**web** **доступность**

Доступность — это практика, позволяющая использовать ваши сайты как можно большему числу людей. Мы традиционно думаем об этом как о доступности для людей с ограниченными возможностями, но на самом деле, в это число входят и другие группы пользователей, которые используют мобильные устройства либо имеют медленное сетевое соединение.

Вы также можете думать о доступности как о способе предоставления равных прав и одинаковых возможностей, независимо от способностей или обстоятельств. Точно так же, как не правильно лишать человека права посещать разнообразные места, только потому что он перемещается в инвалидном кресле (общественные здания в наши дни обычно имеют пандусы для инвалидных колясок или лифты), так и не правильно исключать кого-либо из веб-пространства из-за того, что у них слабое зрение или они используют мобильный телефон. Мы все разные, но все мы люди, и поэтому имеем одинаковые (человеческие) права.

Помимо того, что доступность это просто хороший тон, она также регулируется законодательством в некоторых странах и может поспособствовать открытию важных рынков, которые в противном случае не смогут использовать ваши услуги, покупать ваши продукты и т. д.

Доступность и опыт её применения принесут пользу всем:

* Семантический HTML (который улучшает доступность) также улучшает SEO, делая ваш сайт более доступным для поиска или продажи.
* Забота о доступности демонстрирует хорошую этику и мораль, что улучшает ваш имидж в обществе.
* Другие хорошие практики, улучшающие доступность, также делают сайт более удобным для использования другими группами, такими как пользователи мобильных телефонов, пользователи с низкой скоростью сети и т.д. На самом деле, каждый может извлечь выгоду из многих таких улучшений.

Большая часть содержимого интернета может быть сделана доступной просто благодаря использованию правильных HTML-элементов по назначению.

Вёрстка с помощью семантического HTML не займёт больше времени, чем с помощью не семантического (плохого) HTML, если делать это последовательно с самого начала проекта, и это также имеет другие преимущества помимо доступности:

1. Легче разрабатывать — как сказано выше, вы получаете функциональность «из коробки», плюс проще для восприятия.
2. Лучше для мобильных — семантический HTML легче по размеру, чем не семантический спагетти-код, и его легче сделать адаптивным.
3. Хорошо для SEO — поисковики уделяют больше внимания ключевым словам внутри заголовков, ссылок и т.д., чем ключевым словам, помещённым в не семантический <div> и т.д., поэтому клиентам будет проще найти ваш сайт.

### [Правильная семантика и ожидаемое пользователем поведение](https://developer.mozilla.org/ru/docs/Learn/Accessibility/CSS_and_JavaScript#%D0%BF%D1%80%D0%B0%D0%B2%D0%B8%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D0%BA%D0%B0_%D0%B8_%D0%BE%D0%B6%D0%B8%D0%B4%D0%B0%D0%B5%D0%BC%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D0%B5%D0%BC_%D0%BF%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5)

При помощи CSS мы можем заставить любой HTML-элемент выглядеть как угодно, однако это не означает, что следует так делать. Как мы часто упоминали в статье [HTML: Хорошая основа для доступности](https://developer.mozilla.org/ru/docs/Learn/Accessibility/HTML), для каждой задачи всегда стоит использовать соответствующий семантический элемент. Если не следовать данному правилу, то могут возникнуть проблемы и неудобства для всех, а в особенности для людей с ограниченными возможностями здоровья. Правильная семантика также тесно связана с ожидаемым пользователями поведением — элементы выглядят и ведут себя определенным образом, в соответствии с задачами, которые они решают. Внешний вид и поведение элементов интерфейса часто универсальны между разными сайтами и даже браузерами — пользователи привыкают к этому и им проще ориентироваться в интерфейсах.

Например, пользователь скринридера не сможет передвигаться по странице с помощью заголовков, если разработчик не разметит контент корректно с их помощью. Аналогично, если заголовок стилизован так, что вовсе не распознается как заголовок, то он теряет своё визуальное значение.

Текстовый контент

Одно из самых лучших вспомогательных средств доступности для пользователя скринридера — хорошая структура заголовков, параграфов, список и т.д.

### [Цвета и контраст](https://developer.mozilla.org/ru/docs/Learn/Accessibility/CSS_and_JavaScript#%D1%86%D0%B2%D0%B5%D1%82%D0%B0_%D0%B8_%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%B0%D1%81%D1%82)

При выборе цветовой палитры для сайта, следует следить за тем, чтобы цвет текста имел высокий уровень контрастности относительно цвета фона. Дизайн может выглядеть очень стильно, однако это не будет иметь никакого смысла, если люди с дефектами зрения (такими как дальтонизм) не смогут прочитать ваш контент.

## [JavaScript](https://developer.mozilla.org/ru/docs/Learn/Accessibility/CSS_and_JavaScript#javascript)

При неправильном использовании JavaScript тоже может вредить доступности.

Современный JavaScript — это очень мощный язык, с помощью которого можно делать кучу разных вещей: от простых обновлений пользовательского интерфейса до полноценных 2D- и 3D-игр. Понятно, что нельзя сделать абсолютно любой тип контента доступным на 100% для всех — нужно просто прилагать все усилия, чтобы делать веб-приложения доступными настолько, насколько это возможно.

[Проблема с чрезмерным использованием JavaScript](https://developer.mozilla.org/ru/docs/Learn/Accessibility/CSS_and_JavaScript#%D0%BF%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D0%B0_%D1%81_%D1%87%D1%80%D0%B5%D0%B7%D0%BC%D0%B5%D1%80%D0%BD%D1%8B%D0%BC_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC_javascript)

Данная проблема возникает, когда разработчики слишком сильно полагаются на JavaScript. Порой вы можете встретить сайт, на котором всё генерируется с помощью JavaScript — HTML, CSS и так далее. Такое решение точно так же требует поддержания всех видов доступности и решения связанных с ними проблем.

Для каждой задачи хорош свой инструмент — это правило справедливо как для выбора HTML-элементов, так и для используемой технологии в целом. Стоит хорошенько обдумать, нужно ли использовать тот сияющий, написанный на JavaScript информационный 3D-блок или же воспользоваться старым добрым и обыкновенным текстовым блоком? Нужен ли тот сложный нестандартный виджет в форме, или будет достаточно простого поля для ввода текста? Не следует генерировать весь HTML при помощи JavaScript, если это возможно.

[Ненавязчивый JavaScript](https://developer.mozilla.org/ru/docs/Learn/Accessibility/CSS_and_JavaScript#%D0%BD%D0%B5%D0%BD%D0%B0%D0%B2%D1%8F%D0%B7%D1%87%D0%B8%D0%B2%D1%8B%D0%B9_javascript)

При создании веб-страниц, стоит помнить об идее ненавязчивого JavaScript. Идея гласит, что JavaScript следует использовать только для дополнительного улучшения пользовательского опыта — базовые возможности должны работать без JavaScript. Здесь мы говорим в основном об использовании стандартных браузерных возможностей, когда это возможно.

Хорошие примеры использования подхода с ненавязчивым JavaScript:

* Валидация формы на клиенте, которая показывает пользователю сообщение о проблемах в заполнении формы без необходимости отправлять данные для проверки на сервер. Если скриптов не будет, то форма продолжит работать, просто валидация будет несколько медленнее.
* Создание нестандартных элементов управления для HTML5 <video>, которыми возможно пользоваться с клавиатуры. И в то же самое время, указание прямой ссылки на видео, которой можно воспользоваться, если JavaScript отключён (стандартные элементы управления <video> недоступны с клавиатуры в большинстве браузеров).

Для примера, мы подготовили простенькую черновую версию клиентской валидации формы — [form-validation.html](https://github.com/mdn/learning-area/blob/master/accessibility/css/form-validation.html) (здесь можно [посмотреть вживую](https://mdn.github.io/learning-area/accessibility/css/form-validation.html)). Если попытаться нажать на кнопку отправки, не заполнив одно из полей, то форма на сервер отправлена не будет, зато появится сообщение с информацией об ошибках.

Такая валидации формы соответствует принципам ненавязчивости — форму можно будет абсолютно полноценно использовать вообще без JavaScript. Форма всё равно будет провалидирована. Так как злонамеренные пользователи могут очень просто обойти клиентскую валидацию (например, отключив JavaScript в браузере), для важных форм её всегда дублируют на сервере. При этом валидация на стороне клиента всё ещё остаётся очень полезной для показа ошибок — пользователи узнают о проблемах с заполнением формы мгновенно, вместо того, чтобы ждать цикла: отправка на сервер - валидация - перезагрузка страницы. Это очень хорошее улучшение пользовательского опыта.

#### Специфичные события мыши

Большая часть интерактивности в пользовательских интерфейсах реализуется в JavaScript при помощи обработчиков событий, которые позволяют запускать код, когда происходит определённое событие. У некоторых таких событий имеются проблемы с доступностью. Хороший пример — это специфичные события мыши, например [mouseover (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/Element/mouseover_event" \o "Currently only available in English (US)), [mouseout (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/Element/mouseout_event" \o "Currently only available in English (US)), [dblclick (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/Element/dblclick_event" \o "Currently only available in English (US)) и так далее. Функции, которые активируются при срабатывании этих событий, не будут доступны при использовании других устройств ввода, например клавиатуры.

Чтобы избежать проблем подобного рода, следует дублировать функциональность при помощи независимых от устройств событий. Например, события [focus (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/Element/focus_event" \o "Currently only available in English (US)) и [blur](https://developer.mozilla.org/ru/docs/Web/API/Element/blur_event) будут работать как для мыши, так и для клавиатуры.

Событие [click](https://developer.mozilla.org/ru/docs/Web/API/Element/click_event) довольно любопытное — выглядит так, будто бы оно специфичное для мыши, однако большинство браузеров при нажатии клавиши Enter/Return активируют обработчики событий [onclick](https://developer.mozilla.org/ru/docs/Web/API/Element/click_event) на находящихся в фокусе ссылках или элементах формы, а также при тапе по элементам на сенсорном экране. Впрочем, это не работает автоматически, если вы позволяете принимать фокус неинтерактивному по умолчанию элементу при помощи tabindex — в таких случаях нужно специально прослушивать нажатие нужных кнопок