/vendor/plugins/panmind

Spinoffs from a large Rails Application

Why?

- Code you write in `/app` will be obsolete **soon**
- Code you write in `/lib` will be obsolete less soon (**maybe**)
- Code you you share with the Open Source community **could** live really long and beyond your expectations

- It’s a sane engineering principle to write **reusable** code
- Sharing the code makes you write **good documentation**

How?

- `gem uninstall -f copy-paste ide-tools`
- Abstract early, abstract often
- Create temporary `modules` in your models, helpers, controllers
- Move those `modules` away in `/lib`[1] – **SOON**
- Decouple ‘em from the app assumptions, logic and configuration
- Move ‘em in `/vendor/plugins`

- [1]: Optional but **recommended**:

```
config.load_once_paths.push((Rails.root+'lib').to_s)
```

A real world example

http://github.com/Panmind/bigbro/commits/master

- Write code in `/lib`, include the rusty `module`(s) in your app
- Decouple configuration:

```
-       def account
-         Config[:id]
+       attr_accessor :account
+       def self.set(options = {})
+          self.account = options[:account]
```

- Remove initialization code and put it in `init.rb`
- Rename the module and move code around
- **Write documentation**
- Release (Git is your friend: `co`, `cherry-pick` and `rebase -i`)
- Present at a Ruby event so someone else will **write tests** ;-)

Compatibility checklist

- Ruby `1.9.1-p378`
- Rails `2.3.8`
- `rails_xss` plugin

- Patches to support older versions of Ruby/Rails and/or without the `rails_xss` plugin more than welcome! (it's just an `.html_safe` after all :-)

Release #1: SSLHelper – What

http://github.com/Panmind/ssl_helper

- require_ssl / ignore_ssl / refuse_ssl DSL for your controllers (simple wrap of a before_filter)
- Named route helpers (ssl_ / plain_ relatives) and test helpers (with_ / without_ssl, use_ / forget_ssl) generation

```
redirect_to ssl_login_url
<%= link_to "Sign up", ssl_signup_url %>
<% form_tag plain_search_url do %> ... <% end %>
```

```
without_ssl do
  get :show, :id => @project.id
  assert_redirected_to ssl_project_url(@project.id)
end
```

Release #1: SSLHelper – How

- Checks HTTPS / `X-Forwarded-Proto` variables via Rails' `request.ssl?`
- Includes the controller DSL straight into `ActionController::Base`
- Inserts into Rails' router initialization by extending `ActionController::Routing::Routes` and overriding the `reload!` method (`returning super do ... end`)
- Generates `ssl_` and `plain_` wrappers of every named route helper defined in your app and puts them into an anonymous `Module`
- Includes it in `ActionView::Base` and in `ActionController::{Base,Integration::Session,TestCase}`

Release #2: BigBro – What

http://github.com/Panmind/bigbro

- Google Analytics -- let's get it straight (and async)
- Optimizes GA's protocol check (it's http://www. or https://ssl.?)
- Generates `<noscript>` tracking code
- Contains an embryo of a jQuery GA toolkit (in the `js/` directory)

`<%= analytics %>` **or**

`<%= analytics :track => false %>`

```
context “an user”
  should “be aware to live in 1984”
     get :index
     assert_analytics
  end
end
```

panmind ALPHA SOCIAL KNOWLEDGE

Release #2: BigBro – How

- A submodule contains view helpers, another one test helpers
- The top-level module singleton class holds the initialization method and the GA account into an instance variable:

```
class << self
  attr_accessor :account
  def set(options = {}) ... end
end
```

- View helpers are included in `ActionView::Base`
- Test helpers are included in `ActionController::TestCase`

...**Whoops**, the plugin currently adds 'em in `ActiveSupport::TestCase`! who'll be the first to send out a pull request? :-)

Release #3: ReCaptcha – What

http://github.com/Panmind/recaptcha

- Embeds ReCaptcha JS / Generates `<noscript>` code
- Provides a `require_valid_captcha` controller class method
- Chats with ReCaptcha HTTP service – handling timeouts
- AJAX validation via a custom jQuery plugin (untied to this one)

```
<%= recaptcha :label => 'Human?', :theme => 'clean' %>
require_valid_captcha :only => :create
def invalid_captcha
  @user.errors.add_to_base('Captcha failed')
  render :new, :layout => 'login'
end
```

Release #3: ReCaptcha - Test

Using mocha -- gem `install` it if you don't have it

```
context ‘a guest’ do
  should ‘insert a valid captcha’
    mock_invalid_captcha
    post :signup, :email => ‘vjt@openssl.it’, ...
    assert_response :precondition_failed # 412

    mock_valid_captcha
    post :signup, :email => ‘vjt@openssl.it’, ...
    assert_redirected_to root_url # 302
  end
end
```

Release #3: ReCaptcha – How

- Controller, View and Test helpers live in separate modules
- The top-level module singleton class contains the initialization method and the ReCaptcha keys

```
class << self
  attr_accessor :private_key, :public_key, ...
  def set(options = {}) ... end
end
```

- Controller methods included in `ActionController::Base`, and `self.included()` adds the `require_valid_captcha` method
- View helpers are included in `ActionView::Base`
- Test helpers are included in `ActionController::TestCase`

Release #3: ReCaptcha - AJAX

- ReCaptchas can be validated only once
- The `jquery.ajax-validate` plugin calls a controller action (`Metal` is better) that returns different `HTTP` status codes
- If successful, a flag is saved in the `flash`
- When the form is submitted, if the flag is true, ReCaptcha validation is skipped
- Unless your session cookies can be tampered with, the code is not vulnerable to replay attacks
- Older versions used a DB table first, memcached after.. but the `flash` is the best choice -- see the commit history for details :-)

Release #4: Zendesk – What

http://github.com/Panmind/zendesk

- Zendesk? The best support platform / CRM in town
- View helpers to generate the trendy "feedback" button code --->
  and to generate links that display the feedback form
- Route and controller methods to implement Zendesk's remote authentication: your users won't have to register and log in on the support forum
- View helpers to generate links to the support forum

```
<%= zendesk_dropbox_tags %>
<%= zendesk_link_to 'Support' %>
map.zendesk '/support', :controller => :sessions
```

panmind ALPHA SOCIAL KNOWLEDGE

Release #4: Zendesk – How

- View helpers are included into `ActionView::Base`
- The route generation method is included into `ActionController::Routing::RouteSet::Mapper`
- This time, **you** have to `include` the controller methods into your login controller: in development mode they would be lost because of `ActiveSupport`'s reloading (solutions welcome!)

- Too much configuration is needed to make it work; your login action must implement a `redirect_to params[:return_to]` -- does anyone want to help?

Release #4: Zendesk – Flow

1. Guest clicks on `zendesk_link_to( ‘Support’)`
2. Guest is taken to the support forum
3. Guest clicks ‘login’ in the support forum
4. Guest is redirected to the login page by the `zendesk_handle_guests` filter of the `zendesk_login` action
5. User is redirected to the `zendesk_login` action
6. User is redirected to zendesk’s remote authentication endpoint with a set of query string parameters (hash, timestamp, ...) and logged in

----

1. User clicks on `zendesk_link_to( ‘Support’)`
2. `GOTO 5`

Release #5: Leaker

Don’t use this plugin.

It’s an example of how plugins can be evil. Even if written elegantly.

If you’re really curious why you shouldn’t, read the documentation on GitHub – http://github.com/Panmind/leaker

You have been warned :-p

Where is the live demo?

SSLHelper: `curl -I http://panmind.org/login` -> 301

BigBro: have a look at the source of any panmind.org page, and search for `ga.js`

ReCaptcha: https://panmind.org/signup input wrong data first and then sign up

Zendesk: try the “Support” link in the footer and the “feedback” -> button on the right of every page – both before and after logging in

FEEDBACK

Leaker: no way! :-)

Thank you! :-)

@vjt - vjt@openssl.it

http://panmind.org/