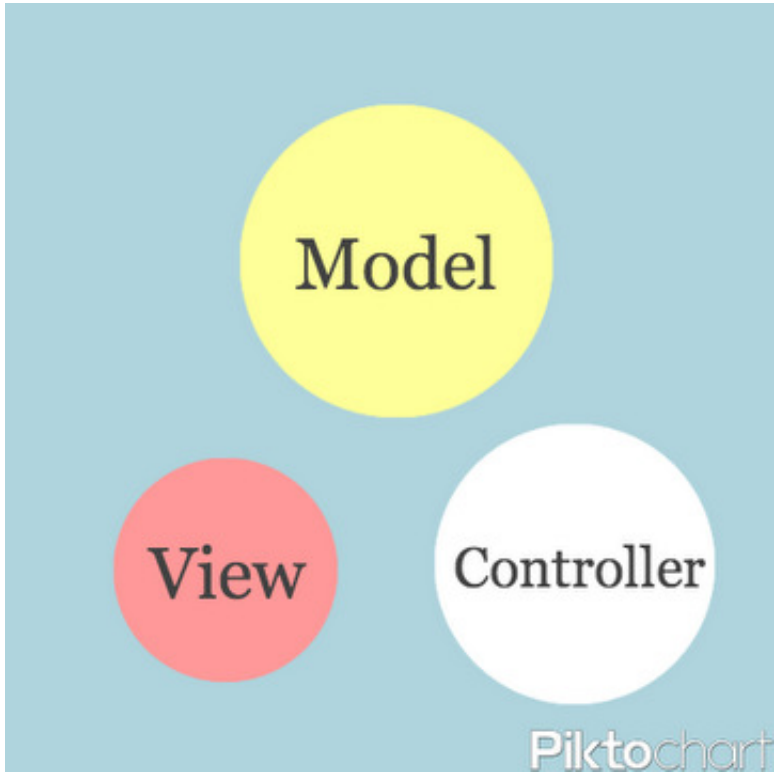
Single page apps & MVC

Irina Dumitrascu

ruby & cofeescript | dira.ro , @dira_geek_girl

February 28, 2013

MVC sounds simple



- **M**odel - store & manage data

- **V**iew - display
- **C**ontroller - manage the user interaction

Model

- store data
- deal with persistence
  - serverside, local storage, etc.
  - when to save

Model

be the one and only, reliable, complete source of data

(and operations on it)

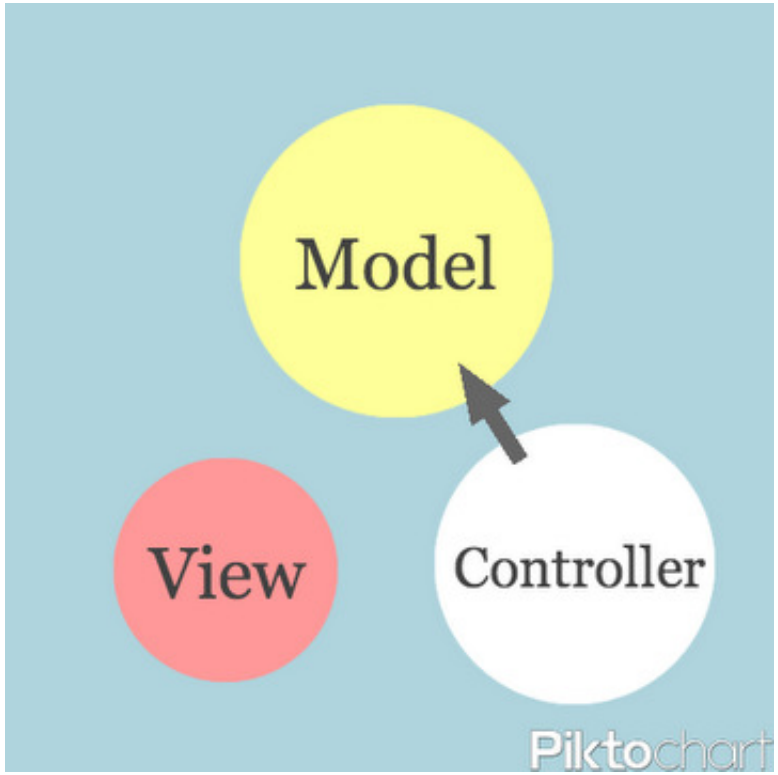
View

- display the data from the model in the DOM

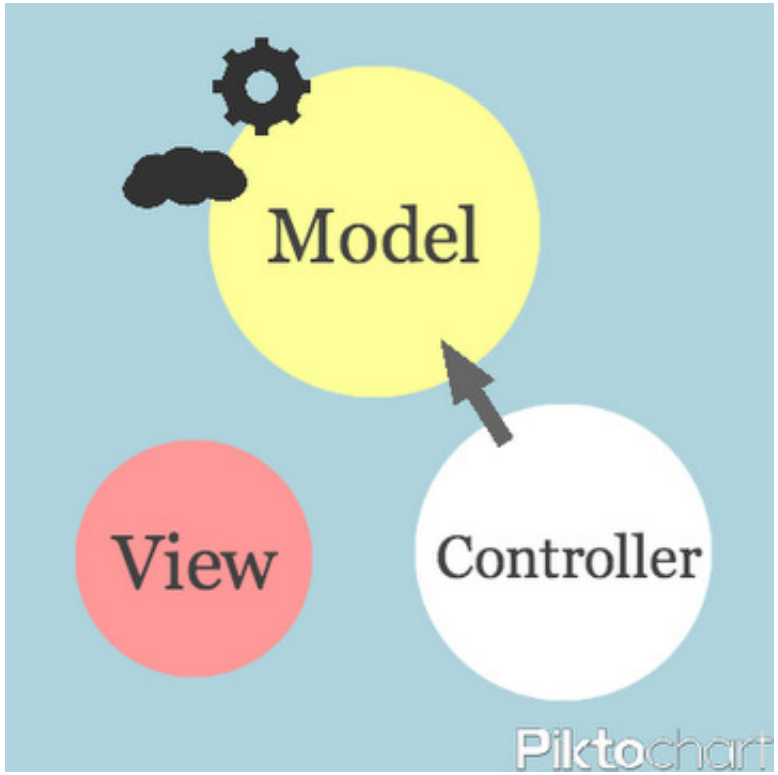
Controller

- manage the interaction with the user
- and pass the intended changes to the model

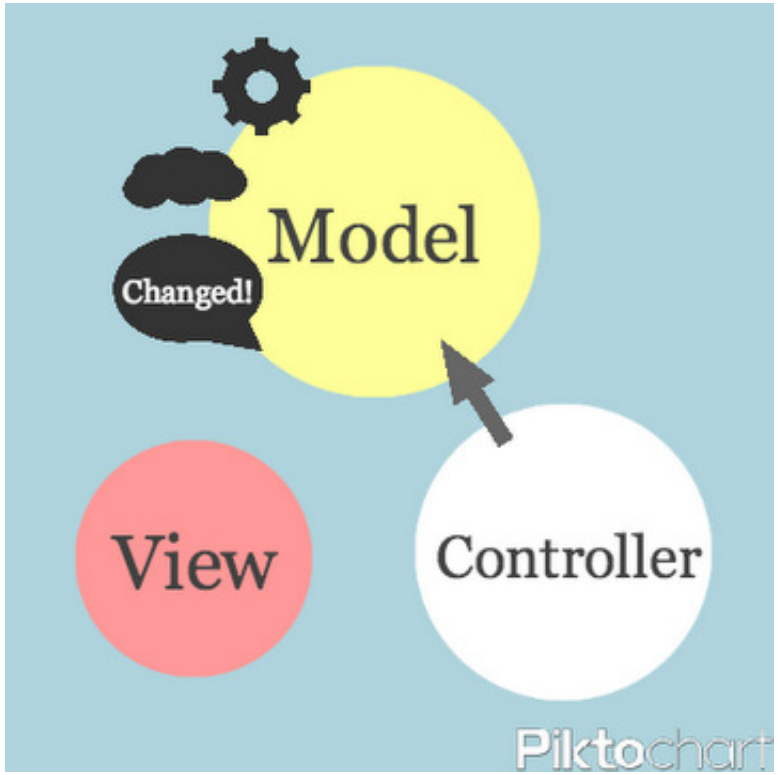
The MVC flow



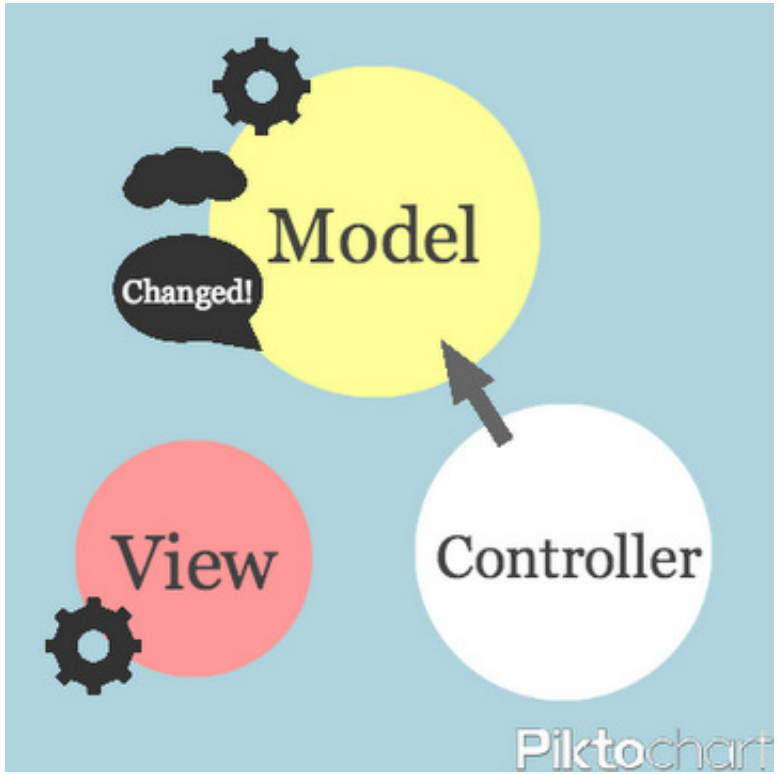
The MVC flow



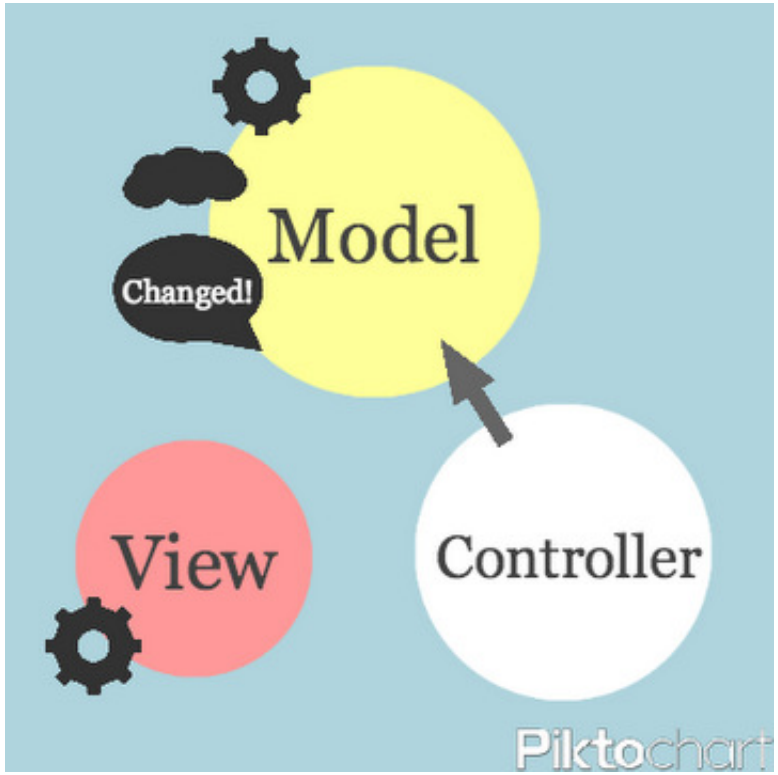
The MVC flow



The MVC flow



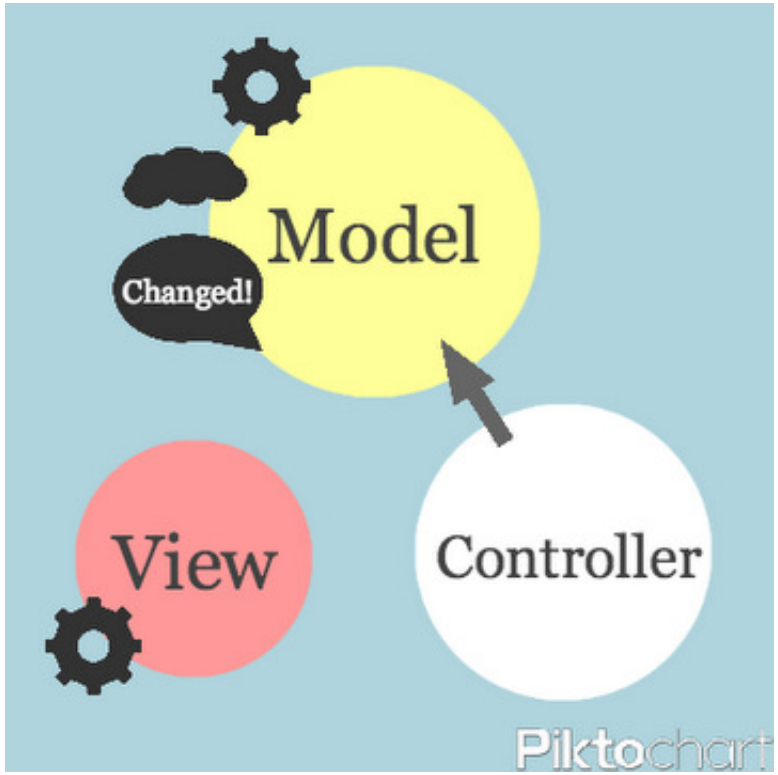
MVC relationships



- Controller updates Model

- View reacts to Model

Forbidden MVC relationships



- Model communicates directly to a View

- Model communicates directly to a Controller

Classic JS web app

- the data is stored in the DOM
- some JS code for managing interaction, persistence

Classic flow

- UI events =>
  - explicitly update all the places where this data appears in the DOM
  - other operations (e.g. persistence)

Classic relationships

- whatever makes sense to you
- it's common for the code that deals with UI to
  - update multiple views
  - update unrelated views

MVC

- provides structure
- different elements with a specific job, and reduced coupling
    - easier to unit test!

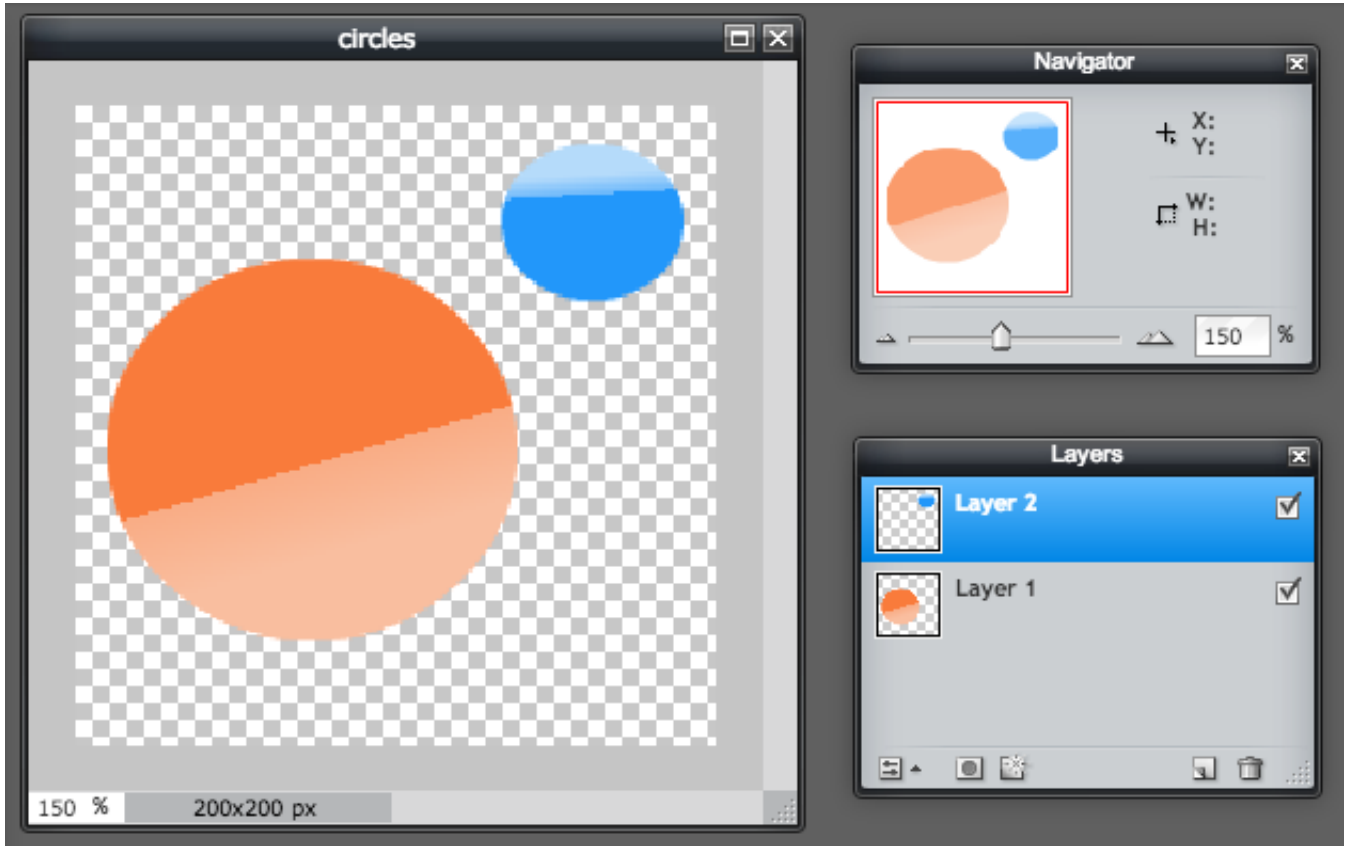
MVC shines

- when you have multiple views for the same data

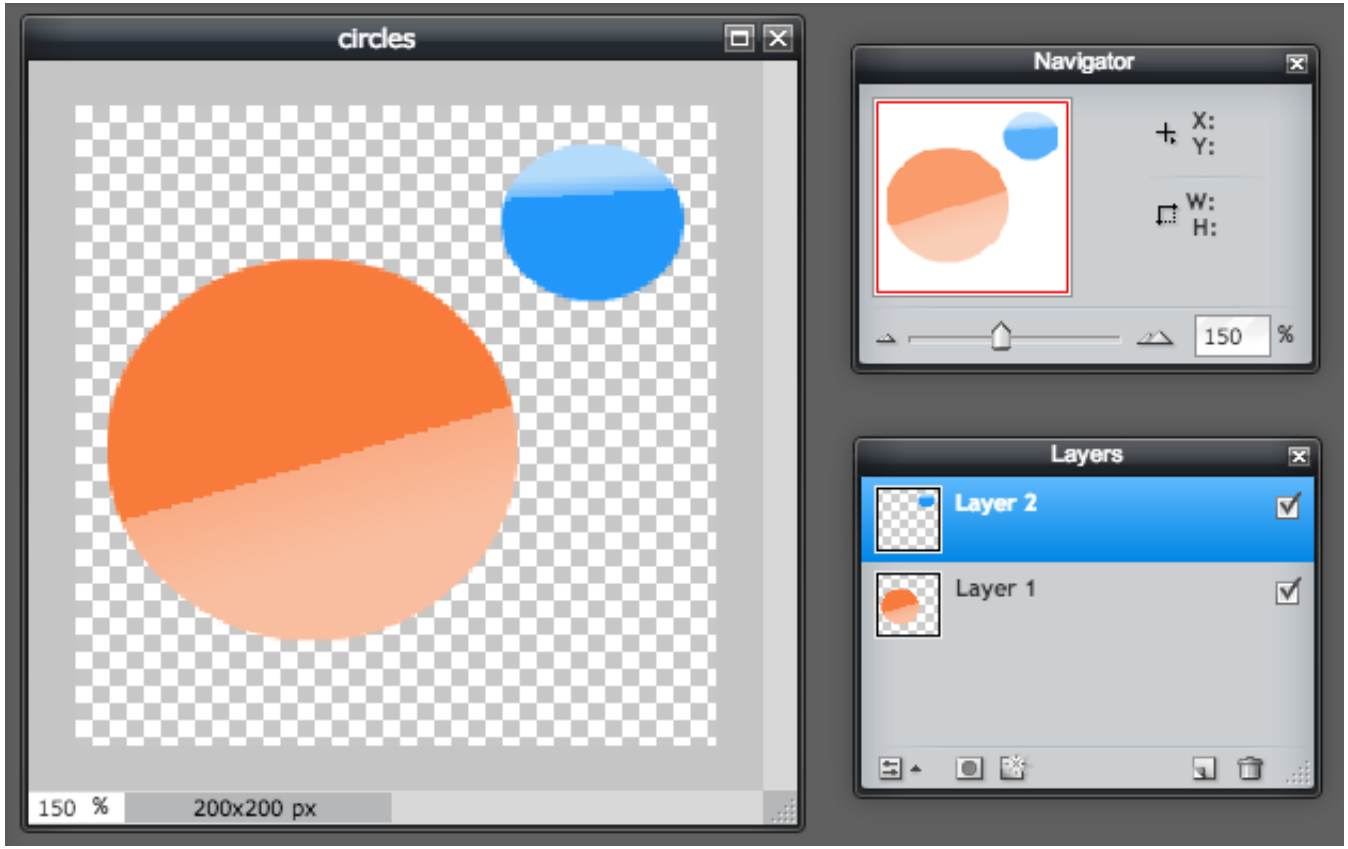
MVC shines

- when you have multiple views for the same data
- which is quite frequent

Multiple views on the same data



Multiple views on the same data



- any controller can change the models

- all the views update when the model changes

Added benefits

- when you add another view, it just listens to the relevant model and updates itself
- you don't change code in the models, or somewhere else to update the new view

Example - without MVC, one view

```
$('#scale').change =>
        scale = $(this).val()

        view.scale_to(scale)


$('#x').change =>
        x = $(this).val()

        view.set_x(x)
```

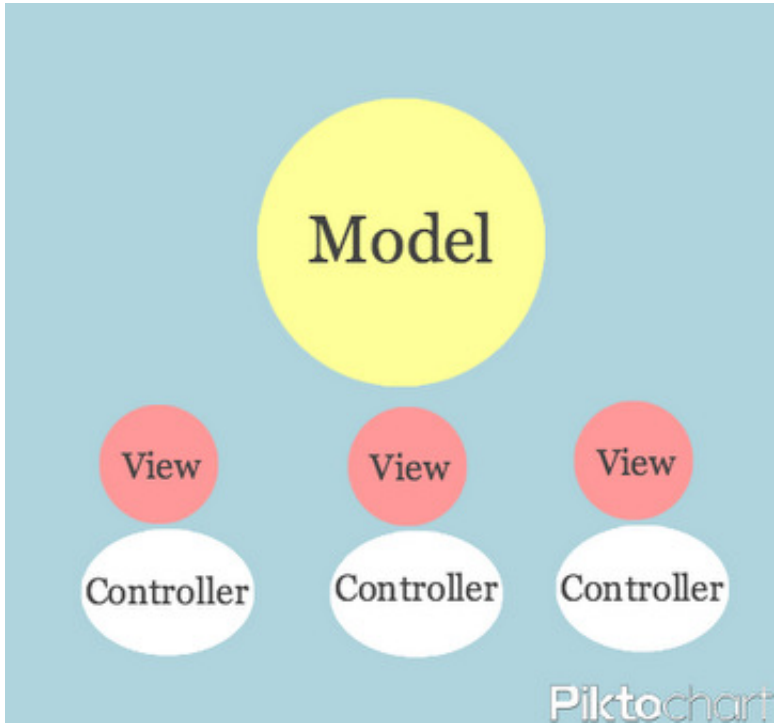
Example - without MVC, two views

```
$('#scale').change =>
        scale = $(this).val()

        view.scale_to(scale)
        another_view.scale_to(scale)

$('#x').change =>
        x = $(this).val()

        view.set_x(x)
        another_view.scale_to(scale)
```

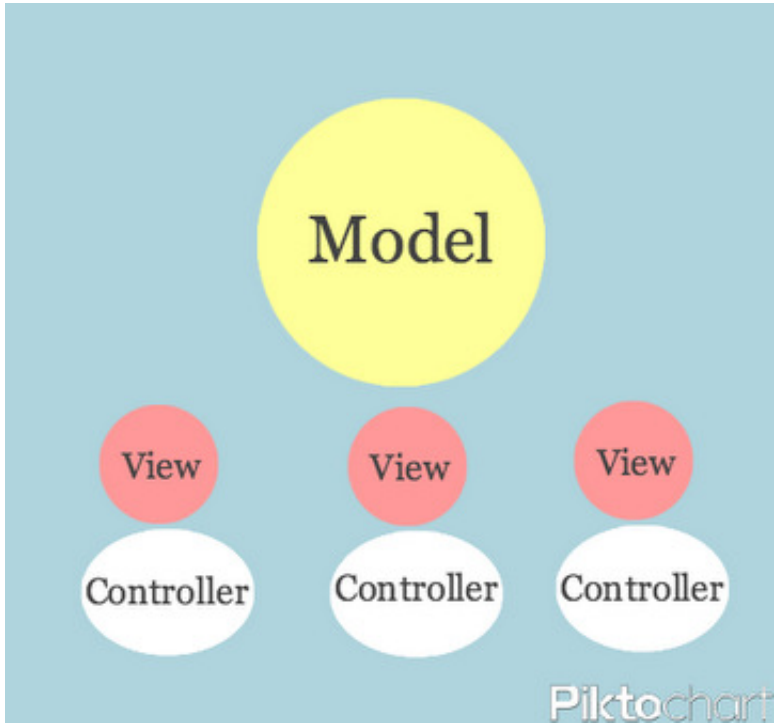
Example - with MVC, two views

```
# view #1
model.on 'change', @render

# view #2
model.on 'change', @render

# ...
```

So it's MVC-VC-VC



So it's MVC-VC-VC



or M(VC)*

When using MVC

it's tempting to sometimes skip the rules...

The VC that does not need a model

Scenario

A data item that

- is quite small
- has only one view

Scenario

- manage it in the view directly, without creating a model

What happens

- it grows
- you get too much model-related code in the VC

What happens

- you extract it

The controller wants to be a model

The controller wants to be a model

Also known as: 'Messing with data in the controller'

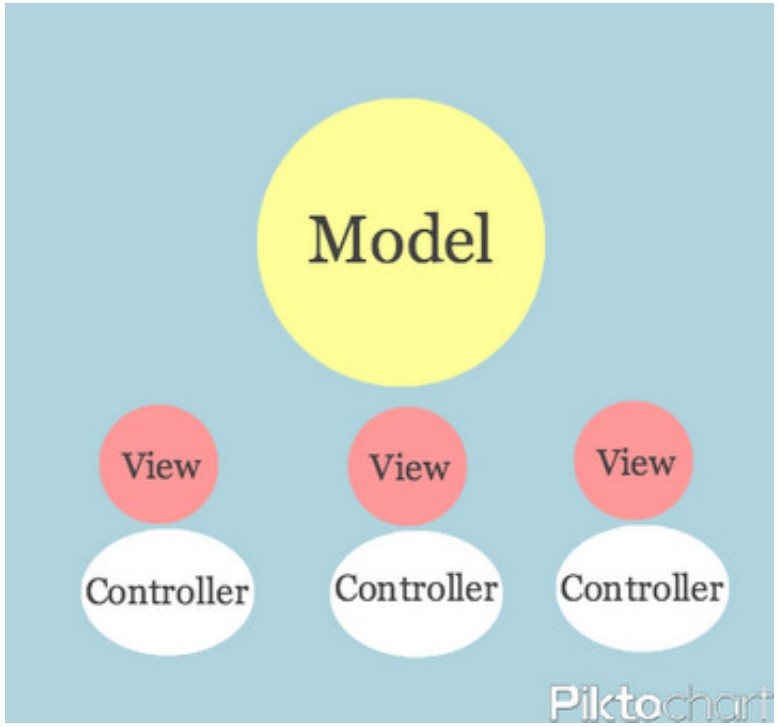
Scenario

- integrate a widget, let's say Sortable from jQuery UI
- on the completion event handler, forget about the model and just `post` the new order to the server on the spot

```
$('#list').sortable
  update: =>
    $.post '/positions', @getPositions()
```

What happens

- the model is not the source of accurate data anymore
- if another view needs to take the order into account, it can not


Model
View
View
View
Controller
Controller
Controller
Piktochart

Summary

Code smell: model-specific tasks (e.g. persistence, manipulating data) in the controller

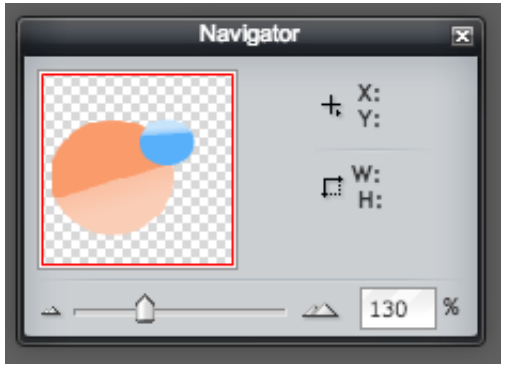
Fix:

```
$('#list').sortable
  update: =>
    model.setPositions @getPositions()
```

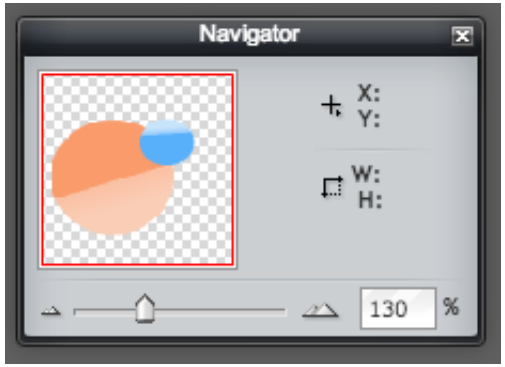
The view-controller that doesn't listen

Also known as: 'I think I know what the model will say!'

Scenario



Scenario




```
$('.scale .slider').change (e) ->
  scale = $(this).val()

  $('.scale .text').val(scale)
```

What happens

What about:

- validation?
- or 'snap to 5 pixels'?

Let the model be the source of wisdom about the data (and only react to it)

Code smell

update a view directly from a UI event handler

```
ui_element.change ->
  other_ui_element.val(value)
```

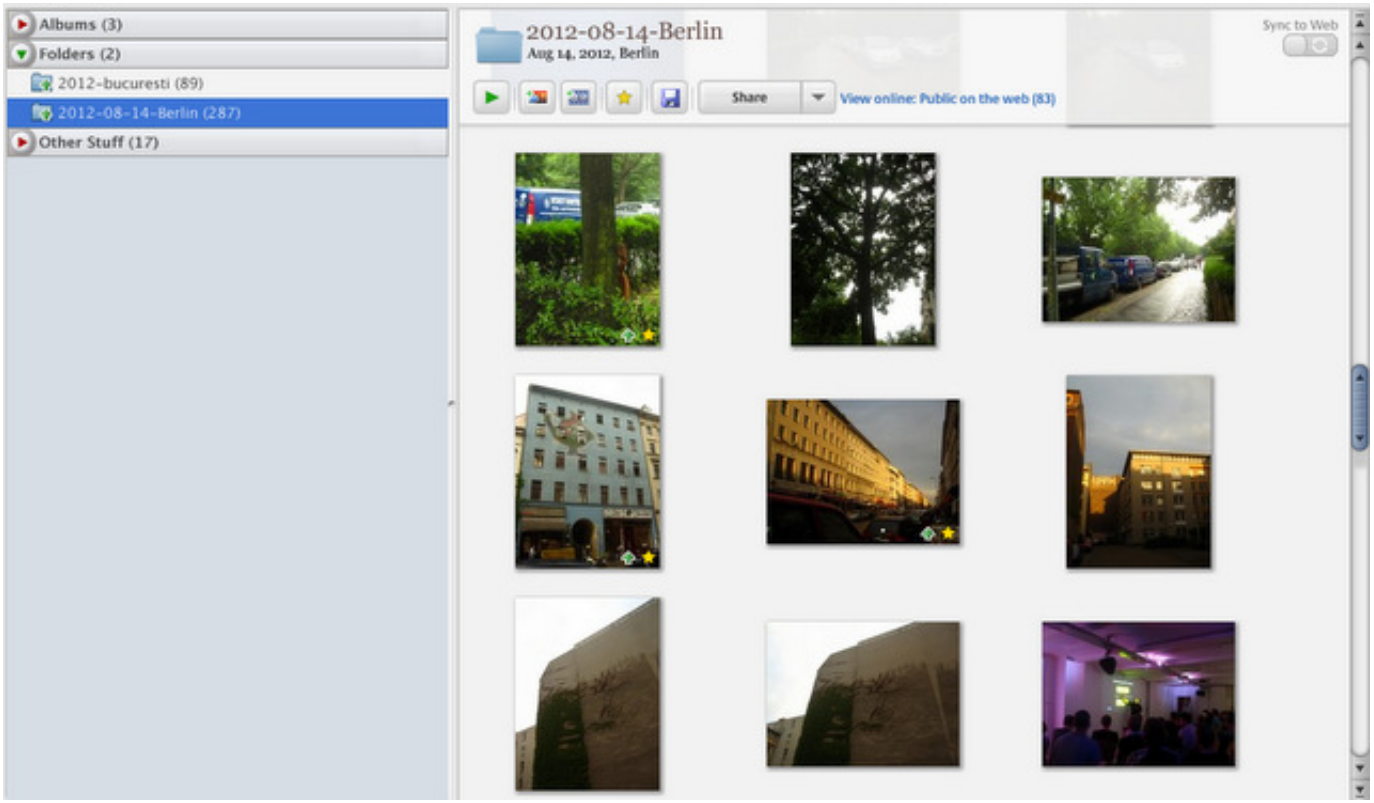
How it should look like

```
ui_element.change (value) ->
  model.set(value)

model.change (value) ->
  ui_element.set(value)
```

The model that gets no trust

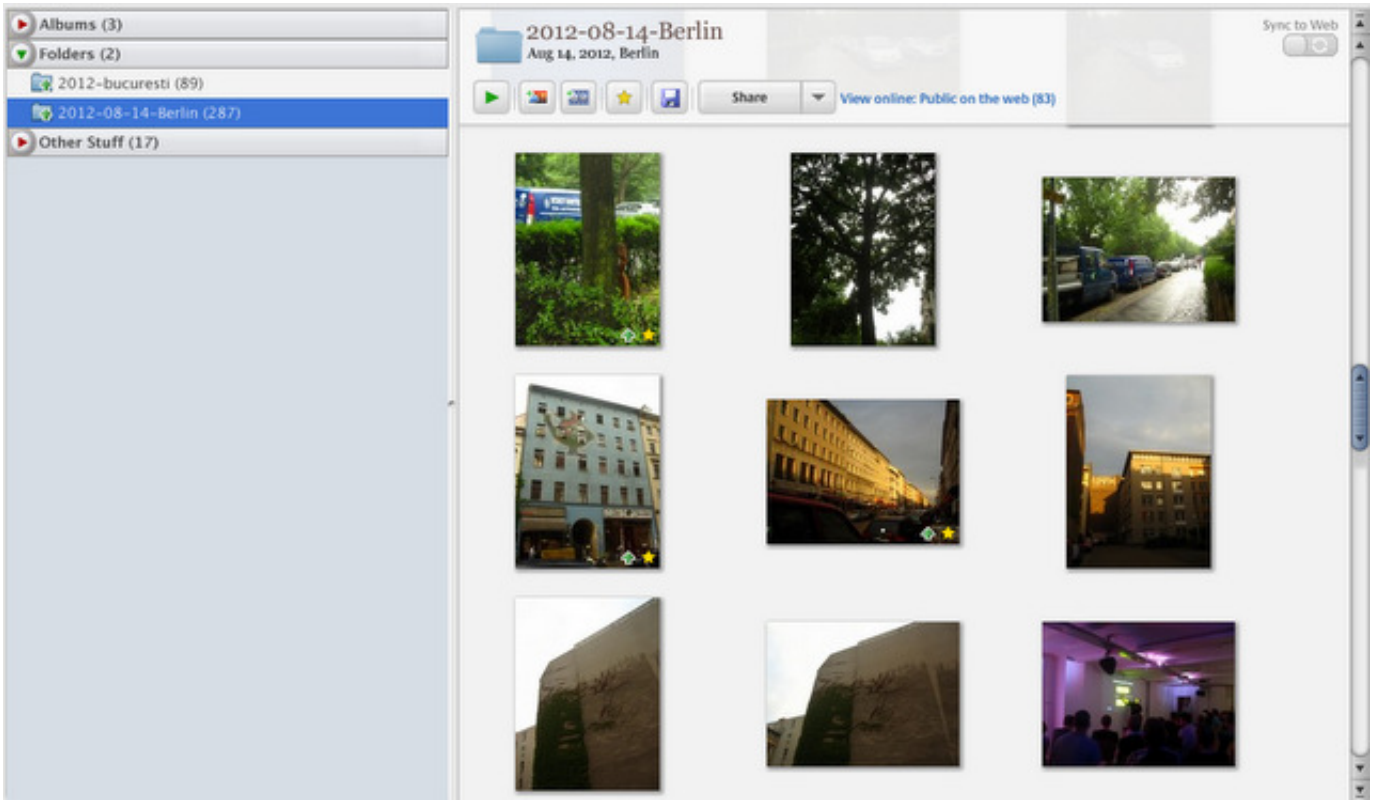
Also known as: using information from controllers or views, in the model

Scenario




```
$('.album').click (e) ->
  window.album_id = $(e.currentTarget).data('id')
```

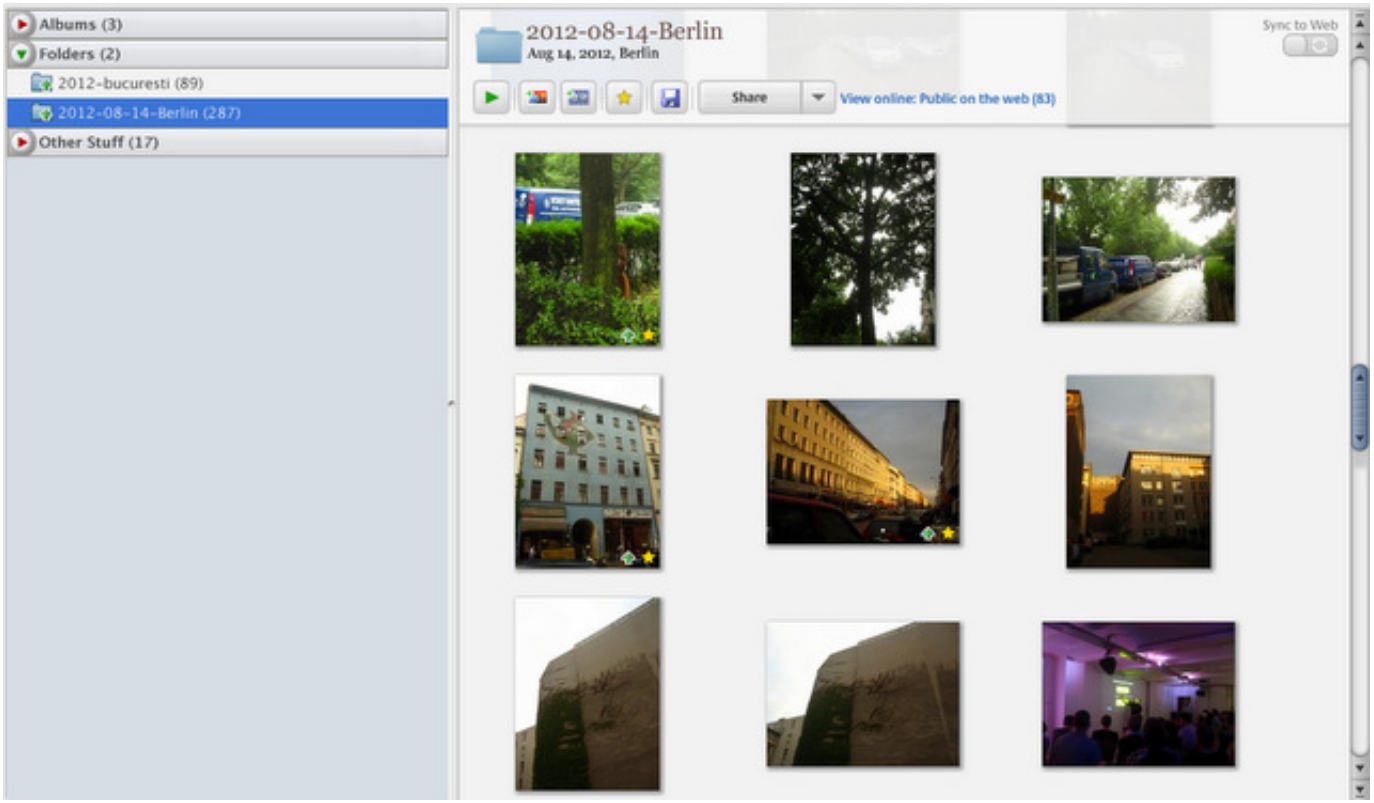
Scenario



- update an image

- reasoning:
  - the UI was shown -> so the image was shown
  - so it belongs to the current album!

Scenario




```
$.post("/albums/" + window.album_id + "/images" + id)
```

Conclusion

MVC is

- simple
- powerful
- it's tempting to circumvent it
- but it's worth it sticking to itG

Thank you!

Questions?