroResgates: {
    type: Number,
    default: 0,
  },
  descricao: {
    type: String,
    required: true,
  },
  fotos: [
    {
      type: String,
      required: true,
    },
  ],
});

const vales = mongoose.model("vale", valeSchema);
module.exports = vales;
```

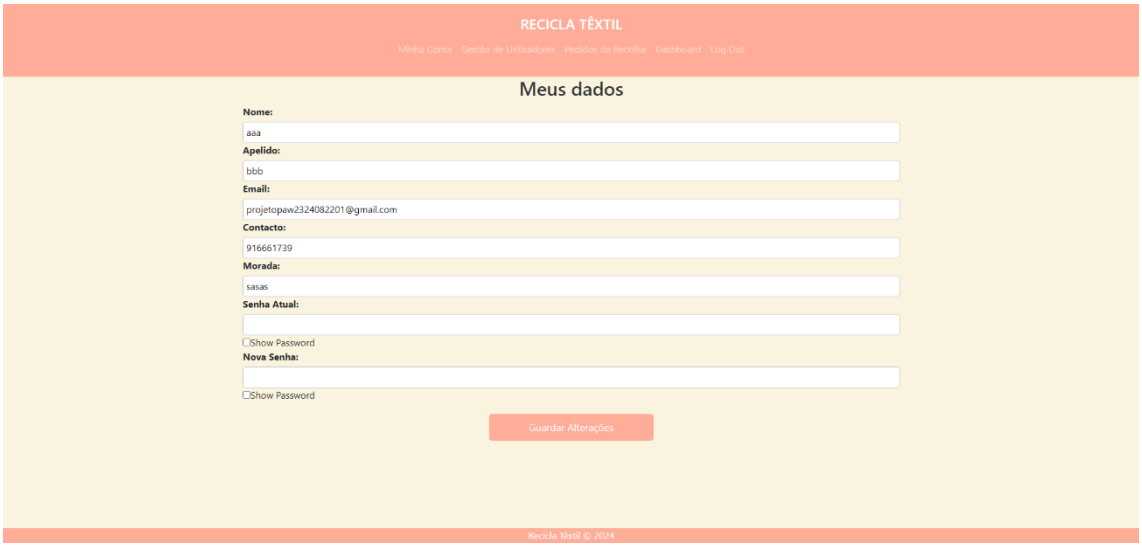
## 4. Funcionalidades

Todas as funcionalidades requisitadas foram desenvolvidas pelo grupo.

### 3.1 Administradores/Funcionários da plataforma

Os funcionários da empresa têm acesso, a partir do BackOffice, a:

- Editar a sua própria conta a partir do menu “Minha conta”:



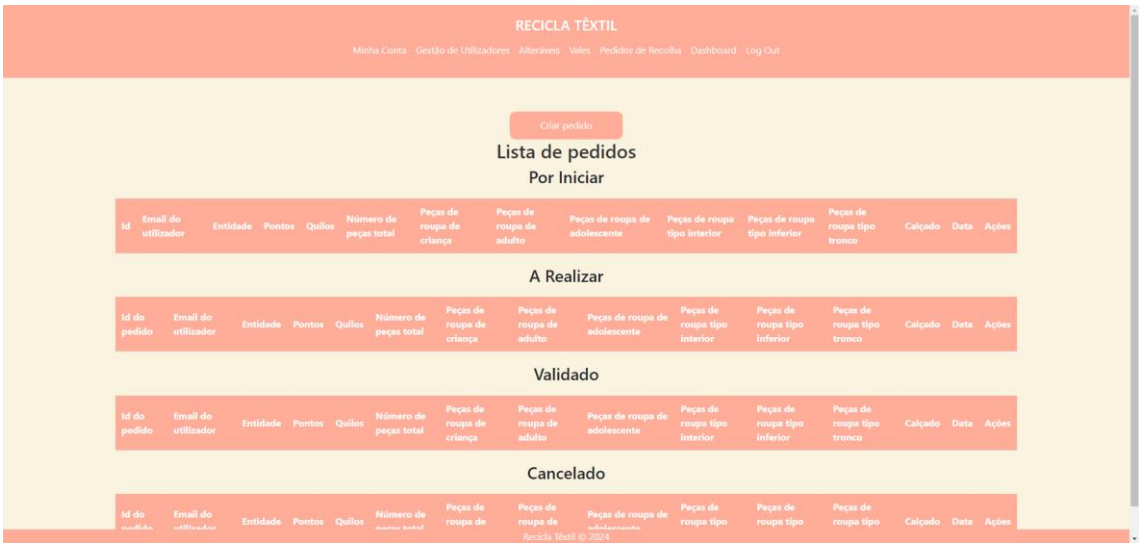

- Gestão de todo o tipo de utilizadores, caso o funcionário seja um administrador da empresa este tem acesso a um maior número de funcionalidades, através do menu “Gestão de Utilizadores”. Através da API do MailGun, que por limitação apenas nos deixa registar 5 emails, os administradores registados que estão disponíveis para receber emails são notificados. Exemplo:

  Caso o funcionário seja um administrador:

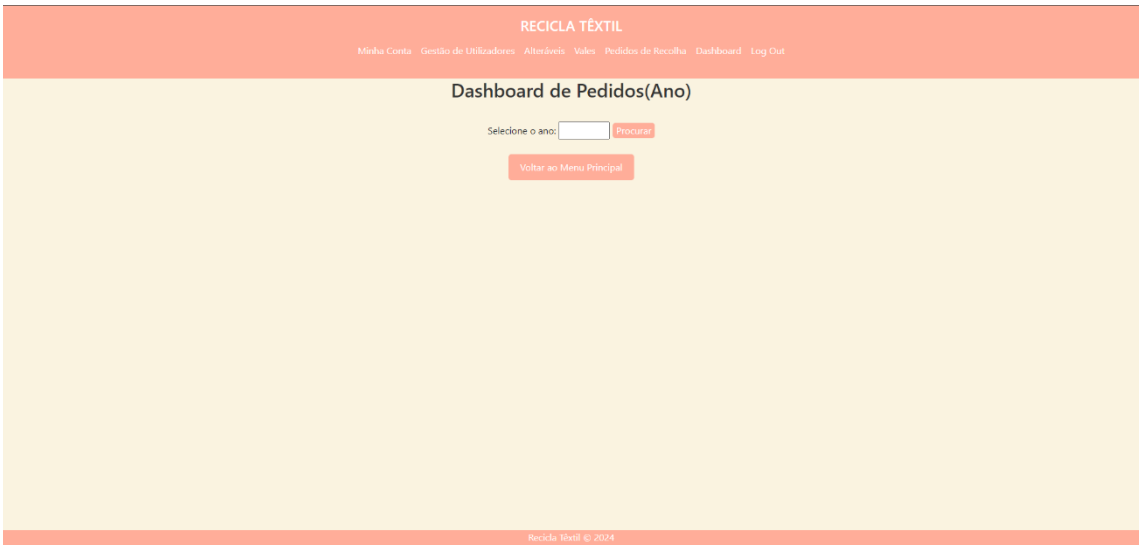

Caso o funcionário não seja um administrador:


- Gerir as regras de atribuição de pontos para as doações (APENAS ADMINISTRADORES), através do menu “Alteráveis”:


- Gestão dos pedidos de recolha dos doadores, através do menu “Pedidos de Recolha”:




- Dashboard, através do menu “Dashboard”:

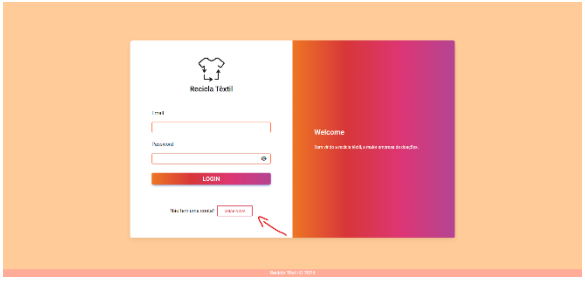


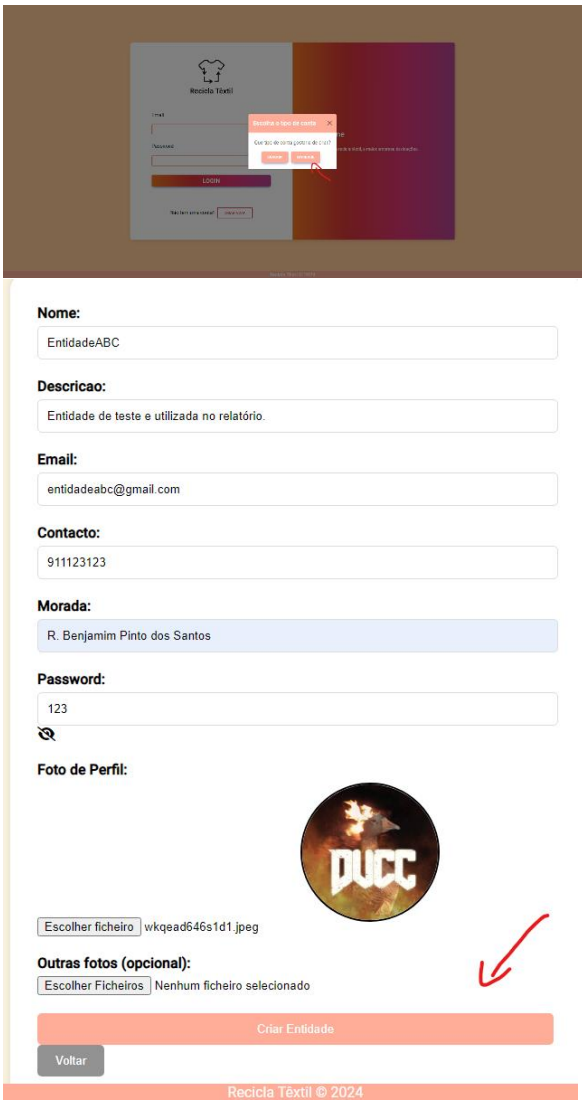
- Os administradores da plataforma podem também registar e remover os vales das empresas com parceria através do menu “Vales”:


## 3.2 Entidades beneficiadoras

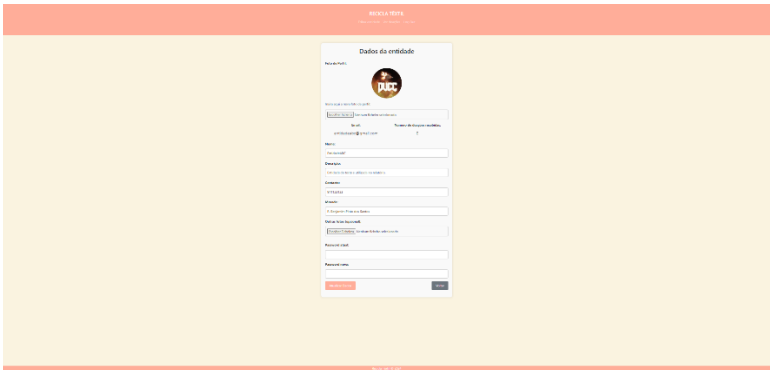
As entidades interessadas têm acesso, a partir do FrontOffice, a:

- Registarem-se na plataforma e ficarem eleitas para o processo de validação por parte da Recicla Têxtil, através dos seguintes passos:

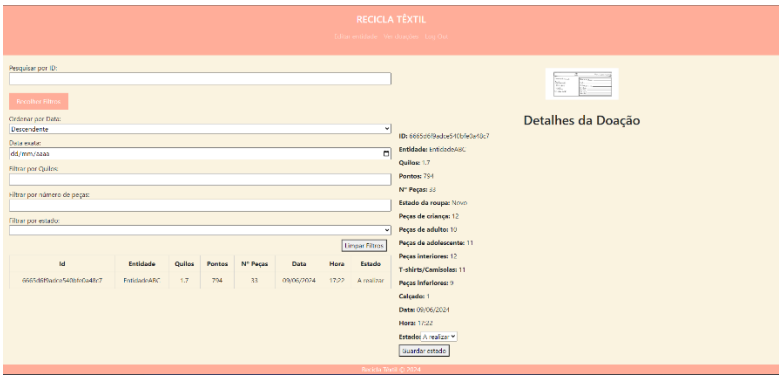





- Editar as suas informações, através do menu “Editar entidade”:



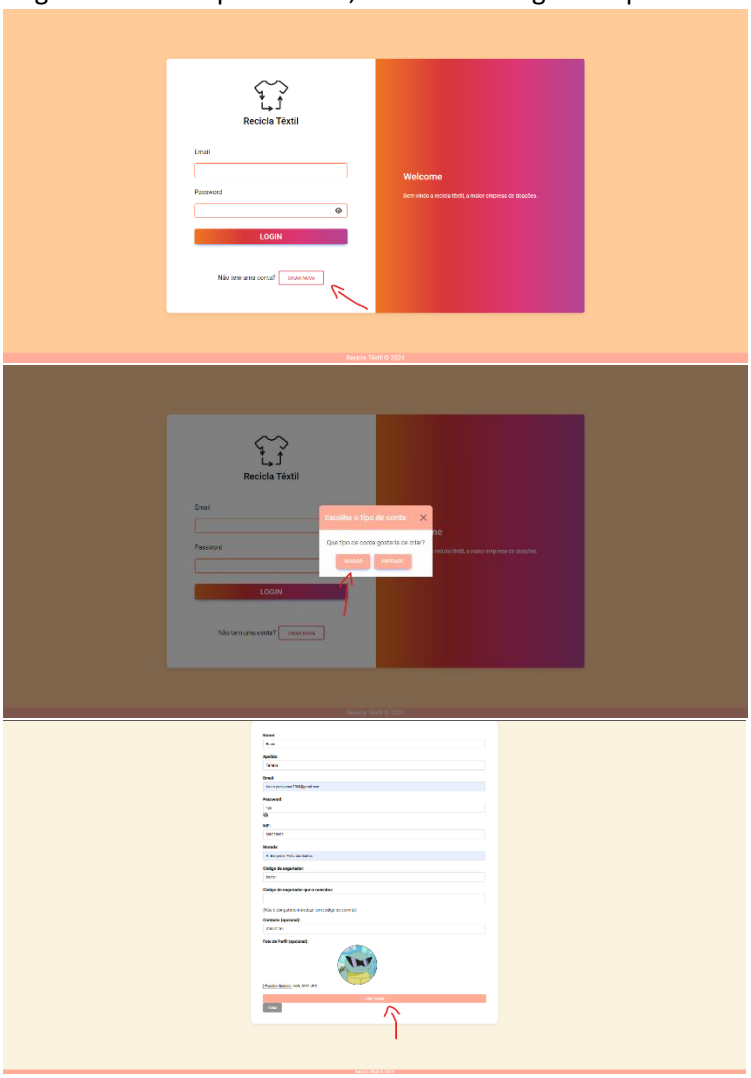
- Consultar informações e validar doações associadas, através do menu “Ver doações”:



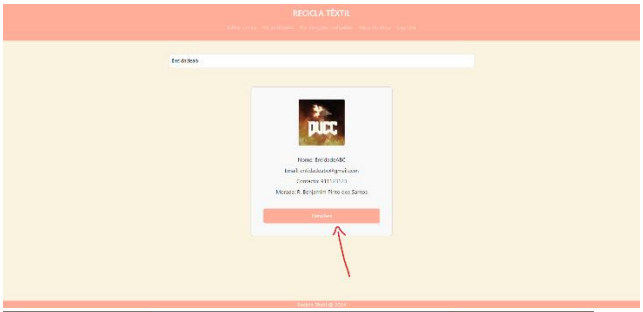

## 3.3 Doadores

Os doadores interessados têm acesso, a partir do FrontOffice, a:

- Registarem-se na plataforma, através dos seguintes passos:

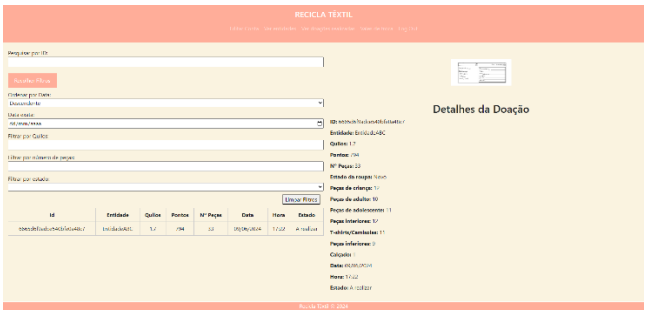


- Efetuar doações a entidades registadas no sistema através do menu “Ver entidades”, onde é possível pesquisar pela entidade pretendida e doar financeiramente, através da integração da API do PayPal, ou a nível têxtil:

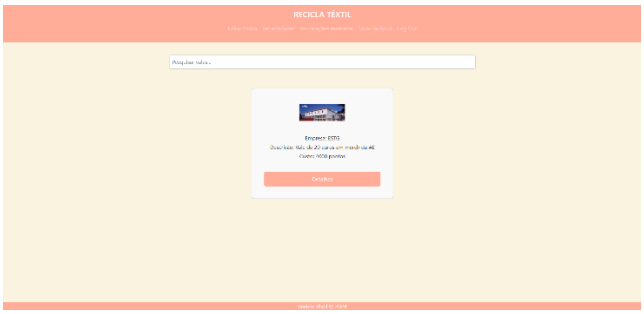



(Caso seja uma doação têxtil):


- Verificar o estado e informações de todas as doações que realizou previamente, através do menu "Ver doações realizadas":




- Resgatar vales de promoção com base nos pontos provenientes das doações, através do menu “Vales de troca”:




## 5. Funcionalidades propostas

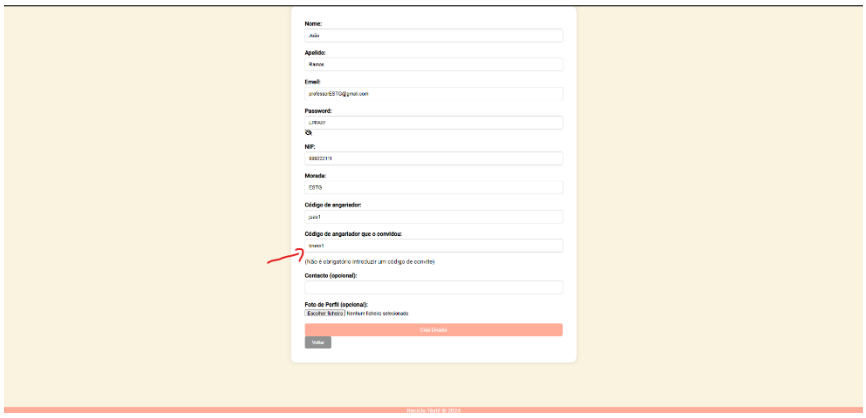
### 4.1 Sistema de envio de emails (integração com MailGun)

A plataforma dispõe de um sistema de notificações baseado na API do MailGun que notifica um limitado número de funcionários sempre que é efetuado um registo de um doador e/ou entidade. Consequentemente o sistema também notifica que determinada entidade está a aguardar a sua validação, e (caso seja esse o caso) quando essa entidade for validada.
O MailGun está "configurado" no ficheiro "emailController.js".

```
const Funcionario = require('../models/funcionarioModel');
const formData = require("form-data");
const Mailgun = require("mailgun.js");
const mailgun = new Mailgun(formData);
const mg = mailgun.client({username: "reciclaTextil", key: process.env.MAILGUN_API_KEY})

const sendEmail = async (emailInfo) => {
    try {
        const Funcionarios = await Funcionario.find();
        Funcionarios.forEach(async (funcionario) => {
            if (funcionario.role === "admin" && funcionario.getsEmails) {
                const emailOptions = {
                    from: '"Recicla Têxtil" <reciclaTextil@gmail.com>',
                    to: funcionario.email,
                    subject: emailInfo.subject,
                    html: emailInfo.text
                };
                await mg.messages.create(process.env.MAILGUN_DOMAIN, emailOptions);
            }
        });
    } catch (err) {
        console.log(err);
        res.status(500).render("error", {
            message: "Ocorreu um erro ao enviar email",
            error: err,
        });
    }
}

module.exports = {
    sendEmail
};
```

### 4.2 Implementação de sistema de angariações

Através do código de angariador associado a um doador previamente registado, um novo doador pode referenciar o tal doador ao preencher o campo "Código de angariador que o convidou". Se for esse o caso, o doador referenciado passa a receber um número estabelecido de pontos, definido pelos administradores da plataforma na página "Alteráveis" no BackOffice.



### 4.3 Integração com API de pagamento para doações diretas em Euros (PayPal)

Como foi previamente referenciado, também é possível doar monetariamente para uma entidade registada. Para tal acontecer basta o doador escolher o botão PayPal presente nos detalhes da entidade escolhida.


## 6. Estrutura analítica do projeto

### 5.1 Arquitetura

#### *5.1.1 Explicação*

• No nosso projeto, a arquitetura cliente-servidor foi implementada através da framework Angular para o frontend, no FrontOffice, e utilizando ficheiros .ejs para o BackOffice (cliente) e Node.js como runtime com a framework ExpressJS para o backend (servidor).

• O frontend, Angular e ficheiros.ejs, é responsável por interagir com o utilizador final, apresentando a interface gráfica e enviando HTTP requests para o servidor.

• O servidor Express processa esses requests, e comunica com a nossa base de dados em MongoDB para obter ou manipular dados, de seguida é enviada uma resposta com os resultados do respetivo request.

### *5.1.2 BackOffice*

No BackOffice obedecemos ao padrão MVC (Model-View-Controller). Este separa a aplicação em três componentes que se conectam, ou seja:

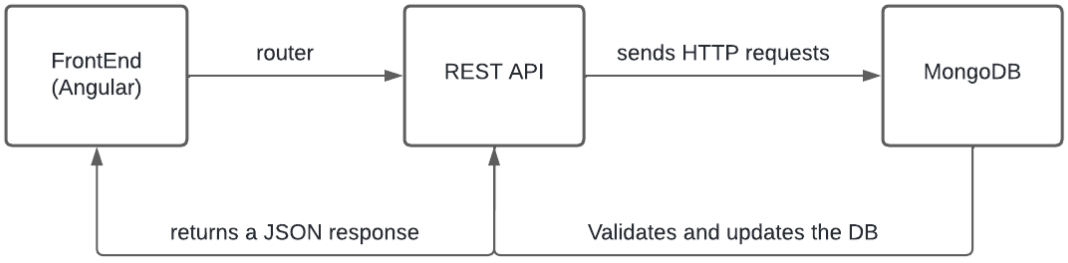
- Model
    - Implementados através do mongoose para definir os esquemas da base de dados.
- View
    - Desenvolvidas através de ficheiros .ejs.
    - Estas representam o frontend desta parte do projeto.
    - São utilizadas para receber e enviar os dados através do router para as respetivas rotas. Rotas essas que comunicam com os DIVERSOS controllers.
- Controller
    - Responsável por comunicar com a base de dados através de HTTP requests.
    - Estes processam os dados recebidos no request , validam-nos e manipulam a base de dados em prol dessa informação.

### *5.1.3 FrontOffice*

Através da framework Angular, o processo de desenvolvimento de um frontend integrado com a API previamente desenvolvido tornou-se mais fácil e direto.

Devido à modelação através de componentes e ao facto de ter sido desenvolvido uma SPA, características próprias desta framework, tornou-se mais acessível a reutilização e a otimização de código, bem como a vasta melhoria na sua leitura e implementação.

O frontend desenvolvido para o FrontOffice, comunica com o backend implementado em EJS, tirando partido dos seus endpoints e separando, de uma forma mais percetível, o desenvolvimento para o cliente (client-side) e para o servidor (server-side).




## 5.2 Desenvolvimento do projeto

O projeto foi organizado de forma a facilitar a sua manutenção e clareza.
Vamos detalhar a sua estrutura principal:

### *5.2.1 Estrutura do BackOffice*

- controllers: pasta onde se encontram todos os ficheiros respetivos aos métodos que garantem a integração da base de dados bem como a integração do MailGun.
- models: pasta onde se encontram todos os schemas da nossa base de dados,
- public: pasta onde se encontram todos os ficheiros javascript e css que estão associados às views.

- routes: pasta com todas as rotas utilizadas, quer pelo BackOffice quer pelo FrontOffice.
- views: pasta com todos os ficheiros .ejs, ou seja, frontend do BackOffice.
- app.js: Ficheiro executado para iniciar o servidor e os middlewares definidos.
- swagger.json: Ficheiro que configura a documentação de todas as rotas que retornam .json.

(Toda a documentação relacionada com as rotas encontra-se em: http://localhost:3000/api-docs/)

### *5.2.2 Estrutura do FrontOffice (Angular)*

- componentes: pasta onde se encontram todos os componentes utilizados no desenvolvimento do Frontend.
    - about-us: componente que contém um curto resumo da empresa.
    - check-donations: componente que contém todas as informações sobre as doações do doador/entidade autenticada. Este componente permite a aplicação de filtros de pesquisa. É reutilizado pelos dois tipos de utilizadores. Quando os doadores acedem a este componente é lhes permitido que cancelem a sua doação caso ela ainda esteja agendada. No caso das entidades, estas podem validar uma doação caso esteja tenha sido aprovada pelos funcionários da Recicla Têxtil, ou seja, se o estado da doação foi "A realizar".
    - check-entitys: componente acessível pelos utilizadores que carrega a informação de todas as entidades validadas no sistema. Este componente permite a pesquisa dinâmica das diversas entidades. Cada entidade tem um botão de doação associado, a partir deste componente é possível ser redirecionado para o componente "criar-doacao".
    - create-entity e create-user: componentes responsáveis pela criação de novas contas.
    - criar-doacao: componente que permite o registo de uma doação para a entidade previamente selecionada.
    - editar-entidade e editar-user: componentes responsáveis pela edição das respetivas contas.
    - entidade-initial-page e user-initial-page: componentes iniciais. Este componente "carrega" o componente previamente mencionado "about-us" e age como página inicial após ser dado o login.
    - footer: footer da aplicação.
    - header-entity e header-user: menus acessíveis pelos respetivos tipos de conta.
    - login: componente responsável pelo login na plataforma.
    - message-dialog: responsável pelos popups de erros presentes nas validações. Este componente é chamado pelos vários componentes do FrontOffice.
    - page-not-found: Em caso de a rota não conseguir ser concluída este componente é carregado.
    - vales: componente que permite ao doador resgatar os vales registados em troca de pontos.

- guards: responsável por verificar as permissões do utilizador.
- interceptor: responsável por editar o pedido, adicionando o token para futura validação.

- services: pasta com todos os services que acedem à API.

### *5.2.3 Extras*

Para agilizar e otimizar o ritmo de trabalho apresentado pelo grupo foram criados dois ficheiros (runner.bat e killer.bat) para iniciar e encerrar automaticamente quer o FrontOffice quer o BackOffice.

- runner.bat: executa o comando “npm install” e “npm start” em ambas as pastas: BackOffice e FrontOffice.
- killer.bat: termina todos os processos node a rodar na máquina.

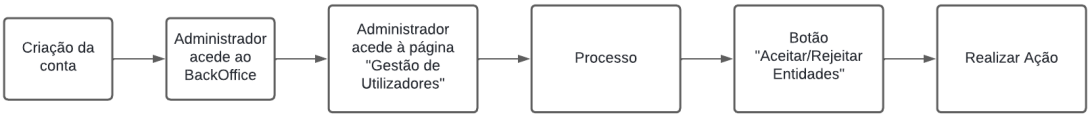
## 5.3 Flows e atividades

Em caso de dúvida esta secção destina-se a explicar a ordem de acontecimentos de eventos mais complexos e outras possíveis dúvidas.

### *5.3.1 Entidades*

Como foi mencionado ao longo deste relatório as entidades necessitam validação por parte dos administradores da Recicla Têxtil para estarem elegíveis a receber doações.

Tendo em conta que a criação da conta pode ser realizada quer pelo BackOffice (funcionário) quer pelo FrontOffice (próprio utilizador), estas são as etapas para a validação da entidade:




Passo a Passo após a criação da respetiva conta e login no BackOffice como administrador:


Após este processo, caso a entidade fique validada, esta passa a ter acesso a toda a plataforma, caso contrário esta fica guardada na base de dados, na eventualidade de mais tarde vir a ser aceite, mas não tem acesso a nenhuma parte do sistema.

### *5.3.2 Doações*

No nosso sistema as doações podem ter apenas um estado de cada vez, esse estado está limitado em: “Agendado”,” Por iniciar”,” Aprovado”,” A realizar”, “Validado” e “Cancelado”, ou seja:

- Agendado: O doador submeteu uma doação e quando a data que ele definiu for verificada a doação passa a “Por iniciar”, durante este tempo o doador pode cancelar a doação. (Nota: Todas as doações registadas através do BackOffice são agendadas para o próprio dia e são iniciadas como “Por iniciar” automaticamente).
- Por iniciar: A data de agendamento aconteceu e os administradores passam a ter acesso ao pedido, através do menu “Pedidos de Recolha” no BackOffice, sendo possível a aprovação do mesmo ou o seu cancelamento, bem como a correção de qualquer detalhe que esteja errado com a doação.
- Aprovado: Os funcionários da Recicla Têxtil aprovaram o pedido e este passa ao estado “A realizar”.
- A realizar: A entidade para onde foi efetuada a doação passa a conseguir validar o pedido recebido.
- Validado: A entidade validou a doação e o doador, e respetivo angariador se for o caso, recebem os pontos na sua conta.
- Cancelado: Em qualquer ponto do processo a doação foi rejeitada ou cancelada pelos intervenientes.

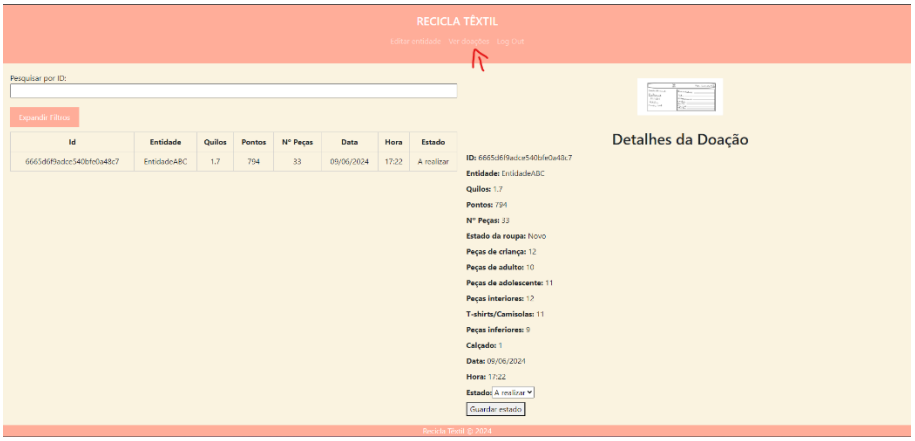
Nota: Para realizar a transição de “Agendado” para “Por iniciar” e de “Aprovado” para “A realizar” está configurado no ficheiro “app.js” um método que a cada minuto verifica o estado e a data de todos os pedidos presentes na base de dados possibilitando assim a sua alteração automática.

As datas atualizam quando ocorre uma alteração de estado na doação.

Pontos de acesso às doações realizadas:

- Pelos doadores:


- Pelas entidades:




- Pelos funcionários:

## 7. Conclusão

Por fim, o grupo reconhece que conseguiu um bom trabalho. Durante todo este processo trabalhamos em equipa e colaboramos de forma otimizada e organizada em prol de um projeto bem conseguido.
Possíveis melhorias futuras envolveriam: interface mais detalhada e “embelezada”, maior organização a nível de nomenclaturas e organização de ficheiros, implementação de dashboards mais completas e mais informativas.