Engenharia Informática - 2ºAno 2ºSemestre

Programação em Ambiente Web

Relatório 1a Milestone

Trabalho elaborado por:

João Santos LEI2T2 8220256
Sónia Oliveira LEI2T4 8220114
Ian Dinarte LEI2T4 8220005

Índice

1. Introdução

a. Contextualização do Projeto

Este projeto é desenvolvido no âmbito da unidade curricular de PAW - Programação em ambiente Web. O desenvolvimento deste trabalho visa melhorar o conhecimento de programação web, bem como ganhar experiência enquanto full stack devs.
Este relatório apenas cobre o trabalho realizado para a primeira entrega.

b. Contextualização do Domínio

A aplicação objetivo destina-se a ser usada como aplicação online para a ReciclaTêxtil, uma caridade que gere doações de roupa.
A empresa ReciclaTêxtil tem como objetivo recolher doações destinadas a diversas entidades de caridade que se podem registar no site. A plataforma será utilizada primariamente :
- pelos utilizadores que queiram realizar donativos, as quais em troca atribuem pontos que podem vir a ser trocados por coupons que podem ser usadas em várias lojas que se afiliam à ReciclaTêxtil;
- Staff e administradores que realizam a gestão das entidades e doações, e entidades beneficiadoras. Para isto, foi desenvolvido, nesta primeira milestone, um

BackOffice para os mesmos, e, para a segunda fase, será criado um FrontOffice para os restantes utilizadores.
Deste modo, a aplicação a ser desenvolvida deve dar suporte ao registo de todo o tipo de entidades que interagem com o site, registo de doações, e interação com a base de dados do negócio.

2. Funcionamento da Plataforma

Como introduzido na contextualização do domínio, a primeira entrega envolve o back office destinado a staff, e administradores, da empresa e da plataforma respetivamente.
As suas responsabilidades são como exposto em seguida:

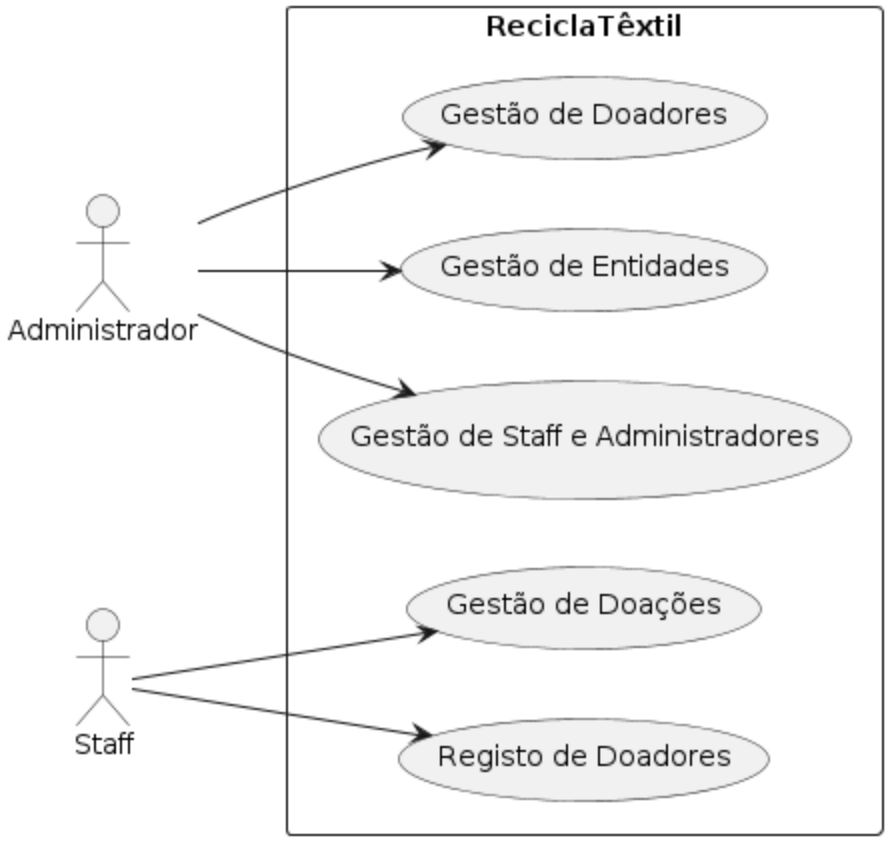


Figura 1 - Use case para BackOffice Users

Modelos usados

Todos os utilizadores trabalham sobre os mesmos modelos.
Esta decisão foi tomada por várias razões, pois era mais simples em termos de implementação, e por facilidade de escalar o projeto mais tarde, para a segunda milestone.
Porém a decisão tem desvantagens, como por exemplo os modelos tenderem a ser um pouco genéricos, e alguns modelos podem vir a ter informação que um utilizador usa, mas outro não. Tendo em conta esta desvantagem, o grupo decidiu continuar neste caminho pois é um primeiro projeto de webDev e concluímos que seria melhor ter um projeto funcional que nos proporcione uma melhor oportunidade de aprender.

```
const mongoose = require('mongoose')

const adminSchema = new mongoose.Schema({
    name:{
        type: String,
        required: true
    },
    email:{
        type: String,
        required: true,
    },
    password:{
        type: String,
        required: true
    },
    role:{
        type: String,
        required: true,
        default: 'admin',
    }
})

module.exports = mongoose.model('Admin', adminSchema)
```

Figura 2 - modelo de administrador

```
const mongoose = require('mongoose')

const couponSchema = new mongoose.Schema({
    sponsor:{
        type: String,
        required: true
    },
    value:{
        type: Number,
        required: true
    },
    cost:{
        type: Number,
        required: true
    },
})

module.exports = mongoose.model('Coupon', couponSchema)
```

Figura 3 - cupom

```
const donationSchema = new mongoose.Schema({
    donor: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'Donor'
    },
    institution: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'Partner'
    },
    location: {
        type: String,
        required: true,  },
    quantity: {
        type: Number,
        required: true, },
    quality: { type: String,
        enum: {
            values: ['Excellent', 'Good', 'Fair', 'Poor'], } },
    pointsAwarded: {       type: Number,
        required: true,   },
    date: { type: Date,
        required: true,
        default: Date.now()},
});
```

Figura 4 - doação

```
const donorSchema = new mongoose.Schema({
    name:{
        type: String,
        required: true
    },

    email:{
        type: String,
        required: true
    },

    password:{
        type: String,
        required: true
    },

    pointsEarned:{
        type: Number,
        default: 0,
        required:true
    },

    role:{
        type: String,
        required: true,
        default: 'donor',
    },

    coupons: [mongoose.Schema.Types.ObjectId]
})
```

Figura 5 - doadores

```
const rewardSchema = new mongoose.Schema({
    baseQuantity:{
        type: Number,
        required: true,
        default: 0},
    points:{
        type: Number,
        required: true,
        default: 0},
    poorPercentage:{
        type:Number,
        min:1,
        max: 100,
        default:100},
    fairPercentage:{
        type:Number,
        min:1,
        max: 100,
        default:100,},
    goodPercentage:{
        type:Number,
        min:1,
        max: 100,
        default:100,},
    excellentPercentage:{
        type:Number,
        min:1,
        max: 100,
        default:100,}
})
```

Figura 6 - recompensas

```
const partnerSchema = new mongoose.Schema({
    name:{
        type: String,
        required: true,},
    password:{
        type: String,
        required: true,},
    email:{
        type: String,
        required: true,
        },
    description:{
        type: String,
        required: true},
    dateOfRegistry:{
        type: Date,
        required: true,
        default: Date.now},
    numberOfDonations:{
        type: Number,
        required: true,
        default: 0 },
    imageName:{
        type: String}
})
```

Figura 7 - entidade

```
const staffSchema = new mongoose.Schema({
    name:{
        type: String,
        required: true
    },
    email:{
        type: String,
        required: true,

    },
    password:{
        type: String,
        required: true
    },
    role:{
        type: String,
        required: true,
        default: 'staff',
    }
})
```

Figura 8 - staff

Justificação dos modelos

Modelo de administrador - O administrador apenas precisa de informação sobre o seu nome, email, password e role, que é administrador por default para efeito de autorizações.

Modelo de staff - O staff apenas precisa de informação sobre o seu nome, email, password e role, que é administrador por default para efeito de autorizações.

Modelo de doadores - O doador apenas precisa de informação sobre o seu nome, email, password e role, que é doador por default para efeito de autorizações. Por fim tem também pontos ganhos para efeito da plataforma, visto que os ganha sempre que faz uma doação, além de um array com a finalidade de guardar os cupões que ele pode obter através dos pontos.

Modelo de entidade - Uma entidade tem nome, descrição, data de registo e imagem para efeitos de, mais tarde, criar uma página onde são apresentadas todas as entidades que trabalham em colaboração com a ReciclaTêxtil.
O email, password e número de doações são usados para efeito de dashboard e autenticação para a conta da entidade na plataforma.

Modelo de recompensa - A recompensa representa o modelo de como os pontos são dados através de cada doação. São dependentes da qualidade apontada na doação, que seria determinada no momento da recolha.

Modelo de coupon - representam o coupon, têm apenas o nome da entidade na qual pode ser usado, o valor e o custo do coupon em pontos. Provavelmente não funcionaria numa situação real mas para efeitos de representação são suficientes.

Modelo de doação - O modelo de doação visa guardar a pessoa que guardou, a entidade à qual a doação é destinada, o número de pontos dado, a data em qual foi realizada, a qualidade e quantidade da roupa doada.

3. Funcionalidades da aplicação

Para a primeira entrega do projeto, foram pedidos um conjunto de funcionalidades para o site da empresa Recicla Têxtil, sendo estas :

• Autenticação
• Gestão de administradores
• Gestão de Funcionários
• Gestão de Entidades
• Gestão de Doadores
• Gestão de Doações

Para além das mesmas, o grupo decidiu implementar as seguintes extra funcionalidades para suplementar o exercício :

• Dashboard de administrador com dados relativos ao número de doações, utilizadores e pontos de coleta;
• Gestão de como os pontos são atribuídos a cada doação dependendo de como a doação for validada pelos funcionários da empresa;
• Está ainda a ser criado um modelo adicional de pontos de coleta, que visa representar armazéns reais.

4. Detalhes da aplicação

Autenticação e Autorização

A autenticação é feita através da biblioteca Passport.js, que realiza o login, verificando se o utilizador existe no sistema, e, se sim, reencaminha-o para a página respetiva. O mecanismo de autorização é baseado em cookies e tokens, em que caso um utilizador seja autenticado, ser-lhe-à dado um token (de acordo com o role que desempenha na aplicação) que lhe dará acesso às respetivas funcionalidades. Sempre que um utilizador tenta aceder a uma página protegida, é verificada a presença do token, verificando se o role está de acordo com as permissões necessárias, e se sim, permite o seu acesso, caso contrário reverte o utilizador para a página de login.
A ação de logout remove o token de autenticação.

Atribuição de pontos

Cada doação dá ao doador um valor X de pontos dependendo de várias informações, a principal sendo o valor em kgs de roupa doada, e a percentagem de pontos que é determinada pela qualidade da roupa doada, sendo que donativos de qualidade excelente recebem 100% dos pontos. Apenas os administradores podem mudar informação relativa às definições de atribuição de pontos.
É assumido que quando uma entidade se regista na plataforma, a mesma aceita a distribuição de pontos definida pela ReciclaTêxtil.

Armazenamento de Imagens

Como o armazenamento de imagens diretamente na base de dados é desaconselhável devido ao espaço que estas ocupam, a abordagem escolhida foi, em vez disso, guardar os nomes dos ficheiros na base de dados, e quando necessário procurá-los dentro da pasta images/uploads. Para garantir que todos os nomes das imagens são únicos, os nomes atribuídos a cada ficheiro vêm do momento e data do upload.

Registo de Doações

As doações são a parte mais complexa do sistema, precisando de informações provenientes dos doadores, bem como das instituições de caridade. Para isso, no modelo de doações são incluídos campos que guardam o id dos objetos (doador e instituição) que são atribuídos no registo através de listas de seleção populadas

com os objetos das coleções respectivas. Deste modo, sempre que é reportada uma doação, são registados:

- O número de pontos a que corresponde a donativo;
- A instituição à qual é destinada o donativo;
- O doador que realizou o donativo;
- Atualizado o número de pontos do doador;
- Atualizado o número de donativos da instituição.

Rotas

Auth

Esta rota trata da autenticação/autorização de utilizadores. Quando é acedido o link /login, é apresentado o formulário, em que a submissão deste resulta num pedido **post** que envia os dados para o servidor onde são processados. Em caso de sucesso, o utilizador é encaminhado para a página principal que lhe diz respeito. Caso contrário, será avisado do motivo pelo qual não é possível realizar a autenticação.

Rotas de Administrador

Como o administrador tem acesso a várias funcionalidades, foram criadas várias rotas.
Gestão de Utilizadores:

- accountManagement.js
- deleteUsers.js
- editUsers.js
- registerUsers.js
- viewUsers.js

Gestão de Recompensas:

- rewardManagement.js

Esta foi a abordagem escolhida para evitar que os ficheiros se tornassem demasiado grandes e dificultasse a manutenção e leitura do código.
Assim, na rota **admin.js** são definidos os routers para gestão de utilizadores e recompensas, bem como os links (admin/manage-accounts e admin/manage-rewards, respetivamente).

A rota **accountManagement.js** contém routers definidos que levam às restantes rotas dependendo do pedido feito. Acedendo a admin/manage-accounts:

```
router.get('/', function (req, res, next) {
  res.render('admin/accountManagement/dashboard', { title: 'Account Management',  user: req.userName  });
});

const registerRouter = require('./registerUsers');
const viewRouter = require('./viewUsers');
const updateRouter = require('./editUsers')
const deleteRouter = require('./deleteUsers')


router.use('/register', registerRouter);
router.use('/view', viewRouter);
router.use('/edit', updateRouter)
router.use('/delete', deleteRouter)

module.exports = router;
```

Figura 9 - accountManagement.js

Depois, dentro de cada routers de gestão de utilizadores, existem rotas para a ação de CRUD correspondente, por exemplo, registo de cada tipo de utilizador (admin/manage-accounts/register), dentro do **registerRouter**, e esta forma de organização mantém-se para os restantes routers de gestão de contas.

Já o router de gestão de recompensas apenas necessita de uma rota, como são menos as funcionalidades e faz sentido encapsulá-las num só ficheiro. Portanto,

acedendo a admin/manage-rewards, temos as seguintes rotas:

```
router.get('/', function(req, res, next) {
    rewardController.home(req,res)
});

router.get('/points', function(req, res, next) {
    rewardController.editRewardsForm(req,res)
});


router.put('/points', async (req, res) => {
    rewardController.editRewards(req,res)
});


router.get('/coupons', function(req, res, next) {
    rewardController.createCouponsForm(req,res)
});

router.get('/coupons/view', async function(req, res, next) {
    rewardController.viewCoupons(req,res)
});

router.post('/coupons', async (req, res) => {
    rewardController.createCoupons(req,res)
})
```

Figura 10 - gestão de recompensas

As atividades de cada rota estão detalhadas nos controladores.

Rotas de Staff

O staff apenas tem como responsabilidade a gestão de donativos, com a possibilidade de registo de doadores (apesar de os poder registar, achámos melhor não permitir apagar ou editar perfis de doadores, por questões de segurança). Deste modo, sendo um modelo mais simples, apenas foi criado um ficheiro de rotas para o staff, que contém todas as rotas necessárias. Assim, acedendo a /staff:

```
const donationController = require ('../../controllers/staff/donations')
const userController = require ('../../controllers/staff/user')

router.get('/', function (req, res) {
    donationController.home(req,res)
});

router.get('/registerDonation', (req, res) => {
    donationController.newDonationForm(req,res)
});

router.get('/viewDonations', async (req, res) => {
    donationController.viewDonations(req,res)
}
)

router.post("/registerDonation", async (req, res) => {
    donationController.newDonation(req,res)
});

router.get("/registerDonators", async (req,res) =>{
    userController.registerDonorForm(req,res)
});

router.post("/registerDonators", async (req,res) =>{
    userController.registerDonor(req,res)
});

module.exports = router;
```

Figura 11 - gestão de doações

Estas são as rotas a que podemos aceder, e os seus comportamentos estão detalhados nos controladores.

Conclusão

Apesar de ser apenas a primeira metade do projeto, damos como bem sucedido o seu desenvolvimento. Sendo a primeira vez que qualquer um de nós faz algo relacionado com desenvolvimento web, conseguimos implementar todas as funcionalidades que foram pedidas, bem como alguns extras, embora simples. Gostávamos de ter feito mais, mas a falta de experiência obrigou-nos a focar no essencial, pois o tempo que tivemos não permitia grande ambição tendo em conta a complexidade dos tópicos abordados, e o conforto que temos com eles.
No entanto, este trabalho foi, e sem dúvida continuará a ser, uma forma de aprender os mecanismos de desenvolvimento web com uma abordagem hands-on, e achamos que conseguimos obter conhecimentos valiosos sobre a prática de programação em ambiente web, e esperamos conseguir compreender ainda mais, e estar ainda mais à vontade no final da próxima milestone.

Engenharia Informática - 2ºAno 2ºSemestre

Programação em Ambiente Web

Relatório 2a Milestone

Trabalho elaborado por:

João Santos LEI2T2 8220256
Sónia Oliveira LEI2T4 8220114

Introdução

a. Contextualização do Projeto

Passando à segunda parte do trabalho, realizado na âmbito da disciplina de PAW, tivemos como objetivo aprender a utilizar a framework Angular para desenvolvimento frontend. Esta parte do relatório incidirá sobre essa vertente, bem como faremos um breve resumo das alterações feitas ao projeto do milestone anterior, assim como quaisquer adições de conteúdo.

b. Contextualização do Domínio

A aplicação objetivo destina-se a ser usada como aplicação online para a ReciclaTêxtil, uma caridade que gere doações de roupa.
A empresa ReciclaTêxtil tem como objetivo recolher doações destinadas a diversas entidades de caridade que se podem registar no site. A plataforma será utilizada primariamente :
- pelos utilizadores que queiram realizar donativos, as quais em troca atribuem pontos que podem vir a ser trocados por coupons que podem ser usadas em várias lojas que se afiliam à ReciclaTêxtil;
- Staff e administradores que realizam a gestão das entidades e doações, e entidades beneficiadoras.

Para isto, foi desenvolvido, neste segundo milestone, um FrontOffice para doadores e entidades beneficiadoras.

Deste modo, o foco da aplicação será a realização de pedidos de doação, a consulta de doações, registo de utilizadores, e usufruto de pontos para efeito de compra de cupões.

2. Funcionamento da Plataforma

Como introduzido na contextualização do domínio, esta segunda entrega contempla o frontoffice de utilização destinada a entidades e doadores. Este diagrama ilustra as suas funções:

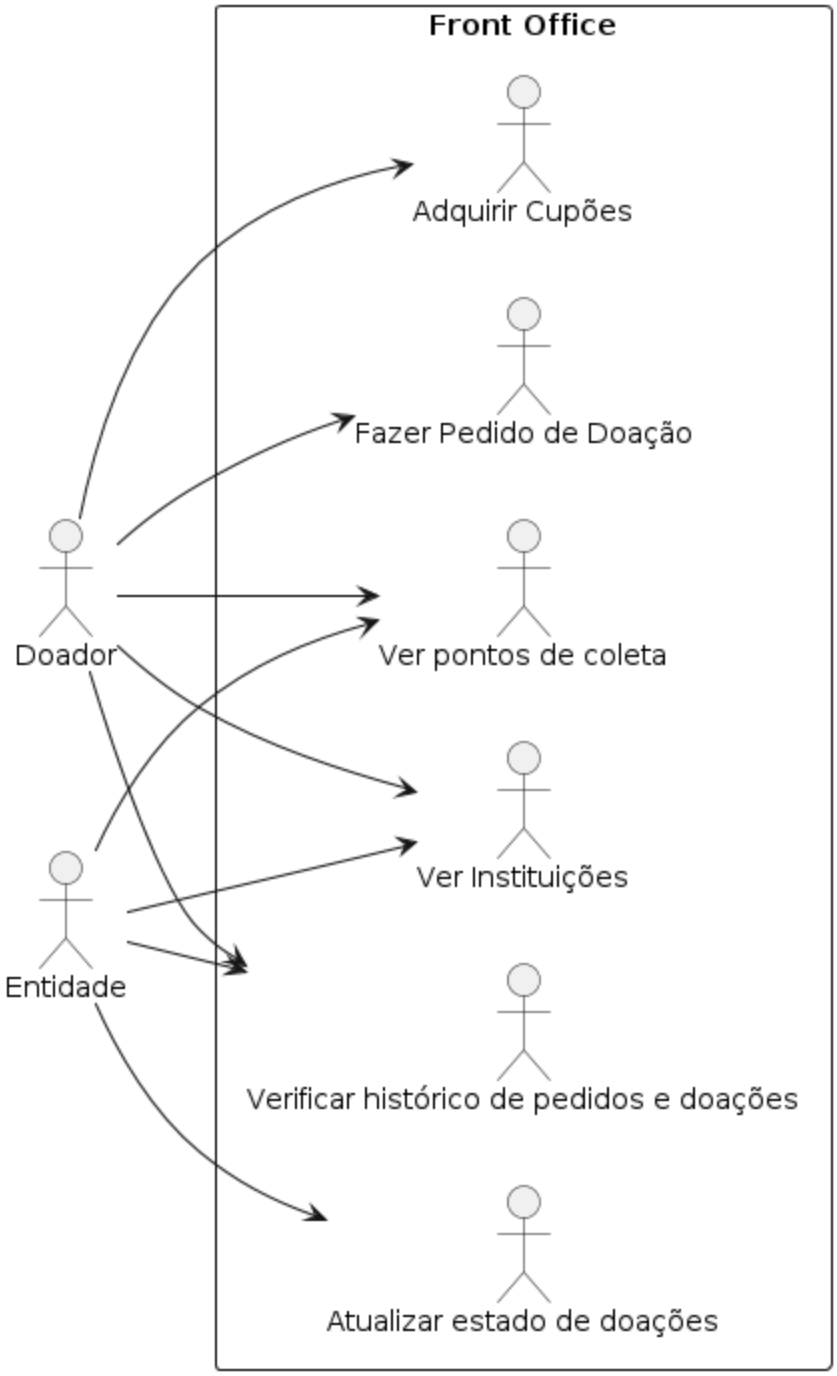


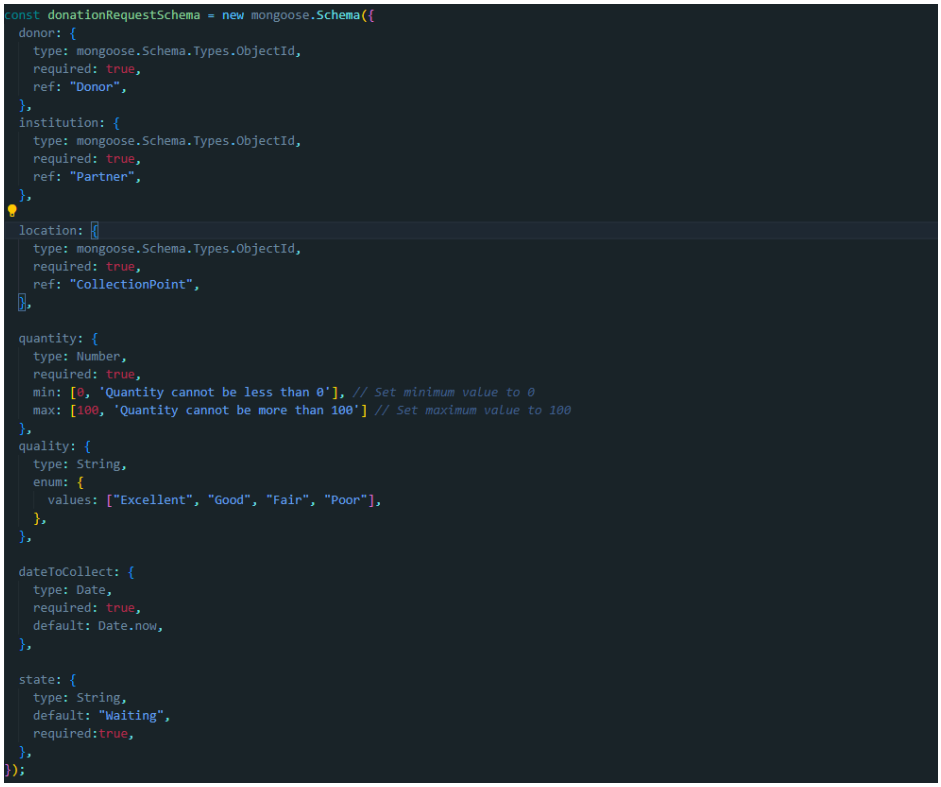
Figura 12 - Funcionamento do Front Office

Modelos e rotas usadas

Os modelos nesta milestone são essencialmente os mesmos, apenas com a adição de modelos para pedidos de doação, e pedidos de registo de entidades. Isto fez com

que fosse necessário alterar o projeto de backend, para que fosse possível fazer pedidos e estes ficassem guardados separadamente das doações e entidades em si.

Para isto, adicionamos estes modelos ao backend:



```
const donationRequestSchema = new mongoose.Schema({
  donor: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: "Donor",
  },
  institution: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: "Partner",
  },
  location: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: "CollectionPoint",
  },

  quantity: {
    type: Number,
    required: true,
    min: [0, 'Quantity cannot be less than 0'], // Set minimum value to 0
    max: [100, 'Quantity cannot be more than 100'] // Set maximum value to 100
  },
  quality: {
    type: String,
    enum: {
      values: ["Excellent", "Good", "Fair", "Poor"],
    },
  },

  dateToCollect: {
    type: Date,
    required: true,
    default: Date.now,
  },

  state: {
    type: String,
    default: "Waiting",
    required:true,
  },
});
```

Figura 12 - Pedido de Doação

```
onst partnershipSchema = new mongoose.Schema({
  name:{
      type: String,
      required: true,
  },

  email: {
      type: String,
      required: true,
      match: [/^\S+@\S+\.\S+$/, 'Please enter a valid email address']
  },

  password:{
      type: String,
      required: true,

  },

  description:{
      type: String,
      required: true
  },

  dateOfRegistry:{
      type: Date,
      required: true,
      default: Date.now
  },

  imageName:{
      type: String
  },

  accepted:{
      type: Boolean,
      required: true,
      default: false
  }
)
```

Figura 13 - Pedido de Registo de uma Entidade

Os restantes modelos permanecem iguais, sendo apenas traduzidos para TypeScript no projeto de angular.

Quanto a rotas, no backend foi criada uma nova pasta para armazenar os ficheiros de rotas de API, e outra para os controladores. A maior parte das funções são semelhantes às utilizadas no backend da entrega anterior, diferindo apenas no tipo de retorno, que passa a ser json.

Para exemplificar as rotas e controladores, voltemos aos pedidos de registo e doação. No caso dos pedidos de doação, na API existe uma função que cria um novo pedido, que ficará armazenado. Posteriormente, um membro de staff pode, através do backoffice aceitar este pedido.

```
donationController.newDonationRequest = async (req, res) => {

        const donation = new DonationRequest({
            donor: req.body.donor,
            quantity: req.body.quantity,
            dateToCollect: req.body.date,
            quality: req.body.quality,
            institution: req.body.institution,
            location: req.body.location,
        });

        try {
            await donation.save()
            res.json(donation)
        } catch (error) {
            console.log(error)
        }

}
```

Figura 14 - Função que será utilizada para realizar novos pedidos de doação

A função apresentada acima, é utilizada num serviço em angular:

```
makeDonationRequest(donationRequest:DonationRequest):Observable<DonationRequest>{
  return this.http.post<DonationRequest>(endpoint, donationRequest).pipe(
    catchError(this.handleError)
  );
}
```

Figura 15 - Método no serviço, que permite criar um novo pedido de doação

Em que o endpoint é o link da api, e donationRequest é um objeto do tipo pedido de doação fruto de um formulário. Aqui está a chamada deste método no componente dedicado à realização de um novo pedido:

```
    const donationRequest = new DonationRequest(
      { email: this.donor.email, _id: this.userId },
      this.institution,
      this.location,
      this.quantity,
      this.quality,
      this.dateToCollect
    );

    this.donService.makeDonationRequest(donationRequest).subscribe({
      next: (result: any) => {
        console.log(result);
        this.resetForm();
        this.router.navigate(['/donor/donation-requests']);
      },
      error: (error: any) => {
        this.errorMessage = error.error ? error.error : 'Error making donation request';
      }
    });
  }
```

Figura 16 - Secção do componente onde é criado o pedido, e feita a chamada do método makeDonationRequest

Isto fará com que seja enviado um pedido à API que resulta na execução da função demonstrada na Figura 14.

Após este processo, o pedido fica armazenado na base dados, e um membro do staff poderá aceitá-lo. Para isto, terá de aceder à listagem de todos os pedidos, selecionar aquele que deseja aceitar, e indicar a data em que foi recolhido, bem como entregar prova fotográfica.

```
router.get('/donationRequests', async(req,res) => {
   donationController.viewDonationRequests(req,res)
})//get list of requests

router.get('/donationRequests/:id', async(req,res) => {
    donationController.viewDonationRequest(req,res)
}) //get specific request

router.delete('/donationRequests/:id', async(req,res) => {
    donationController.deleteRequest(req,res)
}) //delete specific request

router.post('/acceptRequest/:id', async(req,res)=>{
    donationController.acceptRequest(req,res)
})
```

Figura 17 - Rotas de staff relacionadas com pedidos de doação

Ao aceitar um pedido, serão calculados os pontos a atribuir, e atualizadas as informações do doador, instituição, e local de coleta. Depois de aceite o pedido, este pode ser apagado (o que também pode ser feito se o pedido não for aceite).

```
donationController.acceptRequest = async (req, res) => {

  if (request) {
    upload(req, res, async function (err) {
      if (err instanceof multer.MulterError) {
        // A Multer error occurred when uploading.
        console.error(err);
        renderNewPage(req, res, request, 'staff/donationRequest', hasError = true)
      } else if (err) {
        // An unknown error occurred when uploading.
        console.error(err);
        renderNewPage(req, res, request, 'staff/donationRequest', hasError = true)
      }
    });

    let pointsAwardedDonation = await pointsAwardedRequest(request);
    const filename = req.file ? req.file.filename : null;
    const donation = new Donation({
      donor: request.donor,
      location: request.location,
      quantity: request.quantity,
      date: request.date,
      pointsAwarded: pointsAwardedDonation,
      quality: request.quality,
      institution: request.institution,
      imageName: filename,
      date: req.body.date,
      state: "Sent",
    });

    await Donor.findByIdAndUpdate(request.donor, {
      $inc: { pointsEarned: pointsAwardedDonation },
    });
    await CollectionPoint.findByIdAndUpdate(request.location, {
      $inc: { numberOfDonationsProcessed: 1 },
    });

    await Partner.findByIdAndUpdate(request.institution, {
      $inc:{numberOfDonations: 1}
    })

    try {
      await donation.save();
      await DonationRequest.findByIdAndUpdate(request.id, {state:'Sent'});
      res.redirect("/staff/viewDonations");
    } catch (error) {
      console.error(error);
      renderNewPage(req, res, request, 'staff/donationRequest', hasError = true)
    }
  }
}
};
```

Figura 18 - Função dedicada à aceitação de um pedido de doação, resultando numa nova doação efetiva.

Depois disto, tanto o doador como a entidade que recebeu o donativo poderão ver a doação persistida no histórico, sendo que a entidade poderá atualizar o estado, que será depois visível ao doador.

3. Funcionalidades da aplicação

Nesta milestone, além das funcionalidades pedidas (ilustradas no diagrama presente no capítulo anterior), o grupo implementou os seguintes extras:

- Um mapa em que será possível ver os pontos de doação
- A possibilidade de doações via paypal
- Alguns números associados a doadores e a entidades que serão apresentados nas respectivas dashboards
- Possibilidade de resetar a password caso o doador a perca

4. Detalhes da aplicação

Autenticação e Autorização

Foi utilizada uma estratégia semelhante àquela do login no backoffice, com a única diferença de que o pedido é realizado a partir do frontend angular. O token do utilizador é guardado em Local Storage, ao qual se acede quando necessário.

Para autorização, foram utilizados guardas, que permitem verificar o role do utilizador através do token, e caso este não corresponda ao esperado numa determinada rota, o utilizador não lhe poderá aceder.

Pedidos de Registo de entidades

Como pedido no enunciado, após o registo de uma entidade, esta não poderá realizar login na plataforma, até que o seu registo seja autorizado por um administrador. O funcionamento desta estratégia é semelhante ao explicado no capítulo 2 Funcionamento da Plataforma. Portanto, quando um pedido de registo é aceite, aí sim, uma entidade pode realizar login e aceder à sua dashboard.

Loja de Cupões

A loja de cupões implementada permite a um doador trocar os pontos que possui por cupões, tal como era pedido no enunciado. Os cupões serão armazenados no próprio utilizador como um array de chaves, que identificam os cupões pelo ID. Assim, sempre que um utilizador decide "comprar" um cupão, este estará associado a si. Esta informação é apresentada na página da loja de cupões, numa lista na qual o utilizador pode ver os cupões que possui, e aqueles que pode comprar.

Recuperação da Password

Caso um doador perca a sua password, poderá decidir repô-la. Para isto, dado o seu email, caso este exista na base de dados, será enviado um link para a página de reposição da password. O funcionamento desta *feature* inclui um sistema à base de tokens, em que o doador que efetuou o pedido irá receber um link que inclui um token gerado no backend, sem o qual é impossível aceder à reposição.

Mapa de pontos de recolha

Foi implementado um mapa, presente na página de pontos de recolha. Aqui, além do mapa, é apresentada uma listagem dos pontos de coleta, onde será possível selecionar um ponto e ver a sua localização no mapa, bem como o número de doações lá efetuadas.

Conclusão

Dado por concluída a segunda entrega do trabalho, achamos sem dúvida que conseguimos obter um melhor entendimento acerca de programação em ambiente web, mesmo que ainda haja imenso por onde melhorar. Sabemos que há muitas partes do programa que poderíamos ter melhorado, mas dado o tempo, experiência e recursos dos quais dispunhamos, o grupo acha que o resultado é bastante satisfatório.
Apesar de todas as dificuldades, conseguimos chegar a uma plataforma funcional, que possui ainda algumas bonificações, o que por si só, na nossa opinião merece mérito. Esperamos que as bases que desenvolvemos neste trabalho nos permitam continuar a melhorar e aprofundar o nosso conhecimento.

Credenciais de acesso à plataforma

Backoffice

- **Administrador: admin@gmail.com , password: admin**
- **Staff: staff@gmail.com, password: staff**

Frontoffice

- **Doador: donor@gmail.com , password: donor**
- **Entidade: charityemail@gmail.com, password: password**