

Для чего нужны «Чертоги Фрилансера» v3? Презентация шаблона и его ВОЗМОЖНОСТЕЙ

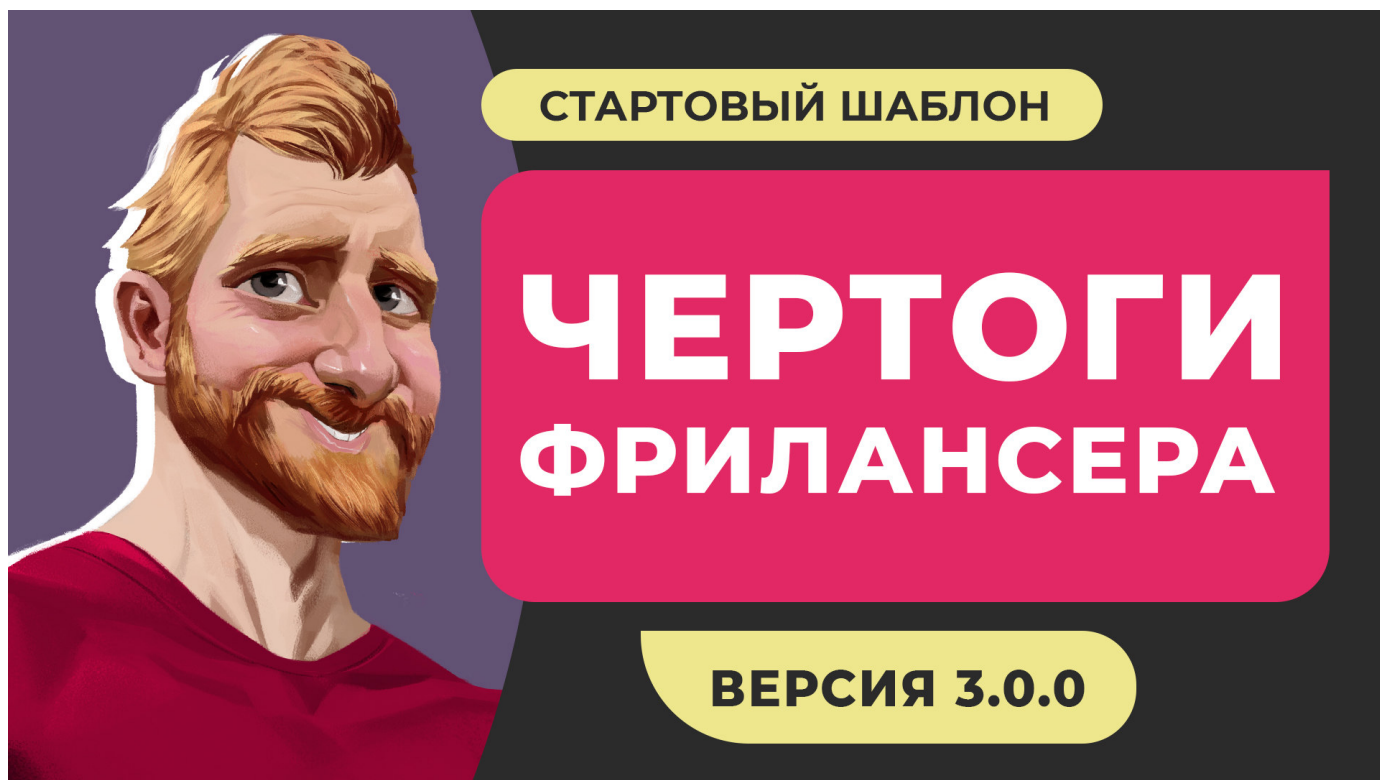
Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Привет! Это Фрилансер по жизни и я рад представить вам свой стартовый шаблон для комфортной разработки сайтов [«Чертоги Фрилансера» v3.0.0](#) (далее ЧФ)

Слово «чертоги» означает дворец, замок, хоромы. Название происходит от устойчивого выражения «Чертоги разума», как место для хранения чего-то важного.

Содержание:

- [Что такое стартовый шаблон и зачем он нужен?](#)
- [Возможности «Чертогов Фрилансера» v3.0.0](#)



Что такое стартовый шаблон и для чего он нужен?

Давайте представим себе обычный процесс подготовки к верстке/разработке сайта. Мы создаем иерархию папок и файлов, настраиваем тот или иной сборщик проекта, подключаем основные JS-дополнения для работы и т.п.

А во время разработки, раз за разом выполняем одни и те же действия: конвертируем шрифты, оптимизируем картинки, пишем один и тот же код для решения стандартных задач. Ведь меню-бургер, слайдеры, попапы, табы, валидация и отправка форм, спойлеры, галереи и так далее давно стали неотъемлемыми частями любого современного сайта.

Кроме стандартного функционала, нам то и дело необходимо решать и как более сложные задачи связанные с хотелками дизайнера/заказчика, так и дорабатывать стандартные модули под конкретные нужды.

Так вот, я более 10 лет разрабатываю сайты на фрилансе и успел поработать с самыми разнообразными дазайн-макетами, а также с очень многими заказчиками и их хотелками.

По мере решения возникающих задач я всячески старался оптимизировать и автоматизировать рутинные процессы. А самые нужные и удачные HTML/CSS/JS решения складывать в свои чертоги.

Наряду со стандартным функционалом стали появляться и собственные разработки и решения. Например [отзывчивое свойство](#), [динамический адаптив](#) и т.д.

В результате, за столько лет добра накопилось достаточно для того чтобы разработка с использованием ЧФ превратилась в сплошное удовольствие.

[Удобная архитектура папок и файлов](#), автоматизация всех рутинных процессов, не только стандартный JS-функционал, но и решения позволяющие удовлетворить кастомные, уникальные запросы. При этом, никакого лишнего кода, подключается только нужный, используемый функционал для конкретного проекта, что делает результат разработки с ЧФ максимально оптимизированным и быстрым.

Отвечая на вопрос что такое стартовый шаблон и зачем он нужен, я приведу три основных пункта:

1. Стартовый шаблон [«Чертоги Фрилансера» v3.0.0](#) — это совокупность оптимизации и автоматизации рутинных процессов во время разработки сайтов. База большого количества как стандартных так и уникальных решений которые помогут решить множество возникающих задач за считанные секунды.
2. ЧФ разработан так, чтобы не притуплять ваши навыки в разработке. Найден баланс между готовыми решениями и полным пониманием пользователем шаблона того как все устроено «под капотом». Также крайне подробное комментирование кода делает из ЧФ обучающий инструмент для развития вас как специалистов.
3. **Шаблон «Чертоги Фрилансера» — работает с разработчиком, а не вместо него!**

Возможности «Чертогов Фрилансера» v3.0.0

Тут собраны все возможности стартового шаблона включая возможности сборщика. Данные будут обновляться по мере добавления функционала.

Сборщики и их режимы работы

В шаблоне работают два сборщика GULP и WEBPACK, каждый делает то в чем он хорош. Есть возможность работать в режимах разработки и продакшена. Режим разработки выполняет только необходимые для работы задачи, а продакшн обеспечивает оптимизацию файлов, конвертацию и прочие необходимые действия. Таким образом повышается скорость работы сборщиков. Добавлю что тип работы указанный в package.json равен module

Сборщик GULP

GULP играет роль вспомогательного сборщика и выполняет следующие задачи:

- Очищает папку с результатом [все режимы]
- Создает файл .gitignore с указанным содержимым [все режимы]
- **Конвертирует шрифты из .OTF и .TTF в .WOFF и .WOFF2. Записывает данные в файл стилей, в том числе формирует font-weight на основе имени файла.** [все режимы]
- Обрабатывает HTML файлы: добавляет конструкцию <picture> с изображением в .WEBP формате, добавляет к подключенным JS CSS файлам версию, что позволяет избежать проблем с кэшированием при демонстрации работы. [режим продакшена]
- Обрабатывает CSS файлы: группирует медиа-запросы, добавляет вендорные префиксы для обеспечения кроссбраузерности, добавляет подключение .WEBP изображения, оптимизирует и сжимает файл. [режим продакшена]
- Обрабатывает JS файлы: только передает информацию сборщику WEBPACK [режим продакшена]
- Обрабатывает файлы изображений: сжимает и оптимизирует форматы .JPG, .PNG, .SVG. На основе .JPG/.PNG файлов формируется .WEBP изображение. [режим продакшена]
- Создает SVG спрайт [все режимы]

- Отправляет файлы с результатом на сервер по FTP [режим продакшена]
- Создает ZIP архив с результатом [режим продакшена]

Сборщик WEBPACK

WEBPACK играет основную роль сборщика HTML/SCSS/JS файлов и выполняет следующие задачи:

- Запускает локальный сервер и открывает HTML страницы результата в браузере. При внесении изменений, сам обновляет страницу. Подключен модуль горячей замены, что увеличивает скорость обновления. Также, созданные файлы не пишутся на диск, а находятся в памяти. [режим разработчика]
- Обработывает SCSS файлы: производится преобразование в CSS файлы, переименовываются псевдонимы [все режимы]
- Обработывает HTML файлы: производится сборка файлов из подключаемых частей, переименовываются псевдонимы [все режимы]
- Обработывает PUG файлы: производится сборка файлов из подключаемых частей, переименовываются псевдонимы, производится преобразование в HTML файлы [все режимы]
- Обработывает JS файлы: производится грамотная сборка подключенных модулей, таким образом в файл с результатом попадает только задействованный в проекте код [все режимы], файл сжимается и оптимизируется [режим продакшена]
- Копирует файлы из папки files в папку с результатом [все режимы]

Возможности работы с HTML файлами

HTML файлы можно собирать из частей благодаря шаблонизатору **FileInclude**. То есть мы можем вынести отдельные части кода, например header и footer, в отдельные файлы и подключать их для всех страниц сайта. Таким образом, при необходимости внести изменения, нам достаточно редактировать только один подключаемый файл. FileInclude позволяет передавать переменные со значением в подключаемый файл. Например мы можем передать свой заголовок (title) для каждой страницы.

Шаблонизатор PUG

Если вам не хватает возможностей FileInclude вы можете использовать шаблонизатор PUG и все его возможности. Сборка сама обрабатывает PUG-файлы и преобразует в HTML

Использование препроцессора SASS/SCSS, возможности при работе со стилями

В шаблоне используется препроцессор SASS в синтаксисе SCSS. Благодаря этому, есть возможность порадовать вас отличными наработками:

- **Отзывчивое свойство**. Миксин, позволяет задать значение для того или иного свойства которое будет зависеть от ширины экрана (вьюпорта), что в свою очередь открывает колоссальные возможности для адаптивной (отзывчивой) верстки.
- **Архитектура подключаемых файлов** позволяет использовать только то что нужно для конкретного проекта.
- Готовые базовые стили для различных JS модулей

JavaScript возможности, использование модулей

Отмечу, что поскольку JS файлы собирает WEBPACK, то у нас появляется возможность использовать подключение ES6 модулей. Таким образом в результат попадает только используемый функционал. Для добавления той или иной возможности, как правило, достаточно раскомментировать строку. Также каждый модуль удобно управляется и настраивается с помощью дата атрибутов в HTML/PUG файлах, а сам код подробно прокомментирован.

Вот перечень готового JS функционала «Чертогов Фрилансера» (для подробностей кликайте на название модуля):

Общие модули

- **Табы** с возможностью превращения в спойлеры на указанной ширине экрана и прочими настройками
- **Спойлеры** с возможностью включения на указанной ширине экрана и прочими настройками

- **Модуль «Показать еще»** с возможностью включения на указанной ширине экрана и прочими настройками
- **Попапы** (всплывающие/модальные окна) с переключением фокуса, открытием по хешу и прочим
- Галерея (куплена лицензия галереи lightgallery.js)
- Слайдер Swiper и дополнительные возможности для интеграции с другими модулями
- Подсказки (tippy.js)
- **Функционал меню «бургер»**
- Функционал определения поддержки WEBP формата
- Вспомогательные, общие для многих модулей, функции блокировки прокрутки с компенсацией скроллбара, плавного раскрытия блока, сбор и уникализация медиа-запросов, форматирования чисел, исправление ошибки полноэкранного блока на мобильных и многое другое.

Модули работы с формами

- **Работа с полями ввода**, добавление классов, работа с placeholder
- **Валидация полей**, возможность моментальной валидации
- **Функционал показа пароля**
- **Отправка форм в разных режимах**: обычный, AJAX, режим разработчика. Возможность вывода попапа после отправки формы.
- **Модуль кастомизации элемента SELECT** с большим набором возможностей
- Модуль работы с масками полей ввода
- Модуль ползунка (range) для указания значений поля
- Модуль звездного рейтинга
- Модуль указания количества кнопками + и —

Модули работы с прокруткой страницы

- Изменение дизайна скроллбара
- **Ленивая (отложенная) загрузка данных (lazyload)**: картинки видео iframe и т.д. при прокрутке страницы
- Наблюдатель за появлением элементов. Позволяет создавать анимацию и многое другое в момент появления/ухода элемента с

видимой области экрана (блока)

- **Навигация по странице** (прокрутка к нужному элементу). Также есть возможность добавления класса активному пункту навигации
- **Работа с шапкой сайта при прокрутке страницы** с дополнительными возможностями
- «Липкий» блок. Возможность закреплять элемент при прокрутке с гибкими настройками

Модули для создания параллакс-эффекта

Модуль параллакса мышью. Элементы плавно реагируют на движение мышью

Система глобального логгинга (Full Logging System или FLS)

Я подумал, почему бы не научить функционал ЧФ общаться (отчитываться) о своей работе пользователю в консоли?! Это очень удобно в процессе разработки и имеет обучающий эффект. **Подробнее**

Список будет постоянно пополняться новыми возможностями. Следите за **обновлениями**.

Спасибо за **поддержку** канала **«Фрилансер по жизни»** и бесплатного обучающего контента!

Обучайся развивайся и помни — живи, а работай в свободное время!

[Следующий пост: Установка и запуск шаблона. Подготовка к работе ->](#)

Установка и запуск шаблона. Подготовка к работе

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Итак, вы [скачали шаблон «Чертоги Фрилансера» v3.0.0](#) (далее ЧФ), какие следующие шаги?

Установка шаблона

Первым делом следует разархивировать zip-архив в вашу папку проекта. Следите за тем, чтобы сама папка и её родительские папки не содержали в названии кириллицу, пробелы, символы # и !



← → ↕ ⬆ > Этот компьютер > DATA (E:) > WEB > DEV > START_TEMPLATES > new_project >			
	config		Дата изменения: 03.01.2022 08:35
	src		Дата изменения: 03.01.2022 08:35
	cover.jpg	Тип: Файл "JPG" Разрешение: 1920 x 1080	Размер: 576 КБ
	gulpfile.js	Тип: Исходный файл JavaScript	Дата изменения: 02.01.2022 07:36 Размер: 2,41 КБ
	package.json	Тип: Исходный файл JSON	Дата изменения: 02.01.2022 07:36 Размер: 2,11 КБ
	README.txt		Дата изменения: 02.01.2022 07:36 Размер: 4,54 КБ
	snippets.txt		Дата изменения: 02.01.2022 07:36 Размер: 24,2 КБ

Содержимое архива в папке проекта

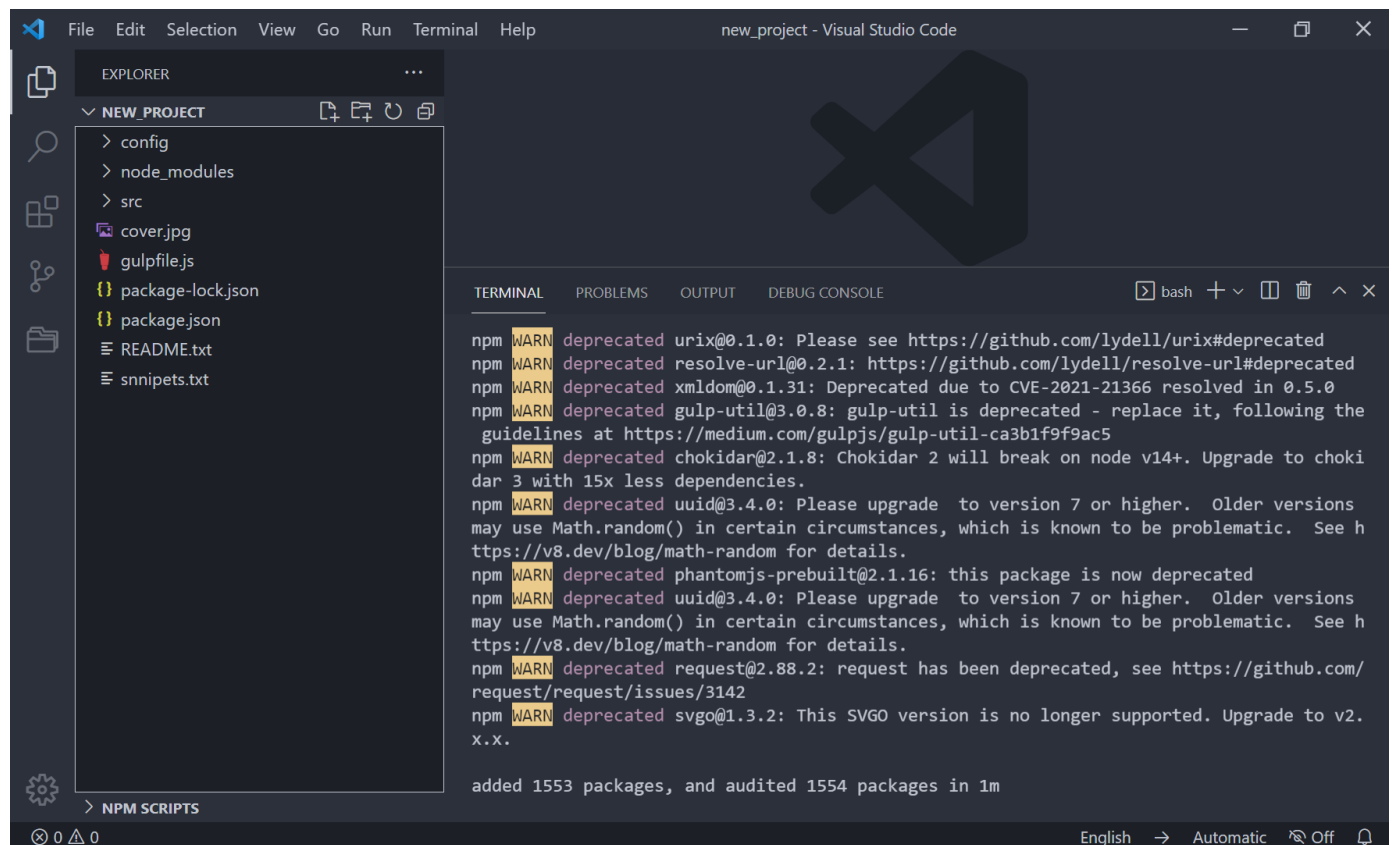
Если вы до этого момента не пользовались сборщиками GULP, WEBPACK и пакетным менеджером NPM, следует [скачать и установить Node.js](#). Качайте версию рекомендованную для большинства.

Далее, открываем терминал в этой папке проекта. Терминал может быть встроен или открыт отдельно от редактора. Рекомендую использовать терминал **GIT Bash**.

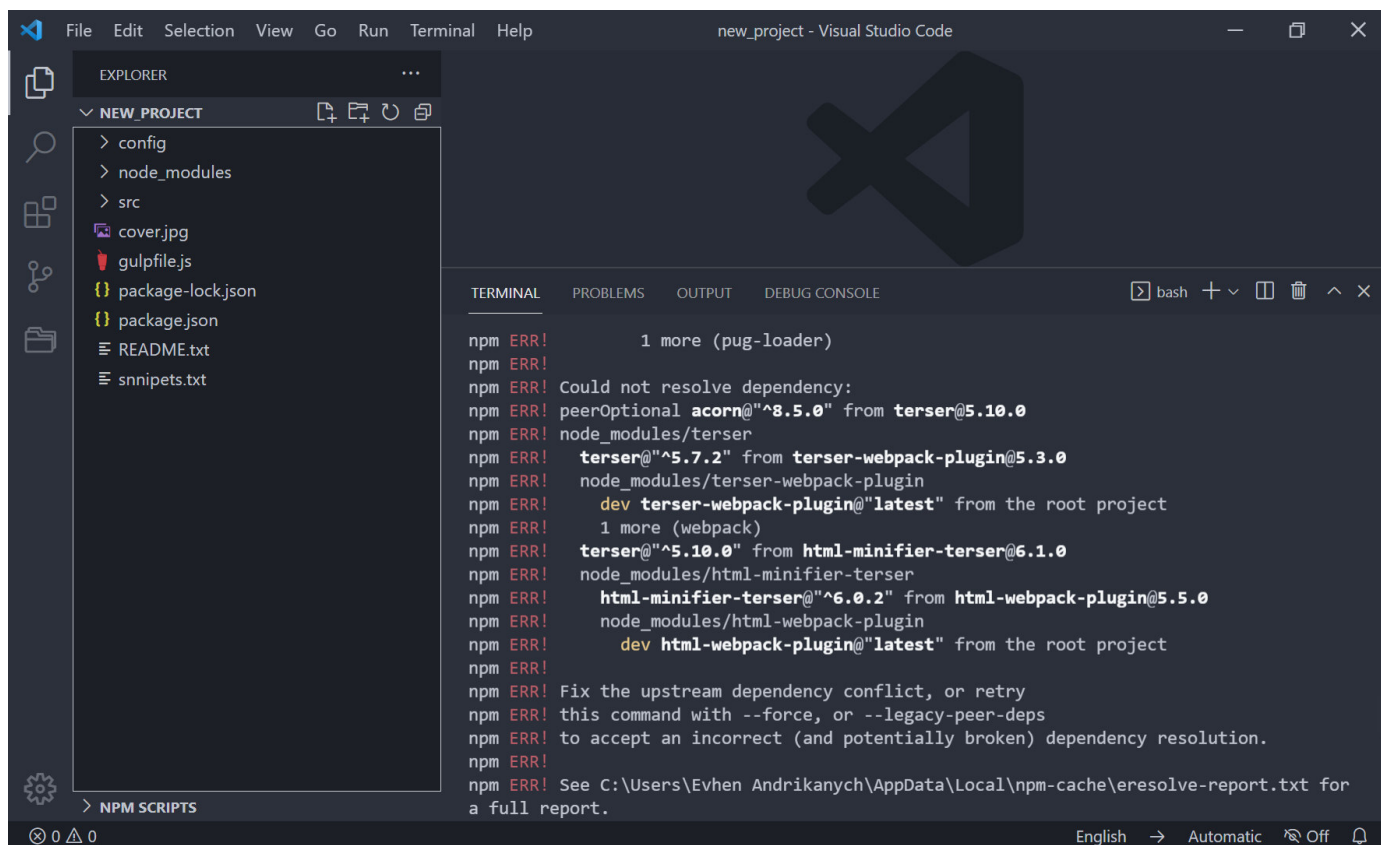
Если вы до этого момента не пользовались сборщиком GULP введите в терминал команду **npm i gulp-cli -g** это установит GULP в систему глобально.

После завершения установки GULP введите команду **npm i**

В процессе установки в терминале могут возникать сообщение с пометкой WARN на желтом фоне. Эти сообщения можно игнорировать. Но, если вы получаете сообщение ERR! красным цветом — это критическая ошибка и её нужно исправлять.

The screenshot shows the Visual Studio Code interface. On the left, the Explorer panel shows a project named 'NEW_PROJECT' with files like 'config', 'node_modules', 'src', 'cover.jpg', 'gulpfile.js', 'package-lock.json', 'package.json', 'README.txt', and 'snippets.txt'. The main editor area is empty. The bottom panel shows the Terminal with the output of an npm installation. The output consists of several 'npm WARN deprecated' messages for various packages: 'urix@0.1.0', 'resolve-url@0.2.1', 'xmldom@0.1.31', 'gulp-util@3.0.8', 'chokidar@2.1.8', 'uid@3.4.0', 'phantomjs-prebuilt@2.1.16', 'request@2.8.2', and 'svgo@1.3.2'. Each warning includes a brief explanation and a link to a GitHub issue or a Medium article. At the bottom of the terminal output, it says 'added 1553 packages, and audited 1554 packages in 1m'. The status bar at the bottom shows '0 errors, 0 warnings, 0 info'.

Предупреждения можно игнорировать



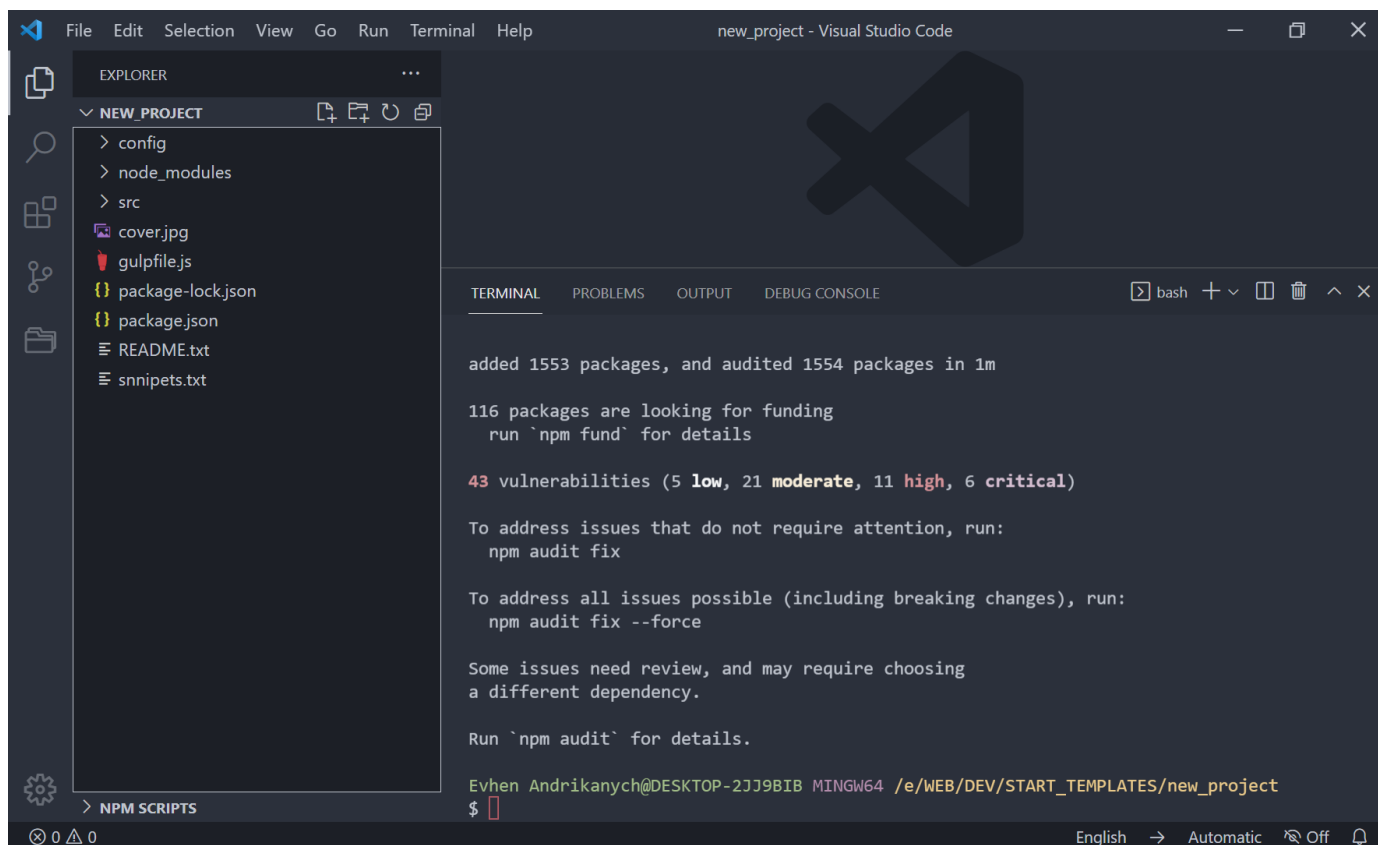
Ошибка зависимостей

Часто может возникнуть ошибка зависимостей устанавливаемых пакетов (плагинов), и в NPM выше 7й версии это приводит к критической ошибке и остановке инсталляции.

Чтобы решить проблему выполните команду **npm i --legacy-peer-deps**, это запустит процесс установки игнорируя подобные несовместимости.

Помните — версии NodeJS и Python в вашей системе должны быть свежих версий.

После успешной установки, у вас появится папка `node_modules` и файл `package-lock.json`



Установка завершена

Подробнее про [архитектуру папок и файлов](#) ЧФ мы поговорим в следующей главе, а пока продолжаем подготовку к работе

Запуск шаблона и режимы работы

Шаблон «Чертоги Фрилансера» может выполнять несколько сценариев:

1. Режим разработчика. Команда запуска **npm run dev**
2. Режим продакшена. Команда запуска **npm run build**
3. Режим продакшена и отправка результата на сервер по FTP.
Команда запуска **npm run deploy**
4. Режим продакшена и создание ZIP-архива с результатом. Команда запуска **npm run zip**
5. Режим продакшена без создания WEBP изображений и действий связанных с этим форматом. Команда запуска **npm run devbuild**

Дополнительные команды:

1. Ручное создание SVG спрайта. Команда запуска **npm run sprite**
2. Конвертация шрифтов с принудительной перезаписью файла стилей. Команда запуска **npm run fonts**

Режим разработчика

В режиме разработчика выполняются только необходимые для процесса разработки задачи:

- Конвертация шрифтов и запись в файл стилей
- Конвертация SCSS файлов в CSS файлы, переименование псевдонимов
- Сборка HTML файлов, переименование псевдонимов
- При использовании PUG, файлы преобразовываются в HTML, переименовываются псевдонимы
- Собираются JS файлы
- Запускается локальный сервер, открывается браузер с индексной страницей.
- Запускается наблюдатель за изменением файлов. При каждом изменении файла браузер обновляет страницу (кроме страницы-содержания)
- Копируются файлы из указанной папки
- HTML CSS JS файлы результата не записываются на диск (папка dist не создается), это увеличивает скорость работы.

Режим продакшена

В режиме продакшена выполняется финализация проекта, а именно:

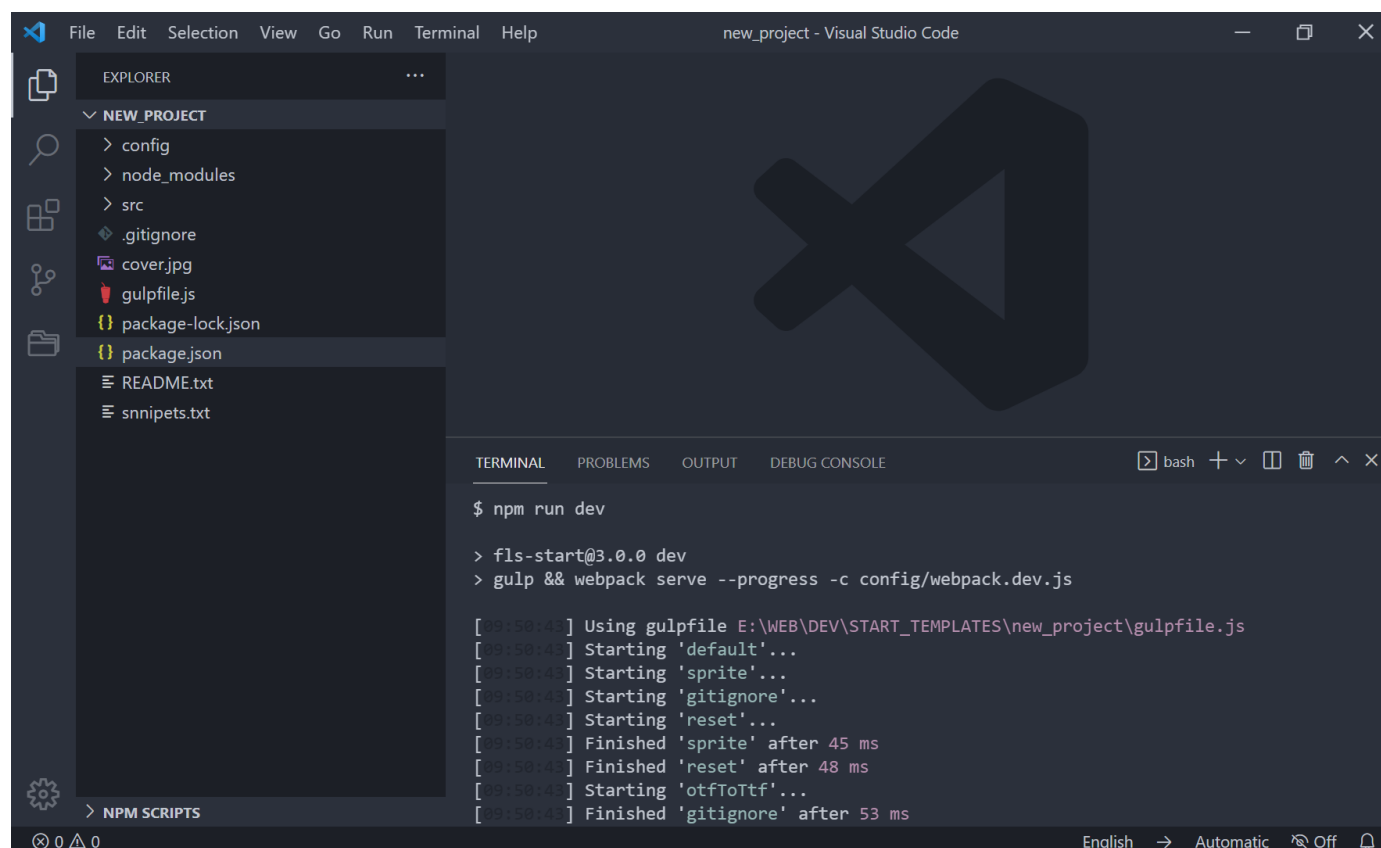
- Конвертация шрифтов и запись в файл стилей
- Конвертация изображений в WEBP формат. Сжатие и оптимизация JPG PNG SVG картинок
- Конвертация SCSS файлов в CSS файлы, переименование псевдонимов, группировка медиа-запросов, добавляются вендорные префиксы для обеспечения кроссбраузерности, обрабатывается подключение WEBP изображений, производится сжатие и оптимизация конечного файла (также создается несжатая копия)
- Сборка HTML файлов, переименование псевдонимов, обрабатывается подключение WEBP изображений
- При использовании PUG, файлы преобразовываются в HTML, переименовываются псевдонимы, обрабатывается подключение WEBP изображений

- Собираются JS файлы, производится сжатие и оптимизация конечного файла. В результат попадает только используемый код. Итогом задачи будет создание двух файлов: сжатого app.min.js (подключен к HTML) и не сжатого app.js для дальнейшего редактирования другими специалистами.
- Копируются файлы из указанной папки
- Все файлы с результатом записываются на диск (обычно в папку dist), локальный сервер не запускается

Итак, давайте же запустим наш шаблон в режиме разработчика, для этого в терминале выполняем команду **npm run dev**

После запуска система выполнит все задачи режима разработчика описанные выше. Результатом работы должна стать открытая в браузере страница содержания.

Внимание, страница содержания не обновляется автоматически при её редактировании



```
File Edit Selection View Go Run Terminal Help
new_project - Visual Studio Code

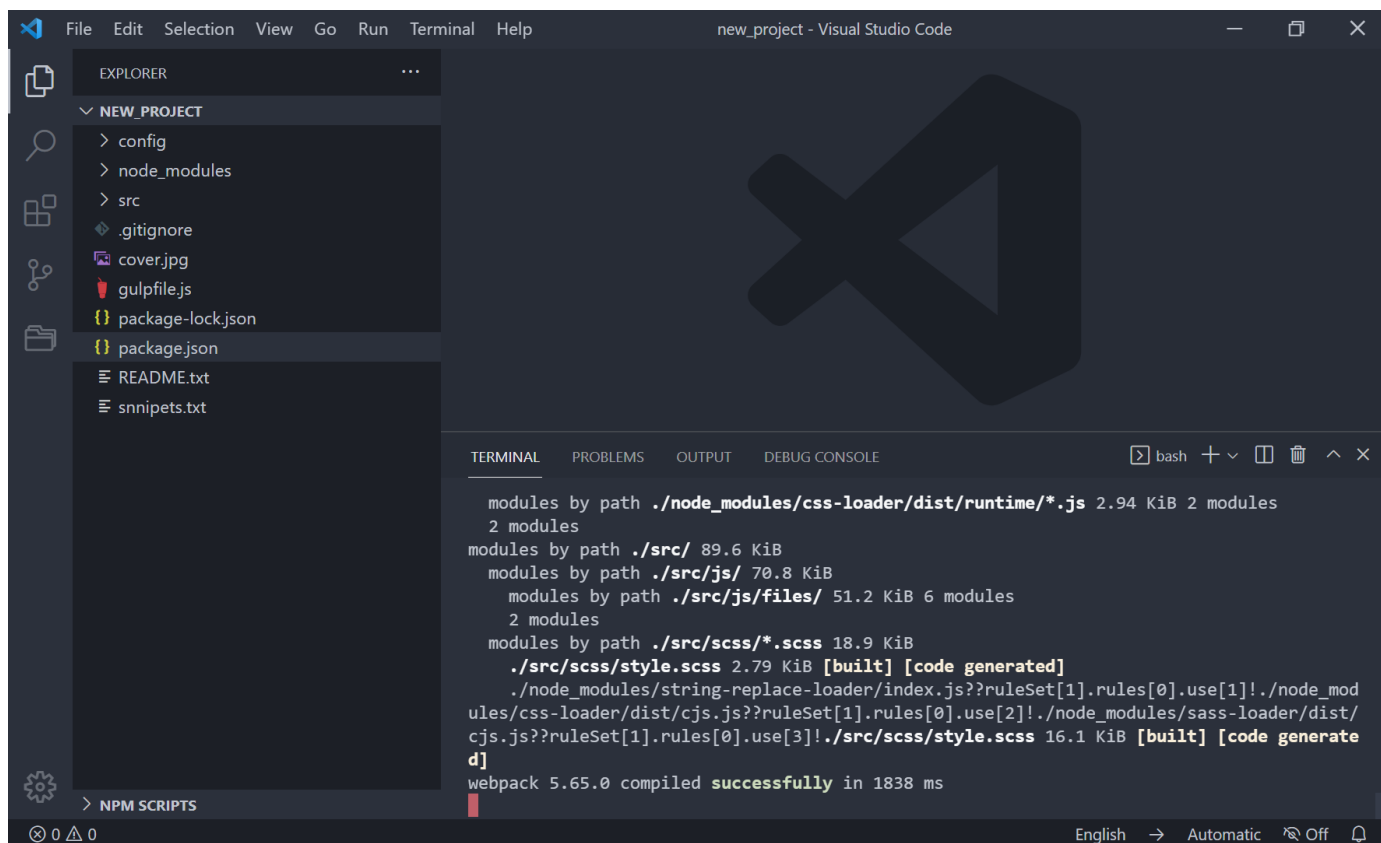
EXPLORER
NEW_PROJECT
  > config
  > node_modules
  > src
  .gitignore
  cover.jpg
  gulpfile.js
  package-lock.json
  package.json
  README.txt
  snippets.txt

TERMINAL
$ npm run dev

> fls-start@3.0.0 dev
> gulp && webpack serve --progress -c config/webpack.dev.js

[ 9:50:4 ] Using gulpfile E:\WEB\DEV\START_TEMPLATES\new_project\gulpfile.js
[ 9:50:4 ] Starting 'default'...
[ 9:50:4 ] Starting 'sprite'...
[ 9:50:4 ] Starting 'gitignore'...
[ 9:50:4 ] Starting 'reset'...
[ 9:50:4 ] Finished 'sprite' after 45 ms
[ 9:50:4 ] Finished 'reset' after 48 ms
[ 9:50:4 ] Starting 'otfToTtf'...
[ 9:50:4 ] Finished 'gitignore' after 53 ms
```

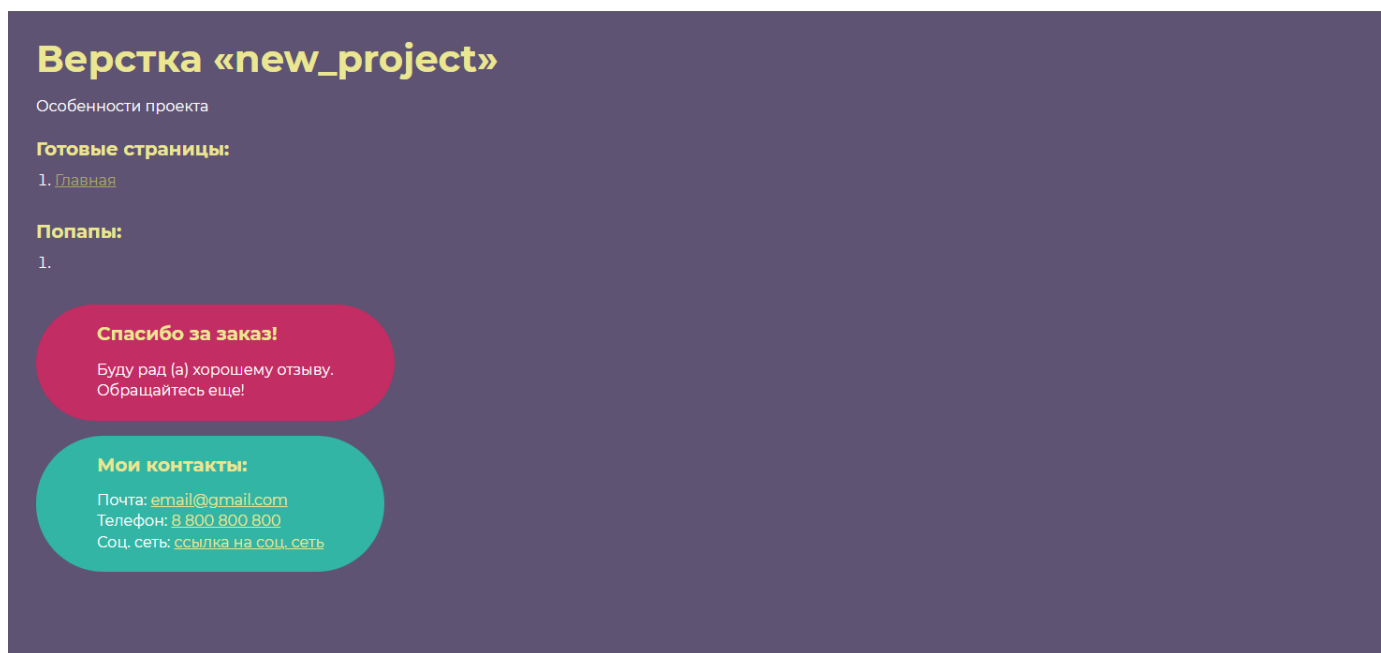
Старт выполнения команды



The screenshot shows the Visual Studio Code interface with a project named 'new_project'. The Explorer sidebar on the left lists files: config, node_modules, src, .gitignore, cover.jpg, gulpfile.js, package-lock.json, package.json, README.txt, and snnipets.txt. The Terminal panel at the bottom displays the output of a webpack build command, showing module sizes and a successful compilation message: 'webpack 5.65.0 compiled successfully in 1838 ms'.

```
modules by path ./node_modules/css-loader/dist/runtime/*.js 2.94 KiB 2 modules
2 modules
modules by path ./src/ 89.6 KiB
modules by path ./src/js/ 70.8 KiB
modules by path ./src/js/files/ 51.2 KiB 6 modules
2 modules
modules by path ./src/scss/*.scss 18.9 KiB
./src/scss/style.scss 2.79 KiB [built] [code generated]
./node_modules/string-replace-loader/index.js??ruleSet[1].rules[0].use[1]!./node_modules/css-loader/dist/cjs.js??ruleSet[1].rules[0].use[2]!./node_modules/sass-loader/dist/cjs.js??ruleSet[1].rules[0].use[3]!./src/scss/style.scss 16.1 KiB [built] [code generated]
webpack 5.65.0 compiled successfully in 1838 ms
```

Команда успешно выполнена



Страница содержания открыта в браузере

Возможные ошибки и их решения

Если браузер не запустился, а в терминале видны ошибки (ERR!) убедитесь что:

- У вас установлен Node.js и Python последней версии
- Терминал открыт с правами администратора
- В названиях папок на всем пути к проекту нет символа # или !

- Папки и файлы должны быть названы латиницей без пробелов
- Тег `img` и его содержимое должны быть записаны в одну строку без переносов
- В атрибуте `src` должен быть указан путь к существующей картинке без пробелов

При ошибке связанной с `node-sass` запустите команду **`npm rebuild node-sass`**

При ошибке связанной с Python запустите команду **`npm install -g windows-build-tools`**

Подготовка редактора к комфортной работе с шаблоном

Для того чтобы наслаждаться возможностями шаблона ЧФ по полной, нам следует произвести некоторые настройки редактора. В качестве примера представлен редактор VS Code

Настройка псевдонимов

В главе [Архитектура шаблона. Файлы и папки](#) вы узнаете что различные части HTML/PUG/SCSS файлов находятся на разных уровнях вложенности, что создает определенные неудобства при подключении, например картинок, в процессе разработки.

Неудобства мы не любим поэтому настроим, так называемые, псевдонимы (алиасы) путей к папкам. Для этого нам нужно установить плагин [Path Autocomplete](#). После установки открываем настройки редактора (`settings.json`), для этого жмем `F1` в редакторе и в строке поиска пишем `Open Settings` и жмем на ссылку `Open Settings (JSON)`.

В этот файл нужно аккуратно вставить следующий код:

```
"path-autocomplete.pathMappings": {
  "@img": "${folder}/src/img", // alias for images
  "@scss": "${folder}/src/scss", // alias for scss
  "@js": "${folder}/src/js", // alias for js
}
```

Очень важно соблюдать синтаксис JSON, обращайте внимания на запятые:

```
    },  
    "path-autocomplete.pathMappings": {  
      "@img": "${folder}/src/img", // alias for images  
      "@scss": "${folder}/src/scss", // alias for scss  
      "@js": "${folder}/src/js", // alias for js  
    },  
    "editor.unicodeHighlight.ambiguousCharacters": false,  
    "window.zoomLevel": 3  
  }  
}
```

Правильная вставка кода

После этого вы смело можете использовать псевдонимы при подключении файлов, например:

```

```

Редактор распознает псевдоним и выведет список файлов в указанной папке, а во время сборки система сама поменяет псевдоним на нужный путь!

Настройка сниппетов

Сниппеты — это короткие коды которые могут вызывать готовые заготовки кода любого объема. Это колоссально повышает скорость разработки.

Конечно же, я использовал эту супер возможность в своих чертогах. То есть, построение, например, правильной HTML структуры для того или иного JS модуля я добавил в сниппеты.

Чтобы добавить набор сниппетов из шаблона к себе в редактор выполняем следующие действия:

1. Открываем файл snippets.txt из архива ЧФ и копируем все содержимое
2. В редакторе VS Code переходим в настройки (шестеренка) -> User Snippets -> New Global Snippets File пишем название, например fls, и

заменяем содержимое на скопированное из snnipets.txt

3. При обновлении шаблона и сниппетов создавать новый файл не нужно, следует открыть существующий и обновить содержимое

Отлично, сниппеты ЧФ на борту! В этой документации, а также в коде шаблона вы часто будете встречать подсказки с указанием сниппета.

В следующей главе мы познакомимся со [структурой файлов и папок шаблона «Чертоги Фрилансера» v3.0.0](#)

Приятной работы

Дополнительные материалы

Для лучшего понимания установки Node.JS, GIT Bash, GULP, настройки псевдонимов посмотрите это видео:

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Для чего нужны «Чертоги Фрилансера» v3?](#)

[Презентация шаблона и его возможностей](#)

[Следующий пост: Архитектура шаблона. Файлы и папки ->](#)

Архитектура шаблона. Файлы и папки

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

В этой главе поговорим об архитектуре, иерархии папок и файлов, [стартового шаблона «Чертоги Фрилансера» v3.0.0](#) (далее ЧФ)

Разархивировав скачанный шаблон, в корне вы обнаружите следующие файлы и папки:

1. Файл **package.json** — файл содержащий команды запуска сборщиков, информацию об установленных плагинах и их версиях (можно редактировать при необходимости). А также прочую информацию о об авторе и версии шаблона.
2. Файл **gulpfile.js** — главный файл сборщика GULP. Содержит подключения различных задач и сценарии их выполнения (можно редактировать при необходимости)
3. Файл **snnipets.txt** — файл со сниппетами которые используются при работе с шаблоном. Как их добавить в свой редактор [описано тут](#)
4. Файл **README.txt** — краткая информация по работе с шаблоном
5. Файл **cover.jpg** — обложка ЧФ
6. Папка **config** — содержит папки и файлы для настройки работы сборщиков, а также файлы с отдельными задачами. [подробнее..](#)
7. Папка **src** — содержит исходные файлы и папки проекта. [подробнее...](#)

Содержимое папки config:

1. Файл **gulp-settings.js** — содержит настройку путей к файлам и папкам для работы GULP, а также настройку FTP соединения для выгрузки результата на сервер

2. Файл **gulp-plugins.js** — вспомогательный файл, содержит подключение и экспорт списка общих плагинов для задач GULP
3. Файл **webpack.dev.js** — содержит конфигурацию работы WEBPACK в режиме разработчика
4. Файл **webpack.prod.js** — содержит конфигурацию работы WEBPACK в режиме продакшн
5. Папка **gulp-tasks** — содержит файлы отдельных задач GULP

Содержимое папки src:

1. Файл **favicon.ico** — иконка сайта
2. Файл **index.html** — индексная страница-содержание, она не обновляется автоматически при редактировании (при желании удалить)
3. Файл **home.html** — главная страница сайта (при желании переименовать в index.html)
4. Папка **files** (пустая) — все файлы из этой папки будут скопированы в папку с результатом (dist/files)
5. Папка **fonts** (пустая) — используется для подключения локальных и иконочных шрифтов
6. Папка **svgicons** (пустая) — используется для создания SVG спрайта
7. Папка **html** — содержит .HTM файлы, которые подключаются в HTML файлы страниц (например в home.html). Тут можно создавать свои подключаемые файлы
8. Папка **scss** — содержит файлы и папки стилей сайта
9. Папка **js** — содержит файлы и папки скриптов сайта
10. Папка **img** — тут хранятся картинки проекта. Изначально, только кавер-фото шаблона

Содержимое папки src/html:

Файлы из этой папки предназначены для подключения в HTML-файлы страниц.

1. Файл **_header.htm** — подготовка для создания шапки сайта. Содержит тег <header> и ограничивающий контейнер. Изначально подключен в home.html

2. Файл **_footer.htm** — подготовка для создания подвала сайта. Содержит тег <footer> и ограничивающий контейнер. Изначально подключен в home.html
3. Файл **_head.htm** — содержит тег <head> внутри которого указаны метатеги, подключена иконка сайта, файл стилей а также тег <title> с переменной-заголовком. Изначально подключен в home.html
4. Файл **_js.htm** — предназначен для подключения общих js-файлов. Содержит подключение основного файла js/app.min.js из результата. Изначально подключен в home.html
5. Файл **_popup.htm** — подготовка для создания попапов. Содержит шаблонный, закомментированный HTML-код. Изначально подключен в home.html. [Смотри работу с попапами](#)
6. Файл **_pagging.htm** — подготовка для создания блока постраничной навигации. Содержит шаблонный HTML-код.

Содержимое папки src/scss:

1. Файл **style.scss** — основной файл стилей. Содержит различные настройки и подключения других файлов. [Подробнее...](#)
2. Файл **home.scss** — файл стилей главной страницы. Изначально пуст и подключен в style.scss
3. Файл **header.scss** — файл стилей шапки сайта. Содержит закомментированный код [кнопки меню-бургера](#). Изначально подключен в style.scss
4. Файл **footer.scss** — файл стилей подвала сайта. Изначально пуст и подключен в style.scss
5. Файл **common.scss** — файл стилей для общих, переиспользуемых блоков конкретного проекта. Изначально пуст и подключен в style.scss
6. Файл **base.scss** — файл базовых стилей шаблона ЧФ. Содержит полезные SCSS-шаблоны и закомментированные подключения файлов с базовыми стилями различных модулей.
7. Папка **libs** — содержит файлы полных (заводских) стилей различных модулей и плагинов
8. Папка **fonts** — содержит файл(ы) стилей связанных со шрифтами. Содержит файл icons.scss для подключения иконочного шрифта.

Также, в процессе подключения локальных шрифтов, тут появляется файл `fonts.scss`

9. Папка **base** — содержит вспомогательные файлы стилей ЧФ, а также файлы с базовыми стилями различных моделей. Для удобства, файлы модулей одной группы (например элементы форм) собраны в отдельные подпапки. Тут же находится файл с обнуляющими стилями **null.scss**

Содержимое папки `src/js`:

1. Файл **app.js** — основной файл скриптов. Служит для подключения необходимого в проекте функционала и прочих настроек.
2. Папка **libs** — содержит файлы готовых дополнений, как правило, не предусмотренных для редактирования.
3. Папка **files** — содержит файлы различных модулей и прочего функционала.

Содержимое папки `src/js/files`:

1. Файл **script.js** — предназначен для написания своего кода для проекта. Изначально импортирован в `app.js`
2. Файл **functions.js** — содержит мелкие модули и различный вспомогательный функционал. Изначально импортирован в `app.js`, но модули закомментированны
3. Файл **gallery.js** — содержит подключение плагина и стилей галереи. Изначально импортирован в `app.js`, закомментированн. При необходимости, в него можно вносить изменения
4. Файл **sliders.js** — содержит подключение плагина и стилей слайдера, функционал для автоматического добавления классов, код для создания конкретных слайдеров. Изначально импортирован в `app.js`, закомментированн.
5. Файл **tippy.js** — содержит подключение плагина и стилей плагина подсказок. Изначально импортирован в `app.js`, закомментированн.
6. Файл **map.js** (в работе) — содержит шаблонный код для реализации функционала карт.
7. Файл **modules.js** — вспомогательный файл для работы ЧФ.
8. Папка **forms** — содержит файлы модулей для работы с формами
9. Папка **scroll** — содержит модули для работы с прокруткой сайта

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Установка и запуск шаблона. Подготовка к работе](#)

[Следующий пост: Работа со шрифтами. Локальные и иконочные шрифты. Подключение из Google Fonts ->](#)

Работа со шрифтами. Локальные и иконочные шрифты. Подключение из Google Fonts

Категории: **Документация «Чертоги Фрилансера» v3.0.0**

В стартовом шаблоне «Чертоги Фрилансера» v3.0.0 (далее ЧФ) весь процесс подключения локальных и иконочных шрифтов максимально автоматизирован. Исходные файлы шрифтов конвертируются с современные форматы .WOFF и .WOFF2, а также производится запись подключения шрифтов в файл стилей, включая значение font-weight на основе имени файла.

Как ЧФ обрабатывает шрифты?

При запуске шаблона в любом режиме, ЧФ проверит, есть ли файлы шрифтов в форматах .TTF и/или .OTF в папке src/fonts.

Далее происходит три этапа конвертации:

1. Файлы .OTF (если они есть) конвертируются в .TTF и сохраняются в папке с исходниками src/fonts.
2. Файлы .TTF конвертируются в .WOFF и записываются в папку с результатом (dist/fonts)
3. Файлы .TTF конвертируются в .WOFF2 и записываются в папку с результатом (dist/fonts)

После конвертации ЧФ проверит наличие файла стилей scss/fonts/fonts.scss и, **если его нет**, запишет в него конструкции @font-

face для всех файлов, включая значение font-weight основанное на имени файла.

Если файл **scss/fonts/fonts.scss** уже существует, данные не перезапишутся. Это сделано для того, что если нам придется внести изменения в файл **scss/fonts/fonts.scss** после работы ЧФ, эти изменения не затерлись.

В каких случаях нам потребуется внести изменения в файл **scss/fonts/fonts.scss в ручную?** Дело в том, что значение font-weight основывается на имени файла шрифта, то есть если файлы называется Roboto-bold, то значение font-weight будет 700. Но если файл будет назван без отделения начертания, например RobotoBold, то адекватное определение не удастся и будет записано значение по умолчанию (400). И вот в этих случаях нам необходимо отредактировать файл указав адекватные значения.

Если мы хотим перезаписать данные в файле **scss/fonts/fonts.scss**, нам следует его удалить и перезапустить систему, или запустить команду **npm run fonts** (система сама удалит файл и создаст новый)

Как подключить локальные файлы шрифтов:

1. Скачать или получить от дизайнера/заказчика файлы шрифтов в формате .TTF и/или .OTF и положить их в папку **src/fonts**
2. Раскомментировать подключение файла **fonts.scss** (строка **fonts/fonts**) в файле **scss/style.scss**
3. Указать семейство шрифта по умолчанию в переменной **\$fontFamily** в файле **scss/style.scss**
4. Запустить ЧФ в любом режиме

При необходимости, вы можете отредактировать созданный и заполненный файл **scss/fonts/fonts.scss**

При необходимости пересоздать данные в файле стилей **scss/fonts/fonts.scss** полностью, следует его удалить и перезапустить систему, или запустить команду **npm run fonts** (система сама удалит файл и создаст новый)

Как подключить локальные файлы иконочных шрифтов:

1. Тем или иным способом создать файл иконочного шрифта в формате .TTF и/или .OTF и положить его в папку **src/fonts** (идет работа над автоматизацией этого процесса)
2. Отредактировать файл **scss/fonts/icons.scss** внеся классы для иконок, а также убедиться в том что имя шрифта (font-family) для SCSS-шаблона **%ic {}** совпадает с тем что указан в файле **scss/fonts/fonts.scss**
3. Раскомментировать подключение файла **icons.scss** (строка **fonts/icons**) в файле **scss/style.scss**
4. Запустить ЧФ в любом режиме

Как подключить шрифты из Google Fonts:

Самый простой способ — это воспользоваться плагином для VS Code **Google Fonts**

1. Нажимаем F1 и ищем плагин Google Fonts
2. Если хотим подключить шрифт отдельным тегом link в HTML файл, выбираем **Google Fonts: insert <link>**
3. Если хотим подключить шрифт в файл стилей (обычно это **scss/style.scss**), выбираем **Google Fonts: insert CSS @import**
4. Из появившегося списка выбираем нужный шрифт, можно воспользоваться поиском
5. После вставки строки подключения следует отредактировать строку оставив только нужные начертания шрифта
6. Также следует добавить к строке подключения флаг **&display=swap**

Пример подключения шрифта Montserrat.

```
@import url(https://fonts.googleapis.com/css?
family=Montserrat:400,500,800&display=swap);
```

Шрифты подключенные из Google Fonts не должны попадать в папку с результатом (dist), они подгружаются с сервера Google.

Также, стоит отметить, что получить строку для подключения шрифта вы можете и без плагина на сайте [Google Fonts](#).

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Архитектура шаблона. Файлы и папки](#)

[Следующий пост: STYLE.SCSS — настройка адаптивной сетки, шрифтов, подключение дочерних файлов ->](#)

STYLE.SCSS — настройка адаптивной сетки, шрифтов, подключение дочерних файлов

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Файл **scss/style.scss** является основным файлом стилей в [стартовом шаблоне «Чертоги Фрилансера» v3.0.0](#). (далее ЧФ). Обычно, стили проекта тут не пишут, файл выполняет роль материнского файла куда подключаются отдельные файлы страниц, модулей и т.д.

Тут же подключаются шрифты и расположены основные переменные для настройки семейства и размера шрифта по умолчанию, цветов, адаптивной сетки, корректной работы миксинов и так далее...

Настройка семейства, размера и цвета шрифта по умолчанию

Для того чтобы работать со шрифтами, сперва нужно их подключить. Все действия подробно описаны в [этой статье](#).

После выполнения действий по подключению шрифтов, необходимо указать значения для следующих переменных:

\$fontFamily — имя семейства шрифта по умолчанию. Указываем имя основного шрифта в проекте.

```
$fontFamily: "Montserrat";
```


\$fontSize — размер шрифта по умолчанию. Этой переменной присвоена одна из SCSS-функций шаблона, она выполняет перевод пикселей в REM. Соответственно, в эту функцию следует передать значение размера шрифта по умолчанию в пикселях (только число без px).

```
$fontSize: rem(14);
```

\$mainColor — цвет шрифта по умолчанию. Указываем цветовой код.

```
$mainColor: #000;
```

Настройка адаптивной сетки

В ЧФ есть возможность настроить ограничивающий контейнер на работу как с отзывчивой адаптивной версткой, так и с версткой по брейкпоинтам.

Перед началом работ, настраиваем следующие переменные:

\$minWidth — минимальная ширина вьюпорта (экрана), поддерживаемая проектом. Обычно это 320px, но, с отмиранием старых устройств, это значение можно менять на любое нужное, указываем только число без px:

```
$minWidth: 320;
```

\$maxWidth — ширина всего макета (полотна), не путать с шириной ограничивающего контейнера. Как правило, дизайнеры предоставляют макеты шириной 1920 или 1440 пикселей, но это значение может быть любым. Меряем макет и указываем только число без px:

```
$maxWidth: 1920;
```

\$maxWidthContainer — ширина ограничивающего контейнера. Собственно, это ширина контента в макете дизайна. Меряем макет и

указываем только число без px:

```
$maxWidthContainer: 1170;
```

Если в макете нет ограничения у контента, то есть контент расположен на всю ширину полотна (с отступами), то следует указать значение **0** (ноль):

```
$maxWidthContainer: 0;
```

\$containerPadding — общий отступ (сумма отступов слева и справа) у ограничивающего контейнера. Указываем только число без px:

```
$containerPadding: 30;
```

Если отступов нет, либо вы хотите использовать адаптивное свойство, следует указать **0** (ноль):

```
$containerPadding: 0;
```

\$containerWidth — ширина срабатывания первого брейкпоинта. Собственно, это сумма ширин ограничивающего контейнера и его отступов. Как правило, менять тут ничего не нужно.

```
$containerWidth: $maxWidthContainer + $containerPadding;
```

Вышеуказанные переменные влияют и на функционал отзывчивого свойства, который описан в отдельной статье

Настройка брейкпоинтов

Переменным брейкпоинтов присвоена одна из SCSS-функций шаблона **em()**, она выполняют перевод пикселей в EM.

\$pc — ПК, ноутбуки, некоторые планшеты в горизонтальном положении. Обычно, тут указывается переменная **\$containerWidth**:

```
$pc: em($containerWidth);
```

\$tablet — планшеты, некоторые телефоны в горизонтальном положении. Обычно, значение равно 991.98px:

```
$tablet: em(991.98);
```

\$mobile — большие телефоны. Обычно, значение равно 767.98px:

```
$mobile: em(767.98);
```

\$mobileSmall — маленькие телефоны. Обычно, значение равно 479.98px:

```
$mobileSmall: em(479.98);
```

Для быстрого вызова медиа запроса с нужным брейкпойнтом можно использовать **сниппеты** md1, md2, md3, md4. Или, для Mobile First, mmd1, mmd2, mmd3, mmd4

Настройка типа адаптива (поведения ограничивающего контейнера)

\$responsiveType — настройка типа адаптива (поведения ограничивающего контейнера):

- **1** — отзывчивость. У контейнера нет брейкпойнтов, он сужается вместе с браузером
- **2** — по брейкпойнтам. Контейнер меняет свою ширину по настроенным брейкпойнтам

```
$responsiveType: 1;
```

Ниже, в файле стилей **scss/style.scss** указан селектор ограничивающего контейнера и его стили, значения которых во многом состоят из настроенных выше переменных.

Стили ограничивающего контейнера будут применяться к любому элементу в классе которого есть строка «**__container**». Для удобства можно использовать снимок `cnt`

```
<div class="block__container">
  ...
</div>
```

Подключение дополнительных файлов стилей

В файл **scss/style.scss** уже подключены и можно подключать прочие файлы стилей. Порядок подключения имеет значение!

@use «sass:math»; — подключает SASS-модуль математических вычислений. Теперь мы можем использовать деление с помощью `math.div(число, число)`.

@import «base/mixins»; — подключение используемых в ЧФ миксинов. Файл `scss/base/mixins.scss`.

@import «base/null»; — подключение обнуляющих стилей. Файл `scss/base/null.scss`.

@import «base»; — подключение общего файла базовых стилей модулей ЧФ, SASS-шаблонов (заготовок) и вспомогательных классов. Файл `scss/base.scss`. Для подключения/отключения конкретных стилей смотри `scss/base.scss`.

@import «common»; — подключение файла стилей общих элементов конкретного проекта. Файл `scss/common.scss` (изначально пуст).

@import «header»; , **@import «footer»;** — подключение стилей отдельных блоков (изначально `scss/header.scss` и `scss/footer.scss`). Вы можете дополнять список подключением своих файлов

@import «home»; — подключение стилей отдельных страниц (изначально `scss/home.scss`). Вы можете дополнять список подключением своих файлов

Селекторы и стили

Изначально в файле стилей **scss/style.scss** есть ряд SCSS-селекторов:

body {} — стили основного тега `<body>` часть из которых описана в файле обнуления `scss/base/null.scss`. Также добавлена подготовка для появления у тега `<html>` двух классов:

- **lock** — блокировка скrolла. Для этого уже написаны соответствующие CSS-стили
- **loaded** — сайт загружен. По этому классу мы можем влиять на `<body>` для отображения контента после полной загрузки данных

.wrapper {} — обертка всего контента на странице. Для неё написаны стили прижатия подвала к низу страницы, важный параметр `overflow: hidden`; который не даст появиться горизонтальному скроллу страницы, а также решение проблемы для слайдеров внутри дочерних flex-элементов.

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Работа со шрифтами. Локальные и иконочные шрифты. Подключение из Google Fonts](#)

[Следующий пост: SCSS-Миксин «Отзывчивое \(адаптивное\) свойство» ->](#)

SCSS-Миксин «Отзывчивое (адаптивное) свойство»

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

С помощью SCSS-миксина «Отзывчивое (адаптивное) свойство» можно отзывчиво (в зависимости ширины экрана) изменять значение того или иного CSS-свойства от начального значения, на определенной ширине экрана, до конечного значения на другой ширине экрана. Можно указывать произвольные промежутки ширин. Также, существует несколько режимов поведения миксина вне указанных промежутках.

Миксин представлен в двух реализациях — `clamp()` и `calc()`. `Clamp` работает быстрее, но если возникают проблемы с поддержкой браузерами миксин автоматически переключится на `calc()`.

Принудительно использовать `calc()` можно изменив код миксина в файле `scss/base/mixins.scss`

Использование миксина

Чтобы работать с миксином нужно в SCSS селекторе вызвать сниппет **av**:

Базовый режим работы миксина:

```
@include adaptiveValue("свойство", начальное значение, конечное значение);
```

Где:

1. **свойство** — CSS-свойство, значение которого нужно адаптировать. Можно указать любое свойство значение которого указывается в

цифрах.

2. **начальное значение** — стартовое значение свойства в пикселях, пишем число без px. Обычно указывается по макету.
3. **конечное значение** — финальное значение свойства в пикселях, пишем число без px. Значение, к которому мы хотим прийти на меньших ширинах экрана.

Примеры:

```
@include adaptiveValue("font-size", 50, 20);  
@include adaptiveValue("padding-top", 80, 10);  
...
```

Алгоритм работы миксина

Миксин работает на основе значений переменных **\$minWidth**, **\$maxWidth**, **\$maxWidthContainer**, **\$containerPadding** и **\$containerWidth** расположенных в блоке «Настройка адаптивной сетки» файла **scss/style.scss**

В итоге, по умолчанию, миксин будет работать следующим образом:

Если **\$maxWidthContainer** больше нуля, то значения свойства будут меняться в промежутке ширин от **\$containerWidth** до **\$minWidth**. То есть, по всей ширине ограничивающего контейнера.

При этом, если ширина экрана больше чем **\$containerWidth**, то значение свойства будет равно **начальному значению**. Если ширина экрана меньше чем **\$minWidth**, то значение свойства будет равно **конечному значению**.

Если **\$maxWidthContainer** равен нулю, то значения свойства будут меняться в промежутке ширин от **\$maxWidth** до **\$minWidth**. То есть, по всей ширине экрана (вьюпорта).

Дополнительные настройки и режимы работы

Миксин позволяет указать свой промежуток ширины внутри которого будет адаптироваться значение свойства.


```
@include adaptiveValue("свойство", начальное значение, конечное значение, ширина от, ширина до);
```

- **ширина от** — стартовая ширина меньше которой начнется адаптация, пишем число без px.
- **ширина до** — конечная ширина до которой будет адаптироваться значение свойства, пишем число без px.

Пример:

```
@include adaptiveValue("font-size", 50, 20, 800, 480);
```

При этом, работать миксин будет только в этом промежутке, а за его пределами значение свойства будет по умолчанию либо наследоваться от предков.

То есть, **font-size** из примера получит значение **50px** только на ширине **800px** и будет адаптироваться до **20px** до ширины **480px**. За пределами этого промежутка свойство унаследует значение по умолчанию.

Это поведение миксина мы тоже можем настроить указав режим работы.

```
@include adaptiveValue("свойство", начальное значение, конечное значение, ширина от, ширина до, режим работы);
```

режим работы — может принимать числовые значения **1**, **2** или **3**:

- **1** — Если ширина экрана больше чем **ширина от**, то значение свойства будет равно **начальному значению**. Если ширина экрана меньше чем **ширина до**, то значение свойства будет равно **конечному значению**.
- **2** — Если ширина экрана больше чем **ширина от**, то значение свойства будет равно **начальному значению**. Если ширина экрана меньше чем **ширина до**, то значение свойства будет по умолчанию либо наследоваться от предков.

- **3** — Если ширина экрана больше чем **ширина от**, то значение свойства будет по умолчанию либо наследоваться от предков. Если ширина экрана меньше чем **ширина до**, то значение свойства будет равно **конечному значению**.

Пример:

```
@include adaptiveValue("font-size", 50, 20, 800, 480, 1);
```

Также мы можем использовать несколько вызовов миксина с разными промежутками:

```
@include adaptiveValue("font-size", 50, 20, 800, 480, 2);  
@include adaptiveValue("font-size", 20, 10, 480, 320);
```

В примере произойдет следующее: значение font-size будет 50px, в промежутке ширин экрана от 800px до 480px, он будет отзывчиво адаптироваться от 50px до 20px. А в промежутке от 480px до 320px отзывчиво адаптироваться от 20px до 10px.

Расположение

Миксин **adaptiveValue** находится в файле **scss/base/mixins.scss**

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: STYLE.SCSS — настройка адаптивной сетки, шрифтов, подключение дочерних файлов](#)

[Следующий пост: Модуль меню «бургер» ->](#)

Модуль меню «бургер»

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Функционал реагирует на клик по кнопке меню «бургера» (элемент с классом `icon-menu`). При этом к тегу `html` добавляется класс **`menu-open`**, а также срабатывает блокировка прокрутки страницы (функция **`bodyLockToggle()`**). При повторном клике происходят обратные действия.

Подключение модуля

1. **[HTML]** Вызывать сниппет `menu`, отредактировать под задачу. Важно чтобы кнопка бургера оставалась с классом `icon-menu`
2. **[JS]** В файле **`js/app.js`** раскомментировать строку **`flsFunctions.menuInit();`**
3. **[SCSS]** По классу **`menu-open`** верстаем открытие меню. В файле **`scss/header.scss`** есть закомментированный стиль кнопки бургера.

Расположение и дополнительные данные

Функционал находится в **`js/files/functions.js`**. Название функции **`menuInit()`**, дополнительные функции **`menuOpen()`** и **`menuClose()`** для открытия/закрытия меню из произвольного кода. Все функции импортируемые.

Дополнительные материалы

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: SCSS-Миксин «Отзывчивое \(адаптивное\) свойство»](#)

[Следующий пост: Модуль «Рорир». Всплывающие \(модальные\) окна ->](#)

Модуль «Рорир». Всплывающие (модальные) окна

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Данный функционал добавляет возможность использовать всплывающие окна. Работа модуля заключается в следующем: пользователь нажимает на указанный элемент (по умолчанию это атрибут с указанным селектором **data-popup='selector'**). При этом к тегу **body** добавляется класс **popup-show**. Также блокируется прокрутка страницы (можно отключить), фокусировка элементов «перелетает» на рорир, с запоминанием предыдущего сфокусированного элемента на странице. Заккрытие рорир происходит при клике на кнопку закрытия (по умолчанию элемент с атрибутом **data-close**), по клику на «пустом месте» (не на рорир), по нажатию кнопки **ESC**.

Подключение функционала

1. **[HTML]** Подключить файл **_popup.htm**, после оболочки **wrapper**, в HTML-файл страницы (уже подключен в **home.html**)
2. **[SCSS]** Подключить файл **scss/base/popup.scss** в файл **scss/base.scss** — раскомментировать строку **@import 'base/popup';**
3. **[JS]** Подключить файл **js/libs/popup.js** в файле **js/app.js** — раскомментировать строку **import './libs/popup.js'**

Использование функционала

Для того, чтобы вызвать попап, необходимо на странице ввести объект с дата-атрибутом, в котором указан селектор (класс или id) всплывающего окна, на которое ссылаемся:

```
<a href="#" data-popup="#popup" class="link">Я открываю попап</a>
```

Далее открыть файл **html/_popup.htm**, раскомментировать HTML-код подготовки попапа, указать селектор (id или класс) по которому вызывается попап, изменить код под свои нужды.

В примере попап вызывается по id popup:

```
<div id="popup" aria-hidden="true" class="popup">
  <div class="popup__wrapper">
    <div class="popup__content">
      <button data-close type="button"
class="popup__close">Закреть</button>
      <div class="popup__text">

    </div>
  </div>
</div>
```

Открытие YouTube видео в попапе

Для того чтобы открыть видеоролик в попапе, следует добавить блоку попапа атрибут **data-youtube**, а в качестве значения указать код ролика. Также следует указать атрибут **data-youtube-place** для объекта в котором хотим вывести ролик:

```
<div data-youtube="OsF9H7yRnvU" id="popup" aria-hidden="true"
class="popup">
  <div class="popup__wrapper">
    <div class="popup__content">
      <button data-close type="button"
class="popup__close">Закреть</button>
      <div data-youtube-place class="popup__text">

    </div>
  </div>
</div>
```

Стили попапа можно писать и изменять в файле **scss/base/popup.scss**

Открытие попапа по хешу

Для того чтобы открыть попап при открытии страницы, добавляем к адресу хеш с именем селектора попапа

```
https://template.fl.s.guru/index.html#popup
```

Методы и события

Методы

Работать с попапом из любого места можно импортировав переменную **flsModules**:

```
import { flsModules } from "./modules.js";
```

Далее обратится к классу **popup** и работать с методом, например **open()**

```
flsModules.popup.open( '#popup' )
```

где **#popup** селектор попапа

События

В классе попапов существуют ряд событий:

1. **beforePopupOpen** — сработает перед открытием попапа
2. **afterPopupOpen** — сработает после открытия попапа
3. **beforePopupClose** — сработает перед закрытием попапа
4. **afterPopupClose** — сработает после закрытия попапа

Чтобы работать с событием вешаем прослушку на document

```
document.addEventListener("afterPopupOpen", function (e) {  
  // Попап  
  const currentPopup = e.detail.popup;  
});
```


Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Модуль меню «бургер»](#)

[Следующий пост: Модуль «Динамический адаптив» ->](#)

Модуль «Динамический адаптив»

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Во время работы над адаптивом сайта, нам то и дело приходится изменять внешний вид объекта. В некоторых случаях нам необходимо изменить порядок элементов, когда при определенном разрешении экрана некоторый блок должен находиться в совершенно другом месте структуры. Особенно часто это требуется при адаптации шапки сайта когда необходимо перенести блок с контактами в меню-бургер.

Функционал динамического адаптива перемещает необходимый блок (на определенном разрешении) в другой блок. Перемещение отображается в разметке HTML.

Подключение функционала

[JS] В файле **js/app.js** раскомментировать строку **import «./libs/dynamic_adapt.js»**

Использование функционала

[HTML] В блок который нужно переместить добавляем атрибут **data-da** со значениями атрибута указанными ниже:



Расположение и дополнительные данные

Функционал находится в `js/libs/dynamic_adapt.js`. Название функции **DynamicAdapt(type)**.

У модуля есть ограничения, например, если перекидывать несколько объектов одновременно в один и тот же блок, может возникнуть путаница с порядком блоков.

Дополнительные материалы

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Модуль «Рорир». Всплывающие \(модальные\) окна](#)

[Следующий пост: Модуль «Прокрутка к нужному блоку». Плавная навигация по странице. ->](#)

Модуль «Прокрутка к нужному блоку». Плавная навигация по странице.

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

В данном модуле реализован плавный скролл после клика по ссылке (кнопке) до нужного блока на странице. Если в адресе страницы будет передан хеш и блок с таким селектором будет найден на странице, прокрутка к блоку произойдет автоматически. Также, модуль позволяет построить навигацию по странице с добавлением класса текущему пункту навигации (пункту меню) при скролле к определенному блоку.

Подключение функционала

[JS] В файле **js/app.js** раскомментировать строку **flsScroll.pageNavigation();**

Использование функционала

[HTML] К элементам навигации (пунктам меню), либо к произвольному объекту, добавляем HTML-атрибут **data-goto** а в качестве значения указываем CSS селектор блока до которого нужно прокрутить:

```
<a href="#" data-goto=".имя класса блока" class="link">Пункт навигации</a>
```

```
<a href="#" data-goto="#id блока" class="link">Пункт навигации</a>
```

Если нужно чтобы скролл учитывал шапку (не докручивал на высоту шапки, используется при фиксированных шапках) нужно добавить к объекту навигации атрибут **data-goto-header**:

```
<a href="#" data-goto-header data-goto=".имя класса блока"
class="link">Пункт навигации</a>
```

Если нужно чтобы скролл не докручивал до блока на указанную высоту необходимо добавить к объекту навигации атрибут **data-goto-top** а в качестве значения указать число — необходимую высоту:

```
<a href="#" data-goto-top="30" data-goto="#id блока"
class="link">Пункт навигации</a>
```

data-goto-top можно совмещать с **data-goto-header**, тогда значение **data-goto-top** добавится к высоте шапки.

Добавление класса к текущему пункту навигации

Для включения функционала добавления класса подключаем модуль наблюдатель:

[JS] В файле **js/app.js** раскомментировать строку **import './libs/watcher.js'**

[HTML] Для блоков к которым прокручивается страница добавляем атрибут **data-watch** со значением **navigator**:

```
<a href="#" data-goto=".some-section" class="link">Пункт
навигации</a>
...
<section data-watch="navigator" class="some-section"></section>
```

После этого, при прокрутке к блоку (объекту) навигации, к соответствующему пункту навигации будет добавлен класс **_navigator-active**

Прокрутка к нужному блоку по хешу (при открытии страницы)

Для того чтобы прокрутить страницу к нужному блоку при открытии страницы необходимо добавить к адресу хеш содержащий **имя класса нужного блока**.

Пример адресной строки и нужного блока:

```
https://template.fls.guru/index.html#some-section
```

```
<section class="some-section">
...
</section>
```

Добавление функционала, плавная прокрутка на iOS

По умолчанию, прокрутка выполняется методом `scrollTo()` с параметром `behavior: «smooth»` без применения дополнительных плагинов. Но это ограничивает функционал этого модуля — нельзя указать скорость прокрутки, а также могут возникнуть проблемы в некоторых версиях браузеров на iOS. Для решения всех проблем, можно подключить дополнительный плагин `SmoothScroll`, сделать это можно в файле **`js/files/scroll/gotoblock.js`** раскомментировав строку **`import SmoothScroll from 'smooth-scroll'`**; дальнейшее переключение прокрутки на плагин произойдет автоматически.

При работе с плагином появляется возможность указать скорость прокрутки, для этого элементу навигации нужно добавить атрибут **`data-goto-speed`** и указать число означающее количество миллисекунд за которые совершится прокрутка (1000 = 1 секунда), по умолчанию 500.

```
<a href="#" data-goto-speed="1000" data-goto=".some-section"
class="link">Пункт навигации</a>
```

Расположение и дополнительные данные

Функционал находится в **`js/files/scroll/scroll.js`**. Название функции **`pageNavigation()`**. Вспомогательный модуль прокрутки **`gotoblock`** находится в **`js/files/scroll/gotoblock.js`**. Модуль наблюдатель находится в файле **`js/libs/watcher.js`**.

Если в момент клика на пункт навигации было открыто меню «бургер», то оно закроется автоматически.

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Модуль «Динамический адаптив»](#)

[Следующий пост: Модуль добавления классов к шапке при прокрутке
страницы ->](#)

Модуль добавления классов к шапке при прокрутке страницы

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Функционал позволяет добавлять класс к тегу header с классом header при прокрутке страницы вниз, а также другой класс при остановке прокрутки. Таким образом, можно добиться эффекта когда при самом скролле вниз шапка не видна, но стоит остановить скролл, как шапка плавно появляется вверху страницы (становится фиксированной). При обратной прокрутке вверх шапка так же остается видна.

Класс **_header-scroll** добавляется к шапке при скролле вниз (через указанное кол-во пикселей).

При остановке скролла добавляется класс **_header-show**.

Подключение функционала

[JS] В файле **js/app.js** раскомментировать строку **flsScroll.headerScroll();**

Использование функционала

[HTML] К тегу header, добавляем HTML-атрибут **data-scroll**, в значении атрибута указываем через какое кол-во прокрученных вниз пикселей нам необходимо добавить класс к header (обычно по высоте шапки, по умолчанию 1px).

```
<header data-scroll="120" class="header">
...
</header>
```

Теперь как только пользователь прокрутит вниз указанные выше 120px к header добавится технический класс **_header-scroll**. Этот класс будет присутствовать до тех пор, пока пользователь не вернется на самый вверх (не доходя 120px).

Результат работы:

```
<header data-scroll="120" class="header _header-scroll">
...
</header>
```

[HTML] Далее к тегу header, добавляем еще один HTML-атрибут **data-scroll-show**. Как только пользователь остановит прокрутку к тегу header, через определенное время, добавится еще один технический класс **_header-show**. Этот класс исчезает только в моменте прокрутке вниз. При прокрутке вверх класс не исчезает.

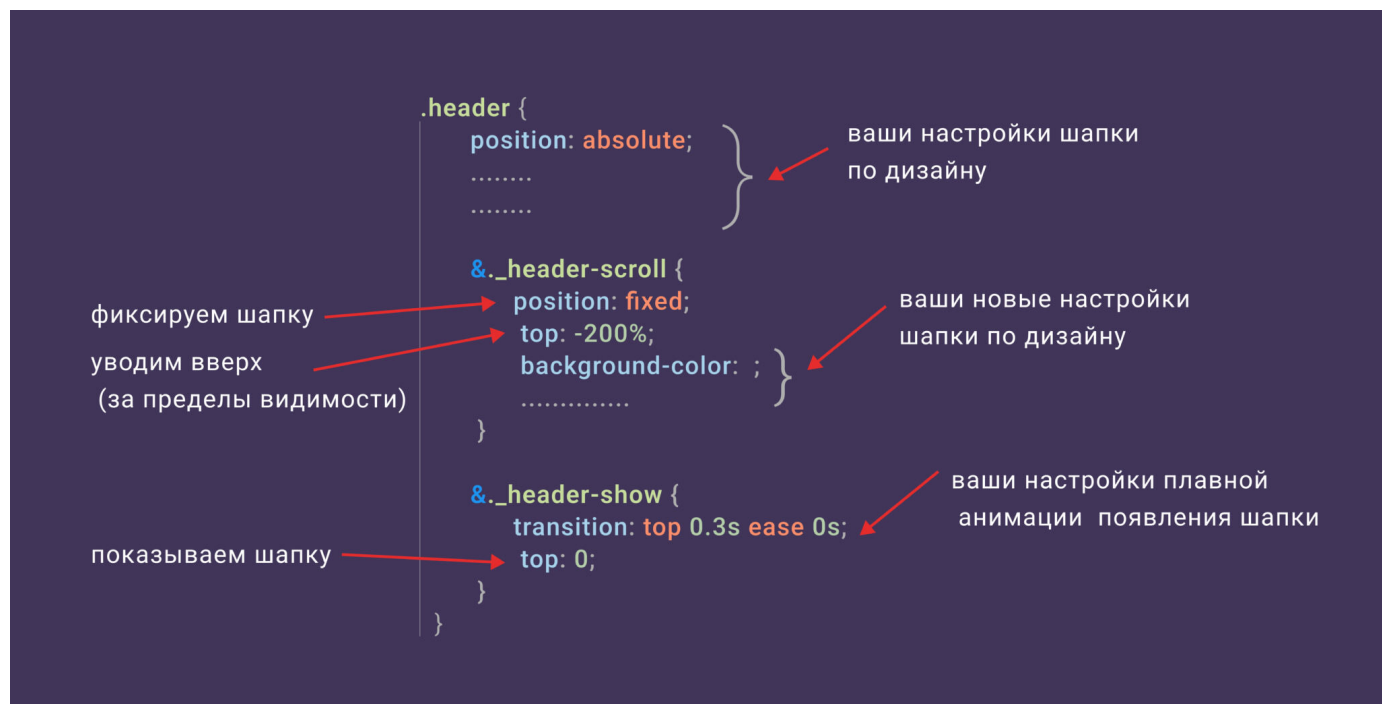
Результат работы:

```
<header data-scroll="120" data-scroll-show class="header _header-
scroll _header-show">
...
</header>
```

Время задержки добавления класса **_header-show** можно менять. Для это следует указать значение атрибута **data-scroll-show** в миллисекундах (по умолчанию 500)

```
<header data-scroll="120" data-scroll-show="1000" class="header
_header-scroll _header-show">
...
</header>
```

[SCSS] Теперь осталось отредактировать свойства этих подключенных классов в scss. Например:



Расположение и дополнительные данные

Функционал находится в `js/files/scroll/scroll.js`. Название функции **headerScroll()**

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Модуль «Прокрутка к нужному блоку». Плавная навигация по странице.](#)

[Следующий пост: Модуль «Наблюдатель» за появлением элементов при прокрутке страницы \(скролле\) ->](#)

Модуль «Наблюдатель» за появлением элементов при прокрутке страницы (скролле)

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Модуль «Наблюдатель» можно использовать для решения самых разных задач: анимация элементов при скролле, подсветка активного пункта меню (используется в модуле «Прокрутка к нужному блоку») и многих других.

Суть работы наблюдателя заключается в добавлении класса **_watcher-view** элементу в момент его появления в вьюпорте (экране) при скролле а также при открытии страницы. И, соответственно, убран при уходе объекта из вьюпорта.

Подключение модуля

[HTML] Для объекта, за которым нужно установить наблюдение, следует добавить атрибут **data-watch**

```
<div data-watch class="block">
  ...
</div>
```

[JS] В файле **js/app.js** раскомментировать строку **import** **'./libs/watcher.js'**

Дополнительные настройки:

- **data-watch-root='селектор'** — селектор родителя внутри которого наблюдать за объектом. По умолчанию `<body>`
- **data-watch-margin='значение'** — отступ от родителя. Указываем значение в PX или в %
- **data-watch-threshold='значение'** — процент показа объекта для срабатывания. Где 1 = 100% показ объекта. Указываем только целые или десятичные числа, по умолчанию 0. Может содержать массив значений через запятую.
- **data-watch-once** — наблюдать только один раз. То есть класс к объекту добавится только один раз и не будет убран при уходе объекта из вьюпорта.

Пример — класс добавится только один раз, при появлении объекта на 50% его высоты:

```
<div data-watch-threshold="0.5" data-watch-once data-watch
class="block">
  ...
</div>
```

События

После каждом срабатывании наблюдателя, возникает событие **watcherCallback**, его можно отловить в любой части кода:

```
document.addEventListener("watcherCallback", function (e) {
  // Полная информация от наблюдателя
  const entry = e.detail.entry;
  // Наблюдаемый объект
  const targetElement = entry.target;
});
```

Расположение и дополнительные данные

Функционал находится в файле **js/libs/watcher.js**. Название класса **ScrollWatcher**. Модуль построен на основе [Intersection Observer API](#).

Модуль снабжен системой **FLS** и будет сообщать о своих действиях в консоль браузера

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Модуль добавления классов к шапке при прокрутке страницы](#)

[Следующий пост: Модуль «Показать ещё» ->](#)

Модуль «Показать ещё»

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Модуль «Показать ещё» позволяет изначально скрыть часть текста или элементов списка показывая только указанную высоту либо количество элементов. Есть возможность включать функционал на определенной ширине экрана (брейкпоинте).

Подключение функционала

[HTML] В нужном месте вызвать [сниппет showmore](#) (классы заменить на нужные). Либо вручную создать структуру где для оболочки добавлен атрибут **data-showmore**, для дочернего элемента **data-showmore-content** и для кнопки **data-showmore-button**. Кнопку изначально нужно скрыть добавив атрибут **hidden** и добавить два тега `` с текстом показа и скрытия контента:

```
<div data-showmore class="block">
  <div data-showmore-content class="block__content"></div>
  <button hidden data-showmore-button type="button"
class="block__more">
    <span>Показать еще</span>
    <span>Скрыть</span>
  </button>
</div>
```

[JS] В файле **js/app.js** раскомментировать строку **flsFunctions.showMore();**

[SCSS] Раскомментировать строку **@import «base/showmore»;** в файле **src/scss/base.scss** — это подключит базовые стили, отредактировать под свои нужды

Использование функционала

В элемент с атрибутом **data-showmore-content** добавляем текст и прочий контент, либо, если это список (UL/OL) элементы списка (LI).

В зависимости от того, какой контент используется (текст или элементы списка) указываем значение для атрибута **data-showmore**:

- **size** — ограничение по высоте блока (*по умолчанию*)
- **items** — ограничение количества выводимых элементов списка

```
<div data-showmore="items" class="block">
  <ul data-showmore-content class="block__content">
    <li>Пункт №1</li>
    <li>Пункт №2</li>
    <li>Пункт №3</li>
    <li>Пункт №4</li>
    <li>Пункт №5</li>
  </ul>
  <button hidden data-showmore-button type="button"
class="block__more">
    <span>Показать еще</span>
    <span>Скрыть</span>
  </button>
</div>
```

В зависимости от того, какой тип выбран, указываем значение для атрибута **data-showmore-content** :

- **Высота блока в пикселях** (*число без px, по умолчанию 150*)
- **Количество выводимых элементов списка** (*число, по умолчанию 3*)

```
<div data-showmore class="block">
  <div data-showmore-content="200" class="block__content">
    Lorem ipsum dolor sit amet consectetur, adipisicing elit.
    Blanditiis explicabo
    voluptates magni culpa, perferendis vel quam consequuntur
    possimus,
    vero placeat quo enim obcaecati quas, veritatis magnam non.
    Architecto,
    porro voluptatum?
```



```

</div>
<button hidden data-showmore-button type="button"
class="block__more">
  <span>Показать еще</span>
  <span>Скрыть</span>
</button>
</div>

```

Если контента будет меньше чем указанное ограничение, кнопка «Показать ещё» **не будет показана**. В противном случае, контент ограничится по высоте либо по количеству элементов и при клике на кнопку будет показан полностью, также, к элементу с атрибутом **data-showmore** добавится класс **_showmore-active** (первый спан в кнопке будет скрыт а второй показан). Повторный клик вернет ограничение.

Есть возможность управлять скоростью разворачивания контента, для этого следует указать значение атрибуту **data-showmore-button** в миллисекундах (по умолчанию 500):

```

<div data-showmore class="block">
  <div data-showmore-content="200" class="block__content">
    Lorem ipsum dolor sit amet consectetur, adipisicing elit.
    Blanditiis explicabo
    voluptates magni culpa, perferendis vel quam consequuntur
    possimus,
    vero placeat quo enim obcaecati quas, veritatis magnam non.
    Architecto,
    porro voluptatum?
  </div>
  <button hidden data-showmore-button="1000" type="button"
class="block__more">
    <span>Показать еще</span>
    <span>Скрыть</span>
  </button>
</div>

```

Включение функционала на определенной ширине экрана

Для того чтобы использовать функционал на определенной ширине экрана, к объекту с атрибутом **data-showmore** добавляем атрибут **data-**

showmore-media где, через запятую, указываем нужную ширину, а также тип:

- **max** (по умолчанию) — функционал включится на ширине меньшей чем указанная
- **min** — функционал включится на ширине большей чем указанная

```
<div data-showmore data-showmore-media="768,min" class="block">
  <div data-showmore-content="200" class="block__content">
    Lorem ipsum dolor sit amet consectetur, adipisicing elit.
    Blanditiis explicabo
    voluptates magni culpa, perferendis vel quam consequuntur
    possimus,
    vero placeat quo enim obcaecati quas, veritatis magnam non.
    Architecto,
    porro voluptatum?
  </div>
  <button hidden data-showmore-button="1000" type="button"
class="block__more">
    <span>Показать еще</span>
    <span>Скрыть</span>
  </button>
</div>
```

Расположение и дополнительные данные

Функционал находится в **js/files/functions.js**. Название функции **showMore()**

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Модуль «Наблюдатель» за появлением элементов при прокрутке страницы \(скролле\)](#)

[Следующий пост: Модуль «Табы» ->](#)

Модуль «Табы»

Категории: **Документация «Чертоги Фрилансера» v3.0.0**

Табы — это заголовки и соответствующие им блоки. Как правило, по умолчанию открыт только один блок остальные скрыты. При клике на заголовок показывается соответствующий ему блок.

Основные возможности

1. Использование множества блоков с табами
2. Открытие нужного таба по хешу
3. Превращение табов в спойлеры на указанной ширине экрана (удобно для адаптива)
4. Возможность анимированного открытия табов
5. Семантика

Подключение функционала

[HTML] В нужном месте вызвать **сниппет tabs** (классы можно заменить на нужные). Либо вручную создать структуру с соответствующими дата-атрибутами. Обратите внимание, что добавление класса **_tab-active** для заголовка таба сделает таб активным (открытым)

Пример блока с тремя табами:

```
<div data-tabs class="tabs">
  <nav data-tabs-titles class="tabs__navigation">
    <button type="button" class="tabs__title _tab-active">Таб
№1</button>
    <button type="button" class="tabs__title">Таб №2</button>
    <button type="button" class="tabs__title">Таб №3</button>
  </nav>
  <div data-tabs-body class="tabs__content">
```

```
<div class="tabs__body">
  Содержимое первого таба
</div>
<div class="tabs__body">
  Содержимое второго таба
</div>
<div class="tabs__body">
  Содержимое третьего таба
</div>
</div>
</div>
```

[JS] В файле **js/app.js** раскомментировать строку **flsFunctions.tabs();**
[SCSS] (не обязательно) Если вы хотите сразу посмотреть на работу табов и оставили классы предложенные сниппетом, вы можете раскомментировать строку **@import «base/tabs»;** в файле **src/scss/base.scss** — это подключит базовые стили, их можно отредактировать под свои нужды.

Использование функционала

Превращение табов в спойлеры

Для того, чтобы табы превращались в спойлеры, необходимо для элемента с атрибутом **data-tabs** указать значение ширины экрана ниже которой произойдет превращение:

```
<div data-tabs="768" class="tabs">
  ...
</div>
```

В момент превращения, к объекту с атрибутом **data-tabs** добавится класс **_tab-spoller**, по которому можно изменить стили для нового представления табов-спойлеров.

Открытие нужного таба по хешу

Если есть необходимость открывать конкретный таб в конкретном блоке табов при открытии страницы по хешу, необходимо, для элемента с атрибутом **data-tabs**, добавить атрибут **data-tabs-hash**:

```
<div data-tabs data-tabs-hash class="tabs">
...
</div>
```

Теперь, при клике на заголовки табов, к адресу страницы будет добавляться хеш вида: #tab-0-1, где 0 — это идентификатор блока с табами, а 1 — идентификатор таба в этом блоке.

Соответственно, перейдя на страницу с хешем #tab-0-1 откроется второй таб в первом блоке с табами. При #tab-2-0 откроется первый таб в третьем блоке с табами и т.д.

Анимация при открытии таба

Для того, чтобы табы открывались плавно, необходимо объекту с атрибутом **data-tabs**, добавить атрибут **data-tabs-animate**, а в качестве значения указать количество миллисекунд за которые откроется таб (по умолчанию 500).

```
<div data-tabs data-tabs-animate="1000" class="tabs">
...
</div>
```

Расположение и дополнительные данные

Функционал находится в **js/files/functions.js**. Название функции **tabs()**

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[← Предыдущий пост: Модуль «Показать ещё»](#)

[Следующий пост: Модуль «Спойлеры» ->](#)



Модуль «Спойлеры»

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Спойлер — это заголовок, при клике на который под ним разворачивается некий контент.

Основные возможности

1. Использование множества блоков со спойлерами
2. Отключение/включение функционала на определенной ширине экрана
3. Функция «аккордеон», когда в блоке может быть открыт только один спойлер
4. Возможность анимированного открытия
5. Семантика

Подключение функционала

[HTML] В нужном месте вызвать сниппет **spollers** (классы можно заменить на нужные). Либо вручную создать структуру с соответствующими дата-атрибутами. Обратите внимание, что добавление класса **_spoller-active** для элемента с атрибутом **data-spoller** сделает спойлер активным (открытым).

Пример блока с двумя спойлерами:

```
<div data-spollers class="spollers">
  <div class="spollers__item">
    <button type="button" data-spoller class="spollers__title">
      Заголовок спойлера №1
    </button>
    <div class="spollers__body">Контент спойлера №1</div>
  </div>
```



```
<div class="spollers__item">
  <button type="button" data-spoller class="spollers__title">
    Заголовок спойлера №2
  </button>
  <div class="spollers__body">Контент спойлера №2</div>
</div>
</div>
```

[JS] В файле **js/app.js** раскомментировать строку **flsFunctions.spollers();**

[SCSS] (не обязательно) Если вы хотите сразу посмотреть на работу спойлеров и оставили классы предложенные сниппетом, вы можете раскомментировать строку **@import «base/spollers»;** в файле **src/scss/base.scss** — это подключит базовые стили, их можно отредактировать под свои нужды.

В момент инициализации (включения) функционала спойлера, контент будет скрыт, а к элементу с атрибутом **data-spollers** будет добавлен класс **_spoller-init**

Использование функционала

Отключение/включение функционала на определенной ширине экрана

Для того, чтобы отключить/включить функционал спойлера на определенной ширине экрана, необходимо для атрибута **data-spollers** через запятую указать нужную ширину экрана а также тип:

- **max** (по умолчанию) — функционал включится на ширине меньшей чем указанная
- **min** — функционал включится на ширине большей чем указанная

```
<div data-spollers="768,min" class="spollers">
  ...
</div>
```

Включение режима «аккордеон»

Для того, чтобы включить режим «аккордеон», необходимо для элемента с атрибутом **data-spollers** добавить атрибут **data-one-spoller**

```
<div data-spollers data-one-spoller class="spollers">
...
</div>
```

Теперь, при открытии спойлера, другой открытый спойлер в блоке будет закрываться

Управление скоростью анимации

Для того, чтобы управлять временем анимации открытия/закрытия спойлера, необходимо для элемента с атрибутом **data-spollers** добавить атрибут **data-spollers-speed**, а в качестве значения указать время анимации в миллисекундах (по умолчанию 500).

```
<div data-spollers data-spollers-speed="1000" class="spollers">
...
</div>
```

Расположение и дополнительные данные

Функционал находится в **js/files/functions.js**. Название функции **spollers()**

Дополнительные материалы

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Модуль «Табы»](#)

[Следующий пост: Модуль «Ленивая подгрузка» \(Lazy Loading\) ->](#)

Модуль «Ленивая подгрузка» (Lazy Loading)

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Модуль «Ленивая подгрузка» позволяет подгружать изображения, а также содержимое тегов `iframe` `video` `audio` только при доскроллинге к элементу. Это существенно повышает скорость загрузки страницы.

Подключение функционала

[HTML] Для подключения картинки с ленивой подгрузкой, в нужном месте следйет вызвать сниппет **imgl**, что выведет тег `` но вместо атрибута `src` будет атрибут `data-src`.

При выводе других тегов (`iframe` `video` `audio`) также следует заменить `src` на `data-src`.

```

```

Во время режима продакшн, система распознает `data-src` и изменит для подключаемого `webp`-файла атрибут на `data-srcset`

[JS] В файле **js/app.js** раскомментировать строку **import** `‘./files/scroll/lazyload.js’`;

Работа функционала

В момент доскроливания до объекта с атрибутом `data-src` либо `data-srcset` модуль подгрузит данные (переместит подключение в атрибут `src/srcset`), а также добавит к объекту класс **_lazy-loaded** и атрибут `data-ll-status` со значением `loaded`

```

```

Расположение и дополнительные данные

Модуль работает на основе плагина **vanilla-lazyload.js**. Подключение и настройки плагина находится в **js/files/scroll/lazyload.js**. Для более тонких настроек читайте [полную документацию плагина](#).

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Модуль «Спойлеры»](#)

[Следующий пост: Модуль слайдера «Swiper» ->](#)

Модуль слайдера «Swiper»

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

В шаблоне [«Чертоги Фрилансера» v3.0.0](#), далее ЧФ, есть подготовка к работе со [слайдером Swiper](#).

[HTML] Существует два сниппета для построения HTML-структуры слайдера:

1. **swiper** — строит минимальную структуру с уже добавленными классами слайдера (для более опытных)
2. **swiperfull** — строит полную структуру слайдера с добавлением всех возможных элементов управления (кнопки «влево/вправо», скролл, пагинация (булеты)) с уже добавленными классами слайдера. Весь код сопровождается комментариями (для новичков).

[JS] В файле **js/app.js** раскомментировать строку **import «./files/sliders.js»;**

В файле **js/files/sliders.js** выполняется подключение самого слайдера «Swiper» из NPM пакета (подключено по умолчанию)

```
import Swiper, { Navigation } from 'swiper';
```

При необходимости, можно подключить больше нужных модулей:

```
import Swiper, { Navigation, Pagination, Lazy, Autoplay } from 'swiper';
```

Полный список модулей — [тут](#)

Также, ниже по коду, есть пример-подготовка для создания конкретного слайдера (функция **initSliders()**;) Тут мы создаем и настраиваем нужные нам слайдеры, не забываем указывать модули для конкретного слайдера:

```
new Swiper('.swiper', {
  modules: [Navigation, Autoplay],
  ...
})
```

Информацию по настройке смотри в [документации](#) на сайте слайдера.

Инициализация слайдера(ов) **initSliders()**; происходит после полной загрузки страницы ниже по коду:

```
window.addEventListener("load", function (e) {
  // Запуск инициализации слайдеров
  initSliders();
});
```

[SCSS] По умолчанию, в файле **js/files/sliders.js** подключены базовые, минимально необходимые для работы, стили слайдера **import «.././scss/base/swiper.scss»;** (для более опытных). Также есть возможность подключить (раскомментирав строку) полные стили слайдера из файла **scss/libs/swiper.scss** или из пакета **import 'swiper/css';** (для начинающих)

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Модуль «Ленивая подгрузка» \(Lazy Loading\)](#)

[Следующий пост: Работа с формами и элементами форм ->](#)



Работа с формами и элементами форм

Категории: **Документация «Чертоги Фрилансера» v3.0.0**

В шаблоне **«Чертоги Фрилансера» v3.0.0** (далее ЧФ) реализован функционал помогающий решать различные рутинные задачи при работе с формами.

В этом документе описан общий функционал работы с полями, плейсхолдерами, валидацией, вариантами отправки форм и т.д. В отдельных разделах будет описан функционал конкретных элементов форм, таких как:

- Отправка писем на почту (PHPMailer)
- **Кастомизация элемента SELECT**
- Кастомизация (стилизация) элементов CHECKBOX и RADIO
- Модуль «Маски» для полей ввода
- Модуль «Звездный рейтинг»
- Модуль «Количество»
- Модуль «Range» (ползунок)

Работа с полями форм

Работа с полями форм подразумевает под собой следующий функционал:

1. Скрытие placeholder при фокусе
2. Добавление классов полю и его родителю при фокусе
3. Функционал «Показать пароль» для полей с типом password
4. Возможность моментальной валидации поля при потере фокуса

Для того чтобы подключить общий функционал для работы с полями форм необходимо раскомментировать функцию **flsForms.formFieldsInit({...})** в файле **js/app.js**:

```
flsForms.formFieldsInit({  
  viewPass: false,  
});
```

Передавать функции можно следующие параметры:

- **viewPass** — позволяет включить функционал «Показать пароль». `true` — включено, `false` — выключено (по умолчанию).

Работа с атрибутом **placeholder**

По умолчанию, после подключения **flsForms.formFieldsInit({...})**, значение добавленного полю HTML-атрибута **placeholder** копируется в созданный скриптом атрибут **data-placeholder**, что позволит скрывать плейсхолдер при фокусе на поле. Если нам нужно отключить скрывание для конкретного поля, ему следует добавить атрибут **data-placeholder-nohide**

Добавление классов

По умолчанию, после подключения **flsForms.formFieldsInit({...})**, при возникновении фокуса, к полю а также к его непосредственному родительскому элементу, добавится класс **_form-focus**. Если нам нужно отключить добавление классов для конкретного поля, ему следует добавить атрибут **data-no-focus-classes**

Валидация поля при потере фокуса

Для того чтобы скрипт начал вызывать **функционал валидации** поля в момент потери им фокуса, следует добавить полю атрибут **data-validate**.

Стоит добавить, что функционал работы с полями форм также удалит ошибку добавленную к полю валидатором при получении им фокуса. Подробнее о функционале валидации полей смотри далее в этом документе.

Функционал «Показать пароль»

Функционал «Показать пароль», позволяет отображать зашифрованное значение поля с типом password при клике на объект с классом содержащим строку «__viewpass» который находится вместе с полем в одном родителе:

```
<form class="form" action="#">
  <div class="form__line">
    <input autocomplete="off" type="password" name="form[]"
value="123456">
    <button class="form__viewpass" type="button">Показать
пароль</button>
  </div>
  <button type="submit" class="button">Отправить</button>
</form>
```

В момент клика на объект с классом содержащим строку «__viewpass» в нему добавляется класс **_viewpass-active**, а тип поля ввода меняется на text. Повторное нажатие выполнит обратные действия.

Напомню, для того чтобы включить функционал «Показать пароль» следует указать true для параметра viewPass при подключении функционала в файле **js/app.js**.

Валидация элементов форм

Для того чтобы элемент формы начал проходить валидацию, ему следует добавить атрибут **data-required**. Теперь при отправке формы (если включена валидация), а также если элементу добавлен атрибут data-validate, он будет проверяться на предмет заполнения.

Для того чтобы включить особые правила валидации поля, атрибуту следует добавить одно из значений:

- **email** — включит валидацию на ввод корректного E-mail
- идет работа над новыми пресетами

Пример:

```
<input data-required="email" type="text" name="form[]">
```

Если элемент заполнен не верно, к нему, а также к его родителю добавится класс **_form-error**. Если мы хотим дополнительно вывести произвольный текст ошибки, нам следует указать его в атрибуте **data-error** и добавить к полю:

```
<input data-error="Введен не верный E-mail" data-required="email" type="text" name="form[]">
```

Теперь, при возникновении ошибки валидации, под элементом добавится объект с классом **form__error** содержащий ваш текст ошибки.

Напомню, что функционал работы с полями удалит классы ошибок и объект с классом **form__error** при получении полем фокуса.

Отправка форм

Для включения функционала следует раскомментировать функцию **flsForms.formSubmit();** в файле **js/app.js**

Валидация элементов формы

По умолчанию, при отправке формы, поля отмеченные **data-required/data-required='...'** будут проходить валидацию. Для отключения валидации элементов конкретной формы, ей следует добавить атрибут **data-no-validate**

Режимы отправки формы

В ЧФ существует несколько режимов контроля отправки форм:

1. **Стандартная HTML отправка формы** (по умолчанию) — если валидация пройдена (была включена), форма отправится на указанный в атрибуте **action** адрес (с переходом страницы), методом указанным в атрибуте **method**.
2. **AJAX отправка формы** — если валидация пройдена (была включена), форма отправится AJAX запросом на указанный в атрибуте **action** адрес, методом указанным в атрибуте **method**. Страница не перезагрузится, все элементы формы вернутся к исходным значениям (очистка формы).

- 3. Режим имитации отправки формы** — если валидация пройдена (была включена), форма никуда не отправится, страница не перезагрузится, все элементы формы вернутся к исходным значениям (очистка формы). Используется для демонстрации работы дополнительных возможностей форм, таких как, например, показ попапа об успешной отправке.

Для включения режима **AJAX отправки** достаточно добавить форме атрибут **data-ajax**, а если нужен режим **имитации отправки** добавляем атрибут **data-dev**.

```
<form data-ajax class="form" method="POST" action="sendmail.php">
  ...
</form>
```

Прокрутка к элементу с ошибкой

Бывает, что форма очень большая и, при возникновении ошибки валидации, хорошо бы показать элемент с ошибкой пользователю. Для этих целей есть функционал «прокрутка к элементу с ошибкой». Для включения, достаточно добавить форме атрибут **data-goto-error**

```
<form data-goto-error class="form" action="#">
  ...
</form>
```

Показ попапа после отправки формы

Если необходимо, после отправки формы, показать попап добавляем к форме атрибут **data-popup-message** и в качестве значения указываем селектор попапа

Только для режимов **data-ajax** или **data-dev**. **Функционал попапов** также должен быть подключен

```
<form data-dev data-popup-message="#form-message" class="form"
action="#">
  ...
</form>
```

События

После каждой отправки формы срабатывает событие **formSent**, его можно отловить в любой части кода:

```
document.addEventListener("formSent", function (e) {  
    // Форма  
    const currentForm= e.detail.form;  
});
```

Расположение и дополнительные данные

Функции **formFieldsInit()**, **formSubmit()** а также объект **formValidate** находятся в файле **js/files/forms/form.js**

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Модуль слайдера «Swiper»](#)

[Следующий пост: Модуль кастомизации элемента SELECT ->](#)

Модуль кастомизации элемента SELECT

Категории: [Документация «Чертоги Фрилансера» v3.0.0](#)

Модуль позволяет как угодно стилизовать стандартный элемент формы SELECT и его пункты OPTION, а также реализует дополнительные возможности.

Подключение модуля

[HTML] В нужном месте вызвать сниппет **sel**, отредактировать HTML-код селекта под свои нужды

[SCSS] Раскомментировать строку **@import «select»;** в файле **src/scss/base/forms/forms.scss** — это подключит базовые стили селекта, отредактировать под свои нужды

[JS] Раскомментировать строку **import './libs/select.js'** в файле **js/app.js**

Настройки и функционал модуля

Для подключения того или иного функционала модуля используются различные HTML-атрибуты

Атрибуты для тега <SELECT>:

- 1. class=имя класса** — модификатор к конкретному селекту. В итоге получится `select select_имя класса`
- 2. multiple** — мультивыбор
- 3. disabled** — недоступен
- 4. data-tags** — режим тегов (только для multiple), позволяет вставлять выбранные значения в виде тегов с крестиком для удаления. Также есть возможность выводить эти теги в любом указанном месте, указав селектор блока в качестве значения атрибута

5. **data-scroll** — включает прокрутку для выпадающего списка, дополнительно можно подключить кастомный скролл `simplebar` в `js/app.js`. Указанное число для атрибута ограничит высоту контейнера выпадающего списка
6. **data-checkbox** (в работе) — стилизация элементов по `checkbox` (только для `multiple`)
7. **data-show-selected** — отключает скрывание выбранного элемента
8. **data-search** — позволяет искать по выпадающему списку
9. **data-speed** — позволяет указать скорость открытия/закрытия списка в миллисекундах, по умолчанию 150
10. **data-open** — селект открыт сразу
11. **data-submit** — отправляет форму при изменении селекта
12. **data-pseudo-label=заголовок** — добавляет псевдоэлемент к заголовку селекта с указанным текстом, а также класс **`_select-pseudo-label`**

Атрибуты для тега **<OPTION>**:

1. **data-class=имя класса** — добавляет класс
2. **data-asset=путь к картинке или текст** — добавляет в элемент списка структуру двух колонок с указанными данными
3. **data-href=адрес ссылки** — добавляет ссылку в элемент списка
4. **data-href-blank** — откроет ссылку в новом окне

Атрибуты для для плейсхолдера (плейсхолдер — это **<OPTION>** с пустым `value`)

1. **data-label** — добавляет `label` к селекту
2. **data-show** — показывает плейсхолдер в списке (только для единичного выбора)

Атрибуты для прочих элементов

1. **data-one-select** — селекты внутри объекта с этим атрибутом будут открываться только по одному. То есть при открытии селекта другой открытый селект закрывается.

Пример селекта с некоторыми атрибутами


```
<select data-submit data-scroll name="form[]" class="form">
  <option value="" selected>Плейсхолдер</option>
  <option value="2">Пункт</option>
  <option data-href="about.html" value="3">Пункт ссылка</option>
  <option value="4">Пункт</option>
</select>
```

Классы которые формируются модулем

1. select — Главный блок
2. select__body — Тело селекта
3. select__title — Заголовок
4. select__value — Значение в заголовке
5. select__label — Лейбл
6. select__input — Поле ввода
7. select__text — Оболочка текстовых данных
8. select__link — Ссылка в элементе
9. select__options — Выпадающий список
10. select__scroll — Оболочка при скролле
11. select__option — Пункт
12. select__content — Оболочка контента в заголовке
13. select__row — Ряд
14. select__asset — Дополнительные данные
15. _select-disabled — Запрещен
16. _select-tag — Класс тега
17. _select-open — Список открыт
18. _select-active — Список выбран
19. _select-focus — Список в фокусе
20. _select-multiple — Мультивыбор
21. _select-checkbox — Стиль чекбокса
22. _select-selected — Выбранный пункт
23. _select-pseudo-label — Псевдолейбл для заголовка селекта

События

После каждого выбора элементы селекта срабатывает событие **selectCallback**, его можно отловить в любой части кода:

```
document.addEventListener("selectCallback", function (e) {  
  // Селект  
  const currentSelect = e.detail.select;  
});
```

Расположение и дополнительные данные

Класс **SelectConstructor** находится в файле **js/libs/select.js**

Нашли неточность? Хотите что-то добавить?

Пишите комментарий!

Этот контент вам открыт. Спасибо за поддержку канала!

[<- Предыдущий пост: Работа с формами и элементами форм](#)