Async Clipboard API

One thing I've always figured out how to use the clipboard on web applications so we can copy and paste into the app.

Until now, we can use document.execCommand to interact with the clipboard; but it's not supported in all browsers and has been deprecated.

in Unblocking clipboard access Jason Miller and Thomas Steiner present a new API for managing clipboard copying and pasting.

This is a promise-based async API that will allow you to copy and paste text.

For this example, we'll assume we have the following HTML code

```
<div>
  <input type="text" id="to-copy">
  <button id="write-btn">Copy to clipboard</button>
</div>

<div>
  <h3 id="clipboard-results"></h3>
  <button id="read-btn">Paste from clipboard</button>
</div>
```

The first block of Javascript will capture references to the items we want to use.

```
const readBtn = document.getElementById('read-btn');
const writeBtn = document.getElementById('write-btn');

const resultsEl = document.getElementById('clipboard-results');
const inputEl = document.getElementById('to-copy');
```

To copy the text of the textarea element into the clipboard we need to attach the action to an event listener or some other type of user interaction.

The function does the following:

1. Captures the white space trimmed value of the input field
2. Writes the text to the clipboard
3. Changes the text of the button to say **Copied!**
4. If there's an error jump to the catch statement and log the eror to the console.

```
writeBtn.addEventListener('click', () => {
  const inputValue = inputEl.value.trim(); // 1
  if (inputValue) {
    navigator.clipboard.writeText(inputValue) // 2
      .then(() => {
        inputEl.value = '';
        if (writeBtn.innerText !== 'Copied!') {
          const originalText = writeBtn.innerText;
          writeBtn.innerText = 'Copied!'; ''// 3       }
      })
      .catch(err => {
        console.log('Something went wrong', err); //'Something went wrong
      })
  }
});
```

Pasting content from the clipboard is bot easier and more complicated.

It is more complicated because it requires user permission to work and will only paste the text if the user authorizes it.

The process is as follows:

1. Grab the text from the clipboard using `navigator.clipboard.readText()`
2. Insert the content into the desired element
3. If there is an error jump to the catch statement and execute the code inside

```
readBtn.addEventListener('click', () => {
  navigator.clipboard.readText() // 1
    .then((text) => {
      resultsEl.innerText = text; // 2
```

```
    })
    .catch((err) => {
      console.error('Something went wrong', err); 'Something went wrong'//
    })
});
```

Feature detection

To make sure we can copy and paste text in as many browsers as possible we need to se feature detection by testing if the `navigator.clipboard` object exists or not.

If it doesn't we log it to console and use `execCommand` or another library.

If it exists then we execute the modern API commands. The code looks like the example below:

```
if (!navigator.clipboard) {
  console.error(`Not supported 😢. Use execCommand or leave the feature of
  // Use execCommand or another alternative
} else {
  console.log('it works');
  'it works'// The code for the previous examples goes here
}
```

What I left out, and limitations of the API

The Async Clipboard API is deliberately simple. The examples we cover in this post copy and paste text, not any other media type.

Thoma Steiner' Multi-MIME Type Copying with the Async Clipboard API provides examples and rationale for working with copy and pasting images and working with these images in different applications:

> If you copy an SVG image, then open macOS Preview, and finally click `"File" > "New from Clipboard"`, you would probably expect an image to be pasted. However, if you copy an SVG image and paste it into Visual Studio Code or into SVGOMG's `"Paste markup"` field, you would probably expect the source code to be pasted.

Raw Clipboard API

There is separate Raw Clipboard API to handle other clipboard activities. This is significantly harder to achieve and both Firefox and Safari have expressed unwillingness to implement the API on privacy grounds.

The Raw Clipboard API design document explains what the API is and how it's expected to work.

The full code

The following listing includes both the feature detection and the code to copy and paste text, and that's all it will copy.

```
const readBtn = document.getElementById('read-btn');
const writeBtn = document.getElementById('write-btn');

const resultsEl = document.getElementById('clipboard-results');
const inputEl = document.getElementById('to-copy');

if (!navigator.clipboard) {
  console.error('Async clipboard API not supported');
} else {
  readBtn.addEventListener('click', () => {
    navigator.clipboard.readText()
      .then((text) => {
        resultsEl.innerText = text;
      })
      .catch((err) => {
        console.error('Something went wrong', err);
      })
```

```
  });

  writeBtn.addEventListener('click', () => {
    const inputValue = inputEl.value.trim();
    if (inputValue) {
      navigator.clipboard.writeText(inputValue)
        .then(() => {
          inputEl.value = '';
          if (writeBtn.innerText !== 'Copied!') {
            const originalText = writeBtn.innerText;
            writeBtn.innerText = 'Copied!';
          }
        })
        .catch(err => {
          console.log('Something went wrong', err);
        })
    }
  });
}
```