**HTTP, Frontend**

1. Протокол HTTP. Устройство HTTP-запроса и HTTP-ответа. Виды и примеры заголовков HTTP-запросов и ответов. Статусы ответов HTTP (приведите примеры их применения).
2. Теговая модель HTML. Теговая модель HTML: теги, атрибуты. Сходства и различия с XML. Основные теги HTML.
3. Как происходит отрисовка страниц в браузере.
4. Технология CSS. Что такое технология CSS и для чего она используется. Что такое CSS-селекторы, приведите примеры использования. Класс-селекторы, id-селекторы. Псевдоклассы. Блочные элементы, inline-элементы.

CSS (Cascading Style Sheets — каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML, но может также применяться к любым XML-документам, например, к SVG или XUL.

CSS используется создателями веб-страниц для задания цветов, шрифтов, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось разделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS). Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом. Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печатное представление, чтение голосом

**Селектор** - это формальное описание того элемента или группы элементов, к которым применяется указанное правило стиля.

Селектор, расположенный в левой части правила, до знака «{», определяет, на какие части документа (возможно, специально обозанченные) распространяется правило. Блок объявлений располагается в правой части правила. Он помещается в фигурные скобки, и, в свою очередь, состоит из одного или более объявлений, разделённых знаком «;». Каждое объявление представляет собой сочетание свойства CSS и значения, разделённых знаком ": ". Селекторы могут группироваться в одной строке через запятую. В таком случае свойство применяется к каждому из них.

**селектор**, **селектор** {

свойство: значение;

свойство: значение;

свойство: значение;

}

[**Селекторы по классу**](https://developer.mozilla.org/en-US/docs/Web/CSS/Class_selectors)

Этот базовый селектор выбирает элементы, основываясь на значении их атрибута class.  
Синтаксис: .имяКласса  
Пример: селектор .index выберет все элементы с соответствующим классом (который был определен в атрибуте class="index" или похожем).

**Селекторы по идентификатору**

Этот базовый селектор выбирает элементы, основываясь на значении их id атрибута. Не забывайте, что идентификатор должен быть уникальным, т. е. использоваться только для одного элемента в HTML-документе.

Синтаксис: #имяИдентификатора

Пример: селектор #toc выберет элемент с идентификатором toc (который был определен в атрибуте id="toc" или похожем).

**Псевдоклассы** позволяют выбирать элементы, основываясь на информации, которой нет в дереве элементов, например, состояние или другие данные, которые сложно вычленить другим способом. Например, они могут выбрать ссылку, которая была или не была ранее посещена.

Псевдоклассы определяют динамическое состояние элементов, которое изменяется с помощью действий пользователя, а также положение в дереве документа. Примером такого состояния служит текстовая ссылка, которая меняет свой цвет при наведении на неё курсора мыши. При использовании псевдоклассов браузер не перегружает текущий документ, поэтому с помощью псевдоклассов можно получить разные динамические эффекты на странице.

Синтаксис применения псевдоклассов следующий.

Селектор:Псевдокласс { Описание правил стиля

Вначале указывается селектор, к которому добавляется псевдокласс, затем следует двоеточие, после которого идёт имя псевдокласса. Допускается применять псевдоклассы к именам идентификаторов или классов (A.menu:hover {color: green}), а также к контекстным селекторам (.menu A:hover {background: #fc0}). Если псевдокласс указывается без селектора впереди (:hover), то он будет применяться ко всем элементам документа.

Условно все псевдоклассы делятся на три группы:

определяющие состояние элементов;

имеющие отношение к дереву элементов;

указывающие язык текста.

**:hover**

Псевдокласс :hover активизируется, когда курсор мыши находится в пределах элемента, но щелчка по нему не происходит.

**:visited**

Данный псевдокласс применяется к посещённым ссылкам. Обычно такая ссылка меняет свой цвет по умолчанию на фиолетовый, но с помощью стилей цвет и другие параметры можно задать самостоятельн

**Блочным называется элемент**, который отображается на веб-странице в виде прямоугольника. Такой элемент занимает всю доступную ширину, высота элемента определяется его содержимым, и он всегда начинается с новой строки. К блочным элементам относятся контейнеры <div>, <h1>, <p> и др.

Допускается вкладывать один блочный элемент внутрь другого, а также размещать внутри них встроенные элементы (<span>, например). Запрещено добавлять внутрь встроенных элементов блочные.

**Встроенные (строчные) элементы** генерируют внутристрочные контейнеры. Они не формируют новые блоки контента. Значения свойства display, такие как inline и inline-table делают элементы строчными. a, b, label, map, time, img, span.

Строчные элементы являются потомками блочных элементов. Они игнорируют верхние и нижние margin и padding, но если для элемента задан фон, он будет распространяться на верхний и нижний padding, заходя на соседние строки текста.

Ширина и высота строчного элемента зависит только от его содержимого, задать размеры с помощью CSS нельзя. Можно увеличить расстояние между соседними элементами по горизонтали с помощью горизонтальных полей и отступов.

Для того чтобы верхние и нижние поля и отступы работали для строчного элемента, нужно использовать конструкцию {display: inline-block}. Элемент останется встроенным, но к нему можно будет полноценно применить поля, отступы, задать ширину и высоту.

1. Основные стили в CSS.

Размеры (width, height) - https://www.w3schools.com/css/css\_dimension.asp

Отступы (margin, padding) - https://www.w3schools.com/css/css\_boxmodel.asp

Граница (border) - http://htmlbook.ru/css/border

Цвета (background, color)

Шрифт (font-family, font-size)

Display (тип отображения) - http://htmlbook.ru/css/display

Позиционирование элемента (position) - <http://htmlbook.ru/css/position>

1. Способы стилизации элементов на странице. Специфичность элементов. Позиционирование элементов.

**Стилизация**

**Атрибут style**

<h1 style="color:blue;margin-left:30px;">This is a heading</h1>

**Тег style**

<head>  
<link rel="stylesheet" type="text/css" href="mystyle.css">  
<style>  
h1 {  
  color: orange;  
}  
</style>  
</head>

**CSS файл**

**Указываем в HTML ссылку на CSS файл.**

<head>  
<link rel="stylesheet" type="text/css" href="mystyle.css">  
</head>

**Специфичность**

Типы селекторов

В следующем списке типы селекторов расположены по возрастанию специфичности:

селекторы типов элементов (например, h1) и псевдоэлементов (например, ::before).

селекторы классов (например, .example), селекторы атрибутов (например, [type="radio"]) и псевдокласов (например, :hover).

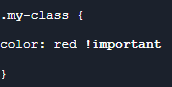
селекторы идентификаторов (например, #example).

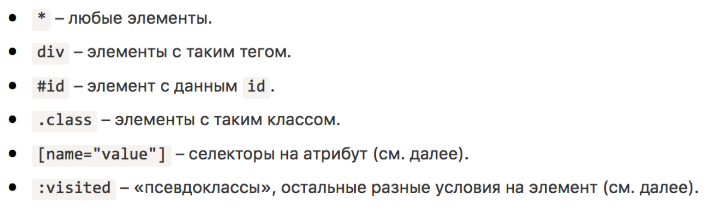
Универсальный селектор (\*), комбинаторы (+, >, ~, ' ') и отрицающий псевдокласс (:not()) не влияют на специфичность. (Однако селекторы, объявленные внутри :not(), влияют)

Стили, обьявленные в элементе (например, style="font-weight:bold"), всегда переопределяют любые правила из внешних файлов стилей и, таким образом, их специфичность можно считать наивысшей.

Важное исключение из правил - !important

Когда при объявлении стиля используется модификатор !important, это объявление получает наивысший приоритет среди всех прочих объявлений. Хотя технически модификатор !important не имеет со специфичностью ничего общего, он непосредственно на неё влияет. Поскольку !important усложняет отладку, нарушая естественное каскадирование ваших стилей, он не приветствуется и следует избегать его использования. Если к элементу применимы два взаимоисключающих стиля с модификатором !important, то применен будет стиль с большей специфичностью.





**ID < CLASS < TAG**

**Типы отображения элементов. Свойство display**

Display: block, inline, inline-block

Block - элемент занимает 100% ширины. Можно указывать свойства размеров и отступов

Inline - элемент занимает столько места, сколько ему нужно

Inline-block - элемент занимает по умолчанию столько, сколько нужно, но можно указать свойства размеров.

**Позиционирование элементов. Свойство position\Position: static, relative, absolute, fixed**

Relative - позиция объекта не меняется, от него отсчитываются позиции дочерних absolute-элементов

Absolute - позиция рассчитывается относительно родительского элемента с absolute или relative

Fixed - позиция рассчитывается относительно страницы

1. БЭМ. Что такое БЭМ, для чего применяется эта технология. Из чего состоит БЭМ. Приведите примеры использования БЭМ.

БЭМ (Блок, Элемент, Модификатор) — компонентный подход к веб-разработке. В его основе лежит принцип разделения интерфейса на независимые блоки. Он позволяет легко и быстро разрабатывать интерфейсы любой сложности и повторно использовать существующий код, избегая «Copy-Paste».

**Блок**

Функционально независимый компонент страницы, который может быть повторно использован. В HTML блоки представлены атрибутом class.

Особенности:

- Название блока характеризует смысл («что это?» — «меню»: menu, «кнопка»: button), а не состояние («какой, как выглядит?» — «красный»: red, «большой»: big).

- Блок не должен влиять на свое окружение, т. е. блоку не следует задавать внешнюю геометрию (в виде отступов, границ, влияющих на размеры) и позиционирование.

- В CSS по БЭМ также не рекомендуется использовать селекторы по тегам или id.

Таким образом обеспечивается независимость, при которой возможно повторное использование или перенос блоков с места на место.

Вложенность

Блоки можно вкладывать друг в друга.

Допустима любая вложенность блоков.

**Элемент**

Составная часть блока, которая не может использоваться в отрыве от него.

Особенности:

Название элемента характеризует смысл («что это?» — «пункт»: item, «текст»: text), а не состояние («какой, как выглядит?» — «красный»: red, «большой»: big).

Структура полного имени элемента соответствует схеме: имя-блока\_\_имя-элемента. Имя элемента отделяется от имени блока двумя подчеркиваниями (\_\_).

**Модификатор – булевый, ключ - значение**

Cущность, определяющая внешний вид, состояние или поведение блока либо элемента.

Особенности:

Название модификатора характеризует внешний вид («какой размер?», «какая тема?» и т. п. — «размер»: size\_s, «тема»: theme\_islands), состояние («чем отличается от прочих?» — «отключен»: disabled, «фокусированный»: focused) и поведение («как ведет себя?», «как взаимодействует с пользователем?» — «направление»: directions\_left-top).

Имя модификатора отделяется от имени блока или элемента одним подчеркиванием (\_).

Методология разработана в компании **Яндекс** и широко используется в продуктах этой компании.

Она нашла применение в составе специально разработанного HTML5-фреймворка при редизайне и рефакторинге почтового сервиса mail.ru

Эту же методологию, помимо всего прочего, использовала и телерадиокомпания Би-Би-Си при разработке своего нового сайта

БЭМ также использована в таком выпущенном в 2015 году продукте Google, как Material Design Lite, HTML5-фреймворке, наподобие Twitter Bootstrap, поддерживающем Material design

**Django**

1. Шаблон проектирования MVC. Реализация шаблона проектирования MVC в Django. Что такое шаблон проектирования MVC, какие преимущества дает этот шаблон разработки ПО. Как реализован шаблон проектирования MVC в Django.

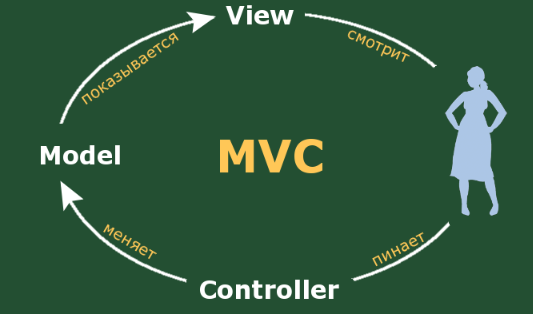


схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

**Модель** (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем) просто предоставляя доступ к данным и управлению ими.

Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом может быть встроено уведомление «наблюдателей».

Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной «модели».

**Представление** (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя

**Контроллер** (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений

Контроллер обеспечивает «связи» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

**Django** следует концепции MVC достаточно строго, что бы его можно было называть “MVC фреймворк“. Вот краткое описание того, как M, V и C разделены в среде Django:

M – часть доступа к данным, обслуживается системой баз данных Django (model)

V – часть, которая занимается выборкой того, что именно и как именно отображать, в Django этим занимаются представления (views) и шаблоны (templates).

C – часть, которая определяет представление в зависимости от указаний пользователя, этим занимается сам фреймворк, следуя настройкам URLconf и вызывая соответствующую функцию Python для заданного URL-а. urls.py

Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения.

1. Опишите процесс обработки запросов в Django.
2. Описать модели Django для конкретной предметной области. Для заданной предметной области разработайте код моделей, описание атрибутов и связей между моделями.
3. Для заданных моделей написать код, который выводит всех пользователей.
4. Для заданных моделей написать код, который получает пользователя с id = 100 или выводит ошибку при его отсутствии.
5. Для заданных моделей написать код, который выводит 10 первых постов пользователя.
6. Для заданных моделей написать код, который выводит все комментарии, которые оставляли к постам пользователя, отсортированные по дате добавления.
7. Для заданных моделей написать код, который считает количество комментариев всех пользователей.
8. Наследование шаблонов. Примеры использования. Для чего применяется наследование шаблонов в Django. Приведите примеры использования.
9. Миграции в Django. Что такое миграции в Django, для чего они используются, какую проблему решают.
10. Шаблонизатор в Django. Синтаксис шаблонов в Django. Базовый функционал: циклы, условия, фильтры, вызов методов объекта, обращение к атрибутам.
11. Файл urls.py. Назначение. Как осуществляется роутинг запросов внутри Django-приложения.
12. Обработка форм. Описание форм с использованием HTML. Какие инструменты Django предоставляет для автоматической генерации форм. Использование CSRF\_TOKEN. Как производится обработка данных, отправленных через форму.

Сама форма создаётся с помощью тега <form>, внутри которой могут быть любые необходимые теги, и характеризуется следующими необязательными параметрами:

адрес программы на веб-сервере, которая будет обрабатывать содержимое данных формы;

элементами формы, которые представляют собой стандартные поля для ввода информации пользователем;

кнопку отправки данных на сервер.

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Формы**</title>**

**</head>**

**<body>**

**<form** action="handler.php"**>**

**<p><input** name="a"**>** **<input** type="submit"**></p>**

**<form** action="handler.php"**>**

**<p><input** name="b"**>** **<input** type="submit"**></p>**

**</form>**

**</form>**

**</body>**

**</html>**

Перед отправкой данных браузер подготавливает информацию в виде пары «имя=значение», где имя определяется атрибутом name тега **<input>** или другим тегом допустимым в форме, а значение введено пользователем или установлено в поле формы по умолчанию. После нажатия пользователем кнопки Submit, происходит вызов обработчика формы, который получает введенную в форме информацию, а дальше делает с ней то, что предполагает разработчик. В качестве обработчика формы обычно выступает программа, заданная атрибутом action тега **<form>**. Программа может быть написана на любом серверном языке вроде PHP, Python, C# и др

Формы - это просто инструмент для упрощения и ускорения процесса создания данных POST из запроса. Ручным способом было бы сделать request.POST.get('somefield') для всех полей, которые есть в некоторой HTML-форме. Но Django может сделать лучше, чем это...

По сути, класс Form содержит несколько полей и выполняет следующие задачи:

* отображать входы HTML,
* собирать и проверять данные, когда пользователь отправляет их,
* Если поля не проверяются, возвращайте значения вместе с сообщениями об ошибках в HTML,
* Если все поля проверяются, укажите form.cleaned\_data словарь как удобный способ доступа к этим значениям в представлении.

[ModelForm](https://docs.djangoproject.com/en/1.10/topics/forms/modelforms/), который помогает созадавать формы непосредственно из модели. Класс ModelForm может применяться в ваших отображениях точно таким же образом как и "классический" класс формы Form.

В соответствии с данной диаграммой, главными моментами, которые берут на себя формы Django являются:

1. Показ формы по умолчанию при первом запросе со стороны пользователя.
   * Форма может содержать пустые поля (например, если вы создаете новую запись в базе данных), или они (поля) могут иметь начальные значения (например, если вы изменяете запись, или хотите заполнить ее каким-либо начальным значением).
   * Форма в данный момент является *несвязанной*, потому что она не ассоциируется с какими-либо введенными пользователем данными (хотя и может иметь начальные значения).
2. Получение данных из формы (из HTML-формы) со стороны клиента и связывание их с формой (классом формы) на стороне сервера.
   * Связывание данных с формой означает, что данные, введенные пользователем, а также возможные ошибки, при переотрисовке в дальнейшем, будут относиться именно к данной форме, а не к какой-либо еще.
3. Очистка и валидация данных.
   * Очистка данных  - это их проверка на наличие возможных значений, или вставок в поля ввода (то есть очистка - это удаление неправильных символов, которые потенциально могут использоваться для отправки вредоносного содержимого на сервер), с последующей конвертацией очищеных данных в подходящие типы данных Python.
   * Валидация проверяет, чтобы значения полей  (например, правильность введенных дат, их диапазон и так далее)
4. Если какие-либо данные являются неверными, то выполнение перерисовки формы, но на этот раз, с уже введенными пользователем данными и сообщениями об ошибках, описывающих возникшие проблемы.
5. Если все данные верны, то исполнение необходимых действий (например, сохранение данных, отправка писем, возврат результата поиска, загрузка файла и так далее)
6. Когда все действия были успешно завершены, то перенаправление пользователя на другую страницу.

[CSRF](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%B4%D0%B5%D0%BB%D0%BA%D0%B0_%D0%BC%D0%B5%D0%B6%D1%81%D0%B0%D0%B9%D1%82%D0%BE%D0%B2%D1%8B%D1%85_%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2) (Cross-Site Request Forgery, также XSRF) – опаснейшая атака, которая приводит к тому, что хакер может выполнить на неподготовленном сайте массу различных действий от имени других, зарегистрированных посетителей.

* CSRF-атака – это когда «злая страница» отправляет форму или запрос на сайт, где посетитель, предположительно, залогинен.

Если сайт проверяет только куки, то он такую форму принимает. А делать это не следует, так как её сгенерировал злой хакер.

* Для защиты от атаки формы, которые генерирует mail.com, подписываются специальным токеном. Можно не все формы, а только те, которые осуществляют действия от имени посетителя, то есть могут служить объектом атаки.
* Для подписи XMLHttpRequest токен дополнительно записывается в куку. Тогда JavaScript с домена mail.com сможет прочитать её и добавить в заголовок, а сервер – проверить, что заголовок есть и содержит корректный токен.
* Динамически сгенерированные формы подписываются аналогично: токен из куки добавляется как URL-параметр или дополнительное поле.

var request = new XMLHttpRequest();

var csrfCookie = document.cookie.match(/CSRF-TOKEN=([\w-]+)/);

if (csrfCookie) {

request.setRequestHeader("X-CSRF-TOKEN", csrfCookie[1]);

}



Класс Form является сердцем системы Django при работе с формами. Он определяет поля формы, их расположение, показ виджетов, текстовых меток, начальных значений, валидацию значений и сообщения об ошибках для "направильных" полей (если таковые имеются). Данный класс, кроме того, предоставляет методы для отрисовки самого себя в шаблоне при помощи предпоределенных форматов (таблицы, списки и так далее), или для получения значения любого элемента (позволяя выполнять более точную отрисовку).

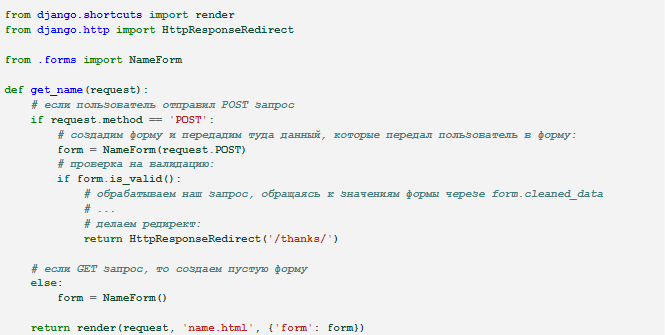
#### Объявление класса формы Form

Синтаксис объявления для класса формы Form очень похожь на объявление класса моделиModel, он даже использует те же типы полей (и некоторые похожие параметра). Это существенный момент, поскольку в обоих случаях нам надо убедиться, что каждое поле управляет правильным типом данных, соответствует нужному диапазону (или другому критерию) и имеет необходимое описание для показа/документации

Django предоставляет несколько мест где вы можете осуществить валидацию ваших данных. Простейшим способом проверки значения одиночного поля является переопределение методаclean\_**<fieldname>**() (здесь, **<fieldname>** это имя поля, которое вы хотите проверить).

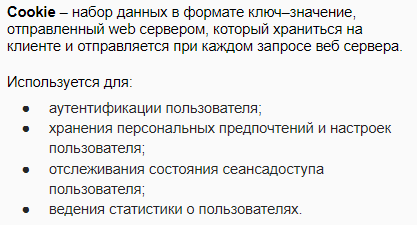
После создания формы мы вызываем функцию render(), чтобы создать HTML страницу; передаем ей в качестве параметров шаблон и контекст, который содержит объект формы. Кроме того, контекст содержит объект типа BookInstance, который мы будем использовать в шаблоне, для получения информации об обновляемой книге.

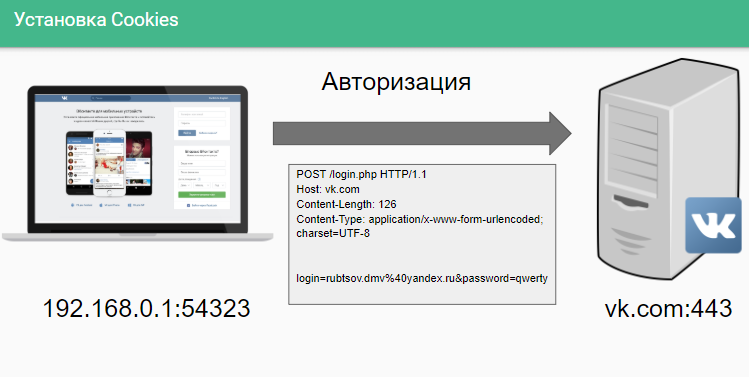
Если все таки у нас POST-запрос, тогда мы создаем объект с  именем form и заполняем его данными, полученными из запроса. Данный процесс называется связыванием (или, биндингом, от англ. "binding") и позволяет нам провести валидацию данных. Далее осуществляется валидация формы, при этом проверяются все поля формы — для этого используются как код обобщенного класса, так и пользовательских функций, в частности нашей функции проверки введенных дат clean\_renewal\_date().

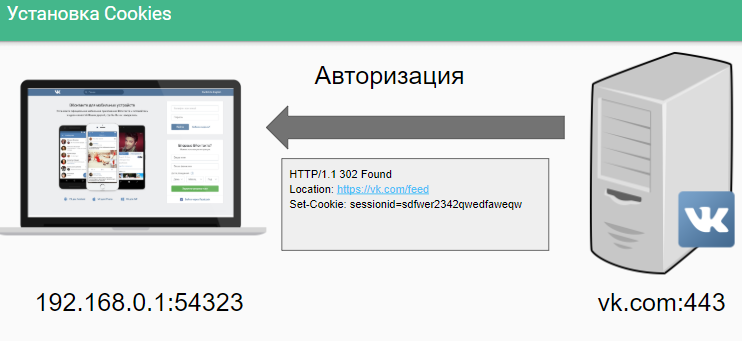


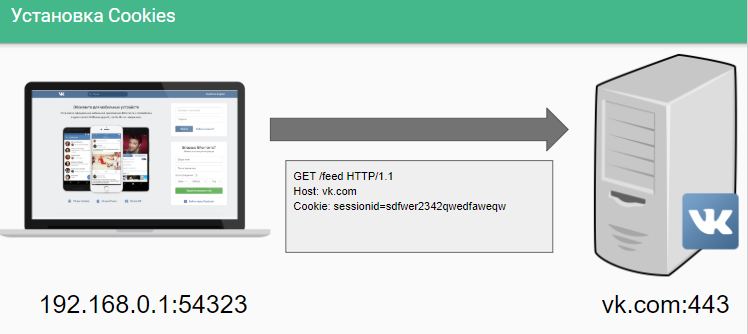


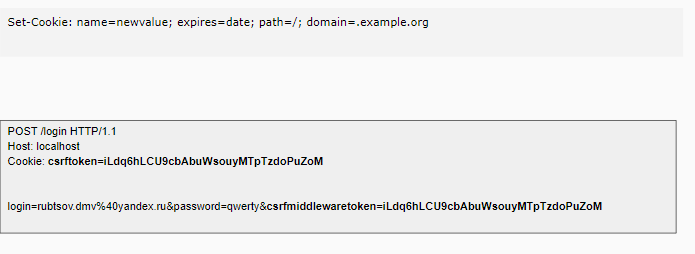
1. Авторизация в Web приложениях. Система авторизации в Django. Авторизация в Web приложениях. Система авторизации в Django. Файлы cookies. Пользовательские сессии. API библиотеки django.contrib.auth.



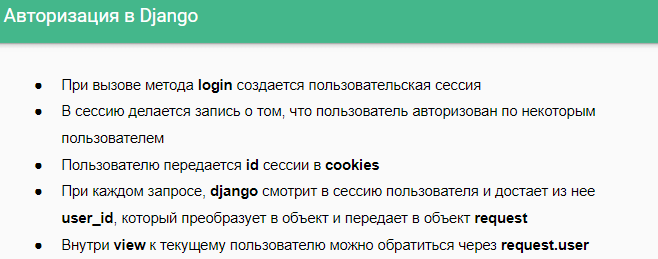












1. Class-based View в Django. Назначение, преимущества, недостатки. Написать пример CRUD сущности, используя class-based views.

Описание View - в виде класса, которые можно расширять, комбинировать

Логическое объединение связанных действий с сущностями (GET, POST, PUT, DELETE, ...)

Часть логики можно скрыть => меньше кода для простых стандартных действий (TemplateView, ListView, …)

Такие классы позволяют создавать структурированные, повторно используемые представления, базируясь на возможностях наследования и, в том числе, множественного наследования, - с использованием примесей(mixins). Django включает в себя набор общих(generic) представлений, которые идеально подходят для решения ряда рутинных задач

Недостаток этого подхода заключается в том, что вы не облегчите работу программистам, которые будут поддерживать ваш код, если они выучили API для Django CBV. Ведь в вашем проекте будет использоваться другой набор базовых классов.



1. SPA. Определение, преимущества, недостатки.

**Javascript**

1. Напишите функцию, суммирующую четные числа.
2. Перечислите основные задачи JS на стороне клиента. Перечислите преимущества и недостатки SPA по сравнению с традиционным приложением.

JavaScript изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются скриптами. В браузере они подключаются напрямую к HTML и, как только загружается страничка – тут же выполняются. Программы на JavaScript – обычный текст.

Современный JavaScript – это «безопасный» язык программирования общего назначения. Он не предоставляет низкоуровневых средств работы с памятью, процессором, так как изначально был ориентирован на браузеры, в которых это не требуется.

Что же касается остальных возможностей – они зависят от окружения, в котором запущен JavaScript. В браузере JavaScript умеет делать всё, что относится к манипуляции со страницей, взаимодействию с посетителем и, в какой-то мере, с сервером:

Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.

Реагировать на действия посетителя, обрабатывать клики мыши, перемещения курсора, нажатия на клавиатуру и т.п.

Посылать запросы на сервер и загружать данные без перезагрузки страницы (эта технология называется "AJAX").

Получать и устанавливать cookie, запрашивать данные, выводить сообщения

* Полная интеграция с HTML/CSS.
* Простые вещи делаются просто.
* Поддерживается всеми распространёнными браузерами и включён по умолчанию.

Одностраничные приложения позволяют имитировать работу десктоп приложений. Архитектура устроена таким образом, что при переходе на новую страницу, обновляется только часть контента. Таким образом, нет необходимости повторно загружать одни и те же элементы. Это очень удобно для разработчиков и пользователей. Для разработки SPA используется один из самых популярных языков программирования - javascript. Небольшое веб приложение можно сделать с библиотекой jQuery.

Основные преимущества Single Page Application: Высокая скорость. Так как SPA не обновляет всю страницу, а только нужную часть, это существенно повышает скорость работы. Высокая скорость разработки. Готовые библиотеки и фреймворки дают мощные инструменты для разработки веб приложений. Над проектом могут параллельно работать back-end и front-end разработчики. Благодаря четкому разделение они не будут мешать друг другу. Мобильные приложения. SPA позволяет легко разработать мобильное приложение на основе готового кода. При всех своих достоинствах, Single Page Application имеет некоторые недостатки, которые сдерживают рост популярности: Плохая SEO оптимизация. SPA работает на основе javascript и загружает информацию по запросу со стороны клиента. Поисковые системы с трудом могут имитировать данное поведение. Потому большинство страниц попросту недоступны для сканирования поисковыми ботами. Не активный javascript. Некоторые пользователи отключают javascript в своих браузерах, а без него ваше приложение не будет работать. Низкий уровень безопасности. JavaScript имеет низкий уровень безопасности, но если использовать современные фреймворки, они могу сделать ваше веб приложение безопасным. Но стоит обратить внимание, что использование jQuery может существенно понизить безопасность вашего проекта. Одностраничные веб приложения хорошо подходят для разработки динамических платформ, с небольшим объемом данных. Кроме того, если Вам потребуется в будущем построить мобильное приложение, SPA отлично подойдет как основа. Основным недостатком, который сдерживает стремительный рост популярности SPA это плохая SEO оптимизация. Проекты, где SEO имеет важнейший приоритет, стоит использовать MPA.

1. Перечислите основные типы данных JS. Приведите примеры операций добавления/удаления в массивах и объектах.
   * + 1. Число number
       2. Строка string
       3. Булевый Boolean
       4. null - ничего
       5. undefined – переменная объявлена и в нее ничего не записано
       6. Объект object {}
       7. Массив – разновидность object []

Способы добавления/удаления:

Из массива:

По номеру элемента: arr[3] = ‘lol’

**[Конец массива](https://learn.javascript.ru/array" \l "konets-massiva)**

**pop**

Удаляет *последний* элемент из массива и возвращает его:

var fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.pop() ); // удалили "Груша"

alert( fruits ); // Яблоко, Апельсин

**push**

Добавляет элемент *в конец* массива:

var fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша

Вызов fruits.push(...) равнозначен fruits[fruits.length] = ....

**[Начало массива](https://learn.javascript.ru/array" \l "nachalo-massiva)**

**shift**

Удаляет из массива *первый* элемент и возвращает его:

var fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.shift() ); // удалили Яблоко

alert( fruits ); // Апельсин, Груша

**unshift**

Добавляет элемент *в начало* массива:

var fruits = ["Апельсин", "Груша"];

fruits.unshift('Яблоко');

alert( fruits ); // Яблоко, Апельсин, Груша

Методы push и unshift могут добавлять сразу по несколько элементов:

var fruits = ["Яблоко"];

fruits.push("Апельсин", "Персик");

fruits.unshift("Ананас", "Лимон");

// результат: ["Ананас", "Лимон", "Яблоко", "Апельсин", "Персик"]

alert( fruits );

**Объекты**

data[name] = value;

data = {} //очистка

delete data //удаление

1. **Приведите все возможные примеры использования операторов if/else и switch.**

**Конструкция switch**

Конструкция switch заменяет собой сразу несколько if.

Она представляет собой более наглядный способ сравнить выражение сразу с несколькими вариантами

switch(x) {

case 'value1': // if (x === 'value1')

...

[break]

case 'value2': // if (x === 'value2')

...

[break]

default:

...

[break]

}

**If else**

if (year == 2011) {

alert( 'Да вы знаток!' );

} else {

alert( 'А вот и неправильно!' ); // любое значение, кроме 2011

}

**Приведите все возможные примеры использования операторов for/while/do while.**

while (условие) {

// код, тело цикла

}

do {

// тело цикла

} while (условие);

for (начало; условие; шаг) {

// ... тело цикла ...

}

1. С помощью каких методов можно получить DOM-элементы на странице. Приведите примеры использования основных методов.

***Элемент Объектной Модели Документа*** *(Document Object Model, сокращенно DOM)* — это какой-либо элемент страницы (div, h1, span ...), представленный в виде javascript-объекта. Он включает в себя свойства этого элемента и методы для работы с ними. Получить DOM-element'ы можно стандартными средствами javascript:

*// получение элемента DOM по идентификатору*

**var** el = document.getElementById('elementId');

*// получение всех элементов DOM, с определенным тегом*

**var** divs = document.getElementsByTagName("div");

Кроме этого, можно получить DOM-element'ы средствами библиотеки jQuery:

*// найдем нужные элементы страницы в виде объекта jQuery*

**var** jqElement = $(".someClass");

*// получим массив найденных элементов DOM из объекта jQuery*

**var** domElements = jqElement.get();

Псевдо-массив childNodes хранит все дочерние элементы, включая текстовые.

for (var i = 0; i < document.body.childNodes.length; i++) {

alert( document.body.childNodes[i] ); // Text, DIV, Text, UL, ..., SCRIPT

}

Свойства firstChild и lastChild обеспечивают быстрый доступ к первому и последнему элементу.

При наличии дочерних узлов всегда верно:

elem.childNodes[0] === elem.firstChild

elem.childNodes[elem.childNodes.length - 1] === elem.lastChild

Доступ к элементам слева и справа данного можно получить по ссылкам previousSibling / nextSibling.

Родитель доступен через parentNode. Если долго идти от одного элемента к другому, то рано или поздно дойдёшь до корня DOM, то есть до document.documentElement, а затем и document.

1. Опишите, какие категории событий на странице бывают. Для каждой категории напишите примеры событий. Приведите пример, как повесить обработчик на событие.

Событие - это сигнал от браузера о том, что что-то произошло.

Есть множество самых различных событий.

* DOM-события, которые инициируются элементами DOM. Например, событие click происходит при клике на элементе, а событие mouseover - когда указатель мыши появляется над элементом,
* События окна. Например событие resize - при изменении размера окна браузера,
* Другие события, например load, readystatechange. Они используются в технологии AJAX.

**События мыши:**

* click – происходит, когда кликнули на элемент левой кнопкой мыши
* contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши
* mouseover – возникает, когда на элемент наводится мышь
* mousedown и mouseup – когда кнопку мыши нажали или отжали
* mousemove – при движении мыши

**События на элементах управления:**

* submit – посетитель отправил форму <form>
* focus – посетитель фокусируется на элементе, например, нажимает на <input>

**Клавиатурные события:**

* keydown – когда посетитель нажимает клавишу
* keyup – когда посетитель отпускает клавишу

**События документа:**

* DOMContentLoaded – когда HTML загружен и обработан, DOM документа полностью построен и доступен.

**События CSS:**

* transitionend – когда CSS-анимация завершена.

Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло.

Именно благодаря обработчикам JavaScript-код может реагировать на действия посетителя.

**HTML**

<input value="Нажми меня" onclick="alert('Клик!')" type="button">

**DOM**

<input id="elem" type="button" value="Нажми меня" />

<script>

elem.onclick = function() {

alert( 'Спасибо' );

};

</script>

Внутри обработчика события this ссылается на текущий элемент, то есть на тот, на котором он сработал.

Это можно использовать, чтобы получить свойства или изменить элемент.

В коде ниже button выводит свое содержимое, используя this.innerHTML:

<button onclick="alert(this.innerHTML)">Нажми меня</button>

Методы addEventListener и removeEventListener являются современным способом назначить или удалить обработчик, и при этом позволяют использовать сколько угодно любых обработчиков.

Назначение обработчика осуществляется вызовом addEventListener с тремя аргументами:

element.addEventListener(event, handler[, phase]);

**event -** Имя события, например click

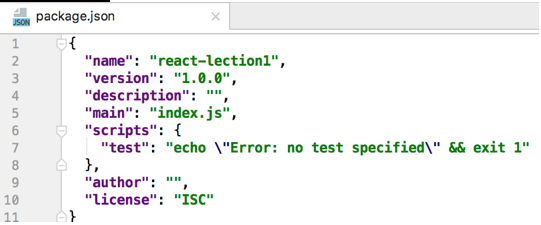
**handler -** Ссылка на функцию, которую надо поставить обработчиком.

**Phase -** Необязательный аргумент, «фаза», на которой обработчик должен сработать

Удаление обработчика осуществляется вызовом removeEventListener

1. Объясните, что такое npm. Опишите основные поля package.json. Укажите, как установить зависимости в проект. Как это сделать для зависимостей для разработки и для продакшна.

**npm** — менеджер пакетов, входящий в состав Node.js. NPM является аббревиатурой Node Package Manager это инструмент (программа) управляющая библиотеками программирования Javascript для Node.js, данный инструмент на самом деле необходим для мира открытого исходного кода.



**"dependencies"**: {

**"react"**: **"^16.1.1"**,

**"react-dom"**: **"^16.1.1"**

}

Файл package.json будет создан в качестве name будет использовано название каталога.

Вы также можете добавить private: true, чтобы предотвратить случайную публикацию частных репозиториев или подавления любых предупреждений при выполнении npm install. Создадим новый каталог и используем package.json для установки зависимостей.

. С помощью start вы можете описать, как нужно запускать приложение, а test используется для запуска тестов.

Когда вы собираете веб-приложения, вам может потребоваться минимизировать ваши JavaScript-файлы, объединить CSS-файлы и так далее. Модули, которые это сделают, будут выполняться только во время создания ресурсов, поэтому работающее приложение не нуждается в них. **"devDependencies":**

1. Расскажите, что такое babel и для чего он используется. Какие основные пресеты вы использовали? Поясните, зачем нужен файл .babelrc?

Babel.JS – это транспайлер, переписывающий код на ES-2015 в код на предыдущем стандарте ES5.

Он состоит из двух частей:

* Собственно транспайлер, который переписывает код.
* Полифилл, который добавляет методы Array.from, String.prototype.repeat и другие.

Обычно Babel.JS работает на сервере в составе системы сборки JS-кода (например webpack или brunch) и автоматически переписывает весь код в ES5.

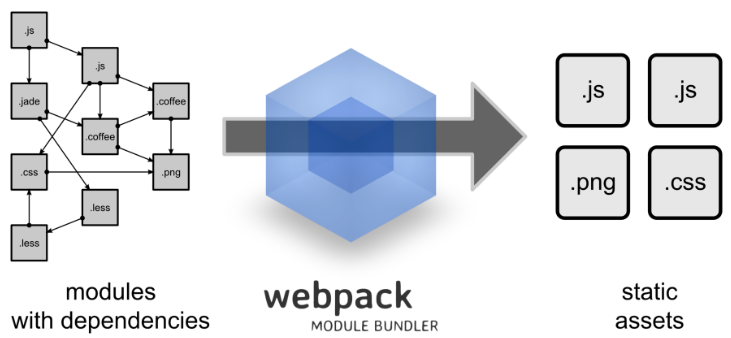
Создадим **.babelrc** в корне проекта. На основе его Babel будет выбирать нужный пресет. Содержимое файла следующее:

{  
 "presets": ["@babel/preset-env", "@babel/preset-react"]  
}

Конфигурируется Babel по средствам плагинов (plugin) и предустановок (preset). Предустановка - это набор плагинов.

Компиляция: npx babel index.js --out-file index-compiled.js --presets=env

1. Webpack. Что это, для чего используется. Приведите примеры основных полей конфига webpack. Напишите минимальный конфиг, необходимый для “сборки” одного js-файла.



это пакет модулей JavaScript с открытым исходным кодом . [2] [3] [4] [5] Его основная цель - объединять файлы JavaScript для использования в браузере, но также он способен преобразовывать, связывать или упаковывать практически любой ресурс. Webpack принимает модули с зависимостями и генерирует статические ресурсы, представляющие эти модули. [7] Это пакетный модуль преимущественно для JavaScript, но он может преобразовывать внешние ресурсы, такие как HTML, CSS, даже изображения, если включены соответствующие плагины.

1. Создаём файл webpack.config.js в корне проекта.
2. Копируем и вставляем следующий фрагмент кода, который подскажет Webpack, что от него нужно.

module.exports = {

entry: './src/app.js',

output: {

path: './dist',

filename: 'bundle.js'

}

};

«Entry» и «Output»

«Entry» (входные данные). Входной файл называется app.js и расположен в каталоге src.

«Output» (выходные данные). Это место, куда Webpack будет размещать готовый файл. Он будет называться bundle.js и появляться в каталоге dist («distribution» — с англ. «размещение»).

1. Что такое loader’ы в webpack’е? Опишите основные loader’ы, которые вы использовали. За что отвечает директива test и exclude в лоадерах?

Webpack позволяет использовать лоадеры для предварительной обработки файлов. Это позволяет связывать любые статические ресурсы, кроме JavaScript. Вы можете легко написать свои собственные загрузчики, используя Node.js.

Лоадеры активируются с помощью loadername!префиксов в require()операторах или автоматически применяются через регулярные выражения из конфигурации вашего веб-пакета.

Лоадер — это инструмент, который определенным образом преобразует исходный код модулей. Лоадер позволяет преобразовывать файлы, при их импорте или «загрузке».

Таким образом, лоадеры похожи на «задачи» (tasks) в некоторых системах сборки и представляют собой мощный инструмент для обработки файлов. Лоадеры могут преобразовывать файлы одного языка (например, TypeScript) в файлы JavaScript, или встраивать изображения в формате data URL. Лоадеры также позволяют импортировать Ваши css-файлы напрямую в js-модули!

rules указывает правила обработки файлов

[json-loader](https://webpack.js.org/loaders/json-loader)Загружает файл [JSON](http://json.org/)

* [babel-loader](https://webpack.js.org/loaders/babel-loader)Загружает код ES2015 + и переносится в ES5 с помощью [Babel](https://babeljs.io/)
* [html-loader](https://webpack.js.org/loaders/html-loader) Экспортирует HTML как строку, требует ссылки на статические ресурсы
* [css-loader](https://webpack.js.org/loaders/css-loader) Загружает файл CSS с разрешенными импортами и возвращает код CSS

test: условие, которое должно быть выполнено

exclude: условие, которое не должно выполняться

Загрузчики (лоадеры) позволяют вебпаку обрабатывать не только файлы JavaScript, т.к. сам по себе вебпак понимает только JS.

Загрузчики трансформируют все типы файлов в модули, которые затем можно добавить в граф зависимостей вашего приложения (а значит, и в бандл).

test: /\.(js|jsx)$/,  
exclude: /node\_modules/

Наша цель — чтобы все статические файлы в нашем проекте стали заботой webpack’а, а не браузера. Но это не значит, что все они должны быть объединены в один файл. Webpack обрабатывает каждый файл (.css, .html, .scss, .jpg и т.д.) как модуль. При этом сам webpack понимает только JavaScript.

Загрузчики webpack трансформируют эти файлы в модули согласно тому, как они добавлены в граф зависимостей. По своей сути, загрузчики — это трансформации, которые производятся над ресурсом зависимости вашего приложения. Они представляют из себя функции, запускаемые на NodeJS, принимающие на вход в качестве параметра содержимое файла зависимости и возвращающие новое содержимое.

На самом верхнем уровне у загрузчиков есть две задачи:

* Определить файлы, которые следует трансформировать этим загрузчиком, с помощью свойства test;
* Трансформировать этот файл так, чтоб он мог быть добавлен в наш граф зависимостей и, в конечном счёте, в наш bundle, с помощью свойства use.
* В этой конфигурации мы определили правило в соответствующем списке правил модуля с двумя обязательными свойствами: test и use. Это говорит webpack-компилятору примерно следующее:
* «Привет, компилятор Webpack! Если ты при просмотре моего кода вдруг повстречаешь пути, в которых содержатся файлы с расширением ‘.js’ или ‘.jsx’ внутри вызовов require() или в выражении import, используй, пожалуйста, babel-loader для того, чтоб трансформировать его перед добавлением в сборку. Спасибо, друг!»
* Важно помнить, когда определяете правила в конфиге webpack, что определять их нужно в module.rule, а не в rules. Если вы сделаете это неправильно, webpack будет на вас орать и ругаться красными буквами.
* Конечно, настройка загрузчиков этими свойствами не ограничивается. Но давайте перейдём к плагинам.

1. Расскажите, что означает запуск webpack в watch-режиме. Поясните, что делает webpack-dev-server и перечислите основные опции его конфига. Опишите, зачем нужен HTMLWebpackPlugin и обязательно ли его использовать?

* флаг наблюдения watch для включения режима просмотра. Он будет следить за изменениями вашего файла и компилировать всякий раз при его изменении:

"scripts": {

"dev": "webpack --mode development ./src/index.js --output ./dist/main.js --watch",

"build": "webpack --mode production ./src/index.js --output ./dist/main.js --watch"

webpack будет следить за файлами и при изменениях перекомпилировать.

Webpack –w

webpack dev server. Это сервер, для загрузки вашего приложение, которое очень просто синхронизируется с webpack скриптами. То есть нам не нужно настраивать свой собственный сервер, например, на nodejs. Позволяет поднять сервер **для разработки**, который будет раздавать созданные файлы. Поддерживает **hot-module-replacement**.

**webpack.config.js**

var path = require('path');

module.exports = {

//...

devServer: {

contentBase: path.join(\_\_dirname, 'dist'), // Сообщите серверу, где следует обслуживать контент. Это необходимо, только если вы хотите обслуживать статические файлы. devServer.publicPathбудет использоваться для определения того, откуда следует обслуживать пакеты, и имеет приоритет.

compress: true, // Включите [сжатие gzip](https://betterexplained.com/articles/how-to-optimize-your-site-with-gzip-compression/) для всего обслуживаемого

port: 9000

}

};

The HtmlWebpackPlugin simplifies creation of HTML files to serve your webpack bundles. This is especially useful for webpack bundles that include a hash in the filename which changes every compilation. You can either let the plugin generate an HTML file for you, supply your own template using lodash templates, or use your own loader.

Installation

npm install --save-dev html-webpack-plugin

Basic Usage

The plugin will generate an HTML5 file for you that includes all your webpack bundles in the body using script tags.

Упрощает создание HTML - файлов для обслуживания WebPack пакетов. Это особенно полезно для webpack пакетов, которые содержат хэш в имени файла, который изменяется каждую компиляцию. Вы можете позволить плагину сгенерировать HTML-файл, предоставить собственный шаблон с использованием шаблонов lodash.

1. Расскажите, что такое React, какие основные принципы он использует. Компонент и методы его жизненного цикла.

React (иногда React.js или ReactJS) — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

Однонаправленная передача данных

Свойства передаются от родительских компонентов к дочерним. Компоненты получают свойства как множество неизменяемых (англ. immutable) значений, поэтому компонент не может напрямую изменять свойства, но может вызывать изменения через callback функции. Такой механизм называют «свойства вниз, события наверх».

Виртуальный DOM

React использует виртуальный DOM (англ. virtual DOM). React создает кэш структуру в памяти, что позволяет вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления DOM браузера. Таким образом программист может работать со страницей, считая, что она обновляется вся, но библиотека самостоятельно решает, какие компоненты страницы необходимо обновить.

JSX

JavaScript XML (JSX) - это расширение синтаксиса JavaScript, которое позволяет использовать похожий на HTML синтаксис для описания структуры интерфейса. Как правило, компоненты написаны с использованием JSX, но также есть возможность использования обычного JavaScript.[10] JSX напоминает другой язык, созданный в компании Фейсбук для расширения PHP, XHP .

Компоненты

JSX

State

Объект state описывает внутреннее состояние компонента, он похож на props за тем исключением, что состояние определяется внутри компонента и доступно только из компонента.

Props

Props представляет коллекцию значений, которые ассоциированы с компонентом. Эти значения позволяют создавать динамические компоненты, которые не зависят от жестко закодированных статических данных.

<p>Имя: {props.name}</p>

<p>Возраст: {props.age}</p>

Параметр props, который передается в функцию компонента, инкапсулирует свойства объекта. В частности, свойство name и age. При рендеринге мы можем создать набор компонентов Hello, но передать в них разные данные для name и age. И таким образом, получим набор однотипной разметки html, наполненной разными данными.

**Компонент** это одна изолированная часть интерфейса. Даже простые HTML теги это компонент сам по себе и они добавлены по-дефолту. Самостоятельная часть приложения. Сожержит часть логики. Описывает свой вид. Может использоватья повторно.

В процессе работы компонент проходит через ряд этапов **жизненного цикла**. На каждом из этапов вызывается определенная функция, в которой мы можем определить какие-либо действия:

constructor(props): конструктор, в котором происходит начальная инициализация компонента

componentWillMount(): вызывается непосредственно перед рендерингом компонента

render(): рендеринг компонента

componentDidMount(): вызывается после рендеринга компонента. Здесь можно выполнять запросы к удаленным ресурсам

componentWillUnmount(): вызывается перед удалением компонента из DOM

Кроме этих основных этапов или событий жизненного цикла, также имеется еще ряд функций, которые вызываются при обновлении состояния после рендеринга компонента:

shouldComponentUpdate(nextProps, nextState): вызывается каждый раз при обновлении объекта props или state. В качестве параметра передаются новый объект props и state. Эта функция должна возвращать true (надо делать обновление) или false (игнорировать обновление). По умолчанию возвращается true. Но если функция будет возвращать false, то тем самым мы отключим обновление компонента, а последующие функции не будут срабатывать.

componentWillUpdate(nextProps, nextState): вызывается перед обновлением компонента (если shouldComponentUpdate возвращает true)

render(): рендеринг компонента (если shouldComponentUpdate возвращает true)

componentDidUpdate(prevProps, prevState): вызывается сразу после обновления компонента (если shouldComponentUpdate возвращает true). В качестве параметров передаются старые значения объектов props и state.

И отдельно стоит отметить функцию componentWillReceiveProps(nextProps), которая вызывается при обновлении объекта props. Новые значения этого объекта передаются в функции в качестве параметра. Как правило, в этой функции устанавливаются те свойства компонента, в том числе из this.state, которые зависят от значений из props.

1. Объясните принцип потока данных в React. Объясните разницу между state и props. Поясните принцип разделения компонентов на контейнеры и компоненты.

В реакте поток данных — однонаправленный. Это значит что данные передаются как водопад, сверху вниз, от родителя к ребенку, через props. Что такое props? Это неизменяемый объект, предназначенный только для чтения. Проще понять если думать о компонентах, как о функциях(а они по сути ими и являются), props—это просто аргумент функции, с которым мы можем работать внутри, но не изменять.

Container - получает и обрабатывает данные

Component - отображает данные

**Родитель ➡️ Ребенок** — используй Props

Это самый простой способ передачи данных.

1. В родительском компоненте передаём данные из стейта в дочерний.

<ChildComponent name={this.state.name} />

2. В дочернем принимаем и выводим.

<p>Props: {this.props.name}</p>

**Родитель ⬅️ Ребенок** — используй Callback

Эта передача данных напоминает мне бумеранг, есть точка старта — это наш родительский компонент, есть точка максимального отдаления — это дочерний компонент. И есть наш инструмент —бумеранг, в реакте это будет функция, которая находится в родителе и передаётся через props в ребенка, где и вызывается.

1. Создаём в родителе функцию updateData. У неё есть входной параметр value, этот параметр мы присваиваем в стейт нашего компонента, с помощью функции setState. Бумеранг готов!

updateData = (value) => {

this.setState({ name: value })

}

2. Передаём в дочерний элемент через props функцию updateData. Мы запустили бумеранг

<ChildComponent updateData={this.updateData} />

**Ребенок ➡️ Ребенок**—используй родителя.

Для того чтобы передать данные между соседними компонентами, нужно использовать посредника—их родителя. Сначала нужно передать данные от ребенка к родителю как аргумент коллбека. Потом присвоить эти данные в стейт родителя и передать через props другому компоненту.

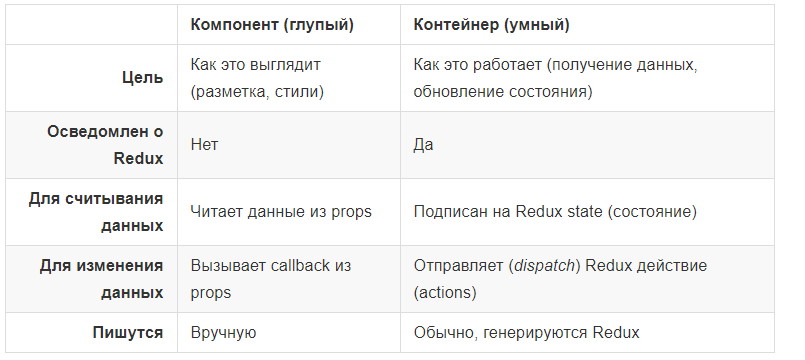
**Props** представляет коллекцию значений, которые ассоциированы с компонентом. Эти значения позволяют создавать динамические компоненты, которые не зависят от жестко закодированных статических данных.

Объект state описывает внутреннее состояние компонента, он похож на props за тем исключением, что состояние определяется внутри компонента и доступно только из компонента.

Если props представляет входные данные, которые передаются в компонент извне, то состояние хранит такие объекты, которые создаются в компоненте и полностью зависят от компонента.

Также в отличие от props значения в state можно изменять.

И еще важный момент - значения из state должны использоваться при рендеринге. Если какой-то объект не используется в рендерниге компонента, то нет смысла сохранять его в state.



Одна вещь, которую вы можете заметить — контейнеры совсем не похожи на обычные компоненты. Они могут:

Получать и передавать части глобального состояния (например, из Redux) в дочерние компоненты.

Выполнять запрос доступа к данным (например, GraphQL) и передавать результат дочерним компонентам.

https://medium.com/@karafizi/%D0%BF%D0%BE%D1%81%D1%82%D0%B8%D0%B3%D0%B0%D0%B5%D0%BC-%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D1%8B-%D0%B2%D1%8B%D1%81%D1%88%D0%B5%D0%B3%D0%BE-%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BA%D0%B0-%D0%BF%D0%B5%D1%80%D0%B5%D0%B2%D0%BE%D0%B4-87ecfad848e5

1. Составьте пример двух react-компонентов, один из которых хранит массив данных в state и передает их через props-ы дочерних компонентам.

**Родитель ➡️ Ребенок** — используй Props

Это самый простой способ передачи данных.

1. В родительском компоненте передаём данные из стейта в дочерний.

<ChildComponent name={this.state.name} />

2. В дочернем принимаем и выводим.

<p>Props: {this.props.name}</p>

import { render } from 'react-dom';

class ChildComponent extends React.Component {

render() {

return (

<div>

<p>Props: {this.props.name}</p>

</div>

)

}

}

class ParentComponent extends React.Component {

state = {

name: ‘Nadi’

};

render() {

return (

<div>

<p>State: {this.state.name}</p>

<ChildComponent name={this.state.name} />

</div>

)

}

}

render(<ParentComponent />, document.getElementById('root'));

1. Составьте пример react-компонента, изменяющего стейт по клику на кнопку.



1. Что такое prop-types. Покажите, как существуют основные типы prop-type’сов.

Для осуществления проверки типов свойств (props) компонента, вы можете определить специальное свойство класса компонента — propTypes:

Greeting.propTypes = { name: React.PropTypes.string };

Модуль React.PropTypes экспортирует набор валидаторов, которые могут быть использованы для проверки получаемых данных. В этом примере мы используем валидатор React.PropTypes.string. В случае, если в свойство будет передано невалидное значение — в консоли JavaScript будет показано предупреждение. В целях производительности, propTypes проверяются только в режиме разработки (development).

*обязательное наличие того или иного свойства не проверяется. // Для этого есть отдельная расширенная нотация валидаторов.* optionalArray: React.PropTypes.array, optionalBool: React.PropTypes.bool, optionalFunc: React.PropTypes.func, optionalNumber: React.PropTypes.number, optionalObject: React.PropTypes.object, optionalString: React.PropTypes.string, optionalSymbol: React.PropTypes.symbol,

optionalElement: React.PropTypes.element,

// Вы можете ограничить значение свойства списком значений.

optionalEnum: React.PropTypes.oneOf(['News', 'Photos']),

*// Ограничение, что свойство должно быть массивом значений определенного типа.* optionalArrayOf: React.PropTypes.arrayOf(React.PropTypes.number),

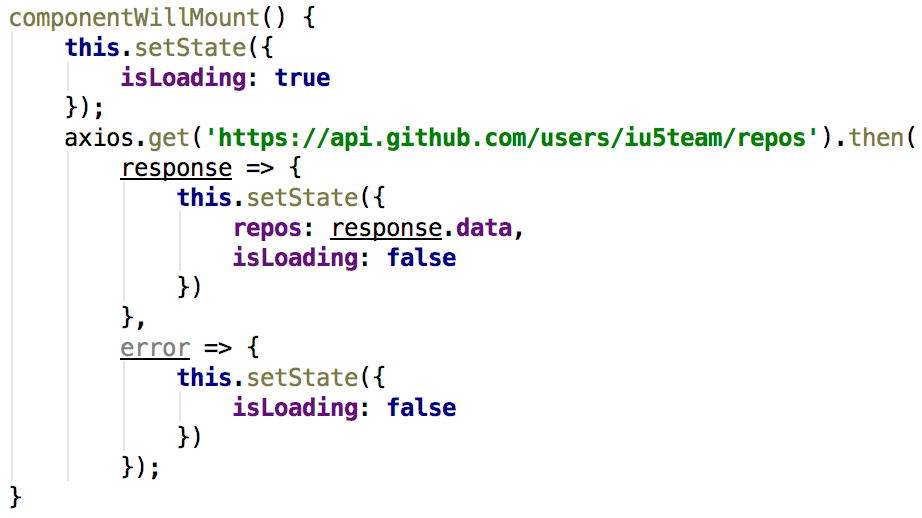
ы можете определить значения по умолчанию для ваших props определив специальное свойство класса компонента — defaultProps:

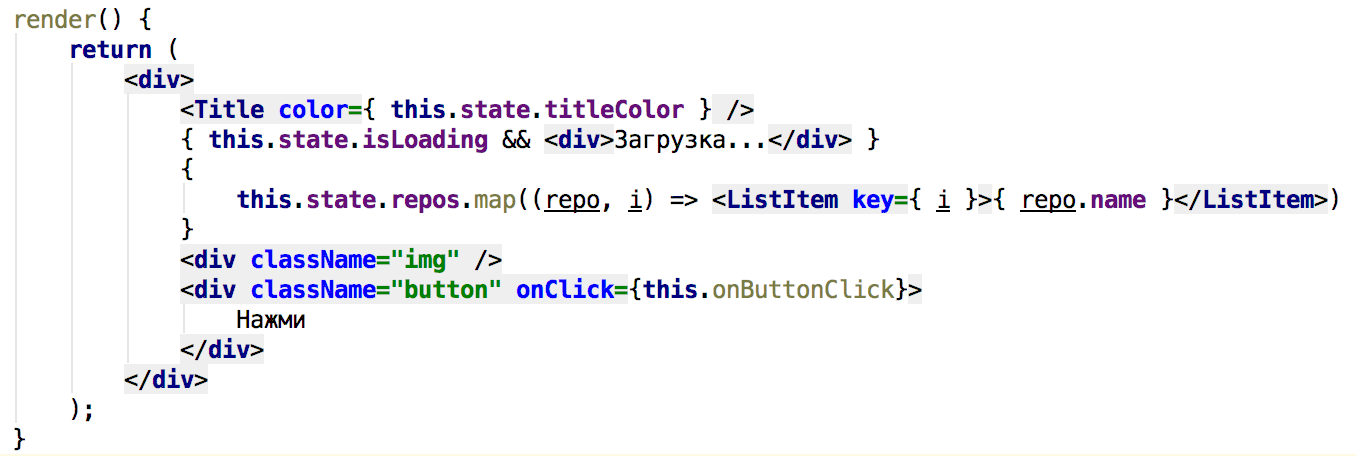
Greeting.defaultProps = {

name: 'Stranger'

};

1. Приведите пример получения данных из api и сохранения в стейт react-компонента. Учтите флаги загрузки и ошибок.





1. Расскажите про react-router. Приведите пример использования react-роутера для приложения с двумя страницами. За что отвечает опция exact у компонента Route?

React Router это стандартная библиотека маршрутизации (routing) в React . Он хранит интерфейс приложения синхронизированным с URL на браузере. React Router позволяет вам маршрутизировать "поток данных" (data flow) в вашем приложении понятным способом. Он подобен утверждению, если у вас есть данный URL, он будет подобен этому Route (маршруту), и интерфейс будет таким!

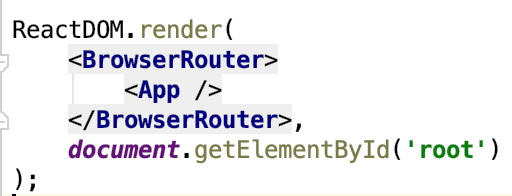
React Router в самом деле имеет пользу и широко используется в приложениях React больше со стороны Server чем приложения React со стороны Client. Точнее React Router обычно используется в приложеии React в среде NodeJS Server, он разрешает вам определять динамичные URL но соответствуюшие философии "Single Page Application" (Одностраничное приложение) у React . Для разработки приложения React вам нужно написать много Component, но нужен только один файл для обслуживания пользователей это index.html (Это основа).

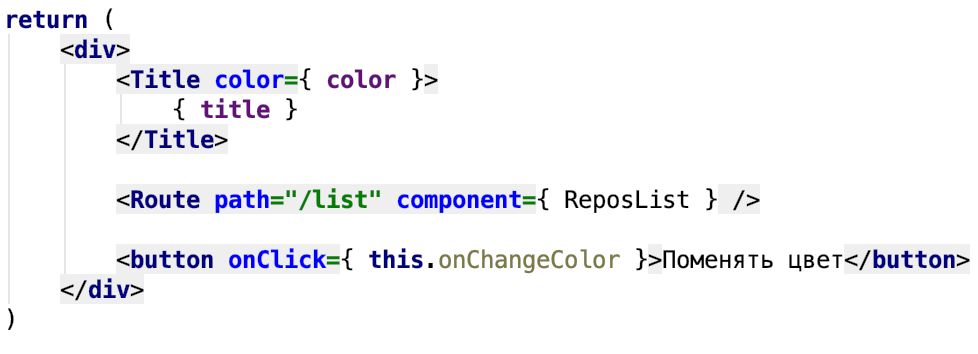
React Router помогает вм определить динамичные URL, и выбрать подходящий Component для render (изображения) на браузере пользователя соответственно каждому URL.

Компонент  **<Route>** определяет маппинг (mapping) между  **URL** и одним  **Component**. Это значит когда пользователь проходит по **URL** на браузере, соответствующий  **Component** будет  **render** (изображен) на интерфейсе

Атрибут exact используется в <Route>, чтобы сказать данный <Route> работает только если URL на браузере абсолютно подходит значению атрибута его path.

<Route exact path="/" component={Home}/>







https://hghltd.yandex.net/yandbtm?fmode=inject&url=https%3A%2F%2Fo7planning.org%2Fru%2F12137%2Fundertanding-react-router-with-example-on-the-client-side&tld=ru&lang=ru&la=1546449664&tm=1546969041&text=react%20router%20%D1%8D%D1%82%D0%BE&l10n=ru&mime=html&sign=e757a24ff5b29384c0bf57a945877394&keyno=0

1. Приведите примеры компонентов, предоставляемых библиотекой react-router. Для чего нужен декоратор withRouter? Опишите, как получить из location’а параметр, передаваемый в урле. Приведите пример.

### <BrowserRouter> vs <HashRouter>

**React** **Router** предоставляет вам 2 компонента  это   **<BrowserRouter> & <HashRouter>**. Эти два компонента отличаются в виде  **URL,** которые они будут создавать и синхронизировать.

// <BrowserRouter>

http://example.com/about

// <HashRouter>

http://example.com/#/about

**<BrowserRouter>** более широко используется, он использует  **History API** имеющийся в  **HTML5** для мониторинга истории вашего роутера. При этом  **<HashRouter>** использует  **hash** у **URL** ( ***window.location.hash***) чтобы запомнить все вещи. Если вы хотите поддерживать старые браузеры или приложение  **React** используя  **Router** со стороны клиента, то  **<HashRouter>** является правильным выбором.

### <Route>

Компонент  **<Route>** определяет маппинг (mapping) между  **URL** и одним  **Component**. Это значит когда пользователь проходит по **URL** на браузере, соответствующий  **Component** будет  **render** (изображен) на интерфейсе.

You can get access to the [history](https://github.com/ReactTraining/react-router/blob/master/packages/react-router/docs/api/history.md) object's properties and the closest [<Route>](https://github.com/ReactTraining/react-router/blob/master/packages/react-router/docs/api/Route.md)'s [match](https://github.com/ReactTraining/react-router/blob/master/packages/react-router/docs/api/match.md) via the withRouter higher-order component. withRouter will pass updated match, location, and history props to the wrapped component whenever it renders.

Можно получить доступ к свойства объекта history и ближайшим матчам роутера через с помощью компонента более высокого порядка. withRouter обновит пропсы match, location, and history обертки всякий раз когда он рендерит.

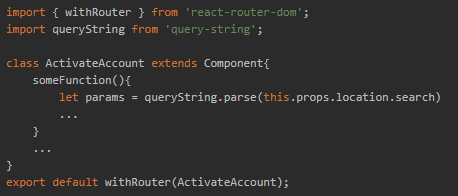
Если вы используете шаблоны реакции, где конец вашего файла реакции выглядит следующим образом: export default SomeComponent вам необходимо использовать компонент более высокого порядка, withRouter,

Во-первых, вам нужно импортировать withRouter: import {withRouter} from 'react-router-dom';

Затем вы захотите использование  withRouter. Вы можете сделать это, изменив экспорт ваших компонентов ... из export default ComponentName в export default withRouter(ComponentName),

Затем вы можете получить маршрут (и другую информацию) из реквизита. В частности, location, match, а также history, Код для вышивания имени пути будет console.log(this.props.location.pathname);

**this.props.location.search – получение параметров из URL**



1. Опишите принципы методологии flux. Проведите аналогии между flux и redux. Какие средства из redux реализуют какие элементы методологии?

Flux- это вид архитектуры, которую Facebook использует, чтобы работать с React.

Итак, React отвечает за V или View в MVC. А как насчет M или Model части? Flux, шаблон проектирования, отвечает за M в MVC.

Это архитектура, ответственная за создание слоя данных в JavaScript приложениях и разработку серверной стороны в веб-приложениях. Flux дополняет составные компоненты вида View в React, используя однонаправленный поток данных.

Также можно сказать, что Flux больше чем шаблон, больше чем фреймворк и имеет 4 главных компонента (более подробно они будут рассмотрены позже):

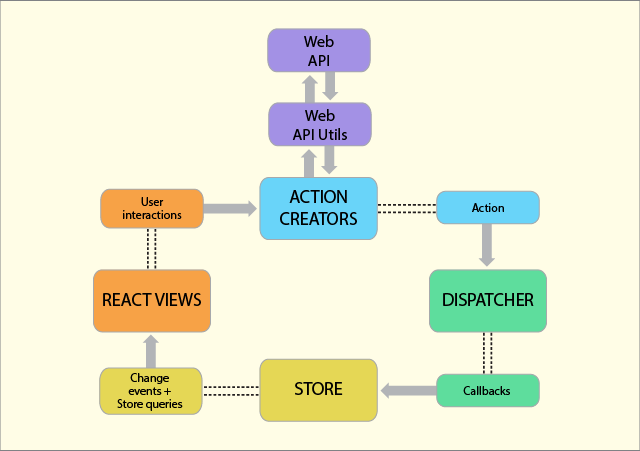
* Диспетчер (Dispatcher)
* Хранилище (Stores)
* Виды (Views) (React компонент)
* Действие (Action)

Flux имеет открытый исходный код, и это скорее шаблон проектирования, а не фреймворк, поэтому его можно использовать сразу. То, что отличает его от других фреймворков, то отличает его и от шаблона проектирования MVC.

Flux помогает сделать код более предсказуемым, в сравнении с МVC фреймворками. Разработчики могут создавать приложения, не беспокоясь о сложных взаимодействиях между источниками данных.

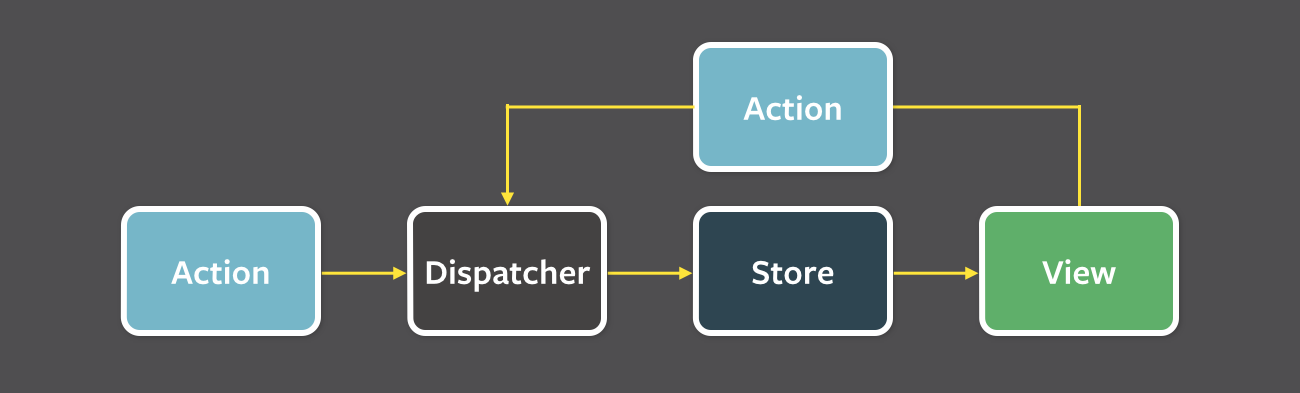
Flux выгодно отличается более организованным потоком данных — однонаправленным. То, что он однонаправленный, главная особенность Flux. Эти действия распространяются на новую систему в отношении взаимодействия с пользователем.

Также вы можете начать использовать Flux, не используя при этом весь новый код, в отличие от React.



* **Actions**— помощники, которые передают данные в Dispatcher
* **Dispatcher**— получает эти действия и передает полезную нагрузку зарегистрированным callback-ом.
* **Stores**— действуют как контейнеры для состояния приложения и логики. Реальная работа приложения происходит в Stores. Stores, зарегистрированные для прослушивания действий Dispatcher, будут соответственно и обновлять View.
* **Controller Views** — React компоненты захватывают состояние из Stores, а затем передают дочерним компонентам.

1. Изобразите диаграмму потока данных в flux и опишите, что происходит на каждом этапе.



Контроллеры в MVC и Flux различаются. Здесь контроллеры являются Controller-View и находятся на самой вершине иерархии. View — это React компоненты.

Весь функционал, как правило, находится в Store. В Store выполняется вся работа и сообщается Dispatcher, какие события/действия он прослушивает.

Когда происходит событие, Dispatcher отправляет “полезную нагрузку” в Store, который зарегестрирован для прослушивания конкретно этого события. Теперь в Store необходимо обновить View, что в свою очередь вызывает действие. Дествие, точно также как и имя, тип события и много другое известны заранее.

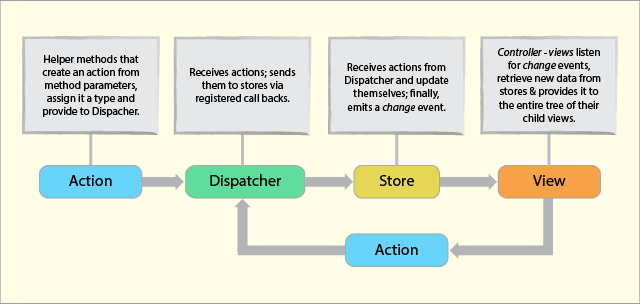
View распространяет Action через центральный Dispatcher, и это будет отправлено в различные Stores. Эти Stores будут отображать бизнес-логику приложения и другие данные. Store обновляет все View.

Наиболее хорошо это работает совместно со стилем программирования React, тогда Store отправляет обновления без необходимости подробно описывать, как изменять представления между состояниями.

Это доказывает, что шаблон проектирования Flux следует за однонаправленным потоком данных. Action, Dispatcher, Store и View — независимые узлы с конкретными входными и выходными данными. Данные проходят через Dispatcher, центральный хаб, который в свою очередь управляет всеми данными.

Dispatcher действует как реестр с зарегистрированными callback-ами, на которые отвечают Store. Store будут давать изменения, которые выбраны Controller-Views.

Это происходит, когда View распространяется в системе:



1. Какие методы используются для подключения react-компонента к redux-стору?
2. Как можно сгенерировать redux-экшн и вызвать его из react-компонента? Приведите пример. Что такое action creator?
3. Для чего используется библиотека redux-thunk. Опишите принцип действия и приведите пример асинхронного action-creator’а.
4. Опишите, что такое reducer и приведите пример использования reducer’а. Как можно скомбинировать несколько редюсеров для получения единого стора? Как обращаться к полям общего стора, составленного из нескольких редюсеров?