Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
Уфимский государственный авиационный технический университет
Факультет информатики и робототехники
Кафедра вычислительной математики и кибернетики

Отчет к лабораторной работе №2
по дисциплине «Архитектура вычислительных систем и компьютерных сетей»
«Разработка приложений для Web-сервера (без работы с БД)»

Выполнил
студент группы ПРО-306
Васильев Д.А.

Проверил
Верхотуров М.А.

Уфа, 2015 г.

Содержание

1 Цель работы и постановка задачи

Цель работы: получение навыков разработки динамических HTML-страниц.

Постановка задачи: разработать веб-приложение - микроблог, позволяющий писать, редактировать и удалять сообщения. Реализация аутентификации и различных уровней доступа. Реализация хранения информации о пользователях и информации о сообщениях.

Формальная постановка задачи

2 Структура решения задачи.

1. Ввод и передача данных.
2. Аутентификация пользователей.
3. Обработка данных.
   - Регистрация нового пользователя в приложении.
   - Добавление нового сообщения.
   - Редактирование сообщений.
   - Редактирование пользовательских данных.
4. Организация хранения информации.
5. Генерация страниц веб-приложения.
   - Генерация главной страницы.
   - Генерация страницы профиля пользователя.

3 Теоретическая часть

3.1 Протоколы и языки для создания веб-приложений

3.1.1 HTTP

HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате HTML, в настоящий момент используется для передачи произвольных данных). Основой HTTP является технология «клиент-сервер», то есть предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

HTTP — протокол прикладного уровня, аналогичными ему являются FTP и SMTP. Обмен сообщениями идёт по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов HTTP использует глобальные URI. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами (например, «куки» на стороне клиента, «сессии» на стороне сервера). Браузер, посылающий запросы, может отслеживать задержки ответов. Сервер может хранить IP-адреса и заголовки запросов последних клиентов. Однако сам протокол не осведомлён о предыдущих запросах и ответах, в нём не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

Метод HTTP (англ. HTTP Method) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Обратите внимание, что название метода чувствительно к регистру.

Метод GET используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.

Метод POST применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы на сервер. В отличие от метода GET, метод POST не считается идемпотентным, то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться очередная копия этого комментария).

Каждый сервер обязан поддерживать как минимум методы GET и HEAD. Часто применяется метод POST.

3.1.2 XML

XML (англ. eXtensible Markup Language — расширяемый язык разметки). Рекомендован Консорциумом Всемирной паутины (W3C). Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программам и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Сочетание простого формального синтаксиса, удобства для человека, расширяемости, а также базирование на кодировках Юникод для представления содержания документов привело к широкому использованию как собственно XML, так и множества производных специализированных языков на базе XML в самых разнообразных программных средствах.

3.1.3 HTML

HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML). Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

Язык XHTML является более строгим вариантом HTML, он следует всем ограничениям XML и, фактически, XHTML можно воспринимать как приложение языка XML к области разметки гипертекста.

Во всемирной паутине HTML-страницы, как правило, передаются браузерам от сервера по протоколам HTTP или HTTPS, в виде простого текста или с использованием шифрования.

Текстовые документы, содержащие разметку на языке HTML (такие документы традиционно имеют расширение .html или .htm), обрабатываются браузерами или «интернет-обозревателями». Эти приложения обычно предоставляют пользователю удобный интерфейс для запроса веб-страниц, их просмотра (и вывода на иные внешние устройства) и, при необходимости, отправки введённых пользователем данных на сервер. Наиболее популярными на сегодняшний день браузерами являются *Google Chrome, Mozilla Firefox, Opera, Internet Explorer и Safari.*

HTML — теговый язык разметки документов. Любой документ на языке HTML представляет собой набор элементов, причём начало и конец каждого элемента обозначается специальными пометками — тегами. Элементы могут быть пустыми, то есть не содержащими никакого текста и других данных (например, тег перевода строки `<br>`). В этом случае обычно не указывается закрывающий тег. Кроме того, элементы могут иметь атрибуты, определяющие какие-либо их свойства (например, размер шрифта для элемента `font`). Атрибуты указываются в открывающем теге.

HTML5 - это пятая версия HTML. Хотя стандарт был завершён (рекомендованная версия к использованию) только в 2014 году (предыдущая, четвёртая, версия опубликована в 1999 году), ещё с 2013 года браузерами оперативно осуществлялась поддержка, а разработчиками — использование рабочего стандарта. Цель разработки HTML5 — улучшение уровня поддержки мультимедиа-технологий с одновременным сохранением обратной совместимости, удобочитаемости кода для человека и простоты анализа для парсеров.

В HTML5 реализовано множество новых синтаксических особенностей. Например, элементы `<video>`, `<audio>` и `<canvas>`, а также возможность использования SVG и математических формул. Эти новшества разработаны для упрощения создания и управления графическими и мультимедийными объектами в сети без необходимости использования сторонних API и плагинов. Другие новые элементы, такие как `<section>`,

<article>, <header> и <nav>, разработаны для того, чтобы обогащать семантическое содержимое документа (страницы). Новые атрибуты были введены с той же целью, хотя ряд элементов и атрибутов был удалён. Некоторые элементы, например <a>, <menu> и <cite>, были изменены, переопределены или стандартизированы. API и DOM стали основными частями спецификации HTML5. HTML5 также определяет некоторые особенности обработки ошибок вёрстки, поэтому синтаксические ошибки должны рассматриваться одинаково всеми совместимыми браузерами.

3.1.4 CSS

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

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

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

До появления CSS оформление веб-страниц осуществлялось исключительно средствами HTML, непосредственно внутри содержимого документа. Однако с появлением CSS стало возможным принципиальное разделение содержания и представления документа. За счёт этого нововведения стало возможным лёгкое применение единого стиля оформления для массы схожих документов, а также быстрое изменение этого оформления.

Преимущества:

- Несколько дизайнов страницы для разных устройств просмотра. Например, на экране дизайн будет рассчитан на большую ширину, во время печати меню не будет выводиться, а на КПК и сотовом телефоне меню будет следовать за содержимым.
- Уменьшение времени загрузки страниц сайта за счет переноса правил представления данных в отдельный CSS-файл. В этом случае браузер загружает только структуру документа и данные, хранимые на странице, а представление этих данных загружается браузером только один раз и может быть закэшировано.
- Простота последующего изменения дизайна. Не нужно править каждую страницу, а лишь изменить CSS-файл.
- Дополнительные возможности оформления. Например, с помощью CSS-вёрстки можно сделать блок текста, который остальной текст будет обтекать (например для меню) или сделать так, чтобы меню было всегда видно при прокрутке страницы.

Недостатки:

- Различное отображение вёрстки в различных браузерах (особенно устаревших), которые по-разному интерпретируют одни и те же данные CSS.
- Часто встречающаяся необходимость на практике исправлять не только один CSS-файл, но и теги HTML, которые сложным и ненаглядным способом связаны с селекторами CSS, что иногда сводит на нет простоту применения единых файлов стилей и значительно удлиняет время редактирования и тестирования.

3.2 Динамические HTML-страницы

Статическая веб-страница выглядит всегда одинаково, независимо от действий пользователя. Например, меню организованное ссылками на отдельные страницы, а не выпадающим списком. В отличие от статических

динамические страницы уже могут реагировать на действия пользователя и изменяться. Например, при щелчке по тексту может показываться всплывающий блок текста с переводом слова.

Динамика на веб страницах реализована при помощи скриптов, которые выполняются браузером. Многие элементы языка HTML поддерживают определение обработчиков событий. Например, можно задать обработку события "нажатия кнопки мыши"на картинке. Тогда если пользователь кликнет на эту картинку, вызовется определенный для этого обработчик.

В отличие от статичной страницы, которая является просто файлом, лежащим на сервере, динамическая страница генерируется сервером. Они обычно обрабатывают и выводят информацию из базы данных.

Для генерации *скриптов*, создающих веб-страницы можно использовать большое количество языков программирования: *Perl, PHP, Python, Ruby, Go* и многие другие.

3.2.1 JavaScript

JavaScript (аббр. JS) — прототипно-ориентированный сценарный язык программирования. Является реализацией языка ECMAScript.

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

JavaScript используется в клиентской части веб-приложений: клиент-серверных программ, в котором клиентом является браузер, а сервером — веб-сервер, имеющих распределённую между сервером и клиентом логику. Обмен информацией в веб-приложениях происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются кроссплатформенными сервисами.

Этот язык также используется в AJAX, популярном подходе к построению интерактивных пользовательских интерфейсов веб-приложений, заключающемся в «фоновом» асинхронном обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью и интерфейс веб-приложения становится быстрее, чем это происходит при традиционном подходе (без применения AJAX).

Пользовательские скрипты, написанные на JavaScript, позволяют автоматически заполнять формы, переформатировать страницы, скрывать нежелательное содержимое и встраивать желательное для отображения содержимое, изменять поведение клиентской части веб-приложений, добавлять элементы управления на страницу и т.д.

3.2.2 AJAX

AJAX (англ. Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.

При использовании AJAX:

- Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент.
- Скрипт (на языке JavaScript) определяет, какая информация необходима для обновления страницы.
- Браузер отправляет соответствующий запрос на сервер.
- Сервер возвращает только ту часть документа, на которую пришёл запрос.
- Скрипт вносит изменения с учётом полученной информации (без полной перезагрузки страницы).

AJAX — не самостоятельная технология, а концепция использования нескольких смежных технологий. AJAX базируется на двух основных принципах:

- использование технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью, например с использованием XMLHttpRequest (основной объект);.

- использование DHTML для динамического изменения содержания страницы; Действия с интерфейсом преобразуются в операции с элементами DOM (англ. Document Object Model), с помощью которых обрабатываются данные, доступные пользователю, в результате чего представление их изменяется. Здесь же производится обработка перемещений и щелчков мышью, а также нажатий клавиш. Каскадные таблицы стилей, или CSS (англ. Cascading Style Sheets), обеспечивают согласованный внешний вид элементов приложения и упрощают обращение к DOM-объектам. Объект XMLHttpRequest (или подобные механизмы) используется для асинхронного взаимодействия с сервером, обработки запросов пользователя и загрузки в процессе работы необходимых данных.

Использование AJAX экономит траффик при работе с веб-приложениями, уменьшает нагрузку на сервер, ускоряет реакцию интерфейса и предоставляет почти безграничные возможности для интерактивной обработки. Однако, у этой технологии есть ряд недостатков. Например отсутствие интеграции со стандартными инструментами браузера, недоступность динамически сгенерированных страниц для поисковых систем, требует включенного *JavaScript* в браузере и усложняет проект.

Три из этих четырёх технологий — CSS, DOM и JavaScript — составляют DHTML (англ. Dynamic HTML).

4 Обзор и анализ методов решения

4.1 Ввод и передача данных

При рассмотрении ввода данных нельзя не упомянуть о том, что данный этап неразрывно связан с этапом вывода. Для того, чтобы пользователь мог ввести свою информацию, предварительно должны быть сгенерирована страница с соответствующим содержимым.

4.1.1 Ввод данных с использованием форм

Для обмена данными между пользователем и сервером предназначен HTML-тег `<form>`.
В качестве основных подзадач на данном этапе можно выделить:

- Ввод данных для отправки нового сообщения.
- Ввод данных для регистрации нового пользователя в приложении.
- Ввод данных для аутентификации пользователя.

Для ввода данных в каждой из подзадач необходимо предоставить пользователю *форму*, на которой будут соответствующие поля ввода или выбора данных. В частности, для отправки текстового сообщения достаточно два поля ввода: поле для заголовка и поле для самого сообщения. Форма для аутентификаии и регистрации достаточно похожа: каждая из них содержит поле для ввода имени и пароля. Для формы регистрации логично предусмотреть дополнительное поле "повтора пароля".

Обобщенный алгоритм ввода данных для отправки нового сообщения можно представить в следующем виде:

1. Сгенерировать страницу с формой для ввода заголовка сообщения и тела сообщения.
2. Показать форму пользователю.

Обобщенный алгоритм ввода данных для регистрации нового пользователя в приложении.

1. Сгенерировать страницу с формой для ввода имени пользователя и пароля.
2. Показать форму пользователю.

Обобщенный алгоритм ввода данных для аутентификации пользователя.

1. Сгенерировать страницу с формой для ввода имени пользователя и пароля.
2. Показать форму пользователю.

Как хорошо видно, последние три алгоритма можно выделить в отдельный класс ползадач для ввода данных, так как в каждом из них используется идентичный набор компонент ввода. Единственное отличие этих форм - разный анализ данных на этапе обработки данных.

Таким образом, можно выделить **обобщенный алгоритм ввода данных**.

1. Сгенерировать страницу с формой для ввода пользовательских данных с заданным набором компонент ввода.
2. Показать форму пользователю.

4.1.2 Передача данных

Для запроса, при котором веб-сервер принимает данные, заключенные в тело сообщения, в протоколе HTTP используется метод POST. Процедура передачи данных заключается в формировании POST запроса и добавлением в него данных из полей для ввода информации.

В дальнейшем функции, реализующие логику работы веб-приложения, получают из данного запроса значения параметров и обрабатывают эти значения.

Отправка POST запроса потребуется при решении всех основных подзадач ввода данных. Передача данных может осуществляться асинхронно, не требуя перезагрузки при получении данных от сервера, и синхронно. Для нашей задачи не требуется использования асинхронных подходов (например используя технологию AJAX), достаточно синхронной передачи данных.

Обобщенный алгоритм передачи данных.

1. Сформировать запрос, передав в него значения параметров и необходимую информацию.
2. Отправить запрос используя протокол передачи.

4.2 Аутентификация и авторизация пользователей

4.2.1 Аутентификация

Для аутентификации пользователя обычно достаточно двух полей: имени пользователя (логина) и поля для ввода пароля.

При аутентификации возможны три случая:

- **Неверное имя пользователя.** Пользователя с данным именем не существует в базе.
- **Неверный пароль.** Пользователь с данным именем существует в базе, но пароль введен неверный.
- **Верное имя пользователя и верный пароль.** Соответствующая запись в базе найдена.

При вводе пароля необходимо продумать еще несколько моментов. Так например, необходимо, чтобы все введенные символы пароля были "скрыты"в виде какого-либо заполнителя. Это не позволит случайно увидеть пароль другим людям.

Кроме того, необходимо предусмотреть обязательное заполнение поля пароля. Если пароль не введен, то нужно проинформировать об этом пользователя и показать форму ввода данных заново. Проверку ввода можно организовать простой проверкой данных на пустоту, что потребует отдельной проверки в обработке или же воспользоваться готовыми решениями в виде оберток над стандартными полями ввода, доступными в различных фреймворках, использующих "валидаторы"(например валидатор "обязательный "Required").

Для решения задачи было решено использовать поля, предоставляемые фреймворком и валидаторы.

После ввода данных необходимо произвести проверку введенных данных, а именно найти соответствующую запись в базе пользователей.

При успешном прохождении проверки корректности данных на этапе ввода, посылается POST запрос. В этом запросе хранятся необходимые данные: имя пользователя и его пароль.

Таким образом, при успешной аутентификации необходимо присвоить глобальной переменной "текущий пользователь"значение имени пользователя.

Обобщенный алгоритм аутентификации можно представить следующим образом:

1. Найти пользователя в базе пользователей.
2. Произвести проверку правильности введенного пароля.
3. При успешном прохождении проверки записать информацию о текущем пользователе в глобальную переменную.

4.2.2 Авторизация

Авторизация пользователей необходима, в первую очередь, для ограничения доступа к различным возможностям веб-приложения. Возможности каждой группы ограничены. Обычно в приложениях подобного типа есть три категории пользователей: *администратор* имеет самые полные возможности. Он может удалять и редактировать все сообщения, редактировать профиль пользователя и изменять его уровень доступа. *Пользователь* - группа для большинства пользователей сервиса. Он может редактировать и удалять только свои сообщения и редактировать только свои данные. Также он может и добавлять новые сообщения. В отличие от него, участники группы *только чтение* не могут писать новые сообщения, но имеют все остальные возможности членов группы *пользователь*.

Для организации авторизации удобно хранить информацию о текущем пользователе и его уровне доступа в некой глобальной переменной, доступной из любого места веб-приложения. Различные фреймворки по разному предоставляют данную функциональность, но идея у них одинакова.

Практически все действия требуют, чтобы пользователь был авторизован для данного вида деятельности. Например, при переадресации на профиль пользователя необходимо произвести проверку, что текущий пользователь имеет права редактирования. Кроме того, нужно проверить уровень доступа при непосредственном переходе по ссылке. При генерации главной страницы также необходимо проверить может ли текущий пользователь писать сообщения, удалять чужие сообщений и т.д. Аналогично предыдущей ситуации, здесь проводится проверка уровня доступа.

Обобщив всё вышесказанное, можно выделить **алгоритм решения общей подзадачи проверки уровня доступа.**

1. Получить из глобальной переменной пользователя значение переменной уровня доступа.
2. В зависимости от значения данной переменной отобразить необходимые компоненты, запретить действие.

4.3 Обработка данных

4.3.1 Регистрация нового пользователя в приложении

Регистрация пользователя обычно подразумевает добавление нового пользователя в базу данных. Перед самой процедурой добавления и обработки введенных данных, пользователь обычно заполняет *форму* на этапе *ввода данных*.

При регистрации возможен случай, когда введенное имя пользователя уже зарегистрировано в базе, пользователь с таким именем уже существует. Для предотвращения этого необходимо после подтверждения регистрации произвести проверку по базе пользователей и в случае совпадения проинформировать его.

Кроме проверки на наличие ввода пароля, аналогично с аутентификацией, необходимо предусмотреть правила заполнения поля пароля: ограничение на минимальное или максимальное количество символов пароля, наличие цифр, букв и специальных символов, наличие символов разного регистра. Если пароль не введен или не соответствует требованиям, то нужно проинформировать об этом пользователя и показать форму ввода данных заново. Для этого можно использовать регулярные выражения, установить соответствующие валидаторы для полей или же вынести проверку в отдельную функцию.

В нашем случае небольшого сервиса микроблогов не требуется столь строгих правил задания пароля, поэтому достаточно проверки на наличие ввода пароля.

Зачастую пользователи хотят дополнить информацию о себе. Например, местоположение, реальное имя или контактные данные. Эту информацию нужно хранить в той же базе пользователей. При регистрации эти поля можно оставлять пустыми и не требовать их заполнения.

Обобщенный алгоритм регистрации можно представить в виде:

1. Проверка доступности имени пользователя.
2. Проверка требований к паролю (заполненное поле, ограничение на длину пароля).

3. Создание записи и добавление пользователя в базу.

4.3.2 Добавление нового сообщения

Структуру данных "сообщение"можно представить различными способами. Обычно, наиболее целесообразно хранить следующие данные: заговок сообщения, само сообщение, автора и дату написания. Сообщения могут хранится в различном виде: в файле, в виде базы данных, в виде сложной структуры данных в самом приложении. При этом, независимо от способа хранения, необходимо обеспечить каждую запись "уникальным идентификатором". При использовании баз данных это можно сделать с помощью соответствующего поля, при хранении в файле или в сложной структуре нужно генерировать уникальный идентификатор и присваивать его соответствующим записям.

На заговок и тело сообщения не накладывается никаких строгих ограничений кроме количества символов.

При этом необходимо произвести проверку на возможность добавления сообщений текущим пользователем. Как уже было сказано выше, пользователи с правами доступа "только чтение"не могут добавлять новые записи.

Процедура добавления нового сообщения в базу достаточно тривиальна.

1. Проверка на возможность добавления новых записей.
2. Создание записи, включающей заголовок сообщения, тело сообщения и автора. Данные получаются из POST запроса.
3. Добавление записи в базу.

4.3.3 Редактирование сообщений

Редактирование сообщений заключается в простом редактировании записей в базе. Для нахождения конкретной записи можно использовать уникальные идентфикаторы, создание которых описано в предыдущем пункте. Перед редактированием и вводом данных пользователь видит сгенерированную страницу с формой для редактирования. Кроме того из этой формы получаем идентификатор записи, по которой можем однозначно определелить нужное сообщение.

Аналогично с добавлением записей, для редактирования необходимо иметь соответствующие права. Администратор может редактировать все сообщения, в то время как простой пользователь и пользователь с правами "только чтение"только сообщения за своим авторством.

Таким образом, **обобщенный алгоритм** можно представить следующим образом

1. Проверка на возможность редактирования сообщений.
2. Изменение соответствующей записи с использованием данных из POST запроса.

4.3.4 Редактирование пользовательских данных

Редактирование пользовательских данных по последовательности действий очень похоже на редактирование сообщений. Изменять профиль могут либо сами пользователи, либо администратор. Перед редактированием и вводом данных пользователь также видит форму для редактирования. Единственное отличие состоит в том, что здесь присутствуют необязательные для заполнения поля личных данных, которые можно оставить незаполненными.

Обобщенный алгоритм

1. Проверка на возможность редактирования пользовательских данных.
2. Изменение соответствующей записи с использованием данных из POST запроса.

4.4 Хранение данных

Для работы многих веб-приложений требуется хранить различные данные: логины и пароли пользователей, группы пользователей, сообщения и комментарии. Простые приложения могут работать и без сохранения информации, но для реализации более сложной логики работы без этого не обойтись. Механизмы аутентификации, механизмы публикации сообщений и механизмы авторизации - все они оперируют со специфическими данными.

Хранение данных можно обеспечить двумя очевидными способами: в файле и в базе данных. У каждого из этих подходов есть свои преимущества и недостатки.

Хранение данных в файле - это один из самых простых способов. При данном подходе на веб-сервере хранится некий файл, который, при необходимости, открывается, считывается и редактируется. Для этого способа не требуется использование сторонних библиотек - достаточно стандартных средств языка для работы с файлами. Однако, существует существенный недостаток - при большом количестве данных доступ к ним будет довольно затратным и медленным, возможно потребуется часто открывать, записывать и закрывать файлы на диске. Кроме того, для поиска информации, например записи о соответствии логина и пароля, потребуется просмотр всего файла.

Использование базы данных - это более гибкий подход. В базе данных можно организовать необходимые таблицы, заполнять их и быстро получать из них необходимую информацию. Этот способ организации данных требует использования сторонних библиотек для работы с БД, возможно потребует установки дополнительного ПО.

Обобщенный алгоритм доступа к данным.

1. Открыть файл для чтения.
2. Считать данные.
3. Обработать данные из файла.
4. Закрыть файл.

Обобщенный алгоритм добавления данных.

1. Открыть файл.
2. Преобразовать данные в необходимый вид.
3. Записать новые данные.
4. Закрыть файл.

4.5 Генерация страниц веб-приложения

Генерация страниц веб-приложения в основном осуществляется *скриптами*, написанными на различных языках. Кроме того, широко используются *шаблонизаторы*. Шаблонизатор (в web) — это программное обеспечение, позволяющее использовать HTML-шаблоны для генерации конечных HTML-страниц. Основная цель использования шаблонизаторов — это отделение представления данных от исполняемого кода. Часто это необходимо для обеспечения возможности параллельной работы программиста и дизайнера-верстальщика. Использование шаблонизаторов часто улучшает читаемость кода и внесение изменений во внешний вид, когда проект целиком выполняет один человек.

Наиболее распространенными шаблонизаторами на данный момент являются:

- **Java**: Apache Velocity, FreeMarker, Histone.
- **PHP**: BH, Smarty, Twig, Separate.
- **Python**: Genshi, Kid, Jinja2.
- **Perl**: Template ToolKit, HTML::Template.
- **Ruby/Rails**: Erubis, Haml, Slim.
- **JavaScript**: BH, Handlebars, Underscore, Jade.

4.5.1 Генерация главной страницы

Главную страницу веб-приложения можно условно разделить на три части:

- Навигационная панель.
- Форма добавления сообщения.
- Список сообщений.

Состав навигационной панели обычно включает в себя следующие компоненты: ссылку на главную страницу, ссылки на форму регистрации и входа для неавторизованного пользователя, ссылки для просмотра профиля и выхода из приложения для авторизованного. Важно отметить, что неавторизованный пользователь на видит форму для добавления сообщения. Также не видит её и пользователь с правами "только чтение".

Список сообщений генерируется исходя из записей в базе сообщений. С помощью шаблонизаторов можно организовать циклическую генерацию форм сообщений, что позволит без труда вывести все сообщения в виде списка.

Обобщенный алгоритм можно представить следующим образом

1. Генерация навигационной панели.
2. При наличии прав на добавление записей, генерация формы добавления.
3. Генерация списка сообщений.

4.5.2 Генерация страницы профиля пользователя

Страница профиля пользователя может содержать текстовую, графическую, аудио- и видео- информацию. Кроме того, на этой странице также присутствует навигационная панель. Данные для генерации также хранятся в базе.

Если пользователь может редактировать данный профиль, то необходимо сделать ссылку на страницу редактирования.

Обобщенный алгоритм

1. Генерация навигационной панели.
2. Генерация полей, заполненных данными о пользователе.
3. При наличии прав редактирования профиля, генерация ссылки на страницу изменения профиля.

5 Описание и реализация применяемых методов

Для создания веб-приложения используется микрофреймворк *Flask*, предлагающий несложные и эффективные проектные решения.

5.1 Ввод и передача данных.

5.1.1 Ввод данных с использованием форм.

6 Заключение