Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2014. április 29.

A gyakorlaton az előző gyakorlat végéről lemarad tördelés megvalósításával, illetve az előző gyakorlatok során elkészített megoldást tesztelésével foglalkozunk, illetve javítjuk az időközben elkövetett hibákat.

A tördelés célja, hogy a nagyszámú rekordot meg se próbáljuk jeleníteni egy oldalon, és próbáljuk kezelhetőnek megtartani az oldalunk függőleges méretét. Mivel ez gyakran előforduló használati eset, így létezik rá a megvalósítást támogató Rails függvénykönyvtár, a gyakorlaton a `will_paginate` függvénykönyvtárt használjuk, amit a `Gemfile`-ban helyezünk el, majd hajtsunk végre egy csomagfrissítést.

```
gem 'will_paginate'
```

Még mielőtt rátérünk a tördelésre, javítunk egy hibát. Sikerült korábban `type` néven attribútumot hozzáadnunk egy attribútumot a felhasználók modellhez. Készítsünk egy migrációt, ami ezt átnevezi `admin`-ra, és hajtsuk végre a migrációt.

```
class RenameUserType < ActiveRecord::Migration
  def up
    rename_column :users, :type, :admin
  end

  def down
    rename_column :users, :admin, :type
  end
end
```

Definiáljuk egy osztálymetódust a felhasználók modellben, amely megvalósítja a tördelést. Egy oldalon maximum 5 felhasználót engedjünk megjeleníteni.

```ruby
def self.get_user_page(page)
  User.all.paginate(:page=>page, :per_page=>5)
end
```

Az összes felhasználót megjelenítő kontroller akció az `index`. Ott az összes felhasználó helyett használjuk az aktuális oldalszámnak megfelelő tördelést az előbbi függvény meghívásával.

```ruby
def index
  @users = User.get_user_page(params[:page])#User.all
end
```

Az `index` nézeten mindössze annyi a teendőnk, hogy a tördelni kívánt lista, táblázat alá és/vagy felé beszúrjuk a tördelési HTTP paramétert beállító linkeket az oldalra kitevő függvényhívást. A paraméter neve a `param_name` szimbólumhoz, mint kulcshoz rendelt értékkel állítható. Ennek akkor van jelentősége, ha egy oldalon több tördelést is el akarunk végezni.

```
<%= will_paginate @users, :param_name=>'page' %>
```

Most térjünk át a laborunk témájára. 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 teszt adatokat 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 teszt eset 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 `scaffold`-dal 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 az `admin` és a `salt`.

```
me:
  username: kovacsg
  encrypted_password: <%= User.encrypt 'titok', 'valami
    ' %>
  first_name: Gabor
  family_name: Kovacs
  email: kovacsg@tmit.bme.hu
  last_login: 2014-03-18 13:16:16
  admin: 1
  salt: valami

two:
  username: two
  encrypted_password: <%= User.encrypt 'titok', 'valami
    ' %>
  first_name: MyString
  family_name: MyString
  email: two@mail.bme.hu
  last_login: 2014-03-18 13:16:16
  admin: 0
  salt: valami
```

A felhasználókhoz hasonló módon megadjuk két feladat tesztadatait is az `tasks. yml` fájlban. A feladatok és a felhasználók között több-egy reláció van, az idegen kulcsunk neve `user_id`, ami alapján a Rails ki tudja következteni, hogy az a `User` modellre hivatkozik. Ha a YAML fájlunkban egy ilyen kulcshoz a hivatkozott tesztadat egy kulcsát rendeljük, a Rails feloldja a kapcsolatot.

```
one:
  deadline: 2014-03-18 12:52:15
  title: Elso
  user: me
  state: 1
  description: Porszivozas

two:
  deadline: 2014-03-18 12:52:15
  title: Masodik
  user: me
  state: 1
```

```
  description: Mosogatas
```

A kommentek tesztadatait (`bids.yml`) is vegyük fel, és egyúttal használjuk ki a másik két tesztadatfájlban definiált kulcsokat az idegen kulcsok inicializációja céljából.

```
one:
  comment: MyText
  user: me
  task: one
  filename: MyString
  mime: MyString

two:
  comment: MyText
  user: me
  task: one
  filename: MyString
  mime: MyString
```

Töltsük be a teszt környezet adatbázisába ezeket az adatokat ügyelve arra, hogy a környezetként a teszt környezet legyen beállítva.

```
RAILS_ENV='test' rake db:test:prepare
RAILS_ENV='test' rake db:migrate
RAILS_ENV='test' rake db:fixtures:load
```

Ha MySQL konzolon megnézzük a betöltött adatokat, azt láthatjuk, hogy az `id` attribútum véletlen értékkel töltődött fel, az időpecsétek pedig a betöltés időpontját vették fel.

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, :uniqueness => true, :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. A másik tesztesetünk azt ellenőrzi, hogy a tesztadatok között megadott felhasználónév, jelszó páros segítségével be tudhatunk-e egyáltalán jelentkezni.

```
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  test "cannot save user without username" do
     u = User.new
     assert !u.save, "Houston"
  end

  test "cannot save user with existing username" do
     u = User.new
     u.username = 'kovacsg'
     assert !u.save, "we have a problem"
  end
end
```

Miután a webszerverünket újraindítottuk úgy, hogy az a tesztkörnyetetet használja, 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.

```
rake test:units
(in /home/kovacsg/gyakorlat)
Run options: --seed 44281

# Running tests:

[deprecated] I18n.enforce_available_locales will
   default to true in the future. If you really want to
    skip validation of your locale you can set I18n.
   enforce_available_locales = false to avoid this
   message.
..

Finished tests in 0.233792s, 8.5546 tests/s, 8.5546
   assertions/s.

2 tests, 2 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 teszteset.

Először egészítsük ki a `SessionsControllerTest` tesztet, amely jelenleg két tesztesetet tartalmaz. 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 paramétere a teszelendő akció, a második a HTTP kérés paraméterei, a harmadik pedig a kérés előtti `session` paraméterek értékeit tartalmazó hash. 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, tehát a második paraméter `nil`, azonban harmadik paraméterként beállítjuk a `:user` session paramétert azt imitálva, hogy van bejelentkezett felhasználónk. Válaszként átirányítást várunk az aktuális oldalunkra, és azt, hogy a session paraméter kinullázódik.

```ruby
require 'test_helper'

class SessionsControllerTest < ActionController::
   TestCase
  test "should get create" do
    post :create, { :username=>users(:me).username, :
       password=>'titok'}
    assert_response :redirect
    assert_not_nil session[:user]
  end

  test "should get destroy" do
    get :destroy, nil, { :user => users(:me).id }
    assert_response :redirect
    assert_nil session[:user]
  end
end
```

A tesztesetünk kész van azonban nem futhat le sikeresen ugyanis a kontrollerben szereplő `redirect_to :back` átirányításban nem definiált a `:back` értéke. Ezt a Rails a javascript history elérhetetlensége miatt a `HTTP_REFERER` nevű HTTP kérés paraméterből veszi, így ezt minden egyes tesztesetben be kell állítanunk. Ezt hatékonyan a `setup` metódusban tehetjük meg, amely minden teszteset előtt lefut.

```
class SessionsControllerTest < ActionController::
    TestCase
  setup do
    @request.env["HTTP_REFERER"]='/say/hello'
  end
end
```

A `UsersControllerTest`-et és a `TasksControllerTest`-et scaffolddal hoztuk létre, amely automatikusan generálta a teszteseteket és tesztadatokat. A tesztadatok kulcsait módosítottuk, így az azokra való hivatkozást frissítenünk kell a `setup`-ban. Mivel a műveleteink jó része csakis bejelentkezett felhasználó számára lehetséges, e két tesztfájlban a `get` és `post` műveletek harmadik paraméterében inicializálunk kell a `sessions` hasht a harmadik paraméterben. Az alábbi kódrészlet erre mutat egy példát, azonban ezt a kontrollerekben szereplő `before_filter` miatt minden akcióban el kell végeznünk, ráadásul ez a művelet csak az adminisztrátor típusú felhasználók számára lehet elérhető, így egyszerre mindkét tesztadatunkat kihasználjuk.

```
class UsersControllerTest < ActionController::TestCase
  setup do
    @user = users(:two)
  end

  test "should get index" do
    get :index, nil, user: users(:me).id
    assert_response :success
    assert_not_nil assigns(:users)
  end
end
```

A felhasználói profil szerkesztésének tesztelése trükkösebb, sok ezzel kapcsolatos hibát követtünk el. Először is mindjárt hibás linkünk a menüben, amit a `_loginfrom.html.erb`-ben orvosolhatunk. Arra kell figyelnünk, hogy az azonosító egész típusú értékét stringgé kell konvertálnunk.

```
<%= link_to "View profile", '/users/'+session[:user].
    to_s %>
```

A szerkesztés nézetre nem csak menüből kerülhetünk, hanem a profiloldalról is. Ha adminisztrátorok vagyunk, akkor az adminisztrátori szerkesztés nézetre kell kerülnünk, ha felhasználók, akkor a felhasználóira. Ezért a `show.html.erb`-ben elhelyezünk egy feltételes kifejezést.

```
<% if is_admin? %>
<%= link_to 'Edit', edit_user_path(@user) %>
<% else %>
<%= link_to 'Edit', '/users/'+@user.id.to_s+'/edit' %>
<% end %>
```

Az `is_admin?` egy helper metódus, amit alapvetően nézetekben használhatunk fel, de a kontrollerben is hasznos lehet, így tegyük ott is elérhetővé. Az összes kontroller ősosztályában hivatkozunk az alkalmazásszintű helper modulunkra, amely így az összes kontrollerben elérhetővé válik.

```
 class ApplicationController < ActionController::Base
  include ApplicationHelper
end
```

Ezután szebbé tehetjük a felhasználók kontrollerében definiált szűrőmetódusainkat is. A bejelentkezett nem adminisztrátor felhasználók esetén elkövettünk még egy apró hibát, a `params` hashből string típusú értékeket bányászhatunk elő, így annak a session azonosítóval való összevetése előtt azt egész típusúvá kell konvertálnunk.

```
def check_admin
  if !is_admin?
    redirect_to url_for(:controller=>:tasks, :action =>
        :index)
  end
end

def check_the_same_user
  if !is_admin?
    if session[:user] != params[:id].to_i
      redirect_to url_for(:controller=>:tasks, :action
          => :index)
    end
  end
end
```

A felhasználókezelés adminisztátorra és felhasználóra való szétválasztása miatt még egy utolsó trükköt is el kell követnünk az eseményeket előállító formban, hogy az a megfelelő eseménykezelőre vonatkozzék. Ezt a felhasználók nézetének formjában lévő `form_for` helperhez rendelt második paraméter hashéhez az `:url` kulcshoz rendelt érték hozzáadásával tesszük meg. Ha új rekordról van szó, amit a `new_record?` metódussal ellenőrzihetünk, akkor a kérést csakis az adminisztrátor követhette el, így az `/admin/users/new` oldalra kell átkerülnünk. Ha létező rekordról van szó, akkor függően attól, hogy adminisztrátor-e a felhasználó (`is_admin?`), a `/users/:id/edit` vagy a `/admin/users/:id/edit` nézetre mászunk át.

```
<%= form_for(@user, {:url=>(@user.new_record? ?
   new_user_path(@user):(is_admin? ?edit_user_path(
   @user):
'/users/'+@user.id.to_s+'/edit'))}) do |f| %>
```

Most ezzel minden saját magunk által okozott hibát elhárítottunk, megnézhetjük a felhasználók tesztjének többi tesztesetét. A `UsersControllerTest` osztályunk az `edit`, `show` és `new` akciókat ellenőrző teszteseteiben a `get` második paramétereként be kell állítanunk az `id` paramétert a tesztadatok alapján vagy a harmadik paraméterrel azt szimuláljuk, hogy a felhasználó már bejelentkezett. Minden mást az egyszerűség kedvéért változatlanul hagyunk.

```
class UsersControllerTest < ActionController::TestCase
  test "should get new" do
    get :new, nil, user:users(:me).id
    assert_response :success
  end

  test "should show user" do
    get :show, {id: @user.id}, {user: @user.id}
    assert_response :success
  end

  test "should get edit" do
    get :edit, {id: @user.id}, {user: @user.id}.
    assert_response :success
  end
end
```

A form eseménykezelőiben, vagyis a `create` és az `update` akciók teszjeiben, illetve a felhasználók törlésekor mind az aktuális felhasználó azonosait, mind a session paramétert meg kell adnunk. A form paramétereinél figyeljünk arra, hogy már létező felhasználóazonosító nem használható fel újra, és, hogy ne feleljtsük el megadni a jelszó megerősítését sem.

```
class UsersControllerTest < ActionController::TestCase
```

```
  test "should create user" do
    assert_difference('User.count') do
      post :create, {user: { email: 'valakimas@mail.bme
          .hu',
          family_name: 'Elek', first_name: 'Teszt',
              last_login:
          @user.last_login, password: 'titok',
              password_confirmation:
          'titok', username: 'tesztelek' }}, {user:
              users(:me).id}
    end

    assert_redirected_to user_path(assigns(:user))
  end

  test "should update user" do
    patch :update, {id: @user.id, user: { email: @user.
        email,
        family_name: @user.family_name, first_name:
            @user.first_name,
        last_login: @user.last_login, password: @user.
            password,
        password_confirmation: @user.password,
        username: @user.username+'_' }}, {user: @user.
            id}
    assert_redirected_to user_path(assigns(:user))
  end

  test "should destroy user" do
    assert_difference('User.count', -1) do
      delete :destroy, {id: @user.id}, {user: users(:me
          ).id}
    end

    assert_redirected_to users_path
  end
end
```

A funkcionális teszteseteket futtatva láthatjuk, hogy sikeresen lefutnak.

```
rake test:functionals
(in /home/kovacsg/gyakorlat)
```

```
Run options: --seed 24003

# Running tests:

......[deprecated] I18n.enforce_available_locales will
   default to true in the future. If you really want to
    skip validation of your locale you can set I18n.
   enforce_available_locales = false to avoid this
   message.
............

Finished tests in 0.748839s, 24.0372 tests/s, 42.7328
   assertions/s.

18 tests, 32 assertions, 0 failures, 0 errors, 0 skips
```

A tesztek harmadik típusa az integrációs teszt, amellyel egy böngészési folyamatot ellenőrzünk. Készítsünk egy tesztet a felhasználói profilkép feltöltésének esetére!

```
rails generate integration_test  upload_koszos_sarok
```

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

A fájl elején töltsük be az összes teszt adatot `fixtures :all`, majd definiáljuk a böngészési folyamatot úgy, hogy az a bejelentkezéstől jusson el legalább egy kép típusú komment feltöltéséig. Ne feledkezzünk el a karakterkódolás beállításáról sem, ha ékezetes karaktert használunk!

Az integrációs tesztesetet a funkcionális teszteknél megismert tesztesetekből mint tesztlépésekből tevődik össze.

Az első lépésben egy be nem jelentkezett felhasználó betölt egy véletlen oldalt, amin elérhető a bejelentkezési form. Itt csak annyit feltételezünk, hogy az oldal betöltődik, és azon van egy `Username` feliratú címke. Ez utóbbit az `assert_select` metódussal ellenőrizzük annak egy CSS selector kifejezést átadja.

A második tesztlépés a bejelentkezés. Az aktuálisan megjelenített nézeten található egy form, amelyen keresztül `username` és `password` kérés paramétereket juttathatunk el a `/sessions/create` kontroller akciónak. Mivel az akció végén minden esetben egy átirányítás áll, `post` helyett a `post_via_redirect` metódust kell használnunk, ami a kérés után követi az összes átirányítást. Ennek a metódusnak a paraméterlistája eltér a megszokottól, vagyis az akció, paraméterekek, session, flash négyestől, itt a második paraméter ugyanúgy a HTTP kérés paraméterlistája, azonban a harmadik paraméter a HTTP fejrész opciók hashe. Mivel az átirányítás az előző oldalra történik a vizsgálat akcióban, a harmadik paraméterben kell megadnunk a `HTTP_REFERER` fejrész opcióval, hogy mely oldal volt az előző oldal. A feltételezésünk az, hogy a sessionünk inicializálódik, amit a session hash vizsgálatával tehetünk meg.

A harmadik tesztlépés megnézi annak a feladatnak az oldalát, amelyhez a képet szeretnénk feltölteni. Sikeres HTTP válasz az elvárásunk.

A negyedik tesztlépésben előhalásszuk az előző tesztfájlunkat. A fájlfeltöltés műveletének tesztelésére töltsünk be a memóriába egy valóságos állományt a `fixture_upload_file` metódussal. Ez a metódus a tesztadatok könyvtárához képesti relatív útvonalat vár első argumentumként, a második, opcionális argumentuma az ott található állomány MIME típusa. Hozzunk létre egy tetszőleges szöveget tartalmazó fájlt a `test/fixturesfiles` könyvtárban például `html_dom.png` néven. A tesztesetünkben három feltételezéssel élünk, egyrészt ellenőrizzük, hogy a komment megjelent-e az adatbázisban, másrészt pedig jó lenne, ha a fájl valóban megjelenne a fájlrendszeren.

```
require 'test_helper'

class UploadKoszosSarokTest < ActionDispatch::
   IntegrationTest
  fixtures :all
  test "upload picture" do
    get url_for(:controller=>'tasks', :action => :index
       )
    assert_response :success
    assert_select 'label', "Username"

    post_via_redirect '/sessions/create', {:username=>
       users(:me).username, :password=>'titok'},
      { 'HTTP_REFERER'=>'/say/hello' }
    assert_equal session[:user], users(:me).id

    get task_url(tasks(:two))
    assert_response :success

    upload_file = fixture_file_upload('test/fixtures/
       files/html_dom.png','image/png')
    post url_for(:controller=>'comments', :action=>'
       upload', :id=>tasks(:two).id),
```

```
      { file: upload_file },
      { 'HTTP_REFERER'=>'/say/hello' }
    c = Comment.where(task_id: tasks(:two).id, user_id:
        users(:me).id)
    assert_not_nil c
    assert File.exists?('public/data/html_dom.png')
  end
end
```

Futtatva az integrációs tesztet láthatjuk, hogy az egy teszteset négy tesztlépése során mind a hat ellenőrzésen sikeresen átment.

```
rake test:integration
(in /home/kovacsg/gyakorlat)
Run options: --seed 363

# Running tests:


.

Finished tests in 0.622700s, 1.6059 tests/s, 9.6355
   assertions/s.

1 tests, 6 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 teljesítményteszteknek 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 általános statisztikát közöl a portál a teszteset által ellenőrzött részéről, a profile pedig a szűk keresztmetszetet jelentő pontokat próbálja azonosítani.

A teljesítményteszteket akárcsak az integrációs teszeket explicit módon kell létrehoznunk:

```
rails generate performance_test browsing
```

Az alábbi teszteset az index nézet támadja meg 5 egymás utáni kéréssel.

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

class BrowsingTest < 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:profile` és a `test:benchmark` `rake` célokkal hajthatunk végre.

```
rake test:profile
(in /home/kovacsg/gyakorlat)
Run options: --seed 40348

# Running tests:

BrowsingTest#test_homepage (140 ms warmup)
        process_time: 20 ms
              memory: 0 Bytes
             objects: 0


Finished tests in 4.390694s, 0.2278 tests/s, 0.0000
   assertions/s.

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

```
rake test:benchmark
(in /home/kovacsg/gyakorlat)
Run options: --seed 30560

# Running tests:

BrowsingTest#test_homepage (133 ms warmup)
           wall_time: 6 ms
              memory: unsupported
             objects: unsupported
             gc_runs: 0
             gc_time: 0 ms


Finished tests in 0.665908s, 1.5017 tests/s, 0.0000
   assertions/s.

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