Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2021. november 23.

1. Javascrtip térkép

Az előző gyakorlat végén nem maradt idő egy térkép hozzáadására, most pótoljuk. A Google térképszolgáltatása speciális szolgáltatási feltételek elfogadásához kötött, ezért egy OpenStreetMaps API-t használunk, a Leaflet-et. Telepítsük!

```
kovacsg@debian:~/gyakorlat> bin/yarn add leaflet
yarn add v1.19.2
[1/4] Resolving packages...
[2/4] Fetching packages...
info fsevents@2.3.2: The platform "linux" is incompatible with
   this module.
info "fsevents@2.3.2" is an optional dependency and failed
   compatibility check. Excluding it from installation.
info fsevents@1.2.13: The platform "linux" is incompatible with
   this module.
info "fsevents@1.2.13" is an optional dependency and failed
   compatibility check. Excluding it from installation.
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Saved lockfile.
success Saved 1 new dependency.
info Direct dependencies
- leaflet@1.7.1
info All dependencies
- leaflet@1.7.1
Done in 3.85s.
```

A modul a `node_modules/leaflet` könyvárban jött létre, ahol a `dist` alkönytár tartalmazza a JavaScript és CSS fájlokat.

A térképet a méhlegelők kontroller `show` nézetében használjuk, ezért a `app/assets/stylesheets/beepastures.scss` fájljában hivatkozzuk meg a térkép API stíluslapját.

```
@import 'leaflet/dist/leaflet.css'
```

A JavaScript forrásokat az `app/javascript/packs` könyvtárban helyezzük el, mondjuk `map` néven. Ebben a fájlban kis módosítással a Leaflet API weboldalán található kódrészletet helyezzük el. A szélességi és hosszúsági koordináták legyenek 47, illetve 19, amelyek valahol Magyarországon vannak, konkrétan Bácsalmás mellett.

```
import 'leaflet';

var map = L.map('map').setView([47, 19], 13);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
    {
    attribution: '&copy; <a href="https://www.openstreetmap.org/
        copyright">OpenStreetMap</a> contributors', subdomains: [
        'a', 'b', 'c']
}).addTo(map);

L.marker([47, 19]).addTo(map);
```

A méhlegelők `show` nézetéhez hozzáadunk `map` `id`-vel egy HTML elemet, ahova a JavaScript forrás elhelyezi a térképet, és meghivatkozzuk magát a forrást.

```
<div id="map"></div>
<%= javascript_pack_tag 'map' %>
```

Az `application.css`-ben az elem magasság attribútumát 200 pixel magasra állítjuk be.

```
<div id="map"></div>
#map {
  height: 200px;
}
```

2. Tesztelés Railsben

Laborunk fő 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 3001-es porton, 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 -p3001
```

3. Tesztadatok felvétele, betöltése

A tesztelés előkészítésére először tesztadatokat definiálunk, amelyekre a teszteseteinkben hivatkozni fogunk. Ezeket a Rails alkalmazásunk `test/fixtures` könyvtárában helyezzük el. A teszt szkriptek 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:
  username: Valaki
  encrypted_password: <%= Digest::SHA1.hexdigest 'titokhello' %>
  email: valaki@mail.bme.hu
  salt: hello
```

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

A tesztadatbázisunkban szükségünk lesz egy méhlegelőre, ennek az adatait `beepastures.yml` fájlban definiáljuk. Itt nem volt változás a struktúrában, csak az adatokat módosítjuk. Használjuk fel az előbbi térképadatokat.

```
racalmas:
  city: Racalmas
  address: Tehenlegelo u. 1
```

```
  zip: 3045
  latitude: 47
  longitude: 19
```

A megfigyelések a `sightings.yml` fájlba kerülnek. A koordináta a semmi közepén van, így alkalmas UFO-leszállóhely, amelyet meg is figyeltünk, illetve kiválóan alkalmas stadionépítésre. Az egyes megfigyelések tevője egy felhasználó, amelyre egy felhasználó tesztadatának kulcsával hivatkozhatunk, legyen a tulajdonos a `me` kulcsú felhasználó, és a megfigyeléseket a fent definiált méhlegelőn tette, amelyre a méhlegelő kulcsával hivatkozunk.

```
ufo:
  sighting: crop_circle
  beepasture: racalmas
  user: me

stadion:
  sighting: stadium
  beepasture: racalmas
  user: me
```

Csatolmány egyelőre nincsenek, ezért az `attachments.yml` fájlból töröljük az adatokat.

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
== 20210928105344 CreateUsers: migrating
   ==========================================
-- create_table(:users)
   -> 0.0058s
== 20210928105344 CreateUsers: migrated (0.0061s)
   =================================

== 20210928112323 CreateBeepastures: migrating
   ============================
-- create_table(:beepastures)
   -> 0.0046s
== 20210928112323 CreateBeepastures: migrated (0.0049s)
   ===================
```

```
== 20210928113437 CreateSightings: migrating
   ================================
-- create_table(:sightings)
   -> 0.0074s
== 20210928113437 CreateSightings: migrated (0.0077s)
   =======================

== 20211026102543 AddSaltToUsers: migrating
   ================================
-- add_column(:users, :salt, :string)
   -> 0.0023s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0025s
== 20211026102543 AddSaltToUsers: migrated (0.0053s)
   ===========================

== 20211026113918 CreateAttachments: migrating
   ================================
-- create_table(:attachments)
   -> 0.0080s
== 20211026113918 CreateAttachments: migrated (0.0083s)
   ===========================

== 20211109101257 ChangeSightingToInteger: migrating
   ================================
-- change_column(:sightings, :sighting, :integer)
   -> 0.0162s
== 20211109101257 ChangeSightingToInteger: migrated (0.0165s)
   ===========================

== 20211109104707 AddUserIdToSightings: migrating
   ================================
-- add_reference(:sightings, :user, {:null=>false, :foreign_key=>
   true})
   -> 0.0325s
== 20211109104707 AddUserIdToSightings: migrated (0.0327s)
   ===========================

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
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 78
Server version: 10.3.29-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]> select * from users;
+-----------+----------+----------------------------------+-----------------
| id        | username | encrypted_password               | email
                | created_at                 | updated_at                  |
     salt   |
+-----------+----------+----------------------------------+-----------------

| 743609028 | Valaki   | 76810c204dc2c0442b7f3ebbeba6f5dce98a231c |
    valaki@mail.bme.hu | 2021-11-23 11:33:26.710130 | 2021-11-23
    11:33:26.710130 | hello |
+-----------+----------+----------------------------------+-----------------

1 row in set (0.000 sec)

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

| id        | city    | address         | zip  | latitude | longitude |
    created_at                 | updated_at                 |
+-----------+---------+-----------+-----+----------+-----------+-----------

| 145835892 | Racalmas | Tehenlegelo u. 1 | 3045 |       47 |        19 |
    2021-11-23 11:33:26.709124 | 2021-11-23 11:33:26.709124 |
+-----------+---------+-----------+-----+----------+-----------+-----------

1 row in set (0.001 sec)

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

| id        | sighting | beepasture_id | created_at                 |
    updated_at                 | user_id   |
+-----------+----------+--------------+----------------------------------+-----------

|  64078658 |        2 |     145835892 | 2021-11-23 11:33:26.707486 |
    2021-11-23 11:33:26.707486 | 743609028 |
| 707531782 |        4 |     145835892 | 2021-11-23 11:33:26.707486 |
    2021-11-23 11:33:26.707486 | 743609028 |
+-----------+----------+--------------+----------------------------------+-----------

2 rows in set (0.001 sec)

MariaDB [gyakorlat_test]> Bye
```

4. 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 :username, 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_email_address" do
    user = User.new username: 'Hello'
    assert !user.save, "Houston,_we_have_a_problem"
  end

  test "cannot_save_user_without_username" do
    user = User.new email: 'hello@mail.bme.hu'
    assert !user.save, "Houston,_we_have_a_problem"
  end

  test "encyption" do
    user = users(:me)
    assert_equal user.encrypted_password, Digest::SHA1.hexdigest('titok'+
        user.salt)
  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 19429
Run options: --seed 45570

# Running:
```

```
....

Finished in 0.067036s, 59.6695 runs/s, 59.6695 assertions/s.
4 runs, 4 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 21232
Run options: --seed 12117

# Running:

.

Finished in 0.047229s, 21.1734 runs/s, 21.1734 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

5. 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):007:0> include Rails.application.routes.url_helpers
=> Object
irb(main):008:0> hello_path
=> "/say/hello"
irb(main):009:0> hello_url
Traceback (most recent call last):
        1: from (irb):9
ArgumentError (Missing host to link to! Please provide the :host
    parameter, set default_url_options[:host], or set :only_path
    to true)
irb(main):010:0> default_url_options[:host] = 'http://localhost
    :3001'
=> "http://localhost:3001"
irb(main):011:0> hello_url
=> "http://localhost:3001/say/hello"
irb(main):012:0> url_for controller: 'say', action: 'hello'
=> "http://localhost:3001/say/hello"
irb(main):013:0> url_for controller: 'beepastures', action: 'show
    ', id: 1
=> "http://localhost:3001/beepastures/1"
```

6. 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.

```
class SayControllerTest < ActionDispatch::IntegrationTest
  test "should get hello" do
    get hello_url
    assert_response :success
  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 21333
Run options: --seed 45788

# Running:

.

Finished in 0.328079s, 3.0480 runs/s, 3.0480 assertions/s.
1 runs, 1 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 kijelentkezés műveletre 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
    user = users(:me)
    post login_path, params: { email: user.email, password: 'titok' },
        headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    assert_equal session[:user], user.id
    follow_redirect!
    assert_select 'a', 'Logout'
  end

  test "unsuccessful_login" do
    user = users(:me)
    post login_path, params: { email: user.email, password: 'titok2' },
        headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    assert_nil session[:user]
    follow_redirect!
    assert_select 'a', 'Register'
  end

  test "logout" do
    user = users(:me)
    post login_path, params: { email: user.email, password: 'titok' },
        headers: { 'HTTP_REFERER': hello_path }

    follow_redirect!
    assert_select 'a', 'Logout'

    get logout_path
    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 21962
Run options: --seed 57596

# Running:

...

Finished in 0.416518s, 7.2026 runs/s, 24.0086 assertions/s.
3 runs, 10 assertions, 0 failures, 0 errors, 0 skips
```

7. Integrációs teszt

A tesztek harmadik típusa az integrációs teszt, amellyel egy új megfigyelés feltöltésének folyamatát ellenőrzük, ugyanis az egyik közeli tóban dokumentáltal felfeldeztük a loch ness-i szörnyet.

```
kovacsg@debian:~/gyakorlat/test/integration> rails g
    integration_test i_saw_nessie
Running via Spring preloader in process 22281
      invoke  test_unit
      create    test/integration/i_saw_nessie_test.rb
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `i_saw_nessie_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 méhlegelők listáját,
- rákattintunk a megfigyeléshez köthető méhlegelő linkjére,
- kitöltjük a megfigyelés formot, és feltöltjük a fotót, és
- végül 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,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 elérhető a kijelentkezés link.

A méhlegelők linkre kattintva lekérdezzük a méhlegelők listáját az `index` nézettel. Az a feltételezésünk, hogy ott lesz egy a tesztadatok között szereplő méhlegelő azonosítójával egy link annak `show` nézetére.

A méhlegelő példány nézetén az a feltételezésünk, hogy az oldal betöltődik, és ott van egy `Sightings` szövegelemmel rendelkező `h3` címsor.

A 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 ISawNessieTest < ActionDispatch::IntegrationTest
  test "I saw Nessie" do
    u = users(:me)
    b = beepastures(:racalmas)

    get hello_path
    assert_response :success
    assert_select 'legend', 'Login'

    post login_path, params: { email: u.email, password: 'titok' }, headers:
        { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    follow_redirect!
    assert_select 'a', 'Logout'
    assert_equal session[:user], u.id

    get beepastures_path
    assert_response :success
    assert_select 'a[href="'+beepasture_path(b.id)+'"]', 1

    get beepasture_path(b.id)
    assert_response :success
    assert_select 'h3', 'Sightings'

    upload_file = fixture_file_upload('test/fixtures/files/1', 'image/png')
    post "/attachments/#{b.id}/Sighting/upload", params: { sighting: 'nessie
        ', file: upload_file }, headers: { 'HTTP_REFERER': hello_path }
    follow_redirect!
    assert_equal Attachment.all.size, 1
    #assert Attachment.last.sighting.nessie?
    assert File.exists? "public/data/#{Attachment.last.id}"
    assert_select 'a', 'Logout'

    get logout_path
```

```
    assert_response :redirect
    follow_redirect!
    assert_nil session[:user]
    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/i_saw_nessie_test.rb
Running via Spring preloader in process 25888
Run options: --seed 27814

# Running:

.

Finished in 0.577799s, 1.7307 runs/s, 25.9606 assertions/s.
1 runs, 15 assertions, 0 failures, 0 errors, 0 skips
```

8. 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 26085
      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 "logging␣un" do
    visit '/say/hello'
    assert_selector 'legend', text: 'Login'

    fill_in "Email", with: 'valaki@mail.bme.hu'
    fill_in "Password", with: 'titok'
```

```
    click_on 'Login'

    assert_select 'a', text: 'Logout'
  end
end
```

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

```
kovacsg@debian:~/gyakorlat/test/system> RAILS_ENV='test' rails
    test:system
Run options: --seed 13057

# Running:

Capybara starting Puma...
* Version 5.5.0 , codename: Zawgyi
* Min threads: 0, max threads: 4
* Listening on http://127.0.0.1:41637


rails test test/system/hellos_test.rb:4



Finished in 5.502970s, 0.9086 runs/s, 0.3634 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

9. 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.