Веб-разработка с использованием PHP

Примечание: Для удобства, лекция будет разделена на три основных раздела, соответствующих обзору принципов работы веб-приложений, созданию динамических веб-страниц с использованием PHP и HTML, а также работе с сессиями и куками.

Обзор принципов работы веб-приложений

Введение в веб-приложения.

Веб-приложение - это программное обеспечение, которое работает через веб-браузер и взаимодействует с пользователем через интерфейс веб-страницы. В отличие от традиционных приложений, веб-приложения не требуют установки на устройстве пользователя и могут быть доступны из любого места, где есть доступ к Интернету.

Основные концепции веб-приложений:

Клиент-серверная архитектура: Веб-приложения основаны на модели клиент-сервера, где клиент (веб-браузер) отправляет запросы к серверу, который обрабатывает эти запросы и отправляет обратно ответы.

Протокол HTTP: Веб-приложения общаются между собой с помощью протокола HTTP (Hypertext Transfer Protocol), который определяет правила передачи данных между клиентом и сервером.

Статические и динамические веб-страницы: Статические страницы содержат фиксированный контент и не меняются без изменения их кода. Динамические страницы формируются на сервере по запросу и могут изменяться в зависимости от данных или действий пользователя.

Языки разметки и программирования

Для создания веб-приложений используются различные языки, такие как HTML (Hypertext Markup Language) для разметки страниц, CSS (Cascading Style Sheets) для стилизации и оформления, JavaScript для взаимодействия с пользователем на стороне клиента, а также серверные языки программирования, такие как PHP, Python, Ruby и другие, для обработки запросов на стороне сервера и формирования динамического контента.

Сессии и куки: Веб-приложения могут использовать сессии и куки для хранения состояния и данных пользователя между различными запросами.

Различие между статическими и динамическими веб-страницами.

Статические и динамические веб-страницы представляют два основных типа веб-контента, каждый из которых имеет свои особенности и применение.

Статические веб-страницы:

Статическая веб-страница - это веб-страница, содержание которой фиксировано на момент ее создания и не изменяется без вмешательства разработчика. Основные характеристики статических веб-страниц:

Фиксированный контент

Быстрая загрузка

Динамические веб-страницы:

Динамическая веб-страница - это веб-страница, содержание которой формируется на сервере в реальном времени в зависимости от параметров запроса пользователя. Основные характеристики динамических веб-страниц:

Генерация контента на лету

Интерактивность и персонализация:

Обработка данных

Подходят для сложных приложений

В целом, выбор между статическими и динамическими веб-страницами зависит от требований к функциональности и содержанию веб-сайта, а также от того, насколько часто информация должна обновляться.

Роль сервера и клиента в веб-приложениях.

В веб-приложениях роль сервера и клиента разделена по принципу клиент-серверной архитектуры, где каждая сторона выполняет определенные функции и взаимодействует друг с другом для обеспечения работы веб-приложения. Давайте рассмотрим роли сервера и клиента более подробно:

Сервер:

Хранение данных: Сервер обычно служит как центральное хранилище данных, которое содержит информацию, необходимую для работы веб-приложения, такую как HTML-страницы, изображения, видео, базы данных и т. д.

Обработка запросов: Сервер принимает запросы от клиентов и обрабатывает их, генерируя и возвращая соответствующий контент или результаты на основе запроса.

Бизнес-логика: В веб-приложениях сервер также обычно содержит бизнес-логику, которая управляет процессами, такими как аутентификация пользователей, валидация данных, выполнение операций с базой данных и т. д.

Безопасность: Сервер отвечает за обеспечение безопасности данных и контроль доступа к ресурсам веб-приложения.

Клиент:

Отображение контента: Клиент отвечает за отображение содержимого веб-приложения, полученного от сервера. Это включает в себя отображение HTML-страниц, изображений, видео, а также выполнение JavaScript-кода для создания интерактивных элементов и анимаций.

Взаимодействие с пользователем: Клиент обрабатывает действия пользователя, такие как клики мышью, нажатия клавиш и ввод данных в формы, и отправляет соответствующие запросы на сервер для обработки.

Хранение локальных данных: Клиент также может хранить некоторую информацию локально, используя куки, хранилище данных браузера (например, LocalStorage) или другие механизмы, чтобы сохранить настройки пользователя или временные данные.

Общение между сервером и клиентом осуществляется посредством протоколов, таких как HTTP или WebSocket, что позволяет передавать данные и управлять состоянием веб-приложения. Клиент отправляет запросы на сервер, а сервер обрабатывает их и отправляет обратно ответы с необходимым контентом или результатами. Это взаимодействие между клиентом и сервером делает веб-приложения динамичными и интерактивными.

Принципы клиент-серверной архитектуры.

Клиент-серверная архитектура - это распределенная модель взаимодействия компьютерных систем, где функциональность и обязанности разделены между двумя типами узлов: клиентом и сервером. Принципы этой архитектуры определяют основные принципы

организации взаимодействия между клиентом и сервером. Вот основные принципы клиент-серверной архитектуры:

Разделение обязанностей: Основной принцип состоит в разделении функциональности и обязанностей между клиентом и сервером. Клиент отвечает за представление информации пользователю и обработку пользовательских действий, в то время как сервер обеспечивает обработку данных, хранение и управление ресурсами.

Масштабируемость: Клиент-серверная архитектура обеспечивает возможность масштабирования системы путем добавления новых клиентов и серверов. Это позволяет обрабатывать большой объем запросов и увеличивать производительность системы при необходимости.

Независимость компонентов: Клиент и сервер могут быть разработаны и развернуты независимо друг от друга. Это позволяет использовать различные технологии и платформы для клиента и сервера, что обеспечивает гибкость и упрощает обновление и модернизацию системы.

Открытость и расширяемость: Клиенты и серверы могут быть расширены и интегрированы с другими системами с помощью стандартизированных протоколов и интерфейсов. Это обеспечивает открытость системы и позволяет легко взаимодействовать с другими приложениями и сервисами.

Надежность и отказоустойчивость: Клиент-серверная архитектура обеспечивает надежную работу системы путем распределения функциональности между различными узлами. Это позволяет изолировать отказы и обеспечить более высокий уровень доступности и отказоустойчивости.

Безопасность: Клиент и сервер могут взаимодействовать друг с другом через защищенные каналы связи и использовать методы аутентификации и авторизации для обеспечения безопасности передачи данных и защиты от несанкционированного доступа.

Эти принципы обеспечивают основу для разработки и проектирования клиент-серверных систем, обеспечивая их гибкость, масштабируемость, надежность и безопасность. Обзор протокола HTTP и его роли в веб-разработке.

Протокол передачи гипертекста (HTTP) - это протокол прикладного уровня, используемый для передачи и получения данных в сети Интернет. HTTP широко применяется в веб-разработке для обмена информацией между веб-серверами и клиентами. Вот обзор основных характеристик протокола HTTP и его роли в веб-разработке:

Принцип работы:
HTTP основан на клиент-серверной архитектуре, где клиент отправляет HTTP-запросы, а сервер отвечает на них с помощью HTTP-ответов. Запросы и ответы могут содержать различные команды и данные, такие как HTML-страницы, изображения, стили CSS, скрипты JavaScript и т. д.

Базовая структура запроса и ответа: HTTP-запрос и HTTP-ответ состоят из трех основных частей: строки запроса/ответа, заголовков и тела сообщения. Строка запроса содержит метод запроса (например, GET, POST), URL-адрес и версию протокола. Заголовки предоставляют дополнительную информацию о запросе или ответе, такую как тип контента,

длина сообщения и др. Тело сообщения содержит данные запроса или ответа (не всегда присутствует).

Методы запросов: HTTP определяет различные методы запросов, которые указывают серверу, какие операции выполнить. Наиболее часто используемые методы включают GET (получение данных), POST (отправка данных на сервер), PUT (обновление данных на сервере), DELETE (удаление данных на сервере) и другие.

Коды ответов: HTTP-ответы содержат код состояния, который указывает на результат выполнения запроса. Например, код состояния 200 означает успешный запрос, а коды 404 и 500 указывают на ошибки - "страница не найдена" и "внутренняя ошибка сервера" соответственно.

Отсутствие состояния: HTTP является протоколом без состояния, что означает, что каждый запрос-ответ обрабатывается независимо от других запросов-ответов. Это требует использования механизмов, таких как куки и сессии, для поддержания состояния между запросами от одного и того же клиента.

Роль в веб-разработке: HTTP является фундаментальным протоколом в веб-разработке, поскольку обеспечивает основу для взаимодействия между веб-серверами и клиентами. Он позволяет передавать данные и контент веб-приложений, обрабатывать пользовательские запросы и отправлять результаты обратно клиентам. Также HTTP обеспечивает безопасную передачу данных с помощью HTTPS, использующего шифрование для защиты конфиденциальности данных.

HTTP играет ключевую роль в обмене информацией и взаимодействии между клиентами и серверами в веб-разработке, обеспечивая быструю, надежную и безопасную доставку контента через Интернет.

Обзор различных языков программирования для веб-разработки (HTML, CSS, JavaScript, PHP и т. д.).

Этот раздел мы подробнее разберём в соответствующей лекции.

Концепции создания динамических веб-страниц с использованием PHP и HTML

Передача данных между HTML-формами и PHP-скриптами (методы GET и POST).

примеры из нашей программы

Обработка данных формы на сервере с использованием PHP.

примеры из нашей программы

Динамическое формирование контента на основе данных, полученных с сервера.

примеры из нашей программы

Работа с сессиями и куками

Введение в сессии и куки: определение и различие между ними.

Сессии:

Определение: Сессия представляет собой временную сущность между клиентом и сервером, которая позволяет сохранять информацию о пользователе в течение одного или нескольких запросов.

Работа с сессиями: Когда пользователь впервые заходит на сайт, сервер создает уникальную сессию для этого пользователя. Идентификатор сессии обычно хранится в виде

куки на стороне клиента или передается в URL-адресе. Вся информация о пользователе (например, имя пользователя, предпочтения, состояние авторизации) сохраняется на сервере в рамках этой сессии и доступна на протяжении всего сеанса.

Безопасность: Сессии обычно более безопасны, чем куки, поскольку данные хранятся на сервере, и только идентификатор сессии передается между клиентом и сервером.

Куки:

Определение: Куки (или cookies) - это небольшие фрагменты данных, хранящиеся на стороне клиента (обычно в браузере) и предназначенные для хранения информации о предпочтениях пользователя, а также для отслеживания его активности на веб-сайтах.

Работа с куками: Куки создаются сервером и отправляются в браузер пользователя вместе с HTTP-ответом. Браузер сохраняет куки и автоматически отправляет их вместе с каждым запросом на сервер, когда пользователь посещает веб-сайт. Куки могут быть постоянными (срок хранения установлен на определенный срок) или временными (удаляются после закрытия браузера).

Безопасность: Куки могут быть менее безопасны, чем сессии, особенно если в них хранится конфиденциальная информация, так как данные куки доступны для просмотра и изменения клиентом.

Работа с сессиями в PHP: создание, использование и уничтожение сессий.

Сессия в PHP создается с помощью функции session_start(). Обычно это делается в начале каждого скрипта, который требует использования сессии. При этом PHP создает уникальный идентификатор сессии для текущего пользователя и ассоциирует его с данными сеанса на сервере.

В нашем случае, так как у нас одна точка входа, мы можем стартовать сессию в ней. Веб-сервер читает стандартную куку PID (которая также передаётся в запросе) и восстанавливает сессию либо создаёт новую если таковой нету и передаёт новую куку PID. это делается без нашего участия. Обычно PHP сам понимает, человек вернулся или пришёл заново.

Теперь мы можем манипулировать переменной $_SESSION. В рамках одной сессии, данные в этой переменной будут храниться постоянно.

Сохранение и чтение данных сессии.

По-умолчанию, в PHP уже настроена и готова к использованию сессия, основанная на файлах. но если хочется можно переделать и сохранять сессию в своей базе данных или где-то ещё. Для этого надо реализовать свои функции создания, закрытия, чтения, записи, и т.д. и передать в функцию session_set_save_handler до того как была запущена сессия.

Использование куков для хранения информации на стороне клиента.

Cookie - фрагменты данных, которые веб-сервер отправляет браузеру, а затем браузер сохраняет на компьютере пользователя. Куки используются для хранения информации на стороне клиента.

Создание, чтение и удаление куков с помощью PHP.

мы можем установить куки функцией
setcookie($name, $value, time() + (86400 * 30), "/")

для чтения мы используем массив $_COOKIE, это те куки, которые отправил браузер клиента в запросе.

для удаления обычно устанавливают данную куку со временем в прошлом. она удалится браузером автоматически

Практика:

Задача:

Мы хотим запоминать ip адрес клиента когда он создаёт ссылку чтобы потом показать с помощью этих данных, какие ссылки он создавал ранее.

1. добавить новые поля ip_address varchar(16) и SID varchar(64) в таблицу short_url
2. в index.php добавляем session_start() и узнаём SID клиента с помощью session_id()
3. при формировании ссылки узнаём IP клиента с помощью $_SERVER
4. сохраняем в таблицу
5. в Url.php добавляем функции find_short_links_by_ip($ip):array и find_short_links_by_sid($sid):array.
6. в default.php и в create.php в конец отдельным блоком добавляем список коротких ссылок с этого IP и ещё одним блоком список с этого SID, используя функции выше