Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2023. május 23.

1. 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 3000-es porton, majd a webfelületen ellenőrizzük, hogy be tudunk-e lépni a tesztadatok között megadott email címet és jelszót megadva.

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

2. Tesztadatok felvétele, betöltése

A tesztelés előkészítésére először tesztadatokat definiálunk, amelyekre a teszteseteinkben hivatkozni fogunk. Ezeket a Rails alkalmazásunk `test/fixtures` könyvtárában helyezzük el. A tesztszkriptek 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, amelyeket 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`. Az `id` és az időpecsét attribútumokat nem kell definiálnunk, azok automatikusan töltődnek, az `id` random értékkel, az időpecsét pedig a pillanatnyi idővel.

```
valaki:
  username: valaki
  email: valaki@mail.bme.hu
  encrypted_password: <%= Digest::SHA1.hexdigest('titok'+'salt')
     %>
  salt: salt

senki:
  username: senki
  email: senki@mail.bme.hu
  encrypted_password: <%= Digest::SHA1.hexdigest('titok'+'salt')
     %>
  salt: salt
```

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

Először a felhasználói adatlapok tesztadait definiáljuk, amely a `people.yml`) fájlban található. Utólag adtuk hozzá a felhasználókra való hivatkozást, azzal bővítjük a listát. A hivatkozást a YAML fájlokban lévő kulcsokkal tesszük meg. A nemre vonatkozó enumhoz az enum hashe kulcsainak sztring reprezentációját rendeljük.

```
valaki:
  name: Vala Ki
  birthdate: 2023-03-28
  gender: male
  into: Hello
  user: valaki

senki:
  name: Shen Ki
  birthdate: 2023-03-28
  gender: female
  into: Hello
  user: senki
```

A kommentek a `comments.yml` fájlba kerülnek. A modellben két idegen kulcs is van, az egyik a felhasználóra, a másik a felhasználói adatlapra hivatkozik. Az első a kommentet tevő felhasználó, a második az az adatlap, ahol a komment van.

```
one:
  user: valaki
  comment: Hello
  person: senki

two:
  user: senki
  comment: Hello
  person: valaki
```

Csatolmányokat egyelőre nem veszünk fel, az `attachmets.yml` fájlból eltávolítjuk a bejegyzéseket.

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:~/randi> RAILS_ENV='test' rails db:drop
Dropped database 'randi_test'
kovacsg@debian:~/randi> RAILS_ENV='test' rails db:create
Created database 'randi_test'
kovacsg@debian:~/randi> RAILS_ENV='test' rails db:migrate
== 20230328112144 CreateUsers: migrating
   ==========================================
-- create_table(:users)
   -> 0.0094s
== 20230328112144 CreateUsers: migrated (0.0096s)
   =================================

== 20230328114143 CreatePeople: migrating
   ==========================================
-- create_table(:people)
   -> 0.0099s
== 20230328114143 CreatePeople: migrated (0.0102s)
   =================================

== 20230418113023 CreateComments: migrating
   ==========================================
-- create_table(:comments)
   -> 0.0099s
== 20230418113023 CreateComments: migrated (0.0101s)
   =================================
```

```
== 20230502102140 AddSaltToUsers: migrating
   ==========================================
-- add_column(:users, :salt, :string)
   -> 0.0053s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0044s
== 20230502102140 AddSaltToUsers: migrated (0.0101s)
   =================================

== 20230502113542 AddUserToPeople: migrating
   ==========================================
-- add_reference(:people, :user, {:null=>false, :foreign_key=>
   true})
   -> 0.0295s
== 20230502113542 AddUserToPeople: migrated (0.0295s)
   =================================

== 20230502114444 AddPersonToComments: migrating
   ==========================================
-- add_reference(:comments, :person, {:null=>false, :foreign_key
   =>true})
   -> 0.0285s
== 20230502114444 AddPersonToComments: migrated (0.0286s)
   =================================

== 20230509101807 CreateJoinTableUsersFriends: migrating
   ==========================================
-- create_join_table(:friends, :users)
   -> 0.0082s
== 20230509101807 CreateJoinTableUsersFriends: migrated (0.0085s)
   =================================

== 20230509105829 CreateAttachments: migrating
   ==========================================
-- create_table(:attachments)
   -> 0.0097s
== 20230509105829 CreateAttachments: migrated (0.0099s)
   =================================

kovacsg@debian:~/randi> RAILS_ENV='test' rails db
MariaDB [randi_test]> show tables;
+-----------------------+
| Tables_in_randi_test  |
+-----------------------+
| ar_internal_metadata  |
| attachments           |
| comments              |
| friends_users         |
| people                |
```

```
| schema_migrations      |
| users                  |
+------------------------+
7 rows in set (0.000 sec)

MariaDB [randi_test]> select * from users;
Empty set (0.000 sec)

MariaDB [randi_test]> ^DBye
kovacsg@debian:~/randi> 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. A felhasználó adatlap enumja feloldódott az igaz, hamis értékeket jelentő 1, 0 értékekre.

Nézzük először meg a tesztadatbázisunkat:

```
kovacsg@debian:~/randi> RAILS_ENV='test' rails db
MariaDB [randi_test]> select * from users;
+----------+----------+-------------------+------------------------------------------------------------+
| id       | username | email             | encrypted_password
                       | created_at                  | updated_at
                  | salt |
+----------+----------+-------------------+------------------------------------------------------------+

| 40473535 | valaki   | valaki@mail.bme.hu | 2
    c4e6eae823622b73f22c6f26d4293884e7ceb8c | 2023-05-23 10:26:59.768496 |
    2023-05-23 10:26:59.768496 | salt |
| 75949963 | senki    | senki@mail.bme.hu  | 2
    c4e6eae823622b73f22c6f26d4293884e7ceb8c | 2023-05-23 10:26:59.768496 |
    2023-05-23 10:26:59.768496 | salt |
+----------+----------+-------------------+------------------------------------------------------------+

2 rows in set (0.000 sec)

MariaDB [randi_test]> select * from people;
+----------+---------+-----------+--------+-------+---------------------------+
| id       | name    | birthdate | gender | into  | created_at
                  | updated_at                  | user_id  |
+----------+---------+-----------+--------+-------+---------------------------+

| 40473535 | Vala Ki | 2023-03-28 |      0 | Hello | 2023-05-23
    10:26:59.767251 | 2023-05-23 10:26:59.767251 | 40473535 |
| 75949963 | Shen Ki | 2023-03-28 |      1 | Hello | 2023-05-23
    10:26:59.767251 | 2023-05-23 10:26:59.767251 | 75949963 |
+----------+---------+-----------+--------+-------+---------------------------+

2 rows in set (0.000 sec)

MariaDB [randi_test]> select * from comments;
+----------+---------+---------+-----------------------------+-----------------------------+
| id       | user_id | comment | created_at                  | updated_at
                  | person_id |
+----------+---------+---------+-----------------------------+-----------------------------+
```

```
| 298486374 | 75949963 | Hello    | 2023-05-23 10:26:59.764407 | 2023-05-23
    10:26:59.764407 |  40473535 |
| 980190962 | 40473535 | Hello    | 2023-05-23 10:26:59.764407 | 2023-05-23
    10:26:59.764407 |  75949963 |
+-----------+----------+----------+----------------------------+----------------------------+

2 rows in set (0.002 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 :username, presence: true
  validates :email, { presence: true, uniqueness: true }
```

validációinak megtörténtét ellenőrzik. Először létrehozunk egy új felhasználót a név attribútumát inicializálatlanul hagyva, majd megpróbáljuk eltárolni az adatbázisba. A feltételezésünk az, hogy a mentésnek nem szabad sikerülnie. Az email attribútum validációjának tesztelésekor hasonlóan járunk el. A jelenlétet ellenőrző tesztesetben az email attribútumot hagyjuk inicializálatlanul, a duplikátumot ellenőrző tesztesetben pedig a tesztadatok közül veszünk egy email címet, és azzal próbáljuk menteni az objektumot.

```
class UserTest < ActiveSupport::TestCase
  # def test_the_truth
  test "the truth" do
    assert true
  end

  test "cannot save user without username" 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 username: 'valaki'
    assert !u.save, "Houston, we have a problem"
  end

  test "cannot save user with existing email address" do
    u = User.new username: users(:valaki).username, email: users(:valaki).
        email
    assert !u.save
  end

#  test "encryption" do
```

```
#     assert_equal User.encrypt('titok', users(:valaki).salt), Digest::SHA1.
    hexdigest("titokvalaki")
#   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:~/randi> rails test test/models/user_test.rb:5
Run options: --seed 55749

# Running:

.

Finished in 0.097996s, 10.2045 runs/s, 10.2045 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/randi> rails test test/models/user_test.rb
Run options: --seed 5577

# Running:

....

Finished in 0.111069s, 36.0137 runs/s, 36.0137 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.

```
kovacsg@debian:~/randi/test> RAILS_ENV='test' rails c
Loading test environment (Rails 7.0.4.3)
irb(main):001:0>
irb(main):002:0> include Rails.application.routes.url_helpers
=> Object
irb(main):003:0> hello_path
=> "/say/hello"
irb(main):005:0> hello_url
Traceback (most recent call last):
```

```
/var/lib/gems/2.7.0/gems/actionpack-7.0.4.3/lib/action_dispatch/
    http/url.rb:64:in 'full_url_for': Missing host to link to!
    Please provide the :host parameter, set default_url_options[:
    host], or set :only_path to true (ArgumentError)
irb(main):006:0> default_url_options[:host] = 'localhost:3000'
=> "localhost:3000"
irb(main):007:0> hello_url
=> "http://localhost:3000/say/hello"
irb(main):009:0> edit_profile_path(1)
=> "/users/edit/1"
irb(main):010:0> url_for controller: 'say', action: 'hello'
=> "http://localhost:3000/say/hello"
```

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.

Módosítsuk a `SessionsControllerTest` tesztet A funkciót a `valaki` kulcshoz tartozó felhasználó tesztadatain végezzük el. Három tesztesetet írunk. Az első a bejelentkezést, a második a harmadik a sikertelen, a harmadik 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 `Login` címkéjű form. 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 `Login` értékű LEGEND 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 url_for(controller: 'say', action: 'hello')
    assert_nil session[:user]
    assert_response :success
    assert_select 'legend', "Login"

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

  test "invalid login" do
    get url_for(controller: 'say', action: 'hello')
    assert_nil session[:user]
    assert_response :success
    assert_select 'legend', "Login"

    post url_for(controller: 'sessions', action: 'create'), params: { email:
        users(:valaki).email, password: 'titok2' }, headers: { '
      HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    follow_redirect!
    assert_nil session[:user]
    assert_select "legend", "Login"
    # post login_path
  end

  test "logout" do
    get url_for(controller: 'say', action: 'hello')
    assert_nil session[:user]
    assert_response :success
    assert_select 'legend', "Login"

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

    get logout_path
    assert_nil session[:user]
```

```
    assert_response :redirect
    follow_redirect!
    assert_select "legend", "Login"
  end
end
```

A tesztet e kontrollerteszt kiválasztásával futtatjuk:

```
kovacsg@debian:~/randi> RAILS_ENV='test' rails test test/
    controllers/sessions_controller_test.rb
Run options: --seed 59317

# Running:

...

Finished in 0.300138s, 9.9954 runs/s, 69.9679 assertions/s.
3 runs, 21 assertions, 0 failures, 0 errors, 0 skips
```

6. Integrációs teszt

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

```
kovacsg@debian:~/randi/test/integration> rails g integration_test
    upload_avatar
      invoke  test_unit
      create    test/integration/upload_avatar_test.rb
```

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

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

- betöltjük a bejelentkező képernyőt, ahol a bejelentkező form van,
- kitöltjük a bejelentkezési formot, és rákattintunk bejelentkezés gombra,
- betöltjük a felhasználói profil szerkesztése oldalt,
- kitöltjük az avatar formot, és feltöltjük az avatart, és
- végül kilépünk.

Az első lépésben egy be nem jelentkezett felhasználó betölti a bejelentkező 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 `Profile` címkéjű link.

A profil szerkesztése linkre kattintva az a feltételezésünk, hogy az oldalon látható a feltöltendő kép címke.

A formon 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 csatolmányok metaadatainak száma eggyel nő a feltöltés előtti állapothoz képest, és a fájl megjelenik az elvárt névvel a fájlrendszeren.

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

```
class UploadAvatarTest < ActionDispatch::IntegrationTest
  def setup
    @valaki = users(:valaki)
  end

  test "logout" do
    get url_for(controller: 'say', action: 'hello')
    assert_nil session[:user]
    assert_response :success
    assert_select 'legend', "Login"

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

    get edit_person_path(users(:valaki).person.id)
    assert_response :success
    assert_select "label", "Uploaded image"

    assert_nil users(:valaki).person.attachment
    upload_file = fixture_file_upload('test/fixtures/files/rails_form.png',
    'image/png')
    patch person_path(users(:valaki).person.id), params: { person: {
    uploaded_image: upload_file} }, headers: {'HTTP_REFERER': person_path(
    users(:valaki).pe
    assert_response :redirect
    assert_equal Attachment.all.size, 1
    assert File.exists?("public/data/"+users(:valaki).person.id.to_s
    follow_redirect!
    assert_select 'a', 'Logout'

    get logout_path
    assert_nil session[:user]
```

```
    assert_response :redirect
    follow_redirect!
    assert_select "legend", "Login"
  end
end
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/randi> RAILS_ENV='test' rails test test/
    integration/upload_avatar_test.rb
Run options: --seed 39586

# Running:

.

Finished in 0.307582s, 3.2512 runs/s, 52.0186 assertions/s.
1 runs, 16 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:~/pluto> rails g system_test hello
      invoke  test_unit
      create    test/system/hellos_test.rb
```

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

```
class HellosTest < ApplicationSystemTestCase
  setup do
    @person = people(:valaki)
  end

  test "login" do
    visit hello_path # '/say/hello'
    assert_selector "legend", text: "Login"

    fill_in "Email", with: @person.user.email
    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. A gyakorlaton a Chrome böngészőnk valamely bővítmény hibájából elszállt, ezért le kellett cserélnünk Firefoxra, amelyet a `test/application_system_test_case.rb` fájlban teszünk meg

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

A módosított böngészővel már fut a rendszerteszt.

```
kovacsg@debian:~/randi$ RAILS_ENV='test' rails test test/system/
    people_test.rb
Run options: --seed 50740

# Running:

2023-05-23 13:37:47 WARN Selenium [:logger_info] Details on how
    to use and modify Selenium logger:
  https://selenium.dev/documentation/webdriver/troubleshooting/
      logging#ruby

2023-05-23 13:37:47 WARN Selenium [DEPRECATION] [:capabilities]
    The :capabilities parameter for Selenium::WebDriver::Firefox
    ::Driver is deprecated. Use :options argument with an
    instance of Selenium::WebDriver::Firefox::Driver instead.
Capybara starting Puma...
* Version 5.6.5 , codename: Birdie's Version
* Min threads: 0, max threads: 4
* Listening on http://127.0.0.1:44299
.

Finished in 6.943262s, 0.1440 runs/s, 0.2880 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:~/pluto 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 "say hello" do
    get '/home/login'
  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.