**I**nstituto **S**uperior de **E**ngenharia de **L**isboa

Licenciatura em Engenharia Informática e de Computadores

Semestre de Verão 2014/2015

CLX – Chelas LX  
Programação na Internet

**Trabalho elaborado por:**

**Grupo 18**

João de Deus Nº35368

Pedro Duarte Nº36832

Índice

[Introdução 1](#_Toc425451823)

[**Manual de Instalação** 1](#_Toc425451824)

[**Endpoints** 1](#_Toc425451825)

[**Acesso a dados** 1](#_Toc425451826)

[Conclusão 1](#_Toc425451827)

# Introdução

# Manual de Utilização

Para ser se possível a execução da aplicação *WEB*, CLX – Chelas LX, é requisito ter instalado o *Node*.js e o sistema de gestão de dados relacional, *postgres*.

Em primeiro lugar, é necessário criar uma base de dados no *postgres* com o nome “PF - PI” e a partir do ficheiro “*PF* *–* *PI*.*backup*” fazer um restauro.

Em seguida será necessário instalar os módulos requeridos pela aplicação presentes no *package*.*json* através do comando “*npm* *install*”. Sendo necessário a execução da aplicação servidora pelo comando “*node server.js”.* Os comandos acima descritos terão de ser executados através da consola do *Node*, na diretoria correspondente ao projeto “%/clx – chelas lx”.

Depois de executados os passos acima já é possível aceder à aplicação através do *browser* colocando a *url* como “*localhost*:3000”, indo para a *home* *page*.

CLX – Chelas LX

Esta aplicação foi desenvolvida com recurso a *NODEJS* como tecnologia *WEB*, *PostgreSQL* como motor de base de dados relacional e *Bootstrap* como *framework* da componente visual, *CSS*, aplicação de regras de estilo em cascata.

De forma a facilitar a explicação das opções tomadas no desenvolvimento da aplicação, a parte de desenvolvimento pode-se dividir em três partes.

*“SERVER SIDE”*

A *framework* base usada para desenvolver a aplicação, foi o *Express* para *NODEJS*. Como *template* *engine*, foi usado o *Jade*, e como suporte para autenticação foi usado o *middleware* *Passport*.

A aplicação desenvolvida tem um sistema de rotas, que permite obter os recursos necessários ao funcionamento da aplicação. Existem dois tipos de rotas, as rotas que podem ser acedidas por qualquer utilizador, isto é, não é necessario autenticação, não havendo necessidade da utilização do middleware de autenticação nessa rota. E as rotas que necessitam de autenticação, utilizando o middleware de forma a verificar se o utilizador se encontra ou não autenticado. Caso o utilizador não se encontre autenticado, e na tentativa de acesso a rotas “protegidas”, o mesmo é reencaminhado para a página inicial ou para a página da lista dos anúncios, no caso de rotas relativas a criação de comentários e edição de anúncios.

Na página de login, é possível também efectuar a recuperação da palavra passe, sendo esta recuperação feita através do email utilizado no registo. Esse email introduzido é verificado contra o email presente na base de dados e em caso afirmativo é enviado para o utilizador através desse mesmo email, o nome do utilizador e a respectiva palavra passe.

Para o envio do respectivo email, foi utlizado o midleware request, que permite a execução de um método http, sendo que neste caso especifico foi usado o método POST. É este o único sítio da aplicação onde este midleware é utilizado.

Para fornecer, suporte ao envio de email, foi usada a aplicação http://mandrill.com/, que após registo fornece um chave, que é enviada no body do pedido, de forma a autenticar o pedido. No pedido de contacto, dúvidas ou questões, que também procede ao envio de um email, mas neste caso para o gestores da aplicação, o pedido e efectuado com um pedido AJAX, não utilizando o request.

------------------------------------------------------------------------------------------------------------------------------

A aplicação foi estruturada recorrendo ás seguintes entidades, mapeadas em tabelas na base de dados.

Criamos uma entidade "queixinha" que representa a para com a entidade "comentário" as principais entidades da aplicação. A primeira agrega toda a informação pertencente a uma queixinha, sendo que ao especificado no enunciado acresentamos um campo de nome "state" que através de um booleano especifica o estado da queixinha (aberta ou fechada). De referir que uma queixinha tem sempre um utilizador "owner" que é o utilizador criador da queixinha. Um utilizador pode visualizar na aplicação uma listgem de todas as suas queixinhas. Esta funcionalidade será descrita mais á frente.

A entidade "comentário" representa os comentários que se podem adicionar a uma queixinha. Tambem aqui um comentário pertence ao utilizador que inseriu o mesmo, assim como um comentário foi inserido no ambito de uma determinada queixinha. Como tal estas duas relações foram modeladas nesta entidade.

Como o tema de uma queixinha insere-se numa determinada categoria de temas, criamos tambem a entidade "categoria" que não é mais do que metadata representada na base de dados na forma de uma tabela simples com o descritivo de cada categoria. A inserção, edição e remoção de categorias não foi prevista na aplicação, mas a ser seria realizada pelo utilizador gestor através de uma eventual aplicação de administração.

A entidade "dbuser" representa um utilizador da aplicação. Nesta tabela guardamos o username, password e email de um utilizador assim como o nome que quer usar no âmbito da aplicação, modelado pelo campo "nickname". Existe tambem um campo "role" que serve para destinguir os utilizadores gestores dos demais utilizadores. A razão de termos chamado a esta entidade dbuser e não user, prende-se com o facto de "user" ser uma palavra reservada em PostgreSQL. Daqui para a frente chamaremos simplesmente user á entidade dbuser.

A última entidade nativa que criamos foi a entidade "vote". Esta representa os votos dos utilizador numa determinada queixinha. Para isso temos nesta entidade como seria de esperar, as relações para um user e para uma queixinha. Acresentamos ainda o campo "value" que com um booleano representa o sentido do voto.

Por último criamos uma tabela auxiliar "queixinha\_dbuser". Esta destina-se a implementar a funcionalidade de seguimento de queixinhas pelos utilizadores. Assim quando um utilizador pretende seguir uma queixinha, ou seja ser notificado das alterações de estado da mesma, é inserido nesta tabela a relação entre o utilizador em questão e a queixinha a seguir. Um utilizador pode seguir tantas queixinhas quantas quiser. Nesta entidade criamos ainda um campo "dirty" que mais uma vez através de um booleano permite assinalar as queixinhas que sofreram alterações ainda não visualizadas pelo utilizador seguidor. Assim este estado começa por default a false, mudando para true quando a queixinha for adicionada de um comentário ou for alterada pelo proprietário da mesma. Assim que o utilizador seguidor observar a queixinha em questão (através de um pedido GET a esta queixinha especifica) o estado deste campo retorna a false.

A utilização destas entidades na aplicação é feita através de DAOs que agrupados na pasta “db” não só manipulam o acesso á base de daos destas mesmas entidades como estabelecem representações das mesmas no context NodeJs. Na pasta “routes” criámos rotas para atender aos aspectos funcionais da aplicação, sendo que a manipulação da informação necessária a cada pedido tratado nestas rotas é feito através dos DAOs antes descritos.

Ao descrevermos o funcionamento da aplicação iremos acompanhar essa descrição com referências aos promenores de implementação mais relevantes.

LOGIN

Depois de autenticados na aplicação podemos aceder a uma lista de todas as queixinhas fechadas existentes. Sendo esta lista paginada, foi necessário acresentar ao URI “/queixinhas/” o campo “page” que representa a página desta listagem. Assim a fim de efectuar esta paginação estabelecemos que no pedido inicial não é inserido o campo “page”, o que vai despoltar um pedido á base de dados da contagem de todas as queixinhas fechadas existentes através do método “getTotalNumberClosedQueixinhas” e posteriormente guardado numa variável global. Sendo o número de queixinhas por página imutável e igual a 5, esta contagem é passada á vista por forma a ser construido em cada pedido o sistema de paginação com o número de páginas certo.

Ao escolher uma queixinha especifica desta lista, somos redirecionados para os detalhes dessa queixinha incluindo todos os comentários efectuados até ao momento e esssa queixinha. Para criar esta vista é necessário recolher toda a informação relativa a esta queixinha. Aqui foi nossa opção efectuar o provisionamento desta informação do lado do servidor Web e não como uma função da base de dados. A razão desta nossa escolha foi o facto de que ainda que saibamos que seria mais acertado fazer esta operação do lado da BD isso não cairia no ânbito desta cadeira. Assim a função que trata da rota “queixinhas/id”, é uma função que executa inumeros pedidos á base de dados em sequência, a fim de obter informação acerca da própria queixinha, os comentários dessa queixinha, o numero total de votos, assim como saber se a queixinha já foi votada e se é seguida pelo utilizador em uso. Esta última informação tem relevância a fim e bloquear ou não as funcionalidades de seguir e votar uma queixinha.

# Endpoints HTTP

Registo do middleware correspondente á Home Page: app.use('/', route\_idx):

Rota definida:

Home Page: router.get('/', function(req, res, next) {});

Registo do middleware correspondente á visualização, edição e inserção de anúncios: app.use('/announcements', route\_clx)

Rotas definidas:

Lista de todos os anúncios: router.get('/’, function(req, res, next) {});

Listagem de anúncios criados e favoritos de um utilizador: router.get('/dashboard', function(req, res, next) {});

Criação de um novo anúncio: router.get('/new', function(req, res, next) {});

Criação de um novo anúncio: router.post('/new', function(req, res, next) {});

Visualização do anúncio com o identificador id: router.get('/:id', function(req, res, next) {});

Edição do anúncio com o identificador id: router.get('/:id/edit', function(req, res, next) {});

Guardar a edição realizada num anúncio: router.post('/:id/edit', function(req, res, next) {});

Pesquisa por Cidade, título e categoria de anúncios: router.post('/find', function(req, res, next){});

### Acesso a dados

O acesso a dados é realizado pela criação de um módulo “*dbacess.js*”, que são disponibilizadas funções para criação, atualização e inserção de anúncios, comentários, utilizadores e categorias para os vários anúncios. Estas funções são exportadas para o seu uso mais facilitado.

Objetos criados para guardar os dados retornados do acesso:

* *anuncio* – entidade principal da aplicação;
* *user* – utilizadores registados na aplicação;
* *comment* – comentários realizados a anúncios;
* *categoria –* categoria do anúncio;
* *favorito* – anúncios favoritos de cada utilizador;
* *pontuacaoUtil –* pontuação atribuída por cada comprador ao vendedor.

Funções disponibilizadas para retorno dos dados presentes na base de dados:

* *getAnnouncs* = function (page, cb){} – permite retornar todos por determinada página suporta á implementação da paginação,
* *getAnnounc* = function (id, cb) {} – retorna o anuncio com o identificador *id*;
* *getCountAnnounc* = function (cb) {} – número dos anúncios presentes na base de dados;
* *getAnnouncUser* = function(username, cb){} – retorna os anúncios publicados por determinado *username*,
* *getAnnouncFavoriteUser* = function(username, cb){} – retorna os anúncios favoritos do *username*;
* *getUser* = function (name, cb){} – retorno do utilizador *name*;
* *getUserbyEmail* = function (email, cb){} - retorno do utilizador com o *e-mai*l *email*;
* *getCategoria* = function (desig, cb){} – retorno da categoria com a designação *desig*;
* *getComentAnnounc* = function (id, cb){} – retorno dos comentários associados ao anúncio com o identificador id;
* *getPontuacaoUtil* = function (username, cb){} – retorno da pontuação atribuídas pelos vários utilizadores ao *username* passado por parâmetro;
* *getAnuncioByFilter* = function (localizacao, titulo, categoria, cb){} – pesquisa por anúncios na localização, título e categoria passada por parâmetro.

Funções de suporte á inserção de anúncios, utilizadores, comentários e categoria:

* *newAnnounc* – inserção de um anúncio;
* *newUser* – inserção de um utilizador;
* *newCategoria* – inserção de uma nova categoria;
* *newComment* – inserção de um comentário.

Todas as funções descritas acima apresentam interface assíncrona por captura dos dados na função *cb*, *callback*.

Base de Dados

# Conclusão

Com a realização deste trabalho foi-nos possível