**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

O Nome dado a aplicação desenvolvida foi “*CLX – Chelas LX*”, esta aplicação consiste em que diversos utilizadores possam comprar e vender produtos, e interagir entre si através de comentários.

A aplicação tem dois tipos possíveis de utilizadores:

- Os utilizadores não registados, tem acesso à lista de anúncios, ao seu detalhe, visualização de comentários efetuados por utilizadores registados e filtragem por título, localização e categoria.

- Os utilizadores registados, tem acesso a ilimitado a todos os anúncios, podem comentar, atribuir uma classificação ao vendedor, seguir, criar, e fechar os seus próprios anúncios.

Todos os utilizadores que não se encontrem autenticados são considerados como utilizadores não registados.

Um utilizador, quando se regista é obrigado e inserir um endereço de *email* válido, no entanto a aplicação não verifica a existência do endereço inserido. Este endereço tem como objetivo poder ser utilizado na recuperação da palavra-passe, isto porque, quando o utilizador tenta recuperar é-lhe pedido o *email* com o qual se registou, pois será esse o endereço utilizado para envio dos dados em questão.

# 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 necessário 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 efetuar 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 respetiva palavra passe.

A aplicação foi estruturada recorrendo às entidades mapeadas na base de dados relacional, descrita abaixo no título Base de Dados.

LOGIN

Após um utilizador se autenticar na aplicação pode aceder ao seu *dashboard* de anúncios.

Ao escolher um anúncio específico desta lista, somos redirecionados para os detalhes desse anúncio incluindo todos os comentários efetuados até ao momento. Para criar esta vista é necessário recolher toda a informação relativa ao anúncio. Assim a função que trata da rota “*announcements/id*”, é uma função que executa inúmeros pedidos á base de dados em sequência, a fim de obter informação acerca do anúncio, através da função “*getAnnounc*”, os comentários do anúncio, através da função *getComentAnnounc*.

No caso do utilizador registado ser o editor do anúncio é-lhe possibilitada a sua edição.

# *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

A entidade "anuncio" representa para com a entidade "comentario" as principais entidades da aplicação. A primeira agrega toda a informação pertencente a um anúncio, sendo que ao especificado no enunciado acrescentamos um campo booleano especificando o estado do anúncio (cancelado ou ativo). De referir que um anúncio tem sempre um utilizador "owner" que é o utilizador editor do anúncio. Um utilizador pode visualizar na aplicação uma listagem de todos os seus anúncios e os anúncios que pretende receber notificações que não foram criados por si.

A entidade "comentário" representa os comentários que se podem adicionar a um anúncio. Também aqui um comentário pertence ao utilizador que inseriu o mesmo, assim como um comentário foi inserido no âmbito de determinado anúncio.

Como cada anúncio tem uma categoria, criamos também a entidade "categoria" que contem as categorias disponíveis para os anúncios a publicar. A 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 "utilizador" representa um utilizador da aplicação. Nesta tabela guardámos o username, password e email do utilizador. 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.

# Conclusão

Com a realização deste trabalho foi-nos possível