Adding More Functions to `user-workflow.js`

In this lesson, you will add more functions to `user-workflow.js` for the visualization of the workflow.

`index.html`

Next, you can start composing the visual part of the workflow. First, let's create a simple web page that will let you show different steps of the workflow. The web page also needs to load the client application JavaScript from an external file. A new file is created called `index.html` in the `web-site` directory, with content similar to the following:

```
<html>
  <body>
    <div step="initial">
      <h1>Select a file</h1>
      <input type="hidden" id="apiurl" value="${API_URL}" />
      <input type="file" id="picker"/>
    </div>
    <div step="uploading" style="display: none">
      Please wait, uploading...
      <br/>
      <progress id="progressbar" max="100" value="0" />
    </div>
    <div step="converting" style="display: none">
      Please wait, converting your file...
    </div>
    <div step="result" style="display: none">
      <h1>Your thumbnail is ready</h1>
      <a id="resultlink">download it</a>
    </div>
    <div step="error" style="display: none">
      <h1>There was an error creating the thumbnail</h1>
      <p id="errortext"></p>
    </div>
    <script src="user-workflow.js"></script>
  </body>
</html>
```

code/ch11/web-site/index.html

Notice the placeholder for the API URL on line 5 of the previous listing. In earlier chapters, you just used a relative URL for the upload form. You could do that because both the form and the processing endpoint were in the same API, so they were under the same web domain. After you have introduced the changes in this chapter, client devices will download the web assets (including the `index.html` file just created) from S3, but they will need to upload the files to your API Gateway, which will be on a different domain. You need to somehow let the client device know about the API Gateway URL. In theory, you could use the actual value of your API endpoint URL directly in the file, but that would make it difficult to deploy different versions for development, testing, and production from the same source. Instead, you'll use a placeholder value here and replace it with the actual URL during deployment.

To change the web page according to the workflow steps, you'll need a utility function that can show a named section of the page and hide all the other sections. The sections of your `index.html` page have a custom HTML attribute called `step` that you can use for this purpose. The following function is added to `user-workflow.js`.

```
function showStep(label) {
  const sections = Array.from(document.querySelectorAll('[step]'));
  sections.forEach(section => {
    if (section.getAttribute('step') === label) {
      section.style.display = '';
    } else {
      section.style.display = 'none';
    }
  });
};
```

Line 91 to Line 100 of code/ch11/web-site/user-workflow.js

You will also want to display a progress bar during uploads. The standard `XMLHttpRequest` progress event contains two fields, `total` and `loaded`, containing the number of bytes that are expected to transfer and the number of bytes transferred so far. You'll populate an HTML5 progress element, which has two corresponding attributes, `max` and `value`. The following function is added to `user-workflow.js`:

```
function progressNotifier(progressEvent) {
  const progressElement = document.getElementById('progressbar');
  const total = progressEvent.total;
  const current = progressEvent.loaded;
  if (current && total) {
    progressElement.setAttribute('max', total);
    progressElement.setAttribute('value', current);
  }
};
```

Line 101 to Line 109 of ch11/web-site/user-workflow.js

You now have all the pieces to compose a thumbnail conversion workflow. You can wait for users to select a file, then show the progress message and upload the `Blob`, wait for the conversion results, and then show the download link. In case of any trouble, you can show the error message. The following function is added to `user-workflow.js`:

```
async function startUpload(evt) {
  const picker = evt.target;
  const file = picker.files && picker.files[0];
  const apiUrl = document.getElementById('apiurl').value;

  if (file && file.name) {
    picker.value = '';
    try {
      showStep('uploading');
      const signatures = await getSignatures(apiUrl);
      console.log('got signatures', signatures);
      await uploadBlob(signatures.upload, file, progressNotifier);
      showStep('converting');
      await pollForResult(signatures.download, 3000, 20);
      const downloadLink = document.getElementById('resultlink');
      downloadLink.setAttribute('href', signatures.download);
      showStep('result');
    } catch (e) {
      console.error(e);
      const displayError = e.message || JSON.stringify(e);
      document.getElementById('errortext').innerHTML = displayError;
      showStep('error');
    }
  }
};
```

Line 110 to Line 134 of code/ch11/web-site/user-workflow.js

Lastly, something has to start the workflow method when a user selects a file. Let's add a bit of code to the web page initialisation that calls the `startUpload` function when the file picker state changes. The following function is added to `user-workflow.js`:

```
function initPage() {
```

```
function initPage() {
  const picker = document.getElementById('picker');
  showStep('initial');
  picker.addEventListener('change', startUpload);
};
window.addEventListener('DOMContentLoaded', initPage);
```

Line 135 to Line 140 of code/ch11/web-site/user-workflow.js

In the next lesson, you will learn how to use S3 as a web server. See you there!