import {

  ForbiddenException,

  Injectable,

  InternalServerErrorException,

} from '@nestjs/common';

import { SCOPES, AUTH } from 'src/config/google';

import { google } from 'googleapis';

import { UserService } from '../user/user.service';

import { AuthService } from '../auth/auth.service';

import { TokenType } from 'src/core/enum/token\_type.enum';

import { GHelperService } from 'src/core/services/g-drive/g-drive.service';

import \* as cheerio from 'cheerio';

@Injectable()

export class GDriveService {

  private scope = SCOPES;

  private auth = AUTH;

  private oAuth2Client: any;

  constructor(

    private userService: UserService,

    private authService: AuthService,

    private gHelperService: GHelperService,

  ) {

    const { client\_id, client\_secret, redirect\_uris } = this.auth;

    this.oAuth2Client = new google.auth.OAuth2({

      clientId: client\_id,

      clientSecret: client\_secret,

      redirectUri: redirect\_uris[0],

    });

  }

  async generateAuthUrl() {

    const authUrl = this.oAuth2Client.generateAuthUrl({

      access\_type: 'offline',

      scope: this.scope,

    });

    return authUrl;

  }

  async generateToken(code: string, userId: string) {

    try {

      const { tokens } = await this.oAuth2Client.getToken(code);

      const result = await this.userService.setDriveTokens(tokens, userId);

      return result;

    } catch (errors) {

      console.error(errors);

      throw new InternalServerErrorException('internal server error');

    }

  }

  async revokeTokennow(userId: string) {

    try {

      const user = await this.userService.getDriveTokens(userId);

      this.oAuth2Client.setCredentials({ access\_token: user.refresh\_token });

      this.oAuth2Client

        .revokeCredentials()

        // eslint-disable-next-line @typescript-eslint/no-unused-vars

        .then(async (response) => {

          return await this.updateStatus(userId, false);

        })

        .catch((error) => {

          console.error(error);

        });

    } catch (error) {

      console.error(error);

    }

  }

  async updateStatus(userId: string, status: boolean) {

    return await this.userService.setDriveConnectionStatus(userId, status);

  }

  async getFiles(userId: string) {

    try {

      const token = await this.userService.getDriveTokens(userId);

      this.oAuth2Client.setCredentials(token);

      const drive = google.drive({ version: 'v3', auth: this.oAuth2Client });

      const response = await drive.files.list({

        pageSize: 50,

        fields:

          'nextPageToken, files(webContentLink, webViewLink, thumbnailLink, id, name, mimeType)',

      });

      const files = response.data.files;

      return files;

    } catch (error) {

      console.error('Error listing files:', error.message);

      return 'could not get files';

    }

  }

  async checkGetFiles(userId: string) {

    try {

      const token = await this.userService.getDriveTokens(userId);

      this.oAuth2Client.setCredentials(token);

      const drive = google.drive({ version: 'v3', auth: this.oAuth2Client });

      // eslint-disable-next-line @typescript-eslint/no-unused-vars

      const response = await drive.files.list({

        pageSize: 50,

        fields:

          'nextPageToken, files(webContentLink, webViewLink, thumbnailLink, id, name, mimeType)',

      });

      // if (response.data.files) {

      //   return true;

      // } else {

      //   return false;

      // }

      return true;

    } catch (error) {

      console.error('Error listing files:', error.message);

      return false;

    }

  }

  async getFile(content: string) {

    try {

      const $data = cheerio.load(content);

      const isSnippetPresent = $data('body').text();

      return isSnippetPresent;

    } catch (err) {

      console.error('Error getting file:', err);

      throw new ForbiddenException('internal server error');

    }

  }

  async doGoogleLogin(data) {

    let finalResult: any;

    const userData = await this.userService.findByEmail(data.email);

    if (!userData) {

      const formatted = {

        provider: 'google',

        email: data.email,

        firstName: data.first\_name,

        lastName: data.last\_name,

        user\_image: data.provider\_pic,

        g\_access\_token: data.token,

      };

      finalResult = await this.userService.createNoPassword(formatted);

    } else {

      // if (userData.provider != 'google')

      //   throw new ForbiddenException('login type error');

      const newData = await this.userService.update(userData.id, {

        g\_access\_token: data.token,

      });

      finalResult = newData;

    }

    const payload = {

      id: finalResult.id,

      email: finalResult.email,

      role: finalResult.user\_role,

    };

    const token = this.authService.generateToken(payload);

    await this.userService.changeLoginStatus(finalResult.id, true);

    await this.userService.setLastLogin(finalResult.id, new Date());

    await this.authService.addToken(token, TokenType.ACCESS, finalResult.id);

    const sessionData = await this.authService.getUserDetails(finalResult.id);

    return {

      auth\_token: token,

      timeSaved: finalResult.total\_time\_saved,

      firstName: finalResult.first\_name,

      lastName: finalResult.last\_name,

      sessionData,

      welcomeCount: finalResult.welcomeCounter,

    };

  }

}