МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Базы данных»

Тема: Тестирование БД на безопасность.

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Михайлова С.В.

Санкт-Петербург

2025

Цель работы.

Написать web-server - api для созданной ранее базы данных. Создать эндпоинты на каждый запрос и протестировать их.

Задание.

Вариант 22.

Пусть требуется создать программную систему для поиска вакансий (аналог hh.ru). Такая система должна обеспечивать хранение сведений о работодателях и работниках. Эти сведения включают в себя (для работника) - паспортные работника, данные трудовой книжки, ИНН, дата рождения, информацию о среднем/высшем(их) образованиях, дата поступления на работу, в институт, информация об предыдущей работе(ах) из трудовой книжки. Данные трудовой книжки – это ее номер и дата выдачи, а также даты и номера приказов о зачислении и увольнении, о переходе в другое подразделение или об изменении должности. Кроме того, для работник может создать 1/несколько резюме, с указанием желаемой должности, ЗП, свои умения/навыки. Работодатель имеет возможность создавать/удалять/помещать в архив вакансии. Вакания имеет название, ЗП, должность, адрес, требования, условия, комментарий, требуемый опыт. У работодателя есть страница с указанием информации о себе - название, фото, описание, файлы презентации, сфера деятельности (IT, финансовая и т.п.), количество вакансий (формируется на основе списка вакансий).

Система должна давать ответы на следующие вопросы:

- Какие вакансии есть у данной компании?
- Какая вакансия подходит мне по названию?
- Сколько поблизости вакансий от меня (указание улицы)?
- Какие вакансии были помещены в архив у компании?
- Средняя ЗП каждого работодателя?

- Сколько работников ищут работу, имея высшее образование?
- Сколько работников имело более 3-х мест работы?
- Какие вакансии имеют ЗП более 100 000р и не требуют опыта работы?

Задачи:

- Сделать простой web-сервер для выполнения запросов из ЛР3, например с express.js. Не обязательно делать авторизацию и т.п., хватит одного эдпоинта на каждый запрос, с параметрами запроса как query parameters.
- Намеренно сделайте несколько (2-3) запроса, подверженных SQL-инъекциям
- Проверьте Ваше API с помощью sqlmap (или чего-то аналогичного), передав эндпоинты в качестве целей атаки. Посмотрите, какие уязвимости он нашёл (и не нашёл), опишите пути к исправлению.
- +2 балла, если напишете эндпоинт с уязвимостью, которая не находится sqlmap-ом.

Выполнение работы.

Ранее был использован Python вместе с SQLAlchemy, поэтому в данной работе для реализации web-сервера использован Flask.

Для использования web-сервера написан скрипт app.py, разворачиваемый на порте 5000. Для каждого запроса из задания создан эндпоинт. Запросы прописаны при помощи ORM. Также созданы 2 отдельных запроса, подверженных SQL-инъекциям. Они реализованы через формирование SQL-запроса в виде строки, в которую вставляется полученное значение без обработки.

Запросы:

```
1. @app.route('/api/get-post-name', methods=['GET'])
   def get_post_name():
         post_name = request.args.get('post-name', type=str,
   default='Разработчик')
       with local_session() as db:
           query = f"SELECT * FROM post WHERE name LIKE '%{post_name}%'"
           result = db.execute(text(query))
           posts_data = [dict(row._mapping) for row in result]
           return jsonify(posts_data)

2. @app.route('/api/get-requirement-id', methods=['GET'])
   def get_requirements_id():
       requirement_id = request.args.get('requirement-id')
       with local_session() as db:
               query = f"SELECT * FROM requirement WHERE requirement_id =
   {requirement_id}"
           result = db.execute(text(query))

           requirement_data = [dict(row._mapping) for row in result]
           return jsonify(requirement_data)
```

В первом используется вставка значения типа str в LIKE. В такой запрос можно включить SQL-инъекции вида:

str'; other_query

Во втором запросе у получаемого аргумента не указан его тип и значение по умолчанию, после чего он так же напрямую добавляется в строку. Так как

тут подразумевается значение int, в такой запрос можно включить sql-инъекцию вида:

int; other_query

Например на рис. 1 показан вывод использования последнего SQL-запроса, подверженного инъекции, без ее использования /api/get-requirement-id?requirement-id=10

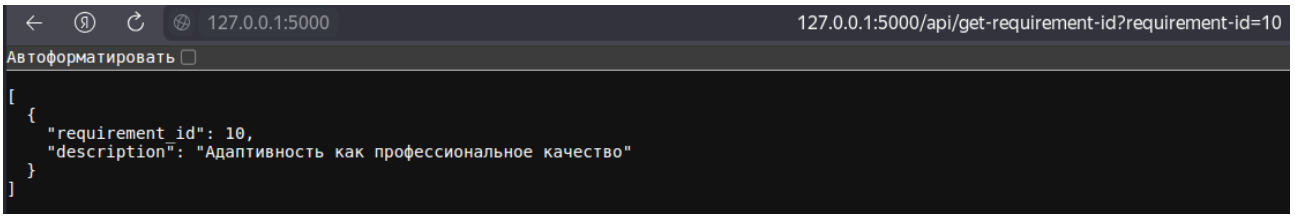

Рисунок 1 - Вывод SQL-запроса, подверженного инъекции без инъекции

На рис. 2 показан вывод использования последнего SQL-запроса, подверженного инъекции с ее использованием /api/get-requirement-id?requirement-id=10; SELECT * FROM post WHERE name LIKE '%Аналитик%

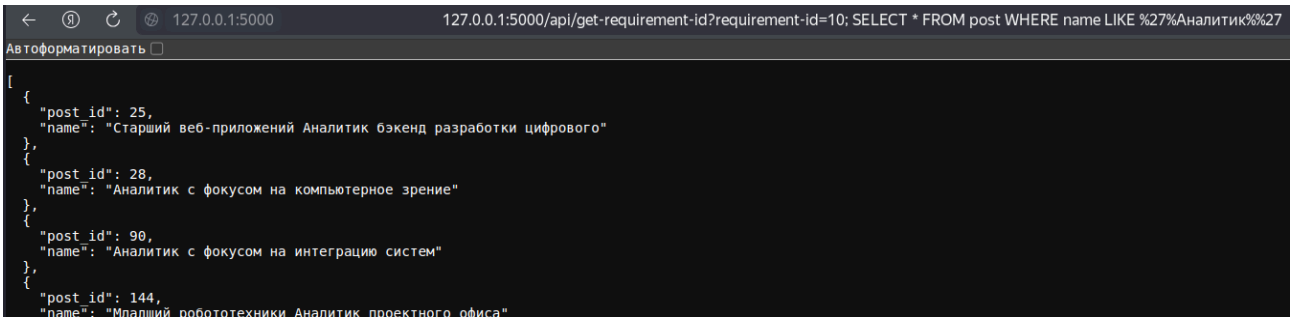

Рисунок 2 - Вывод SQL-запроса, подверженного инъекции с ее использованием

Проверка через использование sqlmap подтверждает наличие уязвимости в данном месте. Результат проверки показан на рис.3

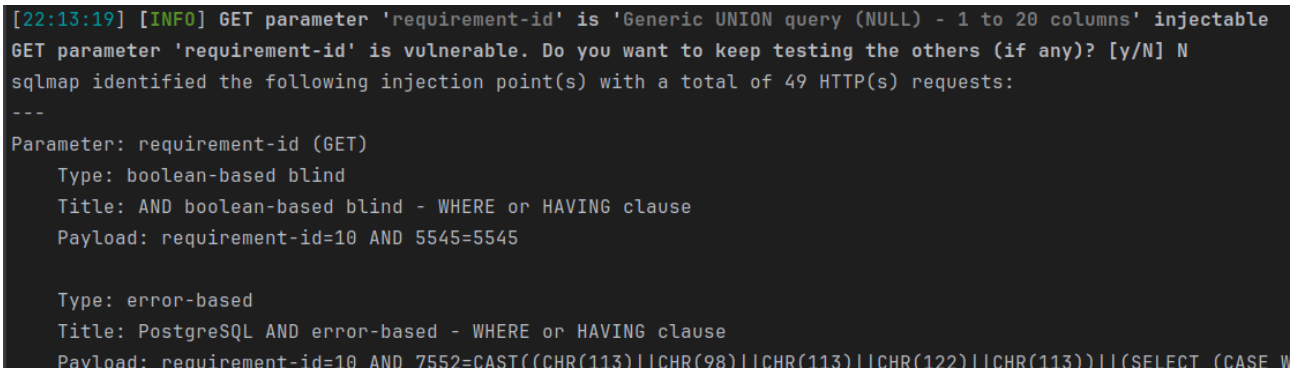

Рисунок 3 - Результат проверки уязвимого запроса через sqlmap

Избавиться от уязвимости можно параметризировав запрос, то есть не вставляя значение на прямую, а вставляя через параметр в определенное место, при этом делая перед этим валидацию. В данном случае требуется значение типа int, поэтому можно сделать проверку на это. Валидацию можно сделать либо отдельным блоком для этого, либо, в данном случае подойдет более простая валидация - уточнение типа принимаемого значения и его значение по умолчанию при определении аргумента. Теперь запрос выглядит так:

```
@app.route('/api/get-requirement-id', methods=['GET'])
def get_requirements_id():
        requirement_id = request.args.get('requirement-id', type=int, default=10)
    with local_session() as db:
        query = text("SELECT * FROM requirement WHERE requirement_id = :requirement_id")
        result = db.execute(query, {'requirement_id': requirement_id})
        requirement_data = [dict(row._mapping) for row in result]
        return jsonify(requirement_data)
```

Результат запроса проверки измененного запроса можно увидеть на рис. 4.

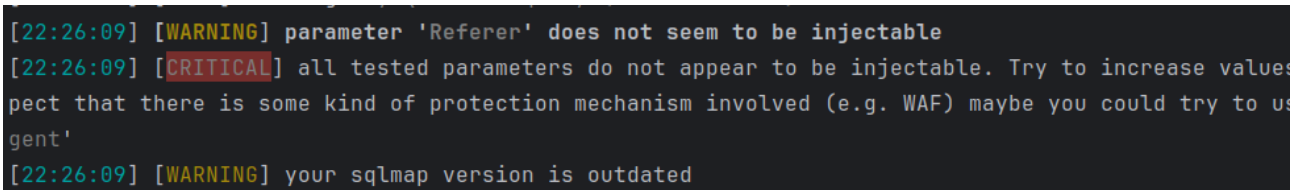

Рисунок 4 - Результат проверки исправленного запроса

Так как в данном запросе был нужен параметр типа int, то его получилось исправить путем валидации заранее. В случае с первым запросом это будет сделать труднее, так как там используются строки. В таком случае надо будет либо ограничивать длину вводимых слов, либо как-то удалять запрещенные символы или пропускать только какие-то отдельные слова, определенные где-нибудь. В данном случае лучше подойдет второй вариант защиты от SQL-инъекций - использование ORM.

Переписанный запрос:

```
@app.route('/api/get-requirement-id', methods=['GET'])
def get_requirements_id():
    requirement_id = request.args.get('requirement-id')
    with local_session() as db:
```

```
    result = db.query(
        Requirement
    ) \
    .filter(Requirement.requirement_id == requirement_id) \
    .all()
    return jsonify([{
        'requirement_id': r.requirement_id,
        'description': r.description
    } for r in result])
```

Результат проверки переписанного запроса с использованием ORM приведен на рис. 5.

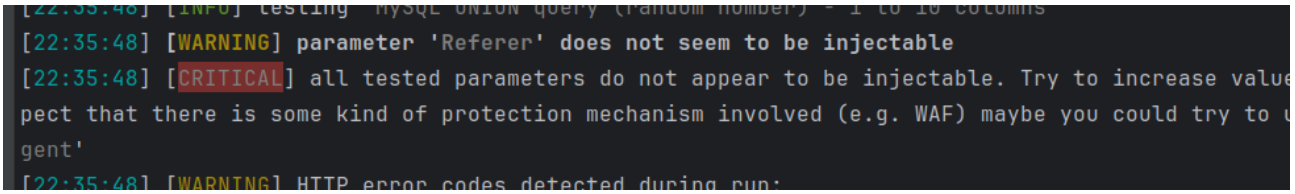

Рисунок 5 - Результат проверки запроса, исправленного с помощью ORM

В идеале объединить эти два подхода и в запрос с использованием ORM добавить начальную валидацию на int, так как при вводе значения не int с базой данных все будет хорошо, но возникнет ошибка при составлении запроса.

Первый из двух SQL запросов, подверженных SQL-инъекции так же не проходит тест от sqlmap. Результат показан на рис. 6.

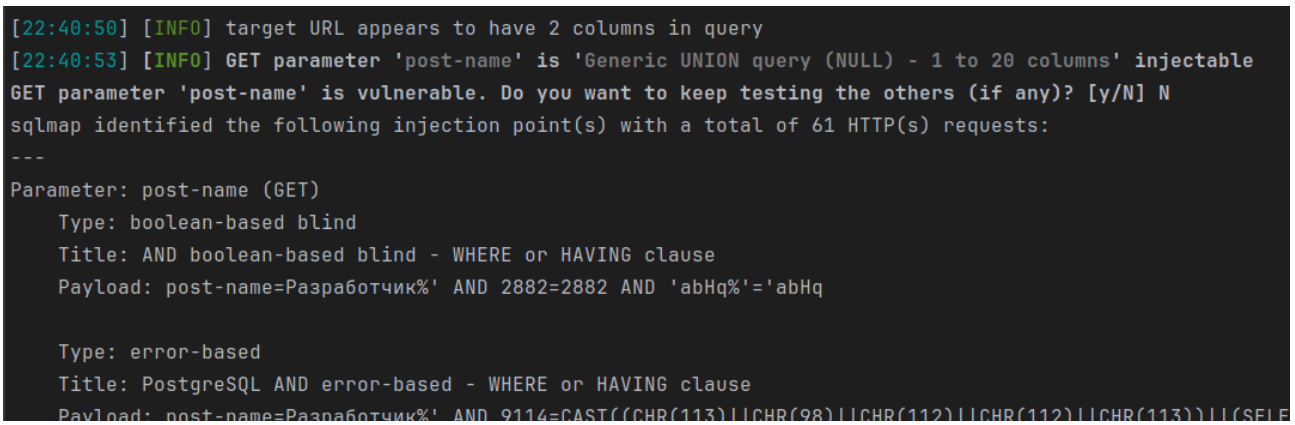

Рисунок 6 - Результат проверки запроса, подверженного SQL-инъекции

Запрос переписан с использованием ORM, теперь он выглядит следующим образом:

```
@app.route('/api/get-post-name', methods=['GET'])
def get_post_name():
    post_name = request.args.get('post-name', type=str,
default='Разработчик')
```

```
    with local_session() as db:
        result = db.query(
            Post
        ) \
        .filter(Post.name.like(f'%{post_name}%')) \
        .all()

        return jsonify([{
            'post_id': r.post_id,
            'name': r.name
        } for r in result])
```

На рис. 7 представлен результат проверки переписанного при помощи ORM запроса через sqlmap.

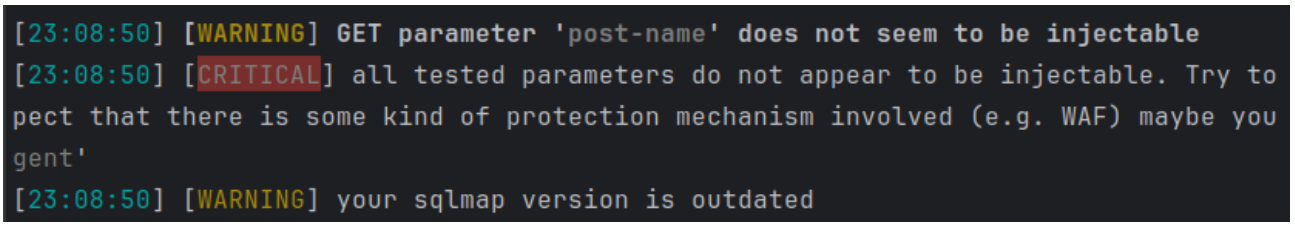

Рисунок 7 - Результат проверки переписанного при помощи ORM уязвимого запроса

Кроме этого через sqlmap поверены остальные запросы. Уязвимостей не выявлено.

Разработанный код см. в приложении А.

Выводы.

При выполнении лабораторной работы изучен процесс создания web-сервера на Flask. Создан web-сервер с эндпоинтами для всех запросов, написанных ранее, а так же добавлены эндпоинты для запросов, подверженных SQL-инъекциям. Через sqlmap найдены уязвимости в запросах, созданных для проверки уязвимости. В запросах, написанных ранее уязвимостей не найдено, так как они написаны с использованием ORM.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Название файла: app.py

```
from flask import Flask, request, jsonify
from sqlalchemy import create_engine, func, and_, desc, or_, text
from sqlalchemy.orm import sessionmaker
from flask.json.provider import DefaultJSONProvider
from create import *
import json
from main import DATABASE_URL
from decimal import Decimal
from datetime import date, datetime

class RussianJSONProvider(DefaultJSONProvider):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.ensure_ascii = False

    def dumps(self, obj, **kwargs):
        kwargs.setdefault('ensure_ascii', False)
        return json.dumps(obj, **kwargs, cls=CustomJSONEncoder)

    def _extract_sqlalchemy_object(self, obj):
        if hasattr(obj, '__table__'):
            result = {}
            for column in obj.__table__.columns:
                value = getattr(obj, column.name)
                if value is not None:
                    result[column.name] = value
            return result
        return obj


class CustomJSONEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Decimal):
            return float(obj)
        elif isinstance(obj, (date, datetime)):
            return obj.isoformat()
        return super().default(obj)
```

```
app = Flask(__name__)
app.json = RussianJSONProvider(app)
app.config['JSON_AS_ASCII'] = False

engine = create_engine(DATABASE_URL)
local_session = sessionmaker(bind=engine)


@app.route('/api/get-vacancy-company', methods=['GET'])
def get_vacancy_company():
       company_name  =  request.args.get('company-name',  type=str,  default='ЗАО
«Одинцов Сидоров»')
    with local_session() as db:
        result = db.query(
            Vacancy.name.label('vacancy_name'),
            Employer.name.label('company'),
            Post.name.label('post'),
            Vacancy.salary,
            Vacancy.address,
            Vacancy.experience,
            Vacancy.comment,
            Vacancy.status
        ).join(Employer)\
        .join(Post)\
        .filter(Employer.name == company_name) \
        .order_by('company')\
        .all()

        return jsonify([{
            'Компания': r.company,
            'Название_вакансии': r.vacancy_name,
            'Должность': r.post,
            'Зарплата': r.salary,
            'Адрес': r.address,
            'Опыт': r.experience if r.experience else None,
            'Комментарий': r.comment if r.comment else None,
            'Статус': r.status.value
        } for r in result])


@app.route('/api/get-vacancy-name', methods=['GET'])
def get_vacancy_name():
```

```
    vacancy_name = request.args.get('vacancy-name', type=str,
default='Художник')
    with local_session() as db:
        result = db.query(
            Vacancy.name.label('vacancy_name'),
            Employer.name.label('company'),
            Post.name.label('post'),
            Vacancy.salary,
        ).join(Employer) \
        .join(Post) \
        .filter(Vacancy.name.like(f'%{vacancy_name}%')) \
        .order_by('company') \
        .all()

        return jsonify([{
            'Название_вакансии': r.vacancy_name,
            'Компания': r.company,
            'Должность': r.post,
            'Зарплата': r.salary
        } for r in result])



@app.route('/api/get-count-vacancy-street', methods=['GET'])
def get_vacancy_street():
    vacancy_street = request.args.get('vacancy-street', type=str,
default='Ленинградская')
    with local_session() as db:
        result = db.query(
            func.count(Vacancy.name).label('vacancy_count')
        ).filter(Vacancy.address.like(f'%{vacancy_street}%')) \
        .order_by('vacancy_count') \
        .all()

        return jsonify([{
            'Количество_вакансий': r.vacancy_count
        } for r in result])



@app.route('/api/get-archived-vacancies', methods=['GET'])
def get_archived_vacancies():
    company_name = request.args.get('company-name', type=str,
default='Пономарева Лимитед')
```

```
    with local_session() as db:
        result = db.query(
            Vacancy.name.label('vacancy_name'),
            Employer.name.label('company'),
            Post.name.label('post'),
            Vacancy.salary
        ).join(Employer) \
            .join(Post) \
            .filter(and_(
            Employer.name == company_name,
            Vacancy.status == VacancyStatus.archive
        )) \
        .order_by('company') \
        .all()

        return jsonify([{
            'Название_вакансии': r.vacancy_name,
            'Компания': r.company,
            'Должность': r.post,
            'Зарплата': r.salary
        } for r in result])


@app.route('/api/get-mean-salary', methods=['GET'])
def get_mean_salary():
    limit=request.args.get('limit', type=int, default=10)
    with local_session() as db:
        result = db.query(
            Employer.name.label('company'),
            func.round(func.avg(Vacancy.salary), 2).label('mean_salary')
        ).join(Employer) \
        .group_by(Employer.name) \
        .order_by(desc('mean_salary')) \
        .limit(limit) \
        .all()

        return jsonify([{
            'Компания': r.company,
            'Средняя_зп': r.mean_salary
        } for r in result])


@app.route('/api/get-worker-education-count', methods=['GET'])
```

```
def get_worker_education_count():
    with local_session() as db:
        result = db.query(
            func.count(func.distinct(Resume.worker_id)).label('count_worker')
        ).join(Worker)\
        .join(WorkerEducation)\
        .join(Education)\
        .filter(Education.type == EducationType.higher) \
        .order_by('count_worker') \
        .all()

        return jsonify([{
            'Количество_работников': r.count_worker
        } for r in result])


@app.route('/api/get-count-worker-place', methods=['GET'])
def get_count_worker_place():
    with local_session() as db:
        result = db.query(
            func.count(func.distinct(Worker.worker_id)).label('count_worker')
        ).join(WorkerWorkbook)\
        .join(Workbook)\
        .join(WorkbookEmployerWriting)\
        .join(WorkbookWriting)\
        .group_by(Worker.worker_id)\
        .having(func.count(func.distinct(WorkbookEmployerWriting.employer_id)) >
3) \
        .order_by('count_worker') \
        .all()
        count = len(result)

        return jsonify([{
            'Количество_работников': count
        }])


@app.route('/api/get-vacancy-salary-experience', methods=['GET'])
def get_vacancy_salary_experience():
    limit = request.args.get('limit', type=int, default=10)
    with local_session() as db:
        result = db.query(
            Vacancy.name.label('vacancy_name'),
```

```
            Employer.name.label('company'),
            Post.name.label('post'),
            Vacancy.salary,
            Vacancy.experience
        ).join(Employer)\
        .join(Post)\
        .filter(and_(
            Vacancy.salary > 100000,
            or_(
                Vacancy.experience == 0,
                Vacancy.experience.is_(None)
            )
        )) \
        .order_by('company') \
        .limit(limit) \
        .all()

        return jsonify([{
            'Название_вакансии': r.vacancy_name,
            'Компания': r.company,
            'Должность': r.post,
            'Зарплата': r.salary,
            'Опыт работы': r.experience
        } for r in result])


# @app.route('/api/get-post-name', methods=['GET'])
# def get_post_name():
#     post_name = request.args.get('post-name', type=str, default='Разработчик')
#     with local_session() as db:
#         result = db.query(
#             Post
#         ) \
#         .filter(Post.name.like(f'%{post_name}%')) \
#         .all()
#
#         return jsonify([{
#             'post_id': r.post_id,
#             'name': r.name
#         } for r in result])


@app.route('/api/get-post-name', methods=['GET'])
```

```
def get_post_name():
    post_name = request.args.get('post-name', type=str, default='Разработчик')
    with local_session() as db:
        query = f"SELECT * FROM post WHERE name LIKE '%{post_name}%'"
        result = db.execute(text(query))

        posts_data = [dict(row._mapping) for row in result]
        return jsonify(posts_data)


# @app.route('/api/get-requirement-id', methods=['GET'])
# def get_requirements_id():
#     requirement_id = request.args.get('requirement-id')
#     with local_session() as db:
#         result = db.query(
#             Requirement
#         ) \
#         .filter(Requirement.requirement_id == requirement_id) \
#         .all()
#         return jsonify([{
#             'requirement_id': r.requirement_id,
#             'description': r.description
#         } for r in result])


# @app.route('/api/get-requirement-id', methods=['GET'])
# def get_requirements_id():
#     requirement_id = request.args.get('requirement-id', type=int, default=10)
#
#     with local_session() as db:
#             query = text("SELECT * FROM requirement WHERE requirement_id =
:requirement_id")
#         result = db.execute(query, {'requirement_id': requirement_id})
#
#         requirement_data = [dict(row._mapping) for row in result]
#         return jsonify(requirement_data)


@app.route('/api/get-requirement-id', methods=['GET'])
def get_requirements_id():
    requirement_id = request.args.get('requirement-id')
    with local_session() as db:
```

```
        query = f"SELECT * FROM requirement WHERE requirement_id = {requirement_id}"
        result = db.execute(text(query))

        requirement_data = [dict(row._mapping) for row in result]
        return jsonify(requirement_data)

@app.route('/')
def index():
    return '''
    <h1>Безопасные запросы</h1>
    <ul>
        <li><strong>1. Какие вакансии есть у данной компании?</strong><br>
            <a href="/api/get-vacancy-company?company-name=ЗАО «Одинцов Сидоров»">/api/get-vacancy-company?company-name=ЗАО «Одинцов Сидоров»</a></li>

        <li><strong>2. Какая вакансия подходит мне по названию?</strong><br>
                                                                    <a href="/api/get-vacancy-name?vacancy-name=Художник">/api/get-vacancy-name?vacancy-name=Художник</a></li>

        <li><strong>3. Сколько поблизости вакансий от меня (указание улицы)?</strong><br>
                                                                    <a href="/api/get-count-vacancy-street?vacancy-street=Ленинградская">/api/get-count-vacancy-street?vacancy-street=Ленинградская</a></li>

        <li><strong>4. Какие вакансии были помещены в архив у компании?</strong><br>
            <a href="/api/get-archived-vacancies?company-name=Пономарева Лимитед">/api/get-archived-vacancies?company-name=Пономарева Лимитед</a></li>

        <li><strong>5. Средняя ЗП каждого работодателя?</strong><br>
                                                                    <a href="/api/get-mean-salary?limit=20">/api/get-mean-salary?limit=20</a></li>

        <li><strong>6. Сколько работников ищут работу, имея высшее образование?</strong><br>
                                                                    <a href="/api/get-worker-education-count">/api/get-worker-education-count</a></li>

        <li><strong>7. Сколько работников имело более 3-х мест работы?</strong><br>
```

```
                                                                    <a
href="/api/get-count-worker-place">/api/get-count-worker-place</a></li>

        <li><strong>8. Какие вакансии имеют ЗП более 100 000р и не требуют опыта
работы?</strong><br>
                                                                    <a
href="/api/get-vacancy-salary-experience?limit=20">/api/get-vacancy-salary-exper
ience?limit=20</a></li>
    </ul>

    <h1>Небезопасные запросы</h1>
    <ul>
            <li><strong>Вывод названий должностей через прямое встраивание в
LIKE</strong><br>
                                                                    <a
href="/api/get-post-name?post-name=Разработчик">/api/get-post-name?post-name=Раз
работчик</a>
            <br><strong>SQL-инъекция в запрос</strong><br>
               <a href="/api/get-post-name?post-name=Разработчик'; SELECT * FROM
post  WHERE  name  LIKE  'Аналитик">/api/get-post-name?post-name=Разработчик';
SELECT * FROM post WHERE name LIKE 'Аналитик</a>
        </li>

            <li><strong>Вывод описаний требований через прямое встраивание в
=</strong><br>
                                                                    <a
href="/api/get-requirement-id?requirement-id=10">/api/get-requirement-id?require
ment-id=10</a>
            <br><strong>SQL-инъекция в запрос</strong><br>
               <a href="/api/get-requirement-id?requirement-id=10; SELECT * FROM
post  WHERE  name  LIKE  '%Аналитик%'">/api/get-requirement-id?requirement-id=10;
SELECT * FROM post WHERE name LIKE '%Аналитик%</a>
        </li>

    </ul>
    '''


if __name__ == '__main__':
     app.run(debug=True, port=5000, use_reloader=True)from flask import Flask,
request, jsonify
from sqlalchemy import create_engine, func, and_, desc, or_, text
from sqlalchemy.orm import sessionmaker
```

```
from flask.json.provider import DefaultJSONProvider
from create import *
import json
from main import DATABASE_URL
from decimal import Decimal
from datetime import date, datetime

class RussianJSONProvider(DefaultJSONProvider):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.ensure_ascii = False

    def dumps(self, obj, **kwargs):
        kwargs.setdefault('ensure_ascii', False)
        return json.dumps(obj, **kwargs, cls=CustomJSONEncoder)

    def _extract_sqlalchemy_object(self, obj):
        if hasattr(obj, '__table__'):
            result = {}
            for column in obj.__table__.columns:
                value = getattr(obj, column.name)
                if value is not None:
                    result[column.name] = value
            return result
        return obj


class CustomJSONEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Decimal):
            return float(obj)
        elif isinstance(obj, (date, datetime)):
            return obj.isoformat()
        return super().default(obj)


app = Flask(__name__)
app.json = RussianJSONProvider(app)
app.config['JSON_AS_ASCII'] = False

engine = create_engine(DATABASE_URL)
local_session = sessionmaker(bind=engine)
```

```
@app.route('/api/get-vacancy-company', methods=['GET'])
def get_vacancy_company():
       company_name  =  request.args.get('company-name',  type=str,  default='ЗАО
«Одинцов Сидоров»')
    with local_session() as db:
        result = db.query(
            Vacancy.name.label('vacancy_name'),
            Employer.name.label('company'),
            Post.name.label('post'),
            Vacancy.salary,
            Vacancy.address,
            Vacancy.experience,
            Vacancy.comment,
            Vacancy.status
        ).join(Employer)\
        .join(Post)\
        .filter(Employer.name == company_name) \
        .order_by('company')\
        .all()

        return jsonify([{
            'Компания': r.company,
            'Название_вакансии': r.vacancy_name,
            'Должность': r.post,
            'Зарплата': r.salary,
            'Адрес': r.address,
            'Опыт': r.experience if r.experience else None,
            'Комментарий': r.comment if r.comment else None,
            'Статус': r.status.value
        } for r in result])


@app.route('/api/get-vacancy-name', methods=['GET'])
def get_vacancy_name():
               vacancy_name    =    request.args.get('vacancy-name',    type=str,
default='Художник')
    with local_session() as db:
        result = db.query(
            Vacancy.name.label('vacancy_name'),
            Employer.name.label('company'),
            Post.name.label('post'),
            Vacancy.salary,
```

```
        ).join(Employer) \
        .join(Post) \
        .filter(Vacancy.name.like(f'%{vacancy_name}%')) \
        .order_by('company') \
        .all()

        return jsonify([{
            'Название_вакансии': r.vacancy_name,
            'Компания': r.company,
            'Должность': r.post,
            'Зарплата': r.salary
        } for r in result])



@app.route('/api/get-count-vacancy-street', methods=['GET'])
def get_vacancy_street():
             vacancy_street   =   request.args.get('vacancy-street',   type=str,
default='Ленинградская')
    with local_session() as db:
        result = db.query(
            func.count(Vacancy.name).label('vacancy_count')
        ).filter(Vacancy.address.like(f'%{vacancy_street}%')) \
        .order_by('vacancy_count') \
        .all()

        return jsonify([{
            'Количество_вакансий': r.vacancy_count
        } for r in result])


@app.route('/api/get-archived-vacancies', methods=['GET'])
def get_archived_vacancies():
               company_name    =    request.args.get('company-name',    type=str,
default='Пономарева Лимитед')
    with local_session() as db:
        result = db.query(
            Vacancy.name.label('vacancy_name'),
            Employer.name.label('company'),
            Post.name.label('post'),
            Vacancy.salary
        ).join(Employer) \
            .join(Post) \
```

```
            .filter(and_(
                Employer.name == company_name,
                Vacancy.status == VacancyStatus.archive
            )) \
            .order_by('company') \
            .all()

        return jsonify([{
            'Название_вакансии': r.vacancy_name,
            'Компания': r.company,
            'Должность': r.post,
            'Зарплата': r.salary
        } for r in result])


@app.route('/api/get-mean-salary', methods=['GET'])
def get_mean_salary():
    limit=request.args.get('limit', type=int, default=10)
    with local_session() as db:
        result = db.query(
            Employer.name.label('company'),
            func.round(func.avg(Vacancy.salary), 2).label('mean_salary')
        ).join(Employer) \
        .group_by(Employer.name) \
        .order_by(desc('mean_salary')) \
        .limit(limit) \
        .all()

        return jsonify([{
            'Компания': r.company,
            'Средняя_зп': r.mean_salary
        } for r in result])


@app.route('/api/get-worker-education-count', methods=['GET'])
def get_worker_education_count():
    with local_session() as db:
        result = db.query(
            func.count(func.distinct(Resume.worker_id)).label('count_worker')
        ).join(Worker)\
        .join(WorkerEducation)\
        .join(Education)\
        .filter(Education.type == EducationType.higher) \
```

```
        .order_by('count_worker') \
        .all()

        return jsonify([{
            'Количество_работников': r.count_worker
        } for r in result])


@app.route('/api/get-count-worker-place', methods=['GET'])
def get_count_worker_place():
    with local_session() as db:
        result = db.query(
            func.count(func.distinct(Worker.worker_id)).label('count_worker')
        ).join(WorkerWorkbook)\
        .join(Workbook)\
        .join(WorkbookEmployerWriting)\
        .join(WorkbookWriting)\
        .group_by(Worker.worker_id)\
        .having(func.count(func.distinct(WorkbookEmployerWriting.employer_id)) >
3) \
        .order_by('count_worker') \
        .all()
        count = len(result)

        return jsonify([{
            'Количество_работников': count
        }])


@app.route('/api/get-vacancy-salary-experience', methods=['GET'])
def get_vacancy_salary_experience():
    limit = request.args.get('limit', type=int, default=10)
    with local_session() as db:
        result = db.query(
            Vacancy.name.label('vacancy_name'),
            Employer.name.label('company'),
            Post.name.label('post'),
            Vacancy.salary,
            Vacancy.experience
        ).join(Employer)\
        .join(Post)\
        .filter(and_(
            Vacancy.salary > 100000,
```

```
            or_(
                Vacancy.experience == 0,
                Vacancy.experience.is_(None)
            )
        )) \
        .order_by('company') \
        .limit(limit) \
        .all()

        return jsonify([{
            'Название_вакансии': r.vacancy_name,
            'Компания': r.company,
            'Должность': r.post,
            'Зарплата': r.salary,
            'Опыт работы': r.experience
        } for r in result])


# @app.route('/api/get-post-name', methods=['GET'])
# def get_post_name():
#     post_name = request.args.get('post-name', type=str, default='Разработчик')
#     with local_session() as db:
#         result = db.query(
#             Post
#         ) \
#         .filter(Post.name.like(f'%{post_name}%')) \
#         .all()
#
#         return jsonify([{
#             'post_id': r.post_id,
#             'name': r.name
#         } for r in result])


@app.route('/api/get-post-name', methods=['GET'])
def get_post_name():
    post_name = request.args.get('post-name', type=str, default='Разработчик')
    with local_session() as db:
        query = f"SELECT * FROM post WHERE name LIKE '%{post_name}%'"
        result = db.execute(text(query))

        posts_data = [dict(row._mapping) for row in result]
        return jsonify(posts_data)
```

```
# @app.route('/api/get-requirement-id', methods=['GET'])
# def get_requirements_id():
#     requirement_id = request.args.get('requirement-id')
#     with local_session() as db:
#         result = db.query(
#             Requirement
#         ) \
#         .filter(Requirement.requirement_id == requirement_id) \
#         .all()
#         return jsonify([{
#             'requirement_id': r.requirement_id,
#             'description': r.description
#         } for r in result])


# @app.route('/api/get-requirement-id', methods=['GET'])
# def get_requirements_id():
#     requirement_id = request.args.get('requirement-id', type=int, default=10)
#
#     with local_session() as db:
#             query = text("SELECT * FROM requirement WHERE requirement_id =
:requirement_id")
#         result = db.execute(query, {'requirement_id': requirement_id})
#
#         requirement_data = [dict(row._mapping) for row in result]
#         return jsonify(requirement_data)


@app.route('/api/get-requirement-id', methods=['GET'])
def get_requirements_id():
    requirement_id = request.args.get('requirement-id')
    with local_session() as db:
             query = f"SELECT * FROM requirement WHERE requirement_id =
{requirement_id}"
        result = db.execute(text(query))

        requirement_data = [dict(row._mapping) for row in result]
        return jsonify(requirement_data)

@app.route('/')
def index():
```

```
    return '''
    <h1>Безопасные запросы</h1>
    <ul>
        <li><strong>1. Какие вакансии есть у данной компании?</strong><br>
                    <a  href="/api/get-vacancy-company?company-name=ЗАО «Одинцов Сидоров»">/api/get-vacancy-company?company-name=ЗАО «Одинцов Сидоров»</a></li>

        <li><strong>2. Какая вакансия подходит мне по названию?</strong><br>
                                                                    <a href="/api/get-vacancy-name?vacancy-name=Художник">/api/get-vacancy-name?vacancy-name=Художник</a></li>

                <li><strong>3. Сколько поблизости вакансий от меня (указание улицы)?</strong><br>
                                                                    <a href="/api/get-count-vacancy-street?vacancy-street=Ленинградская">/api/get-count-vacancy-street?vacancy-street=Ленинградская</a></li>

                    <li><strong>4. Какие вакансии были помещены в архив у компании?</strong><br>
                     <a  href="/api/get-archived-vacancies?company-name=Пономарева Лимитед">/api/get-archived-vacancies?company-name=Пономарева Лимитед</a></li>

        <li><strong>5. Средняя ЗП каждого работодателя?</strong><br>
                                                                    <a href="/api/get-mean-salary?limit=20">/api/get-mean-salary?limit=20</a></li>

                <li><strong>6. Сколько работников ищут работу, имея высшее образование?</strong><br>
                                                                    <a href="/api/get-worker-education-count">/api/get-worker-education-count</a></li>

                    <li><strong>7. Сколько работников имело более 3-х мест работы?</strong><br>
                                                                    <a href="/api/get-count-worker-place">/api/get-count-worker-place</a></li>

        <li><strong>8. Какие вакансии имеют ЗП более 100 000р и не требуют опыта работы?</strong><br>
                                                                    <a href="/api/get-vacancy-salary-experience?limit=20">/api/get-vacancy-salary-experience?limit=20</a></li>
    </ul>
```

```
    <h1>Небезопасные запросы</h1>
    <ul>
            <li><strong>Вывод названий должностей через прямое встраивание в LIKE</strong><br>
                                                                                <a href="/api/get-post-name?post-name=Разработчик">/api/get-post-name?post-name=Разработчик</a>
            <br><strong>SQL-инъекция в запрос</strong><br>
               <a href="/api/get-post-name?post-name=Разработчик'; SELECT * FROM post WHERE name LIKE 'Аналитик">/api/get-post-name?post-name=Разработчик'; SELECT * FROM post WHERE name LIKE 'Аналитик</a>
        </li>

            <li><strong>Вывод описаний требований через прямое встраивание в =</strong><br>
                                                                                <a href="/api/get-requirement-id?requirement-id=10">/api/get-requirement-id?requirement-id=10</a>
            <br><strong>SQL-инъекция в запрос</strong><br>
               <a href="/api/get-requirement-id?requirement-id=10; SELECT * FROM post WHERE name LIKE '%Аналитик%'">/api/get-requirement-id?requirement-id=10; SELECT * FROM post WHERE name LIKE '%Аналитик%</a>
        </li>

    </ul>
    '''


if __name__ == '__main__':
    app.run(debug=True, port=5000, use_reloader=True)
```