# Инструменты создания приложения

# HTML

Язык гипертекстовой разметки (Hypertext Markup Language, HTML) - это код, который используется для структурирования веб-содержимого и придания ему смысла и цели. Например, является ли мое содержимое набором абзацев, либо списком маркированных пунктов? Нужно ли вставить изображения на мою страницу? Есть ли у меня таблица данных?

HTML не является языком программирования; это язык разметки, и используется, чтобы сообщать браузеру, как отображать веб-страницы, которые посещают пользователи. Он может быть сложным или простым, в зависимости от дизайна сайта. HTML состоит из ряда элементов, которые используется, чтобы вкладывать или оборачивать различные части контента, чтобы заставить контент отображаться или действовать определенным образом. Ограждающие теги могут сделать слово или изображение ссылкой на что-то еще, могут сделать слова курсивом, сделать шрифт больше или меньше и так далее. Например, возьмем следующую строку контента (Рис. 1).

Рис. 1 Конструкция элемента html

* Открывающий тег (Opening tag): Состоит из имени элемента (в данном случае, «p»), заключенного в открывающие и закрывающие угловые скобки. Открывающий тег указывает, где элемент начинается или начинает действовать, в данном случае - где начинается абзац;
* Закрывающий тег (Closing tag): Это то же самое, что и открывающий тег, за исключением того, что он включает в себя косую черту перед именем элемента. Закрывающий элемент указывает, где элемент заканчивается, в данном случае - где заканчивается абзац. Отсутствие закрывающего тега является одной из наиболее распространенных ошибок начинающих и может приводить к странным результатам;
* Контент (Content): Это контент элемента, который в данном случае является просто текстом;
* Элемент(Element): Открывающий тег, закрывающий тег и контент вместе составляют элемент.

Элементы также могут иметь атрибуты (Рис. 2).

Рис. 2 Атрибут html

Атрибуты содержат дополнительную информацию об элементе, которую хочется показывать в фактическом контенте. В данном случае, class это имя атрибута, а editor-note это значение атрибута. Класс позволяет дать элементу идентификационное имя, которое может позже использоваться, чтобы обращаться к элементу с информацией о стиле и прочих вещах.

Атрибут всегда должен иметь:

* Пробел между ним и именем элемента (или предыдущим атрибутом, если элемент уже имеет один или несколько атрибутов);
* Имя атрибута, за которым следует знак равенства;
* Значение атрибута, заключенное с двух сторон в кавычки.

Также можно располагать элементы внутри других элементов - это называется вложением. Если хочется заявить, что кошка очень раздражена, можно заключить слово «очень» в элемент <strong>, который указывает, что слово должно быть сильно акцентированно:

«<p>Моя кошка <strong>очень</strong> раздражена.</p>»

Однако, убедитесь, что элементы правильно вложены: в примере выше был открыт первым элемент <p>, затем элемент <strong>, потом должны закрыться сначала элемент <strong>, затем <p>. Приведенное ниже неверно:

«<p>Моя кошка <strong>очень раздражена.</p></strong>»

Некоторые элементы не имеют контента, и называются пустыми элементами. Возьмем элемент <img>, который уже имеется в нашем HTML:

«<img src="images/firefox-icon.png" alt="Моё тестовое изображение">»

Элемент содержит два атрибута, но не имеет закрывающего тега </img>, и никакого внутреннего контента. Это потому, что элемент изображения не оборачивает контент для влияния на него. Его целью является вставка изображения в HTML страницу в нужном месте.

Теперь посмотрим, как отдельные элементы объединяются в целую HTML страницу.

«<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Моя тестовая страница</title>

</head>

<body>

<img src="images/firefox-icon.png" alt="Моё тестовое изображение">

</body>

</html>»

Здесь находится:

* «<!DOCTYPE html>» - доктайп. В прошлом, когда HTML был молод (около 1991/1992), доктайпы должны были выступать в качестве ссылки на набор правил, которым HTML страница должна была следовать, чтобы считаться хорошим HTML, что могло означать автоматическую проверку ошибок и другие полезные вещи. Однако в наши дни, никто не заботится об этом, и они на самом деле просто исторический артефакт, который должен быть включен для того, чтобы все работало правильно;
* «<html></html>» - элемент <html>. Этот элемент оборачивает весь контент на всей странице, и иногда известен как корневой элемент;
* «<head></head>» - элемент <head>. Этот элемент выступает в качестве контейнера для всего, что необходимо включить в HTML страницу, но не являющегося контентом, который показывается пользователям страницы. К ним относятся такие вещи, как ключевые слова и описание страницы, которые будут появляться в результатах поиска, CSS стили контента, кодировка и многое другое;
* «<body></body>» - элемент <body>. В нем содержится весь контент, который показывается пользователям, когда они посещают страницу, будь то текст, изображения, видео, игры, проигрываемые аудиодорожки или что-то еще;
* «<meta charset="utf-8">» - этот элемент устанавливает UTF-8 кодировку документа, которая включает в себя большинство символов из всех известных человечеству языков. По сути, теперь документ может обрабатывать любой текстовый контент, который в него вложен. Нет причин не устанавливать её, так как это может помочь избежать некоторых проблем;
* «<title></title>» - элемент <title>. Этот элемент устанавливает заголовок для вашей страницы, который является названием, появляющимся на вкладке браузера загружаемой страницы, и используется для описания страницы, когда происходит добавление ее в закладки/избранное.

# CSS

Как и HTML, CSS на самом деле не является языком программирования. Это не язык разметки - это язык таблицы стилей. Это означает, что он позволяет применять стили выборочно к элементам в документах HTML. Например, чтобы выбрать все элементы абзаца на HTML странице и изменить текст внутри них с черного на красный (Рис. 3).

«p {

color: red;

}»

Чтобы подключить css файл к html нужно прописать в <head> строчку:

«<link href="styles/style.css" rel="stylesheet" type="text/css">»

Рис. 3 Структура css

* + - Селектор (Selector). Имя HTML элемента в начале набора правил. Он выбирает элемент(ы) для применения стиля (в данном случае, элементы p). Для стилизации другого элемента, просто измените селектор;
    - Объявление (Declaration). Единственное правило, например, color: red; указывает, какие из свойств элемента стилизовать;
    - Свойства (Properties). Способы, которыми можно стилизовать определенный HTML элемент (в данном случае, color является свойством для элементов <p>). В CSS выбирается, какие свойства нужно затронуть в правиле;
* Значение свойства (Property value). Справа от свойства, после двоеточия, есть значение свойства, который выбирает одно из множества возможных значений для данного свойства (существует множество значений color, помимо red).

Важные части синтаксиса:

* Каждый набор правил (кроме селектора) должен быть обернут в фигурные скобки ({});
* В каждом объявлении необходимо использовать двоеточие (:), чтобы отделить свойство от его значений;
* В каждом наборе правил нужно использовать точку с запятой (;), чтобы отделить каждое объявление от следующего.
  1. JavaScript

JavaScript это кросс-платформенный, объектно-ориентированный, скриптовый язык. JavaScript - это маленький и легковесный язык; он спроектирован не для создания законченных, полноценных приложений, а для легкого встраивания в другие продукты и приложения, например, в веб браузеры. Внутри основного приложения, JavaScript может соединяться с объектами этого приложения и осуществлять программный контроль над ними.

Ядро JavaScript содержит набор базовых объектов, например, Array, Date и Math, и набор элементов языка: операторов, управляющих структур, и выражений. Ядро JavaScript может быть расширено для различных целей с помощью дополнений, например:

* Client-side JavaScript расширяет ядро языка предоставляя объекты управления браузером (Navigator или другой веб-браузер) и его Document Object Model (DOM). Например, клиентские расширения позволяют приложению размещать элементы в HTML-форме и реагировать на пользовательские события, такие как щелчок мыши, ввод данных в форму и навигация по страницам;
* Server-side JavaScript расширяет ядро языка предоставляя объекты для запуска JavaScript на сервере. Например, server-side расширения позволяют приложению соединяться с реляционной базой данных, обеспечивать непрерывность информации между вызовами приложения, или выполнять операции над файлами на сервере.

Используя функции JavaScript LiveConnect, можно позволить коду на Java и JavaScript общаться между собой. Можно создать экземпляр объекта Java из JavaScript и получить доступ к его публичным методам и свойствам. Из Java, можно получить доступ к объектам, свойствам и методам JavaScript.

Впервые JavaScript был использован в браузерах Netscape.

JavaScript и Java схожи в некоторых отношениях, но принципиально отличаются в других. Язык JavaScript напоминает Java, но не имеет статической типизации и строгой проверки типов. В основном, JavaScript следует большей части синтаксиса Java в выражениях, именованиях и основного потока управления конструкциями, что стало причиной, почему он был переименован из LiveScript в JavaScript.

В отличии от системы компилированных классов в Java, построенной на объявлениях, JavaScript поддерживает систему исполнения, основанную на небольшом количестве типов данных, представляющих числовые, логические и строковые значения. JavaScript обладает моделью объектов на базе прототипов вместо более общей модели объектов на базе классов. Модель объектов на базе прототипов делает возможным динамическое наследование; то есть, то, что унаследовано, может различаться для отдельных объектов. JavaScript также поддерживает функции без каких-либо специальных декларативных требований. Функции могут быть свойствами объектов, выполняться как слабо типизированные методы.

По сравнению с Java, JavaScript - это язык с очень свободной формой языка. Вам не надо объявлять переменные, классы или методы. Не стоит беспокоиться, на счет того: является ли метод public, private или protected, нет необходимости реализовывать интерфейсы. Переменные, параметры и возвращаемые функциями типы не являются явно типизированными.

Java - язык программирования на основе классов, предназначенный для быстрого выполнения и безопасности типов. Безопасность типов означает, например, что нельзя взять тип Java integer и привести его к типу object reference или получить доступ к закрытой памяти изменяя байт-код Java. Ориентированная на классы модель Java означает, что программы состоят исключительно из классов и их методов. Наследование классов и строгая типизация в Java обычно требуют тесно связанную иерархию объектов. Эти требования делают программирование на Java более сложным чем на JavaScript.

По духу JavaScript происходит от нескольких небольших, динамически типизированных языков программирования, таких как HyperTalk и dBASE. Эти скриптовые языки предлагают инструменты программирования для гораздо более широкой аудитории благодаря более простому синтаксису, специальной встроенной функциональности и минимальным требованиям для создания объектов.

* 1. Electron

Создание программ под ПК стало возможным, после появления библиотек подобных Electron JS.

Приложения в вебе становятся всё мощнее, здесь прогресс на лицо. Тем не менее значительную долю разработки занимают стандартные приложения, которые имеют полный доступ к физическому оборудованию ПК. Уже сегодня есть возможность объединить обе технологии и написать десктопное приложение на известных языках веб-программирования, вроде HTML, JS и Node.js. Это всё можно поместить в исполняемый файл, который можно использовать на Mac OS X, Windows, Linux.

Приложение на Electron – это обычная Node.js программа, поэтому ей нужно добавить npm. Благо, это выполняется предельно легко.

Следует запустить терминал и находясь в каталоге целевого проекта выполнить команду:

npm install

В результате появится папка с названием node\_modules, в которой установлены все нужные зависимости для программы. Дальше стоит ввести ещё одну команду.

npm start

После неё приложение запустится в новом окне. Нужно заметить, что в нём будет исключительно верхнее меню.

Несложно заметить, что приложение включается крайне неудобно для рядового пользователя. Это лишь один из способов запуска, который скорее подходит для разработчика, чем пользователя. После упаковки программы, пользователь сможет включить приложение стандартным способом – дважды кликнуть по ярлыку.

Разработка программы - это создание главного JS файла, а также HTML и CSS файлов, содержащих все стили и разметку для страниц программы.

Существует ещё один важный момент, который помогает достичь целевого пользователя. Вам нужно запереть всё содержимое в исполняемый файл, как раз его и можно включить двойным кликом. Важно создать уникальный дистрибутив под каждую ОС: Windows, OS X, Linux. Как раз в этом и пригодится Electron Packager.

Здесь следует уделить внимание тому, что в готовый файл также добавятся ваши ресурсы, это обусловлено платформой Node JS, и обрезанная копия webkit веб-обозревателя. На выходе должен получиться файл весом около 50 Мб. Это весьма большой вес для обычного приложения в несколько строк. Вопрос с весом программы теряет актуальность при разработке крупных приложений со сложными алгоритмами работы.

Главное из значимых отличий от NW JS сводится к тому, что в NW.js входной файл - HTML, в то время как в Electron - JavaScript-файл. Таким образом Electron дарит больше возможностей по контролю. На его основе можно создать приложение с несколькими окнами, и настроить перенос данных между ними.

* 1. React

React - это JavaScript-библиотека для создания пользовательских интерфейсов. Обратите внимание, что это именно библиотека, а не фреймворк. React часто называют фреймворком, но это ошибка. Во-первых, его использование ни к чему вас не обязывает, не формирует «фрейм» проекта. Во-вторых, React выполняет единственную задачу: показывает на странице компонент интерфейса, синхронизируя его с данными приложения, и только этой библиотеки в общем случае недостаточно для того, чтобы полностью реализовать проект.

Вскоре после появления React и подобные ему решения (Vue.js, Svelte) практически захватили мир фронтенда: потому что они помогают решать проблемы, основываясь на идее декларативного программирования, а не на императивном подходе.

Декларативный подход состоит в описании конечного результата (что мы хотим получить).

При императивном подходе описываются конкретные шаги для достижения конечного результата (как мы хотим что-то получить).

Оказалось, что декларативный подход отлично подходит для создания интерфейсов, и он прижился в сообществе. Этот подход работает не только в вебе: сравнительно недавно компания Apple представила фреймворк SwiftUI, основанный на тех же принципах.

Возможности данной библиотеки:

* С её помощью можно построить интерфейс из отдельных компонентов, которые легко поддерживать;
* Она добавляет удобный слой абстракции, избавляя от необходимости работать с DOM напрямую;
* React — это уже не новая библиотека, за ней стоит крупная компания Facebook и большое сообщество разработчиков. Поэтому она отлично протестирована, регулярно поддерживается и стабильно обновляется, а переход к новым версиям проходит максимально гладко;
* Благодаря сообществу у React хорошо проработанная документация и большой опыт, накопленный в статьях, курсах и конференциях. Это значительно облегчает не только изучение библиотеки новичками, но и поиск ответов на всевозможные вопросы в процессе работы;
* На Гитхабе можно найти готовые React-компоненты почти на все случаи жизни. А если их нет, но есть нужные независимые библиотеки, то можно поискать интеграцию либо сделать её самостоятельно;
* В React-сообществе со временем сформировались определённые подходы и соглашения по организации проекта, кода, решению частых проблем. Для разработчиков это значит, что нужно тратить меньше времени на обсуждение некоторых договорённостей — можно использовать готовые и уже устоявшиеся;
* Хотя в проекте, скорее всего, будет налажен процесс сборки с помощью webpack, parcel, rollup или другого упаковщика, стоит иметь в виду, что это необязательно для использования React. Работая с этой библиотекой, вы пишете на чистом JS, и никакие диалекты HTML, CSS или JS изучать не нужно. Конечно, с React почти всегда используется JSX, но и это тоже [опционально](https://ru.reactjs.org/docs/react-without-jsx.html);
* React — это проект с открытым исходным кодом. Благодаря этому его можно безопасно использовать даже в коммерческих приложениях.

Однако при использовании React есть**особенности, которые важно учитывать:**

* React увеличивает размер приложения, которое нужно загрузить пользователям (~40 kB для пакетов React и React-dom).
* Загруженный код нужно выполнить в браузере: для пользователя это означает, что приложение будет запускаться медленнее. Подробнее про это можно почитать в [материале](https://v8.dev/blog/cost-of-javascript-2019) «The cost of JavaScript in 2019» от команды V8 (перевод: «[Цена JavaScript в 2019 году](https://habr.com/ru/company/ruvds/blog/459296/)»).
* Вместе с виртуальным DOM возникают свои накладные расходы: во-первых, по времени выполнения (сравнение виртуальных деревьев происходит не моментально); во-вторых, по памяти, ведь виртуальные деревья нужно где-то хранить и не в единственном экземпляре. С увеличением количества элементов на странице эти расходы растут, что может стать реальной проблемой на мобильных устройствах. Поэтому при изучении React важно уделить внимание способам оптимизации рендеринга приложения – необходимые инструменты для этого есть в самой библиотеке.
* Средний порог входа в React. Чтобы начать разрабатывать на React, программист должен не только изучить саму библиотеку, но и привыкнуть к используемой парадигме. Про это есть хорошая [статья](https://ru.reactjs.org/docs/thinking-in-react.html) «Философия React» на официальном сайте.
  1. Node js

Подружить две эти технологии, react и electron, можно несколькими способами, например, создать проект CRA (create react app) и установить туда с помощью пакетного модуля npm или yarn «голый» electron. Или же можно пойти от обратного и создать проект electron forge и туда установить react.

Но в любом случае разработка не была бы возможна без технологии node js, который позволяет вытащить js из браузера.

Node.js это программный пакет, основанный на JavaScript-движке Google V8. Он включает уровень абстракции платформы – библиотеку libuv, и базовую библиотеку, которая сама главным образом написана на JavaScript. Кроме того, стоит отметить, что создатель Node.js Райан Даль намеревался создавать сайты, работающие в реальном времени с технологией push, вдохновившись такими приложениями, как Gmail. В Node.js он предоставил разработчикам инструмент для работы с парадигмой неблокирующего событийно-ориентированного ввода/вывода.

Основная идея Node.js заключается в использовании неблокирующего событийно-ориентированного ввода/вывода, чтобы оставаться легковесным и эффективным при работе с приложениями, обрабатывающими большие объемы данных в реальном времени и работающими на распределенных устройствах.

При разговоре о Node.js очень важно не упустить одну важную деталь. Это встроенная поддержка управления пакетами с помощью инструмента NPM, который по умолчанию идет в любой установке Node.js. Принцип модулей NPM схож с принципом Ruby Gems – это набор повторно используемых компонентов, находящихся в открытом доступе. Их можно легко установить из сетевого репозитория, с поддержкой управления версиями и зависимостями.

То есть если подвести итоги, то node js позволяет с помощью библиотеки electron js, создать настольное приложение, интерфейс которому будет рисовать react js.