



# ***ActiveResource, RESTful webszolgáltatások, AJAX, ActionMailer, ActiveJob***

Kovács Gábor

`kovacsg@tmit.bme.hu`

BME-TMIT

# REST ismétlés

- ⑥ A HTTP kérés URI-ja egy erőforrást azonosít
- ⑥ Egy erőforrásnak több reprezentációja van: HTML, XML, JSON stb.
- ⑥ A HTTP metódus határozza meg az erőforrással teendő akciót
- ⑥ Az alkalmazás állapotát az erőforráson elérhető linkek definiálják
- ⑥ Például:

```
DELETE http://localhost:3000/tasks/1
```

```
GET http://localhost:3000/tasks/1
```

```
PUT http://localhost:3000/tasks/1
```

# *REST parancsok*

HTTP metódus	Akció	Eseménykezelő	SQL
GET	index,show	index, show	select
POST	new	create	insert
PUT	edit	update	update
DELETE	delete	destroy	delete

# Resourceful útvonalak 1

- 6 A scaffolddal létrehozott Task egy `resources :tasks` sort szűrt be

HTTP metódus	Útvonal	Kontroller akció	Útvonal helper
GET	<code>/tasks</code>	<code>index</code>	<code>tasks_path</code>
GET	<code>/tasks/new</code>	<code>new</code>	<code>new_task_path</code>
POST	<code>/tasks</code>	<code>create</code>	<code>tasks_path</code>
GET	<code>/tasks/:id</code>	<code>show</code>	<code>tasks_path(id)</code>
GET	<code>/tasks/:id/edit</code>	<code>edit</code>	<code>edit_task_path(id)</code>
PUT	<code>/tasks/:id</code>	<code>update</code>	<code>tasks_path(id)</code>
DELETE	<code>/tasks/:id</code>	<code>destroy</code>	<code>tasks_path(id)</code>

## Resourceful útvonalak 2

- ⌚ Azonosító nélküli útvonalakat a `resource :task` sor szűr be
- ⌚ Szingleton modellek esetén használjuk

HTTP metódus	Útvonal	Kontroller akció	Útvonal helper
GET	/task/new	new	new_task_path
POST	/task	create	task_path
GET	/task	show	task_path
GET	/task/edit	edit	edit_task_path
PUT	/task	update	task_path
DELETE	/task	destroy	task_path

# Resourceful útvonalak 3

- ⑥ Az útvonalhoz tartozó alapértelmezett kontroller átállítható:  
`resources :users, :controller => :felhasznalo`
- ⑥ Útvonal prefix felüldefiniálása: `:as => 'feladat'`
- ⑥ Útvonal postfix felüldefiniálása: `:path_names=>{:new=>"uj"}`
- ⑥ Útvonalak kihagyása: `:only, :except`
- ⑥ Kontrollerek modulokba rendezése esetén az útvonalakat névtérhez rendelhetjük. Például az `Admin` modul útvonalaira:  

```
namespace :admin do
  resources :users
end
```
- ⑥ Ekkor az útvonalak a névtér nevének megfelelő előtaggal bővülnek, a példában `admin`

# Resourceful útvonalak 4

- ⑥ Modell osztályok közötti relációk esetén (`has_many`, `belongs_to`) az útvonalak is egymásba ágyazhatók

```
resources :users do
  resources :tasks
end
```

- ⑥ A relációban lévő modell is elérhető ekkor a másik controller útvonalán keresztül `/users/1/tasks/1`

- ⑥ Új controller akció felvétele:

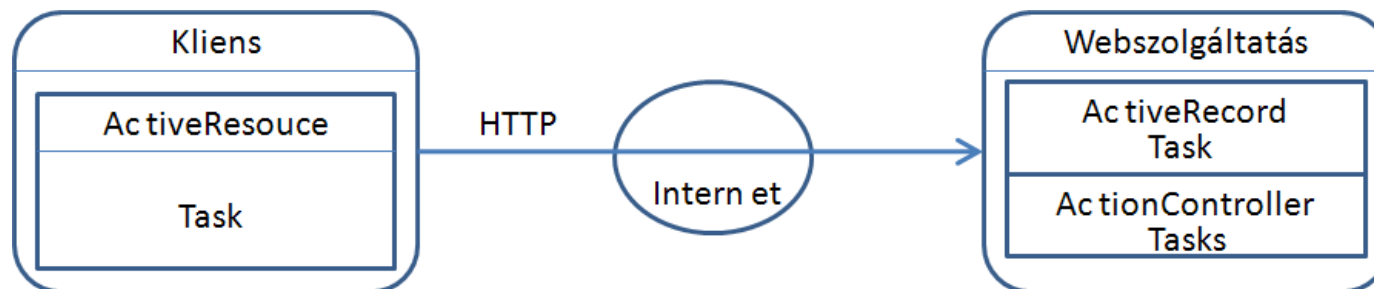
```
resources :users do
  get 'admin', on: :member
  get 'commenters', on: :collection
end
```

- ⑥ Ha egyértelmű, a path helper is elhagyható:

```
link_to 'Task details', @task
```

# RESTful webszolgáltatás 1

- ⑥ Szolgáltatás: „A szolgáltatás egy absztrakt tevékenységek végrehajtásának képességét reprezentáló erőforrás, amely koherens funkcionalitást mutat mind a szolgáltató, mind a felhasználó entitások számára.”
- ⑥ Webszolgáltatás: XML alapú hálózati kommunikáció alkalmazások között meghatározott interfész alapján





# RESTful webszolgáltatás 2

- ⑥ HTTP kérésre XML válasz
- ⑥ A kontroller akció válaszol a kérés formátuma alapján (pl. /tasks/1.xml)

```
# respond_with használata esetén  
# respond_to :html, :xml
```

```
def show  
  @task = Task.find params[:id]  
  respond_to do |format|  
    format.html  
    format.xml { render :xml => @task }  
    # format.xml { render xml: @task }  
  end  
  # respond_with @task  
end
```

# ActiveResource 1

- ⑥ `gem 'activeresource'`
- ⑥ Az ActiveResource erőforrás megegyezik az ActiveRecord erőforrással
- ⑥ Hivatkozás a távoli erőforrásra: `self.site`
- ⑥ Hivatkozás másik erőforrásra, például az User ActiveResource-ból  
`self.site=http://localhost:3000/tasks/:task_id`
- ⑥ A szerver oldali validációs hibából HTTP hibaüzenet lesz
- ⑥ Hibák: a HTTP válasz státusz kódjának megfelelő kivétel dobódik,  
400-499: `ActiveResource::ClientError`,  
500-599: `ActiveResource::ServerError`,  
30x: `ActiveResource::Redirection`
- ⑥ Hibás rekord: `user.errors.on :username`

# ActiveResource 2

```
require 'active_resource'
class Task < ActiveResource::Base
  self.site='http://localhost:3000'
  self.user='me'
  self.password='titok'
  #self.ssl_options ...
end
tasks=Task.find :all # GET
task = Task.find :first
t = Task.new(:deadline=>Time.now)
t.save # POST
t.user=1
t.save # PUT
t.destroy # DELETE
```

## ActiveResource 3

- 6 Az ActiveResource saját sémát is definiálhat string, integer és float attribútumokból

```
class Message < ActiveResource::Base
  schema do
    integer 'from'
    integer 'to'
    string 'message'
    string 'date'
  end
end
```

# Javascript Rails-ben

- ⑥ Az alapértelmezett keretrendszer Rails 3.1 előtt a Prototype, Rails 3.1-től a jQuery
- ⑥ A Prototype kiiktatható a projekt létrehozásakor a `--skip-prototype` kapcsolóval
- ⑥ A JavaScript API Rails 3-ban lecserélhető az `config/application.rb` fájlban

```
config.action_view.JavaScript_expansions[:defaults] =  
  %w(jQuery rails application)
```

- ⑥ Az `app/assets/javascripts` könyvtár tartalma:
  - △ Javascript keretrendszer: `prototype.js` vagy `jquery.js`
  - △ A Rails Javascript keretrendszer: `rails.js`
  - △ Saját Javascript források

# AJAX támogatás Rails-ben

- ⑥ Rails JavaScript attribútumok (`rails.js`)
  - △ `data-remote`: AJAX kérés
  - △ `data-method`: a használandó REST HTTP metódus
  - △ `data-confirm`: ellenőrző kérdés, lásd scaffold delete akció
  - △ `data-disable-with`

# AJAX hívás létrehozása

1. A `data-remote` attribútum `true`-ra állítása a form vagy a link HTML elemében: `:remote => true`
2. A kontroller akciók `respond_to` blokkjaihoz hozzáadjuk a Javascript formázást

```
respond_to do |format|
  format.html { render :action => "new" }
  format.js   #{ render :nothing => true }
end
```

3. A Rails válasza az AJAX kérésre egy Javascript betöltése, amelyet a nézetek között helyezünk el a kontroller akciónak megfelelő néven.
4. A hívás után a visszaadott Javascript (például `$('#i').html('...')`) végrehajtásra kerül.

# Rails AJAX események

- ⑥ A Rails hat darab saját Javascript esemény definiál a `bind` számára:
  - △ `ajax: beforeSend`: az AJAX hívás előtt hajtódik végre
  - △ `ajax: success`: sikeres AJAX hívás esetén hajtódik végre
  - △ `ajax: error`: sikertelen AJAX hívás esetén hajtódik végre
  - △ `ajax: complete`: `ajax: success` vagy `ajax: error` után hajtódik végre



# ActionMailer 1

- ⑥ Az ActionMailer email üzenetek küldését teszi lehetővé Rails alkalmazásból
- ⑥ A kliensek a `app/mailers` alatt jönnek létre a `rails generate mailer` parancs hatására
- ⑥ A levél összeállítása:
  - △ A `default` hash: az összes levélre közös mezők
  - △ A `mail` levélküldő metódus: az aktuális levél mezői
  - △ A `headers` hash: speciális fejrészek beállítása,  
`headers[ 'MIME-Version' ] = '1.0'`
  - △ Az `attachments` csatolmányokat fűz a levélhez,  
`attachments[ 'me.png' ] = File.read( 'me.png' )`
- ⑥ Ezek elemei a levél fejléceit állítják, például `:from`, `:to`, `:subject`
- ⑥ Több TO, CC, stb. elem esetén vesszővel elválasztott listát kell értéként átadni

# ActionMailer 2

- ⑥ A mailer kontrollere az `app/mailers` mailer osztálya, amely mailer akciókat tartalmaz (pl. `def notification`)
- ⑥ A mailer nézetei, vagyis az üzenetek törzse az `app/views` alá kerülnek a mailer nevével megegyező könyvtárba, és a mailer akcióknak megfelelő néven
- ⑥ Például `notification.html.erb` HTML törzsű levél esetén vagy `notification.text.erb` egyszerű szöveg esetén
- ⑥ Ha több nézet is jelen van, akkor multipart MIME üzenetet hoz létre a Rails
- ⑥ Az alapértelmezett nézet felüldefiniálható:

```
mail() do |format|  
  format.html {render 'me'}  
  format.text {render 'body'}  
end
```

# ActionMailer 3

## 6 Mailer:

```
class MyMailer < ActionMailer::Base
  default :from => 'admin@localhost'
  def notification(user)
    @user = user
    mail :to => user.email, :subject => 'Welcome'
  end
end
```

## 6 Nézet (views/notification.text.erb)

Notification ...

# ActionMailer 4

## 6 Csatolmányok befűzése

- △ **Inline:** `attachments.inline['me.png']`
- △ **Linként:** `<%= image_tag attachments['me.png'].url %>`
- △ **Kódolva:**

```
attachments['me.png']={:mime_type => 'application/pdf',  
  :encoding => 'pdf', :content => File.read('me.pdf')}
```

## 6 A levél elküldése egy controller akcióban

```
MyMailer.notification.deliver
```

# ActionMailer 5

## 6 Levél fogadása

1. A mailer osztályban egy fogadó metódus (pl. `receive`) definíciója
2. A levelező portálon a levél továbbításának beállítása  
a `rails runner Mailer.receive(STDIN.read)` alkalmazásnak

# ActionMailer 6

- ⑥ A levelező kiszolgáló paramétereinek beállítása konfigurációs változókkal a `config` könyvtár egyik alkalmas fájljában (`application.rb`, `development.rb`, `boot.rb` stb.)

- ⑥ Például

```
config.action_mailer.delivery_method = :smtp #:sendmail
config.action_mailer.smtp_settings={
  :address => 'mail.tmit.bme.hu',
  :port => '25' }
```

# ActiveJob 1

- ⑥ Rails 4.2-től elérhető API feladatok aszinkron és/vagy ütemezett végrehajtására
- ⑥ Tipikus alkalmazás: levélküldés, takarítás
- ⑥ Létrehozás:  

```
rails generate job my_job
```
- ⑥ Konfiguráció: `queue-adapter` kiválasztása, amivel késleltetés, végrehajtási sor, prioritás, időtűllépés és ismétlés konfigurálható

```
ActiveJob::Base.queue_adapter = :inline
```

# ActiveJob 2

## 6 Feladat definíciója:

```
class MyJob < ActiveJob::Base
  queue_as :my_jobs

  def perform(my_job)
    my_job.do
  end
end
```

## 6 Feladat végrehajtása:

```
MyJob.perform_later my_job
MyJob.set(wait_until: Date.tomorrow.noon).perform_later(my_job)
MyJob.set(wait: 1.week).perform_later(my_job)
```