Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2018. május 8.

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 későbbi migrációval módosított, illetve hozzáadott attribútumokat magunknak kell módosítanunk, illetve hozzáadnunk a struktúrához, ilyen például a `salt`.

```
me:
  id: 5555
  name: Kovacs Gabor
  email: kovacsg@tmit.bme.hu
```

```
  encrypted_password: <%= Digest::SHA1.hexdigest('aaaaaa' + '
      titok') %>
  salt: aaaaaa

two:
  id: 6666
  name: Vala Ki
  email: valaki@mail.bme.hu
  encrypted_password: <%= Digest::SHA1.hexdigest('bbbbbbb' + '
      titok' ) %>
  salt:  bbbbbbb
```

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

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

```
one:
  id: 123
  user_id: 6666
  title: teafozes tanfolyam
  description: Hogyan fozzunk finom teat?
  deadline: 2018-05-08 13:35:33
  priority: 5

two:
  user_id: 5555
  title: MyString
  description: MyText
  deadline: 2018-02-27 13:35:33
  priority: 1
```

Az `addresses.yml` fájlban egy polimorfikus egy-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. Itt a kapcsolatunk polimorfikus, ami azt jelenti, hogy meg kell adnunk a kapcsolódó tesztadat azonosítója mellett annak típusát is.

```
one:
  city: Budapest
  zip: 1117
  street: Magyar tudosok utja.
  number: 2
  others: Q.BF.09.
  addressable_type: Event
  addressable_id: 123
```

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
Created database 'gyakorlat_test'
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails db:migrate
== 20180227122321 CreateUsers: migrating ======================================
-- create_table(:users)
   -> 0.0144s
== 20180227122321 CreateUsers: migrated (0.0146s) =============================

== 20180227123533 CreateEvents: migrating =====================================
-- create_table(:events)
   -> 0.0146s
== 20180227123533 CreateEvents: migrated (0.0148s) ============================

== 20180327102232 AddSaltToUsers: migrating ===================================
-- add_column(:users, :salt, :string)
   -> 0.0147s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0053s
== 20180327102232 AddSaltToUsers: migrated (0.0203s) ==========================

== 20180327113501 CreateJoinTableEventsUsers: migrating =======================
-- create_join_table(:events, :users)
   -> 0.0077s
== 20180327113501 CreateJoinTableEventsUsers: migrated (0.0079s) ==============

== 20180327114256 CreateAddresses: migrating ==================================
-- create_table(:addresses)
```

```
   -> 0.0074s
== 20180327114256 CreateAddresses: migrated (0.0076s)
   ==========================

== 20180327114457 AddAddressableToAddress: migrating
   ==========================
-- add_reference(:addresses, :addressable, {:polymorphic=>true, :
   index=>true})
   -> 0.0339s
== 20180327114457 AddAddressableToAddress: migrated (0.0341s)
   ==========================

== 20180417111331 CreateAttachments: migrating
   ==========================
-- create_table(:attachments)
   -> 0.0096s
== 20180417111331 CreateAttachments: migrated (0.0098s)
   ==========================
```

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   | name         | email              | encrypted_password                       | created_at          | updated_at          | salt   |
+------+--------------+--------------------+------------------------------------------+---------------------+---------------------+--------+
| 5555 | Kovacs Gabor | kovacsg@tmit.bme.hu | 94c8e005827e5dad76070308d1bcb9e6ec695212 | 2018-05-11 12:33:42 | 2018-05-11 12:33:42 | aaaaaa |
| 6666 | Vala Ki      | valaki@mail.bme.hu  | 44ddfe5ccd1a1d3e79970d5c3a28c270e3d271d2 | 2018-05-11 12:33:42 | 2018-05-11 12:33:42 | bbbbbbb |
+------+--------------+--------------------+------------------------------------------+---------------------+---------------------+--------+
2 rows in set (0.00 sec)

MariaDB [gyakorlat_test]> select * from events;
+-----------+---------+-----------------+-----------------------+---------------------+---------------------+---------------------+
| id        | user_id | title           | description           | deadline            | priority | created_at          | updated_at          |
+-----------+---------+-----------------+-----------------------+---------------------+---------+---------------------+---------------------+
|       123 |    6666 | teafozes tanfolyam | Hogyan fozzunk finom teat? | 2018-05-08 13:35:33 |        5 | 2018-05-11 12:33:42 | 2018-05-11 12:33:42 |
| 298486374 |    5555 | MyString        | MyText                | 2018-02-27 13:35:33 |        1 | 2018-05-11 12:33:42 | 2018-05-11
```

```
    12:33:42 |
+-----------+----------+-------------------+-----------------------------+---------------------+
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.

```
class UserTest < ActiveSupport::TestCase
  test "the truth" do
    assert true
  end

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

  test "cannot save user with existing email address" do
    u = User.new name: "Valaki", email: 'valaki@mail.bme.hu'
    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> RAILS_ENV='test' rails test:models
Run options: --seed 38826
```

```
# Running:

...

Finished in 0.071628s, 41.8830 runs/s, 41.8830 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 9761

# Running:

...

Finished in 0.062710s, 47.8396 runs/s, 47.8396 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:8
Run options: --seed 30408

# Running:

.

Finished in 0.062324s, 16.0452 runs/s, 16.0452 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 `two` 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. Az átirányítás válaszban kapott URI-t a `follow_redirect!` függvénnyel nyithatjuk meg, és azon az `assert_select` függvénnyel ellenőrizthetjük, hogy tényleg az az oldal, és tényleg azzal a tartalommal jelent-e meg.

```
class SessionsControllerTest < ActionDispatch::IntegrationTest
  setup do
    @valaki = users(:two)
  end

  test "login" do
    post "http://localhost:3000#{login_path}", params: { email: @valaki.
        email, password: 'titok'},
 as: :html,
      headers: { "HTTP_REFERER": "http://localhost:3000#{welcome_path}" }
    assert_response :redirect
    follow_redirect!
    assert_equal @valaki.id, session[:user]
    assert_select 'div#content'
  end

  test "logout" do
    post "http://localhost:3000#{login_path}", params: { email: @valaki.
        email, password: 'titok'},
      headers: { "HTTP_REFERER": "http://localhost:3000#{welcome_path}" }
    assert_equal @valaki.id, session[:user]
    get "http://localhost:3000#{logout_path}"
    assert_response :redirect
    follow_redirect!
    assert_select 'h1', "Hello world"
    assert_nil session[:user]
  end

end
```

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

```
kovacsg@debian:~/gyakorlat> rails test test/controllers/
    sessions_controller_test.rb
Run options: --seed 11063

# Running:

..

Finished in 0.351358s, 5.6922 runs/s, 19.9227 assertions/s.
```

```
2 runs, 7 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
    edit_user_profile
      invoke  test_unit
      create    test/integration/edit_user_profile_test.rb
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `edit_user_profile_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 a profilszerkesztő oldalt,
- módosítjuk a profilt és egyben feltöltünk egy profilképet.

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, és a felhasználói `session` beállítódik.

Lekérdezzük a módosítani kívánt felhasználó `edit` nézetét. Az a feltételezésünk, hogy az oldal sikeresen betöltődik, és a tartalma az elvárt.

A szerkesztés nézeten található egy form, amelyen kereszül `name`, `email`, `password` , `password_confirmation` és `attachment` kérés paramétereket juttathatunk el az `update` akciónak HTTP PUT/PATCH üzenettel. Az első négy paraméterhez random értékeket rendelünk, 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 az adatbázisban feltöltött fájl méreteként a tesztfájl méretét látjuk, és a fájl megjelenik az elvárt névvel a fájlrendszeren.

```
require 'test_helper'

class EditUserProfileTest < ActionDispatch::IntegrationTest
```

```
  # test "the truth" do
  #   assert true
  # end
  setup do
    @valaki = users(:two)
  end

  test "upload attachment for user profile" do
    get welcome_path
    assert_response :success
    assert_select 'legend', 'Login'

    post login_path, params: { email: @valaki.email, password: 'titok' },
      headers: { "HTTP_REFERER": "http://localhost:3001#{welcome_path}" }
    assert_equal session[:user], @valaki.id
    assert_response :redirect

    get  edit_profile_path(@valaki.id)
    assert_response :success
    assert_select 'legend', "Edit user profile"

    upload_file = fixture_file_upload('test/fixtures/files/kep.png', 'image/
        png')
    put update_profile_path(@valaki.id),
      params: { user:
        { name: '', email: '', password: '', password_confirmation: '',
        attachment: upload_file }
    }
    assert_equal @valaki.attachment.size, 46273
    assert File.exists? Rails.root.join("public", "data", @valaki.attachment
        .id.to_s)

  end
end
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/gyakorlat> rails test test/integration/
    edit_user_profile_test.rb
Run options: --seed 62911

# Running:

.

Finished in 0.486663s, 2.0548 runs/s, 16.4385 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 (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 teljesítményteszteket.

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 hello
      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 HelloTest < 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 11411
```

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

```
# Running:

HelloTest#test_homepage (135 ms warmup)
        process_time: 26 ms
              memory: 0 Bytes
             objects: 2,526
.

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 50734

# Running:

HelloTest#test_homepage (127 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.