Модел-Изглед-Контролер шаблон (MVC -model view controller pattern) в PHP

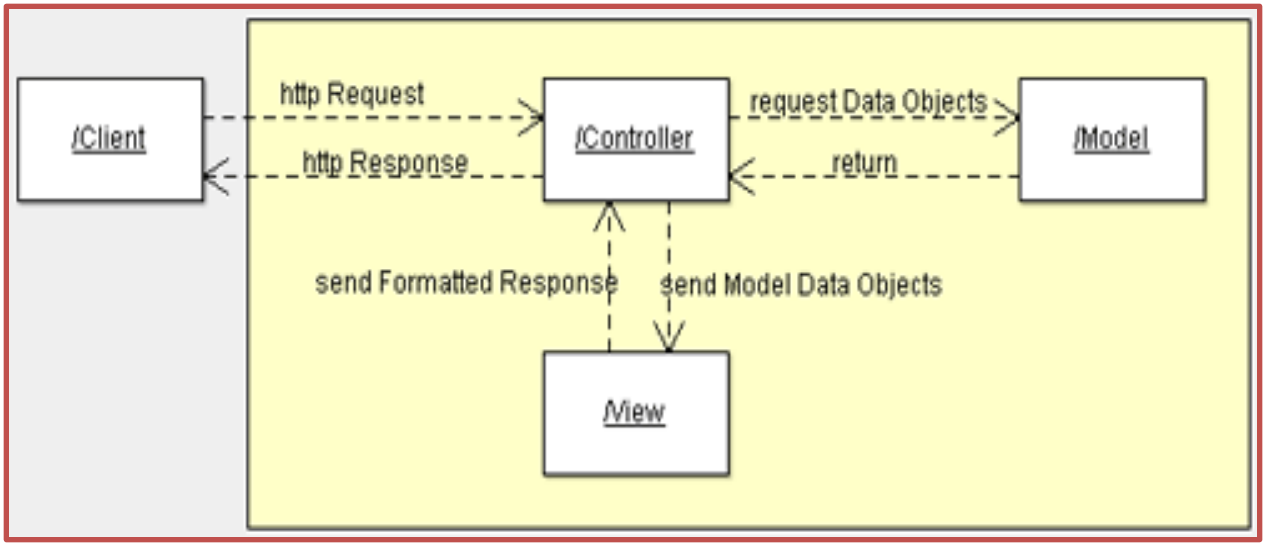
Виолета Божикова

За MVC

- Шаблонът Модел-Изглед-Контролер (MVC - model view controller pattern) е най-използваният шаблон понастоящем за web приложения.
- За първи път е използван в езика Smalltalk, след което е адаптиран за Java. В момента има повече от дузина PHP рамки (frameworks), основани на MVC модел.

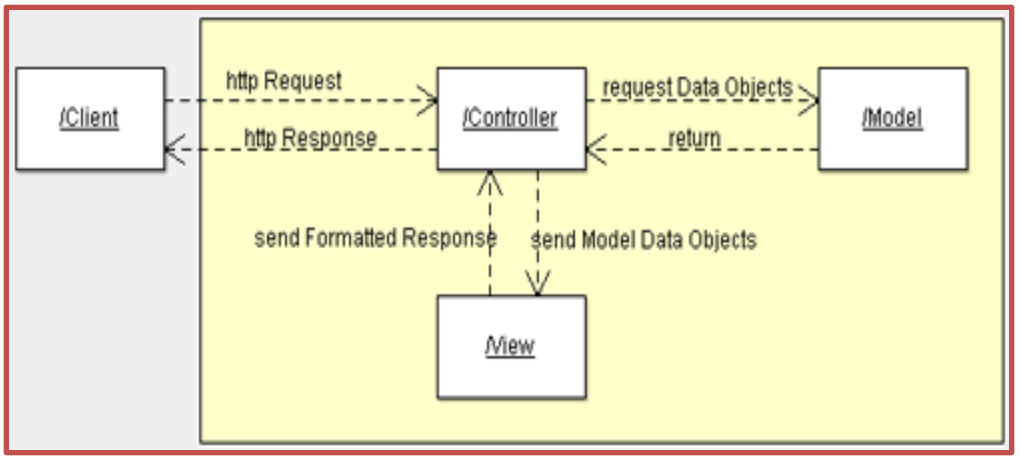
За MVC

- MVC шаблон разделя приложението в 3 модула:
  - Модел (Model),
  - Изглед (View)
  - Контролер (Controller):



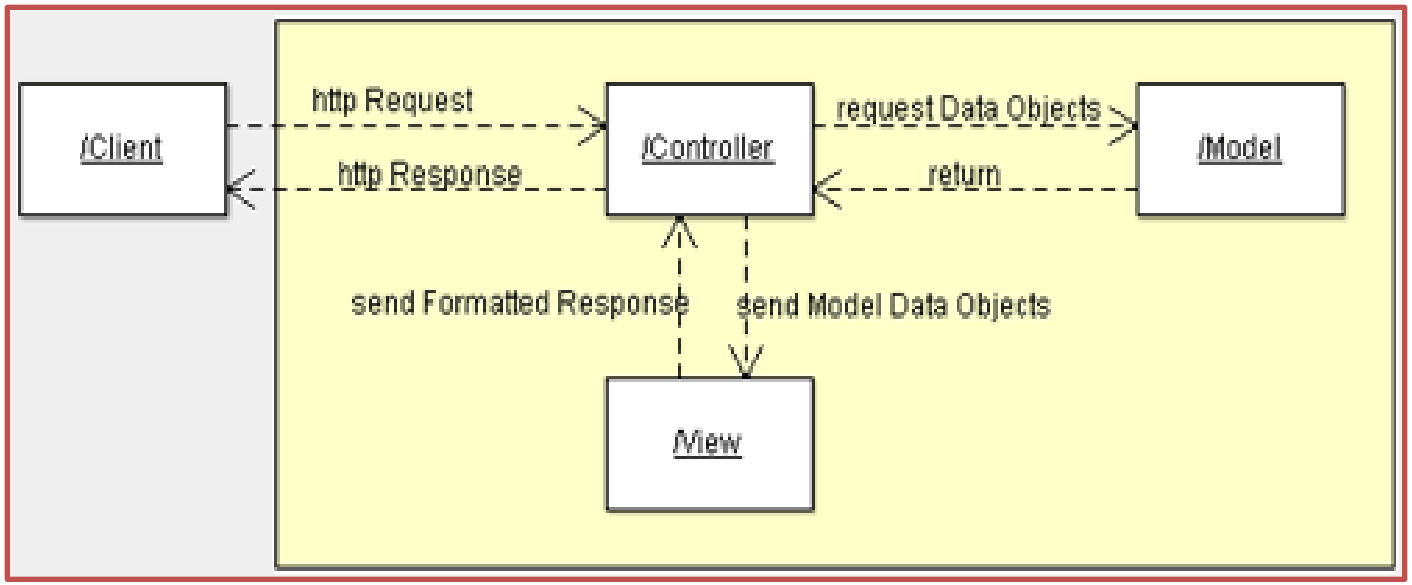
За MVC - **Model**

- Модул **Model** е отговорен за управлението на данните;
- Той **съхранява и извлича данните**, използвани от приложението (обикновено в БД), **както и логиката, изпълнявана от приложението**.



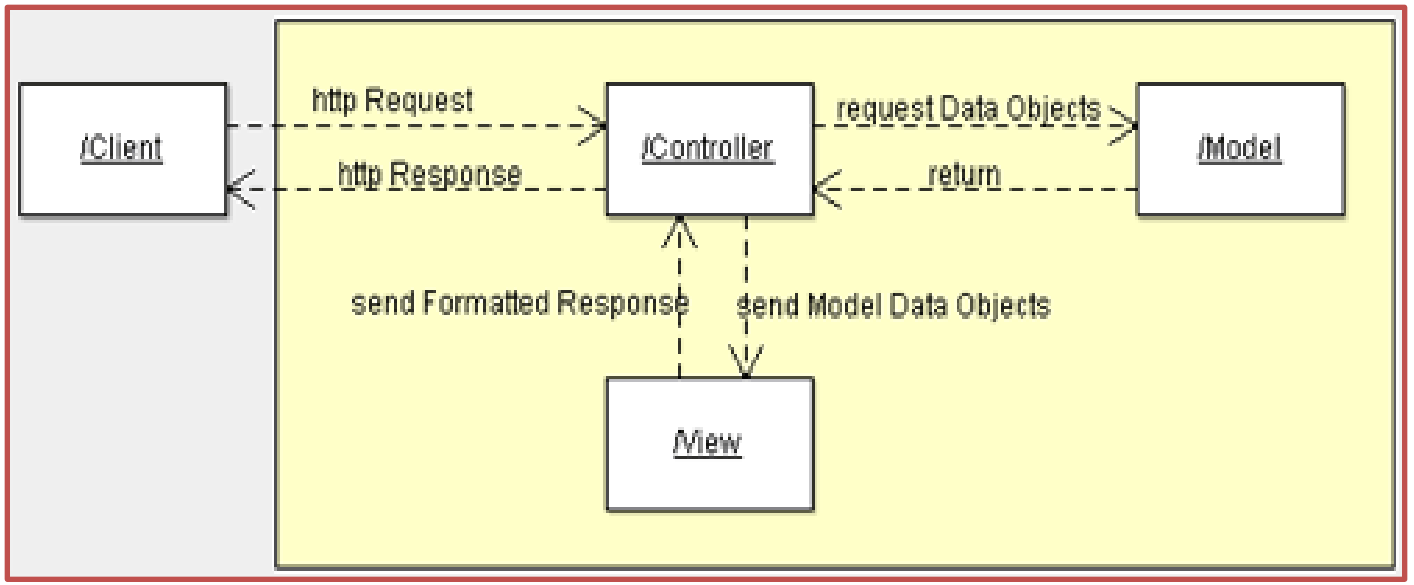
За MVC - **View**

- Изгледът - **View (представянето)** е отговорен за начина (за формата) по който данните, доставени от модела се представят (дисплейват) на потребителите …



За MVC - **Controller**

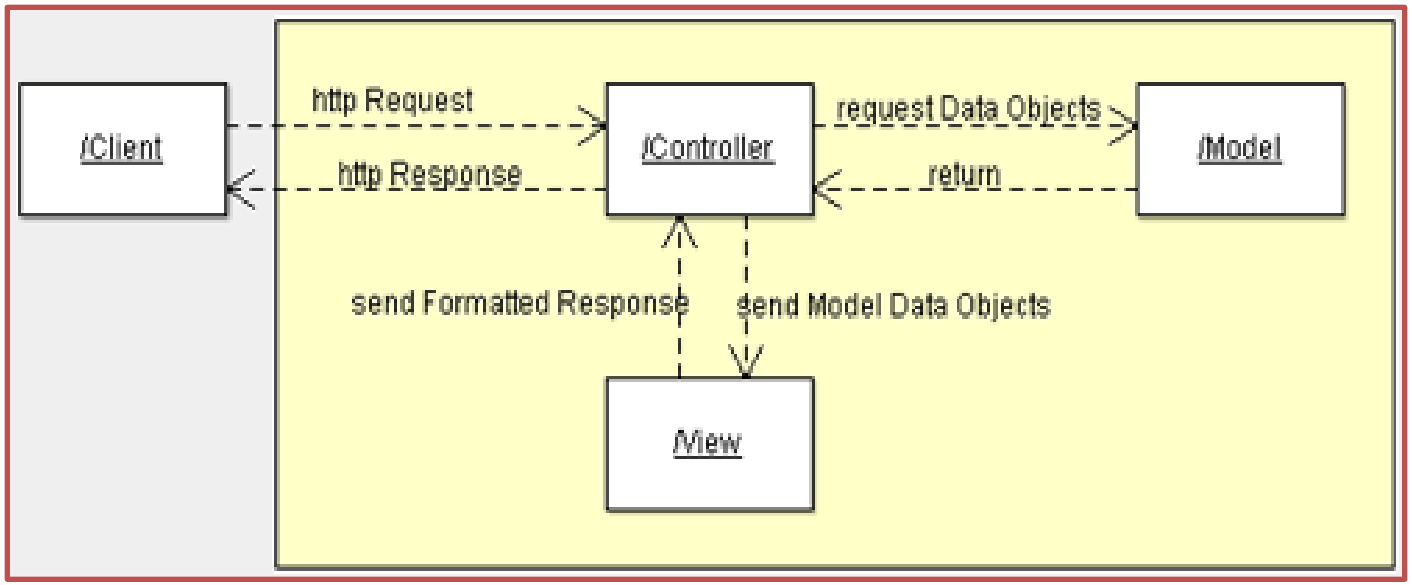
- Контролерът, това е нещо като „администратор” (**Controller), той управлява съвместната работа на двата слоя – на модела (Model) и на изгледа (View)**.



За MVC - **Controller**

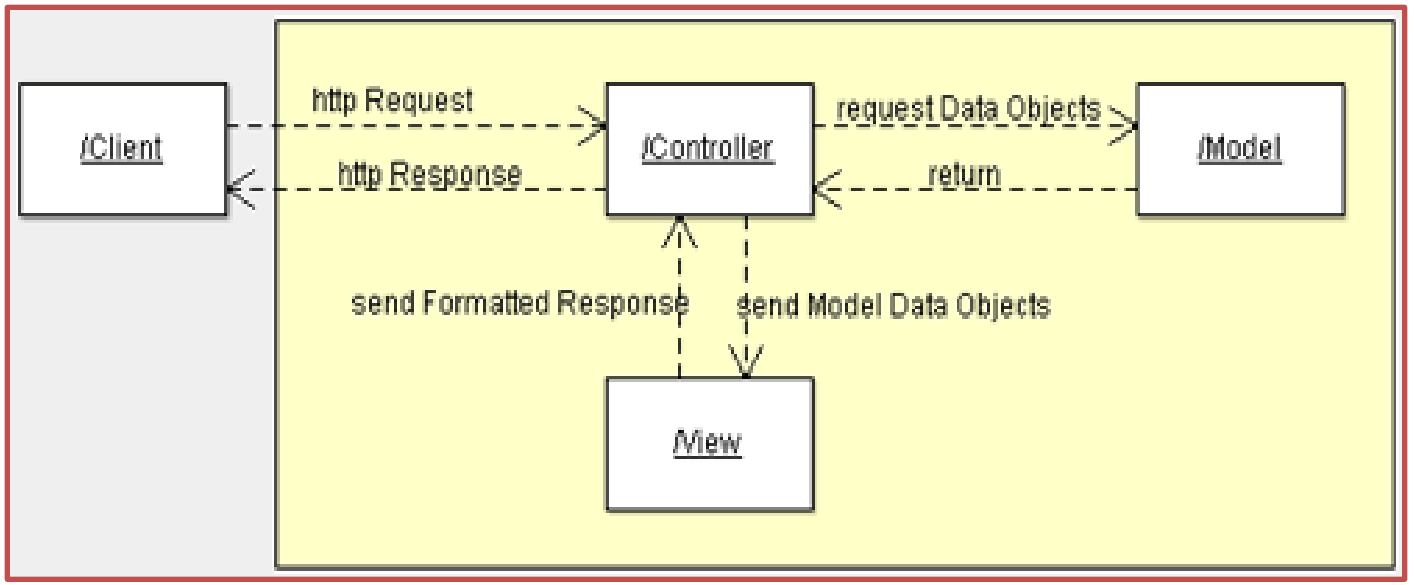
- Той (controller-ът):

1. Получава заявка от клиента, извиква модела (Model) за изпълняване на исканата операция, след което изпраща получените от модела данни на изгледа (View).



За MVC - **Controller**

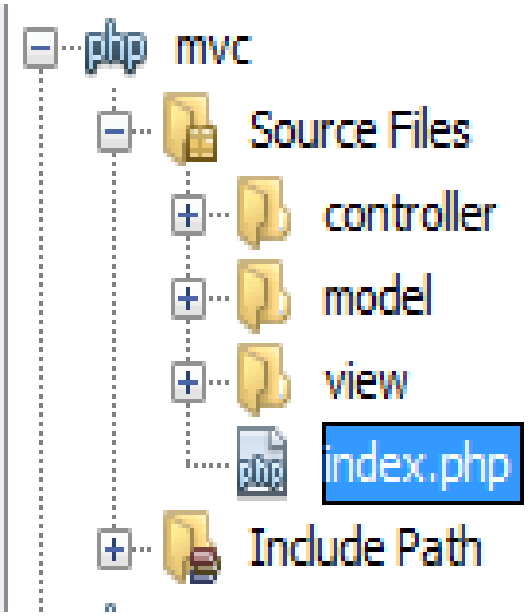
2. Изгледът (view) форматира данните по начин, подходящ за представене на потребителя, в едно web приложение - под формата на html output.

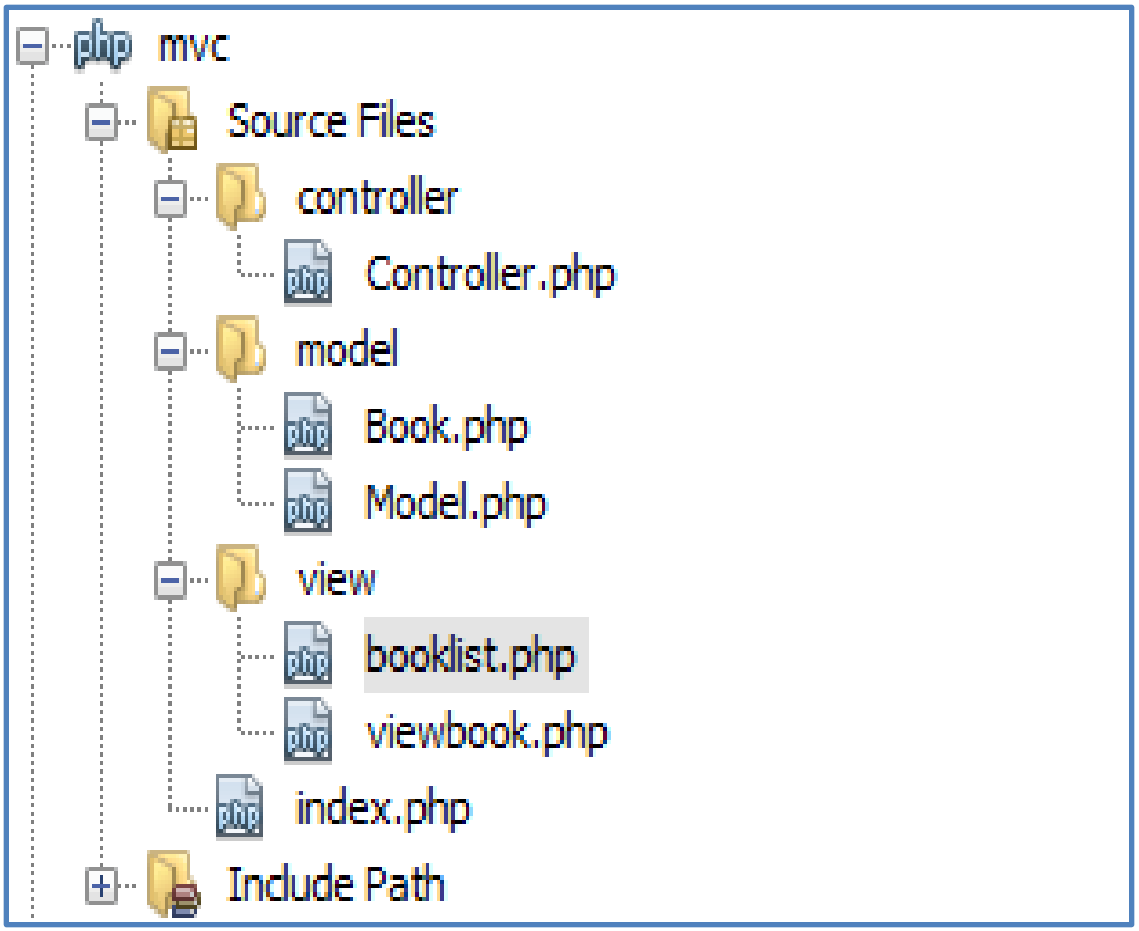


MVC Collaboration Diagram – показва връзките и зависимостите между Модел-Изглед-Контролер

Пример

- php проект, със проста структура: всеки MVC модул е в отделна папка:





Пример: **Контролер (Controller.php)**

- В тази елементарна реализация PHP контролера се изпълнява само от един **клас**, наречен **Controller**, във файл Controller.php, на папка Controller.
- Входна точка на приложението е index.php. Файлът index.php ще **възложи изпълнението** на всички заявки на контролера (**Controller**):

…**$controller->invoke();**

- Контролерът (Controller.php) е този, който **получава заявката, парсва я, инициализира я**, **извиква модела (Model), т.е изпраща му заявката**, след което **получава отговора от модела** и го **изпраща на презентационния слой** (View).
- Той на практика е **връзка** между Model и View - една малка рамка, в която са включени Model и View .

Пример: **Index.php**

Index.php

```
<?php
   include_once("controller/Controller.php");
   $controller = new Controller();
   $controller->invoke();
?>
```

- Файлът index.php **възлага изпълнението** на всички заявки на контролера (**Controller**): инстанциира клас Controller и извиква неговия метод invoke().

Пример: клас Controller

Клас Controller (в **Controller.php**) има само един конструктор и една функция invoke().

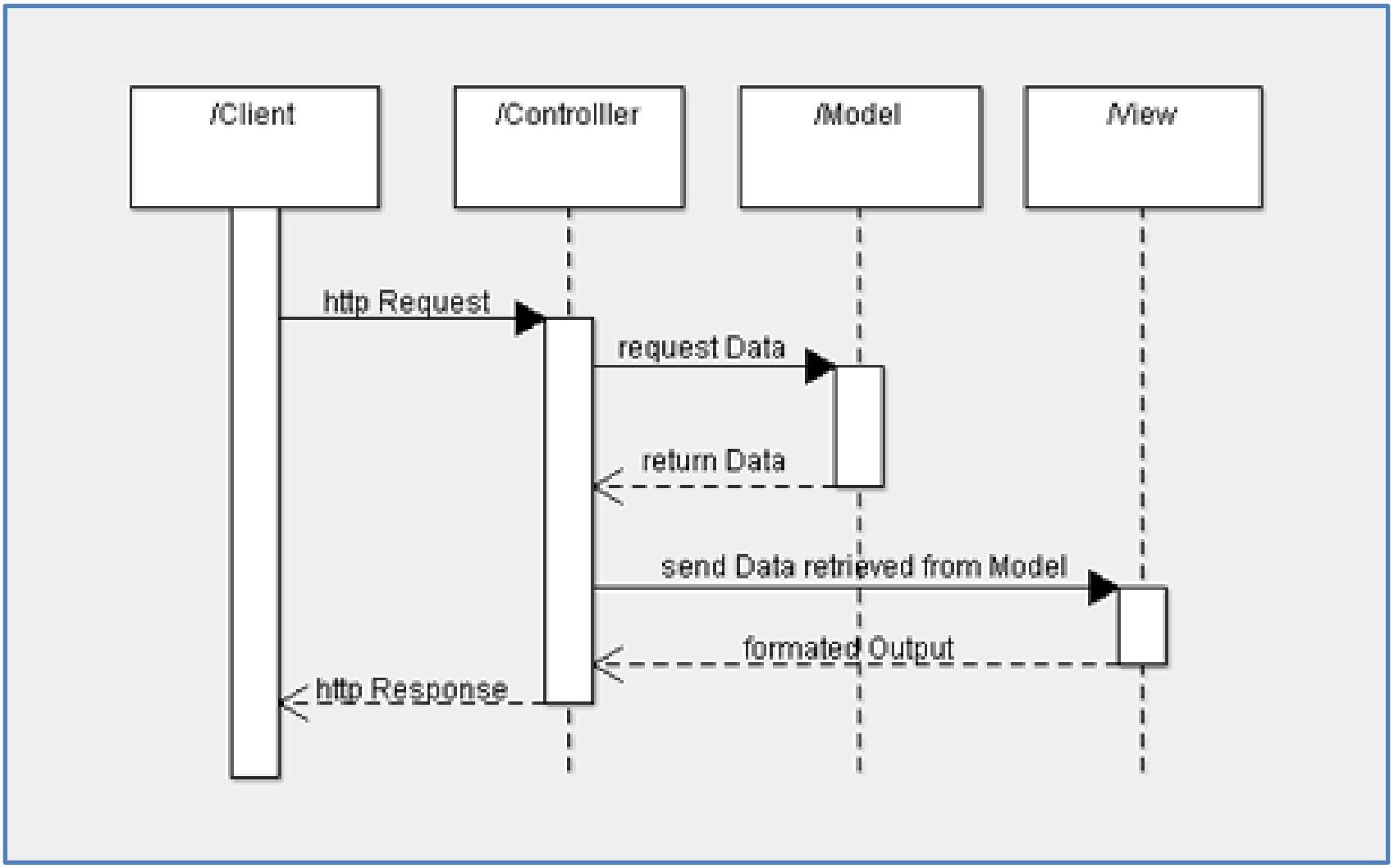
Controller.php

```
<?php
include_once("model/Model.php");  //клас Model в Model.php
class Controller {
      public $model;  //променлива от тип Model
      public function __construct()  // създава една инстация ($model) на клас Model
   {        $this->model = new Model();    }
      public function invoke()
      {        if (!isset($_GET['book']))       //ако не е заявена конкретна книга
               {                                // показва се целия списък от книги
                        $books = $this->model->getBookList();
                        include 'view/booklist.php';
               }
               else                             //ако е заявена конкретна книга
               {                                // показва се заявената книга
                        $book = $this->model->getBook($_GET['book']);
                        include 'view/viewbook.php';
               }
      }
}
?>
```

Пример: клас Controller

- Конструкторът създава една инстация ($model) на клас Model.
- Така, той извиква моделния клас за да получи данните.
- В зависимост от заявката, контролерът решава кои данни да изисква от модела. След което предава данните, идващи от модела към изгледа (клас **booklist или** клас **viewbook**). Кодът е изключително прост.
- Controller-ът не знае нищо за базата данни, или за това, как се генерира страницата.

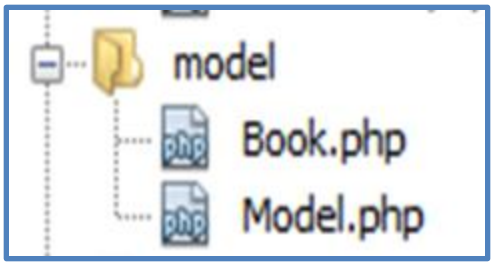
- В следващата MVC Sequence Diagram вие можете да видите потока във времето на една http заявка:



Класове на модела (логика и данни)

- Model-ът представя данните и логиката на едно приложение.

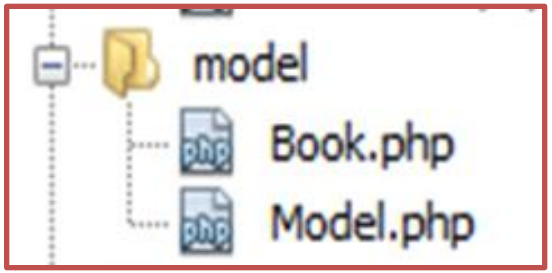
Той е отговорен за:



- **съхраняване, изтриване, актуализиране на данните на приложението**. Обикновено това са операции към базата данни, но прилагането на същите тези операции, за извикване на външни уеб услуги или APIs не е необичайно.
- **капсулиране на логиката на приложението**. Това е слоят, който **трябва да изпълни цялата логика на приложението**. Най-честите грешки са свързани с това, че се възлагат логически операции на някой от другите 2 слоя: controller или view.

Класове на модела (логика и данни)

- В нашия пример, моделът е представен от два класа: клас "Модел" и клас "Book".



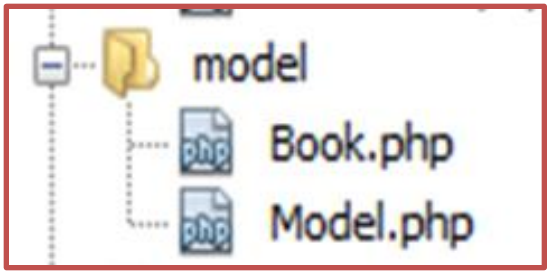
- Клас " Book " е entity клас (т.е това са данните на модела). Този клас ще бъде предложен на View слоя, представя формат, експортиран от моделния слой.
- Клас "Модел" използва клас " Book " и връща (return) конкретна книга, или списък на всички налични книги (клас "Модел"), тоест - реализира бизнес логиката.

Забележка: В някои реализации на MVC реализации се счита, че моделът трябва да предлага единствено entity класове и те не трябва да са обвързани с никаква бизнес логика. Тяхното единствено предназначение е да пазят данни. В зависимост от изпълнението, Entity обектите могат да бъдат заменени с XML или JSON парчета от данни.

Класове на модела (логика)

Model.php

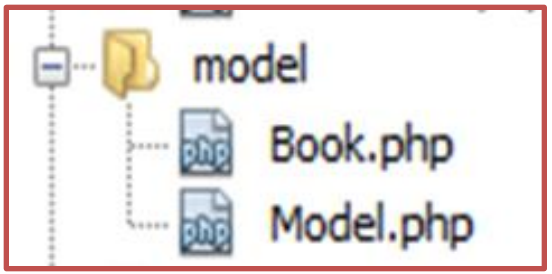
```
<?php
include_once("model/Book.php"); //тук Book class, но в реалния случай БД
class Model {
    public function getBookList()
    { // масив от книги - симулира се извадка, връщана от БД
return array
("Jungle Book" => new Book("Jungle Book", "R. Kipling", "A classic book."),
 "Moonwalker" => new Book("Moonwalker", "J. Walker", ""),
 "PHP for Dummies" => new Book("PHP for Dummies", "Some Smart Guy", "") );
    }
    public function getBook($title)
    {/*за извличане на конкретна книга се използва предната функция за извличане
    на целия масив от книги. Сега се връща само една, а именно - изискваната. В
    реални условия, това би била select команда към БД*/
        $allBooks = $this->getBookList();
        return $allBooks[$title];
    }
}
?>
```



Класове на модела (данни)

- В нашия пример, моделният слой включва и Book class. В реалния случай, това ще е БД, включваща всички даннови единици и класовете, капсулиращи бизнес логиката.

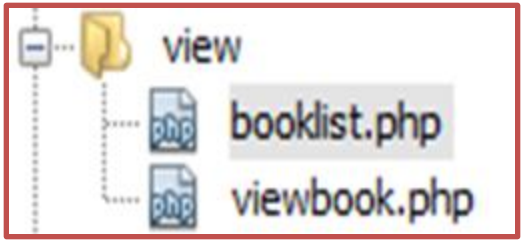
Book.php




```
<?php
class Book {
    public $title;
    public $author;
    public $description;
    public function __construct($title, $author, $description)
   {   $this->title = $title;
       $this->author = $author;
       $this->description = $description;
   }
}
?>
```

Изглед (View – презентиране)

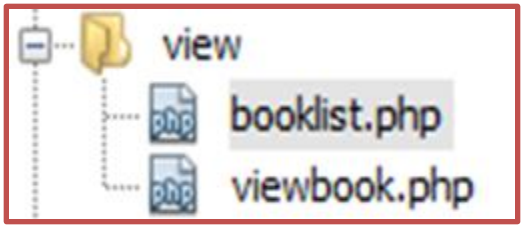
- **View слоят** (presentation layer, т.е презентационния слой) е отговорен за форматиране на получените от модела данни, във формат, достъпен за потребителя.
- Данните могат да се предлагат в различни формати от модела: прости обекти (понякога наричани Value Objects), XML структури, JSON обекти и др ...
- В нашия пример View слоят съдържа само 2 класа, един (във **viewbook.php**) за показване на една книга, а другият (в **booklist.php**) за показване на списък с книги.



Изглед (View – презентиране)

viewbook.php

```
<?php
    echo 'Title:' . $book->title . '<br/>';
    echo 'Author:' . $book->author . '<br/>';
    echo 'Description:' . $book->description . '<br/>';
?>
```



booklist.php

```
<html>
<head></head>
<body>
<table>
    <tr><td>Title</td><td>Author</td><td>Description</td></tr>
    <?php
        foreach ($books as $title => $book)
        {
echo '<tr><td><a href="index.php?book='.$book->title.'">'.$book->title.'</a></td><td>'.$book-
    >author.'</td><td>'.$book->description.'</td></tr>';
        }
    ?>
</table>
</body>
</html>
```

Title:Jungle Book
Author:R. Kipling
Description:A classic book.

Title	Author	Description
Jungle Book	R. Kipling	A classic book.
Moonwalker	J. Walker	
PHP for Dummies	Some Smart Guy	

Изводи, базирани на примера:

- Повечето от PHP уеб рамки, основани на MVC имат подобни реализации (като в показания пример), естествено в много по-сложна форма, т.е това бе една много, много опростена имплементация на MVC .
- Възможността за MVC модел са безкрайни:
  - Например различни слоеве могат да бъдат изпълнени на различни езици или разпределени на различни машини.
  - Контролерът може да бъде частично изпълнен на страната на клиента и частично на сървъра

Предимствата на тази архитектура (Model View Controller)

- Моделният и презентационният слоеве на приложението са разделени, което прави приложението по-гъвкаво:
- Моделът (Model) и представянето (View) могат както **да се променят отделно**, така и **да се заменят отделно**. Например уеб приложението може да се трансформира в по-умно клиентско приложение само чрез писане на нов View модул, или едно приложение може да се промени да използва в „backend“ частта си уеб услуги (вместо да използва база от данни), само чрез подмяна на модул модел.
- Всеки модул може **да се тества** и **да се настройва** отделно.

Още един пример: конвертор от GBP (паунди) към други 3 вида валути: USD, EUR и YEN

Стандартно решение:
(с един компонент):

```
class CurrencyConverter
{
   private $baseValue = 0;
   private $rates = [ 'GBP' => 1.0,  'USD' => 0.6,
      'EUR' => 0.83, 'YEN' => 0.0058 ];
      public function get($currency)
      { if (isset($this->rates[$currency])) {
         $rate = 1/$this->rates[$currency];
         return round($this->baseValue * $rate,
      2); }
      else return 0;
      }
    public function set($amount, $currency =
      'GBP')
   {     if (isset($this->rates[$currency]))
   {  $this->baseValue = $amount * $this-
      >rates[$currency];}
   }
 } //край на класа
```

```
 //Използване на класа
$currencyConverter = new CurrencyConverter();
$currencyConverter->set(100, 'GBP');
echo '<BR>100 GBP is:';
echo $currencyConverter->get('USD') . ' USD / ';
echo $currencyConverter->get('EUR') . ' EUR / ';
echo $currencyConverter->get('YEN') . ' YEN';

$currencyConverter = new CurrencyConverter();
$currencyConverter->set(100, 'USD');
echo '<BR>100 USD is: ';
echo $currencyConverter->get('GBP') . ' GBP / ';
echo $currencyConverter->get('EUR') . ' EUR / ';
echo $currencyConverter->get('YEN') . ' YEN';
```

Изход:

```
100 GBP is: 166.67 USD / 120.48 EUR / 17241.38 YEN
100 USD is: 60 GBP / 72.29 EUR / 10344.83 YEN
```

Към MVC. 1.Реализация на модела

Нека да отделим класа **CurrencyConverter** в един файл, с име Model.php. Използването му ще бъде от index.php:

Model.php

```
class CurrencyConverter {
    private $baseValue = 0;
    private $rates = [
        'GBP' => 1.0,
        'USD' => 0.6,
        'EUR' => 0.83,
        'YEN' => 0.0058
    ];

    public function get($currency) {
        if (isset($this->rates[$currency])) {
            $rate = 1/$this->rates[$currency];
            return round($this->baseValue * $rate, 2);
        }
        else return 0;
    }

    public function set($amount, $currency = 'GBP')
    {     if (isset($this->rates[$currency]))
{   $this->baseValue = $amount * $this->rates[$currency];      }
    }
   } //край на класа
```

Към MVC. 1.Реализация на модела

Използването на Model.php е в Index.php:

Index.php (вариант 1)

```
require_once 'Model.php';
$currencyConverter = new CurrencyConverter;
$currencyConverter->set(100, 'GBP');
echo '<BR>100 GBP is:';
echo $currencyConverter->get('USD') . ' USD / ';
echo $currencyConverter->get('EUR') . ' EUR / ';
echo $currencyConverter->get('YEN') . ' YEN';

$currencyConverter = new CurrencyConverter;
$currencyConverter->set(100, 'USD');
echo '<BR>100 USD is: ';
echo $currencyConverter->get('GBP') . ' GBP / ';
echo $currencyConverter->get('EUR') . ' EUR / ';
echo $currencyConverter->get('YEN') . ' YEN';
```

Изход:

100 GBP is:166.67 USD / 120.48 EUR / 17241.38 YEN
100 USD is: 60 GBP / 72.29 EUR / 10344.83 YEN

Следваща стъпка: 2.Изглед (View)

Изглед (View)

- Тъй като един конвертор на валута изисква потребителят да може да въведе стойност за валутата, която ще конвертираме (например 100), то следната форма би била подходяща за осъществяване на комуникацията:

- Така, ще създадем class CurrencyConverterView (View.php) със следното примерно съдържание (това, ще доведе и съответно до промяна на index.php):

Следваща стъпка – изглед (клас **CurrencyConverterView**)

View.php

```
<?php
class CurrencyConverterView {
    private $converter;
    private $currency;

    public function __construct(CurrencyConverter $converter, $currency) {
        $this->converter = $converter;
        $this->currency = $currency;
    }

    public function output() {
        $html = '<form action="?action=convert" method="post"><input name="currency"
      type="hidden" value="' . $this->currency .'"/><label>' . $this->currency
      .':</label><input name="amount" type="text" value="' . $this->converter->get($this-
      >currency) . '"/><input type="submit" value="Convert"/></form>';
       return $html;
    }
}
?>
```

Актуализиране на **Index.php**

Index.php (вариант 2)

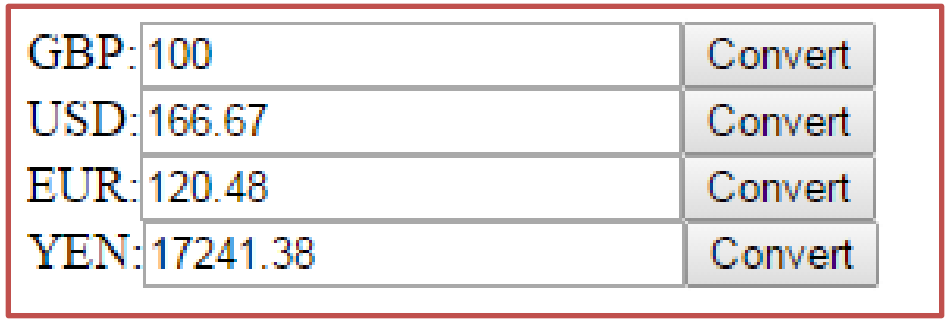
```
<?php
 require_once 'Model.php';
 require_once 'View.php';
$currencyConverter = new CurrencyConverter;
$currencyConverter->set('100', 'GBP');
$view = new CurrencyConverterView($currencyConverter, 'EUR');
echo $view->output();
?>
```

Сега коректно се отпечатва стойността в евро на 100 GBP (без натискане на бутона:

3. Преизползване на изгледа

Index.php (вариант 3)

```
<?php
  require_once 'Model.php';
  require_once 'View.php';

$currencyConverter = new CurrencyConverter;
$currencyConverter->set('100', 'GBP');
```




```
$gbpView = new CurrencyConverterView($currencyConverter, 'GBP');
echo $gbpView->output();

$usdView = new CurrencyConverterView($currencyConverter, 'USD');
echo $usdView->output();

$eurView = new CurrencyConverterView($currencyConverter, 'EUR');
echo $eurView->output();

$yenView = new CurrencyConverterView($currencyConverter, 'YEN');
echo $yenView->output();
?>
```

Сега коректно се отпечатва стойността в различни валути на 100 GBP:
Проблемът е че не се позволява „user input” и нищо не се случва все още при щракане на бутона Convert!

Следваща стъпка към MVC решение:

3. Създаване на контролер

Позволяване на user input и разрешаване работата на бутона

И така, контролерът (controller) е този в MVC, който ще вземе заявката на потребителя („user input”) и ще я изпрати на модела:

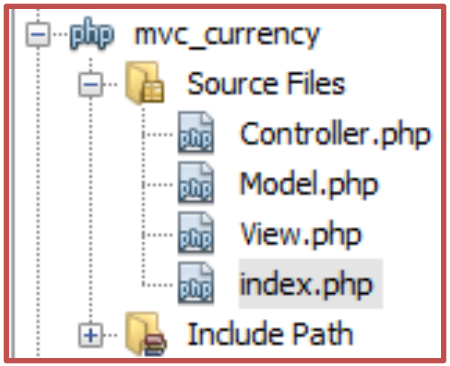
Controller.php

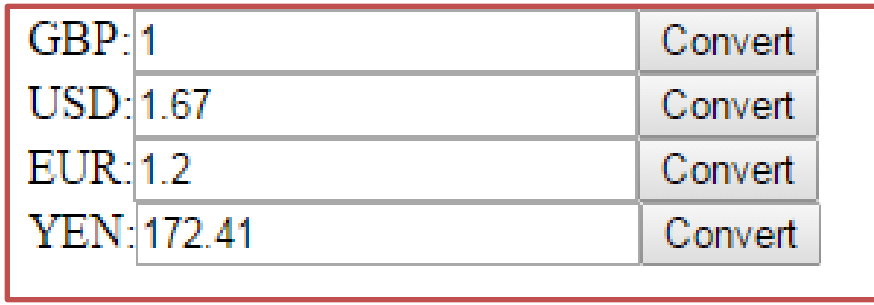
```
<?php
class CurrencyConverterController {
   private $currencyConverter;

   public function __construct(CurrencyConverter $currencyConverter) {
      $this->currencyConverter = $currencyConverter;
   }

   public function convert($request) {
      if (isset($request['currency']) && isset($request['amount'])) {
         $this->currencyConverter->set($request['amount'], $request['currency']);
      }
   }
?>
```

Актуализиране на index.php

Index.php (вариант 4)

```
<?php
  require_once 'Model.php';
  require_once 'View.php';
  require_once 'Controller.php';

$model = new CurrencyConverter();
$controller = new CurrencyConverterController($model);

//Check for presence of $_GET['action'] to see if a controller action is required
if (isset($_GET['action'])) $controller->{$_GET['action']}($_POST);

$gbpView = new CurrencyConverterView($model, 'GBP');
echo $gbpView->output();

$usdView = new CurrencyConverterView($model, 'USD');
echo $usdView->output();

$eurView = new CurrencyConverterView($model, 'EUR');
echo $eurView->output();

$yenView = new CurrencyConverterView($model, 'YEN');
echo $yenView->output();
?>
```





Използвана литература

- http://php-html.net/tutorials/model-view-controller-in-php/
- http://sourceforge.net/projects/mvc-php/files/mvc.zip/download
- https://r.je/mvc-tutorial-real-application-example.html