**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** Создание сайта по отправлению обратной связи

**Обучающийся** Алексеев Тимофей Юрьевич

**Факультет** Факультет инфокоммуникационных технологий

**Группа** К3221

**Направление подготовки** 11.03.02 Инфокоммуникационные технологии и системы связи

**Образовательная программа** Программирование в инфокоммуникационных системах

**Обучающийся** 11.11.2024  Алексеев Т.Ю.

(дата) (подпись) (Ф.И.О.)

**Руководитель**  Марченко Е.В.

(дата) (подпись) (Ф.И.О.)

Санкт-Петербург  
2024 г.

# Цель

Создать сайт для отправки информации по обратной связи, используя для этого gulp, php-скрипт и инструментарий для отладки проектов.

# Задачи

1. Ознакомиться c основными запросами Git;
2. Создать репозиторий, сделать 3 коммита и синхронизировать;
3. Установить gulp;
4. Создать простую задачу с помощью gulp;
5. Создать переключатель веб-страниц с определенным интервалом с помощью gulp.

# Ход работы

**Задание 1**

В данном задании было необходимо произвести работу с последовательными и параллельными функциями.

Для начала было необходимо создать одну параллельную и одну последовательную задачу. В Gulp за это отвечают series() и parallel().

Для выполнения данной задачи были написаны две задачи: hello и bye, которые выводят приветственное и прощальное сообщения соответственно. После чего были созданы задачи, включающие две вышеупомянутые задачи и использующие series() и parallel(), - orderTasks() и parallelTasks().

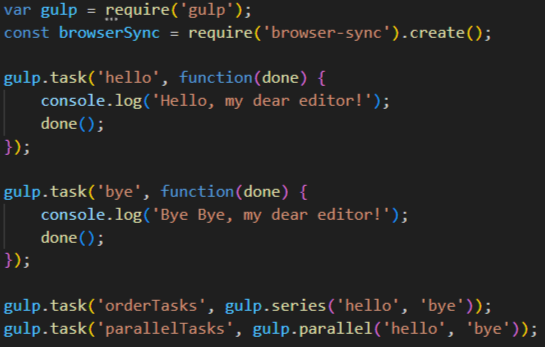


Рисунок 1 – Параллельные и последовательные задачи

Проверим, что задачи выполняются ровно в том порядке, как мы и хотим. Для этого вызовем их в консоли с помощью npm run orderTasks и npm run parallelTasks.

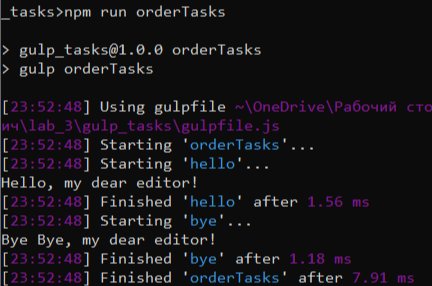


Рисунок 2 – Последовательное выполнение задач

Как мы видим, задачи действительно выполняются по очереди, когда заканчивает выполнение одна, вторая сразу же начинает своё выполнение.

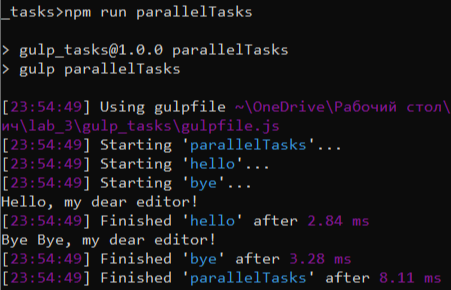


Рисунок 3 – Параллельное выполнение задач

Здесь же ситуация обратная. Сначала запускаются все задачи, после чего происходит их одновременное выполнение.

После этого необходимо было создать задачу, при выполнении который в браузере отображались бы файлы проекта и происходило обновление при каждом изменении.

Для этого в папке app были созданы файлы index.html с тегом <h2> и текстом Hello world и script.js с выводом аналогичного текста в консоль.

Далее были созданы задачи html, script и server.

В задачах html и script просто добавляются соответствующие файлы в поток для отображения их в браузере с помощью browserSync.

В задаче server запускается локальный сервер и папка, в которой он базируется. А также прописываются методы watch, следящие за изменениями в html- и js-файлах и перезагружающие страницу, если таковые произошли.

После чего была написана главная задача startWeb, объединяющая все подзадачи. В ней последовательно вызываются параллельное выполнение задач script и html, а потом выполнение задачи server.



Рисунок 4 – Gulpfile для отображения файлов проекта

При выполнении задачи startWeb будет открываться следующая страница. При внесении изменений в файлы страница автоматически обновляется.

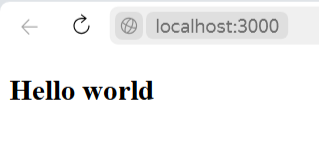


Рисунок 5 – Открывающаяся страница в браузере

**Задание 2**

В этом задании было необходимо реализовать форму обратной связи, которую можно было бы отправить и получить информацию с помощью php-скрипта.

Для начала необходимо было запустить локальный сервер. Для это задачи я выбрал XAMPP из-за простоты работы с ним, графического интерфейса и прозрачной настройки. Для запуска проекта необходимо перенести все файлы запускаемого проекта в папку htdocs.

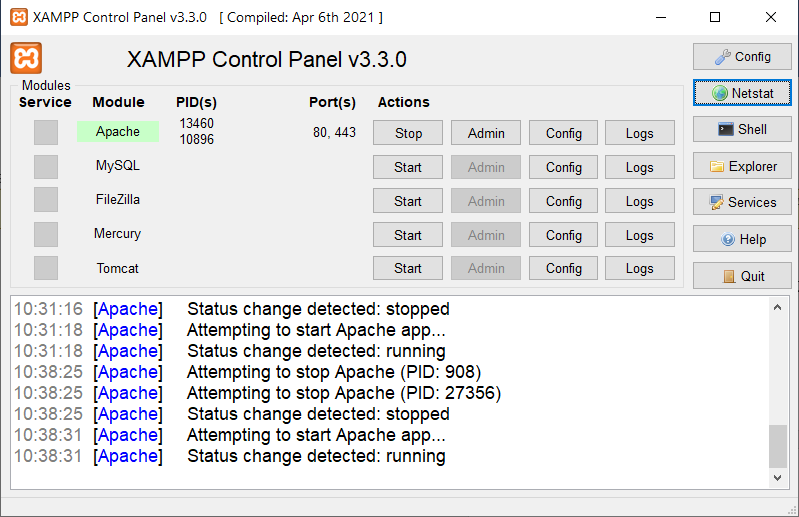
****

Рисунок 5 – Запуска локального сервера с помощью XAMPP

После этого был написан index.html, который и содержал начальную форму. Там пользователю предлагается заполнить имя, фамилию, почту, развернутую обратную связь в textarea, выбрать общее впечатление от мероприятия с помощью radio-input и выбрать понравившиеся аспекты с помощью checkbox-input. Форма отправляется по нажатию на соответствующую кнопку.

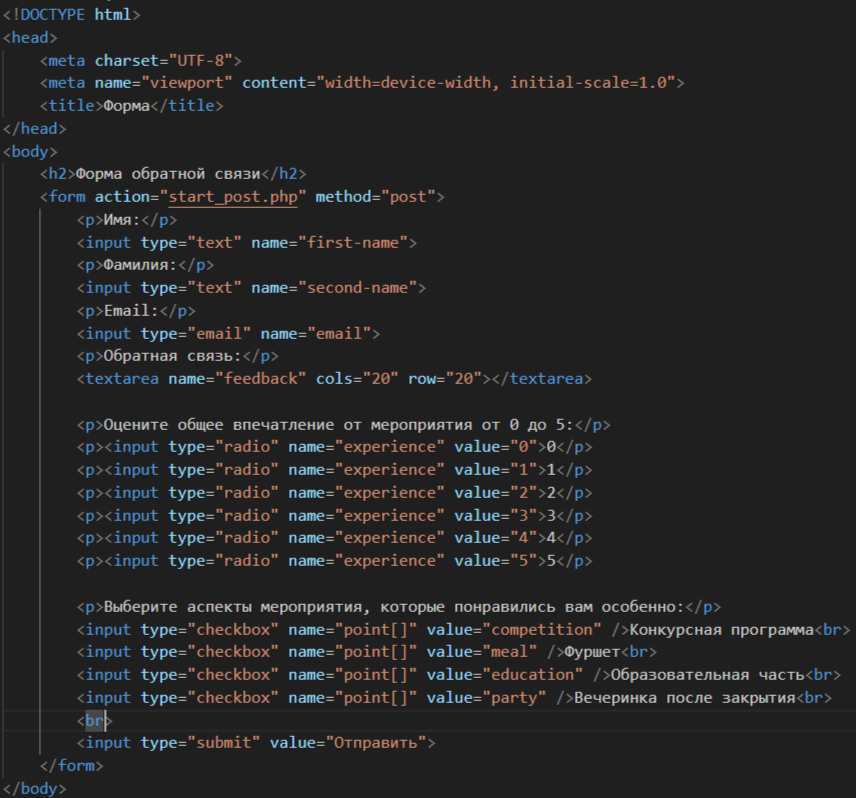
****

Рисунок 6 – HTML-код формы обратной связи

Далее необходимо написать PHP-скрипт, который обрабатывает post- и get-запросы. В post-запросе сначала создаются все переменные, которые были получены после отправления формы. Поле со множественным выбором сначала проверяется на пустоту, после чего идет подсчет выбранных аспектов. После этого происходит запись в txt-файл.

В get-запросе обратная логика. Он вызывается при нажатии на кнопку на новой отрисованной странице с заголовком «Спасибо за ваш отзыв!». После этого считывается вся информация из txt-файла и выводится на экран пользователя. Выводится это с помощью тега <pre> для сохранения исходного форматирования.

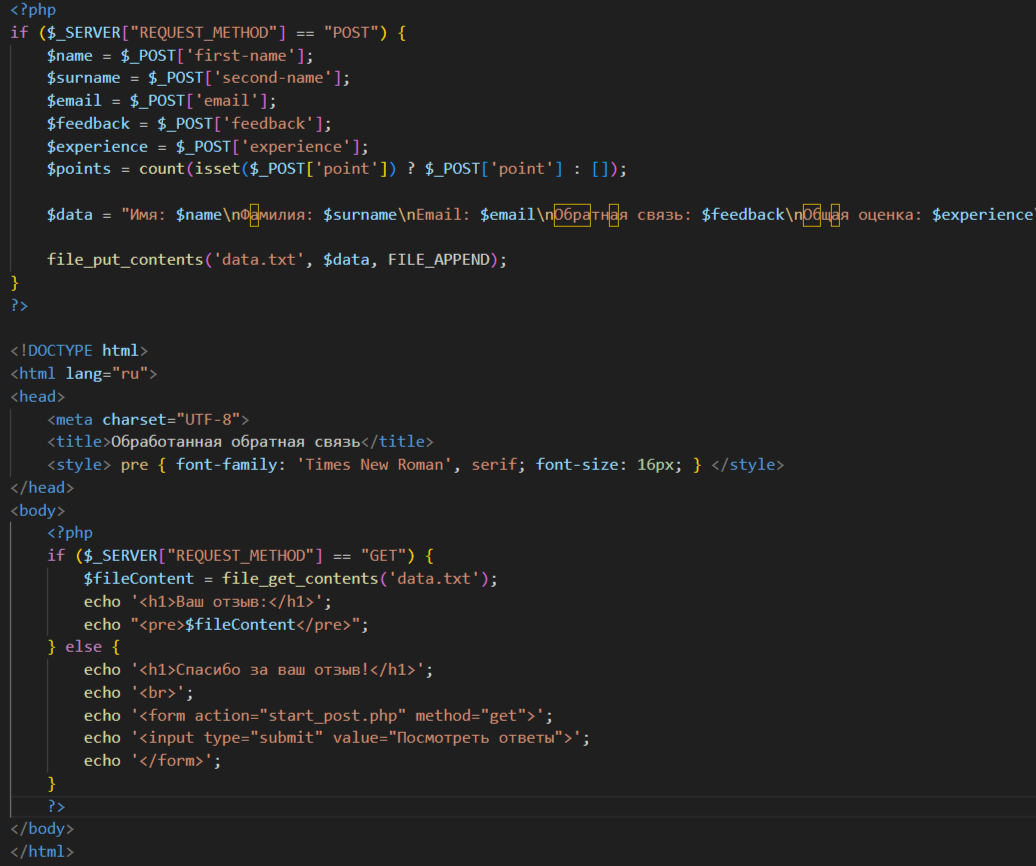
****

Рисунок 7 – Код PHP-скрипта

В данный момент стартовая страница выглядит следующим образом.

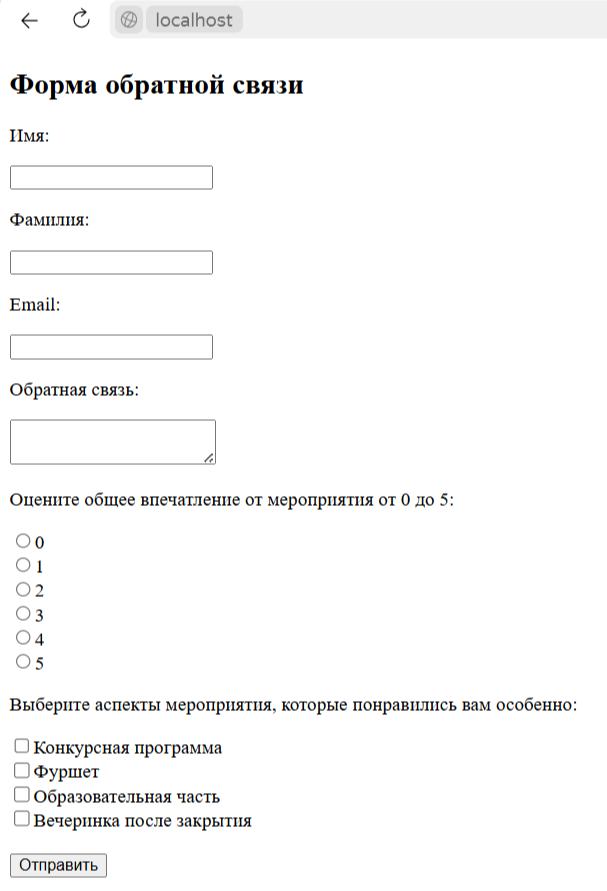
****

Рисунок 8 – Страница сбора обратной связи

После отправки информации появляется следующая страница.

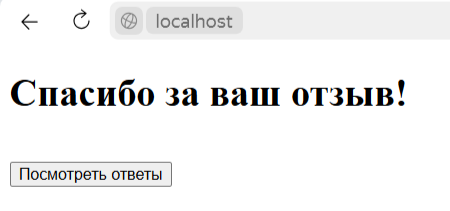
****

Рисунок 9 – Страница-ответ на отправку формы

И при нажатии на кнопку «Посмотреть ответы» открывается страница с формой пользователя.

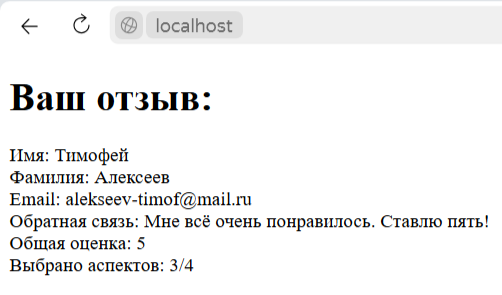
****

Рисунок 10 – Страница с пользовательским запросом

**Вывод**

В ходе выполнения лабораторной работы цель была достигнута. Было произведено ознакомление с основами работы с Git и Gulp.

В течение выполнения работы были изучены основные команды Git, создан репозиторий, сделаны коммиты и произведена синхронизация удаленного и локального репозиториев. Установлен gulp и написана первая задача, выводящая приветственное сообщение. Написана задача, которая вызывает веб-сервер, который переключает сайты из списка с определенным интервалом.