

Name : Hamza Jamshaid

Project title :

Implement a Cloud-Based File Storage System

Introduction:

Internal management systems often require secure storage and controlled access to sensitive documents such as joining letters, certificates, and reports. Traditional local storage solutions lack scalability, security, and remote accessibility. To address these challenges, this project implements a **Cloud-Based File Storage System** using **Microsoft Azure Blob Storage**, providing a secure, scalable, and efficient solution for document management.

Objectives:

The specific objectives include:

- Setting up Azure Blob Storage for cloud file management.
- Organizing documents into structured folders such as joining letters, certificates, and reports.
- Ensuring secure access to files using signed URLs (Shared Access Signatures).
- Providing controlled and temporary access to documents without exposing storage credentials.

Project Structure:

```
azure-file-storage-system/
├── server.js
├── .env
└── controllers/
    └── fileController.js
└── routes/
    └── fileRoutes.js
└── package.json
└── README.md
```

Project Procedure:

Below is a high-level breakdown of all procedures performed in the project. Each step mentions only actions and file names.

1. Create Storage account:

- Create a Storage account
- Create Container
- Get Storage Credentials

The screenshot shows two related Azure pages. The top page is the 'internfilestorage_1767848267683 | Overview' for a deployment. It displays a green checkmark icon and the message 'Your deployment is complete'. Below this, it shows deployment details: Deployment name: internfilestorage_1767848267683, Subscription: Azure subscription 1, Resource group: project2-rg. It also shows start time: 1/8/2026, 10:00:24 AM and Correlation ID: d3fb804-0d97-48a6-9ad1-e97486768a20. There are buttons for 'Delete', 'Cancel', 'Redeploy', 'Download', and 'Refresh'. The bottom page is the 'internfilestorage | Containers' page, showing a list of containers: '\$logs' and 'intern-documents'. Both are private and available. The left sidebar of this page includes links for 'Overview', 'Activity log', 'Tags', 'Diagnose and solve problems', 'Access Control (IAM)', and 'Data migration'.

2. Backend Project Setup:

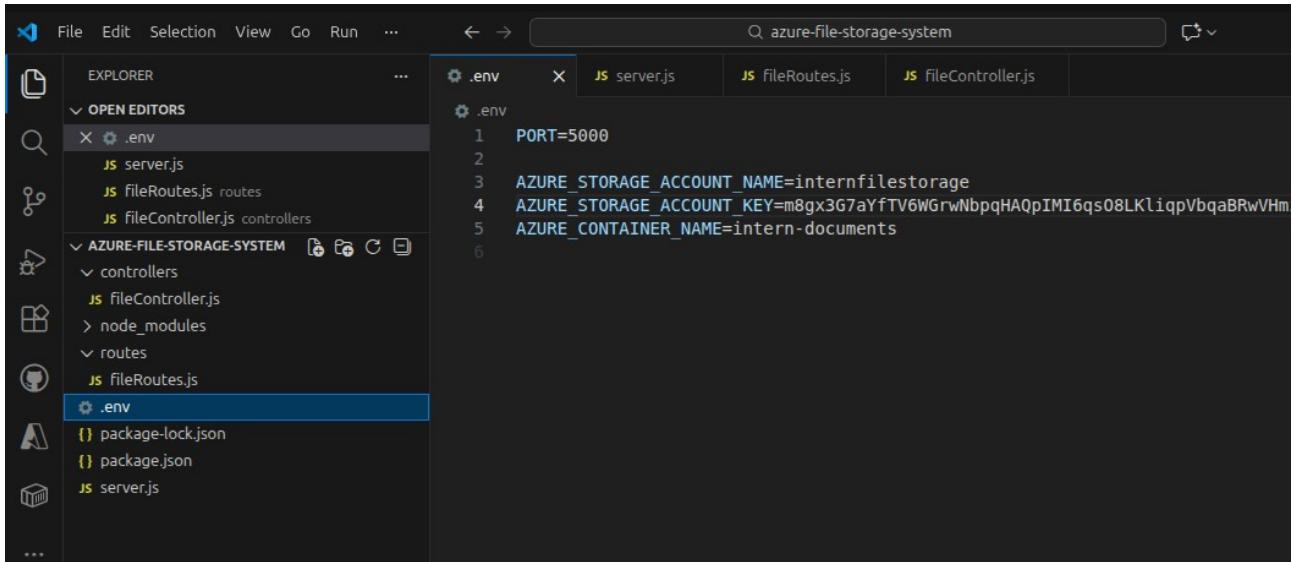
- Create Project folder
- Initialize Node.js Project

```
hamza@hamza-HP-EliteBook-840-G4:~$ mkdir azure-file-storage-system
hamza@hamza-HP-EliteBook-840-G4:~$ cd azure-file-storage-system
hamza@hamza-HP-EliteBook-840-G4:~/azure-file-storage-system$ npm init -y
Wrote to /home/hamza/azure-file-storage-system/package.json:

{
  "name": "azure-file-storage-system",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

hamza@hamza-HP-EliteBook-840-G4:~/azure-file-storage-system$
```

3. Environment Configuration:

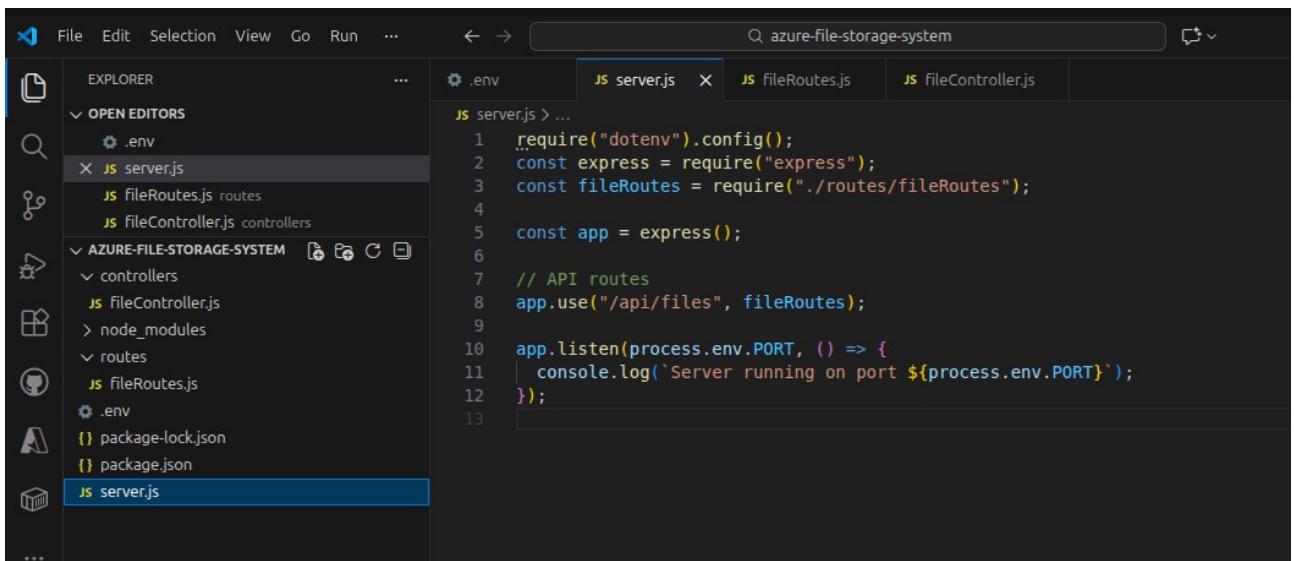


The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor tab bar at the top. The current file being edited is `.env`, which contains environment variables for a Node.js application. The code is as follows:

```
1 PORT=5000
2
3 AZURE_STORAGE_ACCOUNT_NAME=internfilestorage
4 AZURE_STORAGE_ACCOUNT_KEY=m8gx3G7aYftTV6WGrwNbpqHAOpIMI6qs08LKliqpVbqaBRwVHm
5 AZURE_CONTAINER_NAME=intern-documents
```

4. Backend Coding:

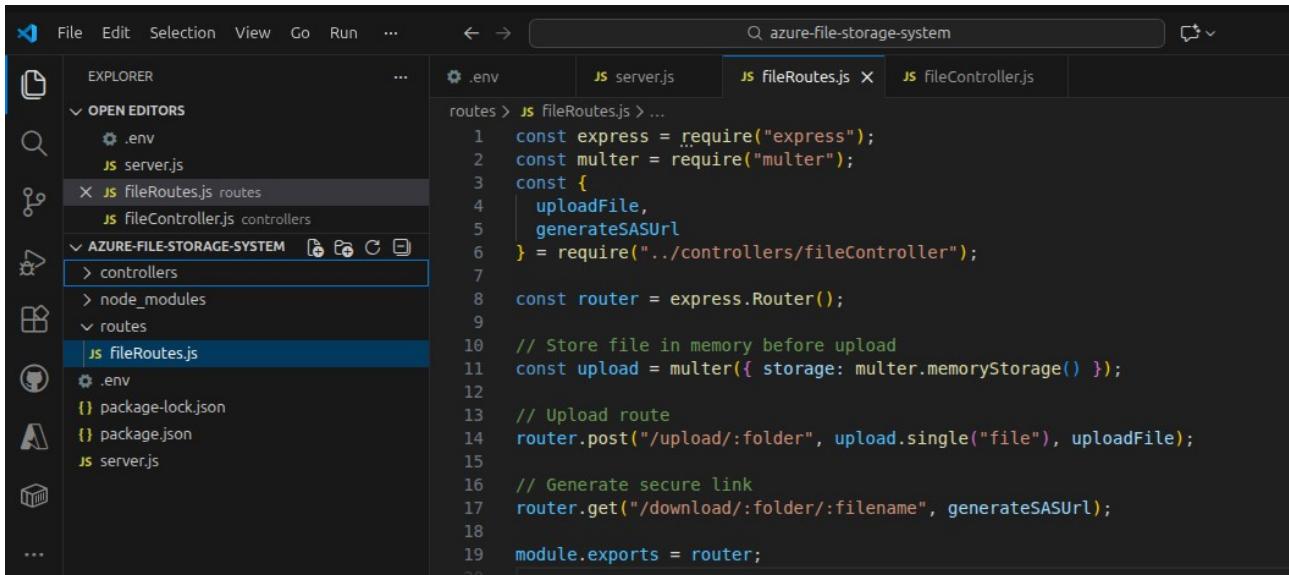
4.1 Server.js:-



The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor tab bar at the top. The current file being edited is `server.js`, which uses the `dotenv` package to load environment variables and sets up an Express API. The code is as follows:

```
1 require("dotenv").config();
2 const express = require("express");
3 const fileRoutes = require("./routes/fileRoutes");
4
5 const app = express();
6
7 // API routes
8 app.use("/api/files", fileRoutes);
9
10 app.listen(process.env.PORT, () => {
11   console.log(`Server running on port ${process.env.PORT}`);
12});
```

4.2 routes/fileRoutes.js :-



The screenshot shows the VS Code interface with the fileRoutes.js file open in the editor. The code implements an Express Router for file uploads and downloads, utilizing the multer middleware for handling file uploads.

```
const express = require("express");
const multer = require("multer");
const {
  uploadFile,
  generateSASUrl
} = require("../controllers/fileController");

const router = express.Router();

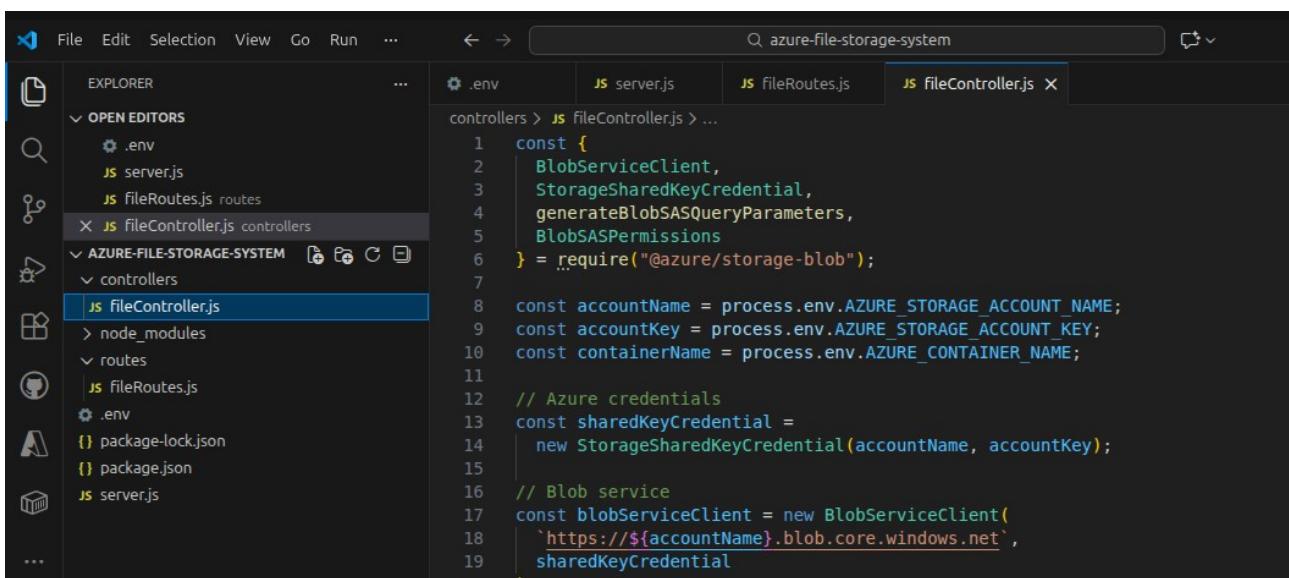
// Store file in memory before upload
const upload = multer({ storage: multer.memoryStorage() });

// Upload route
router.post("/upload/:folder", upload.single("file"), uploadFile);

// Generate secure link
router.get("/download/:folder/:filename", generateSASUrl);

module.exports = router;
```

4.3 controllers/fileController.js :-



The screenshot shows the VS Code interface with the fileController.js file open in the editor. The code initializes a BlobServiceClient using Azure storage account credentials and creates a BlobServiceClient instance for interacting with the blob storage.

```
const {
  BlobServiceClient,
  StorageSharedKeyCredential,
  generateBlobSASQueryParameters,
  BlobSASPermissions
} = require("@azure/storage-blob");

const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
const accountKey = process.env.AZURE_STORAGE_ACCOUNT_KEY;
const containerName = process.env.AZURE_CONTAINER_NAME;

// Azure credentials
const sharedKeyCredential =
  new StorageSharedKeyCredential(accountName, accountKey);

// Blob service
const blobServiceClient = new BlobServiceClient(
  `https://${accountName}.blob.core.windows.net`,
  sharedKeyCredential
);
```

5: Run & Test :

```
1745 node server.js
```

Result :-

The project was successfully implemented using Azure Blob Storage. Separate containers and folders were created for different types of internal documents, ensuring proper organization and easy retrieval. Files were uploaded securely to the cloud, and **signed URLs** were generated to allow temporary and permission-based access to specific documents. The system demonstrated reliable performance, secure data handling, and efficient file access from remote locations.

Conclusion :-

The Cloud-Based File Storage System effectively fulfills its objective of secure and organized document storage using Azure services. By leveraging Azure Blob Storage and signed URLs, the system ensures data security, scalability, and controlled access. This solution reduces the risks associated with local storage, improves accessibility, and provides a practical foundation for enterprise-level intern or document management systems. The project highlights the importance of cloud technologies in modern application development and can be extended further with role-based access control and automation features