Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2021. május 4.

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: Kovacs Gabor
  email: kovacsg@tmit.bme.hu
  salt: titok
  encrypted_password: <%= Digest::SHA1.hexdigest('titoktitok') %>
```

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

A tesztadatbázisunkban több projektre lesz szükségünk, ezeket a `items.yml` fájlban definiáljuk. Itt is volt utólag változás a struktúrában, hozzáadtunk egy idegen kulcsot a felhasználók táblára. Az egyes projektek tulajdonosa egy felhasználó, amelyre egy felhasználó tesztadatának kulcsával hivatkozhatunk, legyen a tulajdonos a `me` kulcsú felhasználó.

```
one:
  name: Project 1
  description: Project 1
  user: me

two:
  name: Project 2
  description: Project 2
  user: me
```

A projektek fájlokból állnak. Itt nem módosítottuk utólag migrációval a struktúrát. A `one` azonosítójú projekthez rendelünk fájlokat.

```
one:
  name: File 1
  item: one
  version: '1.0'

two:
  name: File 2
  item: one
  version: '1.0'
```

Egy-egy fájlhoz csatolmány tartozhat, de egyelőre nem veszünk fel egy csatolmányt sem, töröljük ax `attachments.yml` fájl tartalmát.

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:create
Created database 'gyakorlat_test'
kovacsg@debian:~/gyakorlat/test/fixtures> RAILS_ENV='test' rails
    db:migrate
== 20210302122531 CreateUsers: migrating
    =============================================
-- create_table(:users)
   -> 0.0086s
== 20210302122531 CreateUsers: migrated (0.0090s)
    ====================================

== 20210302124006 CreateItems: migrating
    =============================================
-- create_table(:items)
   -> 0.0097s
== 20210302124006 CreateItems: migrated (0.0101s)
    ====================================

== 20210316121736 CreateResources: migrating
    ============================================
-- create_table(:resources)
   -> 0.0171s
== 20210316121736 CreateResources: migrated (0.0176s)
    ===========================

== 20210330102045 AddSaltToUsers: migrating
    ============================================
-- add_column(:users, :salt, :string)
   -> 0.0029s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0060s
== 20210330102045 AddSaltToUsers: migrated (0.0098s)
    ===========================

== 20210330113551 AddUserToItems: migrating
    ============================================
-- add_reference(:items, :user, {:null=>false, :foreign_key=>true
    , :default=>2})
   -> 0.0435s
== 20210330113551 AddUserToItems: migrated (0.0442s)
    ===========================
```

```
== 20210427110237 CreateAttachments: migrating
   ================================
-- create_table(:attachments)
   -> 0.0151s
== 20210427110237 CreateAttachments: migrated (0.0155s)
   =======================

== 20210427110330 AddResourceToAttachments: migrating
   ============================
-- add_reference(:attachments, :resource, {:null=>false, :
   foreign_key=>true})
   -> 0.0526s
== 20210427110330 AddResourceToAttachments: migrated (0.0528s)
   ===================

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]> show tables;
+-----------------------------+
| Tables_in_gyakorlat_test    |
+-----------------------------+
| ar_internal_metadata        |
| attachments                 |
| items                       |
| resources                   |
| schema_migrations           |
| users                       |
+-----------------------------+
6 rows in set (0.001 sec)

MariaDB [gyakorlat_test]> select * from users;
+-----------+--------------+-------------------+----------------------------------
| id        | name         | email             | encrypted_password
                         | created_at                | updated_at
                  | salt  |
+-----------+--------------+-------------------+----------------------------------

| 743609028 | Kovacs Gabor | kovacsg@tmit.bme.hu |
   fb05cdbb24874c68d2cbe870db0b025fa480e58a | 2021-05-04 10:27:43.053591 |
   2021-05-04 10:27:43.053591 | titok |
+-----------+--------------+-------------------+----------------------------------

1 row in set (0.000 sec)

MariaDB [gyakorlat_test]> select * from items;
+---------+---------+---------+-----------------------+----------------------------
```

```
| id         | name      | description | created_at                 |
    updated_at                 | user_id    |
+------------+-----------+-------------+----------------------------+
----------------------------+------------+

| 298486374 | Project 2 | Project 2   | 2021-05-04 10:27:43.042245 |
    2021-05-04 10:27:43.042245 | 743609028  |
| 980190962 | Project 1 | Project 1   | 2021-05-04 10:27:43.042245 |
    2021-05-04 10:27:43.042245 | 743609028  |
+------------+-----------+-------------+----------------------------+
----------------------------+------------+

2 rows in set (0.000 sec)

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

| id         | name     | item_id    | version | created_at                 |
    updated_at                 |
+------------+----------+------------+---------+----------------------------+
----------------------------+

| 298486374 | File 2 | 980190962 | 1.0     | 2021-05-04 10:27:43.051854 |
    2021-05-04 10:27:43.051854 |
| 980190962 | File 1 | 980190962 | 1.0     | 2021-05-04 10:27:43.051854 |
    2021-05-04 10:27:43.051854 |
+------------+----------+------------+---------+----------------------------+
----------------------------+

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
    assert !u.save, "Houston, we have a problem"
  end

  test "encrypted" do
    u = User.find users(:me).id
    assert_equal u.encrypted_password, Digest::SHA1.hexdigest('titoktitok'),
        "A titkositas nem megfelelo"
  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 15432
Run options: --seed 61149

# Running:

.....

Finished in 0.053725s, 93.0660 runs/s, 93.0660 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:4
Running via Spring preloader in process 15642
Run options: --seed 4407

# Running:

.

Finished in 0.048663s, 20.5495 runs/s, 20.5495 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    models/
Running via Spring preloader in process 15694
Run options: --seed 33282

# Running:
```

```
.....

Finished in 0.067189s, 74.4170 runs/s, 74.4170 assertions/s.
5 runs, 5 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.

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails c
Running via Spring preloader in process 15918
Loading test environment (Rails 6.1.3)
irb(main):001:0> include Rails.application.routes.url_helpers
=> Object
irb(main):002:0> hello_path
=> "/say/hello"
irb(main):003:0> hello_url
Traceback (most recent call last):
        1: from (irb):3
ArgumentError (Missing host to link to! Please provide the :host
    parameter, set default_url_options[:host], or set :only_path
    to true)
irb(main):004:0> default_url_options[:host] = 'localhost:3000'
=> "localhost:3000"
irb(main):005:0> hello_url
=> "http://localhost:3000/say/hello"
irb(main):006:0> resouce_path(1)
Traceback (most recent call last):
        1: from (irb):6
NoMethodError (undefined method `resouce_path' for main:Object)
Did you mean?  resource_path
               resources_path
irb(main):007:0> resource_path(1)
=> "/resources/1"
irb(main):008:0> edit_resource_path(1)
=> "/resources/1/edit"
irb(main):009:0> url_for controller: 'resources', action: 'edit',
    id: 1
=> "http://localhost:3000/resources/1/edit"
```

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 hello_url
    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 17205
Run options: --seed 39697

# Running:

.

Finished in 0.257567s, 3.8825 runs/s, 11.6474 assertions/s.
1 runs, 3 assertions, 0 failures, 0 errors, 0 skips
```

Módosítsuk a `SessionsControllerTest` tesztet A funkciót a `me` kulcshoz tartozó felhasználó tesztadatain végezzük el. Az első a bejelentkezést, a második a kijelentkezést teszteli, nevezzük át a teszteket ennek megfelelően! A bejelentkezés eseményt a `SessionsController` `create` metódusát használja, a kijelentezés esemény pedig a `destroy` metódust. A tesztben a `post` metódust használjuk, amelynek első és egyetlen kötelező paramétere a tesztelendő URL (Rails 5 előtt tesztelendő akció). A `params` kulcshoz rendelhetjük a HTTP kérés paramétereit, a `headers` kulcshoz pedig a kiküldendő HTTP kérés fejléc beállításait. A sikeres bejelentkezés tesztben a tesztadatokban szereplő felhasználónév, jelszó párost küldjük el, és azt feltételezzük, hogy visszairányítódunk a belépés előtti nézetre, a `:user` session paraméter értéke nem `nil`, 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 hello_path
    assert_response :success
    assert_select 'legend', "Login"
    assert_nil session[:user]

    post '/sessions/create', params: { email: users(:me).email,
      password: 'titok' }, headers: { 'HTTP_REFER': '/say/hello' }
```

```
    assert_response :redirect
    assert_equal session[:user], users(:me).id
    follow_redirect!
    assert_select 'a', 'Logout'

    get '/sessions/destroy', headers: { 'HTTP_REFER': '/say/hello' }
    assert_response :redirect
    assert_nil session[:user]
    follow_redirect!
    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
Running via Spring preloader in process 18821
Run options: --seed 39436

# Running:

...

Finished in 0.322239s, 9.3099 runs/s, 65.1690 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/test/integration> rails g
    integration_test upload_resource
Running via Spring preloader in process 19056
      invoke  test_unit
      create    test/integration/upload_resource_test.rb
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `upload_resource_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 projektek listáját,

- rákattintunk a táblázatban az egyik projekt mellett látható `Show` címkéjű linkre,
- a megjelenő listában rákattintunk az egyik erőforrás linkjére,
- kitöltjük feltöltjük a csatolmány a formban, 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 `Login` címkéjű `legend` HTML elem.

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 projektek linkre kattintva lekérdezzük a projektek listáját az `index` nézettel, amely táblázatosan jelenik meg. Az a feltételezésünk, hogy a válasz sikeres, és a táblázat törzsében annyi sor lesz, ahány projekt van az adatbázisban.

Egy projektet, konkrétan a `one` kulcshoz tartozót megnyitva sikeres választ, és a nézeten lévő listában annyi `li` elemet várunk, ahány fájl tartozik ebbe a projektbe.

Az egyik fájl linkjére, konkrétan a `one` kulcshoz tartozóéra kattintva sikeres választ, és a nézeten egy form megjelenését várjuk.

Az erőforrás adatlapja nézeten található egy form, amelyen kereszül egy `attachment` nevű paramétert juttathatunk el az `update` akciónak HTTP POST üzenettel [1]. 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, átirányítás történik, és megjelenik egy letöltés paragrafus a nézeten.

Ehhez előbb módosítjuk az `app/views/resources/show.html.erb` fájt, hogy az csak feltételesen jelenjék meg függően attól, hogy van-e már csatolmány ehhez az erőforráshoz:

```
<% unless @resource.attachment.nil? %>
  <p>
    <strong>Download:</strong>
    <%= link_to "Download", download_resource_path(@resource.id)
        %>
  </p>
```

[1] A gyakorlaton elírtuk az attachment kulcsot, és ezért ennél a feltételnél hibára futottunk.

```
<% end %>
```

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 UploadResourceTest < ActionDispatch::IntegrationTest
  def setup
    @resource = resources(:one)
  end

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

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

    get items_path
    assert_response :success
    assert_select 'tbody tr', Item.count

    get item_path(items(:one).id)
    assert_response :success
    assert_select 'li', Resource.where(item_id: items(:one).id).size

    get resource_path(resources(:one).id)
    assert_response :success
    assert_select 'form', 1

    upload_file = fixture_file_upload('test/fixtures/files/1.txt', 'text/
        plain')
    post upload_resource_path(resources(:one).id), params: { attachment:
        upload_file },
      headers: { 'HTTP_REFERER': "/resources/#{resources(:one).id}" }
    assert_response :redirect
    assert_equal Attachment.count, 1
    follow_redirect!
    assert_select 'strong', 'Download:'

    get '/sessions/destroy', headers: { 'HTTP_REFER': '/say/hello' }
    assert_response :redirect
    assert_nil session[:user]
    follow_redirect!
    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/upload_resource_test.rb
Running via Spring preloader in process 6808
Run options: --seed 45940

# Running:

.

Finished in 0.339000s, 2.9499 runs/s, 53.0974 assertions/s.
1 runs, 18 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
Running via Spring preloader in process 24874
      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 index" do
    visit '/say/hello'
    assert_selector 'legend', text: "Login"

    fill_in "Email", with: 'kovacsg@tmit.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 végrehajtását.

A teszt elsőre elbukott, mert az `app/view/layouts/_guest.html.erb` fájlban hibát vétettünk, `text_field`-et írtunk `text_field_tag` helyett, és így a beviteli mező azonosítója nem egyezett meg a hozzá tartozó címkében megadottal. Az előbbi függvény csak `form_for` függvényen belül használható.

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    system/hellos_test.rb
Running via Spring preloader in process 25542
Run options: --seed 32974

# Running:

Capybara starting Puma...
* Version 5.2.1 , codename: Fettisdagsbulle
* Min threads: 0, max threads: 4
* Listening on http://127.0.0.1:37297
[Screenshot Image]: /home/kovacsg/gyakorlat/tmp/screenshots/
    failures_test_visiting_the_index.png
E

Error:
HellosTest#test_visiting_the_index:
Capybara::ElementNotFound: Unable to find field "Email" that is
    not disabled
    test/system/hellos_test.rb:8:in `block in <class:HellosTest>'


rails test test/system/hellos_test.rb:4



Finished in 6.543447s, 0.1528 runs/s, 0.1528 assertions/s.
1 runs, 1 assertions, 0 failures, 1 errors, 0 skips
```

A hiba javítása után a teszt sikeresen lefutott.

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    system/hellos_test.rb
Running via Spring preloader in process 2441
Run options: --seed 35286

# Running:

Capybara starting Puma...
* Version 5.2.1 , codename: Fettisdagsbulle
* Min threads: 0, max threads: 4
* Listening on http://127.0.0.1:36653
.
```

```
Finished in 2.471363s, 0.4046 runs/s, 0.8093 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
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.