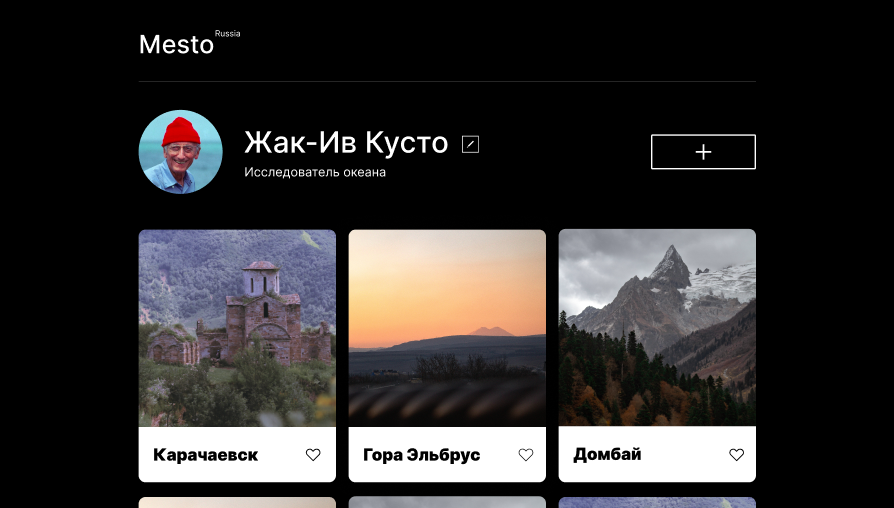
**Проектная работа 4**

В этом и следующих спринтах вы будете работать над сервисом Mesto: интерактивной страницей, куда можно добавлять фотографии, удалять их и ставить лайки.



Вам предстоит сверстать сайт, написать часть логики на JavaScript, опубликовать в Git. Все необходимые знания для этого у вас уже есть.

Как только вы перейдёте в урок проектной работы, репозиторий mesto скопируется в ваш аккаунт GitHub. Клонируйте репозиторий на свой компьютер через командную строку и работайте над проектом внутри папки mesto.

**Задание**

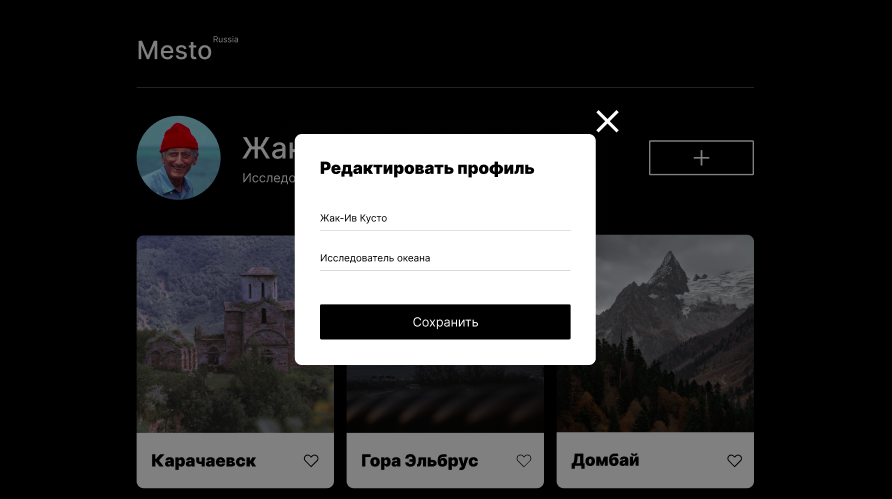
**1. Вёрстка**

**Основные требования**

* Внешний вид сайта соответствует макету: <https://www.figma.com/file/2cn9N9jSkmxD84oJik7xL7/JavaScript.-Sprint-4?node-id=0%3A1>
* Файловая структура организована по БЭМу.
* Вёрстка адаптивная: ширина зоны с содержимым должна меняться вместе с шириной окна браузера. При этом не должна появляться горизонтальная полоса прокрутки. Минимальная ширина: 320px (одна карточка в ряд). Максимальная: 1280px (три карточки в ряд).
* На странице присутствуют 6 карточек с фотографиями. Снимки и названия мест вы можете выбрать сами.

**Попап редактирования профиля**

Реализуйте одно диалоговое окно из макета — «Редактировать профиль». В нём два поля: «Имя» и «О себе», а также кнопка «Сохранить».



Среди разработчиков такие диалоговые окна лаконично называют «попап» от английского глагола “pop-up”, который переводится как «неожиданно возникать» или «всплывать».

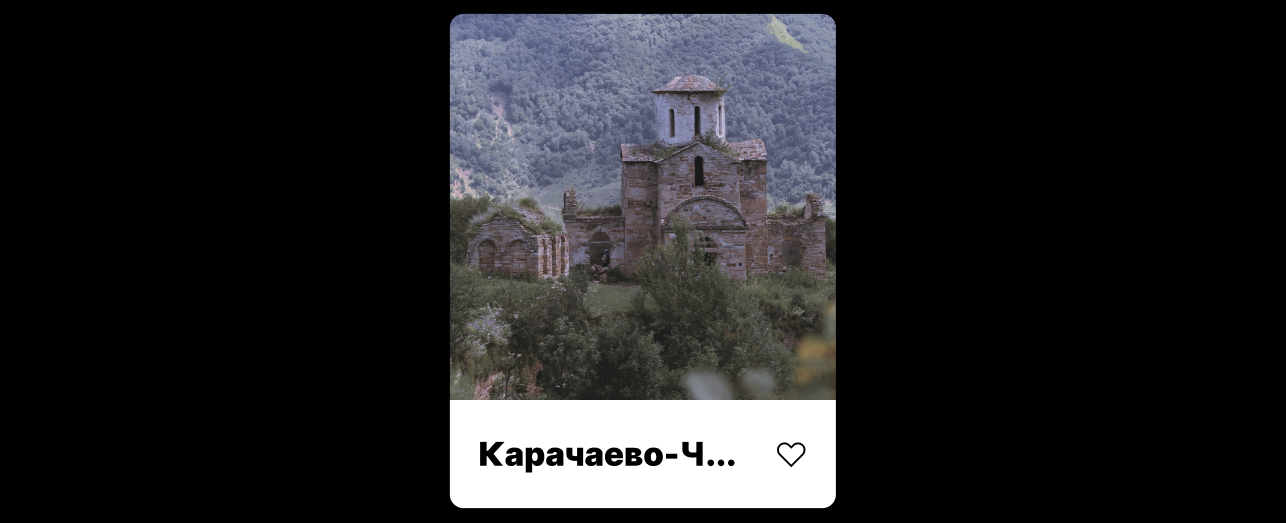
Позиционируйте попап фиксированно по центру экрана и задайте попапу полупрозрачный чёрный фон. Полупрозрачный фон верстайте отдельным блоком popup. Внутри него расположите само окно с формой popup\_\_container, спозиционированное по центру.

Изначально попап не виден (display: none). Чтобы попап открывался, добавляйте ему модификатор popup\_opened с одним-единственным правилом. Правило должно изменять значение свойства display на block или flex. Чтобы закрыть попап, удаляйте у него модификатор popup\_opened.

**Переполнение содержимого в блоке**

В некоторых частях макета возможно переполнение содержимым. Так в карточке «Карачаево-Черкессия» не вместившийся в блок текст обрезан, и появилось многоточие в конце.

Для иконки сердечка подготовьте сразу и активное состояние — иконка должна закрашиваться чёрным. Сейчас это не пригодится, но потребуется в следующем спринте.



При редактировании данных профиля из попапа пользователь тоже может ввести длинный текст. Чтобы реализовать переполнение блока с появляющимся многоточием в конце, используйте такой набор свойств CSS:

Скопировать кодCSS

.block {

text-overflow: ellipsis;

white-space: nowrap;

overflow: hidden;

}

Изучите свойства text-overflow и white-space самостоятельно в документации CSS.

**2. JavaScript**

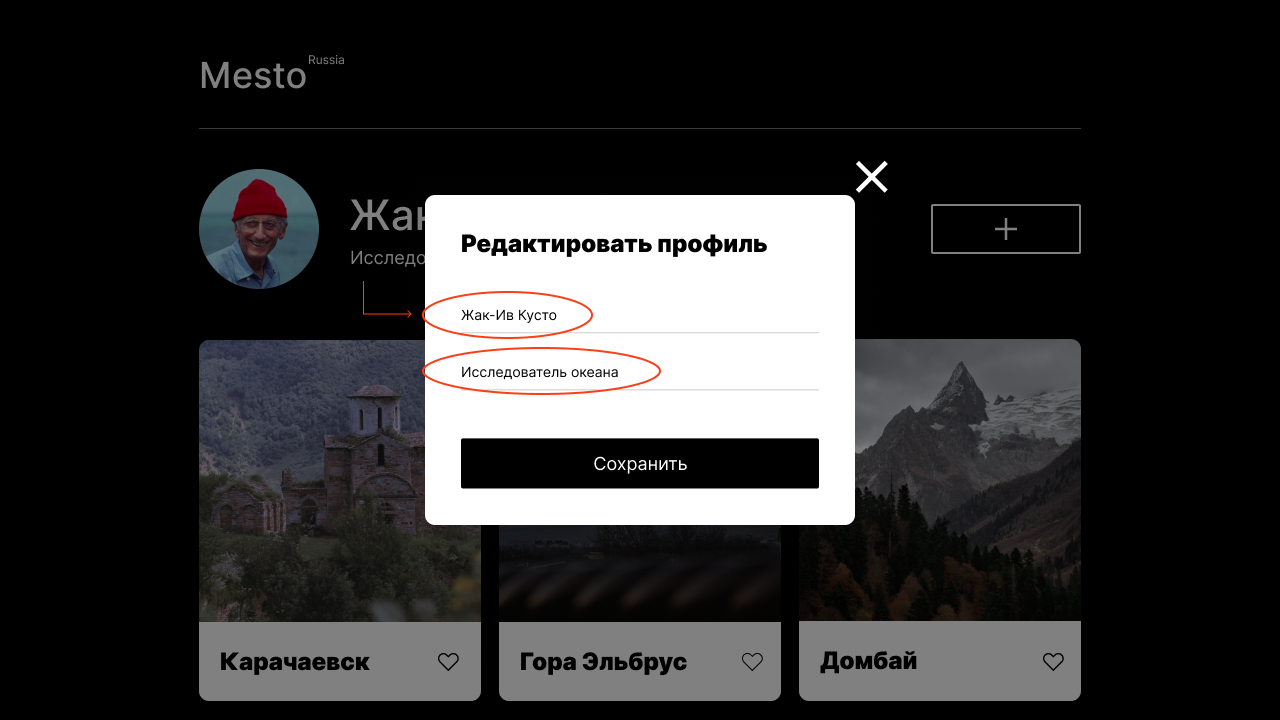
**Открытие и закрытие попапа**

Попап должен открываться по нажатию кнопки «Редактировать», а закрываться — при клике по крестику в правом верхнем углу:

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.



Если пользователь закрывает попап 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 попапа недостаточно. Как следует из названия попапа, он должен уметь редактировать соответствующие поля страницы. После внесения изменений и нажатия кнопки «Сохранить» информация на странице должна обновиться, а попап автоматически закрыться:

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:

Скопировать кодJAVASCRIPT

*// Находим форму в DOM*

let formElement = *// Воспользуйтесь методом querySelector()*

*// Находим поля формы в DOM*

let nameInput = *// Воспользуйтесь инструментом .querySelector()*

let jobInput = *// Воспользуйтесь инструментом .querySelector()*

*// Обработчик «отправки» формы, хотя пока*

*// она никуда отправляться не будет*

function formSubmitHandler (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»*

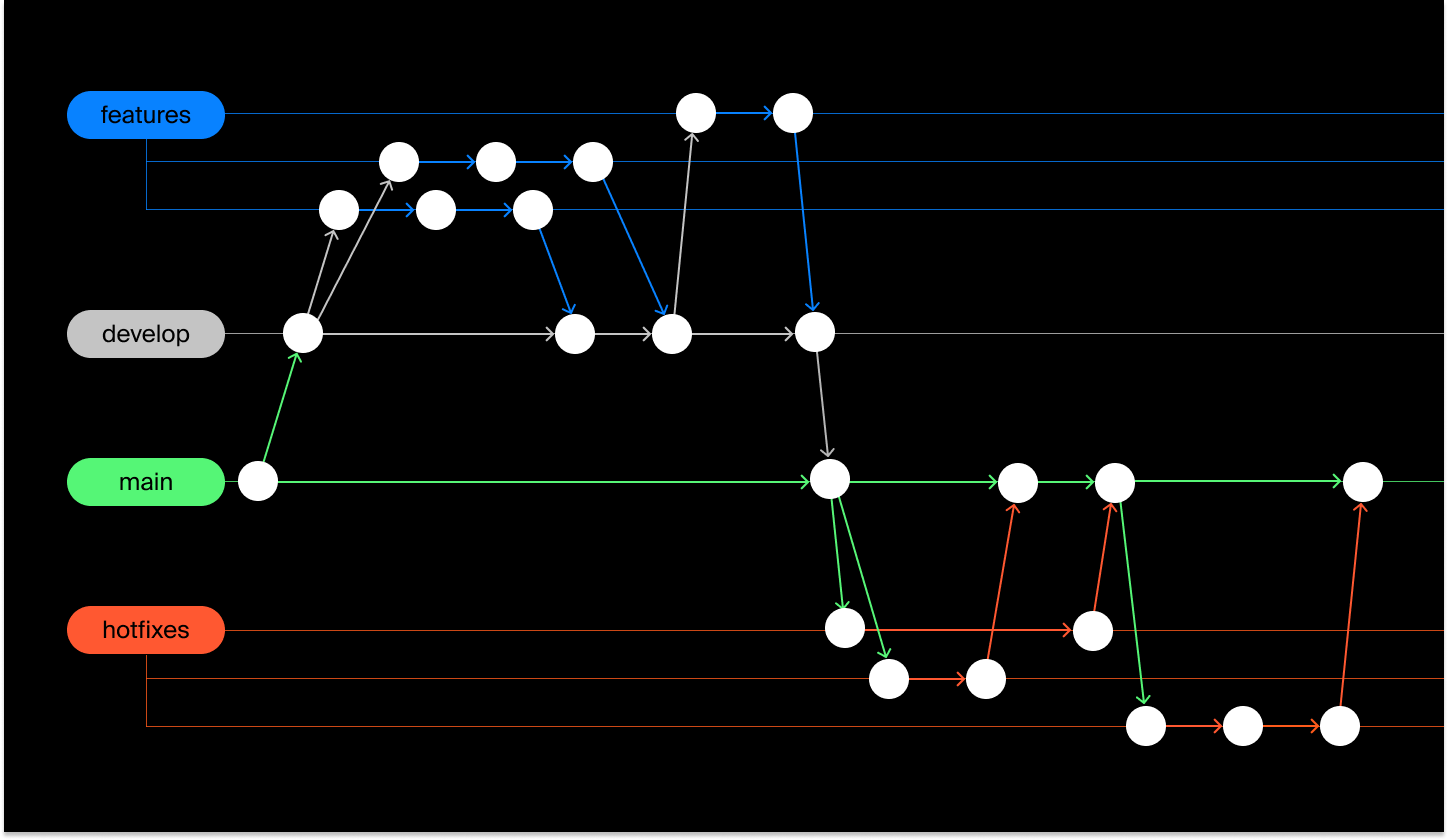
formElement.addEventListener('submit', formSubmitHandler);

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

**3. Git**

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

Работа с «Гитом» 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 требования ревьюеров. 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.
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.
9. По результатам ревью создают новую ветку быстрых исправлений hotfix/bug-name. В ней исправляют все проблемы, выявленные на ревью.
10. После завершения разработки код из hotfix/bug-name мержится в main.
11. Ветку hotfix/bug-name удаляют.
12. Если работа все ещё не принята, повторяют шаги, начиная с 9-го.

Фотографии можно брать не из макета, а с сайта [https://unsplash.com](https://unsplash.com/) — это коллекция бесплатных фотографий, которые можно использовать, не беспокоясь об авторских правах.