Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2019. április 30.

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`.

```
valaki:
  name: Valaki
  email: valaki@mail.bme.hu
  encrypted_password: <%= Digest::SHA1.hexdigest 'titok'+'a' %>
  salt: a
```

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

A tesztadatbázisunkban több témára lesz szükségünk, ezeket a `topics.yml` fájlban definiáljuk, a tulajdonosuk az egyetlen felhasználó. Ha a YAML fájlunkban egy idegen kulcshoz tartozó attribútuma értékének a hivatkozott típus tesztadatai közül egy kulcsát rendeljük, a Rails feloldja a kapcsolatot.

```
ror:
  title: RoR
  user: valaki
  contents: RoR tema adatlap

matek:
  title: Matematika
  user: valaki
  contents: Matematika tema adatlap
```

A linkek (`links.yml`) tesztadatfájlban két idegen kulcsunk is van.

```
ror:
  user: valaki
  topic: ror
  url: http://ror.bme.hu
```

Csatolmányokat és URL-eket nem veszünk fel előzetesen, ezért töröljük azon tesztfájlok tartamát.

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
kovacsg@debian:~/gyakorlat/test/fixtures> RAILS_ENV='test' rails
    db
MariaDB [gyakorlat_test]> show tables;
+-----------------------------+
| Tables_in_gyakorlat_test    |
+-----------------------------+
| ar_internal_metadata        |
| attachments                 |
| comments                    |
| links                       |
| schema_migrations           |
| subtopics_topics            |
| topics                      |
| users                       |
+-----------------------------+
8 rows in set (0.00 sec)

MariaDB [gyakorlat_test]> select * from users;
Empty set (0.00 sec)

MariaDB [gyakorlat_test]> Bye
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   | email             | encrypted_password
                    | created_at          | updated_at          | salt
    |
+----------+--------+-------------------+------------------------------------------------------------------+

| 40473535 | Valaki | valaki@mail.bme.hu |
```

```
    acc7912d641fab211dc622a39b788da34eadc0ee | 2019-04-30 10:23:34 |
    2019-04-30 10:23:34 | a    |
+-----------+----------+----------------------------------+---------------------

1 row in set (0.00 sec)

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

| id        | title    | user_id  | contents              | created_at
            | updated_at          |
+-----------+----------+----------+-----------------------+---------------------

| 878314071 | RoR      | 40473535 | RoR tema adatlap      | 2019-04-30
    10:23:34 | 2019-04-30 10:23:34 |
| 889729851 | Matematika | 40473535 | Matematika tema adatlap | 2019-04-30
    10:23:34 | 2019-04-30 10:23:34 |
+-----------+----------+----------+-----------------------+---------------------

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 :name, presence: true
  validates :email, { presence: true, uniqueness: true }
```

validációinak megtörténtét ellenőrzik. Először létrehozunk egy új felhasználót mindenféle inicializáció nélkül, majd megpróbáljuk eltárolni az adatbázisba. A feltételezésünk az, hogy a mentésnek nem szabad sikerülnie.

```
class UserTest < ActiveSupport::TestCase
  test "the_truth" do
    assert true
  end


  test 'cannot_save_user_without_name' do
    u = User.new email: 'senki@mail.bme.hu'
    assert !u.save, "Houston,_we_have_a_problem"
  end

  test 'cannot_save_user_without_email_address' do
    u = User.new name: 'Senki'
    assert !u.save, "Houston,_we_have_a_problem"
  end
```

```
  test 'cannot save user with existing email address' do
    u = User.new email: 'valaki@mail.bme.hu'
    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# RAILS_ENV='test' rails test test/
    models/user_test.rb:4
Run options: --seed 22499

# Running:

.

Finished in 0.036920s, 27.0855 runs/s, 27.0855 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rails test test/
    models/user_test.rb
Run options: --seed 44789

# Running:

....

Finished in 0.061789s, 64.7360 runs/s, 64.7360 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.

```
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: 'topics', action: 'index'
=> "http://localhost:3000/topics"
irb(main):029:0> url_for controller: 'topics', action: 'show', id
    : 1
=> "http://localhost:3000/topics/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 `valaki` kulcshoz tartozó felhasználó tesztadatain végezzük el. Az első a bejelentkezést, a második a kijelentkezést teszteli, nevezzük át a teszteket ennek megfelelően! A bejelentkezés eseményt a `SessionsController` `create` metódusát használja, a kijelentezés esemény pedig a `destroy` metódust. A tesztben a `post` metódust használjuk, amelynek első és egyetlen kötelező paramétere a tesztelendő URL (Rails 5 előtt tesztelendő akció). A `params` kulcshoz rendelhetjük a HTTP kérés paramétereit, a `headers` kulcshoz pedig a kiküldendő HTTP kérés fejléc beállításait. A sikeres bejelentkezés tesztben a tesztadatokban szereplő felhasználónév, jelszó párost küldjük el, és azt feltételezzük, hogy visszairányítódunk a belépés előtti nézetre, a `:user` session paraméter értéke nem `nil`, hanem a bejelentkezett felhasználó azonosítójával megegyező érték, továbbá az átirányítás utáni oldalon van egy a felhasználói profil szerkesztésére mutató link. Az átirányítás válaszban kapott URI-t a `follow_redirect!` függvénnyel nyithatjuk meg, és azon az `assert_select` függvénnyel ellenőrizthetjük, hogy tényleg az az oldal, és tényleg azzal a tartalommal jelent-e meg. A sikertelen bejelentkezést ellenőrző tesztesetben a jelszó paraméterben teszünk különbséget, és azt várjuk, hogy visszairányítódunk az előző oldalra, a session üres, és van az oldalon egy `Register` címkéjű link. A kilépés tesztben HTTP kérés paramétereket nem adunk meg, viszont azt kell feltételeznünk előzetesen, hogy a felhasználó be van jelentkezve, vagyis a `session` hash `user` kulcshoz tartozó értéke létezik. Ezt úgy érthetjük el, hogy a teszteset prefixeként lefuttatjuk a bejelentkezés tesztesetet. Válaszként átirányítást várunk az aktuális oldalunkra, és azt, hogy a session paraméter kinullázódik, és az átirányítás után oldalon lesz egy `Register` értékű link HTML elem. Mivel az átirányítás mindkét esetben a `back` URL-re történik, és a tesztvégrehajtást nem böngészőből végezzük, és nem érhető el a Javascript `history`, a `HTTP_REFERRER` fejrészt kell beállítanunk.

```
class SessionsControllerTest < ActionDispatch::IntegrationTest
  test "login" do
    post '/sessions/create', params: { email: users(:valaki).email ,
        password: 'titok'}, headers: { 'HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    assert_equal session[:user], users(:valaki).id
    follow_redirect!
    assert_select 'a', 'Profile'
  end

  test "invalid_login" do
```

```
    post '/sessions/create', params: { email: users(:valaki).email ,
        password: 'titok2'}, headers: { 'HTTP_REFERER': '/say/hello' }
    assert_response :redirect
    assert_nil session[:user]
    follow_redirect!
    assert_select 'a', 'Register'
  end



  test "logout" do
    post '/sessions/create', params: { email: users(:valaki).email ,
        password: 'titok'}, headers: { 'HTTP_REFERER': '/say/hello' }
    get '/sessions/destroy', headers: { "HTTP_REFERER": '/say/hello' }
    assert_response :redirect
    assert_nil session[:user]
    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
Run options: --seed 5930

# Running:

...

Finished in 0.248655s, 12.0649 runs/s, 32.1731 assertions/s.
3 runs, 9 assertions, 0 failures, 0 errors, 0 skips
```

A témát tesztjét scaffolddal generáltuk, módosítsuk, hogy alkalmas legyen a módosításaink ellenőrzésére. Az alapértelmezett tesztek csak a HTTP válaszok státusz kódját ellenőrzik, ezen túl nem megyünk. A futáshoz azonban szükséges, hogy a tesztadat megfelelő legyen, és a műveleteket csak egy bejelentkezett felhasználó hajthassa végre (a bejelentkezés végrehajtó HTTP műveletet átemeljük a login tesztből). Ezeket a `setup` törzsében intézhetjük el.

```
class TopicsControllerTest < ActionDispatch::IntegrationTest
  setup do
    @topic = topics(:ror)
    post '/sessions/create', params: { email: users(:valaki).
        email , password: 'titok'}, headers: { 'HTTP_REFERER': '/
        say/hello' }
  end
end
```

6. Integrációs teszt

A tesztek harmadik típusa az integrációs teszt, amellyel egy téma adatlapjára csatolmány feltöltésének folyamatát ellenőrzük.

```
kovacsg@debian:~/gyakorlat/test/integration# rails g
    integration_test attach_to_topic
      invoke  test_unit
      create    test/integration/attach_to_topic_test.rb
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `attach_to_topic_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 témák listáját,
- kiválasztunk egy témát,
- csatolunk egy fájlt a témához, 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, a session üres, és az oldalon HTML nézetének forrásában van egy `Register` címkéjű link HTML elem, valamint egy `Email` és egy `Password` címkéjű HTML címke.

A második tesztlépés a bejelentkezés linkre „kattintás”. Az a feltételezünk, hogy a felhasználói `session` beállítódik, és az átirányítás utáni oldalon a menüben elérhetők a témák listája link.

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

A témák listája nézeten az egyes témák melletti `Show` linkre kattintva a téma szerkeszthető. A tesztesetben az első témát választjuk ki. Az a feltételezésünk, hogy az oldal betöltődik, és ott lehetőség van fájl csatolására.

A téma nézeten található egy form, amelyen kereszül egy `attachment` 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 AttachToTopicTest < ActionDispatch::IntegrationTest
  test "Attach file to topic" do
    get '/say/hello'
    assert_nil session[:user]
    assert_select 'a', 'Register'
    assert_select 'label', "Email"
    assert_select "label", "Password"

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

    get '/topics'
    assert_response :success
    assert_select 'h1', 'Topics'
    assert_select 'table', 1
    assert_select 'tbody tr', Topic.all.size

    get '/topics/'+topics(:ror).id.to_s
    assert_response :success
    assert_select 'legend', 'Attach a new file'

    upload_file = fixture_file_upload('test/fixtures/files/hello.txt', 'text
        /plain')
    post "/attachments/create/#{topics(:ror).id}/#{users(:valaki).id}",
        params: { attachment: { upload:  upload_file  } }
    assert_response :redirect
    assert_equal Attachment.all.size, 1
    assert File.exist? 'public/data/'+Attachment.last.id.to_s
    follow_redirect!
    assert_select 'a', 'Logout'

    get '/sessions/destroy'
    assert_response :redirect
    follow_redirect!
    assert_select 'a', "Register"
    assert_nil session[:user]
  end
end
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/gyakorlat/app# RAILS_ENV='test' rails test test/
    integration/attach_to_topic_test.rb
Run options: --seed 42646

# Running:

.

Finished in 0.315134s, 3.1732 runs/s, 66.6382 assertions/s.
1 runs, 21 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 22289

# Running:

LoadTest#test_homepage (141 ms warmup)
            wall_time: 4 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 44510

# Running:

LoadTest#test_homepage (133 ms warmup)
          process_time: 25 ms
          memory: 0 Bytes
          objects: 2,517
          gc_runs: 0
          gc_time: 0 ms

.
```

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

```
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.