Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2017. április 25.

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`.

```
me:
  name: Kovacs Gabor
  encrypted_password: <%= User.encrypt 'titok', 'salt' %>
  salt: salt
  email: kovacsg@tmit.bme.hu
```

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

A tesztadatbázisunkban több receptre lesz szükségünk, ezeket a `recipes.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 `complexity` és `cook_time` attribútumokat.

```
forro_viz:
  name: Forro viz
  description: Forralj fel vizet
  user: me
  complexity: 1
  cook_time: 3

kenyer:
  name: Kenyer
  description: Keverj ossze lisztet vizzel, dagaszd meg, keleszd
      masfel oraig, susd ki 1 oran keresztul
  user: valaki
  complexity: 1
  cook_time: 180
```

Két tesztadatot veszünk fel az összetevők modellje (`ingredients.yml`) számára. A struktúrát ki kell egészítenünk az utólag felvett `recipe` idegen kulcs attribútummal. A receptek egy-több relációban áll az összetevők modellel, tartozzon mindkét összetevőnk a második receptünkhöz, a kulcsunk itt megfelel a Rails konvencióinak.

```
viz:
  element: viz
  amount: 40
  unit: dl
  recipe: kenyer

liszt:
  element: liszt
  amount: 40
  unit: dkg
  recipe: kenyer
```

A negyedik modellünk, a képek modellje tesztadatait egyelőre nem módosítjuk.

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# RAILS_ENV='test' rails db:drop
Dropped database 'gyakorlat_test'
kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rails db:create
Created database 'gyakorlat_test'
kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rails db:migrate
== 20170228122321 CreateUsers: migrating
   =============================================
-- create_table(:users)
   -> 0.0067s
== 20170228122321 CreateUsers: migrated (0.0072s)
   ================================

== 20170228123223 CreateIngredients: migrating
   ========================================
-- create_table(:ingredients)
   -> 0.0055s
== 20170228123223 CreateIngredients: migrated (0.0084s)
   ===============================

== 20170314121705 CreateRecipes: migrating
   ========================================
-- create_table(:recipes)
   -> 0.0073s
== 20170314121705 CreateRecipes: migrated (0.0077s)
   ===============================

== 20170328101855 AddSaltToUsers: migrating
   ========================================
-- add_column(:users, :salt, :string)
   -> 0.0162s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0474s
== 20170328101855 AddSaltToUsers: migrated (0.0643s)
   ===============================

== 20170328102214 AddComptexityAndCookTimeToRecipes: migrating
   ==========================
-- add_column(:recipes, :complexity, :integer, {:limit=>1})
   -> 0.0161s
```

```
-- add_column(:recipes, :cook_time, :integer, {:limit=>3})
   -> 0.0188s
== 20170328102214 AddComptexityAndCookTimeToRecipes: migrated
   (0.0358s) =======

== 20170328112401 AddRecipeToIngredients: migrating
   ===============================
-- add_reference(:ingredients, :recipe, {:foreign_key=>true})
   -> 0.2078s
== 20170328112401 AddRecipeToIngredients: migrated (0.2080s)
   ==================

== 20170411103107 CreateImages: migrating
   ======================================
-- create_table(:images)
   -> 0.1201s
== 20170411103107 CreateImages: migrated (0.1206s)
   =============================
kovacsg@debian:~/gyakorlat# 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

mysql> show tables;
+-----------------------------+
| Tables_in_gyakorlat_test    |
+-----------------------------+
| ar_internal_metadata        |
| images                      |
| ingredients                 |
| recipes                     |
| schema_migrations           |
| users                       |
+-----------------------------+
6 rows in set (0.00 sec)

mysql> select * from users;
+-----------+--------------+----------------------------------+-----------------------+
| id        | name         | encrypted_password               |
   email                | created_at          | updated_at          | salt |
+-----------+--------------+----------------------------------+-----------------------+

|  40473535 | Vali Ki      | 7bbdaa81448361608259142d64b31a14b254dfc8 |
   valaki@mail.bme.hu   | 2017-04-27 13:29:36 | 2017-04-27 13:29:36 | salt |
| 743609028 | Kovacs Gabor | 2c4e6eae823622b73f22c6f26d4293884e7ceb8c |
   kovacsg@tmit.bme.hu | 2017-04-27 13:29:36 | 2017-04-27 13:29:36 | salt |
+-----------+--------------+----------------------------------+-----------------------+

2 rows in set (0.00 sec)
```

```
mysql> select * from recipes;
+-----------+-----------+------------------------------------------------------------

| id        | name      | description

    | user_id    | created_at           | updated_at           | complexity |
    cook_time |
+-----------+-----------+------------------------------------------------------------

| 557420569 | Forro viz | Forralj fel vizet
                                                                                    |
     743609028 | 2017-04-27 13:29:36 | 2017-04-27 13:29:36 |          1 |
          3 |
| 612104732 | Kenyer    | Keverj ossze lisztet vizzel, dagaszd meg, keleszd
    masfel oraig, susd ki 1 oran keresztul |  40473535 | 2017-04-27 13:29:36
     | 2017-04-27 13:29:36 |          1 |       180 |
+-----------+-----------+------------------------------------------------------------

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 :name, {
          length: {maximum: 40},
          presence: true
  }
  validates :email, {
          length: {maximum: 60},
          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.

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 email address" do
    u = User.new
    assert !u.save, "I could save a user without an email address"
  end

  test "cannot save user with existing email address" do
    u = User.new password: 'a', password_confirmation: 'a', name: 'A', email
        : 'valaki@mail.bme.hu'
    assert !u.save, "I could save a user without an existing email address"
  end

  test "encrypt password" do
    assert_equal User.encrypt('titok', 'salt'), Digest::SHA1.hexdigest('
        titok'+'salt'), "Fix the encryption"
  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:models
Run options: --seed 52507

# Running:

....

Finished in 0.097311s, 41.1055 runs/s, 41.1055 assertions/s.

4 runs, 4 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rails test test/
    models/user_test.rb:18
Run options: --seed 9001

# Running:

.

Finished in 0.087369s, 11.4457 runs/s, 11.4457 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 '/sessions/create',
      params: { email: users(:valaki).email, password: 'a' },
      headers: { "HTTP_REFERER": url_for(controller: :say, action: :hello) }
    assert_response :redirect
    assert_equal session[:user], users(:valaki).id
  end

  test "logout" do
    post '/sessions/create',
      params: { email: users(:valaki).email, password: 'a' },
      headers: { "HTTP_REFERER": url_for(controller: :say, action: :hello) }
    assert_equal session[:user], users(:valaki).id
    get '/sessions/destroy',
      headers: { 'HTTP_REFERER': url_for(controller: :say, action: :hello) }
    assert_nil session[:user]
    assert_response :redirect
  end
end
```

A `UsersControllerTest`-ben szereplő tesztesetek automatikusan generálódtak üres törzzsel, az `IngredientsControllerTest`-t pedig `scaffold` szkripttel hoztuk létre, ami automatikusan generálta a tesztadatokat, a tesz-

teseteket és a tesztesetek törzsét is. A felhasználók kontrollerének tesztjében az `:id`-t használó akciók esetén le kell futtatnunk a `login` tesztesetet, hogy a `session` megfelelően beállítódjék, hogy az alapján elő tudjuk keresni a bejelentkezett felhasználót. A tesztfüggvényekben módosítanunk kell az URL paramétert a `routes.rb` struktúrájának megfelelően.

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 regisztáció műveletének átadjuk a form paramétereit, és átirányítást várunk válaszként, továbbá azt, hogy a session inicializálódik, és három felhasználónk lesz az adatbázisban. Figyeljünk arra, hogy a paramétereknél a form paramétereiben szereplő pl. `user[email]` a `user` hashbeli email kulcsra képeződik le. A felhasználói profil szerkesztéséhez előzetesen be kell jelentkeznünk, 200-as HTTP válasz státusz kódot várunk. A felhasználói profil megtekintéséhez is be kel jelentkeznünk, és 200-as HTTP válasz státusz kódot várunk.

```
require 'test_helper'

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

  test "should get new" do
    get '/users/new'
    assert_select 'legend', 2
    assert_response :success
  end

  test 'registe a new user' do
    assert_equal User.count, 2
    post '/users/create',
      params: { user: { email: 'senki@mail.bme.hu', name: "Sen Ki", password
          : 'a', password_confirmation: 'a' } },
      headers: { "HTTP_REFERER": url_for(controller: :say, action: :hello) }
    assert_response :redirect
    assert_not_nil session[:user]
    assert_equal User.count, 3
  end

  test "should get edit" do
    get '/users/'+users(:valaki).id.to_s+'/edit'
    assert_response :success
  end

  test "should get show" do
    post '/sessions/create',
      params: { email: users(:valaki).email, password: 'a' },
      headers: { "HTTP_REFERER": url_for(controller: :say, action: :hello) }
    assert_equal session[:user], users(:valaki).id
    get '/users/'+@valaki.id.to_s
    assert_response :success
```

```
  end

  test "should_get_forgotten" do
    get '/users/forgotten'
    assert_response :success
  end

end
```

A felhasználók kontroller automatikus tesztjei hibákat mutattak ki, a `show` és `forgotten` útvonalak nem megfelelő sorrendben szerepeltek az útvonalak között, azokat fel kell cserélnünk.

A receptek kontrollerben a `recipe_params` függvényben lévő fehérlistába fel kell vennünk az utólag felvett felhasználóazonosító, sütésidő és nehézség paramétereket.

Az összetevők kontrollerben egyetlen elemet kell módosítanunk, az új összetevő létrehozását, amelyben a paraméterek közé fel kell vennünk annak a receptnek az azonosítóját, amelyhez az összetevő tartozik. Erre azért van szükség, mert ezt az attribútumot utólag adtuk hozzá. A tesztek futtatása ekkor hibát mutat a kontrollerben, ez az attribútum nem szerepel a `ingredient_params` függvény fehérlistájában.

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

```
kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rails test:
    controllers
Run options: --seed 29313

# Running:

........\t\t..........

Finished in 1.918544s, 10.4246 runs/s, 14.5944 assertions/s.

20 runs, 28 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/integration# rails g
    integration_test upload_recipe
      invoke  test_unit
      create    test/integration/upload_recipe_test.rb
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `upload_recipe_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:

- lekérdezünk egy képernyőt, ahol van bejelentkező form,
- kitöltjük a bejelentkezési formot, és rákattintunk bejelentkezés gombra,
- lekérdezzük a receptek listáját,
- a listából kiválaszunk egy elemet adatlapjának módosítását
- az elem adatlapján a fájlválasztó mezőben kiválaszunk egy fájlt, majd a többi mezőt változatlanul hagyva rákattintunk a frissítés gombra.

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 link HTML elem.

A második tesztlépés a bejelentkezés linkre „kattintás”. Az a feltételezünk, hogy az oldal ennek hatására átirányítódunk, továbbá a `session` vizsgálatával ellenőrizzük, hogy a felhasználó be van-e jelentkezve.

Lekérdezzük az `index` oldal első töredékét. Az a feltételezésünk, hogy az oldal sikeresen betöltődik.

Kiválasztuk a tesztadatok közül egy „véletlen” receptet, és betöltjük annak szerkesztő nézetét, itt is sikeres választ feltételezünk.

A szerkesztés nézeten található egy form, amelyen kereszül `name`, `cook_time`, `complexity`, `description` és `image` kérés paramétereket juttathatunk el az `update` akciónak HTTP PUT üzenettel. Az első négy paraméterhez a kiválasztott recept megfelelő mezőit rendeljük, az utolsó paraméternek pedig egy, a `fixture_file_upload` metódussal a tesztadatok közül, a helyi fájlrendszerről kiválasztott fájlt adunk meg. Hivatkozó oldalnak a szerkesző oldalt adjuk meg. Azt várjuk, hogy oda irányítódunk vissza, létrejön egy `Image` rekord az adatbázisban, annak `name` attribútuma megegyezik a feltöltött fájl eredeti nevével, és a fájl létrejön a fájlrendszeren a feltöltött képek tárolására kijelölt területen.

```
require 'test_helper'

class UploadRecipeTest < ActionDispatch::IntegrationTest
  test "upload an image for a recipe" do
    assert true

    get '/say/hello'
    assert_response :success
    assert_select 'form', 1
    assert_select 'legend', 'Login'
```

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

    get url_for(controller: :recipes, action: :index, page: 1)
    assert_response :success

    get '/recipes/'+recipes(:kenyer).id.to_s+'/edit'
    assert_response :success

    recipe = recipes(:kenyer)
    upload_file = fixture_file_upload(
      Rails.root.join('test', 'fixtures', 'files', 'rails_hello.png'),
      'image/png')
    put url_for(controller: :recipes, action: :update, id: recipe.id),
      params: {
        recipe: {
          name: recipe.name,
          description: recipe.description,
          cook_time: recipe.cook_time,
          complexity: recipe.complexity,
          image: upload_file
        }
      },
      headers: { 'HTTP_REFERER': url_for(contorller: :recipes, action: :edit
          , id: recipes(:kenyer).id) }
    image = Image.last
    assert_equal image.name, 'rails_hello.png'
    assert File.exist?(Rails.root.join('public', 'images', image.id.to_s))
  end
end
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/gyakorlat/test/integration# RAILS_ENV='test '
    rails test:integration
Run options: --seed 47549

# Running:

.

Finished in 1.335638s, 0.7487 runs/s, 7.4871 assertions/s.

1 runs, 10 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 teljesítményteszteket.

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 recipes
      create  test/performance/recipes_test.rb
```

Az alábbi teszteset az `hello` akciót 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 "homepage" do
    get '/say/hello'
  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 rails` célokkal hajthatunk végre.

```
kovacsg@debian:~/gyakorlat# RAILS_ENV='test' rails test:benchmark
Run options: --seed 46753

# Running:

RecipesTest#test_homepage (290 ms warmup)
```

```
            wall_time: 11 ms
               memory: unsupported
              objects: unsupported
              gc_runs: 0
              gc_time: 0 ms
.

Finished in 2.251747s, 0.4441 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 7885

# Running:

BenchmarkerTest#test_user_all (13 ms warmup)
            wall_time: 0 ms
               memory: unsupported
              objects: unsupported
              gc_runs: 0
              gc_time: 0 ms
.

Finished in 1.558614s, 0.6416 runs/s, 0.0000 assertions/s.

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