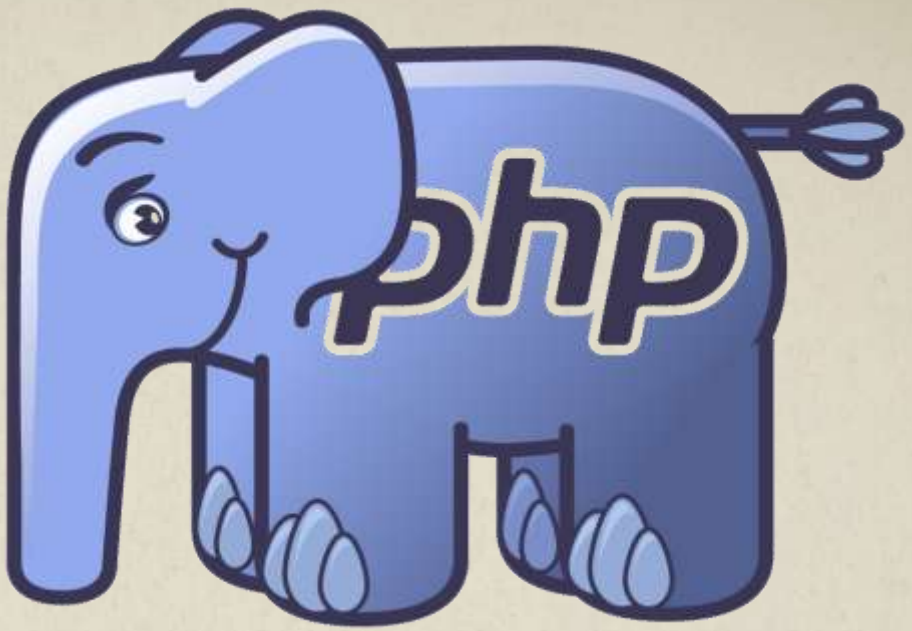
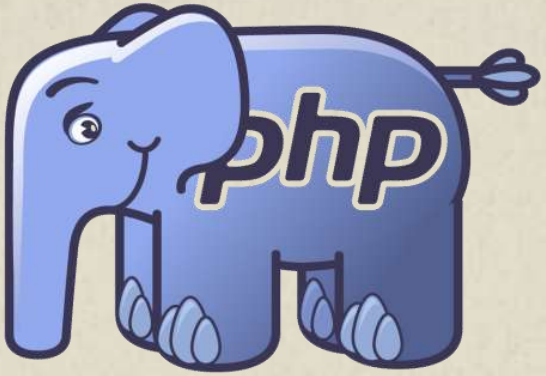


PHP: ВВЕДЕНИЕ В ПРОФЕССИЮ

УРОК 8. АРХИТЕКТУРА ПРОЕКТА



АВТОЗАГРУЗКА

В чём проблема?

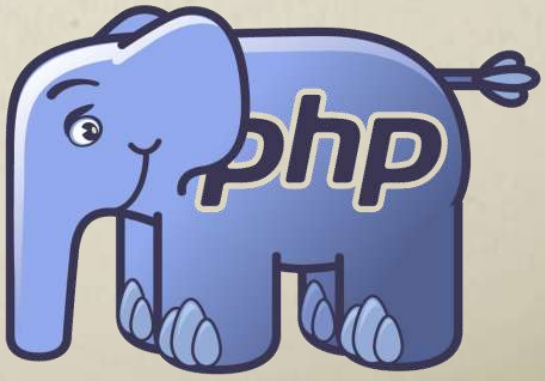
- В том, что когда нам нужен некий класс, мы должны писать `require`
- И даже `require_once` не особенно спасает – всё равно приходится вручную отслеживать загрузку классов из файлов ...

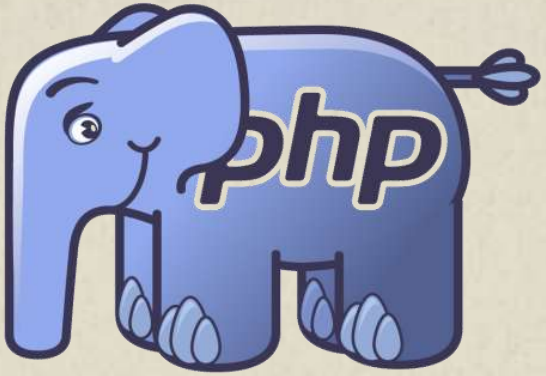
Что делать?

```
function __autoload($className) {
  require
    __DIR__ . '/classes/' .
    $className . '.php';
}
```

- Функцию `__autoload()` нужно только объявить. Вызывать ее не требуется.
- Название функции начинается с двух знаков подчеркивания – это признак «магии» в PHP
- Функция будет вызвана автоматически, когда ваш код встретит ранее неизвестное имя класса. Это имя будет передано функции. Ее задача – подключить файл, где этот класс определен.

АВТОЗАГРУЗКА





ПРОСТРАНСТВА ИМЕН

В чём проблема?

- Однажды в нашем проекте станет много классов.
- Вы пока еще не знаете, что такое «контроллер». Представьте, что это класс «страницы сайта». Десятки и сотни страниц – десятки и сотни классов.
- А еще модели. Тоже десятки и сотни. Кошмар ☹
- Работа с БД? Конфиг? … Современное веб-приложение может легко иметь сотни классов!

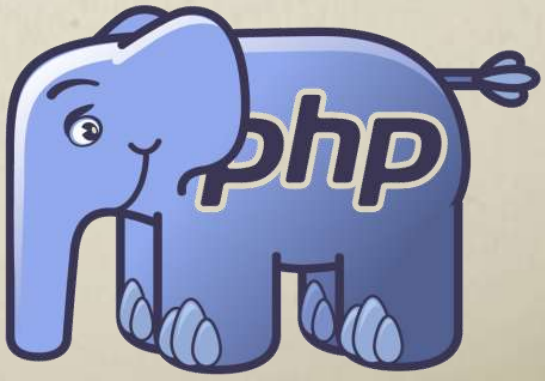
Что делать?

Было бы здорово, если бы мы могли организовать нечто вроде «папок» для классов!

```
Application
🗁 Controllers
   🗁 Index
   🗁 News
🗁 Models
   🗁 News
   🗁 User
```

Круто! Но как же организовать такую «вложенность» классов? Иерархию?

NAMESPACE



Пространство имён – своеобразная «папка» в которой может находиться класс

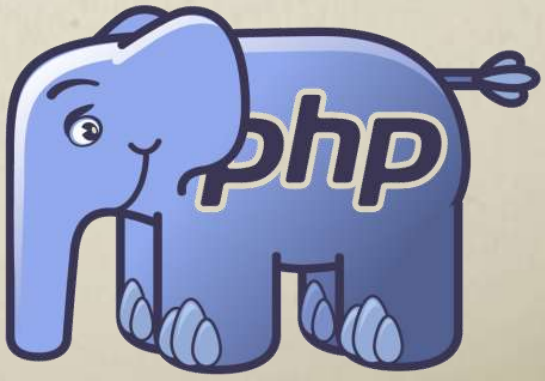
```
namespace App\Models;

class News {
  ...
}
```

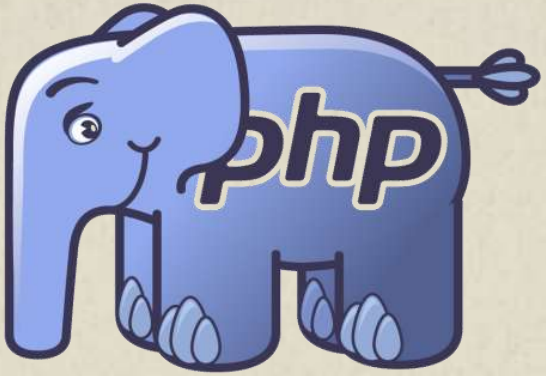
- Конструкция `namespace` пишется в начале файла и действует до его конца.
- Полным именем класса в этом примере будет `\App\Models\News`

```
$model = new \App\Models\News();
```

- Можно использовать конструкцию `use`, чтобы задать короткие синонимы для «длинных» имен:
  ```
  use \App\Models\News as Model;
  $model = new Model();
  ```
- Считается, что классы, у которых не указано пространство имён, находятся в «корневом». То есть можно писать `\Config` или просто `Config`
- **ВАЖНО!** В функцию автозагрузки будет передано ПОЛНОЕ имя класса, включая пространство имён от корневого!

NAMESPACE





СТАНДАРТЫ PSR

PSR - это стандарты кодирования на PHP.

Знание и соблюдение стандартов не менее важно, чем знание самого языка.

PSR-0 Стандарт автозагрузки (устарел, см. PSR-4)

- Каждое пространство имён – папка в файловой системе
- Имя файла с классом = Имя класса.php

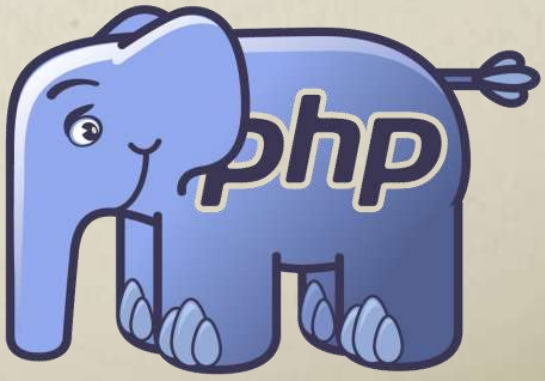
PSR-1 Основной стандарт кодирования

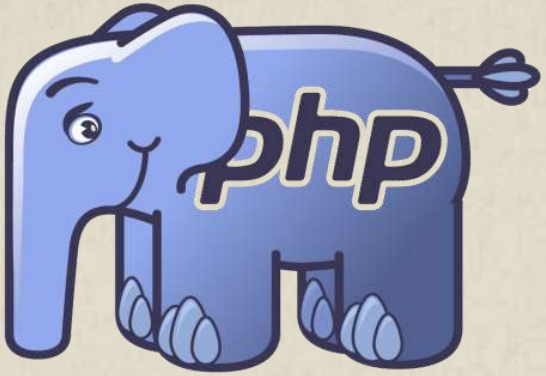
- Только UTF-8!
- Только <?php
- ClassName
- methodName
- CONST_NAME

PSR-1 Руководство по стилю кода

- 4 пробела, никакой табуляции
- true, false, null, ключевые слова
- нельзя использовать var
- видимость – обязательно!

PSR





ПАТТЕРН MVC

Архитектура - это базовая организация системы, воплощенная в ее компонентах, их отношениях между собой и с окружением, а также принципы, определяющие проектирование и развитие системы. [IEEE 1471]

А если по-русски?

- Компоненты кода («на какие части разбить»)
- Структура кода («где что будет лежать»)
- Отношения между компонентами

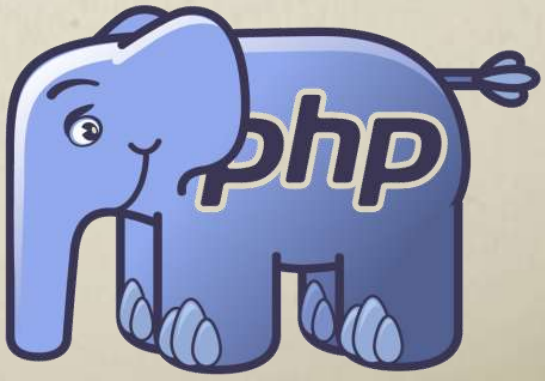
Сложно сразу продумать хорошую архитектуру

- Но делать это нужно!
- Но, к счастью, всё придумано до нас ☺

Паттерны проектирования – «шаблоны» построения программы, следуя которым, можно получить удовлетворительный результат.

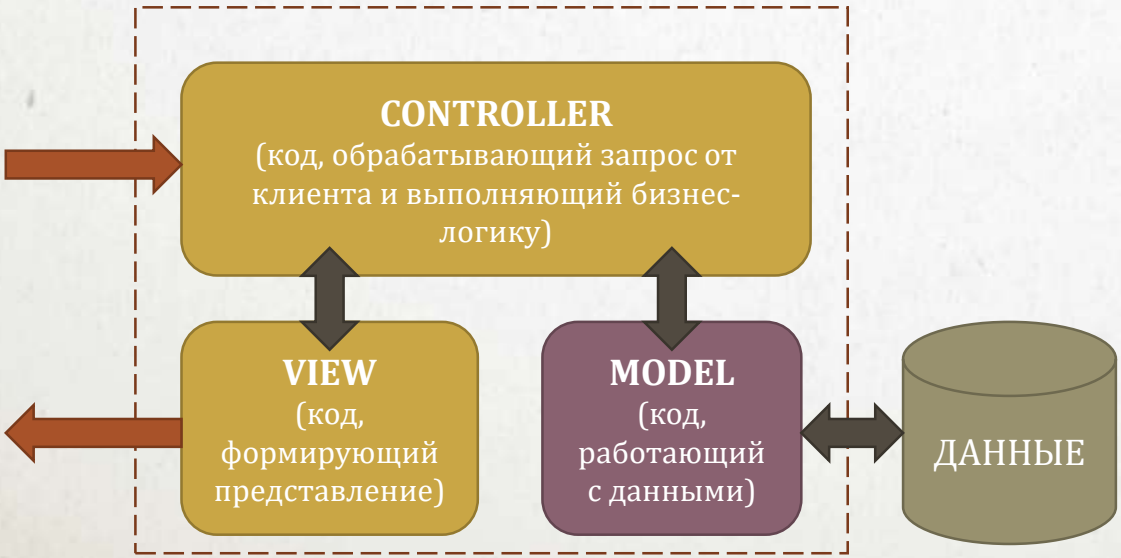
Пожалуй, главный паттерн - **MVC**

АРХИТЕКТУРА

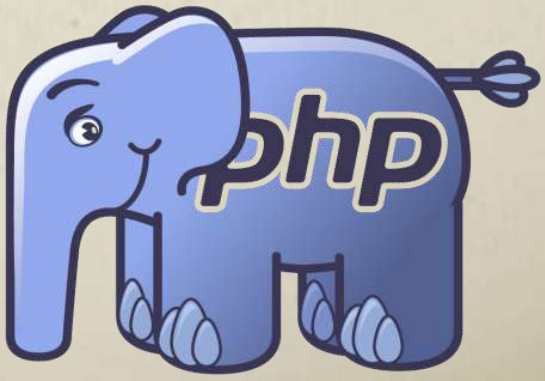


Модель работает с данными

- Ее задача – дать нужные данные тем компонентам, которые их запрашивают
- Модель и только модель, знает, где данные находятся, как они организованы
- Модель и только модель умеет обновлять, вставлять и удалять данные

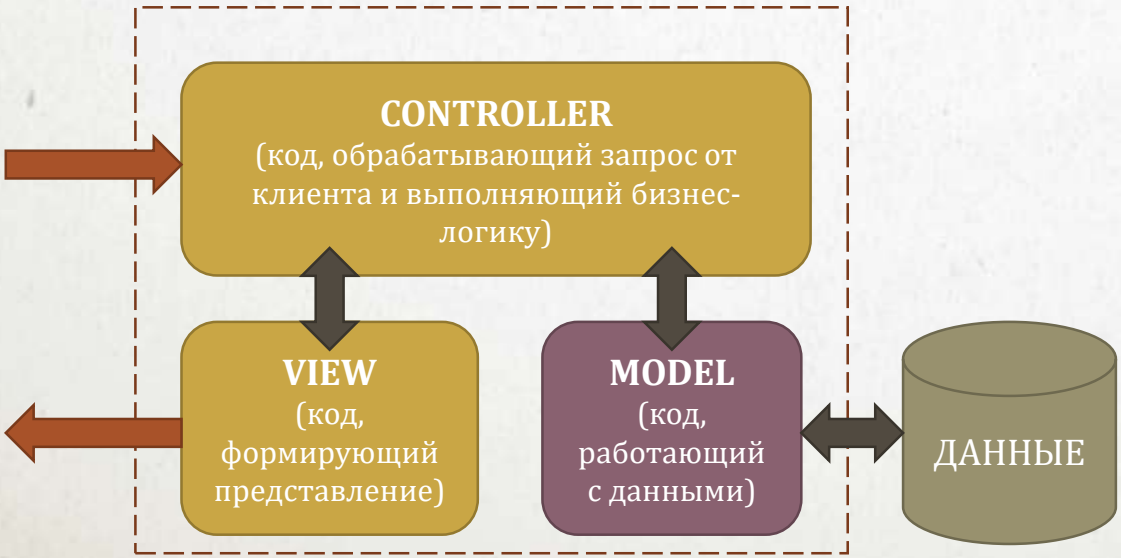


АРХИТЕКТУРА

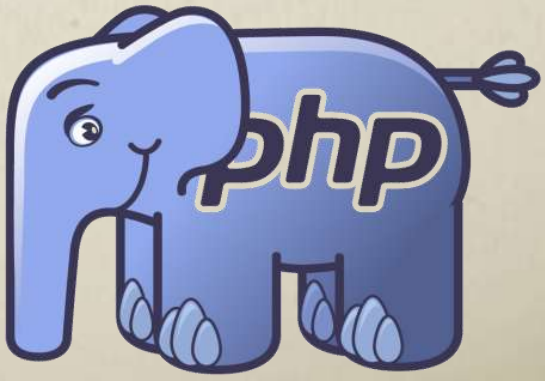


Представление создает внешний вид

- Его задача – сформировать ответ на запрос клиента
- Представление использует HTML, CSS, JavaScript
- Представление может содержать логику, нужную для вывода данных, она может быть написана на PHP – не бойтесь смешивать PHP и HTML!

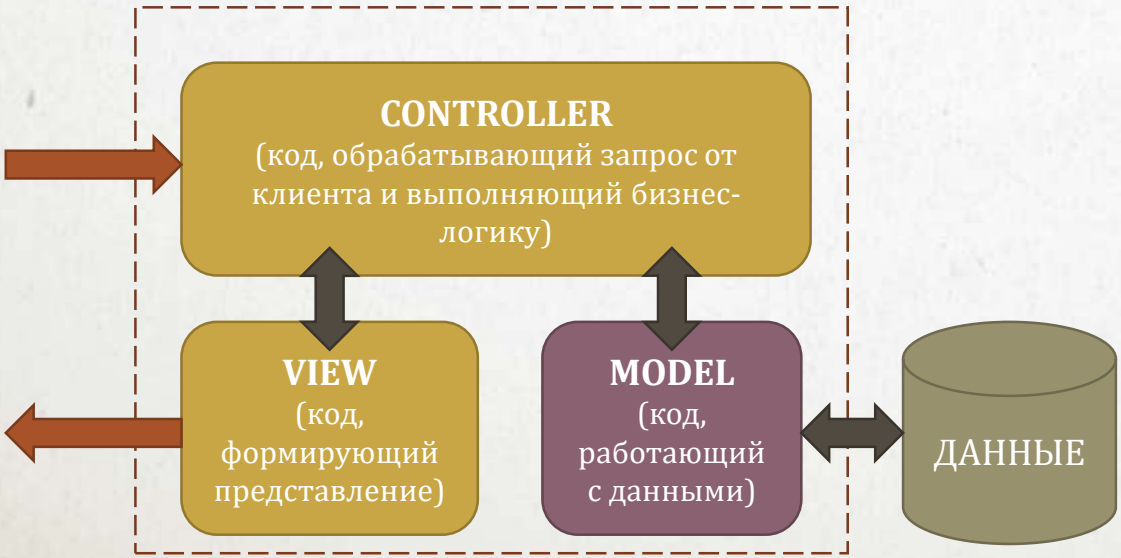


АРХИТЕКТУРА



Контроллер – точка входа и логика приложения

- Принимает запрос от клиента (является «точкой входа»)
- Получает нужные данные от моделей или дает моделям команды на модификацию данных
- Обрабатывает эти данные в соответствии с бизнес-логикой
- Передает данные представлению и инициирует ответ на запрос клиента



АРХИТЕКТУРА