Ruby on Rails training

Wojtek Sykurski

Samsung R&D Poland

w.sykurski@samsung.com

March 13, 2016

Agenda

Day First Agenda:

- Web Applications Frameworks
- Technologies
  - Most popular
  - Ruby on Rails
- Ruby
- Ruby on Rails
  - Fast Start
  - MCV
- Layouts

Technologies

- M$ Technologies
- Rest of the World

Technologies - Rest of the world

- Pure HTML
- Java - different technologies
- PHP
- Python based Frameworks
  - Django
  - web2py
  - TurboGears
  - Others
- Ruby based Frameworks
  - Ruby on Rails
  - Sinatra
  - Cuba, Merb and many others

Technologies - Ruby on Rails

- Fully operational Web Framework:
  - Build in WebServer(Webrick)
  - Database Engine (SQLight)
  - Debuging and Tracking tools
  - Support for plugins:
    - Made "by hand"
    - Ruby Gems
- Use MCV model
  - Model - Database communication
  - View - User Front-end
  - Controller - Integration and synchronization between Model and View
- Integration with most of popular Web servers and Databases

Ruby

- High level programming language
    - Dynamic & Reflective
        - “On Fly” Compilation and execution
        - Drawback - errors handling and detection
        - “Test driven development”
    - Object orientated
- Similar to python
- Interactive shell - irb

Ruby - Interactive Shell

Starting IRB:

- Start terminal
- Type “irb”

Ruby - variables

- Global Variables - sporadic use in Ruby on Rails:
  - $\$Global\_Var = 10$
- Instance Variables - very often used in Ruby on Rails:
  - $@Instance\_Var = 10$
  - RoR usage - transfer data between controller, model and view
- Class Variables - in practice - never used in RoR:
  - $@@Class\_Var = 10$
- Local Variables - common use in RoR:
  - $Local\_Var = 10$

Ruby - operators

- Add, Sub, Multiply, Divide, Power
  - numbers
  - $var1 = 3 * 3$
  - $var1 * *3$
  - string, some other classes
  - $var2 =' jakistekst'$
  - $var2 * 3$
- Assign value
  - Standard operators
  - Ruby special

Ruby - classes

Definition (Class)

Class is a template for custom objects creation. Those objects contains member variables and functions (methods)

```
class TestClass
   def initialize
     @instance_var=10
   end

   def method
     put 'I am test class'
   end

   def instance_var
     return @instance_var
   end

   def instance_var=(var)
     @instance_var = var
   end
end
```

Ruby - classes - attribute accessor

Definition (Attribute Accessor)

Attribute accessor provide easy access to instance variables

```
class TestClass
  attr_accessor :instance_var
  def initialize
    @instance_var=10
  end

  def method
    put 'I am test class'
  end

end
```

Ruby - classes - private/protected

Definition (Private methods)

Private methods - can only be used by other methods of the same object. AND OBJECT OF CLASSES that inherit from this class !!!

```
class TestClass
   attr_accessor :instance_var
   def initialize
     @instance_var=10
     method
   end

   private:

   def method
     put 'I am test class'
   end

end
```

Ruby - classes - private/protected

Definition (Private methods)

Private methods - different annotation.

```
class TestClass
   attr_accessor :instance_var
   def initialize
     @instance_var=10
     method
   end

   def method1
     put 'I_am_test_class'
   end

   def method2
     put 'I_am_test_class'
   end

   private :method, :method2

end
```

Ruby - classes - protected

Definition (Protected)

Protected works almost the same way as private!
Only difference is that protected methods can be used by other objects of the same class

```
class TestClass
  attr_accessor :instance_var
  def initialize
    @instance_var=10
    method
  end

  protected:

  def method1
    put 'I am test class'
  end

  def method2
    put 'I am test class'
  end
end
```

Ruby - gems

Definition

Ruby gems - packs of plugins, libraries, binaries and source code files, which can be fast added to our existing Ruby projects. It is similar to Python pip tool.

Definition

In Ruby on Rails gem's handling is done by tool called "bundler".

Example

Popular RoR gems:

- Carrier
- Authlogic
- RMagic
- Sass - present by default in RoR from version 3

Ruby on Rails - fast start

```
mkdir Rails_Group_1
cd Rails_Group_1
rails new helldesk
rails server

mkdir Rails_Group_2
cd Rails_Group_2
rails new helldesk
rails server
```

Ruby on Rails - install gems to environment

Definition

Bundler - we need to use it, after creating new app. It would install all required gems and libraries in our environment. On most of systems it will start automatic after creating new app. If not, just run:
bundle install

Now, go for http address: http://localhost:3000

Ruby on Rails - structure

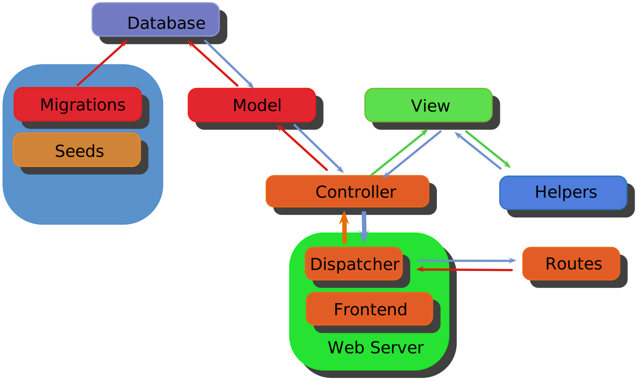
Main components of RoR application:

- MCV model
- Configuration
  - Environments
  - Database
  - Gemfile
  - Routes
- Utils
  - Routes
  - Rake
  - Bundler

Ruby on Rails - MCV

Definition

MCV (Model, Controller, View) is a software architecture used in RoR. It separates database operations (Model), content generation (View) and application logic (Controller) in to different components.



Ruby on Rails - Model

Definition

Model - describes database object (usual table), relations with other models and validations of data.

Example (user.rb)

```
class User < ActiveRecord::Base
  validates :name, :presence => true,
                   :uniqueness => true
  validates :password, :confirmation => true
  ...
end
```

Attention !!!

Models DO NOT describe table structure, column names, data types and so.

Ruby on Rails - controller

Definition

Controller is responsible for application logic by providing connection between models, views, libraries and gems. It handles and generates requests from web server.

Ruby on Rails - View

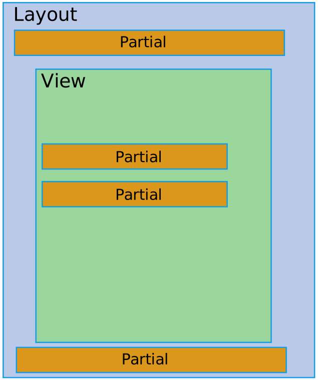
How does Rails generate page:

Main elements:

- Layout
- View
- Partial

Other:

- Helpers
- Assets
  - js, coffee scrpits
  - css
  - images, video, music, etc.



Ruby on Rails - View

Definition

View contains the html web page structure, with additional RoR logic

Example (user.html.erb)

```
<% provide(:title, 'Index') %>
<h1>Listing Users</h1>
<% if notice %>
  <p id="notice"><%= notice %></p>
<% end %>
<table>
<tbody>
    <% @users.each do |user| %>
      <tr>
        <td><%= user.name %></td>
        <td><%= user.email %></td>
        <div class="actions">
            <td><%= link_to 'Show', user %></td>
            <td><%= link_to 'Edit', edit_user_path(user) %></td>
            <td><%= link_to 'Destroy', user, method: :delete, data: { confirm: 'Are_you_sure
        </div>
      </tr>
    <% end %>
  </tbody>
</table>
<br>
<%= link_to 'New_User', new_user_path %>
```

Ruby on Rails - View Layout

Definition

Layout Layout contain general structure of page. Views are generated "inside" application layout file

Example (user.html.erb)

```
<% provide(:title, 'Index') %>
<h1>Listing Users</h1>
<% if notice %>
  <p id="notice"><%= notice %></p>
<% end %>
<table>
<tbody>
    <% @users.each do |user| %>
      <tr>
        <td><%= user.name %></td>
        <td><%= user.email %></td>
        <div class="actions">
            <td><%= link_to 'Show', user %></td>
            <td><%= link_to 'Edit', edit_user_path(user) %></td>
            <td><%= link_to 'Destroy', user, method: :delete, data: { confirm: 'Are you sure
        </div>
      </tr>
    <% end %>
  </tbody>
</table>
<br>
```

Ruby on Rails - Environments

Definition

RoR Environments Environments - describes server run time environment, can be used for other configuration files.

Files:

- All files in ./config/environments
- ./config/environments.rb

Example

- Development - will be used during this course
- Production - will be used during this course
- Test - will be used during this course

Ruby on Rails - Create basic content

Definition

Rails generator - tool for controllers, views, models and helpers automatic creation

For now, lets just put this in terminal:

```
rails generate controller static_content start about help
```

And go to:

http://localhost:3000/static_content/start
http://localhost:3000/static_content/about
http://localhost:3000/static_content/help

Ruby on Rails - Change page root

Definition

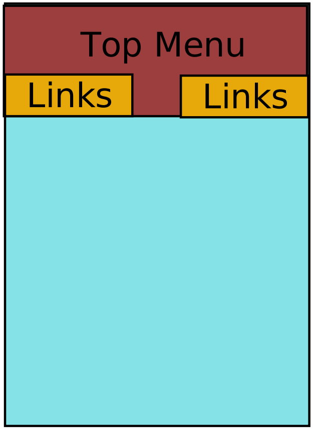
Rails routes describes how to redirect html requests to specific controller/view File location: ./config/routes.rb

To change Web app root to specific controller and view, add following to this file:

```
    root 'static_content#start'
```

Customizing layout

What we want to achieve:



Required Steps:

- Add top menu:
  - List of links
  - Custom CSS with positioning
- Display top menu on all pages
  - Needs to be added to Layout
  - Use of “Partials”
- Dynamic Pages Title generation
  - Provide title for every subpage
  - Error handling if subpage has none
  - Create custom functions for handling all those conditions
  - Place theme in helper

Layouts - Creating top menu

Add folowing code to ./app/views/layouts/application.html.erb in “Body” section:

```
<div id="left_menu">
    <ul>
        <li><%= link_to "Issues", "#" %> </li>
        <li><%= link_to "Admin", "#" %> </li>
    </ul>
</div>

<div>
    <ul>
        <li><%= link_to "Login", "#" %></li>
        <li><%= link_to "About", "#" %></li>
        <li><%= link_to "Help", "#" %></li>
    </ul>
</div>
```

Layouts - Customizing CSS

- Display links in line
- Left and Right menu
- Place menu on top
- Add some CSS3 effects - useful link: http://www.css3maker.com/

Layouts - Move top menu to Partial

Definition

Partial is fragment of HTML code which can be included in other Web Pages. It is very useful for generating lists, tables (for ex. table of products) and navigation menus

In this case, We want to move source code from application layout to partial, to make it more clean and readable. Required steps:

- Create new file ./app/views/layouts/_top_menu.html.erb
- Copy all top menu div to that file
- In application layout add

```
<%= redren 'layouts/top_menu" %>
```

Page Title - dynamic generated

Definition

To send some variable from View to Layout use the “provide” build in function.

```
<% provide(:title, 'Index') %>
```

To handle it in layout, following code can be used:

```
<% unless yield(:title).empty? %>
    <title>Helldesk2 | <%=yield(:title)%> </title>
<%else %>
    <title>Helldesk2</title>
<% end %>
```

Dynamic Title: Use Helper

Definition

Helpers all special files, which behave like libraries - they store functions (helpers), which can be then use in views and layouts, but Not in Controllers and Models !!!

```
    def provide_title(subtitle='')
      hell = "Helldesk"
      if subtitle.empty?
        hell
      else
        hell + " | " + subtitle
    end
  end
```

To call it in application layout:

```
<title><%=provide_title(yield(:title))%></title>
```

Rails routes - update

We can make routes paths more user (developer) friendly, for further use.
Please add following to routes.rb file:

```
controller :static_content do
   get 'start' => :start
   get 'help' => :help
   get 'about' => :about
end
```

And then add some real links to out “top menu”

```
<div id="right_menu">
    <ul>
        <li> <%= link_to "Login", "#" %></li>
        <li> <%= link_to "About", :about %></li>
        <li> <%= link_to "Help", :help %></li>
    </ul>
</div>
```

Second Day Agenda:

- Users

What we need for User ?

- Login and logout
    - Limit access for not logged persons
- Store some information in database
- Create Issues
- Admin role:
    - create new users
    - change existing Users data

Rails Generators

There are several options for rails generators:

- scaffold
- model
- controller
- asset
- plugin
- custom items - created by developer or obtained from gems

Definition

Scaffold generator include both model, controller and assets generator. It also creates migrations files for DB tables creation.

```
rails generate scaffold User name:string email:string hash_password:string salt:string
```

Rails migrations

Definition (Rake)

Rake is tool to perform predefined tasks in Rails applications. Most often it is used to perform database migrations.

Definition (Migrations)

Database migration is any action, which change DB table structure - creates new ones, remove or modify existing. It can be also used to perform data modifications (adding, removing or updating of rows), but there are other tools for those tasks.
Migrations are located in ./db/migrations folder.

```
rake db:migrate
```

Rails migration example

Example

```
class CreateUsers < ActiveRecord::Migration
def change
 create_table :users do |t|
   t.string :name
   t.string :email
   t.string :hash_password
   t.string :salt

   t.timestamps null: false
  end
 end
end
```

User Model

- How to encrypt password
  - Creating Simple Hash
  - Add some random stuff to it ("salt")
  - Store it in DB
- How to save password from formula
- How to compare stored hash with password

Rails Console

Definition

Rails console is (in basic) IRB with loaded Rails Environment.

To start it, please launch in main app folder:

```
rails console development
```

And lets import SHA2 libraries:

```
require 'digest/sha2'
```

Encrypt example:

```
Digest::SHA2.hexdigest(``example_word'')
```

Tunning up User model

Please add the following to User model:

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

Definition

Validations are rules and method if model object meets certain criteria. Those parameters can be both predefined (like presence, length, matching certain regex) or custom defined. In this second case we use validate instead of validates

Tunning up User model cd.

```
require 'digest/sha2'
class User < ActiveRecord::Base
  validates :name, :presence => true, :uniqueness => true
  validates :password, :confirmation => true
  attr_accessor :password_confirmation
  attr_reader :password
  validate :password_must_be_present

  def User.encrypt_password(password, salt)
    Digest::SHA2.hexdigest(password + "slowo" + salt)
  end

  def password=(password)
    @password = password
    if password.present?
      generate_salt
      self.hash_password = self.class.encrypt_password(password,self.salt)
    end
  end

  def User.authenticate(name,password)
    if user = User.find_by_name(name)
      if user.hash_password == encrypt_password(password, user.salt)
        user
      end
    end
  end
```

Tunning up User model cd.

```
  private

  def password_must_be_present
    errors.add(:password, "Missing password") unless hash_password.present?
  end

  def generate_salt
    self.salt = self.object_id.to_s + rand.to_s
  end
end
```

User Controller and Views

What we have to change, to allow new user creation and existing ones modifications ?

- User Controller

```
# Never trust parameters from the scary internet, only allow the white list through.
def user_params
  params.require(:user).permit(:name, :email, :password, :password_confirmation)
end
```

- User views - _form.html.erb

```
<div class="field">
  <%= f.label :name %><br>
  <%= f.text_field :name %>
</div>
<div class="field">
  <%= f.label :email %><br>
  <%= f.text_field :email %>
</div>
<div class="field">
  <%= f.label :password %><br>
  <%= f.password_field :password %>
</div>
<div class="field">
  <%= f.label :password_confirmation %><br>
  <%= f.password_field :password_confirmation %>
</div>
<div class="actions">
  <%= f.submit %>
</div>
```

Third Day Agenda:

- Session

What is Session?

Definition

In case of our app - Session will determine if User is currently logged. It will also check for his privileges and restrict access.

In our example We would use encrypted cookies files to store session data

```
rails generate controller Session new create delete
```

Creating log in formula

Definition (Form_tag)

Form_tag - one of build in Rails helpers. We will use it for creating 'post' data formula.

Definition (url_for)

url_for - another build in helper :)
It is used for generating http addresses from controllers and actions data inputs.

In Session new view:

```
<%= form_tag url_for(:controller => :session, :action => "create"), :method => :post do %
  <fieldset>
    <legend>Please log in</legend>
    <%= label_tag :name, "User_name" %>
    <%= text_field_tag :name, params[:name] %>
    <br>
    <%= label_tag :password, "User_password:" %>
    <%= password_field_tag :password, params[:password] %>
    <br>
    <%= submit_tag 'Log In' %>
  </fieldset>
<% end %>
```

Some changes of routes.rb

Change following line for session part:

```
post 'session/create'
```

Lest add some tunning to session controller:

Definition

redirect_to - Build in functions, for handling redirection in controllers.

```
class SessionController < ApplicationController
  skip_before_action :authorize,
                     :only => [:new, :create]
```

And ... enjoy login :)
You can check in browser, if login was successful (cookies).

Simplify some things

Routes (routes.rb):

```
controller :session do
  get 'login' => :new
  post 'login' => :create
  delete 'logout' => :delete
end
```

Form_tag(Session/new.html.erb)

```
<%= form_tag do %>
  <fieldset>
    <legend>Please log in</legend>
    <%= label_tag :name, "User name" %>
    <%= text_field_tag :name, params[:name] %>
    <br>
    <%= label_tag :password, "User password:" %>
    <%= password_field_tag :password, params[:password] %>
    <br>
    <%= submit_tag 'Log In' %>
  </fieldset>
<% end %>
```

Restricting Access

Objectives:

- Check if user is logged
- If not - redirect him to login path

Check if user has logged(application_controller.rb):

```
before_action :getUser
before_action :authorize

protected

def getUser
  @current_user = User.find_by_id(session[:user_id])
end

def authorize
  unless @current_user
    redirect_to login_url, notice: "You_have_to_log_in_first_!!!"
  end
end
```

Why our page is broken now ?

We need to allow anonymous access to login page:

```
class SessionController < ApplicationController
  skip_before_action :authorize,
                     :only => [:new, :create]
```

Definition

before_action - execute method before every (or by using “only” parameter just for some) action in controller. In case of Application_Controller, this would be done in all controllers.

Definition

skip_before_action - do not perform this method for some actions.

Adding logout functionality

Add to application helper:

```
def login_logout
  if @current_user
    link_to 'Logout', :logout, :method => :delete
  else
    link_to 'Login', :login
  end
end
```

Update "top_menu" layout:

```
<div id="right_menu">
<ul>
    <li> <%= login_logout %></li>
```

How to store session data

Controller: Do we need anything more?

Fourth Day Agenda:

Fifth Day Agenda: