Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2011. árpilis 25.

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk.

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 = 'ezzeltitkositunk' %>
<% PASS = 'titok' %>
me:
  username: kovacsg
  email: kovacsg@tmit.bme.hu
  salt: <%= SALT %>
  encrypted_password: <%= User.encrypt PASS, SALT %>
```

A felhasználókhoz hasonló módon megadunk egy teendőt is a `issues.yml` fájlban. A teendőt `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!

```
one:
  label: teendo
  description: Hosszu, hosszu leiras
  priority: 5
  deadline: <%= -5.days.ago %>
  status: 1
  user: me
```

A csatolmányok tesztadataként (`solutions.yml`) a következő rekordot használjuk. Az idegen kulcsoknál a megfelelő tesztadatfájl egy kulcsát használjuk, ami jelen esetben `one` a `issues.yml`-t tekintve, és `me` a `users.yml`-re nézve. Ennek hatására a `source_id` és a `user_id` attribútum automatikusan állítódik. A tesztadatfájlokban, bár általában nem jelenik meg, az `id` attribútum is állítható. Használjuk ezt ki, és hivatkozzunk az eddig feltöltött egyetlen állományunkra.

```
one:
  file: public/data/1/1
  issue: one
  id: 1
  user: me
```

A teendőkhöz fűzött kommentek tekintetében (`comments.yml`) egyetlen tesztrekordot definiálunk, amely a `one` kulccsal azonosított idézetre vonatkozik, amit a `me` kulccsal azonosított felhasználó hozott létre.

```
one:
  comment: MyText
  issue_id: one
  user_id: 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_uniquness_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 "can not save user without username" do
    u = User.new
    assert !u.save, "Houston, we have a problem"
  end

  test "can not save user with existing username" do
    u = User.new(:username=>users(:me).username,
      :password=>'haho',
      :password_confirmation=>'haho')
    assert !u.save, "Houston, we have a problem"
  end
end
```

A teszt esetet 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 0.265292 seconds.

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

Test run options: --seed 42021
```

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 `SessionControllerTest` tesztet, amely jelenleg nem tartalmaz teszteseteket. A bejelentkezés eseményt a `SessionController` `create` metódusát használja, a kijelentezés esemény pedig a `destroy` metódust. A bejelentkezés vizsgálatára két tesztesetet készítünk, az egyik a sikeres esetet vizsgálja, a másik a sikertelent. A kijelentezés vizsgálatára egyetlen tesztesetet írunk. A tesztben a `post` metódust használjuk, amelynek első paramétere a teszelendő akció, a második a HTTP kérés paraméterei, a harmadok pedig a `session` paraméterek. A sikeres bejelentkezés tesztben a tesztadatokban szereplő felhasználónév, jelszó párost küldjük el, a sikertelenben hibás jelszót adunk meg. Sikeres esetben azt feltételezzük, hogy visszairányítódunk a belépés előtti nézetre, a `:user` session paraméter értéke nem `nil`, ráadásul megegyezik a tesztadat azonosítójával. Sikertelen esetben szintén átirányítást várunk, és azt, hogy a `:user` session paraméter `nil` értéket tartalmaz. 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.

```ruby
class SessionControllerTest < ActionController::
    TestCase
  test "login" do
    get :create
    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"]='/issues/new'
  end
```

Az `IssuesControllerTest`-et scaffolddal hoztuk létre, amely automatikusan generálta a teszteseteket és tesztadatokat. A tesztadatokat módosítottuk, így az azokra való hivatkozást frissítenünk kell a `setup`-ban `:one`-ra. A tördelés bevezetése is hibaforrás az automatikusan generált tesztesetek számára. A tördelést tartalmazó nézetek `get` metódusait paramétereznünk kell az aktuálisan megjelenítendő oldallal. Így az `index` metódus `get` metódushívása második paramétereként át kell adnunk a `page` paraméter értékét. A teljes sikerhez már csak a bejelentkezett felhasználót kell imitálnunk a `session[:user]` beállításával. Az `index` akció ellenőrzésekor ne elégedjünk meg a HTTP státus kód helyességének vizsgálatával, nézzük meg, hogy a `issues` példányváltozó hordoz-e értéket a kontrollerben, ezt az `assigns` hash segítségével tehetjük meg.

```
class IssuesControllerTest < ActionController::TestCase
  def setup
    @issue = issues(:one)
    @me = users(:me)
  end

  def teardown
    @issue = nil
  end

  test "should get index" do
    session[:user]=@me.id
    get :index, :page=>1
    assert_response :success
    assert_not_nil assigns(:issues)
  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. 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, :id => users(:me).id
    assert_response :success
  end

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

Idézetek tesztjeit a `QuotesControllerTest` osztályban definiáljuk. Új idézetet csak bejelenkezett felhasználó hozhat létre. Ezt a források kontrollere esetén bemutatott módon tesszük meg két tesztesettel. Sikertelen esetben átirányítást várunk. Sikeres esetben, amikor beállítjuk a `user` session paramétert a tesztadatok alapján, sikeres státuszkódot várunk, és azt, hogy az idézet kontroller példányváltozója rendelkezik értékkel.

A `comment` akció tesztesetét ki kell egészítenünk egy HTTP kérés paraméterrel, amelyet a tesztadatokból veszünk.

A teendők listázását végző `show` akció által megjelenített nézeten elvárásunk, hogy legyen egy tábla, mely a teendőhöz kapcsolt csatolmányokat listázza ki. A tábla létezését a nézet tesztelésével végezzük el, amelyet a következő `assert_select` illesztéssel teszünk meg. Azt feltételezzük, hogy a `table` CSS szelektor által kijelölt HTML elem pontosan egyszer fordul elő a kérés által előállított HTML dokumentumban. A tesztnek egy komplexebb változatát is elvégezhetjük azáltal, hogy a tábla második sorának értékét összevetjük a tesztadatokban megadott értékkel, vagyis a felhasználónévvel.

```
class IssuesControllerTest < ActionController::TestCase
  test "should show issue" do
    get :show, id: @issue
```

```
    assert_response :success
    assert_select 'table', 1
  end
end
```

A funkcionális tesztek harmadik nagy csoportja útvonalak tesztelését végzi el, ellenőrzi, hogy a `routes.rb` fájban definiált útvonal tábla helyes akcióhoz irányítja-e a beérkező HTTP kéréseket.

Egy saját útvonalat definiáltunk eddig, a csatolmányok feltöltését végző akciót képeztük le a szokottól eltérő útvonalra. Ellenőrizzük ennek helyességét a `assert_routing` metódussal.

```
class IssuesControllerTest < ActionController::TestCase
  test "downloads_mapped_to_issues_download" do
    assert_routing '/downloads/1/1', {:controller=>'
       issues',
      :action=>'download', :id=> '1', :aid=>'1'}
  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 0.669019 seconds.

14 tests, 21 assertions, 0 failures, 0 errors, 0 skips

Test run options: --seed 31838
```

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

```
rails generate integration_test new_issue
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `new_issue_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 teendő 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 megtekinti a teendők listája nézezet. A második tesztlépésben a felhasználó bejelentkezik, a harmadik tesztlépésben létrehozza az új teendőt.

Az első tesztlépésben meglátogatjuk az új teendőt létrehozó oldalt. Ez 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 tömbje. 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 teendők táblájába, ezért e tesztlépés végrehajtása előtt megvizsgáljuk, hogy hány rekord található benne. A tesztadatok között két teendőt definiáltunk ezért feltételezésünk, hogy kettő darab lesz, ezt az értéket eltároljuk egy lokális változóban. Az új teendőt létrehozó oldalon található formunk egy teendők 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 három rekordunk lesz az adatbázisban. Az integrációs teszt lefutása után a teszt során a tesztadatokon végzett módosítások elvesznek, így a beszúrt rekordunk nem lesz hozzáférhető későbbi tesztesetek számára.

```
require 'test_helper'
class NewIssueTest < ActionDispatch::IntegrationTest
  fixtures :all
  test "new_issue" do
    get url_for(:controller=>'issues', :action=>:new)
```

```
    assert_response :success
    post_via_redirect "/sessions/create", {:username=>
       users(:me).username,
      :password=>'titok'}, { "HTTP_REFERER"=>'/issues/
         new' }
    assert_response :success
    assert_equal session[:user], users(:me).id
    s = Issue.all.size
    post_via_redirect "/issues",
      { :issues=>{ :priority => 5, :label => "a", :
         description => "A"}}
    assert_equal s+1, Issue.all.size
  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 0.384403 seconds.

1 tests, 4 assertions, 0 failures, 0 errors, 0 skips

Test run options: --seed 47085
```

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
BrowsingTest#test_homepage (16 ms warmup)
            wall_time: 2 ms
               memory: unsupported

Finished in 0.209148 seconds.

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

Test run options: --seed 64274
```

```
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
BrowsingTest#test_homepage (17 ms warmup)
           wall_time: 1 ms
              memory: unsupported

Finished in 0.352583 seconds.

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

Test run options: --seed 23347
```

Egy-egy metódus teljesítményvizsgálatára nem feltétlenül érdemes tesztet 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 (16 ms warmup)
        process_time: 2 ms
              memory: unsupported
             objects: unsupported

Finished in 0.571877 seconds.

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

Test run options: --seed 55747

rails profiler User.all --metrics wall_time,user_time,
   cpu_time,memory,objects
Loaded suite script/rails
Started
ProfilerTest#test_user_all (15 ms warmup)
           wall_time: 2 ms
            cpu_time: 2 ms
```

```
               memory: unsupported
              objects: unsupported

Finished in 1.019760 seconds.

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

Test run options: --seed 64458

rails benchmarker User.all
Loaded suite script/rails
Started
BenchmarkerTest#test_user_all (17 ms warmup)
            wall_time: 0 ms
               memory: unsupported
              objects: unsupported
              gc_runs: 0
              gc_time: 0 ms

Finished in 0.572408 seconds.

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