Міністерство освіти і науки України
Львівський національний університет ім. Івана Франка
Факультет електроніки та комп’ютерних технологій
Кафедра системного проектування

КУРСОВА РОБОТА
на тему: “Python REST-API Фреймворк ”

Виконав:
студент групи ФеІм-12с
Тупичак В. М.
Науковий керівник:
Огірко І. В.

Львів - 2023р.

Зміст

Вступ 3

Мета курсової роботи 4

Розділ 1. Завдання та вимоги фреймворку..... 4

1.1. Цілі та завдання фреймворку. 4

Цілі та завдання фреймворку: 4

1.2. Функціональні вимоги. 5

1.3. Нефункціональні вимоги. 5

Розділ 2. Технології використані в процесі розробки. 5

Python..... 5

Flask web-framework..... 6

Розділ 3 Архітектура фреймоврку..... 7

3.1. Опис Фреймоврку 7

3.2. Архітектура Фреймоврку 9

3.2 Структура проекту 12

3.3 Функціонал фреймворку.....17

3.4 Користь фреймворку.....**Error! Bookmark not defined.**

Висновок.....**Error! Bookmark not defined.**

Список використаних джерел: .....**Error! Bookmark not defined.**

Вступ

Сьогодні даних доступних онлайн більше, ніж будь-коли раніше. Це включає відкриті урядові дані, соціальні опитування, наукові дослідження, звіти приватних компаній та інші типи структурованих даних. Велика кількість інформації кардинально змінила сучасний світ і ми спостерігаємо цей вплив в повсякденному житті і сприймаємо це як належне. Однак публікація даних часто є далеко не легкою справою. Для прикладу у випадку з відкритими державними джерелами інформація, як правило, не публікується в централізованому сховищі а розкидана по кількох сайтах, не завжди представлена у форматі який є легким для читання комп'ютером і не відповідає сучасним вимогам для публікації даних.

Зазвичай процес відображення даних в інтернеті вимагає використання багатьох сучасних технологій таких як Django, Spring, FastAPI, Flask. Для використання цих технологій необхідно витратити багато часу на вивчення специфіки фреймоврку, мігрування даних та вручну визначати веб маршрути за допомогою яких користувачі будуть отримувати доступ до даних. Крім того необіхідно створити серверну частину, яка буде опрацьовувати кожен запит. Цей процес часто є важким, нудним та рутинним і, внаслідок цього, призводить до великої кількості помилок.

Іншими варіантами відображення даних є використання альтернативних баз даних, з вже збудованим функціоналом, наприклад, ArangoDB або Apache CouchDB. Однак, вони також мають власний набір проблем. Наприклад: ArangoDB має власну мову запитів, і для того щоб здійснювати базові операції користувачу потрібен час, щоб ознайомитися з нею. CouchDB фокусується на забезпеченні високої доступності системи та вимагає у користувача навичок роботи з реплікацією даних.

Мета курсової роботи

З урахуванням вище перелічених проблем я створив веб-фреймворк, щоб максимально полегшити процес розміщення структурованих даних в інтернеті. Метою фреймворку є:

- автоматичне створення та розгортання ендпоінтів для відображення даних які зберігаються в реляційних базах даних;
- Слідування принципам REST (Representational State Transfer)
- розгортати веб-додаток поряд з API, щоб спростити співпрацю між фронтендом та бекендом та забезпечити єдиний сервіс для інженерів даних та розробників.
- Бути гнучким для використання у широкому спектрі галузей

Розділ 1. Завдання та вимоги фреймворку.

1.1. Цілі та завдання фреймворку.

Цілі та завдання фреймворку:

1. Реалізувати для користувачів можливості швидкого відображення інформації з бази даних

2. Забезпечити можливість швидкого та легкого редагування проекту

3. Продумати цікаву інтерактивну складову для залучення користувачів.

1.2. Функціональні вимоги.

Фреймворк розраховано для користування людьми без серйозних технічних навичок.

1. Користувачам потрібно швидко та легко опублікувати дані
2. Конфігурація має бути зрозумілою
3. Фреймворк має працювати швидко
4. Фреймворк має забезпечувати реалізацію стандартної аутендифікації.

1.3. Нефункціональні вимоги.

1. Чітка організація інформаційних блоків.
2. Мова cli інтерфейсу - англійська.
3. Зручний та зрозумілий cli інтерфейс.
4. Зручність експлуатації.

Розділ 2. Технології використані в процесі розробки.

Python

Python - це високорівнева, інтерпретована мова програмування загального призначення, яка здатна вирішувати різноманітні завдання. Основні риси Python, що визначають її популярність серед програмістів, включають простоту вивчення, зрозумілість синтаксису та багатий екосистему інструментів і бібліотек. Одна з найбільш очевидних переваг Python - це легкість вивчення і розуміння коду. Ще однією перевагою Python є його спрощена система керування пам'яттю. Програмісти можуть зосередитися на розробці програми, а не на деталях низькорівневого керування пам'яттю, оскільки Python автоматично вирішує багато проблем, пов'язаних з управлінням пам’яттю.

Flask web-framework

Flask Flask є легковаговим веб-фреймворком для мови програмування Python, призначеним для швидкої розробки веб-додатків. Він надає необхідний мінімум інструментів і має простий синтаксис, що робить його дуже доступним для початківців, але також дозволяє розширити його функціональність для розробки складних веб-додатків.

Основні риси Flask:

- Легкість використання: Flask має простий і зрозумілий синтаксис, що дозволяє швидко вивчити фреймворк і почати розробляти веб-додатки.
- Гнучкість: Flask не накладає жорсткі правила архітектури, що дозволяє розробникам використовувати власні підходи і структури для своїх проектів.
- Розширюваність: Flask має розширену екосистему розширень (extensions) і бібліотек, що дозволяє розробникам додавати нові функціональні можливості до своїх додатків.
- Інтеграція: Flask легко інтегрується з іншими популярними бібліотеками та інструментами Python, такими як SQLAlchemy або Jinja2 для шаблонізації.
- Маштабованість: Хоча Flask починається з основних можливостей, він може бути розширений для розробки складних, масштабованих додатків. Наприклад, використовуючи розширення Flask-RESTful, можна легко створювати RESTful API.

Розділ 3 Архітектура фреймоврку

3.1. Опис Фреймоврку

Створений фреймворк є спрямований на швидке розгортання RESTful API над традиційною реляційною базою даних і автоматичне генерування серії тестів для цього API. Для досягнення цієї мети, відбувається підключається до користувацької бази даних та зчитує вказану схему бази даних, автоматично генеруючи:

- REST API з набором кінцевих точок, які реалізують базові операції CRUD (створення/читання/оновлення/видалення) для ресурсів у базі даних;
- набір модулів JavaScript для програмного доступу та зміни цих ресурсів за допомогою попередньо згенерованих кінцевих точок;
- набір тестів для перевірки того, що операції зі створеного API працюють належним чином.

Це дозволяє користувачам надзвичайно швидко публікувати дані онлайн і дозволяє їм створювати та використовувати сучасні, добре протестовані API для цих цілей.

Кожен API, створений за допомогою фреймоврку, називається проектом. Проект містить всі необхідні файли для автономності та простоти розгортання, зокрема: один або кілька JSON-файлів для визначення кінцевих точок API, за потреби - набір статичних файлів для веб-додатку, один або кілька SQL-файлів для ініціалізації схеми бази даних, серію тестів для API та, файл налаштувань, який контролює багато аспектів проекту, наприклад, дані для аутентифікації та IP-адресу сервера бази даних, або чи можуть користувачі входити та/або реєструватися у додатку. Новий проект можна легко створити за допомогою однієї з доступних консольних

команд фреймворку, а користувач може вибрати один з кількох шаблонів для свого нового проекту.

3.2. Архітектура Фреймоврку

На **рис.3** зображена архітектура фреймоврку

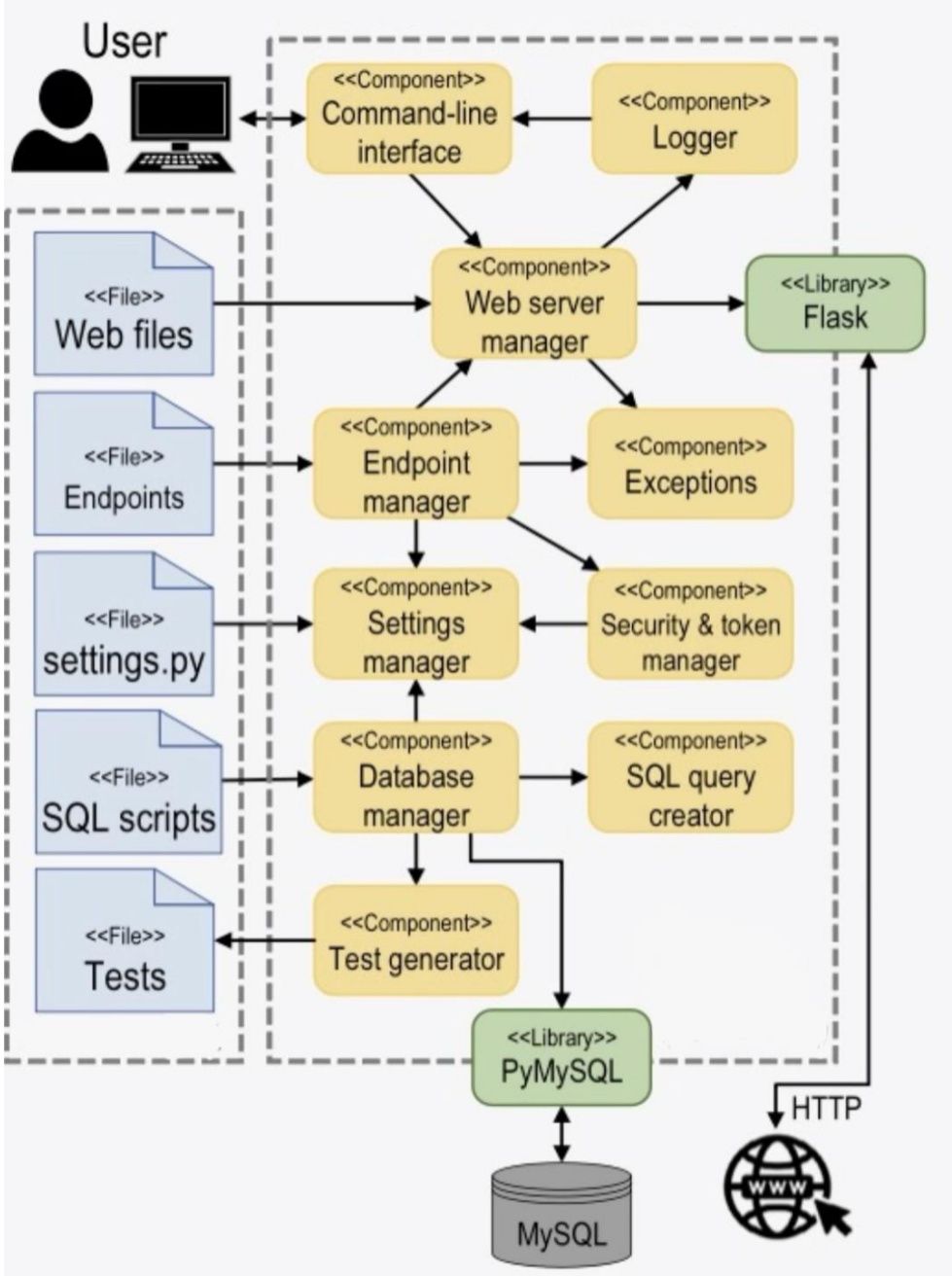


Рис.3

Фреймворк був побудований використовуючи компонентну архітектуру як показазано на рисунку 3. Кожен з його компонентів надає певні функціональні

можливості, має доступ до певних файлів у проекті і взаємодіє з іншими компонентами через інтерфейси.

Короткий опис інтерфейсів:

- Command-line interface: Надає користувачу набір консольних команд для створення та керування проектами.
- Logger: Стандартизує спільний формат виведення у командному рядку і надає іншим компонентам спільний інтерфейс для логування.
- Endpoint Manager: Зчитує користувацькі JSON-файли для завантаження власних ендпоінтів і визначає стандартні ендпоінти для входу та реєстрації.
- Web Server Manager: Реєструє як користувацькі, так і типові кінцеві точки у веб-сервері та налаштовує інші аспекти, такі як порт HTTP, використовуючи Flask як веб-сервер
- Exceptions: Надає набір власних винятків для використання всередині фреймворку для обробки помилок.
- Settings Manager: Зчитує параметри налаштувань проекту і надає значення за замовчуванням, якщо параметр не вказано.
- Security and Token Manager: Генерує і перевіряє токени, які використовуються для управління сесіями
- Data Base Manager: Підключається до вказаної бази даних у налаштуваннях та надає високорівневий інтерфейс для неї.
- SQL query generator: Надає інтерфейс для динамічного генерації SQL-запитів.
- Test generator: Генерує набір CRUD-тестів для API проекту, використовуючи схему бази даних, вказану в цьому проекті.

Ортогональний, компонентно-орієнтований дизайн фреймворку спрощує внесення змін або вдосконалень. Оскільки кожен компонент має чітко визначену відповідальність і загальна зв’язність компонентів низька, зміни в компоненті мають дуже низький вплив на інші компоненти в рамках фреймворку.

3.2 Структура проекту

На **рис.4** зображена структура проекту.

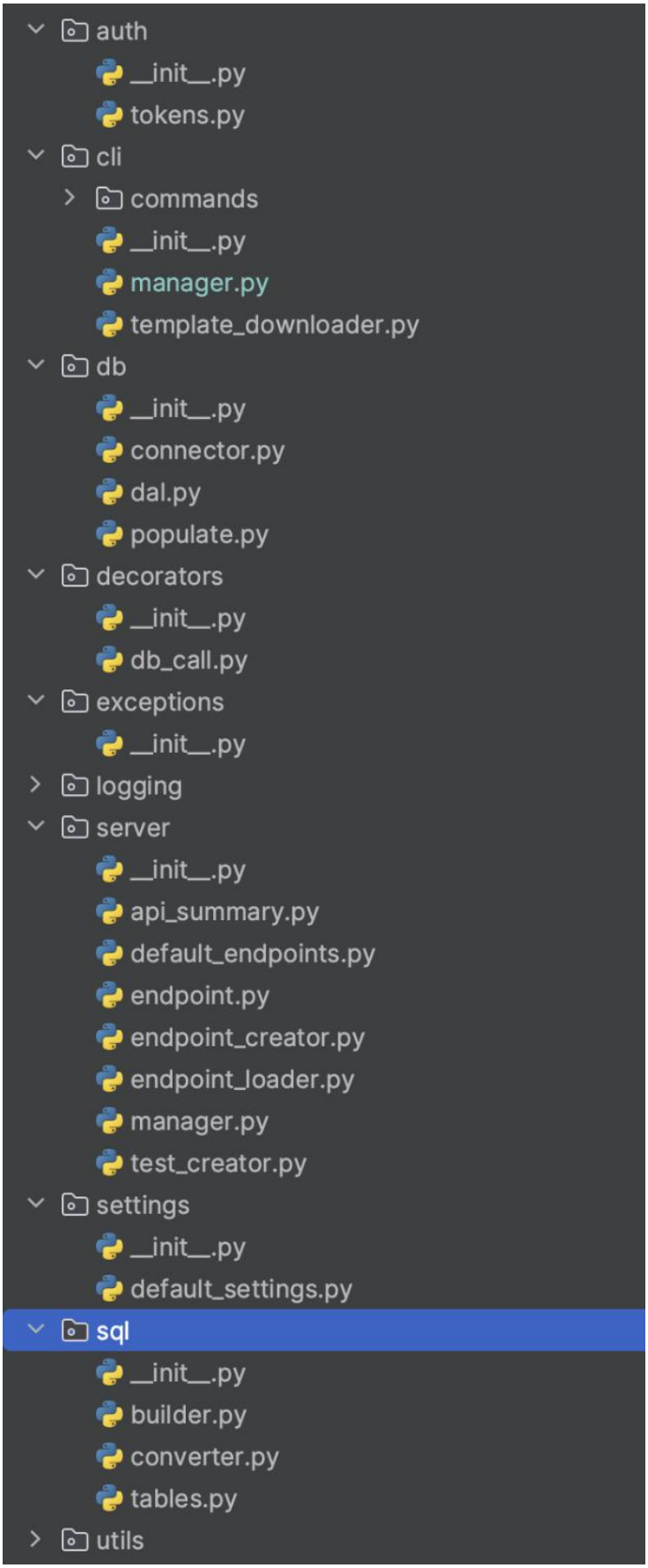


Рис. 4

Auth**:**

tokens.py - файл для створення токенів для аутендифікації

cli**:**

commands - директорія з доступними командами

manager.py - файл з з функціоналом для роботи з терміналом

template_downloader.py - файл для скачування прикладів проекту

db:

connector.py - файл з функціоналом для підключення до бд

dal.py - DAL (Data Access Layer) функцілонал для роботи з бд

populate.py - функціонал для роботи з SQL скріптами для створення та заповнення бд

decorators:

db_call.py - обгортка навколо запитів та оновлення бази даних для перехоплення можливих помилок

server:

api_summary.py - функціонал для виведення списку створених ендпоінтів

default_endpoints.py - функціонал для створення стандартних ендпоінтів

endpoint_creator.py - функціонал який використовує моделі бази даних для створення CRUD операцій над моделями г енеруючи json файл

endpoint_loader.py - функціонал для читання json файлів з інструкціями для створення ендпоінтів і створює їх

manager.py - менеджер серверу який створює Flask веб сервер, конфігурує його та розгортає ендпоінти і веб застосунок

test_creator.py - функціонал який працює з об’єктами бази даних та створює CRUD тести для них

settings:

default_settings.py - стандартні налаштування які можуть бути оновлені користувачами

sql:

всі файли тут є стандартним набором sql команд для роботи фреймоврку

requirements.txt - містить список необхідних бібліотек для запуску проекту.

READM.md - інструкція по запуску та підтримки проекту.

3.3 Функціонал фреймворку

Загальний порядок дій при використанні фреймворку зображено на **рис.5.**

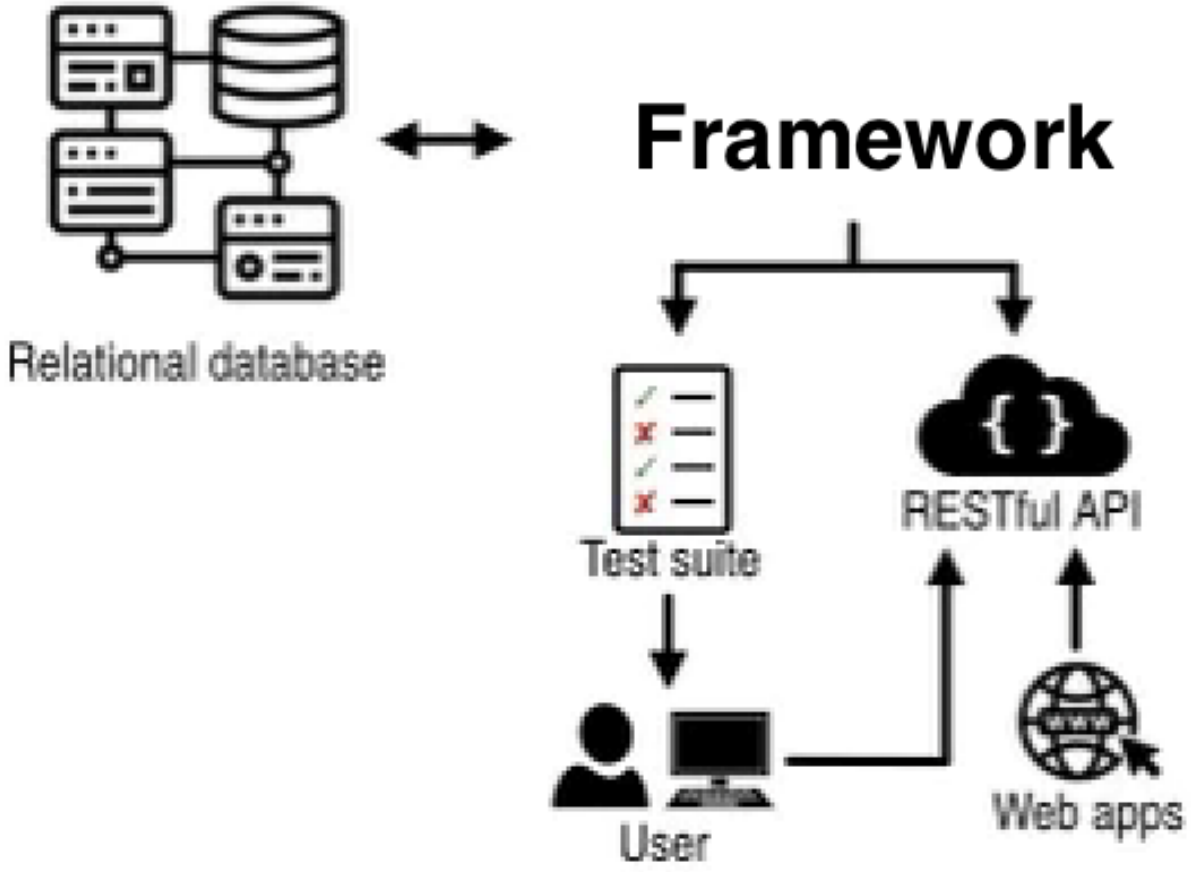


Рис.5

А виглядає він наступним чином:

Для початку користувач може створити новий проект із заданою назвою використовуючи команди **cli/manager.py new** <ім’я проекту>. Кожен проект пов'язаний з схемою бази даних і буде містити API для даних у цій схемі.

Користувач також може додати один з аргументів --template <ім'я_шаблону>, щоб використовувати певний шаблон проекту для нового проекту, --url <url_репозиторію> для його завантаження з репозиторію GitHub або --blank, що створює порожній проект без типових даних.

Список доступних шаблонів можна отримати за допомогою команди silence list-templates. Він отримується зі списку репозиторіїв нашої організації на GitHub, щоб команда завжди відображала найновішу інформацію, а список оновлювався динамічно.

Після створення проекту користувач може використовувати команду **cli/manager.py** createdb для генерації схеми бази даних, за умови, що в папці sql/ проекту розміщено кілька SQL-файлів з визначенням такої схеми.

Після створення схеми бази даних, фреймворк може прочитати її та автоматично створити CRUD-точки доступу для RESTful API для кожного ресурсу в базі даних за допомогою команди **cli/manager.py** createapi.

Крім того, фреймворк може створити основну структуру тестів для згенерованого API, щоб переконатися, що обмеження, визначені в схемі, були дотримані, за допомогою команди **cli/manager.py** createtests.

Нарешті, проект можна запустити за допомогою команди **cli/manager.py** run. Команда запустить як API, так і статичні веб-файли на HTTP-порті та IP-адресі, налаштовані користувачем в налаштуваннях проекту, що дозволить зовнішнім користувачам використовувати їх.

Для видів малого бізнесу можуть використовувати гнучким розгортанням RESTful API над існуючими реляційними даними. Для прикладу візьмемо базу даних працівників та всіх можливих даних які можуть бути пов’язані з ними.

Для створення проекту в терміналі необхідно виконати команду:

```
❯ python3 silence/cli/manager.py new demo_project --template employees
Project "demo_project" has been created using the template 'employees'.
```

Рис.6

python3 /cli/manager.py new demo_project. Результат виконання нижче спостерігаємо на рис.6 :

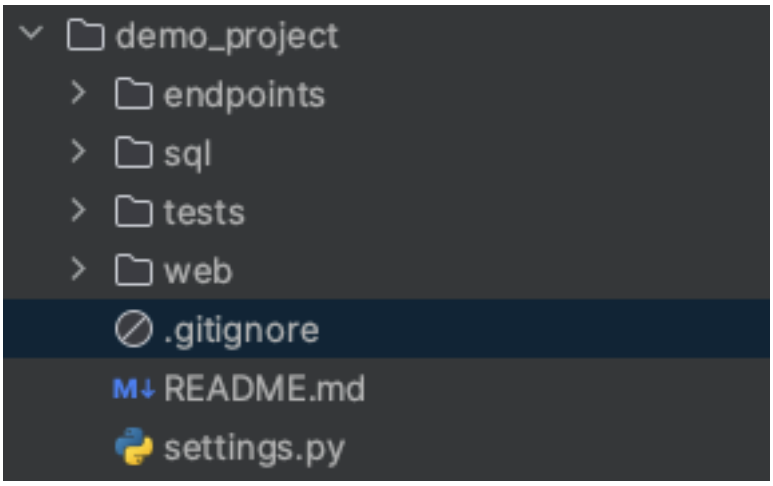


Рис.7

Після виконання цієї команди проект з вказаним раніше ім’ям згенерований. Структура проекту на рис.7

Для запуску проекту спочатку потрібно налаштувати файл settings.py в проекті для використання потрібної бази даних. Деякі інші параметри конфігурації включають HTTP-порт для використання, загальний префікс для всіх маршрутів API та режим налагодження, та інші. Приклад конфігурації в файлі settings.py для цього прикладу проекту показано на рис. 8

```python
# Project settings

DEBUG_ENABLED = False

DB_CONN = {
    "host": "127.0.0.1",
    "port": 3306,
    "username": "iissi_user",
    "password": "iissi$user",
    "database": "employees",
}

SQL_SCRIPTS = [
    "create_tables.sql",
    # "create_views.sql",
    "populate_database.sql",
]

HTTP_PORT = 8080

API_PREFIX = "/api/v1"

USER_AUTH_DATA = {
    "table": "Employees",
    "identifier": "email",
    "password": "password",
}

SECRET_KEY = "MG5VAjurbPEg30IlV-9n7PMM7FzINnumY7IhQa-X6XQ"
```

Рис.8

Після створення бази даних з SQL-файлів (рис. 9) у проекті за допомогою команди createdb, можна автоматично згенерувати точки доступу для API проекту використовуючи команду createapi. Це створить серію JSON-файлів у папці endpoints/auto/ (рис. 10-11), по одному для кожної таблиці у базі даних. Всі точки доступу у папці auto/ повинні бути зміненими або видаленими

кожного разу, коли користувач виконує команду createapi, тому користувачі повинні створювати свої власні настроювані точки доступу в батьківській папці endpoints/. Кожен JSON-файл може визначати одну або кілька точок доступу, вимагаючи принаймні шляху, HTTP-методу та пов'язаного SQL-запиту для кожної з них.

Точки доступу можуть підтримувати будь-яку бажану HTTP операцію і пов'язані з ними SQL-запити, результати яких будуть серіалізовані до JSON формату і відправлені назад клієнту. Запит може містити будь-яку кількість параметрів, які можуть бути отримані з маршруту точки доступу або надіслані користувачем у тілі запиту. Крім того, фреймворк автоматично надає сортування, обмеження, пагінацію та фільтрацію

```sql
DROP TABLE IF EXISTS Employees;
DROP TABLE IF EXISTS Departments;

CREATE TABLE Departments (
  departmentId INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(128) DEFAULT NULL,
  city VARCHAR(64) DEFAULT NULL,
  UNIQUE (name, city)
);

CREATE TABLE Employees (
  employeeId INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  email VARCHAR(256) UNIQUE NOT NULL,
  password VARCHAR(512) NOT NULL,
  departmentId INT DEFAULT NULL,
  bossId INT DEFAULT NULL,
  firstName VARCHAR(256) NOT NULL,
  lastName VARCHAR(256) NOT NULL,
  salary DOUBLE DEFAULT 2000,
  FOREIGN KEY (departmentId) REFERENCES Departments (departmentId) ON DELETE RESTRICT,
  FOREIGN KEY (bossId) REFERENCES Employees (employeeId),
  CONSTRAINT validSalary CHECK (salary > 0)
);
```

Рис.9

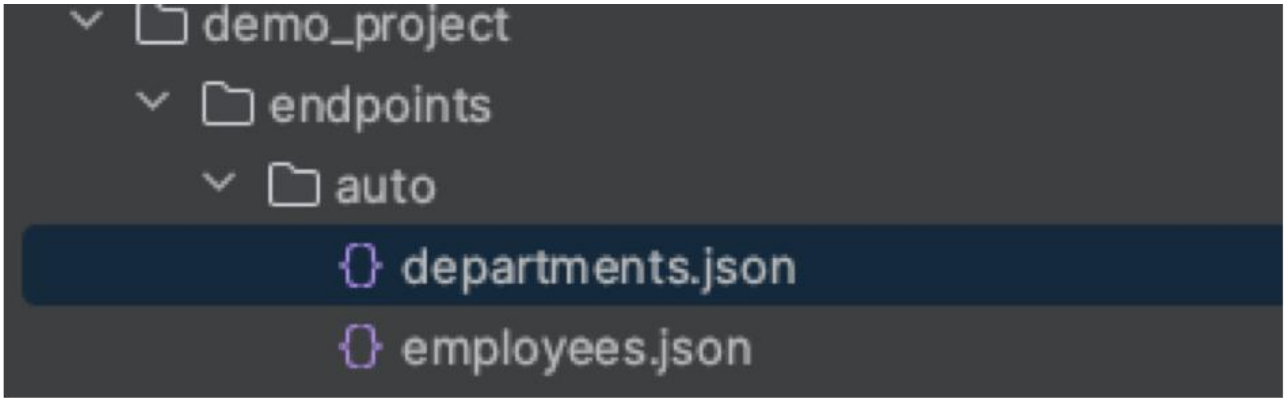
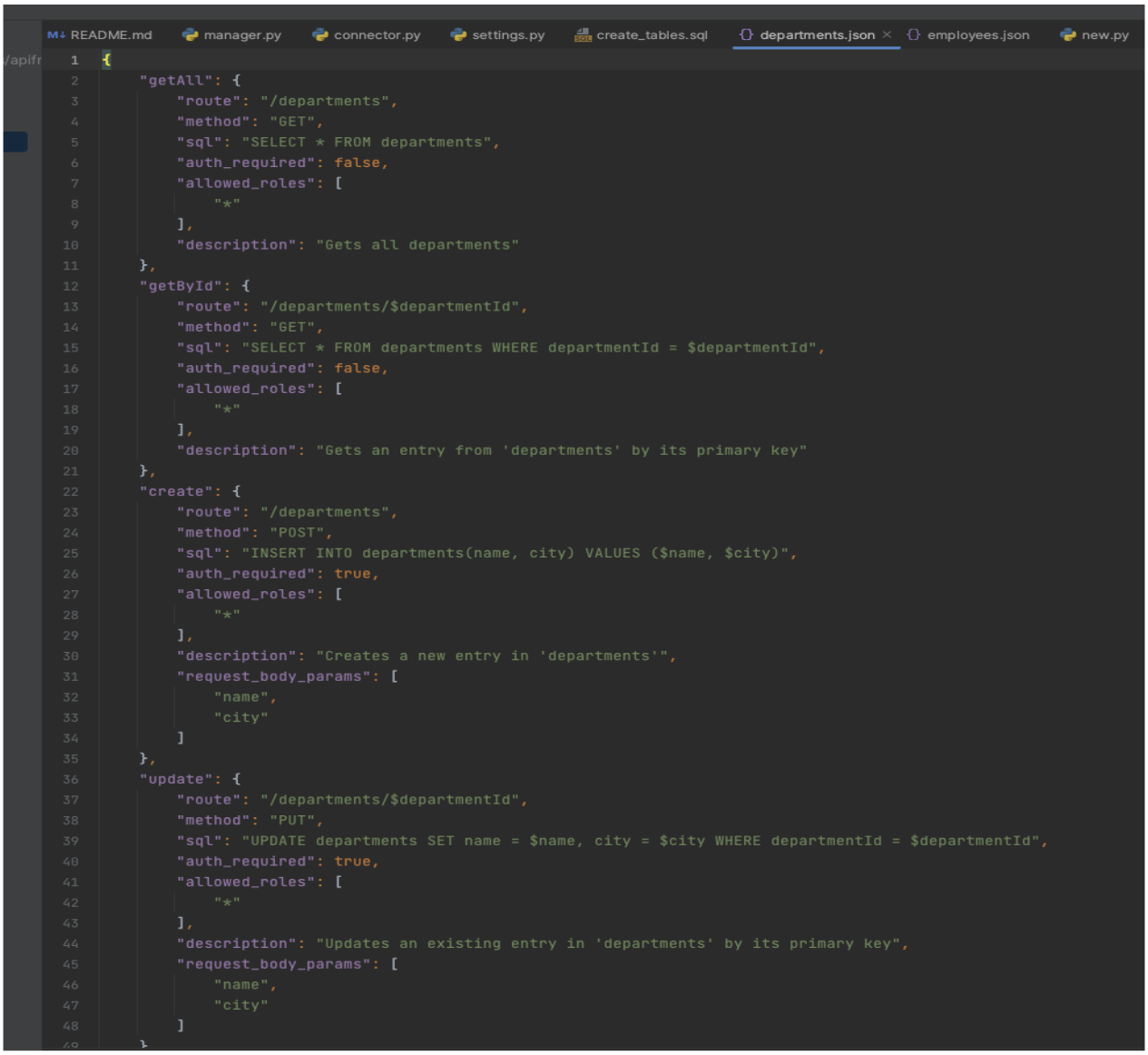


Рис. 10




```
{
    "getAll": {
        "route": "/departments",
        "method": "GET",
        "sql": "SELECT * FROM departments",
        "auth_required": false,
        "allowed_roles": [
            "*"
        ],
        "description": "Gets all departments"
    },
    "getById": {
        "route": "/departments/$departmentId",
        "method": "GET",
        "sql": "SELECT * FROM departments WHERE departmentId = $departmentId",
        "auth_required": false,
        "allowed_roles": [
            "*"
        ],
        "description": "Gets an entry from 'departments' by its primary key"
    },
    "create": {
        "route": "/departments",
        "method": "POST",
        "sql": "INSERT INTO departments(name, city) VALUES ($name, $city)",
        "auth_required": true,
        "allowed_roles": [
            "*"
        ],
        "description": "Creates a new entry in 'departments'",
        "request_body_params": [
            "name",
            "city"
        ]
    },
    "update": {
        "route": "/departments/$departmentId",
        "method": "PUT",
        "sql": "UPDATE departments SET name = $name, city = $city WHERE departmentId = $departmentId",
        "auth_required": true,
        "allowed_roles": [
            "*"
        ],
        "description": "Updates an existing entry in 'departments' by its primary key",
        "request_body_params": [
            "name",
            "city"
        ]
    }
```

Рис. 11

Коли користувач дістається до однієї з точок доступу, виконується пов'язаний SQL-запит, і результати повертаються у форматі JSON. Точки доступу POST і PUT працюють подібним чином, беручи інформацію, надіслану користувачем у тілі запиту, та безпечно вставляючи її у рядок SQL, який буде виконуватись. Ось результати, які повертаються точкою доступу:

GET http://127.0.0.1:8080/api/v1/departments


```
[{
  "city": null,
  "departmentId": 2,
  "name": "Bussines"
 },{
  "city": "Lviv",
  "departmentId": 1,
  "name": "Developer"
 },{
  "city": "Drogobych",
  "departmentId": 3,
  "name": "Support"
 }]
```


Коли база даних налаштована, фреймворк може підключитись до неї і автоматично створити серію тестів для точок доступу кожної таблиці. Ці тести перевіряють функціональність основних CRUD-операцій і містять деякі поля,

які користувач повинен замінити вручну, наприклад, властивості нових елементів, які слід додати у таблицю. Тести складаються з одного .http-файлу для кожної таблиці у базі даних.

Нарешті, користувач може запустити проект за допомогою команди run. Фреймворк автоматично виявляє визначені точки доступу і розгортає їх разом з веб-додатком, як показано на рис. 12

```
Endpoints loaded:
    · http://127.0.0.1:8080/api/v1 (GET)
    · http://127.0.0.1:8080/api/v1/departments (GET/POST)
    · http://127.0.0.1:8080/api/v1/departments/<departmentId> (GET/PUT/DELETE)
    · http://127.0.0.1:8080/api/v1/employees (GET/POST)
    · http://127.0.0.1:8080/api/v1/employees/<employeeId> (GET/PUT/DELETE)
    · http://127.0.0.1:8080/api/v1/employees/current (GET)
    · http://127.0.0.1:8080/api/v1/login (POST)
    · http://127.0.0.1:8080/api/v1/register (POST)

Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
25/05/2023 00:15:09 | [API] GET /api/v1/employees/current from 127.0.0.1 - 401
25/05/2023 00:15:10 | [WEB] GET /favicon.ico from 127.0.0.1 - 404
```

Рис.12

Рис.13

Як можна бачити з рис.13 буквально за декілька хвилин у нас уже є працюючий веб ресурс який готовий опрацьовувати будь які запити користувача та відображувати вміст бази даних. Також вже реалізований функціонал для аутендифікації користувачів. Залишається тільки створити користувацький інтерфейс веб сторінки.

3.4 Користь фреймворку

Створений фреймворк зменшує розрив між генерацією даних та їх доступністю в Інтернеті, надаючи набір інструментів для автоматичного забезпечення доступу, виявлення та змінення структурованих даних. У цьому відношенні, фреймворк вимагає лише наявності Python, та вміння користуватись терміналом, що робить його дуже легким у користуванні для неспеціалістів, в порівнянні з іншими подібними фреймворками. Крім того, для використання фреймворку не потрібні попередні знання Python або будь-якої іншої мови програмування.

Також фреймворк має кілька стандартних функціональних можливостей, які допомагають прискорити процес створення онлайн-інтерфейсу для структурованих даних. Наприклад, вона надає користувачу два заздалегідь визначені кінцеві точки, /login та /register, що дозволяють користувачам увійти або зареєструватися, за бажанням цей функціонал може бути вимкненим за бажанням. Крім того, фреймворк підтримує обмеження доступу, які можуть вимагати аутентифікації для доступу до певної кінцевої точки або вимагати наявності певної ролі у користувачів.

Фреймворк був розроблений метою забезпечити простоту використання, але при цьому залишатись дуже гнучким для а різних можливих сценаріїв. На відміну від інших веб-фреймворків, які менш гнучкі, даний фреймворк потребує невелику кількість налаштувань для запуску API. Однак це не означає, що варіанти користувача обмежені. Навпаки, надається набір типових значень, які можуть бути перевизначені для кожного проекту або кожної кінцевої точки.

Особисто я неодноразово використовував його під час роботи. Оскільки часто дані були доступні реляційній базі даних, але для побудови API над нею завжди небхідно було витрачати багато часу та виконувати рутинну роботу.

Висновок

Підбивши підсумки розробки цього веб-фреймворку, можна ствердити, що було реалізовано всі поставлені цілі. Тобто цей фреймворк готовий для повноцінного користувацького використання. В ході розробки було створено веб фреймворк, метою якого є - дозволити користувачам швидко створювати бекенд для структурованих реляційних даних. Фреймворк здатний автоматично генерувати та розгортати RESTful API на основі інформації, доступних у базі даних, підтримуючи найпоширеніші операції CRUD та значно зменшуючи час та зусилля, необхідні для доступу до цих ресурсів в Інтернеті. Фреймворк легко встановлюється та використовується, але також надає багато можливостей та налаштуваннь для досвідчених користувачів. У майбутньому планується додати сумісність з іншими реляційними базами даних, такими як PostgreSQL та Oracle. Крім того, зробити Silence сумісним з популярними тестовими фреймворками Python, щоб дозволити користувачам автоматично перевіряти правильність свого API та полегшити розгортання додатків, побудованих на його основі. Також планується опублікувати цей фреймворк як Python пакет. Окрім цього хочу відзначити, що застосунок був створений на основі сучасних технологій та відповідає усім архітектурним вимогам.

Отже, можу зробити висновок, що зроблено стабільний, надзвичайно корисний та швидкий продукт, який готовий до масового використання і за рахунок правильно побудованої архітектури можна легко розширювати та підтримувати.

Список використаних джерел:

1. Mastering Python Design Patterns, https://www.amazon.com/Mastering-Python-Design-Patterns-efficient/dp/1788837487)
2. Data Structures and Algorithms in Python, https://www.amazon.com/Structures-Algorithms-Python-Michael-Goodrich/dp/1118290275)
3. Attard J, Orlandi F, Scerri S, Auer S. A systematic review of open government data initiatives. Gov Inf Q 2015;32(4):399–418.
4. Django for Python. 2022, https://www.djangoproject.com
5. The spring framework. 2022, https://spring.io/
6. FastAPI framework. 2022, https://fastapi.tiangolo.com/
7. ArangoDB repository on GitHub. 2022, https://github.com/arangodb/arangodb
8. Apache CloudDB repository on GitHub. 2022, https://github.com/apache/couchdb
9. Toti D, Rinelli M. a system for automatically producing API layers from RDF schemas. 2017 https://link.springer.com/article/10.1007/s12652-016-0394-z
10. Component-based software engineering — new challenges in software development https://onlinelibrary.wiley.com/doi/abs/10.1002/swf.45
11. Flask for Python. 2020, https://flask.palletsprojects.com/en/2.1.x/

12. EST Client. 2022, https://marketplace.visualstudio.com/items?itemName=humao.rest-client