Hanami

beyond Rails

Warm-up questions

Why go beyond Rails?

Rails critique

- Rails is good for some class of applications, but it's not a silver bullet
- As the complexity grows, Rails is not the best at handling it
- A lot of monkey patching
- People usually move from Rails to other languages (Elixir, Go), but perhaps it's unnecessarily radical
- Trailblazer, DDD, Hexagonal

Hanami

What is it?

A modern web framework for Ruby

- The web, with simplicity.
- Fast response times
- Full-featured, but lightweight
- Secure by default
- Simple and productive

Hanami

What is it?

History

- Started some time in 2013 as Lotus by Luca Guidi
- 1.01.2014: First announcement
- 23.07.2014: First release
- 22.01.2016: Renamed to Hanami after "friendly warning" from IBM
- Hanami (花見) is a Japanese tradition of watching the cherry trees to blossom during spring. During this season, they celebrate this event all together in their beautiful gardens.
- 15.11.2016: Released version 0.9
- Latest release: 0.9.2 yesterday

Hanami

What is it?

History

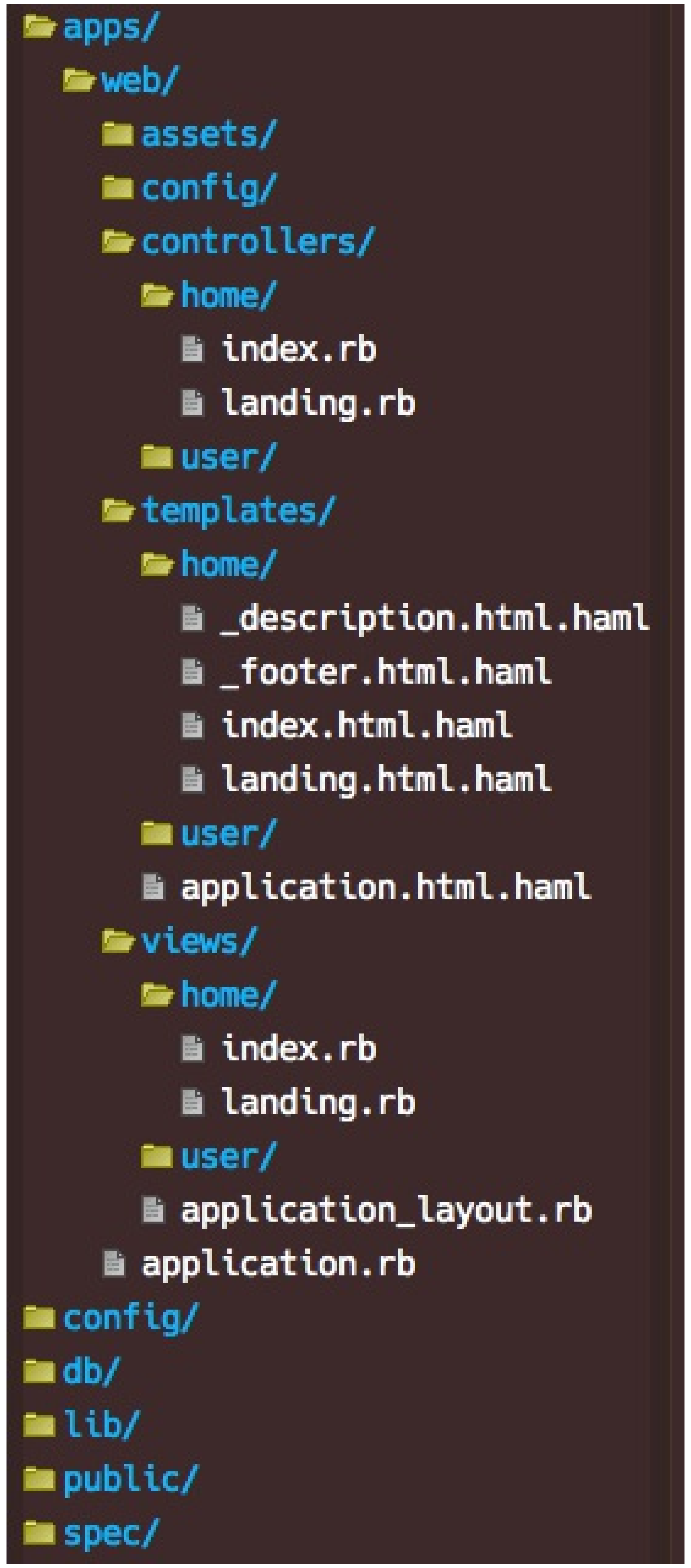
Philosophy

Hanami is a set of relatively small, independent components + one umbrella component. They can be used separately as well.

- hanami/router
- hanami/model
- hanami/controller
- hanami/view
- hanami/assets
- hanami/mailer
- hanami/utils
- hanami/helpers
- hanami/validations
- hanami/hanami

hanami/hanami

- Generators for new application, models, migrations etc.
- CLI tools for database management (if `hanami-model` is plugged in)
- Conventions if all components are tied together - need to write less boilerplate


```
apps/
  web/
    assets/
    config/
    controllers/
      home/
        index.rb
        landing.rb
      user/
    templates/
      home/
        _description.html.haml
        _footer.html.haml
        index.html.haml
        landing.html.haml
      user/
      application.html.haml
    views/
      home/
        index.rb
        landing.rb
      user/
      application_layout.rb
    application.rb
config/
db/
lib/
public/
spec/
```

- `apps` directory may contain many applications
- it's common to have `web`, `admin` and `api` there
- common logic (models etc.) sits in `lib` directory

Router

Example

```ruby
Hanami::Router.new do
  root                    to: ->(env) { [200, {}, ['Hello']] }
  get '/lambda',          to: ->(env) { [200, {}, ['World']] }
  get '/dashboard',       to: Dashboard::Index
  get '/rack-app',        to: RackApp.new
  get '/flowers',         to: 'flowers#index'
  get '/flowers/:id',     to: 'flowers#show'

  redirect '/legacy',     to: '/'

  namespace 'admin' do
    get '/users', to: Users::Index
  end

  resource 'identity' do
    member do
      get '/avatar'
    end

    collection do
      get '/api_keys'
    end
  end
end
```

Controller

Controller

Overview

- Controller per se does not exist. It's only a namespace for actions
- One action = one class
- One public method: `call`
- Instance variables are not automatically exposed to views, you need to use `expose` method

Controller

Overview

Example

```ruby
module Web::Controllers::Images
  class Index
    include Web::Action

    expose :images

    def call(params)
      @images = ImageRepository.new.all
    end
  end
end
```

View

+ template

View

Overview

- View is a separate layer between controller and template
- View should define helper methods used in templates
- Exposures from controller are passed through views to templates
- Might be bypassed if action sets `self.body`

```ruby
module Web::Controllers::Dashboard
  class Index
    include Web::Action

    def call(params)
      self.body = 'OK'
      self.status = 200
      self.headers.merge!({ 'X-Custom' => 'OK' })
    end
  end
end
```

View

Overview

Example

```
module Admin::Views::Dashboard
  class Index
    include Admin::View

    def form
      form_for :item, '/admin/items' do
        div class: 'input-group input-group-lg' do
          text_field :content, class: 'form-control'
          span class: 'input-group-btn' do
            submit 'Submit', class: 'btn btn-search'
          end
        end
      end
    end
  end
end
```

Model

Model

Overview

- Entities and Repositories are separated
- Entity is responsible for behaviour
- Repository is responsible for persisting

```
bundle exec hanami generate model book
  create  lib/bookshelf/entities/book.rb
  create  lib/bookshelf/repositories/book_repository.rb
  create  spec/bookshelf/entities/book_spec.rb
  create  spec/bookshelf/repositories/book_repository_spec.rb
```

- Based on ROM (Ruby Object Mapper; new in 0.9)
- Change to ROM is VERY breaking

Model

Overview

Entity

```
class User < Hanami::Entity
  def approved?
    !self.approved_at.nil?
  end
end
```

- Entity is frozen (new in 0.9) - cannot be changed after initilization
- Schema is autogenerated for SQL databases (new in 0.9)

Model

Overview

Entity

You can still define your own schema (mandatory for NoSQL)

```ruby
class User < Hanami::Entity
  EMAIL_FORMAT = /\@/

  attributes do
    attribute :id,         Types::Int
    attribute :name,       Types::String
    attribute :email,      Types::String
                            .constrained(format: EMAIL_FORMAT)
    attribute :age,        Types::Int.constrained(gt: 18)
    attribute :comments, Types::Collection(Comment)
    attribute :created_at, Types::Time
    attribute :updated_at, Types::Time
  end
end
```

Model

Overview

Entity

Repository

- All queries are private and need to be defined as methods
- Repository methods accept hash of params or entity

```ruby
class BookRepository < Hanami::Repository
  def most_recent_by_author(author, limit: 8)
    books
      .where(author_id: author.id)
      .order(:published_at)
      .limit(limit)
  end
end
```

Usage:

```ruby
repository = BookRepository.new

book = repository.create(title: "Hanami")
book = book.find(book.id)
book = repository.update(book.id, title: "Hanami Book")
repository.delete(book.id)
author = AuthorRepository.new.find(1)
repository.find_recent_by_author(author)
```

Support for associations (new in 0.9)

```
user = User.new(comments: [Comment.new(text: "cool")])
user.comments
#=> [#<Comment:0x007f966be20c58 @attributes={:text=>"cool"}>]
```

- Considered experimental
- Works only for SQL adapter
- No lazy loading
- Only one-to-many relationships supported right now

```
class AuthorRepository < Hanami::Repository
  associations do
    has_many :books
  end

  def create_with_books(data)
    assoc(:books).create(data)
  end

  def find_with_books(id)
    aggregate(:books).where(authors__id: id).as(Author).one
  end
end
```

Model

Overview

Entity

Repository

Associations

Migrations

- Created manually (not by model generator; discussion is ongoing)

```
Hanami::Model.migration do
  change do
    create_table :books do
      primary_key :id
      foreign_key :author_id, :authors,
        on_delete: :cascade, null: false

      column :code,  String,
        null: false, unique: true, size: 128
      column :title, String,
        null: false
      column :price, Integer,
        null: false, default: 100

      check { price > 0 }
    end
  end
end
```

- `hanami-model` is not mandatory (like `ActiveRecord` is not in Rails, BTW)
- You are free to use any other ORM
- It is described in official docs
- Might be a good idea at the moment

Other components

Utils

- A collection of useful tools, not documented
- Deprecations
- Escaping HTML and URLs
- Class attributes
- Load paths
- Enhanced base classes (String, IO, Hash…)
- Inflections
- Interactors

Utils: Interactor

A pattern for units of complex operations

```ruby
class Signup
  include Hanami::Interactor
  expose :user

  def initialize(params)
    @params = params
    @user   = User.new(@params)
  end

  def call
    @user = UserRepository.persist(@user)
  end

  private
  def valid?
    @params.valid?
  end
end

result = Signup.new(name: nil).call
result.successful? # => false
result.failing? # => true

result.user   # => #<User:0x007fa311105778 @id=nil @name="Luca">
```

Helpers

A collection of view helpers, such as form helpers, numbers helpers, escape helpers, routing helpers etc.

```
form_for :book, routes.books_path do
  text_field :title

  submit 'Create'
end

html.aside(id: 'sidebar') do
  p "Numbers", class: 'title'

  ul do
    li format_number('1000')
    li format_number(1/3, precision: 10)
  end
end
```

Validations

- Based on `dry-validations`
- Introduced in 0.8

```ruby
class Signup
  include Hanami::Validations

  validations do
    required(:name) { filled? & str? & size?(3..64) }
  end
end

result = Signup.new(name: "Luca").validate
result.success? # => true
```

Assets

- Supports for assets placed in `apps/*/assets`
- They are lazily copied to `public` in development mode
- Preprocessors support
- View helpers for assets

Mailer

```
Hanami::Mailer.configure do
  delivery_method :smtp,
    address:              "smtp.gmail.com",
    port:                 587,
    domain:               "example.com",
    user_name:            ENV['SMTP_USERNAME'],
    password:             ENV['SMTP_PASSWORD'],
    authentication:       "plain",
    enable_starttls_auto: true
end.load!

class WelcomeMailer
  include Hanami::Mailer

  from 'noreply@sender.com'
  to   'noreply@recipient.com'
  cc   'cc@sender.com'
  bcc  'alice@example.com'

  subject 'Welcome'
end

WelcomeMailer.deliver
```

Ecosystem, community etc.

Ecosystem

Community

- `hanami` organization on Github has 19 people (core team)
- Active chat on Gitter
- One user group in Brasil

Ecosystem

Community

Libraries

- As any young frameworks, Hanami lacks a large ecosystem
- awesome-hanami.org
- For example, there is no Devise for Hanami. There is no "plug-and-play" solution for authentication.
- The only upload library (hanami-shrine) does not work with 0.9 yet (but will work soon)

Ecosystem

Community

Libraries

Production use

- DNSimple - web API
- Chef - download section
- Envato
- Vagas.com - Brasilian Applicant Tracking System, uses 16-years-old legacy database with 100k+ tables and 2TB of data. They use Hanami because it's easy to plug in to old database, which is close-to-impossible with ActiveRecord.

Ecosystem

Community

Libraries

Production use

Future

- Going for 1.0-RC
- Planned for the end of the year, but not gonna happen
- A lot of breaking changes lately, but promised to be the last batch of those

Thank you!

Questions?

Paweł Świątkowski

@katafrakt_pl

http://katafrakt.me