



Universidade do Minho

Escola de Engenharia

Mestrado integrado em Engenharia Informática

Unidade Curricular de Processamento e Representação de Informação

O Meu Eu Digital

Bruno André Araújo Henriques a67664

Luís Carlos da Silva Marques a68691

Samuel Gonçalves Ferreira a76507

Janeiro, 2018

Data de Recepção	
Responsável	
Avaliação	
Observações	

O Meu Eu Digital

Bruno André Araújo Henriques a67664

Luís Carlos da Silva Marques a68691

Samuel Gonçalves Ferreira a76507

Janeiro, 2018

Resumo

O “Meu Eu Digital” consiste no desenvolvimento de uma aplicação Web, constituída por um frontend (interface pública, FE), backend (interface de administração e privada, BE), persistência de dados em base de dados, ficheiros e outros e lógica de controlo em JavaScript.

A aplicação irá suportar o “eu digital” do utilizador, ou seja, será uma espécie de diário digital. Diário digital quer dizer que as operações sobre os dados têm uma cronologia associada e que a linha temporal será o eixo principal da aplicação. Em termos de conteúdos, pretende-se o máximo de possibilidades como por exemplo: fotografias, registos desportivos, crónicas, pensamentos soltos, resultados académicos, participação em eventos, organização de eventos, opiniões, comentários sobre outros recursos Web, etc.

Área de Aplicação: Desenvolvimento Web;

Palavras Chave: eu digital, Eventos, Diário, Java-Script;

Índice

RESUMO	1
ÍNDICE	2
ÍNDICE DE FIGURAS	4
1. INTRODUÇÃO	6
1.1. AMBIENTE	6
1.2. ESTRUTURA DO RELATÓRIO	6
2. LISTA DE REQUISITOS	7
3. MODELO DE DADOS	9
3.1. ATIVIDADE DESPORTIVA	9
3.2. ÁLBUM FOTOGRÁFICO	10
3.3. CRÓNICA	10
3.4. ATIVIDADE CULTURAL	11
3.5. IDEIA	11
3.6. PENSAMENTO	12
3.7. RECEITA CULINÁRIA	12
3.8. TRABALHO ACADÉMICO	13
3.9. TRANSAÇÃO MONETÁRIA	13
3.10. VIAGEM	14
3.11. UTILIZADORES	14
3.12. TIPOS DE EVENTOS	15
4. ROUTES	16
4.1. ROUTE DE REGISTO DE UTILIZADOR	17
4.2. ROUTE DE LOGIN E LOGOUT	19
4.3. ROUTE FEED (PÚBLICO E PRIVADO)	20
4.4. ROUTE DE APRESENTAÇÃO DO PERFIL DO UTILIZADOR	22
4.5. ROUTE DE REGISTO DE ATIVIDADE (TODOS OS TIPOS)	23
4.6. ROUTE DE ELIMINAR PUBLICAÇÕES	25
4.7. ROUTE DE EDITAR PUBLICAÇÃO	26
4.8. ROUTE DE PESQUISA POR CATEGORIAS	28

4.9. ROUTE DE COMENTAR ATIVIDADES	29
5. FRONT END DA APLICAÇÃO	30
5.1. HOME PAGE	30
5.2. REGISTO	31
5.3. FEED “PRIVADO”	32
5.4. FEED “PÚBLICO”	33
5.5. CRIAÇÃO DE ATIVIDADES	33
5.6. PÁGINA DE PERFIL	34
5.7. EDIÇÃO DE ATIVIDADES	35
6. DIAGRAMA DA ARQUITETURA DA APLICAÇÃO	36
7. CONCLUSÕES E TRABALHO FUTURO	37

Índice de Figuras

Figura 1 Schema da Atividade Desportiva	9
Figura 2 Schema do álbum Fotográfico	10
Figura 3 Schema da Crónica	10
Figura 4 Schema da Atividade Cultural	11
Figura 5 Schema da Ideia	11
Figura 6 Schema do Pensamento	12
Figura 7 Schema da Receita Culinária	12
Figura 8 Schema do Trabalho Académico	13
Figura 9 Schema da Transação Monetária	13
Figura 10 Schema da viagem	14
Figura 11 Schema de utilizadores	14
Figura 12 Schema de Tipos de Eventos	15
Figura 13 Pedidos suportados pelo servidor da aplicação	16
Figura 14 Verificação de campos obrigatórios no registo	17
Figura 15 verificações e armazenamento do utilizador na BD	18
Figura 16 Funcionamento da Route de login	19
Figura 17 Funcionamento da route de Logout	19
Figura 18 Tipos de Eventos do Feed	20
Figura 19 ID do utilizador e cálculo da idade	20
Figura 20 Render do ficheiro feed.pug	21
Figura 21 Querie à BD para armazenar localmente a informação do User	22
Figura 22 Porção de código exemplificativo da edição do perfil de utilizador	23
Figura 23 Criação de uma Nova Atividade Desportiva	23
Figura 24 Verificação de fotografias e armazenamento da atividade na BD	24
Figura 25 Eliminar Publicação	25
Figura 26 Apagar fotos de atividades removidas	25
Figura 27 Disponibilização do formulário de edição	26
Figura 28 Remoção de fotos editadas	27
Figura 29 Pesquisa por categoria/hashtag de atividades	28
Figura 30 Comentário de atividades	29
Figura 31 Home page da aplicação	30
Figura 32 Página de Registo	31
Figura 33 Página do feed Privado	32
Figura 34 Página do Feed público	33
Figura 35 Página de Criação de Atividades	33
Figura 36 Página de Perfil	34
Figura 37 Página de edição de atividade	35

1. Introdução

1.1. Ambiente

O “Meu Eu Digital” enquadra-se na unidade curricular de Processamento e Representação de Informação pertencente ao perfil de Processamento de Linguagens e Conhecimento e, tem como objetivo a consolidação de conceitos lecionados ao longo do semestre. O seu principal foco é a utilização da Framework Node Js para a criação de uma aplicação Web capaz de registar vários eventos do seu utilizador. Os eventos podem ser públicos ou privados possibilitando, ou não, a visualização dos mesmos a outros utilizadores.

De um modo geral, a aplicação é composta por um ficheiro chamado app.js que contém o servidor que receberá os pedidos e os servirá. Para que isso acontecer foram criadas routes que contêm a logica de cada um dos pedidos suportados pelo servidor, estas fazem a ligação entre o modelo de dados e o frontend para apresentação de informação. No nosso caso, o “Meu Eu Digital” suporta dez tipos diferentes de eventos: álbum fotográfico, desportivo, cultural, culinário, crónicas, pensamentos, académicos, viagens, ideias e transações monetárias.

Tendo em conta os requisitos apresentados a equipa de desenvolvimento decidiu desenvolver uma espécie de uma rede social, dando possibilidade ao utilizador de tornar privada cada atividade, não tirando assim possibilidade de a socializar.

1.2. Estrutura do Relatório

- Introdução
- Requisitos da Aplicação
- Modelo de dados
- Routes
- Front End
- Diagrama da Arquitetura da Solução
- Conclusões e Trabalho Futuro

2. Lista de Requisitos

1. A aplicação é destinada a apenas um utilizador, ou seja, depois de instalada, o backend (BE) tratará apenas um “eu” individual. No entanto, o frontend (FE) de acesso à parte pública deverá estar disponível na Web para qualquer utilizador; Se a equipa de trabalho decidir preparar a aplicação para suportar simultaneamente múltiplos “eus” poderá fazê-lo o que será considerado como um extra;
2. Para se poder distinguir entre um acesso público e um acesso privado no FE deverá haver um módulo de autenticação transversal à aplicação. Quem estiver a usar JavaScript/NodeJS é aconselhado a usar o módulo passport e a implementar as autenticações: tradicional (username/password), Facebook e Google;
3. Independentemente, dos limites que as equipas de trabalho definirem, a aplicação terá de ter um formato/linguagem para importação e exportação de dados. Para esse efeito os equipas terão de definir uma DSL e implementar o respectivo processador;
4. A linguagem aconselhada para o desenvolvimento da aplicação é o JavaScript, no entanto, as equipas poderão decidir incluir módulos ou mesmo desenvolver a aplicação recorrendo a outra linguagem;
5. O domínio da aplicação deverá ser especificado e devidamente caracterizado antes do desenvolvimento ter início. Para o efeito devem desenhar uma ontologia do domínio do EU, a qual terá de ser especificada na linguagem OntoDL;
6. A aplicação terá como base o eixo temporal e a disposição cronológica de eventos e conteúdos será a sua base;
7. Cada elemento granular de conteúdo deverá ficar sempre classificado com um ou mais classificadores, de acordo com os conceitos da ontologia referida em 5;
8. Os classificadores do conteúdo deverão vir de um vocabulário controlado/taxonomia que terá de ser especificado pela equipa de trabalho (poderá e deverá ser discutido com o docente), conforme indicado no ponto 7;
9. A classificação de conteúdos deverá materializar-se em páginas de navegação com ordenação e agrupamento semântico que serão alternativas à navegação cronológica base;

10. Um diário digital vai ser uma lista de itens publicados. Um item pode ser constituído por vários componentes: bloco de texto, fotografia, ficheiro associado, etc.;
11. Numa fase inicial, a equipa deverá fazer um levantamento de todos os tipos de item que irá suportar e definir quais os campos de meta-informação que ficarão associados a cada item. Estes campos permitirão filtrar os itens para os mais variados fins: extrair o meu percurso desportivo, extrair o meu percurso académico, calcular a lista de locais turísticos que já visitei, etc. Exemplos de alguns tipos e sugestões de respetivos campos de meta-informação:
12. Um item pode ser público ou privado, por omissão deverá ser privado;
13. Se for privado apenas aparece no no frontend do seu criador, se for público deverá aparecer no frontend público da aplicação;
14. Um item público pode ser tornado privado e vice-versa;
15. Deve ser possível incluir comentários em cada item: serão notas que ficarão associadas ao item e que permitirão qualificar melhor o item quanto ao seu âmbito e conteúdo e mesmo à sua aplicação;
16. Pretende-se que haja uma ligação às redes sociais. Por exemplo, deverá haver uma operação para partilhar um item no Facebook ou no Twitter (eventualmente, com as devidas adaptações);

3. Modelo de Dados

Para a implementação do modelo de dados o grupo decidiu utilizar o Mongo DB, ou seja, uma base de dados não relacional. Para sustentar os dez tipos de eventos disponíveis a equipa decidiu implementar uma coleção para cada um deles. Assim para dez eventos existem dez coleções diferentes. Esta solução é eficiente quando tentamos fazer queries a uma única atividade de um utilizador, no entanto veio a revelar-se pouco eficiente quando queremos, por exemplo, saber todas as publicações de um utilizador, isto porque vamos ter de fazer 10 queries à base de dados. A outra solução seria guardar toda a informação numa só coleção o que diminuía o código e aumentava a eficiência, no entanto, aumentava a complexidade das queries. Mostra-se de seguida os schemas de cada uma das atividades suportadas pela aplicação. Para além das coleções das atividades foram criadas mais duas coleções: uma para armazenar os utilizadores e outra para armazenar os tipos de eventos disponíveis.

3.1. Atividade Desportiva

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema

var SchemaAtividadeDesportiva= new Schema({

  userId :Schema.Types.ObjectId,
  titulo: {type: String , required: true, max: 100},
  descricao: {type: String , required: true, max: 100},
  data: {type: Date , required: true},
  local: {type: String , required: false},
  privado: {type: Boolean , required: true},
  duracao: {type: String, required:true},
  desporto:{type: String , required: true},
  comentarios:[{type: String}],
  keys:[{type: String}],
  fotografia:{type: String , required: false}
})

module.exports = mongoose.model('atividadesdesportivas',SchemaAtividadeDesportiva)
```

Figura 1 Schema da Atividade Desportiva

3.2. Álbum Fotográfico

```
var SchemaAlbumFotografico= new Schema({
  userId :Schema.Types.ObjectId,
  titulo: {type: String , required: true, max: 100},
  descricao: {type: String , required: true, max: 100},
  data: {type: Date , required: true},
  local: {type: String , required: false},
  privado: {type: Boolean , required: true},
  comentarios:[{type: String}],
  keys:[{type: String}],
  fotografias:[
    {
      nome: {type: String},
      descricao: {type: String},
      pessoas: [{type:String}]
    }
  ]
})

module.exports = mongoose.model('albunsfotograficos',SchemaAlbumFotografico,'albunsfotograficos')
```

Figura 2 Schema do álbum Fotográfico

3.3. Crónica

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema

var SchemaCronica= new Schema({
  userId :Schema.Types.ObjectId,
  titulo: {type: String , required: true, max: 100},
  descricao: {type: String , required: true, max: 100},
  data: {type: Date , required: true},
  local: {type: String , required: false},
  privado: {type: Boolean , required: true},
  comentarios:[{type: String}],
  keys:[{type: String}],
  ficheiros:[{type: String}]
})

module.exports = mongoose.model('cronicas',SchemaCronica,'cronicas')
```

Figura 3 Schema da Crónica

3.4. Atividade Cultural

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema

var SchemaCultural= new Schema({
  userId :Schema.Types.ObjectId,
  titulo: {type: String , required: true, max: 100},
  descricao: {type: String , required: true, max: 100},
  data: {type: Date , required: true},
  local: {type: String , required: false},
  privado: {type: Boolean , required: true},
  tipo: {type: String, required:true},
  comentarios:[{type: String}],
  keys:[{type: String}],
  fotografias:[{type: String}]
})

module.exports = mongoose.model('culturais',SchemaCultural,'cultural')
```

Figura 4 Schema da Atividade Cultural

3.5. Ideia

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema

var SchemaIdeia= new Schema({
  userId :Schema.Types.ObjectId,
  titulo: {type: String , required: true, max: 100},
  descricao: {type: String , required: true, max: 100},
  data: {type: Date , required: true},
  local: {type: String , required: false},
  comentarios:[{type: String}],
  keys:[{type: String}],
  privado: {type: Boolean , required: true},
})

module.exports = mongoose.model('ideias',SchemaIdeia,'ideias')
```

Figura 5 Schema da Ideia

3.6. Pensamento

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema

var SchemaPensamento= new Schema({
  userId :Schema.Types.ObjectId,
  titulo: {type: String , required: true, max: 100},
  descricao: {type: String , required: true, max: 100},
  data: {type: Date , required: true},
  local: {type: String , required: false},
  comentarios:[{type: String}],
  keys:[{type: String}],
  privado: {type: Boolean , required: true},
})

module.exports = mongoose.model('pensamentos',SchemaPensamento)
```

Figura 6 Schema do Pensamento

3.7. Receita Culinária

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema

var SchemaReceitaCulinaria= new Schema({
  userId :Schema.Types.ObjectId,
  titulo: {type: String , required: true, max: 100},
  descricao: {type: String , required: true, max: 100},
  data: {type: Date , required: true},
  privado: {type: Boolean , required: true},
  comentarios:[{type: String}],
  keys:[{type: String}],
  ingredientes:[{type: String}]
})

module.exports = mongoose.model('receitasculinarias',SchemaReceitaCulinaria,'receitasculinarias')
```

Figura 7 Schema da Receita Culinária

3.8. Trabalho Académico

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema

var SchemaTrabalhoAcademico= new Schema({
  userId :Schema.Types.ObjectId,
  titulo: {type: String , required: true, max: 100},
  descricao: {type: String , required: true, max: 100},
  data: {type: Date , required: true},
  local: {type: String , required: false},
  privado: {type: Boolean , required: true},
  uc: {type: String, required:true},
  docente:{type: String},
  comentarios:[{type: String}],
  keys:[{type: String}],
  resultado:{type: String , required: true},
  ficheiros:[{type: String}]
})

module.exports = mongoose.model('trabalhosacademicos',SchemaTrabalhoAcademico,'trabalhosacademicos')
```

Figura 8 Schema do Trabalho Académico

3.9. Transação Monetária

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema

var SchemaTransacaoMonetaria= new Schema({
  userId :Schema.Types.ObjectId,
  titulo: {type: String , required: true, max: 100},
  descricao: {type: String , required: true, max: 100},
  data: {type: Date , required: true},
  privado: {type: Boolean , required: true},
  interveniente: {type: String, required:true},
  tipo: {type: String, required:true},
  comentarios:[{type: String}],
  keys:[{type: String}],
  montante:{type: String , required: true},
})

module.exports = mongoose.model('transacoesmonetarias',SchemaTransacaoMonetaria,'transacoesmonetarias')
```

Figura 9 Schema da Transação Monetária

3.10. Viagem

```
var mongoose = require("mongoose")
var Schema = mongoose.Schema

var SchemaViagem= new Schema({

  userId :Schema.Types.ObjectId,
  titulo: {type: String , required: true, max: 100},
  descricao: {type: String , required: true, max: 100},
  data:{type:Date, required:true},
  datainicio: {type: Date , required: true},
  datafim: {type: Date , required: true},
  local: {type: String , required: false},
  privado: {type: Boolean , required: true},
  comentarios:[{type: String}],
  keys:[{type: String}],
  acompanhantes:[{type: String}],
  fotografias:[{type: String}]
})

module.exports = mongoose.model('viagens',SchemaViagem)
```

Figura 10 Schema da viagem

3.11. Utilizadores

```
var SchemaUser= new Schema({
  username: {type: String , required: true, max: 100},
  password: {type: String , required: true, max: 100},
  dataNascimento: {type: Date , required: true},
  email: {type: String , required: true},
  pnome: {type: String , required: true},
  unome: {type: String , required:true},
  sexo: {type: String , required:true},
  foto: {type: String , required: false}
})

module.exports = mongoose.model('users',SchemaUser)
```

Figura 11 Schema de utilizadores

3.12. Tipos de Eventos

```
var SchemaTiposEventos= new Schema({
  nome: {type: String , required: true, max: 100}
})

module.exports = mongoose.model('tiposEventos',SchemaTiposEventos,'tiposEventos')
```

Figura 12 Schema de Tipos de Eventos

4. Routes

As routes definem as ações permitidas pela aplicação. Nestes ficheiros estão incluídos o acesso à camada de dados para ir buscar a informação de modo a apresenta-las ao utilizador através do render das páginas de FrontEnd. Estamos então perante uma camada de comunicação entre o front end e a camada de dados. Cada uma destas routes são chamadas quando o servidor recebe pedidos, quando o pedido não é reconhecido o servidor devolve erro 404(not found).

```
app.use('/', index)
app.use('/feed', feed)
app.use('/registarAct', registar)
app.use('/users', users)
app.use('/signup', signup)
app.use('/login', login)
app.use('/logout', logout)
app.use('/editarTraAcad', editarTraAcad)
app.use('/editarCronica', editarCronica)
app.use('/registarPensamento', pensamento)
app.use('/registarCronica', cronica)
app.use('/registarTransacao', transacao)
app.use('/registarCultural', cultural)
app.use('/registarTrabalho', trabalho)
app.use('/pesquisarCat', pesquisa)
app.use('/pesquisarCatPub', pesquisaPub)
app.use('/perfil', perfil)
app.use('/editarTransMon', editarTransMon)
app.use('/editarRecCul', editarRecCul)
app.use('/editarPensamento', editarPensaento)
app.use('/registarEve', evento)
app.use('/comentario', comentario)
app.use('/editarCultural', editarCultural)
app.use('/editarIdeia', editarIdeia)
app.use('/editarActDesp', editarActDesp)
app.use('/editarEvento', editarEvento)
app.use('/registarIdeia', ideia)
app.use('/registarViagem', viagem)
app.use('/editarViagem', editarViagem)
app.use('/registarReceita', receita)
app.use('/registarAlbum', album)
app.use('/eliminar', eliminar)
app.use('/editar', editar)
app.use('/feedPublico', feedPublico)
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});
```

Figura 13 Pedidos suportados pelo servidor da aplicação

Foram então implementadas as seguintes ações :

- ✓ Registo de utilizador
- ✓ Login e Logout
- ✓ Apresentação do feed(público e privado)
- ✓ Apresentação do perfil do utilizador
- ✓ Registo de Atividades (todos os tipos)
- ✓ Eliminar Publicações (todos os tipos)
- ✓ Editar Publicações (todos os tipos)
- ✓ Pesquisa por Categorias
- ✓ Comentar Atividades (todos os Tipos)

De modo a não haver repetições apenas se vão mostrar pedaços de código executado para que as funcionalidades acima descritas sejam possíveis.

4.1. Route de Registo de utilizador

No registo do utilizador começa-se por verificar se os campos obrigatórios do formulário estão preenchidos (username*, password*, data de nascimento*, primeiro nome*, ultimo nome*, sexo*, fotografia e email):

```
form.parse(req,function(err,fields,files){
  // verificar se todos os campos obrigatorios foram preenchidos
  var requiredFields = Object.keys(user.schema.tree).filter(x => user.schema.tree[x].required ===true)
  var emptyField = false
  console.log(fields)
  var data_partida=fields.dataNascimento.split('-');
  var dataAux=new Date(data_partida[0], data_partida[1]-1, data_partida[2]);
  fields.dataNascimento=dataAux

  for(f in requiredFields){
    if(fields[requiredFields[f]] === undefined)
      emptyField=true
  }
  if(emptyField){
    status="Por favor preencha todos os campos obrigatórios"
    res.render('registarUser',{ title: 'Registar utilizador' , status: status });
  }
}
```

Figura 14 Verificação de campos obrigatórios no registo

A seguir verifica-se se já existe um utilizador com o mesmo username na BD e se a confirmação da password não falha. Se essas verificações não produzirem nenhum erro então procede-se à criação de um novo utilizador na BD.

```

else{
  //verificar se já existe algum utilizador com o username inserido no formulário
  user.find({'username': fields.username},function(err2,docs){
    if(!err2){
      if(docs.length === 0){
        if(fields.password!=fields.password2){
          status="Erro na confirmação de password"
          res.render('registarUser',{ title: 'Registar utilizador' , status: status});
        }
        else{
          var newUser = new user()
          newUser.username = fields.username
          newUser.password = fields.password
          newUser.dataNascimento = fields.dataNascimento
          newUser.email = fields.email
          newUser.pnome = fields.pnome
          newUser.unome = fields.unome
          newUser.sexo = fields.sexo
          if(files.fotografia.name!=""){
            var fenviado=files.fotografia.path
            var extension = files.fotografia.name.split(".")
            extension = extension[extension.length-1]
            var novoNome = 'fp-'+fields.username + "." + extension
            newUser.foto = novoNome
            var fnovo='./public/images/upload/'+novoNome
            fs.rename(fenviado,fnovo,function(err3){
              if(!err3){
                status="Ficheiro Recebido e guardado com sucesso"
                console.log(status);
              }
              else{
                status="Ocorreram erros na gravação do ficheiro enviado";
              }
            })
          }
          newUser.save(function(err3, doc){
            if(!err3){
              status="Registo Efetuado com sucesso."
              // MANDAR STATUS COMO PARAMETRO
              res.redirect("/login")
            }
            else{
              console.log("Erro ao efetuar o registo:\r\n" + err + "\r\n\r\n");
              status+="Ocorreu um erro: "+err3
              res.render('registarUser',{ title: 'Registar utilizador' , status: status });
            }
          });
        }
      }
    }
  });
}

```

Figura 15 verificações e armazenamento do utilizador na BD

4.2. Route de Login e Logout

Na route de login é feita a verificação básica dos campos username e password. É importante referir a presença de cookies de utilizador, se o cookie estiver guardado na cache do browser o utilizador quando acede à plataforma é automaticamente redirecionado para o feed:

```
router.post('/',function(req,res,next) {
  var form= new formidable.IncomingForm();
  var status=""
  form.parse(req,function(err,fields,files){
    user.find({'username':fields.username},function(err2,docs){
      if(!err2){
        if(docs.length>0){
          // nao é suposto haver mais do que um resultado
          if(docs[0].password === fields.password){
            res.cookie('online', fields.username)
            res.redirect('/feed')
          }
          else{
            res.render('login', { title: 'Login' , status: 'Password errada para o utilizador inserido'});
          }
        }
        else{
          res.render('login', { title: 'Login' , status: 'Não existe nenhum utilizador com esse nome'});
        }
      }
      else{
        console.log("Ocorreu um erro ao fazer o login do user "+fields.username + " : \r\n"+err+"\r\n\r\n")
        res.render('login', { title: 'Login' , status: 'Ocorreu um erro, por favor tente novamente'});
      }
    })
  })
})
```

Figura 16 Funcionamento da Route de login

Na **route de logout** apenas é feito um redireccionamento para a página de login.

```
var express = require('express');
var router = express.Router();
var cookieParser = require("cookie-parser")

/* GET home page. */
router.get('/', function(req, res, next) {
  res.clearCookie("online")
  res.redirect('/login')
});

module.exports = router;
```

Figura 17 Funcionamento da route de Logout

4.3. Route Feed (Público e Privado)

Na route do Feed público começa-se por saber qual é o nome do utilizador (através dos cookies) para depois se aceder à BD para se ir buscar todos os tipos de eventos armazenados:

```
sync.fiber(function(){
  var currentUser = req.cookies.online;

  // Ir à base de dados buscar os tipos de eventos disponíveis
  var tipos=[]
  var OpTipos=[]
  try{
    var docsTipos = sync.await(TiposEventos.find().exec(sync.defer()))
    for(var i=0;i<docsTipos.length;i++){
      tipos.push(docsTipos[i].nome);
      OpTipos.push(docsTipos[i].nome)
    }
  }
}
```

Figura 18 Tipos de Eventos do Feed

Descoberto o username do utilizador vai-se à base de dados para se poder armazenar na variável userID o id de utilizador do utilizador. Para além disso, é também calculada a idade do utilizador através da data de nascimento incluída no registo.

```
// Ir à base de dados buscar o ID do utilizador que fez o pedido
var userID
var userDoc
try{
  var docsUsers = sync.await(User.find({'username': currentUser}).exec(sync.defer()))
  if(docsUsers.length>0){
    userDoc = docsUsers[0]
    userID = userDoc._id
    //Calculo da Idade//
    var ageDiffMs = Date.now() - userDoc.dataNascimento.getTime();
    var ageDate = new Date(ageDiffMs); // milliseconds from epoch
    userDoc.idade=Math.abs(ageDate.getUTCFullYear() - 1970);
  }
  else{
    console.log("Tentativa de acesso ao feed de um utilizador que não existe na base de dados");
    res.clearCookie("online")
    res.render('error', { message:'Ocorreu um erro ao carregar a página de feed, por favor tente novamente'});
  }
}
```

Figura 19 ID do utilizador e cálculo da idade

A partir daqui é feito o rastreamento de todos os eventos na BD que têm o Id do utilizador em causa associado. Todos os eventos vão estar guardados num array chamado eventos no qual é acrescentado a cada objeto vindo da base de dados, o respetivo tipo (para efeitos de apresentação ao utilizador). Este processo é repetido para todos os outros eventos suportados pela aplicação. Todo este processo ocorre de modo síncrono através do uso do módulo synchronize. Quando este módulo não estava a ser utilizado a mesma informação no feed aparecia de modo diferente devido ao assincronismo do código.

Por fim e, quando já estamos na posse de toda a informação necessária, e existem eventos, é feita a ordenação temporal dos mesmos e o ficheiro PUG do feed é renderizado.

```
if(eventos.length!=0){
  eventos = eventos.sort(function(a,b) {return (a.data > b.data) ? -1 : ((b.data > a.data) ? 1 : 0);} )
  res.render('feed',{
    user: {
      username: currentUser ,
      foto: userDoc.foto,
      idade: userDoc.idade,
      email:userDoc.email,
      pnome:userDoc.pnome,
      unome:userDoc.unome
    },
    tipos:tipos,
    eventos:eventos,
    OpTipos:OpTipos
  })
}
```

Figura 20 Render do ficheiro feed.pug

No caso de não existirem eventos apenas é acrescentado um status na renderização do ficheiro pug, a informar o utilizador que não foram encontrados eventos.

No caso do Feed publico é feito o mesmo processo, só que, para todos os utilizadores presentes na base de dados e para todos os eventos públicos.

4.4. Route de apresentação do Perfil do Utilizador

Na route de perfil do utilizador começa-se por descobrir qual o respetivo username através dos cookies do pedido. Depois de se saber o username é feita uma query à BD para se ir buscar toda a informação disponível sobre o utilizador em questão e apresentá-la. Este procedimento é demonstrado na figura em baixo.

```
router.get('/', function(req, res, next) {
  var currentUser = req.cookies.online;
  User.find({'username': currentUser}).exec((err1, docs1) => {
    if(!err1){
      if(docs1.length > 0){
        var userDoc = docs1[0]
        //Calculo da Idade//
        var ageDifMs = Date.now() - userDoc.dataNascimento.getTime();
        var ageDate = new Date(ageDifMs); // milliseconds from epoch
        userDoc.idade = Math.abs(ageDate.getUTCFullYear() - 1970);
        var userID = userDoc._id
        res.render('perfil', {
          user: {
            username: currentUser,
            foto: userDoc.foto,
            idade: userDoc.idade,
            email: userDoc.email,
            pnome: userDoc.pnome,
            unome: userDoc.unome,
            password: userDoc.password
          }
        })
      }
    }
    else{
      console.log("Utilizador não existe")
    }
  })
})
```

Figura 21 Query à BD para armazenar localmente a informação do User

Na apresentação do perfil do utilizador decidiu-se fazer uma fusão com a edição do mesmo, ou seja, esta área tanto serve para o utilizador consultar a sua informação pessoal como também a pode editar. Assim, o utilizador se quiser alterar a informação pessoal vai ter de submeter um formulário que será recebido e, através dos seus campos vai ser feito um update à BD dos campos em questão. Apenas são permitidas alterações no email, password e na fotografia.


```

    User.update(
      {'username':fields.username},
      {$set:{'email': fields.email,
        'password': fields.password}}).exec(function(err,docs){
        if(!err){
          console.log("Perfil alterado com sucesso")
        }
        else{
          console.log("Ocorreu um erro tentar editar o perfil")
        }
      })

    res.redirect('/feed')

```

Figura 22 Porção de código exemplificativo da edição do perfil de utilizador

4.5. Route de Registo de Atividade (Todos os Tipos)

Tal como nos outros casos, no registo de uma atividade começa-se por saber a informação do utilizador(username) através dos cookies e de seguida fazer uma query à BD utilizando o username como factor de procura. Na callback da query à BD é criada uma atividade de um determinado tipo com os campos do formulário que é recebido. A figura em baixo mostra esse processo para o caso da atividade desportiva.

```

var currentUser = req.cookies.online;
User.find({'username':req.cookies.online }).exec((err1,docs)=>{
  if(!err1){
    userID = docs[0]._id
    var ativDesp = new Ad({
      userId : userID,
      titulo:fields.titulo,
      descricao: fields.descricao,
      data: new Date(),
      local: fields.local,
      keys: hashtags,
      privado: fields.privado,
      duracao: fields.duracao,
      desporto: fields.desporto,
    })
  }
})

```

Figura 23 Criação de uma Nova Atividade Desportiva

Criada a atividade, esta é guardada, no entanto, antes disso tem de se verificar se existem fotografias associadas. Para isso, é verificado se o campo files do formulário vem vazio ou não. Feita essa verificação se existirem fotos é feito um rename da fotografia colocando o id da atividade, data e um contador (para quando existem mais que uma). O path das fotografias é adicionado à atividade e por fim esta é guardada na BD.

```
ativDesp.save(function(err1, doc){
  if(!err1){
    if(files.fotografia.name != ""){
      var extension = files.fotografia.name.split(".")
      extension = extension[extension.length-1]
      var data = new Date()
      var novadata = data.toISOString().split(':').join('-')
      novadata = novadata.split('.').join('-')
      ativDesp.fotografia = doc._id + "-" + novadata + "." + extension
      fs.rename(files.fotografia.path, './public/images/upload/' + ativDesp.fotografia, function(err1){
        if(!err1){
          ativDesp.save(function(err2, doc2){
            if(!err2){
              res.redirect('/feed')
            }
            else{
              console.log("Erro ao adicionar fotografia da atividade desportiva à base de dados:\r\n" + err2 + "\r\n\r\n");
              status="Ocorreu um erro, por favor tente novamente mais tarde."
              res.render('registarAct', {'status' : status , 'desportos' : listaDesportos })
            }
          })
        }
        console.log("Ficheiro recebido e guardado com sucesso")
      }
      else{
        console.log("Ocorreram erros na gravação do ficheiro enviado")
      }
    }
  })
})
```

Figura 24 Verificação de fotografias e armazenamento da atividade na BD

Este processo é repetido para todas as outras atividades suportadas pela aplicação.

4.6. Route de Eliminar Publicações

Para a eliminação de publicações é apenas necessário saber o id da publicação na BD. Quando o utilizador escolhe a opção de eliminar, é enviado um campo “tipo” no pedido. Dessa forma a aplicação sabe qual é o tipo de publicação que se quer apagar para assim poder fazer a querie à BD na respetiva coleção, tal como se demonstra na figura em baixo. (A opção de eliminar só aparece disponível para eventos criados pelo próprio utilizador)

```
if(tipo === "Cultural"){
  Cultural.find({'_id':req.params.id},(function(err,docs) {
    if (!err) {
      var fotos = docs[0].fotografias
      for(foto in fotos){
        try{
          curPath = images_dir + fotos[foto]
          fs.unlinkSync(curPath);
        } catch (err) {console.log("ERRO AO REMOVER A IMAGEM:" + err)}
      }
    }
    else {
      var status = "Ocorreu um erro ao remover o evento!"
      res.redirect('/feed')
    }
  })
)
```

Figura 25 Eliminar Publicação

No caso de a publicação ter fotografias associadas é necessário apaga-las da diretoria de fotografias. As fotografias das atividades são guardadas com id da atividade como nome do ficheiro. Assim basta aceder à diretoria e apaga-las.

```
for(foto in fotos){
  try{
    curPath = images_dir + fotos[foto]
    fs.unlinkSync(curPath);
  } catch (err) {console.log("ERRO AO REMOVER A IMAGEM:" + err)}
}
```

Figura 26 Apagar fotos de atividades removidas

Este processo repete-se para qualquer tipo de evento.

4.7. Route de Editar Publicação

Na route de edição de publicação existem dois passos a serem executados:

- O primeiro passo é feito pela route geral de editar que, basicamente descobre qual o tipo da atividade a editar e faz render do ficheiro pug para edição

```
router.get('/:tipo/:id', function(req, res, next) {
  var tipo = req.params.tipo
  console.log("tipo: "+tipo + "; id: " + req.params.id)
  if(tipo == "Cultural"){
    Cultural.find({'_id':req.params.id},(function(err,docs){
      if(!err){
        res.render('editarCultural', {valores:docs[0]})
      }
      else{
        var status = "Ocorreu um erro ao tentar editar o evento!"
        res.redirect('/feed')
      }
    })
  )
})
```

Figura 27 Disponibilização do formulário de edição

- O segundo passo é executado pelas routes de edição específica de cada atividade. No fim do utilizador submeter as alterações que quer fazer, essa route específica é que trata da atualização na BD. Este último processo exige mais código na medida em que se tem de fazer verificações em relação às fotos pois podem ter que ser removidas para dar lugar às novas fotos incluídas pelo utilizador. Nesse sentido, tem de ser feito um ciclo pelo campo files do pedido, renomear os ficheiros de acordo com a nossa convenção e guardá-los.

```

var i = 0
for(x in files){
  i++
  if(x.startsWith("foto")){
    console.log("nome da foto: " + files[x].name)
    console.log("caminho da foto: " + files[x].path)
    if(files[x].name != ""){
      var extension = files[x].name.split(".")
      extension = extension[extension.length-1]
      console.log("valor = " + i)
      var data = new Date()
      var novadata = data.toISOString().split(':').join('-')
      novadata = novadata.split('.').join('-')
      var novoNome = fields._id + "-" + novadata + "-" + i + "." + extension
      var fenviado=files[x].path
      var fnovo=images_dir+novoNome
      fs.rename(files[x].path, fnovo, function(err3){
        if(!err3){
          console.log("Ficheiro recebido e guardado com sucesso")
        }
        else{
          console.log("Ocorreram erros na gravação do ficheiro enviado")
        }
      })
      Cultural.update({'_id': fields._id}, {$push: {fotografias: novoNome}}).exec(function(err,docs){
        if(!err){
          console.log("Evento Cultural alterado com sucesso")
        }
        else{
          console.log("Ocorreu um erro tentar editar evento cultural")
        }
      })
    }
  }
}
}

```

Figura 28 Remoção de fotos editadas

4.8. Route de Pesquisa por Categorias

Na pesquisa por categorias começa-se por saber qual o utilizador em questão (do mesmo modo descrito nas routes anteriores) para mais tarde na apresentação do feed termos acesso ao utilizador em questão. No formulário de pesquisa existe uma dropdown e uma caixa de texto para colocar uma hashtag, se for o caso (ver na secção do front end). Assim através da análise dos campos do pedido consegue-se saber o tipo de evento e/ou a hashtag que queremos filtrar. Depois de termos acesso a todas estas variáveis são feitas queries à base de dados e posterior render do feed com a informação filtrada por categoria e/ou hashtag.

```
if(categoria === "Cultural" || key!=""){
  try{
    docs2 = sync.await(Cultural.find({'userId':userID }).sort({data:-1}).exec(sync.defer()))
    if(docs2.length> 0){
      for(var i=0;i<docs2.length;i++){
        var x=JSON.parse(JSON.stringify(docs2[i]));
        x.tipoEvento="Cultural"

        if(categoria === "Cultural" && key === ""){
          eventos.push(x)
        }
        else if(key === ""){
          eventos.push(x)
        }
        else{
          opcoes = x.keys.filter( y => y === key)
          if(opcoes.length>0 ) {
            eventos.push(x)
          }
        }
      }
    }
  }
  else{
    status="Não foram encontrados resultados"
  }
  if(key === ""){
    eventos = eventos.sort(function(a,b) {return (a.data > b.data) ? -1 : ((b.data > a.data) ? 1 : 0);} )
    res.render('feed',{
      user: {
        username: currentUser ,
        foto: userDoc.foto,
        idade: userDoc.idade,
        email:userDoc.email,
        pnome:userDoc.pnome,
        unome:userDoc.unome
      },
      status:status,
      eventos:eventos,
      OpTipos:OpTipos
    })
    return
  }
}
```

Figura 29 Pesquisa por categoria/hashtag de atividades

Como se pode verificar pelo código apresentado na imagem todo o código de pesquisa de atividades por categoria corre de modo síncrono para evitar aleatoriedade na apresentação de resultados no feed. No caso de haver uma pesquisa no feed público a pesquisa ocorre da mesma forma, mas é restringida também pelo campo privado (só eventos públicos aparecem).

4.9. Route de Comentar Atividades

A route de comentários simplesmente faz update às atividades de modo a adicionar o comentário à coleção na BD e redireciona novamente para o feed. O conteúdo do comentário e o id da publicação é retirado dos campos do request.

```
form.parse(req,function(err,fields,files){
  if(!err){
    var tipo = fields.tipo
    var id = fields._id
    fields.comentario = req.cookies.online + " : " + fields.comentario

    if(tipo == "Cultural"){
      Cultural.update({'_id': id},
        {$push: {'comentarios': fields.comentario}}).exec(function(err,docs){
        if(!err){
          var status = "Comentario adicionado com sucesso!"
          res.redirect('/feed')
        }
        else{
          var status = "Ocorreu um erro ao tentar comentar!"
          res.redirect('/feed')
        }
      })
    }
  })
})
```

Figura 30 Comentário de atividades

O mesmo acontece para qualquer atividade suportada pela aplicação.

5. Front End da Aplicação

Nesta secção serão apresentados os resultados produzidos pelos ficheiros das views criadas pela equipa de desenvolvimento. Foram criadas as seguintes views:

- ✓ Home page/login
- ✓ Registo
- ✓ Feed público e privado
- ✓ Formulários das várias atividades
- ✓ Perfil do Utilizador
- ✓ Edição das várias atividades

5.1. Home Page

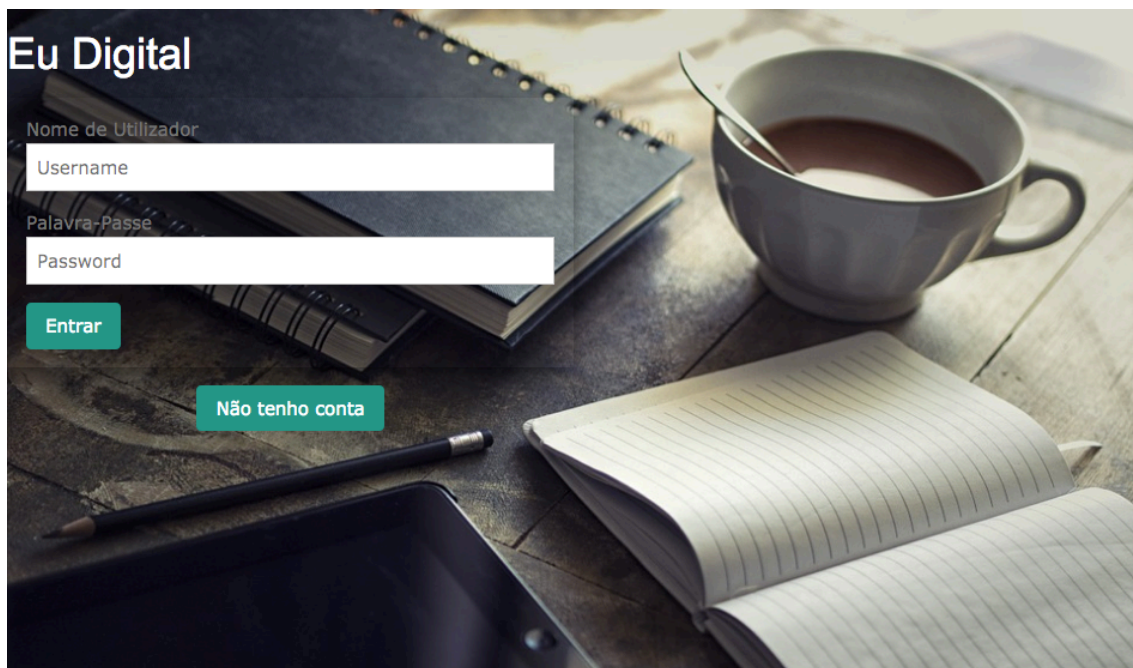


Figura 31 Home page da aplicação

5.2. Registo



Registo

*Nome de Utilizador
Username

*Palavra-Passe
Password

*Confirmar Palavra-Passe
Password

Data de Nascimento
mm / dd / yyyy

*Email
Email

*Primeiro Nome
Primeiro Nome

*Ultimo Nome
Ultimo Nome

*Sexo ☐ Masculino ☐ Feminino

Fotografia
Browse... No file selected.

registar

Figura 32 Página de Registo

5.3. Feed “Privado”

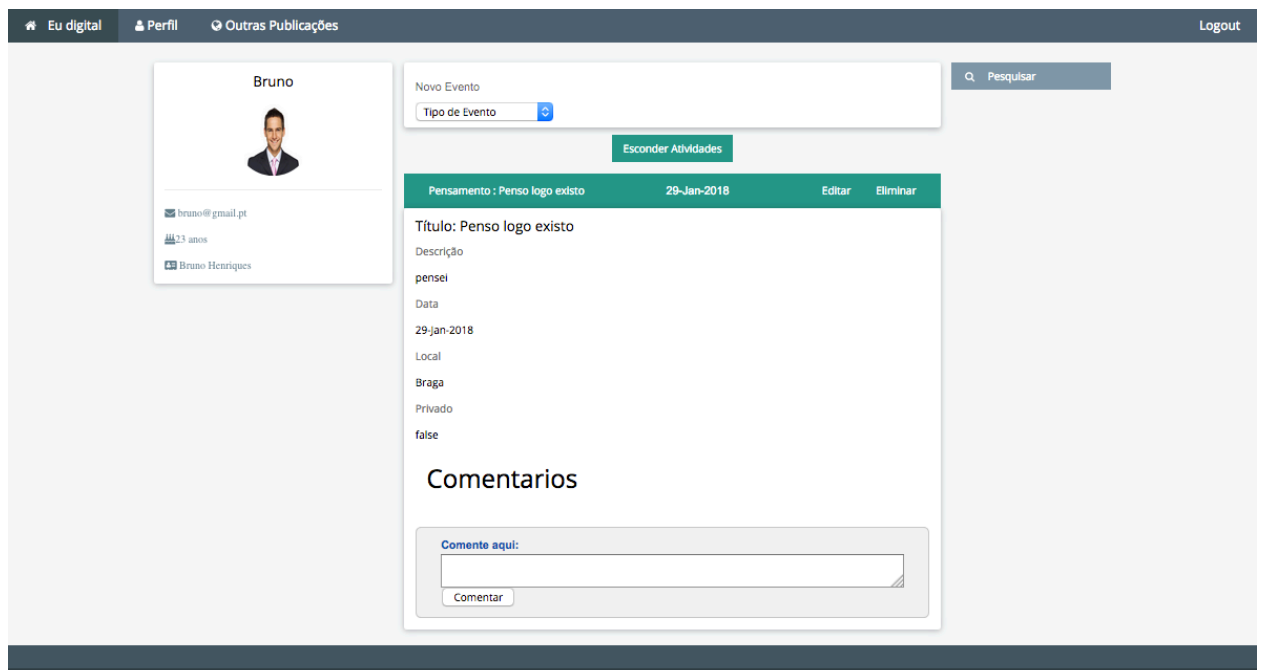
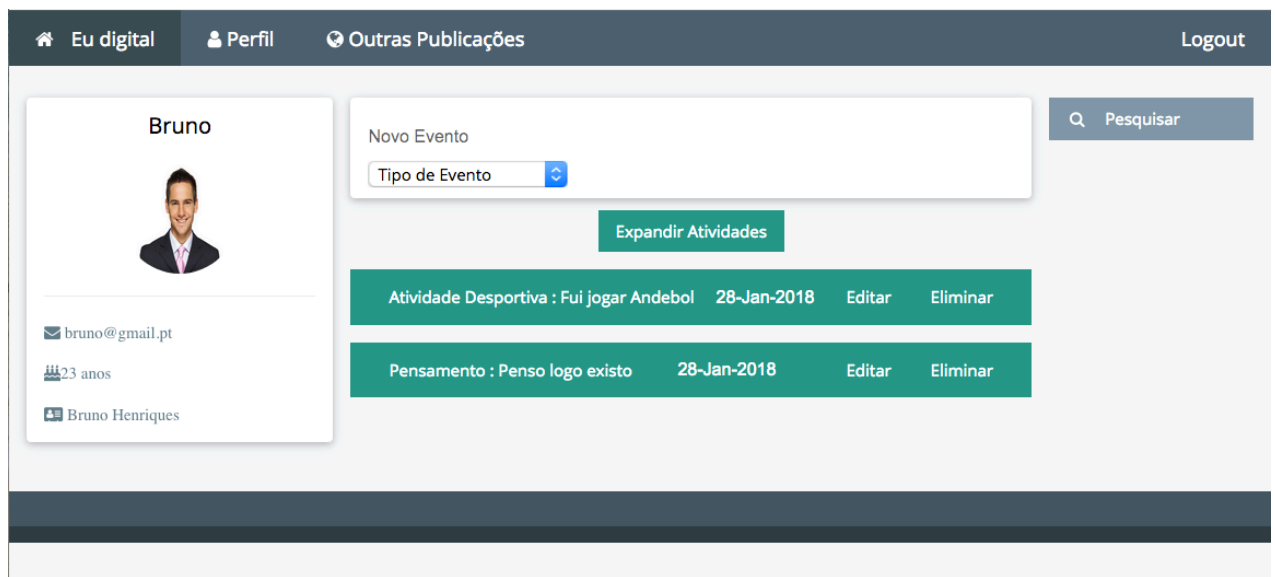


Figura 33 Página do feed Privado

5.4. Feed “Público”



Figura 34 Página do Feed público

5.5. Criação de Atividades

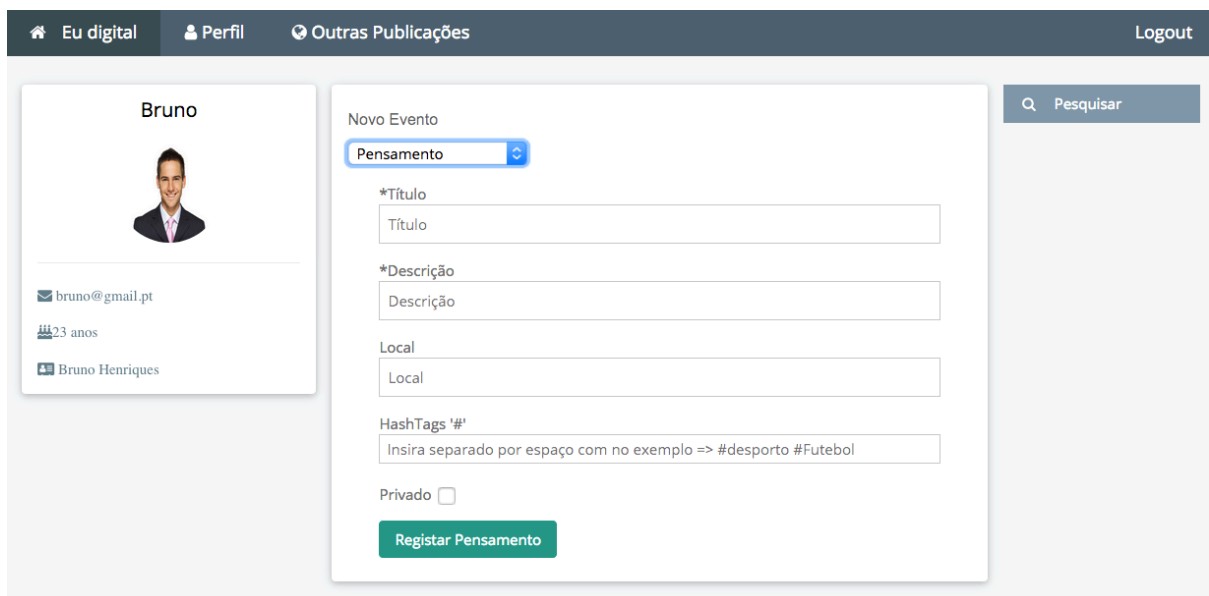




Figura 35 Página de Criação de Atividades

Nota: Difere de atividade para atividade

5.6. Página de Perfil



Remover Imagem

 Informações Pessoais

Primeiro Nome

Bruno

Ultimo Nome

Henriques

Idade

23

Email

bruno@gmail.pt

Password

.....

Guardar Alterações

Figura 36 Página de Perfil

5.7. Edição de Atividades

*Título

Penso logo existo

*Descrição

pensei

Local

Braga

Privado

☐

Guardar Alterações

 Voltar

Figura 37 Página de edição de atividade

Nota: Difere de atividade para atividade

6. Diagrama da Arquitetura da Aplicação

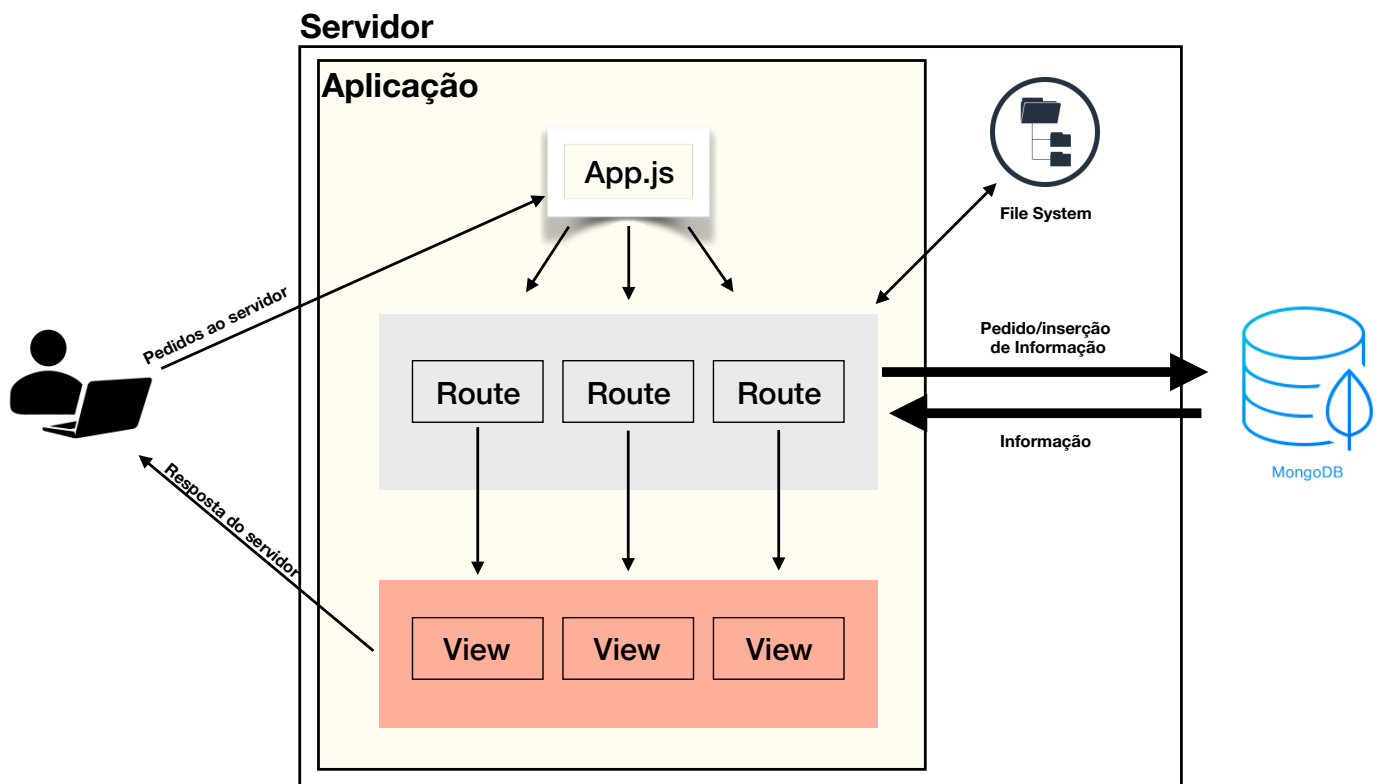


Figura 38 Arquitetura da Aplicação

7. Conclusões e trabalho futuro

Finalizado este trabalho fazemos um balanço positivo em termos de resultados obtidos. Foram encontradas várias dificuldades na realização do projeto devido principalmente a uma má estruturação do modelo de dados, tal como foi referido na respetiva secção. Apareceram outras dificuldades principalmente nos eventos que continham fotografias, isto porque, o facto de haver ficheiros associados a publicações dificulta a sua edição e remoção. Tomemos o exemplo do álbum fotográfico: cada evento deste tipo tem várias fotos e, a cada uma delas está associada uma descrição e várias pessoas, isto causa imensa complexidade na sua edição porque os objetos JSON já começam a ter muitos níveis de complexidade.

O assincronismo do node js também foi um pouco complicado de gerir devido à nossa pouca experiência com esta Framework. Entendemos que o assincronismo é bastante vantajoso em termos de processamento, no entanto, é preciso ter cuidado quando precisamos de executar queries em que o resultado da próxima depende do anterior. Para resolver este problema tivemos que refazer algumas vezes o código para o adequar à utilização do módulo `synchronize`, o que nos tomou algum tempo.

Em termos de requisitos cumpridos temos 14 de 16. Ficaram por fazer os seguintes requisitos:

- Partilha de eventos nas redes sociais
- Autenticação com o facebook e Google

Estes requisitos não foram implementados porque, no caso da partilha nas redes sociais, era necessário a geração de um link para cada atividade a partilhar e no caso da autenticação com o Facebook/Google optamos por não “perder tempo” em analisar a implementação do módulo `passport`.