Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2010. november 26.

Ez a segédlet az elmaradt gyakorlat anyagát tartalmazza. A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk.

A tesztelés előkészítésére először teszt adatokat definiálunk, amelyekre a teszt eseteinkben 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. Először definiáljuk a titkosításhoz használt `salt` attribútum értékét, amelyre minden egyes felhasználó példányában hivatkozni fogunk. A jelszó megadához a modell osztály `encrypt` metódusát hívjuk segítségül.

```
<% SALT= Digest::SHA1.hexdigest("#{Time.now}--HELLO")
   %>
me:
    neptun: oweyoa
    email: kovacsg@tmit.bme.hu
    student: false
    salt: <%=SALT%>
    encrypted_password: <%= User.encrypt 'haho', SALT %>
b:
    neptun: bbbbbb
    email: b@server.com
    student: true
    salt: <%=SALT%>
    encrypted_password: <%= User.encrypt 'hello', SALT
      %>
```

A felhasználókhoz hasonló módon két már kiadott feladatot is megadunk a `tasks.yml` fájlban.

```
one:
  number: 1
  url: http://server.com/task1
  deadline: 2010-09-30 23:59:59

two:
  number: 2
  url: http://server.com/task2
  deadline: 2010-10-14 23:59:59
```

Töltsük be a teszt adatbázisba ezeket az adatokat

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

A fájlfeltöltés tesztelésére szükségünk lesz egy feltölthető fájlra. Szintén a `fixtures` könyvtárban létrehozunk egy `files` alkönyvtárat, és ott elhelyezünk egy tetszőleges fájlt. A gyakorlaton ez a fájl az `1.txt` nevű szöveges állomány lesz.

Először egy egységtesztet írunk. Az egységtesztek a modell osztályok metódusait és validációt hivatottak ellenőrizni. Az egyégteszteket a Rails projektünk `test/unit` könyvtárában találjuk. Minden egyes modell létrehozása után automatikusan létrejön hozzá egy ahhoz kapcsolódó teszt osztály itt.

Írjunk egy tesztet, amely a `User` modellünk

```
validates_presence_of :neptun
```

validációjának megtörténtét ellenőrzi. Először létrehuzunk egy új felhasználót mindenféle inicializáció nélkül, majd megpróbáljuk eltárolni az adatbázisba. A feltételezésünk az, hogy a mentésnek nem szabad sikerülnie.

```
test "neptun␣null" do
  u = User.new
  assert !u.save, "User␣with␣no␣neptun␣code␣saved"
end
```

A teszt esetet futtatva meggyőződhetünk róla, hogy tényleg nem történik meg a mentés.

```
$ ruby  user_test.rb -n test_neptun_null
```

```
Loaded suite user_test
Started
.
Finished in 0.635653 seconds.

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

Test run options: --seed 43096 --name "test_neptun_null
   "
```

A funkcionális tesztek a kontrollerek és a nézetek helyes működését ellenőrzik. Vizsgáljuk meg a scaffolddal létrehozott `Tasks` kontroller egyik automatikusan generált teszt esetét. A teszt eset azt állítja, hogy a `Tasks` kontroller `new` akcióját HTTP GET metódussal lekérve sikeres HTTP státusszal kapunk választ.

```
test "should get new" do
  get :new
  assert_response :success
end
```

A teszt esetet futtatva láthatjuk, hogy ez valóban így történik.

```
$ ruby tasks_controller_test.rb -n test_should_get_new
Loaded suite tasks_controller_test
Started
.
Finished in 0.528928 seconds.

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

Test run options: --seed 2494 --name "
   test_should_get_new"
```

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

```
rails generate integration_test task_submission
```

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

A fájl elején töltsük be az összes teszt adatot, majd definiáljuk a böngészési folyamatot úgy, hogy az a bejelentkezéstől jusson el egészen a sikeres

feladatbeadásig.

A teszt metódusunk első teszt lépése a bejelentkezési képernyő lekérése HTTP GET-tel, amely a `sessions` kontroller `new` akciójához kapcsolódó nézet. Ezt akkor tekintjük sikeresnek, ha a HTTP válasz státusz kódja sikeres.

A második teszt lépésünk az oldalon található form kitöltése és elküldése HTTP POST metódussal a `create` kontroller akciónak. Ezt a `post_via_redirect` metódussal tesszük meg, amelynek első paramétere a kontroller akció, a további paraméterek pedig az egyes form mezők. A neptun kódot a `users` teszt adathalmaz `:me` kulccsal azonosított példányából vesszük. Mivel abban a példányban nem adtunk meg `password` attribútumot, itt most stringként adjuk azt meg. A bejelentkezést akkor tekintjük sikeresnek, ha a következő oldal a felhasználó saját oldalát mutatja.

A harmadik és negyedik teszt lépéssel átnavigálunk a feladatok listája oldalra, ami a `tasks` kontroller `index` nézete, majd abból a listából kiválasztjuk a sorban első feladat mellett látható `Bead` linket. Ehhez felhasználjuk a `tasks` teszt adathalmaz `:one` kulccsal rendelkező elemének azonosítóját. Mindkét lépést akkor tekintjük sikeresnek, ha a HTTP válasz státusz kódja 200 volt.

Az ötödik és egyben utolsó teszt lépés a fájlfeltöltést ellenőrzi. Először a teszt adatok között eltárolt fájl alapján a `fixture_file_upload` metódussal létrehozunk egy fájl objektumot, amelyet fel fogunk tölteni. A fetöltést a `post` metódussal végezzük el, amit a `solutions` kontroller, `new` akciójának címzünk a `tasks` teszt adat `:one` kulccsal reprezentált példányának adatbázisbeli azonosítójával mint `:id`-vel. A `:upload` nevű form `:file` mezőjéhez hozzárendeljük az imént létrehozott fájl objektumot. A sikeresség ellenőrzése végett megnézzük, hogy a `:me` kulcsú `users` teszt adathoz és a `:one` kulcsú `tasks` teszt adathoz tartozik-e adatbázisbeli rekord. Akkor sikeres a teszt, ha a rekordban a fájl neve `1.txt` lett és a webszerver könyvárában megjelent maga a feltöltött állomány.

```
require File.dirname(__FILE__)+'/../test_helper'

class TaskSubmissionTest < ActionDispatch::
   IntegrationTest
  fixtures :all

  # Replace this with your real tests.
  test "task_submission" do
    u = users(:me)
    get url_for(:controller=>'sessions',:action=>'new')
    assert_response :success
    post_via_redirect "/sessions/create",
```

```
        :neptun=>u.neptun,
        :password=>'haho'
    assert_equal '/users/show/'+u.id.to_s,path
    get '/tasks'
    assert_response :success
    get '/solutions/new/'+tasks(:one).id.to_s
    assert :success
    upload_file = fixture_file_upload('files/1.txt','
        text/plain')
    post url_for(:controller=>'solutions',:action=>'new
        ',
        :id=>tasks(:one).id.to_s),
        :upload => {:file => upload_file}
    s = Solution.find_by_task_id_and_user_id tasks(:one
        ).id, users(:me).id
    assert_not_nil s.file_name
    assert File.exists?('../../public/data/1.txt')
  end
end
```

Futtatva az integrációs tesztet láthatjuk, hogy az egy teszt eset öt teszt lépése során mind a hat ellenőrzésen sikeresen átment.

```
$ ruby task_submission_test.rb
Loaded suite task_submission_test
Started
.
Finished in 0.619282 seconds.

1 tests, 6 assertions, 0 failures, 0 errors, 0 skips

Test run options: --seed 8213
```