Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2011. április 26.

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: aaaaaaa
    email: a@bme.hu
    student: true
    salt: <%=SALT%>
    encrypted_password: <%= User.encrypt 'hello', SALT
      %>
```

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

```
first:
    number: 1
    url: https://twiki.db.bme.hu/twiki/bin/view/Student
       /Ruby/RubyElsoFeladat20111
    deadline: 2011-02-28 23:59:59

second:
  number: 2
  url:https://twiki.db.bme.hu/twiki/bin/view/Student/
     Ruby/RubyMasodikFeladat20111
  deadline: 2011-03-14 23:59:59
```

A megoldások tesztadataként (`solutions.yml`) a következő rekordot használjuk. Az idegen kulcsoknál a megfelelő tesztadatfájl egy kulcsát használjuk, ami jelen esetben `first` a `task.yml`-t tekintve, és `student` a `user.yml`-re nézve. Ennek hatására a `task_id` és a `user_id` attribútum automatikusan állítódik.

```
egy:
   task: first
   user: student
   late: false
   accepted: true
```

Töltsük be a teszt adatbázisba 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
```

Először egy egységtesztet írunk. Az egységtesztek a modell osztályok metódusait és validációit 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
   "
```

Végezzük el egy másik validáció ellenőrzését is, ellenőrizzük, hogy a felhasználó által a regisztáricó során megadott neptun kód valóban csak számokból és betűkből áll-e. Először felvesszük a modell osztályba ez a validációt.

```
validates_format_of :neptun, :with => /[0-9a-zA-Z]{6}/
```

A teszteset, amely ezt a validációt ellenőrzi, az alábbi. A helyességet a tesztadatok módosításával tudjuk ellenőrizni.

```
  test "neptun_format" do
    u = users(:me)
    assert u.neptun.match /[0-9a-zA-Z]{6}/
  end
```

A funkcionális tesztek a kontrollerek és a nézetek helyes működését ellenőrzik. Alapértelmezés szerint minden egyes a létrehozáskor megadott kontroller akcióra létrejön egy az akció sikeres megjelenítését ellenőrző tesztteset. Ezeket most ne használjuk, kommentezzük ki őket. Egészítsük ki a `SessionsControllerTest` tesztesetét még egy feltételezéssel. Állítsuk azt, hogy a `/session/new` útvonalat lekérve a HTML DOM H1-es elemének értéke „Bejelenkezés”.

```
test "should get new" do
  get :new
  assert_response :success
  assert_select "h1", "Bejelentkezés"
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 feladat értékelés esetére!

```
rails generate integration_test correct_a_task
```

E parancs kiadása után a `test/integration` könyvtárban létrejött egy `correct_a_task_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. Ne feledkezzünk el a karakterkódolás beállításáról sem.

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, és az oldal HTML DOM-jában a H1 elem értéke „Bejelentkezés”.

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. A `password` értékének az ott megjelölt „haho” stringet adjuk meg. A bejelentkezést akkor tekintjük sikeresnek, ha a `session` hash `user` kulcsú értéke a felhasználó azonosítóját mutatja, és a HTML DOM H2 elemének értéke a felhasználó neptun kódja.

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 `Javít` linket. Az előbbi akkor sikeres, ha van ilyen értékkel rendelkező HTML DOM A elemünk. Az utóbbihoz felhasználjuk a `tasks` teszt adathalmaz `:first` kulccsal rendelkező elemének azonosítóját. Ezt a lépést akkor tekintjük sikeresnek, ha a HTML DOM h3-as elemének értéke „1. feladat”, és az oldalon található egyetlen link a `student` kulccsal hivatkozott felhasználó neptun kódját és a helyes feladat sorszámot tartalmazza.

Az ötödik teszt lépés átnavigál a feladat értékelése oldalra, ahol a feladat azonosítója, illetve a hallgató azonosítója a teszt adatok közül kerül kiválasztásra. A lépés sikeres, ha a HTTP státusz kód 2xx.

A hatodik és egyben utolsó teszt lépés az értékelés oldalon található formot kezelő kontroller akciót vizsgálja. A paraméterlista tartalmazza a `:id`, és a `solution` kulcsot, amely önmaga is egy hash. A késés mező értéke 0, az elfogadott mező értéke 1, a komment „OK”. A teszt lépés akkor sikeres, ha a javítandó feladatok között ezután nem jelenik meg ez a megoldás lévén elfogadott lett a státusza.

```
#Encoding: UTF-8
require 'test_helper'

class CorrectATaskTest < ActionDispatch::
   IntegrationTest
  fixtures :all

  test "correct a task" do
    get url_for(:controller=>"sessions", :action=>"new"
       )
    assert_response :success
    assert_select "h1", "Bejelentkezés"
    post_via_redirect '/sessions/create', :neptun=>'
       oweyoa', :password=>'haho'
    assert_equal users(:me).id, session[:user]
    assert_select "h2", users(:me).neptun
    get url_for(:controller=>"task", :action=>"index")
    assert_select "table" do
      assert_select "tr", 1..6
    end
    get '/task'
    assert_select "td", "Javít"
```

```
    get '/solutions/show?task_id='+tasks(:first).number
      .to_s
    assert_select "h3", "1. feladat"
    assert_select "a", "aaaaaa  által beadott 1.
      feladat"
    get url_for(:controller=>"solutions",:action=>"edit
      ", :task_id=>tasks(:first).id, :user_id=>users(:
      student).id)
    assert_response :success
    post_via_redirect '/solutions/update/'+solutions(:
      egy).id.to_s, {:solution => {:late=>0, :accepted
      =>1, :comment=>"OK"}}
    assert_select "li", 0
  end
end
```

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

```
Test run options: --seed 62956
Loaded suite /var/lib/gems/1.9.1/gems/rake-0.8.7/lib/
   rake/rake_test_loader
Started
Finished in 0.377281 seconds.

1 tests, 12 assertions, 0 failures, 0 errors, 0 skips

Test run options: --seed 13724
```