Testing Your Archive

Delivering on the Promise of Persistence

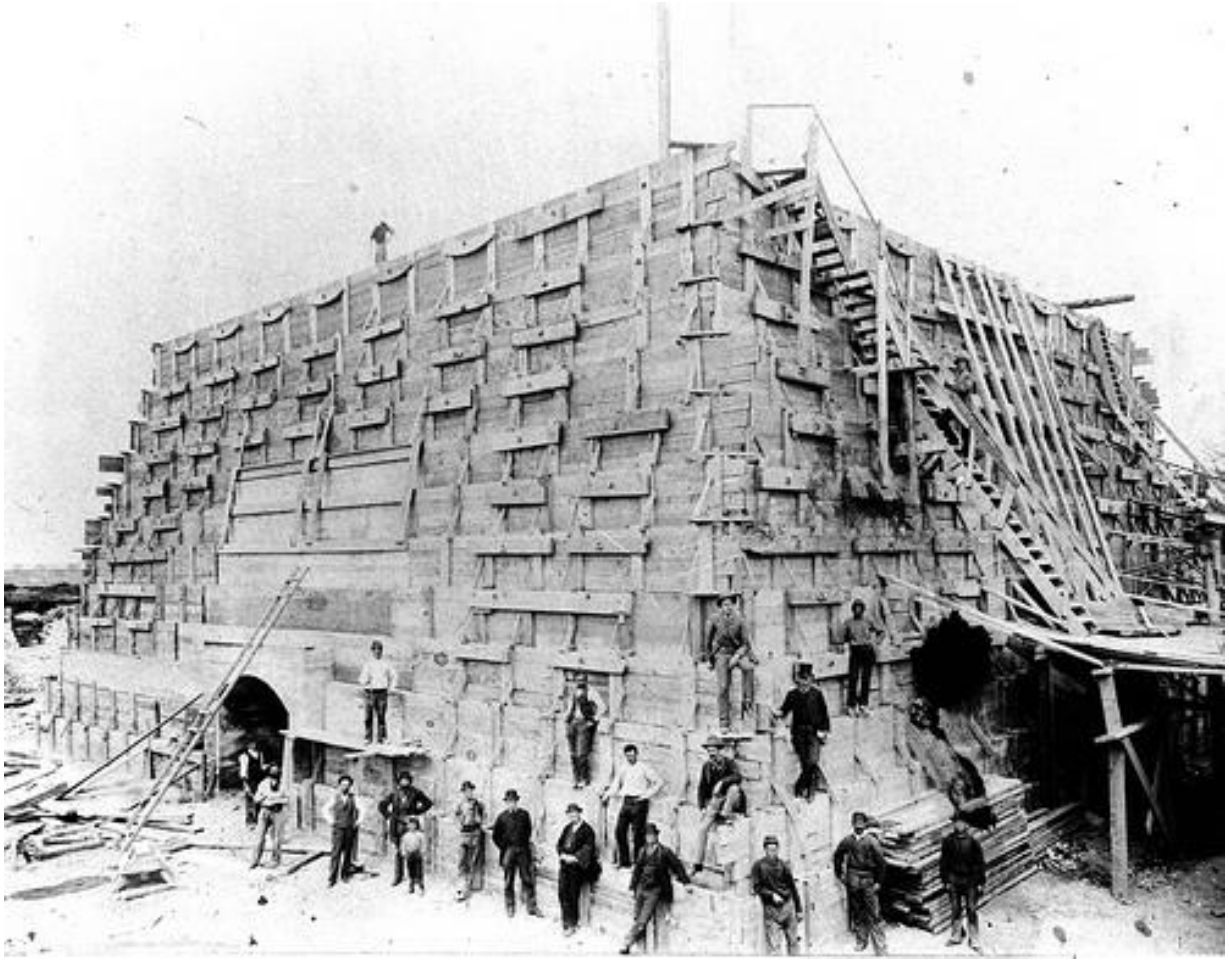
@JeremyFriesen

Deck URL @ bit.ly/12VuLIp

Our Goal

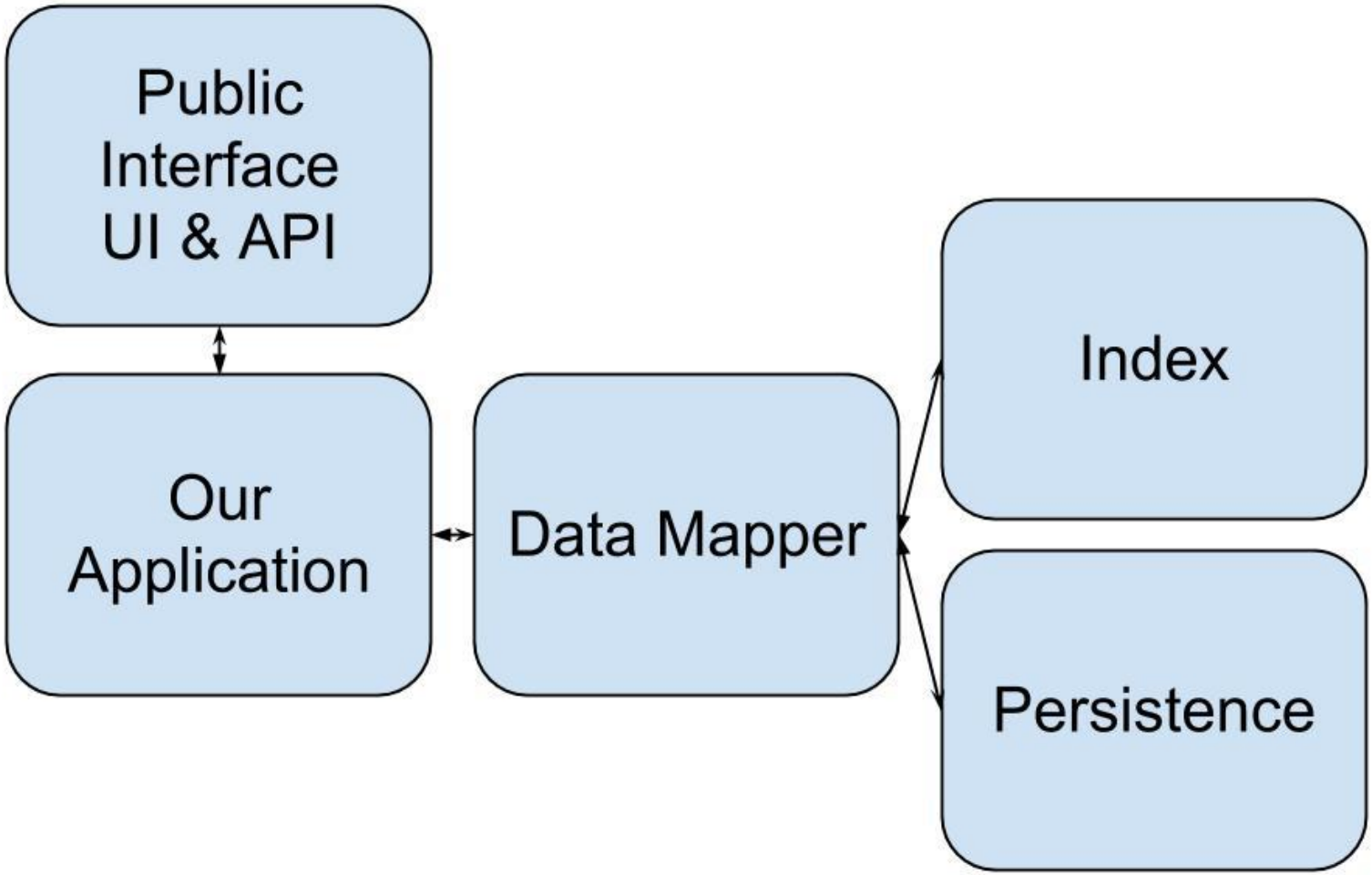
An object created yesterday and accessed today should continue to be that object.

Our Reality



The Repository Software Stack Is Ever Changing

Our Stack


Public
Interface
UI & API
Our
Application
Data Mapper
Index
Persistence

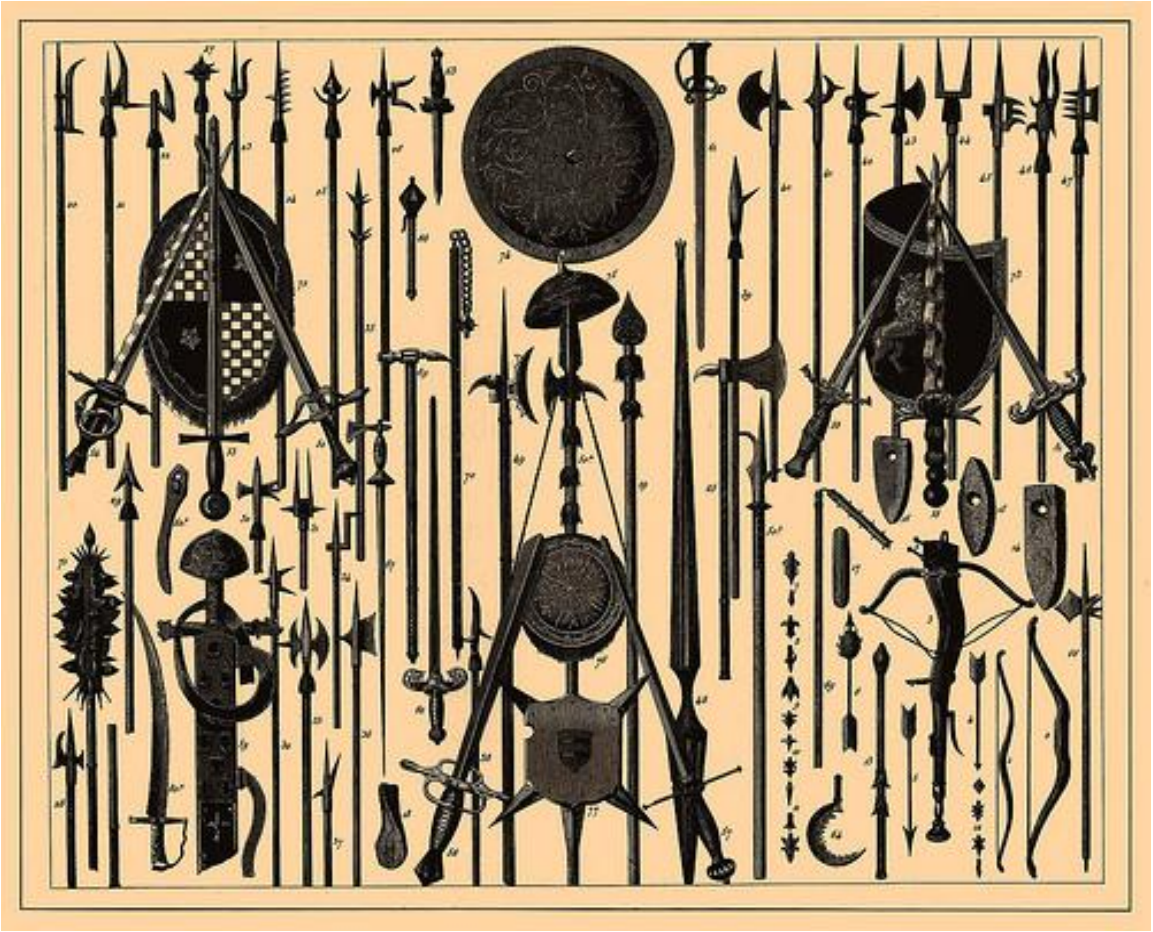
Our Weapons

Testing

Testing

And more

Testing!



"From the Brockhaus and Efron Encyclopedic Dictionary" by Mary Margaret @ http://flic.kr/p/8biHru [CC-BY]

Our First Test

When I create it

Then it is stored as expected

Our Second Test

When I didn't view it yesterday

And don't view it today

Then it is the same

Our Third Test

When I view it today

Then it is not changed

Our Fourth Test

When I update it

Then it is changed as expected

"Is there any other way to prepay?" by William Clifford @ http://flic.kr/p/6m28Zv [CC-BY]

Our Test Summary

When I:

* Create it
* Leave it
* View it
* Update it

Then I test it

Testing Hint

Each test should be for one thing.

Questions?

You want specifics?

A Real Weapon: Capybara

Have it do your testing

http://git.io/capybara

"Capybara" by Karoly Lorentey @http://flic.kr/p/9wcG1g [CC-BY]

Our First Test - Redux

When I create it, then it is stored as expected.

```
1  let(:title) { 'My Title' }
2  it 'is stored as expected when I create it' do
3    visit new_thing_path
4    within('#new_thing') do
5      fill_in('Title', with: title)
6      click_button('Create Thing')
7    end
8    pid = extract_pid_from_path(page.current_path)
9    expect(class: Thing, pid: pid).to(
10     have_valid_persistence(title: title)
11   )
12 end
```

Our Third Test - Redux

When I view it today, then it is not changed.

```
1  let(:title) { 'My Title' }
2  it 'is not changed when I view it' do
3    thing = Thing.create!(title: title)
4    expect{
5      visit thing_path(thing)
6    }.to_not change_persistence(thing)
7  end
```

Our Fourth Test - Redux

When I update it, then it is changed.

```
1  let(:title) { 'My Title' }
2  let(:new_title) { title + ' New' }
3  it 'is changed when I update it' do
4    thing = Thing.create!(title: title)
5    expect {
6      visit edit_thing_path(thing)
7      within('.edit_thing') do
8        fill_in('Title', with: new_title)
9        click_button('Update Thing')
10     end
11   }.to change_persistence(thing).
12   from(title: title).
13   to(title: new_title)
14 end
```

Our First Test - Redux^2

When I change a translation layer

And I create the object

Then it is stored as expected.

The first test as written hopefully should be caught if there is a problem.

Our Third Test - Redux^2

When I change a translation layer

And I view the object today

Then it is not changed.

This sounds like a case where you will want to clone a segment of the production persistence and index, and use that for a custom test.

After all Fedora Futures!

Our Fourth Test - Redux^2

When I change a translation layer

And I update the object

Then it is changed.

Again you will want a clone of your production system to run a custom test.

But What About Second Test?

When I didn't view it … then it is the same.

Run fixity checks against your repo. Log them?

"The Fellowship of the Rings" - New Line Cinema, used without permission

Hydra Capybara Walkthrough

"A Hydra-powered Rails project built with the goal of helping you write your first set of tests. You can follow the changes to the code by replaying the commit history."

Available @ http://git.io/MkzQyA

https://github.com/jeremyf/hydra-capybara-walkthrough

Capybara without a Rails/Rack App

"Normally Capybara expects to be testing an in-process Rack application, but you can also use it to talk to a web server running anywhere on the internet"

capybara @ http://git.io/_tilyg

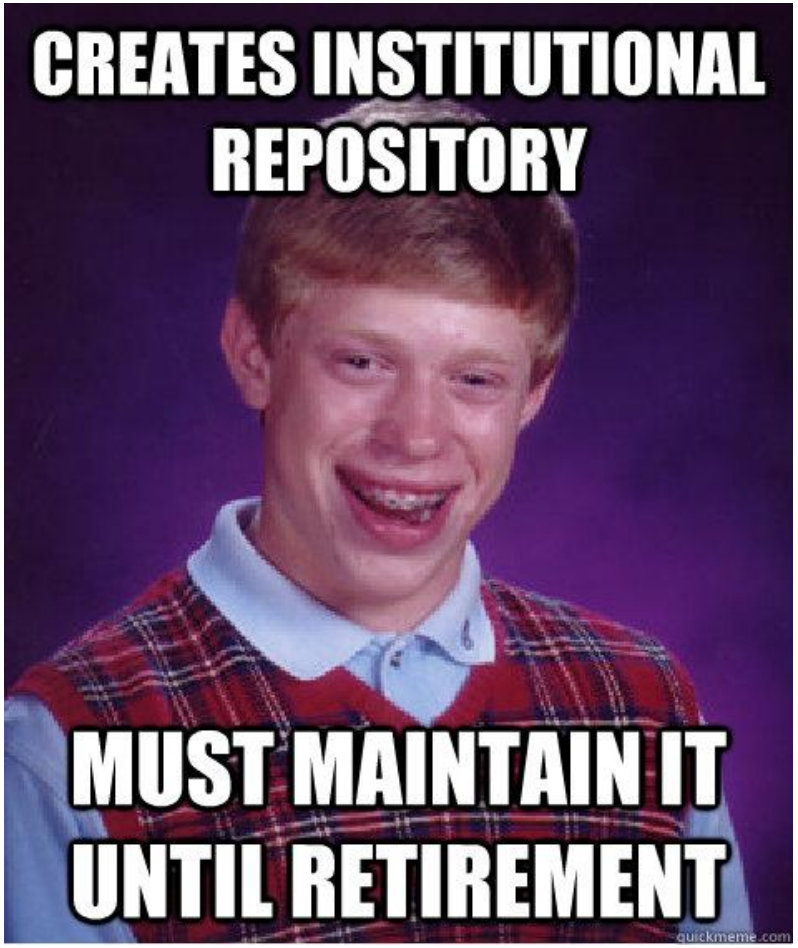
capybara-mechanize @ http://git.io/C4wIwQ

https://github.com/jnicklas/capybara#calling-remote-servers

Spend Time to Automate Your Tests

Remember, as a software developer, anything you do on a computer can almost certainly be automated.

If what you are doing today will be useful tomorrow…encode it!



Bad Luck Brian meme, created @ quickmeme.com

Thank You

Jeremy Friesen

Project Application Developer

Hesburgh Library

University of Notre Dame

Slide Available @ http://bit.ly/12VuLIp

@jeremyfriesen

ndlib.github.io - *a blog for helping Libraries code better*