# Веб-приложение

## Общие сведения

Веб-приложение – это разновидность программного обеспечения, доступ к которому обеспечивается ~~по средствам~~ средствами веб-браузера. Веб-браузер – это программа, которая отправляет запросы, получает и обрабатывает ответы от имени пользователя на его устройстве. Обязательным условием функционирования веб-приложения является наличие стабильного соединения с глобальной сетью Интернет. Одно из главных отличительных особенностей такого вида приложений является кроссплатформенность и независимость от аппаратной конфигурации устройства, у которого есть доступ в сеть. Не мешало бы дать экскурс в «клиент-серверную архитектуру». Или связь…

Процесс разработки веб-приложений весьма разнообразен и охватывает широкую разновидность технологий. Не складно

В этом разделе изложены основные идеи и концепции построения веб-приложения, начиная от архитектуры и краткого описания протокола HTTP, заканчивая описанием реализации обработчиков запросов фреймворка Flask, с помощью которого реализуется веб-приложение для идентификации динамических систем. Точно ли, что обработчик запросов реализует веб-приложение? Поправьте предложение

## Архитектура веб-приложения Следите за красной строкой (отступом). Везде должен быть одного размера

**Архитектура** — это описание системы на самом высоком уровне(???). Глобально любое веб-приложение основано на клиент-серверной архитектуре(вижу ответ на вопрос, который задал в начале страницы. Но вопрос возник там, а ответ здесь.). В такой архитектуре есть сущность(???), которая обслуживает запросы, и есть устройство, которые эти запросы направляет. Сервер – это устройство, на котором установлены программы, которые обслуживают запросы клиентов. Клиент – это программа пользователя, которая удовлетворяет его потребностям. Как уже писал в ТИМСЕ, ссылки на литературу надо вставлять

В основе вышеописанного взаимодействия лежит протокол прикладного уровня модели OSI: протокол HTTP (HyperText Transfer Protocol). HTTP был разработан как протокол обмена данными между веб-сервером и веб-браузером. Который используется для передачи между клиентом и сервером (хранителем) файлов HTML, CSS, JS, изображений, аудио, видео, введённых пользователем данных и прочего. Клиент (веб-браузер) отправляет серверу (веб-серверу) запросы и получает от него ответы. Сервер в рамках взаимодействия всегда «ожидает» инициации от клиента. В одном пункте и архитектура и протокол взаимодействия. Протоколы – это может быть отдельный пункт. Или переделайте заголовок

На этапе инициации клиент устанавливает соединение с сервером с помощью протокола транспортного уровня TCP. Клиент может переиспользовать (как это понимать?) одно и то же соединение для работы с сервером или создавать его каждый раз. Это зависит от задачи, конфигурации сети и конкретных настроек оборудования. После установки соединения клиент посылает HTTP-сообщение с телом и параметрами запроса. Сервер принимает это сообщение и на основании логики работы формирует и отправляет HTTP-сообщение ответа. Таким образом, организовывается сеанс сессии.

След. предложение – нескладно получилось.

Важно упомянуть о необходимости сохранения данных сеанса, например, нужно хранить сведения об авторизации пользователя для того, чтобы пользователь не производил операции по авторизации после отправки каждого запроса. Такой вид информации сохраняется на стороне клиента и сервера и называется Cookie.

## Виды веб-приложений

Все веб-приложения разрабатываются с помощью одних и тех же технологий: HTML, CSS и JavaScript и прочих инструментов. Технологии и инструменты – это совсем не синонимы. Однако есть много вариаций организации работы приложения. Выбор способа зависит от цели приложения и пользовательского опыта, которого мы хотим добиться (добиваться опыта нельзя!). Основных подхода всего два: многостраничные приложения и Single Page Applications (одностраничные приложения), но каждый из них делится на подвиды.

### Многостраничные приложения

Многостраничные приложения состоят в основном из статичных страниц. В таких страницах нет изменяемого контента. Их просто разрабатывать, но если страниц становится много (сотни или тысячи), или данные на странице меняются, то придётся генерировать их в режиме реального времени. Для этого нужно затрачивать программные и аппаратные ресурсы: подключать серверные мощности и писать дополнительный код и на каждый переход от одной страницы к другой нужно генерировать и загружать новую страницу, а это занимает время.

Рассмотрим подвиды многостраничных приложений: приложение с набором готовых страниц (файлы расширения .HTML), и приложение, с множеством страниц, которые генерируются на сервере.

*Совокупность готовых статичных страниц*, которые связаны между собой ссылками друг на друга тоже могут (смотрите за падежами) называться веб-приложением, но точнее их будет назвать сайтом. Как правило, такие сайты содержат в себе данные, которые не меняются или слабо меняются с течением времени: это может быть персональный блог или страница с индивидуальными достижениями. Для организации доступа к таким страницам используются веб-сервера, А выше было написано совсем иное назначение веб-сервера которые по запросу пользователя вернут ему нужный готовый ресурс.

*В случае приложения с динамической генерацией HTML* файлов есть возможность размещать на страницах нестатичные данные, то есть такие данные, которые меняются с течением времени в зависимости от действий пользователя или в зависимости от внешних изменений, например, в приложении может отображаться курс валют. Не пишите длинных предложений. Особенно, когда в отдельных его частях разная смысловая нагрузка В данной работе используется именно такой подход с помощью шаблонизатора Jinja2 – это шаблонный движок на языке Python, который используется для генерации HTML-страниц и других текстовых форматов. Он предоставляет инструменты для управления данными и создания динамических веб-страниц. Jinja2 может быть использован вместе с различными фреймворками веб-приложений, такими как Flask, Django и Pyramid.

### Одностраничные приложения

Одностраничные приложения (SPA) дают возможность разрабатывать клиентские приложения со сложной логикой с помощью JavaScript. В этом подходе отрисовкой содержимого на странице управляет JavaScript. Можно и чуть подробнее. Переходы между экранами будут мгновенными, и пользователь сразу увидит результат своих действий. Однако такой подход создаёт новые проблемы, а именно: большая нагрузка на браузер, вследствие этого падение производительности.

Разработка таких приложений часто сложнее, поэтому в этой работе (в какой работе?. Если это вывод, то лучше писать их (формулировать) где-то в конце раздела – пункты. Используйте единую технологию!) вид веб-приложения – многостраничный, но с размещением на них нестатичных элементов, например, интерактивный график переходного процесса. Этот выбор себя оправдывает, так как само приложение не нагружено большим количеством страниц.

## Архитектура и паттерны проектирования веб-приложений

**Архитектура приложения** — это набор решений о том, как модули приложения будут связаны друг с другом и с внешними компонентами (правильное ли дано определение архитектуры?). **Паттерны проектирования** – шаблонное решение частой архитектурной проблемы. Важно спроектировать веб-приложения согласно определенном паттерну, исходя из архитектурного решения. Рассмотрим паттерн проектирования MVC, который используется в веб-приложении для идентификации динамических систем.

### Паттерн проектирования MVC

MVC (Model-View-Controller) — структура приложения, в которой за данные, их обработку и их вывод отвечают три разных сущности:

* Модель (model) отвечает за данные и их структуру, cодержит в себе схему данных приложения;
* Представление (view) отвечает за отображение данных, показывает данные в понятном для пользователя виде;
* Контроллер (controller) или обработчик – отвечает за обработку данных.

Таким образом разделяется ответственность между разными программными модулями. Важно упомянуть о потоке данных между этими сущностями (см. рис. 1):

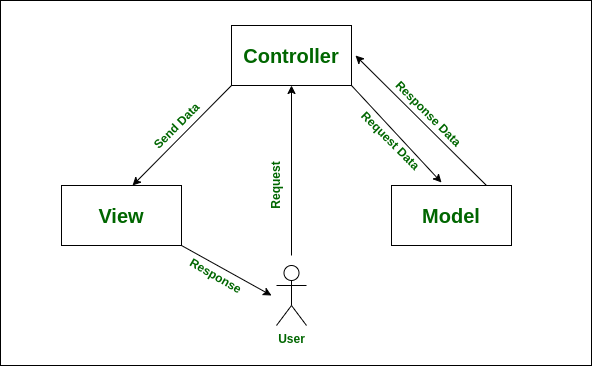
* от пользователя передаются представлению;
* от представления — контроллеру;
* через контроллер обновляется модель;
* модель уведомляет представление о том, что что-то изменилось.

Рисунок 1 – Паттерн проектирования MVC

### Архитектурный стиль RESTful API

Это архитектурный стиль, который используется для создания веб-сервисов. Он опирается на принципы REST (Representational State Transfer) и включает в себя использование HTTP-методов (GET, POST, PUT, DELETE) для управления ресурсами.

*Информация в этом подразделе будет дописываться.*

## Фреймворки для веб-разработки

Фреймворк – набор готовых программных пакетов, программных модулей для облегчения труда программистов и экономии времени разработки программного обеспечения. Фреймворк конкретизирует архитектурное решение и позволяет реализовать выбранный паттерн проектирования. Рассмотрим один из фреймворков.

*Информация в этом подразделе будет дописываться.*

### Flask

Flask – это минималистичный фреймворк для создания веб-приложений на языке Python. Он предоставляет широкий спектр возможностей и инструментов для быстрой и качественной разработки отзывчивых приложений.

Вот некоторые ключевые особенности данного фреймворка:

1. Легкость использования: Flask имеет простую и интуитивно понятную структуру, которая делает его легким в освоении и использовании. Он предоставляет только базовый функционал, позволяя разработчикам гибко выбирать сторонние инструменты и библиотеки для своих проектов.
2. Маршрутизация URL: Flask предлагает простой способ определения URL-маршрутов и связывания их с соответствующими функциями-обработчиками. Это позволяет определить, как приложение должно отвечать на запросы по определенным URL-адресам.
3. Встроенный сервер разработки: Flask поставляется с встроенным сервером разработки, который упрощает запуск и тестирование веб-приложений без необходимости настройки отдельного веб-сервера.
4. Расширяемость: Flask имеет модульную архитектуру, позволяющую разработчикам добавлять функциональные возможности с помощью расширений (extensions). Существуют множество расширений для Flask, которые обеспечивают поддержку баз данных, авторизацию, формы, обработку файлов, асинхронные запросы и многое другое.
5. Шаблонизация: Flask предоставляет инструменты для использования шаблонов, которые упрощают создание динамических веб-страниц. Шаблонизация позволяет разделить логику и представление веб-приложения, что делает код более удобочитаемым и обеспечивает повторное использование кода. Такой движок шаблонизации называется Jinja2.
6. Поддержка HTTP-методов: Flask обрабатывает различные HTTP-методы, такие как GET, POST, PUT, DELETE и другие, что позволяет легко определять обработчики для разных типов запросов.
7. Отладочные возможности: Flask предлагает отладочные инструменты, которые помогают разработчикам в процессе отладки приложений. Это включает в себя режим разработки, отображение отладочной информации при ошибке и возможность интерактивной отладки.
8. Flask является очень популярным фреймворком для разработки веб-приложений на Python благодаря своей простоте и гибкости. Он хорошо подходит для небольших и средних проектов, где требуется быстрый старт и минимальная настройка.

*Информация в этом подразделе будет дописываться*