Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2020. május 5.

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`.

```
lecturer:
  name: Oraado Bela
  neptun: ORDBLA
  email: bela@mail.bme.hu
  salt: ordbla
  encrypted_password: <%= Digest::SHA1.hexdigest("titokordbla")
     %>
  role: 0

student:
  name: Kituno Ottokar
  neptun: QWERTY
  email: ottokar@mail.bme.hu
  salt: qwerty
  encrypted_password: <%= Digest::SHA1.hexdigest("titokqwerty")
     %>
  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
  title: Elso feladat
  description: Hany eves vagy?

two:
  number: 2
  title: Masodik feladat
  description: Tolts fel egy kepet!
```

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: student
  task: two
  reviewer: lecturer
```

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

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
== 20200303120832 CreateUsers: migrating
   =============================================
-- create_table(:users)
   -> 0.0137s
== 20200303120832 CreateUsers: migrated (0.0139s)
   ================================

== 20200317134522 CreateTasks: migrating
   =============================================
-- create_table(:tasks)
   -> 0.0084s
== 20200317134522 CreateTasks: migrated (0.0086s)
   ================================

== 20200403095521 AddSaltToUsers: migrating
   =============================================
-- add_column(:users, :salt, :string)
   -> 0.0020s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0056s
== 20200403095521 AddSaltToUsers: migrated (0.0080s)
   ================================

== 20200403102224 ChangeEncrpytedPasswordLengthInUsers: migrating
   ===========
-- change_column(:users, :encrypted_password, :string, {:limit
   =>50})
```

```
   -> 0.0032s
== 20200403102224 ChangeEncrpytedPasswordLengthInUsers: migrated
   (0.0034s) =====

== 20200403105841 CreateSolutions: migrating
   ===================================
-- create_table(:solutions)
   -> 0.0188s
== 20200403105841 CreateSolutions: migrated (0.0190s)
   ==========================

== 20200420061133 AddRoleToUsers: migrating
   ===================================
-- add_column(:users, :role, :integer, {:limit=>1, :null=>false,
   :default=>1})
   -> 0.0013s
== 20200420061133 AddRoleToUsers: migrated (0.0015s)
   ==========================

== 20200420070231 CreateAttachments: migrating
   ===================================
-- create_table(:attachments)
   -> 0.0152s
== 20200420070231 CreateAttachments: migrated (0.0154s)
   ==========================

== 20200420084525 AddReviewerToSolutions: migrating
   ===================================
-- add_column(:solutions, :reviewer_id, :integer)
   -> 0.0013s
== 20200420084525 AddReviewerToSolutions: migrated (0.0015s)
   ==========================
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# RAILS_ENV='test' rails db
MariaDB [gyakorlat_test]> select * from users;
+------------+-----------------+---------+-------------------+
| id         | name            | neptun | email              |
   encrypted_password                    | created_at         |
   updated_at          | salt   | role |
+------------+-----------------+---------+-------------------+

| 349131078 | Oraado Bela     | ORDBLA | bela@mail.bme.hu   |
   e0b08f3d1adeb9cd63fce3f60e17850e5009a71d | 2020-05-04 07:40:42.177291 |
   2020-05-04 07:40:42.177291 | ordbla |    0 |
```

```
| 925085493 | Kituno Ottokar | QWERTY | ottokar@mail.bme.hu | 6
    c10203ff48555c59d94a5aba57f1cd952b822cd | 2020-05-04 07:40:42.177291 |
    2020-05-04 07:40:42.177291 | qwerty |     1 |
+-----------+----------------+--------+---------------------+----------------------------------
2 rows in set (0.000 sec)

MariaDB [gyakorlat_test]> select * from tasks;
+-----------+--------+---------------+-------------------+---------------------
| id        | number | title         | description       | created_at
                  | updated_at                |
+-----------+--------+---------------+-------------------+---------------------
| 298486374 |      2 | Masodik feladat | Tolts fel egy kepet! | 2020-05-04
    07:40:42.174721 | 2020-05-04 07:40:42.174721 |
| 980190962 |      1 | Elso feladat    | Hany eves vagy?      | 2020-05-04
    07:40:42.174721 | 2020-05-04 07:40:42.174721 |
+-----------+--------+---------------+-------------------+---------------------
2 rows in set (0.000 sec)

MariaDB [gyakorlat_test]> select * from solutions;
+-----------+-----------+-----------+---------------------+---------------------
| id        | user_id   | task_id   | created_at                |
    updated_at                | reviewer_id |
+-----------+-----------+-----------+---------------------+---------------------
| 980190962 | 925085493 | 298486374 | 2020-05-04 07:40:42.169599 |
    2020-05-04 07:40:42.169599 |   349131078 |
+-----------+-----------+-----------+---------------------+---------------------
1 row in set (0.000 sec)

MariaDB [gyakorlat_test]> Bye
```

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
```

```
  # def test_the_truth
  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 address' do
    u = User.new name: 'Senki'
    assert !u.save, "Houston, we have a problem"
  end

  test 'cannot save user with existing email address' do
    u = User.new name: 'Senki', password: 'titok', email: users(:student).
        email
    assert !u.save, "Houston, we have a problem"
  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 test test/models/user_test.rb
Run options: --seed 24647

# Running:

....

Finished in 0.053133s, 75.2821 runs/s, 75.2821 assertions/s.
4 runs, 4 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat# rails test test/models/user_test.rb
    :19
Run options: --seed 14800

# Running:

.

Finished in 0.047971s, 20.8458 runs/s, 20.8458 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat# rails test test/models
Run options: --seed 10869

# Running:
```

```
....

Finished in 0.062122s, 64.3894 runs/s, 64.3894 assertions/s.
4 runs, 4 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):033:0> include Rails.application.routes.url_helpers
=> Object
irb(main):034:0> hello_path
=> "/say/hello"
irb(main):035:0> hello_url
Traceback (most recent call last):
        1: from (irb):35
ArgumentError (Missing host to link to! Please provide the :host
    parameter, set default_url_options[:host], or set :only_path
    to true)
irb(main):036:0> default_url_options[:host] = 'http://localhost
    :3000'
=> "http://localhost:3000"
irb(main):037:0> hello_url
=> "http://localhost:3000/say/hello"
irb(main):038:0> url_for controller: 'say', action: 'hello'
=> "http://localhost:3000/say/hello"
irb(main):039:0> url_for controller: 'sessions', action: 'create'
    , email: 'valaki@mail.bme.hu', password: 'titok'
=> "http://localhost:3000/sessions/create?email=valaki%40mail.bme
    .hu&password=titok"
irb(main):040:0> url_for controller: 'tasks', action: 'index',
    page: 2
=> "http://localhost:3000/tasks/page/2"
irb(main):041:0> url_for controller: 'tasks', action: 'show', id:
     2
=> "http://localhost:3000/tasks/2"
```

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 test test/controllers/say_controller_test.
    rb
Run options: --seed 10072

# Running:

.

Finished in 0.173147s, 5.7754 runs/s, 17.3263 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
    post '/sessions/create', params: { email: users(:student).email,
        password: 'titok' }, headers: { 'HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    assert_equal session[:user], users(:student).id
    follow_redirect!
    assert_select 'a', 'Profile'
  end

  test "invalid_login" do
    post '/sessions/create', params: { email: users(:student).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
    post '/sessions/create', params: { email: users(:student).email,
        password: 'titok' }, headers: { 'HTTP_REFERER': '/say/hello' }
    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
```

A tesztet e kontrollerteszt kiválasztásával futtatjuk:

```
kovacsg@debian:~/gyakorlat# rails test test/controllers/
    sessions_controller_test.rb
Run options: --seed 27537

# Running:

...

Finished in 0.224995s, 13.3336 runs/s, 44.4453 assertions/s.
```

```
3 runs, 10 assertions, 0 failures, 0 errors, 0 skips
```

A feladatok tesztjét scaffolddal generáltuk, módosítsuk, hogy alkalmas legyen a módosításaink ellenőrzésére. Az alapértelmezett tesztek csak a HTTP válaszok státusz kódját ellenőrzik, ezen túl nem megyünk. A futáshoz azonban szükséges, hogy a tesztadat megfelelő legyen, és a műveleteket csak egy bejelentkezett felhasználó hajthassa végre (a bejelentkezés végrehajtó HTTP műveletet átemeljük a login tesztből). Ezeket a `setup` törzsében intézhetjük el.

```
class TasksControllerTest < ActionDispatch::IntegrationTest
  setup do
    @task = tasks(:two)
    post '/sessions/create', params: { email: users(:student).
        email, password: 'titok'}, headers: { 'HTTP_REFERER': '/
        say/hello' }
  end
end
```

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 `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 feladatok listája link.

A feladatok linkre kattintva lekérdezzük a feladatko listáját az `index` nézettel. Az a feltételezésünk, hogy ott a táblázat törzsében annyi feladatot találunk, ahányat definiáltunk a tesztadatok között.

A feladatok listája nézeten az egyes feladatok melletti `Show` linkre kattintva a feladat szerkeszthető. A tesztesetben a második sorszámú feladatot választjuk ki. Az a feltételezésünk, hogy az oldal betöltődik, és ott lehetőség van fájl csatolására.

A feladat 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 SubmitSolutionTest < ActionDispatch::IntegrationTest
  # test "the truth" do
  #   assert true
  # end
  test "upload_file_to_task_as_solution" do
    get '/say/hello'
    assert_nil session[:user]
    assert_select 'a', 'Register'
    assert_select 'label', 'Email'
    assert_select 'label', 'Password'

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

    get '/tasks'
    assert_response :success
    assert_select 'table', 1
    assert_select 'tbody_tr', Task.all.size
```

```
    get "/tasks/#{tasks(:two).id}"
    assert_response :success
    assert_select 'label', 'Solution'
    assert_equal Attachment.all.size, 0

    upload_file = fixture_file_upload('test/fixtures/files/rails_hello.png',
        'image/png')
    post "/solutions/upload/#{users(:student).id}/#{tasks(:two).id}", params
        : { file: upload_file }, headers: { 'HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    assert_equal Attachment.all.size, 1
    assert File.exists? "public/data/#{Attachment.last.id}"
    follow_redirect!
    assert_select 'a', 'Logout'

    get '/sessions/destroy'
    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 test test/integration/
    submit_solution_test.rb
Run options: --seed 18611

# Running:

.

Finished in 0.287811s, 3.4745 runs/s, 69.4901 assertions/s.
1 runs, 20 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# 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.

```
class HellosTest < ApplicationSystemTestCase
  test "visiting the root page" do
    visit hello_path
    assert_selector "legend", text: 'Login'

    fill_in "Email", with: 'ottokar@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:system
Run options: --seed 48079

# Running:

Capybara starting Puma...
* Version 4.3.3 , codename: Mysterious Traveller
* Min threads: 0, max threads: 4
* Listening on tcp://127.0.0.1:41905
.

Finished in 2.535724s, 0.3944 runs/s, 0.7887 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
      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.