

Проектная работа «Посмотри в окно»

В последнем уроке этой темы, «Сдача проектной работы», к вам в аккаунт GitHub автоматически скопируется стартовый репозиторий. Поэтому лучше всего двигаться в таком порядке: ознакомиться с описанием проекта в этом уроке, чек-листом — в следующем, получить заготовку — в последнем. Когда склонируете к себе на компьютер заготовку, вернитесь в этот урок и приступайте к выполнению проекта.

В этой проектной работе вам предстоит написать CSS для работающего приложения. В стартовом репозитории вы получите много готовых файлов: всю HTML-разметку, CSS для компонентов прелоадера и вывода ошибки, файлы шрифтов и стили для них, файл со скриптом с описанной логикой работы приложения. Ваша задача — дополнить файл `style.css` так, чтобы интерфейс приложения соответствовал [макету](#).

Открыв страницу из заготовки, вы увидите подгруженные видео и даже сможете выполнить поиск. Вот краткое описание функциональности приложения:

- При загрузке страницы скрипт получает данные из внешнего источника, отрисовывает пять карточек с видео и кнопку, чтобы подгрузить дополнительные карточки. Ещё он подставляет адрес первого из загруженных роликов в тег `<video>` внутри крупного блока на странице.
- Скрипт отслеживает клик по карточкам и переключает текущее видео в зависимости от выбранной карточки.
- А ещё скрипт следит за отправкой формы. После отправки он ищет в базе данных совпадения по введённым параметрам и перерисовывает страницу с данными, полученными из нового запроса.

При работе над вёрсткой вам нужно сохранить эту функциональность и не навредить ей. Поэтому пишите в основном CSS, не меняйте уже написанные имена классов, старайтесь минимизировать свои действия с HTML-кодом. Сейчас его достаточно, чтобы реализовать задачу.

Устройство HTML-кода

Присмотритесь к устройству файла `index.html`. В нём есть кое-что незнакомое для вас. Это теги `<template>`. Мы расположили их внизу `<body>`, чтобы все они были собраны в одном месте. Теги `<template>` — это шаблоны для разметки, которая появится на странице только по запросу. При загрузке страницы браузер не отрисовывает содержимое этих тегов, но он знает об их существовании. В нужный момент программист может создать сколько угодно экземпляров разметки из этого тега на JavaScript, подставить внутрь этой разметки любые значения текста и атрибутов, а потом вставить полученный экземпляр разметки в нужную часть страницы.

Мы создали четыре таких шаблона:

- Шаблон пункта списка с карточкой видео — в него мы подставляем данные и создаём столько карточек на странице, сколько нужно, добавляя их в список.
- Шаблон прелоадера — пока данные загружаются, мы создаём экземпляры прелоадера и вставляем в зоны, куда будет вставлен контент. Когда контент готов, мы убираем прелоадеры.
- Шаблон кнопки «Показать ещё» — мы подставляем эту кнопку в конец списка карточек каждый раз, когда доступны дополнительные видео. Нажатие на эту кнопку подгружает пять дополнительных карточек.
- Шаблон блока с ошибкой — в зависимости от типа ошибки, мы выбираем текст для неё, создаём экземпляр шаблона с текстом и вставляем на страницу.

Теги `<template>` нужны для разработки на JavaScript и создания интерактивности. При вёрстке вам нужно понимать одно: стилизация экземпляров, которые созданы из шаблонов, не отличается от стилизации любых HTML-элементов. Вы всё так же можете использовать селекторы для элементов, которые описаны внутри шаблонов, и писать для них CSS-правила, — они применятся к экземплярам.

Вот пример:

HTML

```
<template class="more-button-template">
  <li><button class="button more-button">Показать еще</button></li>
</template>
```

Для такой конструкции в HTML сработает следующий CSS. Он применится к любой из кнопок «Показать ещё» на странице, где бы она ни появилась.

CSS

```
.more-button {
  background-color: transparent;
}
```

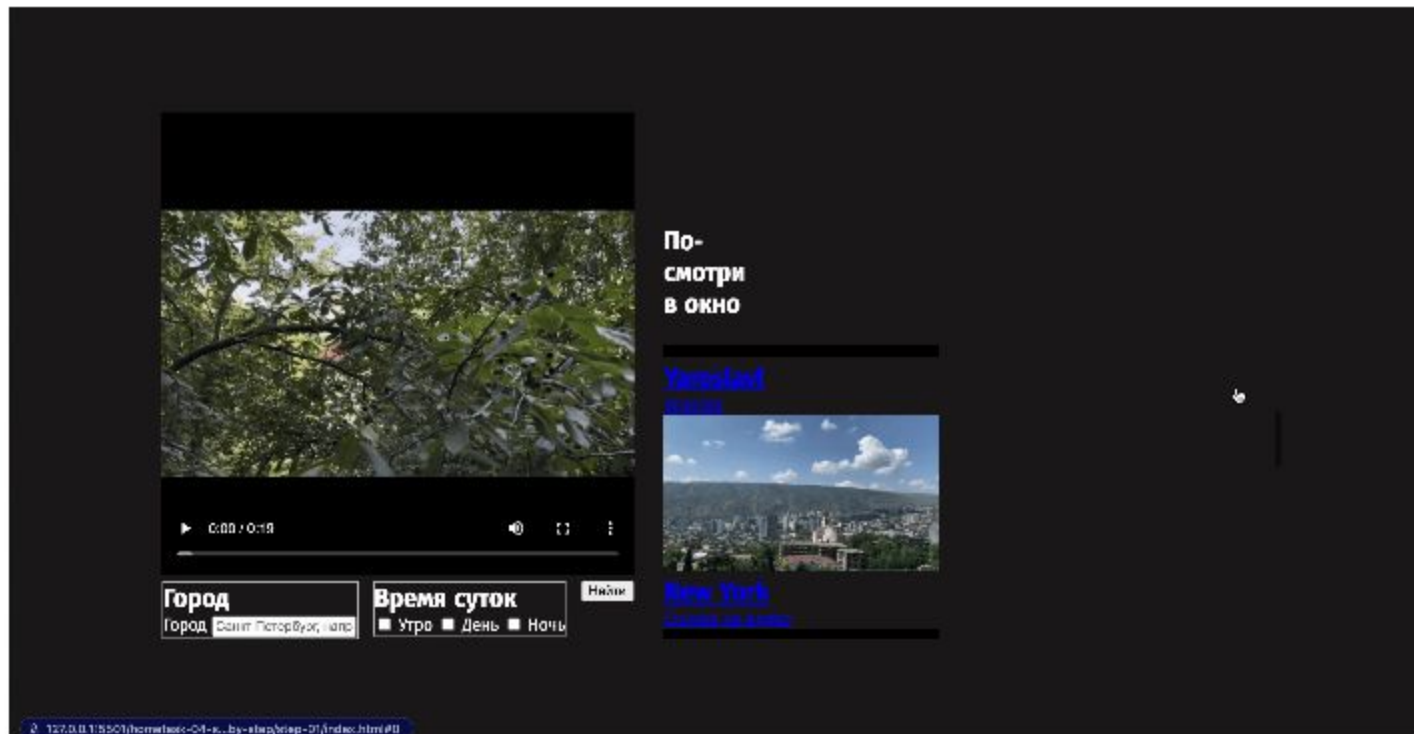
Шаг 1. Лейаут

В файле `style.css` уже есть часть базовых стилей. Они сбрасывают отступы и некоторые браузерные стили, добавляют минимальный набор стартовых правил. Обратите внимание, что в этой работе мы начали использовать логические свойства. Продолжайте эту практику. Обратите внимание, что мы зафиксировали размеры сайта. Адаптивность у нас ещё впереди. Весь макет можно верстать в абсолютных единицах измерения.

Первым делом попробуйте собрать общий лейаут страницы. Сделайте так, чтобы взаимное расположение больших секций сайта примерно соответствовало макету. Этого можно добиться разными способами. Вам понадобится либо

Первым делом попробуйте собрать общий лейаут страницы. Сделайте так, чтобы взаимное расположение больших секций сайта примерно соответствовало макету. Этого можно добиться разными способами. Вам понадобится либо флексбокс, либо гриды в зависимости от выбранного подхода.

Попробуйте на этом этапе прийти примерно к такому результату:



Задачи, которые вам предстоит решить для этого:

Задачи, которые вам предстоит решить для этого:

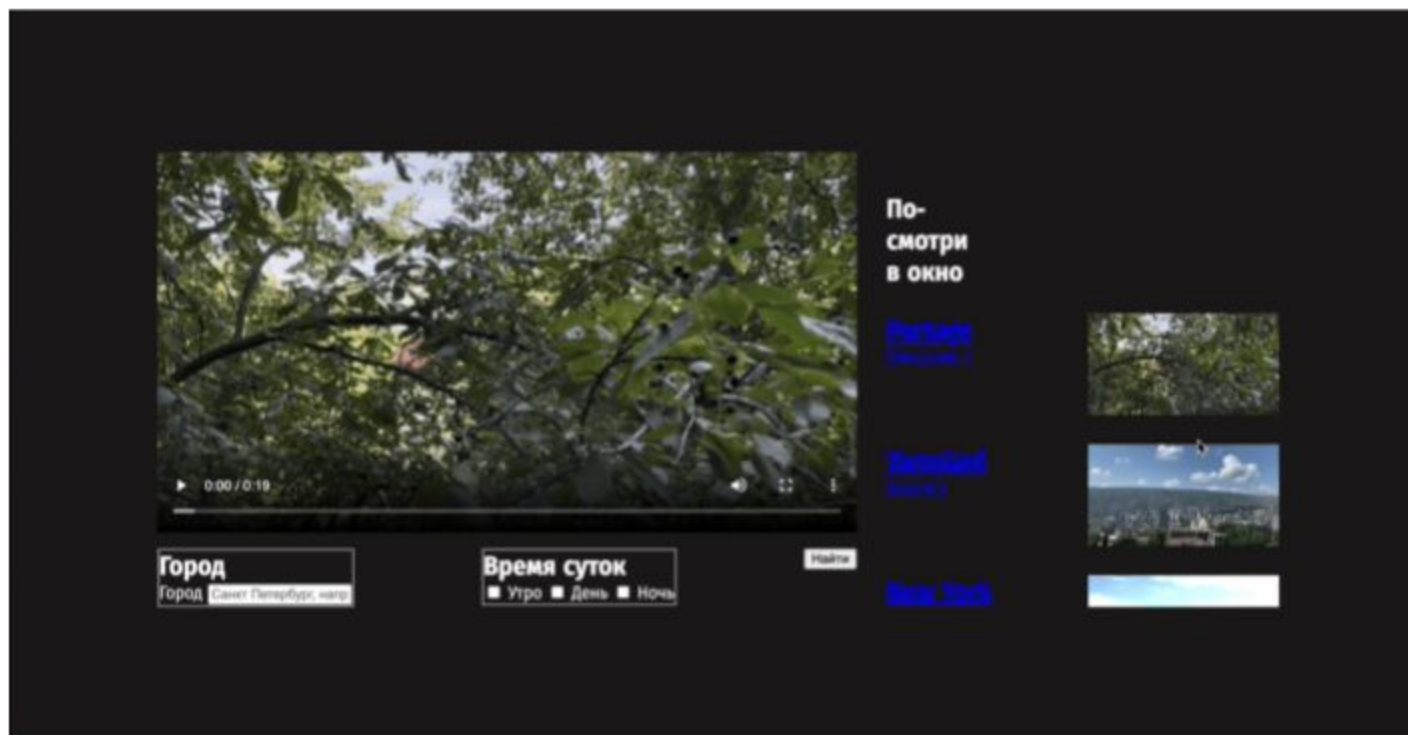
1. Расположить блок `.content` посередине `.page`.
2. Расположить элементы `.content` в строку, прижав к низу вертикальной оси.
3. Задать горизонтальные размеры `.content` и отступы между его элементами.
4. Расположить элементы `.search-form` в строку и пропорционально распределить между ними свободное пространство, прижать их к началу контейнера.
5. Установить вертикальный размер для `.content__list` и добавить скролл при переполнении.
6. Растянуть `.content__details` во всю доступную ширину и высоту контейнера и добавить отступы между заголовком и зоной с карточками.

Шаг 2. Уточнение размеров

Вторым этапом предлагаем вам чуть лучше подогнать ряд элементов под размеры, чтобы страница обрела близкий к задумке вид. Это позволит детальнее прорабатывать компоненты уже в близком к готовности лейауте.

Попробуйте добиться примерно такого состояния вёрстки:

Попробуйте добиться примерно такого состояния вёрстки:

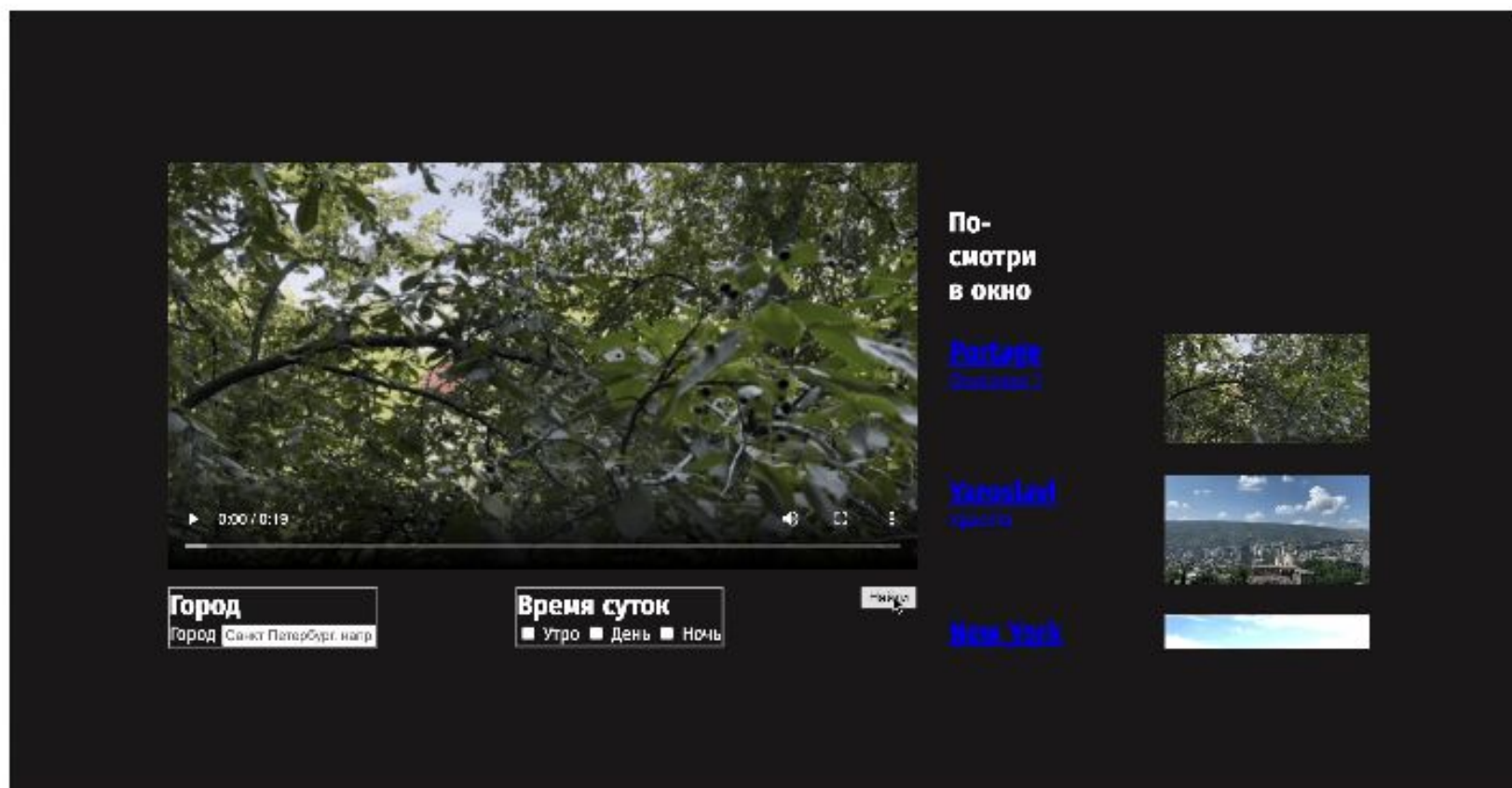


Задачи, которые встанут перед вами:

1. Задать размеры контейнеру `.result__video-container` и добавить отступ снизу.
2. Замостить видео в этот контейнер так, чтобы оно занимало всю ширину и высоту и не выходило за пределы.
3. Создать вертикальные отступы между элементами `.content__list-item` (у всех, кроме последнего).
4. Расположить элементы внутри контейнера `.content__video-card` в строку, прижав к началу контейнера по дополнительной оси.
5. Задать размеры `.content__video-card-thumbnail` и замостить в них изображение.
6. Сделать так, чтобы `.content__video-card-description-container` занимал всё доступное пространство, не занятое картинкой.

Шаг 3. Позиционирование прелоадеров

Если к этому моменту вы ничего не делали, чтобы прелоадеры оказались на своих местах, сейчас любое действие с данными вызывает их и они перекрывают всю страницу. Вот так:




Прелоадеры должны появляться в двух местах:

- поверх `.result__video-container`;
- поверх `.content__list`.

Спозиционируйте эти элементы относительно. Тогда прелоадеры станут вставать в них. Это происходит оттого, что прелоадеры заданы абсолютным позиционированием и ищут спозиционированного родителя.

Вот результат, который вам нужен:



0:00 / 0:19

Город

Город Санкт Петербург, напр

Время суток

■ Утро ■ День ■ Ночь

Найти

По-смотри в окно

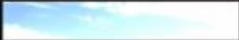


Portage

Самолет 3

Yaroslavl

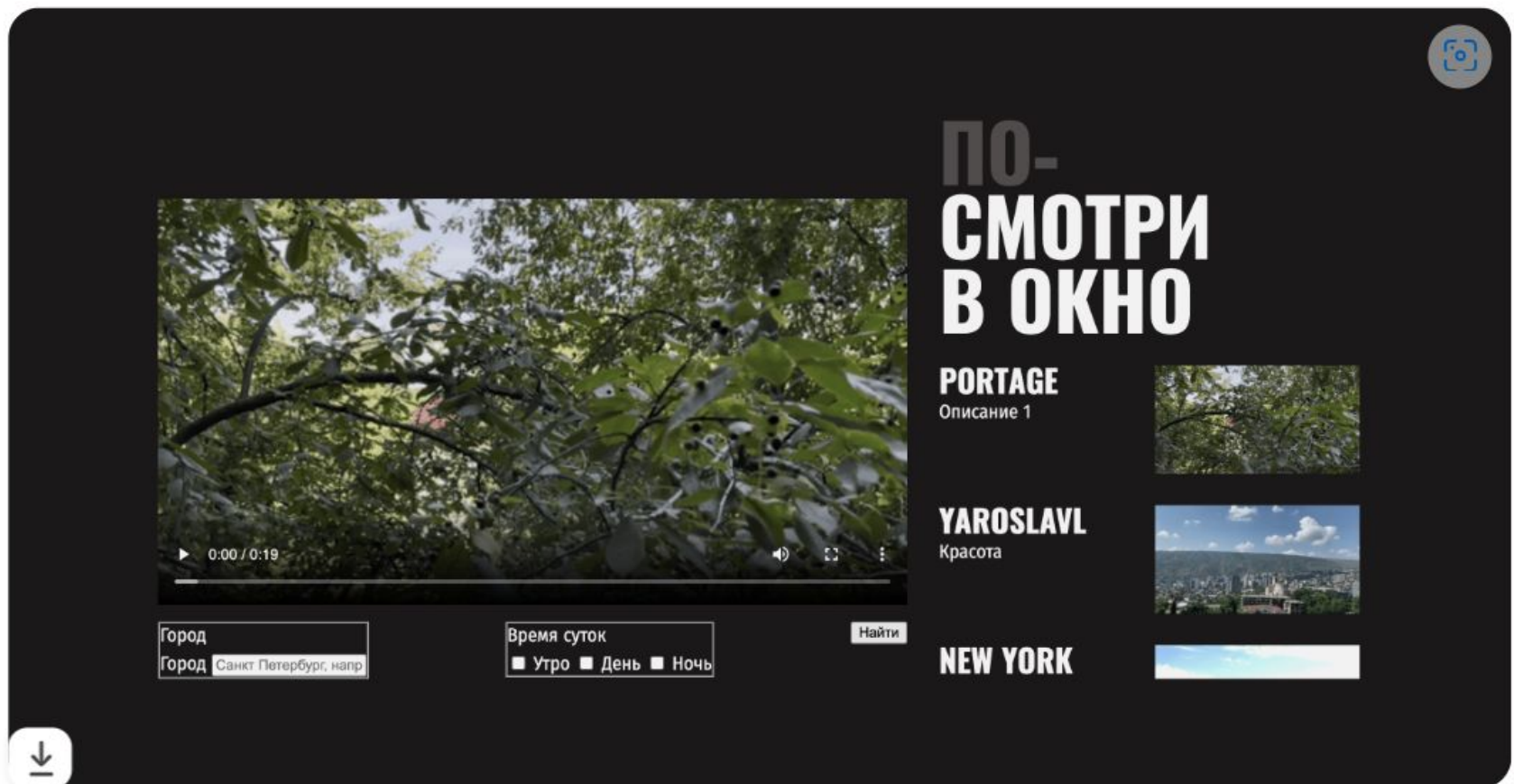
Видео

New York



Шаг 4. Стили текстов

Следующим шагом предлагаем вам привести текстовые элементы сайта в соответствие с макетом. Сейчас для всего сайта заданы глобальные настройки шрифта, но для некоторых элементов они должны отличаться. Попробуйте прийти к такому результату:



Вот какие задачи предстоит решить:

1. Оформить текст элементов `.search-form__fieldset-title`. Обратите внимание на вес шрифта, размер, отступы.
2. Оформить элемент `.title` и вложенный в него `.content__accent`.
3. Убрать дефолтное подчёркивание и заменить цвет текста у ссылки `.content__card-link`.
4. Оформить текст `.content__video-card-title`.
5. И присмотреться к высоте линии в `.content__video-card-description`.

Обращайте особое внимание на интерлиньяж. В этом макете много особенностей, связанных с ним. Отчасти из-за необычного использования высоты линии макет приобретает ту брутальность, которую задумал дизайнер.

Шаг 5. Элементы формы и кнопки

Для работы с элементами формы нужно активно применять паттерн `.visually-hidden`. С его помощью вам нужно скрыть лишние лейблы и браузерные чекбоксы, чтобы потом нарисовать поверх них свои.

Добавьте в ваш код класс `.visually-hidden`. В HTML он уже добавлен необходимым элементам. Остались стили. Вот они:

```
.visually-hidden {  
  position: absolute;  
  inline-size: 1px;  
  block-size: 1px;  
  overflow: hidden;  
  clip: rect(0 0 0 0);  
  clip-path: inset(50%);  
  white-space: nowrap;  
}
```

CSS

Поля формы

Самостоятельно сверстать кастомные элементы форм — важная часть этой проектной работы. Поэтому подсказок будет немного.

На этом этапе не верстайте состояния инпута и чекбоксов. Отложите это до следующего шага. Пусть чекбоксы пока не нажимаются, там есть особенности, на которые мы вам укажем.

Вот несколько полезных замечаний для вёрстки текстового поля ввода и чекбоксов:

1. Сбросьте `border` у `fieldset`.
2. Вам нужно расставить элементы внутри `.search-form__label` в линию. И при этом подогнать ширину контейнера под размер внутренних элементов. Для этого пригодится правило `width: fit-content;`.
3. Мы рекомендуем установить на кликабельные элементы `cursor: pointer;`.
4. При вёрстке текстового поля обращайтесь внимание на управление размерами, границей, цветом текста, семейством шрифта, размером шрифта, фоном. Некоторые из этих свойств не наследуются.
5. Обёртка `.search-form__checkbox-list` создана специально для того, чтобы выстроить чекбоксы в линию.
6. Текстовому полю нужно задать свойство `appearance: none;`. Так в разных браузерах будут нужные вам стили. Это трюк.
7. Когда будете рисовать `.search-form__pseudo-checkbox`, учитывайте, что в нём появится элемент, который должен оказаться точно по центру. Если присмотритесь к макету, заметите, что внутренний квадрат включённого чекбокса чуть меньше самого чекбокса. Это мы сверстаем отдельно следующим шагом.

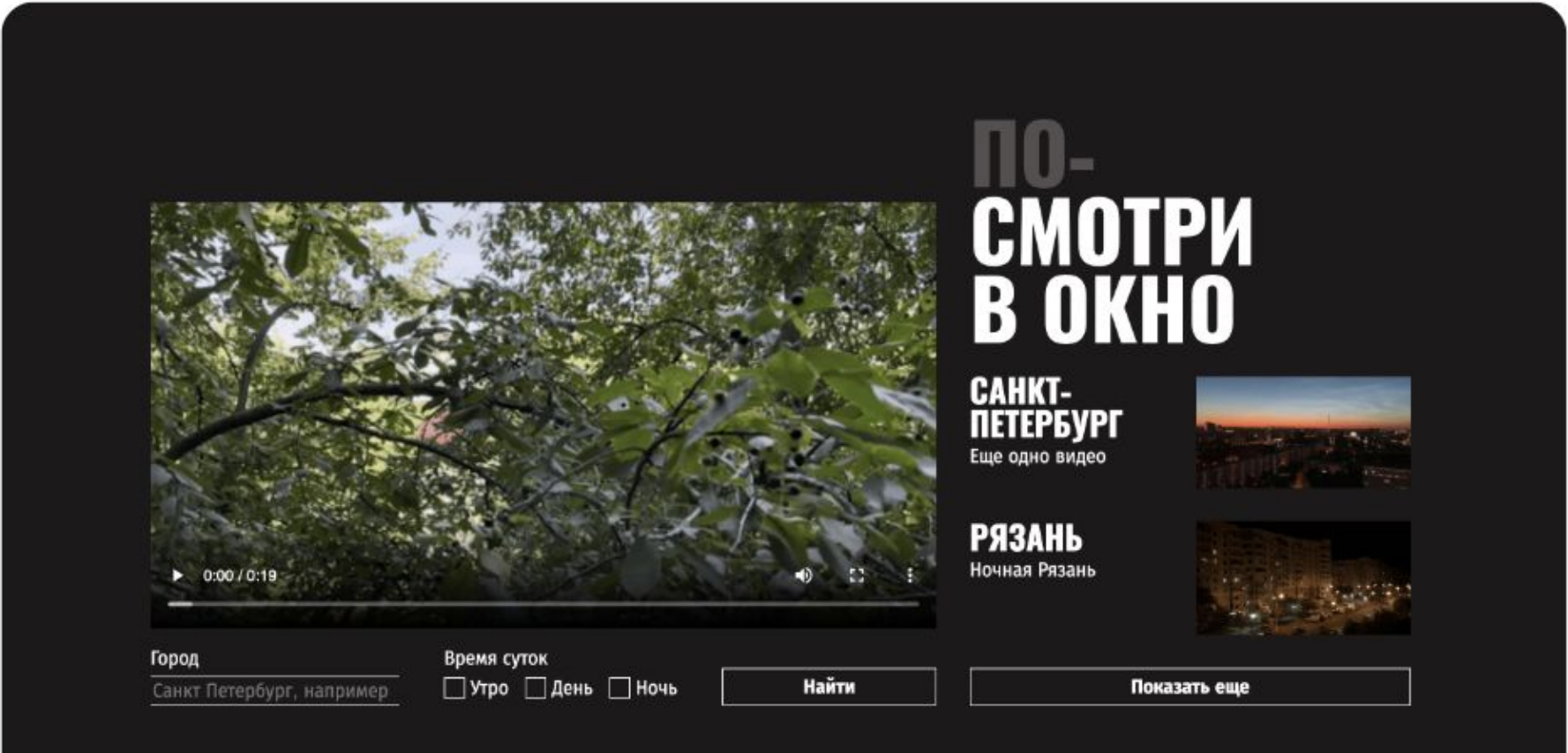
Кнопки

На странице две кнопки: одна в форме, другая в списке карточек. У них есть общие стили, но у кнопки в списке карточек есть свои собственные. Чтобы всё сделать как надо, созданы три класса: `button`, `search-form__submit-button` и `more-button`. Подумайте, какие стили должны быть в каждом из них.

Из подсказок:

- Кнопке тоже можно задать `cursor: pointer;`.
- Кнопка в форме прижата к низу своего контейнера, подумайте, как этого добиться.

Вот примерный результат этого шага:



Шаг 6. Состояния элементов формы и кнопок

У элементов формы очень много необычных состояний. Поработать над ними — отдельная большая задача.

Текстовое поле ввода

У этого элемента нужно сбросить обводку в состоянии фокуса и вместо этого делать границу у лейбла, который его окружает. Лейбл должен получать границу только когда у поля находится в состоянии `focus-visible`, но не `focus`. Чтобы такое реализовать, можно применить хитрую комбинацию псевдоклассов — `:has(:focus-visible)`.

Активный чекбокс

На макете при нажатии чекбокса внутри появляется квадрат. Он чуть меньше чекбокса. Мы предлагаем вам реализовать его псевдоэлементом для `.search-form__pseudo-checkbox`. У вас получится непростая комбинация селекторов «псевдоэлемент для первого соседа зажатого чекбокса». Подумайте, как такое описать.

Фокус в чекбоксе

Если вы применяли псевдокласс `:has()` для текстового инпута, здесь тоже будет работать это правило. Вам не придётся прописывать отдельно состояние `focus` и `focus-visible` для чекбокса. Но если вы пошли другим путём, задумайтесь, как добиться того, чтобы состояние `focus` не создавало обводку у родительского элемента, а состояние `focus-visible` создавало.

Подчёркивание текста у чекбокса при наведении на лейбл

Обратите внимание, что подчёркивание должно появляться при наведении на любую точку лейбла, а не только на сам текст.

Состояния кнопок

У всех кнопок на странице три состояния: `:hover`, `:active`, `:focus-visible`.

Состояние `:focus` должно быть сброшено. Эти правила применяются ко всем кнопкам на странице.

Шаг 7. Состояния карточек

У карточек, которые выводятся справа, тоже много состояний.

Текущая карточка

Первая карточка должна быть активной, а при переключении активная карточка должна выделяться. Это поведение уже написано на JavaScript. Но в CSS должны оказаться соответствующие стили для селектора `.content__card-link_current`. Есть одна особенность: чтобы фон применился, ссылку `.content__card-link` нужно сделать блочным элементом.

Другие состояния карточки

У каждой карточки доступны такие состояния: `:hover`, `:active` и `:focus-visible`. Чтобы состояние `:focus-visible` работало корректно, придётся сбросить обводку на состоянии `:focus`. Все стили вы легко скопируете из макета.

Есть только одна хитрость. Чтобы обводка в состоянии `:focus-visible` была видна целиком, задайте небольшой `margin` элементу `.content__card-link`. Хватит двух пикселей со всех сторон.

Вот и всё!

В этом проекте вы тренируете очень важный навык верстальщика — работу с частично готовым модулем. В реальных проектах вам часто придётся стилизовать готовые плагины, менять темы, созданные другими разработчиками, дописывать стили в уже готовые модули. Здесь вы в той же ситуации. Приложение уже работало, но выглядело плохо, а вы сделали хорошо.

Чек-лист проектной работы «Посмотри в окно»

В чек-листе собраны требования, которые нужно соблюсти, чтобы пройти автоматическое тестирование и профессиональное ревью.

Работа отклоняется от проверки ревьюером

Пока эти ошибки не будут исправлены, ревьюер не примет работу на проверку и не даст обратную связь:

- Пять или больше элементов макета отсутствуют.
- Не хватает секции.
- Порядок блоков не соответствует брифу.
- На повторных итерациях не исправлены критические замечания.
- Работа содержит вопросы или просьбы о помощи к ревьюеру.

Критические требования

Без соблюдения этих требований ваш проект не пройдет автоматическое тестирование. Если по какой-то причине автотесты пропустят работу (такое может случиться), эти пункты обязательно попросит исправить ревьюер.

Отображение в браузере

- Проект визуально соответствует макету:
 - есть все секции, блоки и элементы макета, и они корректно отображаются в Firefox, Google Chrome или Yandex Browse,
 - отличие не более 10% при проверке системой скриншот-проверки автотестов,
 - отличие не более 5px по вертикали и 3px по горизонтали при проверке ревьюером.

Рекомендуем пользоваться плагином Pixel Perfect для доводки вёрстки.

- Сетка макета не сбита: например, при сжатии страницы сохраняется расположение элементов относительно друг друга на странице, содержимое не выпадает за пределы своего блока, никакие блоки не наезжают друг на друга.
- Страница не прокручивается по горизонтали и не появляется других лишних полос прокрутки при ширине экрана, которая больше или равна минимально допустимой (согласно рекомендациям к проекту).
- Свёрстаны все состояния элементов страницы описанные в рекомендациях и макете.

Соблюдены рекомендации к текущему проекту

- Вёрстка проекта не вредит функциональности:
 - карточки проекта появляются во всех предусмотренных ситуациях,
 - работают прелоадеры и заглушки ошибок,
 - работает поиск,
 - видеоролики подставляются в основной блок по клику.
- Позиционирование `absolute` в файле `style.css` применяется только для блока `.visually-hidden`. Остальной лейаут выполнен с применением гибких раскладок.
- Изображения и видео подогнаны под заданные размеры через `object-fit`.
- Учтены особенности поведения интерактивных элементов в состояниях `:focus` и `:focus-visible`. Например, в состоянии `:focus-visible` у элементов формы лейбл обводится, а в состоянии `:focus` — не обводится.
- Состояния выбранных чекбоксов реализованы через псевдоэлементы.
- Все элементы доступны при навигации с клавиатуры, они визуально выделены при работе с ними.
- Подчеркивание у кнопок и чекбоксов появляется при наведении на всю активную зону (включая сам чекбокс), а не только на текст.

CSS

- Аккуратное форматирование кода:
 - между селекторами и открывающими скобками стоит пробел,
 - каждое правило начинается с новой строки,
 - стоят точки с запятой,
 - закрывающие скобки вынесены на отдельную строку.
- Соблюдены требования к именованию CSS классов:
 - для CSS классов выбраны подходящие по смыслу имена,
 - отсутствует слитное написание слов в именах классов,
 - единообразное разделение слов в именах CSS-классов во всем проекте, например используется только `kebab-case` — разделение слов знаком `-`, или `camelCase` — разделение слов регистром символов,
 - в именах не используется транслитерация и сокращения, которые не являются общеупотребимыми.

Если в брифе к проекту описаны имена классов, то старайтесь использовать их.
- У `body` и типовых элементов сбрасываются браузерные отступы.
- Для всех элементов сайта корректно заданы:
 - цвета фона,
 - размеры,
 - межстрочные расстояния,
 - межбуквенные расстояния и расстояния между словами при необходимости.

- Корректно используются шрифты:
 - разные форматы шрифтов подключаются в правильном порядке — от самых современных к более старым с указанием формата как свойства CSS;
 - указаны альтернативные шрифты;
 - семейство, вес, начертание и размеры шрифтов во всех элементах страницы соответствуют заданным в макете.
- Правильно организован лейаут страницы:
 - для организации лейаута ключевых блоков использован `flex` или `grid`;
 - корректно отцентрированы необходимые элементы на странице,
 - не задана фиксированная высота и ширина элементов там, где их можно не использовать или применено минимальные или максимальные значения. Блок растягивается, если в него вставлено в два-три раза больше текста;
 - абсолютное позиционирование используется только там, где нельзя применить статичное или относительное позиционирование;
 - контекст позиционирования указан корректно (например, `position: relative` у нужного элемента);
 - корректно задано свойство `z-index`, нет спрятавшихся элементов;
 - нет лишних отступов у первых и последних элементов в списках и блоках с абзацами.

- Нет пустых CSS-правил.
- Нет дублирующихся селекторов.
- Нет дублирующихся свойств внутри CSS-правил.
- Стили одинаковых элементов не дублируются, а переиспользуются.
- Не используются инлайновые стили в HTML.

Хорошие практики

В этом списке собраны приёмы, которые помогут сделать проект ещё лучше. На данном этапе их использовать необязательно — автотесты и ревьюер примут работу без них. Однако такие хорошие практики помогут вам сделать портфолио более профессиональным. Опытные верстальщики обратят на них внимание.

- В разметке нет лишних обёрток `<div>` у элементов.
- В разметке у ссылок нет пустых атрибутов `href`.
- У всех изображений задан атрибут `alt` с описанием на языке страницы.
- Всем изображениям заданы ограничения размеров.
- Элементам не добавлены классы, которые не используются.
- В стилях не указаны неработающие свойства (например, не заданы размеры строчных элементов).
- Для всех элементов на странице переопределено дефолтное значение свойства `box-sizing`.
- При организации внутренней геометрии блоков внутренние отступы задаются через `padding`.
- Раскладки `flex` или `grid` не применяются без необходимости там, где можно обойтись дефолтным отображением.

- Для задания отступов между элементами использован `gap` вместо `margin`.
 - Нет чрезмерной стилизации. Например, одинаковые размеры прописаны и для обёртки, и для изображения внутри.
 - Интерлиньяж задан в относительных единицах измерения.
 - Если в проекте используется методология БЭМ, то нет нарушений соглашения по именованию классов и требований методологии. Например:
 - применено единообразное разделение имён блоков, элементов и модификаторов во всем проекте;
 - элемент не используется вне своего блока;
 - нет элементов элементов;
 - класс модификатор не используется без указания блока или элемента, который он модифицирует;
 - классы модификаторы описывают только модифицированное значение, остальные описаны в стилях блока;
 - внешняя геометрия блоков задана через миксы.
 - В элементах формы при необходимости используются `inherit` или `currentColor`.
 - Для предсказуемого поведения инпутов в разных браузерах использован `appearance: none;`.
-

- Вместо `overflow: scroll` использован `overflow: auto`, чтобы не отображался ненужный скролл.
- Вид курсора меняется при наведении на интерактивные элементы.
- Для наилучшего позиционирования изображения при задании свойства `object-fit` задаётся также подходящее значение свойства `object-position`.
- Для стилизации интерактивных элементов используется `outline`, а не `border` (`outline` не влияет на размеры элемента в потоке).
- CSS-правила в коде расположены в примерном соответствии с позицией элементов в DOM.
- Для стилизации используются только селекторы классов, псевдоэлементов и псевдоклассов, если речь не идёт о сбросе браузерных дефолтных стилей.
- Вместо физических CSS-свойств используются логические CSS-свойства.

