4. Учебный проект: время разбивать камни

В этом разделе мы разобьем `main.js` на отдельные JS-модули, чтобы избавиться от монолитного кода.

Задачи

Сборка

1. Настройте сборку проекта так, чтобы зависимости в JS собирались при помощи сборщика модулей Webpack. Для этого выполните следующие шаги:

1. Установите в качестве npm-зависимости `webpack` и `webpack-cli`.

2. Создайте в корне проекта файл `webpack.config.js` и опишите конфигурацию сборки:

- установите режим сборки `development`;
- в качестве точки входа установите `src/main.js`;
- директорию для сборки задайте `public`, но помните, что путь должен быть абсолютный. Будет лучше, если он будет вычисляться сам, в момент чтения конфига webpack'ом. За это отвечает команда среды `node.js`: `path.join(__dirname, \`public\`)`, где `path` — стандартный модуль `node.js` для работы с путями, а `__dirname` — глобальная переменная в `node.js`, которая содержит путь до текущей директории;
- файл сборки назовите `bundle.js`;
- активируйте генерацию source-maps.

3. Добавьте npm-скрипт в `package.json` с именем `build` и значением `webpack`.

2. В файле `public/index.html` замените подключение скрипта `main.js` на `bundle.js`.

3. Файлы сборки (именно файлы сборки, а не вся директория `public`) не должны попасть в репозиторий. Поэтому добавьте их в `.gitignore`.

4. Настройте линтер (**ESLint**) так, чтобы он работал с вашими файлами как с модулями ES2015, для этого укажите в начале файла `.eslintrc.yml`:

```
parserOptions:

  ecmaVersion: 2015

  sourceType: 'module'
```

После того, как вы включили поддержку модулей в вашем проекте, директива `'use strict';` вам больше не нужна.

Разбить на модули

1. Создайте модули и перенесите в них написанные в прошлом задании функции для генерации DOM-элементов из `main.js`. Эти модули — компоненты. Поэтому:

- для них нужно завести отдельную директорию `src/components`;
- для их именования используйте существительные;
- для экспорта функций используйте именованный экспорт.

2. Подключите эти модули в `main.js`, сохранив логику работы.

3. Проверьте, что все хорошо и проект собирается выполнив `npm run build`.

В директории `public` должны появиться два файла: `bundle.js` и `bundle.js.map`. Так и есть? Тогда идем дальше. Если что-то пошло не так, то сравните ваш конфиг, с конфигом на слайдах. Найдите несоответствия и устраните их.

Настройка дев-сервера

1. Установите в качестве npm-зависимости `webpack-dev-server`.

2. Настройте его используя файл `webpack.config.js`:

- нужно добавить в него путь до директории с нашей сборкой `public`. Но путь не простой, а абсолютный;
- и прописать url, по которому будет доступно приложение из браузера.

3. Осталось добавить npm-скрипт и можно пользоваться. Назовем его `start` со значением `webpack-dev-server --open`.

Теперь достаточно выполнить команду `npm start`, открыть браузер с URLом из конфигурации dev-server'а и убедиться, что все работает. Скоро начнется самое интересное: =)

Задание можно отправить на проверку только после привязки к нему пулреквеста, отправленного из ветки `module2-task1`.

5. Личный проект: время разбивать камни

Это задание идентично заданию учебного проекта «Время разбивать камни». Нам нужно также разбить main.js на отдельные JS-модули, чтобы избавиться от монолитного кода.

Задачи

Сборка

1. Настройте сборку проекта так, чтобы зависимости в JS собирались при помощи сборщика модулей Webpack. Для этого выполните следующие шаги:

1. Установите в качестве npm-зависимости `webpack` и `webpack-cli`.

2. Создайте в корне проекта файл `webpack.config.js` и опишите конфигурацию сборки:

- установите режим сборки `development`;
- в качестве точки входа установите `src/main.js`;
- директорию для сборки задайте `public`, но помните, что путь должен быть абсолютный. Будет лучше, если он будет вычисляться сам, в момент чтения конфига webpack'ом. За это отвечает команда среды `node.js`: `path.join(__dirname, \`public\`)`, где `path` — стандартный модуль `node.js` для работы с путями, а `__dirname` — глобальная переменная в `node.js`, которая содержит путь до текущей директории;
- файл сборки назовите `bundle.js`;
- активируйте генерацию source-maps;

3. Добавьте npm-скрипт в `package.json` с именем `build` и значением `webpack`.

2. В файле `public/index.html` замените подключение скрипта `main.js` на `bundle.js`.

3. Файлы сборки (именно файлы сборки, а не вся директория `public`) не должны попасть в репозиторий. Поэтому добавьте их в `.gitignore`.

4. Настройте линтер (**ESLint**) так, чтобы он работал с вашими файлами как с модулями ES2015, для этого укажите в начале файла `.eslintrc.yml`:

```
parserOptions:

  ecmaVersion: 2015

  sourceType: 'module'
```

После того, как вы включили поддержку модулей в вашем проекте, директива `'use strict';` вам больше не нужна.

Разбить на модули

1. Создайте модули и перенесите в них написанные в прошлом задании функции для генерации DOM-элементов из `main.js`. Эти модули — компоненты. Поэтому:

- для них нужно завести отдельную директорию `src/components`;
- для их именования используйте существительные;
- для экспорта функций используйте именованный экспорт;

2. Подключите эти модули в `main.js`, сохранив логику работы.

3. Проверьте, что все хорошо и проект собирается выполнив `npm run build`.

В директории `public` должны появиться два файла: `bundle.js` и `bundle.js.map`. Так и есть? Тогда идем дальше. Если что-то пошло не так, то сравните ваш конфиг, с конфигом на слайдах. Найдите несоответствия и устраните их.

Настройка дев-сервера

1. Установите в качестве npm-зависимости `webpack-dev-server`.

2. Настройте его используя файл `webpack.config.js`:

- нужно добавить в него путь до директории с нашей сборкой `public`. Но путь не простой, а абсолютный;

- и прописать url, по которому будет доступно приложение из браузера.

3. Осталось добавить npm-скрипт и можно пользоваться. Назовем его `start` со значением `webpack-dev-server --open`.

Теперь достаточно выполнить команду `npm start`, отрыть браузер с URLом из конфигурации dev-server'а и убедиться, что все работает. Скоро начнется самое интересное: =)

Задание можно отправить на проверку только после привязки к нему пулреквеста, отправленного из ветки `module2-task1`.