



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

Teszt típusok Rails-ben

- ⑥ A Rails a következő teszt típusokat 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

Teszt adatok 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 teszt adatait elérhetővé teszi (`fixtures :all`) egység- és funkcionális tesztekhez
- ⑥ Ruby kódrészlet beágyazható
- ⑥ Betöltés az adatbázisba: `rake db:fixtures:load`
- ⑥ Az `id` és a `timestamp` attribútumok automatikusan töltődnek, de felüldefiniálhatóak

Teszt adatok 2

users.yml

```
<% SALT= '...' %>
<% pass='hello' %>
me:
  neptun: oweyoa
  email: kovacsg@tmit.bme.hu
  student: false
  salt: <%= SALT %>
  password: <%= pass%>
  encrypted_password: <%= User.encrypt pass, SALT %>
```

Teszt adatok 3

solutions.yml

```
first:
  user: me #referencia hash kulccsal
  task: one
  version: 2
  file_name: 1.txt
  late: false
  missing: false
  comment: OK
```

A teszt adatbázis előkészítése

1. A teszt adatbázis létrehozása
2. Migráció
3. A teszt adatokat betöltése az adatbázisba

```
RAILS_ENV='test' rake test:load:fixtures
```

Teszt fájlok 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_trush
```


Állítások

- ⑥ Az `assert` egy objektum vagy kifejezés igaz voltát ellenőrzi
- ⑥ Egyes kifejezés típusok 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 teszt adatokra hash-en keresztül:
`users(:me)`
- ⑥ A teszt végrehajtása
 - △ Egyenként:
`ruby user_test -n test_neptun_null`
 - △ Mindet: `rake test`
- ⑥ A teszt eset 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 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 validitása, attribútum összevetése egy blokk által visszaadott értékkel, enumerációba tartozás vizsgálata

Egységteszt példa

6 A validates_presence_of :neptun működésének tesztelése

```
require File.dirname(__FILE__)+'../test_helper'
class UserTest < ActiveSupport::TestCase
  test "neptun 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

```
require File.dirname(__FILE__)+'../test_helper'
class UsersControllerTest < ActionController::TestCase
  test "should get new" do
    get :new
    assert_response :success
  end
  ...
end
```

Funkcionális tesztek kérés típusai

- ⑥ A következő HTTP metódusok használhatók: `get`, `post`, `put`, `head`, `delete`
- ⑥ Ezek négy paraméterrel rendelkeznek:
 - △ Az akció neve, például `:new`
 - △ A `params` hash
 - △ A `session` hash
 - △ A `flash` hash

Funkcionális tesztek hash objektumai és példányváltozói

6 Hash-ek

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

6 Példányváltozók

- △ `@controller`: a kérést feldolgozó kontroller
- △ `@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 feladatok nézet egy és hat közötti számú feladatot tartalmaz-e

```
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 task_submission
```
- ⑥ Az alap eszköztára megegyezik a funkcionális tesztekével

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

6 Példa

```
test "task submission"
  get url_for(:controller=>'session', :action=>:new)
  assert_response :success
  post_via_redirect "/sessions/create",
    :neptun=>users(:me).neptun,
    :password=>users(:me).password
  assert_equal '/users/show/' + users(:me).id.to_s, path
  get '/tasks'
  assert_response :success
```

Integrációs teszt 4

6 Példa folytatás

```
get '/solutions/new/' + tasks(:one).id.to_s
assert_response :success
upload_file=fixture_file_upload('files/1.txt', 'text/plain')
post url_for(:controller=>'solutions', :action=>'new',
  :id=>tasks(:one).id.to_s), :upload=>{:file=>upload_file}
s = Solution.find_by_task_id_and_user_id tasks(:one).id,
  users(:me).id
assert_not_nil s.file_name
assert File.exists?(' ../../public/data/1.txt')
end
```


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 }
```