# FLYC Key Processes (as built)

# External components

## REST calls : REST-Client

1. Install from:
   1. <https://github.com/archiloque/rest-client>

# Authentication

## Auto login (cookie or session)

1. **LoginController / login\_required**
   1. Except calls for general pages which include:
      1. Home
      2. Process\_login
      3. Admin
      4. Bl\_login (bookmarklet login)
      5. Bl\_process\_login
      6. Bl\_logout
2. [admin login request]
   1. Check if admin is already logged in
   2. Otherwise redirect to ‘admin’ login page
3. [admin was logged in and ‘standard’ user is trying to access non-admin page]
   1. Logout admin
   2. Clear session
4. [user doesn’t exist in session] => *session[:user] is null*, Login from cookie
   1. Check if a cookie with a login key exists on the computer => *cookies[:auth\_token] is not null*
   2. Find a ‘person’ (user) by the ‘login key’
   3. Check if the token hasn’t expired
   4. If all good, put the ‘person’ instance in session
5. [cookie login failed], execute **‘access\_denied’**
   1. Store ‘calling page’ in session (as ‘*session[:return\_to]*’)
   2. Redirect to ‘login’ page with login error message

## Standard login

1. User enters ‘username’ and ‘password’ attributes in ‘login’ screen
2. **LoginController / process\_login**
3. [valid]
   1. Authenticate => **Person.authenticate(params)**
   2. [authenticated]
      1. [‘save\_login’ field checked]
         1. **@person.remember\_me**
            * Set token expiry: 1 minute from now
            * Create token using the **Digest::SHA1.hexdigest** method using the *primary\_email* and *remember\_token\_expires* parameters as the string to digest
            * Store additional attributes to @person instance
         2. Store token to a cookie => *cookies[:auth\_token]*
      2. [status check == 1] => User was added to site but not registered (i.e. invited to join)
         1. Login failed with appropriate message
      3. [status check == 2] => user registered but hasn’t confirmed registration
         1. Login failed with appropriate message
      4. [status check == 3] => user confirmed registration
         1. Login successful
   3. [login failed]
      1. [*controller* parameter exists] => i.e. a *bookmarklet* call
         1. Don’t redirect, return
      2. [otherwise]
         1. Redirect to homepage
   4. [login successful]
      1. [*controller* parameter exists] => i.e. a *bookmarklet* call
         1. Don’t redirect, return
      2. [session[:return\_to] parameter exists]
         1. Redirect to last visited page
      3. [otherwise]
         1. Redirect to ‘**RoleApplicationsController / view\_applications**’

## Registration [as Company Contact] [TBD]

1. Select ‘Company Contact’ from the drop-down on the homepage
2. Display message informing the user to perform a search on the company and ask to be invited if it exists
3. Standard registration rules apply
4. Addition: type in the company name
5. Wizard for creating the company profile (?)

## Registration

1. Registration can only be done from the homepage (**ApplicationController / home**)
2. **AccountController / process\_registration**
3. Create a *Person* instance
   1. Set the person status to 2 (registered but not verified)
4. Create a *BetaPassword* instance
   1. *BetaPassword* inherits from *ActiveForm* (which has validation capabilities but no DB model dependencies)
5. [beta\_password not valid]
   1. Set registration flag to false
6. Store *Person* instance in DB
7. [successful]
   1. Redirect to **MyMailer / email\_confirm**
      1. Retrieve *Person* instance from DB based on *person\_id*
      2. Create new *Message* instance
         1. Set the *subject*
         2. Set the *recipients*
         3. Set the *sender*
      3. Create an *EmailConfirmation* instance
         1. Set the *confirmation\_string* using the *Digest::MD5.hexdigest* method
         2. Set the
      4. Set additional *Message* parameters (person name and ‘confirmation\_string’)
      5. Create a *MyMailer* instance (by calling the **MyMailer::create\_setup** method)
      6. Set message content to ‘text/html’
      7. Deliver the message (calling **MyMailer::deliver**)
      8. Attempt to store *Message* and *EmailConfirmation* instances
      9. [successful]
         1. Redirect to **AccountController / email\_confirm\_sent**
      10. [failed]
          1. Redirect to homepage with error message
8. [failed]
   1. Return to homepage with appropriate error message

## Login via Facebook [TBD]

1. ‘Sandbox’ mode
   1. When ‘Sandbox Mode’ is set to ‘On’ (in ‘Apps -> Flyc -> Settings -> Advanced’), the site will only be available to the application ‘developers’
   2. ‘Developers’ can be defined under ‘Apps -> Flyc -> Roles’. I think ‘Administrator’ is automatically a ‘developer’
2. Using ‘Facebook login’ in development mode
   1. Make sure ‘Sandbox’ mode is set to ‘On’ (see above)
   2. Set the ‘Site URL’ under ‘Website’ on the main Flyc applicaton page to
      1. [**http://localhost:3002/**](http://localhost:3002/)
   3. Make sure the ‘App Domain’ under ‘Basic info’ field is blank
3. Code for login

<div id="fb-root"></div>

<script>

window.fbAsyncInit = **function**() {

FB.init({

appId : '156127131075698',

status : true,

cookie : true,

xfbml : true

});

};

(**function**(d){

**var** js, id = 'facebook-jssdk'; **if** (d.getElementById(id)) {**return**;}

js = d.createElement('script'); js.id = id; js.async = true;

js.src = "//connect.facebook.net/en\_US/all.js";

d.getElementsByTagName('head')[0].appendChild(js);

}(document));

</script>

<div class="fb-login-button">Login with Facebook</div>

# Email communications

## Creating emails

1. General explanation of main classes
   1. **MyMailerEmail** – object that holds the finalized email for delayed procesing
2. Places where messages are being sent
   1. **GeneralApplicationController / send\_feedback**
   2. **GeneralController / send\_contact\_us**
   3. **UserMessagesController / send\_reply\_to\_message**
   4. **AdminMessagesController / admin\_send\_reply\_to\_message**
3. Creating a message
   1. Create a *MessageForm* object and set the following parameters (*MessageForm* inherits from *ActiveForm*)
      1. **Sender\_name**
      2. **Sender\_email**
      3. **Body**
   2. Create a parameters hash to hold all message metadata (envelope type information) including the body of the message
      1. **parent\_message\_id** – in case this message is a reply (to maintain the correspondence chain)
      2. **subject**
      3. **message\_body**
      4. **sender\_id** – user id
      5. **sender\_reply\_to\_email**
      6. **sender\_name**
      7. **recipients** []– An array of Hash objects containing the following attributes
         1. **id**
         2. **email**
         3. **name**
4. Sending the message – **BaseController / send\_message**
   1. Set the **http\_host** parameter to hold the appropriate host name (taken from request.env["HTTP\_HOST"]). This ensures that the appropriate domain name is used for links in the message body (e.g. ‘localhost’ or ‘flyc.co.nz’ etc)
   2. Call *send\_message* with the following parameters
      1. Metadata\_params\_h
      2. Body\_params\_h
      3. Is\_reply
         1. True
         2. False
      4. Type
         1. 1 = Contact us
         2. 2 = System notification
         3. 3 = Action missing
         4. 4 = Forgot password
         5. 5 = Registration confirmed
         6. 6 = General message
         7. 7 = Feedback
      5. Exchange\_type
         1. 1 = User to Admin
         2. 2 = Admin to User
         3. 3 =
         4. 4 = System to Admin
         5. 5 = System to User
         6. 7 = User to System (Feedback recipient, currently admin)
   3. Above method calls **MyMailer::send\_message\_delayed** with above parameters
      1. Set the system email addresses:
         1. System email address – [system@flyc.co.nz](mailto:system@flyc.co.nz)
         2. Admin email address – [admin@flyc.co.nz](mailto:admin@flyc.co.nz)
         3. Member email address – [member@flyc.co.nz](mailto:member@flyc.co.nz)
         4. No-reply email address – [no-reply@flyc.co.nz](mailto:no-reply@flyc.co.nz)
      2. Extract the ‘admin’ user details from the database (*person\_type\_id = 0*)
      3. Set the system *user\_id = 0*
   4. [not reply] – brand new email
      1. Set the email subject based on the type (e.g. type = 1 -> “‘Contact Us’ message”)
   5. Set the sender and recipient details based on the *exchange\_type*
   6. [system message to user or admin]
      1. Iterate through all recipients
      2. Create individual messages using each recipient’s information (e.g. name)
      3. Create **MyMailer** object
         1. Currenty there is support only for ‘action\_missing’ message sent from the system to the adminby calling *action\_missing\_message* (calling *create\_action\_missing\_message* creates a new instance of *MyMailer* and calls *action\_missing\_message* method)
         2. There is support for sending multiple system messages to users
   7. Store HTML message in database
      1. Create instance of **MyMailerMessage**
      2. Store in database
   8. Strip HTML tags from message body and store in **@body\_plain** attribute
   9. Store **@body\_plain\_summary** by using first 80 characters of the stripped message body
   10. Create and populate the **MyMailerMetadata** object
       1. **Message\_type\_id** – Same as the *type* parameter above
       2. **Sender\_id**
       3. **Message\_status\_id**
          1. 1 = Unread
          2. 2 = Read
          3. 3 = Replied
          4. 4 = Archived
          5. 5 = Deleted
       4. **Parent\_message\_id**
       5. **Recipients** – Iterate through all recipients
          1. Recipient\_id
          2. Message\_status\_id – set default status to ‘Unread’ (status id = 1)
       6. **Subject**
       7. **Message** -> **MyMailerMessage** from previous step
       8. **Message\_summary**
   11. Store the **MyMailerMetadata** object in database
   12. [not reply]
       1. Update the *parent\_message\_id* to be the same as the current *message\_id*
   13. Create the email action redirector
       1. Set the appropriate controller and action based on the **exchange\_type** of the message
       2. [not system generated email]
          1. Create **MyMailerAction** by using **MyMailerAction::create\_o**
          2. Create the redirection key by using **Digest::MD5.hexdigest** with *sender\_reply\_to\_email* and current time as string to be digested
          3. Store **MyMailerAction** in database
          4. Iterate through all recipients
             * Create a new Hash to hold message metadata and populate it
             * Add *redirection\_key* and *person\_name* into the **body\_params\_h** hash (received by this method)
          5. Create **MyMailerEmail** object
             * **My\_mailer\_metadata\_id**
             * **Headers** – metadata information (stored in the newly created hash above)
             * **Body\_attributes** – *body\_params\_h*
             * **Template –** template to use for email (e.g. ‘new\_message\_message’ is the template for a new message)
          6. Store **MyMailerEmail** in database
       3. [system generated email]
          1. Iterate through all recipients
          2. Construct emails by adding header and footer to the created body
          3. Deliver the message using the **deliver** method

## Delivering emails (delayed)

1. HTML emails’ header and footer
   1. Header: **MyMailer::email\_header**
   2. Footer: **MyMailer::email\_footer**
2. Important files
   1. **/lib/daemons/Flyc\_mailer.rb** - the daemon
   2. **/config/environment.rb** – main configuration file for application. The *flyc\_mailer.rb* daemon relies on this configuration file as well
3. Running on Linux (HostingRails)
   1. From command line:
      1. *ruby ~/{env}/script/runner ~/{env}/lib/daemons/flyc\_mailer.rb*
   2. Ctrl-C to exit command line and leave it running in background
   3. Running the daemon for the test environment:
      1. *uby ~/jobby\_test/script/runner ~/jobby\_test/lib/daemons/flyc\_mailer.rb*
4. Running on Windows
   1. From the command line:
      1. {needs testing} - *start "spec\_server" /min ruby.exe lib/daemons/flyc\_mailer.rb*
   2. Execute this from the rails’ root:
      1. *ruby.exe lib/daemons/flyc\_mailer.rb*
5. **flyc\_mailer.rb** explained
   1. Required
      1. **Net/pop** – general pop3 ruby access api
      2. **Tmail *–*** general email manipulation api
   2. Set sleep time to **10 seconds**
   3. Perform an SQL query from table **my\_mailer\_emails** ordered by ascending *priority*
   4. For each email, execute **MyMailer.deliver\_init**. I assume that this syntax instantiates a **MyMailer** object, calls **init** and then executes the **deliver** method. Parameters passed:
      1. Headers
      2. Body\_attributes
      3. Template
   5. **MyMailer::init**
      1. Set the MyMailer parameters with the attributes received
      2. Render the template by calling **render\_message** with:
         1. Template name
         2. Template attributes
      3. Create a ‘part’ object and include the appropriate header and footer to the generated rendered message

## Importing emails

1. Prerequisites
   1. Ensure the **member\_catcher@flyc.co.nz** account is configured on the SMTP server
   2. Ensure the **default address** (forwarding of all unrouted emails for @flyc.co.nz) is configured to forward emails to **member\_catcher@flyc.co.nz**
2. Important files
   1. **/lib/daemons/Flyc\_mailer.rb** - the daemon
   2. **/config/mail.yml –** mail configuration for outgoing mail (SMTP configuration)
      1. Can configure each environment individually. Environments:
         1. **Default** – for all environments
         2. **Development**
         3. **Test**
         4. **production**
      2. Important parameters:
         1. **Type:** pop
         2. **Server**: [www.flyc.co.nz](http://www.flyc.co.nz)
         3. **Port:** 43
         4. **Ssl:**
         5. **Username:** [member\_catcher@flyc.co.nz](mailto:member_catcher@flyc.co.nz). Make sure the user ‘member\_catcher’ exists in the SMTP server used with the appropriate password below
         6. **Password**: tomer66
   3. **/config/environment.rb** – main configuration file for application. The *flyc\_mailer.rb* daemon relies on this configuration file as well
3. Shared logic with **Delivering emails (delayed)** section above
   1. Running on Linux (HostingRails)
   2. Running on Windows
   3. **‘flyc\_mailer.rb** explained’ up to ‘Set sleep time to 10 seconds’ (inclusive)
4. Load **/config/mail.yml** configuration file
5. Create a new **Net::POP3** object with the parameters:
   1. **Server**
   2. **SSL**
   3. **Username** – this is the ‘**member\_catcher**’ account
   4. **Password**
6. Iterate through all emails and execute **TMail::Mail.parse** which creates a **TMail** object representing the email received

## Reminders [TBD]

## Notifications [TBD]

# Logging

## Activity log

1. Important *session* and *request* attributes
   1. **Session[:last\_activity\_o**] – Last object
   2. **session[:last\_primary\_activity\_o]** – Last primary activity object
   3. **Session[:last\_5\_urls] –** Array containing 5 last urls visited (including current one)
   4. **Request.env[“REQUEST\_URI”] –** full url of current page
2. Activity types
   1. Primary -
   2. Secondary
3. Other important attributes
   1. **Activity::activities** – Hash containing meta-data information for each action in the system
      1. Structure of the Hash:
         1. **Key** – action name
         2. **Value** – array of meta-data information
      2. Structure of the Array:
         1. **Action friendly name** – a friendly name to be displayed on the website
         2. **Entity type**
            * 0 – General
            * 1 – Job Application
            * 2 – Agency
            * 3 – company
            * 4 – contact
            * 5 – General (i.e. Contact Us, Feedback)
            * 6 – Application notes
            * 7 - File
         3. **Entity view mode**
            * 1 - New
            * 2 - Save
            * 3 - View
            * 4 - Edit
            * 5 – Update
            * 6 – Delete
            * 7 – Send
         4. **Is primary view**? – 0 or 1
4. **BaseController / activity\_logger\_start** [before\_filter] is executed at each controller call
   1. *@friendly\_names\_h* hash is instantiated
5. **BaseController / activity\_logger\_end** [after\_filter] is executed at the end of each controller call
   1. [same *action* and *controller* were used] – indicating a ‘refresh’ or clicking a link that leads to the same page the user was on
      1. Update only the ‘update\_at’ field using the *touch* method and return
   2. Add current *request\_uri* to the session array **Session[:last\_5\_urls]**
   3. Execute **Activity::create\_o** [static method]
      1. Create new *Activity*object
      2. Set the following parameters
         1. *Person\_id*
         2. *Action*
         3. *Controller*
      3. Identify all attributes in *params* that have “\_id” in them (e.g. “organisation\_id”)
      4. Create an array of all “id” parameters and their values
      5. If there are ‘friendly names’ for this type of id (e.g. “Hudson” for this particular ‘organisation\_id’), set it as well
      6. Example:
         1. *Parameter\_ids*: **organisation\_id** [id]
         2. *Parameter\_names*: **432** [name]
         3. *Parameter\_values*: **Hudson** [friendly\_name]
            * The value must be set from within the relevant controller, e.g. **ContactsController / edit\_contact:** @friendly\_names\_h["contact\_id"] = @contact.first\_name
   4. [current activity is primary] – check 3rd element in *Activity::activities* array
      1. Store activity instance into *session[:last\_primary\_activity\_o]* attribute
   5. [current activity is secondary AND no primary activity stored in *session*]
      1. n/a
   6. [current action is not mapped] (possibly new action that wasn’t classified and defined)
      1. Set the primary activity to be *home*
      2. Send an email to the admin letting them know there is an unclassified action
   7. Store the current activity (action) in *session[:last\_activity\_o]*

# Data display

## Sorting [TBD]

## Filtering [TBD]

## Pagination [TBD]

# Bookmarklet

## Initialisation

1. Shortcut for activating the bookmarklet:
   1. *javascript:(function(){document.body.appendChild(document.createElement('script')).src='http://localhost:3002/javascripts/bl/flyc.js';})();*
2. Place the shortcut above inside an ‘<a href>’ tag. The user can then drag the link onto the bookmarks bar in any browser. I.e.
   1. *<a href="javascript:(function(){document.body.appendChild(document.createElement('script')).src='http://localhost:3002/javascripts/bl/flyc.js';})();">Flyc Magic</a>*
3. Bookmarklet initialization explained
   1. The bookmarklet creates a new ‘script’ element and appends it to the ‘document.body’ of the current document
   2. The script’s src points to the **flyc.js** file which is located in the **/javascripts/bl** folder
   3. This file is executed as soon as the ‘script’ above is run
4. Bookmarklet processing explained
   1. ‘**current\_action**’ parameter
   2. Send multiple requests using ‘GET’
   3. Each request has 2 parameters
      1. **Flyc\_token** – the token to ensure the requests are authorised
      2. **Split** – the part no. out of the total no. of parts expected. Syntax is: “X-Y” where X is current part and Y is the total no. of parts expected
5. **Flyc.js** explained
   1. [jQuery alias doesn’t exist] – i.e. jQuery is not initialized
      1. Load **jQuery-1.5.1.min.js**
   2. Load the ‘processing’ component (called throbber) - **jquery.throbber.js**
   3. Execute the ‘**flyc\_runthis**’ function (from within the ‘**load\_javascript**’ function)
      1. [panel doesn’t exist]
         1. Main panel HTML is created here
         2. ‘**slide\_it**’ is called (sliding the panel from the side)
         3. Show the ‘processing’ icon
         4. ‘**ajax\_it**’ is called with the ‘**bookmark\_site**’ action
      2. [panel exists]
         1. The panel slides out by calling ‘**ajax\_it**’ with the ‘**close**’ action
   4. ‘**bookmark\_site’** calls ‘bookmark\_site’ action in the flyc application
      1. Current URL is sent (using ‘*document.location.href*’)
      2. ‘bookmarket\_site’ calls ‘**bl\_bookmark\_site**’
   5. ‘**bl\_bookmark\_site**’ - initialisation
      1. Key pages to capture for flyc processing:
         1. Search results page
         2. Job ad description page
         3. Job application page
      2. **Job board websites** - Define the job board websites that will be auto-detected based on the page the user had arrived from. The websites are made out of a Hash with the following pairs (in future should be read from database):
         1. **web-site name** (e.g. [www.seek.co.nz](http://www.seek.co.nz))
         2. **site’s details** (hash):
            * **name** - **Friendly name** (e.g. “Seek (New Zealand)”)
            * **pages** – **pages’ url querystring syntax and their configuration** (array). Each array element represents a page with 2 elements:

**[0] Url syntax** – e.g. “/JobSearch?” (from ‘Seek’)

**[1] Parameters hash** (currently supports only **‘inline’ => true / false**. Inline means that the job board shows jobs as a jQuery collapsible / expandable component on the page. There is no separate page for displaying the job ad details)

* + 1. Current supported job boards
       1. [www.seek.co.nz](http://www.seek.co.nz)
       2. [www.jobmaster.co.il](http://www.jobmaster.co.il)
    2. **Agencies’ websites** – define the list of agencies for auto-detection
       1. Similar to above logic, define and store agencies web-sites
       2. Currently only supports
       3. This part needs further development and testing
    3. Define the ‘page’ dropdown which allows the user to change the type of page they’re on – *This functionality needs reviewing*
  1. ‘**bl\_bookmark\_site**’ – processing
     1. 1 - Detect website
     2. 2 – Detect page based on the pages’ definitions above
     3. 3 – Detect role by matching the full url to the ‘**external\_link**’field in the **roles** table

## Action processing - ‘Ajax\_it’ explained

* 1. Show the ‘processing’ animated icon
  2. Replace html attribute ‘**reply\_received\_flag**’ with the value ‘waiting’
  3. If-then-else for each ‘action’ called from the client
  4. Important logic for use in ‘action’ segments
     1. **is\_no\_post\_action** (Boolean) defines if there is a ‘post’ at the end of processing or just a visual display change
     2. Grabbing fields off the main web-site (from ‘Seek’) - logic
        1. *$("#title").attr("value", $("h1[class='jobtitle']").text());*
        2. i.e. Assigning the value stored in an ‘<hr>’ tag and with class ‘**jobtitle**’ to the flyc jQuery field ‘**title**’
     3. Populating fields in the main web-site (‘Seek’ example)
        1. *$("#FirstName").attr("value", "Tomer");*
     4. Setting the ‘action’ defined in the application
        1. *url\_action = "http://localhost:3002/bookmarklet/parse\_page";*
     5. Opening the Flyc window with the current role
        1. *window.open("http://localhost:3002/application/" + $("#role\_application\_id").val() + "/view", 'flyc\_window');*
     6. Animating the size of the panel using jQuery
        1. *$("#flyc\_panel").animate({width: '400px', height: '300px'});*
     7. Hiding the ‘processing’ icon:
        1. *$.throbberHide({parent: $("#processing\_indicator")});*

1. Important functions
   1. **Slide\_it** – performs a slide of the panel from side to side (if it’s on the right, it will slide to the left and vice versa)
   2. **Send\_ajax** – the method used for sending the final request. Parameters used:
      1. **url**: url\_action,
      2. **type**: "GET",
      3. **data**: params\_s,
      4. **processData**: false,
      5. **crossDomain**: true,
      6. **dataType**: "script"
   3. **getSelText –** Function for grabbing the selected text off the main web-site
   4. **flyc\_runthis** – First function called when user first uses the bookmarklet. This function creates the main HTML template for the FLYC panel
   5. **load\_javascript** – function for loading a Javascript script file
      1. This function receives a Boolean flag determining if this is the last script to be loaded. If it is, the ‘flyc\_runthis’ function will be called to initialize the panel
2. Important panel elements used in the panel
   1. **flyc\_panel** (div) – the div containing the entire panel
   2. **inner\_panel** (div) – inner div containing the inner html
   3. **flyc\_token** (input hidden) – token ensuring the requests are authorized
   4. **header** (div) – dev containing the header information
   5. **processing\_indicator** (td) – table ‘td’ containing the ‘processing’ icon
   6. **primary\_messages** (div) – div containing the ‘soft’ messages for the panel
   7. **content\_panel** (div) – div containing the core content area for the appropriate action

## Ajax content swap

1. Rendering a page
   1. **Header (/views/bookmarklet/\_bl\_header.js.erb)** – a partial included in each ‘transaction’ which includes the following:
      1. Setting of the **current action** (for automatic querystring construction prior to posting a form from the panel)
         1. *<input type='hidden' id='current\_action' value='<%= params[:action] %>'>");*
      2. Updating a **communication status flag** indicating a reply was received
         1. *<input type='hidden' id='reply\_received\_flag' value='received'>");*
      3. Setting the **‘flyc\_token’** attribute stored in the application’s session
         1. *$('#flyc\_token').attr('value', '<%= session[:flyc\_token] %>');*
   2. Main content swap…
      1. Content is defined by states, see below
   3. Footer (/views/bookmarklet/\_bl\_footer.js.erb) – a partial including the following:
      1. Hiding of the ‘processing’ icon
         1. *$.throbberHide({parent: $("#processing\_indicator")});*
2. **States** – Changing the content of the panel without performing a ‘refresh’ requires the use of ‘*states*’. Currently all states are defined in an ‘if-then-else’ block in the main ‘js’ file (in this example, **bookmark\_site.js**). The following states are defined:
   1. **0** – Failed communications or validation, stay on current page
   2. **1** – Initial behavior, page start-up
   3. **2** – Successful transmission, stay on current page
   4. **3 –** Successful transaction, redirect to a new page
   5. **4** – Site already exists, allow update and delete options
   6. **5** – Actions subpage
   7. **6** – Job ad subpage
3. **Bookmark\_site.js (/views/bookmarklet/bookmark\_site.js.erb)**– Method for replacing HTML content on the flyc panel. By rendering a ‘JS’ file with a ‘false’ layout, the page will not ‘refresh’ or ‘redirect’ the page, just perform javascript instructions defined in the appropriate .js file (in this case, the *bookmark\_site.js* file). At the end of each ‘action’ in the **bookmarklet\_controller** file, the following code snippet will perform the required content swap (the ‘*state*’ must be defined prior to running this code)
   1. *respond\_to do |format|*
   2. *format.js {render '/bookmarklet/bookmark\_site.js', :layout => false}*
   3. *end*

## Error handling

1. Error handling on the client’s side is done by the following try and catch block
   1. The *header* partial defines the overall ‘try’ block as follows
      1. *try{*
   2. The *footer* partial defines the ‘catch’ block as follows. In case there is a client side error, a detailed ‘alert’ will pop-up :
      1. *} catch(e){ var ee = e.message || 0; alert('Error: \n\n'+e+'\n'+ee); }*

## Login [TBD]

## Registration [TBD]

## Parsing / Scrapping HTML code [TBD]

# Administrations

## Messaging [TBD]

## Managing user feedback [TBD]

## Managing job boards, agencies, companies and contacts [TBD]