Getting started

Template description

Olympus HTML Social Network Toolkit was developed using the most popular streaming build system called Gulp. It made Olympus responsive, fast and easy to use and edit. Olympus HTML Toolkit is based on one of the most popular frameworks - Bootstrap v4.0.0-alpha.6. SCSS/CSS styles in the template are based on SASS technology, that let us make the template well-structured and user-friendly. Due to SASS, users are able to change main template colors only in only one click by changing styles for global variables. Accurate code structure with table of contents, comments and block structure throughout the entire template is another valuable advantage. All necessary information about it and even more is located in this documentation, that comes bundled with the template. All this makes Olympus easy customizable, clean, with clear options and simple ways to change them. Olympus includes 40+ HTML pages, where you can find ready solutions for the entire page or separate components suitable for your project. The template comes with SVG icons, great variety of network components, thematic widgets, popup windows and many other things - this is a short list of the features you really get with this template. Our preview demo will tell you everything about Olympus and the solution will come along by itself!

Working with GULP

As already mentioned above, the most important changes have touched on the principles of the project construction and methods of its assembly. It's time to talk about Gulp.

A gulp is a tool for automating front-end development. It will help you automate routine tasks and speed up your work.

The main tasks that Gulp will help you to solve:

- Minimize and concatenate Javascript and CSS files;
- HTML minimization;
- Using CSS preprocessors - Sass, Less, Stylus, and others;
- Images optimization;
- Automatic installation of vendor prefixes;
- Using template engines;
- Code validation and testing.

Useful links:

Gulp official site - gulpjs.com

Gulp at Github https://github.com/gulpjs/gulp

Gulp plugins catalog http://gulpjs.com/plugins/

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

So, we give you a brief instruction on how to get started with the templates assembler - GULP.

Step1. Node.js installation

To install Node, visit the nodejs.org site, click the green "Install" button. When the download is complete, start the application and install Node.

Step 2. NPM installation

At the command prompt, go to the main folder of your project: **.../Olympus HTML**.

If you created the project from scratch, you would have to run the command: npm init

It creates a package.json file that contains information about the project (project description and dependencies). But, since Olympus already contains this file, this step should be skipped.

Step 3. Gulp.js installation

Installation of Gulp is done with the command: **npm install -g gulp**

I want to clarify a few details right away:

- **npm** - is the application by which we install the package (plugin).

- **install** - we tell the command line 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 we installed Gulp.js, let's make sure that everything is correct and enter the old command by analogy with Node and npm: **gulp -v**

The next command is to install Gulp locally in the project folder: **npm install --save-dev gulp**

The only difference is that we specify in our "flag" that we record dependencies in our package.json.

Step 4. Gulp.js setup and launch

To make Gulp work, you need to explain to him what it should do. There are several methods: task, start, and path to files. All tasks, their settings, and configurations are recorded in the**../Olympus HTML/gulpfile.js** file.

Let's analyze our file in more detail.

```
/*=========== GULP + Plugins init ==============*/

var gulp = require('gulp'),
    plumber = require('gulp-plumber'), // generates an error message
    prefixer = require('gulp-autoprefixer'), // automatically prefixes to css properties
    uglify = require('gulp-uglify-es').default, // for minimizing js-files
    cssmin = require('gulp-cssmin'), // for minimizing css-files
    svgmin = require('gulp-svgmin'), // for minimizing svg-files
    rename = require('gulp-rename'), // to rename files
    sass = require('gulp-sass'), // for compiling scss-files to css
    webserver = require('browser-sync'), // for online synchronization with the browser
    imagemin = require('gulp-imagemin'), // for minimizing images-files
    cache = require('gulp-cache'), // connecting the cache library
    concat = require('gulp-concat'),
    zip = require('gulp-zip'),
    webp = require('gulp-webp'), // for convert png in webp
    del = require('del'),
    replace = require('gulp-replace'),
    sourcemaps = require('gulp-sourcemaps'),
    babel = require('gulp-babel'),
    htmlhint = require("gulp-htmlhint"); // for HTML-validation
```

With the help of these commands, we enable Gulp and plugins that perform our tasks.

It's important to note, that each plugin needs to be installed by analogy with Gulp installation before one can use it. Once they are installed, you can begin to set them up in the very **gulpfile.js** file.

We won't review all the Gulp plugins and their possibilities, since there are a great many of them. But we will review the plugins used in Olympus in detail. So:

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

Installation: **install --save-dev gulp-plumber**

Description: This plugin makes development easier so, that when there occurs an error, it outputs information about it in the console and doesn't interrupt the implementation of tasks flow.

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

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

Installation: **npm install --save-dev gulp-autoprefixer**

Description: The plugin automatically complements CSS styles with vendor prefixes (-webkit, -ms, -o, -moz, etc). These prefixes ensure the support of CSS3 properties by browsers. This plugin is used in our 'sass' task:

```
/*============ Compile SCSS ==============*/

gulp.task('sass', function(cb) {

    gulp.src('html/sass/*.scss')
        .pipe(sourcemaps.init())
        .pipe(plumber())
        .pipe(sass(
            {
                linefeed: "crlf"
            }
        ))
        .pipe(prefixer())
        .pipe(gulp.dest('./html/css'))
        .pipe(rename({
            suffix: '.min'
        }))
        .pipe(cssmin())
        .pipe(sourcemaps.write('./maps'))
        .pipe(gulp.dest('./html/css'));

    gulp.src('html/Bootstrap/scss/*.scss')
        .pipe(plumber())
        .pipe(sass(
            {
                linefeed: "crlf"
            }
        ))
        .pipe(cssmin())
        .pipe(sourcemaps.write('./maps'))
        .pipe(gulp.dest('./html/Bootstrap/dist/css'))
        .pipe(webserver.reload({stream: true}));
        cb();
});
```

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

3) uglify = require('gulp-uglify'), // for minimizing js-files

Installation: **npm install --save-dev gulp-uglify**

Description: This plugin serves for JS file minification. It deletes unwanted gaps, commas, semicolons. As a result, the JS file is smaller in size. This plugin is used in our 'js' task:

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

4) cssmin = require('gulp-cssmin'), // for minimizing css-files

Installation: **npm install --save-dev gulp-cssmin**

Description: The plugin minifies CSS files with help of Gulp. It is used in our 'sass' task:

...

.pipe(cssmin())

...

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

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

Installation: **npm install --save-dev gulp-svgmin**

Description: The plugin minimizes SVG files with help of Gulp. It is used in our 'svg-min' task:

```
/*========== Minimization SVG ==========*/

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

    cb();
});
```

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

6) rename = require('gulp-rename'), // to rename files

Installation: **npm install --save-dev gulp-rename**

Description: The plugin enables rename files. It is used in the "sass" task. To rename the file, you need to add ".min" after minification

.pipe(rename({

  suffix: '.min'

}))

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

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

Installation: **npm install --save-dev gulp-sass**

Description: The plugin serves to compilation SCSS files into CSS. It is used in the 'sass' task:

...

.pipe(sass(

```
  {
    linefeed: "crlf"
  }
))
...
```

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

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

Installation: **npm install --save-dev browser-sync**

Description: BrowserSync is probably one of the most useful plugins, that a developer would like to have. It actually gives you the possibility to start a server, on which you can run your applications.

```
...
.pipe(webserver.reload({stream: true}));
…
```

And now imagine for a second that you have Gulp running, which automatically compiles, for example, scss, and in the same task uses browserSync. That is, after compiling scss, the browser automatically reboots the page and pulls up all compiled changes to CSS accordingly. Convenient, isn't it?

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

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

Installation: **npm install --save-dev gulp-imagemin**

Description: This plugin is intended for working with graphics, it optimizes images and records the result in the specified path. It is used in the 'images' task:

```
/*=========== Minimization IMAGE ==============*/

gulp.task('images', function(cb){
    gulp.src('html/img/*')
        .pipe(cache(imagemin({
            interlaced: true
        })))
        .pipe(gulp.dest('html/img'));

    gulp.src('html/screenshots/*')
        .pipe(cache(imagemin({
            interlaced: true
        })))
        .pipe(gulp.dest('html/screenshots'));

    cb();
});
```

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

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

Installation: **npm install --save-dev gulp-htmlhint**

Description: This is a simple HTML validator that we used in the 'html-valid' task:

```
/*============= HTML-validator ==============*/

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

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

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

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

Description: It converts images from the “.jpg”, “.png” formats to “.webp”

```
/*=========== Convert PNG in WEBP ==============*/

gulp.task('convert-webp', function (cb) {
    gulp.src('html/img/**/*.png')
        .pipe(webp({quality: 80}))
        .pipe(gulp.dest('html/img'));
    gulp.src('html/img/**/*.jpg')
        .pipe(webp({quality: 80}))
        .pipe(gulp.dest('html/img'));
    gulp.src('html/screenshots/**/*.png')
        .pipe(webp({quality: 80}))
        .pipe(gulp.dest('html/screenshots'));
    gulp.src('html/screenshots/**/*.jpg')
        .pipe(webp({quality: 80}))
        .pipe(gulp.dest('html/screenshots'));
    cb();
});
```

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

12) babel = require('gulp-babel')

Installation: **npm install --save-dev gulp-babel**

Description: It uses next-generation JavaScript, today, with Babel.

```
/*=========== Babel for svg-loader ==============*/

gulp.task('babel-js', function (cb) {
    gulp.src('./html/js/svg-loader.js')
        .pipe(babel({
            presets: ['@babel/env']
        }))
        .pipe(gulp.dest('./html/js/'));
    cb();
});
```

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

13)

Description: By default, when you run Gulp, the tasks in it are executed asynchronously, which can sometimes lead to very undesirable and not logical actions ... Therefore, some tasks we will launch together, but some of them step-by-step. You can see it in the united tasks:

```
/*============= Join tasks ==============*/

gulp.task('default', gulp.parallel('webserver', 'watch', 'sass'));

gulp.task('build', gulp.series('html-valid', 'sass', 'svg-min', 'convert-webp', 'images', 'compress'));
```

And now a few words about the launch of tasks and combining them. To run 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', gulp.parallel('webserver', 'watch', 'sass'));

gulp.task('build', gulp.series('clean-app', 'html', 'html-valid', 'sass', 'js', 'svg-min', 'images', 'compress', 'copy-files'));
```

- **'default' is used in the main development including** ['webserver', 'watch', 'sass']
- **'build' is a full assembly of the project, that includes tasks** ('clean-app', 'html', 'html-valid', 'sass', 'js', 'svg-min', 'images', 'compress', 'copy-files'), **but before executing it, it is necessary to clean the target folder ../app, with help of the ['clean-app'] task.**

Official Gulp documentation:

https://gulpjs.com/

https://gulpjs.org/

Template installation

After unzipping the Olympus download pack, you'll find a **'app'** folder with all the files.

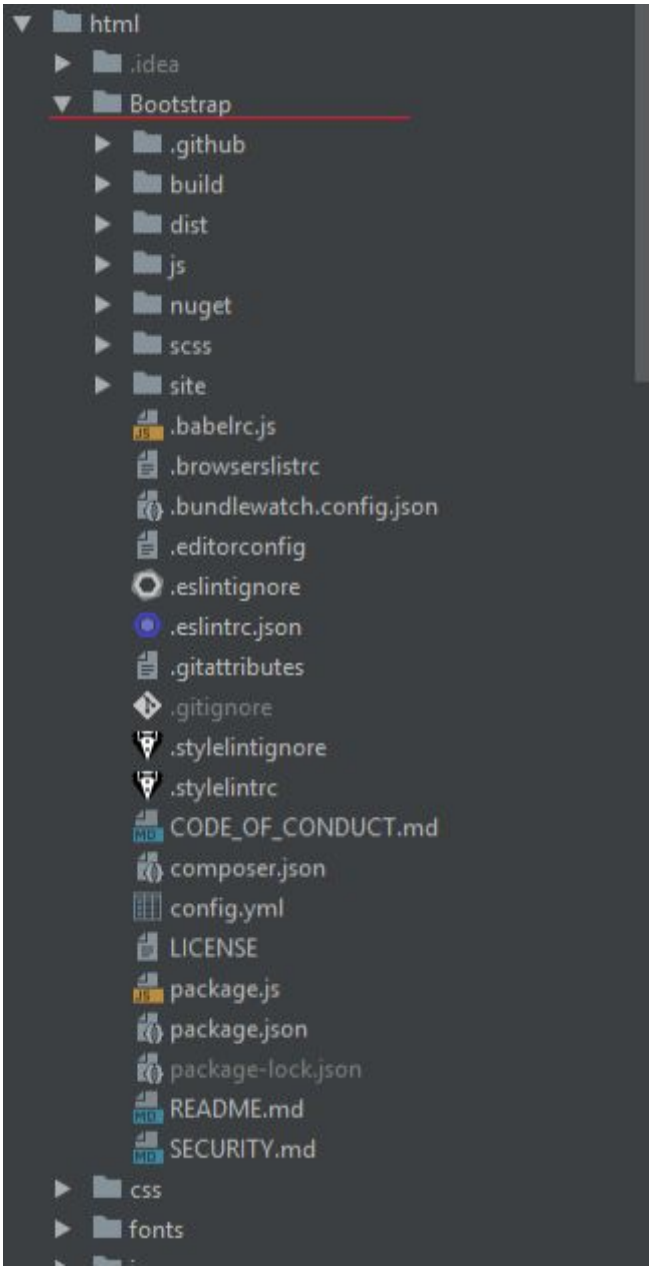
You can view this template in any browser. You can also view or edit the template without an internet connection.

In order to install template on your server open your FTP Client (like Filezilla) or use you cpanel directly, and upload the content of the **'app'** folder on your server root.

Once you are ready, go to **www.yourdomainname.com/index.html**

Bootstrap

Special attention is paid to the **Bootstrap** folder. Its detailed structure, file and folder description, and usage instructions are represented on the official framework website by following this link.



Changes in an update from 12 May 2017

There were made no changes in the Bootstrap folder in the update. It has its original structure and looks. But there are 2 exceptions:

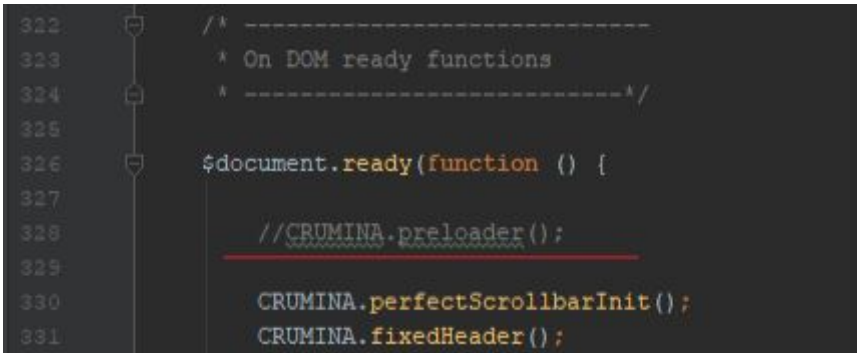
- **Bootstrap\scss_custom.scss** file. It includes changes for variables that were required by the template design.

- **app\sass\theme-styles_bootstrap-customization.scss** file. It includes changes in stylesheet for standard form elements, headings, paragraphs, lists etc.

Components and their usage

Preloader

"Preloader" functionality is implemented in the template, by default this option is disabled. To activate it, you need to find the initialization of the function CRUMINA.preloader(); inside the file **\html\js\main.js** and uncomment it:



Video background for a row/block/section

In order to implement video background in a block, row, or section, the following steps must be performed:

1. Add the following code into HTML markup of the required block:

<video autoplay loop muted class="video-background">

<source src="http://techslides.com/demos/sample-videos/small.mp4" type="video/mp4">

</video>

2. Block, row or section, in which we need to apply video background, must be positioned as follows: **position: relative;** or **position: sticky;** or **position: absolute;**

3. Pay attention that the "video" element must have **class="video-background"** in its markup, a**utoplay** – must be added for autoplay function accordingly.

Header and Footer

Standard header and footer sections with color variations are represented on the following pages:

99_headerAndFooter.html

100_HeaderResponsive.html

Header

Standard header is represented in 2 color variations:

- Light header has parent block

...

- dark header can be implemented by adding class **.header--standard-dark** :

...

It is required to add class **.has-standard-header** into <body> tag for proper work of the template when using the standard header:

<body class="has-standard-header">...</body>

Footer

Standard footer is also represented in 2 color variations:

- Light footer has parent block:

```
<div class="footer" id="footer">...</div>
```

- dark footer can be implemented by adding class **.footer--dark** :

```
<div class="footer footer--dark" id="footer">...</div>
```

Header and footer containers

There are 2 variants of content positioning in header as well as in footer:

- Content is located in the standard container with fixed width:

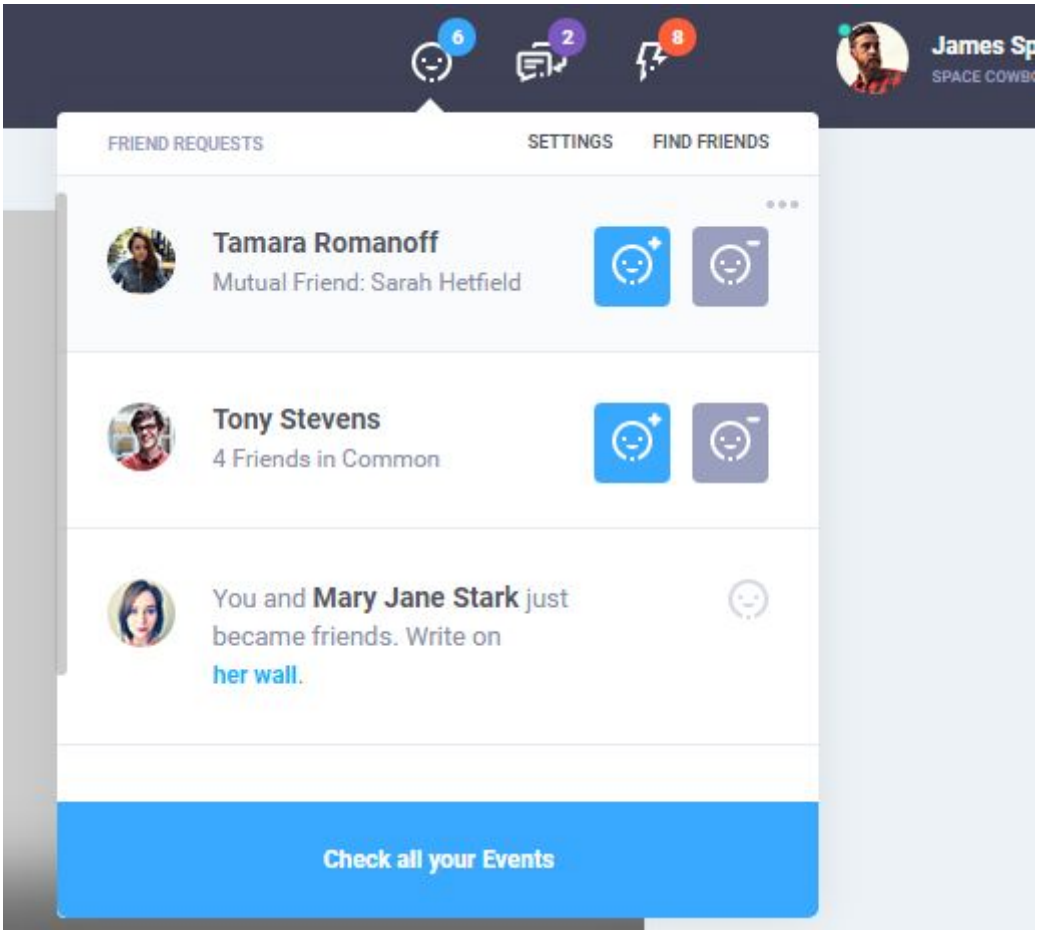
```
<div class="footer" id="footer">
                    <div class="container">
            ....
            </div>
 </div>
```

- Content occupied the entire screen width:

```
<div class="header--standard" id="header--standard">
            <div class="container-fluid">
             ...
            </div>
 </div>
```

Opening notification boxes by click

To open notification boxes by click (instead of on hover), perform the following actions:



1) Add the following styles to your project

```
.control-icon.has-items {
position: relative;
cursor: pointer;
}
.control-icon .more-dropdown.open {
opacity: 1;
visibility: visible;
transition: .3s ease;
}
```

2) Find block ** ... ** in the header section of your HTML page. This block contains such items with icons ** ... .** They must be edited as follows:

- a) Delete class "more" from each item to get such look ** ... **
- b) Add class "js-notification-open" to the icon itself.

This is what you should get as a result:

<svg class="... js-notification-open"><use
xlink:href="icons/icons.svg#olymp-happy-face-icon"></use>
</svg>
…….

```
<div class="control-block">

    <div class="control-icon has-items">
        <svg class="olymp-happy-face-icon js-notification-open"><use xlink:href="#olymp-happy-face-icon"></use></svg>
        <div class="label-avatar bg-blue">6</div>

        <div class="more-dropdown more-with-triangle triangle-top-center">
            <div class="ui-block-title ui-block-title-small">
                <h6 class="title">FRIEND REQUESTS</h6>
                <a href="#">Find Friends</a>
                <a href="#">Settings</a>
            </div>

            <div class="mCustomScrollbar" data-mcs-theme="dark">
                <ul class="notification-list friend-requests">
                    <li>
                        <div class="author-thumb">
                            <img loading="lazy" src="img/avatar55-sm.webp" alt="author" width="34" height="34">
                        </div>
                        <div class="notification-event">
                            <a href="#" class="h6 notification-friend">Tamara Romanoff</a>
                            <span class="chat-message-item">Mutual Friend: Sarah Hetfield</span>
                        </div>
```

3) There are two ways to close the box: on mouse leave or click outside the box.

- **Mouse leave.** In order to close the box on mouse leave, add the following script in the file **js/main.js** (after the comment * Toggle functions).

$('.js-notification-open').on('click', function () {
$(this).closest('.control-block').find('.more-dropdown.open').removeClass('open');
$(this).siblings('.more-dropdown').addClass('open');
$(this).siblings('.more-dropdown') .mouseleave(function(){
$(this).removeClass('open');
return false;
});
});

- **Click outside the box.** If you want to close the notification box by Escape or click outside the box, add the following script in the file **js/main.js** (after comment * Toggle functions) :

$('.js-notification-open').on('click', function () {
$(this).closest('.control-block').find('.more-dropdown.open').removeClass('open');
$(this).siblings('.more-dropdown').addClass('open');
});
// Close on click outside elements.
$document.on('click', function (event) {

```
if (!$(event.target).closest('.control-icon').length) {
$('.control-icon').children('.more-dropdown').removeClass('open');
}
});
// Close on "Esc" click
$window.keydown(function (eventObject) {
if (eventObject.which == 27) {
$('.js-notification-open').siblings('.more-dropdown').removeClass('open');
}
});
```

PS: Don't forget to flush the cache in your browser after applying changes.

Popup windows

There are numerous popup windows used in the template. For the sake of convenience, all of them are listed on a separate page **Popup-Windows.html.**

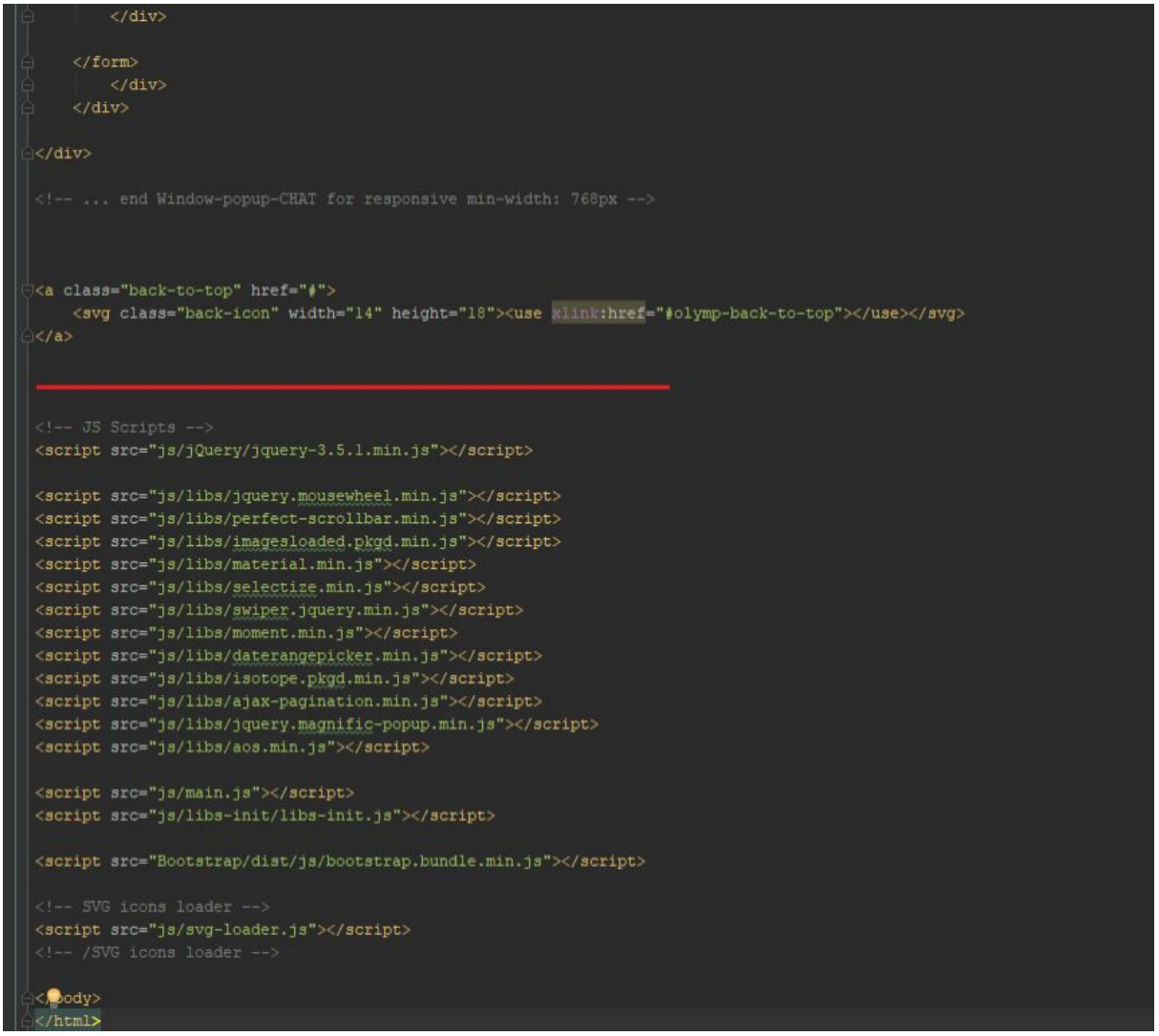
We have used standard Bootstrap popups in the template. Stylesheet for all popup windows are taken out into a separate file **..sass/blocks/popup-windows.scss**.

Creating a popup and connecting it to a page element

Bootstrap popup windows have a parent element with class **'modal'** and unique **'#id'** for each window. This element serves as a wrapper for the window. The window itself has a parent element with class **'modal-dialog'**, it is located inside the wrapper with class **'modal'**. Example:

```
<div class="modal fade" id="update-header-photo">
 <div class="modal-dialog ui-block window-popup update-header-photo">
   .....................................
 </div>
</div>
```

RECOMMENDATION: If you want to add some popup window on your page, it is recommended to paste its code, the example of which is shown above, at the bottom of the page right before JS files links.



In order to open a Bootstrap popup window by click on some page element, you need to add the following attributes to that page element:
<a href="#" data-toggle="modal" data-target="#update-header-photo">

where **'#update-header-photo'** is an id of the required popup window.

Multi-level menu

Multi-level menu in the left side panel

The standard menu in the left side panel has the following markup:

```
<ul class="left-menu">
  <li>
    <a href="#">
   <svg class="olymp-star-icon left-menu-icon">
  <use xlink:href="icons/icons.svg#olymp-star-icon"></use>
   </svg>
       <span class="left-menu-title">Fav Pages Feed</span>
```

```
 </a>
  </li>
.....................
</ul>
```

The dropdown is implemented based on the Bootstrap accordion.
In order to add a dropdown menu into the standard menu, you need to change menu item markup to the following look:

```
<ul class="left-menu">
   <li class="accordion" id="accordionExample">
      <div class="accordion-item">
         <h2 class="accordion-header" id="headingOne">
            <button class="accordion-button" type="button" data-bs-toggle="collapse"
data-bs-target="#collapseOne" aria-expanded="true" aria-controls="collapseOne">
               <svg class="olymp-newsfeed-icon left-menu-icon"><use
xlink:href="#olymp-newsfeed-icon"></use></svg>
               Profile Settings
            </button>
         </h2>
         <div id="collapseOne" class="accordion-collapse collapse show"
aria-labelledby="headingOne" data-bs-parent="#accordionExample">
            <div class="accordion-body">
               <ul class="your-profile-menu">
                  <li>
                     <a href="#">Personal Information</a>
                  </li>
                  <li>
                     <a href="#">Account Settings</a>
                  </li>
                  <li>
                     <a href="#">Change Password</a>
                  </li>
                  <li>
                     <a href="#">Hobbies and Interests</a>
                  </li>
                  <li>
                     <a href="#">Education and Employement</a>
                  </li>
               </ul>
            </div>
         </div>
      </div>
   </li>
    .......
</ul>
```

To make the dropdown menu expanded by default, you need to add class **'show'** to its container, namely:

```
<div id="collapseOne" class="accordion-collapse collapse show">
 ...
</div>
```

Widgets

The parent element for all widgets is the element with the class '**widget w-****'. All stylesheet files for widgets are located in the**..sass/blocks/..** folder and have the **«_w-...»** prefix.

Styles for each widget are taken out into a separate scss-file for ease in editing.

UI-blocks

Almost all blocks that you may see in Olympus template have the same style look and consist of **the main block frame** ... , that has **block header** ... and **block content** ... inside it.

For your convenience all styles related to these block components are located in a separate file **..sass/blocks/ui-block.scss**.

Left and right side panels

Parent elements for each side panel are ** ... ** and ** ... ** accordingly. There are 2 inner blocks inside each of them:

```
<div class="fixed-sidebar left">
   <div class="fixed-sidebar-left sidebar--small" id="sidebar-left">...</div>
   <div class="fixed-sidebar-left sidebar--large" id="sidebar-left-1">...</div>
</div>
and
<div class="fixed-sidebar right">
   <div class="fixed-sidebar-right sidebar--small" id="sidebar-right">...</div>
   <div class="fixed-sidebar-right sidebar--large" id="sidebar-right-1">...</div>
</div>
```

They are our sliding side panels. The one with **sidebar--small** class is displayed on the screen by default, another one with **sidebar--large** class - slides out from the small side panel by clicking on the element with **js-sidebar-open** class.

In other words, we have 2 blocks for the left panel and 2 blocks for the right panel. One should take it into account while editing content and styles in the side panels.

In addition, it is worth noting that for the Mobile version of the panel duplicates, respectively:

- the left panel:

```
<div class="fixed-sidebar left fixed-sidebar-responsive">
<div class="fixed-sidebar-left sidebar--small" id="sidebar-left- responsive">...</div>
div class="fixed-sidebar-left sidebar--large" id="sidebar-left-1- responsive">...</div>
</div>
```

- the right panel:

```
<div class="fixed-sidebar right fixed-sidebar-responsive" id="sidebar-right-responsive">
<div class="fixed-sidebar-right sidebar—small">...</div>
<div class="fixed-sidebar-right sidebar--large">...</div>
</div>
```

RTL version

One of the main features of Olympus is the RTL version. For activating it, you need to find and uncomment the next code in the page markup:

```
<! - Main RTL CSS ->

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

As the result you will get such a code snippet:

```
<! - Main RTL CSS ->

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

To edit the rtl-version styles use the file **\sass\rtl.scss** or **\css\rtl.min.css**

Contact Forms

To organize feedback from users, there is a possibility to send messages from a contact form,

which is provided at the **\html\95-FunctionalityContactForm.html** page.

Let's consider the organization of its functionality.

Form markup:

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

... </form> `` '

where - **** ". crumina-submit" **** - is a required class for initializing scripts for sending messages;

- **** data-nonce = "crumina-submit-form-nonce" **** - date attribute responsible for checking the data entered in the form fields and generating a message about their correctness or incorrectness;

- **** data-type = "standard" **** - date attribute that can take 2 values:

**** "standard" **** - to send messages to the specified mailbox, which is specified in the file ..html\modules\forms\inc\config.php"mailchimp", '"standard' => array('email' => 'lorem@ipsum.dolor',...)"'

and **** "mailchimp" **** - for correct interaction with MailChimp;

- **** action = "modules / forms / submit.php" **** - contains the path to the handler file.

Form field markup:

<input name = "name" class = "form-control" type = "text" placeholder = "" value = "" required>

<input name = "lastname" class = "form-control" type = "text" placeholder = "" value = "" required>

<input name = "email" class = "form-control" type = "email" placeholder = "" value = "" required>

<textarea name = "message" class = "form-control" type = "text" placeholder = "" required> </textarea>

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

- **** name = "name" **** - for the username input field;

- **** name = "lastname" **** - for entering the first name or the last name;

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

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

Note that for the form to work correctly, as well as for the user to generate messages, the next js scripts must be connected on the 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>` ``

CSS and JS

SCSS and CSS files overview

The template stylesheet is based on the SASS language. All **.scss** files are located in the **sass** folder:

1. The **sass/theme-styles** folder includes main stylesheet files of the template, namely:

- **_bootstrap-customization.scss** - styles for Bootstrap customization
- **_footer.scss** – styles for footer and components related to it
- **_header.scss** - stylesheet for the header, header variations, and all elements bundled with it;
- **_mobile-notification-tabs.scss** – stylesheet for mobile notification tabs;
- **_section.scss** — styles for rows (sections), animated background etc.
- **_shop-cart.scss** — styles for shop cart.
- **_stunning-header.scss** — styles for stunning-header;
- **_variables.scss** - includes all global variables and their styles, main colors used in the template, font size, font families, etc.

2. The **\sass\plugins** folder includes stylesheets for all the plugins, used in the template.

3. The **\sass\blocks** folder includes stylesheets of all the template's blocks and components.

4. The sass\widgets folder contains styles for all widgets in the theme. For convenience, the styles of a separate widget are moved to a separate file with the corresponding name. The file names of all widgets begin with the prefix "_w - *. Scss".

In the root of **the sass/.. folder** you will find the following files:

- **theme-font.scss** - Theme font stylesheet file;
- **rtl.scss** - stylesheets for the RTL version of the template;
- **main.scss** - it provides paths for proper compilation of the blocks stylesheet into a single file - /css/main.css.

CSS files loaded in the template

```
<!-- Bootstrap CSS -->
<link rel="stylesheet" type="text/css" href="Bootstrap/dist/css/bootstrap.css">
<!-- Main Styles CSS -->
<link rel="stylesheet" type="text/css" href="css/main.min.css">
<link rel="preload" type="text/css" href="css/theme-font.min.css" as="style">

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

bootstrap.css - Bootstrap framework stylesheet;

main.css - the main stylesheet of the template;

theme-font.min.css - theme font stylesheet file;

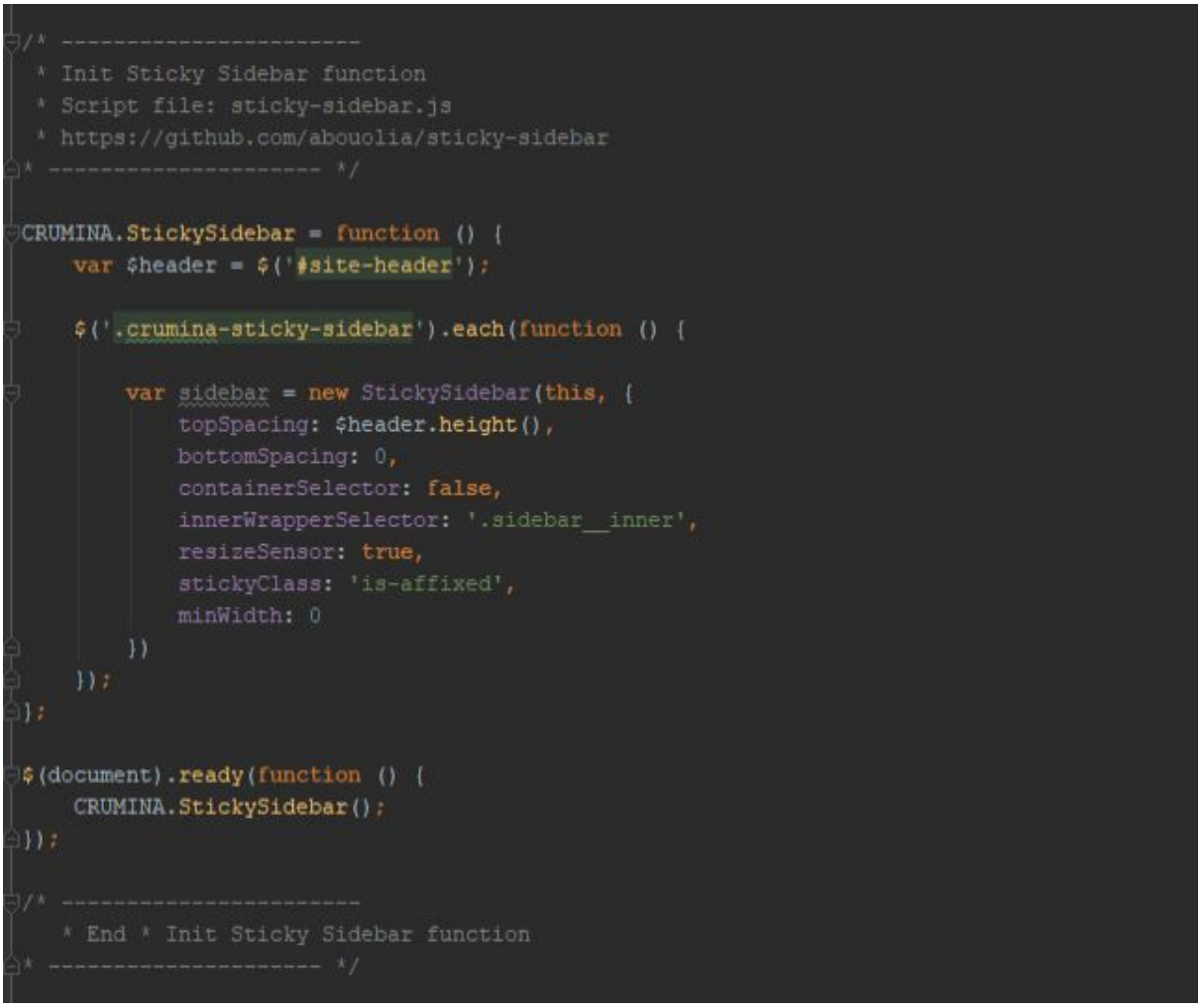
rtl.min.css - stylesheet for the RTL version of the template.

JS files description and management

All js plugins used in the template are initialized in **js/libs-init.js.**

The **libs-init.js** file is well organized and provides the following structure:

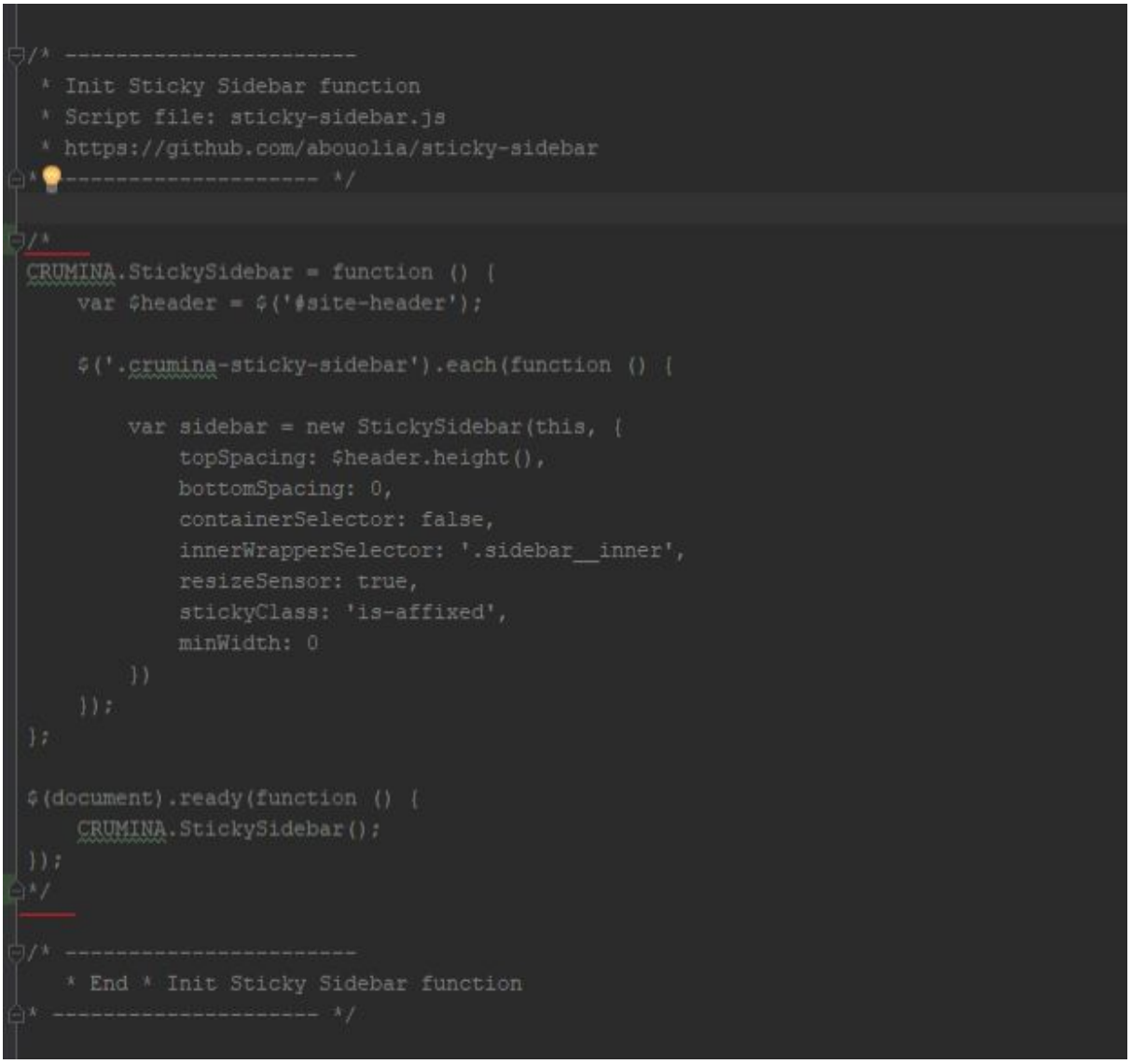
1. Description of the script plugin
2. File name responsible for the current script plugin
3. Link for the plugin documentation
4. Initialize function.



```
/* ------------------------
 * Init Sticky Sidebar function
 * Script file: sticky-sidebar.js
 * https://github.com/abouolia/sticky-sidebar
* ---------------------- */

CRUMINA.StickySidebar = function () {
    var $header = $('#site-header');

    $('.crumina-sticky-sidebar').each(function () {

        var sidebar = new StickySidebar(this, {
            topSpacing: $header.height(),
            bottomSpacing: 0,
            containerSelector: false,
            innerWrapperSelector: '.sidebar__inner',
            resizeSensor: true,
            stickyClass: 'is-affixed',
            minWidth: 0
        })
    });
};

$(document).ready(function () {
    CRUMINA.StickySidebar();
});

/* ------------------------
   * End * Init Sticky Sidebar function
* ---------------------- */
```

JS files management

In order to disable any of the scripts used, you need either to delete the script function or comment it out like this **/* commented code */**



```
/* ------------------------
 * Init Sticky Sidebar function
 * Script file: sticky-sidebar.js
 * https://github.com/abouolia/sticky-sidebar
 * ---------------------- */

/*
CRUMINA.StickySidebar = function () {
    var $header = $('#site-header');

    $('.crumina-sticky-sidebar').each(function () {

        var sidebar = new StickySidebar(this, {
            topSpacing: $header.height(),
            bottomSpacing: 0,
            containerSelector: false,
            innerWrapperSelector: '.sidebar__inner',
            resizeSensor: true,
            stickyClass: 'is-affixed',
            minWidth: 0
        })
    });
};

$(document).ready(function () {
    CRUMINA.StickySidebar();
});
*/

/* ------------------------
   * End * Init Sticky Sidebar function
 * ---------------------- */
```

JS files loaded in template

Javascript files, that load right before the closing **</body>** tag of the template are as follows:

<script src="js/jQuery/jquery-3.5.1.min.js"></script>
<script src="js/libs/jquery.appear.min.js"></script>
<script src="js/libs/jquery.mousewheel.min.js"></script>
<script src="js/libs/perfect-scrollbar.min.js"></script>
<script src="js/libs/imagesloaded.pkgd.min.js"></script>
<script src="js/libs/material.min.js"></script>
<script src="js/libs/selectize.min.js"></script>
<script src="js/libs/swiper.jquery.min.js"></script>
<script src="js/libs/moment.min.js"></script>
<script src="js/libs/daterangepicker.min.js"></script>

```
<script src="js/libs/fullcalendar.min.js"></script>
<script src="js/libs/isotope.pkgd.min.js"></script>
<script src="js/libs/ajax-pagination.min.js"></script>
<script src="js/libs/Chart.min.js"></script>
<script src="js/libs/chartjs-plugin-deferred.min.js"></script>
<script src="js/libs/circle-progress.min.js"></script>
<script src="js/libs/loader.min.js"></script>
<script src="js/libs/run-chart.min.js"></script>
<script src="js/libs/jquery.magnific-popup.min.js"></script>
<script defer src="js/libs/jquery.gifplayer.min.js"></script>
<script defer src="js/libs/mediaelement-and-player.min.js"></script>
<script defer src="js/libs/mediaelement-playlist-plugin.min.js"></script>
<script defer src="js/libs/ion.rangeSlider.min.js"></script>
<script src="js/libs/aos.min.js"></script>

<!-- JS library for Map Leaflet -->
<script src="js/libs/leaflet.min.js"></script>
<script src="js/libs/MarkerClusterGroup.min.js"></script>
<!-- ...end JS library for Map Leaflet -->

<script src="js/main.js"></script>
<script src="js/libs-init/libs-init.js"></script>
<script src="Bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<!-- SVG icons loader -->
<script src="js/svg-loader.js"></script>
<!-- /SVG icons loader -->
```

- jquery-3.5.1.js - the latest version of the jQuery library connected to the template;
- jquery.appear.js - jQuery plugin for tracking element's appearance in the browser viewport
- jquery.mousewheel.js - A jQuery plugin that adds cross-browser mouse wheel support.
- perfect-scrollbar.js — script for custom scrollbar
- svgxuse.js - a simple polyfill that fetches external SVGs referenced in use elements when the browser itself fails to do so
- imagesloaded.pkgd.js - JavaScript is all like "You images have done yet or what?"
- Headroom.js — script for sticky header
- jquery.waypoints.js - You're looking for Waypoints.
- jquery.countTo.js - A jQuery plugin that will count up (or down) to a target number at a specified speed.
- popper.min.js — Script for tooltips
- material.min.js – script for animation and building form elements.
- bootstrap-select.js - Bootstrap-select requires jQuery v1.9.1+, Bootstrap’s dropdown.js component, and Bootstrap's CSS.
- smooth-scroll.js - A lightweight script to animate scrolling to anchor links

- selectize.js - Selectize is the hybrid of a textbox and <select> box. It's jQuery-based and it has autocompleted and native-feeling keyboard navigation; useful for tagging, contact lists, etc.
- swiper.jquery.js — scripts for sliders.
- moment.js - Parse, validate, manipulate, and display dates in javascript.
- daterangepicker.js - JavaScript Date Range, Date and Time Picker Component
- fullcalendar.js - Full-sized drag & drop event calendar
- isotope.pkgd.js - Filter & sort magical layouts
- ajax-pagination.js — scripts for adding a content (load more)
- Chart.js - Simple, clean, and engaging charts for designers and developers
- chartjs-plugin-deferred.js - Chart.js plugin to defer initial chart updates
- jquery-circle-progress.js — script for building round progress bar
- loader.js - minimal, and the loader was mostly stolen from wycats.
- run-chart.js - Simple HTML5 Charts using the <canvas> tag
- jquery.magnific-popup.js - Light and responsive lightbox script with focus on performance.
- jquery.gifplayer.js - Customizable jquery plugin to play and stop animated gifs. Similar to 9gag's. Support for video formats
- mediaelement-and-player.js - HTML5 <audio> or <video> player with support for MP4, WebM, and MP3 as well as HLS, Dash, YouTube, Facebook, SoundCloud, and others with a common HTML5 MediaElement API, enabling a consistent UI in all browsers.
- ion.rangeSlider.js – script for building Range Sliders
- aos.js — Animate on scroll library
- leaflet.js - Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps.
- MarkerClusterGroup.js - Marker Clustering plugin for Leaflet
- main.js – basic java scripts
- libs-init.js – initialization of main theme scripts
- bootstrap.bundle.js — plugin for the correct positioning of elements

SVG-icons

All the theme icons locate in the folder ..\js\svg-loader.js.

They can be added to every HTML-page with the help of the following script

```
<!-- SVG icons loader -->
<script src="js/svg-loader.js"></script>
<!-- /SVG icons loader -->
```

Each icon has a unique ID. To call and place any sprite icon in the content, you should use the layout, for example:

```
<svg class="olymp-three-dots-icon"><use xlink:href="#olymp-three-dots-icon"></use></svg>
```

where:

- class="olymp-three-dots-icon” — a unique class of the current icon;

- href="#olymp-three-dots-icon" — it's a link to a unique ID of a necessary icon our sprite.

Load more in news feed

Button with **id="load-more-button"** is responsible for adding content to a page. The button itself has the following look:

```
<a id="load-more-button" href="#" class="btn btn-control btn-more"
data-load-link="items-to-load.html" data-container="newsfeed-items-grid">
 <svg class="olymp-three-dots-icon">
  <use xlink:href="icons/icons.svg#olymp-three-dots-icon"></use>
 </svg>
</a>
```

It includes the following data-attributes:
data-load-link="items-to-load.html" – where you need to specify the path to the HTML page with content to display;
data-container="newsfeed-items-grid" — where you need to specify the class of the block (div), where the content will be added.