# TaskManager

## projekt megvalósításának lépési Laravellel

### Alapfeltételek:

PHP + Adatbázis (XAMPP csomag tartalmazza, DB: MySQL)

composer - php-s külső függőségek menedzseléséhez szükséges kis program.

Telepítés: https://getcomposer.org/download/

A composer parancs minden mappában elérhető parancs lesz

Futtatásakor mindig az aktuális parancssori könyvtárban fog dolgozni. ***composer install*** parancs esetén egy composer.json fájlt fog keresni.

### Laravel – keretrendszer

**Projekt létrehozás:**

laravel projektet a composer program segítségével tudunk létrehozni az alábbi parancs futtatásával:

**composer create-project laravel/laravel myfancyproject**

Laravel **projekt klónozása** meglévő repository-ból:

**.env.example** fájl tartalmának másolása .env fájlba

**.env** módosítása, hogy a kapcsolódási paraméterek legyenek élesek (DB kapcsolódási paraméterek beállítása)

**composer install**

**php artisan migrate**

**Laravel projekt elindítása:**

**php artisan serve**

*Tipp: Érdemes inicializálni a git repo-t, hogy könnyebben megtaláljuk, melyik fájlt módosítjuk éppen!*

## feladat

1. **Feladatok létrehozása**
2. **Feladatok listázása**
3. **Feladatok módosítása**
4. **Feladatok törlése**

Minden ***task***-hoz tartozik:

* + - név
    - leírás
    - állapot (folyamatban, kész, elvégzendő, törlendő)
    - határidő
    - ki fogja elkészíteni

### Táblák

|  |
| --- |
| **TASK** |
| id |
| title |
| description |
| end-date |
| user-id |
| status |

|  |
| --- |
| **USER** |
| user\_id |
| name |

#### TASK API végpontok és feladatok

GET all /api/tasks

GET id /api/task/id

POST /api/task

PUT id /api/task/id

DELETE id /api/task/id

#### USER API végpontok és feladatok

GET all /api/users

#### VIEW – ahol megjeleníthetem az adatokat

Task-ok listázása /task

Task módosítása /task/edit/1

Új Task létrehozása /task/create

## Lépésről lépésre

1. Indítsuk el a Xamp-ot.
2. Hozzunk létre egy laravel táblát
3. Ellenőrizzük, hogy a .env fájlban ez a táblanév van megadva!

A képen szöveg, képernyőkép, képernyő látható

Automatikusan generált leírás

### Adatbázis táblák létrehozása

1. Létre kell hozni egy modelt a Tasks táblához   
   php artisan make:model Task -m
2. Létre kell hozni egy controllert is a Tasks táblához   
   php artisan make:controller TaskController -m

Fájl elérési helyek:

migrációs fájlok: database/migrations

model fájlok: app/Models

controller fájlok: app/Http/Controllers

view fájlok: resources/views

1. database/migration útvonalon létrejött egy új migrátiós fájl egy CreateTasksTable osztállyal.   
   Az up() metódusában megadjuk a táblában használni kívánt mezőneveket a típusukkal és a méretükkel együtt. A mezőtípusokhoz itt találunk útmutatót: <https://laravel.com/docs/7.x/migrations>

public function up()

    {

        Schema::create('tasks', function (Blueprint $table) {

            $table->id();//bigIncrements('id')

            $table->string('title',50);

            $table->longText('description',255);

            $table->date('end\_date');

            //Külső kulcsok megadása

$table->unsignedBigInteger('user\_id')

            //$table->foreignId('user\_id');

            $table->foreignId('userId')->references('id')->on('users');

            $table->integer('status');

            $table->timestamps();

        });

    }

1. A tábla migrálása:   
   php artisan migrate
2. Ha módosul a tábla mezője, akkor vissza kell vonni:   
   php artisan migrate:rollback  
   Majd módosítani a tábla mezőit és újra futtatni (3. 4. pont)
3. Tábla módosítása: Új migrációt kell létrehozni, de most update-tel:  
   php artisan make:migration update\_tasks\_table  
   A képen szöveg látható

   Automatikusan generált leírás
4. Majd a módosítást is migrálni kell  
   php artisan migrate

### Api útvonalak előállítása

1. routes/web.php fájlban a Route-olás elkészítése

/\* TASK API végpontok és feladatok\*/

/\* GET   –all   /api/tasks \*/

Route::get('/api/tasks', [TaskController::class,'index']);

/\* GET  -1  /api/task/1 \*/

Route::get('/api/tasks/{id}', [TaskController::class,'show']);

/\* POST     /api/task \*/

Route::post('/api/tasks', [TaskController::class,'store']);

/\* PUT      /api/task/1 \*/

Route::put('/api/tasks/{id}', [TaskController::class,'update']);

/\* DELETE       /api/task/1 \*/

Route::delete('/api/tasks/{id}', [TaskController::class,'destroy']);

/\* User API végpontok és feladatok

GET     – all   /api/users \*/

Route::get('/api/users', [UserController::class,'index']);

/\* VIEW – ahol megjeleníthetem az adatokat \*/

/\* Task-ok listázása        /task\*/

Route::get('/task/new', [TaskController::class, 'newView']);

/\* Task módosítása      /task/edit/1 \*/

Route::get('/task/edit/{id}', [TaskController::class, 'editView']);

/\* Új Task létrehozása      /task/create \*/ \*/

Route::get('/task/list', [TaskController::class, 'listView']);

require \_\_DIR\_\_ . '/auth.php';

1. A web.php elején be kell állítani a fájlok elérési útvonalát

use App\Http\Controllers\TaskController;//TaskController elérési útja

use App\Http\Controllers\UserController;

### Végpontokhoz rendelt függvények megírása

1. TaskController megnyitása a App\Http\Controllers\TaskController útvonalról.
2. Csatoljuk be a Task és a User osztályokhoz tartozó útvonalakat. (namespace megadása után)

use App\Models\Task;

use App\Models\User;

1. Az előző pontokban megadott függvények megírása

public function index()

    {

        $tasks = response()->json(Task::all());

        return $tasks;

    }

    public function show($id)

    {

        $tasks = response()->json(Task::find($id));

        return $tasks;

    }

    public function destroy($id)

    {

        $tasks = Task::find($id)->delete();

        //return redirect('/task/list');

    }

    public function store(Request $request)

    {

        echo $request;

        $task = new Task();

        $task->title = $request->title;

        $task->description = $request->description;

        $task->end\_date = $request->end\_date;

        // $task->userId = $request->userId;

        $task->status = $request->status;

        $task->save();

        return Task::find($task->id);

    }

    public function update(Request $request, $id)

    {

        $task = Task::find($id);

        $task->title = $request->title;

        $task->description = $request->description;

        $task->end\_date = $request->end\_date;

        $task->user\_id = $request->userId;

        $task->status = $request->status;

        $task->save();

        return Task::find($task->id);

    }

1. A html-t tartalmazó útvonalak definiálása:

//A html-t tartalmazó útvonalakhoz tartozó függvények definiálása:

    public function newView()

    {

        $users = User::all();

        return view('task.new', ['users' => $users]);

    }

    public function editView($id)

    {

        $users = User::all();

        $task = Task::find($id);

        return view('task.edit', ['users' => $users, 'task' => $task]);

    }

    public function listView()

    {

        $tasks = Task::all();

        return view('task.list', ['tasks' => $tasks]);

    }

### A HTML kódok elkészítése

1. resources/view-s mappában blade.php fájlok készítése  
   Mivel a controllerben view('task.list', ['tasks' => $tasks]);így adtam meg a view-t, ezért új mappába rendezem a fájlokat (task)

A képen szöveg látható

Automatikusan generált leírás

1. A list.blade.php megjeleníti az adattábla rekordjait. VAGY NEM?????

A képen szöveg látható

Automatikusan generált leírás

1. new.blade.php – Új Task létrehozására alklamas formot tartalmazza

A hidden input mező szerepe, hogy a formban megadott POST method paraméteret felülírjukt PUT-ra.

Ezen kívül kell egy másik hidden input mező, ami az autentikációért felel. Innen tudjuk, hogy ki van belépve az oldalra és van jogosultsága adatokat írni az adatbázisba.

A képen szöveg látható

Automatikusan generált leírás

1. edit.blade.php – a Task módosítására alkalmas formot tartalmazza.

A képen szöveg látható

Automatikusan generált leírás