Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2015. november 24.

Mielőtt a gyakorlat témájával, a teszteléssel foglalkoznánk, teszünk egy kiegészítést az előző gyakorlathoz. Mivel a szavak nagyságrendekkel több lehet, mint amennyit a nézeten áttekinthetően meg tudunk jeleníteni egymás alatt egy táblázatban, ezért e tartalmak tekintetében megvalósítjuk a tördelés funkciót. A tördeléshez a `will_paginate` Ruby függvénykönyvtárt használjuk, a `Gemfile`-ba beszúrjuk az alábbi sort, majd kiadjuk a `bundle update` parancsot:

```
gem 'will_paginate'
```

Három módosítást kell a tördelésre elvégeznünk. A nézethez hozzá kell adnunk a lapozó linkeket, a kontrollerben csak az aktuális oldalon megjelenítendő töredékeket kérjük el a modelltől. A modellben pedig definiáljuk a tördelés metódusát.

A szavak `index` nézetében tördelve megjelenítjük az eddig rögzített szavakat. A navigációs linkeket a `will_paginate` helper helyezi ki az oldalra.

```
<%= will_paginate @solutions %>
```

Az `index` akcióban a nézet számára rendelkezésre bocsátjuk a `page` paraméternek megfelelő megoldáshalmazt a `words` példányváltozón keresztül.

```
@words = Word.get_page(params[:page])
```

Csak két megoldásunk van az adatbázisban, ezért egy oldalon csak egy megoldást jelenítünk meg. A tördelt adatok lekérdezését a modellben valósítjuk meg. Az oldalon megjelenítendő rekordok számát a `:per_page` értéke, az oldalt a `:page` értéke adja meg.

```
class Word < ActiveRecord::Base
  def self.get_page(page)
    paginate per_page: 5, page: page
```

```
  end
end
```

Laborunk témája az elkészített rendszer tesztelése. A Rails tesztkörnyezete három elemből áll:

- tesztadat-definíció, ami a tesztadatbázisba betöltendő rekordokat definiája,
- teszteset-specifikáció,
- tesztvégrehajtó környezet, amely automatikusan végrehajtja a teszteseteket a tesztadatokat felhasználva.

A tesztelés előkészítésére először tesztadatokat definiálunk, amelyekre a teszteseteinkben hivatkozni fogunk. Ezeket a Rails alkalmazásunk `test/fixtures` könyvtárában helyezzük el. Az egységtesztek és funkcionális tesztek számára ezeket az adatokat a minden egyes teszt eset elején hivatkozott `test/test_helper.rb` fájl fogja elérhetővé tenni, integrációs tesztek esetén pedig magunk töltjük be.

Definiáljuk az alábbi két `User` példányt az `users.yml` fájlban. A modellt szkripttel hoztuk létre így ott már láthatunk kezdeti adatokat, amiket módosítunk. Az időközben egy migrációban átnevetett titkosított jelszó attribútum megadához a modell osztály `encrypt` osztálymetódusát hívjuk segítségül. A felhasználóspecifikus a titkosítás során használt `salt` attribútumot külön-külön definiáljuk. A később migrációval módosított, illetve hozzáadott attribútumokat magunknak kell módosítanunk, illetve hozzáadnunk a struktúrához, ilyen például a `salt`..

```
me:
  username: kovacsg
  salt: a
  encrypted_password: <%= User.encrypt 'titok', 'a' %>
  email: kovacsg@mail.bme.hu
  birthdate: 2015-10-01 10:44:01
  type: 0

valaki:
  username: Vala Ki
  salt: b
  encrypted_password: <%= User.encrypt 'titok', 'b' %>
  email: valaki@mail.bme.hu
  birthdate: 2015-10-01 10:44:01
  type: 0
```

A tesztadatbázisunkban több szóra lesz szükségünk, ezeket a `words. yml` fájlban definiáljuk egy ciklussal.

```
<% ('a'..'z').each do |l| %>
letter_<%= l%>:
  word: <%= l %>
  description: <%= l %>
  lang: hu
  pronounce: <%= l %>
<% end %>
```

Egy tesztadatot veszünk fel a tesztek modellje számára (`tests.yml`). A teszt több-egy relációban áll a felhasználók modellel, és a kulcsunk itt megfelel a Rail konvencióinak. Ez lehetővé teszi számunkra a másik tesztadat kulcsával való hivatkozást. A `questions` attribútumot a modell osztályban szereplő minta alapján inicializáljuk. Ha a YAML fájlunkban egy ilyen kulcshoz a hivatkozott tesztadat egy kulcsát rendeljük, a Rails feloldja a kapcsolatot.

```
one:
  user: me
  questions: <%= Word.all.sample(20).collect do |x| x.id end.
      to_json %>
  solution: <%= [] %>
```

Töltsük be a teszt környezet adatbázisába ezeket az adatokat ügyelve arra, hogy a környezetként a teszt környezet legyen beállítva.

```
RAILS_ENV='test' rake db:test:prepare
RAILS_ENV='test' rake db:migrate
RAILS_ENV='test' rake db:fixtures:load
```

Ha MySQL konzolon megnézzük a betöltött adatokat, azt láthatjuk, hogy az `id` attribútum véletlen értékkel töltődött fel, ahol azt explicite nem definiáltuk, az időpecsétek pedig a betöltés időpontját vették fel.

Először írjunk egységteszteket! Az egységtesztek a modell osztályok metódusait és validációit hivatottak ellenőrizni. Az egyégteszteket a Rails projektünk `test/models` (Rails 4 előtt `test/unit`) könyvtárában találjuk. Minden egyes modell létrehozása után automatikusan létrejön hozzá egy ahhoz kapcsolódó teszt osztály itt.

Írjunk teszteseteket, amelyek a `User` modellünk

```
  validates :username,
    {
      :presence => true,
      :uniqueness => true,
      :length => {:in => 4..18},
    }
```

validációinak megtörténtét ellenőrzik. Először létrehozunk egy új felhasználót mindenféle inicializáció nélkül, majd megpróbáljuk eltárolni az adatbázisba. A feltételezésünk az, hogy a mentésnek nem szabad sikerülnie.

Írjunk még egy tesztesetet, ami a modell egy példánymetódusát ellenőrzi. Ez az általunk írt `encrypt` osztálymetódust teszteli egy ismert tesztadat alapján.

```
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  # def test_the_truth
  test "the truth" do
     assert true
  end

  test "cannot save user without username" do
    u = User.new
    assert !u.save, "Houston, we have a problem"
  end

  test "cannot save user with existing username" do
    me = users(:me)
    u = User.new username: me.username
    assert !u.save
  end

  test "check if encryption is deterministic" do
    assert_equal User.encrypt('titok','a'), users(:me).
       encrypted_password
  end
end
```

A szavak modell osztályában egy függvényt definiáltunk, ami a tördelést ellenőrzi. Nézzük meg, hogy a számossága megfelelő-e.

```
require 'test_helper'

class WordTest < ActiveSupport::TestCase
  # test "the truth" do
  #   assert true
  # end
  test "count words per page" do
    assert_equal 5, Word.get_page(1).size
  end
end
```

Miután a webszerverünket újraindítottuk úgy, hogy az a tesztkörnyetetet használja, a tesztesetet futtatva meggyőződhetünk róla, hogy tényleg nem történik meg a mentés. Ha mégis sikerülne, a megadott hibaüzenetnek kell megjelennie. Egyúttal böngészőből is kipróbálhatjuk, hogy működik a bejelentkezés.

```
kovacsg@debian:~/gyakorlat/test/models$ rake test:models
(in /home/kovacsg/gyakorlat)
Run options: --seed 53883

# Running:

.....

Finished in 0.306865s, 16.2938 runs/s, 16.2938 assertions/s.

5 runs, 5 assertions, 0 failures, 0 errors, 0 skips
```

A funkcionális tesztek a kontrollerek és a nézetek helyes működését ellenőrzik, a `test/controllers` (Rails 4 előtt `test/functional`) könyvtárban találhatók. Alapértelmezés szerint minden egyes a létrehozáskor megadott kontroller akcióra létrejön egy az akció sikeres megjelenítését ellenőrző tesztteset.

Először egészítsük ki a `SessionsControllerTest` tesztet, amely jelenleg két tesztesetet tartalmaz. A funkciót a `me` kulcshoz tartozó felhasználó tesztadatain végezzük el, ehhez a `setup` metódusban inicializálunk egy példányváltozót az összes teszteset számára. Az első a bejelentkezést, a második a kijelentkezést teszteli, nevezzük át a teszteket ennek megfelelően! A bejelentkezés eseményt a `SessionController create` metódusát használja, a kijelentezés esemény pedig a `destroy` metódust. A tesztben a `post` metódust használjuk, amelynek első paramétere a teszelendő akció, a második a HTTP kérés paraméterei, a harmadik pedig a kérés előtti `session` paraméterek értékeit tartalmazó hash. A sikeres bejelentkezés tesztben a tesztadatokban szereplő felhasználónév, jelszó párost küldjük el, és azt feltételezzük, hogy visszairányítódunk a belépés előtti nézetre, a `:user` session paraméter értéke nem `nil`, ráadásul megegyezik a tesztadat azonosítójával. A kilépés tesztben HTTP kérés paramétereket nem adunk meg, tehát a második paraméter `nil`, azonban harmadik paraméterként beállítjuk a `:user` session paramétert azt imitálva, hogy van bejelentkezett felhasználónk. Válaszként átirányítást várunk az aktuális oldalunkra, és azt, hogy a session paraméter kinullázódik.

```
require 'test_helper'

class SessionsControllerTest < ActionController::TestCase
  test "login" do
```

```
    post :create, { username: @u.username, password: 'titok' }
    assert_response :redirect
    assert_not_nil session[:user]
  end

  test "logout" do
    get :destroy, nil, { user: @u.id }
    assert_response :redirect
    assert_nil session[:user]
  end
end
```

A tesztesetünk kész van azonban nem futhat le sikeresen ugyanis a kontrollerben szereplő `redirect_to :back` átirányításban nem definiált a `:back` értéke. Ezt a Rails a javascript history elérhetetlensége miatt a `HTTP_REFERER` nevű HTTP kérés paraméterből veszi, így ezt minden egyes tesztesetben be kell állítanunk. Ezt hatékonyan a `setup` metódusban tehetjük meg, amely minden teszteset előtt lefut.

```
class SessionsControllerTest < ActionController::TestCase
  setup do
    @u = users(:me)
    @request.env['HTTP_REFERER']='/say/hello'
  end

  teardown do
  end
end
```

A `WordsControllerTest`-et szkripttel hoztuk létre, ami automatikusan generálta a tesztadatokat, a teszteseteket és a tesztesetek törzsét. A tesztadatok kulcsait módosítottuk, így az azokra való hivatkozást frissítenünk kell a `setup`-ban.

A felhasználók kontrollere esetén a műveleteink jó része csakis bejelentkezett felhasználó számára lehetséges, e két tesztfájlban a `get` és `post` műveletek harmadik paraméterében inicializálunk kell a `sessions` hasht. Az alábbi kódrészlet erre mutat egy példát, azonban ezt a kontrollerekben szereplő `before_filter` miatt minden akcióban el kell végeznünk, ráadásul egyes a műveletek csak oktató típusú felhasználók számára lehet elérhető. A teszteseteink legyenek az automatikusan generáltak kijavítva.

```
require 'test_helper'

class UsersControllerTest < ActionController::TestCase
  test "should get new" do
    get :new
    assert_response :success
```

```
  end

  test "should get edit" do
    get :edit, { id: users(:valaki).id }, { user: users(:me).id }
    assert_response :success
  end

  test "should get show" do
    get :show, { id: users(:valaki).id }, { user: users(:me).id }
    assert_response :success
  end

  test "should get index" do
    get :index
    assert_response :success
  end

end
```

A funkcionális teszteseteket a `rake test:functionals` szkripttel futtathatjuk.

```
kovacsg@debian:~/gyakorlat/test/controllers$ RAILS_ENV='test'
   rake test:functionals
(in /home/kovacsg/gyakorlat)
Run options: --seed 34590

# Running:

..............

Finished in 0.725708s, 19.2915 runs/s, 31.6932 assertions/s.

14 runs, 23 assertions, 0 failures, 0 errors, 0 skips
```

A tesztek harmadik típusa az integrációs teszt, amellyel egy böngészési folyamatot ellenőrzünk. Készítsünk egy tesztet egy szóteszt kitöltésére!

```
kovacsg@debian:~/gyakorlat/test$ rails g integration_test do_test
      invoke  test_unit
      create    test/integration/do_test_test.rb
kovacsg@debian:~/gyakorlat/test$
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `do_test_test.rb` nevű állomány, ahol az integrációs tesztünk metódusait helyezzük el.

A tesztadatok a `test_helper` fájlra való hivatkozás miatt automatikusan elérhetők. A teszt során a `me` kulcsú felhasználó kezdjen hozzá egy teszt kitöl-

téséhez. A böngészési folyamatot úgy definiáljuk, hogy az a bejelentkezéstől jusson el legalább egy szó kitalálásáig.

Az integrációs tesztesetet a funkcionális teszteknél megismert tesztesetekből mint tesztlépésekből tevődik össze.

Az első lépésben egy be nem jelentkezett felhasználó betölt egy véletlen oldalt, amin elérhető a bejelentkezési form. Itt annyit feltételezünk, hogy az oldal betöltődik, és az oldalon HTML nézetének forrásában van egy `Login` értékű `legend` HTML elem.

A második tesztlépés a bejelentkezés. Az aktuálisan megjelenített nézeten található egy form, amelyen kereszül `username` és `password` kérés paramétereket juttathatunk el a `/sessions/create` kontroller akciónak. Mivel az akció végén minden esetben egy átirányítás áll, `post` helyett a `post_via_redirect` metódust kell használnunk, ami a kérés után követi az összes átirányítást. Ennek a metódusnak a paraméterlistája eltér a megszokottól, vagyis az akció, paraméterekek, session, flash négyestől, itt a második paraméter ugyanúgy a HTTP kérés paraméterlistája, azonban a harmadik paraméter a HTTP fejrész opciók hashe. Mivel az átirányítás az előző oldalra történik a vizsgálat akcióban, a harmadik paraméterben kell megadnunk a `HTTP_REFERER` fejrész opcióval, hogy mely oldal volt az előző oldal. A feltételezésünk az, hogy az átirányítás sikeres, valamint a sessionünk inicializálódik, amit a session hash vizsgálatával tehetünk meg.

A harmadik tesztlépés egy új tesztet indít, annak a nézetére navigál. Sikeres HTTP válasz az elvárásunk, valamint az, hogy az adatbázisban a tesztek száma eggyel növekedjék, és az oldalon legyen pontosan egy `#word_word` azonosítójú beviteli mező.

A negyedik tesztlépésben a kontrollertől elkérjük az újonnan létrehozott teszt objektumát, és a kitalálandó szó objektumát. Az adatbázisból lekérdezzük a válaszok aktuális számát. Ezután egy megfejtést adunk a szóra, az URL-ben azonosítóval hivatkozva a tesztre és a szóra. A feltételezésünk az, hogy a válaszok száma tömb mérete a teszt után eggyel nagyobb.

A tesztet úgy folytathatnánk, hogy a negyedik tesztlépést egy ciklusba szervezzük, ami annyiszor ismétlődik, ahány szó van a tesztben, majd ellenőrizzük a teszt végeredményét.

```
class DoTestTest < ActionDispatch::IntegrationTest
  # test "the truth" do
  #   assert true
  # end
  test "take a test" do
    get url_for(controller: 'say', action: 'hello')
    assert_response :success
    assert_select "div#header", 1
    post_via_redirect '/sessions/create',
```

```
      { username: users(:me).username,
        password: 'titok'
      },
      {
        'HTTP_REFERER' => '/say/hello'
      }
    assert_equal session[:user], users(:me).id
    n = Test.count
    get test_path
    assert_response :success
    assert_equal Test.count, n+1
    assert_select "#word_word", 1
    test = assigns[:test]
    word = assigns[:word]
    number_of_answers = JSON.parse(test.solution).size
    post url_for(controller: 'words', action: 'solve', tid: test.
       id, sid: word.id),
      { word: { word: 'a' } },
      { user: users(:me).username }
    assert_equal number_of_answers+1, JSON.parse(assigns[:test].
       solution).size
  end
end
```

Futtatva az integrációs tesztet láthatjuk, hogy az egy teszteset négy tesztlépése során mind az öt ellenőrzésen sikeresen átment.

```
kovacsg@debian:~/gyakorlat/test/integration$ RAILS_ENV='test'
    rake test:integration
(in /home/kovacsg/gyakorlat)
Run options: --seed 57162

# Running:

.

Finished in 0.738854s, 1.3534 runs/s, 6.7672 assertions/s.

1 runs, 5 assertions, 0 failures, 0 errors, 0 skips
```

A tesztek ötödik nagy csoportja a portál teljesítőképességét hivatott ellenőrizni. Rails 4-től ez már nem része a teszt-keretrendszernek, külön telepítenünk kell a megvalósító függvénykönyvtárakat. A teljesítményteszek futtatásához szükségünk van a `ruby-perftest` és a `ruby-prof` függvénykönyvtárra, melyeket a `Gemfile`-ban kell meghivatkoznunk, illetve egy olyan `ruby` értelmezőre, amely képes monitorozni a végrehajtást során felhasznált memóriát és a végrehajtáshoz szükséges időt. Az alapértelmezett értelmező csak korlátosan képes kiszolgálni a teljesítménytesztek.

A teljesítménytesztteknek két fajtája van a hosszú végrehajtási időt igénylő metódusok tesztje (`profile`) és a statisztika (`benchmark`). A két teszttípus ugyanazokat a teszteseteket hajtja végre, és hasonló paraméterekkel konfigurálhatók. A benchmark a tesztesetek többszöri futtatása alapján általános statisztikát közöl a portál a teszteset által ellenőrzött részéről, a profile pedig a szűk keresztmetszetet jelentő pontokat próbálja azonosítani a sorrol sorra mért végrehajtási költség meghatározásával. Az előbbi egy fekete dobozos teszt annak eldöntésére, hogy van-e teljesítmény szempontjából probléma egy nézeten, az utóbbi pedig azt mondja meg, hogy hol.

A teljesítménytesztteket akárcsak az integrációs teszeket explicit módon kell létrehoznunk:

```
kovacsg@debian:~/gyakorlat/test$ rails g performance_test load
      create  test/performance/load_test.rb
```

Az alábbi teszteset az index nézet támadja meg 5 egymás utáni kéréssel.

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

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

  test "homepage" do
    get '/'
  end
end
```

A tesztesek végrehajtásának naplója a `logs` könyvtárba kerül, a teljesítménytesztek részletes adatai emellett a `tmp/performance` könyvtárban is megjelennek szövegesen, táblázatosan esetleg képként grafikusan.

A gyakran végrehajtandó teljesítményteszt-célok vonatkozásában teszteseteket érdemes definiálnunk, amelyeket a `test:profile` és a `test:benchmark` `rake` célokkal hajthatunk végre.

```
kovacsg@debian:~/gyakorlat$ perftest benchmarker
Run options: --seed 45523

# Running:



Finished in 0.003734s, 0.0000 runs/s, 0.0000 assertions/s.
```

```
0 runs, 0 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat$ ^C
```

A szűk keresztmetszet általában egy számításigényes függvény vagy egy komplex adatbázislekérdezés. Egy-egy függvény vizsgálatára nincs szükség az összes teszteset lefuttatására, azt a `perftest` paranccsal is megtehetjük, ami során a `profiler` és a `benchmarker` opciók használhatók.

```
kovacsg@debian:~/gyakorlat$ perftest benchmarker User.all
Run options: --seed 38525

# Running:

BenchmarkerTest#test_user_all (15 ms warmup)
            wall_time: 0 ms
               memory: unsupported
              objects: unsupported
              gc_runs: 0
              gc_time: 0 ms
.

Finished in 1.118938s, 0.8937 runs/s, 0.0000 assertions/s.

1 runs, 0 assertions, 0 failures, 0 errors, 0 skips
```