Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2017. november 21.

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 teszteset 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` osztálymetódusának törzsét használjuk, hiszen a szoftver akkor fog a teszteknek megfelelően működni, ha a teszteknek megfelelő titkosítást használja. 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
  encrypted_password: <%= Digest::SHA1.hexdigest('a'+'b') %>
  name: Senki
```

```
  phone: 3612345678
  salt: b

elek:
  email: teszt@elek.hu
  encrypted_password: <%= Digest::SHA1.hexdigest('b'+'c') %>
  name: Teszt Elek
  phone: 3611111111
  salt: c

<% for i in 1..3 do %>
user_<%=i%>:
  email: user_<%=i%>@mail.bme.hu
  encrypted_password: <%= Digest::SHA1.hexdigest('a'+'b') %>
  name: User <%= i%>
  phone: 3612345678
  salt: b

<% end %>
```

E fájlokba Ruby kódrészlet ágyazható be a nézeteknél megismert módon, így tesztadatok tömeges gyártása egy ciklussal megoldható.

A tesztadatbázisunkban több sporteszközre lesz szükségünk, ezeket a `items.yml` fájlban definiáljuk.

```
labda:
  name: Labda
  barcode: 11111
  price: 1500

teniszuto:
  name: Teniszuto
  barcode: 12222
  price: 5000
```

A `borrows.yml` fájlban egy több-több kapcsolat adatait adhatjuk meg. Ha a YAML fájlunkban egy idegen kulcshoz tartozó attribútuma értékének a hivatkozott típus tesztadatai közül egy kulcsát rendeljük, a Rails feloldja a kapcsolatot.

```
one:
  user: me
  item: labda
  status: 1
  return_time: <%= 1.year.ago.to_datetime %>
  created_at: <%= 2.years.ago.to_datetime %>

two:
  user: elek
```

```
  item: teniszuto
  status: 1
  return_time: <%= 1.year.ago.to_datetime %>
  created_at: <%= 2.years.ago.to_datetime %>
```

Mindkét sporteszközünkhöz rendelünk egy-egy képet (`images.yml`), és megadjuk annak metaadatait. Azok egy, már korábban feltöltött kép adatai, a kép maga pedig elérhető a Rails alkalmazás megfelelő könyvtárában.

```
one:
  path: /home/kovacsg/gyakorlat/public/data/2
  mime: image/png
  size: 29154
  filename: teniszuto.png
  item: teniszuto

two:
  path: /home/kovacsg/gyakorlat/public/data/2
  mime: image/png
  size: 29154
  filename: labda.png
  item: labda
```

Nézzük először meg a tesztadatbázisunkat:

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails db
MariaDB [gyakorlat_test]> show tables;
Empty set (0.00 sec)

MariaDB [gyakorlat_test]> 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# RAILS_ENV='test' rails db:drop
Dropped database 'gyakorlat_test'
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails db:create
Database 'gyakorlat_test' already exists
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails db:migrate
== 20170926110216 CreateUsers: migrating
   ==========================================
-- create_table(:users)
   -> 0.0157s
== 20170926110216 CreateUsers: migrated (0.0159s)
   =================================

== 20170926112507 CreateItems: migrating
   ==========================================
-- create_table(:items)
```

```
   -> 0.0154s
== 20170926112507 CreateItems: migrated (0.0162s)
   ===============================

== 20171024101520 AddSaltToUser: migrating
   ===========================================
-- add_column(:users, :salt, :string)
   -> 0.0242s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0074s
== 20171024101520 AddSaltToUser: migrated (0.0322s)
   ==============================

== 20171024112323 CreateBorrows: migrating
   ===========================================
-- create_table(:borrows)
   -> 0.0179s
== 20171024112323 CreateBorrows: migrated (0.0181s)
   ==============================

== 20171107113029 CreateImages: migrating
   ===========================================
-- create_table(:images)
   -> 0.0093s
== 20171107113029 CreateImages: migrated (0.0095s)
   ==============================

kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rails 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/fixtures> RAILS_ENV='test' rails db
MariaDB [gyakorlat_test]> select * from users;
+-----------+-----------------+------------------------------------------
| id        | email           | encrypted_password
                        | name       | phone      | created_at          |
   updated_at          | salt |
+-----------+-----------------+------------------------------------------

| 220281934 | user_3@mail.bme.hu  | da23614e02469a0d7c7bd1bdab5c9c474b1904dc
     | User 3     | 3612345678 | 2017-11-21 11:46:58 | 2017-11-21 11:46:58 |
     b    |
| 590306657 | user_1@mail.bme.hu  | da23614e02469a0d7c7bd1bdab5c9c474b1904dc
     | User 1     | 3612345678 | 2017-11-21 11:46:58 | 2017-11-21 11:46:58 |
     b    |
| 743609028 | kovacsg@tmit.bme.hu | da23614e02469a0d7c7bd1bdab5c9c474b1904dc
     | Senki      | 3612345678 | 2017-11-21 11:46:58 | 2017-11-21 11:46:58 |
     b    |
```

```
| 781962589 | teszt@elek.hu        | 5b2505039ac5af9e197f5dad04113906a9cf9a2a
      | Teszt Elek | 3611111111 | 2017-11-21 11:46:58 | 2017-11-21 11:46:58 |
      c     |
| 975572189 | user_2@mail.bme.hu   | da23614e02469a0d7c7bd1bdab5c9c474b1904dc
      | User 2     | 3612345678 | 2017-11-21 11:46:58 | 2017-11-21 11:46:58 |
      b     |
+-----------+----------------------+------------------------------------------+------------+---

5 rows in set (0.00 sec)

MariaDB [gyakorlat_test]> select * from borrows;
+-----------+-----------+-----------+--------+---------------------+---------------------+---

| id        | user_id   | item_id   | status | return_time         |
    created_at          | updated_at          |
+-----------+-----------+-----------+--------+---------------------+---------------------+---

| 298486374 | 781962589 | 430698982 |      1 | NULL                |
    2017-11-21 11:46:58 | 2017-11-21 11:46:58 |
| 980190962 | 743609028 | 688725269 |      1 | 2016-11-21 11:46:58 |
    2015-11-21 11:46:58 | 2017-11-21 11:46:58 |
+-----------+-----------+-----------+--------+---------------------+---------------------+---

2 rows in set (0.00 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# RAILS_ENV='test' rails s
```

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
  }
```

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.

```
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  test "the_truth" do
    assert true
  end

  test "cannot_save_user_without_email" do
    u = User.new
    assert !u.save, "Houston,_we_have_a_problem"
```

```
  end

  test "cannot save user with existing email" do
    u = User.new email: users(:me).email
    assert !u.save
  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. A futtatásra használhatjuk a `rake` és a `rails` parancsot is. Az utóbbi paranccsal futtathatjuk egy modell összes tesztesetét, valamint egy, fájlbeli sorszámmal megadot konkrét tesztesetét.

```
kovacsg@debian:~/gyakorlat/test/models> RAILS_ENV='test' rails
    test:models
Run options: --seed 32372

# Running:

...

Finished in 0.072159s, 41.5747 runs/s, 41.5747 assertions/s.
3 runs, 3 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    models/user_test.rb
Run options: --seed 45020

# Running:

...

Finished in 0.076731s, 39.0976 runs/s, 39.0976 assertions/s.
3 runs, 3 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    models/user_test.rb:4
Run options: --seed 47123

# Running:

.

Finished in 0.058900s, 16.9779 runs/s, 16.9779 assertions/s.
1 runs, 1 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 `me` kulcshoz tartozó felhasználó tesztadatain végezzük el. Az első a bejelentkezést, a második a kijelentkezést teszteli, nevezzük át a teszteket ennek megfelelően! A bejelentkezés eseményt a `SessionsController` `create` metódusát használja, a kijelentezés esemény pedig a `destroy` metódust. A tesztben a `post` metódust használjuk, amelynek első és egyetlen kötelező paramétere a tesztelendő URL (Rails 5 előtt tesztelendő akció). A `params` kulcshoz rendelhetjük a HTTP kérés paramétereit, a `headers` kulcshoz pedig a kiküldendő HTTP kérés fejléc beállításait. 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, viszont azt kell feltételeznünk előzetesen, hogy a felhasználó be van jelentkezve, vagyis a `session` hash `user` kulcshoz tartozó értéke létezik. Ezt úgy érthetjük el, hogy a teszteset prefixeként lefuttatjuk a bejelentkezés tesztesetet. Válaszként átirányítást várunk az aktuális oldalunkra, és azt, hogy a session paraméter kinullázódik. Mivel az átirányítás mindkét esetben a `back` URL-re történik, és a tesztvégrehajtást nem böngészőből végezzük, és nem érhető el a Javascript `history`, a `HTTP_REFERRER` fejrészt kell beállítanunk.

```
class SessionsControllerTest < ActionDispatch::IntegrationTest
  test "login" do
    post '/sessions/create',
      params: { email: users(:me).email, password: 'a' },
      headers: { "HTTP_REFERER" => '/say/hello' }
    assert_response :redirect
    assert_equal session[:user], users(:me).id
  end

  test "logout" do
    post '/sessions/create',
      params: { email: users(:me).email, password: 'a' },
      headers: { "HTTP_REFERER" => '/say/hello' }
    assert_not_nil session[:user]
    get '/sessions/destroy',
      headers: { "HTTP_REFERER" => '/say/hello' }
    assert_response :redirect
    assert_nil session[:user]
  end
end
```

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

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    controllers/sessions_controller_test.rb
```

```
Run options: --seed 40659

# Running:

..

Finished in 0.198000s, 10.1010 runs/s, 25.2525 assertions/s.
2 runs, 5 assertions, 0 failures, 0 errors, 0 skips
```

A tesztek harmadik típusa az integrációs teszt, amellyel egy sporteszköz adatlapjának módosítása folyamatát ellenőrzük.

```
kovacsg@debian:~/gyakorlat/test> rails g integration_test
    upload_image_for_item
       invoke  test_unit
       create    test/integration/upload_image_for_item_test.rb
```

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

A tesztadatok a `test_helper` fájlra való hivatkozás miatt automatikusan elérhetők. A teszt öt tesztlépésből áll:

- lekérdezünk egy képernyőt, ahol van bejelentkező form,
- kitöltjük a bejelentkezési formot, és rákattintunk bejelentkezés gombra,
- lekérdezzük egy sporteszköz adatait,
- az elem adatlapján a fájlválasztó mezőben kiválaszunk egy fájlt, majd a többi mezőt változatlanul hagyva rákattintunk a frissítés gombra.

Az első lépésben egy be nem jelentkezett felhasználó betölt egy véletlen oldalt, amin elérhető a bejelentkezési form. Itt annyit feltételezünk, hogy az oldal sikeresen betöltődik, és az oldalon HTML nézetének forrásában van pontosan egy form, valamint egy `Login` címkéjű link HTML elem.

A második tesztlépés a bejelentkezés linkre „kattintás”. Az a feltételezünk, hogy az oldal ennek hatására átirányítódunk.

Lekérdezzük a módosítani kívánt `items` erőforrás `edit` nézetét. Az a feltételezésünk, hogy az oldal sikeresen betöltődik.

A szerkesztés nézeten található egy form, amelyen kereszül `name`, `barcode`, `price` és `image` kérés paramétereket juttathatunk el az `update` akciónak HTTP PUT/PATCH üzenettel. Az első három paraméterhez a kiválasztott recept megfelelő mezőit rendeljük, az utolsó paraméternek pedig egy, a `fixture_file_upload` metódussal a tesztadatok közül, a helyi fájlrendszerről kiválasztott fájlt adunk meg. Hivatkozó oldalnak a szerkesző oldalt adjuk

meg. Azt várjuk, hogy oda irányítódunk vissza, létrejön egy `Image` rekord az adatbázisban, annak `name` attribútuma megegyezik a feltöltött fájl eredeti nevével, és a fájl létrejön a fájlrendszeren a feltöltött képek tárolására kijelölt területen.

```
require 'test_helper'

class UploadImageForItemTest < ActionDispatch::IntegrationTest
  # test "the truth" do
  #   assert true
  # end

  setup do
    labda = items(:labda)
  end

  test "upload_image_for_item" do
    get '/say/hello'
    assert_response :success
    assert_select 'legend', 'Login'

    post '/sessions/create',
      params: { email: users(:me).email, password: 'a' },
      headers: { "HTTP_REFERER" => '/say/hello' }

    get '/items/'+items(:labda).id.to_s+'/edit'
    assert_response :success

    upload_file = fixture_file_upload('test/fixtures/files/html_dom.png',
      'image/png')
    patch '/items/'+items(:labda).id.to_s,
      params: {
        item: {
          name: items(:labda).name,
          barcode: items(:labda).barcode,
          price: items(:labda).price,
          image: upload_file
        }
      }
    assert_response :redirect
    assert File.exists? 'public/data/'+items(:labda).id.to_s
  end
end
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    integration/upload_image_for_item_test.rb
Run options: --seed 47720

# Running:

.

Finished in 0.373150s, 2.6799 runs/s, 13.3994 assertions/s.
1 runs, 5 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 (és a `bundle install` paranccsal telepítenünk), 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> rails g performance_test load
      create  test/performance/load_test.rb
```

Az alábbi teszteset a kezdőoldalt támadja meg 5 egymás utáni kéréssel.

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

class LoadTest < 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`

`rails` célokkal hajthatunk végre. [1]

```
kovacsg@debian:~/gyakorlat> rails test:profile
Run options: --seed 56040

# Running:

LoadTest#test_homepage (150 ms warmup)
        process_time: 26 ms
              memory: 0 Bytes
             objects: 2,553
.

Finished in 2.771761s, 0.3608 runs/s, 0.0000 assertions/s.
1 runs, 0 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat: rails test:benchmark
Run options: --seed 44141

# Running:

LoadTest#test_homepage (122 ms warmup)
           wall_time: 4 ms
              memory: unsupported
             objects: unsupported
             gc_runs: 0
             gc_time: 0 ms
.

Finished in 1.032117s, 0.9689 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.

[1] Az ruby-prof aktuális verziójában hiba van, amelyet kijavítva kapunk csak eredményt: https://github.com/ruby-prof/ruby-prof/issues/203