# PHP MVC

Reggie Niccolo Santos
UP ITDC

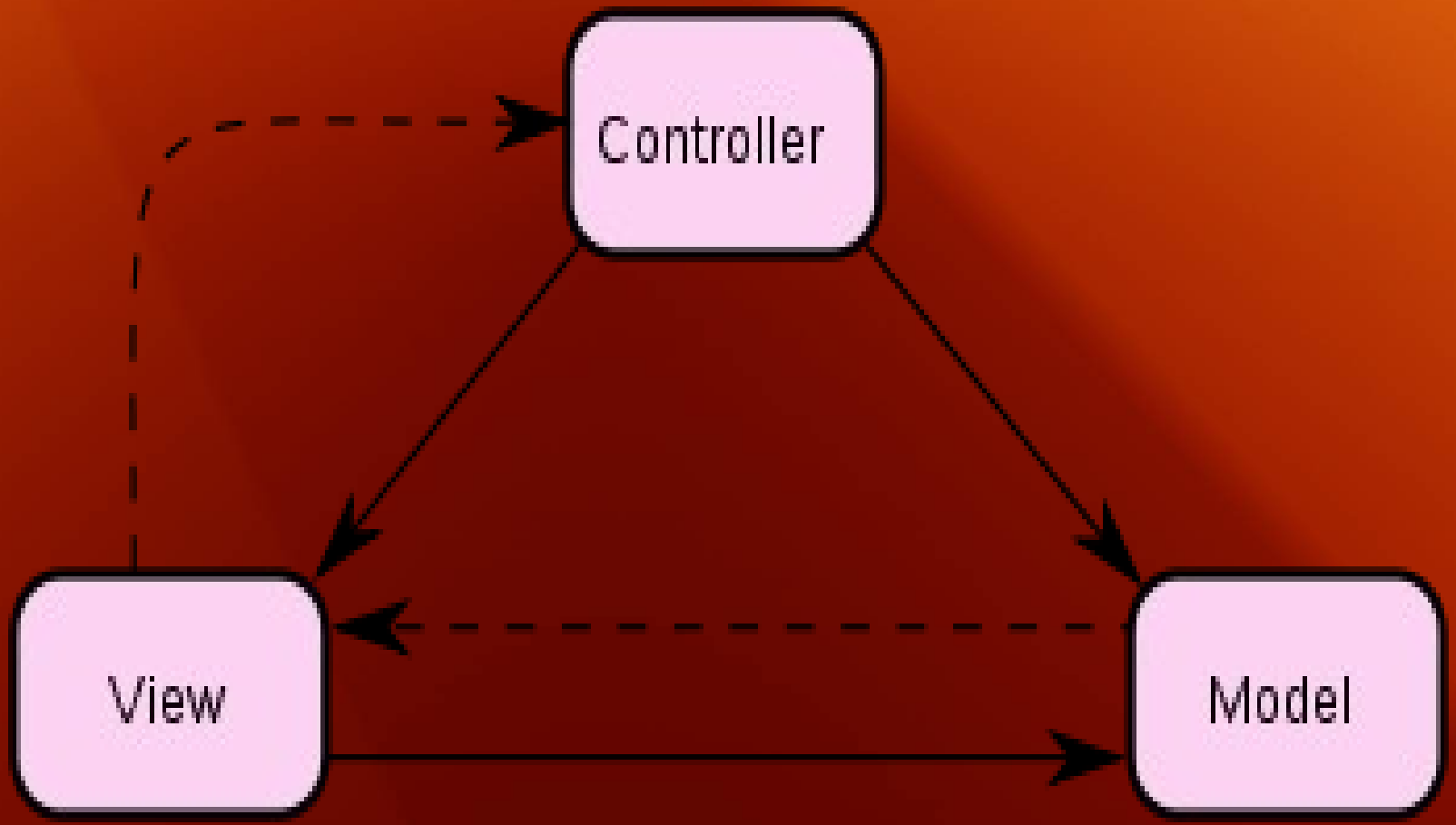
# Outline

- Definition
- Sample MVC application
- Advantages

# MVC

- Model-View-Controller
- Most used architectural pattern for today's web applications
- Originally described in terms of a design pattern for use with Smalltalk by Trygyve Reenskaug in 1979
  - The paper was published under the title ”Applications Programming in Smalltalk-80: How to use Model-View-Controller”

# MVC

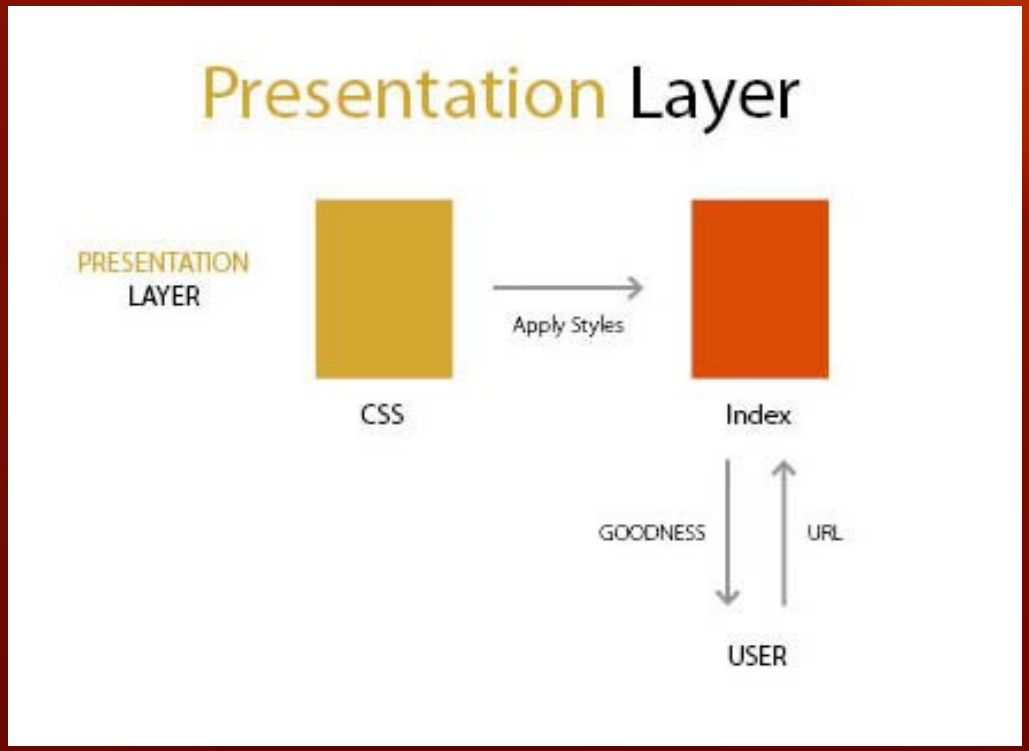


# Model

- Responsible for managing the data
- Stores and retrieves entities used by an application, usually from a database but can also include invocations of external web services or APIs
- Contains the logic implemented by the application (business logic)

# View (Presentation)

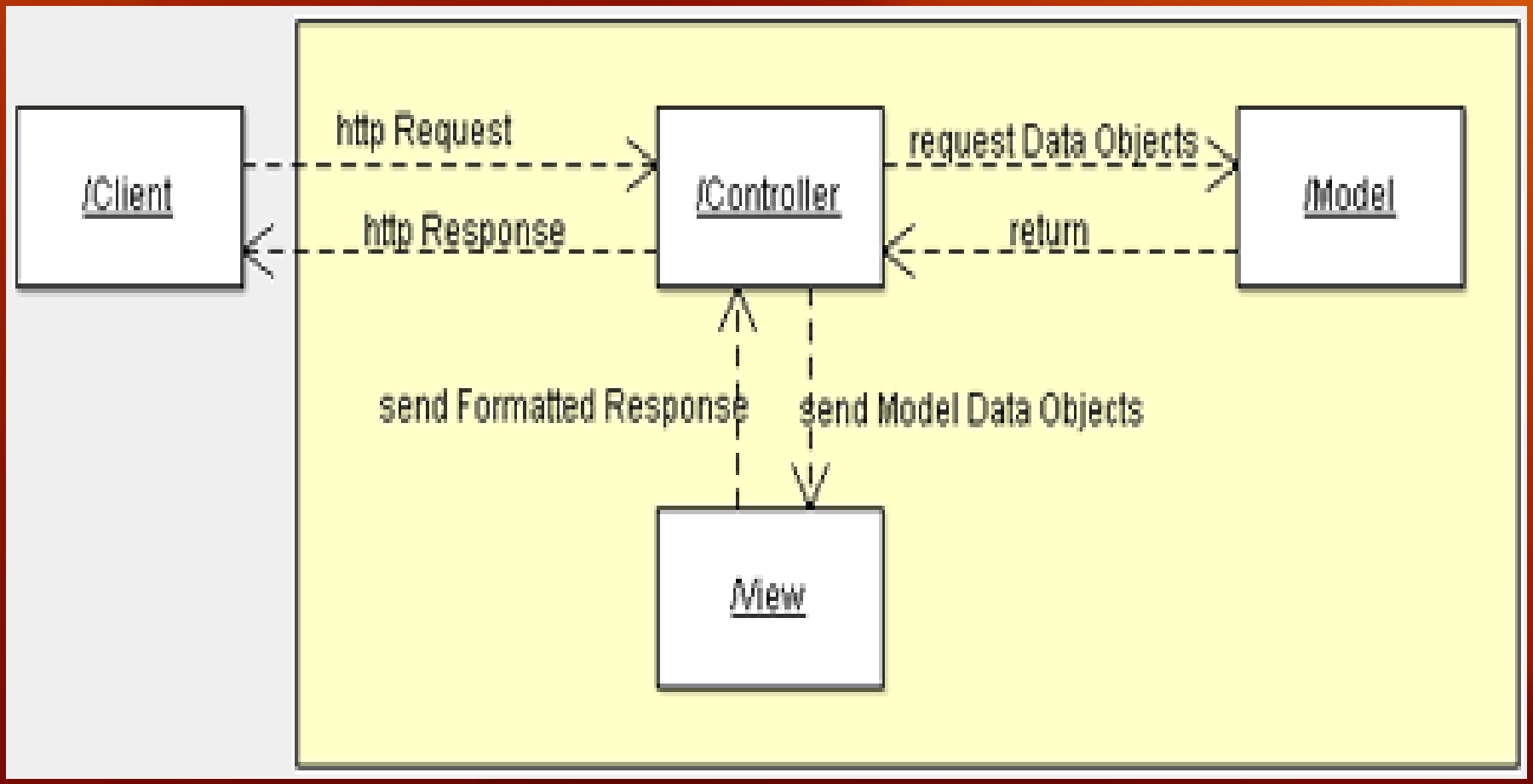
- Responsible to display the data provided by the model in a specific format



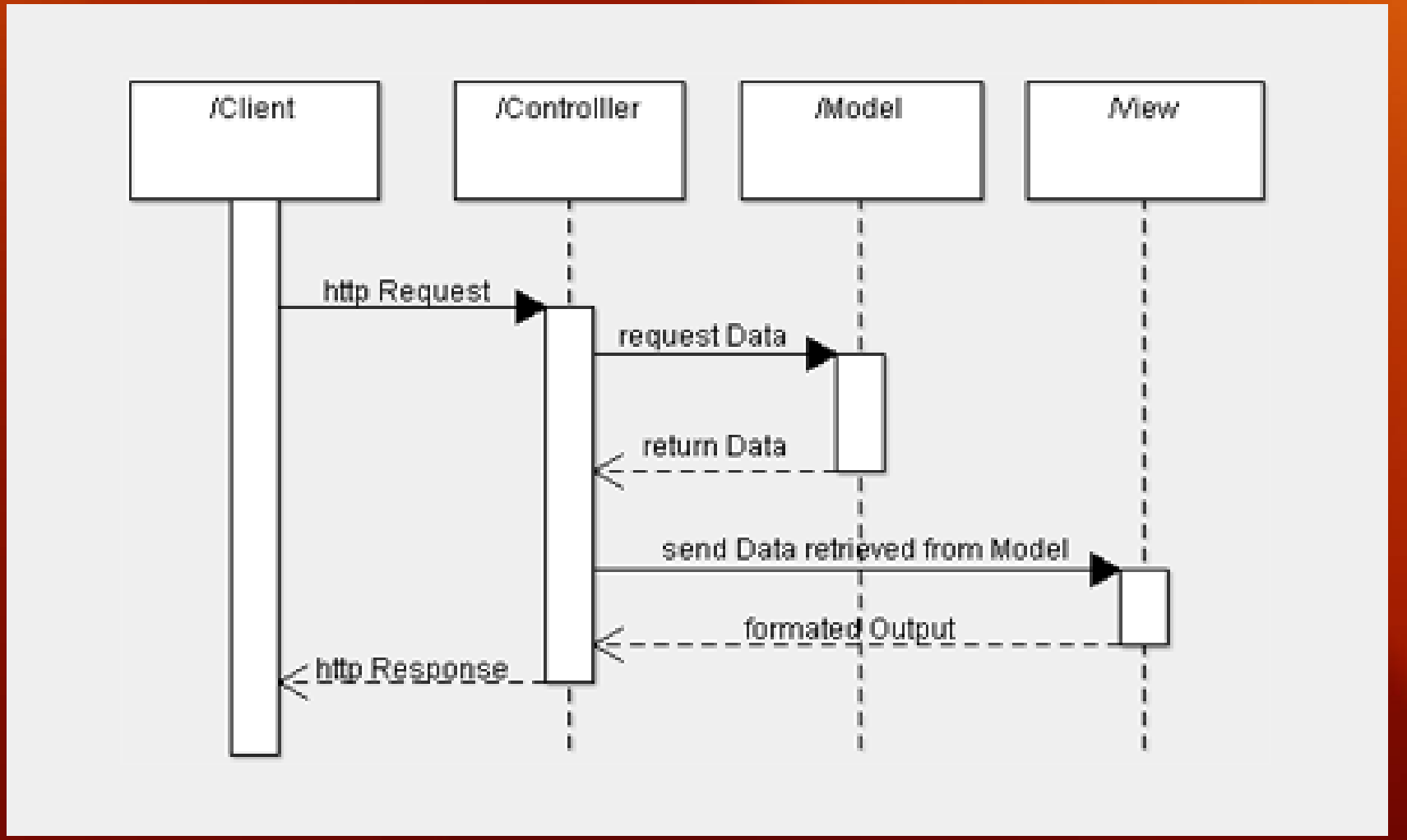
# Controller

- Handles the model and view layers to work together
- Receives a request from the client, invokes the model to perform the requested operations and sends the data to the view

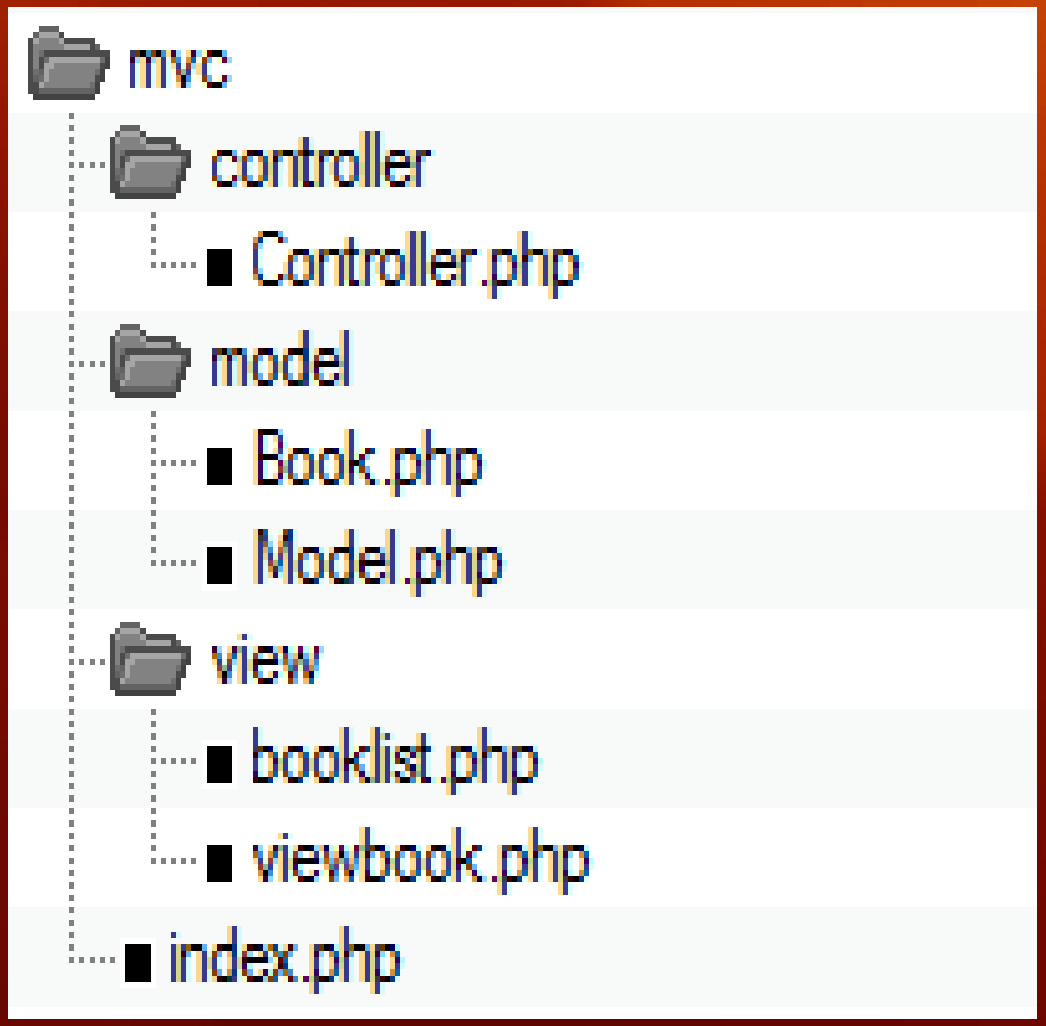
# MVC Collaboration Diagram



# MVC Sequence Diagram



# Sample MVC Application



# Entry point - index.php

```
require_once("controller/Controller.php");

$controller = new Controller();
$controller->invoke();
```

# Entry point - index.php

- The application entry point will be index.php
- The index.php file will delegate all the requests to the controller

# Controller – controller/Controller.php

```
require_once("model/Model.php");

class Controller {
     public $model;

     public function __construct()
     {
          $this->model = new Model();
     }

    // continued...
```

# Controller – controller/Controller.php

```
require_once("model/Model.php");

class Controller {
     public $model;

     public function __construct()
     {
          $this->model = new Model();
     }

    // continued...
```

# Controller – controller/Controller.php

```
    public function invoke()
    {
            if (!isset($_GET['book']))
            {
                // no special book is requested, we'll
    show a list of all available books
                $books = $this->model->getBookList();
                include 'view/booklist.php';
            }
            else
            {
                // show the requested book
                $book = $this->model->getBook
    ($_GET['book']);
                include 'view/viewbook.php';
            }
        }
}
```

# Controller – controller/Controller.php

- The controller is the first layer which takes a request, parses it, initializes and invokes the model, takes the model response and sends it to the view or presentation layer

# Model – model/Book.php

```
class Book {
  public $title;
  public $author;
  public $description;

  public function __construct($title, $author,
  $description)
     {
        $this->title = $title;
       $this->author = $author;
       $this->description = $description;
     }
}
```

# Model - Entities

- Their sole purpose is to keep data
- Depending on the implementation, entity objects can be replaced by XML or JSON chunk of data
- It is recommended that entities do not encapsulate any business logic

# Model – model/Model.php

```
require_once("model/Book.php");

class Model {
  public function getBookList()
  {
    // here goes some hardcoded values to simulate
  the database
    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", "")
    );
  }

  // continued...
```

# Model – model/Model.php

```
  public function getBook($title)
  {
    // we use the previous function to get all the
  books and then we return the requested one.
    // in a real life scenario this will be done
  through a db select command
    $allBooks = $this->getBookList();
    return $allBooks[$title];
  }
}
```

# Model

- In a real-world scenario, the model will include all the entities and the classes to persist data into the database, and the classes encapsulating the business logic

# View – view/viewbook.php

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

# View – view/booklist.php

```
<html>
  <head></head>
  <body>
    <table><tbody>
      <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>';
        }
    ?>
  </tbody></table>
  </body>
</html>
```

# View

- Responsible for formatting the data received from the model in a form accessible to the user
- The data can come in different formats from the model: simple objects (sometimes called Value Objects), XML structures, JSON, etc.

# Advantages

- The Model and View are separated, making the application more flexible
- The Model and View can be changed separately, or replaced
- Each  module can be tested and debugged separately

# References

http://php-html.net/tutorials/model-view-controller-in-p
http://www.htmlgoodies.com/beyond/php/article.php/3

http://www.phpro.org/tutorials/Model-View-Controller-

http://www.particletree.com/features/4-layers-of-sepa