



# ***Tesztelés, validáció***

Kovács Gábor

`kovacsg@tmit.bme.hu`

BME-TMIT

# ***Tesztelés Rails-ben***

- ⑥ Feladata a Rails alkalmazás integritásának megőrzése a fejlesztési iterációk során
- ⑥ A webalkalmazás létrehozásakor automatikusan létrejön a teszt infrastruktúra

# Tesztípusok Rails-ben

- ⑥ A Rails a következő tesztípusokat különbözteti meg:
  - △ Egységteszt (unit test): feladata a modellek tesztelése
  - △ Funkcionális teszt (functional test): feladata egyetlen controller akció tesztelése
  - △ Integrációs teszt (integration test): feladata a kontrollerek együttműködésének tesztelése
  - △ Teljesítményteszt (performance test): feladata a Rails alkalmazás skálázhatóságának és terhelhetőségének ellenőrzése
  - △ Kérések irányításának (útvonalak) tesztelése: feladata a beérkező kérések controller akciókhoz rendelésének ellenőrzése

# Tesztadatok 1

- ⑥ A tesztelés során használt modell objektumok szerializált változata
- ⑥ A `test/fixtures` könyvtárban található
- ⑥ Alapértelmezés szerint YAML formátumban, de lehet CSV is
- ⑥ A `test_helper.rb` az összes modell tesztadatát elérhetővé teszi (`fixtures :all`) egység- és funkcionális tesztekhez
- ⑥ Ruby kódrészlet beágyazható
- ⑥ Betöltés az adatbázisba: `rails db:fixtures:load`
- ⑥ Az `id` és a `timestamp` attribútumok automatikusan töltődnek, de felüldefiniálhatóak

## Tesztadatok 2

### users.yml

```
<% SALT= '...' %>
<% pass='hello' %>
me:
  email: kovacsg@tmit.bme.hu
  name: Gabor Kovacs
  salt: <%= SALT %>
  encrypted_password: <%= User.encrypt pass, SALT %>
```

## Tesztadatok 3

### recipes.yml

```
one:  
  name: Forro viz  
  description: Forralj fel vizet  
  user: me  
  cook_time: 1  
  complexity: 1
```

# *A tesztadatbázis előkészítése*

1. A tesztadatbázis létrehozása
2. Migráció
3. A tesztadatok betöltése az adatbázisba

```
RAILS_ENV='test' rails db:fixtures:load
```

# Tesztfájlok struktúrája

```
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  # Replace this with your real tests.
  test "the truth" do
    assert true
  end
end
```

- a `test "the truth" do` a következő metódusdefinícióval ekvivalens:

```
def test_the_truth
```



# Állítások

- ⑥ Az `assert` egy objektum vagy kifejezés igaz voltát ellenőrzi
- ⑥ Egyes kifejezéstípusok saját `assert_` változattal rendelkeznek egyedi paraméterezéssel, például
  - △ Ruby specifikus ellenőrzések: (nem) egyenlőség, (nem) azonosság, nil, kivétel (nem) dobás, típusellenőrzés stb.
  - △ Rails specifikus ellenőrzések: útvonal felismerése, útvonal paraméterezése, HTTP válasz státusz kód, átirányítás, nézet használata renderelés során, modell példány validitása
- ⑥ Minden állítás utolsó paraméterének átadható egy string típusú üzenet

# Tesztek végrehajtása

## ⑥ A tesztek megírása

- △ Hivatkozás a tesztadatokra hash-en keresztül: `users(:me)`

## ⑥ A teszteset végrehajtása

- △ Egyenként: `bin/rails test test/models/user_test.rb:5`
- △ Mindet: `bin/rails test`
- △ vagy `bin/rails test test/models`

## ⑥ A teszteset befejeződik, ha

- △ hiba lép fel (E – error)
- △ valamelyik állítás (`assert`) nem áll fenn (F – failure)
- △ teljesül az (összes) állítás (.)

# *Mit egységteszteljük?*

- ⑥ A modell osztály minden egyes metódusát
- ⑥ A modell osztály minden egyes validációját

# Mit validálhatunk a modellben?

- ⑥ Adatbázisbeli kényszerek helyett
- ⑥ Futásidőben ellenőrizhetjük egy modell objektum egy példányváltozójának értékét (`validates...`)
  - △ Példányváltozó inicializáltsága (`_presence_of`)
  - △ Példányváltozó egyedisége (`_uniqueness_of`)
  - △ Jelszó ellenőrzés (`_confirmation_of`)
  - △ Példányváltozó értéke hosszának ellenőrzése (`_length_of`)
  - △ Példányváltozó értékének összevetése egy reguláris kifejezéssel (`_format_of`)
  - △ Szolgáltatási feltételek elfogadása (`_acceptance_of`), nem feltétlenül kell adatbázis attribútumot létrehozni számára
  - △ Továbbá: adatbázis relációk validitásak (`_associated`), attribútum összevetése egy blokk által visszaadott értékkel (`_each`), enumerációba tartozás (`_inclusion, _exclusion`) vizsgálata, számtípus (`_numericality`)

# Validáció Rails 3-tól

- ⑥ A validációk attribútumonként
- ⑥ A `validates_with` metódussal a validációk egy `validate` metódust tartalmazó `ActiveModel::Validator` leszármazott osztályba helyezhetők át
- ⑥ Validációs hibák: `errors` modell attribútum

```
class User < ActiveRecord::Base
  validates :email,
    {
      presence: true,
      uniqueness: true,
      length: { :minimum => 3, :maximum => 255 },
      format: { :with => /^[^\s]+@((?:[-a-z0-9]+\.)+[a-z]{2,})$/i }
    }
  validates :name, presence: true
  validates :password, confirmation: true, :if => :password_required?
end
```

# Egységteszt példa

## 6 A `validates_presence_of :email` működésének tesztelése

```
class UserTest < ActiveSupport::TestCase
  test "email null" do
    u = User.new
    assert !u.save, "Houston, we have a problem"
  end
end
```

# Milyen céllal írjunk funkcionális tesztet?

- ⑥ HTTP státusz kód megfelelő-e?
- ⑥ Az átirányítás a jó oldalra vitt-e át?
- ⑥ Session elemek megfelelőek-e?
- ⑥ A megfelelő flash üzenet jelenik-e meg?
- ⑥ Hitelesítést igénylő akciók, például a bejelentkezés sikeres volt-e?
- ⑥ A megfelelő modell példány jelent-e meg a nézetben?

# *Funkcionális teszt példa*

## 6 Az users kontroller new akciója válaszol-e

```
class UsersControllerTest < ActionController::TestCase
  test "should get new" do
    get :new
    assert_response :success #:redirect, :missing, :error
  end
  ...
end
```



# *Funkcionális tesztek kérés típusai*

- ⑥ A következő HTTP metódusok használhatók: `get`, `post`, `put`, `head`, `delete`
- ⑥ Ezek a következő paraméterekkel rendelkeznek:
  - △ Az akció URI-ja, például `new_user_url`
  - △ Opcionálisan a `params` hash
  - △ Opcionálisan a `headers` hash
  - △ Opcionálisan az `env` hash
  - △ AJAX kérésekhez az `xhr` értéke `true`
  - △ Az `as` értéke a Content-type-ot adja meg, alapértelmezés szerint az értéke `:json`
- ⑥ A `cookies`, a `session` és a `flash` hash-ek értékei megelőző HTTP kérésekkel állítandók be

# Funkcionális tesztek hash objektumai és példányváltozói

## 6 Hash-ek

- △ `assigns`: a megjelenítendő nézet számára elérhető controller példányváltozók hash-e
- △ `cookies`: a süti hash-e
- △ `flash`: az aktív flash üzenetek hash-e
- △ `session`: a session hash

## 6 Példányváltozók

- △ `@controller`: a kérést feldolgozó controller
- △ `@request`: a kérés
- △ `@response`: a válasz

# Nézetek tesztelése 1

- ⑥ Annak ellenőrzése, hogy a HTML oldalon belül egy komponens megfelelő állapotban van-e
- ⑥ Az `assert_select` az összes illeszkedő csomópontot és azok leszármazottját kiválasztja a HTML DOM-ból
- ⑥ Az `assert_select` egyenlőséget vizsgál az első, minta által kiválasztott HTML elemek és a második paraméter között
  - △ `true`, legalább egy elemre illeszkedik a mintára
  - △ `false`, ha egy elem sem illeszkedik a mintára
  - △ `String`, ha legalább egy elem illeszkedik a stringre
  - △ `Integer`, ha pontosan ennyi az illeszkedő elemek száma
  - △ `Range`, ha az illeszkedő elemek száma az intervallumban van

## Nézetek tesztelése 2

- ⑥ Több feltétel fennállása egyszerre is ellenőrizhető, ekkor a második paraméter egy hash
  - △ :`text`, az erre a String értékre illeszkedő elemek
  - △ :`html`, az erre a HTML kódrészletre illeszkedő elemek
  - △ :`count`, pontosan ennyi a kiválasztott elemek száma
  - △ :`minimum`, legalább ennyi a kiválasztott elemek száma
  - △ :`maximum`, legfeljebb ennyi a kiválasztott elemek száma

## Nézetek tesztelése 3

- ⑥ Az `assert_select` beágyazható, de akkor az előzőleg kiválasztott halmazra fut csak le
- ⑥ A teendők nézet egy és hat közötti számú teendőt listáz-e ki a táblázatban

```
assert_select "table" do
  assert_select "tr", 1..6
end
```

- ⑥ Bőbeszédű változatban ugyanez

```
assert_select "table" do |tables|
  tables.each do |table|
    assert_select table, "tr", 1..6
  end
end
```

# Integrációs teszt 1

- ⑥ Célja az egy vagy több kontrolleren áthaladó böngészés folyamának tesztelése
- ⑥ Az integrációs tesztek explicit módon létre kell hozni  

```
rails generate integration_test send_post
```
- ⑥ Az alap eszköztára megegyezik a funkcionális tesztek eszköztárával

# Integrációs teszt 2

## 6 Integrációs teszt helperek

- △ `https!` és `https?`: biztonságos kapcsolat imitálása és ellenőrzése
- △ `open_session`: új session létrehozása
- △ `host!`: a kliens gépnevének beállítása
- △ `redirect?`: az utolsó választ átirányítás adta-e meg
- △ `follow_redirect`: követi az átirányítást, ha volt, ha nem kivételt dob
- △ `request_via_redirect`: paramétereket ad át egy átirányításnak

# Integrációs teszt 3

```
fixtures :all
test "send post with image"
  get url_for(:controller=>'users',:action=>:show, :id=>users(:me).id)
  assert_response :success
  post_via_redirect "/sessions/create",
    :username=>users(:me).username,
    :password=>users(:me).password
  assert_equal '/users/'+users(:me).id.to_s, path
  get '/posts/'+posts(:one).id.to_s
  assert_response :success
  upload_file=fixture_file_upload('files/1.txt','text/plain')
  post url_for(:controller=>'posts',:action=>'create',
    :id=>posts(:one).id.to_s),
    { :attachment=>upload_file,
      :text => posts(:one).text,
      :author_id => users(:me).id,
      :user_id => users(:me).id }
  s = Attachment.find_by_post_id posts(:one).id
  assert_not_nil s.filename
  assert File.exists?("../../public/data/#{s.id.to_s}")
end
```



# Teljesítményteszt 1

- ⑥ A webalkalmazás sebesség- és memóriaproblémái forrásának felderítésére
- ⑥ Gem függőség: rails-perftest és ruby-prof
- ⑥ Generálás:  
`rails generate performance_test index`
- ⑥ Végrehajtás: `rails test:profile`
- ⑥ Egyszeri tesztelés:  
`perftest profiler 'User.first'`

# Teljesítményteszt 2

```
require 'test_helper'
require 'rails/performance_test_help'

class IndexTest < ActionDispatch::PerformanceTest
  # Refer to the documentation for all available options
  # self.profile_options = { :runs => 5, :metrics => [:wall_time, :memory]
  #                       :output => 'tmp/performance', :formats => [:flat] }
  def test_homepage
    get '/'
  end
end
```

# Teljesítményértékelés

- ⑥ Gyors statisztika a weboldalról teljesítmények végrehajtásával
- ⑥ Végrehajtás: `rails test:benchmark`
- ⑥ Egyszeri tesztelés:  
`perftest benchmarker 'User.first'`

# Elő- és utófeltételek

- 6 A teszt lefutása előtt és után végrehajtandó kódrészlet definiálható a `setup` és `teardown` metódusokkal

```
def setup
  @me=users(:me)
end
```

```
def teardown
  @me=nil
end
```

# Útvonalak tesztelése

- ⑥ Az `assert_routing` metódus
- ⑥ Ellenőrizhetjük, hogy a második paraméterként megadott hash az első paraméterként megadott útvonalat adja-e

```
assert_routing '/users/show/1',  
  {:controller=>'users',  
   :action=>'show',  
   :id=>1}
```