ДОДАТОК А

Вихідний код програми

Роутер route

```
from flask import render_template, url_for, flash, redirect, request
from app import app, db
from app.recommender import runRecommender
from app.forms import RegistrationForm, LoginForm, updateEmail, RatingForm
from flask_login import login_user, current_user, logout_user, login_required
from app.models import User
import json
import os


@app.route('/')
@app.route('/index', methods=['GET'])
def index():
    return render_template('index.html', title="BookPicker")


@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = RegistrationForm()
    if form.validate_on_submit():
        max_id = db.session.query(db.func.max(User.user_id)).scalar()
        user = User(user_id=max_id + 1, email=form.email.data, password=form.password.data)
        db.session.add(user)
        db.session.commit()
        flash(f'Account created for {form.email.data}!Please login', 'success')
        return redirect(url_for('index'))
    return render_template('register.html', title='Register', form=form)


@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and int(form.password.data) == user.password:
```

```
            login_user(user, remember=form.remember.data)

            return redirect(url_for('home'))
        else:
            flash('Login Unsuccessful', 'danger')

    return render_template('login.html', title='Login', form=form)


@app.route('/home', methods=['GET', 'POST'])
@login_required
def home():
    print("User:::::", current_user.user_id)
    form = updateEmail()
    if request.method == 'POST':
        if form.validate_on_submit():
            current_user.email = form.email.data
            db.session.commit()
            flash('Your account has been updated!', 'success')
            return redirect(url_for('home'))
        elif request.method == 'GET':
            form.email.data = current_user.email
        return render_template('home.html', title='Home', form=form)
    else:
        available = False
        fileRating = os.path.join(app.static_folder, 'ratings.json')
        fileBook = os.path.join(app.static_folder, 'books.json')

        with open(fileRating) as f:
            dataRating = json.load(f)
        for i in range(len(dataRating)):
            if dataRating[i]["user_id"] == current_user.user_id:
                print(dataRating[i]["user_id"])
                available = True
        if available == False:
            suggested = [{"book_id": 0, "original_title": "No rating, data unavailable", "genre": "-"}]
            rated = [{"book_id": 0, "original_title": "No rating, data unavailable", "genre": "-"}]
        if available == True:
            predict, rated = runRecommender(fileBook, fileRating, current_user.user_id)
            suggested = json.loads(predict)
            rated = json.loads(rated)
        return render_template('home.html', title='Home', form=form, suggested=suggested, rated=rated)

    return render_template('home.html', title='Home', form=form)


@app.route('/logout')
```

```
def logout():
    logout_user()
    return redirect(url_for('index'))



@app.route('/book', methods=['GET', 'POST'])
@login_required
def book():
    book = ""
    form = RatingForm()

    if request.method == 'POST':
        print(form.book_id.data)
        if form.validate_on_submit():
            redundant = False
            filename = os.path.join(app.static_folder, 'ratings.json')

            jsonData = {"user_id": current_user.user_id, "book_id": form.book_id.data, "rating": form.rating.data}
            with open(filename) as f:
                data = json.load(f)
            for i in range(len(data)):
                if data[i]["user_id"] == current_user.user_id and data[i]["book_id"] == form.book_id.data:
                    redundant = True
            if redundant == False:
                data.append(jsonData)

                with open(filename, 'w') as f:
                    json.dump(data, f)
                    flash('Your rating has been updated!', 'success')
            else:
                flash('Rating unsuccessful, you have rated the movie', 'danger')
            return redirect(url_for('book'))
        else:
            flash('Rating Unsuccessful', 'danger')

    if request.method == 'GET':
        filename = os.path.join(app.static_folder, 'books.json')
        with open(filename, encoding="utf-8") as blog_file:
            book = json.load(blog_file)
    return render_template('book.html', book=book, form=form)
```

Сервіс recommender

```
import pandas as pd
import numpy as np
```

```
from scipy.sparse.linalg import svds


def recommend_books(predictions_df, user_id, original_title, rating, num_recommendations=5):
    # Get and sort the user's predictions
    rating.columns = ["user_id", "book_id", "rating"]
    user_row_number = user_id - 1  # user_id starts at 1, not 0
    sorted_user_predictions = predictions_df.iloc[user_row_number].sort_values(ascending=False)

    # Get the user's data and merge in the movie information.
    user_data = rating[rating.user_id == (user_id)]
    user_full = (user_data.merge(original_title, how='left', left_on='book_id', right_on='book_id').
                 sort_values(['rating'], ascending=False)
                 )

    print('User {0} has already rated {1} movies.'.format(user_id, user_full.shape[0]))
    print('Recommending the highest {0} predicted ratings movies not already rated.'.format(num_recommendations))

    recommendations = (original_title[~original_title['book_id'].isin(user_full['book_id'])].
                       merge(pd.DataFrame(sorted_user_predictions).reset_index(), how='left',
                           left_on='book_id',
                           right_on='book_id').
                       rename(columns={user_row_number: 'Predictions'}).
                       sort_values('Predictions', ascending=False).
                       iloc[:num_recommendations, :-1]
                       )

    return user_full, recommendations, user_full


def runRecommender(file1, file2, user_id):
    pd.set_option('display.float_format', lambda x: '%.3f' % x)

    book_list = pd.read_json(file1)
    ratings_list = pd.read_json(file2)

    # book_data = pd.merge(ratings_list, book_list, on='book_id')
    book_list['book_id'] = book_list['book_id'].apply(pd.to_numeric)
    book_data = ratings_list.pivot(index='user_id', columns='book_id', values='rating').fillna(0)
    R = book_data.to_numpy()
    user_ratings_mean = np.mean(R, axis=1)
    R_demeaned = R - user_ratings_mean.reshape(-1, 1)
    U, sigma, Vt = svds(R_demeaned, k=50)
    sigma = np.diag(sigma)

    all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-1, 1)
    preds_df = pd.DataFrame(all_user_predicted_ratings, columns=book_data.columns)
```

```
    already_rated, predictions, rated = recommend_books(preds_df, user_id, book_list, ratings_list, 10)
    predict = predictions.to_json(orient="records")
    rated = rated.to_json(orient="records")
    return predict, rated
```

Сервіс forms

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, BooleanField, IntegerField
from wtforms.validators import DataRequired, Email, EqualTo, ValidationError, NumberRange
from flask_login import current_user

from app.models import User


class RegistrationForm(FlaskForm):
    email = StringField('Email:', validators=[DataRequired(), Email()])
    password = PasswordField('Password:', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password:', validators=[DataRequired(), EqualTo('password')])

    submit = SubmitField('Sign Up')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('That email is taken')


class LoginForm(FlaskForm):
    email = StringField('Email:', validators=[DataRequired(), Email()])
    password = PasswordField('Password:', validators=[DataRequired()])
    remember = BooleanField('Remember Me:')

    submit = SubmitField('Login')


class updateEmail(FlaskForm):
    email = StringField('Email:', validators=[DataRequired(), Email()])
    submit = SubmitField('Update')

    def validate_email(self, email):
        if email.data != current_user.email:
            user = User.query.filter_by(email=email.data).first()
            if user:
                raise ValidationError('That email is taken')
```

```
class RatingForm(FlaskForm):
    book_id = IntegerField('Book ID:', validators=[DataRequired(), NumberRange(min=1, max=9999)])

    rating = IntegerField('Rating:', validators=[DataRequired(), NumberRange(min=0, max=5)])

    def validateRating(self, rating):
        if rating.data > 5 and rating.data < 0:
            raise ValidationError('Invalid rating')

    submit = SubmitField('Submit')
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	66с.
	Інші документи	
2.	Презентаційні матеріали	-- плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Тетерук Д.В.			Розробка рекомендаційної системи «BookPicker»	Шифр групи	Код напр./спец.
Перевірив		Кулішова Н.Є.				СШІм-20-3	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					