Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2013. április 30.

A gyakorlaton az előző gyakorlatok során elkészített megoldást tesztelésével foglalkozunk.

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. Először definiáljunk egy közös jelszót az összes felhasználó számára, aminek kódolatlan értékét a `PASS` attribútumban tároljuk el. A titkosított jelszó megadához a modell osztály `encrypt` osztálymetódusát hívjuk segítségül. A felhasználóspecifikus a titkosítás során használt `salt` attribútumot külön-külön definiáljuk. A jelszó attribútumot időközben átneveztük, és felvettünk két új attribútumot is, amikkel ki kell egészítenünk az automatikusan generált mintát, ugyanis az utólag hozzáadott attribútumok, idegen kulcsok azonban nem jelennek meg.

```
<% MY_SALT = 'valami' %>
<% PASS = 'titok' %>
```

```
<% STUDENT_SALT = 'valamimas' %>
me:
  neptun: oweyoa
  encrypted_passwd: <%= User.encrypt PASS,MY_SALT%>
  salt: <%= MY_SALT %>
  email: kovacsg@tmit.bme.hu
  student: 1

student:
  neptun: tttttt
  encrypted_passwd: <%= User.encrypt PASS, STUDENT_SALT
      %>
  email: t@mail.bme.hu
  salt: <%= STUDENT_SALT %>
  student: 2
```

A felhasználókhoz hasonló módon megadjuk két feladat tesztadatait is a `tasks.yml` fájlban. A feladatok modellt `scaffold`-dal hoztuk létre így ott már láthatunk kezdeti adatokat, amiket módosítunk

```
elso:
  number: 1
  deadline: 2013-03-04
  url: https://twiki.db.bme.hu/twiki/bin/view/Student/
    Ruby/RubyElsoFeladat20131

masodik:
  number: 2
  deadline: 2013-03-18
  url: https://twiki.db.bme.hu/twiki/bin/view/Student/
    Ruby/RubyMasodikFeladat20131
```

A megoldások tesztadatait (`clazzs.yml`) egyelőre hagyjuk üresen.

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, 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/unit` könyvtárában találjuk. Minden egyes modell létrehozása után automatikusan létrejön hozzá egy ahhoz kapcsolódó teszt osztály itt.

Írjunk teszteseteket, amelyek a `User` modellünk

```
validates :neptun, :uniqueness => true, :presence =>
  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 tesztesetben megpróbálunk egy a tesztadatok betöltésével létrehozott felhasználó Neptun-kódjával azonos Neptun-kóddal elmenteni egy új felhasználó objektumot.

```
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  test "cannot save user without neptun" do
    u = User.new
    assert !u.save
  end

  test "cannot save user with existing neptun" do
    u = User.new
    u.neptun=users(:student)
    assert !u.save
  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.

```
RAILS_ENV=test rake test:units
(in /home/kovacsg/gyakorlat)
Loaded suite /var/lib/gems/1.9.1/gems/rake-10.0.3/lib/
  rake/rake_test_loader
Started
...
Finished in 0.313193 seconds.
```

```
3 tests, 3 assertions, 0 failures, 0 errors, 0 skips

Test run options: --seed 25740
```

A funkcionális tesztek a kontrollerek és a nézetek helyes működését ellenőrzik. Alapértelmezés szerint minden egyes a létrehozáskor megadott kontroller akcióra létrejön egy az akció sikeres megjelenítését ellenőrző teszteset.

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.

```
class SessionControllerTest < ActionController::
    TestCase
  test "login" do
    port :create, { :neptun => users(:me).neptun, :
        password=> 'titok' }
    assert_response :redirect
    assert_equal session[:user], users(:me).id
  end

  test "logout" do
    get :destroy, nil, {:user=>users(:me).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 SessionControllerTest < ActionController::
    TestCase
  setup do
    @request.env["HTTP_REFERER"]='/say/hello'
  end
end
```

Az `TasksControllerTest`-et scaffolddal hoztuk létre, amely automatikusan generálta a teszteseteket és tesztadatokat. A tesztadatokat módosítottuk, így az azokra való hivatkozást frissítenünk kell a `setup`-ban `:elso`-re. A feladatbeadás fájlfelöltés műveletének tesztelésére töltsünk fel egy valóságos állományt a `fixture_upload_file` metódussal egy új tesztesetben. Ez a metódus a tesztadatok könyvtárához képesti relatív útvonalat vár első argumentumként, a második, opcionális argumentuma az ott található állomány MIME típusa. Hozzunk létre egy tetszőleges szöveget tartalmazó fájlt a `test/fixturesfiles` könyvtárban `test_solution.txt` néven. A tesztesetünkben két feltételezéssel élünk, egyrészt azzal, hogy a következő nézetünk az aktuális feladat nézete, másrészt pedig jó lenne, ha a fájl valóban megjelenne a fájlrendszeren. Mivel feladatot csak bejelentkezett felhasználó tud beadni, ezért itt is meg kell adnunk a sessionre vonatkozó paramétert.

```
class TasksControllerTest < ActionController::TestCase
  def setup
    @task = tasks(:elso)
  end

  test "submit_a_solution" do
    post :upload,
      {
        id: @task.id,
        uid: users(:student).id,
        upload: {
          original_filename:'test_solution.txt',
          file:fixture_file_upload('files/test_solution
              .txt')
        }
      },
```

```
      {user: users(:student).id}
    assert_redirected_to task_path(assigns(:task))
    assert File.exists?('public/data/test_solution.txt'
      )
  end
end
```

A `UsersControllerTest` osztályunk az `edit` és `forgotten` akciókat ellenőrző teszteseteiben a `get` második paramétereként be kell állítanunk az `id` paramétert a tesztadatok alapján vagy a harmadik paraméterrel azt szimuláljuk, hogy a felhasználó már bejelentkezett. Minden mást az egyszerűség kedvéért változatlanul hagyunk.

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

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

  test "should get forgotten" do
    get :forgotten, nil, { :user => users(:me).id }
    assert_response :success
  end
end
```

A funkcionális teszteseteket futtatva láthatjuk, hogy sikeresen lefutnak.

```
> rake test:functionals
(in /home/kovacsg/gyakorlat)
Loaded suite /var/lib/gems/1.9.1/gems/rake-10.0.3/lib/
   rake/rake_test_loader
Started
..............
Finished in 0.495508 seconds.

14 tests, 20 assertions, 0 failures, 0 errors, 0 skips

Test run options: --seed 31199
```

A tesztek harmadik típusa az integrációs teszt, amellyel egy böngészési folyamatot ellenőrzünk. Készítsünk egy tesztet egy megoldás beküldésének esetére!

```
rails generate integration_test submit_solution
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `submit_solution_test.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 a feladat beküldéséig. 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 `neptun` 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.

A harmadik tesztlépés letölti az egyik tesztadathoz tartozó feladatot leíró oldalt, ahol majd lehetséges lesz a feladat beadása. Itt is csak azt feltételezzük, hogy az oldal sikeresen megjelenik.

A negyedik tesztlépésben előhalásszuk az előző tesztfájlunkat. Persze mindenek előtt eltakarítjuk a funkcionális tesztek nyomait a `public/data` könyvtárból. Majd lemásoljuk a funkcionális tesztnél a feladatbeadás ellenőrzése végett írt tesztünket. Az ellenőrzést annyival bővítjük ki, hogy előkeressük a beadott feladatot a `Submission` modell osztály példányai közül.

```
require 'test_helper'
```

```
class SubmitSolutionTest < ActionDispatch::
    IntegrationTest
  fixtures :all
  test "submit a new solution" do
    get url_for(controller:'say',action:'hello')
    assert_response :success
    post_via_redirect '/sessions/create',
      { neptun:users(:student).neptun, password:'titok'
        },
      { 'HTTP_REFERER'=>'/say/hello' }
    assert_equal session[:user], users(:student).id
    get task_url(tasks(:elso))
    assert_response :success
    upload_file=fixture_file_upload('test/fixtures/
       files/test_solution.txt','text/plain')
    post url_for(controller:'tasks',action:'upload'), {
      id: tasks(:elso).id,
      uid: users(:student).id,
      upload: {
         original_filename:'test_solution.txt',
         file:upload_file
      }
    }
    s = Submission.find_by_task_id_and_user_id tasks(:
       elso).id, users(:student).id
    assert_not_nil s.filename
    assert File.exists?('public/data/test_solution.txt'
       )
  end
end
```

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

```
rake test:integration
(in /home/kovacsg/gyakorlat)
Loaded suite /var/lib/gems/1.9.1/gems/rake-10.0.3/lib/
   rake/rake_test_loader
Started
.
Finished in 0.302538 seconds.
```

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

Test run options: --seed 25382
```

A tesztek ötödik nagy csoportja a portál teljesítőképességét hivatott ellenőrizni. A teljesítménytesztek akárcsak az integrációs teszeket explicit módon kell létrehoznunk `rails generate performace_test`.

A teljesítményteszek futtatásához szükségünk van a `ruby-prof` függvénykönyvtárra, melyet 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 teljesítménytesztek 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.

Egy teszt a projekt létrehozásakor automatikusan generálódik. Az egyetlen tesztesete a portálunk főoldalának kiszolgálását. A vizsgálatunkhoz használjuk azt fel. Állítsuk be az opciókat, hogy a számunkra érdekes adatok jelenjenek meg a konzolon.

```
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]
    }

  def test_homepage
    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 teszte-

seteket érdemes definiálnunk, amelyeket a `test:profile` és a `test:benchmark` `rake` célokkal hajthatunk végre.

```
rake test:profile
(in /home/kovacsg/gyakorlat)
Loaded suite /var/lib/gems/1.9.1/gems/rake-10.0.3/lib/
   rake/rake_test_loader
Started
BrowsingTest#test_homepage (116 ms warmup)
        process_time: 24 ms
              memory: 0 Bytes
             objects: 0

Finished in 32.788773 seconds.

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

Test run options: --seed 38842
```

```
rake test:benchmark
(in /home/kovacsg/gyakorlat)
Loaded suite /var/lib/gems/1.9.1/gems/rake-10.0.3/lib/
   rake/rake_test_loader
Started
BrowsingTest#test_homepage (127 ms warmup)
           wall_time: 8 ms
              memory: unsupported
             objects: unsupported
             gc_runs: 0
             gc_time: 0 ms

Finished in 0.854406 seconds.

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

Test run options: --seed 52804
```

Egy-egy – jellemzően modell osztály – metódus teljesítményvizsgálatára nem feltétlenül érdemes teszteset írnunk, az elvégezhetjük a `rails` szkript `profiler` és `benchmarker` céljaival is. Az alábbi példák az összes felhasználó lekérdezése adatbázis műveletének teljesítményét vizsgálják meg.

```
rails profiler User.all
Loaded suite script/rails
Started
ProfilerTest#test_user_all (19 ms warmup)
        process_time: 3 ms
              memory: 0 Bytes
             objects: 0

Finished in 1.761155 seconds.

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

Test run options: --seed 33691
```

```
rails profiler User.all --metrics wall_time,user_time,
   cpu_time,memory,objects
Loaded suite script/rails
Started
ProfilerTest#test_user_all (21 ms warmup)
           wall_time: 3 ms
            cpu_time: 1 ms
              memory: 0 Bytes
             objects: 0

Finished in 2.482132 seconds.

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

Test run options: --seed 1749
```

```
rails benchmarker User.all
Loaded suite script/rails
Started
BenchmarkerTest#test_user_all (19 ms warmup)
           wall_time: 1 ms
              memory: unsupported
             objects: unsupported
             gc_runs: 0
             gc_time: 0 ms

Finished in 0.621627 seconds.
```

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

Test run options: --seed 42334
```