ДИСЦИПЛИНА	Фронтенд и бэкенд разработка
ИНСТИТУТ	ИПТИП
КАФЕДРА	Индустриального программирования
ВИД УЧЕБНОГО МАТЕРИАЛА	Методические указания к практическим занятиям
ПРЕПОДАВАТЕЛЬ	Астафьев Рустам Уралович
СЕМЕСТР	1 семестр, 2025/2026 уч. год

Ссылка на материал:
https://github.com/astafiev-rustam/frontend-and-backend-development/tree/practice-1-17

Практическое занятие 17: A11y-аудит и обеспечение доступности

В рамках данного занятия будут использоваться основные подходы к обеспечению доступности и его аудиту.

Для восполнения знаний по данной теме рекомендуется повторить материалы лекции. Дополнительно можно ознакомиться с материалом по ссылке:

https://web-standards.ru/articles/a11y-audit-basics/

Зачем нужна доступность (A11y) и что мы будем делать?

Доступность (Accessibility, A11y) — это практика разработки веб-интерфейсов так, чтобы ими могли пользоваться люди с различными ограничениями: слабовидящие, незрячие, люди с двигательными нарушениями, когнитивными расстройствами и другими особенностями. Мы не всегда видим этих пользователей, но они существуют — и для многих из них доступность является не опцией, а единственным способом взаимодействия с цифровыми продуктами.

На этом занятии:

- Узнаем, как проводить A11y-аудит — находить типичные проблемы доступности.
- Научимся исправлять их с помощью:
- Семантической HTML-разметки
- ARIA-атрибутов
- Управления с клавиатуры
- Корректной работы со скринридерами

Каждый пример ниже построен по схеме:

- Исходный код — пример с типичными ошибками доступности.

- Проблемы — что именно мешает пользователям.
- Исправленный код — готовое доступное решение.
- Что изменилось и почему — подробное описание правок и их значения.

Примеры

Рассмотрим несколько примеров на обеспечение доступности в веб-приложениях.

Пример 1: Доступная навигация

Рассмотрим пример создания доступного навигационного меню.

Исходный файл:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Сайт</title>
    <style>
        .nav { display: flex; gap: 20px; }
        .nav-item {
            padding: 10px;
            cursor: pointer;
            background: #f0f0f0;
        }
    </style>
</head>
<body>
    <div class="nav">
        <div class="nav-item" onclick="showPage('home')">Главная</div>
        <div class="nav-item" onclick="showPage('about')">О нас</div>
        <div class="nav-item" onclick="showPage('contact')">Контакты</div>
    </div>
</body>
</html>
```

Здесь, как мы видим, есть проблемы:

- Используются `div` вместо семантических элементов
- Нет возможности навигации с клавиатуры (Tab не работает)
- Скринридер не понимает, что это меню навигации
- Нет визуального индикатора фокуса

Вот так это можно исправить:

1. Контейнер навигации

```
    <!-- БЫЛО -->
    <div class="nav">
```

```
  <!-- Простой div без семантики -->
</div>

<!-- СТАЛО -->
<nav aria-label="Основная навигация">
  <!--
    <nav> - семантический тег для навигации
    aria-label - описывает назначение для скринридеров
  -->
</nav>
```

2. Элементы меню

```
<!-- БЫЛО -->
<div class="nav-item" onclick="showPage('home')">Главная</div>
<!--
  Проблемы:
  - div не семантичен для навигации
  - onclick недоступен с клавиатуры
  - нет состояния текущей страницы
-->

<!-- СТАЛО -->
<ul>
  <li>
    <a href="#home" class="nav-link" aria-current="page">Главная</a>
    <!--
      <a> - семантическая ссылка, доступна с клавиатуры
      href - работает без JavaScript
      aria-current="page" - указывает текущую страницу
    -->
  </li>
</ul>
```

3. Стили фокуса

```
/* БЫЛО */
.nav-item {
  padding: 10px;
  cursor: pointer;
  background: #f0f0f0;
  /* Нет индикатора фокуса для клавиатуры */
}

/* СТАЛО */
.nav-link:focus {
  outline: none;
  border-color: #0066cc;
  background: #e6f3ff;
```

```
        /* Четкий визуальный индикатор фокуса */
    }
```

Полный код исправления:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Сайт</title>
    <style>
        nav ul {
            display: flex;
            gap: 20px;
            list-style: none;
            padding: 0;
        }
        .nav-link {
            padding: 10px;
            background: #f0f0f0;
            text-decoration: none;
            color: #333;
            border: 2px solid transparent;
            display: block;
        }
        /* Критически важные стили для доступности */
        .nav-link:focus {
            outline: none;
            border-color: #0066cc;
            background: #e6f3ff;
        }
        .nav-link:hover {
            background: #ddefff;
        }
        /* Индикатор текущей страницы */
        .nav-link[aria-current="page"] {
            background: #0066cc;
            color: white;
        }
    </style>
</head>
<body>
    <!--
        aria-label описывает назначение навигации для скринридеров
        Скринридер зачитает: "Основная навигация, список из 3 пунктов"
    -->
    <nav aria-label="Основная навигация">
        <ul>
            <li>
                <!--
                    aria-current="page" указывает скринридеру на текущую страницу
                    При фокусе скринридер зачитает: "Главная, текущая страница,
```

```
ссылка"
                -->
                <a href="#home"
                   class="nav-link"
                   aria-current="page">
                   Главная
                </a>
            </li>
            <li>
                <a href="#about" class="nav-link">О нас</a>
            </li>
            <li>
                <a href="#contact" class="nav-link">Контакты</a>
            </li>
        </ul>
    </nav>
</body>
</html>
```

Таким образом, мы получили меню, которое:

- Работает с клавиатуры (Tab/Shift+Tab)
- Имеет четкий визуальный фокус
- Правильно озвучивается скринридерами
- Семантически корректно размечено

Пример 2: Доступная кнопка с иконкой

Рассмотрим пример создания доступной кнопки с иконкой.

Исходный файл:

```
<!DOCTYPE html>
<html>
<head>
    <style>
        .close-btn {
            background: #ff4444;
            color: white;
            border: none;
            padding: 10px;
            border-radius: 50%;
            cursor: pointer;
            font-size: 20px;
        }
    </style>
</head>
<body>
    <!-- Проблема: скринридер зачитает только "×" без контекста -->
    <button class="close-btn">×</button>
```

```
</body>
</html>
```

Здесь, как мы видим, есть проблемы:

- Скринридер зачитает только символ "×" без контекста
- Непонятно назначение кнопки для незрячих пользователей
- Нет текстовой альтернативы для иконки

Вот так это можно исправить:

1. Структура кнопки

```
<!-- БЫЛО -->
<button class="close-btn">x</button>
<!--
  Проблемы:
  - скринридер зачитает только "x"
  - нет контекста назначения кнопки
-->

<!-- СТАЛО -->
<button class="close-btn" aria-label="Закрыть диалоговое окно">
  <span class="icon" aria-hidden="true">x</span>
  <span class="sr-only">Закрыть диалоговое окно</span>
</button>
<!--
  aria-label - текстовое описание для скринридера
  aria-hidden="true" - скрывает иконку от скринридера
  .sr-only - скрытый текст для дополнительного описания
-->
```

2. Стили доступности

```
/* БЫЛО */
.close-btn {
  background: #fff4444;
  /* Нет индикатора фокуса */
}

/* СТАЛО */
.close-btn:focus {
  outline: none;
  border-color: #0066cc;
  box-shadow: 0 0 0 3px rgba(0, 102, 204, 0.3);
  /* Яркий индикатор фокуса для клавиатурной навигации */
}

/* Класс для скрытия текста визуально */
```

```
    .sr-only {
      position: absolute;
      width: 1px;
      height: 1px;
      padding: 0;
      margin: -1px;
      overflow: hidden;
      clip: rect(0, 0, 0, 0);
      white-space: nowrap;
      border: 0;
    }
```

Полный код исправления:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <style>
        .close-btn {
            background: #ff4444;
            color: white;
            border: none;
            padding: 10px 15px;
            border-radius: 4px;
            cursor: pointer;
            font-size: 16px;
            border: 2px solid transparent;
        }
        /* Обязательные стили для фокуса */
        .close-btn:focus {
            outline: none;
            border-color: #0066cc;
            box-shadow: 0 0 0 3px rgba(0, 102, 204, 0.3);
        }
        .icon {
            font-size: 20px;
            margin-right: 8px;
        }
        /* Скрываем текст визуально, но оставляем для скринридеров */
        .sr-only {
            position: absolute;
            width: 1px;
            height: 1px;
            padding: 0;
            margin: -1px;
            overflow: hidden;
            clip: rect(0, 0, 0, 0);
            white-space: nowrap;
            border: 0;
        }
    </style>
</head>
```

```
<body>
    <!--
        aria-label дает текстовое описание для скринридера
        Скринридер зачитает: "Закрыть диалоговое окно, кнопка"
    -->
    <button class="close-btn" aria-label="Закрыть диалоговое окно">
        <span class="icon" aria-hidden="true">×</span>
        <!--
            aria-hidden="true" скрывает иконку от скринридера
            чтобы не зачитывался символ "×"
        -->
        <span class="sr-only">Закрыть диалоговое окно</span>
        <!--
            Альтернативный вариант: видимый текст
            <span>Закрыть</span>
        -->
    </button>

    <!-- Дополнительный пример: кнопка с состоянием -->
    <button class="close-btn"
            aria-label="Добавить в избранное"
            aria-pressed="false">
        <span class="icon" aria-hidden="true">♥</span>
        Избранное
    </button>
</body>
</html>
```

Таким образом, мы получили кнопку, которая:

- Имеет понятное текстовое описание для скринридеров
- Корректно работает с клавиатурой
- Имеет четкий индикатор фокуса
- Может передавать свое состояние (aria-pressed)

Пример 3: Доступное модальное окно

Рассмотрим пример создания доступного модального окна.

Исходный файл:

```
<!DOCTYPE html>
<html>
<head>
    <style>
        .modal {
            display: none;
            position: fixed;
            top: 50%; left: 50%;
            transform: translate(-50%, -50%);
```

```
            background: white;
            padding: 20px;
            border: 1px solid #ccc;
            z-index: 1000;
        }
        .overlay {
            display: none;
            position: fixed;
            top: 0; left: 0;
            width: 100%; height: 100%;
            background: rgba(0,0,0,0.5);
            z-index: 999;
        }
    </style>
</head>
<body>
    <button onclick="openModal()">Открыть окно</button>

    <div class="overlay" onclick="closeModal()"></div>
    <div class="modal" id="modal">
        <h2>Важное сообщение</h2>
        <p>Это тестовое модальное окно</p>
        <button onclick="closeModal()">×</button>
    </div>

    <script>
        function openModal() {
            document.getElementById('modal').style.display = 'block';
            document.querySelector('.overlay').style.display = 'block';
        }
        function closeModal() {
            document.getElementById('modal').style.display = 'none';
            document.querySelector('.overlay').style.display = 'none';
        }
    </script>
</body>
</html>
```

Здесь, как мы вишем, есть проблемы:

- При открытии модалки фокус остается на кнопке открытия
- Нельзя закрыть модалку клавишей Escape
- Скринридер продолжает читать контент под модалкой
- Нет правильной семантики для модального окна

Вот так это можно исправить:

1. Разметка модального окна

```
    <!-- БЫЛО -->
    <div class="modal" id="modal">
```

```
  <h2>Важное сообщение</h2>
  <button onclick="closeModal()"></button>
</div>
<!--
  Проблемы:
  - нет семантической роли
  - скринридер не понимает, что это модальное окно
-->

<!-- СТАЛО -->
<div class="modal"
     id="modal"
     role="dialog"
     aria-modal="true"
     aria-labelledby="modal-title"
     tabindex="-1">
  <h2 id="modal-title">Важное сообщение</h2>
  <button aria-label="Закрыть диалоговое окно" autofocus></button>
</div>
<!--
  role="dialog" - определяет диалоговое окно
  aria-modal="true" - указывает на модальный режим
  aria-labelledby - связывает с заголовком
  tabindex="-1" - позволяет программный фокус
  autofocus - автоматический фокус при открытии
-->
```

2. Управление фокусом

```
// БЫЛО
function openModal() {
  document.getElementById('modal').style.display = 'block';
  // Фокус остается на предыдущем элементе
}

// СТАЛО
function openModal() {
  // Запоминаем активный элемент для возврата фокуса
  previousActiveElement = document.activeElement;

  // Фокусируемся на модалке
  modal.focus();

  // Скрываем основной контент от скринридера
  document.querySelectorAll('body > *:not(.modal):not(.overlay)')
    .forEach(el => el.setAttribute('aria-hidden', 'true'));
}

function closeModal() {
  // Возвращаем фокус на предыдущий элемент
  if (previousActiveElement) {
    previousActiveElement.focus();
```

```
    }
  }
```

3. Захват фокуса внутри модалки

```
  // НОВЫЙ КОД - ловим фокус внутри модалки
  modal.addEventListener('keydown', function(event) {
    if (event.key === 'Tab') {
      const focusableElements = modal.querySelectorAll(
        'button, [href], input, select, textarea,
[tabindex]:not([tabindex="-1"])'
      );
      const firstElement = focusableElements[0];
      const lastElement = focusableElements[focusableElements.length - 1];

      // Зацикливаем фокус внутри модалки
      if (event.shiftKey && document.activeElement === firstElement) {
        event.preventDefault();
        lastElement.focus();
      } else if (!event.shiftKey && document.activeElement === lastElement) {
        event.preventDefault();
        firstElement.focus();
      }
    }
  });
```

4. Закрытие по Escape

```
  // НОВЫЙ КОД - обработка клавиши Escape
  function handleEscape(event) {
    if (event.key === 'Escape') {
      closeModal();
    }
  }

  // Добавляем при открытии модалки
  document.addEventListener('keydown', handleEscape);
```

Полный код исправления:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <style>
        .modal {
            display: none;
            position: fixed;
```

```
            top: 50%; left: 50%;
            transform: translate(-50%, -50%);
            background: white;
            padding: 30px;
            border-radius: 8px;
            box-shadow: 0 10px 30px rgba(0,0,0,0.3);
            z-index: 1000;
            max-width: 500px;
            width: 90%;
        }
        .overlay {
            display: none;
            position: fixed;
            top: 0; left: 0;
            width: 100%; height: 100%;
            background: rgba(0,0,0,0.5);
            z-index: 999;
        }
        .modal:focus {
            outline: 3px solid #0066cc;
            outline-offset: 2px;
        }
        .close-btn {
            position: absolute;
            top: 15px;
            right: 15px;
            background: none;
            border: none;
            font-size: 24px;
            cursor: pointer;
            padding: 5px;
            border-radius: 4px;
        }
        .close-btn:focus {
            outline: 2px solid #0066cc;
        }
        /* Скрыть контент за модалкой от скринридера */
        [aria-hidden="true"] {
            display: none;
        }
    </style>
</head>
<body>
    <button onclick="openModal()">Открыть модальное окно</button>

    <div class="overlay" onclick="closeModal()"></div>

    <!--
        role="dialog" указывает, что это диалоговое окно
        aria-modal="true" сообщает скринридеру о модальном режиме
        aria-labelledby связывает заголовок с окном
        tabindex="-1" позволяет программно фокусироваться
    -->
    <div class="modal"
```

```
         id="modal"
         role="dialog"
         aria-modal="true"
         aria-labelledby="modal-title"
         tabindex="-1">

        <h2 id="modal-title">Важное сообщение</h2>
        <p>Это тестовое модальное окно с полной доступностью.</p>

        <!--
            aria-label дает описание для кнопки закрытия
            Автофокус на кнопке закрытия при открытии
        -->
        <button class="close-btn"
                onclick="closeModal()"
                aria-label="Закрыть диалоговое окно"
                autofocus>
            ×
        </button>
    </div>

    <script>
        let previousActiveElement;
        let modal = document.getElementById('modal');

        function openModal() {
            // Запоминаем активный элемент для возврата фокуса
            previousActiveElement = document.activeElement;

            // Показываем модалку
            modal.style.display = 'block';
            document.querySelector('.overlay').style.display = 'block';

            // Скрываем основной контент от скринридера
            document.querySelectorAll('body > *:not(.modal):not(.overlay)')
                .forEach(el => el.setAttribute('aria-hidden', 'true'));

            // Фокусируемся на модалке
            modal.focus();

            // Добавляем обработчик Escape
            document.addEventListener('keydown', handleEscape);
        }

        function closeModal() {
            // Скрываем модалку
            modal.style.display = 'none';
            document.querySelector('.overlay').style.display = 'none';

            // Возвращаем видимость основному контенту
            document.querySelectorAll('[aria-hidden="true"]')
                .forEach(el => el.removeAttribute('aria-hidden'));

            // Возвращаем фокус на предыдущий элемент
```

```
            if (previousActiveElement) {
                previousActiveElement.focus();
            }

            // Убираем обработчик Escape
            document.removeEventListener('keydown', handleEscape);
        }

        function handleEscape(event) {
            if (event.key === 'Escape') {
                closeModal();
            }
        }

        // Ловим фокус внутри модалки, чтобы он не уходил наружу
        modal.addEventListener('keydown', function(event) {
            if (event.key === 'Tab') {
                const focusableElements = modal.querySelectorAll(
                    'button, [href], input, select, textarea,
[tabindex]:not([tabindex="-1"])'
                );
                const firstElement = focusableElements[0];
                const lastElement = focusableElements[focusableElements.length -
1];

                if (event.shiftKey && document.activeElement === firstElement) {
                    event.preventDefault();
                    lastElement.focus();
                } else if (!event.shiftKey && document.activeElement ===
lastElement) {
                    event.preventDefault();
                    firstElement.focus();
                }
            }
        });
    </script>
</body>
</html>
```

Таким образом, мы получили модальное окно, которое:

- Захватывает фокус и не позволяет ему уйти за пределы окна
- Закрывается по клавише Escape
- Правильно скрывает фоновый контент от ск ринридера
- Возвращает фокус на предыдущий элемент при закрытии
- Имеет правильную семантику для вспомогательных технологий

После представленных примеров можем перейти к самостоятельной работе по практическому занятию.

Самостоятельная работа

В рамках самостоятельной работы необходимо обеспечить доступность и провести её ручное тестирование для страницы Контактов (contacts.html) из проекта.

Общие требования:

- **Валидная HTML-разметка** (проверить через validator.w3.org)
- **Работоспособность с клавиатуры** (полная навигация Tab/Shift+Tab)

1. Семантическая разметка

- [ ] Использованы правильные HTML5 теги (`<form>`, `<fieldset>`, `<legend>`)
- [ ] Все интерактивные элементы семантически корректны
- [ ] Правильная иерархия заголовков (`<h1>`-`<h6>`)
- [ ] Логическая структура документа

2. Доступность форм

- [ ] Все поля имеют связанные `<label>` с `for`/`id`
- [ ] Обязательные поля помечены `aria-required="true"` и `required`
- [ ] Поля с ошибками имеют `aria-invalid="true"` и `aria-describedby`
- [ ] Группы полей объединены в `<fieldset>` с `<legend>`
- [ ] Есть визуальные и текстовые индикаторы обязательных полей

3. Навигация с клавиатуры

- [ ] Все интерактивные элементы доступны с Tab
- [ ] Логический порядок фокуса
- [ ] Видимый и четкий индикатор фокуса
- [ ] Клавиша Enter/Space работают на всех элементах
- [ ] Escape закрывает модальные окна (если есть)

4. ARIA-атрибуты

- [ ] Правильное использование `aria-label`, `aria-labelledby`, `aria-describedby`
- [ ] Состояния элементов (`aria-expanded`, `aria-pressed`, `aria-current`)
- [ ] Живые регионы (`aria-live`) для динамического контента
- [ ] Логические роли (`role="button"`, `role="navigation"`)

5. Цвет и контраст

- [ ] Соотношение контрастности минимум 4.5:1 для текста
- [ ] Цвет не используется как единственный способ передачи информации
- [ ] Состояния элементов различимы без цвета