Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2020. november 24.

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, 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
```

2. Tesztadatok felvétele, betöltése

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`.

```
me:
  name: En
  encrypted_password: <%= Digest::SHA1.hexdigest('titokhello') %>
  email: en@mail.bme.hu
  salt: hello

nezo:
  name: Nezo
  encrypted_password: <%= Digest::SHA1.hexdigest('titokbello') %>
  email: nezo@mail.bme.hu
  salt: bello
```

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

A tesztadatbázisunkban több videóra lesz szükségünk, ezeket a `videos.yml` fájlban definiáljuk. Itt nem volt változás a struktúrában, csak az adatokat módosítjuk, a tagek kapcsolatot kihagyjuk. Az egyes videók tulajdonosa egy felhasználó, amelyre egy felhasználó tesztadatának kulcsával hivatkozhatunk, legyen a tulajdonos a `me` kulcsú felhasználó.

```
one:
  title: swe-eng
  user: me

two:
  title: swe-eng2
  user: me
```

A videókhoz csatolmány tartozik, ugyanazt a fájlt rendeljük mindkét videóhoz. A mime attribútumot utólag adtuk hozzá, ezért hozzá kell adnunk a YAML fájlhoz.

```
one:
  path: public/videos/swe-eng.mp4
  filename: swe-eng.mp4
  video: one
  size: 23519740
  mime: video/mp4

two:
```

```
  path: public/videos/swe-eng.mp4
  filename: swe-eng.mp4
  video: two
  size: 23519740
  mime: video/mp4
```

Töröljük, majd hozzuk újra létre a tesztadatbázist, utána 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/fixtures> RAILS_ENV='test' rails
    db:drop
Dropped database 'gyakorlat_test'
kovacsg@debian:~/gyakorlat/test/fixtures> RAILS_ENV='test' rails
    db
ERROR 1049 (42000): Unknown database 'gyakorlat_test'
kovacsg@debian:~/gyakorlat/test/fixtures> RAILS_ENV='test' rails
    db:create
Created database 'gyakorlat_test'
kovacsg@debian:~/gyakorlat/test/fixtures> RAILS_ENV='test' rails
    db
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 159
Server version: 10.3.23-MariaDB-0+deb10u1 Debian 10

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and
    others.

Type 'help;' or '\h' for help. Type '\c' to clear the current
    input statement.

MariaDB [gyakorlat_test]> show tables;
Empty set (0.000 sec)

MariaDB [gyakorlat_test]> Bye
kovacsg@debian:~/gyakorlat/test/fixtures> RAILS_ENV='test' rails
    db:migrate
== 20200929112630 CreateUsers: migrating
   ===============================================
-- create_table(:users)
   -> 0.0167s
== 20200929112630 CreateUsers: migrated (0.0170s)
   ======================================

== 20201013110647 CreateVideos: migrating
   ===============================================
-- create_table(:videos)
   -> 0.0179s
== 20201013110647 CreateVideos: migrated (0.0183s)
```

```
    ==========================

== 20201013114030 CreateTags: migrating
    ==========================================
-- create_table(:tags)
   -> 0.0094s
== 20201013114030 CreateTags: migrated (0.0098s)
    ==========================

== 20201027111905 AddSaltToUsers: migrating
    ==========================================
-- add_column(:users, :salt, :string)
   -> 0.0030s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0046s
== 20201027111905 AddSaltToUsers: migrated (0.0083s)
    ==========================

== 20201027123209 TagsVideos: migrating
    ==========================================
-- create_join_table(:tags, :videos)
   -> 0.0118s
== 20201027123209 TagsVideos: migrated (0.0121s)
    ==========================

== 20201110115054 CreateAttachments: migrating
    ==========================================
-- create_table(:attachments)
   -> 0.0208s
== 20201110115054 CreateAttachments: migrated (0.0249s)
    ==========================

== 20201110120544 AddMimeToAttachments: migrating
    ==========================================
-- add_column(:attachments, :mime, :string)
   -> 0.0053s
== 20201110120544 AddMimeToAttachments: migrated (0.0057s)
    ==========================
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 | encrypted_password                       | email
                | created_at                 | updated_at                 |
    salt   |
+-----------+------+------------------------------------------+-------------------+-----------

|  743609028 | En   | 76810c204dc2c0442b7f3ebbeba6f5dce98a231c | en@mail.bme
    .hu    | 2020-11-24 11:33:12.577754 | 2020-11-24 11:33:12.577754 | hello
    |
| 1014807380 | Nezo | 98087f96070ccc0fadae34b269eae6d6d80245c6 | nezo@mail.
    bme.hu | 2020-11-24 11:33:12.577754 | 2020-11-24 11:33:12.577754 | bello
     |
+-----------+------+------------------------------------------+-------------------+-----------

2 rows in set (0.000 sec)

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

| id        | title     | user_id   | created_at                 | updated_at
             |
+-----------+-----------+-----------+----------------------------+----------------------------+

| 298486374 | swe-eng2 | 743609028 | 2020-11-24 11:33:12.574602 | 2020-11-24
     11:33:12.574602 |
| 980190962 | swe-eng  | 743609028 | 2020-11-24 11:33:12.574602 | 2020-11-24
     11:33:12.574602 |
+-----------+-----------+-----------+----------------------------+----------------------------+

2 rows in set (0.000 sec)

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

| id        | path                      | filename    | video_id  | size
        | created_at                 | updated_at                 | mime
         |
+-----------+---------------------------+-------------+-----------+----------+-----------

| 298486374 | public/videos/swe-eng.mp4 | swe-eng.mp4 | 298486374 | 23519740
     | 2020-11-24 11:33:12.575842 | 2020-11-24 11:33:12.575842 | video/mp4 |
| 980190962 | public/videos/swe-eng.mp4 | swe-eng.mp4 | 980190962 | 23519740
     | 2020-11-24 11:33:12.575842 | 2020-11-24 11:33:12.575842 | video/mp4 |
+-----------+---------------------------+-------------+-----------+----------+-----------

2 rows in set (0.000 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.

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

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

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

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

  test 'encrpytion' do
    u = User.find users(:me).id
    assert_equal u.encrypted_password, Digest::SHA1.hexdigest('titokhello'),
        "Baj_van"
  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
Running via Spring preloader in process 2544
Run options: --seed 28680

# Running:

DEPRECATION WARNING: Uniqueness validator will no longer enforce
    case sensitive comparison in Rails 6.1. To continue case
    sensitive comparison on the :email attribute in User model,
    pass `case_sensitive: true` option explicitly to the
    uniqueness validator. (called from block in <class:UserTest>
    at /home/kovacsg/gyakorlat/test/models/user_test.rb:20)
```

```
.DEPRECATION WARNING: Uniqueness validator will no longer enforce
    case sensitive comparison in Rails 6.1. To continue case
    sensitive comparison on the :email attribute in User model,
    pass `case_sensitive: true` option explicitly to the
    uniqueness validator. (called from block in <class:UserTest>
    at /home/kovacsg/gyakorlat/test/models/user_test.rb:10)
....

Finished in 0.065944s, 75.8219 runs/s, 75.8219 assertions/s.
5 runs, 5 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    models/user_test.rb:8
Running via Spring preloader in process 4401
Run options: --seed 8009

# Running:

DEPRECATION WARNING: Uniqueness validator will no longer enforce
    case sensitive comparison in Rails 6.1. To continue case
    sensitive comparison on the :email attribute in User model,
    pass `case_sensitive: true` option explicitly to the
    uniqueness validator. (called from block in <class:UserTest>
    at /home/kovacsg/gyakorlat/test/models/user_test.rb:10)
.

Finished in 0.060106s, 16.6374 runs/s, 16.6374 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

4. Útvonalak tesztelése

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.

```
irb(main):017:0> include Rails.application.routes.url_helpers
=> Object
irb(main):018:0> hello_path
=> "/say/hello"
irb(main):019:0> hello_url
Traceback (most recent call last):
        1: from (irb):19
ArgumentError (Missing host to link to! Please provide the :host
    parameter, set default_url_options[:host], or set :only_path
    to true)
irb(main):020:0> default_url_options[:host] = 'http://localhost
    :3000'
```

```
=> "http://localhost:3000"
irb(main):021:0> hello_url
=> "http://localhost:3000/say/hello"
irb(main):022:0> url_for controller: 'say', action: 'hello'
=> "http://localhost:3000/say/hello"
irb(main):023:0> url_for controller: 'videos', action: 'show', id
    : 1
=> "http://localhost:3000/videos/1"
irb(main):024:0> url_for controller: 'videos', action: 'index',
    page: 1
=> "http://localhost:3000/videos/page/1"
irb(main):025:0> url_for controller: 'videos', action: 'create'
=> "http://localhost:3000/videos"
```

5. Kontrollerek és nézetek tesztelése

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 `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
Running via Spring preloader in process 4740
Run options: --seed 63356

# Running:

.

Finished in 0.260133s, 3.8442 runs/s, 11.5326 assertions/s.
1 runs, 3 assertions, 0 failures, 0 errors, 0 skips
```

Módosítsuk a `SessionsControllerTest` tesztet A funkciót a `student` 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 `Register` címkéjű link. 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 '/say/hello'
    assert_response :success
    assert_select 'legend', 'Login'
    assert_nil session[:user]

    post '/sessions/create', params: { email: users(:me).email, password: '
        titok' }, headers: { 'HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    assert_equal session[:user], users(:me).id
    follow_redirect!
    assert_select 'a', 'Profile'
  end

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

```
    assert_nil session[:user]

    post '/sessions/create', params: { email: users(:me).email, password: '
        titok2' }, headers: { 'HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    assert_nil session[:user]
    follow_redirect!
    assert_select 'a', 'Register'
  end

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

    post '/sessions/create', params: { email: users(:me).email, password: '
        titok' }, headers: { 'HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    assert_equal session[:user], users(:me).id
    follow_redirect!
    assert_select 'a', 'Profile'

    get '/sessions/destroy', headers: { 'HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    assert_nil session[:user]
    follow_redirect!
    assert_select 'a', 'Register'
  end
end
```

A tesztet e kontrollerteszt kiválasztásával futtatjuk:

```
kovacsg@debian:~/gyakorlat$ RAILS_ENV='test' rails test test/
    controllers/sessions_controller_test.rb
Running via Spring preloader in process 6287
Run options: --seed 42168

# Running:

...

Finished in 0.378673s, 7.9224 runs/s, 55.4568 assertions/s.
3 runs, 21 assertions, 0 failures, 0 errors, 0 skips
```

6. Integrációs teszt

A tesztek harmadik típusa az integrációs teszt, amellyel egy videó feltöltésének folyamatát ellenőrzük.

```
kovacsg@debian:~/gyakorlat$ rails g integration_test upload_video
Running via Spring preloader in process 6907
      invoke  test_unit
      create    test/integration/upload_video_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 videók listáját,
- rákattintunk az új videó linkre,
- kitöltjük a formot, és feltöltjük a videót, és
- kilépünk.

Az első lépésben egy be nem jelentkezett felhasználó betölt egy véletlen oldalt, amelyen elérhető a bejelentkezési form. Itt annyit feltételezünk, hogy az oldal sikeresen betöltődik, a session üres, és az oldalon HTML nézetének forrásában van egy `Register` címkéjű link HTML elem, valamint egy `Email` és egy `Password` címkéjű HTML címke.

A második tesztlépés a bejelentkezés linkre „kattintás”. Az a feltételezünk, hogy a felhasználói `session` beállítódik, és az átirányítás utáni oldalon a menüben elérhetők a videók listája link.

A videók linkre kattintva lekérdezzük a videók listáját az `index` nézettel. Az a feltételezésünk, hogy ott lesz egy Videos címkéjű H1 elem.

A videók listája nézeten, a táblázat alatti `New video` linkre kattintva új videó hozható létre. Az a feltételezésünk, hogy az oldal betöltődik, és ott van egy darab form.

Az új videó nézeten található egy form, amelyen kereszül egy `file` nevű paramétert juttathatunk el az `update` akciónak 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 állományok 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 `Register` értékű link HTML elem.

```
require 'test_helper'

class UploadVideoTest < ActionDispatch::IntegrationTest
  # test "the truth" do
```

```
  #   assert true
  # end
  test 'upload_video' do
    get '/say/hello'
    assert_response :success
    assert_select 'legend', 'Login'
    assert_nil session[:user]

    post '/sessions/create', params: { email: users(:me).email, password: '
        titok' }, headers: { 'HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    assert_equal session[:user], users(:me).id
    follow_redirect!
    assert_select 'a', 'Profile'

    get '/videos'
    assert_response :success
    assert_select 'h1', "Videos"

    get '/videos/new'
    assert_response :success
    assert_select 'form', 1

    upload_file = fixture_file_upload('test/fixtures/files/swe-eng.mp4', '
        video/mp4')

    post '/videos', params: { video: { title: 'SWE-ENG', file: upload_file }
         }, headers: { 'HTTP_REREFER': '/say/hello' }
    assert_response :redirect
    assert_equal Attachment.all.size, 3
    follow_redirect!
    assert_select 'a', 'Logout'

    get '/sessions/destroy', headers: { 'HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    assert_nil session[:user]
    follow_redirect!
    assert_select 'a', 'Register'
  end
end
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/gyakorlat$ RAILS_ENV='test' rails test test/
    integration/upload_video_test.rb
Running via Spring preloader in process 8967
Run options: --seed 27865

# Running:

.

Finished in 0.790869s, 1.2644 runs/s, 16.4376 assertions/s.
1 runs, 13 assertions, 0 failures, 0 errors, 0 skips
```

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 böngészőre, a többit a végrehajtó rendszer elintézi.

A rendszerteszteket explicit paranccsal kell létrehoznunk.

```
kovacsg@debian:~/gyakorlat/test/system> rails g system_test hello
Running via Spring preloader in process 9539
      invoke  test_unit
   identical    test/application_system_test_case.rb
      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.

```
class HellosTest < ApplicationSystemTestCase
  test 'login' do
    visit '/say/hello'
    assert_selector 'legend', text: "Login"

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

    assert_selector 'a', text: "Logout"

  end
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
Running via Spring preloader in process 10445
Run options: --seed 41151

# Running:

Capybara starting Puma...
* Version 4.3.6 , codename: Mysterious Traveller
* Min threads: 0, max threads: 4
* Listening on tcp://127.0.0.1:36291
.

Finished in 2.960970s, 0.3377 runs/s, 0.6755 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 teljesítménytesztteknek 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 teljesítménytesztteket akárcsak az integrációs teszeket explicit módon kell létrehoznunk:

```
kovacsg@debian:~/gyakorlat> rails g performance_test hello
Running via Spring preloader in process 10929
      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 "say hello" do
    get '/say/hello'
  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.