МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
НА ТЕМУ

Веб-застосунок пошуку робочих місць з IT спеціальностей

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма «**Прикладне програмування»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи ПП-42

Піцик Є. В.
(прізвище та ініціали)

Керівник Пирог М. В.
(прізвище та ініціали)

Унікальність тексту: 89%

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *прикладних інформаційних систем*
Протокол № 14 від 23.05.2023р.
зав. кафедри Плескач В. Л.

Київ – 2023

Київський національний університет імені Тараса Шевченка

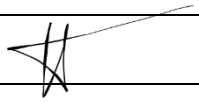
Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

Назва теми: "Веб-застосунок пошуку робочих місць з IT спеціальностей"

Освітня програма: Прикладне програмування

Спеціальність: Комп'ютерні науки

Піцик Єгор Володимирович	

Назва роботи українською та англійською мовами:

Веб-застосунок пошуку робочих місць з IT спеціальностей
Web-based job search applicaITon for IT specialITes

Мета бакалаврської роботи: надання актуальних даних про епідеміологічну ситуацію, пов'язану з пошуком робочих місць з IT спеціальностей за допомогою веб-застосунку.
План роботи: 1. Сучасні методики створення та використання веб-сервісів. 2. Оцінка архітектурних рішень і підбір програмних інструментів для створення веб-системи 3. Програмна реалізація веб-сервісу для пошуку робочих місць з IT спеціальностей

Асистент Пирог Микола Володимирович 

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	14.10.2022	Виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	24.10.2022	Виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	31.10.2022	Виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.11.2022	Виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.11.2022	Виконано
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	21.12.2022	Виконано
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2023	Виконано
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	30.03.2023	Виконано
9.	Подання роботи у першому варіанті	28.04.2023	Виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2023	Виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**22.05.2023**	Виконано

12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	26.05.2023	Виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	12.06.2023	Виконано
14.	Захист кваліфікаційної роботи бакалавра	28.06.2023	Виконано

Здобувач вищої освіти _____
(підпис)

Керівник _____
(підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	2
Відомість дипломної роботи	1
Заява	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	2
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
1	10
2	21
3	21
Висновки	2
Перелік використаних джерел	4
Додатки	2

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3">ДП ХХХХ 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td><td colspan="3"></td></tr>
<tr><td>Розробн.</td><td>Піцик Є. В.</td><td></td><td></td><td rowspan="4">Відомість дипломної роботи</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Пирог М.В.</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Н/контр.</td><td>Кравченко К.В.</td><td></td><td></td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Зав.каф.</td><td>Плескач В.Л.</td><td></td><td></td></tr>
</table>

АНОТАЦІЯ

Дипломна робота: 72 с., 13 рис., 2 табл., 30 джерел, 2 додатка.

Ця дипломна робота присвячена розробці веб-застосунку для моніторингу ринку праці та пошуку робочих місць з ІТ спеціальностей

Метою дипломної роботи є ефективний пошук робочих місць з ІТ спеціальностей на основі розробленого веб-застосунку.

Для досягнення поставленої мети треба вирішити такі **завдання**:

Дослідити сучасні підходи до розроблення і впровадження веб-сервісів.

Проаналізувати архітектурні рішення і обрати програмні засоби для реалізації веб-системи.

Розробити та реалізувати веб-інтерфейс користувача, який забезпечуватиме зручний та швидкий пошук вакансій за певними параметрами, такими як спеціалізація, регіон, рівень заробітної плати тощо.

Об’єкт дослідження – процеси моніторингу ринку праці та пошуку робочих місць з ІТ спеціальностей.

Предмет дослідження – програмно-технічні, організаційні засади, принципи, підходи щодо побудови веб-застосунку для моніторингу ринку праці та пошуку робочих місць з ІТ спеціальностей

Методи дослідження теорія управління, аналіз і синтез систем, моделювання, системний підхід.

Ключові слова: веб-застосунок, моніторинг, ІТ спеціальності, ринок праці.

ABSTRACT

Diploma thesis: 72 p., 13 figures, 2 tables, 30 sources, 2 appendices.

This thesis is devoted to the development of a web application for monitoring the labor market and searching for jobs in IT specialties.

The purpose of the thesis is to develop and implement a web application that will effecITvely collect, process and display informaITon about vacancies with specialization in the field of IT, providing users with convenient and quick access to current offers of employers in this field.

To achieve this goal, the following tasks need to be solved:

- Research modern approaches to the development and implementation of web services;
- Analyze architectural soluITons and choose software tools for implementing a web system;
- Develop and implement a web-based user interface that will provide a convenient and quick search for vacancies by certain parameters, such as specialization, region, salary level, etc.

Object of research. The processes of monitoring the labor market and searching for jobs in IT specialties.

Subject of research. Software and hardware, organizaITonal principles, principles, approaches to building a web applicaITon for monitoring the labor market and searching for jobs in IT specialties.

Management theory for the study of theoretical aspects of building web applications, empirical analysis and synthesis of systems, which was used to study examples of modern methods of building web applications for monitoring objects and phenomena, modeling, diagramming, designing data structures.

Keywords: web application, monitoring, IT specialties, labor market.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

Фреймворк – інфраструктура програмних рішень, що полегшує розробку.

Бібліотека – збірка об’єктів чи підпрограм для вирішення близьких за тематикою задач.

Модуль – функціонально завершений фрагмент програми, оформлений у вигляді окремого файлу з сирцевим кодом або його іменованої частини, призначений для використання в інших програмах.

Деплой – розгортання програмного забезпечення, усі дії, що роблять програмну систему готовою до використання.

Інтрефейс – сукупність засобів для обробки та відбиття інформації, якнайбільше пристосованих для зручності користувача.

HTTP – HyperText Transfer Protocol.

SSH – Secure Shell.

HTTPS – HyperText Transfer Protocol Secure.

HTML – HyperText Markup Language.

CSS – Cascading Style Sheets.

API – ApplicaITon Programming Interface.

Бекенд (Back-end) – серверна частина веб-сервісу.

Фронтенд (Back-end) – браузерна частина веб-сервісу, яка охоплює HTML, CSS, JavaScript.

ВСТУП

В сучасному світі інформаційних технологій, розвиток галузі технічних наук, таких як інформаційні технології, високі технології та програмна інженерія, відкриває широкі можливості для забезпечення доступу до актуальної та достовірної інформації про робочі місця в галузі технологій інформаційної безпеки. Одним із найефективніших інструментів вирішення цієї проблеми є створення веб-застосунку пошуку робочих місць з IT спеціальностей, що забезпечить користувачів інформацією про доступні робочі місця в галузі технічних наук, а також полегшить пошук вакансій і взаємодію з роботодавцями. У цьому дослідженні буде розглянуто процес розробки веб-застосунку з використанням сучасних технологій та методів програмної інженерії для досягнення мети полегшення процесу пошуку робочих місць в галузі IT.

Актуальність теми веб-застосунку пошуку робочих місць за IT спеціальностями пов'язана зі швидким розвитком інформаційних технологій і значним зростанням попиту на фахівців у цій галузі. В умовах постійно мінливого ринку праці та необхідності швидкого доступу до актуальної інформації, створення такого веб-застосунку стає необхідністю. Крім того, він допоможе молодим фахівцям у цій галузі, а також тим, хто шукає перспективну роботу в IT-секторі, швидко і зручно знаходити підходящі вакансії та отримувати інформацію про компанії та їхні вимоги.

Метою дипломної роботи є ефективний пошук робочих місць з IT спеціальностей на основі розробленого веб-застосунку..

Для досягнення поставленої мети треба вирішити такі **завдання**:

- Дослідити сучасні підходи до розроблення і впровадження веб-сервісів;
- Проаналізувати архітектурні рішення і обрати програмні засоби для реалізації веб-системи;

- Розробити та реалізувати веб-інтерфейс користувача, який забезпечуватиме зручний та швидкий пошук вакансій за певними параметрами, такими як спеціалізація, регіон, рівень заробітної плати тощо.

Об’єкт дослідження – процеси моніторингу ринку праці та пошуку робочих місць з ІТ спеціальностей.

Предмет дослідження – програмно-технічні, організаційні засади, принципи, підходи щодо побудови веб-застосунку для моніторингу ринку праці та пошуку робочих місць з ІТ спеціальностей

Методи дослідження теорія управління, аналіз і синтез систем, моделювання, системний підхід.

Практичне значення одержаних результатів полягає у тому, що розроблена веб-платформа може стати корисним та сучасним рішенням для шукачів роботи в галузі ІТ. Веб-платформа надасть зручний та швидкий пошук відповідних робочих місць з ТІ спеціальностей з можливістю фільтрації та сортування вакансій за різними параметрами. Це може допомогти безробітним людям знайти роботу, а також роботодавцям - знайти кваліфікованих фахівців у галузі ІТ.

Структура роботи:

Кваліфікаційна робота бакалавра складається із вступу, трьох розділів, висновку, додатків і списку джерел.

РОЗДІЛ 1
СУЧАСНІ ПІДХОДИ ДО РОЗРОБЛЕННЯ І ВПРОВАДЖЕННЯ ВЕБ-СЕРВІСІВ

1.1 Веб-сервіс

Веб-сервіс або веб-служба - це програмна система, яка забезпечує можливість взаємодії з різними застосунками та платформами через використання стандартизованих інтерфейсів та протоколів. Одним з ключових переваг веб-сервісу є його доступність через Інтернет, що дозволяє забезпечити широкий спектр функцій та послуг без необхідності встановлення додаткових програм. Застосування веб-сервісів дозволяє забезпечити стандартизацію обміну даними між різними системами, що підвищує ефективність та надійність взаємодії між ними. Так, наприклад, веб-сервіс пошуку робочих місць з IT спеціальностей може забезпечити швидкий та зручний пошук вакансій відповідно до вимог користувача, що робить його необхідним та популярним серед професіоналів цієї галузі.

Веб-сервіси складаються зі скупчення декількох протоколів, кожен з яких відповідає за різні частини роботи веб-служби. Ці протоколи можуть бути розподілені на кілька рівнів:

- Фізичний
    - Включає протоколи, такі як ISDN та RS-232
- Канальний
    - Містить Ethernet, Fibre channel та Token ring
- Мережевий
    - Включає IP, IPX та ICMP
- Транспортний
    - Містить TCP, UDP та SPX.

- Прикладний
    - Включає HTTP, HTTPS, SSH, FTP, SMTP та інші.

Кожен протокол відповідає за свої власні аспекти взаємодії з зовнішніми застосунками і забезпечує стандартизовані інтерфейси для взаємодії між різними системами. Також вирізняють поняття «стеків протоколів», які являють собою систематизований набір протоколів, який є достатнім для організації роботи вузлів мережі. Найбільш популярним на даний момент є стек протоколів TCP/IP назва якого походить від двох основних протоколів стеку. Окрім нього, також використовуються IPX/SPX, NetBIOS/SMB, SNA, DECnet, AppleTals. Стек протоколів TCP/IP також корелює з моделлю OSI (вона теж має відповідний стес OSI), яка описує абстрактну еталонну модель для розробки мережевих протоколів. Ця модель описує більшу кількість рівнів, ніж стек TCP/IP, але їх можна зіставити одне з одним.

Таблиця 1.1 – Порівняння стеку протоколів TCP/IP з еталонною моделлю OSI

TCP/IP	**OSI**	**Приклад протоколів**
Прикладний	Прикладний	HTTP, FTP
	Представлення	SSL, TLS
	Сеансовий	ASP, RPC
Транспортний	Транспортний	TCP, UDP, SPX
Мережевий	Мережевий	IP, ICMP, IPX
Канальний	Канальний	Ethernet, Fibre channel, Token ring
	Фізичний	Проводи, радіозв’язок

При створенні основного веб-сервісу розробники зазвичай використовують протоколи на рівні прикладного програмного забезпечення. Нижче ми розглянемо деякі з цих протоколів докладніше:

- HTTP є головним протоколом в Інтернеті, відповідаючи за передачу гіпертекстових документів, які мають зазвичай розмітку HTML. Також, HTTP може передавати і складніші об'єкти, наприклад, зображення.
- SSH є протоколом, який забезпечує шифрування даних в Інтернеті. Цей протокол має декілька версій, але рекомендується використовувати останню, SSH-2.
- FTP є протоколом, який використовується для передачі файлів між клієнтом та сервером. Щоб забезпечити безпеку передачі даних, у цьому протоколі використовуються SSL-надбудови.
- DNS є протоколом, який використовується для перетворення імені хоста на його IP-адресу.
- POP3 є протоколом, який відповідає за доступ клієнта до повідомлень електронної пошти, які знаходяться на сервері.
- VoIP є протоколом, який дозволяє передавати голосові повідомлення через Інтернет.

1.2 Протоколи реалізації веб-сервісів

Найбільш популярними стандартами API є SOAP, REST та RPC. Кожен з них має свої переваги та недоліки. SOAP, наприклад, забезпечує високий рівень захисту даних, однак, з іншого боку, вимагає значних зусиль при розробці. Це може стати проблемою, особливо якщо дані не потребують високого рівня захисту. RPC, з іншого боку, забезпечує простоту та швидкість розробки, але більш підходить для мікросервісної архітектури.

У свою чергу, REST API вважається найбільш гнучким та простим в розробці, оскільки не має строгих правил та стандартів. Це дозволяє розробникам

створювати API, які більш точно відповідають потребам конкретного веб-застосунку. Однак, використання REST може бути менш безпечним порівняно з SOAP. Це пов'язано з тим, що REST не забезпечує такого рівня захисту даних, як SOAP, тому його використання потребує додаткових заходів щодо захисту інформації.

Отже, вибір стандарту API повинен здійснюватися на основі конкретних потреб веб-застосунку. Якщо дані не потребують високого рівня захисту, а оптимізація пошуку користувачем не є важливою, REST API може стати найбільш очевидним вибором для розробки. Однак, якщо потрібний високий рівень захисту, SOAP може бути більш підходящим варіантом. RPC варто розглядати, якщо веб-застосунок має бути використаний зовнішніми користувачами та заснований на мікросервісній архітектурі

У веб-розробці існує декілька архітектурних стилів, які застосовуються для побудови веб-застосунків. Наприклад:

- RPC – стиль, який працює на віддалених процедурах. В цьому контексті віддалені процедури – це повідомлення з параметрами і додатковою інформацією, які відправляється від клієнта на сервер, десеріалізується там, після чого сервер, при валідності операції, виконує її та відправляє результат клієнту.
- SOAP – це жорстко стандартизований протокол, який використовує формат XML. SOAP вимагає певний формат повідомлень, якими обмінюються клієнт та сервер. Через високий рівень стандартизації та жорсткі вимоги до структури повідомлень SOAP є найбільш детальним стилем API. Протокол є достатньо гнучким для використання різних транспортних протоколів.
- REST - цей стиль базується на використанні HTTP-протоколу і передбачає, що кожен ресурс має свій унікальний ідентифікатор (URL), за яким можна звернутися до нього для отримання або модифікації інформації. REST

архітектура передбачає кешування і кеши можуть бути використані для зменшення навантаження на сервер і збільшення швидкості роботи веб-застосунку.

- GraphQL - цей стиль заснований на мові запитів GraphQL, яка дозволяє клієнтам отримувати тільки ту інформацію, яку вони потребують. GraphQL передбачає, що клієнти створюють запити на сервер, які описують, яку інформацію вони хочуть отримати, а сервер повертає тільки ту інформацію, яку клієнт запросив.

Таблиця 1.2 – Порівняння архітектурних стилів [4]

	RPC	SOAP	REST	GraphQL
Організація у вигляді	локальних процедур	обгорнутого повідомлення з жорсткою сруктурою	відповідності шести архітектурним принципам	схеми та типів даних
Формат даних	JSON, XML, Protobuf, Thrift, FlatBuffers	лише XML	XML, JSON, HTML, текст	JSON
Простота вивчення	Просто	Складно	Просто	Посередньо
Ком'юніті	Середнє	Мале	Велике	Велике
Використання	Командні і подіє-орієнтовані API; внутрішні мікросервіси з високою	Білінгові системи, фінансові сервіси, сервіси, які вимагають	Просто публічні застосунки	Мобільні API, складні системи, мікросервіси

	продуктивність	велиої захищенності		

При розробці веб-застосунку виникає необхідність обрати протокол, який буде використовуватися для обміну даними. SOAP та REST - це два найпопулярніших протоколи, які можуть бути використані для цієї мети.

SOAP є протоколом, що дозволяє забезпечити високий рівень захисту даних, що передаються між сервером та клієнтом. Однак, якщо мої дані не потребують такого рівня захисту, то використання SOAP може ускладнити процес розробки та не мати значних переваг для застосунку.

RPC також може бути використаний для обміну даними між сервером та клієнтом. Але, оскільки мій застосунок має бути використаний зовнішніми користувачами, RPC більш підходить для мікросервісної архітектури, де мікросервіси зазвичай знаходяться в межах одного дата-центру або хмарної інфраструктури.

Отже, залишається вибір між SOAP та REST. Один з способів прийняття рішення - оцінити складність даних. Оскільки мої дані не є дуже складними, оптимізація пошуку користувачем не є необхідною. Це робить REST API більш очевидним вибором для розробки.

Отже, розглянувши всі переваги та недоліки різних протоколів, було прийняте рішення використовувати REST API для розробки веб-застосунку.

1.3 Принципи роботи REST API

REST - це архітектурний стиль, що використовується для створення стандартизованих веб-інтерфейсів. Для того, щоб веб-застосунок можна було вважати RESTful, він повинен відповідати наступним шести критеріям:

- Єдиний інтерфейс: REST надає єдиний спосіб взаємодії з сервером на будь-якому пристрої. Це спрощує розробку та забезпечує єдність у способах комунікації.
- Відсутність стану: Сервер не зберігає стан застосунку. Усю необхідну інформацію передає клієнт у запиті, використовуючи параметри та тіло запиту.
- Кешування: REST підтримує кешування, що дозволяє клієнтам зберігати локальні копії відповідей сервера. Це поліпшує продуктивність та зменшує навантаження на сервер.
- Клієнт-серверна архітектура: REST використовує розділення між клієнтом та сервером, що дозволяє незалежний розвиток обох компонентів. Це дає можливість вдосконалювати інтерфейс користувача без впливу на серверну логіку та навпаки.
- Багаторівнева система: REST підтримує багаторівневу систему, де кожен рівень має свою функціональність та відповідальність. Це полегшує масштабування та розподілення функцій між компонентами.

REST API архітектура дозволяє серверу передавати код для виконання на стороні клієнта, розширюючи можливості обміну логікою та виконання додаткових операцій на клієнтському пристрої. Цей підхід дозволяє створювати повноцінні веб-сервіси, базові операції яких пов'язані з чотирма методами HTTP-запитів, забезпечуючи роботу з даними:

- GET - отримання даних з сервера у форматі JSON. Цей запит використовується для отримання інформації з ресурсу.
- POST - додавання нових даних на сервер, що викликає зміну стану сервера та може мати побічні ефекти. Цей метод використовується для створення нових сутностей на сервері.
- PUT - модифікація існуючих даних на сервері шляхом заміни інформації про певний ресурс, надіслану клієнтом. Це дозволяє змінювати наявні дані.
- DELETE - видалення даних з сервера, знищуючи вказаний клієнтом ресурс. Цей метод використовується для видалення існуючих сутностей.

Таким чином, REST API забезпечує зручний спосіб обміну даними між сервером і клієнтом, використовуючи стандартні методи HTTP для виконання різноманітних операцій над ресурсами.

Схему роботи REST API зображено на рисунку 1.1.

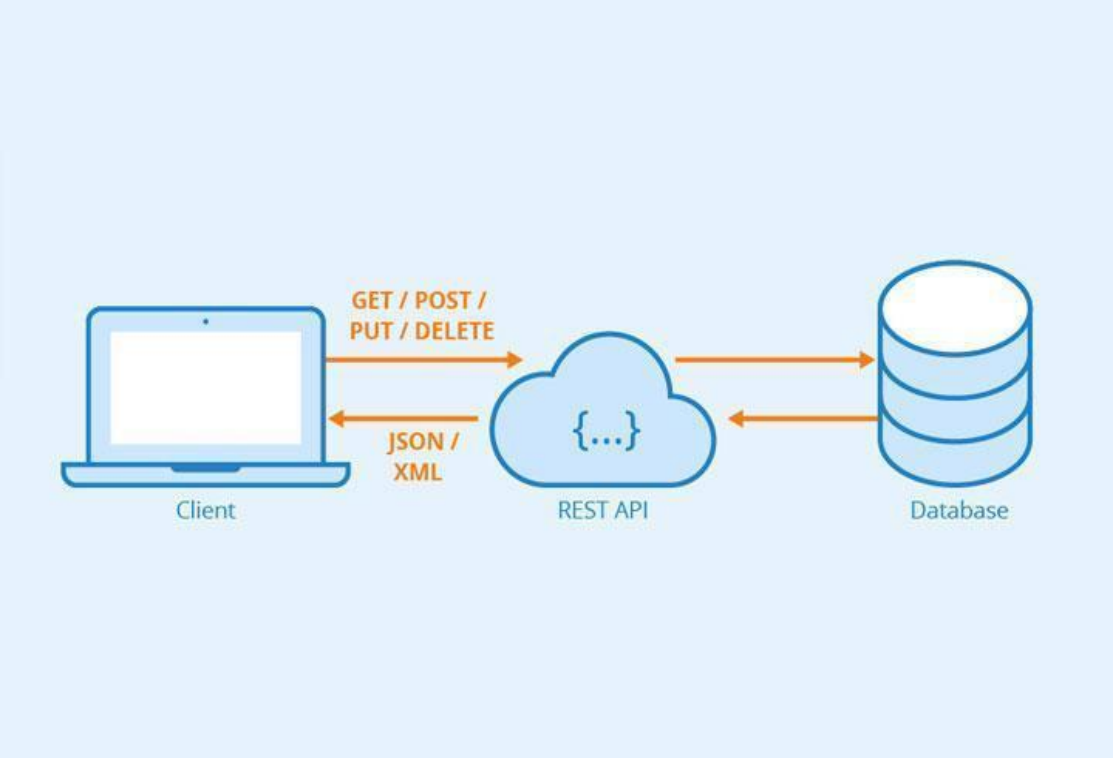


Рисунок 1.1 − Схема роботи REST API

REST в кожній відповіді постачає цінні метадані, що вичерпно описують актуальне використання API. Ці метадані не лише дозволяють REST розрізняти клієнтів від сервера, але й забезпечують незалежний розвиток обох сторін без ускладнень у комунікації між ними. Важливо зазначити, що існує кілька рівнів зрілості API, які відрізняються способом розрізнення клієнтської та серверної частин. Кожен рівень забезпечує певний рівень гнучкості та функціональності, дозволяючи використовувати API з максимальною ефективністю. (рисунок 1.2)

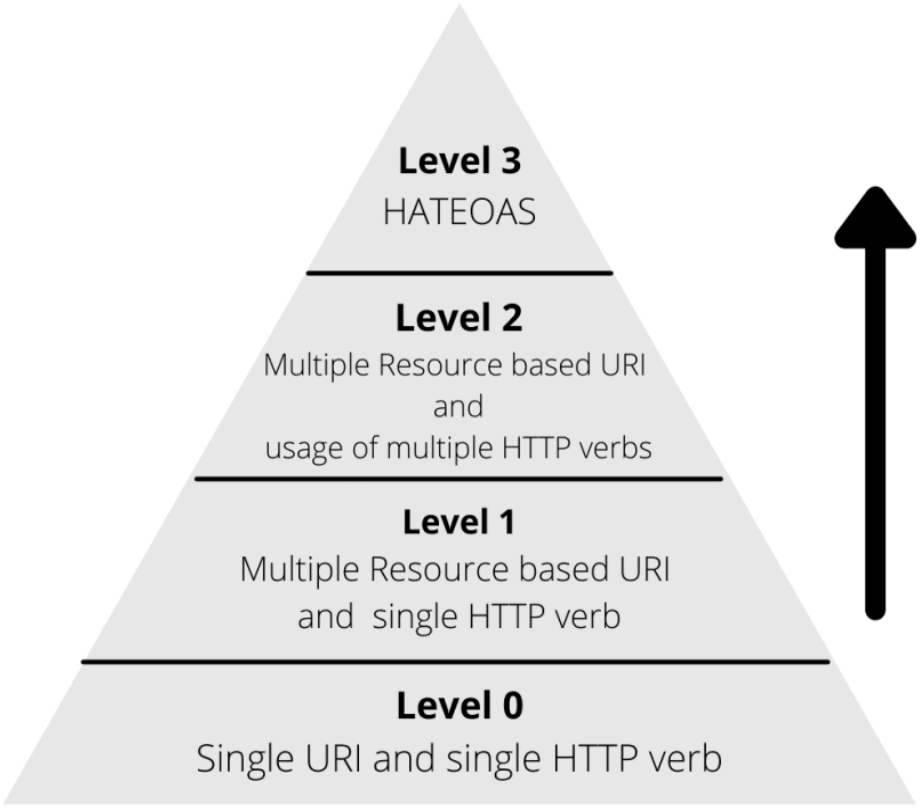


Рисунок 1.2 − Схема зрілості Річардсона, яка слугує орієнтиром для створення корисних та наповнених API. REST досягає найвищого рівня за рахунок метаданих.

Висновки до розділу

При розробці веб-застосунків, вибір стандартизованого програмного інтерфейсу (API) є критичним, оскільки він забезпечує зручну та безпечну комунікацію між клієнтом та сервером. Цей етап потребує уважного аналізу, оскільки на ринку існує безліч різних стандартів API, кожен з яких має свої переваги та особливості.

При виборі підходящого API варто враховувати конкретні потреби та характеристики проекту. Спочатку необхідно визначити складність даних, з якими буде працювати веб-застосунок. Якщо проект має великі обсяги даних або включає розподілені системи з різноманітними джерелами інформації, рекомендується обрати API, який забезпечує швидку та ефективну обробку такої інформації.

Другим аспектом, який варто врахувати при виборі API, є рівень захисту. Якщо проект передбачає обробку конфіденційної інформації, такої як особисті дані користувачів або фінансові транзакції, слід звернути увагу на API з високим рівнем безпеки та шифрування даних. Забезпечення конфіденційності та запобігання можливим атакам є ключовими факторами успіху веб-застосунку.

Отже, при виборі стандартизованого програмного інтерфейсу (API) для веб-застосунків, важливо ретельно аналізувати доступні варіанти, враховувати складність даних та потреби проекту, а також забезпечити високий рівень безпеки для обробки конфіденційної інформації.

РОЗДІЛ 2
АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ І ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-СИСТЕМИ

2.1 Бекенд фреймворк

Для успішної реалізації бекенду мого веб-ресурсу, я планую створити власний REST API, що забезпечить надійну і ефективну комунікацію між сервером та клієнтом. Це буде потужний інструмент, який виконуватиме різноманітні серверні операції, такі як оновлення даних, передача даних клієнту, фільтрація та адаптація даних.

Для створення цього REST API я можу вибрати одну з двох можливостей. Перша опція - це розробка REST API власноруч з нуля, використовуючи будь-яку мову програмування, що найкраще відповідає моїм потребам. Це забезпечить повний контроль над функціональністю та архітектурою системи. Я зможу налаштувати API точно під мої потреби та використовувати найновіші технології та практики.

Однак, розробка REST API з нуля може бути часо- та ресурсомістким завданням. Тому, для спрощення процесу, другою опцією є використання фреймворків та бібліотек, які надають готові рішення та інструменти для швидкої розробки REST API. Це може значно скоротити час розробки та забезпечити високу якість коду завдяки перевіреним практикам та рішенням, що надаються цими інструментами.

Наприклад, я можу використати популярний фреймворк Django для розробки REST API на мові програмування Python. Django надає потужні засоби для створення високопродуктивних веб-застосунків із вбудованим механізмом обробки запитів REST API. Він має широкий спектр модулів та пакетів, що

дозволяють легко виконувати автентифікацію користувачів, валідацію даних, кешування та інші операції, що часто використовуються в розробці REST API.

Крім Django, існують інші популярні фреймворки та бібліотеки, які можна використовувати для розробки REST API. Наприклад, Flask, який також побудований на мові програмування Python, є легким та гнучким фреймворком, що дозволяє швидко створювати API з мінімальним набором залежностей. Його простота використання і гарна документація роблять його популярним вибором для розробки REST API початківцями та досвідченими розробниками.

Крім того, для інших мов програмування також існують потужні фреймворки для розробки REST API. Наприклад, для JavaScript є Express.js, який забезпечує простоту та гнучкість у розробці API. Laravel для PHP, Ruby on Rails для Ruby та ASP.NET для C# - це лише кілька з численних фреймворків, доступних для створення REST API.

Враховуючи, що я маю більше всього досвіду роботи з мовою програмування Python, серед інших фреймворків найбільш зручним буде використання Django який я описав вище.

Крім вибору фреймворку, розробка REST API також вимагає ретельного проектування архітектури. Важливо правильно визначити роути, моделі даних, контролери та сервіси, які виконують різні операції над даними. Оптимальна організація коду та використання патернів проектування, таких як MVC (Model-View-Controller), можуть спростити розробку та підтримку REST API.

Крім основного функціоналу, такого як оновлення, передача, фільтрація та адаптація даних, розробка REST API також може включати реалізацію додаткових функцій. Наприклад, можливість обробки завдань в фоновому режимі за допомогою черг, надання документації та інше. Django забезпечується широкий спектр можливостей, що включають, але не обмежуються наступними:

Маршрутизація та обробка запитів: Django пропонує зручні засоби для визначення маршрутів та обробки різних типів запитів. За допомогою декораторів, можна легко визначити обробники для методів GET та POST. Django забезпечує зручний інтерфейс для роботи з цими маршрутами, що дозволяє гнучко керувати поведінкою веб-застосунків.

Middleware: Django має підтримку проміжного програмного забезпечення (middleware), яке дозволяє виконувати певні дії при кожному запиті на сервер. За допомогою функції use, розробники можуть визначити middleware, які виконують певні команди або перевірки перед тим, як обробити запит. Це може бути корисно для автентифікації користувачів, логування дій або перевірки прав доступу.

Прослуховування порту та хоста: Django дозволяє визначити порт та хост, на яких слід виконувати веб-застосунок, використовуючи функцію listen. Це дозволяє зручно налаштовувати сервер і забезпечує можливість вибору оптимальних налаштувань для конкретного середовища.

Надаючи ці та багато інших функцій, Django дозволяє розробникам швидко створювати високоякісні веб-застосунки з мінімальними витратами часу і зусиль. Крім базових функціональних можливостей, фреймворк Django також надає багато додаткових компонентів і модулів, які спрощують розробку веб-застосунків. Ось ще деякі з них:

- ORM (Об'єктно-реляційне відображення): Django має вбудовану систему ORM, яка дозволяє розробникам працювати з базою даних безпосередньо з коду Python, уникнувши необхідності писати складні SQL-запити. Це значно полегшує розробку та підтримку бази даних, а також забезпечує більшу переносимість веб-застосунків.
- Шаблонізація: Django має потужну систему шаблонів, яка дозволяє розробникам легко створювати вигляд веб-сторінок. За допомогою шаблонів Django можна впроваджувати різноманітні функціональність,

вставляти дані з бази даних, працювати з формами та здійснювати багато інших операцій, що дозволяє створювати динамічні та привабливі веб-інтерфейси.

- Аутентифікація та авторизація: Django надає вбудовану систему аутентифікації та авторизації, що дозволяє легко управляти користувачами, їх правами доступу та забезпечувати безпеку веб-застосунків. За допомогою вбудованих модулів Django можна легко реалізувати функції реєстрації користувачів, входу, відновлення паролю та багато інших аспектів ідентифікації.

Використання Django дозволяє розробникам писати код на мові Python, що забезпечує зрозумілість і читабельність програмного коду. Python є однією з найпопулярніших мов програмування, що дозволяє залучити багато розробників до проекту та знайти велику кількість готових рішень та документації.

Одним з ключових принципів Django є концепція "зроби один раз і використовуй скрізь" (DRY - Don't Repeat Yourself). Це означає, що Django надає зручні інструменти для використання шаблонів, які можна повторно використовувати на різних сторінках сайту. Це забезпечує однаковий зовнішній вигляд і поведінку всього застосунку, що робить його зручним та привабливим для користувачів.

Django також надає потужну систему управління базами даних, що дозволяє зберігати та маніпулювати даними. Фреймворк надає вбудовану підтримку для різних СУБД, таких як PostgreSQL, MySQL, SQLite та багатьох інших. Це дозволяє розробникам використовувати найефективніші і надійні інструменти для зберігання даних залежно від вимог проекту.

Окрім того, Django має вбудовану систему аутентифікації та авторизації, що дозволяє легко і безпечно керувати доступом користувачів до різних частин

застосунку. За допомогою Django можна легко налаштувати ролі та права доступу, що забезпечує високий рівень безпеки веб-застосунка.

Крім базової функціональності, Django також надає безліч додаткових модулів та розширень, які розширюють можливості фреймворка. Наприклад, модуль Django REST Framework дозволяє швидко побудувати API для вашого застосунку, що дозволяє інтегрувати його з іншими сервісами та сторонніми додатками.

Також варто відзначити, що Django має активну спільноту розробників, що постійно внесуть нові покращення та розробляють додаткові розширення для фреймворка. Це означає, що ви можете легко знайти допомогу та підтримку у разі потреби.

Загалом, використання Django дозволяє значно прискорити процес розробки веб-застосунків, забезпечуючи широкий набір інструментів та рішень для будь-яких потреб проекту. Цей фреймворк підходить як для маленьких невеликих проектів, так і для складних та масштабних систем. З ним ви можете будувати потужні та функціональні веб-застосунки, які задовольнять навіть найвимогливіших користувачів.

2.2 Вибір бази даних

Необхідністю для роботи мого застосунку є база даних, де буде зберігатися необхідна для клієнта інформація, в моєму випадку це дані:

- Користувачі та їхні ролі
- Дані користувачів
- Інформація про вакансії та відгуки на ці вакансії
- Чат

Організація даних у реляційних базах даних є важливою складовою ефективного управління і обробки інформації. Для цього дані структуруються у вигляді таблиць, що дозволяє зберігати та організовувати їх відповідно до потреб бізнесу. Однак, зі зростанням обсягу даних та складності взаємозв'язків між таблицями виникають проблеми з швидкістю обробки запитів.

Відповідно до цього були розроблені «нормальні форми», що встановлюють правила для ефективного побудови та групування таблиць у реляційних базах даних. Нормалізація даних забезпечує оптимальну структуру бази даних, спрощує роботу з ними та полегшує виконання запитів.

У нормальних формах бази даних, дані розподіляються за різними таблицями, залежно від їх характеристик та взаємозв'язків. В першій нормальній формі (1НФ) кожна таблиця містить лише атомарні значення, не розділені на підрядки. Це дозволяє уникнути повторювання даних та забезпечує простоту обробки і пошуку інформації.

Друга нормальна форма (2НФ) вимагає, щоб кожен неключовий атрибут залежав від цілого первинного ключа, а не від частини його. Це сприяє більшій гнучкості та ефективності запитів.

Третя нормальна форма (3НФ) додає ще одну вимогу до другої нормальної форми, а саме, щоб кожен неключовий атрибут залежав тільки від первинного ключа таблиці. Це допомагає забезпечити уникнення аномалій оновлення, видалення та вставки даних.

На вищому рівні досконалості ми маємо четверту нормальну форму (4НФ), яка вводить поняття "мультимножини" і дозволяє уникнути аномалій, пов'язаних з залежностями між неключовими атрибутами. В 4НФ таблиці розбиваються на більш малі підмножини, які можуть бути пов'язані з допомогою спеціальних зв'язків.

П'ята нормальна форма (5НФ) піднімає поняття мультимножин на ще вищий рівень, вводячи інформацію про залежності між спільними підмножинами атрибутів. Це дозволяє ефективно використовувати спільну інформацію та зменшує зайві дублікати даних.

Остання в списку нормальна форма - доменно-ключова нормальна форма (ДКНФ) - встановлює додаткові обмеження на зв'язки між даними та їх типами. В цій нормальній формі домени даних встановлюються на основі ключових властивостей, що дозволяє краще контролювати та оптимізувати роботу зі значеннями.

Застосування нормальних форм в реляційних базах даних сприяє створенню структурованих та ефективних систем зберігання та обробки даних. Кожна нормальна форма вносить певні обмеження і вимоги, що допомагають уникнути аномалій та забезпечити високу якість даних. Оптимальна організація даних у базі даних дозволяє підвищити продуктивність системи, зменшити зайвість та дублювання даних, а також полегшити роботу з ними для користувачів та розробників.

Приклад данних зображено на рисунку 2.1:

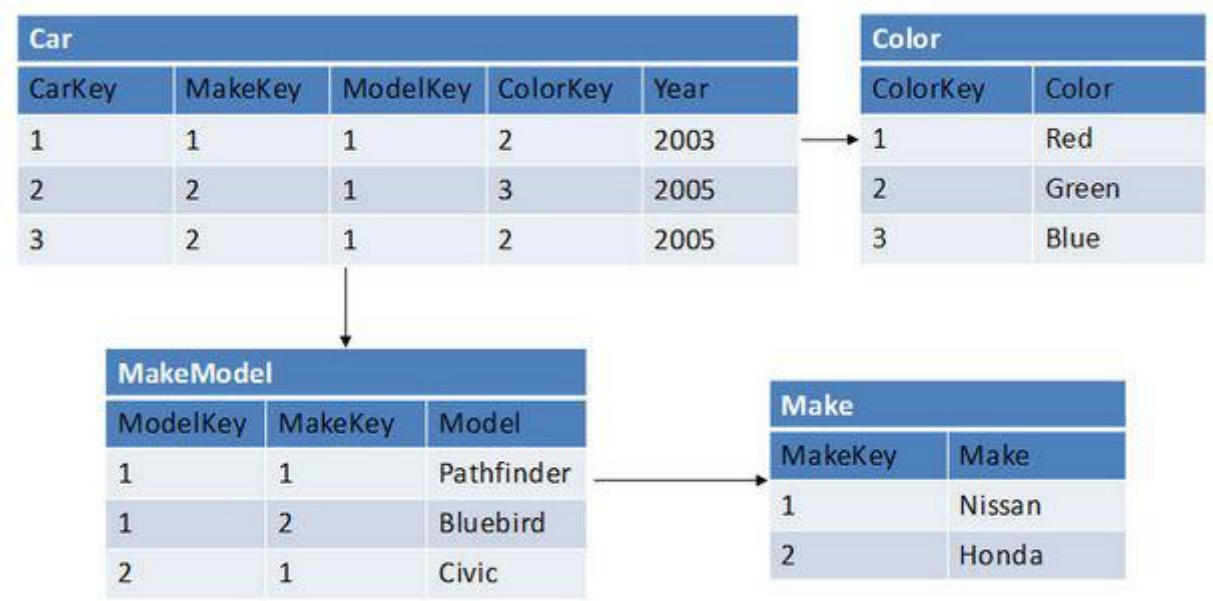


Рисунок 2.1 – Таблиця PostgreSQL

У сучасному світі існує широкий спектр систем керування базами даних (СКБД), але серед них особливе місце займають реляційні системи, такі як PostgreSQL, MySQL, Microsoft SQL Server та Oracle Database. Ці впливові та надійні СКБД використовуються в різних галузях, від малих бізнесів до великих корпорацій.

Нереляційні бази даних, також відомі як NoSQL (Not Only SQL), представляють собою потужні механізми зберігання та роботи з даними, які відрізняються від традиційних таблиць-зв'язків, що використовуються в реляційних базах даних. Цей новий підхід до управління даними набуває все більшої популярності в сучасному світі і знаходить широке застосування у різних сферах.

Основною перевагою нереляційних баз даних є їх гнучкість та масштабованість. Вони дозволяють зберігати великі обсяги даних і ефективно опрацьовувати їх навіть при високих навантаженнях. Крім того, нереляційні бази даних можуть бути легко розширені, що дозволяє їм адаптуватися до зростаючих потреб бізнесу.

У світі нереляційних баз даних можна виділити кілька класів моделей даних, які пропонують різні підходи до зберігання та організації даних. Один з таких класів – це ключ-значення. Популярними системами баз даних цього типу є Aerospike, ArangoDB та Redis. Вони пропонують простий спосіб зберігання даних у вигляді ключів та значень, що робить їх ідеальними для швидкого доступу до інформації.

Інший клас нереляційних баз даних - це стовпчикові системи, такі як Cassandra, VerITca та Druid. Вони використовують орієнтовану на стовпчики модель даних, де дані групуються у стовпчикові сітки для полегшення операцій аналізу і фільтрації даних.

Графові бази даних, такі як OrientDB, ArangoDB та MarkLogic, зосереджені на зв'язках між об'єктами даних і використовують графову структуру для представлення залежностей між ними. Це особливо корисно для аналізу соціальних мереж, рекомендаційних систем, а також у галузі біології і транспортних мереж.

Ще одним типом нереляційних баз даних є документ-орієнтовані системи, такі як ArangoDB, MongoDB і OrientDB. Вони зберігають дані у вигляді документів, які можуть бути представлені у форматі JSON або XML. Цей підхід дозволяє гнучко зберігати структуровані та неструктуровані дані, що робить їх популярними для розробки веб-застосунків, систем керування контентом та аналітичних інструментів.

Окрім цього, існує також мульти-модельний підхід до нереляційних баз даних, який поєднує різні моделі даних для забезпечення більш широкого спектру можливостей. Такі системи, як MarkLogic, ArangoDB та OrientDB, надають засоби для зберігання і обробки даних з використанням різних підходів, що дозволяє вибрати найбільш ефективну модель для конкретного типу даних.

Важливо зауважити, що класифікація нереляційних баз даних не є жорсткою і чіткою, оскільки багато систем можуть використовувати комбінацію різних підходів. Наприклад, ArangoDB може бути використана як база даних ключ-значення, документ-орієнтована база даних або графова база даних залежно від вимог конкретного проекту.

Приклад документу mongoDB зображено на рисунку 2.2:

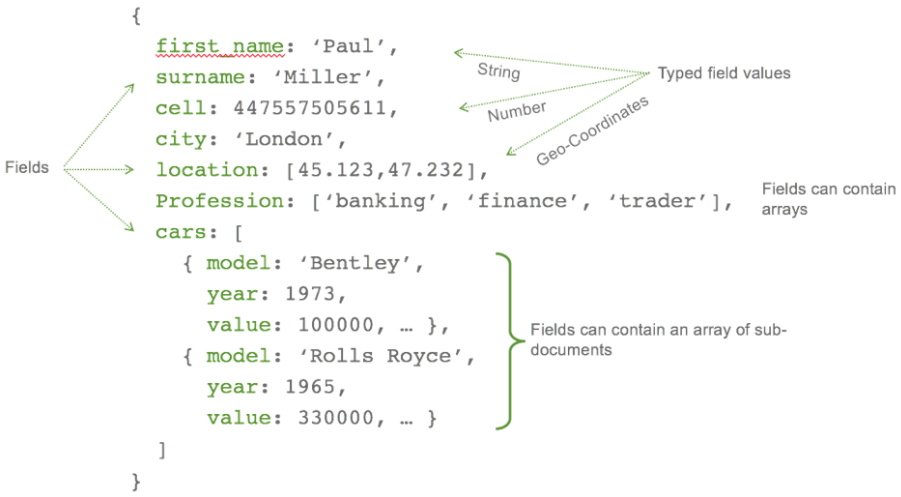


Рисунок 2.2 − Документ MongoDB

При виборі системи керування базами даних для мого веб-сервісу я ставив перед собою завдання знайти оптимальне рішення, яке б задовольнило мої потреби в середовищі розробки з використанням фреймворка Django. Після детального аналізу різних варіантів я вирішив обрати Postgres, оскільки вона прекрасно поєднується з Django і надає багато переваг.

Postgres - це потужна система керування базами даних, яка підтримує широкий спектр функцій і можливостей. Її реляційна модель дозволяє ефективно організувати дані у вигляді таблиць зі зв'язками між ними, що забезпечує надійність і цілісність інформації. Безперечною перевагою Postgres є його довга історія розвитку і активна спільнота користувачів, що робить його надійним і стабільним рішенням.

Використання Postgres у поєднанні з Django виявляється вкрай зручним і простим. Django надає широкий набір інструментів і бібліотек, які полегшують роботу з базою даних. Інтеграція з Postgres дозволяє легко створювати моделі даних, виконувати запити і здійснювати міграції. Крім того, Django автоматично створює SQL-запити, що дозволяє зосередитися на розробці функціональності веб-сервісу, не витрачаючи багато часу на написання складних SQL-запитів.

Окрім цього, Postgres надає широкі можливості для оптимізації продуктивності. Він підтримує індексування, що дозволяє прискорити виконання запитів, особливо в разі роботи з великими обсягами

2.3 Single-page application

Крім серверної складової, мій застосунок передбачає також наявність клієнтської частини, яка включатиме в себе роботу з ключовими складовими фронтенд розробки, такими як HTML, CSS та JavaScript. При проектуванні клієнтської частини мені довелося врахувати, що клієнт постійно отримуватиме нові дані, тому стандартна модель фронтенду, коли користувач отримує цілі сторінки, не підходить. Використання цієї моделі призводило б до постійного перезавантаження вікна браузера та втрати даних після кожної взаємодії з ресурсом. З цією метою було вирішено використати іншу модель, якою є односторінкова аплікація (SPA).

Односторінкова аплікація фактично дозволяє користувачеві взаємодіяти з однією основною сторінкою, на якій динамічно змінюються лише певні елементи. Це дозволяє переписувати окремі частини сторінки динамічно, замість того, щоб отримувати нові сторінки з сервера кожного разу. Цей підхід широко застосовується в таких популярних веб-застосунках, як Facebook, Trello, YouTube та Gmail.

Односторінкові аплікації дуже ефективно використовують модель REST та маршрутизацію, оскільки доступ до контенту здійснюється за допомогою різних параметрів у адресному рядку. Це дозволяє отримувати доступ до конкретних статичних частин ресурсу за простим запитом. Такий підхід дозволяє користувачам легко ділитися інформацією один з одним, незважаючи на те, що ресурс все одно представляє їм одну сторінку.

Односторінкові аплікації (SPA) зазвичай працюють швидше, ніж багатосторінкові веб-сайти, оскільки динамічна зміна контенту відбувається з вищою швидкістю, ніж обмін повноцінними сторінками. Винятком може бути стартова розмітка сторінки, де часто знаходиться скрипт всього фреймворку, необхідного для функціонування SPA. Таким чином, при використанні SPA може виникати невелика затримка при завантаженні початкової структури сторінки.

Розробка односторінкової аплікації з нуля є цілком можливою, але для спрощення процесу були створені різноманітні фреймворки. Найпопулярнішими з них є Django, Flask та FastAPI.

- Django - це потужний фреймворк розробки веб-застсоунків на мові Python, який надає широкі можливості для створення як серверної, так і клієнтської частини застосунку. Він забезпечує розширюваність, надійність та безпеку застсоунків, а також має багато вбудованих інструментів для роботи з базами даних, аутентифікації та авторизації користувачів.
- Flask - ще один популярний фреймворк на мові Python, який має легковагу архітектуру та простий у використанні синтаксис. Він надає базовий набір функцій для створення веб-застосунків та дозволяє легко розширювати їх за допомогою різних плагінів та розширень.
- FastAPI - це новий, але швидко набираючий популярність фреймворк для розробки веб-застосунків на Python. Його особлив енність полягає у високій швидкості обробки запитів та автоматичній генерації документації за допомогою стандарту OpenAPI. FastAPI забезпечує ефективну роботу з асинхронним кодом та дозволяє легко розгортати застсоунки на різних платформах.

При розробці SPA з використанням фреймворків, розробникам надаються інструменти та бібліотеки, які спрощують процес створення клієнтської частини. Наприклад, для роботи зі структурою сторінки та її виглядом використовуються

HTML, CSS і JavaScript. HTML відповідає за структуру сторінки, CSS - за її стилізацію, а JavaScript - за взаємодію з користувачем та динамічні зміни.

2.3.1 Django

Django - це потужний веб-фреймворк, написаний на мові програмування Python. Він забезпечує розробку високоякісних веб-застосунків швидко, легко та ефективно. Django був створений з метою забезпечити простоту у використанні та зручність для розробників, дозволяючи їм зосередитися на розробці функціональності, а не на вирішенні загальних проблем веб-розробки.

Основні можливості Django:

- Вбудована адміністративна панель: Django надає готову адміністративну панель, що дозволяє легко створювати, змінювати та видаляти дані з бази даних без написання власного коду.
- ORM (об'єктно-реляційне відображення): Django має потужний шар ORM, що дозволяє взаємодіяти з базою даних за допомогою об'єктно-орієнтованого підходу.
- URL-маршрутизація: Django надає зручний спосіб визначати URL-шаблони для різних сторінок вашого веб-застосунку.
- Шаблонізація: Django має вбудовану систему шаблонів, що дозволяє розділити логіку розробки та представлення.
- Безпека: Django забезпечує різні заходи безпеки, включаючи захист від хакерських атак, обробку форм, автентифікацію та авторизацію користувачів.
- Міжнародна підтримка: Django підтримує міжнародну локалізацію і міжнароднізацію, що дозволяє створювати веб-застосунки для різних мов та регіонів.

Деякі з плюсів Django:

- Прискорена розробка: Django надає зручність та швидкість розробки завдяки великій
- Велика спільнота: Django є дуже популярним фреймворком з великою та активною спільнотою розробників. Це означає, що ви можете легко знайти документацію, розширення, готові модулі та допомогу від інших розробників.
- Розширюваність: Django надає багато готових рішень та розширень, які допомагають спростити розробку веб-застосунків. Ви можете використовувати розширення, які вже розроблені спільнотою, або створити власні розширення для ваших потреб.
- Тестування: Django має вбудовану систему для тестування, яка допомагає перевірити, чи працює ваш код правильно. Це спрощує процес розробки та дозволяє швидко виявляти та виправляти помилки.
- Сумісність з іншими технологіями: Django можна легко поєднувати з іншими технологіями та бібліотеками, такими як JavaScript-фреймворки (наприклад, React або Vue.js) або бази даних, такі як PostgreSQL або MySQL.

Незважаючи на багато переваг, Django також має деякі мінуси:

- Велика складність для початківців: Django може бути складним для освоєння для новачків у веб-розробці або програмуванні загалом. Він має досить великий набір функцій та концепцій, які потребують часу для вивчення.

- Обмежена гнучкість: Django пропонує стандартизований підхід до веб-розробки, що може бути обмежувальним для проектів зі специфічними вимогами або нетрадиційною архітектурою.

Великий розмір проекту: Django включає в себе багато компонентів т а, що може призвести до великого розміру проекту. Це може стати проблемою, якщо ви маєте обмежені ресурси або потребуєте легкості та компактності.

Незважаючи на ці мінуси, Django є одним з найпопулярніших веб-фреймворків у світі. Він має широку базу користувачів та активну спільноту розробників, що призводить до швидкого розростання екосистеми Django. Багато великих компаній та проектів використовують Django для своїх веб-застосунків, що свідчить про його надійність та стабільність.

Українська спільнота Django також є досить активною. Є багато розробників та компаній, які використовують Django для своїх проектів в Україні. Це забезпечує доступ до ресурсів, навчальних матеріалів та можливостей для спілкування зі спільнотою розробників.

2.3.2 Flask

Flask - це легкий та гнучкий веб-фреймворк для розробки веб-застосунків на мові програмування Python. Він зосереджується на простоті та мінімалістичному підході, дозволяючи розробникам вибирати лише необхідні компоненти для своїх проектів. Flask надає базовий функціонал, але дозволяє розширити його за допомогою різних розширень.

Основні можливості Flask:

- Маршрутизація: Flask дозволяє визначити URL-шаблони для різних сторінок вашого веб-застосунку.
- Шаблонізація: Flask має просту та гнучку систему шаблонів, яка дозволяє відокремити логіку розробки від представлення.

- Інтеграція з ORM: Flask легко поєднується з різними ORM-бібліотеками, такими як SQLAlchemy, дозволяючи вам працювати з базою даних в зручний спосіб.
- Розширення: Flask має широкий вибір розширень, які допомагають вам додати більшу функціональність до вашого застосунку, наприклад, роботу з формами, автентифікацію користувачів або інтеграцію з іншими сервісами.
- Підтримка тестування: Flask надає зручні інструменти для тестування вашого коду та веб-застосунків.

  Деякі з плюсів Flask:

- Легкість вивчення: Flask має просту структуру та мінімалістичний дизайн, що дозволяє швидко ознайомитися з фреймворком та почати розробку.
- Гнучкість: Flask дозволяє розробникам вибирати лише необхідні компоненти та розширення для своїх проектів, що дає більшу гнучкість та контроль над застосунком.
- Широка підтримка спільноти: Flask має активну та розмаїту спільноту розробників, яка надає безліч ресурсів, документації, розширень та підручників. Ви можете швидко знайти відповіді на свої питання або отримати підтримку від інших розробників.
- Швидкість: Завдяки своєму мінімалістичному підходу, Flask пропонує швидку роботу та високу продуктивність. Він має менше накладних витрат у порівнянні з більш великими фреймворками, що дозволяє побудувати швидкий та ефективний веб-застосунок.

Інтеграція з іншими технологіями: Flask добре поєднується з іншими технологіями та бібліотеками Python. Ви можете використовувати його в

поєднанні з різними базами даних, фронтенд-фреймворками або іншими сервісами.

Щодо недоліків Flask:

- Недостатня "батарейка в комплекті": Flask надає базовий функціонал, але для деяких складних завдань вам може знадобитися додаткове програмне забезпечення або розширення.
- Більша відповідальність розробника: Оскільки Flask не має вбудованих компонентів, розробник має відповідальність вибрати правильні розширення та інструменти для свого проекту.
- Відсутність жорсткого стандарту: У Flask відсутній жорсткий стандарт структури проекту, що може призвести до розбіжностей у стилях та організації коду між різними проектами.

Незважаючи на це, Flask є дуже популярним фреймворком, особливо серед професійних веб-розробників, які цінують його гнучкість, швидкість та простоту використання. Flask також має велику популярність серед початківців, які хочуть швидко розпочати розробку своїх веб-застосунків та навчитися основам веб-розробки.

Українська спільнота Flask також є активною та зростаючою. Багато розробників в Україні використовують Flask для своїх проектів, а також організовують зустрічі, конференції та події для обміну досвідом та підтримки.

Загалом, Flask - це легкий та гнучкий фреймворк, який дозволяє швидко розробляти веб-застосунки на мові програмування Python. Він підходить для проектів будь-якого розміру, починаючи від простих односторінкових застосунків до складних веб-платформ. Flask надає вам контроль над вашим проектом та дозволяє використовувати лише необхідний функціонал, що робить його привабливим для розробників з різним рівнем досвіду та вимогами.

2.3.3 FastAPI

FastAPI - це сучасний веб-фреймворк для швидкої розробки API на мові програмування Python. Він пропонує високу продуктивність, масштабованість та простоту використання. FastAPI побудований на базі типів та асинхронності Python, що дозволяє забезпечити швидку обробку запитів та ефективне використання ресурсів.

Основні можливості FastAPI:

- Швидкість: FastAPI використовує підхід, що спирається на веб-сервер Starlette та бібліотеку PydanITc для роботи з типами даних. Це дозволяє досягти високої продуктивності та швидкої обробки запитів.
- Асинхронність: FastAPI підтримує асинхронні запити, що дозволяє ефективно працювати з багатопотоковими задачами та забезпечує високу масштабованість застосунків.
- Валідація та автоматична документація: FastAPI використовує бібліотеку PydanITc для автоматичної валідації та серіалізації даних. Він також генерує інтерактивну документацію Swagger, що спрощує роботу з API та дозволяє швидко розробляти його.
- Захист: FastAPI надає вбудовану підтримку автентифікації та авторизації, дозволяючи легко захистити ваші API від несанкціонованого доступу.
- Інтеграція з базами даних: FastAPI підтримує різні бази даних та ORM, такі як SQLAlchemy та Tortoise ORM, дозволяючи легко працювати з базами даних у вашому застосунку.

Деякі з плюсів FastAPI:

- Швидкість та продуктивність: FastAPI пропонує швидку обробку запитів та масштабованість, що робить його ідеальним вибором для високонавантажених проектів, де важлива швидкість роботи.

- Асинхронність: Завдяки підтримці асинхронного програмування, FastAPI дозволяє ефективно обробляти багатопотокові запити та використовувати ресурси більш ефективно.
- Автоматична валідація та документація: FastAPI використовує PydanITc для валідації даних та генерує інтерактивну документацію Swagger, що спрощує роботу з API та покращує комунікацію між розробниками.
- Інтеграція з базами даних: FastAPI підтримує різні ORM та бази даних, що дозволяє легко працювати з даними та забезпечує гнучкість вибору технологій.
- Легкість використання: FastAPI має чистий та простий синтаксис, який сприяє швидкому розгортанню проекту та дозволяє швидко створювати потужні API.

  Проте, є кілька недоліків, які слід враховувати:

- Вимоги до версії Python: FastAPI вимагає версію Python 3.7 або вище, що може бути обмеженням для деяких проектів, які використовують старші версії мови.
- Навчання: Оскільки FastAPI базується на типах та асинхронному програмуванні, може бути потрібно трохи часу та зусиль для освоєння цих концепцій, особливо для початківців.

FastAPI стає все більш популярним серед розробників Python, особливо в галузі веб-розробки та розробки API. Його швидкість, продуктивність та легкість використання роблять його привабливим вибором для проектів різних розмірів та складності. Спільнота розробників FastAPI також швидко зростає. Існує багато ресурсів, таких як документація, підручники, приклади коду та форуми підтримки, які допомагають розробникам знайти відповіді на свої питання та розв'язати проблеми.

FastAPI має широкий спектр застосувань, від створення мікросервісів та простих API до складних застосунків з багатьма ендпоінтами. Він підходить для будь-яких потреб, де необхідна швидкість, ефективність та висока продуктивність.

2.4 Вибір протоколу реалізації

У попередньому розділі я детально розглянув різні архітектурні стилі для реалізації веб-сервісів і після виваженого аналізу вибрав для свого проекту REST API - розповсюджений та добре зарекомендував себе протокол.

Зараз перейду до другого розділу, в якому надам більш докладний опис та плани щодо створення власного бекенду для веб-застосунку. Моя мета - створити ефективну та розширювану інфраструктуру, що забезпечить високу продуктивність та забезпечить масштабованість мого веб-застосунку.

З метою досягнення цієї мети, я планую використовувати передові інструменти та технології. Один з них - фреймворк Django, який дозволить мені розробляти бекенд на мові програмування Python. Django володіє потужними функціональними можливостями, що спрощують розробку та підтримку веб-застосунків. Я збираюся використовувати його для створення та керування моделями даних, маршрутизації запитів, обробки авторизації та аутентифікації, а також для реалізації логіки мого веб-застосунку.

Крім того, я планую використовувати базу даних PostgreSQL для зберігання і управління даними мого веб-застосунку. PostgreSQL - це потужна та надійна реляційна база даних, яка надає широкі можливості для роботи з даними, включаючи складні запити, індексацію та транзакційну безпеку. Використання PostgreSQL допоможе мені забезпечити швидкий та надійний доступ до даних мого веб застосунку.

Висновки до розділу

В другій частині мого дослідження було проведено детальний аналіз сучасних архітектурних рішень, що успішно використовуються при створенні веб-

застосунків. Цей аналіз охопив різноманітні аспекти, такі як масштабованість, продуктивність та надійність.

Для побудови серверної частини веб-застосунку ми вибрали потужний та надійний фреймворк Django. Його популярність серед розробників свідчить про його ефективність та широкі можливості. Django надає зручний інтерфейс для розробки серверної частини, що дозволяє зосередитися на реалізації бізнес-логіки та виконанні основних завдань веб-застосунку.

Наслідуючи принцип "все є об'єктом" (англ. "everything is an object"), Django використовує об'єктно-орієнтований підхід до розробки. Це дозволяє розробникам легко створювати повторно використовувані компоненти, що прискорює процес розробки та забезпечує високу якість коду.

Окрім того, для реалізації клієнтської частини веб-застосунку ми обрали Django templates. Цей інструмент дозволяє нам створювати елегантний та динамічний користувацький інтерфейс, який легко змінюється та адаптується до потреб користувачів.

Django templates забезпечує потужну систему керування шаблонами, яка дозволяє відокремлювати логіку та представлення. Це спрощує процес розробки та підтримки, оскільки розробникам не потрібно втручатися в код серверної частини для внесення змін до інтерфейсу користувач , оскільки всі зміни можна внести безпосередньо в шаблони.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ ДЛЯ ПОШУКУ РОБОЧИХ МІСЦЬ З ІТ СПЕЦІАЛЬНОСТЕЙ

3.1 Структура бази даних

Для початку потрібно підготувати саму базу даннтих та таблиці для неї. Створимо базу даних під назвою cq_db. Всередині неї створимо декілька таблиць (загалом їх буде 11).

У вас мене декілька таблиць, проте однією з найважливіших є таблиця, яка описує користувача. Цей користувач має зв'язок за зовнішнім ключем до двох інших таблиць, які містять поля для кандидатів і співробітників відповідно.

Таблиця кандидатів також має кілька інших зв'язків зі сторонніми таблицями. Наприклад, вона може мати зв'язок з таблицею категорій роботи, що дозволяє користувачу вказати свій досвід у певній сфері діяльності. Це дає можливість знайти кандидатів з певним досвідом або кваліфікацією у конкретній галузі.

Таблиця співробітників також має свої зв'язки з іншими таблицями. Наприклад, кожен співробітник прив'язаний до певної компанії, що дозволяє відстежувати, в якій компанії працює кожен співробітник.

Після визначення структури бази даних ви можете переходити до наступного етапу - розробки сервера веб-застосунку. Цей сервер буде відповідальний за обробку запитів і надання доступу до бази даних користувачам через веб-інтерфейс. Ви можете розробити сервер за допомогою різних технологій, таких як Node.js, Python/Django, Ruby on Rails тощо. Основна мета сервера - забезпечити ефективний та безперебійний доступ до даних користувачам і забезпечити правильну обробку запитів до бази даних згідно з встановленою структурою. Структура бази данних з її ві дносинами(рисунок 3.1)

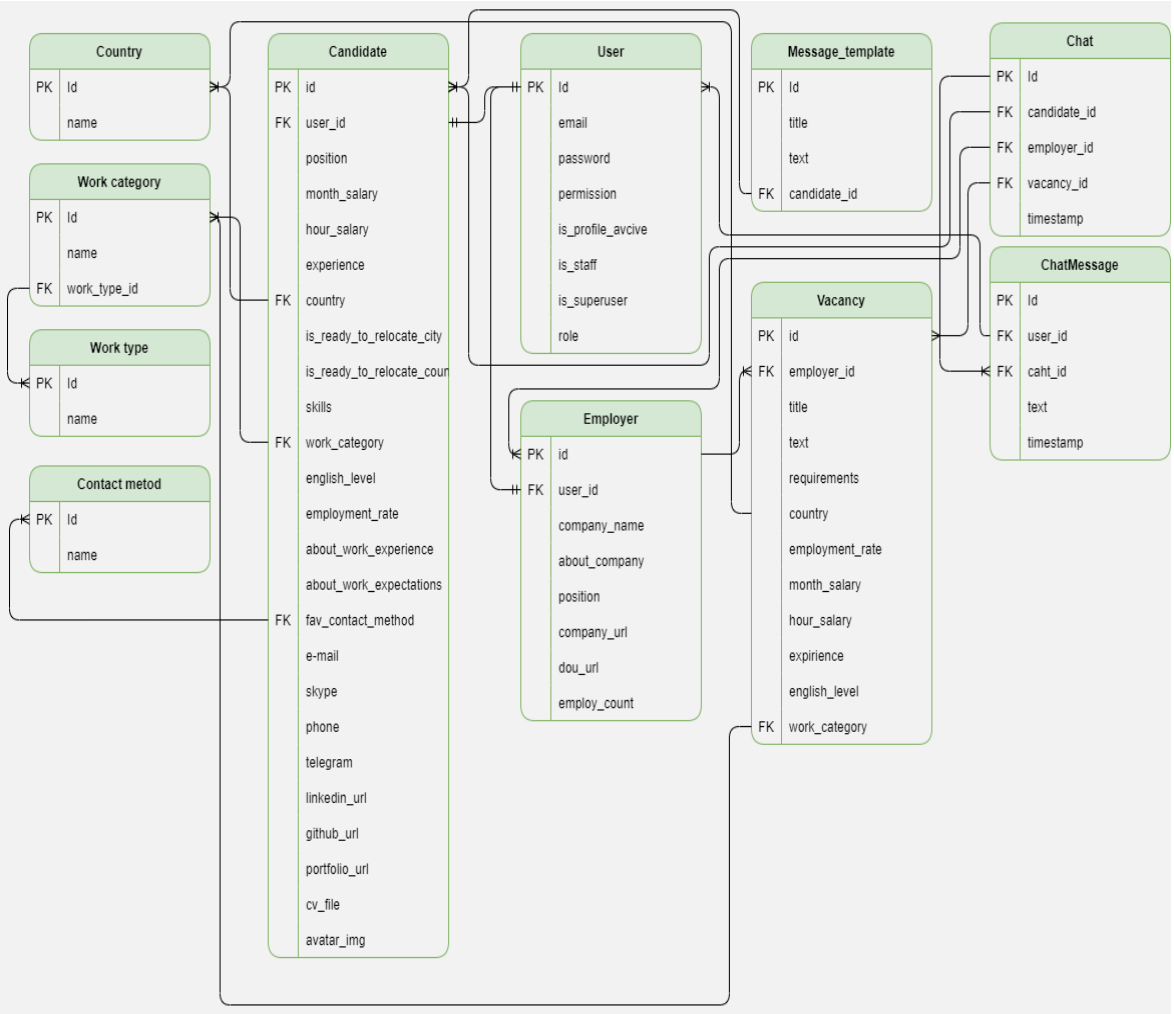


Рисунок 3.1 − Структура бази данних

Веб-застосунок може також містити інші функціональні можливості, такі як автентифікація користувачів, створення нових записів у базі даних, відображення розширеної інформації про користувачів і т. д. Залежно від ваших потреб і специфікацій проекту, ви можете розширити функціональність веб-застосунку за бажанням.

Загалом, розробка сервера веб-застосунку є важливим кроком у створенні повноцінної системи, яка дозволить ефективно управляти даними про користувачів, кандидатів і співробітників. Це дозволить забезпечити гладку роботу вашого рекрутингового або управлінського процесу та покращити ефективність вашої компанії.

3.2 Розробка бекенд-частини веб-застосунку

Django - це веб-фреймворк, розроблений на мові програмування Python, і він використовує патерн проектування Model-View-Controller (MVC). Патерн MVC є одним із найпоширеніших підходів до організації коду у веб-застосунках і дозволяє розділити логіку програми на три компоненти: модель (Model), представлення (View) і контролер (Controller).

Модель (Model) відповідає за управління даними і бізнес-логікою програми. Вона визначає структуру даних, зберігає і отримує дані з бази даних, валідує їх і виконує необхідні операції з даними.

Представлення (View) відповідає за відображення даних користувачу. Воно визначає, як дані повинні бути відображені на веб-сторінці або іншому інтерфейсі користувача. Представлення отримує дані з моделі і передає їх у відповідному форматі для відображення.

Контролер (Controller) відповідає за обробку запитів користувача і взаємодію з моделлю та представленням. Він отримує запити від користувача, виконує необхідні операції з моделлю, обробляє дані і передає їх у відповідному форматі в представлення.

Застосування патерна MVC в Django дозволяє розділити логіку програми на окремі компоненти, що спрощує розробку і підтримку коду. Крім того, Django надає багато готових інструментів і бібліотек для роботи з базами даних, аутентифікацією користувачів, обробкою форм і багато іншого, що полегшує процес розробки веб-застосунків.

Загалом, використання патерна MVC у Django допомагає створювати доброорганізовані. Спрощена схема роботи цього патерну в реалізації Django зображено на рис 3.3.

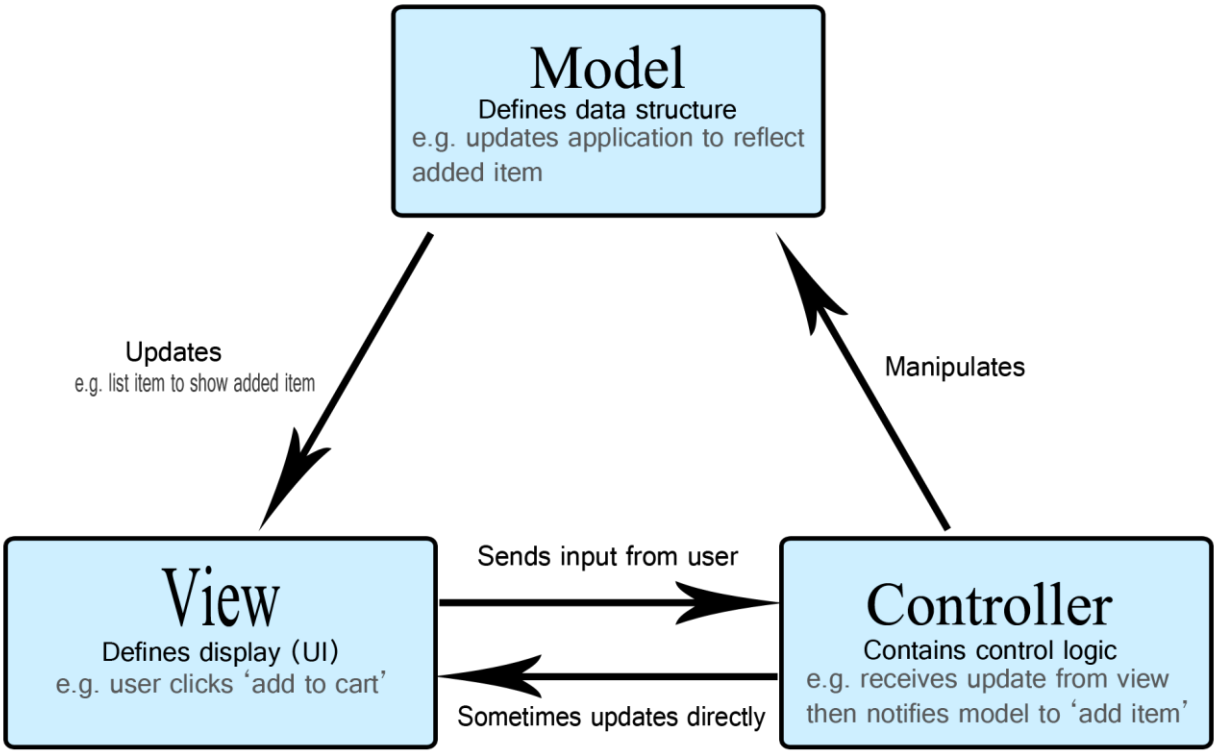


Рисунок 3.2 − Спрощена схема роботи патерну MVC

В нашому випадку частина представлення буде реалізована у фронтендчастині тому на сервері нам треба буде реалізувати лише контролер, моделі та маршрутизацію.

3.2.1 Реалізація моделей

Для початку потрібно за допомогою засобів пакету mongoose створити моделі, які будуть працювати з визначеною структурою бази данних PostgreSQL.

Модель User представлена в ДОДАТКУ А.

Модель Vacancy представлена в ДОДАТКУ Б.

3.2.2 Створення маршрутів

Маршрутизація в Django відповідає за визначення того, які URL-адреси повинні бути пов'язані з якими уявленнями (views) і функціями. Django використовує потужну систему маршрутизації, яка дає змогу легко визначити маршрути для вашого веб-застосунку.

Основний компонент маршрутизації в Django - це файл urls.py, який знаходиться в кореневій директорії вашого проекту Django. У цьому файлі визначаються шаблони URL-адрес і їхні пов'язані подання. Давайте розглянемо його докладніше.

Нижче наведено приклад того, як я розбиваю маршрути на більш дрібні частини в рамках основної структури проєкту. Для цього я використовую внутрішні додатки Django. Кожен з цих додатків також має свої власні маршрути, якщо це необхідно. Кожен маршрут містить шлях і вказує на клас або функцію, яка обробляє запити, що надходять по цьому маршруту.

Така організація маршрутів дозволяє розділити функціональність проєкту на менші компоненти, що полегшує розробку та утримання коду. Кожен внутрішній застосунок може мати свою власну логіку та функціональність, яка обробляє певний тип запитів. Це дозволяє розподілити завдання між командами розробників та підтримувати структурованість проєкту. (рис. 3.4)

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('profile/', include('user_profile.urls')),
    path('vacancy/', include('vacancy.urls')),
    path('chat/', include('chat.urls')),
    path('', include('user_auth.urls')),
]
```

Рисунок 3.3 − Спрощена схема роботи патерну MVC

```
urlpatterns = [
    path('vacancy-list/', VacancyList.as_view(), name='vacancy-list'),
    path('vacancy-filter/', VacancyFilter.as_view(), name='vacancy-filter'),
    path('vacancy-detail/<int:pk>/', VacancyDetail.as_view(), name='vacancy-detail'),
    path('vacancy-favorite/<int:pk>/', VacancyFavorite.as_view(), name='vacancy-favorite'),
    path('vacancy-favorite/', FavoriteVacancyList.as_view(), name='list-vacancy-favorite'),
    path('vacancy-by-profile/', VacancyByProfile.as_view(), name='vacancy-by-profile'),

    path('vacancy-create/', VacancyCreate.as_view(), name='vacancy-create'),

    path('employer-vacancies-list/', EmployerVacanciesList.as_view(), name='employer-vacancies-list')

]
```

Рисунок 3.4 – Спрощена схема роботи патерну MVC

Зразок маршрутів, які я надав, демонструє організацію маршрутів у моєму проєкті. Я використовую модуль path з Django для визначення шляхів та класів або функцій, які обробляють запити.

Основна частина маршрутів визначається у змінній urlpatterns, яка містить список шляхів. Кожен шлях вказується за допомогою функції path(). У моєму випадку я використовую різні шляхи, включаючи 'admin/', 'profile/', 'vacancy/', 'chat/' та ''.

Я також використовую функцію include() для включення інших файлів з маршрутами, які знаходяться у відповідних додатках. Наприклад, 'user_profile.urls' включає маршрути з файлу urls.py додатку 'user_profile'.

Додаткові маршрути, пов'язані з 'vacancy/', визначені у окремому списку маршрутів urlpatterns. Кожен маршрут вказує на певний клас за допомогою функції as_view() або на функцію, яка обробляє запит. Наприклад, 'vacancy-list/' вказує на клас VacancyList.as_view().

Використання такої структури маршрутів дозволяє мені логічно групувати функціональність мого проєкту та забезпечує зручний спосіб навігації. Кожен маршрут відповідає за певну дію, що полегшує розробку, підтримку та розширення мого проєкту в майбутньому.

3.2.2 Отримання даних

Класи ListView і RetrieveView в Django використовуються для відображення списків об'єктів і деталей окремого об'єкта відповідно. Давайте розглянемо кожен з них детальніше.

ListView:

- Клас ListView використовується для відображення списків об'єктів з бази даних. Він надає стандартну логіку для витягування об'єктів з бази даних і відображення їх на веб-сторінці. Зазвичай ListView використовується для відображення списку записів моделі у вигляді таблиці або списку елементів.
- Основні функції ListView включають витягування списку об'єктів з бази даних, передачу цього списку до шаблону для відображення та виконання пагінації, яка дозволяє розділити довгі списки на сторінки.

RetrieveView:

- Клас RetrieveView використовується для відображення деталей окремого об'єкта з бази даних. Він надає можливість переглядати окремий об'єкт моделі з докладними даними на окремій сторінці. Зазвичай RetrieveView використовується для відображення деталей статті, користувача, продукту або будь-якого іншого об'єкта.
- Основні функції RetrieveView включають витягування окремого об'єкта з бази даних, передачу цього об'єкта до шаблону для відображення та реагування на запити користувача для редагування, видалення або виконання інших дій з об'єктом.

- Обидва класи можна налаштувати, перевизначивши деякі методи або атрибути, щоб вони відповідали конкретним потребам проєкту. Наприклад, можна налаштувати шаблони, URL-адреси, атрибути моделі, а також методи, які виконують певну логіку перед відображенням або після нього.

Загалом, класи ListView і RetrieveView в Django дозволяють швидко і легко створювати сторінки для відображення списків об'єктів і деталей окремих об'єктів відповідно, забезпечуючи стандартизовану логіку і зручний спосіб роботи з базою даних.

```python
class VacancyList(LoginRequiredMixin, VacanciesDataMixin, UserProfileFormDataMixin, ListView):
    login_url = LOGIN_URL
    redirect_field_name = 'user_profile:my-contact'
    template_name = 'vacancy/vacancy_list.html'
    queryset = Vacancy.objects.all()
    context_object_name = 'vacancies'

    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data()
        context = self.get_custom_context_form_data(old_context=context)
        context['tabs'] = self.vacancies_tabs
        context['selected_tab'] = 'All vacancies'

        return context
```

Рисунок 3.5 – Клас для відображення списку вакансій

3.2.3 Створення даних

Клас CreateView в Django є одним з представлень (views), які використовуються для обробки створення об'єктів у базі даних. Він надає шаблонізований підхід до створення нових об'єктів на основі моделей.

Основна мета CreateView полягає в тому, щоб надати шаблонну форму для створення нових записів у базі даних, так щоб розробникам не потрібно було писати весь код обробки форми вручну. Він автоматично генерує необхідну логіку

для відображення форми, валідації даних, збереження об'єкта та повернення користувача до відповідної сторінки після успішного створення об'єкта.

Основні переваги використання CreateView в Django:

- Шаблонізація: CreateView надає готовий шаблон, який відображає форму для створення об'єктів. Це полегшує розробку, оскільки не потрібно писати шаблони з нуля або вказувати весь HTML-код вручну.
- Автоматична обробка форми: CreateView автоматично обробляє всі етапи, пов'язані з обробкою форми. Це включає валідацію даних, збереження об'єкта в базі даних та повернення користувача до відповідної сторінки.
- Розширення та налаштування: CreateView можна легко розширити та налаштувати для виконання специфічних вимог. Він надає різні методи, такі як form_valid(), які дозволяють виконати додаткову логіку після успішного створення об'єкта.
- Інтеграція з моделями: CreateView пов'язаний з певною моделлю, що дозволяє автоматично створювати об'єкти цієї моделі. Він отримує доступ до полів моделі та дозволяє користувачеві вводити значення для цих полів у формі.

При використанні CreateView, необхідно налаштувати деякі атрибути, такі як model (модель, на основі якої створюється об'єкт), form_class (клас форми, який використовується для відображення та обробки даних форми) та success_url (URL, на який перенаправляється користувач після успішного створення об'єкта).

Крім того, можна перевизначити деякі методи в CreateView для налаштування його поведінки, наприклад, form_valid(), для виконання додаткової логіки перед збереженням об'єкта.

Загалом, клас CreateView є потужним інструментом для швидкого розроблення форм для створення об'єктів у Django та полегшує процес розробки, забезпечуючи автоматичну обробку форм та стандартизовану логіку.

```
class VacancyCreate(LoginRequiredMixin, UserProfileFormDataMixin, CreateView):
    login_url = LOGIN_URL
    redirect_field_name = 'user_profile:my-contact'
    template_name = 'vacancy/create.html'
    form_class = VacancyCreateForm
    context_object_name = 'vacancy'

    def get_context_data(self, **kwargs):
        context = super().get_context_data()
        context = self.get_custom_context_form_data(context)
        return context
```

Рисунок 3.6 – Клас для створення вакансій

Клас CreateView в Django використовується для створення нових об'єктів моделі. У ньому можна налаштувати кілька атрибутів для належного функціонування і відображення сторінки створення об'єкта. Ось детальний опис кожного з цих атрибутів:

login_url:

Цей атрибут використовується для вказівки URL-адреси, на яку користувач буде перенаправлений, якщо він не має прав доступу до сторінки створення об'єкта. Використовується для автентифікації користувача перед доступом до форми створення.

redirect_field_name:

Цей атрибут вказує назву поля в запиті, в якому передається URL-адреса, на яку користувач буде перенаправлений після успішного створення об'єкта.

Використовується для перенаправлення користувача на певну сторінку після створення об'єкта.

template_name:

Цей атрибут вказує шлях до шаблону, який використовується для відображення сторінки створення об'єкта. Ви можете вказати власний шлях до шаблону, якщо потрібно використовувати власний вигляд для сторінки створення.

form_class:

Цей атрибут вказує клас форми, який використовується для відображення та обробки даних форми створення об'єкта. Ви можете вказати свій власний клас форми або використовувати стандартний клас форми Django. Клас форми повинен містити валідацію і збереження даних, пов'язаних з моделлю, з якою працює CreateView.

context_object_name:

Цей атрибут вказує ім'я змінної контексту, яка містить об'єкт форми на шаблоні сторінки створення. Ви можете вказати власне ім'я для змінної контексту, щоб звертатися до неї на шаблоні. За замовчуванням, ім'я змінної контексту буде "form".

Ці атрибути дозволяють налаштувати різні аспекти сторінки створення об'єкта, такі як автентифікація, відображення шаблону, валідація форми та збереження даних. Вони допомагають спростити процес створення об'єктів і забезпечити їх належне функціонування у веб-застосунку Django.

3.2.4 Деплой серверу

Для деплою веб-застосунків на базі Django ідеально підходить хмарна платформа Heroku. Вона надає мені можливість безкоштовно розгортувати мої Django-застосунки у їхньому середовищі з невеликим трафіком.

Щоб розгорнути мій Django-застосунок на Heroku, я повинен виконати кілька кроків. По-перше, мені потрібно мати встановлені Git та Heroku CLI

(Command Line Interface) на моїй локальній машині. Heroku CLI дозволяє мені керувати моїми додатками на Heroku через командний рядок.

Після встановлення Heroku CLI, я можу створити новий проект на Heroku, використовуючи команду heroku create. Ця команда створить новий проект на серверах Heroku та налаштує зв'язок між моїм локальним проектом і сервісом Heroku.

Далі мені потрібно додати свій код до репозиторію Heroku. Це можна зробити за допомогою команди git push heroku main. Перед виконанням цієї команди я маю переконатися, що я вже ініціалізував Git-репозиторій у моєму проекті, виконавши команду git init. Я також можу виключити деякі файли або папки з коміту, створивши файл .giITgnore в кореневій папці мого проекту і перерахувавши там правила для файлів, які не потрібно включати до репозиторію Heroku.

Окрім цього, у мене є можливість налаштувати додаткові параметри, такі як login_url, redirect_field_name, template_name, form_class і context_object_name для мого Django-застосунку на Heroku. Вони допоможуть мені налаштувати автентифікацію, вигляд сторінок, використання форм та змінні контексту відповідно до моїх потреб.

__pypackages__/

.env

.idea

venv

site/

.pytype/

Це виключить всі непотрібні файли з коміту.

3.3 Розробка фронтенд частини веб-застосунку

Команда django-admin startproject використовується для створення нового проекту Django. Вона запускає генератор початкової структури проекту та створює необхідні файли та папки.

Основним етапом роботи команди django-admin startproject є створення кореневої папки проекту і налаштування початкової структури файлів та папок. Після виконання команди створюється нова папка з назвою проекту, яка містить кілька важливих файлів:

manage.py: Це виконуваний файл, який використовується для управління проектом Django. Він дозволяє виконувати команди для розгортання сервера, запуску тестів, міграцій бази даних та багато іншого.

__init__.py: Цей файл позначає папку проекту як пакет Python. Він може залишатися порожнім або містити додатковий код, який виконується при імпортуванні проекту.

setITngs.py: Цей файл містить налаштування проекту Django. В ньому визначаються база даних, статичні файли, шаблони, мови, аутентифікація та інші параметри, необхідні для правильної роботи проекту.

urls.py: Цей файл визначає шляхи (URL) проекту Django. В ньому вказуються маршрутизатори, які вказують, які функції або класи відповідають на різні URL-шляхи.

wsgi.py: Цей файл використовується для розгортання проекту Django на WSGI-серверах (Web Server Gateway Interface). Він забезпечує взаємодію між веб-сервером та додатком Django.

Коли команда django-admin startproject завершує свою роботу, створюється початкова структура проекту Django. Ви можете редагувати файли setITngs.py та urls.py, щоб налаштувати ваш проект, додати нові URL-шляхи, підключити додатки, встановити базу даних та зробити інші необхідні зміни.

Після створення проекту ви можете виконувати команди python manage.py, щоб запустити розробку сервера, створити та застосувати міграції, створити адміністратора та виконати багато інших дій, пов'язаних з управлінням вашим проектом Django.

3.3.1 Структура інтерфейсу веб-застосунку

Для створення свого прикладу веб інтерфейсу я вирішив використати ще один професійний інструмент Figma.

Figma – це популярний інструмент для дизайну та прототипування веб-інтерфейсів. Він надає можливість створювати професійні дизайни, співпрацювати з командою та створювати інтерактивні прототипи веб-застосунків. Ось кілька кроків, які можуть допомогти вам розпочати роботу з Figma:

Створення проекту: Після входу в Figma ви можете створити новий проект. Виберіть опцію "New File" або "Create New" і виберіть тип дизайну, наприклад, "Web Design".

Розмір холста: Встановіть розміри холста, які відповідають типу веб-застосунку, над яким ви працюєте. Зазвичай вибирають стандартні розміри, такі як 1280 пікселів шириною для настільних комп'ютерів або 375 пікселів шириною для мобільних пристроїв.

Розташування елементів: Використовуйте інструменти Figma, щоб створити елементи вашого веб-інтерфейсу, такі як кнопки, поля введення, меню та інші. Ви можете малювати форми, вставляти текст, використовувати графічні елементи та багато іншого.

Використання макетів: Figma дозволяє створювати та використовувати макети (components) для елементів, які повторюються на вашому веб-сайті. Наприклад, ви можете створити макет для заголовків або навігаційного меню, а потім використовувати його в усьому проекті. Якщо ви змінюєте макет, він автоматично оновлюється у всіх місцях, де він був використаний.

Прототипування: Figma дозволяє створювати інтерактивні прототипи вашого веб-застосунку. Ви можете додавати посилання між сторінками, налаштовувати анімацію, взаємодіювати з формами та іншими елементами. Це допоможе вам відтворити досвід взаємодії з вашим веб-застосунком і отримати зворотний зв'язок від команди або користувачів.

Співпраця та коментарі: Figma надає можливість співпрацювати з командою над проектом. Ви можете запрошувати інших користувачів, ділитися посиланнями на дизайн, залишати коментарі та відповідати на них. Це дозволяє ефективно працювати разом і покращує комунікацію всередині команди.

Експорт: Коли ваш дизайн готовий, ви можете експортувати його у різних форматах, таких як PNG, JPEG, SVG або згенерувати CSS-код для елементів. Це допоможе передати дизайн розробникам або іншим зацікавленим сторонам.

Figma надає безліч інших функцій, які полегшують роботу над дизайном веб-застосунків. Важливо вивчити різні інструменти та можливості, які він надає, і використовувати їх на користь вашого проекту.

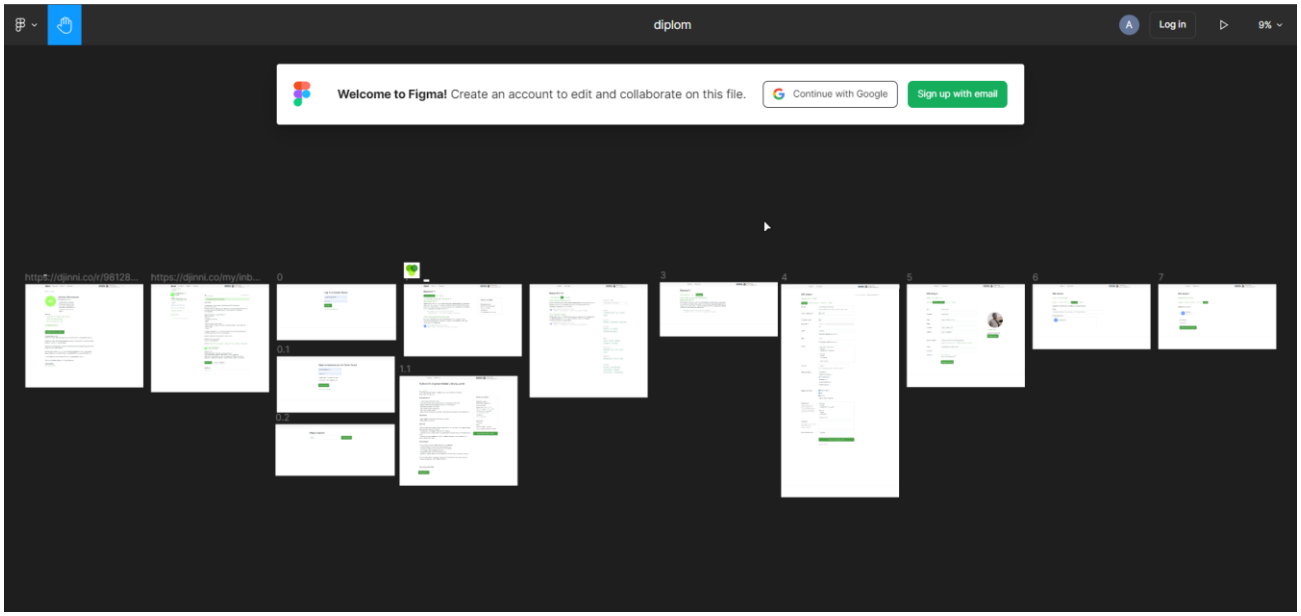


Рисунок 3.7 – Весь проєкт Figma

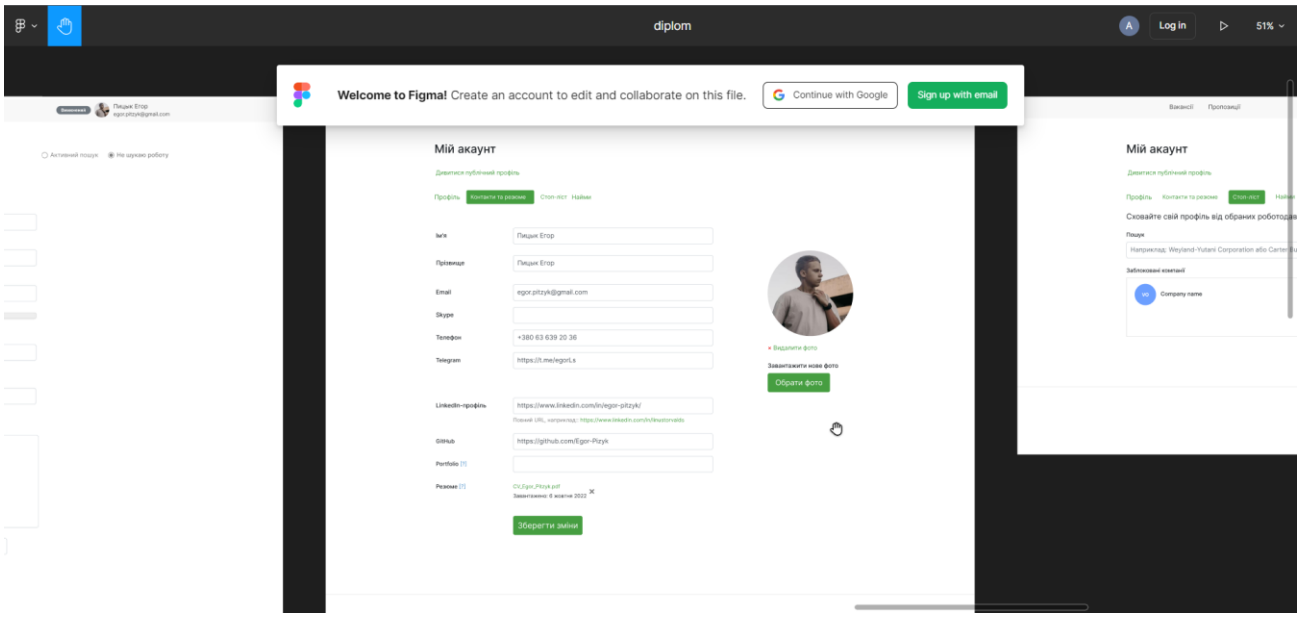


Рисунок 3.8 – Один з екранів проекту Figma

3.3.2 Сторонні модулі, використанні у веб-застосунку, їх опис та призначення

Для простішої та швидшої розробки веб-застосунку необхідно встановити деякі додаткові модулі, які допоможуть створити необхідний функціонал. Слід зазначити, що всі модулі пристосовані до роботи в браузерах на базі Chromium, тому гарантувати коректну роботу у всіх браузерах не можу.

3.3.2 Компоненти веб-застосунку

Під час розробки веб-застосунку я розробив деякі додаткові компоненти та допоміжні файли, щоб розбити застосунок по функціоналу. Нижче представлено таблицю, з описом цих компонентів.

Бібліотека requests:

Опис: requests - це популярна бібліотека Python для виконання HTTP-запитів. Вона надає простий та елегантний інтерфейс для взаємодії з веб-серверами, отримання даних, відправки параметрів запиту, установки заголовків тощо.

Використання: За допомогою requests ви можете здійснювати GET-, POST-, PUT-, DELETE- та інші види запитів до серверів, обробляти отримані відповіді та працювати з кукісами, авторизацією та сеансами.

Бібліотека io:

Опис: io - це модуль Python, який надає базові класи та функції для роботи з потоками вводу-виводу. Він дозволяє зчитувати та записувати дані з різних джерел та у різні цільові об'єкти.

Використання: io використовується для роботи зі стрічками, байтами, файлами, мережевими сокетами та іншими об'єктами. Ви можете створювати об'єкти io для зчитування, запису та обробки даних, а також для перенаправлення виводу та вводу.

Бібліотека json:

Опис: json - це модуль Python для роботи з форматом обміну даними JSON (JavaScript Object NotaITon). Він надає функції для кодування (серіалізації) та декодування (десеріалізації) об'єктів Python в формат JSON та навпаки.

Використання: json використовується для обміну даними між різними програмами або системами, які підтримують формат JSON. Ви можете перетворювати об'єкти Python у рядки JSON для відправки або збереження даних, а також декодувати рядки JSON у відповідні об'єкти Python для подальшого використання.

Бібліотека jinja2:

Опис: jinja2 - це популярний движок шаблонів для Python, який надає потужні можливості для генерації тексту на основі шаблонів. Він використовує синтаксис, схожий на HTML або XML, з можливістю використання змінних, циклів, умовних виразів тощо.

Використання: За допомогою jinja2 ви можете створювати шаблони для генерації HTML-сторінок, текстових файлів, електронних листів тощо. Ви можете

вставляти дані з об'єктів Python, створювати умовні вирази та цикли для динамічного форматування виводу.

Ці бібліотеки широко використовуються у спільноті Django та допомагають розширити можливості фреймворка, забезпечуючи зручний та потужний інструментарій для роботи з мережевими запитами, даними, шаблонами та іншими аспектами розробки веб-застосунків.

3.3.3 Опис структури веб-застосунку

Розроблений веб-застосунок складається з файлів, що розташовані в певній ієрархії та працюють між собою в користувальницькому режимі. Структура проекту Django, створеного за допомогою django-admin, має наступну об'ємну структуру:

Коренева папка проекту:

Це головна папка проекту, що містить всі інші компоненти. При створенні проекту Django за допомогою django-admin, ця папка отримує назву, вказану при створенні проекту.

Файл manage.py:

Це файл, що використовується для управління проектом. Він містить команди для запуску розробленого сервера, виконання міграцій бази даних, створення адміністративного користувача та інші команди, пов'язані з управлінням проектом.

Файл setITngs.py:

Цей файл містить налаштування проекту Django. Він включає інформацію про базу даних, шляхи до шаблонів, статичних файлів, мову, часовий пояс та інші глобальні налаштування. Ви можете настроїти цей файл для відповідності вашим потребам.

Папка urls.py:

Ця папка містить файли URL-маршрутизації для вашого проекту. Вона визначає, які URL-адреси співпадають з якими в'юхами. Тут ви можете налаштувати шляхи до різних сторінок вашого веб-застосунку.

Папка staITc:

Ця папка містить статичні файли, такі як CSS-стилі, JavaScript-скрипти, зображення та інші ресурси, які використовуються в вашому проекті. Ви можете створити підпапки для організації статичних файлів за потреби.

Папка templates:

У цій папці зберігаються шаблони HTML-файлів для вашого проекту. Вона може містити підпапки для організації шаблонів за різними категоріями або сторінками вашого веб-застосунку.

Папка apps:

Ця папка містить окремі додатки (програмні модулі) вашого проекту. Кожен застосунок може містити власну структуру, включаючи моделі, в'юхи, шаблони та інші файлі. Використання додатків дозволяє організувати ваш проект на модульний підхід.

Файл models.py:

Цей файл містить моделі бази даних вашого проекту. Ви можете визначити таблиці, поля, зв'язки та правила поведінки об'єктів бази даних у цьому файлі.

Файл views.py:

Цей файл містить в'юхи вашого проекту. В'юхи визначають, як обробляти запити веб-сторінок та як відображати дані. Вони містять логіку вашого додатку.

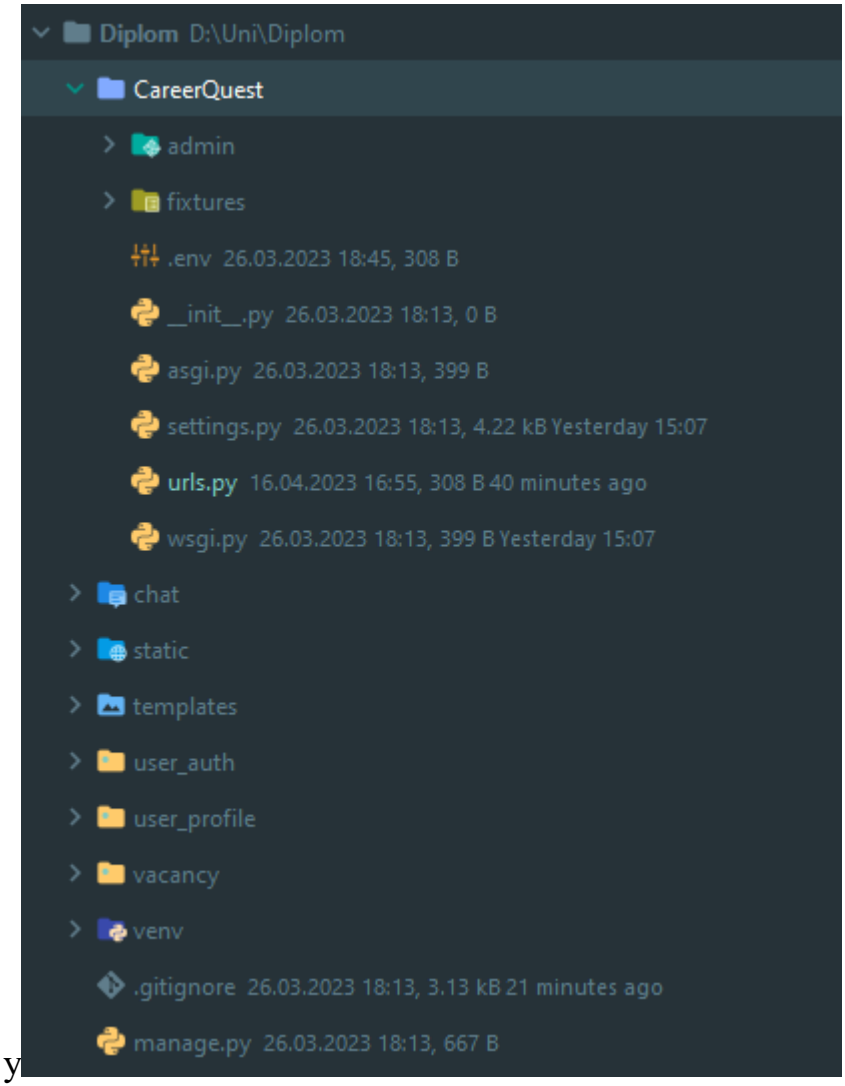
Файл admin.py:

Цей файл містить налаштування адміністративного інтерфейсу Django. Ви можете зареєструвати ваші моделі, щоб вони були доступні для редагування через адміністративний інтерфейс.

Файл forms.py:

У цьому файлі ви можете визначати форми, які використовуються в вашому проекті. Форми дозволяють збирати дані від користувачів та обробляти їх.

Це загальна структура проекту Django, створеного за допомогою django-admin. Ви можете створювати додаткові файли та папки відповідно до ваших потреб та організувати проект залежно від його складності та розміру.



у

Рисунок 3.9 − Структура файлів проекту

Висновки до розділу

Я був відповідальний за розробку веб-сервісу для моніторингу вакансій у сфері IT. Цей проект вимагав використання передових бібліотек, модулів та пакетів, які спроможні ефективно опрацьовувати та візуалізувати дані. Моя

команда та я ретельно досліджували найкращі практики та рішення, які стосуються розробки веб-застосунків, для того, щоб забезпечити високу якість та надійність сервісу.

У процесі реалізації веб-сервісу, ми використовували потужні бібліотеки для збору даних про вакансії з різних джерел. Модуль requests дозволяв нам здійснювати HTTP-запити до веб-сайтів з вакансіями, отримувати відповіді та обробляти їх для подальшого аналізу.

Для зручного опрацювання та обробки даних, ми використовували модуль json, який дозволяє нам працювати з форматом JSON. Ми збирали отримані дані та перетворювали їх у зручну для подальшої обробки структуру.

Для збереження та маніпулювання файлами, ми використовували модуль io. Цей модуль надавав нам зручні інструменти для роботи з різними типами файлів, такими як текстові файли, зображення, архіви та інші.

Наша команда також використовувала потужний шаблонний движок jinja2 для побудови динамічних HTML-сторінок. Цей двигун надавав нам можливість легко вбудовувати змінні, цикли та умови в шаблони, що дозволяло створювати динамічний контент для користувачів.

Весь процес розробки вимагав нашої уваги до найкращих практик у сфері побудови веб-застосунків, таких як застосування шаблонів проектування, оптимізація продуктивності та забезпечення безпеки даних. Ми прагнули забезпечити високу якість та надійність нашого веб-сервісу, а також надати зручний та ефективний інтерфейс для користувачів.

ВИСНОВОК

У результаті виконання бакалаврської кваліфікаційної роботи було спроектовано та розроблено веб-застосунок для пошуку вакансій IT спеціальностей та виконано такі завдання:

- Досліджено сучасні підходи до розроблення і впровадження вебсервісів;
- Проаналізовано архітектурні рішення та обрано програмні засоби для реалізації веб-системи;
- Програмно реалізовано веб-сервіс для пошуку вакансій IT спеціальностей.

Зокрема, було досліджено аналоги побудованої системи, підібрано найкращі практики та формат веб-застосунку, розглянуто різні технології, які на сьогодні використовуються для побудови сучасних веб-застосунків, оглянуто джерела статистичних даних про пошук вакансій IT спеціальностей.

Велика частина часу була виділена в першу чергу на проектування системи, бази даних, окремих компонентів та їх взаємодії, куди також входила розробка дизайну веб-застосунку. Досліджено сучасні підходи до розробки користувацького інтерфейсу, серверної частини веб-застосунків. Зроблено огляд та вибір бібліотек, модулів, пакетів, фреймворків, необхідних для створення системи для моніторингу за предметами та явищами, де необхідне використання та візуалізація статистичних даних.

Для проектування та розробки системи кваліфікаційної бакалаврської роботи був використаний стек технологій MERN на мові програмування Python, куди входять такі технології, як PostgreSQL, Jinja2. Для серверної частини веб-застосунку було використано Django.

База даних спроектована та розроблена за допомогою засобів PostgreSQL та бібліотеки postgrsqlpy. Для створення користувацького інтерфейсу використовувалася HTML та CSS (SCSS), до допоміжних засобів відносяться JavaScript. Код фронтенд-частини веб-застосунку було розгорнуто в мережі

інтернет за допомогою засобів Netlify. Розробка виконувалася в IDE Pycharm від JetBrains, яке було безкоштовно надане мені за студентською ліцензією.

В результаті було створено веб-застосунок для пошуку вакансій IT спеціальностей.

Головними особливостями веб-застосунку є інтерактивна карта, щогодинне оновлення, інфографіка, а також доступ у мережі інтернет, завдяки чому сайт доступний на всіх платформах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Встановлення і запуск python/django на cpanel • блог компанії hostpro. Блог компанії Hostpro. URL: https://hostpro.ua/blog/ua/installation-and-launch-python-django (дата звернення: 21.03.2023).
2. Как использовать Django, PostgreSQL и Docker: Статья из блога IT-школы Hillel. Корисні матеріали: Статті та новини IT-індустрії | Комп'ютерна школа Hillel. URL: https://blog.ithillel.ua/ru/articles/how-to-use-django-postgresql-and-docker (дата звернення: 14.03.2023).
3. Поради по роботі з БД у Django. Codeguida. URL: https://codeguida.com/post/1640 (дата звернення: 25.06.2023).
4. Что такое рефакторинг кода и зачем его проводить: статья из блога IT-школы Hillel. Корисні матеріали: Статті та новини IT-індустрії | Комп'ютерна школа Hillel. URL: https://blog.ithillel.ua/ru/articles/zachem-i-kak-provodit-refaktoring-koda (дата звернення: 18.03.2023).
5. API reference | mapbox GL JS. Mapbox. URL: https://docs.mapbox.com/mapbox-gl-js/api/ (дата звернення: 25.06.2023).
6. Comparisons of API architectural styles. Comparisons of API Architectural Styles | API Guide. URL: https://www.moesif.com/blog/api-guide/comparisons-of-api-architectural-styles/ (дата звернення: 15.03.2023).
7. Deploying a django application on ubuntu. Honeybadger Developer Blog. URL: https://www.honeybadger.io/blog/deploy-django-ubuntu/ (date of access: 20.04.2023).
8. Design patterns - MVC pattern. Online Courses and eBooks Library. URL: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm (дата звернення: 25.05.2023).

9. Django відображення - час творити! · HonKit. Choose a language · HonKit. URL: https://tutorial.djangogirls.org/uk/django_views/ (дата звернення: 25.05.2023).
10. Django - add static files. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/django/django_add_static_files.php (дата звернення: 25.02.2023).
11. Django. Django Project. URL: https://docs.djangoproject.com/en/4.2/howto/static-files/ (дата звернення: 15.03.2023).
12. Ellingwood J. How to set up django with postgres, nginx, and gunicorn on ubuntu 18.04. DigitalOcean | The Cloud for Builders. URL: https://www.digitalocean.com/community/tutorials/how-to-set-up-django-with-postgres-nginx-and-gunicorn-on-ubuntu-18-04 (дата звернення: 25.06.2023).
13. How to lookup dictionary value with key in django template - fedingo. Fedingo. URL: https://fedingo.com/how-to-lookup-dictionary-value-with-key-in-django-template/ (дата звернення: 11.03.2023).
14. HTML Підручник. Початок. HTML CSS JavaScript PHP SQL. W3Schools українською. URL: https://w3schoolsua.github.io/html/index.html#gsc.tab=0 (дата звернення: 04.04.2023).
15. Podila P. HTTP: Протокол, який повинен розуміти кожний веб-розробник (Частина 1). Code Envato Tuts+. URL: https://code.tutsplus.com/uk/tutorials/http-theprotocol-every-web-developer-must-know-part-1--net-31177 (дата звернення: 25.02.2023).
16. PostgreSQL - INDEXES. Online Courses and eBooks Library. URL: https://www.tutorialspoint.com/postgresql/postgresql_indexes.htm (дата звернення: 16.02.2023).

17. REST architectural constraints. REST API Tutorial. URL: https://restfulapi.net/rest-architectural-constraints/ (дата звернення: 25.06.2023).
18. Richardson maturity model. martinfowler.com. URL: https://martinfowler.com/articles/richardsonMaturityModel.html (дата звернення: 03.02.2023).
19. Sass: sass basics. Sass: Syntactically Awesome Style Sheets. URL: https://sass-lang.com/guide (дата звернення: 25.06.2023).
20. What is the difference between SCSS and SASS ? - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/what-is-the-difference-between-scss-and-sass/ (дата звернення: 08.05.2023).
21. Abba I. V. What is an ORM – the meaning of object relational mapping database tools. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/ (дата звернення: 25.06.2023).
22. Awati R. What is object-relational mapping (ORM)? – TechTarget Definition. TheServerSide.com. URL: https://www.theserverside.com/definition/object-relational-mapping-ORM (дата звернення: 05.02.2023).
23. Django ORM under the hood - iterables. HackSoft - Your development partner beyond code. URL: https://www.hacksoft.io/blog/django-orm-under-the-hood-iterables (дата звернення: 15.03.2023).
24. Frond-end options for django. Django Forum. URL: https://forum.djangoproject.com/t/frond-end-options-for-django/14934 (дата звернення: 22.05.2023).
25. Kellton tech solutions limited. Technology Consulting | IoT & Digital Solutions | IT Services. URL: https://www.kellton.com/kellton-tech-blog/why-django-web-development-with-python-for-backend-web-development (дата звернення: 25.06.2023).

26. Object relational tutorial (1.x API) – sqlalchemy 1.4 documentation. SQLAlchemy Documentation – SQLAlchemy 2.0 Documentation. URL: https://docs.sqlalchemy.org/en/14/orm/tutorial.html (дата звернення: 01.01.2023).
27. Patarkatsishvili G. Django ORM relationships cheat sheet | hackernoon. HackerNoon - read, write and learn about any technology. URL: https://hackernoon.com/django-orm-relationships-cheat-sheet-14433d6cf68c (дата звернення: 25.02.2023).
28. rabindraRegmi. Which frontend do you prefer with Django?. Reddit. URL: https://www.reddit.com/r/django/comments/ancuh7/which_frontend_do_you_prefer_with_django/ (дата звернення: 25.06.2023).
29. Real Python. Build a django front end with bulma – part 2 – real python. Python Tutorials – Real Python. URL: https://realpython.com/django-social-front-end-2/ (дата звернення: 04.03.2023).
30. Top 10 python web development frameworks in 2023 | browserstack. BrowserStack. URL: https://www.browserstack.com/guide/top-python-web-development-frameworks (дата звернення: 21.03.2023).

ДОДАТКИ

ДОДАТОК А

Код сервера, файл user_profile/models.py

```
        class User(AbstractUser, PermissionsMixin):
    username = None
    email = models.EmailField(_('email address'), unique=True)
    role = models.CharField(max_length=3, choices=ROLE)

    objects = UserProfileManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    def get_full_name(self):
        return f'{self.first_name} {self.last_name}'

    def get_avatar_img(self):
        if self.role == 'CAN' and
Candidate.objects.filter(user_id=self.pk).values('avatar_img'):
            return
Candidate.objects.filter(user_id=self.pk).values('avatar_img')[0].get('avatar_img').replac
e('staITc/', '')
        elif self.role == 'EMP' and
Employer.objects.filter(user_id=self.pk).values('avatar_img'):
            return
Employer.objects.filter(user_id=self.pk).values('avatar_img')[0].get('avatar_img').replac
e('staITc/', '')
        return 'media/defualt_user_icon.svg'
```

ДОДАТОК Б

Код сервера, файл vacancy/models.py

```
class Vacancy(models.Model):
    employer = models.ForeignKey(Employer, on_delete=models.CASCADE)
    ITtle = models.CharField(max_length=255)
    text = models.TextField()
    requirements = models.TextField()
    country = models.ForeignKey(Country, on_delete=models.CASCADE)
    employment_rate = models.CharField(max_length=3,
```

```
choices=EMPLOYMENT_RATE)
    month_salary = models.PosiITveSmallIntegerField()
    hour_salary = models.PosiITveSmallIntegerField()
    experience = models.FloatField()
    english_level = models.CharField(max_length=2, choices=ENGLISH_LEVEL)
    work_category = models.ForeignKey(WorkCategory, on_delete=models.CASCADE)
    favorite = models.ManyToManyField(User, related_name='fav_vacancy')

    def __str__(self):
        return f'{self.employer.company_name} - {self.ITtle}'

    def get_absolute_url(self):
        return reverse('vacancy:vacancy-detail', kwargs={'pk': self.pk})
```