Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2011. november 21.

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk.

A nézeteken a megjelenítendő tarntárgyi adatokat a bejelentkezett felhasználó szerepe szerint kell kialakítanunk. A be nem jelentkezett felhasználó csak egy általános listát láthat, az oktató szerepkörrel rendelkező felhasználó a saját tantárgyaira vonatkozó információkhoz és műveletekhez férhet hozzá, míg a hallgató szerepkörrel rendelkező felhasználó az általa látogatott tárgyak adatait láthatja. Ez utóbbi vonatkozásában még nincs relációnk a modell osztályaink között. A hallgató tantárgyainak listájához a hallgató által látogatott tanórák (`:through => :classes`) tantárgyain (`:source => :subject`) keresztül férünk hozzá. Mivel egy kurzusnak több tanórája van, ezért az egyes kurzusok többször is szerepelnek a visszadott enumerációban, ezért alkalmazzuk még a `uniq => true` szűrőfeltételt.

```
class User < ActiveRecord::Base
  has_many :attended_subjects, :through => :classes, :
      source => :subject, :uniq => true
end
```

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 definiáljuk a titkosításhoz használt `SALT` attribútum értékét, amelyre minden egyes felhasználó példányában hivatkozni fogunk. A jelszó megadához a modell osztály `encrypt` metódusát hívjuk segítségül.

```
<% SALT = 'hello' %>
<% PASS = 'titok' %>
me:
  neptun: oweyoa
  encrypted_password: <%= User.encrypt PASS, SALT %>
  salt: <%= SALT %>
  email: kovacsg@tmit.bme.hu
  role: 1
student:
  neptun: aaaaaa
  encrypted_password: <%= User.encrypt 'a', 'hello' %>
  salt: hello
  email: a@bme.hu
  role: 2
```

A felhasználókhoz hasonló módon megadunk egy tantárgyat is a `subjects.yml` fájlban. A tantárgyat `scaffold`-dal hoztuk létre így ott már láthatunk kezdeti adatokat, az utólag hozzáadott attribútumok, idegen kulcsok azonban nem jelennek meg. Adjuk hozzá a felhasználók táblára vonatkozó idegen kulcsot, amelynek értékére az előbb definiált rekord kulcsával, `me`-vel hivatkozzunk! A Rails a `Subject` modell osztály asszociációi nyomán tudni fogja, hogy a `lecturer` attribútum a `User` modell osztály példányára hivatkozik, így az előzőekben definiált tesztadatok (`users.yml`) között keres egy olyan rekordot, aminek a kulcsa `me` volt. Ennek hatására a `user_id` attribútum automatikusan állítódik az adatbázisban. Alternatív megoldás lehet a `user_id` attribútum beállítása a megfelelő felhasználó elsődleges kulcsának értékére. A tesztadatfájlokban, bár általában nem jelenik meg, az `id` attribútum is állítható. Ha nem tesszük meg, akkor a tesztadat `id` attribútuma egy véletlen, még nem létező értéket vesz fel.

```
one:
  name: RoR
  code: BMEVITMBV17
```

```
  sheet: /home/kovacsg/gyakorlat/public/data/
    BMEVITMBV17.txt
  lecturer: me
```

A tanórából tesztadataként (`clazzs.yml`) a két rekordot definiálunk. Az idegen kulccsal (`subject: ror`) hivatkozunk az imént definiált tantárgyra, megadjuk a tanóra időpontját és helyiségét. Az időpontot a már ismert `ActiveSupport` helper metódusokkal állítjuk be, majd az adatbázisformátumnak megfelelő stringgé konvertáljuk

```
one:
  date: <%= 12.weeks.ago.to_s(:db)
  room: I.B.138
  lecturer: me
two:
  date: <%= 11.weeks.ago.to_s(:db)
  room: I.B.138
  lecturer: me
```

Töltsük be a teszt adatbázisba 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
```

Először egy egységtesztet írunk. 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_presence_of :username
validates_uniqueness_of :username
```

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 felhasználónevével azonos felhasználónévvel 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, "Houston, we have a problem"
  end

  test "cannot save user with existing neptun" do
    u = User.new
    u.neptun = users(:student).neptun
    assert !u.save, "Houston, we have a problem"
  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.

```
> rake test:units
(in /home/kovacsg/gyakorlat)
Loaded suite /var/lib/gems/1.9.1/gems/rake-0.9.2.2/lib/
    rake/rake_test_loader
Started
...
Finished in 0.277513 seconds.

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

Test run options: --seed 44178
```

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 nem tartalmaz teszteseteket. A bejelentkezés eseményt a `SessionController` `create` metódusát használja, a kijelentezés esemény pedig a `destroy` metódust. Két tesztesetet készítünk, az egyik a sikeres bejelentkezés esetét vizsgálja, a másik a kijelentkezést. 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, { :username => users(:me).neptun, :
        password=> users(:me).passwd }
    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 `HTTP_REFERER` 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"]='/subjects'
  end
end
```

Az `SubjectsControllerTest`-et scaffolddal hoztuk létre, amely automatikusan generálta a teszteseteket és tesztadatokat. A tesztadatokat módosítottuk, így az azokra való hivatkozást frissítenünk kell a `setup`-ban `:ror`-ra. A fájlfelöltés műveletet használó tesztesetek hibásak, a tesztadatok között megadott string paraméter helyett töltsünk fel egy valóságos állományt a `fixture_upload_file` metódussal a tárgyat létrehozó és módosító akciókra

írt tesztesetekben. 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 `ror.txt` néven.

```
class SubjectsControllerTest < ActionController::
    TestCase
  def setup
    @subject = subjects(:ror)
  end

  def teardown
    @subject = nil
  end

  test "should update subject" do
    put :update, id:@subject, subject: { code: @subject
        .code, name:
@subject.name, sheet: fixture_file_upload('files/ror.
    txt') }
    assert_redirected_to subject_path(assigns(:subject)
        )
  end
end
```

A `UsersControllerTest` osztályunk tesztesetei jelenleg elbuknak, az `edit` és `forgotten` akciók 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-0.9.2.2/lib/
    rake/rake_test_loader
Started
.............
Finished in 0.614388 seconds.

13 tests, 19 assertions, 0 failures, 0 errors, 0 skips

Test run options: --seed 6488
```

A tesztek harmadik típusa az integrációs teszt, amellyel egy böngészési folyamatot ellenőrzünk. Készítsünk egy tesztet egy új tantárgy létrehozásának esetére!

```
rails generate integration_test new_subject
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `new_subject_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 az új tantárgy adatlapjának feltöltéséig. 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 megtekinti a tantárgyak listája nézetet. A második tesztlépésben a felhasználó bejelentkezik, a harmadik tesztlépésben létrehozza az új tantárgyat.

A nézetek tesztelése végett végezzünk el egy apró módosítást a `subjects` nézet template-ek között található `show.html.erb` állományban, a paragrafusokat (`<p>` HTML elemek) címkézzük meg a bennük található attribútum nevével `id`-ként.

Az első tesztlépésben meglátogatjuk az új tantárgyat létrehozó oldalt. Ez pontosan megegyezik az egyik funkcionális teszttel, emeljük onnan át.

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 bejelentkezés 200-as státuszkóddal generál választ, és, hogy a sessionünk inicializálódik.

A harmadik tesztlépés egy új rekordot szúr be a tantárgyak táblájába, ezért e tesztlépés végrehajtása előtt megvizsgáljuk, hogy hány rekord található benne. A tesztadatok között egy tantárgyat definiáltunk csak ezért feltételezésünk az, hogy egy lesz a visszaadott érték, amit eltárolunk egy lokális változóban. A feltöltendő tantrágy adatlapjaként használjuk az imént létrehozott és a tesztadatok között elhelyezett szöveges állományt (`files/ror.txt`), MIME-típusként adjuk meg, hogy `text/plain`. Az új tantárgyat létrehozó oldalon található formunk egy tantárgy típusú hash-t vár paraméterként. Az akció célja ismét átirányítást hajt végre, ezért ismét a `post_via_redirect` metódust használjuk. A feltételezésünk az, hogy a `create` akció végrehatjása után kettő rekordunk lesz az adatbázisban, a fájl létrejön a tantárgy adatlapokat tároló könyvtárban (`public/data`) a tantárgy kódjának megfelelő néven, és az, hogy a nézet pontosan egy olyan HTML elemet fog tartalmazni, amire illeszkedik a `p#sheet` CSS-selector. Az integrációs teszt lefutása után a teszt során a tesztadatokon végzett módosítások elvesznek, így a beszúrt rekordunk nem lesz hozzáférhető későbbi tesztesetek számára, ugyanakkor a feltöltött fájl megmarad.

```
require 'test_helper'
class NewSubjectTest < ActionDispatch::IntegrationTest
  fixtures :all
  test "new subject" do
    get url_for(:controller=>subjects, :action=>:new)
    assert_response :success
    post_via_redirect "/sessions/create",
      {:username=>users(:me).username, :password=>'
```

```
          titok'},
        { "HTTP_REFERER"=>'/subjects/new' }
    assert_equal session[:user], users(:me).id
    assert_response :success
    s = Subject.all.size
    f = fixture_file_upload('text/fixtures/files/ror.
       txt', 'text/plain')
    post_via_redirect "/subjects/create",
      { :subject => { name: 'Matek', code: "BME...",
         sheet: f }}
    assert_equal s+1, Subject.all.size
    assert File.exists? File.join(Rails.root.join("
       public","data"), "BME....txt")
    assert_select "p#sheet", 1
  end
end
```

Futtatva az integrációs tesztet láthatjuk, hogy az egy teszt eset hat teszt lépése során mind a tizenegy ellenőrzésen sikeresen átment.

```
rake test:integration
(in /home/kovacsg/gyakorlat)
Loaded suite /var/lib/gems/1.9.1/gems/rake-0.9.2.2/lib/
   rake/rake_test_loader
Started
.
Finished in 0.443689 seconds.

1 tests, 6 assertions, 0 failures, 0 errors, 0 skips

Test run options: --seed 64392
```

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 teszteseteket 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-0.9.2.2/lib/
    rake/rake_test_loader
Started
ATest#test_homepage (19 ms warmup)
            wall_time: 1 ms
               memory: 0 Bytes
BrowsingTest#test_homepage (1 ms warmup)
            wall_time: 2 ms
               memory: 0 Bytes
              objects: 0
```

```
Finished in 1.197252 seconds.

2 tests, 0 assertions, 0 failures, 0 errors, 0 skips

Test run options: --seed 23835
```

```
rake test:benchmark
(in /home/kovacsg/gyakorlat)
Loaded suite /var/lib/gems/1.9.1/gems/rake-0.9.2.2/lib/
   rake/rake_test_loader
Started
ATest#test_homepage (16 ms warmup)
           wall_time: 1 ms
              memory: unsupported
BrowsingTest#test_homepage (1 ms warmup)
           wall_time: 1 ms
              memory: unsupported
             objects: unsupported
             gc_runs: 0
             gc_time: 0 ms

Finished in 0.886789 seconds.

2 tests, 0 assertions, 0 failures, 0 errors, 0 skips

Test run options: --seed 24395
```

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 (32 ms warmup)
        process_time: 3 ms
              memory: unsupported
             objects: unsupported

Finished in 1.442566 seconds.
```

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

Test run options: --seed 40749
```

```
rails profiler User.all --metrics wall_time,user_time,
   cpu_time,memory,objects
Loaded suite script/rails
Started
ProfilerTest#test_user_all (38 ms warmup)
           wall_time: 2 ms
            cpu_time: 1 ms
              memory: unsupported
             objects: unsupported

Finished in 1.766661 seconds.

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

Test run options: --seed 3887
```

```
rails benchmarker User.all
Loaded suite script/rails
Started
BenchmarkerTest#test_user_all (33 ms warmup)
           wall_time: 1 ms
              memory: unsupported
             objects: unsupported
             gc_runs: 0
             gc_time: 0 ms

Finished in 1.133827 seconds.

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

Test run options: --seed 60888
```