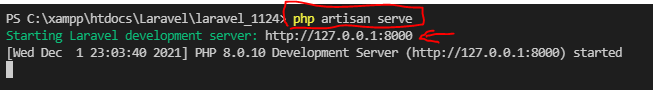
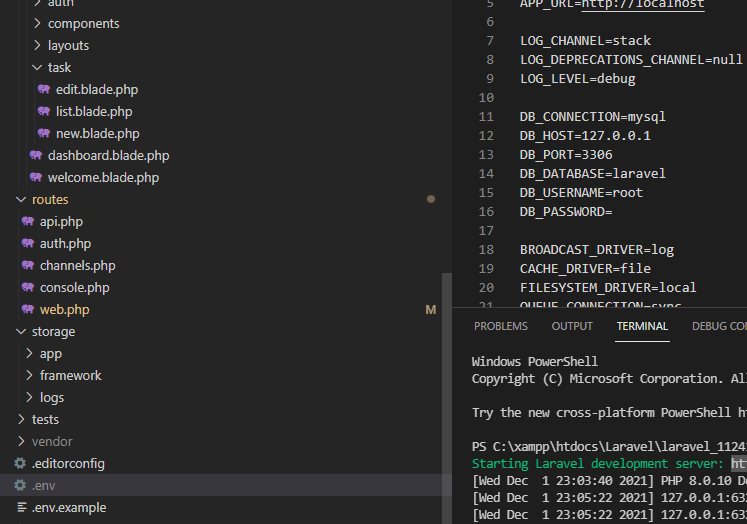
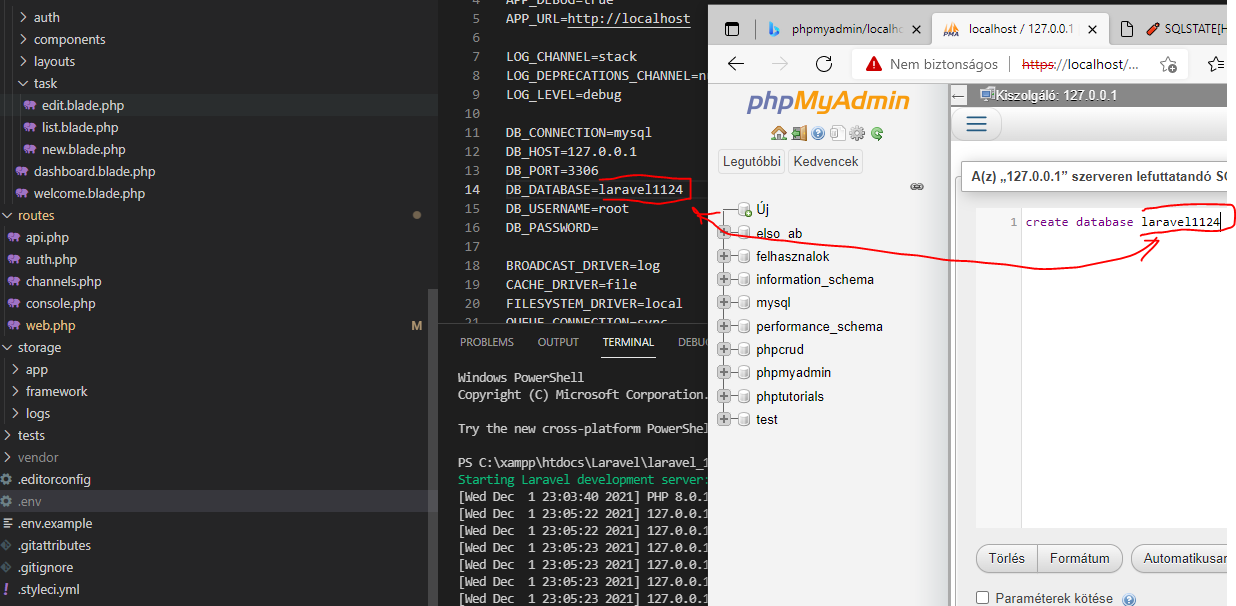
**Projekt otthoni betöltésének lépései:**

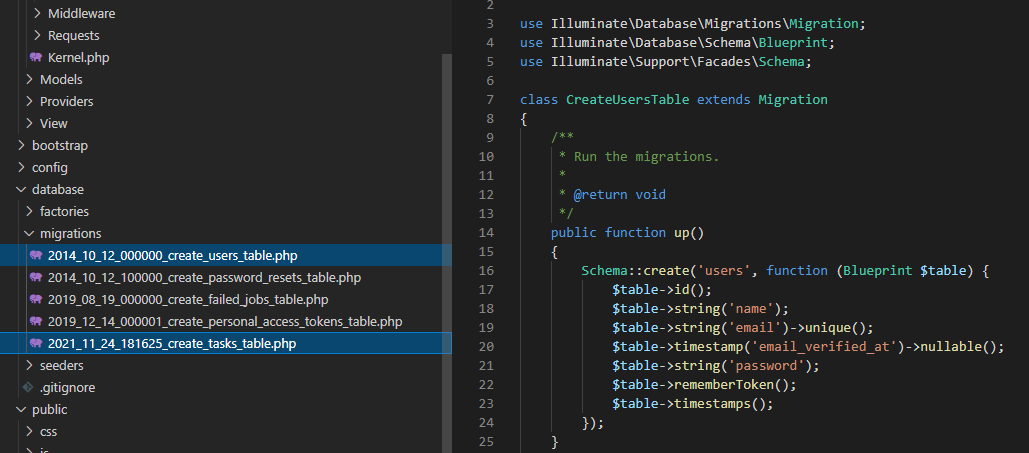
1. Projekt klónozása lokális mappába (a kreálandó projektmappához képest egy szinttel feljebb lépünk, elindítjuk a Gitbasht, majd 🡪 git clone *gites\_http\_link*)
2. CMD 🡪 cd projektmappa\_eleresi\_utja  
   
3. Közvetlen a mappából *composer install* (CMD)
4. XAMPP (Apache, MySQL) bekapcsol
5. Projekt betöltése VS Code-ba, új terminál, majd:  
   
6. env.example fájlról másolat készítése, majd annak átnevezése simán env-re:

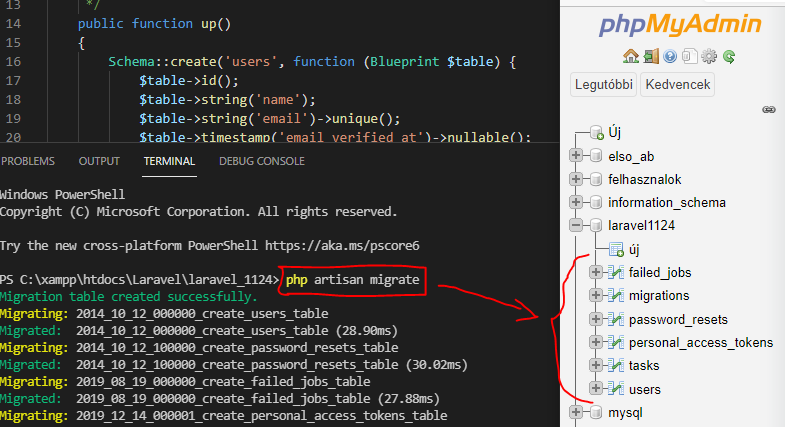


1. <http://localhost:8000/> 🡪 generate app key 🡪 refresh now 🡪 log in
2. Létrehozok egy adatbázist MyPHP admin alatt laravel1124 néven, és átírom az env fájlban is az AB nevet:

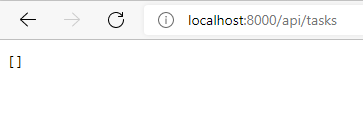


1. A táblák adottak, következhet a migráció:





Be kell majd jelentkezni a <http://localhost:8000/> alatt (furcsa módon itthon újra kellett regisztrálnom, mert nem fogadta el a bejelentkezési adataimat. Az újbóli regisztrációnál ugyanezeket az adatokat adtam meg….) .Tesztelem a <http://localhost:8000/api/tasks> linkkel. Nincs hibaüzenet, betöltődik az – egyelőre még üres – adatbázis:



**CSRF token**

A sessionhöz van eltárolva. Biztonságot nyújt, pl. hogy más oldalon ne tudjanak visszaélni az adatainkkal, ne tudják azokat elérni. Pl. megszabhatjuk, hogy a létrehozást, módosítást ne engedje meg. A token határozza meg a jogosultságokat, pl. ha valaki létrehozhat új elemet.

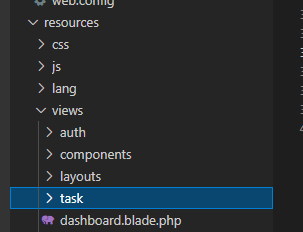
**Form és metódusok**

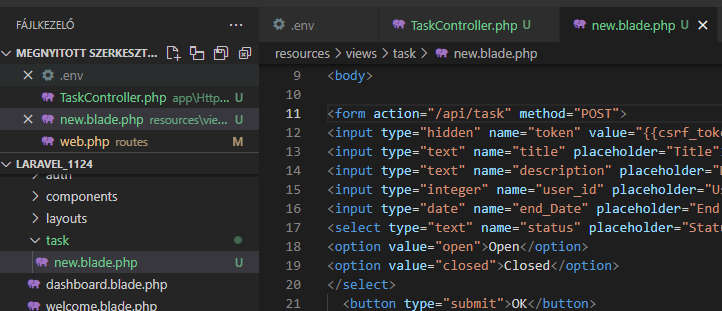
A PUT és a DELETE nem támogatott metódusok a form által. A form csak GET és POST metódusokat tud kezelni.

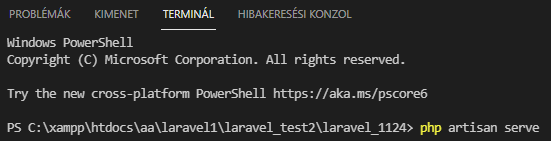
PUT és a DELETE : másik formot kell létrehozni ezekhez, rejtett inputmezővel, amely nem POST, hanem PUT metódussal van ellátva.

A *new.blade.php*-ben található form a *web-php store()* metódusára mutat. Submit-tal elküldi az action-ben meghatározott útvonalra („/api/task”) a POST-ot, és ez az útvonal hozza létre az adatbázis rekordunkat.

A *views* alatt létrehozzuk a *task* mappát, azon belül a *new.blade.php*-t, benne pedig egy formot:



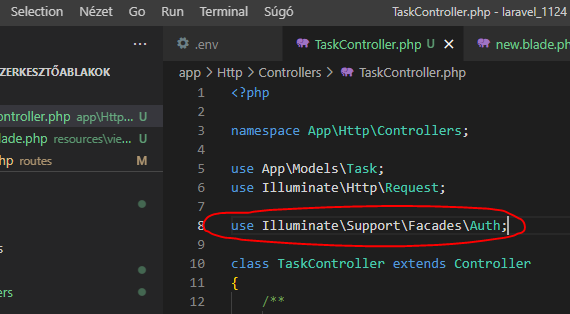




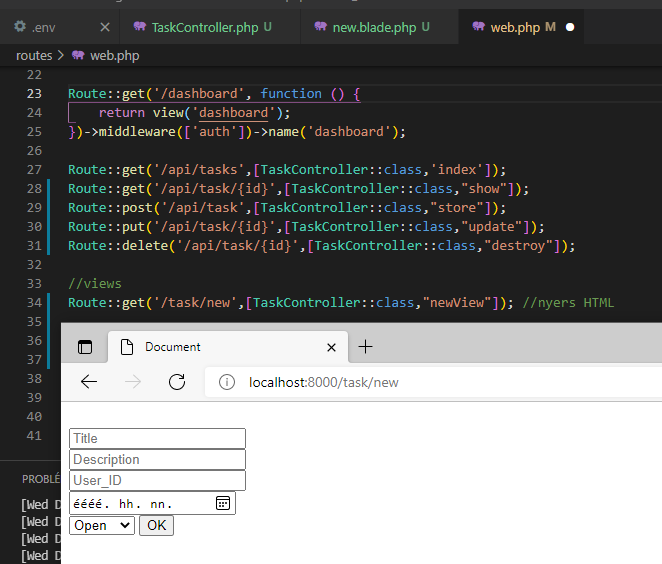
Laravel segéd classok (Facades) elérése: use Illuminate\Support\Facades\XXXXX

(Hiányuk hibát okozhat (az adott osztály előfordulásai pirossal aláhúzottak, és view error jelenik meg.)

pl. az Auth esetében: use Illuminate\Support\Facades\Auth

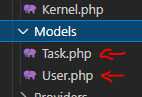


Routes 🡪 web alatt vannak az elérhető útvonalak.

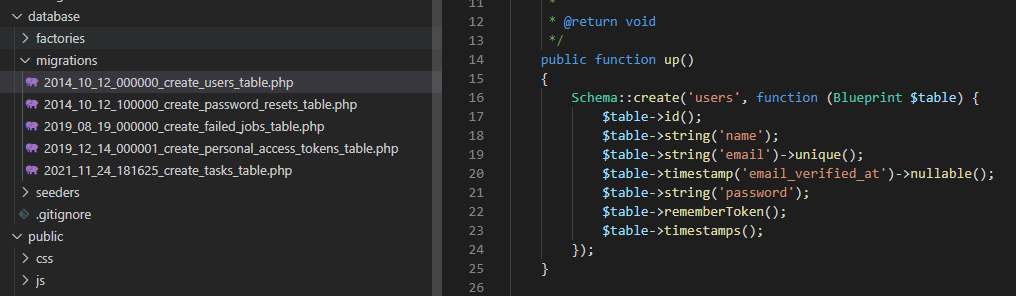


Model: magát az osztályt (és annak adattábláját/adatait) reprezentálja.

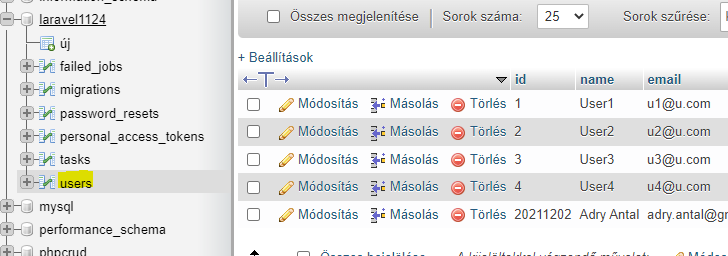
Ebben a példában 2 modell szerepel: Task és User. (app 🡪 Models)



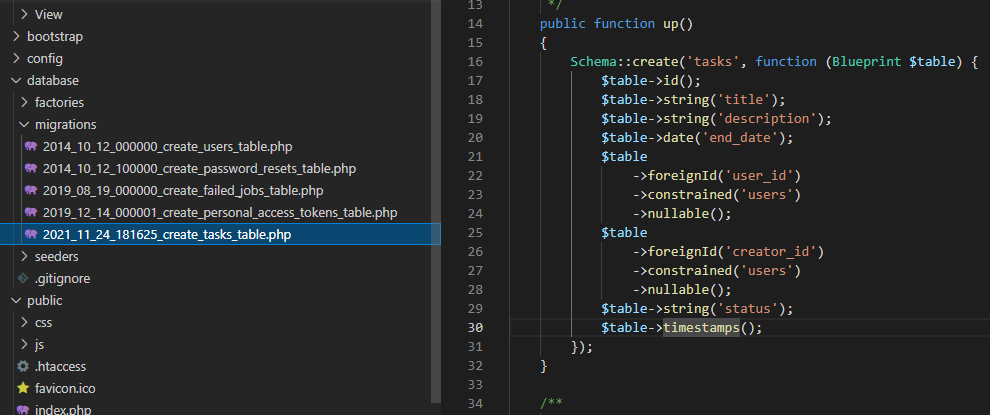
Már adott a felhasználók adatait tároló ’users’ tábla (database 🡪 Migrations):

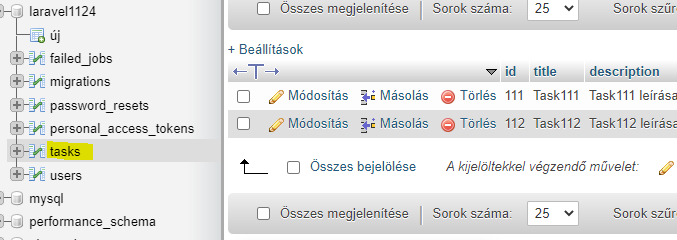


(Migráció után):

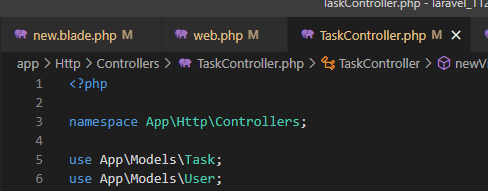


Illetve létrehoztuk (és migráltuk is) a ’tasks’ táblát is:



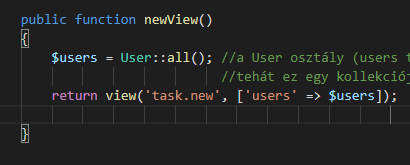


Létrehoztuk a *TaskControllert*, amely a *web.php* Route utasításaihoz rendelt függvényeket fejti ki. Mindkét modell használatát engedélyeztük benne:



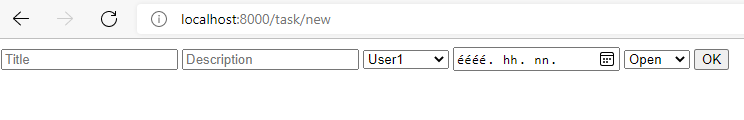
Ami történik:

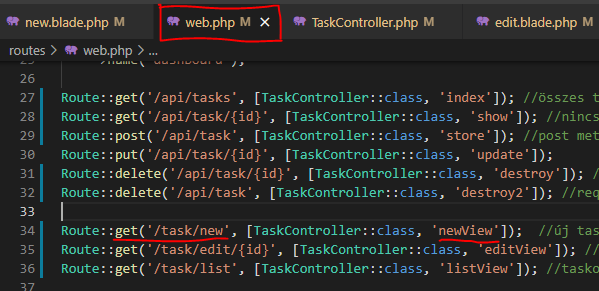
* **Új adat beszúrása**

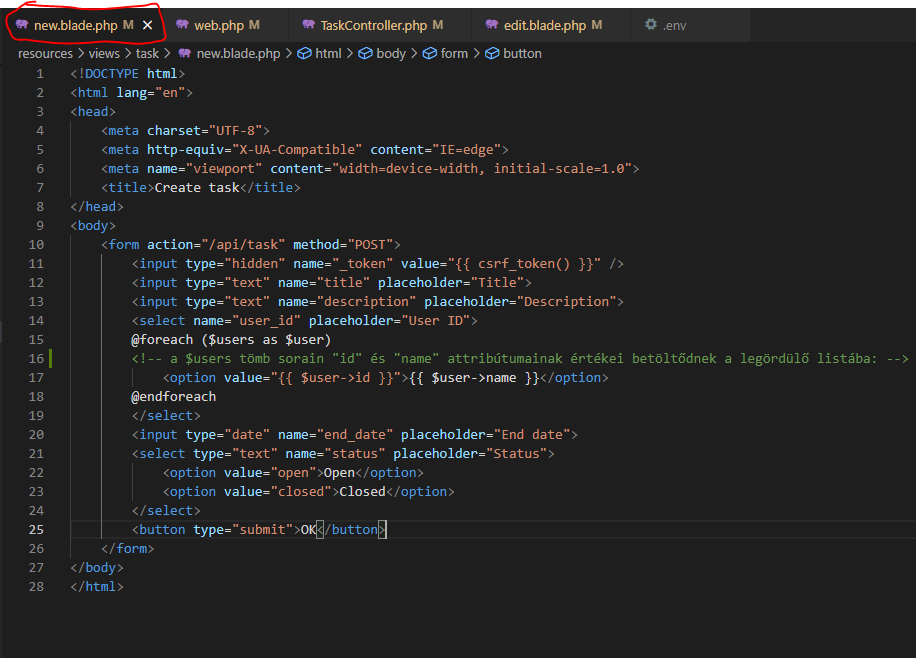


A *TaskController* newView() metódusa a User osztály (tehát a hozzá kapcsolódó 'users' tábla) összes rekordját kiolvassa az adatbázisból, és a $users változóba tölti, mely tehát egy kollekciója lesz a user rekordoknak. A rekordokat tömb formájában elküldi a new.blade.php view-nak (a 'users' kulccsal ellátva). A tömb minden sora egy rekord, és ezt a tömböt járjuk majd be a *new.blade.php* view-ban.

<http://localhost:8000/task/new> 🡪 a kérést teljesítő GET metódus hatására megjelenik a newView által megjelölt views 🡪 task 🡪 *new.blade.php* tartalma a böngészőben, amely a lentebbi form lesz:

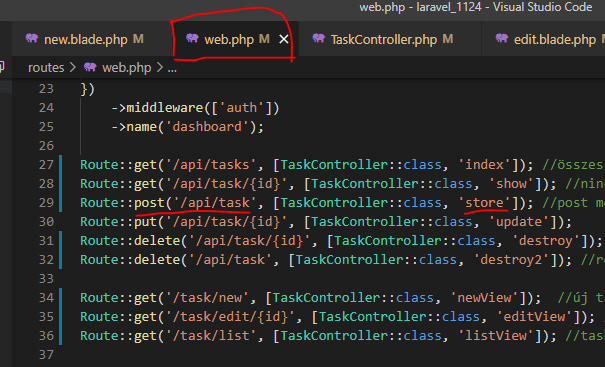


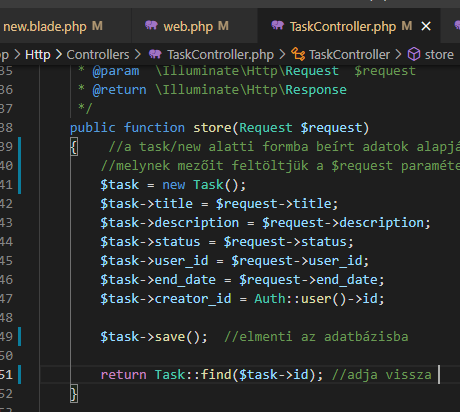




Megjegyzés: mindig kell a CSRF token, különben *Page Expired* hibát fogunk kapni!!!

A formot ha kitöltjük, akkor a beírt adatokat egy POST metódus továbbítja az *action*-ben meghatározott */api/task* API végpontra. A web.php-ban látjuk, hogy a végponthoz hozzá van rendelve egy Route::post utasítás, amely a *TaskController* store() metódusát hívja meg:

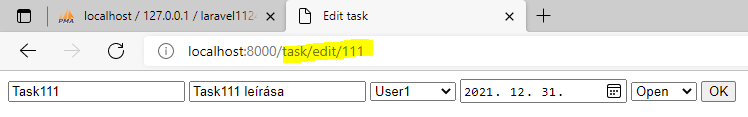


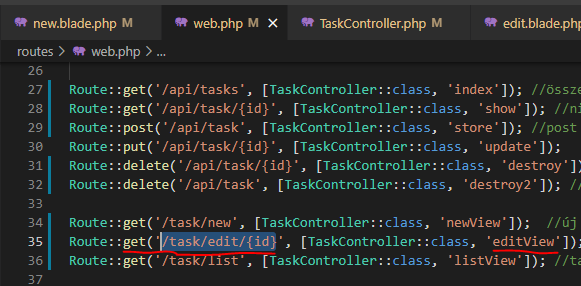


A task/new útvonal alatti formba beírt adatok alapján létrehoz egy új Task típusú rekord-objektumot, melynek mezőit feltölti a $request paraméteren - tehát a formon - keresztül érkező értékekkel, és elmenti az adatbázisba.

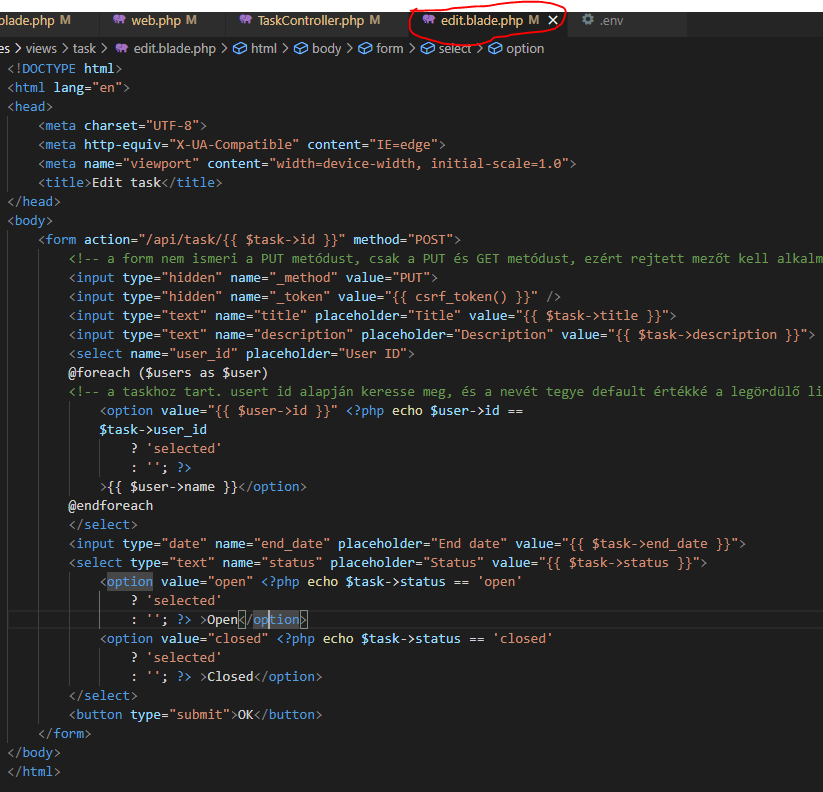
* **Adott id-jú task szerkesztése**

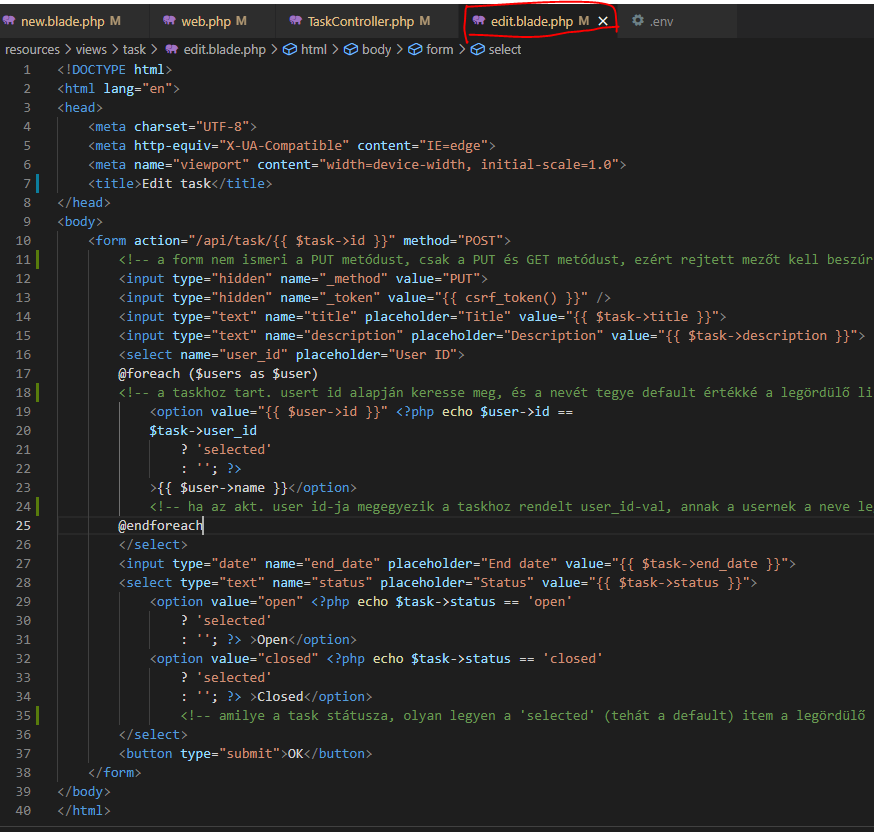
A <http://localhost:8000/task/edit/111> útvonalat a böngészőbe beírva a kérést teljesítő GET metódus hatására megjelenik az editView által megjelölt views 🡪 task 🡪 *edit.blade.php* tartalma a böngészőben, amely a lentebbi form lesz:



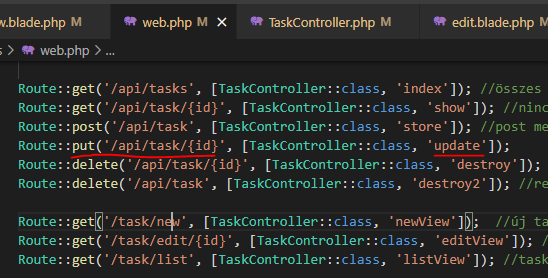


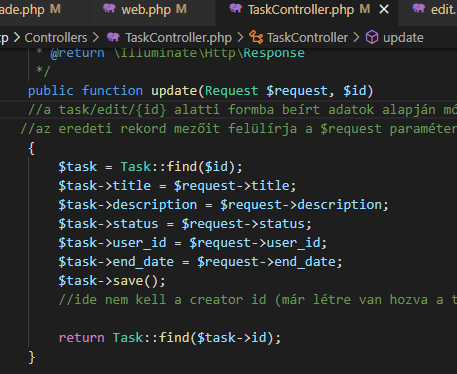
Mivel a form nem ismeri a PUT metódust, csak a POST és GET metódust, ezért rejtett mezőt kell beszúrni, és abba beágyazni a PUT-ot. (Ez „felülírja” az egy sorral lejjebb található POST-ot).





A SUBMIT hatására az adatokat egy PUT metódus az /api/task/{id} API végpontra továbbítja. A web.php-ban látjuk, hogy ehhez a végponthoz hozzá van rendelve egy Route::put utasítás, amely a *TaskController* update() metódusát hívja meg:

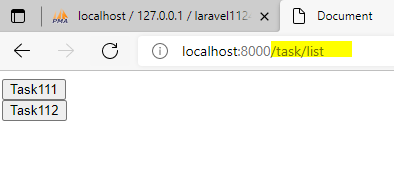


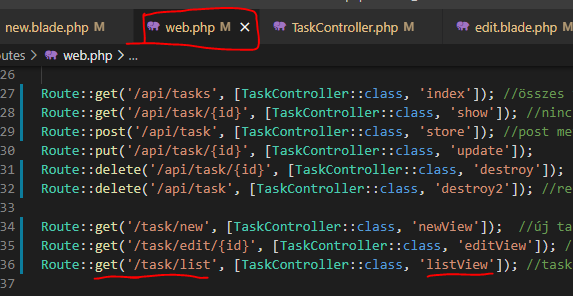


A metódus a task/edit/{id} útvonal alatti formba beírt adatok alapján módosítja az adott id-jú Task típusú rekord-objektumot. Az eredeti rekord mezőit felülírja a $request paraméteren - tehát a formon - keresztül érkező értékekkel az adatbázisban.

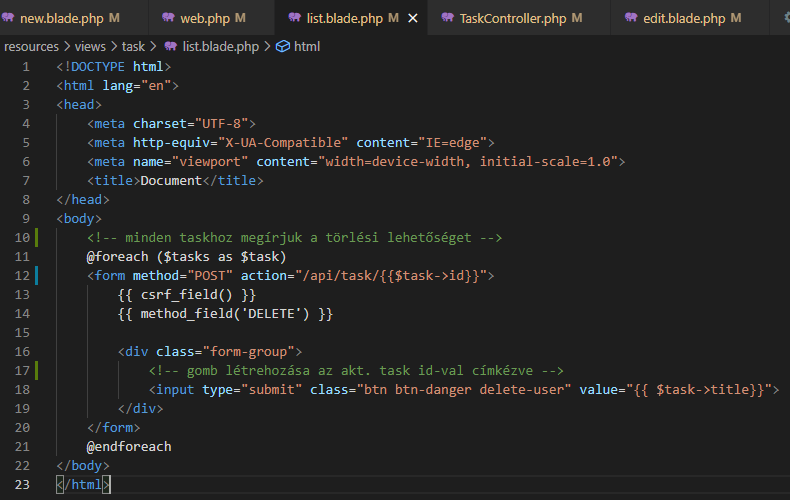
* **Adott id-jú task törlése**

A <http://localhost:8000/task/list> útvonalat a böngészőbe beírva a kérést teljesítő GET metódus hatására megjelenik a listView által megjelölt views 🡪 task 🡪 *list.blade.php* tartalma a böngészőben, amely a lentebbi oldal lesz:

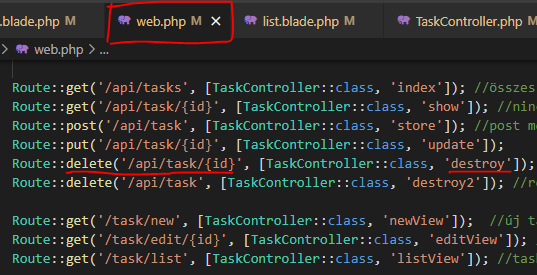


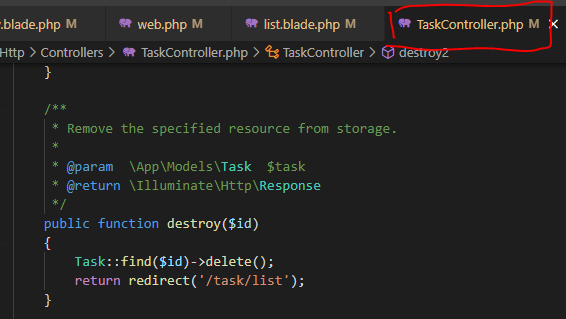


Mivel a form nem ismeri a DELETE metódust, csak a PUT és GET metódust, itt is „trükkhöz” kell folyamodnunk. (Ez „felülírja” az egy sorral lejjebb található POST-ot).



A SUBMIT hatására az adatokat egy DELETE metódus az /api/task/{id} API végpontra továbbítja. A web.php-ban látjuk, hogy ehhez a végponthoz hozzá van rendelve egy Route::delete utasítás, amely a *TaskController* destroy() metódusát hívja meg:





A függvény paraméterként megadott id-jú rekord adatbázisból való eltávolítása után visszairányít a törlési felületre.

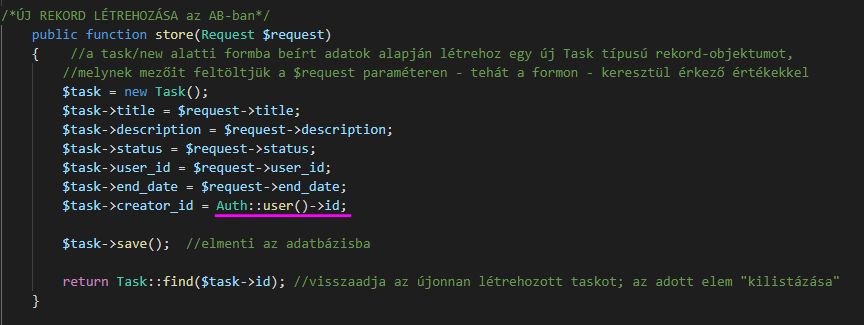
Ha formon keresztül POST (ill. PUT/DELETE) metódussal, tehát kérés alapján kell módosítást végezni az adatbázisban, akkor az request alapján történik, tehát a Controllerben található függvény bemenő paramétere Request $request lesz.

Ha pedig egy bizonyos rekordot szeretnénk módosítani/törölni, akkor ID alapján történik a „kiválasztás, tehát a Controllerben található függvény bemenő paramétere $id lesz. 🡪 pl. ez a verzió alkalmazandó gombok esetében (adott rekord törlése, módosítása)

---

**FONTOS!**

$task->creator\_id = Auth::user()->id;

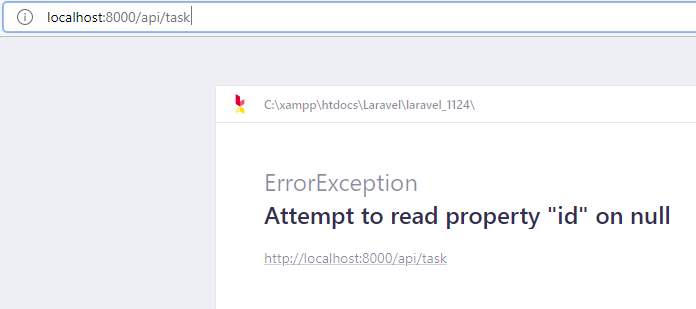


Az **Auth** osztály *user()* függvénye miatt CSAK akkor kreálhatunk taskot, ha be vagyunk jelentkezve. Tehát vagy belépünk a <http://localhost:8000/login> oldalon keresztül egy, a projekthez tartozó adatbázisban (Users tábla) létező login ID / jelszó párossal, vagy a <http://localhost:8000/register> linken keresztül regisztrálunk, és az újonnan felvitt bejelentkezési adatokkal bejelentkezünk. Amennyiben ezt nem tesszük meg, a <http://localhost:8000/task/new> megnyitásakor a következő hibaüzenetbe botlunk:

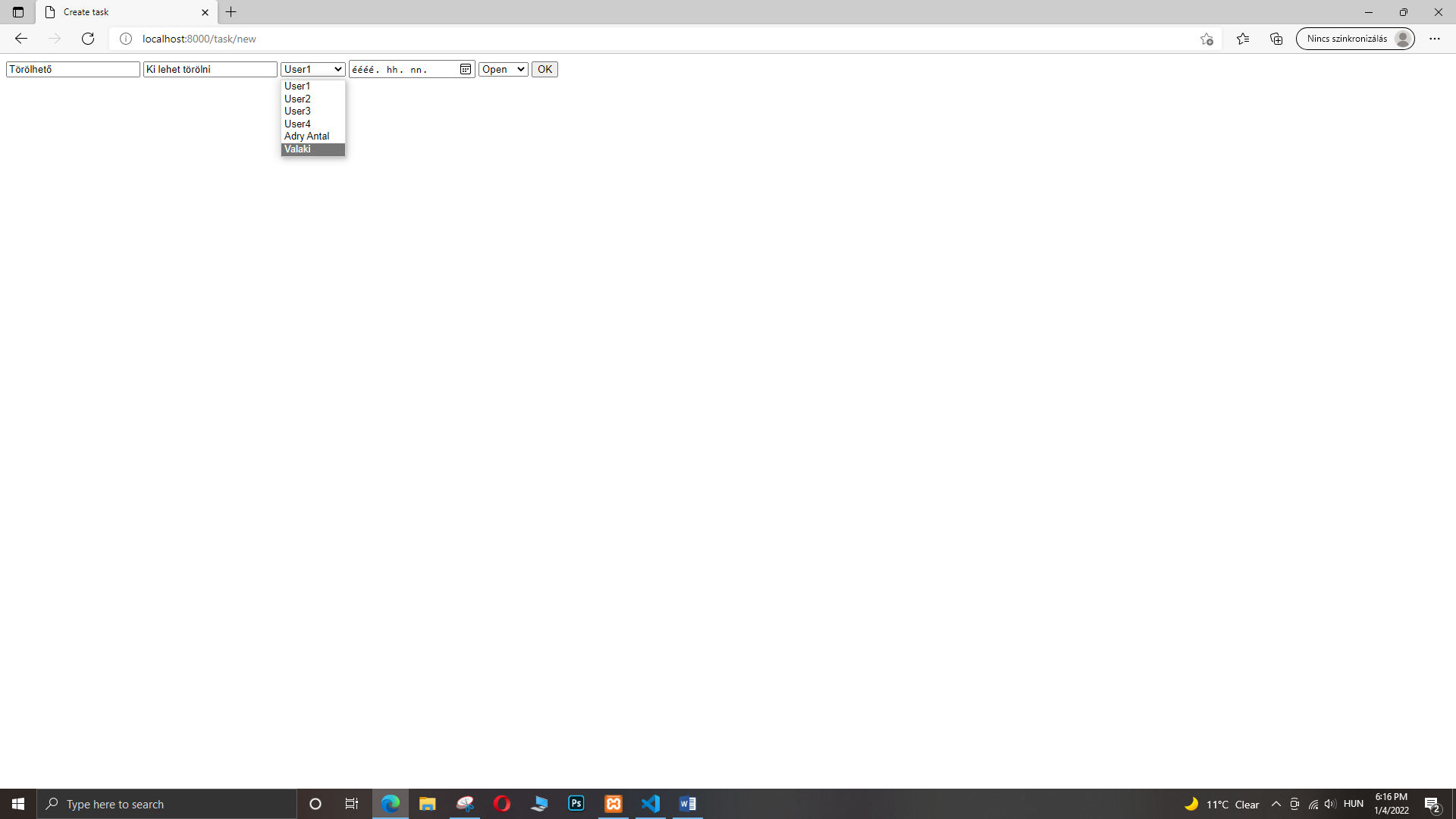
ErrorException

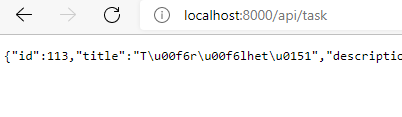
**Attempt to read property "id" on null**

<http://localhost:8000/api/task>



Jelen példában Valaki ([valaki@user.com](mailto:valaki@user.com), pw: 12345678) néven létrehoztunk egy új regisztrációt. Az újonnan létrehozott adatokkal belépve már betöltődik az oldal és funkcionál is:





Megjegyzés: az authentikáció előkészítéséhez elengedhetetlen a „Felhasználókezelés” fejezet a 2021-12-15-ös dokumentumban!