Hello Minsk!

❤️💜💛💚💙

Hello Minsk!
When I was preparing for my first conference all my friends scared me with various myths about Minsk

I met amazing people and I'm really happy to be here.

Anton Davydov

github.com/davydovanton
twitter.com/anton_davydov
davydovanton.com

I'm very nervous, cause it's my first presentation in English and if I make any mistakes I apologize in advance

my name is Anton

💚 OpenSource 💚

I'm an opensource lover

- Sidekiq
- Hanami

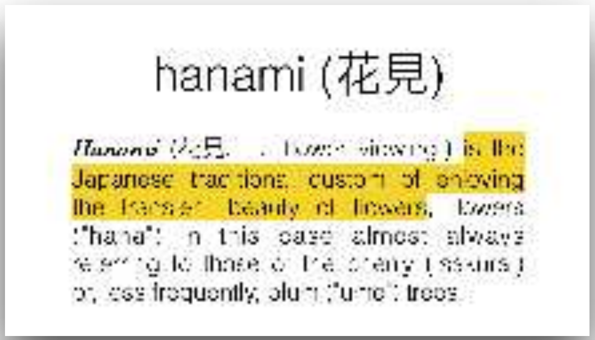
sidekiq and hanami commiter

- ruby doc
- rails
- crystal
- etc

Also you can find my commits in other big projects like ruby, rails, crystal, etc.

- moscow.rb
- rubyunderhood

Besides I am trying to improve my local user group. That's why I am making drynkups in moscow.rb and I'm a curator of Collective twitter account for ruby developers.



hanami (花見)

Hanami (花見, "flower viewing") is the Japanese traditional custom of enjoying the transient beauty of flowers, flowers ("hana") in this case almost always referring to those of the cherry ("sakura") or, less frequently, plum ("ume") trees.

There is a nice word in Japanese which sounds as “hanami”. It means “watching the flowers bloom” The most popular flower is sakura but other flowers such as tulips are watched as well.

There are many books and images illustrating this process . I found this one.

As you might understand today we aren’t going to watch blooming sakura , although the weather is perfect outside.
I’m going to talk about ruby web framefork which is called hanami.

This framework was written by developer from Italy, his name is Luca. The first commit was created three years ago. As you can see, hanami is a relatively new framework

Alfonso Uceda

[illegible]

Trung Lê

[illegible]

Core team consists of two people besides Luca.

At this moment only few companies used this framefork in production. I found five companies and two of them - are Russian companies.

The Base Ideas

Why is that framework getting more and more popular?

Let's talk about the basic ideas

Modularity

[illegible]

the first idea is modularity
It enables you to you to switch code and framework parts.

Do you want to change model to AR or ROM?
No problem, it’s easy. Do you think that there is too much of hanami ? You can use just routes.

Simplicity

[illegible]

Simplicity. If you use simple tool you can start working on production application faster.
I want to ask few questions:

- Who read at least one book about sinatra or grape?
- about rails?
- And who read more than five books about rails?

I think I read five different books about rails.
Framework is just a tool, don't make a cult of it

Less DSLs

DSL is rather a controversial approach. And Martin Fowler has great posts about this.

We all love configuration DSL. But if you write DSL code using business logic, you have a big problem.

Few Conventions

few conventionals:
If you are going to use hanami, you gain more freedom. You don't need to think how you can mix your application and framework conventional. You're like a Samurai who chooses his own path

Pure Objects

[illegible]

the next idea is using pure objects. I think everything on this slide is clear

Zero Monkey-Patching

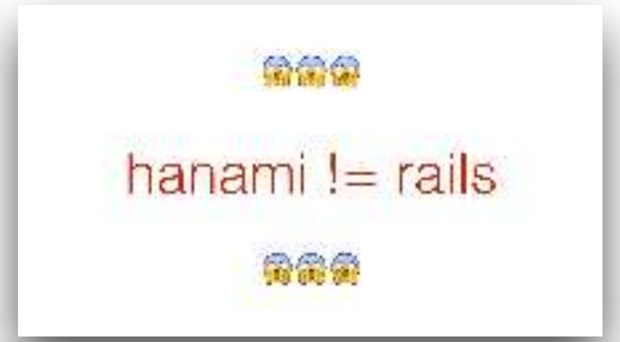
[illegible]

Zero Monkey-Patching
Don’t think whether this method comes from framework or language

Threadsafe

Using Tread safe you don’t have to worry about parallel computing

the most important part of my talk is in the next slide. Please, pay attention to it.


😱😱😱

hanami != rails

😱😱😱

hanami is not rails.
hanami is not rails killer.
And comparison of these frameworks is a stupid idea.
But we'll compare them later

Typical parts of web app

Let's start with simple things: all web applications contain two different parts:

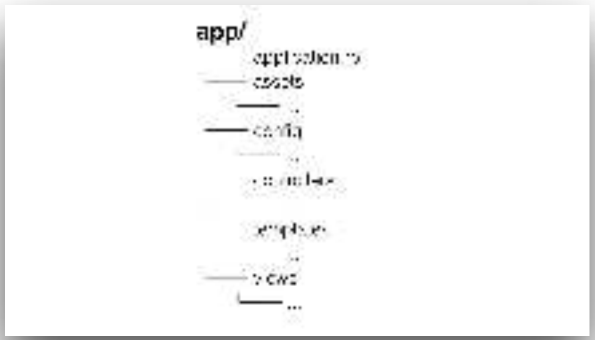
Web App
Business logic
Data flow

business logic and data flow That's why it's normal to split this parts in your application. I'll start talking about a data flow

Application Architecture

Monolith First

Let's refresh in the memory what great people say. They recommend a monolith- first strategy. And hanami has a simple way to create monolith apps. This way is called Application Architecture. It looks like a typical rails application



```
app/
    application.js
──── assets
      ──── ...
──── config
      ──── ...
    controllers

    templates
          ...
──── views
      └──── ...
```

As you can see, app folder has 3 different parts.

```
app/
    application.js
  ─── assets
      ─── ...
  ─── config
      controllers
      templates
  ─── views
      ─── ...
```

The first part is an application configuration

```
app/
    application.rb
  ├── assets
  │   └── ...
  ├── config
  │   └── ...
    controllers
    [illegible]
  ├── views
      └── ...
```

The next part are controllers and routes

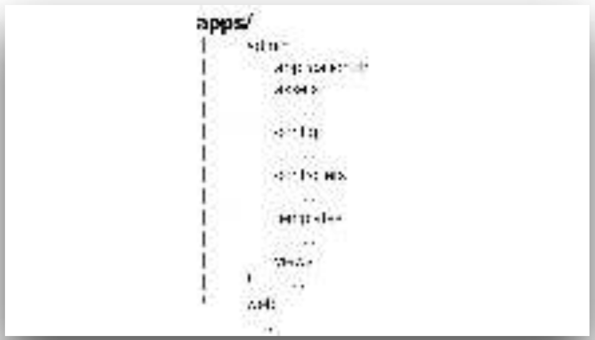
```
app/
    [illegible]
├── [illegible]
│   └── ..
├── [illegible]
    [illegible]
    [illegible]
├── [illegible]
│   └── ..
└── [illegible]
    └── ...
```

and the last part is responsible for displaying your data

Container Architecture

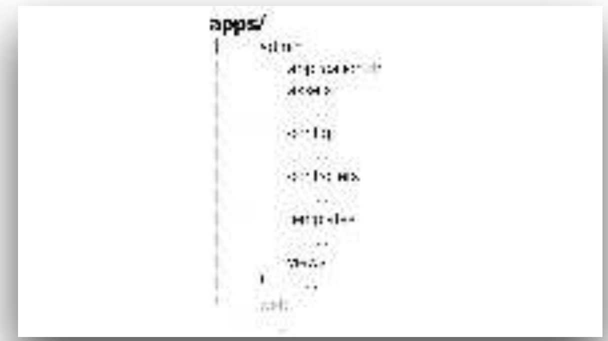
Clear Architecture

After that, your startup will make money and ,of course, you'll want to rewrite all you code to microservices.
And hanami has a simple solution of this problem. It looks like this:



```
apps/
|     admin/
|        application.rb
|        assets
|          ...
|        config
|          ...
|        controllers
|          ...
|        templates
|          ...
|        views
|     |     ...
|     web/
        ...
```

You can see that now hanami app has an apps folder. In this folder you can find two different applications.

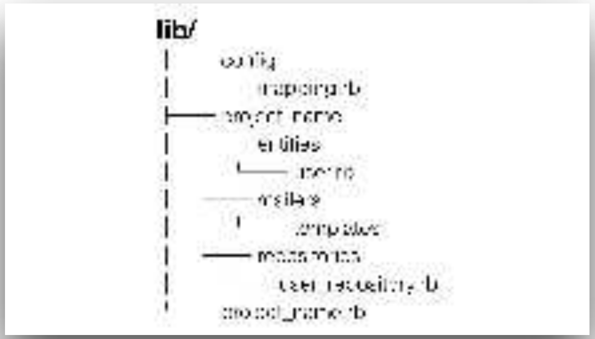


```
apps/
```

In my example the first app is admin. This app contains all parts of monolithe app.

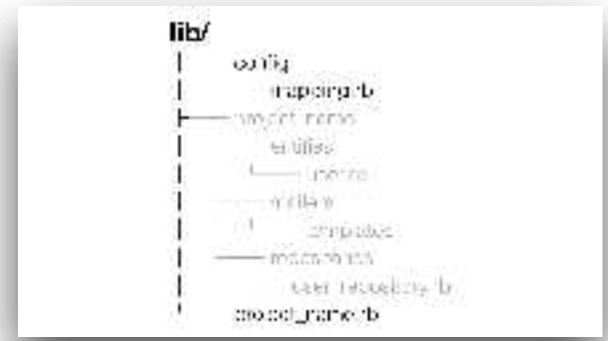
```
apps/
[illegible]
```

and the second app is web



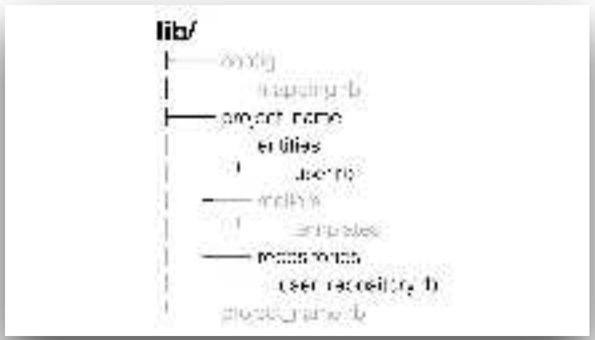
```
lib/
|     config
|        mapping.rb
|---- project_name
|        entities
|        └---- user.rb
|     ---- mailers
|        |     templates
|     ---- repositories
|           user_repository.rb
|     project_name.rb
```

I told that our applications have business logic too. This logic is contained in lib folder.



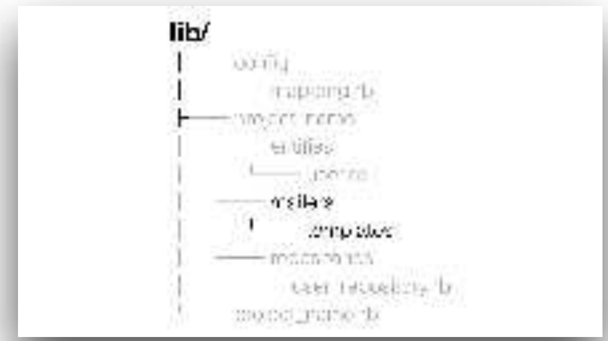
```
lib/
|     config
|         mapping.rb
|---- project_name
|         entities
|     └──── [illegible]
|     ---- mailers
|     |        templates
|     ---- repositories
|            user_repository.rb
|     project_name.rb
```

And this folder has 3 parts too config and mappings



```
lib/
├── config
│       mapping.rb
├── project_name
│       entities
│   │       user.rb
│   ── mailers
│   │       templates
│   ── repositories
│           user_repository.rb
│   project_name.rb
```

model (character M in mvc)

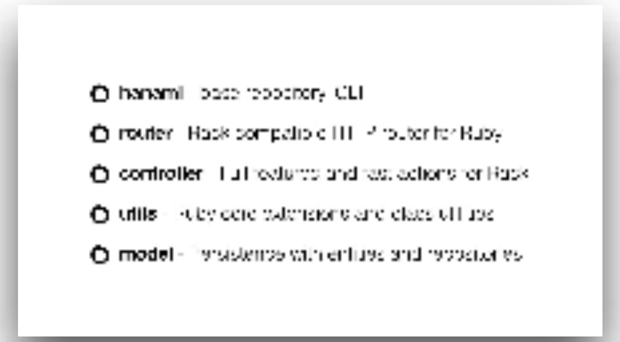


```
lib/
|     config
|         mapping.rb
├── project_name
|       entities
|     └── [illegible]
|   ── mailers
|     |     templates
|   ── repositories
|         user_repository.rb
|     project_name.rb
```

and other stuff, mailer and users libraries

General Parts

I told that hanami is a modular web framefork. Let's look into its parts. The hanami organization has ten different gems



- hanami - base repository, CLI
- router - Rack compatible HTTP router for Ruby
- controller - Full featured and fast actions for Rack
- utils - Ruby core extensions and class utilities
- model - Persistence with entities and repositories

- hanami - base part, this gem mixes all other gem together and provides CLI
- router - Rack compatible HTTP router for Ruby
- controller - Full featured and fast actions for Rack
- utils - Ruby core extensions and class utilities
- model - Persistence with entities and repositories

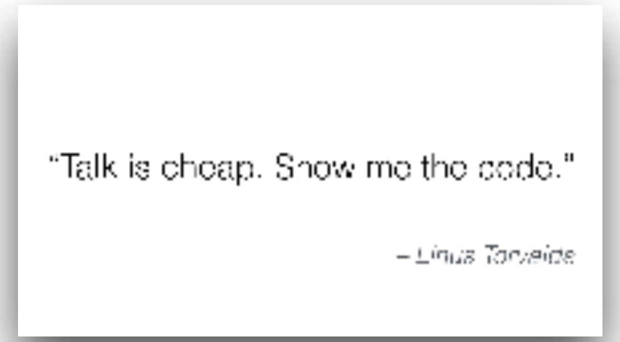
- **validations** - Validations mixin for Ruby objects
- **helpers** - View helpers for Ruby applications
- **view** - Presentation with a separation between views and templates
- **assets** - Assets management for Ruby
- **mailer** - Mail for Ruby applications

the next five

- validations - Validations mixin for Ruby objects
- helpers - View helpers for Ruby applications
- view - Presentation with a separation between views and templates
- assets - Assets management for Ruby
- mailer - Mail for Ruby applications

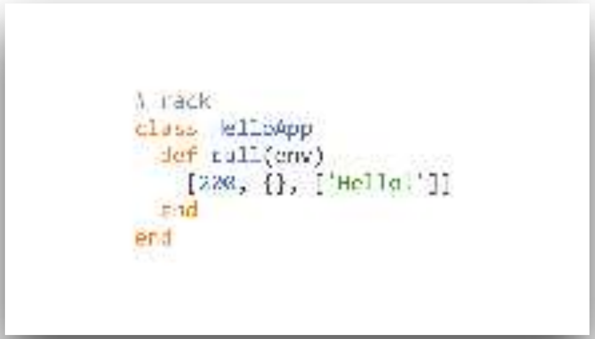
Differences

I know that all of this looks frightening. That’s why let's look at differences with other frameforks


"Talk is cheap. Show me the code."

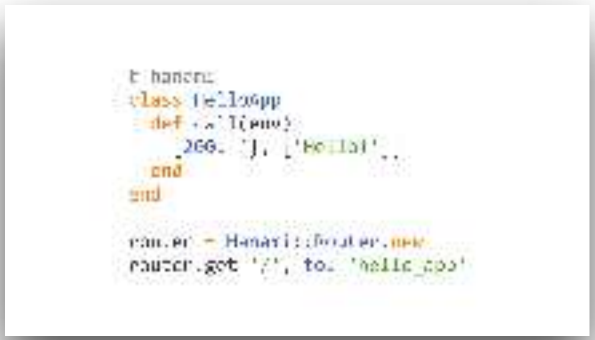
– Linus Torvalds

In this part of my speech I'll show you only code samples, because one famous person Said:



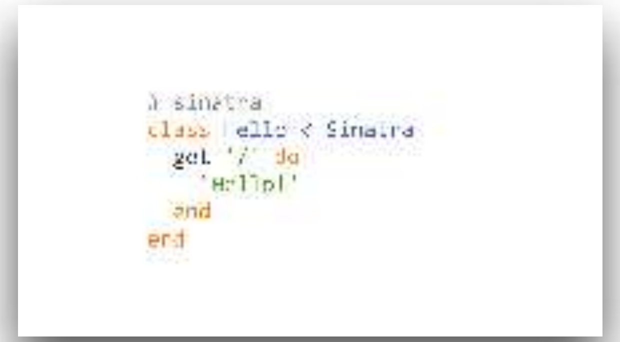
```
# rack
class HelloApp
  def call(env)
    [200, {}, ['Hello!']]
  end
end
```

The first example - is simple rack app I think everyone knows about this.



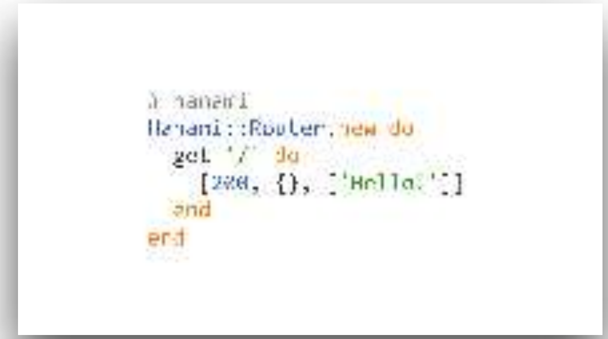
```ruby
# hanami
class HelloApp
  def call(env)
    [200, {}, ['Hello!']]
  end
end

router = Hanami::Router.new
router.get '/', to: 'hello_app'
```

in hanami routes you can use rack apps and it'll look like this:



```
# sinatra
class Hello < Sinatra
  get '/' do
    'Hello!'
  end
end
```

Sinatra. I think it's clear too.



```
# hanami
Hanami::Router.new do
  get '/' do
    [200, {}, ['Hello']]
  end
end
```

Hanami routes again. I lied to you. Because in this example I use block notation instead of class.

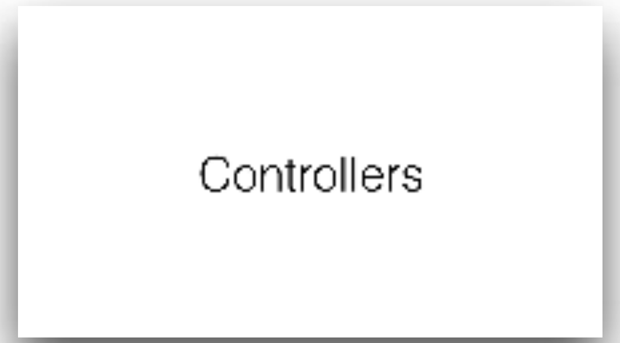
it's logical to compare rails action and hanami action. And I have a really good example of rails action

relax, I'm kidding

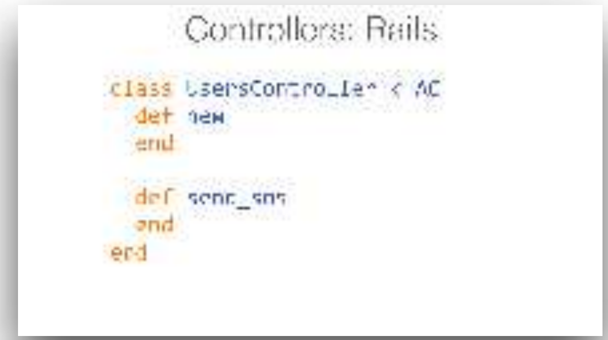
Rails and Hanami

I told about it. hanami and rails are very different
the only thing that unites them is MVC and ruby. That's why I'll cover all parts of MVS and show you how it is realized in rails and hanami.

Rai
FIGHT
anami


Controllers

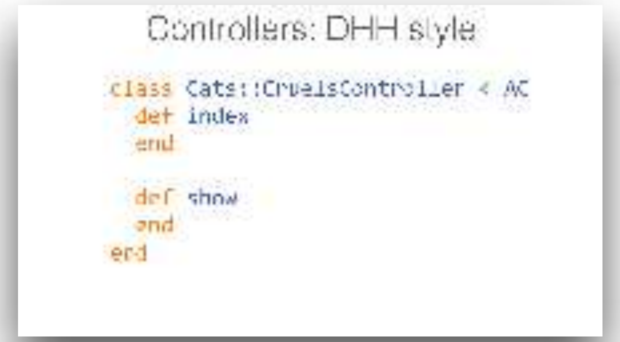
Controllers



Controllers: Rails

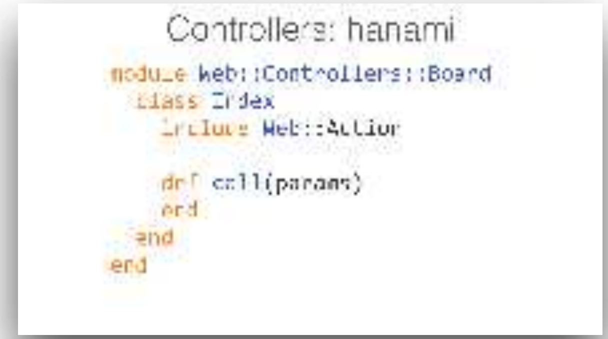
```
class UsersController < AC
  def new
  end

  def send_sms
  end
end
```

Typical rails controller. This is class where each method is action. Action can have any name.


Controllers: DHH style

```
class Cats::CruelsController < AC
  def index
  end

  def show
  end
end
```

rails controller in DHH style. One controller is one class too. But this class can be included to only REST actions.



Controllers: hanami

```
module Web::Controllers::Board
  class Index
    include Web::Action

    def call(params)
    end
  end
end
```

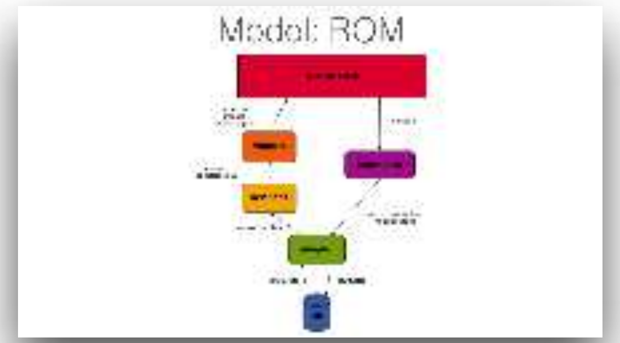
Hanami. You can see that action is a class and controller module. Action has only one public method call. Yes, it looks a like service object.

models

Model: Rails

```
class User < ActiveRecord::Base
  validates :name, presence: true

  # ...
end
```

on this slide you can see a simple AR class. With validations, database logic, data logic and associations.



of course, you can use the ROM. but I think rom has a big number of unnecessary parts and also you'll need to realize adaptors, command mappers, etc

Model: hanami entity

```ruby
class User
  include Hanami::Entity
  attributes :name

  # ...
end
```

Hanami is a cross between rails and ROM. The model has two parts: entity and repository. In entity you work only with data

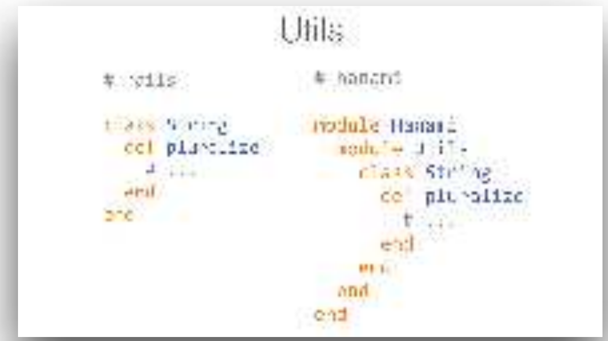
Model: hanami repository

```
class UserRepository
  include Hanami::Repository

  def find_by_name(name)
    query do
      # ...
    end
  end
end
```

in repository you work only with DB logic. For example if you want to create, delete or select record you need to use repository

Utils

utils

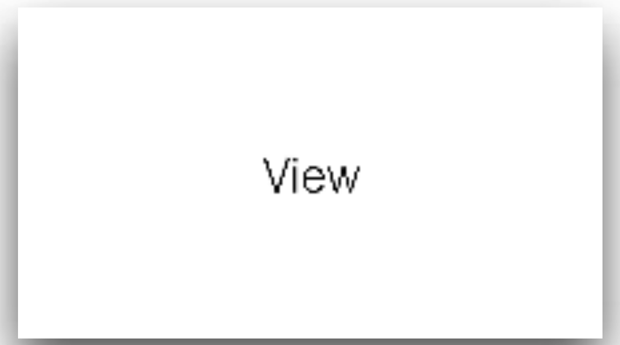


Utils

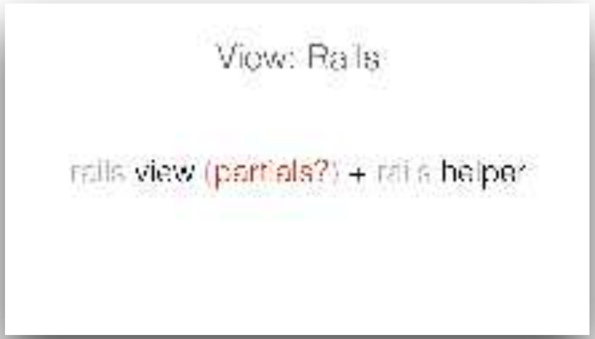
```
# rails

class String
  def pluralize
    # ...
  end
end
```

```
# hanami

module Hanami
  module Utils
    class String
      def pluralize
        # ...
      end
    end
  end
end
```

I have only one slide with string puralize method. You can see that rails monkeypatch core class and hanami create separate class



View


View: Rails

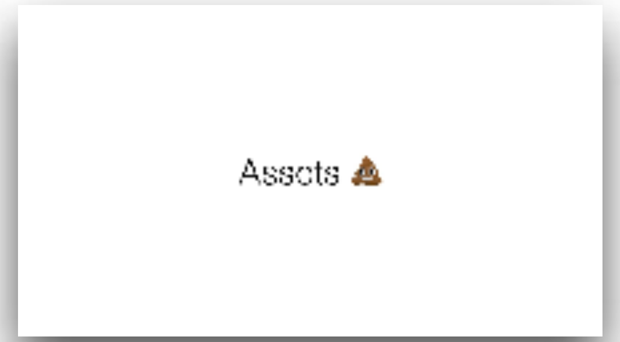
rails view (partials?) + rails helper

How it works in rails
we have a view folder with templates and instance variables which we call in our templates. Also we have helper modules.
I think that everybody in this room knows that rails helpers have some problems.
For example a few days ago I got a bug when one person initialized three methods with one name and after that he had problems.

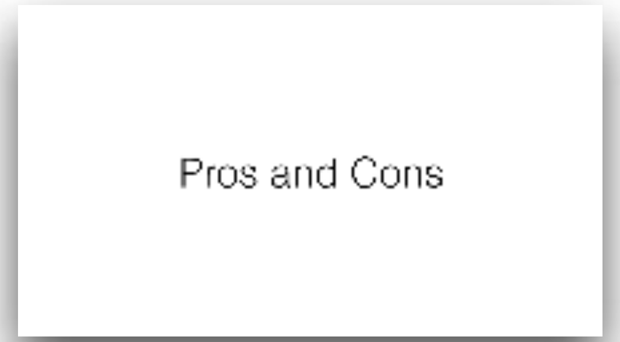
View: Hanami

hanami view (ruby class) + parна

In hanami we have a view object. A view object is a typical ruby class where you can put all your? view logic and call this in templates.
Also we have templates and also hanami has getters from controller instead instance variable.
so and the last part


Assets 💩

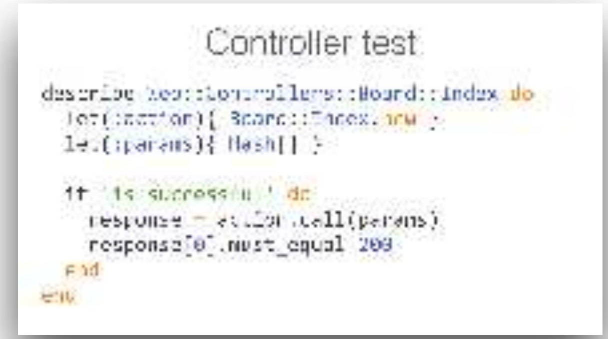
assets
Unfortunately I don't like this, that's why I'll leave it as an elective for independent review


Pros and Cons

The next part of my speech is about Pros and Cons

No magic

No magic:
let's look in to this action helper



Controller test

```
describe Web::Controllers::Board::Index do
  let(:action){ Board::Index.new }
  let(:params){ Hash[] }

  it 'is successful' do
    response = action.call(params)
    response[0].must_equal 200
  end
end
```

this is a real test from my hanami application.

Controller test

```
describe Api::Controllers::Boards::Index do
  let(:action){ Board::Index.new }
  let(:params){ Hash[] }

  it "is successful" do
    response = action.call(params)
    response[0].must_equal 200
  end
end
```

As you can see in let block I initialize new action this is a simple ruby instance.

Controller test

```
[illegible]
  [illegible]
  [illegible]

  [illegible]
    response = action.call(params)
    response[0].must_equal 200
  [illegible]
[illegible]
```

after that I can test it like a usual object

no magic with get and post helpers. Only ruby objects

No monkey-patching

[illegible]

No monkey-patching
I don't know about you but I really often have similar questions on SO.
Why is it important? The general idea lies in erasing boundaries between language and framework.

Best practices

Best practices.
I hope that great tool does not just work correctly this tool inculcates good practices for developer and product
I think that hanami is a great tool because:

Modularity

This framework encourages modularity, but no one should make a cult of it.

The logic separation

also this framework encourages the separation logic. Many developers told about this. Remember SOLID for example.

TDD

This framework uses test first principles.
As you can see, hanami applications have a good testing API.
You've seen earlier how you can test controller, views and models are easily tested as well

oh no, I'm so sorry, I forgot that tdd is dead

Let's talk about cons

TDD

TDD. this is not just a problem of hanami but sometimes you find yourself writing too many tests Also after some time you don't understand how and why you need to test some class

The same type of
code in the controllers

I think you notice that this framework is verbose. - and if you write a simple admin with   CRUD? Itstarts driving you mad

Not stable version

The evident problem is the framework version instability. We occurred active changes in modell remember those times when link_to helper didn’t work correctly. Gem was renamed thanks to IBM

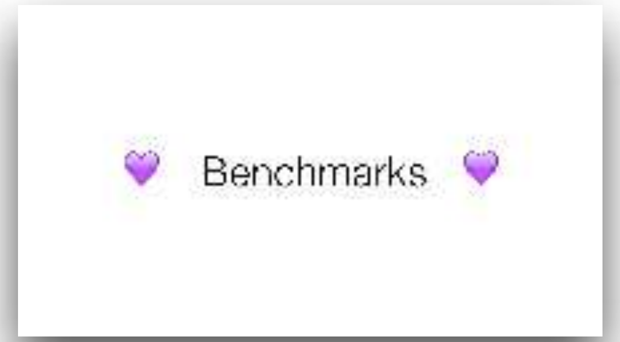
Missing Gems

Hanami is very young comparing with rails which is 10 years old and Sinatra is 8.That's why some useful gems are missing.And I think you can’t create a blog for 15 minutes because many parts you‘ll have to create manually

WebSockets
WebPack
React

if you are a mode boy and want to work with all of this I have some bad news for you. Hanami doesn’t support all of this.But if you want to try to use it, don't forget a developer’s manifesto

I know that I’ve said a lot but be patient, we are almost done



💜 Benchmarks 💜

yes, we all love benchmarks. And I know that benchmarks are only fanaticism, but everyone loves charts.

```
puma *.ru

-

wrk   connections 4 \
      duration 30   \
      threads 4 localhost:9292
```

davydovanton/hanami-bench

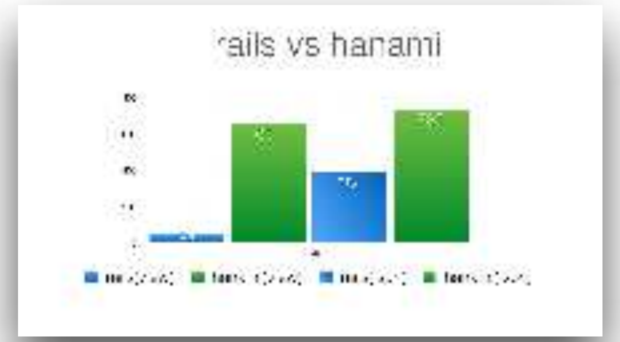
for this I used puma with rackup files and wrk for stress test.

you can find github link bellowif you have any ideas how I can improve my benchmarks I'll be happy to talk about this

on the first chart I compare rack and hanami routes.

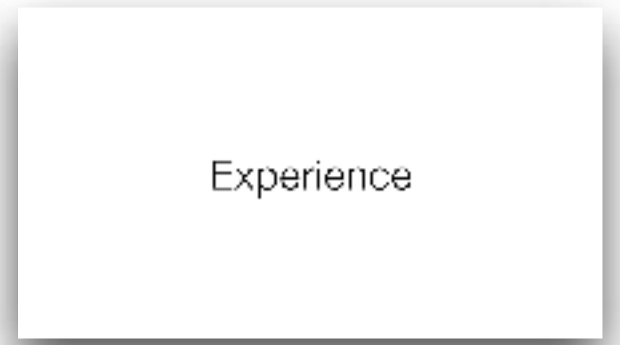
I think that this result was predictable

the next chart compares JSON API serversI use sinatra, grape and hanami routes with controllers



and the last chart shows rails and hanami.
For this I created empty rails and hanami applications. After that I added two actions for each application. first action responde view, and second api.

Draw your own conclusions.


Experience

so, now I want to talk about my experienceI can group All my hanami applications in two and half parts

Types of apps

- Pure API
- link shortener (davydovanton/link_shortener)
- Full app with admin and web parts

There were either pure json api, or web applications with admin pages.Also I created public application which mixed api and web part.

Types of apps

- Pure API
- link shortener ([illegible]/link shortener)
- Full app with admin and web parts

I liked doing apis and actions. Actions let you test your urls. Well, you have more coding but it brings you positive feelings. Now I can say that my next api will be on hamani. I really love controllers and models realisation

Types of apps

- Pure API
- [illegible]
- Full app with admin and web parts

Link shooter has been very controversial. This project contained JSON API and view parts.

Types of apps

- Pure API
- [illegible]
- Full app with admin and web parts

And finally. Web apps with admin pages. I’ve told before , typical admin pages are a very bad part of hanami projects. You need to work with assets and with views and now it is still raw. But I hope that in the future it is going to be much better.

Third-party gems

Last point of my talk: gems

I will answer honestly, there are few of them , but they are being written. Now I’m working with integrate with rodauth from Jeremy. Previously I worked with file upload gem

Rack middleware

but anyway, don't forget that hanami is rack wrapper and you can use rack gems anytime.

Contacts

- hanamirb.org
- gitter.im/hanami/chat
- discourse.hanamirb.org

if you are interested with hanami, see this links. On this slide you can find all necessary links: link to the main site chat and forum

Thank you ❤️

[illegible]

[illegible]

Thank you for listening, any questions?