**Лекция 1. Введение в веб-разработку. Основы HTML, CSS, JavaScript, React.js, Django Framework**

**Содержание**

[Введение 2](#_Toc139571379)

[Основные этапы веб-разработки 3](#_Toc139571380)

[Редактор кода 6](#_Toc139571381)

[Git 7](#_Toc139571382)

[HTML 13](#_Toc139571383)

[CSS 21](#_Toc139571384)

[Bootstrap Framework 31](#_Toc139571385)

[Язык программирования javaScript 34](#_Toc139571386)

[Библиотека React.js. Создание реактивных интерфейсов. 40](#_Toc139571387)

[Основы Pythion 52](#_Toc139571388)

[Django Framework 61](#_Toc139571389)

[API 70](#_Toc139571390)

[REST API 71](#_Toc139571391)

# Введение

Веб-разработка одна из современных, перспективных и карьерных профессий в IT отрасли. Технологии веб разработки в руках специалиста позволяют создавать сайты, приложения, работать над другими задачами в сфере бизнеса и маркетинга. Эти знания будут полезны и нужны руководителям и специалистам всех уровней.

Веб-разработчики, веб-дизайнеры при обсуждении проектов очень часто уходят в технические разговоры и их становится очень сложно понять людям из других сфер бизнеса, особенно владельцам бизнеса и маркетологам.

При этом, заказчику проекта важно понимать некоторые технические аспекты веб-дизайна и услуг разработки. Хотя бы для того, чтобы они могли провести информированные обсуждения со своими группами разработчиков или поставщиками. Это может дать им концептуальное представление о том, как будет построен проект с технической точки зрения.

Сегодня мы рассмотрим следующие моменты:

- основные этапы разработки проекта;

- инструменты для разработки клиентской части веб-приложения – Frontend (html, css, JavaScript);

- система контроля версий Git, основные команды;

- Frontend-фреймворк Bootstrap;

- библиотека для создания реактивных интерфейсов React.js;

- стек Python+Django и REST API.

Для того чтобы разрабатывать веб-приложения или веб-сайты, вам необходимо знать несколько веб-технологий, без которых веб-разработка невозможна. Любой веб-разработчик должен знать следующие технологии: HTML, CSS, JavaScript, один из фреймворков для разработки интерфейсов (React.js , Angular.js или Vue.js ), язык SQL для работы с базами данных, серверный язык программирования, а также IDE, в которой используется этот язык. Конечно, это далеко не полный список. Однако эти технологии обязательны для изучения хотя бы на базовом уровне.

Я дам вам ссылки, по которым вы сможете узнать более подробно, как работать с этими инструментами.

# Основные этапы веб-разработки

В современную цифровую эпоху веб-разработка стала неотъемлемой частью нашей жизни, позволяя предприятиям, организациям и частным лицам обеспечить присутствие в сети. Понимание этапов веб-разработки имеет решающее значение для всех, кто заинтересован в создании веб-сайта, приложения или в продолжении карьеры в этой области. Итак, давайте углубимся в основные этапы веб-разработки и изучим их содержание.

**Стадия планирования:** Первым этапом веб-разработки является стадия планирования. На этом этапе важно определить назначение, задачи и целевую аудиторию веб-сайта. Ключевые мероприятия на этом этапе включают:

* Определение целей создания веб-сайта и желаемых результатов.
* Проведение маркетинговых исследований и анализа конкурентов.
* Определение целевой аудитории и ее потребностей.
* Разработка карты сайта или структуры для описания страниц веб-сайта и навигации по ним.
* Создание графика и бюджета для проекта.

**Стадия разработки технического задания:** Этот этап включает в себя создание всеобъемлющего документа с техническим заданием, в котором излагаются конкретные требования, функциональные возможности и технические детали проекта. Техническое задание служит основой для команды разработчиков и гарантирует, что все участники, вовлеченные в проект, согласовали между собой все задачи и понимают масштаб проекта. Ключевые аспекты этапа разработки технического задания включают:

* + *Сбор требований:* сотрудничество с заинтересованными сторонами, клиентами и командой разработчиков для сбора всех необходимых требований к веб-сайту, понимание желаемых характеристик, функциональности и любых конкретных технических ограничений или предпочтений.
  + *Определение технических спецификаций:* Основываясь на требованиях, этап технического задания фокусируется на уточнении технических деталей проекта. Здесь происходит выбор подходящих языков программирования, фреймворков и инструментов, которые будут использоваться, определение структуры базы данных, описание мер безопасности и определение требований к интеграции с другими системами, если это применимо.
  + *Создание прототипов:* В рамках разработки технического задания могут быть созданы прототипы с низкой точностью, чтобы визуализировать макет и UX/UI веб-сайта. Прототипы помогают подтвердить правильность технического подхода и собрать отзывы заинтересованных сторон.
  + *Документирование технических спецификаций:* Документ с техническим заданием должен содержать подробную информацию об архитектуре веб-сайта, структурах данных, ролях пользователей и разрешениях, сторонних интеграциях, API-интерфейсах и любых других технических соображениях. В нем также должны быть указаны любые конкретные требования к производительности или масштабируемости.
  + *Рассмотрение и утверждение:* Документ технического задания должен быть рассмотрен заинтересованными сторонами проекта, включая клиентов, проектировщиков, разработчиков и персонал по обеспечению качества, чтобы убедиться, что он точно отражает требования проекта. Любые необходимые изменения или уточнения должны быть внесены до перехода к этапам проектирования и разработки.

Этап разработки технического задания имеет решающее значение для установления четкого понимания технических аспектов проекта и содействия эффективной коммуникации и сотрудничеству между командой разработчиков. Он служит ориентиром на всех последующих этапах проектирования, разработки и тестирования, гарантируя соответствие конечного веб-сайта целям проекта и техническим требованиям.

**Стадия проектирования:** Как только техническое задание создано, мы переходим к стадии проектирования. На этом этапе основное внимание уделяется визуальным аспектам и пользовательскому интерфейсу веб-сайта. Основные элементы этапа проектирования включают в себя:

* + Создание каркасов или низкокачественных макетов для визуализации макета и структуры веб-сайта.
  + Разработка элементов пользовательского интерфейса (UI), таких как кнопки, формы и меню.
  + Выбор подходящих цветовых схем, типографики и изображений.
  + Обеспечение адаптивности дизайна веб-сайта и его совместимости с различными устройствами и размерами экранов.

**Стадия разработки:** На стадии разработки веб-сайт начинает обретать форму. Это включает в себя превращение дизайнерских концепций в полнофункциональный веб-сайт. Ключевые аспекты этапа разработки включают:

* + Написание HTML, CSS и JavaScript кода для создания структуры, стилей и интерактивности веб-сайта.
  + Интеграция с системой управления контентом (CMS) или платформой электронной коммерции, если требуется.
  + Внедрение систем баз данных для вывода динамического контента и хранения данных.
  + Оптимизация производительности веб-сайта, включая скорость загрузки и отзывчивость.
  + Проведение тщательного тестирования и отладки для обеспечения функциональности и кроссбраузерной совместимости.

**Стадия создания контента:** Стадия создания контента часто совпадает со стадией разработки. Она фокусируется на создании привлекательного и релевантного контента для веб-сайта. Включает в себя:

* + Создание текстового контента для каждой страницы.
  + Создание или выбор подходящих визуальных элементов, таких как изображения, видео и инфографика.
  + Оптимизация контента для поисковых систем (SEO) путем включения релевантных ключевых слов и метаданных.
  + Обеспечение доступности контента и соответствия рекомендациям по обеспечению доступности в Интернете (WCAG).

**Этап развертывания:** Как только веб-сайт разработан и его содержимое готово, наступает время для развертывания. Этап развертывания включает в себя:

* + Настройка хостинга и регистрация доменного имени.
  + Загрузка всех файлов веб-сайта на сервер.
  + Настройка параметров сервера и обеспечение надлежащих мер безопасности.
  + Проведение окончательных проверок, чтобы убедиться в корректной работе всех функций веб-сайта.
  + Запуск веб-сайта для публичного доступа.

**Этап поддержки и обновлений:** После запуска веб-сайта работа на ним не заканчивается. Регулярное техническое обслуживание и обновления имеют решающее значение для его успеха. Ключевые мероприятия на этом этапе включают:

* + Мониторинг производительности веб-сайта, времени безотказной работы и безопасности.
  + Внедрение обновлений в используемые CMS, плагины и фреймворки.
  + Анализ аналитики веб-сайта для получения информации о поведении пользователей и внесение улучшений.
  + Добавление нового контента, функций или разделов по мере необходимости.
  + Создание резервных копий и проверок безопасности для защиты данных веб-сайта.

В заключение отметим, что веб-разработка включает в себя несколько этапов, каждый из которых имеет свое уникальное содержание и цели. Понимая эти этапы и их содержание, вы будете лучше подготовлены к планированию, дизайну, разработке и сопровождению успешного проекта. Помните, что веб-разработка - это непрерывный процесс. Для создания привлекательных, удобных в использовании веб-сайтов и приложений необходимо быть в курсе новейших технологий и тенденций.

# Редактор кода

Далее мы рассмотрим редакторы кода.

Для написания кода разметки можно использовать различные редакторы кода: NotePad++, SubLime Text, MS Visual Studio Code. Мы будем использовать последний из представленных.

Это бесплатный редактор от Microsoft. Достаточно «легкий», имеется возможность установки различных плагинов для ускорения работы.

Для его установки переходим по следующему адресу: <https://code.visualstudio.com/>. Скачиваем дистрибутив и устанавливаем на компьютер.

После установки редактора кода, необходимо его настроить для работы. Перейдем к установке некоторых плагинов.

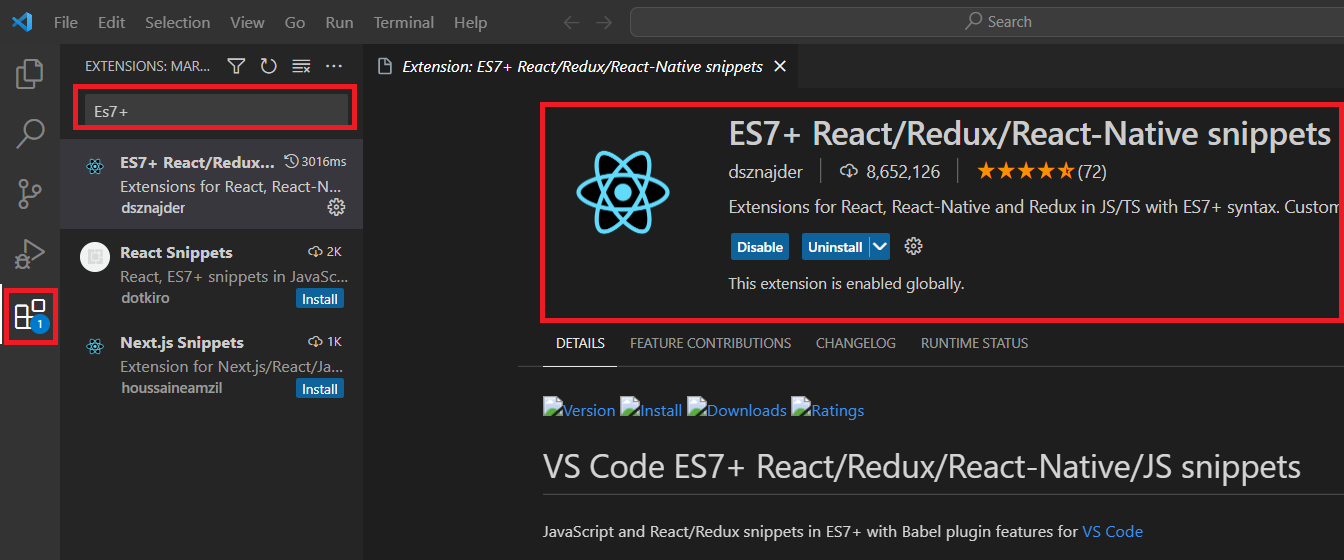
Я часто использую следующие плагины:

1. Auto Close Tag.
2. Auto Complete Tag.
3. Auto Rename Tag.
4. ES7+ React/Redux/React-Native/JS snippets.
5. C# for Visual Studio.
6. HTML CSS Support.
7. Intellisense for CSS class names in HTML.
8. Live Server.

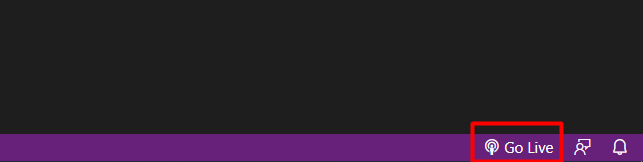
Для синхронного обновления веб-страницы в браузере, используется плагин Live Server.

Для запуска плагина делаем следующее:

1. Перейдите на вкладку «Расширения (Extansions)». В строке поиска введите название необходимого плагина. Кроме того, нажмите кнопку "Установить".



1. Нажимаем на нижней панели редактора (с правой стороны) на иконку «Go Live».



1. «VS Code» запустит встроенный веб-сервер и откроет html-код в браузере по адресу <http://127.0.0.1:5500/index.html>.

Теперь, если на странице будут внесены какие-либо правки, и она будет сохранена, то все изменения будут отображены в браузере автоматически.

# Git

Не менее важным инструментом для работы программиста в команде является система контроля версий.

**Git** – широко используемая система контроля версий, которая позволяет эффективно управлять своими проектами. Независимо от того, являетесь ли вы разработчиком, дизайнером или создателем контента, Git может помочь в отслеживании изменений, сотрудничать с другими пользователями и вести историю вашей работы. Далее рассмотрим, что такое Git, его преимущества и рассмотрим некоторые основные команды, которые помогут начать работу с этим инструментом.

**Понимание Git: Система контроля версий**

Системы контроля версий - это инструменты, используемые для отслеживания изменений в файлах и проектах с течением времени и управления ими. Они обеспечивают совместную работу, предоставляют возможности резервного копирования и восстановления, а также облегчают проведение экспериментов.

**Что такое Git?**

Git - это распределенная система контроля версий, предназначенная для быстрой и эффективной работы со всеми, от небольших до крупных проектов. Он был создан Линусом Торвальдсом (Linus Torvalds) в 2005 году для управления разработкой ядра Linux.

**Преимущества Git:**

1. Распределенный характер: у каждого пользователя есть полная копия проекта, что позволяет работать в автономном режиме и легко разветвляться.
2. Скорость и производительность: Git разработан таким образом, чтобы быть быстрым даже при работе с большими репозиториями.
3. Целостность данных: Git использует криптографическую хэш-функцию для обеспечения целостности файлов.
4. Ветвление и слияние: Git предоставляет мощные возможности ветвления и слияния, обеспечивая параллельную разработку и легкую совместную работу.
5. Отслеживание изменений: Git ведет полную историю всех изменений, внесенных в проект, позволяя вам отслеживать, кто внес изменения и почему.

**Git** позволяет отслеживать изменений в любом наборе файлов. Обычно Git используется для координации работы программистов, которые совместно разрабатывают исходный код во время разработки программного обеспечения.

Для начала работы, прежде всего вы должны создать учетную запись GitHub на официальном сайте и установить Git на свой локальный компьютер. Чтобы загрузить Git на свой локальный компьютер, вам следует воспользоваться этой ссылкой: <https://github.com/git-guides/install-git>.

**Установка Git**

Чтобы установить Git на свой компьютер, выполните следующие действия:

1. Windows:
   * Загрузите установщик Git с официального веб-сайта Git: https://git-scm.com/downloads .
   * Запустите загруженный установщик и следуйте инструкциям мастера установки.
   * Выберите желаемые параметры установки и завершите процесс установки.
2. macOS:
   * Git обычно предустановлен на macOS. Откройте терминал и введите команду "git --version", чтобы проверить, установлен ли Git.
   * Если он не установлен, вы можете загрузить установщик для macOS с: https://git-scm.com/downloads .
   * Запустите загруженный установщик и следуйте инструкциям мастера установки.
3. Linux:
   * Git, возможно, уже доступен в менеджере пакетов вашего дистрибутива Linux. Вы можете использовать менеджер пакетов для установки Git.
   * Например, в Ubuntu вы можете использовать команду: "sudo apt-get install git".
   * Если он недоступен в вашем менеджере пакетов или вам нужна последняя версия, вы можете загрузить его с: https://git-scm.com/downloads .
   * Следуйте инструкциям, приведенным на веб-сайте, чтобы установить Git в ваш дистрибутив Linux.

Работа с любой программой всегда начинается с ее настроек. Git можно настроить один раз и изменять только по мере необходимости:

Установка имени пользователя, от которого будут отправлены изменения.

*git config --global user.name "User Name"*

Установка адреса электронной почты. Пожалуйста, обратите внимание, что адрес должен совпадать с тем, на который зарегистрирована учетная запись GitHub.

*git config --global user.email " mail@gmail.com "*

Установка текстового редактора, который будет открывать файлы для разрешения конфликтов:

*git config --global core.editor "editor"*

С помощью команды

*git config --list*

Вы можете увидеть список всех установленных настроек.

**Клонирование репозитория**

Допустим, руководитель вашей команды поручил вам новый проект. Этот проект находится в удаленном репозитории (в нашем случае это будет GitHub). Нам нужно скопировать этот репозиторий, внести в него некоторые изменения и загрузить его обратно в облако. Давайте посмотрим, как все это можно сделать.

Чтобы клонировать репозиторий, введите команду:

*git clone*

и дайте его адрес. Репозиторий клонируется в текущую выбранную папку в консоли.

**Работа с изменениями**

Любая работа с изменениями начинается с получения последней версии удаленного репозитория. Вы можете получить последнюю версию с помощью команды:

*git pull [<options>] [<repository> [<refspec>…]]*

Будьте осторожны: вызов этой команды приведет к удалению всех незафиксированных изменений.

После внесения любых изменений в проект вы можете просмотреть состояние файлов с помощью команды:

*git status*

В нем будут показаны файлы, которые были изменены, удалены, а также новые, которые необходимо добавить.

Чтобы добавить отслеживание для новых файлов, вы должны использовать команду:

*git add <filename> <filename>*

чтобы добавить несколько файлов по имени.

Если вам нужно добавить все файлы, вы можете использовать команду:

*git add .*

который добавляет отслеживание для всех новых файлов из текущего каталога. И команда:

*git add -A*

добавляет также удаленные файлы не только из текущего каталога, но и из всего локального хранилища.

В дополнение к добавлению файлов, их также необходимо удалить. Для этого есть команда:

*git rm <filename> <filename>*

который удаляет файлы по их названию.

После того как все новые файлы будут добавлены, а старые удалены, вы можете зафиксировать изменения. Фиксация изменений, или commit, очень важна, потому что до тех пор, пока эта команда не будет выполнена, ваши локальные изменения нигде не будут записаны. Чтобы добавить фиксацию, вы должны ввести команду:

*git commit -m "comment to commit"*

Стоит отметить, что необходимо правильно разбивать изменения на коммиты и давать полные комментарии к коммитам.

Если вы внесли изменения и хотите быстро отменить их, то воспользуйтесь командой:

*git reset*

который отменяет все незафиксированные изменения.

По умолчанию эта команда удаляет только из индекса. И команда:

*git reset --hard*

безвозвратно удаляет незафиксированные текущие изменения из локального репозитория и из индекса.

Поскольку все вышеперечисленные действия выполняются в локальной копии репозитория, эта копия должна быть отправлена на сервер, чтобы другие участники процесса могли получить последнюю версию. Для этого есть команда

*git push*

который отправляет все зафиксированные изменения в удаленный репозиторий.

**Работа с файлами**

С помощью Git вы можете создавать ветки и перемещаться по ним.

Команда:

*git checkout -b "branch-name"*

создаст ветку с указанным именем и автоматически переключится на нее.

После создания ветвь может быть отправлена на сервер с помощью команды:

*git push origin "branch-name"*

Аналогично, вы можете перенести ветку из удаленного репозитория на свой компьютер с помощью команды:

*git checkout origin/"branch-name" -b "branch-name"*

Чтобы не хранить имена ветвей в памяти и не искать их названия, существуют две специальные команды, которые позволяют просматривать все существующие ветви локального репозитория:

*git branch*

или все существующие ветви удаленного репозитория:

*git branch -r*

Вы можете переключиться на любую локальную ветку с помощью команды:

*git checkout "branch-name"*

После работы над проектом в репозитории могут остаться различные ненужные, неотслеживаемые файлы и прочий мусор. Чтобы удалить все ненужные файлы, используйте команду:

*git clean -f -d*

Вот несколько дополнительных ресурсов для изучения Git и процесса его установки на ваш компьютер:

1. Официальная документация по Git: Официальная документация по Git является отличным ресурсом для изучения Git с нуля. Он охватывает все, от базовых до продвинутых тем, и содержит подробные объяснения концепций и команд Git. Вы можете найти его по адресу: <https://git-scm.com/doc>.

2. Книга Pro Git: "Pro Git" - это всеобъемлющая книга Скотта Чакона и Бена Страуба, которая подробно описывает Git. Она начинается с основ и переходит к более сложным темам, таким как ветвление, слияние и перебазирование. Книга доступна бесплатно онлайн по адресу: <https://git-scm.com/book/en/v2>.

3. Учебные пособия по Git на Atlassian Bitbucket: Bitbucket предоставляет серию пошаговых руководств, которые охватывают основы Git, ветвление, слияние и сотрудничество с другими. Эти учебные пособия удобны для начинающих и содержат практические примеры. Вы можете получить к ним доступ по адресу: <https://www.atlassian.com/git/tutorials>.

4. Учебная лаборатория GitHub: Учебная лаборатория GitHub предлагает интерактивные практические курсы по изучению Git и GitHub. Это обеспечивает управляемый процесс обучения с использованием реальных сценариев и упражнений. Вы можете получить доступ к курсам по адресу: <https://lab.github.com/>.

5. GitKraken Git Client: GitKraken - популярный Git-клиент, который предлагает визуально привлекательный и удобный интерфейс. На их веб-сайте представлены учебные пособия и ресурсы, которые помогут вам освоить Git и начать работу с их клиентом: <https://www.gitkraken.com/learn-git>.

Не забудьте ознакомиться с официальной документацией или конкретными ресурсами для получения подробных инструкций по установке Git на основе вашей операционной системы. Эти ресурсы и инструкции по установке помогут вам начать работу с Git.

# HTML

HTML - это стандартный язык, используемый для создания и структурирования веб-страниц во Всемирной паутине. Он составляет основу каждого веб-сайта, который вы посещаете, позволяя нам представлять информацию структурированным и интерактивным образом.

HTML предоставляет набор тегов (или элементов), которые определяют структуру и содержимое веб-страницы. Эти теги заключены в угловые скобки (< >) и используются для описания различных частей документа. Используя комбинацию этих тегов, мы можем форматировать текст, добавлять изображения, создавать ссылки, вставлять видео и многое другое. Затем браузер интерпретирует эти теги для отображения веб-страницы должным образом.

Давайте начнем с рассмотрения некоторых основных HTML-тегов и их использования:

1. Тег **<html>**: Этот тег используется для определения начала и конца HTML-документа. Он обтекает все остальные HTML-теги.
2. Тег **<head>**: Внутри тега <html> у нас есть тег <head>, который содержит метаданные о документе, такие как заголовок, кодировка символов и связанные таблицы стилей.
3. Тег **<body>**: Внутри тега <html> у нас есть тег <body>, который представляет основное содержимое веб-страницы. Все, что мы хотим отобразить пользователю, находится внутри этого тега.
4. Теги от **<h1>** до **<h6>**: Эти теги используются для заголовков разного размера, причем <h1> является самым большим, а <h6> - самым маленьким.
5. Тег **<p>**: Этот тег используется для определения абзацев. Текст, заключенный в теги <p>, будет отображаться в виде отдельного абзаца.

Некоторые распространенные варианты использования HTML:

1. *Создание персонального веб-сайта*: HTML обеспечивает основу для создания персональных веб-сайтов, на которых вы можете демонстрировать свои навыки, портфолио или вести блог. Используя HTML-теги, вы можете структурировать контент и добавлять различные элементы, чтобы сделать ваш сайт визуально привлекательным.
2. *Разработка онлайн-форм*: HTML позволяет создавать формы, которые собирают вводимые пользователем данные, такие как контактная информация или ответы на опрос. Элементы формы, такие как текстовые поля, флажки, переключатели и кнопки отправки, можно использовать для создания интерактивных форм.
3. *Разработка веб-сайтов электронной коммерции*: HTML играет решающую роль в разработке интернет-магазинов. Вы можете создавать списки товаров, корзины покупок и процессы оформления заказа, используя HTML-теги в сочетании с другими веб-технологиями, такими как CSS (каскадные таблицы стилей) и JavaScript.
4. *Встраивание мультимедиа*: HTML позволяет встраивать мультимедийные элементы, такие как изображения, видео и аудиофайлы, в веб-страницы. Таким образом, вы можете обогатить пользовательский опыт за счет плавного включения визуального и интерактивного контента.

Чтобы узнать больше о HTML и его различных тегах и атрибутах, существует несколько отличных онлайн-ресурсов. Вот несколько ссылок, которые помогут вам начать:

MDN Web Docs: HTML - <https://developer.mozilla.org/en-US/docs/Web/HTML>.

W3Schools HTML Tutorial - <https://www.w3schools.com/html/>.

HTML Dog: The Best-Practice Guide to XHTML and CSS - <http://www.htmldog.com/>.

Я призываю вас всех изучить эти ресурсы, поэкспериментировать с HTML-тегами и попрактиковаться в создании собственных веб-страниц.

Давайте углубимся в некоторые основные HTML-теги и поймем, как их можно использовать:

**<html>**: Этот тег служит корневым элементом HTML-документа и обтекает все остальные HTML-теги. Это указывает на то, что содержимое внутри является HTML-документом.

**<head>**: Этот тег содержит метаданные о документе, такие как заголовок, кодировка символов, связанные таблицы стилей и скрипты. Он не отображается на самой веб-странице, но предоставляет важную информацию браузеру и поисковым системам.

**<body>**: Этот тег представляет основное содержимое веб-страницы. Он инкапсулирует все видимые элементы, которые пользователи увидят при посещении страницы. Текст, изображения, ссылки, заголовки, абзацы и другие HTML-элементы размещаются внутри тегов <body>.

**<h1> - <h6>**: Эти теги используются для создания заголовков разного размера, где <h1> представляет самый большой заголовок, а <h6> - самый маленький. Например:

<h1>This is the largest heading</h1>

<h2>This is a subheading</h2>

<h3>Another subheading</h3>

**<p>**: Этот тег используется для определения абзацев текста. Он создает новую строку и добавляет интервал до и после содержимого. Например:

<p>This is a paragraph of text.</p>

**<a>**: Этот тег создает гиперссылку на другую веб-страницу или определенное местоположение на той же странице. Он обычно используется для навигации по меню, перехода на внешние веб-сайты или создания внутренних ссылок внутри документа. Например:

<a href="https://www.urfu.ru">Visit URFU website</a>

**<img>**: Этот тег используется для встраивания изображений на веб-страницу. Требуется атрибут src для указания источника изображения (URL или путь к файлу) и атрибут alt для предоставления альтернативного текста для обеспечения доступности. Например:

<img src="image.jpg" alt="A beautiful image">

**<ul> и <li>**: Эти теги используются для создания неупорядоченных списков. Тег **<ul>** представляет весь список, и каждый элемент списка определяется с помощью тега **<li>**. Например:

<ul>

    <li>Item 1</li>

    <li>Item 2</li>

    <li>Item 3</li>

</ul>

**<ol>** и **<li>**: Эти теги используются для создания упорядоченных списков. Подобно неупорядоченным спискам, тег **<ol>** представляет список, и каждый элемент определяется с помощью тега **<li>**. Разница в том, что упорядоченные списки пронумерованы. Например:

<ol>

    <li> First item </li>

    <li> Second item </li>

    <li> Third item </li>

</ol>

**<div>**: Этот тег является универсальным контейнером, используемым для группировки и стилизации содержимого. Он часто используется для верстки или для применения стилей CSS к определенному разделу веб-страницы. Например:

<div>

    <h2>Section Title</h2>

    <p>This is the content of the section.</p>

</div>

Это всего лишь несколько примеров основных HTML-тегов и их использования. HTML предоставляет широкий спектр тегов и атрибутов для структурирования и форматирования содержимого веб-страниц. Творчески комбинируя и размещая эти теги, вы можете создавать сложные и визуально привлекательные веб-страницы.

**Семантическая верстка**

Семантическая верстка в HTML относится к практике использования HTML-тегов, которые передают смысл и описывают структуру контента, а не просто внешний вид. Используя семантические теги, вы делаете свой HTML-код более осмысленным, доступным и дружественным к поисковым системам. Давайте рассмотрим несколько примеров элементов семантической компоновки:

**<header>**: Тег <header> представляет вводный контент или верхнюю часть веб-страницы. Обычно он включает в себя логотип веб-сайта, навигационное меню или заголовок страницы. Например:

<header>

    <h1>My Website</h1>

    <nav>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">About</a></li>

        <li><a href="#">Contact</a></li>

      </ul>

    </nav>

</header>

**<nav>**: Тег <nav> используется для определения раздела навигационных ссылок. Он представляет собой набор ссылок, которые позволяют пользователям перемещаться по веб-сайту или к различным разделам страницы. Например:

<nav>

    <ul>

      <li><a href="#">Home</a></li>

      <li><a href="#">About</a></li>

      <li><a href="#">Contact</a></li>

    </ul>

</nav>

**<main>**: Тег <main> представляет основную область содержимого веб-страницы. Обычно он содержит основное содержимое, уникальное для данной страницы. Его следует использовать только один раз на странице. Например:

<main>

    <h1>Welcome to My Website</h1>

    <p>Here is some important information.</p>

</main>

**<article>**: Тег <article> используется для определения самодостаточной, независимой части контента на веб-странице. Это может быть запись в блоге, новостная статья, сообщение на форуме или любой другой контент, который имеет смысл сам по себе. Например:

<article>

    <h2>Article Title</h2>

    <p>This is the content of the article.</p>

</article>

**<section>**: Тег <section> используется для разделения содержимого веб-страницы на различные разделы или группы. Это помогает упорядочить связанный контент. Например:

<section>

    <h2>About Me</h2>

    <p>Some information about me goes here.</p>

 </section>

 <section>

    <h2>My Skills</h2>

    <ul>

      <li>HTML</li>

      <li>CSS</li>

      <li>JavaScript</li>

    </ul>

</section>

**<aside>**: Тег <aside> представляет контент, который косвенно связан с основным контентом, но может рассматриваться отдельно от него. Он часто используется для боковых панелей, извлечения цитат или рекламы. Например:

<aside>

    <h3>Related Articles</h3>

    <ul>

      <li><a href="#">Article 1</a></li>

      <li><a href="#">Article 2</a></li>

      <li><a href="#">Article 3</a></li>

    </ul>

</aside>

Это примеры семантических HTML-тегов, которые помогают структурировать и описывать содержимое веб-страницы. Правильно используя эти теги, вы предоставляете дополнительный контекст, помогающий программам чтения с экрана, поисковым системам и другим устройствам понять назначение и иерархию вашего контента.

Семантический HTML не только улучшает доступность и оптимизацию в поисковых системах, но и делает ваш код более удобным в обслуживании и понятным для других разработчиков.

Вот пример кода для веб-страницы ("Pizza Order") с семантическим макетом:

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Pizza Order</title>

  <style>

    /\* Some basic styling for demonstration purposes \*/

    body {

      font-family: Arial, sans-serif;

      margin: 0;

      padding: 20px;

    }

    header {

      background-color: #f2f2f2;

      padding: 10px;

      text-align: center;

    }

    nav {

      margin-bottom: 20px;

    }

    main {

      margin-bottom: 20px;

    }

    section {

      padding: 10px;

      background-color: #f9f9f9;

      margin-bottom: 10px;

    }

    footer {

      background-color: #f2f2f2;

      padding: 10px;

      text-align: center;

    }

  </style>

</head>

<body>

  <header>

    <h1>Pizza Order</h1>

    <nav>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">Menu</a></li>

        <li><a href="#">Order</a></li>

        <li><a href="#">Contact</a></li>

      </ul>

    </nav>

  </header>

  <main>

    <section>

      <h2>Choose Your Pizza</h2>

      <p>Select your favorite pizza from our menu:</p>

      <ul>

        <li>Margherita</li>

        <li>Pepperoni</li>

        <li>Hawaiian</li>

        <li>Veggie Supreme</li>

      </ul>

    </section>

    <section>

      <h2>Customize Your Pizza</h2>

      <p>Customize your pizza with the following options:</p>

      <ul>

        <li>Size: <select>

            <option>Small</option>

            <option>Medium</option>

            <option>Large</option>

          </select></li>

        <li>Toppings: <input type="checkbox"> Pepperoni <input type="checkbox"> Mushrooms <input type="checkbox"> Olives</li>

        <li>Crust: <input type="radio" name="crust" value="thin"> Thin <input type="radio" name="crust" value="thick"> Thick</li>

      </ul>

    </section>

    <section>

      <h2>Place Your Order</h2>

      <p>Enter your details and submit your order:</p>

      <form>

        <label for="name">Name:</label>

        <input type="text" id="name" name="name" required>

        <br>

        <label for="address">Address:</label>

        <textarea id="address" name="address" required></textarea>

        <br>

        <input type="submit" value="Submit Order">

      </form>

    </section>

  </main>

  <footer>

    &copy; 2023 Pizza Order. All rights reserved.

  </footer>

</body>

</html>

В этом примере мы структурировали веб-страницу с помощью семантических HTML-тегов. <header> содержит заголовок страницы и навигационное меню. Тег <nav> представляет навигационные ссылки. Основное содержимое заключено в тег <main>, причем каждый раздел содержимого заключен в теги <section>. Нижний колонтитул помечается с помощью тега <footer>.

Предоставленный CSS (стили страниц) предназначен для демонстрационных целей и может быть дополнительно настроен для достижения желаемого визуального стиля.

# CSS

CSS, сокращение от «Cascading Style Sheets - Каскадные Таблиц Стилей», является фундаментальной технологией, используемой для оформления и форматирования веб-страниц. Он позволяет управлять визуальным оформлением HTML-элементов, таких как шрифты, цвета, макет и позиционирование. Далее рассмотрим основные правила CSS, позиционирование элементов и концепцию адаптивности веб-страницы.

**Основные правила CSS:** CSS использует набор правил для определения того, как элементы должны быть оформлены. Каждое правило состоит из селектора и блока объявления. Селектор нацелен на один или несколько HTML-элементов, а блок объявления содержит одну или несколько пар свойство-значение.

Пример: Допустим, у нас есть HTML-элемент с классом "box". Мы можем оформить его с помощью CSS следующим образом:

.box {

    background-color: blue;

    color: white;

    font-size: 18px;

}

В этом примере селектором является .box, который содержит три пары свойство-значение. Свойство background-color устанавливает цвет фона на синий, свойство color устанавливает цвет текста на белый, а свойство font-size устанавливает размер шрифта равным 18 пикселям.

**Позиционирование элементов:** CSS предоставляет несколько способов позиционирования элементов на веб-странице. Наиболее часто используемыми методами позиционирования являются "static," "relative," "absolute," and "fixed".

* *Static Positioning*: Это расположение элементов HTML по умолчанию, в котором они отображаются в обычном потоке документа.
* *Relative Positioning*: Элементы, расположенные относительно друг друга, регулируются относительно их нормального положения. Вы можете использовать такие свойства, как top, bottom, left и right, чтобы сместить элемент от его нормального положения.
* *Absolute Positioning*: При абсолютном позиционировании элементы располагаются относительно их ближайшего родительского элемента. Если ни у одного из родительских элементов нет определенной позиции, она будет находиться относительно самого документа.
* *Fixed Positioning*: Фиксированное позиционирование фиксирует положение элемента относительно окна браузера. Он остается в том же положении даже при прокрутке страницы.

Пример: Рассмотрим следующий фрагмент кода CSS:

.box {

    position: relative;

    top: 20px;

    left: 30px;

}

В этом примере элемент .box расположен относительно, и он будет перемещен на 20 пикселей вниз и на 30 пикселей вправо от своего обычного положения.

**Адаптивность веб-страниц:** В современном мире крайне важно создавать веб-страницы, которые адаптируются к различным размерам экрана и устройствам. CSS предоставляет несколько методов для достижения адаптивности веб-страницы.

* *Responsive Design*: Адаптивный дизайн использует медиа-запросы для применения различных стилей в зависимости от характеристик устройства, таких как ширина, высота или ориентация экрана.

Пример:

@media screen and (max-width: 600px) {

    body {

      font-size: 14px;

    }

}

В этом примере, когда ширина экрана составляет 600 пикселей или меньше, размер шрифта основного элемента будет уменьшен до 14 пикселей.

* *Fluid Layouts*: Fluid layouts используют относительные единицы измерения, такие как проценты, вместо фиксированных единиц измерения, таких как пиксели, для указания размеров элемента. Это позволяет элементам пропорционально изменять размер по мере изменения размера экрана.

Пример:

.container {

    width: 80%;

}

.box {

    width: 50%;

}

В этом примере элемент .container будет занимать 80% ширины своего родительского элемента, в то время как элемент .box будет занимать 50% ширины элемента .container.

**Оформление текста:** CSS позволяет применять к тексту различные стили, такие как изменение шрифта, цвета, размера и выравнивания.

Example:

h1 {

    font-family: Arial, sans-serif;

    color: #FF0000;

    font-size: 24px;

    text-align: center;

}

В этом примере элемент заголовка h1 будет иметь семейство шрифтов Arial, красный цвет (#FF0000), размер шрифта 24 пикселя и будет расположен по центру.

**Задание теней элементам:** Тени в CSS добавляют глубину и визуальные эффекты элементам. Вы можете управлять цветом, размером и положением тени.

Пример:

.box {

    box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.3);

}

В этом примере элемент .box будет иметь тень со смещением по горизонтали на 2 пикселя, смещением по вертикали на 2 пикселя, радиусом размытия в 5 пикселей и слегка прозрачным черным цветом (rgba(0, 0, 0, 0.3)).

**Эффекты наведения (Hover Effects)**: CSS позволяет добавлять эффекты наведения к элементам, которые меняют свой внешний вид при наведении на них курсора.

Пример:

.button {

    background-color: #007BFF;

    color: #FFFFFF;

    padding: 10px 20px;

    transition: background-color 0.3s ease;

}

.button:hover {

    background-color: #0056b3;

}

В этом примере элемент .button будет иметь синий цвет фона и белый текст. При наведении курсора цвет фона плавно перейдет в более темный оттенок синего.

**Flexbox Layout**: CSS Flexbox предоставляет гибкий способ компоновки элементов внутри контейнера, обеспечивая легкое выравнивание и распределение.

Пример:

.container {

    display: flex;

    justify-content: space-between;

}

.item {

    flex: 1;

    margin: 10px;

}

В этом примере элемент .container расположит свои дочерние элементы с равным расстоянием между ними по горизонтали. Элементы .item будут иметь одинаковые значения «flex grow», в результате чего они будут занимать одинаковое пространство внутри контейнера.

Эти примеры демонстрируют лишь некоторые из множества возможностей, которые предлагает CSS для стилизации и верстки веб-страниц. Эксперименты с различными свойствами и значениями помогут вам получить более глубокое представление о CSS и его возможностях.

**Адаптивная верстка**

Адаптивная верстка, также известная как «responsive layout», - это практика проектирования веб-страниц, которые адаптируются к различным размерам экрана и устройствам и реагируют на них. Это гарантирует, что содержимое и макет остаются читаемыми и пригодными для использования на различных устройствах, таких как настольные компьютеры, ноутбуки, планшеты и смартфоны.

Для достижения адаптивной верстки необходимо учитывать несколько ключевых принципов и приемов:

**Fluid Grids:** Одной из фундаментальных концепций является использование Fluid Grids. Вместо использования фиксированных измерений ширины элементов на основе пикселей в fluid grids используются относительные единицы измерения, такие как проценты. Это позволяет пропорционально изменять размер элементов в зависимости от доступной ширины экрана.

Пример:

.container {

    width: 100%;

    display: grid;

    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));

    gap: 20px;

}

В этом примере элемент .container использует макет сетки со столбцами, которые динамически настраиваются в зависимости от доступной ширины. Правило repeat(autofit, min max(200px, 1fr)) гарантирует, что каждый столбец будет иметь минимальную ширину в 200 пикселей, но будет расширяться, чтобы равномерно заполнить любое дополнительное пространство.

**Media Queries (медиа запросы)**: Медиа-запросы - это мощная функция CSS, которая позволяет применять различные стили в зависимости от конкретных условий, таких как ширина экрана, высота, ориентация или тип устройства. Медиа-запросы позволяют создавать breakpoints, в которых макет и стили могут быть соответствующим образом скорректированы.

Пример:

@media screen and (max-width: 768px) {

    .container {

      grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));

    }

}

В этом примере медиазапрос нацелен на экраны с максимальной шириной 768 пикселей. Внутри запроса, столбцы элемента .container настраиваются таким образом, чтобы их минимальная ширина составляла 150 пикселей, обеспечивая более подходящий макет для экранов меньшего размера.

**Гибкие изображения и мультимедиа (Flexible Images and Media):** Адаптивные макеты также должны учитывать изображения и мультимедийные элементы. Чтобы они не переполнялись или не становились слишком маленькими, можно применить такие свойства CSS, как max-width: 100% и height: auto, чтобы гарантировать, что изображения и мультимедийные материалы надлежащим образом масштабируются в своих контейнерах.

Пример:

img {

    max-width: 100%;

    height: auto;

}

Это правило CSS гарантирует, что изображения никогда не будут превышать ширину своего контейнера, сохраняя соотношение сторон и уменьшая масштаб при необходимости.

**Mobile-First Approach**: Одной из распространенных стратегий в адаптивном макетировании является использование подхода, ориентированного в первую очередь на мобильные устройства. Это означает первоначальное проектирование и стилизацию для экранов меньшего размера, а затем постепенное улучшение макета по мере увеличения размера экрана. Начав с минималистичного, удобного для мобильных устройств макета, вы гарантируете, что контент останется доступным на всех устройствах.

Адаптивная верстка необходима для создания удобных и доступных веб-страниц на разных устройствах. Используя гибкие сетки, медиа-запросы, гибкие изображения и применяя подход, ориентированный на мобильные устройства, вы можете создавать макеты, которые легко адаптируются и обеспечивают оптимальный пользовательский опыт.

Вот обновленная версия кода («Pizza Order)» с добавленным разделом галереи и примененными дополнительными стилями CSS:

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Pizza Order</title>

  <style>

    /\* Basic styling \*/

    body {

      font-family: Arial, sans-serif;

      margin: 0;

      padding: 20px;

    }

    /\* Header styles \*/

    header {

      background-color: #f2f2f2;

      padding: 10px;

      text-align: center;

    }

    nav ul {

      list-style: none;

      padding: 0;

      margin: 0;

      display: flex;

      justify-content: center;

    }

    nav ul li {

      margin: 0 10px;

    }

    nav ul li a {

      text-decoration: none;

      color: #333;

    }

    /\* Main styles \*/

    main {

      margin-bottom: 20px;

    }

    section {

      padding: 10px;

      background-color: #f9f9f9;

      margin-bottom: 10px;

    }

    section h2 {

      color: #333;

    }

    section p {

      color: #666;

    }

    ul {

      list-style: disc;

      margin-left: 20px;

    }

    /\* Form styles \*/

    form label {

      display: block;

      margin-bottom: 5px;

      color: #333;

    }

    form input[type="text"],

    form textarea {

      width: 100%;

      padding: 5px;

      margin-bottom: 10px;

      border: 1px solid #ccc;

    }

    form input[type="submit"] {

      background-color: #333;

      color: #fff;

      padding: 10px 20px;

      border: none;

      cursor: pointer;

    }

    /\* Footer styles \*/

    footer {

      background-color: #f2f2f2;

      padding: 10px;

      text-align: center;

    }

    /\* Gallery styles \*/

    .gallery {

      display: flex;

      flex-wrap: wrap;

      justify-content: center;

    }

    .gallery-item {

      width: 200px;

      margin: 10px;

    }

    .gallery-item img {

      width: 100%;

      height: auto;

    }

    /\* Media queries \*/

    @media (max-width: 767px) {

      .gallery-item {

        width: 100%;

        max-width: 300px;

      }

    }

  </style>

</head>

<body>

  <header>

    <h1>Pizza Order</h1>

    <nav>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">Menu</a></li>

        <li><a href="#">Order</a></li>

        <li><a href="#">Contact</a></li>

      </ul>

    </nav>

  </header>

  <main>

    <section>

      <h2>Choose Your Pizza</h2>

      <p>Select your favorite pizza from our menu:</p>

      <div class="gallery">

        <div class="gallery-item">

          <img src="pizza1.jpg" alt="Margherita Pizza">

          <p>Margherita Pizza</p>

        </div>

        <div class="gallery-item">

          <img src="pizza2.jpg" alt="Pepperoni Pizza">

          <p>Pepperoni Pizza</p>

        </div>

        <div class="gallery-item">

          <img src="pizza3.jpg" alt="Hawaiian Pizza">

          <p>Hawaiian Pizza</p>

        </div>

        <div class="gallery-item">

          <img src="pizza4.jpg" alt="Veggie Supreme Pizza">

          <p>Veggie Supreme Pizza</p>

        </div>

      </div>

    </section>

    <section>

      <h2>Customize Your Pizza</h2>

      <p>Customize your pizza with the following options:</p>

      <ul>

        <li>Size:

          <select>

            <option>Small</option>

            <option>Medium</option>

            <option>Large</option>

          </select>

        </li>

        <li>Toppings:

          <label><input type="checkbox"> Pepperoni</label>

          <label><input type="checkbox"> Mushrooms</label>

          <label><input type="checkbox"> Olives</label>

        </li>

        <li>Crust:

          <label><input type="radio" name="crust" value="thin"> Thin</label>

          <label><input type="radio" name="crust" value="thick"> Thick</label>

        </li>

      </ul>

    </section>

    <section>

      <h2>Place Your Order</h2>

      <p>Enter your details and submit your order:</p>

      <form>

        <label for="name">Name:</label>

        <input type="text" id="name" name="name" required>

        <br>

        <label for="address">Address:</label>

        <textarea id="address" name="address" required></textarea>

        <br>

        <input type="submit" value="Submit Order">

      </form>

    </section>

  </main>

  <footer>

    &copy; 2023 Pizza Order. All rights reserved.

  </footer>

</body>

</html>

В этой обновленной версии мы добавили раздел галереи, используя элемент <div>, и применили стили CSS для создания адаптивного макета галереи. Каждый элемент пиццы представлен <div> с классом "gallery-item", содержащим тег <img> для изображения пиццы и тег <p> для названия пиццы. Вы можете заменить URL-адреса изображений (pizza1.jpg , pizza2.jpg , и т.д.) на ваши собственные URL-адреса изображений или используйте соответствующие пути к файлам изображений.

Также мы внесли следующие изменения:

* + Добавлены стили Flexbox в класс .gallery для создания гибкого макета галереи с переносом элементов.
  + Увеличена ширина элемента .gallery до 200 пикселей и добавлено поле для интервала.
  + Добавлен медиа-запрос с использованием @media для изменения ширины элемента .gallery до 100% с максимальной шириной 300 пикселей, когда ширина области просмотра составляет 767 пикселей или меньше. Это гарантирует, что элементы галереи будут располагаться вертикально на экранах меньшего размера.

Не стесняйтесь дополнительно настраивать стили галереи для достижения желаемой визуальной презентации.

# Bootstrap Framework

Bootstrap - популярный интерфейсный фреймворк для создания адаптивных веб-сайтов. В этом разделе мы рассмотрим основы Bootstrap и рассмотрим несколько примеров, которые помогут вам начать работу с ним.

**Что такое Bootstrap?**

* + Bootstrap - это CSS-фреймворк с открытым исходным кодом, разработанный Twitter.
  + Он предоставляет набор готовых адаптивных компонентов, таких как сетки, типографика, формы, кнопки и навигация.
  + Bootstrap позволяет разработчикам быстро создавать современные и визуально привлекательные веб-сайты.

**Основные особенности Bootstrap:**

* + *Адаптивная система сеток*: система сеток Bootstrap основана на макете из 12 столбцов, что позволяет разработчикам создавать адаптивные дизайны для экранов различных размеров.
  + *Компоненты CSS*: Bootstrap включает в себя широкий спектр повторно используемых компонентов CSS, таких как кнопки, формы, панели навигации и оповещения.
  + *Типографика:* Bootstrap предоставляет набор типографских стилей и утилит для улучшения форматирования текста.
  + *Плагины JavaScript:* Bootstrap поставляется в комплекте с плагинами JavaScript, такими как карусели, модалы, всплывающие подсказки и многое другое, для улучшения функциональности веб-сайта.
  + *Настройка:* Bootstrap можно настроить в соответствии с конкретными требованиями к дизайну, изменив переменные или используя пользовательские таблицы стилей.

**Начало работы с Bootstrap:**

1. Загрузка Bootstrap: Посетите официальный веб-сайт Bootstrap (getbootstrap.com ) и загрузите предварительно скомпилированные файлы CSS и JavaScript.
2. Подключение файлов Bootstrap: Добавьте ссылки на файли Bootstrap CSS в раздел head и ссылки на файлы JavaScript непосредственно перед закрывающим тегом body в HTML-файле.
3. Использование Bootstrap CDN: В качестве альтернативы вы можете использовать сеть доставки контента Bootstrap (CDN), добавив соответствующие ссылки на файлы CSS и JavaScript в свой HTML-файл.

**Bootstrap Grid System (система сеток):**

• Система сеток Bootstrap обеспечивает гибкий и отзывчивый макет для создания веб-страниц.

• Система сеток основана на макете из 12 столбцов, который позволяет разделить страницу на несколько столбцов.

Примеры:

*<div class="container">*: Создает контейнер, в котором размещается система сеток.

*<div class="row">*: Определяет строку для хранения столбцов.

*<div class="col-md-6">*: Создает столбец шириной 6 из 12 столбцов на экранах среднего размера.

**Bootstrap CSS компоненты:**

Bootstrap предлагает широкий спектр CSS-компонентов, которые могут быть легко интегрированы в веб-страницы.

Примеры:

*Buttons*: <button class="btn btn-primary">Primary Button</button>

*Forms*: <input class="form-control" type="text" placeholder="Enter your name">

*Navigation*: <nav class="navbar navbar-expand-lg navbar-dark bg-dark">...</nav>

*Alerts*: <div class="alert alert-info" role="alert">...</div>

**Bootstrap типографика:**

Bootstrap предоставляет набор предопределенных стилей типографики и утилит.

Примеры:

*Заголовки:* <h1>Heading 1</h1> to <h6>Heading 6</h6>

*Форматирование текста*: <p class="lead">This is a lead paragraph.</p>

*Выравнивание текста*: <div class="text-center">Center-aligned text</div>

**Bootstrap JavaScript плагины:**

Bootstrap включает в себя множество плагинов JavaScript, которые могут расширить функциональность вашего веб-сайта.

Примеры:

*Карусели (слайдеры):* <div id="myCarousel" class="carousel slide">...</div>

*Модальные окна*: <div class="modal fade" tabindex="-1" role="dialog">...</div>

*Подсказки*: <button class="btn" data-toggle="tooltip" data-placement="top" title="Tooltip text">...</button>

**Кастомизация Bootstrap:**

Bootstrap может быть настроен в соответствии с вашими требованиями к дизайну.

Параметры настройки включают в себя:

* + *Изменение переменных*: Bootstrap предоставляет набор SCSS-переменных, которые можно переопределять для изменения цветов, интервалов и многого другого.
  + *Пользовательские таблицы стилей*: Вы можете создавать пользовательские таблицы стилей CSS и переопределять стили Bootstrap по умолчанию.

Bootstrap - это мощный фреймворк, который позволяет быстро создавать адаптивные веб-сайты. Он предоставляет широкий спектр готовых компонентов и адаптивную систему сеток. Используя Bootstrap, вы можете сэкономить время и усилия при проектировании и разработке современных веб-сайтов.

**Полезные ресурсы:**

Официальный веб-сайт: <https://getbootstrap.com/>.

Документация: <https://getbootstrap.com/docs/>.

Примеры: <https://getbootstrap.com/docs/examples/>.

# Язык программирования javaScript

JavaScript - популярный язык программирования, используемый для веб-разработки и создания интерактивных элементов на веб-сайтах. Далее рассмотрим фундаментальные концепции javaScript и приведем примеры, которые помогут вам понять его использование.

1. Переменные и типы данных: JavaScript позволяет хранить данные и манипулировать ими с помощью переменных. Вот пример:

// Declaration and assignment of variables

let name = "John";

let age = 25;

let isStudent = true;

В этом примере мы объявляем три переменные: name, age и isStudent. Мы присваиваем этим переменным значения "John" (строка), 25 (число) и true (логическое значение) соответственно. JavaScript поддерживает различные типы данных, включая строки, числа, логические значения, массивы, объекты и многое другое.

1. Операторы и выражения: JavaScript предоставляет ряд операторов для выполнения арифметических, сравнительных и логических операций. Давайте рассмотрим несколько примеров:

let a = 5;

let b = 10;

let sum = a + b; // Addition

let product = a \* b; // Multiplication

let isGreater = a > b; // Comparison

let logicalAnd = (a > 0) && (b > 0); // Logical AND

В этом примере мы выполняем сложение, умножение, сравнение и логические операции AND, используя переменные «a» и «b».

1. Управление потоком: JavaScript позволяет вам управлять потоком вашей программы с помощью условных операторов и циклов. Есть два распространенных примера:

Условные операторы (if-else):

let age = 18;

if (age >= 18) {

  console.log("You are eligible to vote.");

} else {

  console.log("You are not eligible to vote yet.");

}

В этом примере мы проверяем, превышает ли возраст 18 лет или равен ему. Если это так, мы печатаем сообщение о праве на участие в голосовании; в противном случае мы печатаем другое сообщение.

Циклы (for loop):

for (let i = 0; i < 5; i++) {

    console.log(i);

  }

Этот цикл выведет числа от 0 до 4. Цикл начинается с i = 0, проверяет условие (i < 5), выполняет код внутри цикла и увеличивает i на 1 до тех пор, пока условие не перестанет быть истинным.

1. Функции: Функции - это повторно используемые блоки кода в JavaScript. Они позволяют определить набор инструкций, которые могут быть вызваны несколько раз. Вот пример:

function greet(name) {

    console.log("Hello, " + name + "!");

  }

  greet("John");

В этом примере мы определяем функцию под названием «greet», которая принимает параметр «name» и печатает приветственное сообщение. Затем мы вызываем функцию с аргументом «John», в результате чего получаем вывод "Hello, John!".

JavaScript - это универсальный язык программирования, который является неотъемлемой частью веб-разработки. Мы рассмотрели некоторые основные понятия, включая переменные, типы данных, операторы, поток управления и функции. Продолжайте практиковаться и изучать JavaScript, чтобы еще больше улучшить свои навыки программирования.

Вот несколько ресурсов, которые могут помочь вам изучить JavaScript:

1. Mozilla Developer Network (MDN) JavaScript Guide:
   * Веб-сайт: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
   * Руководство MDN по JavaScript - отличный ресурс для начинающих. В нем рассматриваются основы JavaScript, включая типы данных, поток управления, функции и многое другое. В нем также содержатся подробные объяснения и примеры.
2. JavaScript.info:
   * Веб-сайт: https://javascript.info/
   * JavaScript.info - это всеобъемлющий ресурс для изучения JavaScript с нуля. Там рассматриваются темы, начиная от основ и заканчивая более продвинутыми концепциями, включая объектно-ориентированное программирование и асинхронное программирование. Веб-сайт предлагает интерактивные примеры и упражнения для углубления вашего понимания.
3. Eloquent JavaScript by Marijn Haverbeke:
   * Книга: "Eloquent JavaScript" Marijn Haverbeke
   * Эта книга высоко ценится в сообществе JavaScript. Она предлагает удобное для начинающих введение в JavaScript, охватывающее фундаментальные концепции и постепенно переходящее к более продвинутым темам. Книга включает в себя практические упражнения для закрепления ваших знаний.
4. JavaScript: The Good Parts by Douglas Crockford:
   * Книга: "JavaScript: The Good Parts" Douglas Crockford
   * Эта книга посвящена основным разделам JavaScript, содержит рекомендации по наилучшей практике и позволяет избежать распространенных ошибок. Это краткий и содержательный ресурс для понимания основных концепций JavaScript.
5. FreeCodeCamp:
   * Веб-сайт: https://www.freecodecamp.org/
   * freeCodeCamp предлагает бесплатную интерактивную платформу для обучения веб-разработке, включая JavaScript. Он предоставляет структурированную учебную программу с практическими упражнениями по кодированию и проектами. Раздел JavaScript охватывает широкий спектр тем и подходит для начинающих.
6. Codecademy:
   * Веб-сайт: https://www.codecademy.com/learn/introduction-to-javascript
   * Codecademy предлагает интерактивный курс JavaScript, который обучает основам языка. Курс предоставляет практический опыт обучения, позволяя вам писать код непосредственно в браузере и получать мгновенную обратную связь.

Примеры того, как можно использовать JavaScript:

1. Динамическое изменение содержимого HTML JavaScript можно использовать для манипулирования элементами HTML и динамического изменения их контента. Вот пример, который изменяет текст элемента абзаца при нажатии кнопки:

<!DOCTYPE html>

<html>

<head>

  <title>JavaScript Example</title>

</head>

<body>

  <p id="demo">Click the button to change this text.</p>

  <button onclick="changeText()">Click Me</button>

  <script>

    function changeText() {

      document.getElementById("demo").innerHTML = "Text changed!";

    }

  </script>

</body>

</html>

При нажатии на кнопку вызывается функция «changeText()», которая выбирает элемент абзаца с идентификатором "demo" и изменяет его свойство «innerHTML» для отображения нового текста.

1. Обработка отправки формы JavaScript можно использовать для проверки вводимых данных в форму и выполнения действий при отправке формы. Вот пример, который проверяет простую форму перед ее отправкой:

<!DOCTYPE html>

<html>

<head>

  <title>JavaScript Example</title>

</head>

<body>

  <form onsubmit="return validateForm()">

    <input type="text" id="name" placeholder="Enter your name">

    <input type="email" id="email" placeholder="Enter your email">

    <input type="submit" value="Submit">

  </form>

  <script>

    function validateForm() {

      var name = document.getElementById("name").value;

      var email = document.getElementById("email").value;

      if (name === "" || email === "") {

        alert("Please fill in all fields.");

        return false;

      }

      // Additional validation or form processing code can be added here

      // The form will only be submitted if all validations pass

      alert("Form submitted successfully!");

      return true;

    }

  </script>

</body>

</html>

Функция «validateForm()» вызывается при отправке формы. Она извлекает значения полей имени и электронной почты, проверяет, пусты ли они, и выводит предупреждающее сообщение, если какое-либо поле отсутствует. Если все проверки пройдены, выводится предупреждение, указывающее на успешную отправку формы.

1. Создание простого слайд-шоу. JavaScript можно использовать для создания интерактивных элементов, таких как слайд-шоу. Вот пример, который реализует базовое слайд-шоу с изображениями:

<!DOCTYPE html>

<html>

<head>

  <title>JavaScript Example</title>

  <style>

    .slideshow {

      max-width: 500px;

      height: 300px;

      position: relative;

      margin: 0 auto;

    }

    .slideshow img {

      width: 100%;

      height: 100%;

      object-fit: cover;

    }

  </style>

</head>

<body>

  <div class="slideshow">

    <img src="image1.jpg" id="slide" alt="Slideshow Image">

  </div>

  <script>

    var images = ["image1.jpg", "image2.jpg", "image3.jpg"];

    var currentIndex = 0;

    function changeSlide() {

      var slide = document.getElementById("slide");

      slide.src = images[currentIndex];

      currentIndex++;

      if (currentIndex >= images.length) {

        currentIndex = 0;

      }

      setTimeout(changeSlide, 2000); // Change slide every 2 seconds

    }

    changeSlide(); // Start the slideshow

  </script>

</body>

</html>

В этом примере определен массив URL-адресов изображений, и функция «changeSlide()» используется для обновления атрибута «src» элемента изображения следующим изображением в массиве. Функция «setTimeout()» используется для повторного вызова «changeSlide()» каждые 2 секунды, создавая автоматический эффект слайд-шоу.

Это лишь несколько примеров того, как JavaScript можно использовать для улучшения веб-сайтов и добавления интерактивности. Универсальность JavaScript позволяет использовать его для широкого спектра целей, включая проверку форм, анимацию, манипулирование DOM и многое другое.

# Библиотека React.js. Создание реактивных интерфейсов.

Разработка любого веб-проекта включает в себя написание клиентской части приложения. Сегодня существует довольно много фреймворков и библиотек для решения этой задачи, например: jQuery, React, Angular.js, Vue.js. Некоторые из этих инструментов являются фреймворками, некоторые - библиотеками. Давайте посмотрим, в чем разница между ними.

**Framework**

Любой разработчик рано или поздно сталкивается с двумя новыми для себя терминами: библиотека и фреймворк. Термины разные, но часто их значение считают одинаковым. Особенно, когда дело доходит до контекста интерфейса. Давайте попробуем разобраться в сходствах и различиях.

Слово "фреймворк" буквально переводится как "основа". Говоря еще проще: фреймворк - это фундамент, на котором может быть построено любое приложение. Для фреймворка не имеет значения, "что" вы будете разрабатывать с его помощью, важно "как". Для этого в фреймворке есть специальные правила - ограничения.

Архитектура фреймворка может включать в себя файловую структуру проекта, встроенный набор инструментов для решения различных задач, набор внутренних абстракций, CLI-инструменты для автоматизации задач и так далее.

Существует множество фреймворков для интерфейсной разработки. Из наиболее популярных выделяются Angular, Vue и Ember. Все они предоставляют набор встроенных инструментов, абстракций и так далее. Их можно использовать прямо из коробки, не тратя время на поиск подходящих модулей и налаживание связей.

**Библиотека**

Если фреймворк - это готовый инструмент для разработки приложения, то библиотека - это просто отдельный инструмент для решения какой-то задачи или нескольких задач. Библиотека не накладывает никаких ограничений на архитектуру или другие инструменты, используемые в проекте. Она сама по себе.

Здесь не нужно далеко ходить за примерами. Некогда популярная библиотека jQuery - это не что иное, как набор функций, которые упрощают взаимодействие с DOM. Другой пример - Lodash — библиотека с обширным набором полезных функций на все случаи жизни.

Перечисленные выше библиотеки, как и любые другие, не предъявляют никаких требований к архитектуре или типу проекта. Неважно, как организован ваш проект, ничто не мешает вам включить библиотеку и использовать ее функциональность.

Библиотеку можно рассматривать как дополнительный модуль с полезными функциями. Опять же, библиотека - это инструмент. Вполне логично, что в одном проекте можно использовать несколько разных инструментов - библиотек. Это нормальная ситуация, но при этом в проекте может быть только один фреймворк.

**Что такое React.js?**

React.js представляет собой библиотеку JavaScript с открытым исходным кодом, созданную Facebook. Она обеспечивает декларативный и эффективный способ построения пользовательских интерфейсов путем разбиения их на повторно используемые компоненты.

**Базовые концепции React.js**

Чтобы понять React.js , важно понимать некоторые из его основных концепций:

1. **Components:** В самом сердце React.js являются компонентами. Компонент - это повторно используемый автономный модуль, который отображает часть пользовательского интерфейса. Компоненты можно рассматривать как строительные блоки, которые можно комбинировать для создания более крупных и сложных пользовательских интерфейсов. Существует два типа компонентов: функциональные компоненты (также называемые компонентами без сохранения состояния) и компоненты класса.
2. **JSX (JavaScript XML):** JSX - это расширение синтаксиса, используемое React, которое позволяет писать HTML-подобный код в JavaScript. Это позволяет описать, как должны выглядеть ваши компоненты. JSX делает код React более читабельным и интуитивно понятным, помогая преодолеть разрыв между HTML и JavaScript.
3. **Virtual DOM (Document Object Model):** React использует виртуальный DOM для эффективного обновления и рендеринга компонентов. Виртуальный DOM - это облегченная копия реального DOM, которая представляет структуру вашей веб-страницы. Когда состояние компонента изменяется, React сравнивает виртуальный DOM с фактическим DOM и обновляет только необходимые части, что обеспечивает оптимальную производительность.
4. **State:** Состояние представляет данные, которые компонент может хранить и которыми он может управлять. Это позволяет компонентам вести себя динамично, обновляя и отражая изменения в пользовательском интерфейсе. Состояние обычно используется для хранения информации, которая может изменяться с течением времени, такой как пользовательский ввод, ответы API или результат вычислений.
5. **Props (Properties):** Props - это входные данные, которые компоненты получают от своих родительских компонентов. Они доступны только для чтения и позволяют передавать данные от родительского компонента к его дочерним элементам. Props необходимы для передачи информации и поведения между компонентами, что упрощает создание многоразового и модульного кода.

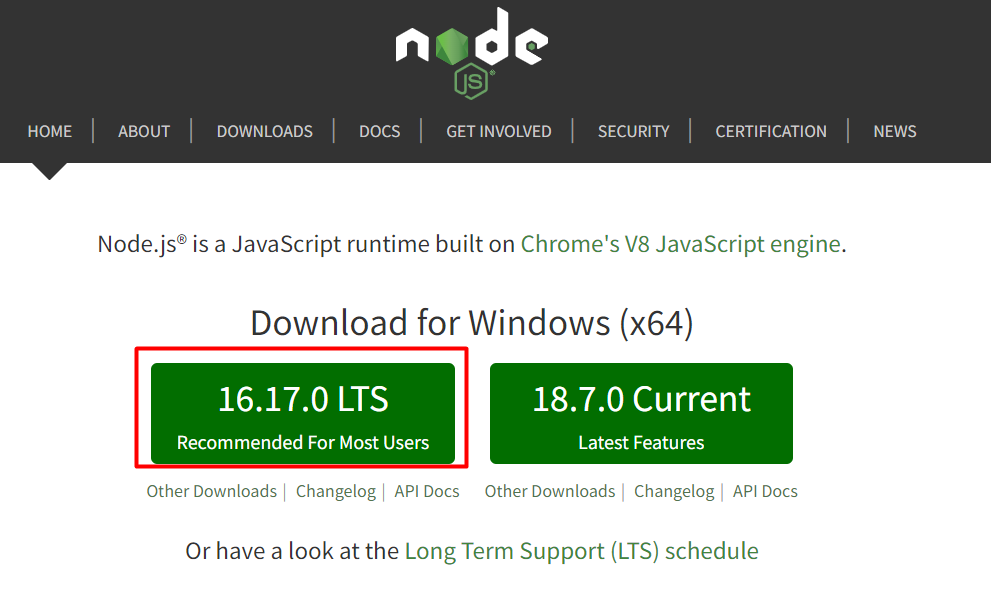
React - это *библиотека* JavaScript, используемая для создания пользовательских интерфейсов. В отличие от фреймворка, React предоставляет основанный на компонентах подход к разработке пользовательского интерфейса. Его можно использовать для визуализации в сочетании с другими библиотеками. Например, React Native используется для разработки мобильных приложений, в то время как React 360 используется для создания приложений виртуальной реальности.

Основная цель React - свести к минимуму ошибки при разработке пользовательского интерфейса. Это достигается за счет использования автономных и повторно используемых компонентов, которые описывают различные части пользовательского интерфейса. Эти компоненты могут быть объединены для формирования полноценного пользовательского интерфейса. Абстрагируясь от большей части процесса рендеринга, React позволяет разработчикам больше сосредоточиться на дизайне и функциональности своих приложений.

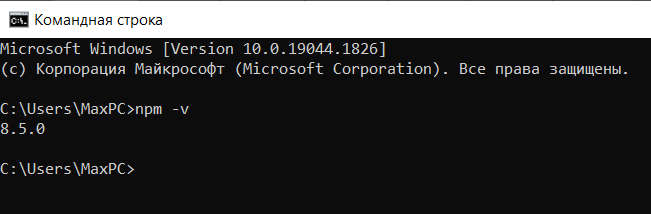
**Настройка среды разработки React.js**

Чтобы начать строить с React.js , нам нужно настроить среду разработки. Необходимо выполнить следующие действия:

1. Установите Node.js: React.js полагается на Node.js для запуска JavaScript на сервере. Посетите официальный Node.js веб-сайт (https://nodejs.org ) и загрузите последнюю версию LTS (долгосрочная поддержка), подходящую для вашей операционной системы. После установки у вас будет доступ к диспетчеру пакетов Node (npm).



Чтобы убедиться в правильности установки, мы можем проверить установленную версию npm с помощью команды "npm -v":



1. Создайте папку на рабочем столе.
2. Откройте командную строку из текущей папки (для этого можно просто ввести команду в адресной строке текущего окна и нажать Enter).
3. Создайте новый проект React: запустите следующую команду, чтобы создать новый проект React:

npx create-react-app my-app

Эта команда создает новый проект React со всеми необходимыми зависимостями и базовой структурой проекта. Обратите внимание, что во время установки проекта все необходимые зависимости загружаются через «npm». Это может занять некоторое время.

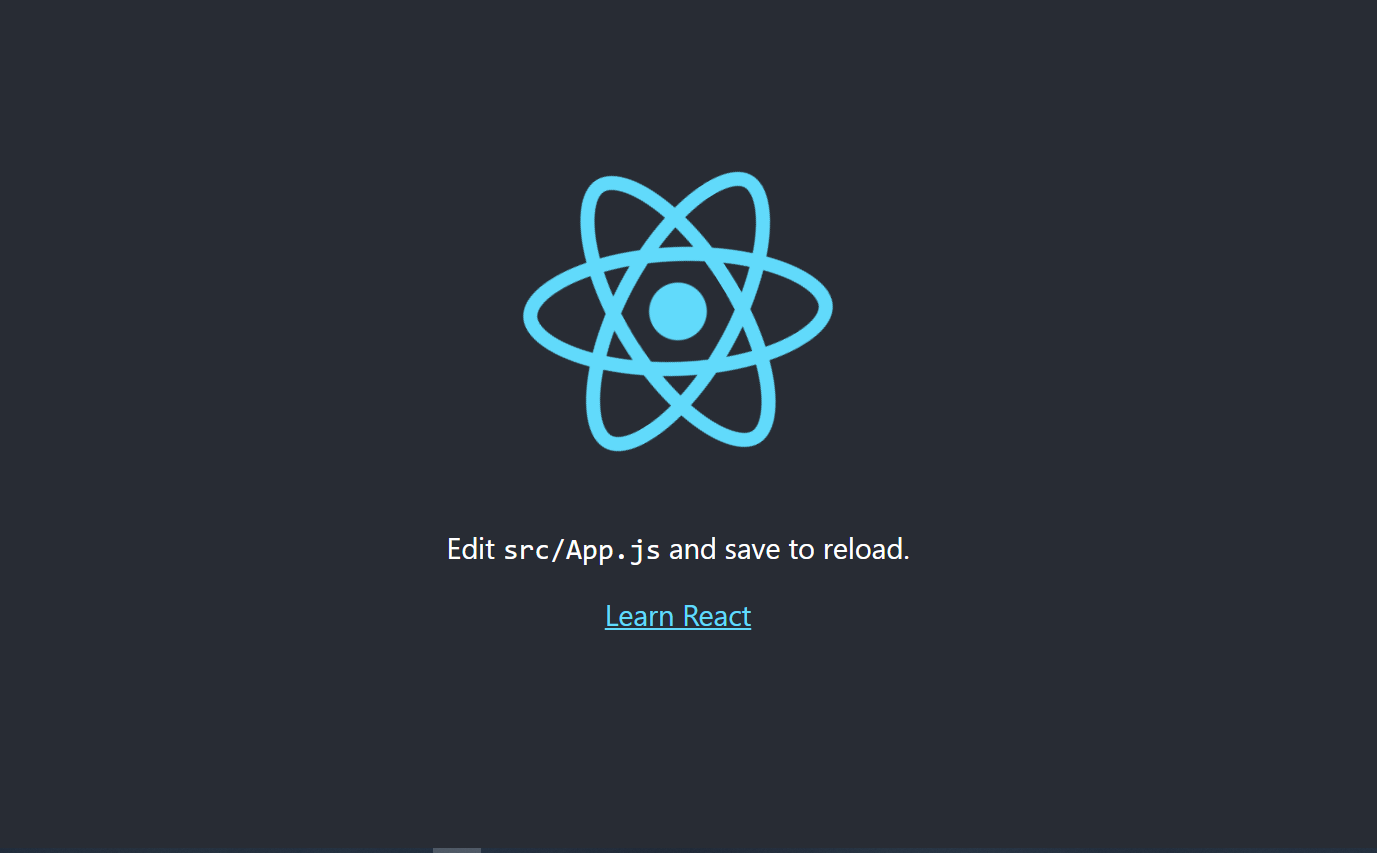
1. Запустите сервер разработки: перейдите в папку проекта, выполнив:

cd my-app

Затем запустите сервер разработки с помощью следующей команды:

npm start

Главная страница приложения React по умолчанию выглядит следующим образом:

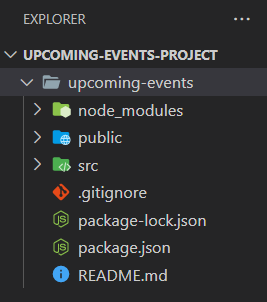


В проекте включена функция мгновенного "горячего обновления". Это означает, что любые изменения, внесенные в код, автоматически вызовут перезагрузку страницы, что позволит вам увидеть обновления в режиме реального времени после сохранения.

Затем откройте проект в Visual Studio Code. (File -> Open Folder -> Выберите папку проекта).

**Структура проекта**

CRA (команда «create-react-app») подготавливает базовую структуру для каждого проекта (см. рисунок ниже). Перейдя в каталог проекта, вы увидите несколько каталогов:



* + node\_modules. Каталог с загруженными зависимостями;
  + public. Каталог для статических файлов. На эту папку настроен сервер разработки. Она должна содержать все статические ресурсы проекта (шрифты, общие стили, html, изображения и так далее);
  + src. Каталог для исходного кода проекта. CRA автоматически подготавливает несколько файлов с компонентами для демонстрацонных целей. Затем их можно будет удалить. В корне проекта находятся файлы, стандартные для любого проекта: package.json, package-lock.json и readme.md .

**Готовые сценарии из коробки**

В проекте, созданном с помощью CRA, в файле package.json доступны несколько скриптов. Этот набор может отличаться в зависимости от шаблона, который использовался при создании проекта:

* + start. Запуск проекта. По умолчанию сервер разработки подключается к порту 3000;
  + build. Подготовка сборки проекта. Он используется при подготовке проекта к отправке на сервер продукта;
  + test. Запуск автоматизированного тестирования;
  + eject. Сброс конфигурации. Для более точной настройки.

**Дополнительные настройки**

Инструмент CRA настраивает все самостоятельно. Однако настройки по умолчанию не всегда подходят. Может возникнуть, например, такая проблема: порт 3000 (он используется по умолчанию) может быть занят другой службой. В качестве альтернативы, вы, как разработчик, хотите, чтобы проект запускался в определенном браузере, а не в том, который загружается по умолчанию.

Некоторые настройки по умолчанию могут быть переопределены с помощью переменных окружения. В документации для CRA есть отдельный раздел "Расширенная конфигурация". В нем перечислены все переменные окружения, которые позволяют вам это делать.

Вы можете установить желаемое значение для определенной переменной во время запуска проекта прямо в терминале. Команда будет выглядеть следующим образом:

PORT=6000 npm run start

Здесь мы устанавливаем новое значение для переменной PORT. Когда приложение запустится, оно будет загружено на этот порт 6000 вместо 3000 по умолчанию. Другие доступные переменные окружения могут быть установлены следующим образом:

PORT=6000 HOST=127.0.0.10 npm run start

У этого метода есть один серьезный недостаток: переменные окружения должны устанавливаться при каждом запуске приложения. Это очень трудоемко и неэффективно при использовании различных сценариев. Эта проблема может быть также решена.

Все переменные окружения могут быть заданы в специальном конфигурационном файле. В этом случае необходимо создать файл в корне проекта с именем «.env.local» и задать в нем необходимые переменные окружения. При запуске проекта файл будет проанализирован и переменные окружения будут установлены автоматически.

Файл «.env.local» не может быть передан в систему управления версиями. В разных случаях могут использоваться разные среды. Этот файл создается для конкретной задачи и используется только во время разработки. Переменные окружения могут содержать некоторую важную информацию или конфиденциальные данные, поэтому настройки указанного файла не должны попадать в Git. Добавьте соответствующее правило в файл gitignore.

Полный список поддерживаемых переменных среды приведен в документации. Наиболее часто используются следующие:

* + HOST. Адрес хостинга, на котором запущен проект;
  + PORT. Порт для запуска сервера разработки;
  + HTTPS. Использование HTTPS по умолчанию;
  + BROWSER\_ARGS. Дополнительные аргументы для браузера, в котором приложение открывается после запуска;
  + DISABLE\_ESLINT\_PLUGIN. Активация/деактивация плагина eslint-webpack-plugin.

**Создание простого компонента React**

Давайте теперь создадим простой компонент React, чтобы понять, как различные концепции, которые мы обсуждали, сочетаются друг с другом. Откройте созданный с помощью CRA проект в редакторе кода и найдите в нем папку «src». Внутри нее создайте новый файл с именем «Greeting.js». Добавьте следующий код компонента:

import React from 'react';

const Greeting = () => {

  return <h1>Hello, React!</h1>;

}

export default Greeting;

В приведенном выше коде мы импортировали React и определили функциональный компонент под названием «Greeting.js». Компонент возвращает код «JSX», который отображает элемент <h1> с текстом «Hello, React!».

Чтобы использовать этот компонент, откройте src/App.js файл и замените его содержимое следующим:

import React from 'react';

import Greeting from './Greeting';

const App = () => {

  return <Greeting />;

}

export default App;

В приведенном выше коде мы импортировали наш компонент «Greeting.js» и использовали его в компоненте «App.js». Когда мы запустим приложение React, мы увидим сообщение "Hello, React!", отображаемое в браузере.

Давайте создадим пример регистрационной формы, используя React.js . Мы создадим компонент формы, который будет собирать имя пользователя, адрес электронной почты и пароль. Вот как вы можете это сделать:

1. Создайте новый файл с именем «RegistrationForm.js » внутри папки «src» проекта React.
2. Добавьте следующий код для определения компонента регистрационной формы:

import React, { useState } from 'react';

const RegistrationForm = () => {

  const [name, setName] = useState('');

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const handleSubmit = (e) => {

    e.preventDefault();

    // Here, you can perform validation or submit the form data to an API

    console.log('Form submitted:', { name, email, password });

    // Reset the form

    setName('');

    setEmail('');

    setPassword('');

  }

  return (

    <form onSubmit={handleSubmit}>

      <h2>Registration Form</h2>

      <label>

        Name:

        <input

          type="text"

          value={name}

          onChange={(e) => setName(e.target.value)}

          required

        />

      </label>

      <label>

        Email:

        <input

          type="email"

          value={email}

          onChange={(e) => setEmail(e.target.value)}

          required

        />

      </label>

      <label>

        Password:

        <input

          type="password"

          value={password}

          onChange={(e) => setPassword(e.target.value)}

          required

        />

      </label>

      <button type="submit">Register</button>

    </form>

  );

}

export default RegistrationForm;

В приведенном выше коде мы импортировали React и используем State из пакета 'react'. Мы использовали функцию (hook) userState() для управления состоянием формы. Мы объявили три переменные состояния: «name», «email» и «password», а также соответствующие им функции установки их значений (setName, setEmail, SetPassword).

Функция «handleSubmit» запускается при отправке формы. Она предотвращает отправку формы по умолчанию, выводит данные формы в консоли (в этом конкретном случае) и сбрасывает поля формы, устанавливая переменные состояния как пустые строки.

Внутри инструкции «return» мы отображаем элемент <form> с полями регистрационной формы: «name», «email» и «password». Каждое поле воода связано со своей соответствующей переменной состояния и обработчиком события «onChange», который обновляет состояние по мере ввода пользователем.

1. Чтобы использовать компонент «RegistrationForm.js», откройте "src/App.js » файл и замените его содержимое следующим кодом:

import React from 'react';

import RegistrationForm from './RegistrationForm';

const App = () => {

  return <RegistrationForm />;

}

export default App;

В приведенном выше коде мы импортировали компонент «RegistrationForm.js» и отрисовали его внутри компонента «App».

1. Запустите приложение React командой «npm start» в терминале. Теперь вы должны увидеть регистрационную форму в браузере.

Когда вы заполните поля формы и отправите ее, вы заметите, что данные формы заносятся в консоль, а поля формы очищаются.

Давайте добавим несколько стилей CSS в наш пример регистрационной формы. Мы создадим отдельный CSS-файл и импортируем его в компоненты. Вот как мы можем это сделать:

1. Создайте новый файл с именем «RegistrationForm.css» внутри папки «src» проекта React.
2. Добавьте следующий CSS-код для оформления регистрационной формы:

.registration-form {

    max-width: 400px;

    margin: 0 auto;

    padding: 20px;

    background-color: #f4f4f4;

    border: 1px solid #ccc;

    border-radius: 5px;

  }

  .registration-form h2 {

    margin-top: 0;

    text-align: center;

  }

  .registration-form label {

    display: block;

    margin-bottom: 10px;

  }

  .registration-form input[type="text"],

  .registration-form input[type="email"],

  .registration-form input[type="password"] {

    width: 100%;

    padding: 8px;

    font-size: 16px;

    border-radius: 3px;

    border: 1px solid #ccc;

  }

  .registration-form button[type="submit"] {

    width: 100%;

    padding: 10px;

    font-size: 16px;

    background-color: #4caf50;

    color: white;

    border: none;

    border-radius: 3px;

    cursor: pointer;

  }

  .registration-form button[type="submit"]:hover {

    background-color: #45a049;

  }

В приведенном выше CSS-коде мы добавили стили для класса «.registration-form», а также специальные стили для заголовков, меток, полей ввода и кнопки отправки.

1. Затем откройте «RegistrationForm.js» и импортируйте CSS-файл, добавив следующую строку вверху:

import './RegistrationForm.css';

Наконец, примените класс «.registration-form» к элементу <form> в функции «return» компонента «RegistrationForm.js» следующим образом:

return (

    <form className="registration-form" onSubmit={handleSubmit}>

      {/\* Остальной код формы \*/}

    </form>

  );

Сохраните изменения. Если запустить приложение React на данном этапе, мы увидим, что регистрационная форма оформлена в соответствии со стилями CSS, которые мы определили.

С помощью стилей CSS форма будет иметь центрированный макет, светло-серый фон и рамку. Поля ввода и кнопка отправки будут иметь согласованные стили.

Не стесняйтесь изменять стили CSS в соответствии с вашими предпочтениями или добавлять дополнительные стили для дальнейшей настройки внешнего вида регистрационной формы.

**Документация**

Более подробную информацию вы можете найти в официальной документации. Документацию можно найти на следующем ресурсе: "https://create-react-app.dev ". Интересные разделы: «Расширенное использование», «Внутренняя интеграция».

**Заключение**

CRA - это инструмент для подготовки основы проекта с использованием React. Вы получаете готовый проект и не задумываетесь о настройке webpack, babel и других вспомогательных модулей. Все это настраивается "из коробки" и, при необходимости, конфигурация может быть изменена.

# Основы Pythion

Python - это высокоуровневый язык общего назначения, который используется для решения современных компьютерных задач. В наши дни у людей сложилось неправильное представление о Python. Они думают, что python - это язык анализа данных и машинного обучения. Однако на самом деле Python - это язык программирования общего назначения. Гвидо ван Россум изобрел язык программирования Python в начале 1990-х годов.

Python - универсальный и популярный язык программирования, известный своей простотой и удобочитаемостью. Независимо от того, являетесь ли вы новичком или опытным программистом, Python может стать ценным инструментом в вашем арсенале. В этой лекции мы рассмотрим фундаментальные концепции Python и рассмотрим несколько примеров, чтобы закрепить наше понимание.

Язык программирования Python находит свое применение в различных областях. Вот некоторые области, в которых можно использовать Python:

1. Веб-разработка: Python предлагает надежные фреймворки веб-разработки, такие как Django и Flask. Эти фреймворки предоставляют инструменты и библиотеки для создания масштабируемых и многофункциональных веб-приложений. Простота Python и доступность фреймворков делают его отличным выбором для быстрой веб-разработки.
2. Анализ и визуализация данных: Python стал языком де-факто для анализа и визуализации данных. Такие библиотеки, как NumPy, Pandas и Matplotlib, предоставляют мощные инструменты для обработки и анализа данных, создания визуализаций и выполнения статистических вычислений. Кроме того, Python хорошо интегрируется с другими инструментами, связанными с данными, такими как Jupyter Notebook и Apache Spark.
3. Машинное обучение и искусственный интеллект: Простота Python и обширные библиотеки, такие как Scikit-learn, TensorFlow и PyTorch, делают его предпочтительным языком для задач машинного обучения и искусственного интеллекта. Эти библиотеки предоставляют алгоритмы, модели и инструменты для разработки и обучения моделей машинного обучения, приложений для обработки естественного языка (NLP), систем компьютерного зрения и многого другого.
4. Научные вычисления: Python широко используется в научных вычислениях и исследованиях. Такие библиотеки, как SciPy и NumPy, предоставляют функции и процедуры для численных вычислений, оптимизации, линейной алгебры и обработки сигналов. Интеграция с Jupyter Notebook еще больше повышает его удобство в научных исследованиях и экспериментах.
5. Автоматизация и написание сценариев: простой для чтения синтаксис Python и богатый набор библиотек делают его отличным выбором для автоматизации повторяющихся задач и написания сценариев. Начиная с задач системного администрирования и заканчивая обработкой файлов, Python может упростить и рационализировать различные рабочие процессы автоматизации.
6. Интернет вещей (IoT): Простота Python, доступность библиотек и поддержка микроконтроллеров делают его подходящим для приложений Интернета вещей. Python можно использовать для разработки встроенного ПО, управления датчиками и исполнительными механизмами, а также для взаимодействия с устройствами Интернета вещей, что делает его универсальным языком для создания решений Интернета вещей.
7. Разработка игр: Хотя Python, возможно, и не является лучшим выбором для высокопроизводительной разработки игр, его можно использовать для создания простых игр, прототипов и игровых сценариев. Библиотеки, такие как Pygame, предоставляют функциональные возможности для разработки графики и игр, позволяя разработчикам создавать 2D-игры и интерактивные возможности.
8. Настольные приложения: Python можно использовать для разработки настольных приложений с помощью таких фреймворков, как PyQt и Tkinter. Эти фреймворки позволяют разработчикам быстро создавать кроссплатформенные графические приложения.
9. Написание сценариев для программного обеспечения и DevOps: Python обычно используется для написания сценариев в задачах разработки программного обеспечения и DevOps. Его можно использовать для автоматизации тестирования программного обеспечения, процессов развертывания и других повторяющихся задач, связанных с разработкой программного обеспечения и управлением инфраструктурой.
10. Образование и обучаемость: Простота, удобочитаемость и обширная документация Python делают его идеальным языком для преподавания концепций программирования и информатики. Многие учебные заведения и онлайн-курсы используют Python в качестве основного языка для вводных курсов программирования.

Компании, использующие Python в своих проектах:

* + Google
  + Facebook
  + Instagram
  + Spotify
  + Netflix
  + Dropbox

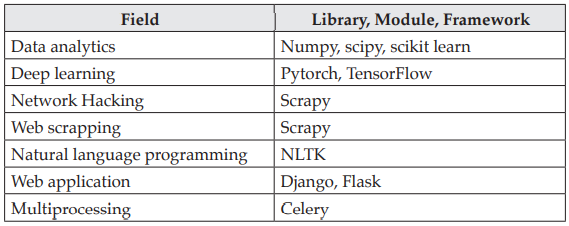
**Есть много причин выбрать именно этот язык программирования.**

**Этот язык является многоцелевым**

Python - это язык общего назначения. Разработчики используют python для анализа данных, машинного обучения, искусственного интеллекта, веб-приложений, мониторинга сети, написания ETL-сценариев, взлома и многого другого.

**Обширная поддержка библиотек и модулей**

У Python очень большое сообщество разработчиков. В Python доступны различные библиотеки и модули:



**Простота чтения кода**

Код на Python легко читается и понятен. Он не содержит большого количества синтаксиса, как Java или C#. Python использует табуляцию для управления блоками кода, вот почему отступы являются важной частью программирования на Python. Табуляция облегчает чтение кода. Синтаксис Python довольно краток. Это помогает новичкам быстро освоить основы этого языка с меньшим умственным напряжением и быстро осваивать более сложные темы.

**Он объектно-ориентирован**

Python обладает мощью объектно-ориентированного программирования (ООП). Однако мы можем написать программу без определения каких-либо классов.

**Независимость от платформы**

Код Python не зависит от платформы и может быть запущен в любой операционной системе. Все, что вам нужно сделать, это скопировать и вставить код.

**Python - динамический и строго типизированный**

При динамической типизации тип данных переменной интерпретируется во время выполнения. В Python нет необходимости определять тип данных типа «Int», «Float» и т.д. Следующий пример должен дать больше разъяснений относительно переменных:

>>> a = 10

>>> type(a)

<class 'int'>

>>> b = 10.9

>>>type(b)

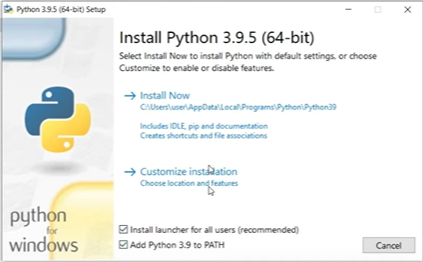
<class 'float'>

>>>

При строгой типизации тип переменной не изменяется во время выполнения программы. Если a = 10, то значение остается равным 10 на протяжении всего выполнения, пока мы не переназначим значение этой переменной.

**Установка Python**

Перейдите на официальный сайт <http://python.org>, скачайте и установите последнюю версию Python на свой компьютер (сегодня это 3.10.3). Во время установки отметьте флажок "Add Python 3.10 в PATH".



Далее проверьте, успешно ли прошла установка Python на компьютере. Для этого откройте командную строку (меню Пуск->Поиск->cmd) и введите команду:

python –version

Нажмите "Enter". Если вы видите версию python, это означает, что установка прошла успешно. Мы также можем проверить, правильно ли работает интерпретатор Python. Для того чтобы сделать это, мы можем ввести команду и нажать "Enter":

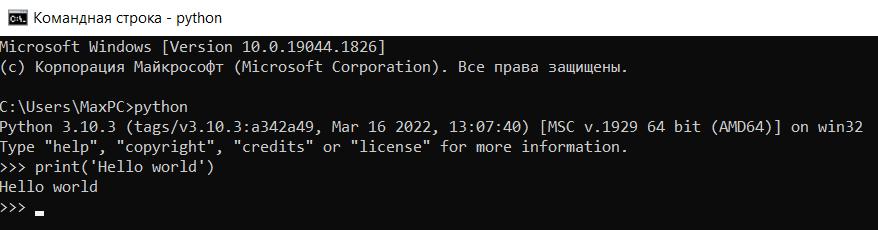
python

После этого вы можете ввести инструкцию на Python, например:

print('Hello world!')

и нажать "Enter".

Если все работает правильно, вы увидите результат выполнения команды, т.е. строку, отображаемую на экране:



Вы можете писать код на Python в различных редакторах кода. Все известные сегодня редакторы кода и среды разработки имеют встроенную спецификацию для этого языка.

Среди редакторов и сред разработки наиболее предпочтительными являются следующие:

* + Sublime Text
  + Visual Studio Code
  + Jupyter Notebook
  + PyCharm
  + Spyder
  + Atom
  + Notepad++ (this option is the least preferred)

Более подробно о языке Python и его использовании можно прочитать на официальном сайте <https://www.python.org/doc/>.

**Синтаксис Python**

Функция print(): Используется для отображения выходных данных на консоли. Пример:

print("Hello, World!")

**Переменные и типы данных**: Переменные – это своего рода контейнеры для хранения данных. Python использует динамическую типизацию, что означает, что вам не нужно явно объявлять тип переменной. Пример:

message = "Hello, Python!"

print(message)

Типы данных: Python поддерживает различные типы данных, включая:

* + Числа: int, float, complex;
  + Строки: str;
  + Логическое значение: bool (истина или ложь);
  + Списки, кортежи и наборы;
  + Словари c. Преобразование типов: изменение типа данных переменной.

Пример:

x = 10

y = float(x)

**Операторы и выражения**:

a. Арифметические операторы: +, -, \*, /, // (деление с округлением), % (по модулю), \*\* (возведение в степень);

b. Операторы сравнения: <, >, <=, >=, == (равенство), != (неравенство);

c. Логические операторы: and, or, not

d. Выражения: Объединение переменных, значений и операторов.

Пример:

x = 5

y = 3

z = x + y \* 2

**Поток управления** (Control Flow):

a. Условные операторы: if, elif, else.

Пример:

x = 10

if x > 0:

print("Positive")

elif x < 0:

print("Negative")

else:

print("Zero")

b. Циклы:

Цикл «for»: Выполняет итерацию по последовательности или итератору (объекту с возможностью перебора). Пример:

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

Цикл «while»: повторяет блок кода до тех пор, пока условие имеет значение true.

Пример:

x = 0

while x < 5:

print(x)

x += 1

**Функции:**

a. Определение функций: Создание повторно используемых блоков кода.

Пример:

def greet(name):

print("Hello, " + name + "!")

greet("Alice")

b. Оператор Return: Возвращает значение из функции.

Пример:

def square(x):

return x \*\* 2

result = square(5)

print(result)

**Преимущества Python:**

1. Удобочитаемость и простота: Python использует чистый и понятный синтаксис. Он использует отступы и пробелы для разделения блоков кода, что делает его легко читаемым и снижает вероятность синтаксических ошибок. Эта функция делает Python отличным выбором для начинающих разработчиков.

2. Универсальность и переносимость: Python - это универсальный язык, который можно использовать для широкого спектра приложений, включая веб-разработку, анализ данных, машинное обучение, научные вычисления, автоматизацию и многое другое. Он работает на нескольких платформах, включая Windows, macOS, Linux, и имеет большую стандартную библиотеку и сторонние пакеты, предоставляя разработчикам обширную функциональность.

3. Большое и активное сообщество: Python обладает обширным и динамичным сообществом разработчиков по всему миру. Это сообщество вносит свой вклад в разработку библиотек, фреймворков и инструментов, что делает Python постоянно развивающимся языком. Доступность ресурсов, документации и поддержки сообщества гарантирует, что разработчики смогут быстро найти помощь и решения своих проблем.

4. Возможности интеграции: Python может легко интегрироваться с другими языками, такими как C, C++, Java и .NET. Эта функция позволяет разработчикам использовать существующие кодовые базы и использовать преимущества простоты и производительности Python при взаимодействии с модулями или компонентами, написанными на других языках.

5. Скорость разработки: Простота и выразительность Python позволяют разработчикам писать код быстро и эффективно. Его обширные библиотеки и фреймворки, такие как Django и Flask для веб-разработки, NumPy и pandas для анализа данных, а также TensorFlow и PyTorch для машинного обучения, предоставляют высокоуровневые абстракции, которые ускоряют разработку, сокращая время и усилия, необходимые для создания надежных приложений.

**Недостатки Python:**

1. Производительность: Python является интерпретируемым языком, что означает, что он может быть медленнее по сравнению с компилируемыми языками, такими как C или Java. Несмотря на то, что производительность Python значительно улучшилась за прошедшие годы, он может оказаться не лучшим выбором для задач с высокой интенсивностью вычислений. Однако разделы, критически важные для производительности, можно оптимизировать с помощью собственных расширений или с помощью библиотек, написанных на языках более низкого уровня.

2. Глобальная блокировка интерпретатора (GIL): В Python есть глобальная блокировка интерпретатора, которая позволяет только одному потоку выполнять байт-код Python одновременно. Это ограничение может снизить производительность многопоточных программ на Python, поскольку оно препятствует истинному параллелизму на нескольких ядрах. Однако Python предоставляет многопроцессорные библиотеки, которые могут преодолеть это ограничение, используя несколько процессов вместо потоков.

3. Разработка мобильных устройств и игр: Хотя Python широко используется в различных областях, он может оказаться неоптимальным выбором для разработки мобильных приложений или высокопроизводительных игр. Мобильные платформы часто полагаются на определенные языки программирования, такие как Swift (iOS) и Java/Kotlin (Android). Аналогичным образом, для разработки игр часто требуются языки, ориентированные на производительность, такие как C++.

4. Потребление памяти: Динамическая типизация Python и высокоуровневые абстракции реализуются за счет увеличения потребления памяти по сравнению с языками более низкого уровня. Эти накладные расходы могут ограничить его пригодность для сред с ограниченным объемом памяти или приложений, которым необходимо обрабатывать огромный объем данных.

5. Совместимость версий: Python претерпел несколько крупных выпусков, наиболее заметными из которых являются Python 2.x и Python 3.x. В Python 3 были внесены значительные улучшения, но он не имеет обратной совместимости с Python 2. Хотя Python 2 устарел, некоторые устаревшие базы кода или сторонние библиотеки все еще могут полагаться на него, создавая проблемы совместимости для разработчиков, переходящих с Python 2 на Python 3.

Стоит отметить, что упомянутые выше преимущества и недостатки следует рассматривать в контексте конкретных вариантов использования и требований проекта. Сильные стороны Python часто перевешивают его ограничения, что делает его популярным выбором среди разработчиков по всему миру.

Это всего лишь несколько примеров того, где обычно используется Python, но его универсальность позволяет применять его и во многих других областях. Обширная экосистема Python и активное сообщество обеспечивают непрерывный рост и разработку новых библиотек и инструментов для различных приложений.

# Django Framework

Django - это мощный и популярный фреймворк, который позволяет быстро и эффективно создавать веб-приложения. Он написан на Python и соответствует архитектурному шаблону Model-View-Controller (MVC), что упрощает организацию и обслуживание кода.

Django разработан некоммерческой организацией «Django Software Foundation».

**Преимущества Django:**

• Быстрая разработка: встроенная функциональность и соглашения Django позволяют разработчикам быстро создавать веб-приложения.

• Масштабируемость: Масштабируемость Django делает его подходящим для работы с крупномасштабными приложениями с высоким трафиком.

• Безопасность: Django включает встроенные функции безопасности, такие как защита от распространенных веб-уязвимостей.

• Универсальность: Django можно использовать для создания различных типов приложений, от простых блогов до сложных платформ электронной коммерции.

**Недостатки Django:**

• Кривая обучения: Django имеет более крутую кривую обучения по сравнению с облегченными фреймворками. Освоение его обширного набора функций может занять некоторое время.

• Накладные расходы: всеобъемлющий характер Django сопряжен с дополнительными накладными расходами. Возможно, это не лучший выбор для небольших, простых проектов.

• Наличие соглашений: Django следует определенным соглашениям, которые могут ограничить гибкость для разработчиков, предпочитающих большую свободу в кодировании.

**Django был разработан в период с 2003 по 2005 год:**

• 2003: Разработка фреймворка была начата Adrian Golovati и Simon Willison в качестве внутреннего проекта в газете;

• 2005: Веб-фреймворк с открытым исходным кодом, выпущенный в 2005 году и названный в честь джазового гитариста Джанго Рейнхардта;

• 2008: Первый релиз (версия 1.0) состоялся в сентябре 2008 года.

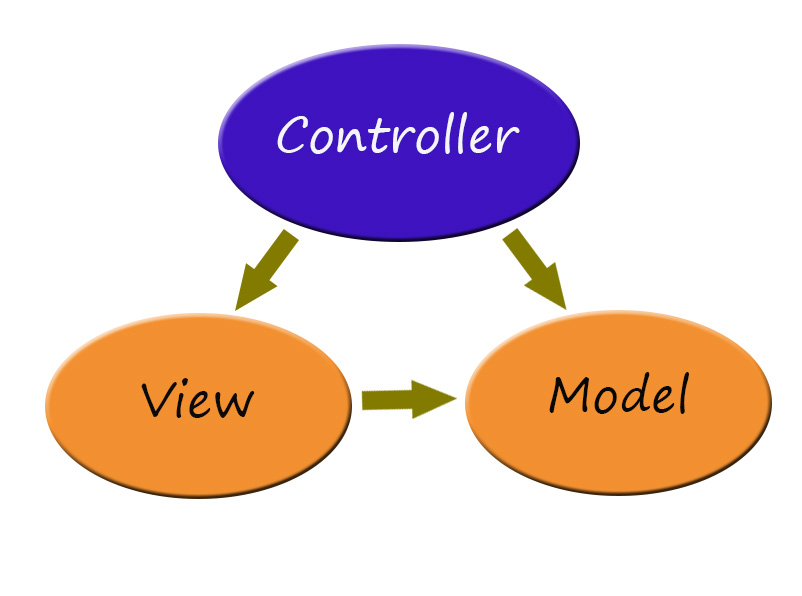
В настоящее время это развивающийся совместный проект с открытым исходным кодом, в котором участвуют разработчики по всему миру.

**Шаблон MVC**

Как и большинство современных фреймворков, Django поддерживает шаблон MVC.

**Что такое шаблон Model-View Controller (MVC)?**

Когда мы говорим о приложениях, которые предоставляют пользовательский интерфейс (веб- или десктопный), мы обычно имеем в виду архитектуру MVC. Архитектурный шаблон MVC делит логику приложения на три основные части: модель, представление, контроллер.



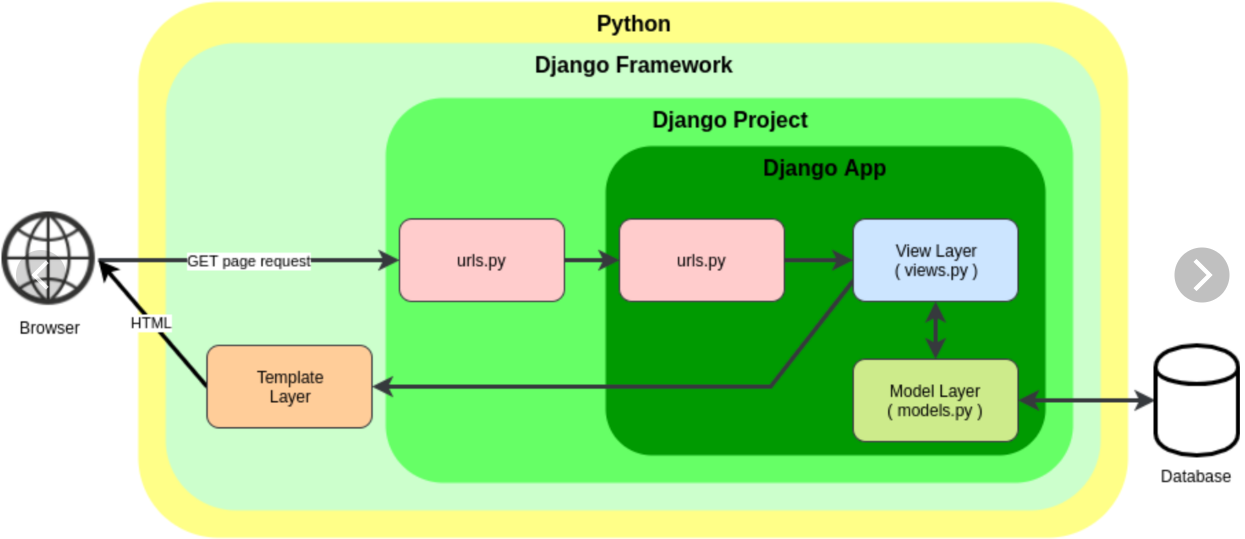
* + **Модели:** Модели определяют структуру и поведение данных в приложении. Они представляют собой таблицы базы данных и определены в «models.py» файле в каталоге приложения. Например, простая модель для записи в блоге может содержать такие поля, как "заголовок", "содержание" и "дата публикации".
  + **Представления:** Представления обрабатывают логику, лежащую в основе каждого URL-адреса в вашем приложении. Они обрабатывают запросы, взаимодействуют с моделями и отображают шаблоны. Представления определены в «views.py» файле в каталоге приложения.
  + **Шаблоны:** Шаблоны определяют структуру и макет пользовательского интерфейса. Обычно это HTML-файлы, которые Django заполняет динамическим содержимым. Шаблоны хранятся в каталоге с именем "templates" в каждом каталоге приложений.

**Django MVC - MVT Pattern**

Django использует слегка измененный шаблон, называемый Model-View-Template (MVT).

Шаблон Model-View-Template (MVT) немного отличается от MVC. На самом деле, основное различие между этими двумя шаблонами заключается в том, что Django сам выполняет часть работы контроллера. Контроллер здесь - это код, который управляет взаимодействием между моделью и представлением и создает шаблон в качестве выходных данных. Шаблон представляет собой HTML-файл, смешанный с Django Template Language (DTL).

Следующая диаграмма иллюстрирует, как каждый из компонентов шаблона MVT взаимодействует друг с другом для выполнения запроса пользователя:



<https://github.com/fariba-Dk/django-urls-views-templates>

**Среда Django**

Среда разработки Django содержит такие модули, как интерпретатор Python, фреймворк Django и база данных.

Django полностью написан на Python, поэтому для начала вам необходимо установить Python. Его можно скачать с официального сайта <http://python.org>.

Установив Python, мы можем приступить к установке Django.

Существует несколько различных методов установки Django, каждый из которых имеет свои преимущества в определенных ситуациях:

• *Установка Django глобально с помощью «pip»*. Инструмент «pip» - это менеджер пакетов Python. Его можно использовать для выполнения общесистемной установки Django. Обычно он предоставляет последнюю доступную версию пакета. Однако глобальная (общесистемная) установка всегда менее гибка.

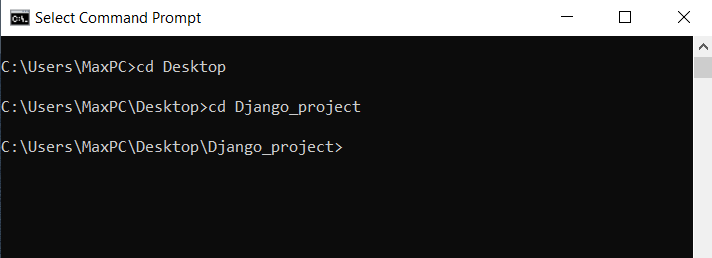
• *Установка через pip в Virtualenv*. Пакет virtualenv позволяет создавать автономные среды для различных проектов. Используя эту технологию, вы можете установить Django в каталог проекта, не затрагивая систему в целом. Это позволяет устанавливать индивидуальные настройки для каждого проекта. Виртуальная среда - это более гибкий способ установки пакета.

• *Установка версии для разработки с помощью Git*. Чтобы установить последнюю версию для разработки вместо стабильной версии, вам необходимо получить код из репозитория git. Это обеспечит программу новейшими функциями и исправлениями; вы можете установить эту версию как глобально, так и локально. Но имейте в виду: версии для разработки нестабильны.

Чтобы установить Django в виртуальной среде, необходимо выполнить следующие действия (используем командную строку Windows в качестве примера):

Для работы с виртуальной средой на Python используется встроенный модуль "venv".

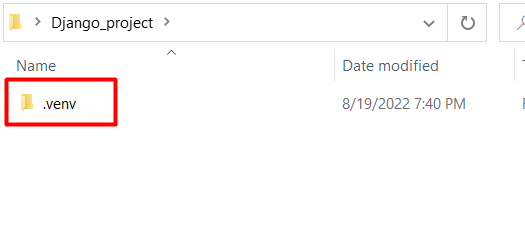
Давайте создадим папку "Django-project" на рабочем столе и перейдем к ней с помощью команды cd:



Затем внутри текущей папки запустите следующую команду:

python -m venv .venv

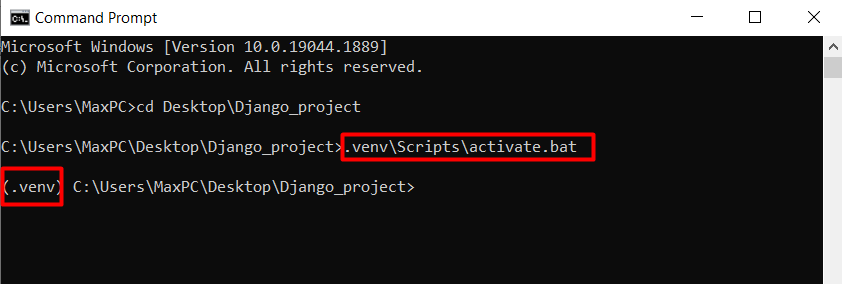
Модулю **venv** передается имя среды, которое в данном случае будет "**.venv**". Вы можете указать любое имя для среды. После выполнения команды в текущей папке будет создан каталог "**.venv**".



Для работы с установленной виртуальной средой она должна быть активирована. Чтобы сделать это, запустите следующую команду (активация в командной строке Windows):

.venv/Scripts/activate.bat

В последней строке терминала появятся символы "(.venv)", которые сообщают нам о том, что виртуальная среда активирована.



***Примечание:*** *Виртуальная среда должна активироваться каждый раз, когда вы работаете над определенным проектом. После завершения работы вам следует деактивировать виртуальную среду командой: "deactivate".*

Установите фреймворк Django в созданную и активированную виртуальную среду. Чтобы сделать это, введите следующую команду и нажмите Enter:

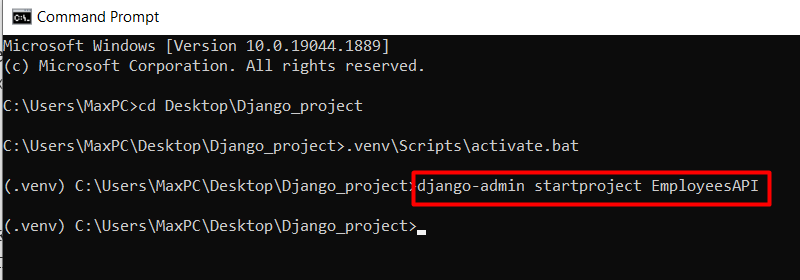
python -m pip install Django

Фреймворк будет установлен локально в виртуальной среде.

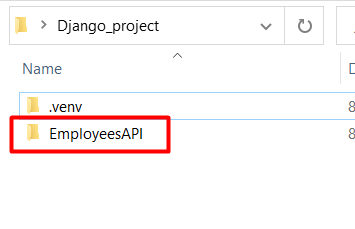
Django также устанавливает специальную утилиту **django-admin** в каталоге виртуальной среды, которая предоставляет набор команд для управления проектом Django.

Давайте создадим первый проект, который будет находиться внутри папки с виртуальной средой. Чтобы сделать это, введите следующую команду:

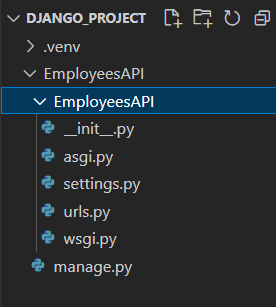
django-admin startproject EmployeesAPI



После этого вы увидите новую папку с созданным проектом:



Давайте откроем проект в редакторе VS Code. Структура проекта будет выглядеть следующим образом:



Давайте посмотрим на назначение файлов в созданном проекте:

* + **\_init\_.py** это пустой файл, который указывает на то, что проект является проектом Python или модулем Python;
  + **asgi.py** является точкой входа для серверов asgi (Asynchronous Server Gateway Interface);
  + **wsgi.py** - точка входа для серверов wsgi (Web Server Gateway interface);
  + **urls.py** это файл, в котором объявлены все маршруты, необходимые для проекта;
  + **settings.py** это файл, содержащий все необходимые параметры конфигурации проекта;
  + **manage.py** это служебный файл, который взаимодействует с проектом Django.

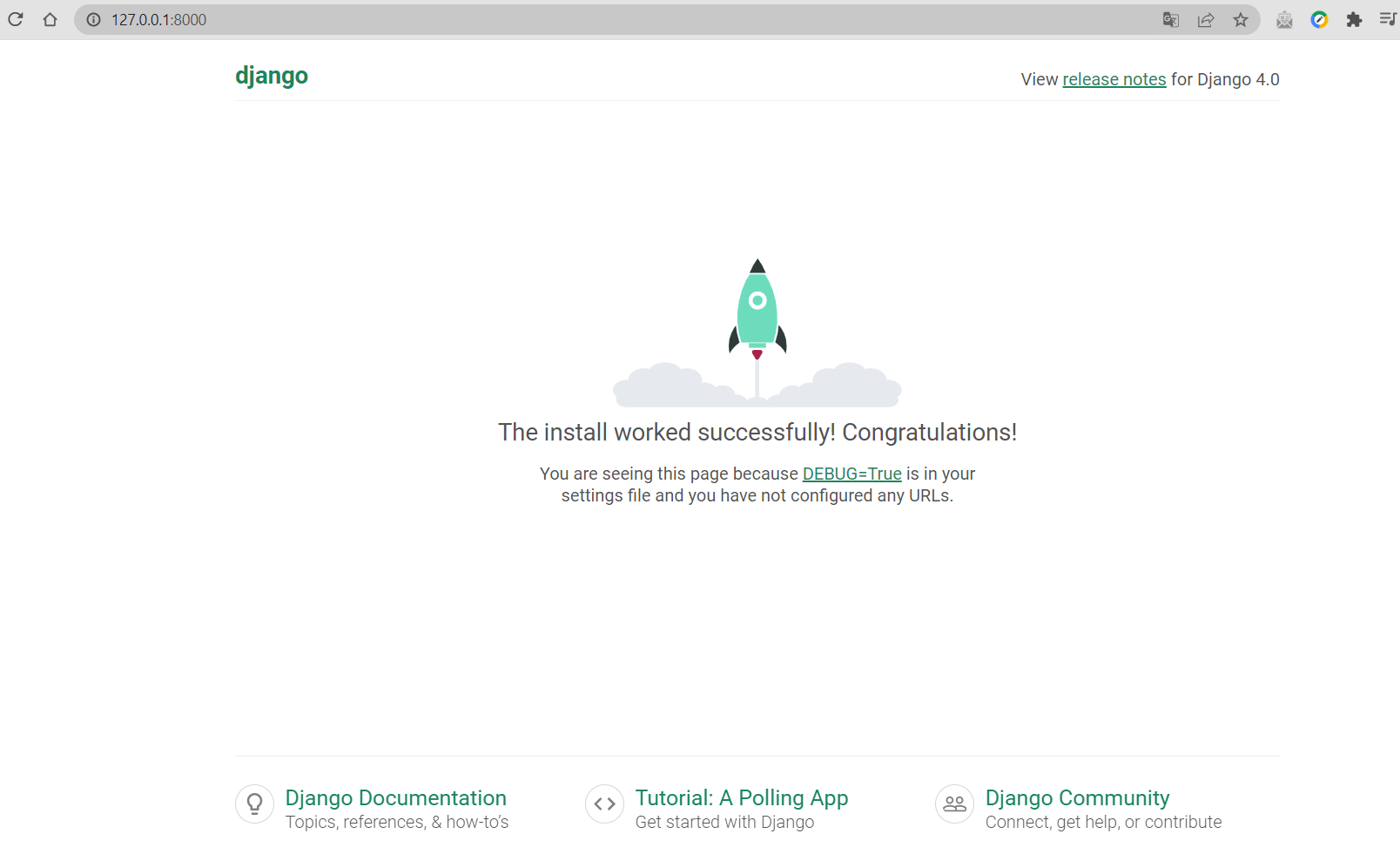
Теперь запустим проект на выполнение и посмотрим на его работу. Это можно сделать, выполнив следующую команду:

python manage.py runserver

Приложение будет запущено на порту 8000.



Скопируйте URL-адрес и вставьте его в строку поиска браузера:



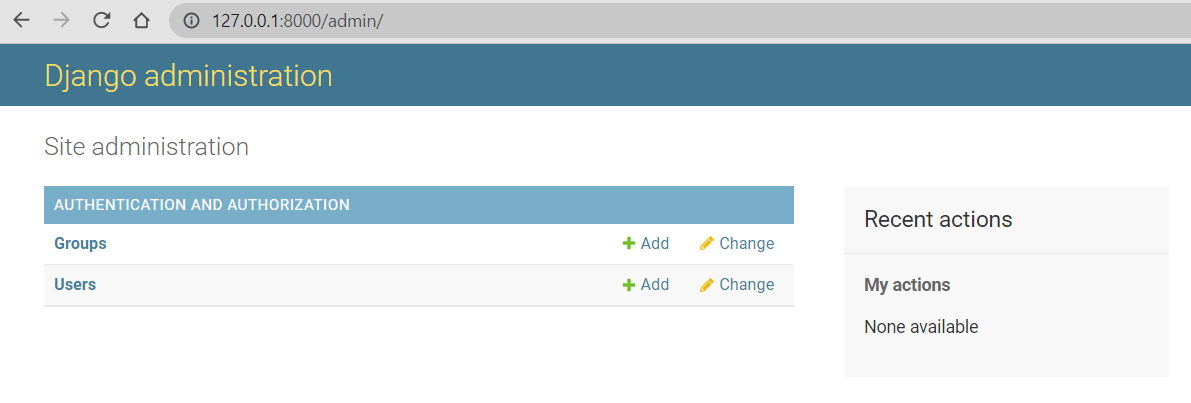
После запуска проекта мы увидим базовую страницу Django.

Если добавить к адресу «/admin», мы сможем войти в панель администратора проекта Django.

Чтобы войти в систему, у вас должна быть учетная запись. Чтобы создать учетную запись с полным доступом к проекту и со всеми необходимыми разрешениями (суперпользователь), необходимо запустить следующую команду в терминале из общей папки проекта, в которой находится **manage.py** файл находится:

python manage.py createsuperuser

Система попросит ввести новое имя пользователя и пароль. После создания учетной записи вы сможете войти в панель администрирования с указанными учетными данными.



Django обеспечивает прочную основу для создания веб-приложений, а его универсальность и масштабируемость делают его популярным выбором среди разработчиков.

Чтобы найти дополнительную информацию о фреймворке Django, вы можете обратиться к следующим ресурсам:

1. Официальная документация Django: Официальная документация Django является отличной отправной точкой. Он предоставляет исчерпывающую и подробную информацию о функциях, концепциях Django и о том, как их использовать. Вы можете получить доступ к документации по адресу <https://docs.djangoproject.com/>.
2. Веб-сайт проекта Django: Веб-сайт проекта Django (<https://www.djangoproject.com/>) предлагает различные ресурсы, включая учебные пособия, документацию, форумы сообщества и ссылки на дополнительные учебные материалы.
3. Сообщество Django: Сообщество Django активно и оказывает поддержку. Вы можете участвовать в обсуждениях, задавать вопросы и обращаться за помощью к другим разработчикам Django. Официальный форум Django <https://forum.djangoproject.com/> и форум на <https://www.reddit.com/r/django/> являются популярными платформами для взаимодействия с сообществом.
4. Онлайн-руководства и курсы: Существует множество доступных онлайн-руководств и курсов, которые подробно охватывают Django. Платформы, подобные Udemy (<https://www.udemy.com/>), Coursera (<https://www.coursera.org/>), и <https://realpython.com/> предлагают курсы по Django, которые варьируются от начального до продвинутого уровней.
5. Книги: Например: "Two Scoops of Django" by Daniel Roy Greenfeld and Audrey Roy Greenfeld, "Django for Beginners" by William S. Vincent. Эти книги содержат исчерпывающие рекомендации и лучшие практики для разработки на Django.
6. GitHub: Изучение проектов Django с открытым исходным кодом на таких платформах, как GitHub (https://github.com /) может быть отличным способом учиться на примерах из реального мира. Вы можете найти хранилища кода, внести свой вклад в проекты и получить представление о том, как Django используется в различных сценариях.

# API

API (Application Programming Interfaces) играют жизненно важную роль в разработке современного программного обеспечения, обеспечивая связь и взаимодействие между различными программными системами, приложениями и сервисами.

1. **Что такое API?**

API, или программный интерфейс приложения, - это набор правил и протоколов, который позволяет различным программным приложениям взаимодействовать друг с другом. API определяют, как должны взаимодействовать программные компоненты, какими данными можно обмениваться, а также методы или функции, которые можно использовать для доступа к этим данным и манипулирования ими.

1. **API компоненты:** 
   * Конечные точки (Endpoints): Конечная точка - это определенный URL-адрес или URI, который представляет уникальный ресурс или функциональность, предоставляемую API. Он служит точкой входа для доступа к API и взаимодействия с ним;
   * Методы запроса (Request Methods): API-интерфейсы используют стандартные методы HTTP, такие как GET, POST, PUT, DELETE и т.д., для определения типа операции, которая должна быть выполнена с ресурсом;
   * Параметры запроса (Request Parameters): API-интерфейсы часто принимают параметры как часть запроса, чтобы указать дополнительную информацию или изменить поведение вызова API;
   * Ответ (Response): Ответ от вызова API содержит данные или информацию о состоянии, запрошенные клиентским приложением. Обычно он отформатирован в определенной структуре, такой как JSON или XML.
2. **Типы API:** 
   * WebAPI: Эти API предназначены для предоставления функциональных возможностей веб-сервисов через Интернет, позволяя разработчикам получать доступ к таким сервисам, как данные о погоде, карты, платформы социальных сетей и многое другое, и использовать их;
   * Библиотечные или API фреймворков: Эти API предоставляются языками программирования, фреймворками или библиотеками и предлагают готовые функции и классы для упрощения разработки программного обеспечения. Примеры включают стандартную библиотеку Python, Java API и jQuery API;
   * API операционной системы: Эти API позволяют разработчикам взаимодействовать с базовой операционной системой, получая доступ к системным ресурсам, управлению файлами, сетевым операциям и многому другому;
   * API баз данных: Эти API облегчают взаимодействие с базами данных, позволяя разработчикам выполнять такие операции, как запрос, обновление данных и управление ими;
3. **Примеры использования API:** 
   * API социальных сетей: такие платформы, как Twitter, Facebook и Instagram, предоставляют API, которые позволяют разработчикам интегрировать функции социальных сетей в свои собственные приложения. Например, получение профилей пользователей, публикация твитов или обновлений статуса, а также выборка новостных лент;
   * API платежных шлюзов: такие компании, как PayPal, Stripe или Braintree, предоставляют API для обработки платежей в приложениях электронной коммерции, позволяя разработчикам безопасно принимать платежи;
   * API-интерфейсы данных о погоде: такие сервисы, как OpenWeatherMap, предлагают API-интерфейсы, которые предоставляют разработчикам данные о погоде в режиме реального времени и прогноз погоды для интеграции в их приложения;
   * Картографические API: Такие поставщики, как Google Maps, Mapbox или OpenStreetMap, предлагают API для доступа к картографическим данным и службам геолокации, что позволяет разработчикам встраивать интерактивные карты или создавать приложения, основанные на местоположении.

API-интерфейсы действуют как мосты между различными программными системами, обеспечивая плавную интеграцию и коммуникацию. Понимание основных концепций API крайне важно для разработчиков, чтобы использовать широкий спектр функциональных возможностей, предлагаемых различными API.

# REST API

REST API, или Representational State Transfer API, - это архитектурный стиль и подход к проектированию сетевых приложений. Он широко используется в веб-разработке для создания масштабируемых API, не зависящих от состояния и легко используемых клиентами. REST API использует существующий протокол HTTP и его методы для обеспечения связи и передачи данных между клиентскими приложениями и серверами.

**Ключевые понятия REST API:**

1. Resources: REST API предоставляют ресурсы, которые являются ключевыми сущностями или объектами, с которыми имеет дело API. Ресурсом может быть все, что может быть однозначно идентифицировано, например статья, пользователь, продукт или любой другой объект данных. Каждый ресурс обычно представлен уникальным URL-адресом или конечной точкой.
2. HTTP Verbs/Methods: REST API используют HTTP-методы (также известные как verbs) для определения операций, которые могут выполняться с ресурсами. Обычно используемыми HTTP-методами в REST API являются:
   * GET: Извлекает представление ресурса или коллекции ресурсов.
   * POST: Создать новый ресурс;
   * PUT: Обновить существующий ресурс или создать новый ресурс, если он не существует;
   * DELETE: Удалить ресурс.
3. Единый идентификатор ресурса (URI): REST API используют единые идентификаторы ресурсов (URI) для уникальной идентификации ресурсов. URI - это URL-адреса или конечные точки, которые клиенты используют для доступа к ресурсам и управления ими. Например, https://api.example.com/users может быть URI для доступа к коллекции пользователей.
4. Stateless Communication: REST API не имеют состояния, это означает, что каждый запрос от клиента к серверу должен содержать всю необходимую информацию, чтобы сервер мог понять и обработать запрос. Сервер не хранит никакой информации или состояния о предыдущих запросах клиента. Этот принцип проектирования улучшает масштабируемость и упрощает реализацию на стороне сервера.
5. Representation of Resources: В REST API ресурсы представлены в определенном формате, таком как JSON (объектная нотация JavaScript) или XML (расширяемый язык разметки). Эти форматы определяют структуру и представление данных ресурсов, которыми обмениваются клиент и сервер.

**Числовые коды - HTTP statuses**

Каждый запрос REST API сообщает о результатах работы числовыми кодами - HTTP-статусами.

Коды состояния HTTP - это трехзначные цифровые коды, которые возвращаются сервером в ответ на запрос клиента, чтобы указать статус запроса или способность сервера выполнить его. Эти коды являются неотъемлемой частью протокола HTTP и предоставляют ценную информацию о результатах HTTP-запроса. Вот некоторые часто встречающиеся коды состояния HTTP:

1. Информационный (1xx):
   * 100 Continue: Сервер подтверждает начальную часть запроса и ожидает, что клиент выполнит остальную часть.
   * 101 Switching Protocols: Сервер соглашается переключать протоколы по запросу клиента.
2. Успех (2xx):
   * 200 OK: запрос был выполнен успешно, и сервер возвращает запрошенный контент.
   * 201 Created: запрос привел к успешному созданию нового ресурса.
   * 204 No Content: Сервер успешно обработал запрос, но возвращаемого содержимого нет.
3. Перенаправление (3xx):
   * 301 Moved Permanently: Запрошенный ресурс был безвозвратно перемещен в новое местоположение.
   * 302 Found: Запрошенный ресурс был временно перемещен в другое местоположение.
   * 304 Not Modified: Кэшированная версия запрошенного ресурса клиента по-прежнему действительна, и нет необходимости передавать ее повторно.
4. Ошибки клиента (4xx):
   * 400 Bad Request: Сервер не может обработать запрос из-за неправильного синтаксиса или недопустимых параметров.
   * 401 Unauthorized: Клиенту необходимо предоставить действительные учетные данные для аутентификации, чтобы получить доступ к запрошенному ресурсу.
   * 404 Not Found: Запрошенный ресурс не удалось найти на сервере.
5. Ошибки сервера (5xx):
   * 500 Internal Server Error: Произошла общая ошибка сервера, и сервер не может выполнить запрос.
   * 503 Service Unavailable: Сервер временно не может обработать запрос из-за перегрузки или технического обслуживания.
   * 504 Gateway Timeout: Сервер, действующий в качестве шлюза или прокси-сервера, не получил своевременного ответа от вышестоящего сервера.

**Преимущества REST API:**

1. Масштабируемость: REST API разработаны таким образом, чтобы быть масштабируемыми, позволяя нескольким клиентам взаимодействовать с сервером одновременно. Не зависящий от состояния характер REST облегчает горизонтальное масштабирование, позволяя распределять нагрузку между несколькими серверами.

2. Простота: REST API использует простоту и универсальность HTTP-методов и кодов состояния, что упрощает его понимание и использование. Он соответствует стандартному и предсказуемому шаблону проектирования, который помогает в разработке и интеграции API.

3. Совместимость: REST API совместимы с широким спектром платформ, языков программирования и устройств. До тех пор, пока клиент может выполнять HTTP-запросы и понимать формат представления, он может взаимодействовать с REST API.

4. Кэширование: REST API могут использовать преимущества механизмов кэширования HTTP, позволяя клиентам кэшировать ответы и сокращать количество обратных обращений к серверу, тем самым повышая производительность и сокращая использование полосы пропускания сети.

REST API используются для различных целей при разработке современного программного обеспечения. Некоторые распространенные варианты использования и приложения REST API:

1. Разработка веб-приложений: REST API широко используются для создания веб-приложений, как на стороне клиента, так и на стороне сервера. Они позволяют клиентским приложениям взаимодействовать с ресурсами сервера, извлекать данные, отправлять их и выполнять различные операции.

2. Разработка мобильных приложений: REST API играют жизненно важную роль в разработке мобильных приложений. Мобильные приложения могут использовать REST API для доступа к ресурсам на стороне сервера, извлечения данных, отправки пользовательского ввода и выполнения транзакций. Это обеспечивает плавную интеграцию мобильных приложений с серверными системами.

3. Интеграция систем и сервисов: REST API облегчают интеграцию различных программных систем, позволяя им взаимодействовать и обмениваться данными. API-интерфейсы позволяют приложениям получать доступ к функциональности и сервисам, предоставляемым сторонними системами, такими как платформы социальных сетей, платежные шлюзы, службы электронной почты, службы геолокации и многое другое.

4. Архитектура микросервисов: REST API часто используются в архитектурах микросервисов, где приложение создается как набор небольших независимых сервисов. Каждый микросервис предоставляет REST API, который обеспечивает четко определенный интерфейс для связи и взаимодействия между сервисами.

5. Интернет вещей (IoT): REST API используются в приложениях Интернета вещей для обеспечения связи между устройствами Интернета вещей и другими системами. Устройства могут отправлять данные на сервер или извлекать их с него посредством RESTful-взаимодействий, что позволяет осуществлять удаленное управление, мониторинг и обмен данными.

6. Веб-хуки и уведомления о событиях: REST API можно использовать для настройки веб-хуков или уведомлений о событиях, позволяя системам отправлять обновления в режиме реального времени или запускать действия на основе определенных событий. Это обычно используется в таких приложениях, как чат-боты, инструменты совместной работы в режиме реального времени или службы уведомлений.

7. Системы управления контентом: REST API используются в системах управления контентом (CMS) для предоставления контента и управления ресурсами, такими как статьи, изображения, профили пользователей и многое другое. Они позволяют разработчикам создавать пользовательские интерфейсы front-end или мобильные приложения, взаимодействующие с CMS.

8. Открытые данные и общедоступные API: REST API используются для того, чтобы сделать данные и сервисы доступными для общественности. Организации и государственные органы часто предоставляют общедоступные API-интерфейсы для обмена такими данными, как информация о погоде, транспортные данные, финансовые данные и многое другое, что позволяет разработчикам создавать инновационные приложения на основе этих данных.

Это всего лишь несколько примеров того, как REST API используются в различных доменах. REST API обеспечивают стандартизированный, масштабируемый и гибкий подход к созданию программных систем и обеспечивают бесшовную интеграцию и взаимодействие между приложениями и сервисами.