Министерство образования РБ
Учреждение образования «Витебский государственный технологический университет»
Кафедра «Информационные системы и технологии»

Курсовая работа
по дисциплине «Современные средства разработки серверных приложений»
Разработка приложения «Разработка REST-сервиса «Система управления программными продуктами»»

Выполнил: студент группы ИТС-12
Чупринский Максим Валерьевич
Проверил: Заведующий кафедрой ИСиТ
Казаков Вадим Евгеньевич

Витебск, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данного курсового проекта является разработка REST-сервиса "Система управления программными проектами" с использованием технологий веб-фреймворка Flask. Сервис должен предоставлять API для управления программными проектами.

В настоящее время IT-сфера развивается с огромной скоростью и эффективное управление проектами становится фактором успеха. Разработка REST-сервиса позволит создать гибкую и масштабируемую систему, упрощающую ведение управленческой деятельности.

Веб-фреймворк Flask выбран для разработки серверной части приложения из-за его масштабируемости и легковесности.

В результате выполнения проекта ожидается получить полноценный REST-сервис, способный управлять работниками и программными проектами, обеспечивая операции добавления, удаления и редактирования данных.

Разработка REST-сервиса "Система управления программными проектами" изучить процесс разработки с использованием веб-фреймворка Flask, а получить практический опыт в написании веб-сервисов.

1.Описание предметной области

1.1 Назначение и область применения программного продукта

Учитывая растущую популярность информационных технологий и всё более высокие требования к качеству разработки программного обеспечения, знание и использование систем управления программными продуктами является важным и востребованным в современном мире. Такая система позволяет хранить информацию о работниках и их проектах, а также легко контролировать суммарную статистику.

1.2. Источники входной информации

Для разработки REST-сервиса "Система управлен программными продуктами" были использованы следующие источники входной информации:

1. Бизнес-требования и потребности пользователей: Проведена работа по определению бизнес-целей проекта и требований к функциональности. Основной акцент был сделан на удобстве использования, надежности и масштабируемости сервиса.

2. Анализ существующих решений и конкурентов: Проведен обзор существующих телефонных систем и аналогичных REST-сервисов для выявления особенностей функциональности и лучших практик. Это помогло определить ключевые особенности, которые следует учесть при разработке нашего сервиса.

Эти источники информации были ключевыми для определения функциональных требований и обеспечения соответствия сервиса потребностям пользователей и бизнес-целям проекта.

1.3. Выходная информация

Разработанный REST-сервис "Система управления программными продуктами" предоставляет следующую выходную информацию:

Работающий API. Сервис предоставляет RESTful API, который позволяет выполнять различные операции над работниками и проектами. API обеспечивает стандартные методы HTTP для создания (POST), чтения (GET), обновления (PUT) и удаления (DELETE) данных.

Ответы на запросы. При обращении к API сервис возвращает структурированные данные в формате JSON. Это позволяет клиентским приложениям легко обрабатывать ответы и взаимодействовать с сервисом.

Документация API. Для удобства использования сервиса разработана документация API, описывающая доступные эндпоинты, параметры запросов, ожидаемые ответы и примеры использования. Документация создана с использованием утилиты curl.

1.4. Требования к программному продукту

Для успешного развертывания и функционирования REST-сервиса "Система управления программными продуктами" необходимо учитывать следующие требования к конфигурации электронно-вычислительных средств:

1. Операционная система: Сервис разработан для работы на различных операционных системах семейств Microsoft Windows и Linux. Для работы необходим Python версии 3.6 и старше и установленные библиотеки Flask, Flask_Cors.

2. Хостинг и облачные сервисы: Сервис может быть развернут как на собственных серверах, так и в облачных сервисах, таких как Amazon Web Services (AWS), Microsoft Azure или Google Cloud Platform. Требуемые характеристики сервера зависят от ожидаемой нагрузки и объема данных, но рекомендуется использовать выделенные или виртуальные серверы с достаточным объемом оперативной памяти и процессорной мощности.

2. **Описание маппинга**

В ходе разработки программного средства для управления программными продуктами модели для описания таблиц в базе данных.

Всего в проекте 2 модели, каждая из них описывает свою таблицу в базе данных:

1. Модель Worker:

Эта модель описывает таблицу Worker в базе данных, которая хранит информацию о работниках.

В таблице Worker следующие поля:

- Id: идентификатор работника (primary key),
- Name: ФИО работника,
- Speciality: специальность работника,
- Experience: опыт работы по специальности (в годах).

2. Модель Project:

Эта модель описывает таблицу Project в базе данных, которая содержит информацию о проектах.

В таблице Project следующие поля:

- Id: идентификатор проекта (primary key),
- WId: идентификатор работника, за которым закреплен проект (foreign key),
- Description: описание проекта,
- Stage: стадия проекта.

3.Описание точек доступа

3.1. Точки доступа для управления данными о работниках

3.1.1 Получение списка всех работников

Возвращает список, содержащий данные всех работников.

GET /workers

Ответы:

- 200 – успешный ответ.

3.1.2 Получение работника по идентификатору

Возвращает конкретного работника по его идентификатору.

GET /workers/[id]

Параметры:

- id – идентификатор работника (int, required).

Ответы:

- 200 – успешный ответ;
- 400 – недопустимый идентификатор.

3.1.3 Создание нового работника

Создает нового работника.

POST /workers/add/[fio]/[specialty]/[experience]

Параметры:

- fio – ФИО работника (string, required, unique);
- specialty – специальность работника (string, required, unique);
- experience – стаж работника (float)

Ответы:

- 201 – успешный ответ;
- 400 – неверные параметры.

3.1.4 Изменение информации о работнике

Изменяет информацию о существующем работнике
PUT /workers/update/[id]/[fio]/[specialty]/[experience]
Параметры:

- id – идентификатор работника (int, required);
- fio – ФИО работника (string, unique);
- specialty – специальность работника (string);
- experience – стаж работника (float) (больше ранее указанного).

Ответы:

- 205 – успешный ответ;
- 400 – неверные параметры.

3.1.5 Удаление работника

Удаляет существующего работника.
POST /workers/delete/[id]
Параметры:

- id – идентификатор работника (int, required).

Ответы:

- 205 – успешный ответ;
- 400 – недопустимый идентификатор.

3.1.6 Получение статистики

Возвращает статистику выполненных, проваленных, разрабатываемых проектов для каждого работника.

GET /workers/stats

Ответы:

- 200 – успешный ответ.

3.2 Описание точек доступа для управления данными о проектах

3.2.1 Получение списка всех проектов

Возвращает список, содержащий данные всех проектов.

GET /projects

Ответы:

- 200 – успешный ответ.

3.2.2 Получение проекта по идентификатору

Возвращает конкретный проект по его идентификатору.

GET /projects/[id]

Параметры:

- id – идентификатор проекта (int, required).

Ответы:

- 200 – успешный ответ;
- 400 – недопустимый идентификатор.

3.2.3 Перевод проекта на следующую стадию

Переводит проект на следующую стадию, если он не находится на последней стадии и не отмечен как проваленный

POST /projects/next_stage/[id]

Параметры:

- id – идентификатор проекта (int, required).

Ответы:

- 205 – успешный ответ;
- 400 – недопустимый идентификатор.

3.2.4 Провал проекта

Отмечает проект как проваленный, если он не находится на последней стадии и не отмечен как проваленный

POST /projects/fail/[id]

Параметры:

- id – идентификатор проекта (int, required).

Ответы:

- 205 – успешный ответ;
- 400 – недопустимый идентификатор.

3.2.5 Создание нового проекта

Метод: POST

Создает новый проект.

POST /projects/add/[name]/[wid]/[description]/[stage]

Параметры:

- name – название проекта (string, required, unique);
- wid – идентификатор существующего работника (int, required);

- description – описание проекта (string);
- stage – стаж проекта (int) (0-7).

Ответы:

- 201 – успешный ответ;
- 400 – неверные параметры.

3.2.6 Изменение информации о проекте

Изменяет информацию о существующем проекте.

PUT /projects/update/[id]/[name] /[wid]/[description]/[stage]

Параметры:

- id – идентификатор проекта (int, required);
- name – название проекта (string, required, unique);
- wid – идентификатор существующего работника (int);
- description – описание проекта (string);
- stage – стаж проекта (int) (0-7).

Ответы:

- 205 – успешный ответ;
- 400 – неверные параметры.

3.2.7 Удаление проекта

Удаляет существующий проект.

POST /projects/delete/[id]

Параметры:

- id – идентификатор проекта (int, required).

Ответы:

- 200 – успешный ответ;
- 400 – недопустимый идентификатор.

3.2.8 Получение статистики

Возвращает суммарную статистику выполненных, проваленных и разрабатываемых проектов.

GET /projects/stats

Ответы:

- 200 – успешный ответ.

4.Разработка REST-сервиса

4.1 Структура проекта

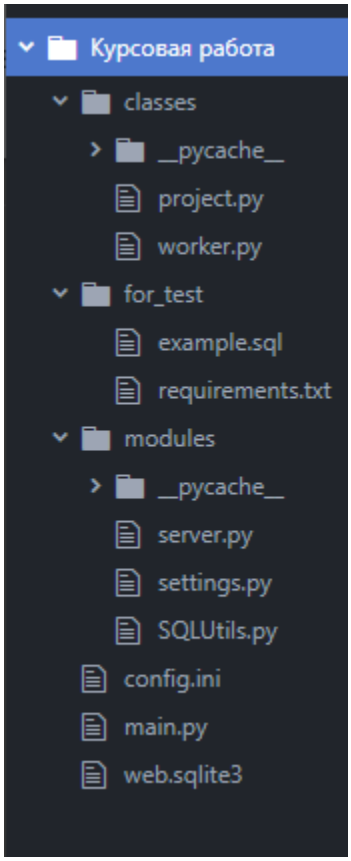

Рисунок 4.1 Структура проекта

В папке modules находятся различные модули программы (сервер, функции работы с базой данных).

В папке classes находятся файлы, которые содержат модели базы данных. Модели базы данных представляют собой описание структуры данных, которые хранятся в базе данных. Они используются для удобного обмена данными с базой.

Папка for_test содержит файлы для демонстрации и удобной настройки.

Файл config.ini является файлом настроек приложения.

Файл main.py – основная часть программы.

4.2 Запуск проекта

Для запуска проекта необходим установленный Python 3.6 или выше, а также библиотеки из файла for_test/requirements.txt. Также в папке for_test находится файл example.sql, который можно импортировать в пустую базу данных для удобной оценки возможностей приложения.

Задать порт сервера, путь к базе данных (sqlite3), префикс таблиц можно в файле config.ini.

Для запуска нужно выполнить файл main.py с помощью интерпритатора Python.После успешного запуска проекта вы увидите сообщение в консоли, которое указывает на то, что сервер запущен и слушает указанный порт. Так же можно выполнить запрос GET /, чтобы удостовериться в корректной работе сервера

4.3 Разработка контроллера

```
from os import path, chdir
from flask import url_for, request, Response, jsonify, make_response
from json import dumps
from classes.worker import Worker
from classes.project import Project
from modules.server import Server

chdir(path.dirname(path.realpath(__file__)))

app = Server()

@app.errorhandler(404)
def page_not_found(error):
    return 'Ошибка.\nУказанный маршрут не найден.', 404

@app.errorhandler(405)
def method_not_allowed(error):
    return 'Ошибка.\nМетод не зоответствует запросу.', 405


@app.route('/', methods=['GET'])
def index():
    return 'Сервер исправен и готов к работе!', 200
```

```
@app.route('/projects', methods=['GET'])
def project():
    p = app.sql.project_all()
    if p:
        return dumps(p, indent=4, ensure_ascii=False), 200
    return 'Пустота...', 200

@app.route('/projects/<int:id>', methods=['GET'])
def project_id(id):
    p = app.sql.project_by_id(id, json=True)
    if p:
        return dumps(p, indent=4, ensure_ascii=False), 200
    return 'Ошибка!\nУказанная запись не найдена.', 400

@app.route('/projects/stats', methods=['GET'])
def project_stats():
    projects = app.sql.project_all(json=False)
    if not projects:
        return {'projects': projects}, 200
    s = 0
    f = 0
    p = 0
    for i in projects:
        x = i.stat()
        if x == 1:
            s += 1
        elif x == -1:
            f += 1
        else:
            p += 1
    return {'projects': {'successed': s, 'failed': f, 'in_progess': p}}, 200

@app.route('/projects/add/<string:name>/<int:wid>/<string:desc>/<int:stage>', methods=['POST'])
@app.route('/projects/add/<string:name>/<int:wid>/<string:desc>', methods=['POST'])
@app.route('/projects/add/<string:name>/<int:wid>', methods=['POST'])
def project_add(name, wid, desc='', stage=1):
    if app.sql.project_add(Project(0, name, wid, desc, stage)):
        return 'Проект добавлен!', 201
    return 'Ошибка!\nПроверьте запрос.', 400

@app.route('/projects/update/<int:id>/<string:name>/<int:wid>/<string:desc>/<int:stage>',
methods=['PUT'])
@app.route('/projects/update/<int:id>/<string:name>/<int:wid>/<string:desc>', methods=['PUT'])
@app.route('/projects/update/<int:id>/<string:name>/<int:wid>', methods=['PUT'])
@app.route('/projects/update/<int:id>/<string:name>', methods=['PUT'])
def project_update(id, name=' ', wid=0, desc=' ', stage=-1):
    p0 = Project(id, name, wid, desc, stage)
```

```
    p = app.sql.project_by_id(id)
    if not p:
        return 'Ошибка!\nУказанная запись не найдена.', 400
    if p.merge(p0) and app.sql.project_update(p):
        return 'Проект обновлен!', 205
    return 'Ошибка!\nПроверьте запрос.', 400

@app.route('/projects/delete/<int:id>', methods=['POST'])
def project_delete(id):
    if app.sql.project_by_id(id) and app.sql.project_delete(id):
        return 'Проект удален!', 205
    return 'Ошибка!\nПроверьте запрос.', 400

@app.route('/projects/next_stage/<int:id>', methods=['PUT'])
def project_next_stage(id):
    p = app.sql.project_by_id(id)
    if not p:
        return 'Ошибка!\nУказанная запись не найдена.', 400
    if not p.next_stage():
        return 'Ошибка!\nУказанный проект невозможно перевести на следующую стадию.', 400
    if app.sql.project_update(p):
        return 'Проект переведен на следующую стадию!', 205
    return 'Ошибка!\nПроверьте запрос.', 400

@app.route('/projects/fail/<int:id>', methods=['PUT'])
def project_fail(id):
    p = app.sql.project_by_id(id)
    if not p:
        return 'Ошибка!\nУказанная запись не найдена.', 400
    if not p.fail():
        return 'Ошибка!\nУказанный проект закрыт.', 400
    if app.sql.project_update(p):
        return 'Проект отмечен как проваленный!', 205
    return 'Ошибка!\nПроверьте запрос.', 400


@app.route('/workers', methods=['GET'])
def worker():
    p = app.sql.worker_all()
    if p:
        return dumps(p, indent=4, ensure_ascii=False), 200
    return 'Пустота...', 200

@app.route('/workers/stats', methods=['GET'])
def worker_stats():
    workers = app.sql.worker_all(json=False)
    if not workers:
```

```
        return {'workers': workers}, 200
    res = {'workers': {}}
    for w in workers:
        res['workers'][w.id] = {'successed': 0, 'failed': 0, 'in_progress': 0}
        projects = app.sql.project_by_wid(w.id)
        if projects:
            for p in projects:
                x = p.stat()
                if x == 1:
                    res['workers'][w.id]['successed'] += 1
                elif x == -1:
                    res['workers'][w.id]['failed'] += 1
                else:
                    res['workers'][w.id]['in_progress'] += 1
    return res, 200

@app.route('/workers/<int:id>', methods=['GET'])
def worker_id(id):
    p = app.sql.worker_by_id(id, json=True)
    if p:
        return dumps(p, indent=4, ensure_ascii=False), 200
    return 'Ошибка!\nУказанная запись не найдена.', 400

@app.route('/workers/add/<string:name>/<string:speciality>/<string:experience>', methods=['POST'])
@app.route('/workers/add/<string:name>/<string:speciality>', methods=['POST'])
def worker_add(name, speciality, experience=''):
    if Worker.is_exp(experience) and app.sql.worker_add(Worker(0, name, speciality, experience)):
        return 'Работник добавлен!', 201
    return 'Ошибка!\nПроверьте запрос.', 400

@app.route('/workers/update/<int:id>/<string:name>/<string:speciality>/<string:experience>',
methods=['PUT'])
@app.route('/workers/update/<int:id>/<string:name>/<string:speciality>', methods=['PUT'])
@app.route('/workers/update/<int:id>/<string:name>', methods=['PUT'])
def worker_update(id, name=' ', speciality=' ', experience='0'):
    if Worker.is_exp(experience):
        w0 = Worker(id, name, speciality, experience)
        w = app.sql.worker_by_id(id)
        if not w:
            return 'Ошибка!\nУказанная запись не найдена.', 400
        if w.merge(w0) and app.sql.worker_update(w):
            return 'Работник обновлен!', 205
    return 'Ошибка!\nПроверьте запрос.', 400

@app.route('/workers/delete/<int:id>', methods=['POST'])
def worker_delete(id):
    if app.sql.worker_by_id(id) and app.sql.worker_delete(id):
```

```
        return 'Работник удален!', 205
    return 'Ошибка!\nПроверьте запрос.', 400


app.start()
```

Этот контроллер предоставляет методы для выполнения операций CRUD (создание, чтение, обновление, удаление) с объектами в базе данных. Он связывает соответствующие точки входа и методы работы с базой данных. Также он отвечает за запуск веб-сервера.

Точки входа и основные методы описаны выше (см. раздел 3.2). В случае успешного выполнения запроса будут возвращены коды: 200 (данные получены), 201 (данные добавлены), 205 (данные обновлены или удалены).

Возможны следующие коды ошибок: 400 (ошибка в запросе), 404 (точка входа не найдена), 405 (использован недопустимый метод).

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы был разработан REST-сервис "Система управления программными продуктами", предоставляющий API для управления данными о проектах. Для реализации данного сервиса были использованы современные технологии, такие как веб-фреймворк Flask и система управления базами данных sqlite3.

В процессе разработки были выполнены следующие этапы:

Изучение предметной области. Была проведена аналитическая работа по изучению требований и особенностей предметной области "Система управления программными продуктами".

Проектирование и моделирование. На основе полученных требований была разработана структура базы данных. Были определены основные сущности и их атрибуты, а также взаимосвязи между ними.

Реализация серверной части. Был разработан REST API, обеспечивающий CRUD операции (создание, чтение, обновление, удаление) с данными о работниках и проектах. Были созданы контроллеры для обработки запросов и модели для работы с базой данных.

Тестирование и отладка. После реализации функциональности было проведено тестирование API с использованием инструментов Swagger и curl. Были выявлены и устранены ошибки и недочеты в работе сервиса.

Документация. Была создана документация API, описывающая доступные эндпоинты, формат запросов и ответов.

В результате выполнения курсовой работы был создан функциональный REST-сервис "Система управления программными продуктами", который может быть использован для управления данными о программных продуктах.

ЛИТЕРАТУРА

1. https://flask.palletsprojects.com/en/3.0.x/tutorial/
2. https://flask.palletsprojects.com/en/3.0.x/tutorial/
3. https://habr.com/ru/articles/804245/
4. https://docs.python.org/3/library/sqlite3.html
5. https://www.sqlite.org/docs.html

ПРИЛОЖЕНИЕ 1

Репозиторий проекта

Проект размещен в репозитории на Github.

Прямая ссылка на репозиторий: https://github.com/amorator/Education-2/tree/main/Курсовая%20работа

QR-код:

Рисунок 1 QR-код репозитория