Sprint 2 research:

Upload Progress Bar Code:

```
<!DOCTYPE html>
<html lang=”en”>
<head>
 <meta charset=”UTF-8”>
 <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
 <title>File Upload Progress Bar</title>
 <style>
  /* Simple styles for the progress bar */
  #progress-bar-container {
   Width: 100%;
   Height: 25px;
   Background-color: #f3f3f3;
   Border: 1px solid #ccc;
   Border-radius: 5px;
  }
  #progress-bar {
   Height: 100%;
   Width: 0;
   Background-color: #4caf50;
   Text-align: center;
   Color: white;
   Border-radius: 5px;
  }
  #upload-status {
      Margin-top: 10px;
    }
  </style>
</head>
<body>

  <h2>File Upload with Progress Bar</h2>

  <input type=”file” id=”file-upload” />
  <button onclick=”uploadFile()”>Upload File</button>

  <div id=”progress-bar-container”>
    <div id=”progress-bar”>0%</div>
  </div>

  <div id=”upload-status”></div>

  <script src=”upload.js”></script>
</body>
</html>
function uploadFile() {
  const fileInput = document.getElementById('file-upload');
  const file = fileInput.files[0];

  if (!file) {
    alert("Please select a file to upload.");
    return;
  }

  const formData = new FormData();
  formData.append('file', file);

  const xhr = new XMLHttpRequest();

  // Set up progress tracking
  xhr.upload.addEventListener('progress', function (event) {
    if (event.lengthComputable) {
      const percent = (event.loaded / event.total) * 100;
      updateProgressBar(percent);
    }
  });

  // Handle upload completion
  xhr.onload = function() {
    if (xhr.status === 200) {
      updateProgressBar(100); // Set the progress bar to 100% on success
      document.getElementById('upload-status').textContent = "Upload Complete!";
    } else {
      document.getElementById('upload-status').textContent = "Upload Failed. Please try 
again.";
    }
  };
```

```
  // Handle errors
  xhr.onerror = function() {
    document.getElementById('upload-status').textContent = "Upload Failed. Please try
again.";
  };

  // Set the request method and URL for the server-side upload endpoint
  xhr.open('POST', '/upload', true);

  // Send the file data
  xhr.send(formData);
}

function updateProgressBar(percent) {
  const progressBar = document.getElementById('progress-bar');
  progressBar.style.width = percent + '%';
  progressBar.textContent = Math.round(percent) + '%';
}

npm install express multer
// server.js (Node.js with Express and Multer)
const express = require('express');
const multer = require('multer');
const path = require('path');
const app = express();

// Set up multer storage
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname));
  }
});

const upload = multer({ storage: storage });

app.post('/upload', upload.single('file'), (req, res) => {
  if (req.file) {
    res.status(200).send('File uploaded successfully');
  } else {
    res.status(400).send('File upload failed');
  }
});

// Serve the HTML page
app.use(express.static('public'));

// Start the server
```

```
app.listen(3000, () => {
  console.log('Server running on http://localhost:3000');
});
```

Amazon S3 metadata:

Amazon S3 handles metadata by associating it with objects stored in the service. Metadata in S3 is key-value data that provides additional information about the objects you store. This can include system-generated metadata, user-defined metadata, and HTTP headers used for storage and retrieval.

1. Types of Metadata in Amazon S3

There are two primary types of metadata in S3:

System Metadata: Managed by S3 and automatically assigned to objects. Examples include:

Content-Type: The MIME type of the object (e.g., image/jpeg, application/pdf).

Last-Modified: The timestamp when the object was last modified.

ETag: A unique identifier for the object, often used for caching and validating file integrity.

Content-Length: The size of the object in bytes.

Cache-Control, Content-Encoding, x-amz-storage-class, etc.

};

User Metadata: Custom metadata that you assign to an object. It allows you to attach arbitrary key-value pairs to objects for use in your applications. User metadata is typically set when uploading an object to S3. For example, you might store information like:

x-amz-meta-author: “John Doe”

x-amz-meta-project: “Project X”

2. Adding and Retrieving Metadata

When uploading an object: You can specify both system and user-defined metadata.

System metadata can be set via the S3 API or SDK (e.g., setting Content-Type, Cache-Control).

User metadata is added by including custom headers with the format x-amz-meta-<key>.

```
Const params = {
  Bucket: ‘your-bucket-name’,
  Key: ‘your-object-key’,
  Body: fileContent,
  Metadata: {
   ‘x-amz-meta-author’: ‘John Doe’,
   ‘x-amz-meta-project’: ‘Project X’,
  }
};
S3.putObject(params, function(err, data) {
  If (err) {
    Console.log("Error uploading file: ", err);
  } else {
    Console.log("File uploaded successfully!");
  }
});
```

When retrieving an object: You can fetch both system and user metadata by calling the HEAD operation (to get metadata of an object) or by retrieving the object and checking the metadata from the response headers.

```
Const params = { Bucket: 'your-bucket-name', Key: 'your-object-key' };
S3.headObject(params, function(err, data) {
  If (err) {
    Console.log("Error fetching metadata: ", err);
  } else {
    Console.log("Object metadata: ", data.Metadata);
  }
});
```

3. How Metadata is Processed in S3

System Metadata: This metadata is set automatically by S3 based on the operations performed. For example, when an object is uploaded, the Content-Type might be inferred from the file extension, or you might manually specify it. Similarly, S3 automatically sets the Last-Modified timestamp based on the upload time.

User Metadata: This metadata is set by you (the user) at the time of upload. It is stored in the object and can be accessed at any time but does not affect the functionality of S3 itself. The keys must be prefixed with x-amz-meta-, which is how S3 distinguishes user metadata from system metadata.

4. Metadata Limitations

Size: User-defined metadata has a size limit. Each object can have up to 2 KB of metadata. This includes both the keys and values, so you should keep your metadata as concise as possible.

Case Sensitivity: S3 metadata keys are case-insensitive, but the values are case-sensitive. For example, x-amz-meta-author and x-amz-meta-Author would be treated as the same key, but their values could differ.

5. Metadata in Object Operations

Updating Metadata: In S3, you cannot directly modify the metadata of an existing object. If you want to change the metadata of an object, you would need to:

Download the object.

Re-upload it with new metadata.

Optionally, delete the old version if you're using versioning.

Copying Objects with New Metadata: You can copy an object to the same or a different location within S3 and apply new metadata at the same time. This allows you to update metadata without needing to download the object and re-upload it.

```
Const params = {
  Bucket: 'your-bucket-name',
  CopySource: 'your-bucket-name/your-object-key',
  Key: 'new-object-key',
  Metadata: {
    'x-amz-meta-author': 'Jane Doe'
  },
  MetadataDirective: 'REPLACE'  // Important! Ensures metadata is replaced
};

S3.copyObject(params, function(err, data) {
  If (err) {
    Console.log("Error copying object: ", err);
  } else {
    Console.log("Object copied and metadata updated successfully!");
  }
});
```

6. Metadata Access via APIs

S3 GET and HEAD operations: When you retrieve an object via the GET operation, you get system metadata such as Content-Type, ETag, and Last-Modified. To get both system and user metadata, you can use the HEAD operation.

```
S3.headObject({ Bucket: ‘your-bucket-name’, Key: ‘your-object-key’ }, function(err, data) {
 If (err) {
   Console.log(‘Error retrieving metadata:’, err);
 } else {
   Console.log(‘Metadata:’, data.Metadata); // Custom user metadata
   Console.log(‘Content-Type:’, data.ContentType); // System metadata
 }
});
```

7. Metadata Use Cases

Search and Filtering: User metadata can be useful for tagging objects with custom attributes such as authors, projects, or file versions. You can later use this metadata for efficient searching or filtering in your applications.

Content Handling: You can use Content-Type, Cache-Control, and Content-Encoding to control how your objects are served to clients and cached.

Object Management: Use metadata for versioning, security purposes (like tracking object owners or permissions), and more complex management workflows.

Summary of Key Metadata Concepts in S3:

System Metadata: Automatically set by S3 (e.g., Last-Modified, Content-Type, ETag).

User Metadata: Custom key-value pairs (e.g., x-amz-meta-author) set by you at the time of upload.

Metadata Limits: Each object can hold up to 2 KB of metadata.

Metadata Access: Use GET or HEAD operations to retrieve metadata.

Metadata Update: To change metadata, you must re-upload or copy the object with new metadata.