

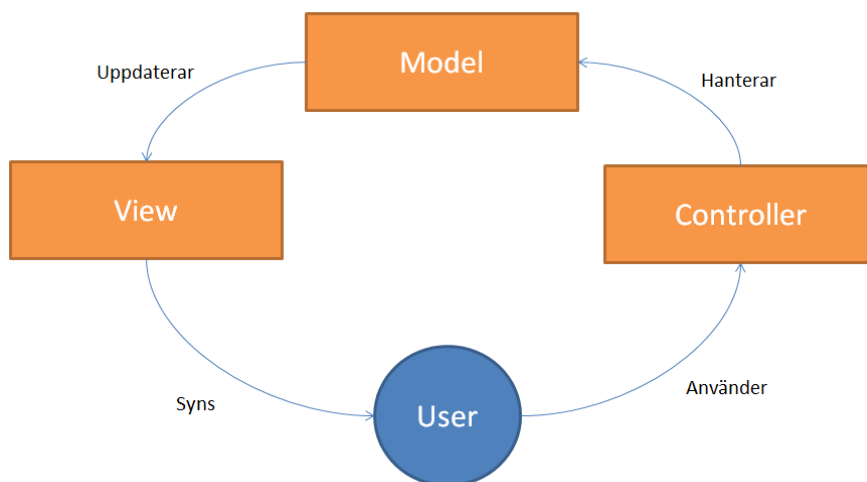
Workshop 7 – Laravel II (arbetsdokument)

Innehållsförteckning

Vad har vi gjort hittills?.....	2
Några resurser för Laravel.....	2
Hur arbetar vi med databaser i Laravel?.....	3
Uppgift 1 – Vi startar projekt och skapar en databas	3
Migration & schema builder.....	5
Uppgift 2 – Skapa migrationsfiler för vår databas	5
Tillbaka till routes, vyer och controller	7
Uppgift 3 – kopiera över vyer, controller och css-ramverket.....	7
Avslutningsvis	7

Vad har vi gjort hittills?

I tidigare workshop, Laravel I, bekantade vi oss med Laravel och MVC. Vi kikade på hur ett MVC-pattern kan användas för att ge oss en mer strukturerad kod vilket självklart blir allt viktigare om vi arbetar i lite större projekt, med många involverade utvecklare och designers. Ett viktigt inslag var routing och hur vi kunde dirigera användaren till rätt controller eller vy beroende på vad som efterfrågades (i uri-fältet). Vi arbetade med vyer för att på så vis visa innehåll för användaren. Vi arbetade även med en controller för att samla funktionalitet i en klass för en enkel CRUD-applikation. Ni fick med andra ord lite smak på hur Laravel hjälper er med att strukturera upp vår kod. Vi bekantade oss även successivt med Laravels filstruktur och de mappar som vi normalt arbetar i, framför allt i **app**.



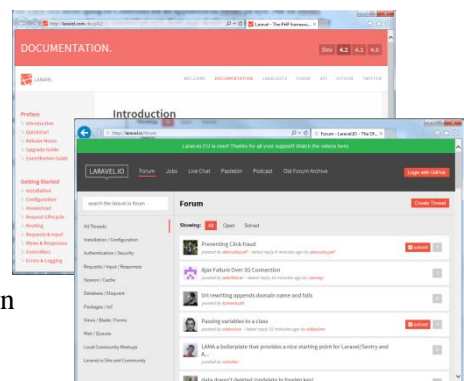
Figur 1: En schematisk figur över hur MVC fungerar.

I figur 1 ser ni en översiktsbild över hur det kan se ut när vi använder vårt MVC-ramverk Laravel. Controller har ni skapat (dirigerar), vyer har ni skapat (det som syns), men ni har hittills inte skapat modeller (data för vyer). Vi ska strax kika på detta med så alla komponenter i MVC faller på plats.

Några resurser för Laravel

Den viktigaste resursen som vi kommer att använda mest är Laravels webbplats, se <http://laravel.com/docs/4.2>. Här kan vi läsa om allt från installation till detaljer i referensmanualen. Det finns också ett forum som är mycket bra källa för lösningar på de problem som ofta dyker upp, se <http://laravel.io/forum>. Ytterligare ett forum som är användbart är: <http://laravel-recipes.com/>.

I boken Learning Laravel 4 Application Development är också en bra resurs att ha till hands. Inte minst för att komma igång med installation av Composer och Laravel. Det finns såklart tutorials också, t.ex denna <https://laracasts.com/series/laravel-from-scratch>.



Hur arbetar vi med databaser i Laravel?

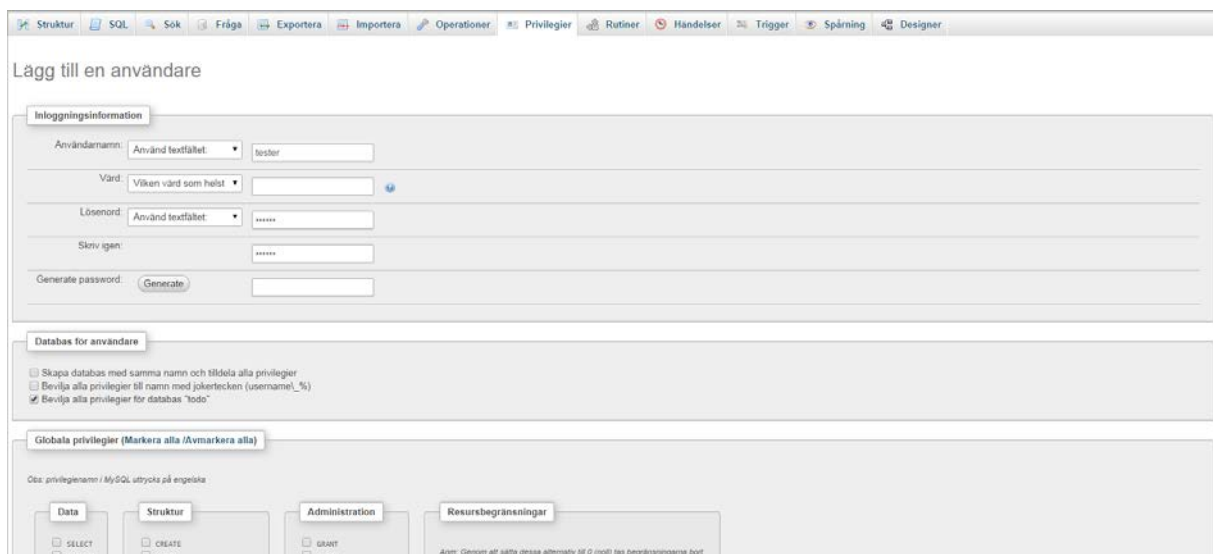
När vi arbetar i ett projekt med många designers och utvecklare gäller det att vara noga med hur vi arbetar med vår databas som ofta finns i anknytning till de projekt vi ska skapa. I Laravel försöker vi alltid att skapa **migrations** och **seeds** för att på så vis kunna skapa/ta bort tabeller och fylla på/ta bort data ifrån vår databas. Ett exempel på detta skulle kunna vara:

Åsa skapar en databas i MySQL och därefter tabeller i en migrationfil. Därefter lägger hon till lite data för att testköra applikationen så allt visas och fungerar så som det är tänkt. Efter Åsa är klar laddar hon upp sina filer till GitHub. Peter tar vid och laddar ner senaste projektfilerna från GitHub. Han ser att Åsa har uppdaterat databasen med tabeller och poster. Peter kör därför metoden som tar bort tabellerna i sin gamla databas (samma namn som Åsas) så den är helt tom. Därefter kör Peter Åsas migration- och seedfiler för att få exakt samma uppsättning tabeller och data som Åsa.

Vi ska arbeta med ett nytt projekt som är att skapa en att göra lista. Vi ska kunna lägga till, ändra och ta bort flera olika listor och i varje lista kunna lägga till, ändra och ta bort saker att göra. Utifrån exemplet ovan kikar vi på hur vi använder just **migration** och **seed** mot en MySQL databas.

Uppgift 1 – Vi startar projekt och skapar en databas

1. Skapa ett nytt laravelprojekt genom att öppna **GitBash**, t.ex. i roten på webservern, och kör därefter kommandot (byt ut klammrar och projektnamnet mot **todo**):
`composer create-project laravel/laravel [projektnamn] --prefer-dist`
2. Öppna **phpMyAdmin** och skapa en tom databas i **MySQL** (kollationering t.ex: utf8_swedish_ci eller utf8_unicode_ci beroende på om du vill arbeta med svensk eller engelsk sorteringsordning). Kalla databasen för **todo**.
3. Skapa en användare för databasen, du kan göra detta under privilegier för aktuell databas. Jag har skapat en användare som heter tester med lösenord tester som förses med **all privileges**, se figur 2.



Lägg till en användare

Inloggningsinformation

Användarnamn:

Vård:

Lösenord:

Skriv igen:

Generate password:

Databas för användare

☐ Skapa databas med samma namn och tilldel alla privilegier

☐ Bevilja alla privilegier till namn med jokertecken (username_%)

☒ Bevilja alla privilegier för databas "todo"

Globala privilegier (Markera alla /Avmarkera alla)

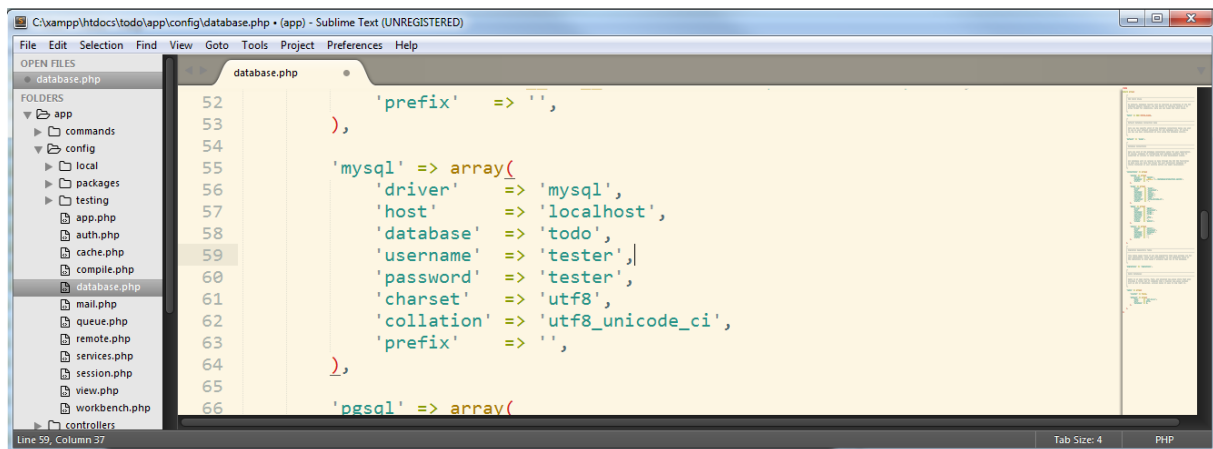
Obs: privilegier i MySQL uttrycks på engelska

Data	Struktur	Administration	Resursbegränsningar
<input type="checkbox"/> SELECT	<input type="checkbox"/> CREATE	<input type="checkbox"/> GRANT	
<input type="checkbox"/> INSERT	<input type="checkbox"/> ALTER	<input type="checkbox"/> SUPER	

Även: Genom att sätta dessa alternativ till 0 (noll) tas begränsningarna bort

Figur 2: Att lägga till användaren tester för databasen todo i fliken privilegier.

- Skapa en connection för MySQL i Laravel (kika i **/app/config/database.php**) och jämför med figur 3. OBS, om det finns en **/app/config/local/database.php** så behöver du konfigurera den med på liknande sätt som i figur 3.



Figur 3: Konfigurering av databasanslutning för databasen todo.

- Vi ska även sätta **'debug' => true**, i **/app/config/app.php**. I händelse av fel skrivs det ut information om var och vad som det är fel vilket underlättar felsökning då vi utvecklar.
- Om du skriver kommandot **hostname** i GitBash listas din dators namn. Kika så att det står rätt i filen **/bootstrap/start.php**. Min dator har hostname = IDE9660 vilket ger följande utseende:
`$env = $app->detectEnvironment(array(
 'local' => array('IDE9660'),
));`
 Du kan även kolla att det är local environment som du är i (finns testing, production med). Kör kommandot: **php artisan env**
 nu ska du få upp "local" om det är rätt. Du kan dubbelkolla att du konfigurerat databasnamn, användarnamn, password rätt för lokala miljön i **/app/config/local database.php**.
- Skapa nu en tabell i phpMyAdmin som vi döper till **todo_lists**.



Figur 4: Skapa en tabell för våra att göra listor.

- Lägg till 2 olika listor i tabellen todo_lists (det ska alltså finnas 2 poster). Observera att vi har auto_increment på id så den behöver du inte fylla i då du lägger till poster!
- För att testa vår databaskoppling kan vi skapa en route i filen **/app/routes.php** enligt följande:
`Route::get('/db', function(){
 return DB::select('select database();');
});`
- Testa nu att i webbläsaren skriva in urlen: **localhost/todo/public/db**
- Du borde få upp namnet på databasen om du gjort konfigureringen och databasen rätt, typ såhär: `[{"database()":"todo"}]`

12. För att lista tabellerna i databasen kan vi ändra i vår route i filen **/app/routes.php** enligt följande:

```
Route::get('/db', function(){
    return DB::select(show tables;');
});
```
13. För att lista posterna kan vi skriva om vår route i filen **/app/routes.php** enligt följande:

```
Route::get('/db', function(){
    return DB::table('todo_lists')->get();
});
```
14. Om du inte lagt till några poster ännu visas bara [], i annat fall får du upp de poster som finns i tabellen.
15. Vi vill nu lägga till en post till i tabellen och kan i filen **/app/routes.php** skriva följande:

```
Route::get('/db', function(){
    DB::table('todo_lists')->insert(array('name'=> 'Your List'));
    return DB::table('todo_lists')->get();
});
```
16. Vi vill ha ut namnet på de olika att göra listorna och får skriva så här i filen **/app/routes.php**:

```
Route::get('/db', function(){
    $result = DB::table('todo_lists')->where('name', 'Your List')->first();
    return $result->name;
});
```
17. I dokumentationen kan vi läsa mer om hur vi kan lista ifrån vår databas, se <http://laravel.com/docs/4.2/queries>
18. Bra jobbat!

Migration & schema builder

Vi har hittills skapat databaser och dess poster manuellt genom att använda phpMyAdmin. Allt eftersom projekten blir större och många olika personer är med då vi utvecklar projektet kan vi använda olika scheman och versioner för vår databas. Tar man bort en post där och uppdaterar en där blir snart datan förvanskad i databasen. Migration är ett sätt att skapa om databasen och dess poster med ett enkelt kommando.

Uppgift 2 – Skapa migrationsfiler för vår databas

1. Vi ska nu skapa en migrationsfil genom att köra kommandot:

```
php artisan migrate:make create_todo_lists_table --create=todo_lists
```
2. Vi öppnar filen **/app/database/migrations/20xx_xx_xx_xxxxx_create_todo_lists_table.php** i sublime text.
3. Som du ser finns det 2 funktioner, up() och down(). up() används när vi ska skapa och down() när vi ska ta bort. Man kan köra dem med **php artisan migrate** och **php artisan migrate:rollback**.
4. Vi ska nu skriva in de kolumner vi vill ha i vår tabell. För att se hur detta görs kan vi kika på <http://laravel.com/docs/4.2/schema#adding-columns>. Skriv in följande:

```
public function up()
{
    Schema::create('todo_lists', function(Blueprint $table)
    {
        $table->increments('id');
        $table->string('name')->unique();
        $table->timestamps();
    });
}
```

- Id är auto increment och vi har lagt till villkoret unique för att todo-listorna inte ska kunna ha likadana namn. Sist ligger en tidsstämpel för att se när en todo-lista har skapats.
- I funktionen down() skriver vi in följande:

```
public function down()
{
    Schema::drop('todo_lists');
}
```

- Testa nu att skapa tabellen genom kommandot: **php artisan migrate**
Kika i phpMyAdmin om det dök upp någon tabell...
- Testa även: **php artisan migrate:rollback**
Kika i phpMyAdmin om tabellen försvann...
- För varje todo-lista måste de kunna finnas saker att göra (items)! Vi ska därför skapa ytterligare en tabell som är relaterad till den tidigare **todo_lists**. Vi skapar en migration igen: **php artisan migrate:make create_todo_items_table --create=todo_items**
- Vi öppnar filen
/app/database/migrations/20xx_xx_xx_xxxxx_create_todo_items_table.php i sublime text.
- Skapa följande kolumner för function up():

```
public function up()
{
    Schema::create('todo_items', function(Blueprint $table)
    {
        $table->increments('id');
        $table->integer('todo_list_id');
        $table->string('content')->unique();
        $table->dateTime('completed_on')->nullable();
        $table->timestamps();
    });
}
```

- Vi har skapat en främmande nyckel (todo_list_id) för att kunna relatera tabellerna.
- Vi skapar även följande för function down():

```
public function down()
{
    Schema::drop('todo_items');
}
```

- Då var det dags att skapa en 1:M samband mellan tabellerna, se <http://laravel.com/docs/4.2/eloquent#one-to-many>. En todo-lista kan ha många items och ett item hör till en tod-lista.
- Vi skapar en modellfil **TodoList.php** för att relatera tabellerna. Skapa alltså en ny fil: **/app/models/TodoList.php** i sublime text. Skriv in följande:

```
<?php
class TodoList extends Eloquent {

    public function listItems()
    {
        return $this->hasMany('TodoItem');
    }

    public function delete()
    {
        TodoItem::where('todo_list_id', $this->id)->delete();
        parent::delete();
    }
}
```

- I den första funktionen talar vi om att varje listItems finns många TodoItem. I den andra funktionen tar vi bort alla items om vi tar bort en todo-lista.

17. Vi behöver även skapa en modelfil för **TodoItem** på liknade sätt. Skapa alltså en ny fil: **/app/models/TodoItem.php** i sublime text. Skriv in följande:

```
<?php
class TodoItem extends Eloquent {

    public function todoList()
    {
        return $this->belongsTo('TodoList');
    }
}
```

18. I denna funktion talar vi om att ett **TodoItem** tillhör en **TodoList**.
19. Då var det klart.

Vi har nu skapat databasen och de tabeller som behövs med migration. Det är nu dags att testa applikationen med de färdiga vyerna och kontrollern, se uppgift 3.

Tillbaka till routes, vyer och controller

Vi har nu ordnat med databasen och migrations. Nu är det dags att kopiera in filer för att kunna testköra applikationen. Ni hittar zip-filerna på Bb.

Uppgift 3 – kopiera över vyer, controller och css-ramverket

1. För att stylea vår applikation kan vi **/public/** mappen packa upp filen **copy_into_public.zip**, som ni hittar på Bb. Den innehåller mapparna css, img och js – d.v.s. ramverket foundation.
2. Kopiera nu in **TodoListController.php** genom att packa upp filen **copy_into_controllers.zip** i foldern **/app/controllers/**.
3. Vi modifierar vår **routes.php** i **/app/** mappen så den har följande routes:

```
Route::get('/', 'TodoListController@index');
Route::resource('todos', 'TodoListController');
```

4. Om du är osäker på om du skapat modelfilerna rätt kan du även hämta ner dessa ifrån Bb. Den filen heter, filen heter **copy_into_models.zip**. Packa upp dem i mappen **/app/models/**.
5. Det som återstår är de vyer som behövs för visning av innehållet. Ladda ner filen **copy_into_views.zip** ifrån Bb och packa därefter upp den i **/app/views/**. De ska vara 2 undermappar i views, **layouts** och **todos**. Inne i dessa undermappar finns de olika vyerna som används. Det kan vara bra att kika närmre även på hur blade fungerar så ni kan få till vyerna, se <http://laravel.com/docs/4.2/templates>. Ni kan såklart kika även på de exempel som ni nu laddat ner för att se exempel på användning av blade.
6. Då återstår det att testa att det fungerar att lista alla todo-listorna och de items som finns på respektive. Testa även att skapa en ny todo-lista och därefter att ta bort densamma. Om allt fungerar har du klarat dagens workshop finfint! Bra!

Avslutningsvis

Vi har nu arbetat lite mer med Laravel så att ni kan komma igång med era egna projekt lite bättre. Det finns förstås mycket mer att lära och utforska i Laravel och ett tips är att läsa igenom eller surfa runt på <http://laravel.com/docs/4.2/quick> så mycket ni kan för att lära mer.