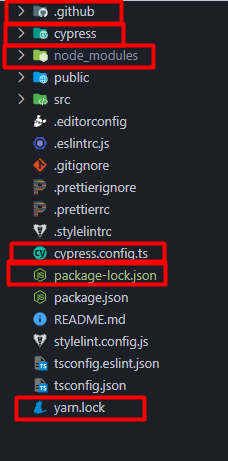
**С чего начать**

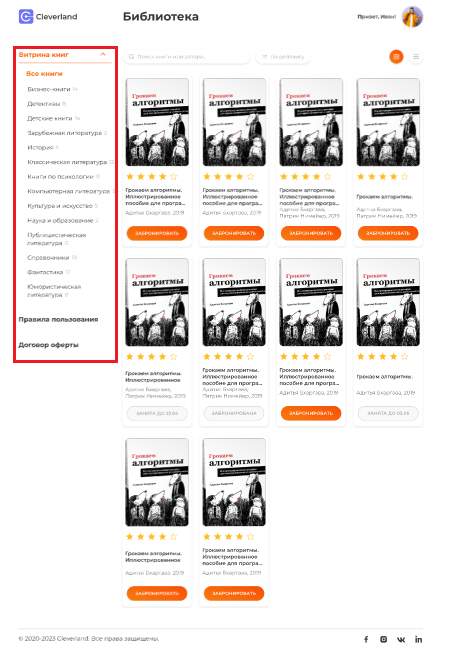
****

1. Все файлы, за исключением выделенных на данном скрине (файлы и папки могут отличаться), нужно перенести во второй спринт из первого
2. Для корректной работы приложения файлы yarn.lock и package-lock.json должны отсутствовать перед установкой зависимостей.
3. Далее вам нужно установить зависимости (лучше всего это сделать командой yarn)
4. Вы прекрасны и готовы к выполнению 2 спринта.

**Burger-menu**

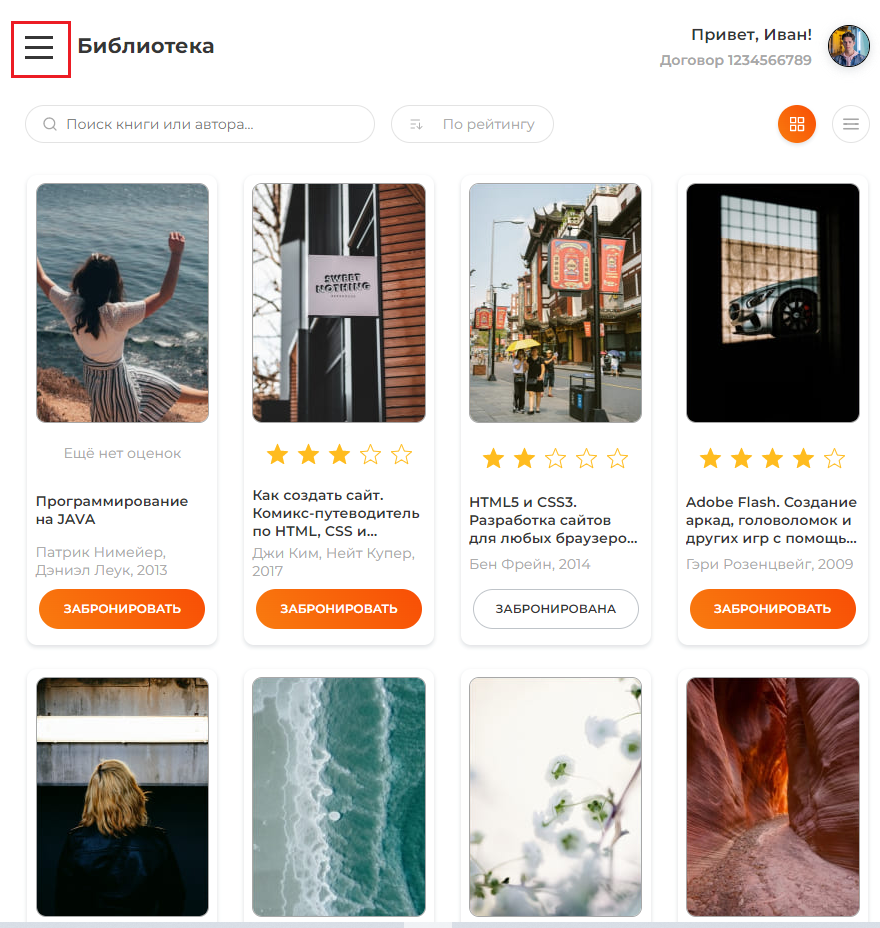
[Ссылка на макет](https://www.figma.com/file/d8LhhLjMkaTfPvAcYQULNv/Library---students-file?node-id=3601%3A32183&t=o8Uuc7S0CMknEpmU-0)

Все знают Burger-menu!) Знаменитые три полоски, позволяющие спрятать меню за экраном.

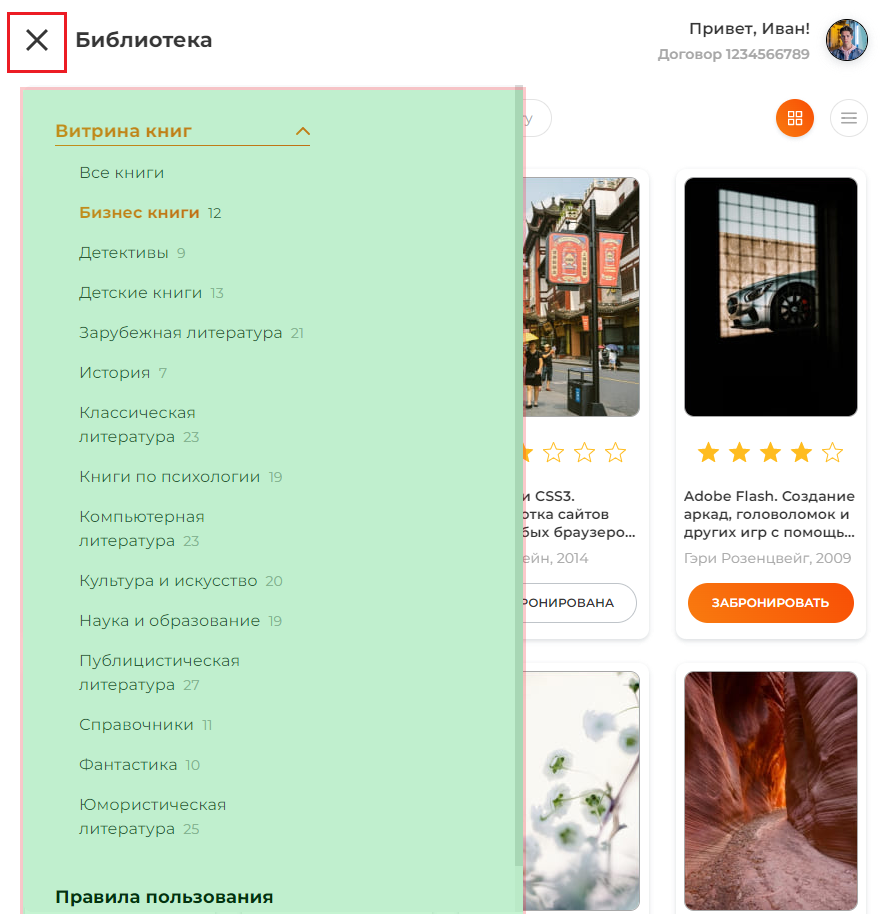


Оно особенно востребовано на мобильных устройствах.

В проекте его нужно сделать при таком разрешении экрана, при котором навигационное меню будет уменьшать ширину меню до момента невозможности отображать все элементы (инпут поиска, кнопки сортировки, изменения вида). И выглядит оно так:



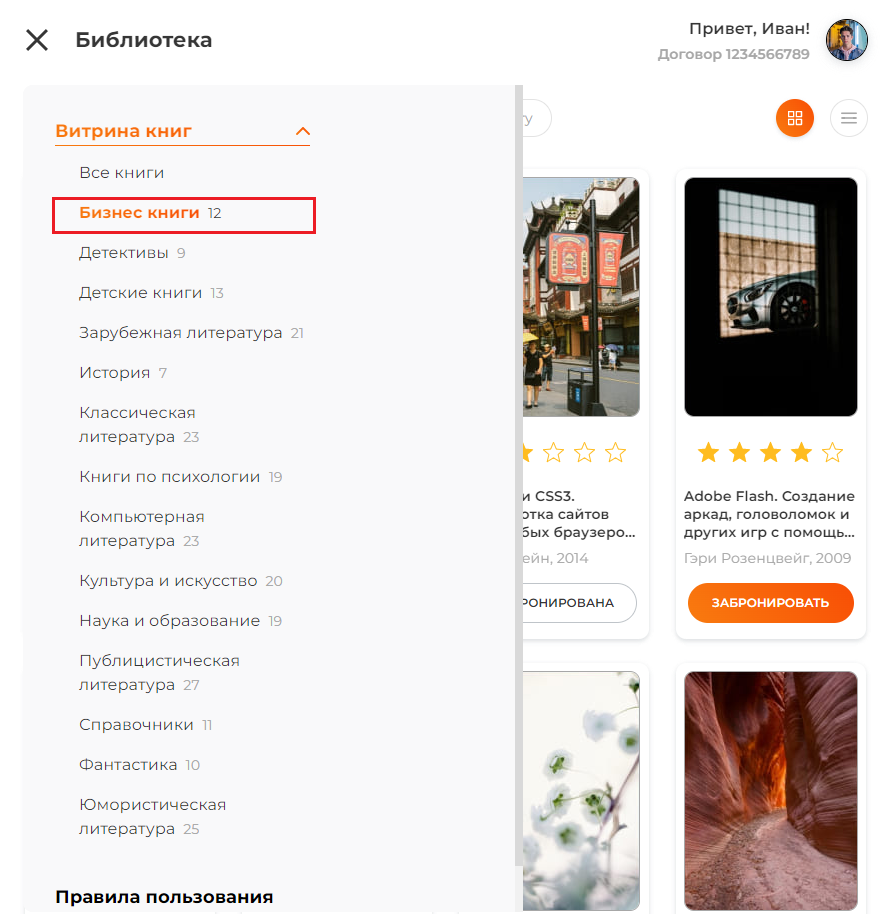
Принцип работы Burger-menu заключается в том, что при нажатии на него вместо трех полосок будет появляться крестик и адаптивное меню:



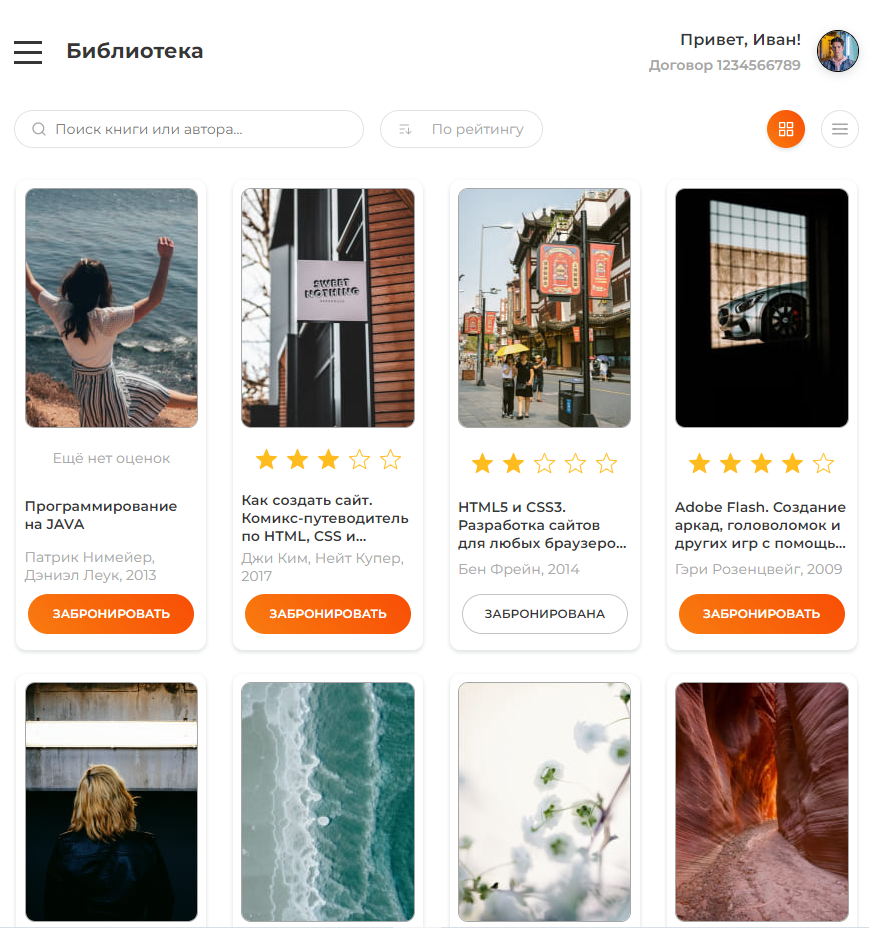
При нажатии на крестик и область, не относящуюся к навигационному меню (зеленая зона), все должно вернуться в первоначальное положение.

Адаптивное меню также должно быть кликабельным и переходить на страницы аналогично главному меню.

При нажатии на пункт адаптивного меню



оно исчезает и осуществляется переход на страницу выбранного жанра.



Изменение трех полосок на крестик.

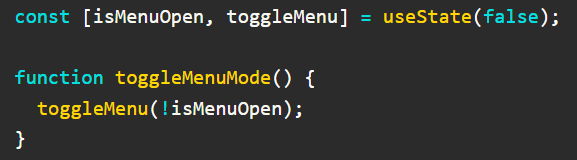
Чтобы три полоски изменялись на крестик, в коде использовалось добавление класса CSS. Это реализовывалось при помощи [classnames (библиотека для простого условного объединения имен классов)](https://www.npmjs.com/package/classnames) и хука useState.

При помощи useState мы будем менять состояние путем добавления класса.

**Например:**

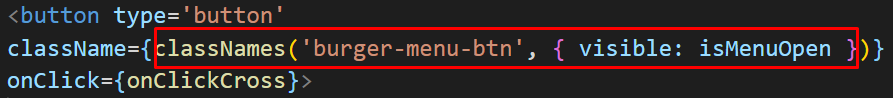
Иконка гамбургера состоит из 3-х span-элементов. При клике мы просто меняем длину верхней и нижней полоски. Когда пользователь нажимает на кнопку, мы вызываем функцию toggleMenuMode(), которая переключит состояние isMenuOpen. Если isMenuOpen истинно, тогда кнопке присваивается класс visible. Активное состояние превращает кнопку в крестик, что достигается двумя трансформациями – поворотом на 45 градусов и перемещением на несколько пикселей. В это же время средняя полоска скрывается.

Вот пример useState:



Вот пример добавления класса при помощи classnames:





## Выезжающее меню.

Кнопка-бургер у нас уже есть, осталось дело за малым – добавить само меню. Есть несколько способов реализации данной задачи. Мы использовали абсолютное позиционирование и положение относительно левого края экрана. Используя отрицательные значения left мы скрыли меню за пределами видимой части экрана. При нажатии на кнопку бургер-меню происходит добавление класса, содержащего новую позицию left, и соответственно перемещение меню из невидимой левой части в экрана в положение согласно макету.

Здесь также используются useState и classnames.

Как дополнительное задание (необязательное), но дающее большой плюс для вас, – это сделать так, чтобы при открытом адаптивном меню не скроллилась страница.

**Slider**

Слайдер – это скрипт, позволяющий осуществить циклическую смену содержимого, построенного на основе CSS и JavaScript. В проекте мы используем слайдер на основе библиотеки Swiper, который позволяет создать современный адаптивный и чувствительный к прикосновениям слайдер. Но если у вас есть опыт работы с другими слайдерами, то можете использовать их.

[Ссылка на документацию Swiper](https://swiperjs.com/react#styles)

Swiper

Превосходный слайдер, заточенный строго для работы с мобильными сайтами или веб-приложениями. Swiper обеспечивает аппаратное ускорение анимации на уровне нативного.

Установка производится через [yarn](https://yarnpkg.com/package/swiper) или [npm](https://www.npmjs.com/package/swiper) (в зависимости от пакетного менеджера, который вы используете в проекте).

Рассмотрим пример:

Стили CSS уже встроены в Swiper. Такой параметр, например, как slidesPerView отвечает за количество слайдов, на которое будет меняться изображение при переключении.

spaceBetween отвечает за расстояние между слайдами, если их сразу отображается несколько.

Это используется, например, вот здесь:



Что необходимо реализовать в нашем проекте:



Для реализации данного слайдера можно использовать одну и ту же картинку пять раз (например). Должно работать:

* touch (в Swiper он присутствует при использовании, так что и заморачиваться не надо!),
* при нажатии на миниатюру обложки происходит переход на данный слайд в верхнем элементе.

Swiper имеет отличную документацию, ссылка на которую приведена выше, так что рассказывать о каждом компоненте нет смысла.

Для большего понимания и изучения, как это все работает, вот [ссылка](https://swiperjs.com/demos) на примеры слайдеров, где можно посмотреть код их реализации.

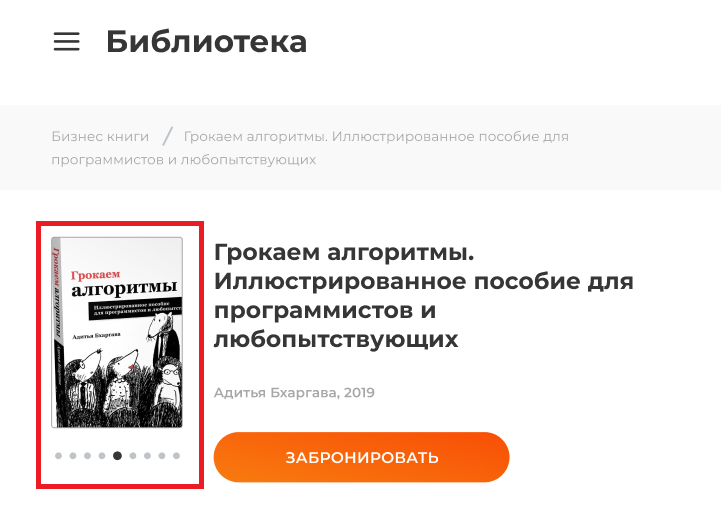
Если вы не хотите использовать Swiper, ничего страшного, главное, чтобы функционал был идентичным!

Как должно выглядеть переключение на слайдере:



При нажатии на миниатюру основной слайдер должен переключиться на соответствующую обложку.

***Обратите внимание!*** На разрешениях от планшета и менее свайпер выглядит иначе:

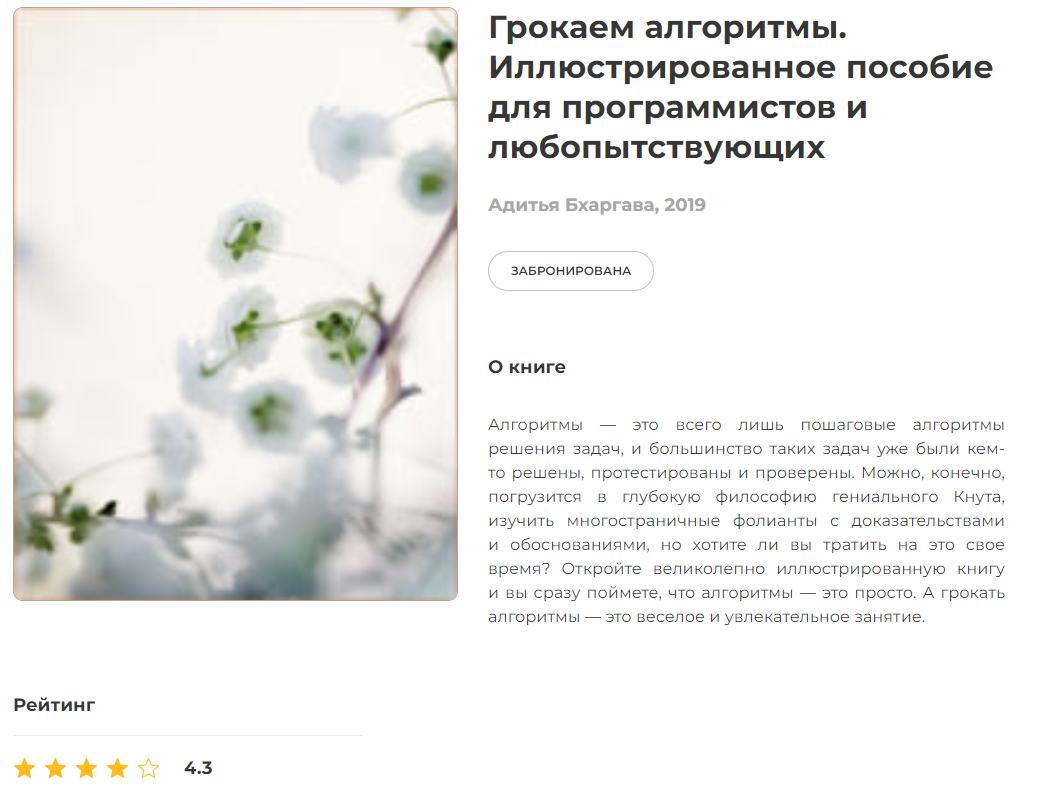


Для успешного прохождения тестов необходимо:

Первая книга **не должна содержать фото**, соответственно вид компонента с обложкой будет выглядеть следующим образом:



Вторая книга **должна содержать 1 фото,** соответственно вид компонента с обложкой будет выглядеть следующим образом:

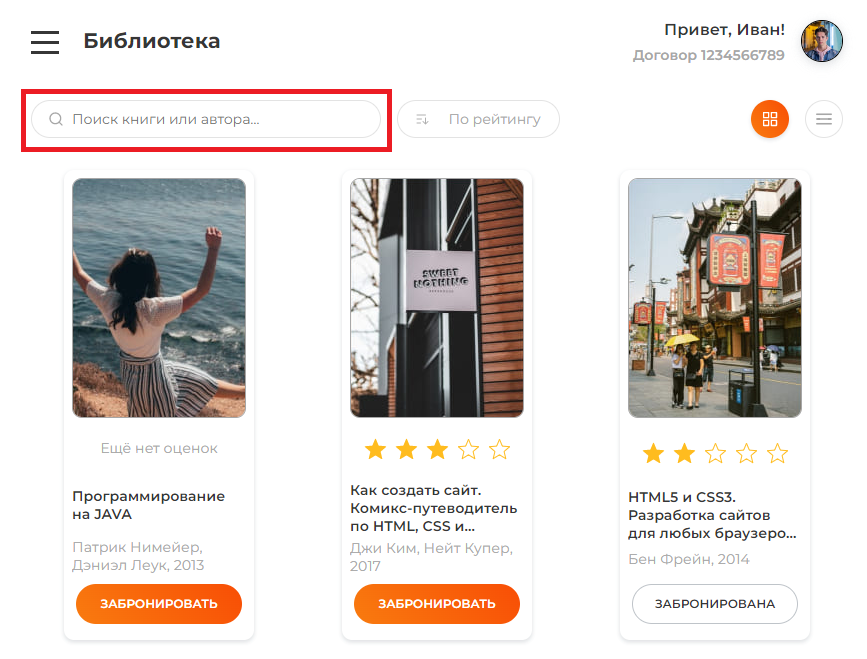


Третья книга **должна содержать 2 фото или более**, соответственно вид компонента с обложкой будет выглядеть следующим образом:

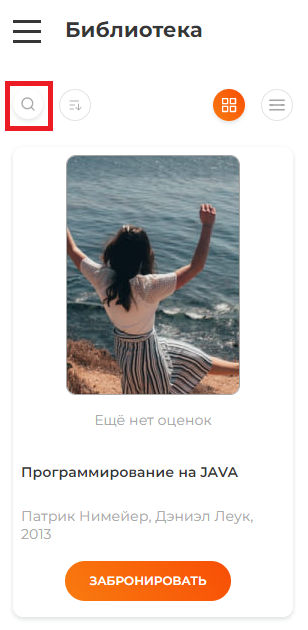


**Компонент Search**

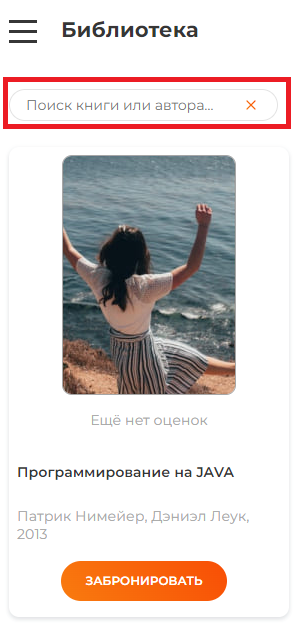
В данном спринте также необходимо реализовать изменение вида компонента поиска. В изначальном виде он выглядит так:



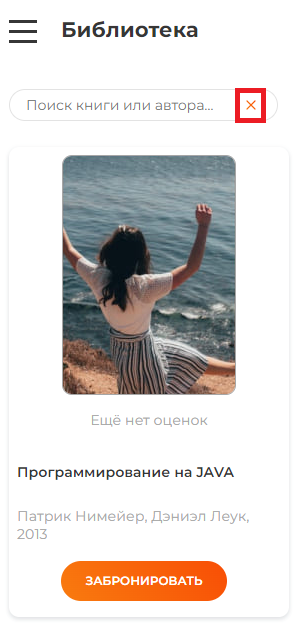
Когда контент перестает помещаться в ширину экрана - происходит переход и вместо инпута ввода появляется значок:



По нажатию на который, происходит скрытие всех значков в данном меню и появляется инпут ввода поиска книги:

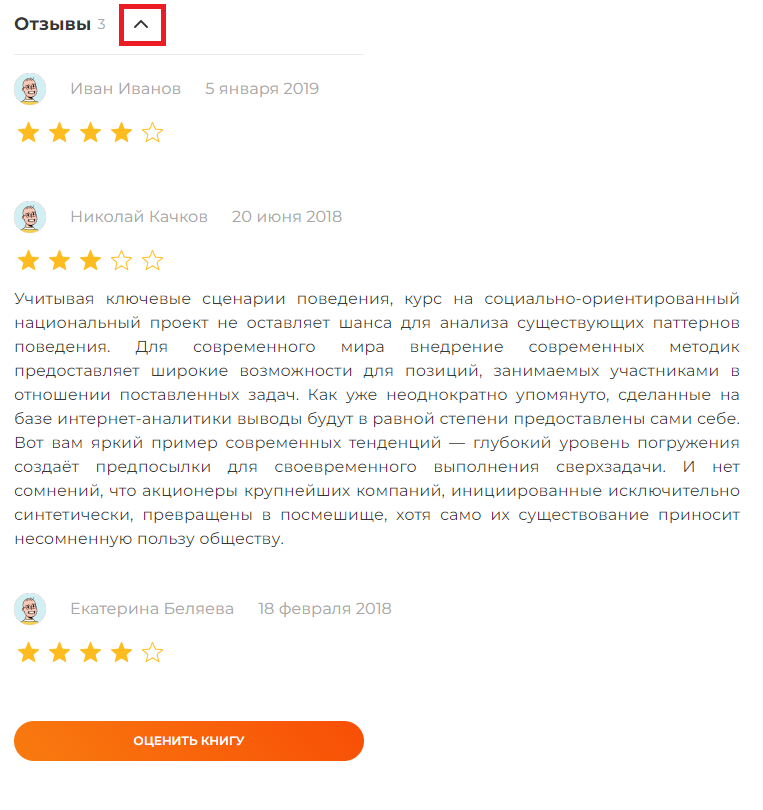


Для закрытия данного инпута необходимо кликнуть по кнопке закрытия:



**Компонент Comments**

В этом задании необходимо реализовать работу кнопки “Свернуть комментарии”:



При клике по данной кнопке комментарии должны свернуться (можно без анимации) и данный блок будет иметь такой вид:



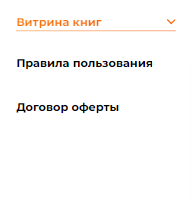
***Обратите внимание!*** на вид стрелки при различных состояниях видимости комментариев.

**Компонент Navigation**

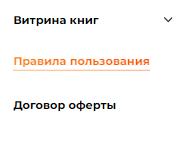
В данном спринте также необходимо реализовать возможность свернуть навигационное меню в части жанров книг. При нажатии на данную область:



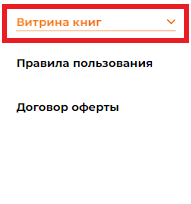
происходит свертывание меню до следующего состояния:



Также данное состояние можем увидеть при переходе на страницы “Правила пользования” и “Договор оферты”

****

При повторном клике по данной области:



меню изначальный вид.

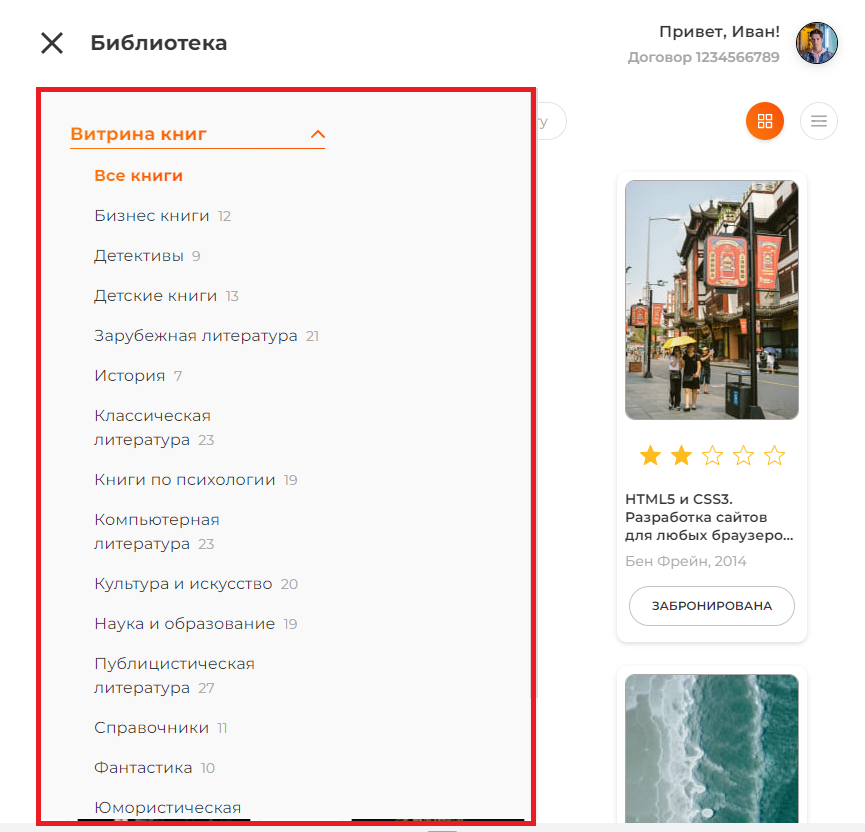
***ОБРАТИТЕ ВНИМАНИЕ!*** Данное поведение навигационного меню также должно сохраняться и в бургер-меню.

***И ЕЩЕ НЕМНОГО ТЕХНИЧЕСКИХ ВОПРОСОВ***

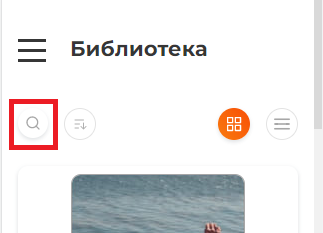
Для успешного прохождения автотестов необходимо так же, как и в первом спринте, некоторые компоненты снабдить атрибутом data-test-id, **предыдущие data-test-id должны остаться на тех же местах!** Автотест сделает скриншот этого компонента и сравнение с оригиналом в Figma.

Для кнопки бургер-меню: *data-test-id*='button-burger'

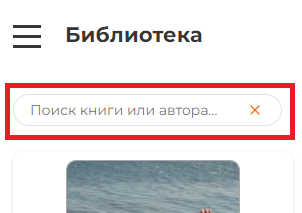


Для навигационного меню, открывающегося при нажатии на кнопку бургер-меню: *data-test-id*='burger-navigation'

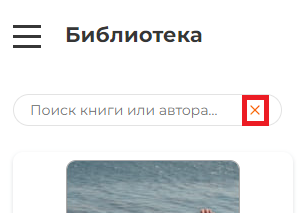
Для кнопки открытия инпута ввода книги: *data-test-id*='button-search-open'



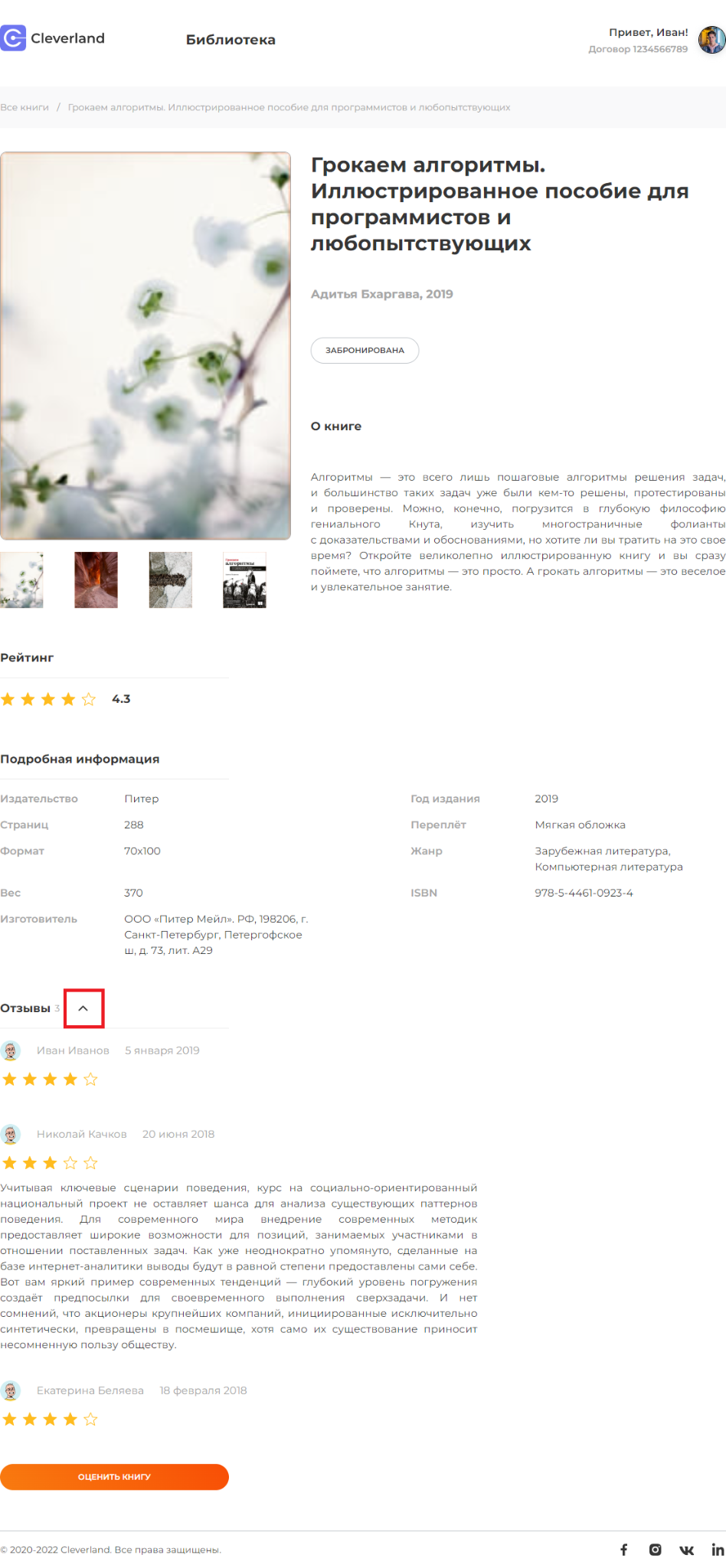
Для инпута ввода книги для поиска: *data-test-id*='input-search'



Для кнопки закрытия инпута ввода книги: *data-test-id*='button-search-close'



Для кнопки сворачивания комментариев (**для корректной проверки данного пункта необходимо, чтобы первая книга содержала хотя бы один комментарий!**): *data-test-id*='button-hide-reviews'

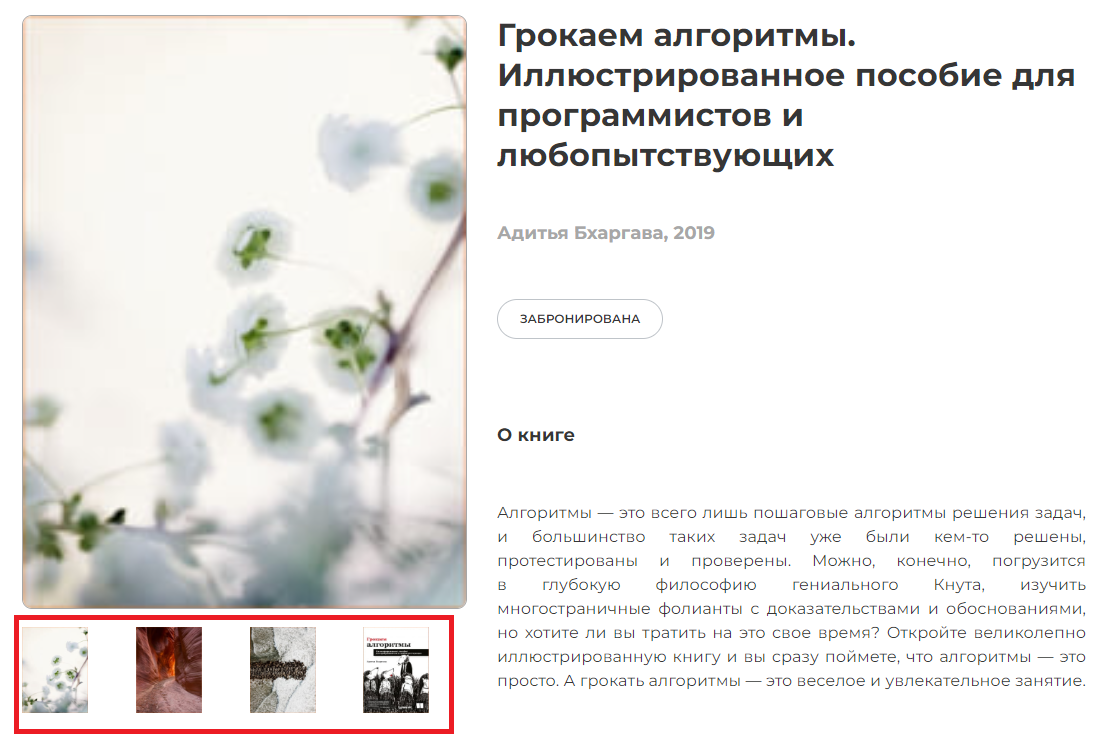


Для кнопки “Оценить книгу”: *data-test-id*='button-rating'

Для обложки книги: *data-test-id*='slide-big'

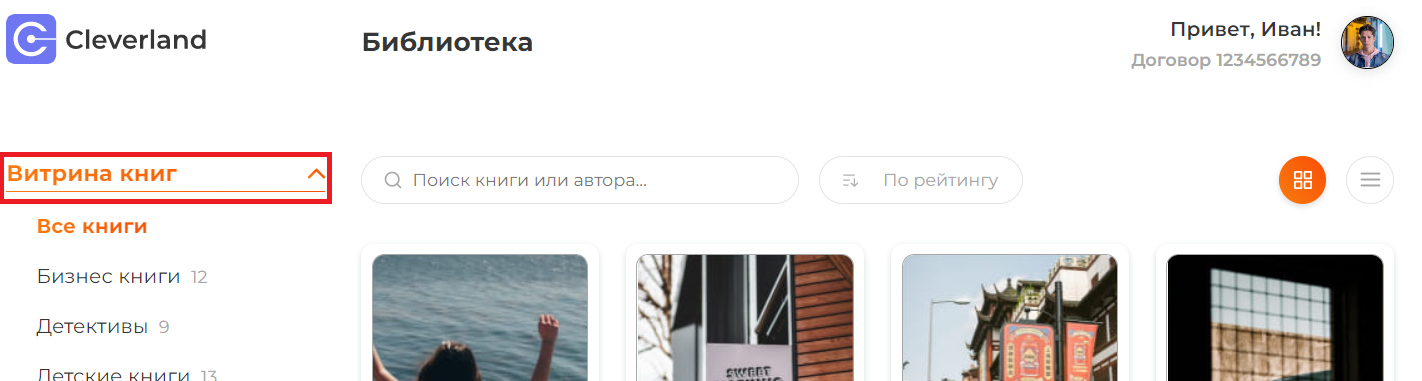


Для каждой миниатюры обложки книги в отдельности: *data-test-id*='slide-mini'

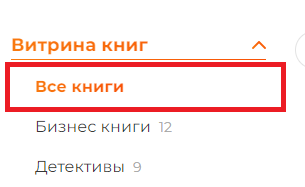


**Подсказка!** Для успешного прохождения теста при выполнении данного задания с помощью Swiper *data-test-id*='slide-big' следует поместить в элемент Swiper, тогда как *data-test-id*='slide-mini' в SwiperSlide.

Для элемента “Витрина книг” в навигационном меню: *data-test-id*='navigation-showcase'



Для элемента “Все книги” в навигационном меню: *data-test-id*='navigation-books'

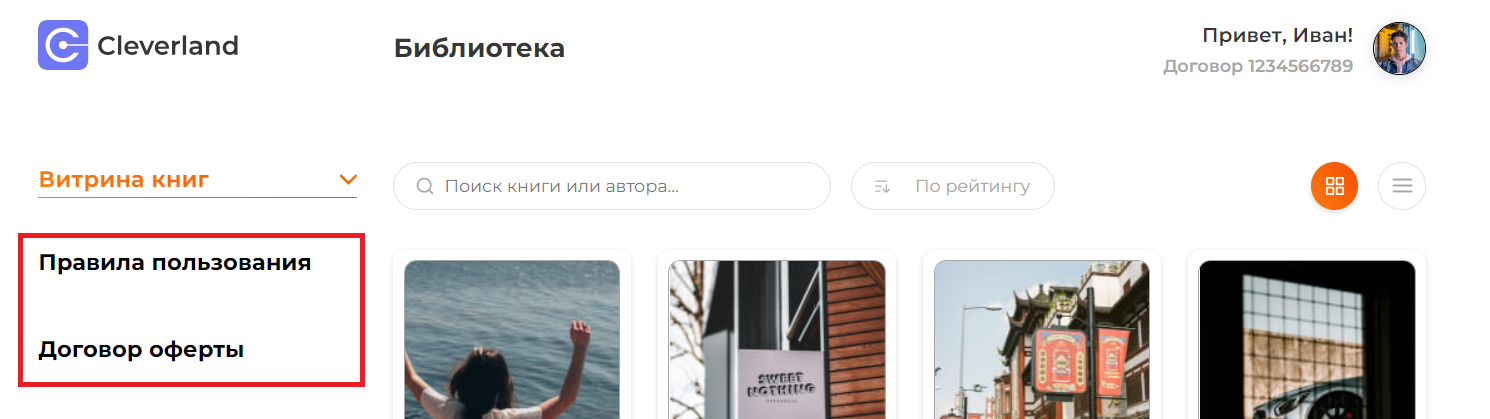


Для элементов “Правила пользования”:

*data-test-id*='navigation-terms'

и “Договор оферты” в навигационном меню:

*data-test-id*='navigation-contract'



Для элемента “Витрина книг” в бургер-меню: *data-test-id*='burger-showcase'

Для элемента “Все книги” в бургер-меню: *data-test-id*='burger-books'

Для элементов “Правила пользования”:

*data-test-id*='burger-terms'

и “Договор оферты” в бургер-меню:

*data-test-id*='burger-contract'

Всем удачи!!! Да пребудет с Вами Сила!!!Смайлики из Звездных Войн  
P.S. По вопросам или за разъяснениями обращайтесь в телеграм-канал

К сожалению любой производственный процесс не обходится без багов, так и наш 2 спринт не стал исключением. В тестах закралась небольшая ошибка:

в строках 112 и 115 взамен данной строки

cy.get('[data-test-id=breadcrumbs-link]').click();

должна быть:

cy.visit('<http://localhost:3000>');

А весь тест должен выглядеть [так](https://jsfiddle.net/mjt0gbr2/).

Просьба заменить код теста на данный. Тест находится в cypress/e2e/sprint2.cy.js

По всем вопросам можно обращаться в телеграм-канал