!SLIDE shout

Cells & Apotomo

!SLIDE shout

Cells

!SLIDE

Cells

> Components for Rails.

Qu'est-ce qu'un composant ? Exemple canonique : *un panier d'achat* .

- M. **Product Owner** dit : le Panier d'Achat
- M. **Framework** dit : un controlleur + une vue
- M. **Programmeur** dit : controlleur + filtres + partials + helpers + …
- …

!SLIDE bullets incremental

Cells

- Axé sur la notion de composant fonctionnel.
- Composants réutilisables.
- Composition de composants.

⟹ cellule

!SLIDE

Cells — une cellule…

- permet de modéliser un composant fonctionnel
- ressemble à un (mini-)controlleur
- sera au final rendue sous la forme d'une chaîne de caractères
- peut être rendue n'importe où dans un controlleur ou une vue
- peut être mixée avec une autre cellule (~~DoubleRenderError~~)
- peut être mise en cache et testée indépendament des autres
- remet la POO au cœur du framework :)

!SLIDE

Cells — un exemple

```
$ rails generate cell cart display -e haml
  create  app/cells/
  create  app/cells/cart
  create  app/cells/cart_cell.rb
  create  app/cells/cart/display.html.haml
  create  test/cells/cart_test.rb
```

!SLIDE

Cells – app/cells/cart_cell.rb

```
class CartCell < Cell::Rails
  def display(args)
    user   = args[:user]
    @items = user.items_in_cart
    render # renders display.html.haml
  end
end
```

!SLIDE

Cells — app/cells/cart/display.html.haml

```
#cart
  You have #{@items.size} items in your shopping cart.
```

!SLIDE

Cells — layouts/application.html.erb & …

```
<%= render_cell :cart, :display,
                :user => @current_user %>
```

!SLIDE

Cells — bonnes pratiques

- Dependency injection
- Éviter les partials
- Utiliser l'héritage de vues
- Déléguer avec les Builders
- Penser à mettre en cache

!SLIDE

Cells

- **github.com/apotonick/cells**
- github.com/apotonick/cells_examples
- github.com/apotonick/cells-blog-example

!SLIDE shout

Apotomo

!SLIDE

Apotomo

> **Web** Components for Rails.

- Construit autour de la notion de Widget.
- Widget = Cell + states + events

Ex. canonique : *le bloc de commentaires*.

!SLIDE

Apotomo — un exemple

```
$ rails generate apotomo:widget CommentsWidget\
                display write -e haml
  create  app/widgets/
  create  app/widgets/comments_widget
  create  app/widgets/comments_widget.rb
  create  app/widgets/comments_widget/display.html.haml
  create  app/widgets/comments_widget/write.html.haml
  create  test/widgets/comments_widget_test.rb
```

!SLIDE

Apotomo — app/widgets/comments_widget.rb (1/2)

```
class CommentsWidget < Apotomo::Widget
  responds_to_event :submit, :with => :write!

  def display(args)
    @comments = args[:post].comments
    render
  end
```

!SLIDE

Apotomo — app/widgets/comments_widget.rb (2/2)

```
  def write!(evt)
    @comment = Comment.new(:post_id => evt[:post_id])
    @comment.update_attributes evt[:comment]

    update :state => :display
  end
end
```

!SLIDE

Apotomo — app/widgets/comments/display.haml

```
# let's talk about this later
```

!SLIDE

Apotomo — app/controllers/posts_controller.rb

```
class PostsController < ApplicationController
  include Apotomo::Rails::ControllerMethods
  has_widgets do |root|
    root << widget('comments_widget',
                   'post-comments',
                   :post => @post)
  end
end
```

!SLIDE

Apotomo - app/views/posts/show.haml

```
%h1 @post.title
%p
  @post.body
%p
  = render_widget 'post-comments'
```

!SLIDE

Apotomo – app/widgets/comments/display.haml (1/2)

```
= widget_div do
  - form_for :comment, @comment, :html => {
              "data-event-url" => url_for_event(:submit)
            } do |f|
    = f.error_messages
    = f.text_field :text
    = submit_tag "Don't be shy, comment!"
```

!SLIDE

Apotomo – app/widgets/comments/display.haml (2/2)

```
:javascript
  var form = $("##{widget_id} form");
  form.submit(function() {
    $.ajax({url: form.attr("data-event-url"),
            data: form.serialize()})
    return false;
  });
```

!SLIDE

Apotomo — testing

```
class CommentsWidgetTest < Apotomo::TestCase
  has_widgets { |root| ... }

  def test_render
    render_widget 'me'; assert_select "li#me"
    trigger :submit, :comment => {:text => ""}
    assert_response 'blabla'
  end
end
```

!SLIDE shout

Atypic / CDGP

!SLIDE

Atypic / CDGP

Le besoin : une API de widgets (*blocks*) en cross-domain

Notre solution : un backend Apotomo adapté et du JSONP

- Tout CDGP, certaines parties d'Atypic seulement (coffre-fort…)
- Loves Backbone : composants visuels == fonctionnels

!SLIDE

Atypic / CDGP — API (1/3)

```
class Widgets::DispatcherController < ApplicationController
  # rappel : exécuté dans le contexte d'une requête
  has_widgets do |root|
    root << widget("block_widgets/#{params[:block_type]}",
                   params[:source] || widget_uuid,
                   params)
  end
```

!SLIDE

Atypic / CDGP — API (2/3)

```
def index
  res = render_widget(widget_uuid, :display)
  render_widget!(res)
end
def render_event_response
  res = apotomo_request_processor.process_for(params).first
  render_widget!(res)
end
```

!SLIDE

Atypic / CDGP — API (3/3)

```
def render_widget!(res)
  # take content, status, metadata from res
  respond_to do |format|
    format.html { render :inline => content,
                         :status => status}
    format.json { render :json => {
                           # smart things here
                         }.to_json,
                         :callback => params[:callback] }
```

!SLIDE

Atypic / CDGP – Custom Apotomo backend (1/4)

```
class RemoteWidget < Apotomo::Widget
  include ActiveSupport::Rescuable
  include ApplicationHelper
  include WidgetsHelper
  helper ApplicationHelper
  helper Widgets::CssHelper
  helper Widgets::SocializeHelper
  before_filter :setup!
```

!SLIDE

Atypic / CDGP — Custom Apotomo backend (2/4)

```
def render(*args, &block)
```

```ruby
    # if    ok?         render_ok
    # elsif no_content? render_no_content
    # elsif error?      render_error
    # else  super
  end
```

!SLIDE

Atypic / CDGP — Custom Apotomo backend (3/4)

```ruby
  def render_ok(options = {})
    escape = options.delete :escape
    view = render(:view => @state, :locals => options)
    view = CGI::escapeHTML(view) if escape
    {:content => view,
     :status  => :ok,
     :metadata => @metadata}
  end
```

!SLIDE

Atypic / CDGP — Custom Apotomo backend (4/4)

- FormWidget
- widgets.js

!SLIDE

Apotomo++

- Du code concis, réutilisable
- Bubbling events
- Composition de widgets
- Simple d'utilisation en équipe
- ✔ Test/Unit / MiniTest

!SLIDE

Apotomo

- **github/apotonick/apotomo**
- apotomo.de
- screencasts

!SLIDE shout

:)