

# Agil PHP med CodeIgniter

Ved Per Sikker Hansen

## Indholdsfortegnelse

- Introduktion
- Model, View, Controller
- Object Oriented Programming
- Agile Development
- CodeIgniter
- Lad os komme igang
- Opsætning af CodeIgniter
- Filstrukturen
- Controllers og Views
- Models
- Konventioner
- Øvelse: blogsystem
  - Databasen
  - Oprettelsesform
  - Liste over indlæg
  - Læs et indlæg
  - Slet/rediger indlæg
- Afrunding
- Videre læsning

## Introduktion

Denne artikel er skrevet med fokus på den erfarne PHP-udvikler, der er træt af at gentage sig selv hver gang han/hun starter et nyt PHP-projekt, og bare gerne vil have noget fra hånden hurtigt, pænt og effektivt. Der vil derfor ikke blive taget særligt meget hensyn til nybegynderen indenfor PHP, men begreber såsom MVC og OOP vil blive kort forklaret for at alle kan være med. Hvis du er ny til PHP bør du læse en artikel rettet imod at komme igang med sproget.

Artiklen lægger ud med nogle teoretiske afsnit, der forklarer lidt om tankerne bag systemet og hvordan strukturen er bygget op. Føl dig fri til at springe de afsnit over hvis du bare gerne vil se noget kode.

## Model, View, Controller

En forkortelse du måske er stødt på før er MVC, der står for Model, View, Controller. Dette er et udviklingsmønster der har vundet indtog indenfor moderne udvikling de sidste år, på grund af dens strukturerede natur og mulighed for at adskille logik, præsentation og databaseabstraktion.

- Models er datalag, der håndterer kommunikationen med den rå information der kan ligge i fx en database, flade filer eller på en webservice. Udover at snakke med databasen sørger models også for den dataspecifikke behandlingslogik. For eksempel er det modellens opgave at sammenligne passwords ved login, eller at slette brugerens private beskeder sammen med brugeren.
- Views er præsentationslaget, hvor dataene der bliver trukket ud igennem modellen bliver vist på skærmen for brugeren, for det meste spundet ind i (X)HTML/CSS.
- Controllers er adgangslaget, der holder styr på hvad det er brugeren gerne vil se, kontakter modellen for at se om dataene findes og sender de behandlede data videre til et view, der præsenterer det hele på skærmen.

Fordelen ved dette mønster er, at alting foregår så modulært, at den ene del kan skiftes ud, uden at resten af applikationen berøres - fx kan rutinen til at hente en liste af brugere ændres til at hente fra en anden tabel i User modellen, hvorefter ændringen vil træde i kraft automatisk på alle sider der har noget med brugere at gøre.

## Object Oriented Programming

Du har næsten med garanti stødt på begrebet Objektorienteret Programmering, eller forkortet OOP, i dine google endeavours, og du har ganske sikkert også arbejdet i det. Hvis du ikke har, anbefales det at du læser op på det, da moderne programmering generelt og CodeIgniter specifikt, efterhånden er uhensigtsmæssigt/næsten umuligt, uden. CodeIgniter benytter sig dog af PHP3/4's objektmodel, for at have support for shared hosting servere der ikke er gået over til PHP5 endnu, hvilket giver en simplere syntax der er lettere at forstå for en OOP-begynder.

## Agile Development

Agile development, eller "agil udvikling" som man siger på dansk, er en fremgangsmåde indenfor programmering der handler om at udvikle hurtigt og effektivt. Når man udvikler agilt fokuserer man på at udvikle de features der bliver vigtige at have med før andre features, istedet for at prøve at lave alle features på én gang. På den måde sparer man tid og kræfter man ville spille på features, der måske viser sig irelevante når det første er på plads. Agil udvikling betyder naturligvis ikke at man ikke skal planlægge og tænke fremad, men fokus skal ligge på de mest presserende features. At udvikle agilt kan opnås ad forskellige stier, og at bruge et framework der fjerner dobbeltarbejdet er et godt skridt på vejen.

## CodeIgniter

CodeIgniter er et open source framework, en samling færdiglavede klasser og funktioner, til agil udvikling af modulære PHP-applikationer. Det bliver vedligeholdt af firmaet EllisLab, og er i og for sig en klon af Ruby on Rails frameworket. Det er sammen med Zend Framework, Symfony og CakePHP meget populært blandt seriøse PHP-udviklere. KohanaPHP er et projekt afledt af CodeIgniter, med fokus på en mere decentraliseret udviklingsmodel end den firmabaserede man ser hos EllisLab. KohanaPHP er sidenhen blevet skrevet forfra fra bunden, men ligner stadig meget CodeIgniter.

Denne artikel vil give et indblik i hvordan man kommer igang med at udvikle opad CodeIgniter, men skal langt fra ses som en komplet bibel udi CI-udvikling. Til dette formål har CodeIgniter sin egen yderst velskrevne [User Guide](#).

## Lad os komme igang

Mindre snak, mere action. For at komme igang med CodeIgniter skal du bruge følgende:

- En fungerende webserver, enten egen server, et webhotel eller en lokal installation. Har du ingen af delene kan XAMPP, en færdigsamlet pakke af Apache-serveren, Mysql-databasen og PHP-sproget der kan køre lokalt, downloades fra <http://www.apachefriends.org/en/xampp.html> i varianter til både Linux, Windows og Mac.
- En kopi af CodeIgniter's kildekode. Den kan hentes fra CodeIgniter.com.
- En teksteditor der kan gemme i UTF-8
- Hvis du bruger en ekstern server skal du også bruge en (S)FTP-klient eller tilsvarende for at overføre filerne.

## Opsætning af CodeIgniter

Når du har sikret dig at din webserver fungerer, og har hentet zipfilen med CodeIgniter i, pakker du den ud i din servers mappe, og omdøber den resulterende mappe til at hedde det, som din applikation skal kaldes. Fx MinApp.

Nu er CodeIgniter allerede kørende, og du kan browse ind på <http://dinServerUrl/MinApp/> og se CodeIgniter's velkomstsider. Den vil se nogenlunde således ud:

### Welcome to CodeIgniter!

The page you are looking at is being generated dynamically by CodeIgniter.

If you would like to edit this page you'll find it located at:

```
system/application/views/welcome_message.php
```

The corresponding controller for this page is found at:

```
system/application/controllers/welcome.php
```

If you are exploring CodeIgniter for the very first time, you should start by reading the [User Guide](#).

Page rendered in 0.6023 seconds

## Filstrukturen

Som du kan se henviser velkomstsiden til nogle filer der har med dit system at gøre. Som du kan se

på billedet nedenfor indeholder CodeIgniter en del filer og mapper, og for begynderen kan det være svært at regne ud, hvad de forskellige ting står for.



Det er derfor på tide at give et hurtigt indblik i, hvad de forskellige filer og mapper betyder:

- system : Indeholder din CodeIgniter-installation og al den generiske CodeIgniter-kode
- system/application : Indeholder din egen CodeIgniter kode
- system/application/controllers : Indeholder dine adgangs-filer
- system/application/models : Indeholder dine datalag
- system/application/views : Indeholder din præsentation
- system/application/config : indeholder konfigurationsvariabler til at indstille på de forskellige dele af CodeIgniter, deriblandt basale konfigurationer, hvilke libraries der skal loades automatisk og lignende.

Vi tager her udgangspunkt i de filer og mapper du kommer til at bruge i øbet denne artikel. Til sidst i artiklen vil de mapper og filer vi efterlader til din videre læsning kort blive ridset op.

Som du kan se ligger applikationen og systemet for sig. Dette er gjort for at sørge for, at du kan opdatere din version af CodeIgniter, uden at dine applikationsspecifikke filer bliver berørt.

### **Controllers og Views**

Lad os komme igang med at pille ved noget kode. Som du kan se på velkomstsiden ligger der nogle

filer der bliver brugt ligenu, i henholdsvis system/application/views/welcome\_message.php og system/application/controllers/welcome.php. Efter deres filnavne burde det være åbenlyst hvilket formål de hver især tjener. Hvis vi starter med at åbne system/application/views/welcome\_message.php vil vi blive mødt af følgende kildetekst:

```
<html>
<head>
<title>Welcome to CodeIgniter</title>

<style type="text/css">

body {
background-color: #fff;
margin: 40px;
font-family: Lucida Grande, Verdana, Sans-serif;
font-size: 14px;
color: #4F5155;
}

a {
color: #003399;
background-color: transparent;
font-weight: normal;
}

h1 {
color: #444;
background-color: transparent;
border-bottom: 1px solid #D0D0D0;
font-size: 16px;
font-weight: bold;
margin: 24px 0 2px 0;
padding: 5px 0 6px 0;
}

code {
font-family: Monaco, Verdana, Sans-serif;
font-size: 12px;
background-color: #f9f9f9;
border: 1px solid #D0D0D0;
color: #002166;
display: block;
margin: 14px 0 14px 0;
padding: 12px 10px 12px 10px;
}

</style>
</head>
<body>

<h1>Welcome to CodeIgniter!</h1>

<p>The page you are looking at is being generated dynamically by
CodeIgniter.</p>

<p>If you would like to edit this page you'll find it located at:</p>
<code>system/application/views/welcome_message.php</code>

<p>The corresponding controller for this page is found at:</p>
<code>system/application/controllers/welcome.php</code>
```

```
<p>If you are exploring CodeIgniter for the very first time, you should
start by reading the <a href="user_guide/">User Guide</a>.</p>
```

```
<p><br />Page rendered in {elapsed_time} seconds</p>
```

```
</body>
</html>
```

Du kan prøve at ændre lidt i filen og se dem slå igennem på velkomstsiden. Dette er ikke videre avanceret, så lad os prøve at kigge på controller-koden, og se hvad der sker derinde i “motorrummet”:

```
<?php

class Welcome extends Controller {

    function Welcome()
    {
        parent::Controller();
    }

    function index()
    {
        $this->load->view('welcome_message');
    }
}

/* End of file welcome.php */
/* Location: ./system/application/controllers/welcome.php */
```

Det er jo næsten skræmmende simpelt. Læg iøvrigt mærke til at filen ikke lukkes med et `>` tag. Dette er for at undgå whitespace (det kan være et mellemrum eller linieskift efter et afsluttende `>` tag i en vilkårlig fil et sted), der kan forårsage problemer med det outbutbuffering-system som CodeIgniter gør brug af. Istedet afsluttes filen med en erklæring om at filen er slut, hvad den hedder, og hvor den ligger. Det anbefales kraftigt at denne konvention følges. Mere om dette og andre konventioner i afsnittet **konventioner**.

Hvis vi gennemgår hvad der sker i Controlleren kan vi se at den til at starte med bliver erklæret med navnet `Welcome`, og er en udvidelse af `Controller` klassen, der er en integreret del af CodeIgniter. At den er en `Controller` betyder at den bliver til CodeIgniters superobjekt - det objekt som hele applikationen kommer til at køre igennem. Når Controlleren instantieres bliver diverse libraries loadet, urlens segmenter bliver filtreret for XSS-forsøg og stillet til rådighed gennem metoden `$this->uri->segment(n)`, samt diverse andre småting der er rare at have. Hvis du har tilføjelser til denne procedure kan du definere dem i controllerens konstruktør, i dette tilfælde `welcome()` - gør du det skal du huske, som det også bliver vist i eksemplet, at køre `parent::Controller()` for at få den basale Controller-opstartslogik med også. Når det er gjort bliver den requestede “action” kaldt som metode. En action skrives i urlen. Hvis ingen action er defineret, kaldes `index()` som default. Hvis du ikke kører med url rewriting vil et kald til velkomstsiden se således ud:

<http://dinServerUrl/MinApp/index.php/welcome/index>

I det ovenstående eksempel vil `$this->uri->segment(1)` returnere “welcome”, og `$this->uri->segment(2)` vil returnere “index”.

Prøv at jøk lidt rundt i controllerkoden og ret lidt hist og pist, eventuelt smide nogle echo statements ind eller prøve at kalde nogle andre view-filer, og se hvordan det går. Når du føler dig klar kan du gå videre til næste afsnit.

## Models

Nu har vi kigget lidt på hvordan output er struktureret, nu skal vi kigge lidt på hvor vi får vores data fra. En model er i og for sig ikke nødvendig for en CodeIgniter applikation. Det er et view i grunden heller ikke, da man bare kan echo'e de data der skal ud. Men ligesom views er models en rigtig god idé i projekter af stortset alle størrelser, for at gøre det så enkelt som muligt at skifte dele af logikken ud uden at røre ved resten. I CodeIgniters eksempelkode bliver der ikke brugt en model, så lad os lave en selv, der bare returnerer noget flad data til at starte med. Skriv følgende kode i din editor og gem filen som `welcome_model.php` i mappen `system/application/models`:

```
<?php

class Welcome_model extends Model {

    function Welcome_model()
    {
        parent::Model();
    }

    function get()
    {
        return 'Hello, World!';
    }
}

/* End of file welcome_model.php */
/* Location: ./system/application/models/welcome_model.php */
```

Den ovenstående fil er meget, meget basisk, men tjener sit formål ganske udmærket – den definerer en model der hedder Welcome, har en konstruktør der kalder Model-klassens konstruktør, hvorefter din egen kode kan tilføjes til konstruktøren hvis du ønsker det.

Ligenu gør vores model ikke så meget, da den bare ligger i mappen uden at blive hentet nogen steder. Åbn din `welcome.php` controller igen og tilføj `$this->load->model('welcome_model');` til din konstruktør, efter kaldet af parent-konstruktøren, for at hente modellen ind. Nu kan modellen kaldes fra objektet `$this->welcome_model`.

Nu har vi mulighed for at hente data ud fra vores model – omend det endnu er noget simple data vi har med at gøre – og få dem ind i vores controller. Nu skal vi have fundet ud af at skubbe de data videre til vores view, så brugeren kan se dem. Dette gøres ved at smide dem ind i et array, som du sender som argument til `$this->load->view()`. Efterhånden skulle din controller-fil se således ud:

```
<?php

class Welcome extends Controller {

    function Welcome()
    {
        parent::Controller();
        $this->load->model('welcome_model');
    }
}
```

```

function index()
{
    $data['welcome'] = $this->welcome_model->get();
    $this->load->view('welcome_message', $data);
}

/* End of file welcome.php */
/* Location: ./system/application/controllers/welcome.php */

```

Dette er imidlertid ikke nok, for viewet ved ikke at dataene kommer endnu. Derfor skal vi åbne `welcome_message.php` og tilføje følgende kode i `body`-elementet:

```

<p>Resultatet fra vores model er:</p>
<code><?php echo $welcome; ?></code>

```

Gem filen, og test den i din browser. Du har nu en fin Hello, World! besked i dit view, der kommer fra modellen.

Resultatet fra vores model er:

Hello, World!

Dette kaldes at pushe data til viewet gennem controlleren, og fungerer ved at det array eller objekt du giver til viewet bliver splittet op til enkeltstående variabler og skubbet ind i koden, hvor de så kan echoes, loopes igennem etc. som normalt. Tillykke, du kender nu til det basale i CodeIgniter, og kan i princippet gå igang med at skrive applikationer! Inden da kan det dog være smart at læse det følgende afsnit, der fortæller om hvilke kodekonventioner du bør følge når du udvikler i CodeIgniter, for at din kode er forståelig for andre, samt for at undgå fejl. Efterfølgende vil du blive sat igang med en øvelse, hvor artiklen vil guide dig igennem udviklingen af en meget simpel weblog.

## Konventioner

Som du nok har bemærket følger CodeIgniters filer nogle regler for, hvordan ting skal navngives, og hvordan filer skal afsluttes. Det kan dog være lidt forvirrende med de forskellige konventioner, da de er anderledes alt efter situation. Jeg vil her ridse navngivningsreglerne op i forståeligt dansk.

- Flere ord i et fil-, klasse- eller funktionsnavn symboliseres med en underscore: “\_”
  - Rigtigt: `code_igniter`
  - Forkert: `codeIgniter`, `CodeIgniter`, `Code igniter`
- Klasser navngives med stort forbogstav og resten småt
  - Rigtigt: `welcome_model`
  - Forkert: `welcome_Model`, `Welcome_Model`, `welcome_model`
- Metoder navngives med små bogstaver hele vejen igennem
  - Rigtigt: `index()`
  - Forkert: `Index()`
- Filer med klasser navngives efter den klasse der ligger i den. Filnavne har kun små bogstaver.



- Rigtigt: `welcome_model.php`
- Forkert: `welcome_model.php`, `welcome_model.php`
- Model-klasser navngives med “\_model” som suffix.
  - Rigtigt: `welcome_model`
  - Forkert: `welcome`, `welcomemodel`

Som nævnt i afsnittet **Controllers og Views** afsluttes filer med ren PHP-logik ikke med et `?>`, men snarere med to linier kommentarer, der fortæller hvad filen hedder, og hvor den ligger i filtræet, efter denne prototype:

```
/* End of file fil_navn.php */
/* Location: ./system/application/controllers/fil_navn.php */
```

Dette er som nævnt for at undgå problemer med whitespace, der bliver anset som output af serveren, hvilket vil sende dataene til browseren udenom output bufferen som CodeIgniter bruger – resulterende i en blank side der er svær at debugge.

Til indrykning benyttes tabulator-tasten, istedet for mellemrumstasten. Dette gør det nemmere for den enkelte udvikler at indstille størrelsen på indrykningerne som det passer vedkommende, i sin egen editor, uden at det har betydning for andre udvikler.

Filer bør gemmes med tegnsættet UTF-8 for at understøtte flest mulige sprog og systemer, og for at undgå problemer med at nogle filer er det ene, og andre det andet.

CodeIgniter gør ydermere, som nævnt, brug af PHP4-syntaksen til metoder og egenskaber. Det betyder at istedet for at angive et access-niveau på en egenskab eller metode, angives enten `var` eller `function` keywords, respektivt.

Sidst men ikke mindst anvendes et lineskift efter funktionserklæringer/ifsætninger og lignende, inden den åbnende og lukkende tuborgklamme. Den eneste undtagelse herfor er classes, hvor den åbnende tuborgklamme er på samme linie som klassedefinitionen.

```
Rigtigt:
    if($foo == 'bar')
    {
        echo 'variablen $foo er lig med bar';
    }
Forkert:
    if($foo == 'bar') { echo 'variablen $foo er lig med bar'; }
```

### Øvelse: Blogsystem

Nu har vi gennemgået de basale begreber indenfor CodeIgniter's MVC, og har lært hvordan vi redigerer models, controllers og views, og får dem til at kommunikere sammen. Vi har sprunget lidt let hen over nogle af forklaringerne – det har indtil nu været tilstrækkeligt at det virkede. Nu skal vi til at lære lidt om *hvordan* det virker, så vi kan bruge det til noget fornuftigt. Derfor vil artiklen nu guide dig igennem udviklingen af en simpel weblog, der kan oprette, liste, læse, redigere og slette posts, og på vejen vil vi kigge nærmere på de forskellige libraries og metoder der bliver taget i brug.

**BEMÆRK: undervejs i denne øvelse vil urlen `http://dinServerUrl/` blive brugt istedet for din egen url. Sørg for at skifte denne url ud inden du kører eksemplerne!**

## Øvelse: Blogsystem - Databasen

Før at have en weblog kørende skal vi have en eller anden form for database op at stå. Artiklen tager udgangspunkt i en MySQL-database, men CodeIgniter kan indstilles til at bruge både PostgreSQL, Sqlite, Oracle, MSSQL m.v i konfigurationen. Derudover tilbyder CodeIgniter en databaseabstraktion der gør det let at skrive database-uafhængige queries. Det vil vi alt sammen se nærmere på senere – vi fokuserer på det der ligger umiddelbart foran os, og det er at få lavet databasen. Resten ser vi på senere - agil udvikling.

Udfør følgende SQL-sætning i din database (skal eventuelt rettes lidt til for at passe, hvis du bruger noget andet end MySQL). Hvis du bruger PHPMyAdmin kan du gøre det via SQL-funktionen i interfacet:

```
CREATE TABLE `codeigniter_blog` (  
  `id` BIGINT UNSIGNED AUTO_INCREMENT NOT NULL,  
  `title` VARCHAR(255) NOT NULL,  
  `body` TEXT NOT NULL,  
  PRIMARY KEY (`id`)  
);
```

Nu har vi tabellen, så skal vi fortælle CodeIgniter at den findes. Det gør vi ved at gå ind i system/application/config/database.php og ændre lidt på konfigurationsvariablerne. Som du kan se er MySQL valgt som default, og det er jo fint. Variablerne burde være selvforklarende, så ret dem til så de passer til dine specifikationer, og gem filen igen. Nu er databasen sat op, og vi kan komme igang med at producere kode!

## Øvelse: Blogsystem – Oprettelsesform

Vi skal igang med vores blog, og førend vi kan læse eller liste data, er vi nødt til at have noget data at arbejde med. Derfor kan vi ligeså godt starte med oprettelsesdelen – det er agilt, det er smart, og det er nemt.

Lad os analysere situationen engang: For at kunne oprette blogindlæg skal vi bruge en database, den har vi lavet, en funktion der sætter dataene ind, og et sted at skrive dem. Intet af dette kan lade sig gøre uden adgangslaget, så lad os starte med at oprette en controller:

```
<?php  
  
class Weblog extends Controller {  
  
    function Weblog()  
    {  
        parent::Controller();  
        $this->load->database();  
        $this->load->model('weblog_model');  
    }  
  
    function index()  
    {  
  
    }  
  
    function create()  
    {  
        $data['message'] = $this->weblog_model->opret();  
        $this->load->view('weblog_create', $data);  
    }  
}
```

```

    }
}

/* End of file weblog.php */
/* Location: ./system/application/controllers/weblog.php */

```

Filen er umiddelbart ganske simpel, men for en god ordens skyld løber vi den igennem. Den starter med en konstruktør, der loader database-delen af CodeIgniter, og weblog-modellen – den har vi endnu ikke lavet, den kommer bagefter. Derefter defineres to metoder, `index()` og `create()`. `index()` skal vi ikke bruge til noget endnu, så den efterlader vi bare tom. I `create()` har vi to linier kode. Den første putter resultatet af `$this->weblog_model->create()` ind i en data-variabel, og den anden åbner weblog-oprettelsesformularen og pusher resultatet ind. Dette betyder at når både vores model og view er lavet, vil formularen fra viewet blive postet til controlleren, der sender dataene videre til modellen, der igen sender resultatet tilbage til controlleren der til sidst sparker det ind i viewet. På den måde kan brugeren få sine data lagt ind, og derefter få at vide hvorvidt det lykkedes. Lige nu vil koden dog ikke fungere, da både modellen og formularen mangler.

Lad os starte med at oprette modellen. Den skal til at starte med bare kunne oprette, så vi laver den ganske simpel for nu:

```

<?php

class Weblog_model extends Model {

    function Weblog_model()
    {
        parent::Model();
    }

    function create()
    {
        if(!$this->input->post('title') &&
            !$this->input->post('body'))
        {
            return '';
        }

        if(empty($this->input->post('title')))
        {
            return 'Titel er ikke angivet!';
        }

        if(empty($this->input->post('body')))
        {
            return 'Der er ingen tekst!';
        }

        $data = array(
            'title' => $this->input->post('title'),
            'body' => $this->input->post('body')
        );
        $this->db->insert('codeigniter_blog', $data);

        return 'Dit indlæg fik ID: ' . $this->db->insert_id();
    }
}

/* End of file weblog_model.php */

```

```
/* Location: ./system/application/models/weblog_model.php */
```

For atter at gennemgå vores kode kan vi starte med at konkludere, at konstruktøren ikke gør andet end at ligge der, til hvis vi får brug for den. Det interessante her er `create()` metoden, der gør nogle ting med vores data. Som du kan se ligger post-data i en metode i CodeIgniter's input-klasse, der hedder `post()`. Denne metode sørger for at filtrere XSS-forsøg og lignende fra inputtet, uden at du behøves tænke over det. Metoden returnerer `FALSE` hvis den ikke er sat.

Det allerførste der sker er at tjekke hvorvidt titel og brødtekst overhovedet er forsøgt postet. Hvis det ikke er tilfældet for nogen af dem, konkluderes det at man ikke forsøger at poste endnu, og der returneres derfor en tom meddelelse. Ellers fortsættes der til at tjekke hvorvidt titelfeltet er tomt eller ej. Hvis det er tomt stopper metoden med meddelelsen om at der skal indtastes en titel. Ellers fortsættes der til at tjekke hvorvidt der er indtastet en brødtekst.

Hvis det hele går glat bliver dataene indsat i databasen via CodeIgniters ActiveRecord klasse, der giver en nydelig databaseabstraktion. Det fungerer ved at vi først angiver et array der indeholder de data vi vil ind – hvor index-nøglerne repræsenterer tabel-feltet – og skubber det ind som argument til `insert()` metoden, lige efter vi har fortalt den hvilken tabel vi vil snakke med. Bemærk at når vi indsætter data på denne måde, bliver input automatisk filtreret og escaped så vi ikke behøves bekymre os om SQL-injections.

Det kan også lade sig gøre at køre almindelige queries med `$this->db->query('INSERT ....')` men de anbefales at bruge ActiveRecord istedet, da dette både er mere sikkert og gør det nemmere at skifte database senere.

Til sidst bliver det indsatte ID sendt tilbage til brugeren i en besked, så vedkommende ved hvilken post han nu skal kigge på for at se det endelige resultat.

Så langt, så godt. Det eneste vi reelt mangler nu for at kunne poste til vores blog er en formular i html. Vi har allerede sagt til vores controller at den ligger under navnet `weblog_create`, så nu laver vi en fil i `system/application/views/` og kalder den `weblog_create.php` med følgende indhold:

```
<html>
<head>
<title>Vores oprettelsesformular</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
</head>
<body>
    <p><?php echo $message; ?></p>
    <form method='post' action=''>
        <p><label for='title'>Titel</label><br/>
        <input type='text' name='title' /></p>

        <p><label for='body'>Brødtekst</label><br/>
        <textarea name='body'></textarea></p>

        <p><input type='submit' value='Opret indlæg' /></p>
    </form>
    <a href='http://dinServerUrl/MinApp/index.php/weblog/overview/'>
        Tilbage til oversigten
    </a>
</body>
</html>
```

Nu er du færdig, og kan tilgå siden med urlen <http://dinServerUrl/MinApp/index.php/weblog/create> og oprette blogs – lav et par stykker med forskellige titler og forskelligt indhold. Du kan ikke se dem endnu, medmindre du går ind og kigger direkte i databasen, men det kommer vi til i næste afsnit.

### Øvelse: Blogsystem – Liste af indlæg

Nu har vi lavet nogle blogs, og nu skal vi have en måde at se dem på. Den bedste måde at gøre det på er at lave en liste som vi kan vælge ud fra. For at opnå dette mål skal vi have lavet et adgangslag til det i controlleren, få vores weblog\_model til at kunne returnere en liste af indlæg, og lave et view til det.

Vi starter med controlleren, og tilføjer følgende metode:

```
function overview()
{
    $data['blogs'] = $this->weblog_model->overview();
    $this->load->view('weblog_overview', $data);
}
```

Nu har vi sagt at der findes en model-metode der hedder overview(), såvel som et view der hedder weblog\_overview. Nu går vi videre til at oprette metoden i weblog\_model, så vi kan få fat i noget data:

```
function overview()
{
    $blogs = $this->db->get('codeigniter_blog');
    return $blogs->result_array();
}
```

Igen benytter vi os af ActiveRecord til at få hevet data ud, istedet for at skulle skrive en lang SELECT FROM .... sætning selv. Resultatet bliver et objekt indeholdende nogle metoder til at hente de selectede data med. Vi bruger result\_array() for at få resultatet tilbage som et array med arrays i. Hvis vi brugte result() ville vi få det som et array med objekter, hvilket kan være en fordel i nogle situationer, men her er vi interesseret i en så simpel løsning som muligt.

Sidst men ikke mindst skal vi have oprettet et view der hedder weblog\_overview.php. Denne fil skal have noget html der kan liste et ubekendt antal blog posts, så vi skal have fat i en lille løkke. Dataene vi får fra modellen er i form af et numerisk array der hver indeholder associative arrays med blogindholdet, så vi skal ganske enkelt lave en PHP foreach og udskrive de forskellige data således:

```
<html>
<head>
<title>Liste over indlæg</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<body>
    <?php foreach($blogs as $blog_entry) { ?>
        <h1>
            <a href='
            http://dinServerUrl/MinApp/index.php/weblog/display/
            <?php echo $blog_entry['id']; ?>
            '>
                <?php echo $blog_entry['title']; ?>
            </a>
```

```

        </h1>
        <p>
            <?php echo nl2br($blog_entry['body']); ?>
        </p>
        <?php } ?>
        <a href='http://dinServerUrl/MinApp/index.php/weblog/create'>
            Opret nyt indlæg
        </a>
    </body>
</html>

```

Når du har gemt filen kan du gå ind på <http://dinServerUrl/MinApp/index.php/weblog/overview> og se alle de indlæg vi oprettede med oprettelsesformularen. Som du kan se har vi lagt et link ind til display-logikken med et id til blogindlægget. Linket duer ikke endnu, men det er det næste vi skal lave.

### Øvelse: Blogsystem - Læs et indlæg

Nu kan vi både oprette og se vores data, men det kan da være interessant at vælge ét enkelt blogindlæg ud og se det på skærmen. Derfor skal vi nu ind og tilføje lidt ekstra features til vores weblog. Først og fremmest skal controlleren have et adgangslag til at fremvise et indlæg, og den controller skal så hente dataene fra modellen, og sende dem videre til et view, ganske som vi er vant til.

```

function display()
{
    $data = $this->weblog_model->display($this->uri->segment(3));
    $this->load->view('weblog_display', $data);
}

```

Det nye her er, at vi skal fortælle vores model hvilket indlæg det specifikt er vi vil se. Det gør vi som ovenfor ved at hente id'et ud af det tredje url segment (det første er weblog og det andet er display) og sende det som argument til metodekaldet. Nu skal metoden laves, så tilføj til weblog\_model.php følgende kode:

```

function display($id)
{
    $data = array('id'=>$id);
    $blog = $this->db->get_where('codeigniter_blog', $data);
    return $blog->row_array();
}

```

Der er to nye ting i dette. Den første er at vi nu bruger `get_where()` istedet for `get()` til at hente data ud med. Dette er åbenlyst for at kunne give et ekstra argument der indeholder et array med identificerende data, så den ved hvad vi leder efter. Det andet er at vi bruger `row_array()` istedet for `result_array()` til at hente de endelige data. `row_array()` henter en enkelt række som et associativt array, snarere end et numerisk array indeholdende associative arrays, som før. Igen bruger vi `_array` varianten af metoden, for at få så simpelt et output som muligt.

Nu er databaseabstraktionen og adgangslaget ordnet, så mangler vi bare at lave viewet. Opret en fil i view-mappen der hedder `weblog_display.php` og fyld den op med HTML og PHP:

```

<html>
<head>
<title>Læs et indlæg</title>

```

```

<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<body>
    <h1><?php echo $title; ?></h1>
    <p><?php echo nl2br($body); ?></p>
    <p><a href='
        http://dinServerUrl/MinApp/index.php/weblog/edit/<?php echo $id;?>
        '>Rediger</a> | <a
        href='http://dinServerUrl/MinApp/index.php/weblog/delete/<?php echo $id;?
        >'>Slet</a></p>
    <a href='http://dinServerUrl/MinApp/index.php/weblog/overview/'>
        Tilbage til oversigten
    </a>
</body>
</html>

```

Nu kan du klikke dig ind på de forskellige posts via listen, og se dem enkeltvis. Vi har ydermere lagt nogle links ind til redigering og sletning af indlægget, og at implementere de funktioner bliver vores næste, og sidste, opgave i denne omgang.

### Øvelse: Blogsystem – Slet/rediger indlæg

Vi nærmer os noget der kan bruges til noget, nu mangler vi sådan set bare at kunne slette vores posts igen, eller rette i dem hvis vi har skrevet forkert. Vi er efterhånden så rutinerede udi at tilføje funktionalitet, at vejen frem burde være klar: kontrolløren skal have et adgangslag til sletning og redigering, modellen skal kunne opdatere databasen med de nye informationer, og vi skal have et interface til at redigere indlægget i. I kontrolløren skal vi derfor bruge to nye metoder:

```

function delete()
{
    $this->weblog_model->delete($this->uri->segment(3));
    $this->overview();
}

function edit()
{
    $message = $this->weblog_model->edit($this->uri->segment(3));
    $data = $this->weblog_model->display($this->uri->segment(3));
    $data['message'] = $message;
    $this->load->view('weblog_edit', $data);
}

```

Den ene tager sig af sletning, og viser derefter vores oversigt ved at kalde overview() - så vi undgår at skulle skrive den kode to gange - den anden af redigering. Først kører vi modellens logik og sender resultatet til en message-variabel. Dernæst henter vi indlægget ud, så vi kan fylde det ind i formularen, og til sidst sætter vi de to data-bidder sammen, og sender dem til viewet. Nu skal vi også have tilføjet funktionaliteten til modellen:

```

function delete($id)
{
    $data = array('id'=>$id);
    $this->db->delete('codeigniter_blog', $data);
}

function edit($id)
{
    if(!$this->input->post('title') && !$this->input->post('body'))
    {
        return '';
    }
}

```

```

    }

    if(empty($this->input->post('title')))
    {
        return 'Titel er ikke angivet!';
    }

    if(empty($this->input->post('body')))
    {
        return 'Der er ingen tekst!';
    }

    $data = array(
        'title' => $this->input->post('title'),
        'body' => $this->input->post('body')
    );
    $where = array(
        'id' => $id
    );
    $this->db->update('codeigniter_blog', $data, $where);

    return 'indlægget blev redigeret';
}

```

Vi har genbrugt meget af logikken fra `create()` med nogle `edit()` specifikke twists. Så mangler vi sådan set kun redigeringsformularen, som skal gemmes som `weblog_edit.php` i `view`-mappen:

```

<html>
<head>
<title>Rediger indlæg</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
</head>
<body>
    <p><?php echo $message; ?></p>
    <form method='post' action=''>
        <p><label for='title'>Titel:</label><br/>
        <input type='text' name='title' value='<?php echo $title; ?>'
        /></p>

        <p><label for='body'>Brødtekst:</label><br/>
        <textarea name='body'><?php echo $body; ?></textarea></p>

        <p><input type='submit' value='Rediger' /></p>
    </form>
    <a href='http://dinServerUrl/MinApp/index.php/weblog/overview/'>
        Tilbage til oversigten
    </a>
</body>
</html>

```

Et voila! Nu har du et fungerende blogsystem skrevet lynhurtigt i CodeIgniter. Det mangler en masse sjove features som for eksempel passwordbeskyttelse, kategorier og et lækkert design, men det vil jeg lade dig selv undersøge i din videre læsning.

## Afrunding

Artiklen her har kun lige ridset lidt i overfladen af hvad CodeIgniter er i stand til, men allerede nu burde det stå klart at CodeIgniter er et stærkt værktøj til at få en masse fra hånden hurtigt. Ved din videre læsning vil du opdage at til stortset enhver situation, har CodeIgniter en fremgangsmåde og



et library til formålet.

### Videre læsning

Dit næste stop må uden tvivl være CodeIgniter's veludviklede [User Guide](#) – den indeholder både tutorials og dokumentation af de forskellige libraries' metoder, samt information om dele af CodeIgniter vi slet ikke har berørt her i artiklen. Som lovet i starten vil artiklen her lige kort ridse op hvad der ligger af godter i de øvrige mapper af en CodeIgniter installation:

- `system/application/errors` : indeholder templates til fejlmeddelelser
- `system/application/helpers` : indeholder såkaldte helpers. Helpers er funktioner der bruges i view filer, til at gøre typografi mv. Lettere at ordne server side.
- `system/application/hooks` : indeholder såkaldte hooks, der fungerer ved at kunne tilføje ekstra funktionalitet til bestemte punkter af en CI applikations kørselsfaser.
- `system/application/language` : indeholder sprogfiler til dynamisk oversættelse af dine systemer.
- `system/application/libraries` : libraries er klasser der tilføjer ekstra funktionalitet til din CodeIgniter applikation, men som er så generelle at de ikke giver mening at have i en model.

Hvis du har virkelig meget blod på tanden kan du også gå ind på CodeIgniter.com's forum og wiki, og se de mange tredjeparts libraries der er lavet, og se om du kan bruge noget af det. Held og lykke!