Проект. Mesto на JavaScript

интерактивная страница

Где описание и иллюстрации: figma

Откуда взять основу: 05_files_mesto.zip

Где работать: опубликовать сайт в Git, а когда закончите — запушить работу себе в GitHub в репозиторий mesto-project

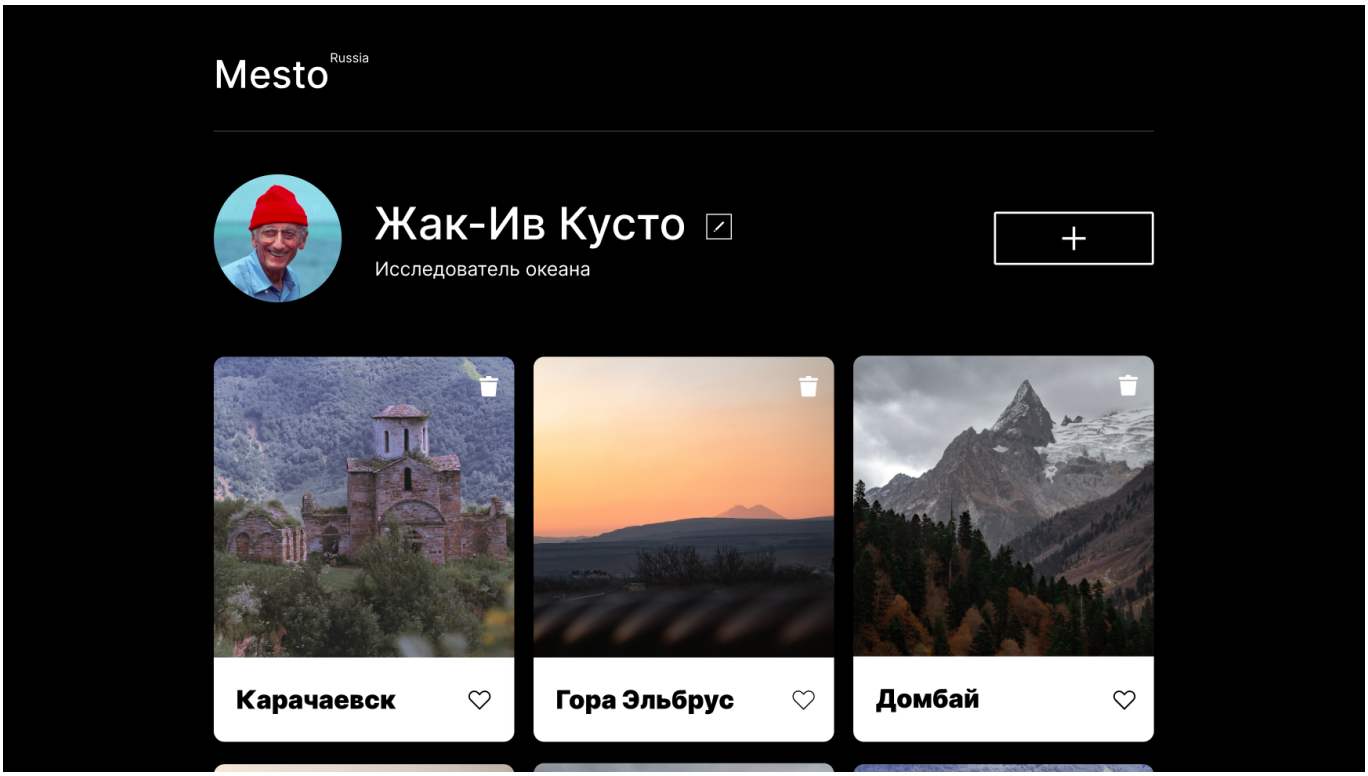
Как проверить: по чек-листу

Эту проектную работу нужно выполнить и проверить самостоятельно. Ды найдёте чек-лист для самопроверки — пройдитесь по всем пунктам и убедитесь, что всё сделано правильно.

Вы будете работать над сервисом Mesto: интерактивной страницей, на которую можно добавлять фотографии, удалять их и ставить «лайки».

Возьмите уже свёрстанный сайт (05_files_mesto.zip) и опубликуйте его в Git.

Вы будете добавлять в Mesto функциональность на JavaScript. Не забудьте запушить готовую работу к себе в GitHub в репозиторий `pinYY_Familiya`.



1. Шесть карточек «из коробки»

При загрузке на странице должно быть шесть карточек, которые добавит JavaScript. Возьмите их названия и фотографии из готового массива.

```
const initialCards = [
  {
    name: 'Архыз',
   link: 'https://my-server/cards-nameproject/arkhyz.jpg'
  },
  {
    name: 'Челябинская область',
   link: 'https://my-server/cards-nameproject/chelyabinsk-oblast.jpg'
  },
  {
    name: 'Иваново',
   link: 'https://my-server/cards-nameproject/ivanovo.jpg'
  },
  {
    name: 'Камчатка',
   link: 'https://my-server/cards-nameproject/kamchatka.jpg'
  },
  {
    name: 'Холмогорский район',
   link: 'https://my-server/cards-nameproject/kholmogorsky-rayon.jpg'
  },
  {
    name: 'Байкал',
   link: 'https://my-server/cards-nameproject/baikal.jpg'
  }
];
```

При создании карточки необходимо использовать технологию работы с template. Создайте отдельную функцию, например createCard(), которая будет отвечать за создание разметки карточки. Обратите внимание, что данная функция должна только создавать карточку и возвращать ее элемент с помощью return. Для создания новой карточки вам необходимо сделать клон разметки карточки из содержимого темплейта. Вы получите чистую карточку. Затем заполняете эту карточку данными. Подставляете информацию о названии карточки, ссылку на картинку, альтернативную информацию. Будет правильно все элементы карточки найти и сохранить в константы, чтобы не повторять поиск элементов в DOM, так как это расходует много ресурсов браузера. После заполнения всех атрибутов элементов карточки верните ее элемент (уже заполненный данными клон) через return. Для создания всех карточек из массива необходимо перебрать массив (старайтесь использовать методы массивов, это хорошая практика) и для каждого элемента массива вызвать функцию создания карточки createCard() которая вернет готовую карточку. Эту готовую карточку можно теперь добавлять в разметку страницы. Для размещения карточек в вашей разметке сайта есть элемент списка с классом places__list.

2. Работа модальных окон

В проекте предусмотрены диалоговые окна, которые будут содержать формы для работы с данными или контент для отображения его пользователю. Окна должны быть «модальными» — это значит, что такое окно закрывает собой все содержимое страницы и запрещает работу с другими элементами, пока окно открыто. Среди разработчиков такие диалоговые окна, которые могут открываться на переднем плане и закрываться, лаконично называют «поп-ап» от английского глагола “pop-up”, который переводится как «неожиданно возникать» или «всплывать».

Все необходимые поп-апы для работы приложения уже присутствуют в верстке. Обратите внимание, что все основные элементы поп-апов имеют одинаковые классы. Это позволяет использовать единый стиль оформления любого поп-апа, а также создавать универсальные функции для работы с ними. Для

того, чтобы получить возможность найти в разметке конкретный поп-ап, у каждого поп-апа, кроме основного класса “pop-up” присутствует модификатор.

На данном этапе вам нужно будет использовать три поп-апа: Поп-ап редактирования профиля (его модификатор popup_type_edit), Поп-ап добавления карточки (popup_type_new-card), Поп-ап с картинкой (popup_type_image)

Каждый поп-ап нужно будет найти 1 раз вверху файла index.js и дать ему понятное название: profilePopup, cardPopup, imagePopup

Изначально поп-ап не виден (display: none). Чтобы поп-ап открылся, добавляйте ему модификатор popup_is-opened. Чтобы закрыть поп-ап, удаляйте у него модификатор popup_is-opened.

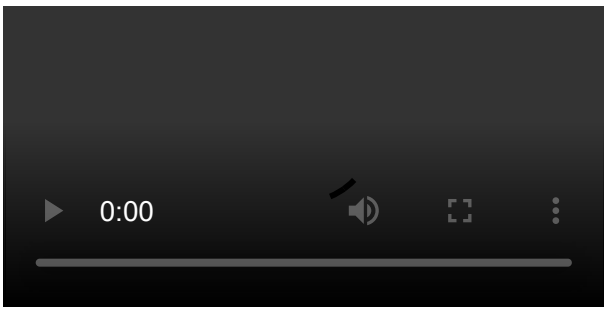
classList.add("popup_is-opened"); — вот эта операция будет происходить со всеми поп-апами, которые нужно «открыть». Это значит, что нужно вынести эту логику в отдельную функцию openModal:

```
function openModal(popup) {
    popup.classList.add('popup_is-opened');
}
```

Она будет принимать в вызов «любой поп-ап» и добавлять ему класс popup_is-opened, открывая его. Аналогично нужно реализовать общую функцию для закрытия любого поп-апа. Обратите внимание, что данные функции должны быть универсальными и не должны содержать никакого другого функционала, относящегося к конкретным поп-апам. О реализации работы с каждым конкретным поп-апом мы расскажем ниже.

2.1. Форма редактирования профиля пользователя

Модальное окно должно открываться по нажатию кнопки «Редактировать», а закрываться — при клике по крестику в правом верхнем углу:



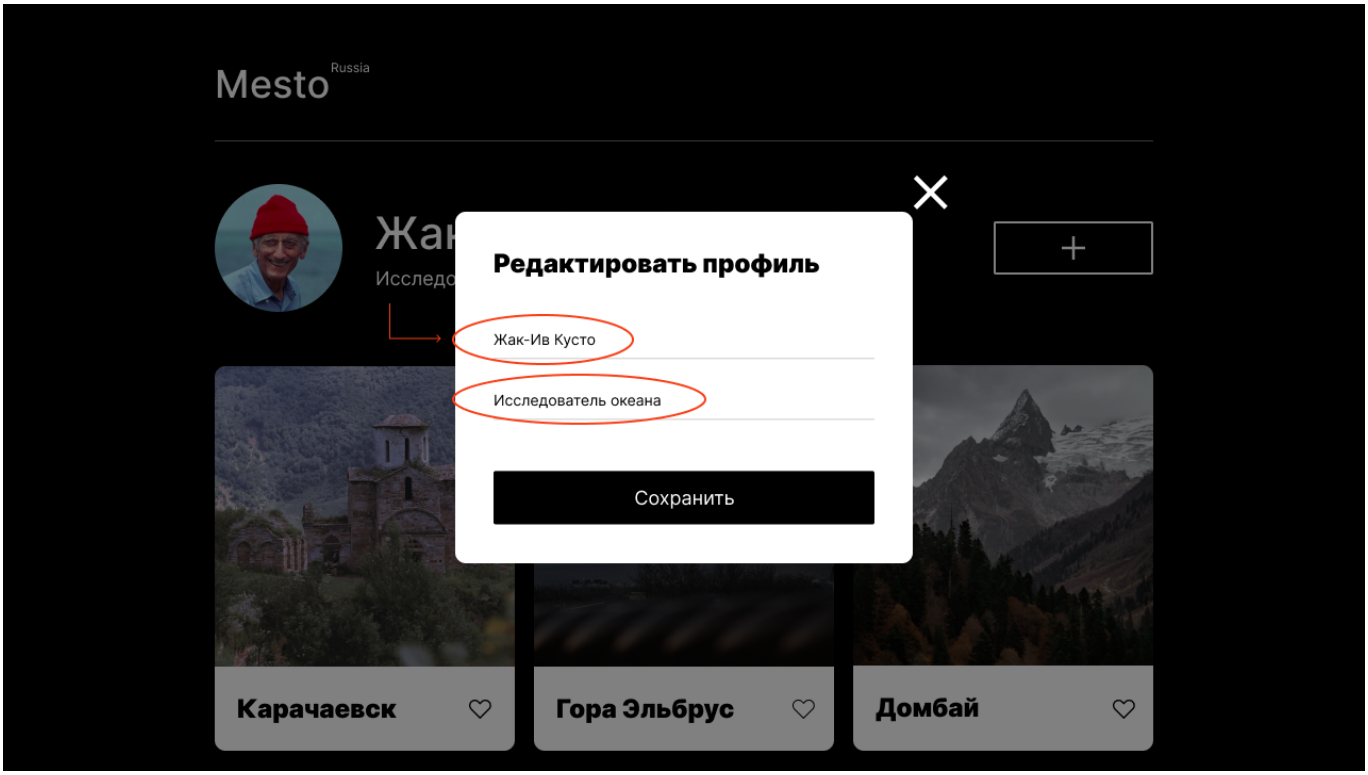
--1project-4-01_Механизм открытия и закрытия поп-апа.mp4

Отслеживайте клик по кнопке методом addEventListener.

Поля формы

При открытии формы поля «Имя» и «О себе» должны быть заполнены теми значениями, которые отображаются на странице.

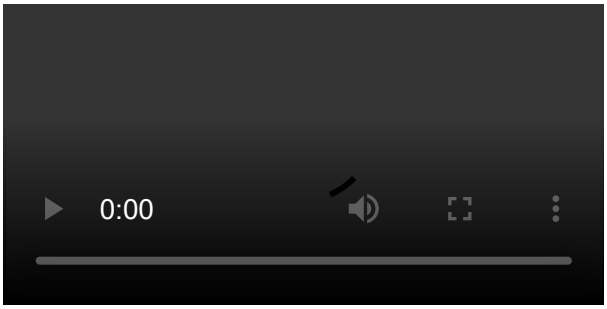
Заполнение полей формы данными это функционал одной конкретной формы, поэтому он должен быть оформлен отдельной функцией. Данная функция будет являться обработчиком для события клика, о котором сказано выше. В этой функции нужно заполнить поля формы необходимыми данными и вызвать универсальную функцию openModal() передав в нее поп-ап профиля.



Если пользователь закрывает модальное окно, нажав на крестик, то введённые значения не сохраняются. О том, как работает кнопка «Сохранить», расскажем дальше.

Редактирование имени и информации о себе

Открытия и закрытия модального окна недостаточно. Как следует из названия поп-апа, он должен уметь редактировать соответствующие поля страницы. После внесения изменений и нажатия кнопки «Сохранить» информация на странице должна обновиться, а поп-ап автоматически закрыться:



--1project-4-02_Изменение страницы через поп-ап copy.mp4

Вам предстоит часто разбираться в непонятном коде или незнакомой теме. Смоделируем такую ситуацию.

Специальное событие submit отправляет форму (его мы ещё не проходили). Перед вами шаблон кода, реализующий его обработку. Постарайтесь в нём разобраться. Мы оставили в коде комментарии, которые с этим помогут:

```
// Находим форму в DOM
const profileFormElement = // Воспользуйтесь методом querySelector()
// Находим поля формы в DOM
const nameInput = // Воспользуйтесь инструментом .querySelector()
const jobInput = // Воспользуйтесь инструментом .querySelector()

// Обработчик «отправки» формы, хотя пока
// она никуда отправляться не будет
function handleProfileFormSubmit(evt) {
    evt.preventDefault(); // Эта строчка отменяет стандартную отправку формы.
                                                // Так мы можем определить свою
логику отправки.
                                                // О том, как это делать,
расскажем позже.

    // Получите значение полей jobInput и nameInput из свойства value

    // Выберите элементы, куда должны быть вставлены значения полей

    // Вставьте новые значения с помощью textContent
}

// Прикрепляем обработчик к форме:
// он будет следить за событием “submit” - «отправка»
profileFormElement.addEventListener('submit', handleProfileFormSubmit);
```

Пока эта информация не сохраняется между перезагрузками страницы. Мы научимся сохранять её позже, когда подключим сайт к серверу.

2.2. Форма добавления карточки

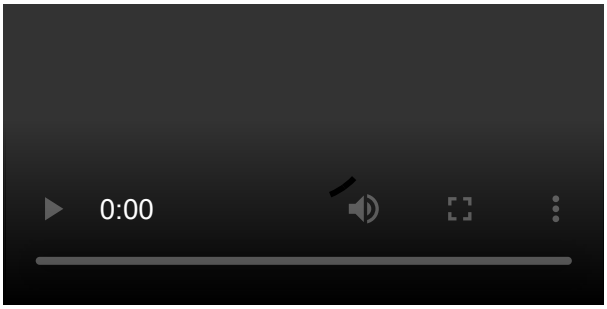
Сделайте так, чтобы форма открывалась нажатием на кнопку «+» и закрывалась кликом на крестик. Для открытия данной формы также нужен свой собственный обработчик. В нем вам нужно очистить поля формы и после этого использовать универсальную функцию открытия поп-апа.

1project-4-05-Так открывается и закрывается форма добавления карточек.mov

Так открывается и закрывается форма добавления карточек

Добавление карточки

Дайте пользователю возможность добавлять карточки:



--1project-4-06_Можно написать имя карточки и дать ссылку на картинку.mp4

Сделайте так, чтобы при клике на «сохранить» новая карточка попадала в начало контейнера с ними. А диалоговое окно после добавления автоматически закрывалось.

Чтобы создавать новые карточки, добавьте обработчик событий submit. Сделайте это аналогично прошлому шагу, в котором вы настраивали редактирование информации о пользователе (вы можете использовать для названий cardFormElemnt и handleCardFormSubmit, чтобы не запутаться).

Обратите внимание, что для создания карточки вам нужно использовать ту же самую функцию, которую вы использовали для создания карточек из массива. Только теперь нужно передать в нее данные из формы, которые ввел пользователь.

Теперь давайте наполним наши карточки функционалом. На карточке у нас три интерактивных элемента: сердечко, корзина, фото. Сейчас эти элементы не реагируют на клик. Необходимо на каждый элемент установить слушатель. Сделать это нужно внутри функции создания карточки, чтобы слушатели были установлены на каждый элемент внутри карточки.

3. «Лайк» карточки

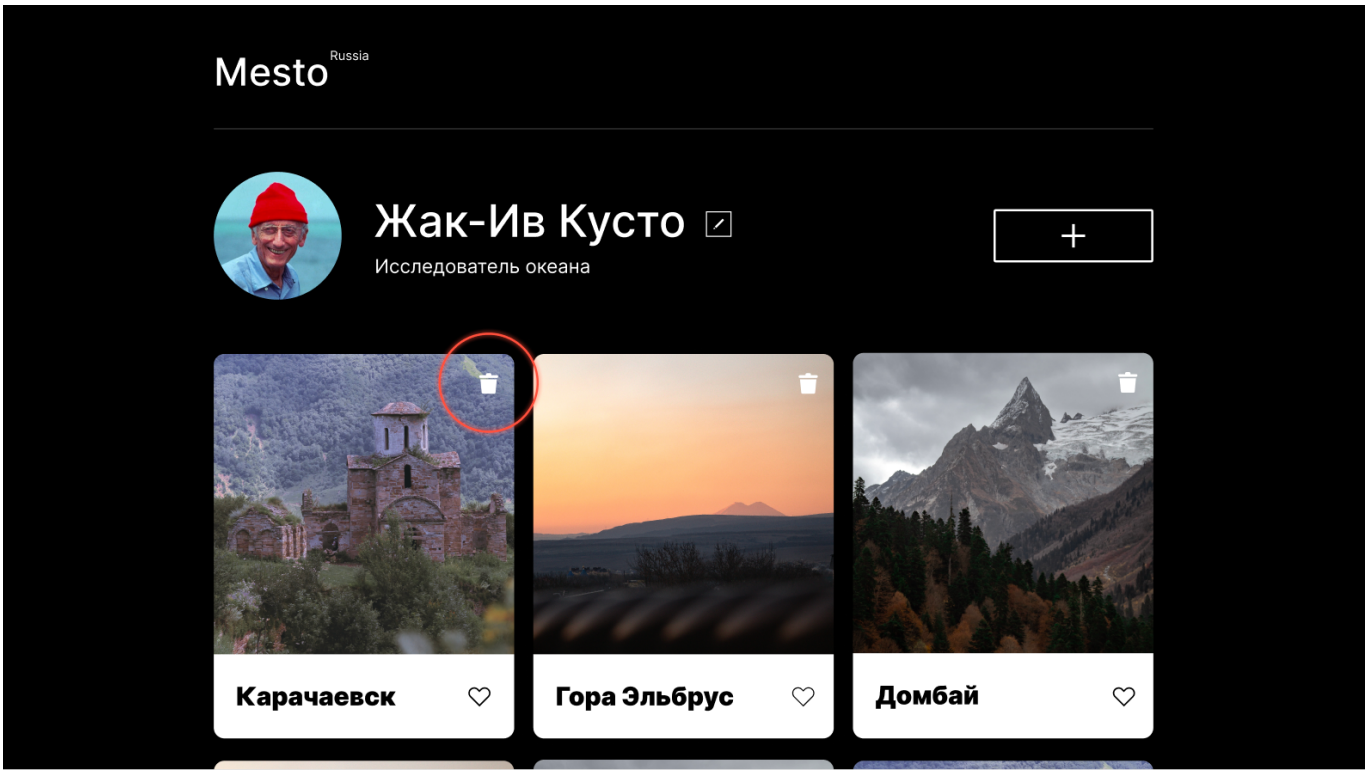
Сделайте так, чтобы карточки можно было «лайкать». Для изменения состояния сердечка лайка используйте модификатор card__like-button_is-active для кнопки «лайка». Модификатор можно добавлять и удалять с помощью одного метода toggle. Подробнее о нем можно прочесть здесь: https://doka.guide/js/element-classlist/#classlist.toggle

--1project-4-07-like.mov

Если «лайкнуть» карточку, сердечко поменяет цвет

4. Удаление карточки

Для реализации удаления используйте элемент корзины, его класс card__delete-button. Обратите внимание, что удаляться должна именно та карточка, в которой была нажата корзина. Для поиска карточки-родителя нажатой кнопки можно использовать метод closest https://doka.guide/js/element-closest/



Кнопка «удалить» — классическая урна

Теперь настройте, чтобы карточка удалялась при клике на эту иконку:

--1project-4-08-delete.mov

5. Открытие и закрытие поп-апа с картинкой

Настройте просмотр фотографий. Пусть открываются нажатием на картинку и закрываются кликом на крестик. Также как и для всех описанных ранее элементов, слушатель на картинку нужно установить внутри функции createCard(). При клике в картинку необходимо взять название карточки и ссылку на картинку и заполнить атрибуты соответствующих элементов поп-апа с картинкой. Заметьте, что сами элементы поп-апа нужно найти заранее, за пределами функции createCard(), чтобы их поиск в DOM не повторялся при каждом создании карточки или открытии изображения. Заполнив атрибуты элементов поп-апа останется только использовать openModal() для его открытия.

--1project-4-09-image.mov

Плавное открытие и закрытие поп-апов

Сделайте так, чтобы все поп-апы плавно открывались. Пусть проявляются из прозрачности и уходят в неё при закрытии:

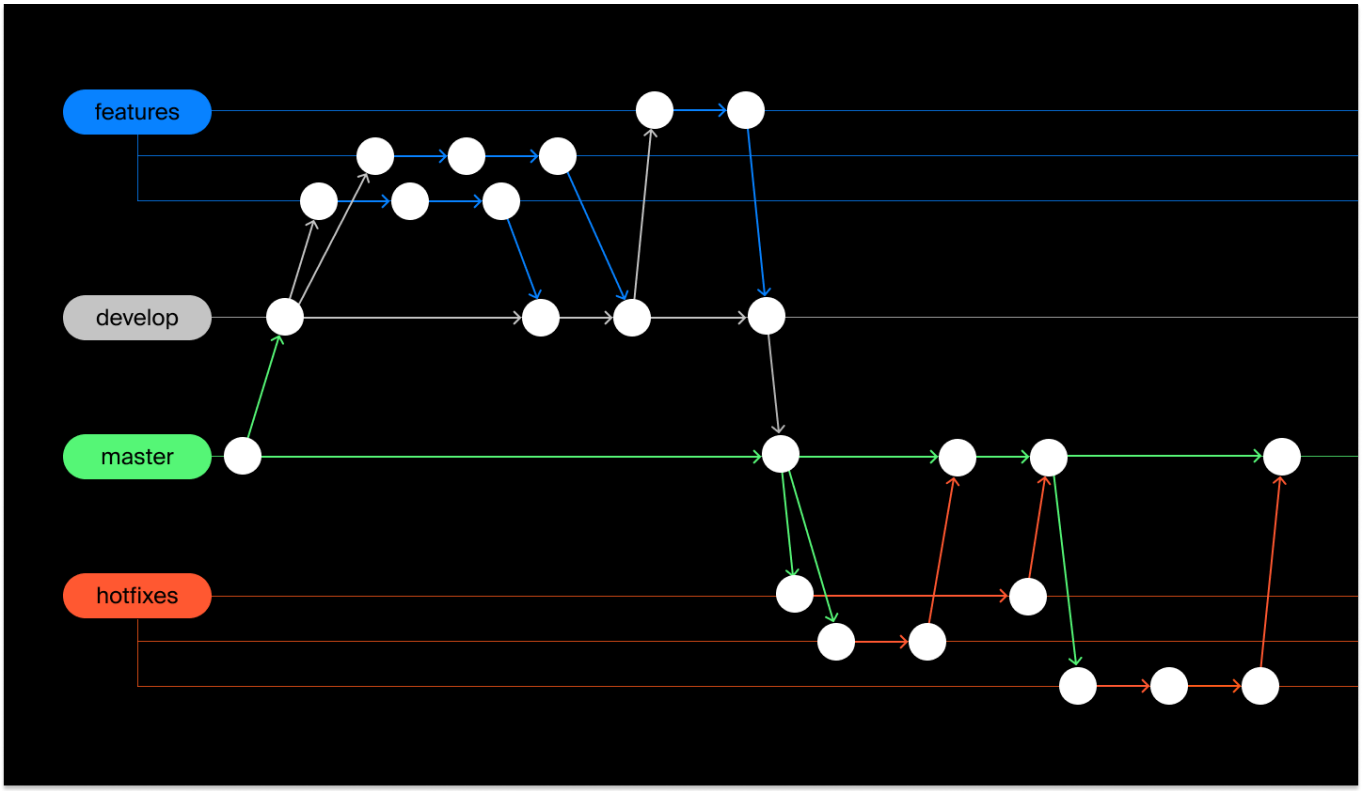
--1project-4-10-smooth.mov

Для подключения стилей, необходимых для плавного открытия и закрытия поп-апов, нужно при загрузке приложения один раз добавить каждому поп-апу модификатор popup_is-animated.

6. Git

Вы уже знаете основы работы с Git: умеете создавать ветки и коммиты. Научимся делать это более осмысленно. Так, чтобы в релиз попадало меньше багов, разработчики не мешались друг другу, а история коммитов и ветвлений была понятной.

Работа с Git в большом проекте может выглядеть так. Горизонтальные линии — это ветки, а круги — коммиты:



Main — главная ветка репозитория. Код в ней должен быть полностью работоспособен и готов к публикации на сервере. В нашем случае публикацию будет заменять проверка ревьюера, поэтому смержьте нужные коммиты в main до отправки работы на ревью.

Develop — это основная «рабочая» ветка. Все коммиты должны быть смержены в develop, протестированы и исправлены перед тем, как попадут в main.

Features — «функциональные» ветки, их может быть сколько угодно. Такие ветки создают, чтобы реализовать небольшую единицу функциональности: шапку сайта, форму контакта, логику добавления карточки. После того как функциональность реализована, коммиты переносят в develop. А feature-ветку — удаляют.

Имена функциональных веток начинают со слова feature, а затем кратко описывают функциональность, для которой ветка создана:

- feature/header;
- feature/contact-form;
- feature/insert-card.

Обычно сайт тестируют и отлаживают в ветке develop, а затем новые возможности переносят в main. Но иногда ошибки всё-таки всплывают, исправлять их нужно как можно скорее. Для этого предназначены ветки с приставкой hotfixes.

Hotfixes — ветки «быстрых исправлений», когда нужно что-то исправить в ветке main. В нашем случае исправлениями будут ваши собственные замеченные несоответствия чек-листу.

Веток быстрых фиксов может быть сколько угодно. Их называют по имени бага с приставкой hotfix/, например:

- hotfix/bem-naming;
- hotfix/mobile-layout-400;
- hotfix/card-duplicates.

Профессиональная работа с ветками в общем виде

1. Перед началом нового этапа разработки весь код из main мержат в develop. (В вашем случае старткит окажется в main, а работать над проектом вы будете в ветке develop).
2. Создают функциональную ветку feature/feature-name из develop.
3. Разрабатывают новую функциональность в ветке feature/feature-name.
4. После завершения разработки код из feature/feature-name мержат в develop.
5. Ветку feature/feature-name удаляют.
6. Если нужно реализовать что-то ещё, повторяют шаги с пункта 2.
7. Тестируют и исправляют всю функциональность в ветке develop, а затем мержат в main.
8. Работу отправляют на ревью. (В вашем случае вы проверяете себя по чек-листу)
9. По результатам проверки создают новую ветку быстрых исправлений hotfix/bug-name. В ней исправляют все проблемы, выявленные на ревью. Это является хорошей практикой, так как при исправлении ошибок работа кода может стать хуже, и в таком случае его можно откатить до предыдущего состояния и начать заново.
10. После завершения разработки код из hotfix/bug-name мержится в main.
11. Ветку hotfix/bug-name удаляют.

Заключение

Если перезагрузить страницу, карточки и «лайки» не сохранятся. Для этого пришлось бы писать бэкэнд. Тогда бы сайт получал карточки с сервера и там же сохранял новые. Вы же пока делали только фронтенд сайта, логику его интерактивности.

В реальных проектах так и происходит: фронтенд и бэкенд пишутся отдельно, а потом их связывают друг с другом.