Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2014. november 26.

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 teszt adatokat 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`. Két játékost definiáljunk, akik között így játékot hozhatunk létre.

```
<% SALT = 'titkositosegedvaltozoerteke' %>
<% PASS = 'titok' %>
```

```
jatekos1:
  id: 1
  username: en
  salt: <%= SALT %>
  encrypted_password: <%= User.encrypt PASS, SALT %>
  email: en@mail.bme.hu

jatekos2:
  id: 13
  username: vesztes
  salt: <%= SALT %>
  encrypted_password: <%= User.encrypt PASS, SALT %>
  email: vesztes@mail.bme.hu
```

A felhasználókhoz hasonló módon megadjuk két játék tesztadatait is az `games.yml` fájlban. A játékok és a felhasználók között kettő-több reláció van, az idegen kulcsaink neve `player1` és `player2`, amik nem felelnek meg a Rails konvencióinak, mert nem `_id`-re végződnek, így elsődleges kulcsaikkal hivatkozunk a felhasználókra.

```
elso:
  player1: 13
  player2: 1
  result: 2

masodik:
  player1: 1
  player2: 13
  result: 1
```

A lépések tesztadatait (`moves.yml`) is vegyük fel. A lépések egy-több relációban áll a játékok modellel, és a kulcsunk itt megfelel a Rail konvencióinak. Ez lehetővé teszi számunkra a másik tesztadat kulcsával való hivatkozást. Ha a YAML fájlunkban egy ilyen kulcshoz a hivatkozott tesztadat egy kulcsát rendeljük, a Rails feloldja a kapcsolatot.

```
one:
  game: elso
  move: e5
  user_id: 1

two:
  game: elso
```

```
  move: e6
  user_id: 13
```

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,
    length: {in: 4..14, too_long: 'Too long username'},
    uniqueness: true
```

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. A másik tesztesetünk azt ellenőrzi, hogy a tesztadatok között megadott felhasználónév, jelszó páros segítségével be tudhatunk-e egyáltalán jelentkezni. A harmadik a felhasználónév hosszát.

Írjunk még egy tesztesetet, ami a modell egy példánymetódusát ellenőrzi. Ez az általunk írt `games` metódust teszteli, amit azon játékokból rakunk össze, ahol a játékosunk az első, illetve a második pozícióban áll. Az ellenőrzést a számosság alapján vizsgáljuk.

```
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  test "the truth" do
  #def test_the_truth
      assert true, "Valami nagy baj van, ha ez nem true"
  end
```

```
  test "cannot save without username" do
    u = User.new
    assert !u.save, "Houston"
  end

  test "cannot save with existing username" do
    u = User.new
    u.username = 'en'
    assert !u.save, " we have a problem"
  end

  test "cannot save user with username of 2 characters"
      do
    u = User.new
    u.username = "jo"
    assert !u.save
  end

  test "count games of en" do
    @user = users(:jatekos1)
    assert_equal @user.games.size, Game.all.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.

```
RAILS_ENV='test' rake test:units
(in /home/kovacsg/gyakorlat)
Run options: --seed 63727

# Running:

.....

Finished in 0.257367s, 19.4275 runs/s, 19.4275
   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. 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 "the truth" do
  #   assert true
  # end
  setup do
    @request.env["HTTP_REFERER"]='/say/hello'
  end

  test "login" do
    post :create, { :username => users(:jatekos1).
        username, :password => 'titok' }
    assert_response :redirect
    assert_not_nil session[:user]
  end

  test "logout" do
```

```
    get :destroy, nil, { user: users(:jatekos1).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
    @request.env["HTTP_REFERER"]='/say/hello'
  end
end
```

A `UsersControllerTest`-et és a `TasksControllerTest`-et szkripttel hoztuk létre, amely automatikusan generálta a teszteseteket és tesztadatokat, az utóbbi esetében még a tesztesetek törzse is létrejött. A tesztadatok kulcsait módosítottuk, így az azokra való hivatkozást frissítenünk kell a `setup`-ban. Mivel a műveleteink jó része csakis bejelentkezett felhasználó számára lehetséges, e két tesztfájlban a `get` és `post` műveletek harmadik paraméterében inicializálunk kell a `sessions` hasht a harmadik paraméterben. Az alábbi kódrészlet erre mutat egy példát, azonban ezt a kontrollerekben szereplő `before_filter` miatt minden akcióban el kell végeznünk, ráadásul ez a művelet csak az adminisztrátor típusú felhasználók számára lehet elérhető, így egyszerre mindkét tesztadatunkat kihasználjuk.

A teszteseteink legyenek a következők. A regisztrációs oldal betöltésének sikerességét és az azon megjelenő H1 HTML elem címkéjét vizsgáljuk. A profil oldalon a betöltés sikerességét ellenőrizzük, ha a felhasználó be van jelentkezve, és a saját oldalát nézi. Az index oldalon a tördelést ellenőrizzük, mivel 2 tesztadatunk van, a második oldalon nem jelenhet meg LI HTML elem.

```
require 'test_helper'

class UsersControllerTest < ActionController::TestCase
  test "should_get_new" do
    get :new
```

```
    assert_response :success
    assert_select "h1", "Registration"
  end

  test "should get edit" do
    get :edit, { id: users(:jatekos1).id}, { user:
       users(:jatekos1).id}
    assert_response :success
  end

  test "should get index page" do
    get :index, { page: 2 }
    assert_response :success
    assert_select "li", 0
  end

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

  test "should get forgotten" do
    get :forgotten
    assert_response :success
  end
end
```

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

A tesztek harmadik típusa az integrációs teszt, amellyel egy böngészési folyamatot ellenőrzünk. Készítsünk egy tesztet a felhasználói profilkép feltöltésének esetére!

```
rails generate integration_test new_game
```

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

A fájl elején töltsük be az összes teszt adatot `fixtures :all`, majd definiáljuk a böngészési folyamatot úgy, hogy az a bejelentkezéstől jusson el legalább egy játék létrehozásáig. Ne feledkezzünk el a karakterkódolás beállításáról sem, ha ékezetes karaktert használunk!

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 csak annyit feltételezünk, hogy az oldal betöltődik.

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 a sessionünk inicializálódik, amit a session hash vizsgálatával tehetünk meg.

A harmadik tesztlépés megnézi az index oldalt tölti be, ahol kihívhatunk egy játékost. Sikeres HTTP válasz az elvárásunk.

A negyedik tesztlépésben létrehozzuk a játékot, a `/games/` URL mellé átadva a `:cid` paramétert.

```
require 'test_helper'

class NewGameTest < ActionDispatch::IntegrationTest
  # test "the truth" do
  #   assert true
  # end
  test "new_game" do
    get '/say/hello'
    assert_response :success

    @user = users(:jatekos1)
    post_via_redirect '/sessions/create',
      {:username=>@user.username, :password=>'titok'},
      { "HTTP_REFERER" => '/say/hello' }
    assert_equal session[:user], @user.id

    get '/users/index'
    assert_response :success
    assert_select "li", User.all.size
```

```
    g = Game.all.size
    get '/games', { :cid => users(:jatekos2)}
    assert_response :success
    assert_equal g, Game.all.size

  end
end
```

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

```
RAILS_ENV='test' rake test:integration(in /home/kovacsg
   /gyakorlat)
Run options: --seed 7457

# Running:

.

Finished in 0.745469s, 1.3414 runs/s, 8.0486 assertions
   /s.

1 runs, 6 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 teljesítménytesztet.

A teljesítménytesztnek 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 általános statisztikát közöl a portál a teszteset által ellenőrzött részéről, a profile pedig a szűk keresztmetszetet jelentő pontokat próbálja azonosítani.

A teljesítménytesztet akárcsak az integrációs teszeket explicit módon kell létrehoznunk:

```
rails generate performance_test browsing
```

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

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

class BrowsingTest < 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.

```
rake test:profile
(in /home/kovacsg/gyakorlat)
Run options: --seed 40348

# Running tests:

BrowsingTest#test_homepage (140 ms warmup)
        process_time: 20 ms
              memory: 0 Bytes
             objects: 0


Finished tests in 4.390694s, 0.2278 tests/s, 0.0000
   assertions/s.

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

```
rake test:benchmark
(in /home/kovacsg/gyakorlat)
Run options: --seed 30560

# Running tests:

BrowsingTest#test_homepage (133 ms warmup)
            wall_time: 6 ms
               memory: unsupported
              objects: unsupported
              gc_runs: 0
              gc_time: 0 ms


Finished tests in 0.665908s, 1.5017 tests/s, 0.0000
   assertions/s.

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