Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2016. november 22.

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.

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 két `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` osztálymetódusát hívjuk segítségül. 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`..

```
<% MY_SALT = 'valami' %>
me:
  username: kovacsg
  encrypted_password: <%= User.encrypt 'titok', MY_SALT %>
  salt: <%= MY_SALT %>
```

```
  email: kovacsg@tmit.bme.hu

valaki:
  username: valaki
  encrypted_password: <%= User.encrypt 'a', MY_SALT %>
  salt: <%= MY_SALT %>
  email: valaki@mail.bme.hu
```

A tesztadatbázisunkban több kérdőívre lesz szükségünk, ezeket a `questionnaires.yml` fájlban definiáljuk. Ha a YAML fájlunkban egy ilyen kulcshoz a hivatkozott tesztadat egy kulcsát rendeljük, a Rails feloldja a kapcsolatot. Az adatstruktúrához hozzá kell adnunk az utólag felvett `status` attribútumot.

```
nagy:
  title: NagyZH
  description: Az elso nagy zh kerdesei
  user: me
  status: 0

kicsi:
  title: KisZH
  description: Az elso kis zh kerdesei
  user: valaki
  status: 1
```

Két tesztadatot veszünk fel a kérdések modellje (`questions.yml`) számára. A struktúrát ki kell egészítenünk az utólag felvett `question` attribútummal. A kérdések egy-több relációban áll az kérdőívekkel modellel, tartozzon mindkét kérdésünk az első kérdőívünkhöz, a kulcsunk itt megfelel a Rails konvencióinak.

```
elso:
  questionnaire: nagyzh
  type: 1
  index: 1
  question: Mennyi 2 + 2?

masodik:
  questionnaire: nagyzh
  type: 1
  index: 2
  question: Hanyast kapok ebbol a targybol?
```

A negyedik modellünk, a válaszlehetőségek modell tesztadatai közé vegyünk fel négy tesztadatot, melyek mind az `elso` kérdésre adható lehetőségek.

```
one:
  question: elso
```

```
  index: 1
  answer: 3

two:
  question: elso
  index: 2
  answer: 5

three:
  question: elso
  index: 3
  answer: 4

four:
  question: elso
  index: 4
  answer: Egyik sem
```

Nézzük először meg a tesztadatbázisunkat:

```
kovacsg@debian:~/gyakorlat/test/fixtures# RAILS_ENV='test' rails
    db
mysql> show tables;
Empty set (0.00 sec)
```

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' rake
    db:create
(in /home/kovacsg/gyakorlat)
Database 'gyakorlat_test' already exists
kovacsg@debian:~/gyakorlat/test/fixtures# RAILS_ENV='test' rake
    db:migrate
(in /home/kovacsg/gyakorlat)
== 20160913192820 CreateUsers: migrating
   ==========================================
-- create_table(:users)
   -> 0.0305s
== 20160913192820 CreateUsers: migrated (0.0312s)
   =================================

== 20161011110729 CreateQuestionnaires: migrating
   ==========================================
-- create_table(:questionnaires)
   -> 0.0906s
== 20161011110729 CreateQuestionnaires: migrated (0.0911s)
   =================
```

```
== 20161011111938 CreateQuestions: migrating
   ================================================
-- create_table(:questions)
   -> 0.0252s
== 20161011111938 CreateQuestions: migrated (0.0258s)
   ===========================

== 20161011112248 CreateAnswers: migrating
   ================================================
-- create_table(:answers)
   -> 0.0688s
== 20161011112248 CreateAnswers: migrated (0.0695s)
   ===========================

== 20161025101806 AddSaltToUsers: migrating
   ================================================
-- add_column(:users, :salt, :string)
   -> 0.0918s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0390s
== 20161025101806 AddSaltToUsers: migrated (0.1316s)
   ===========================

== 20161108111740 AddStatusToQuestionnaire: migrating
   ================================================
-- add_column(:questionnaires, :status, :integer, {:limit=>1})
   -> 0.0405s
== 20161108111740 AddStatusToQuestionnaire: migrated (0.0408s)
   ===========================

== 20161108123313 AddQuestionToQuestions: migrating
   ================================================
-- add_column(:questions, :question, :text)
   -> 0.0712s
== 20161108123313 AddQuestionToQuestions: migrated (0.0714s)
   ===========================
kovacsg@debian:~/gyakorlat/test/fixtures# RAILS_ENV='test' rake
    db:fixtures:load
(in /home/kovacsg/gyakorlat)
```

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

mysql> show tables;
```

```
+-----------------------------+
| Tables_in_gyakorlat_test    |
+-----------------------------+
| answers                     |
| ar_internal_metadata        |
| questionnaires              |
| questions                   |
| schema_migrations           |
| users                       |
+-----------------------------+
6 rows in set (0.00 sec)

mysql> select * from users;
+-----------+----------+------------------------------------------+-----------------+---------------------+---------+
| id        | username | encrypted_password                       | email           | created_at          | updated_at          | salt    |
+-----------+----------+------------------------------------------+-----------------+---------------------+---------+
|  40473535 | valaki   | d75856db66fe0e84ac32947089300dd3393bce9c | valaki@mail.bme.hu | 2016-11-22 11:22:38 | 2016-11-22 11:22:38 | valami |
| 743609028 | kovacsg  | 4438dd3bffd7c4843a8606b8083ae9980951c3fa | kovacsg@tmit.bme.hu | 2016-11-22 11:22:38 | 2016-11-22 11:22:38 | valami |
+-----------+----------+------------------------------------------+-----------------+---------------------+---------+

2 rows in set (0.00 sec)

mysql> select * from questionnaires;
+-----------+---------+-----------------------------+-----------+---------------------+---------------------+--------+
| id        | title   | description                 | user_id   | created_at          | updated_at          | status |
+-----------+---------+-----------------------------+-----------+---------------------+---------------------+--------+
| 335777656 | KisZH   | Az elso kis zh kerdesei     |  40473535 | 2016-11-22 11:22:38 | 2016-11-22 11:22:38 |      1 |
| 818793157 | NagyZH  | Az elso nagy zh kerdesei    | 743609028 | 2016-11-22 11:22:38 | 2016-11-22 11:22:38 |      0 |
+-----------+---------+-----------------------------+-----------+---------------------+---------------------+--------+

2 rows in set (0.00 sec)
```

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
```

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,
      uniqueness: true,
      length: { in: 4..18 }
    }
  validates :email,
    presence: 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.

Írjunk még egy tesztesetet, ami a modell két metódusát ellenőrzi: az általunk írt `encrypt` osztálymetódust teszteli egy ismert tesztadat alapján.

```
require 'test_helper'

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
     assert !u.save, "Houston, we have a problem"
  end

  test "cannot save user with existing username" do
     u = User.new
     u.username = 'valaki'
     assert !u.save
  end

  test "cannot save user with username of length of 1" do
    u = User.new
    u.username = 'a'
    assert !u.save
  end
```

```
  test "encrypt with SHA1 hexdigest" do
    assert_equal User.encrypt('titok', 'salt'), Digest::SHA1.
        hexdigest("salttitok")
  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# rake test:models
(in /home/kovacsg/gyakorlat)
Run options: --seed 36273

# Running:

.....

Finished in 0.282193s, 17.7184 runs/s, 17.7184 assertions/s.

5 runs, 5 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat# rails test test/models
Run options: --seed 33336

# Running:

.....

Finished in 0.109673s, 45.5899 runs/s, 45.5899 assertions/s.

5 runs, 5 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat/test/models# rails test test/models/
    user_test.rb
Run options: --seed 38752

# Running:

.....

Finished in 0.291094s, 17.1766 runs/s, 17.1766 assertions/s.

5 runs, 5 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat/test/models# rails test test/models/
    user_test.rb:9
Run options: --seed 36420
```

```
# Running:

.

Finished in 0.099225s, 10.0781 runs/s, 10.0781 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

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 `SessionsControllerTest` tesztet, amely jelenleg két tesztesetet tartalmaz. 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 `SessionController 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`, ráadásul megegyezik a tesztadat azonosítójával. 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. 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 login_url, params: { username: users(:valaki).username,
        password: 'a'}, headers: { "HTTP_REFERER" => '/say/hello'
         }
    assert_response :redirect
    assert_equal session[:user], users(:valaki).id
  end
```

```
  test "logout" do
    post login_url, params: { username: users(:valaki).username,
        password: 'a'}, headers: { "HTTP_REFERER" => '/say/hello'
         }
    assert_equal session[:user], users(:valaki).id
    get logout_url, headers: { "HTTP_REFERER" => '/say/hello' }
    # Rails 4-ig
    # get :destroy, nil, { user: users(:valaki).id }
    assert_response :redirect
    assert_nil session[:user]
  end
end
```

A `QuestionnairesControllerTest`-et, a `QuestionsControllerTest`-et, és az `AnswersControllerTest`-et `scaffold` szkripttel hoztuk létre, ami automatikusan generálta a tesztadatokat, a teszteseteket és a tesztesetek törzsét. A tesztadatok kulcsait módosítottuk, így az azokra való hivatkozást frissítenünk kell a `setup`-ban.

A felhasználók kontroller automatikus tesztjei hiányosak, a `show` és `edit` útvonalakhoz és nézetekhez utólag hozzáadtuk a felhasználó azonosítóját, így azt paraméterként át kell adnunk. Mivel a HTTP kérés első paramétere az URI, nevezzük el a `routes.rb`-ben az útvonalakat, hogy a helperekkel könnyebben hivatkozhassuk őket.

```
  get 'users/new', to: 'users#new', as: 'new_user'
  post 'users/create', to: 'users#create', as: 'user_create'

  get 'users/edit/:id', to: 'users#edit', as: 'edit_user'
  put 'users/update/:id', to: 'users#update', as: 'update_user'

  get 'users/show/:id', to: 'users#show', as: 'user_show'

  get 'users/forgotten', to: 'users#forgotten', as: '
      forgotten_user'
  post 'users/send_forgotten', to: 'users#send_forgotten', as: '
      send_forgotten_user'
```

Nézzük a felhasználók kontroller tesztjeit! A `valaki` kulccsal definiált felhasználót vizsgáljuk, a `setup` inicializáló metódusban beállítunk egy, az összes tesztesetben felhasználható példányváltozót e célból. Az új felhasználók nézetén azt várjuk, hogy két formunk van `legend` fejléccel, egy bejelentkezési és a regisztrációs, valamint a HTTP válasz státusz kódja 200. A regisztráció műveletének átadjuk a form paramétereit, és átirányítást várunk válaszként, továbbá azt, hogy a session inicializálódik. A felhasználói profil szerkesztéséhez előzetesen be kell jelentkeznünk, 200-as HTTP válasz státusz

kódot várunk. A profil módosítása akció teszteléséhez is be kell jelentkezünk előzetesen, a felhasználó `email` attribútumának módosítása után azt várjuk, hogy az adatbázisban megváltozik az előzetesen megadott adat. A felhasználói profil megtekintéséhez is be kel jelentkeznünk, és 200-as HTTP válasz státusz kódot várunk.

```
require 'test_helper'

class UsersControllerTest < ActionDispatch::IntegrationTest
  setup do
    @valaki = users(:valaki)
  end

  teardown do
  end

  test "should get new" do
    get new_user_url
    assert_response :success
    assert_select 'legend', 'Register a new user'
    assert_select 'legend', 2
  end

  test "register a new user" do
    post user_create_url, params: { user: { username: 'senki',
        password: 'a', password_confirmation: 'a', email: '
        senki@mail.bme.hu' } }
    assert_response :redirect
    assert_not_nil session[:user]
  end

  test "should get edit" do
    post login_url, params: { username: users(:valaki).username,
        password: 'a'}, headers: { "HTTP_REFERER" => '/say/hello'
         }
    assert_equal session[:user], users(:valaki).id
    get edit_user_url(@valaki.id)
    assert_response :success
  end

  test "update user profile" do
    post login_url, params: { username: users(:valaki).username,
        password: 'a'}, headers: { "HTTP_REFERER" => '/say/hello'
         }
    assert_equal session[:user], users(:valaki).id
    put update_user_url(@valaki.id), params: { user: { username:
        @valaki.username, email: 'valaki2@mail.bme.hu' } },
        headers: { "HTTP_REFERER" => say_he
llo_path }
```

```
    @valaki2 = User.find @valaki.id
    assert_not_equal @valaki2.email, @valaki.email
  end

  test "should get show" do
    post login_url, params: { username: users(:valaki).username,
        password: 'a'}, headers: { "HTTP_REFERER" => '/say/hello'
         }
    assert_equal session[:user], users(:valaki).id
    get user_show_url(@valaki.id)
    assert_response :success
  end

  test "should get forgotten" do
    get forgotten_user_url
    assert_response :success
  end

end
```

A funkcionális teszteseteket a `rake test:functionals` szkripttel vagy a `rails test` paranccsal futtathatjuk.

```
kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rails test test/
    controllers/sessions_controller_test.rb
Run options: --seed 52761

# Running:

.

Finished in 0.629814s, 3.1755 runs/s, 7.9388 assertions/s.

2 runs, 5 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rails test test/
    controllers/users_controller_test.rb
Run options: --seed 53176

# Running:

...

Finished in 1.158039s, 5.1812 runs/s, 10.3623 assertions/s.

6 runs, 12 assertions, 0 failures, 0 errors, 0 skips
```

A tesztek harmadik típusa az integrációs teszt, amellyel egy regisztrációs folyamatot ellenőrzünk. Készítsünk egy tesztet a felhasználói profil szerkesztésére!

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

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

A tesztadatok a `test_helper` fájlra való hivatkozás miatt automatikusan elérhetők. A teszt három tesztlépésből áll: lekérdezünk egy képernyőt, ahol van bejelentkező form, „rákattintuk" a regisztrációs linkre betöltve a regisztrációs formot, kitöltjük a formot és elküljük, végül ellenőrizzük, hogy sikerült-e a regisztráció.

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 `Register` címkéjű link HTML elem.

A második tesztlépés a regisztrációs linkre „kattintás". Az a feltételezünk, hogy az oldal ennek hatására sikeresen betöltődik, két darab formunk van, egy-egy `legend` HTML elemmel, és azok közül az egyik felirata a regisztrációra vonatkozik. Végül a `session` vizsgálatával ellenőrizzük, hogy véletlenül sincs a felhasználó bejelentkezve.

A regisztrációs nézeten található egy form, amelyen kereszül `username`, `email`, `password` és `password_confirmation` kérés paramétereket juttathatunk el a `/users/create` kontroller akciónak. Azt várjuk, hogy átirányítás történik, a `session` inicializálódik, és az adatbázisba a paraméterként megadott felhasználónévvel létrejön egy új rekord.

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

  test "register a new user with valid parameters" do
    get say_hello_url
    assert_response :success
    assert_select 'a', "Register"

    get new_user_url
    assert_response :success
    assert_select 'legend', 'Register a new user'
    assert_select 'legend', 2
    assert_nil session[:user]
```

```
    post user_create_url, params: { user: { username: 'senki',
        password: 'a', password_confirmation: 'a', email: '
        senki@mail.bme.hu' } }
    assert_response :redirect
    assert_not_nil session[:user]
    assert User.where(username: 'senki').count > 0
  end
end
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rails test test/
    integration/register_user_test.rb
Run options: --seed 19540

# Running:

.

Finished in 0.986057s, 1.0141 runs/s, 9.1273 assertions/s.

1 runs, 9 assertions, 0 failures, 0 errors, 0 skips
```

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, 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/test# rails g performance_test
    say_hello_test
      create  test/performance/say_hello_test_test.rb
```

Az alábbi teszteset az `hello` nézetet támadja meg 5 egymás utáni kéréssel.

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

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

  test "say_hello" do
    get say_hello_url
  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:profile` és a `test:benchmark` `rake` célokkal hajthatunk végre.

```
kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rake test:benchmark
Run options: --seed 2439

# Running:

SayHelloTestTest#test_say_hello (257 ms warmup)
           wall_time: 13 ms
              memory: unsupported
             objects: unsupported
             gc_runs: 0
             gc_time: 0 ms
.

Finished in 2.499283s, 0.4001 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.

```
kovacsg@debian:~/gyakorlat# perftest benchmarker User.all
```

```
Run options: --seed 10028

# Running:

BenchmarkerTest#test_user_all (21 ms warmup)
            wall_time: 0 ms
               memory: unsupported
              objects: unsupported
              gc_runs: 0
              gc_time: 0 ms
.

Finished in 1.559985s, 0.6410 runs/s, 0.0000 assertions/s.

1 runs, 0 assertions, 0 failures, 0 errors, 0 skips
```