Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 3

По дисциплине Web-программирование

Тема работы Gulp, PHP, WordPress

Обучающийся Бабаев Руслан Сагибович

Факультет Факультет инфокоммуникационных технологий

Группа К3321

Направление подготовки 11.03.02 Инфокоммуникационные технологии и системы связи

Образовательная программа Программирование в инфокоммуникационных системах

Обучающийся	09.12.2024 (дата)	_____ (подпись)	Бабаев Р.С. (Ф.И.О.)
Руководитель	_____ (дата)	_____ (подпись)	Марченко Е.В. (Ф.И.О.)

Санкт-Петербург
2024 г.

Цель

Целью лабораторной работы является освоение базовых навыков фронтенд- и бэкенд-разработки, включая автоматизацию задач с помощью Gulp, создание интерактивной веб-формы для сбора обратной связи с последующей обработкой данных на сервере, а также установка и настройка локального веб-сервера для работы с CMS WordPress, что позволит получить практический опыт в создании и администрировании веб-проектов.

Задачи

1. **Настройка Gulp:**

- Создать два таска:
    - **Последовательное выполнение:** один таск выполняется после завершения другого.
    - **Параллельное выполнение:** оба таска запускаются одновременно.
- Настроить **browser-sync** для автоматического обновления браузера при изменении файлов проекта.

2. **Разработка формы обратной связи:**

- Реализовать HTML-форму, содержащую:
    - Поля для имени, фамилии, электронной почты и текста сообщения.
    - Радиокнопки (минимум 2).
    - Чекбоксы (минимум 3).
- Написать PHP-скрипт для обработки данных, переданных с формы.
- Объяснить разницу между использованием методов GET и POST, а также привести примеры их применения.

3. **Развертывание WordPress:**

- Установить локальный сервер (LAMP, Денвер или аналогичный).
- Скачать и установить движок WordPress.

- Настроить сайт, чтобы он открывался по адресу http://test.site.
- По желанию, выбрать и установить понравившуюся тему для сайта.

Ход работы

Часть 1

Для выполнения задачи по настройке Gulp была создана конфигурация, включающая два таска для последовательного и параллельного выполнения, а также функция для запуска локального PHP-сервера с автоматической перезагрузкой браузера.

В первую очередь был установлен и настроен Gulp. Основные зависимости, такие как gulp и browser-sync, были добавлены в проект с использованием npm install. Для работы с PHP был использован встроенный сервер PHP, который запускается через команду php -S.

В коде представлено три основные функции:

1. **hello** и **goodbye** (см. рисунок 1): Эти функции демонстрируют выполнение простых задач в Gulp. Они выводят в консоль сообщения "Hello, from Gulp!" и "Goodbye, from Gulp!". Эти таски были объединены в два варианта:

    - **Последовательное выполнение:** Функция gulp.series() обеспечивает выполнение задач строго одна за другой. Это полезно в случаях, когда одна задача зависит от завершения другой.
    - **Параллельное выполнение:** Функция gulp.parallel() запускает задачи одновременно, что ускоряет выполнение несвязанных между собой задач.

```
// Greeting function
const hello = (cb) => {
    console.log('Hello, from Gulp!');
    cb();
};


// Goodbye function
const goodbye = (cb) => {
    console.log('Goodbye, from Gulp!');
    cb();
};

exports.series = gulp.series(hello, goodbye);
exports.parallel = gulp.parallel(hello, goodbye);
```

Рисунок 1 – функции hello и goodbye

2. **phpServer** (см. рисунок 2)**:** Эта функция запускает встроенный PHP-сервер на localhost:3000, используя команду php -S localhost:3000 -t ./app. Также она настраивает browser-sync, который автоматически отслеживает изменения в файлах HTML и PHP в указанной директории. При обнаружении изменений browser-sync обновляет браузер, обеспечивая удобный процесс разработки.

```
// Function to start php server and watch for files (with auto-reload)
const phpServer = (done) => {
    const phpServer = exec('php -S localhost:3000 -t ./app');

    phpServer.stdout.on('data', (data) => console.log(data));
    phpServer.stderr.on('data', (data) => console.error(data));

    browserSync.init({
        proxy: 'localhost:3000',
        notify: false,
        startPath: 'feedback.html',
    });

    gulp.watch('./app/*.html').on('change', browserSync.reload);
    gulp.watch('./app/*.php').on('change', browserSync.reload);

    done();
};
```

Рисунок 2 – Функция phpServer()

Логи подтверждают успешное выполнение задач. Команды gulp series и gulp parallel отработали корректно, выводя ожидаемые сообщения в консоль (см. рисунок 3).

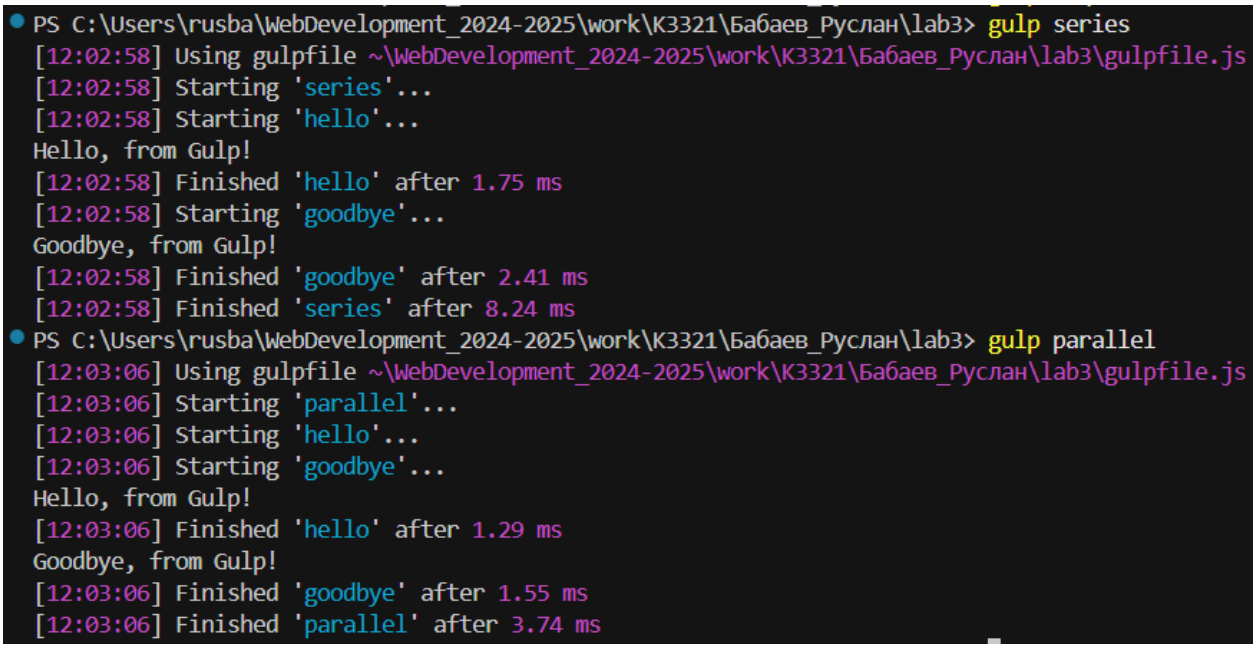


Рисунок 3 – Результаты выполнения функций последовательно и параллельно

Запуск команды gulp serve стартует PHP-сервер, проксирует его через browser-sync, что позволяет просматривать сайт по адресу http://localhost:3001/feedback.html. Изменения в файлах автоматически обновляют содержимое в браузере, исключая необходимость ручной перезагрузки (см. рисунок 4).

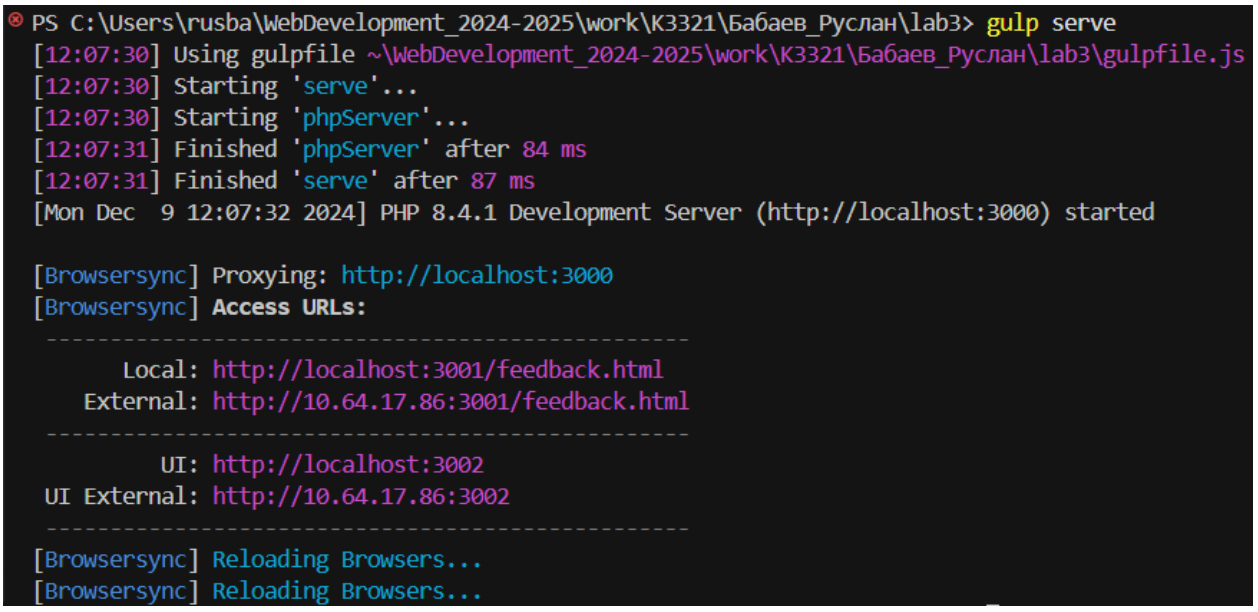


Рисунок 4 – Результаты выполнения функции phpServer и обновления файлов

Эта настройка Gulp оптимизирует процесс разработки, обеспечивая автоматизацию рутинных задач, таких как сборка проекта и обновление браузера.

Часть 2

Для решения задачи по созданию формы обратной связи и обработки данных через PHP был разработан набор файлов, включающий HTML-страницу с формой и PHP-скрипт для обработки данных. В процессе реализации была организована передача данных с использованием методов GET и POST, а также обеспечена запись полученной информации в файл на сервере.

HTML-файл (см. рисунок 5) включает форму, предоставляющую пользователю возможность ввода личных данных, сообщения, выбора источника информации (радиокнопки) и указания интересов (чекбоксы). Поля имени, фамилии, электронной почты и сообщения являются обязательными для заполнения благодаря атрибуту required. Радиокнопки позволяют выбрать, как пользователь узнал о сайте, а чекбоксы — указать интересующие тематики. Кнопка отправки формы передает данные на сервер через метод POST, указанный в атрибуте method тега <form>. Данные отправляются на обработку в файл feedback.php, заданный в атрибуте action.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Обратная связь</title>
</head>
<body>
    <h1>Форма обратной связи</h1>
    <form action="feedback.php" method="POST">
        <label for="first_name">Имя:</label><br>
        <input type="text" id="first_name" name="first_name" required><br><br>

        <label for="last_name">Фамилия:</label><br>
        <input type="text" id="last_name" name="last_name" required><br><br>

        <label for="email">Электронная почта:</label><br>
        <input type="email" id="email" name="email" required><br><br>

        <label for="message">Ваше сообщение:</label><br>
        <textarea id="message" name="message" rows="5" required></textarea><br><br>

        <p>Как вы узнали о нас?</p>
        <input type="radio" id="internet" name="source" value="Интернет" required>
        <label for="internet">Интернет</label><br>
        <input type="radio" id="friends" name="source" value="Друзья">
        <label for="friends">Друзья</label><br><br>

        <p>Что вас интересует?</p>
        <input type="checkbox" id="news" name="interests[]" value="Новости">
        <label for="news">Новости</label><br>
        <input type="checkbox" id="offers" name="interests[]" value="Спецпредложения">
        <label for="offers">Спецпредложения</label><br>
        <input type="checkbox" id="events" name="interests[]" value="Мероприятия">
        <label for="events">Мероприятия</label><br><br>

        <button type="submit">Отправить</button>
    </form>
</body>
</html>
```

Рисунок 5 – Содержимое файла feedback.html

В браузере данная форма выглядит следующим образом (см. рисунок 6):

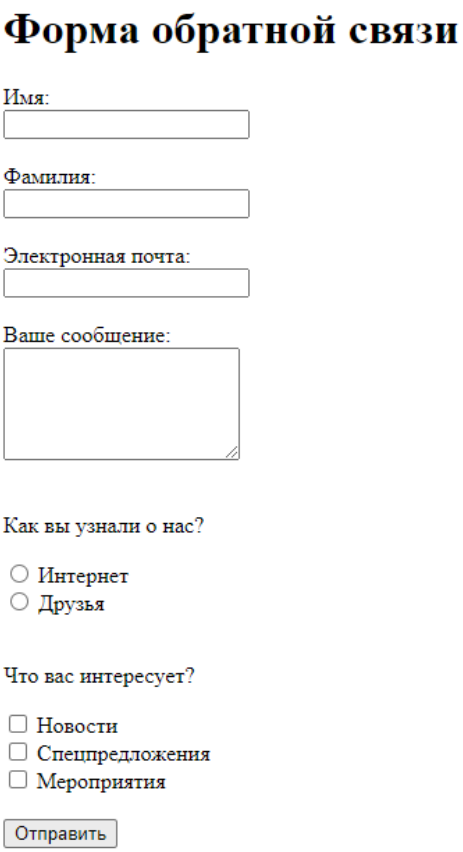
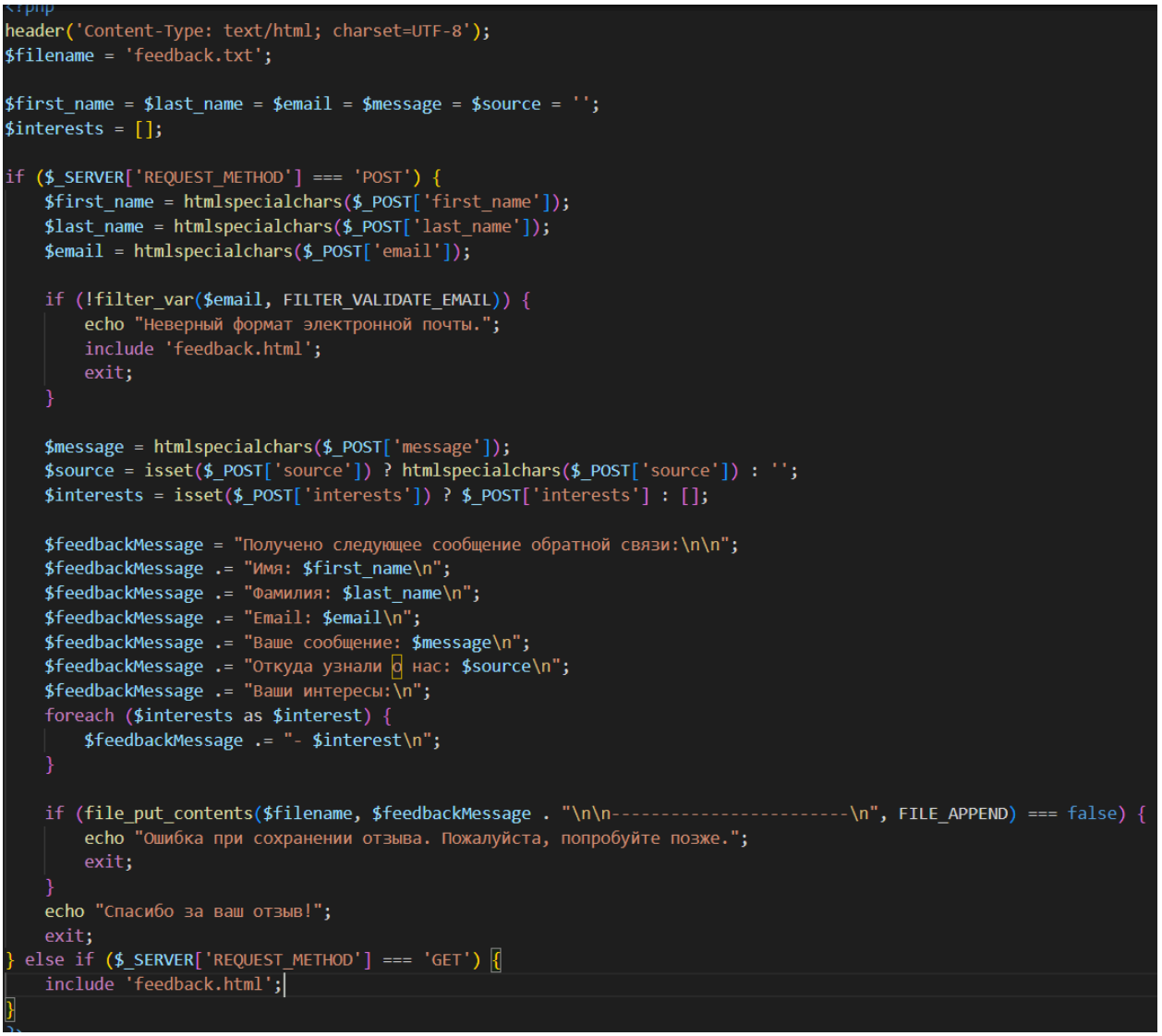


Рисунок 6 – Вид формы в браузере

PHP-скрипт (см. рисунок 7) обеспечивает обработку данных, переданных через форму. В зависимости от метода запроса (GET или POST), выполняются разные действия:

- **Метод GET:** При вызове страницы feedback.php напрямую или при ошибке ввода данных, HTML-форма отображается заново. Это полезно для удобного взаимодействия пользователя с формой.
- **Метод POST:** При отправке формы данные валидируются, чтобы предотвратить ошибки и возможные атаки. Например:
    - Поля обрабатываются функцией htmlspecialchars() для предотвращения XSS-атак.
    - Электронная почта валидируется с использованием функции filter_var().

После успешной проверки данные формируются в текстовое сообщение, которое добавляется в файл feedback.txt. Этот файл служит хранилищем отзывов и позволяет администратору просматривать их содержимое. Если при записи файла возникает ошибка, пользователю отображается сообщение о сбое.



```php
<?php
header('Content-Type: text/html; charset=UTF-8');
$filename = 'feedback.txt';

$first_name = $last_name = $email = $message = $source = '';
$interests = [];

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $first_name = htmlspecialchars($_POST['first_name']);
    $last_name = htmlspecialchars($_POST['last_name']);
    $email = htmlspecialchars($_POST['email']);

    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo "Неверный формат электронной почты.";
        include 'feedback.html';
        exit;
    }

    $message = htmlspecialchars($_POST['message']);
    $source = isset($_POST['source']) ? htmlspecialchars($_POST['source']) : '';
    $interests = isset($_POST['interests']) ? $_POST['interests'] : [];

    $feedbackMessage = "Получено следующее сообщение обратной связи:\n\n";
    $feedbackMessage .= "Имя: $first_name\n";
    $feedbackMessage .= "Фамилия: $last_name\n";
    $feedbackMessage .= "Email: $email\n";
    $feedbackMessage .= "Ваше сообщение: $message\n";
    $feedbackMessage .= "Откуда узнали о нас: $source\n";
    $feedbackMessage .= "Ваши интересы:\n";
    foreach ($interests as $interest) {
        $feedbackMessage .= "- $interest\n";
    }

    if (file_put_contents($filename, $feedbackMessage . "\n\n------------------------\n", FILE_APPEND) === false) {
        echo "Ошибка при сохранении отзыва. Пожалуйста, попробуйте позже.";
        exit;
    }
    echo "Спасибо за ваш отзыв!";
    exit;
} else if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    include 'feedback.html';
}
```

Рисунок 7 – Содержимое файла feedback.php

Пример использования методов GET и POST:

- **Метод GET:** Когда пользователь впервые заходит на страницу с формой или происходит перенаправление из-за неверного ввода данных, HTML-код формы загружается с использованием метода GET.
- **Метод POST:** При отправке данных формы с заполненными полями и выполнении всех проверок происходит их обработка, запись в файл, и пользователю отображается сообщение об успешной отправке отзыва.

После заполнения формы и отправки данных пользователь видит сообщение "Спасибо за ваш отзыв!". В это время информация записывается в файл feedback.txt. Пример записи в файле выглядит следующим образом (см. рисунок 8):

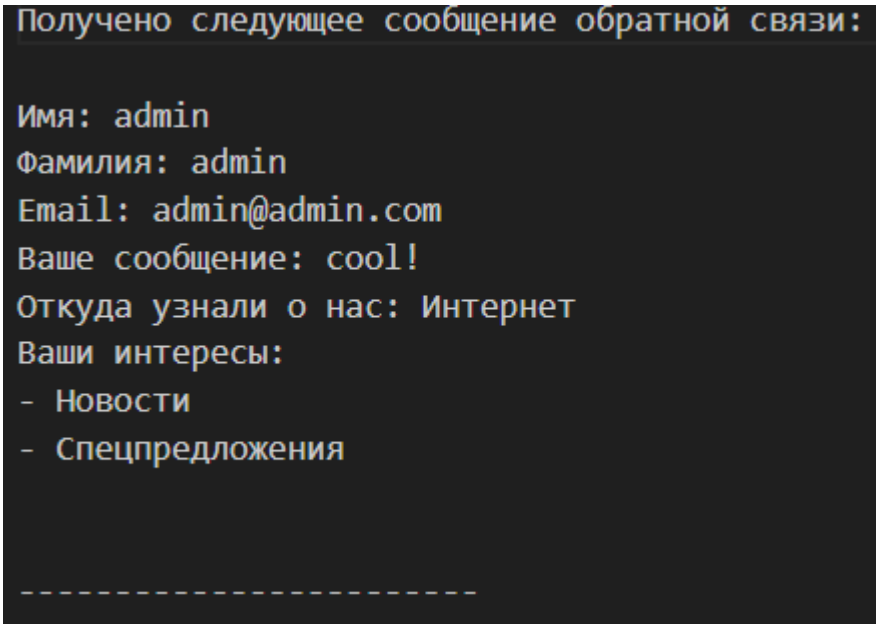


Рисунок 8 – Запись с информацией об обратной связи

Этот подход обеспечивает корректную обработку данных, удобство для пользователей и возможность их хранения для последующего анализа.

Часть 3

Для выполнения третьего задания была произведена установка локального сервера XAMPP для обеспечения среды разработки и отладки веб-проектов. XAMPP включает в себя необходимые инструменты, такие как Apache и MySQL, а также дополнительные модули для работы с PHP и серверной частью веб-приложений.

Сначала была загружена и установлена программа XAMPP. После завершения установки через XAMPP Control Panel были запущены модули Apache и MySQL (см. рисунок 6). Эти модули обеспечивают работу веб-сервера и системы управления базами данных, необходимых для последующей установки WordPress.

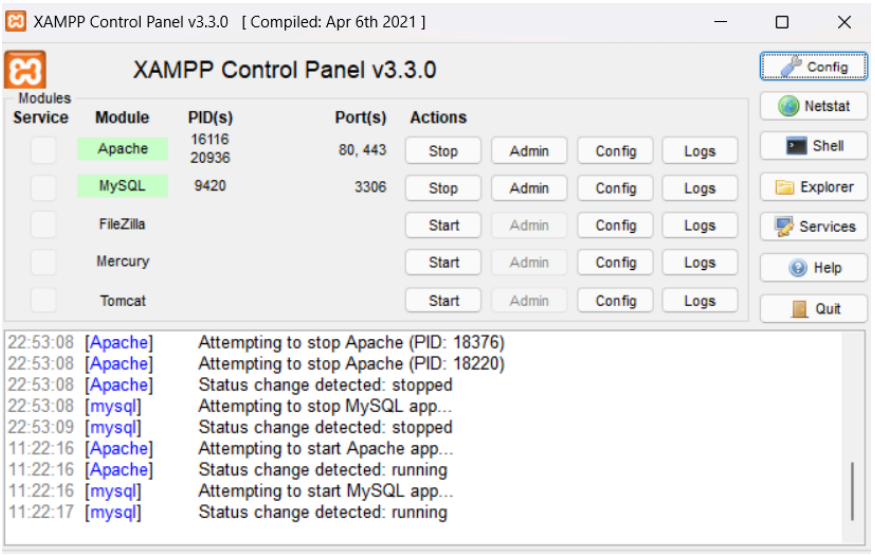


Рисунок 6 – XAMPP Control Panel.

Следующим шагом был скачан движок WordPress с официального портала. Архив с файлами был распакован, и содержимое перенесено в папку xampp\htdocs, которая используется XAMPP для размещения файлов веб-приложений.

Для корректной работы WordPress была создана база данных web_db в MySQL с помощью встроенного инструмента phpMyAdmin. После этого через веб-установщик WordPress была произведена настройка системы управления контентом, включающая подключение к созданной базе данных, создание администратора сайта и завершение установки.

Чтобы при вводе адреса http://test.site в браузере открывался установленный WordPress, была выполнена настройка виртуального хоста и сопутствующие изменения:

- Изменения в базе данных: В таблице wp_options базы данных web_db были изменены параметры siteurl и home, чтобы они указывали на адрес http://test.site (см. рисунок 9). Это позволяет WordPress корректно обрабатывать ссылки и ресурсы, привязанные к указанному доменному имени.
- Настройка виртуального хоста: В файле конфигурации Apache (httpd-vhosts.conf) был добавлен виртуальный хост, ассоциирующий имя test.site с локальной папкой, содержащей файлы WordPress. Пример настроек представлен на рисунке 10.
- Изменения в файле hosts: В системном файле hosts, который связывает доменные имена с IP-адресами, была добавлена строка 127.0.0.1 test.site. Это позволяет браузеру перенаправлять запросы на test.site на локальный сервер (см. рисунок 11).

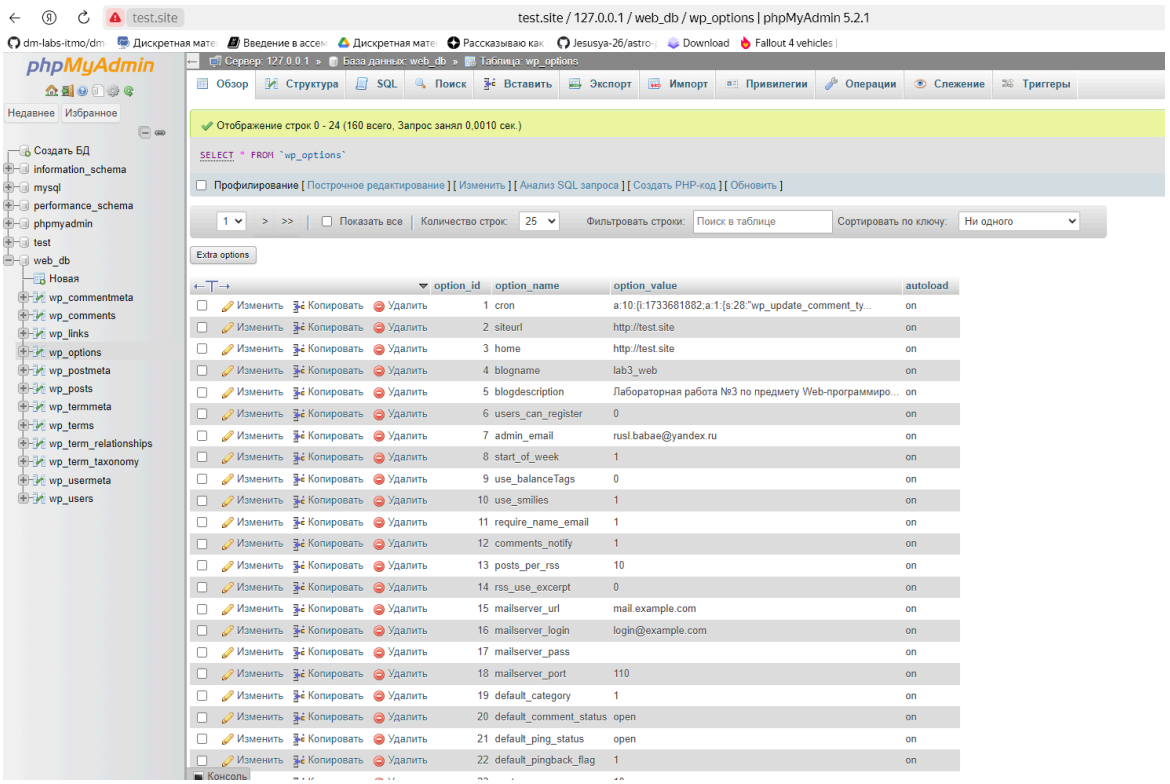


Рисунок 9 – Изменение параметров в таблице wp_options базы данных.

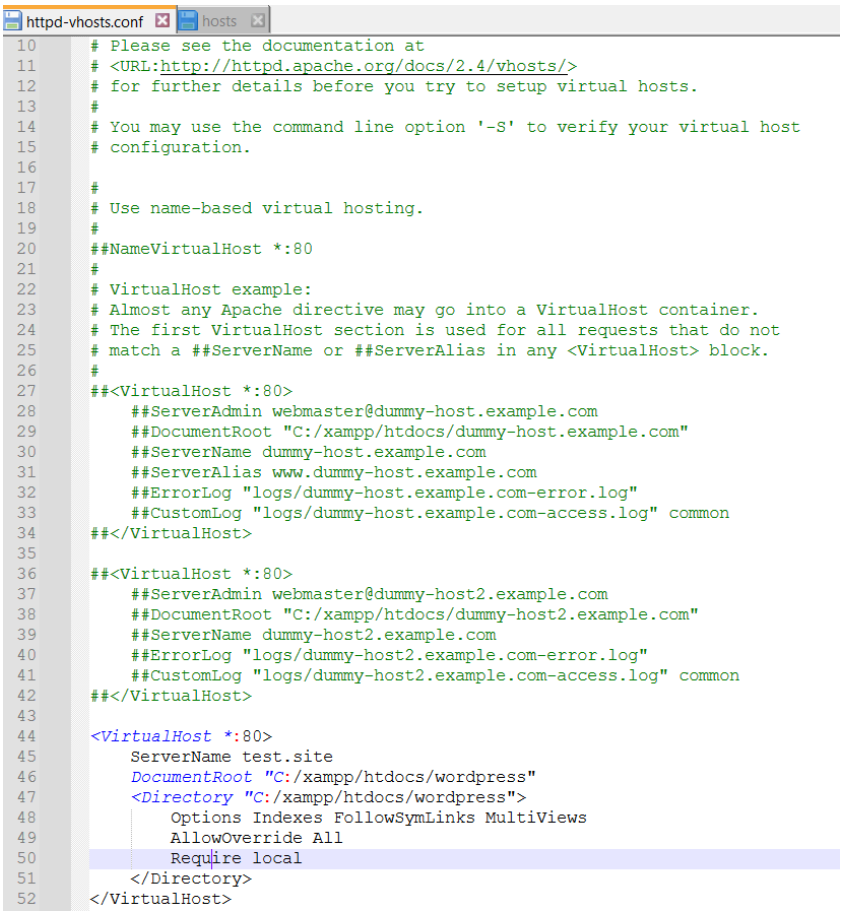


Рисунок 10 – Настройка виртуального хоста в файле httpd-vhosts.conf.

httpd-vhosts.conf | hosts

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#   127.0.0.1       localhost
#   ::1             localhost
# Added by Docker Desktop
192.168.31.45 host.docker.internal
192.168.31.45 gateway.docker.internal
# To allow the same kube context to work on the host and the container:
127.0.0.1 kubernetes.docker.internal
127.0.0.1 test.site
# End of section
```

Рисунок 11 – Добавление записи в файл hosts.

После завершения настройки при вводе адреса http://test.site в браузере стал отображаться установленный WordPress-сайт (см. рисунок 12). Для улучшения визуального оформления была выбрана и активирована одна из стандартных тем WordPress.

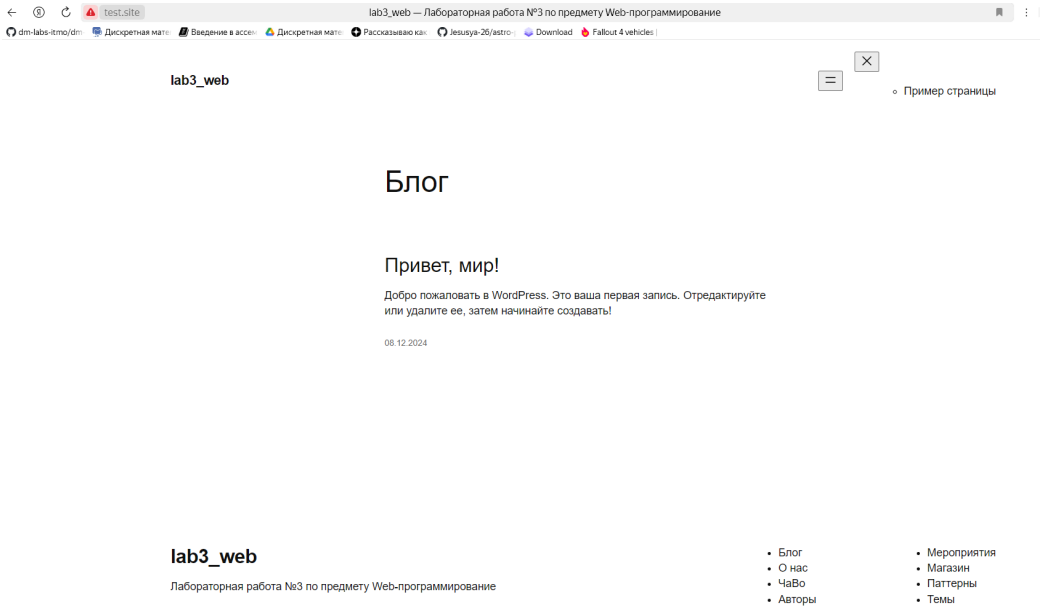


Рисунок 12 – Отображение сайта WordPress по адресу test.site.

В результате проделанной работы был настроен локальный сервер с установленным WordPress. Виртуальный хост позволяет открывать сайт по адресу http://test.site, что полностью соответствует требованиям задания. Такая конфигурация удобна для разработки и тестирования веб-приложений без необходимости использовать публичные серверы.

Заключение

В ходе выполнения работы были достигнуты все поставленные задачи, а также успешно реализована среда для локальной разработки веб-приложений с использованием XAMPP и WordPress. Установлен и настроен локальный сервер, обеспечивающий работу веб-сайта, который доступен по доменному имени http://test.site. Были выполнены все необходимые шаги по настройке базы данных, конфигурации виртуального хоста, а также интеграции WordPress с сервером Apache.

Особое внимание было уделено настройке системы таким образом, чтобы она обеспечивала корректную обработку доменного имени, ресурсов и ссылок, связанных с установленным сайтом. Эти действия позволяют использовать портал для дальнейших экспериментов с разработкой, тестированием плагинов, тем и других компонентов WordPress в безопасной локальной среде.