DOM and Accessibility (a11y)

- `tabindex` and `disabled` control whether an element can be accessed
- `focused` elements should visually indicate the selected stat
- Screen readers generate audio (text) descriptions from the markup of the web page
  - Relies on proper HTML semantics
  - Need to have `alt` or `label` text on images and visuals
- `details` and `dialog` are interactive html elements
  - `main`, `section`, `article`, `nav` are pseudo-elements
  - `time`, `progress`, `meter`, and `cite` are all also notable elements
- Normally, text inside an element is the text read by a screen reader / search engine; images need more help
  - ARIA (accessible rich internet applications) define visual behavior (such as the `alt` attribute on an `img` or `role` and `label` on icons)
    - Includes the state of the element; eg: `aria_expanded`
- Example ARIA code:

```
$('.open-menu').on('click', show_menu);
function show_menu(){
    $('.menu').show().attr('aria-expanded', true)
}
```

- `label` element describes what the form element applies to
  - Rails uses `label_tag` or `check_box_tag`
    - Use `for` attribute with the id of the element it connects to
  - Should be clickable to interact with the element

Events

- Occurrences that affect the user interface
- Usually (not always) associated with a DOM element
- Bind a handler (function) for a particular event type on one or more elements
  - Generally done in a setup function passed to `document.ready()` or `$()`

AJAX (Asynchronous JavaScript and XML)

- DOM API called `XmlHttpRequest` (`xhr`) contacts server asynchronously (in background) and without redrawing page
  - Normal HTTP request w/ special header: `X-Requested-With: XmlHttpRequest`
- Controller action receives request via route
  - Controller decides what should be rendered in responses:
    - `render layout: false` renders an empty layout
    - `render partial: 'movies/show'` for a partial (common with browser requests)
    - `render json: @movies` (calls `to_json`) (common with APIs)
    - `render xml: @movie.title` (calls `to_xml`)
    - `render text: @movie.title`
    - `render nothing: true` renders nothing

Basic Lifecycle

```
r = new XmlHttpRequest;
r.open(<method>, <url>, <async>);
r.send(<request content>)
r.abort();
```

- `method` $\in$ {GET, POST, HEAD, PUT, DELETE, ...}

- Usually, `async` is set to `true` because we don't want it to block
- Callbacks during XHR processing:
  - `r.onReadyStateChange=function(XmlHttpRequest r){...}`
    - Function that inspects `r.readyState` which takes on $v \in$ {`uninitialized`, `open`, `sent`, `receiving`, `loaded`}
  - `r.status` contains HTTP status of response
  - `r.responseText` contains response content string
- Can also be done with jQuery

```
$.ajax({
    type: 'GET',
    url: <url>,
    timeout: <miliseconds>,
    success: <function>
    error: <function>,
    // Many other options possible.
});
```

- JavaScript `fetch`:

```
fetch(<url>, <options>).then((response) => {
    if (response.ok) {
        return response.json();
    }
    throw new Error('Something went wrong!');
}).then((responseJson) => {
    // Do something with the response.
}).catch((err) => {
    console.error(err);
});
```

Rails with AJAX

- `javascript_include_tag 'application'` used to use js
- Code goes in `app/assets/javascript/*.js`
- Define handler function that:
  - Optionally inspects element state, attributes, etc.
  - calls `$.ajax()`
- Define controller action and route to receive requests and determine what will be returned
- Define callback function to receive server reply
  - Unpack JSON objects and modify DOM
  - Replace existing HTML element in place
  - Add, change, or remove CSS classes or properties
- eg:

```
class MoviesController < ApplicationController
    def show
        @movie = Movie.find(params[:id])
        render partial: 'movie', movie: @movie if
            request.xhr?
        # same as render @movie.to_json if request.xhr?
        else render default (show.html.erb)
    end
end
```