Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2018. november 20.

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: En
  encrypted_password: <%= Digest::SHA1.hexdigest 'a' + 'titok' %>
  salt: a
  email: en@mail.bme.hu
  neptun: aaaaaa
  usertype: 0

<% for i in 1..5 do %>
student_<%=i%>:
  name: Hallgato_<%=i%>
  encrypted_password: <%=  Digest::SHA1.hexdigest 'a' + 'titok'
      %>
  salt: a
  email: hallgato_<%=i%>@mail.bme.hu
  neptun: aaaaa<%=i%>
  usertype: 1
<% end %>
```

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

A tesztadatbázisunkban több feladatsorra lesz szükségünk, ezeket a `quizzes.yml` fájlban definiáljuk.

```
nagyzh:
  deadline: 2018-11-21
  number: 1
  title: ZH

kiszh:
  deadline: 2018-11-22
  number: 2
  title: zh
```

A `tasks.yml` fájlban egy egy-több kapcsolat adatait adhatjuk meg. Ha a YAML fájlunkban egy idegen kulcshoz tartozó attribútuma értékének a hivatkozott típus tesztadatai közül egy kulcsát rendeljük, a Rails feloldja a kapcsolatot.

```
one:
```

```
  quiz: nagyzh
  number: 1
  text: Mi a neved?
  points: 1
  solution: A nev helyes megadasa

two:
  quiz: nagyzh
  number: 2
  text: Mi a Neptun-kodod?
  points: 1
  solution: A Neptun-kod helyes megadasa
```

A megoldások (`solutions.yml`) tesztadatfájlban két idegen kulcsunk is van.

```
one:
  user: student_1
  task: one


two:
  user: student_1
  task: two
```

Nézzük először meg a tesztadatbázisunkat:

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails db
MariaDB [gyakorlat_test]> show tables;
Empty set (0.00 sec)

MariaDB [gyakorlat_test]> Bye
```

Nincsenek se tábláink, se adataink a tesztadatbázisban, ezért töltsük be a teszt környezet adatbázisába ezeket a frissen létrehozott adatokat ügyelve arra, hogy a környezetként a teszt környezet legyen beállítva.

```
kovacsg@debian:~/gyakorlat/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
== 20180925111126 CreateUsers: migrating
    ==========================================
-- create_table(:users)
   -> 0.0097s
== 20180925111126 CreateUsers: migrated (0.0100s)
    =================================
```

```
== 20180925112158 CreateQuizzes: migrating ====================================
-- create_table(:quizzes)
   -> 0.0072s
== 20180925112158 CreateQuizzes: migrated (0.0074s) ===========================

== 20180925112707 CreateTasks: migrating ======================================
-- create_table(:tasks)
   -> 0.0096s
== 20180925112707 CreateTasks: migrated (0.0098s) =============================

== 20181009113053 CreateReviews: migrating ====================================
-- create_table(:reviews)
   -> 0.0095s
== 20181009113053 CreateReviews: migrated (0.0096s) ===========================

== 20181030111950 AddSaltToUsers: migrating ===================================
-- add_column(:users, :salt, :string)
   -> 0.0149s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0049s
== 20181030111950 AddSaltToUsers: migrated (0.0203s) ==========================

== 20181030122515 CreateSolutions: migrating ==================================
-- create_table(:solutions)
   -> 0.0171s
== 20181030122515 CreateSolutions: migrated (0.0174s) =========================

== 20181030122724 AddReviewableToReviews: migrating ===========================
-- add_column(:reviews, :reviewable_id, :integer)
   -> 0.0134s
-- add_column(:reviews, :reviewable_type, :string)
   -> 0.0188s
== 20181030122724 AddReviewableToReviews: migrated (0.0326s) ==================

== 20181106112024 AddUserToReviews: migrating =================================
```

```
-- add_reference(:reviews, :user, {:foreign_key=>true})
   -> 0.0416s
== 20181106112024 AddUserToReviews: migrated (0.0417s)
   ============================

== 20181106122450 CreateAttachments: migrating
   ============================
-- create_table(:attachments)
   -> 0.0061s
== 20181106122450 CreateAttachments: migrated (0.0062s)
   ============================

== 20181106122600 AddSolutionToAttachments: migrating
   ============================
-- add_reference(:attachments, :solution, {:foreign_key=>true})
   -> 0.0441s
== 20181106122600 AddSolutionToAttachments: migrated (0.0442s)
   ============================
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]> select * from users;
+-----------+-----------+----------------------------------+-----------------------
| id        | name      | encrypted_password               | email
                | neptun | usertype | created_at          | updated_at
          | salt |
+-----------+-----------+----------------------------------+-----------------------

| 103771790 | Hallgato_2 | a18f40e3799b53ea428c9e53e37ce92263f0b68b |
   hallgato_2@mail.bme.hu | aaaaa2 |        1 | 2018-11-20 11:32:06 |
   2018-11-20 11:32:06 | a    |
| 407632687 | Hallgato_5 | a18f40e3799b53ea428c9e53e37ce92263f0b68b |
   hallgato_5@mail.bme.hu | aaaaa5 |        1 | 2018-11-20 11:32:06 |
   2018-11-20 11:32:06 | a    |
| 522600246 | Hallgato_1 | a18f40e3799b53ea428c9e53e37ce92263f0b68b |
   hallgato_1@mail.bme.hu | aaaaa1 |        1 | 2018-11-20 11:32:06 |
   2018-11-20 11:32:06 | a    |
| 743609028 | En         | a18f40e3799b53ea428c9e53e37ce92263f0b68b |
   en@mail.bme.hu         | aaaaaa |        0 | 2018-11-20 11:32:06 |
   2018-11-20 11:32:06 | a    |
| 793562046 | Hallgato_4 | a18f40e3799b53ea428c9e53e37ce92263f0b68b |
   hallgato_4@mail.bme.hu | aaaaa4 |        1 | 2018-11-20 11:32:06 |
   2018-11-20 11:32:06 | a    |
| 824729113 | Hallgato_3 | a18f40e3799b53ea428c9e53e37ce92263f0b68b |
   hallgato_3@mail.bme.hu | aaaaa3 |        1 | 2018-11-20 11:32:06 |
   2018-11-20 11:32:06 | a    |
+-----------+-----------+----------------------------------+-----------------------
```

```
6 rows in set (0.00 sec)

MariaDB [gyakorlat_test]> select * from solutions;
+-----------+-----------+-----------+---------------------+---------------------+
| id        | user_id   | task_id   | created_at          | updated_at
          |
+-----------+-----------+-----------+---------------------+---------------------+
| 298486374 | 522600246 | 298486374 | 2018-11-20 11:32:06 | 2018-11-20
    11:32:06 |
| 980190962 | 522600246 | 980190962 | 2018-11-20 11:32:06 | 2018-11-20
    11:32:06 |
+-----------+-----------+-----------+---------------------+---------------------+

2 rows in set (0.00 sec)

MariaDB [gyakorlat_test]> Bye
```

3. Modellek tesztelése

Először írjunk egységteszteket! Az egységtesztek a modell osztályok metódusait és validációit hivatottak ellenőrizni. Az egyégteszteket a Rails projektünk `test/models` (Rails 4 előtt `test/unit`) könyvtárában találjuk. Minden egyes modell létrehozása után automatikusan létrejön hozzá egy ahhoz kapcsolódó teszt osztály itt.

Írjunk teszteseteket, amelyek a `User` modellünk

```
  validates :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, "Ha ez nem igaz, akkor nagyon nagy baj van"
  end

  test "encrypt password" do
    assert_equal users(:me).encrypted_password, Digest::SHA1.hexdigest(users
        (:me).salt+'titok')
  end

  test "cannot save without email address" do
    u = User.new
    assert !u.save, "Houston, we have a problem"
  end
end
```

A tesztesetet futtatva meggyőződhetünk róla, hogy tényleg nem történik meg a mentés. Ha mégis sikerülne, a megadott hibaüzenetnek kell megjelennie. Egyúttal böngészőből is kipróbálhatjuk, hogy működik a bejelentkezés. A futtatásra használhatjuk a `rake` és a `rails` parancsot is. Az utóbbi paranccsal futtathatjuk egy modell összes tesztesetét, valamint egy, fájlbeli sorszámmal megadot konkrét tesztesetét.

```
kovacsg@debian:~/gyakorlat/test/models> RAILS_ENV='test' rails
    test test/models/user_test.rb
Run options: --seed 46337

# Running:

...

Finished in 0.063687s, 47.1055 runs/s, 47.1055 assertions/s.
3 runs, 3 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat/test/models> RAILS_ENV='test' rails
    test test/models/user_test.rb:12
Run options: --seed 30315

# Running:

.

Finished in 0.068145s, 14.6745 runs/s, 14.6745 assertions/s.
1 runs, 1 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):018:0> include Rails.application.routes.url_helpers
=> Object
irb(main):020:0> hello_path
=> "/say/hello"
irb(main):021:0> default_url_options[:host] = 'http://localhost
    :3000'
=> "http://localhost:3000"
irb(main):022:0> hello_url
=> "http://localhost:3000/say/hello"
irb(main):023:0> url_for controller: 'say', action: 'hello'
=> "http://localhost:3000/say/hello"
```

```
irb(main):024:0> url_for controller: 'sessions', action: 'create'
=> "http://localhost:3000/sessions/create"
irb(main):025:0> url_for controller: 'sessions', action: 'create'
    , email: 'en@mail.bme.hu', password: 'titok'
=> "http://localhost:3000/sessions/create?email=en%40mail.bme.hu&
    password=titok"
irb(main):026:0> url_for controller: 'quizzes', action: 'index'
=> "http://localhost:3000/quizzes"
irb(main):029:0> url_for controller: 'quizzes', action: 'show',
    id: 1
=> "http://localhost:3000/quizzes/1"
irb(main):030:0> upload_path
```

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, és azon van-e egy `legend` HTML elem `Login` értékkel.

```
class SayControllerTest < ActionDispatch::IntegrationTest
  test "should get hello" do
    get url_for(controller: 'say', action: 'hello')
    assert_response :success
    assert_select "legend", 'Login'
  end

end
```

A funkcionális teszteseteket a `rake test:functionals` 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
Run options: --seed 32307

# Running:

.

Finished in 0.458123s, 2.1828 runs/s, 4.3656 assertions/s.
1 runs, 2 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`, továbbá az átirányítás utáni oldalon van egy kijelentkezést megvalósító 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 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 SessionControllerTest < ActionDispatch::IntegrationTest
  def setup
    @me = users(:me)
  end

  test "login" do
    get url_for(controller: 'say', action: 'hello')
    assert_response :success

    post url_for(controller: 'sessions', action: 'create'), params: { email:
        @me.email, password: 'titok' }, headers: { 'HTTP_REFERER': url_for
       (controller: 'say', a
ction: 'hello') }
    assert_response :redirect
    follow_redirect!
    assert_select 'a', 'Logout'
    assert_not_nil session[:user]
  end

  test "logout" do
    get url_for(controller: 'say', action: 'hello')
    assert_response :success

    post url_for(controller: 'sessions', action: 'create'), params: { email:
        @me.email, password: 'titok' }, headers: { 'HTTP_REFERER': url_for
```

```
        (controller: 'say', a
ction: 'hello') }
    assert_response :redirect

    assert_not_nil session[:user]

    get url_for(controller: 'sessions', action: 'destroy')
    assert_response :redirect
    follow_redirect!
    assert_select 'legend', "Login"

    assert_nil session[:user]
  end

end
```

A tesztet e kontrollerteszt kiválasztásával futtatjuk:

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    controllers/session_controller_test.rb
Run options: --seed 50152

# Running:

..

Finished in 0.541429s, 3.6939 runs/s, 18.4696 assertions/s.
2 runs, 10 assertions, 0 failures, 0 errors, 0 skips
```

6. Integrációs teszt

A tesztek harmadik típusa az integrációs teszt, amellyel egy feladatsor egy feladata megoldása beadásának folyamatát ellenőrzük.

```
kovacsg@debian:~/gyakorlat/test> rails g integration_test
    upload_solution
      invoke  test_unit
      create    test/integration/upload_solution_test.rb
```

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

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

- lekérdezünk egy képernyőt, ahol van bejelentkező form,
- kitöltjük a bejelentkezési formot, és rákattintunk bejelentkezés gombra,

- lekérdezzük a megoldandó feladatsorok listáját,
- kiválasztunk egy feladatsort,
- megoldjuk az első feladatot, és
- kilépünk.

Az első lépésben egy be nem jelentkezett felhasználó betölt egy véletlen oldalt, amin elérhető a bejelentkezési form. Itt annyit feltételezünk, hogy az oldal sikeresen betöltődik, és az oldalon HTML nézetének forrásában van pontosan egy form, valamint 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 feladatsorok

A feladatsorok megoldása linkre kattintva lekérdezzük a megoldható feladatsorok listáját az `index` nézettel. Az a feltételezésünk, hogy ott a táblázat törzsében két feladatsort találunk, lévén annyit definiáltunk a tesztadatok között.

A feladatsorok listája nézeten az egyes feladatsorok melletti `Show` linkre kattintva a feladatsor megoldásai szerkeszthetők. A tesztesetben az első feladatsort választjuk ki. Az a feltételezésünk, hogy az oldal betöltődik, és ott olvasható a feladat sorszáma és szövege.

A megoldás nézeten található egy form, amelyen kereszül egy `solution` 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 `Login` címkéjű `legend` HTML elem.

```
require 'test_helper'

class UploadSolutionTest < ActionDispatch::IntegrationTest
  # test "the truth" do
  #   assert true
  # end
  def setup
    @s = users(:student_1)
    @q = quizzes(:nagyzh)
  end

  test "upload_solution" do
    get url_for(controller: 'say', action: 'hello')
```

```
    assert_response :success
    assert_select 'legend', "Login"

    post url_for(controller: 'sessions', action: 'create'), params: { email:
        @s.email, password: 'titok' }, headers: { "HTTP_REFERER": url_for(
        controller: 'say', act
ion: 'hello') }
    assert_not_nil session[:user]
    follow_redirect!
    assert_select 'a', 'Solve'

    get url_for(controller: 'quizzes', action: 'index')
    assert_response :success
    assert_select 'tbody tr', Quiz.count

    get url_for(controller: 'quizzes', action: 'show', id: @q.id)
    assert_response :success
    first_task = @q.tasks.order(:number).first
    assert_select 'b', first_task.number.to_s+ '. '+first_task.text

    assert_equal 2, Attachment.count

    upload_file = fixture_file_upload('test/fixtures/files/
        rails_html_users_list_20182.png', 'image/png')
    post upload_solution_path(quiz: @q.id, task: first_task.id), params: {
        solution: upload_file }

    assert_equal 3, Attachment.count
    a = Attachment.order(:created_at).last
    assert File.exists?(Rails.root.join('public', 'data', a.id.to_s))

    get url_for(controller: 'sessions', action: 'destroy')
    assert_nil session[:user]
    follow_redirect!
    assert_select 'legend', "Login"

  endend
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    integration/upload_solution_test.rb
Run options: --seed 9849

# Running:

.

Finished in 0.363700s, 2.7495 runs/s, 35.7437 assertions/s.
1 runs, 13 assertions, 0 failures, 0 errors, 0 skips
```

7. 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
      create  test/performance/hello_test.rb
```

Az alábbi teszteset a kezdőoldalt támadja meg 5 egymás utáni kéréssel.

```
require 'test_helper'
require 'rails/performance_test_help'

class HelloTest < ActionDispatch::PerformanceTest
  # Refer to the documentation for all available options
  # self.profile_options = { runs: 5, metrics: [:wall_time, :
      memory],
  #                          output: 'tmp/performance', formats:
      [:flat] }

  test "homepage" do
    get '/'
  end
end
```

A tesztesek végrehajtásának naplója a `logs` könyvtárba kerül, a teljesítménytesztek részletes adatai emellett a `tmp/performance` könyvtárban is megjelennek szövegesen, táblázatosan esetleg képként grafikusan.

A gyakran végrehajtandó teljesítményteszt-célok vonatkozásában teszteseteket érdemes definiálnunk, amelyeket a `test:benchmark` és a `test:profile` `rails` célokkal hajthatunk végre. [1]

```
kovacsg@debian:~/gyakorlat> rails test:benchmark
Run options: --seed 13482

# Running:

HelloTest#test_homepage (0 ms warmup)
           wall_time: 0 ms
              memory: unsupported
             objects: unsupported
             gc_runs: 0
             gc_time: 0 ms
.

Finished in 1.032117s, 0.9689 runs/s, 0.0000 assertions/s.
1 runs, 0 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat$ RAILS_ENV='test' rails test:profile
Run options: --seed 55493

# Running:

HelloTest#test_homepage (0 ms warmup)
       process_time: 4 ms
             memory: 0 Bytes
            objects: 394
.

Finished in 2.771761s, 0.3608 runs/s, 0.0000 assertions/s.
1 runs, 0 assertions, 0 failures, 0 errors, 0 skips
```

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.

[1] Az ruby-prof aktuális verziójában hiba van, amelyet kijavítva kapunk csak eredményt: https://github.com/ruby-prof/ruby-prof/issues/203