Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2023. november 21.

1. Tesztelés Railsben

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.

Lőjük le a fejlesztői üzemmódban futó webszerverünket, és indítsuk újra teszt üzemmódban, a `RAILS_ENV` környezeti változót beállítva.

```
kovacsg@debian:~/gyakorlat$ RAILS_ENV='test' rails s -b
    10.211.55.3
```

2. Tesztadatok felvétele, betöltése

Először is nézzük meg a teszt adatbázisunkat. A táblák egy korábbi változata már létezik, viszont azok üresek.

```
kovacsg@debian:~/gyakorlat$ RAILS_ENV='test' rails db
MariaDB [gyakorlat_test]> show tables;
+-----------------------------+
| Tables_in_gyakorlat_test    |
+-----------------------------+
| ar_internal_metadata        |
| schema_migrations           |
```

```
| submissions                |
| tasks                      |
| users                      |
+----------------------------+
5 rows in set (0.001 sec)

MariaDB [gyakorlat_test]> select * from users;
Empty set (0.001 sec)
```

Ezért juttassuk érvényre az adatmodellünkben bekövetkezett változásokat egy migráció futtatásával.

```
kovacsg@debian:~/gyakorlat/test/fixtures$ RAILS_ENV='test' rails
    db:migrate
== 20231114114606 CreateAttachments: migrating
    ==========================================
-- create_table(:attachments)
   -> 0.0177s
== 20231114114606 CreateAttachments: migrated (0.0183s)
    =================================

== 20231114123620 AddRoleToUsers: migrating
    ============================================
-- add_column(:users, :role, :integer, {:limit=>1, :default=>1})
   -> 0.0078s
== 20231114123620 AddRoleToUsers: migrated (0.0175s)
    ===================================
```

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 néhány `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` és a `role`.

```
hallgato:
  name: Teszt Elek
```

```
  email: elek@mail.bme.hu
  neptun: aaaaaa
  encrypted_password: <%= Digest::SHA1.hexdigest "titokhello"%>
  salt: hello
  role: 1
```

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

A tesztadatbázisunkban több feladatra lesz szükségünk, ezeket a `tasks.yml` fájlban definiáljuk. Itt nem volt változás a struktúrában, csak az adatokat módosítjuk.

```
one:
  number: 1
  url: http://gyakorlat.com/tasks/1
  description: Hany eves vagy?
  deadline: 2023-10-24

two:
  number: 2
  url: http://gyakorlat.com/tasks/2
  description: Hany ora van?
  deadline: 2023-10-31
```

A megoldások kvázi kapcsolótáblaként viselkedik a felhasználók és a feladatok között, három idegen kulcsunk is van. 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: hallgato
  task: one

two:
  user: hallgato
  task: two
```

Csatolmányokat nem veszünk fel előzetesen, ezért töröljük azon tesztfájlok tartamát.

Töltsük be a tesztadatokat a teszt adatbázisba:

```
kovacsg@debian:~/gyakorlat/test/fixtures$ 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        | name        | email           | neptun  | encrypted_password                       | created_at          | updated_at          | salt | role |
+-----------+-------------+-----------------+---------+------------------------------------------+---------------------+---------------------+------+
| 295950541 | Teszt Elek  | elek@mail.bme.hu | aaaaaa  | 76810c204dc2c0442b7f3ebbeba6f5dce98a231c | 2023-11-21 11:37:04.913881 | 2023-11-21 11:37:04.913881 | hello |    1 |
+-----------+-------------+-----------------+---------+------------------------------------------+---------------------+---------------------+------+

1 row in set (0.000 sec)

MariaDB [gyakorlat_test]> select * from tasks;
+-----------+--------+-------------------------------+-----------------+---------------------+---------------------+
| id        | number | url                           | description     | deadline    | created_at          | updated_at          |
+-----------+--------+-------------------------------+-----------------+---------------------+---------------------+
| 298486374 |      2 | http://gyakorlat.com/tasks/2 | Hany ora van?   | 2023-10-31 | 2023-11-21 11:37:04.912383 | 2023-11-21 11:37:04.912383 |
| 980190962 |      1 | http://gyakorlat.com/tasks/1 | Hany eves vagy? | 2023-10-24 | 2023-11-21 11:37:04.912383 | 2023-11-21 11:37:04.912383 |
+-----------+--------+-------------------------------+-----------------+---------------------+---------------------+

2 rows in set (0.000 sec)

MariaDB [gyakorlat_test]> select * from submissions;
+-----------+-----------+-----------+---------------------+---------------------+
| id        | user_id   | task_id   | created_at          | updated_at          |
+-----------+-----------+-----------+---------------------+---------------------+
| 298486374 | 295950541 | 298486374 | 2023-11-21 11:37:04.908203 | 2023-11-21 11:37:04.908203 |
| 980190962 | 295950541 | 980190962 | 2023-11-21 11:37:04.908203 | 2023-11-21 11:37:04.908203 |
+-----------+-----------+-----------+---------------------+---------------------+

2 rows in set (0.001 sec)
```

3. Modellek tesztelése

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 :name, presence: true
  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.

Ellenőrizzük továbbá, hogy a jelszó titkosítása tényleg az `SHA1` osztály `hexdigest` metódusával történik azt megfelelően paraméterezve.

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

  test "cannot_save_user_without_name" do
    u = User.new email: 'senki@mail.bme.hu', password: 'titok'
    assert !u.save, "Houston,_we_have_a_problem"
  end

  test "cannot_save_user_without_email" do
    u = User.new name: "Shen_Ki", password: 'titok'
    assert !u.save, "Houston,_we_have_a_problem"
  end

  test "cannot_save_user_with_existing_email" do
    u = User.new name: "Shen_Ki", email: users(:hallgato).email, password: '
        titok'
    assert !u.save, "Houston,_we_have_a_problem"
  end

  test "encryption" do
    assert_equal Digest::SHA1.hexdigest("titok#{users(:hallgato).salt}"),
        users(:hallgato).encrypted_password
  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 test/
    models/user_test.rb:4
Run options: --seed 22438

# Running:

.

Finished in 0.045583s, 21.9379 runs/s, 21.9379 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    models/user_test.rb
Run options: --seed 29298

# Running:

.....

Finished in 0.061220s, 81.6729 runs/s, 81.6729 assertions/s.
5 runs, 5 assertions, 0 failures, 0 errors, 0 skips
```

4. Útvonalak tesztelése

Az útvonalak és a kontrollerek kapcsolatának ellenőrzése előfeltétele a kontrollerek tesztelésének.

Útvonalak megfelelő generálását Rails konzolon ellenőrizhetjük, ehhez be kell töltenünk az `url_helper` modult. Ezután az összes, a `routes.rb`-ben definiált útvonal helpert ellenőrizhetjük, valamint a kontroller tesztekben használható `url_for` helperrel útvonalakat rakhatunk össze.

```
kovacsg@debian:~/gyakorlat/test/models$ rails c
irb(main):003> include Rails.application.routes.url_helpers
=> Object
irb(main):004> hello_path
=> "/say/hello"
irb(main):005> task_path
/var/lib/gems/3.1.0/gems/actionpack-7.0.8/lib/action_dispatch/
    journey/formatter.rb:44:in `path': No route matches {:action
    =>"show", :controller=>"tasks"}, missing required keys: [:id]
    (ActionController::UrlGenerationError)
irb(main):006> task_path(1)
=> "/tasks/1"
irb(main):007> task_url(1)
/var/lib/gems/3.1.0/gems/actionpack-7.0.8/lib/action_dispatch/
    http/url.rb:64:in `full_url_for': Missing host to link to!
    Please provide the :host parameter, set default_url_options[:
    host], or set :only_path to true (ArgumentError)
irb(main):008> default_url_options[:host] = 'http://gyakorlat.com
    '
=> "http://gyakorlat.com"
irb(main):009> task_url(1)
=> "http://gyakorlat.com/tasks/1"
irb(main):011> url_for controller: 'tasks', action: 'show', id: 1
=> "http://gyakorlat.com/tasks/1"
```

5. Kontrollerek és nézetek tesztelése

A kontroller, vagy korábbi nevén 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 teszteset.

Először egészítsük ki a `SayControllerTest` tesztet, amely jelenleg egy tesztesetet tartalmaz. Ellenőrizzük, hogy az oldal betölthető-e, azon van-e egy `legend` HTML elem `Login` értékkel, és hogy a session inicializálatlan-e.

```
class SayControllerTest < ActionDispatch::IntegrationTest
  test "should_get_hello" do
    get url_for(controller: 'say', action: 'hello')
    assert_response :success
    assert_select 'legend', 'Login'
    assert_nil session[:user]
  end
end
```

A funkcionális teszteseteket a `rake test:controllers` szkripttel vagy a `rails test` paranccsal futtathatjuk. A kimeneten a `.` azt jelenti, hogy egy `assert` teljesült, az `F` azt, hogy nem teljesült, és az `E`, hogy hiba van vagy a tesztesetben, vagy a kódban, és így a teszteset nem tudott lefutni.

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/controllers/
    say_controller_test.rb
Run options: --seed 37732

# Running:

.

Finished in 0.210188s, 4.7577 runs/s, 14.2730 assertions/s.
1 runs, 3 assertions, 0 failures, 0 errors, 0 skips
```

Módosítsuk a `SessionsControllerTest` tesztet A funkciót a `hallgato` 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`, hanem a bejelentkezett felhasználó azonosítójával megegyező érték, továbbá az átirányítás utáni oldalon van egy a felhasználói

profil szerkesztésére mutató link. 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. A sikertelen bejelentkezést ellenőrző tesztesetben a jelszó paraméterben teszünk különbséget, és azt várjuk, hogy visszairányítódunk az előző oldalra, a session üres, és van az oldalon egy `Login` címkéjű `legend` HTML tag. 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, és az átirányítás után oldalon lesz egy `Register` értékű link HTML elem. 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
    get hello_path
    assert_response :success
    assert_select 'legend', "Login"
    assert_nil session[:user]

    post login_path, params: { email: users(:hallgato).email, password: '
        titok' }, headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    follow_redirect!
    assert_select 'a', 'Logout'
    assert_not_nil session[:user]
  end

  test "invalid_login" do
    get hello_path
    assert_response :success
    assert_select 'legend', "Login"
    assert_nil session[:user]

    post login_path, params: { email: 'senki@mail.bme.hu', password: 'titok'
        }, headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    follow_redirect!
    assert_select 'legend', "Login"
    assert_nil session[:user]
  end

   test "logout" do
    get hello_path
    assert_response :success
    assert_select 'legend', "Login"
    assert_nil session[:user]

    post login_path, params: { email: users(:hallgato).email, password: '
        titok' }, headers: { 'HTTP_REFERER': hello_path }
```

```
    assert_response :redirect
    follow_redirect!
    assert_select 'a', 'Logout'
    assert_not_nil session[:user]

    get logout_path
    assert_response :redirect
    follow_redirect!
    assert_nil session[:user]
    assert_select 'legend', "Login"
  end
end
```

A tesztet e kontrollerteszt kiválasztásával futtatjuk:

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    controllers/sessions_controller_test.rb
Run options: --seed 4068

# Running:

...

Finished in 0.268843s, 11.1589 runs/s, 78.1124 assertions/s.
3 runs, 21 assertions, 0 failures, 0 errors, 0 skips
```

6. Integrációs teszt

A tesztek harmadik típusa az integrációs teszt, amellyel egy hallgató által egy feladat megoldásához csatolmány feltöltésének folyamatát ellenőrzük.

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

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

A tesztadatok a `test_helper` fájlra való hivatkozás miatt automatikusan elérhetők. A teszt hat tesztlépésből áll egy felhasználó számára:

- lekérdezünk egy képernyőt, ahol van bejelentkező form,
- kitöltjük a bejelentkezési formot, és rákattintunk bejelentkezés gombra,
- lekérdezzük a feladatok listáját,
- kiválasztunk egy feladatot,

- csatolunk egy fájlt a feladathoz megoldásként, és
- kilépünk.

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, a session üres, és az oldalon HTML nézetének forrásában van egy `Login` címkéjű `legend` HTML elem.

A második tesztlépés a bejelentkezés linkre „kattintás". Az a feltételezünk, hogy átirányítás történik, a felhasználói `session` beállítódik, és az átirányítás utáni oldalon van egy `Logout` felíratú link.

A feladatok linkre kattintva lekérdezzük a feladatok listáját az `index` nézettel. Az a feltételezésünk, hogy a betöltés 2xx státusz kóddal történik meg, és annyiszor 4 darab `strong` HTML taget találunk, ahány tesztadatunk van a feladatok típusból tekintettel arra, hogy a feladat modellnek négy attribútuma van, és azok `strong` elemként jelennek meg.

A feladatok listája nézeten az egyik feladatok melletti `Show` linkre kattintva a feladat adatlapja betölthető. A tesztesetben az első sorszámú feladatot választjuk ki. Az a feltételezésünk, hogy az oldal betöltődik, és ott van egy `Submit solution` felíratú link, amelyre kattintva bejön a megoldást feltöltő form. Továbbá feltételezzük, hogy az adatbázisban nincs egyetlen csatolmányunk sem.

A megoldás feltöltése nézeten található egy form, amelyen kereszül egy `attachment` nevű paramétert juttathatunk el az `upload_path` útvonalra HTTP POST üzenettel. A paraméternek egy, a `fixture_file_upload` metódussal a tesztadatok közül, a helyi fájlrendszerről kiválasztott fájlt adunk meg. Azt várjuk, hogy az adatbázisban a feltöltött csatolmányok metaadatainak száma eggyel nő a feltöltés előtti állapothoz képest, és a fájl megjelenik az elvárt névvel a fájlrendszeren.

Végül a funkcionális tesztből ismert módon kilépünk, és azt feltételezzük, hogy a session törlődik, és az átirányítás utáni oldalon van egy `Login` értékű `legend` HTML elem.

```
require "test_helper"

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

  def setup
    @user = users(:hallgato)
    @task = tasks(:one)
  end

  def teardown
  end
```

```
  test "submit a solution" do
    get hello_path
    assert_response :success
    assert_select 'legend', "Login"
    assert_nil session[:user]

    post login_path, params: { email: users(:hallgato).email, password: '
        titok' }, headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    follow_redirect!
    assert_select 'a', 'Logout'
    assert_not_nil session[:user]

    get tasks_path
    assert_response :success
    assert_select 'strong', Task.all.size * 4

    get task_path(@task.id)
    assert_response :success
    assert_select 'a', 'Submit solution'
    assert_equal 0, Attachment.all.size

    upload_file = fixture_file_upload('test/fixtures/files/rails_hello.png')
    post  upload_path, params: { submission: { attachment: upload_file,
        task_id: @task.id } }, headers: { "HTTP_REFERER": hello_path }
    assert_response :redirect
    follow_redirect!
    assert_equal 1, Attachment.all.size
    assert File.exists?("public/data/#{Attachment.first.id}")
    assert 'a', "Logout"

    get logout_path
    assert_response :redirect
    follow_redirect!
    assert_nil session[:user]
    assert_select 'legend', "Login"..
  end
end
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    integration
Run options: --seed 64519

# Running:

/home/kovacsg/gyakorlat/app/models/attachment.rb:8: warning: File
    .exists? is deprecated; use File.exist? instead
/home/kovacsg/gyakorlat/test/integration/submit_solution_test.rb
    :42: warning: File.exists? is deprecated; use File.exist?
    instead
.

Finished in 0.303652s, 3.2932 runs/s, 59.2784 assertions/s.
```

```
1 runs, 18 assertions, 0 failures, 0 errors, 0 skips
```

A feltöltött fájl a fájlrendszeren ugyan megmaradt, de a teszt adatbázisunk visszakerült az alapállapotába.

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails db

MariaDB [gyakorlat_test]> select * from attachments;
Empty set (0.000 sec)
```

7. Rendszertesztek

A rendszertesztek fekete dobozos tesztek, a rendszert úgy ellenőrzik, ahogy azt a felhasználó látja, és nem használnak fel belső információt. A futtatáshoz szükségünk lesz egy telepített Google Chrome vagy Firefox böngészőre, a többit a végrehajtó rendszer elintézi.

A rendszerteszteket explicit paranccsal kell létrehoznunk.

```
kovacsg@debian:~/gyakorlat> rails g system_test hello
      invoke  test_unit
      create    test/system/hellos_test.rb
```

A bejelentkezés rendszertesztje a következőképp néz ki. Megnyitunk egy oldalt, kitöltjük az egyes címkékhez tartozó beviteli mezőket, rákattintunk a nyomógombra, és megnézzük a következő oldal tartalmát.

```
require "application_system_test_case"

class HellosTest < ApplicationSystemTestCase
  test "visiting the root page" do
    visit 'say/hello'
    assert_selector 'legend', text: "Login"

    fill_in "Email", with: 'elek@mail.bme.hu'
    fill_in "Password", with: 'titok'
    click_on "Login"

    assert_selector 'a', text: "Logout"
  end
end
```

A teszt a Chrome böngészővel nem működött, ezért lecseréltük azt Firefoxra az `application_system_test_case.rb` fájlban.

```
class ApplicationSystemTestCase < ActionDispatch::SystemTestCase
  driven_by :selenium, using: :firefox, screen_size: [1400, 1400]
end
```

A rendszertesztek futtatásakor meg-megnyílik a böngésző, és láthatjuk a tesztek végrehajtátás.

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    system/hellos_test.rb
Run options: --seed 54518

# Running:

Capybara starting Puma...
* Version 5.6.7 , codename: Birdie's Version
* Min threads: 0, max threads: 4
* Listening on http://127.0.0.1:35307
.

Finished in 6.110634s, 0.1636 runs/s, 0.3273 assertions/s.
1 runs, 2 assertions, 0 failures, 0 errors, 0 skips
```

8. Teljesítménytesztek

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 hello
      create  test/performance/hello_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:benchmark` és a `test:profile` `rails` célokkal hajthatunk végre.

A teljesítménytesztek függvénykönyvtáraiban jelenleg inkompatibilitás van, így a teszteket nem tudtuk végrehajtani.

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.