Laravel

Deze les

- Sessions
- Validation
- Migrations
- Query builder
- Eloquent
- Database seeder
- Opdracht

Sessions

http://laravel.com/docs/5.1/session

```php
// Met Request object
$name = $request->session()->get('name');
$request->session()->put('name', 'Leon');

// Een value in een session array zetten.
$request->session()->push('employees', 'Leon');



// Met laravel helper
$value = session('name');
session(['name' => 'Leon']);


// Kijk of een sessie een value heeft
return $request->session()->has('users');
```

Validation

http://laravel.com/docs/5.1/validation

Als er data wordt verstuurd(zowel POST en PUT) wil je er zeker van zijn dat je de juiste data binnen krijgt. Daarom is het goed om de data te valideren voor dat je de data opslaat in de database. Validation kun je op de volgende manieren uitvoeren:

```php
// Direct een request valideren
$this->validate($request, [
    'num1' => 'required|digits:0,100',
    'num2' => 'required|digits:0,100',
    'calc_method' => 'required|in:plus,minus,multiply,divide'
]);

// Met de Validator Facade(voor gebruik buiten controllers)
$validator = Validator::make($request->all(), [
    'num1' => 'required|digits:0,100',
    'num2' => 'required|digits:0,100',
    'calc_method' => 'required|in:plus,minus,multiply,divide'
]);

if ($validator->fails()) {
    return redirect('calculator')->withErrors($validator)->withInput();
}
```

Migrations

1/migrations

http://laravel.com/docs/5.

Migrations zijn versie beheer voor je database. Met migrations kun je het datamodel van je database gemakkelijk delen met andere personen binnen je team. Een migration kun je maken met het artisan commando: `php artisan make:migration add_votes_to_users_table --table=users`

Om vervolgens de migration uit te voeren kun je het commando `php artisan migrate` gebruiken. Om de oude database te resetten kun je het commando `php artisan migrate:reset` gebruiken.

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

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

Query builder

http://laravel.com/docs/5.1/queries

In Laravel maak je niet direct gebruik van PDO of Mysqli. In plaats daarvan kun je gebruik maken van bijvoorbeeld de Query builder. Het gebruik van de Query builder is te zien in de volgende voorbeelden(CRUD):

```php
// Create
DB::table('users')->insert([
    ['name' => 'Leon', 'email' => 'leon@noorderpoort.com' , 'votes' => 200],
    ['name' => 'Reygan', 'email' => 'reygan@noorderpoort.com' , 'votes' => 0]
]);

// Read
$users = DB::table('users')->get();
$users = DB::table('users')->where('name', 'Reygan')->first();
$users = DB::table('users')->where('votes', '>=', 100)->get();
$users = DB::table('users')->orderBy('name', 'desc')->get();

// Update
DB::table('users')->update(['votes' => 0]);

// Delete
DB::table('users')->where('votes', '<', 100)->delete();
```

Eloquent

Wat is Eloquent
Eloquent is een implementatie van Active Record. In active record maak je gebruik van models waar in je de relaties, database naam en invulbare velden vast legt. Vervolgens kun je met de query builder functies data uit je database halen(Via je Eloquent class). Een Eloquent “read” returned altijd een Collection.

Model aanmaken
`php artisan make:model Locations` (Gebruik de -m flag om ook direct een migration aan te maken)

Eloquent class

```php
// User model
class User extends Model
{
      // Database tabel naam
      protected $table = 'users';
      // De database collommen die "Mass fillable" zijn
      protected $fillable = ['name', 'email', 'password'];
      // Verborgen van JSON output
      protected $hidden = ['password'];
}

// User model aanroepen
$users = User::all();
$users = User::where('name', 'LIKE', '%L')->get();

// Mass fillable
$user = User::create([
    'name' => 'Foo',
    'email' => 'leon@noorderpoort.com',
    'password' => Hash::make($password)
]);

User::findOrFail(1)->delete();
```

```php
// Update
$user = User::findOrFail($id);
$user->name = 'Henk';
$user->save();

// Create
$user = new User;
$user->name = 'Henk';
$user->email = 'henk@noorderpoort.nl';
$user->password = Hash::make($password);
$user->save();
```

Eloquent relaties

```php
// in User model
public function posts()
{
    return $this->hasMany(Post::class);
}
// In Post model
public function user()
{
    return $this->belongsTo(User::class);
}


$user_posts = User::find($this->user_id)->posts;
$user = User::with('posts')->where('id', $this->user_id)->get();

$post::find(1)->user;
$post::with('user')->where('id', 1)->get();
```

Database seeder

Om in een lege database test informatie toe te voegen kun je gebruik maken van database seeds.In een databse seed kun je zowel Eloquent als de Query builder gebruiken om data te inserten.

Maak een Database Seeder

```
php artisan make:seeder UsersTableSeeder
```

Vervolgens kun je in het aangemaakte bestand(in: database/seeders map) in de methode run aangeven welke database records aangemaakt moeten worden.

Vervolgens `$this->call(PostsTableSeeder::class);` toevoegen aan DatabaseSeeder.php om deze database seed te kunnen uitvoeren. Een database seed kun je vervolgens uitvoeren met het commando:

```
php artisan db:seed
```

Opdracht

Maak de volgende database structuur met migrations. Maak vervolgens 1 database seed waarmee je 1 standaard gebruiker toevoegt. Maak vervolgens 1 route aan waarmee je alle gebruikers ophaalt inclusief de posts die deze gebruik heeft. En maak 1 route aan waarmee je een nieuwe post voor een user kan toevoegen(aan de hand van user_id).

Maak gebruik van:

- Migrations
- Database seeds
- Models(met relaties)
- Validator(required)
-