Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2016. május 3.

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:
  email: kovacsg@tmit.bme.hu
  name: Gabor Kovacs
  encrypted_password: <%= User.encrypt PASS, SALT %>
  salt: <%= SALT %>
```

```
  birthdate: 1901-03-08 13:16:13
  account_number: 11111111-11111111-11111111

valaki:
  email: valaki@mail.bme.hu
  name: Vala Ki
  encrypted_password: <%= User.encrypt 'titok', SALT %>
  salt: <%= SALT %>
  birthdate: 1996-03-08 13:16:13
  account_number: 22222222-22222222-22222222
```

A tesztadatbázisunkban több üzenetre lesz szükségünk, ezeket a `posts.yml` fájlban definiáljuk. Mindkét üzenet a `me` kulccsal azonosított felhasználó üzenőfalára kerül. Ha a YAML fájlunkban egy ilyen kulcshoz a hivatkozott tesztadat egy kulcsát rendeljük, a Rails feloldja a kapcsolatot.

```
one:
  text: Micsoda remek hozzaszolas!
  author: valaki
  user:  me

two:
  text: Igazad van!
  author: me
  user: me
```

Egy tesztadatot veszünk fel a csatolmányok modellje (`attachments.yml`) számára. A csatolmányok egy-egy relációban áll az üzenetek modellel, és a kulcsunk itt megfelel a Rails konvencióinak. Ez lehetővé teszi számunkra a másik tesztadat kulcsával való hivatkozást.

```
one:
  mime: image/png
  post: one
  filename: users_table1.png
  original_name: users_table1.png
  size: 39728
```

Nézzük először meg a tesztadatbázisunkat:

```
kovacsg@debian:~/gyakorlat/test# rails db
mysql> use gyakorlat_test;
Database changed
mysql> show tables;
Empty set (0.00 sec)

mysql> Bye
```

Nincsenek se tábláink, se adataink a tesztadatbázisban, ezért töltsük be a teszt környezet adatbázisába ezeket a frissen létrehozott adatokat ügyelve arra, hogy a környezetként a teszt környezet legyen beállítva.

```
kovacsg@debian:~/gyakorlat/test# RAILS_ENV='test' RAILS_ENV='test
    ' rake db:test:prepare
kovacsg@debian:~/gyakorlat/test# RAILS_ENV='test' rake db:migrate
kovacsg@debian:~/gyakorlat/test# 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.

Nézzük először meg a tesztadatbázisunkat:

```
kovacsg@debian:~/gyakorlat/test# rails db
mysql> select * from users;
+-----------+--------------------+--------------+---------------------------------------

| id        | email              | name         |
    encrypted_password
                                                  | birthdate
              | account_number              | created_at
    | updated_at            | salt                |
+-----------+--------------------+--------------+---------------------------------------

|  40473535 | valaki@mail.bme.hu | Vala Ki      |
    c71d9c713039b5d828cccb47c6a6f0c337982a2a86d5f06bdc80abee756f58ef
     | 1996-03-08 13:16:13 | 22222222-22222222-22222222 |
    2016-05-03 10:31:00 | 2016-05-03 10:31:00 | ezzeltitkositunk
    |
| 743609028 | kovacsg@tmit.bme.hu | Gabor Kovacs |
    c71d9c713039b5d828cccb47c6a6f0c337982a2a86d5f06bdc80abee756f58ef
     | 1901-03-08 13:16:13 | 11111111-11111111-11111111 |
    2016-05-03 10:31:00 | 2016-05-03 10:31:00 | ezzeltitkositunk
    |
+-----------+--------------------+--------------+---------------------------------------

2 rows in set (0.00 sec)

mysql> select * from posts;
+-----------+-------------------------------+-----------+---------------------------------

| id        | text                          | author_id | user_id
    | created_at            | updated_at            |
+-----------+-------------------------------+-----------+---------------------------------

| 298486374 | Igazad van!                   | 743609028 | 743609028
    | 2016-05-03 10:31:00 | 2016-05-03 10:31:00 |
```

```
| 980190962 | Micsoda remek hozzaszolas! |   40473535 | 743609028
    | 2016-05-03 10:31:00 | 2016-05-03 10:31:00 |
+-----------+----------------------------+------------+-----------

2 rows in set (0.01 sec)
```

Lőjük le a fejlesztői üzemmódban futó webszerverünket, és indítsuk újra teszt üzemmódban, majd a webfelületen ellenőrizzük, hogy be tudunk-e lépni a tesztadatok között megadott email címet és jelszót megadva.

```
kovacsg@debian:~/gyakorlat/test/fixtures# RAILS_ENV='test' rails
    s -p 3001
```

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 :email,
    {
      presence: true,
      uniqueness: true
    }
  validates :name, presence: true
  validates :password, confirmation: true, :if => :
     password_required?
```

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-egz tesztesetet, ami a modell két példánymetódusát ellenőrzi. Az egyik az általunk írt `encrypt` osztálymetódust teszteli egy ismert tesztadat alapján, a másik a tördelés utáni tömb méretét vizsgálja.

```
require 'test_helper'

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

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

```
  test "cannot save user with exising email address" do
    u = User.new email: 'kovacsg@tmit.bme.hu', name: 'KG'
    assert !u.save, "Houston, we have a problem"
  end

  test "cannot save user if password and its confirmation do not
      match" do
    u = User.new email: 'kovacsg@bme.hu', name: 'KG', password: '
        titok1', password_confirmation: 'titok2'
    assert !u.save, "Houston, we have a problem"
  end

  test "if encrypt works properly" do
    v = users(:me)
    assert_equal Digest::SHA2.hexdigest(v.salt+'titok'), v.
        encrypted_password, "Passwords do not match after
        encryption"
  end

  test "if pagination returns less than 5 items per page" do
    v = users(:me)
    assert v.get_posts_page(1).size <= 5
  end
end
```

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# RAILS_ENV='test' rake
    test:models
(in /home/kovacsg/gyakorlat)
Run options: --seed 30080

# Running:

......

Finished in 0.193063s, 31.0780 runs/s, 31.0780 assertions/s.

6 runs, 6 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 `valaki` 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 email cím, 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 SessionsControllerTest < ActionController::TestCase
  test "login" do
    u = users(:valaki)
    post :create, { email: u.email, password: 'titok' }
    assert_not_nil session[:user]
    assert_response :redirect
    assert_redirected_to hello_path
    assert_select "span", u.name
  end

  test "logout" do
    get :destroy, nil, { user: users(:valaki).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"] = hello_path
  end
```

```
end
```

A `PostsControllerTest`-et `scaffold` 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.

Mivel az üzenet létrehozása saját nézetből az üzenőfalra került át, módosítottuk a `create` akciót, hogy az ne az üzenet `show` nézetére, hanem az előző oldalra, vagyis az üzenőfalra navigáljon vissza sikeres létrehozás után. Ez az átirányítás a tesztet is érinti, az előző példa mintájára meg kell adnunk az előző HTTP kérés URI-ját.

```
class PostsControllerTest < ActionController::TestCase
  setup do
    @request.env["HTTP_REFERER"] = hello_path
    @post = posts(:one)
  end
end
```

A felhasználók kontroller automatikus tesztjei is hibásak, a `show` és `edit` útvonalakhoz és nézetekhez utólag hozzáadtuk a felhasználó azonosítóját, így azt paraméterként át kell adnunk. Továbbá az üzenőfalunkon egy új paramétert bevezettünk az oldal tördelésére, a `page`-et, ahhoz is rendeljünk értéket. Mivel az üzenőfal megtekintése bejelentkezéshez kötött, a harmadik, `session` paraméterben adjuk át az egyik teszt felhasználó azonosítóját!

```
require 'test_helper'

  test "should get show" do
    get :show, { id: users(:me).id, page: 1 }, { user: users(:me)
        .id }
    assert_response :success
  end

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

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

A funkcionális teszteseteket a `rake test:functionals` szkripttel futtathatjuk. Azt láttuk, hogy az `assert_select` nem akaródzott az elvártaknak megfelelően működni.

```
kovacsg@debian:~/gyakorlat/test/controllers# RAILS_ENV='test'
    rake test:functionals
(in /home/kovacsg/gyakorlat)
Run options: --seed 45707

# Running:

...F.........

Finished in 0.860076s, 15.1149 runs/s, 25.5791 assertions/s.

  1) Failure:
SessionsControllerTest#test_login [/home/kovacsg/gyakorlat/test/
    controllers/sessions_controller_test.rb:14]:
Expected at least 1 element matching "span", found 0..
Expected 0 to be >= 1.

13 runs, 22 assertions, 1 failures, 0 errors, 0 skips
```

A tesztek harmadik típusa az integrációs teszt, amellyel egy böngészési folyamatot ellenőrzünk. Készítsünk egy tesztet egy üzenet küldésére, amely egy képet tartalmaz csatolmányként!

```
kovacsg@debian:~/gyakorlat/test# rails g integration_test
    send_post_image
      invoke  test_unit
      create    test/integration/send_post_image_test.rb
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `send_post_image_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 `valaki` kulcsú felhasználó kép csatolmányt tartalmazó üzenetet küld a `me` kulcsú felhasználó üzenőfalára. A böngészési folyamatot úgy definiáljuk, hogy az a bejelentkezéstől jusson el a fájlfeltöltésig.

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 pontosan egy `#email` azonosítójú és pontosan egy `#password` azonosítójú 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 ak-

ció 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 betölti a `me` kulcsú felhasználó üzenőfalát. Sikeres HTTP válasz az elvárásunk.

A negyedik tesztlépésben egy új üzenetet küldünk a felhasználónak, amihez csatolunk egy képet. A képet a fájlrendszerről olvassuk be a tesztadatok alkönyvtárában lévő `files` alkönyvtárból a `fixture_file_upload` metódussal. A fájlt előzetesen elhelyeztük ott. A fájlt HTTP POST üzenettel küldjük el a `/posts/create` akciónak, paraméterként beállítva a `user_id`, `author_id`, a `post[text]` és `post[attachment]` értékeket. Mivel sikeres létrehozás esetén az előző oldalra navigálunk, harmadik paraméterként a HTTP fejlécnek szóló értéket adunk át. Ezután megvizsgáljuk, hogy az adatbázisban létrejött-e a csatolmány, megegyezik-e a feltöltött fájl eredeti neve annak tényleges nevével, illetve megnézzük, hogy a fájlok tárolási helyén létrejött-e maga a fájl az adatbázisbeli azonosítója nevével.

```
class SendPostImageTest < ActionDispatch::IntegrationTest
  # test "the truth" do
  #   assert true
  # end
  test "send an image in a post" do
    get url_for(controller: :say, action: :hello)
    assert_response :success
    assert_select "#email", 1
    assert_select "#password", 1

    u = users(:valaki)
    post_via_redirect url_for(controller: :sessions, action: :
        create), { email: u.email, password: 'titok' }, { "
        HTTP_REFERER" => hello_path }
    assert_not_nil session[:user]
    assert_select "span", u.name

    v = users(:me)
    get url_for(controller: :users, action: :show, id: v.id)
    assert_response :success
    #assert_select "p", v.posts.count
```

```
    upload_file = fixture_file_upload(Rails.root.join('test','
        fixtures','files','users_table2.png'), 'image/png')
    post url_for(controller: :posts, action: :create),
       { user_id: v.id, author_id: u.id, post: { text: 'Ezt␣nezd␣
          meg!', attachment: upload_file} },
       { "HTTP_REFERER" => '/say/hello' }
    s = Attachment.last
    assert_equal s.original_name,'users_table2.png'
    assert File.exists?(Rails.root.join('public', 'data', s.id.
        to_s))
  end
end
```

Futtatva az integrációs tesztet hibákat találtunk az `Attachment` modell `saveFile` osztálymetódusában, a `path` változó rossz helyen volt inicializálva, illetve a `filename` attribútum helytelenül volt mentve, ugyanis az a `path` értékére hivatkozott.

```
kovacsg@debian:~/gyakorlat/test/integration# RAILS_ENV='test '
    rake test:integration
(in /home/kovacsg/gyakorlat)
Run options: --seed 28348

# Running:

.

Finished in 0.765699s, 1.3060 runs/s, 10.4480 assertions/s.

1 runs, 8 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 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 teljesítménytesztet akárcsak az integrációs teszeket explicit módon kell létrehoznunk:

```
kovacsg@debian:~/gyakorlat/test# rails g performance_test root
      create  test/performance/root_test.rb
```

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

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

class RootTest < 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/test/performance# RAILS_ENV='test'
    rake test:benchmark
(in /home/kovacsg/gyakorlat)
Run options: --seed 32425

# Running:

RootTest#test_homepage (387 ms warmup)
            wall_time: 7 ms
               memory: unsupported
              objects: unsupported
              gc_runs: 0
              gc_time: 0 ms
.

Finished in 2.209116s, 0.4527 runs/s, 0.0000 assertions/s.
```

```
1 runs, 0 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat/test/performance# RAILS_ENV='test'
    rake test:profile
(in /home/kovacsg/gyakorlat)
Run options: --seed 49271

# Running:

RootTest#test_homepage (369 ms warmup)
        process_time: 32 ms
              memory: 0 Bytes
             objects: 2,909
.

Finished in 5.830403s, 0.1715 runs/s, 0.0000 assertions/s.

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

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 63502

# Running:

BenchmarkerTest#test_user_all (16 ms warmup)
           wall_time: 0 ms
              memory: unsupported
             objects: unsupported
             gc_runs: 0
             gc_time: 0 ms
.

Finished in 1.097728s, 0.9110 runs/s, 0.0000 assertions/s.

1 runs, 0 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat# perftest profiler User.all
Run options: --seed 8344

# Running:

ProfilerTest#test_user_all (16 ms warmup)
        process_time: 1 ms
              memory: 0 Bytes
             objects: 63
```

```
.

Finished in 0.820541s, 1.2187 runs/s, 0.0000 assertions/s.

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