За всички задачата е една:

Създайте php приложение, организирано по модела MVC. Приложението да използва SEF URI, въвеждани на адресния ред, свързани с изпълнение на съответните операции с данните от базата. Използвайте URI, при четенето на които да е ясна изпълняваната операция.

Накратко за алгоритъма на решение на задачата.

Основните файлове в приложението според шаблона за проектиране MVC са моделите, изгледите и контролерите. За проектните примери следва да напишете един модел, един контролер и няколко изгледа.

Модел – клас, съдържащ методи за работа с обектите от базата данни (методите може да са статични, както в разгледания на лекцията пример, но това не е задължително, зависи от конкретната задача).

Контролер – клас, който осъществява връзката между модела и изгледите. Съдържа методи-действия (имената им препоръчително започват с action), които извикват методите на модела и предават резултатите на съответните изгледи.

Изгледи – php файлове за визуализация на резултатите. При желание може да намерите готови красиви стилове и да ги адаптирате, в предлагания пример не съм се занимавала с това.

Допълнителни пояснения към задачата

1. Създаване на конфигурационния файла .htaccess в коренната папка на приложението

   Както бе описано по-горе, създаваното приложение ще използва SEF адреси. За тях може да прочетете примерно тук:

   https://www.siteground.com/kb/what_are_search_engine_friendly_urls_sef/

   С цел опростяване SEF адресите ще се въвеждат от адресния ред. За примера, който Ви изпращам, са предвидени следните адреси (те се задават в routes.php, както ще стане дума по-нататък)

   http://localhost/yourName/users

   http://localhost/yourName/users/12

   http://localhost/yourName/users/add

   Вие следва да сами да си подберете адреси, съответващи на Вашата задача.

   За да не се получава съобщение, че страницата не е намерена, когато въведем SEF URI(Uniform Resource Identifier), в конфигурационния файл .htaccess следва да се запише:

   ```
   RewriteEngine on

   RewriteRule ^(.*)$  index.php
   ```

   По този начин независимо от въведения URI (регулярният израз `^(.*)$` описва всички възможни идентификатори), **се осъществява пренасочване към главния контролер `index.php`**

2. Описание на начина, по който ще се обработват въвежданите SEF URI, т.е. задаване на т.н. маршрути (routes)

   Това се прави във файла `routes.php`, който логично се разполага в папката config

   Следвайки обектно-ориентирания подход на шаблона MVC, за изпълняване на изискванията с даден URI операция, се извиква метод-действие на съответния контролер. **Кой метод на кой контролер се указва във файла `routes.php`**. При това се препоръчват следните правила за имената : името на класа на контролера завършва с думата Controller, а имената на методите на контролера започват с думата action. При указание на маршрутите в `routes.php` тези стандартни думи се пропускат. **`routes.php`** връща асоциативен масив с ключове обработваните URI и значения - начинът на обработка. Значението задължително включва името на контролера без ключовата дума *Controller* и името на метода без ключовата дума *action*. Така, ако извикваният метод не изисква параметри, на всеки URI-ключ съответства значение, включващо 2 елемента - ***<контролер>/<метод>,*** например routes.php може да има вида (обърнете внимание, че **URI се изброяват от по-специфичните към общите**):

```
return array(
 " persons /add"=> " persons /add",  // метод actionAdd на класа PersonsController
 " persons /delete"=> " persons /delete ",  // метод actionDelete  на класа PersonsController
 " persons "=> " persons /list ",  // метод actionList  на класа PersonsController
 );
```

   Когато е необходимо да подадем параметри на съответния action метод, в общия случай това съответствие не е възможно да се запише директно. Например, нека искаме да изведем списък на лицата, родени в определен град в определена година – в този случай на метода за извеждането с име примерно actionView трябва да се подадат 2 параметъра –града и годината. Директно това би изглеждало примерно по следния начин:

```
return array(
  ….
 " persons /1980/Бургас"=> " persons /view/Бургас/1980",
 " persons /1999/Варна"=> " persons /view/Варна/1999",
  ….
);
```

   Ясно е, че по този начин не могат да се запишат всички варианти, броят им в общия случай може да бъде много голям. За записване на съответствието по обобщен начин ключът се задава чрез регулярен израз, а в маршрута задаваме референции към съответствията във вида **$n**, където **n** е номера на **поредната заградена в малки скоби част от шаблона.**

```
return array(
  ….
   " persons/([1-2][0-9]{3})/([\w|\W]{3,})"=> " persons/view/$2/$1",
  ….
);
```

Замяната на референциите в маршрута с конкретните съвпадения става с помощта на функцията preg_replace:

http://php.net/manual/bg/function.preg-replace.php

Пример за използване на preg_replace()

```
<?php
$pattern="~persons/([1-2][0-9]{3})/([\w|\W]{3,})~";
$uri="persons/1998/Варна";
$general_route="persons/view/$2/$1";
$route= preg_replace($pattern, $general_route, $uri);
 echo $route;       // persons/view/Варна/1998
 ?>
```

В случая за ограничителя на шаблона за търсене е използван символа **~** (php позволява), тъй като символа / се съдържа в шаблона.
Следва да се има предвид, че ако съвпадение не е намерено, preg_replace () връща оригиналния низ, в случая persons/1998/Варна

3. В папката config записваме също така файла db_params.php, връщащ асоциативен масив с параметрите, необходими за осъществяване на връзката с базата данни.

4. В папката components създаваме файла router.php, съдържащ класа **`Router`**. Методът `run()` на класа Router търси съвпадение на въведения URI със зададените URI във вече разгледания файл routes.php и ако такова съвпадение е намерено, се извличат елементите на маршрута (клас на контролера, метод на действие и параметри, ако има такива), създава се обект на контролера и се извиква необходимият му метод за действие.
   Детайлите на работа на този клас може да си изясните чрез тестване на работата му и справки за работата на включените библиотечни функции. В крайния вариант включените спомагателните извеждания следва да се премахнат.

5. В папката components създаваме файла database.php, съдържащ класа Database с единствен статичен метод, връщащ референция към обект PDO за работа с базата данни. Повече за PDO можете да прочетете примерно тук:

   http://php.net/manual/en/book.pdo.php

6. Както вече бе описано в точка 1, изпълнението на приложението започва винаги от файла index.php – главният контролер. В него изпълняваме някои основни настройки (показването на грешките е само за етапа на разработка), включваме необходимите файлове от папката components, след което създаваме обект Router и извикваме метода му run(). Последният от своя страна при намерено съвпадение на URI извиква необходимият action метод на необходимия контролер. Контролерът използва модела за получаване на резултатите и ги предава на изгледа, който ги визуализира. По този начин шаблона MVC позволява отговорностите за изпълнение на отделните части от задачата (обработката на данните и визуалното им представяне), да се разделят.

Предложеният вариант е примерна опростена реализация на MVC- актуалният в момента шаблон за проектиране на web приложения. За интересуващите се в нета има огромно количество информация.

За да тествате примера, който Ви изпращам, копирайте папката `yourName` в `xampp/htdocs`. Създайте таблица `users` и импортирайте таблицата `data` от файла. Обработваните адреси са дадени по-горе, копирам ги отново:

http://localhost/yourName/users

http://localhost/yourName/users/1

http://localhost/yourName/users/add

Следват индивидуалните варианти за база данни.

Моля, запишете проектите си в папка с **Вашето име и ФН** на htdocs. Съответно, на мястото на `yourName` в `routes.php` на примера, който Ви изпращам, да е името на Вашата папка.

Започнете със скелета на приложението – това, което правихме на лекцията - `testRoutes.php`, но с използване на отделни файлове - използвайте примера, който Ви изпращам, и го адаптирайте за Вашата задача. Ако Ви е трудно, първо го повторете в същия вид, но с зададените от Вас адреси и съответните им маршрути. `testRoutes2par.php` е същият пример, но с извличане на 2 параметъра от адреса. След като се убедите, че е извикан нужният метод на нужния клас на контролера (класът –контролер при Вас е един), се заемете с детайлите – база данни и т.н.

Ето конкретните варианти на задачата. Зададена е **минимално изискваната функционалност**, всякаква допълнителна, за която се сетите, се приветства! Свободни сте също и да разширявате базата данни, за да е по-реално приложението, аз съм гледала да има по 4 полета в повечето задачи - да не Ви претоварвам ;)

Мирослав

- База данни prog_languages(програмни езици) с таблица prog_language(програмен език)– id, име, създател (лице или фирма), година на създаване, рейтинг за текущата година (https://www.tiobe.com/tiobe-index/)
  Функционалност :
  - Извежда списък с всички програмни езици
  - Извежда информацията първите 5 по рейтинг програмни езици
  - Извежда списък програмните езици, създадени след определена година(параметър)
  - Добавя програмен език