IBM Innovation Center for Education

PHEMESOFT Software That Matters
TRUSTED DELIVERY PARTNER for IBM ICE

YENEPOYA UNIVERSITY

Final Project Report

on

Build a Scalable File Processing System

Team members:

Mohammed Ayad 22BDACC158

Guided by:

Ms. Manjula

SUMMARY

Project Title:

Build a Scalable File Processing System

Domain:

Cloud Computing & Virtualization

Objective:

The objective of this project is to design and implement a **serverless, scalable, and efficient file processing and management system** using **Amazon Web Services (AWS)**. The system ensures secure user authentication, automatic file handling via AWS Lambda, and metadata storage in DynamoDB, eliminating the need for traditional server management.

Description:

The project utilizes **AWS S3** for file storage, **AWS Lambda** for serverless file processing, and **DynamoDB** for fast and scalable metadata storage. A responsive **React.js** frontend allows users to upload, download, view, and manage their files.

The backend is developed using **Node.js** and **Express.js**, ensuring secure authentication and seamless integration with AWS services.

Files uploaded by users automatically trigger Lambda functions to process and store metadata, creating an **event-driven, serverless architecture**.

Technologies Used:

• **Frontend**: React.js, Tailwind CSS, Axios

• **Backend**: Node.js, Express.js

• **Cloud Services**: AWS Lambda, AWS S3, AWS DynamoDB

• **Database**: MongoDB Atlas (for user management)

• **Authentication**: JWT Tokens, HTTP-only Cookies

• **APIs Testing Tools**: Postman

Keywords: Cloud Computing, Serverless Architecture, AWS Lambda, AWS S3,

DynamoDB, React.js, Node.js, Scalable File Management, Event-Driven Systems

Table of Contents

1. BACKGROUND

1.1 Aim

To build a fully serverless and scalable file management system using AWS Lambda, S3, and DynamoDB, integrated through a responsive frontend and secure backend APIs.

1.2 Technologies

- **Frontend**: React.js, Tailwind CSS
- **Backend**: Node.js, Express.js
- **Cloud Services**: AWS Lambda, AWS S3, AWS DynamoDB
- **Database**: MongoDB Atlas (for user management)
- **Authentication**: JWT Tokens, HTTP-only Cookies
- **APIs Testing Tools**: Postman

1.3 Hardware Architecture

- Client devices (Laptop/Desktop/Mobile) → Internet → Cloud-hosted React app
- AWS cloud handles backend processing and storage.

1.4 Software Architecture

- Event-driven serverless model with React frontend and Node.js API backend.
- Backend connects with AWS Lambda (processing), S3 (storage), DynamoDB (metadata).

2. SYSTEM

2.1 Requirements

2.1.1 Functional Requirements

- User authentication (Login/Register)
- File upload, download, preview
- Metadata storage and retrieval
- File starring and un-starring

2.1.2 User Requirements

- Clean, intuitive UI
- Secure and responsive operations
- Quick access to uploaded files

2.1.3 Environmental Requirements

- AWS account for cloud service access
- Modern browser with JavaScript support

2.2 Design and Architecture

- DFD Level 0 and 1: show user interaction, backend processing, cloud services
- ERD: Users → Files → StarredFiles
- Class Diagram: AuthController, FileController, React Components

2.3 Implementation

- React frontend with protected routes and components
- Express.js API server with JWT-based auth
- AWS Lambda function triggered by S3 upload
- Metadata stored in DynamoDB, files in S3

2.4 Test Cases

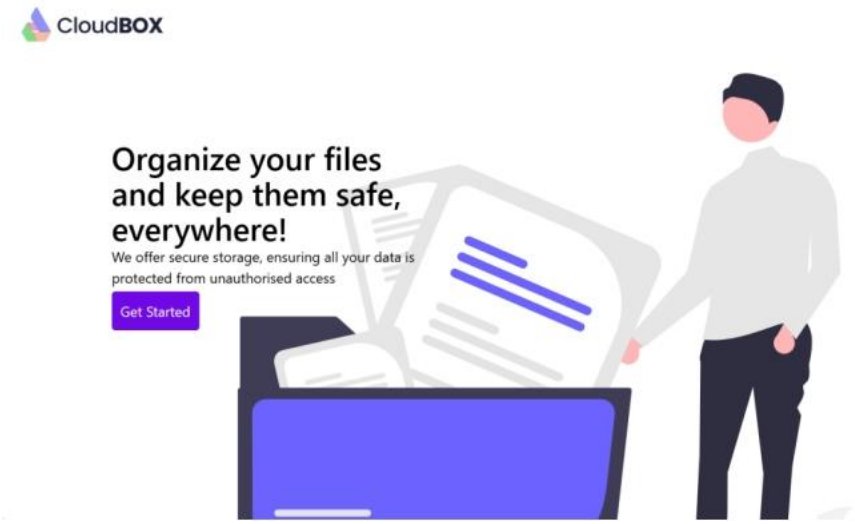
Here are some important test cases:

Test Case ID	Description	Input	Expected Output
TC001	User Login	Valid credentia ls	Dashboard page loads
TC002	User Login	Invalid credentials	Show error toast
TC003	Upload File	Select valid file	File saved to S3 metadata saved
TC004	Download File	Click download on file	File downloade d successfully
TC005	Star a File	Click star button	File added to starred list
TC006	Delete File	Click delete button	File removed from S3 and dashboard
TC007	Unauthorized Access	No token provided	Redirect to login page

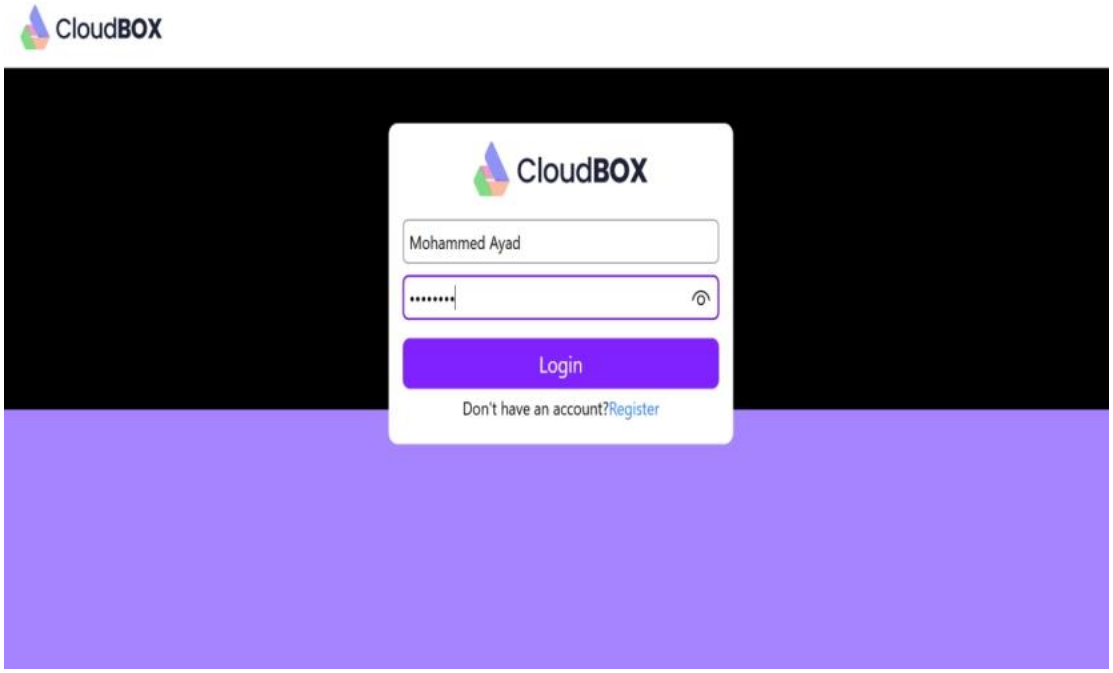
3. SNAPSHOTS OF THE PROJECT

10.1 User Interface Design

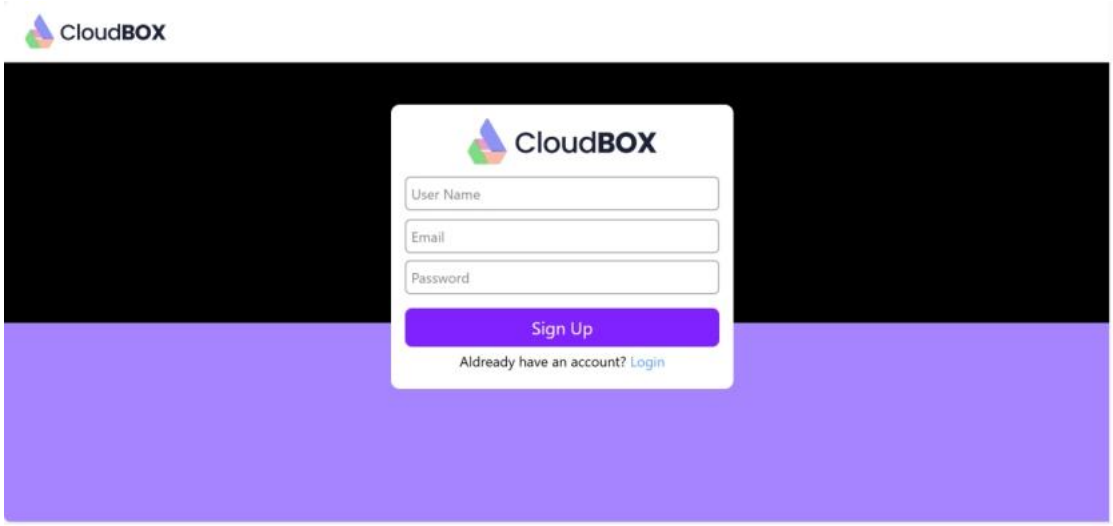
- **Home Page**



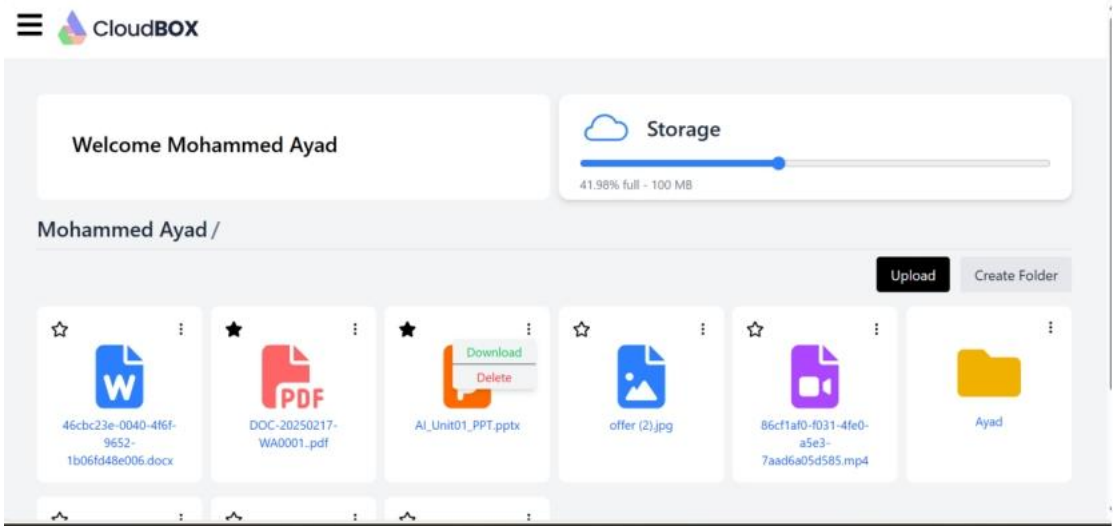
- **Login Page**



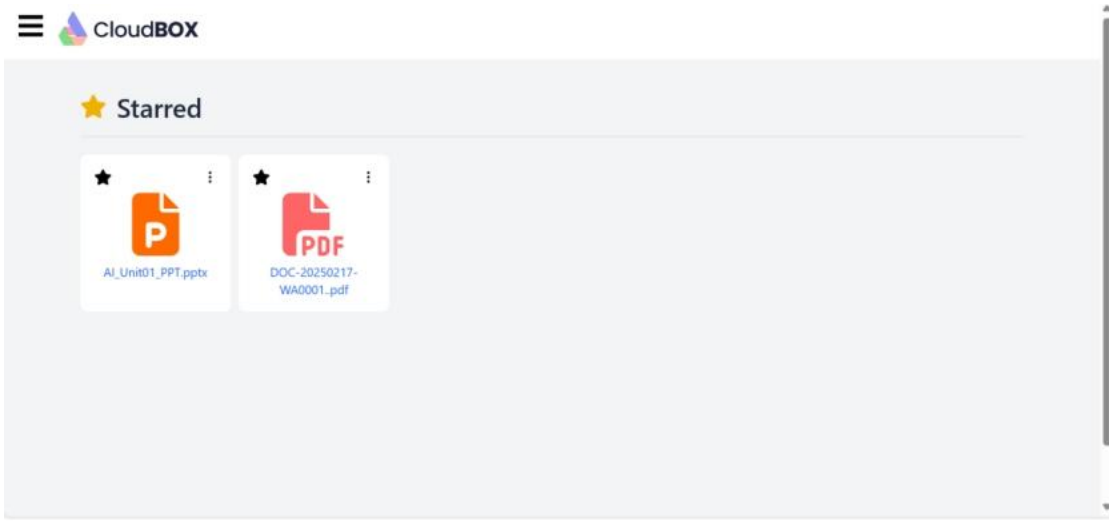
- **SignUp Page**



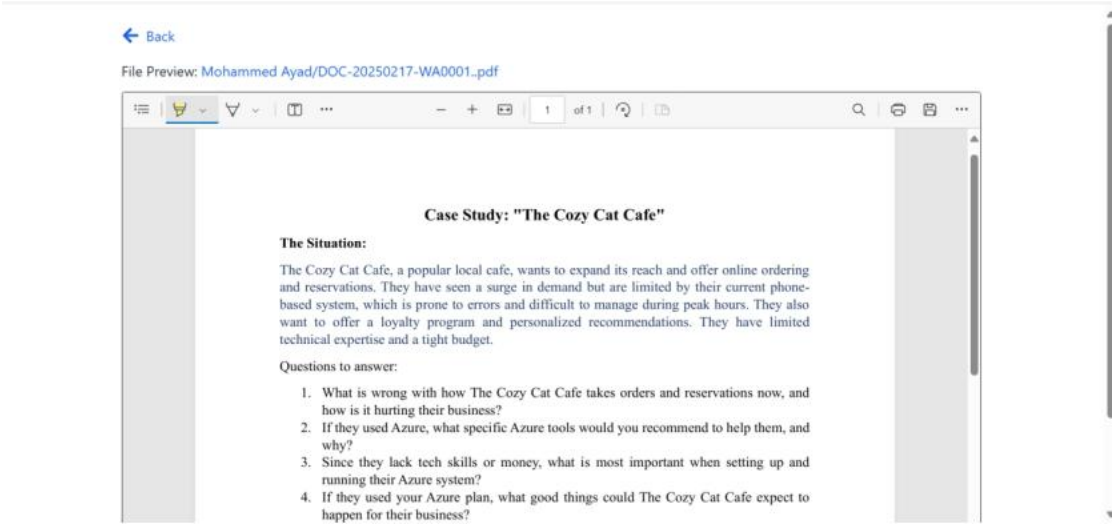
- **Dashboard Page**



- **Starred File Page**



- **Preview File Page**



Back

File Preview: Mohammed Ayad/DOC-20250217-WA0001..pdf

Case Study: "The Cozy Cat Cafe"

The Situation:

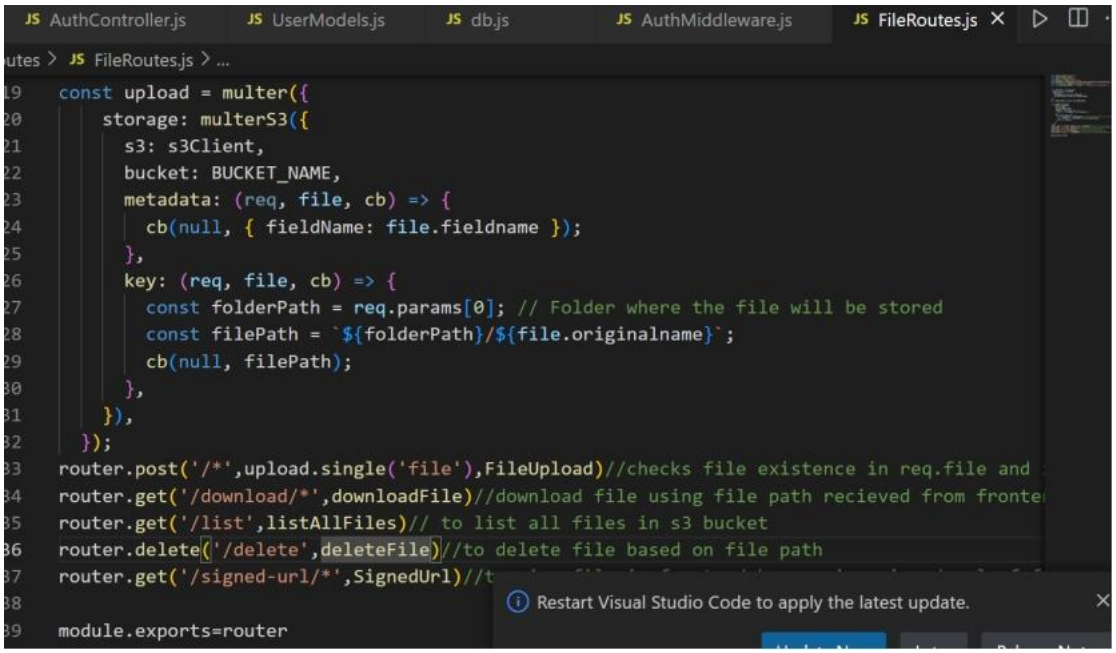
The Cozy Cat Cafe, a popular local cafe, wants to expand its reach and offer online ordering and reservations. They have seen a surge in demand but are limited by their current phone-based system, which is prone to errors and difficult to manage during peak hours. They also want to offer a loyalty program and personalized recommendations. They have limited technical expertise and a tight budget.

Questions to answer:

1. What is wrong with how The Cozy Cat Cafe takes orders and reservations now, and how is it hurting their business?
2. If they used Azure, what specific Azure tools would you recommend to help them, and why?
3. Since they lack tech skills or money, what is most important when setting up and running their Azure system?
4. If they used your Azure plan, what good things could The Cozy Cat Cafe expect to happen for their business?

10.2 Backend Interface Design

- **File Routes**



AuthController.js | UserModels.js | db.js | AuthMiddleware.js | FileRoutes.js

utes > FileRoutes.js > ...

```
const upload = multer({
    storage: multerS3({
      s3: s3Client,
      bucket: BUCKET_NAME,
      metadata: (req, file, cb) => {
        cb(null, { fieldName: file.fieldname });
      },
      key: (req, file, cb) => {
        const folderPath = req.params[0]; // Folder where the file will be stored
        const filePath = `${folderPath}/${file.originalname}`;
        cb(null, filePath);
      },
    }),
  });
router.post('/*',upload.single('file'),FileUpload)//checks file existence in req.file and
router.get('/download/*',downloadFile)//download file using file path recieved from fronte
router.get('/list',listAllFiles)// to list all files in s3 bucket
router.delete('/delete',deleteFile)//to delete file based on file path
router.get('/signed-url/*',SignedUrl)//t

module.exports=router
```

Restart Visual Studio Code to apply the latest update.

- **File Upload**

```
const FileUpload = async (req, res) => {
  try {
    const folderPath = req.params[0];
    const file = req.file;
    const { userName } = req.body;

    // 1. Check if file already exists in DB
    const existingFile = await DirectoryModel.findOne({
      name: file.originalname,
      type: file.mimetype,
      parentPath: folderPath,
    });

    if (existingFile) {
      return res.status(409).json({
        message: "File already exists in this folder",
        success: false,
      });
    }

    const folderSizeMB = await fetchFolderSize(userName); // Ensure you await async call

    const fileSizeMB = file.size / (1024 * 1024); // Convert file size from bytes to MB

    if (folderSizeMB + fileSizeMB >= 100) {
      const deleteCommand = new DeleteObjectCommand({
        Bucket: BUCKET_NAME,
        Key: `${folderPath}/${file.originalname}`,
      });

      await s3Client.send(deleteCommand);
      return res.status(409).json({
        message: "Upload exceeds 100 MB limit.",
        success: false,
      });
    }

    const newFile = new DirectoryModel({
      name: file.originalname,
      path: `${folderPath}/${file.originalname}`,
      parentPath: folderPath,
      type: file.mimetype,
    });

    await newFile.save();

    const updatedFolderData = await DirectoryModel.find({
      parentPath: folderPath,
    });

    return res.status(200).json({
      message: "File uploaded successfully",
      success: true,
      data: updatedFolderData,
    });
  } catch (err) {
    console.error("File upload error:", err);
    return res.status(500).json({
      message: "Internal server error",
      success: false,
    });
  }
};
```

- **Download File**

```
const downloadFile = async (req, res) => {
  try {
    Click to collapse the range.
    [0];

    if (!filePath || filePath.includes("..")) {
      return res
        .status(400)
        .json({ message: "Invalid file path", success: false });
    }

    const command = new GetObjectCommand({
      Bucket: BUCKET_NAME,
      Key: filePath,
    });

    const response = await s3Client.send(command);

    res.setHeader(
      "Content-Disposition",
      `attachment; filename="${filePath.split("/").pop()}"`
    );
    res.setHeader(
      "Content-Type",
      response.ContentType || "application/octet-stream"
    );
    res.setHeader("Content-Length", response.ContentLength);

    response.Body.on("error", (err) => {
      console.error("Stream error:", err);
      res.status(500).end("File stream error");
    });

    response.Body.pipe(res);
  } catch (error) {
    if (error.name === "NoSuchKey") {
      return res
        .status(404)
        .json({ message: "File not found", success: false });
    }
    console.error("Download error:", error.stack || error);
    res.status(500).json({ message: "Internal server error", success: false });
  }
};
```

- **Delete File**

```
const deleteFile = async (req, res) => {
  try {
    const { path, parentPath } = req.body;
    const command = new DeleteObjectCommand({ Bucket: BUCKET_NAME, Key: path });
    await s3Client.send(command);
    await DirectoryModel.findOneAndDelete({ path: path });
    await StarredModel.findOneAndDelete({ path });
    const folderData = await DirectoryModel.find({ parentPath });
    res
      .status(200)
      .json({
        message: "File deleted successfully",
        success: true,
        data: folderData,
      });
  } catch (error) {
    res.status(500).json({ message: "Internal server error", success: false });
  }
};

const SignedUrl = async (req, res) => {
  try {
    const path = req.params[0]; // if your route is router.get("/*", SignedUrl)

    if (!path) {
      return res.status(400).json({
        message: "Missing file path in request",
        success: false,
      });
    }

    const command = new GetObjectCommand({
      Bucket: BUCKET_NAME,
      Key: path,
    });

    const signedUrl = await getSignedUrl(s3Client, command, {
      expiresIn: 3600,
    }); // 1 hour expiry

    res.status(200).json({
      success: true,
      message: "Signed URL generated",
      url: signedUrl,
    });
  } catch (error) {
    console.error("Error generating signed URL:", error);
    return res.status(500).json({
      success: false,
      message: "Internal server error",
      error: error.message,
    });
  }
```

- **Signed Url**

```
const SignedUrl = async (req, res) => {
  try {
    const path = req.params[0]; // if your route is router.get("/*", SignedUrl)

    if (!path) {
      return res.status(400).json({
        message: "Missing file path in request",
        success: false,
      });
    }

    const command = new GetObjectCommand({
      Bucket: BUCKET_NAME,
      Key: path,
    });

    const signedUrl = await getSignedUrl(s3Client, command, {
      expiresIn: 3600,
    }); // 1 hour expiry

    res.status(200).json({
      success: true,
      message: "Signed URL generated",
      url: signedUrl,
    });
  } catch (error) {
    console.error("Error generating signed URL:", error);
    return res.status(500).json({
      success: false,
      message: "Internal server error",
      error: error.message,
    });
  }
};
```

- **Folder Routes**

```
Routes > JS FolderRoutes.js > {} <unknown>
const { DownloadFolder } = require('../Controller/DownloadFolder')
const { FetchFolder, CreateFolder, DeleteFolder, getFolderSizeInMB } = require('../Controller/FolderController')

const router=require('express').Router()

router.get('/fetch/*',FetchFolder)//to fetch folder and list all files and folder under path
router.post('/*',CreateFolder)//to create new folder
router.delete('/',DeleteFolder)//delete folder and files under folder path
router.get('/download/*',DownloadFolder)//to download folder and files under folder path
router.get('/size/*',getFolderSizeInMB)//to get of folder path


module.exports=router
```

- **Fetch Folder**

```
const FetchFolder = async (req, res) => {
  try {
    const folderPath = req.params[0];
    if (!folderPath) {
      return res
        .status(400)
        .json({ message: "Folder path is required", success: false });
    }
    const folderData = await DirectoryModel.find({ parentPath: folderPath });
    res
      .status(200)
      .json({ message: "Folder fetched", success: true, data: folderData });
  } catch (error) {
    res
      .status(500)
      .json({ message: "Internal server error", success: false, error });
  }
};
```

- **Create Folder**

```
const createFolder = async (req, res) => {
  try {
    const folderPath = req.params[0];
    const { folderName } = req.body;
    if (!folderName) {
      return res
        .status(409)
        .json({ message: "Folder name required", success: false });
    }
    const filePath = `${folderPath}/${folderName}`;
    const isFolderExist = await DirectoryModel.findOne({
      name: folderName,
      parentPath: folderPath,
      type: "folder",
    });
    if (isFolderExist) {
      return res
        .status(409)
        .json({ message: "Folder already exist", success: false });
    }
    const command = new PutObjectCommand({
      Key: `${filePath}/`,
      Bucket: BUCKET_NAME,
    });
    await s3Client.send(command);
    const newFolder = new DirectoryModel({
      name: folderName,
      parentPath: folderPath,
      path: filePath,
      type: "folder",
    });
    await newFolder.save();
    const folderData = await DirectoryModel.find({ parentPath: folderPath });
    res.status(200).json({ message: "Folder created", success: true ,data:folderData});
  } catch (error) {
    res
      .status(500)
      .json({ message: "Internal server error", success: false, error });
  }
};
```

- **Delete Folder**

```
const DeleteFolder = async (req, res) => {
  try {
    const {path,parentPath}= req.body
    const folderKey = `${path}/`;
    const listCommand = new ListObjectsV2Command({
      Bucket: BUCKET_NAME,
      Prefix: folderKey,
    });
    const listedObjects = await s3Client.send(listCommand);

    if (!listedObjects.Contents || listedObjects.Contents.length === 0) {
      return res
        .status(404)
        .json({ message: "Folder not found or empty folder" });
    }

    const objectsToDelete = listedObjects.Contents.map((item) => ({
      Key: item.Key,
    }));

    const deleteCommand = new DeleteObjectsCommand({
      Bucket: BUCKET_NAME,
      Delete:{
        Objects:objectsToDelete
      }
    });
    await s3Client.send(deleteCommand);
    await DirectoryModel.deleteMany({
      $or: [
        { path: { $regex: `^${path}` } },
        { parentPath: { $regex: `^${path}` } },
      ],
    });
    const folderData=await DirectoryModel.find({parentPath})
     res.status(200).json({message:"Folder deleted",success:true,data:folderData})

  } catch (error) {
    res.status(500).json({message:"Internal server error",success:false,error})
  }
};
```

- ## Download Folder

```
const fs = require("fs-extra");
const path = require("path");
const archiver = require("archiver");
const {
  S3Client,
  ListObjectsV2Command,
  GetObjectCommand,
} = require("@aws-sdk/client-s3");


const s3Client = new S3Client({
    region: process.env.REGION,
    credentials: {
      accessKeyId: process.env.ACCESS_KEY,
      secretAccessKey: process.env.ACCESS_SECRET,
    },
  });
  const BUCKET_NAME = process.env.BUCKET_NAME;

// Download folder as zip
const DownloadFolder= async (req, res) => {
  const folderKey = req.params[0]; // e.g. 'projects/demo'

  const tempFolder = path.join(__dirname, "temp", folderKey); // temp/projects/demo
  const zipPath = path.join(__dirname, "temp", `${path.basename(folderKey)}.zip`);

  try {
    // Step 1: List all S3 objects with the given prefix
    const listCommand = new ListObjectsV2Command({
      Bucket: BUCKET_NAME,
      Prefix: `${folderKey}/`, // trailing slash to limit to the folder
    });
    const listedObjects = await s3Client.send(listCommand);
    if (!listedObjects.Contents || listedObjects.Contents.length === 0) {
      return res.status(404).json({ message: "Folder not found", success: false });
    }
    console.log('step 1');

    // Step 2: Download each file to temp directory
    for (const object of listedObjects.Contents) {
      const key = object.Key;
      console.log(key);

      if (key.endsWith("/")) continue; // skip empty folders
```

```
    await zipFolder(path.join(__dirname, "temp", folderKey), zipPath);

    console.log('step 3');
    let stat;
    try {
      stat = fs.statSync(zipPath);
      res.setHeader('Content-Length', stat.size);
    } catch (err) {
      console.error('Stat error:', err);
      return res.status(500).json({ message: "Error preparing file", success: false });
    }

    // Step 4: Send the zip file
    res.download(zipPath, `${path.basename(folderKey)}.zip`, async (err) => {
      // Optional: cleanup temp files after sending
      await fs.remove(path.join(__dirname, "temp"));
      if (err) {
        console.error("Download error:", err);
      }
    });
    console.log('step 4');

  } catch (error) {
    console.error("Download error:", error);
    res.status(500).json({ message: "Internal server error", success: false });
  }
}

// Helper to stream to buffer
const streamToBuffer = async (stream) => {
  return new Promise((resolve, reject) => {
    const chunks = [];
    stream.on("data", (chunk) => chunks.push(chunk));
    stream.on("end", () => resolve(Buffer.concat(chunks)));
    stream.on("error", reject);
  });
};

// Helper to zip a folder
const zipFolder = (sourceFolder, outPath) => {
  return new Promise((resolve, reject) => {
    const archive = archiver("zip", { zlib: { level: 9 } });
    const output = fs.createWriteStream(outPath);

    output.on("close", resolve);
    archive.on("error", reject);

    archive.pipe(output);
    archive.directory(sourceFolder, false);
    archive.finalize();
  });
};
```

- ## Get Folder Size

```
const getFolderSizeInMB = async (req,res) => {
  try {
    let continuationToken = undefined;
    let totalSizeInBytes = 0;
    const folderPath=req.params[0]

    do {
      const command = new ListObjectsV2Command({
        Bucket: BUCKET_NAME,
        Prefix: `${folderPath}/`,
        ContinuationToken: continuationToken,
      });

      const response = await s3Client.send(command);

      if (response.Contents) {
        response.Contents.forEach((object) => {
          if (!object.Key.endsWith("/")) {
            totalSizeInBytes += object.Size || 0;
          }
        });
      }

      continuationToken = response.IsTruncated ? response.NextContinuationToken : undefined;
    } while (continuationToken);

    const sizeInMB = totalSizeInBytes / (1024 * 1024);
    res.status(200).json({message:'Size fetched',size:sizeInMB.toFixed(2),success:true})
  } catch (error) {
    res.status(500).json({message:'Internal server error',error,success:false})
  }
};
```

- **Auth Routes**

```
const { SignUp, Login, verifyToken, SignOut } = require('../Controller/AuthController')
const { LoginValidation, SignUpValidation } = require('../Middlewares/AuthMiddleware')

const router=require('express').Router()

router.post('/login',LoginValidation,Login)//for login validation middleware and login and passing jwtToken through
router.post('/signup',SignUpValidation,SignUp)//for Sign Up validation middleware and login
router.get('/verify-token',verifyToken)//to verify user and pass userName
router.post('/logout',SignOut)//to logout


module.exports=router
```

- **Login**

```
const Login = async (req, res) => {
  try {
    const { userName, password } = req.body;

    if (!userName || !password) {
      return res.status(400).json({ message: "Please fill all fields" ,success:false});
    }

    const user = await UserModel.findOne({ userName });

    if (!user) {
      return res
        .status(400)
        .json({ message: "User not found", success: false });
    }

    const comparePassword = await bcrypt.compare(password, user.password);

    if (!comparePassword) {
      return res
        .status(400)
        .json({ message: "Incorrect password", success: false });
    }
    const key = process.env.JWT_SECRET_KEY;
    const jwtToken = jwt.sign({ userName }, key, { expiresIn: "5d" });

   res.cookie("token", jwtToken, {
  httpOnly: true,
  secure: false, // Set to true in production with HTTPS
  sameSite: "Lax",
});

    res
      .status(200)
      .json({ message: "SignIn succesfully", success: true, jwtToken });
  } catch (error) {
    res.status(500).json({ message: "Internal error", error, success: false });
  }
};
```

- **SignUp**

```
const SignUp = async (req, res) => {
  try {
    const { email, userName, password } = req.body;

    if (!email || !userName || !password) {
      return res
        .status(404)
        .json({ message: "Please fill all fields", success: false });
    }

    const isEmailExist = await UserModel.findOne({ email }); // it is not showing suggesion for .findOne
    const isUserNameExist = await UserModel.findOne({ userName });

    if (isEmailExist) {
      return res
        .status(400)
        .json({ message: "Email Already Exist", success: false });
    }
    if (isUserNameExist) {
      return res
        .status(400)
        .json({ message: "UserName already Exist", success: false });
    }

    const newUser = new UserModel({ email, userName, password });
    newUser.password = await bcrypt.hash(password, 10);
    await newUser.save();
    const s3Client = new S3Client({
      region: process.env.REGION,
      credentials: {
        accessKeyId: process.env.ACCESS_KEY,
        secretAccessKey: process.env.ACCESS_SECRET,
      },
    });
    const BUCKET_NAME = process.env.BUCKET_NAME;
    const command = new PutObjectCommand({
      Bucket: BUCKET_NAME,
      Key: `${userName}/`,
    });
    const respond = await s3Client.send(command);
    const newDirectory=new DirectoryModel({name:userName,type:"folder",path:`${userName}/`})
    await newDirectory.save()
    res
      .status(201)
      .json({ message: "Signup successfully", success: true });
  } catch (error) {
    res.status(500).json({ message: "Internal server error", success: false });
  }
};
```

- **Verify Token**

```
const verifyToken = async (req, res) => {
  try {
    const token = req.cookies.token;

    if (!token) {
      return res.status(400).json({ message: "Access Denied", success: false });
    }
    const key = process.env.JWT_SECRET_KEY;
    const verify = jwt.verify(token, key);
    if (verify) {
      res.status(200).json({ message: "", success: true, user: verify });
    } else {
      res.status(400).json({ message: "Access denied", success: false });
    }
  } catch (error) {
    res
      .status(500)
      .json({ message: "Internal sever error ", success: false, error });
  }
};
```

- **Sign Out**

```
const verifyToken = async (req, res) => {
  try {
    const token = req.cookies.token;

    if (!token) {
      return res.status(400).json({ message: "Access Denied", success: false });
    }
    const key = process.env.JWT_SECRET_KEY;
    const verify = jwt.verify(token, key);
    if (verify) {
      res.status(200).json({ message: "", success: true, user: verify });
    } else {
      res.status(400).json({ message: "Access denied", success: false });
    }
  } catch (error) {
    res
      .status(500)
      .json({ message: "Internal sever error ", success: false, error });
  }
};
```

- **Starred Routes**

```
const { addToStarred, getStarredFiles, removeFromStarred } = require('../Controller/StarredContoller')

const router=require('express').Router()


router.post('/',addToStarred)//put to starred
router.get('/:userName',getStarredFiles)//fetch starred
router.delete('/:id',removeFromStarred)//remove from starred

module.exports=router
```

- **Add to Starred**

```
const addToStarred = async (req, res) => {
  try {
    const { _id, name, path, parentPath, type, userName } = req.body;
    const newStarredFile = new StarredModel({
      name,
      path,
      parentPath,
      type,
      fileId: _id,
      userName,
    });
    await newStarredFile.save();
    const data = await StarredModel.find({ userName });
    res
      .status(200)
      .json({ message: "File added to starred", success: true, data });
  } catch (error) {
    res
      .status(500)
      .json({ message: "Internal server error", success: false, error });
  }
};
```

- **Get from Starred**

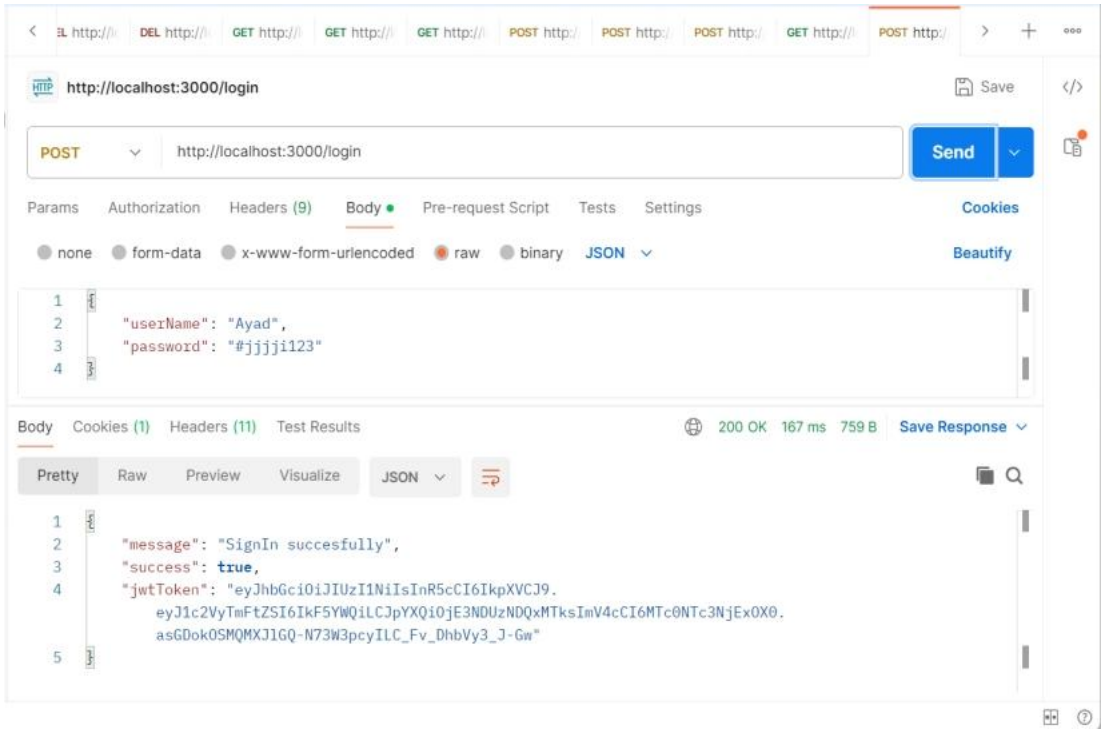
```
const getStarredFiles = async (req, res) => {
  try {
    const { userName } = req.params;
    // If userId is stored in the model, consider using req.user.id
    const data = await StarredModel.find({ userName });
    res
      .status(200)
      .json({ message: "Fetched starred files", success: true, data });
  } catch (error) {
    res
      .status(500)
      .json({ message: "Internal server error", success: false, error });
  }
};
```

- **Remove from Starred**

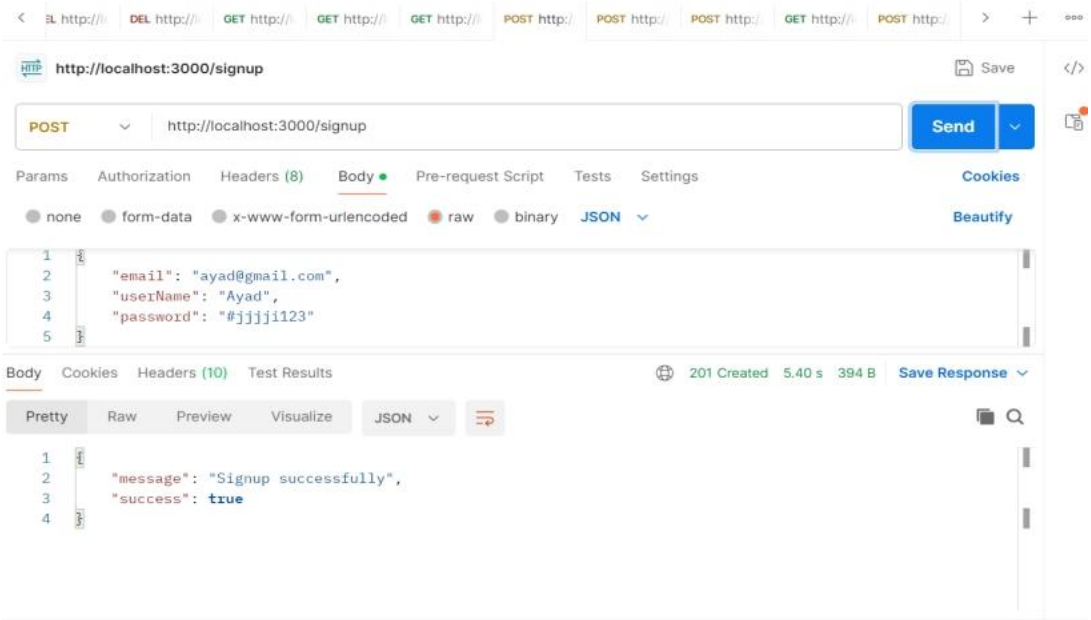
```
const removeFromStarred = async (req, res) => {
  try {
    const { id } = req.params;
    const { userName } = req.query;

    await StarredModel.findOneAndDelete({ fileId: id });

    const data = await StarredModel.find({userName});
    res
      .status(200)
      .json({ message: "File removed from starred", success: true, data });
  } catch (error) {
    res
      .status(500)
      .json({ message: "Internal server error", success: false, error });
  }
};
```

Backend API Testing Screenshots

- **User Login API**



- **User SignUp API**



- **File Upload API**

http://localhost:3000/file/Ayad

POST http://localhost:3000/file/Ayad

Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary

☑	file	3.2.2.png	Auto
☑	userName	Ayad	Auto
	Key	Value	Auto

Body Cookies (1) Headers (10) Test Results 200 OK 14.64 s 648 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "message": "File uploaded successfully",
    "success": true,
    "data": [
        {
            "_id": "6807d6a9f41bf91fbd48031e",
            "name": "Folder",
            "path": "Ayad/Folder",
```


- **Folder Upload Api**

http://localhost:3000/folder/Ayad

POST http://localhost:3000/folder/Ayad

Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary JSON Beautify


```
{
    "folderName": "Folder"
}
```

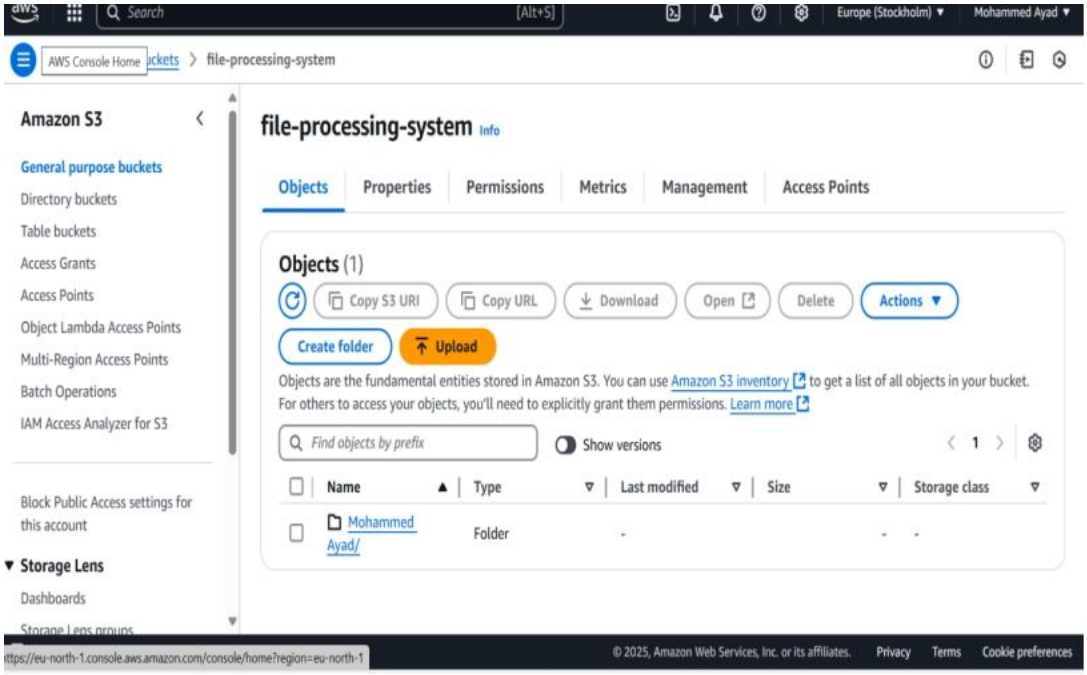

Body Cookies (1) Headers (10) Test Results 200 OK 824 ms 510 B Save Response

Pretty Raw Preview Visualize JSON

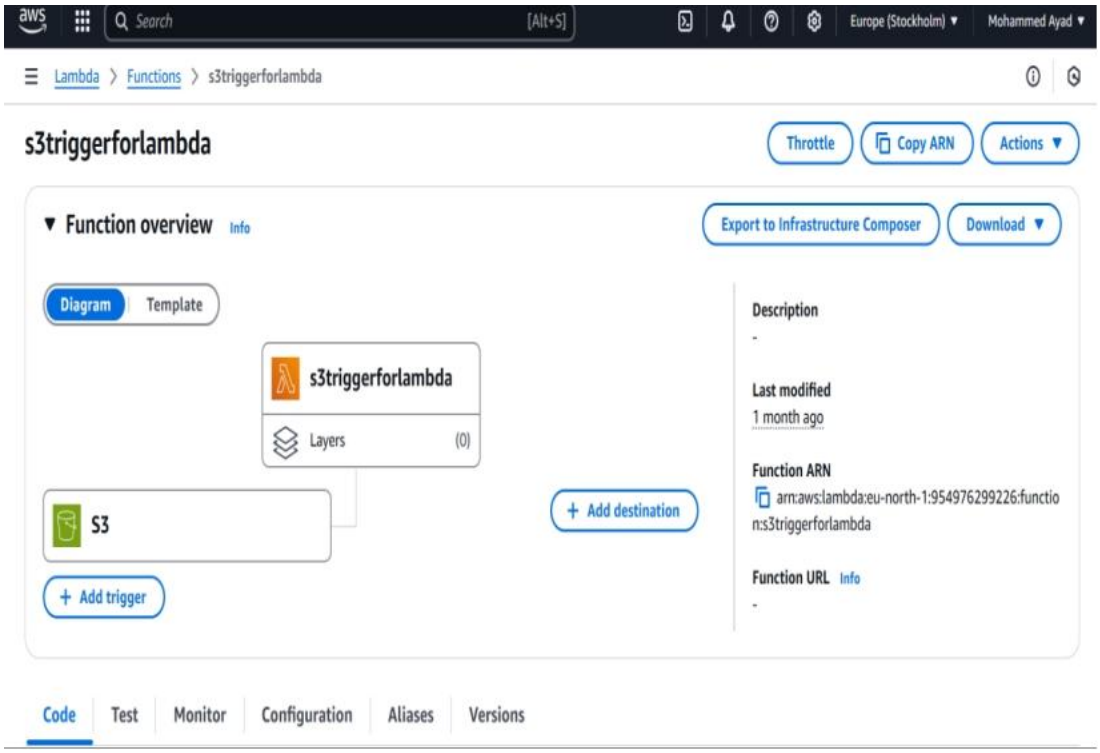

```
{
    "message": "Folder created",
    "success": true,
    "data": [
        {
            "_id": "6807d6a9f41bf91fbd48031e",
            "name": "Folder",
            "path": "Ayad/Folder",
```

AWS Console Screenshots

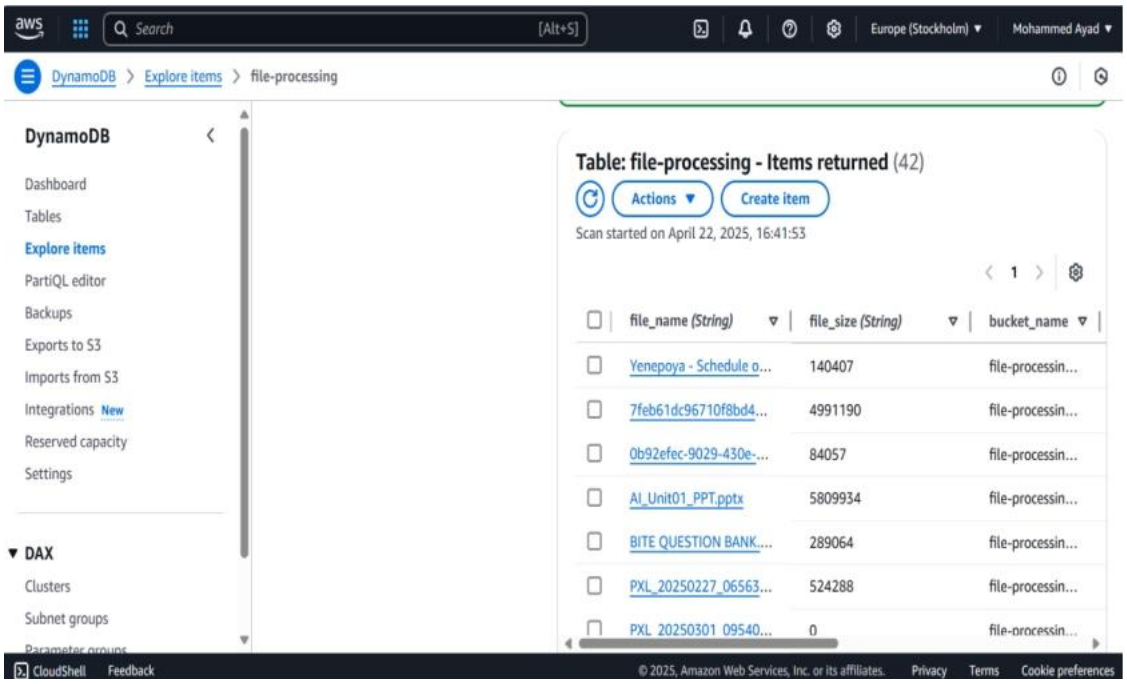
- **S3 buckets**



- **Lambda triggers**



DynamoDB tables.



MongoDB Model Screenshot

Directory

Models > DirectoryModel.js > DirectorySchema > type

```
const mongoose=require('mongoose')

const DirectorySchema=new mongoose.Schema({
    name:{
        required:true,
        type:String,
    },
    path:{
        required:true,
        type:String,
    },
    parentPath:{
        required:true,
        type:String,
        default:"Users"
    },
    type:{
        required:true,
        type:String
    }
})

const DirectoryModel=mongoose.model('Directory',DirectorySchema)
```

Users

Models > JS UserModels.js > UserSchema > userName > unique

```
const mongoose=require('mongoose')

const UserSchema=new mongoose.Schema({
    userName:{
        required:true,
        type:String,
        unique:true
    },
    email:{
        required:true,
        type:String,
        unique:true
    },
    password:{
        required:true,
        type:String,
    },

})

const UserModel=mongoose.model('users',UserSchema)
module.exports=UserModel
```

Starred Files

```
const StarredSchema=new mongoose.Schema({
    fileId:{
        type:mongoose.Schema.Types.ObjectId,
        ref:'Directory',
        required:true
    },
    name:{
        required:true,
        type:String,
    },
    path:{
        required:true,
        type:String,
    },
    parentPath:{
        required:true,
        type:String,
        default:"Users"
    },
    type:{
        required:true,
        type:String
    },
    userName:{
        required:true,
        type:String
    }
})
```

4. CONCLUSIONS

6.1 Introduction

The project titled **"Build a Scalable File Processing System"** was undertaken with the goal of designing a cloud-native, serverless file management solution using AWS technologies. Throughout the project, an emphasis was placed on building a system that is **scalable**, **secure**, **cost-efficient**, and **highly available**.

6.2 Achievements

- The main objectives of the project were successfully achieved:
- Developed a fully **serverless** backend using **AWS Lambda**, **S3**, and **DynamoDB**.
- Implemented a **responsive, user-friendly frontend** with React.js and Tailwind CSS and TypeScript.
- Enabled **secure user authentication** using JWT tokens and HTTP-only cookies.
- Built **REST APIs** for handling file operations such as upload, download, delete, and star.
- Implemented **robust security measures** across all components.

Additionally, a strong understanding of **event-driven architectures**, **cloud-based file management**,and **serverless computing models** was developed, fulfilling the learning objectives of the project.

6.3 Key Outcomes

1. **Efficiency**: Files are processed instantly with event triggers.
2. **Scalability**: No need to worry about server load — AWS services scale automatically.
3. **Reliability**: S3 provides 99.999999999% (11 nines) durability.
4. **Security**: The system enforces authentication, authorization, and secure data storage.

6.4 Conclusion Statement

In conclusion, the project successfully demonstrates how modern cloud services and serverless architectures can be utilized to build powerful, scalable, and cost-effective file management systems without managing traditional servers.The skills learned during this project are highly relevant to today's cloud computing industry.

5. FURTHER DEVELOPMENT / RESEARCH

5.1 Introduction

Although the system in its current form meets the basic requirements,there are several ways it could be extended and enhanced in the future to add more features and scalability.

5.2 Suggested Future Enhancements

5.2.1 Multi-User Role Management

- **Admin**, **Editor**, and **Viewer** roles can be added to provide controlled access based on user types.

5.2. 2 File Sharing Functionality

- Enable secure sharing links so that users can share uploaded files with others externally.

5.2. 3 Version Control

- Implement file versioning using AWS S3 so that users can retrieve older versions of a file.

5.2.4 Multi-File and Folder Upload

- Allow users to upload multiple files and even full folders in a single action, enhancing usability.

5.2.5 File Compression and Optimization

- Integrate functionality where large files are automatically compressed before upload to save storage space.

5.2.6 AI-based File Tagging (Advanced Feature)

- Implement machine learning models to automatically classify and tag uploaded files based on content (e.g., images, documents).

5.2. 7 Mobile App Development

- Build a mobile application using React Native or Flutter to allow users to manage files directly from smartphones.

5.3 Conclusion

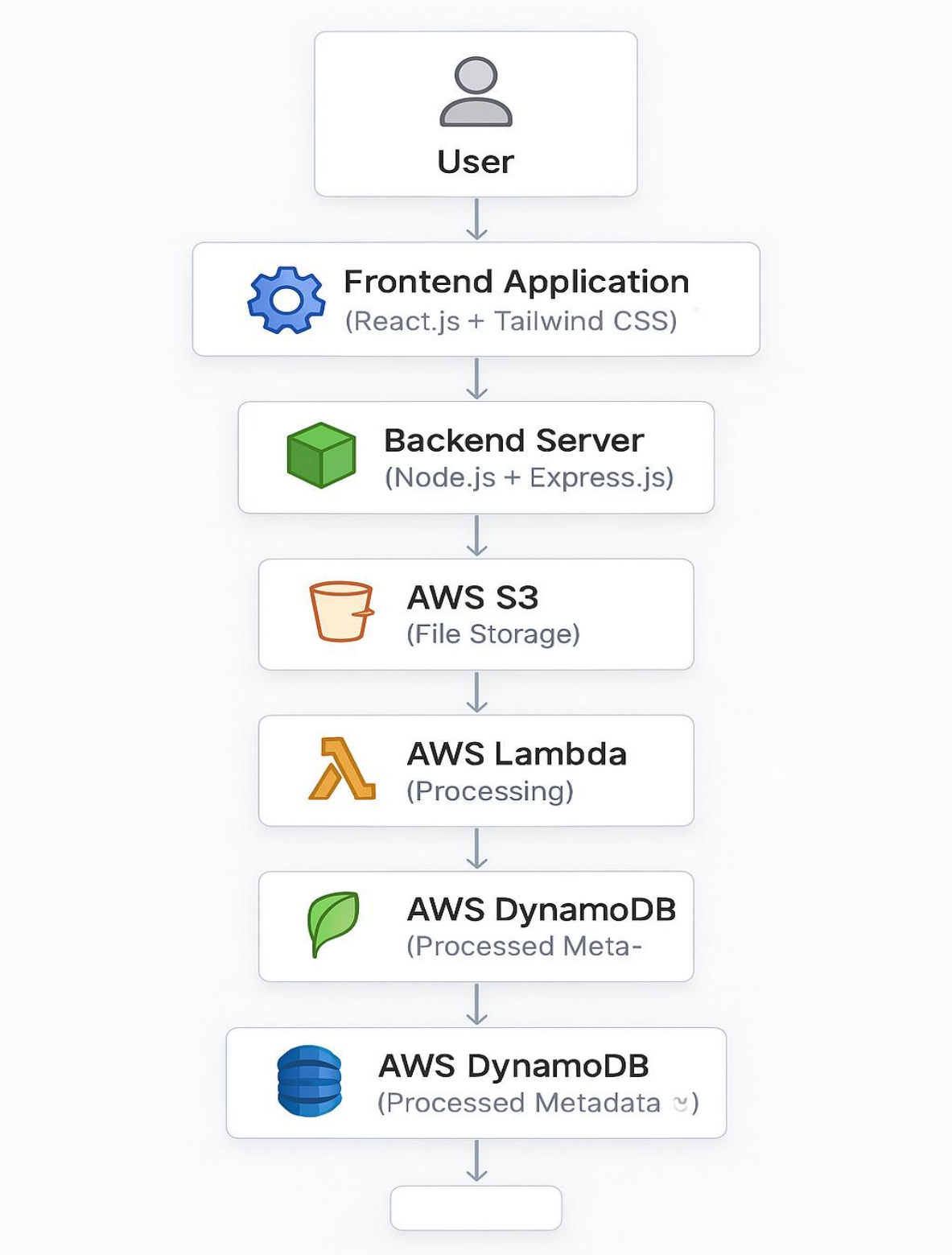
Future enhancements can greatly increase the functionality, usability, and value of the system. Incorporating advanced features like AI tagging, mobile access, and role-based security would further modernize the platform and make it suitable for large-scale deployment in production environments

6. REFERENCES

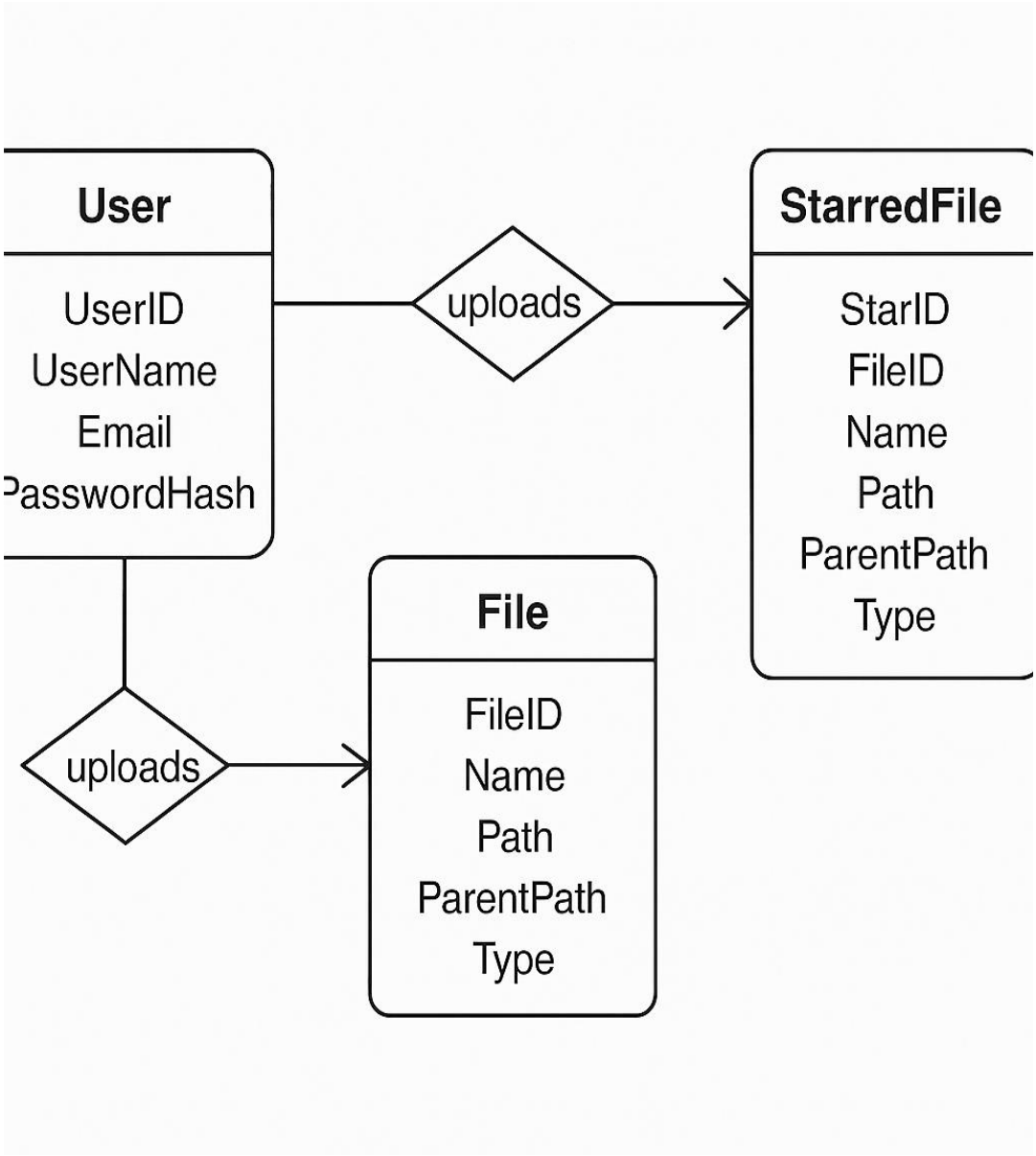
1. **Amazon Web Services (AWS) Documentation**
   https://docs.aws.amazon.com
2. **AWS Lambda Developer Guide**
   https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
3. **AWS S3 Developer Guide**
   https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
4. **AWS DynamoDB Developer Guide**
   https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
5. **React.js Official Documentation**
   https://react.dev
6. **Amazon Web Services (AWS) Documentation**
   https://docs.aws.amazon.com
7. **AWS Lambda Developer Guide**
   https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
8. **AWS S3 Developer Guide**
   https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
9. **AWS DynamoDB Developer Guide**
   https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
10. **React.js Official Documentation**
    https://react.dev
11. **Node.js Official Documentation**
    https://nodejs.org/en/docs/
12. **Express.js Guide**
    https://expressjs.com/
13. **MongoDB Official Documentation**
    https://www.mongodb.com/docs/
14. **Tailwind CSS Documentation**
    https://tailwindcss.com/docs
15. **Express.js Guide**
    https://expressjs.com/
16. **MongoDB Official Documentation**
    https://www.mongodb.com/docs/
17. **Tailwind CSS Documentation**
    https://tailwindcss.com/docs

7. APPENDIX

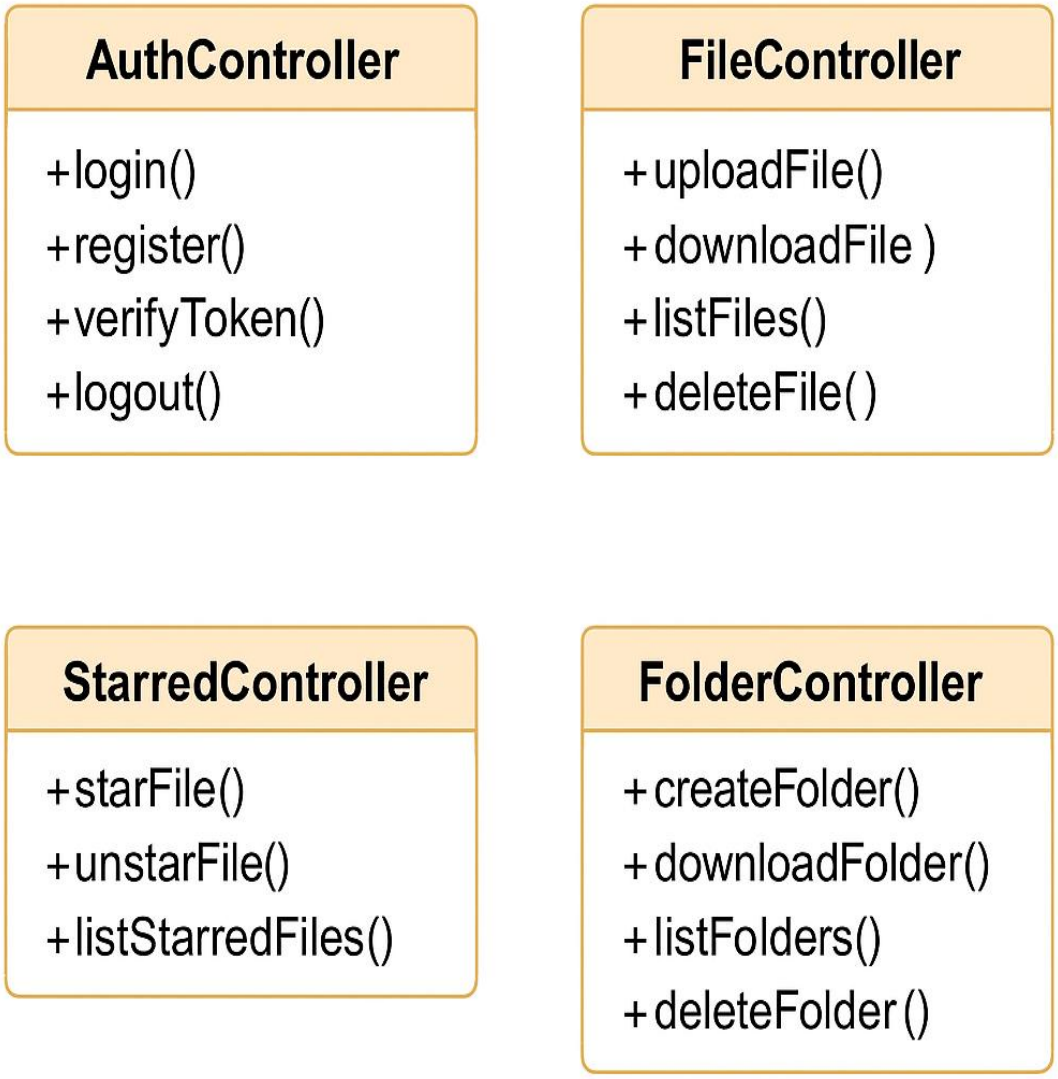
DFD Diagrams



Entity-Relationship Diagram (ERD)



Class Diagram



Test Case Sheets

Test Case ID	Description	Input	Expected Output
TC001	User Login	Valid credentia ls	Dashboard page loads
TC002	User Login	Invalid credentials	Show error toast
TC003	Upload File	Select valid file	File saved to S3 metadata saved
TC004	Download File	Click download on file	File downloade d successfully
TC005	Star a File	Click star button	File added to starred list
TC006	Delete File	Click delete button	File removed from S3 and dashboard
TC007	Unauthorized Access	No token provided	Redirect to login page

Weekly Progress Table

Build a Scalable File Processing System

Mohammed Ayad

Week	Activities/Work Completed
Week 1	Project planning: selected technologies: AWS Lambda, S3, DynamoDB, React.js, Node.js
Week 2	Frontend design using React.js and Tailwind CSS
Week 3	Backend API development for file handling and user authentication
Week 4	Configured AWS Lambda functions and DynamoDB integration
Week 5	System testing, bug fixing, optimization; project documentation