hostsight
Hosting and Technology HTML Template
32 PAGES
FAST WORDPRESS HOSTING FOR EVERYBODY
SHARED HOSTING THAT SUPPORT YOUR GOAL
CLOUD COMPUTING & OBJECT STORAGE

HostSight

HTML TEMPLATE FOR HOSTING COMPANIES

BY CRUMINA

Table of Contents

Template description

Introduction

Hostsight is an HTML template specially created for IT companies offering hosting services. It contains all the necessary elements and pages for building a hosting website, and it has a fresh, light and unique design.
Hostsight is primarily designed for those looking for a cool, niche design with simple and affordable functionality. If you want to create the perfect website with all the necessary characteristics of a hosting website, then Hostsight is just for you!
The template contains 30+ HTML pages in which you can find not only whole page solutions, but also take any of the components for your web needs. A huge variety of sliders, theme widgets, as well as copyright work is a shortlist of what you get with
this template.

Template structure

SCSS structure

You can find all SCSS files in the **"sass"** folder.
In the folder **"sass/theme-styles"** are the main, global files with styles for the entire template, namely:

/blogs.scss - styles for the blog;

/breadcrumbs.scss - contains styles for breadcrumbs;

/custom-variables.scss - the file contains all global variables and their values, such as the primary colors used in the theme, size variables and font families, etc.

/forms.scss - contains styles for forms, as well as for all form components (input, textarea, select, radio-button, checkboxes ...);

/global.scss - contains standard styles for the main web components, such as "body", "a", "p", "table", "ol", "ul", "form", "input", etc .;

/helper.scss - this file contains all the helper classes and styles for them, which were used to build the template;

/mixins.scss - mixins;

/preloader.scss - styles for the preloader;

/root.scss - css variables;

/stunning-header.scss - styles for stunning-header;

/typography.scss - contains styles responsible for the typography of the template;

The **"sass/layouts"** folder contains files with styles of individual template layouts, namely:

/footer.scss - contains styles for the footer, as well as for all components
related to it;

/free-ssl-section.scss - contains styles for free-ssl-section;

/header.scss - contains styles for header, animation styles for
stick-header, as well as for all components associated with it;

/status-pages.scss - contains styles for status-pages.

In the folders **"sass/blocks"** and **"sass/widgets"** you will find styles for all blocks and widgets in the template.

In the folder **"sass/plugins"** there are styles of plugins used in the template.

In the **"vendors"** folder are Bootstrap styles v4.1.3.

For your convenience and ease of use, all style files are separated almost block-wise. You can edit the necessary styles in them, as well as change responsive settings for the block you need.

The detailed structure of these folders is described in the file **"sass/table-of-content.scss"**.

The following files are located in the root of the SASS folder:

main.scss - it contains all the paths for the proper compilation of all SCSS
files.

rtl.scss - styles for the RTL version of the template.

theme-font.scss - template fonts connection.

CSS files structure

CSS files, that load in html <head> section of page in the current template are the following:

<! - Theme Styles CSS ->

<link rel = "stylesheet" type = "text / css" href = "css / vendors / Bootstrap / bootstrap.min.css">

<link rel = "stylesheet" type = "text / css" href = "css / vendors / Bootstrap / bootstrap-grid.min.css">

<link href = "css / plugins / navigation.min.css" rel = "stylesheet">

<link rel = "stylesheet" type = "text / css" href = "css / main.min.css">

```
<link rel = "stylesheet" type = "text / css" href = "css / theme-font.min.css">
```

bootstrap.min.css - bootstrap styles

bootstrap-grid.min.css - bootstrap modular grid styles

navigation.min.css - styles of the main menu (navigation)

main.min.css - main site's stylesheet;

theme-font.min.css - font styles of the template.

IMPORTANT: if necessary, in the HTML/css folder you can find the non-minified style files and use them.

Javascript files structure

Javascript files, that load right before closing html <body> tag of page in this template are the following:

```
<script src="js/jquery.min.js"></script>

<script src="js/Bootstrap/bootstrap.bundle.min.js"></script>

<script src="js/js-plugins/navigation.min.js"></script>

<script src="js/js-plugins/select2.min.js"></script>

<script src="js/js-plugins/material.min.js"></script>

<script src="js/js-plugins/swiper.min.js"></script>

<script src="js/js-plugins/jquery-countTo.min.js"></script>

<script src="js/js-plugins/waypoints.min.js"></script>

<script src="js/js-plugins/leaflet.min.js"></script>

<script src="js/js-plugins/MarkerClusterGroup.min.js"></script>
```

```
<script src="js/js-plugins/jquery.magnific-popup.min.js"></script>
<script src="js/js-plugins/TimeCircles.min.js"></script>
<script src="js/js-plugins/smooth-scroll.min.js"></script>
<script src="js/js-plugins/jquery.matchHeight.min.js"></script>
<script src="js/js-plugins/imagesLoaded.min.js"></script>
<script src="js/js-plugins/isotope.pkgd.min.js"></script>
<script src="js/js-plugins/ajax-pagination.min.js"></script>
<script src="js/js-plugins/Chart.min.js"></script>
<script src="js/js-plugins/chartjs-plugin-deferred.min.js"></script>
<script src="js/js-plugins/modernizr-custom.min.js"></script>
<script type="text/javascript">
  if ('loading' in HTMLImageElement.prototype) {
    const images = document.querySelectorAll('img[loading="lazy"]');
    images.forEach(img => {
      img.src = img.dataset.src;
  });
  } else {
    // Dynamically import the LazySizes library
    const script = document.createElement('script');
    script.src =
      'https://cdnjs.cloudflare.com/ajax/libs/lazysizes/4.1.8/lazysizes.min.js';
    document.body.appendChild(script);
  }
  Modernizr.on('webp', function (result) {
      if (result) {
        // supported
      } else {
        // not-supported
      }
  });
</script>
<script  src="js/main.js"></script>
<script src="js/js-plugins/leaflet-init.js"></script>


<!-- jQuery-scripts for Modules (Send Message) -->
<script src="modules/forms/src/js/jquery.validate.min.js"></script>
<script src="modules/forms/src/js/sweetalert2.all.js"></script>
<script src="modules/forms/src/js/scripts.js"></script>
```

jquery.min.js - the latest version of jQuery library connected

bootstrap.bundle.min.js - scripts for working with the Bootstrap framework elements

select2.min.js - script for selects

material.min.js - script for animations and building form elements

swiper.min.js - Most Modern Mobile Touch Slider

jquery-countTo.min.js - script for counters

waypoints.min.js - Waypoints is a library that makes it easy to execute a function whenever you scroll to an element

leaflet.min.js - Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps.

MarkerClusterGroup.min.js - Marker Clustering plugin for Leaflet

jquery.magnific-popup.min.js - script to open photos and videos in a pop-up window

TimeCircles.min.js - jQuery plugin to show time since or time until a specific time

smooth-scroll.min.js - A lightweight script to animate scrolling to anchor links

jquery.matchHeight.min.js - script to set the equal height for elements in the same row

imagesLoaded.min.js - script for proper work of isotope

isotope.pkgd.min.js - sorting scripts

ajax-pagination.min.js - script to add content to the page by click

Chart.js - Simple HTML5 Charts using the tag

chartjs-plugin-deferred.min.js - Chart.js plugin to defer initial chart updates

modernizr-custom.min.js - adds the “webp” class for the <html> tag if the used browser supports the “* .webp” image format, or the “no-webp” class, if it doesn't.

leaflet-init.js - leaflet map initialization script

main.js - main custom javascript file of the template

jquery.validate.min.js, sweetalert2.all.js, scripts.js - scripts for proper work of the contact form,
as well as to generate messages to the user.

IMPORTANT:
1. If necessary, in the HTML/js/js-plugins-uncompressed folder you can find the non-minified js files and use them.
2. On all HTML-pages all js-scripts that are used in the template are connected, but at the

same time, to ensure an increase the “speed test” indicators, scripts that are not used on the current page are commented out (disabled).

For example, on a page where there are no elements with counters, the scripts responsible for their work are commented out:

```
<!--<script src="js/js-plugins/jquery-countTo.min.js"></script>-->

        <!--<script src="js/js-plugins/waypoints.min.js"></script>-->
```

Therefore, if during development, you need to use an element (for example, a map) on some of the pages which it was not used previously on, you only need to add the map code to the HTML markup of the page and uncomment the scripts responsible for its operation and initialization, namely:

```
<script src="js/js-plugins/leaflet.min.js"></script>

<script src="js/js-plugins/MarkerClusterGroup.min.js"></script>

<script src="js/js-plugins/leaflet-init.js"></script>
```

Work with GULP

Gulp introduction

Gulp is a front-end development automation tool. It will help you automate routine tasks and speed up your work.

The main tasks that Gulp helps you solve:

- Minimization and concatenation of Javascript and CSS files;
- HTML minimization;
- Using CSS preprocessors Sass, Less, Stylus and others;
- Image optimization;
- Automatic installation of vendor prefixes;
- Use of template engines;
- Code validation and testing;

Useful links:
Gulp official website https://gulpjs.com/
Gulp at https://github.com/gulpjs/gulp
Gulp plugin repository http://gulpjs.com/plugins/

Gulp Documentation https://github.com/gulpjs/gulp/blob/master/docs/README.md#articles

In the next articles, we provide you with brief instructions on getting started with the project builder GULP.

Gulp installation and setup

Step 1. Installing Node.js

To install Node, visit nodejs.org website and then click on the green "Install" button. When the download is complete, launch the application and install Node.

Step 2. Installing NPM

At the command line, go to the main folder of your project: **cd/html-hostsite**

Note, that the path to the project for everyone can be individual, depending on its location on the local PC.

If you created the project from scratch, you would need to run the command **"npm init"** at the command line. It creates a package.json file that contains information about the project (description of the project and dependencies). But, since Hostsight already contains this file, skip this step.

Step 3. Installing Gulp.js

Gulp installation is done with the command **"npm install -g gulp"**

We want to immediately clarify a few details:

npm is the application with help of which we install the package (plugin).

install - you tell the command line that you need to install the package.

-g is a kind of flag indicating that we want to install the package globally.

gulp is the name of the package we want to install.

After you have installed Gulp.js, make sure that everything is correct, enter the old command by analogy with Node and npm "gulp -v". It should display version of the installed gulp.

With the following command, we install Gulp locally in the project folder - **"npm install --save-dev gulp**"
The only difference is that we indicate in our "flag" that we write dependencies in our package.json.

Step 4. Setting up and running Gulp.js

TO make Gulp work, you need to explain to it what it should do.

There are some methods: task, launch and file path. All tasks, their settings, and configurations are registered in the file **../html-hostsite/gulpfile.js.**

Let's analyze our file in more detail:

```
var gulp = require('gulp'),
plumber = require('gulp-plumber'), // generates an error message

prefixer = require('gulp-autoprefixer'), // automatically prefixes to css properties

sass = require('gulp-sass'), // for compiling scss-files to css

browserSync = require('browser-sync').create(), // for online synchronization with the
browser

imagemin = require('gulp-imagemin'), // for minimizing images-files

webp = require('gulp-webp'), // for convert png in webp

cache = require('gulp-cache'), // connecting the cache library

htmlhint = require("gulp-htmlhint"), // for HTML-validation

runSequence = require('run-sequence'), // for sequential execution of Gulp-tasks

svgmin = require('gulp-svgmin'); // for minimizing svg-files
```

Using these commands, we call Gulp and plugins that perform our tasks.

Note, that to use a particular plugin, it needs to be installed first, similar to installing Gulp itself.

After installing plugins, you can begin to configure them in the gulpfile.js file.

We will not describe all Gulp plugins and their features, as there is a great variety of them. But the plugins used in our template will be described in detail. So, let's start!

plumber = require ('gulp-plumber'), // generates an error message

Installation: **install --save-dev gulp-plumber**
In a nutshell, this plugin makes development easier by the fact that when
errors occur, it displays information about them in the console and does not interrupt the
execution of the task flow.
Documentation - https://www.npmjs.com/package/gulp-plumber

prefixer = require ('gulp-autoprefixer'), // automatically prefixes to css properties

Installation: **npm install --save-dev gulp-autoprefixer**
It automatically appends vendor prefixes (-webkit, -ms, -o, -moz, etc.) to css styles. These prefixes provide browser support for CSS3 properties. This plugin is used in our "sass" task:

```
gulp.task('sass', function() {
  gulp.src('./HTML/sass/**/*.scss')
  .pipe(plumber())
  .pipe(sass())
  .pipe(prefixer(
    {
      browsers: ['last 12 versions'],
      cascade: false
    }
  ))
  .pipe(gulp.dest('./HTML/css'))
  .pipe(browserSync.stream());
});
```

Documentation - https://www.npmjs.com/package/gulp-autoprefixer

sass = require ('gulp-sass'), // for compiling scss-files to css

Installation: **npm install --save-dev gulp-sass**
It serves to compile scss files into css. It is used in the **"sass"** task:

```
...
.pipe(sass())
...
```

The result is recorded in **.pipe (gulp.dest ('./ HTML / css'))**

Documentation - https://www.npmjs.com/package/gulp-sass

browserSync = require ('browser-sync'), // for online synchronization with the browser

Installation: **npm install --save-dev browser-sync**
BrowserSync is probably one of the most useful plugins a developer would ever want to have.

It actually gives you the opportunity to start the server, on which you can run your applications.

```
...
.pipe(browserSync.reload({
   stream: true
}))
...
```

Now imagine for a second, that you have GULP running, which automatically compiles, for example, scss, and browserSync is used in the same task. That is, after scss compilation is completed, the browser automatically reloads the page and pulls up all entered and compiled changes in css accordingly. Convenient, isn't it?

Documentation - https://www.npmjs.com/package/browser-sync

imagemin = require ('gulp-imagemin'), // for minimizing images-files

Installation: **npm install --save-dev gulp-imagemin**
This plugin is designed for working with graphics, it optimizes images and
The result is recorded by the specified path. It is used in the "images" task:

```
/*========== Minimization IMAGE =============*/
gulp.task('images', function () {
  gulp.src('./HTML/img/**/*.*')
  .pipe(cache(imagemin({
    interlaced: true
  })))
  .pipe(gulp.dest('./HTML/img'));
});
```

Documentation - https://www.npmjs.com/package/gulp-imagemin

webp = require('gulp-webp'), // for convert png in webp

Installation: npm install --save-dev gulp-webp

Task for converting images from PNG format into WEBP format.

Documentation - https://www.npmjs.com/package/gulp-webp

```
gulp.task('build', function(done) {

 runSequence('sass', 'html-valid', 'svg-min', 'images', 'convert-webp', done);

});
```

cache = require ('gulp-cache'), // connecting the cache library

Installation: **npm install --save-dev gulp-cache**
As the name implies, it is used to cache data. It is used specifically in the "images" task.
A large number of images will be processed much longer, therefore, it would be nice to add cache to image processing, to save our time.

Documentation - https://www.npmjs.com/package/gulp-cache

htmlhint = require ("gulp-htmlhint"), // for HTML-validation

Installation: **npm install --save-dev gulp-htmlhint**
A simple HTML validator that is used in the **"html-valid"** task:

```
/*============ HTML-validator =============*/
gulp.task('html-valid', function () {
  gulp.src("./HTML/*.html")
  .pipe(htmlhint());
});
```

Documentation - https://www.npmjs.com/package/gulp-htmlhint

runSequence = require ('run-sequence'); // for sequential execution of Gulp-tasks

Installation: **npm install --save-dev run-sequence**
By default, when Gulp starts, tasks are executed asynchronously, which sometimes can lead to very undesirable and illogical actions. It is for to avoid such situations, we use the run-sequence plugin, which helps establish dependencies on tasks.

Documentation - https://www.npmjs.com/package/run-sequence

svgmin = require('gulp-svgmin'); // for minimizing svg-files

Installation: **npm install --save-dev gulp-svgmin**
This is a plugin for SVG icons optimization.

```
/*=========== Minimization SVG =============*/
gulp.task('svg-min', function () {
   gulp.src('./HTML/svg-icons/*.svg')
      .pipe(svgmin({
         plugins: [{
          removeDoctype: true
        }, {
          removeComments: true
        }, {
          cleanupNumericValues: {
            floatPrecision: 2
          }
        }, {
          convertColors: {
            names2hex: true,
            rgb2hex: true
          }
        }]
      }))
      .pipe(gulp.dest('./HTML/svg-icons/*.svg'));
});
```

Documentation - https://www.npmjs.com/package/gulp-svgmin

Now a few words about launching tasks, as well as about combining them.
To start any task you need to write on the command line: gulp 'task name' and run it.

We also have 2 sets of joint tasks:

```
/*============ Join tasks =============*/
gulp.task('default', function (callback) {
  runSequence(['sass', 'watch'],
    callback
  )
});

gulp.task('build', function(done) {
  runSequence('sass', 'html-valid', 'svg-min', 'images', done);
});
```

'default' - used mainly during the development process, includes ['sass', 'watch']
'build' - a complete assembly of the template, which includes tasks ('sass', 'html-valid', 'svg-min', 'images').

Gulp official documentation:
https://gulpjs.com/
https://gulpjs.org/

Main functions, components and their usage

RTL version

One of the main features of Hostsight is the availability of the RTL version.
For its activation you need to find and uncomment the following code in the page layout:

```
<!--Styles for RTL-->
```

```
<!--<link rel="stylesheet" type="text/css" href="css/rtl.css">-->
```

to get it look like this:

```
<!--Styles for RTL-->
<link rel="stylesheet" type="text/css" href="css/rtl.css">
```

Headers

The template provides the sticky header function (stuck to the top edge of the browser window). Sticky header markup:

```
<nav id = "navigation" class = "site-header navigation navigation-justified header - sticky"> ... </header>
```

where **id="navigation"** is the unique identifier of the header;
class="site-header" - a common class for all header types;
class="header-sticky" - a unique class that is responsible for the type of header.

To activate/deactivate the sticky header function, just add remove class **".header — sticky"** in header markup.

Footer parallax

Footer Parallax function is implemented for demonstration purpose only on the main page of the template - index.html.

To deactivate/activate it, you must remove/add class "js-fixed-footer" in the footer markup:

```
<footer id="site-footer" class="footer footer--dark footer--with-decoration js-fixed-footer">...</footer>
```

Preloader

The template also implements the preloader function for loading pages. By default it is deactivated.

To activate it, it is necessary to find the next code in the content of each page:

```
<!--<div id="hellopreloader">....</div>-->
```

and uncomment it.

If you don’t need the preloader function, just remove the markup above.

Load More button

To add content to the page when you click on the load more button, we use library **<script src = "js / js-plugins / ajax-pagination.js"> </script>**

The button itself has the following markup:

```
<a href="#" class="crumina-button button--dark button--xl load-more-button" data-load-link="blogs-to-load.html" data-container="blogs-items"> LOAD MORE POSTS </a>
```

class = ". load-more-button" is required to initialize it

data-load-link = "blogs-to-load.html" is data attribute indicating, where to get the added content from

data-container = "blogs-items" is a data attribute indicating the container id, where we add content.

Popup windows

Pop-up windows are fully implemented using Bootstrap. Complete instructions on their description and settings are presented in the official documentation - https://getbootstrap.com/docs/4.1/components/modal/

Leaflet maps

To add a map to a page, do the following:

- Add the following markup on your page

```
<section>

<div class="crumina-module crumina-map height-430" id="map"></div>

</section>
```

- make sure that the scripts responsible for the proper work of maps are connected on the current page:

```
<script src="js/js-plugins/leaflet.min.js"></script>

<script src="js/js-plugins/MarkerClusterGroup.min.js"></script>

<script src="js/js-plugins/leaflet-init.js"></script>
```

In the HTML/js/js-plugins/leaflet-init.js file you will find all the map settings, markers, coordinates for each
of the specified id (function - CRUMINA.maps)

Video section

Blocks containing video content have the main parent class **“crumina-our-video”**. To start the video, one uses the button (example):

```
<a href="https://www.youtube.com/watch?v=wnJ6LuUFpMo" class="video-control video-control-standard js-popup-iframe">
  <img src="img/theme-content/images/play-video.jpg" alt="play">
</a>
```

where **"js-popup-iframe"** is the class responsible for opening a video in a pop-up window.
In the **href=""** attribute we record a link to the video itself.

Contact form

To organize feedback, the template has the ability to send messages from the contact form, which is presented on page 17_contacts.html.
Let's review the organization of its functionality.
Form layout:

```
<form class="send-message-form crumina-submit" method="post" data-nonce="crumina-submit-form-nonce" data-type="standard" action="modules/forms/submit.php">...</form>
```

where

- **".crumina-submit"** - a required class for initializing scripts for sending messages;
- **data-nonce = "crumina-submit-form-nonce"** - the data attribute that is responsible for checking the data entered in the form fields and generating a message about their correctness or incorrectness;
- **data-type = "standard"** - data attribute, which can take 2 values: **"standard"** - to send messages to the specified mailboxes, which is indicated in the file ..html-hostsite \ HTML \ modules \ forms \ inc \ config.php "mailchimp", '' 'standard' => array ('email' => 'lorem@ipsum.dolor', ...) ' '' , and **"mailchimp"** - for proper interaction with MailChimp;
- **action = "modules / forms / submit.php"** - contains the path to the handler file.

Form fields layout:

```
<input class="input--grey input--squared" name="name" type="text" placeholder="Your Name" required>
<input class="input--grey input--squared" name="email" type="email" placeholder="Email Address" required>
<textarea class="input--grey input--squared" name="message" placeholder="How can we help you?" rows="4" required></textarea>
```

For the form to work correctly, each of the fields must necessarily contain the corresponding name, namely:

name = "name" - for the user name input field;

name = "email" - for the input field of the user's email address;

name = "message" - for the message text field.

Note, that for the correct operation of the form, as well as for generating messages, the user must call js scripts on page:

```
<!-- jQuery-scripts for Modules (Send Message) -->
<script src="modules/forms/src/js/jquery.validate.min.js"></script>
<script src="modules/forms/src/js/sweetalert2.all.js"></script>
<script src="modules/forms/src/js/scripts.js"></script>
```

SVG Icons

All svg-icons used in the theme are located in "HTML/svg-icons" folder.
In the folder "HTML/svg-icons/sprite there is the icons.svg file. This is svg-sprite, which contains a set of svg-icons, each of which has its own unique ID. To call and place any icon from the sprite in the content, you should use markup (for example):

```
<svg class="crumina-icon"><use xlink:href="svg-icons/sprite/icons.svg#icon-close"></use></svg>
```

where:

class = "crumina-icon” is a required class that is used for all theme icons;

href="svg-icons/sprite/icons.svg#icon-close is the so-called "link" to the unique ID of the required icon from the sprite.

For simplicity and clarity, all the icons used in the template with their usage code are placed in the typography page - **31_typography.html**

Lazy Load

To improve the page loading speed, the template uses Lazy Load for images. The script is:

```
if ('loading' in HTMLImageElement.prototype) {

  const images = document.querySelectorAll('img[loading="lazy"]');

  images.forEach(img => {

    img.src = img.dataset.src;

});

} else {

  // Dynamically import the LazySizes library

  const script = document.createElement('script');
  script.src =

    'https://cdnjs.cloudflare.com/ajax/libs/lazysizes/4.1.8/lazysizes.min.js';

  document.body.appendChild(script);
```

The markup of the images is as follows:

```
<picture>

  <source type="image/webp" srcset="data:image/gif;base64,R0lGODlhAQA-BAAAAACH5BAEKAAEALAAAAAABAAEAAAICTAEAOw==" data-srcset="img/demo-content/logo-white.webp">

  <img class="lazyload" loading="lazy" src="data:image/gif;base64,R0lGODlhAQA-BAAAAACH5BAEKAAEALAAAAAABAAEAAAICTAEAOw==" data-src="img/demo-content/logo-white.png" alt="logo">

</picture>
```

where:

- **<picture>** - tag wrapper for the image and its alternative formats
- the **<source>** tag contains in its data attribute **“data-srcset”** the path to the image in the “webp” format (for those browsers that support this format)
- the **<img>** tag itself contains class = "lazyload" responsible for initializing LazyLoad, the data attribute **loading = "lazy"** is used to initialize the standard LazyLoad in browsers that support this functionality, the **"data-src"** data attribute contains the path to the image in the format of "png" or "jpg", for browsers that do not support the format of "webp".

It should be noted that in the style files there is possibility of using modern formats as back-grounds, as well as standard formats, for example:

```
.no-webp .popup-search .modal-content::after {

  background-image: url("../img/theme-content/backgrounds/header-footer-gradient-bg.png");

}



.webp .popup-search .modal-content::after {

  background-image: url(../img/theme-content/backgrounds/header-footer-gradient-bg.webp);
}
```

Here:
 - **“no-webp”** and **“webp”** are the classes that are added to the <html> tag automatically when the page loads, using a js script.
“Webp” - if the used browser supports the “* .webp” image format, or the “no-webp” class if it does not.
Therefore, for some browsers, some styles will be loaded, for others - others, depending on their functionality.