Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2022. november 29.

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 3001-es porton, majd a webfelületen ellenőrizzük, hogy be tudunk-e lépni a tesztadatok között megadott email címet és jelszót megadva.

```
kovacsg@debian:~/pluto> RAILS_ENV='test' rails s -p 3001
```

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, 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`. Az `id` és az időpecsét attribútumokat nem kell definiálnunk, azok automatikusan töltődnek, az `id` random értékkel, az időpecsét pedig a pillanatnyi idővel.

```
valaki:
  name: Vala Ki
  email: valaki@mail.bme.hu
  encrypted_password: <%= Digest::SHA1.hexdigest 'titokvalaki' %>
  pluto: valaki
  salt: valaki

shenki:
  name: Shen Ki
  email: shenki@mail.bme.hu
  encrypted_password: <%= Digest::SHA1.hexdigest 'titokshenki' %>
  pluto: shenki
  salt: shenki
```

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

Először a szemeszterek tesztadait definiáljuk. Utólag két új attribútumot adtunk a modellhez, azzal bővítjük a listát.

```
spring:
  name: 2022/23 spring
  year: 2023
  season: spring
  begins: 2023-02-27
  ends: 2023-07-10

fall:
  name: 2022/23 fall
  year: 2022
  season: fall
  begins: 2023-09-05
  ends: 2023-01-27
```

A tárgyak attribútumaiban (`subjects.yml`) nem volt változás.

```
testneveles:
  name: Testneveles
  lecturer: Futo Bela
```

```
  pluto: futobe
  credit: 5

fizika:
  name: Fizika
  lecturer: Orosz Laszlo
  pluto: oroszl
  credit: 5
```

Az egyes kurzusok a `candidates.yml` fájlba kerülnek. A modellben két idegen kulcs is van, az egyik a tárgyra, a másik a félévre hivatkozik. A hivatkozást a YAML fájlokban lévő kulcsokkal tesszük meg.

```
one:
  subject: fizika
  semester: fall
  t: 0
  date1: 2022-10-11 13:32:39
  date2: 2022-10-11 13:32:39
  location1: F.E
  location2: F.E
  limit: 200

two:
  subject: fizika
  semester: spring
  t: 0
  date1: 2022-10-11 13:32:39
  date2: 2022-10-11 13:32:39
  location1: F.E
  location2: F.e
  limit: 200
```

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:~/pluto> RAILS_ENV='test' rails db:drop
Dropped database 'pluto_test'

kovacsg@debian:~/pluto> RAILS_ENV='test' rails db:create
Created database 'pluto_test'

kovacsg@debian:~/pluto$ RAILS_ENV='test' rails db:migrate
== 20220927113845 CreateUsers: migrating
   ==============================================
-- create_table(:users)
```

```
   -> 0.0080s
== 20220927113845 CreateUsers: migrated (0.0082s)
   ================================

== 20221011111456 CreateSubjects: migrating
   ===================================
-- create_table(:subjects)
   -> 0.0082s
== 20221011111456 CreateSubjects: migrated (0.0084s)
   ================================

== 20221011112331 CreateSemesters: migrating
   ===================================
-- create_table(:semesters)
   -> 0.0052s
== 20221011112331 CreateSemesters: migrated (0.0054s)
   ================================

== 20221011113239 CreateCourses: migrating
   ====================================
-- create_table(:courses)
   -> 0.0151s
== 20221011113239 CreateCourses: migrated (0.0153s)
   ================================

== 20221018102006 AddSaltToUsers: migrating
   ===================================
-- add_column(:users, :salt, :string)
   -> 0.0083s
-- change_column(:users, :name, :string, {:null=>false})
   -> 0.0179s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0055s
== 20221018102006 AddSaltToUsers: migrated (0.0319s)
   ================================

== 20221025113215 CreateJoinTableUserCourse: migrating
   ============================
-- create_join_table(:courses, :users)
   -> 0.0043s
== 20221025113215 CreateJoinTableUserCourse: migrated (0.0044s)
   ===================

== 20221115114430 CreateAttachments: migrating
   ================================
-- create_table(:attachments)
   -> 0.0051s
== 20221115114430 CreateAttachments: migrated (0.0052s)
   ================================
```

```
== 20221115122524 AddBeginAndEndsToSemester: migrating
   ========================
-- add_column(:semesters, :begins, :date)
   -> 0.0044s
-- add_column(:semesters, :ends, :date)
   -> 0.0029s
== 20221115122524 AddBeginAndEndsToSemester: migrated (0.0076s)
   ===============

kovacsg@debian:~/pluto> 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:

```
MariaDB [pluto_test]> select * from users;
+-----------+---------+-----------------+------------------------------------------+--------+
| id        | name    | email           | encrypted_password                       | pluto  |
| created_at          | updated_at          | salt                             |
+-----------+---------+-----------------+------------------------------------------+--------+
|  40473535 | Vala Ki | valaki@mail.bme.hu | 29afb30886516ea55fb1af614a43e4144ed40cd6 | valaki |
| 2022-11-29 12:42:30.833715 | 2022-11-29 12:42:30.833715 | valaki |
| 702572748 | Shen Ki | shenki@mail.bme.hu | 6fbd5d3ccdc7b98d253915203a412e084601248b | shenki |
| 2022-11-29 12:42:30.833715 | 2022-11-29 12:42:30.833715 | shenki |
+-----------+---------+-----------------+------------------------------------------+--------+
2 rows in set (0.000 sec)

MariaDB [pluto_test]> select * from semesters;
+-----------+-------------+------+--------+----------------------------+
| id        | name        | year | season | created_at                 |
| updated_at                 | begins     | ends       |
+-----------+-------------+------+--------+----------------------------+
| 725488746 | 2022/23 spring | 2023 |      0 | 2022-11-29 12:42:30.829007 |
| 2022-11-29 12:42:30.829007 | 2023-02-27 | 2023-07-10 |
| 995155727 | 2022/23 fall   | 2022 |      1 | 2022-11-29 12:42:30.829007 |
| 2022-11-29 12:42:30.829007 | 2023-09-05 | 2023-01-27 |
+-----------+-------------+------+--------+----------------------------+
2 rows in set (0.000 sec)

MariaDB [pluto_test]> select * from subjects;
+-----------+---------+----------+--------+--------+----------------------------+
| id        | name    | lecturer | pluto  | credit | created_at                 |
| updated_at                 |
+-----------+---------+----------+--------+--------+----------------------------+
```

```
|  74779437 | Fizika        | Orosz Laszlo | oroszl |        5 | 2022-11-29
    12:42:30.831034 | 2022-11-29 12:42:30.831034 |
| 437717502 | Testneveles | Futo Bela      | futobe |        5 | 2022-11-29
    12:42:30.831034 | 2022-11-29 12:42:30.831034 |
+-----------+-------------+--------------+--------+----------+----------------------------+

2 rows in set (0.000 sec)

MariaDB [pluto_test]> select * from courses;
+-----------+------------+-------------+------+-----------------------------+

| id        | subject_id | semester_id | t    | date1                       |
     date2                       | location1 | location2 | limit | created_at
                   | updated_at                 |
+-----------+------------+-------------+------+-----------------------------+

| 298486374 |   74779437 |   725488746 |    0 | 2022-10-11 13:32:39.000000 |
     2022-10-11 13:32:39.000000 | F.E        | F.e        |   200 | 2022-11-29
     12:42:30.832209 | 2022-11-29 12:42:30.832209 |
| 980190962 |   74779437 |   995155727 |    0 | 2022-10-11 13:32:39.000000 |
     2022-10-11 13:32:39.000000 | F.E        | F.E        |   200 | 2022-11-29
     12:42:30.832209 | 2022-11-29 12:42:30.832209 |
+-----------+------------+-------------+------+-----------------------------+

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 :username, presence: true
```

validációjának 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.

```
class UserTest < ActiveSupport::TestCase
  # def test_the_truth
  test "the truth" do
     assert true, "Houston, we have a problem"
  end

  test "cannot save user without name" do
    u = User.new
    u.email = 'ures@mail.bme.hu'
    u.pluto = 'aaaaaa'
    assert !u.save, "Houston, we have a problem"
  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:~/pluto> RAILS_ENV='test' rails test test/models/
    user_test.rb:9
Run options: --seed 43662

# Running:

.

Finished in 0.050167s, 19.9333 runs/s, 19.9333 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/pluto> RAILS_ENV='test' rails test test/models/
    user_test.rb
Run options: --seed 49778

# Running:

..

Finished in 0.047950s, 41.7099 runs/s, 41.7099 assertions/s.
2 runs, 2 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/pluto> RAILS_ENV='test' rails test test/models/
Run options: --seed 16951

# Running:

..

Finished in 0.049798s, 40.1624 runs/s, 40.1624 assertions/s.
2 runs, 2 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):023:0> include Rails.application.routes.url_helpers
=> Object
irb(main):024:0> errors_not_found_path
=> "/errors/not_found"
irb(main):025:0> errors_not_found_url
Traceback (most recent call last):
/var/lib/gems/2.7.0/gems/actionpack-7.0.4/lib/action_dispatch/
    http/url.rb:64:in `full_url_fo: Missing host to link to!
    Please provide the :host parameter, set default_url_options[:
    host or set :only_path to true (ArgumentError)
irb(main):026:0> default_url_options[:host] = 'http://localhost
    :3000'
=> "http://localhost:3000"
irb(main):027:0> errors_not_found_url
=> "http://localhost:3000/errors/not_found"
```

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. Az első a bejelentkező képernyő betöltését, a második bejelentkezést, 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 "laod_login_page" do
    get hogyhivjuk_path
    assert_response :success
    assert_select 'legend', "Login"
    assert_nil session[:user]
  end

  test "login" do
    get hogyhivjuk_path
    assert_response :success
    assert_select 'legend', "Login"
    assert_nil session[:user]

    post login_path, params: { pluto: users(:valaki).pluto, password: 'titok
        ' }
    assert_response :redirect
    follow_redirect!
    assert_select 'a', 'Logout'
    assert_not_nil session[:user]
  end

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

    post login_path, params: { pluto: users(:valaki).pluto, password: 'titok
        ' }
    assert_response :redirect
    follow_redirect!
    assert_select 'a', 'Logout'
    assert_not_nil session[:user]

    get logout_path
    follow_redirect!
```

```
    assert_nil session[:user]
    assert_select 'legend', "Login"
  end
end
```

A tesztet e kontrollerteszt kiválasztásával futtatjuk:

```
kovacsg@debian:~/pluto> RAILS_ENV='test' rails test test/
    controllers/sessions_controller_test.rb
Run options: --seed 18707

# Running:

...

Finished in 1.322267s, 2.2688 runs/s, 12.8567 assertions/s.
3 runs, 17 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:~/pluto> 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 az avatar feltöltése form.

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.

```
require 'test_helper'

class UploadAvatarTest < ActionDispatch::IntegrationTest
  # test "the truth" do
  #   assert true
  # end

  test "upload avatar for user" do
    get hogyhivjuk_path
    assert_response :success
    assert_select 'legend', "Login"
    assert_nil session[:user]

    post login_path, params: { pluto: users(:valaki).pluto, password: 'titok
        ' }
    assert_response :redirect
    follow_redirect!
    assert_select 'a', 'Profile'
    assert_not_nil session[:user]

    get edit_user_path(users(:valaki).id)
    assert_response :success
    assert_select 'legend', "Set Avatar"

    file = fixture_file_upload('test/fixtures/files/1', 'image/png')
    post upload_path(users(:valaki).id), params: { 'file': file }, headers:
        { "HTTP_REFERER": hello_path }
    assert_response :redirect
    assert_equal Attachment.all.size, 1
    assert File.exists?("public/avatars/#{Attachment.last.id}")
    follow_redirect!
    assert_select 'a', 'Logout'

    get logout_path
    follow_redirect!
    assert_nil session[:user]
    assert_select 'legend', "Login"
  end
end
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/pluto> RAILS_ENV='test' rails test test/
    integration/upload_avatar_test.rb
Run options: --seed 16510

# Running:

.

Finished in 1.321518s, 0.7567 runs/s, 10.5939 assertions/s.
1 runs, 14 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
  test "login" do
    visit '/home/login'

    assert_selector "legend", text: "Login"

    fill_in 'pluto', 'valaki'
    fill_in 'password', 'titok'
    click_on "Login"

    assert_selector "a", "Logout"
  end
end
```

A rendszertesztek futtatásakor meg-megnyílik a böngésző, és láthatjuk a tesztek végrehajtátás. A gyakorlaton a Chrome böngészőnk valamely bő-

vítmény 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:~/pluto> RAILS_ENV='test' rails test test/system/
    hellos_test.rb
Run options: --seed 46383

# Running:

DEBUGGER: Attaching after process 321211 fork to child process
    321220
Capybara starting Puma...
* Version 5.6.5 , codename: Birdie's Version
* Min threads: 0, max threads: 4
* Listening on http://127.0.0.1:40475
.

Finished in 6.455603s, 0.1549 runs/s, 0.3098 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 teljesítménytesztek.

A teljesítménytesztek 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.