Чеклист по самопроверке №10

Работа принимается

Если в проекте есть:

- HTML, CSS, JS-файлы и изображения в папке `src`;
- файл `index.html`;
- стили подключены в отдельном файле;
- директория `blocks`;
- папка `images` с картинками;
- файлы скриптов `Card.js`, `FormValidator.js`, `UserInfo.js`, `Section.js`, `Popup.js`, `PopupWithForm.js`, `PopupWithImage.js`, `Api.js` в директории `components`;
- файл `.gitignore` с исключениями `node_modules` и `dist`;
- файл `README.md`, который содержит:
  1. Заголовок-название.
  2. Описание проекта и его функциональности.
  3. Указание использованных технологий.
  4. Ссылку на GitHub Pages.

Webpack настроен:

- установлены `webpack`, `webpack-cli` и `webpack-dev-server`;
- настроены сборки `build` и `dev`;
- скрипты созданы в `package.json`;
- настроена минификация и транспиляция JS-бабелем. Webpack собирает весь JavaScript в один файл и автоматически добавляет в HTML-тег `script` со ссылкой на него;
- настроена обработка CSS;
- настроена минификация CSS и автоматическое добавление вендорных префиксов;
- настроена обработка изображений и шрифтов;
- если в HTML есть ссылки на локальные картинки, при сборке всё работает.

В HTML/CSS:

- корректно задан `viewport`, прописаны `title` и `lang`;
- `normalize.css` импортируется в `index.css` выше остальных файлов стилей;
- соблюдается форматирование и иерархия отступов;
- соблюдена методология БЭМ.

В JavaScript + HTML/CSS:

Код объектно-ориентирован:

- Использованы ES6-классы;
- Каждый класс описан в отдельном JS-файле и импортирован в `index.js`;
- Каждый класс выполняет строго одну задачу. Всё, что относится к решению этой задачи, находится в классе. Ни один другой класс к решению этой задачи не относится.
- Для описания взаимодействия между классами используется слабое связывание, то есть внутри классов напрямую не создаются экземпляры других классов.
- Создан класс `Api`, внутри которого описаны запросы к серверу. Запросы к серверу не должны быть описаны внутри других классов или `index.js`
  - Каждый метод, включающий обращение к серверу содержит `return fetch`, т.е возвращает объект `Promise`.

- Все операции над DOM включены внутрь цепочки промисов.
- Ответ от сервера всегда проверяется на корректность:

```
.then(res => {
  if (res.ok) {
    return res.json();
  }

  // если ошибка, отклоняем промис
  return Promise.reject(`Ошибка: ${res.status}`);
});
```

- Каждый промис содержит обработку ошибок после обращения к серверу.

Классы соответствует описанию из проектной работы:

- Экземпляр класса `Section` создается для каждого контейнера, в который требуется отрисовывать элементы.
- Экземпляр класса `FormValidator` создаётся для каждой проверяемой формы.
- Экземпляр класса `UserInfo` создается единожды.
- Экземпляр класса `Api` создается единожды.

▼ Внутри `Api` не создаются экземпляры других классов, не вызываются методы других классов. Используется слабое связывание между классами.

Цепочка промисов продолжается вне класса `Api` в `index.js` благодаря `return fetch` внутри методов.

В продолжение цепочки промисов может включаться инстанцирование других классов и обращение к их методам.

- Класс `Popup` — базовый, имеет двух наследников, которые создаются для каждого модального окна. Наследники соответствуют описанию из проектной работы.
- `index.js` содержит только логику инстанцирования классов и взаимодействия между ними. `index.js` содержит только остаточную часть кода, не относящуюся ни к одному классу, но описывает некоторую функциональность интерфейса (работа кнопок профиля и добавления карточек)
- Связь между классами — слабая, то есть описана колбэк-функциями, которые внедряются в класс.

Работает корректно вся функциональность из предыдущего задания:

- Карточки, полученные с сервера, добавляются на страницу с помощью JS.
- При редактировании профиля текст кнопки меняется на: «Сохранение...», пока данные загружаются.
- При наведении указателя мыши на аватар, на нём должна появляться иконка редактирования. При клике открывается модальное окно.
- Карточки должны отображаться на странице только после получения `id` пользователя.
- У карточек отображается количество лайков.
- Реализованы все запросы к серверу, указанные в описании проектной работы.
- Форма добавления карточки свёрстана, открывается, добавляет карточку.
- Работает нажатие на кнопку лайка.
- Удаление карточки реализовано корректно.
- Для всех полей ввода в формах включена лайв-валидация.
- Кнопка отправки формы неактивна, если хотя бы одно из полей не проходит валидацию.
- Карточку можно добавить из любого текстового поля нажатием `Enter`.
- Слушатель событий, который закрывает модальное окно по нажатию на `Esc`, добавляется при открытии модального окна и удаляется при его закрытии.

Модальные окна:

- плавно открываются и закрываются CSS-стилями;
- открываются с картинкой внутри, изображение не теряет пропорции;
- закрываются по клику в любом месте вне этого окна и по нажатию на `Esc`.

- закрываются на любых разрешениях экрана;
- не закрываются, если кликнуть по само́й форме, а не по оверлею.

Проверка данных работает так:

- Данные любого поля ввода проверяются одной унифицированной функцией, которая принимает текущую форму, поле ввода и конфиг в качестве параметров.
- Для проверки данных в поле используются HTML5-атрибуты и JS-свойство `ValidityState`.
- Создана отдельная функция, которая отвечает за состояние кнопки отправки сабмита формы.
- Функция `enableValidation` принимает объект параметров, который передаёт вложенным функциям.

Код оформлен без ошибок:

- имена переменных — существительные;
- имена коллекций NodeList — существительные во множественном числе;
- имена переменных отражают то, что в них хранится, имена функций — то, что они делают;
- если в проекте есть несколько переменных с похожими данными, то такие переменные имеют конкретные наименования;
- имена переменных и функций написаны в camelCase;
- отсутствует транслит и неуместные сокращения в наименованиях.

Код оптимизирован:

- Данные от пользователя не передаются свойству `innerHTML`.
- Код не повторяется. Если строчка нужна в нескольких местах, она должна быть вынесена в отдельную функцию.
- Если переменная объявлена через `let`, её значение должно быть напрямую перезаписано.
- Если переменная не перезаписывает своё значение, она объявлена через `const`.
- Отсутствуют «магические значения»: все числовые значения записаны в переменные.
- Операции над DOM-элементами выполняются до их вставки в разметку.
- Поиск одного и того же DOM-элемента не должен происходить дважды.
- Функция выполняет только одну задачу, например, возвращает разметку карточки.

Доступность интерфейса

- Все ссылки и интерактивные элементы имеют состояние наведения `:hover`.
- Контентные изображения имеют `alt` с корректным описанием, которое соответствует языку страницы.

Работа отклоняется от проверки

- Пул-реквест не отправлен на проверку.
- Не соблюдена концепция парного программирования: один из напарников контрибьютил в разы меньше/не контрибьютил вовсе.
- При открытии `index.html` в консоли возникают ошибки.
- Часть функциональности не реализована: отсутствует класс `Card` или `FormValidator`.
- Работа содержит вопросы или просьбы о помощи к ревьюеру.
- На повторных итерациях не исправлены критические замечания.