Szoftvertesztelés

*Unit tesztelés – Laravel*

Czimbalmos Ákos

Tartalomjegyzék

[Ábrajegyzék 2](#_Toc133336347)

[Bevezető 3](#_Toc133336348)

[Laravel 3](#_Toc133336349)

[A weboldal 4](#_Toc133336350)

[Unit tesztek Laravel-ben 4](#_Toc133336351)

[A tesztek 5](#_Toc133336352)

[A tesztek listája 5](#_Toc133336353)

[Az űrlappal kapcsolatos tesztek 5](#_Toc133336354)

[Person Model tesztek 5](#_Toc133336355)

[Interest Model tesztek 5](#_Toc133336356)

[Blood Type Model tesztek 5](#_Toc133336357)

[Az űrlap működésének tesztelése 5](#_Toc133336358)

[Model-ek tesztelése 8](#_Toc133336359)

[Összefoglaló 10](#_Toc133336360)

# Ábrajegyzék

[1. Példa teszt 4](#_Toc133334574)

[2. Interests teszt 5](#_Toc133334575)

[3. Person is in database teszt (1) 6](#_Toc133334576)

[4. Person is in database teszt (2) 6](#_Toc133334577)

[5. Person model 7](#_Toc133334578)

[6. Interest model 8](#_Toc133334579)

[7. Model teszt setUp() metódusa 8](#_Toc133334580)

[8. has\_interests() teszt 9](#_Toc133334581)

# Bevezető

A **unit tesztek** az alkalmazásfejlesztés során nélkülözhetetlen eszközök, amelyek segítenek a kód minőségének és megbízhatóságának javításában. A unit tesztek arra szolgálnak, hogy az alkalmazás egyes részeinek, egységeknek a működését ellenőrizzék. A unit tesztek alapvetően fontosak, mert segítenek a hibák korai felfedezésében és azok javításában, ezzel pénzt és időt spórolva a fejlesztőnek, mivel már a fejlesztési szakaszban kiszűrhetőek a kisebb – nagyobb hibák, és a fejlesztésnek e fázisában könnyen kijavíthatóak.

# Laravel

A **Laravel** egy nyílt forráskódú PHP alapú webes alkalmazásfejlesztő keretrendszer, amelyet Taylor Otwell fejlesztett ki. Az egyik legnépszerűbb PHP keretrendszerként ismert, és széles körben használják a webalkalmazások és API-k fejlesztésére. A Laravel keretrendszer számos beépített funkciót és eszközt kínál az alkalmazásfejlesztéshez, beleértve az adatbázis-kezelést, a sablonrendszert, az események kezelését és még sok mást. Emellett támogatja a modern fejlesztési elveket, mint például az MVC (Model-View-Controller) architektúrát, ami azt jelenti, hogy az alkalmazás kódja három fő részre van szétosztva: modelre, nézetre (view) és vezérlőre (controller). A model osztály felelős az adatok feldolgozásáért, az adatbázissal való kommunikációért, az adatok szabályozásáért valamint a modellek egymással való kapcsolata is itt van definiálva. A nézet az alkalmazás felhasználói felületét reprezentálja. Ez az osztály felelős az adatok megjelenítéséért a felhasználó számára. A nézetek az alkalmazás kinézetét és megjelenítését határozzák meg, és lehetővé teszik a felhasználóval való interakciót. Laravel-ben általában a nézetek is több részre vannak bontvan, mint például a bejelentkezéssel kapcsolatos nézetek, modellekkel kapcsolatos nézetek és más dolgokkal kapcsolatos nézetek (pl: navbar, más layout-ok). A vezérlő az alkalmazás logikáját és működését irányítja. Ez az osztály felelős a felhasználói kérések fogadásáért, az adatok feldolgozásáért és az alkalmazás állapotának frissítéséért.

# A weboldal

A projekt egy weboldalat valósít meg, ami három részből áll:

1. Egy főoldal, aminek különösebb jelentősége nincs, ezt az odlalt látja a felhasználó, ha elindítja az oldalt.
2. Egy űrlap, ahol számos kérdésre kell válaszolnia a felhasználónak mielőtt beküldené.
3. A tagok oldala, ahol láthatóak a személyek akik eddig kitöltötték az űrlapot.

# Unit tesztek Laravel-ben

Alapértelmezetten az alkalmazásod tesztek könyvtára két almappát tartalmaz: Feature és Unit. Az Unit tesztek azok, amelyek nagyon kis, elszigetelt részekre fókuszálnak a kódban, mint például itt teszteljük, hogy a modellek a megfelelő relációval kapcsolódnak-e egymáshoz, vagy hogy egy kontroller metódusa megfelelően működik-e. Új teszt létrehozásához a ***make:test*** Artisan parancsot használjuk. Alapértelmezetten a tesztek a tests/Feature könyvtárba kerülnek. Ha tesztet a tests/Unit könyvtárban szeretnénk létrehozni, akkor a --unit opciót kell használni a make:test parancs végrehajtásakor:

php artisan make:test UserTest –unit

Miután elkészült a teszt generálása, külön függvényekben kell megírni a teszteket. A teszteket a ***php artisan test*** paranccsal lehet futtatni a terminálodból, vagy a ***Run*** gombbal futathatunk teszteket egyenként .

1. Példa teszt

# A tesztek

## A tesztek listája

### Az űrlappal kapcsolatos tesztek

* test\_response(): A member.create oldal megnyitása után a 200-as státusz kódot kapjuk-e
* test\_interests()
* test\_person\_is\_in\_database()
* test\_first\_10\_persons\_name(): teszteljük, hogy a member.index oldalon egyszerrre valóban csak 10 ember nevét látjuk-e
* test\_persons\_data(): egy személy űrlapjának elmentése után a member.show oldalon a megfelelő adatokat látjuk-e

### Person Model tesztek

* has\_interests()
* has\_bloodtype()

### Interest Model tesztek

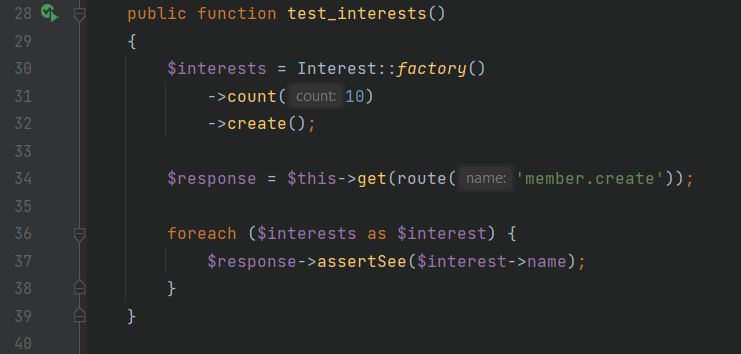
* has\_person()

### Blood Type Model tesztek

* has\_person()

## Az űrlap működésének tesztelése

A ***Mytest.php*** fájlban levő tesztek az űrlap helyes működését tesztelik, hogy a kitöltött űrlap elküldése után az adatok helyesen mentődnek-e el az adatbázisba.

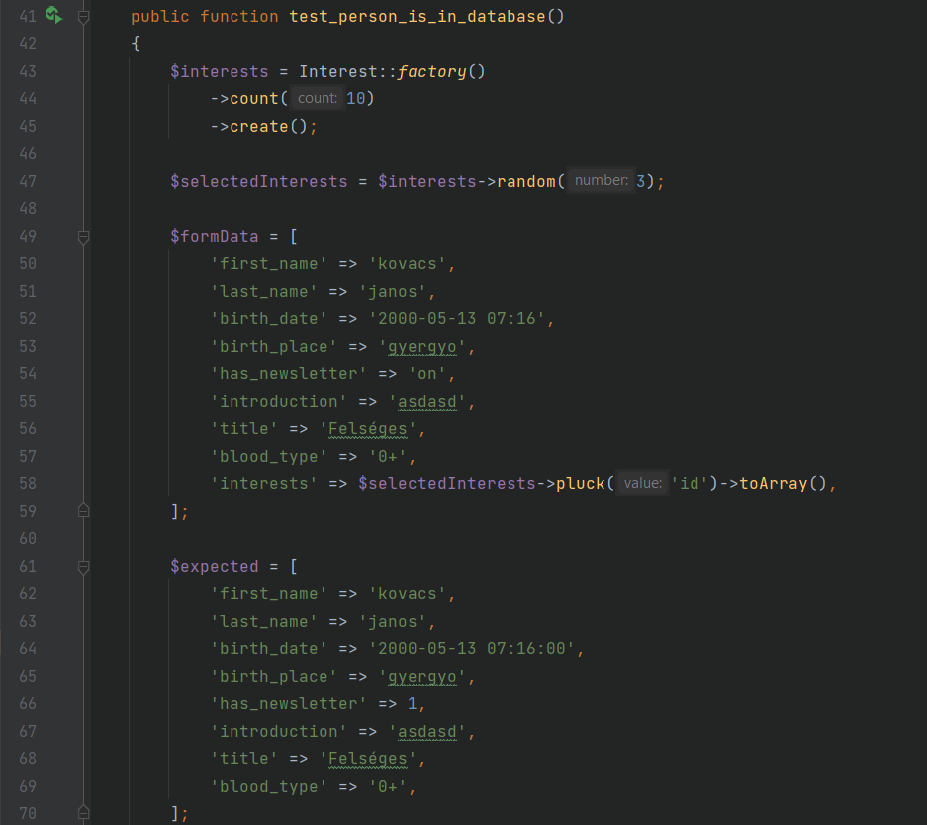
Az érdeklődési körök külön seeder-el vannak bevíve az adatbázisba, ezért arra külön teszt van, hogy a bevitel után ténylegesen elmentődnek-e:

2. Interests teszt

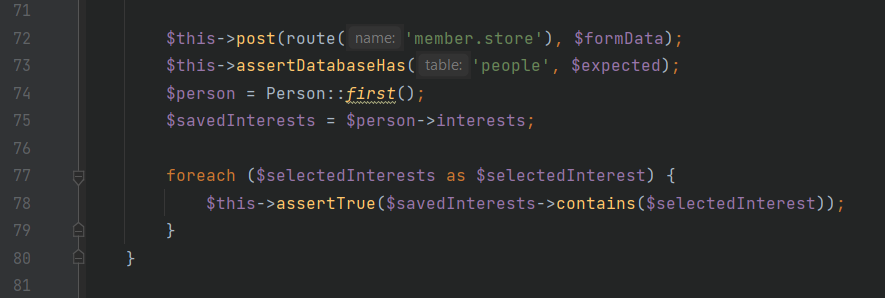
Ebben a tesztben az Interest factory segítségével generálunk random érdeklődési köröket (a teszt szempontjából nem lényeges, hogy a generált szavak valóban érdeklődési körök-e vagy sem, itt csak az számít, hogy helyesen lettek-e elmentve), majd ezeket egy seeder segítségével bevisszük az adatbázisba, és meghívva a member.create útvonalat megnézzük, hogy valóban látszódnak-e a bevitt adatok az űrlapon.

A következő tesztben azt vizsgáljuk, hogy az űrlap kitöltése és beküldése után úgy mentődik-e el az adatbázisba, ahogy azt elvárjuk:

Először létrehozunk ***formData*** változót, amibe elmentjük az űrlap adatait, majd egy ***expected*** változóba beírjuk az elvárt eredményt.



3. Person is in database teszt (1)

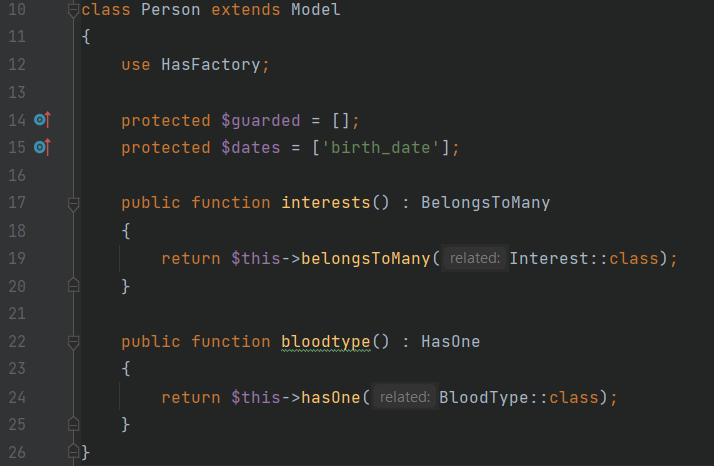
Ezt követően meghívjuk a kontroller *store* metódusát, ami eltárolja a ***formData*** adatait az adatbázisba, és megnézzük hogy az eltárolt adatok megegyeznek-e az ***expected*** adataival.

4. Person is in database teszt (2)

## Model-ek tesztelése

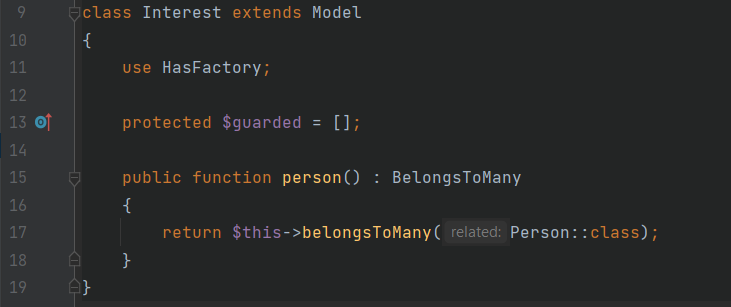
A model tesztelésére azért van szükség, hogy tudjuk valóban jó kapcsolat van két vagy több model között. A kapcsolatok lehetnek: HasOne, HasMany, BelongsTo, BelongsToMany, stb.

A Person Model a következő képpen van implementálva:

Az érdeklődési körök több a többhöz kapcsolatban, és a vércsoport pedig egy a többhöz kapcsolatban áll minden személlyel.

5. Person model

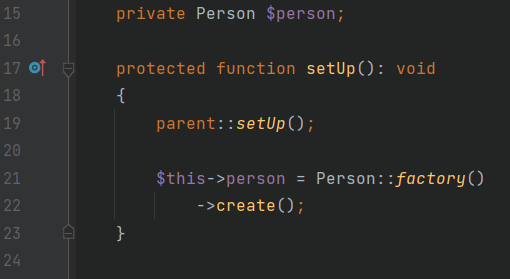
Ehhez hasonlóan az Interest Model-ben is implementálva van a kapcsolat a Person Model-lel.



6. Interest model

Ezek tesztelése a következő képpen zajlik:

PersonTest:

1. Importáljuk a teszt osztályba a tesztelendő model-ek namespace-jét
2. Létrehozunk egy setUp() metódust ahol létrehozunk egy Person objektumot amit a tesztekben használni fogunk.

7. Model teszt setUp() metódusa

1. Az assertInstanceOf() metódus segítségével ellenőrizzük, hogy a kapott érték egy példánya-e az adott osztályból

8. has\_interests() teszt

Ha a teszt sikeresen lefut, akkor az azt jelenti, hogy a *$this -> person -> interests* reláció első eleme egy Interest objektum. Tehát azáltal, hogy a $this->person->interests relációt meghívom és ellenőrzöm az első elemét az assertInstanceOf() metódussal, tesztelve van, hogy a Person és az Interest osztály kapcsolatban vannak egymással az adott reláció alapján.

# Összefoglaló

Az weboldal tesztelése gördülékenyen ment, megismerkedtem egy űrlap oldal tesztelésével. Eddig csak model teszteket írtam de most már az adatbázisa mentett adatok minőségét is tudom tesztelni, a tantárgynak köszönhetően.