УДК 004

Автоматизация и оптимизация веб-сайтов с помощью системы сборки задач Gulp

Халиманенков Андрей Сергеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассматривается автоматизация создания веб-сайта. Для этой цели используется сборщик задач Gulp и менеджер пакетов NPM. Итогом исследования является настроенный сборщик задач, который автоматически оптимизирует исходные файлы сайта.
Ключевые слова: создание веб-сайтов, Gulp, система сборки задач, NPM.

Automation and optimization of websites using the Gulp Task Assembly system

Khalimanenkov Andrey Sergeevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article discusses the automation of creating a website. For this purpose, the Gulp task collector and the NPM package manager are used. The result of the study is a customized task collector that automatically optimizes the source files of the site.
Keywords: websites creating, Gulp, task assembly system, NPM.

Во времена становления интернет технологий, все веб-сайты состояли из нескольких файлов и были крайне примитивны в своей сути. Но с развитием всемирной паутины появилось множество технологий для построения веб-приложений. Это и различные фреймворки, и языки программирования, а также подходы к архитектуре веб-сайтов. Количество файлов для одного сайта растёт с каждым годом и на данный момент существует проблема в их поддержке и редактировании содержимого, т.к. на одну страницу могут влиять несколько JavaScript файлов, при этом в каждом есть зависимость относительно остальных, например, ссылки на функции из других файлов. Помимо трудностей в поддержке, также это влияет на скорость загрузки веб-сайтов в худшую сторону, т.к. лучше загрузить один файл на 1000 килобайт, чем 100 файлов по 10 килобайт. Для таких случаев существуют так называемые «сборщики». Это модульное ПО, которое помогает сжимать картинки, объединять файлы, превращать scss код в css и множество других полезных решений.

Цель исследования – автоматизировать оптимизацию веб-сайта в сборщике задач Gulp с помощью пакетного менеджера NPM для Node.js и главного JavaScript файла, в котором описывается сценарий выполнения функций и модулей.

Вопрос автоматизации создания сайтов волнует многих исследователей и специалистов: Ю.Н. Косников и В. В. Мелешкин [1] Для облегчения работы дизайнера созданы программные генераторы шаблонов интерфейсов. А. В Харченко [2] провёл исследование инструмента Just Scroll Me, который используется для автоматизации создания анимаций с прокруткой. В. Л. Просветов и Н. Е. Конева [3] рассмотрели способы автоматизации сбора данных на веб-сайтах. В. А. Отдельный и И. А. Евсеенко [4] предложили методику автоматизированного тестирования для решения задачи ускорения процесса разработки веб-приложения путем повышения эффективности тестирования для обеспечения качества разрабатываемого продукта. Е. А. Арсирий [5] автоматизировал разработку и обновления семантического ядра сайта с динамическим контентом.

Для начала использования Gulp нужно установить программную платформу Node.js, чтобы операционная система могла интерпретировать JavaScript код вне браузера в машинный код. Для этого переходим по адресу https://nodejs.org/en/ [6], скачиваем дистрибутив с постфиксом LTS, т.к. он является стабильной версией, в которой нет критических ошибок и устанавливаем его.

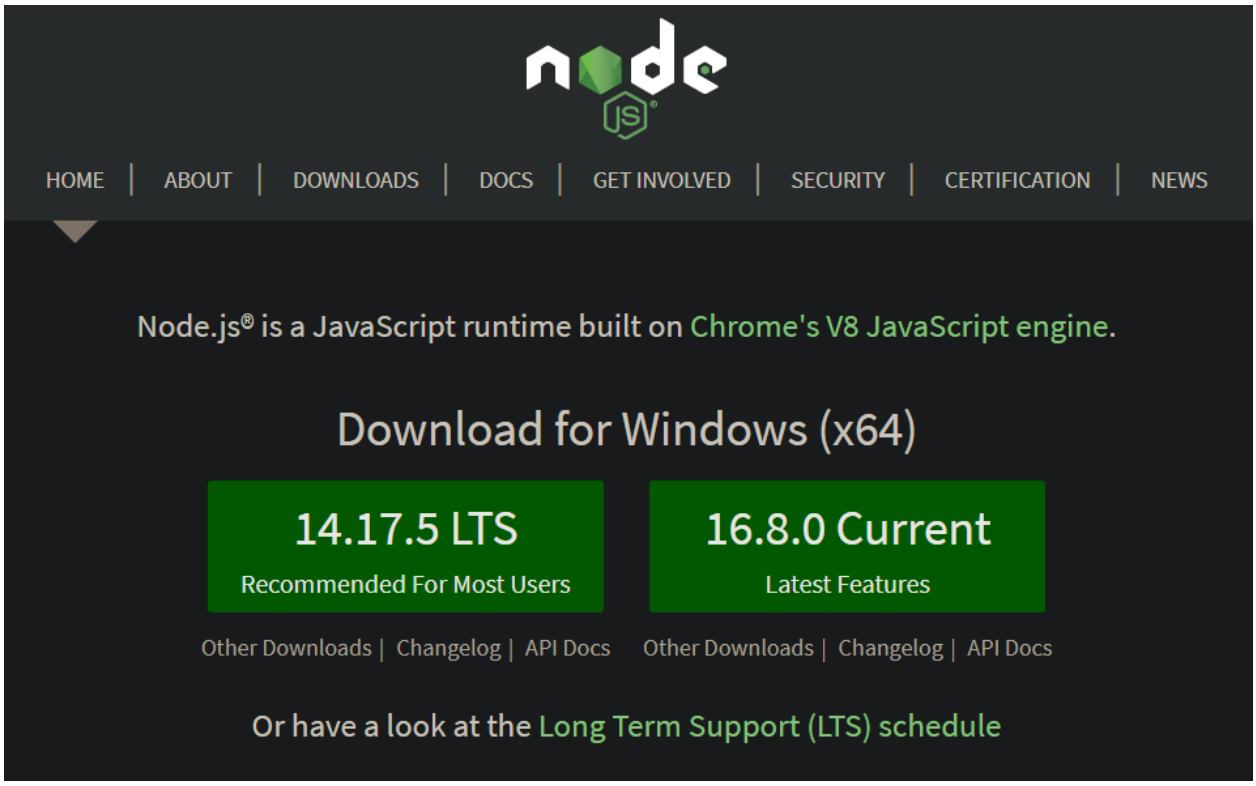

Рисунок 1 – Сайт для загрузки Node.js

После этого нужно глобально установить Gulp в ОС. Для этого следует запустить Windows PowerShell с правами администратора или аналог в других ОС и вставить команду «Set-ExecutionPolicy RemoteSigned» для изменения политики выполнения, которая даст возможность устанавлить пакеты и приложения из репозиториев по аналогии с UNIX системами, такими как Linux или MacOS.

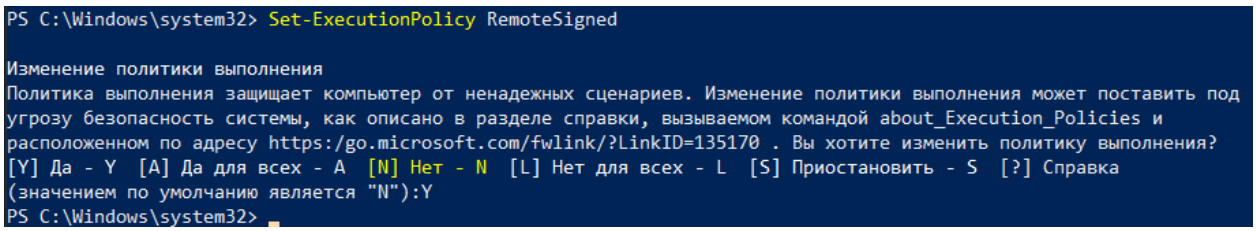

Рисунок 2 – Изменение политики выполнения

Затем нужно указать команду для установки Gulp глобально в ОС с доступом к консоли Gulp - «npm install –global gulp-cli».

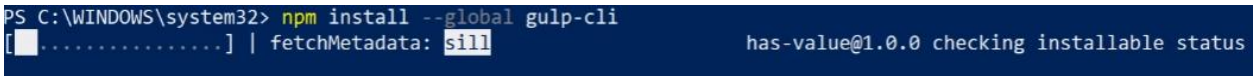

Рисунок 3 – Установка Gulp

Следующий шаг – создание конфигурационных файлов Gulp, в которых будут записанным различные модули, а также данные о сайте, авторе, лицензии распространения, версии веб-сайта и т.д. В консоль вводим команду «npm init», которая запускает процесс конфигурации в папке с будущим сайтом и создаёт файл «package.json».

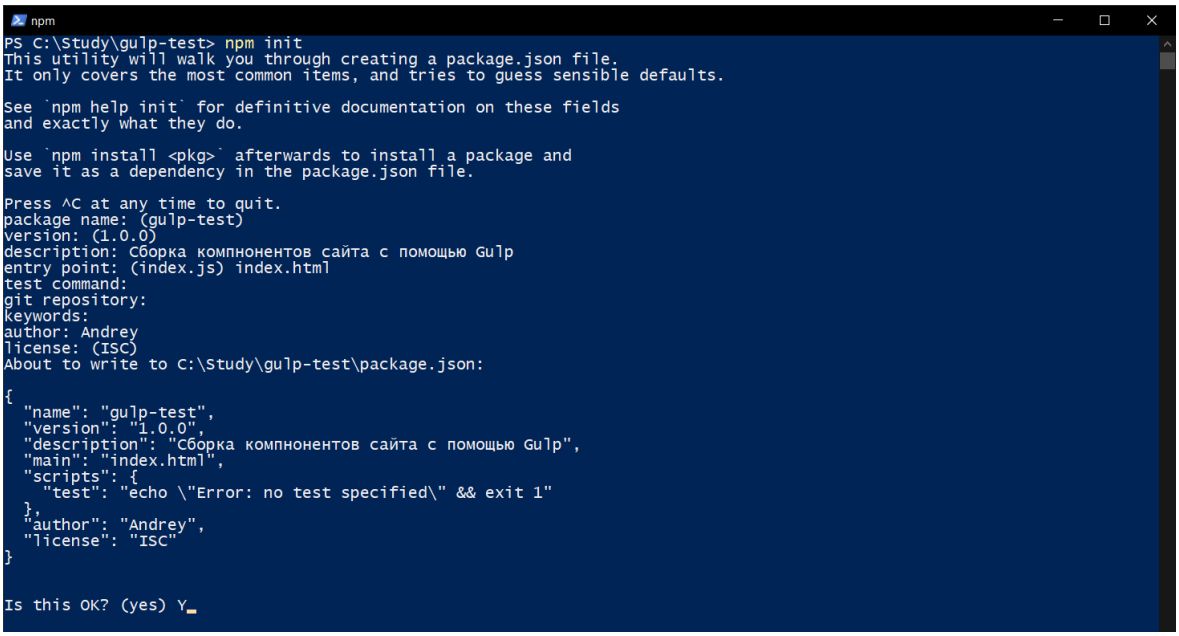

Рисунок 4 – Конфигурация Gulp

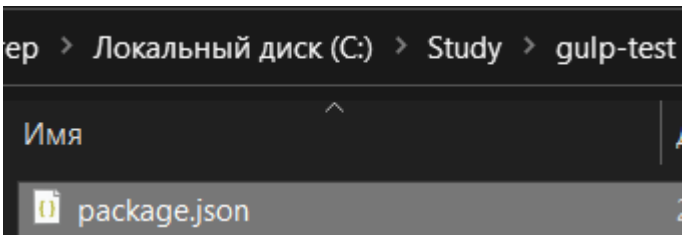

Рисунок 5 – Созданный файл

Затем нужно открыть папку проекта в редакторе кода. В этой статье используется VSCode. В терминале редактора кода нужно ввести следующую команду – «npm i gulp –save-dev», что установит Gulp в директорию сайта и создаст папку с npm модулями.

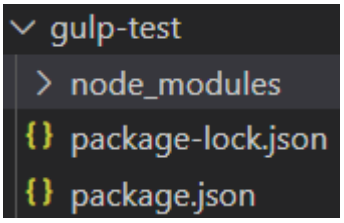

Рисунок 6 – Файлы конфигурации и папка с модулями

После этого шага нужно создать папки для хранения скриптов, картинок, шрифтов и т.д. и установить все необходимые модули, которые являются идеальной основой для старта в автоматизированном создании оптимизированных сайтов. Команда «npm i --save-dev @babel/core @babel/preset-env @babel/preset-react browser-sync del gulp-autoprefixer gulp-babel gulp-clean-css gulp-file-include gulp-fonter gulp-group-css-media-queries gulp-imagemin gulp-rename gulp-sass gulp-svg-sprite gulp-ttf2woff gulp-ttf2woff2 gulp-uglify-es gulp-webp gulp-webp-html gulp-webpcss sass webp-converter» сделает это.

После этой команды файл package.json пример следующее содержание:

```
{
 "name": "andrey-gulp ",
 "version": "1.0.0",
 "description": "Сборка GULP",
 "main": "index.html",
 "scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
 },
 "author": "Andrey",
 "license": "ISC",
 "devDependencies": {
  "@babel/core": "^7.15.0",
  "@babel/preset-env": "^7.15.0",
  "@babel/preset-react": "^7.14.5",
  "browser-sync": "^2.27.5",
```

```
    "del": "^6.0.0",
    "gulp": "^4.0.2",
    "gulp-autoprefixer": "^8.0.0",
    "gulp-babel": "^8.0.0",
    "gulp-clean-css": "^4.3.0",
    "gulp-file-include": "^2.3.0",
    "gulp-fonter": "^0.3.0",
    "gulp-group-css-media-queries": "^1.2.2",
    "gulp-imagemin": "^7.1.0",
    "gulp-rename": "^2.0.0",
    "gulp-sass": "^5.0.0",
    "gulp-svg-sprite": "^1.5.0",
    "gulp-ttf2woff": "^1.1.1",
    "gulp-ttf2woff2": "^4.0.1",
    "gulp-uglify-es": "^3.0.0",
    "gulp-webp": "^4.0.1",
    "gulp-webp-html": "^1.0.2",
    "gulp-webpcss": "^1.1.1",
    "sass": "^1.38.1",
    "webp-converter": "^2.2.3"
  }
}
```


В этом списке есть модули:

1. Преобразование JS кода из версии ES6+ в версию ES5, для того, чтобы максимальное количество браузеров поддерживало JS код;
2. Обновление страницы при внесении изменений в файлах;
3. Добавление вендорных префиксов в CSS;
4. Сжатие CSS файлов для ускорения обработки;
5. Подключение html кода из других файлов в редактируемый файл;
6. Добавление списка шрифтов в CSS;
7. Объединение одинаковых media-запросов, которые разбиты на части;
8. Сжатие изображений без заметной потери качества;
9. Добавление поддержки SCSS файлов и трансляция в CSS код;
10. Добавление всех SVG иконок в один файл с секторным разбиением;
11. Преобразование TTF шрифтов в WOFF, для ускорения загрузки сайтов;
12. Сжатие JS файлов для ускорения обработки;
13. Поддержка WEBP изображений, которые занимают меньше места и не теряют в качестве, а также преобразование всех расширений изображений в WEBP;

По итогу директория будет выглядеть следующим образом.

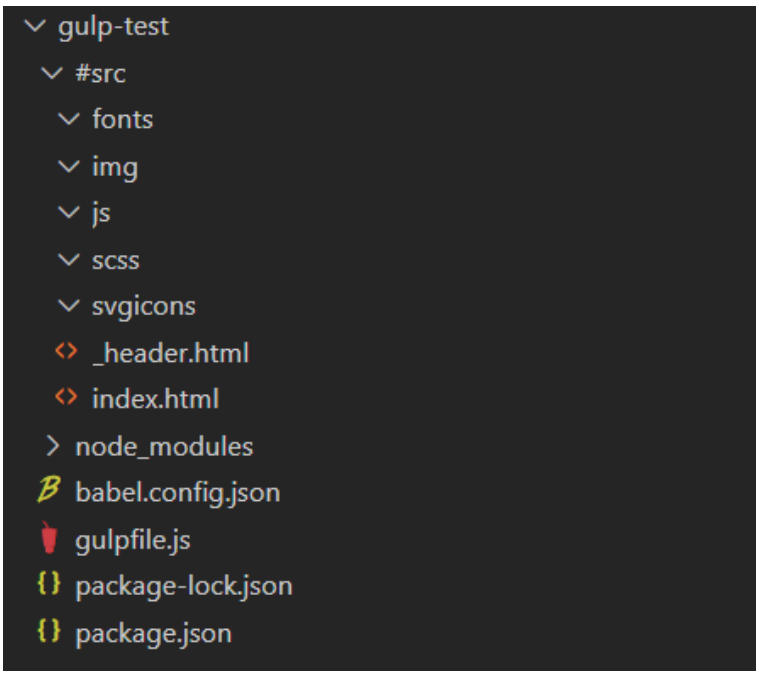

Рисунок 7 – Директория сайта

Добавим иконки, картинки, скрипты и шрифты в папку #src, которая является папкой с исходными и необработанными файлами.

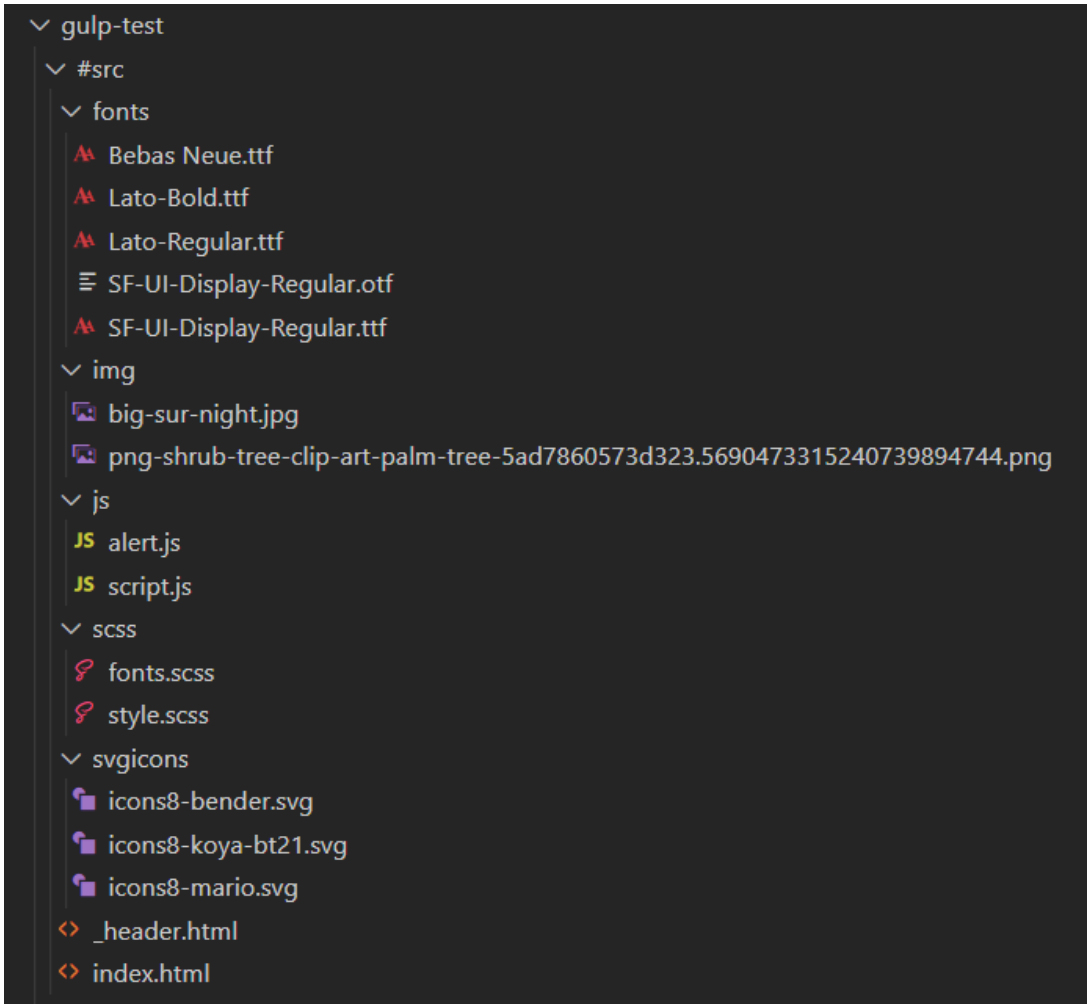

Рисунок 8 – Директория сайта с исходными файлами

После этого нужно создать главный файл «gulpfile.js» со сценарием, в котором будет описан алгоритм работы сборщика. Для того, чтобы всё работало, нужно описать сценарий использования модулей. Код выглядит следующим образом.

```
const fileinclude = require('gulp-file-include');
let project_folder= require("path").basename(__dirname);
let source_folder= "#src";
let fs = require('fs');
let path= {
   build: {
      html: project_folder + "/",
      css: project_folder + "/css/",
      js: project_folder + "/js/",
      img: project_folder + "/img/",
      fonts: project_folder + "/fonts/",
   },
   src: {
      html: [source_folder + "/*.html", "!" + source_folder + "/_*.html"],
      css: source_folder + "/scss/style.scss",
      js: source_folder + "/js/script.js",
      img: source_folder + "/img/**/*.{jpg,png,svg,gif,ico,webp}",
      fonts: source_folder + "/fonts/*.ttf",
      svgicons: source_folder + "/svgicons/**/*.svg",
   },
   watch: {
      html: source_folder + "/**/*.html",
      css: source_folder + "/scss/**/*.scss",
      js: source_folder + "/js/**/*.js",
      img: source_folder + "/img/**/*.{jpg,png,svg,gif,ico,webp}",
      svgicons: source_folder + "/svgicons/**/*.svg",
      fonts: source_folder + "/fonts/*.ttf",
   },
   clean:"./" + project_folder + "/"
}

let { src, dest } = require('gulp'),
   gulp = require('gulp'),
   browsersync = require("browser-sync").create(),
   del = require("del"),
   scss = require ('gulp-sass')(require ('sass')),
   autoprefixer = require('gulp-autoprefixer'),
   clean_css = require('gulp-clean-css'),
   rename = require('gulp-rename'),
   uglify = require('gulp-uglify-es').default,
   babel = require("gulp-babel"),
   imagemin = require('gulp-imagemin'),
   webp = require('gulp-webp'),
   webphtml = require('gulp-webp-html'),
```

```
    webpcss = require('gulp-webpcss'),
    svgSprite = require('gulp-svg-sprite'),
    ttf2woff = require('gulp-ttf2woff'),
    ttf2woff2 = require('gulp-ttf2woff2'),
    fonter = require('gulp-fonter'),
    group_media = require('gulp-group-css-media-queries');

function browserSync(params) {
    browsersync.init({
        server: {
            baseDir: "./" + project_folder + "/"
        },
        port: 3000,
        notify: false
    })
}

function html() {
    return src(path.src.html)
        .pipe(fileinclude())
        .pipe(webphtml())
        .pipe(dest(path.build.html))
        .pipe(browsersync.stream())
}

function css() {
    return src(path.src.css)
        .pipe(
            scss({
                outputStyle: "expanded"
            }).on('error', scss.logError)
        )
        .pipe(
            group_media()
        )
        .pipe(
            autoprefixer({
                overrideBrowserslist: ["last 10 versions"],
                cascade: true
            })
        )
        .pipe (webpcss(
            {webpClass: '.webp',noWebpClass: '.no-webp'
            })
        )
        .pipe(dest(path.build.css))
        .pipe(clean_css())
        .pipe(
            rename({
```

```
            extname: ".min.css"
         })
      )
      .pipe(dest(path.build.css))
      .pipe(browsersync.stream())
}

function js() {
   return src(path.src.js)
      .pipe(fileinclude())
      .pipe(babel())
      .pipe(dest(path.build.js))
      .pipe(
         uglify()
      )
      .pipe(
         rename({
            extname: ".min.js"
         })
      )
      .pipe(dest(path.build.js))
      .pipe(browsersync.stream())
}

function images() {
   return src(path.src.img)
      .pipe(
         webp({
            quality: 70
         })
      )
      .pipe(dest(path.build.img))
      .pipe(src(path.src.img))
      .pipe(
         imagemin({
            progressive: true,
            svgoPlugins: [{ removeViewBox: false}],
            interlaced: true,
            optimizationLevel: 3
         })
      )
      .pipe(dest(path.build.img))
      .pipe(browsersync.stream())
}

function fonts() {
   src(path.src.fonts)
      .pipe(ttf2woff())
      .pipe(dest(path.build.fonts));
```

```
    return src(path.src.fonts)
        .pipe(ttf2woff2())
        .pipe(dest(path.build.fonts));
}

/* gulp.task('svgSprite', function(){
    return gulp.src([source_folder + '/iconsprite/*.svg'])
        .pipe(svgSprite({
            mode: {
                stack: {
                    sprite: "../icons/icons.svg",
                    example: true
                }
            },
        }
        ))
        .pipe(dest(path.build.img))
}) */

function svgIcons(params){
    return gulp.src([source_folder + '/svgicons/*.svg'])
        .pipe(svgSprite({
            mode: {
                stack: {
                    sprite: "../icons/icons.svg",
                    example: true
                }
            },
        }
        ))
        .pipe(dest(path.build.img))
}

function otf2ttf(params){
    return src([source_folder + '/fonts/*.otf'])
        .pipe(fonter({
            formats:['ttf']
        }))
        .pipe(dest(source_folder + '/fonts/'));
}

function fontsStyle(params) {

    let file_content = fs.readFileSync(source_folder + '/scss/fonts.scss');
    if (file_content == '') {
        fs.writeFile(source_folder + '/scss/fonts.scss', '', cb);
        return fs.readdir(path.build.fonts, function (err, items) {
            if (items) {
                let c_fontname;
```

```
            for (var i = 0; i < items.length; i++) {
                let fontname = items[i].split('.');
                fontname = fontname[0];
                if (c_fontname != fontname) {
                    fs.appendFile(source_folder + '/scss/fonts.scss', '@include font("' + fontname + '", "' + fontname + '", "400", "normal");\r\n', cb);
                }
                c_fontname = fontname;
            }
        }
    })
  }
}



function cb() {

}

function watchFiles(params) {
  gulp.watch([path.watch.html], html);
  gulp.watch([path.watch.css], css);
  gulp.watch([path.watch.js], js);
  gulp.watch([path.watch.img], images);
  gulp.watch([path.watch.svgicons], svgIcons);
  gulp.watch([path.watch.fonts], fonts);
}

function clean(params) {
  return del(path.clean);
}

let build = gulp.series(clean, gulp.parallel(js, css, html, images, otf2ttf, fonts, svgIcons), fontsStyle);
let watch = gulp.parallel(build, watchFiles, browserSync);

exports.fontsStyle = fontsStyle;
exports.otf2ttf = otf2ttf;
exports.svgIcons = svgIcons;
exports.fonts = fonts;
exports.images = images;
exports.js = js;
exports.css = css;
exports.html = html;
exports.build = build;
exports.watch = watch;
exports.default = watch;
```

Для запуска локального сервера для отображения сайта нужно ввести команду «gulp» в терминал редактора кода.

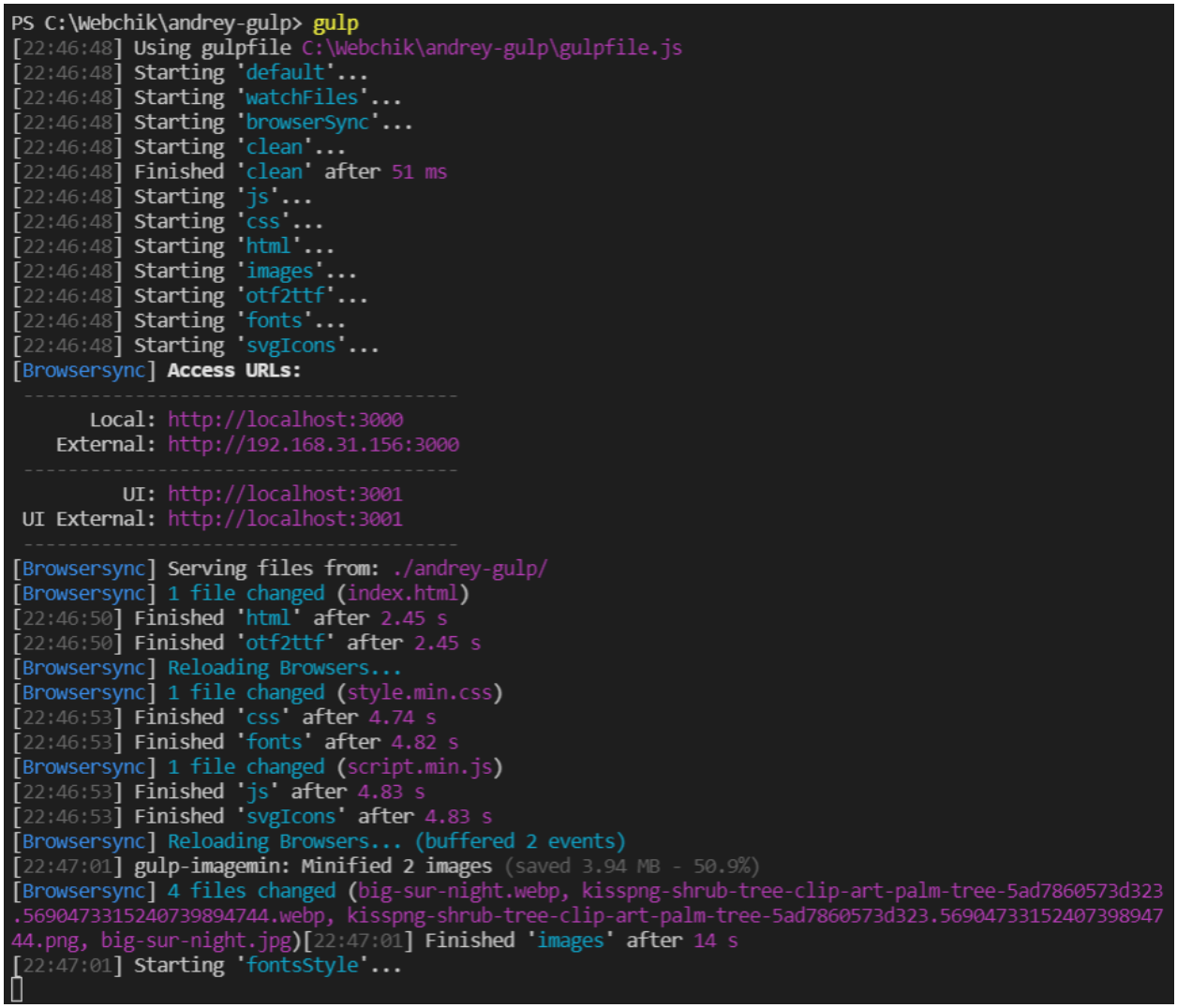

Рисунок 9 – Запуск сервера

Готовая директива сайта выглядит следующим образом.

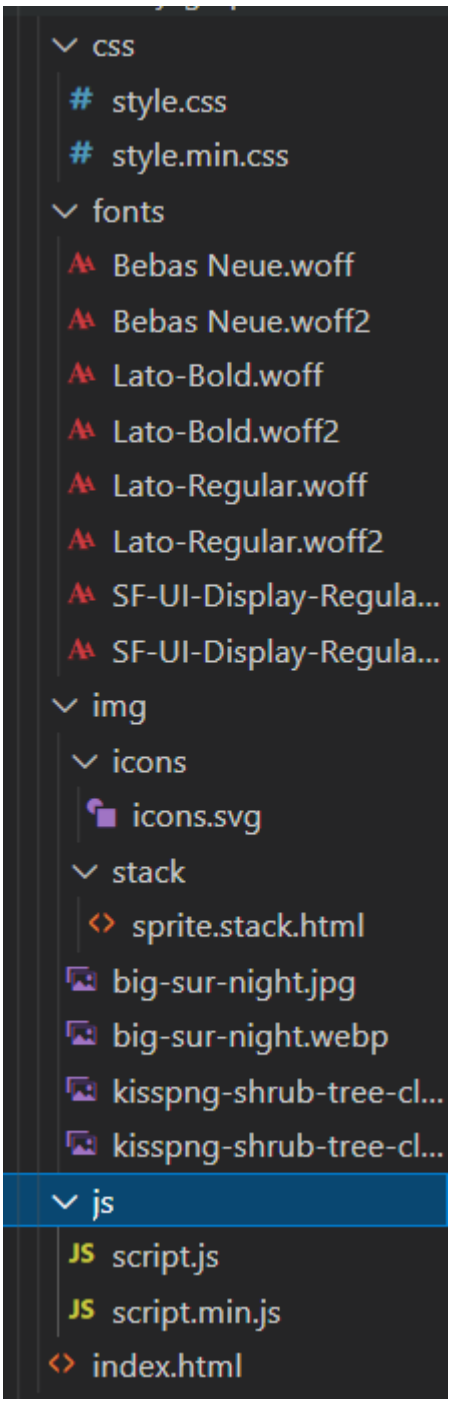

Рисунок 10 – Оптимизированные файлы

Изначально было два HTML-файла – index и header. Но после оптимизации они объединились в один.

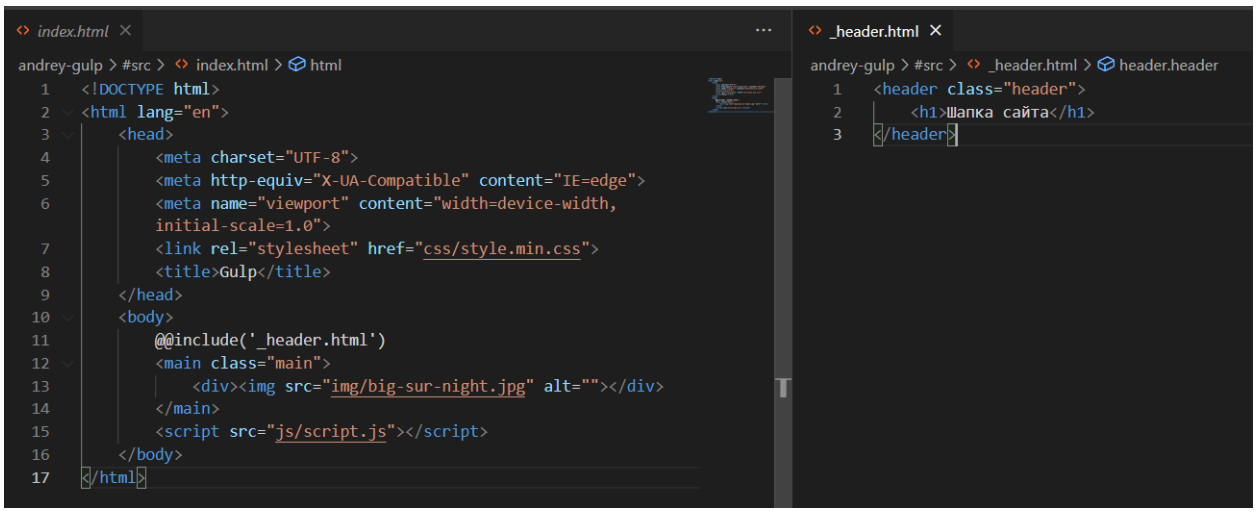

Рисунок 11 – HTML файлы из папки исходников

Рисунок 11 – HTML файлы из папки оптимизированного сайта

Сжатие изображений в формат webp четырёхкратно экономит место.

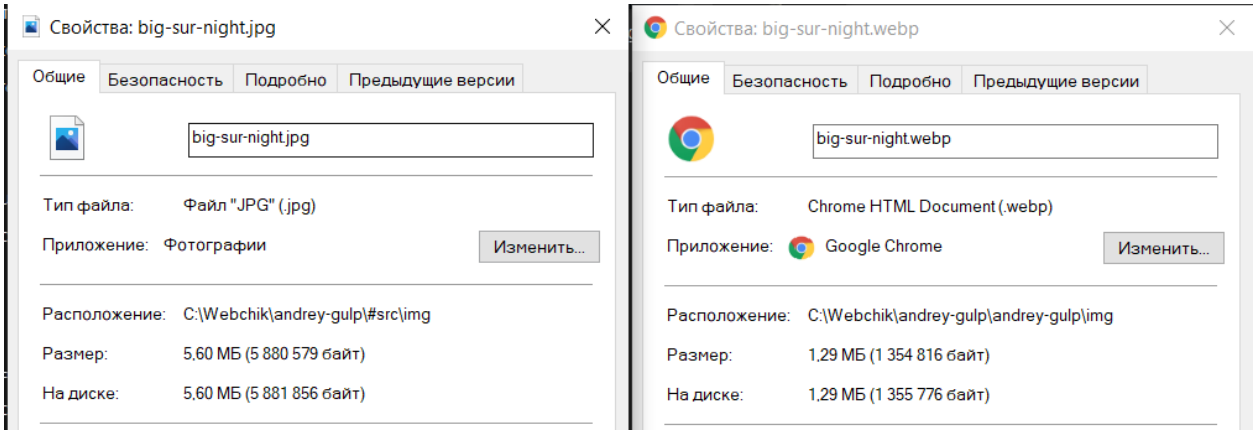

Рисунок 12 – Сравнение размера jpg и webp версий одного и того же изображения

Трансляция ES6+ в ES5 происходит при запуске Gulp. Также работает вложение кода из других js файлов путём использования команды «@@include».

Рисунок 13 – Главный js файл в исходных файлах

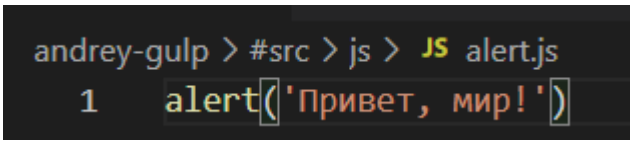

Рисунок 14 – Дополнительный js файл с вызовом сообщения в браузере в исходных файлах

andrey-gulp > andrey-gulp > js > JS script.js > ...

```
"use strict";

alert('Привет, мир!');

function testWebP(callback) {
  var webP = new Image();

  webP.onload = webP.onerror = function () {
    callback(webP.height == 2);
  };

  webP.src = "
  0PP8bA//LwYAAA";
}

testWebP(function (support) {
  if (support == true) {
    document.querySelector('body').classList.add('webp');
  } else {
    document.querySelector('body').classList.add('no-webp');
  }
});
```

Рисунок 15 – Оптимизированный js файл

Как видно на скриншотах, синтаксис изменился и теперь будет поддерживаться даже старыми версиями Internet Explorer.
Файлы scss тоже преобразуются в понятный браузеру css файл. Добавляются вендорные префиксы там, где они нужны, как в случае со свойством «display:flex» Также объединяются несколько media запросов в один, что ускоряет загрузку сайта.

style.scss

andrey-gulp > #src > scss > style.scss > font

```
body {
    font-size: 18px;
    display: flex;
}

span {
    font-size: 20px;
    @media (max-width:767px) {
        font-size: 15px;
    }
}

div {
    @media (max-width:767px) {
        font-size: 20px;
    }
}
```

Рисунок 16 – Исходный scss файл

```
andrey-gulp > andrey-gulp > css > # style.css > ...
32
33    body {
34      font-size: 18px;
35      display: -webkit-box;
36      display: -ms-flexbox;
37      display: flex;
38    }
39
40    span {
41      font-size: 20px;
42    }
43
44
45    @media (max-width: 767px) {
46      span {
47        font-size: 15px;
48      }
49
50      div {
51        font-size: 20px;
52      }
53    }
```

Рисунок 17 – Оптимизированный css файл

Все иконки теперь доступны в одном html файле и могут использоваться отдельно, при этом они будут находиться в отдельном svg файле, который разбивается на секторы и каждый сектор имеет свой html код.

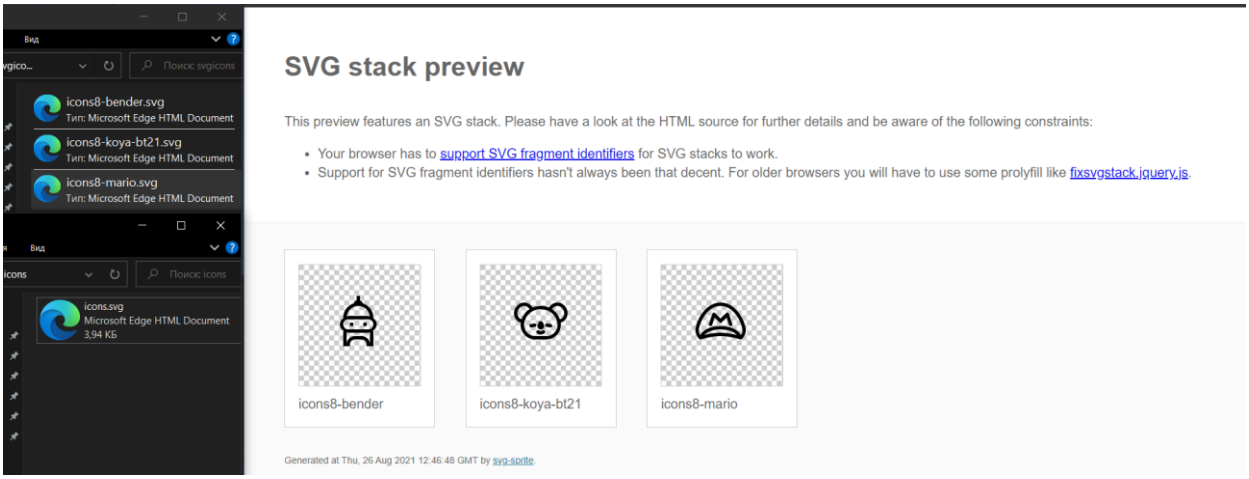

Рисунок 18 – Объединённые иконки из файла icons.svg

Таким образом, была настроена система сборки задач Gulp. Это лишь один из примеров конфигурации, которых может быть всевозможное количество, отталкиваясь от потребностей в разработке и профиля

специалиста, использующего данный сборщик. Данное решение помогает автоматизировать создание оптимизированного сайта, избавляю от использования большого количества сторонних сайтов и приложений для трансляции кода, сжатия изображения и т.д. Также пропадает необходимость изучения сразу нескольких версий языка JavaScript, ведь весь код, написанный по новым стандартам легко транслируется в старую версия языка, что сильно экономит время на изучение JavaScript начинающими разработчиками.

Библиографический список

1. Косников Ю. Н., Мелешкин В. В. Автоматизация создания цветовой модели сайта // Модели, системы, сети в экономике, технике, природе и обществе. 2018. №. 4 (28).
2. Харченко А. В. Автоматизация процесса создания веб-сайтов с элементами скроллинг анимации // Вестник науки и образования. 2017. Т. 1. №. 7 (31).
3. Просветов В. Л., Конева Н. Е. Анализ методов и средств автоматизации процессов обработки данных веб-сайтов // Евразийское Научное Объединение. 2019. №. 1-2. С. 89-94.
4. Отдельный В. А., Евсеенко И. А. Автоматизация тестирования веб-сайтов. М., 2012.
5. Арсирий Е. А. и др. Автоматизация разработки и обновления семантического ядра сайта с динамическим контентом // Штучний інтелект. 2012.
6. Node.JS // URL: https://nodejs.org/en/ (дата обращения: 21.08.2021)
7. Gulp // URL: https://gulpjs.com/ (дата обращения: 21.08.2021)