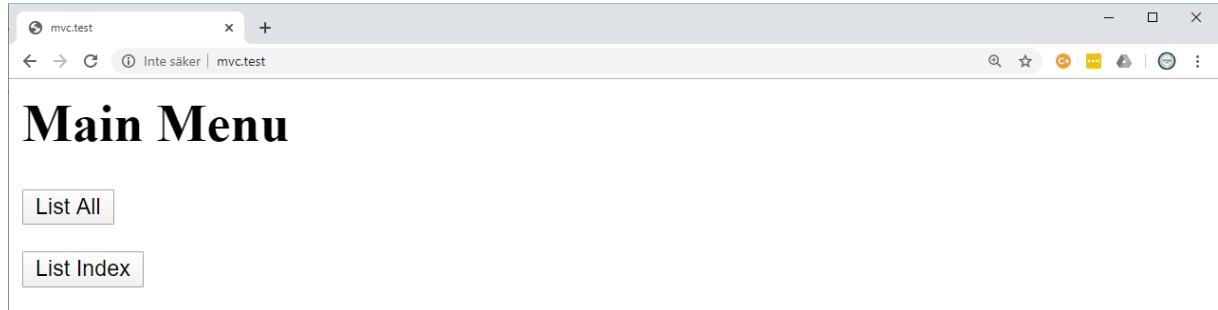
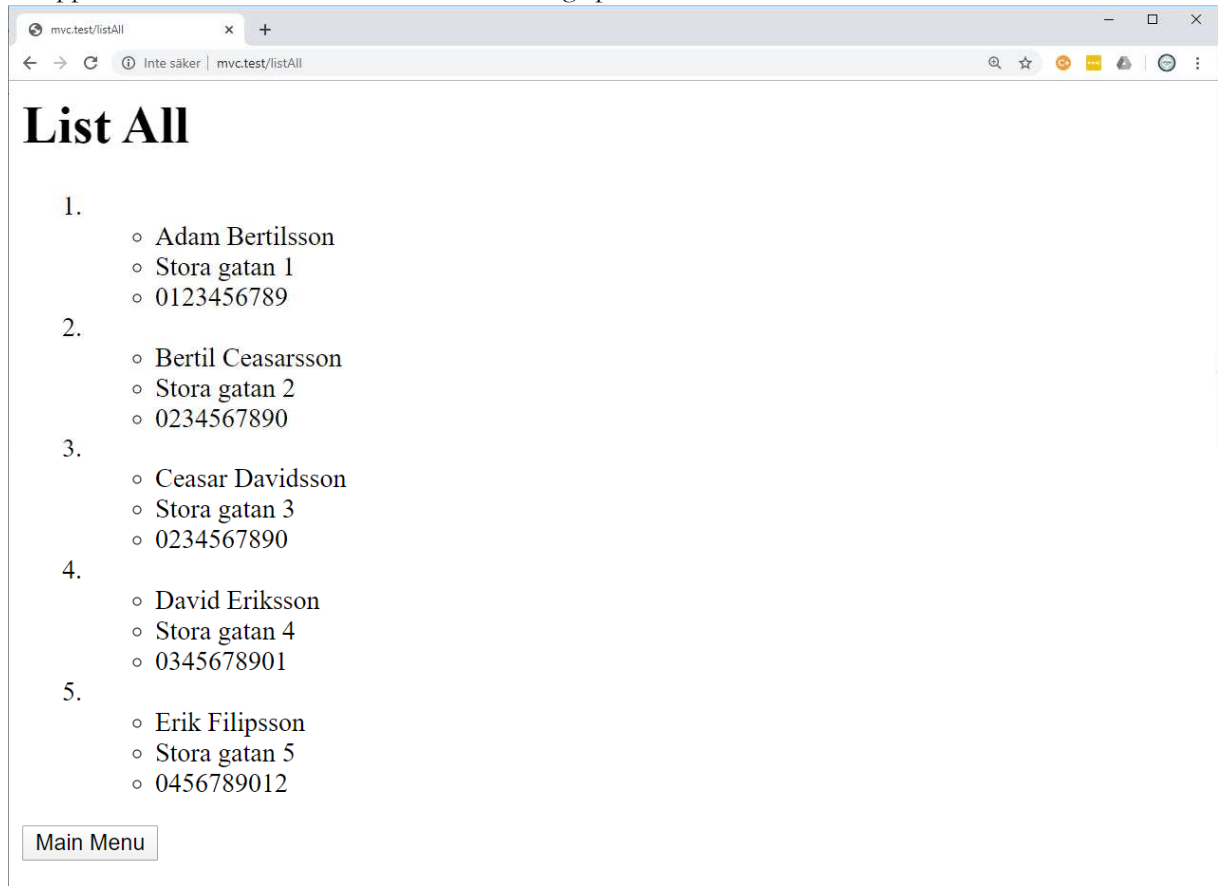


MVC

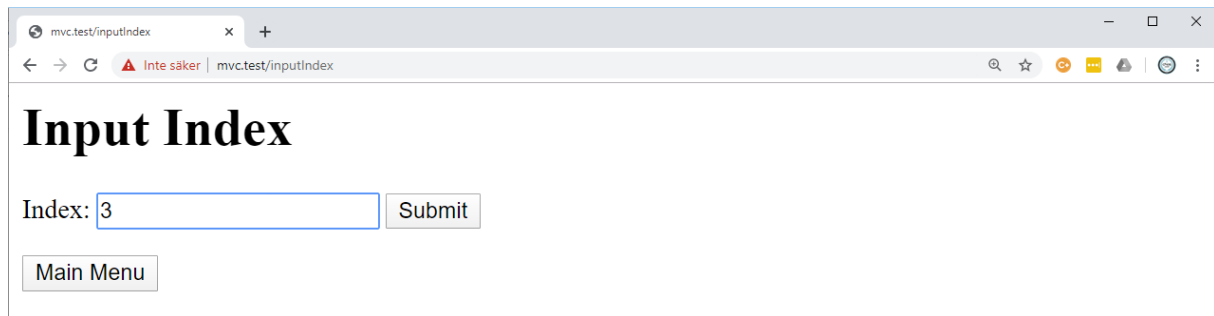
I denna guide går vi igenom hur vi utvecklar en MVC-applikation i PHP, JavaScript och Twig med hjälp av Composer i Homestead. Applikationen hanterar en lista av personer och dess huvudmeny består av två knappar: ”ListAll” och ”ListIndex”.



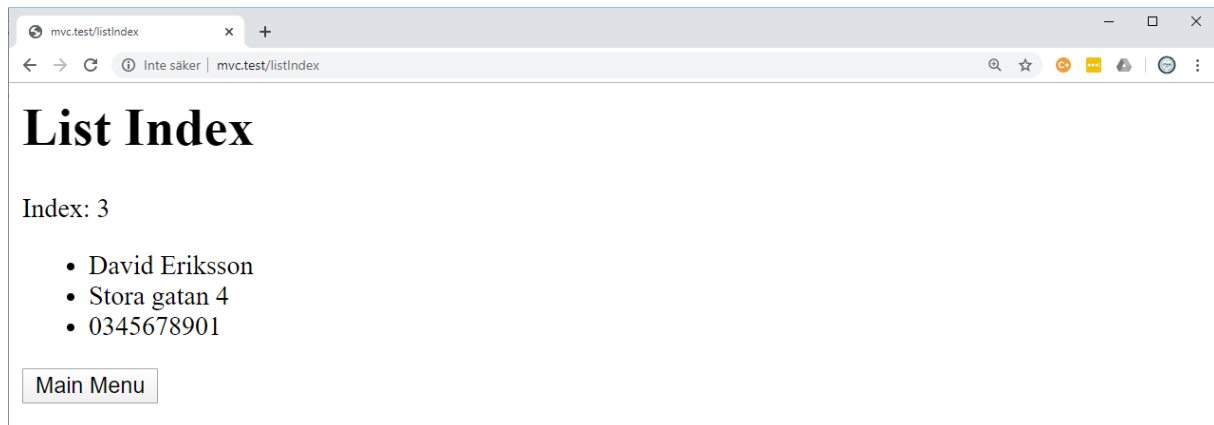
Knappen ”List All” skriver ut en lista med samtliga personer.



Knappen ”List Index” låter användaren mata in ett index ...

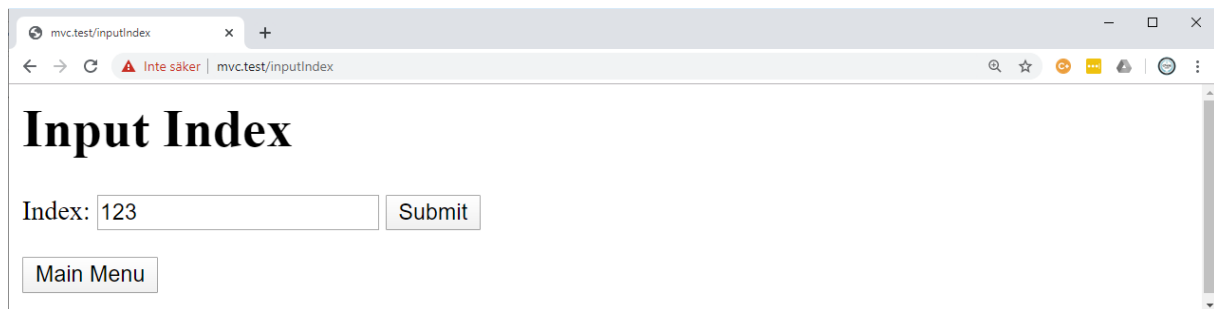


... varefter personen med det angivna indexet skrivs ut ...

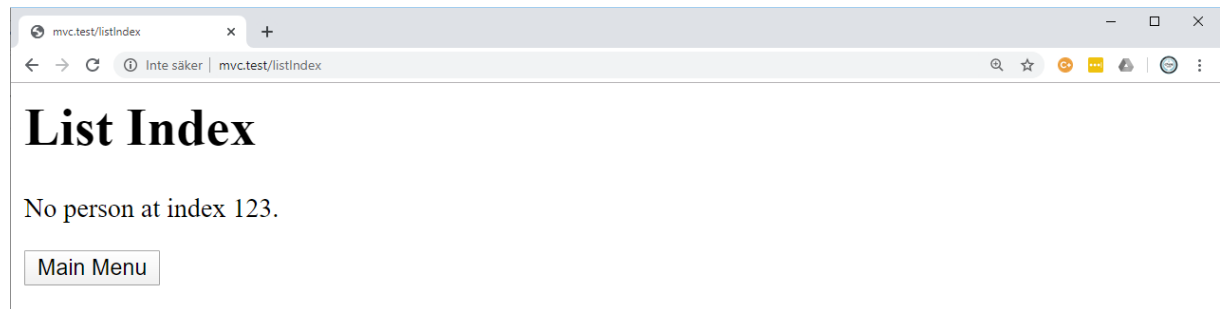


Om användaren matar in ett icke-numeriskt värde skriv ett felmeddelande ut.

... och om användare matar in ett index som ligger utanför listan ...



... meddelas att det inte finns någon person med det indexet.



Domäner

Vi börjar med att uppdatera filen C:\Windows\System32\drivers\etc\hosts med vårt nya projekt, genom att starta notebook som administratör.

På Mac OS använder vi sudo (super user do) och vi (visual instrument):

```
sudo vi \private\etc\hosts
```

eller

```
sudo vi etc\hosts
```

I vi går vi med piltangenterna till den position där vi vill lägga till en ny rad. Då trycker vi tangenten i (insert) och skriver den nya raden. Därefter Escape och :wq (write quit) för att spara och avsluta, :q! för att avsluta utan att spara.

hosts

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
192.168.10.10 homestead.test
192.168.10.10 test.mvc
```

Därefter öppnar vi filen i Homestead.yaml i vår Homestead-katalog och lägger till sökvägen till vår domän.

Homestead.yaml

```
ip: "192.168.10.10"
memory: 2048
cpus: 2
provider: virtualbox
```

folders:

- map: C:\Users\Stefan Bjornander\Documents\homestead\code
to: /home/vagrant/code

sites:

- map: homestead.test
to: /home/vagrant/code/public
- **map: mvc.test**
to: /home/vagrant/code/public/test_mvc

databases:

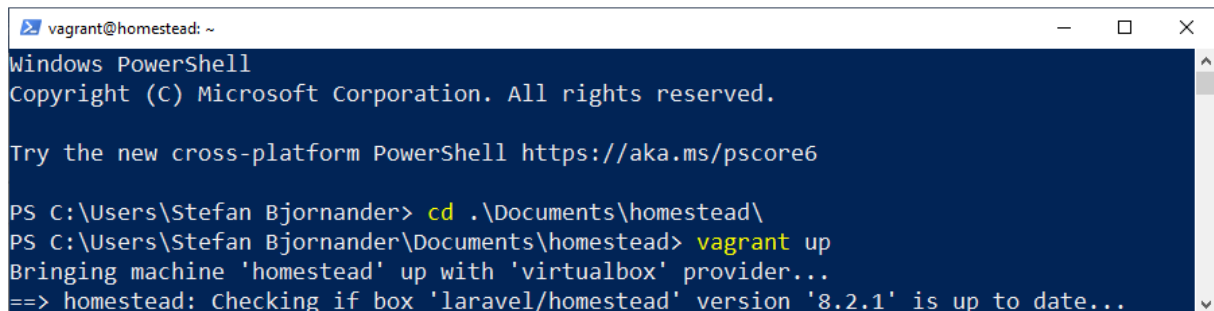
- homestead

features:

- mariadb: false
- ohmyzsh: false
- webdriver: false

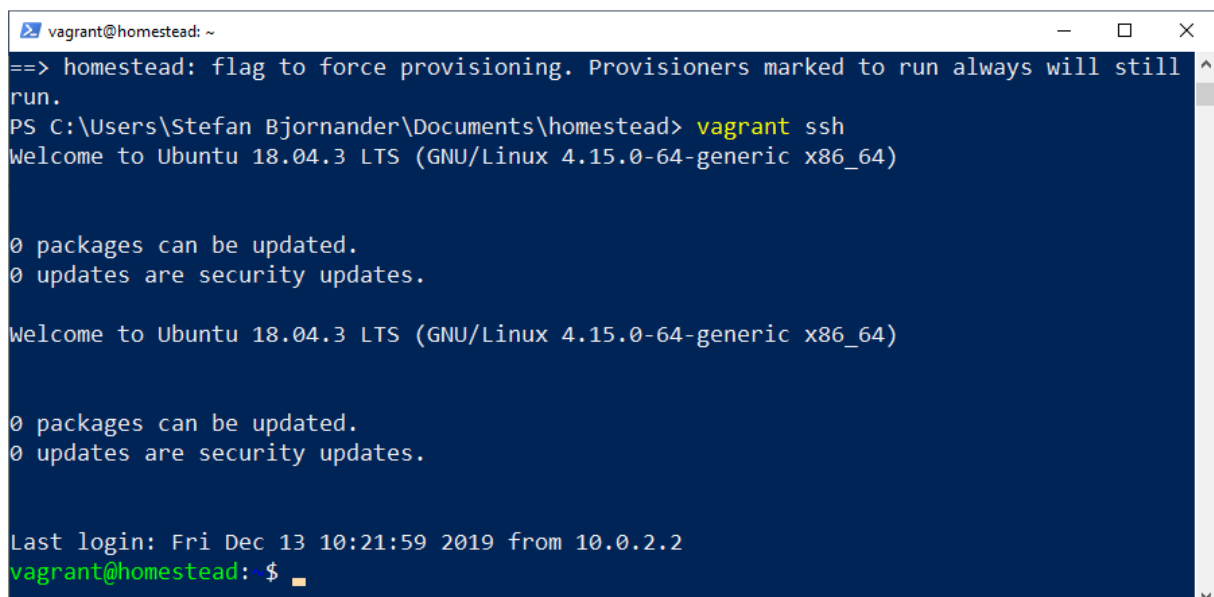
PowerShell

Vi öppnar ett fönster i PowerShell i Windows, i bash i Mac OS, går till Homestead-katalogen och skriver "vagrant up".



```
vagrant@homestead: ~  
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
Try the new cross-platform PowerShell https://aka.ms/pscore6  
  
PS C:\Users\Stefan Bjornander> cd .\Documents\homestead\  
PS C:\Users\Stefan Bjornander\Documents\homestead> vagrant up  
Bringing machine 'homestead' up with 'virtualbox' provider...  
==> homestead: Checking if box 'laravel/homestead' version '8.2.1' is up to date...
```

Därefter "vagrant ssh".



```
vagrant@homestead: ~  
==> homestead: flag to force provisioning. Provisioners marked to run always will still run.  
PS C:\Users\Stefan Bjornander\Documents\homestead> vagrant ssh  
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-64-generic x86_64)  
  
0 packages can be updated.  
0 updates are security updates.  
  
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-64-generic x86_64)  
  
0 packages can be updated.  
0 updates are security updates.  
  
Last login: Fri Dec 13 10:21:59 2019 from 10.0.2.2  
vagrant@homestead:~$
```

Vi går till vår public-katalog, såsom /home/vagrant/code/public. Därefter skapar vi en ny katalog för vårt projekt (mkdir test_mvc) och går ner i den katalogen (cd test_mvc).

```
vagrant@homestead: ~/code/public/test_mvc
vagrant@homestead:~$ vagrant@homestead:~$ cd /home/vagrant/code/public
vagrant@homestead:~/code/public$ mkdir test_mvc
vagrant@homestead:~/code/public$ cd test_mvc
vagrant@homestead:~/code/public/test_mvc$
```

Vi laddar in bibliotekskod för Twig med hjälp av Composer: composer require "twig/twig:^2.0".

```
vagrant@homestead: ~/code/public/test_mvc
vagrant@homestead:~/code/public/test_mvc$ composer require "twig/twig:^2.0"
1/10: http://repo.packagist.org/p/provider-2019-01$1d684176b35beea328f4690daf131dd8058a3aa57ed1c2d7de0e8ad9ee5d1d65.json
2/10: http://repo.packagist.org/p/provider-2014$56b6fbd47495ddb582eb50255016992f4e78d9ee25ac7707882b1bb87bc27989.json
3/10: http://repo.packagist.org/p/provider-latest$b280894125d9fa9ad67477ddbb37eb008d11056e1c3e3256d8c6124d632fea4b.json
4/10: http://repo.packagist.org/p/provider-2015$8530b3832d2b6963360650e3d2d22883320f54f1b789fa378e88cb31dfe76091.json
5/10: http://repo.packagist.org/p/provider-2019-07$a65ed47950cd155981f22ee61d01de9f2dbd524ac8ba4a566fef11c3a9561897.json
6/10: http://repo.packagist.org/p/provider-2019-04$521d3b7d7b3543589f138625102d39db8a47f245f1692c9e73c47468b74c7cc2.json
7/10: http://repo.packagist.org/p/provider-2019-10$75a794bbcf3879acb8caf3c6a45a39b8155fbfd3db9853bc32c9b0809cd0cb1e.json
8/10: http://repo.packagist.org/p/provider-2017$080965516aecb29ea431b88840c6b24cf1e12481913e09ed4c423581bc2d86bf.json
9/10: http://repo.packagist.org/p/provider-2016$d4fafed83db90d00ca95f227a5e9345b679a0d7942583bb1159eb65b2451499e.json
10/10: http://repo.packagist.org/p/provider-2018$48073909e27df0e3e2adfd2a463e551a71c2d62af6580b5bad6bce3127ece310.json
Finished: success: 10, skipped: 0, failure: 0, total: 10
./composer.json has been updated
1/2: http://repo.packagist.org/p/provider-latest$5a6cf7cd257d2e0ade565265c0d44cda5da7ac445021251f9c7b428991c17866.json
2/2: http://repo.packagist.org/p/provider-2019-10$3c7543ce9cf2251306807af971bd937eea0e4b6b2342bd87bd311c3377b5558b.json
Finished: success: 2, skipped: 0, failure: 0, total: 2
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 1 update, 0 removals
- Installing symfony/polyfill-php (v1.13.1): Loading from cache
- Updating twig/twig (v2.0.0 => v2.12.2): Checking out d761fd1f1c
Writing lock file
Generating autoload files
vagrant@homestead:~/code/public/test_mvc$
```

Detta genererar bland annat filen composer.json.

Composer.json

```
{
    "require": {
        "twig/twig": "^2.0"
    }
}
```

Vi lägger till några rader för att autoloading skall fungera.

```
{
    "require": {
        "twig/twig": "^2.0"
    },
    "autoload": {
        "psr-4": {
            "Main\\": ""
        }
    }
}
```

Request och Router

Nu är det dags att skriva vår första källkodsfil i PHP: Index.php. Vi placerar alla filerna i vår projektkatalog, katalogen kommer i slutändan innehålla dessa filer.

```
vagrant@homestead: ~/code/public/test_mvc
vagrant@homestead:~/code/public/test_mvc$ pwd
/home/vagrant/code/public/test_mvc
vagrant@homestead:~/code/public/test_mvc$ ls
composer.json  InputController.php  ListController.php  MainMenuView.twig  Request.php
composer.lock  InputIndexView.twig  ListIndexView.twig  Model.php           Router.php
Index.php      ListAllView.twig     MainController.php  Approxi            Vendor
```

Index.php

```
<?php
    use Main\Router;
    use Main\Request;

    // Vi behöver lägga till dessa tre rader för att kunna använda oss av Twig.
    require_once __DIR__ . "/vendor/autoload.php";
    $loader = new Twig_Loader_Filesystem(__DIR__);
    $twig = new Twig_Environment($loader);

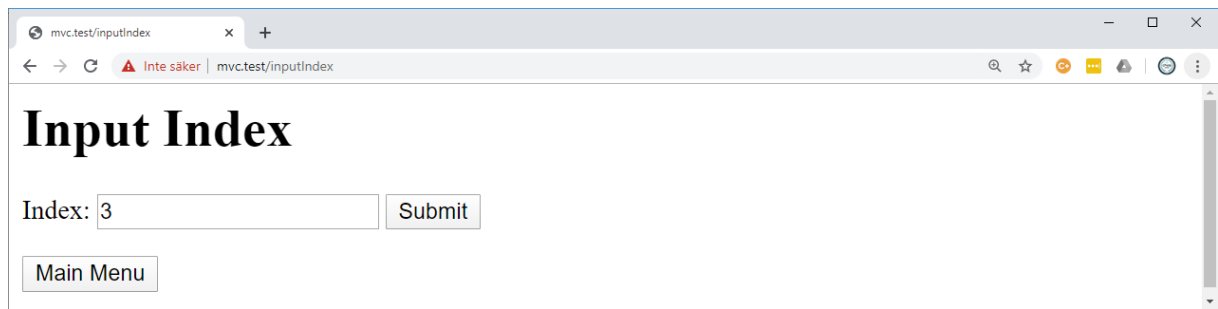
    // Vi skapar objekt av klasserna Request och Router.
    $request = new Request();
    $router = new Router();

    // Vi anropar route i Router-objektet, som returnerar
    // den färdiga HTML-koden, som vi skriver ut med echo.
    echo $router->route($request, $twig);
?>
```

I Index.php ovan skapade vi objekt av klasserna Request och Router. Systemet skickar en begäran som Router tolkar och agerar på. Request samlar ihop den information som Router behöver för att kunna genomföra rätt åtgärd. Routern kan liknas vid en trafikpolis i en vägkorsning, dess uppgift är att tolka meddelanden och utföra rätt åtgärd för varje meddelande.

Vår klass Request är ganska enkel. Dess konstruktor sätter medlemsvariablerna \$path och \$form, där \$path är icke-domän delen av en URI (Uniform Resource Locator). En domän är en vanlig internetadress och path är den extra del som skickas med till routern. Till exempel, om URL:en är "https://www.dn.se/ekonomi/motor/darfor-ar-det-bra-med-is-i-magen/", är domänen "https://www.dn.se" och path "/ekonomi/motor/darfor-ar-det-bra-med-is-i-magen/". Det är path vi kollar på i routern.

Den andra medlemsvariabeln \$form är en map (array i PHP) som håller indata från en HTML-formulär. Om indatafältet i nedanstående formulär har namnet "index" kommer mappen bli ["index" => "3"] när användaren klickar på Submit-knappen.



Request.php

```
<?php
    namespace Main;

    class Request {
        private $path, $form;

        public function __construct() {
            $this->path = $_SERVER["REQUEST_URI"]; // path-delen av URL:en
            $this->form = $_POST;                  // indata till formulären
        }

        public function getPath() {
            return $this->path;
        }

        public function getForm() {
            return $this->form;
        }
    }
?>
```

Router.php

```
<?php
    namespace Main;

    use Main\ListController;
    use Main\InputController;
    use Main>MainController;

    // Routers metoden route tar ett objekt vardera av Request och Twig.
    class Router {
        public function route($request, $twig) {
            $path = $request->getPath();
            $form = $request->getForm();
        }
    }
}
```

```

// Vi jämför path med de olika alternativen, skapar upp ett objekt av
// rätt Controller-klass, och anropar rätt metod. Anropet returnerar
// den färdiga HTML-koden, som vi returnerar tillbaka till filen
// Index.php, och som därefter skrivs ut i Webbläsaren.
if ($path == "/listAll") {
    $controller = new ListController();
    return $controller->listAll($twig);
}
else if ($path == "/inputIndex") {
    $controller = new InputController();
    return $controller->inputIndex($twig);
}

// När vi skall skriva ut en specifik person behöver vi dess index,
// som matades in av användaren via ett formulär. Det hämtar vi från
// medlemsvariabeln $form.
else if ($path == "/listIndex") {
    $controller = new ListController();
    $index = $form["index"];
    return $controller->listIndex($twig, $index);
}

// När vi först laddar filen Index.php kommer path vara endast "/",
// och då skall huvudmenyn presenteras.
else if ($path == "/") {
    $controller = new MainController();
    return $controller->mainMenu($twig);
}

// Om inget alternativ passar, returnerar vi endast ett felmeddelande.
else {
    return "Router Error!";
}
}
}
?>

```


Modellen

Innan vi tittar på kontrollers och vyer börjar vi med modellen. I denna applikation har vi endast en modell i klassen Model, som har två metoder: getAll, som ger hela personlistan, och getIndex, som ger personen med ett givet index.

Model.php

```
<?php
    namespace Main;

    class Model {
        private $personArray;

        // Konstruktorn initierar medlemsvariabeln $personArray med fem personer.
        public function __construct() {
            $adam = ["name" => "Adam Bertilsson", "address" => "Stora gatan 1",
                    "phone" => "0123456789"];
            $bertil = ["name" => "Bertil Ceasarsson", "address" => "Stora gatan 2",
                    "phone" => "0234567890"];
            $ceasar = ["name" => "Ceasar Davidsson", "address" => "Stora gatan 3",
                    "phone" => "0234567890"];
            $david = ["name" => "David Eriksson", "address" => "Stora gatan 4",
                    "phone" => "0345678901"];
            $erik = ["name" => "Erik Filipsson", "address" => "Stora gatan 5",
                    "phone" => "0456789012"];
            $this->personArray = [$adam, $bertil, $ceasar, $david, $erik];
        }

        // Metoden getAll returnerar hela medlemsvariabeln $personArray.
        public function getAll() {
            return $this->personArray;
        }

        // Metoden getIndex returnerar personen med det givna indexet.
        public function getIndex($index) {
            if ($index < count($this->personArray)) {
                return $this->personArray[$index];
            }
            // Om inte indexet ryms i arrayen, returneras null.
            else {
                return null;
            }
        }
    }
?>
```

Kontrollers

I denna applikation använder vi tre kontroller: ListController, som skriver ut listor, InputController, som sköter inmatning i ett formulär, och MainController, som tar hand om huvudmenyn.

Klassen ListController har två metoder: listAll, som anropas när alla personer i listan skall skrivas ut, och listIndex, som anropas när en person med ett givet index skall skrivas ut.

ListController.php

```
<?php
    namespace Main;

    // Metoden listAll skapar ett objekt av klassen Model och anropar metoden
    // getAll, som ger hela personarrayen.
    class ListController {
        public function listAll($twig) {
            $model = new Model();
            $personArray = $model->getAll();

            // Personarrayen läggs i en map som skickas med twig:en. På så sätt
            // kan vi i twig-koden skriva ut personarrayen.
            $map = ["personArray" => $personArray];
            return $twig->loadTemplate("ListAllView.twig")->render($map);
        }

        // Metoden listIndex tar förutom twig:en det index för personen vi söker.
        // Vi skapar ett objekt av klassen Model och anropar getIndex, vilket ger
        // oss den sökta personen, eller null om indexet var utanför arrayens
        // räckvidd.
        public function listIndex($twig, $index) {
            $model = new Model();
            $person = $model->getIndex($index);

            // Vi placerar indexet och personen i en map som vi skickar med i
            // Twig-anropet. Twig-anropet returnerar den färdiga HTML-koden, som
            // vi returnerar tillbaka till routern.
            $map = ["index" => $index, "person" => $person];
            return $twig->loadTemplate("ListIndexView.twig")->render($map);
        }
    }
?>
```

Metoderna `inputIndex` och `mainMenu` i klasserna `InputController` och `MainController` anropar endast sina motsvarande Twig-filer, med tomma mappar.

InputController.php

```
<?php
    namespace Main;

    class InputController {
        public function inputIndex($twig) {
            return $twig->loadTemplate("InputIndexView.twig")->render([]);
        }
    }
?>
```

MainController.php

```
<?php
    namespace Main;

    class MainController {
        public function mainMenu($twig) {
            return $twig->loadTemplate("MainMenuView.twig")->render([]);
        }
    }
?>
```

Vyer

Vi har fyra Twig-filer som beskriver applikationens vyer. Den första användaren stöter på är `MainView`, vars kod består av enbart HTML, och som skriver ut de två knapparna "List All" och "List Index". Varje knapp är omgiven av ett formulär. Notera att i bägge fallen använder vi `post`-metoden. Det finns även den mer komplicerade `get`-metoden. Som nämnts ovan läser routern av `path`-delen av URL:en. Vi anger lämpligt `path` som `action`. Routern kommer läsa `path`-värdena `/listAll` respektive `/listIndex` och vidta lämpliga åtgärder.

MainMenuView.twig

```
<h1>Main Menu</h1>
<p>
    <form method="post" action="/listAll">
        <input type="submit" value="List All">
    </form>
<p>
    <form method="post" action="/inputIndex">
        <input type="submit" value="List Index">
    </form>
```

ListAllView är ganska enkel. Den loopar igenom personarrayen med hjälp av en Twig-for-sats, och skriver ut varje persons uppgifter i en lista.

ListAllView.twig

```
<h1>List All</h1>
<p>
<ol>
    {% for person in personArray %}
        <li>
            <ul>
                <li>{{person.name}}</li>
                <li>{{person.address}}</li>
                <li>{{person.phone}}</li>
            </ul>
        </li>
    {% endfor %}
</ol>
<p>
<form method="post" action="/">
    <input type="submit" value="Main Menu">
</form>
```

InputIndexView sätter upp ett formulär där användaren matar in det önskade indexet. Indexet lagras då i Request-klassens medlemsvariabel \$form. Vi kan också med hjälp av JavaScript undersöka om indexet är ett icke-negativt heltal.

För att kunna kontrollera ett värde i ett formulär adderar vi onsubmit i form-taggen. Där anropar vi JavaScript-funktionen validateIndex, som returnerar sant eller falskt. Om den returnerar sant kommer formuläret submittas på normalt sätt, om den returnerar falskt händer ingenting. Vi behöver också sätta id="index" på indatafältet vi skall kontrollera.

InputIndexView.twig

```
<h1>Input Index</h1>
<p>
<form method="post" action="/listIndex" onsubmit="return validateIndex();">
    Index: <input type="text" name="index" id="index">
    <input type="submit" value="Submit">
</form>
<p>
<form method="post" action="/">
    <input type="submit" value="Main Menu">
</form>

<script>
    // Funktionen allDigits returnerar sant om texten innehåller minst en
    // siffra, och enbart siffror. Om texten alltså utgör ett icke-negativt
    // heltal.
```

```

// Vi använder mönstermatchning för att avgöra om texten består av
// minst en siffra, och enbart siffror.
// ^ början av texten, $ slutet av texten
// [0-9] något av siffrorna 0-9
// + ett eller flera, * noll eller flera, ? exakt ett
function allDigits(text) {
  return text.match(/^[0-9]+$/);
}

// Funktionen validateIndex anropas då användaren klickar på Submit-
// knappen. Om användaren angav ett korrekt värde i indatafältet
// returneras sant.
function validateIndex() {
  let index = document.getElementById("index").value;

  if (allDigits(index)) {
    return true;
  }

  // Om användaren angav ett inkorrekt värde i indatafältet skrivs ett
  // felmeddelande ut med alert, och falskt returneras.
  else {
    alert "\"" + index + "\" is not a valid index.");
    return false;
  }
}
</script>

ListView skriver ut personen om den inte är null med hjälp av en Twig-
if-sats. Är den null skrivs ett meddelande ut om att indexet saknas.
ListIndexView.twig
<h1>List Index</h1>
<p>
  <!-- ListView undersöker först om personen är null.
  Om inte skriver den ut indexet och personen med det indexet. -->
  {% if person is not null %}
    Index: {{index}}
  <p>
    <ul>
      <li>{{person.name}}</li>
      <li>{{person.address}}</li>
      <li>{{person.phone}}</li>
    </ul>

```

```
<!-- Om personen är null skriver vi ut att det
      inte finns en person med det avgivna indexet. -->
{% else %}
    No person at index {{index}}.
{% endif %}
<p>
<form method="post" action="/">
    <input type="submit" value="Main Menu">
</form>
```

Stefan Björnander