Tesztelés Rails-ben
Gyakorlat

Kovács Gábor

2022. május 10.

1. Tesztelés Railsben

Laborunk fő 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.

Lőjük le a fejlesztői üzemmódban futó webszerverünket, és indítsuk újra teszt üzemmódban a 3001-es porton, 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 -p3001
```

2. Tesztadatok felvétele, betöltése

A tesztelés előkészítésére először tesztadatokat definiálunk, amelyekre a teszteseteinkben hivatkozni fogunk. Ezeket a Rails alkalmazásunk `test/fixtures` könyvtárában helyezzük el. A teszt szkriptek 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 néhány `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` osztálymetódusának törzsét használjuk, hiszen a szoftver akkor fog a teszteknek megfelelően működni, ha a teszteknek megfelelő titkosítást használja. 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`. Az `id` és az időpecsét attribútumokat nem kell definiálnunk, azok automatikusan töltődnek, az `id` random értékkel, az időpecsét pedig a pillanatnyi idővel, azonban a felhasználó és a cím közötti polimorfikus kapcsolat miatt meg kell adnunk.

```
citizen:
  name: Varos Lako
  email: varos.lako@magyarorszag.hu
  encrypted_password: <%= Digest::SHA1.hexdigest('titoka') %>
  salt: a
  idnum: AAAAAA22
  birthdate: 2022-03-22 13:18:50
  account: 11111111-11111111
  id: 22
```

E fájlokba Ruby kódrészlet ágyazható be a nézeteknél megismert módon, így tesztadatok tömeges gyártása egy ciklussal megoldható.

Először a pártlisták tesztadait definiáljuk. Itt nem volt változás a struktúrában, csak az adatokat módosítjuk.

```
atesz-ptnp:
  name: ATESZ-PTNP

ketsegben:
  name: Ketsegben Magyarorszagert

mihasznak:
  name: Mihasznak
```

A tesztadatbázisunkban szükségünk lesz pártokra, ennek az adatait `parties.yml` fájlban definiáljuk. A modellek generálása után hozzáadtunk a modellhez egy, a pártlistára vonatkozó idegen kulcsot, azzal bővítjük az adatleírást. A `list` attribútum a pártlisták YAML fájlban definiált kulcsokat vehet fel, és ez automatikusan létrehozza a kapcsolatot. A Rails template beágyazó módszerekkel egyszerre több pártot is létrehozhatunk.

```
atesz:
  name: Agg Teokratak Szovetsege
  abbreviation: ATESZ
  account: 11111111-11111112
```

```
  active: true
  list: atesz-ptnp

tk:
  name: Teokratikus Koalicio
  abbreviation: TK
  account: <%= '11111111-11111113' %>
  active: true
  list: ketsegben

mh:
  name: Mihasznak
  abbreviation: MH
  account: 11111111-11111114
  active: true
  list: mihasznak

<% for i in 1..4 %>
maradek<%= i%>:
  name: Tormekek<%=i %>
  abbreviation: T<%=i%>
  account: 11111111-1111112<%=i%>
  active: true
  list: ketsegben
<% end %>
```

Az egyes pártok jelöltjei a `candidates.yml` fájlba kerülnek. A pártra a párt YAML fájlban lévő kulccsal hivatkozhatunk.

```
one:
  name: I. Szent Viktor
  party: atesz

two:
  name: Frank Underwood
  party: tk
```

Az egyes jelöltek körzetekben indulnak, amelyeket a `districts.yml` fájlban definiálhatunk. Itt is definiáljuk az `id` attribútum értékét.

```
muegyetem:
  name: Muegyetem
  id: 1117
```

A felhasználókhoz, illetve a körzetekhez polimorfikus módon kapcsolódik a cím modell (`districts.yml`). Az `addr` polimofikus attribútumot lecseréljük egy `addr_type` és egy `addr_id` attribútumra.

```
citizen_address:
  city: Budapest
```

```
  street: Bogdanffy O. 3
  zip: 1117
  lat: 47
  lng: 19
  addr_type: User
  addr_id: 22

district_address:
  city: Budapest
  street: "Magyar tudosok utja 2."
  zip: 1117
  lat: 47
  lng: 19
  addr_type: District
  addr_id: 1117
```

Logo egyelőre nincsenek, ezért az `logos.yml` fájlból töröljük az adatokat.

Töröljük, majd hozzuk újra létre a tesztadatbázist, utána 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' rails
    db:drop
Dropped database 'gyakorlat_test'
kovacsg@debian:~/gyakorlat/test/fixtures> RAILS_ENV='test' rails
    db:create
Created database 'gyakorlat_test'
kovacsg@debian:~/gyakorlat/test/fixtures> RAILS_ENV='test' rails
    db:migrate
== 20220322121850 CreateUsers: migrating
    =============================================
-- create_table(:users)
   -> 0.0061s
== 20220322121850 CreateUsers: migrated (0.0063s)
    ====================================

== 20220322123406 CreateParties: migrating
    =============================================
-- create_table(:parties)
   -> 0.0056s
== 20220322123406 CreateParties: migrated (0.0056s)
    ====================================

== 20220329112120 CreateCandidates: migrating
    =============================================
-- create_table(:candidates)
   -> 0.0157s
== 20220329112120 CreateCandidates: migrated (0.0158s)
    ====================================
```

```
== 20220412102119 AddSaltToUsers: migrating ===================================
-- add_column(:users, :salt, :string)
   -> 0.0033s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0028s
== 20220412102119 AddSaltToUsers: migrated (0.0063s) ==========================

== 20220412112448 CreateLists: migrating ======================================
-- create_table(:lists)
   -> 0.0076s
== 20220412112448 CreateLists: migrated (0.0077s) =============================

== 20220412112535 AddListToParties: migrating =================================
-- add_reference(:parties, :list, {:foreign_key=>true})
   -> 0.0194s
== 20220412112535 AddListToParties: migrated (0.0196s) ========================

== 20220412113757 CreateAddresses: migrating ==================================
-- create_table(:addresses)
   -> 0.0085s
== 20220412113757 CreateAddresses: migrated (0.0086s) =========================

== 20220412113906 CreateDistricts: migrating ==================================
-- create_table(:districts)
   -> 0.0049s
== 20220412113906 CreateDistricts: migrated (0.0049s) =========================

== 20220503104847 CreateLogos: migrating ======================================
-- create_table(:logos)
   -> 0.0070s
== 20220503104847 CreateLogos: migrated (0.0071s) =============================

== 20220503112041 CreateVotes: migrating ======================================
-- create_table(:votes)
   -> 0.0138s
```

```
== 20220503112041 CreateVotes: migrated (0.0139s)
   ==============================

kovacsg@debian:~/gyakorlat/test/fixtures> 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
MariaDB [gyakorlat_test]> select * from users;
+----+------------+-------------------------+----------------------------------
| id | name       | email                   | encrypted_password
                        | idnum    | birthdate           | account
              | created_at                  | updated_at                  |
    salt |
+----+------------+-------------------------+----------------------------------

| 22 | Varos Lako | varos.lako@magyarorszag.hu |
    acc7912d641fab211dc622a39b788da34eadc0ee | AAAAAA22 | 2022-03-22
    13:18:50.000000 | 11111111-11111111 | 2022-05-11 11:36:02.738457 |
    2022-05-11 11:36:02.738457 | a    |
+----+------------+-------------------------+----------------------------------

1 row in set (0.000 sec)

MariaDB [gyakorlat_test]> select * from parties;
+-----------+-------------------------+--------------+-----------------------
| id        | name                    | abbreviation | account
     active | created_at                 | updated_at                 |
    list_id    |
+-----------+-------------------------+--------------+-----------------------

|  205128879 | Tormekek1               | T1           | 11111111-11111121 |
          1 | 2022-05-11 11:36:02.737376 | 2022-05-11 11:36:02.737376 |
    995480875 |
|  283776736 | Teokratikus Koalicio    | TK           | 11111111-11111113 |
          1 | 2022-05-11 11:36:02.737376 | 2022-05-11 11:36:02.737376 |
    995480875 |
|  316927618 | Mihasznak               | MH           | 11111111-11111114 |
          1 | 2022-05-11 11:36:02.737376 | 2022-05-11 11:36:02.737376 |
    558877005 |
|  355685655 | Tormekek2               | T2           | 11111111-11111122 |
          1 | 2022-05-11 11:36:02.737376 | 2022-05-11 11:36:02.737376 |
    995480875 |
|  573859200 | Tormekek3               | T3           | 11111111-11111123 |
          1 | 2022-05-11 11:36:02.737376 | 2022-05-11 11:36:02.737376 |
    995480875 |
| 1011937317 | Tormekek4               | T4           | 11111111-11111124 |
          1 | 2022-05-11 11:36:02.737376 | 2022-05-11 11:36:02.737376 |
    995480875 |
| 1022567928 | Agg Teokratak Szovetsege | ATESZ        | 11111111-11111112 |
          1 | 2022-05-11 11:36:02.737376 | 2022-05-11 11:36:02.737376 |
    893346719 |
+-----------+-------------------------+--------------+-----------------------
```

```
7 rows in set (0.000 sec)

MariaDB [gyakorlat_test]> select * from candidates;
+-----------+-----------------+----------+----------------------------+
| id        | name            | party_id | created_at                 |
  updated_at                 |
+-----------+-----------------+----------+----------------------------+
| 298486374 | Frank Underwood |  283776736 | 2022-05-11 11:36:02.731622 |
  2022-05-11 11:36:02.731622 |
| 980190962 | I. Szent Viktor | 1022567928 | 2022-05-11 11:36:02.731622 |
  2022-05-11 11:36:02.731622 |
+-----------+-----------------+----------+----------------------------+

2 rows in set (0.000 sec)

MariaDB [gyakorlat_test]> Bye
```

3. Modellek tesztelése

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
  validates :email, { 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.

```
class UserTest < ActiveSupport::TestCase
  test "the_truth" do # def test_the_truth
    assert true
  end

  test "cannot_save_user_without_name" do
    u = User.new
    assert_not u.save, "Houston,_we_have_a_problem"
  end

  test "cannot_save_user_without_email_address" do
    u = User.new name: "Hello"
    assert_not u.save, "Houston,_we_have_a_problem"
  end

  test "cannot_save_user_with_existing_email_address" do
    u = User.new name: "Hello", password: 'titok', email: users(:citizen).
        email
```

```
    assert_not u.save, "Houston, we have a problem"
  end

  test "encryption" do
    assert_equal User.encrypt('titok', users(:citizen).salt), Digest::SHA1.
        hexdigest("titoka")
  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 test/
    models/user_test.rb
Run options: --seed 25669

# Running:

.....

Finished in 0.145362s, 34.3968 runs/s, 34.3968 assertions/s.
5 runs, 5 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    models/
Run options: --seed 45720

# Running:

.....

Finished in 0.057545s, 86.8888 runs/s, 86.8888 assertions/s.
5 runs, 5 assertions, 0 failures, 0 errors, 0 skips
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    models/user_test.rb:4
Run options: --seed 52413

# Running:

.

Finished in 0.040184s, 24.8856 runs/s, 24.8856 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

4. Útvonalak tesztelése

Útvonalak megfelelő generálását Rails konzolon ellenőrizhetjük, ehhez be kell töltenünk az `url_helper` modult. Ezután az összes, a `routes.rb`-ben definiált útvonal helpert ellenőrizhetjük, valamint a kontroller tesztekben használható `url_for` helperrel útvonalakat rakhatunk össze.

```
irb(main):009:0> include Rails.application.routes.url_helpers
=> Object
irb(main):010:0> party_path(3)
=> "/parties/3"
irb(main):011:0> page_parties_path(2)
=> "/parties/page/2"
irb(main):012:0> hello_path
=> "/say/hello"
irb(main):013:0> default_url_options[:host]='http://localhost
    :3000'
=> "http://localhost:3000"
irb(main):014:0> hello_url
=> "http://localhost:3000/say/hello"
```

5. Kontrollerek és nézetek tesztelése

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 `SayControllerTest` tesztet, amely jelenleg egy tesztesetet tartalmaz. Ellenőrizzük, hogy az oldal betölthető-e, és van-e benne pontosan 1 darab H1 HTML elem.

```
class SayControllerTest < ActionDispatch::IntegrationTest
  test "should_get_hello" do
    get hello_path
    assert_response :success #:redirect
    assert_select 'h1', 1
  end
end
```

A funkcionális teszteseteket a `rake test:controllers` szkripttel vagy a `rails test` paranccsal futtathatjuk. A kimeneten a `.` azt jelenti, hogy egy `assert` teljesült, az `F` azt, hogy nem teljesült, és az `E`, hogy hiba van vagy a tesztesetben, vagy a kódban, és így a teszteset nem tudott lefutni.

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/controllers/
    say_controller_test.rb
```

```
Run options: --seed 42164

# Running:

.

Finished in 0.466162s, 2.1452 runs/s, 4.2904 assertions/s.
1 runs, 2 assertions, 0 failures, 0 errors, 0 skips
```

Módosítsuk a `SessionsControllerTest` tesztet A funkciót a `citizen` 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 `SessionsController` `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`, hanem a bejelentkezett felhasználó azonosítójával megegyező érték, továbbá az átirányítás utáni oldalon van egy a kijelentkezés műveletre mutató link. Az átirányítás válaszban kapott URI-t a `follow_redirect!` függvénnyel nyithatjuk meg, és azon az `assert_select` függvénnyel ellenőrizthetjük, hogy tényleg az az oldal, és tényleg azzal a tartalommal jelent-e meg. A sikertelen bejelentkezést ellenőrző tesztesetben a jelszó paraméterben teszünk különbséget, és azt várjuk, hogy visszairányítódunk az előző oldalra, a session üres, és van az oldalon egy `Login` címkéjű form. 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, és az átirányítás után oldalon lesz egy `Login` értékű LEGEND HTML elem. 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
    get hello_path
    assert_response :success
    assert_select 'legend', 'Login'

    post '/sessions/create', params: { email: users(:citizen).email,
        password: 'titok' }, headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    assert_equal session[:user], users(:citizen).id
```

```
    follow_redirect!
    assert_select 'a', 'Logout'
  end

  test "invalid login" do
    get hello_path
    assert_response :success
    assert_select 'legend', "Login"

    post '/sessions/create', params: { email: users(:citizen).email,
        password: 'titok2' }, headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    assert_not_equal session[:user], users(:citizen).id
    follow_redirect!
    assert_select 'legend', 'Login'
  end

  test "logout" do
    get hello_path
    assert_response :success
    assert_select 'legend', 'Login'

    post '/sessions/create', params: { email: users(:citizen).email,
        password: 'titok' }, headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    assert_equal session[:user], users(:citizen).id
    follow_redirect!
    assert_select 'a', 'Logout'

    get '/sessions/destroy', headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    assert_nil session[:user]
    follow_redirect!
    assert_select 'legend', 'Login'
  end
end
```

A tesztet e kontrollerteszt kiválasztásával futtatjuk:

```
kovacsg@debian:~/gyakorlat: RAILS_ENV='test' rails test test/
    controllers/sessions_controller_test.rb
Run options: --seed 2353

# Running:

...

Finished in 0.253767s, 11.8218 runs/s, 70.9311 assertions/s.
3 runs, 18 assertions, 0 failures, 0 errors, 0 skips
```

6. Integrációs teszt

A tesztek harmadik típusa az integrációs teszt, amellyel egy párt logója feltöltésének folyamatát ellenőrzük.

```
kovacsg@debian:~/gyakorlat/test/integration> rails g
    integration_test upload_party_logo
       invoke  test_unit
       create    test/integration/upload_party_logo_test.rb
```

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

A tesztadatok a `test_helper` fájlra való hivatkozás miatt automatikusan elérhetők. A teszt hat tesztlépésből áll egy felhasználó számára:

- lekérdezünk egy random képernyőt, ahol van bejelentkező form,
- kitöltjük a bejelentkezési formot, és rákattintunk bejelentkezés gombra,
- lekérdezzük a pártok listáját,
- átmegyünk a pártok listájának második lapjára, ahol a `T3` rövidítéssel rendelkező párt látható,
- rákattintunk az e párt sorában lévő szerkesztés linkre,
- kitöltjük a párt adatlapjának formját, és feltöltjük a logót, és
- végül kilépünk.

Az első lépésben egy be nem jelentkezett felhasználó betölt egy véletlen oldalt, amelyen elérhető a bejelentkezési form. Itt annyit feltételezünk, hogy az oldal sikeresen betöltődik,és az oldalon HTML nézetének forrásában van egy `Login` címkéjű `legend` HTML elem.

A második tesztlépés a bejelentkezés linkre „kattintás”. Az a feltételezünk, hogy a felhasználói `session` beállítódik, és az átirányítás utáni oldalon a menüben elérhető a `Parties` címkéjű link.

A pártok linkre kattintva lekérdezzük a pártok listáját az `index` nézettel. Az a feltételezésünk, hogy egy oldalon egyszerre csak két párt lesz látható.

Mivel a pártok első listaoldalán nem látható az általunk keresett párt, átlapozunk a második oldalra. Az a feltételezésünk, hogy az oldal betöltődik, és ott van egy a párt rövidítésével megegyező szövegelemmel rendelkező `td` cella a táblázatban.

A szerkesztés linkre kattintva azt feltételezzük, hogy az oldal betöltődik, és azon lesz pontosan 1 darab HTML form.

A nézeten található egy form, amelyen kereszül egy `file` nevű paramétert juttathatunk el az `update` akciónak HTTP POST üzenettel. A paraméternek

egy, a `fixture_file_upload` metódussal a tesztadatok közül, a helyi fájlrendszerről kiválasztott fájlt adunk meg. Azt várjuk, hogy az adatbázisban a feltöltött logók metaadatainak száma eggyel nő a feltöltés előtti állapothoz képest, és a fájl megjelenik az elvárt névvel a fájlrendszeren.

Végül a funkcionális tesztből ismert módon kilépünk, és azt feltételezzük, hogy a session törlődik, és az átirányítás utáni oldalon van egy `Login` értékű LEGEND HTML elem.

```
require 'test_helper'

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

  test "upload logo for party" do
    get hello_path
    assert_response :success
    assert_select "legend", "Login"

    post '/sessions/create', params: { email: users(:citizen).email,
        password: 'titok' }, headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    assert_equal session[:user], users(:citizen).id
    follow_redirect!
    assert_select 'a', 'Parties'

    get parties_path
    assert_response :success
    assert_select 'tr', 2

    get page_parties_path(2)
    assert_response :success
    assert_select 'td', 'T3'

    get edit_party_path(parties(:maradek3).id)
    assert_response :success
    assert_select 'form', 1

    upload_file = fixture_file_upload('test/fixtures/files/logo.png', 'image
        /png')
    patch party_path(parties(:maradek3).id), headers: { "HTTP_REFERER":
        party_path(parties(:maradek3).id) }, params: { party: { logo:
        upload_file, name: parties(:maradek3).name, abbreviation: parties(:
        maradek3).abbreviation } }
    assert_response :redirect
    follow_redirect!
    assert_select 'div.logo', 1
    assert_equal Logo.all.size, 1
    assert File.exist? "public/data/#{Logo.last.id}"

    get '/sessions/destroy', headers: { 'HTTP_REFERER': hello_path }
    assert_response :redirect
    assert_nil session[:user]
    follow_redirect!
    assert_select 'legend', 'Login'
  end
end
```

Az integrációs tesztet a funkcionális tesztekkel megegyező módon futtathatjuk:

```
kovacsg@debian:~/gyakorlat/test> RAILS_ENV='test' rails test test
    /integration/upload_party_logo_test.rb
Run options: --seed 55511

# Running:

.

Finished in 0.666600s, 1.5001 runs/s, 27.0027 assertions/s.
1 runs, 18 assertions, 0 failures, 0 errors, 0 skips
```

7. Rendszertesztek

A rendszertesztek fekete dobozos tesztek, a rendszert úgy ellenőrzik, ahogy azt a felhasználó látja, és nem használnak fel belső információt. A futtatáshoz szükségünk lesz egy telepített Google Chrome böngészőre, a többit a végrehajtó rendszer elintézi.

A rendszerteszteket explicit paranccsal kell létrehoznunk.

```
kovacsg@debian:~/gyakorlat/test/system> rails g system_test hello
      invoke  test_unit
      create    test/system/hellos_test.rb
```

A bejelentkezés rendszertesztje a következőképp néz ki. Megnyitunk egy oldalt, kitöltjük az egyes címkékhez tartozó beviteli mezőket, rákattintunk a nyomógombra, és megnézzük a következő oldal tartalmát.

```
class HellosTest < ApplicationSystemTestCase
  test "visiting the root page" do
    visit '/say/hello'
    assert_selector 'legend', text: "Login"

    fill_in "Email", with: 'varos.lako@magyarorszag.hu'
    fill_in "Password", with: 'titok'
    click_on "Login"

    assert_selector 'a', text: "Logout"
  end
end
```

A rendszertesztek futtatásakor meg-megnyílik a böngésző, és láthatjuk a tesztek végrehajtátás.

```
kovacsg@debian:~/gyakorlat> RAILS_ENV='test' rails test test/
    system/hellos_test.rb
Run options: --seed 14296

# Running:

DEBUGGER: Attaching after process 907811 fork to child process
    907950
Capybara starting Puma...
* Version 5.6.2 , codename: Birdie's Version
* Min threads: 0, max threads: 4
* Listening on http://127.0.0.1:45615
.

Finished in 12.845764s, 0.0778 runs/s, 0.1557 assertions/s.
1 runs, 2 assertions, 0 failures, 0 errors, 0 skips
```

8. Teljesítménytesztek

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 (és a `bundle install` paranccsal telepítenünk), 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> rails g performance_test hello
Running via Spring preloader in process 10929
      create  test/performance/hello_test.rb
```

Az alábbi teszteset a kezdőoldalt támadja meg 5 egymás utáni kéréssel.

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

class HelloTest < 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'
  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:benchmark` és a `test:profile` `rails` célokkal hajthatunk végre.

A teljesítménytesztek függvénykönyvtáraiban jelenleg inkompatibilitás van, így a teszteket nem tudtuk végrehajtani.

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.