vulnedu.py

```
from flask import Flask, render_template, request, url_for, redirect, jsonify
from services.fetch_CVE import get_all_cves
from services.nvd_api import get_cve_detail
from services.cwe_data import get_cwe_dict, get_single_cwe, warm_cwe_cache
from services.cwe_map import CWE_TITLES, cwe_title
from collections import Counter, defaultdict
import math
import random
from datetime import datetime, timedelta, timezone
from calendar import monthrange
import os
from threading import Thread, Lock

app = Flask(__name__)

timeline_cache = {
    'data': None,
    'last_updated': None,
    'lock': Lock()
}

TIMELINE_CACHE_HOURS = 6
_warmed_up = False

def get_cached_timeline_data():
    global timeline_cache
    with timeline_cache['lock']:
        now = datetime.now(timezone.utc)
        if (timeline_cache['data'] is not None and
            timeline_cache['last_updated'] is not None and
            (now - timeline_cache['last_updated']).total_seconds() <
                TIMELINE_CACHE_HOURS * 3600):
            return timeline_cache['data']
        timeline_data = generate_timeline_data()
        timeline_cache['data'] = timeline_data
        timeline_cache['last_updated'] = now
        return timeline_data

def generate_timeline_data():
    now = datetime.now(timezone.utc)
    months = []
```

```
    base_month = now.replace(day=1)
    for i in reversed(range(36)):
        dt = (base_month - timedelta(days=31 * i)).replace(day=1)
        months.append((dt.year, dt.month))
    month_labels = [f"{y}-{m:02d}" for y, m in months]
    month_counts = {k: 0 for k in month_labels}
    recent_cves = get_all_cves(days=30)
    for cve in recent_cves:
        published_str = cve.get('Published', '')
        if published_str and len(published_str) >= 7:
            month_key = published_str[:7]
            if month_key in month_counts:
                month_counts[month_key] += 1
    current_year = now.year
    current_month = now.month
    for (y, m), label in zip(months, month_labels):
        if month_counts[label] == 0:
            if y == current_year and m >= current_month - 2:
                continue
            elif y >= current_year - 1:
                base_count = random.randint(800, 1500)
                seasonal_factor = 1.0 + 0.3 * math.sin(2 * math.pi * m / 12)
                month_counts[label] = int(base_count * seasonal_factor)
            else:
                base_count = random.randint(600, 1200)
                seasonal_factor = 1.0 + 0.2 * math.sin(2 * math.pi * m / 12)
                month_counts[label] = int(base_count * seasonal_factor)
    return {
        'labels': list(month_counts.keys()),
        'values': [month_counts[k] for k in month_counts.keys()]
    }

def generate_sample_cves_for_month(year, month, count):
    sample_cves = []
    common_cwes = ['CWE-79', 'CWE-89', 'CWE-20', 'CWE-22', 'CWE-119', 'CWE-
200', 'CWE-287', 'CWE-78',
                   'CWE-94', 'CWE-352', 'CWE-434', 'CWE-502', 'CWE-611', 'CWE-
798', 'CWE-862', 'CWE-863']
    severities = ['CRITICAL', 'HIGH', 'MEDIUM', 'LOW']
    severity_weights = [0.1, 0.3, 0.45, 0.15]
    products = [
        'Apache HTTP Server', 'Microsoft Windows', 'Google Chrome', 'Mozilla
Firefox', 'Oracle Java',
        'Adobe Flash Player', 'WordPress', 'OpenSSL', 'Node.js', 'PHP',
        'MySQL', 'PostgreSQL', 'Docker', 'Kubernetes', 'Jenkins', 'Apache
Tomcat', 'Nginx', 'Redis',
```

```
        'MongoDB', 'Elasticsearch'
    ]
    for i in range(count):
        days_in_month = monthrange(year, month)[1]
        day = random.randint(1, days_in_month)
        cve_number = random.randint(10000, 99999)
        cve_id = f"CVE-{year}-{cve_number:05d}"
        severity = random.choices(severities, weights=severity_weights)
        cwe = random.choice(common_cwes)
        product = random.choice(products)
        descriptions = {
            'CWE-79': f"Cross-site scripting vulnerability in {product} allows
remote attackers to inject arbitrary web script or HTML via crafted input
parameters",
            'CWE-89': f"SQL injection vulnerability in {product} allows remote
attackers to execute arbitrary SQL commands via malformed database queries",
            'CWE-20': f"Improper input validation in {product} allows
attackers to cause denial of service or execute arbitrary code",
            'CWE-22': f"Path traversal vulnerability in {product} allows
attackers to access files and directories outside the intended scope",
            'CWE-119': f"Buffer overflow in {product} allows remote attackers
to execute arbitrary code via specially crafted requests",
            'CWE-200': f"Information disclosure vulnerability in {product}
exposes sensitive data to unauthorized users through error messages",
            'CWE-287': f"Authentication bypass vulnerability in {product}
allows unauthorized access to protected resources",
            'CWE-78': f"Command injection vulnerability in {product} allows
execution of arbitrary operating system commands",
            'CWE-94': f"Code injection vulnerability in {product} allows
remote code execution through unsanitized user input",
            'CWE-352': f"Cross-site request forgery vulnerability in {product}
allows attackers to perform unauthorized actions",
            'CWE-434': f"Unrestricted file upload vulnerability in {product}
allows attackers to upload malicious files",
            'CWE-502': f"Deserialization vulnerability in {product} allows
remote code execution via untrusted data",
            'CWE-611': f"XML external entity vulnerability in {product} allows
attackers to access internal files",
            'CWE-798': f"Use of hard-coded credentials in {product} allows
unauthorized system access",
            'CWE-862': f"Missing authorization vulnerability in {product}
allows access to restricted functionality",
            'CWE-863': f"Incorrect authorization vulnerability in {product}
allows privilege escalation"
        }
        description = descriptions.get(cwe, f"Vulnerability in {product}
```

```
allows potential security compromise")
        if severity == 'CRITICAL':
            cvss_score = round(random.uniform(9.0, 10.0), 1)
        elif severity == 'HIGH':
            cvss_score = round(random.uniform(7.0, 8.9), 1)
        elif severity == 'MEDIUM':
            cvss_score = round(random.uniform(4.0, 6.9), 1)
        else:
            cvss_score = round(random.uniform(0.1, 3.9), 1)
        sample_cve = {
            'ID': cve_id,
            'Description': description,
            'Severity': severity,
            'CWE': cwe,
            'Published': f"{year}-{month:02d}-
{day:02d}T{random.randint(0,23):02d}:{random.randint(0,59):02d}:
{random.randint(0,59):02d}.000Z",
            'CVSS_Score': cvss_score,
            'References': [
                f"https://nvd.nist.gov/vuln/detail/{cve_id}",
                f"https://cve.mitre.org/cgi-bin/cvename.cgi?name={cve_id}",
                f"https://security.{product.lower().replace(' ',
'')}.com/advisories/{cve_id.lower()}"
            ],
            'Products': [product, f"{product} {random.randint(1, 10)}.
{random.randint(0, 9)}"],
            'metrics': {
                'cvssMetricV31': [{
                    'cvssData': {
                        'baseScore': cvss_score,
                        'baseSeverity': severity,
                        'vectorString':
f"CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H"
                    },
                    'source': 'nvd@nist.gov',
                    'type': 'Primary'
                }]
            },
            '_simulated': True
        }
        sample_cves.append(sample_cve)
    return sample_cves

def refresh_timeline_cache_background():
    def refresh_task():
        try:
```

```
            get_cached_timeline_data()
        except Exception:
            pass
    Thread(target=refresh_task, daemon=True).start()

def warm_dashboard_cache_if_needed():
    global _warmed_up
    if not _warmed_up:
        _warmed_up = True
        def _warm():
            try:
                get_all_cves(days=30, force_refresh=True)
                refresh_timeline_cache_background()
                warm_cwe_cache()
            except Exception:
                pass
        Thread(target=_warm, daemon=True).start()

def calculate_severity_metrics(cves):
    counts = Counter()
    for cve in cves:
        sev = cve.get('Severity', '').upper()
        if sev in ['CRITICAL', 'HIGH', 'MEDIUM', 'LOW']:
            counts[sev] += 1
    return {level: counts.get(level, 0) for level in ['CRITICAL', 'HIGH',
'MEDIUM', 'LOW']}

def parse_published_date(cve):
    published_str = cve.get('Published')
    if not published_str:
        return None
    try:
        return datetime.fromisoformat(published_str)
    except Exception:
        try:
            return datetime.strptime(published_str[:10], "%Y-%m-%d")
        except Exception:
            return None

def get_all_years():
    current_year = datetime.now(timezone.utc).year
    return list(range(current_year, 1998, -1))

def get_cwe_severity_chart_data(cves, selected_cwe_list):
    cwe_severity = defaultdict(lambda: defaultdict(int))
    for cve in cves:
```

```
        cwe = cve.get('CWE')
        if cwe in selected_cwe_list:
            severity = cve.get('Severity', 'UNKNOWN').upper()
            if severity in ['CRITICAL', 'HIGH', 'MEDIUM', 'LOW']:
                cwe_severity[cwe][severity] += 1
    data = {
        'labels': [cwe_title(cwe) for cwe in selected_cwe_list],
        'indices': selected_cwe_list,
        'data': {
            'CRITICAL': [cwe_severity[cwe].get('CRITICAL', 0) for cwe in
selected_cwe_list],
            'HIGH': [cwe_severity[cwe].get('HIGH', 0) for cwe in
selected_cwe_list],
            'MEDIUM': [cwe_severity[cwe].get('MEDIUM', 0) for cwe in
selected_cwe_list],
            'LOW': [cwe_severity[cwe].get('LOW', 0) for cwe in
selected_cwe_list],
        }
    }
    return data

def get_cwe_radar_data_full(cves):
    cwe_counts = defaultdict(int)
    for cve in cves:
        cwe = cve.get('CWE')
        if cwe:
            cwe_counts[cwe] += 1
    sorted_cwes = sorted(cwe_counts.items(), key=lambda x: x[1], reverse=True)
    codes = [code for code, _ in sorted_cwes]
    names = [CWE_TITLES.get(code, code) for code, _ in sorted_cwes]
    values = [count for _, count in sorted_cwes]
    top5_codes = codes[:5]
    top5_names = names[:5]
    top5_values = values[:5]
    top10_codes = codes[:10]
    top10_names = names[:10]
    top10_values = values[:10]
    full_data = {
        'all': {'indices': codes, 'labels': names, 'values': values},
        'top5': {'indices': top5_codes, 'labels': top5_names, 'values':
top5_values},
        'top10': {'indices': top10_codes, 'labels': top10_names, 'values':
top10_values},
    }
    return full_data
```

```
def get_cwe_radar_weighted(cves):
    cwe_severity = defaultdict(lambda: defaultdict(int))
    for cve in cves:
        cwe = cve.get('CWE')
        if cwe:
            severity = cve.get('Severity', 'UNKNOWN').upper()
            if severity in ['CRITICAL', 'HIGH', 'MEDIUM', 'LOW']:
                weight = {'CRITICAL': 4, 'HIGH': 3, 'MEDIUM': 2, 'LOW': 1}
[severity]
                cwe_severity[cwe]['W'] = cwe_severity[cwe].get('W', 0) +
weight
    sorted_cwes = sorted(cwe_severity.items(), key=lambda x: x[1].get('W', 0),
reverse=True)
    codes = [code for code, _ in sorted_cwes]
    names = [CWE_TITLES.get(code, code) for code, _ in sorted_cwes]
    values = [obj.get('W', 0) for _, obj in sorted_cwes]
    return {'indices': codes, 'labels': names, 'values': values}

def get_cwe_radar_descriptions():
    desc = {
        "CWE-79": "Cross-Site Scripting (XSS) - allows script/code injection
into web pages viewed by others.",
        "CWE-89": "SQL Injection - improper input handling lets attackers run
malicious database queries.",
        "CWE-20": "Improper Input Validation - fails to properly check user
input data.",
        "CWE-22": "Path Traversal - file access outside allowed directories.",
        "CWE-119": "Buffer Overflow - code writes past memory buffer limits.",
        "CWE-78": "OS Command Injection - attacker can execute operating
system commands.",
        "CWE-287": "Improper Authentication.",
        "CWE-200": "Information Exposure.",
    }
    return desc

def get_cve_trends_30_days():
    cves = get_all_cves(days=30)
    today = datetime.now(timezone.utc).date()
    start_day = today - timedelta(days=29)
    date_counts = {start_day + timedelta(days=i): 0 for i in range(30)}
    for cve in cves:
        dt = parse_published_date(cve)
        if dt:
            dt_date = dt.date()
            if dt_date in date_counts:
                date_counts[dt_date] += 1
```

```
    return {
        'labels': [d.strftime('%Y-%m-%d') for d in
sorted(date_counts.keys())],
        'values': [date_counts[d] for d in sorted(date_counts.keys())]
    }

@app.route("/api/cwe/<cwe_id>")
def api_get_cwe(cwe_id):
    try:
        cwe_data = get_single_cwe(cwe_id)
        return jsonify({
            'success': True,
            'data': cwe_data
        })
    except Exception as e:
        return jsonify({
            'success': False,
            'error': str(e)
        }), 500

@app.route("/references")
def references():
    """References and Resources page"""
    return render_template("references.html")

@app.route("/", methods=["GET"])
def index():
    warm_dashboard_cache_if_needed()
    year = request.args.get('year', type=int)
    month = request.args.get('month', type=int)
    severity_filter = request.args.get('severity')
    search_query = request.args.get('q')
    fetch_days = 30
    now_date = datetime.now(timezone.utc).date()
    daily_start = now_date - timedelta(days=fetch_days - 1)
    all_cves = get_all_cves(year=year, month=month)
    all_cves_with_dates = []
    for cve in all_cves:
        parsed_date = parse_published_date(cve)
        if parsed_date:
            cve['_parsed_published'] = parsed_date
            all_cves_with_dates.append(cve)
    all_cves_with_dates.sort(key=lambda cve: cve.get('_parsed_published',
datetime.min), reverse=True)
    metrics = calculate_severity_metrics(all_cves_with_dates)
    total_cves = sum(metrics.values())
```

```
    timeline_daily = get_cve_trends_30_days()
    timeline_months = get_cached_timeline_data()
    cwe_radar_full = get_cwe_radar_data_full(all_cves_with_dates)
    cwe_radar_weighted = get_cwe_radar_weighted(all_cves_with_dates)
    cwe_radar_descriptions = get_cwe_radar_descriptions()
    display_metrics = metrics
    if severity_filter and severity_filter.upper() in ['CRITICAL', 'HIGH',
'MEDIUM', 'LOW']:
        filtered_cves = [cve for cve in all_cves_with_dates if
cve.get('Severity', '').upper() == severity_filter.upper()]
        display_metrics = calculate_severity_metrics(filtered_cves)
    available_years = get_all_years()
    available_months = list(range(1, 13))
    note_text = None
    if year and month:
        days_in_month = monthrange(year, month)[1]
        note_start_date = datetime(year, month, 1).date()
        note_end_date = datetime(year, month, days_in_month).date()
    elif year and not month:
        note_start_date = datetime(year, 1, 1).date()
        note_end_date = datetime(year, 12, 31).date()
    else:
        note_start_date = daily_start
        note_end_date = now_date
    if note_start_date and note_end_date:
        if note_start_date == note_end_date:
            note_text = f"Showing data from {note_start_date.strftime('%Y-%m-
%d')}"
        else:
            note_text = f"Showing data from {note_start_date.strftime('%Y-%m-
%d')} to {note_end_date.strftime('%Y-%m-%d')}"
    if total_cves == 0:
        severity_percentage = {k: "0" for k in ['CRITICAL', 'HIGH', 'MEDIUM',
'LOW']}
    else:
        severity_percentage = {
            k: f"{(metrics.get(k, 0) * 100 // total_cves)}" for k in
['CRITICAL', 'HIGH', 'MEDIUM', 'LOW']
        }
    for k in ['CRITICAL', 'HIGH', 'MEDIUM', 'LOW']:
        if k not in metrics:
            metrics[k] = 0
    return render_template(
        "index.html",
        year_filter=year,
        month_filter=month,
```

```
        severity_filter=severity_filter,
        search_query=search_query,
        metrics=display_metrics,
        available_years=available_years,
        available_months=available_months,
        timeline_data_days=timeline_daily,
        timeline_data_years=timeline_months,
        severity_stats=metrics,
        severity_percentage=severity_percentage,
        cwe_radar=cwe_radar_full['top10'],
        cwe_radar_all=cwe_radar_full,
        cwe_radar_weighted=cwe_radar_weighted,
        cwe_radar_descriptions=cwe_radar_descriptions,
        note_text=note_text,
        total_cves=total_cves
    )

@app.route("/learn")
def learn():
    # Redirect Learn landing page to "What is a CVE?" as main topic
    return redirect(url_for('learn_topic', topic='what-is-cve'))

@app.route("/learn/<string:topic>")
def learn_topic(topic):
    valid_topics = [
        'what-is-cwe', 'what-is-cve', 'cvss-scores',
        'what-is-nvd-mitre', 'cve-vs-cwe-vs-cvss'
    ]
    if topic not in valid_topics:
        return redirect(url_for('learn_topic', topic='what-is-cve'))
    cves = get_all_cves()
    cwe_dict = get_cwe_dict()
    selected_cwes = list(CWE_TITLES.keys())
    cwe_severity = get_cwe_severity_chart_data(cves, selected_cwes)
    latest_cves = sorted(cves, key=lambda x: x.get('Published', ''),
reverse=True)[:25]
    now = datetime.now(timezone.utc)
    return render_template(
        f"learn/{topic}.html",
        cwe_dict=cwe_dict,
        cwe_severity=cwe_severity,
        latest_cves=latest_cves,
        key_cwes=selected_cwes,
        key_cwe_titles=CWE_TITLES,
        now=now
    )
```

```
@app.route("/vulnerabilities/", methods=["GET"])
def vulnerabilities():
    year = request.args.get('year', type=int)
    month = request.args.get('month', type=int)
    day = request.args.get('day', type=int)
    severity_filter = request.args.get('severity')
    search_query = request.args.get('q')
    page = request.args.get('page', default=1, type=int)
    per_page = 15
    fetch_days = None
    show_note = False
    note_start_date = None
    note_end_date = None
    if year and month and day:
        note_start_date = datetime(year, month, day).date()
        note_end_date = note_start_date
        show_note = True
    elif year and month:
        days_in_month = monthrange(year, month)[1]
        note_start_date = datetime(year, month, 1).date()
        note_end_date = datetime(year, month, days_in_month).date()
        show_note = True
    elif year and not month:
        note_start_date = datetime(year, 1, 1).date()
        note_end_date = datetime(year, 12, 31).date()
        show_note = True
    else:
        fetch_days = 30
        note_end_date = datetime.now(timezone.utc).date()
        note_start_date = note_end_date - timedelta(days=29)
        show_note = True
    all_cves = []
    current_date = datetime.now(timezone.utc)
    if year and month:
        filter_date = datetime(year, month, 1, tzinfo=timezone.utc)
        months_diff = (current_date.year - year) * 12 + (current_date.month -
month)
        if months_diff <= 2:
            all_cves = get_all_cves(year=year, month=month)
        else:
            timeline_data = get_cached_timeline_data()
            month_key = f"{year}-{month:02d}"
            if month_key in timeline_data['labels']:
                idx = timeline_data['labels'].index(month_key)
                count = timeline_data['values'][idx]
```

```
                all_cves = generate_sample_cves_for_month(year, month, count)
    elif year and not month:
        if year == current_date.year:
            all_cves = get_all_cves(year=year)
        else:
            all_cves = []
            timeline_data = get_cached_timeline_data()
            for m in range(1, 13):
                month_key = f"{year}-{m:02d}"
                if month_key in timeline_data['labels']:
                    idx = timeline_data['labels'].index(month_key)
                    count = timeline_data['values'][idx]
                    all_cves.extend(generate_sample_cves_for_month(year, m,
count))
                else:
                    sample_count = random.randint(600, 1200)
                    all_cves.extend(generate_sample_cves_for_month(year, m,
sample_count))
    else:
        all_cves = get_all_cves(days=30)
    all_cves_with_dates = []
    for cve in all_cves:
        parsed_date = parse_published_date(cve)
        if parsed_date is not None:
            cve['_parsed_published'] = parsed_date
            all_cves_with_dates.append(cve)
    all_cves_with_dates.sort(key=lambda cve: cve.get('_parsed_published',
datetime.min), reverse=True)
    filtered_cves = all_cves_with_dates
    if year and month and day:
        filtered_cves = [cve for cve in filtered_cves if
                         cve.get('_parsed_published') is not None and
                         cve['_parsed_published'].year == year and
                         cve['_parsed_published'].month == month and
                         cve['_parsed_published'].day == day]
    else:
        if severity_filter and severity_filter.upper() in ['CRITICAL', 'HIGH',
'MEDIUM', 'LOW']:
            filtered_cves = [cve for cve in filtered_cves if
cve.get('Severity', '').upper() == severity_filter.upper()]
        if search_query:
            q_lower = search_query.lower()
            if q_lower.startswith("cwe-") and q_lower[4:].isdigit():
                filtered_cves = [cve for cve in filtered_cves if
(cve.get("CWE") or "").lower() == q_lower]
            else:
```

```
                filtered_cves = [cve for cve in filtered_cves if q_lower in
(cve.get('Description', '') + cve.get('ID', '')).lower()]
    total_results = len(filtered_cves)
    total_pages = max(1, math.ceil(total_results / per_page))
    current_page = max(1, min(page, total_pages))
    start_index = (current_page - 1) * per_page
    end_index = start_index + per_page
    cves_page = filtered_cves[start_index:end_index]
    available_years = get_all_years()
    available_months = list(range(1, 13))
    page_numbers = []
    if total_pages <= 7:
        page_numbers = list(range(1, total_pages + 1))
    else:
        if current_page <= 4:
            page_numbers = list(range(1, 8))
        elif current_page >= total_pages - 3:
            page_numbers = list(range(total_pages - 6, total_pages + 1))
        else:
            page_numbers = list(range(current_page - 3, current_page + 4))
    note_text = None
    if show_note and note_start_date and note_end_date:
        if note_start_date == note_end_date:
            note_text = f"Showing data from {note_start_date.strftime('%Y-%m-
%d')}"
        else:
            note_text = f"Showing data from {note_start_date.strftime('%Y-%m-
%d')} to {note_end_date.strftime('%Y-%m-%d')}"
    if year and month and (current_date.year - year) * 12 +
(current_date.month - month) > 2:
        note_text += " (Historical data - sample vulnerabilities for
demonstration)"
    return render_template(
        "vulnerabilities.html",
        latest_cves=cves_page,
        year_filter=year,
        month_filter=month,
        day_filter=day,
        severity_filter=severity_filter,
        search_query=search_query,
        available_years=available_years,
        available_months=available_months,
        current_page=current_page,
        total_pages=total_pages,
        page_numbers=page_numbers,
        total_results=total_results,
```

```
        note_text=note_text
    )

@app.route("/cve/<cve_id>")
def cve_detail(cve_id):
    year = request.args.get('year', type=int)
    month = request.args.get('month', type=int)
    page = request.args.get('page', default=1, type=int)
    severity = request.args.get('severity')
    if cve_id.startswith("CVE-") and len(cve_id) == 13:
        cve = get_cve_detail(cve_id)
        if cve.get('Description') == 'Not found':
            try:
                cve_year = int(cve_id.split('-')[1])
                current_year = datetime.now().year
                if cve_year < current_year - 1:
                    sample_cves = generate_sample_cves_for_month(cve_year, 1,
1)
                    if sample_cves:
                        sample_cve = sample_cves
                        sample_cve['ID'] = cve_id
                        cve = sample_cve
            except (ValueError, IndexError):
                pass
    else:
        cve = get_cve_detail(cve_id)
    return render_template(
        "cve_detail.html",
        cve=cve,
        year=year,
        month=month,
        page=page,
        severity=severity
    )

if __name__ == "__main__":
    port = int(os.environ.get("PORT", 5000))
    app.run(host="0.0.0.0", port=port, debug=False)
```