**ДОДАТОК Б**

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

\_\_\_\_\_\_\_\_ Людмила ЗУБИК

“28” грудня 2023 р.

**Веб-застосунок для автоматизованого формування \*.pdf файлів**

**Текст програми**

КПІ.ІП-1512.045440.05.13

“ПОГОДЖЕНО”

Керівник роботи:

Зубик Л.В.

|  |  |
| --- | --- |
| Консультант: | Виконавець: |
| Головченко М.М. | Кондрацька С.Л. |

Київ – 2023

**Файл app.py**

import json  
import os  
from datetime import datetime  
from io import BytesIO  
import requests  
from flask\_cors import CORS  
from flask import Flask, render\_template, request, send\_file, flash, redirect, url\_for, jsonify, make\_response  
from flask\_sqlalchemy import SQLAlchemy  
from flask\_login import UserMixin, login\_user, LoginManager, login\_required, logout\_user, current\_user  
from flask\_wtf import FlaskForm  
from wtforms import StringField, PasswordField, SubmitField  
from wtforms.validators import InputRequired, Length, ValidationError  
from flask\_bcrypt import Bcrypt  
from flask\_migrate import Migrate  
from werkzeug.utils import secure\_filename  
  
from backend\_func.functions import \*  
app = Flask(\_\_name\_\_)  
app.secret\_key = os.environ.get("SECRET\_KEY") or os.urandom(24)  
CORS(app)  
  
  
  
app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///database.db'  
app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False  
app.app\_context().push()  
db = SQLAlchemy(app)  
bcrypt = Bcrypt(app)  
migrate = Migrate(app, db)  
  
  
os.environ['OAUTHLIB\_INSECURE\_TRANSPORT'] = '1'  
*#---------------------------------GOOGLE---------------------------------*from oauthlib.oauth2 import WebApplicationClient  
GOOGLE\_CLIENT\_ID = os.environ.get("GOOGLE\_CLIENT\_ID", None)  
GOOGLE\_CLIENT\_SECRET = os.environ.get("GOOGLE\_CLIENT\_SECRET", None)  
GOOGLE\_DISCOVERY\_URL = os.environ.get("GOOGLE\_DISCOVERY\_URL", None)  
client = WebApplicationClient(GOOGLE\_CLIENT\_ID)  
def get\_google\_provider\_cfg():  
 return requests.get(GOOGLE\_DISCOVERY\_URL).json()  
  
@app.route("/google-login")  
def google\_login():  
 google\_provider\_cfg = get\_google\_provider\_cfg()  
 authorization\_endpoint = google\_provider\_cfg["authorization\_endpoint"]  
  
 request\_uri = client.prepare\_request\_uri(  
 authorization\_endpoint,  
 redirect\_uri=request.base\_url + "/callback",  
 scope=["openid", "email", "profile"],  
 )  
 return redirect(request\_uri)  
  
@app.route("/google-login/callback")  
def callback():  
 code = request.args.get("code")  
 google\_provider\_cfg = get\_google\_provider\_cfg()  
 token\_endpoint = google\_provider\_cfg["token\_endpoint"]  
 token\_url, headers, body = client.prepare\_token\_request(  
 token\_endpoint,  
 authorization\_response=request.url,  
 redirect\_url=request.base\_url,  
 code=code  
 )  
 token\_response = requests.post(token\_url, headers=headers, data=body, auth=(GOOGLE\_CLIENT\_ID, GOOGLE\_CLIENT\_SECRET))  
 client.parse\_request\_body\_response(json.dumps(token\_response.json()))  
  
 userinfo\_endpoint =google\_provider\_cfg["userinfo\_endpoint"]  
 uri,headers,body = client.add\_token(userinfo\_endpoint)  
 userinfo\_response = requests.get(uri, headers=headers, data=body)  
 if userinfo\_response.json().get("email\_verified"):  
 unique\_id = userinfo\_response.json()["sub"]  
 users\_email = userinfo\_response.json()["email"]  
  
 user = User.query.filter\_by(username=users\_email).first()  
 if user:  
 if bcrypt.check\_password\_hash(user.password, unique\_id):  
 login\_user(user)  
 return redirect(url\_for('profile'))  
 else:  
 hashed\_password = bcrypt.generate\_password\_hash(unique\_id)  
 new\_user = User(username=users\_email, password=hashed\_password)  
 db.session.add(new\_user)  
 db.session.commit()  
 return redirect(url\_for('profile'))  
 return redirect(url\_for('index'))  
  
*#---------------------------------FILES---------------------------------*class File(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 filename = db.Column(db.String(100))  
 data = db.Column(db.LargeBinary)  
 created\_at = db.Column(db.DateTime, default=datetime.utcnow)  
 user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)  
 file\_size = db.Column(db.Float)  
  
  
*#---------------------------------USER---------------------------------*login\_manager = LoginManager()  
login\_manager.init\_app(app)  
login\_manager.login\_view = 'login'  
  
  
@app.before\_request  
def before\_request():  
 if current\_user.is\_authenticated:  
 app.jinja\_env.globals['user\_authenticated'] = True  
 else:  
 app.jinja\_env.globals['user\_authenticated'] = False  
  
@login\_manager.user\_loader  
def load\_user(user\_id):  
 return User.query.get(int(user\_id))  
  
class User(db.Model, UserMixin):  
 id = db.Column(db.Integer, primary\_key=True)  
 username = db.Column(db.String(100), nullable=False, unique=True)  
 password = db.Column(db.String(80), nullable=False)  
 files = db.relationship('File', backref='user', lazy=True)  
  
 def is\_authenticated(self):  
 return True if self.is\_authenticated else False  
  
class RegisterForm(FlaskForm):  
 username = StringField(validators=[  
 InputRequired(), Length(min=4, max=20)], render\_kw={"placeholder": "Username"})  
  
 password = PasswordField(validators=[  
 InputRequired(), Length(min=8, max=20)], render\_kw={"placeholder": "Password"})  
  
 submit = SubmitField('Sing Up')  
  
 def validate\_username(self, username):  
 existing\_user\_username = User.query.filter\_by(  
 username=username.data).first()  
 if existing\_user\_username:  
 raise ValidationError(  
 'That username already exists. Please choose a different one.')  
  
class LoginForm(FlaskForm):  
 username = StringField(validators=[  
 InputRequired(), Length(min=4, max=20)], render\_kw={"placeholder": "Username"})  
  
 password = PasswordField(validators=[  
 InputRequired(), Length(min=8, max=20)], render\_kw={"placeholder": "Password"})  
  
 submit = SubmitField('Log In')  
  
@app.route('/login', methods=['GET', 'POST'])  
def login():  
 form = LoginForm()  
 if form.validate\_on\_submit():  
 user = User.query.filter\_by(username=form.username.data).first()  
 if user:  
 if bcrypt.check\_password\_hash(user.password, form.password.data):  
 login\_user(user)  
 return redirect(url\_for('profile'))  
 return render\_template('login.html', form=form)  
  
@ app.route('/register', methods=['GET', 'POST'])  
def register():  
 form = RegisterForm()  
  
 if form.validate\_on\_submit():  
 hashed\_password = bcrypt.generate\_password\_hash(form.password.data)  
 new\_user = User(username=form.username.data, password=hashed\_password)  
 db.session.add(new\_user)  
 db.session.commit()  
 return redirect(url\_for('login'))  
  
 return render\_template('register.html', form=form)  
  
@app.route('/logout', methods=['GET', 'POST'])  
@login\_required  
def logout():  
 logout\_user()  
 return redirect(url\_for('login'))  
  
*#---------------------------------PAGES---------------------------------*@app.route('/profile', methods=['GET', 'POST'])  
@login\_required  
def profile():  
 user\_files = File.query.filter\_by(user=current\_user).order\_by(File.created\_at.desc()).all()  
  
 for file in user\_files:  
 file\_path = os.path.join(RESULT\_FOLDER, file.filename)  
 with open(file\_path, 'wb') as f:  
 f.write(file.data)  
 return render\_template('profile.html', user\_files=user\_files)  
  
@app.route("/")  
def index():  
 clean\_folder(RESULT\_FOLDER)  
 clean\_folder(UPLOAD\_FOLDER)  
  
 return render\_template("index.html")  
  
@app.route("/merge\_pdf")  
def merge\_pdf():  
 clean\_folder(RESULT\_FOLDER)  
 clean\_folder(UPLOAD\_FOLDER)  
 return render\_template("merge.html")  
  
@app.route("/split\_pdf")  
def split\_pdf():  
 clean\_folder(RESULT\_FOLDER)  
 clean\_folder(UPLOAD\_FOLDER)  
 return render\_template("split.html")  
  
@app.route("/convert\_to\_pdf")  
def convert\_to\_pdf():  
 clean\_folder(RESULT\_FOLDER)  
 clean\_folder(UPLOAD\_FOLDER)  
 return render\_template("convert.html")  
  
@app.route("/convert\_web\_to\_pdf")  
def convert\_web\_to\_pdf():  
 clean\_folder(RESULT\_FOLDER)  
 clean\_folder(UPLOAD\_FOLDER)  
 return render\_template("web\_convert.html")  
  
@app.route("/result\_pdf/<filename>")  
def result\_pdf(filename):  
 file\_path = os.path.join(RESULT\_FOLDER, f"{filename}.pdf")  
 pdf\_read = PdfReader(file\_path)  
 pages\_count = pdf\_read.pages  
 file\_size = round(os.stat(file\_path).st\_size / 1024 / 1024, 2)  
 if current\_user.is\_authenticated:  
 with open(file\_path, 'rb') as file:  
 file\_data = file.read()  
 new\_file = File(filename=f"{filename}.pdf", data=file\_data, user=current\_user, file\_size = file\_size)  
 db.session.add(new\_file)  
 db.session.commit()  
 return render\_template('result\_pdf.html',  
 filename=filename,  
 pages\_count=len(pages\_count),  
 filesize=file\_size)  
  
@app.route("/result\_pdfs/<zipName>")  
def result\_pdfs(zipName):  
 infos = read\_folder(RESULT\_FOLDER)  
 files = [file\_info for file\_info in infos if file\_info["file\_extension"] == ".pdf"]  
 if current\_user.is\_authenticated:  
 for file in files:  
 new\_file = File(filename=file["file\_name"], data=file["file\_data"], user=current\_user, file\_size = file["file\_size"])  
 db.session.add(new\_file)  
 db.session.commit()  
  
  
 make\_zip(zipName)  
 zip\_file\_size = os.path.getsize(f"backend\_func/result\_zip/{zipName}.zip")  
 return render\_template("result\_pdfs.html",  
 filename=zipName,  
 filesize=round(zip\_file\_size / 1024 / 1024, 2),  
 files=files)  
  
*#---------------------------------FUNCTIONS---------------------------------*@app.route("/upload", methods = ['GET', 'POST'])  
def uploader():  
 if request.method == 'POST':  
 files = request.files.getlist('files[]')  
 for file in files:  
 if file.filename == '':  
 print("there is no file")  
 if file and allowed\_file(file.filename):  
 file\_converter(file)  
 return make\_response(jsonify({  
 "message": "success"  
 }), 200)  
  
@app.route("/merge", methods = ['GET', 'POST'])  
def merger():  
 if request.method == 'POST':  
 files = request.files.getlist('files[]')  
 merger\_obj = PdfWriter()  
 for file in files:  
 if '.' in file.filename and file.filename.rsplit('.', 1)[-1].lower()=='pdf':  
 merger\_obj.append(file)  
 try:  
 merger\_obj.write(os.path.join(RESULT\_FOLDER, "mergedPdf.pdf"))  
 merger\_obj.close()  
 return make\_response(jsonify({  
 "message": "success"  
 }), 200)  
 except Exception:  
 return make\_response(jsonify({  
 "message": "error"  
 }), 400)  
  
@app.route("/split", methods=['POST'])  
def divider():  
 if request.method == 'POST':  
 try:  
 files = request.files.getlist('files[]')  
 for file in files:  
 if '.' in file.filename and file.filename.rsplit('.', 1)[-1].lower() == 'pdf':  
 input\_path = os.path.join(UPLOAD\_FOLDER, f'{file.filename}')  
 file.save(input\_path)  
 pdf = PdfReader(input\_path)  
 for page in range(len(pdf.pages)):  
 pdf\_writer = PdfWriter()  
 pdf\_writer.add\_page(pdf.pages[page])  
  
 output\_filename = os.path.join(RESULT\_FOLDER, '{}\_page\_{}.pdf'.format(file.filename.rsplit('.', 1)[0], page + 1))  
  
 with open(output\_filename, 'wb') as out:  
 pdf\_writer.write(out)  
 return make\_response(jsonify({  
 "message": "success"  
 }), 200)  
 except Exception as e:  
 print(e)  
 return make\_response(jsonify({  
 "message": "error"  
 }), 500)  
  
@app.route('/web\_convert', methods=['GET', 'POST'])  
def web\_converter():  
 if request.method == 'POST':  
 data = request.get\_json()  
 link = data.get('link')  
  
 try:  
 converter.convert(link, os.path.join(RESULT\_FOLDER, 'convertedPage.pdf'))  
 return make\_response(jsonify({  
 "message": "success"  
 }), 200)  
 except Exception:  
 return make\_response(jsonify({  
 "message": "error"  
 }), 400)  
  
@app.route("/result\_pdf/show\_file/<filename>")  
@app.route("/result\_pdfs/show\_file/<filename>")  
@app.route("/show\_file/<filename>")  
def show\_file(filename):  
 return send\_file(os.path.join(RESULT\_FOLDER, f"{filename}.pdf"), as\_attachment=False)  
  
@app.route("/download", methods=['GET', 'POST'])  
def download():  
 if request.method == 'POST':  
 data = request.get\_json()  
 filename = data.get('filename')  
 filetype= data.get('filetype')  
 if filetype==".pdf": file\_path = f"{RESULT\_FOLDER}/{filename}{filetype}"  
 else: file\_path = f"backend\_func/result\_zip/{filename}{filetype}"  
 download\_name = secure\_filename(data.get('newfilename', filename))  
  
 return send\_file(file\_path, as\_attachment=True, download\_name=f"{download\_name}{filetype}")  
  
@app.route('/DBdownload', methods=['GET', 'POST'])  
def download\_from\_db():  
 if request.method == 'POST':  
 data = request.get\_json()  
 file\_id = data.get('file\_id')  
 filetype = data.get('filetype')  
 download\_name =secure\_filename(data.get('newfilename', data.get('filename')))  
  
 file = File.query.get(file\_id)  
 if file:  
 return send\_file(BytesIO(file.data), download\_name=f"{download\_name}{filetype}", as\_attachment=True)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

**Файл functions.py**

import textwrap  
import PIL.Image  
from pyhtml2pdf import converter  
  
from fpdf import FPDF  
  
import shutil  
  
from PyPDF2 import PdfWriter, PdfReader, PdfFileWriter, PdfFileReader  
import pandas as pd  
  
  
from backend\_func.cloudmersive\_converter import \*  
  
  
ALLOWED\_EXTENSIONS = {'txt', 'png', 'jpg', 'jpeg', 'docx', 'doc', 'csv', 'xlsx'}  
  
def allowed\_file(filename):  
 return '.' in filename and \  
 filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS  
  
  
def clean\_folder(folder\_path: str):  
 for entry in os.scandir(folder\_path):  
 file\_path = os.path.join(folder\_path, entry.name)  
 try:  
 if entry.is\_file() or entry.is\_symlink():  
 os.unlink(file\_path)  
 elif entry.is\_dir():  
 shutil.rmtree(file\_path)  
 except Exception as e:  
 print('Failed to delete %s. Reason: %s' % (file\_path, e))  
  
def read\_folder(folder\_path:str):  
 files\_info = []  
 for entry in os.scandir(folder\_path):  
 file\_path = os.path.join(folder\_path, entry.name)  
 with open(file\_path, 'rb') as file:  
 file\_data = file.read()  
 try:  
 pdf\_read = PdfReader(file\_path)  
 pages\_count = len(pdf\_read.pages)  
 except Exception as e:  
 pages\_count = "Error: " + str(e)  
  
 file\_stats = os.stat(file\_path)  
 file\_size = round(file\_stats.st\_size / 1024 / 1024, 2)  
 \_, file\_extension = os.path.splitext(entry.name)  
  
 file\_info = {  
 "file\_name":entry.name,  
 "path": file\_path,  
 "pages\_count": pages\_count,  
 "file\_size": file\_size,  
 "name": '.'.join(entry.name.split('.')[:-1]).lower(),  
 "file\_data": file\_data,  
 "file\_extension": file\_extension  
 }  
 files\_info.append(file\_info)  
 return files\_info  
  
  
def file\_converter(file):  
 file\_extension = file.filename.rsplit('.', 1)[-1].lower()  
 file\_name = file.filename.rsplit('.', 1)[0].lower()  
  
 input\_path = os.path.join(UPLOAD\_FOLDER, f'{file\_name}.{file\_extension}')  
 output\_path = os.path.join(RESULT\_FOLDER, f'{file\_name}.pdf')  
 match file\_extension:  
 case 'png' | 'jpg' | 'jpeg':  
 image = PIL.Image.open(file)  
 im = image.convert('RGB')  
 im.save(output\_path)  
 case 'docx' | 'doc':  
 file.save(input\_path)  
 cloudmersive\_convert(input\_path, output\_path)  
 case 'csv':  
 file.save(input\_path)  
 CSV = pd.read\_csv(input\_path)  
 path\_to\_html = os.path.join(UPLOAD\_FOLDER, f'{file\_name}.html')  
 CSV.to\_html(path\_to\_html)  
 converter.convert(f'file:///{os.path.abspath(path\_to\_html)}', output\_path)  
 case 'xlsx':  
 file.save(input\_path)  
 wb = pd.read\_excel(input\_path)  
 path\_to\_html = os.path.join(UPLOAD\_FOLDER, f'{file\_name}.html')  
 wb.to\_html(path\_to\_html)  
 converter.convert(f'file:///{os.path.abspath(path\_to\_html)}', output\_path)  
 case 'txt':  
 file.save(input\_path)  
 with open(input\_path, "r") as f:  
 file\_content = f.read()  
  
 a4\_width\_mm = 210  
 pt\_to\_mm = 0.35  
 fontsize\_pt = 14  
 fontsize\_mm = fontsize\_pt \* pt\_to\_mm  
 margin\_bottom\_mm = 10  
 character\_width\_mm = 7 \* pt\_to\_mm  
 width\_text = a4\_width\_mm / character\_width\_mm  
  
 pdf = FPDF(orientation='P', unit='mm', format='A4')  
 pdf.add\_font("NotoSansDisplay", style="", fname="static/fonts/NotoSansDisplay-VariableFont\_wdth,wght.ttf", uni=True)  
  
 pdf.set\_auto\_page\_break(True, margin=margin\_bottom\_mm)  
 pdf.add\_page()  
 pdf.set\_font(family='NotoSansDisplay', size=fontsize\_pt)  
 splitted = file\_content.split('\n')  
  
 for line in splitted:  
 lines = textwrap.wrap(line, width\_text)  
  
 if len(lines) == 0:  
 pdf.ln()  
  
 for wrap in lines:  
 pdf.cell(0, fontsize\_mm, wrap, ln=1)  
 pdf.output(output\_path)  
  
 case \_:  
 *# default case* pass  
def make\_zip(zipName):  
 shutil.make\_archive(os.path.join("backend\_func/result\_zip", f'{zipName}'), 'zip', RESULT\_FOLDER)

**Файл cloudmersive\_converter.py**

from \_\_future\_\_ import print\_function  
  
import os  
import cloudmersive\_convert\_api\_client  
from cloudmersive\_convert\_api\_client.rest import ApiException  
from pprint import pprint  
RESULT\_FOLDER='backend\_func/resulted\_files'  
UPLOAD\_FOLDER='backend\_func/uploaded\_files'  
  
  
configuration = cloudmersive\_convert\_api\_client.Configuration()  
configuration.api\_key['Apikey'] = os.environ.get("APIKEY")  
  
*#configuration.api\_key\_prefix['Apikey'] = 'Bearer'*api\_instance = cloudmersive\_convert\_api\_client.ConvertDocumentApi(cloudmersive\_convert\_api\_client.ApiClient(configuration))  
  
def cloudmersive\_convert(input\_path, output\_path):  
 try:  
 api\_response = api\_instance.convert\_document\_autodetect\_to\_pdf(input\_path)  
 *#pprint(api\_response)* with open(output\_path, 'wb') as output\_file:  
 output\_file.write(api\_response)  
 except ApiException as e:  
 print("Exception when calling ConvertDocumentApi->convert\_document\_autodetect\_to\_pdf: %s\n" % e)