Step 1. API code that will take input as zip file and copy it to upload folder and extract there

const express = require('express');

const jwt = require('jsonwebtoken');

const multer = require('multer');

const { exec } = require('child\_process');

const fs = require('fs');

const path = require('path');

const unzipper = require('unzipper');

const app = express();

const upload = multer({ dest: 'uploads/' });

const secretKey = 'your-secret-key'; // Use the same secret key as used in your JWT token generation

app.post('/scan', upload.single('project'), async (req, res) => {

// Extract the JWT token from the Authorization header

const token = req.headers['authorization'].split(' ')[1];

try {

// Verify the JWT token

jwt.verify(token, secretKey);

// Get the path of the uploaded zip file

const uploadedZipPath = req.file.path;

const extractedPath = `${uploadedZipPath}-extracted`;

// Unzip the project files to the extractedPath

await unzipProjectFiles(uploadedZipPath, extractedPath);

// Define the scan command

const scanCommand = `snyk test ${extractedPath} --json > ${extractedPath}/report.html`;

// Run the scan command

exec(scanCommand, (error, stdout, stderr) => {

if (error) {

console.error(`Error executing scan: ${error.message}`);

return res.status(500).send('Scan failed');

}

// Define the path of the generated report

const reportPath = path.join(extractedPath, 'report.html');

// Send the generated HTML report as the response

res.download(reportPath, 'report.html', (err) => {

if (err) {

console.error(`Error sending report: ${err.message}`);

}

// Clean up: Remove the uploaded zip file and the extracted files

fs.rmSync(uploadedZipPath, { recursive: true, force: true });

fs.rmSync(extractedPath, { recursive: true, force: true });

});

});

} catch (err) {

// Handle unauthorized access

return res.status(401).send('Unauthorized');

}

});

async function unzipProjectFiles(sourceZip, destDir) {

return new Promise((resolve, reject) => {

fs.createReadStream(sourceZip)

.pipe(unzipper.Extract({ path: destDir }))

.on('close', resolve)

.on('error', reject);

});

}

// Start the server on port 3000

app.listen(3000, () => {

console.log('Server running on port 3000');

});

**Explanation:**

1. **JWT Verification**: The JWT token is extracted from the Authorization header and verified using the jsonwebtoken library.
2. **Accessing the Uploaded File**: The multer middleware saves the uploaded file in the uploads directory, and its path is available in req.file.path.
3. **Unzipping the File**: The unzipper library is used to extract the contents of the uploaded zip file to a directory named <uploadedZipPath>-extracted.
4. **Running the Scan Command**: The exec function from the child\_process module runs the specified scan command (snyk test) in the extracted project directory and generates a report.html file.
5. **Sending the Report**: The generated report.html file is sent back to the client as a downloadable file.
6. **Cleanup**: The uploaded zip file and the extracted files are deleted after the response is sent.

This setup ensures that your API can handle the uploaded zip file, process it, and return the scan report as requested.

**Vscode extention code.**

### Using the Token in Your VS Code Extension

In your VS Code extension code, replace the placeholder token generation with the generated token. Update the sendProjectFiles function to include the JWT token in the headers.

const vscode = require('vscode');

const axios = require('axios');

const fs = require('fs');

const archiver = require('archiver');

const path = require('path');

// Add a new command to trigger the scan

function activate(context) {

let disposable = vscode.commands.registerCommand('extension.runScan', function () {

const projectPath = vscode.workspace.rootPath;

const token = 'YOUR\_GENERATED\_JWT\_TOKEN'; // Replace with the generated JWT token

vscode.window.withProgress({

location: vscode.ProgressLocation.Notification,

title: "Running Scan",

cancellable: false

}, async (progress) => {

progress.report({ increment: 0, message: "Packaging project files..." });

const zipPath = path.join(projectPath, 'project.zip');

await zipProjectFiles(projectPath, zipPath);

progress.report({ increment: 50, message: "Sending project files to server..." });

try {

const response = await sendProjectFiles(zipPath, token);

const reportPath = path.join(projectPath, 'scan-report.html');

fs.writeFileSync(reportPath, response.data);

progress.report({ increment: 100, message: "Scan completed. Report generated." });

vscode.window.showInformationMessage('Scan completed. Report generated.');

// Optionally, open the report in VS Code

const openPath = vscode.Uri.file(reportPath);

vscode.workspace.openTextDocument(openPath).then(doc => {

vscode.window.showTextDocument(doc);

});

} catch (error) {

vscode.window.showErrorMessage('Scan failed: ' + error.message);

}

});

});

context.subscriptions.push(disposable);

}

async function zipProjectFiles(sourceDir, zipPath) {

return new Promise((resolve, reject) => {

const output = fs.createWriteStream(zipPath);

const archive = archiver('zip', { zlib: { level: 9 } });

output.on('close', resolve);

archive.on('error', reject);

archive.pipe(output);

// Add files to the archive, excluding specified folders

archive.glob('\*\*/\*', {

cwd: sourceDir,

ignore: ['node\_modules/\*\*', 'target/\*\*', 'build/\*\*', '\_\_pycache\_\_/\*\*']

});

archive.finalize();

});

}

async function sendProjectFiles(zipPath, token) {

const formData = new FormData();

formData.append('project', fs.createReadStream(zipPath));

return axios.post('https://your-api-endpoint/scan', formData, {

headers: {

'Authorization': `Bearer ${token}`,

...formData.getHeaders()

},

responseType: 'arraybuffer'

});

}

exports.activate = activate;

function deactivate() {}

module.exports = {

activate,

deactivate

};

**Generate JWT Token using Node.js**:

Create a script to generate a JWT token. Save it as generateToken.js:

*const jwt = require('jsonwebtoken');*

*const secretKey = 'your-secret-key'; // Replace this with your actual secret key*

*// Payload data for the token*

*const payload = {*

*user: 'your-username', // You can add more data here as needed*

*role: 'developer' // Example additional data*

*};*

*// Options for the token*

*const options = {*

*expiresIn: '1h' // Token expiry time*

*};*

*// Generate the token*

*const token = jwt.sign(payload, secretKey, options);*

*console.log('Generated JWT Token:', token);*

**Run the Script**:

Run the script to generate a token:

node generateToken.js

### Testing the API

To test the API, you can use tools like Postman or curl. Here's an example using curl:

curl -X POST http://localhost:3000/scan \

-H "Authorization: Bearer YOUR\_JWT\_TOKEN" \

-F "project=@path/to/your/project.zip" \

-o report.html

Replace YOUR\_JWT\_TOKEN with a valid JWT token and path/to/your/project.zip with the path to your project zip file.

This setup should allow you to receive the project files, run the scan, and send back the HTML report.

**Running the API**

* The API will run on http://localhost:3000.
* You can test it by sending a POST request to http://localhost:3000/scan with a project file and a valid JWT token in the Authorization header.