import os

import logging

from datetime import datetime

from flask import Flask, render\_template, request, redirect, url\_for, flash

from flask\_sqlalchemy import SQLAlchemy

from flask\_login import LoginManager, UserMixin, login\_user, login\_required, logout\_user, current\_user

from werkzeug.security import generate\_password\_hash, check\_password\_hash

import pandas as pd

from apscheduler.schedulers.background import BackgroundScheduler

import requests

import xml.etree.ElementTree as ET

from werkzeug.utils import secure\_filename

# Logging ayarları

logging.basicConfig(level=logging.DEBUG)

logger = logging.getLogger(\_\_name\_\_)

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = os.urandom(24)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///parts\_system.db'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

app.config['UPLOAD\_FOLDER'] = 'static/uploads'

app.debug = True

# Upload klasörünü oluştur

if not os.path.exists(app.config['UPLOAD\_FOLDER']):

    os.makedirs(app.config['UPLOAD\_FOLDER'])

db = SQLAlchemy(app)

login\_manager = LoginManager(app)

login\_manager.login\_view = 'login'

# Döviz kuru saklama

exchange\_rates = {'EUR': 0.0}

TCMB\_URL = "https://www.tcmb.gov.tr/kurlar/today.xml"

def fetch\_exchange\_rates():

    try:

        response = requests.get(TCMB\_URL)

        if response.status\_code == 200:

            tree = ET.fromstring(response.content)

            for currency in tree.findall('Currency'):

                if currency.attrib.get('CurrencyCode') == 'EUR':

                    forex\_selling = currency.find('ForexSelling').text

                    if forex\_selling:

                        eur\_rate = float(forex\_selling.replace(',', '.'))

                        exchange\_rates['EUR'] = eur\_rate

                        logger.info(f"Updated EUR rate: {eur\_rate}")

                        break

    except Exception as e:

        logger.error(f"Error fetching exchange rates: {e}")

# Veritabanı modelleri

class User(UserMixin, db.Model):

    id = db.Column(db.Integer, primary\_key=True)

    username = db.Column(db.String(80), unique=True, nullable=False)

    password\_hash = db.Column(db.String(120), nullable=False)

    role = db.Column(db.String(20), nullable=False, default='user')

    def set\_password(self, password):

        self.password\_hash = generate\_password\_hash(password)

    def check\_password(self, password):

        return check\_password\_hash(self.password\_hash, password)

class Catalog(db.Model):

    id = db.Column(db.Integer, primary\_key=True)

    name = db.Column(db.String(100), nullable=False)

    slug = db.Column(db.String(100), unique=True, nullable=False)

    description = db.Column(db.Text)

    image\_url = db.Column(db.String(255))

    items = db.relationship('CatalogItem', backref='catalog', lazy=True)

class CatalogItem(db.Model):

    id = db.Column(db.Integer, primary\_key=True)

    catalog\_id = db.Column(db.Integer, db.ForeignKey('catalog.id'), nullable=False)

    model\_name = db.Column(db.String(100), nullable=False)

    model\_code = db.Column(db.String(50), nullable=False)

    serial\_number\_pattern = db.Column(db.String(100))

    description = db.Column(db.Text)

    parts = db.relationship('Part', backref='catalog\_item', lazy=True)

class Part(db.Model):

    id = db.Column(db.Integer, primary\_key=True)

    catalog\_item\_id = db.Column(db.Integer, db.ForeignKey('catalog\_item.id'), nullable=True)

    part\_code = db.Column(db.String(20), unique=True, nullable=False, index=True)

    name = db.Column(db.String(200), nullable=False, index=True)

    alternate\_code = db.Column(db.String(20))

    price\_eur = db.Column(db.Float, nullable=False)

    last\_updated = db.Column(db.DateTime, nullable=False)

    image\_url = db.Column(db.String(500))

    description = db.Column(db.Text)

@login\_manager.user\_loader

def load\_user(user\_id):

    return db.session.get(User, int(user\_id))

def is\_admin():

    return current\_user.is\_authenticated and current\_user.role == 'admin'

# Rotalar

@app.route('/')

def index():

    return redirect(url\_for('login'))

@app.route('/login', methods=['GET', 'POST'])

def login():

    if request.method == 'POST':

        username = request.form.get('username')

        password = request.form.get('password')

        user = User.query.filter\_by(username=username).first()

        if user and user.check\_password(password):

            login\_user(user)

            logger.info(f"Successful login for user: {username}")

            return redirect(url\_for('dashboard'))

        flash('Invalid username or password!')

    return render\_template('login.html')

@app.route('/dashboard')

@login\_required

def dashboard():

    return render\_template('dashboard.html', is\_admin=is\_admin())

@app.route('/parts')

@login\_required

def parts():

    search = request.args.get('search', '')

    parts = []

    alternate\_parts = []

    if search:

        parts = Part.query.filter(

            (Part.part\_code.like(f'%{search}%')) |

            (Part.name.like(f'%{search}%'))

        ).all()

        alternate\_code = f"{search}-C"

        alternate\_parts = Part.query.filter\_by(part\_code=alternate\_code).all()

        eur\_rate = exchange\_rates['EUR']

        parts\_with\_prices = [{

            'id': p.id,

            'part\_code': p.part\_code,

            'name': p.name,

            'price\_try': p.price\_eur \* eur\_rate,

            'selling\_price\_try': p.price\_eur \* eur\_rate \* 3,

            'last\_updated': p.last\_updated,

            'alternate\_code': p.alternate\_code

        } for p in parts]

        alternate\_parts\_with\_prices = [{

            'id': p.id,

            'part\_code': p.part\_code,

            'name': p.name,

            'price\_try': p.price\_eur \* eur\_rate,

            'selling\_price\_try': p.price\_eur \* eur\_rate \* 3,

            'last\_updated': p.last\_updated,

            'alternate\_code': p.alternate\_code

        } for p in alternate\_parts]

        return render\_template('parts.html', parts=parts\_with\_prices, alternate\_parts=alternate\_parts\_with\_prices, search=search, is\_admin=is\_admin())

    return render\_template('parts.html', parts=[], alternate\_parts=[], search=search, is\_admin=is\_admin())

@app.route('/part/<int:part\_id>')

@login\_required

def part\_detail(part\_id):

    part = Part.query.get\_or\_404(part\_id)

    eur\_rate = exchange\_rates['EUR']

    price\_try = part.price\_eur \* eur\_rate

    selling\_price\_try = price\_try \* 3

    alternate\_code = f"{part.part\_code}-C"

    muadil\_part = Part.query.filter\_by(part\_code=alternate\_code).first()

    muadil\_selling\_price\_try = muadil\_part.price\_eur \* eur\_rate \* 3 if muadil\_part else None

    return render\_template('part\_detail.html', part=part, price\_try=price\_try, selling\_price\_try=selling\_price\_try, muadil\_part=muadil\_part, muadil\_selling\_price\_try=muadil\_selling\_price\_try, is\_admin=is\_admin())

@app.route('/upload\_part\_image/<int:part\_id>', methods=['POST'])

@login\_required

def upload\_part\_image(part\_id):

    if not is\_admin():

        flash('Unauthorized!')

        return redirect(url\_for('part\_detail', part\_id=part\_id))

    part = Part.query.get\_or\_404(part\_id)

    if 'image' not in request.files:

        flash('No file selected!')

        return redirect(url\_for('part\_detail', part\_id=part\_id))

    file = request.files['image']

    if file.filename == '':

        flash('No file selected!')

        return redirect(url\_for('part\_detail', part\_id=part\_id))

    if file and allowed\_file(file.filename):

        filename = secure\_filename(file.filename)

        filepath = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

        file.save(filepath)

        part.image\_url = url\_for('static', filename=f'uploads/{filename}')

        db.session.commit()

        flash('Image uploaded successfully!')

    else:

        flash('Invalid file format!')

    return redirect(url\_for('part\_detail', part\_id=part\_id))

@app.route('/update\_part\_description/<int:part\_id>', methods=['POST'])

@login\_required

def update\_part\_description(part\_id):

    if not is\_admin():

        flash('Unauthorized!')

        return redirect(url\_for('part\_detail', part\_id=part\_id))

    part = Part.query.get\_or\_404(part\_id)

    description = request.form.get('description')

    if description:

        part.description = description

        db.session.commit()

        flash('Description updated successfully!')

    else:

        flash('Description cannot be empty!')

    return redirect(url\_for('part\_detail', part\_id=part\_id))

@app.route('/upload\_excel', methods=['POST'])

@login\_required

def upload\_excel():

    if not is\_admin():

        flash('Unauthorized!')

        return redirect(url\_for('parts'))

    if 'excel\_file' not in request.files:

        flash('No file selected!')

        return redirect(url\_for('parts'))

    file = request.files['excel\_file']

    if file.filename == '':

        flash('No file selected!')

        return redirect(url\_for('parts'))

    if file and file.filename.endswith(('.xlsx', '.xls')):

        filename = secure\_filename(file.filename)

        filepath = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

        file.save(filepath)

        try:

            df = pd.read\_excel(filepath)

            for index, row in df.iterrows():

                part\_code = str(row['Parça Kodu']).strip()

                name = str(row['Parça Adı']).strip()

                alternate\_code = str(row.get('Değişen Parça Kodu', '')).strip() if pd.notna(row.get('Değişen Parça Kodu')) else None

                price\_eur = float(row.get('Geliş Fiyatı (EUR)', 0))

                if not part\_code or not name:

                    logger.warning(f"Empty part\_code or name at row {index + 2}, skipping")

                    continue

                if price\_eur == 0:

                    flash(f"Parça kodu '{part\_code}' için EUR fiyatı bulunamadı!", 'error')

                    continue

                existing\_part = Part.query.filter\_by(part\_code=part\_code).first()

                if existing\_part:

                    existing\_part.name = name

                    existing\_part.alternate\_code = alternate\_code

                    existing\_part.price\_eur = price\_eur

                    existing\_part.last\_updated = datetime.utcnow()

                else:

                    new\_part = Part(

                        part\_code=part\_code,

                        name=name,

                        alternate\_code=alternate\_code,

                        price\_eur=price\_eur,

                        last\_updated=datetime.utcnow()

                    )

                    db.session.add(new\_part)

            db.session.commit()

            flash('Excel file uploaded and processed successfully!', 'success')

        except Exception as e:

            db.session.rollback()

            flash(f'Error processing file: {str(e)}', 'error')

            logger.error(f"Excel processing error: {str(e)}")

        finally:

            os.remove(filepath)

    else:

        flash('Invalid file type. Only .xlsx or .xls files are accepted!', 'error')

    return redirect(url\_for('parts'))

@app.route('/logout')

@login\_required

def logout():

    logout\_user()

    return redirect(url\_for('login'))

@app.route('/catalogs')

@login\_required

def catalogs():

    catalogs = Catalog.query.all()

    return render\_template('catalogs.html', catalogs=catalogs, is\_admin=is\_admin())

@app.route('/catalog/<string:catalog\_slug>')

@login\_required

def catalog\_detail(catalog\_slug):

    catalog = Catalog.query.filter\_by(slug=catalog\_slug).first\_or\_404()

    items = CatalogItem.query.filter\_by(catalog\_id=catalog.id).all()

    return render\_template('catalog\_detail.html', catalog=catalog, items=items, is\_admin=is\_admin())

@app.route('/catalog-item/<int:item\_id>/parts')

@login\_required

def catalog\_item\_parts(item\_id):

    item = CatalogItem.query.get\_or\_404(item\_id)

    parts = Part.query.filter\_by(catalog\_item\_id=item.id).all()

    eur\_rate = exchange\_rates['EUR']

    parts\_with\_prices = [{

        'id': p.id,

        'part\_code': p.part\_code,

        'name': p.name,

        'price\_try': p.price\_eur \* eur\_rate,

        'selling\_price\_try': p.price\_eur \* eur\_rate \* 3,

        'image\_url': p.image\_url

    } for p in parts]

    return render\_template('catalog\_item\_parts.html', item=item, parts=parts\_with\_prices, is\_admin=is\_admin())

@app.route('/admin/panel')

@login\_required

def admin\_panel():

    if not is\_admin():

        flash('Unauthorized!', 'error')

        return redirect(url\_for('dashboard'))

    catalogs = Catalog.query.all()

    users = User.query.all()

    return render\_template('admin\_panel.html', catalogs=catalogs, users=users, is\_admin=is\_admin())

@app.route('/admin/init-db', methods=['GET', 'POST'])

@login\_required

def admin\_init\_db():

    if not is\_admin():

        flash('Unauthorized!', 'error')

        return redirect(url\_for('dashboard'))

    if request.method == 'POST':

        try:

            if not Catalog.query.filter\_by(slug='takeuchi').first():

                takeuchi = Catalog(

                    name='Takeuchi',

                    slug='takeuchi',

                    description='Takeuchi ekskavatör parça kataloğu'

                )

                db.session.add(takeuchi)

                models = [

                    {'model\_name': 'TB108', 'model\_code': 'TB108', 'serial\_pattern': 'TB108\_%'},

                    {'model\_name': 'TB014', 'model\_code': 'TB014', 'serial\_pattern': 'TB014\_%'},

                    {'model\_name': 'TB016', 'model\_code': 'TB016', 'serial\_pattern': 'TB014\_%'},

                    {'model\_name': 'TB219', 'model\_code': 'TB219', 'serial\_pattern': 'TB219\_%'}

                ]

                for model in models:

                    if not CatalogItem.query.filter\_by(model\_code=model['model\_code']).first():

                        item = CatalogItem(

                            catalog=takeuchi,

                            model\_name=model['model\_name'],

                            model\_code=model['model\_code'],

                            serial\_number\_pattern=model['serial\_pattern'],

                            description=f"{model['model\_name']} model parçaları"

                        )

                        db.session.add(item)

                db.session.commit()

                flash('Sample catalog data added!', 'success')

            else:

                flash('Takeuchi catalog already exists!', 'warning')

        except Exception as e:

            db.session.rollback()

            flash(f'Error: {str(e)}', 'error')

        return redirect(url\_for('admin\_panel'))

    return render\_template('admin\_init\_db.html', is\_admin=is\_admin())

def allowed\_file(filename):

    return '.' in filename and filename.rsplit('.', 1)[1].lower() in {'png', 'jpg', 'jpeg', 'gif'}

if \_\_name\_\_ == '\_\_main\_\_':

    scheduler = BackgroundScheduler()

    scheduler.add\_job(func=fetch\_exchange\_rates, trigger="interval", hours=1)

    scheduler.start()

    with app.app\_context():

        # Veritabanı yoksa veya tablolar eksikse oluştur

        db.create\_all()  # Mevcut tabloları silmeden sadece eksik olanları oluşturur

        logger.info("Database tables checked/created successfully.")

        # Admin kullanıcılarını ekle (eğer yoksa)

        if not User.query.filter\_by(role='admin').first():

            admins = [

                {'username': 'admin1', 'password': 'admin123', 'role': 'admin'},

                {'username': 'admin2', 'password': 'admin123', 'role': 'admin'},

                {'username': 'admin3', 'password': 'admin123', 'role': 'admin'},

                {'username': 'admin4', 'password': 'admin123', 'role': 'admin'},

            ]

            for admin in admins:

                if not User.query.filter\_by(username=admin['username']).first():

                    user = User(username=admin['username'], role=admin['role'])

                    user.set\_password(admin['password'])

                    db.session.add(user)

            db.session.commit()

            logger.info("Admin users created.")

    fetch\_exchange\_rates()

    app.run(host='0.0.0.0', port=5000)