**Проектная работа 8**

В проектной работе этого спринта вы продолжите заниматься рефакторингом. Создадите ещё несколько классов и настроите связи между ними. В завершении проектной работы нужно будет настроить сборку проекта Вебпаком.

**Создайте класс Section**

Создайте класс Section, который отвечает за отрисовку элементов на странице. Этот класс:

* Первым параметром конструктора принимает объект с двумя свойствами: items и renderer. Свойство items — это массив данных, которые нужно добавить на страницу при инициализации класса. Свойство renderer — это функция, которая отвечает за создание и отрисовку данных на странице.
* Второй параметр конструктора — селектор контейнера, в который нужно добавлять созданные элементы.
* Содержит публичный метод, который отвечает за отрисовку всех элементов. Отрисовка каждого отдельного элемента должна осуществляться функцией renderer.
* Содержит публичный метод addItem, который принимает DOM-элемент и добавляет его в контейнер.

У класса Section нет своей разметки. Он получает разметку через функцию-колбэк и вставляет её в контейнер.

**Создайте класс Popup**

Создайте класс Popup, который отвечает за открытие и закрытие попапа. Этот класс:

* Принимает в конструктор единственный параметр — селектор попапа.
* Содержит публичные методы open и close, которые отвечают за открытие и закрытие попапа.
* Содержит приватный метод \_handleEscClose, который содержит логику закрытия попапа клавишей Esc.
* Содержит публичный метод setEventListeners, который добавляет слушатель клика иконке закрытия попапа. Модальное окно также закрывается при клике на затемнённую область вокруг формы.

**Создайте класс PopupWithImage**

Создайте класс PopupWithImage, который наследует от Popup. Этот класс должен перезаписывать родительский метод open. В методе open класса PopupWithImage нужно вставлять в попап картинку с src изображения и подписью к картинке.

**Создайте класс PopupWithForm**

Создайте класс PopupWithForm, который наследует от Popup. Этот класс:

* Кроме селектора попапа принимает в конструктор колбэк сабмита формы.
* Содержит приватный метод \_getInputValues, который собирает данные всех полей формы.
* Перезаписывает родительский метод setEventListeners. Метод setEventListeners класса PopupWithForm должен не только добавлять обработчик клика иконке закрытия, но и добавлять обработчик сабмита формы.
* Перезаписывает родительский метод close, так как при закрытии попапа форма должна ещё и сбрасываться.

Для каждого попапа создавайте свой экземпляр класса PopupWithForm.

**Создайте класс UserInfo**

Класс UserInfo отвечает за управление отображением информации о пользователе на странице. Этот класс:

* Принимает в конструктор объект с селекторами двух элементов: элемента имени пользователя и элемента информации о себе.
* Содержит публичный метод getUserInfo, который возвращает объект с данными пользователя. Этот метод пригодится когда данные пользователя нужно будет подставить в форму при открытии.
* Содержит публичный метод setUserInfo, который принимает новые данные пользователя и добавляет их на страницу.

**Преобразуйте класс Card**

Свяжите класс Card c попапом. Сделайте так, чтобы Card принимал в конструктор функцию handleCardClick. Эта функция должна открывать попап с картинкой при клике на карточку.

**Создайте файл .gitignore**

Следующим заданием будет инициализация npm и настройка Webpack. Это значит, что в проекте появятся папки node\_modules и dist. Эти папки не принято добавлять в git-репозиторий, так как любой разработчик может сгенерировать их у себя на компьютере. Для этого достаточно иметь package.json.

Чтобы node\_modules и dist не попадали в репозиторий, нужно научить git их игнорировать. Для этого в корне проекта создайте файл .gitignore и добавьте в него две строки:

Скопировать код

node\_modules

dist

Сохраните файл. Теперь git не будет обращать внимание на эти папки.

Примечание: перед тем, как отправить проект на код-ревью, добавьте node\_modules в файл .gitignore. Иначе проект не загрузится или ревьюер отклонит его от проверки.

**Настройте сборку Вебпаком**

1. Инициализируйте npm в проекте.
2. Установите webpack, webpack-cli и webpack-dev-server.
3. Настройте две сборки: build и dev. Создайте соответствующие скрипты в package.json. Скрипт build должен пересоздавать папку dist с собранным проектом. Скрипт dev запускать проект на локальном сервере.
4. Настройте минификацию и транспиляцию JS бабелем. Из index.html уберите теги script с подключением скриптов. Вебпак должен собирать весь JavaScript в один файл и автоматически добавлять в HTML тег script со ссылкой на него.
5. Настройте обработку CSS: в HTML больше не должно быть тега link со ссылкой на CSS-файл. За обработку и подключение CSS должен отвечать Webpack.
6. Настройте минификацию CSS и автоматическое добавление вендорных префиксов.
7. Настройте обработку изображений и шрифтов.
8. Настройте обработку HTML: если в HTML есть ссылки на локальные картинки, при сборке всё должно работать.

HTML, CSS и JS-файлы должны быть в папке src. Итоговая структура проекта должна быть такой:



Пока что ваш GH-pages не умеет работать с Вебпаком. Поэтому на данном этапе обновлять проект на GH-pages не нужно. Про то, как правильно выкладывать проект с Вебпаком на GH-pages вы узнаете в следующем спринте.

**Требования к коду**

* Добавьте в проект классы Section, Popup, PopupWithForm, PopupWithImage и UserInfo. Каждый из них выполняет строго одну задачу. Всё, что относится к решению этой задачи, находится внутри класса.
* Если классы нужно связать друг с другом, делайте это передаваемой в конструктор функцией-колбэком.
* Все классы должны быть вынесены в отдельные файлы.
* В файле index.js должно остаться только создание классов и добавление некоторых обработчиков.