Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Розробка ПЗ для створення звіту про роклад у форматі MS Excel

Текстова частина до курсової роботи
За спеціальністю «Комп'ютерні науки і технології» 113

Керівник курсової роботи
к.н., с.в. Демківський Є. О.

_____
(підпис)
"____" _____ 2020 р.

Виконав студент 4-го курсу
"____" _____ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

доцент, к.ф.-м.н.

_____ С. С. Гороховський

(підпис)

“____”_____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту _____Хоменку Дмитру_____

__4-го__ курсу факультету інформатики

ТЕМА: Розробка ПЗ для створення звіту про роклад у форматі MS Excel

Вихідні дані:

- Застосунок для генерації розкладу в xlsx форматі за вхідним JSON

Зміст ТЧ до курсової роботи:

Вступ
1. Аналіз предметної області.
2. Порівняльна характеристика існуючих рішень
3. Опис реалізації програмного продукту
Висновки
Список джерел
Додатки (за необхідністю)

Дата видачі “___” _____ 2020 р.

Керівник _____ Завдання отримано _____

Зміст

Календарний план виконання курсової роботи

Тема: Публічні та приватні ключі в блокчейн. Засоби безпеки операцій

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Жовтень-листопад 2020р.	
2.	Огляд літератури за темою роботи	Листопад-грудень 2020р.	
3.	Аналіз предметної області	Січень 2021р.	
4.	Порівняльна характеристика існуючих рішень	Лютий 2021р.	
5.	Реалізація програмного забезпечення та тестування	Березень 2021р.	
6.	Написання текстової частини.	Квітень 2021р.	
7.	Перегляд курсової роботи науковим керівником	Квітень 2021р.	
8.	Захист курсової роботи	12.04.2020	
9.	Перегляд змісту роботи керівником	12.04. 2020	
10.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	12.04. 2020	
11.	Створення презентації	12.04.2020	

Студент Хоменко Д.. _____

Керівник Демківський Є.О. _____

“_____”_____ 2021 р.

Вступ

Ця робота має на меті огляд сучасних технологій розробки та перевірених часом рішень. Порівняння популярних архітектурних підходів та фрейморків, які можуть бути використані для розробки програмного забезпечення для створення звіту про розклад у форматі Office Open XML format по вхідному JSON.

Розробка веб-застосунку, реалізованого за допомогою мови програмування Python з використанням фреймворку Django.

▪ Аналіз предметної області

1.1 Огляд мови програмування Python

Python - це інтерпретована, інтерактивна та об’єктно-орієнтована мова високого рівня. Python розроблений таким чином, щоб він був високочитабельним. Він часто використовує англійські ключові слова, де як інші мови використовують розділові знаки, і він має менше синтаксичних конструкцій, ніж інші мови.

Мова, що задумана нідерландським дослідником – Гвідо ван Роуссумом наприкінці 1980-років, є нащадком мови програмування ABC, походить з багатьох інших мов програмування, таких як: Modula-3, C, C ++, Algol-68, SmallTalk. Python захищений авторським правом. Як і Perl, вихідний код Python тепер доступний під загальною публічною ліцензією GNU (GPL).[2]

Зараз Python підтримується основною командою розробників в інституті, хоча Гвідо ван Россум все ще відіграє важливу роль у керуванні його прогресом.

1.1.1 Основні переваги та недоліки Python:

- Легкий у засвоєнні, проста структура (мала кількість ключових слів) та чітко виражений синтаксис дозволяють швидко розібрати мову.
- Легкий в обслуговуванні
- Широка стандартна бібліотека
- Інтерактивний режим, що дозволяє інтерактивне тестування та налагодження фрагментів коду
- Портативний, може працювати на різних апаратних платформах і має однаковий вигляд на всіх платформах

- Розширюваний, оскільки, можна додавати модулі низького рівня до інтерпретатора Python. Вони дозволяють програмістам додавати або налаштовувати свої інструменти, щоб бути ефективнішими.
- Бази даних, Python забезпечує інтерфейси до всіх основних комерційних баз даних.
- Python підтримує графічні програми.
- Масштабованість - забезпечує підтримку великих програм, та широко використовується найбільшими IT компаніями.

Python також можна використовувати для написання функціонального коду, немає необхідності оголошувати типи змінних. Це мова, яка заповнює прогалини між бізнесом та розробниками. Для виведення на ринок програми на Python потрібно менше часу порівняно з іншими мовами, такими як C # / Java. Крім того, існує велика кількість пакетів для машинного навчання та аналітики на python. Для підтримки розробників Python доступна велика кількість спільнот та книг. Головними недоліками мови є те, що вона не підходить для розробки низькорівневих систем та апаратної взаємодії і його швидкість роботи (варто зазначити, відсутність оптимізаторів Just In Time). Але, останній з перелічених недоліків можна вирішити шляхом впровадження розширення на основі мови C.

1.1.2 Принцип роботи Python

- Створюється віртуальна машина Python, де встановлюються пакети (бібліотеки).
- Код python записується у файли .py
- CPython компілює код Python в байт-код. Цей байт-код призначений для віртуальної машини Python.
- Коли ви хочете виконати байт-код, тоді код буде інтерпретований під час виконання. Потім код буде перетворений з байт-коду в машинний код. Байт-код не залежить від машини, на якій ви використовуєте код. Це робить Python незалежним від машини.

1.1.3 Нові функції в останній версії Python 3.8

- Вираз присвоєння :=

Він також відомий як моржовий оператор. Присвоює значення змінним як частину виразу без необхідності ініціалізувати змінні заздалегідь.

```
if (my_variable := get_input()) is not None:

 print(my_variable) #exists now
 perform_action(my_variable);
```

Рисунок 1.1

У наведеному вище коді my_variable не існував до виконання першого рядка. Морж-оператор: = був використаний для оголошення та ініціалізації змінної “my_variable”. Потім повернення функції “get_input ()” використовується для присвоєння значення змінній. В результаті, тепер ми можемо надрукувати значення змінної my_variable.

2. Лише позиційні параметри ("/")

Якщо викликати функцію в Python, яка приймає параметри, можна передавати аргументи за позицією або за ключовим словом.

Якщо ми хочемо обмежити абонентів нашого API лише викликати нашу функцію, передаючи параметри за позицією, або, якщо імена наших параметрів не мають сенсу для зовнішнього світу або, можливо, ми плануємо перейменувати параметри в майбутньому -

Функція параметру "/", вирішує це.

```
def add(a, b, c, d=None, /):
    x = a+b+c
    if d is not None:
        x = x+d
    return x
```

Рисунок 1.2

Функція "add" приймає три обов’язкові параметри: a, b і c, а також необов’язковий параметр d. Останній параметр “/” вказує на те, що

параметри функції повинні бути задані позиційно. Як результат, ми не можемо викликати функцію, передаючи параметри за ключовим словом. add (1,2,3) та add (1,2,3,4) є дійсними дзвінками.
add (a = 1, b = 2, c = 3) або add (1,2,3, d = 4) - це недійсні виклики.
Результат параметра “/” вказує на те, що функція приймає лише позиційні параметри, і, отже, аргументи повинні бути зіставлені з параметрами, виходячи лише з їх порядку. За допомогою позиційних функцій, автори API можуть перейменовувати параметри функції (якщо потрібно), не порушуючи код викликаючих API. Отже, це заохочує API бути зворотно сумісним.

3. Для полегшення налагодження f-рядки тепер підтримують «=»

Це вдосконалення f-рядків. Специфікатор “=” тепер можна додати до f-рядків. f '{expr =}' Зверніть увагу на '='. Знак рівності по суті обчислює вираз і друкує результат.
Давайте розберемося в цьому на прикладі.
Є дві змінні “input” і “output”, і ми хочемо надрукувати “input-output” разом із результатом. Ми можемо використовувати f-strings =

```
input = 100
output = 50
print(f'{input-output=}')
```

Рисунок 1.3

Тепер буде надруковано ввід-вихід = 50.
Як результат, це може зробити код акуратнішим, оскільки, нам не потрібно копіювати формулу праворуч від “=”, щоб оцінити її, а побічним ефектом є те, що її можна широко використовувати під час налагодження.

1.2 REST API

REST пропонує створити об’єкт даних, що запитує клієнт і надіслати значення об’єкта у відповідь користувачеві. Наприклад, якщо користувач запитує фільм у Києві у певному місці у певний час, тоді ми можемо створити об’єкт на стороні сервера.

Отже, є об’єкт і ми надсилаємо стан об'єкта. Ось чому REST відомий як Representional State Transfer. REST – це архітектурний стиль, а також підхід для комунікаційних цілей, який часто використовується при розробці різних веб-служб. Архітектурний стиль REST допомагає використовувати менше пропускної здатності мережі, щоб зробити програму більш придатною для всесвітньої мережі. Його часто розглядають, як «мова Інтернету». [4]

Зануримось трохи глибше і подивимось, як саме працює REST API. В основному, REST API розбиває транзакцію з метою створення невеликих модулів. Тепер кожен із цих модулів використовується для адресування певної частини транзакції. Такий підхід забезпечує більшу гнучкість, але вимагає великих зусиль, щоб побудувати його з нуля.

1.1.4 Принципи REST API

Отже, зрозуміємо обмеження, які повинні бути виконані, щоб програма розглядалась як REST API. Є шість основних приципів, викладених доктором Філдінгом, який у 2000 році визначив дизайн REST API:

1. **Не залежить від стану (Stateless)**

Запити, надіслані від клієнта на сервер, будуть містити всю необхідну інформацію, щоб сервер зрозумів запити, надіслані від клієнта. Це може бути або частина URL-адреси, параметри рядка запиту, тіло або навіть заголовки. URL-адреса використовується для однозначної ідентифікації ресурсу, а тіло містить стан запитуваного ресурсу. Як тільки сервер

обробляє запит, відповідь надсилається клієнту через тіло, статус або заголовки.

2. Клієнт-Сервер

Архітектура клієнт-сервер забезпечує єдиний інтерфейс та відокремлює клієнтів від серверів. Це покращує портативність на декількох платформах, а також масштабованість серверних компонентів.

3. Уніфікований інтерфейс

Щоб отримати однорідність у застосунку, REST має наступні чотири обмеження інтерфейсу:

1) Ідентифікація ресурсу

Ресурс є ключовою абстракцією REST. Все, що можна назвати, - це ресурс - відео, документ, зображення. Ми ідентифікуємо ресурси за допомогою унікального ідентифікатора, який називається Uniform Resource Identifier (URI). При обговоренні Інтернету ми майже завжди використовуємо більш конкретну форму URI, яка називається Uniform Resource Locator (URL).

Створюючи REST API через HTTP, ми можемо використовувати URL-адреси для ідентифікації ресурсів, до яких отримує доступ наш API. Щоб задовольнити обмеження REST, кожна URL-адреса повинна зіставитись з одним ресурсом, і весь доступ до цього ресурсу здійснюється через цю URL-адресу. Як приклад, якщо я хочу запропонувати API для програми доставки, я можу мати ресурс, що представляє замовлення. URL-адреса для окремого замовлення під номером 12345 матиме шлях, що включає номер замовлення.

/ замовлення / 12345

Цей простий приклад показує, як ми можемо створити URL-адресу для унікального представлення кожного ресурсу, представленого нашим API.

2) Маніпулювання ресурсами за допомогою подань

Коли робить запит на ресурс, сервер відповідає поданням ресурсу. Це подання фіксує поточний стан ресурсу у форматі, який клієнт може зрозуміти та маніпулювати ним. Абстрактно представлення - це послідовність байтів разом з метаданими, що описують ці байти. Ці метадані відомі як медіа-тип представлення. Типовими прикладами API є HTML, JSON та XML. Оскільки сервер надсилає подання ресурсу, клієнт може запитувати конкретне подання, яке відповідає потребам клієнта. Наприклад, клієнт може попросити представлення JSON ресурсу або XML представлення ресурсу. Сервер може надати це подання, якщо він здатний це зробити. Ця концепція називається переговором про зміст. Ви можете використовувати узгодження вмісту у своєму API, щоб дозволити декільком клієнтам отримати доступ до різних подань ресурсу з тієї самої URL-адреси.

Клієнт запитує конкретне подання через заголовок HTTP Accepts. Наступний запит вимагає подання замовлення у простому тексті.

```
GET /orders/12345
Accept: text/plain
```

Рисунок 1.4

Тоді як наступний запит вимагає подання JSON

```
GET /orders/12345
Accept: application/json
```

Рисунок 1.5

Використовуючи узгодження вмісту у своєму API, ви можете пропонувати нові подання ресурсів, не змінюючи URL-адресу ресурсів або не порушуючи існуючих клієнтів. Це дозволяє гнучко розв’язати клієнта та сервер.

3) Самоописові повідомлення

Представлення, що обслуговуються системою RESTful, містять усі дані, необхідні клієнту для розуміння та дії ресурсу. Якщо потрібна будь-яка

додаткова інформація, але вона не міститься у відповіді, у відповіді слід вказати посилання на цю інформацію. Це означає, що тип носія, на який ви вибрали відповідь, повинен самодокументовуватися і містити перелік усіх пов'язаних ресурсів або дій, які можуть зацікавити клієнта.

4) Гіпермедіа як двигун стану застосування

Разом перші три обмеження REST передбачають четверте найважливішу - гіпермедіа як двигун стану програми. Завдяки унікальній ідентифікації ресурсів, використанню подань для передачі стану ресурсів та використанню самоописових повідомлень за допомогою типів носіїв, усі стани додатків залишаються у клієнта.

Зберігаючи весь стан програми з клієнтом, не потрібно прямого зв'язку між клієнтом і сервером, що дозволяє масштабуватись, щоб обслуговувати багатьох клієнтів з мінімальними ресурсами. Ця здатність масштабувати призвела до масштабованості та стійкості Мережі.

4. Можна кешувати (Cacheable)

Щоб забезпечити кращу продуктивність, програми часто робляться кешованими. Це робиться шляхом позначення відповіді сервера як кешованої чи некешованої або неявно, або явно. Якщо відповідь визначено як кешоване, тоді клієнтський кеш може повторно використовувати дані відповідей для еквівалентних відповідей у майбутньому.

5. Система шарів (Layered system)

Архітектура багатошарової системи дозволяє додатку бути більш стабільним, обмежуючи поведінку компонентів. Цей тип архітектури допомагає підвищити безпеку програми, оскільки, компоненти кожного шару не можуть взаємодіяти далі наступного шару, в якому вони перебувають. Крім того, він забезпечує збалансування навантаження та забезпечує спільні кеші для сприяння масштабованості.

6. Код на вимогу (Code on demand)

Це необов’язкове обмеження і використовується найменше. Це дозволяє завантажувати код клієнта та використовувати їх у програмі. По суті, це спрощує клієнтів, створюючи розумний додаток, який не покладається на власну структуру коду.

1.1.5 Методи REST API

Всі ми, хто працює з технологією Інтернет, виконуємо операції CRUD. Операції CRUD – створємо ресурс, читаємо ресурс, оновлюємо ресурс і видаляємо ресурс. Тепер, що виконати ці дії, можна використовувати методи HTTP, які є ні чим іншим, як методами REST API.

HTTP Method	CRUD	Description	Example
GET	Read/Retrieve	Getting a resource	Getting all recipes back http://localhost:5000/recipes
GET	Read /Retrieve	Getting a resource	Getting a recipe with ID = 20 http://localhost:5000/recipes/20
POST	Create	Creating a resource	Adding a recipe http://localhost:5000/recipes
PUT	Update	Updating a resource	Updating a recipe with ID = 20 http://localhost:5000/recipes/20
DELETE	Delete	Deleting a resource	Deleting a recipe with ID = 20 http://localhost:5000/recipes/20

Рисунок 1.6

- **Порівняльна характеристика існуючих рішень**

2.1 Порівняння мікросервісної та монолітної архітектури

Архітектура мікросервісів популярна сьогодні і використовується у Uber, Netflix, Linkedin та багатьма іншими компаніями, що хочуть мати можливість швидко щось змінити, щоб швидше реагувати на зміни вимог бізнесу, випередивши конкурентів. Мікросервіси допомагають розробникам здійснювати швидші, безпечніші та якісніші зміни, тобто підтримувати швидкість розробки продукту, навіть коли він надзвичайно великий. Зрештою, не тісно пов’язані служби дозволяють вносити зміни з більшою частотою ітерацій, мінімізуючи вплив змін на решту системи.

Існують значні проблеми з монолітною архітектурою, з точки зору розгортання, масштабування, розуміння величезної бази коду, переходу до нового фреймворку, безперервної доставки нових функцій, в термінах з'єднання між компонентами тощо.

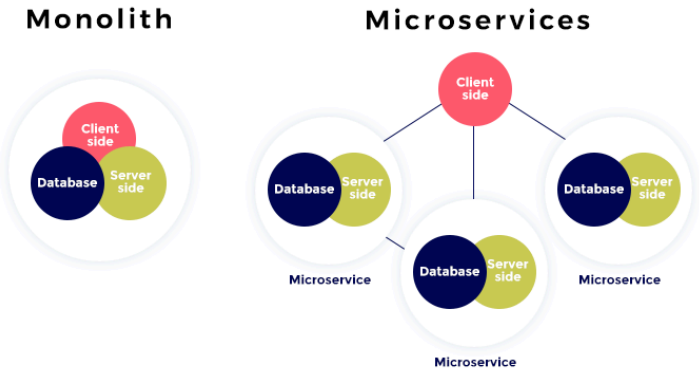

Рисунок 2.1

Розглянемо ці архітектури більш детально:

2.1.1 Монолітна архітектура

Монолітний додаток має єдину кодову базу з декількома модулями. Він має єдину систему складання, яка створює весь додаток. Він також має єдиний виконуваний або розгортається двійковий файл. При розробці

серверного додатка ви можете запустити його з модульною архітектурою, яка складається з різних типів компонентів.

Презентація - відповідає за обробку запитів HTTP та відповідь за допомогою HTML або JSON / XML (для API веб-служб).

Бізнес-логіка - ділова логіка програми.

Доступ до бази даних - об’єкти доступу до даних, відповідальні за доступ до бази даних.

Інтеграція додатків - інтеграція з іншими службами (наприклад, за допомогою обміну повідомленнями або REST API).

Незважаючи на наявність логічно модульної архітектури, програма упакована та розгорнута як моноліт.

Плюси:

1) Проста у розробці та тестуванні.

2)Проста у розгортанні. Вам просто потрібно скопіювати пакувальну програму на сервер.

3) Просте масштабування по горизонталі шляхом запуску декількох копій.

Це добре працює на ранніх стадіях проекту. Більшість великих та успішних програм, які існують сьогодні, були розпочаті як моноліт.

Мінуси:

1) Простий підхід має обмеження в розмірах і складності.
2) Додаток занадто великий і складний, щоб повністю зрозуміти та внести зміни швидко та правильно.
3) Розмір програми може сповільнити час запуску.
4) Потрібно переносити всю програму під час кожного оновлення.
5) Вплив змін, як правило, не дуже добре зрозумілий, що призводить до необхідності широкого ручного тестування.
6) Постійне розгортання стає важким.

7) Монолітні програми також може бути важко масштабувати, коли різні модулі мають суперечливі вимоги до ресурсів.
8) Ще однією проблемою монолітних додатків є надійність. Помилка в будь-якому модулі (наприклад, витік пам'яті) може потенційно збити весь процес. Більше того, оскільки всі екземпляри програми однакові, ця помилка вплине на доступність усієї програми.
9) Монолітні програми мають перешкоду для впровадження нових технологій. Оскільки зміни у фреймворках або мовах вплинуть на весь додаток, це надзвичайно дорого як за часом, так і за вартістю.

2.1.2 Мікросервісна архітектура

Ідея полягає в тому, щоб розділити вашу програму на набір менших, взаємопов’язаних служб, замість того, щоб створювати єдиний монолітний додаток. Кожен мікросервіс - це невелике додаток, яке має власну архітектуру, що складається з бізнес-логіки разом з різними адаптерами. Деякі мікросервіси можуть використовувати REST, RPC або API на основі повідомлень, а більшість служб використовують API, що надаються іншими службами. Інші мікросервіси можуть реалізувати веб-інтерфейс.

Шаблон мікросервісної архітектури суттєво впливає на взаємозв'язок між програмою та базою даних. Замість спільного використання однієї схеми бази даних з іншими службами, кожна служба має власну схему бази даних. З одного боку, такий підхід суперечить ідеї загально-корпоративної моделі даних. Крім того, це часто призводить до дублювання деяких даних.

Плюси:

1) Підхід вирішує проблему складності, розкладаючи додаток на набір керованих служб, які набагато швидше розробляються і набагато простіші для розуміння та обслуговування.
2) Це дозволяє розробляти кожну послугу самостійно командою, яка зосереджена на цій послузі.

3) Це зменшує бар'єр на шляху впровадження нових технологій, оскільки розробники можуть вільно вибирати, які технології мають сенс для їх обслуговування та не обмежуються вибором, зробленим на початку проекту.
4) Архітектура мікросервісів дозволяє розгортати кожну мікросервіс незалежно. Як результат, це робить можливим постійне розгортання складних додатків.
5) Архітектура мікросервісів дозволяє масштабувати кожну послугу незалежно.

Мінуси:

1) Архітектура мікросервісів додає проекту складності лише тим, що додаток є розподіленою системою.
2) Архітектура розділених баз даних, яка стає більш складною для розробників для підтримки та управління інфраструктурою.
3) Тестування мікросервісів також набагато складніше, ніж у випадку монолітної програми.
4) Складніше впровадити зміни, що охоплюють кілька служб. Потрібно ретельно планувати та координувати розповсюдження змін до кожної із служб.
5) Розгортання програми на базі мікросервісів також є більш складним.
6) Оскільки загальна інфраструктура стає великою порівняно з монолітною, це потребує більше ресурсів для розробки та обслуговування вашого додатка.

Обидва підходи мають свої плюси і мінуси, але це залежить від кожного сценарію чи вимог до продукту / проекту та того, який компроміс ви оберете. Оскільки монолітний підхід найкраще підходить для легких застосувань, рекомендується спочатку застосувати монолітний підхід і,залежно від потреб, поступово переходити до підходу мікросервісів.

2.2 Порівняння фреймворків Django та Flask

Завдяки своїй відносній легкості та універсальності, Python доступна мова з великим простором для формування фреймворків.

Великою популярністю серед програмістів користуються фреймворки для створення веб-програм Django та Flask , спробуємо зрозуміти різницю між ними.

1. Принципова різниця

Існує одна основна різниця, яка розділяє ці два фреймворки: Django - це повнофункціональний веб-фреймворк Python, тоді як Flask - це легкий тип фреймворку. Це означає, що Django - це найкращий шлях, якщо ви шукаєте рішення, яке має багато в собі - є панель адміністратора, інтерфейси бази даних, ORM, структура каталогів для програм та проектів. Усе це в сукупності дозволяє набагато швидше розпочати процес розробки додатків.

З іншого боку, Flask спочатку може здатися дещо біднішим, але це тому, що він створений в першу чергу для гнучкості. Оскільки він не постачається з дюжиною функцій прямо з коробки - він простий у використанні та дає вам достатньо місця для впровадження ваших інструментів. Він може дуже легко адаптуватися до ваших потреб, надаючи набагато більше контролю над усім процесом розробки додатків. Завдяки впорядкованому підходу розробникам легше забезпечити, щоб додаток залишався, по суті, простим та легким для розширення.

2. Продуктивність

Ефективність Flask проти Django є важливим показником функціональності системи. Якщо фреймворк є високопродуктивним, це призведе до масштабованої, швидкої та безпечної програми. Слабка продуктивність може призвести до нарощування болю та архітектурних проблем.

Якщо розглядати ефективність Django vs Flask, вони мають хороші результати і використовуються веб-сайтами з високим трафіком, що є чудовим показником їх ефективності.

3. Особливості

Перелік функцій, налаштованих для Django за замовчуванням, буде суттєво ширшим, включаючи:

1. ORM (Object Relational Mapping), який підтримує безліч різних реляційних баз даних, включаючи SQLite, PostgreSQL, MySQL, Oracle тощо. Підтримується міграція, як і створення форм, подань та шаблонів.

2. Автентифікація - Django оснащений автентифікацією, поєднаною з авторизацією, управлінням обліковими записами та підтримкою сеансів.

3. Функціональна адміністративна панель, що значно полегшує управління даними через інтуїтивно зрозумілий інтерфейс.

4. Форми за замовчуванням поставляються в Django - їх можна створювати з моделей даних і вони можуть обробляти різні аспекти розробки додатків, включаючи перевірку на стороні клієнта та сервера, маркери безпеки, підробку міжсайтових запитів тощо.

Flask, хоч і набагато більше спрямований на те, щоб його можна було налаштувати, поставляється цих функцій. Він пропонує підтримку всіх типів функцій, перелічених вище, але ви маєте повну свободу у виборі типу бази даних, адміністративної панелі, автентифікації тощо. Ви також можете просто відмовитися від функцій, які, на вашу думку, не будуть потрібні вашому проекту.

4. Безпека

Django оснащений деякими досить сильними заходами безпеки. Він захищений від деяких найпоширеніших векторів атак, включаючи CSRF, XSS, введення SQL тощо. Захист, який він пропонує, є достатнім у більшості випадків.

Однак у Flask важливо відзначити, що менша база коду працює на свою користь з точки зору безпеки, оскільки це означає, що менша кількість областей фреймворку відкрито для атак. Ви, звісно, захочете застосувати деякі сторонні функції безпеки, але хоча це дає вам набагато більше свободи у виборі способу захисту, вам також потрібно буде переконатися, що всі ваші розширення актуальні.

5. Пакети

Flask мінімалістичний і не має обмежень, тобто розробники можуть реалізувати саме те, що хочуть, використовуючи зовнішні бібліотеки. Це робить Flask гнучким та розширюваним.

Django, навпаки, має величезну кількість вбудованих пакетів. Якщо бути точним, станом на квітень 2020 року було 4665 пакетів Django [5]. Це означає, що ви, швидше за все, знайдете пакет для створення та запуску вашого додатка з меншими зусиллями.

6. Використання

Останнє, що слід врахувати, - це те, наскільки широко використовуються ці два фреймворки.

Flask - одна з провідних платформ веб-розробки Python. Згідно з опитуваннями розробників Python від JetBrains [6], використання Flask серед розробників зросло з 41% у 2017 році до 47% у 2018 році. Причин, чому такі відомі світові компанії, як Airbnb та Reddit, використовують Flask. Flask надає вам більше контролю над вашим проектом, оскільки ви можете вибрати, які компоненти використовувати і як ви з ними взаємодієте. Крім того, ви можете підключити будь-яке розширення, яке вам потрібно.

Хоча Flask на сьогодні є популярнішим вибором, багато відомих веб-сайтів все ще використовують Django (Spotify, Instagram, Mozilla, Dropbox тощо)

Отже, Flask – це кращий варіант, якщо ви набагато більше зосереджені на процесі та хочете гарантувати, що ваш додаток зможе легко розвиватися,

оскільки ви набагато більше контролюєте на кожному етапі і вам дозволяється вирішувати, як взаємодіяти з базами даних.

Django, навпаки, є кращим варіантом для тих, хто хоче розробити набагато більше додатків - що він втрачає в гнучкості, він компенсує у передбачуваності своїх процесів.

▪ Розробка програмного продукту

3.1 Вхідні дані

JSON (JavaScript Object Notation) - це легкий формат обміну даними. Людям легко читати та писати його. Машини легко його аналізують і генерують. Він заснований на підмножині стандарту мови програмування JavaScript ECMA-262 3-е видання - грудень 1999 р. JSON - це текстовий формат, який повністю не залежить від мови, але використовує конвенції, знайомі програмістам сімейства мов C, включаючи C, C ++, C #, Java, JavaScript, Perl, Python та багато інших. Ці властивості роблять JSON ідеальною мовою обміну даними. [8]

На вхід застосунок отримує JSON з масивом об’єктів з ключами “time”, “discipline”, “day”, “teacher”, “group”, “weeks”, “classroom”.

```
{
  "data" : [
    {
      "time" : "10:00-11:20",
      "discipline" : "Англійська мова( за професійним спрявуванням)",
      "day" : "Понеділок",
      "teacher" : "ст. викладач Л.М. Залєська-Зарічна",
      "group" : "А40",
      "weeks" : "2 – 15",
      "classroom" : "Дистанційно"
    },
    {
      "time" : "11:40-13:00",
      "discipline" : "Англійська мова( за професійним спрявуванням)",
      "day" : "Понеділок",
      "teacher" : "ст. викладач Л.В. Сергієнко",
      "group" : "А41",
      "weeks" : "2 – 15",
      "classroom" : "Дистанційно"
    }
  ]
}
```

3.2 Робота з Excel таблицями використовуючи openpyxl

Openpyxl - бібліотека Python для читання та запису файлів Excel у форматах xlsx, xlsm, xltx та xltm. Створена через відсутність існуючої бібліотеки для читання та запису файлів у форматі Office Open XML format.

```
import json

from openpyxl import load_workbook, Workbook
from openpyxl.styles import Font, Border, Side, Alignment

teacher_font = Font(name="Times New Roman", size=12, bold=True)
font = Font(name="Times New Roman", size=12, bold=False)
thin_side = Side(border_style="thin", color="000000")
thick_side = Side(border_style="thick", color="000000")
border = Border(left=thin_side, right=thin_side,
                top=thin_side, bottom=thin_side)
alignment = Alignment(horizontal="center", vertical="center")
header_border = Border(left=thin_side, right=thin_side,
                       top=thick_side,
                       bottom=thick_side)
header_left_side_border = Border(left=thick_side, right=thin_side,
                                 top=thick_side,
                                 bottom=thick_side)
header_right_side_border = Border(left=thin_side, right=thick_side,
                                  top=thick_side,
                                  bottom=thick_side)
body_left_side_border = Border(left=thin_side, right=thin_side,
                               top=thick_side,
                               bottom=thick_side)
body_right_side_border = Border(left=thin_side, right=thick_side,
                                top=thin_side,
                                bottom=thin_side)
body_bottom_side_border = Border(left=thin_side, right=thin_side,
                                 top=thin_side,
                                 bottom=thick_side)
body_bottom_tight_side_border = Border(left=thin_side, right=thick_side,
                                       top=thin_side,
                                       bottom=thick_side)
```

Рисунок 3.1

На Рисунку 3.1 приведено код під’єднання потрібних зовнішніх ресурсів та винесені константи для стилів тексту та стилів рамок.

```
40  def create_draft():
41      draft = Workbook()
42      ws1 = draft.active
43      ws1.title = "Schedule"
44      ws1.cell(1, 1).value = "День"
45      ws1.cell(1, 2).value = "Час"
46      ws1.cell(1, 3).value = "Дисципліна"
47      ws1.cell(1, 4).value = "Група"
48      ws1.cell(1, 5).value = "Тижні"
49      ws1.cell(1, 6).value = "Аудиторія"
50      time_array = get_all_times()
51      time_iter = 0
52      for i in range(2, len(time_array) * len(days_array) + 2):
53          ws1.cell(i, 2).value = time_array[time_iter]
54          if time_iter != len(time_array) - 1:
55              time_iter += 1
56          else:
57              time_iter = 0
58      return draft
```

Рисунок 3.2

На рисунку 3.2 зображено створення шаблону в який буде поміщено розклад.

```
73  def get_all_times():
74      json_file = open("data.json")
75      data = json.load(json_file)
76      times_dict = {}
77      for d in data['lessons']:
78          if d['time'] not in times_dict:
79              times_dict[int(d['time'].split(':')[0])] = d['time']
80      times_array = []
81      for key in sorted(times_dict):
82          times_array.append(times_dict[key])
83      print(times_array)
84      return times_array
```

Рисунок 3.3

На рисунку 3.3 приведено код функції, яка знаходить всі можливі години проведення занять, та повертає їх масив.

```python
def read_json_file(ws):
    json_file = open("data.json")
    data = json.load(json_file)
    for p in data['lessons']:
        row = 1
        cell = ws.cell(row, 2)
        day = days[p['day']]
        print(day)
        check_day = 0
        while check_day != day:
            print("check")
            if cell.value == "8:30-9:50":
                check_day += 1
            row += 1
            cell = ws.cell(row, 2)
        cell = ws.cell(row, 2)
        while cell.value != p['time']:
            print(p['time'])
            print(cell.value)
            row += 1
            cell = ws.cell(row, 2)
        if ws.cell(row, 3).value is not None:
            ws.insert_rows(row + 1)
            row += 1
            ws.cell(row, 2).value = p['time']

        ws.cell(row, 3).value = f"{p['discipline']}, {p['teacher']}"
        ws.cell(row, 3).border = border
        ws.cell(row, 3).font = teacher_font
        ws.cell(row, 3).alignment = alignment

        ws.cell(row, 4).value = p['group']
        ws.cell(row, 4).border = border
        ws.cell(row, 4).font = font
        ws.cell(row, 4).alignment = alignment

        ws.cell(row, 5).value = p['weeks']
        ws.cell(row, 5).border = border
        ws.cell(row, 5).font = font
        ws.cell(row, 5).alignment = alignment

        ws.cell(row, 6).value = p['classroom']
        ws.cell(row, 6).border = border
        ws.cell(row, 6).font = font
        ws.cell(row, 6).alignment = alignment
```

Рисунок 3.4

На рисунку 3.4 функція, яка отримує шаблон та послідовно заповнює рядки шаблону даними з JSON.

3.3 Створення користувацького інтерфейсу

Інтерфейс було вирішено зробити в вигляді веб-застосунку з використанням фреймворку для Python Django, HTML та CSS з бібліотекою Bootstrap

Django було використано через його простоту при розробці клієнт-серверного веб-застосування з використанням мови Python. HTML, CSS та– універсальні рішення для розробки веб-застосувань.

У веб-застосунку передбачено додавання факультетів, спеціальностей та студентів, реалізована можливість додавати кілька розкладів для одного студента. Після завантаження JSON файлів розкладу, вони конвертуються у xlsx і доступні для завантаження на головній сторінці.

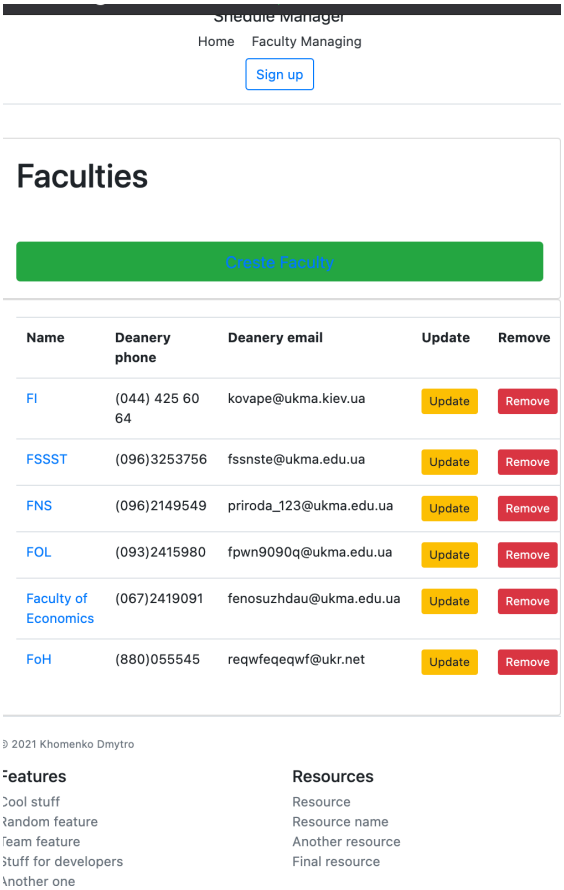

Рисунок 3.5 Інтерфейс веб-застосунку

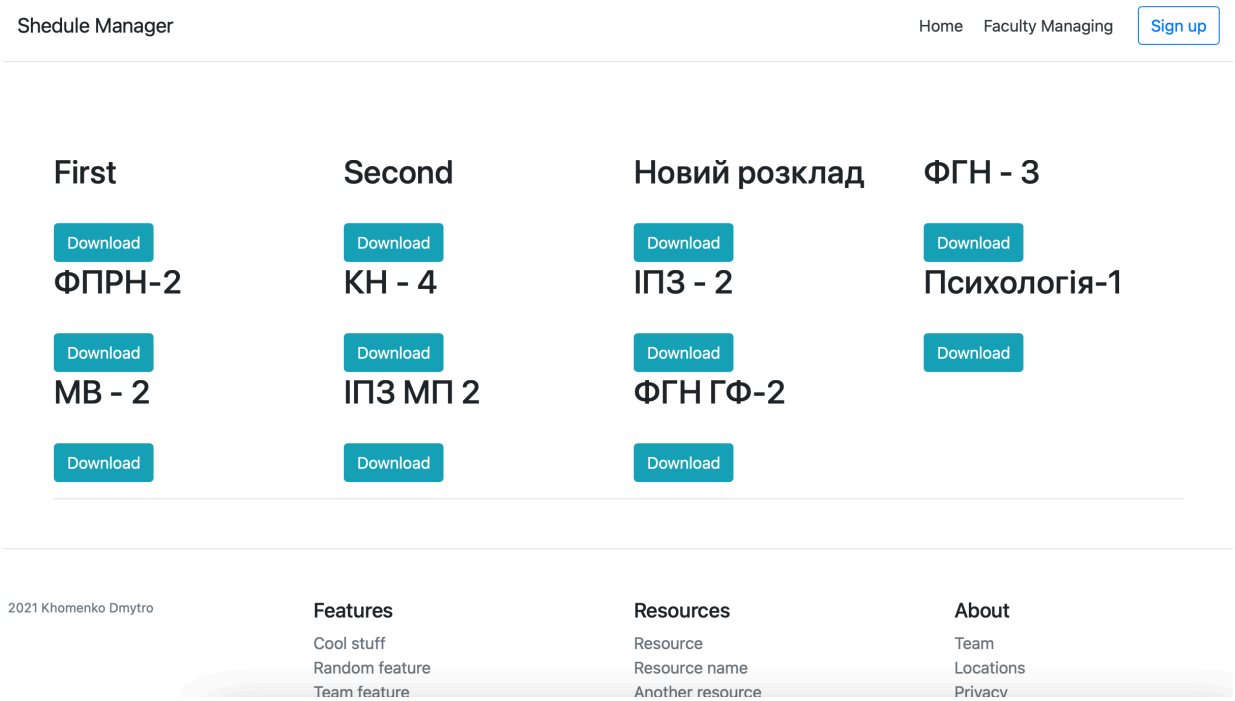

Рисунок 3.6 Домашня сторінка

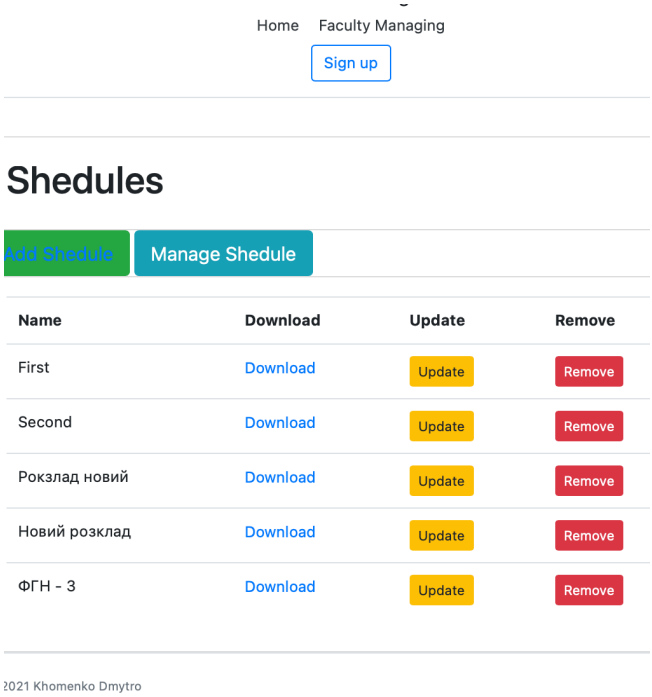

Рисунок 3.6 Меню додавання розкладів

Висновок

В результаті роботи мета була досягнута. Детально проаналізовані сучасні технології розробки та архітектурні підходи. Отримані знання при підготовці є важливими та актуальними сьогодні. Реалізовано універсальне рішення генерації розкладу за вхідним json, яке можна використати не тільки у НаУКМА, але і у інших вищих навчальних закладах, школах тощо. Скрипт, який використаний при розробці Веб-сервісу на Django, можна використати для створення телеграм-боту, для інтеграції з уже існуючими рішеннями чи навіть для мобільної розробки. Створений веб-сайт, хоч і програє в зручності @KMASchedulerBot'у чи системі my.ukma.edu.ua, але і має свої переваги – наприклад можна згенерувати декілька індивідуальних розкладів, врахувавши пари у інших навчальних закладах, відкриті лекції чи навіть тренування в спортзалі.

Список джерел

1. Python 3.8.3rc1 documentation[Електронний ресурс] – режим доступу: https://docs.python.org/3/
2. REST Roy Thomas Fielding[Електронний ресурс] – режим доступу https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
3. Django documentation[Електронний ресурс] – режим доступу https://docs.djangoproject.com/en/3.0/
4. Openpyxl documentation[Електронний ресурс] – режим доступу: https://openpyxl.readthedocs.io/en/stable/
5. Django packages[Електронний ресурс] – режим доступу: https://djangopackages.org/
6. Flask documentation[Електронний ресурс] – режим доступу: https://flask.palletsprojects.com/en/1.1.x/
7. Python Developers Survey[Електронний ресурс] – режим доступу: https://www.jetbrains.com/research/python-developers-survey-2018/
8. Introducing JSON[Електронний ресурс] – режим доступу: https://www.json.org/json-en.html