

# Internet ТЕХНОЛОГИИ

ЛЕКЦИЯ №7  
СПОСОБЫ ПОСТРОЕНИЯ АРХИТЕКТУРЫ WEB-  
ПРИЛОЖЕНИЙ.  
ПРИМЕНЕНИЕ ШАБЛОНОВ.  
NodeJS и ANGULARJS

# Содержание

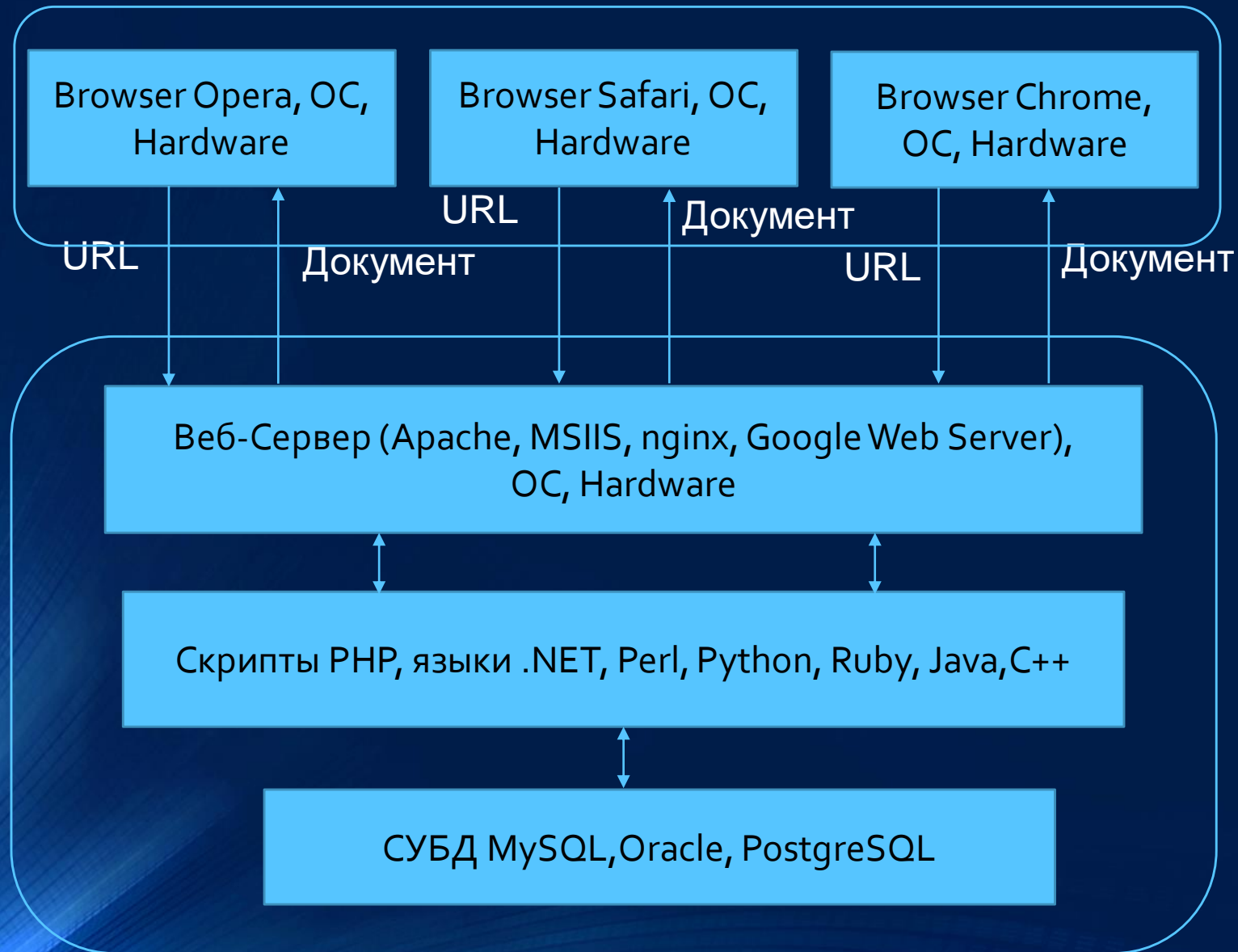
- Способы построения архитектуры веб-приложения
- Принципы двухзвенной, трехзвенной и  $N$ -звенной клиент-серверной архитектуры веб-приложения.
- Принципы и разновидности MVC.
- Применение шаблонов. Классификация шаблонов.
- Механизмы шаблонов PHP. Шаблонизаторы Smarty, Twig.
- CMS и фреймворки.
- Node.js
- SPA
- AngularJS. Введение, директивы, сервисы, разбор примеров.

# Способы построения архитектуры веб-приложения

Архитектура	Описание
<i>Клиент/сервер</i> (2-уровневая)	Система разделяется на два приложения, где <b>клиент</b> выполняет запросы к <b>серверу</b> , ожидает ответы и обрабатывает их при получении. Сервер не обращается к сторонним ресурсам для выполнения какой-либо части запроса.
<i>N-уровневая</i> (3-уровневая)	Функциональные части приложения разделяются на уровни, которые могут физически располагаться на разных компьютерах.
<i>Offline first</i>	Клиентское приложение должно быть способно обойтись без сервера, предоставляя минимальный функционал пользователя. Размещение всего стека MVC на стороне клиента.

**Архитектура** веб-приложения определяет используемые компоненты и способы их взаимодействия между собой. Детали, относящиеся исключительно к внутренней реализации – не являются архитектурными.

# 2-уровневая архитектура клиент-сервер



Клиентская часть



Сеть

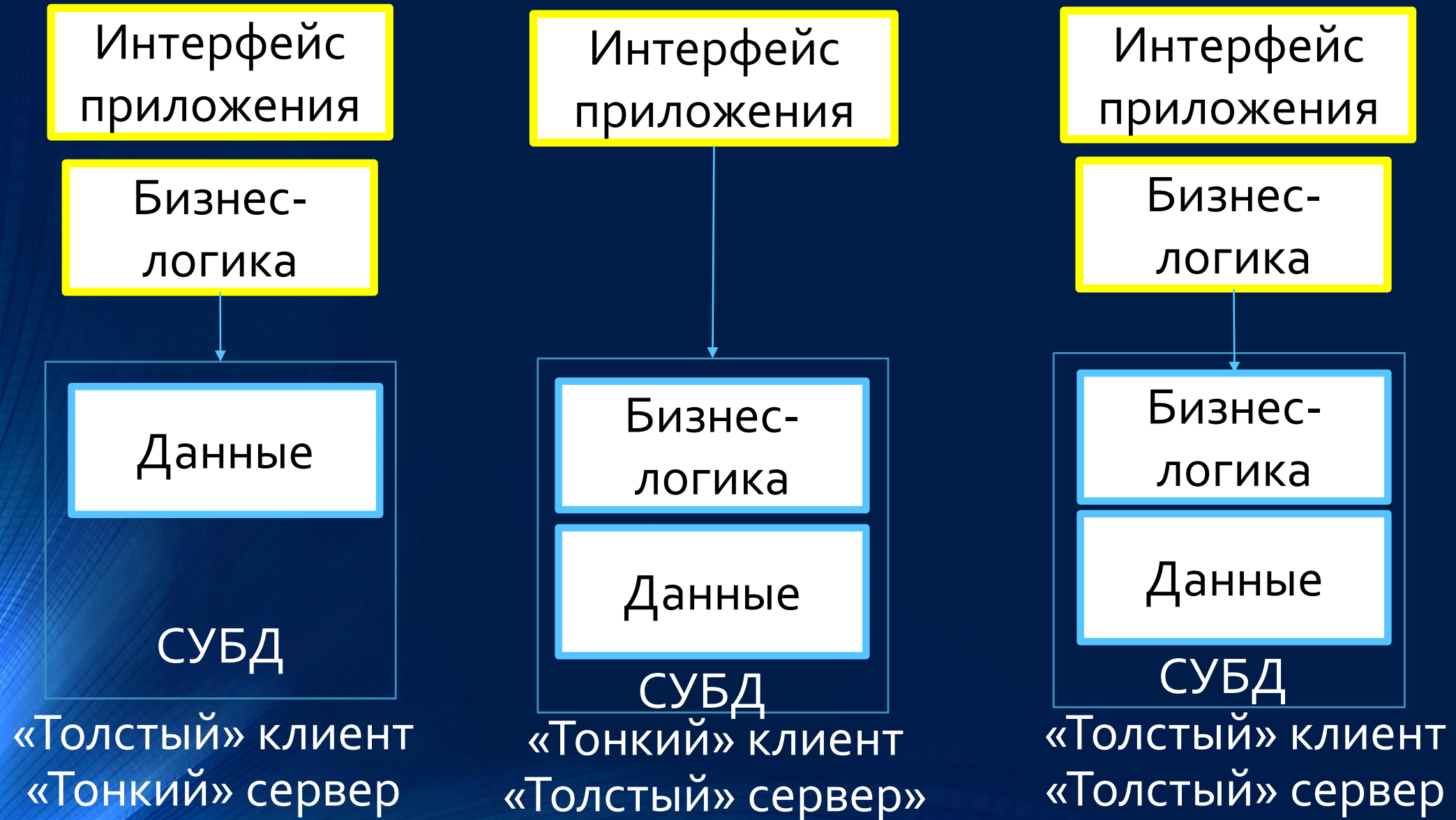
Серверная часть



## *2-уровневая архитектура клиент-сервер*

- + безопасность;
- + централизованный доступ к данным;
- + низкие требования к производительности и техническим характеристикам клиентов.
- + - простота обслуживания;
- **связывание** данных и бизнес-логики на сервере;
- проблема **масштабирования** системы;
- **неработоспособность** сервера может сделать неработоспособной всю вычислительную сеть.

# Распределение функций в 2-уровневой архитектуре

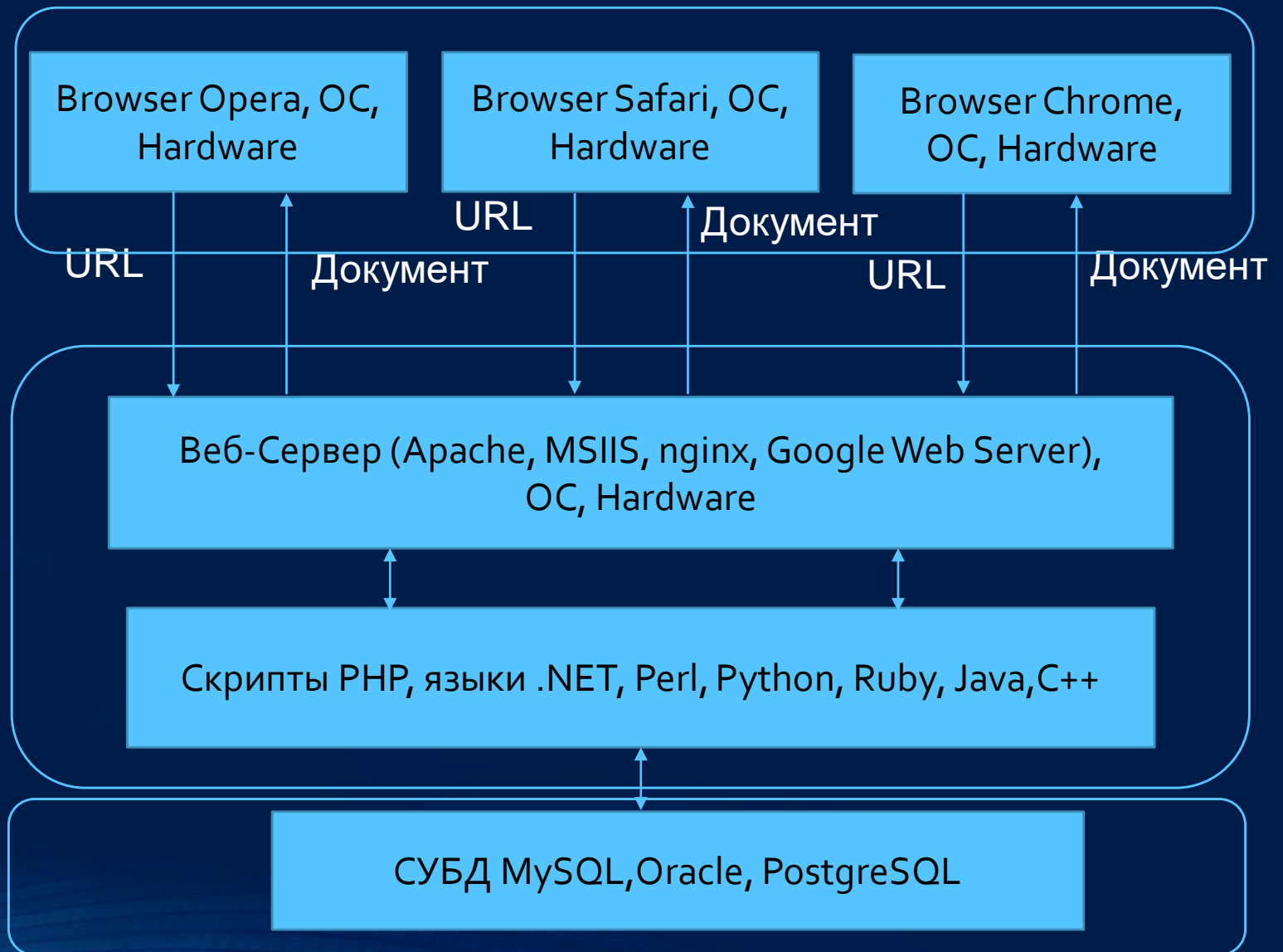


# Трехзвенная (N-уровневая) архитектура

- клиентское приложение («тонкий клиент»),

- сервер приложений,

- сервер базы данных.



# Трехзвенная (N-уровневая) архитектура

- + масштабируемость;
- + конфигурируемость – изолированность уровней друг от друга позволяет переконфигурировать систему при возникновении сбоев или при плановом обслуживании на одном из уровней;
- + низкие требования к скорости канала (сети) между клиентом и сервером приложений;
- растет сложность серверной части, более высокая сложность создания приложений;
- высокие требования к производительности серверов приложений и сервера базы данных;
- высокие требования к скорости канала (сети) между сервером базы данных и серверами приложений.



# Многозвенная архитектура

1. Представление;
2. Уровень представления;
3. Уровень логики;
4. Уровень данных;
5. Данные.



# Компоненты шаблона MVC

Шаблон MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

**Модель (Model)** – логика приложения, предоставляет данные (обычно для View), а также реагирует на запросы (обычно от контролера), изменяя свое состояние.

**Представление (View)** – отвечает за отображение информации (пользовательский интерфейс).

**Контролер (Controller)** – интерпретирует данные, введенные пользователем, и информирует модель и представление о необходимости соответствующей реакции.

# Архитектура MVC (Model View Controller)

Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида).

- к одной модели можно присоединить несколько представлений, при этом не затрагивая реализацию модели.
- не затрагивая реализацию представлений, **возможно изменить реакции** на действия пользователя (нажатие мышью на кнопке, ввод данных), для этого достаточно использовать другой контроллер.
- разделение труда разработчиков – программисты, занимающиеся разработкой бизнес-логики (модели), вообще могут не быть осведомлены о том, какое представление будет использоваться.



# Архитектура MVC (Model View Controller)



**Пассивная модель** — модель не имеет никаких способов воздействовать на представление или контроллер, и используется ими в качестве источника данных для отображения.

**Активная модель** — модель оповещает представление о том, что в ней произошли изменения, а представления, которые заинтересованы в оповещении, подписываются на эти сообщения.



# MVC и 3-звенная архитектура приложения

## Слой клиента

Самый верхний уровень приложения с интерфейсом пользователя. Главная функция интерфейса представление задач и результатов понятных пользователю.



## Слой логики

Этот слой координирует программу, обрабатывает команды, выполняет логические решения и вычисления, выполняет расчеты. она также перемещается и обрабатывает данные между двумя окружающими слоями.

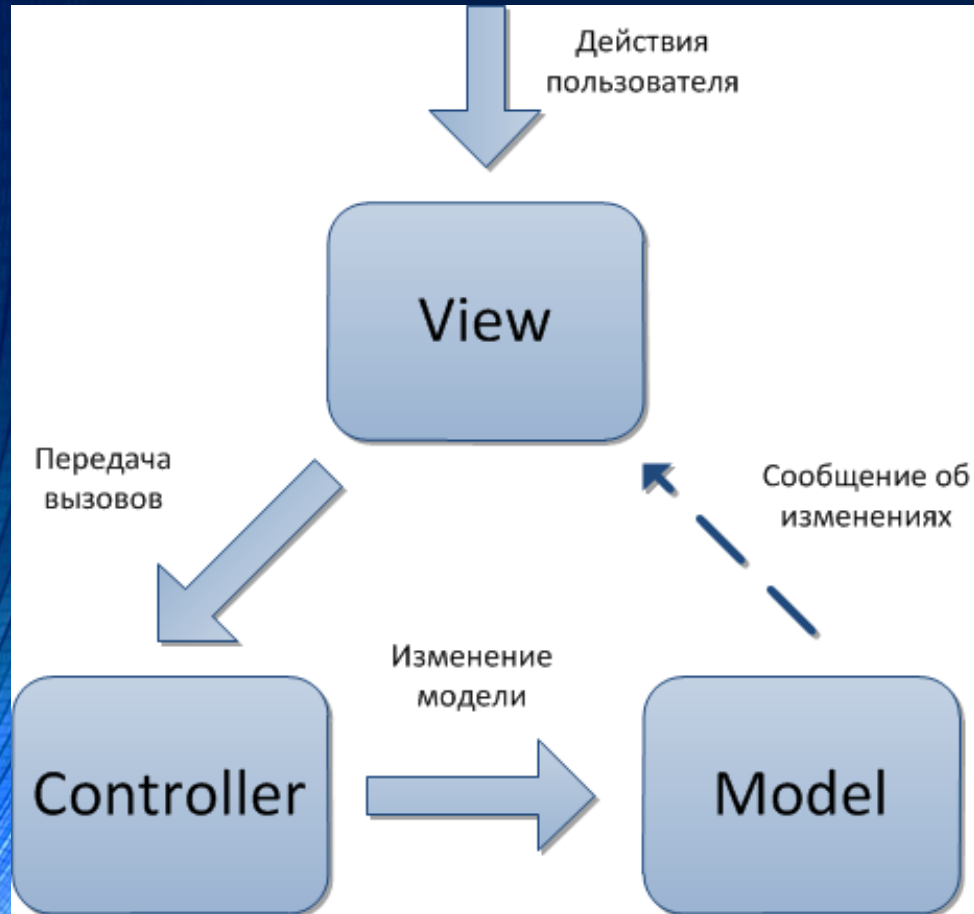


## Слой данных

Здесь храниться информация и извлекается из базы данных и файловой системе. Информация отправляется в логический слой для обработки, и в конечном счете возвращается пользователю.



# Разновидности MVC

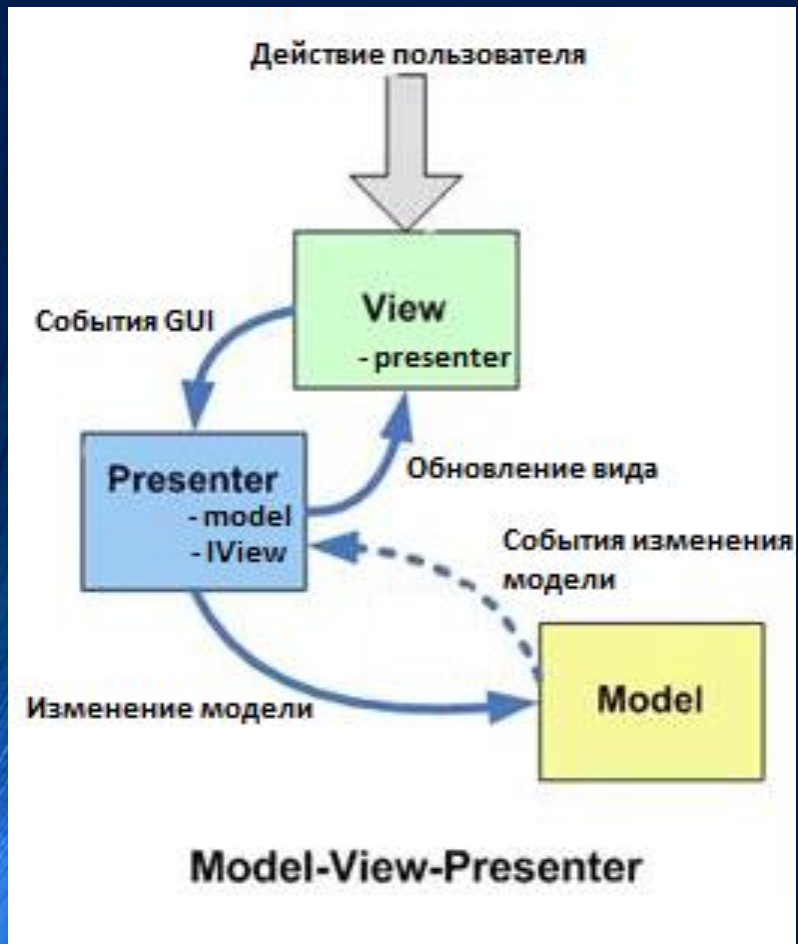


MVC с активной моделью

Наиболее распространенные виды MVC-паттерна:

- Model-View-Controller
- Model-View-Presenter
- Model-View-ViewModel
- MVPVM (MVP + MVVM)

# Разновидности MVC. Model-View-Presenter



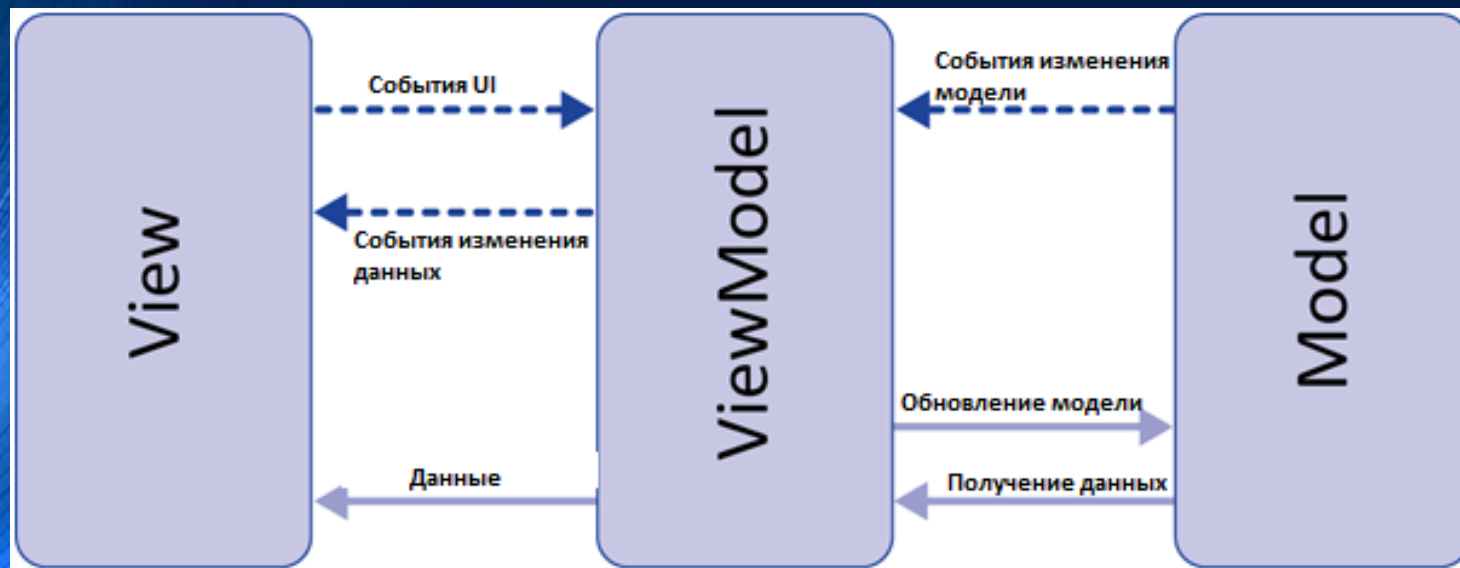
Признаки презентера:

- Двухсторонняя коммуникация с представлением;
- Представление взаимодействует напрямую с презентером, путем вызова соответствующих функций или событий экземпляра презентера;
- Презентер взаимодействует с View путем использования специального интерфейса, реализованного представлением;
- Один экземпляр презентера связан с **одним** отображением.

# Разновидности MVC. Model-View-View Model

Признаки View Model:

- Двухсторонняя коммуникация с представлением;
- View Model — это абстракция представления. Обычно означает, что свойства представления совпадают со свойствами View-модели / модели
- Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings)
- Один экземпляр View-модели связан с одним отображением.

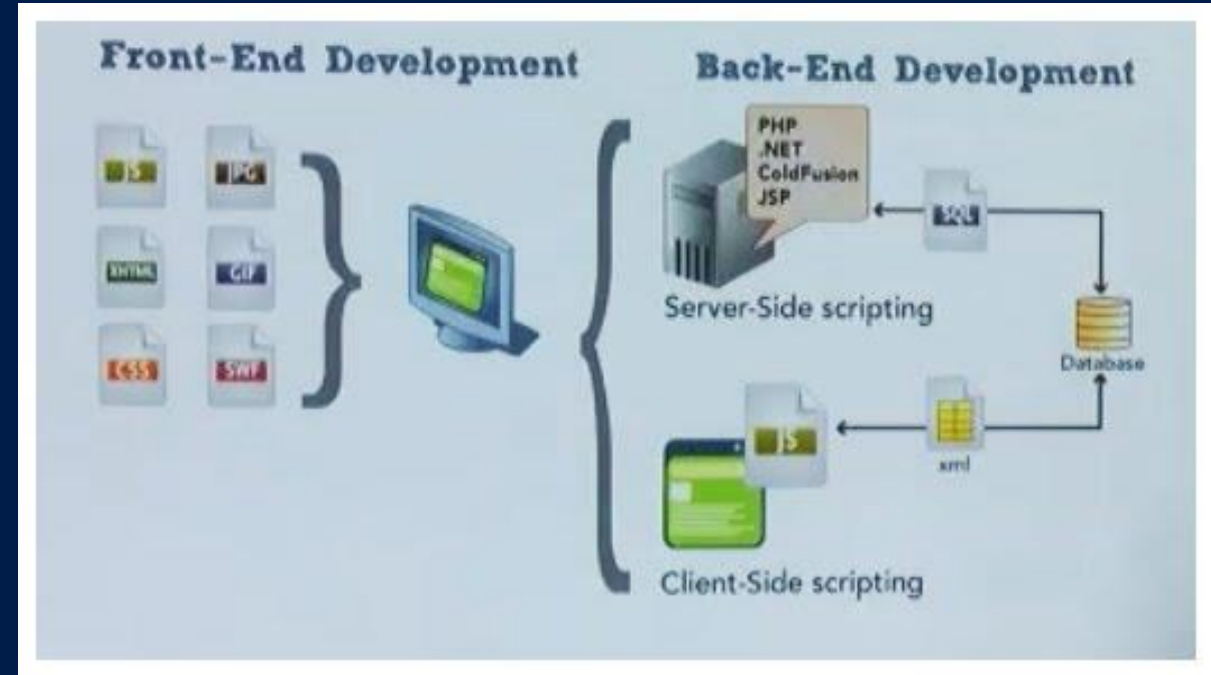


Пример использования:  
**WPF.**



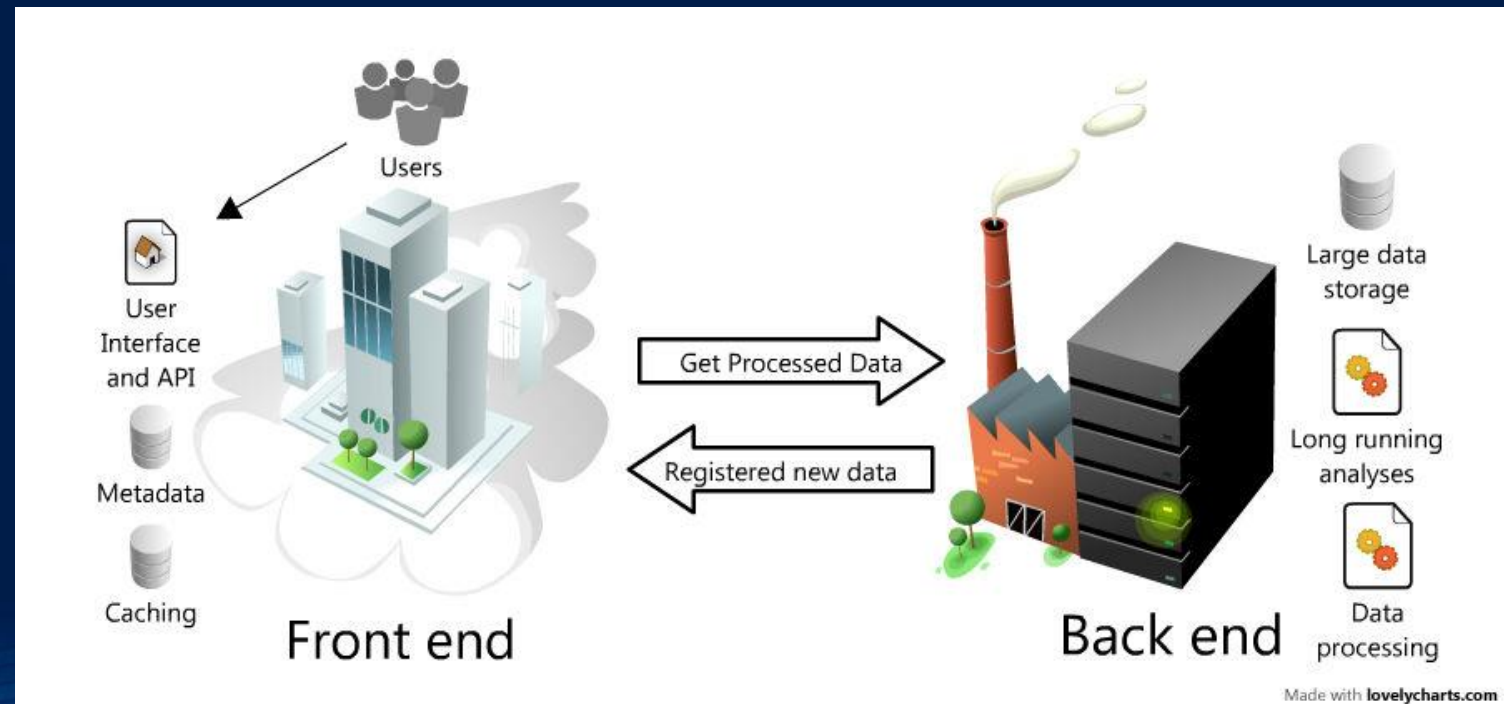
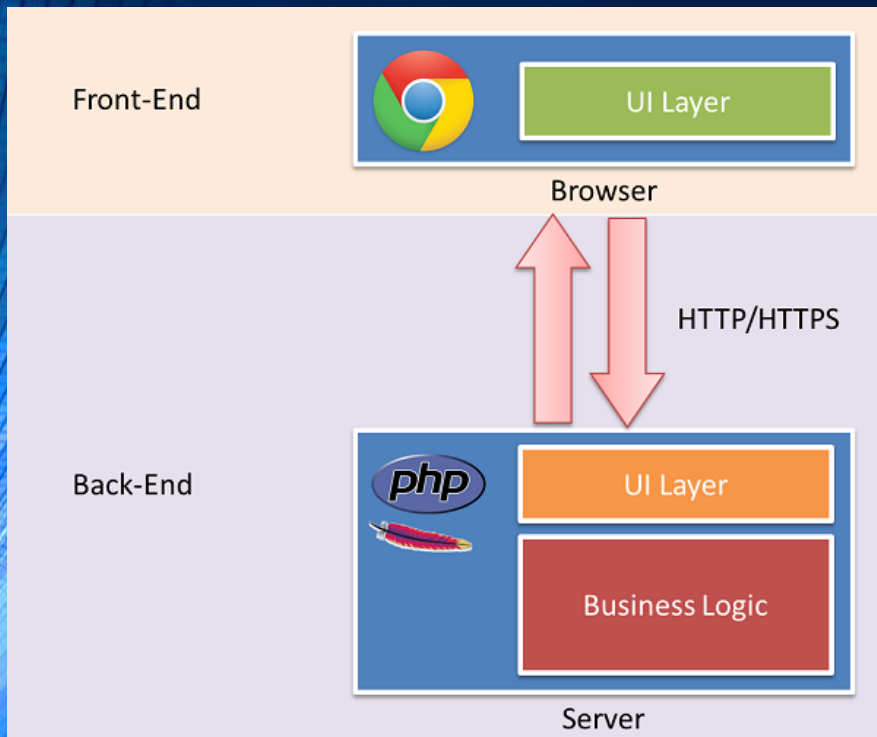
# Отделение логики от представления

- Отделение бизнес-логики от контента
- Отделение контента от дизайна
- Упрощение разработки
  - Разделение сложного целого на составные части
  - Стимулирует бóльшую степень абстракции и структуризации
- Разделение труда программиста и верстальщиков, дизайнеров.



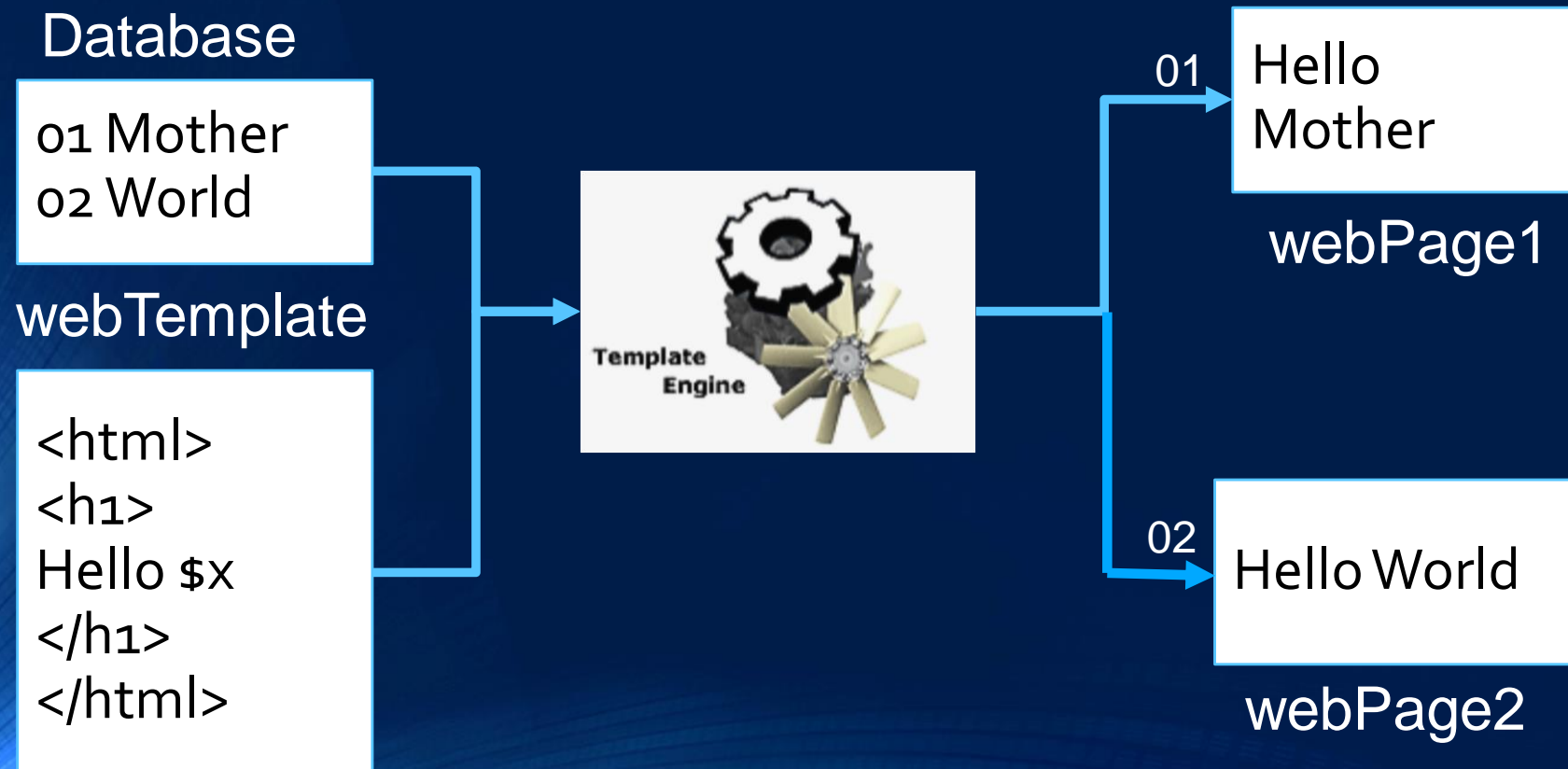
# Front-End и Back-End

- Front-End – публичная часть проекта, обеспечивающая прием запросов от пользователей, трансляцию запросов к Back-End и выдачу непосредственного содержимого пользователю.
- Back-End – исполнительная часть системы, которая обеспечивает выполнение PHP-скриптов, формирование контентных страниц, запросы к БД и работу бизнес-логики приложений.



# Шаблоны

HTML-теги и логика представления содержатся в шаблоне. Код приложения, который не содержит логики представления, осуществляет разбор запросов, выполняет необходимые операции, а затем передает неоформленные данные шаблону для их форматирования и отображения.



# Шаблоны. Классификация

- **Статические** – страница генерируется перед публикацией на сервере.
- **Серверные шаблоны** – функционирующие на стороне сервера (server-side) – страница генерируется сервером «на лету».
- **Клиентские шаблоны** – функционирующие на стороне клиента (client-side) – страница формируется на стороне клиента.

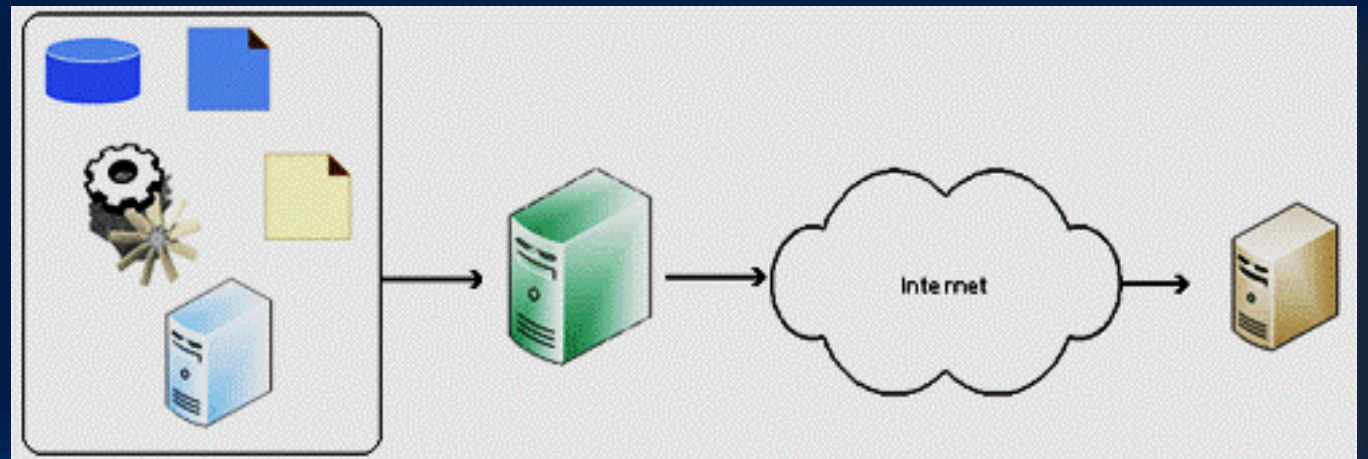




# Статические шаблоны

**Статический шаблон** – это полностью собранная страница, в которой оставлены **редактируемые поля**, индивидуально специфичные для каждой отдельной страницы.

Статические шаблоны предполагают **предварительную подготовку** HTML-страниц вне веб-сервера (например, используя редакторы Dreamweaver, FrontPage).



# Статические шаблоны

## **Преимущества:**

- Нет дополнительной нагрузки на сервер. *И на клиент тоже.*
- Легкость повторного использования.
- Простая переносимость.

## **Недостатки:**

- Нет автоматизации процесса изменения данных или дизайна.
- Сложность поддержки.

## Примеры использования:

- Сайт-визитка
- Домашние страницы пользователей

# Серверные шаблоны

Серверные шаблоны предполагают формирование HTML-страниц на сервере из результатов выполнения скриптов (SSI, PHP, PERL, ASP, ASP.NET, JSP) и шаблонов дизайна страницы.

**Server Side Includes** (SSI, Server-Side Includes, включения на стороне сервера) — инструмент для динамической сборки веб-страниц на сервере из отдельных составных частей. Позволяет реализовывать условные операции (if/else), работать с переменными и т.п. SSI-файлы, имеют расширение *.shtml*.

```
<!--#set var="Zed" value="{a}bc_{abc}" -->
```

Использование **шаблонизаторов** – Smarty, Twig и др.



# Серверные шаблоны. Преимущества

- Простота **разделения** уровня представления и уровня бизнес-логики приложения.
- Большое количество **готовых решений** в области серверных шаблонов.
- Простота **смены дизайна** (фактически, необходимо только менять дизайн шаблонов страниц, которых обычно существенно меньше чем самих страниц данных). **Скорость**.
- **Независимость** от клиентского ПО (версии и настроек браузера), т.к. к клиенту приходит готовый HTML-код, не требующий дальнейших преобразований.



# *Серверные шаблоны. Недостатки*

- Увеличение **нагрузки на сервер** (особенно при достаточно сложном дизайне шаблона).
- Требования к **каналу** передачи.
- Не учитывается специфика клиента – в качестве которого может выступать не только браузер, но и, например, мобильное приложение
- Относительная сложность составления шаблонов страниц.

# Клиентские шаблоны

Клиентские шаблоны предполагают заполнение шаблонов дизайна содержимым непосредственно на клиентской стороне.

Примеры применения шаблонов на стороне клиента:

- **AJAX** – подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером.
- **XML+XSLT** (язык преобразования XML-документов, в клиентских решениях может использоваться для приведения структуры данных из внутреннего формата сервера к некоторому внешнему формату, понятному клиенту). Может использоваться и на сервере.
- **jQuery Templates** (более не поддерживается).
- **JsRender, Dust** и другие JS-движки шаблонов.
- функционал создания шаблонов, в составе **фрэймворков**.

# Клиентские шаблоны

## *Преимущества:*

- Снижение дополнительной нагрузки на сервер (по сравнению с серверными шаблонами).
- Возможность гибко и быстро реагировать на действия пользователя.

## *Недостатки:*

- Сильная зависимость от версии и настроек браузера.
- Проблема индексации ресурса поисковыми роботами.
- Код шаблонов внутри кода html-страницы делает ее невалидной.
- Клиенту становится доступен шаблон.

# Классификация шаблонов по набору возможностей

- Простые – только подстановка переменных.

**Примеры:** команда `print` в Perl, `echo` в PHP "Hello \$x"

- С поддержкой итерации – возможности простых шаблонов + возможность повторения блоков с разными наборами данных.

**Примеры:** HTML::Template (Perl), phpBB-шаблоны

- Сложные – поддержка логики, параметризация блоков, циклы, выражения.

**Примеры:** XSL, Template::Toolkit (Perl), Smarty, Twig (PHP)



# Шаблонизаторы

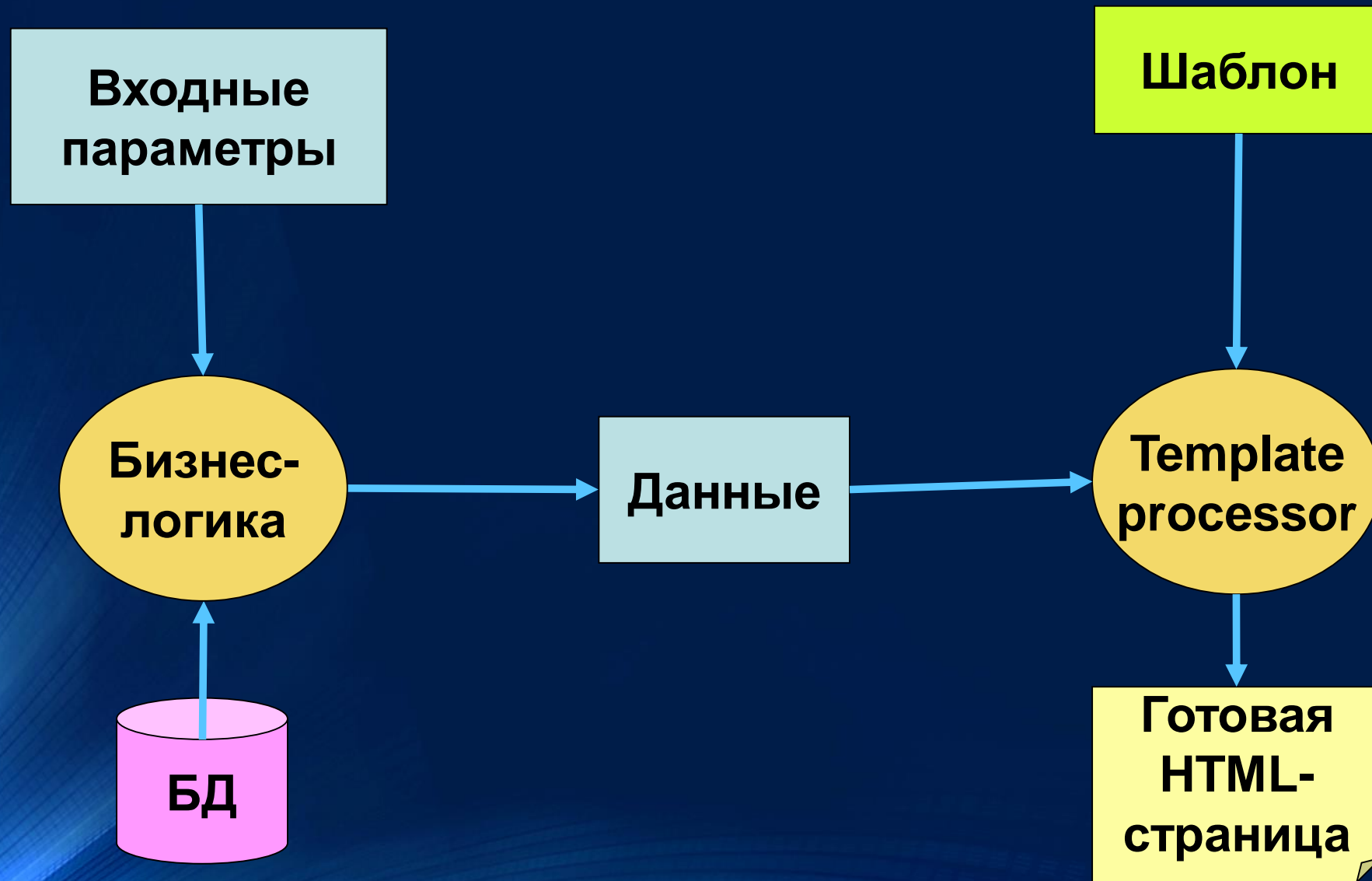
**Шаблонизатор** (в web) – это ПО, позволяющее использовать html-шаблоны для генерации конечных html-страниц.

Основная цель использования шаблонизаторов – это **отделение** представления данных от исполняемого кода.

Примеры: Blade, Mustache, Smarty, Twig, Volt.

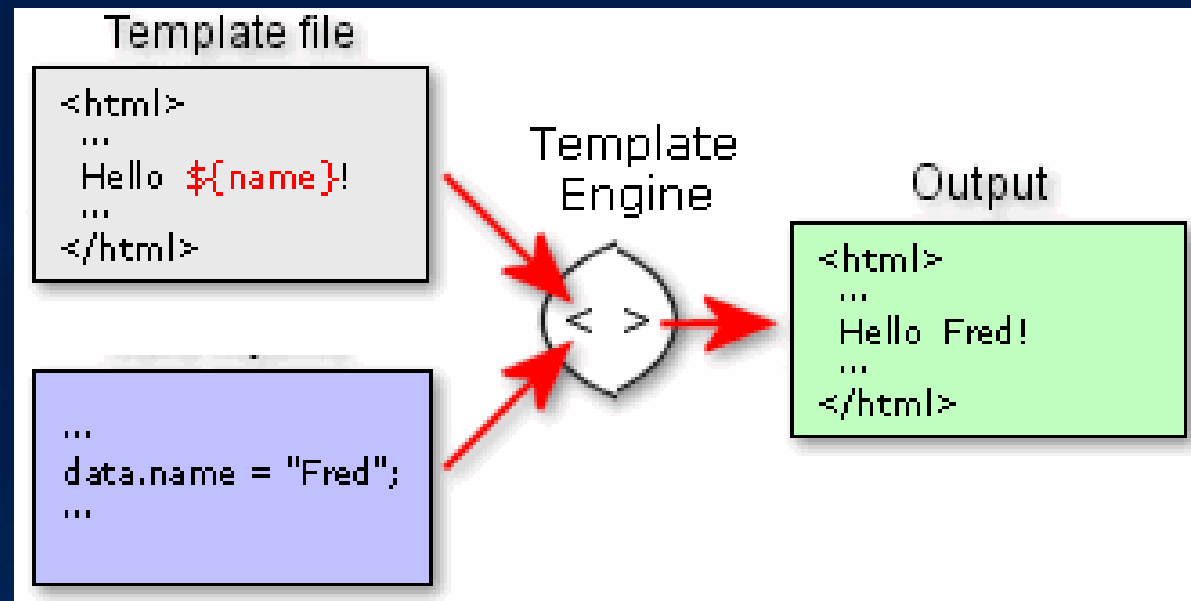
	Компиляция шаблона	Выполнение
Smarty 3.1.1	16.320 сек.	0.058 сек.
Twig 1.2.0	9.757 сек.	0.083 сек.

# Шаблонизаторы. Схема работы web-приложения



# Шаблонизаторы. Smarty

**Smarty** — компилирующий обработчик шаблонов для PHP, один из инструментов, позволяющих отделить прикладную логику и данные от представления в духе концепции Model-view-controller.



# *Smarty. Пример. Код шаблона, index.tpl*

```
<html>
<head>
<title>User Info</title>
</head>
<body>
User Information:<p>

Name: {$name}<br>
Address: {$address}<br>

</body>
</html>
```



# *Smarty. Пример. index.php*

```
include('Smarty.class.php');  
$smarty = new Smarty;  
  
// Присваиваем значения переменным шаблона  
// Обычно это результаты запросов к БД  
// или результаты вычисления бизнес-логики веб-приложения  
// В данном примере для простоты используются статические значения  
$smarty->assign('name', 'george smith');  
$smarty->assign('address', '45th & Harris');  
  
// вывод шаблона  
$smarty->display('index.tpl');
```

# *Smarty. Пример. Результат*

```
<html>
<head>
<title>User Info</title>
</head>
<body>
  User Information:<p>

  Name: george smith <br>
  Address: 45th & Harris <br>
</body>
</html>
```

# *Smarty. Пример. Код шаблона, index.tpl*

```
<thead>
  {if $query_result}
    <tr>
      <th>#</th>
      {foreach from=$result_list[o]
key=k item=v}
        <th>{$k}</th>
      {/foreach}
    </tr>
  {/if}
</thead>
```

```
<tbody>
  {if $query_result}
    {foreach from=$result_list key=i item=v}
      <tr><td>{$i}</td>
      {foreach from=$v key=k item=v}
        <td>{$v}</td>
      {/foreach}
    </tr>
  {/foreach}
  {/if}
</tbody>
```

# *Smarty. Отличительные особенности*

- **Высокая скорость**, эффективность за счет использования ресурсов PHP.
- Шаблоны **компилируются только один раз**. Перекомпилируются только те шаблоны, которые изменились.
- Можно создавать **пользовательские функции и модификаторы**.
- **Настраиваемые разделители тегов шаблона**, то есть вы можете использовать {}, {}, и т. д. В третьей версии – автоматическое **игнорирование** фигурных скобок "{", "}", если они окружены пробелами (больше не требуется окружать Javascript {literal}}/literal} или использовать другой маркер тега).
- Допустимо **неограниченное вложение секций, условий** и т. д.
- Существует **возможность включения PHP-кода** прямо в шаблон.
- Встроенный механизм **кеширования**, пользовательские функции кеширования. В третьей версии - управление кешированием на уровне элементов:  
`{foo nocache}` – не кешировать содержимое этой переменной  
`{include file="foo.tpl" nocache}` – не кешировать содержимое включаемого файла
- **Компонентная архитектура**. Возможность наследования шаблонов (с 3 версии).



# Подходы к разработке web-приложений

- PHP + стандартные и дополнительные библиотеки.
- **CMS** (content management system) – система управления содержимым, предоставляющая инструменты для добавления, редактирования, удаления информации на сайте.
- Использовать **фрэймворк**, определяющий архитектуру (взаимосвязи между компонентами) вашего веб-приложения и расширяющий функциональность. Фреймворк отличается от **библиотеки** тем, что библиотека может быть использована в программном продукте просто как набор подпрограмм близкой функциональности, **не влияя на архитектуру** программного продукта и не накладывая на неё никаких ограничений.

# CMS

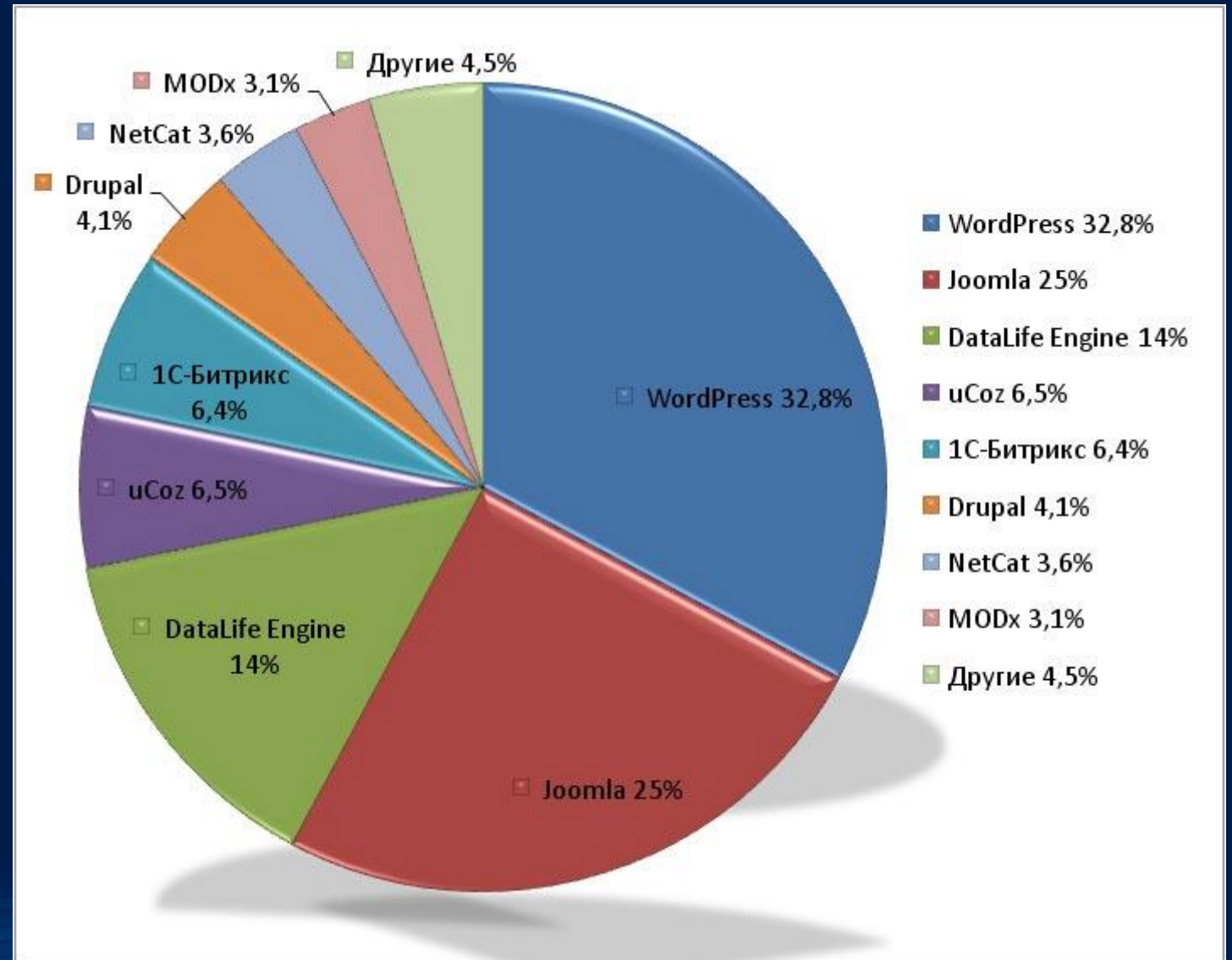
Система управления содержимым (контентом) (англ. Content management system, CMS) — информационная система, используемая для обеспечения и организации совместного процесса создания, редактирования и управления **содержимым** веб-ресурса.

Основные функции CMS:

- Предоставление инструментов для **создания** содержимого, организация совместной работы над содержимым;
- **Управление** содержимым: хранение, контроль версий, соблюдение режима доступа;
- **Публикация** содержимого, представление информации в виде, удобном для навигации, поиска.

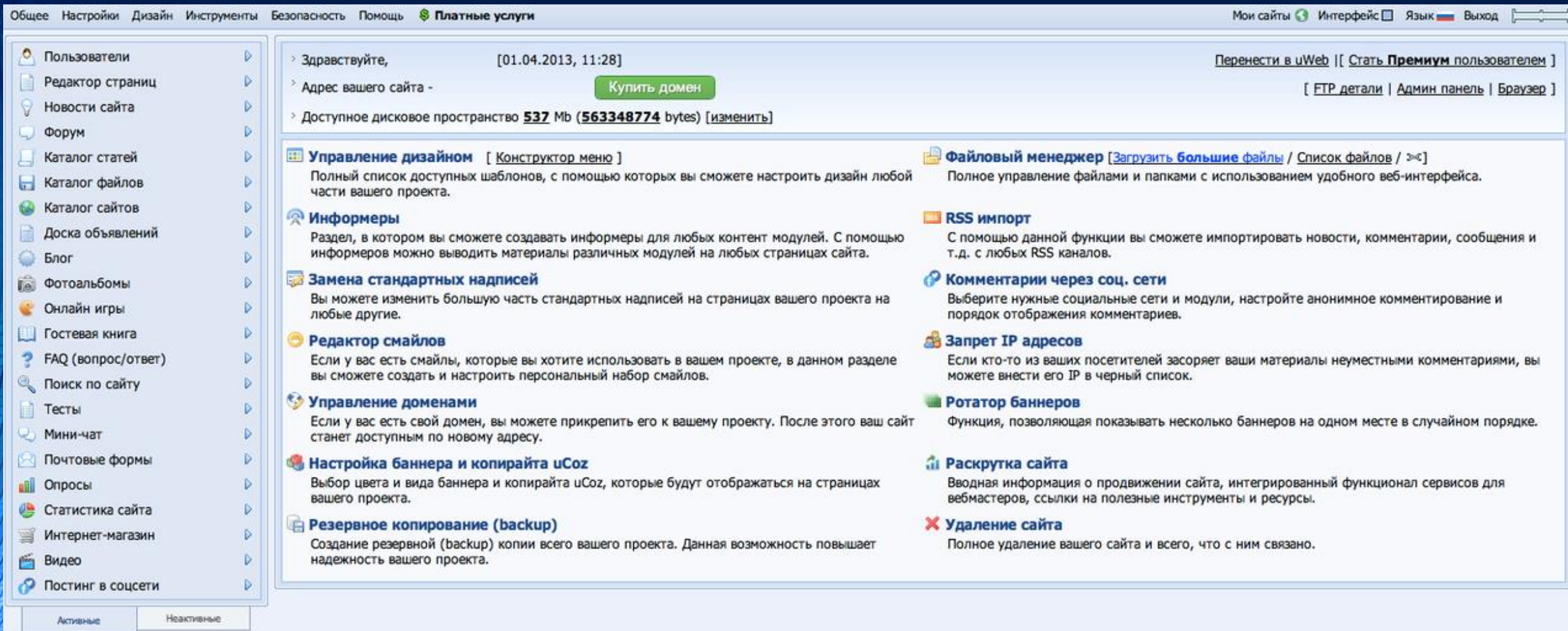
# CMS

- WordPress (PHP, Javascript)
- Joomla (PHP, Javascript)
- Drupal (PHP)
- Magento (PHP)
- MediaWiki (PHP)
- DotNetNuke
- uCoz







# CMS. Пример панели управления





# CMS. Пример панели управления

 Flowers Shop 1 + Добавить

Привет, liza 

Настройки экрана ▾Помощь ▾

Консоль

Главная  
Обновления

Записи  
Медиафайлы  
Страницы  
Комментарии 1  
WooCommerce  
Товары  
Внешний вид  
Плагины  
Пользователи  
Инструменты  
Настройки  
Meta Slider  
Свернуть меню

Добро пожаловать в WordPress!  
Мы собрали несколько ссылок для вашего удобства:

Заккрыть

Для начала

Настройте свой сайт

или выберите другую тему

Следующие шаги

Отредактируйте главную страницу

+ Добавьте другие страницы

Просмотрите свой сайт

Другие действия

Настройте виджеты и меню

Включите или выключите комментарии

Узнайте больше о работе с WordPress

На виду ▲

2 записи4 страницы1 комментарий1 на проверке

WordPress 4.5 с темой оформления Storefront.

Активность ▲

Недавно опубликованы

17.04.2016, 19:26 (без названия)

15.04.2016, 22:40 Привет, мир!

Быстрый черновик ▲

Заголовок

О чём хотите написать?

Сохранить

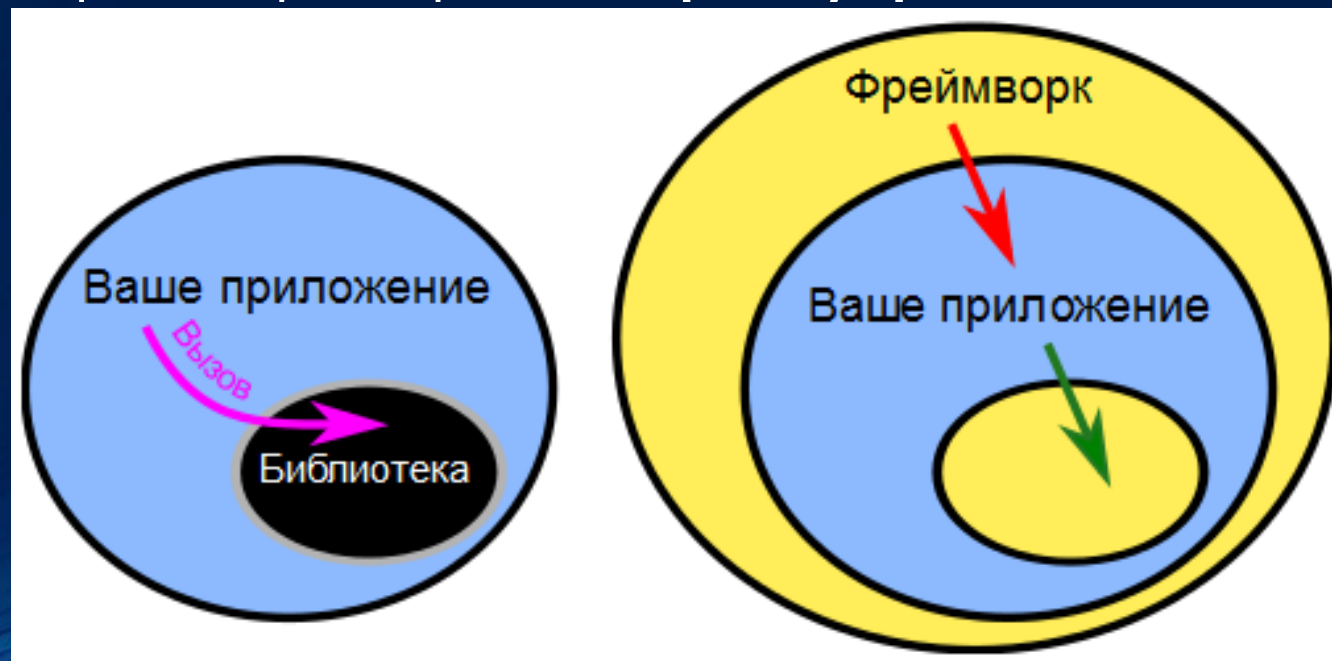
Новости WordPress ▲

# Применение фреймворков. Инверсия управления

Фреймворк (Web application framework) — это каркас, предназначенный для создания динамических вебсайтов, сетевых приложений или сервисов с заданной **архитектурой**.

Применение фреймворков упрощает веб-разработку и уменьшает дублирование кода. Многие каркасы упрощают доступ к базам данных, разработку интерфейса, обмен данными с сервером и т.д.

Для фреймворков характерна **инверсия управления**:



# Фреймворк. Функции

Обычно фреймворки обеспечивают следующий функционал:

- Определение базовой архитектуры приложения (MVC и т.п.) ;
- Взаимодействие с web-сервером (разбор параметров запроса, Cookie, работа с сессией и т.п.);
- Взаимодействие с БД, объектная модель базы (ORM);
- Обработка AJAX-запросов (выдача структур данных JSON и т.п.);
- Кэширование;
- Обработка ошибок;
- Журнализация;
- Обработка **шаблонов**.

# Фреймворки. Примеры

JavaScript:

- Backbone.js, AngularJS, Knockout.js.

PHP:

- CakePHP, Symfony, Zend Framework, Yii, Laravel.

Python:

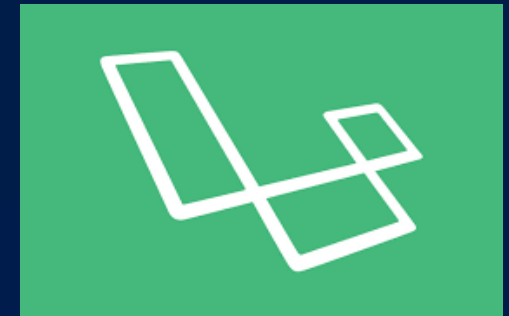
- Zope, Django, TurboGears.

Ruby:

- Ruby On Rails, Nitro.

Java:

- Spring, GRails, Struts.





# Фреймворки. Особенности использования

- Значительное **сокращение времени разработки**:
  - все взаимосвязи уже выстроены и отлажены. **Зависимость от заданной архитектуры.**
  - разработчик следует чётким правилам, не нужно тратить время на обдумывание архитектуры и принципов функционирования приложения
  - прирост скорости особенно заметен в больших проектах.
- Лучшая функциональность «из коробки». **Для простых проектов – избыточна, для сложных – наверняка чего-то не хватит.**
- Выше надёжность и безопасность. **Малый «автобусный фактор»?**
- Лишние слои абстракции.
- Громоздкое конфигурирование.

# Серверный JavaScript-фреймворк Node.js

**Node** или **Node.js** – программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код). Node.js применяется преимущественно на сервере, исполняя роль **веб-сервера**, но есть возможность разрабатывать на Node.js и **десктопные** оконные приложения (при помощи Node-webkit (NW.js)).

В основе Node.js лежит **событийно-ориентированное** и **асинхронное** программирование с неблокирующим вводом/выводом.

## Преимущества:

- масштабируемость
- единая языковая среда как для client-side, так и для server-side разработчиков
- Кроссплатформенность (Win, Linux, Mac)
- Использование самой быстрой на текущий момент JavaScript VM (**Google V8**)
- Подробная документация.

# MEAN

Стек технологий MEAN отражает современный подход к веб-разработке: когда на каждом уровне приложения, от клиента до сервера применяется один и тот же язык (JavaScript).

**MEAN** – комплекс технологий, включающий в себя:

- **MongoDB** – документо-ориентированное NoSQL-хранилище (JSON-подобная схема данных).
- **Express.js** – веб-фреймворк для Node.js.
- **AngularJS** – JavaScript MVC-фрэймворк.
- **Node.JS**

# Node.JS. Пример приложения

Исходный код файла example.js:

```
console.log("Hello World"); // вывод в консоль
// запуск сервера, слушающего порт 9090
var http = require('http');
var port = 9090;
http.createServer(responseHandler).listen(port);
/*var server = http.createServer().listen(port);
server.on('request', responseHandler);*/
console.log('Server running at http://127.0.0.1:' + port + '/');

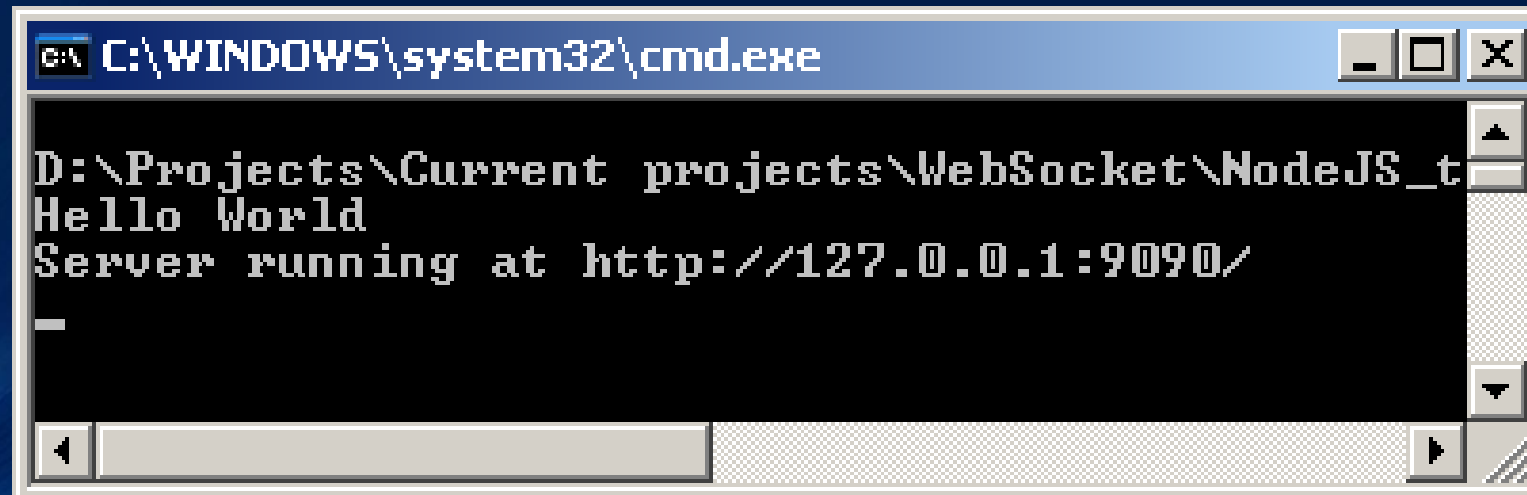
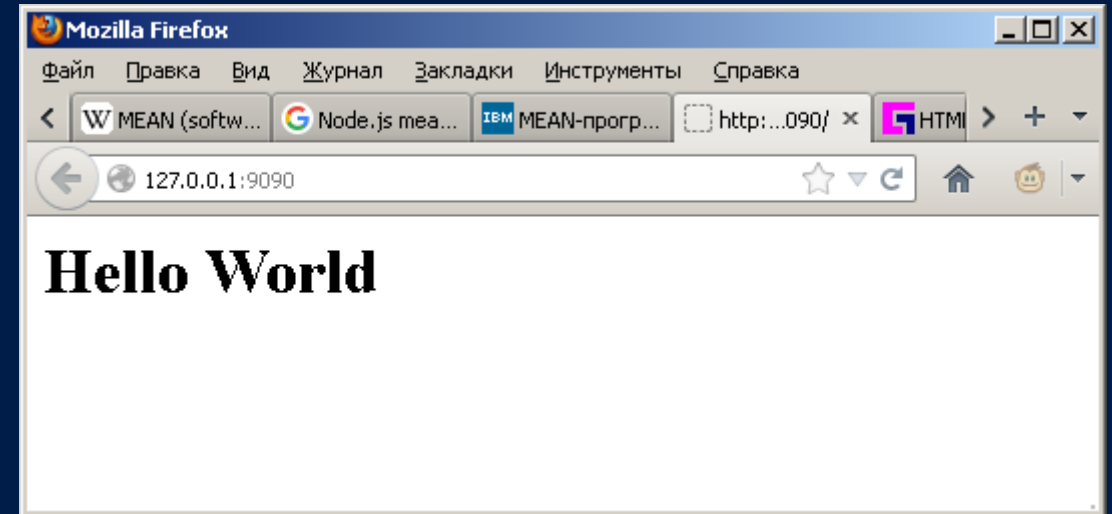
// создание страницы в ответ на запрос
function responseHandler(request, response){
    response.writeHead(200, {'Content-Type': 'text/html'});
    response.end('<html><body><h1>Hello World</h1></body></html>');
}
```



# Node.JS. Пример приложения

Запуск сервера:  
`node example.js`

Адрес приложения:  
`http://localhost:9090/`



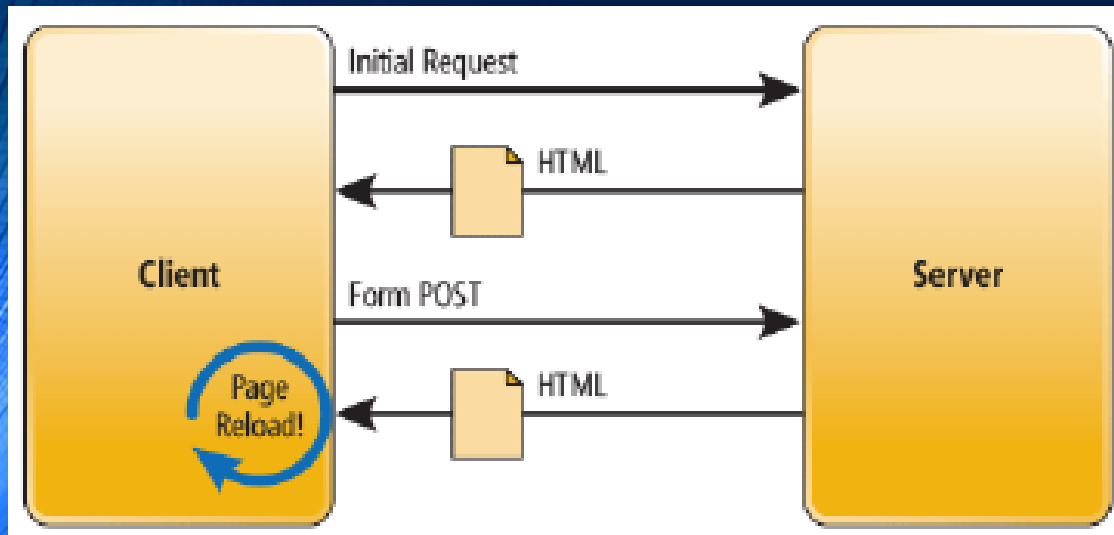
# Пакеты для разработки с помощью Node.JS

- **npm** – Node Packaged Modules, менеджер пакетов для node. Устанавливается вместе с node. **npm install express mongoose jade less**
- node-inspector и nodemon – отладка и авторестарт разрабатываемых приложений.
- node-validator – библиотека для проверки, фильтрация и санитизации строк.
- bcrypt – библиотека для хеширования паролей.
- mongoose – mongodb для node.
- node\_redis – клиент для Redis для node.
- Jade – шаблонизатор для node.
- Nodemailer – модуль для отправки электронной почты с помощью node.
- express – node-фреймворк для построения одно- и многостраничных веб приложений.
- **socket.io** – унифицированное средство обмена данными.

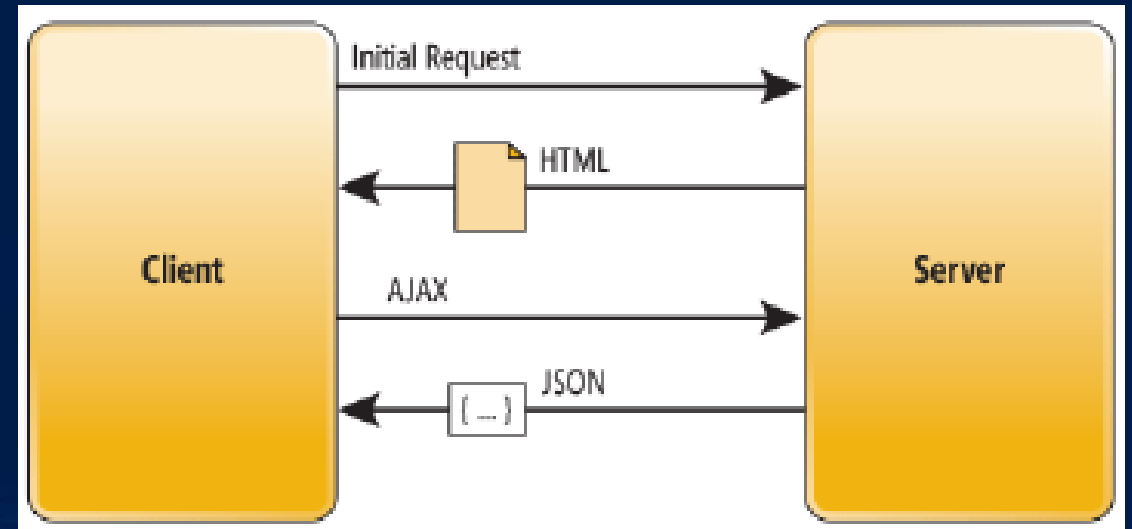
# SPA

Одностраничное приложение (англ. single page application, SPA) – это веб-приложение или веб-сайт, использующий единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript, обычно посредством AJAX.

Традиционное «Round-Trip» приложение:



SPA:



# SPA. Особенности

- богатый пользовательский интерфейс;
- меньше нагрузка на сервер, обращение к серверу только за данными;
- проще хранить информацию о сеансе;
- нет переключения страниц, сокращена загрузка одного и того же контента;
- лучше интерактивность, возможность работы offline;
- загружает **все скрипты**, требующиеся для старта приложения при инициализации web-страницы; нет возможности скрыть логику;
- необходимость обеспечения **клиентской навигации**.

Существует большое количество библиотек и фреймворков, обеспечивающих базовые принципы SPA, минимизируя трудозатраты на решение универсальных задач при разработке Single Page Application:





# AngularJS

AngularJS – JavaScript-фреймворк с открытым исходным кодом, использующий шаблон MVC (**MVW**). Фреймворк подходит для разработки одностраничных приложений.

Поддерживает двустороннее связывание данных – изменения *Модели* передаются в *Представление*, а изменения *Представления* автоматически отражаются в *Модели*. Таким образом, Модель становится актуальным источником данных о состоянии приложения.



# AngularJS. Пример

```
<!doctype html>
<html ng-app>
<head>
  <meta charset="utf-8">
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs
/1.3.11/angular.min.js"> </script>
</head>
<body>
  <label> Введите имя: </label>
  <input type="text" ng-model="name" placeholder="Введите имя">
  <h1> Добро пожаловать {{name}}!</h1>
</body>
</html>
```

# AngularJS. Основные директивы

- **ng-app** – главная директива приложения, задает область приложения или создает область для модуля;
- **ng-controller** – связывание элемента с **контроллером**;
- **ng-model** – задает **модель** для связывания;
- **ng-change** – вызывает определенную функцию, при изменении значения;
- **ng-click** (**ng-dblclick**) – обработчик клика (двойного клика) по элементу;
- **ng-submit** – позволяет определить действие, которое будет выполняться при отправке данных из формы;
- **ng-pattern** – используется при валидации с помощью регулярных выражений;
- **ng-repeat** – цикл перебора массива:
  - **\$index** – номер текущей итерации
  - **\$first** – первая итерация
  - **\$last** – последняя итерация

# *AngularJS. MVC-пример. Модель.*

```
var model = {  
  students: [  
    {name: "Ivanov", Department: "KIU" },  
    {name: "Petrov", Department: "PI" },  
    {name: "Sidorov", Department: "KIU" },  
  ]  
}
```



# *AngularJS. MVC-пример. Контроллер.*

```
var studApp = angular.module("studApp", []);
studApp.controller("studController", function ($scope) {
    $scope.list = model;
    $scope.addItem = function (name, dep) {
        if(name != "" && dep != "") {
            $scope.list.students.push({ surname: name, department: dep });
        }
    }
});
```

# AngularJS. MVC-пример. Представление.

```
<html ng-app="studApp">
```

```
...
```

```
<body ng-controller="studController">
```

```
  <input type="text" ng-model="surname" placeholder="Фамилия" />
```

```
  <input type="text" ng-model="department" placeholder="Факультет" />
```

```
  <button ng-click="addItem(surname, department)">Добавить студента</button>
```

```
  <table>
```

```
    <tr><th>Surname</th><th>Department</th></tr>
```

```
    <tr ng-repeat="stud in list.students">
```

```
      <td>{{stud.surname}}</td>
```

```
      <td>{{stud.department}}</td>
```

```
    </tr>
```

```
  </table>
```

```
</body>
```

Васильев	
КИУ	
Добавить студента	
<b>Surname</b>	<b>Department</b>
Ivanov	KIU
Petrov	PI
Sidorov	KIU
Васильев	КИУ

# AngularJS. Пример создания директивы.

```
<body>  
  <my-directive></my-directive>
```

Hello World!

```
</body>
```

```
<script>
```

```
  angular.module("studApp", []).directive("myDirective", function() {
```

```
    return {
```

```
      restrict: 'E', //EACM'
```

```
      template: '<p>Hello {{ name }}!</p>',
```

```
      link: function(scope, el, attr) {
```

```
        scope.name = "World";
```

```
      }
```

```
    }
```

```
  });
```

```
</script>
```

# AngularJS. Основные сервисы

Сервисы AngularJS – специальные объекты или функции, которые выполняют некоторые общие для всего приложения задачи.

- `$cacheFactory` – работа с memory cache
- `$compile` – компиляция темплейтов и их связка со scope
- `$document`, `$window` – обертки над соответствующими объектами браузера
- `$http` – коммуникация с удаленным сервером посредством XMLHttpRequest или JSONP
- `$injector` – контейнер зависимостей
- `$interval`, `$timeout` – обертки над `setInterval()` и `setTimeout()` функциями объекта `window`
- `$location` – работа со строкой адреса браузера
- `$q` – помогает работать с асинхронными функциями



# AngularJS

Файл	Модуль	Для чего нужен
angular.js	ng	Ядро angular + множество компонент для обеспечения базовой функциональности
angular-animate.js	ngAnimate	Обеспечивает поддержку для JavaScript, CSS3 transition и CSS3 keyframe анимаций
angular-cookies.js	ngCookies	Компоненты для работы с cookies браузера
angular-resource.js	ngResource	Сервис для упрощенной работы с RESTful сервисами
angular-route.js	ngRoute	Компоненты для реализации роутинга в приложении
angular-touch.js	ngTouch	Компоненты для работы с сенсорными экранами

# Вопросы

- Какие существуют способы построения архитектуры веб-приложения?
- Какие функции выполняют элементы архитектуры Model-View-Controller?
- Преимущества подхода MVC. Разновидности MVC.
- В чем преимущества отделения логики от представления?
- Что такое Front-End и Back-End?
- Какие вы знаете типы шаблонов и чем они отличаются?
- Что такое Smarty? Каковы его возможности?
- В чем недостатки использования CMS?
- Особенности применения фреймворков?
- Что такое Node.js? Какие технологии входят в стек MEAN?
- Чем отличаются SPA от традиционных веб-приложений?
- Какие директивы AngularJS вам известны?